XMC4100 / XMC4200

Microcontroller Series
for Industrial Applications

XMC4000 Family

ARM® Cortex®-M4
32-bit processor core

Reference Manual

V1.6 2016-07

Microcontrollers

Edition 2016-07

Published by
Infineon Technologies AG
81726 Munich, Germany

Legal Disclaimer

The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics. With respect to any examples or hints given herein, any typical values stated herein and/or any information regarding the application of the device, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation, warranties of non-infringement of intellectual property rights of any third party.

Information

For further information on technology, delivery terms and conditions and prices, please contact the nearest Infineon Technologies Office (**www.infineon.com**).

Warnings

Due to technical requirements, components may contain dangerous substances. For information on the types in question, please contact the nearest Infineon Technologies Office.

Infineon Technologies components may be used in life-support devices or systems only with the express written approval of Infineon Technologies, if a failure of such components can reasonably be expected to cause the failure of that life-support device or system or to affect the safety or effectiveness of that device or system. Life support devices or systems are intended to be implanted in the human body or to support and/or maintain and sustain and/or protect human life. If they fail, it is reasonable to assume that the health of the user or other persons may be endangered.

XMC4100 / XMC4200

Microcontroller Series
for Industrial Applications

XMC4000 Family

ARM® Cortex®-M4
32-bit processor core

Reference Manual

V1.6 2016-07

Microcontrollers

XMC4[12]00 Reference Manual

Revision History: V1.6 2016-07

Previous Versions:
V1.5, 2014-04

Page	Subjects
2-83, **2-108**ff.	Updated CPU chapter • Corrected register AFSR and MVFRx to match register definition used by compilers and other development tools.
multiple, **4-21**ff., **4-32**ff.	Updated Service Request Processing chapter • Corrected SCU register name RAWSR to SRRAW. • “Service Request Generation” section revised. • ERU0 outputs not connected directly to NVIC but routed thru SCU. Corrected in ERU0 tables.
1-177, **1-181**	Updated SCU chapter • Removed bit ECAT0SEL from SYSCLKCR register • Corrected bit field description of register EXTCLKCR
1-36	Updated LEDTS chapter • Removed connection to P5.10, port not available in XMC4400 devices
20-5 and ff.	Updated HRPWM chapter • Corrected “inverting comparator input”, the input is non-inverting.
23-1	Adjusted text style only to match Infineon standard.

Previous Versions:
V1.4, 2013-11

Page	Subjects
2-102, **2-108**f.	Updated CPU chapter • Corrected FPCCR reset value • Added MVFR0 and MVFR1 register descriptions
5-23	Corrected figure “Channel FIFO contents” in GPDMA chapter.
11-1ff.	SCU chapter enhanced for readability.
12-13, **12-23**, **12-26**, **12-31**	Updated LEDTS chapter • Added typical fequency range in section on Finger Sensing • Module ID.MOD_REV starts with 00_H • Clarified limitation on bit field FNCTL.NR_LEDCOL for value 111_B • Corrected TSVAL.TSCTRVALR bit field type to “rh”

XMC4[12]00 Reference Manual

Revision History: V1.6 2016-07

14-163, **14-226**ff.	Updated USIC chapter • CCFG register reset value corrected • Added hints on the signal usage to Interconnect tables
17-13	Extended 1st step of initialisation sequence.
20-130	Corrected the maximum values for the HRCyCR1, HRCyCr2, HRCySCR1 and HRCySCR2 registers for operation @ 80 MHz.
24-11	Added Infineon JTAG ID code information to table in DEBUG chapter.

Previous Versions:
V1.3, 2013-09

Page	Subjects
20-127	HRPWM module initialization sequence was updated.
23-4	Startup Software figure was updated to contain HRPWM initialization data.

Previous Versions:
V1.2, 2013-08

Page	Subjects
11-9, **11-67**ff.	Updated SCU chapter • Added Die temperature traps to table • Completed registers overview table • Added missing table rows in registers section

Previous Versions:
V1.1, 2012-11

Page	Subjects
	Replaced occurences of f_{PB} by f_{PERIPH} in several locations.
11-187, **11-10**ff.	SCU chapter • Width of OSCHPCTRL.OSVAL bit field changed from 5 to 4 bits. • Described DTS calibration mechanism.
12-36ff.	Added PORT connections to Interconnects section in LEDTS chapter.

XMC4[12]00 Reference Manual

Revision History: V1.6 2016-07

14-1ff.	USIC chapter • Abbreviations table is added • When in slave mode and SCLKCFG=01_B, bit DX1CR.DPOL needs to be set to 1, see **Section 14.4.1.2** • Description on SCLKCFG setting is removed from **Section 14.6.3** • Assignment of USICx_SRy to GPDMA is updated. • Coding for WLENx bit field in RBUF01SR register is updated to consider dual and quad SSC modes.
17-1ff.	DAC chapter • Behavior after setting ANAEN bit ($t_{STARTUP}$ time) clarified
22-30ff., **22-35**ff.	PORTS chapter • Added Ethernet MII input connections shared with RMII connections. • Added note on driver mode on hardwired pins (HIB_IOs and TMS).
24-11	Revised the ROM table section in DEBUG chapter.

Trademarks

C166™, TriCore™, XMC™ and DAVE™ are trademarks of Infineon Technologies AG.

ARM®, ARM Powered® and AMBA® are registered trademarks of ARM, Limited.

Cortex™, CoreSight™, ETM™, Embedded Trace Macrocell™, Embedded Trace Buffer™ and Thumb® are trademarks of ARM, Limited.

Synopsys™ is a trademark of Synopsys, Inc.

We Listen to Your Comments
Is there any information in this document that you feel is wrong, unclear or missing? Your feedback will help us to continuously improve the quality of this document. Please send your proposal (including a reference to this document) to:
mcdocu.comments@infineon.com

Table of Contents

About this Document

This Reference Manual is addressed to embedded hardware and software developers. It provides the reader with detailed descriptions about the behavior of the XMC4[12]00 series functional units and their interaction.

The manual describes the functionality of the superset device of the XMC4[12]00 microcontroller series. For the available functionality (features) of a specific XMC4[12]00 derivative (derivative device), please refer to the respective Data Sheet. For simplicity, the various device types are referenced by the collective term XMC4[12]00 throughout this manual.

XMC4000 Family User Documentation

The set of user documentation includes:

- **Reference Manual**
  – decribes the functionality of the superset device.
- **Data Sheets**
  – list the complete ordering information, available features and electrical characteristics of derivative devices.
- **Errata Sheets**
  – list deviations from the specifications given in the related Reference Manual or Data Sheets. Errata Sheets are provided for the superset of devices.

Attention: Please consult all parts of the documentation set to attain consolidated knowledge about your device.

Application related guidance is provided by **Users Guides** and **Application Notes**.

Please refer to **http://www.infineon.com/xmc4000** to get access to the latest versions of those documents.

Related Documentation

The following documents are referenced:

- ARM® Cortex®-M4
  – Technical Reference Manual
  – User Guide, Reference Material
- ARM®v7-M Architecture Reference Manual
- AMBA® 3 AHB-Lite Protocol Specification

Copyright Notice

- Portions of ETH, USB and GPDMA chapter Copyright © 2009, 2010 by Synopsys, Inc. All rights reserved. Used with permission.

Text Conventions

This document uses the following naming conventions:

- Functional units of the device are given in plain UPPER CASE. For example: “The USIC0 unit supports…”.
- Pins using negative logic are indicated by an overline. For example: “The $\overline{\text{WAIT}}$ input has…”.
- Bit fields and bits in registers are generally referenced as “Module_RegisterName.BitField” or “Module_RegisterName.Bit”. For example: “The USIC0_PCR.MCLK bit enables the…”. Most of the register names contain a module name prefix, separated by an underscore character “_” from the actual register name (for example, “USIC0_PCR”, where “USIC0” is the module name prefix, and “PCR” is the kernel register name). In chapters describing the kernels of the peripheral modules, the registers are mainly referenced with their kernel register names. The peripheral module implementation sections mainly refer to the actual register names with module prefixes.
- Variables used to describe sets of processing units or registers appear in mixed upper and lower cases. For example, register name “MOFCRn” refers to multiple “MOFCR” registers with variable n. The bounds of the variables are always given where the register expression is first used (for example, “n = 0-31”), and are repeated as needed in the rest of the text.
- The default radix is decimal. Hexadecimal constants are suffixed with a subscript letter “H”, as in 100_H. Binary constants are suffixed with a subscript letter “B”, as in: 111_B.
- When the extent of register fields, groups register bits, or groups of pins are collectively named in the body of the document, they are represented as “NAME[A:B]”, which defines a range for the named group from B to A. Individual bits, signals, or pins are given as “NAME[C]” where the range of the variable C is given in the text. For example: CFG[2:0] and SRPN[0].
- Units are abbreviated as follows:
  - **MHz** = Megahertz
  - **µs** = Microseconds
  - **kBaud, kbit/s** = 1000 characters/bits per second
  - **MBaud, Mbit/s, Mbps** = 1,000,000 characters/bits per second
  - **Kbyte, KB** = 1024 bytes of memory
  - **Mbyte, MB** = 1048576 bytes of memory
    In general, the k prefix scales a unit by 1000 whereas the K prefix scales a unit by 1024. Hence, the Kbyte unit scales the expression preceding it by 1024. The kBaud unit scales the expression preceding it by 1000. The M prefix scales by 1,000,000 or 1048576. For example, 1 Kbyte is 1024 bytes, 1 Mbyte is

1024 × 1024 bytes, 1 kBaud/kbit are 1000 characters/bits per second, 1 MBaud/Mbit are 1000000 characters/bits per second, and 1 MHz is 1,000,000 Hz.

- Data format quantities are defined as follows:
  - **Byte** = 8-bit quantity
  - **Half-word** = 16-bit quantity
  - **Word** = 32-bit quantity
  - **Double-word** = 64-bit quantity

Bit Function Terminology

In tables where register bits or bit fields are defined, the following conventions are used to indicate the access types.

Table 1 Bit Function Terminology

Bit Function	Description
rw	The bit or bit field can be read and written.
rwh	As rw, but bit or bit field can be also set or reset by hardware. If not otherwise documented the software takes priority in case of a write conflict between software and hardware.
r	The bit or bit field can only be read (read-only).
w	The bit or bit field can only be written (write-only). A read to this register will always give a default value back.
rh	This bit or bit field can be modified by hardware (read-hardware, typical example: status flags). A read of this bit or bit field give the actual status of this bit or bit field back. Writing to this bit or bit field has no effect to the setting of this bit or bit field.

Register Access Modes

Read and write access to registers and memory locations are sometimes restricted. In memory and register access tables, the following terms are used.

Table 2 Register Access Modes

Symbol	Description
U	Access permitted when software executes on Unprivileged level.
PV	Access permitted when software executes on Privileged level.
32	Only 32-bit word accesses are permitted to this register/address range.
NC	No change, indicated register is not changed.

Table 2 Register Access Modes (cont'd)

Symbol	Description
BE	Indicates that an access to this address range generates a Bus Error.
nBE	Indicates that no Bus Error is generated when accessing this address range.

Reserved Bits

Register bit fields named **Reserved** or **0** indicate unimplemented functions with the following behavior:

- Reading these bit fields returns 0.
- These bit fields should be written with 0 if the bit field is defined as r or rh.
- These bit fields must to be written with 0 if the bit field is defined as rw.

Abbreviations and Acronyms

The following acronyms and terms are used in this document:

ADC	Analog-to-Digital Converter
AHB	Advanced High-performance Bus
AMBA	Advanced Microcontroller Bus Architecture
ASC	Asynchronous Serial Channel
BMI	Boot Mode Index
BROM	Boot ROM
CAN	Controller Area Network
CMSIS	Cortex Microcontroller Software Interface Standard
CPU	Central Processing Unit
CRC	Cyclic Redundancy Code
CCU4	Capture Compare Unit 4
CCU8	Capture Compare Unit 8
DAC	Digital to Analog Converter
DSD	Delta Sigma Demodulator
DSRAM	Data SRAM
DMA	Direct Memory Access
EBU	External Bus Interface
ECC	Error Correction Code

ERU	Event Request Unit
ETH	Ethernet Unit
FCE	Flexible CRC Engine
FCS	Flash Command State Machine
FIM	Flash Interface and Control Module
FPU	Floating Point Unit
GPDMA	General Purpose Direct Memory Access
GPIO	General Purpose Input/Output
HMI	Human-Machine Interface
HRPWM	High Resolution PWM
IIC	Inter Integrated Circuit (also known as I^2C)
IIS	Inter-IC Sound Interface
I/O	Input / Output
JTAG	Joint Test Action Group = IEEE1149.1
LED	Light Emitting Diode
LEDTS	LED and Touch Sense (Control Unit)
LIN	Local Interconnect Network
MPU	Memory Protection Unit
MSB	Most Significant Bit
NC	Not Connected
NMI	Non-Maskable Interrupt
NVIC	Nested Vectored Interrupt Controller
OCDS	On-Chip Debug Support
OTP	One Time Programmable
PBA	Peripheral Bridge AHB to AHB
PFLASH	Program Flash Memory
PLL	Phase Locked Loop
PMU	Program Memory Unit
POSIF	Position Interface
PWM	Pulse Width Modulation
PSRAM	Program SRAM
RAM	Random Access Memory

RTC	Real Time Clock
SCU	System Control Unit
SDMMC	Secure Digital / Multi Media Card (Interface)
SDRAM	Synchronous Dynamic Random Access Memory
SFR	Special Function Register
SPI	Serial Peripheral Interface
SRAM	Static RAM
SR	Service Request
SSC	Synchronous Serial Channel
SSW	Startup Software
UART	Universal Asynchronous Receiver Transmitter
UCB	User Configuration Block
USB	Universal Serial Bus
USIC	Universal Serial Interface Channel
WDT	Watchdog Timer

Introduction

1 Introduction

The XMC4[12]00 series belongs to the XMC4000 family of industrial microcontrollers based on the ARM Cortex-M4 processor core. The XMC4000 series devices are optimized for electrical motor control, power conversion, industrial connectivity and sense & control applications.

The growing complexity of today's energy efficient embedded control applications are demanding microcontroller solutions with higher performance CPU cores featuring DSP (Digital Signal Processing) and FPU (Floating Point Unit) capabilities as well as integrated peripherals that are optimized for performance. Complemented with a development environment designed to shorten product development time and increase productivity, the XMC4[12]00 series of microcontrollers take advantage of Infineon's decades of experience in microcontroller design, providing an optimized solution to meet the performance challenges of today's embedded control applications.

1.1 Overview

The XMC4[12]00 series devices combine the extended functionality and performance of the ARM Cortex-M4 core with powerful on-chip peripheral subsystems and on-chip memory units. The following key features are available in the XMC4100 / XMC4200 series devices:

CPU Subsystem

- CPU Core
  - High Performance 32-bit ARM Cortex-M4 CPU
  - 16-bit and 32-bit Thumb2 instruction set
  - DSP/MAC instructions
  - System timer (SysTick) for Operating System support
- Floating Point Unit
- Memory Protection Unit
- Nested Vectored Interrupt Controller
- One General Purpose DMA with up to 8 channels
- Event Request Unit (ERU) for programmable processing of external and internal service requests
- Flexible CRC Engine (FCE) for multiple bit error detection

On-Chip Memories

- 16 KB on-chip boot ROM
- 16 KB on-chip high-speed program memory
- 24 KB on-chip high speed data memory
- 256 KB on-chip Flash Memory with 1 KB instruction cache

Communication Peripherals

- Universal Serial Bus, USB 2.0 host, with integrated PHY
- Controller Area Network interface (MultiCAN), Full-CAN/Basic-CAN with two nodes, 64 message objects, data rate up to 1 Mbit/s
- Four Universal Serial Interface Channels (USIC), usable as UART, double-SPI, quad-SPI, IIC, IIS and LIN interfaces
- LED and Touch-Sense Controller (LEDTS) for Human-Machine interface

Analog Frontend Peripherals

- Two Analog-Digital Converters (VADC) of 12-bit resolution, 8 channels each with input out-of-range comparators for over-voltage detection
- Digital-Analogue Converter (DAC) with two channels of 12-bit resolution

Industrial Control Peripherals

- One Capture/Compare Units 8 (CCU8) for motor control and power conversion
- Two Capture/Compare Units 4 (CCU4) for use as general purpose timers
- Four High Resolution PWM (HRPWM) channels
- One Position Interfaces (POSIF) for hall and quadrature encoders and motor positioning
- Window Watchdog Timer (WDT) for safety sensitive applications
- Die Temperature Sensor (DTS)
- Real Time Clock module with alarm support
- System Control Unit (SCU) for system configuration and control

Input/Output Lines

- Programmable port driver control module (PORTS)
- Individually bit addressable
- Tri-stated in input mode
- Push/pull or open drain output mode
- Boundary scan test support over JTAG interface

On-Chip Debug Support

- Full support for debug features: 8 breakpoints, CoreSight, trace
- Various interfaces: ARM-JTAG, SWD, single wire trace

Packages

- PG-LQFP-64
- PG-VQFN-48

Note: For details about package availability for a particular derivative please check the datasheet. For information on available delivery options for assembly support and general package see ***http://www.infineon.com/packages***

1.1.1 Block Diagram

The diagram below shows the functional blocks and their basic connectivity within the XMC4100 / XMC4200 System.

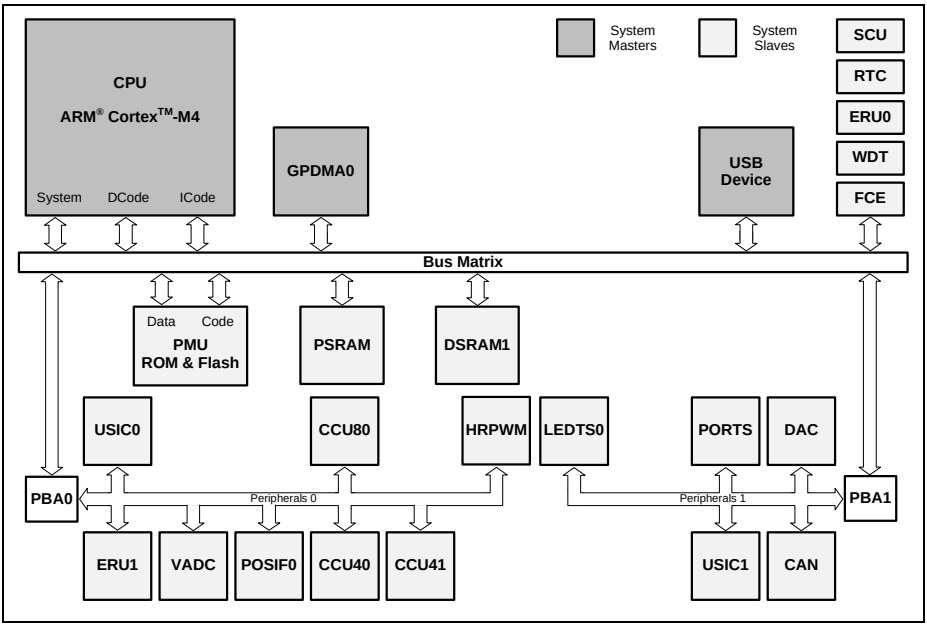


Figure 1-1 XMC4[12]00 System

1.2 CPU Subsystem

The XMC4[12]00 system core consists of the CPU (including FPU and MPU) and the memory interface blocks for program and data memories (including PMU).

Central Processing Unit (CPU)

The Cortex-M4 processor is built on a high-performance processor core with a 3-stage pipelined Harvard architecture, making it ideal for demanding embedded applications. The processor delivers exceptional power efficiency through an efficient instruction set and a design optimized for energy efficient control applications. To address the growing complexity of embedded control it also includes a IEEE754-compliant single-precision floating-point computation and a range of single-cycle/SIMD multiplication and multiply-and-accumulate capabilities, saturating arithmetic and dedicated hardware division.

To facilitate the design of cost-sensitive devices, the Cortex-M4 processor implements tightly-coupled system components that reduce processor area while significantly improving interrupt handling and system debug capabilities.

To ensure high code density and reduced program memory requirements the processor also implements a version of the Thumb® instruction set based on Thumb-2 technology. The instruction set provides the exceptional performance expected of a modern 32-bit architecture with the high code density of 8-bit and 16-bit microcontrollers.

Floating Point Unit (FPU)

The Floating-point unit (FPU) provides IEEE754-compliant operations on single precision, 32-bit, floating-point values.

Memory Protection Unit (MPU)

The MPU improves system reliability by defining the memory attributes for different memory regions. It provides fine grain memory control, enabling applications to utilize multiple privilege levels, separating and protecting code, data and stack on a task-by-task basis. Up to eight different regions are supported as well as an optional predefined background region. These features are becoming critical to support safety requirements in many embedded applications.

Programmable Multiple Priority Interrupt System (NVIC)

The XMC4[12]00 implements the ARM NVIC with 112 interrupt nodes and 64 priority levels. Most interrupt sources are connected to a dedicated interrupt node. In addition the XMC4[12]00 allows to route service request directly to dedicated units like DMA, Timer and ADC. In some cases, multi-source interrupt nodes are incorporated for efficient use of system resources. These nodes can be activated by several source requests and are controlled via interrupt sub node control registers.

Direct Memory Access (GPDMA)

The GPDMA is a highly configurable DMA controller that allows high-speed data transfers between peripherals and memories. Complex data transfers can be done with minimal intervention of the processor, keeping the CPU resources free for other operations. Provides multi block, scatter/gather and linked list transfers.

Flexible CRC Engine (FCE)

The FCE provides a parallel implementation of Cyclic Redundancy Code (CRC) algorithms. It implements the IEEE 802.3 CRC32, the CCITT CRC16 and the SAE J1850 CRC8 polynomials. The primary target of FCE is to be used as a hardware acceleration engine for software applications or operating systems services using CRC signatures.

1.3 On-Chip Memories

The on-chip memories provide zero-waitstate accesses to code and data. The memories can also be accessed concurrently from various system masters.

Various types of dedicated memories are available on-chip. The suggested use of the memories aims to improve performance and system stability in most typical application cases. However, the user has the flexibility to use the memories in any other way in order to fulfill application specific requirements.

Boot ROM (BROM)

The Boot ROM memory contains the boot code and the exception vector table. The basic system initialization sequence code, also referred to as firmware, is executed immediately after reset release.

Flash memory

The Flash is for nonvolatile code or constant data storage. The single supply Flash module is programmable at production line end and in application via built-in erase and program commands. Read and write protection mechanism are offered. A hardware error correction ensures data consistency over the whole life time under rugged industrial environment and temperatures.

The integrated cache provides an average performance boost factor of 3 in code execution compared to uncached execution.

Code RAM (PSRAM)

The Code RAM is intended for user code or Operating System data storage. The memory is accessed via the Bus Matrix and provides zero-wait-state access for the CPU for code execution or data access.

System RAM (DSRAM1)

The System RAM is intended for general user data storage. The System RAM is accessed via the Bus Matrix and provides zero-wait-state access for data.

1.4 Communication Peripherals

Communication features are key requirements in today's industrial systems. The XMC4[12]00 offers a set of peripherals supporting advanced communication protocols. Besides Ethernet, USB, CAN and the USIC the XMC4[12]00 provides interfaces to various memories as well as a unit to realize a human-machine interface via LED and Touch Sense.

LED and Touch Sense (LEDTS)

The LEDTS module drives LEDs and controls touch pads used in human-machine interface (HMI) applications. The LEDTS can measure the capacitance of up to 7 touch pads using the relaxation oscillator (RO) topology. The module can also drive up to 35 LEDs in an LED matrix. Touch pads and LEDs can share pins to minimize the number of pins needed for such applications.

Universal Serial Bus (USB)

The USB module supports device functions and complies to the USB 2.0 Specification with full-speed (12 Mbit/s) transfer capabilities.

USB usage is optimized for the following applications and systems:

- Portable electronic devices
- Point-to-point applications (direct connection to FS device)

Universal Serial Interface Channel (USIC)

The USIC is a flexible interface module covering several serial communication protocols such as ASC, LIN, SSC, I2C, I2S. A USIC module contains two independent communication channels. Two USIC modules are implemented, hence four channels can be used in parallel. A FIFO allows transmit and result buffering for relaxing real-time conditions. Multiple chip select signals are available for communication with multiple devices on the same channel.

Controller Area Network (CAN)

The MultiCAN module contains three independently operating CAN nodes with Full-CAN functionality that are able to exchange Data and Remote Frames via a gateway function. Transmission and reception of CAN frames is handled in accordance with CAN specification V2.0 B (active). Each CAN node can receive and transmit standard frames

with 11-bit identifiers as well as extended frames with 29-bit identifiers. Transmission rate is up to 1 Mbit/s

All CAN nodes share a common set of message objects. Each message object can be individually allocated to one of the CAN nodes. Besides serving as a storage container for incoming and outgoing frames, message objects can be combined to build gateways between the CAN nodes or to setup a FIFO buffer.

1.5 Analog Frontend Peripherals

The XMC4[12]00 hosts a number of interfaces to connect to the analog world.

Analog to Digital Converter (VADC)

The Versatile Analog-to-Digital Converter module consists of two independent kernel groups which operate according to the successive approximation principle (SAR). The resolution is programmable from 8 to 12 bits with a total conversion time of less than 500 ns @12 bit.

Each group provides a versatile state machine allowing complex measurement sequences. The groups can be synchronized and conversions may run completely in background. Multiple trigger events can be prioritized and allow the exact measurement of time-critical signals. The result buffering and handling avoids data loss and ensures consistency. Self-test mechanisms can be used for plausibility checks.

The basic structure supports a clean software architecture where tasks may only read valid results and do not need to care for starting conversions.

A number of out-of-range on-chip comparators serve the purpose of over-voltage monitoring for analog input pins of the VADC.

Digital to Analog Convertor (DAC)

The module consists of two separate 12-bit Digital-to-Analog Converters (DACs). It converts two digital input signals into two analog voltage signal outputs at a maximum conversion rate of 5 MHz.

A built-in wave generator mode allows stand alone generation of a selectable choice of wave forms. Alternatively values can be fed via CPU or DMA directly to one or both DAC channels. Additionally an offset can be added and the amplitude can be scaled. Several time trigger sources are possible.

1.6 Industrial Control Peripherals

Core components needed for motion and motor control, power conversion and other time based applications.

Capture/Compare Unit 4 (CCU4)

The CCU4 peripheral is a major component for systems that need general purpose timers for signal monitoring/conditioning and Pulse Width Modulation (PWM) signal generation. Power electronic control systems like switched mode power supplies or uninterruptible power supplies can easily be implemented with the functions inside the CCU4 peripheral.

The internal modularity of CCU4 translates into a software friendly system for fast code development and portability between applications.

Capture/Compare Unit 8 (CCU8)

The CCU8 peripheral functions play a major role in applications that need complex Pulse Width Modulation (PWM) signal generation, with complementary high side and low side switches, multi phase control or output parity checking. The CCU8 is optimized for state of the art motor control, multi phase and multi level power electronics systems.

The internal modularity of CCU8 translates into a software friendly system for fast code development and portability between applications.

High Resolution PWM (HRPWM)

The HRPWM extends the capabilities of the associated Capture/Compare Unit with a suite of features especially tailored for Switched Mode Power Supply (SMPS) applications.

These features not only enable an easy and fast control loop (voltage or current control) development for SMPS, that can run up to 4 MHz (with more than 10 bit resolution), but also helps the application developer to have an optimized solution in terms of total number of external components, avoid susceptibility to environmental and process variations and reducing the size of the power supply.

Position Interface Unit (POSIF)

The POSIF unit is a flexible and powerful component for motor control systems that use Rotary Encoders or Hall Sensors as feedback loop. The configuration schemes of the module target a very large number of motor control application requirements.

This enables the build of simple and complex control feedback loops for industrial and automotive motor applications, targeting high performance motion and position monitoring.

1.7 On-Chip Debug Support

The On-Chip Debug Support system based on the ARM CoreSight provides a broad range of debug and emulation features built into the XMC4[12]00. The user software can therefore be debugged within the target system environment.

Introduction

The On-Chip Debug Support is controlled by an external debugging device via the debug interface and an optional break interface. The debugger controls the On-Chip Debug Support via a set of dedicated registers accessible via the debug interface. Additionally, the On-Chip Debug Support system can be controlled by the CPU, e.g. by a monitor program.

CPU Subsystem

2 Central Processing Unit (CPU)

The XMC4[12]00 features the ARM Cortex-M4 processor. A high performance 32-bit processor designed for the microcontroller market. This CPU offers significant benefits to users, including:

- outstanding processing performance combined with fast interrupt handling
- enhanced system debug with extensive breakpoint and trace capabilities
- platform security robustness, with integrated memory protection unit (MPU).
- ultra-low power consumption with integrated sleep modes

References to ARM Documentation

The following documents can be found through **http://infocenter.arm.com**

[1] Cortex-M4 Devices, Generic User Guide (ARM DUI 0553A)

[2] Cortex Microcontroller Software Interface Standard (CMSIS)

References to ARM Figures

[3] **http://www.arm.com**

References to IEEE Documentation

[4] IEEE Standard IEEE Standard for Binary Floating-Point Arithmetic 754-2008.

2.1 Overview

The Cortex-M4 processor is built on a high-performance processor core, with a 3-stage pipeline Harvard architecture, making it ideal for demanding embedded applications. The processor delivers exceptional power efficiency through an efficient instruction set and extensively optimized design, providing high-end processing hardware including IEEE754-compliant single-precision floating-point computation, a range of single-cycle and SIMD multiplication and multiply-with-accumulate capabilities, saturating arithmetic and dedicated hardware division.

To facilitate the design of cost-sensitive devices, the Cortex-M4 processor implements tightly-coupled system components that reduce processor area while significantly improving interrupt handling and system debug capabilities. The Cortex-M4 processor implements a version of the Thumb® instruction set based on Thumb-2 technology, ensuring high code density and reduced program memory requirements. The Cortex-M4 instruction set provides the exceptional performance expected of a modern 32-bit architecture, with the high code density of 8-bit and 16-bit microcontrollers.

The Cortex-M4 processor closely integrates a configurable NVIC, to deliver industry-leading interrupt performance. The NVIC includes a non-maskable interrupt (NMI), and provides up to 64 interrupt priority levels. The tight integration of the processor core and

NVIC provides fast execution of interrupt service routines (ISRs), dramatically reducing the interrupt latency. This is achieved through the hardware stacking of registers, and the ability to suspend load-multiple and store-multiple operations. Interrupt handlers do not require wrapping in assembler code, removing any code overhead from the ISRs. A tail-chain optimization also significantly reduces the overhead when switching from one ISR to another.

To optimize low-power designs, the NVIC integrates with the sleep modes, that include a deep sleep function that enables the entire device to be rapidly powered down while still retaining program state.

2.1.1 Features

The XMC4[12]00 CPU features comprise

- Thumb2 instruction set combines high code density with 32-bit performance
- IEEE754-compliant single-precision FPU
- power control optimization of system components
- integrated sleep modes for low power consumption
- fast code execution permits slower processor clock or increases sleep mode time
- hardware division and fast digital-signal-processing orientated multiply accumulate
- saturating arithmetic for signal processing
- deterministic, high-performance interrupt handling for time-critical applications
- memory protection unit (MPU) for safety-critical applications
- extensive debug and trace capabilities:
  – Serial Wire Debug and Serial Wire Trace reduce the number of pins required for debugging, tracing, and code profiling.

2.1.2 Block Diagram

The Cortex-M4 core components comprise:

Processor Core

The CPU provides 16-bit and 32-bit Thumb2 instruction set and DSP/MAC instructions.

Floating-point unit

The FPU provides IEEE754-compliant operations on single-precision, 32-bit, floating-point values.

Nested Vectored Interrupt Controller

The NVIC is an embedded interrupt controller that supports low latency interrupt processing.

Memory Protection Unit

The MPU improves system reliability by defining the memory attributes for different memory regions. It provides up to eight different regions, and an optional predefined background region.

Debug Solution

The XMC4[12]00 implements a complete hardware debug solution.

- Embedded Trace Macrocell
- Traditional JTAG port or a 2-pin Serial Wire Debug Access Port
- Trace port or Serial Wire Viewer
- Flash breakpoints and Data watchpoints

This provides high system control and visibility of the processor and memory even in small package devices.

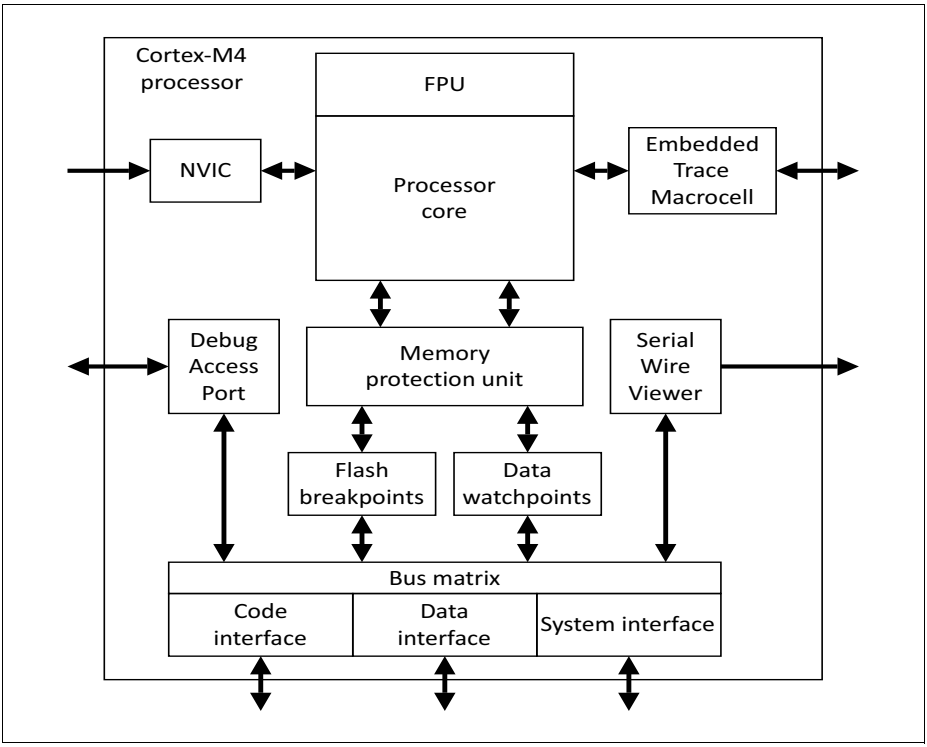


Figure 2-1 Cortex-M4 Block Diagram

System Level Interfaces

The Cortex-M4 processor provides a code, data and system interface using AMBA® technology to provide high speed, low latency accesses.

2.2 Programmers Model

This section describes the Cortex-M4 programmers model. In addition to the individual core register descriptions, it contains information about the processor modes and privilege levels for software execution and stacks.

2.2.1 Processor Mode and Privilege Levels for Software Execution

The processor modes are:

- **Thread mode**
  Used to execute application software. The processor enters Thread mode when it comes out of reset.
- **Handler mode**
  Used to handle exceptions. The processor returns to Thread mode when it has finished all exception processing.

The privilege levels for software execution are:

- **Unprivileged**
  Unprivileged software executes at the unprivileged level.
  The software:
  – has limited access to the MSR and MRS instructions, and cannot use the CPS instruction
  – cannot access the system timer, NVIC, or system control block
  – might have restricted access to memory or peripherals.
- **Privileged**
  Privileged software executes at the privileged level.
  The software can use all the instructions and has access to all resources.

In Thread mode, the CONTROL register controls whether software execution is privileged or unprivileged, see CONTROL register on **Page 2-15**. In Handler mode, software execution is always privileged.

Only privileged software can write to the CONTROL register to change the privilege level for software execution in Thread mode. Unprivileged software can use the SVC instruction to make a supervisor call to transfer control to privileged software.

2.2.2 Stacks

The processor uses a full descending stack. This means the stack pointer holds the address of the last stacked item in memory. When the processor pushes a new item onto the stack, it decrements the stack pointer and then writes the item to the new memory

location. The processor implements two stacks, the main stack and the process stack, with a pointer for each held in independent registers, see Stack Pointer on **Page 2-8**.

In Thread mode, the CONTROL register controls whether the processor uses the main stack or the process stack, see CONTROL register on **Page 2-15**. In Handler mode, the processor always uses the main stack. The options for processor operations are:

Table 2-1 Summary of processor mode, execution privilege level, and stack use options

Processor mode	Used to execute	Privilege level for software execution	Stack used
Thread	Applications	Privileged or unprivileged[1)]	Main stack or process stack[1)]
Handler	Exception handlers	Always privileged	Main stack

1) See CONTROL register on **Page 2-15**.

2.2.3 Core Registers

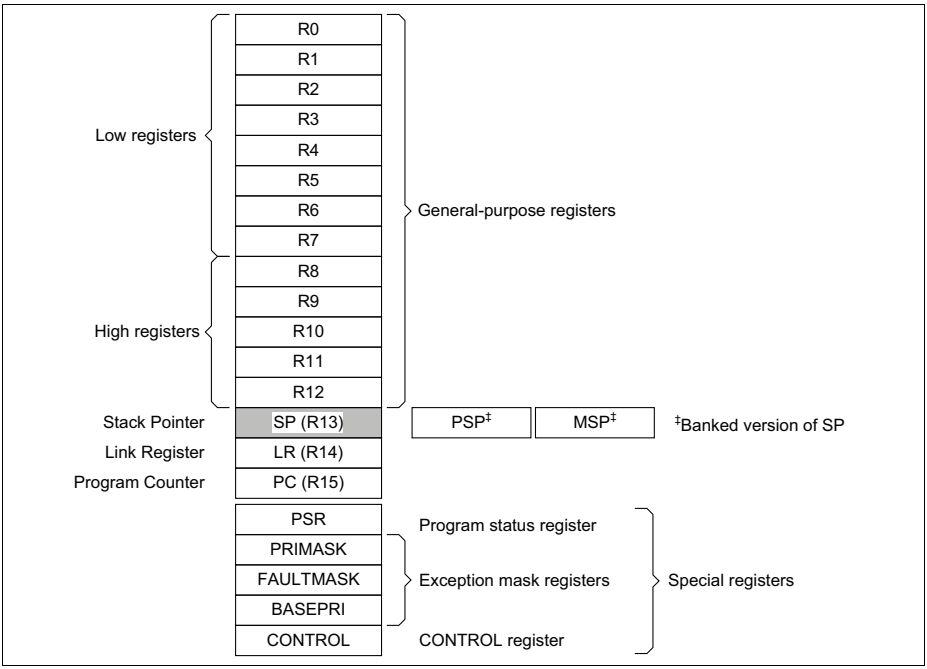


Figure 2-2 Core registers

The processor core registers are:

Table 2-2 Core register set summary

Name	Type [1]	Required privilege [2]	Reset value	Description
R0-R12	rw	Either	Unknown	General-purpose registers on **Page 2-7**
MSP	rw	Privileged	See description	Stack Pointer on **Page 2-8**
PSP	rw	Either	Unknown	Stack Pointer on **Page 2-8**
LR	rw	Either	$FFFFFFFF_H$	Link Register on **Page 2-8**
PC	rw	Either	See description	Program Counter on **Page 2-8**

Table 2-2 Core register set summary (cont'd)

Name	Type [1]	Required privilege [2]	Reset value	Description
PSR	rw	Privileged	01000000_H	Program Status Register on **Page 2-9**
ASPR	rw	Either	Unknown	Application Program Status Register on **Page 2-9**
IPSR	r	Privileged	00000000_H	Interrupt Program Status Register on **Page 2-10**
EPSR	r	Privileged	01000000_H	Execution Program Status Register on **Page 2-11**
PRIMASK	rw	Privileged	00000000_H	Priority Mask Register on **Page 2-13**
FAULTMASK	rw	Privileged	00000000_H	Fault Mask Register on **Page 2-14**
BASEPRI	rw	Privileged	00000000_H	Base Priority Mask Register on **Page 2-14**
CONTROL	rw	Privileged	00000000_H	CONTROL register on **Page 2-15**

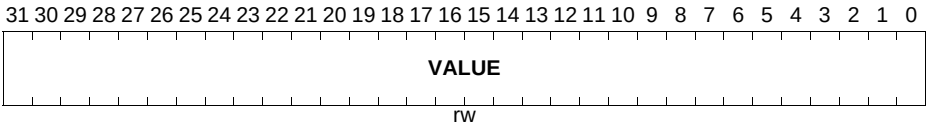
1) Describes access type during program execution in thread mode and Handler mode. Debug access can differ.

2) An entry of Either means privileged and unprivileged software can access the register.

General-purpose registers

R0-R12 are 32-bit general-purpose registers for data operations

Rx (x=0-12)
General Purpose Register Rx **Reset Value: XXXX XXXX$_H$**



Field	Bits	Type	Description
VALUE	[31:0]	rw	**Content of Register**

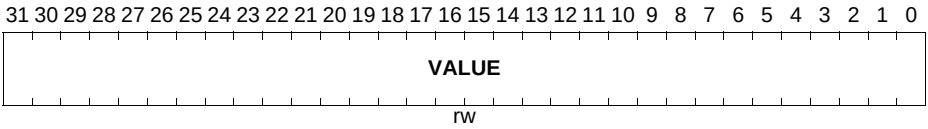
Stack Pointer

The Stack Pointer (SP) is register R13. In Thread mode, bit[1] of the CONTROL register indicates the stack pointer to use:

- 0 = Main Stack Pointer (MSP). This is the reset value.
- 1 = Process Stack Pointer (PSP).

On reset, the processor loads the MSP with the value from address 00000000_H.

SP
Stack Pointer **Reset Value: 2000 FF3C$_H$**

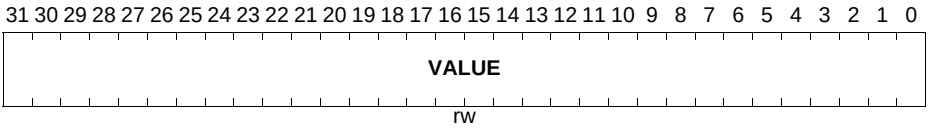


Field	Bits	Type	Description
VALUE	[31:0]	rw	**Content of Register**

Link Register

The Link Register (LR) is register R14. It stores the return information for subroutines, function calls, and exceptions. On reset, the processor sets the LR value to $FFFFFFFF_H$.

LR
Link Register **Reset Value: FFFF FFFF$_H$**

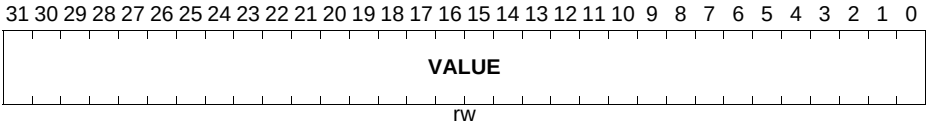


Field	Bits	Type	Description
VALUE	[31:0]	rw	**Content of Register**

Program Counter

The Program Counter (PC) is register R15. It contains the current program address. On reset, the processor loads the PC with the value of the reset vector, which is at address 00000004H. Bit[0] of the value is loaded into the EPSR T-bit at reset and must be 1.

PC
Program Counter **Reset Value: 0000 0004$_H$**



Field	Bits	Type	Description
VALUE	[31:0]	rw	**Content of Register**

Program Status Register

The Program Status Register (PSR) combines:

- Application Program Status Register (APSR)
- Interrupt Program Status Register (IPSR)
- Execution Program Status Register (EPSR)

These registers are mutually exclusive bit fields in the 32-bit PSR.

Access these registers individually or as a combination of any two or all three registers, using the register name as an argument to the MSR or MRS instructions. For example:

- read all of the registers using PSR with the MRS instruction
- write to the APSR N, Z, C, V, and Q bits using APSR_nzcvq with the MSR instruction.

The PSR combinations and attributes are:

Table 2-3 PSR register combinations

Register	Type	Combination
PSR	rw[1)2)]	APSR, EPSR, and IPSR
IEPSR	r	EPSR and IPSR
IAPSR	rw[1)]	APSR and IPSR
EAPSR	rw[2)]	APSR and EPSR

1) The processor ignores writes to the IPSR bits.

2) Reads of the EPSR bits return zero, and the processor ignores writes to the these bits

Application Program Status Register

The APSR contains the current state of the condition flags from previous instruction executions. See the register summary in **Table 2-2** on **Page 2-6** for its attributes.

APSR
Application Program Status Register

Reset Value: XXXX XXXX$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
N	Z	C	V	Q	0							GE[3:0]				0															
rw	rw	rw	rw	rw	r							rw				r															

Field	Bits	Type	Description
GE[3:0]	[19:16]	rw	**Greater than or Equal flags** Please refer also to SEL instruction.
Q	27	rw	**DSP overflow and saturation flag**
V	28	rw	**Overflow flag**
C	29	rw	**Carry or borrow flag**
Z	30	rw	**Zero flag**
N	31	rw	**Negative flag**
0	[26:20], [15:0]	r	**Reserved** Read as 0; should be written with 0.

Interrupt Program Status Register

The IPSR contains the exception type number of the current Interrupt Service Routine (ISR). See the register summary in **Table 2-2** on **Page 2-6** for its attributes.

IPSR
Interrupt Program Status Register

Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0																							ISR_NUMBER								
r																							r								

Field	Bits	Type	Description
ISR_NUMBER	[8:0]	r	**Number of the current exception** 0_D Thread mode 1_D Reserved 2_D NMI 3_D HardFault 4_D MemManage 5_D BusFault 6_D UsageFault 7_D Reserved 8_D Reserved 9_D Reserved 10_D Reserved 11_D SVCall 12_D Reserved for Debug 13_D Reserved 14_D PendSV 15_D SysTick 16_D IRQ0 ... 127_D IRQ111 Values > 127_D undefined. See Exception types on **Page 2-26** for more information.
0	[31:9]	r	**Reserved** Read as 0; should be written with 0.

Execution Program Status Register

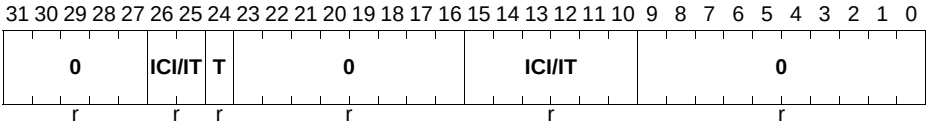
The EPSR contains the Thumb state bit, and the execution state bits for either the:

- If-Then (IT) instruction
- Interruptible-Continuable Instruction (ICI) field for an interrupted load multiple or store multiple instruction.

See the register summary in **Table 2-2** on **Page 2-6** for the EPSR attributes.

Attempts to read the EPSR directly through application software using the MSR instruction always return zero. Attempts to write the EPSR using the MSR instruction in application software are ignored.

EPSR
Execution Program Status Register **Reset Value: 0100 0000$_H$**



Field	Bits	Type	Description
ICI/IT	[26:25], [15:10]	r	**Interruptible-continuable instruction bits/Execution state bits of the IT instruction** Please refer also to IT instruction.
T	24	r	**Thumb state bit** Thumb state.
0	[31:27], [23:16], [9:0]	r	**Reserved** Read as 0; should be written with 0.

Interruptible-continuable instructions

When an interrupt occurs during the execution of an LDM, STM, PUSH, POP, VLDM, VSTM, VPUSH, or VPOP instruction, the processor:

- stops the load multiple or store multiple instruction operation temporarily
- stores the next register operand in the multiple operation to EPSR bits[15:12]

After servicing the interrupt, the processor:

- returns to the register pointed to by bits[15:12]
- resumes execution of the multiple load or store instruction.

When the EPSR holds ICI execution state, bits[26:25,11:10] are zero.

If-Then block

The If-Then block contains up to four instructions following an IT instruction. Each instruction in the block is conditional. The conditions for the instructions are either all the same, or some can be the inverse of others. See IT on page 3-122 for more information.

Thumb state

The Cortex-M4 processor only supports execution of instructions in Thumb state. The following can clear the T bit to 0:

- instructions BLX, BX and POP{PC}

- restoration from the stacked xPSR value on an exception return
- bit[0] of the vector value on an exception entry or reset.

Attempting to execute instructions when the T bit is 0 results in a fault or lockup. See Lockup on **Page 2-39** for more information.

Exception mask registers

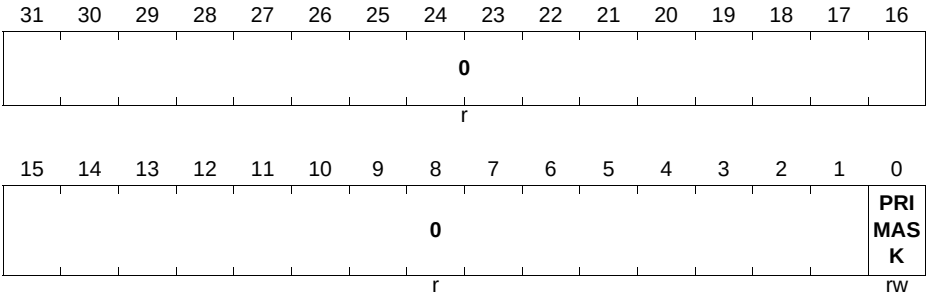
The exception mask registers disable the handling of exceptions by the processor. Disable exceptions where they might impact on timing critical tasks.

To access the exception mask registers use the MSR and MRS instructions, or the CPS instruction to change the value of PRIMASK or FAULTMASK.

Priority Mask Register

The PRIMASK register prevents activation of all exceptions with configurable priority. See the register summary in **Table 2-2** on **Page 2-6** for its attributes.

PRIMASK
Priority Mask Register **Reset Value: 0000 0000$_H$**

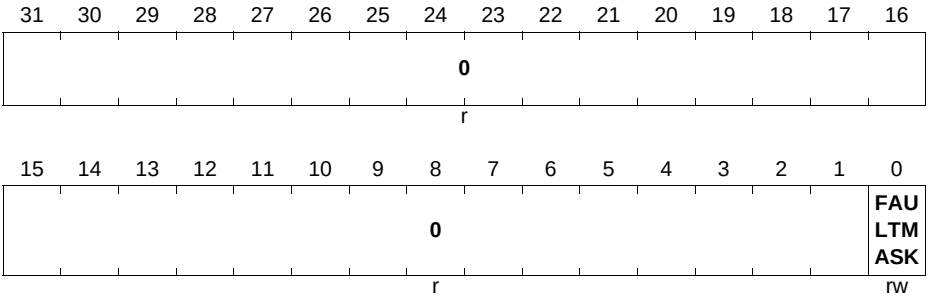


Field	Bits	Type	Description
PRIMASK	0	rw	**Priority Mask** 0_B No effect 1_B Prevents the activation of all exceptions with configurable priority.
0	[31:1]	r	**Reserved** Read as 0; should be written with 0.

Fault Mask Register

The FAULTMASK register prevents activation of all exceptions except for Non-Maskable Interrupt (NMI). See the register summary in **Table 2-2** on **Page 2-6** for its attributes.

FAULTMASK
Fault Mask Register **Reset Value: 0000 0000$_H$**



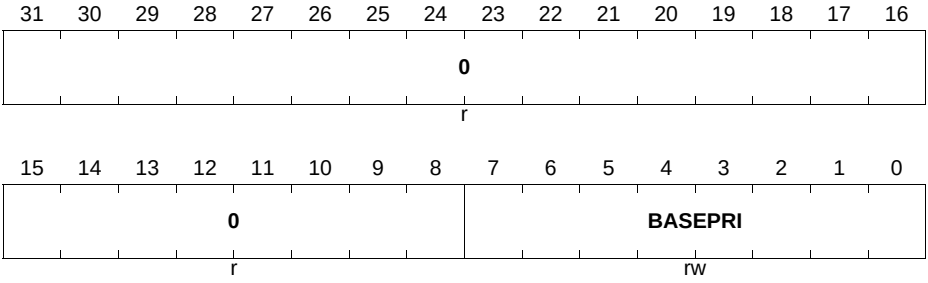
Field	Bits	Type	Description
FAULTMASK	0	rw	**Fault Mask** 0_B no effect 1_B prevents the activation of all exceptions except for NMI.
0	[31:1]	r	**Reserved** Read as 0; should be written with 0.

The processor clears the FAULTMASK bit to 0 on exit from any exception handler except the NMI handler.

Base Priority Mask Register

The BASEPRI register defines the minimum priority for exception processing. When BASEPRI is set to a nonzero value, it prevents the activation of all exceptions with the same or lower priority level as the BASEPRI value. See the register summary in **Table 2-2** on **Page 2-6** for its attributes.

BASEPRI
Base Priority Mask Register **Reset Value: 0000 0000$_H$**



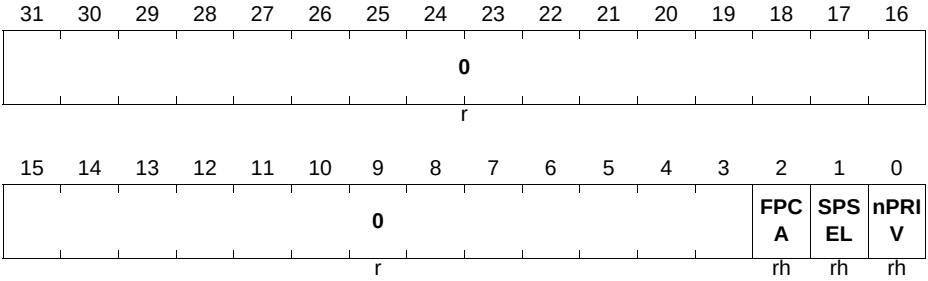
Field	Bits	Type	Description
BASEPRI[1)]	[7:0]	rw	**Priority mask bits** 0_H no effect **others**, defines the base priority for exception processing. The processor does not process any exception with a priority value greater than or equal to BASEPRI.
0	[31:8]	r	**Reserved** Read as 0; should be written with 0.

1) This field is similar to the priority fields in the interrupt priority registers. The XMC4[12]00 implements only bits[7:2] of this field, bits[1:0] read as zero and ignore writes. See **Interrupt Priority Registers** on **Page 2-90** for more information. Remember that higher priority field values correspond to lower exception priorities.

CONTROL register

The CONTROL register controls the stack used and the privilege level for software execution when the processor is in Thread mode and indicates whether the FPU state is active. See the register summary in **Table 2-2** on **Page 2-6** for its attributes.

CONTROL
CONTROL register **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
nPRIV	0	rh	**Thread mode privilege level** 0_B Privileged 1_B Unprivileged
SPSEL	1	rh	**Currently active stack pointer** In Handler mode this bit reads as zero and ignores writes. The Cortex-M4 updates this bit automatically on exception return. 0_B MSP is the current stack pointer 1_B PSP is the current stack pointer
FPCA	2	rh	**Floating-point context currently active** 0_B No floating-point context active 1_B Floating-point context active The Cortex-M4 uses this bit to determine whether to preserve floating-point state when processing an exception.
0	[31:3]	r	**Reserved** Read as 0; should be written with 0.

Handler mode always uses the MSP, so the processor ignores explicit writes to the active stack pointer bit of the CONTROL register when in Handler mode. The exception entry and return mechanisms automatically update the CONTROL register based on the EXC_RETURN value, see **Table 2-9** on **Page 2-36**.

In an OS environment, ARM recommends that threads running in Thread mode use the process stack and the kernel and exception handlers use the main stack.

By default, Thread mode uses the MSP. To switch the stack pointer used in Thread mode to the PSP, either:

- use the MSR instruction to set the Active stack pointer bit to 1.
- perform an exception return to Thread mode with the appropriate EXC_RETURN value, see **Table 2-9** on **Page 2-36**.

Note: When changing the stack pointer, software must use an ISB instruction immediately after the MSR instruction. This ensures that instructions after the ISB instruction execute using the new stack pointer.

2.2.4 Exceptions and Interrupts

The Cortex-M4 processor supports interrupts and system exceptions. The processor and the NVIC prioritize and handle all exceptions. An exception changes the normal flow of software control. The processor uses Handler mode to handle all exceptions except for reset. See Exception entry on **Page 2-33** and Exception return on **Page 2-36** for more information.

The NVIC registers control interrupt handling. See **Page 2-43** for more information.

2.2.5 Data Types

The processor:

- supports the following data types:
  - 32-bit words
  - 16-bit halfwords
  - 8-bit bytes
- manages all data memory accesses as little-endian. See Memory regions, types and attributes on **Page 2-20** for more information.

2.2.6 The Cortex Microcontroller Software Interface Standard

For a Cortex-M4 microcontroller system, the Cortex Microcontroller Software Interface Standard (CMSIS) **[2]** defines:

- a common way to:
  - access peripheral registers
  - define exception vectors
- the names of:
  - the registers of the core peripherals
  - the core exception vectors
- a device-independent interface for RTOS kernels, including a debug channel.

The CMSIS includes address definitions and data structures for the core peripherals in the Cortex-M4 processor.

CMSIS simplifies software development by enabling the reuse of template code and the combination of CMSIS-compliant software components from various middleware vendors. Software vendors can expand the CMSIS to include their peripheral definitions and access functions for those peripherals.

This document includes the register names defined by the CMSIS, and gives short descriptions of the CMSIS functions that address the processor core and the core peripherals.

Note: This document uses the register short names defined by the CMSIS. In a few cases these differ from the architectural short names that might be used in other documents.

The following sections give more information about the CMSIS:

- Power management programming hints on **Page 2-42**
- CMSIS functions on **Page 2-18**
- Using CMSIS functions to access NVIC on **Page 2-45**

For additional information please refer to **http://www.onarm.com/cmsis**

2.2.7 CMSIS functions

ISO/IEC C code cannot directly access some Cortex-M4 instructions. This section describes intrinsic functions that can generate these instructions, provided by the CMSIS and that might be provided by a C compiler. If a C compiler does not support an appropriate intrinsic function, you might have to use inline assembler to access some instructions.

The CMSIS provides the following intrinsic functions to generate instructions that ISO/IEC C code cannot directly access:

Table 2-4 CMSIS functions to generate some Cortex-M4 instructions

Instruction	CMSIS function
CPSIE I	`void __enable_irq(void)`
CPSID I	`void __disable_irq(void)`
CPSIE F	`void __enable_fault_irq(void)`
CPSID F	`void __disable_fault_irq(void)`
ISB	`void __ISB(void)`
DSB	`void __DSB(void)`
DMB	`void __DMB(void)`
REV	`uint32_t __REV(uint32_t int value)`
REV16	`uint32_t __REV16(uint32_t int value)`

Table 2-4 CMSIS functions to generate some Cortex-M4 instructions (cont'd)

Instruction	CMSIS function
REVSH	`uint32_t __REVSH(uint32_t int value)`
RBIT	`uint32_t __RBIT(uint32_t int value)`
SEV	`void __SEV(void)`
WFE	`void __WFE(void)`
WFI	`void __WFI(void)`

The CMSIS also provides a number of functions for accessing the special registers using MRS and MSR instructions:

Table 2-5 CMSIS functions to access the special registers

Special register	Access	CMSIS function
PRIMASK	Read	`uint32_t __get_PRIMASK (void)`
	Write	`void __set_PRIMASK (uint32_t value)`
FAULTMASK	Read	`uint32_t __get_FAULTMASK (void)`
	Write	`void __set_FAULTMASK (uint32_t value)`
BASEPRI	Read	`uint32_t __get_BASEPRI (void)`
	Write	`void __set_BASEPRI (uint32_t value)`
CONTROL	Read	`uint32_t __get_CONTROL (void)`
	Write	`void __set_CONTROL (uint32_t value)`
MSP	Read	`uint32_t __get_MSP (void)`
	Write	`void __set_MSP (uint32_t TopOfMainStack)`
PSP	Read	`uint32_t __get_PSP (void)`
	Write	`void __set_PSP (uint32_t TopOfProcStack)`

2.3 Memory Model

This section describes the processor memory map and the behavior of memory accesses. The processor has a fixed default memory map that provides up to 4GB of addressable memory. The memory map is:

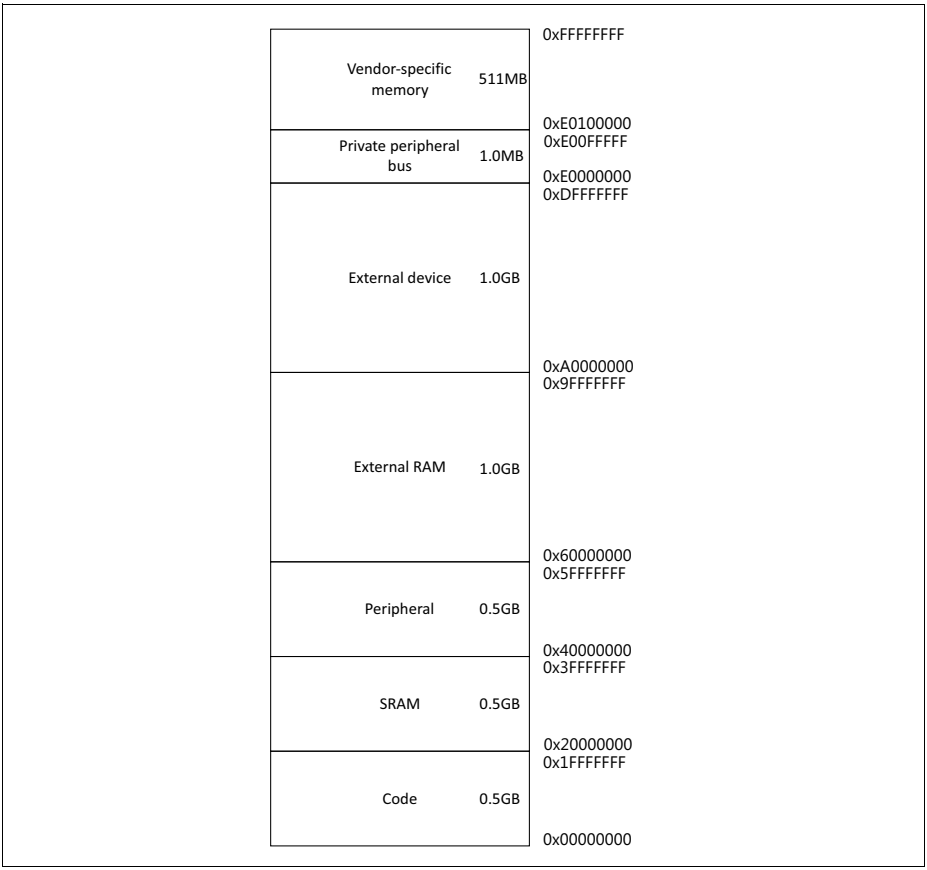


Figure 2-3 Memory map

The processor reserves regions of the Private peripheral bus (PPB) address range for core peripheral registers, see About the Private Peripherals on **Page 2-42**.

2.3.1 Memory Regions, Types and Attributes

The memory map and the programming of the MPU splits the memory map into regions. Each region has a defined memory type, and some regions have additional memory

attributes. The memory type and attributes determine the behavior of accesses to the region.

The memory types are:

Normal	The processor can re-order transactions for efficiency, or perform speculative reads.
Device	The processor preserves transaction order relative to other transactions to Device or Strongly-ordered memory.
Strongly-ordered	The processor preserves transaction order relative to all other transactions.

The different ordering requirements for Device and Strongly-ordered memory mean that the memory system can buffer a write to Device memory, but must not buffer a write to Strongly-ordered memory.

The additional memory attributes include:

Execute Never (XN)	Means the processor prevents instruction accesses. A fault exception is generated only on execution of an instruction executed from an XN region.

2.3.2 Memory System Ordering of Memory Accesses

For most memory accesses caused by explicit memory access instructions, the memory system does not guarantee that the order in which the accesses complete matches the program order of the instructions, providing this does not affect the behavior of the instruction sequence. Normally, if correct program execution depends on two memory accesses completing in program order, software must insert a memory barrier instruction between the memory access instructions. See Software ordering of memory accesses on **Page 2-23**.

However, the memory system does guarantee some ordering of accesses to Device and Strongly-ordered memory. For two memory access instructions A1 and A2, if A1 occurs before A2 in program order, the ordering of the memory accesses caused by two instructions is:

A1 \ A2	Normal access	Device access	Strongly-ordered access
Normal access	-	-	-
Device access	-	<	<
Strongly-ordered access	-	<	<

Figure 2-4 Ordering of Memory Accesses

Where:

- "-" Means that the memory system does not guarantee the ordering of the accesses.
- "<" Means that accesses are observed in program order, that is, A1 is always observed before A2.

2.3.3 Behavior of Memory Accesses

The behavior of accesses to each region in the memory map is:

Table 2-6 Memory access behavior

Address range	Memory region	Memory type[1)]	XN[1)]	Description
0x00000000-0x1FFFFFFF	Code	Normal	-	Executable region for program code. You can also put data here.
0x20000000-0x3FFFFFFF	SRAM	Normal	-	Executable region for data. You can also put code here.
0x40000000-0x5FFFFFFF	Peripheral	Device	XN	Peripherals region.
0x60000000-0x9FFFFFFF	External RAM	Normal	-	Executable region for data.
0xA0000000-0xDFFFFFFF	External device	Device	XN	External Device memory.
0xE0000000-0xE00FFFFF	Private Peripheral Bus	Strongly-ordered	XN	This region includes the NVIC, System timer, and system control block.
0xE0100000-0xFFFFFFFF	Vendor-specific device	Device	XN	Accesses to this region are to vendor-specific peripherals.

1) See Memory regions, types and attributes on **Page 2-20** for more information.

The Code, SRAM, and external RAM regions can hold programs. However, it is recommended that programs always use the Code region. This is because the processor has separate buses that enable instruction fetches and data accesses to occur simultaneously.

The MPU can override the default memory access behavior described in this section. For more information, see Memory protection unit on **Page 2-46**.

Instruction prefetch and branch prediction

The Cortex-M4 processor:

- prefetches instructions ahead of execution
- speculatively prefetches from branch target addresses.

2.3.4 Software Ordering of Memory Accesses

The order of instructions in the program flow does not always guarantee the order of the corresponding memory transactions. This is because:

- the processor can reorder some memory accesses to improve efficiency, providing this does not affect the behavior of the instruction sequence.
- The processor has multiple bus interfaces
- memory or devices in the memory map have different wait states
- some memory accesses are buffered or speculative.

Memory system ordering of memory accesses on **Page 2-21** describes the cases where the memory system guarantees the order of memory accesses. Otherwise, if the order of memory accesses is critical, software must include memory barrier instructions to force that ordering. The processor provides the following memory barrier instructions:

DMB The Data Memory Barrier (DMB) instruction ensures that outstanding memory transactions complete before subsequent memory transactions.

DSB The Data Synchronization Barrier (DSB) instruction ensures that outstanding memory transactions complete before subsequent instructions execute.

ISB The Instruction Synchronization Barrier (ISB) ensures that the effect of all completed memory transactions is recognizable by subsequent instructions.

MPU programming

Use a DSB followed by an ISB instruction or exception return to ensure that the new MPU configuration is used by subsequent instructions.

2.3.5 Memory Endianness

The processor views memory as a linear collection of bytes numbered in ascending order from zero. For example, bytes 0-3 hold the first stored word, and bytes 4-7 hold the second stored word. The XMC4[12]00 stores information “Little-endian” format.

Little-endian format

In little-endian format, the processor stores the least significant byte of a word at the lowest-numbered byte, and the most significant byte at the highest-numbered byte. For example:

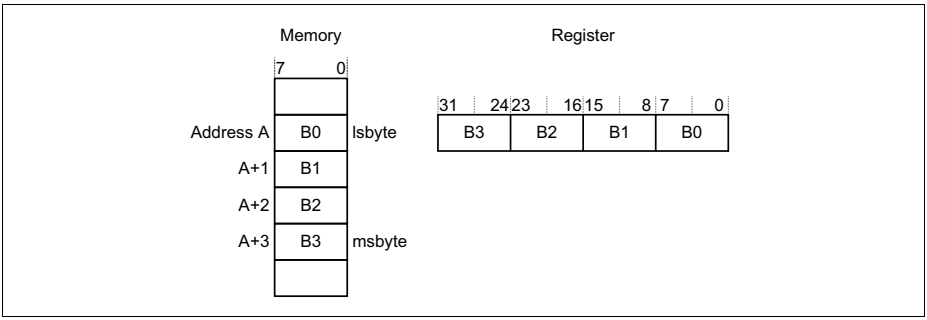


Figure 2-5 Little-endian format

2.3.6 Synchronization Primitives

The Cortex-M4 instruction set includes pairs of synchronization primitives. These provide a non-blocking mechanism that a thread or process can use to obtain exclusive access to a memory location. Software can use them to perform a guaranteed read-modify-write memory update sequence, or for a semaphore mechanism.

A pair of synchronization primitives comprises:

A Load-Exclusive instruction

Used to read the value of a memory location, requesting exclusive access to that location.

A Store-Exclusive instruction

Used to attempt to write to the same memory location, returning a status bit to a register. If this bit is:

0 it indicates that the thread or process gained exclusive access to the memory, and the write succeeds,

1 it indicates that the thread or process did not gain exclusive access to the memory, and no write was performed.

The pairs of Load-Exclusive and Store-Exclusive instructions are:

- the word instructions LDREX and STREX
- the halfword instructions LDREXH and STREXH
- the byte instructions LDREXB and STREXB.

Software must use a Load-Exclusive instruction with the corresponding Store-Exclusive instruction.

To perform an exclusive read-modify-write of a memory location, software must:

1. Use a Load-Exclusive instruction to read the value of the location.
2. Modify the value, as required.
3. Use a Store-Exclusive instruction to attempt to write the new value back to the memory location.
4. Test the returned status bit. If this bit is:
   0 The read-modify-write completed successfully.
   1 No write was performed. This indicates that the value returned at step 1 might be out of date. The software must retry the entire read-modify-write sequence.

Software can use the synchronization primitives to implement a semaphores as follows:

1. Use a Load-Exclusive instruction to read from the semaphore address to check whether the semaphore is free.
2. If the semaphore is free, use a Store-Exclusive to write the claim value to the semaphore address.
3. If the returned status bit from step 2 indicates that the Store-Exclusive succeeded then the software has claimed the semaphore. However, if the Store-Exclusive failed, another process might have claimed the semaphore after the software performed step 1.

The Cortex-M4 includes an exclusive access monitor, that tags the fact that the processor has executed a Load-Exclusive instruction.

The processor removes its exclusive access tag if:

- It executes a CLREX instruction.
- It executes a Store-Exclusive instruction, regardless of whether the write succeeds.
- An exception occurs. This means the processor can resolve semaphore conflicts between different threads.

2.3.7 Programming Hints for the Synchronization Primitives

ISO/IEC C cannot directly generate the exclusive access instructions. CMSIS provides intrinsic functions for generation of these instructions:

Table 2-7 CMSIS functions for exclusive access instructions

Instruction	CMSIS function
LDREX	`uint32_t __LDREXW (uint32_t *addr)`
LDREXH	`uint16_t __LDREXH (uint16_t *addr)`
LDREXB	`uint8_t __LDREXB (uint8_t *addr)`
STREX	`uint32_t __STREXW (uint32_t value, uint32_t *addr)`
STREXH	`uint32_t __STREXH (uint16_t value, uint16_t *addr)`
STREXB	`uint32_t __STREXB (uint8_t value, uint8_t *addr)`
CLREX	`void __CLREX (void)`

For example:

```
uint16_t value;
uint16_t *address = 0x20001002;
value = __LDREXH (address);     // load 16-bit value from memory
address 0x20001002
```

2.4 Instruction Set

The Cortex-M4 instruction set reference is available through **[1]**

2.5 Exception Model

This section describes the exception model. It describes:

- Exception states
- Exception types
- Exception handlers on **Page 2-27**
- Vector table on **Page 2-30**
- Exception priorities on **Page 2-31**
- Interrupt priority grouping on **Page 2-31**
- Exception entry and return on **Page 2-32**

2.5.1 Exception States

Each exception is in one of the following states:

Inactive	The exception is not active and not pending.
Pending	The exception is waiting to be serviced by the processor. An interrupt request from a peripheral or from software can change the state of the corresponding interrupt to pending.
Active	An exception that is being serviced by the processor but has not completed. *Note: An exception handler can interrupt the execution of another exception handler. In this case both exceptions are in the active state.*
Active and pending	The exception is being serviced by the processor and there is a pending exception from the same source.

2.5.2 Exception Types

The exception types are:

Reset	Reset is invoked on power up or a warm reset. The exception model treats reset as a special form of exception. When reset is asserted, the operation of the processor stops, potentially at any point in an instruction. When reset is deasserted, execution restarts from the address provided by the reset entry in the vector table. Execution restarts as privileged execution in Thread mode.
NMI	A NonMaskable Interrupt (NMI) can be signalled by a peripheral or triggered by software. This is the highest priority exception other than reset. It is permanently enabled and has a fixed priority of -2. NMIs cannot be: • masked or prevented from activation by any other exception • preempted by any exception other than Reset.
HardFault	A HardFault is an exception that occurs because of an error during exception processing, or because an exception cannot be managed by any other exception mechanism. HardFaults have a fixed priority of -1, meaning they have higher priority than any exception with configurable priority.
MemManage	A MemManage fault is an exception that occurs because of a memory protection related fault. The MPU or the fixed memory protection constraints determines this fault, for both instruction and data memory transactions. This fault is always used to abort instruction accesses to Execute Never (XN) memory regions.

BusFault A BusFault is an exception that occurs because of a memory related fault for an instruction or data memory transaction. This might be from an error detected on a bus in the memory system.

UsageFault A UsageFault is an exception that occurs because of a fault related to instruction execution. This includes:

- an undefined instruction
- an illegal unaligned access
- invalid state on instruction execution
- an error on exception return.

The following can cause a UsageFault when the core is configured to report them:

- an unaligned address on word and halfword memory access
- division by zero.

SVCall A supervisor call (SVC) is an exception that is triggered by the SVC instruction. In an OS environment, applications can use SVC instructions to access OS kernel functions and device drivers.

PendSV PendSV is an interrupt-driven request for system-level service. In an OS environment, use PendSV for context switching when no other exception is active.

SysTick A SysTick exception is an exception the system timer generates when it reaches zero. Software can also generate a SysTick exception. In an OS environment, the processor can use this exception as system tick.

Interrupt (IRQ) A interrupt, or IRQ, is an exception signalled by a peripheral, or generated by a software request. All interrupts are asynchronous to instruction execution. In the system, peripherals use interrupts to communicate with the processor.

Table 2-8 Properties of the different exception types

Exception number[1)]	IRQ number[1)]	Exception type	Priority	Vector address or offset[2)]	Activation
1	-	Reset	-3, the highest	0x00000004	Asynchronous
2	-14	NMI	-2	0x00000008	Asynchronous
3	-13	HardFault	-1	0x0000000C	-
4	-12	MemManage	Configurable[3)]	0x00000010	Synchronous

Table 2-8 Properties of the different exception types (cont'd)

Exception number[1]	IRQ number[1]	Exception type	Priority	Vector address or offset[2]	Activation
5	-11	BusFault	Configurable[3]	0x00000014	Synchronous when precise, asynchronous when imprecise
6	-10	UsageFault	Configurable[3]	0x00000018	Synchronous
7-10	-	Reserved	-	-	-
11	-5	SVCall	Configurable[3]	0x0000002C	Synchronous
12-13	-	Reserved	-	-	-
14	-2	PendSV	Configurable[3]	0x00000038	Asynchronous
15	-1	SysTick	Configurable[3]	0x0000003C	Asynchronous
16 and above	0 and above	Interrupt (IRQ)	Configurable[4]	0x00000040 and above[5]	Asynchronous

1) To simplify the software layer, the CMSIS only uses IRQ numbers and therefore uses negative values for exceptions other than interrupts. The IPSR returns the Exception number, see Interrupt Program Status Register on **Page 2-10**.

2) See Vector table for more information.

3) See System Handler Priority Registers on **Page 2-70**

4) See Interrupt Priority Registers on **Page 2-90**.

5) Increasing in steps of 4.

For an asynchronous exception, other than reset, the processor can execute another instruction between when the exception is triggered and when the processor enters the exception handler.

Privileged software can disable the exceptions that **Table 2-8** on **Page 2-28** shows as having configurable priority, see:

- System Handler Control and State Register on **Page 2-72**
- Interrupt Clear-enable Registers on **Page 2-88**.

For more information about HardFaults, MemManage faults, BusFaults, and UsageFaults, see Fault handling on **Page 2-36**.

2.5.3 Exception Handlers

The processor handles exceptions using:

Interrupt Service Routines (ISRs)	Interrupts IRQ0 to IRQ111 are the exceptions handled by ISRs.
Fault handlers	HardFault, MemManage fault, UsageFault, and BusFault are fault exceptions handled by the fault handlers.
System handlers	NMI, PendSV, SVCall SysTick, and the fault exceptions are all system exceptions that are handled by system handlers.

2.5.4 Vector Table

The vector table contains the reset value of the stack pointer, and the start addresses, also called exception vectors, for all exception handlers. **Figure 2-6** on **Page 2-30** shows the order of the exception vectors in the vector table. The least-significant bit of each vector must be 1, indicating that the exception handler is Thumb code, see Thumb state on **Page 2-12**.

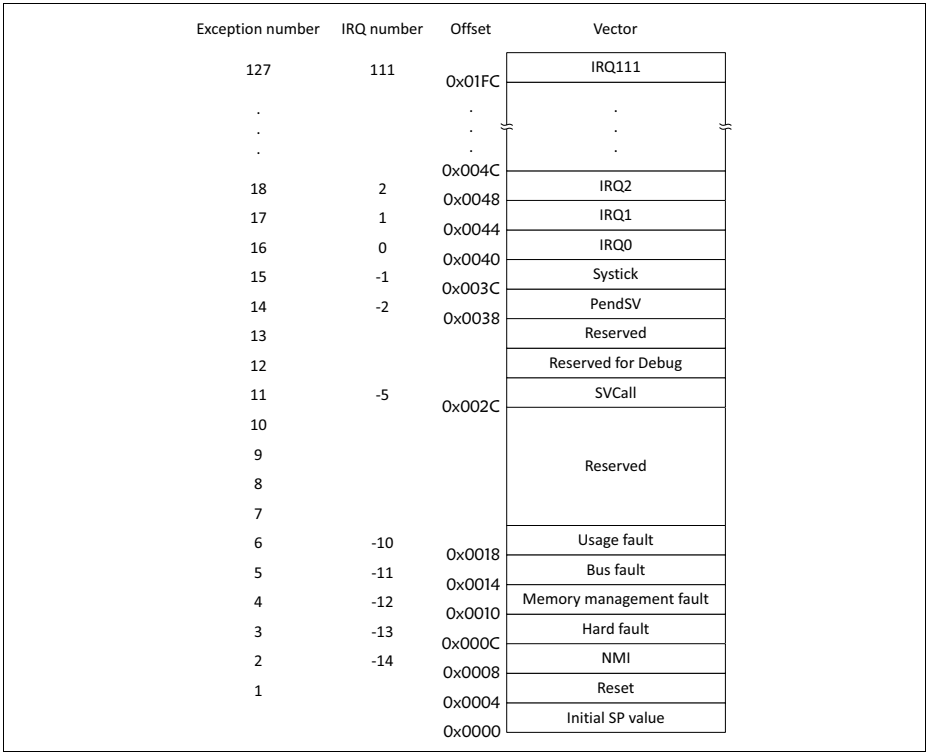


Figure 2-6 Vector table

On system reset, the vector table is fixed at address 0x00000000. Privileged software can write to the VTOR to relocate the vector table start address to a different memory location, in the range 0x00000400 to 0x3FFFFC00, see Vector Table Offset Register on **Page 2-63**.

2.5.5 Exception Priorities

As **Table 2-8** on **Page 2-28** shows, all exceptions have an associated priority, with:

- a lower priority value indicating a higher priority
- configurable priorities for all exceptions except Reset, HardFault, and NMI.

If software does not configure any priorities, then all exceptions with a configurable priority have a priority of 0. For information about configuring exception priorities see

- System Handler Priority Registers on **Page 2-70**
- Interrupt Priority Registers on **Page 2-90**.

Note: Configurable priority values are in the range 0-63. This means that the Reset, HardFault, and NMI exceptions, with fixed negative priority values, always have higher priority than any other exception.

For example, assigning a higher priority value to IRQ[0] and a lower priority value to IRQ[1] means that IRQ[1] has higher priority than IRQ[0]. If both IRQ[1] and IRQ[0] are asserted, IRQ[1] is processed before IRQ[0].

If multiple pending exceptions have the same priority, the pending exception with the lowest exception number takes precedence. For example, if both IRQ[0] and IRQ[1] are pending and have the same priority, then IRQ[0] is processed before IRQ[1].

When the processor is executing an exception handler, the exception handler is preempted if a higher priority exception occurs. If an exception occurs with the same priority as the exception being handled, the handler is not preempted, irrespective of the exception number. However, the status of the new interrupt changes to pending.

2.5.6 Interrupt Priority Grouping

To increase priority control in systems with interrupts, the NVIC supports priority grouping. This divides each interrupt priority register entry into two fields:

- an upper field that defines the group priority
- a lower field that defines a subpriority within the group.

Only the group priority determines preemption of interrupt exceptions. When the processor is executing an interrupt exception handler, another interrupt with the same group priority as the interrupt being handled does not preempt the handler,

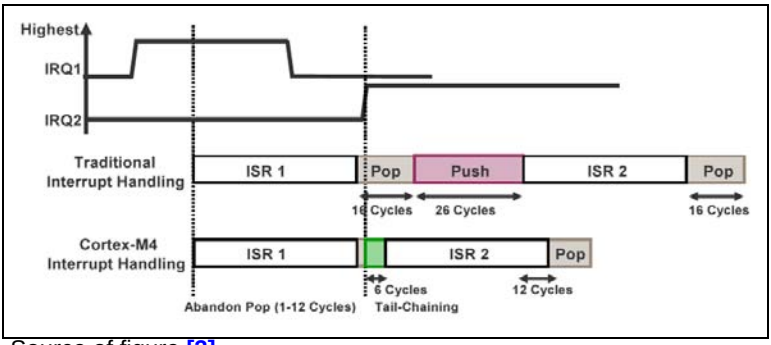
If multiple pending interrupts have the same group priority, the subpriority field determines the order in which they are processed. If multiple pending interrupts have the same group priority and subpriority, the interrupt with the lowest IRQ number is processed first.

For information about splitting the interrupt priority fields into group priority and subpriority, see Application Interrupt and Reset Control Register on **Page 2-64**.

2.5.7 Exception Entry and Return

Descriptions of exception handling use the following terms:

Preemption When the processor is executing an exception handler, an exception can preempt the exception handler if its priority is higher than the priority of the exception being handled. See Interrupt priority grouping for more information about preemption by an interrupt.
When one exception preempts another, the exceptions are called nested exceptions. See Exception entry on **Page 2-33** more information.



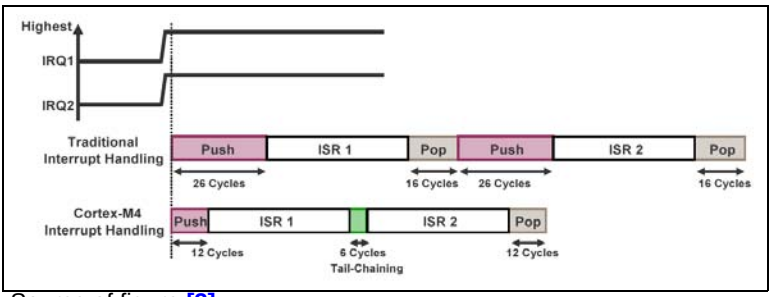
Source of figure **[3]**.

Return This occurs when the exception handler is completed, and:

- there is no pending exception with sufficient priority to be serviced
- the completed exception handler was not handling a late-arriving exception.

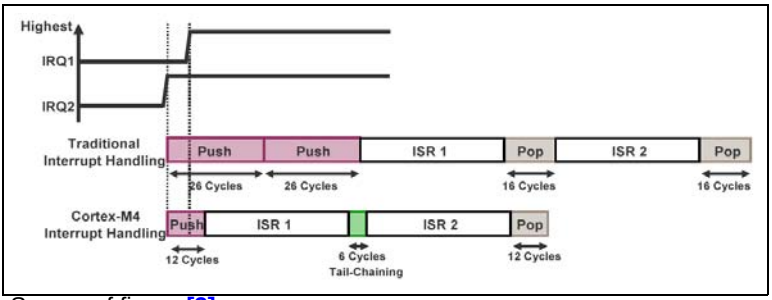
The processor pops the stack and restores the processor state to the state it had before the interrupt occurred. See Exception return on **Page 2-36** for more information.

Tail-chaining This mechanism speeds up exception servicing. On completion of an exception handler, if there is a pending exception that meets the requirements for exception entry, the stack pop is skipped and control transfers to the new exception handler.



Source of figure [3].

Late-arriving This mechanism speeds up preemption. If a higher priority exception occurs during state saving for a previous exception, the processor switches to handle the higher priority exception and initiates the vector fetch for that exception. State saving is not affected by late arrival because the state saved is the same for both exceptions. Therefore the state saving continues uninterrupted. The processor can accept a late arriving exception until the first instruction of the exception handler of the original exception enters the execute stage of the processor. On return from the exception handler of the late-arriving exception, the normal tail-chaining rules apply.



Source of figure [3].

Exception entry

Exception entry occurs when there is a pending exception with sufficient priority and either:

- the processor is in Thread mode
- the new exception is of higher priority than the exception being handled, in which case the new exception preempts the original exception.

When one exception preempts another, the exceptions are nested.

Sufficient priority means the exception has more priority than any limits set by the mask registers, see Exception mask registers on **Page 2-13**. An exception with less priority than this is pending but is not handled by the processor.

When the processor takes an exception, unless the exception is a tail-chained or a late-arriving exception, the processor pushes information onto the current stack. This operation is referred to as stacking and the structure of eight data words is referred as the stack frame.

When using floating-point routines, the Cortex-M4 processor automatically stacks the architected floating-point state on exception entry. **Figure 2-7** on **Page 2-35** shows the Cortex-M4 stack frame layout when floating-point state is preserved on the stack as the result of an interrupt or an exception.

*Note: Where stack space for floating-point state is not allocated, the stack frame is the same as that of ARMv7-M implementations without an FPU. **Figure 2-7** on **Page 2-35** shows this stack frame also.*

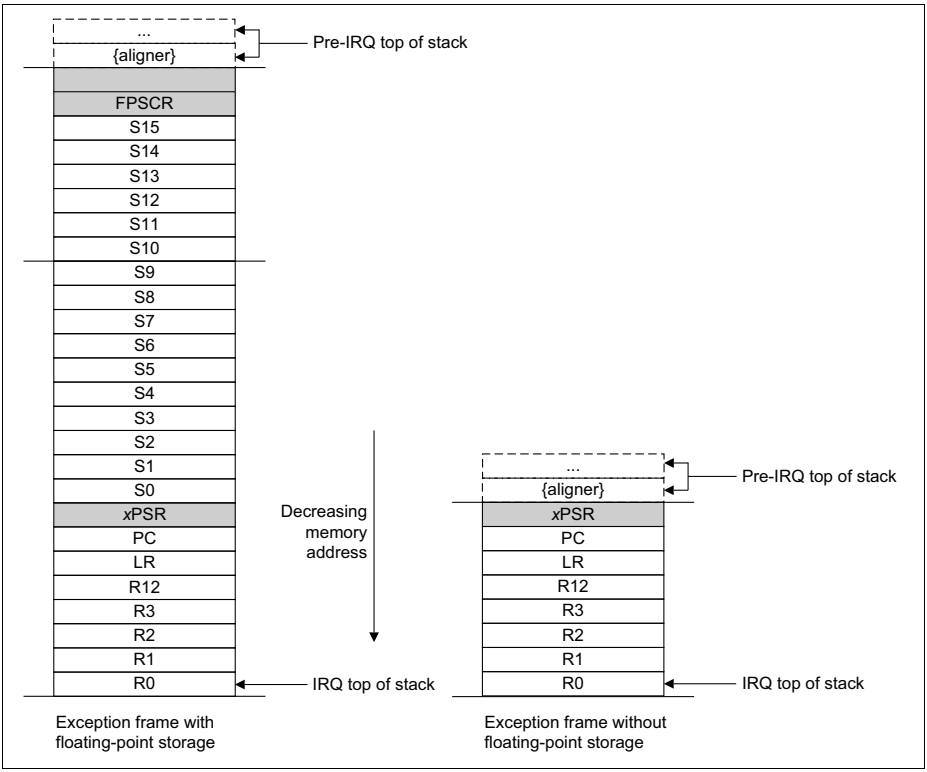


Figure 2-7 Exception stack frame

Immediately after stacking, the stack pointer indicates the lowest address in the stack frame. The alignment of the stack frame is controlled via the STKALIGN bit of the Configuration Control Register (CCR).

The stack frame includes the return address. This is the address of the next instruction in the interrupted program. This value is restored to the PC at exception return so that the interrupted program resumes.

In parallel to the stacking operation, the processor performs a vector fetch that reads the exception handler start address from the vector table. When stacking is complete, the processor starts executing the exception handler. At the same time, the processor writes an EXC_RETURN value to the LR. This indicates which stack pointer corresponds to the stack frame and what operation mode the processor was in before the entry occurred.

If no higher priority exception occurs during exception entry, the processor starts executing the exception handler and automatically changes the status of the corresponding pending interrupt to active.

If another higher priority exception occurs during exception entry, the processor starts executing the exception handler for this exception and does not change the pending status of the earlier exception. This is the late arrival case.

Exception return

Exception return occurs when the processor is in Handler mode and executes one of the following instructions to load the EXC_RETURN value into the PC:

- an LDM or POP instruction that loads the PC
- an LDR instruction with PC as the destination
- a BX instruction using any register.

EXC_RETURN is the value loaded into the LR on exception entry. The exception mechanism relies on this value to detect when the processor has completed an exception handler. The lowest five bits of this value provide information on the return stack and processor mode. **Table 2-9** shows the EXC_RETURN values with a description of the exception return behavior.

All EXC_RETURN values have bits[31:5] set to one. When this value is loaded into the PC it indicates to the processor that the exception is complete, and the processor initiates the appropriate exception return sequence.

Table 2-9 Exception return behavior

EXC_RETURN[31:0]	Description
0xFFFFFFF1	Return to Handler mode, exception return uses non-floating-point state from the MSP and execution uses MSP after return.
0xFFFFFFF9	Return to Thread mode, exception return uses non-floating-point state from MSP and execution uses MSP after return.
0xFFFFFFFD	Return to Thread mode, exception return uses non-floating-point state from the PSP and execution uses PSP after return.
0xFFFFFFE1	Return to Handler mode, exception return uses floating-point-state from MSP and execution uses MSP after return.
0xFFFFFFE9	Return to Thread mode, exception return uses floating-point state from MSP and execution uses MSP after return.
0xFFFFFFED	Return to Thread mode, exception return uses floating-point state from PSP and execution uses PSP after return.

2.6 Fault Handling

Faults are a subset of the exceptions, see Exception model on **Page 2-26**. Faults are generated by:

- a bus error on:

  - an instruction fetch or vector table load
  - a data access.
- an internally-detected error such as an undefined instruction
- attempting to execute an instruction from a memory region marked as Non-Executable (XN).
- a privilege violation or an attempt to access an unmanaged region causing an MPU fault

2.6.1 Fault Types

Table 2-10 shows the types of fault, the handler used for the fault, the corresponding fault status register, and the register bit that indicates that the fault has occurred. See Configurable Fault Status Register on page 4-24 for more information about the fault status registers.

Table 2-10 Faults

Fault	Handler	Bit name	Fault status register
Bus error on a vector read	HardFault	VECTTBL	HardFault Status Register on **Page 2-81**
Fault escalated to a hard fault		FORCED	
MPU or default memory map mismatch:	MemManage	-	-
on instruction access		IACCVIOL[1)]	MemManage Fault Address Register on **Page 2-82**
on data access		DACCVIOL	
during exception stacking		MSTKERR	
during exception unstacking		MUNSKERR	
during lazy floating-point state preservation		MLSPERR	

Table 2-10 Faults (cont'd)

Fault	Handler	Bit name	Fault status register
Bus error:	BusFault	-	-
during exception stacking		STKERR	BusFault Status Register on Page 2-74
during exception unstacking		UNSTKERR	
during instruction prefetch		IBUSERR	
during lazy floating-point state preservation		LSPERR	
Precise data bus error		PRECISERR	
Imprecise data bus error		IMPRECISERR	
Attempt to access a coprocessor	UsageFault	NOCP	UsageFault Status Register on Page 2-74
Undefined instruction		UNDEFINSTR	
Attempt to enter an invalid instruction set state[2)]		INVSTATE	
Invalid EXC_RETURN value		INVPC	
Illegal unaligned load or store		UNALIGNED	
Divide By 0		DIVBYZERO	

1) Occurs on an access to an XN region even if the processor does not include an MPU or the MPU is disabled.

2) Attempting to use an instruction set other than the Thumb instruction set or returns to a non load/store-multiple instruction with ICI continuation.

2.6.2 Fault Escalation and Hard Faults

All faults exceptions except for HardFault have configurable exception priority, see System Handler Priority Registers on page 4-21. Software can disable execution of the handlers for these faults, see System Handler Control and State Register on page 4-23.

Usually, the exception priority, together with the values of the exception mask registers, determines whether the processor enters the fault handler, and whether a fault handler can preempt another fault handler. as described in Exception model on **Page 2-26**.

In some situations, a fault with configurable priority is treated as a HardFault. This is called priority escalation, and the fault is described as escalated to HardFault. Escalation to HardFault occurs when:

- A fault handler causes the same kind of fault as the one it is servicing. This escalation to HardFault occurs because a fault handler cannot preempt itself because it must have the same priority as the current priority level.
- A fault handler causes a fault with the same or lower priority as the fault it is servicing. This is because the handler for the new fault cannot preempt the currently executing fault handler.
- An exception handler causes a fault for which the priority is the same as or lower than the currently executing exception.
- A fault occurs and the handler for that fault is not enabled.

If a BusFault occurs during a stack push when entering a BusFault handler, the BusFault does not escalate to a HardFault. This means that if a corrupted stack causes a fault, the fault handler executes even though the stack push for the handler failed. The fault handler operates but the stack contents are corrupted.

Note: Only Reset and NMI can preempt the fixed priority HardFault. A HardFault can preempt any exception other than Reset, NMI, or another HardFault.

2.6.3 Fault Status Registers and Fault Address Registers

The fault status registers indicate the cause of a fault. For BusFaults and MemManage faults, the fault address register indicates the address accessed by the operation that caused the fault, as shown in **Table 2-11**.

Table 2-11 Fault status and fault address registers

Handler	Status register name	Address register name	Register description
HardFault	HFSR	-	HardFault Status Register on **Page 2-81**
MemManage	MMFSR	MMFAR	MemManage Fault Status Register **Page 2-74** MemManage Fault Address Register **Page 2-82**
BusFault	BFSR	BFAR	BusFault Status Register on **Page 2-74** BusFault Address Register on **Page 2-83**
UsageFault	UFSR	-	UsageFault Status Register on **Page 2-74**

2.6.4 Lockup

The processor enters a lockup state if a fault occurs when executing the NMI or HardFault handlers. When the processor is in lockup state it does not execute any instructions. The processor remains in lockup state until either:

- it is reset

- an NMI occurs
- it is halted by a debugger

Note: If lockup state occurs from the NMI handler a subsequent NMI does not cause the processor to leave lockup state.

2.7 Power Management

The Cortex-M4 processor sleep modes reduce power consumption:

- Sleep mode stops the processor clock.
- Deep sleep mode stops the system clock and switches off the PLL and flash memory.

The SLEEPDEEP bit of the SCR selects which sleep mode is used, see System Control Register on **Page 2-67**. For more information about the behavior of the sleep modes see section “Power Management” in SCU chapter.

The following section describes the mechanisms for entering sleep mode, and the conditions for waking up from sleep mode.

2.7.1 Entering Sleep Mode

This section describes the mechanisms software can use to put the processor into sleep mode

The system can generate spurious wakeup events, for example a debug operation wakes up the processor. Therefore software must be able to put the processor back into sleep mode after such an event. A program might have an idle loop to put the processor back to sleep mode.

Wait for interrupt

The wait for interrupt instruction, WFI, causes immediate entry to sleep mode unless the wake-up condition is true, see Wakeup from WFI or sleep-on-exit on **Page 2-41**. When the processor executes a WFI instruction it stops executing instructions and enters sleep mode.

Wait for event

The wait for event instruction, WFE, causes entry to sleep mode depending on the value of a one-bit event register. When the processor executes a WFE instruction, it checks the value of the event register:

0 The processor stops executing instructions and enters sleep mode.
1 The processor clears the register to 0 and continues executing instructions without entering sleep mode.

If the event register is 1, this indicate that the processor must not enter sleep mode on execution of a WFE instruction. Typically, this is because an external event signal is

asserted, or a processor in the system has executed an SEV instruction, see SEV on page 3-166. Software cannot access this register directly.

Sleep-on-exit

If the SLEEPONEXIT bit of the SCR is set to 1, when the processor completes the execution of all exception handlers it returns to Thread mode and immediately enters sleep mode. Use this mechanism in applications that only require the processor to run when an exception occurs.

2.7.2 Wakeup from Sleep Mode

The conditions for the processor to wakeup depend on the mechanism that cause it to enter sleep mode.

Wakeup from WFI or sleep-on-exit

Normally, the processor wakes up only when it detects an exception with sufficient priority to cause exception entry. Some embedded systems might have to execute system restore tasks after the processor wakes up, and before it executes an interrupt handler. To achieve this set the PRIMASK bit to 1 and the FAULTMASK bit to 0. If an interrupt arrives that is enabled and has a higher priority than current exception priority, the processor wakes up but does not execute the interrupt handler until the processor sets PRIMASK to zero. For more information about PRIMASK and FAULTMASK see Exception mask registers on **Page 2-13**.

Wakeup from WFE

The processor wakes up if:

- it detects an exception with sufficient priority to cause exception entry
- it detects an external event signal, see The external event input

In addition, if the SEVONPEND bit in the SCR is set to 1, any new pending interrupt triggers an event and wakes up the processor, even if the interrupt is disabled or has insufficient priority to cause exception entry. For more information about the SCR see System Control Register on **Page 2-67**.

2.7.3 The External Event Input

The processor provides an external event input signal. Peripherals can drive this signal, either to wake the processor from WFE, or to set the internal WFE event register to one to indicate that the processor must not enter sleep mode on a later WFE instruction. See Wait for event on **Page 2-40** for more information.

2.7.4 Power Management Programming Hints

ISO/IEC C cannot directly generate the WFI and WFE instructions. The CMSIS provides the following functions for these instructions:

```
void __WFE(void)   // Wait for Event
void __WFI(void)   // Wait for Interrupt
```

2.8 Private Peripherals

The following sections are the reference material for the ARM Cortex-M4 core peripherals.

2.8.1 About the Private Peripherals

The address map of the Private Peripheral Bus (PPB) is:

Table 2-12 Core peripheral register regions

Address	Core peripheral	Description
0xE000E008-0xE000E00F	System control block	**Section 2.8.2** and **Section 2.9.1**
0xE000E010-0xE000E01F	System timer	**Section 2.8.3** and **Section 2.9.2**
0xE000E100-0xE000E4EF	Nested Vectored Interrupt Controller	**Section 2.8.4** and **Section 2.9.3**
0xE000ED00-0xE000ED3F	System control block	**Section 2.8.2** and **Section 2.9.1**
0xE000ED90-0xE000EDB8	Memory protection unit	**Section 2.8.5** and **Section 2.9.4**
0xE000EF00-0xE000EF03	Nested Vectored Interrupt Controller	**Section 2.8.4** and **Section 2.9.3**
0xE000EF30-0xE000EF44	Floating Point Unit	**Section 2.8.6** and **Section 2.9.5**

2.8.2 System control block

The System control block (SCB) provides system implementation information, and system control. This includes configuration, control, and reporting of the system exceptions. The system control block registers are:

2.8.2.1 System control block design hints and tips

Ensure software uses aligned accesses of the correct size to access the system control block registers:

- except for the CFSR and SHPR1-SHPR3, it must use aligned word accesses
- for the CFSR and SHPR1-SHPR3 it can use byte or aligned halfword or word accesses.

The processor does not support unaligned accesses to system control block registers.

In a fault handler, to determine the true faulting address:

1. Read and save the MMFAR or BFAR value.
2. Read the MMARVALID bit in the MMFSR, or the BFARVALID bit in the BFSR. The MMFAR or BFAR address is valid only if this bit is 1.

Software must follow this sequence because another higher priority exception might change the MMFAR or BFAR value. For example, if a higher priority handler preempts the current fault handler, the other fault might change the MMFAR or BFAR value.

2.8.3 System timer, SysTick

The processor has a 24-bit system timer, SysTick, that counts down from the reload value to zero, reloads, that is wraps to, the value in the **SYST_RVR** register on the next clock edge, then counts down on subsequent clocks.

Note: When the processor is halted for debugging the counter does not decrement.

2.8.3.1 SysTick design hints and tips

The SysTick counter runs on the clock selected by **SYST_CSR**.CLKSOURCE. If the selected clock signal is stopped, the SysTick counter stops.

Ensure software uses aligned word accesses to access the SysTick registers.

The SysTick counter reload and current value are undefined at reset, the correct initialization sequence for the SysTick counter is:

1. Program reload value.
2. Clear current value.
3. Program Control and Status register.

2.8.4 Nested Vectored Interrupt Controller (NVIC)

This section describes the NVIC and the registers it uses. The XMC4[12]00 NVIC supports:

- 112 interrupts.
- A programmable priority level of 0-63 for each interrupt. A higher level corresponds to a lower priority, so level 0 is the highest interrupt priority.
- Level and pulse detection of interrupt signals.

- Dynamic reprioritization of interrupts.
- Grouping of priority values into group priority and subpriority fields.
- Interrupt tail-chaining.
- An external Non-maskable interrupt (NMI)

The processor automatically stacks its state on exception entry and unstacks this state on exception exit, with no instruction overhead. This provides low latency exception handling. The hardware implementation of the NVIC registers is:

2.8.4.1 Level-sensitive and pulse interrupts

The processor supports both level-sensitive and pulse interrupts. Pulse interrupts are also described as edge-triggered interrupts.

A level-sensitive interrupt is held asserted until the peripheral deasserts the interrupt signal. Typically this happens because the ISR accesses the peripheral, causing it to clear the interrupt request. A pulse interrupt is an interrupt signal sampled synchronously on the rising edge of the processor clock. To ensure the NVIC detects the interrupt, the peripheral must assert the interrupt signal for at least one clock cycle, during which the NVIC detects the pulse and latches the interrupt.

When the processor enters the ISR, it automatically removes the pending state from the interrupt, see next section. For a level-sensitive interrupt, if the signal is not deasserted before the processor returns from the ISR, the interrupt becomes pending again, and the processor must execute its ISR again. This means that the peripheral can hold the interrupt signal asserted until it no longer requires servicing.

See section “Service Request Distribution” in the “Service Request Processing” chapter for details about which interrupts are level-based and which are pulsed.

Hardware and software control of interrupts

The Cortex-M4 latches all interrupts. A peripheral interrupt becomes pending for one of the following reasons:

- the NVIC detects that the interrupt signal is HIGH and the interrupt is not active
- the NVIC detects a rising edge on the interrupt signal
- software writes to the corresponding interrupt set-pending register bit, see Interrupt Set-pending Registers on **Page 2-89** or to the STIR to make an interrupt pending, see Software Trigger Interrupt Register on **Page 2-92**.

A pending interrupt remains pending until one of the following:

- The processor enters the ISR for the interrupt. This changes the state of the interrupt from pending to active. Then:
  - For a level-sensitive interrupt, when the processor returns from the ISR, the NVIC samples the interrupt signal. If the signal is asserted, the state of the interrupt changes to pending, which might cause the processor to immediately re-enter the ISR. Otherwise, the state of the interrupt changes to inactive.

  - For a pulse interrupt, the NVIC continues to monitor the interrupt signal, and if this is pulsed the state of the interrupt changes to pending and active. In this case, when the processor returns from the ISR the state of the interrupt changes to pending, which might cause the processor to immediately re-enter the ISR.
    If the interrupt signal is not pulsed while the processor is in the ISR, when the processor returns from the ISR the state of the interrupt changes to inactive.
- Software writes to the corresponding interrupt clear-pending register bit.
  For a level-sensitive interrupt, if the interrupt signal is still asserted, the state of the interrupt does not change. Otherwise, the state of the interrupt changes to inactive.
  For a pulse interrupt, state of the interrupt changes to:
  - inactive, if the state was pending
  - active, if the state was active and pending.

2.8.4.2 NVIC design hints and tips

Ensure software uses correctly aligned register accesses. The processor does not support unaligned accesses to NVIC registers. See the individual register descriptions for the supported access sizes.

A interrupt can enter pending state even if it is disabled. Disabling an interrupt only prevents the processor from taking that interrupt.

Before programming VTOR to relocate the vector table, ensure the vector table entries of the new vector table are setup for fault handlers, NMI and all enabled exception like interrupts. For more information see Vector Table Offset Register on **Page 2-63**.

2.8.4.3 Using CMSIS functions to access NVIC

CMSIS functions enable software portability between different Cortex-M profile processors. To ensure Cortex-M portability, use the functions marked for Cortex-M portability in the table below.

CMSIS provides a number of functions for NVIC control, including:

Table 2-13 CMSIS functions for NVIC control

CMSIS interrupt control function	Description	Cortex-M Portable
`void NVIC_SetPriorityGrouping( uint32_t priority_grouping)`	Set the priority grouping.	No
`uint32_t NVIC_GetPriorityGrouping( void)`	Get the priority grouping.	No
`void NVIC_EnableIRQ( IRQn_t IRQn)`	Enables IRQn.	Yes

Table 2-13 CMSIS functions for NVIC control (cont'd)

CMSIS interrupt control function	Description	Cortex-M Portable
`void NVIC_DisableIRQ( IRQn_t IRQn)`	Disables IRQn.	Yes
`uint32_t NVIC_GetPendingIRQ( IRQn_t IRQn)`	Return IRQ-Number (true) if IRQn is pending.	Yes
`void NVIC_SetPendingIRQ( IRQn_t IRQn)`	Set IRQn pending.	Yes
`void NVIC_ClearPendingIRQ( IRQn_t IRQn)`	Clear IRQn pending.	Yes
`uint32_t NVIC_GetActive( IRQn_t IRQn)`	Return the IRQ number of the active interrupt.	No
`void NVIC_SetPriority( IRQn_t IRQn, uint32_t priority)`	Set priority for IRQn.	Yes
`uint32_t NVIC_GetPriority( IRQn_t IRQn)`	Read priority of IRQn.	Yes
`uint32_t NVIC_EncodePriority( uint32_t PriorityGroup, uint32_t PreemptPriority, uint32_t SubPriority)`	Encodes the priority for an interrupt with the given priority group, preemptive priority value and sub priority value.	No
`void NVIC_DecodePriority( uint32_t Priority, uint32_t PriorityGroup, uint32_t* pPreemptPriority, uint32_t* pSubPriority)`	Decodes an interrupt priority value with the given priority group to preemptive priority value and sub priority value.	No
`void NVIC_SystemReset(void)`	Reset the system	Yes

The parameter IRQn is the IRQ number, see **Table 2-8** on **Page 2-28**. For more information about these functions see the CMSIS documentation **[4]**.

2.8.5 Memory Protection Unit (MPU)

The MPU divides the memory map into a number of regions, and defines the location, size, access permissions, and memory attributes of each region. It supports:

- independent attribute settings for each region
- overlapping regions
- export of memory attributes to the system

The memory attributes affect the behavior of memory accesses to the region. The Cortex-M4 MPU defines:

- eight separate memory regions, 0-7
- a background region

When memory regions overlap, a memory access is affected by the attributes of the region with the highest number. For example, the attributes for region 7 take precedence over the attributes of any region that overlaps region 7.

The background region has the same memory access attributes as the default memory map, but is accessible from privileged software only.

The Cortex-M4 MPU memory map is unified. This means instruction accesses and data accesses have same region settings.

If a program accesses a memory location that is prohibited by the MPU, the processor generates a MemManage fault. This causes a fault exception, and might cause termination of the process in an OS environment.

In an OS environment, the kernel can update the MPU region setting dynamically based on the process to be executed. Typically, an embedded OS uses the MPU for memory protection.

Configuration of MPU regions is based on memory types, see Memory regions, types and attributes on **Page 2-20**.

Table 2-14 shows the possible MPU region attributes.

Note: The shareability and cache attributes are not relevant to the XMC4[12]00.

Table 2-14 Memory attributes summary

Address	Shareability	Other attributes	Description
Strongly-ordered	-	-	All accesses to Strongly-ordered memory occur in program order. All Strongly-ordered regions are assumed to be shared.
Device	Shared	-	Memory-mapped peripherals that several processors share.
	Non-shared	-	Memory-mapped peripherals that only a single processor uses.
Normal	Shared	Non-cacheable Write-through or Write-back Cacheable	Normal memory that is shared between several processors.
	Non-shared	Non-cacheable Write-through or Write-back Cacheable	Normal memory that only a single processor uses.

2.8.5.1 MPU Access Permission Attributes

This section describes the MPU access permission attributes. The access permission bits, TEX, C, B, S, AP, and XN, of the RASR, control access to the corresponding memory region. If an access is made to an area of memory without the required permissions, then the MPU generates a permission fault. **Table 2-15** shows encodings for the TEX, C, B, and S access permission bits.

Table 2-15 TEX, C, B, and S encoding

TEX	C	B	S	Memory type	Shareability	Other attributes
0b000	0	0	x	Strongly-ordered	Shareable	-
		1	x	Device	Shareable	-
	1	0	0	Normal	Not shareable	Outer and inner write-through. No write allocate.
			1		Shareable	
		1	0	Normal	Not shareable	Outer and inner write-back. No write allocate.
			1		Shareable	
0b001	0	0	0	Normal	Not shareable	Outer and inner noncacheable.
			1		Shareable	
		1	x[1)]	Reserved encoding		-
	1	0	x[1)]	Implementation defined attributes.		
		1	0	Normal	Not shareable	Outer and inner write-back. Write and read allocate.
			1		Shareable	
0b010	0	0	x[1)]	Device	Not shareable	Nonshared Device.
		1	x[1)]	Reserved encoding		-
	1	x	x[1)]	Reserved encoding		-
0b1BB	A	A	0	Normal	Not shareable	Cached memory, BB = outer policy, AA = inner policy. See **Table 2-16** on **Page 2-49** for the encoding of the AA and BB bits.
			1		Shareable	

1) The MPU ignores the value of this bit.

Table 2-16 shows the cache policy for memory attribute encodings with a TEX value is in the range 4-7.

Table 2-16 Cache policy for memory attribute encoding

Encoding, AA or BB	Corresponding cache policy
00	Non-cacheable
01	Write back, write and read allocate
10	Write through, no write allocate
11	Write back, no write allocate

MPU configuration for the XMC4[12]00

The XMC4[12]00 has only a single processor and no caches. However to enable portability it is recommended to program the MPU as follows:

Table 2-17 Memory region attributes for a microcontroller

Memory region	TEX	C	B	S	Memory type and attributes
Internal Flash memory	0b000	1	0	0	Normal memory, Non-shareable, write-through
Internal SRAM memories	0b000	1	0	1	Normal memory, Shareable, write-through
External memories	0b000	1	1	1	Normal memory, Shareable, write-back, write-allocate
Peripherals	0b000	0	1	1	Device memory, Shareable

Table 2-18 shows the AP encodings that define the access permissions for privileged and unprivileged software.

Table 2-18 AP encoding

AP[2:0]	Privileged permissions	Unprivileged permissions	Description
000	No access	No access	All accesses generate a permission fault
001	rw	No access	Access from privileged software only
010	rw	r	Writes by unprivileged software generate a permission fault
011	rw	rw	Full access

Table 2-18 AP encoding (cont'd)

AP[2:0]	Privileged permissions	Unprivileged permissions	Description
100	Unpredictable	Unpredictable	Reserved
101	r	No access	Reads by privileged software only
110	r	r	Read only, by privileged or unprivileged software
111	r	r	Read only, by privileged or unprivileged software

2.8.5.2 MPU Mismatch

When an access violates the MPU permissions, the processor generates a MemManage fault, see Exceptions and interrupts on **Page 2-17**. The MMFSR indicates the cause of the fault. See MemManage Fault Status Register on **Page 2-74** for more information.

2.8.5.3 Updating an MPU Region

To update the attributes for an MPU region, update the MPU_RNR, MPU_RBAR and MPU_RASR registers. You can program each register separately, or use a multiple-word write to program all of these registers. You can use the MPU_RBAR and MPU_RASR aliases to program up to four regions simultaneously using an STM instruction.

Updating an MPU region using separate words

Simple code to configure one region:

```
; R1 = region number
; R2 = size/enable
; R3 = attributes
; R4 = address
LDR R0,=MPU_RNR          ; 0xE000ED98, MPU region number register
STR R1, [R0, #0x0]       ; Region Number
STR R4, [R0, #0x4]       ; Region Base Address
STRH R2, [R0, #0x8]      ; Region Size and Enable
STRH R3, [R0, #0xA]      ; Region Attribute
```

Disable a region before writing new region settings to the MPU if you have previously enabled the region being changed. For example:

```
; R1 = region number
; R2 = size/enable
; R3 = attributes
; R4 = address
```

```
LDR   R0,=MPU_RNR           ;  0xE000ED98,  MPU  region  number
register
STR R1, [R0, #0x0]         ; Region Number
BIC R2, R2, #1             ; Disable
STRH R2, [R0, #0x8]        ; Region Size and Enable
STR R4, [R0, #0x4]         ; Region Base Address
STRH R3, [R0, #0xA]        ; Region Attribute
ORR R2, #1                 ; Enable
STRH R2, [R0, #0x8]        ; Region Size and Enable
```

Software must use memory barrier instructions:

- before MPU setup if there might be outstanding memory transfers, such as buffered writes, that might be affected by the change in MPU settings
- after MPU setup if it includes memory transfers that must use the new MPU settings.

However, memory barrier instructions are not required if the MPU setup process starts by entering an exception handler, or is followed by an exception return, because the exception entry and exception return mechanism cause memory barrier behavior.

Software does not require any memory barrier instructions during MPU setup, because it accesses the MPU through the PPB, which is a Strongly-Ordered memory region.

For example, if you want all of the memory access behavior to take effect immediately after the programming sequence, use a DSB instruction and an ISB instruction. A DSB is required after changing MPU settings, such as at the end of context switch. An ISB is required if the code that programs the MPU region or regions is entered using a branch or call. If the programming sequence is entered using a return from exception, or by taking an exception, then you do not require an ISB.

Updating an MPU region using multi-word writes

You can program directly using multi-word writes, depending on how the information is divided. Consider the following reprogramming:

```
; R1 = region number
; R2 = address
; R3 = size, attributes in one
LDR R0, =MPU_RNR    ; 0xE000ED98, MPU region number register
STR R1, [R0, #0x0]  ; Region Number
STR R2, [R0, #0x4]  ; Region Base Address
STR R3, [R0, #0x8]  ; Region Attribute, Size and Enable
```

Use an STM instruction to optimize this:

```
; R1 = region number
; R2 = address
; R3 = size, attributes in one
LDR R0, =MPU_RNR    ; 0xE000ED98, MPU region number register
```

```
STM R0, {R1-R3}     ; Region Number, address, attribute, size and
enable
```

You can do this in two words for pre-packed information. This means that the MPU_RBAR contains the required region number and had the VALID bit set to 1, see MPU Region Base Address Register on **Page 2-96**. Use this when the data is statically packed, for example in a boot loader:

```
; R1 = address and region number in one
; R2 = size and attributes in one
LDR R0, =MPU_RBAR   ; 0xE000ED9C, MPU Region Base register
STR R1, [R0, #0x0]  ; Region base address and
                    ; region number combined with VALID (bit 4)
set to 1
STR R2, [R0, #0x4]  ; Region Attribute, Size and Enable
```

Subregions

Regions of 256 bytes or more are divided into eight equal-sized subregions. Set the corresponding bit in the SRD field of the MPU_RASR to disable a subregion, see MPU Region Attribute and Size Register on **Page 2-98**. The least significant bit of SRD controls the first subregion, and the most significant bit controls the last subregion. Disabling a subregion means another region overlapping the disabled range matches instead. If no other enabled region overlaps the disabled subregion the MPU issues a fault.

Regions of 32, 64, and 128 bytes do not support subregions, With regions of these sizes, you must set the SRD field to 0x00, otherwise the MPU behavior is Unpredictable.

Example of SRD use

Two regions with the same base address overlap. Region one is 128KB, and region two is 512KB. To ensure the attributes from region one apply to the first 128KB region, set the SRD field for region two to 0b00000011 to disable the first two subregions, as the figure shows.

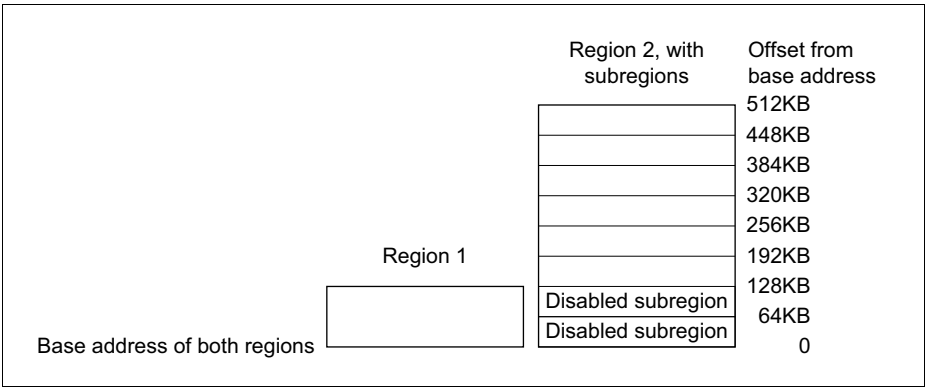


Figure 2-8 Example of SRD use

2.8.5.4 MPU Design Hints and Tips

To avoid unexpected behavior, disable the interrupts before updating the attributes of a region that the interrupt handlers might access.

Ensure software uses aligned accesses of the correct size to access MPU registers:

- except for the MPU_RASR, it must use aligned word accesses
- for the MPU_RASR it can use byte or aligned halfword or word accesses.

The processor does not support unaligned accesses to MPU registers.

When setting up the MPU, and if the MPU has previously been programmed, disable unused regions to prevent any previous region settings from affecting the new MPU setup.

In the XMC4[12]00 the shareability and cache policy attributes do not affect the system behavior. However, using these settings for the MPU regions can make the application code more portable.

2.8.6 Floating Point Unit (FPU)

The Cortex-M4 FPU implements the FPv4-SP floating-point extension.

The FPU fully supports single-precision add, subtract, multiply, divide, multiply and accumulate, and square root operations. It also provides conversions between fixed-point and floating-point data formats, and floating-point constant instructions.

The FPU provides floating-point computation functionality that is compliant with the ANSI/IEEE Std 754-2008, IEEE Standard for Binary Floating-Point Arithmetic, referred to as the IEEE 754 standard **[4]**.

The FPU contains 32 single-precision extension registers, which you can also access as 16 doubleword registers for load, store, and move operations.

2.8.6.1 Enabling the FPU

The FPU is disabled from reset. You must enable it before you can use any floating-point instructions. The Example shows an example code sequence for enabling the FPU in both privileged and user modes. The processor must be in privileged mode to read from and write to the CPACR.

Example: Enabling the FPU

```
; CPACR is located at address 0xE000ED88
LDR.W   R0, =0xE000ED88
; Read CPACR
LDR     R1, [R0]
; Set bits 20-23 to enable CP10 and CP11 coprocessors
ORR     R1, R1, #(0xF << 20)
; Write back the modified value to the CPACR
STR     R1, [R0]; wait for store to complete
DSB
;reset pipeline now the FPU is enabled
ISB
```

2.9 PPB Registers

The CPU private peripherals registers base address is $E000E000_H$.

Table 2-19 Registers Overview

Register Short Name	Register Long Name	Offset Address	Access Mode		Description see
			Read	Write	
SCS					
ACTLR	Auxiliary Control Register	008_H	PV, 32	PV, 32	Page 2-58
CPUID	CPUID Base Register	$D00_H$	PV, 32	PV, 32	Page 2-60
ICSR	Interrupt Control and State Register	$D04_H$	PV, 32	PV, 32	Page 2-60
VTOR	Vector Table Offset Register	$D08_H$	PV, 32	PV, 32	Page 2-63
AIRCR	Application Interrupt and Reset Control Register	$D0C_H$	PV, 32	PV, 32	Page 2-64
SCR	System Control Register	$D10_H$	PV, 32	PV, 32	Page 2-67

Table 2-19 Registers Overview (cont'd)

Register Short Name	Register Long Name	Offset Address	Access Mode		Description see
			Read	Write	
CCR	Configuration and Control Register	$D14_H$	PV, 32	PV, 32	Page 2-68
SHPR1	System Handler Priority Register 1	$D18_H$	PV, 32	PV, 32	Page 2-71
SHPR2	System Handler Priority Register 2	$D1C_H$	PV, 32	PV, 32	Page 2-71
SHPR3	System Handler Priority Register 3	$D20_H$	PV, 32	PV, 32	Page 2-72
SHCRS	System Handler Control and State Register	$D24_H$	PV, 32	PV, 32	Page 2-72
CFSR	Configurable Fault Status Register	$D28_H$	PV, 32	PV, 32	Page 2-74
MMSR[1)]	MemManage Fault Status Register	$D28_H$	PV, 32	PV, 32	Page 2-74
BFSR[1)]	BusFault Status Register	$D29_H$	PV, 32	PV, 32	Page 2-74
UFSR[1)]	UsageFault Status Register	$D2A_H$	PV, 32	PV, 32	Page 2-74
HFSR	HardFault Status Register	$D2C_H$	PV, 32	PV, 32	Page 2-81
MMAR	MemManage Fault Address Register	$D34_H$	PV, 32	PV, 32	Page 2-82
BFAR	BusFault Address Register	$D38_H$	PV, 32	PV, 32	Page 2-83
AFSR	Auxiliary Fault Status Register	$D3C_H$	PV, 32	PV, 32	Page 2-83
SysTick					
SYST_CSR	SysTick Control and Status Register	010_H	PV, 32	PV, 32	Page 2-84
SYST_RVR	SysTick Reload Value Register	014_H	PV, 32	PV, 32	Page 2-85
SYST_CVR	SysTick Current Value Register	018_H	PV, 32	PV, 32	Page 2-86

Table 2-19 Registers Overview (cont'd)

Register Short Name	Register Long Name	Offset Address	Access Mode		Description see
			Read	Write	
SYST_CALIB	SysTick Calibration Value Register	$01C_H$	PV, 32	-	Page 2-86
NVIC					
NVIC_ISER0-NVIC_ISER3	Interrupt Set-enable Registers	100_H	PV, 32	PV, 32	Page 2-87
NVIC_ICER0-NVIC_ICER3	Interrupt Clear-enable Registers	180_H	PV, 32	PV, 32	Page 2-88
NVIC_ISPR0-NVIC_ISPR3	Interrupt Set-pending Registers	200_H	PV, 32	PV, 32	Page 2-89
NVIC_ICPR0-NVIC_ICPR3	Interrupt Clear-pending Registers	280_H	PV, 32	PV, 32	Page 2-89
NVIC_IABR0-NVIC_IABR3	Interrupt Active Bit Registers	300_H	PV, 32	PV, 32	Page 2-90
NVIC_IPR0-NVIC_IPR27	Interrupt Priority Registers	400_H	PV, 32	PV, 32	Page 2-90
STIR	Software Trigger Interrupt Register	$F00_H$	Configurable[2)]		Page 2-92
MPU					
MPU_TYPE	MPU Type Register	$D90_H$	PV, 32	PV, 32	Page 2-93
MPU_CTRL	MPU Control Register	$D94_H$	PV, 32	PV, 32	Page 2-93
MPU_RNR	MPU Region Number Register	$D98_H$	PV, 32	PV, 32	Page 2-96
MPU_RBAR	MPU Region Base Address Register	$D9C_H$	PV, 32	PV, 32	Page 2-96
MPU_RASR	MPU Region Attribute and Size Register	$DA0_H$	PV, 32	PV, 32	Page 2-98
MPU_RBAR_A1	Alias of RBAR, see MPU Region Base Address Register	$DA4_H$	PV, 32	PV, 32	Page 2-96
MPU_RASR_A1	Alias of RASR, see MPU Region Attribute and Size Register	$DA8_H$	PV, 32	PV, 32	Page 2-98

Table 2-19 Registers Overview (cont'd)

Register Short Name	Register Long Name	Offset Address	Access Mode		Description see
			Read	Write	
MPU_RBAR_A2	Alias of RBAR, see MPU Region Base Address Register	DAC_H	PV, 32	PV, 32	Page 2-96
MPU_RASR_A2	Alias of RASR, see MPU Region Attribute and Size Register	$DB0_H$	PV, 32	PV, 32	Page 2-98
MPU_RBAR_A3	Alias of RBAR, see MPU Region Base Address Register	$DB4_H$	PV, 32	PV, 32	Page 2-96
MPU_RASR_A3	Alias of RASR, see MPU Region Attribute and Size Register	$DB8_H$	PV, 32	PV, 32	Page 2-98
FPU					
CPACR	Coprocessor Access Control Register	$D88_H$	PV, 32	PV, 32	Page 2-101
FPCCR	Floating-point Context Control Register	$F34_H$	U, PV, 32	U, PV, 32	Page 2-102
FPCAR	Floating-point Context Address Register	$F38_H$	U, PV, 32	U, PV, 32	Page 2-104
FPSCR	Floating-point Status Control Register	-	U, PV, 32	U, PV, 32	Page 2-105
FPDSCR	Floating-point Default Status Control Register	$F3C_H$	U, PV, 32	U, PV, 32	Page 2-107
MVFR0	Media and FP Feature Register 0	$F40_H$	U, PV, 32	U, PV, 32	Page 2-107
MVFR1	Media and FP Feature Register 1	$F44_H$	U, PV, 32	U, PV, 32	Page 2-109
Reserved	Unused address space	All gaps	nBE	nBE	

1) A subregister of the CFSR.

2) See the register description for more information.

2.9.1 SCS Registers

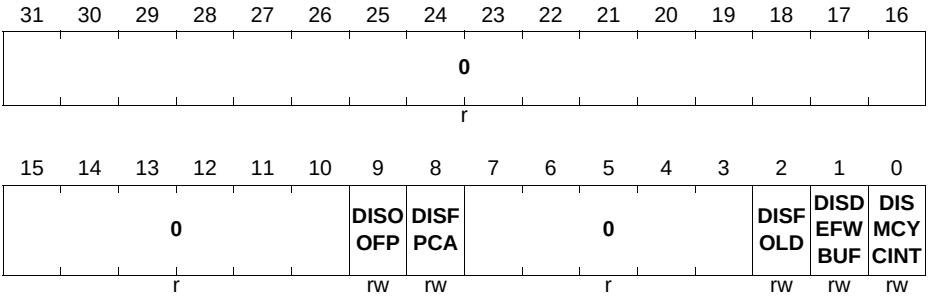
Auxiliary Control Register

The ACTLR provides disable bits for the following processor functions:

- IT folding
- write buffer use for accesses to the default memory map
- interruption of multi-cycle instructions.

By default this register is set to provide optimum performance from the Cortex-M4 processor, and does not normally require modification.

ACTLR
Auxiliary Control Register **(E000 E008$_H$)** **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
DISMCYCINT	0	rw	**Disable load/store multiple** When set to 1, disables interruption of load multiple and store multiple instructions. This increases the interrupt latency of the processor because any LDM or STM must complete before the processor can stack the current state and enter the interrupt handler.

Field	Bits	Type	Description
DISDEFWBUF	1	rw	**Disable write buffer** When set to 1, disables write buffer use during default memory map accesses. This causes all BusFaults to be precise BusFaults but decreases performance because any store to memory must complete before the processor can execute the next instruction. *Note: This bit only affects write buffers implemented in the Cortex-M4 processor.*
DISFOLD	2	rw	**Disable IT folding** When set to 1, disables IT folding.
DISFPCA	8	rw	**Disable FPCA update** Disable automatic update of CONTROL.FPCA.
DISOOFP	9	rw	**Disable out of order FP execution** Disables floating point instructions completing out of order with respect to integer instructions.
0	[31:10], [7:3]	r	**Reserved** Read as 0; should be written with 0.

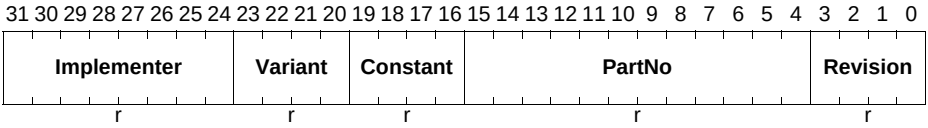
About IT folding

In some situations, the processor can start executing the first instruction in an IT block while it is still executing the IT instruction. This behavior is called IT folding, and improves performance, However, IT folding can cause jitter in looping. If a task must avoid jitter, set the DISFOLD bit to 1 before executing the task, to disable IT folding.

CPUID Base Register

The CPUID register contains the processor part number, version, and implementation information.

CPUID
CPUID Base Register **(E000 ED00$_H$)** **Reset Value: 410F C241$_H$**



Field	Bits	Type	Description
Revision	[3:0]	r	**Revision number** the y value in the “rxpy” product revision identifier 1_H Patch 1
PartNo	[15:4]	r	**Part number of the processor** $C24_H$ Cortex-M4
Constant	[19:16]	r	**Reads as 0xF**
Variant	[23:20]	r	**Variant number** the x value in the “rxpy” product revision identifier 0_H Revision 0
Implementer	[31:24]	r	**Implementer code** 41_H ARM

Interrupt Control and State Register

The ICSR:

- provides:
  - a set-pending bit for the Non-Maskable Interrupt (NMI) exception
  - set-pending and clear-pending bits for the PendSV and SysTick exceptions
- indicates:
  - the exception number of the exception being processed
  - whether there are preempted active exceptions
  - the exception number of the highest priority pending exception
  - whether any interrupts are pending.

ICSR
Interrupt Control and State Register

(E000 ED04$_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
NMIPENDSET	0		PENDSVSET	PENDSVCLR	PENDSTSET	PENDSTCLR	0	Res	ISRPENDING	0				VECTPENDING	
rw	r		rw	w	rw	w	r	r	r	r				r	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
VECTPENDING				RETTOBASE	0		VECTACTIVE								
r				r	r		r								

Field	Bits	Type	Description
VECTACTIVE[1)]	[8:0]	r	**Active exception number** 00_H Thread mode Nonzero = The exception number of the currently active exception. *Note: Subtract 16 from this value to obtain the CMSIS IRQ number required to index into the Interrupt Clear-Enable, Set-Enable, Clear-Pending, Set-Pending, or Priority Registers.*
RETTOBASE	11	r	**Return to Base** Indicates whether there are preempted active exceptions: 0_B there are preempted active exceptions to execute 1_B there are no active exceptions, or the currently-executing exception is the only active exception.

Field	Bits	Type	Description
VECTPENDING	[17:12]	r	**Vector Pending** Indicates the exception number of the highest priority pending enabled exception: 0_H no pending exceptions Nonzero = the exception number of the highest priority pending enabled exception. The value indicated by this field includes the effect of the BASEPRI and FAULTMASK registers, but not any effect of the PRIMASK register.
ISRPENDING	22	r	**Interrupt pending flag** excluding NMI and Faults: 0_B interrupt not pending 1_B interrupt pending.
Res	23	r	**Reserved** This bit is reserved for Debug use and reads-as-zero when the processor is not in Debug.
PENDSTCLR	25	w	**SysTick exception clear-pending bit** 0_B no effect 1_B removes the pending state from the SysTick exception. *Note: This bit is w. On a register read its value is Unknown.*
PENDSTSET	26	rw	**SysTick exception set-pending bit** 0_B Write: no effect Read: SysTick exception is not pending 1_B Write: changes SysTick exception state to pending Read: SysTick exception is pending
PENDSVCLR	27	w	**PendSV clear-pending bit** 0_B no effect 1_B removes the pending state from the PendSV exception. *Note: This bit is w. On a register read its value is Unknown.*

Field	Bits	Type	Description
PENDSVSET	28	rw	**PendSV set-pending bit** 0_B Write: no effect Read: PendSV exception is not pending 1_B Write: changes PendSV exception state to pending Read: PendSV exception is pending Writing 1 to this bit is the only way to set the PendSV exception state to pending.
NMIPENDSET	31	rw	**NMI set-pending bit** 0_B Write: no effect Read: NMI exception is not pending 1_B Write: changes NMI exception state to pending Read: NMI exception is pending Because NMI is the highest-priority exception, normally the processor enter the NMI exception handler as soon as it registers a write of 1 to this bit, and entering the handler clears this bit to 0. A read of this bit by the NMI exception handler returns 1 only if the NMI signal is reasserted while the processor is executing that handler.
0	[30:29], 24, [21:18], [10:9]	r	**Reserved** Read as 0; should be written with 0.

1) This is the same value as IPSR bits[8:0], see Interrupt Program Status Register on page 2-6.

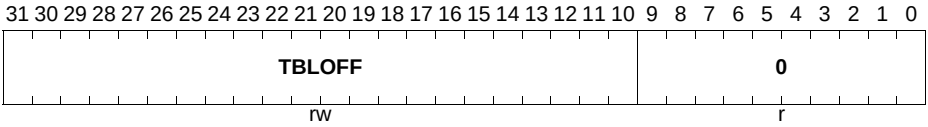
When you write to the ICSR, the effect is Unpredictable if you:

Note: write 1 to the PENDSVSET bit and write 1 to the PENDSVCLR bit
write 1 to the PENDSTSET bit and write 1 to the PENDSTCLR bit.

Vector Table Offset Register

The VTOR indicates the offset of the vector table base address from memory address 0x00000000.

VTOR
Vector Table Offset Register (E000 ED08$_H$) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
TBLOFF	[31:10]	rw	**Vector table base offset field** It contains bits[29:10] of the offset of the table base from the bottom of the memory map. *Note: Bit[29] determines whether the vector table is in the code or SRAM memory region:* *0 = code* *1 = SRAM* *Bit[29] is sometimes called the TBLBASE bit.*
0	[9:0]	r	**Reserved** Read as 0; should be written with 0.

When setting TBLOFF, you must align the offset to the number of exception entries in the vector table. The XMC4[12]00 provides 112 interrupt nodes - minimum alignment is therefore 256 words, enough for up to 128 interrupts.

Notes

1. *XMC4[12]00 implements 112 interrupts, the remaining nodes to 128 are not used.*
2. *Table alignment requirements mean that bits[9:0] of the table offset must always be zero.*

Application Interrupt and Reset Control Register

The AIRCR provides priority grouping control for the exception model, endian status for data accesses, and reset control of the system.

To write to this register, you must write 0x5FA to the VECTKEY field, otherwise the processor ignores the write.

AIRCR
Application Interrupt and Reset Control Register
(E000 ED0C$_H$) **Reset Value: FA05 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
VECTKEY															
rw															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ENDIANNESS	0				PRIGROUP			0					SYSRESETREQ	VECTCLRACTIVE	VECTRESET
r	r				rw			r					w	w	w

Field	Bits	Type	Description
VECTRESET	0	w	**Reserved for Debug use.** This bit reads as 0. When writing to the register you must write 0 to this bit, otherwise behavior is Unpredictable
VECTCLRACTIVE	1	w	**Reserved for Debug use.** This bit reads as 0. When writing to the register you must write 0 to this bit, otherwise behavior is Unpredictable.
SYSRESETREQ	2	w	**System reset request** 0_B no system reset request 1_B asserts a signal to the outer system that requests a reset. This is intended to force a large system reset of all major components except for debug. This bit reads as 0.
PRIGROUP	[10:8]	rw	**Interrupt priority grouping field** This field determines the split of group priority from subpriority, see Binary point on **Page 2-66**.
ENDIANNESS	15	r	**Data endianness bit** 0_B Little-endian 1_B Big-endian.

Field	Bits	Type	Description
VECTKEY	[31:16]	rw	**Register key** Read: = VECTKEY, reads as 0xFA05 Write: = VECTKEYSTAT, On writes, write 0x5FA to VECTKEY, otherwise the write is ignored.
0	[14:11], [7:3]	r	**Reserved** Read as 0; should be written with 0.

Binary point

The PRIGROUP field indicates the position of the binary point that splits the PRI_n fields in the Interrupt Priority Registers into separate group priority and subpriority fields. **Table 2-20** shows how the PRIGROUP value controls this split.

Table 2-20 Priority grouping

Interrupt priority level value, PRI_N[7:0]				Number of	
PRIGROUP	**Binary point[1)]**	**Group priority bits**	**Subpriority bits**	**Group priorities**	**Sub-priorities**
0b000	bxxxxxx0.0	[7:2]	None	64	1
0b001	bxxxxxx.00	[7:2]	None	64	1
0b010	bxxxxx.y00	[7:3]	[2]	32	2
0b011	bxxxx.yy00	[7:4]	[3:2]	16	4
0b100	bxxx.yyy00	[7:5]	[4:2]	8	8
0b101	bxx.yyyy00	[7:6]	[5:2]	4	16
0b110	bx.yyyyy00	7	[6:2]	2	32
0b111	b.yyyyyy00	None	[7:2]	1	64

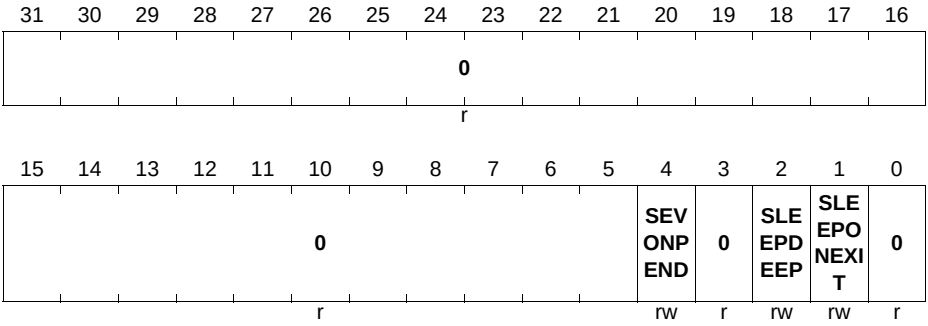
1) PRI_n[7:0] field showing the binary point. x denotes a group priority field bit, and y denotes a subpriority field bit.

*Note: Determining preemption of an exception uses only the group priority field, see Interrupt Priority Grouping on **Page 2-31**.*

System Control Register

The SCR controls features of entry to and exit from low power state.

SCR
System Control Register (E000 ED10$_H$) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
SLEEPONEXIT	1	rw	**Sleep on Exit** Indicates sleep-on-exit when returning from Handler mode to Thread mode: 0_B do not sleep when returning to Thread mode. 1_B enter sleep, or deep sleep, on return from an ISR. Setting this bit to 1 enables an interrupt driven application to avoid returning to an empty main application.
SLEEPDEEP	2	rw	**Sleep or Deep Sleep** Controls whether the processor uses sleep or deep sleep as its low power mode: 0_B sleep 1_B deep sleep

Field	Bits	Type	Description
SEVONPEND	4	rw	**Send Event on Pending bit:** 0_B only enabled interrupts or events can wakeup the processor, disabled interrupts are excluded 1_B enabled events and all interrupts, including disabled interrupts, can wakeup the processor. When an event or interrupt enters pending state, the event signal wakes up the processor from WFE. If the processor is not waiting for an event, the event is registered and affects the next WFE. The processor also wakes up on execution of an SEV instruction or an external event.
0	[31:5], 3, 0	r	**Reserved** Read as 0; should be written with 0.

Configuration and Control Register

The CCR controls entry to Thread mode and enables:

- the handlers for NMI, hard fault and faults escalated by FAULTMASK to ignore BusFaults
- trapping of divide by zero and unaligned accesses
- access to the STIR by unprivileged software, see **Software Trigger Interrupt Register** on **Page 2-92**

CCR
Configuration and Control Register

(E000 ED14$_H$) **Reset Value: 0000 0200$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0						STKALIGN	BFHFNMIGN	0			DIV_0_TRP	UNALIGN_TRP	0	USERSETMPEND	NONBASETHRDENA
r						rw	rw	r			rw	rw	r	rw	rw

Field	Bits	Type	Description
NONBASETHRDENA	0	rw	**Non Base Thread Mode Enable** Indicates how the processor enters Thread mode: 0_B processor can enter Thread mode only when no exception is active. 1_B processor can enter Thread mode from any level under the control of an EXC_RETURN value, see **Exception return**.
USERSETMPEND	1	rw	**User Set Pending Enable** Enables unprivileged software access to the STIR, see **Software Trigger Interrupt Register**. 0_B disable 1_B enable
UNALIGN_TRP	3	rw	**Unaligned Access Trap Enable** Enables unaligned access traps: 0_B do not trap unaligned halfword and word accesses 1_B trap unaligned halfword and word accesses. If this bit is set to 1, an unaligned access generates a UsageFault. Unaligned LDM, STM, LDRD, and STRD instructions always fault irrespective of whether UNALIGN_TRP is set to 1.
DIV_0_TRP	4	rw	**Divide by Zero Trap Enable** Enables faulting or halting when the processor executes an SDIV or UDIV instruction with a divisor of 0: 0_B do not trap divide by 0 1_B trap divide by 0. When this bit is set to 0,a divide by zero returns a quotient of 0.

Field	Bits	Type	Description
BFHFNMIGN	8	rw	**Bus Fault Hard Fault and NMI Ignore** Enables handlers with priority -1 or -2 to ignore data BusFaults caused by load and store instructions. This applies to the hard fault, NMI, and FAULTMASK escalated handlers: 0_B data bus faults caused by load and store instructions cause a lock-up 1_B handlers running at priority -1 and -2 ignore data bus faults caused by load and store instructions. Set this bit to 1 only when the handler and its data are in absolutely safe memory. The normal use of this bit is to probe system devices and bridges to detect control path problems and fix them.
STKALIGN	9	rw	**Stack Alignment** Indicates stack alignment on exception entry: 0_B 4-byte aligned 1_B 8-byte aligned. On exception entry, the processor uses bit[9] of the stacked PSR to indicate the stack alignment. On return from the exception it uses this stacked bit to restore the correct stack alignment.
0	[31:10], [7:5], 2	r	**Reserved** Read as 0; should be written with 0.

System Handler Priority Registers

The SHPR1-SHPR3 registers set the priority level, 0 to 63 of the exception handlers that have configurable priority.

SHPR1-SHPR3 are byte accessible.

The system fault handlers and the priority field and register for each handler are:

Table 2-21 System fault handler priority fields

Handler	Field	Register description
MemManage	PRI_4	System Handler Priority Register 1 on **Page 2-71**
BusFault	PRI_5	
UsageFault	PRI_6	
SVCall	PRI_11	System Handler Priority Register 2 on **Page 2-71**

Table 2-21 System fault handler priority fields (cont'd)

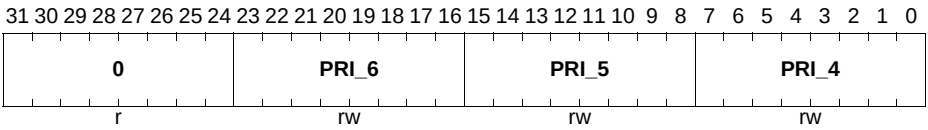
Handler	Field	Register description
PendSV	PRI_14	System Handler Priority Register 3 on **Page 2-72**
SysTick	PRI_15	

Each PRI_N field is 8 bits wide, but the XMC4[12]00 implements only bits[7:2] of each field, and bits[1:0] read as zero and ignore writes.

System Handler Priority Register 1

SHPR1
System Handler Priority Register 1

(E000 ED18$_H$) **Reset Value: 0000 0000$_H$**

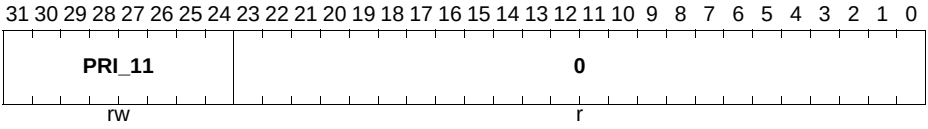


Field	Bits	Type	Description
PRI_4	[7:0]	rw	**Priority of system handler 4, MemManage**
PRI_5	[15:8]	rw	**Priority of system handler 5, BusFault**
PRI_6	[23:16]	rw	**Priority of system handler 6, UsageFault**
0	[31:24]	r	**Reserved** Read as 0; should be written with 0.

System Handler Priority Register 2

SHPR2
System Handler Priority Register 2

(E000 ED1C$_H$) **Reset Value: 0000 0000$_H$**

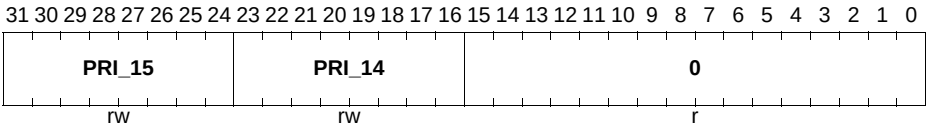


Field	Bits	Type	Description
PRI_11	[31:24]	rw	**Priority of system handler 11, SVCall**
0	[23:0]	r	**Reserved** Read as 0; should be written with 0.

System Handler Priority Register 3

SHPR3
System Handler Priority Register 3

(E000 ED20$_H$) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
PRI_14	[23:16]	rw	**Priority of system handler 14** PendSV
PRI_15	[31:24]	rw	**Priority of system handler 15** SysTick exception
0	[15:0]	r	**Reserved** Read as 0; should be written with 0.

System Handler Control and State Register

The SHCSR enables the system handlers, and indicates:

- the pending status of the BusFault, MemManage fault, and SVC exceptions
- the active status of the system handlers.

SHCSR
System Handler Control and State Register

(E000 ED24$_H$) Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0													USGFAULTENA	BUSFAULTENA	MEMFAULTENA
r													rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SVCALLPENDED	BUSFAULTPENDED	MEMFAULTPENDED	USGFAULTPENDED	SYSTICKACT	PENDSVACT	0	MONITORACT	SVCALLACT	0			USGFAULTACT	0	BUSFAULTACT	MEMFAULTACT
rw	rw	rw	rw	rw	rw	r	rw	rw	r			rw	r	rw	rw

Field	Bits	Type	Description
MEMFAULTACT	0	rw	**MemManage exception active bit** Reads as 1 if exception is active.
BUSFAULTACT	1	rw	**BusFault exception active bit** Reads as 1 if exception is active.
USGFAULTACT	3	rw	**UsageFault exception active bit** Reads as 1 if exception is active.
SVCALLACT	7	rw	**SVCall active bit** Reads as 1 if SVC call is active.
MONITORACT	8	rw	**Debug monitor active bit** Reads as 1 if Debug monitor is active.
PENDSVACT	10	rw	**PendSV exception active bit** Reads as 1 if exception is active.
SYSTICKACT	11	rw	**SysTick exception active bit** Reads as 1 if exception is active.
USGFAULTPENDED	12	rw	**UsageFault exception pending bit** Reads as 1 if exception is pending.
MEMFAULTPENDED	13	rw	**MemManage exception pending bit** Reads as 1 if exception is pending.

Field	Bits	Type	Description
BUSFAULTPENDED	14	rw	**BusFault exception pending bit** Reads as 1 if exception is pending.
SVCALLPENDED	15	rw	**SVCall pending bit** Reads as 1 if exception is pending.
MEMFAULTENA	16	rw	**MemManage enable bit** Set to 1 to enable.
BUSFAULTENA	17	rw	**BusFault enable bit** Set to 1 to enable.
USGFAULTENA	18	rw	**UsageFault enable bit** Set to 1 to enable.
0	[31:19], 9, [6:4], 2	r	**Reserved** Read as 0; should be written with 0.

Notes

1. *Active bits, read as 1 if the exception is active, or as 0 if it is not active. You can write to these bits to change the active status of the exceptions, but see the Caution in this section.*
2. *Pending bits, read as 1 if the exception is pending, or as 0 if it is not pending. You can write to these bits to change the pending status of the exceptions.*
3. *Enable bits, set to 1 to enable the exception, or set to 0 to disable the exception.*

If you disable a system handler and the corresponding fault occurs, the processor treats the fault as a hard fault.

You can write to this register to change the pending or active status of system exceptions. An OS kernel can write to the active bits to perform a context switch that changes the current exception type.

Note: Software that changes the value of an active bit in this register without correct adjustment to the stacked content can cause the processor to generate a fault exception. Ensure software that writes to this register retains and subsequently restores the current active status.

Note: After you have enabled the system handlers, if you have to change the value of a bit in this register you must use a read-modify-write procedure to ensure that you change only the required bit.

Configurable Fault Status Register

The CFSR indicates the cause of a MemManage fault, BusFault, or UsageFault.

The flags in the MMFSR indicate the cause of memory access faults.

The flags in the BFSR indicate the cause of a bus access fault.

The UFSR indicates the cause of a UsageFault.

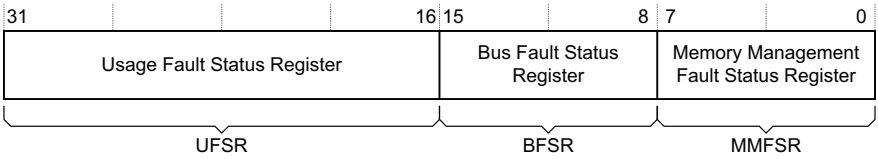


Figure 2-9 CFSR

The CFSR is byte accessible. You can access the CFSR or its subregisters as follows:

- access the complete CFSR with a word access to 0xE000ED28
- access the MMFSR with a byte access to 0xE000ED28
- access the MMFSR with a byte access to 0xE000ED28
- access the MMFSR and BFSR with a halfword access to 0xE000ED28
- access the BFSR with a byte access to 0xE000ED29
- access the UFSR with a halfword access to 0xE000ED2A

Note: The UFSR bits are sticky. This means as one or more fault occurs, the associated bits are set to 1. A bit that is set to 1 is cleared to 0 only by writing 1 to that bit, or by a reset.

CFSR
Configurable Fault Status Register

(E000 ED28$_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0						DIVBYZERO	UNALIGNED	0				NOCP	INVPC	INVSTATE	UNDEFINSTR
r						rw	rw	r				rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BFARVALID	0	LSPERR	STKERR	UNSTKERR	IMPRECISERR	PRECISERR	IBUSERR	MMARVALID	0	MLSPERR	MSTKERR	MUNSTKERR	0	DACCVIOL	IACCVIOL
rw	r	rw	rw	rw	rw	rw	rw	rw	r	rw	rw	rw	r	rw	rw

Field	Bits	Type	Description
IACCVIOL	0	rw	**Instruction access violation flag** 0_B no instruction access violation fault 1_B the processor attempted an instruction fetch from a location that does not permit execution. This fault occurs on any access to an XN region, even when the MPU is disabled or not present. When this bit is 1, the PC value stacked for the exception return points to the faulting instruction. The processor has not written a fault address to the MMAR.
DACCVIOL	1	rw	**Data access violation flag** 0_B no data access violation fault 1_B the processor attempted a load or store at a location that does not permit the operation. When this bit is 1, the PC value stacked for the exception return points to the faulting instruction. The processor has loaded the MMAR with the address of the attempted access.
MUNSTKERR	3	rw	**MemManage fault on unstacking for a return from exception** 0_B no unstacking fault 1_B unstack for an exception return has caused one or more access violations. This fault is chained to the handler. This means that when this bit is 1, the original return stack is still present. The processor has not adjusted the SP from the failing return, and has not performed a new save. The processor has not written a fault address to the MMAR.
MSTKERR	4	rw	**MemManage fault on stacking for exception entry** 0_B no stacking fault 1_B stacking for an exception entry has caused one or more access violations. When this bit is 1, the SP is still adjusted but the values in the context area on the stack might be incorrect. The processor has not written a fault address to the MMAR.

Field	Bits	Type	Description
MLSPERR	5	rw	**MemManage fault during floating point lazy state preservation** 0_B No MemManage fault occurred during floating-point lazy state preservation 1_B A MemManage fault occurred during floating-point lazy state preservation
MMARVALID	7	rw	**MemManage Fault Address Register (MMFAR) valid flag** 0_B value in MMAR is not a valid fault address 1_B MMAR holds a valid fault address. If a MemManage fault occurs and is escalated to a HardFault because of priority, the HardFault handler must set this bit to 0. This prevents problems on return to a stacked active MemManage fault handler whose MMAR value has been overwritten.
IBUSERR	8	rw	**Instruction bus error** 0_B no instruction bus error 1_B instruction bus error. The processor detects the instruction bus error on prefetching an instruction, but it sets the IBUSERR flag to 1 only if it attempts to issue the faulting instruction. When the processor sets this bit is 1, it does not write a fault address to the BFAR.
PRECISERR	9	rw	**Precise data bus error** 0_B no precise data bus error 1_B a data bus error has occurred, and the PC value stacked for the exception return points to the instruction that caused the fault. When the processor sets this bit is 1, it writes the faulting address to the BFAR.

Field	Bits	Type	Description
IMPRECISERR	10	rw	**Imprecise data bus error** 0_B no imprecise data bus error 1_B a data bus error has occurred, but the return address in the stack frame is not related to the instruction that caused the error. When the processor sets this bit to 1, it does not write a fault address to the BFAR. This is an asynchronous fault. Therefore, if it is detected when the priority of the current process is higher than the BusFault priority, the BusFault becomes pending and becomes active only when the processor returns from all higher priority processes. If a precise fault occurs before the processor enters the handler for the imprecise BusFault, the handler detects both IMPRECISERR set to 1 and one of the precise fault status bits set to 1.
UNSTKERR	11	rw	**BusFault on unstacking for a return from exception** 0_B no unstacking fault 1_B stacking for an exception entry has caused one or more BusFaults. This fault is chained to the handler. This means that when the processor sets this bit to 1, the original return stack is still present. The processor does not adjust the SP from the failing return, does not performed a new save, and does not write a fault address to the BFAR.
STKERR	12	rw	**BusFault on stacking for exception entry** 0_B no stacking fault 1_B stacking for an exception entry has caused one or more BusFaults. When the processor sets this bit to 1, the SP is still adjusted but the values in the context area on the stack might be incorrect. The processor does not write a fault address to the BFAR.

Field	Bits	Type	Description
LSPERR	13	rw	**BusFault during floating point lazy state preservation** 0_B No bus fault occurred during floating-point lazy state preservation. 1_B A bus fault occurred during floating-point lazy state preservation
BFARVALID	15	rw	**BusFault Address Register (BFAR) valid flag** 0_B value in BFAR is not a valid fault address 1_B BFAR holds a valid fault address. The processor sets this bit to 1 after a BusFault where the address is known. Other faults can set this bit to 0, such as a MemManage fault occurring later. If a BusFault occurs and is escalated to a hard fault because of priority, the hard fault handler must set this bit to 0. This prevents problems if returning to a stacked active BusFault handler whose BFAR value has been overwritten.
UNDEFINSTR	16	rw	**Undefined instruction UsageFault** 0_B no undefined instruction UsageFault 1_B the processor has attempted to execute an undefined instruction. When this bit is set to 1, the PC value stacked for the exception return points to the undefined instruction. An undefined instruction is an instruction that the processor cannot decode.
INVSTATE	17	rw	**Invalid state UsageFault** 0_B no invalid state UsageFault 1_B the processor has attempted to execute an instruction that makes illegal use of the EPSR. When this bit is set to 1, the PC value stacked for the exception return points to the instruction that attempted the illegal use of the EPSR. This bit is not set to 1 if an undefined instruction uses the EPSR.

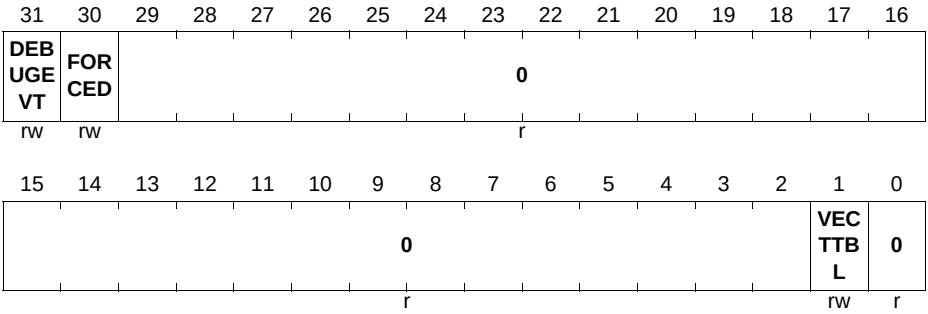
Field	Bits	Type	Description
INVPC	18	rw	**Invalid PC load UsageFault** caused by an invalid PC load by EXC_RETURN: 0_B no invalid PC load UsageFault 1_B the processor has attempted an illegal load of EXC_RETURN to the PC, as a result of an invalid context, or an invalid EXC_RETURN value. When this bit is set to 1, the PC value stacked for the exception return points to the instruction that tried to perform the illegal load of the PC.
NOCP	19	rw	**No coprocessor UsageFault** 0_B no UsageFault caused by attempting to access a coprocessor 1_B the processor has attempted to access a coprocessor.
UNALIGNED	24	rw	**Unaligned access UsageFault** 0_B no unaligned access fault, or unaligned access trapping not enabled 1_B the processor has made an unaligned memory access. Enable trapping of unaligned accesses by setting the UNALIGN_TRP bit in the CCR to 1, see Configuration and Control Register on **Page 2-68**. Unaligned LDM, STM, LDRD, and STRD instructions always fault irrespective of the setting of UNALIGN_TRP.
DIVBYZERO	25	rw	**Divide by zero UsageFault** 0_B no divide by zero fault, or divide by zero trapping not enabled 1_B the processor has executed an SDIV or UDIV instruction with a divisor of 0 When the processor sets this bit to 1, the PC value stacked for the exception return points to the instruction that performed the divide by zero. Enable trapping of divide by zero by setting the DIV_0_TRP bit in the CCR to 1, see Configuration and Control Register on **Page 2-68**.
0	[31:26], [23:20], 14, 6, 2	r	**Reserved** Read as 0; should be written with 0.

HardFault Status Register

The HFSR gives information about events that activate the HardFault handler.

This register is read, write to clear. This means that bits in the register read normally, but writing 1 to any bit clears that bit to 0. The bit assignments are:

HFSR
HardFault Status Register **(E000 ED2C$_H$)** **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
VECTTBL	1	rw	**BusFault on vector table read** Indicates a BusFault on a vector table read during exception processing: 0_B no BusFault on vector table read 1_B BusFault on vector table read This error is always handled by the hard fault handler. When this bit is set to 1, the PC value stacked for the exception return points to the instruction that was preempted by the exception.
FORCED	30	rw	**Forced HardFault** Indicates a forced hard fault, generated by escalation of a fault with configurable priority that cannot be handles, either because of priority or because it is disabled: 0_B no forced HardFault 1_B forced HardFault. When this bit is set to 1, the HardFault handler must read the other fault status registers to find the cause of the fault.

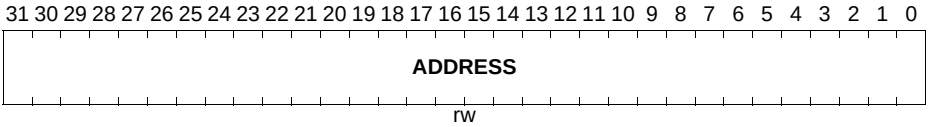
Field	Bits	Type	Description
DEBUGEVT	31	rw	**Reserved for Debug use** When writing to the register you must write 0 to this bit, otherwise behavior is Unpredictable
0	[29:2], 0	r	**Reserved** Read as 0; should be written with 0.

Note: The HFSR bits are sticky. This means as one or more fault occurs, the associated bits are set to 1. A bit that is set to 1 is cleared to 0 only by writing 1 to that bit, or by a reset.

MemManage Fault Address Register

The MMFAR contains the address of the location that generated a MemManage fault.

MMFAR
MemManage Fault Address Register
(E000 ED34$_H$) **Reset Value: XXXX XXXX$_H$**



Field	Bits	Type	Description
ADDRESS	[31:0]	rw	**Address causing the fault** When the MMARVALID bit of the MMFSR is set to 1, this field holds the address of the location that generated the MemManage fault

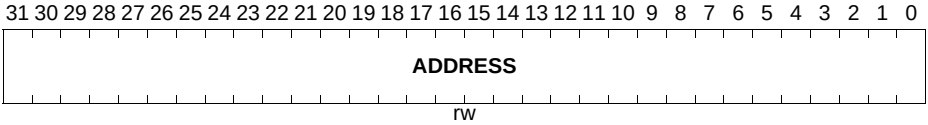
When an unaligned access faults, the address is the actual address that faulted. Because a single read or write instruction can be split into multiple aligned accesses, the fault address can be any address in the range of the requested access size.

Flags in the MMFSR indicate the cause of the fault, and whether the value in the MMFAR is valid. See MemManage Fault Status Register on **Page 2-74**.

BusFault Address Register

The BFAR contains the address of the location that generated a BusFault.

BFAR
BusFault Address Register (E000 ED38$_H$) Reset Value: XXXX XXXX$_H$



Field	Bits	Type	Description
ADDRESS	[31:0]	rw	**Address causing the fault** When the BFARVALID bit of the BFSR is set to 1, this field holds the address of the location that generated the BusFault

When an unaligned access faults the address in the BFAR is the one requested by the instruction, even if it is not the address of the fault.

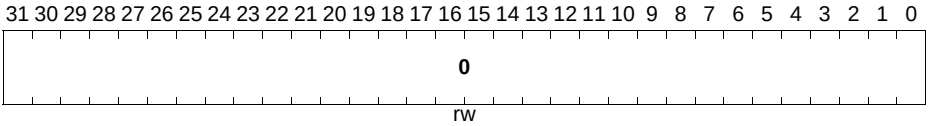
Flags in the BFSR indicate the cause of the fault, and whether the value in the BFAR is valid. See BusFault Status Register on **Page 2-74**.

Auxiliary Fault Status Register

The AFSR contains additional system fault information.

This register is read, write to clear. This means that bits in the register read normally, but writing 1 to any bit clears that bit to 0.

AFSR
Auxiliary Fault Status Register (E000 ED3C$_H$) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
VALUE	[31:0]	rw	**Reserved** Read as 0; should be written with 0.

Each AFSR bit maps directly to an AUXFAULT input of the processor, and a single-cycle HIGH signal on the input sets the corresponding AFSR bit to one. It remains set to 1 until you write 1 to the bit to clear it to zero.

When an AFSR bit is latched as one, an exception does not occur. Use an interrupt if an exception is required.

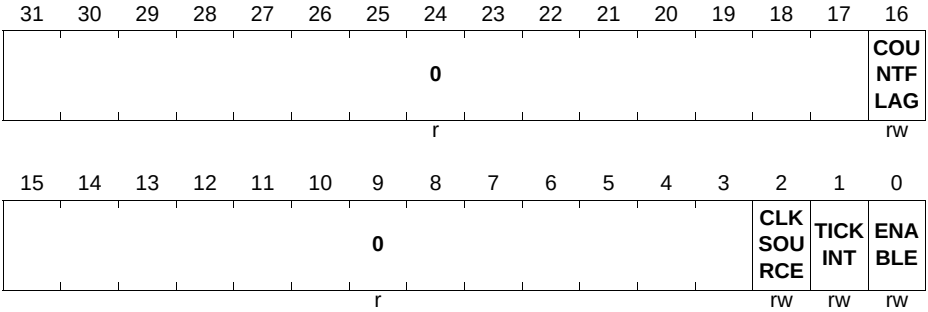
2.9.2 SysTick Registers

SysTick Control and Status Register

The SysTick SYST_CSR register enables the SysTick features.

SYST_CSR
SysTick Control and Status Register

(E000 E010$_H$) **Reset Value: 0000 0004$_H$**



Field	Bits	Type	Description
ENABLE	0	rw	**Enable** Enables the counter: 0_B counter disabled 1_B counter enabled.
TICKINT	1	rw	**Tick Interrupt Enable** Enables SysTick exception request: 0_B counting down to zero does not assert the SysTick exception request 1_B counting down to zero to asserts the SysTick exception request. Software can use COUNTFLAG to determine if SysTick has ever counted to zero.

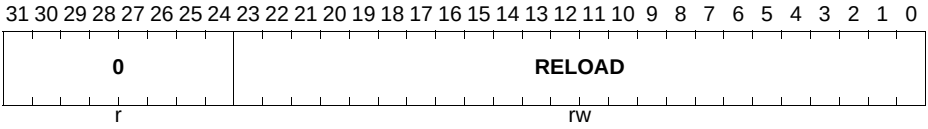
Field	Bits	Type	Description
CLKSOURCE	2	rw	**Clock source** 0_B f_{STDBY} / 2 1_B f_{CPU}
COUNTFLAG	16	rw	**Counter Flag** Returns 1 if timer counted to 0 since last time this was read.
0	[31:17], [15:3]	r	**Reserved** Read as 0; should be written with 0.

When ENABLE is set to 1, the counter loads the RELOAD value from the SYST_RVR register and then counts down. On reaching 0, it sets the COUNTFLAG to 1 and optionally asserts the SysTick depending on the value of TICKINT. It then loads the RELOAD value again, and begins counting.

SysTick Reload Value Register

The SYST_RVR register specifies the start value to load into the SYST_CVR register.

SYST_RVR
SysTick Reload Value Register (E000 E014$_H$) **Reset Value: XXXX XXXX$_H$**



Field	Bits	Type	Description
RELOAD	[23:0]	rw	**Reload Value** Value to load into the SYST_CVR register when the counter is enabled and when it reaches 0, see Calculating the RELOAD value.
0	[31:24]	r	**Reserved** Read as 0; should be written with 0.

Notes on calculating the RELOAD value

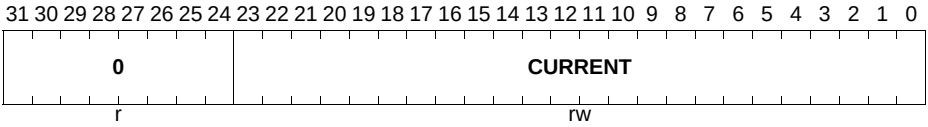
1. *The RELOAD value can be any value in the range 0x00000001-0x00FFFFFF. A start value of 0 is possible, but has no effect because the SysTick exception request and COUNTFLAG are activated when counting from 1 to 0.*

2. *The RELOAD value is calculated according to its use. For example, to generate a multi-shot timer with a period of N processor clock cycles, use a RELOAD value of N-1. If the SysTick interrupt is required every 100 clock pulses, set RELOAD to 99.*

SysTick Current Value Register

The SYST_CVR register contains the current value of the SysTick counter.

SYST_CVR
SysTick Current Value Register (E000 E018$_H$) Reset Value: XXXX XXXX$_H$

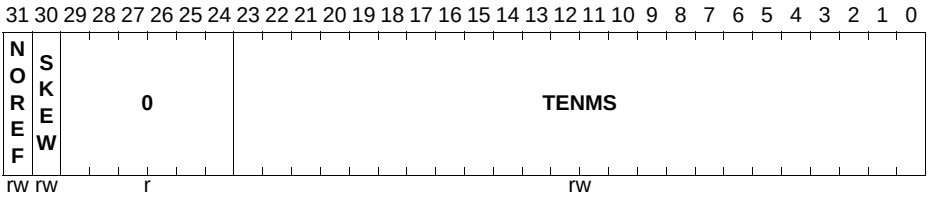


Field	Bits	Type	Description
CURRENT	[23:0]	rw	**Current Value** Reads return the current value of the SysTick counter. A write of any value clears the field to 0, and also clears the SYST_CSR COUNTFLAG bit to 0.
0	[31:24]	r	**Reserved** Read as 0; should be written with 0.

SysTick Calibration Value Register

The SYST_CALIB register indicates the SysTick calibration properties.

SYST_CALIB
SysTick Calibration Value Register
r (E000 E01C$_H$) Reset Value: C000 0000$_H$



Field	Bits	Type	Description
TENMS	[23:0]	rw	**Ten Milliseconds Reload Value** Reload value for 10ms (100Hz) timing, subject to system clock skew errors. If the value reads as zero, the calibration value is not known.
SKEW	30	rw	**Ten Milliseconds Skewed** Indicates whether the TENMS value is exact: 0_B TENMS value is exact 1_B TENMS value is inexact, or not given. An inexact TENMS value can affect the suitability of SysTick as a software real time clock.
NOREF	31	rw	**No Reference Clock** Indicates whether the device provides a reference clock to the processor: 0_B reference clock provided 1_B no reference clock provided. If your device does not provide a reference clock, the SYST_CSR.CLKSOURCE bit reads-as-one and ignores writes.
0	[29:24]	r	**Reserved** Read as 0; should be written with 0.

2.9.3 NVIC Registers

Interrupt Set-enable Registers

The NVIC_ISERx (x=0-3) registers enable interrupts, and show which interrupts are enabled.

NVIC_ISERx (x=0-3)
Interrupt Set-enable Register x

(E000 E100$_H$ + 4*x) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SETENA																															
rw																															

Field	Bits	Type	Description
SETENA	[31:0]	rw	**Interrupt set-enable bits** 0_B Write: no effect Read: interrupt disabled 1_B Write: enable interrupt Read: interrupt enabled

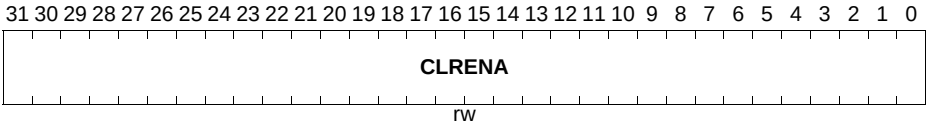
If a pending interrupt is enabled, the NVIC activates the interrupt based on its priority. If an interrupt is not enabled, asserting its interrupt signal changes the interrupt state to pending, but the NVIC never activates the interrupt, regardless of its priority.

Interrupt Clear-enable Registers

The NVIC_ICERx registers disable interrupts, and show which interrupts are enabled.

NVIC_ICERx (x=0-3)
Interrupt Clear-enable Register x

(E000 E180$_H$ + 4*x) **Reset Value: 0000 0000$_H$**



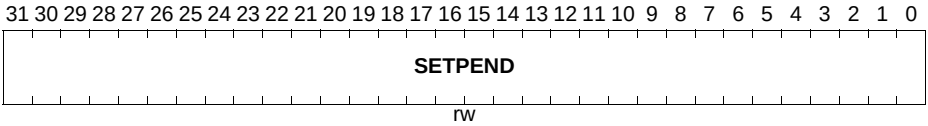
Field	Bits	Type	Description
CLRENA	[31:0]	rw	**Interrupt clear-enable bits.** 0_B Write: no effect Read: interrupt disabled 1_B Write: disable interrupt Read: interrupt enabled

Interrupt Set-pending Registers

The NVIC_ISPRx registers force interrupts into the pending state, and show which interrupts are pending.

NVIC_ISPRx (x=0-3)
Interrupt Set-pending Register x

(E000 E200$_H$ + 4*x) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
SETPEND	[31:0]	rw	**Interrupt set-pending bits.** 0_B Write: no effect Read: interrupt is not pending 1_B Write: changes interrupt state to pending Read: interrupt is pending

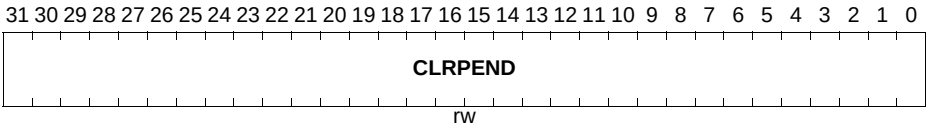
Writing 1 to the ISPR bit corresponding to:
- an interrupt that is pending has no effect
- a disabled interrupt sets the state of that interrupt to pending

Interrupt Clear-pending Registers

The NVIC_ICPRx registers remove the pending state from interrupts, and show which interrupts are pending.

NVIC_ICPRx (x=0-3)
Interrupt Clear-pending Register x

(E000 E280$_H$ + 4*x) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
CLRPEND	[31:0]	rw	**Interrupt set-pending bits.** 0_B Write: no effect Read: interrupt is not pending 1_B Write: removes pending state an interrupt Read: interrupt is pending

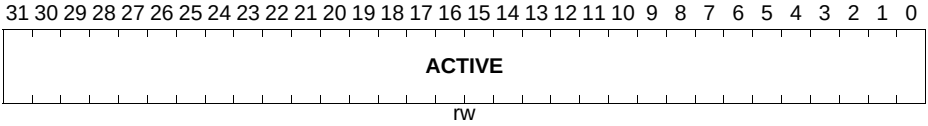
Note: Writing 1 to an ICPR bit does not affect the active state of the corresponding interrupt.

Interrupt Active Bit Registers

The NVIC_IABRx registers indicate which interrupts are active.

NVIC_IABRx (x=0-3)
Interrupt Active Bit Register x

(E000 E300$_H$ + 4*x) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
ACTIVE	[31:0]	rw	**Interrupt active flags:** 0_B interrupt not active 1_B interrupt active

A bit reads as one if the status of the corresponding interrupt is active or active and pending.

Interrupt Priority Registers

The NVIC_IPRx (x=0-27) registers provide an 8-bit priority field for each interrupt. These registers are byte-accessible. Each register holds four priority fields as shown:

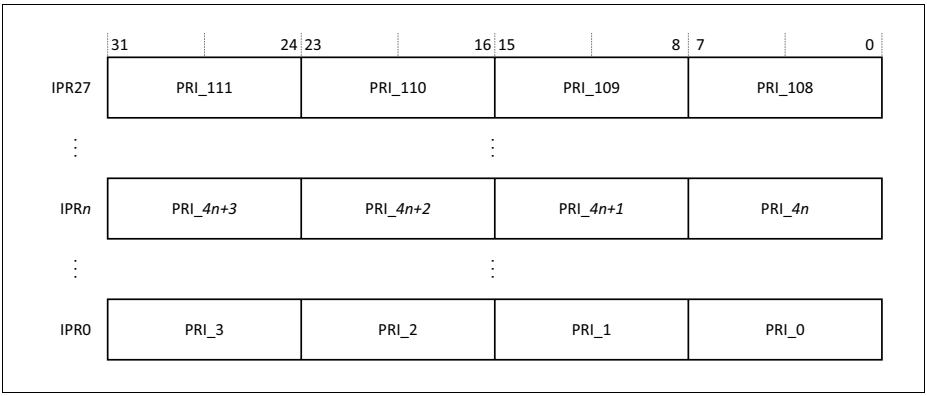


Figure 2-10 Interrupt Priority Register

NVIC_IPRx (x=0-27)
Interrupt Priority Register x

(E000 E400$_H$ + 4*x) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PRI_3								PRI_2								PRI_1								PRI_0							
rw								rw								rw								rw							

Field	Bits	Type	Description
PRI_0	[7:0]	rw	**Priority value 0**
PRI_1	[15:8]	rw	**Priority value 1**
PRI_2	[23:16]	rw	**Priority value 2**
PRI_3	[31:24]	rw	**Priority value 3** The lower the value, the greater the priority of the corresponding interrupt. The processor implements only bits[7:n] of each field, bits[n-1:0] read as zero and ignore writes.

See “Using CMSIS functions to access NVIC” on **Page 2-45** for more information about the access to the interrupt priority array, which provides the software view of the interrupt priorities.

Find the IPR number and byte offset for interrupt m as follows:

- the corresponding IPR number n, see **Figure 2-10** on **Page 2-917**, is given by
  n = m DIV 4
- the byte offset of the required Priority field in this register is m MOD 4, where:
  – byte offset 0 refers to register bits[7:0]
  – byte offset 1 refers to register bits[15:8]
  – byte offset 2 refers to register bits[23:16]
  – byte offset 3 refers to register bits[31:24].

Software Trigger Interrupt Register

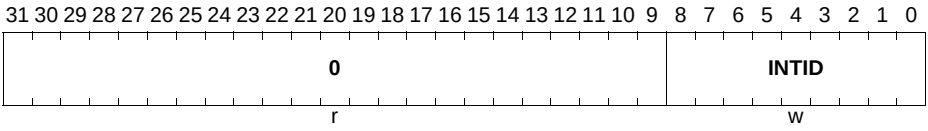
Write to the STIR to generate an interrupt from software.

When the USERSETMPEND bit in the SCR is set to 1, unprivileged software can access the STIR, see System Control Register on **Page 2-67**.

Note: Only privileged software can enable unprivileged access to the STIR.

STIR
Software Trigger Interrupt Register

(E000 EF00$_H$) **Reset Value: 0000 0000$_H$**



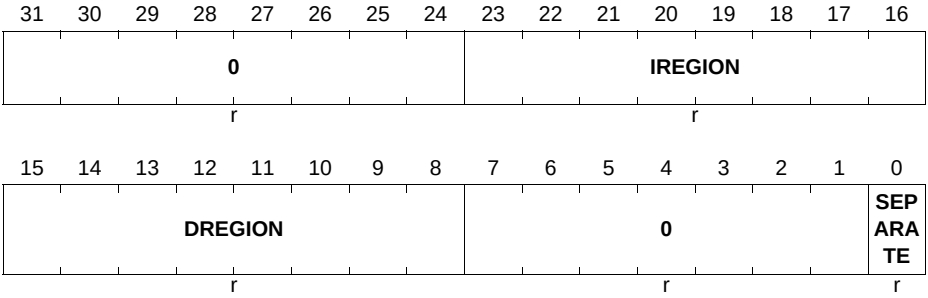
Field	Bits	Type	Description
INTID	[8:0]	w	**Interrupt ID of the interrupt to trigger** in the range 0-111. For example, a value of 0x03 specifies interrupt IRQ3.
0	[31:9]	r	**Reserved** Read as 0; should be written with 0.

2.9.4 MPU Registers

MPU Type Register

The MPU_TYPE register indicates whether the MPU is present, and if so, how many regions it supports.

MPU_TYPE
MPU Type Register **(E000 ED90$_H$)** **Reset Value: 0000 0800$_H$**



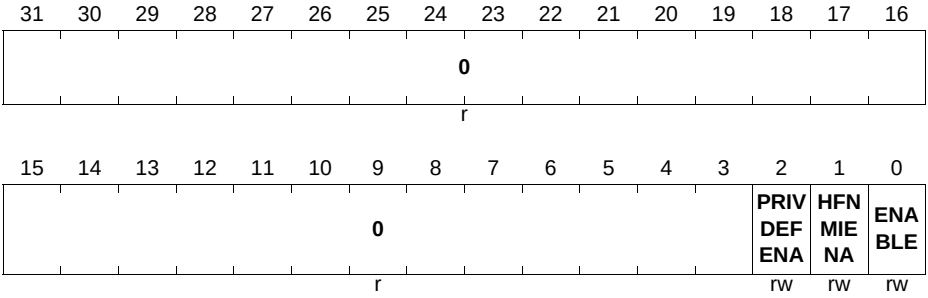
Field	Bits	Type	Description
SEPARATE	0	r	**Support for unified or separate instruction and date memory maps** 0_B unified
DREGION	[15:8]	r	**Number of supported MPU data regions** 08_H Eight MPU regions
IREGION	[23:16]	r	**Number of supported MPU instruction regions** Always contains 0x00. The MPU memory map is unified and is described by the DREGION field.
0	[31:24], [7:1]	r	**Reserved** Read as 0; should be written with 0.

MPU Control Register

The MPU_CTRL register:

- enables the MPU
- enables the default memory map background region
- enables use of the MPU when in the hard fault, Non-maskable Interrupt (NMI), and FAULTMASK escalated handlers.

MPU_CTRL
MPU Control Register **(E000 ED94$_H$)** **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
ENABLE	0	rw	**Enable MPU** 0_B MPU disabled 1_B MPU enabled.
HFNMIENA	1	rw	**Enable the operation of MPU during hard fault, NMI, and FAULTMASK handlers** When the MPU is enabled: 0_B MPU is disabled during hard fault, NMI, and FAULTMASK handlers, regardless of the value of the ENABLE bit 1_B the MPU is enabled during hard fault, NMI, and FAULTMASK handlers. When the MPU is disabled, if this bit is set to 1 the behavior is Unpredictable.

Field	Bits	Type	Description
PRIVDEFENA	2	rw	**Enables privileged software access to the default memory map** 0_B If the MPU is enabled, disables use of the default memory map. Any memory access to a location not covered by any enabled region causes a fault. 1_B If the MPU is enabled, enables use of the default memory map as a background region for privileged software accesses. When enabled, the background region acts as if it is region number -1. Any region that is defined and enabled has priority over this default map. f the MPU is disabled, the processor ignores this bit.
0	[31:3]	r	**Reserved** Read as 0; should be written with 0.

When ENABLE and PRIVDEFENA are both set to 1:

- For privileged accesses, the default memory map is as described in Memory model on **Page 2-20**. Any access by privileged software that does not address an enabled memory region behaves as defined by the default memory map.
- Any access by unprivileged software that does not address an enabled memory region causes a MemManage fault.

XN and Strongly-ordered rules always apply to the System Control Space regardless of the value of the ENABLE bit.

When the ENABLE bit is set to 1, at least one region of the memory map must be enabled for the system to function unless the PRIVDEFENA bit is set to 1. If the PRIVDEFENA bit is set to 1 and no regions are enabled, then only privileged software can operate.

When the ENABLE bit is set to 0, the system uses the default memory map. This has the same memory attributes as if the MPU is not implemented, see **Table 2-6** on **Page 2-22**. The default memory map applies to accesses from both privileged and unprivileged software.

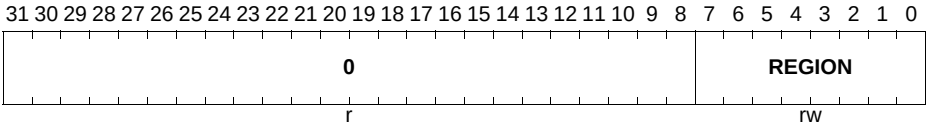
When the MPU is enabled, accesses to the System Control Space and vector table are always permitted. Other areas are accessible based on regions and whether PRIVDEFENA is set to 1.

Unless HFNMIENA is set to 1, the MPU is not enabled when the processor is executing the handler for an exception with priority –1 or –2. These priorities are only possible when handling a hard fault or NMI exception, or when FAULTMASK is enabled. Setting the HFNMIENA bit to 1 enables the MPU when operating with these two priorities.

MPU Region Number Register

The MPU_RNR selects which memory region is referenced by the MPU_RBAR and MPU_RASR registers.

MPU_RNR
MPU Region Number Register (E000 ED98$_H$) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
REGION	[7:0]	rw	**Region** Indicates the MPU region referenced by the MPU_RBAR and MPU_RASR registers. The MPU supports 8 memory regions, so the permitted values of this field are 0-7.
0	[31:8]	r	**Reserved** Read as 0; should be written with 0.

Normally, you write the required region number to this register before accessing the MPU_RBAR or MPU_RASR. However you can change the region number by writing to the MPU RBAR with the VALID bit set to 1, see MPU Region Base Address Register. This write updates the value of the REGION field.

MPU Region Base Address Register

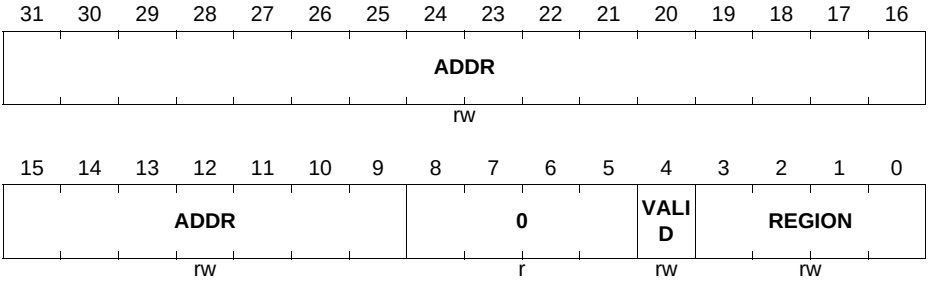
The MPU_RBAR defines the base address of the MPU region selected by the MPU_RNR, and can update the value of the MPU_RNR.

MPU_RBAR
MPU Region Base Address Register
(E000 ED9C$_H$) **Reset Value: 0000 0000$_H$**

MPU_RBAR_A1
MPU Region Base Address Register A1
(E000 EDA4$_H$) **Reset Value: 0000 0000$_H$**

MPU_RBAR_A2
MPU Region Base Address Register A2
(E000 EDAC$_H$) **Reset Value: 0000 0000$_H$**

MPU_RBAR_A3
MPU Region Base Address Register A3
(E000 EDB4$_H$) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
REGION	[3:0]	rw	**MPU region field** For the behavior on writes, see the description of the VALID field. On reads, returns the current region number, as specified by the RNR.

Field	Bits	Type	Description
VALID	4	rw	**MPU Region Number valid bit** Write: 0_B MPU_RNR not changed, and the processor: - updates the base address for the region specified in the MPU_RNR - ignores the value of the REGION field 1_B the processor: - updates the value of the MPU_RNR to the value of the REGION field - updates the base address for the region specified in the REGION field. Always reads as zero.
ADDR	[31:9]	rw	**Region base address field** The value of N (N = 9 for bit definition) depends on the region size. For more information see The ADDR field.
0	[8:5]	r	**Reserved** Read as 0; should be written with 0.

The ADDR field

The ADDR field is bits[31:N] of the MPU_RBAR. The region size, as specified by the SIZE field in the MPU_RASR, defines the value of N:

N = Log2(Region size in bytes),

If the region size is configured to 4GB, in the MPU_RASR, there is no valid ADDR field. In this case, the region occupies the complete memory map, and the base address is 0x00000000.

The base address is aligned to the size of the region. For example, a 64KB region must be aligned on a multiple of 64KB, for example, at 0x00010000 or 0x00020000.

MPU Region Attribute and Size Register

The MPU_RASR defines the region size and memory attributes of the MPU region specified by the MPU_RNR, and enables that region and any subregions.

MPU_RASR is accessible using word or halfword accesses:

- the most significant halfword holds the region attributes
- the least significant halfword holds the region size and the region and subregion enable bits.

MPU_RASR
MPU Region Attribute and Size Register
(E000 EDA0$_H$) **Reset Value: 0000 0000$_H$**

MPU_RASR_A1
MPU Region Attribute and Size Register A1
(E000 EDA8$_H$) **Reset Value: 0000 0000$_H$**

MPU_RASR_A2
MPU Region Attribute and Size Register A2
(E000 EDB0$_H$) **Reset Value: 0000 0000$_H$**

MPU_RASR_A3
MPU Region Attribute and Size Register A3
(E000 EDB8$_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0			XN	0	AP			0		TEX			S	C	B
r			rw	r	rw			r		rw			rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SRD								0		SIZE					ENABLE
rw								r		rw					rw

Field	Bits	Type	Description
ENABLE	0	rw	**Region enable bit.**
SIZE	[5:1]	rw	**MPU protection region size** The minimum permitted value is 3 (0b00010), see See SIZE field values for more information.
SRD	[15:8]	rw	**Subregion disable bits** For each bit in this field: 0_B corresponding sub-region is enabled 1_B corresponding sub-region is disabled See **Subregions**on **Page 2-52** for more information. Region sizes of 128 bytes and less do not support subregions. When writing the attributes for such a region, write the SRD field as 0x00.
B	16	rw	**Memory access attribute** see **Table 2-15** on**Page 2-48**.

Field	Bits	Type	Description
C	17	rw	**Memory access attribute** see **Table 2-15** on**Page 2-48**.
S	18	rw	**Shareable bit** see **Table 2-15** on**Page 2-48**.
TEX	[21:19]	rw	**Memory access attribute** see **Table 2-15** on**Page 2-48**.
AP	[26:24]	rw	**Access permission field** see **Table 2-18** on **Page 2-49**.
XN	28	rw	**Instruction access disable bit** 0_B instruction fetches enabled 1_B instruction fetches disabled.
0	[31:29, 27, [23:22], [7:6]	r	**Reserved** Read as 0; should be written with 0.

For information about access permission, see **MPU Access Permission Attributes** on **Page 2-48**.

SIZE field values

The SIZE field defines the size of the MPU memory region specified by the RNR. as follows:

(Region size in bytes) = 2(SIZE+1)

The smallest permitted region size is 32B, corresponding to a SIZE value of 4. **Table 2-22** gives example SIZE values, with the corresponding region size and value of N in the MPU_RBAR.

Table 2-22 Example SIZE field values

SIZE value	Region size	Value of N[1)]	Note
0b00100 (4)	32B	5	Minimum permitted size
0b01001 (9)	1KB	10	-
0b10011 (19)	1MB	20	-
0b11101 (29)	1GB	30	-
0b11111 (31)	4GB	32	Maximum possible size

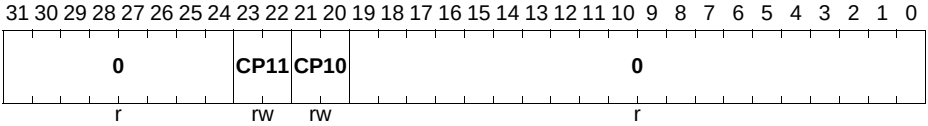
1) In the MPU_RBAR, see MPU Region Base Address Register on **Page 2-96**.

2.9.5 FPU Registers

Coprocessor Access Control Register

The CPACR register specifies the access privileges for coprocessors.

CPACR
Coprocessor Access Control Register
(E000 ED88$_H$) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
CP10	[21:20]	rw	**Access privileges for coprocessor 10** The possible values of each field are: 00_B Access denied. Any attempted access generates a NOCP UsageFault. 01_B Privileged access only. An unprivileged access generates a NOCP fault. 10_B Reserved. The result of any access is Unpredictable. 11_B Full access.
CP11	[23:22]	rw	**Access privileges for coprocessor 11** The possible values of each field are: 00_B Access denied. Any attempted access generates a NOCP UsageFault. 01_B Privileged access only. An unprivileged access generates a NOCP fault. 10_B Reserved. The result of any access is Unpredictable. 11_B Full access.
0	[31:24], [19:0]	r	**Reserved** Read as 0; should be written with 0.

Floating-point Context Control Register

The FPCCR register sets or returns FPU control data.

FPCCR
Floating-point Context Control Register
(E000 EF34$_H$) **Reset Value: C000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ASPEN	LSPEN	0													
rw	rw	r													

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0							MONRDY	0	BFRDY	MMRDY	HFRDY	THREAD	0	USER	LSPACT
r							rw	r	rw	rw	rw	rw	r	rw	rw

Field	Bits	Type	Description
LSPACT	0	rw	**Lazy State Preservation Active** 0_B Lazy state preservation is not active. 1_B Lazy state preservation is active. floating-point stack frame has been allocated but saving state to it has been deferred.
USER	1	rw	**User allocated Stack Frame** 0_B Privilege level was not user when the floating-point stack frame was allocated. 1_B Privilege level was user when the floating-point stack frame was allocated.
THREAD	3	rw	**Thread Mode allocated Stack Frame** 0_B Mode was not Thread Mode when the floating-point stack frame was allocated. 1_B Mode was Thread Mode when the floating-point stack frame was allocated.
HFRDY	4	rw	**HardFault Ready** 0_B Priority did not permit setting the HardFault handler to the pending state when the floating-point stack frame was allocated. 1_B Priority permitted setting the HardFault handler to the pending state when the floating-point stack frame was allocated.

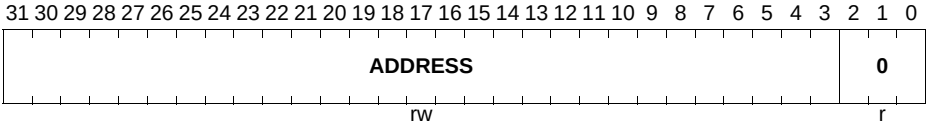
Field	Bits	Type	Description
MMRDY	5	rw	**MemManage Ready** 0_B MemManage is disabled or priority did not permit setting the MemManage handler to the pending state when the floating-point stack frame was allocated. 1_B MemManage is enabled and priority permitted setting the MemManage handler to the pending state when the floating-point stack frame was allocated.
BFRDY	6	rw	**BusFault Ready** 0_B BusFault is disabled or priority did not permit setting the BusFault handler to the pending state when the floating-point stack frame was allocated. 1_B BusFault is enabled and priority permitted setting the BusFault handler to the pending state when the floating-point stack frame was allocated.
MONRDY	8	rw	**Monitor Ready** 0_B Debug Monitor is disabled or priority did not permit setting MON_PEND when the floating-point stack frame was allocated. 1_B Debug Monitor is enabled and priority permits setting MON_PEND when the floating-point stack frame was allocated.
LSPEN	30	rw	**Lazy State Preservation Enabled** 0_B Disable automatic lazy state preservation for floating-point context. 1_B Enable automatic lazy state preservation for floating-point context.
ASPEN	31	rw	**Automatic State Preservation** Enables CONTROL setting on execution of a floating-point instruction. This results in automatic hardware state preservation and restoration, for floating-point context, on exception entry and exit. 0_B Disable CONTROL setting on execution of a floating-point instruction. 1_B Enable CONTROL setting on execution of a floating-point instruction.

Field	Bits	Type	Description
0	[29:9], 7, 2	r	**Reserved** Read as 0; should be written with 0.

Floating-point Context Address Register

The FPCAR register holds the location of the unpopulated floating-point register space allocated on an exception stack frame.

FPCAR
Floating-point Context Address Register
(E000 EF38$_H$) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
ADDRESS	[31:3]	rw	**Address** The location of the unpopulated floating-point register space allocated on an exception stack frame.
0	[2:0]	r	**Reserved** Read as 0; should be written with 0.

Floating-point Status Control Register

The FPSCR register provides all necessary User level control of the floating-point system.

FPSCR
Floating-point Status Control Register

Reset Value: XXXX XXXX$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
N	**Z**	**C**	**V**	**0**	**AHP**	**DN**	**FZ**	**RMode**		**0**					
rw	rw	rw	rw	r	rw	rw	rw	rw		r					

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0								**IDC**	**0**		**IXC**	**UFC**	**OFC**	**DZC**	**IOC**
r								rw	r		rw	rw	rw	rw	rw

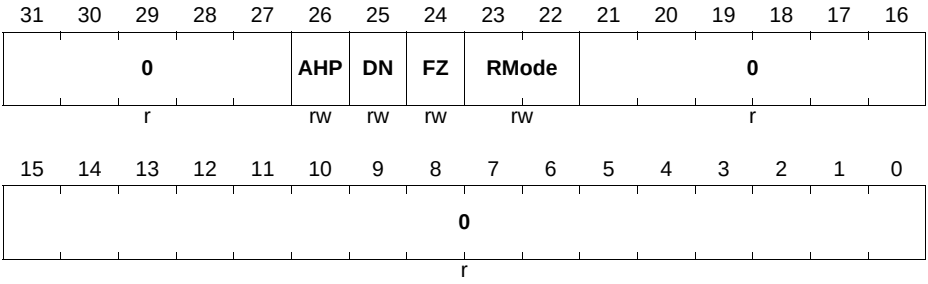
Field	Bits	Type	Description
IOC	0	rw	**Invalid Operation cumulative exception bit** IOC set to 1 indicates that the Invalid Operation cumulative exception has occurred since 0 was last written to IOC.
DZC	1	rw	**Division by Zero cumulative exception bit** DZC set to 1 indicates that the Division by Zero cumulative exception has occurred since 0 was last written to DZC.
OFC	2	rw	**Overflow cumulative exception bit** OFC set to 1 indicates that the Overflow cumulative exception has occurred since 0 was last written to OFC.
UFC	3	rw	**Underflow cumulative exception bit** UFC set to 1 indicates that the Underflow cumulative exception has occurred since 0 was last written to UFC.
IXC	4	rw	**Inexact cumulative exception bit** IXC set to 1 indicates that the Inexact cumulative exception has occurred since 0 was last written to IXC.
IDC	7	rw	**Input Denormal cumulative exception bit** see bits [4:0].

Field	Bits	Type	Description
RMode	[23:22]	rw	**Rounding Mode control field** 00_B Round to Nearest (RN) mode 01_B Round towards Plus Infinity (RP) mode 10_B Round towards Minus Infinity (RM) mode 11_B Round towards Zero (RZ) mode. The specified rounding mode is used by almost all floating-point instructions.
FZ	24	rw	**Flush-to-zero mode control bit** 0_B Flush-to-zero mode disabled. Behavior of the floating-point system is fully compliant with the IEEE 754 standard. 1_B Flush-to-zero mode enabled.
DN	25	rw	**Default NaN mode control bit** 0_B NaN operands propagate through to the output of a floating-point operation. 1_B Any operation involving one or more NaNs returns the Default NaN.
AHP	26	rw	**Alternative half-precision control bit** 0_B IEEE half-precision format selected. 1_B Alternative half-precision format selected.
V	28	rw	**Overflow condition code flag** Floating-point comparison operations update this flag.
C	29	rw	**Carry condition code flag** Floating-point comparison operations update this flag.
Z	30	rw	**Zero condition code flag** Floating-point comparison operations update this flag.
N	31	rw	**Negative condition code flag** Floating-point comparison operations update this flag.
0	27, [21:8], [6:5]	r	**Reserved** Read as 0; should be written with 0.

Floating-point Default Status Control Register

The FPDSCR register holds the default values for the floating-point status control data.

FPDSCR
Floating-point Default Status Control Register
(E000 EF3C$_H$) **Reset Value: 0000 0000$_H$**



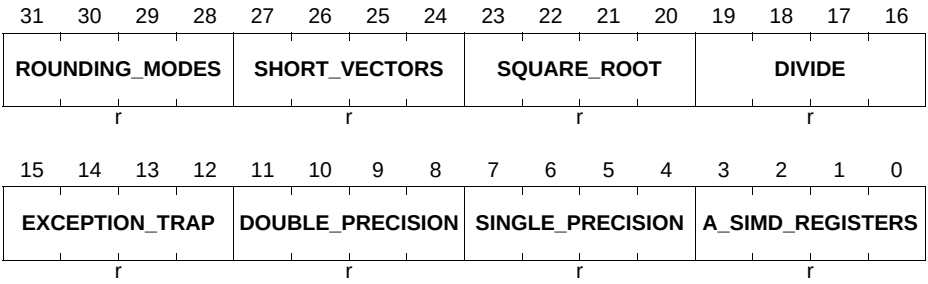
Field	Bits	Type	Description
RMode	[23:22]	rw	**Default value for FPSCR.RMode**
FZ	24	rw	**Default value for FPSCR.FZ**
DN	25	rw	**Default value for FPSCR.DN**
AHP	26	rw	**Default value for FPSCR.AHP**
0	[31:27], [21:0]	r	**Reserved** Read as 0; should be written with 0.

Media and FP Feature Register 0

Describes the features provided by the Floating-point extension. Must be interpreted with **MVFR1**.

MVFR0
Media and FP Feature Register 0

(E000 EF40$_H$) **Reset Value: 1011 0021$_H$**



Field	Bits	Type	Description
ROUNDING_MODES	[31:28]	r	**Indicates the rounding modes supported by the FP floating-point hardware.** 1_H : All rounding modes supported.
SHORT_VECTORS	[27:24]	r	**Indicates the hardware support for FP short vectors.** 0_H : Not supported.
SQUARE_ROOT	[23:20]	r	**Indicates the hardware support for FP square root operations.** 1_H : Supported.
DIVIDE	[19:16]	r	**Indicates the hardware support for FP divide operations.** 1_H : Supported.
EXCEPTION_TRAP	[15:12]	r	**Indicates whether the FP hardware implementation supports exception trapping.** 0_H : Not supported.
DOUBLE_PRECISION	[11:8]	r	**Indicates the hardware support for FP double-precision operations.** 0_H : Not supported.

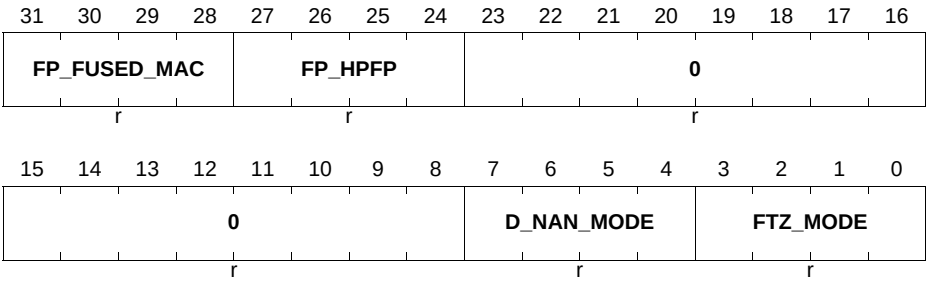
Field	Bits	Type	Description
SINGLE_PRECISION	[7:4]	r	**Indicates the hardware support for FP single-precision operations.** 2_H : Supported (Instruction to load a single-precision floating-point constant, and conversions between single-precision and fixed-point values is available).
A_SIMD_REGISTERS	[3:0]	r	**Indicates the size of the FP register bank.** 1_H : Supported, 16 x 64-bit registers.

Media and FP Feature Register 1

Describes the features provided by the Floating-point extension. Must be interpreted with **MVFR0**.

MVFR1
Media and FP Feature Register 1

(E000 EF44$_H$) **Reset Value: 1100 0011$_H$**



Field	Bits	Type	Description
FP_FUSED_MAC	[31:28]	r	**Indicates whether the FP supports fused multiply accumulate operations.** 1_H : Supported.
FP_HPFP	[27:24]	r	**Indicates whether the FP supports half-precision floating-point conversion operations.** 1_H : Supported.

Field	Bits	Type	Description
D_NAN_MODE	[7:4]	r	**Indicates whether the FP hardware implementation supports only the Default NaN mode.** 1_H : Hardware supports propagation of NaN values.
FTZ_MODE	[3:0]	r	**Indicates whether the FP hardware implementation supports only the Flush-to-Zero mode of operation.** 1_H : Hardware supports full denormalized number arithmetic.
0	[23:8]	r	**Reserved** Read as 0; should be written with 0.

3 Bus System

The XMC4[12]00 is targeted for use in embedded systems. Therefore the key features are timing determinism and low latency on real time events. Bus bandwidth is required particularly for communication peripherals.

The bus system will therefore provide:

- Timing Determinism
- Low Latency
- Performance
- Throughput

3.1 Bus Interfaces

This chapter describes the features for the two kinds of interfaces.

- **Memory Interface**
- **Peripheral Interface**

All on-chip peripherals and memories are attached to the Bus Matrix, in some cases via peripheral bridges. All on-chip modules implement Little Endian data organization. The following types of transfer are supported:

- Locked Transfers
- Burst Operation
- Protection Control

Pipelining is also supported for bandwidth critical transfers.

Memory Interface

The on-chip memories capable to accept a transfer request with each bus clock cycle.

The memory interface data bus width is 32-bit. Each memory slave support 32-bit, 16-bit and 8-bit access types.

Peripheral Interface

Each slave supports 32-bit accesses. Some slaves also support 8-bit and/or 16-bit accesses.

3.2 Bus Matrix

The central part of the bus system is built up around a multilayer AHB-lite compliant matrix. By means of this technique the bus masters and bus slaves can be connected in a flexible way while maintaining high bus performance.

The Bus Martix depicted in **Figure 3-1** implements an optimized topology enabling zero wait state data accesses between the Masters and Slaves connected to it. Dedicated

arbitration scheme enables optimal access conflicts resolution resulting in improved system stability and real time behavior.

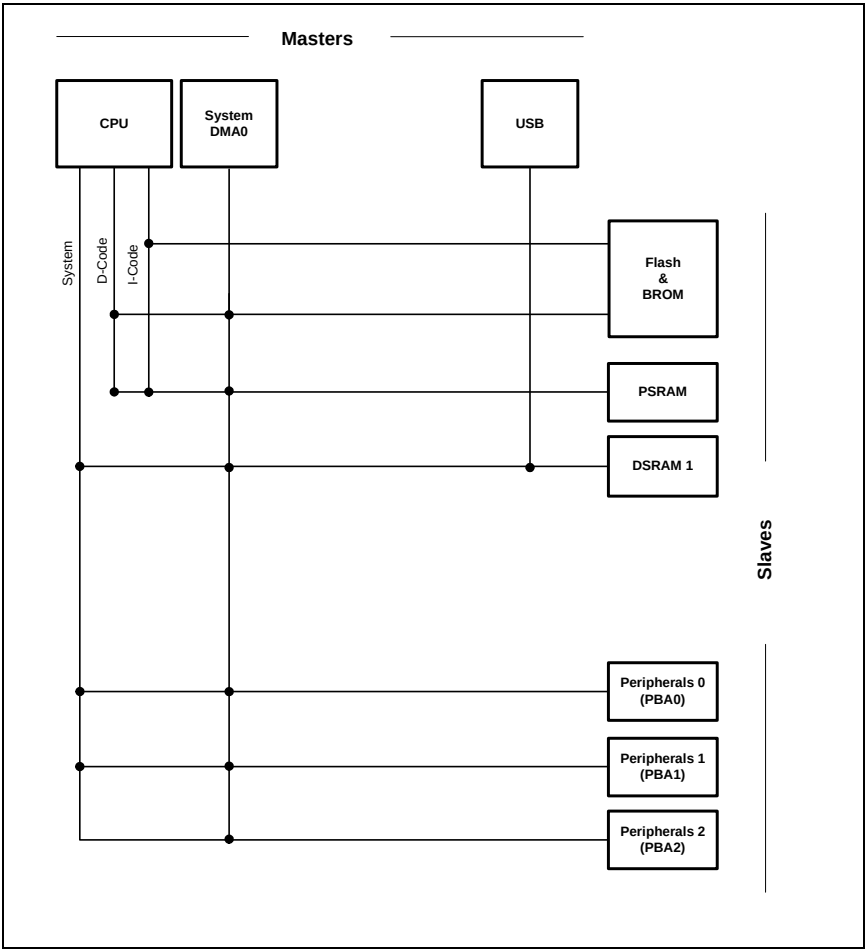


Figure 3-1 Multilayer Bus Matrix

Arbitration Priorities

In case of concurring access to the same slave the master with the highest priority is granted the bus.

Table 3-1 Access Priorities per Slave[1)]

	CPU	GPDMA0	USB
PMU/FLASH	1	2	-
PSRAM	1	2	-
DSRAM1	1	2	3
PBA0	1	2	-
PBA1	1	2	-
PBA2	1	2	-

1) Lower number means higher priority

4 Service Request Processing

A hardware pulse or level change is called Service Request (SR) in an XMC4[12]00 system. Service Requests are the fastest way to send trigger “messages” between connected on-chip resources.

An SR can generate any of the following requests

- Interrupt
- DMA
- Peripheral action

This chapter describes the available Service Requests and the different ways to select and process them.

Notes

1. *The CPU exception model and interrupt processing (by NVIC unit) are described in the CPU chapter.*
2. *General Purpose DMA request processing is described in the GPDMA chapter*

Table 4-1 Abbreviations

DLR	DMA Line Router
ERU	Event Request Unit
NVIC	Nested Vectored Interrupt Controller
SR	Service Request

4.1 Overview

Efficient Service Request Processing is based on the interconnect between the request sources and the request processing units. XMC4[12]00 provides both fixed and programmable interconnect.

4.1.1 Features

The following features are provided for Service Request processing:

- Connectivity matrix between Service Requests and request processing units
  - Fixed connections
  - Programmable connections using ERU
- Event Request Unit (ERU)
  - Flexible processing of external and internal service requests
  - Programmable for edge and/or level triggering
  - Multiple inputs per channel
  - Triggers combinable from multiple inputs
  - Input and output gating

- DMA Line Router (DLR)
  – Routing and processing of DMA requests

4.1.2 Block Diagram

The shaded components shown in **Figure 4-1** are described in this chapter.

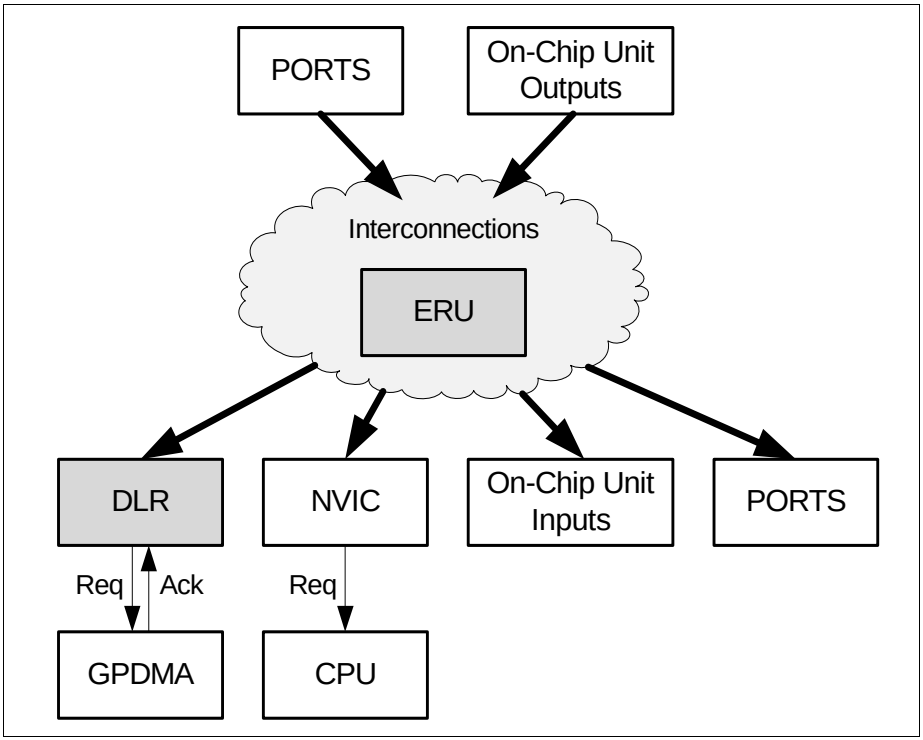


Figure 4-1 Block Diagram on Service Request Processing

4.2 Service Request Distribution

The following figure shown an example of how a service request can be distributed concurrently. To support the concurrent distribution to multiple receivers, the receiving modules are capable to enable/disable incoming requests.

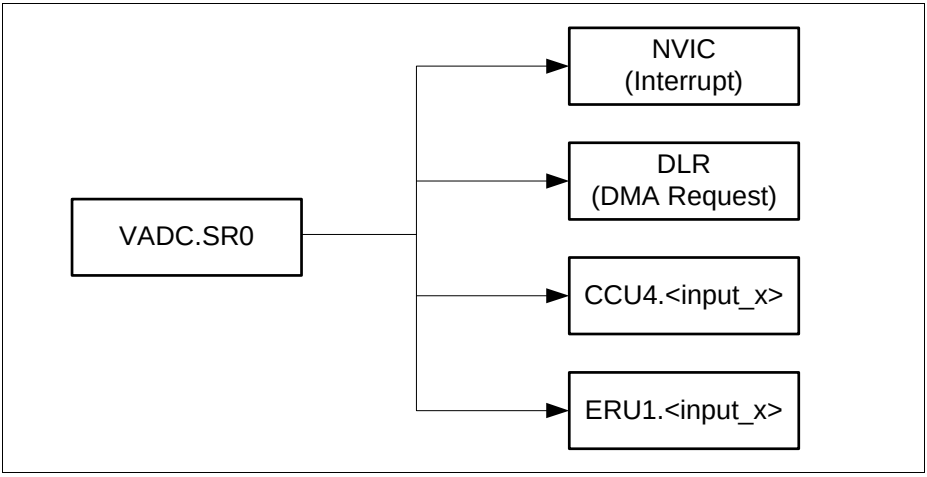


Figure 4-2 Example for Service Request Distribution

The units involved in Service Request distribution can be subdivided into

- Embedded real time services
- Interrupt and DMA services

Embedded real time services

Connectivity between On-Chip Units and PORTS is real time application and also chip package dependant. Related connectivity and availability of pins can be looked up in the

- “Interconnects” Section of the respective module(s) chapters
- “Parallel Ports” chapter and Data Sheet for PORTS
- Event Request Unit (**Section 4.5**)

Interrupt and DMA services

The following table gives an overview on the number of service requests per module and how the service requests are assigned to NVIC Interrupt and DLR/GPDMA service providers.

Service Requests can be of type “Level” or “Pulse”. The DLR/GPDMA can only process “Pulse” type of requests while the NVIC can process both. The type of Service Requests generated is listed in column “Type” in **Table 4-2**.

Table 4-2 Interrupt and DMA services per Module

Modules	Request Sources	NVIC	DLR/GPDMA	Type
VADC	12	12	12	Pulse
DAC	2	2	2	Pulse
CCU40-1	8	8	4	Pulse
CCU80	4	4	2	Pulse
HRPWM0	4	4	2	Pulse
POSIF0	2	2	-	Pulse
CAN	8	8	4	Pulse
USIC0-1	12	12	4	Pulse
LEDTS0	1	1	-	Pulse
FCE	1	1	-	Pulse
PMU0/Flash	1	1	-	Pulse
GPDMA0	1	1	-	Level
SCU	1	1	-	Level
ERU0-1	8	8	4	Pulse
USB0	1	1	-	Level
Totals	66	66	34	-

4.3 Interrupt Service Requests

The NVIC is an integral part of the Cortex M4 processor unit. Due to a tight coupling with the CPU it allows to achieve lowest interrupt latency and efficient processing of late arriving interrupts.

NVIC Features

- 112 interrupt nodes
- Programmable priority level of 0-63 for each interrupt node. A higher level corresponds to a lower priority, so level 0 is the highest interrupt priority
- Request source can be level or edge signal type
- Dynamic reprioritization of interrupts.
- Grouping of priority values into group priority and subpriority fields.
- Interrupt tail-chaining.
- One external Non-maskable interrupt (NMI)
- Relocatable vector table

- Software interrupt generation

Level-sensitive and pulse interrupts

The NVIC is capable to capture both level-sensitive and pulse interrupts.

- A level-sensitive interrupt is held asserted until the peripheral deasserts the interrupt signal. Deassertion is typically triggered by the interrupt service routine (ISR). It is
  - used for less frequent requests and
  - the ISR is often more complex and longer.
- A pulse interrupt is asserted and after a fixed period of time automatically deasserted. The period of time depends on the peripheral, please refer to the “Service Request Generation” section of the respective peripheral. It is
  - used for more frequent requests and
  - the ISR is often more simple and shorter.

The way to process both types of requests differs and is described in section “Level-sensitive and pulse interrupts” in the CPU chapter.

Service Request to IRQ Number Assignment

Table 4-3 lists the service request sources per on-chip unit and their assignment to NVIC IRQ numbers. The resulting exception number is calculated by adding 16 to the IRQ Number. The first 16 exception numbers are used by the Cortex M4 CPU. For calculation of the resulting exception routine address please refer to the CPU chapter.

Table 4-3 Interrupt Node assignment

Service Request	IRQ Number	Description
SCU.SR0	0	System Control
ERU0.SR0 ERU0.SR3	1...4	External Request Unit 0
ERU1.SR0 ERU1.SR3	5...8	External Request Unit 1
NC	9, 10, 11	Reserved
PMU0.SR0	12	Program Management Unit
NC	13	Reserved
VADC.C0SR0 - VADC.C0SR3	14...17	Analog to Digital Converter Common Block 0
VADC.G0SR0 - VADC.G0SR3	18...21	Analog to Digital Converter Group 0
VADC.G1SR0 - VADC.G1SR3	22...25	Analog to Digital Converter Group 1

Table 4-3 Interrupt Node assignment (cont'd)

Service Request	IRQ Number	Description
NC	26...41	Reserved
DAC.SR0 - DAC.SR1	42, 43	Digital to Analog Converter
CCU40.SR0 - CCU40.SR3	44...47	Capture Compare Unit 4 (Module 0)
CCU41.SR0 - CCU41.SR3	48...51	Capture Compare Unit 4 (Module 1)
NC	52...59	Reserved
CCU80.SR0 - CCU80.SR3	60...63	Capture Compare Unit 8 (Module 0)
NC	64...67	Reserved
POSIF0.SR0 - POSIF0.SR1	68...69	Position Interface (Module 0)
NC	70...71	Reserved
HRPWM0.SR0 - HRPWM0.SR1	72...75	High Resolution Pulse Width Modulation (Module 0)
CAN.SR0 - CAN.SR7	76...83	MultiCAN
USIC0.SR0 - USIC0.SR5	84...89	Universal Serial Interface Channel (Module 0)
USIC1.SR0 - USIC1.SR5	90...95	Universal Serial Interface Channel (Module 1)
NC	96...101	Reserved
LEDTS0.SR0	102	LED and Touch Sense Control Unit (Module 0)
NC	103	Reserved
FCE.SR0	104	Flexible CRC Engine
GPDMA0.SR0	105	General Purpose DMA unit 0
NC	106	Reserved
USB0.SR0	107	Universal Serial Bus
NC	108...111	Reserved

4.4 DMA Line Router (DLR)

The DMA line router provides the following functionality:

- Selection of DMA request sources
- Handling of the DMA request and acknowledge handshake
- Detection of service request overruns

4.4.1 Functional Description

This unit enables the user to select 8 DMA service requests out of the set of DMA capable service request sources. It handles the Request and Acknowledge handshake to the GPDMA unit. Furthermore it detects service request overruns.

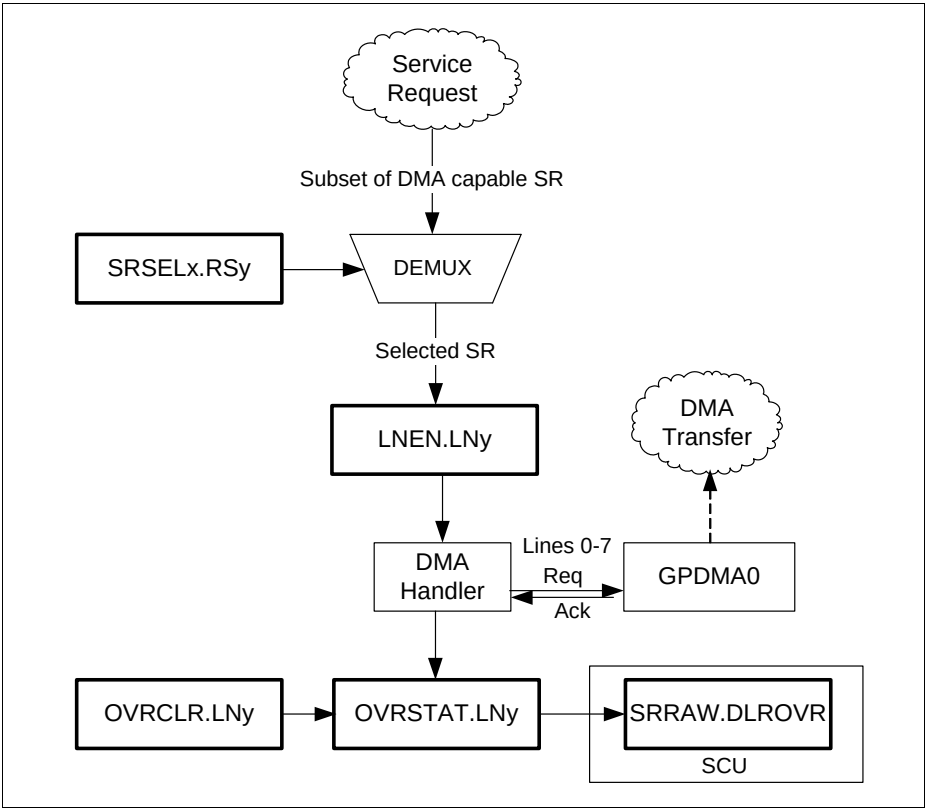


Figure 4-3 DMA Line Handler

For each DMA line the user can assign one service request source from the subset of DMA capable XMC4[12]00 service request sources. The assignment is done by programming the SRSx bit field of register **DLR_SRSEL0**. The DLR lines 0-7 are connected to GPDMA0.

If the selected service request pulse occurs and if the according line is enabled by the **DLR_LNEN** register, then the DMA handler forwards the request and stores it until the GPDMA responds with an acknowledge. A request pulse occurring while another transfer is ongoing is ignored and the according overrun status bit is set in the **DLR_OVRSTAT** register.

Once the overrun condition is entered the user can clear the overrun status bits by writing to the **DLR_OVRCLR** register. Additionally the pending request must be reset by successively disabling and enabling the respective line.

If any bit within the **DLR_OVRSTAT** register is set, a service request is flagged by setting the SCU_SRRAW.DLROVR bit.

The DLR unit has the following inputs:

Table 4-4 DMA Handler Service Request inputs

Service Request	# of Inputs	Description
ERU0.SR1 - ERU0.SR4	4	ERU0 (System Control) requests
VADC.C0SR0 - VADC.C0SR3	4	Analog to Digital Converter Common Block 0
VADC.G0SR0 - VADC.G0SR3	4	Analog to Digital Converter Group 0
VADC.G1SR0 - VADC.G1SR3	4	Analog to Digital Converter Group 1
DAC.SR0 - DAC.SR1	2	Digital to Analog Converter
CCU40.SR0 - CCU40.SR1	2	Capture Compare Unit 4 (Module 0)
CCU41.SR0 - CCU41.SR1	2	Capture Compare Unit 4 (Module 1)
CCU80.SR0 - CCU80.SR1	2	Capture Compare Unit 8 (Module 0)
CAN.SR0 - CAN.SR3	4	MultiCAN
USIC0.SR0 - USIC0.SR1	2	Universal Serial Interface Channel (Module 0)

Table 4-4 DMA Handler Service Request inputs (cont'd)

Service Request	# of Inputs	Description
USIC1.SR0 - USIC1.SR1	2	Universal Serial Interface Channel (Module 1)
HRPWM0.SR0 - HRPWM0.SR1	2	High Resolution PWM (Module 0)

4.4.2 DMA Service Request Source Selection

The selection of the request sources is done according to the following table by programming the **DLR_SRSEL0** register. Please note that each service request source can be assigned to 2 different lines to provide maximum flexibility. For example VADC.SR0 can be assigned to line 0 and 4.

Table 4-5 DMA Request Source Selection

DMA Line	DMA Request Line	Selected by DLR_SRSEL bit field
0	ERU0.SR0	RS0 = 0000_B
	VADC.C0SR0	RS0 = 0001_B
	VADC.G0SR3	RS0 = 0010_B
	Reserved	RS0 = 0011_B
	Reserved	RS0 = 0100_B
	Reserved	RS0 = 0101_B
	CCU40.SR0	RS0 = 0110_B
	CCU80.SR0	RS0 = 0111_B
	Reserved	RS0 = 1000_B
	CAN.SR0	RS0 = 1001_B
	USIC0.SR0	RS0 = 1010_B
	USIC1.SR0	RS0 = 1011_B
	Reserved	RS0 = 1100_B
	Reserved	RS0 = 1101_B
	Reserved	RS0 = 1110_B
	HRPWM0.SR0	RS0 = 1111_B
1	ERU0.SR3	RS1 = 0000_B
	VADC.C0SR1	RS1 = 0001_B
	VADC.G0SR2	RS1 = 0010_B

Table 4-5 DMA Request Source Selection (cont'd)

DMA Line	DMA Request Line	Selected by DLR_SRSEL bit field
	VADC.G1SR0	RS1 = 0011_B
	Reserved	RS1 = 0100_B
	DAC.SR0	RS1 = 0101_B
	CCU40.SR0	RS1 = 0110_B
	CCU80.SR0	RS1 = 0111_B
	Reserved	RS1 = 1000_B
	CAN.SR0	RS1 = 1001_B
	USIC0.SR0	RS1 = 1010_B
	USIC1.SR0	RS1 = 1011_B
	Reserved	RS1 = 1100_B
	Reserved	RS1 = 1101_B
	Reserved	RS1 = 1110_B
	HRPWM0.SR1	RS1 = 1111_B
2	ERU0.SR1	RS2 = 0000_B
	VADC.C0SR2	RS2 = 0001_B
	VADC.C0SR3	RS2 = 0010_B
	VADC.G1SR3	RS2 = 0011_B
	Reserved	RS2 = 0100_B
	Reserved	RS2 = 0101_B
	Reserved	RS2 = 0110_B
	CCU40.SR1	RS2 = 0111_B
	CCU80.SR1	RS2 = 1000_B
	Reserved	RS2 = 1001_B
	CAN.SR1	RS2 = 1010_B
	USIC0.SR1	RS2 = 1011_B
	USIC1.SR1	RS2 = 1100_B
	Reserved	RS2 = 1101_B
	Reserved	RS2 = 1110_B
	Reserved	RS2 = 1111_B
3	ERU0.SR2	RS3 = 0000_B

Table 4-5 DMA Request Source Selection (cont'd)

DMA Line	DMA Request Line	Selected by DLR_SRSEL bit field
	VADC.C0SR2	RS3 = 0001_B
	VADC.C0SR3	RS3 = 0010_B
	VADC.G1SR1	RS3 = 0011_B
	VADC.G1SR2	RS3 = 0100_B
	Reserved	RS3 = 0101_B
	DAC.SR1	RS3 = 0110_B
	CCU40.SR1	RS3 = 0111_B
	CCU80.SR1	RS3 = 1000_B
	Reserved	RS3 = 1001_B
	CAN.SR1	RS3 = 1010_B
	USIC0.SR1	RS3 = 1011_B
	USIC1.SR1	RS3 = 1100_B
	Reserved	RS3 = 1101_B
	Reserved	RS3 = 1110_B
	Reserved	RS3 = 1111_B
4	ERU0.SR2	RS4 = 0000_B
	VADC.G0SR0	RS4 = 0001_B
	VADC.G0SR1	RS4 = 0010_B
	Reserved	RS4 = 0011_B
	Reserved	RS4 = 0100_B
	Reserved	RS4 = 0101_B
	DAC.SR1	RS4 = 0110_B
	CCU41.SR0	RS4 = 0111_B
	Reserved	RS4 = 1000_B
	Reserved	RS4 = 1001_B
	CAN.SR2	RS4 = 1010_B
	USIC0.SR0	RS4 = 1011_B
	USIC1.SR0	RS4 = 1100_B
	Reserved	RS4 = 1101_B
	Reserved	RS4 = 1110_B

Table 4-5 DMA Request Source Selection (cont'd)

DMA Line	DMA Request Line	Selected by DLR_SRSEL bit field
	Reserved	RS4 = 1111_B
5	ERU0.SR1	RS5 = 0000_B
	VADC.G0SR0	RS5 = 0001_B
	VADC.G0SR1	RS5 = 0010_B
	VADC.G1SR2	RS5 = 0011_B
	Reserved	RS5 = 0100_B
	DAC.SR0	RS5 = 0101_B
	CCU41.SR0	RS5 = 0110_B
	Reserved	RS5 = 0111_B
	Reserved	RS5 = 1000_B
	CAN.SR2	RS5 = 1001_B
	USIC0.SR0	RS5 = 1010_B
	USIC1.SR0	RS5 = 1011_B
	Reserved	RS5 = 1100_B
	Reserved	RS5 = 1101_B
	Reserved	RS5 = 1110_B
	HRPWM0.SR0	RS5 = 1111_B
6	ERU0.SR3	RS6 = 0000_B
	VADC.C0SR1	RS6 = 0001_B
	VADC.G0SR2	RS6 = 0010_B
	VADC.G1SR1	RS6 = 0011_B
	Reserved	RS6 = 0100_B
	Reserved	RS6 = 0101_B
	Reserved	RS6 = 0110_B
	CCU41.SR1	RS6 = 0111_B
	Reserved	RS6 = 1000_B
	Reserved	RS6 = 1001_B
	CAN.SR3	RS6 = 1010_B
	USIC0.SR1	RS6 = 1011_B
	USIC1.SR1	RS6 = 1100_B

Table 4-5 DMA Request Source Selection (cont'd)

DMA Line	DMA Request Line	Selected by DLR_SRSEL bit field
	Reserved	RS6 = 1101_B
	Reserved	RS6 = 1110_B
	Reserved	RS6 = 1111_B
7	ERU0.SR0	RS7 = 0000_B
	VADC.C0SR0	RS7 = 0001_B
	VADC.G0SR3	RS7 = 0010_B
	VADC.G1SR0	RS7 = 0011_B
	VADC.G1SR3	RS7 = 0100_B
	Reserved	RS7 = 0101_B
	CCU41.SR1	RS7 = 0110_B
	Reserved	RS7 = 0111_B
	Reserved	RS7 = 1000_B
	CAN.SR3	RS7 = 1001_B
	USIC0.SR1	RS7 = 1010_B
	USIC1.SR1	RS7 = 1011_B
	Reserved	RS7 = 1100_B
	Reserved	RS7 = 1101_B
	Reserved	RS7 = 1110_B
	HRPWM0.SR1	RS7 = 1111_B

4.5 Event Request Unit (ERU)

The Event Request Unit (ERU) is a versatile multiple input event detection and processing unit. The XMC4[12]00 provides two units - ERU0 and ERU1.

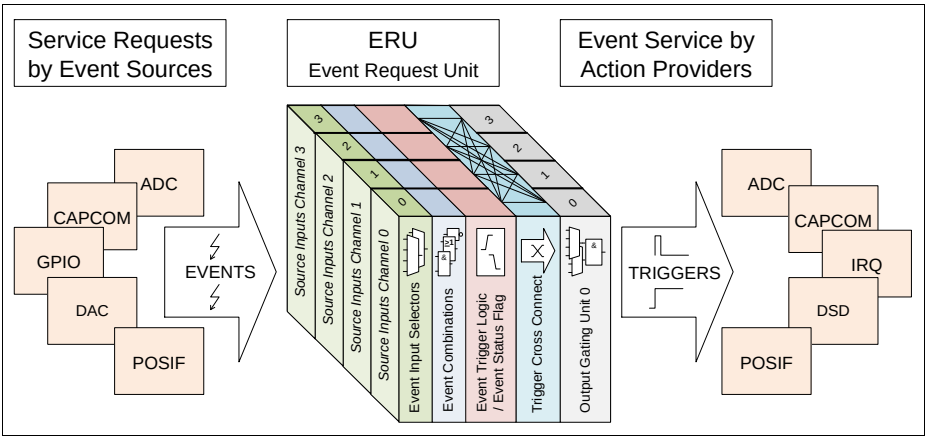


Figure 4-4 Event Request Unit Overview

Each ERU unit consists of the following blocks:

- An **Event Request Select (ERS)** unit.
  - Event Input Selectors allow the selection of one out of two inputs. For each of these two inputs, an vector of 4 possible signals is available.
  - Event Combinations allow a logical combination of two input signals to a common trigger.
- An **Event Trigger Logic (ETL)** per Input Channel allows the definition of the transition (edge selection, or by software) that lead to a trigger event and can also store this status. Here, the input levels of the selected signals are translated into events.
- The Trigger **Cross Connect Matrix** distributes the events and status flags to the Output Channels. Additionally, trigger signals from other modules are made available and can be combined with the local triggers.
- An **Output Gating Unit (OGU)** combines the trigger events and status information and gates the Output depending on a gating signal.

Note: An event of one Input can lead to reactions on several Outputs, or also events on several Inputs can be combined to a reaction on one Output.

4.5.1 Event Request Select Unit (ERS)

For each Input Channel x (x = 0-3), an ERSx unit handles the input selection for the associated ETLx unit. Each ERSx performs a logical combination of two signals (Ax, Bx) to provide one combined output signal ERSxO to the associated ETLx. Input Ax can be selected from 4 options of the input vector ERU_xA[3:0] and can be optionally inverted. A similar structure exists for input Bx (selection from ERU_xB[3:0]).

In addition to the direct choice of either input Ax or Bx or their inverted values, the possible logical combinations for two selected inputs are a logical AND or a logical OR.

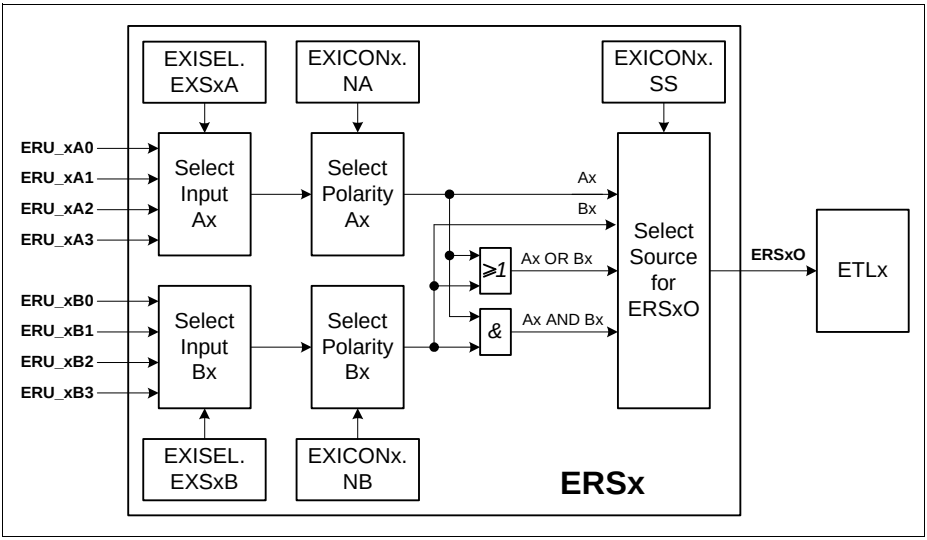


Figure 4-5 Event Request Select Unit Overview

The ERS units are controlled via register **ERU0_EXISEL** (one register for all four ERSx units) and registers EXICONx (one register for each ERSx and associated ETLx unit, e.g. **ERU0_EXICONx (x=0-3)** for Input Channel 0).

4.5.2 Event Trigger Logic (ETLx)

For each Input Channel x (x = 0-3), an event trigger logic ETLx derives a trigger event and related status information from the input ERSxO. Each ETLx is based on an edge detection block, where the detection of a rising or a falling edge can be individually enabled. Both edges lead to a trigger event if both enable bits are set (e.g. to handle a toggling input).

Each of the four ETLx units has an associated EXICONx register, that controls all options of an ETLx (the register also holds control bits for the associated ERSx unit, e.g. **ERU0_EXICONx (x=0-3)** to control ERS0 and ETL0).

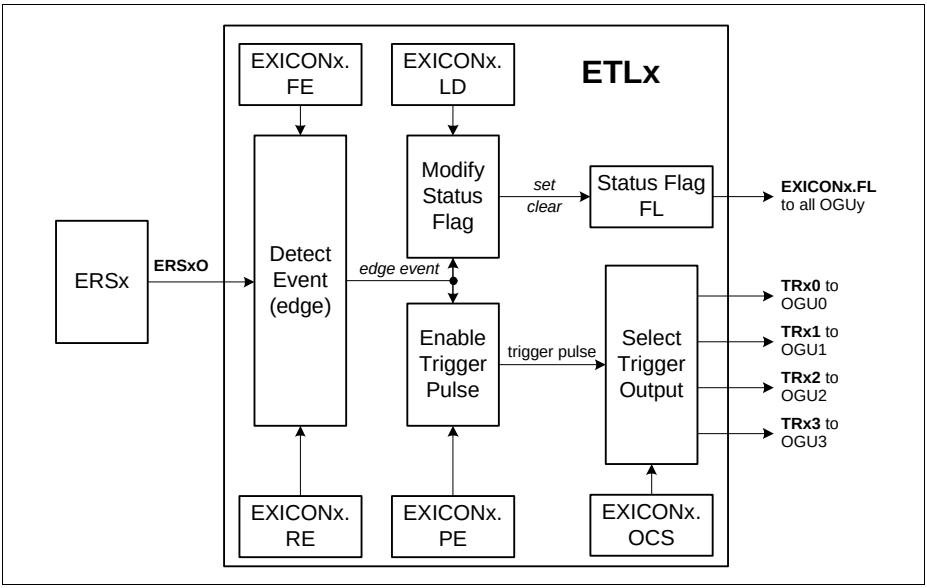


Figure 4-6 Event Trigger Logic Overview

When the selected event (edge) is detected, the status flag EXICONx.FL becomes set. This flag can also be modified by software. Two different operating modes are supported by this status flag.

It can be used as “sticky” flag, which is set by hardware when the desired event has been detected and has to be cleared by software. In this operating mode, it indicates that the event has taken place, but without indicating the actual status of the input.

In the second operating mode, it is cleared automatically if the “opposite” event is detected. For example, if only the falling edge detection is enabled to set the status flag, it is cleared when the rising edge is detected. In this mode, it can be used for pattern detection where the actual status of the input is important (enabling both edge detections is not useful in this mode).

The output of the status flag is connected to all following Output Gating Units (OGUy) in parallel (see **Figure 4-7**) to provide **pattern detection capability of all OGUy** units based on different or the same status flags.

In addition to the modification of the status flag, a trigger pulse output TRxy of ETLx can be enabled (by bit EXICONx.PE) and selected to **trigger actions in one of the OGUy** units. The target OGUy for the trigger is selected by bit field EXICON.OCS.

The trigger becomes active when the selected edge event is detected, independently from the status flag EXICONx.FL.

4.5.3 Cross Connect Matrix

The matrix shown in **Figure 4-7** distributes the trigger signals (TRxy) and status signals (EXICONx.FL) from the different ETLx units between the OGUy units. In addition, it receives peripheral trigger signals that can be OR-combined with the ETLx trigger signals in the OGUy units.

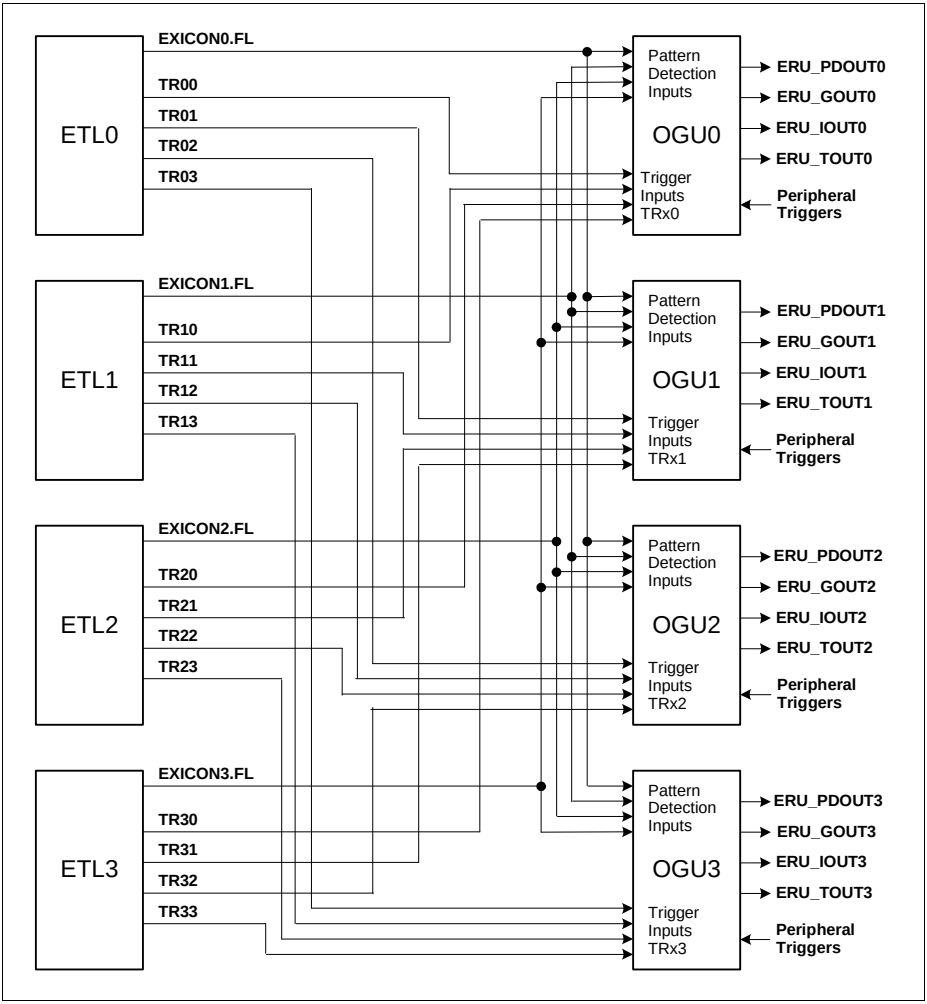


Figure 4-7 ERU Cross Connect Matrix

4.5.4 Output Gating Unit (OGUy)

Each OGUy (y = 0-3) unit combines the available trigger events and status flags from the Input Channels and distributes the results to the system. **Figure 4-8** illustrates the logic blocks within an OGUy unit. All functions of an OGUy unit are controlled by its associated EXOCONy register, e.g. **ERU0_EXOCONx (x=0-3)** for OGU0. The function of an OGUy unit can be split into two parts:

- **Trigger Combination**:
  All trigger signals TRxy from the Input Channels that are enabled and directed to OGUy, a selected peripheral-related trigger event, and a pattern change event (if enabled) are logically OR-combined.
- **Pattern Detection**:
  The status flags EXICONx.FL of the Input Channels can be enabled to take part in the pattern detection. A pattern match is detected while all enabled status flags are set.

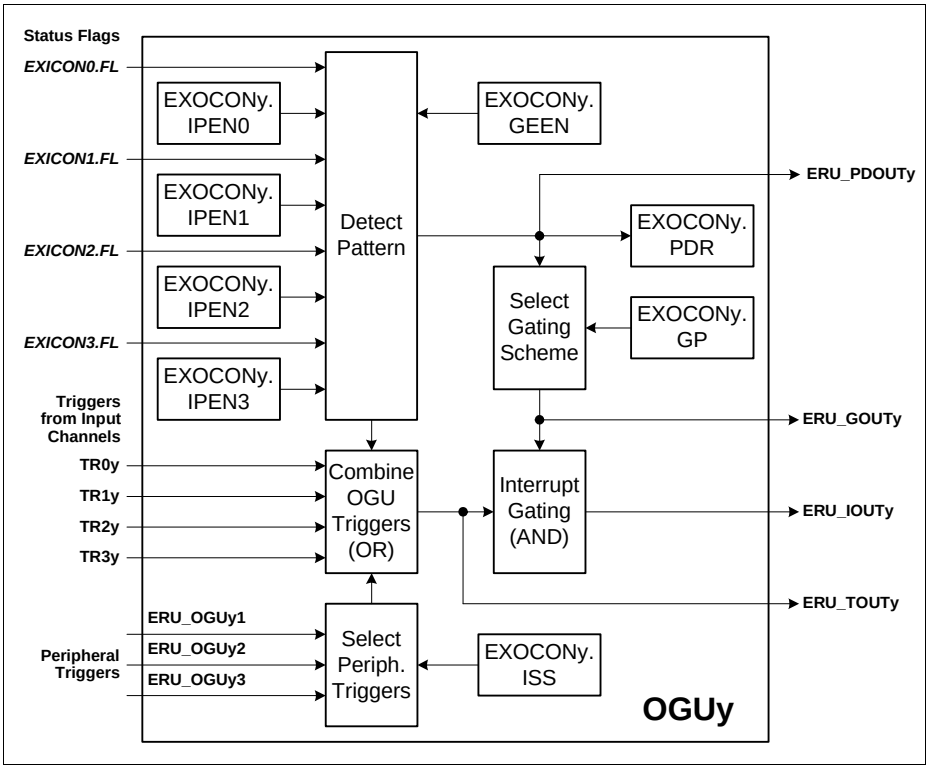


Figure 4-8 Output Gating Unit for Output Channel y

Each OGUy unit generates 4 output signals that are distributed to the system (not all of them are necessarily used):

- **ERU_PDOUTy** to directly output the pattern match information for gating purposes in other modules (pattern match = 1).
- **ERU_GOUTy** to output the pattern match or pattern miss information (inverted pattern match), or a permanent 0 or 1 under software control for gating purposes in other modules.
- **ERU_TOUTy** as combination of a peripheral trigger, a pattern detection result change event, or the ETLx trigger outputs TRxy to trigger actions in other modules.
- **ERU_IOUTy** as gated trigger output (ERU_GOUTy logical AND-combined with ERU_TOUTy) to trigger service requests (e.g. the service request generation can be gated to allow service request activation during a certain time window).

Trigger Combination

The trigger combination logically OR-combines different trigger inputs to form a common trigger ERU_TOUTy. Possible trigger inputs are:

- In each ETLx unit of the **Input Channels**, the trigger output TRxy can be enabled and the trigger event can be directed to one of the OGUy units.
- One out of three **peripheral trigger** signals per OGUy can be selected as additional trigger source. These peripheral triggers are generated by on-chip peripheral modules, such as capture/compare or timer units. The selection is done by bit field EXOCONy.ISS.
- In the case that at least one **pattern detection** input is enabled (EXOCONy.IPENx) and a change of the pattern detection result from pattern match to pattern miss (or vice-versa) is detected, a trigger event is generated to indicate a pattern detection result event (if enabled by ECOCONy.GEEN).

The trigger combination offers the possibility to program different trigger criteria for several input signals (independently for each Input Channel) or peripheral signals, and to combine their effects to a single output, e.g. to generate an service request or to start an ADC conversion. This combination capability allows the generation of a service request per OGU that can be triggered by several inputs (multitude of request sources results in one reaction).

The selection is defined by the bit fields ISS in registers **ERU0_EXOCONx (x=0-3)** (for ERU0.OGUx) and **ERU1_EXOCONy (y=0-3)** (for ERU1.OGUy).

Pattern Detection

The pattern detection logic allows the combination of the status flags of all ETLx units. Each status flag can be individually included or excluded from the pattern detection for each OGUy, via control bits EXOCONy.IPENx. The pattern detection block outputs the following pattern detection results:

- **Pattern match** (EXOCONy.PDR = 1 and ERU_PDOUTy = 1):
  A pattern match is indicated while all status flags FL that are included in the pattern detection are 1.
- **Pattern miss** (EXOCONy.PDR = 0 and ERU_PDOUTy = 0):
  A pattern miss is indicated while at least one of the status flags FL that are included in the pattern detection is 0.

In addition, the pattern detection can deliver a trigger event if the pattern detection result changes from match to miss or vice-versa (if enabled by EXOCONy.GEEN = 1). The pattern result change event is logically OR-combined with the other enabled trigger events to support service request generation or to trigger other module functions (e.g. in the ADC). The event is indicated when the pattern detection result changes and EXOCONy.PDR becomes updated.

The service request generation in the OGUy is based on the trigger ERU_TOUTy that can be gated (masked) with the pattern detection result ERU_PDOUTy. This allows an automatic and reproducible generation of service requests during a certain time window, where the request event is elaborated by the trigger combination block and the time window information (gating) is given by the pattern detection. For example, service requests can be issued on a regular time base (peripheral trigger input from capture/compare unit is selected) while a combination of input signals occurs (pattern detection based on ETLx status bits).

A programmable gating scheme introduces flexibility to adapt to application requirements and allows the generation of service requests ERU_IOUTy under different conditions:

- **Pattern match** (EXOCONy.GP = 10_B):
  A service request is issued when a trigger event occurs while the pattern detection shows a pattern match.
- **Pattern miss** (EXOCONy.GP = 11_B):
  A service request is issued when the trigger event occurs while the pattern detection shows a pattern miss.
- **Independent** of pattern detection (EXOCONy.GP = 01_B):
  In this mode, each occurring trigger event leads to a service request. The pattern detection output can be used independently from the trigger combination for gating purposes of other peripherals (independent use of ERU_TOUTy and ERU_PDOUTy with service requests on trigger events).
- **No service requests** (EXOCONy.GP = 00_B, default setting)
  In this mode, an occurring trigger event does not lead to a service request. The pattern detection output can be used independently from the trigger combination for gating purposes of other peripherals (independent use of ERU_TOUTy and ERU_PDOUTy without service requests on trigger events).

4.6 Service Request Generation

If any bit within the DLR.**DLR_OVRSTAT** register is set, a service request is flagged by setting the SCU_SRRAW.DLROVR bit.

To clear a request user must program the corresponding bit in the DLR.**DLR_OVRCLR** register. Then also clear SCU_SRRAW.DLROVR bit as described in the SCU chapter.

It is possible to mask out promotion of the service request by setting SCU_SRMSK.DLROVR bit.

4.7 Debug Behavior

Service request processing behavior is unchanged in debug mode.

4.8 Power, Reset and Clock

Service request processing is

- consuming power in all operating modes.
- running on f_{CPU}.
- asynchronously initialized by the system reset.

4.9 Initialization and System Dependencies

Service Requests must always be enabled at the source and at the destination. Additionally it must be checked whether it is necessary to program the ERU process and route a request.

Enabling Peripheral SRx Outputs

- Peripherals SRx outputs must be selectively enabled. This procedure depends on the individual peripheral. Please look up the section “Service Request Generation” within a peripherals chapter for details.
- Optionally ERUx must be programmed to process and route the request

Enabling External Requests

- Selected PORTS must be programmed for input
- ERUx must be programmed to process and route the external request

Note: The number of external service request inputs may be limited by the package used.

Enabling NVIC and GPDMA

Interrupt and DMA service request processing must be enabled. Please refer to the CPU and GPDMA chapters for details.

4.10 Registers

Registers Overview

The absolute register address is calculated by adding:

Module Base Address + Offset Address

Table 4-6 Registers Address Space

Module	Base Address	End Address	Note
DLR	5000 4900$_H$	5000 49FF$_H$	
ERU0	5000 4800$_H$	5000 48FF$_H$	
ERU1	4004 4000$_H$	4004 7FFF$_H$	

Table 4-7

Short Name	Description	Offset Addr.	Access Mode		Description See
			Read	Write	
DLR Registers					
OVRSTAT	Status of DMA Service Request Overruns	000$_H$	U, PV	PV	Page 4-23
OVRCLR	Clear Status of DMA Service Request Overruns	004$_H$	U, PV	PV	Page 4-24
SRSEL0	DLR Service Request Selection 0	008$_H$	U, PV	PV	Page 4-25
LNEN	Enable DLR Line	010$_H$	U, PV	PV	Page 4-24
ERU Registers					
EXISEL	ERU External Input Control Selection	0000$_H$	U, PV	PV	Page 4-26
EXICON0	ERU External Input Control Selection	0010$_H$	U, PV	PV	Page 4-28
EXICON1	ERU External Input Control Selection	0014$_H$	U, PV	PV	Page 4-28
EXICON2	ERU External Input Control Selection	0018$_H$	U, PV	PV	Page 4-28
EXICON3	ERU External Input Control Selection	001C$_H$	U, PV	PV	Page 4-28

Table 4-7 (cont'd)

Short Name	Description	Offset Addr.	Access Mode		Description See
			Read	Write	
EXOCON0	ERU Output Control Register	0020_H	U, PV	PV	Page 4-30
EXOCON1	ERU Output Control Register	0024_H	U, PV	PV	Page 4-30
EXOCON2	ERU Output Control Register	0028_H	U, PV	PV	Page 4-30
EXOCON3	ERU Output Control Register	$002C_H$	U, PV	PV	Page 4-30

4.10.1 DLR Registers

DLR_OVRSTAT

The DLR_OVRSTAT register is used to track status of GPDMA service request overruns. Upon overrun detection, additionally a service request flag is set in the SCU_SRRAW.DLROVR bit.

DLR_OVRSTAT
Overrun Status **(00_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0								LN7	LN6	LN5	LN4	LN3	LN2	LN1	LN0
r								rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
LNx (x = 0-7)	x	rh	**Line x Overrun Status** Set if an overrun occurred on this line.
0	[31:8]	r	**Reserved** Read as 0; should be written with 0.

DLR_OVRCLR

The DLR_OVRCLR register is used to clear the DLR_OVRSTAT register bits.

DLR_OVRCLR
Overrun Clear **(04_H)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0								LN7	LN6	LN5	LN4	LN3	LN2	LN1	LN0
r								w	w	w	w	w	w	w	w

Field	Bits	Type	Description
LNx (x = 0-7)	x	w	**Line x Overrun Status Clear** Clears the corresponding bit in the DLR_OVRSTAT register when set to 1.
0	[31:8]	r	**Reserved** Read as 0; should be written with 0.

DLR_LNEN

The DLR_LNEN register is used to enable each individual DLR line and to reset a previously stored and pending service request.

DLR_LNEN
Line Enable **(10_H)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0								LN7	LN6	LN5	LN4	LN3	LN2	LN1	LN0
r								rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
LNx (x = 0-7)	x	rw	**Line x Enable** 0_B Disables the line 1_B Enables the line and resets a pending request
0	[31:8]	r	**Reserved** Read as 0; should be written with 0.

DLR_SRSEL0

The DLR_SRSEL0 register is used to select the service request source used to trigger a DMA transfer.

DLR_SRSEL0
Service Request Selection 0

(08_H) **Reset Value: 0000 0000$_H$**

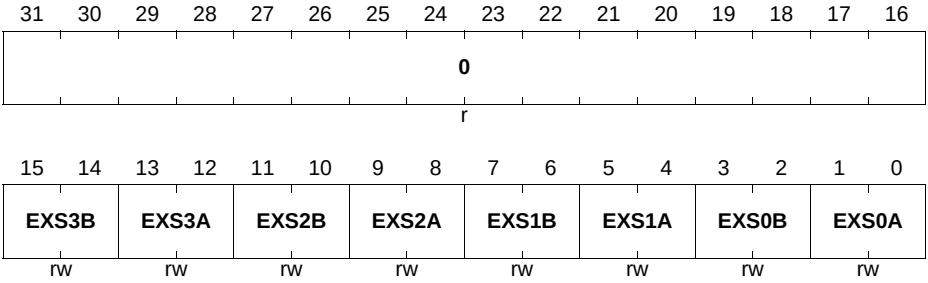
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RS7				RS6				RS5				RS4			
rw				rw				rw				rw			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RS3				RS2				RS1				RS0			
rw				rw				rw				rw			

Field	Bits	Type	Description
RSx (x = 0-7)	[x*4+3: x*4]	rw	**Request Source for Line x** The request source according to **Table 4-5** is selected for DMA line x.These lines are connected to GPDMA0

4.10.2 ERU Registers

ERU0_EXISEL
Event Input Select **(00_H)** **Reset Value: 0000 0000_H**
ERU1_EXISEL
Event Input Select **(0000_H)** **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
EXS0A	[1:0]	rw	**Event Source Select for A0 (ERS0)** This bit field defines which input is selected for A0. 00_B Input ERU_0A0 is selected 01_B Input ERU_0A1 is selected 10_B Input ERU_0A2 is selected 11_B Input ERU_0A3 is selected
EXS0B	[3:2]	rw	**Event Source Select for B0 (ERS0)** This bit field defines which input is selected for B0. 00_B Input ERU_0B0 is selected 01_B Input ERU_0B1 is selected 10_B Input ERU_0B2 is selected 11_B Input ERU_0B3 is selected
EXS1A	[5:4]	rw	**Event Source Select for A1 (ERS1)** This bit field defines which input is selected for A1. 00_B Input ERU_1A0 is selected 01_B Input ERU_1A1 is selected 10_B Input ERU_1A2 is selected 11_B Input ERU_1A3 is selected

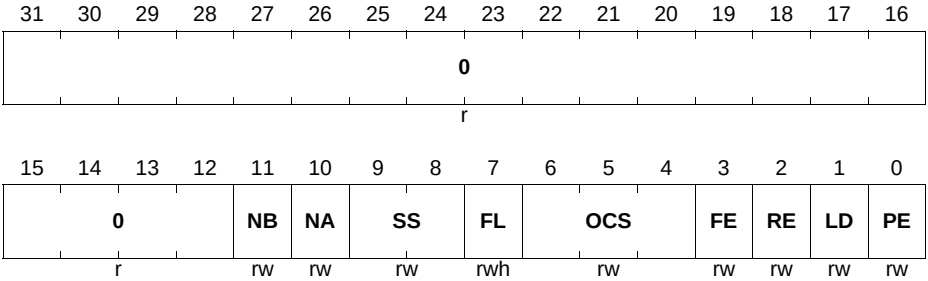
Field	Bits	Type	Description
EXS1B	[7:6]	rw	**Event Source Select for B1 (ERS1)** This bit field defines which input is selected for B1. 00_B Input ERU_1B0 is selected 01_B Input ERU_1B1 is selected 10_B Input ERU_1B2 is selected 11_B Input ERU_1B3 is selected
EXS2A	[9:8]	rw	**Event Source Select for A2 (ERS2)** This bit field defines which input is selected for A2. 00_B Input ERU_2A0 is selected 01_B Input ERU_2A1 is selected 10_B Input ERU_2A2 is selected 11_B Input ERU_2A3 is selected
EXS2B	[11:10]	rw	**Event Source Select for B2 (ERS2)** This bit field defines which input is selected for B2. 00_B Input ERU_2B0 is selected 01_B Input ERU_2B1 is selected 10_B Input ERU_2B2 is selected 11_B Input ERU_2B3 is selected
EXS3A	[13:12]	rw	**Event Source Select for A3 (ERS3)** This bit field defines which input is selected for A3. 00_B Input ERU_3A0 is selected 01_B Input ERU_3A1 is selected 10_B Input ERU_3A2 is selected 11_B Input ERU_3A3 is selected
EXS3B	[15:14]	rw	**Event Source Select for B3 (ERS3)** This bit field defines which input is selected for B3. 00_B Input ERU_3B0 is selected 01_B Input ERU_3B1 is selected 10_B Input ERU_3B2 is selected 11_B Input ERU_3B3 is selected
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

ERU0_EXICONx (x=0-3)
Event Input Control x

(10_H + 4*x) **Reset Value: 0000 0000_H**

ERU1_EXICONy (y=0-3)
Event Input Control y

(0010_H + 4*y) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
PE	0	rw	**Output Trigger Pulse Enable for ETLx** This bit enables the generation of an output trigger pulse at TRxy when the selected edge is detected (set condition for the status flag FL). 0_B The trigger pulse generation is disabled 1_B The trigger pulse generation is enabled
LD	1	rw	**Rebuild Level Detection for Status Flag for ETLx** This bit selects if the status flag FL is used as “sticky” bit or if it rebuilds the result of a level detection. 0_B The status flag FL is not cleared by hardware and is used as “sticky” bit. Once set, it is not influenced by any edge until it becomes cleared by software. 1_B The status flag FL rebuilds a level detection of the desired event. It becomes automatically set with a rising edge if RE = 1 or with a falling edge if FE = 1. It becomes automatically cleared with a rising edge if RE = 0 or with a falling edge if FE = 0.

Field	Bits	Type	Description
RE	2	rw	**Rising Edge Detection Enable ETLx** This bit enables/disables the rising edge event as edge event as set condition for the status flag FL or as possible trigger pulse for TRxy. 0_B A rising edge is not considered as edge event 1_B A rising edge is considered as edge event
FE	3	rw	**Falling Edge Detection Enable ETLx** This bit enables/disables the falling edge event as edge event as set condition for the status flag FL or as possible trigger pulse for TRxy. 0_B A falling edge is not considered as edge event 1_B A falling edge is considered as edge event
OCS	[6:4]	rw	**Output Channel Select for ETLx Output Trigger Pulse** This bit field defines which Output Channel OGUy is targeted by an enabled trigger pulse TRxy. 000_B Trigger pulses are sent to OGU0 001_B Trigger pulses are sent to OGU1 010_B Trigger pulses are sent to OGU2 011_B Trigger pulses are sent to OGU3 Others: Reserved, do not use this combination
FL	7	rwh	**Status Flag for ETLx** This bit represents the status flag that becomes set or cleared by the edge detection. 0_B The enabled edge event has not been detected 1_B The enabled edge event has been detected
SS	[9:8]	rw	**Input Source Select for ERSx** This bit field defines which logical combination is taken into account as ERSxO. 00_B Input A without additional combination 01_B Input B without additional combination 10_B Input A OR input B 11_B Input A AND input B
NA	10	rw	**Input A Negation Select for ERSx** This bit selects the polarity for the input A. 0_B Input A is used directly 1_B Input A is inverted

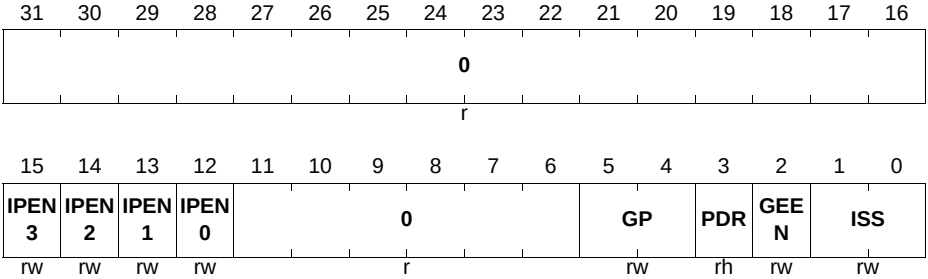
Field	Bits	Type	Description
NB	11	rw	**Input B Negation Select for ERSx** This bit selects the polarity for the input B. 0_B Input B is used directly 1_B Input B is inverted
0	[31:12]	r	**Reserved** Read as 0; should be written with 0.

ERU0_EXOCONx (x=0-3)
Event Output Trigger Control x

(20_H + 4*x) **Reset Value: 0000 0008$_H$**

ERU1_EXOCONy (y=0-3)
Event Output Trigger Control y

(0020_H + 4*y) **Reset Value: 0000 0008$_H$**



Field	Bits	Type	Description
ISS	[1:0]	rw	**Internal Trigger Source Selection** This bit field defines which input is selected as peripheral trigger input for OGUy. 00_B The peripheral trigger function is disabled 01_B Input ERU_OGUy1 is selected 10_B Input ERU_OGUy2 is selected 11_B Input ERU_OGUy3 is selected
GEEN	2	rw	**Gating Event Enable** Bit GEEN enables the generation of a trigger event when the result of the pattern detection changes from match to miss or vice-versa. 0_B The event detection is disabled 1_B The event detection is enabled

Field	Bits	Type	Description
PDR	3	rh	**Pattern Detection Result Flag** This bit represents the pattern detection result. 0_B A pattern miss is detected 1_B A pattern match is detected
GP	[5:4]	rw	**Gating Selection for Pattern Detection Result** This bit field defines the gating scheme for the service request generation (relation between the OGU output ERU_PDOUTy and ERU_GOUTy). 00_B ERU_GOUTy is always disabled and ERU_IOUTy can not be activated 01_B ERU_GOUTy is always enabled and ERU_IOUTy becomes activated with each activation of ERU_TOUTy 10_B ERU_GOUTy is equal to ERU_PDOUTy and ERU_IOUTy becomes activated with an activation of ERU_TOUTy while the desired pattern is detected (pattern match PDR = 1) 11_B ERU_GOUTy is inverted to ERU_PDOUTy and ERU_IOUTy becomes activated with an activation of ERU_TOUTy while the desired pattern is not detected (pattern miss PDR = 0)
IPENx (x = 0-3)	12+x	rw	**Pattern Detection Enable for ETLx** Bit IPENx defines whether the trigger event status flag EXICONx.FL of ETLx takes part in the pattern detection of OGUy. 0_B Flag EXICONx.FL is excluded from the pattern detection 1_B Flag EXICONx.FL is included in the pattern detection
0	[31:16], [11:6]	r	**Reserved** Read as 0; should be written with 0.

4.11 Interconnects

This section describes how the ERU0 and ERU1 modules are connected within the XMC4[12]00 system.

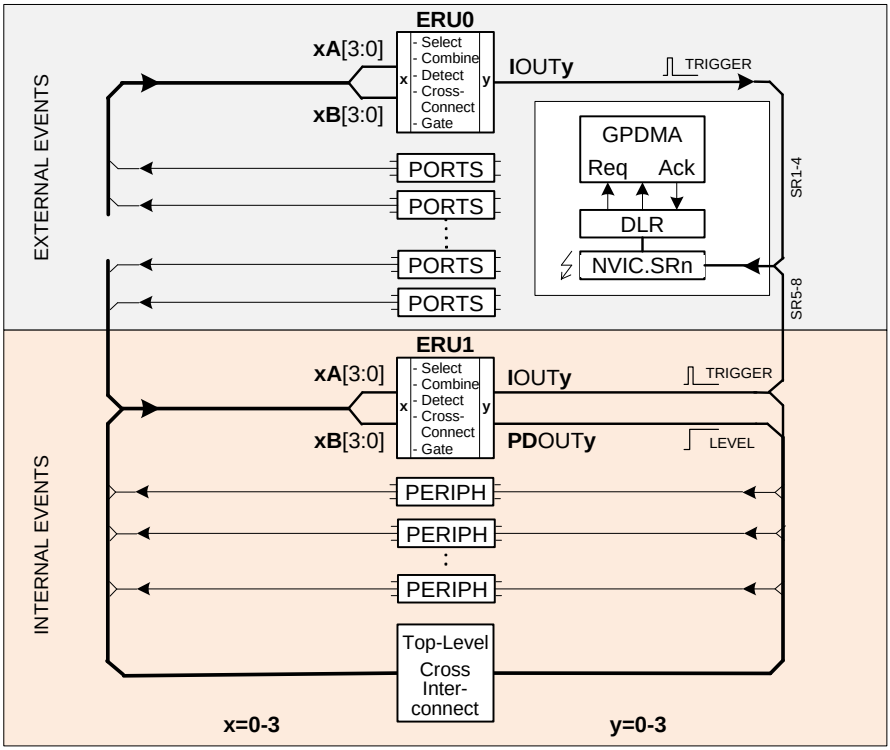


Figure 4-9 ERU Interconnects Overview

4.11.1 ERU0 Connections

The following table shows the ERU0 connections. Please refer to the ports chapter for details about PORTS connections.

Table 4-8 ERU0 Pin Connections

Global Inputs/Outputs	I/O	Connected To	Description
ERU0.0A0	I	PORTS	
ERU0.0A1	I	PORTS	
ERU0.0A2	I	PORTS	
ERU0.0A3	I	SCU.G0ORCOUT6	

Table 4-8 ERU0 Pin Connections

Global Inputs/Outputs	I/O	Connected To	Description
ERU0.0B0	I	PORTS	
ERU0.0B1	I	PORTS	
ERU0.0B2	I	PORTS	
ERU0.0B3	I	PORTS	
ERU0.1A0	I	PORTS	
ERU0.1A1	I	SCU.HIB_SR0	
ERU0.1A2	I	PORTS	
ERU0.1A3	I	SCU.G0ORCOUT7	
ERU0.1B0	I	PORTS	
ERU0.1B1	I	SCU.HIB_SR1	
ERU0.1B2	I	PORTS	
ERU0.1B3	I	PORTS	
ERU0.2A0	I	PORTS	
ERU0.2A1	I	PORTS	
ERU0.2A2	I	PORTS	
ERU0.2A3	I	SCU.G1ORCOUT6	
ERU0.2B0	I	PORTS	
ERU0.2B1	I	PORTS	
ERU0.2B2	I	PORTS	
ERU0.2B3	I	PORTS	
ERU0.3A0	I	PORTS	
ERU0.3A1	I	PORTS	
ERU0.3A2	I	PORTS	
ERU0.3A3	I	SCU.G1ORCOUT7	
ERU0.3B0	I	PORTS	
ERU0.3B1	I	PORTS	
ERU0.3B2	I	PORTS	
ERU0.3B3	I	PORTS	
ERU0.OGU01	I	0	
ERU0.OGU02	I	0	

Table 4-8 ERU0 Pin Connections

Global Inputs/Outputs	I/O	Connected To	Description
ERU0.OGU03	I	1	
ERU0.OGU11	I	0	
ERU0.OGU12	I	0	
ERU0.OGU13	I	1	
ERU0.OGU21	I	0	
ERU0.OGU22	I	0	
ERU0.OGU23	I	1	
ERU0.OGU31	I	0	
ERU0.OGU32	I	0	
ERU0.OGU33	I	1	
ERU0.PDOUT0	O	not connected	
ERU0.GOUT0	O	not connected	
ERU0.TOUT0	O	not connected	
ERU0.IOUT0	O	SCU.ERU0.SR0 DLR	
ERU0.PDOUT1	O	not connected	
ERU0.GOUT1	O	not connected	
ERU0.TOUT1	O	not connected	
ERU0.IOUT1	O	SCU.ERU0.SR1 DLR	
ERU0.PDOUT2	O	not connected	
ERU0.GOUT2	O	not connected	
ERU0.TOUT2	O	not connected	
ERU0.IOUT2	O	SCU.ERU0.SR2 DLR	
ERU0.PDOUT3	O	not connected	
ERU0.GOUT3	O	not connected	
ERU0.TOUT3	O	not connected	
ERU0.IOUT3	O	SCU.ERU0.SR3 DLR	

4.11.2 ERU1 Connections

The following table shows the ERU1 connections. Please refer to the ports chapter for details about PORTS connections.

Table 4-9 ERU1 Pin Connections

Global Inputs/Outputs	I/O	Connected To	Description
ERU1.0A0	I	PORTS	
ERU1.0A1	I	POSIF0.SR1	
ERU1.0A2	I	CCU40.ST0	
ERU1.0A3	I	DAC.SIGN_0	
ERU1.0B0	I	PORTS	
ERU1.0B1	I	CCU80.ST0	
ERU1.0B2	I	VADC.G0BFL3	
ERU1.0B3	I	ERU1.IOUT3	
ERU1.1A0	I	PORTS	
ERU1.1A1	I	POSIF0.SR1	
ERU1.1A2	I	CCU40.ST1	
ERU1.1A3	I	ERU1.IOUT2	
ERU1.1B0	I	PORTS	
ERU1.1B1	I	CCU80.ST1	
ERU1.1B2	I	VADC.G1BFL3	
ERU1.1B3	I	ERU1.IOUT2	
ERU1.2A0	I	PORTS	
ERU1.2A1	I	not connected	
ERU1.2A2	I	CCU40.ST2	
ERU1.2A3	I	DAC.SIGN_1	
ERU1.2B0	I	PORTS	
ERU1.2B1	I	CCU80.ST2	
ERU1.2B2	I	VADC.G0BFL3	
ERU1.2B3	I	HRPWM0.C0O	
ERU1.3A0	I	PORTS	
ERU1.3A1	I	not connected	

Table 4-9 ERU1 Pin Connections

Global Inputs/Outputs	I/O	Connected To	Description
ERU1.3A2	I	CCU40.ST3	
ERU1.3A3	I	HRPWM0.C1O	
ERU1.3B0	I	PORTS	
ERU1.3B1	I	CCU80.ST3	
ERU1.3B2	I	VADC.G1BFL3	
ERU1.3B3	I	HRPWM0.C2O	
ERU1.OGU01	I	VADC.C0SR0	
ERU1.OGU02	I	CCU40.ST0	
ERU1.OGU03	I	1	
ERU1.OGU11	I	VADC.C0SR1	
ERU1.OGU12	I	CCU41.ST0	
ERU1.OGU13	I	1	
ERU1.OGU21	I	VADC.C0SR2	
ERU1.OGU22	I	1	
ERU1.OGU23	I	1	
ERU1.OGU31	I	VADC.C0SR3	
ERU1.OGU32	I	1	
ERU1.OGU33	I	1	

Table 4-9 ERU1 Pin Connections

Global Inputs/Outputs	I/O	Connected To	Description
ERU1.PDOUT0	O	CCU40.IN0J CCU41.IN0J CCU40.IN1D CCU40.IN2D CCU40.IN3D CCU41.IN1D CCU41.IN2D CCU41.IN3D CCU80.IN0J CCU80.IN1J CCU80.IN2J CCU80.IN3J VADC.G0REQGTO VADC.G1REQGTO VADC.BGREQGTO POSIF0.IN0D PORTS	
ERU1.GOUT0	O	not connected	
ERU1.TOUT0	O	not connected	

Table 4-9 ERU1 Pin Connections

Global Inputs/Outputs	I/O	Connected To	Description
ERU1.IOUT0	O	CCU40.IN0K CCU41.IN0K CCU80.IN0G VADC.G0REQTRM VADC.G1REQTRM VADC.BGREQTRM CCU40.MCLKA CCU41.MCLKA CCU80.MCLKA NVIC.ERU1.SR0 POSIF0.EWHEB HRPWM0.ECLK0B HRPWM0.ECLK1B HRPWM0.ECLK2B HRPWM0.BL0O HRPWM0.BL1O HRPWM0.BL2O HRPWM0.SC0IO HRPWM0.SC1IO HRPWM0.SC2IO	
ERU1.PDOUT1	O	CCU40.IN1J CCU41.IN1J CCU40.IN0D CCU41.IN0D VADC.G0REQGTP VADC.G1REQGTP VADC.BGREQGTP POSIF0.IN1D PORTS	
ERU1.GOUT1	O	not connected	
ERU1.TOUT1	O	not connected	

Table 4-9 ERU1 Pin Connections

Global Inputs/Outputs	I/O	Connected To	Description
ERU1.IOUT1	O	CCU40.IN1K CCU41.IN1K CCU80.IN1G VADC.G0REQTRN VADC.G1REQTRN VADC.BGREQTRN CCU40.MCLKB CCU41.MCLKB CCU80.MCLKB NVIC.ERU1.SR1 POSIF0.EWHEC HRPWM0.BL0P HRPWM0.BL1P HRPWM0.BL2P HRPWM0.SC0IP HRPWM0.SC1IP HRPWM0.SC2IP	
ERU1.PDOUT2	O	CCU40.IN2J CCU41.IN2J CCU80.IN2F POSIF0.IN2D PORTS	
ERU1.GOUT2	O	not connected	
ERU1.TOUT2	O	not connected	
ERU1.IOUT2	O	CCU40.IN2K CCU41.IN2K CCU80.IN2G ERU1.1B3 NVIC.ERU1.SR2 POSIF0.MSETF	
ERU1.PDOUT3	O	CCU40.IN3J CCU41.IN3J CCU80.IN3F PORTS	
ERU1.GOUT3	O	not connected	

Table 4-9 ERU1 Pin Connections

Global Inputs/Outputs	I/O	Connected To	Description
ERU1.TOUT3	O	not connected	
ERU1.IOUT3	O	CCU40.IN3K CCU41.IN3K CCU80.IN3G ERU1.0B3 NVIC.ERU1.SR3 HRPWM0.ECLK0C HRPWM0.ECLK1C HRPWM0.ECLK2C	

5 General Purpose DMA (GPDMA)

The GPDMA is a highly configurable DMA controller, that allows high-speed data transfers between peripherals and memories. Complex data transfers can be done with minimal intervention of the processor, keeping this way the CPU resources free for other operations.

Extensive support for the microcontroller peripherals, like A/D and D/A converters, Timers, Communication Interfaces (USIC) via the GPDMA, unload the CPU and increase the efficiency and parallelism, for a high arrangement of real-time applications.

Table 5-1 Abbreviations table

GPDMAx	General Purpose DMA instance x
SCU	System Control Unit
DLR	DMA Line Router
f_{DMA}	GPDMA clock frequency

5.1 Overview

The GPDMA module enables hardware or software controlled data transfers between all microcontroller modules with the exclusion of those modules which provide built-in DMA functionality (USB and Ethernet).

Each GPDMA module contains a dedicated set of highly programmable channels, that can accommodate several type of peripheral-to-peripheral, peripheral-to-memory and memory-to-memory transfers.

The link between a highly programmable channel allocation and channel priority, gives a high benefit for applications that need high efficiency and parallelism.

The built-in fast DMA request handling together with the flexible peripheral configuration, enables the implementation of very demanding application software loops.

5.1.1 Features

The GPDMA component includes the following features.

General

- Bus interfaces
  - 1 Bus master interface per DMA unit
  - 1 Bus slave interface per DMA unit
- Channels
  - One GPDMA0 unit with 8 channels
  - Programmable channel priority
- Transfers

– Support for memory-to-memory, memory-to-peripheral, peripheral-to-memory, and peripheral-to-peripheral DMA transfers

Channels

All channels can be programmed for the following transfer modes

- DMA triggered by software or selectable from hardware service request sources
- Programmable source and destination addresses
- Address increment, decrement, or no change

Channels 0 and 1 of GPDMA0 can be programmed for the following transfer modes

- Multi-block transfers achieved through:
  – Linked Lists (block chaining)
  – Auto-reloading of channel registers
  – Contiguous address between blocks
- Independent source and destination selection of multi-block transfer type
- Scatter/Gather - source and destination areas do not need to be in a contiguous memory space

The GPDMA0 channels 0 and 1 provide a FIFO of 32 Bytes (eight 32-bit entries). These channels can be used to execute burst transfers up to a fixed length burst size of 8. The remaining channels FIFO size is 8 Bytes.

Channel Control

- Programmable source and destination for each channel
- Programmable burst transaction size for each channel
- Programmable enable and disable of DMA channel
- Support for disabling channel without data loss
- Support for suspension of DMA operation
- Support for ERROR response
- Bus locking - programmable over transaction, block, or DMA transfer level
- Channel locking - programmable over transaction, block, or DMA transfer level
- Optional writeback of the Channel Control register at the end of every block transfer

Interrupts

- Combined and separate interrupt service requests
- Request generation on:
  – DMA transfer completion
  – Block transfer completion
  – Single and burst transaction completion
  – Error condition
- Support of interrupt enabling and masking

5.1.2 Block Diagram

Figure 5-1 shows the following functional groupings of the main interfaces to the GPDMA block:

- DMA hardware request interface (DLR)
- Up to eight channels
- Arbiter
- Bus Master and Slave interfaces

One channel of the GPDMA is required for each source/destination pair. The master interface reads the data from a source peripheral and writes it to a destination peripheral. Two physical transfers are therefore required for each DMA transaction.

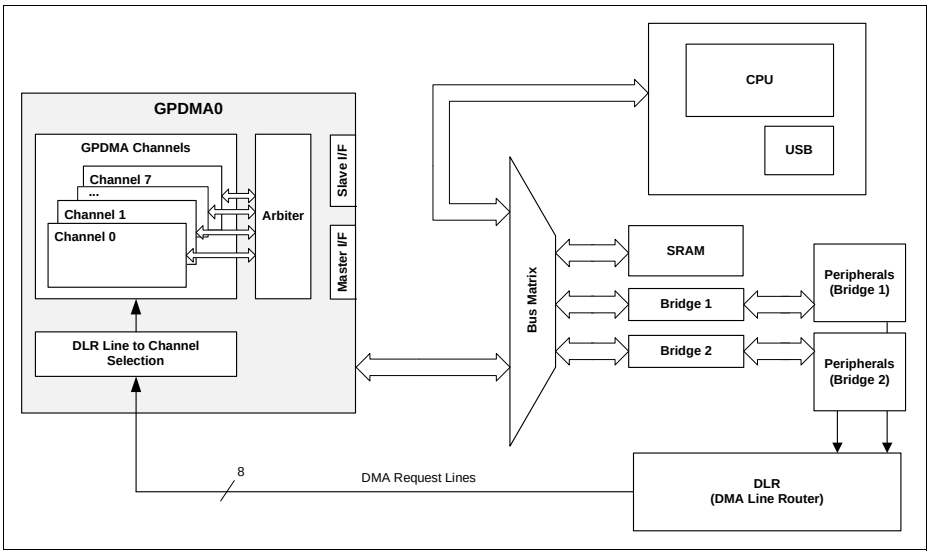


Figure 5-1 GPDMA Block Diagram

5.2 Functional Description

This chapter describes the functional details of the GPDMA.

5.2.1 Terminology

The following terms are concise definitions of the DMA concepts are used throughout this chapter:

Service Partner Terms

- **Source peripheral** - Device from which the GPDMA reads data; the GPDMA then stores the data in the channel FIFO. The source peripheral teams up with a destination peripheral to form a channel.
- **Destination peripheral** - Device to which the GPDMA writes the stored data from the FIFO (previously read from the source peripheral).
- **Memory** - Source or destination that is always "ready" for a DMA transfer and does not require a handshaking interface to interact with the GPDMA. Note that
- **Channel** - Read/write data path between a source peripheral and a destination peripheral, that occurs through the channel FIFO. If the source peripheral is not memory, then a source handshaking interface is assigned to the channel. If the destination peripheral is not memory, then a destination handshaking interface is assigned to the channel. Source and destination handshaking interfaces can be assigned dynamically by programming the channel registers.

Interface Terms

- **Master interface** - GPDMA is a master on the AHB, reading data from the source and writing it to the destination over the bus. Each channel has to arbitrate for the master interface.
- **Slave interface** - The AHB interface over which the GPDMA is programmed.
- **Handshaking interface** - A set of signals or software registers that conform to a protocol and handshake between the GPDMA and source or destination peripheral in order to control transferring a single or burst transaction between them. This interface is used to request, acknowledge, and control a GPDMA transaction. A channel can receive a request through one of two types of handshaking interface: software, or peripheral trigger.
  - **Software handshaking interface**- Software uses registers to control transferring a single or burst transaction between the GPDMA and the source or destination peripheral. This mode is useful if the total block size is unknown at the beginning of a transfer. For more information about this interface, refer to **Section 5.2.4.2**.
  - **Peripheral trigger interface** - In this mode, a DLR service request line is used to trigger single or burst transactions. For using this mode, the total block size must be known at the beginning of a transfer. For more information about this interface, refer to **Section 5.2.4**.

Flow Control Terms

- **Flow controller** - Device that determines the length of a DMA block transfer and terminates it.
  - If you know the length of a block before enabling the channel, then GPDMA should be programmed as the flow controller.
  - If the length of a block is not known prior to enabling the channel, the source or destination peripheral needs to control and terminate a transfer. In this mode, the peripheral, using the software handshaking interface, is the flow controller.
- **Flow control mode** (**CFG**.FCMODE) - Special mode that only applies when the destination peripheral is the flow controller. It controls the data pre-fetching from the source peripheral.

Transfer Terms

- **Transfer hierarchy** - **Figure 5-2** illustrates the hierarchy between GPDMA transfers, block transfers, transactions (single or burst), and AHB transfers (single or burst) for peripherals. **Figure 5-3** shows the transfer hierarchy for memory.

Note: For memory type transfers, there is no "Transaction Level".

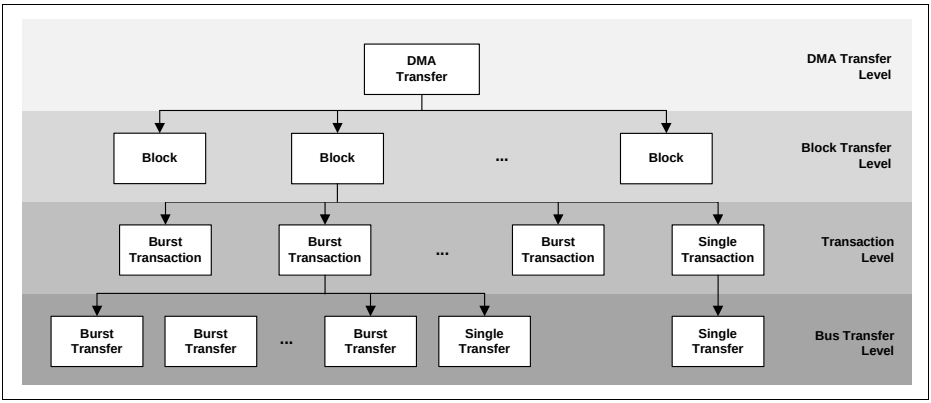


Figure 5-2 Transfer Hierarchy for Peripherals

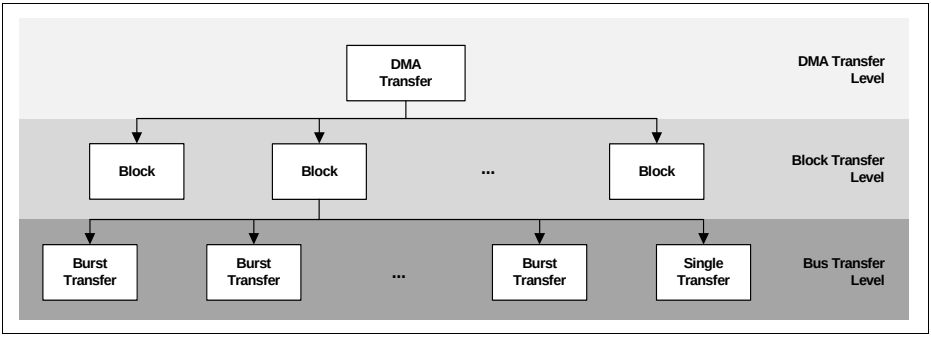


Figure 5-3 Transfer Hierarchy for Memory

- **DMA transfer** - Can be programmed to single or multiple blocks (depends on channel features). Once a DMA transfer has finished, the hardware within the GPDMA disables the channel and can generate an interrupt to signal the DMA transfer completion. You can then reprogram the channel for a new DMA transfer.
  - **Single-block DMA transfer** - Consists of a single block.
  - **Multi-block DMA transfer** - DMA transfer may consist of multiple GPDMA blocks. Multi-block DMA transfers are supported through **Linked lists (block chaining)**, **Auto-reloading** and **Contiguous blocks**. The source and destination can independently select which method to use.
- **Block** - Block of GPDMA data, the amount of which is the block length and is determined by the flow controller. For transfers between the GPDMA and memory, a block is broken directly into a sequence of bursts and single transfers. For transfers between the GPDMA and a peripheral, a block is broken into a sequence of GPDMA transactions (single and bursts). These are in turn broken into a sequence of AHB transfers.
- **Transaction** - Basic unit of a GPDMA transfer, as determined by either the hardware or software handshaking interface. A transaction is relevant only for transfers between the GPDMA and a source or destination peripheral. There are two types of transactions:
  - **Single transaction** - is always converted to a single AHB transfer.
  - **Burst transaction** - Length of a burst transaction is programmed into the GPDMA. The burst transaction is converted into a sequence of AHB fixed length bursts and AHB single transfers. GPDMA executes each burst transfer by performing incremental bursts that are no longer than the maximum burst size set; the only type of burst in this kind of transaction is incremental. The burst transaction length is under program control and normally bears some relationship to the FIFO sizes in the GPDMA and in the source and destination peripherals.

Specific Transfer Mode Terms

- **Scatter** - Relevant to destination transfers within a block. The destination address is incremented or decremented by a programmed amount when a scatter boundary is reached. The number of AHB transfers between successive scatter boundaries is under software control.
- **Gather** - Relevant to source transfers within a block. The source address is incremented or decremented by a programmed amount when a gather boundary is reached. The number of AHB transfers between successive gather boundaries is under software control.
- **Channel locking** - Software can program a channel to keep the AHB master interface by locking arbitration of the master AHB interface for the duration of a DMA transfer, block, or transaction (single or burst).
- **Bus locking** - Software can program a channel to maintain control of the AHB bus for the duration of a DMA transfer, block, or transaction (single or burst). At minimum, channel locking is asserted during bus locking.
- **FIFO mode** - Special mode to improve bandwidth. When enabled, the channel waits until the FIFO is less than half full to fetch the data from the source peripheral, and waits until the FIFO is greater than or equal to half full in order to send data to the destination peripheral. Because of this, the channel can transfer the data using bursts, which eliminates the need to arbitrate for the AHB master interface in each single AHB transfer. When this mode is not enabled, the channel waits only until the FIFO can transmit or accept a single AHB transfer before it requests the master bus interface.

5.2.2 Variable Definitions

The following variable definitions are used in this chapter:

Source single transaction size in bytes

```
src_single_size_bytes = CTLL.SRC_TR_WIDTH/8
```

Source burst transaction size in bytes

```
src_burst_size_bytes = CTLL.SRC_MSIZE * src_single_size_bytes
```

Destination single transaction size in bytes

```
dst_single_size_bytes = CTLL.DST_TR_WIDTH/8
```

Destination burst transaction size in bytes

```
dst_burst_size_bytes = CTLL.DEST_MSIZE * dst_single_size_bytes
```

Block size in bytes

- GPDMA is the flow controller:
  With the GPDMA as the flow controller, the processor programs the GPDMA with the number of data items (block size) of source transfer width (**CTL**.SRC_TR_WIDTH) to be transferred by the GPDMA in a block transfer; this is programmed into the **CTL**.BLOCK_TS field. Therefore, the total number of bytes to be transferred in a block is defined by:
  `blk_size_bytes_dma = CTL.BLOCK_TS * src_single_size_bytes`

- Source peripheral is block flow controller:
  `blk_size_bytes_src = (Number of source burst transactions in block * src_burst_size_bytes) + (Number of source single transactions in block * src_single_size_bytes)`

- Destination peripheral is block flow controller:
  `blk_size_bytes_dst = (Number of destination burst transactions in block * dst_burst_size_bytes) + (Number of destination single transactions in block * dst_single_size_bytes)`

*Note: In the above equations, references to **CTL**.SRC_MSIZE, **CTL**.DEST_MSIZE, **CTL**.SRC_TR_WIDTH, and **CTL**.DST_TR_WIDTH refer to the decoded values of the parameters; for example, **CTL**.SRC_MSIZE = 001_B decodes to 4, and **CTL**.SRC_TR_WIDTH = 010_B decodes to 32 bits.*

5.2.3 Flow Controller and Transfer Type

The device that controls the length of a block is known as the flow controller. Either the GPDMA, the source peripheral, or the destination peripheral must be assigned as the flow controller.

- If the block size is known prior to when the channel is enabled, then the GPDMA must be programmed as the flow controller. The block size is programmed into the **CTL**.BLOCK_TS field.
- If the block size is unknown when the GPDMA channel is enabled, either the source or destination peripheral must be the flow controller.

Attention: If a peripheral is assigned as the flow controller then hardware handshaking is not supported.

The **CTL**.TT_FC field indicates the transfer type and flow controller for that channel.

Table 5-2 lists valid transfer types and flow controller combinations.

Table 5-2 Transfer Type, Flow Control and Handshake Combinations

Transfer Type	Flow Controller	Handshaking
Memory to Memory	GPDMA	-
Memory to Peripheral	GPDMA	Hardware or Software
Peripheral to Memory	GPDMA	Hardware or Software
Peripheral to Peripheral	GPDMA	Hardware or Software
Peripheral to Memory	Peripheral	Software
Peripheral to Peripheral	Source Peripheral	Software
Memory to Peripheral	Peripheral	Software
Peripheral to Peripheral	Destination Peripheral	Software

5.2.4 Handshaking Interface

Handshaking interfaces are used at the transaction level to control the flow of single or burst transactions. The operation of the handshaking interface depends on whether the peripheral or the GPDMA is the flow controller.

The peripheral uses the handshaking interface to indicate to the GPDMA that it is ready to transfer data over the AHB bus.

A peripheral can request a DMA transaction through the GPDMA using one of two types of handshaking interfaces:

- Hardware
- Software

The user selects between the hardware or software handshaking interface on a per-channel basis. Software handshaking is accomplished through memory-mapped registers, while hardware handshaking is accomplished using a dedicated handshaking interface.

Notes

1. *Throughout the remainder of this chapter, references to both source and destination hardware handshaking interfaces assume an active-high interface (refer to **CFG**.SRC(DST)_HS_POL bits in the Channel Configuration register, **CFG**). When active-low handshaking interfaces are used, then the active level and edge are reversed from that of an active-high interface.*
2. *Source and destination peripherals can independently select the handshaking interface type; that is, hardware or software handshaking. For more information, refer to the **CFG**.HS_SEL_SRC and **CFG**.HS_SEL_DST parameters in the **CFG** register.*

5.2.4.1 Hardware Handshaking

Before the transfer can begin the GPDMA and DLR units must be set up according to the user requirements (shown as step 1 in **Figure 5-4**).

Once the peripheral (source or destination) is ready for a transaction it sends a service request. This request is taken by the DLR which in turn forwards it to the GPDMA (step 2). The GPDMA finally executes the transaction (step 3).

Steps 2 and 3 repeat until the programmed transfer is complete.

*Note: Optionally interrupts can be generated after block or transaction completion as described in **Section 5.5***

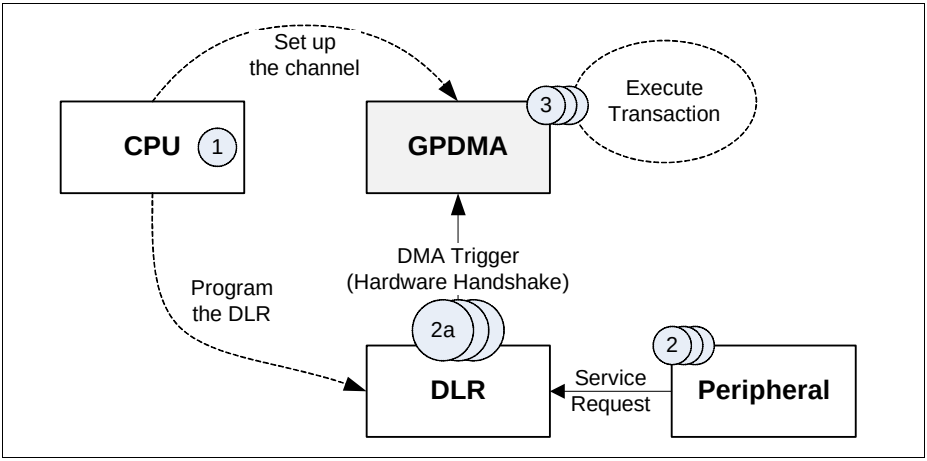


Figure 5-4 Hardware Handshaking Interface

5.2.4.2 Software Handshaking

Before the transfer can begin the GPDMA and NVIC units must be set up according to the user requirements (shown as step 1 in **Figure 5-5**).

Once the peripheral (source or destination) is ready for a transaction it sends a service request to the CPU. The interrupt service routine then uses the software registers, detailed in **Section 5.8.4**, to initiate and control a DMA transaction (step 2). The GPDMA finally executes the transaction (step 3).

Steps 2 and 3 repeat until the programmed transfer is complete.

*Note: Optionally interrupts can be generated after block or transaction completion as described in **Section 5.5***

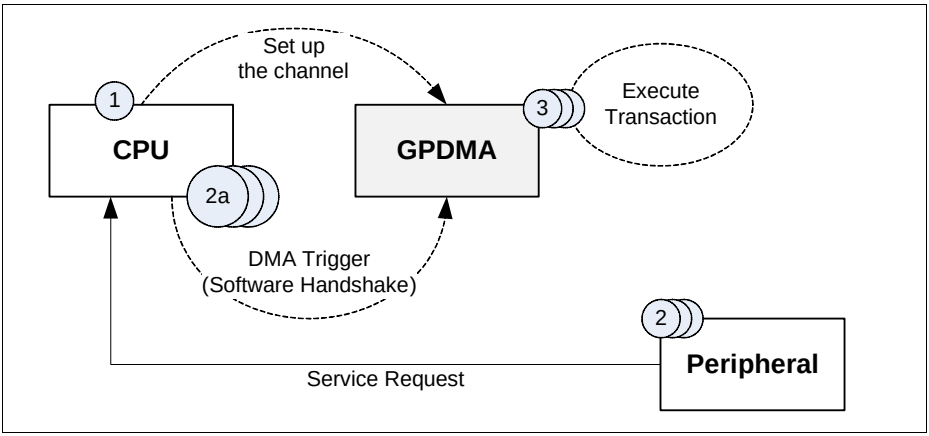


Figure 5-5 Software Handshaking Interface

5.2.4.3 Handshaking with GPDMA as Flow Controller

The GPDMA tries to efficiently transfer the data using as little of the bus bandwidth as possible. Generally, the GPDMA tries to transfer the data using burst transactions and, where possible, fill or empty the channel FIFO in single bursts - provided that the software has not limited the burst length.

The GPDMA can also lock the arbitration for the master bus interface so that a channel is permanently granted the master bus interface. Additionally, the GPDMA can assert the lock signal to lock the system arbiter. For more information, refer to **Section 5.2.6**.

Before describing the handshaking interface operation, the following sections define the terms "Single Transaction Region" and "Early-Terminated Burst Transaction".

Single Transaction Region

The single transaction region is the time interval where the GPDMA can no longer use full burst transactions to complete the block transfer.

There are cases where a DMA block transfer cannot be completed using only burst transactions. Typically this occurs when the block size is not a multiple of the burst transaction length. In these cases, the block transfer uses burst transactions up to the point where the amount of data left to complete the block is less than the amount of data in a burst transaction. At this point, the GPDMA completes the block transfer using single or early-terminated burst transactions.

Early-Terminated Burst Transaction

When a source or destination peripheral is in the Single Transaction Region, a burst transaction can still be requested when using Software Handshaking. In this case, the burst transaction is started and "early-terminated" at block completion without transferring the programmed amount of data, that is, src_burst_size_bytes or dst_burst_size_bytes, but only the amount required to complete the block transfer.

Hardware Handshaking

Works as described above in **Chapter 5.2.4.1**.

Software Handshaking

When the GPDMA is the flow controller, then the last transaction registers - **LSTSRCREG** and **LSTDSTREG** - are not used, and the values in these registers are ignored.

- Operation - Peripheral Not In Single Transaction Region

Writing a 1 to the **REQSRCREG**[x], **REQDSTREG**[x] bit fields is always interpreted as a burst transaction request, where x is the channel number. However, in order for a burst transaction request to start, software must write a 1 to the **SGLREQSRCREG**[x], **SGLREQDSTREG**[x] register.

You can write a 1 to the SGLREQSRCREG[x], SGLREQDSTREG[x] and REQSRCREG[x], REQDSTREG[x] registers in any order, but both registers must be asserted in order to initiate a burst transaction. Upon completion of the burst transaction, the hardware clears the SGLREQSRCREG[x], SGLREQDSTREG[x] and REQSRCREG[x], REQDSTREG[x] registers.

- Operation - Peripheral In Single Transaction Region

Writing a 1 to the **SGLREQSRCREG**, **SGLREQDSTREG** initiates a single transaction. Upon completion of the single transaction, both the SGLREQSRCREG, SGLREQDSTREG and **REQSRCREG**, **REQDSTREG** bits are cleared by hardware. Therefore, writing a 1 to the REQSRCREG, REQDSTREG is ignored while a single transaction has been initiated, and the requested burst transaction is not serviced.

Again, writing a 1 to the REQSRCREG, REQDSTREG register is always a burst transaction request. However, in order for a burst transaction request to start, the corresponding channel bit in the SGLREQSRCREG, SGLREQDSTREG must be asserted. Therefore, to ensure that a burst transaction is serviced in this region, you must write a 1 to the REQSRCREG, REQDSTREG before writing a 1 to the SGLREQSRCREG, SGLREQDSTREG register. If the programming order is reversed, a single transaction is started instead of a burst transaction. The hardware clears both the REQSRCREG, REQDSTREG and the SGLREQSRCREG, SGLREQDSTREG registers after the burst transaction request completes. When a burst transaction is initiated in the

Single Transaction Region, then the block completes using an Early-Terminated Burst Transaction.

Software can poll the relevant channel bit in the SGLREQSRCREG, SGLREQDSTREG and REQSRCREG, REQDSTREG registers. When both are 0, then either the requested burst or single transaction has completed. Alternatively, the IntSrcTran or IntDstTran interrupts can be enabled and unmasked in order to generate an interrupt when the requested source or destination transaction has completed.

Note: The transaction-complete interrupts are triggered when both single and burst transactions are complete. The same transaction-complete interrupt is used for both single and burst transactions.

5.2.4.4 Handshaking with Peripheral as Flow Controller

When the peripheral is the flow controller, it controls the length of the block and must communicate to the GPDMA when the block transfer is complete. The peripheral does this by telling the GPDMA that the current transaction - burst or single - is the last transaction in the block. When the peripheral is the flow controller and the block size is not a multiple of the **CTL**.SRC_MSIZE, **CTL**.DEST_MSIZE, then the peripheral must use single transactions to complete a block transfer.

Note: Since the peripheral can terminate the block on a single transaction, there is no notion of a Single Transaction Region such as there is when the GPDMA is the flow controller.

When the peripheral is the flow controller, it indicates to the GPDMA which type of transaction - single or burst - to perform by using **Software Handshaking**. Where possible, the GPDMA uses the maximum possible burst length. It can also lock the arbitration for the master bus so that a channel is permanently granted the master bus interface. The GPDMA can also assert the hlock signal to lock the system arbiter. For more information, refer to **Section 5.2.6**.

Hardware Handshaking

This mode is not supported in the XMC4[12]00.

Software Handshaking

Writing a 1 to the Source/Destination Software Transaction Request initiates a transaction (refer to **REQSRCREG** and **REQDSTREG**, respectively). The type of transaction - single or burst - depends on the state of the corresponding channel bit in the Single Source/Destination Transaction Request register (refer to **SGLREQSRCREG** or **SGLREQDSTREG**, respectively)

If SGLREQSRCREG[n], SGLREQDSTREG[n] = 1 when a 1 is written to the REQSRCREG[n], REQDSTREG[n] register, this means that software is requesting a single transaction on channel n, or a burst transaction otherwise.

The request is the last in the block if the corresponding channel bit in the Last Source/Destination Request register is asserted; refer to **LSTSRCREG** and **LSTDSTREG**, respectively.

If LSTSRCREG[n], LSTDSTREG[n] = 1 when a 1 is written to the REQSRCREG[n], REQDSTREG[n] register, this means that software is requesting that this transaction is the last transaction in the block. The **SGLREQSRCREG**, **SGLREQDSTREG** and **LSTSRCREG**, **LSTDSTREG** registers must be written to before the **REQSRCREG**, **REQDSTREG** registers.

On completion of the transaction - single or burst - the relevant channel bit in the **REQSRCREG**, **REQDSTREG** register is cleared by hardware. Software can therefore poll this bit in order to determine when the requested transaction has completed. Alternatively, the IntSrcTran or IntDstTran interrupts can be enabled and unmasked in order to generate an interrupt when the requested transaction - single or burst - has completed.

When the peripheral is the flow controller and the block size is not a multiple of the **CTL**.SRC_MSIZE, **CTL**.DEST_MSIZE, then software must use single transactions to complete the block transfer.

5.2.5 FIFO Usage

Each channel has a source state machine and destination state machine running in parallel. These state machines generate the request inputs to the arbiter, which arbitrates for the master bus interface (one arbiter per master bus interface).

When the source/destination state machine is granted control of the master bus interface, then AHB transfers between the peripheral and the GPDMA (on behalf of the granted state machine) can take place.

AHB transfers from the source peripheral or to the destination peripheral cannot proceed until the channel FIFO is ready. For burst transaction requests and for transfers involving memory peripherals, the criterion for "FIFO readiness" is controlled by the FIFO_MODE field of the **CFG** register.

The definition of FIFO readiness is the same for:

- Single transactions
- Burst transactions, where **CFG**.FIFO_MODE = 0
- Transfers involving memory peripherals, where **CFG**.FIFO_MODE = 0

The channel FIFO is deemed ready when the space/data available is sufficient to complete a single AHB transfer of the specified transfer width. FIFO readiness for source transfers occurs when the channel FIFO contains enough room to accept at least a single transfer of **CTL**.SRC_TR_WIDTH width. FIFO readiness for destination transfers occurs when the channel FIFO contains data to form at least a single transfer of **CTL**.DST_TR_WIDTH width.

*Note: An exception to FIFO readiness for destination transfers occurs in "FIFO flush mode" In this mode, FIFO readiness for destination transfers occurs when the channel FIFO contains data to form at least a single transfer of **CTL**.SRC_TR_WIDTH width (and not **CTL**.DST_TR_WIDTH width, as is the normal case).*

When **CFG**.FIFO_MODE = 1, then the criteria for FIFO readiness for burst transaction requests and transfers involving memory peripherals are as follows:

- A FIFO is ready for a source burst transfer when the FIFO is less than half empty.
- A FIFO is ready for a destination burst transfer when the FIFO is greater than or equal to half full.

Exceptions to this "readiness" occur. During these exceptions, a value of **CTL**. FIFO_MODE = 0 is assumed. The following are the exceptions:

- Near the end of a burst transaction or block transfer - The channel source state machine does not wait for the channel FIFO to be less than half empty if the number of source data items left to complete the source burst transaction or source block transfer is less than FIFO DEPTH/2. Similarly, the channel destination state machine does not wait for the channel FIFO to be greater than or equal to half full, if the number of destination data items left to complete the destination burst transaction or destination block transfer is less than FIFO DEPTH/2.
- In FIFO flush mode
- When a channel is suspended - The destination state machine does not wait for the FIFO to become half empty to flush the FIFO, regardless of the value of the FIFO_MODE field.

When the source/destination peripheral is not memory, the source/destination state machine waits for a single/burst transaction request. Upon receipt of a transaction request and only if the channel FIFO is "ready" for source/destination AHB transfers, a request for the master bus interface is made by the source/destination state machine.

*Note: There is one exception to this, which occurs when the destination peripheral is the flow controller and **CFG**.FCMODE = 1 (data pre-fetching is disabled). Then the source state machine does not generate a request for the master bus interface (even if the FIFO is "ready" for source transfers and has received a source transaction request) until the destination requests new data.*

When the source/destination peripheral is memory, the source/destination state machine must wait until the channel FIFO is "ready". A request is then made for the master bus interface. There is no handshaking mechanism employed between a memory peripheral and the GPDMA.

5.2.6 Bus and Channel Locking

It is possible to program the GPDMA for:

- Bus locking

- Channel locking - Locks the arbitration for the AHB master interface, which grants ownership of the master bus interface to one of the requesting channel state machines (source or destination).

Bus and channel locking can proceed for the duration of a DMA transfer, a block transfer, or a single or burst transaction.

Bus Locking

If the LOCK_B bit in the channel configuration register (**CFG**) is set, then the AHB bus is locked for the duration specified in the LOCK_B_L field.

Channel Locking

If the LOCK_CH field is set, then the arbitration for the master bus interface is exclusively reserved for the source and destination peripherals of that channel for the duration specified in the LOCK_CH_L field.

If bus locking is activated for a certain duration, then it follows that the channel is also automatically locked for that duration. Three cases arise:

- **CFG**.LOCK_B = 0 - Programmed values of **CFG**.LOCK_CH and **CFG**.LOCK_CH_L are used.
- **CFG**.LOCK_B = 1 and **CFG**.LOCK_CH = 0 - DMA transfer proceeds as if **CFG**.LOCK_CH = 1 and **CFG**.LOCK_CH_L = **CFG**.LOCK_B_L. The programmed values of **CFG**.LOCK_CH and **CFG**.LOCK_CH_L are ignored.
- **CFG**.LOCK_B = 1 and **CFG**.LOCK_CH = 1 - Two cases arise:
  - **CFG**.LOCK_B_L <= **CFG**.LOCK_CH_L - In this case, the DMA transfer proceeds as if **CFG**.LOCK_CH_L = **CFG**. LOCK_B_L and the programmed value of **CFG**.LOCK_CH_L is ignored. Thus, if bus locking is enabled over the DMA transfer level, then channel locking is enabled over the DMA transfer level, regardless of the programmed value of **CFG**.LOCK_CH_L.
  - LOCK_B_L > **CFG**.LOCK_CH_L - The programmed value of **CFG**.LOCK_CH_L is used. Thus, if bus locking is enabled over the DMA block transfer level and channel locking is enabled over the DMA transfer level, then channel locking is performed over the DMA transfer level.

Locking Levels

If locking is enabled for a channel, then locking of the AHB master bus interface at a programmed locking transfer level is activated when the channel is first granted the AHB master bus interface at the start of that locking transfer level. It continues until the locking transfer level has completed; that is, if channel 0 has enabled channel level locking at the block transfer level, then this channel locks the master bus interface when it is first granted the master bus interface at the start of the block transfer, and continues to lock the master bus interface until the block transfer has completed.

Source and destination block transfers occur successively in time, and a new source block cannot commence until the previous destination block has completed.

Block and DMA transfer level locking are both terminated on completion of the block or DMA transfer to the destination.

Transaction-level locking is different due to the fact that source and destination transactions occur independently in time, and the number of source and destination transactions in a DMA block or DMA transfer do not have to match. Transaction-level locking is cleared at the end of a source or destination transaction only if the opposing peripheral is not currently in the middle of a transaction.

If channel-level or bus-level locking is enabled for a channel at the transaction level, and either the source or destination of the channel is a memory device, then the locking is ignored and the channel proceeds as if locking (bus or channel) is disabled.

Note: Since there is no notion of a transaction level for a memory peripheral, then transaction-level locking is not allowed when either source or destination is memory.

5.2.7 Scatter/Gather

Scatter is relevant to a destination transfer. The destination address is incremented or decremented by a programmed amount - the scatter increment - when a scatter boundary is reached. **Figure 5-6** shows an example destination scatter transfer. The destination address is incremented or decremented by the value stored in the destination scatter increment (DSRx.DSI) field (refer to **DSR**), multiplied by the number of bytes in a single AHB transfer to the destination s (decoded value of **CTL**.DST_TR_WIDTH)/8 - when a scatter boundary is reached. The number of destination transfers between successive scatter boundaries is programmed into the Destination Scatter Count (DSC) field of the DSR register.

Scatter is enabled by writing a 1 to the **CTL**.DST_SCATTER_EN field. The **CTL**.DINC field determines if the address is incremented, decremented, or remains fixed when a scatter boundary is reached. If the **CTL**.DINC field indicates a fixed-address control throughout a DMA transfer, then the **CTL**.DST_SCATTER_EN field is ignored, and the scatter feature is automatically disabled.

Gather is relevant to a source transfer. The source address is incremented or decremented by a programmed amount when a gather boundary is reached. The number of source transfers between successive gather boundaries is programmed into the Source Gather Count (SGRx.SGC) field. The source address is incremented or decremented by the value stored in the source gather increment (SGRx.SGI) field (refer to **SGR**), multiplied by the number of bytes in a single AHB transfer from the source - (decoded value of **CTL**.SRC_TR_WIDTH)/8 - when a gather boundary is reached.

Gather is enabled by writing a 1 to the **CTL**.SRC_GATHER_EN field. The **CTL**.SINC field determines if the address is incremented, decremented, or remains fixed when a

gather boundary is reached. If the **CTL**.SINC field indicates a fixed-address control throughout a DMA transfer, then the **CTL**.SRC_GATHER_EN field is ignored, and the gather feature is automatically disabled.

Note: For multi-block transfers, the counters that keep track of the number of transfers left to reach a gather/scatter boundary are re-initialized to the source gather count (SGRx.SGC) and destination scatter count (DSC), respectively, at the start of each block transfer.

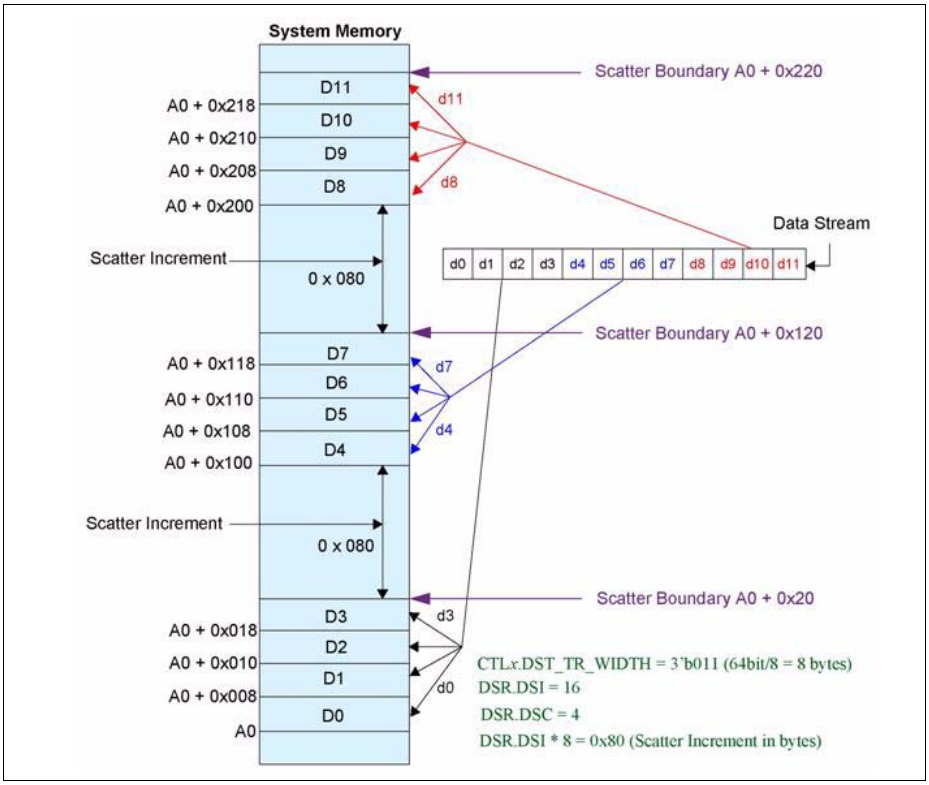


Figure 5-6 Example of Destination Scatter Transfer

As an example of gather increment, consider the following:

```
SRC_TR_WIDTH = 3'b010 (32 bits)
SGR.SGC = 0x04 (source gather count)
CTL.SRC_GATHER_EN = 1 (source gather enabled)
SAR = A0 (starting source address)
```

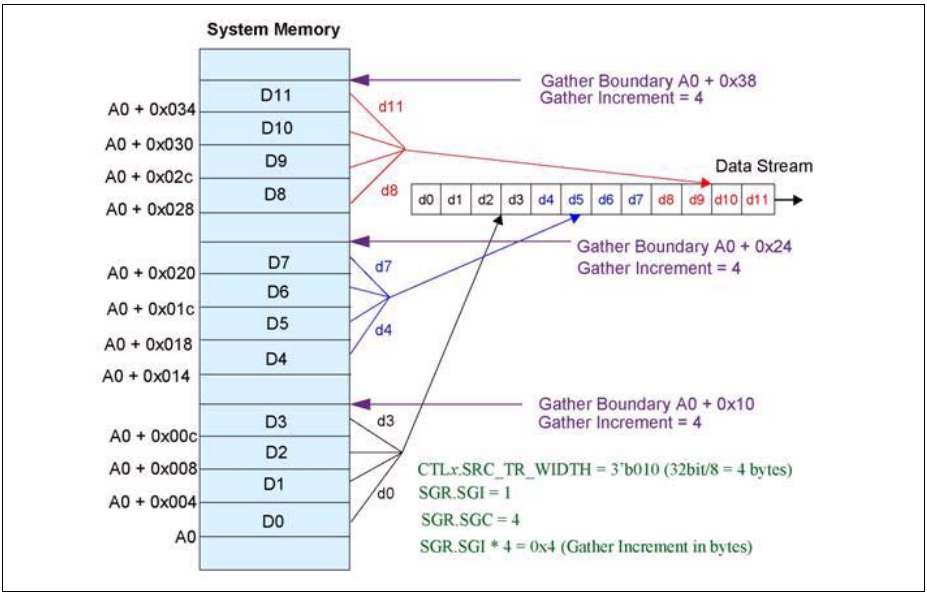


Figure 5-7 Source Gather when SGR.SGI = 0x1

In general, if the starting address is A0 and **CTL**.SINC = 00_B (increment source address control), then the transfer will be:

```
A0, AO + TWB, A0 + 2*TWB                (A0 + (SGR.SGC-1)*TWB)
<-scatter_increment-> (A0 + (SGR.SGC*TWB) + (SGR.SGI *TWB))
```

where TWB is the transfer width in bytes, decoded value of **CTL**.SRC_TR_WIDTH/8 = src_single_size_bytes.

5.2.8 Abnormal Transfer Termination

A GPDMA DMA transfer may be terminated abruptly by software by clearing the channel enable bit, **CHENREG**.CH_EN or by clearing the global enable bit in the GPDMA Configuration Register (**DMACFGREG**[0]).

If a transfer is in progress while a channel is disabled, abnormal transfer termination and data corruption occurs. Also the transfer acknowledge may be lost. Therefore this must be avoided.

Attention: *Disabling a channel via software prior to completing a transfer is not supported.*

5.3 Basic Transfers

From a users perspective DMA transfers can be grouped into

- software triggered transfers and
- hardware triggered transfers.

The setup procedure for both kinds of transfers is identical to a large extent and is described in more detail later in this section after highlighting the differences of the trigger types.

Software triggered transfers

are set up as memory-to-memory types and start automatically when the channel is enabled. After transfer completion the channel is disabled. There is no way to trigger the transactions by on-chip hardware.

Hardware triggered transfers

are set up as peripheral-to-memory, memory-to-peripheral or peripheral-to-peripheral types. Additionally the trigger source, signal routing and trigger generation must be programmed. Details on trigger generation are found n the “Service Request Generation” section of each peripherals chapter. Signal routing options (ERU) and trigger generation (DLR) are described in the “Service Request Processing” chapter.

GPDMA set up

Transfers are set up by programming fields of the **CTL** and **CFG** registers for that channel. As shown in **Figure 5-2**, a single block is made up of numerous transactions - single and burst - which are in turn composed of AHB transfers.

*Note: There are references to software parameters throughout this chapter. The software parameters are the field names in each register description table and are prefixed by the register name; for example, the Block Transfer Size field in the Control Register is designated as "**CTL**.BLOCK_TS."*

Table 5-3 lists the parameters that are investigated in the following examples. The effects of these parameters on the flow of the block transfer are highlighted.

Table 5-3 Parameters Used in Transfer Examples

Parameter	Description
CTL.TT_FC	Transfer type and flow control
CTL.BLOCK_TS	Block transfer size
CTL.SRC_TR_WIDTH	Source transfer width
CTL.DST_TR_WIDTH	Destination transfer width

Table 5-3 Parameters Used in Transfer Examples (cont'd)

Parameter	Description
CTL.SRC_MSIZE	Source burst transaction length
CTL.DEST_MSIZE	Destination burst transaction length
CFG.MAX_ABRST	Maximum AMBA burst length
CFG.FIFO_MODE	FIFO mode select
CFG.FCMODE	Flow-control mode

The GPDMA is programmed with the number of data items that are to be transferred for each burst transaction request, **CTL**.SRC_MSIZE and **CTL**.DEST_MSIZE. Similarly, the width of each data item in the transaction is set by the **CTL**.SRC_TR_WIDTH and **CTL**.DST_TR_WIDTH fields.

5.3.1 Block transfer with GPDMA as the flow controller

Table 5-4 lists the DMA parameters for this example (the FIFO depth is taken as 16 bytes).

Table 5-4 Parameters in Transfer Operation

Parameter	Description
CTL.TT_FC = 011_B	Peripheral-to-peripheral transfer with GPDMA as flow controller
CTL.BLOCK_TS = 12	-
CTL.SRC_TR_WI DTH = 010_B	32 bits
CTL. DST_TR_WI DTH = 010_B	32 bits
CTL.SRC_MSIZE = 001_B	Source burst transaction length = 4
CTL.DEST_MSIZE = 001_B	Destination burst transaction length = 4
CFG.MAX_ABRST = 0_B	No limit on maximum AMBA burst length

A total of 48 bytes are transferred in the block (that is blk_size_bytes_dma = 48). As shown in **Figure 5-8**, this block transfer consists of three bursts of length 4 from the source, interleaved with three bursts, again of length 4, to the destination.

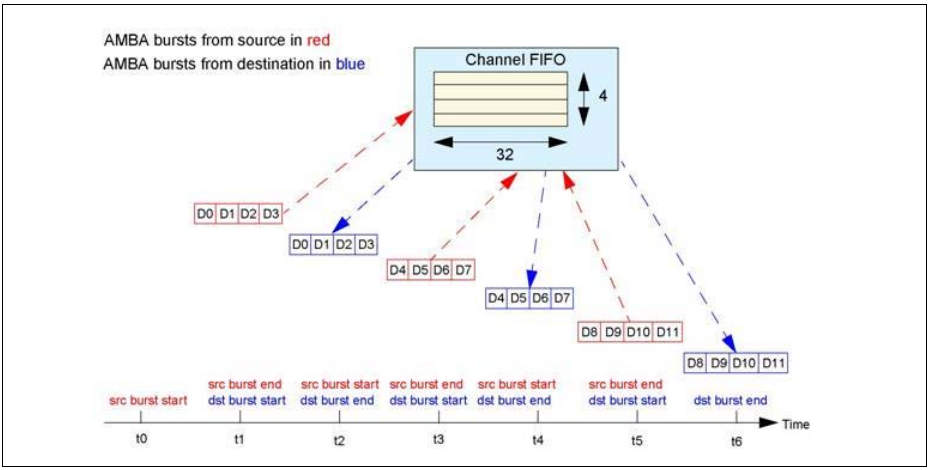


Figure 5-8 Breakdown of Block Transfer

The channel FIFO is alternatively filled by a burst from the source and emptied by a burst to the destination until the block transfer has completed, as shown in **Figure 5-9**.

D3	Empty	D7	Empty	D11	Empty
D2	Empty	D6	Empty	D10	Empty
D1	Empty	D5	Empty	D9	Empty
D0	Empty	D4	Empty	D8	Empty
Time t1	Time t2	Time t3	Time t4	Time t5	Time t6

Figure 5-9 Channel FIFO Contents

Burst transactions are completed in one burst. Additionally neither the source or destination peripherals enter their Single Transaction Region at any stage throughout the DMA transfer, and the block transfer from the source and to the destination consists of burst transactions only.

5.3.2 Effect of maximum AMBA burst length on a block transfer

If the **CFG**.MAX_ABRST = 2 parameter and all other parameters are left unchanged from previous example, then the block transfer would look like that shown in **Figure 5-10**.

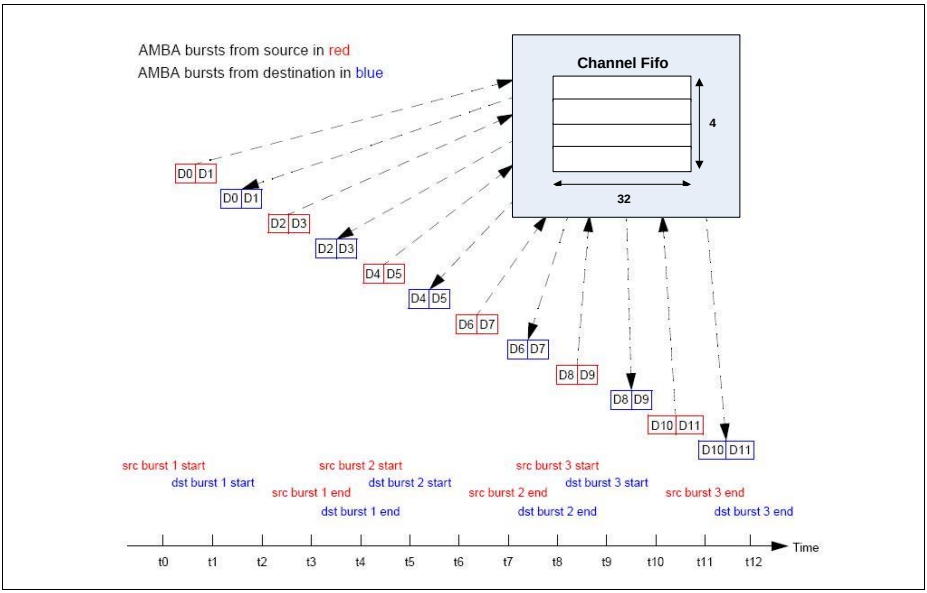


Figure 5-10 Breakdown of Block Transfer where max_abrst = 2, Case 1

The channel FIFO is alternatively half filled by a burst from the source, and then emptied by a burst to the destination until the block transfer has completed; this is illustrated in **Figure 5-11**.

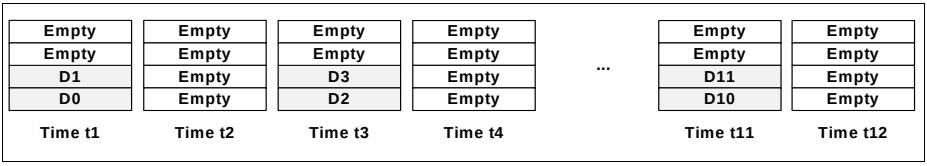


Figure 5-11 Channel FIFO Contents

In this example block transfer, each source or destination burst transaction is made up of two bursts, each of length 2. As **Figure 5-11** illustrates, the top two channel FIFO locations are redundant for this block transfer. However, this is not the general case. The block transfer could proceed as indicated in **Figure 5-12**.

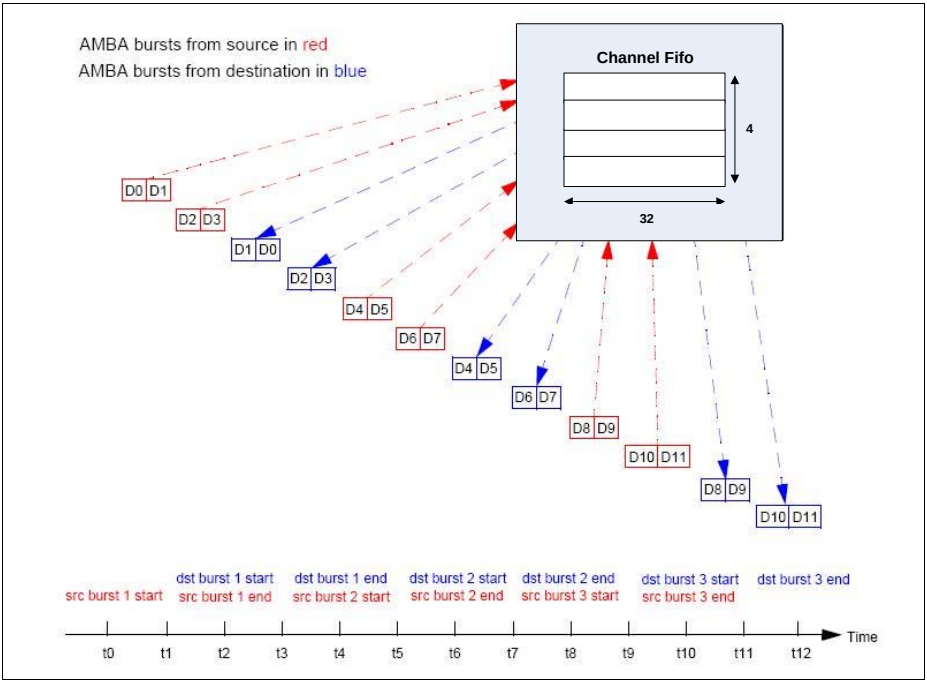


Figure 5-12 Breakdown of Block Transfer where max_abrst = 2, Case 2

This depends on the timing of the source and destination transaction requests, relative to each other. **Figure 5-13** illustrates the channel FIFO status for **Figure 5-12**.

D3	**Empty**	**D7**	**Empty**	**D11**	**Empty**
D2	**Empty**	**D6**	**Empty**	**D10**	**Empty**
D1	**Empty**	**D5**	**Empty**	**D9**	**Empty**
D0	**Empty**	**D4**	**Empty**	**D8**	**Empty**
Time t2	**Time t4**	**Time t6**	**Time t8**	**Time t10**	**Time t12**

Figure 5-13 Channel FIFO Contents

Recommendation

To allow a burst transaction to complete in a single burst, the following should be true:

```
CFGL.MAX_ABRST >= max(src_burst_size_bytes,
                      dst_burst_size_bytes)
```

Adhering to the above recommendation results in a reduced number of bursts per block, which in turn results in improved bus utilization and lower latency for block transfers.

Limiting a burst to a maximum length prevents the GPDMA from saturating the AHB bus when the system arbiter is configured to only allow changing of the grant signals to bus masters at the end of an undefined length burst. It also prevents a channel from saturating a GPDMA master bus interface.

5.4 Multi Block Transfers

A DMA transfer may consist of

- single block transfer, supported by all channels.
- multi-block transfers, supported by channels 0 and 1 of GPDMA0.

On successive blocks of a multi-block transfer, the **SAR**, **DAR** register in the GPDMA is reprogrammed using either of the following methods:

- Block chaining using linked lists
- Auto-reloading
- Contiguous address between blocks

On successive blocks of a multi-block transfer, the **CTL** register in the GPDMA is reprogrammed using either of the following methods:

- Block chaining using linked lists
- Auto-reloading

When block chaining, using Linked Lists is the multi-block method of choice. On successive blocks, the **LLP** register in the GPDMA is reprogrammed using block chaining with linked lists.

A block descriptor consists of six registers: **SAR**, **DAR**, **LLP**, **CTL**, **SSTAT** and **DSTAT**. The first four registers, along with the **CFG** register, are used by the GPDMA to set up and describe the block transfer.

Note: The term Link List Item (LLI) and block descriptor are synonymous.

5.4.1 Block Chaining Using Linked Lists

In this case, the GPDMA reprograms the channel registers prior to the start of each block by fetching the block descriptor for that block from system memory. This is known as an LLI update.

GPDMA block chaining uses a Linked List Pointer register (**LLP**) that stores the address in memory of the next linked list item. Each LLI contains the corresponding block descriptors:

1. **SAR**
2. **DAR**
3. **LLP**
4. **CTL**
5. **SSTAT**
6. **DSTAT**

To set up block chaining, you program a sequence of Linked Lists in memory.

LLI accesses are always 32-bit accesses aligned to 32-bit boundaries and cannot be changed or programmed to anything other than 32-bit, even if the AHB master interface of the LLI supports more than a 32-bit data width.

The **SAR**, **DAR**, **LLP**, and **CTL** registers are fetched from system memory on an LLI update. The updated contents of the **CTL**, **SSTAT**, and **DSTAT** registers are optionally written back to memory on block completion. **Figure 5-14** and **Figure 5-15** show how you use chained linked lists in memory to define multi-block transfers using block chaining.

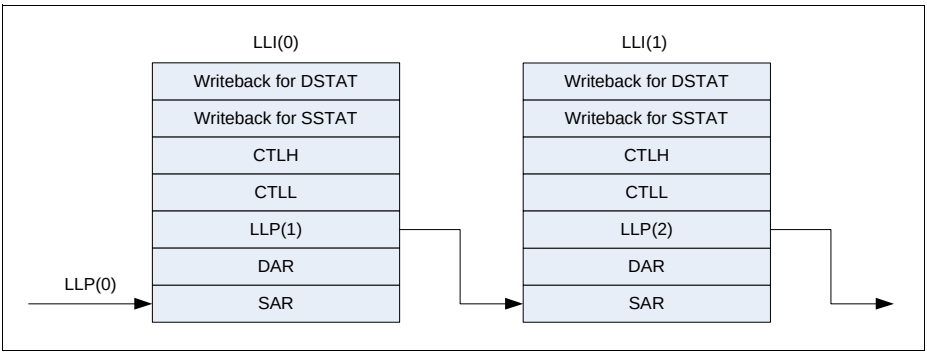


Figure 5-14 Multi-Block Transfer Using Linked Lists When CFG.SS_UPD_EN is set to '1'

It is assumed that no allocation is made in system memory for the source status when the parameter **CFG**.SS_UPD_EN is set to '0'. In this case, then the order of a Linked List item is as follows:

1. **SAR**
2. **DAR**
3. **LLP**
4. **CTL**
5. **DSTAT**

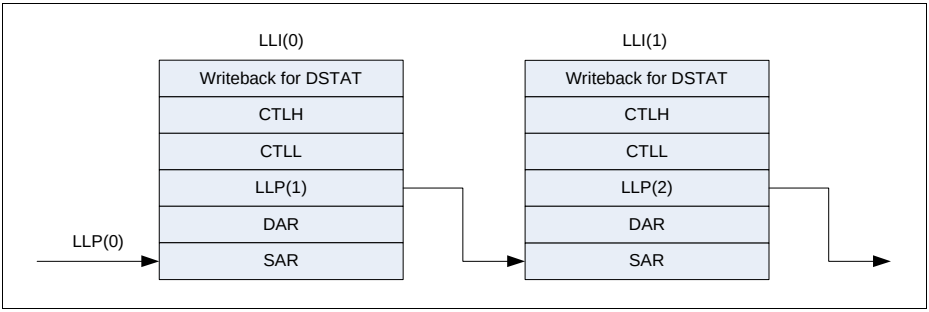


Figure 5-15 Multi-Block Transfer Using Linked Lists When CFG.SS_UPD_EN is set to '0'

*Note: In order to not confuse the **SAR**, **DAR**, **LLP**, **CTL**, **SSTAT** and **DSTAT** register locations of the LLI with the corresponding GPDMA memory mapped register locations, the LLI register locations are prefixed with LLI; that is, LLI.SAR, LLI.DAR, LLI.LLP, LLI.CTLH/L, LLI.SSTATx, and LLI.DSTATx.*

Figure 5-14 and **Figure 5-15** show the mapping of a Linked List Item stored in memory to the channel registers block descriptor.

Rows 6 through 10 of **Table 5-5** show the required values of **LLP**, **CTL**, and **CFG** for multi-block DMA transfers using block chaining.

*Note: For rows 6 through 10 of **Table 5-5**, the LLI.CTLH/L, LLI.LLP, LLI.SAR, and LLI.DAR register locations of the LLI are always affected at the start of every block transfer. The LLI.LLP and LLI.CTLH/L locations are always used to reprogram the GPDMA **LLP** and **CTL** registers. However, depending on the **Table 5-5** row number, the LLI.SAR, LLI.DAR address may or may not be used to reprogram the GPDMA **SAR**, **DAR** registers.*

Table 5-5 Programming of Transfer Types and Channel Register Update Method

Transfer Type	LLP. LOC = 0	CTL.LLP_ SRC EN	CFG.RELOAD_SRC	CTL.LLPDST_EN	CFG.RELOAD_DST	CTL, LLP UpdateMethod	SAR UpdateMethod	DAR UpdateMethod	Write Back
1. Single-block or last transfer of multi-block.	Yes	0	0	0	0	None, user reprograms	None (single)	None (single)	No
2. Auto-reload multi-block transfer with contiguous **SAR**	Yes	0	0	0	1	**CTL**, **LLP** are reloaded from initial values.	Contig uous	Auto-Reload	No
3. Auto-reload multi-block transfer with contiguous **DAR**.	Yes	0	1	0	0	**CTL**, **LLP** are reloaded from initial values	Auto-Reload	Contig uous	No
4. Auto-reload multi-block transfer	Yes	0	1	0	1	**CTL**, **LLP** are reloaded from initial values	Auto-reload	Auto-Reload	No

Table 5-5 Programming of Transfer Types and Channel Register Update Method (cont'd)

Transfer Type	LLP.LOC = 0	CTL.LLP_SRC EN	CFG.RELOAD_SRC	CTL.LLPDST_EN	CFG.RELOAD_DST	CTL, LLP UpdateMethod	SAR UpdateMethod	DAR UpdateMethod	Write Back
5. Single-block or last transfer of multi-block.	No	0	0	0	0	None, user reprograms	None (single)	None (single)	Yes
6. Linked list multi-block transfer with contiguous SAR	No	0	0	1	0	CTL, LLP loaded from next Linked List item.	Contiguous	Linked List	Yes
7. Linked list multi-block transfer with auto-reload SAR	No	0	1	1	0	CTL, LLP loaded from next Linked List item.	Auto-Reload	Linked List	Yes
8. Linked list multi-block transfer with contiguous DAR	No	1	0	0	0	CTL, LLP loaded from next Linked List item.	Linked List	Contiguous	Yes
9. Linked list multi-block transfer with auto-reload DAR	No	1	0	0	1	CTL, LLP loaded from next Linked List item.	Linked List	Auto-Reload	Yes
10. Linked list multi-block transfer	No	1	0	1	0	CTL, LLP loaded from next Linked List item.	Linked List	Linked List	Yes

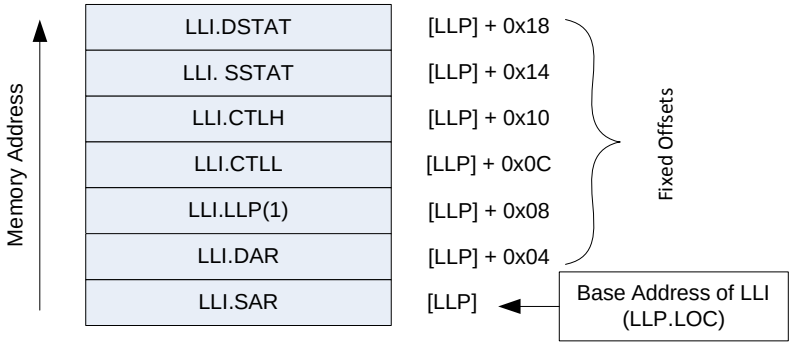


Figure 5-16 Mapping of Block Descriptor (LLI) in Memory to Channel Registers When CFG.SS_UPD_EN = 1

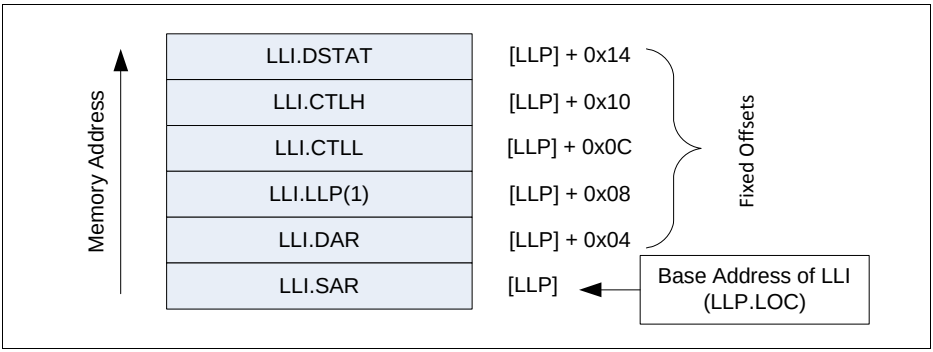


Figure 5-17 Mapping of Block Descriptor (LLI) in Memory to Channel Registers When CFG.SS_UPD_EN = 0

Notes

1. *Throughout this chapter, there are descriptions about fetching the LLI.CTLH/L register from the location pointed to by the **LLP** register. This exact location is the LLI base address (stored in **LLP** register) plus the fixed offset. For example, in **Figure 5-16** the location of the LLI.CTLH/L register is **LLP**.LOC + 0xc.*
2. *Referring to **Table 5-5**, if the Write Back column entry is "Yes" and the channel is 0 or 1, then the CTLH register is always written to system memory (to LLI.CTLH) at the end of every block transfer.*
3. *The source status is fetched and written to system memory at the end of every block transfer if the Write Back column entry is "Yes" and **CFG**.SS_UPD_EN is enabled.*

4. *The destination status is fetched and written to system memory at the end of every block transfer if the Write Back column entry is "Yes" and **CFG**.DS_UPD_EN is enabled.*

5.4.2 Auto-Reloading of Channel Registers

During auto-reloading, the channel registers are reloaded with their initial values at the completion of each block and the new values used for the new block. Depending on the row number in **Table 5-5**, some or all of the **SAR**, **DAR**, and **CTL** channel registers are reloaded from their initial value at the start of a block transfer.

5.4.3 Contiguous Address Between Blocks

In this case, the address between successive blocks is selected as a continuation from the end of the previous block.

Enabling the source or destination address to be contiguous between blocks is a function of the **CTL**.LLP_SRC_EN, **CFG**.RELOAD_SRC, **CTL**.LLP_DST_EN, and **CTL**.RELOAD_DST registers (see **Table 5-5**).

*Note: You cannot select both **SAR** and **DAR** updates to be contiguous. If you want this functionality, you should increase the size of the Block Transfer (**CTL**.BLOCK_TS), or if this is at the maximum value, use Row 10 of **Table 5-5** and set up the LLI.SAR address of the block descriptor to be equal to the end **SAR** address of the previous block. Similarly, set up the LLI.DAR address of the block descriptor to be equal to the end **DAR** address of the previous block.*

5.4.4 Suspension of Transfers Between Blocks

At the end of every block transfer, an end-of-block interrupt is asserted if:

1. Interrupts are enabled, **CTL**.INT_EN = 1, and
2. The channel block interrupt is unmasked, MASKBLOCK[n] = 1, where n is the channel number.

Note: The block-complete interrupt is generated at the completion of the block transfer to the destination.

For rows 6, 8, and 10 of **Table 5-5**, the DMA transfer does not stall between block transfers. For example, at the end-of-block N, the GPDMA automatically proceeds to block N + 1.

For rows 2, 3, 4, 7, and 9 of **Table 5-5** (**SAR** and/or **DAR** auto-reloaded between block transfers), the DMA transfer automatically stalls after the end-of-block interrupt is asserted, if the end-of-block interrupt is enabled and unmasked.

The GPDMA does not proceed to the next block transfer until a write to the CLEARBLOCK[n] block interrupt clear register, done by software to clear the channel block-complete interrupt, is detected by hardware.

For rows 2, 3, 4, 7, and 9 of **Table 5-5** (**SAR** and/or **DAR** auto-reloaded between block transfers), the DMA transfer does not stall if either:

- Interrupts are disabled, **CTL**.INT_EN = 0, or
- The channel block interrupt is masked, MASKBLOCK[n] = 0, where n is the channel number.

Channel suspension between blocks is used to ensure that the end-of-block ISR (interrupt service routine) of the next-to-last block is serviced before the start of the final block commences. This ensures that the ISR has cleared the **CFG**.RELOAD_SRC and/or **CFG**.RELOAD_DST bits before completion of the final block. The reload bits **CFG**.RELOAD_SRC and/or **CFG**.RELOAD_DST should be cleared in the end-of-block ISR for the next-to-last block transfer.

5.4.5 Ending Multi-Block Transfers

All multi-block transfers must end as shown in either Row 1 or Row 5 of **Table 5-5**. At the end of every block transfer, the GPDMA samples the row number, and if the GPDMA is in the Row 1 or Row 5 state, then the previous block transferred was the last block and the DMA transfer is terminated.

Note: Row 1 and Row 5 are used for single-block transfers or terminating multi-block transfers. Ending in the Row 5 state enables status fetch and write-back for the last block. Ending in the Row 1 state disables status fetch and write-back for the last block.

For rows 2, 3, and 4 of **Table 5-5**, (**LLP**.LOC = 0 and **CFG**.RELOAD_SRC and/or **CFG**.RELOAD_DST is set), multi-block DMA transfers continue until both the **CFG**.RELOAD_SRC and **CFG**.RELOAD_DST registers are cleared by software. They should be programmed to 0 in the end-of-block interrupt service routine that services the next-to-last block transfer; this puts the GPDMA into the Row 1 state.

For rows 6, 8, and 10 of **Table 5-5** (both **CFG**.RELOAD_SRC and **CFG**.RELOAD_DST cleared), the user must set up the last block descriptor in memory so that both LLI.CTLH/L.LLP_SRC_EN and LLI.CTLH/L.LLP_DST_EN are 0. If the LLI.LLP register of the last block descriptor in memory is non-zero, then the DMA transfer is terminated in Row 5. If the LLI.LLP register of the last block descriptor in memory is 0, then the DMA transfer is terminated in Row 1.

*Note: The only allowed transitions between the rows of **Table 5-5** are from any row into Row 1 or Row 5. As already stated, a transition into row 1 or row 5 is used to terminate the DMA transfer; all other transitions between rows are not allowed. Software must ensure that illegal transitions between rows do not occur between blocks of a multi-block transfer. For example, if block N is in row 10, then the only allowed rows for block N +1 are rows 10, 5, or 1.*

5.4.6 Programing Examples

Three registers - **LLP**, **CTL**, and **CFG** - need to be programmed to determine whether single- or multi-block transfers occur, and which type of multi-block transfer is used. The different transfer types are shown in **Table 5-5**.

The GPDMA can be programmed to fetch the status from the source or destination peripheral; this status is stored in the **SSTAT** and **DSTAT** registers. When the GPDMA is programmed to fetch the status from the source or destination peripheral, it writes this status and the contents of the **CTL** register back to memory at the end of a block transfer. The Write Back column of **Table 5-5** shows when this occurs.

The "Update Method" columns indicate where the values of **SAR**, **DAR**, **CTL**, and **LLP** are obtained for the next block transfer when multi-block GPDMA transfers are enabled.

*Note: In **Table 5-5**, all other combinations of **LLP**.LOC = 0, **CTL**.LLP_SRC_EN, **CFG**.RELOAD_SRC, **CTL**.LLP_DST_EN, and **CFG**.RELOAD_DST are illegal, and will cause indeterminate or erroneous behavior.*

Generic Setup of Transfer Type and Characteristics

This generic sequence is referenced by the examples further below in this section.

1. Set up the transfer type (memory or non-memory peripheral for source and destination) and flow control device by programming the TT_FC of the **CTL** register. **Table 5-10** lists the decoding for this field.
2. Set up the transfer characteristics, such as:
   a) Transfer width for the source in the SRC_TR_WIDTH field.
   Table 5-9 lists the decoding for this field.
   b) Transfer width for the destination in the DST_TR_WIDTH field.
   Table 5-9 lists the decoding for this field.
   c) Incrementing/decrementing or fixed address for the source in the SINC field.
   d) Incrementing/decrementing or fixed address for the destination in the DINC field.

5.4.6.1 Single-block Transfer

This section is an example for the transfer listed in row 1 in **Table 5-5**.

*Note: Row 5 in **Table 5-5** is also a single-block transfer with write-back of control and status information enabled at the end of the single-block transfer.*

1. Read the Channel Enable register to choose a free (disabled) channel.
2. Clear any pending interrupts on the channel from the previous DMA transfer by writing to the Interrupt Clear registers: CLEARTFR, CLEARBLOCK, CLEARSRCTRAN, CLEARDSTTRAN, and CLEARERR. Reading the Interrupt Raw Status and Interrupt Status registers confirms that all interrupts have been cleared.
3. Program the following channel registers:
   a) Write the starting source address in the **SAR** register for channel x.

b) Write the starting destination address in the **DAR** register for channel x.
c) Program **CTL** and **CFG** according to Row 1, as shown in **Table 5-5**. Program the **LLP** register with 0.
d) Write the control information for the DMA transfer in the **CTL** register for channel x.
e) Write the channel configuration information into the **CFG** register for channel x.
1. Designate the handshaking interface type (hardware or software) for the source and destination peripherals; this is not required for memory.
This step requires programming the **CFG**.HS_SEL_SRC or **CFG**.HS_SEL_DST bits, respectively. Writing a 0 activates the hardware handshaking interface to handle source/destination requests. Writing a 1 activates the software handshaking interface to handle source and destination requests.
2. If the hardware handshaking interface is activated for the source or destination peripheral, assign a handshaking interface to the source and destination peripheral; this requires programming the **CFG**.SRC_PER and **CFG**.DEST_PER bits, respectively.
f) If gather is enabled (**CTL**.SRC_GATHER_EN = 1), program the **SGR** register for channel x.
g) If scatter is enabled (**CTL**.DST_SCATTER_EN = 1), program the **DSR** register for channel x.

4. After the GPDMA-selected channel has been programmed, enable the channel by writing a 1 to the **GPDMA0_CHENREG**.CH_EN bit. Ensure that bit 0 of the **GPDMA0_DMACFGREG** register is enabled.
5. Source and destination request single and burst DMA transactions in order to transfer the block of data (assuming non-memory peripherals). The GPDMA acknowledges at the completion of every transaction (burst and single) in the block and carries out the block transfer.
6. Once the transfer completes, hardware sets the interrupts and disables the channel. At this time, you can respond to either the Block Complete or Transfer Complete interrupts, or poll for the transfer complete raw interrupt status register (RAWTFR[n], n = channel number) until it is set by hardware, in order to detect when the transfer is complete. Note that if this polling is used, the software must ensure that the transfer complete interrupt is cleared by writing to the Interrupt Clear register, CLEARTFR[n], before the channel is enabled.

5.4.6.2 Multi-Block Transfer with Source Address Auto-Reloaded and Contiguous Destination Address

This section is an example for the transfer listed in row 3 in **Table 5-5**.

Note: This type of transfer is supported by GPDMA0 channels 0 and 1 only.

1. Read the Channel Enable register (see **GPDMA0_CHENREG**) to choose a free (disabled) channel.

2. Clear any pending interrupts on the channel from the previous DMA transfer by writing to the Interrupt Clear registers: CLEARTFR, CLEARBLOCK, CLEARSRCTRAN, CLEARDSTTRAN, and CLEARERR. Reading the Interrupt Raw Status and Interrupt Status registers confirms that all interrupts have been cleared.
3. Program the following channel registers:
   a) Write the starting source address in the **SAR** register for channel x.
   b) Write the starting destination address in the **DAR** register for channel x.
   c) Program **CTL** and **CFG** according to Row 3, shown in **Table 5-5**. Program the **LLP** register with 0.
   d) Write the control information for the DMA transfer in the **CTL** register for channel x.
   e) If gather is enabled (**CTL**.SRC_GATHER_EN = 1), program the **SGR** register for channel x.
   f) If scatter is enabled (**CTL**.DST_SCATTER_EN = 1), program the **DSR** register for channel x.
   g) Write the channel configuration information into the **CFG** register for channel x.
      1. Designate the handshaking interface type (hardware or software) for the source and destination peripherals; this is not required for memory.
      This step requires programming the HS_SEL_SRC, HS_SEL_DST bits, respectively. Writing a 0 activates the hardware handshaking interface to handle source/destination requests for the specific channel. Writing a 1 activates the software handshaking interface to handle source/destination requests.
      2. If the hardware handshaking interface is activated for the source or destination peripheral, assign the handshaking interface to the source and destination peripheral. This requires programming the SRC_PER and DEST_PER bits, respectively.
4. After the GPDMA channel has been programmed, enable the channel by writing a 1 to the **GPDMA0_CHENREG**.CH_EN bit. Ensure that bit 0 of the **GPDMA0_DMACFGREG** register is enabled.
5. Source and destination request single and burst GPDMA transactions to transfer the block of data (assuming non-memory peripherals). The GPDMA acknowledges at the completion of every transaction (burst and single) in the block and carries out the block transfer.
6. When the block transfer has completed, the GPDMA reloads the **SAR** register; the **DAR** register remains unchanged. Hardware sets the block-complete interrupt. The GPDMA then samples the row number, as shown in **Table 5-5**. If the GPDMA is in Row 1, then the DMA transfer has completed. Hardware sets the transfer-complete interrupt and disables the channel. You can either respond to the Block Complete or Transfer Complete interrupts, or poll for the transfer complete raw interrupt status register (RAWTFR[n], n = channel number) until it is set by hardware, in order to detect when the transfer is complete. Note that if this polling is used, software must ensure that the transfer complete interrupt is cleared by writing to the Interrupt Clear register, CLEARTFR[n], before the channel is enabled. If the GPDMA is not in Row 1, the next step is performed.

7. The DMA transfer proceeds as follows:
   a) If interrupts are enabled (**CTL**.INT_EN = 1) and the block-complete interrupt is unmasked (MASKBLOCK[x] = 1_B, where x is the channel number), hardware sets the block-complete interrupt when the block transfer has completed. It then stalls until the block-complete interrupt is cleared by software. If the next block is to be the last block in the DMA transfer, then the block-complete ISR (interrupt service routine) should clear the source reload bit, **CFG**.RELOAD_SRC. This puts the GPDMA into Row 1, as shown in **Table 5-5**. If the next block is not the last block in the DMA transfer, then the source reload bit should remain enabled to keep the GPDMA in Row 3, as shown in **Table 5-5**.
   b) If interrupts are disabled (**CTL**.INT_EN = 0) or the block-complete interrupt is masked (MASKBLOCK[x] = 0_B, where x is the channel number), then hardware does not stall until it detects a write to the block-complete interrupt clear register; instead, it starts the next block transfer immediately. In this case, software must clear the source reload bit, **CFG**.RELOAD_SRC, to put the device into Row 1 of **Table 5-5** before the last block of the DMA transfer has completed.

The transfer is similar to that shown in **Figure 5-18**.

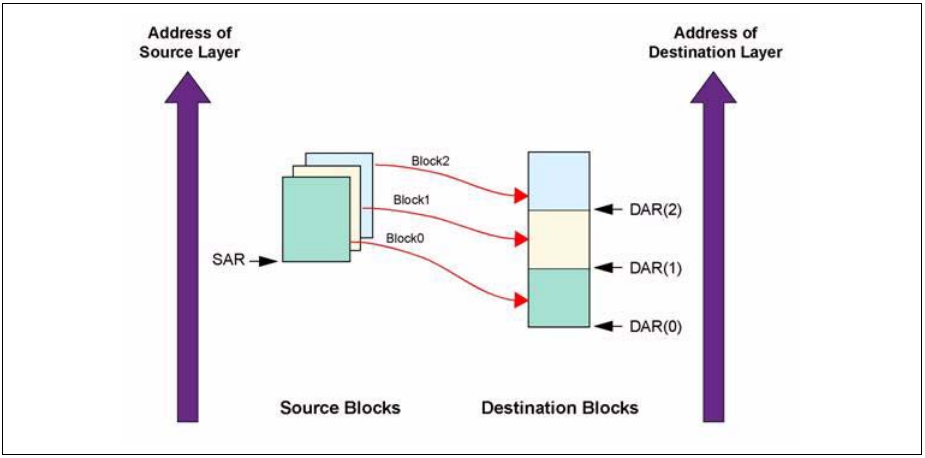


Figure 5-18 Multi-Block DMA Transfer with Source Address Auto-Reloaded and Contiguous Destination Address

The DMA transfer flow is shown in **Figure 5-19**.

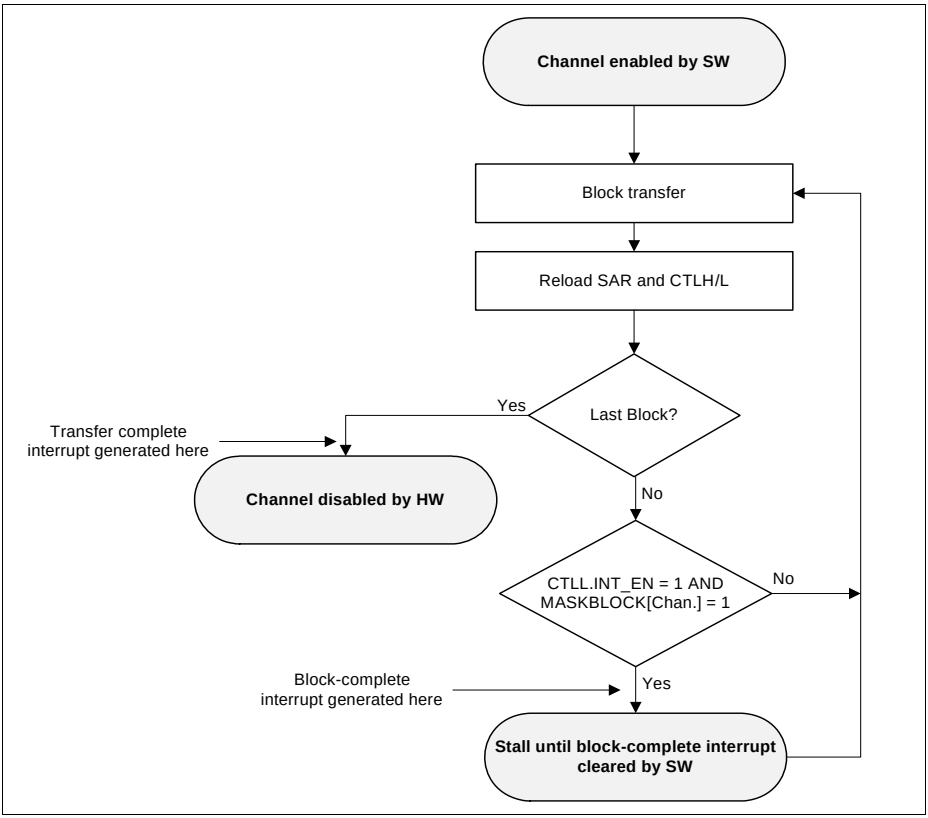


Figure 5-19 DMA Transfer Flow for Source Address Auto-Reloaded and Linked List Destination Address

5.4.6.3 Multi-Block Transfer with Source and Destination Address Auto-Reloaded

This section is an example for the transfer listed in row 4 in **Table 5-5**.

Note: This type of transfer is supported by GPDMA0 channels 0 and 1 only.

1. Read the Channel Enable register (see **GPDMA0_CHENREG**) to choose an available (disabled) channel.
2. Clear any pending interrupts on the channel from the previous DMA transfer by writing to the Interrupt Clear registers: CLEARTFR, CLEARBLOCK, CLEARSRCTRAN, CLEARDSTTRAN, and CLEARERR. Reading the Interrupt Raw Status and Interrupt Status registers confirms that all interrupts have been cleared.
3. Program the following channel registers:
   a) Write the starting source address in the **SAR** register for channel x.
   b) Write the starting destination address in the **DAR** register for channel x.
   c) Program **CTL** and **CFG** according to Row 4, as shown in **Table 5-5**. Program the **LLP** register with 0.
   d) Write the control information for the DMA transfer in the **CTL** register for channel x.
   e) If gather is enabled (**CTL**.SRC_GATHER_EN = 1), program the **SGR** register for channel x.
   f) If scatter is enabled (**CTL**.DST_SCATTER_EN = 1), program the **DSR** register for channel x.
   g) Write the channel configuration information into the **CFG** register for channel x. Ensure that the reload bits, **CFG**. RELOAD_SRC and **CFG**.RELOAD_DST, are enabled.
      1. Designate the handshaking interface type (hardware or software) for the source and destination peripherals; this is not required for memory.
      This step requires programming the HS_SEL_SRC, HS_SEL_DST bits, respectively. Writing a 0 activates the hardware handshaking interface to handle source/destination requests for the specific channel. Writing a 1 activates the software handshaking interface to handle source/destination requests.
      2. If the hardware handshaking interface is activated for the source or destination peripheral, assign the handshaking interface to the source and destination peripheral. This requires programming the SRC_PER and DEST_PER bits, respectively.
4. After the GPDMA selected channel has been programmed, enable the channel by writing a 1 to the **GPDMA0_CHENREG**.CH_EN bit. Ensure that bit 0 of the **GPDMA0_DMACFGREG** register is enabled.
5. Source and destination request single and burst GPDMA transactions to transfer the block of data (assuming non-memory peripherals). The GPDMA acknowledges on completion of each burst/single transaction and carries out the block transfer.
6. When the block transfer has completed, the GPDMA reloads the **SAR**, **DAR**, and **CTL** registers. Hardware sets the block-complete interrupt. The GPDMA then

samples the row number, as shown in **Table 5-5**. If the GPDMA is in Row 1, then the DMA transfer has completed. Hardware sets the transfer complete interrupt and disables the channel. You can either respond to the Block Complete or Transfer Complete interrupts, or poll for the transfer complete raw interrupt status register (RAWTFR[n], where n is the channel number) until it is set by hardware, in order to detect when the transfer is complete. Note that if this polling is used, software must ensure that the transfer complete interrupt is cleared by writing to the Interrupt Clear register, CLEARTFR[n], before the channel is enabled. If the GPDMA is not in Row 1, the next step is performed.

7. The DMA transfer proceeds as follows:
   a) If interrupts are enabled (**CTL**.INT_EN = 1) and the block-complete interrupt is unmasked (MASKBLOCK[x] = 1_B, where x is the channel number), hardware sets the block-complete interrupt when the block transfer has completed. It then stalls until the block-complete interrupt is cleared by software. If the next block is to be the last block in the DMA transfer, then the block-complete ISR (interrupt service routine) should clear the reload bits in the **CFG**.RELOAD_SRC and **CFG**.RELOAD_DST registers. This puts the GPDMA into Row 1, as shown in **Table 5-5**. If the next block is not the last block in the DMA transfer, then the reload bits should remain enabled to keep the GPDMA in Row 4.
   b) If interrupts are disabled (**CTL**.INT_EN = 0) or the block-complete interrupt is masked (MASKBLOCK[x] = 0_B, where x is the channel number), then hardware does not stall until it detects a write to the block-complete interrupt clear register; instead, it immediately starts the next block transfer. In this case, software must clear the reload bits in the **CFG**.RELOAD_SRC and **CFG**.RELOAD_DST registers to put the GPDMA into Row 1 of **Table 5-5** before the last block of the DMA transfer has completed.

The transfer is similar to that shown in **Figure 5-20**.

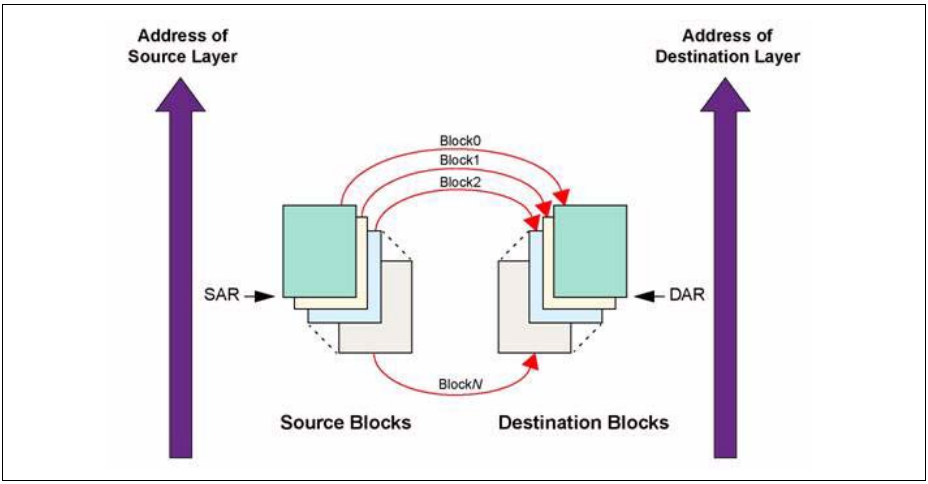


Figure 5-20 Multi-Block DMA Transfer with Source and Destination Address Auto-Reloaded

The DMA transfer flow is shown in **Figure 5-21**.

Figure 5-21 DMA Transfer Flow for Source and Destination Address Auto-Reloaded

5.4.6.4 Multi-Block Transfer with Source Address Auto-Reloaded and Linked List Destination Address

This section is an example for the transfer listed in row 7 in **Table 5-5**.

Note: This type of transfer is supported by GPDMA0 channels 0 and 1 only.

1. Read the Channel Enable register (see **GPDMA0_CHENREG**) in order to choose a free (disabled) channel.
2. Set up the chain of linked list items (otherwise known as block descriptors) in memory. Write the control information in the LLI.**CTL** register location of the block descriptor for each LLI in memory (see **Figure 5-14**) for channel x.
3. Write the starting source address in the **SAR** register for channel x.
   Note: *The values in the LLI.SAR register locations of each of the Linked List Items (LLIs) set up in memory, although fetched during an LLI fetch, are not used.*
4. Write the channel configuration information into the **CFG** register for channel x.
   a) Designate the handshaking interface type (hardware or software) for the source and destination peripherals; this is not required for memory.
      This step requires programming the HS_SEL_SRC, HS_SEL_DST bits. Writing a 0 activates the hardware handshaking interface to handle source/destination requests for the specific channel. Writing a 1 activates the software handshaking interface source/destination requests.
   b) If the hardware handshaking interface is activated for the source or destination peripheral, assign the handshaking interface to the source and destination peripheral; this requires programming the SRC_PER and DEST_PER bits, respectively.
5. Make sure that the LLI.CTLH/L register locations of all LLIs in memory (except the last) are set as shown in Row 7 of **Table 5-5**, while the LLI.CTLH/L register of the last Linked List item must be set as described in Row 1 or Row 5 of **Table 5-5**. Figure 7-1 shows a Linked List example with two list items.
6. Ensure that the LLI.LLP register locations of all LLIs in memory (except the last) are non-zero and point to the next Linked List Item.
7. Ensure that the LLI.DAR register location of all LLIs in memory point to the start destination block address preceding that LLI fetch.
8. Ensure that the LLI.CTLH/L.DONE fields of the LLI.CTLH/L register locations of all LLIs in memory are cleared.
9. If source status fetching is enabled (**CFG**.SS_UPD_EN is enabled), program the **SSTATAR** register so that the source status information can be fetched from the location pointed to by the **SSTATAR**. For conditions under which the source status information is fetched from system memory, refer to the Write Back column of **Table 5-5**.
10. If destination status fetching is enabled (**CFG**.DS_UPD_EN is enabled), program the **DSTATAR** register so that the destination status information can be fetched from the location pointed to by the **DSTATAR** register. For conditions under which the

destination status information is fetched from system memory, refer to the Write Back column of **Table 5-5**.

11. If gather is enabled (**CTL**.SRC_GATHER_EN = 1), program the **SGR** register for channel x.
12. If scatter is enabled (**CTL**.DST_SCATTER_EN = 1), program the **DSR** register for channel x.
13. Clear any pending interrupts on the channel from the previous DMA transfer by writing to the Interrupt Clear registers: CLEARTFR, CLEARBLOCK, CLEARSRCTRAN, CLEARDSTTRAN, and CLEARERR. Reading the Interrupt Raw Status and Interrupt Status registers confirms that all interrupts have been cleared.
14. Program the **CTL** and **CFG** registers according to Row 7, as shown in **Table 5-5**.
15. Program the **LLP** register with **LLP**(0), the pointer to the first Linked List item.
16. Finally, enable the channel by writing a 1 to the **GPDMA0_CHENREG**.CH_EN bit; the transfer is performed. Ensure that bit 0 of the **GPDMA0_DMACFGREG** register is enabled.
17. The GPDMA fetches the first LLI from the location pointed to by **LLP**(0).
    Note: *The LLI.SAR, LLI.DAR, LLI.LLP, and LLI.CTLH/L registers are fetched. The LLI.SAR register - although fetched - is not used.*
18. Source and destination request single and burst GPDMA transactions in order to transfer the block of data (assuming non-memory peripherals). The GPDMA acknowledges at the completion of every transaction (burst and single) in the block and carries out the block transfer.
19. Once the block of data is transferred, the source status information is fetched from the location pointed to by the **SSTATAR** register and stored in the **SSTAT** register if **CFG**.SS_UPD_EN is enabled. For conditions under which the source status information is fetched from system memory, refer to the Write Back column of **Table 5-5**.
    The destination status information is fetched from the location pointed to by the **DSTATAR** register and stored in the **DSTAT** register if **CFG**.DS_UPD_EN is enabled. For conditions under which the destination status information is fetched from system memory, refer to the Write Back column of **Table 5-5**.
20. The **CTLH** register is written out to system memory. For conditions under which the **CTLH** register is written out to system memory, refer to the Write Back column of **Table 5-5**.
    The **CTLH** register is written out to the same location where it was originally fetched; that is, the location of the **CTL** register of the linked list item fetched prior to the start of the block transfer. Only the **CTLH** register is written out, because only the **CTL**.BLOCK_TS and **CTL**.DONE fields have been updated by hardware within the GPDMA. The LLI.CTLH/L.DONE bit is asserted to indicate block completion. Therefore, software can poll the LLI.**CTL**.DONE bit field of the **CTL** register in the LLI to ascertain when a block transfer has completed.
    Note: *Do not poll the* **CTL***.DONE bit in the* GPDMA *memory map. Instead, poll the LLI.CTLH/L.DONE bit in the LLI for that block. If the polled LLI.CTLH/L.DONE bit is*

asserted, then this block transfer has completed. This LLI.CTLH/L.DONE bit was cleared at the start of the transfer (Step 8).

21. The **SSTAT** register is now written out to system memory if **CFG**.SS_UPD_EN is enabled. It is written to the SSTAT register location of the LLI pointed to by the previously saved **LLP**.LOC register.
    The **DSTAT** register is now written out to system memory if **CFG**.DS_UPD_EN is enabled. It is written to the DSTAT register location of the LLI pointed to by the previously saved **LLP**.LOC register.
    The end-of-block interrupt, int_block, is generated after the write-back of the control and status registers has completed.
    Note: *The write-back location for the control and status registers is the LLI pointed to by the previous value of the **LLP**.LOC register, not the LLI pointed to by the current value of the **LLP**.LOC register.*
22. The GPDMA reloads the **SAR** register from the initial value. Hardware sets the block-complete interrupt. The GPDMA samples the row number, as shown in **Table 5-5**. If the GPDMA is in Row 1 or Row 5, then the DMA transfer has completed. Hardware sets the transfer complete interrupt and disables the channel. You can either respond to the Block Complete or Transfer Complete interrupts, or poll for the transfer complete raw interrupt status register (RAWTFR[n], n = channel number) until it is set by hardware, in order to detect when the transfer is complete. Note that if this polling is used, software must ensure that the transfer complete interrupt is cleared by writing to the Interrupt Clear register, CLEARTFR[n], before the channel is enabled. If the GPDMA is not in Row 1 or Row 5 as shown in **Table 5-5**, the following steps are performed.
23. The DMA transfer proceeds as follows:
    a) If interrupts are enabled (**CTL**.INT_EN = 1) and the block-complete interrupt is unmasked (MASKBLOCK[x] = 1_B, where x is the channel number), hardware sets the block-complete interrupt when the block transfer has completed. It then stalls until the block-complete interrupt is cleared by software. If the next block is to be the last block in the DMA transfer, then the block-complete ISR (interrupt service routine) should clear the **CFG**.RELOAD_SRC source reload bit. This puts the GPDMA into Row 1, as shown in **Table 5-5**. If the next block is not the last block in the DMA transfer, then the source reload bit should remain enabled to keep the GPDMA in Row 7, as shown in **Table 5-5**.
    b) If interrupts are disabled (**CTL**.INT_EN = 0) or the block-complete interrupt is masked (MASKBLOCK[x] = 0_B, where x is the channel number), then hardware does not stall until it detects a write to the block-complete interrupt clear register; instead, it immediately starts the next block transfer. In this case, software must clear the source reload bit, **CFG**.RELOAD_SRC in order to put the device into Row 1 of **Table 5-5** before the last block of the DMA transfer has completed.
24. The GPDMA fetches the next LLI from memory location pointed to by the current **LLP** register and automatically reprograms the **DAR**, **CTL**, and **LLP** channel registers. Note that the **SAR** is not reprogrammed, since the reloaded value is used for the next

DMA block transfer. If the next block is the last block of the DMA transfer, then the **CTL** and **LLP** registers just fetched from the LLI should match Row 1 or Row 5 of **Table 5-5**.

The DMA transfer might look like that shown in **Figure 5-22**.

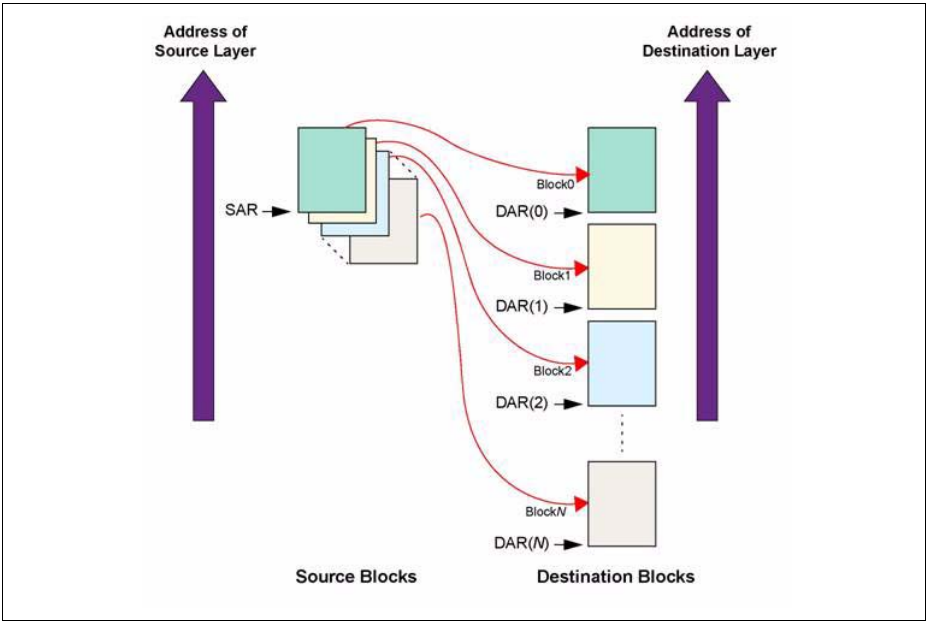


Figure 5-22 Multi-Block DMA Transfer with Source Address Auto-Reloaded and Linked List Destination Address

The DMA transfer flow is shown in **Figure 5-23**.

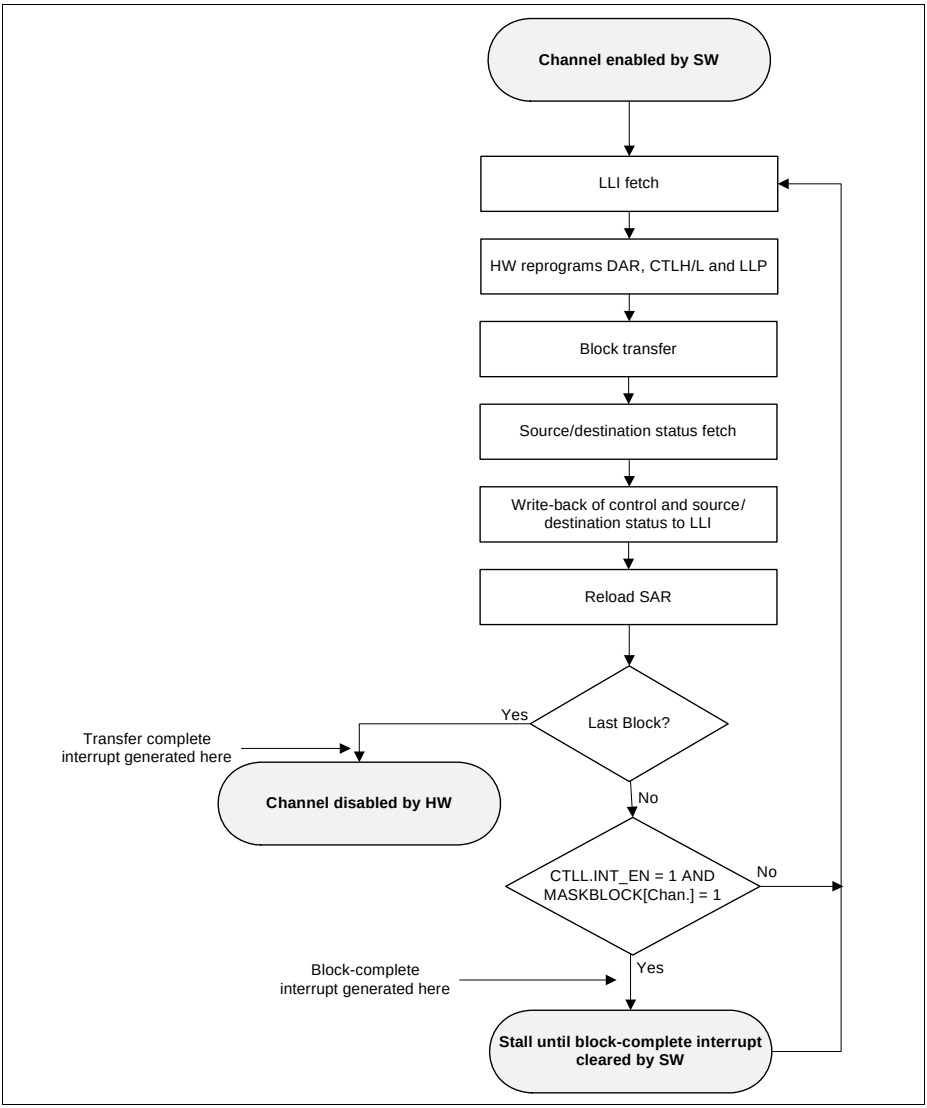


Figure 5-23 DMA Transfer Flow for Source Address Auto-Reloaded and Linked List Destination Address

5.4.6.5 Multi-Block DMA Transfer with Linked List for Source and Contiguous Destination Address

This section is an example for the transfer listed in row 8 in **Table 5-5**.

Note: This type of transfer is supported by GPDMA0 channels 0 and 1 only.

1. Read the Channel Enable register (see **GPDMA0_CHENREG**) to choose a free (disabled) channel.
2. Set up the linked list in memory. Write the control information in the LLI.**CTL** register location of the block descriptor for each LLI in memory (see **Figure 5-14**) for channel x.
3. Write the starting destination address in the **DAR** register for channel x.
   Note: *The values in the LLI.DAR register location of each Linked List Item (LLI) in memory, although fetched during an LLI fetch, are not used.*
4. Write the channel configuration information into the **CFG** register for channel x.
   1. Designate the handshaking interface type (hardware or software) for the source and destination peripherals; this is not required for memory.
   This step requires programming the HS_SEL_SRC, HS_SEL_DST bits. Writing a 0 activates the hardware handshaking interface to handle source/destination requests for the specific channel. Writing a 1 activates the software handshaking interface to handle source/destination requests.
   2. If the hardware handshaking interface is activated for the source or destination peripheral, assign the handshaking interface to the source and destination peripherals. This requires programming the SRC_PER and DEST_PER bits, respectively.
5. Ensure that all LLI.CTLH/L register locations of the LLI (except the last) are set as shown in Row 8 of **Table 5-5**, while the LLI.CTLH/L register of the last Linked List item must be set as described in Row 1 or Row 5 of **Table 5-5**. **Figure 5-14** shows a Linked List example with two list items.
6. Ensure that the LLI.LLP register locations of all LLIs in memory (except the last) are non-zero and point to the next Linked List Item.
7. Ensure that the LLI.SAR register location of all LLIs in memory point to the start source block address preceding that LLI fetch.
8. Ensure that the LLI.CTLH/L.DONE fields of the LLI.CTLH/L register locations of all LLIs in memory are cleared.
9. If source status fetching is enabled (**CFG**.SS_UPD_EN is enabled), program the **SSTATAR** register so that the source status information can be fetched from the location pointed to by **SSTATAR**. For conditions under which the source status information is fetched from system memory, refer to the Write Back column of **Table 5-5**.
10. If destination status fetching is enabled (**CFG**.DS_UPD_EN is enabled), program the **DSTATAR** register so that the destination status information can be fetched from the location pointed to by the **DSTATAR** register. For conditions under which the

destination status information is fetched from system memory, refer to the Write Back column of **Table 5-5**.

11. If gather is enabled (**CTL**.SRC_GATHER_EN = 1), program the **SGR** register for channel x.
12. If scatter is enabled (**CTL**.DST_SCATTER_EN = 1), program the **DSR** register for channel x.
13. Clear any pending interrupts on the channel from the previous DMA transfer by writing to the Interrupt Clear registers: CLEARTFR, CLEARBLOCK, CLEARSRCTRAN, CLEARDSTTRAN, and CLEARERR. Reading the Interrupt Raw Status and Interrupt Status registers confirms that all interrupts have been cleared.
14. Program the **CTL** and **CFG** registers according to Row 8, as shown in **Table 5-5**.
15. Program the **LLP** register with **LLP**(0), the pointer to the first Linked List item.
16. Finally, enable the channel by writing a 1 to the **GPDMA0_CHENREG**.CH_EN bit; the transfer is performed. Ensure that bit 0 of the **GPDMA0_DMACFGREG** register is enabled.
17. The GPDMA fetches the first LLI from the location pointed to by **LLP**(0).
    Note: *The LLI.SAR, LLI.DAR, LLI.LLP, and LLI.CTLH/L registers are fetched. The LLI.DAR register location of the LLI - although fetched - is not used. The* **DAR** *register in the* GPDMA *remains unchanged.*
18. Source and destination request single and burst GPDMA transactions to transfer the block of data (assuming non-memory peripherals). The GPDMA acknowledges at the completion of every transaction (burst and single) in the block and carries out the block transfer.
19. Once the block of data is transferred, the source status information is fetched from the location pointed to by the **SSTATAR** register and stored in the **SSTAT** register if **CFG**.SS_UPD_EN is enabled. For conditions under which the source status information is fetched from system memory, refer to the Write Back column of **Table 5-5**.The destination status information is fetched from the location pointed to by the **DSTATAR** register and stored in the **DSTAT** register if **CFG**.DS_UPD_EN is enabled. For conditions under which the destination status information is fetched from system memory, refer to the Write Back column of **Table 5-5**.
20. The **CTLH** register is written out to system memory. For conditions under which the **CTLH** register is written out to system memory, refer to the Write Back column of **Table 5-5**.The **CTLH** register is written out to the same location where it was originally fetched; that is, the location of the **CTL** register of the linked list item fetched prior to the start of the block transfer. Only the second word of the **CTL** register is written out, **CTLH**, because only the **CTL**.BLOCK_TS and **CTL**.DONE fields have been updated by hardware within the GPDMA. Additionally, the **CTL**.DONE bit is asserted to indicate block completion. Therefore, software can poll the LLI.**CTL**.DONE bit field of the **CTL** register in the LLI to ascertain when a block transfer has completed.
    Note: Do not poll the **CTL**.DONE bit in the GPDMA memory map. Instead, poll the LLI.CTLH/L.DONE bit in the LLI for that block. If the polled LLI.CTLH/L.DONE bit is

asserted, then this block transfer has completed. This LLI.CTLH/L.DONE bit was cleared at the start of the transfer (Step 8).

21. The **SSTAT** register is now written out to system memory if **CFG**.SS_UPD_EN is enabled. It is written to the **SSTAT** register location of the LLI pointed to by the previously saved **LLP**.LOC register.The **DSTAT** register is now written out to system memory if **CFG**.DS_UPD_EN is enabled. It is written to the **DSTAT** register location of the LLI pointed to by the previously saved **LLP**.LOC register.The end-of-block interrupt, int_block, is generated after the write-back of the control and status registers has completed.
    Note: The write-back location for the control and status registers is the LLI pointed to by the previous value of the **LLP**.LOC register, not the LLI pointed to by the current value of the **LLP**.LOC register.
22. The GPDMA does not wait for the block interrupt to be cleared, but continues and fetches the next LLI from the memory location pointed to by the current **LLP** register and automatically reprograms the **SAR**, **CTL**, and **LLP** channel registers. The **DAR** register is left unchanged. The DMA transfer continues until the GPDMA samples that the **CTL** and **LLP** registers at the end of a block transfer match those described in Row 1 or Row 5 of **Table 5-5** (as discussed earlier). The GPDMA then knows that the previously transferred block was the last block in the DMA transfer.

The GPDMA transfer might look like that shown in **Figure 5-24**. Note that the destination address is decrementing.

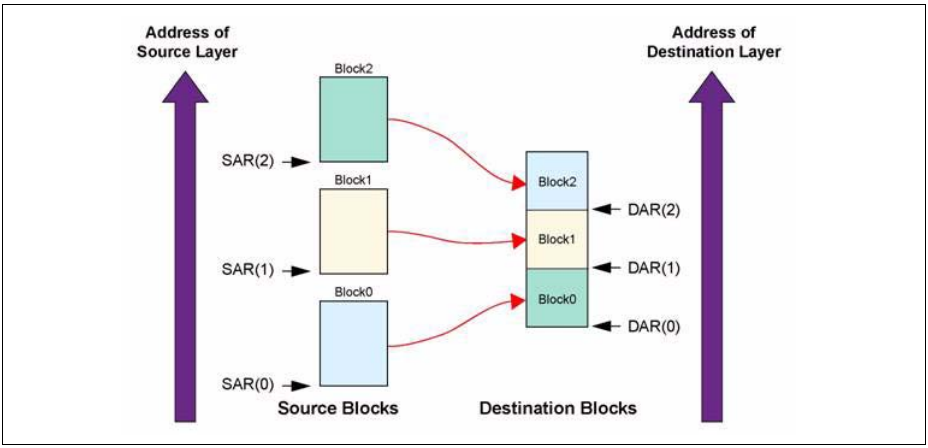


Figure 5-24 Multi-Block DMA Transfer with Linked List Source Address and Contiguous Destination Address

The DMA transfer flow is shown in **Figure 5-25**.

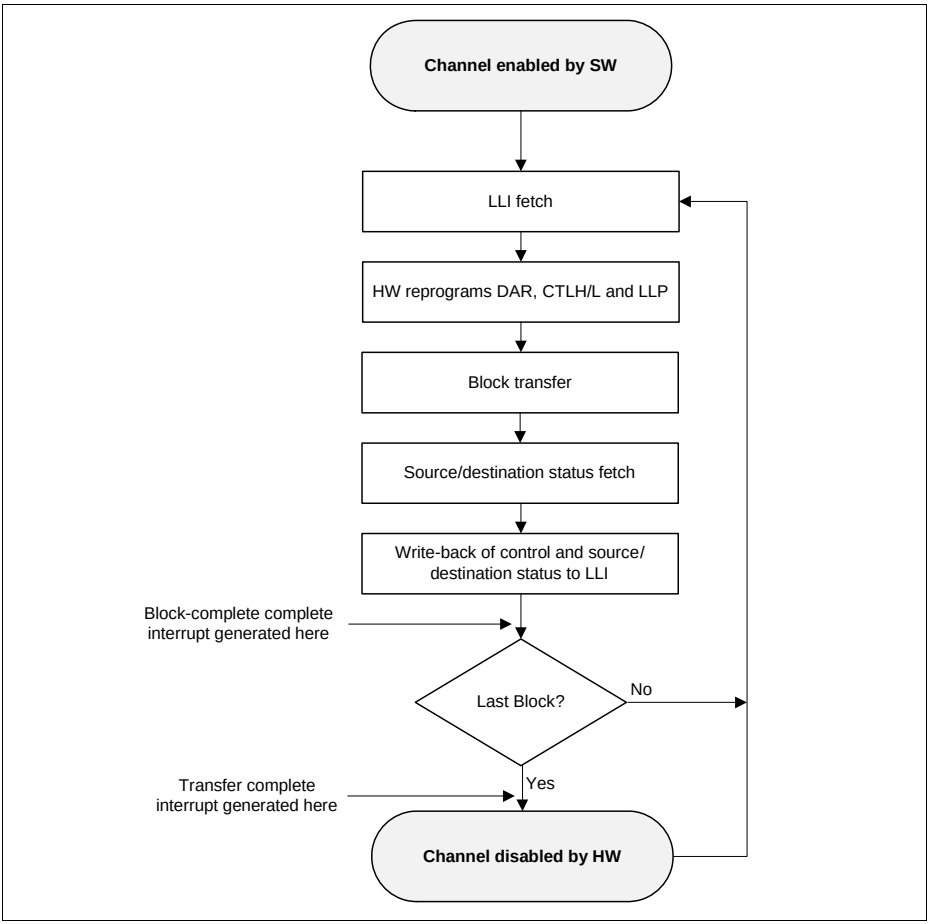


Figure 5-25 DMA Transfer Flow for Source Address Auto-Reloaded and Linked List Destination Address

5.4.6.6 Multi-Block Transfer with Linked List for Source and Destination

This section is an example for the transfer listed in row 10 in **Table 5-5**.

Note: This type of transfer is supported by GPDMA0 channels 0 and 1 only.

1. Read the Channel Enable register (see **GPDMA0_CHENREG**) to choose a free (disabled) channel.

2. Set up the chain of Linked List Items (otherwise known as block descriptors) in memory. Write the control information in the LLI.**CTL** register location of the block descriptor for each LLI in memory (see **Figure 5-14**) for channel x.
3. Write the channel configuration information into the **CFG** register for channel x.
   a) Designate the handshaking interface type (hardware or software) for the source and destination peripherals; this is not required for memory.
   This step requires programming the **CFG**.HS_SEL_SRC or **CFG**.HS_SEL_DST bits, respectively. Writing a 0 activates the hardware handshaking interface to handle source/destination requests for the specific channel. Writing a 1 activates the software handshaking interface to handle source/destination requests.
   b) If the hardware handshaking interface is activated for the source or destination peripheral, assign the handshaking interface to the source and destination peripheral. This requires programming the **CFG**.SRC_PER and **CFG**.DEST_PER bits, respectively.
4. Make sure that the LLI.CTLH/L register locations of all LLI entries in memory (except the last) are set as shown in Row 10 of **Table 5-5**. The LLI.CTLH/L register of the last Linked List Item must be set as described in Row 1 or Row 5 of **Table 5-5**. **Figure 5-14** shows a Linked List example with two list items.
5. Make sure that the LLI.LLP register locations of all LLI entries in memory (except the last) are non-zero and point to the base address of the next Linked List Item.
6. Make sure that the LLI.SAR, LLI.DAR register locations of all LLI entries in memory point to the start source/destination block address preceding that LLI fetch.
7. Ensure that the LLI.CTLH/L.DONE field of the LLI.CTLH/L register locations of all LLI entries in memory is cleared.
8. If source status fetching is enabled (**CFG**.SS_UPD_EN is enabled), program the **SSTATAR** register so that the source status information can be fetched from the location pointed to by the **SSTATAR**. For conditions under which the source status information is fetched from system memory, refer to the Write Back column of **Table 5-5**.
9. If destination status fetching is enabled (**CFG**.DS_UPD_EN is enabled), program the **DSTATAR** register so that the destination status information can be fetched from the location pointed to by the **DSTATAR** register. For conditions under which the destination status information is fetched from system memory, refer to the Write Back column of **Table 5-5**.
10. If gather is enabled (**CTL**.SRC_GATHER_EN = 1), program the **SGR** register for channel x.
11. If scatter is enabled (**CTL**.DST_SCATTER_EN = 1), program the **DSR** register for channel x.
12. Clear any pending interrupts on the channel from the previous DMA transfer by writing to the Interrupt Clear registers: CLEARTFR, CLEARBLOCK, CLEARSRCTRAN, CLEARDSTTRAN, and CLEARERR. Reading the Interrupt Raw Status and Interrupt Status registers confirms that all interrupts have been cleared.
13. Program the **CTL** and **CFG** registers according to Row 10, as shown in **Table 5-5**.

14. Program the **LLP** register with LLP(0), the pointer to the first linked list item.
15. Finally, enable the channel by writing a 1 to the **GPDMA0_CHENREG**.CH_EN bit; the transfer is performed.
16. The GPDMA fetches the first LLI from the location pointed to by LLP(0). The LLI.SAR, LLI.DAR, LLI.LLP, and LLI.CTL registers are fetched and the GPDMA automatically reprograms the according **SAR**, **DAR**, **LLP**, and **CTL** channel registers.
17. Source and destination request single and burst DMA transactions to transfer the block of data (assuming non-memory peripheral). The GPDMA acknowledges at the completion of every transaction (burst and single) in the block and carries out the block transfer.
18. Once the block of data is transferred, the source status information is fetched from the location pointed to by the **SSTATAR** register and stored in the **SSTAT** register if **CFG**.SS_UPD_EN is enabled. For conditions under which the source status information is fetched from system memory, refer to the Write Back column of **Table 5-5**.
    The destination status information is fetched from the location pointed to by the **DSTATAR** register and stored in the **DSTAT** register if **CFG**.DS_UPD_EN is enabled. For conditions under which the destination status information is fetched from system memory, refer to the Write Back column of **Table 5-5**.
19. The **CTLH** register is written out to system memory. For conditions under which the **CTLH** register is written out to system memory, refer to the Write Back column of **Table 5-5**.
    The **CTLH** register is written out to the same location where it was originally fetched; that is, the location of the **CTL** register of the linked list item fetched prior to the start of the block transfer. Only the **CTLH** register is written out, because only the **CTL**.BLOCK_TS and **CTL**.DONE fields have been updated by the GPDMA hardware. Additionally, the **CTL**.DONE bit is asserted to indicate block completion. Therefore, software can poll the LLI.CTLH/L.DONE bit of the **CTL** register in the LLI to ascertain when a block transfer has completed.
    Note: *Do not poll the* **CTL***.DONE bit in the* GPDMA *memory map; instead, poll the LLI.CTLH/L.DONE bit in the LLI for that block. If the polled LLI.CTLH/L.DONE bit is asserted, then this block transfer has completed. This LLI.CTLH/L.DONE bit was cleared at the start of the transfer (Step 7).*
20. The **SSTAT** register is now written out to system memory if **CFG**.SS_UPD_EN is enabled. It is written to the **SSTAT** register location of the LLI pointed to by the previously saved **LLP**.LOC register.
    The **DSTAT** register is now written out to system memory if **CFG**.DS_UPD_EN is enabled. It is written to the **DSTAT** register location of the LLI pointed to by the previously saved **LLP**.LOC register.
    The end-of-block interrupt, int_block, is generated after the write-back of the control and status registers has completed.
    Note: The write-back location for the control and status registers is the LLI pointed to

by the previous value of the **LLP**.LOC register, not the LLI pointed to by the current value of the **LLP**.LOC register.

21. The GPDMA does not wait for the block interrupt to be cleared, but continues fetching the next LLI from the memory location pointed to by the current **LLP** register and automatically reprograms the **SAR**, **DAR**, **CTL**, and **LLP** channel registers. The DMA transfer continues until the GPDMA determines that the **CTL** and **LLP** registers at the end of a block transfer match the ones described in Row 1 or Row 5 of **Table 5-5** (as discussed earlier). The GPDMA then knows that the previously transferred block was the last block in the DMA transfer.

The DMA transfer might look like that shown in **Figure 5-26**.

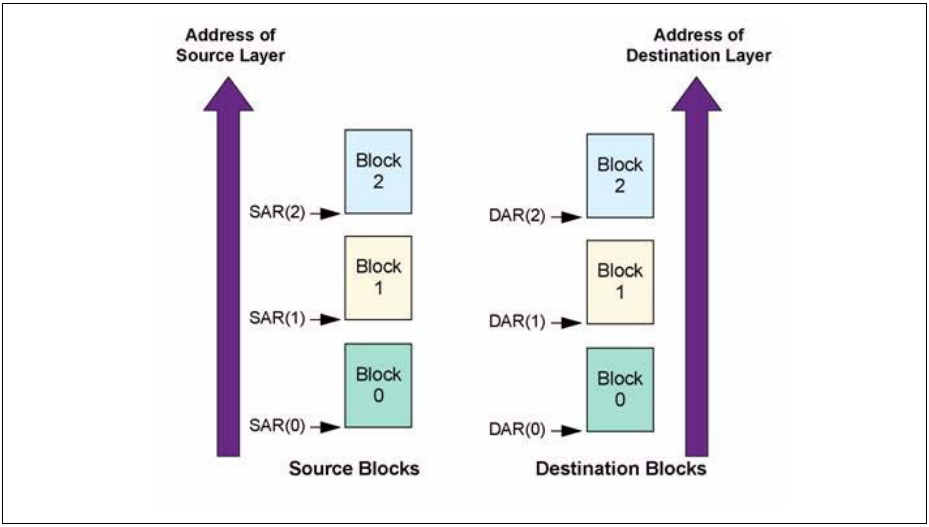


Figure 5-26 Multi-Block with Linked Address for Source and Destination

If the user needs to execute a DMA transfer where the source and destination address are contiguous, but where the amount of data to be transferred is greater than the maximum block size **CTL**.BLOCK_TS, then this can be achieved using the type of multi-block transfer shown in **Figure 5-27**.

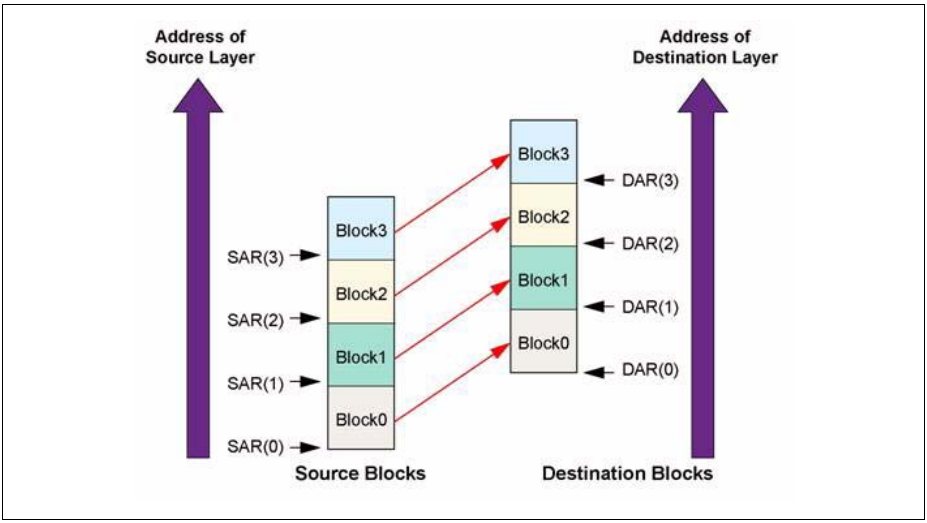


Figure 5-27 Multi-Block with Linked Address for Source and Destination Where SAR and DAR Between Successive Blocks are Contiguous

The DMA transfer flow is shown in **Figure 5-28**.

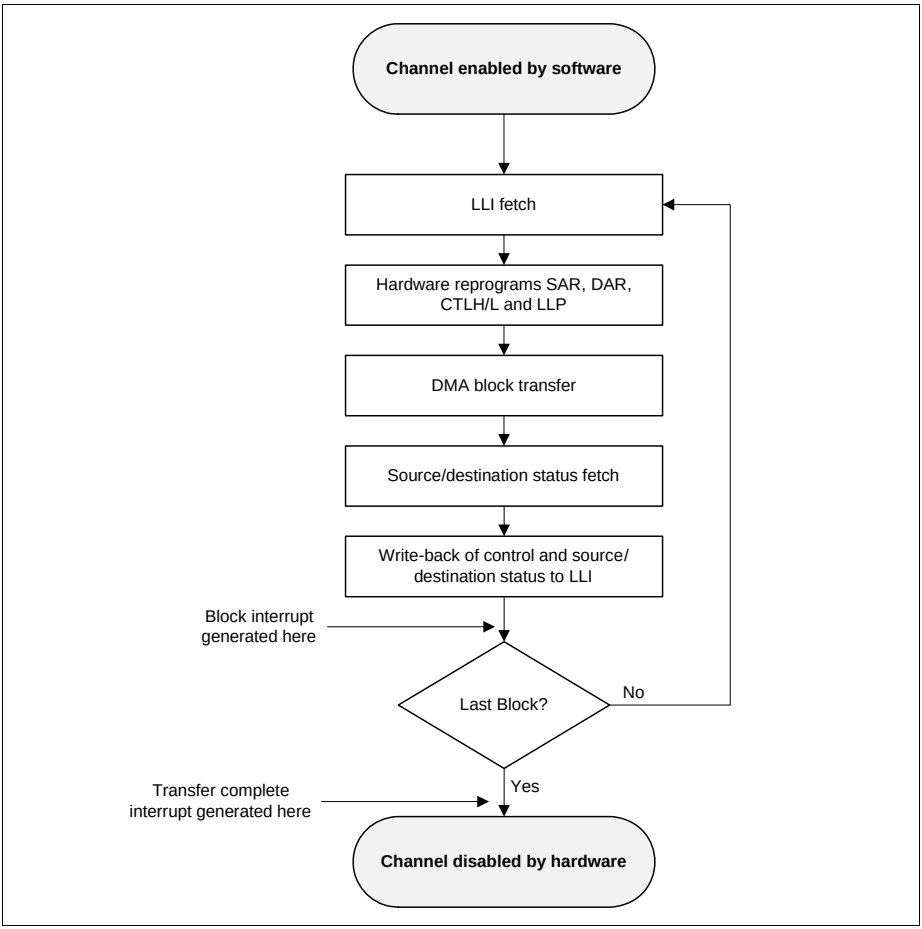


Figure 5-28 DMA Transfer Flow for Source and Destination Linked List Address

5.5 Service Request Generation

Each GPDMA block provides a number of registers (see **Section 5.8.3**) to control the request behavior and to provide an interface for software to check for request occurrence.

The following DMA Events can be generated for each channel due to DMA activity:

- IntSrcTran - Source Transaction Complete
- IntDstTran - Destination Transaction Complete
- IntBlock - Block Transfer Complete
- IntTfr - DMA Transfer Complete
- IntErr - Error

DMA Event processing per channel

Each DMA Event for each channel is directly stored in the according “RAW Status” bit shown in **Figure 5-29**. The user software can control the processing by writing to the according “Mask” and “Clear” bits.

Note: Request forwarding is disabled by default setting of the “Mask” register.

Once the event is forwarded to the “Status” bit its occurrence is registered in the **Combined Interrupt Status Register** and a service request is triggered to the NVIC.

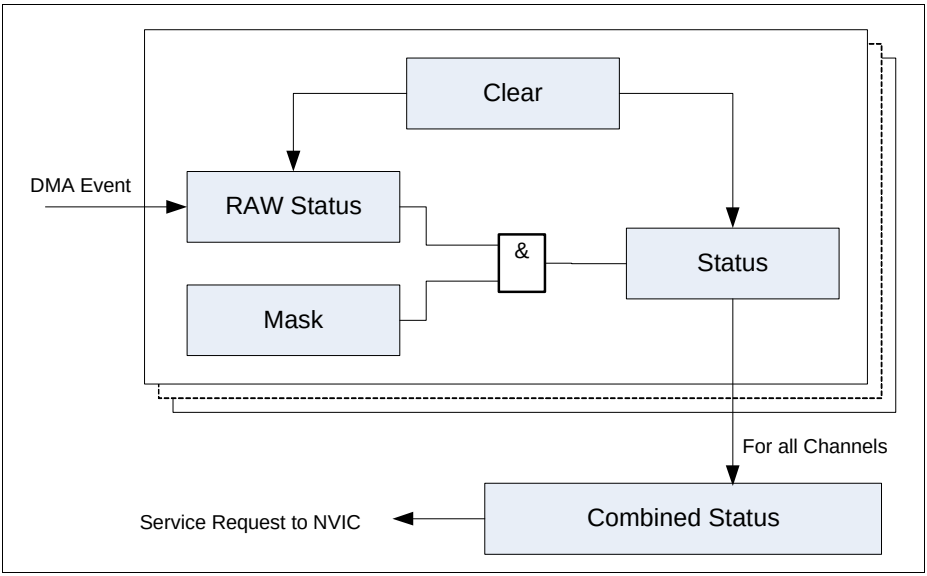


Figure 5-29 DMA Event to Service Request Flow

5.6 Power, Reset and Clock

The GPDMA unit is inside the core power domain, therefore no special considerations about power up or power down sequences need to be taken. For an explanation about the different power domains, please address the SCU (System Control Unit) chapter.

Additionally, if a GPDMA unit is not needed, it can be held in reset via the PRSET2.DMAyRS bitfield (address the SCU chapter for a full description).

The clock used for the GPDMA unit is described on the SCU chapter as f_{DMA}. Please address the specific section under the SCU chapter for a detailed description on the clock configuration schemes.

5.7 Initialization and System Dependencies

The generic initialization sequence for an application that is using the GPDMA, should be the following:

1st Step: Release reset of the GPDMA, via the specific SCU bitfield on the PRCLR2 register.

2nd Step: If the GPDMA is already under use (step 1 was not performed) do the following steps:

- read the channel Enable register to choose a free channel, **CHENREG**. Clear also any pending requests of the specific channel, by writing into the **CLEARTFR**, **CLEARBLOCK**, **CLEARSRCTRAN**, **CLEARDSTTRAN** and **CLEARERR**
- confirm that all the interrupts have been cleared via the Status and RAW registers.

3rd Step: Configure the GPDMA channels accordingly with the wanted transfer type:

- Configure the starting source address and starting destination address, on the **SAR** and **DAR**, respectively.
- Configure the type of transfer that are going to be used via the **LLP**, **CTL** and **CFG** registers.

4th Step: Enable the GPDMA channel, by setting the specific bitfield on the **CHENREG**.

5th Step: Configure the DLR (DMA Line Router) block to map the DMA requests from the peripherals to the wanted DMA request lines (if not previously done).

6th Step: Configure the peripherals that are linked with DMA requests.

7th Step: Enable the specific Service requests on the peripheral blocks.

8th Step: Start the peripheral(s)

*Note: This is a generic channel initialization example. Please refer to **Section 5.3** and **Section 5.4** for a complete description and examples of how to control the complete flow for a GPDMA* channel.

5.8 Registers

This chapter includes information on how to program the GPDMA.

Register references

There are references to software parameters throughout this chapter. The software parameters are the field names in each register description table and are prefixed by the register name; for example, the Block Transfer Size field in the Control register for channel x of GPDMA0 is designated as "GPDMA0_CHx_CTLH.BLOCK_TS”

Illegal Register Access

An illegal access can be any of the following:

1. A write to the **SAR**, **DAR**, **LLP**, **CTL**, **SSTAT**, **DSTAT**, **SSTATAR**, **DSTATAR**, **SGR**, or **DSR** registers occurs when the channel is enabled.
2. A read from the Interrupt Clear Registers is attempted.
3. A write to the Interrupt Status Registers, **GPDMA0_STATUSINT**, **ID** or **VERSION** is attempted.

An illegal access (read/write) returns an AHB error response.

Table 5-6 Registers Address Space

Module	Base Address	End Address	Note
GPDMA0_CH0	5001 4000$_H$	5001 4054$_H$	
GPDMA0_CH1	5001 4058$_H$	5001 40AC$_H$	
GPDMA0_CH2	5001 40B0$_H$	5001 4104$_H$	
GPDMA0_CH3	5001 4108$_H$	5001 415C$_H$	
GPDMA0_CH4	5001 4160$_H$	5001 41B4$_H$	
GPDMA0_CH5	5001 41B8$_H$	5001 420C$_H$	
GPDMA0_CH6	5001 4210$_H$	5001 4264$_H$	
GPDMA0_CH7	5001 4268$_H$	5001 42BC$_H$	
GPDMA0	5001 4000$_H$	5001 7FFF$_H$	

Table 5-7 Register Overview

Short Name	Description	Offset Addr.[1)]	Access Mode		Description See
			Read	Write	
ChannelRegisters					
SAR	Source Address Register	0000_H + x*58_H	U, PV	U, PV	Page 5-65
DAR	Destination Address Register	0008_H + x*58_H	U, PV	U, PV	Page 5-66
Control Registers					
CTLH	Control Register High	$001C_H$ + x*$5C_H$	U, PV	U, PV	Page 5-69
CTLL	Control Register Low	0018_H + x*58_H	U, PV	U, PV	Page 5-71
LLP	Linked List Pointer Register	0010_H + x*58_H	U, PV	U, PV	Page 5-68
SSTAT	Source Status Register	0020_H + x*58_H	U, PV	U, PV	Page 5-77
DSTAT	Destination Status Register	0028_H + x*58_H	U, PV	U, PV	Page 5-78
SSTATAR	Source Status Register	0030_H + x*58_H	U, PV	U, PV	Page 5-79
DSTATAR	Destination Status Register	0038_H + x*58_H	U, PV	U, PV	Page 5-80
CFGH	Configuration Register High	0044_H + x*$5C_H$	U, PV	U, PV	Page 5-81
CFGL	Configuration Register Low	0040_H + x*58_H	U, PV	U, PV	Page 5-87
SGR	Source Gather Register	0048_H + x*58_H	U, PV	U, PV	Page 5-94
DSR	Destination Scatter Register	0050_H + x*58_H	U, PV	U, PV	Page 5-95
Interrupt Registers					

Table 5-7 Register Overview (cont'd)

Short Name	Description	Offset Addr.[1)]	Access Mode		Description See
			Read	Write	
RAW* with *TFR, *BLOCK, *SRCTRAN, *DSTTRAN, *ERR	Interrupt Raw Status Registers	$02C0_H$ - $02E0_H$	U, PV	U, PV	Page 5-98
STATUS* with *TFR, *BLOCK, *SRCTRAN, *DSTTRAN, *ERR	Interrupt Status Registers	$02E8_H$ - 0308_H	U, PV	U, PV	Page 5-100
MASK* with *TFR, *BLOCK, *SRCTRAN, *DSTTRAN, *ERR	Interrupt Mask Registers	0310_H - 0330_H	U, PV	U, PV	Page 5-102
CLEAR* with *TFR, *BLOCK, *SRCTRAN, *DSTTRAN, *ERR	Interrupt Clear Registers	0338_H - 0358_H	U, PV	U, PV	Page 5-104
STATUSINT	Combined Interrupt Status Register	0360_H	U, PV	U, PV	Page 5-105
Software Handshaking Registers					
REQSRCREG	Source Software Transaction Request Register	0368_H	U, PV	U, PV	Page 5-107
REQDSTREG	Destination Software Transaction Request Register	0370_H	U, PV	U, PV	Page 5-108
SGLREQSRCREG	Single Source Transaction Request Register	0378_H	U, PV	U, PV	Page 5-109

Table 5-7 Register Overview (cont'd)

Short Name	Description	Offset Addr.[1]	Access Mode		Description See
			Read	Write	
SGLREQDSTREG	Single Destination Transaction Request Register	0380_H	U, PV	U, PV	Page 5-109
LSTSRCREG	Last Source Transaction Request Register	0388_H	U, PV	U, PV	Page 5-110
LSTDSTREG	Last Destination Transaction Request Register	0390_H	U, PV	U, PV	Page 5-111
Configuration and Channel Enable Registers					
DMACFGREG	Configuration Register	0398_H	U, PV	U, PV	Page 5-63
CHENREG	Channel Enable Register	$03A0_H$	U, PV	U, PV	Page 5-63
Miscellaneous GPDMA Registers					
ID	GPDMA Module ID	$03A8_H$	U, PV	U, PV	Page 5-113
Reserved	Reserved	$03B0_H$ - $03F4_H$	nBE	nBE	
TYPE	GPDMA Component Type	$03F8_H$	U, PV	U, PV	Page 5-113
VERSION	GPDMA Component Version	$03FC_H$	U, PV	U, PV	Page 5-113
Reserved	Reserved	0400_H - $7FFC_H$	nBE	nBE	

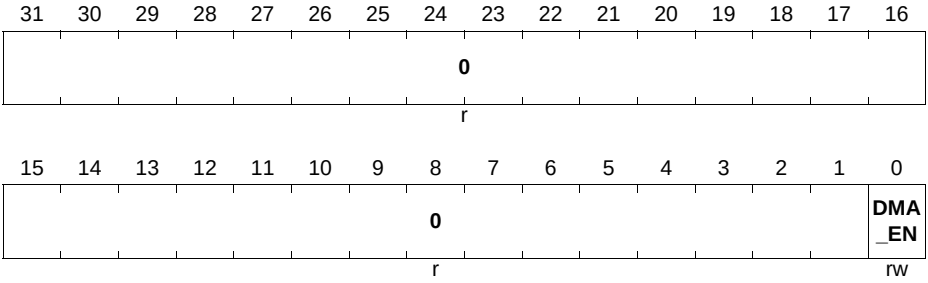
1) x = channel number

5.8.1 Configuration and Channel Enable Registers

DMACFGREG

This register is used to enable the GPDMA, which must be done before any channel activity can begin.

GPDMA0_DMACFGREG
GPDMA Configuration Register (398_H) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
DMA_EN	0	rw	**GPDMA Enable bit.** 0_B GPDMA Disabled 1_B GPDMA Enabled.
0	[31:1]	r	**Reserved**

If the global channel enable bit is cleared while any channel is still active, then DMACFGREG.DMA_EN still returns 1 to indicate that there are channels still active until hardware has terminated all activity on all channels, at which point the DMACFGREG.DMA_EN bit returns 0.

CHENREG

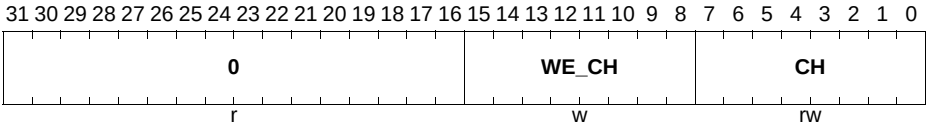
This is the GPDMA “Channel Enable Register”. If software needs to set up a new channel, then it can read this register in order to find out which channels are currently inactive; it can then enable an inactive channel with the required priority.

All bits of this register are cleared to 0 when the global GPDMA channel enable bit, **DMACFGREG**[0], is 0. When the global channel enable bit is 0, then a write to the **CHENREG** register is ignored and a read will always read back 0.

The channel enable bit, **CHENREG**.CH_EN, is written only if the corresponding channel write enable bit, **CHENREG**.CH_EN_WE, is asserted on the same AHB write transfer. For example, writing hex 01x1 writes a 1 into **CHENREG**[0], while **CHENREG**[7:1]

remains unchanged. Writing hex 00xx leaves **CHENREG**[7:0] unchanged. Note that a read-modified write is not required.

GPDMA0_CHENREG
GPDMA Channel Enable Register (3A0$_H$) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
CH	[7:0]	rw	**Enables/Disables the channel** Setting this bit enables a channel; clearing this bit disables the channel. 0_B Disable the Channel 1_B Enable the Channel The CHENREG.CH_EN bit is automatically cleared by hardware to disable the channel after the last AMBA transfer of the DMA transfer to the destination has completed. Software can therefore poll this bit to determine when this channel is free for a new DMA transfer.
WE_CH	[15:8]	w	**Channel enable write enable**
0	[31:16]	r	**Reserved**

5.8.2 Channel Registers

The **SAR**, **DAR**, **LLP**, **CTL**, and **CFG** channel registers should be programmed prior to enabling the channel. However, if an LLI update occurs before commencing data transfer, **SAR** and **DAR** may not need to be programmed prior to enabling the channel; refer to rows 6 to 10 in **Table 5-5** . It is an illegal register access when a write to the **SAR**, **DAR**, **LLP**, **CTL**, **SSTAT**, **DSTAT**, **SSTATAR**, **DSTATAR**, **SGR**, or **DSR** registers occurs when the channel is enabled.

SAR

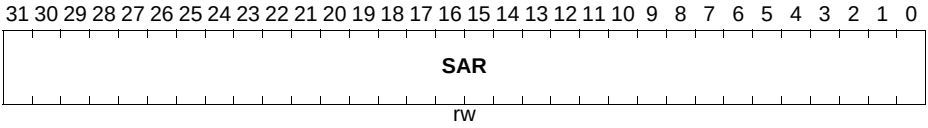
The starting source address is programmed by software before the DMA channel is enabled, or by an LLI update before the start of the DMA transfer. While the DMA transfer is in progress, this register is updated to reflect the source address of the current AHB transfer.

*Note: You must program the SAR address to be aligned to **CTL**.SRC_TR_WIDTH.*

For information on how the **SAR** is updated at the start of each DMA block for multi-block transfers, refer to **Table 5-5**.

GPDMA0_CHx_SAR (x=0-7)
Source Address Register for Channel x

(00_H + x*58_H) **Reset Value: 0000 0000_H**



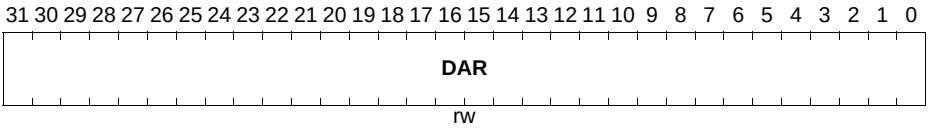
Field	Bits	Type	Description
SAR	[31:0]	rw	**Current Source Address of DMA transfer** Updated after each source transfer. The SINC field in the **CTL** register determines whether the address increments, decrements, or is left unchanged on every source transfer throughout the block transfer. Reset: 0_D

DAR

The starting destination address is programmed by software before the DMA channel is enabled, or by an LLI update before the start of the DMA transfer. While the DMA transfer is in progress, this register is updated to reflect the destination address of the current AHB transfer.

*Note: You must program the DAR to be aligned to **CTL**.DST_TR_WIDTH.*

GPDMA0_CHx_DAR (x=0-7)
Destination Address Register for Channel x
(08_H + x*58_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
DAR	[31:0]	rw	**Current Destination address of DMA transfer** Updated after each destination transfer. The DINC field in the **CTL** register determines whether the address increments, decrements, or is left unchanged on every destination transfer throughout the block transfer. Reset: 0_D

Hardware Realignment of SAR/DAR Registers

In a particular circumstance, during contiguous multi-block DMA transfers, the destination address can become misaligned between the end of one block and the start of the next block. When this situation occurs, GPDMA re-aligns the destination address before the start of the next block.

Consider the following example. If the block length is 9, the source transfer width is 16 (halfword), and the destination transfer width is 32 (word) — the destination is programmed for contiguous block transfers — then the destination performs four word transfers followed by a halfword transfer to complete the block transfer to the destination. At the end of the destination block transfer, the address is aligned to a 16-bit transfer as the last AMBA transfer is halfword. This is misaligned to the programmed transfer size of 32 bits for the destination. However, for contiguous destination multi-block transfers, GPDMA re-aligns the DAR address to the nearest 32-bit address (next 32-bit address upwards if address control is incrementing or next address downwards if address control is decrementing).

The destination address is automatically realigned by the GPDMA in the following DMA transfer setup scenario:

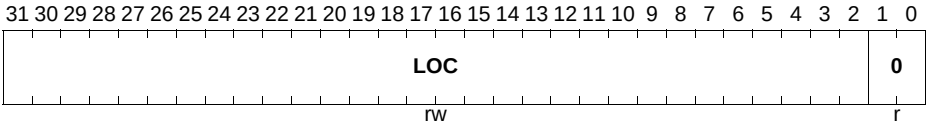
- Contiguous multi-block transfers on destination side, AND
- DST_TR_WIDTH > SRC_TR_WIDTH, AND
- (BLOCK_TS * SRC_TR_WIDTH)/DST_TR_WIDTH != integer (where SRC_TR_WIDTH, DST_TR_WIDTH is byte width of transfer)

LLP

You need to program this register to point to the first Linked List Item (LLI) in memory prior to enabling the channel if block chaining is enabled.

GPDMA0_CHx_LLP (x = 0-1)
Linked List Pointer Register for Channel x

(10_H + x*58_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
LOC	[31:2]	rw	**Starting Address In Memory** of next LLI if block chaining is enabled. Note that the two LSBs of the starting address are not stored because the address is assumed to be aligned to a 32-bit boundary.
0	[1:0]	r	**Reserved**

The LLP register has two functions:

- The logical result of the equation LLP.LOC != 0 is used to set up the type of DMA transfer — single or multi-block. **Table 5-5** shows how the method of updating the channel registers is a function of LLP.LOC != 0. If LLP.LOC is set to 0, then transfers using linked lists are not enabled. This register must be programmed prior to enabling the channel in order to set up the transfer type.
- LLP.LOC != 0 contains the pointer to the next LLI for block chaining using linked lists. The **LLP** register can also point to the address where write-back of the control and source/destination status information occur after block completion.

CTL

These registers contain fields that control the DMA transfer.

The CTLH and CTLL registers are part of the block descriptor (linked list item - LLI) when block chaining is enabled. It can be varied on a block-by-block basis within a DMA transfer when block chaining is enabled.

If status write-back is enabled, the upper control register, CTLH, is written to the control register location of the LLI in system memory at the end of the block transfer.

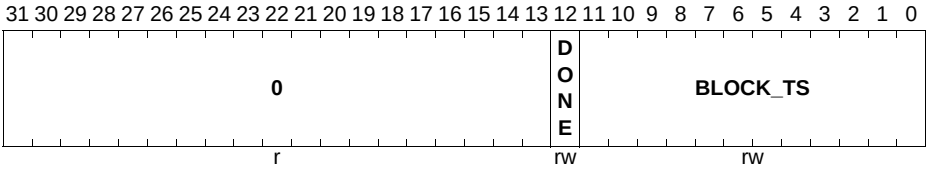
Note: You need to program these registers prior to enabling the channel.

CTLH

Control Register High.

GPDMA0_CHx_CTLH (x=0-7)
Control Register High for Channel x

($1C_H$ + x*58_H) **Reset Value: 0000 0002$_H$**



Field	Bits	Type	Description
BLOCK_TS	[11:0]	rw	**Block Transfer Size** When the GPDMA is the flow controller, the user writes this field before the channel is enabled in order to indicate the block size. The number programmed here indicates the total number of single transactions to perform for every block transfer. Once the transfer starts, the read-back value is the total number of data items already read from the source peripheral. *Note: The width of the single transaction is determined by **CTL**.SRC_TR_WIDTH.*
DONE	12	rw	**Done bit** If this bit is set and status write-back is enabled then CTLH is written to the control register location of the Linked List Item (LLI) in system memory at the end of the block transfer. Software can poll the LLI CTLH.DONE bit to see when a block transfer is complete. The LLI CTLH.DONE bit should be cleared when the linked lists are set up in memory prior to enabling the channel.
0	[31:13]	r	**Reserved**

CTLL

Control Register Low.

GPDMA0_CHx_CTLL (x=0-1)
Control Register Low for Channel x

(18_H + x*58_H) **Reset Value: 0030 4801$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0			LLP_SRC_EN	LLP_DST_EN	0				TT_FC			0	DST_SCATTER_EN	SRC_GATHER_EN	SRC_MSIZE
r			rw	rw	r				rw			r	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SRC_MSIZE		DEST_MSIZE			SINC		DINC		SRC_TR_WIDTH			DST_TR_WIDTH			INT_EN
rw		rw			rw		rw		rw			rw			rw

Field	Bits	Type	Description
INT_EN	0	rw	**Interrupt Enable Bit** If set, then all interrupt-generating sources are enabled. Functions as a global mask bit for all interrupts for the channel; Raw* interrupt registers still assert if INT_EN = 0.
DST_TR_WIDTH	[3:1]	rw	**Destination Transfer Width** Table 5-9 lists the decoding for this field.
SRC_TR_WIDTH	[6:4]	rw	**Source Transfer Width** Table 5-9 lists the decoding for this field.

Field	Bits	Type	Description
DINC	[8:7]	rw	**Destination Address Increment** Indicates whether to increment or decrement the destination address on every destination transfer. If your device is writing data to a destination peripheral FIFO with a fixed address, then set this field to "No change". 00_B Increment 01_B Decrement $1x_B$ No change *Note: Incrementing or decrementing is done for alignment to the next CTLL.DST_TR_WIDTH boundary.*
SINC	[10:9]	rw	**Source Address Increment** Indicates whether to increment or decrement the source address on every source transfer. If the device is fetching data from a source peripheral FIFO with a fixed address, then set this field to "No change". 00_B Increment 01_B Decrement $1x_B$ No change *Note: Incrementing or decrementing is done for alignment to the next CTLL.SRC_TR_WIDTH boundary.*
DEST_MSIZE	[13:11]	rw	**Destination Burst Transaction Length** Number of data items, each of width DST_TR_WIDTH, to be written to the destination every time a destination burst transaction request is made from either the corresponding hardware or software handshaking interface. **Table 5-8** lists the decoding for this field. *Note: This value is not related to the AHB bus master HBURST bus.*

Field	Bits	Type	Description
SRC_MSIZE	[16:14]	rw	**Source Burst Transaction Length** Number of data items, each of width SRC_TR_WIDTH, to be read from the source every time a source burst transaction request is made from either the corresponding hardware or software handshaking interface. **Table 5-8** lists the decoding for this field; *Note: This value is not related to the AHB bus master HBURST bus.*
SRC_GATHER_EN	17	rw	**Source gather enable** 0_B Gather disabled 1_B Gather enabled Gather on the source side is applicable only when the SINC bit indicates an incrementing or decrementing address control.
DST_SCATTER_EN	18	rw	**Destination scatter enable** 0_B Scatter disabled 1_B Scatter enabled Scatter on the destination side is applicable only when the DINC bit indicates an incrementing or decrementing address control.
TT_FC	[22:20]	rw	**Transfer Type and Flow Control** The following transfer types are supported. • Memory to Memory • Memory to Peripheral • Peripheral to Memory • Peripheral to Peripheral **Table 5-10** lists the decoding for this field.
LLP_DST_EN	27	rw	**Linked List Pointer for Destination Enable** Block chaining is enabled on the destination side only if the LLP_DST_EN field is high and LLP.LOC is non-zero.
LLP_SRC_EN	28	rw	**Linked List Pointer for Source Enable** Block chaining is enabled on the source side only if the LLP_SRC_EN field is high and LLP.LOC is non-zero.
0	[31:29], [26:23], 19	r	**Reserved**

GPDMA0_CHx_CTLL (x=2-7)
Control Register Low for Channel x

(18_H + x*58_H) **Reset Value: 0030 4801_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0									TT_FC			0			SRC_MSIZE
r									rw			r			rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SRC_MSIZE		DEST_MSIZE			SINC		DINC		SRC_TR_WIDTH			DST_TR_WIDTH			INT_EN
rw		rw			rw		rw		rw			rw			rw

Field	Bits	Type	Description
INT_EN	0	rw	**Interrupt Enable Bit** If set, then all interrupt-generating sources are enabled. Functions as a global mask bit for all interrupts for the channel; Raw* interrupt registers still assert if INT_EN = 0.
DST_TR_WIDTH	[3:1]	rw	**Destination Transfer Width** Table 5-9 lists the decoding for this field.
SRC_TR_WIDTH	[6:4]	rw	**Source Transfer Width** Table 5-9 lists the decoding for this field.
DINC	[8:7]	rw	**Destination Address Increment** Indicates whether to increment or decrement the destination address on every destination transfer. If your device is writing data to a destination peripheral FIFO with a fixed address, then set this field to "No change". 00_B Increment 01_B Decrement $1x_B$ No change *Note: Incrementing or decrementing is done for alignment to the next CTLL.DST_TR_WIDTH boundary.*

Field	Bits	Type	Description
SINC	[10:9]	rw	**Source Address Increment** Indicates whether to increment or decrement the source address on every source transfer. If the device is fetching data from a source peripheral FIFO with a fixed address, then set this field to "No change". 00_B Increment 01_B Decrement $1x_B$ No change *Note: Incrementing or decrementing is done for alignment to the next CTLL.SRC_TR_WIDTH boundary.*
DEST_MSIZE	[13:11]	rw	**Destination Burst Transaction Length** Number of data items, each of width DST_TR_WIDTH, to be written to the destination every time a destination burst transaction request is made from either the corresponding hardware or software handshaking interface. **Table 5-8** lists the decoding for this field. *Note: This value is not related to the AHB bus master HBURST bus.*
SRC_MSIZE	[16:14]	rw	**Source Burst Transaction Length** Number of data items, each of width SRC_TR_WIDTH, to be read from the source every time a source burst transaction request is made from either the corresponding hardware or software handshaking interface. **Table 5-8** lists the decoding for this field. *Note: This value is not related to the AHB bus master HBURST bus.*
TT_FC	[22:20]	rw	**Transfer Type and Flow Control** The following transfer types are supported. • Memory to Memory • Memory to Peripheral • Peripheral to Memory • Peripheral to Peripheral **Table 5-10** lists the decoding for this field.
0	[31:23], [19:17]	r	**Reserved**

Table 5-8 CTLL.SRC_MSIZE and CTLL.DST_MSIZE Field Decoding

CTLL.SRC_MSIZE / CTLL.DEST_MSIZE	Number of data items to be transferred(of width CTLL.SRC_TR_WIDTH or CTLL.DST_TR_WIDTH)
000_B	1
001_B	4
010_B	8
others	reserved

Table 5-9 CTLL.SRC_TR_WIDTH and CTLL.DST_TR_WIDTH Field Decoding

CTLL.SRC_TR_WIDTH / CTLL.DST_TR_WIDTH	Size (bits)
000_B	8
001_B	16
010_B	32
others	reserved

Table 5-10 CTLL.TT_FC Field Decoding

CTLL.TT_FC Field	Transfer Type	Flow Controller
000_B	Memory to Memory	GPDMA
001_B	Memory to Peripheral	GPDMA
010_B	Peripheral to Memory	GPDMA
011_B	Peripheral to Peripheral	GPDMA
100_B	Peripheral to Memory	Peripheral
101_B	Peripheral to Peripheral	Source Peripheral
110_B	Memory to Peripheral	Peripheral
111_B	Peripheral to Peripheral	Destination Peripheral

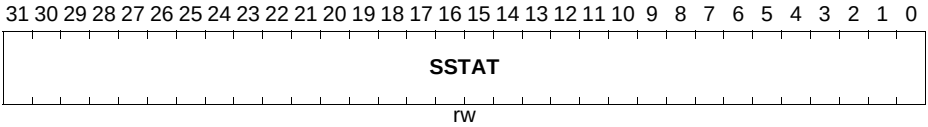
SSTAT

After each block transfer completes, hardware can retrieve the source status information from the address pointed to by the contents of the **SSTATAR** register. This status information is then stored in the SSTAT register and written out to the SSTAT register location of the LLI before the start of the next block.

Note: This register is a temporary placeholder for the source status information on its way to the SSTAT register location of the LLI. The source status information should be retrieved by software from the SSTAT register location of the LLI, and not by a read of this register over the GPDMA slave interface.

GPDMA0_CHx_SSTAT (x=0-1)
Source Status Register for Channel x

(20_H + x*58_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
SSTAT	[31:0]	rw	**Source Status** retrieved by hardware from the address pointed to by the contents of the **SSTATAR** register.

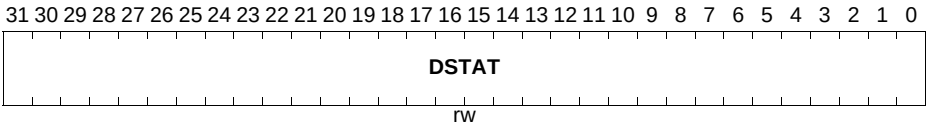
DSTAT

After the completion of each block transfer, hardware can retrieve the destination status information from the address pointed to by the contents of the **DSTATAR** register. This status information is then stored in the DSTAT register and written out to the DSTAT register location of the LLI before the start of the next block. This register does only exist for channels 0 and 1, for other channels the read-back value is always 0.

Note: This register is a temporary placeholder for the destination status information on its way to the DSTAT register location of the LLI. The destination status information should be retrieved by software from the DSTAT register location of the LLI and not by a read of this register over the GPDMA slave interface.

GPDMA0_CHx_DSTAT (x=0-1)
Destination Status Register for Channel x

(28_H + x*58_H) **Reset Value: 0000 0000_H**

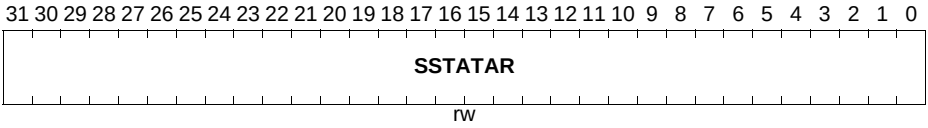


Field	Bits	Type	Description
DSTAT	[31:0]	rw	**Destination Status** retrieved by hardware from the address pointed to by the contents of the **DSTATAR** register.

SSTATAR

After the completion of each block transfer, hardware can retrieve the source status information from the address pointed to by the contents of the SSTATAR register.

GPDMA0_CHx_SSTATAR (x=0-1)
Source Status Address Register for Channel x
(30_H + x*58_H) **Reset Value: 0000 0000_H**

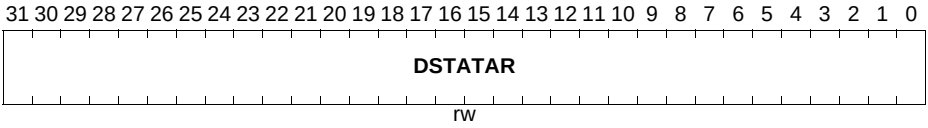


Field	Bits	Type	Description
SSTATAR	[31:0]	rw	**Source Status Address** Pointer from where hardware can fetch the source status information, which is registered in the **SSTAT** register and written out to the SSTAT register location of the LLI before the start of the next block.

DSTATAR

After the completion of each block transfer, hardware can retrieve the destination status information from the address pointed to by the contents of the DSTATAR register.

GPDMA0_CHx_DSTATAR (x=0-1)
Destination Status Address Register for Channel x
(38_H + x*58_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
DSTATAR	[31:0]	rw	**Destination Status Address** Pointer from where hardware can fetch the destination status information, which is registered in the **DSTAT** register and written out to the DSTAT register location of the LLI before the start of the next block.

CFG

These registers contain fields that configure the DMA transfer. The channel configuration register remains fixed for all blocks of a multi-block transfer.

Note: You need to program this register prior to enabling the channel.

GPDMA0_CHx_CFGH (x=0-1)
Configuration Register High for Channel x

(44_H + x*58_H) **Reset Value: 0000 0004_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	DEST_PER				SRC_PER				SS_UPD_EN	DS_UPD_EN	PROTCTL			FIFO_MODE	FCMODE
r	rw				rw				rw	rw	rw			rw	rw

Field	Bits	Type	Description
FCMODE	0	rw	**Flow Control Mode** Determines when source transaction requests are serviced when the Destination Peripheral is the flow controller. 0_B Source transaction requests are serviced when they occur. Data pre-fetching is enabled. 1_B Source transaction requests are not serviced until a destination transaction request occurs. In this mode, the amount of data transferred from the source is limited so that it is guaranteed to be transferred to the destination prior to block termination by the destination. Data pre-fetching is disabled.
FIFO_MODE	1	rw	**FIFO Mode Select** Determines how much space or data needs to be available in the FIFO before a burst transaction request is serviced. 0_B Space/data available for single AHB transfer of the specified transfer width. 1_B Data available is greater than or equal to half the FIFO depth for destination transfers and space available is greater than half the fifo depth for source transfers. The exceptions are at the end of a burst transaction request or at the end of a block transfer.
PROTCTL	[4:2]	rw	**Protection Control** Used to drive the AHB HPROT[3:1] bus. The AMBA Specification recommends that the default value of HPROT indicates a non-cached, non-buffered, privileged data access. The reset value is used to indicate such an access. HPROT[0] is tied high because all transfers are data accesses, as there are no opcode fetches. There is a one-to-one mapping of these register bits to the HPROT[3:1] master interface signals. **Table 5-11** shows the mapping of bits in this field to the AHB HPROT[3:1] bus.

Field	Bits	Type	Description
DS_UPD_EN	5	rw	**Destination Status Update Enable** Destination status information is fetched only from the location pointed to by the **DSTATAR** register, stored in the **DSTAT** register and written out to the DSTAT location of the LLI if DS_UPD_EN is high.
SS_UPD_EN	6	rw	**Source Status Update Enable** Source status information is fetched only from the location pointed to by the **SSTATAR** register, stored in the **SSTAT** register and written out to the **SSTAT** location of the LLI if SS_UPD_EN is high.
SRC_PER	[10:7]	rw	**Source Peripheral** Assigns a DLR line as hardware handshaking interface to the source of channel x 00_H assigns DLR line 0 01_H assigns DLR line 1 02_H assigns DLR line 2 03_H assigns DLR line 3 04_H assigns DLR line 4 05_H assigns DLR line 5 06_H assigns DLR line 6 07_H assigns DLR line 7 Other values not defined. **Notes** *1. For correct DMA operation, only one peripheral (source or destination) should be assigned to the same handshaking interface.* *2. If GPDMA0_CHx_CFGL.HS_SEL_SRC=1 this field is ignored.*

Field	Bits	Type	Description
DEST_PER	[14:11]	rw	**Destination Peripheral** Assigns a DLR line as hardware handshaking interface to the destination of channel x 00_H assigns DLR line 0 01_H assigns DLR line 1 02_H assigns DLR line 2 03_H assigns DLR line 3 04_H assigns DLR line 4 05_H assigns DLR line 5 06_H assigns DLR line 6 07_H assigns DLR line 7 Other values not defined. **Notes** *1. For correct DMA operation, only one peripheral (source or destination) should be assigned to the same handshaking interface.* *2. If GPDMA0_CHx_CFGL.HS_SEL_DST=1 this field is ignored.*
0	[31:15]	r	**Reserved**

GPDMA0_CHx_CFGH (x=2-7)
Configuration Register High for Channel x

(44_H + x*58_H) **Reset Value: 0000 0004_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	DEST_PER				SRC_PER				0		PROTCTL			FIFO_MODE	FCMODE
r	rw				rw				r		rw			rw	rw

Field	Bits	Type	Description
FCMODE	0	rw	**Flow Control Mode** Determines when source transaction requests are serviced when the Destination Peripheral is the flow controller. 0_B Source transaction requests are serviced when they occur. Data pre-fetching is enabled. 1_B Source transaction requests are not serviced until a destination transaction request occurs. In this mode, the amount of data transferred from the source is limited so that it is guaranteed to be transferred to the destination prior to block termination by the destination. Data pre-fetching is disabled.
FIFO_MODE	1	rw	**FIFO Mode Select** Determines how much space or data needs to be available in the FIFO before a burst transaction request is serviced. 0_B Space/data available for single AHB transfer of the specified transfer width. 1_B Data available is greater than or equal to half the FIFO depth for destination transfers and space available is greater than half the fifo depth for source transfers. The exceptions are at the end of a burst transaction request or at the end of a block transfer.
PROTCTL	[4:2]	rw	**Protection Control** Used to drive the AHB HPROT[3:1] bus. The AMBA Specification recommends that the default value of HPROT indicates a non-cached, non-buffered, privileged data access. The reset value is used to indicate such an access. HPROT[0] is tied high because all transfers are data accesses, as there are no opcode fetches. There is a one-to-one mapping of these register bits to the HPROT[3:1] master interface signals. **Table 5-11** shows the mapping of bits in this field to the AHB HPROT[3:1] bus.

Field	Bits	Type	Description
SRC_PER	[10:7]	rw	**Source Peripheral** Assigns a DLR line as hardware handshaking interface to the source of channel x 00_H assigns DLR line 0 01_H assigns DLR line 1 02_H assigns DLR line 2 03_H assigns DLR line 3 04_H assigns DLR line 4 05_H assigns DLR line 5 06_H assigns DLR line 6 07_H assigns DLR line 7 Other values not defined. **Notes** *1. For correct DMA operation, only one peripheral (source or destination) should be assigned to the same handshaking interface.* *2. If GPDMA0_CHx_CFGL.HS_SEL_SRC=1 this field is ignored.*

Field	Bits	Type	Description
DEST_PER	[14:11]	rw	**Destination Peripheral** Assigns a DLR line as hardware handshaking interface to the destination of channel x 00_H assigns DLR line 0 01_H assigns DLR line 1 02_H assigns DLR line 2 03_H assigns DLR line 3 04_H assigns DLR line 4 05_H assigns DLR line 5 06_H assigns DLR line 6 07_H assigns DLR line 7 Other values not defined. **Notes** *1. For correct DMA operation, only one peripheral (source or destination) should be assigned to the same handshaking interface.* *2. If GPDMA0_CHx_CFGL.HS_SEL_DST=1 this field is ignored.*
0	[31:15], [6:5]	r	**Reserved**

GPDMA0_CHx_CFGL (x=0-1)
Configuration Register Low for Channel x

(40_H + x*58_H) **Reset Value: 0000 0EX0$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RELOAD_DST	RELOAD_SRC	MAX_ABRST										SRC_HS_POL	DST_HS_POL	LOCK_B	LOCK_CH
rw	rw	rw										rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LOCK_B_L		LOCK_CH_L		HS_SEL_SRC	HS_SEL_DST	FIFO_EMPTY	CH_SUSP	CH_PRIOR			0				
rw		rw		rw	rw	r	rw	rw			r				

Field	Bits	Type	Description
CH_PRIOR	[7:5]	rw	**Channel priority** A priority of 7 is the highest priority, and 0 is the lowest. The value programmed to this field must be within 0 and 7. A programmed value outside this range will cause erroneous behavior. Reset: Channel Number For example: Chan0 = 000_B Chan1 = 001_B
CH_SUSP	8	rw	**Channel Suspend** Suspends all DMA data transfers from the source until this bit is cleared. There is no guarantee that the current transaction will complete. Can also be used in conjunction with CFGLx.FIFO_EMPTY to cleanly disable a channel without losing any data. 0_B Not suspended. 1_B Suspend DMA transfer from the source.
FIFO_EMPTY	9	r	**Indicates if there is data left in the channel FIFO** Can be used in conjunction with CFGLx.CH_SUSP to cleanly disable a channel. 1_B Channel FIFO empty 0_B Channel FIFO not empty
HS_SEL_DST	10	rw	**Destination Software or Hardware Handshaking Select** This register selects which of the handshaking interfaces - hardware or software - is active for destination requests on this channel. 0_B Hardware handshaking interface. Software-initiated transaction requests are ignored. 1_B Software handshaking interface. Hardware-initiated transaction requests are ignored. If the destination peripheral is memory, then this bit is ignored.

Field	Bits	Type	Description
HS_SEL_SRC	11	rw	**Source Software or Hardware Handshaking Select** This register selects which of the handshaking interfaces - hardware or software - is active for source requests on this channel. 0_B Hardware handshaking interface. Software-initiated transaction requests are ignored. 1_B Software handshaking interface. Hardware-initiated transaction requests are ignored. If the source peripheral is memory, then this bit is ignored.
LOCK_CH_L	[13:12]	rw	**Channel Lock Level** Indicates the duration over which CFGLx.LOCK_CH bit applies. 00_B Over complete DMA transfer 01_B Over complete DMA block transfer $1x_B$ Over complete DMA transaction
LOCK_B_L	[15:14]	rw	**Bus Lock Level** Indicates the duration over which CFGLx.LOCK_B bit applies. 00_B Over complete DMA transfer 01_B Over complete DMA block transfer $1x_B$ Over complete DMA transaction
LOCK_CH	16	rw	**Channel Lock Bit** When the channel is granted control of the master bus interface and if the CFGLx.LOCK_CH bit is asserted, then no other channels are granted control of the master bus interface for the duration specified in CFGLx.LOCK_CH_L. Indicates to the master bus interface arbiter that this channel wants exclusive access to the master bus interface for the duration specified in CFGLx.LOCK_CH_L.
LOCK_B	17	rw	**Bus Lock Bit** When active, the AHB bus master signal hlock is asserted for the duration specified in CFGLx.LOCK_B_L. For more information, refer to **Section 5.2.6**.

Field	Bits	Type	Description
DST_HS_POL	18	rw	**Destination Handshaking Interface Polarity** 0_B Active high 1_B Active low For information on this, refer to **Section 5.2.4**.
SRC_HS_POL	19	rw	**Source Handshaking Interface Polarity** 0_B Active high 1_B Active low For information on this, refer to **Section 5.2.4**.
MAX_ABRST	[29:20]	rw	**Maximum AMBA Burst Length** Maximum AMBA burst length that is used for DMA transfers on this channel. A value of 0 indicates that software is not limiting the maximum AMBA burst length for DMA transfers on this channel.
RELOAD_SRC	30	rw	**Automatic Source Reload** The **SAR** register can be automatically reloaded from its initial value at the end of every block for multi-block transfers. A new block transfer is then initiated. For conditions under which this occurs, refer to **Table 5-5**.
RELOAD_DST	31	rw	**Automatic Destination Reload** The **DAR** register can be automatically reloaded from its initial value at the end of every block for multi-block transfers. A new block transfer is then initiated. For conditions under which this occurs, refer to **Table 5-5**.
0	[4:0]	r	**Reserved**

GPDMA0_CHx_CFGL (x=2-7)
Configuration Register Low for Channel x

(40_H + x*58_H) **Reset Value: 0000 0EX0$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0		MAX_ABRST										SRC_HS_POL	DST_HS_POL	LOCK_B	LOCK_CH
r		rw										rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LOCK_B_L		LOCK_CH_L		HS_SEL_SRC	HS_SEL_DST	FIFO_EMPTY	CH_SUSP	CH_PRIOR			0				
rw		rw		rw	rw	r	rw	rw			r				

Field	Bits	Type	Description
CH_PRIOR	[7:5]	rw	**Channel priority** A priority of 7 is the highest priority, and 0 is the lowest. The value programmed to this field must be within 0 and 7. A programmed value outside this range will cause erroneous behavior. Reset: Channel Number For example: Chan0 = 000_B Chan1 = 001_B
CH_SUSP	8	rw	**Channel Suspend** Suspends all DMA data transfers from the source until this bit is cleared. There is no guarantee that the current transaction will complete. Can also be used in conjunction with CFGLx.FIFO_EMPTY to cleanly disable a channel without losing any data. 0_B Not suspended. 1_B Suspend DMA transfer from the source.
FIFO_EMPTY	9	r	**Indicates if there is data left in the channel FIFO** Can be used in conjunction with CFGLx.CH_SUSP to cleanly disable a channel. 1_B Channel FIFO empty 0_B Channel FIFO not empty

Field	Bits	Type	Description
HS_SEL_DST	10	rw	**Destination Software or Hardware Handshaking Select** This register selects which of the handshaking interfaces - hardware or software - is active for destination requests on this channel. 0_B Hardware handshaking interface. Software-initiated transaction requests are ignored. 1_B Software handshaking interface. Hardware-initiated transaction requests are ignored. If the destination peripheral is memory, then this bit is ignored.
HS_SEL_SRC	11	rw	**Source Software or Hardware Handshaking Select** This register selects which of the handshaking interfaces - hardware or software - is active for source requests on this channel. 0_B Hardware handshaking interface. Software-initiated transaction requests are ignored. 1_B Software handshaking interface. Hardware-initiated transaction requests are ignored. If the source peripheral is memory, then this bit is ignored.
LOCK_CH_L	[13:12]	rw	**Channel Lock Level** Indicates the duration over which CFGLx.LOCK_CH bit applies. 00_B Over complete DMA transfer 01_B Over complete DMA block transfer $1x_B$ Over complete DMA transaction
LOCK_B_L	[15:14]	rw	**Bus Lock Level** Indicates the duration over which CFGLx.LOCK_B bit applies. 00_B Over complete DMA transfer 01_B Over complete DMA block transfer $1x_B$ Over complete DMA transaction

Field	Bits	Type	Description
LOCK_CH	16	rw	**Channel Lock Bit** When the channel is granted control of the master bus interface and if the CFGLx.LOCK_CH bit is asserted, then no other channels are granted control of the master bus interface for the duration specified in CFGLx.LOCK_CH_L. Indicates to the master bus interface arbiter that this channel wants exclusive access to the master bus interface for the duration specified in CFGLx.LOCK_CH_L.
LOCK_B	17	rw	**Bus Lock Bit** When active, the AHB bus master signal hlock is asserted for the duration specified in CFGLx.LOCK_B_L. For more information, refer to **Section 5.2.6**.
DST_HS_POL	18	rw	**Destination Handshaking Interface Polarity** 0_B Active high 1_B Active low For information on this, refer to **Section 5.2.4**.
SRC_HS_POL	19	rw	**Source Handshaking Interface Polarity** 0_B Active high 1_B Active low For information on this, refer to **Section 5.2.4**.
MAX_ABRST	[29:20]	rw	**Maximum AMBA Burst Length** Maximum AMBA burst length that is used for DMA transfers on this channel. A value of 0 indicates that software is not limiting the maximum AMBA burst length for DMA transfers on this channel.
0	[31:30], [4:0]	r	**Reserved**

Table 5-11 PROTCTL field to HPROT Mapping

1_B	HPROT[0]
CFGHx.PROTCTL[1]	HPROT[1]
CFGHx.PROTCTL[2]	HPROT[2]
CFGHx.PROTCTL[3]	HPROT[3]

SGR

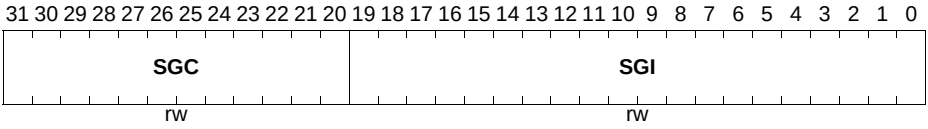
The Source Gather register contains two fields:

- Source gather count field (SGRx.SGC) - Specifies the number of contiguous source transfers of **CTL**.SRC_TR_WIDTH between successive gather intervals. This is defined as a gather boundary.
- Source gather interval field (SGRx.SGI) - Specifies the source address increment/decrement in multiples of **CTL**.SRC_TR_WIDTH on a gather boundary when gather mode is enabled for the source transfer.

The **CTL**.SINC field controls whether the address increments or decrements. When the **CTL**.SINC field indicates a fixed-address control, then the address remains constant throughout the transfer and the SGR register is ignored. For more information, see **Section 5.2.7**.

GPDMA0_CHx_SGR (x=0-1)
Source Gather Register for Channel x

(48_H + x*58_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
SGI	[19:0]	rw	**Source gather interval**
SGC	[31:20]	rw	**Source gather count** Source contiguous transfer count between successive gather boundaries.

DSR

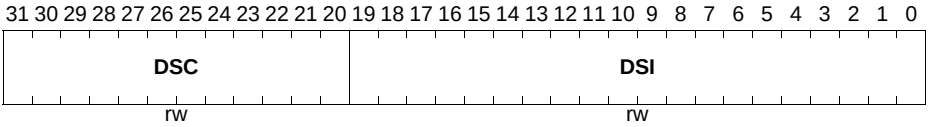
The Destination Scatter register contains two fields:

- Destination scatter count field (DSRx.DSC) - Specifies the number of contiguous destination transfers of **CTL**.DST_TR_WIDTH between successive scatter boundaries.
- Destination scatter interval field (DSRx.DSI) - Specifies the destination address increment/decrement in multiples of **CTL**.DST_TR_WIDTH on a scatter boundary when scatter mode is enabled for the destination transfer.

The **CTL**.DINC field controls whether the address increments or decrements. When the **CTL**.DINC field indicates a fixed address control, then the address remains constant throughout the transfer and the DSR register is ignored. For more information, see **Section 5.2.7**.

GPDMA0_CHx_DSR (x=0-1)
Destination Scatter Register for Channel x

(50_H + x*58_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
DSI	[19:0]	rw	**Destination scatter interval**
DSC	[31:20]	rw	**Destination scatter count** Destination contiguous transfer count between successive scatter boundaries.

5.8.3 Interrupt Registers

The following sections describe the registers pertaining to interrupts, their status, and how to clear them. For each channel, there are five types of interrupt sources:

- IntBlock - Block Transfer Complete Interrupt. This interrupt is generated on DMA block transfer completion to the destination peripheral.
- IntDstTran - Destination Transaction Complete Interrupt
  This interrupt is generated after completion of the last AHB transfer of the requested single/burst transaction from the handshaking interface (either the hardware or software handshaking interface) on the destination side.

Note: If the destination for a channel is memory, then that channel will never generate the IntDstTran interrupt. Because of this, the corresponding bit in this field will not be set.

- IntErr - Error Interrupt
  This interrupt is generated when an ERROR response is received from an AHB slave on the HRESP bus during a DMA transfer. In addition, the DMA transfer is cancelled and the channel is disabled.
- IntSrcTran - Source Transaction Complete Interrupt
  This interrupt is generated after completion of the last AHB transfer of the requested single/burst transaction from the handshaking interface (either the hardware or software handshaking interface) on the source side.

Note: If the source or destination is memory, then IntSrcTran/IntDstTran interrupts should be ignored, as there is no concept of a "DMA transaction level" for memory.

- IntTfr - DMA Transfer Complete Interrupt
  This interrupt is generated on DMA transfer completion to the destination peripheral.

There are several groups of interrupt-related registers:

- **Interrupt Raw Status Registers**
- **Interrupt Status Registers**
- **Interrupt Mask Registers**
- **Interrupt Clear Registers**
- **Combined Interrupt Status Register**

When a channel has been enabled to generate interrupts, the following is true:

- Interrupt events are stored in the Raw Status registers.
- The contents of the Raw Status registers are masked with the contents of the Mask registers.
- The masked interrupts are stored in the Status registers.
- The contents of the Status registers are used to drive the int_* port signals.
- Writing to the appropriate bit in the Clear registers clears an interrupt in the Raw Status registers and the Status registers on the same clock cycle.

The contents of each of the five Status registers is ORed to produce a single bit for each interrupt type in the Combined Status register; that is, STATUSINT.

Note: For interrupts to propagate past the raw interrupt register stage, **CTL**.INT_EN must be set to 1_B, and the relevant interrupt must be unmasked in the mask* interrupt register.*

Interrupt Raw Status Registers

Interrupt events are stored in these Raw Interrupt Status registers before masking: RAWBLOCK, RawDstTran, RawErr, RawSrcTran, and RAWTFR. Each Raw Interrupt Status register has a bit allocated per channel; for example, RAWTFR[2] is the Channel 2 raw transfer complete interrupt.

Each bit in these registers is cleared by writing a 1 to the corresponding location in the CLEARTFR, CLEARBLOCK, CLEARSRCTRAN, CLEARDSTTRAN, CLEARERR registers.

Note: Write access is available to these registers for software testing purposes only. Under normal operation, writes to these registers are not recommended.

RAWTFR

Raw DMA Transfer Complete Interrupt Status.

RAWBLOCK

Raw Block Transfer Complete Interrupt Status.

RAWSRCTRAN

Raw Source Transaction Complete Interrupt Status.

RAWDSTTRAN

Raw Destination Transaction Complete Interrupt Status.

RAWERR

Raw Error Interrupt Status.

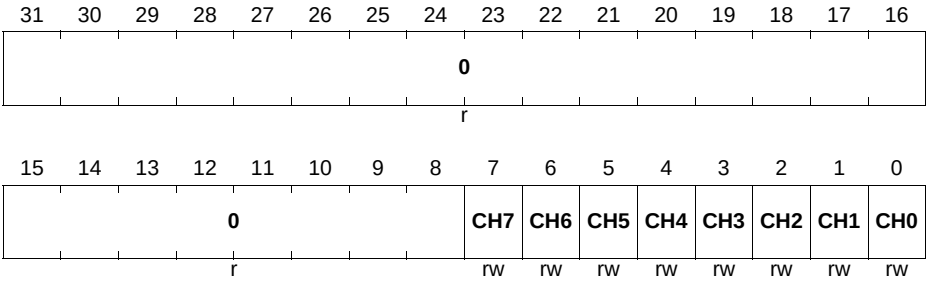
GPDMA0_RAWTFR
Raw IntTfr Status **($2C0_H$)** **Reset Value: 0000 0000$_H$**

GPDMA0_RAWBLOCK
Raw IntBlock Status **($2C8_H$)** **Reset Value: 0000 0000$_H$**

GPDMA0_RAWSRCTRAN
Raw IntSrcTran Status **($2D0_H$)** **Reset Value: 0000 0000$_H$**

GPDMA0_RAWDSTTRAN
Raw IntBlock Status **($2D8_H$)** **Reset Value: 0000 0000$_H$**

GPDMA0_RAWERR
Raw IntErr Status **($2E0_H$)** **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
CHx (x=0-7)	x	rw	**Raw Interrupt Status for channel x**
0	[31:8]	r	**Reserved**

Interrupt Status Registers

All interrupt events from all channels are stored in these Interrupt Status registers after masking: STATUSBLOCK, STATUSDSTTRAN, STATUSERR, STATUSSRCTRAN, and STATUSTFR. Each Interrupt Status register has a bit allocated per channel; for example, STATUSTFR[2] is the Channel 2 status transfer complete interrupt. The contents of these registers are used to generate the interrupt signals (int or int_n bus, depending on interrupt polarity) leaving the GPDMA.

STATUSTFR

DMA Transfer Complete Interrupt Status.

STATUSBLOCK

Block Transfer Complete Interrupt Status.

STATUSSRCTRAN

Source Transaction Complete Interrupt Status.

STATUSDSTTRAN

Block Transfer Complete Interrupt Status.

STATUSERR

Error Interrupt Status.

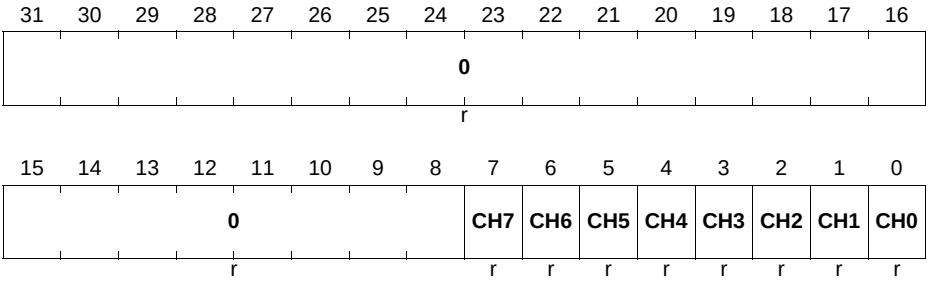
GPDMA0_STATUSTFR
IntTfr Status **($2E8_H$)** **Reset Value: 0000 0000$_H$**

GPDMA0_STATUSBLOCK
IntBlock Status **($2F0_H$)** **Reset Value: 0000 0000$_H$**

GPDMA0_STATUSSRCTRAN
IntSrcTran Status **($2F8_H$)** **Reset Value: 0000 0000$_H$**

GPDMA0_STATUSDSTTRAN
IntBlock Status **(300_H)** **Reset Value: 0000 0000$_H$**

GPDMA0_STATUSERR
IntErr Status **(308_H)** **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
CHx (x=0-7)	x	r	**Interrupt Status for channel x**
0	[31:8]	r	**Reserved**

Interrupt Mask Registers

The contents of the Raw Status registers are masked with the contents of the Mask registers: MASKBLOCK, MASKDSTTRAN, MASKERR, MASKSRCTRAN, and MASKTFR. Each Interrupt Mask register has a bit allocated per channel; for example, MASKTFR[2] is the mask bit for the Channel 2 transfer complete interrupt.

When the source peripheral of DMA channel i is memory, then the source transaction complete interrupt, MASKSRCTRAN[z], must be masked to prevent an erroneous triggering of an interrupt on the int_combined signal. Similarly, when the destination peripheral of DMA channel i is memory, then the destination transaction complete interrupt, MASKDSTTRAN[i], must be masked to prevent an erroneous triggering of an interrupt on the int_combined(_n) signal.

A channel INT_MASK bit will be written only if the corresponding mask write enable bit in the INT_MASK_WE field is asserted on the same AHB write transfer. This allows software to set a mask bit without performing a read-modified write operation. For example, writing hex 01x1 to the MASKTFR register writes a 1 into MASKTFR[0], while MASKTFR[7:1] remains unchanged. Writing hex 00xx leaves MASKTFR[7:0] unchanged.

Writing a 1 to any bit in these registers unmasks the corresponding interrupt, thus allowing the GPDMA to set the appropriate bit in the Status registers and int_* port signals.

MASKTFR

Mask for Raw DMA Transfer Complete Interrupt Status.

MASKBLOCK

Mask for Raw Block Transfer Complete Interrupt Status.

MASKSRCTRAN

Mask for Raw Source Transaction Complete Interrupt Status.

MASKDSTTRAN

Mask for Raw Block Transfer Complete Interrupt Status.

MASKERR

Mask for Raw Error Interrupt Status.

GPDMA0_MASKTFR
Mask for Raw IntTfr Status **(310_H)** **Reset Value: 0000 0000$_H$**

GPDMA0_MASKBLOCK
Mask for Raw IntBlock Status **(318_H)** **Reset Value: 0000 0000$_H$**

GPDMA0_MASKSRCTRAN
Mask for Raw IntSrcTran Status **(320_H)** **Reset Value: 0000 0000$_H$**

GPDMA0_MASKDSTTRAN
Mask for Raw IntBlock Status **(328_H)** **Reset Value: 0000 0000$_H$**

GPDMA0_MASKERR
Mask for Raw IntErr Status **(330_H)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WE_CH7	WE_CH6	WE_CH5	WE_CH4	WE_CH3	WE_CH2	WE_CH1	WE_CH0	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0
w	w	w	w	w	w	w	w	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
CHx (x=0-7)	x	rw	**Mask bit for channel x** 0_B masked 1_B unmasked
WE_CHx (x=0-7)	8+x	w	**Write enable for mask bit of channel x** 0_B write disabled 1_B write enabled
0	[31:16]	r	**Reserved**

Interrupt Clear Registers

Each bit in the Raw Status and Status registers is cleared on the same cycle by writing a 1 to the corresponding location in the Clear registers: CLEARBLOCK, CLEARDSTTRAN, CLEARERR, CLEARSRCTRAN, and CLEARTFR. Each Interrupt Clear register has a bit allocated per channel; for example, CLEARTFR[2] is the clear bit for the Channel 2 transfer complete interrupt. Writing a 0 has no effect. These registers are not readable.

CLEARTFR

Clear DMA Transfer Complete Interrupt Status and Raw Status.

CLEARBLOCK

Clear Block Transfer Complete Interrupt Status and Raw Status.

CLEARSRCTRAN

Clear Source Transaction Complete Interrupt Status and Raw Status.

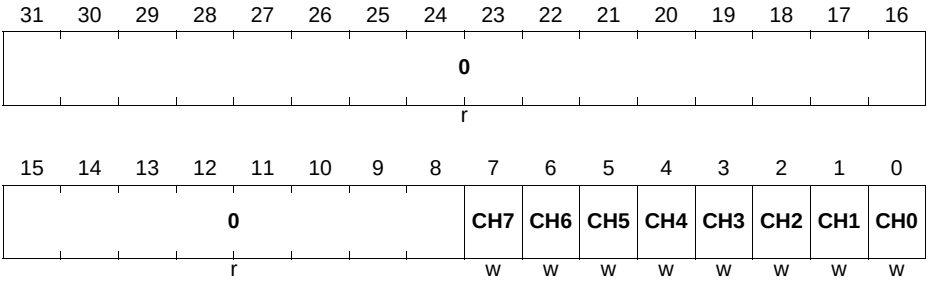
CLEARDSTTRAN

Clear Block Transfer Complete Interrupt Status and Raw Status.

CLEARERR

Clear Error Interrupt Status and Raw Status.

GPDMA0_CLEARTFR
IntTfr Status **(338_H)** **Reset Value: 0000 0000$_H$**
GPDMA0_CLEARBLOCK
IntBlock Status **(340_H)** **Reset Value: 0000 0000$_H$**
GPDMA0_CLEARSRCTRAN
IntSrcTran Status **(348_H)** **Reset Value: 0000 0000$_H$**
GPDMA0_CLEARDSTTRAN
IntBlock Status **(350_H)** **Reset Value: 0000 0000$_H$**
GPDMA0_CLEARERR
IntErr Status **(358_H)** **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
CHx (x=0-7)	x	w	**Clear Interrupt Status and Raw Status for channel x** 0_B no effect 1_B clear status
0	[31:8]	r	**Reserved**

Combined Interrupt Status Register

The contents of each of the five Status registers - STATUSTFR, STATUSBLOCK, STATUSSRCTRAN, STATUSDSTTRAN, STATUSERR - is ORed to produce a single bit for each interrupt type in the Combined Status register (STATUSINT). This register is read-only.

GPDMA0_STATUSINT
Combined Interrupt Status Register (360_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0											ERR	DSTT	SRCT	BLOCK	TFR
r											r	r	r	r	r

Field	Bits	Type	Description
TFR	0	r	**OR of the contents of STATUSTFR register**
BLOCK	1	r	**OR of the contents of STATUSBLOCK register**
SRCT	2	r	**OR of the contents of STATUSSRCTRAN register**
DSTT	3	r	**OR of the contents of STATUSDSTTRAN register**
ERR	4	r	**OR of the contents of STATUSERR register**
0	[31:5]	r	**Reserved**

5.8.4 Software Handshaking Registers

The registers that comprise the software handshaking registers allow software to initiate single or burst transaction requests in the same way that handshaking interface signals do in hardware.

Setting **CFG**.HS_SEL_SRC to 1 enables software handshaking on the source of channel x. Setting **CFG**.HS_SEL_DST to 1 enables software handshaking on the destination of channel x.

REQSRCREG

A bit is assigned for each channel in this register. REQSRCREG[n] is ignored when software handshaking is not enabled for the source of channel n.

A channel SRC_REQ bit is written only if the corresponding channel write enable bit in the SRC_REQ_WE field is asserted on the same AHB write transfer, and if the channel is enabled in the **CHENREG** register. For example, writing hex 0101 writes a 1 into REQSRCREG[0], while REQSRCREG[7:1] remains unchanged. Writing hex 00xx leaves REQSRCREG[7:0] unchanged. This allows software to set a bit in the REQSRCREG register without performing a read-modified write operation.

The functionality of this register depends on whether the source is a flow control peripheral or not.

GPDMA0_REQSRCREG
Source Software Transaction Request Register
(368_H) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WE_CH7	WE_CH6	WE_CH5	WE_CH4	WE_CH3	WE_CH2	WE_CH1	WE_CH0	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0
w	w	w	w	w	w	w	w	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
CHx (x=0-7)	x	rw	**Source request for channel x**
WE_CHx (x=0-7)	8+x	w	**Source request write enable for channel x** 0_B write disabled 1_B write enabled
0	[31:16]	r	**Reserved**

REQDSTREG

A bit is assigned for each channel in this register. REQDSTREG[n] is ignored when software handshaking is not enabled for the source of channel n.

A channel DST_REQ bit is written only if the corresponding channel write enable bit in the DST_REQ_WE field is asserted on the same AHB write transfer, and if the channel is enabled in the **CHENREG** register.

The functionality of this register depends on whether the destination is a flow control peripheral or not.

GPDMA0_REQDSTREG
Destination Software Transaction Request Register

(370_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WE_CH7	**WE_CH6**	**WE_CH5**	**WE_CH4**	**WE_CH3**	**WE_CH2**	**WE_CH1**	**WE_CH0**	**CH7**	**CH6**	**CH5**	**CH4**	**CH3**	**CH2**	**CH1**	**CH0**
w	w	w	w	w	w	w	w	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
CHx (x=0-7)	x	rw	**Source request for channel x**
WE_CHx (x=0-7)	8+x	w	**Source request write enable for channel x** 0_B write disabled 1_B write enabled
0	[31:16]	r	**Reserved**

SGLREQSRCREG

A bit is assigned for each channel in this register. SGLREQSRCREG[n] is ignored when software handshaking is not enabled for the source of channel n.

A channel SRC_SGLREQ bit is written only if the corresponding channel write enable bit in the SRC_SGLREQ_WE field is asserted on the same AHB write transfer, and if the channel is enabled in the **CHENREG** register.

The functionality of this register depends on whether the source is a flow control peripheral or not.

GPDMA0_SGLREQSRCREG
Single Source Transaction Request Register
(378_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WE_CH7	WE_CH6	WE_CH5	WE_CH4	WE_CH3	WE_CH2	WE_CH1	WE_CH0	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0
w	w	w	w	w	w	w	w	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
CHx (x=0-7)	x	rw	**Source request for channel x**
WE_CHx (x=0-7)	8+x	w	**Source request write enable for channel x** 0_B write disabled 1_B write enabled
0	[31:16]	r	**Reserved**

SGLREQDSTREG

A bit is assigned for each channel in this register. SGLREQDSTREG[n] is ignored when software handshaking is not enabled for the destination of channel n.

A channel DST_SGLREQ bit is written only if the corresponding channel write enable bit in the DST_SGLREQ_WE field is asserted on the same AHB write transfer, and if the channel is enabled in the **CHENREG** register.

The functionality of this register depends on whether the destination is a flow control peripheral or not.

GPDMA0_SGLREQDSTREG
Single Destination Transaction Request Register

(380$_H$) Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WE_CH7	WE_CH6	WE_CH5	WE_CH4	WE_CH3	WE_CH2	WE_CH1	WE_CH0	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0
w	w	w	w	w	w	w	w	rw	rw	rw	rw	rw	rw	rw	rw

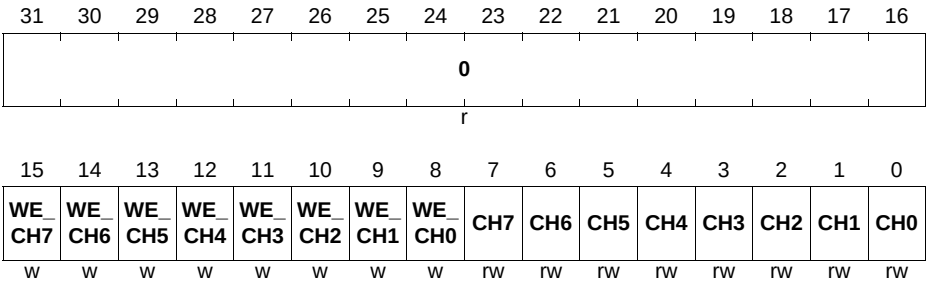
Field	Bits	Type	Description
CHx (x=0-7)	x	rw	**Source request for channel x**
WE_CHx (x=0-7)	8+x	w	**Source request write enable for channel x** 0_B write disabled 1_B write enabled
0	[31:16]	r	**Reserved**

LSTSRCREG

A bit is assigned for each channel in this register. LSTSRCREG[n] is ignored when software handshaking is not enabled for the source of channel n, or when the source of channel n is not a flow controller.

A channel LSTSRC bit is written only if the corresponding channel write enable bit in the LSTSRC_WE field is asserted on the same AHB write transfer, and if the channel is enabled in the **CHENREG** register.

GPDMA0_LSTSRCREG
Last Source Transaction Request Register
(388_H) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
CHx (x=0-7)	x	rw	**Source last request for channel x** 0_B Not last transaction in current block 1_B Last transaction in current block
WE_CHx (x=0-7)	8+x	w	**Source last transaction request write enable for channel x** 0_B write disabled 1_B write enabled
0	[31:16]	r	**Reserved**

LSTDSTREG

A bit is assigned for each channel in this register. LSTDSTREG[n] is ignored when software handshaking is not enabled for the destination of channel n or when the destination of channel n is not a flow controller.

A channel LSTDST bit is written only if the corresponding channel write enable bit in the LSTDST_WE field is asserted on the same AHB write transfer, and if the channel is enabled in the **CHENREG** register.

GPDMA0_LSTDSTREG

Last Destination Transaction Request Register

(390$_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WE_CH7	WE_CH6	WE_CH5	WE_CH4	WE_CH3	WE_CH2	WE_CH1	WE_CH0	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0
w	w	w	w	w	w	w	w	rw	rw	rw	rw	rw	rw	rw	rw

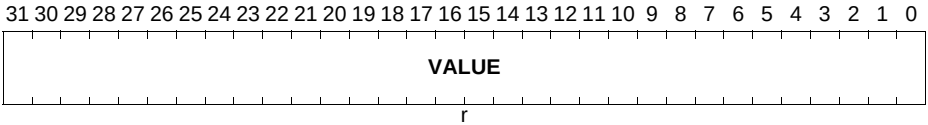
Field	Bits	Type	Description
CHx (x=0-7)	x	rw	**Destination last request for channel x** 0_B Not last transaction in current block 1_B Last transaction in current block
WE_CHx (x=0-7)	8+x	w	**Destination last transaction request write enable for channel x** 0_B write disabled 1_B write enabled
0	[31:16]	r	**Reserved**

5.8.5 Miscellaneous GPDMA Registers

ID

This is the GPDMA ID register, which is a read-only register that reads back the hardcoded module ID number.

GPDMA0_ID
GPDMA0 ID Register **($3A8_H$)** **Reset Value: $00AF\ C0XX_H$**

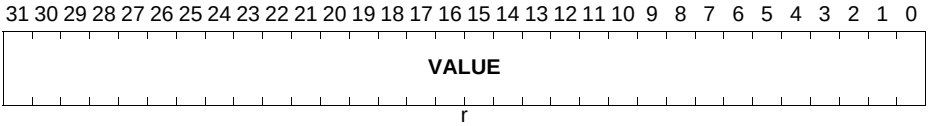


Field	Bits	Type	Description
VALUE	[31:0]	r	**Hardcoded GPDMA Peripheral ID**

TYPE

This is the GPDMA Component Type register, which is a read-only register that specifies the type of the packaged component.

GPDMA0_TYPE
GPDMA Component Type **($3F8_H$)** **Reset Value: $4457\ 1110_H$**

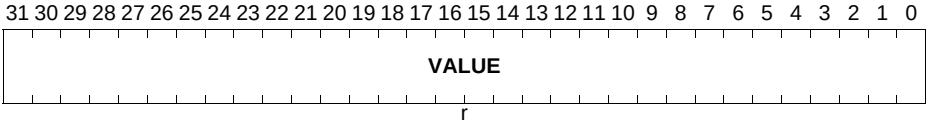


Field	Bits	Type	Description
VALUE	[31:0]	r	**Component Type** number = 44_57_11_10.

VERSION

This is the GPDMA Component Version register, which is a read-only register that specifies the version of the packaged component.

GPDMA0_VERSION
DMA Component Version **($3FC_H$)** **Reset Value: $3231\ 342A_H$**



Field	Bits	Type	Description
VALUE	[31:0]	r	**Version number of the component**

6 Flexible CRC Engine (FCE)

The FCE provides a parallel implementation of Cyclic Redundancy Code (CRC) algorithms. The current FCE version for the XMC4[12]00 microcontroller implements the IEEE 802.3 ethernet CRC32, the CCITT CRC16 and the SAE J1850 CRC8 polynomials. The primary target of FCE is to be used as an hardware acceleration engine for software applications or operating systems services using CRC signatures.

The FCE operates as a standard peripheral bus slave and is fully controlled through a set of configuration and control registers. The different CRC algorithms are independent from each other, they can be used concurrently by different software tasks.

Note: The FCE kernel register names described in ***"Registers" on Page 6-11*** *are referenced in a product Reference Manual by the module name prefix "FCE_".*

Input documents

[5] A painless guide to CRC Error Detection Algorithms, Ross N. Williams

[6] 32-Bit Cyclic Redundancy Codes for Internet Applications, Philip Koopman, International Conference on Dependable Systems and Networks (DSN), 2002

Related standards and norms

[7] IEEE 802.3 Ethernet 32-bits CRC

Table 6-1 FCE Abbreviations

CRC	Cyclic Redundancy Checksum
FCE	Flexible CRC Engine
IR	Input Register
RES	Result
STS	Status
CFG	Configuration

6.1 Overview

This section provides on overview of the features, applications and architecture of the FCE module.

6.1.1 Features

The FCE provides the following features:

- The FCE implements the following CRC polynomials:

  - CRC kernel 0 and 1: IEEE 802.3 CRC32 ethernet polynomial: 0x04C11DB7[1] - $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+x^{2}+x+1$
  - CRC kernel 2: CCITT CRC16 polynomial: 0x1021 - $x^{16}+x^{12}+x^{5}+1$
  - CRC kernel 3: SAE J1850 CRC8 polynomial: 0x1D - $x^{8}+x^{4}+x^{3}+x^{2}+1$
- Parallel CRC implementation
  - Data blocks to be computed by FCE shall be a multiple of the polynomial degree
  - Start address of Data blocks to be computed by FCE shall be aligned to the polynomial degree
- Register Interface:
  - Input Register
  - CRC Register
  - Configuration Registers enabling to control the CRC operation and perform automatic checksum checks at the end of a message.
  - Extended register interface to control reliability of FCE execution in safety applications.
- Error notification scheme via dedicated interrupt node for:
  - Transient error detection: error interrupt generation (maskable) with local status register (cleared by software)
  - Checksum failure: error interrupt generation (maskable) with local status register (cleared by software)
- FCE provides one interrupt line to the interrupt system. Each CRC engine has its own set of flag registers.

6.1.2 Application Mapping

Among other applications, CRC algorithms are commonly used to calculate message signatures to:

- Check message integrity during transport over communication channels like internal buses or interfaces between microcontrollers
- Sign blocks of data residing in variable or invariable storage elements
- Compute signatures for program flow monitoring

One important property to be taken into account by the application when choosing a polynomial is the hamming distance: see **Section 6.9**.

6.1.3 Block Diagram

The FCE is a standard peripheral slave module which is controlled over a set of memory mapped registers. The FCE is fully synchronous with the CPU clock and runs with a 1:1 clock ratio.

1) The polynomial hexadecimal representation covers the coefficients (degree - 1) down to 0.

Depending on the hardware configuration the FCE may implement more CRC kernels with different CRC polynomials. The specific configuration for the XMC4[12]00 microcontroller is shown in the **Figure 6-1 “FCE Block Diagram” on Page 6-3**.

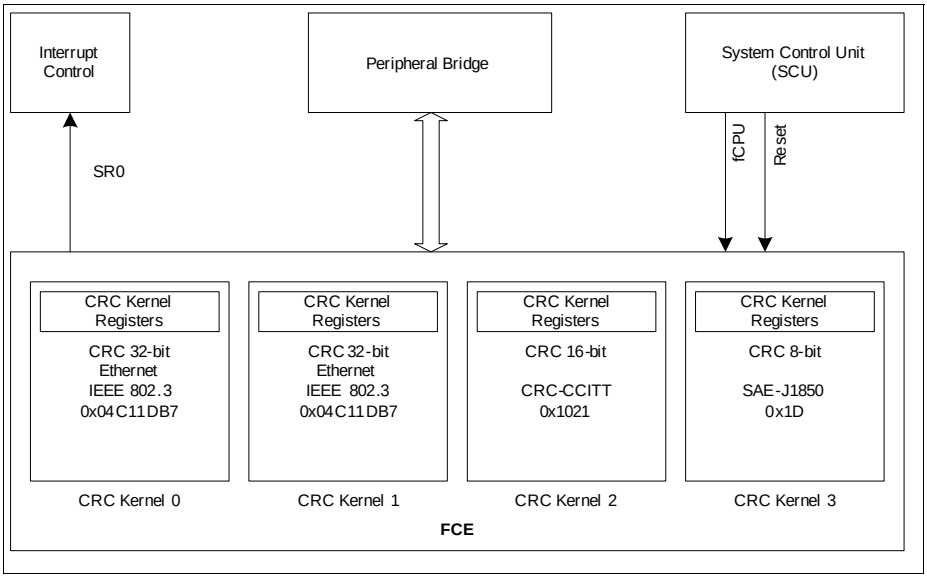


Figure 6-1 FCE **Block Diagram**

Every CRC kernel will present the same hardware and software architecture. The rest of this document will focus only on the description of the generic CRC kernel architecture.

In a multi-kernel implementation the interrupt lines are ored together, the FCE only presents a single interrupt node to the system. Each CRC kernel implements a status register that enables the software to identify which interrupt source is active. Please refer to the **STSm (m = 0-3)** register for a detailed description of the status and interrupt handling.

6.2 Functional Description

.

A checksum algorithm based on CRC polynomial division is characterized by the following properties:

1. polynomial degree (e.g. 32, that represents the highest power of two of the polynomial)
2. polynomial (e.g. 0x04C11DB7: the 33rd bit is omitted because always equal to 1)
3. init value: the initial value of the CRC register

4. input data reflected: indicates if each byte of the input parallel data is reflected before being used to compute the CRC
5. result data reflected: indicates if the final CRC value is reflected or not
6. XOR value: indicates if a final XOR operation is done before returning the CRC result

All the properties are fixed once a polynomial has been chosen. However the FCE provides the capability to control the two reflection steps and the final XOR through the CFG register. The reset values are compatible with the implemented algorithm. The final XOR control enables to select either 0xFFFFFFFF or 0x00000000 to be XORed with the POST_CRC1 value. These two values are those used by the most common CRC polynomials.

Note: The reflection steps and final XOR do not modify the properties of the CRC algorithm in terms of error detection, only the CRC final signature is affected.

The next two figures provides an overview of the control and status features of a CRC kernel.

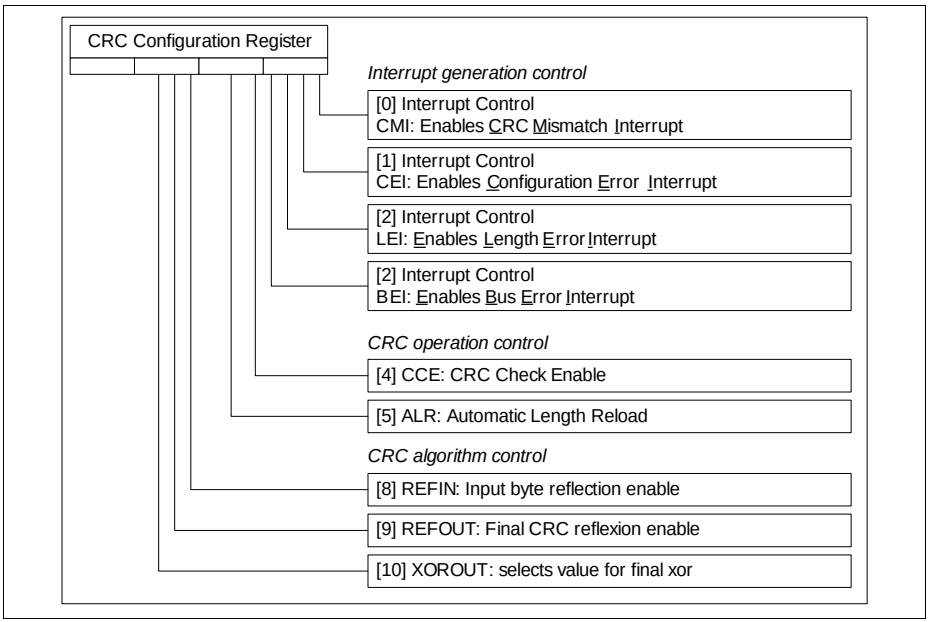


Figure 6-2 CRC kernel configuration register

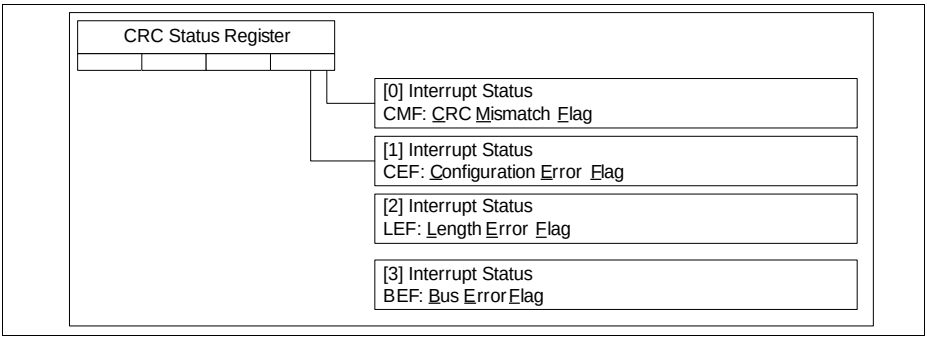


Figure 6-3 CRC kernel status register

6.2.1 Basic Operation

The software must first ensure that the CRC kernel is properly configured, especially the initial CRC value written via the **CRC** register. Then, it writes as many times as necessary into the **IR** register according to the length of the message. The resulting signature is stored in the CRC engine result register, **RESm**, which can be read by the software.

Depending on the CRC kernel accesses by software the following rules apply:

- When accessing a CRC kernel of degree <N> only the bits N-1 down to 0 are used by the CRC kernel. The upper bits are ignored on write. When reading from a CRC kernel register the non-used upper bits are set to 0.

6.2.2 Automatic Signature Check

The automatic signature check compares the signature at the end of a message with the expected signature configured in the CHECK register. In case of a mismatch, an event is generated (see **Section 6.3**. This feature is enabled by the CFG.CCE bit field (see **CFGm (m = 0-3)** register).

If the software whishes to use this feature, the LENGTH register and CHECK registers must be configured with respectively the length as number of words of the message and the expected signature (CHECK). The word length is defined by the degree of the polynomial used. The CHECK value takes into account the final CRC reflection and XOR operation.

When the **CFG.CCE** bit field is set, every time the IR register is written, the LENGH register is decremented by one until it reaches zero. The hardware monitors the transition of the LENGTH register from 1 to 0 to detect the end of the message and proceed with the comparison of the result register RESvalue with the CHECK register value. If the automatic length reload feature is enabled by the CFG.ALR bit field (see

CFGm (m = 0-3)), the LENGTH register is reinitialized with the previously configured value. This feature is especially suited when the FCE is used in combination with a DMA engine.

In the case the automatic length reload feature is not enabled, if LENGTH is already at zero but software still writes to IR (by mistake) every bit of the LENGTH should be set to 1 and hold this value until software initializes it again for the processing of a new message. In such case the STS.LEF (Length Error Flag) should be set and an interrupt generated if the CFG.LEI (Length Error Interrupt) is set.

Usually, the CRC signature of a message M0 is computed and appended to M0 to form the message M1 which is transmitted. One interesting property of CRCs is that the CRC signature of M1 shall be zero. This property is particularly useful when automatically checking the signature of data blocks of fixed length with the automatic length reload enabled. LENGTH should be loaded with the length of M1 and CHECK with 0.

6.2.3 Register protection and monitoring methods

Register Monitoring: applied to CFG and CHECK registers

Because CFG and CHECK registers are critical to the CRC operation, some mechanisms to detect and log transient errors are provided. Early detection of transient failures enables to improve the failure detection time and assess the severity of the failure. The monitoring mechanisms are implemented using two redundant instances as presented in **Figure 6-4**.

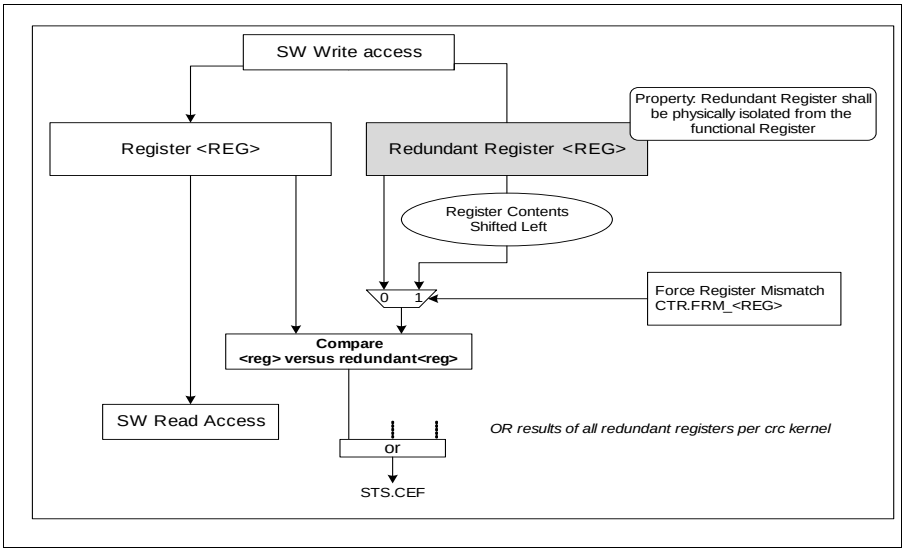


Figure 6-4 Register monitoring scheme

Let <REG> designate either CFG or CHECK registers. When a write to <REG> takes place the redundant register is also updated. **Redundant registers are not visible to software**. Bits of <REG> reserved have no storage and are not used for redundancy. A compare logic continuously compares the two stored values and provides a signal that indicates if the compare is successful or not. The result of all compare blocks are ored together to provide a single flag information. If a mismatch is detected the **STS.CEF** (Configuration Error Flag) bit is set. For run-time validation of the compare logic a Force Register Mismatch bit field (**CTR.FRM_<REG>**) is provided. When set to 1 by software the contents of the redundant register is shifted left by one bit position (redundant bit 0 position is always replaced by a logical 0 value) and is given to the compare logic instead of the redundant register value. This enables to check the compare logic is functional. Using a walking bit pattern, the software can completely check the full operation of the compare logic. Software needs to clear the **CTR.FRM_<REG>** bit to ‘0’ to be able to trigger again a new comparison error interrupt.

Register Access Protection: applies to LENGTH and CHECK registers

In order to reduce the probability of a mis-configuration of the CHECK and LENGTH registers (in the case the automatic check is used), the write access to the CHECK and LENGTH registers must follow a procedure:

Let **<REG>** designate **CHECK** or **LENGTH** registers. Before being able to configure a new <value> value into the <REG> register of a CRC kernel, software must first write the

0xFACECAFE value to the <REG> address. The 0xFACECAFE is not written into the <REG> register. The next write access will proceed as a normal bus write access. The write accesses shall use full 32-bit access only. This procedure will then be repeated every time software wants to configure a new <REG> value. If software reads the CHECK register just after writing 0xFACECAFE it returns the current <REG> contents and not 0xFACECAFE. **A read access to <REG> has no effect on the protection mechanism**.

The following C-code shows write accesses to the CHECK and LENGTH registers following this procedure:

```
//set CHECK register
FCE_CHECK0.U = 0xFACECAFE;
FCE_CHECK0.U = 0;

//set LENGTH register
FCE_LENGTH0.U = 0xFACECAFE;
FCE_LENGTH0.U = 256;
```

6.3 Service Request Generation

Each FCE CRC kernel provides one internal interrupt source. The interrupt lines from each CRC kernel are ored together to be sent to the interrupt system. The system interrupt is an active high pulse with the duration of one cycle (of the peripheral clock). The FCE interrupt handler can use the status information located within the **STS** status register of each CRC kernel.

Each CRC kernel provides the following interrupt sources:

- CRC Mismatch Interrupt controlled by **CFG.CMI** bit field and observable via the status bit field **STS.CMF** (CRC Mismatch Flag).
- Configuration Error Interrupt controlled by **CFG.CEI** bit field and observable via the status bit field **STS.CEF** (Configuration Error Flag).
- Length Error Interrupt controlled by **CFG.LEI** bit field and observable via the status bit field **STS.LEF** (Length Error Flag).
- Bus Error Interrupt controlled by **CFG.BEI** bit field and observable via the status bit field **STS.BEF** (Bus Error Flag).

Interrupt generation rules

- A status flag shall be cleared by software by writing a **1** to the corresponding bit position.
- If an status flag is set and a new hardware condition occurs, no new interrupt is generated by the kernel: the STS.<FLAG> bit field masks the generation of a new

interrupt from the same source. If a SW access to clear the interrupt status bit takes place and in the same cycle the hardware wants to set the bit, the hardware condition wins the arbitration.

As all the interrupts are caused by an error condition, the interrupt shall be handled by a Error Management software layer. The software services using the FCE as acceleration engine may not directly deal with error conditions but let the upper layer using the service to deal with the error handling.

6.4 Debug Behavior

The FCE has no specific debug feature.

6.5 Power, Reset and Clock

The FCE is inside the power core domain, therefore no special considerations about power up or power down sequences need to be taken. For an explanation about the different power domains, please address the SCU (System Control Unit) section.

A power down mode can be achieved by disabling the module using the **Clock Control Register (CLC)**.

The FCE module has one reset source. This reset source is handled at system level and it can be generated independently via a system control register (address SCU section for full description).

After release, the complete IP is set to default configuration. The default configuration for each register field is addressed on **Section 6.7**.

The FCE uses the CPU clock, fCPU (address SCU section for more details on clocking).

6.6 Initialization and System Dependencies

The FCE may have dependencies regarding the bus clock frequency. This dependencies should be addressed in the SCU and System Architecture sections.

Initialization:

The FCE is enabled by writing 0x0 to the CLC register. Software must first ensure that the CRC kernel is properly configured, especially the initial CRC register value written via the CRC register, the input and result reflection as well as the final xored value via the CFG register. The following source code is an example of initialization for the basic operation of the FCE kernel 0:

```
//enable FCE
FCE_CLC.U = 0x0;
//final result to be xored with 0xFFFFFFFF, no reflection
```

```
FCE_CFG0.U = 0x400;
//set CRC initial value (seed)
FCE_CRC0.U = 0xFFFFFFFF;
```

6.7 Registers

Table 6-3 show all registers associated with a FCE CRC-kernel. All FCE kernel register names are described in this section. They should get the prefix “FCE_” when used in the context of a product specification.

The registers are numbered by one index to indicate the related FCE CRC Kernel (m = 0-3). Some kernel registers are adapted to the degree of the polynomial implemented by the kernel.

Table 6-2 Registers Address Space - FCE Module

Module	Base Address	End Address	Note
FCE	$5002\ 0000_H$	$5002\ 3FFF_H$	

Table 6-3 Registers Overview - CRC Kernel Registers

Short Name	Description	Offset Addr.[1)]	Access Mode		Reset Class	Description See
			Read	Write		
System Registers						
CLC	Clock Control Register	00_H	U, PV	PV	3	Page 6-12
ID	Module Identification Register	08_H	U, PV	BE	3	Page 6-12
Generic CRC Engine Registers						
IRm	Input Register m	$20_H + m*20_H$	U, PV	U, PV	3	Page 6-13
RESm	CRC Result Register m	$24_H + m*20_H$	U, PV	BE	3	Page 6-15
CFGm	CRC Configuration Register m	$28_H + m*20_H$	U, PV	PV	3	Page 6-17
STSm	CRC Status Register m	$2C_H + m*20_H$	U, PV	U, PV	3	Page 6-19
LENGTHm	CRC Length Register m	$30_H + m*20_H$	U, PV	U, PV	3	Page 6-20
CHECKm	CRC Check Register m	$34_H + m*20_H$	U, PV	U, PV	3	Page 6-20
CRCm	CRC Register m	$38_H + m*20_H$	U, PV	U, PV	3	Page 6-22
CTRm	CRC Test Register m	$3C_H + m*20_H$	U, PV	U, PV	3	Page 6-23

1) The absolute register byte address for each CRC kernel m is calculated as follows:
CRC kernel register base Address (**Table 6-2**) + m*20H, m = 0-3

Disabling the FCE

The FCE module can be disabled using the **CLC** register.

When the disable state is requested all pending transactions running on the bus slave interface must be completed before the disabled state is entered. The CLC Register Module Disable Bit Status CLC.DISS indicates whether the module is currently disabled (DISS == 1). Any attempt to write any register with the exception of the CLC Register will generate a bus error. A read operation is allowed and does not generate a bus error.

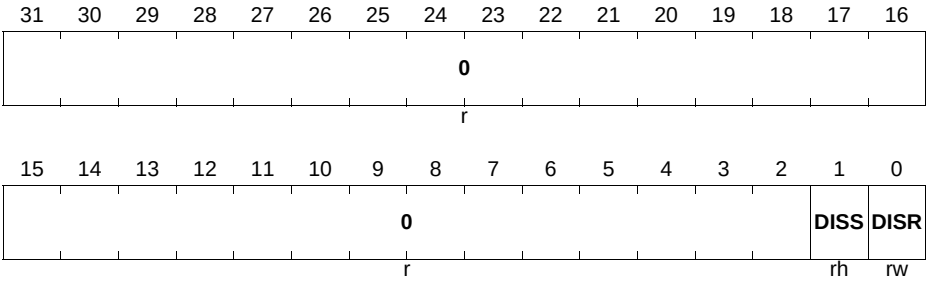
6.7.1 System Registers description

This section describes the registers related to the product system architecture.

Clock Control Register (CLC)

The Clock Control Register allows the programmer to adapt the functionality and power consumption of the module to the requirements of the application.

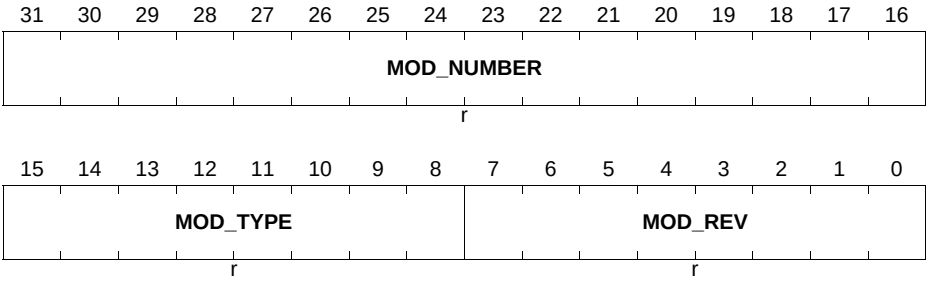
CLC
Clock Control Register **(00_H)** **Reset Value: $0000\ 0003_H$**



Field	Bits	Type	Description
DISR	0	rw	**Module Disable Request Bit** Used for enable/disable control of the module.
DISS	1	rh	**Module Disable Status Bit** Bit indicates the current status of the module.
0	[31:2]	r	**Reserved** Read as 0; should be written with 0.

Module Identification Register

ID
Module Identification Register **(08_H)** **Reset Value: $00CA\ C001_H$**

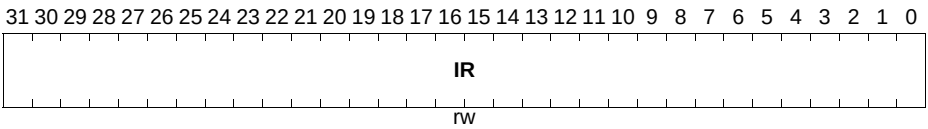


Field	Bits	Type	Description
MOD_REV	[7:0]	r	**Module Revision Number** This bit field defines the module revision number. The value of a module revision starts with 01_H (first revision). The current revision number is 01_H
MOD_TYPE	[15:8]	r	**Module Type** The bit field is set to $C0_H$ which defines the module as a 32-bit module.
MOD_NUMBER	[31:16]	r	**Module Number Value** This bit field defines a module identification number. The value for the FCE module is $00CA_H$.

6.7.2 CRC Kernel Control/Status Registers

CRC Engine Input Register

IRm (m = 0-1)
Input Register m **($20_H + m*20_H$)** **Reset Value: $0000\ 0000_H$**

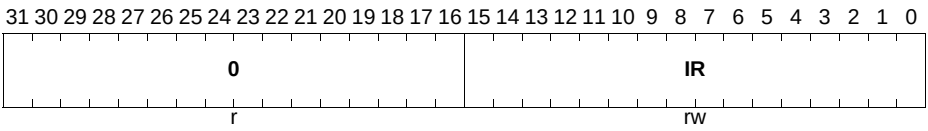


Field	Bits	Type	Description
IR	[31:0]	rw	**Input Register** This bit field holds the 32-bit data to be computed

A write to IRm triggers the CRC kernel to update the message checksum according to the IR contents and to the current CRC register contents. Only 32-bit write transactions are allowed to this IRm registers, any other bus write transaction will lead to a Bus Error.

CRC Engine Input Register

IRm (m = 2-2)
Input Register m **(20_H + m*20_H)** **Reset Value: 0000 0000_H**

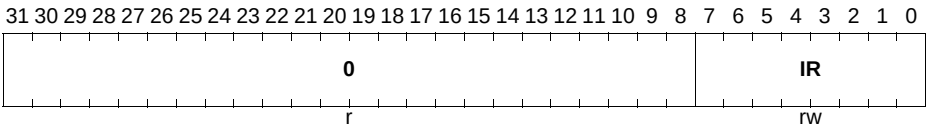


Field	Bits	Type	Description
IR	[15:0]	rw	**Input Register** This bit field holds the 16-bit data to be computed
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

A write to IRm triggers the CRC kernel to update the message checksum according to the IR contents and to the current CRC register contents. Only 32-bit or 16-bit write transactions are allowed to this IRm register, any other bus write transaction will lead to a Bus Error. Only the lower 16-bit of the write transactions will be used.

CRC Engine Input Register

IRm (m = 3-3)
Input Register m **(20_H + m*20_H)** **Reset Value: 0000 0000_H**

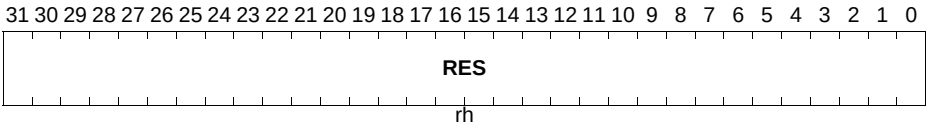


Field	Bits	Type	Description
IR	[7:0]	rw	**Input Register** This bit field holds the 8-bit data to be computed
0	[31:8]	r	**Reserved** Read as 0; should be written with 0.

A write to IRm triggers the CRC kernel to update the message checksum according to the IR contents and to the current CRC register contents. Any write transaction is allowed to this IRm register. Only the lower 8-bit of the write transactions will be used.

CRC Engine Result Register

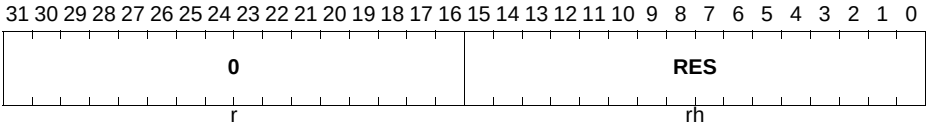
RESm (m = 0-1)
CRC Result Register m **(24_H + m*20_H)** **Reset Value: FFFF FFFF$_H$**



Field	Bits	Type	Description
RES	[31:0]	rh	**Result Register** Returns the final CRC value including CRC reflection and final XOR according to the CFG register configuration. Writing to this register has no effect.

CRC Engine Result Register

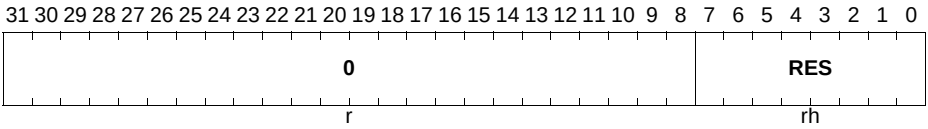
RESm (m = 2-2)
CRC Result Register m **(24_H + m*20_H)** **Reset Value: 0000 FFFF$_H$**



Field	Bits	Type	Description
RES	[15:0]	rh	**Result Register** Returns the final CRC value including CRC reflection and final XOR according to the CFG register configuration. Writing to this register has no effect.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

CRC Engine Result Register

RESm (m = 3-3)
CRC Result Register m **(24_H + m*20_H)** **Reset Value: 0000 00FF_H**



Field	Bits	Type	Description
RES	[7:0]	rh	**Result Register** Returns the final CRC value including CRC reflection and final XOR according to the CFG register configuration. Writing to this register has no effect.
0	[31:8]	r	**Reserved** Read as 0; should be written with 0.

CRC Engine Configuration Register

CFGm (m = 0-3)
CRC Configuration Register m

(28_H + m*20_H) **Reset Value: 0000 0700_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

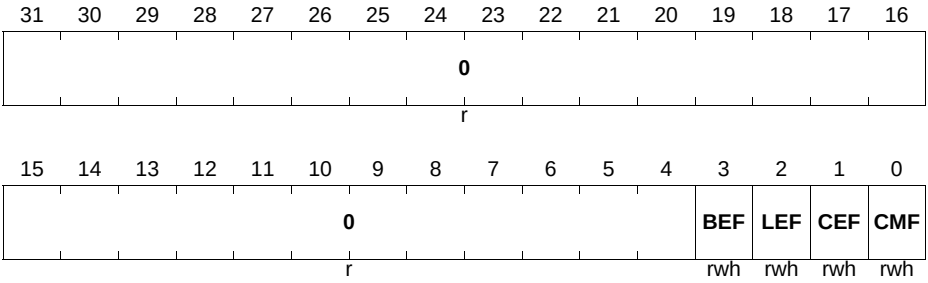
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0					XSEL	REFOUT	REFIN	0		ALR	CCE	BEI	LEI	CEI	CMI
r					rw	rw	rw	r		rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
CMI	0	rw	**CRC Mismatch Interrupt** 0_B CRC Mismatch Interrupt is disabled 1_B CRC Mismatch Interrupt is enabled
CEI	1	rw	**Configuration Error Interrupt** When enabled, a Configuration Error Interrupt is generated whenever a mismatch is detected in the CFG and CHECK redundant registers. 0_B Configuration Error Interrupt is disabled 1_B Configuration Error Interrupt is enabled
LEI	2	rw	**Length Error Interrupt** When enabled, a Length Error Interrupt is generated if software writes to IR register with LENGTH equal to 0 and CFG.CCE is set to 1. 0_B Length Error Interrupt is disabled 1_B Length Error Interrupt is enabled
BEI	3	rw	**Bus Error Interrupt** When enabled, an interrupt is generated if a bus write transaction with an access width smaller than the kernel width is issued to the input register. 0_B Bus Error Interrupt is disabled 1_B Bus Error Interrupt is enabled

Field	Bits	Type	Description
CCE	4	rw	**CRC Check Comparison** 0_B CRC check comparison at the end of a message is disabled 1_B CRC check comparison at the end of a message is enabled
ALR	5	rw	**Automatic Length Reload** 0_B Disables automatic reload of the LENGTH field. 1_B Enables automatic reload of the LENGTH field at the end of a message.
REFIN	8	rw	**IR Byte Wise Reflection** 0_B IR Byte Wise Reflection is disabled 1_B IR Byte Wise Reflection is enabled
REFOUT	9	rw	**CRC 32-Bit Wise Reflection** 0_B CRC 32-bit wise is disabled 1_B CRC 32-bit wise is enabled
XSEL	10	rw	**Selects the value to be xored with the final CRC** 0_B 0x00000000 1_B 0xFFFFFFFF
0	[7:6], [31:11]	r	**Reserved** Read as 0; should be written with 0.

CRC Engine Status Register

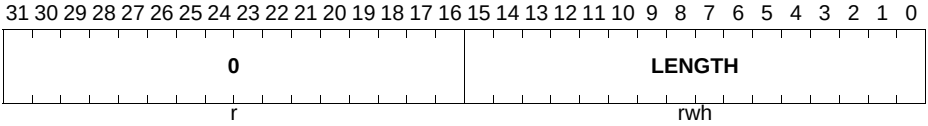
STSm (m = 0-3)
CRC Status Register m **($2C_H$ + m*20_H)** **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
CMF	0	rwh	**CRC Mismatch Flag** This bit is set per hardware only. To clear this bit, software must write a 1 to this bit field location. Writing 0 per software has no effect.
CEF	1	rwh	**Configuration Error Flag** This bit is set per hardware only. To clear this bit, software must write a 1 to this bit field location. Writing 0 per software has no effect.
LEF	2	rwh	**Length Error Flag** This bit is set per hardware only. To clear this bit, software must write a 1 to this bit field location. Writing 0 per software has no effect.
BEF	3	rwh	**Bus Error Flag** This bit is set per hardware only. To clear this bit, software must write a 1 to this bit field location. Writing 0 per software has no effect.
0	[31:4]	r	**Reserved** Read as 0; should be written with 0.

CRC Engine Length Register

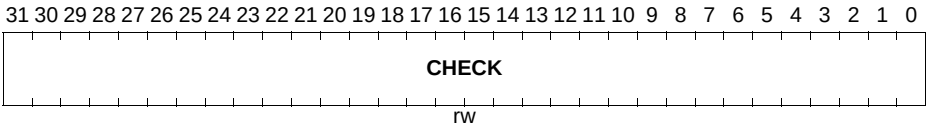
LENGTHm (m = 0-3)
CRC Length Register m **(30_H + m*20_H)** **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
LENGTH	[15:0]	rwh	**Message Length Register** Number of words building the message over which the CRC checksum is calculated. This bit field is modified by the hardware: every write to the IR register decrements the value of the LENGTH bit field. If the CFG.ALR field is set to 1, the LENGTH field shall be reloaded with its configuration value at the end of the cycle where LENGTH reaches 0.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

CRC Engine Check Register

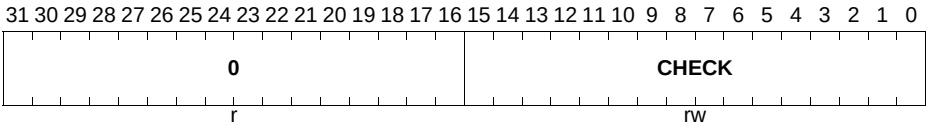
CHECKm (m = 0-1)
CRC Check Register m **(34_H + m*20_H)** **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
CHECK	[31:0]	rw	**CHECK Register** Expected CRC value to be checked by the hardware upon detection of a 1 to 0 transition of the LENGTH register. The comparison is enabled by the CFG.CCE bit field

CRC Engine Check Register

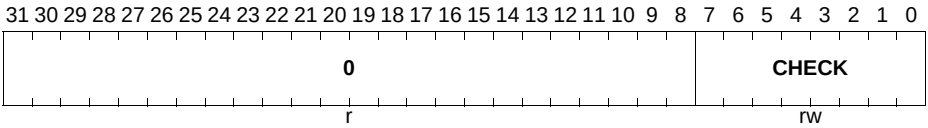
CHECKm (m = 2-2)
CRC Check Register m **(34_H + m*20_H)** **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
CHECK	[15:0]	rw	**CHECK Register** Expected CRC value to be checked by the hardware upon detection of a 1 to 0 transition of the LENGTH register. The comparison is enabled by the CFG.CCE bit field
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

CRC Engine Check Register

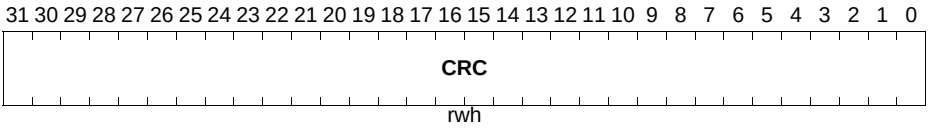
CHECKm (m = 3-3)
CRC Check Register m **(34_H + m*20_H)** **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
CHECK	[7:0]	rw	**CHECK Register** Expected CRC value to be checked by the hardware upon detection of a 1 to 0 transition of the LENGTH register. The comparison is enabled by the CFG.CCE bit field
0	[31:8]	r	**Reserved** Read as 0; should be written with 0.

CRC Engine Initialization Register

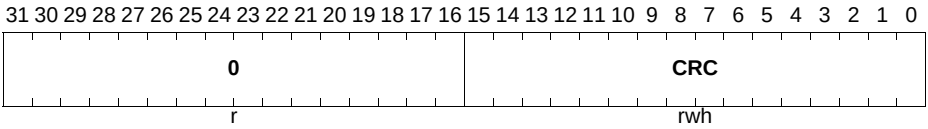
CRCm (m = 0-1)
CRC Register m **(38_H + m*20_H)** **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
CRC	[31:0]	rwh	**CRC Register** This register enables to directly access the internal CRC register

CRC Engine Initialization Register

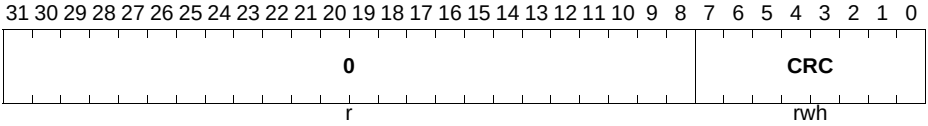
CRCm (m = 2-2)
CRC Register m **(38_H + m*20_H)** **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
CRC	[15:0]	rwh	**CRC Register** This register enables to directly access the internal CRC register
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

CRC Engine Initialization Register

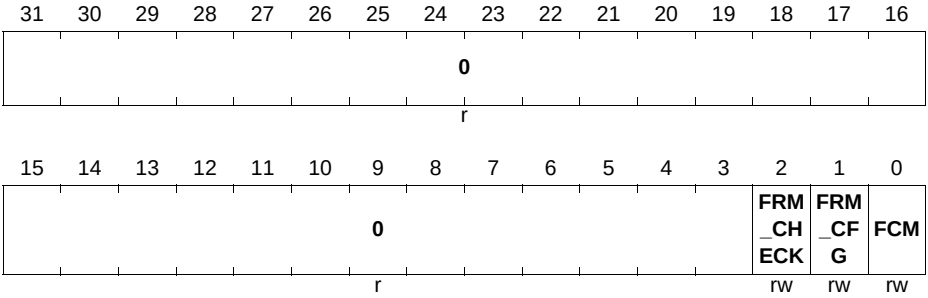
CRCm (m = 3-3)
CRC Register m **(38_H + m*20_H)** **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
CRC	[7:0]	rwh	**CRC Register** This register enables to directly access the internal CRC register
0	[31:8]	r	**Reserved** Read as 0; should be written with 0.

CRC Test Register

CTRm (m = 0-3)
CRC Test Register m **($3C_H$ + m*20_H)** **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
FCM	0	rw	**Force CRC Mismatch** Forces the CRC compare logic to issue an error regardless of the CHECK and CRC values. The hardware detects a 0 to 1 transition of this bit field and triggers a CRC Mismatch interrupt

Field	Bits	Type	Description
FRM_CFG	1	rw	**Force CFG Register Mismatch** This field is used to control the error injection mechanism used to check the compare logic of the redundant CFG registers. This is a one shot operation. When the hardware detects a 0 to 1 transition of this bit field it triggers a Configuration Mismatch interrupt (if enabled by the corresponding CFGm register).
FRM_CHECK	2	rw	**Force Check Register Mismatch** This field is used to control the error injection mechanism used to check the compare logic of the redundant CHECK registers. This is a one shot operation. The hardware detects a 0 to 1 transition of this bit field and triggers a Check Register Mismatch interrupt (if enabled by the corresponding CFGm register).
0	[31:3]	r	**Reserved** Read as 0; should be written with 0.

6.8 Interconnects

The interfaces of the FCE module shall be described in the module design specification.

The **Table 6-4** shows the services requests of the FCE module.

Table 6-4 FCE Service Requests

Inputs/Outputs	I/O	Connected To	Description
FCE.SR0	O	NVIC	Service request line

6.9 Properties of CRC code

Hamming Distance

The Hamming distance defines the error detection capability of a CRC polynomial. A cyclic code with a Hamming Distance of D can detect all D-1 bit errors. **Table 6-5 “Hamming Distance as a function of message length (bits)” on Page 6-25** shows the dependency of the Hamming Distance with the length of the message.

Table 6-5 Hamming Distance as a function of message length (bits)[1)]

Hamming Distance	IEEE-802.3 CRC32	CCITT CRC16	J1850 CRC8
15	8 - 10	Information not available	Information not available
14	8 - 10		
13	8 - 10		
12	11 - 12		
11	13 - 21		
10	22 - 34		
9	35 - 57		
8	58 - 91		
7	92 - 171		
6	172 - 268		
5	269 - 2974		
4	2973 - 91607		
3	91607 - 131072		

1) Data from technical paper “32-Bit Cyclic Redundancy Codes for Internet Applications” by Philip Koopman, Carnegie Mellon University, 2002

On-Chip Memories

7 Memory Organization

This chapter provides description of the system Memory Organization and basic information related to Parity Testing and Parity Error handling.

References

[8] Cortex™-M4 User Guide, ARM DUI 0508B (ID062910)

7.1 Overview

The Memory Map is intended to balance decoding cost at various level of the system bus infrastructure.

7.1.1 Features

The Memory Map implements the following features:

- Compatibility with standard ARM Cortex-M4 CPU **[8]**
- Compatibility across entire XMC4000 Family
- Optimal functional module address spaces grouping

7.1.2 Cortex-M4 Address Space

The system memory map defines several regions. Address boundaries of each of the regions are determined by the Cortex-M4 core architecture.

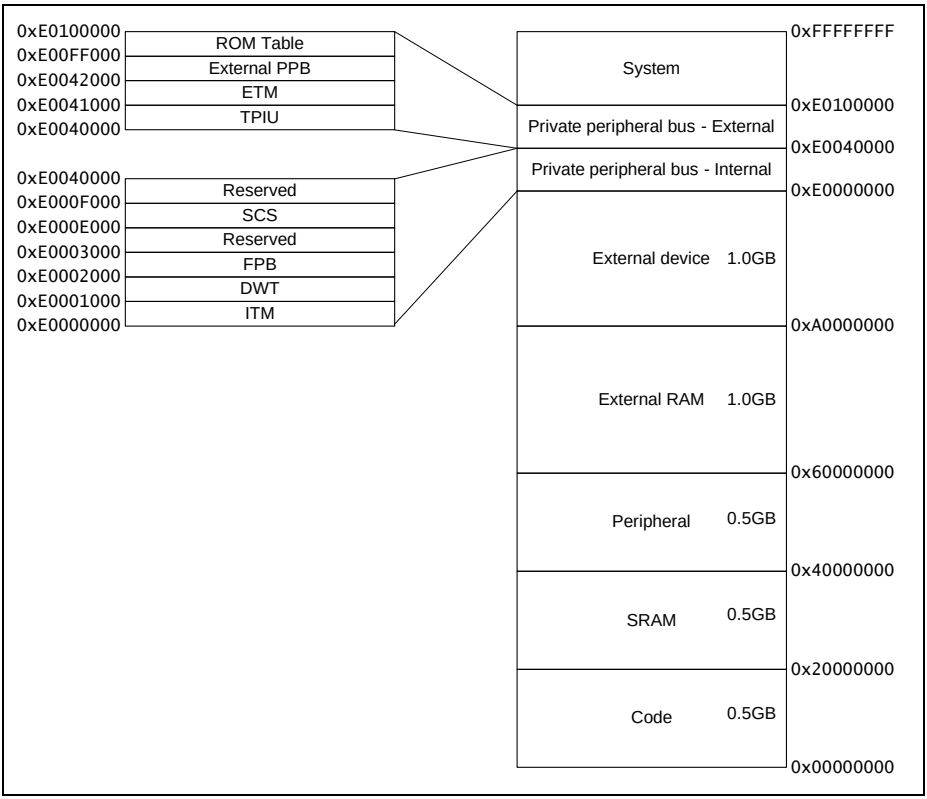


Figure 7-1 Cortex-M4 processor address space

7.2 Memory Regions

The XMC4[12]00 device specific address map assumes presence of internal and external memories and peripherals. The memory regions for XMC4[12]00 are described in **Table 7-1**.

Table 7-1 Memory Regions

Start	End	Size (hex)	Space name	Usage
00000000	1FFFFFFF	20000000	Code	Boot ROM Flash Program SRAM
20000000	3FFFFFFF	20000000	SRAM	Fast internal SRAMs
40000000	47FFFFFF	08000000	Peripheral 0	Internal Peripherals group 0
48000000	4FFFFFFF	08000000	Peripheral 1	Internal Peripherals group 1
50000000	57FFFFFF	08000000	Peripheral 2	Internal Peripherals group 2
58000000	5FFFFFFF	08000000	Peripheral 3	Internal Peripherals group 3
60000000	9FFFFFFF	40000000	External SRAM	External Memories
A0000000	DFFFFFFF	40000000	External Device	External Devices
E0000000	E00FFFFF	00100000	Private Peripheral Bus	CPU
E0100000	EFFFFFFF	0FF00000	Vedor specific 1	reserved
F0000000	FFFFFFFF	10000000	Vedor specific 2	reserved

7.3 Memory Map

Table 7-2 defines detailed system memory map of XMC4[12]00 where each individual peripheral or memory instance implement its own address spaces. For detailed register description of the system components and peripherals please refer to respective chapters of this document.

Table 7-2 Memory Map

Addr space	Start Address (hex)	End Address (hex)	Modules
Code	00000000	00003FFF	BROM (PMU ROM)
	00004000	07FFFFFF	reserved
	08000000	0803FFFF	PMU/FLASH (cached)
	08040000	09E1FFFF	reserved
	09E20000	09E23FFF	reserved
	09E24000	0BFFFFFF	reserved
	0C000000	0C03FFFF	PMU/FLASH (uncached)
	0C040000	0FFFFFFF	reserved
	0DE20000	0DE23FFF	reserved
	0DE24000	0FFFFFFF	reserved
	10000000	1FFFBFFF	reserved
	1FFFC000	1FFFFFFF	PSRAM (code)
SRAM	20000000	20005FFF	DSRAM1 (system)
	20006000	2FFFFFFF	reserved
	30000000	30007FFF	reserved
	30008000	3FFFFFFF	reserved

Table 7-2 Memory Map (cont'd)

Addr space	Start Address (hex)	End Address (hex)	Modules
Peripherals 0	40000000	40003FFF	PBA0
	40004000	40007FFF	VADC
	40008000	4000BFFF	reserved
	4000C000	4000FFFF	CCU40
	40010000	40013FFF	CCU41
	40014000	40017FFF	reserved
	40018000	4001BFFF	reserved
	4001C000	4001FFFF	reserved
	40020000	40023FFF	CCU80
	40024000	40027FFF	reserved
	40028000	4002BFFF	POSIF0
	4002C000	4002FFFF	reserved
	40030000	40033FFF	USIC0
	40034000	40037FFF	reserved
	40038000	4003BFFF	reserved
	4003C000	4003FFFF	reserved
	40044000	40047FFF	ERU1
	40048000	47FFFFFF	reserved
Peripherals 1	48000000	48003FFF	PBA1
	48004000	48007FFF	reserved
	48008000	4800BFFF	reserved
	4800C000	4800FFFF	reserved
	48010000	48013FFF	LEDTS0
	48014000	48017FFF	CAN
	48018000	4801BFFF	DAC
	4801C000	4801FFFF	reserved
	48020000	48023FFF	USIC1
	48024000	48027FFF	reserved
	48028000	4802BFFF	PORTS
	4802C000	4FFFFFFF	reserved

Table 7-2 Memory Map (cont'd)

Addr space	Start Address (hex)	End Address (hex)	Modules
Peripherals 2	50000000	50003FFF	PBA2
	50004000	50007FFF	SCU & RTC
	50008000	5000BFFF	WDT
	5000C000	5000FFFF	reserved
	50010000	50013FFF	reserved
	50014000	50017FFF	DMA0
	50018000	5001BFFF	reserved
	5001C000	5001FFFF	reserved
	50020000	50023FFF	FCE
	50024000	5003FFFF	reserved
	50040000	5007FFFF	USB
	50080000	57FFFFFF	reserved
Peripherals 3	58000000	58003FFF	PMU0 registers
	58004000	58007FFF	PMU0 prefetch
	58008000	5800BFFF	reserved
	5800C000	5800FFFF	reserved
	58010000	58013FFF	reserved
	58014000	58017FFF	reserved
	58018000	5FFFFFFF	reserved
External SRAM	60000000	63FFFFFF	reserved
	64000000	67FFFFFF	reserved
	68000000	6BFFFFFF	reserved
	6C000000	6FFFFFFF	reserved
	70000000	9FFFFFFF	reserved
External Device	A0000000	A3FFFFFF	reserved
	A4000000	A7FFFFFF	reserved
	A8000000	ABFFFFFF	reserved
	AC000000	AFFFFFFF	reserved
	B0000000	DFFFFFFF	reserved

Table 7-2 Memory Map (cont'd)

Addr space	Start Address (hex)	End Address (hex)	Modules
Private Peripheral Bus	E0000000	E0000FFF	ITM
	E0001000	E0001FFF	DWT
	E0002000	E0002FFF	FPB
	E0003000	E000DFFF	reserved
	E000E000	E000EFFF	SCS
	E000E010	E000E01C	SysTick
	E000EF34	E000EF47	FPU
	E000F000	E003FFFF	reserved
	E0040000	E0040FFF	TPIU
	E0041000	E0041FFF	ETM
	E0042000	E00FEFFF	reserved
	E00FF000	E00FFFFF	ROM Table
Vedor specific 1	E0100000	EFFFFFFF	reserved
Vedor specific 2	F0000000	FFFFFFFF	reserved

7.4 Service Request Generation

Memory modules and other system components are capable of generating error responses indicated to the CPU as bus error exceptions or interrupts.

Types of error causes

- Unsupported Access Mode
- Access to Invalid Address
- Parity Error (memories only)
- Bufferable Write Access to Peripheral

Errors that cannot be indicated with bus errors get indicated with service requests that get propagated to the CPU as interrupts. Typically lack of bus error response capability applies to memory modules that lack of direct access from the system bus This applies to memories that serve the purpose of internal FIFOs and local storage buffers.

Unsupported Access Modes

Unsupported access modes can be classified in various ways and are usually specific to the module that access is performed to. The typical examples of unsupported access modes are read access to write-only or write access to read-only type of address

mapped resources, unsupported access data widths, protected memory regions. For module specific limitations please refer to individual module chapters.

Invalid Address

Accesses to invalid addresses result in error responses. Invalid addresses are defined as those that do not mapped to any valid resources. This applies to single addresses and to wider address ranges. Some invalid addresses within valid module address ranges may not produce error responses and this is specific to individual modules.

Parity Errors

Parity test is performed on the XMC4[12]00 memories in normal functional mode. Parity errors are generated in case of failure of parity test performed inside of each of the memory module. The mechanism of parity testing depends on memory data width and access mode, i.e. memory modules that are accessible byte-wise implement parity check for each data byte individually while for memory modules that are accessible double-word-wise it is sufficient to perform joint check for all bits.

The occurrence of a parity error gets signalized to the system with system bus error or an interrupt (parity trap). For details on parity error generation control and handling please refer to the SCU chapter. For more details please refer to **Table 7-3**.

Table 7-3 Parity Test Enabled Memories and Supported Parity Error Indication

Memory	Number of Parity Bits	Parity Test Granularity	Bus Error	Parity Trap
Program SRAM (PSRAM)	1	4 bytes	yes	yes
System SRAM (DSRAM1)	4	1 byte	yes	yes
USIC 0 Buffer Memory	1	4 bytes	no	yes
USIC 1 Buffer Memory	1	4 bytes	no	yes
MultiCAN Buffer Memory	1	4 bytes	no	yes
PMU Prefetch Buffer Memory	1	4 bytes	no	yes
USB Buffer Memory	1	4 bytes	no	yes

Bufferable Write Access to Peripheral

Bufferable writes to peripheral may result in error responses as described above. Bus error responses from modules attached to peripheral bridges PBA0 and PBA1 trigger service request from the respective bridge that will result in NMI to the CPU. Error status and access address that caused the service request get stored in dedicated registers of the peripheral bridges. For detail please refer to **Registers**.

7.5 Debug Behavior

The bus system in debug mode allows debug probe access to all system resources except for the Flash sectors protected with a dedicated protection mechanism (for more details please refer to Flash Memory chapter). No special handling of HALT mode is implemented and all interfaces respond with a valid bus response upon accesses.

7.6 Power, Reset and Clock

The bus system clocking scheme enables stable system operation and accesses to system resources for all valid system clock rates. Some parts of the system may also run at a half of the system clock rate and no special handling is required as appropriate alignment of the bus system protocol is provided on the clock domain boundary (for details please refer to clocking system description in SCU chapter).

7.7 Initialization and System Dependencies

No initialization is required for the memory system from user point of view. All valid memories are available after reset. Some peripherals may need to be initialized (e.g. released from reset state) before accessed. For details please refer to individual peripheral chapters.

7.8 Registers

This section describes registers of the Peripheral Bridges. The purpose of the registers is handling of errors signalized during bufferable accesses to peripherals connected to the respective bridges. Active errors on bufferable writes trigger interrupt requests generated from the Peripheral Bridges that can be monitored and cleared in the register defiled in this chapter.

Table 7-4 Registers Address Space

Module	Base Address	End Address	Note
PBA0	$4000\ 0000_H$	$4000\ 3FFF_H$	Peripheral Bridge 0
PBA1	$4800\ 0000_H$	$4800\ 3FFF_H$	Peripheral Bridge 1

Table 7-5 Registers Overview

Register Short Name	Register Long Name	Offset Address	Access Mode		Description
			Read	Write	
PBA0_STS	PBA 0 Status Register	0000_H	U, PV	PV	Page 7-10
PBA0_WADDR	PBA 0 Write Error Address	0004_H	U, PV		Page 7-11
PBA1_STS	PBA 1 Status Register	0000_H	U, PV	PV	Page 7-12
PBA1_WADDR	PBA 1 Write Error Address	0004_H	U, PV		Page 7-12

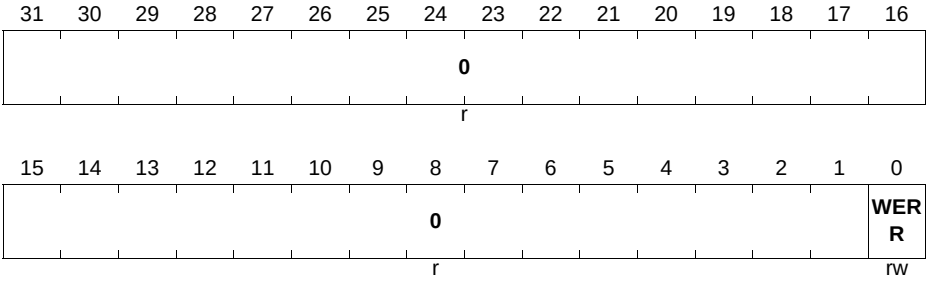
PBA0_STS

The status register of PBA0 bridge indicates bus error occurrence for write access. Is meant to be used for errors triggered upon buffered writes. The bit gets set and interrupt request has been generated to the SCU.

Write one to clear, writing zero has no effect.

PBA0_STS

Peripheral Bridge Status Register (0000$_H$) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
WERR	0	rw	**Bufferable Write Access Error** 0_B no write error occurred. 1_B write error occurred, interrupt request is pending.
0	[31:1]	r	**Reserved bits. Write zeros**

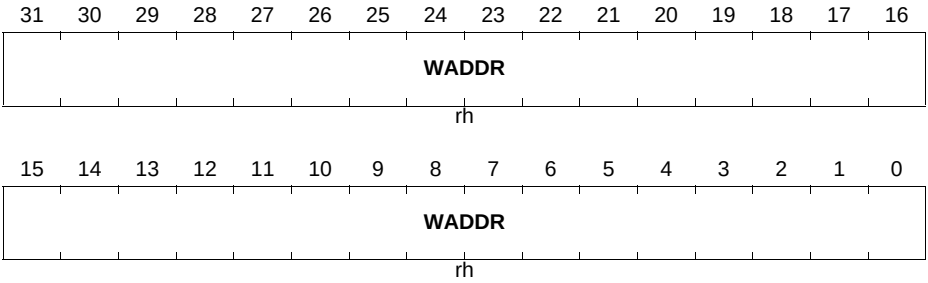
PBA0_WADDR

The Write Error Address Register keeps write access address that caused a bus error upon bufferable write attempt to a peripheral connected to PBA0 bridge. This register store the address that of the bufferable write access attempt that caused error resulting in setting WERR bit of the **PBA0_STS** register.

This register value remains unchanged when WERR bit of **PBA0_STS** register is set.

PBA0_WADDR

PBA Write Error Address Register (0004$_H$) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
WADDR	[31:0]	rh	**Write Error Address** Address of the write access that caused a bus error on the bridge Master port.

PBA1_STS

The status register of PBA1 bridge indicates bus error occurrence for write access. Is meant to be used for errors triggered upon buffered writes. The bit gets set and interrupt request has been generated to the SCU.

Write one to clear, writing zero has no effect.

PBA1_STS
Peripheral Bridge Status Register (0000_H) **Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0															**WERR**
r															rw

Field	Bits	Type	Description
WERR	0	rw	**Bufferable Write Access Error** 0_B no write error occurred. 1_B write error occurred, interrupt request is pending.
0	[31:1]	r	**Reserved bits. Write zeros**

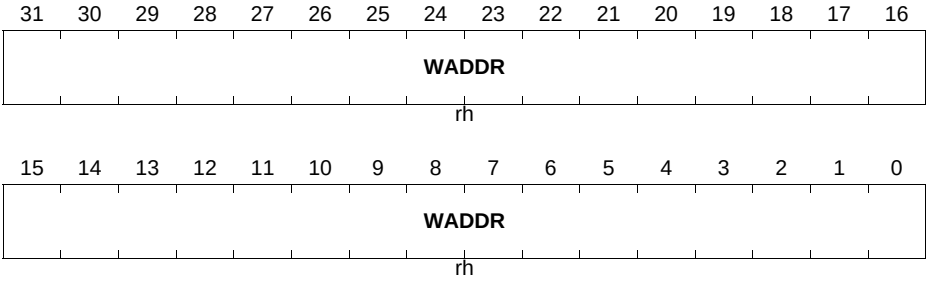
PBA1_WADDR

The Write Error Address Register keeps write access address that caused a bus error upon bufferable write attempt to a peripheral connected to PBA1 bridge. This register store the address that of the bufferable write access attempt that caused error resulting in setting WERR bit of the **PBA1_STS** register.

This register value remains unchanged when WERR bit of **PBA1_STS** register is set.

PBA1_WADDR

PBA Write Error Address Register (0004_H) Reset Value: $0000\ 0000_H$



Field	Bits	Type	Description
WADDR	[31:0]	rh	**Write Error Address** Address of the write access that caused a bus error on the bridge Master port.

8 Flash and Program Memory Unit (PMU)

The Program Memory Unit (PMU) controls the Flash memory and the BROM and connects these to the system. The Prefetch unit maximizes system performance with higher system frequencies, by buffering instruction and data accesses to the Flash.

8.1 Overview

In the XMC4[12]00, the PMU controls the following interfaces:

- The Flash command and fetch control interface for Program Flash
- The Boot ROM interface
- The PMU interfaces via the Prefetch unit to the Bus Matrix

Following memories are controlled by and belong to the PMU:

- 256 Kbyte of Program Flash memory (PFLASH)
- 16 Kbyte of BROM (BROM)
- 1 Kbyte of Instruction Cache memory in the Prefetch unit
- 256-bit Data Buffer in the Prefetch unit

8.1.1 Block Diagram

The PMU block diagram is shown in **Figure 8-1**.

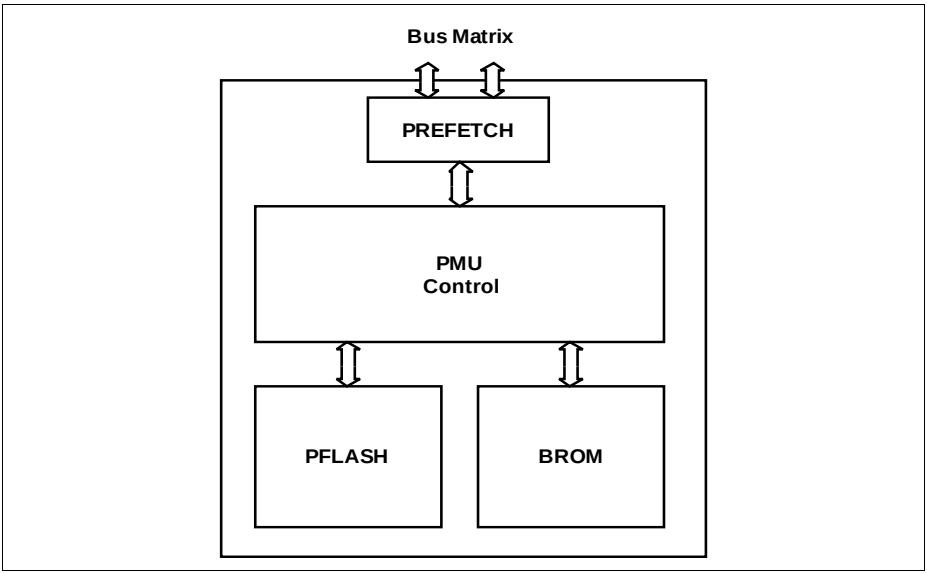


Figure 8-1 PMU Block Diagram

8.2 Boot ROM (BROM)

The Boot ROM in PMU0 has a capacity of 16 KB. The BROM contains the Firmware with:

- startup routines
- bootstrap loading software.

Details on the operations of the BROM are given in the chapter “Startup Modes”.

8.2.1 BROM Addressing

The BROM is visible at one location, as can be seen in the memory map:

- (non-cached space) starting at location $0000\ 0000_H$

After any reset, the hardware-controlled start address is $0000\ 0000_H$. At this location, the startup procedure is stored and started. As no other start location after reset is supported, the startup software within the BROM is always executed first after any reset.

8.3 Prefetch Unit

The purpose of the Prefetch unit is to reduce the Flash latency gap at higher system frequencies to increase the instruction per cycle performance.

8.3.1 Overview

The Prefetch unit separates between instruction and data accesses to the Flash with the following configuration:

- 1 Kbyte Instruction Buffer
  - 2-way set associative
  - Least-Recently-Used (LRU) replacement policy
  - Cache line size: 256 bits
  - Critical word first
  - Streaming[1)]
  - Line wrap around
  - Parity, 32-bit granularity
  - Buffer can be bypassed
  - Buffer can be globally invalidated
- 256-bit Data Buffer
  - Single line
  - Critical word first
  - Streaming[1)]
  - Line Wrap around
  - Buffer can be bypassed

1) The first 32-bit data from Flash gets immediately forwarded to the CPU

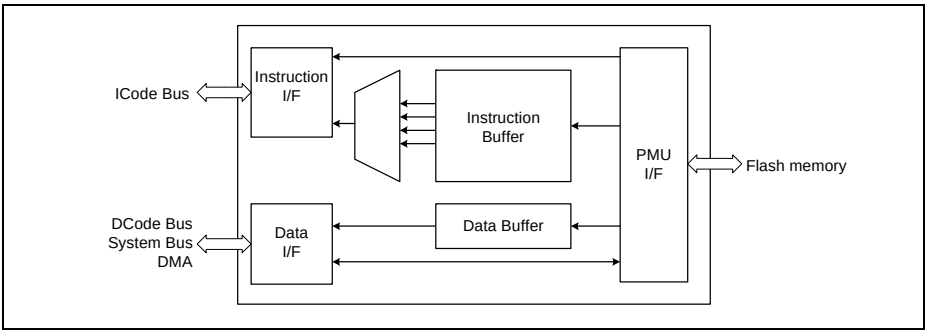


Figure 8-2 Prefetch Unit

8.3.2 Operation

8.3.2.1 Instruction Buffer

The instruction buffer acts like a regular instruction cache with the characteristics described in the overview, optimized for minimum latency via the dedicated instruction interface. Instruction fetches to the non-cacheable address space bypass the instruction buffer. For software development and benchmarking purposes the cacheable accesses can also bypass the instruction buffer by setting **PREF_PCON**.IBYP to 1_B.

Prefetch buffer hits are without any penalty i.e. single cycle access rate. This ensures a minimized latency.

The instruction buffer may be invalidated by writing a 1_B to **PREF_PCON**.IINV. After system reset, the instruction buffer is automatically invalidated.

A parity error during a buffer read operation is automatically turned into a buffer miss, triggering a refill operation of the cache line.

Note: The complete invalidation operation is performed in a single cycle.

Note: The parity information is generated on the fly during the cache refill operation. Parity is checked for each read operation targeting the instruction buffer.

The streaming operation is on the fly - it does not cause any additional latency.

8.3.2.2 Data Buffer

The characteristics of the data buffer are described in the overview. It is used for data read requests from the CPU using the DCode interface and for data read requests from

the DMA. CPU read accesses to the prefetch buffer are without any penalty i.e. single cycle access rate. The miss latency is minimized.

The data interface is shared between DMA requests, CPU DCode bus requests and CPU System bus requests. The CPU System bus is attached to the Prefetch unit to access configuration and status registers within the Prefetch unit and the PMU and Flash. All read requests outside the cacheable address space and all write accesses bypass the data buffer. For software development and benchmarking purposes the cacheable accesses can also bypass the data buffer by setting **PREF_PCON**.DBYP to 1_B.

Note: The streaming operation is on the fly - it does not cause any additional latency.

8.3.2.3 PMU Interface

Each Flash read access returns 256 bits, intermediately stored in a “global read buffer” in the Flash (**Section 8.4.4**). The Prefetch unit reads from this buffer via a 64-bit interface. Cacheable read accesses that are not yet stored in the Prefetch buffer (cache miss) trigger a refill operation by a 4x64-bit burst transfer. By that burst transfer the data from the global buffer is copied, refilling the instruction buffer (code fetch) or data buffer (data fetch) respectively.
Only the initial Flash read access is affected by the Flash latency. The subsequent read accesses of the burst transfer are serviced by the global read buffer with no additional delay. An additional prefetch mechanism in the PFLASH further reduces the latency for linear Flash accesses (**Section 8.4.4**).

Non-cacheable accesses benefit from the global read buffer in the same way, as long as its content is not “trashed” by a new Flash read access (e.g. from a different bus master).

Accesses to the BROM and register address spaces and write operations are ignored by the Prefetch buffers.

8.4 Program Flash (PFLASH)

This chapter describes the embedded Flash module of the XMC4[12]00 and its software interface.

8.4.1 Overview

The embedded Flash module of XMC4[12]00 includes 256 KB of Flash memory for code or constant data (called Program Flash).

8.4.1.1 Features

The following list gives an overview of the features implemented in the Program Flash. Absolute values can be found in the “Data Sheet”.

- Consists of one bank.
- Commonly used for instructions and constant data.
- High throughput burst read based on a 256-bit Flash access.
- Application optimized sector structure with sectors ranging from 16 Kbytes to 256 Kbytes.
- High throughput programming of a 256 byte page (see Data Sheet t_{PRP}).
- Sector-wise erase on logical and physical sectors (see Data Sheet t_{ERP}).
- Write protection separately configurable for groups of sectors.
- Hierarchical write protection control with 3 levels of which 2 are password based and 1 is a one-time programmable one.
- Password based read protection combined with write protection for the whole Flash.
- Separate configuration sector containing the protection configuration and boot configuration (BMI).
- All Flash operations initiated by command sequences as protection against unintended operation.
- Erase and program performed by a Flash specific control logic independent of the CPU.
- End of erase and program operations reported by interrupt.
- Dynamic Error Correcting Code (ECC) with Single-bit Error Correction and Double-bit Error Detection (“SEC-DED”).
- Error reporting by bus error, interrupts and status flags.
- Margin reads for quality assurance.
- Delivery in the erased state.
- Configurable wait state configuration for optimum read performance depending on CPU frequency (see **FCON**.WSPFLASH).
- High endurance and long retention.
- Pad supply voltage used for program and erase.

8.4.2 Definition of Terms

The description of Flash memories uses a specific terminology for operations and the hierarchical structure.

Flash Operation Terms

- **Erasing**: The erased state of a Flash cell is logical ‘0’. Forcing a cell to this state is called “erasing”. Depending on the Flash area and command sequence complete logical or physical sectors are erased. All Flash cells in this area incur one “cycle” that counts for the “endurance”.
- **Programming**: The programmed state of a cell is logical ‘1’. Changing an erased Flash cell to this state is called “programming”. The 1-bits of a page are programmed concurrently.
- **Retention**: This is the time during which the data of a Flash cell can be read reliably. The retention time is a statistical figure that depends on the operating conditions of the device (e.g. temperature profile) and is affected by operations on other Flash cells in the same word-line and physical sector. With an increasing number of program/erase cycles (see endurance) the retention is lowered. Figures are documented in the Data Sheet separately for physical sectors (t_{RET}) and UCBs (t_{RTU}).
- **Endurance**: The maximum number of program/erase cycles of each Flash cell is called “endurance”. The endurance is a statistical figure that depends on operating conditions and the use of the flash cells and also on the required quality level. The endurance is documented in the Data Sheet as a condition to the retention parameters.

Flash Structure Terms

- **Flash Module**: The PMU contains one “Flash module” with its own operation control logic.
- **Bank**: A “Flash module” may contain separate “banks”. “Banks” support concurrent operations (read, program, erase) with some limitations due to common logic.
- **Physical Sector**: A Flash “bank” consists of “physical sectors” ranging from 64 Kbytes to 256 Kbytes. The Flash cells of different “physical sectors” are isolated from each other. Therefore cycling Flash cells in one physical sectors does not affect the retention of Flash cells in other physical sectors. A “physical sector” is the largest erase unit.
- **Logical Sector**: A “logical sector” is a group of word-lines of one physical sector. They can be erased with a single operation but other Flash cells in the same physical sector are slightly disturbed.
- **Sector**: The plain term “sector” means “logical sector” when a physical sector is divided in such, else it means the complete physical sector.
- **User Configuration Block “UCB”**: A “UCB” is a specific logical sector contained in the configuration sector. It contains the protection settings and other data configured

by the user. The “UCBs” are the only part of the configuration sector that can be programmed and erased by the user.

- **Word-Line**: A “word-line” consists of two pages, an even one and an odd one. In the PFLASH a word-line contains aligned 512 bytes.
- **Page**: A “page” is a part of a word-line that is programmed at once. In PFLASH a page is an aligned group of 256 bytes.

8.4.3 Flash Structure

The PMU contains one PFLASH bank, accessible via the cacheable or non-cacheable address space. The offset address of each sector is relative to the base address of its bank which is given in **Table 8-1**.

Derived devices (see Data Sheet) can have less Flash memory. The PFLASH bank shrinks by cutting-off higher numbered physical sectors.

Table 8-1 Flash Memory Map

Range Description	Size	Start Address
PMU0 Program Flash Bank non-cached	256 Kbyte	$0C00\ 0000_H$
PMU0 Program Flash Bank cached space (different address space for the same physical memory, mapped in the non-cached address space)	256 Kbyte	$0800\ 0000_H$
PMU0 UCB User Configuration Blocks	3 Kbyte	$0C00\ 0000_H$
PMU0 Flash Registers	1 Kbyte	$5800\ 2000_H$

PFLASH

All addresses offset to the start addresses given in **Table 8-1**.

Table 8-2 Sector Structure of PFLASH

Sector	Phys. Sector	Size	Offset Address
S0	PS0	16 KB	$00'0000_H$
S1		16 KB	$00'4000_H$
S2		16 KB	$00'8000_H$
S3		16 KB	$00'C000_H$

Table 8-2 Sector Structure of PFLASH (cont'd)

Sector	Phys. Sector	Size	Offset Address
S4	PS4	16 KB	$01'0000_H$
S5		16 KB	$01'4000_H$
S6		16 KB	$01'8000_H$
S7		16 KB	$01'C000_H$
S8	–	128 KB	$02'0000_H$

UCB

All addresses offset to the start addresses given in **Table 8-1**. As explained before the UCBx are logical sectors.

Table 8-3 Structure of UCB Area

Sector	Size	Offset Address
UCB0	1 KB	$00'0000_H$
UCB1	1 KB	$00'0400_H$
UCB2	1 KB	$00'0800_H$

8.4.4 Flash Read Access

Flash banks that are active and in read mode can be directly read like a ROM.

The wait cycles for the Flash read access must be configured based on the CPU frequency f_{CPU} (incl. PLL jitter) in relation to the Flash access time t_a defined in the Data Sheet. The follwing formula applies for **FCON**.WSPFLASH > 0_H[1)]:

$$WSPFLASH \times (1 / f_{CPU}) \geq t_a \quad (8.1)$$

The PFLASH delivers 256 bits per read access. All read data from the PFLASH passes through a 256-bit "global read buffer".

The PMU allows 4x64-bit burst accesses to the cached address space and single 32-bit read accesses to the non-cached address space of the PFLASH.

The Prefetch generates the 4x64-bit bursts for code and data fetches from the cached address range in order to fill one cache line or the data buffer respectively. Data reads from the non-cached address range are performed with single 32-bit transfers.

Following an inital Flash access, the PFLASH automatically starts a prefetch of the next linear address (even before it has been requested). Has the content of the global read

1) WSPFLASH = 0_H deviates from this formula and results in the same timing as WSPFLASH = 1_H.

buffer been read completely (e.g. by a burst from the Prefetch unit), the new prefetched data is copied to the read buffer and another prefetch to the PFLASH is started. This significantly reduces the Flash latency for mostly linearly accessed code or data sections. To avoid additional wait states due to these prefetches, they can be aborted in case a new (initial) read access is requested from a different address. For power saving purposes these prefetch operations can be disabled by **FCON**.IDLE (**Idle Read Path**).

Read accesses from Flash can be blocked by the read protection (see **Section 8.4.8**).

ECC errors can be detected and corrected (see **Section 8.4.9**).

8.4.5 Flash Write and Erase Operations

Flash write and erase operations are triggered by **Command Sequences** to avoid harm to the stored data by “accidential” accesses from faulty code. Erase operations are executed on sectors, write operations on pages.

Attention: Flash write and erase operations must be executed to the non-cacheable address space.

8.4.6 Modes of Operation

A Flash module can be in one of the following states:

- Active (normal) mode.
- Sleep mode (see **Section 8.6.2**).

In sleep mode write and read accesses to all Flash ranges of this PMU are refused with a bus error.

When the Flash module is in active mode the Flash bank can be in one of these modes:

- Read mode.
- Command mode.

In read mode a Flash bank can be read and command sequences are interpreted. In read mode a Flash bank can additionally enter page mode which enables it to receive data for programming.

In command mode an operation is performed. During its execution the Flash bank reports BUSY in **FSR**. In this mode read accesses to this Flash bank are refused with a bus error. At the end of an operation the Flash bank returns to read mode and BUSY is cleared. Only operations with a significant duration (shown in the command documentation) set BUSY.

Register read and write accesses are not affected by these modes.

8.4.7 Command Sequences

All Flash operations except read are performed with command sequences. When a Flash bank is in read mode or page mode all write accesses to its reserved address

range are interpreted as command cycle belonging to a command sequence. Write accesses to a busy bank cause a sequence error (SQER).

Attention: For the proper execution of the command sequences and the triggered operations f_{CPU} must be equal or above 1 MHz.

Command sequences consist of 1 to 6 command cycles. The command interpreter checks that a command cycle is correct in the current state of command interpretation. Else a SQER is reported.

When the command sequence is accepted the last command cycle finishes read mode and the Flash bank transitions into command mode.

These write accesses must be single transfers and must address the non-cacheable address range.

Generally when the command interpreter detects an error it reports a sequence error by setting **FSR**.SQER. Then the command interpreter is reset and a page mode is left. The next command cycle must be the 1st cycle of a command sequence. The only exception is “Enter Page Mode” when a bank is already in page mode (see below).

8.4.7.1 Command Sequence Definitions

Table 8-4 gives an overview of the supported command sequence, with the following nomenclature:

The parameter “addr” can be one of the following:

- **$CCCC_H$**: The “addr” must point into the bank that performs the operation. The last 16 address bits must match $CCCC_H$. It is recommended to use as address the base address of the bank incremented by $CCCC_H$.
- **PA**: Absolute start address of the Flash page.
- **UCPA**: Absolute start address of a user configuration block page.
- **SA**: Absolute start address of a Flash sector. Allowed are the PFLASH sectors Sx.
- **PSA**: Absolute start address of a physical sector. Allowed are the PFLASH physical sectors PSx.
- **UCBA**: Absolute start address of a user configuration block.

The parameter “data” can be one of the following:

- **WD**: 32-bit write data to be loaded into the page assembly buffer.
- **xxYY**: 8-bit write data as part of a command cycle. Only the byte “YY” is used for command interpretation. The higher order bytes “xx” are ignored.
  - **xx5y**: Specific case for “YY”. The “y” can be “0_H” for selecting the PFLASH bank.
- **UL**: User protection level ($xxx0_H$ or $xxx1_H$ for user levels 0 and 1).
- **PWx**: 32-bit password.

Command Sequence Overview Table

The Table 8-4 summarizes all commands sequences. The following sections describe each command sequence in detail.

Table 8-4 Command Sequences for Flash Control

Command Sequence		1. Cycle	2. Cycle	3. Cycle	4. Cycle	5. Cycle	6. Cycle
Reset to Read	Address Data	.5554 ..xx**F0**					
Enter Page Mode	Address Data	.5554 ..xx**5y**					
Load Page	Address Data	.55F0 **WD**	.55F4 **WD**				
Write Page	Address Data	.5554 ..xx**AA**	.AAA8 ..xx**55**	.5554 ..xx**A0**	**PA** ..xx**AA**		
Write User Configuration Page	Address Data	.5554 ..xx**AA**	.AAA8 ..xx**55**	.5554 ..xx**C0**	**UCPA** ..xx**AA**		
Erase Sector	Address Data	.5554 ..xx**AA**	.AAA8 ..xx**55**	.5554 ..xx**80**	.5554 ..xx**AA**	.AAA8 ..xx**55**	**SA** ..xx**30**
Erase Physical Sector	Address Data	.5554 ..xx**AA**	.AAA8 ..xx**55**	.5554 ..xx**80**	.5554 ..xx**AA**	.AAA8 ..xx**55**	**SA** ..xx**40**
Erase User Configuration Block	Address Data	.5554 ..xx**AA**	.AAA8 ..xx**55**	.5554 ..xx**80**	.5554 ..xx**AA**	.AAA8 ..xx**55**	**UCBA** ..xx**C0**
Disable Sector Write Protection	Address Data	.5554 ..xx**AA**	.AAA8 ..xx**55**	.553C **UL**	.AAA8 **PW 0**	.AAA8 **PW 1**	.5558 ..xx**05**
Disable Read Protection	Address Data	.5554 ..xx**AA**	.AAA8 ..xx**55**	.553C ..xx**00**	.AAA8 **PW 0**	.AAA8 **PW 1**	.5558 ..xx**08**
Resume Protection	Address Data	.5554 ..xx**5E**					
Clear Status	Address Data	.5554 ..xx**F5**					

Reset to Read

This function resets the command interpreter to its initial state (i.e. the next command cycle must be the 1st cycle of a sequence). A page mode is aborted.

This command is the only one that is accepted without generating a SQER when the command interpreter has already received command cycles of a different sequence but

is still not in command mode. Thus “Reset to Read” can cancel every command sequence before its last command cycle has been received.

The error flags of **FSR** (PFOPER, SQER, PROER, PFDBER, ORIER, VER) are cleared. The flags can be also cleared in the status registers without command sequence.

If any Flash bank is busy this command is executed but the flag SQER is set.

Enter Page Mode

The PFLASH enters page mode. The selection of the PFLASH assembly buffer (256 bytes) is additionally done by the parameter “$y_H = 0_H$”.

The write pointer of the page assembly buffer is set to 0, its previous content is maintained.

The page mode is signalled by the flag PAGEx in the FSR.

If a new “Enter Page Mode” command sequence is received while any Flash bank is already in page mode SQER is set but this sequence is correctly executed (i.e. in this case the command interpreter is not reset).

Load Page

Loads the data “WD” into the page assembly buffer. It is required to transfer 64-bit with

two consecutive 32-bit data transfers, first addressing the low word with $55F0_H$, followed by the high word with $55F4_H$. The 64-bit are then transfered to the assembly buffer and the write pointer is incremented to the next position.

32 “Load Page” operations are required to fill the assembly buffer for one 256 byte page.

The addressed bank must be in page mode, else SQER is issued.

If “Load Page” is called more often than necessary for filling the page SQER is issued and if configured an interrupt is triggered. The overflow data is discarded. The page mode is not left.

Write Page

This function starts the programming process for one page with the data transferred previously by “Load Page” commands. Upon entering command mode the page mode is finished (indicated by clearing the corresponding PAGE flag) and the BUSY flag of the bank is set.

This command is refused with SQER when the addressed Flash bank is not in page mode.

SQER is also issued when PA addresses an unavailable Flash range or when PA does not point to a legal page start address.

If after “Enter Page Mode” too few data or no data was transferred to the assembly buffer with “Load Page” then “Write Page” programs the page but sets SQER. The missing data is programmed with the previous content of the assembly buffer.

When the page “PA” is located in a sector with active write protection or the Flash module has an active global read protection the execution fails and PROER is set.

Write User Configuration Page

As for “Write Page”, except that the page “UCPA” is located in a user configuration block. This changes the Flash module’s protection configuration.

When the page “UCPA” is located in an UCB with active write protection or the Flash module has an active global read protection the execution fails and PROER is set.

When UCPA is not the start address of a page in a valid UCB the command fails with SQER.

Erase Sector

The sector “SA” is erased.

SQER is returned when SA does not point to the base address of a correct sector (as specified at the beginning of this section) or to an unavailable sector.

When SA has an active write protection or the Flash module has an active global read protection the execution fails and PROER is set.

Erase Physical Sector

The physical sector “PSA” is erased.

SQER is returned when PSA does not point to the base address of a correct sector (as specified at the beginning of this section) or an unavailable sector.

When PSA has an active write protection or the Flash module has an active global read protection the execution fails and PROER is set.

Erase User Configuration Block

The addressed user configuration block “UCB” is erased.

When the UCB has an active write protection or the Flash module has an active global read protection the execution fails and PROER is set.

The command fails with SQER when UCBA is not the start address of a valid UCB.

Disable Sector Write Protection

The sector write protection belonging to user level “UL” is temporarily disabled by setting FSR.WPRODIS when the passwords PW0 and PW1 match their configured values in the corresponding UCB.

The command fails by setting PROER when any of PW0 and PW1 does not match. In this case until the next application reset all further calls of “Disable Sector Write Protection” and “Disable Read Protection” fail with PROER independent of the supplied password.

Disable Read Protection

The Flash module read protection including the derived module wide write protection are temporarily disabled by setting FSR.RPRODIS when the passwords PW0 and PW1 match their configured values in the UCB0.

The command fails by setting PROER when any of PW0 and PW1 does not match. In this case until the next application reset all further calls of “Disable Sector Write Protection” and “Disable Read Protection” fail with PROER independent of the supplied password.

Resume Protection

This command clears all FSR.WPRODISx and the FSR.RPRODIS effectively enabling again the Flash protection as it was configured.

A FSR.WPRODISx is not cleared when corresponding UCBx is not in the “confirmed” state (see **Section 8.4.8.1**).

Clear Status

The flags FSR.PROG and FSR.ERASE and the error flags of **FSR** (PFOPER, SQER, PROER, PFDBER, ORIER, VER) are cleared. These flags can be also cleared in the status registers without command sequence.

When any Flash bank is busy this command fails by setting additionally SQER.

8.4.7.2 Flash Page Programming Example

Figure 8-3 shows the basic flow of command sequences to program a Flash page. All commands are write accesses to the non-cached Flash address space. E.g. the “Enter Page Mode” command can be executed by a write access to the address 0C00 5554_H with the data 0000 0050_H.

In the first step the Flash bank is switched to Page Mode. Only in this mode the other command sequences are accepted.

Then the data is loaded in the assembly buffer with a series of “Load Page” commands.

The “Write Page” command triggers the actual write operation, transfering the 256 bytes data from the assembly buffer to the addressed page in the Flash.
FSR.PROG is set with the last cycle of the “Write Page” command sequence, indicating that a program operation is started.

While this write operation is processed the Flash bank is not accessible, indicated by the **FSR**.PBUSY flag.

In every step the **FSR**.SQER flag provides a direct feedback on the successful execution of the command sequence.

After the program operation has been completed successfully, the “sticky” status flags like **FSR**.PROG can be cleared by the “Clear Status” sequence, before the next operation is started.

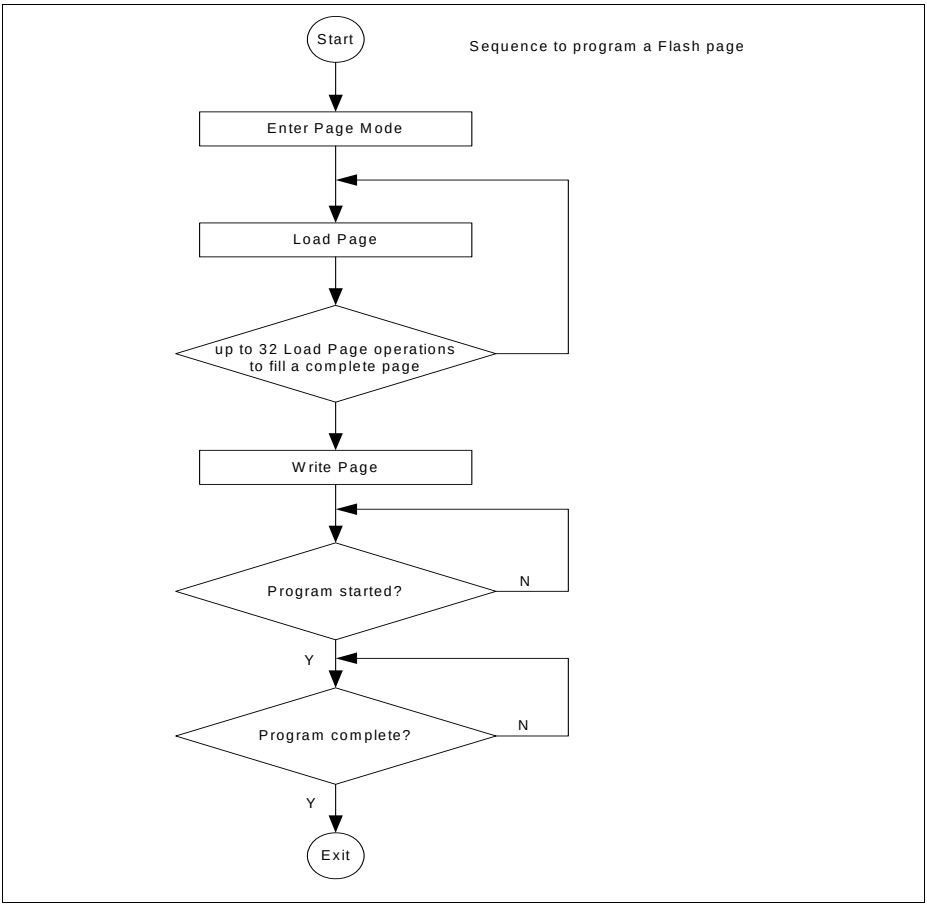


Figure 8-3 Basic Flash Program Sequence

8.4.8 Flash Protection

The Flash memory can be read and write protected. The protection is configured by programming the User Configuration Blocks “UCB”.

For an effective IP protection the Flash read protection must be activated. This ensures system wide that the Flash cannot be read from external or changed without authorization.

8.4.8.1 Configuring Flash Protection in the UCB

As indicated above the effective protection is determined by the content of the **Protection Configuration Indication** PROCON0–2 registers. These are loaded during startup from the UCB0–2. Each UCB comprises 1 Kbyte of Flash organized in 4 UC pages of 256 bytes. The UCBs have the following structure:

Table 8-5 UCB Content

UC Page	Bytes	UCB0	UCB1	UCB2
0	[1:0]	PROCON0	PROCON1	PROCON2
	[7:2]	unused	unused	unused
	[9:8]	PROCON0 (copy)	PROCON1 (copy)	PROCON2 (copy)
	[15:10]	unused	unused	unused
	[19:16]	PW0 of User 0	PW0 of User 1	unused
	[23:20]	PW1 of User 0	PW1 of User 1	unused
	[27:24]	PW0 of User 0 (copy)	PW0 of User 1 (copy)	unused
	[31:28]	PW1 of User 0 (copy)	PW1 of User 1 (copy)	unused
	others	unused	unused	unused
1		unused	unused	BMI and configuration data (details in Startup Mode chapter)
2	[3:0]	confirmation code	confirmation code	confirmation code
	[11:8]	confirmation code (copy)	confirmation code (copy)	confirmation code (copy)
	others	unused	unused	unused
3	unused	unused	unused	unused

If the confirmation code field is programmed with 8AFE 15C3$_H$ the UCB content is “confirmed” otherwise it is “unconfirmed”. The status flags **FSR**.PROIN, **FSR**.RPROIN and **FSR**.WPROIN0–2 indicate this confirmation state:

- **FSR**.PROIN: set when any UCB is in the confirmed state.
- **FSR**.RPROIN: set when **PROCON0**.RPRO is ‘1’ and the UCB0 is in “confirmed” state.
- **FSR**.WPROIN0–2: set when their UCB0–2 is in “confirmed” state.

An UCB can be erased with the command “Erase User Configuration Block”. An UCB page can be programmed with the command “Write User Configuration Page”. These commands fail with PROER when the UCB is write-protected.

An UCB is write-protected if:

- UCB0: (FSR.RPROIN and not **FSR**.RPRODIS) or (**FSR**.WPROIN0 and not **FSR**.WPRODIS0)
- UCB1: **FSR**.WPROIN1 and not **FSR**.WPRODIS1.
- UCB2: **FSR**.WPROIN2

So when the UCB2 is in the “confirmed” state its protection can not be changed anymore. Therefore this realizes a one-time programmable protection.

Changing UCBs

The protection installation is modified by erasing and programming the UCBs with dedicated command sequences, described in **Section 8.4.7.1**. These operations need to be performed with care as described in the following.

Aborting an “Erase UC Block” operation (e.g. due to reset or power failure) must be avoided at all means, as it can result in an unusable device.

UCBs are logical sectors, and as such the allowed number of program/erase cycles of the UCBs must not be exceeded. Over-cycling the UCBs can also lead to an unusable device.

The installation of the protection and its confirmation on different pages of the UCB offers the possibility to check the installation before programming the confirmation. First the protection needs to be programmed, then an application reset must be triggered to trigger the reading of the UCBs by the PMU and after that the protection can be verified (e.g. “Disable … Protection” to check the password and by checking PROCONs and **FCON**). The application reset is inevitable because the PMU reads the UCBs only during the startup phase.

8.4.8.2 Flash Read Protection

Read protection can be activated for the whole Flash module.

Read Protection Status

A read access to PFLASH fails with bus error under the following conditions:

- Code fetch: **FCON**.DCF and **FCON**.RPA.
- Data read: **FCON**.DDF and **FCON**.RPA.

The read protection bit **FCON**.RPA is determined during startup by the protection configuration of UCB0. It can be temporarily modified by the command sequences “Disable Read Protection” and “Resume Protection” which modify **FSR**.RPRODIS. **FCON**.RPA is determined by the following equation:

- **FCON**.RPA = **PROCON0**.RPRO and not **FSR**.RPRODIS.

The bits **FCON**.DDF and **FCON**.DCF are initialized by the startup software depending on the configured protection and the startup mode. They can also be directly modified by the user software under conditions noted in the description of **FCON**.

Initializing Read Protection

Installation of read protection is performed with the “Write User Configuration Page” operation, controlled by the user 0. With this command, user 0 writes the protection configuration bits RPRO, and the two 32-bit keywords into the UCB0 page 0. Additionally, with a second “Write User Configuration Page” command, a special 32-bit confirmation (lock-) code is written into the UCB0 page 2. Only this confirmation code enables the protection and thus the keywords. The confirmation write operation to the second wordline of the User Configuration Block shall be executed only after check of keyword-correctness (with command “Disable Read Protection” after next reset). The confirmed state and thus the installation of protection is indicated with the FSR-bit PROIN in Flash Status Register FSR and for read protection with bit RPROIN in FSR. If read protection is not correctly confirmed and thus not enabled, the bits PROIN and RPROIN in the FSR are not set. The configured read protection as fetched from UCB0 is indicated in the protection configuration register **PROCON0**.

For safety of the information stored in the UCB pages, all keywords, lock bits and the confirmation code are stored two-times in the two wordlines. In case of a disturbed original data detected during ramp up, its copy is used**FSR**. Layout of the four UC pages belonging to the user’s UC block is shown in **Table 8-5**, the command “Write User Configuration Page” is described in **Section 8.4.7.1**.

Disabling Read Protection

With the command sequence “Disable Read Protection” short-term disabling of read protection is possible. This command disables the Flash protection (latest until next reset) for user controlled erase and re-program operations as well as for clearing of DCF and DDF control bits after external program execution. The “Disable Read Protection” command sequence is a protected command, which is only processed by the command state machine, if the included two passwords are identical to the two keywords of user 0.

The disabled state of read protection is controlled with the **FCON**.RPA='0' and indicated in the Flash Status Register **FSR** with the RPRODIS bit (see **Section 8.7.3.1**). As long as read protection is disabled (and thus not active), the **FCON**-bits DDF and DCF can be cleared.

Resumption of read protection after disablement is performed with the "Resume Read/Write Protection" command. After execution of this single cycle command, read protection (if installed) is again active, indicated by the **FCON** bit RPA='1'.

Generally, Flash read protection will remain installed as long as it is confirmed (locked) in the User Configuration Block 0. Erase of UC block and re-program of UC pages can be performed up to 4 times. But note, after execution of the Erase UC block command (which is protected and therefore requires the preceding disable command with the user's specific passwords), all keywords and all protection installations of user 0 are erased; thus, the Flash is no more read protected (beginning with next reset) until re-programming the UC pages. But the division and separation of the protection configuration data and of the confirmation data into two different UCB-wordlines guarantees, that a disturb of keywords can be discovered and corrected before the protection is confirmed. For this reason, the command sequence "Disable Read Protection" can also be used when protection is programmed (configured) but not confirmed; wrong keywords are then indicated by the error flag PROER.

Read protection can be combined with sector specific write protection. In this case, after execution of the command 'Disable Read Protection' only those sectors are unlocked for write accesses, which are not separately write protected.

8.4.8.3 Flash Write and OTP Protection

A range of Flash can be write protected by several means:

- The complete PFLASH can be write protected by the read protection.
- Groups of sectors of PFLASH can be write-protected by three different "users", i.e. UCBs:
  - UCB0: Write protection that can be disabled with the password of UCB0.
  - UCB1: Write protection that can be disabled with the password of UCB1.
  - UCB2: Write protection that can not be disabled anymore (ROM or OTP function: "One-Time Programmable").

Write and OTP Protection Status

An active write protection is indicated by WPROIN bits in **FSR** register. It causes the program and erase command sequences to fail with a PROER.

A range "x" (i.e. a group of sectors, see **PROCON0**) of the PFLASH is write protected if any of the following conditions is true:

- **FCON**.RPA
- **PROCON2**.SxROM

- **PROCON0**.SxL and not(**FSR**.WPRODIS0)
- **PROCON1**.SxL and not(**PROCON0**.SxL) and not(**FSR**.WPRODIS1)

Thus with the password of UCB0 the write protection of sectors protected by user 0 and user 1 can be disabled, however with the password of UCB1 only those sectors that are only protected by user 1. The write protection of user 2 (OTP) can be obviously not disabled. The global write protection caused by the read protection can be disabled as described above by using the password of UCB0 to disable the read protection.

Initialization of Write and OTP Protection

Installation of write protection is performed with the “Write User Configuration Page” operation, controlled by the user. With this command, the user defines and writes into the UCBx page 0 the write protection configuration bits for all sectors, which shall be locked by the specific user, and the user-specific two keywords (not necessary for user 2). The position of sector lock bits is identical as defined for the PROCON registers (**Section 8.7.3.5**). The correctness of keywords shall then (after next reset) be checked with the command ‘Disable Sector Write Protection’, which delivers a protection error PROER in case of wrong passwords. Only if the keywords are correct, the special 32-bit confirmation code must be written into the page 2 of UCBx with a second “Write User Configuration Page” command. Only this confirmation code enables the write protection of the User Control Block UCBx, and only in this case the installation bit(s) in **FSR** is (are) set during ramp up.

Note: If the write protection is configured in the user’s UCB page 0 but not confirmed via page 2 (necessary for check of keywords), the state after next reset is as follows:
- The selected sector(s) are protected (good for testing of protection, but not OTP!)
- The respective PROCON register is set accordingly (also for OTP!)
- The UCBx is not protected, thus it can be erased without passwords
*- The related WPROINx bit in **FSR** is not set*
- The Disable Write Protection command sets the WPRODISx bit
- The Resume command does not clear the WPRODISx bit.

The structure and layout of the three UC blocks is shown in **Table 8-3** below, the command “Write User Configuration Page” is described in **Section 8.4.7.1**.

Disabling Write Protection (not applicable to OTP)

With the command sequence “Disable Sector Write Protection” short-term disabling of write protection for user 0 or user 1 is possible. This command unlocks temporarily all locked sectors belonging to the user. The “Disable Sector Write Protection” command sequence is a protected command, which is only processed by the command state machine, if the included two passwords are correct. The disabled state of sector protection is indicated in the Flash Status Register **FSR** with the WPRODIS bit of the user 0 or/and user 1 (see **Section 8.7.3.1**). For user 2 who owns the sectors with ROM functionality, a disablement of write protection and thus re-programming is not possible.

Resumption of write protection after disablement is performed with the “Resume Read/Write Protection” command, which is identical for user 0 and user 1.

Generally, sector write protection will remain installed as long as it is configured and confirmed in the User Configuration Block belonging to the user. Erase of UC block and re-program of UC pages can be performed up to 4 times, for user 0 and user 1 only. But note, after execution of the Erase UC block command (which is still protected and therefore requires the preceding disablement of write protection with the user’s passwords), the complete protection configuration including the keywords of the specific user (not user 2) is erased; thus, the sectors belonging to the user are unprotected until the user’s UC pages are re-programmed. Only exception: sectors protected by user 2 are locked for ever because the UCB2 can no more be erased after installation of write protection in UCB2.

8.4.8.4 System Wide Effects of Flash Protection

An active Flash read protection needs to be respected in the complete system.

The startup software (SSW) checks if the Flash read protection is active in the PMU, if yes:

- If the selected boot mode executes from internal PFLASH.
  - The SSW clears the DCF and DDF.
  - The SSW leaves the debug interface locked.
- If the selected boot mode does not execute from internal PFLASH:
  - The SSW either leaves DCF and DDF set or actively sets them again in the PMU after evaluating the configuration sector.
  - The debug interface is unlocked.

If the read protection is inactive in the PMU the DCF and DDF flags are cleared by the SSW and the debug interface is unlocked.

Note: Full Flash analysis of an FAR device is only possible when the customer has removed all installed protections or delivers the necessary passwords with the device. As the removal of an OTP protection in UCB2 is not possible the OTP protection inevitably limits analysis capabilities.

8.4.9 Data Integrity and Safety

The data in Flash is stored with error correcting codes “ECC” in order to protect against data corruption. The healthiness of Flash data can be checked with margin checks.

8.4.9.1 Error-Correcting Code (ECC)

The data in the PFLASH is stored with ECC codes. These are automatically generated when the data is programmed. When data is read these codes are evaluated. Data in

PFLASH uses an ECC code with SEC-DED (Single Error Correction, Double Error Detection) capabilities. Each block of 64 data bits is accompanied with 8 ECC bits.

Standard PFLASH ECC

In the standard PFLASH ECC the 8-bit ECC value is calculated over 64 data bits. An erased data block (all bits '0') has an ECC value of 00_H. Therefore an erased sector is free of ECC errors. A data block with all bits '1' has an ECC value of FF_H.

The ECC is automatically generated when programming the PFLASH.

The ECC is automatically evaluated when reading data.

This algorithm has the following capabilities:

- Single-bit error:
  - Is noted in **FSR**.PFSBER.
  - Data and ECC value are corrected.
  - Interrupt is triggered if enabled with **FCON**.PFSBERM.
- Double-bit error:
  - Is noted in **FSR**.PFDBER.
  - Causes a bus error if not disabled by **MARP**.TRAPDIS.
  - Interrupt is triggered if enabled with **FCON**.PFDBERM. This interrupt shall only be used for margin check, when the bus error is disabled.

8.4.9.2 Margin Checks

The Flash memory offers a "margin check feature": the limit which defines if a Flash cell is read as logic '0' or logic '1' can be shifted. This is controlled by the register **MARP**. The Margin Control Register **MARP** is used to change the margin levels for read operations to find problematic array bits. The array area to be checked is read with more restrictive margins. "Problematic" bits will result in a single or double-bit error that is reported to the CPU by an error interrupt or a bus error trap. The double-bit error trap can be disabled for margin checks and also redirected to an error interrupt.

After changing the read margin at least $t_{FL\_MarginDel}$ have to be waited before reading the affected Flash module. During erase or program operation only the standard (default) margins are allowed.

8.5 Service Request Generation

Access and/or operational errors (e.g. wrong command sequences) may be reported to the user by interrupts, and they are indicated by flags in the Flash Status Register **FSR**. Additionally, bus errors may be generated resulting in CPU traps.

8.5.1 Interrupt Control

The PMU and Flash module supports immediate error and status information to the user by interrupt generation. One CPU interrupt request is provided by the Flash module.

The Flash interrupt can be issued because of following events:

- End of busy state: program or erase operation finished
- Operational error (OPER): program or erase operation aborted
- Verify error (VER): program or erase operation not correctly finished
- Protection error
- Sequence error
- Single-bit error: corrected read data from PFLASH delivered
- Double-bit error in Program Flash.

Note: In case of an OPER or VER error, the error interrupt is issued not before the busy state of the Flash is deactivated.

The source of interrupt is indicated in the Flash Status Register **FSR** by the error flags or by the PROG or ERASE flag in case of end of busy interrupt. An interrupt is also generated for a new error event, even if the related error flag is still set from a previous error interrupt.

Every interrupt source is masked (disabled) after reset and can be enabled via dedicated mask bits in the Flash Configuration Register **FCON**.

8.5.2 Trap Control

CPU traps are triggered because of bus errors, generated by the PMU in case of erroneous Flash accesses. Bus errors are generated synchronously to the bus cycle requesting the not allowed Flash access or the disturbed Flash read data. Bus errors are issued because of following events:

- Not correctable double-bit error of 64-bit read data from PFLASH (if not disabled for margin check)
- Not allowed write access to read only register (see **Table 8-11**)
- Not allowed write access to Privileged Mode protected register (see **Table 8-11**)
- Not allowed data or instruction read access in case of active read protection
- Access to not implemented addresses within the register or array space.
- Read-modify-write access to the Flash array.

Write accesses to the Flash array address space are interpreted as command cycles and initiate not a bus error but a sequence error if the address or data pattern is not correct. However, command sequence cycles, which address a busy Flash bank, are serviced with busy cycles, not with a sequence error.

If the trap event is a double-bit error in PFLASH, it is indicated in the **FSR**. With exception of this error trap event, all other trap sources cannot be disabled within the PMU.

*Note: A double-bit error trap during margin check can be disabled (via **MARP** register) and redirected to an interrupt request.*

8.5.3 Handling Errors During Operation

The previous sections described shortly the functionality of “error indicating” bits in the flash status register **FSR**. This section elaborates on this with more in-depth explanation of the error conditions and recommendations how these should be handled by customer software. This first part handles error conditions occurring during operation (i.e. after issuing command sequences) and the second part (**Section 8.5.3.6**) error conditions detected during startup.

8.5.3.1 SQER “Sequence Error”

Fault conditions:

- Improper command cycle address or data, i.e. incorrect command sequence.
- New “Enter Page” in Page Mode.
- “Load Page” and not in Page Mode.
- “Load Page” results in buffer overflow.
- First “Load Page” addresses 2. word.
- “Write Page” with buffer underflow.
- “Write Page” and not in Page Mode.
- “Write Page” to wrong Flash type.
- Byte transfer to password or data.
- “Clear Status” or “Reset to Read” while busy[1)].
- Erase UCB with wrong UCBA.

New state:

Read mode is entered with following exceptions:

- “Enter Page” in Page Mode re-enters Page Mode.
- “Write Page” with buffer underflow is executed.
- After “Load Page” causing a buffer overflow the Page Mode is not left, a following “Write Page” is executed.

Proposed handling by software:

Usually this bit is only set due to a bug in the software. Therefore in development code the responsible error tracer should be notified. In production code this error should not occur. It is however possible to clear this flag with “Clear Status” or “Reset to Read” and simply issue the corrected command sequence again.

1) When the command addresses the busy Flash bank, the access is serviced withbusy cycles.

8.5.3.2 PFOPER “Operation Error”

Fault conditions:

ECC double-bit error detected in Flash module internal SRAM during a program or erase operation in PFLASH. This can be a transient event due to alpha-particles or illegal operating conditions or it is a permanent error due to a hardware defect. This situation will practically not occur.

Attention: these bits can also be set during startup (see **Section 8.5.3.6**).

New state:

The Flash operation is aborted, the BUSY flag is cleared and read mode is entered.

Proposed handling by software:

The flag should be cleared with “Clear Status”. The last operation can be determined from the PROG and ERASE flags. In case of an erase operation the affected physical sector must be assumed to be in an invalid state, in case of a program operation only the affected page. Other physical sectors can still be read. New program or erase commands must not be issued before the next reset.

Consequently a reset must be performed. This performs a new Flash ramp up with initialization of the microcode SRAM. The application must determine from the context which operation failed and react accordingly. Mostly erasing the addressed sector and re-programming its data is most appropriate. If a “Program Page” command was affected and the sector can not be erased the wordline could be invalidated if needed by marking it with all-one data and the data could be programmed to another empty wordline.

Only in case of a defective microcode SRAM the next program or erase operation will incur again this error.

Note: Although this error indicates a failed operation it is possible to ignore it and rely on a data verification step to determine if the Flash memory has correct data. Before re-programming the Flash the flow must ensure that a new reset is applied.

*Note: Even when the flag is ignored it is recommended to clear it. Otherwise all following operations — including “sleep” — could trigger an interrupt even when they are successful (see **Section 8.5.1**, interrupt because of operational error).*

8.5.3.3 PROER “Protection Error”

Fault conditions:

- Password failure.
- Erase/Write to protected sector.
- Erase UCB and protection active.
- Write UC-Page to protected UCB.

Attention: a protection violation can even occur when a protection was not explicitly installed by the user. This is the case when the Flash startup detects an error and starts

the user software with read-only Flash (see **Section 8.5.3.6**). Trying to change the Flash memory will then cause a PROER.

New state:

Read mode is entered. The protection violating command is not executed.

Proposed handling by software:

Usually this bit is only set during runtime due to a bug in the software. In case of a password failure a reset must be performed in the other cases the flag can be cleared with “Clear Status” or “Reset to Read”. After that the corrected sequence can be executed.

8.5.3.4 VER “Verification Error”

Fault conditions:

This flag is a warning indication and not an error. It is set when a program or erase operation was completed but with a suboptimal result. This bit is already set when only a single bit is left over-erased or weakly programmed which would be corrected by the ECC anyhow.

However, excessive VER occurrence can be caused by operating the Flash out of the specified limits, e.g. incorrect voltage or temperature. A VER after programming can also be caused by programming a page whose sector was not erased correctly (e.g. aborted erase due to power failure).

Under correct operating conditions a VER after programming will practically not occur. A VER after erasing is not unusual.

Attention: this bit can also be set during startup (see **Section 8.5.3.6**).

New state:

No state change. Just the bit is set.

Proposed handling by software:

This bit can be ignored. It should be cleared with “Clear Status” or “Reset to Read”. In-spec operation of the Flash memory must be ensured.

If the application allows (timing and data logistics), a more elaborate procedure can be used to get rid of the VER situation:

- VER after program: erase the sector and program the data again. This is only recommended when there are more than 3 program VERs in the same sector. When programming the Flash in field ignoring program VER is normally the best solution because its most likely cause are violated operating conditions. Take care that never a sector is programmed in which the erase was aborted.
- VER after erase: the erase operation can be repeated until VER disappears. Repeating the erase more than 3 times consecutively for the same sector is not recommended. After that it is better to ignore the VER, program the data and check

its readability. Again its most likely cause are violated operating conditions. Therefore it is recommended to repeat the erase at most once or ignore it altogether.

For optimizing the quality of Flash programming see the following section about handling single-bit ECC errors.

*Note: Even when this flag is ignored it is recommended to clear it. Otherwise all following operations — including "sleep" — could trigger an interrupt even when they are successful (see **Section 8.5.1**, interrupt because of verify error).*

8.5.3.5 PFSBER/DFSBER "Single-Bit Error"

Fault conditions:

When reading data or fetching code from PFLASH the ECC evaluation detected a single-bit error ("SBE") which was corrected.

This flag is a warning indication and not an error. A certain amount of single-bit errors must be expected because of known physical effects.

New state:

No state change. Just the bit is set.

Proposed handling by software:

This flag can be used to analyze the state of the Flash memory. During normal operation it should be ignored. In order to count single-bit errors it must be cleared by "Clear Status" or "Reset to Read" after each occurrence[1)].

Usually it is sufficient after programming data to compare the programmed data with its reference values ignoring the SBE bits. When there is a comparison error the sector is erased and programmed again.

When programming the PFLASH (end-of-line programming or SW updates) customers can further reduce the probability of future read errors by performing the following check after programming:

- Change the read margin to "high margin 0".
- Verify the data and count the number of SBEs.
- When the number of SBEs exceeds a certain limit (e.g. 10 in 2 Mbyte) the affected sectors could be erased and programmed again.
- Repeat the check for "high margin 1".
- Each sector should be reprogrammed at most once, afterwards SBEs can be ignored.

1) Further advice: remember that the ECC is evaluated when the data is read from the PMU. When counting single-bit errors use always the non-cached address range otherwise the error count can depend on cache hit or miss and it refers to the complete cache line. As the ECC covers a block of 64 data bits take care to evaluate the **FSR** only once per 64-bit block.

Due to the specificity of each application the appropriate usage and implementation of these measures (together with the more elaborate VER handling) must be chosen according to the context of the application.

8.5.3.6 Handling Flash Errors During Startup

During startup, a fatal error during Flash ramp up forces the Firmware to terminate the startup process and to end in the Debug Monitor Mode (see Firmware chapter).

The reason for a failed Flash startup can be a hardware error or damaged configuration data.

FSR bits set after startup are of informative warning nature.

FSR.PFOPER can indicate a problem of a program/erase operation before the last system reset or an error when restoring the Flash module internal SRAM content after the last reset. In both cases it is advised to clear the flag with the command sequence “Clear Status” and trigger a system reset. If the error shows up again it is an indication for a permanent fault which will limit the Flash operation to read accesses. Under this condition program and erase operations are forbidden (but not blocked by hardware!).

8.6 Power, Reset and Clock

The following chapters describe the required power supplies, the power consumption and its possible reduction, the control of Flash Sleep Mode and the basic control of Reset.

8.6.1 Power Supply

The Flash module uses the standard V_{DDP} I/O power supply to generate the voltages for both read and write functions. Internally generated and regulated voltages are provided for the program and erase operations as well as for read operations. The standard V_{DDC} is used for all digital control functions.

8.6.2 Power Reduction

The “Flash Sleep Mode” can be used to drastically reduce power consumption while the Flash is not accessed for longer periods of time.

The “Idle Read Path” slightly reduces the dynamic power consumption during normal operation with marginal impact on the Flash read performance.

Flash Sleep Mode

As power reduction feature, the Flash module provides the Flash Sleep mode which can be selected by the user individually for the Flash. The Sleep mode can be requested by:

- Programming 1_B to the bit **FCON**.SLEEP.
- "External" sleep mode by the SCU (see "Flash Power Control" in the SCU). Only executed by the Flash when **FCON**.ESLDIS = 0_B.

Attention: f_{CPU} must be equal or above 1 MHz when Sleep mode is requested until the Sleep mode is indicated in FSR.SLM, and when a wake-up request is triggered, until FSR.PBUSY is cleared.

The requested Sleep mode is only taken if the Flash is in idle state and when all pending or active requests are processed and terminated. Only then, the Flash array performs the ramp down into the Sleep mode: the sense amplifiers are switched off and the voltages are ramped down.

During ramp down to Sleep mode **FSR**.PBUSY is set.

As long as the Flash is in Sleep mode, this state is indicated by the bit **FSR**.SLM. The **FSR**.PBUSY stays set as well.

Wake-up from sleep is controlled with clearing of bit **FCON**.SLEEP, if selected via this bit, or wake-up is initiated by releasing the "external" sleep signal from SCU. After wake-up, the Flash enters read mode and is available again after the wake-up time t_{WU}. During the wake-up phase the **FSR**.PBUSY is set until the wake-up process is completed.

Note: During sleep and wake-up, the Flash is reported to be busy. Thus, read and write accesses to the Flash in Sleep mode are acknowledged with busy' and should therefore be avoided; those accesses make sense only during wake-up, when waiting for the Flash read mode.

3. *The wake-up time t_{WU} is documented in the Data Sheet. This time may fully delay the interrupt response time in Sleep mode.*
4. *Note: A wake-up is only accepted by the Flash if it is in Sleep mode. The Flash will first complete the ramp down to Sleep mode before reacting to a wake-up trigger.*

Idle Read Path

An additional power saving feature is enabled by setting **FCON**.IDLE. In this case the PFLASH read path (**Flash Read Access**) is switched off when no read access is pending. System performance for sequential accesses is slightly reduced because internal linear prefetches of the PFLASH are disabled. Non-sequential read accesses requested by the CPU or any other bus master see no additional delay.

8.6.3 Reset Control

All PMU and Flash functionality is reset with the system reset with the exception of the register bits: **FSR**.PROG, **FSR**.ERASE, **FSR**.PFOPER. These bits are reset with the power-on reset.

The flash will be automatically reset to the read mode after every reset.

8.6.3.1 Resets During Flash Operation

A reset or power failure during an ongoing Flash operation (i.e. program or erase) must be considered as violation of stable operating conditions. However the Flash was designed to prevent damage to non-addressed Flash ranges when the reset is applied as defined in the data sheet. The exceptions are erasing logical sectors and UCBs. Aborting an erase process of a logical sector can leave the complete physical sector unreadable.When an UCB erase is aborted the complete Flash can become unusable. So UCBs must be only erased in a controlled environment. The addressed Flash range is left in an undefined state.

When an erase operation is aborted the addressed logical or physical sector can contain any data. It can even be in a state that doesn't allow this range to be programmed.

When a page programming operation is aborted the page can still appear as erased (but contain slightly programmed bits), it can appear as being correctly programmed (but the data has a lowered retention) or the page contains garbage data. It is also possible that the read data is instable so that depending on the operating conditions different data is read.

For the detection of an aborted Flash process the flags **FSR**.PROG and **FSR**.ERASE could be used as indicator but only when the reset was a System Reset. Power-on resets can not be determined from any flags. It is not possible to detect an aborted operation simply by reading the Flash range. Even the margin reads don't offer a reliable indication.

When erasing or programming the PFLASH usually an external instance can notice the reset and simply restart the operation by erasing the Flash range and programming it again.

However, in cases where this external instance is not existing, a common solution is detecting an abort by performing two operations in sequence and determine after reset from the correctness of the second the completeness of the first operation.

E.g. after erasing a sector a page is programmed. After reset the existence of this page proves that the erase process was performed completely.

The detection of aborted programming processes can be handled similarly. After programming a block of data an additional page is programmed as marker. When after reset the block of data is readable and the marker is existent it is ensured that the block of data was programmed without interruption.

If a complete page can be spent as marker, the following recipe allows to reduce the marker size to 8 bytes. This recipe violates the rule that a page may be programmed only once. This violation is only allowed for this purpose and only when the algorithm is robust against disturbed pages (see also recommendations for handling single-bit errors) by repeating a programming step when it detects a failure.

Robust programming of a page of data with an 8 byte marker:

1. After reset program preferably always first to an even page (“Target Page”).
2. If the Other Page on the same wordline contains active data save it to SRAM (the page can become disturbed because of the 4 programming operations per wordline).
3. Program the data to the Target Page.
4. Perform strict check of the Target Page (see below).
5. Program 8 byte marker to Target Page.
6. Perform strict check of the Target Page.
7. In case of any error of the strict check go to the next wordline and program the saved data and the target data again following the same steps.
8. Ensure that the algorithm doesn’t repeat unlimited in case of a violation of operating conditions.

Strict checking of programmed data:

1. Ignore single-bit errors and the VER flag.
2. Switch to tight margin 0.
3. If the data (check the complete page) is not equal to the expected data report an error.
4. If a double-bit error is detected report an error.

After reset the algorithm has to check the last programmed page if it was programmed completely:

1. Read with normal read level. Ignore single-bit errors.
2. Read 8-byte marker and check for double-bit error.
3. Read data part and verify its consistency (e.g. by evaluating a CRC). Check for double-bit error.
4. If the data part is defective don’t use it (e.g. by invalidating the page).
5. If the data part is ok:
   a) If the marker is erased the data part could have been programmed incompletely. Therefore the data part should not be used or alternatively it could be programmed again to a following page.
   b) If the marker contains incorrect data the data part was most likely programmed correctly but the marker was programmed incompletely. The page could be used as is or alternatively the data could be programmed again to a following page.
   c) If the marker is ok the data part was programmed completely and has the full retention. However this is not ensured for the marker part itself. Therefore the algorithm must be robust against the case that the marker becomes unreadable later.

8.6.4 Clock

The Flash interface is operating at the same clock speed as the CPU, f_{CPU}. Depending on the frequency, wait states must be inserted in the Flash accesses. Further details onthe wait states configuration are give in **Section 8.4.4**.

For proper operation of command sequences and when entering or waking up from Sleep mode, f_{CPU} must be equal or above 1 MHz.

8.7 Registers

The register set consists of the PMU ID register (**Section 8.7.1**), the Prefetch Control register (**Section 8.7.2**). The other registers control Flash functionality (**Section 8.7.3**).

All accesses prevented due to access mode restrictions fail with a bus error.

Also accesses to unoccupied register addresses fail with a bus error.

8.7.1 PMU Registers

The PMU only contains the ID register.

Table 8-6 Registers Address Space

Module	Base Address	End Address	Note
reserved	$5800\ 0000_H$	$5800\ 04FF_H$	Bus Error
PMU0	$5800\ 0500_H$	$5800\ 05FF_H$	
reserved	$5800\ 0600_H$	$5800\ 0FFF_H$	Bus Error
reserved	$5800\ 2400_H$	$5800\ 3FFF_H$	Bus Error

Table 8-7 Registers Overview

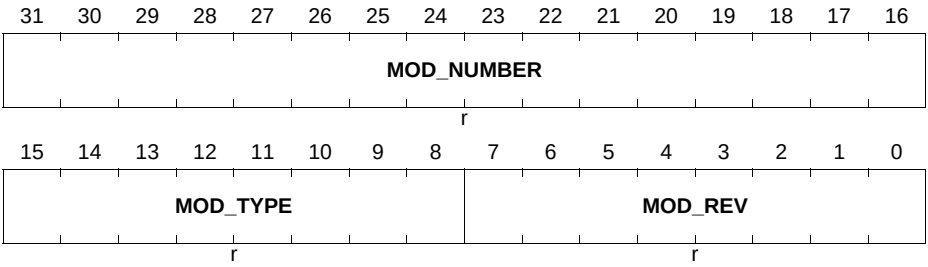
Short Name	Description	Offset Addr.[1]	Access Mode		Reset Class	Page Number
			Read	Write		
ID	Module Identification	08_H	U, PV	BE	System Reset	8-33

1) The absolute register address is calculated as follows:
Module Base Address (**Table 8-6**) + Offset Address (shown in this column)

8.7.1.1 PMU ID Register

The PMU0_ID register is a read-only register, thus write accesses lead to a bus error trap. Read accesses are permitted in Privileged Mode PV and in User Mode. The PMU0_ID register is defined as follows:

PMU0_ID
PMU0 Identification Register (5800 0508$_H$) Reset Value: 009B C0XX$_H$



Field	Bits	Type	Description
MOD_REV	[7:0]	r	**Module Revision Number** MOD_REV defines the module revision number. The value of a module revision starts with 01$_H$ (first rev.).
MOD_TYPE	[15:8]	r	**Module Type** This bit field is C0$_H$. It defines the module as a 32-bit module.
MOD_NUMBER	[31:16]	r	**Module Number Value** This bit field defines the module identification number for PMU0.

8.7.2 Prefetch Registers

This section describes the register of the Prefetch unit.

Table 8-8 Registers Address Space

Module	Base Address	End Address	Note
PREF	5800 4000$_H$	5800 7FFF$_H$	Prefetch Module Registers

Table 8-9 Registers Overview

Short Name	Description	Offset Addr.[1)]	Access Mode		Reset Class	Page Number
			Read	Write		
PCON	Prefetch Configuration Register	0$_H$	U, PV	U, PV	System Reset	Page 8-34

1) The absolute register address is calculated as follows:
Module Base Address (Table 8-6) + Offset Address (shown in this column)

8.7.2.1 Prefetch Configuration Register

This register provides control bits for instruction buffer invalidation and bypass.

PREF_PCON
Prefetch Configuration Register (5800 4000$_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
							0								0
							r								rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
					0						DBYP	0	0	IINV	IBYP
					r						rw	r	r	w	rw

Field	Bits	Type	Description
IBYP	0	rw	**Instruction Prefetch Buffer Bypass** 0_B Instruction prefetch buffer not bypassed. 1_B Instruction prefetch buffer bypassed.
IINV	1	w	**Instruction Prefetch Buffer Invalidate** Write Operation: 0_B No effect. 1_B Initiate invalidation of entire instruction cache.
DBYP	4	rw	**Data Buffer Bypass** 0_B Prefetch Data buffer not bypassed. 1_B Prefetch Data buffer bypassed.
0	16	rw	**Reserved** Must be written with 0.
0	[15:5], 3, 2, [31:17]	r	**Reserved** returns 0 if read; should be written with 0.

8.7.3 Flash Registers

All register addresses are word aligned, independently of the register width. Besides word-read/write accesses, also byte or half-word read/write accesses are supported.

The absolute address of a Flash register is calculated by the base address from **Table 8-10** plus the offset address of this register from **Table 8-11**.

Table 8-10 Registers Address Space

Module	Base Address	End Address	Note
FLASH0	5800 1000_H	5800 $23FF_H$	Flash registers of PMU0

The following table shows the addresses, the access modes and reset types for the Flash registers in PMU0:

Table 8-11 Addresses of Flash0 Registers

Short Name	Description	Address	Access Mode		Reset	See
			Read	Write		
–	Reserved	5800 2000_H – 5800 2004_H	BE	BE	–	–
FLASH0_ ID	Flash Module Identification Register	5800 2008_H	U, PV	BE	System Reset	**Page 8-47**
–	Reserved	5800 $200C_H$	BE	BE	–	–
FLASH0_ FSR	Flash Status Register	5800 2010_H	U, PV	BE	System + PORST	**Page 8-37**
FLASH0_ FCON	Flash Configuration Register	5800 2014_H	U, PV	PV	System Reset	**Page 8-43**
FLASH0_ MARP	Flash Margin Control Register PFLASH	5800 2018_H	U, PV	PV	System Reset	**Page 8-48**
FLASH0_ PROCON0	Flash Protection Configuration User 0	5800 2020_H	U, PV	BE	System Reset	**Page 8-49**
FLASH0_ PROCON1	Flash Protection Configuration User 1	5800 2024_H	U, PV	BE	System Reset	**Page 8-50**
FLASH0_ PROCON2	Flash Protection Configuration User 2	5800 2028_H	U, PV	BE	System Reset	**Page 8-50**
–	Reserved	5800 $202C_H$ – 5800 $23FC_H$	BE	BE	–	

8.7.3.1 Flash Status Definition

The Flash Status Register FSR reflects the overall status of the Flash module after Reset and after reception of the different commands. Sector specific protection states are not indicated in the FSR, but in the registers PROCON0, PROCON1 and PROCON2. The status register is a read-only register. Only the error flags and the two status flags (PROG, ERASE) are affected with the “Clear Status” command. The error flags are also cleared with the “Reset to Read” command.

The FSR is defined as follows:

FSR
Flash Status Register **(1010_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
VER	**X**	**0**	**SLM**	**0**	**WPRODIS1**	**WPRODIS0**	**0**	**WPROIN2**	**WPROIN1**	**WPROIN0**	**0**	**RPRODIS**	**RPROIN**	**0**	**PROIN**
rh	rh	r	rh	r	rh	rh	r	rh	rh	rh	r	rh	rh	r	rh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**PFDBER**	**0**	**PFSBER**	**PROER**	**SQER**	**0**	**PFOPER**	**0**	**PFPAGE**	**ERASE**	**PROG**	**0**	**0**	**FABUSY**	**PBUSY**
r	rh	r	rh	rh	rh	r	rh	r	rh	rh	rh	r	r	rh	rh

Field	Bits	Type	Description
PBUSY[1)]	0	rh	**Program Flash Busy** HW-controlled status flag. 0_B PFLASH ready, not busy; PFLASH in read mode. 1_B PFLASH busy; PFLASH not in read mode. Indication of busy state of PFLASH because of active execution of program or erase operation; PFLASH busy state is also indicated during Flash recovery (after reset) and in power ramp-up state or in sleep mode; while in busy state, the PFLASH is not in read mode.
FABUSY[1)]	1	rh	**Flash Array Busy** Internal busy flag for testing purposes. Must be ignored by application software, which must use PBUSY instead.

Field	Bits	Type	Description
PROG[3)4)]	4	rh	**Programming State** HW-controlled status flag. 0_B There is no program operation requested or in progress or just finished. 1_B Programming operation (write page) requested (from FIM) or in action or finished. Set with last cycle of Write Page command sequence, cleared with Clear Status command (if not busy) or with power-on reset. If one BUSY flag is coincidently set, PROG indicates the type of busy state. If xOPER is coincidently set, PROG indicates the type of erroneous operation. Otherwise, PROG indicates, that operation is still requested or finished.
ERASE[3)4)]	5	rh	**Erase State** HW-controlled status flag. 0_B There is no erase operation requested or in progress or just finished 1_B Erase operation requested (from FIM) or in action or finished. Set with last cycle of Erase command sequence, cleared with Clear Status command (if not busy) or with power-on reset. Indications are analogous to PROG flag.
PFPAGE[1)2)]	6	rh	**Program Flash in Page Mode** HW-controlled status flag. 0_B Program Flash not in page mode 1_B Program Flash in page mode; assembly buffer of PFLASH (256 byte) is in use (being filled up) Set with Enter Page Mode for PFLASH, cleared with Write Page command *Note: Concurrent page and read modes are allowed*
PFOPER[2)3)4)]	8	rh	**Program Flash Operation Error** 0_B No operation error reported by Program Flash 1_B Flash array operation aborted, because of a Flash array failure, e.g. an ECC error in microcode. This bit is not cleared with System Reset, but with power-on reset. Registered status bit; must be cleared per command

Field	Bits	Type	Description
SQER[1)2)3)]	10	rh	**Command Sequence Error** 0_B No sequence error 1_B Command state machine operation unsuccessful because of improper address or command sequence. A sequence error is not indicated if the Reset to Read command aborts a command sequence. Registered status bit; must be cleared per command
PROER[1)2)3)]	11	rh	**Protection Error** 0_B No protection error 1_B Protection error. A Protection Error is reported e.g. because of a not allowed command, for example an Erase or Write Page command addressing a locked sector, or because of wrong password(s) in a protected command sequence such as “Disable Read Protection” Registered status bit; must be cleared per command
PFSBER[1)2)3)]	12	rh	**PFLASH Single-Bit Error and Correction** 0_B No Single-Bit Error detected during read access to PFLASH 1_B Single-Bit Error detected and corrected Registered status bit; must be cleared per command
PFDBER[1)2)3)]	14	rh	**PFLASH Double-Bit Error** 0_B No Double-Bit Error detected during read access to PFLASH 1_B Double-Bit Error detected in PFLASH Registered status bit; must be cleared per command
PROIN	16	rh	**Protection Installed** 0_B No protection is installed 1_B Read or/and write protection for one or more users is configured and correctly confirmed in the User Configuration Block(s). HW-controlled status flag

Field	Bits	Type	Description
RPROIN	18	rh	**Read Protection Installed** 0_B No read protection installed 1_B Read protection and global write protection is configured and correctly confirmed in the User Configuration Block 0. Supported only for the master user (user zero). HW-controlled status flag
RPRODIS[1)5)]	19	rh	**Read Protection Disable State** 0_B Read protection (if installed) is not disabled 1_B Read and global write protection is temporarily disabled. Flash read with instructions from other memory, as well as program or erase on not separately write protected sectors is possible. HW-controlled status flag
WPROIN0	21	rh	**Sector Write Protection Installed for User 0** 0_B No write protection installed for user 0 1_B Sector write protection for user 0 is configured and correctly confirmed in the User Configuration Block 0. HW-controlled status flag
WPROIN1	22	rh	**Sector Write Protection Installed for User 1** 0_B No write protection installed for user 1 1_B Sector write protection for user 1 is configured and correctly confirmed in the User Configuration Block 1. HW-controlled status flag
WPROIN2	23	rh	**Sector OTP Protection Installed for User 2** 0_B No OTP write protection installed for user 2 1_B Sector OTP write protection with ROM functionality is configured and correctly confirmed in the UCB2. The protection is locked for ever. HW-controlled status flag

Field	Bits	Type	Description
WPRODIS0[1)5)]	25	rh	**Sector Write Protection Disabled for User 0** 0_B All protected sectors of user 0 are locked if write protection is installed 1_B All write-protected sectors of user 0 are temporarily unlocked, if not coincidently locked by user 2 or via read protection. Hierarchical protection control: User-0 sectors are also unlocked, if coincidently protected by user 1. But not vice versa. HW-controlled status flag
WPRODIS1[1)5)]	26	rh	**Sector Write Protection Disabled for User 1** 0_B All protected sectors of user 1 are locked if write protection is installed 1_B All write-protected sectors of user 1 are temporarily unlocked, if not coincidently locked by user 0 or user 2 or via read protection. HW-controlled status flag
SLM[1)]	28	rh	**Flash Sleep Mode** HW-controlled status flag. Indication of Flash sleep mode taken because of global or individual sleep request; additionally indicates when the Flash is in shut down mode. 0_B Flash not in sleep mode 1_B Flash is in sleep or shut down mode
X	30	rh	**Reserved** Value undefined
VER[1)3)]	31	rh	**Verify Error** 0_B The page is correctly programmed or the sector correctly erased. All programmed or erased bits have full expected quality. 1_B A program verify error or an erase verify error has been detected. Full quality (retention time) of all programmed (“1”) or erased (“0”) bits cannot be guaranteed. See **Section 8.5.3** and **Section 8.5.3.6** for proper reaction. Registered status bit; must be cleared per command

Field	Bits	Type	Description
0	2,3,7, 9,13, 15,17, 20,24, 27, 29	r	**Reserved** Read zero, no write

Note: The footnote numbers of FSR bits describe the specific reset conditions:
1)Cleared with System Reset
2)Cleared with command “Reset to Read”
3)Cleared with command “Clear Status”
4)Cleared with power-on reset (PORST)
5)Cleared with command “Resume Protection”

Note: The xBUSY flags as well as the protection flags cannot be cleared with the “Clear Status” command or with the “Reset to Read” command. These flags are controlled by HW.

*Note: The reset value above is indicated after correct execution of Flash ramp up. Additionally, errors are possible after ramp up (see **Section 8.5.3.6**).*

8.7.3.2 Flash Configuration Control

The Flash Configuration Register FCON reflects and controls the following general Flash configuration functions:

- Number of wait states for Flash accesses.
- Indication of installed and active read protection.
- Instruction and data access control for read protection.
- Interrupt mask bits.
- Power reduction and shut down control.

FCON is a Privileged Mode protected register. It is defined as follows:

FCON
Flash Configuration Register (1014_H) Reset value: 000X 0006$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EOBM	0	PFDBERM	0	PFSBERM	PROERM	SQERM	VOPERM	0	0	0	0	0	DDF	DCF	RPA
rw	r	rw	r	rw	rw	rw	rw	r	r	r	r	r	rwh	rwh	rh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SLEEP	ESLDIS	IDLE	0								WSECPF	WSPFLASH			
rw	rw	rw	r				r				rw	rw			

Field	Bits	Type	Description
WSPFLASH	[3:0]	rw	**Wait States for read access to PFLASH** This bit field defines the number of wait states n, which are used for an initial read access to the Program Flash memory area, with WSPFLASH x (1 / f_{CPU}) ≥ t_a[1)]. 0000_B PFLASH access in one clock cycle 0001_B PFLASH access in one clock cycle 0010_B PFLASH access in two clock cycles 0011_B PFLASH access in three clock cycles .... PFLASH access in four up to fourteen clock cycles. 1111_B PFLASH access in fifteen clock cycles.
WSECPF	4	rw	**Wait State for Error Correction of PFLASH** 0_B No additional wait state for error correction 1_B One additional wait state for error correction during read access to Program Flash. If enabled, this wait state is only used for the first transfer of a burst transfer. Set this bit only when requested by Infineon.
IDLE	13	rw	**Dynamic Flash Idle** 0_B Normal/standard Flash read operation 1_B Dynamic idle of Program Flash enabled for power saving; static prefetching disabled

Field	Bits	Type	Description
ESLDIS	14	rw	**External Sleep Request Disable** 0_B External sleep request signal input is enabled 1_B Externally requested Flash sleep is disabled The ‘external’ signal input is connected with a global power-down/sleep request signal from SCU.
SLEEP	15	rw	**Flash SLEEP** 0_B Normal state or wake-up 1_B Flash sleep mode is requested Wake-up from sleep is started with clearing of the SLEEP-bit.
RPA	16	rh	**Read Protection Activated** This bit monitors the status of the Flash-internal read protection. This bit can only be ‘0’ when read protection is not installed or while the read protection is temporarily disabled with password sequence. 0_B The Flash-internal read protection is not activated. Bits DCF, DDF are not taken into account. Bits DCF, DDFx can be cleared 1_B The Flash-internal read protection is activated. Bits DCF, DDF are enabled and evaluated.
DCF	17	rwh	**Disable Code Fetch from Flash Memory** This bit enables/disables the code fetch from the internal Flash memory area. Once set, this bit can only be cleared when RPA=’0’. This bit is automatically set with reset and is cleared during ramp up, if no RP installed, and during startup (BROM) in case of internal start out of Flash. 0_B Code fetching from the Flash memory area is allowed. 1_B Code fetching from the Flash memory area is not allowed. This bit is not taken into account while RPA=’0’.

Field	Bits	Type	Description
DDF	18	rwh	**Disable Any Data Fetch from Flash** This bit enables/disables the data read access to the Flash memory area (Program Flash and Data Flash). Once set, this bit can only be cleared when RPA='0'. This bit is automatically set with reset and is cleared during ramp up, if no RP installed, and during startup (BROM) in case of internal start out of Flash. 0_B Data read access to the Flash memory area is allowed. 1_B Data read access to the Flash memory area is not allowed. This bit is not taken into account while RPA='0'.
VOPERM	24	rw	**Verify and Operation Error Interrupt Mask** 0_B Interrupt not enabled 1_B Flash interrupt because of Verify Error or Operation Error in Flash array (FSI) is enabled
SQERM	25	rw	**Command Sequence Error Interrupt Mask** 0_B Interrupt not enabled 1_B Flash interrupt because of Sequence Error is enabled
PROERM	26	rw	**Protection Error Interrupt Mask** 0_B Interrupt not enabled 1_B Flash interrupt because of Protection Error is enabled
PFSBERM	27	rw	**PFLASH Single-Bit Error Interrupt Mask** 0_B No Single-Bit Error interrupt enabled 1_B Single-Bit Error interrupt enabled for PFLASH
PFDBERM	29	rw	**PFLASH Double-Bit Error Interrupt Mask** 0_B Double-Bit Error interrupt for PFLASH not enabled 1_B Double-Bit Error interrupt for PFLASH enabled. Especially intended for margin check
EOBM	31	rw	**End of Busy Interrupt Mask** 0_B Interrupt not enabled 1_B EOB interrupt is enabled
0	[12:5], [23:19], 28, 30	r	**Reserved** Always read/write zero

1) WSPFLASH = 0_H deviates from this formula and results in the same timing as WSPFLASH = 1_H.

Note: The default numbers of wait states represent the slow cases. This is a general proceeding and additionally opens the possibility to execute higher frequencies without changing the configuration.

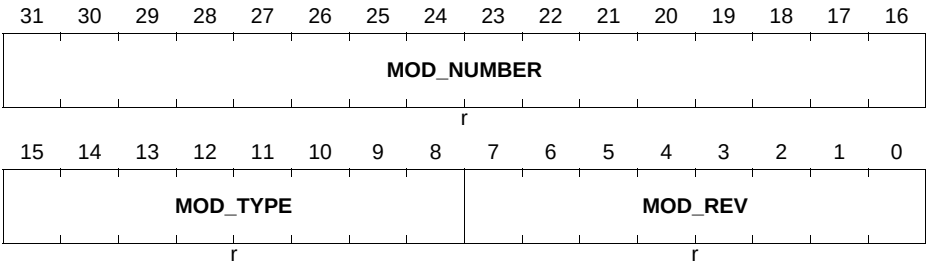
Note: After reset and execution of Firmware, the read protection control bits are coded as follows:
DDF, DCF, RPA = “110”: No read protection installed
DDF, DCF, RPA = “001”: Read protection installed; start in internal Flash
DDF, DCF, RPA = “111”: Read protection installed; start not in internal Flash.

8.7.3.3 Flash Identification Register

The module identification register of Flash module is directly accessible by the CPU via PMU access. This register is mapped into the space of the Flash Interface Module's registers (see **Table 8-11**).

FLASH0_ID

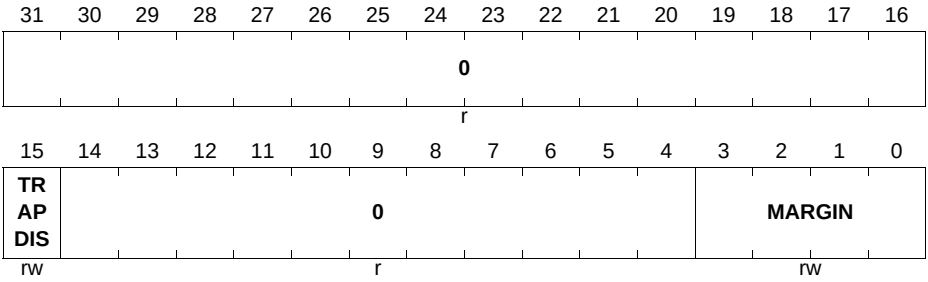
Flash Module Identification Register (1008_H) **Reset Value: $009C\ C0XX_H$**



Field	Bits	Type	Description
MOD_REV	[7:0]	r	**Module Revision Number** MOD_REV defines the module revision number. The value of a module revision starts with 01_H (first revision).
MOD_TYPE	[15:8]	r	**Module Type** This bit field is $C0_H$. It defines the module as a 32-bit module.
MOD_NUMBER	[31:16]	r	**Module Number Value** This bit field defines a module identification number. For the XMC4[12]00 Flash0 this number is $009C_H$.

8.7.3.4 Margin Check Control Register

MARP
Margin Control Register PFLASH (1018$_H$) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
MARGIN	[3:0]	rw	**PFLASH Margin Selection** 0000_B **Default**, Standard (default) margin. 0001_B **Tight0**, Tight margin for 0 (low) level. Suboptimal 0-bits are read as 1s. 0100_B **Tight1**, Tight margin for 1 (high) level. Suboptimal 1-bits are read as 0s. – Reserved.
TRAPDIS	15	rw	**PFLASH Double-Bit Error Trap Disable** 0_B If a double-bit error occurs in PFLASH, a bus error trap is generated[1]. 1_B The double-bit error trap is disabled. Shall be used only during margin check
0	[14:4], [31:16]	r	**Reserved** Always read as 0; should be written with 0.

1) After Boot ROM exit, double-bit error traps are enabled (TRAPDIS = 0).

8.7.3.5 Protection Configuration Indication

The configuration of read/write/OTP protection is indicated with registers PROCON0, PROCON1 and PROCON2, thus separately for every user, and it is generally indicated in the status register FSR.

If write protection is installed for user 0 or 1 or OTP protection for user 2, for each sector of the Program Flash it is indicated in the user-specific Protection Configuration register PROCONx, if it is locked or unlocked for program or erase operations.

The Flash Protection Configuration registers PROCONx are loaded out of the user's configuration block directly after reset during ramp up. For software the three PROCONx registers are read-only registers.

PROCON0
Flash Protection Configuration Register User 0

(1020_H) **Reset Value: 0000 XXXX$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RPRO	0	0	0	0	0	0	S8L	S7L	S6L	S5L	S4L	S3L	S2L	S1L	S0L
rh	r	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
SnL (n=0-8)	n	rh	**Sector n Locked for Write Protection by User 0** These bits indicate whether PFLASH sector n is write-protected by user 0 or not. 0_B No write protection is configured for sector n. 1_B Write protection is configured for sector n.
RPRO	15	rh	**Read Protection Configuration** This bit indicates whether read protection is configured for PFLASH by user 0. 0_B No read protection configured 1_B Read protection and global write protection is configured by user 0 (master user)
0	13, 12, 11, 10, 9	rh	**Reserved** deliver the corresponding UCB0 entry. Shall be configured to 0.
0	[31:16], 14	r	**Reserved** Always reads as 0.

PROCON1
Flash Protection Configuration Register User 1

(1024_H) **Reset Value: 0000 XXXX$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		**0**	**0**	**0**	**0**	**0**	**S8L**	**S7L**	**S6L**	**S5L**	**S4L**	**S3L**	**S2L**	**S1L**	**S0L**
r		rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
SnL (n=0-8)	n	rh	**Sector n Locked for Write Protection by User 1** These bits indicate whether PFLASH sector n is write-protected by user 1 or not. 0_B No write protection is configured for sector n. 1_B Write protection is configured for sector n.
0	13, 12, 11, 10, 9	rh	**Reserved** Deliver the corresponding UCB1 entry. Shall be configured to 0.
0	[31:16], 15, 14	r	**Reserved** Always reads as 0.

PROCON2
Flash Protection Configuration Register User 2

(1028_H) **Reset Value: 0000 XXXX$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**S8 ROM**	**S7 ROM**	**S6 ROM**	**S5 ROM**	**S4 ROM**	**S3 ROM**	**S2 ROM**	**S1 ROM**	**S0 ROM**
r	r	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
SnROM (n=0-8)	n	rh	**Sector n Locked Forever by User 2** These bits indicate whether PFLASH sector n is an OTP protected sector with read-only functionality, thus if it is locked for ever. 0_B No ROM functionality configured for sector n. 1_B ROM functionality is configured for sector n. Re-programming of this sector is no longer possible.
0	13, 12, 11, 10, 9	r	**Reserved** Deliver the corresponding UCB2 entry. Shall be configured to 0.
0	[31:16], 15, 14	r	**Reserved** Always reads as 0.

System Control

9 Window Watchdog Timer (WDT)

Purpose of the Window Watchdog Timer module is improvement of system integrity. WDT triggers the system reset or other corrective action like e.g. non-maskable interrupt if the main program, due to some fault condition, neglects to regularly service the watchdog (also referred to as “kicking the dog”, “petting the dog”, “feeding the watchdog” or “waking the watchdog”). The intention is to bring the system back from unresponsive state into normal operation.

References

[9] Cortex-M4 User Guide, ARM DUI 0508B (ID062910)

9.1 Overview

A successful servicing of the WDT results in a pulse on the signal wdt_service. The signal is offered also as an alternate function output. It can be used to show to an external watchdog that the system is alive.

The WDT timer is a 32-bit counter, which counts up from 0_H. It can be serviced while the counter value is within the window boundary, i.e. between the lower and the upper boundary value. Correct servicing results in a reset of the counter to 0_H. A so called “Bad Service” attempt results in a system reset request.

The timer block is running on the f_{WDT} clock which is independent from the bus clock. The timer value is updated in the corresponding register **TIM**, whenever the timer value increments. This mechanism enables immediate response on a read access from the bus.

The WDT module provides register interface for configuration. A write to writable registers is only allowed, when the access is in privileged mode. A write access in user mode results in a bus error response.

9.1.1 Features

The watchdog timer (WDT) is an independent window watchdog timer.

The features are:

- Triggers system reset when not serviced on time or serviced in a wrong way
- Servicing restricted to be within boundaries of a user definable refresh window
- Can run from an independent clock
- Provides service indication to an external pin
- Can be suspended in HALT mode
- Provides optional pre-warning alarm before reset

Table 9-1 Application Features

Feature	Purpose/Application
System reset upon Bad Servicing	Triggered to restore system stable operation and ensure system integrity
Servicing restricted to be within defined boundaries of refresh window	Allows to consider minimum and maximum software timing
Independent clocks	To ensure that WDT counts even in case of the system clock failure
Service indication on external pin	For dual-channel watchdog solution, additional external control of system integrity
Suspending in HALT mode	Enables safe debugging with productive code
Pre-warning alarm	Software recovery to allow corrective action via software recovery routine bringing system back from the unresponsive state into normal operation

9.1.2 Block Diagram

The WDT block diagram is shown in **Figure 9-1**.

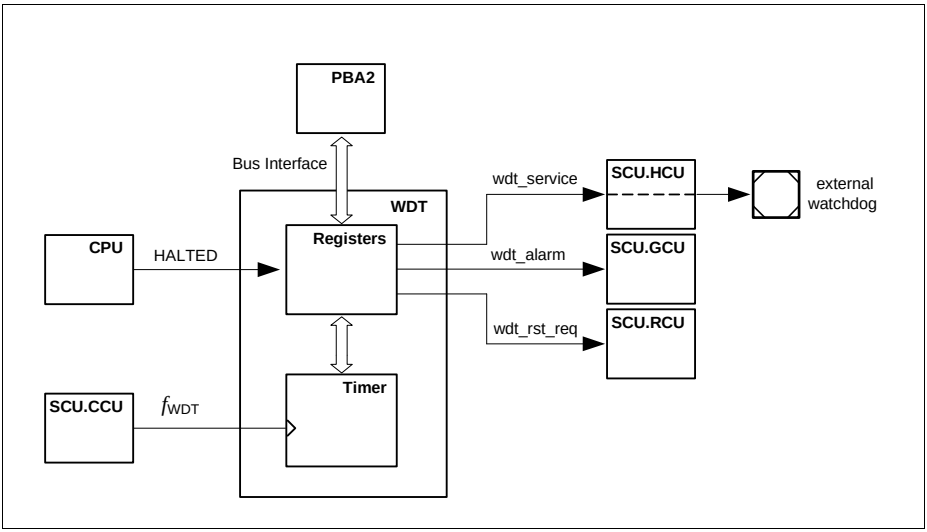


Figure 9-1 Watchdog Timer Block Diagram

9.2 Time-Out Mode

An overflow results in an immediate reset request going to the RCU of the SCU via the signal wdt_rst_req whenever the counter crosses the upper bonundary it triggers an overflow event pre-warning is not enabled with **CTR** register. A successful servicing performed with writing a unique value, referred to as “Magic Word” to the **SRV** register of the WDT within the valid servicing window, results in a pulse on the signal wdt_service and reset of the timer counter.

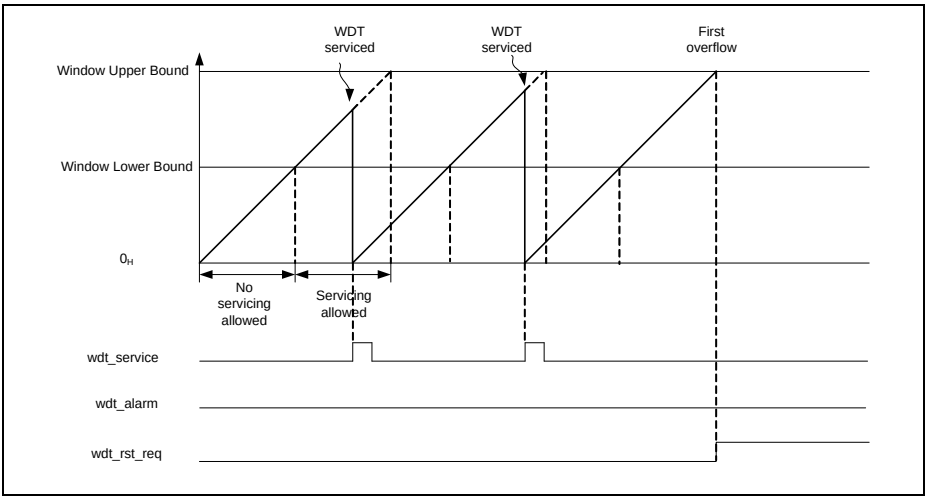


Figure 9-2 Reset without pre-warning

The example scenario depicted in **Figure 9-2** shows two consecutive service pulses generated from WDT module as the result of successful servicing within valid time windows. The situation where no service has been performed immediately triggers generation of reset request on the wdt_rst_req output after the counter value has exceeded window upper bound value.

9.3 Pre-warning Mode

While in pre-warning mode the effect of the overflow event is different with and without pre-warning enabled. The first crossing of the upper bound triggers the outgoing alarm signal wdt_alarm when pre-warning is enabled. Only the next overflow results a reset request. The alarm status is shown via register **WDTSTS** and can be cleared via register **WDTCLR**. A clear of the alarm status will bring the WDT back to normal state. The alarm signal is routed as request to the SCU, where it can be promoted to NMI.

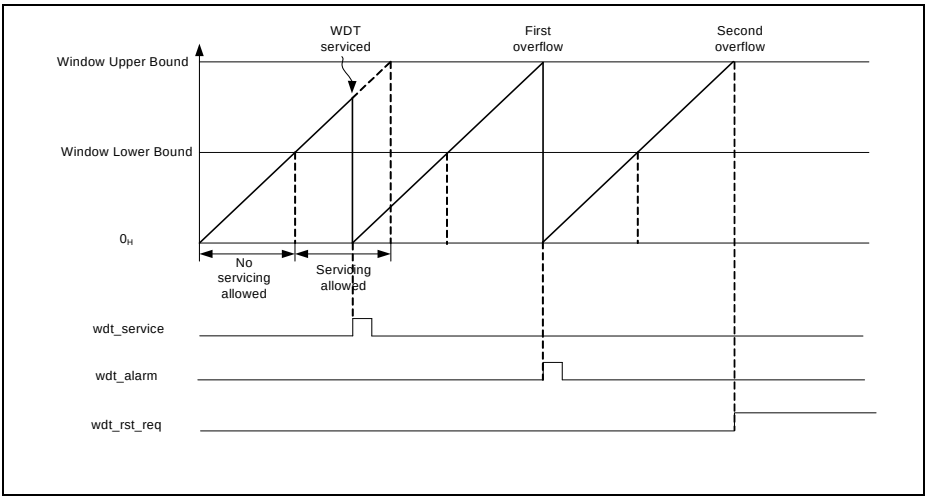


Figure 9-3 Reset after pre-warning

The example scenario depicted in **Figure 9-3** shows service pulse generated from WDT module as the result of successful servicing within valid time window. WDT generates alarm pulse on wdt_alarm upon first missing servicing. The alarm signal is routed as interrupt request to the SCU, where it can be promoted to NMI. Within this alarm service request the user can clear the WDT status bit and give a proper WDT service before it overflows next time. Otherwise WDT generates reset request on wdt_rst_req upon the second missing service.

9.4 Bad Service Operation

A bad service attempt results in a reset request. A bad service attempt can be due to servicing outside the window boundaries or servicing with an invalid Magic Word.

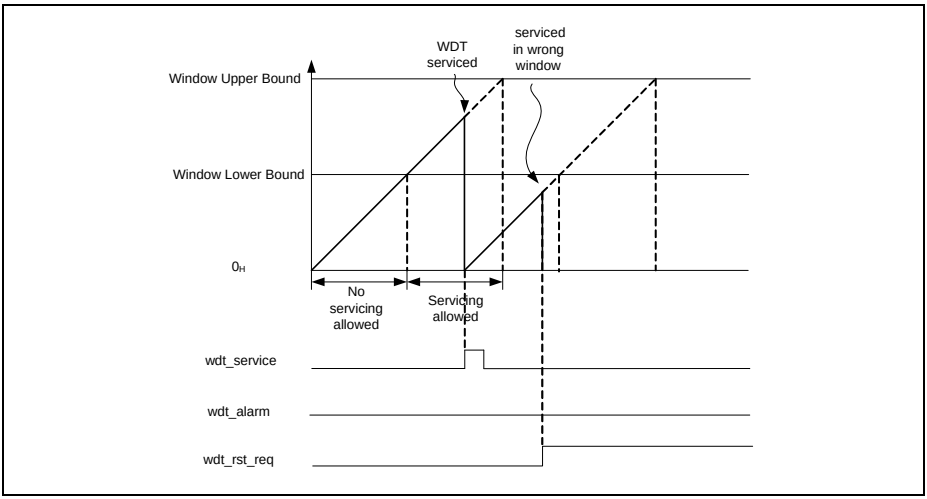


Figure 9-4 Reset upon servicing in a wrong window

The example in **Figure 9-4** shows servicing performed outside of valid servicing window. Attempt to service WDT while counter value remains below the Window Lower Bound results in immediate reset request on wdt_rst_req signal.

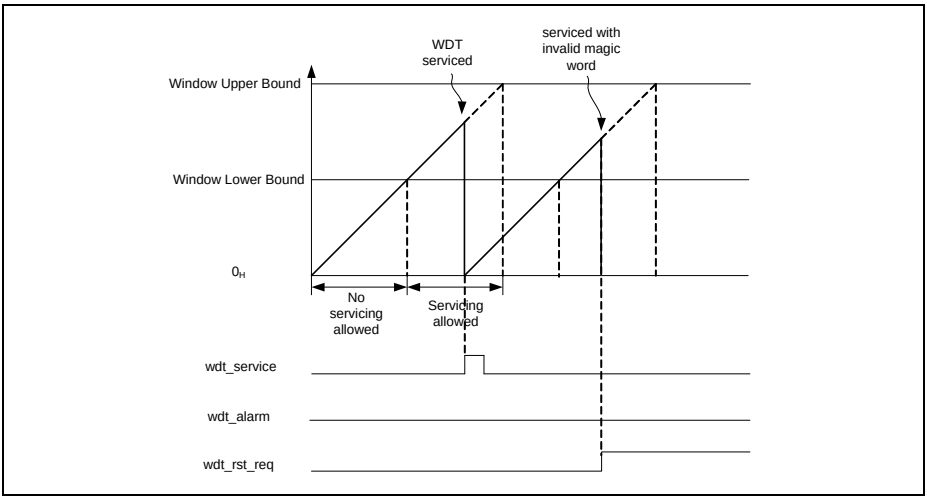


Figure 9-5 Reset upon servicing with a wrong magic word

The example in **Figure 9-5** shows servicing performed within a valid servicing window but with an invalid Magic Word. Attempt to write a wrong word to the **SRV** register results in immediate reset request on wdt_rst_req signal.

9.5 Service Request Processing

The WDT generates watchdog alarm service requests via wdt_alarm output signal upon first counter overflow over Watchdog Upper Bound when pre-warning mode is enabled. The alarm service request may be promoted by the SCU in two alternative modes:

- service request
- trap request causing NMI interrupt

Service requests can be disabled i SCU with service request mask or trap request disable registers respectively.

9.6 Debug Behavior

The WDT function can be suspended when the CPU enters HALT mode. WDT debug function is controlled by DSP bit field in **CTR** register.

9.7 Power, Reset and Clock

The WDT module is a part of the core domain and supplied with VDDC voltage.

All WDT registers get reset with the system reset.

A sticky bit in the RSSTAT register of SCU/RCU module indicates whether the last system reset has been triggered by the WDT module. This bit does not get reset with system reset.

The input clock of the WDT counter can be selected by the user between system PLL output, direct output of the internal system oscillator or 32kHz clock of hibernate domain, independently from the AHB interface clock. Selection of the WDT input clock is performed in SCU using WDTCLKCR register (for details please refer to the SCU/CCU chapter).

9.8 Initialization and Control Sequence

The programming model of the WDT module assumes several scenarios where different control sequences apply.

Note: Some of the scenarios described in this chapter require operations on system level the that are not in the scope of the WDT module description, therefore for detailed information please refer to relevant chapters of this document.

9.8.1 Initialization & Start of Operation

Complete WDT module initialization is required after system reset.

- check reason for last system reset in order to determine power state
  - read out SCU_RSTSTAT.RSTSTAT register bit field to determine last system reset cause
  - perform appropriate operations dependent on the last system reset cause
- WDT software initialization sequence
  - enable WDT clock with SCU_CLKSET.WDTCEN register bit field
  - release WDT reset with SCU_PRCLR2.WDTRS register bit field
  - set lower window bound with WDT_WLB register
  - set upper window bound with WDT_WUB register
  - configure external watchdog service indication (optional, please refer to SCU/HCU chapter)
  - select and enable WDT input clock with SCU_WDTCLKCR register
  - enable system trap for pre-warning alarm on system level with SCU_NMIREQEN register (optional, used in WDT pre-warning mode only)
- software start sequence
  - select mode (Time-Out or Pre-warning) and enable WDT module with WDT_CTR register
- service the watchdog
  - check current timer value in WDT_TIM register against programmed time window
  - write magic word to WDT_SRV register within valid time window

9.8.2 Reconfiguration & Restart of Operation

Reset and initialization of the WDT module is required in order to update its settings.

- software initialization sequence
  - assert WDT reset with SCU_PRSET2.WDTCEN register bit field
  - release WDT reset with SCU_PRCLR2.WDTRS register bit field register
  - set lower window bound with WDT_WLB register
  - set upper window bound with WDT_WUB register
  - configure external watchdog service indication (optional, please refer to SCU/HCU chapter)
  - select and enable WDT input clock (if change of the clock settings required) with SCU_WDTCLKCR register
  - enable system trap for pre-warning alarm on system level with SCU_NMIREQEN register (optional, used in WDT pre-warning mode only)
- software start sequence
  - select mode (Time-Out or Pre-warning) and enable WDT module with WDT_CTR register

- service the watchdog
  – check current timer value in WDT_TIM register against programmed time window
  – write magic word to WDT_SRV register within valid time window

9.8.3 Software Stop & Resume Operation

The WDT module can be stopped and re-started at any point of time for e.g. debug purpose using software sequence.

- software stop sequence
  – disable WDT module with WDT_CTR register
- perform any user operations
- software start (resume) sequence
  – enable WDT module with WDT_CTR register with WDT_CTR register
- service the watchdog
  – check current timer value in WDT_TIM register against programmed time window
  – write magic word to WDT_SRV register within valid time window

9.8.4 Enter Sleep/Deep Sleep & Resume Operation

The WDT counter clock can be configured to stop while in sleep or deep-sleep mode. No direct software interaction with the WDT is required in those modes and no watchdog time-out will fire if the WDT clock is configured to stop while CPU is sleeping.

- software configuration sequence for sleep/deep-sleep mode
  – configure WDT behavior with SCU register SLEEPCR or DSLEEPCR
- enter sleep/deep-sleep mode software sequence
  – select sleep or deep-sleep mode in CPU (for details please refer to Cortex-M4 documentation **[9]**)
  – enter selected mode (for details please refer to Cortex-M4 documentation **[9]**)
- wait for a wake-up event (no software interaction, CPU stopped)
- resume operation (CPU clock restarted automatically on an event)
- service the watchdog
  – check current timer value in WDT_TIM register against programmed time window
  – write magic word to WDT_SRV register within valid time window

9.8.5 Prewarning Alarm Handling

The WDT will fire prewarning alarm before requesting system reset while in pre-warning mode and not serviced within valid time window. The WDT status register indicating alarm must be cleared before the timer counter value crosses the upper bound for the second time after firing the alarm. After clearing of the alarm status regular watchdog servicing must be performed within valid time window.

- alarm event
  - exception routine (system trap or service request) clearing WDT_WDTSTAT register with WDT_WDTCLR register
- service the watchdog
  - check current timer value in WDT_TIM register against programmed time window
  - write magic word to WDT_SRV register within valid time window

9.9 Registers

Registers Overview

All these registers can be read in User Mode, but can only be written in Supervisor Mode.

The absolute register address is calculated by adding:

Module Base Address + Offset Address

Table 9-2 Registers Address Space

Module	Base Address	End Address	Note
WDT	5000 8000$_H$	5000 BFFF$_H$	Watchdog Timer Registers

Table 9-3 Register Overview

Short Name	Register Long Name	Offset Addr.	Access Mode		Description
			Read	Write	
WDT Kernel Registers					
ID	Module ID Register	00$_H$	U, PV	PV	Page 9-11
CTR	Control Register	04$_H$	U, PV	PV	Page 9-12
SRV	Service Register	08$_H$	BE	PV	Page 9-13
TIM	Timer Register	0C$_H$	U, PV	BE	Page 9-14
WLB	Window Lower Bound	10$_H$	U, PV	PV	Page 9-14
WUB	Window Upper Bound	14$_H$	U, PV	PV	Page 9-14
WDTSTS	Watchdog Status Register	18$_H$	U, PV	PV	Page 9-15
WDTCLR	Watchdog Status Clear Register	1C$_H$	U, PV	PV	Page 9-16

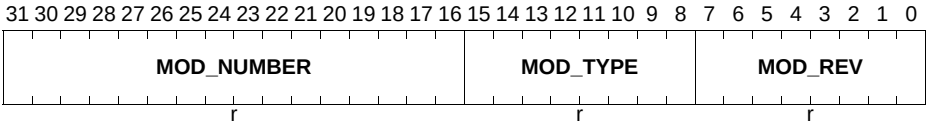
9.9.1 Registers Description

ID

The module ID register.

ID

WDT ID Register **(00_H)** **Reset Value: 00AD C0XX$_H$**



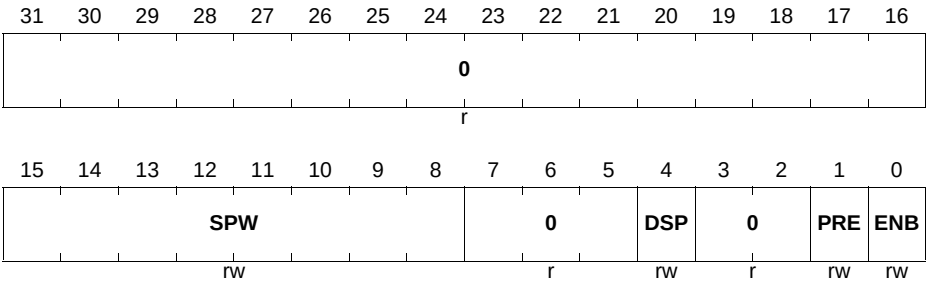
Field	Bits	Type	Description
MOD_REV	[7:0]	r	**Module Revision** Indicates the revision number of the implementation. This information depends on the design step.
MOD_TYPE	[15:8]	r	**Module Type** This internal marker is fixed to $C0_H$.
MOD_NUMBER	[31:16]	r	**Module Number** Indicates the module identification number

CTR

The operation mode control register.

CTR

WDT Control Register **(04_H)** **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
ENB	0	rw	**Enable** 0_B disables watchdog timer, 1_B enables watchdog timer

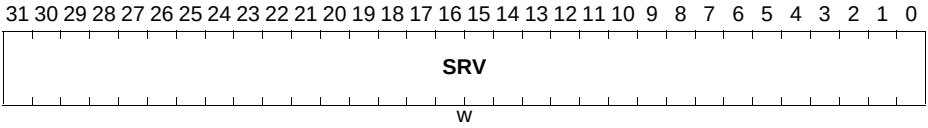
Field	Bits	Type	Description
PRE	1	rw	**Pre-warning** 0_B disables pre-warning 1_B enables pre-warning,
DSP	4	rw	**Debug Suspend** 0_B watchdog timer is stopped during halting mode debug, 1_B watchdog timer is not stopped during halting mode debug
SPW	[15:8]	rw	**Service Indication Pulse Width** Pulse width (SPW+1) of service indication in f_{WDT} cycles
0	[3:2], [7:5], [31:16]	r	**Reserved**

SRV

The WDT service register. Software must write a magic word while the timer value is within the valid window boundary. Writing the magic word while the timer value is within the window boundary will service the watchdog and result a reload of the timer with 0H.

Upon writing data different than the magic word within valid time window or writing even correct Magic Word but outside of the valid time window no servicing will be performed. Instead will request an immediate system reset request.

SRV
WDT Service Register **(08_H)** **Reset Value: 0000 0000$_H$**

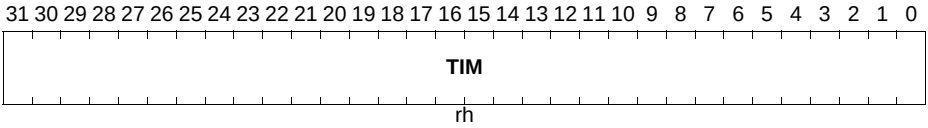


Field	Bits	Type	Description
SRV	[31:0]	w	**Service** Writing the magic word ABADCAFE$_H$ while the timer value is within the window boundary will service the watchdog.

TIM

The actual watchdog timer register count value. This register can be read by software in order to determine current position in the WDT time window.

TIM
WDT Timer Register (0C$_H$) Reset Value: 0000 0000$_H$

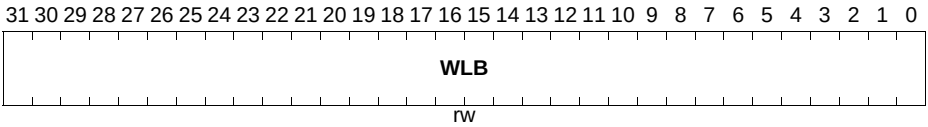


Field	Bits	Type	Description
TIM	[31:0]	rh	**Timer Value** Actual value of watchdog timer value.

WLB

The Window Lower Bound register defines the lower bound for servicing window. Servicing of the watchdog has only effect within the window boundary

WLB
WDT Window Lower Bound Register (10$_H$) Reset Value: 0000 0000$_H$



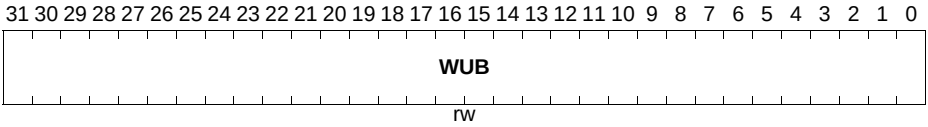
Field	Bits	Type	Description
WLB	[31:0]	rw	**Window Lower Bound** Lower bound for servicing window. Setting the lower bound to 0$_H$ disables the window mechanism.

WUB

The Window Upper Bound register defines the upper bound for servicing window. Servicing of the watchdog has only effect within the window boundary.

WUB

WDT Window Upper Bound Register (14_H) **Reset Value: FFFF FFFF$_H$**



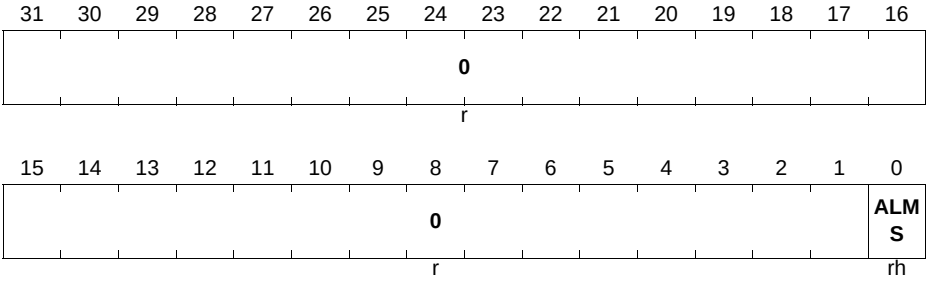
Field	Bits	Type	Description
WUB	[31:0]	rw	**Window Upper Bound** Upper Bound for servicing window. The WDT triggers an reset request when the timer is crossing the upper bound value without pre-warning enabled. With pre-warning enabled the first crossing triggers a watchdog alarm and the second crossing triggers a system reset.

WDTSTS

The status register contains sticky bit indicating occurrence of alarm condition.

WDTSTS

WDT Status Register **(0018_H)** **Reset Value: 00000000$_H$**



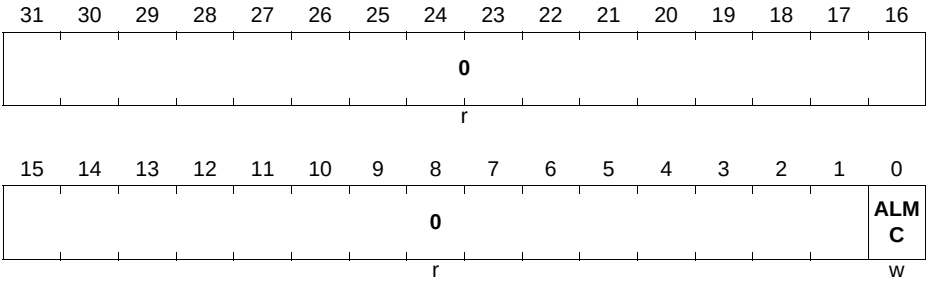
Field	Bits	Type	Description
ALMS	0	rh	**Pre-warning Alarm** 1_B pre-warning alarm occurred, 0_B no pre-warning alarm occurred

Field	Bits	Type	Description
0	[31:1]	r	**Reserved**

WDTCLR

The status register contains sticky bitfield indicating occurrence of alarm condition.

WDTCLR
WDT Clear Register **(001C$_H$)** **Reset Value: 00000000$_H$**



Field	Bits	Type	Description
ALMC	0	w	**Pre-warning Alarm** 1_B clears pre-warning alarm 0_B no-action
0	[31:1]	r	**Reserved**

9.10 Interconnects

Table 9-4 Pin Table

Input/Output	I/O	Connected To	Description
Clock and Reset Signals			
f_{WDT}	I	SCU.CCU	timer clock
Timer Signals			
wdt_service	O	SCU.HCU	service indication to external watchdog

Table 9-4 Pin Table (cont'd)

Input/Output	I/O	Connected To	Description
HALTED	I	CPU	In halting mode debug. HALTED remains asserted while the core is in debug.
Service Request Connectivity			
wdt_alarm	O	SCU.GCU	pre-warning alarm
wdt_rst_req	O	SCU.RCU	reset request

10 Real Time Clock (RTC)

Real-time clock (RTC) is a clock that keeps track of the current time. RTCs are present in almost any electronic device which needs to keep accurate time in a digital format for clock displays and real-time actions.

10.1 Overview

The RTC module tracks time with separate registers for hours, minutes, and seconds. The calendar registers track date, day of the week, month and year with automatic leap year correction.

The RTC is capable of running from an alternate source of power, so it can continue to keep time while the primary source of power is off or unavailable. The timer remains operational when the core domain is in power-down. The kernel part of the RTC keeps running as long as the hibernate domain is powered with an alternate supply source. The alternate source can be for example a lithium battery or a supercapacitor.

10.1.1 Features

The features of the Real Time Clock (RTC) module are:

- Precise real time keeping with
  - 32.768 kHz external crystal clock
  - 32.768 kHz high precision internal clock
- Periodic time-based interrupt
- Programmable alarm interrupt on time match
- Supports wake-up mechanism from hibernate state

Table 10-1 Application Features

Feature	Purpose/Application
Precise real-time keeping	Reduced need for time adjustments
Periodic time-based interrupt	Scheduling of operations performed on precisely defined intervals
Programmable alarm interrupt on time match	Scheduling of operations performed on precisely defined times
Supports wake-up mechanism from hibernate state	Autonomous wake up from hibernate for system state control and maintaintenance routine operations

10.1.2 Block Diagram

The RTC block diagram is shown in **Figure 10-1**.

The main building blocks of the RTC is Time Counter implementing real time counter and RTC registers containing multi-field registers for the time counter and alarm programming register. Dedicated fields represent values for elapsing second, minutes, hours, days, days of week, months and years.

The kernel of the RTC module is instantiated in the hibernate domain.

The RTC registers are instantiated in hibernate domain and mirrored in SCU. Access to the RTC registers is performed via register mirror updated over serial interface.

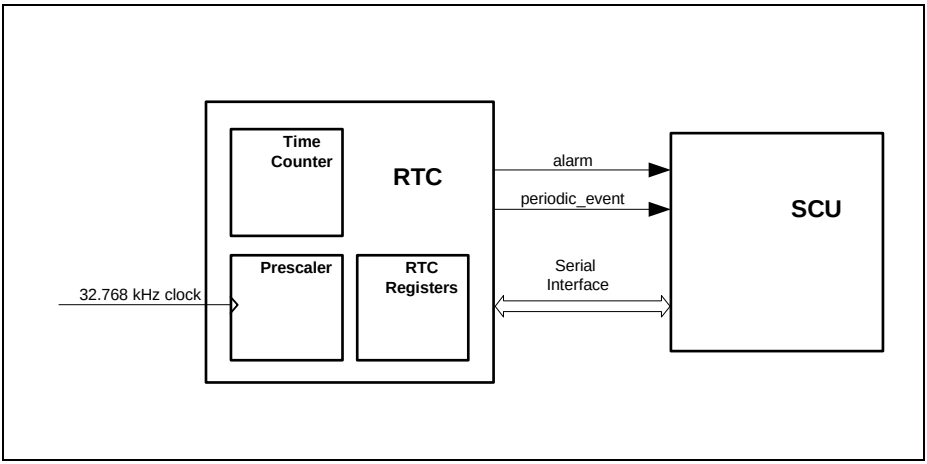


Figure 10-1 Real-Time Clock Block Diagram Structure

10.2 RTC Operation

The RTC timer counts seconds, minutes, hours, days of month, days of week, months and years each in a separate field (see **Figure 10-2**). Individual bit fields of the RTC counter can be programmed and read with software over serial interface via mirror registers in SCU module. For details of the serial communication please refer to SCU chapter.

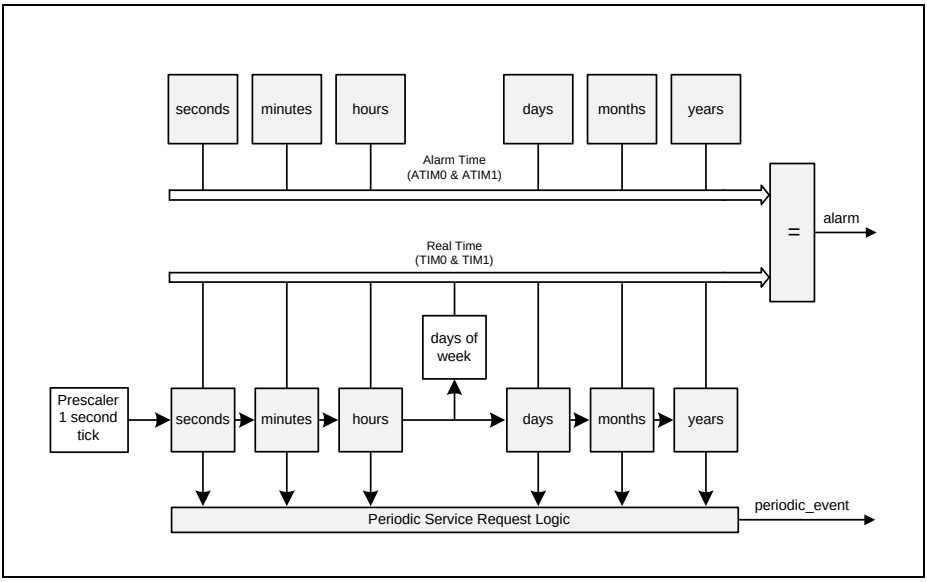


Figure 10-2 Block Diagram of RTC Time Counter

Occurrence of an internal timer event is stored in the service request raw status register **RAWSTAT**. The values of the status register **RAWSTAT** drive the outgoing service request lines alarm and periodic_event.

10.3 Register Access Operations

The RTC module is a part of SCU from programmming model perepcetive and shares register address space for configuration with other sub-modulesof SCU. RTC registers are instantiated in hibernate domain are mirrored in SCU. The registers get updated in both clock domains over serial interface running at 32kHz clock rate.

Any update of the registers is performed with some delay required for data to propagate to and from the mirror registers over serial interface. Accesses to the RTC registers in core domain must not block the bus interface of SCU module. For details of the register mirror and serial communication handling please refer to SCU chapter.

A write to writable registers is only allowed, when the access is in privileged mode. A write access in user mode results in a bus error response.

For consistent write to the timer registers **TIM0** and **TIM1**, the register **TIM0** has to be written before the register **TIM1**. Transfer of the new values from the register mirror starts only after both registers have been written.

After wake-up from hibernate state the content of the mirror registers **TIM0** and **TIM1** is undefined until the first update of the corresponding RTC timers occurs and is propagated to the registers.

10.4 Service Request Processing

The RTC generates service requests upon:

- periodic timer events
- configured alarm condition

The service requests can be processed in the core domain as regular service requests.

10.4.1 Periodic Service Request

The periodic timer service request is raised whenever a non-masked field of the timer counter gets updated. The Periodic Service requests can be enabled/disbled with the **MSKSR** register.

10.4.2 Timer Alarm Service Request

The alarm interrupt is triggered when **TIM0** and **TIM1** bit fields values match all corresponding bit fields values of **ATIM0**, **ATIM1** registers. The Timer Alarm Service requests can be enabled/disbled with the **MSKSR** register.

10.5 Wake-up From Hibernation Trigger

The RTC generates wake-up triggers upon:

- periodic timer events
- configured alarm condition

The timer events can be processed in the hibernate domain as wake-up triggers from hibernate mode, in the HCU module of hibernate domain (for more details please refer to hibernate control description in SCU chapter).

10.5.1 Periodic Wake-up Trigger Generation

The periodic timer wake-up trigger gets generated whenever a non-masked field of the timer counter gets updated. The Periodic Wake-up Trigger generation can be enabled/disbled with the **CTR** register.

10.5.2 Timer Alarm Wake-up Trigger Generation

The alarm wakeu-up gets trigger gets generated when **TIM0** and **TIM1** bit fields values match all corresponding bit fields values of **ATIM0**, **ATIM1** registers. Timer Alarm Wake-up Trigger generation can be enabled/disbled with the **CTR** register.

10.6 Debug behavior

The RTC clock does not implement dedicated debug mechanisms.

10.7 Power, Reset and Clock

RTC is instantiated entirely in hibernate domain and remains powered up when hibernate domain is powered up. Supply voltage is passed via power switch either from VDDP or VBAT pin as specified in the SCU chapter.

The RTC module remains in reset state along with entire hibernate domain after initial power up of hibernate domain until reset released with software.

The RTC timer is running from ether internal or external 32.768 kHz clock selectable with HDCR control register of SCU/HCU module. The prescaler setting of $7FFF_H$ results in an once per second update of the RTC timer.

10.8 Initialization and Control Sequence

Programming model of the RTC module assumes several scenarios where different control sequences apply.

Note: Some of the scenarios described in this chapter require operations on system level the that are not in the scope of the RTC module description, therefore for detailed information please refer to relevant chapters of this document.

10.8.1 Initialization & Start of Operation

Complete RTC module initialization is required upon hibernate domain reset. The hibernate domain needs to be enabled before any programming of RTC registers takes place. Accesses to RTC registers are performed via dedicated mirror registers (for more details please refer to SCU chapter)

- enable hibernate domain (if not disabled)
  - write one to SCU_PWRSET.HIB
- release reset of hibernate domain reset (if asserted)
  - write one to SCU_RSTCLR.HIBRS
- enable RTC module to start counting time
  - write one to RTC_CTR.ENB
- program RTC_TIM0 and RTC_TIM1 registers with current time
  - check SCU_MIRRSTS to ensure that no transfer over serial interface is pending to the RTC_TIM0 register
  - write a new value to the RTC_TIM0 register
  - check SCU_MIRRSTS to ensure that no transfer over serial interface is pending to the RTC_TIM1 register
  - write a new value to the RTC_TIM1 register

10.8.2 Re-configuration & Re-start of Operation

Reset and re-initialization of the RTC module may be required without complee power up sequence of the hibernate domain.

- apply and release reset of hibernate domain reset
  - write one to SCU_RSTSET.HIBRS
  - write one to SCU_RSTCLR.HIBRS
- enable RTC module to start counting time
  - write one to RTC_CTR.ENB
- program RTC_TIM0 and RTC_TIM1 registers with current time
  - check SCU_MIRRSTS to ensure that no transfer over serial interface is pending to the RTC_TIM0 register
  - write a new value to the RTC_TIM1 register
  - check SCU_MIRRSTS to ensure that no transfer over serial interface is pending to the RTC_TIM1 register
  - write a new value to the RTC_TIM1 register

10.8.3 Configure and Enable Periodic Event

The RTC periodic event configuration require programming in order to enable intrrupt request generation out upon a change of value in the corresponding bit fields.

- enable service request for periodic timer events in RTC module
  - check SCU_MIRRSTS to ensure that no transfer over serial interface is pending to the RTC_MSKSR register
  - set MAI bit field of RTC_MSKSR register in order enable individual periodic timer events
- enable service request for periodic timer events in RTC module
  - set PI bit field of SCU_SRMSK register in order enable generation of interrupts upon periodic timer events

10.8.4 Configure and Enable Timer Event

The RTC periodic event configuration require programming in order to enable intrrupt request generation out upon compare match of values in the corresponding bit fields of TIM0 and TIM1 against ATIM0 and ATIM1 respectively.

- program compare values in individual bit fields of ATIM0 and ATIM1 in RTC module
  - check SCU_MIRRSTS to ensure that no transfer over serial interface is pending to the RTC_ATIM0 register
  - write to RTC_ATIM0 register bit fields
  - check SCU_MIRRSTS to ensure that no transfer over serial interface is pending to the RTC_ATIM1 register
  - write to RTC_ATIM1 register bit fields
- enable service request for timer alarm events in RTC module

  - check SCU_MIRRSTS to ensure that no transfer over serial interface is pending to the RTC_CTR register
  - set TAE bit field of RTC_CTR register in order enable individual periodic timer events
- enable service request for timer alarm events in RTC module
  - write one to AI bit field of SCU_SRMSK register in order enable generation of interrupts upon periodic timer events

10.9 Registers

Registers Overview

The absolute register address is calculated by adding:

Module Base Address + Offset Address

Table 10-2 Registers Address Space

Module	Base Address	End Address	Note
RTC	5000 4A00$_H$	5000 4BFF$_H$	Accessible via Mirror Registers

Table 10-3 Register Overview

Short Name	Register Long Name	Offset Addr.	Access Mode		Description
			Read	Write	
RTC Kernel Registers					
ID	ID Register	0000$_H$	U, PV	BE	Page 10-8
CTR	Control Register	0004$_H$	U, PV	PV	Page 10-9
RAWSTAT	Raw Service Request Register	0008$_H$	U, PV	BE	Page 10-11
STSSR	Status Service Request Register	000C$_H$	U, PV	BE	Page 10-12
MSKSR	Mask Service Request Register	0010$_H$	U, PV	PV	Page 10-13
CLRSR	Clear Service Request Register	0014$_H$	BE	PV	Page 10-14
ATIM0	Alarm Time Register 0	0018$_H$	U,PV	PV	Page 10-15
ATIM1	Alarm Time Register 1	001C$_H$	U,PV	PV	Page 10-16
TIM0	Time Register 0	0020$_H$	U, PV	PV	Page 10-17
TIM1	Time Register 1	0024$_H$	U, PV	PV	Page 10-19

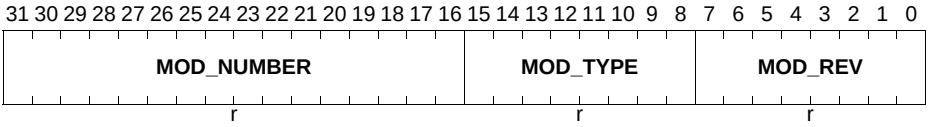
10.9.1 Registers Description

ID

Read-only ID register of the RTC module containing unique identification code of the RTC module.

ID

RTC ID Register **(00_H)** **Reset Value: 00A3 C0XX$_H$**



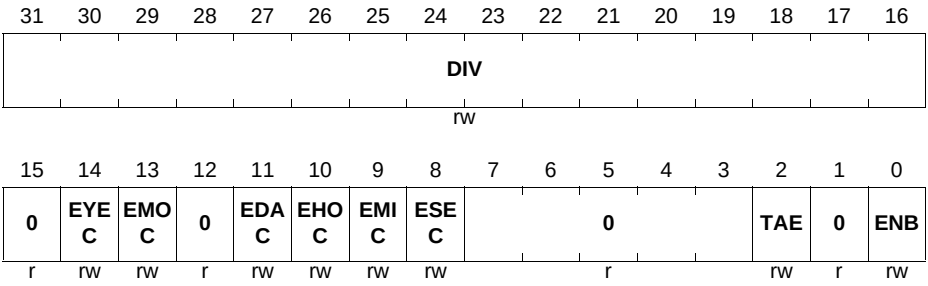
Field	Bits	Type	Description
MOD_REV	[7:0]	r	**Module Revision** Indicates the revision number of the implementation. This information depends on the design step.
MOD_TYPE	[15:8]	r	**Module Type** This internal marker is fixed to $C0_H$.
MOD_NUMBER	[31:16]	r	**Module Number** Indicates the module identification number

CTR

RTC Control Register providing control means of the operation mode of the module.

CTR

RTC Control Register **(04_H)** **Reset Value: 7FFF 0000$_H$**



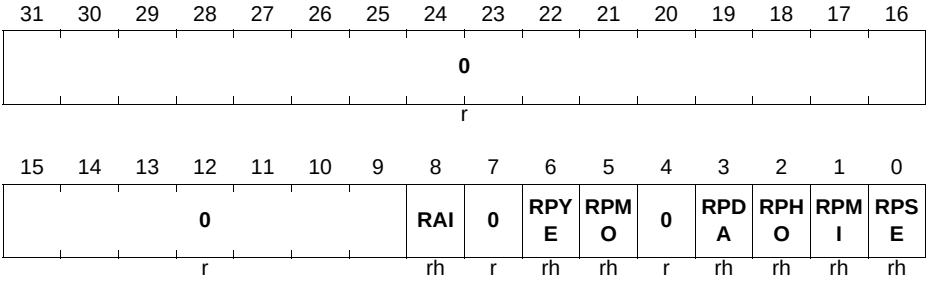
Field	Bits	Type	Description
ENB	0	rw	**RTC Module Enable** 0_B disables RTC module 1_B enables RTC module

Field	Bits	Type	Description
TAE	2	rw	**Timer Alarm Enable for Hibernation Wake-up** 0_B disable timer alarm 1_B enable timer alarm
ESEC	8	rw	**Enable Seconds Comparison for Hibernation Wake-up** 0_B disabled 1_B enabled
EMIC	9	rw	**Enable Minutes Comparison for Hibernation Wake-up** 0_B disabled 1_B enabled
EHOC	10	rw	**Enable Hours Comparison for Hibernation Wake-up** 0_B disabled 1_B enabled
EDAC	11	rw	**Enable Days Comparison for Hibernation Wake-up** 0_B disabled 1_B enabled
EMOC	13	rw	**Enable Months Comparison for Hibernation Wake-up** 0_B disabled 1_B enabled
EYEC	14	rw	**Enable Years Comparison for Hibernation Wake-up** 0_B disabled 1_B enabled
DIV	[31:16]	rw	**RTC Clock Divider Value** reload value of RTC prescaler. Clock is divided by DIV+1. $7FFF_H$ is default value for RTC mode with 32.768 kHz crystal or internal clock
0	1,[7:3], 12,15	r	**Reserved** Read as 0; should be written with 0

RAWSTAT

RTC Raw Service Request Register contains raw status info i.e. before status mask takes effect on generation of service requests. This register serves debug purpose but can be also used for polling of the status without generating serice requests.

RAWSTAT
RTC Raw Service Request Register (08$_H$) **Reset Value: 0000 0000$_H$**

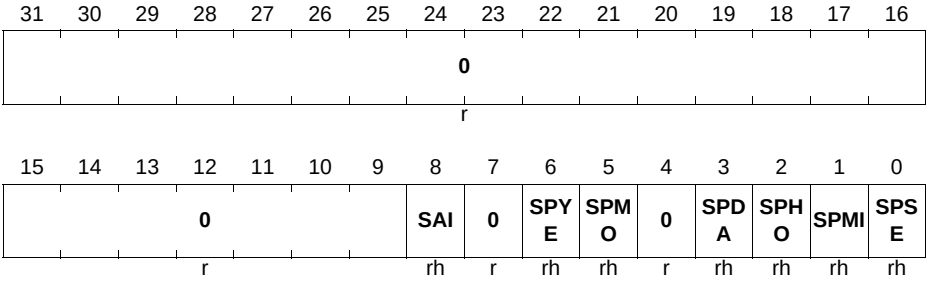


Field	Bits	Type	Description
RPSE	0	rh	**Raw Periodic Seconds Service Request** Set whenever seconds count increments
RPMI	1	rh	**Raw Periodic Minutes Service Request** Set whenever minutes count increments
RPHO	2	rh	**Raw Periodic Hours Service Request** Set whenever hours count increments
RPDA	3	rh	**Raw Periodic Days Service Request** Set whenever days count increments
RPMO	5	rh	**Raw Periodic Months Service Request** Set whenever months count increments
RPYE	6	rh	**Raw Periodic Years Service Request** Set whenever years count increments
RAI	8	rh	**Raw Alarm Service Request** Set whenever count value matches compare value
0	4, 7, [31:9]	r	**Reserved**

STSSR

RTC Service Request Status Register contains status info reflecting status mask effect on generation of service requests. This register needs to be accessed by software in order to determine the actual cause of an event.

STSSR
RTC Service Request Status Register ($0C_H$) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
SPSE	0	rh	**Periodic Seconds Service Request Status after Masking**
SPMI	1	rh	**Periodic Minutes Service Request Status after Masking**
SPHO	2	rh	**Periodic Hours Service Request Status after Masking**
SPDA	3	rh	**Periodic Days Service Request Status after Masking**
SPMO	5	rh	**Periodic Months Service Request Status after Masking**
SPYE	6	rh	**Periodic Years Service Request Status after Masking**
SAI	8	rh	**Alarm Service Request Status after Masking**
0	4, 7, [31:9]	r	**reserved**

MSKSR

RTC Service Request Mask Register contains masking value for generation control of service requests or interrupts.

MSKSR

RTC Service Request Mask Register (10$_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0							MAI	0	MPYE	MPMO	0	MPDA	MPHO	MPMI	MPSE
r							rw	r	rw	rw	r	rw	rw	rw	rw

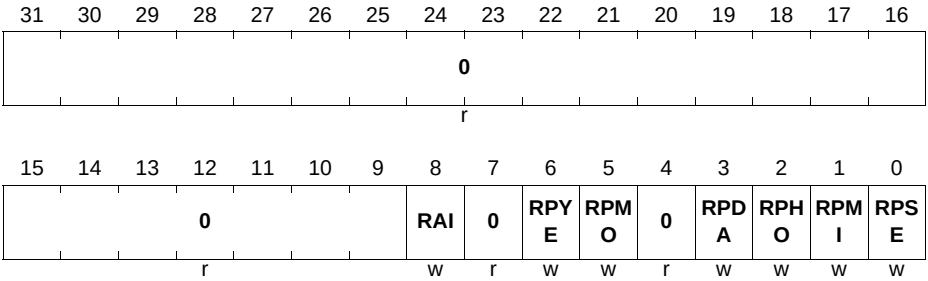
Field	Bits	Type	Description
MPSE	0	rw	**Periodic Seconds Interrupt Mask** 0_B disable 1_B enable
MPMI	1	rw	**Periodic Minutes Interrupt Mask** 0_B disable 1_B enable
MPHO	2	rw	**Periodic Hours Interrupt Mask** 0_B disable 1_B enable
MPDA	3	rw	**Periodic Days Interrupt Mask** 0_B disable 1_B enable
MPMO	5	rw	**Periodic Months Interrupt Mask** 0_B disable 1_B enable
MPYE	6	rw	**Periodic Years Interrupt Mask** 0_B disable 1_B enable
MAI	8	rw	**Alarm Interrupt Mask** 0_B disable 1_B enable

Field	Bits	Type	Description
0	4, 7, [31:9]	r	**Reserved**

CLRSR

RTC Clear Service Request Register serves purpose of clearing sticky bits of **RAWSTAT** and **STSSR** registers. Write one to a bit in order to clear it is set. Writing zero has no effect on the set nor reset bits.

CLRSR
RTC Clear Service Request Register (14_H) **Reset Value: 0000 0000$_H$**



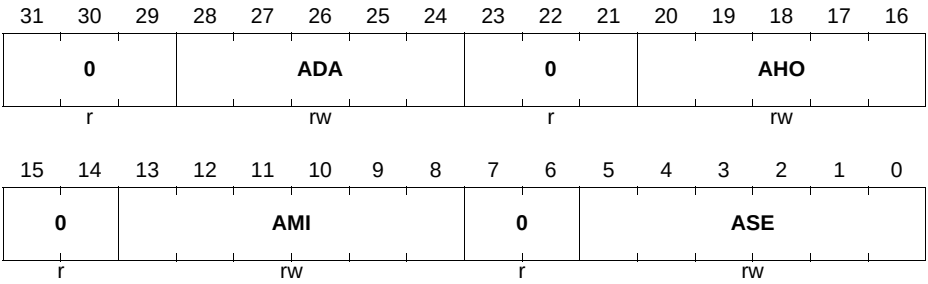
Field	Bits	Type	Description
RPSE	0	w	**Periodic Seconds Interrupt Clear** 0_B no effect 1_B clear status bit
RPMI	1	w	**Periodic Minutes Interrupt Clear** 0_B no effect 1_B clear status bit
RPHO	2	w	**Periodic Hours Interrupt Clear** 0_B no effect 1_B clear status bit
RPDA	3	w	**Periodic Days Interrupt Clear** 0_B no effect 1_B clear status bit
RPMO	5	w	**Periodic Months Interrupt Clear** 0_B no effect 1_B clear status bit

Field	Bits	Type	Description
RPYE	6	w	**Periodic Years Interrupt Clear** 0_B no effect 1_B clear status bit
RAI	8	w	**Alarm Interrupt Clear** 0_B no effect 1_B clear status bit
0	4, 7, [31:9]	r	**Reserved**

ATIM0

RTC Alarm Time Register 0 serves purpose of programming single alarm time at a desired point of time reflecting comparison configuration in the **CTR** for individual fields against **TIM0** register. The register contains portion of bit fields for seconds, minutes, hours and days. Upon attempts to write an invalid value to a bit field e.g. exceeding maximum value default value gets programmed as described for each individual bit fields.

ATIM0
RTC Alarm Time Register 0 (18_H) Reset Value: 0000 0000_H



Field	Bits	Type	Description
ASE	[5:0]	rw	**Alarm Seconds Compare Value** Match of seconds timer count to this value triggers alarm seconds interrupt. Setting value equal or above $3C_H$ results in setting the field value to 0_H

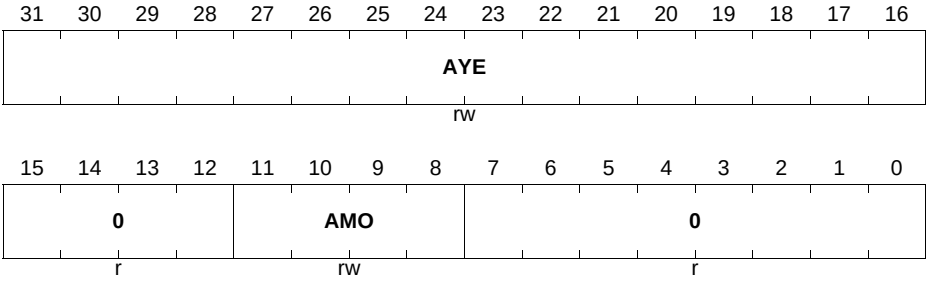
Field	Bits	Type	Description
AMI	[13:8]	rw	**Alarm Minutes Compare Value** Match of minutes timer count to this value triggers alarm minutes interrupt. Setting value equal or above $3C_H$ results in setting the field value to 0_H
AHO	[20:16]	rw	**Alarm Hours Compare Value** Match of hours timer count to this value triggers alarm hours interrupt. Setting value equal or above 18_H results in setting the field value to 0_H
ADA	[28:24]	rw	**Alarm Days Compare Value** Match of days timer count to this value triggers alarm days interrupt. Setting valueequal above $1F_H$ results in setting the field value to 0_H
0	[7:6], [15:14], [23:21], [31:29]	r	**Reserved**

ATIM1

RTC Alarm Time Register 1 serves purpose of programming single alarm time at a desired point of time reflecting comparison configuration in the **CTR** for individual fields against **TIM1** register. The ATM1 register contains portion of bit fields for days of week, months and years. Upon attempts to write an invalid value to a bit field e.g. exceeding maximum value default value gets programmed as described for each individual bit fields.

ATIM1

RTC Alarm Time Register 1 **($1C_H$)** **Reset Value: 0000 0000$_H$**

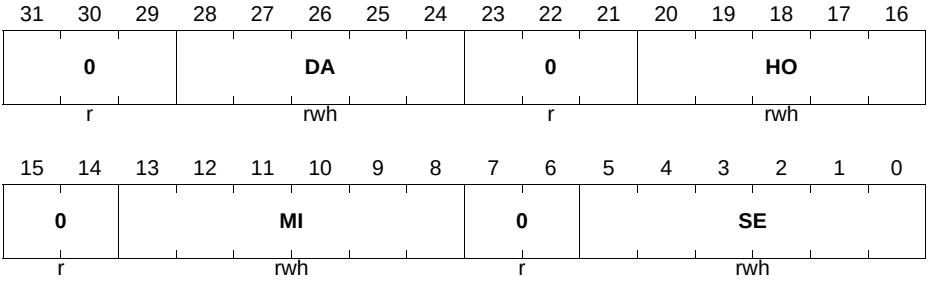


Field	Bits	Type	Description
AMO	[11:8]	rw	**Alarm Month Compare Value** Match of months timer count to this value triggers alarm month interrupt. Setting value equal or above the number of days of the actual month count results in setting the field value to 0_H
AYE	[31:16]	rw	**Alarm Year Compare Value** Match of years timer count to this value triggers alarm years interrupt.
0	[7:0], [15:12]	r	**Reserved**

TIM0

RTC Time Register 0 contains current time value for seconds, minutes, hours and days. The bit fields get updated in intervals corresponding with their meaning accordingly. The register needs to be programmed to reflect actual time after initial power up and will continue counting time also while in hibernate mode. Upon attempts to write an invalid value to a bit bield e.g. exceeding maximum value a default value gets programmed as described for each individual bit fields.

TIM0
RTC Time Register 0 **(20_H)** **Reset Value: 0000 0000$_H$**



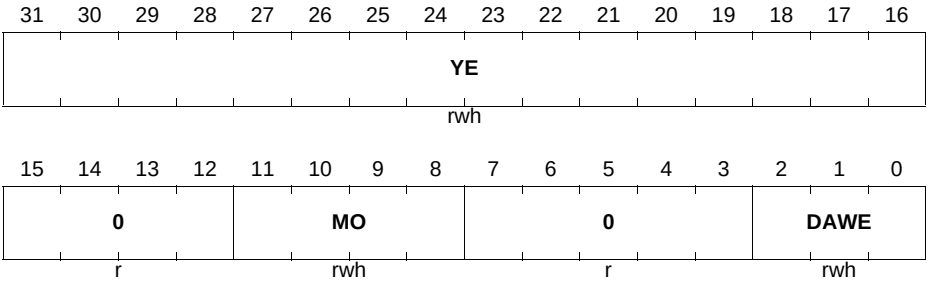
Field	Bits	Type	Description
SE	[5:0]	rwh	**Seconds Time Value** Setting value equal or above $3C_H$ results in setting the field value to 0_H. Value can only be written, when RTC is disabled. After wake-up from hibernate, value is undefined until first update of RTC.
MI	[13:8]	rwh	**Minutes Time Value** Setting value equal or above $3C_H$ results in setting the field value to 0_H. Value can only be written, when RTC is disabled. After wake-up from hibernate, value is undefined until first update of RTC.
HO	[20:16]	rwh	**Hours Time Value** Setting value equal or above 18_H results in setting the field value to 0_H Value can only be written, when RTC is disabled. After wake-up from hibernate, value is undefined until first update of RTC.

Field	Bits	Type	Description
DA	[28:24]	rwh	**Days Time Value** Setting value equal or above the number of days of the actual month count results in setting the field value to 0_H Value can only be written, when RTC is disabled. After wake-up from hibernate, value is undefined until first update of RTC. Days counter starts with value 0 for the first day of month.
0	[7:6], [15:14], [23:21], [31:29]	r	**Reserved**

TIM1

RTC Time Register 1 contains current time value for days of week, months and years. The bit fields get updated in intervals corresponding with their meaning accordingly. The register needs to be programmed to reflect actual time after initial power up and will continue counting time also while in hibernate mode. Upon attempts to write an invalid value to a bit bield e.g. exceeding maximum value a default value gets programmed as described for each individual bit fields.

TIM1
RTC Time Register 1 **(24_H)** **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
DAWE	[2:0]	rwh	**Days of Week Time Value** Setting value above 6_H results in setting the field value to 0_H Value can only be written, when RTC is disabled. After wake-up from hibernate, value is undefined until first update of RTC. Days counter starts with value 0 for the first day of week.
MO	[11:8]	rwh	**Month Time Value** Setting value equal or above C_H results in setting the field value to 0_H. Value can only be written, when RTC is disabled. After wake-up from hibernate, value is undefined until first update of RTC. Months counter starts with value 0 for the first month of year.
YE	[31:16]	rwh	**Year Time Value** Value can only be written, when RTC is disabled. After wake-up from hibernate, value is undefined until first update of RTC.
0	[7:3], [15:12]	r	**Reserved**

10.10 Interconnects

Table 10-4 Pin Connections

Input/Output	I/O	Connected To	Description
Clock Signals			
f_{RTC}	I	SCU.HCU	32.768 kHz clock selected in hibernate domain
Service Request Connectivity			
periodic_event	O	SCU.GCU	Timer periodic service request
alarm	O	SCU.GCU	Alarm service request

11 System Control Unit (SCU)

The SCU is the SoC power, reset and a clock manager with additional responsibility of providing system stability protection and other auxiliary functions.

11.1 Overview

The functionality of the SCU described in this chapter is organized in the following sub-chapters, representing different aspects of system control:

- Miscellaneous control functions, **Chapter 11.2**
- Power Control, **Chapter 11.3**
- Hibernate Control, **Chapter 11.4**
- Reset Control, **Chapter 11.5**
- Clock Control, **Chapter 11.6**

11.1.1 Features

The following features are provided for monitoring and controlling the system:

- General Control
  - Boot Mode Detection
  - Memory Parity Protection
  - Trap Generation
  - Die Temperature Measurement
  - Retention Memory Support
- Power Control
  - Power Sequencing
  - EVR Control
  - Supply Watchdog
  - Voltage Monitoring
  - Power Validation
  - Power State Indication
  - Flash Power Control
- Hibernate Control
  - Control of on-chip core supply generation
  - Hibernate Mode Control
  - Wake-up from Hibernate Mode
  - Hibernate Domain Control
- Reset Control
  - Reset assertion on various reset request sources
  - System Reset Generation
  - Inspection of Reset Sources After Reset
  - Selective Module reset

- Clock Control
  - Individual peripheral clock gating
  - Input clock selection
  - Clock Generation
  - Clock Distribution
  - Clock Supervision
  - Power Management
  - RTC Clock

11.1.2 Block Diagram

The block diagram shown in **Figure 11-1** reflects logical organization of the System Control Unit.

- Power Control Unit (PCU)
- Hibernate Control Unit (HCU)
- Reset Control Unit (RCU)
- General Control Unit (GCU)
- Clock Control Unit (CCU)

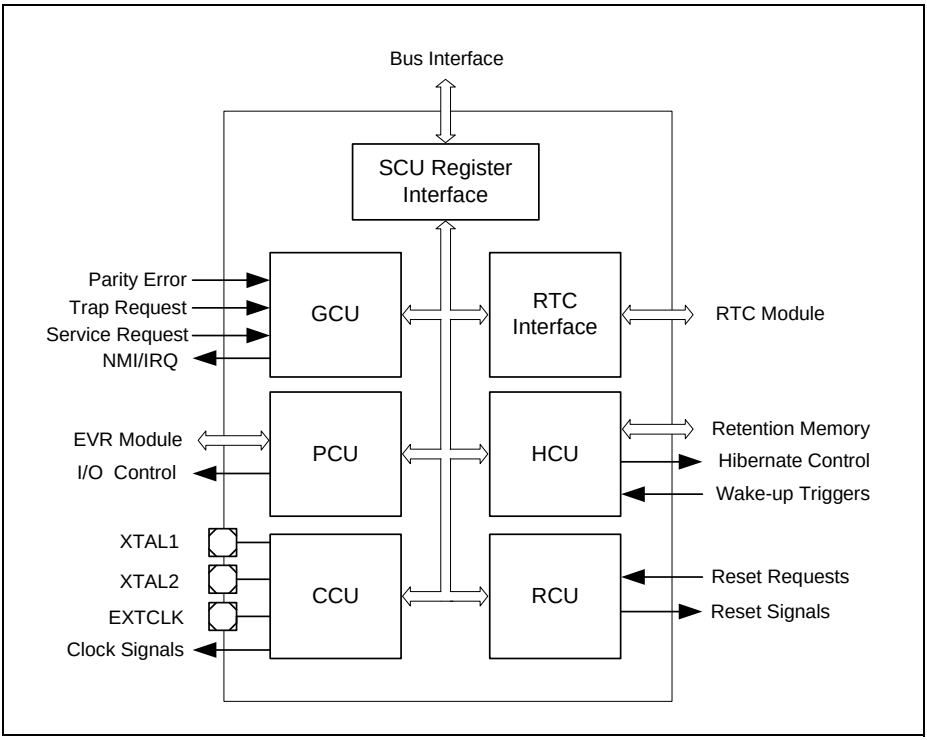


Figure 11-1 SCU Block Diagram

Interface of General Control Unit

The General Control Unit GCU has a memory fault interface to the memory validation logic of each on-chip SRAM and the Flash to receive memory fault events, as parity errors. NMI request are routed to the unit to be gated and combined. The GCU provides the start up protection to all units, which require an additional level of protection.

Interface of Power Control Unit

The Power Control Unit PCU has an interface to the Embedded Voltage Regulator (EVR) and an interface to the PORTS module. The PCU related signals are described in more detail in **Chapter 11.3**.

Interface of Reset Control Unit

The Reset Control Unit RCU has an interface to the Embedded Voltage Regulator (EVR). The RCU receives from the EVR the power-on reset and the reset information for

each power related reset. Reset requests are coming to the unit from the watchdog, the CPU and the test control unit. The RCU is providing the reset signals to all other units of the chip in the Core power domain. The RCU related signals are described in more detail in **Chapter 11.5**.

Interface of Clock Control Unit

The Clock Control Unit (CCU) receives the external clock source via the crystal pins XTAL1 and XTAL2. As further clock source the CCU receives the standby clock f_{STDBY} from the Hibernate domain. The CCU drives an external clock output, where internal clocks can be routed out. The CCU provides the clock signals to all other units of the chip.

Interface of Hibernate Power Domain

The interface to the Hibernate domain provides mirror registers updated via a shared serial interface to the Retention Memory, RTC module registers and Hibernate domain control registers. Update of the mirror registers over the serial is controlled using **MIRRSTS**, **RMDATA** and **RMACR** registers. End of update can also trigger service requests via **SRSTAT** register. Refresh of the Hibernate domain registers in the register mirror is performed continuously, in order to instantly reflect any register state change on both sides. The serial interface is inactivated while in hibernate mode in order to reduce current consumption.

Interface of Retention Memory

Access to the Retention Memory is served over shared serial interface identical to the one used to access Hibernate domain registers, for detail please refer to **“Interface of Hibernate Power Domain” on Page 11-4**.

Interface of RTC

Access to the RTC module is served over shared serial interface identical to the one used to access Hibernate domain registers, for detail please refer to **“Interface of Hibernate Power Domain” on Page 11-4**. The RTC module functionality is described in separate RTC chapter.

11.2 Miscellaneous Control Functions

System Control implements system management functions accessible via GCU registers. General system control including various auxiliary function is performed in General Control Unit (GCU).

11.2.1 Startup Software Support

Externally driven boot mode pins determine the boot mode after a power on reset. It also possible for applications to decide the boot mode. Ability to determine boot mode values on boot mode pins and user application desired boot modes is provided by SCU.

11.2.2 Service Requests

Service request events listed in **Table 11-1** can result in assertion of a regular interrupt or an NMI. Please refer to **SRMSK** and **NMIREQEN** register description.

The interrupt structure is shown in **Figure 11-2**. The interrupt request or the corresponding interrupt set bit (in register **SRSET**) can trigger the interrupt generation at the selected interrupt node x. The service request pulse is generated independently from the interrupt flag in register **SRSTAT**. The interrupt flag can be cleared by software by writing to the corresponding bit in register **SRCLR**. In addition several service requests can be promoted to NMI trigger level using **NMIREQEN** register

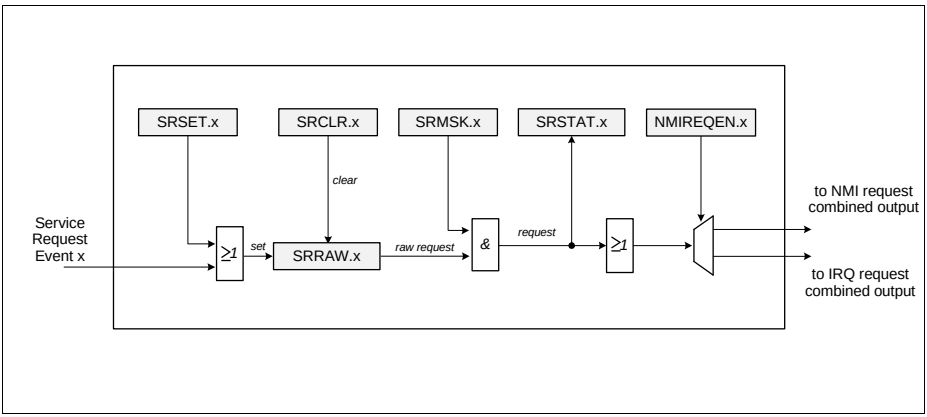


Figure 11-2 Service Request Subsystem

The service request flag in register **SRSTAT** can be cleared by software by writing to the corresponding bit in register **SRCLR**. All service requests are combined to one common line and connected to a regular interrupt node or NMI node of NVIC.

The service requests have a sticky flag in register **SRRAW**.

Note: When servicing an SCU service request, make sure that all related request flags are cleared after the identified request has been handled.

11.2.2.1 Service Request Sources

The SCU supports service request sources listed in **Table 11-2** and reflected in the **SRSTAT**, **SRRAW**, **SRMSK**, **SRCLR** and **SRSET** registers.

Table 11-1 Service Requests

Service Request Name	Service Request Short Name
WDT pre-warning	PRWARN
RTC Periodic Event	PI
RTC Alarm	AI
DLR Request Overrun	DLROVR
HDCLR Mirror Register Updated	HDCLR
HDSET Mirror Register Updated	HDSET
HDCR Mirror Register Updated	HDCR
OSCSICTRL Mirror Register Updated	OSCSICTRL
OSCULCTRL Mirror Register Updated	OSCULCTRL
RTC CTR Mirror Register Updated	RTC_CTR
RTC ATIM0 Mirror Register Updated	RTC_ATIM0
RTC ATIM1 Mirror Register Updated	RTC_ATIM1
RTC TIM0 Mirror Register Updated	RTC_TIM0
RTC TIM1 Mirror Register Updated	RTC_TIM1
Retention Memory Mirror Register Updated	RMX

11.2.3 Memory Parity Protection

For supervising the content of the on-chip memories, the following mechanism is provided:

All on-chip SRAMs provide protection of integrity via parity checking. The parity logic generates additional parity bits which are stored along with each data word at a write operation. A read operation implies checking of the previous stored parity information.

An occurrence of a parity error is observable at the memory error raw status register. It is configurable whether a memory error should trigger an NMI or System Reset.

11.2.3.1 Parity Error Handling

The on-chip RAM modules check parity information during read accesses and in case of an error a signal can be generated if enabled with **PEEN** register. Two modes of parity error signalling are implemented:

- bus error
- parity error trap (NMI)

The bus error generation applies to memories that can be accessed directly from the bus system level. Apart from that, all memories, including those that are not accessible

directly and are internal to peripherals are capable of generating system traps resulting in NMI. Parity trap requests get enabled with **PETE** register implementing individual control for each memory. Parity error signalling with trap generation is not recommended to be used for memories capable of bus error generation and therefore should be disabled.

Parity error trap generation mechanism can be also used to generate System Reset if enabled with **PERSTEN** register in conjunction with the **PETE** register configuration. For more details of the parity error generation scheme please refer to **Figure 11-3**.

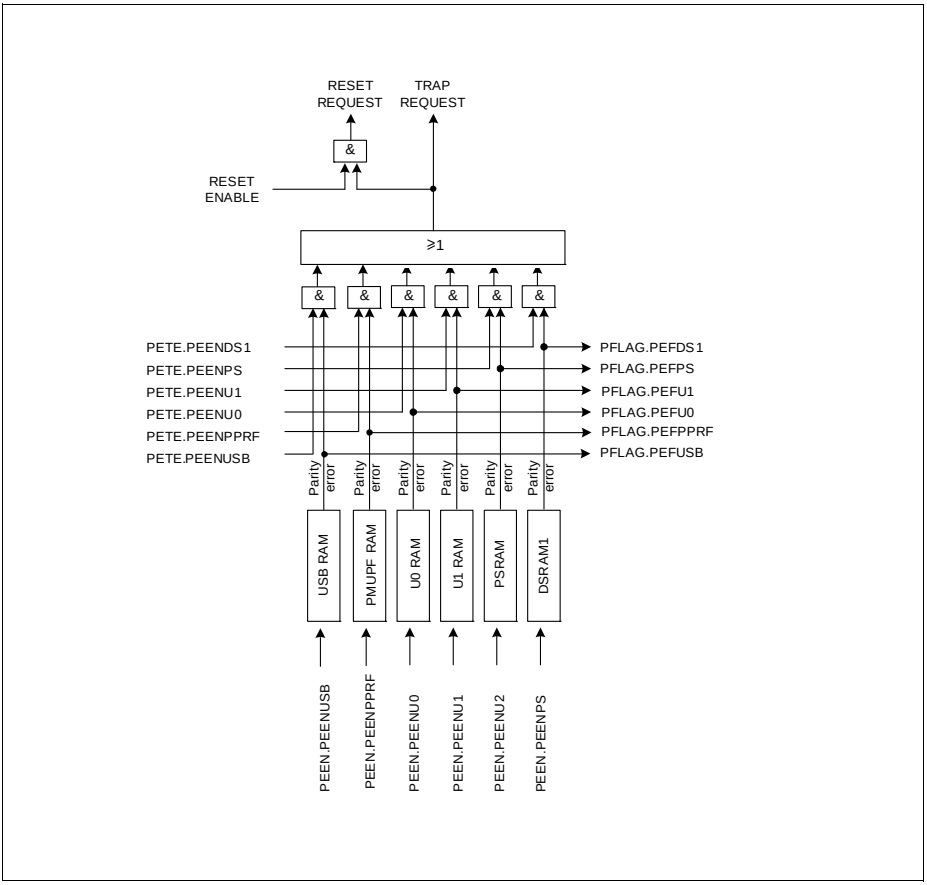


Figure 11-3 Parity Error Control Logic

The logic is controlled by registers **PMTSR** and **PMTPR**. Via bit field **PMTPR**.PWR a parity value can be written to any address of every memory for software driven testing

purpose. The parity control software test update has to be enabled with bit **PMTSR** for each memory individually. Otherwise a write to the parity control has no effect. With each read access to a memory the parity from the memory parity control is stored in register **PMTPR** and accessible with software.

11.2.4 Trap Generation

Several abnormal events listed in **Table 11-2** can result in assertion of NMI. Please refer to **TRAPDIS** register description.

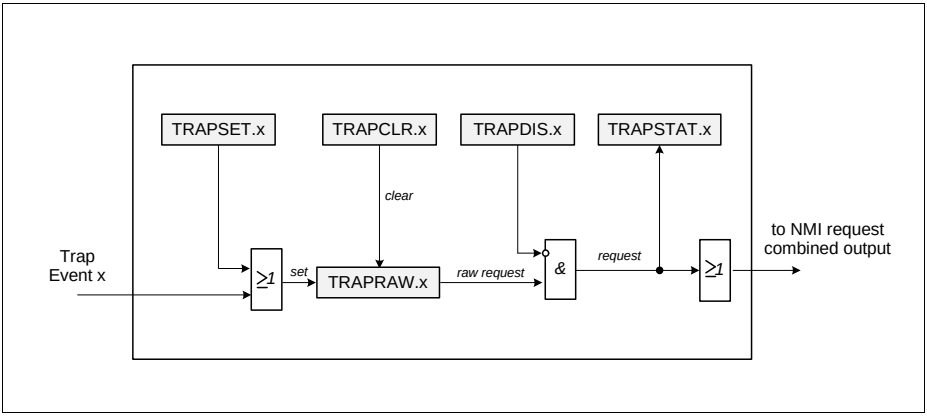


Figure 11-4 Trap Subsystem

The trap flag in register **TRAPSTAT** can be cleared by software by writing to the corresponding bit in register **TRAPCLR**. All trap requests are combined to one common line and connected to NMI node of NVIC.

The trap requests have a sticky flag in register **TRAPRAW**.

Note: When servicing an SCU trap request, make sure that all related request flags are cleared after the identified request has been handled.

11.2.4.1 Trap Sources

The SCU supports trap sources listed in **Table 11-2** and reflected in the **TRAPSTAT**, **TRAPRAW**, **TRAPDIS**, **TRAPCLR** and **TRAPSET** registers.

Table 11-2 SCU Trap Request Overview

Source of Trap	Short Trap Name
OSC_HP Oscillator Watchdog Trap	SOSCWDGT
USB VCO Lock Trap	UVCOLCKT

Table 11-2 SCU Trap Request Overview (cont'd)

Source of Trap	Short Trap Name
System VCO Lock Trap	SVCOLCKT
Parity Error Trap	PET
Brownout Trap	BRWNT
OSC_ULP Oscillator Watchdog Trap	ULPWDGT
Peripheral Bus 0 Write Error Trap	BWERR0T
Peripheral Bus 1 Write Error Trap	BWERR1T
Die temperature too high	TEMPHIT
Die temperature too low	TEMPLOT

11.2.5 Die Temperature Measurement

The Die Temperature Sensor (DTS) generates a measurement result that indicates directly the current temperature. The result of the measurement is displayed via bit field **DTSSTAT**.RESULT. In order to start one measurement bit **DTSCON**.START needs to be set.

The DTS has to be enabled before it can be used via bit **DTSCON**.PWD. When the DTS is powered temperature measurement can be started.

In order to adjust production variations of temperature measurement accuracy bit field **DTSCON**.BGTRIM is provided. **DTSCON**.BGTRIM can be programmed by the user software.

Measurement data is available certain time after measurement started. Register **DTSSTAT**.RDY bit indicated that the DTS is ready to start a measurement. If a started measurement is finished or still in progress is indicated via the status bit **DTSSTAT**.BUSY.

The formula to calculate the die temperature is defined in the Target Data Sheet.

Note: The first measurement after the DTS was powered delivers a result without calibration adjustment and should be ignored.

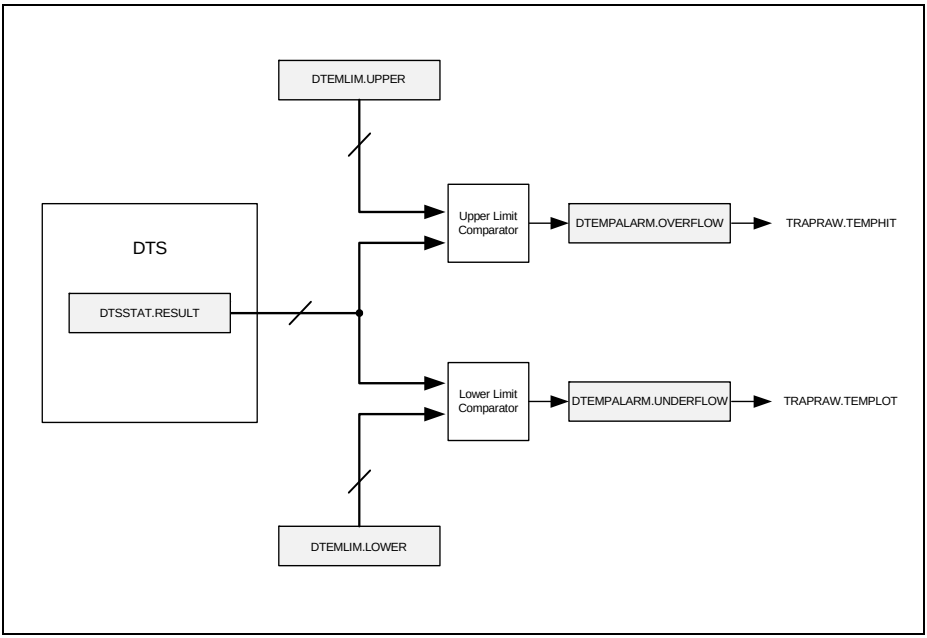


Figure 11-5 DTS service request generation

The GCU is capable of generating system traps requests when DTS temperature measurement in result crosses upper and/or lower threshold value configured with **DTEMPLIM** register. GCU implements digital comparators performing comparison of actual measurement result against configured limits and triggers trap requests accordingly if the value fall outside of valid range. Comparisons are performed only when DTS is active and the outputs remain inactive during DTS power down. Result of comparison is stored in **DTEMPALARM** register.

11.2.5.1 Temperature Measurement

The default formula can be applied in order to calculate the die temperature expressed in °C:

T_{DTS} = (RESULT - 605) / 2.05 [°C]

Where RESULT is a value of RESULT bit field in **DTSSTAT** register.

The accuracy of the measurement can be improved with adjustment of the OFFSET and GAIN bit fields in the **DTSCON** register.

The following formula can be applied in order to reflect relation between the actual adjustment settings and the resulting value in the **DTSSTAT** register:

$RESULT_{CALIB} = RESULT_{UNCALIB} * (1023 / (1023\text{-}GAIN)) + OFFSET$

Where the $RESULT_{CALIB}$ is the value read from the RESULT bit field of the **DTSSTAT** register, reflecting the effect of gain and offset calibration.

Hence, the default formula above is a special case with the default calibration values and can be substituted with:

$T_{DTS} = (RESULT_{CALIB} - 605) / 2.05$ [°C]

*Note: Please always apply the following values to the bit fields of the **DTSCON** register:*

5. ***DTSCON**.REFTRIM = 4_H*
6. ***DTSCON**.BGTRIM = 8_H*

11.2.5.2 Offset Adjustment

Offset Adjustment is defined as a shift of the result. The range of the offset adjustment is 7 bits with a ½LSB resolution, which corresponds to ±12.5 °C.

The offset can be adjusted with OFFSET bit field of **DTSCON** register. The range of the result is limited to 1023_D, The bits are in twos complement format, with the MSB as the sign bit.

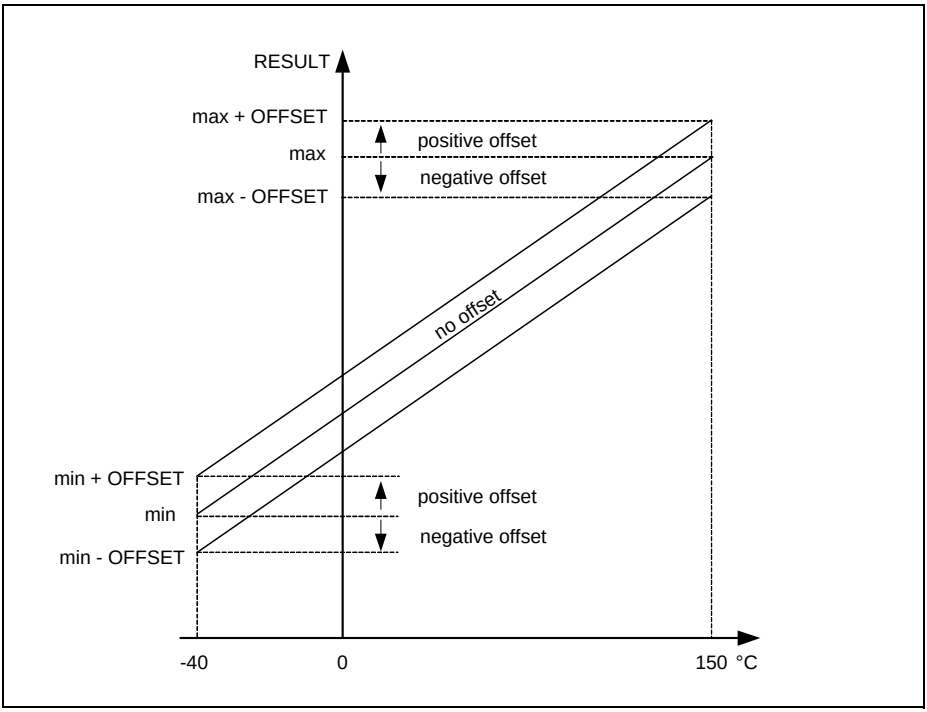


Figure 11-6 Offset adjustment of the RESULT curve

Offset adjustment has direct effect on the RESULT value in **DTSSTAT** register as illustrated in **Figure 11-6**.

11.2.5.3 Gain Adjustment

The most convenient way to express effect of gain on the result is by a curve, fitted to pass from 0 level point to its maximum value at the highest value in the RESULT bit field of **DTSSTAT** register register as illustrated in **Figure 11-7**. The range of the RESULT is limited to 1023_D.

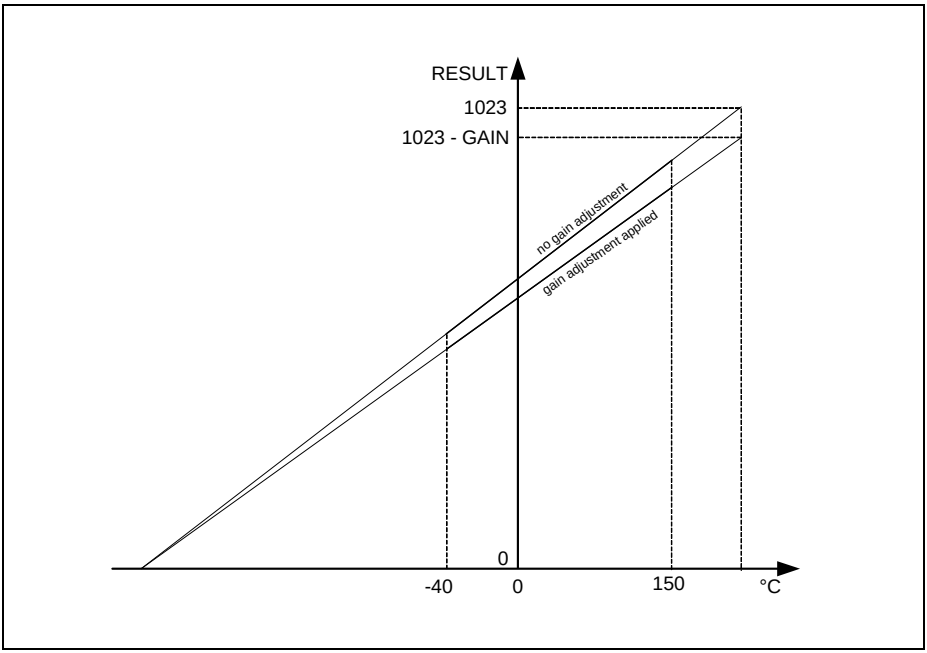


Figure 11-7 Gain adjustment of the RESULT curve

The gain error can be minimzed using 6-bit GAIN bit field of **DTSCON** register where 0 corresponds with a maximum gain and value of 63 corresponds with minimum gain i.e. RESULT value reduced by 63 at the upper imaginary end of the curve.

11.2.6 USB Host Detection

USB Host Detection circuit in **Figure 11-8** enables software controlled selection of an active USB host.

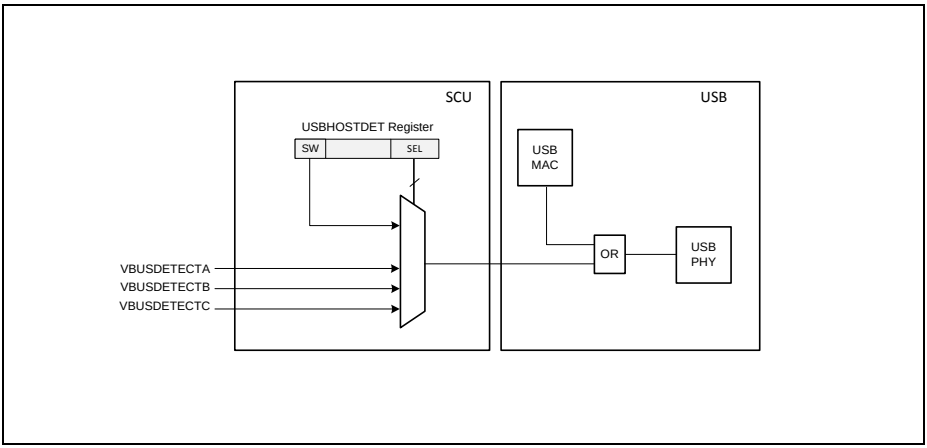


Figure 11-8 USB Host Detection

11.2.7 Retention Memory

Retention memory for context store/restore is available for hibernate mode support. The retention memory is located in Hibernate domain and accessible via regular memory address space. Retention memory content is retained in hibernate mode, while the core might be wpowered off. The user software can store some context critical data in the memory before entering hibernate mode. Access to the data after booting up can be performed at any time with user software. Occurence of a wake-up from hibernate bode is signalized in **RSTSTAT** register.

Access to the 64 Bytes of retention memory is provided via **RMDATA** register, controlled with **RMACR** register. The purpose of **RMACR** register is addressing of the memory cells and issuing read/write command. The **RMDATA** is read or written by software according to the access direction after data transfer has been completed as indicated with RMX bit field in **MIRRSTS**.

11.2.8 Out of Range Comparator Control

The out of range comparator serves the purpose of overvoltage monitoring for analog input pins of the chip. A number of analog channels are associated with dedicated pads connected to inputs of analog modules. They get supervised by dedicated circuits constituted of analog comparator controlled by digital signals as depicted in **Figure 11-9**.

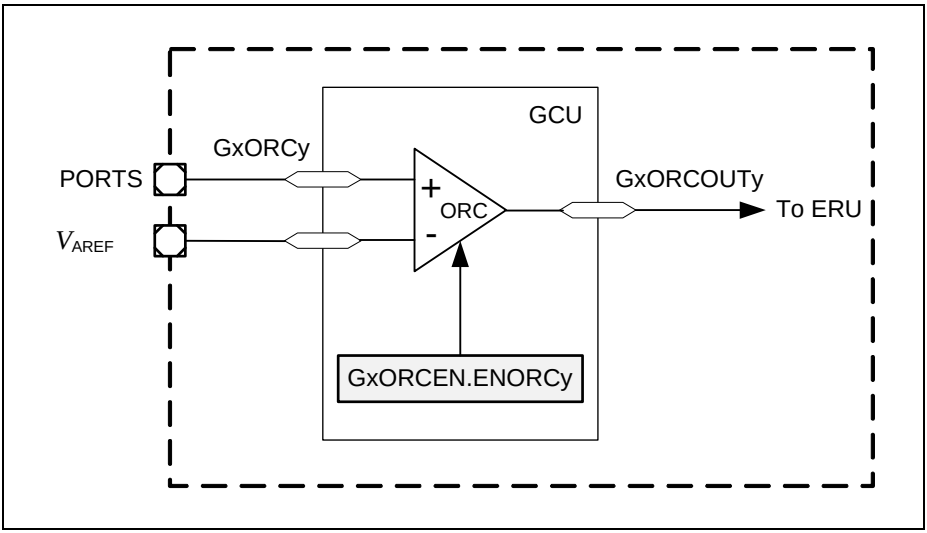


Figure 11-9 Out of Range Comparator Control

Detection of input voltage exceeding V_{AREF} triggers a service request to the ERU. Digital control signals are generated in GCU submodule of SCU. Dedicated registers **G0ORCEN** and **G1ORCEN** provide control means for enabling and disabling monitoring of analog channels.

11.3 Power Management

Power management control is performed with the Power Control Unit (PCU).

11.3.1 Functional Description

The XMC4[12]00 is running from a single external power supply (V_{DDP}). The main supply voltage is supervised by a supply watchdog.

The I/Os and the main part of the flash block are running directly from the external supply voltage. The core voltage (V_{DDC}) is generated by an on-chip Embedded Voltage Regulator (EVR). The safe voltage range of the core voltage is supervised by a power validation circuit, which is part of the EVR.

Logic in the hibernate domain, mainly the real-time clock RTC, hibernate control and retention memory, is supplied by an auxiliary power supply using an additional power pad. The auxiliary V_{BAT} voltage, supplied from e.g. a coin battery, enables the RTC to operate while the main supply is switched off.

11.3.2 System States

The system has the following general system states:

- Active
- Sleep
- Deep Sleep
- Hibernate

Figure 11-10 shows the state diagram and the transitions between these power modes. The additional state power-up is only a transient state which is passed on cold or warm start-up from Off state.

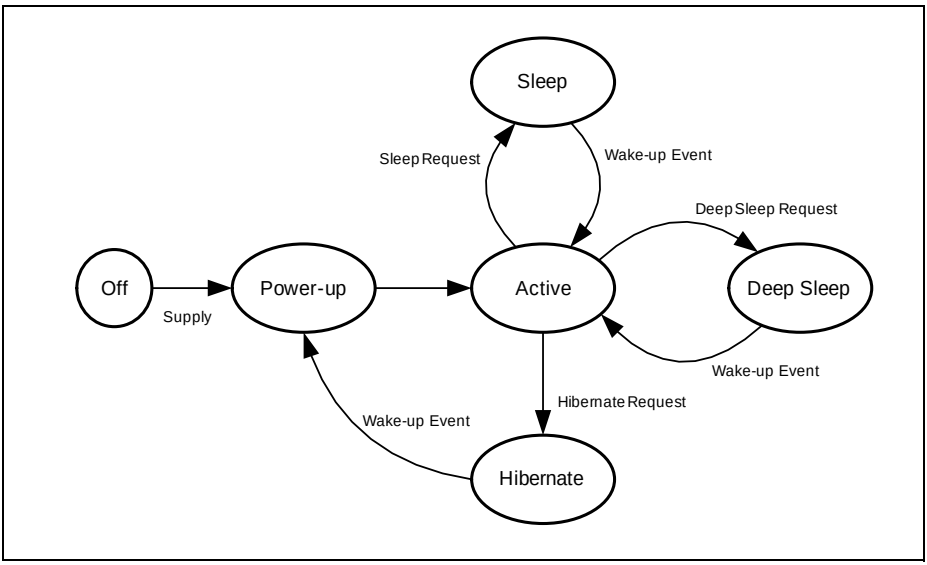


Figure 11-10 System States Diagram

Active State

The Active state is the normal operation state. The system is fully powered. The CPU is usually running from a high-speed clock. Depending on the application the system clock might be slowed down. The PLL output clock or another clock can be selected as clock source. Unused peripherals might be stopped. Stopping a peripheral means that the peripheral i put into reset and the clock to this peripherals is disabled.

After a cold start the hibernate domain stays disabled until activated by user code.

Sleep State

The Sleep state of the system corresponds to the Sleep state of the CPU. The state is entered via WFI or WFE instruction of the CPU. In this state the clock to the CPU is stopped. The source of the system clock may be altered. Peripherals clocks are gated according to the **SLEEPCR** register.

Peripherals can continue to operate unaffected and eventually generate an event to wake-up the CPU. Any interrupt to the NVIC will bring the CPU back to operation. The clock tree upon exit from SLEEP state is restored to what it was before entry into SLEEP state.

Deep Sleep State

The Deep Sleep state is entered on the same mechanism as the Sleep state. User code configures the Deep Sleep state in **DSLEEPCR** control register. In Deep Sleep state the OSC_HP and the PLL may be switched off. The wake-up logic in the NVIC is still clocked by a free-running clock. Peripherals are only clocked when configured to stay enabled in the **DSLEEPCR** register. Configuration of peripherals and any SRAM content is preserved.

The Flash module can be put into low-power mode to achieve a further power reduction. On wake-up Flash module will be restarted again before instructions or data access is possible.

Any interrupt will bring the system back to operation via the NVIC.The clock setup before entering Deep Sleep state is restored upon wake-up.

Hibernate State

In Hibernate mode the power supply to the core is switched off. Additionally the power to the analog domain and the main supply V_{DDP} can be switched off. Only the Hibernate power domain will stay powered. The power supply of the Hibernate domain is switched automatically to the auxiliary supply when the main supply is no longer present.

The Hibernate State is entered using control register **HDCR** of HCU in the Hibernate domain that will drive the Embedded Voltage Regulator (EVR) with HIB signal to switch off power to the Core and Standby Domains.

Depending on configuration the following wake-up sources will wake-up the system to normal operation:

- Edge detection on external WKUP signal
- RTC Alarm Event
- RTC Periodic Event
- OSC_ULP Watchdog Event
- Wake-up ADC interrupt
- Analog input event detection

The system can only wake-up from Hibernate if V_{DDP} is present. An external power supply can be switched on by the HIBOUT signal of the Hibernate Control Unit.

All blocks outside of the Hibernate domain will see a complete power-up sequence upon wake-up.

11.3.3 Hibernate Domain Operating Modes

The standard use case of the Hibernate domain is the time keeping function with V_{BAT} available, keeping the Real Time Counter active and data in Retention Memory while the System Supply is off.

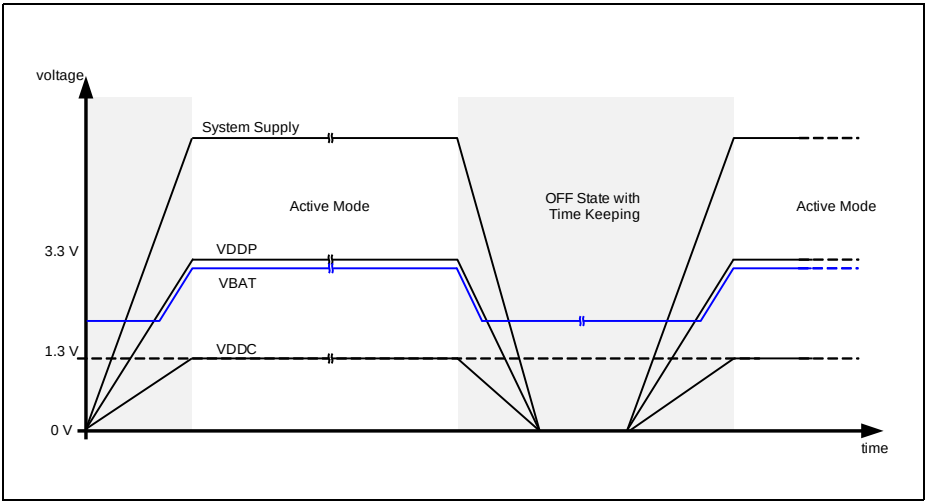


Figure 11-11 Hibernate state in time keeping mode

A special case of the Hibernate domain assumes that the V_{BAT} supply voltage is available only after the core domain power-up. This may occur if, for example, the battery gets plugged in or replaced after the core domain startup, or, if only a capacitor is available to hold V_{BAT} voltage for some limited time in case of absence of the main supply voltage in order to keep the Real Time Counter unaffected. The Hibernate domain requires to be switched on once after core domain power up with the dedicated register **PWRSET**.HIB. In this application case even switching off the main supply of the board does not affect availability of the V_{HIB} voltage required to keep RTC running.

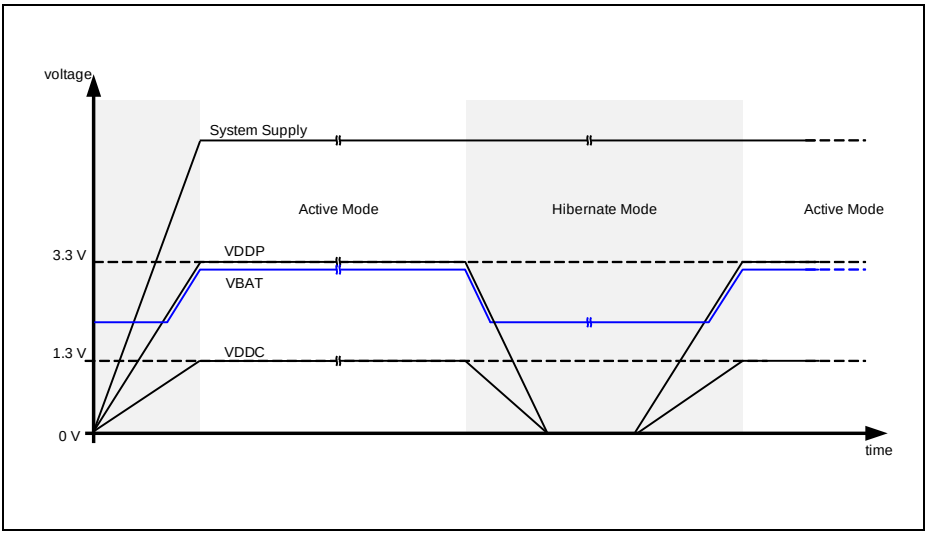


Figure 11-12 Hibernate controlled with external voltage regulator

The externally controlled Hibernate mode is realized with HIB pin and external power supply device. As illustrated in **Figure 11-12** the main supply voltage of the board stays active allowing selective disabling of the XMC4100 / XMC4200 in order to save power while other devices of on the board remain active. At the time HIB pin indicates Hibernate mode the dedicated voltage supply connected to the chip will stop generating V_{DDP} and voltage and in effect complete chip except the Hibernate domain will be powered off. On a wake-up event the Hibernate domain will deactivate the HIB control signal and enable generation of V_{DDP} voltage, hence complete power-up sequence of the core domain.

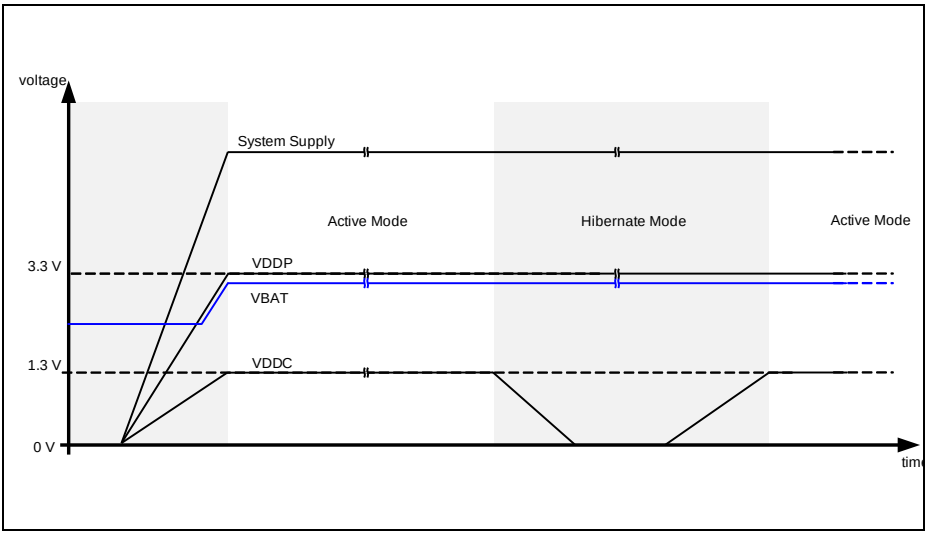


Figure 11-13 Hibernate controlled with internal voltage regulator

The internally controlled Hibernate mode is realized with EVR. As illustrated in **Figure 11-13** the VDDP supply voltage remains stays active. When Hibernate mode is requested EVR stops generating V_{DDC} voltage and in effect complete Core domain gets powered off. On a wake-up event the Hibernate domain will enable generation of V_{DDC} voltage, hence complete power-up sequence of the core domain.

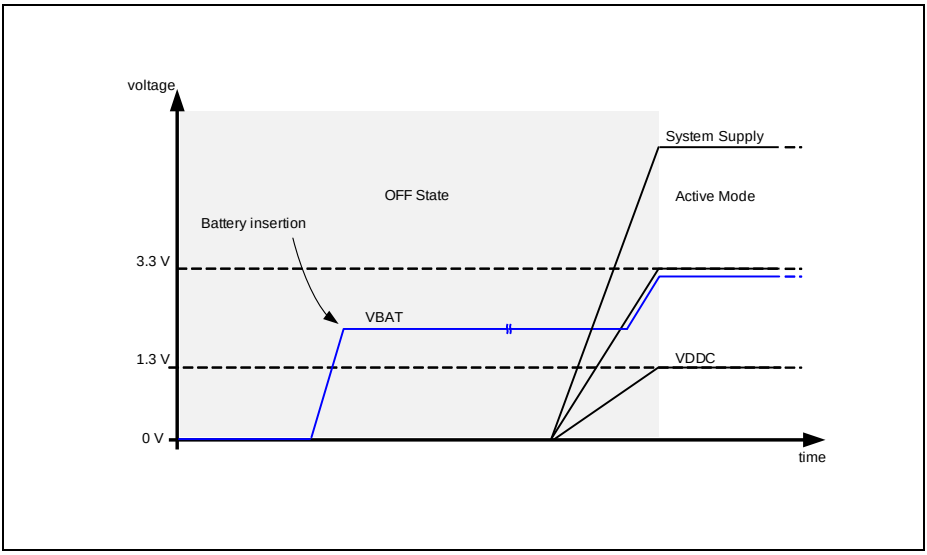


Figure 11-14 Initial power-up sequence

One of the valid power-up scenarios assumes that battery will be installed, possibly soldered, before core supply is available (see **Figure 11-14**). That would be a common situation after PCB assembly, before the complete device is installed in the target system. The battery voltage V_{BAT} gets connected before main supply of the board is available and only the Hibernate domain is supplied. No current (only leakage is allowed) is drawn from the battery before explicit switching on Hibernate domain with software after Core domain power up using dedicated register **PWRSET**.HIBEN. This feature allows to keep the device in a storage for a long time before shipment to the end user, without significant loss of charge in the battery.

11.3.4 Embedded Voltage Regulator (EVR)

The EVR generates the core voltage V_{DDC} out of the external supplied voltage V_{DDP}. The EVR provides a supply watchdog (SWD) for the input voltage V_{DDP}. The generated core voltage V_{DDC} is monitored by a power validation circuit (PV).

11.3.5 Power-on Reset

The EVR starts operation as soon as V_{DDP} is above defined minimum level. It releases the reset, when the external voltage V_{DDP} and the generated voltage V_{DDC} are above the reset thresholds and reaching the nominal values.

11.3.6 Supply Watchdog (SWD)

Figure 11-15 shows the operation of the supply monitor. The supply watchdog compares the supply voltage against the reset threshold V_{POR}. The Data Sheet defines the nominal value and applied hysteresis.

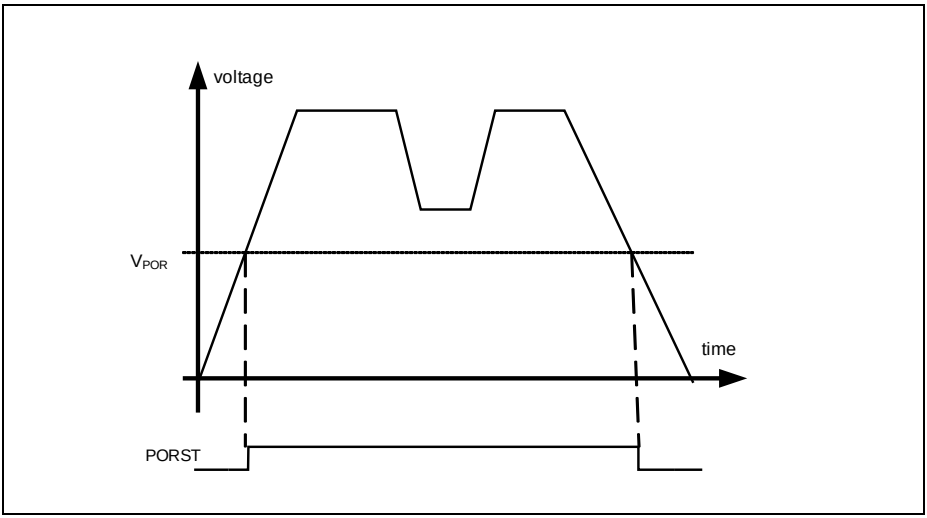


Figure 11-15 Supply Voltage Monitoring

While the supply voltage is below V_{POR} the device is held in reset. As soon as the voltage falls below V_{POR} a power on reset is triggered.

11.3.7 Power Validation

A power validation circuit monitors the internal core supply voltage of the core domain. It monitors that the core voltage is above the voltage threshold V_{PV} which guarantees save operation. Whenever the voltage falls below the threshold level a power-on reset is generated.

11.3.8 Supply Voltage Brown-out Detection

Brown-out detection is an additional voltage monitoring feature that enables the user software to perform some corrective action that bring the chip into safe operation in case a critical supply voltage drop and avoids System Reset generated by the Supply Voltage Monitoring.

A drop of supply voltage to a critical threshold level programmed by the user can be signalized to the CPU with an NMI. An emergency corrective action may involve e.g. reduction of current consumption by switching of some modules or some interaction with external devices that should result in recovery of the supply voltage level.

Automatic monitoring of the voltage against programmed voltage allows efficient operation without a need for software interaction if the supply voltage remains at a safe level. The supply voltage can be monitored also directly by software in register **EVRVADCSTAT**. The threshold value and the inspection interval is configured at **PWRMON**.

11.3.9 Hibernate Domain Power Management

It is strongly recommended to supply Hibernate domain with V_{DDP} when available in order to extend the battery life time. An example of an external supply voltage switching solution based on Schottky diodes is shown in **Figure 11-22**.

11.3.10 Flash Power Control

The Flash module can be configured to operate in low power mode while the system is in Deep Sleep mode. Control of the Flash power mode it is performed using the **DSLEEPCR** register prior to entering the Deep Sleep mode. The Flash cannot be accessed while in the lower mode. Upon a wake-up event the Flash gets automatically reactivated. The user needs to trade the reduced leakage current against wake-up time of the Flash.

11.4 Hibernate Control

Hibernate is the operation mode of lowest power consumption. Activity of the microcontroller is limited to real time keeping and wake up functionality.

11.4.1 Hibernate Mode

Hibernate control is performed with Hibernate Control Unit (HCU).

Externally Controlled Hibernate Mode

In this operation mode of the with lowest power consumption only the hibernate domain remains powered up. In this state the hibernate domain enable switching off externally the main power supply (see **Figure 11-22**).

External Voltage Regulator is controlled from Hibernate domain and V_{DDP} gets switched off when hibernate mode is entered. No reset of hibernate control occurs in hardware upon power-up but the chip will boot up as normal since in this mode no internal state of the chip is affected except for the hibernate control pin from hibernate domain to the External Voltage Regulator. Re-initialization of is possible with software upon boot-up.

Internally Controlled Hibernate Mode

V_{DDP} is applied to the chip but EVR stops generating V_{DDC} voltage. PORST is ignored while in hibernate mode and pulled down internally while in internally controlled hibernate mode and reset of hibernate control logic brings it back to normal operation (see **Figure 11-23**).

Externally Controlled Hibernation Support

The entry of the hibernate state is configured via the register **HDCR** by setting of the HIB bit. The HIBOUT bit in conjunction with selected HIBIOnPOL bit of **HDCR** register drives HIB_IO_n pad.

The hibernation control signal HIBOUT is connected to an open-drain pad to enable control of an external power regulator for V_{DDP}.

Internally Controlled Hibernation Support

The entry of the hibernate state is configured via the **HINTSET** register. Generation of V_{DDC} voltage gets switched of on EVR.

Wake-up from Hibernate Mode

The hibernation control supports multiple wake-up sources. Occurrence of any of the Wake-up from hibernate triggers resets of **HDCR**.HIB bit which results resetting of the HIBOUT signal that controls optional External Voltage Regulator.

Recovery from Internally Controlled Hibernate State without Wake-Up Event

Removal and re-applying of V_{DDP} will always result reset of internally controlled hibernate state i.e. restore default values of HCU control signals in order to enable EVR to generate V_{DDC} required for normal power-up and boot sequence and enable normal Flash operation, PORST pad pull-down will be deactivated. This mechanism is primarily intended for recovery from internally controlled hibernate mode without applying valid wake-up trigger. This may be the useful e.g. if the internal hibernate mode has been entered without configured wake-up condition.

11.4.2 Low Power Analog Comparator (LPAC)

The analog wake up Request module serves a function of voltage level watchdog for external voltage source and/or for battery supply V_{BAT}. The LPAC module triggers wake-up event from Hibernate state or an interrupt while in normal operation mode if an analog voltage level crossed pre-programmed voltage level in pre-programmed direction.

The alternate function of the LPAC module in active mode is continuous monitoring of external voltage source and generation of interrupts upon threshold crossing according to applied configuration.

Typical use cases:

- battery supervisory
- temperature watchdog for external sensor
- user potentiometer
- supply current supervisory
- wake-up from application idle state
- tamper protection
- general wake-up from Hibernate, Sleep and Deep Sleep

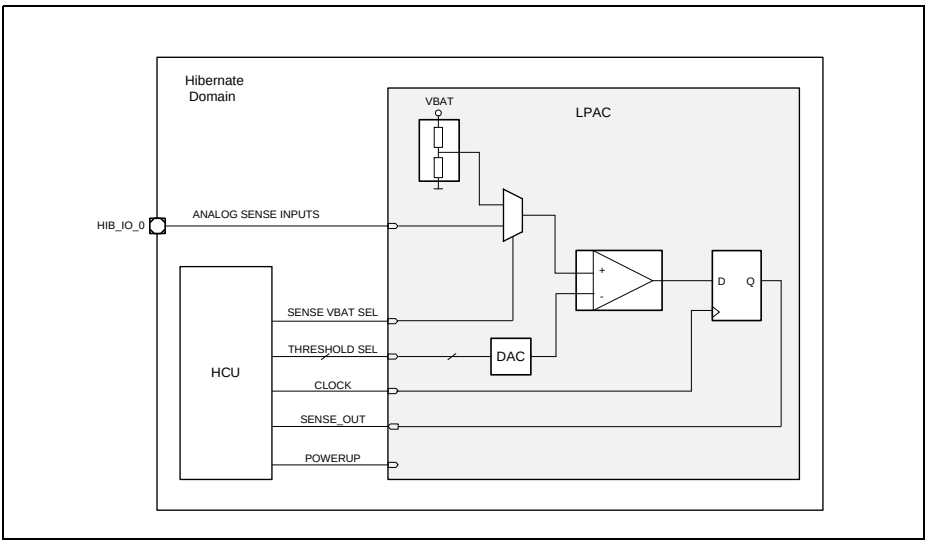


Figure 11-16 Low Power Analog Comparator (LPAC) with singleanalog input

Control logic of the LPAC module is located in HCU module of Hibernate domain.

The HCU implement state machine that periodically powers up the LPAC module, ensures required time to settle down for the analog circuit and performs measurement - comparison of a selected analog input against DAC programmed reference voltage. Result of comparison gets latched in HCU module until next measurement and transferred to mirror registers in SCU. The value latched of comparison output is also passed directly to SCU service request register in Core domain where it can be used by software while in normal mode.

The LPAC as such may be considered independent from the HCU wake-up functionality and may be used for the purpose of a wake-up trigger as well as in normal application mode.

The LPAC control state machine in HCU module is capable of performing multiple interleaved measurements from time multiplexed analog inputs. For each voltage source

individual configuration values get applied accordingly upon measurement. For battery voltage measurement the input voltage divider (see **Figure 11-16**) is enabled only during measurement and remain powered off for the rest of the time. Configuration of the measurements is applied via LPAC dedicated registers sliced in HCU module.

Measurement interval is configurable and may be practically continuous which implies that the analog circuits are powered up all the time, or, periodic, where analog modules get powered up at the interval of the range of 1 ms up to long periods configurable with the RTC module. Another possible mode is measurement on RTC alarm.

Digitally Controlled Hysteresis

The Hysteresis is digitally emulated by the means of dynamic re-programming of the reference value to the DAC of LPAC accordingly from HCU state machine. After reset of HCU the upper threshold is applied to LPAC for all consecutive measurements until it has been crossed upwards (see **Figure 11-17**). Once upper threshold crossed upwards the lower threshold gets applied and remains applied for all consecutive measurements until it has been crossed downwards and the threshold values get swapped again.

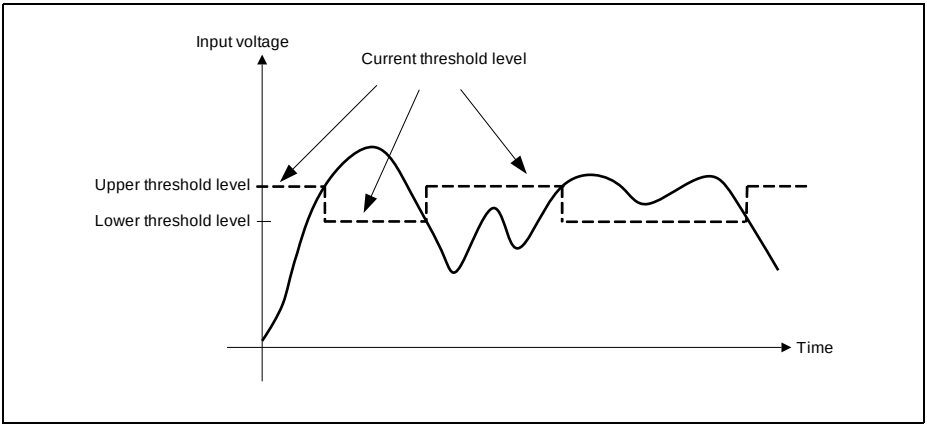


Figure 11-17 Hysteresis with default initial threshold level

The threshold selection depends on the last measurement result but it can be also initialized with **LPACSET/HINTCLR** bits e.g. LPAC may be configured to assume that the last crossing was upwards and therefore lower threshold needs to be applied (see **Figure 11-18**).

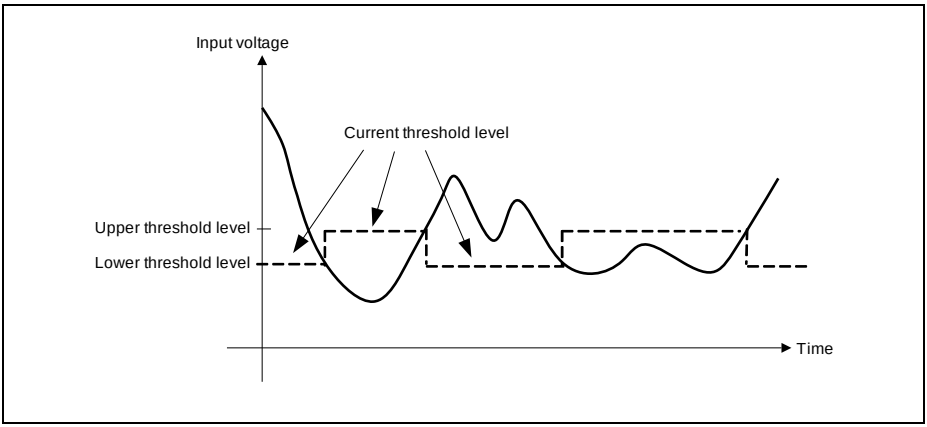


Figure 11-18 Hysteresis with initial threshold level inverted

Multi-channel Operation

Multi-channel LPAC operation is emulated with the means of time-multiplexing of analog input signal. The control state machine of HCU applies threshold values and all relevant control signals accordingly. Number of active channels is configurable and measurement interval ranging from continuos compare operation to practically unlimited interval duration.

Software Controlled Threshold Calibration

The accuracy of LPAC threshold can be improved with software means enabling runtime calibration of the intended threshold level. The LPAC analog components are shared for all analog channels the therefore it is sufficient to determine the digital threshold setting value corresponding with a known voltage level on any of the analog inputs. The programming values of the LPAC threshols values need to be adjusted accordingly for valid analog channels, reflecting the mesurement error performed with the LPAC module. The threshold values need to be remapped proportionally for entire valid voltage range of each of the the calibrated channels.

The VBAT correction concept illustrated in **Figure 11-19** assumes use of a headroom available in order to enable mapping of the sampled values in the presence of a mesurement error (for details please refer to the Data Sheet)

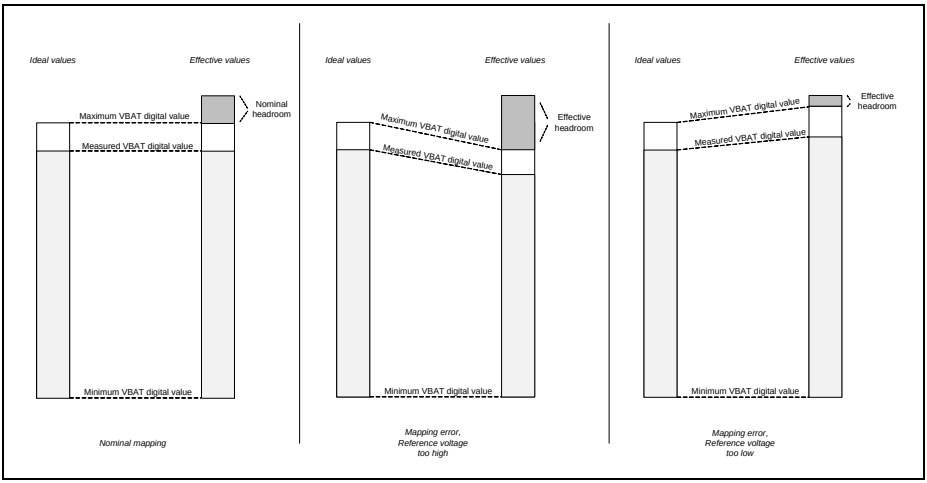


Figure 11-19 VBAT correction example

A simple procedure assuming that the VBAT voltage equals VDDP voltage can be applied. The VDDP voltage level value is continuously monitored in the EVR module and can be read out via **EVRVADCSTAT** register. The measured voltage level shall serve as a reference for VBAT input mesurements using LPAC. The VBAT level will be determined with an iterative approach where the VBAT channel threshold level in the **LPACTH0** register gets re-programmed accordingly in order to find the value causing transition of the level detection staus bit VBAT of the **LPACST** register.The procedure should be performed once after initial power up of the chip. The correction values can be stored for a later use in the retention memory of the Hibernate domain.

An example of a simple measurement procedure is shown in **Figure 11-20**.

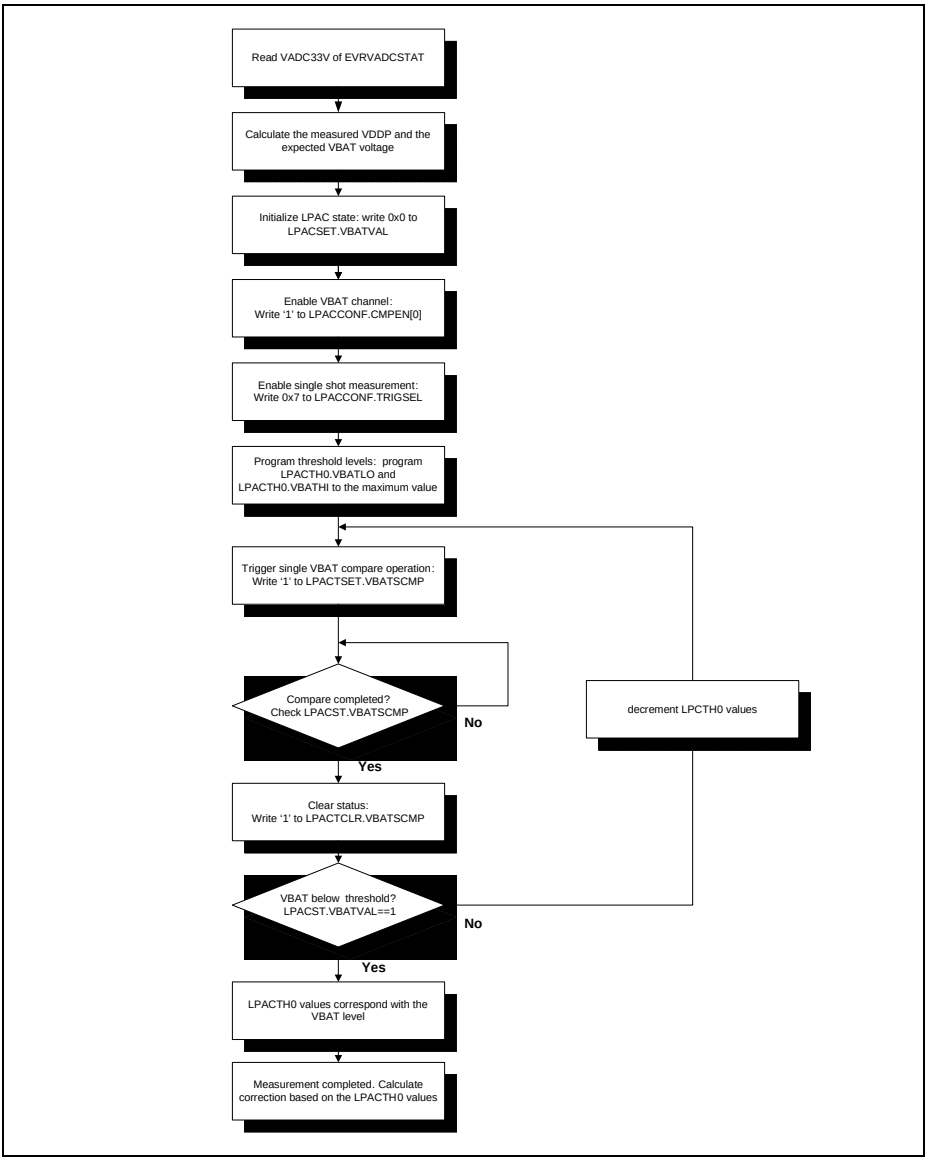


Figure 11-20 LPAC Calibration Procedure

Note: The accuracy of the LPAC threshold calibration may vary depending on the application specific conditions, e.g. voltage difference on the VBAT and VDDP pins of the chip, presence of noise on supply during conscutive mesurement iterations.

11.4.3 Hibernate Domain Pin Functions

Hibernate domain control register **HDCR** implements fields for alternate pin function selection.

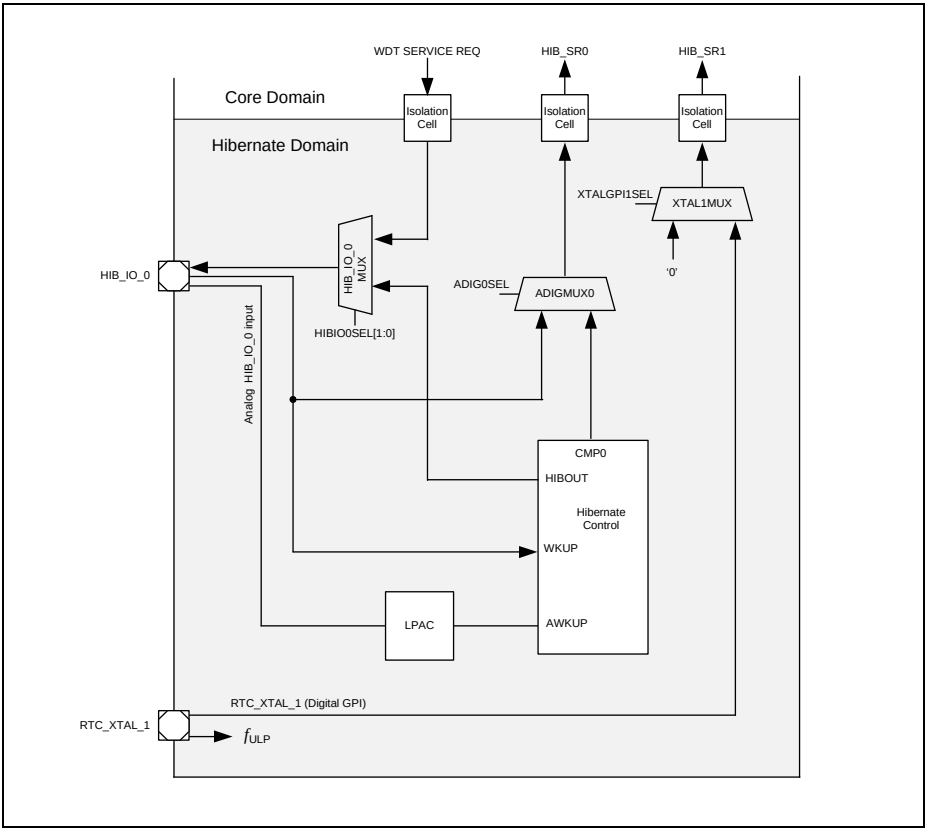


Figure 11-21 Alternate function selection of HIB_IO_0 pin of Hibernate Domain

11.4.4 System Level Integration

The XMC4[12]00 enables various system level configuration options for Hibernate mode support, determined by application specific requirements. The examples shown below illustrate functional principles of the hibernate control mechanism in the system level intergration aspect.

Externally Controlled Hibernate mode

The Externally Controlled Power Supply scheme require external devices on in order to fully support power management related functions.

The external power supply needs to support on/off control of the VDDP voltage generation. This scheme enables the XMC4100 / XMC4200 e.g. to control supply voltage for multiple on-board devices.

The HIB_IO_0 pad is configured as open drain low

In case of application cases with only one hibernate control pin available for external power control, it is possible to use the same pin for control of the External Voltage Regulator and for an external wake-up trigger signal. The example setup is shown in **Figure 11-22**, where HIB_IO_0 pin is by default configured to open-drain mode.

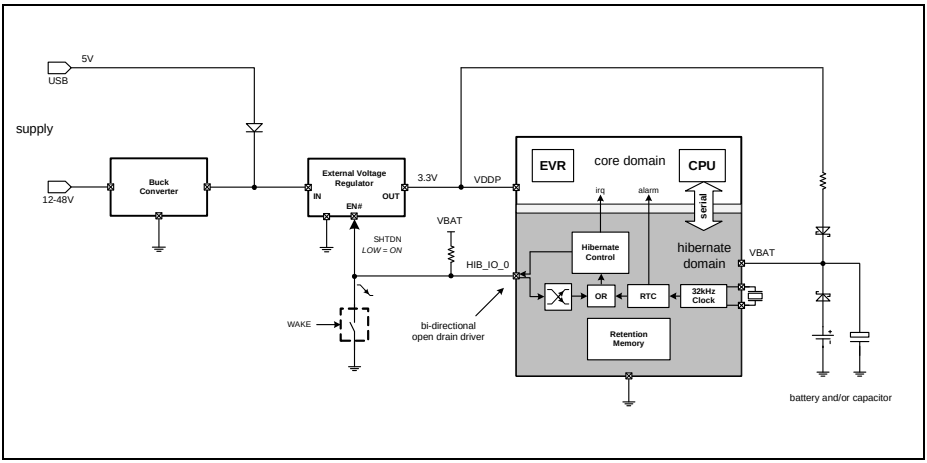


Figure 11-22 System Level Power Control example - externally controlled with single pin

After power up and before entering Hibernate mode the HIB_IO_0 needs to be reconfigured to bidirectional (but still open-drain driver) mode. Hibernate mode activation automatically changes state of the HIB_IO_0 output to high impedance which allow the external pull up resistor to drive high value of the External Voltage Regulator in order to switch off VDDP generation. A wake-up trigger signal from external source to the

Hibernate Control logic will propagate via the bidirectional HIB_IO_0 pin. A wake-up trigger needs to be driven by an external open-drain device capable to overcome driving strength of the pull-up resistor.

Internally Controlled Hibernate Mode

The Internally Controlled Power Supply scheme assumes that VDDP generation isn't controlled by the XMC4100 / XMC4200 and VDDC voltage is controlled with the on-chip Embedded Voltage Regulator (EVR) instead, as shown in **Figure 11-23**. An External Voltage Regulator without on/off control may be used in this scenario.

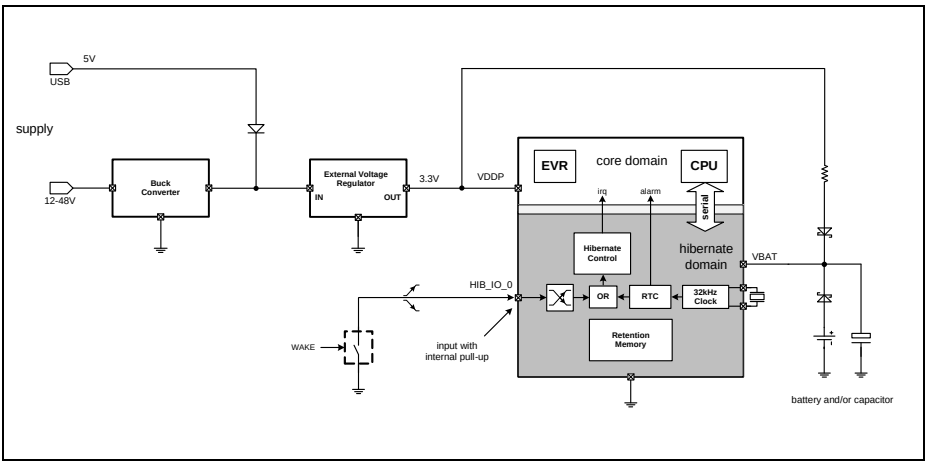


Figure 11-23 System Level Power Control example - internally controlled

The external switch based on a Shottky diode prevents discharging battery when voltage is supplied to the chip from the external power supply.

11.5 Reset Control

Reset Control Unit (RCU) performs control of all reset related functionality including:

- Reset assertion on various reset request sources
- Inspection of reset sources after reset
- Selective reset of peripherals

11.5.1 Supported Reset types

The XMC4[12]00 implements the following reset signals:

- Power-on Reset, $\overline{\text{PORESET}}$
- System Reset, $\overline{\text{SYSRESET}}$

- Standby Reset, $\overline{\text{STDBYRESET}}$
- Debug Reset, $\overline{\text{DBGRESET}}$

Power-on Reset ($\overline{\text{PORESET}}$)

A complete reset of the core domain of the device is executed upon power-up. Whenever the supply V_{DDP} is ramped-up and crossing the PORST voltage threshold the power-on reset is released. Additional a power-on reset is triggered by asserting the external PORST pin.

A Power-on reset is asserted again whenever the V_{DDP} voltage or the V_{DDC} voltage falls below defined reset thresholds.

System Reset ($\overline{\text{SYSRESET}}$)

System Reset is triggered by sources:

- Power-on reset
- Software reset via Cortex-M4 Application Interrupt and Reset Control Register (AIRCR)
- Lockup signal from Cortex-M4 when enabled at RCU
- Watchdog reset
- Memory Parity Error

The System Reset resets almost all logic in the core domain. The only exceptions in the core domain are RCU Registers and Debug Logic.

Standby Reset, ($\overline{\text{STDBYRESET}}$)

The Hibernate domain including the RTC is only reset by a standby reset. A standby reset is triggered by a power-on reset specific to the Hibernate domain. Additionally a standby reset can be activated by software:

- Power-on reset specific to the Hibernate domain
- Software reset via **RSTSET** register

Debug Reset ($\overline{\text{DBGRESET}}$)

Debug reset is triggered by the following sources:

- Debug Reset request from DAP while in mission mode and with debug probe present
- System Reset while in normal mode and debug probe not present

The Debug Reset is triggered by System Reset while in normal operation mode while debug probe is not present.

11.5.2 Peripheral Reset Control

Software can activate the reset of all peripherals individually via the registers **PRSET0**, **PRSET1**, and **PRSET2**. The default state is that all peripherals are in reset after power-up. A return to the default state of a peripheral can be performed by forcing it to reset state by a separate reset.

The user needs to properly configure the port values before resetting a module to assure that the default output values of a peripheral do not harm external circuitry. Similarly the user has to take care of top-level interconnects, which will not be affected by a module specific reset.

11.5.3 Reset Status

The EVR provides the cause of a power reset to the RCU. The reset cause can be inspected after resuming operation by reading register **RSTSTAT**.

All register of the RCU undergo reset only by a power-on reset $\overline{\text{PORESET}}$.

Table 11-3 shows an overview of the reset signals their source and effects on the various parts of the system.

Table 11-3 Reset Overview

Name	Source	System (Core) Domain	PAD Domain	Hibernate Domain	Debug & Trace System
$\overline{\text{PORESET}}$	EVR	yes	yes	no	yes
$\overline{\text{SYSRESET}}$	$\overline{\text{PORESET}}$ Software WDT	yes	no	no	no
$\overline{\text{STDBYRESET}}$	Power-on Software	no	no	yes	no
$\overline{\text{DBGRESET}}$	DAP Software	no	no	no	yes

11.6 Clock Control

Clock generation and control is performed in the Clock Control Unit (CCU).

11.6.1 Block Diagram

The Clock Control Unit (CCU) consists of two major sub blocks:

- Clock Generation Unit (CGU)
- Clock Selection Unit (CSU)

The CGU provides in parallel three clocks to the CSU:

- USB PLL clock f_{PLLUSB},
- System PLL output clock f_{PLL}
- internally generated clock f_{OFI} from the Backup Clock Source.

The f_{STDBY} clock of 32.768 kHz is generated in the Standby Clock Generation Unit (SCGU) of the hibernate domain by either the external crystal oscillator OSC_ULP or by Internal Slow Clock Source.

The module clocks available in the Core Domain get derived from the CGU, passed via CSU module which implements a number of multiplexers and clock dividers enabling clock selection for all system modules and scaling of the frequencies.

The CSU receives in addition a slow standby clock f_{STDBY} from the hibernate domain that can be used in the WDT module for safety sensitive application purpose.

Major clock sources can be selected for driving the external clock pin EXTCLK.

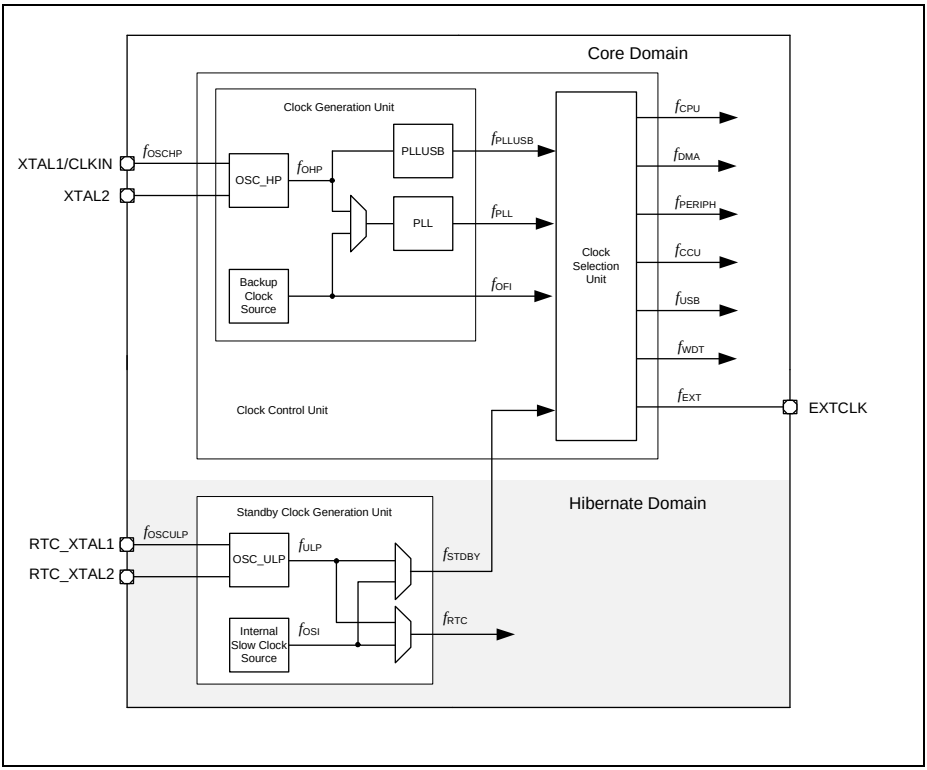


Figure 11-24 Clock Control Unit

11.6.2 Clock Sources

The system has multiple clock sources distributed over the core power domain and the hibernate power domain.

The source clock for CGU can be supplied from:

- internally generate in the Backup Clock Source
- external clock source directly via CLKIN pin
- external crystal High Precision Oscillator (OSC_HP) utilizing XTAL1 and XTAL2 pins

During system start-up the backup clock f_{OFI} of is supplied to the system. The f_{OFI} clock may be used during normal or low power operation after the startup.

The high precision oscillator OSC_HP can be used with an external crystal to generate high precision clock via XTAL1 and XTAL2 pins. Either, the OSC_HP or Backup Clock Source can be selected as input clock source for the main PLL. Alternatively, an external

clock can also be supplied directly to CGU from CLKIN pin, via the OSC_HP module, bypassing the oscillator circuit.

The clock sources in the hibernate power domain are:

- OSC_ULP, an ultra low power external crystal oscillator 32.768 kHz clock f_{ULP}
- Internal Slow Clock source 32.768 kHz clock f_{OSI}

The clocks drive the logic in the hibernate domain. In addition one of the two can be used as standby clock for low power operation in the core power domain.

To generate the required high precision f_{PLLUSB} clock for operation of the USB moduleadditional PLL can be optionally used if the desired clock frequency cannot be derived from the system PLL output f_{PLL}. The dedicated PLL referred to as PLLUSB, shown the block diagram in **Figure 11-25**.

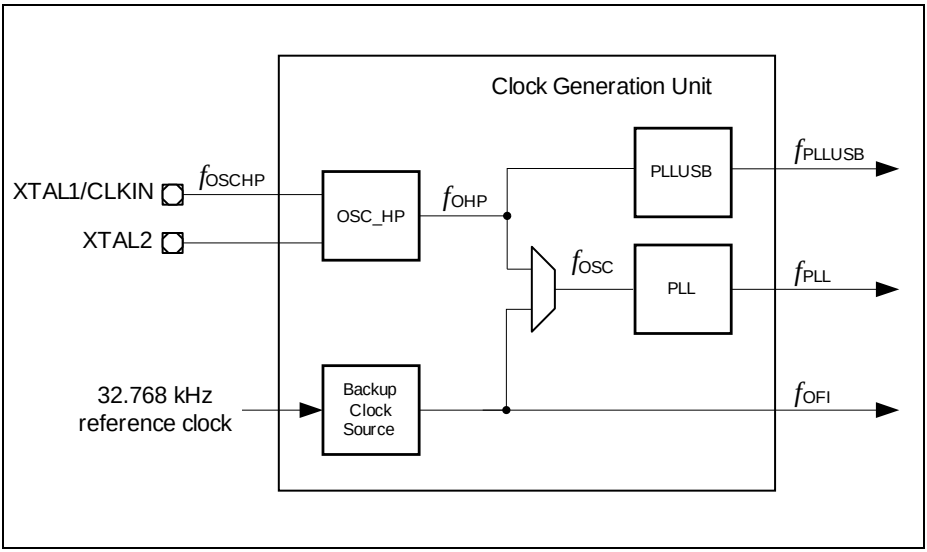


Figure 11-25 Clock Generation Block Diagram

11.6.3 Clock System Overview

The clock selection unit CSU provides the following clocks to the system:

Table 11-4 Clock Signals

Clock name	From/to module or pin	Description
System Clock Signals		
f_{SYS}	SCU.CCU	System master clock
f_{CPU}	CPU	CPU and NVIC clock
f_{DMA}	DMA0	DMA clock
f_{PERIPH}	PBA0 PBA1	Peripheral clock for modules connected to peripheral bridges PBA0 and PBA1
f_{CCU}	CCU4 CCU8 POSIF	Clock for CCU4, CCU8 and POSIF modules
f_{USB}	USB	UTMI clock of USB
f_{WDT}	WDT	Clock for independent watchdog timer
Internal Clock Signals		
f_{PLL}	System PLL	System PLL output clock
f_{PLLUSB}	USB PLL	USB PLL output clock
f_{OHP}	OSC_HP	External crystal oscillator output clock
f_{OFI}	Internal Backup Clock Source	System Backup Block
f_{ULP}	OSC_ULP	External crystal slow oscillator output clock
f_{OSI}	Slow Internal Backup Clock Source	32.768 kHz backup clock for Hibernate domain
f_{OSC}	PLL input	PLL input clock
f_{STDBY}	Hibernate Domain	32.768 kHz clock, optional WDT independent clock
f_{FLASH}	FLASH	Internal Flash clock
External Clock Signals		
f_{OSCHP}	XTAL1/CLKIN	External crystal input, optionally also direct input clock to system PLL

Table 11-4 Clock Signals (cont'd)

Clock name	From/to module or pin	Description
f_{OSCULP}	RTC_XTAL1	External crystal input, optionally also direct input clock to Hibernate domain and RTC module
f_{EXT}	Clock output to pin EXTCLK	Chip output clock

The clock sources of the clock selection unit are the four clocks from the clock generation unit, i.e. the USB clock f_{USB}, the PLL output clocks f_{PLL} and a fast internal generated clock f_{OFI}.The clock selection unit receives as additional clocks source a slow standby clock from the hibernate domain.

11.6.3.1 Clock System Architecture

The system clock f_{SYS} drives the CPU clock and all peripheral modules. It can be selected from the following clock sources:

- f_{PLL}, main PLL output clock
- f_{OFI}, fast internal clock, bypassing PLL
- f_{OSCHP}, external clock, bypassing PLL
- f_{OHP}, external crystal oscillator clock, bypassing PLL

Please note that in dependence of the PLL mode setting the PLL output clock f_{PLL} is either a scaled version of the VCO clock in normal mode or a scaled version of one of the input clocks.

In Deep Sleep mode the system clocks can be switched to a clock which does not require PLL and thereby allowing the power down of the system PLL. This is either the fast internal clock or the slow standby clock. The **DSLEEPCR** register controls the clock settings for Deep Sleep mode.

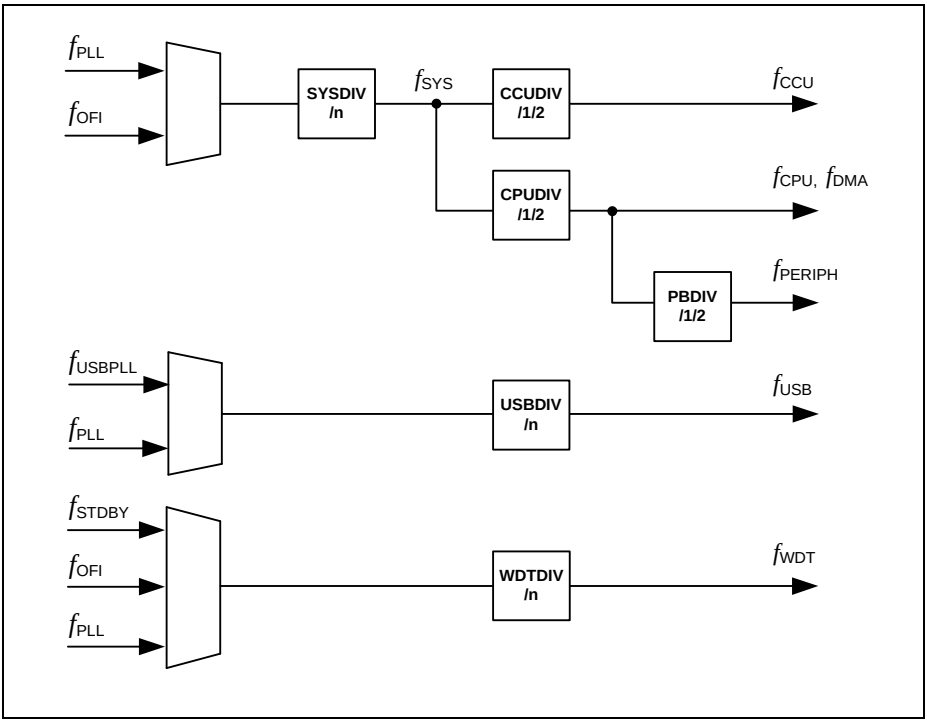


Figure 11-26 Clock Selection Unit

Some limitations apply on clock ratio combinations between f_{CCU}, f_{CPU} and f_{PERIPH}. Only divider setting listed in the table **Table 11-5** are allowed for the f_{CCU}, f_{CPU} and f_{PERIPH} clocks. All other clock dividers settings must be prohibited by software applications by hardware when programmed with **MLINKCLKCR** register in order to avoid invalid clock ratios leading to system malfunctions. The **Table 11-5** table shows also example clock rate values assuming f_{SYS} rate of 80 MHz.

Table 11-5 Valid values of clock divide registers for f_{CCU}, f_{CPU} and f_{PERIPH} clocks

CCUCLKCR.CCUDIV	CPUCLKCR.CPUDIV	PBCLKCR.PBDIV
0 (80 MHz)	0 (80 MHz)	0 (80 MHz)
0 (80 MHz)	0 (80 MHz)	1 (40 MHz)
0 (80 MHz)	1 (40 MHz)	0 (40 MHz)
1 (40 MHz)	0 (80 MHz)	1 (40 MHz)
1 (40 MHz)	1 (40 MHz)	0 (40 MHz)

CPU Clock Selection

The CPU clock f_{CPU} may be equal to or a half of the system clock f_{SYS}.

Peripheral Bus Clock Selection

The Peripheral Bus clock is derived from the f_{CPU} clock and may be equal t or a half of the f_{CPU}. There are some limitations imposed on peripheral bus configuration in term of clock ratio with respect to the f_{CCU} clock. The Peripheral Bus clock rate must never be less than half and more than the f_{CCU} clock.

CAPCOM and POSIF Clock Selection

The capture compare blocks CCU4, CCU8 and POSIF use as timer clock the clock f_{CCU} derived from system f_{SYS} via CCUDIV clock divider. The clock divider allows to adjust clock frequency of the timers with respect to the rest of the system. Relation between f_{CCU} clock and other clocks in the system is constrained as described in the **Table 11-5**.

The f_{CCU} clock frequency must only be configured while all CAPCOM and POSIF modules are not enabled.

Watchdog Clock Selection

The watchdog module uses as independent clock either the internal fast clock f_{OFI} or the slow clock f_{STDBY}. The system clock f_{PLL} is available as additional clock source.

Both the clock divider and the clock MUX must only be configured while the WDT is not enabled.

External Clock Output

An external clock is provided via clock pin EXTCLK. All clock sources can be selected as clock signal for the external clock output. Optionally a divider can be used before bringing the system clock to the outside to stay within the limit of the supported frequencies for the pads.

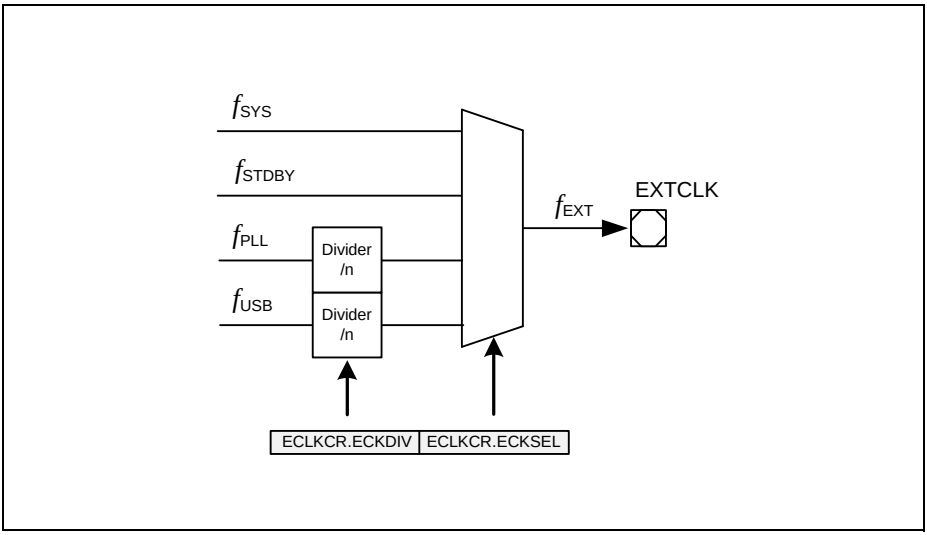


Figure 11-27 External Clock Selection

11.6.4 High Precision Oscillator Circuit (OSC_HP)

The high precision oscillator circuit can drive an external crystal or accepts an external clock source. It consists of an inverting amplifier with XTAL1 as input, and XTAL2 as output.

Figure 11-29 and **Figure 11-28** show the recommended external circuitries for both operating modes, External Crystal Mode and External Input Clock Mode.

External Input Clock Mode

In this usage an external clock signal is supplied directly not using an external crystal and bypassing the amplifier of the oscillator. The input frequency must be in the range from 4 to 40 MHz.When using an external clock signal it must be connected to XTAL1. XTAL2 is left open i.e. unconnected.

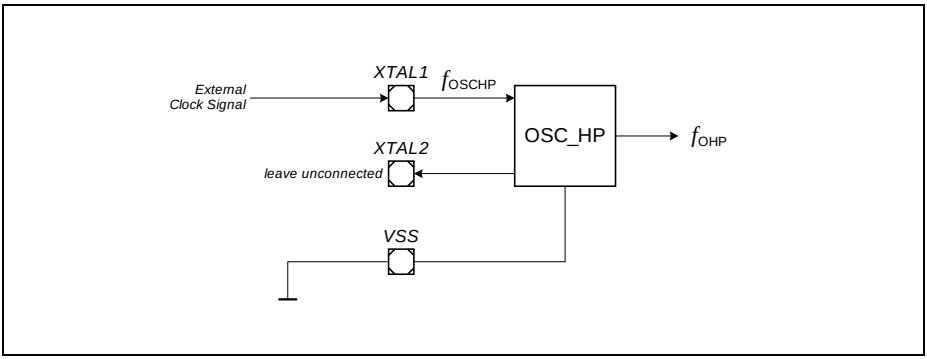


Figure 11-28 External Clock Input Mode for the High-Precision Oscillator

External Crystal Mode

For the external crystal mode an external oscillator load circuitry is required. The circuitry must be connected to both pins, XTAL1 and XTAL2. It consists normally of the two load capacitances C1 and C2. For some crystals a series damping resistor might be necessary. The exact values and related operating range depend on the crystal and have to be determined and optimized together with the crystal vendor using the negative resistance method.

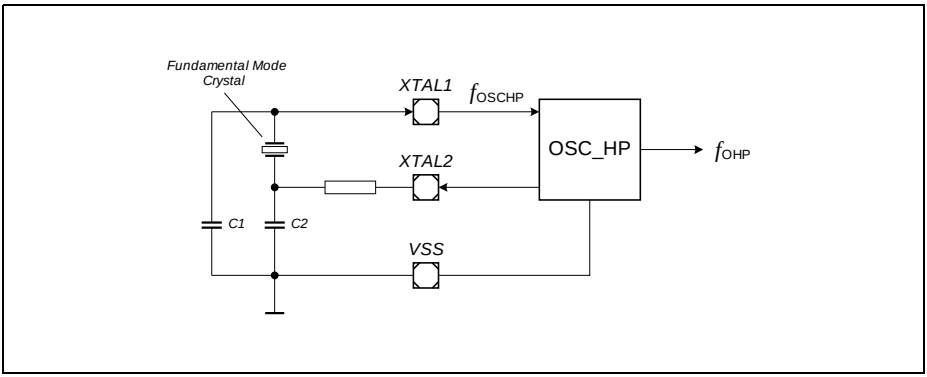


Figure 11-29 External Crystal Mode Circuitry for the High-Precision Oscillator

11.6.5 Backup Clock Source

The backup clock f_{OFI} generated internally is the default clock after start-up. It is used for bypassing the PLL for startup of the system without external clock. Furthermore it can be used as independent clock source for the watchdog module or even as system clock

source during normal operation. While in prescaler mode this clock is automatically used as emergency clock if the external clock failure is detected.

Clock adjustment is required to reach desired level of f_{OFI} precision. The backup clock source provides two adjustment procedures:

- loading of adjustment value during start-up
- continuos adjustment using the high-precision f_{STDBY} clock as reference

11.6.6 Main PLL

The main PLL converts a low-frequency external clock signal to a high-speed internal clock. The PLL also has fail-safe logic that detects degenerative external clock behavior such as abnormal frequency deviations or a total loss of the external clock. The PLL triggers autonomously emergency action if it loses its lock on the external clock and switches to the Fast Internal Backup Clock.

This module is a phase locked loop for integer frequency synthesis. It allows the use of input and output frequencies of a wide range by varying the different divider factors.

11.6.6.1 Features

- VCO lock detection
- 4-bit input divider **P:** (divide by PDIV+1)
- 7-bit feedback divider **N**: (multiply by NDIV+1)
- 7-bit output dividers **K1 or K2:** (divide by KxDIV+1)
- Oscillator Watchdog
  - Detecting too low input frequencies
  - Detecting too high input frequencies
  - Spike detection for the OSC input frequency
- Different operating modes
  - Bypass Mode
  - Prescaler Mode
  - Normal Mode
- VCO Power Down
- PLL Power Down
- Glitch less switching between K-Dividers
- Switching between Normal Mode and Prescaler Mode

11.6.6.2 System PLL Functional Description

The PLL consists of a Voltage Controlled Oscillator (VCO) with a feedback path. A divider in the feedback path (N-Divider) divides the VCO frequency down. The resulting frequency is then compared with the externally provided and divided frequency (P-Divider). The phase detection logic determines the difference between the two clocks

and accordingly controls the frequency of the VCO (f_{VCO}). A PLL lock detection unit monitors and signals this condition. The phase detection logic continues to monitor the two clocks and adjusts the VCO clock if required.

The following figure shows the PLL block structure.

Clock Source Control

The input clock for the PLL f_{OSC} can be one of the following two clock sources:

- The internal generated fast clock f_{OFI}
- The clock f_{OHP} sourced by the crystal oscillator OSC_HP

The PLL clock f_{PLL} is generated from the input clock in one of two software selectable operation modes:

- Normal Mode, using VCO output clock
- Prescaler Mode, using input clock directly

The PLL output clock f_{PLL} is derived from either

- VCO clock divided by the K2-Divider, Normal Mode
- external oscillator clock divided by the K1-Divider, Prescaler Mode
- backup clock divided by the K1-Divider, Prescaler Mode

The PLL clock f_{PLL} is generated in emergency from one of two sources:

- Free running VCO if Emergency entered from Normal Mode of PLL
- Backup Clock if emergency entered from Prescaler Mode of PLL

Configuration and Operation of the Normal Mode

In Normal Mode, the PLL is running at the frequency f_{OSC} and f_{PLL} is divided down by a factor P, multiplied by a factor N and then divided down by a factor K2.

The output frequency is given by:

(11.1)

$$f_{PLL} = \frac{N}{P \cdot K_2} \cdot f_{OSC}$$

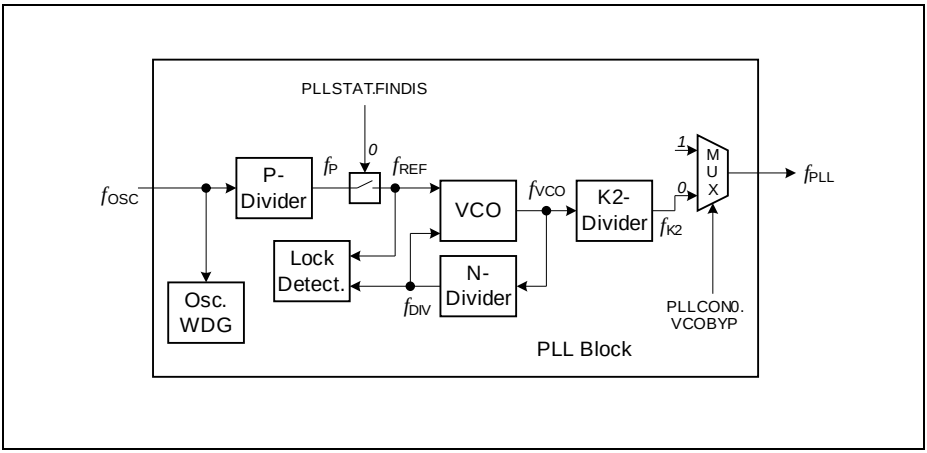


Figure 11-30 PLL Normal Mode

It is strongly recommended to apply even value of P and K2 Divider parameters in order to minimize PLL output clock jitter effect. Please find PLL configuration examples values in **Table 11-6**.

Table 11-6 PLL example configuration values

Target Frequency of f_{PLL} [MHz]	External Crystal Frequency [MHz]	P Parameter	N Parameter	K2 Parameter
80	8	1	40	4
	12	3	80	4
	16	1	20	4

*Note: The values of P, N and K2 configuration parameters specified in **Table 11-6** should must decremented by one before programmed into corresponding registers.*

The Normal Mode is selected by the following settings

- Register Setting
  - PLLCON0.VCOBYP = 0
  - PLLCON0.FINDIS = 0

The Normal Mode is entered when the following requirements are all together valid:

- Register Values
  - PLLCON0.FINDIS = 0
  - PLLSTAT.VCOBYST = 0
  - PLLSTAT.VCOLOCK = 1

– PLLSTAT.PLLLV = 1
– PLLSTAT.PLLLH = 1

Operation on the Normal Mode does require an input clock frequency of f_{OSC}. Therefore it is recommended to check and monitor if an input frequency f_{OSC} is available at all by checking PLLSTAT.PLLLV. For a better monitoring also the upper frequency can be monitored via PLLSTAT.PLLHV.

For the Normal Mode there is the following requirement regarding the frequency of f_{OSC}.

A modification of the two dividers P and N has a direct influence to the VCO frequency and lead to a loss of the VCO Lock status. A modification of the K2-divider has no impact on the VCO Lock status.

When the frequency of the Normal Mode should be modified or entered the following sequence needs to be followed:

- Configure and enter Prescaler Mode
- Disable NMI trap generation for the VCO Lock
- Configure Normal Mode
- Wait for a positive VCO Lock status (PLLSTAT.VCOLOCK = 1).
- Switch to Normal Mode by clearing PLLCON.VCOBYP

The Normal Mode is entered when the status bit PLLSTAT.VCOBYST is cleared.

When the Normal Mode is entered, the NMI status flag for the VCO Lock trap should be cleared and then enabled again. The intended PLL output target frequency can now be configured by changing only the K2-Divider. This can result in multiple changes of the K2-Divider to avoid to big frequency changes. Between the update of two K2-Divider values 6 cycles of f_{PLL} should be waited. For ramping up PLL output frequency in Normal Mode the following steps are required:

- The first target frequency of the Normal Mode should be selected in a way that it matches or is only slightly higher as the one used in the Prescaler Mode. This avoids big changes in the system operation frequency and therefore power consumption when switching later from Prescaler Mode to Normal Mode.
- Selecting P and N in a way that f_{VCO} is in the upper area of its allowed values leads to a slightly increased power consumption but to a slightly reduced jitter
- Selecting P and N in a way that f_{VCO} is in the lower area of its allowed values leads to a slightly reduced power consumption but to a slightly increased jitter
- It is recommended to reset the f_{VCO} Lock detection (PLLCON0.RESLD = 1) after the new values of the dividers are configured to get a defined VCO lock check time.

Depending on the selected divider value of the K2-Divider the duty cycle of the clock is selected. This can have an impact for the operation with an external communication interface. The duty cycles values for the different K2-divider values are defined in the Data Sheet.

PLL VCO Lock Detection

The PLL has a lock detection that supervises the VCO part of the PLL in order to differentiate between stable and unstable VCO circuit behavior. The lock detector marks the VCO circuit and therefore the output f_{VCO} of the VCO as instable if the two inputs f_{REF} and f_{DIV} differ too much. Changes in one or both input frequencies below a level are not marked by a loss of lock because the VCO can handle such small changes without any problem for the system.

PLL VCO Loss-of-Lock Event

The PLL may become unlocked, caused by a break of the crystal or the external clock line. In such a case, an NMI trap is generated if it were enabled. Additionally, the OSC clock input f_{OSC} is disconnected from the PLL VCO to avoid unstable operation due to noise or sporadic clock pulses coming from the oscillator circuit. Without a clock input f_{OSC}, the PLL gradually slows down to its VCO base frequency and remains there. This default feature can be disabled by setting bit PLLCON0.OSCDISCDIS. If this bit is set the OSC clock remains connected to the VCO.

11.6.6.3 Configuration and Operation of the Prescaler Mode

In Prescaler Mode, the PLL is running at the external frequency f_{OSC} and f_{PLL1} is derived from f_{OSC} only by the K1-Divider.

The output frequency is given by

$$f_{PLL} = \frac{f_{OSC}}{K_1} \quad (11.2)$$

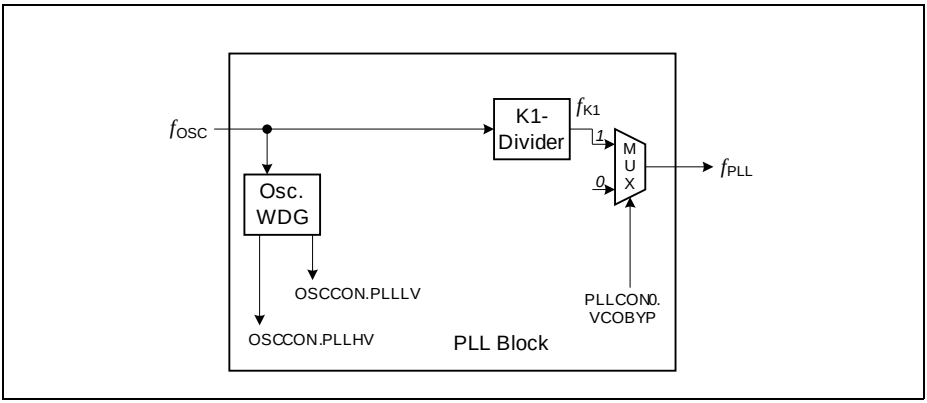


Figure 11-31 PLL Prescaler Mode Diagram

The Prescaler Mode is selected by the following settings

- PLLCON0.VCOBYP = 1

The Prescaler Mode is entered when the following requirements are all together valid:

- PLLSTAT.VCOBYST = 1
- PLLSAT.PLLLV = 1

Operation on the Prescaler Mode does require an input clock frequency of f_{OSC}. Therefore it is recommended to check and monitor if an input frequency f_{OSC} is available at all by checking PLLSAT.PLLLV. For a better monitoring also the upper frequency can be monitored via PLLSAT.PLLHV.

For the Prescaler Mode there are no requirements regarding the frequency of f_{OSC}.

The system operation frequency is controlled in the Prescaler Mode by the value of the K1-Divider. When the value of PLLCON1.DIV was changed the next update of this value should not be done before bit PLLSTAT.K1RDY is set.

Depending on the selected divider value of the K1-Divider the duty cycle of the clock is selected. This can have an impact for the operation with an external communication interface. The duty cycles values for the different K1-divider values are defined in the Data Sheet.

The Prescaler Mode is requested from the Normal Mode by setting bit PLLCON.VCOBYP. The Prescaler Mode is entered when the status bit PLLSTAT.VCOBYST is set. Before the Prescaler Mode is requested the K1-Divider should be configured with a value generating a PLL output frequency f_{PLL} that matches the one generated by the Normal Mode as much as possible. In this way the frequency change resulting out of the mode change is reduced to a minimum.

The Prescaler Mode is requested to be left by clearing bit PLLCON.VCOBYP. The Prescaler Mode is left when the status bit PLLSTAT.VCOBYST is cleared.

11.6.6.4 Bypass Mode

The bypass mode is used only for testing purposes. In Bypass Mode the input clock f_{OSC} is directly connected to the PLL output f_{PLL}.

The output frequency is given by:

(11.3)

$$f_{PLL} = f_{OSC}$$

11.6.6.5 System Oscillator Watchdog (OSC_WDG)

The oscillator watchdog monitors the incoming clock frequency f_{OSC} from the OSC_HP oscillator. A stable and defined input frequency is a mandatory requirement for operation

in both Prescaler Mode and Normal Mode. In addition for the Normal Mode it is required that the input frequency f_{OSC} is in a certain frequency range to obtain a stable master clock from the VCO part.

The expected input frequency is selected via the bit field **OSCHPCTRL**.OSCVAL. The OSC_WDG checks for spikes, too low frequencies, and for too high frequencies.

The frequency that is monitored is f_{OSCREF} which is derived from f_{OSC}.

(11.4)

$$f_{OSCREF} = \frac{f_{OSC}}{OSCVAL + 1}$$

The divider value **OSCHPCTRL**.OSCVAL has to be selected in a way that f_{OSCREF} is 2.5 MHz.

Note: f_{OSCREF} has to be within the range of 2 MHz to 3 MHz and should be as close as possible to 2.5 MHz.

The monitored frequency is too low if it is below 1.25 MHz and too high if it is above 7.5 MHz. This leads to the following two conditions:

- Too low: f_{OSCREF} < 1.25 MHz × (**OSCHPCTRL**.OSCVAL+1)
- Too high: f_{OSCREF} > 7.5 MHz × (**OSCHPCTRL**.OSCVAL+1)

Before configuring the OSC_WDG function all the trap options should be disabled in order to avoid unintended traps. Thereafter the value of **OSCHPCTRL**.OSCVAL can be changed. Then the OSC_WDG should be reset by setting **PLLCON0**.OSCVAL. This requests the start of OSC_WDG monitoring with the new configuration. When the expected positive monitoring results of **PLLSTAT**.PLLLV and / or **PLLSTAT**.PLLHV are set the input frequency is within the expected range. As setting **PLLCON0**.OSCVAL clears all three bits **PLLSTAT**.PLLSP, **PLLSTAT**.PLLLV, and **PLLSTAT**.PLLHV all three trap status flags will be set. Therefore all three flags should be cleared before the trap generation is enabled again. The trap disabling-clearing-enabling sequence should also be used if only bit **PLLCON0**.OSCVAL is set without any modification of **OSCHPCTRL**.OSCVAL.

11.6.6.6 VCO Power Down Mode

The PLL offers a VCO Power Down Mode. This mode can be entered to save power within the PLL. The VCO Power Down Mode is entered by setting bit **PLLCON0**.VCOPWD. While the PLL is in VCO Power Down Mode only the Prescaler Mode is operable. Please note that selecting the VCO Power Down Mode does not automatically switch to the Prescaler Mode. So before the VCO Power Down Mode is entered the Prescaler Mode must be active.

11.6.6.7 PLL Power Down Mode

The PLL offers a Power Down Mode. This mode can be entered to save power if the PLL is not required. The Power Down Mode is entered by setting bit **PLLCON0**.PLLPWD. While the PLL is in Power Down Mode no PLL output frequency is generated.

11.6.7 Internally Generated System Clock Calibration

The internal Backup Clock Source by default generates output clock signal of a frequency parameters as specified in the Data Sheet, referred to as uncalibrated OSC_FI clock. In order to improve accuracy of the Backup Clock Source clock calibration mechanisms are available. For details please refer to the following sub-chapters.

11.6.7.1 Factory Calibration

The Factory Calibration mechanism allows the Backup Clock Source clock output adjustment to parameters specified in the Data Sheet, referred to as OSC_FI factory calibrated clock. The factory calibration can be enabled with user software via bit field FOTR of the **PLLCON0** register. Enabling of this calibration has an immediate effect on the clock output.

11.6.7.2 Automatic Calibration

The Automatic Calibration mechanism enables Backup Clock Source clock output continuous adjustment based on the relation to a high precision reference clock f_{STDBY}. This calibration brings the clock output parameters to the level specified in the Data Sheet as the OSC_FI automatically calibrated clock. The automatic calibration can be enabled with user software via bit field AOTREN of the **PLLCON0** register. It is required that the reference clock f_{STDBY} generated in the Hibernate domain is activated prior to enabling this method of clock calibration.

11.6.7.3 Alternative Internal Clock Calibration

An alternative system clock calibration can be performed by programming of the system PLL with register values reflecting individual chip calibration characteristics determined by the **CLKCALCONST**.CALIBCONST register value. This way of calibration allows to achieve clock of frequency variation comparable with the factory calibration enabled Internal Backup Clock Source, but with significantly lower output clock jitter as defined for f_{OFI} untrimmed oscillator. For more details please refer to Data Sheet.

The formula describing dependencies between calibration constant and PLL configuration settings that allow to program desired output frequency is provided in **Equation (11.5)**

(11.5)

$$f_{PLL} = 24 \times \left(\frac{CALIBCONST \times 488 + 11651}{11146}\right) \times \left(\frac{N}{P \times K2}\right) [MHz]$$

It is strongly recommended to apply even value of P and K2 parameters in order to minimize PLL output clock jitter effect. Please find PLL configuration examples for different calibration constant *CALIBCONST* values in **Table 11-7**.

Table 11-7 PLL example configuration values

Target Frequency of f_{PLL} [MHz]	TRIM Constant	P Parameter	N Parameter	K2 Parameter
80	0	6	77	4
	1	6	74	4
	2	6	71	4
	3	6	69	4
	4	6	66	4
	5	6	64	4
	6	6	62	4
	7	6	60	4
	8	6	58	4
	9	6	56	4
	10	6	54	4
	11	6	53	4
	12	6	51	4
	13	6	50	4
	14	8	97	6
	15	8	63	4

*Note: The values of P, N and K2 configuration parameters specified in **Table 11-7** should must decremented by one before programmed into corresponding registers.*

11.6.8 USB PLL

The USB PLL serves the special purpose to provide an accurate clock for USB 2.0 full-speed operation. The basic functionality of the USB PLL is similar to the main PLL (see **Figure 11-32**).

Configuration and Operation

The PLLUSB is running at the frequency f_{OHP} and f_{PLLUSB} is divided down by a factor P, multiplied by a factor N and then divided down by the factor of 2.

The output frequency is given by:

(11.6)

$$f_{PLLUSB} = \frac{N}{P \cdot 2} \cdot f_{OHP}$$

Operation of the PLLUSB require an input clock frequency of f_{OHP}.

The following requirement must be fulfilled regarding the frequency of f_{OHP} (see).

A modification of the two dividers P and N has a direct influence to the VCO frequency and lead to a loss of the VCO Lock status.

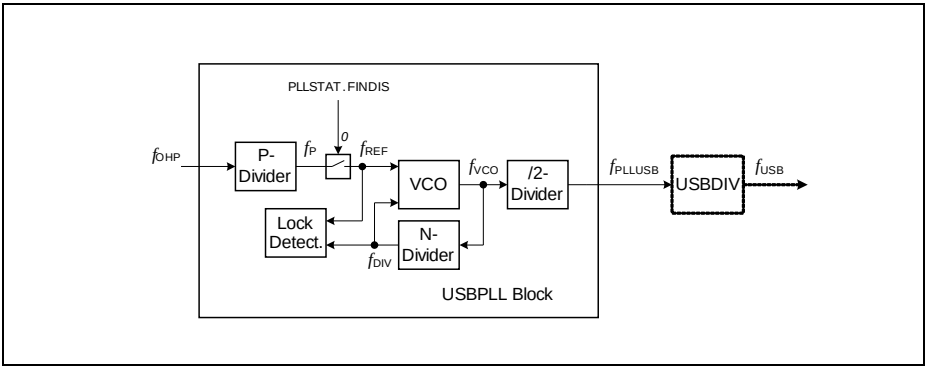


Figure 11-32 PLLUSB Block Diagram

It is strongly recommended to apply even value of P and USB Clock Divider (referred to as USBDIV divider) parameters in order to minimize the PLL output clock jitter effect. Please find PLLUSB configuration examples values in **Table 11-8**.

Table 11-8 PLLUSB example configuration values

Crystal frequency [MHz]	P Parameter	N Parameter	USB Divider Parameter
8	2	96	4
12	2	64	4
16	2	48	4

*Note: The values of P, N and USB Divider configuration parameters specified in **Table 11-8** should must decremented by one before programmed into corresponding registers.*

Note: Re-configuration of the P-Divider before the USBPLL has locked must be avoided.

11.6.9 Ultra Low Power Oscillator

The ultra low power oscillator is providing a real time clock source of 32.768 kHz when paired with an external crystal. It operates in the supply voltage range of the Hibernate power domain. The crystal pads are always powered by V_{BAT} or V_{DDP}.

The precise and stable clock can be propagated to Core domain as f_{STDBY} and used for continuos adjustments the fast internal backup clock source f_{OFI}.

11.6.9.1 OSC_ULP Oscillator Watchdog (ULPWDG)

The slow oscillator watchdog monitors the incoming clock frequency f_{ULP} from OSC_ULP. A reliable clock is required in the Hibernate domain in Hibernate state in order to perform system wake-up upon occurrence of configured events. In order to ensure that a clock watchdog is required to continuously monitor the enabled clock sources.

In case of external crystal failure the clock source switches automatically to the Internal Slow Clock Source generating f_{OSI}.

11.6.10 Internal Slow Clock Source

The slow internal clock source provides a clock f_{OSI} of 32.768 kHz. This clock can be used as independent clock source for WDT module and as the clock for periodic wake-up events in power saving modes and for continuos adjustments the fast internal backup clock source f_{OFI}.

11.6.11 Clock Gating Control

The clock to peripherals can be individually gated and parts of the system stopped by these means.

Clock gating in Sleep and Deep Sleep modes

Global power management related to clock generation and selection is supported for the sleep modes of the system. The user has full control on the clock configuration for these modes. The registers **SLEEPCR** and **DSLEEPCR** control which clocks should remain active and which are to be gated by entering the corresponding Sleep or Deep Sleep mode. Furthermore the system clock can be switched to a slow standby clock and the PLLs put into power-down mode in Deep Sleep mode.

11.7 Debug Behavior

The SCU module does not get affected with the HALTED signal from CPU upon debug activities performed using external debug probe.

11.8 Power, Reset and Clock

The SCU module consists of sub-modules that interact with different power, clock and reset domains. Some of the sub-modules are controlled via dedicated interfaces across the power, clock and reset boundries.The sub-modules are considered parts of the SCU in the functional sense therefore the complete SCU module is considered a multi domain circuit.

Power domains

Power domains get separated with appropriate power separation cells.

- Pad domain supplied with V_{DDP} voltage
- Analog domain supplied with V_{DDA} voltage
- Core domain supplied with V_{DDC} voltage
- Hibernate domain supplied with V_{BAT} (optionally with a battery or capacitor) or V_{DDP} voltage

Clock domains

All cross-domain interfaces of SCU implement signal synchronization.

- SCU clock in the core domain is f_{SYS}
- Hibernate Control Unit (HCU) clock is f_{OSI} or f_{ULP} (32.768 kHz)
- The Register Mirror Interface of HCU and RTC clock is clocked with f_{STDBY} (32.768 kHz) and f_{SYS} clocks

Reset domains

All reset signals get synchronized to the respective clocks (please refer to **“Reset Control” on Page 11-33** chapter for more details)

- System Reset ($\overline{\text{SYSRESET}}$) resets the core logic and can be triggered from various sources.

- Power-on Reset ($\overline{\text{PORESET}}$) resets analog modules and contributes in generation of the System Reset. The reset is controlled from internal power validation circuit or driven externally via the bi-directional $\overline{\text{PORST}}$ pin.
- Standby Reset ($\overline{\text{STDBYRESET}}$) resets HCU part of SCU. It is triggered by power-up sequence of Hibernate domain and is not affected by power-up sequence of the Core domain. It can also be controlled with software access to **RSTCLR** and **RSTSET** registers.
- Debug Reset ($\overline{\text{DBGRESET}}$) is used in debug with an external debug probe.

11.9 Initialization and System Dependencies

The initialization sequence of the XMC4100 / XMC4200 is a process taking place before user application software takes control of the system and is comprising of two major phases (see **Figure 11-33**), split in several distinctive steps:

Hardware Controlled Initialization Phase

The hardware controlled initialization phase gets performed automatically after power up of the microcontroller. This part is generic and it ensures basic configuration common to most applications. The hardware setup needs to ensure fulfillment of requirements specified in Data Sheet in order to enable reliable start up of the microcontroller before control is handed over to the user software. The sequence where boot code gets executed is considered a part of the hardware controlled phase of the initialization sequence.

For details of the setup requirements please refer to Data Sheet.

Software Controlled Initialization Phase

The software controlled initialization phase is the part of the complete start-up sequence where the application specific configuration gets applied with user software. It involves several steps that are critical for proper operation of the microcontroller in the application context and may also involve some optional configuration actions in order to improve system performance and stability in the application context.

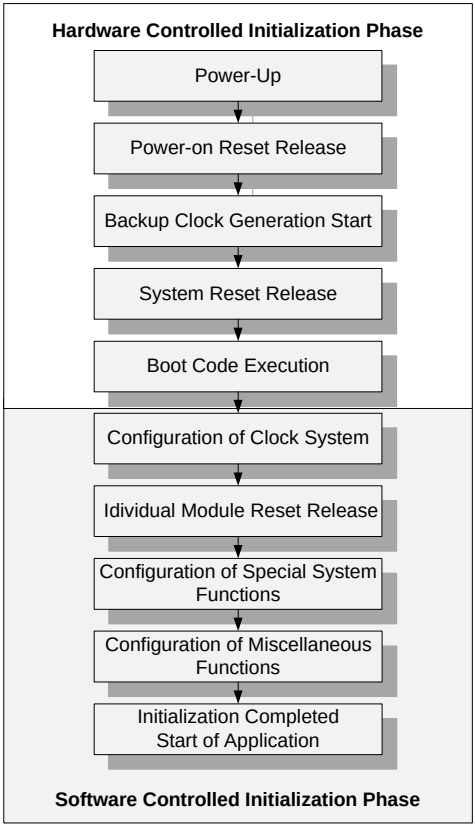


Figure 11-33 Initialization sequence

A more detailed description of the configuration steps is available in the following sub-chapters.

11.9.1 Power-Up

Power up of the microcontroller gets performed by applying VDDP and VDDA supply. Internal voltage generation in the EVR module gets activated automatically after VDDP has been applied (for details of the supply requirements please refer to Data Sheet), unless the microcontroller is in Internally Controlled Hibernate mode.

In order to power up the microcontroller when it remains in the Internally Controlled Hibernate it is required to trigger a wake-up event to which it has been configured to react.

The XMC4100 / XMC4200 will always start after the VDDP has been removed and re-applied, regardless of hibernate mode was entered before and regardless of the programmed wake-up trigger.

11.9.2 Power-on Reset Release

The XMC4100 / XMC4200 implements bi-directional pin $\overline{\text{PORST}}$ for the Power-on Reset $\overline{\text{PORESET}}$ control. The internal Power-on Reset generation is based on the supply and core voltage validation.

The $\overline{\text{PORST}}$ pin may be used either to control the $\overline{\text{PORESET}}$ from an external source in the system or to control the reset of external components from the XMC4100 / XMC4200. The $\overline{\text{PORST}}$ function, implemented as open drain driver, allows to share the pin between multiple devices in the system.

An example of the system where XMC4100 / XMC4200 may act as the reset control master is shown in **Figure 11-34**. Any of the devices capable to drive low level of the reset signal is potentially able to assert reset of the system. Release of the reset is effectively performed when devices with output driving capability are not driving low level and the signal is driven high via the pull-up resistance. The driving strength of the pull-up resistance is required to ensure fast release of $\overline{\text{PORST}}$ pin and effectively also fast release of $\overline{\text{PORESET}}$ reset (for details on Power Sequencing please refer to Data Sheet).

Release of the $\overline{\text{PORESET}}$ of XMC4100 / XMC4200 results in start of Backup Clock generation.

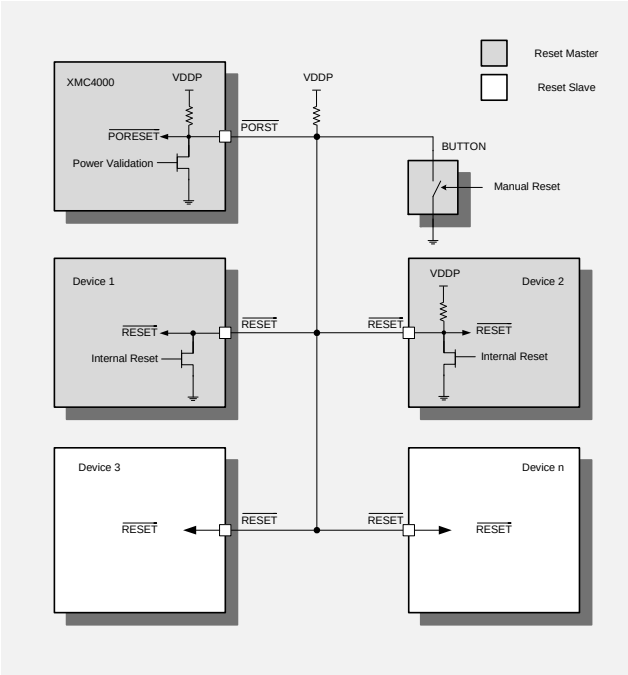


Figure 11-34 System Level Power On Reset Control

11.9.3 System Reset Release

Release of the $\overline{\text{PORESET}}$ and start of the Backup Clock generation results in automatic System Reset $\overline{\text{SYSRESET}}$ release and start of boot code execution.

The System Reset $\overline{\text{SYSRESET}}$ can be triggered from various sources like software controlled CPU reset register, watchdog time-out-triggered reset or a system trap triggered reset. For more details on Reset Control details please refer to **“Reset Control” on Page 11-33**

The cause of the last reset gets automatically stored in the **RSTSTAT** register and can be check by user software to determine the state of the system and for debug purpose. The reset status in the **RSTSTAT** register shall be reset with **RSTCLR** register after each startup in order to ensure consistent source indication after the next reset.

After $\overline{\text{SYSRESET}}$ release a number of modules clocked with f_{PERIPH} still remain in reset state. Reset release of these modules needs to be released with **PRCLR0**, **PRCLR1**, and **PRCLR2**registers which by default keep the peripheral resets active. It is

recommended to keep the unused modules in reset state in order to reduce power consumption. It is also highly recommended to ensure that clock of the corresponding modules are active before individual peripheral reset release.

11.9.4 Clock System Setup

The following system clocking modes are supported:

- PLL Normal
- PLL Prescaler
- Backup Clock

The default clock signal available after power-up is the internally generated Backup Clock f_{OFI}. The user software starts execution of code clocked with the Backup Clock and can change the clock at any point of time applying a system clock configuration sequence. For details of the clock structure please refer to **"Clock System Overview" on Page 11-38**.

The flowchart in **Figure 11-35** illustrates the recommended system clock configuration sequence. It is strongly recommended to follow the steps in order to ensure proper initialization of the PLL and power sequencing within specified limits (for details on Power Sequencing please refer to Data Sheet). Each of the steps depicted in the diagram may require a sequence of register access actions.

The configuration of the clock system may involve various actions like:

- internal clock trimming configurations, election between Factory or Automatic trimming of the Backup Clock f_{OFI} using **PLLCON0** register
- calibration of the PLL, calculation of the optimal PLL settings using the **Equation (11.5)** on **Page 11-53**
- enabling of external crystal oscillator or direct clock input, configuration of the external clock watchdog with **OSCHPCTRL** register, according to **Equation (11.4)** on **Page 11-51**, selection of the external crystal oscillator or direct clock f_{OSC} using **OSCHPCTRL**
- powering up of the System PLL in **PLLCON0** register
- locking-up of the System PLL, configuration of the System PLL using **PLLCON0**, **PLLCON1** and **PLLCON2** registers
- configuration of clock dividers, switching the system clock to the selected source (please refer to **Figure 11-26**)

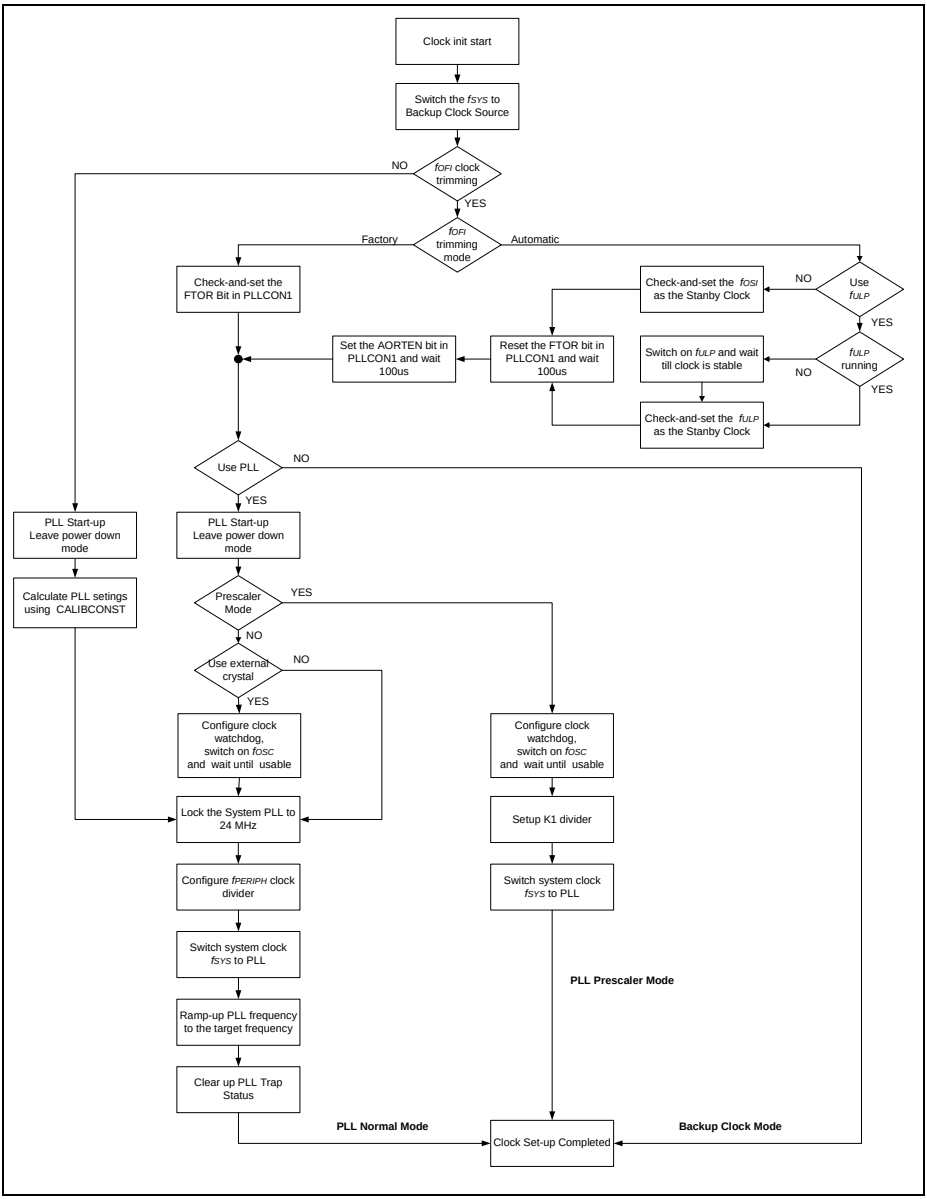


Figure 11-35 Clock initialization sequence

After reset release the system is clocked with a clock derived from the Backup Clock source. If a PLL output clock is required as the system clock source then it is necessary to initialize the respective PLL with a software routine. For details please refer to PLL section in **"Main PLL" on Page 11-45**.

The system relevant and module clocks are configured with the dedicated registers **SYSCLKCR**, **CPUCLKCR** and **PBCLKCR**, or, alternatively with the **MLINKCLKCR** register combining functionalities of several clock control relevant registers.

Some peripheral clocks require explicit enable with **CLKSET** register.

In order to reduce power consumption of the microcontroller it is recommended to activate individual clock gating on modules that are not used in the application with the registers **CGATSET0**, **CGATSET1** and **CGATSET2**.

11.9.5 Configuration of Special System Functions

Special system functions are may be required to perform actions that improve system stability and robustness. The following special system functions or modules require initialization before the actual user application starts:

- Memory Parity Protection
- Supply Voltage Brown-out Detection
- Initialization of the Hibernate Domain
- Watchdog Timer (WDT)
- Real Time Clock (RTC)
- System Trap Initialization

Memory Parity Protection

Memory Parity Protection is performed using memory parity check. The parity check can be enabled individually for each instance of memory with **PEEN** register. A trap generation can be individually enabled with **PETE** register in order to get the trap flag reflected in **TRAPRAW** register or to generate System Reset if enabled in **PERSTEN** register.

For details of the Memory Parity Protection please refer to the **"Memory Parity Protection" on Page 11-7**.

Brown Out Detection

Brown out detection mechanism allow active monitoring of the supply voltage and a corrective reaction in case the voltage level is below a programmed threshold. A trap request will be flagged if a programmed condition is detected. For details please refer to **"Supply Voltage Brown-out Detection" on Page 11-23**

Initialization of the Hibernate Domain

Hibernate Control logic and the RTC module needs to be activated with **PWRSET** register before it can be used. This initialization needs to be performed only once after the V_{BAT} has been applied and it stays enabled if V_{BAT} remains applied. After power off of the main supply of the chip i.e. VDDP, the hibernate domain will remain intact if V_{BAT} is still supplied. For details of hibernate control please refer to **"Hibernate Control" on Page 11-24**. For details of RTC module control please refer to RTC chapter.

Watchdog Timer

The Watchdog Timer requires a clock source selection and activation. It is highly recommended to use reliable clock source, preferably independent from the system clock source in order to ensure corrective action in case of a system failure which will bring the microcontroller into a safe operation state. For more details please refer to WDT chapter.For details of the WDT module configuration please refer to the "Initialization and Control Sequence" section of the WDT chapter.

Real Time Clock

If the real time clock is required then the initialization must take place after Hibernate domain has been activated. For details of the RTC module configuration please refer to the "Initialization and Control Sequence" section of the RTC chapter.

System Trap Initialization

System Traps are by default disabled and need to be enabled with user software before used. Any active trap flag will be reflected in the **TRAPRAW** register. In order to enable an NMI interrupt generation it needs to be unmasked in **TRAPDIS** register.

For details of the System Trap configuration please refer to the **"Trap Generation" on Page 11-9**.

11.9.6 Configuration of Miscellaneous Functions

A number of miscellaneous functions may be used as a part of the user application, or, e.g. by an Operating System. The following functions are available and require configuration before used:

- Power Saving modes
- Die Temperature Sensor (DTS)
- Out of Range Comparators for analog I/Os

Power Saving modes

Different power modes like Sleep, Deep Sleep and Hibernate mode require configuration of their functions and configuration of wake-up triggers prior to entering the modes. For

details of the available modes and configuration please refer to **“Power Management” on Page 11-16**

Die Temperature Sensor

The Die Temperature Sensor allows to perform temperature measurements of the die. The module needs to be enabled with **DTSCON** register before used. For details please refer to **“Die Temperature Measurement” on Page 11-10**

Out of Range Comparators

The out of range comparator serves the purpose of overvoltage monitoring for analog input pins of the chip. This functionality can be enabled with registers **G0ORCEN** and **G1ORCEN**.

11.10 Registers

This section describes the registers of SCU. Most of the registers are reset $\overline{\text{SYSRESET}}$ reset signal but some of the registers can be reset only with PORST reset.

Table 11-9 Base Addresses of sub-sections of SCU registers

Short Name	Description	Offset Addr.[1)]
GCU Registers	Offset address of General Control Unit	0000_H
PCU Registers	Offset address of Power Control Unit	0200_H
HCU Registers	Offset address of Hibernate Control Unit	0300_H
RCU Registers	Offset address of Reset Control Unit	0400_H
CCU Registers	Offset address of Clock Control Unit	0600_H

1) The absolute register address is calculated as follows:
Module Base Address + Sub-Module Offset Address (shown in this column) + Register Offset Address

Following access result an AHB error response:

- Read or write access to undefined address
- Write access in user mode to registers which allow only privileged mode write
- Write access to read-only registers
- Write access to startup protected registers

Table 11-10 Registers Address Space

Module	Base Address	End Address	Note
SCU	5000 4000_H	5000 $7FFF_H$	System Control Unit Registers

Table 11-11 Registers Overview

Short Name	Register Long Name	Offset Addr. 1)	Access Mode		Description
			Read	Write	
General SCU Registers					
GCU Registers					
ID	Module Identification Register	0000_H	U, PV	BE	Page 11-73
IDCHIP	Chip ID	0004_H	U, PV	BE	Page 11-74

Table 11-11 Registers Overview (cont'd)

Short Name	Register Long Name	Offset Addr. [1]	Access Mode		Description
			Read	Write	
IDMANUF	Manufactory ID	0008_H	U, PV	BE	Page 11-74
STCON	Start-up Control	0010_H	U, PV	PV	Page 11-75
GPR0	General Purpose Register 0	$002C_H$	U, PV	PV	Page 11-76
GPR1	General Purpose Register 1	0030_H	U, PV	PV	Page 11-76
USBHOSTDET	USB HOST Detection	$003C_H$	U, PV	PV	Page 11-77
CCUCON	CCUx Global Start Control Register	$004C_H$	U, PV	U	Page 11-78
SRSTAT	Service Request Status	0074_H	U, PV	U, PV	Page 11-78
SRRAW	RAW Service Request Status	0078_H	U, PV	BE	Page 11-81
SRMSK	Service Request Mask	$007C_H$	U, PV	PV	Page 11-84
SRCLR	Service Request Clear	0080_H	nBE	PV	Page 11-87
SRSET	Service Request Set	0084_H	nBE	PV	Page 11-90
NMIREQEN	Enable Promoting Events to NMI Request	0088_H	U, PV	PV	Page 11-92
DTSCON	DTS Control	$008C_H$	U, PV	PV	Page 11-94
DTSSTAT	DTS Status	0090_H	U, PV	BE	Page 11-95
G0ORCEN	Out-Of-Range Comparator Enable Register	$00A0_H$	U, PV	U, PV	Page 11-96
G1ORCEN	Out-Of-Range Comparator Enable Register	$00A4_H$	U, PV	U, PV	Page 11-97
DTEMPLIM	Die Temperature Limit Register	$00A8_H$	U, PV	U, PV	Page 11-98
DTEMPALARM	Die Temperature Limit Alarm	$00AC_H$	U, PV	U, PV	Page 11-99
MIRRSTS	Mirror Update Status Register	$00C4_H$	U, PV	BE	Page 11-100

Table 11-11 Registers Overview (cont'd)

Short Name	Register Long Name	Offset Addr. [1]	Access Mode		Description
			Read	Write	
RMACR	Retention Memory Access Control Register	$00C8_H$	U, PV	U, PV	Page 11-102
RMADATA	Retention Memory Access Data Register	$00CC_H$	U, PV	U, PV	Page 11-103
MIRRALLRSTAT	Mirror All Status Register	$00D0_H$	U, PV	U, PV	Page 11-104
MIRRALLREQ	Mirror All Request Register	$00D4_H$	U, PV	U, PV	Page 11-104
PEEN	Parity Error Enable Register	$013C_H$	U, PV	PV	Page 11-105
MCHKCON	Memory Checking Control Register	0140_H	U, PV	PV	Page 11-106
PETE	Parity Error Trap Enable Register	0144_H	U, PV	PV	Page 11-107
PERSTEN	Reset upon Parity Error Enable Register	0148_H	U, PV	PV	Page 11-109
PEFLAG	Parity Error Control Register	0150_H	U, PV	PV	Page 11-109
PMTPR	Parity Memory Test Pattern Register	0154_H	U, PV	PV	Page 11-111
PMTSR	Parity Memory Test Select Register	0158_H	U, PV	PV	Page 11-112
TRAPSTAT	Trap Status Register	0160_H	U, PV	BE	Page 11-113
TRAPRAW	Trap Raw Status Register	0164_H	U, PV	BE	Page 11-115
TRAPDIS	Trap Mask Register	0168_H	U, PV	PV	Page 11-117
TRAPCLR	Trap Clear Register	$016C_H$	nBE	PV	Page 11-118
TRAPSET	Trap Set Register	0170_H	nBE	PV	Page 11-119
PCU Registers					
PWRSTAT	Power Status Register	0000_H	U, PV		Page 11-122
PWRSET	Power Set Control Register	0004_H	nBE	PV	Page 11-123

Table 11-11 Registers Overview (cont'd)

Short Name	Register Long Name	Offset Addr. [1]	Access Mode		Description
			Read	Write	
PWRCLR	Power Clear Control Register	0008_H	nBE	PV	Page 11-123
EVRSTAT	EVR Status Register	0010_H	U, PV	BE	Page 11-124
EVRVADCSTAT	EVR VADC Status Register	0014_H	U, PV	BE	Page 11-125
PWRMON	Power Monitor Value	$002C_H$	U, PV	PV	Page 11-126
HCU Registers					
HDSTAT	Hibernate Domain Status Register	0000_H	U, PV	PV	Page 11-127
HDCLR	Hibernate Domain Status Clear Register	0004_H	U, PV	PV	Page 11-129
HDSET	Hibernate Domain Status Set Register	0008_H	U, PV	PV	Page 11-130
HDCR	Hibernate Domain Control Register	$000C_H$	U, PV	PV	Page 11-132
OSCSICTRL	Internal 32.768 kHz Clock Source Control Register	0014_H	U, PV	PV	Page 11-134
OSCULSTAT	OSC_ULP Status Register	0018_H	U, PV	BE	Page 11-135
OSCULCTRL	OSC_ULP Control Register	$001C_H$	U, PV	PV	Page 11-136
LPACCONF	LPAC Comparator Configuration Register	0020_H	U, PV	PV	Page 11-137
LPACTH0	LPAC Comparator Threshold Register 0	0024_H	U, PV	PV	Page 11-138
LPACTH1	LPAC Comparator Threshold Register 1	0028_H	U, PV	PV	Page 11-139
LPACST	Low Power Analog Comparator State Register	$002C_H$	U, PV	PV	Page 11-140

Table 11-11 Registers Overview (cont'd)

Short Name	Register Long Name	Offset Addr. [1]	Access Mode		Description
			Read	Write	
LPACCLR	Low Power Analog Comparator Control Clear Register	0030_H	U, PV	PV	Page 11-141
LPACSET	Low Power Analog Comparator Control Set Register	0034_H	U, PV	PV	Page 11-142
HINTST	Hibernate Internal Control State Register	0038_H	U, PV	PV	Page 11-144
HINTCLR	Hibernate Internal Control Clear Register	$003C_H$	U, PV	PV	Page 11-145
HINTSET	Hibernate Internal Control Set Register	0040_H	U, PV	PV	Page 11-146
RCU Registers					
RSTSTAT	System Reset Status	0000_H	U, PV	BE	Page 11-148
RSTSET	Reset Set Register	0004_H	nBE	PV	Page 11-149
RSTCLR	Reset Clear Register	0008_H	nBE	PV	Page 11-150
PRSTAT0	Peripheral Reset Status Register 0	$000C_H$	U, PV	PV	Page 11-151
PRSET0	Peripheral Reset Set Register 0	0010_H	nBE	PV	Page 11-153
PRCLR0	Peripheral Reset Clear Register 0	0014_H	nBE	PV	Page 11-154
PRSTAT1	Peripheral Reset Status Register 1	0018_H	U, PV	BE	Page 11-156
PRSET1	Peripheral Reset Set Register 1	$001C_H$	nBE	PV	Page 11-157
PRCLR1	Peripheral Reset Clear Register 1	0020_H	nBE	PV	Page 11-158
PRSTAT2	Peripheral Reset Status Register 2	0024_H	U, PV	BE	Page 11-159
PRSET2	Peripheral Reset Set Register 2	0028_H	nBE	PV	Page 11-160

Table 11-11 Registers Overview (cont'd)

Short Name	Register Long Name	Offset Addr. [1]	Access Mode		Description
			Read	Write	
PRCLR2	Peripheral Reset Clear Register 2	$002C_H$	nBE	PV	Page 11-161
CCU Registers					
CLKSTAT	Clock Status Register	0000_H	U,PV	BE	Page 11-162
CLKSET	Clock Set Control Register	0004_H	nBE	PV	Page 11-163
CLKCLR	Clock clear Control Register	0008_H	nBE	PV	Page 11-164
SYSCLKCR	System Clock Control	$000C_H$	U, PV	PV	Page 11-165
CPUCLKCR	CPU Clock Control	0010_H	U, PV	PV	Page 11-166
PBCLKCR	Peripheral Bus Clock Control	0014_H	U, PV	PV	Page 11-167
USBCLKCR	USB Clock Control	0018_H	U, PV	PV	Page 11-168
CCUCLKCR	CCU Clock Control	0020_H	U, PV	PV	Page 11-169
WDTCLKCR	WDT Clock Control	0024_H	U, PV	PV	Page 11-169
EXTCLKCR	External clock Control Register	0028_H	U, PV	PV	Page 11-170
MLINKCLKCR	Multi-Link Clock Control Register	$002C_H$	U, PV	PV	Page 11-171
SLEEPCR	Sleep Control Register	0030_H	U, PV	PV	Page 11-173
DSLEEPCR	Deep Sleep Control Register	0034_H	U, PV	PV	Page 11-174
CGATSTAT0	Clock Gating Status for Peripherals 0	0040_H	U, PV	BE	Page 11-176
CGATSET0	Clock Gating Set for Peripherals 0	0044_H	U, PV	PV	Page 11-177
CGATCLR0	Clock Gating Clear for Peripherals 0	0048_H	U, PV	PV	Page 11-178
CGATSTAT1	Clock Gating Status for Peripherals 1	$004C_H$	U, PV	BE	Page 11-180
CGATSET1	Clock Gating Set for Peripherals 1	0050_H	U, PV	PV	Page 11-181

Table 11-11 Registers Overview (cont'd)

Short Name	Register Long Name	Offset Addr. 1)	Access Mode		Description
			Read	Write	
CGATCLR1	Clock Gating Clear for Peripherals 1	0054_H	U, PV	PV	Page 11-182
CGATSTAT2	Clock Gating Status for Peripherals 2	0058_H	U, PV	BE	Page 11-183
CGATSET2	Clock Gating Set for Peripherals 2	$005C_H$	U, PV	PV	Page 11-184
CGATCLR2	Clock Gating Clear for Peripherals 2	0060_H	U, PV	PV	Page 11-185
OSCHPSTAT	OSC_HP Status Register	0100_H	U, PV	BE	Page 11-186
OSCHPCTRL	OSC_HP Control Register	0104_H	U, PV	PV	Page 11-187
CLKCALCONST	Clock Calibration Constant Register	$010C_H$	U, PV	SP	Page 11-188
PLLSTAT	System PLL Status Register	0110_H	U, PV	BE	Page 11-189
PLLCON0	System PLL Configuration 0 Register	0114_H	U, PV	PV	Page 11-191
PLLCON1	System PLL Configuration 1 Register	0118_H	U, PV	PV	Page 11-193
PLLCON2	System PLL Configuration 2 Register	$011C_H$	U, PV	PV	Page 11-194
USBPLLSTAT	USB PLL Status Register	0120_H	U, PV	PV	Page 11-194
USBPLLCON	USB PLL Control Register	0124_H	U, PV	PV	Page 11-196
CLKMXSTAT	Clock Multiplexing Status Register	0138_H	U, PV	BE	Page 11-198

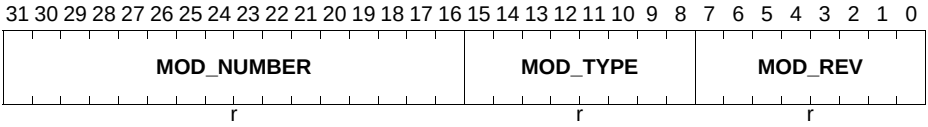
1) The absolute register address is calculated as follows:
Module Base Address + Sub-Module Offset Address + Offset Address (shown in this column)

11.10.1 GCU Registers

ID

Register containing unique ID of the module.

ID
SCU Module ID Register **(0000_H)** **Reset Value: 009A C0XX$_H$**

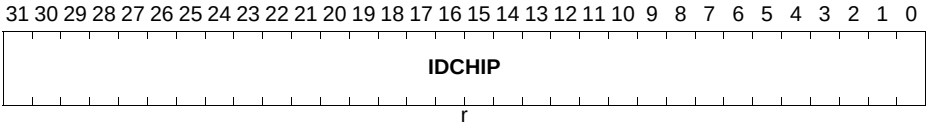


Field	Bits	Type	Description
MOD_REV	[7:0]	r	**Module Revision** Indicates the revision number of the implementation. This information depends on the design step.
MOD_TYPE	[15:8]	r	**Module Type** This internal marker is fixed to $C0_H$.
MOD_NUMBER	[31:16]	r	**Module Number** Indicates the module identification number

IDCHIP

Register containing unique ID of the chip.

IDCHIP
Chip ID Register **(0004_H)** **Reset Value: XXXX XXXX$_H$**

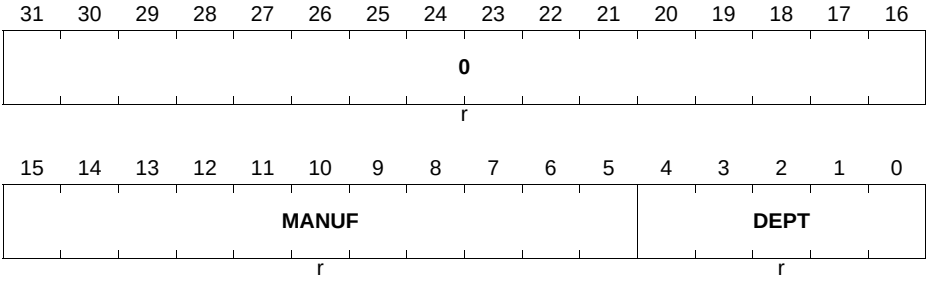


Field	Bits	Type	Description
IDCHIP	[31:0]	r	**Chip ID**

IDMANUF

Register containing unique manufactory ID of the chip.

IDMANUF
Manufactory ID Register **(0008_H)** **Reset Value: 0000 1820$_H$**

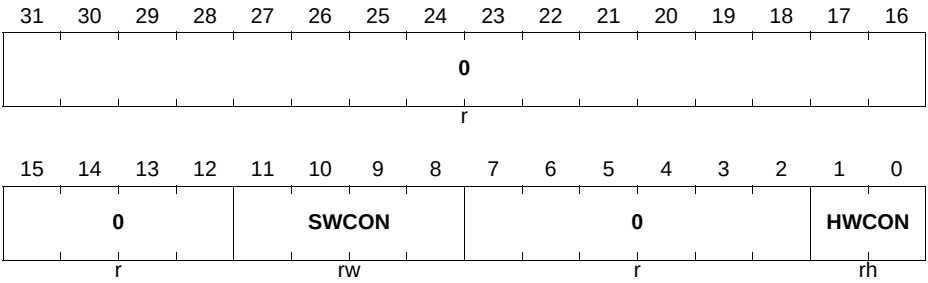


Field	Bits	Type	Description
DEPT	[4:0]	r	**Department Identification Number** DEPT indicated department within Infineon Technologies.
MANUF	[15:5]	r	**Manufacturer Identification Number** JEDEC normalized Manufacturer code. MANUF = $C1_H$ stands for Infineon Technologies.
0	[31:16]	r	**Reserved**

STCON

Startup configuration register determining boot process of the chip.

STCON
Startup Configuration Register **(0010_H)** **Reset Value: 0000 0000$_H$**

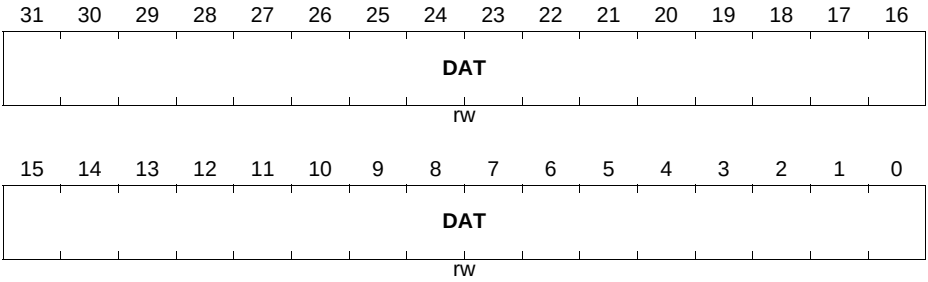


Field	Bits	Type	Description
HWCON	[1:0]	rh	**HW Configuration** At PORESET the following values are latched HWCON.0 = not (TMS) HWCON.1 = TCK 00_B Normal mode, JTAG 01_B ASC BSL enabled 10_B BMI customized boot enabled 11_B CAN BSL enabled
SWCON	[11:8]	rw	**SW Configuration** Bit[9:8] is copy of Bit[1:0] after PORESET 0000_B Normal mode, boot from Boot ROM 0001_B ASC BSL enabled 0010_B BMI customized boot enabled 0011_B CAN BSL enabled 0100_B Boot from Code SRAM 1000_B Boot from alternate Flash Address 0 1100_B Boot from alternate Flash Address 1 1110_B Enable fallback Alternate Boot Mode (ABM) *Note: Only reset with Power-on Reset*
0	[7:2], [31:12]	r	**Reserved** Read as 0; should be written with 0.

GPRx

Software support registers. Can be reset only with PORST reset.

GPRx (x=0-1)
General Purpose Register x **($002C_H$+ x*4)** **Reset Value: $0000\ 0000_H$**

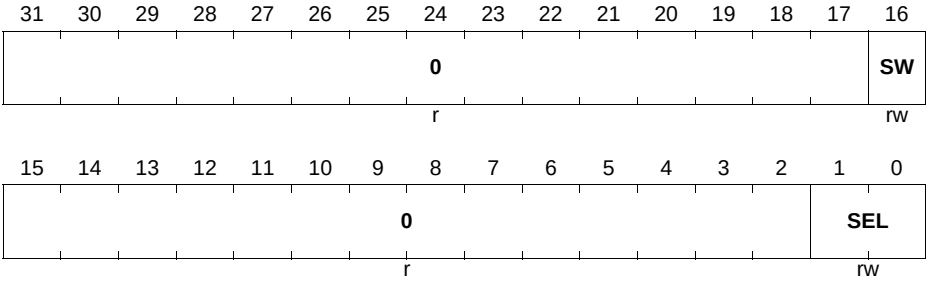


Field	Bits	Type	Description
DAT	[31:0]	rw	**User Data** 32-bit data *Note: GPRx registers can be reset with PORST reset only*

USBHOSTDET

USB Host Detection.

USBHOSTDET
USB Host Detection **(003C$_H$)** **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
SEL	[1:0]	rw	**VBUS Input Multiplexer Control** This bit field controls VBUS selection input. 00_B Software bit input is selected 01_B VBUSDETECTA input is selected 10_B VBUSDETECTB input is selected 11_B VBUSDETECTC input is selected
SW	16	rw	**USB Host Software Select** This bit field enables software control of USB PHY pull-up. 0_B Pull device controlled from USB MAC 1_B Host connected
0	[15:2], [31:17]	r	**Reserved** Read as 0; should be written with 0.

CCUCON

CAPCOM module control register. Individual signals signal is generated with f_{CCU} clock.

CCUCON
CCU Control Register **($004C_H$)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0							GSHR0	0							
r							rw	r							

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0						0	GSC80	0				0	0	GSC41	GSC40
r						r	rw	r				r	r	rw	rw

Field	Bits	Type	Description
GSC40	0	rw	**Global Start Control CCU40** 0_B Disable 1_B Enable
GSC41	1	rw	**Global Start Control CCU41** 0_B Disable 1_B Enable
GSC80	8	rw	**Global Start Control CCU80** 0_B Disable 1_B Enable
GSHR0	24	rw	**Global Start Control HRPWM0** 0_B Disable 1_B Enable
0	2,3,9, [7:4], [23:10], [31:25]	r	**Reserved** Read as 0; should be written with 0.

SRSTAT

Service request status reflecting masking with SRMSK mask register. Write one to a bit in SRCLR register to clear a bit or SRSET to set a bit. Writing zero has no effect. Outputs of this register are used to trigger interrupts or service requests.

SRSTAT

SCU Service Request Status **(0074_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0		RMX	RTC_TIM1	RTC_TIM0	RTC_ATIM1	RTC_ATIM0	RTC_CTR	OSCULCTRL	0	OSCSICTRL	0	HDCR	HDSET	HDCLR	0
r		rh	rh	rh	rh	rh	rh	rh	r	rh	r	rh	rh	rh	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	HINTSET	HINTCLR	HINTST	LPACSET	LPACCLR	LPACST	LPACTH1	LPACTH0	LPACCR	0		DLROVR	AI	PI	PRWARN
r	rh	rh	rh	rh	rh	rh	rh	rh	rh	r		rh	rh	rh	rh

Field	Bits	Type	Description
PRWARN	0	rh	**WDT pre-warning Interrupt Status** 0_B Inactive 1_B Active
PI	1	rh	**RTC Periodic Interrupt Status** Set whenever periodic counter increments
AI	2	rh	**Alarm Interrupt Status** Set whenever count value matches compare value
DLROVR	3	rh	**DLR Request Overrun Interrupt Status** Set whenever DLR overrun condition occurs.
LPACCR	6	rh	**LPACLR Mirror Register Update Status** 0_B Not updated 1_B Update completed
LPACTH0	7	rh	**LPACTH0 Mirror Register Update Status** 0_B Not updated 1_B Update completed
LPACTH1	8	rh	**LPACTH1 Mirror Register Update Status** 0_B Not updated 1_B Update completed
LPACST	9	rh	**LPACST Mirror Register Update Status** 0_B Not updated 1_B Update completed

Field	Bits	Type	Description
LPACCLR	10	rh	**LPACCLR Mirror Register Update Status** 0_B Not updated 1_B Update completed
LPACSET	11	rh	**LPACSET Mirror Register Update Status** 0_B Not updated 1_B Update completed
HINTST	12	rh	**HINTST Mirror Register Update Status** 0_B Not updated 1_B Update completed
HINTCLR	13	rh	**HINTCLR Mirror Register Update Status** 0_B Not updated 1_B Update completed
HINTSET	14	rh	**HINTSET Mirror Register Update Status** 0_B Not updated 1_B Update completed
HDCLR	17	rh	**HDCLR Mirror Register Update Status** 0_B Not updated 1_B Update completed
HDSET	18	rh	**HDSET Mirror Register Update Status** 0_B Not updated 1_B Update completed
HDCR	19	rh	**HDCR Mirror Register Update Status** 0_B Not updated 1_B Update completed
OSCSICTRL	21	rh	**OSCSICTRL Mirror Register Update Status** 0_B Not updated 1_B Update completed
OSCULCTRL	23	rh	**OSCULCTRL Mirror Register Update Status** 0_B Not updated 1_B Update completed
RTC_CTR	24	rh	**RTC CTR Mirror Register Update Status** 0_B Not updated 1_B Update completed
RTC_ATIM0	25	rh	**RTC ATIM0 Mirror Register Update Status** 0_B Not updated 1_B Update completed

Field	Bits	Type	Description
RTC_ATIM1	26	rh	**RTC ATIM1 Mirror Register Update Status** 0_B Not updated 1_B Update completed
RTC_TIM0	27	rh	**RTC TIM0 Mirror Register Update Status** 0_B Not updated 1_B Update completed
RTC_TIM1	28	rh	**RTC TIM1 Mirror Register Update Status** 0_B Not updated 1_B Update completed
RMX	29	rh	**Retention Memory Mirror Register Update Status** 0_B Not updated 1_B Update completed
0	[5:4], 15,16, 20, 22, [31:30]	r	**Reserved**

SRRAW

Service request status without masking. Write one to a bit in **SRCLR** register to clear a bit or **SRSET** to set a bit. Writing zero has no effect.

SRRAW

SCU Raw Service Request Status **(0078_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0		**RMX**	**RTC_TIM1**	**RTC_TIM0**	**RTC_ATIM1**	**RTC_ATIM0**	**RTC_CTR**	**OSCULCTRL**	**0**	**OSCSICTRL**	**0**	**HDCR**	**HDSET**	**HDCLR**	**0**
r		rh	rh	rh	rh	rh	rh	rh	r	rh	r	rh	rh	rh	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**HINTSET**	**HINTCLR**	**HINTST**	**LPACSET**	**LPACCLR**	**LPACST**	**LPACTH1**	**LPACTH0**	**LPACCR**	**0**		**DLROVR**	**AI**	**PI**	**PRWARN**
r	rh	rh	rh	rh	rh	rh	rh	rh	rh	r		rh	rh	rh	rh

Field	Bits	Type	Description
PRWARN	0	rh	**WDT pre-warning Interrupt Status Before Masking** 0_B Inactive 1_B Active
PI	1	rh	**RTC Raw Periodic Interrupt Status Before Masking** Set whenever periodic counter increments
AI	2	rh	**RTC Raw Alarm Interrupt Status Before Masking** Set whenever count value matches compare value
DLROVR	3	rh	**DLR Request Overrun Interrupt Status Before Masking** Set whenever DLR overrun condition occurs.
LPACCR	6	rh	**LPACLR Mirror Register Update Status Before Masking** 0_B Not updated 1_B Update completed
LPACTH0	7	rh	**LPACTH0 Mirror Register Update Status Before Masking** 0_B Not updated 1_B Update completed
LPACTH1	8	rh	**LPACTH1 Mirror Register Update Status Before Masking** 0_B Not updated 1_B Update completed
LPACST	9	rh	**LPACST Mirror Register Update Status Before Masking** 0_B Not updated 1_B Update completed
LPACCLR	10	rh	**LPACCLR Mirror Register Update Status Before Masking** 0_B Not updated 1_B Update completed
LPACSET	11	rh	**LPACSET Mirror Register Update Status Before Masking** 0_B Not updated 1_B Update completed

Field	Bits	Type	Description
HINTST	12	rh	**HINTST Mirror Register Update Status Before Masking** 0_B Not updated 1_B Update completed
HINTCLR	13	rh	**HINTCLR Mirror Register Update Status Before Masking** 0_B Not updated 1_B Update completed
HINTSET	14	rh	**HINTSET Mirror Register Update Status Before Masking** 0_B Not updated 1_B Update completed
HDCLR	17	rh	**HDCLR Mirror Register Update Status Before Masking** 0_B Not updated 1_B Update completed
HDSET	18	rh	**HDSET Mirror Register Update Status Before Masking** 0_B Not updated 1_B Update completed
HDCR	19	rh	**HDCR Mirror Register Update Status Before Masking** 0_B Not updated 1_B Update completed
OSCSICTRL	21	rh	**OSCSICTRL Mirror Register Update Status Before Masking** 0_B Not updated 1_B Update completed
OSCULCTRL	23	rh	**OSCULCTRL Mirror Register Update Status Before Masking** 0_B Not updated 1_B Update completed
RTC_CTR	24	rh	**RTC CTR Mirror Register Update Status Before Masking** 0_B Not updated 1_B Update completed

Field	Bits	Type	Description
RTC_ATIM0	25	rh	**RTC ATIM0 Mirror Register Update Status Before Masking** 0_B Not updated 1_B Update completed
RTC_ATIM1	26	rh	**RTC ATIM1 Mirror Register Update Status Before Masking** 0_B Not updated 1_B Update completed
RTC_TIM0	27	rh	**RTC TIM0 Mirror Register Update Before Masking Status** 0_B Not updated 1_B Update completed
RTC_TIM1	28	rh	**RTC TIM1 Mirror Register Update Status Before Masking** 0_B Not updated 1_B Update completed
RMX	29	rh	**Retention Memory Mirror Register Update Status Before Masking** 0_B Not updated 1_B Update completed
0	[5:4], 15,16, 20, 22, [31:30]	r	**Reserved**

SRMSK

Service request mask used to mask outputs of **SRRAW** register outputs connected to **SRSTAT** register.

SRMSK
SCU Service Request Mask **(007C$_H$)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0		RMX	RTC_TIM1	RTC_TIM0	RTC_ATIM1	RTC_ATIM0	RTC_CTR	OSCULCTRL	0	OSCSICTRL	0	HDCR	HDSET	HDCLR	0
r		rw	rw	rw	rw	rw	rw	rw	r	rw	r	rw	rw	rw	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	HINTSET	HINTCLR	HINTST	LPACSET	LPACCLR	LPACST	LPACTH1	LPACTH0	LPACCR	0		DLROVR	AI	PI	PRWARN
r	rw	rw	rw	rw	rw	rw	rw	rw	rw	r		rw	rw	rw	rw

Field	Bits	Type	Description
PRWARN	0	rw	**WDT pre-warning Interrupt Mask** 0_B Disabled 1_B Enabled
PI	1	rw	**RTC Periodic Interrupt Mask** 0_B Disabled 1_B Enabled
AI	2	rw	**RTC Alarm Interrupt Mask** 0_B Disabled 1_B Enabled
DLROVR	3	rw	**DLR Request Overrun Interrupt Mask** 0_B Disabled 1_B Enabled
LPACCR	6	rw	**LPACLR Mirror Register Update Interrupt Mask** 0_B Disabled 1_B Enabled
LPACTH0	7	rw	**LPACTH0 Mirror Register Update Interrupt Mask** 0_B Disabled 1_B Enabled
LPACTH1	8	rw	**LPACTH1 Mirror Register Update Interrupt Mask** 0_B Disabled 1_B Enabled

Field	Bits	Type	Description
LPACST	9	rw	**LPACST Mirror Register Update Interrupt Mask** 0_B Disabled 1_B Enabled
LPACCLR	10	rw	**LPACCLR Mirror Register Update Interrupt Mask** 0_B Disabled 1_B Enabled
LPACSET	11	rw	**LPACSET Mirror Register Update Interrupt Mask** 0_B Disabled 1_B Enabled
HINTST	12	rw	**HINTST Mirror Register Update Interrupt Mask** 0_B Disabled 1_B Enabled
HINTCLR	13	rw	**HINTCLR Mirror Register Update Interrupt Mask** 0_B Disabled 1_B Enabled
HINTSET	14	rw	**HINTSET Mirror Register Update Interrupt Mask** 0_B Disabled 1_B Enabled
HDCLR	17	rw	**HDCLR Mirror Register Update Mask** 0_B Disabled 1_B Enabled
HDSET	18	rw	**HDSET Mirror Register Update Mask** 0_B Disabled 1_B Enabled
HDCR	19	rw	**HDCR Mirror Register Update Mask** 0_B Disabled 1_B Enabled
OSCSICTRL	21	rw	**OSCSICTRL Mirror Register Update Mask** 0_B Disabled 1_B Enabled
OSCULCTRL	23	rw	**OSCULCTRL Mirror Register Update Mask** 0_B Disabled 1_B Enabled
RTC_CTR	24	rw	**RTC CTR Mirror Register Update Mask** 0_B Disabled 1_B Enabled

Field	Bits	Type	Description
RTC_ATIM0	25	rw	**RTC ATIM0 Mirror Register Update Mask** 0_B Disabled 1_B Enabled
RTC_ATIM1	26	rw	**RTC ATIM1 Mirror Register Update Mask** 0_B Disabled 1_B Enabled
RTC_TIM0	27	rw	**RTC TIM0 Mirror Register Update Mask** 0_B Disabled 1_B Enabled
RTC_TIM1	28	rw	**RTC TIM1 Mirror Register Update Mask** 0_B Disabled 1_B Enabled
RMX	29	rw	**Retention Memory Mirror Register Update Mask** 0_B Disabled 1_B Enabled
0	[5:4], 15,16, 20, 22, [31:30]	r	**Reserved**

SRCLR

Clear service request bits of registers **SRRAW** and **SRSTAT**. Write one to clear corresponding bits. Writing zeros has no effect.

SRCLR
SCU Service Request Clear **(0080_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0		**RMX**	**RTC_TIM1**	**RTC_TIM0**	**RTC_ATIM1**	**RTC_ATIM0**	**RTC_CTR**	**OSCULCTRL**	**0**	**OSCSICTRL**	**0**	**HDCR**	**HDSET**	**HDCLR**	**0**
r		w	w	w	w	w	w	w	r	w	r	w	w	w	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**HINTSET**	**HINTCLR**	**HINTST**	**LPACSET**	**LPACCLR**	**LPACST**	**0**	**LPACTH0**	**LPACCR**	**0**		**DLROVR**	**AI**	**PI**	**PRWARN**
r	w	w	w	w	w	w	r	w	w	r		w	w	w	w

Field	Bits	Type	Description
PRWARN	0	w	**WDT pre-warning Interrupt Clear** 0_B No effect 1_B Clear the status bit
PI	1	w	**RTC Periodic Interrupt Clear** 0_B No effect 1_B Clear the status bit
AI	2	w	**RTC Alarm Interrupt Clear** 0_B No effect 1_B Clear the status bit
DLROVR	3	w	**DLR Request Overrun Interrupt clear** 0_B No effect 1_B Clear the status bit
LPACCR	6	w	**LPACLR Mirror Register Update Interrupt Clear** 0_B No effect 1_B Clear the status bit
LPACTH0	7	w	**LPACTH0 Mirror Register Update Interrupt Clear** 0_B No effect 1_B Clear the status bit
LPACTH1	8	w	**LPACTH1 Mirror Register Update Interrupt Clear** 0_B No effect 1_B Clear the status bit
LPACST	9	w	**LPACST Mirror Register Update Interrupt Clear** 0_B No effect 1_B Clear the status bit
LPACCLR	10	w	**LPACCLR Mirror Register Update Interrupt Clear** 0_B No effect 1_B Clear the status bit
LPACSET	11	w	**LPACSET Mirror Register Update Interrupt Clear** 0_B No effect 1_B Clear the status bit
HINTST	12	w	**HINTST Mirror Register Update Interrupt Clear** 0_B No effect 1_B Clear the status bit
HINTCLR	13	w	**HINTCLR Mirror Register Update Interrupt Clear** 0_B No effect 1_B Clear the status bit

Field	Bits	Type	Description
HINTSET	14	w	**HINTSET Mirror Register Update Interrupt Clear** 0_B No effect 1_B Clear the status bit
HDCLR	17	w	**HDCLR Mirror Register Update Clear** 0_B No effect 1_B Clear the status bit
HDSET	18	w	**HDSET Mirror Register Update Clear** 0_B No effect 1_B Clear the status bit
HDCR	19	w	**HDCR Mirror Register Update Clear** 0_B No effect 1_B Clear the status bit
OSCSICTRL	21	w	**OSCSICTRL Mirror Register Update Clear** 0_B No effect 1_B Clear the status bit
OSCULCTRL	23	w	**OSCULCTRL Mirror Register Update Clear** 0_B No effect 1_B Clear the status bit
RTC_CTR	24	w	**RTC CTR Mirror Register Update Clear** 0_B No effect 1_B Clear the status bit
RTC_ATIM0	25	w	**RTC ATIM0 Mirror Register Update Clear** 0_B No effect 1_B Clear the status bit
RTC_ATIM1	26	w	**RTC ATIM1 Mirror Register Update Clear** 0_B No effect 1_B Clear the status bit
RTC_TIM0	27	w	**RTC TIM0 Mirror Register Update Clear** 0_B No effect 1_B Clear the status bit
RTC_TIM1	28	w	**RTC TIM1 Mirror Register Update Clear** 0_B No effect 1_B Clear the status bit
RMX	29	w	**Retention Memory Mirror Register Update Clear** 0_B No effect 1_B Clear the status bit

Field	Bits	Type	Description
0	[5:4], 15,16, 20, 22, [31:30]	r	**Reserved**

SRSET

Set service request fits of registers **SRRAW** and **SRSTAT**. Write one to clear corresponding bits. Writing zeros has no effect.

SRSET
SCU Service Request Set **(0084$_H$)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0		**RMX**	**RTC_TIM1**	**RTC_TIM0**	**RTC_ATIM1**	**RTC_ATIM0**	**RTC_CTR**	**OSCULCTRL**	**0**	**OSCSICTRL**	**0**	**HDCR**	**HDCRSET**	**HDCRCLR**	**0**
r		w	w	w	w	w	w	w	r	w	r	w	w	w	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**HINTSET**	**HINTCLR**	**HINTST**	**LPACSET**	**LPACCLR**	**LPACST**	**LPACTH1**	**LPACTH0**	**LPACCR**	**0**		**DLROVR**	**AI**	**PI**	**PRWARN**
r	w	w	w	w	w	w	w	w	w	r		w	w	w	w

Field	Bits	Type	Description
PRWARN	0	w	**WDT pre-warning Interrupt Set** 0_B No effect 1_B set the status bit
PI	1	w	**RTC Periodic Interrupt Set** 0_B No effect 1_B set the status bit
AI	2	w	**RTC Alarm Interrupt Set** 0_B No effect 1_B set the status bit
DLROVR	3	w	**DLR Request Overrun Interrupt Set** 0_B No effect 1_B set the status bit

Field	Bits	Type	Description
LPACCR	6	w	**LPACLR Mirror Register Update Interrupt Set** 0_B No effect 1_B set the status bit
LPACTH0	7	w	**LPACTH0 Mirror Register Update Interrupt Set** 0_B No effect 1_B set the status bit
LPACTH1	8	w	**LPACTH1 Mirror Register Update Interrupt Set** 0_B No effect 1_B set the status bit
LPACST	9	w	**LPACST Mirror Register Update Interrupt Set** 0_B No effect 1_B set the status bit
LPACCLR	10	w	**LPACCLR Mirror Register Update Interrupt Set** 0_B No effect 1_B set the status bit
LPACSET	11	w	**LPACSET Mirror Register Update Interrupt Set** 0_B No effect 1_B set the status bit
HINTST	12	w	**HINTST Mirror Register Update Interrupt Set** 0_B No effect 1_B set the status bit
HINTCLR	13	w	**HINTCLR Mirror Register Update Interrupt Set** 0_B No effect 1_B set the status bit
HINTSET	14	w	**HINTSET Mirror Register Update Interrupt Set** 0_B No effect 1_B set the status bit
HDCRCLR	17	w	**HDCRCLR Mirror Register Update Set** 0_B No effect 1_B set the status bit
HDCRSET	18	w	**HDCRSET Mirror Register Update Set** 0_B No effect 1_B set the status bit
HDCR	19	w	**HDCR Mirror Register Update Set** 0_B No effect 1_B set the status bit

Field	Bits	Type	Description
OSCSICTRL	21	w	**OSCSICTRL Mirror Register Update Set** 0_B No effect 1_B set the status bit
OSCULCTRL	23	w	**OSCULCTRL Mirror Register Update Set** 0_B No effect 1_B set the status bit
RTC_CTR	24	w	**RTC CTR Mirror Register Update Set** 0_B No effect 1_B set the status bit
RTC_ATIM0	25	w	**RTC ATIM0 Mirror Register Update Set** 0_B No effect 1_B set the status bit
RTC_ATIM1	26	w	**RTC ATIM1 Mirror Register Update Set** 0_B No effect 1_B set the status bit
RTC_TIM0	27	w	**RTC TIM0 Mirror Register Update Set** 0_B No effect 1_B set the status bit
RTC_TIM1	28	w	**RTC TIM1 Mirror Register Update Set** 0_B No effect 1_B set the status bit
RMX	29	w	**Retention Memory Mirror Register Update Set** 0_B No effect 1_B set the status bit
0	[5:4], 15,16, 20, 22, [31:30]	r	**Reserved**

NMIREQEN

The **NMIREQEN** register serves purpose of promoting service requests to NMI requests. Is a bit is set then corresponding service request reflected in **SRSTAT** otherwise will be mirrored in the **TRAPSTAT** register instead.

NMIREQEN
SCU Service Request Mask **(0088_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0												ERU03	ERU02	ERU01	ERU00
r												rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0													AI	PI	PRWARN
r													rw	rw	rw

Field	Bits	Type	Description
PRWARN	0	rw	**Promote Pre-Warning Interrupt Request to NMI Request** 0_B Disabled 1_B Enabled
PI	1	rw	**Promote RTC Periodic Interrupt request to NMI Request** 0_B Disabled 1_B Enabled
AI	2	rw	**Promote RTC Alarm Interrupt Request to NMI Request** 0_B Disabled 1_B Enabled
ERU00	16	rw	**Promote Channel 0 Interrupt of ERU0 Request to NMI Request** 0_B Disabled 1_B Enabled
ERU01	17	rw	**Promote Channel 1 Interrupt of ERU0 Request to NMI Request** 0_B Disabled 1_B Enabled
ERU02	18	rw	**Promote Channel 2 Interrupt of ERU0 Request to NMI Request** 0_B Disabled 1_B Enabled

Field	Bits	Type	Description
ERU03	19	rw	**Promote Channel 3 Interrupt of ERU0 Request to NMI Request** 0_B Disabled 1_B Enabled
0	[15:3], [31:20]	r	**Reserved**

DTSCON

Die temperature sensor control register

DTSCON
Die Temperature Sensor Control Register ($008C_H$) **Reset Value: $0000\ 0001_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0								BGTRIM				REFTRIM			GAIN
r								rw				rw			rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GAIN					OFFSET							0		START	PWD
rw					rw							r		w	rw

Field	Bits	Type	Description
PWD	0	rw	**Sensor Power Down** This bit defines the DTS power state. 0_B The DTS is powered 1_B The DTS is not powered
START	1	w	**Sensor Measurement Start** This bit starts a measurement of the DTS. 0_B No DTS measurement is started 1_B DTS measurement is started If set this bit is automatically cleared. This bit always reads as zero.

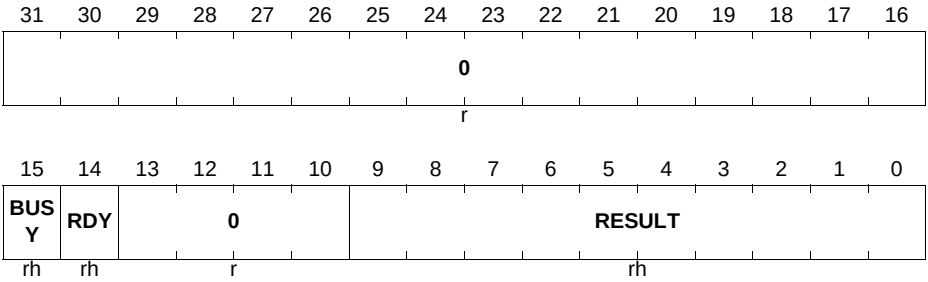
Field	Bits	Type	Description
OFFSET	[10:4]	rw	**Offset Calibration Value** This bit field interfaces the offset calibration values to the DTS. The calibration values are forwarded to the DTS by setting bit START.
GAIN	[16:11]	rw	**Gain Calibration Value** This bit field interfaces the gain calibration values to the DTS. The calibration values are forwarded to the DTS by setting bit START.
REFTRIM	[19:17]	rw	**Reference Trim Calibration Value** This bit field interfaces the reference trim calibration values to the DTS. The calibration values are forwarded to the DTS by setting bit START.
BGTRIM	[23:20]	rw	**Bandgap Trim Calibration Value** This bit field interfaces the bandgap trim calibration values to the DTS. The calibration values are forwarded to the DTS by setting bit START.
0	[3:2], [31:24]	r	**Reserved** Read as 0; should be written with 0.

DTSSTAT

Die temperature status register

DTSSTAT

Die Temperature Sensor Status Register (0090_H) **Reset Value: $0000\ 0000_H$**

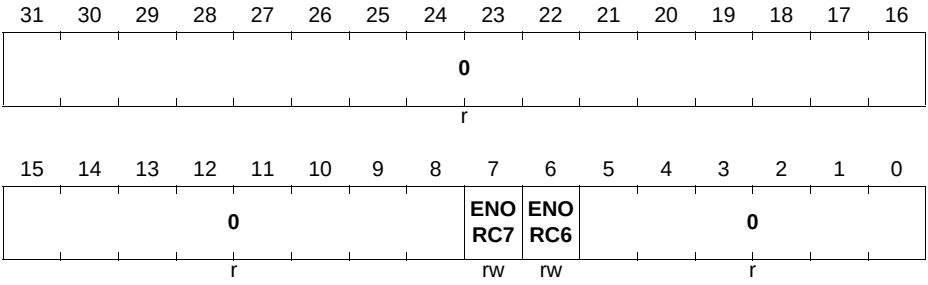


Field	Bits	Type	Description
RESULT	[9:0]	rh	**Result of the DTS Measurement** This bit field shows the result of the DTS measurement. The value given is directly related to the die temperature.
RDY	14	rh	**Sensor Ready Status** This bit indicate the DTS is ready or not. 0_B The DTS is not ready 1_B The DTS is ready
BUSY	15	rh	**Sensor Busy Status** This bit indicate if the DTS is currently busy. If the sensor is busy a measurement is still running and the result should not be used. 0_B not busy 1_B busy
0	[13:10], [31:16]	r	**Reserved**

G0ORCEN

Enable register for out-of-range comparators of group 0 of analog input channels.

G0ORCEN
Out of Range Comparator Enable Register 0 ($00A0_H$) **Reset Value: $0000\ 0000_H$**

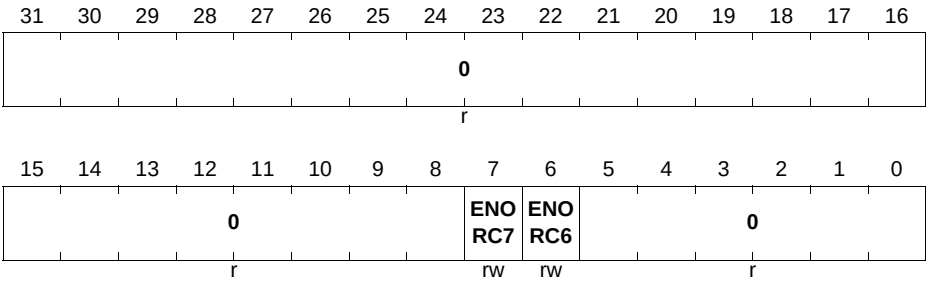


Field	Bits	Type	Description
ENORC6	6	rw	**Enable Out of Range Comparator, Channel 6** Each bit (when set) enables the out of range comparator of the associated channel 0_B Disabled 1_B Enabled
ENORC7	7	rw	**Enable Out of Range Comparator, Channel 7** Each bit (when set) enables the out of range comparator of the associated channel 0_B Disabled 1_B Enabled
0	[5:0], [31:8]	r	**Reserved** returns 0 if read; should be written with 0;

G1ORCEN

Enable register for out-of-range comparators of group 1 of analog input channels.

G1ORCEN
Out of Range Comparator Enable Register 1 ($00A4_H$) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
ENORC6	6	rw	**Enable Out of Range Comparator, Channel 6** Each bit (when set) enables the out of range comparator of the associated channel 0_B Disabled 1_B Enabled

Field	Bits	Type	Description
ENORC7	7	rw	**Enable Out of Range Comparator, Channel 7** Each bit (when set) enables the out of range comparator of the associated channel 0_B Disabled 1_B Enabled
0	[5:0], [31:8]	r	**Reserved** returns 0 if read; should be written with 0;

DTEMPLIM

Configuration of upper and lower temperature limits for service request generation using DTS measurement output values.

DTEMPLIM
Die Temperature Sensor Limit Register(00A8$_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0						**UPPER**									
r						rw									

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0						**LOWER**									
r						rw									

Field	Bits	Type	Description
LOWER	[9:0]	rw	**Lower Limit** This bit field defines the lower limit of the DTS temperature check. The DTS measurement result is compared against this value and if the measurement result value is equal or lower than LOWER then bit UNDERFL of **DTEMPALARM** is set. Otherwise UNDERFL of **DTEMPALARM** remains low.

Field	Bits	Type	Description
UPPER	[25:16]	rw	**Upper Limit** This bit field defines the upper limit of the DTS temperature check. The DTS measurement result is compared against this value and if the measurement result value is equal or greater than UPPER bit then OVERFL of **DTEMPALARM** is set. Otherwise OVERFL of **DTEMPALARM** remains low.
0	[15:10], [31:26]	r	**Reserved** Read as 0; should be written with 0.

DTEMPALARM

Status register of service request generated according to settings in DTEMPLIM.

DTEMPALARM

Die Temperature Sensor Alarm Register(00AC$_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															OVERFL
r															rh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0															UNDERFL
r															rh

Field	Bits	Type	Description
UNDERFL	0	rh	**Lower Limit Underflow** 0_B No temperature underflow was detected 1_B A temperature underflow was detected
OVERFL	16	rh	**Upper Limit Overflow** 0_B No temperature overflow was detected 1_B A temperature overflow was detected
0	[15:1], [31:17]	r	**Reserved**

MIRRSTS

Mirror status register for control of communication between SCU and other modules in hibernate domain. The bitfields of the register indicate that a corresponding register of the hibernate domain is ready to accept a write, or, that the communication interface is busy with executing the previous to the register.

MIRRSTS

Mirror Write Status Register **($00C4_H$)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0							HINTSET	HINTCLR	0	LPACSET	LPACCLR	0	LPACTH1	LPACTH0	LPACCONF
r							rh	rh	r	rh	rh	r	rh	rh	rh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RTC_CLRSR	RTC_MSKSR	RMX	RTC_TIM1	RTC_TIM0	RTC_ATIM1	RTC_ATIM0	RTC_CTR	OSCULCTRL	0	OSCSICTRL	0	HDCR	HDSET	HDCLR	0
rh	rh	rh	rh	rh	rh	rh	rh	rh	r	rh	r	rh	rh	rh	r

Field	Bits	Type	Function
HDCLR	1	rh	**HDCLR Mirror Register Write Status** 0_B Ready 1_B Busy
HDSET	2	rh	**HDSET Mirror Register Write Status** 0_B Ready 1_B Busy
HDCR	3	rh	**HDCR Mirror Register Write Status** 0_B Ready 1_B Busy
OSCSICTRL	5	rh	**OSCSICTRL Mirror Register Write Status** 0_B Ready 1_B Busy
OSCULCTRL	7	rh	**OSCULCTRL Mirror Register Write Status** 0_B Ready 1_B Busy
RTC_CTR	8	rh	**RTC CTR Mirror Register Write Status** 0_B Ready 1_B Busy

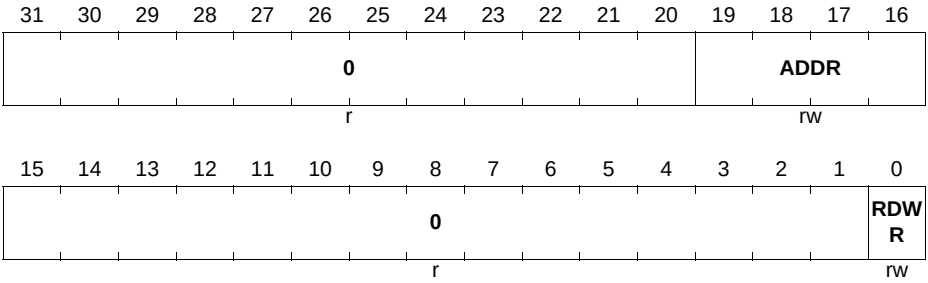
Field	Bits	Type	Function
RTC_ATIM0	9	rh	**RTC ATIM0 Mirror Register Write Status** 0_B Ready 1_B Busy
RTC_ATIM1	10	rh	**RTC ATIM1 Mirror Register Write Status** 0_B Ready 1_B Busy
RTC_TIM0	11	rh	**RTC TIM0 Mirror Register Write Status** 0_B Ready 1_B Busy
RTC_TIM1	12	rh	**RTC TIM1 Mirror Register Write Status** 0_B Ready 1_B Busy
RMX	13	rh	**Retention Memory Access Register Update Status** This fields indicates status of retention memory update from **RMDATA** register to Hibernate domain retention memory or from Hibernate domain to **RMDATA** 0_B Ready 1_B Busy
RTC_MSKSR	14	rh	**RTC MSKSSR Mirror Register Write Status** 0_B Ready 1_B Busy
RTC_CLRSR	15	rh	**RTC CLRSR Mirror Register Write Status** 0_B Ready 1_B Busy
LPACCONF	16	rh	**LPACCONF Mirror Register Write Interrupt Set** 0_B Ready 1_B Busy
LPACTH0	17	rh	**LPACTH0 Mirror Register Write Interrupt Set** 0_B Ready 1_B Busy
LPACTH1	18	rh	**LPACTH1 Mirror Register Write Interrupt Set** 0_B Ready 1_B Busy
LPACCLR	20	rh	**LPACCLR Mirror Register Write Status** 0_B Ready 1_B Busy

Field	Bits	Type	Function
LPACSET	21	rh	**LPACSET Mirror Register Write Status** 0_B Ready 1_B Busy
HINTCLR	23	rh	**HINTCLR Mirror Register Write Status** 0_B Ready 1_B Busy
HINTSET	24	rh	**HINTSET Mirror Register Write Status** 0_B Ready 1_B Busy
0	0,4,6, 19, 22, [31:25]	r	**Reserved**

RMACR

Access control to retention memory in hibernate domain.

RMACR
Retention Memory Access Control Register ($00C8_H$) **Reset Value: $0000\ 0000_H$**

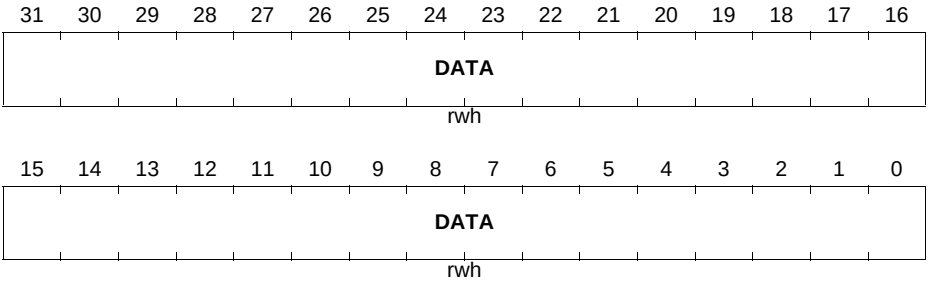


Field	Bits	Type	Function
RDWR	0	rw	**Hibernate Retention Memory Register Update Control** This field controls access to Retention Memory using address selected in ADDR slice 0_B transfer data from Retention Memory in Hibernate domain to **RMDATA** register 1_B transfer data from **RMDATA** into Retention Memory in Hibernate domain
ADDR	[19:16]	rw	**Hibernate Retention Memory Register Address Select** This field selects Retention Memory address of 0 to 15 for read or write access.
0	[15:1], [31:20]	r	**Reserved** Read as 0; should be written with 0.

RMDATA

Access data of retention memory in hibernate domain.

RMDATA
Retention Memory Access Data Register ($00CC_H$) **Reset Value: $0000\ 0000_H$**

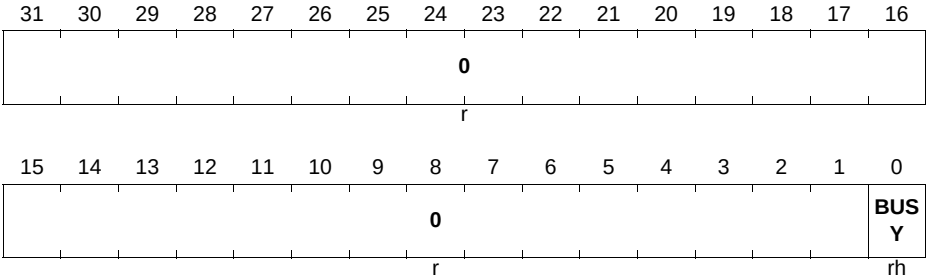


Field	Bits	Type	Function
DATA	[31:0]	rwh	**Hibernate Retention Memory Data** This field data of selected of Retention Memory using address. The address of 0-15 is selected with **RMACR** register.

MIRRALLSTAT

Indicates that mirroring of all hibernate domain registers has been completed after last update request. The update request can be triggered with **MIRRALLREQ** register and also status of update triggered after reset relese.

MIRRALLSTAT
Mirror All Status **($00D0_H$)** **Reset Value: 0000 0000$_H$**

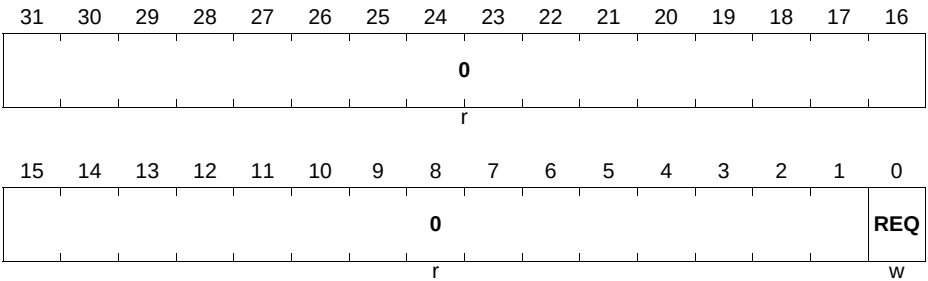


Field	Bits	Type	Function
BUSY	0	rh	**Mirror All Execution Status** 0_B No update is pening 1_B Update is pending
0	[31:1]	r	**Reserved**

MIRRALLREQ

Request mirror update of all hibernate domain registers. Status of the update execution gets reflected in the **MIRRALLSTAT** register.

MIRRALLREQ
Mirror All Request **($00D4_H$)** **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Function
REQ	0	w	**Mirror All Execution Request** 0_B No action 1_B Start mirror update
0	[31:1]	r	**Reserved**

PEEN

The following register enables parity check mechanism on peripheral modules.

PEEN
Parity Error Enable Register **($013C_H$)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0											0		0		PEENUSB
r											r		r		rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		PEENPPRF	PEENMC	0	0	PEENU1	PEENU0	0					0	PEENDS1	PEENPS
r		rw	rw	r	r	rw	rw	r					r	rw	rw

Field	Bits	Type	Description
PEENPS	0	rw	**Parity Error Enable for PSRAM** 0_B Disabled 1_B Enabled
PEENDS1	1	rw	**Parity Error Enable for DSRAM1** 0_B Disabled 1_B Enabled
PEENU0	8	rw	**Parity Error Enable for USIC0 Memory** 0_B Disabled 1_B Enabled
PEENU1	9	rw	**Parity Error Enable for USIC1 Memory** 0_B Disabled 1_B Enabled

Field	Bits	Type	Description
PEENMC	12	rw	**Parity Error Enable for MultiCAN Memory** 0_B Disabled 1_B Enabled
PEENPPRF	13	rw	**Parity Error Enable for PMU Prefetch Memory** 0_B Disabled 1_B Enabled
PEENUSB	16	rw	**Parity Error Enable for USB Memory** 0_B Disabled 1_B Enabled
0	2, [7:3], 10, 11, [15:14], [18:17], [20:19], [31:21]	r	**Reserved** Should be written with 0.

MCHKCON

The following register enables the functional parity check mechanism for testing purpose. MCHKCON register is used to support access to parity bits of SRAM modules for various types of peripherals. The SRAM modules providing direct access natively need to be selected in order to enable direct write to parity bits using **PMTPR** register.

MCHKCON
Memory Checking Control Register (0140_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0											0		0		SEL USB
r											r		r		rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		PPR FDR A	MCA NDR A	0	0	USIC 1DR A	USIC 0DR A	0					0	SEL DS1	SEL PS
r		rw	rw	r	r	rw	rw	r					r	rw	rw

Field	Bits	Type	Description
SELPS	0	rw	**Select Memory Check for PSRAM** 0_B Not selected 1_B Selected
SELDS1	1	rw	**Select Memory Check for DSRAM1** 0_B Not selected 1_B Selected
USIC0DRA	8	rw	**Select Memory Check for USIC0** 0_B Not selected 1_B Selected
USIC1DRA	9	rw	**Select Memory Check for USIC1** 0_B Not selected 1_B Selected
MCANDRA	12	rw	**Select Memory Check for MultiCAN** 0_B Not selected 1_B Selected
PPRFDRA	13	rw	**Select Memory Check for PMU** 0_B Not selected 1_B Selected
SELUSB	16	rw	**Select Memory Check for USB SRAM** 0_B Not selected 1_B Selected
0	2, [7:3], 10, 11, [15:14], [18:17], [20:19], [31:21]	r	**Reserved** Should be written with 0.

PETE

The following register enables the functional parity error trap generation mechanism. The trap flag gets reflected in **TRAPRAW** register and needs to be enabled with **TRAPDIS** register before can be effectively used to generate an NMI. The same tap flag can be configured with **PERSTEN** register to generate System Reset instead of an NMI.

PETE
Parity Error Trap Enable Register (0144$_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0											0		0		PETEUSB
r											r		r		rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		PETEPPRF	PETEMC	0	0	PETEU1	PETEU0	0					0	PETEDS1	PETEPS
r		rw	rw	r	r	rw	rw	r					r	rw	rw

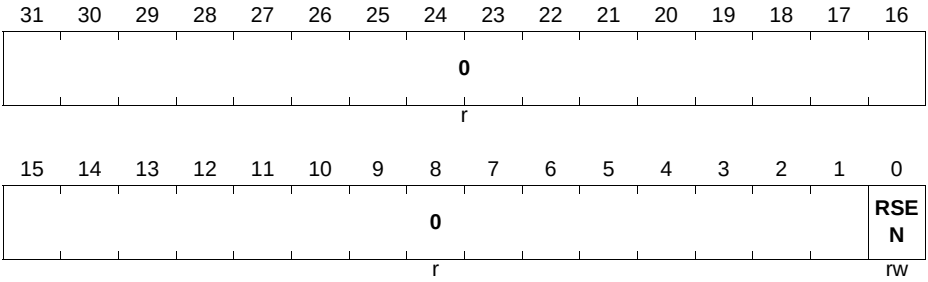
Field	Bits	Type	Description
PETEPS	0	rw	**Parity Error Trap Enable for PSRAM** 0_B Disabled 1_B Enabled
PETEDS1	1	rw	**Parity Error Trap Enable for DSRAM1** 0_B Disabled 1_B Enabled
PETEU0	8	rw	**Parity Error Trap Enable for USIC0 Memory** 0_B Disabled 1_B Enabled
PETEU1	9	rw	**Parity Error Trap Enable for USIC1 Memory** 0_B Disabled 1_B Enabled
PETEMC	12	rw	**Parity Error Trap Enable for MultiCAN Memory** 0_B Disabled 1_B Enabled
PETEPPRF	13	rw	**Parity Error Trap Enable for PMU Prefetch Memory** 0_B Disabled 1_B Enabled
PETEUSB	16	rw	**Parity Error Trap Enable for USB Memory** 0_B Disabled 1_B Enabled

Field	Bits	Type	Description
0	2, [7:3], 10, 11, [15:14], [18:17], [20:19], [31:21]	r	**Reserved** Should be written with 0.

PERSTEN

The following register enables reset upon parity error flag from the functional parity check mechanism indicated in **PEFLAG** register.

PERSTEN
Parity Error Reset Enable Register (0148$_H$) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
RSEN	0	rw	**System Reset Enable upon Parity Error Trap** 0_B Reset request disabled 1_B Reset request enabled
0	[31:1]	r	**Reserved** Should be written with 0.

PEFLAG

The PEFLAG register controls the functional parity check mechanism.

The register bits can only get set by corresponding parity error assertion if enabled and can only be cleared via software. Writing a zero to this bit does not change the content. Writing a one to this bit does clear the bit.

PEFLAG
Parity Error Flag Register **(0150_H)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0											0		0		PEUSB
r											r		r		rwh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		PEFPPRF	PEFMC	0	0	PEFU1	PEFU0	0					0	PEFDS1	PEFPS
r		rwh	rwh	r	r	rwh	rwh	r					r	rwh	rwh

Field	Bits	Type	Description
PEFPS	0	rwh	**Parity Error Flag for PSRAM** 0_B No parity error detected 1_B Parity error detected
PEFDS1	1	rwh	**Parity Error Flag for DSRAM1** 0_B No parity error detected 1_B Parity error detected
PEFU0	8	rwh	**Parity Error Flag for USIC0 Memory** 0_B No parity error detected 1_B Parity error detected
PEFU1	9	rwh	**Parity Error Flag for USIC1 Memory** 0_B No parity error detected 1_B Parity error detected
PEFMC	12	rwh	**Parity Error Flag for MultiCAN Memory** 0_B No parity error detected 1_B Parity error detected
PEFPPRF	13	rwh	**Parity Error Flag for PMU Prefetch Memory** 0_B No parity error detected 1_B Parity error detected
PEUSB	16	rwh	**Parity Error Flag for USB Memory** 0_B No parity error detected 1_B Parity error detected

Field	Bits	Type	Description
0	2, [7:3], 10, 11, [15:14], [18:17], [20:19], [31:21]	r	**Reserved** Should be written with 0.

PMTPR

The following register provides direct access to parity bits of a selected module.

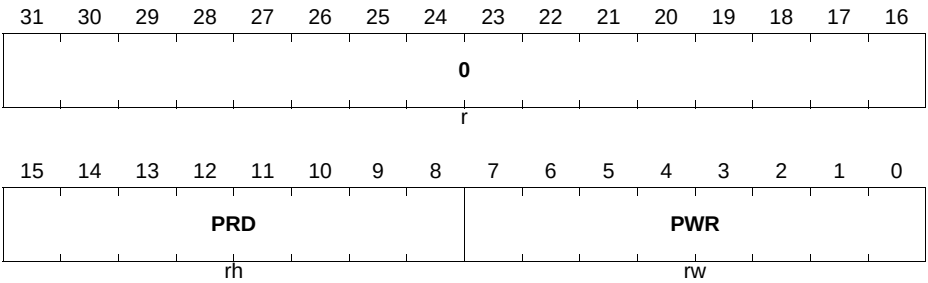
The width and therefore the valid bits in register **PMTPR** is listed in **Table 11-12**.

Table 11-12 Memory Parity Bus Widths

Memory Instance	Number of Parity Bits	Valid Bits in PWR/PRD
Program SRAM (PSRAM)	4	PWR[3:0]/PRD[11:8]
System SRAM (DSRAM1)	4	PWR[3:0]/PRD[11:8]
USIC 0 Buffer (U0)	1	PWR[0]/PRD[8]
USIC 1 Buffer (U1)	1	PWR[0]/PRD[8]
MultiCAN Buffer (MC)	1	PWR[0]/PRD[8]
PMU Prefetch Buffer (PPRF)	4	PWR[3:0]/PRD[11:8]
USB Buffer (USB)	1	PWR[0]/PRD[8]

PMTPR

Parity Memory Test Pattern Register (0154_H) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
PRD	[15:8]	rh	**Parity Read Values for Memory Test** For each byte of a memory module the parity bits generated during the most recent read access are indicated here.
PWR	[7:0]	rw	**Parity Write Values for Memory Test** For each byte of a memory module the parity bits corresponding to the next write access are stored here.
0	[31:16]	r	**Reserved** Should be written with 0.

PMTSR

This register selects parity test output from a memory instance that will be reflected in PRD bit field of **PMTPR** register.

*Note: Only one bit shall be set at the same time in register **PMTPR**. Otherwise the result of the parity software test is unpredictable.*

PMTSR
Parity Memory Test Select Register (0158_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0											**0**		**0**		**MTUSB**
r											r		r		rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		**MTEPPRF**	**MTEMC**	**0**	**0**	**MTEU1**	**MTEU0**	**0**					**0**	**MTENDS1**	**MTENPS**
r		rwh	rwh	r	r	rwh	rwh	r					r	rw	rw

Field	Bits	Type	Description
MTENPS	0	rw	**Test Enable Control for PSRAM** 0_B Standard operation 1_B Parity bits under test

Field	Bits	Type	Description
MTENDS1	1	rw	**Test Enable Control for DSRAM1** 0_B Standard operation 1_B Parity bits under test
MTEU0	8	rwh	**Test Enable Control for USIC0 Memory** 0_B Standard operation 1_B Parity bits under test
MTEU1	9	rwh	**Test Enable Control for USIC1 Memory** 0_B Standard operation 1_B Parity bits under test
MTEMC	12	rwh	**Test Enable Control for MultiCAN Memory** 0_B Standard operation 1_B Parity bits under test
MTEPPRF	13	rwh	**Test Enable Control for PMU Prefetch Memory** 0_B Standard operation 1_B Parity bits under test
MTUSB	16	rw	**Test Enable Control for USB Memory** 0_B Standard operation 1_B Parity bits under test
0	2, [7:3], 10, 11, [15:14], [18:17], [20:19], [31:21]	r	**Reserved** Should be written with 0.

TRAPSTAT

This register contains the status flags for all trap request trigger sources of the SCU. A trap flag is set when a corresponding emergency event occurs. Trap mechanism supports testing and debug of these status bits by software using registers **TRAPSET** and **TRAPCLR**. This register reflects masking with **TRAPDIS** register.

TRAPSTAT
Trap Status Register **(0160_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															0
r															r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	TEMPLOT	TEMPHIT	0			BWERR1T	BWERR0T	ULPWDGT	BRWNT	PET	UVCOLCKT	SVCOLCKT	0	SOSCWDGT
r	r	rh	rh	r			rh	rh	rh	rh	rh	rh	rh	r	rh

Field	Bits	Type	Description
SOSCWDGT	0	rh	**OSC_HP Oscillator Watchdog Trap Status** 0_B No pending trap request 1_B Pending trap request
SVCOLCKT	2	rh	**System VCO Lock Trap Status** 0_B No pending trap request 1_B Pending trap request
UVCOLCKT	3	rh	**USB VCO Lock Trap Status** 0_B No pending trap request 1_B Pending trap request
PET	4	rh	**Parity Error Trap Status** 0_B No pending trap request 1_B Pending trap request
BRWNT	5	rh	**Brown Out Trap Status** 0_B No pending trap request 1_B Pending trap request
ULPWDGT	6	rh	**OSC_ULP Oscillator Watchdog Trap Status** 0_B No pending trap request 1_B Pending trap request
BWERR0T	7	rh	**Peripheral Bridge 0 Trap Status** This trap flags error responses for buffered write operations on the Peripheral Bridge 0 0_B No pending trap request 1_B Pending trap request

Field	Bits	Type	Description
BWERR1T	8	rh	**Peripheral Bridge 1 Trap Status** This trap flags error responses for buffered write operations on the Peripheral Bridge 1 0_B No pending trap request 1_B Pending trap request
TEMPHIT	12	rh	**Die Temperature Too High Trap Status** 0_B No pending trap request 1_B Pending trap request
TEMPLOT	13	rh	**Die Temperature Too Low Trap Status** 0_B No pending trap request 1_B Pending trap request
0	1, [11:9], 15,14, 16, [31:17]	r	**Reserved** Read as 0; should be written with 0.

TRAPRAW

This register contains the status flags for all trap request trigger sources of the SCU before masking with **TRAPDIS**.
A trap flag is set when a corresponding emergency event occurs. For setting and clearing of these status bits by software see registers **TRAPSET** and **TRAPCLR**, respectively.

TRAPRAW
Trap Raw Status Register **(0164_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
							0								**0**
							r								r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**TEMPLOT**	**TEMPHIT**	**0**			**BWERR1T**	**BWERR0T**	**ULPWDGT**	**BRWNT**	**PET**	**UVCOLCKT**	**SVCOLCKT**	**0**	**SOSCWDGT**
r	r	rh	rh	r			rh	rh	rh	rh	rh	rh	rh	r	rh

Field	Bits	Type	Description
SOSCWDGT	0	rh	**OSC_HP Oscillator Watchdog Trap Raw Status** 0_B No pending trap request 1_B Pending trap request
SVCOLCKT	2	rh	**System VCO Lock Trap Raw Status** 0_B No pending trap request 1_B Pending trap request
UVCOLCKT	3	rh	**USB VCO Lock Trap Raw Status** 0_B No pending trap request 1_B Pending trap request
PET	4	rh	**Parity Error Trap Raw Status** 0_B No pending trap request 1_B Pending trap request
BRWNT	5	rh	**Brown Out Trap Raw Status** 0_B No pending trap request 1_B Pending trap request
ULPWDGT	6	rh	**OSC_ULP Oscillator Watchdog Trap Raw Status** 0_B No pending trap request 1_B Pending trap request
BWERR0T	7	rh	**Peripheral Bridge 0 Trap Raw Status** 0_B No pending trap request 1_B Pending trap request
BWERR1T	8	rh	**Peripheral Bridge 1 Trap Raw Status** 0_B No pending trap request 1_B Pending trap request
TEMPHIT	12	rh	**Die Temperature Too High Trap Raw Status** 0_B No pending trap request 1_B Pending trap request
TEMPLOT	13	rh	**Die Temperature Too Low Trap Raw Status** 0_B No pending trap request 1_B Pending trap request
0	1, [11:9], 15,14, 16, [31:17]	r	**Reserved** Read as 0; should be written with 0.

TRAPDIS

Disable corresponding traps.

TRAPDIS

Trap Disable Register **(0168_H)** **Reset Value: 0000 31FF$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															0
r															r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	TEMPLOT	TEMPHIT	0			BWERR1T	BWERR0T	ULPWDGT	BRWNT	PET	UVCOLCKT	SVCOLCKT	0	SOSCWDGT
r	r	rw	rw	r			rw	rw	rw	rw	rw	rw	rw	r	rw

Field	Bits	Type	Description
SOSCWDGT	0	rw	**OSC_HP Oscillator Watchdog Trap Disable** 0_B Trap request enabled 1_B Trap request disabled
SVCOLCKT	2	rw	**System VCO Lock Trap Disable** 0_B Trap request enabled 1_B Trap request disabled
UVCOLCKT	3	rw	**USB VCO Lock Trap Disable** 0_B Trap request enabled 1_B Trap request disabled
PET	4	rw	**Parity Error Trap Disable** 0_B Trap request enabled 1_B Trap request disabled
BRWNT	5	rw	**Brown Out Trap Disable** 0_B Trap request enabled 1_B Trap request disabled
ULPWDGT	6	rw	**OSC_ULP Oscillator Watchdog Trap Disable** 0_B Trap request enabled 1_B Trap request disabled
BWERR0T	7	rw	**Peripheral Bridge 0 Trap Disable** 0_B Trap request enabled 1_B Trap request disabled

Field	Bits	Type	Description
BWERR1T	8	rw	**Peripheral Bridge 1 Trap Disable** 0_B Trap request enabled 1_B Trap request disabled
TEMPHIT	12	rw	**Die Temperature Too High Trap Disable** 0_B Trap request enabled 1_B Trap request disabled
TEMPLOT	13	rw	**Die Temperature Too Low Trap Disable** 0_B Trap request enabled 1_B Trap request disabled
0	1, [11:9], 15,14, 16, [31:17]	r	**Reserved** Read as 0; should be written with 0.

TRAPCLR

This register contains the software clear control for the trap status flags in register **TRAPRAW** and **TRAPSTAT**.

TRAPCLR
Trap Clear Register **($016C_H$)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
							0								**0**
							r								r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**TEMPLOT**	**TEMPHIT**	**0**			**BWERR1T**	**BWERR0T**	**ULPWDGT**	**BRWNT**	**PET**	**UVCOLCKT**	**SVCOLCKT**	**0**	**SOSCWDGT**
r	r	w	w	r			w	w	w	w	w	w	w	r	w

Field	Bits	Type	Description
SOSCWDGT	0	w	**OSC_HP Oscillator Watchdog Trap Clear** 0_B No effect 1_B Clear trap request

Field	Bits	Type	Description
SVCOLCKT	2	w	**System VCO Lock Trap Clear** 0_B No effect 1_B Clear trap request
UVCOLCKT	3	w	**USB VCO Lock Trap Clear** 0_B No effect 1_B Clear trap request
PET	4	w	**Parity Error Trap Clear** 0_B No effect 1_B Clear trap request
BRWNT	5	w	**Brown Out Trap Clear** 0_B No effect 1_B Clear trap request
ULPWDGT	6	w	**OSC_ULP Oscillator Watchdog Trap Clear** 0_B No effect 1_B Clear trap request
BWERR0T	7	w	**Peripheral Bridge 0 Trap Clear** 0_B No effect 1_B Clear trap request
BWERR1T	8	w	**Peripheral Bridge 1 Trap Clear** 0_B No effect 1_B Clear trap request
TEMPHIT	12	w	**Die Temperature Too High Trap Clear** 0_B No effect 1_B Clear trap request
TEMPLOT	13	w	**Die Temperature Too Low Trap Clear** 0_B No effect 1_B Clear trap request
0	1, [11:9], 15,14, 16, [31:17]	r	**Reserved** Read as 0; should be written with 0.

TRAPSET

This register contains the software set control for the trap status flags in register TRAPRAW.

TRAPSET
Trap Set Register **(0170_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															0
r															r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	TEMPLOT	TEMPHIT	0			BWERR1T	BWERR0T	ULPWDT	BRWNT	PET	UVCOLCKT	SVCOLCKT	0	SOSCWDGT
r	r	w	w	r			w	w	w	w	w	w	w	r	w

Field	Bits	Type	Description
SOSCWDGT	0	w	**OSC_HP Oscillator Watchdog Trap Set** 0_B No effect 1_B Set trap request
SVCOLCKT	2	w	**System VCO Lock Trap Set** 0_B No effect 1_B Set trap request
UVCOLCKT	3	w	**USB VCO Lock Trap Set** 0_B No effect 1_B Set trap request
PET	4	w	**Parity Error Trap Set** 0_B No effect 1_B Set trap request
BRWNT	5	w	**Brown Out Trap Set** 0_B No effect 1_B Set trap request
ULPWDT	6	w	**OSC_ULP Oscillator Watchdog Trap Set** 0_B No effect 1_B Set trap request
BWERR0T	7	w	**Peripheral Bridge 0 Trap Set** 0_B No effect 1_B Set trap request
BWERR1T	8	w	**Peripheral Bridge 1 Trap Set** 0_B No effect 1_B Set trap request

Field	Bits	Type	Description
TEMPHIT	12	w	**Die Temperature Too High Trap Set** 0_B No effect 1_B Set trap request
TEMPLOT	13	w	**Die Temperature Too Low Trap Set** 0_B No effect 1_B Set trap request
0	1, [11:9], 15,14, 16, [31:17]	r	**Reserved** Read as 0; should be written with 0.

PWRSTAT

Power status register.

PWRSTAT
PCU Status Register **(0200_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
						0							**USB PUWQ**	**0**	**USB PHY PDQ**
						r							r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
						0								**0**	**HIBEN**
						r								r	r

Field	Bits	Type	Description
HIBEN	0	r	**Hibernate Domain Enable Status** 0_B Inactive 1_B Active
USBPHYPDQ	16	r	**USB PHY Transceiver State** 0_B Power-down 1_B Active

Field	Bits	Type	Description
USBPUWQ	18	r	**USB Weak Pull-Up at PADN State** 0_B Pull-up active 1_B Pull-up not active
0	1, [15:2], 17, [31:19]	r	**Reserved**

11.10.2 PCU Registers

PWRSTAT

Power status register.

PWRSTAT
PCU Status Register **(0200_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
						0							**USBPUWQ**	**0**	**USBPHYPDQ**
						r							r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
						0								**0**	**HIBEN**
						r								r	r

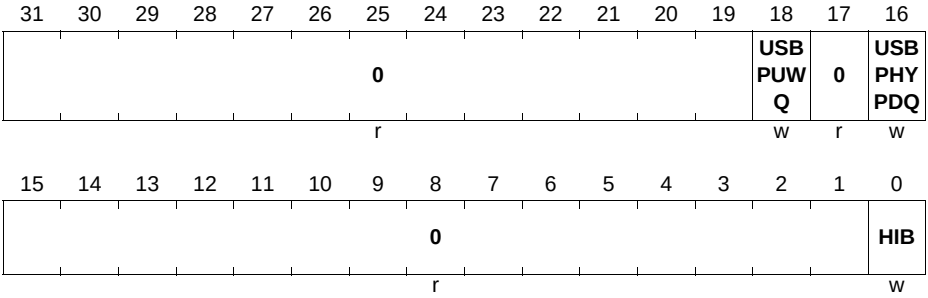
Field	Bits	Type	Description
HIBEN	0	r	**Hibernate Domain Enable Status** 0_B Inactive 1_B Active
USBPHYPDQ	16	r	**USB PHY Transceiver State** 0_B Power-down 1_B Active
USBPUWQ	18	r	**USB Weak Pull-Up at PADN State** 0_B Pull-up active 1_B Pull-up not active

Field	Bits	Type	Description
0	1, [15:2], 17, [31:19]	r	**Reserved**

PWRSET

Power control register. Write one to set, writing zeros have no effect.

PWRSET
PCU Set Control Register **(0204_H)** **Reset Value: 0000 0000$_H$**

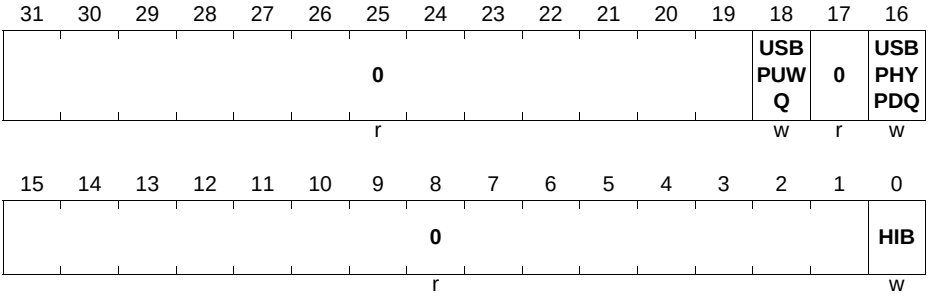


Field	Bits	Type	Description
HIB	0	w	**Set Hibernate Domain Enable** 0_B No effect 1_B Enable Hibernate domain
USBPHYPDQ	16	w	**Set USB PHY Transceiver Disable** 0_B No effect 1_B Active
USBPUWQ	18	w	**Set USB Weak Pull-Up at PADN Enable** 0_B No effect 1_B Pull-up not active
0	[15:1], 17, [31:19]	r	**Reserved**

PWRCLR

Power control register. Write one to clear, writing zeros have no effect.

PWRCLR
PCU Clear Control Register **(0208_H)** **Reset Value: 0000 0000$_H$**



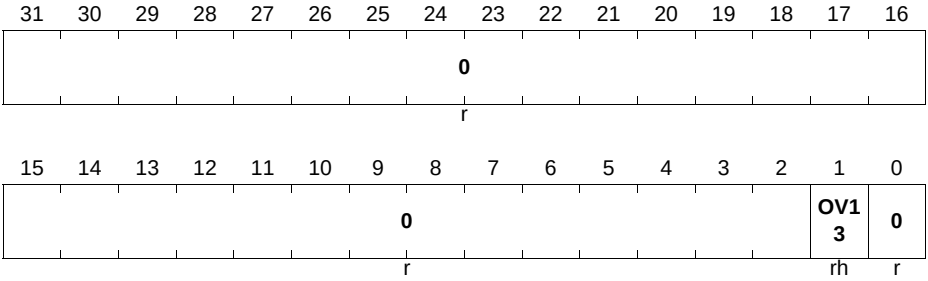
Field	Bits	Type	Description
HIB	0	w	**Clear Disable Hibernate Domain** 0_B No effect 1_B Disable Hibernate domain
USBPHYPDQ	16	w	**Clear USB PHY Transceiver Disable** 0_B No effect 1_B Power-down
USBPUWQ	18	w	**Clear USB Weak Pull-Up at PADN Enable** 0_B No effect 1_B Pull-up active
0	[15:1], 17, [31:19]	r	**Reserved**

EVRSTAT

EVR status register.

EVRSTAT

EVR Status Register **(0210_H)** **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
OV13	1	rh	**Regulator Overvoltage for 1.3 V** 0_B No overvoltage condition 1_B Regulator is in overvoltage
0	0, [31:2]	r	**Reserved**

EVRVADCSTAT

Supply voltage monitor register. The actual voltage represented by the VADC13V and VADC33V can be calculated according to the formulas:

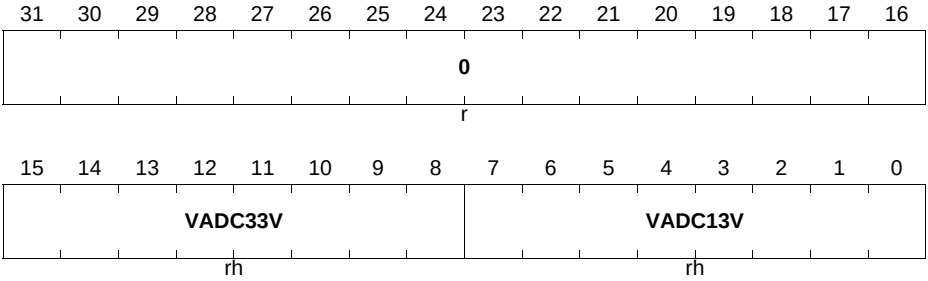
VADC13V = (V_{DDC} / LSB13V) +1 (11.7)

where LSB13V is 5.8 mV

VADC33V = (V_{DDP} / LSB33V) +1 (11.8)

where LSB33V is 22.5 mV

EVRVADCSTAT
EVR VADC Status Register **(0214_H)** **Reset Value: 0000 0000$_H$**

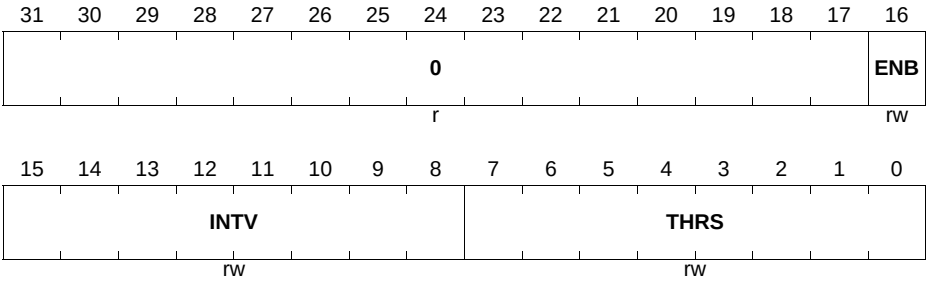


Field	Bits	Type	Description
VADC13V	[7:0]	rh	**VADC 1.3 V Conversion Result** This bit field contains the last conversion result of the VADC for the EVR13.
VADC33V	[15:8]	rh	**VADC 3.3 V Conversion Result** This bit field contains the last conversion result of the VADC for the EVR33. The value is used for brown-out detection
0	[31:16]	r	**Reserved** Read as 0.

PWRMON

Power monitoring control register for brown-out detection.

PWRMON
Power Monitor Control **(022C$_H$)** **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
THRS	[7:0]	rw	**Threshold** Threshold value for comparison to V_{DDP} for brown-out detection
INTV	[15:8]	rw	**Interval** Interval value for comparison to V_{DDP} expressed in cycles of system clock
ENB	16	rw	**Enable** Enable of comparison and interrupt generation
0	[31:17]	r	**Reserved**

11.10.3 HCU Registers

HDSTAT

Hibernate domain status register

HDSTAT

Hibernate Domain Status Register **(0300_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0								0							
r								r							

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		0		AHIBIO0NEV	AHIBIO0PEV	VBATNEV	VBATPEV	0			HIBNOUT	ULPWDG	RTCEV	ENEV	EPEV
r		r		rh	rh	rh	rh	r			rh	rh	rh	rh	rh

Field	Bits	Type	Description
EPEV	0	rh	**Wake-up Pin Event Positive Edge** 0_B Wake-up on positive edge pin event inactive 1_B Wake-up on positive edge pin event active
ENEV	1	rh	**Wake-up Pin Event Negative Edge** 0_B Wake-up on negative edge pin event inactive 1_B Wake-up on negative edge pin event active
RTCEV	2	rh	**RTC Event** 0_B Wake-up on RTC event inactive 1_B Wake-up on RTC event active
ULPWDG	3	rh	**ULP WDG Alarm Status** 0_B Watchdog alarm did not occur 1_B Watchdog alarm occurred
HIBNOUT	4	rh	**Hibernate Control Status** 0_B Hibernate not driven active to pads 1_B Hibernate driven active to pads
VBATPEV	8	rh	**Wake-Up on LPAC Positive Edge of V_{BAT} Threshold Crossing** 0_B Wake-up on rising above threshold event inactive 1_B Wake-up on rising above threshold event active

Field	Bits	Type	Description
VBATNEV	9	rh	**Wake-Up on LPAC Negative Edge of V_{BAT} Threshold Crossing** 0_B Wake-up on falling below threshold event inactive 1_B Wake-up on falling below threshold event active
AHIBIO0PEV	10	rh	**Wake-Up on LPAC Positive Edge of HIB_IO_0 Threshold Crossing** 0_B Wake-up on rising above threshold event inactive 1_B Wake-up on rising above threshold event active
AHIBIO0NEV	11	rh	**Wake-Up on LPAC Negative Edge of HIB_IO_0 Threshold Crossing** 0_B Wake-up on falling below threshold event inactive 1_B Wake-up on falling below threshold event active
0	[7:5], [13:12], [30:14], 31	r	**Reserved**

HDCLR

Hibernate domain clear status register. Write one to clear, writing zeros has no effect.

HDCLR

Hibernate Domain Status Clear Register (0304_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0								**0**							
r								r							

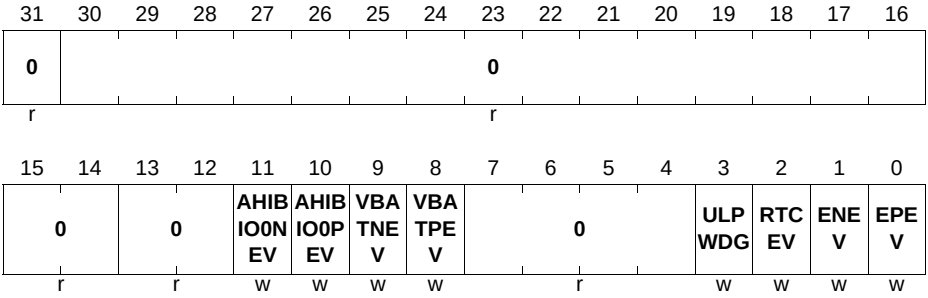
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		**0**		**AHIBIO0NEV**	**AHIBIO0PEV**	**VBATNEV**	**VBATPEV**	**0**				**ULPWDG**	**RTCEV**	**ENEV**	**EPEV**
r		r		w	w	w	w	r				w	w	w	w

Field	Bits	Type	Description
EPEV	0	w	**Wake-up Pin Event Positive Edge Clear** 0_B No effect 1_B Clear wake-up event
ENEV	1	w	**Wake-up Pin Event Negative Edge Clear** 0_B No effect 1_B Clear wake-up event
RTCEV	2	w	**RTC Event Clear** 0_B No effect 1_B Clear wake-up event
ULPWDG	3	w	**ULP WDG Alarm Clear** 0_B No effect 1_B Clear watchdog alarm
VBATPEV	8	w	**Wake-Up on LPAC Positive Edge of V_{BAT} Threshold Crossing Clear** 0_B No effect 1_B Clear wake-up event
VBATNEV	9	w	**Wake-Up on LPAC Negative Edge of V_{BAT} Threshold Crossing Clear** 0_B No effect 1_B Clear wake-up event
AHIBIO0PEV	10	w	**Wake-Up on LPAC Positive Edge of HIB_IO_0 Threshold Crossing Clear** 0_B No effect 1_B Clear wake-up event
AHIBIO0NEV	11	w	**Wake-Up on LPAC Negative Edge of HIB_IO_0 Threshold Crossing Clear** 0_B No effect 1_B Clear wake-up event
0	[7:4], [13:12], [30:14], 31	r	**Reserved** Read as 0; should be written with 0.

HDSET

Hibernate domain set status register. Write one to set, writing zeros has no effect.

HDSET
Hibernate Domain Status Set Register (0308_H) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
EPEV	0	w	**Wake-up Pin Event Positive Edge Set** 0_B No effect 1_B Set wake-up event
ENEV	1	w	**Wake-up Pin Event Negative Edge Set** 0_B No effect 1_B Set wake-up event
RTCEV	2	w	**RTC Event Set** 0_B No effect 1_B Set wake-up event
ULPWDG	3	w	**ULP WDG Alarm Set** 0_B No effect 1_B Set watchdog alarm
VBATPEV	8	w	**Wake-Up on LPAC Positive Edge of V_{BAT} Threshold Crossing Set** 0_B No effect 1_B Set wake-up event
VBATNEV	9	w	**Wake-Up on LPAC Negative Edge of V_{BAT} Threshold Crossing Set** 0_B No effect 1_B Set wake-up event
AHIBIO0PEV	10	w	**Wake-Up on LPAC Positive Edge of HIB_IO_0 Threshold Crossing Set** 0_B No effect 1_B Set wake-up event

Field	Bits	Type	Description
AHIBIO0NEV	11	w	**Wake-Up on LPAC Negative Edge of HIB_IO_0 Threshold Crossing Set** 0_B No effect 1_B Set wake-up event
0	[7:4], [13:12], [30:14], 31	r	**Reserved** Read as 0; should be written with 0.

HDCR

Hibernate domain configuration register.

HDCR

Hibernate Domain Control Register (030C$_H$) **Reset Value: 000C 2000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0		**0**		**AHIBIO0HI**	**AHIBIO0LO**	**VBATHI**	**VBATLO**	**0**				**HIBIO0SEL**			
r		r		rw	rw	rw	rw	r				rw			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**ADIG0SEL**	**0**	**HIBIO0POL**	**0**	**GPI0SEL**	**0**	**WKUPSEL**	**STDBYSEL**	**RCS**	**XTALGPI1SEL**	**HIB**	**ULPWDGEN**	**RTCE**	**WKPEN**	**WKPEP**
r	rw	r	rw	r	rw	r	rw	rw	rw	rw	rwh	rw	rw	rw	rw

Field	Bits	Type	Description
WKPEP	0	rw	**Wake-Up on Pin Event Positive Edge Enable** 0_B Wake-up event disabled 1_B Wake-up event enabled
WKPEN	1	rw	**Wake-up on Pin Event Negative Edge Enable** 0_B Wake-up event disabled 1_B Wake-up event enabled
RTCE	2	rw	**Wake-up on RTC Event Enable** 0_B Wake-up event disabled 1_B Wake-up event enabled

Field	Bits	Type	Description
ULPWDGEN	3	rw	**ULP WDG Alarm Enable** 0_B Wake-up event disabled 1_B Wake-up event enabled
HIB	4	rwh	**Hibernate Request Value Set** 0_B External hibernate request inactive 1_B External hibernate request active *Note: This bit get automatically cleared by hardware upon occurrence of any wake-up event enabled in this register*
XTALGPI1SEL	5	rw	**Multiplex Control for RTC_XTAL_1 Select as GPI Input** 0_B RTC_XTAL_1 input selected 1_B Analog comparator output for HIB_IO_1 or pre-selected digital IO input
RCS	6	rw	**f_{RTC} Clock Selection** 0_B f_{OSI} selected 1_B f_{ULP} selected
STDBYSEL	7	rw	**f_{STDBY} Clock Selection** 0_B f_{OSI} selected 1_B f_{ULP} selected
WKUPSEL	8	rw	**Wake-Up from Hibernate Trigger Input Selection** 0_B HIB_IO_1 pin selected 1_B HIB_IO_0 pin selected
GPI0SEL	10	rw	**General Purpose Input 0 Selection** This bit field selects input to ERU0 module that optionally can be used with software as a general purpose input. 0_B reserved 1_B HIB_IO_0 pin selected
HIBIO0POL	12	rw	**HIBIO0 Polarity Set** Selects the output polarity of the HIBIO0 0_B Direct value 1_B Inverted value
ADIG0SEL	14	rw	**Select Analog Channel 0 or Digital Output Path** Selects the output from analog comparator latch or from pre-selected digital IO input 0_B Digital input 1_B Analog comparator result for HIB_IO_0

Field	Bits	Type	Description
HIBIO0SEL	[19:16]	rw	**HIB_IO_0 Pin I/O Control (default HIBOUT)** This bit field determines the Port n line x functionality. 0000_B Direct input, No input pull device connected 0001_B Direct input, Input pull-down device connected 0010_B Direct input, Input pull-up device connected 1000_B Push-pull HIB Control output 1001_B Push-pull WDT service output 1010_B Push-pull GPIO output 1100_B Open-drain HIB Control output 1101_B Open-drain WDT service output 1110_B Open-drain GPIO output 1111_B Analog input
VBATLO	24	rw	**Wake-Up on V_{BAT} Falling Below Threshold Enable** 0_B Wake-up event disabled 1_B Wake-up event enabled
VBATHI	25	rw	**Wake-Up on V_{BAT} Rising Above Threshold Enable** 0_B Wake-up event disabled 1_B Wake-up event enabled
AHIBIO0LO	26	rw	**Wake-Up on Analog HIB_IO_0 Falling Below Threshold Enable** 0_B Wake-up event disabled 1_B Wake-up event enabled
AHIBIO0HI	27	rw	**Wake-Up on Analog HIB_IO_0 Rising Above Threshold Enable** 0_B Wake-up event disabled 1_B Wake-up event enabled
0	9, 11, 13, 15, [23:20], [29:28], [31:30]	r	**Reserved** Read as 0; should be written with 0.

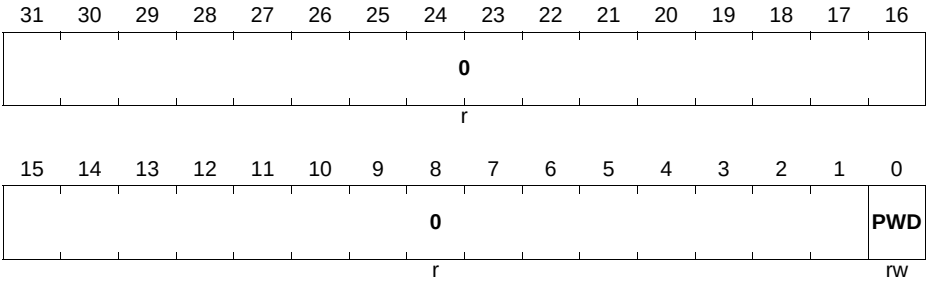
OSCSICTRL

Control register for f_{OSI} clock source. A special mechanism keeps the the f_{OSI} clock active if the external crystal oscillator is switched off, regardless of the value of the PWD bit

field. The f_{OSI} can be switched off only if the external crystal oscillator is enabled and the f_{ULP} clock toggling.

OSCSICTRL

f_{OSI} Control Register **(0314_H)** **Reset Value: $0000\ 0001_H$**

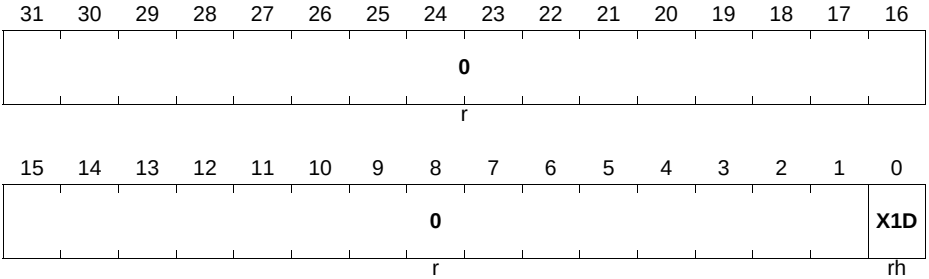


Field	Bits	Type	Description
PWD	0	rw	**Turn OFF the f_{OSI} Clock Source** 0_B Enabled 1_B Disabled *Note: The f_{OSI} needs to be enabled in order to prevent potential deadlock in case of external crystal failure.*
0	[31:1]	r	**Reserved** Read as 0; should be written with 0.

OSCULSTAT

Status register of the OSC_ULP oscillator.

OSCULSTAT
OSC_ULP Status Register **(0318$_H$)** **Reset Value: 0000 0000$_H$**

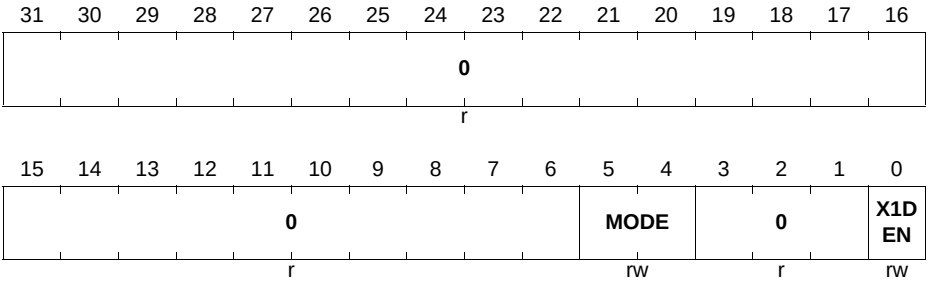


Field	Bits	Type	Description
X1D	0	rh	**XTAL1 Data Value** This bit monitors the value (level) of pin XTAL1. If XTAL1 is not used as clock input it can be used as GPI pin. This bit is only updated if X1DEN is set.
0	[31:1]	r	**Reserved**

OSCULCTRL

Control register for OSC_ULP oscillator. This register allows selection of clock generation with external crystal, direct clock input, or power down mode. Alternate GPI function of the pin is also controlled with this register.

OSCULCTRL
OSC_ULP Control Register **(031C$_H$)** **Reset Value: 0000 0020$_H$**



Field	Bits	Type	Description
X1DEN	0	rw	**XTAL1 Data General Purpose Input Enable** The GPI data can be monitored with X1D bit of **OSCULSTAT** register 0_B Data input inactivated, power down 1_B Data input active *Note: It is strongly recommended to keep this function inactivated if the XTAL1 input is used as clock source*
MODE	[5:4]	rw	**Oscillator Mode** 00_B Oscillator is enabled, in operation 01_B Oscillator is enabled, in bypass mode 10_B Oscillator in power down 11_B Oscillator in power down, can be used as GPI *Note: Use of the oscillator input require that X1DEN bit is activated*
0	[3:1], [31:6]	r	**Reserved** Read as 0; should be written with 0.

LPACCONF

LPAC module configuration register.

LPACCONF

Analog Wake-up Configuration Register (0320_H) **Reset Value: $7000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SETTLECNT				INTERVCNT											
rw				rw											

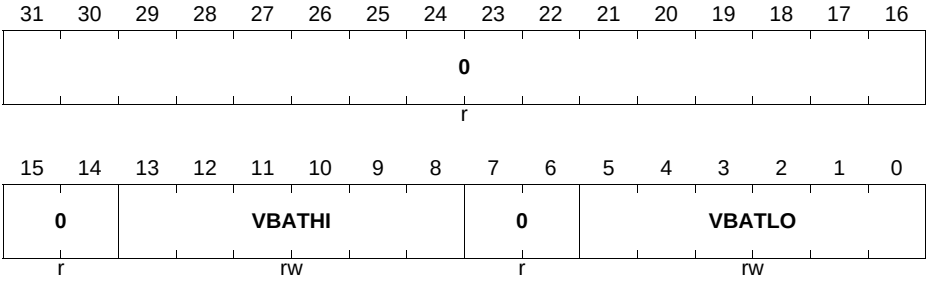
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0			CONVDEL	0					TRIGSEL			0	CMPEN		
r			rw	r					rw			r	rw		

Field	Bits	Type	Description
CMPEN	[2:0]	rw	**Compare Enable for Input Selection** 000_B Comparator permanently in power down 001_B Comparator activated for V_{BAT} input 010_B Comparator activated for HIB_IO_0 input 100_B Comparator activated for HIB_IO_1 input *Note: A combination of bits may be set in order to activate compare operations for multiple inputs*
TRIGSEL	[6:4]	rw	**Analog Compare Trigger Select** Select compare start trigger for inputs selected with CMPEN bit field 000_B Sub-second interval counter 001_B RTC alarm event 010_B RTC periodic event 011_B On digital WKUP input positive edge event 101_B On digital WKUP input negative edge event 110_B Continuous measurement 111_B Single-shot on software request
CONVDEL	12	rw	**Conversion Delay** Configure conversion delay
INTERVCNT	[27:16]	rw	**Sub-second Interval Counter** Configure compare interval of: (INTERVCNT + 0x10)*(32.768 kHz clock period) [s]
SETTLECNT	[31:28]	rw	**LPAC Settle Time Counter** Configure settling time of LPAC after powered up before measurement start: (SETTLECNT + 1) *(32.768 kHz clock period)[s]
0	3, [11:7], [15:13]	r	**Reserved** Read as 0; should be written with 0.

LPACTH0

LPAC module threshold configuration register for alternate battery voltage supply monitoring.

LPACTH0
LPAC Threshold Register 0 **(0324_H)** **Reset Value: $0000\ 0000_H$**

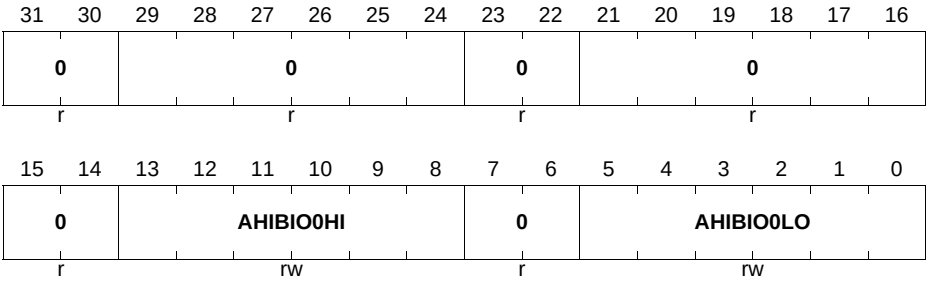


Field	Bits	Type	Description
VBATLO	[5:0]	rw	**V_{BAT} Lower Threshold Value** Threshold value reflecting lower hysteresis limit
VBATHI	[13:8]	rw	**V_{BAT} Upper Threshold Value** Threshold value reflecting upper hysteresis limit
0	[7:6], [31:14]	r	**Reserved** Read as 0; should be written with 0.

LPACTH1

LPAC module threshold configuration register for analog input voltage monitoring.

LPACTH1
LPAC Threshold Register 1 **(0328_H)** **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
AHIBIO0LO	[5:0]	rw	**Analog HIB_IO_0 Lower Threshold Value** Threshold value reflecting lower hysteresis limit
AHIBIO0HI	[13:8]	rw	**Analog HIB_IO_0 Upper Threshold Value** Threshold value reflecting upper hysteresis limit
0	[7:6], [15:14], [21:16], [23:22], [29:24], [31:30]	r	**Reserved** Read as 0; should be written with 0.

LPACST

LPAC module configuration state register.

LPACST

Hibernate Analog Control State Register ($032C_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
						0							**0**	**AHIBIO0VAL**	**VBATVALL**
						r							r	rh	rh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
						0							**0**	**AHIBIO0SCMP**	**VBATSCMP**
						r							r	rh	rh

Field	Bits	Type	Description
VBATSCMP	0	rh	**Trigger V_{BAT} Single Compare Operation Status** Sticky bit indicating that compare operation on V_{BAT} voltage compare completed. This bit gets cleared if new single shot requested. 0_B Ready to start new compare operation 1_B Compare operation completed *Note: Single shot trigger V_{BAT} measurement needs to be enabled for LPAC with **LPACCONF** register*
AHIBIO0SCMP	1	rh	**Trigger HIB_IO_0 Input Single Compare Operation Status** Sticky bit indicating that compare operation on HIB_IO_0 voltage compare operation completed. This bit gets cleared if new single shot requested. 0_B Ready to start new compare operation 1_B Compare operation completed *Note: Single shot trigger HIB_IO_0 measurement must be enabled for LPAC with **LPACCONF** register*
VBATVAL	16	rh	**V_{BAT} Compare Operation Result** 0_B Below programmed threshold 1_B Above programmed threshold
AHIBIO0VAL	17	rh	**HIB_IO_0 Input Compare Operation Result** 0_B Below programmed threshold 1_B Above programmed threshold
0	2,18, [15:3], [31:19]	r	**Reserved**

LPACCLR

LPAC module configuration clear register. Write one to clear, writing zero is ignored.

LPACCLR
LPAC Control Clear Register **(0330_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
						0							**0**	**AHIBIO0VAL**	**VBATVAL**
						r							r	w	w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
						0							**0**	**AHIBIO0SCMP**	**VBATSCMP**
						r							r	w	w

Field	Bits	Type	Description
VBATSCMP	0	w	**Trigger V_{BAT} Single Compare Operation Clear** Clear sticky bit indicating that compare operation on V_{BAT} voltage compare completed. 0_B No effect 1_B Clear the sticky bit
AHIBIO0SCMP	1	w	**Trigger HIB_IO_0 Input Single Compare Operation Clear** Clear sticky bit indicating that compare operation on HIB_IO_0 voltage compare operation completed. 0_B No effect 1_B Clear the sticky bit
VBATVAL	16	w	**V_{BAT} Compare Operation Initial Value Clear** 0_B No effect 1_B Below programmed threshold
AHIBIO0VAL	17	w	**HIB_IO_0 Input Compare Initial Value Clear** 0_B No effect 1_B Below programmed threshold
0	2, 18, [15:3], [31:19]	r	**Reserved** Read as 0; should be written with 0.

LPACSET

LPAC module configuration set register. Write one to set, writing zero is ignored.

LPACSET
LPAC Control Set Register **(0334_H)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
						0							**0**	**AHIBIO0VAL**	**VBATVAL**
						r							r	w	w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
						0							**0**	**AHIBIO0SCMP**	**VBATSCMP**
						r							r	w	w

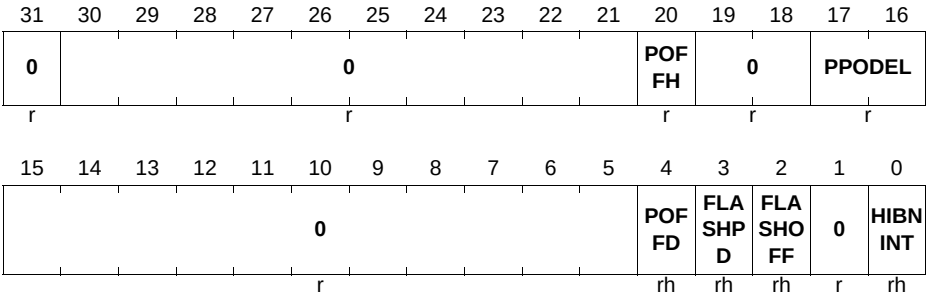
Field	Bits	Type	Description
VBATSCMP	0	w	**Trigger V_{BAT} Single Compare Operation Set** Request single compare operation on V_{BAT} voltage. 0_B No effect 1_B Start compare operation
AHIBIO0SCMP	1	w	**Trigger HIB_IO_0 Input Single Compare Operation Set** Request single compare operation HIB_IO_0 voltage. 0_B No effect 1_B Start compare operation
VBATVAL	16	w	**V_{BAT} Compare Operation Initial Value Set** 0_B No effect 1_B Above programmed threshold
AHIBIO0VAL	17	w	**HIB_IO_0 Input Compare Initial Value Set** 0_B No effect 1_B Above programmed threshold
0	2, 18, [15:3], [31:19]	r	**Reserved** Read as 0; should be written with 0.

HINTST

Control state register internally controlled hibernate mode. This register reflects actual state of the hardware that may change with some delay after some control bits have been modified with software.

HINTST

Hibernate Internal Control State Register (0338_H) **Reset Value: $0010\ 0000_H$**



Field	Bits	Type	Description
HIBNINT	0	rh	**Internally Controlled Hibernate Sequence Request State** Status bit indicating that internally controlled hibernate mode has been effectively entered. Setting VCOREOFF bit field also results in setting this bit. 0_B Hibernate not entered 1_B Hibernate entered
FLASHOFF	2	rh	**V_{DDP} Supply Switch of Flash State** 0_B V_{DDP} supply of Flash switched on 1_B V_{DDP} supply of Flash switched off
FLASHPD	3	rh	**Flash Power Down State** 0_B Normal mode 1_B Power down mode effectively entered *Note: Flash power down can be requested directly with this field, or, indirectly by HIBNINT field*
POFFD	4	rh	**PORST Pull-up OFF Direct Control State** 0_B Pull-up on 1_B Pull-up off
PPODEL	[17:16]	r	**Delay on PORST Pull-up Switching OFF on Hibernate Request**

Field	Bits	Type	Description
POFFH	20	r	**PORST Pull-up OFF in Hibernate Mode State** 0_B Pull-up on 1_B Pull-up off *Note: This setting takes effect after entering hibernate mode*
0	1, [15:5], [19:18], [30:21], 31	r	**Reserved**

HINTCLR

Clear register for internally controlled hibernate mode. Write one to clear, writing zero is ignored.

HINTCLR
Hibernate Internal Control Clear Register ($033C_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0											POFFH	0		PPODEL	
r											w	r		w	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0											POFFD	FLASHPD	FLASHOFF	0	HIBNINT
r											w	w	w	r	w

Field	Bits	Type	Description
HIBNINT	0	w	**Internally Controlled Hibernate Sequence Request Clear** 0_B No effect 1_B Hibernate bit clear *Note: This field clears internally controlled hibernate mode. Intended for testing purpose.*

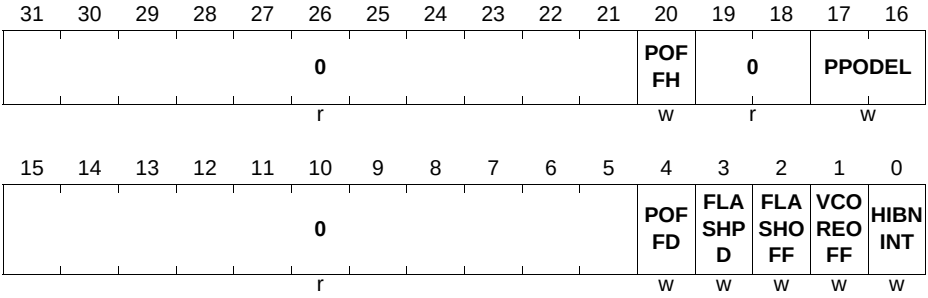
Field	Bits	Type	Description
FLASHOFF	2	w	**V_{DDP} Supply Switch of Flash Clear** 0_B No effect 1_B Switch on V_{DDP} supply of Flash *Note: Flash power off state leave can be requested directly with this field or indirectly by wake-up sequence*
FLASHPD	3	w	**Flash Power Down Clear** 0_B No effect 1_B Flash power down mode leave request *Note: Flash power down state leave can be requested directly with this field or, or, indirectly by wake-up sequence*
POFFD	4	w	**PORST Pull-up OFF Direct Control Clear** This setting has effect immediately 0_B No effect 1_B Pull-up on
PPODEL	[17:16]	w	**Delay on PORST Pull-up Switching OFF on Hibernate Request Clear** Write one to selected bits to clear. Writing zero is ignored.
POFFH	20	w	**PORST Pull-up OFF in Hibernate Mode Clear** This setting takes effect after entering hibernate mode 0_B No effect 1_B Pull-up on
0	1, [15:5], [19:18], [31:21]	r	**Reserved** Read as 0; should be written with 0.

HINTSET

Set register for internally controlled hibernate mode. Write one to set, writing zero is ignored.

HINTSET

Hibernate Internal Control Set Register (0340_H) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
HIBNINT	0	w	**Internally Controlled Hibernate Sequence Request Set** 0_B No effect 1_B Hardware controlled hibernate sequence request active *Note: This field requests internally controlled hibernate mode*
VCOREOFF	1	w	**V_{DDC} Generation off on EVR Set** 0_B No effect 1_B V_{DDC} off to EVR set *Note: for test/debug purpose only* *Note: V_{DDC} off can be requested directly with this field or indirectly by HIBNINT field*
FLASHOFF	2	w	**V_{DDP} Supply Switch of Flash Set** 0_B No effect 1_B Switch off V_{DDP} supply of Flash *Note: Flash power off can be requested directly with this field, or, indirectly by HIBNINT field*
FLASHPD	3	w	**Flash Power Down Set** 0_B No effect 1_B Flash power down mode request set *Note: Flash power down can be requested directly with this field, or, indirectly by HIBNINT field*

Field	Bits	Type	Description
POFFD	4	w	**PORST Pull-up OFF Direct Control Set** 0_B No effect 1_B Pull-up off *Note: This setting takes effect immediately after programming*
PPODEL	[17:16]	w	**Delay on PORST Pull-up Switching OFF on Hibernate Request Set** Write one to selected bits to set them. Writing zero is ignored.
POFFH	20	w	**PORST Pull-up OFF in Hibernate Mode Set** 0_B No effect 1_B Pull-up off *Note: This setting takes effect after entering hibernate mode*
0	[15:5], [19:18], [31:21]	r	**Reserved** Read as 0; should be written with 0.

11.10.4 RCU Registers

RSTSTAT

Reset status register. This register needs to be checked after system startup in order to determine last reset reason.

RSTSTAT
RCU Reset Status **(0400_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				0	LCK EN	HIBR S	HIB WK	RSTSTAT							
r				r	r	r	rh	rh							

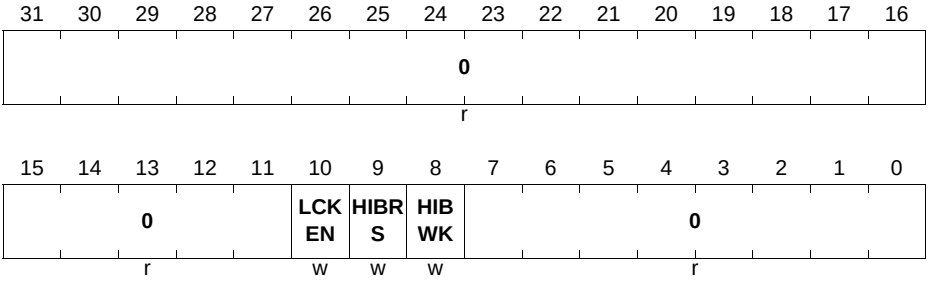
Field	Bits	Type	Description
RSTSTAT	[7:0]	rh	**Reset Status Information** Provides reason of last reset 00000001_B PORST reset 00000010_B SWD reset 00000100_B PV reset 00001000_B CPU system reset 00010000_B CPU lockup reset 00100000_B WDT reset 01000000_B Reserved 10000000_B Parity Error reset
HIBWK	8	rh	**Hibernate Wake-up Status** 0_B No Wake-up 1_B Wake-up event *Note: Field is cleared with enable of Hibernate mode*
HIBRS	9	r	**Hibernate Reset Status** 0_B Reset de-asserted 1_B Reset asserted
LCKEN	10	r	**Enable Lockup Status** 0_B Reset by Lockup disabled 1_B Reset by Lockup enabled
0	11, [31:12]	r	**Reserved**

RSTSET

Selective configuration of reset behavior in the system. Write one to set selected bit, writing zeros has no effect.

RSTSET

RCU Reset Set Register **(0404_H)** **Reset Value: 0000 0000$_H$**

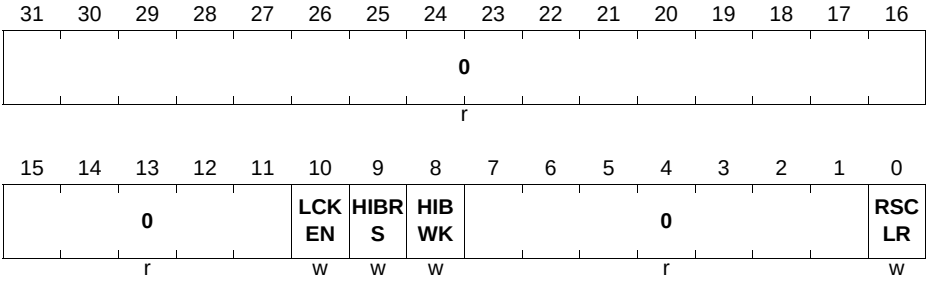


Field	Bits	Type	Description
HIBWK	8	w	**Set Hibernate Wake-up Reset Status** 0_B No effect 1_B Assert reset status bit
HIBRS	9	w	**Set Hibernate Reset** 0_B No effect 1_B Assert reset
LCKEN	10	w	**Enable Lockup Reset** 0_B No effect 1_B Enable reset when Lockup gets asserted
0	[7:0], [31:11]	r	**Reserved**

RSTCLR

Selective configuration of reset behavior in the system. Write one to clear selected bit, writing zeros has no effect.

RSTCLR
RCU Reset Clear Register **(0408_H)** **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
RSCLR	0	w	**Clear Reset Status** 0_B No effect 1_B Clears field **RSTSTAT**.**RSTSTAT**
HIBWK	8	w	**Clear Hibernate Wake-up Reset Status** 0_B No effect 1_B De-assert reset status bit
HIBRS	9	w	**Clear Hibernate Reset** 0_B No effect 1_B De-assert reset
LCKEN	10	w	**Enable Lockup Reset** 0_B No effect 1_B Disable reset when Lockup gets asserted
0	[7:1], [31:11]	r	**Reserved**

PRSTAT0

Selective reset status register for peripherals for Peripherals 0.

Note: Reset release must be effectively prevented for unless module clock is gated or off in cases where kernel clock and bus interface clocks are shared, in order to avoid system hang-ups.

PRSTAT0
RCU Peripheral 0 Reset Status (040C$_H$) **Reset Value: 0081 0A8D$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0								HRPWM0RS	0						ERU1RS
r								r	r						r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				USIC0RS	0	POSIF0RS	0	CCU80RS	0		0	CCU41RS	CCU40RS	0	VADCRS
r				r	r	r	r	r	r		r	r	r	r	r

Field	Bits	Type	Description
VADCRS	0	r	**VADC Reset Status** 0_B Reset de-asserted 1_B Reset asserted
CCU40RS	2	r	**CCU40 Reset Status** 0_B Reset de-asserted 1_B Reset asserted
CCU41RS	3	r	**CCU41 Reset Status** 0_B Reset de-asserted 1_B Reset asserted
CCU80RS	7	r	**CCU80 Reset Status** 0_B Reset de-asserted 1_B Reset asserted
POSIF0RS	9	r	**POSIF0 Reset Status** 0_B Reset de-asserted 1_B Reset asserted
USIC0RS	11	r	**USIC0 Reset Status** 0_B Reset de-asserted 1_B Reset asserted
ERU1RS	16	r	**ERU1 Reset Status** 0_B Reset de-asserted 1_B Reset asserted

Field	Bits	Type	Description
HRPWM0RS	23	r	**HRPWM0 Reset Status** 0_B Reset de-asserted 1_B Reset asserted
0	1,4, [6:5], 8,10, [15:12], [22:17], [31:24]	r	**Reserved**

PRSET0

Selective reset assert register for peripherals for Peripherals 0. Write one to assert selected reset, writing zeros has no effect.

Note: Reset release must be effectively prevented for unless module clock is gated or off in cases where kernel clock and bus interface clocks are shared, in order to avoid system hang-ups.

PRSET0
RCU Peripheral 0 Reset Set **(0410_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0								HRPWM0RS	0						ERU1RS
r								w	r						w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				USIC0RS	0	POSIF0RS	0	CCU80RS	0		0	CCU41RS	CCU40RS	0	VADCRS
r				w	r	w	r	w	r		r	w	w	r	w

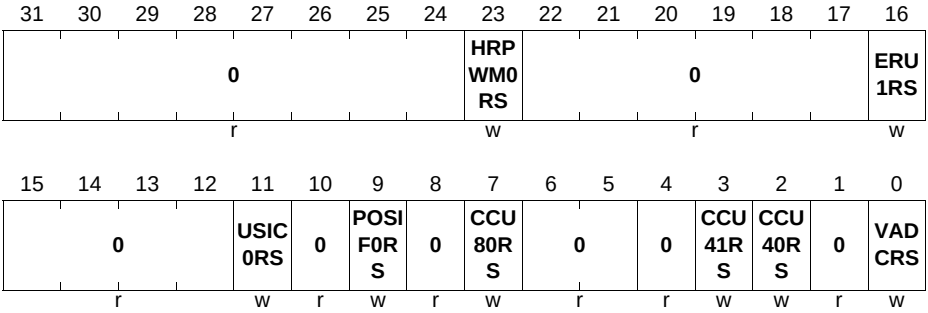
Field	Bits	Type	Description
VADCRS	0	w	**VADC Reset Assert** 0_B No effect 1_B Assert reset
CCU40RS	2	w	**CCU40 Reset Assert** 0_B No effect 1_B Assert reset

Field	Bits	Type	Description
CCU41RS	3	w	**CCU41 Reset Assert** 0_B No effect 1_B Assert reset
CCU80RS	7	w	**CCU80 Reset Assert** 0_B No effect 1_B Assert reset
POSIF0RS	9	w	**POSIF0 Reset Assert** 0_B No effect 1_B Assert reset
USIC0RS	11	w	**USIC0 Reset Assert** 0_B No effect 1_B Assert reset
ERU1RS	16	w	**ERU1 Reset Assert** 0_B No effect 1_B Assert reset
HRPWM0RS	23	w	**HRPWM0 Reset Assert** 0_B No effect 1_B Assert reset
0	1,4, [6:5], 8,10, [15:12], [22:17], [31:24]	r	**Reserved**

PRCLR0

Selective reset de-assert register for peripherals for Peripherals 0. Write one to de-assert selected reset, writing zeros has no effect.

PRCLR0
RCU Peripheral 0 Reset Clear **(0414_H)** **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
VADCRS	0	w	**VADC Reset Clear** 0_B No effect 1_B De-assert reset
CCU40RS	2	w	**CCU40 Reset Clear** 0_B No effect 1_B De-assert reset
CCU41RS	3	w	**CCU41 Reset Clear** 0_B No effect 1_B De-assert reset
CCU80RS	7	w	**CCU80 Reset Clear** 0_B No effect 1_B De-assert reset
POSIF0RS	9	w	**POSIF0 Reset Clear** 0_B No effect 1_B De-assert reset
USIC0RS	11	w	**USIC0 Reset Clear** 0_B No effect 1_B De-assert reset
ERU1RS	16	w	**ERU1 Reset Clear** 0_B No effect 1_B De-assert reset

Field	Bits	Type	Description
HRPWM0RS	23	w	**HRPWM0 Reset Clear** 0_B No effect 1_B De-assert reset
0	1,4, [6:5], 8,10, [15:12], [22:17], [31:24]	r	**Reserved**

PRSTAT1

Selective reset status register for peripherals for Peripherals 1.

Note: Reset release must be effectively prevented for unless module clock is gated or off in cases where kernel clock and bus interface clocks are shared, in order to avoid system hang-ups.

PRSTAT1
RCU Peripheral 1 Reset Status **(0418_H)** **Reset Value: 0000 00B8$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0						PPORTSRS	0	USIC1RS	0	DACRS	MCAN0RS	LEDTSCU0RS	0		0
r						r	r	r	r	r	r	r	r		r

Field	Bits	Type	Description
LEDTSCU0RS	3	r	**LEDTS Reset Status** 0_B Reset de-asserted 1_B Reset asserted
MCAN0RS	4	r	**MultiCAN Reset Status** 0_B Reset de-asserted 1_B Reset asserted

Field	Bits	Type	Description
DACRS	5	r	**DAC Reset Status** 0_B Reset de-asserted 1_B Reset asserted
USIC1RS	7	r	**USIC1 Reset Status** 0_B Reset de-asserted 1_B Reset asserted
PPORTSRS	9	r	**PORTS Reset Status** 0_B Reset de-asserted 1_B Reset asserted
0	0, [2:1], 6,8, [31:10]	r	**Reserved**

PRSET1

Selective reset assert register for peripherals for Peripherals 1. Write one to assert selected reset, writing zeros has no effect.

PRSET1
RCU Peripheral 1 Reset Set **($041C_H$)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0						PPORTSRS	0	USIC1RS	0	DACRS	MCAN0RS	LEDTSCU0RS	0		0
r						w	r	w	r	w	w	w	r		r

Field	Bits	Type	Description
LEDTSCU0RS	3	w	**LEDTS Reset Assert** 0_B No effect 1_B Assert reset

Field	Bits	Type	Description
MCAN0RS	4	w	**MultiCAN Reset Assert** 0_B No effect 1_B Assert reset
DACRS	5	w	**DAC Reset Assert** 0_B No effect 1_B Assert reset
USIC1RS	7	w	**USIC1 Reset Assert** 0_B No effect 1_B Assert reset
PPORTSRS	9	w	**PORTS Reset Assert** 0_B No effect 1_B Assert reset
0	0, [2:1], 6,8, [31:10]	r	**Reserved**

PRCLR1

Selective reset de-assert register for peripherals for Peripherals 1. Write one to de-assert selected reset, writing zeros has no effect.

PRCLR1
RCU Peripheral 1 Reset Clear **(0420_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0						PPORTSRS	0	USIC1RS	0	DACRS	MCAN0RS	LEDTSCU0RS	0		0
r						w	r	w	r	w	w	w	r		r

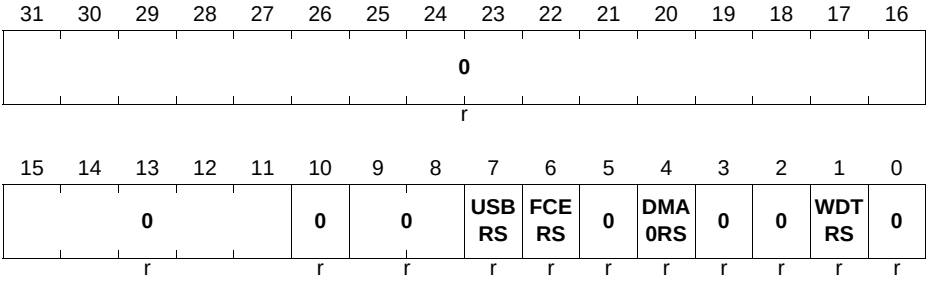
Field	Bits	Type	Description
LEDTSCU0RS	3	w	**LEDTS Reset Clear** 0_B No effect 1_B De-assert reset
MCAN0RS	4	w	**MultiCAN Reset Clear** 0_B No effect 1_B De-assert reset
DACRS	5	w	**DAC Reset Clear** 0_B No effect 1_B De-assert reset
USIC1RS	7	w	**USIC1 Reset Clear** 0_B No effect 1_B De-assert reset
PPORTSRS	9	w	**PORTS Reset Clear** 0_B No effect 1_B De-assert reset
0	0, [2:1], 6,8, [31:10]	r	**Reserved**

PRSTAT2

Selective reset status register for peripherals for Peripherals 2.

Note: Reset release must be effectively prevented for unless module clock is gated or off in cases where kernel clock and bus interface clocks are shared, in order to avoid system hang-ups.

PRSTAT2
RCU Peripheral 2 Reset Status **(0424_H)** **Reset Value: 0000 00D2$_H$**



Field	Bits	Type	Description
WDTRS	1	r	**WDT Reset Status** 0_B Reset de-asserted 1_B Reset asserted
DMA0RS	4	r	**DMA0 Reset Status** 0_B Reset de-asserted 1_B Reset asserted
FCERS	6	r	**FCE Reset Status** 0_B Reset de-asserted 1_B Reset asserted
USBRS	7	r	**USB Reset Status** 0_B Reset de-asserted 1_B Reset asserted
0	0, 2, 3, 5, [9:8], 10, [31:11]	r	**Reserved**

PRSET2

Selective reset assert register for peripherals for Peripherals 2. Write one to assert selected reset, writing zeros has no effect.

PRSET2

RCU Peripheral 2 Reset Set **(0428_H)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

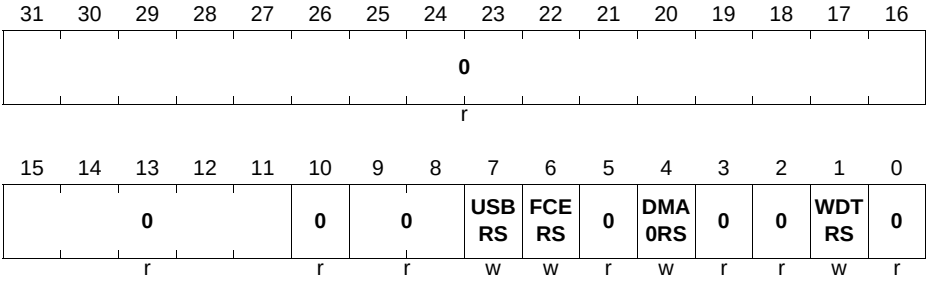
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0					0	0		USB RS	FCE RS	0	DMA 0RS	0	0	WDT RS	0
r					r	r		w	w	r	w	r	r	w	r

Field	Bits	Type	Description
WDTRS	1	w	**WDT Reset Assert** 0_B No effect 1_B Assert reset
DMA0RS	4	w	**DMA0 Reset Assert** 0_B No effect 1_B Assert reset
FCERS	6	w	**FCE Reset Assert** 0_B No effect 1_B Assert reset
USBRS	7	w	**USB Reset Assert** 0_B No effect 1_B Assert reset
0	0, 2, 3, 5, [9:8], 10, [31:11]	r	**Reserved**

PRCLR2

Selective reset de-assert register for peripherals for Peripherals 2. Write one to de-assert selected reset, writing zeros has no effect.

PRCLR2
RCU Peripheral 2 Reset Clear **($042C_H$)** **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
WDTRS	1	w	**WDT Reset Clear** 0_B No effect 1_B De-assert reset
DMA0RS	4	w	**DMA0 Reset Clear** 0_B No effect 1_B De-assert reset
FCERS	6	w	**FCE Reset Clear** 0_B No effect 1_B De-assert reset
USBRS	7	w	**USB Reset Clear** 0_B No effect 1_B De-assert reset
0	0, 2, 3, 5, [9:8], 10, [31:11]	r	**Reserved**

11.10.5 CCU Registers

CLKSTAT

Global clock status register.

CLKSTAT

Clock Status Register **(0600_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

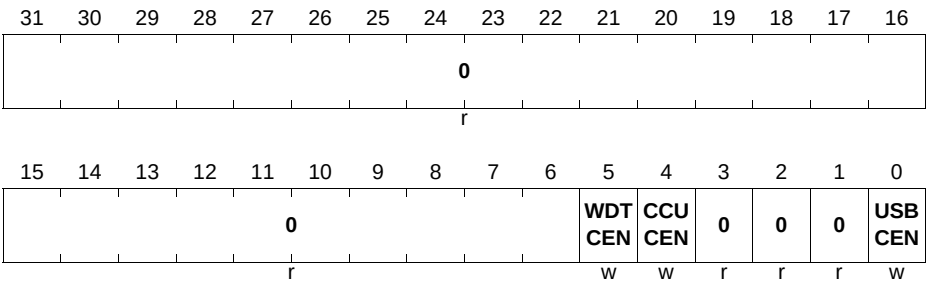
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0										**WDT CST**	**CCU CST**	**0**	**0**	**0**	**USB CST**
r										r	r	r	r	r	r

Field	Bits	Type	Description
USBCST	0	r	**USB Clock Status** 0_B Clock disabled 1_B Clock enabled
CCUCST	4	r	**CCU Clock Status** 0_B Clock disabled 1_B Clock enabled
WDTCST	5	r	**WDT Clock Status** 0_B Clock disabled *Note: WDT clock can be put on hold in debug mode when this behavior is enabled at the watchdog* 1_B Clock enabled
0	1, 2, 3, [31:6]	r	**Reserved** Read as 0.

CLKSET

Global clock enable register. Write one to enable selected clock, writing zeros has no effect.

CLKSET
CLK Set Register **(0604_H)** **Reset Value: 0000 0000$_H$**

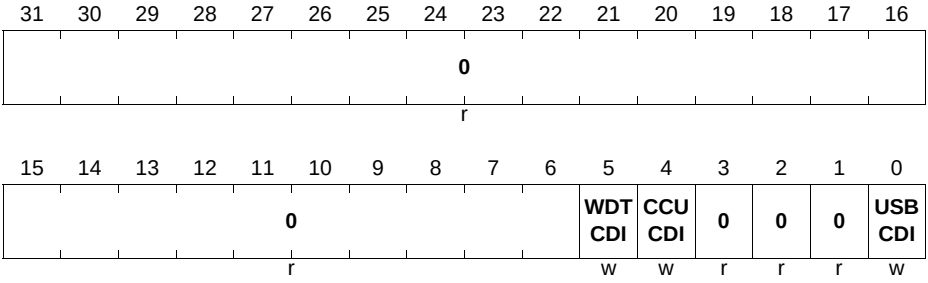


Field	Bits	Type	Description
USBCEN	0	w	**USB Clock Enable** 0_B No effect 1_B Enable
CCUCEN	4	w	**CCU Clock Enable** 0_B No effect 1_B Enable
WDTCEN	5	w	**WDT Clock Enable** 0_B No effect 1_B Enable
0	1,2, 3, [31:6]	r	**Reserved** Read as 0.

CLKCLR

Global clock disable register. Write one to disable selected clock, writing zeros has no effect.

CLKCLR
CLK Clear Register **(0608_H)** **Reset Value: 0000 0000$_H$**

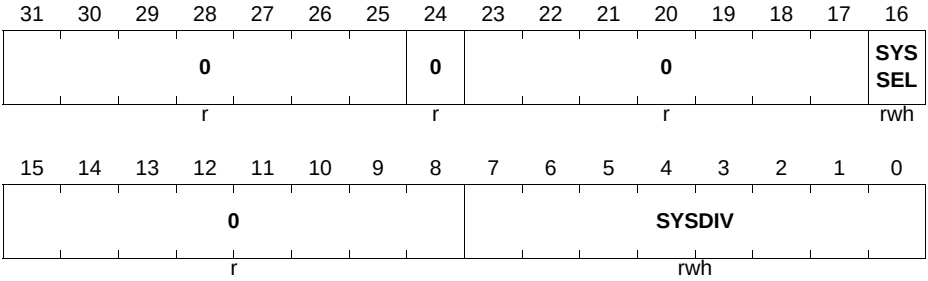


Field	Bits	Type	Description
USBCDI	0	w	**USB Clock Disable** 0_B No effect 1_B Disable clock
CCUCDI	4	w	**CCU Clock Disable** 0_B No effect 1_B Disable clock
WDTCDI	5	w	**WDT Clock Disable** 0_B No effect 1_B Disable clock
0	1,2, 3, [31:6]	r	**Reserved** Read as 0.

SYSCLKCR

System clock control register.

SYSCLKCR
System Clock Control Register (060C$_H$) Reset Value: 0000 0000$_H$



Field	Bits	Type	Function
SYSDIV	[7:0]	rwh	**System Clock Division Value** The value the divider operates is (SYSDIV+1).
SYSSEL	16	rwh	**System Clock Selection Value** 0_B f_{OFI} clock 1_B f_{PLL} clock
ECAT0SEL		rw	**System Clock Selection Value** 0_B f_{USBPLL} clock 1_B f_{SYS} clock
0	[15:8], 24, [23:17], [31:25]	r	**Reserved** Read as 0; should be written with 0.

CPUCLKCR

CPU clock control register.

CPUCLKCR
CPU Clock Control Register (0610$_H$) Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0															CPUDIV
r															rwh

Field	Bits	Type	Function
CPUDIV	0	rwh	**CPU Clock Divider Enable** This bit enables division of f_{SYS} clock to produce f_{CPU} clock. 0_B $f_{CPU} = f_{SYS}$ 1_B $f_{CPU} = f_{SYS} / 2$ *Note: Some clock division settings are not allowed. See **Table 11-5** for more details.*
0	[31:1]	r	**Reserved** Read as 0; should be written with 0.

PBCLKCR

Peripheral clock control register.

PBCLKCR
Peripheral Bus Clock Control Register (0614$_H$) Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

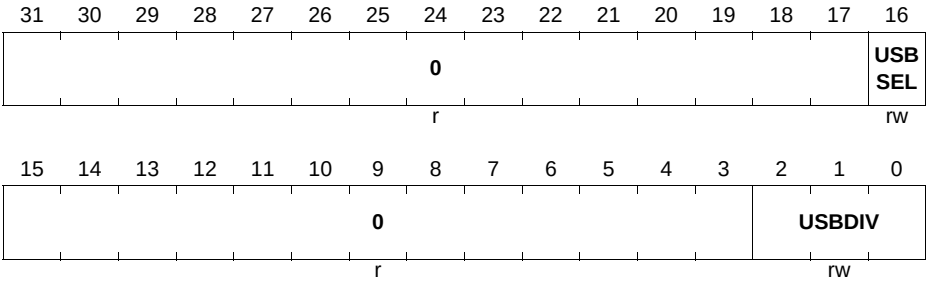
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0															PBDIV
r															rwh

Field	Bits	Type	Function
PBDIV	0	rwh	**PB Clock Divider Enable** This bit enables division of f_{SYS} clock to produce f_{PERIPH} clock. 0_B $f_{PERIPH} = f_{CPU}$ 1_B $f_{PERIPH} = f_{CPU} / 2$ *Note: Some clock division settings are not allowed. See **Table 11-5** for more details.*
0	[31:1]	r	**Reserved** Read as 0; should be written with 0.

USBCLKCR

USB clock control register.

USBCLKCR
USB Clock Control Register (0618$_H$) Reset Value: 0000 0000$_H$

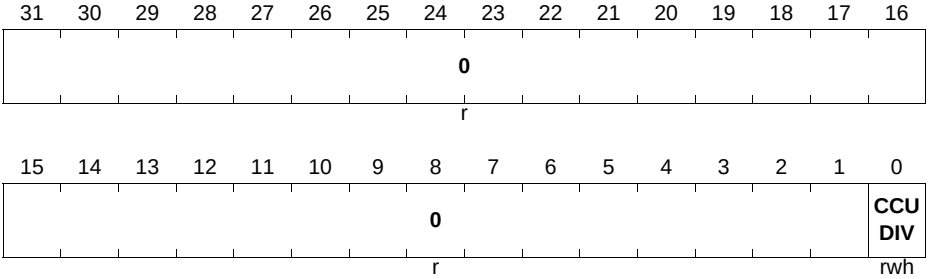


Field	Bits	Type	Function
USBDIV	[2:0]	rw	**USB Clock Divider Value** PLL clock is divided by USBDIV + 1 Must only be programmed, when clock is not used
USBSEL	16	rw	**USB Clock Selection Value** 0_B USB PLL Clock 1_B PLL Clock
0	[15:3], [31:17]	r	**Reserved** Read as 0; should be written with 0.

CCUCLKCR

CCUx clock control register.

CCUCLKCR
CCU Clock Control Register (0620$_H$) Reset Value: 0000 0000$_H$

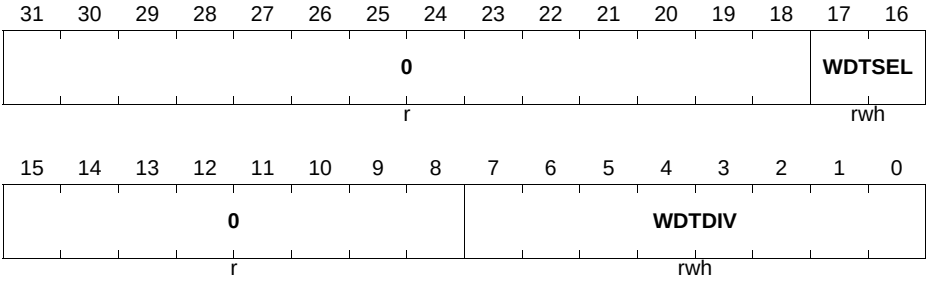


Field	Bits	Type	Function
CCUDIV	0	rwh	**CCU Clock Divider Enable** This bit enables division of f_{SYS} clock to produce f_{CCU} clock. 0_B $f_{CCU} = f_{SYS}$ 1_B $f_{CCU} = f_{SYS} / 2$ *Note: Some clock division settings are not allowed. See **Table 11-5** for more details.*
0	[31:1]	r	**Reserved** Read as 0; should be written with 0.

WDTCLKCR

System watchdog (WDT) clock control register.

WDTCLKCR
WDT Clock Control Register (0624$_H$) Reset Value: 0000 0000$_H$

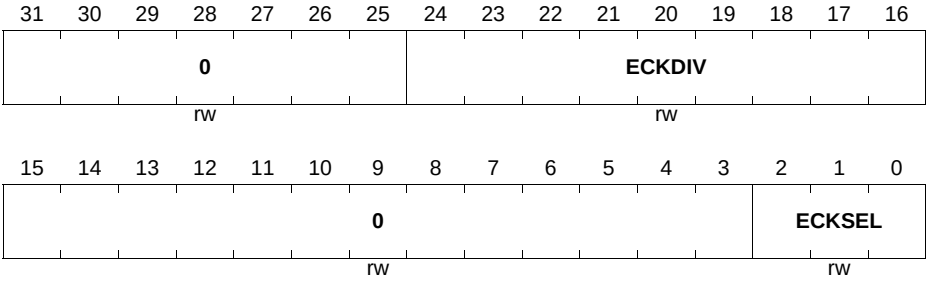


Field	Bits	Type	Function
WDTDIV	[7:0]	rwh	**WDT Clock Divider Value** WDT is divided by WDTDIV + 1 Must only be programmed, when clock is not used
WDTSEL	[17:16]	rwh	**WDT Clock Selection Value** 00_B f_{OFI} clock 01_B f_{STDBY} clock 10_B f_{PLL} clock 11_B Reserved
0	[15:8], [31:18]	r	**Reserved** Read as 0; should be written with 0.

EXTCLKCR

External clock control register. Use this register to select output clock.

EXTCLKCR
External Clock Control **(0628_H)** **Reset Value: $0000\ 0000_H$**

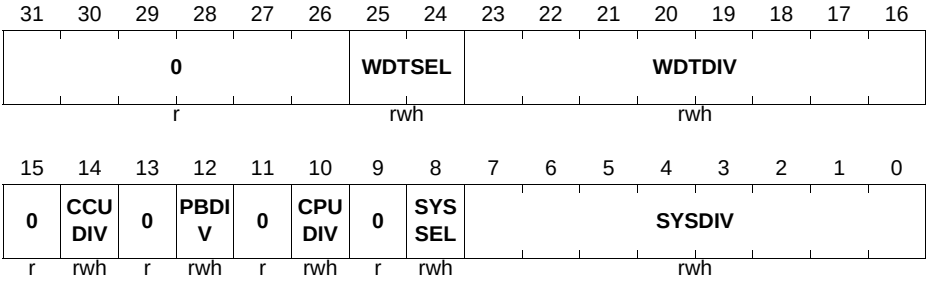


Field	Bits	Type	Description
ECKSEL	[2:0]	rw	**External Clock Selection Value** 000_B f_{SYS} clock 001_B Reserved 010_B f_{USB} clock divided according to ECKDIV bit field configuration 011_B f_{PLL} clock divided according to ECKDIV bit field configuration 0100_B f_{STDBY} clock
ECKDIV	[24:16]	rw	**External Clock Divider Value** PLL clock is divided by ECKDIV + 1 Must only be programmed, when clock is not used
0	[15:3], [31:25]	rw	**Reserved** Read as 0.

MLINKCLKCR

Multi-link register for central clock control. This register allows consistent reconfiguration of multiple clocks with a single register access and prevents invalid configurations i.e. clock division combinations are not allowed (see **Table 11-5** for more details). An attempt to write invalid value will be rejected and result in generation of bus error response.

MLINKCLKCR
Multi-Link Clock Control **($062C_H$)** **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
SYSDIV	[7:0]	rwh	**System Clock Division Value** The value the divider operates is (SYSDIV+1).
SYSSEL	8	rwh	**System Clock Selection Value** 0_B f_{OFI} clock 1_B f_{PLL} clock
CPUDIV	10	rwh	**CPU Clock Divider Enable** This bit enables division of f_{SYS} clock to produce f_{CPU} clock. 0_B $f_{CPU} = f_{SYS}$ 1_B $f_{CPU} = f_{SYS} / 2$ *Note: Some clock division combinations are not allowed. See **Table 11-5** for more details. This register must ignore the write and generate error response if invalid configuration attempt detected.*
PBDIV	12	rwh	**PB Clock Divider Enable** This bit enables division of f_{SYS} clock to produce f_{PERIPH} clock. 0_B $f_{PERIPH} = f_{CPU}$ 1_B $f_{PERIPH} = f_{CPU} / 2$ *Note: Some clock division combinations are not allowed. See **Table 11-5** for more details. This register must ignore the write and generate error response if invalid configuration attempt detected.*

Field	Bits	Type	Description
CCUDIV	14	rwh	**CCU Clock Divider Enable** This bit enables division of f_{SYS} clock to produce f_{CCU} clock. 0_B $f_{CCU} = f_{SYS}$ 1_B $f_{CCU} = f_{SYS} / 2$ *Note: Some clock division combinations are not allowed. See **Table 11-5** for more details. This register must ignore the write and generate error response if invalid configuration attempt detected.*
WDTDIV	[23:16]	rwh	**WDT Clock Divider Value** WDT is divided by WDTDIV + 1 Must only be programmed, when clock is not used
WDTSEL	[25:24]	rwh	**WDT Clock Selection Value** 00_B f_{OFI} clock 01_B f_{STDBY} clock 10_B f_{PLL} clock 11_B Reserved
0	9, 11,13, 15, [31:26]	r	**Reserved** Read as 0.

SLEEPCR

Configuration register that defines some system behavior aspects while in sleep mode. The original system state gets restored upon wake-up from sleep mode.

SLEEPCR
Sleep Control Register **(0630_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0							0	0		WDTCR	CCUCR	0	0	0	USBCR
r							r	r		rw	rw	r	r	r	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0														0	SYSSEL
r														r	rwh

Field	Bits	Type	Description
SYSSEL	0	rwh	**System Clock Selection Value** 0_B f_{OFI} clock 1_B f_{PLL} clock
USBCR	16	rw	**USB Clock Control in Sleep Mode** 0_B Disabled 1_B Enabled
CCUCR	20	rw	**CCU Clock Control in Sleep Mode** 0_B Disabled 1_B Enabled
WDTCR	21	rw	**WDT Clock Control in Sleep Mode** 0_B Disabled 1_B Enabled
0	1, 17, 18, 19, [15:2], [23:22], 24, [31:25]	r	**Reserved** Read as 0.

DSLEEPCR

Configuration register that defines some system behavior aspects while in Deep Sleep mode. The original system state gets restored upon wake-up from sleep mode except for PLL re-start if enabled before entering Deep Sleep mode and configured to go into power down while in Deep Sleep mode.

DSLEEPCR
Deep Sleep Control Register **(0634_H)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0							0	0		WDTCR	CCUCR	0	0	0	USBCR
r							r	r		rw	rw	r	r	r	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		VCOPDN	PLLPDN	FPDN	0									0	SYSSEL
r		rw	rw	rw	r									r	rwh

Field	Bits	Type	Description
SYSSEL	0	rwh	**System Clock Selection Value** 0_B f_{OFI} clock 1_B f_{PLL} clock
FPDN	11	rw	**Flash Power Down** 1_B Flash power down module 0_B No effect
PLLPDN	12	rw	**PLL Power Down** 1_B Switch off main PLL 0_B No effect
VCOPDN	13	rw	**VCO Power Down** 1_B Switch off VCO of main PLL 0_B No effect
USBCR	16	rw	**USB Clock Control in Deep Sleep Mode** 0_B Disabled 1_B Enabled
CCUCR	20	rw	**CCU Clock Control in Deep Sleep Mode** 0_B Disabled 1_B Enabled
WDTCR	21	rw	**WDT Clock Control in Deep Sleep Mode** 0_B Disabled 1_B Enabled

Field	Bits	Type	Description
0	1, [10:2], [15:14], 17,18, 19, [23:22], 24, [31:25]	r	**Reserved** Read as 0.

CGATSTAT0

Clock gating status for peripherals 0.

CGATSTAT0
Peripheral 0 Clock Gating Status **(0640_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0								HRPWM0	0						ERU1
r								r	r						r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				USIC0	0	POSIF0	0	CCU80	0		0	CCU41	CCU40	0	VADC
r				r	r	r	r	r	r		r	r	r	r	r

Field	Bits	Type	Description
VADC	0	r	**VADC Gating Status** 0_B Gating de-asserted 1_B Gating asserted
CCU40	2	r	**CCU40 Gating Status** 0_B Gating de-asserted 1_B Gating asserted
CCU41	3	r	**CCU41 Gating Status** 0_B Gating de-asserted 1_B Gating asserted
CCU80	7	r	**CCU80 Gating Status** 0_B Gating de-asserted 1_B Gating asserted

Field	Bits	Type	Description
POSIF0	9	r	**POSIF0 Gating Status** 0_B Gating de-asserted 1_B Gating asserted
USIC0	11	r	**USIC0 Gating Status** 0_B Gating de-asserted 1_B Gating asserted
ERU1	16	r	**ERU1 Gating Status** 0_B Gating de-asserted 1_B Gating asserted
HRPWM0	23	r	**HRPWM0 Gating Status** 0_B Gating de-asserted 1_B Gating asserted
0	1,4, [6:5], 8, 10, [15:12], [22:17], [31:24]	r	**Reserved**

CGATSET0

Clock gating enable for peripherals 0. Write one to selected bit to Enable gating of corresponding clock, writing zeros has no effect.

Note: Clock gating shall not be activated unless the module is in reset state.

CGATSET0
Peripheral 0 Clock Gating Set **(0644_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0								HRPWM0	0						ERU1
r								w	r						w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				USIC0	0	POSIF0	0	CCU80	0		0	CCU41	CCU40	0	VADC
r				w	r	w	r	w	r		r	w	w	r	w

Field	Bits	Type	Description
VADC	0	w	**VADC Gating Set** 0_B No effect 1_B Enable gating
CCU40	2	w	**CCU40 Gating Set** 0_B No effect 1_B Enable gating
CCU41	3	w	**CCU41 Gating Set** 0_B No effect 1_B Enable gating
CCU80	7	w	**CCU80 Gating Set** 0_B No effect 1_B Enable gating
POSIF0	9	w	**POSIF0 Gating Set** 0_B No effect 1_B Enable gating
USIC0	11	w	**USIC0 Gating Set** 0_B No effect 1_B Enable gating
ERU1	16	w	**ERU1 Gating Set** 0_B No effect 1_B Enable gating
HRPWM0	23	w	**HRPWM0 Gating Set** 0_B No effect 1_B Enable gating
0	1,4, [6:5], 8, 10, [15:12], [22:17], [31:24]	r	**Reserved**

CGATCLR0

Clock gating disable for peripherals 0. Write one to selected bit to disable gating of corresponding clock, writing zeros has no effect.

CGATCLR0

Peripheral 0 Clock Gating Clear **(0648_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0								HRPWM0	0						ERU1
r								w	r						w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				USIC0	0	POSIF0	0	CCU80	0		0	CCU41	CCU40	0	VADC
r				w	r	w	r	w	r		r	w	w	r	w

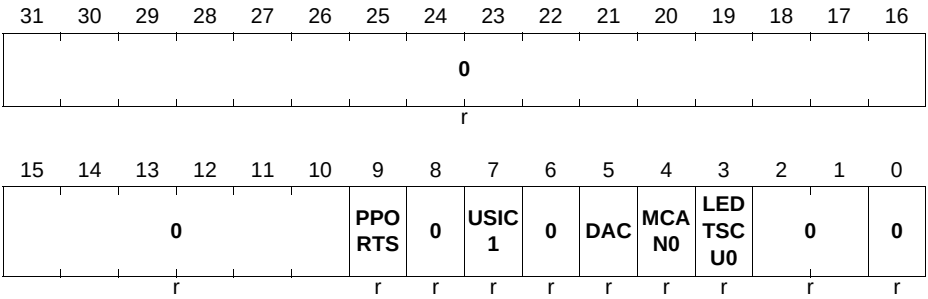
Field	Bits	Type	Description
VADC	0	w	**VADC Gating Clear** 0_B No effect 1_B Disable gating
CCU40	2	w	**CCU40 Gating Clear** 0_B No effect 1_B Disable gating
CCU41	3	w	**CCU41 Gating Clear** 0_B No effect 1_B Disable gating
CCU80	7	w	**CCU80 Gating Clear** 0_B No effect 1_B Disable gating
POSIF0	9	w	**POSIF0 Gating Clear** 0_B No effect 1_B Disable gating
USIC0	11	w	**USIC0 Gating Clear** 0_B No effect 1_B Disable gating
ERU1	16	w	**ERU1 Gating Clear** 0_B No effect 1_B Disable gating
HRPWM0	23	w	**HRPWM0 Gating Clear** 0_B No effect 1_B Disable gating

Field	Bits	Type	Description
0	1,4, [6:5], 8, 10, [15:12], [22:17], [31:24]	r	**Reserved**

CGATSTAT1

Clock gating status for peripherals 1.

CGATSTAT1
Peripheral 1 Clock Gating Status **($064C_H$)** **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
LEDTSCU0	3	r	**LEDTS Gating Status** 0_B Gating de-asserted 1_B Gating asserted
MCAN0	4	r	**MultiCAN Gating Status** 0_B Gating de-asserted 1_B Gating asserted
DAC	5	r	**DAC Gating Status** 0_B Gating de-asserted 1_B Gating asserted
USIC1	7	r	**USIC1 Gating Status** 0_B Gating de-asserted 1_B Gating asserted

Field	Bits	Type	Description
PPORTS	9	r	**PORTS Gating Status** 0_B Gating de-asserted 1_B Gating asserted
0	0, [2:1], 6,8, [31:10]	r	**Reserved**

CGATSET1

Clock gating enable for peripherals 1. Write one to selected bit to Enable gating of corresponding clock, writing zeros has no effect.

Note: Clock gating shall not be activated unless the module is in reset state.

CGATSET1
Peripheral 1 Clock Gating Set **(0650_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0						PPORTS	0	USIC1	0	DAC	MCAN0	LEDTSCU0	0		0
r						w	r	w	r	w	w	w	r		r

Field	Bits	Type	Description
LEDTSCU0	3	w	**LEDTS Gating Set** 0_B No effect 1_B Enable gating
MCAN0	4	w	**MultiCAN Gating Set** 0_B No effect 1_B Enable gating
DAC	5	w	**DAC Gating Set** 0_B No effect 1_B Enable gating

Field	Bits	Type	Description
USIC1	7	w	**USIC1 Gating Set** 0_B No effect 1_B Enable gating
PPORTS	9	w	**PORTS Gating Set** 0_B No effect 1_B Enable gating
0	0, [2:1], 6,8, [31:10]	r	**Reserved**

CGATCLR1

Clock gating disable for peripherals 1. Write one to selected bit to disable gating of corresponding clock, writing zeros has no effect.

CGATCLR1
Peripheral 1 Clock Gating Clear **(0654_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0						PPORTS	0	USIC1	0	DAC	MCAN0	LEDTSCU0	0		0
r						w	r	w	r	w	w	w	r		r

Field	Bits	Type	Description
LEDTSCU0	3	w	**LEDTS Gating Clear** 0_B No effect 1_B Disable gating
MCAN0	4	w	**MultiCAN Gating Clear** 0_B No effect 1_B Disable gating

Field	Bits	Type	Description
DAC	5	w	**DAC Gating Clear** 0_B No effect 1_B Disable gating
USIC1	7	w	**USIC1 Gating Clear** 0_B No effect 1_B Disable gating
PPORTS	9	w	**PORTS Gating Clear** 0_B No effect 1_B Disable gating
0	0, [2:1], 6,8, [31:10]	r	**Reserved**

CGATSTAT2

Clock gating status for peripherals 2.

CGATSTAT2

Peripheral 2 Clock Gating Status (0658_H) Reset Value: $0000\ 0000_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0					0	0		USB	FCE	0	DMA0	0	0	WDT	0
r					r	r		r	r	r	r	r	r	r	r

Field	Bits	Type	Description
WDT	1	r	**WDT Gating Status** 0_B Gating de-asserted 1_B Gating asserted
DMA0	4	r	**DMA0 Gating Status** 0_B Gating de-asserted 1_B Gating asserted

Field	Bits	Type	Description
FCE	6	r	**FCE Gating Status** 0_B Gating de-asserted 1_B Gating asserted
USB	7	r	**USB Gating Status** 0_B Gating de-asserted 1_B Gating asserted
0	0, 2, 3, 5, [9:8], 10, [31:11]	r	**Reserved**

CGATSET2

Clock gating enable for peripherals 2. Write one to selected bit to Enable gating of corresponding clock, writing zeros has no effect.

Note: Clock gating shall not be activated unless the module is in reset state.

CGATSET2
Peripheral 2 Clock Gating Set **($065C_H$)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0					**0**	**0**		**USB**	**FCE**	**0**	**DMA 0**	**0**	**0**	**WDT**	**0**
r					r	r		w	w	r	w	r	r	w	r

Field	Bits	Type	Description
WDT	1	w	**WDT Gating Set** 0_B No effect 1_B Enable gating

Field	Bits	Type	Description
DMA0	4	w	**DMA0 Gating Set** 0_B No effect 1_B Enable gating
FCE	6	w	**FCE Gating Set** 0_B No effect 1_B Enable gating
USB	7	w	**USB Gating Set** 0_B No effect 1_B Enable gating
0	0, 2, 3, 5, [9:8], 10, [31:11]	r	**Reserved**

CGATCLR2

Clock gating disable for peripherals 2. Write one to selected bit to disable gating of corresponding clock, writing zeros has no effect.

CGATCLR2
Peripheral 2 Clock Gating Clear (0660_H) Reset Value: $0000\ 0000_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0					0	0		USB	FCE	0	DMA 0	0	0	WDT	0
r					r	r		w	w	r	w	r	r	w	r

Field	Bits	Type	Description
WDT	1	w	**WDT Gating Clear** 0_B No effect 1_B Disable gating

Field	Bits	Type	Description
DMA0	4	w	**DMA0 Gating Clear** 0_B No effect 1_B Disable gating
FCE	6	w	**FCE Gating Clear** 0_B No effect 1_B Disable gating
USB	7	w	**USB Gating Clear** 0_B No effect 1_B Disable gating
0	0, 2, 3, 5, [9:8], 10, [31:11]	r	**Reserved**

OSCHPSTAT

Status register of the OSC_HP oscillator.

OSCHPSTAT
OSC_HP Status Register **(0700_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

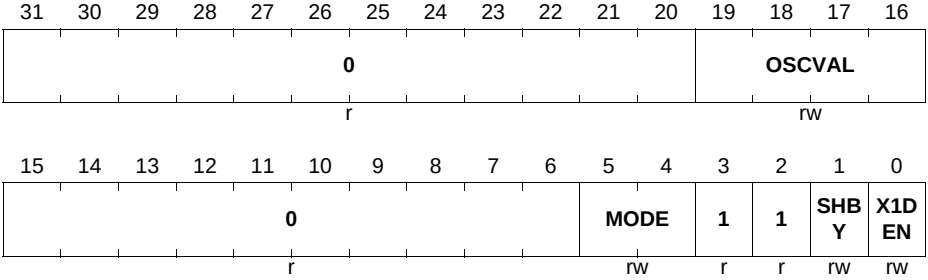
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0															**X1D**
r															rh

Field	Bits	Type	Description
X1D	0	rh	**XTAL1 Data Value** This bit monitors the value (level) of pin XTAL1. If XTAL1 is not used as clock input it can be used as GPI pin. This bit is only updated if X1DEN is set.
0	[31:1]	r	**Reserved**

OSCHPCTRL

Control register of the OSC_HP oscillator.

OSCHPCTRL
OSC_HP Control Register **(0704_H)** **Reset Value: 0000 003C$_H$**



Field	Bits	Type	Description
X1DEN	0	rw	**XTAL1 Data Enable** 0_B Bit X1D is not updated 1_B Bit X1D can be updated
SHBY	1	rw	**Shaper Bypass** 0_B The shaper is not bypassed 1_B The shaper is bypassed

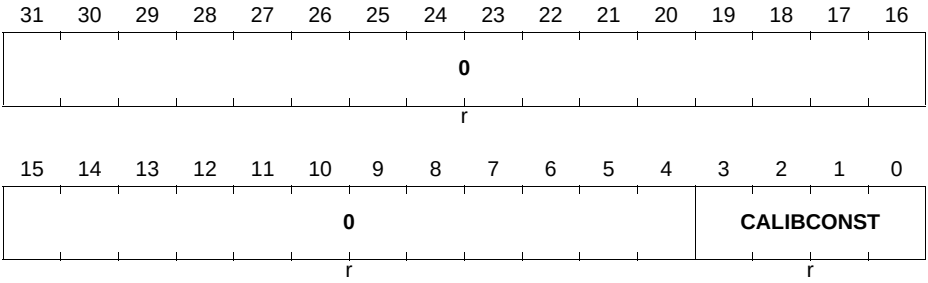
Field	Bits	Type	Description
MODE	[5:4]	rw	**Oscillator Mode** 00_B External Crystal Mode and External Input Clock Mode. The oscillator Power-Saving Mode is not entered. 01_B OSC is disabled. The oscillator Power-Saving Mode is not entered. 10_B External Input Clock Mode and the oscillator Power-Saving Mode is entered 11_B OSC is disabled. The oscillator Power-Saving Mode is entered.
OSCVAL	[19:16]	rw	**OSC Frequency Value** This bit field defines the divider value that generates the reference clock that is supervised by the oscillator watchdog. f_{OSC} is divided by OSCVAL + 1 in order to generate f_{OSCREF}.
0	[15:6], [31:20]	r	**Reserved** Read as 0; should be written with 0.
1	2,3	r	**Reserved** Read as 1; should be written with 1

CLKCALCONST

Clock calibration constant for PLL programming.

CLKCALCONST

Clock Calibration Constant Register ($070C_H$) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
CALIBCONST	[3:0]	r	**Clock Calibration Constant Value** This field contains clock calibration constant value for PLL configuration.
0	[31:4]	r	**Reserved** Read as 0; should be written with 0.

PLLSTAT

System PLL Status register.

PLLSTAT
PLL Status Register **(0710_H)** **Reset Value: $0000\ 0002_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0						**PLLSP**	**PLLHV**	**PLLLV**	**BY**	**K2RDY**	**K1RDY**	**0**	**VCOLOCK**	**PWDSTAT**	**VCOBYST**
r						rh	rh	rh	rh	rh	rh	r	rh	rh	rh

Field	Bits	Type	Description
VCOBYST	0	rh	**VCO Bypass Status** 0_B Free-running / Normal Mode is entered 1_B Prescaler Mode is entered
PWDSTAT	1	rh	**PLL Power-saving Mode Status** 0_B PLL Power-saving Mode was not entered 1_B PLL Power-saving Mode was entered
VCOLOCK	2	rh	**PLL LOCK Status** 0_B PLL not locked 1_B PLL locked

Field	Bits	Type	Description
K1RDY	4	rh	**K1 Divider Ready Status** This bit indicates if the K1-divider operates on the configured value or not. This is of interest if the value is changed. 0_B K1-Divider does not operate with the new value 1_B K1-Divider operate with the new value
K2RDY	5	rh	**K2 Divider Ready Status** This bit indicates if the K2-divider operates on the configured value or not. This is of interest if the value is changed. 0_B K2-Divider does not operate with the new value 1_B K2-Divider operate with the new value
BY	6	rh	**Bypass Mode Status** 0_B Bypass Mode is not entered 1_B Bypass Mode is entered. Input f_{OSC} is selected as output f_{PLL}.
PLLLV	7	rh	**Oscillator for PLL Valid Low Status Bit** This bit indicates if the frequency output of OSC is usable for the VCO part of the PLL. This is checked by the Oscillator Watchdog of the PLL. 0_B The OSC frequency is not usable. Frequency f_{REF} is too low. 1_B The OSC frequency is usable
PLLHV	8	rh	**Oscillator for PLL Valid High Status Bit** This bit indicates if the frequency output of OSC is usable for the VCO part of the PLL. This is checked by the Oscillator Watchdog of the PLL. 0_B The OSC frequency is not usable. Frequency f_{OSC} is too high. 1_B The OSC frequency is usable

Field	Bits	Type	Description
PLLSP	9	rh	**Oscillator for PLL Valid Spike Status Bit** This bit indicates if the frequency output of OSC is usable for the VCO part of the PLL. This is checked by the Oscillator Watchdog of the PLL. 0_B The OSC frequency is not usable. Spikes are detected that disturb a locked operation 1_B The OSC frequency is usable
0	3, [31:10]	r	**Reserved** Read as 0.

PLLCON0

System PLL configuration register 0.

PLLCON0
PLL Configuration 0 Register (0714$_H$) Reset Value: 0003 0003$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0											FOTR	AOTREN	RESLD	OSCRES	PLLPWD
r											rw	rw	w	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0									OSCDISCDIS	0	FINDIS	0	VCOTR	VCOPWD	VCOBYP
r									rw	r	rwh	r	rw	rw	rw

Field	Bits	Type	Description
VCOBYP	0	rw	**VCO Bypass** 0_B Normal operation, VCO is not bypassed 1_B Prescaler Mode, VCO is bypassed
VCOPWD	1	rw	**VCO Power Saving Mode** 0_B Normal behavior 1_B The VCO is put into a Power Saving Mode and can no longer be used. Only the Bypass and Prescaler Mode are active if previously selected.

Field	Bits	Type	Description
VCOTR	2	rw	**VCO Trim Control** 0_B VCO bandwidth is operation in the normal range. VCO output frequency is between 260 and 520 MHz for a input frequency between 8 and 16 MHz. 1_B VCO bandwidth is operation in the test range. VCO output frequency is between 260 and 520 MHz for a input frequency between 8 and 16 MHz. Selecting a VCO trim value of one can result in a high jitter but the PLL is still operable.
FINDIS	4	rwh	**Disconnect Oscillator from VCO** 0_B connect oscillator to the VCO part 1_B disconnect oscillator from the VCO part.
OSCDISCDIS	6	rw	**Oscillator Disconnect Disable** This bit is used to disable the control FINDIS in a PLL loss-of-lock case. 0_B In case of a PLL loss-of-lock bit FINDIS is set 1_B In case of a PLL loss-of-lock bit FINDIS is cleared
PLLPWD	16	rw	**PLL Power Saving Mode** 0_B Normal behavior 1_B The complete PLL block is put into a Power Saving Mode and can no longer be used. Only the Bypass Mode is active if previously selected.
OSCRES	17	rw	**Oscillator Watchdog Reset** This bit controls signal *osc_fail_res_i* at the PLL module. 0_B The Oscillator Watchdog of the PLL is not reset and remains active 1_B The Oscillator Watchdog of the PLL is reset
RESLD	18	w	**Restart VCO Lock Detection** Setting this bit will clear bit PLLSTAT.VCOLOCK and restart the VCO lock detection. Reading this bit returns always a zero.

Field	Bits	Type	Description
AOTREN	19	rw	**Automatic Oscillator Calibration Enable** Setting this bit will enable automatic adjustment of the f_{OFI} clock with f_{STDBY} clock used as reference clock. 0_B Disable 1_B Enable
FOTR	20	rw	**Factory Oscillator Calibration** Force adjustment of the internal oscillator with the firmware defined value. 0_B No effect 1_B Force fixed-value trimming
0	3, 5, [15:7], [31:21]	r	**Reserved** Read as 0; should be written with 0.

PLLCON1

System PLL configuration register 1.

PLLCON1
PLL Configuration 1 Register **(0718_H)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0				PDIV				0	K2DIV						
r				rw				r	rw						

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	NDIV							0	K1DIV						
r	rw							r	rw						

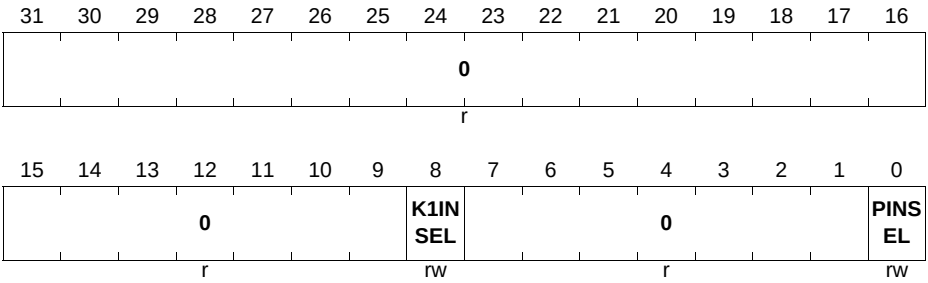
Field	Bits	Type	Description
K1DIV	[6:0]	rw	**K1-Divider Value** The value the K1-Divider operates is K1DIV+1.
NDIV	[14:8]	rw	**N-Divider Value** The value the N-Divider operates is NDIV+1.
K2DIV	[22:16]	rw	**K2-Divider Value** The value the K2-Divider operates is K2DIV+1.

Field	Bits	Type	Description
PDIV	[27:24]	rw	**P-Divider Value** The value the P-Divider operates is PDIV+1.
0	7,15, 23, [31:28]	r	**Reserved** Read as 0; should be written with 0.

PLLCON2

System PLL configuration register 2.

PLLCON2
PLL Configuration 2 Register (071C$_H$) Reset Value: 0000 0001$_H$

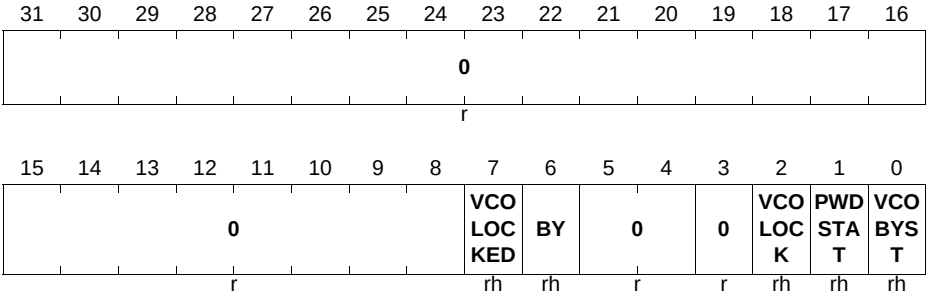


Field	Bits	Type	Description
PINSEL	0	rw	**P-Divider Input Selection** 0_B PLL external oscillator selected 1B Backup clock f_{ofi} selected
K1INSEL	8	rw	**K1-Divider Input Selection** 0_B PLL external oscillator selected 1_B Backup clock f_{ofi} selected
0	[7:1], [31:9]	r	**Reserved** Read as 0; should be written with 0.

USBPLLSTAT

USB PLL Status register.

USBPLLSTAT
USB PLL Status Register **(0720_H)** **Reset Value: $0000\ 0002_H$**



Field	Bits	Type	Description
VCOBYST	0	rh	**VCO Bypass Status** 0_B Normal Mode is entered 1_B Prescaler Mode is entered
PWDSTAT	1	rh	**PLL Power-saving Mode Status** 0_B PLL Power-saving Mode was not entered 1_B PLL Power-saving Mode was entered
VCOLOCK	2	rh	**PLL VCO Lock Status** 0_B The frequency difference of f_{REF} and f_{DIV} is greater than allowed. The VCO part of the PLL can not lock on a target frequency. 1_B The frequency difference of f_{REF} and f_{DIV} is small enough to enable a stable VCO operation *Note: In case of a loss of VCO lock the f_{VCO} goes to the upper boundary of the VCO frequency if the reference clock input is greater than expected.* *Note: In case of a loss of VCO lock the f_{VCO} goes to the lower boundary of the VCO frequency if the reference clock input is lower than expected.*
BY	6	rh	**Bypass Mode Status** 0_B Bypass Mode is not entered 1_B Bypass Mode is entered. Input f_{OSC} is selected as output f_{PLL}.

Field	Bits	Type	Description
VCOLOCKED	7	rh	**PLL LOCK Status** 0_B PLL not locked 1_B PLL locked
0	3, [5:4], [31:8]	r	**Reserved** Read as 0.

USBPLLCON

USB PLL configuration register 0.

USBPLLCON
USB PLL Configuration Register (0724_H) **Reset Value: $0001\ 0003_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0				**PDIV**				**0**					**RES LD**	**0**	**PLL PWD**
r				rw				r					w	r	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**NDIV**							**0**	**OSC DISC DIS**	**0**	**FIND IS**	**0**	**VCO TR**	**VCO PWD**	**VCO BYP**
r	rw							r	rw	r	rwh	r	rw	rw	rw

Field	Bits	Type	Description
VCOBYP	0	rw	**VCO Bypass** 0_B Normal operation, VCO is not bypassed 1_B Prescaler Mode, VCO is bypassed
VCOPWD	1	rw	**VCO Power Saving Mode** 0_B Normal behavior 1_B The VCO is put into a Power Saving Mode

Field	Bits	Type	Description
VCOTR	2	rw	**VCO Trim Control** 0_B VCO bandwidth is operating in the normal range. VCO output frequency is between 260 and 520 MHz for a input frequency between 8 and 16 MHz. 1_B VCO bandwidth is operating in the test range. VCO output frequency is between 260 and 520 MHz for a input frequency between 8 and 16 MHz. Selecting a VCO trim value of one can result in a high jitter but the PLL is still operable.
FINDIS	4	rwh	**Disconnect Oscillator from VCO** 0_B Connect oscillator to the VCO part 1_B Disconnect oscillator from the VCO part.
OSCDISCDIS	6	rw	**Oscillator Disconnect Disable** This bit is used to disable the control FINDIS in a PLL loss-of-lock case. 0_B In case of a PLL loss-of-lock bit FINDIS is set 1_B In case of a PLL loss-of-lock bit FINDIS is cleared
NDIV	[14:8]	rw	**N-Divider Value** The value the N-Divider operates is NDIV+1.
PLLPWD	16	rw	**PLL Power Saving Mode** 0_B Normal behavior 1_B The complete PLL block is put into a Power Saving Mode. Only the Bypass Mode is active if previously selected.
RESLD	18	w	**Restart VCO Lock Detection** Setting this bit will clear bit PLLSTAT.VCOLOCK and restart the VCO lock detection. Reading this bit returns always a zero.
PDIV	[27:24]	rw	**P-Divider Value** The value the P-Divider operates is PDIV+1.

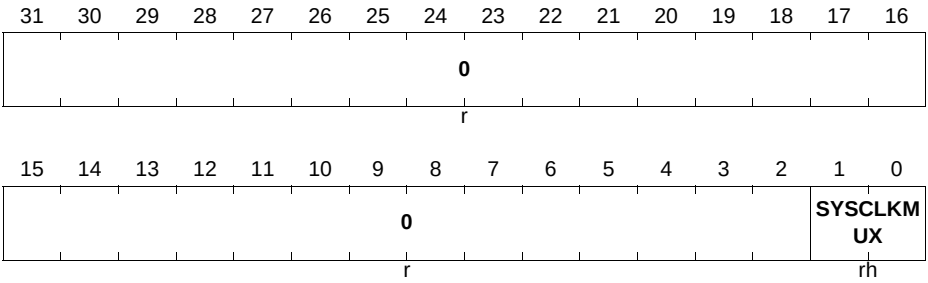
Field	Bits	Type	Description
0	3, 5, 7, 15, 17, [23:19], [31:28]	r	**Reserved** Should be written with 0.

CLKMXSTAT

Clock Multiplexer Switching Status

This register shows status of clock multiplexing upon switching from one clock source to another. This register should be checked before disabling any of the multiplexer input clock sources after switching. Bits of this registers indicate which of the corresponding input clocks must not be switched off under any circumstances until indicated as inactive. The clocks sources that are indicated as active are still contributing in driving the output clock from respective multiplexer. This is a side effect of glitch-free clock switching mechanism.

CLKMXSTAT
Clock Multiplexing Status Register (0738$_H$) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
SYSCLKMUX	[1:0]	rh	**Status of System Clock Multiplexing Upon Source Switching** Clock sources that are indicated active are still contributing in glitch-free switching $x1_B$ f_{OFI} clock active $1x_B$ f_{PLL} clock active

Field	Bits	Type	Description
0	[31:2]	r	**Reserved**

Communication Peripherals

12 LED and Touch-Sense (LEDTS)

The LED and Touch-Sense (LEDTS) drives LEDs and controls touch pads used as human-machine interface (HMI) in an application.

Table 12-1 Abbreviations in chapter

Abbreviation	Meaning
LEDTS	LED and Touch-sense
TSD	time slice duration
TFD	time frame duration
TPD	time period duration

12.1 Overview

The LEDTS can measure the capacitance of up to 8 touch pads using the relaxation oscillator (RO) topology. The pad capacitance is measured by generating oscillations on the pad for a fixed time period and counting them. The module can also drive up to 64 LEDs in an LED matrix. Touch pads and LEDs can share pins to minimize the number of pins needed for such applications. This configuration is realized by the module controlling the touch pads and driving the LEDs in a time-division multiplexed manner.

The LEDs in the LED matrix are organized into columns and lines. Every line can be shared between up to 8 LEDs and one touch pad. Certain functions such as column enabling, function selection and control are controlled by hardware. Application software is required to update the LED lines and evaluate the touch pad measurement results.

12.1.1 Features

This device contains 1 LEDTS kernel. Each kernel has an LED driving function and a touch-sensing function.

For the LED driving function, an LEDTS kernel provides these features:

- Selection of up to 8 LED columns; Up to 7 LED columns if touch-sense function is also enabled
- Configurable active time in LED columns to control LED brightness
- Possibility to drive up to 8 LEDs per column, common-anode or common-cathode
- Shadow activation of line pattern for LED column time slice; LED line patterns are updated synchronously to column activation
- Configurable interrupt enable on selected event
- Line and column pins controlled by port SFR setting

For the touch-sensing function, an LEDTS kernel provides these features:

- Up to 8 touch input lines

- Only one pad can be measured at any time; selection of active pad controllable by software or hardware round-robin
- Flexible measurement time on touch pads
- Pin oscillation control circuit with adjustments for oscillation
- 16-bit counter: For counting oscillations on pin.
- Configurable interrupt enable on selected event
- Pin over-rule control for active touch input line (pin)

*Note: This chapter refers to the LED or touch-sense pins, e.g. 'pin COL[x]', 'pin TSIN[x]'. In all instances, it refers to the user-configured pin(s) which selects the LED/touch-sense function. Refer to **Section 12.9.5** for more elaboration.*

*Note: This chapter describes the full capability of the LEDTS. The amount of features available on the device is limited by the device pins assigned. Please refer to the **Section 12.11** for the pins assigned to the LEDTS.*

Table 12-2 LEDTS Applications

Use Case	Application
Non-mechanical switch	HMI
LED feedback	HMI
Simple PWM	PWM

12.1.2 Block Diagram

The LEDTS block diagram is shown in **Figure 12-1**.

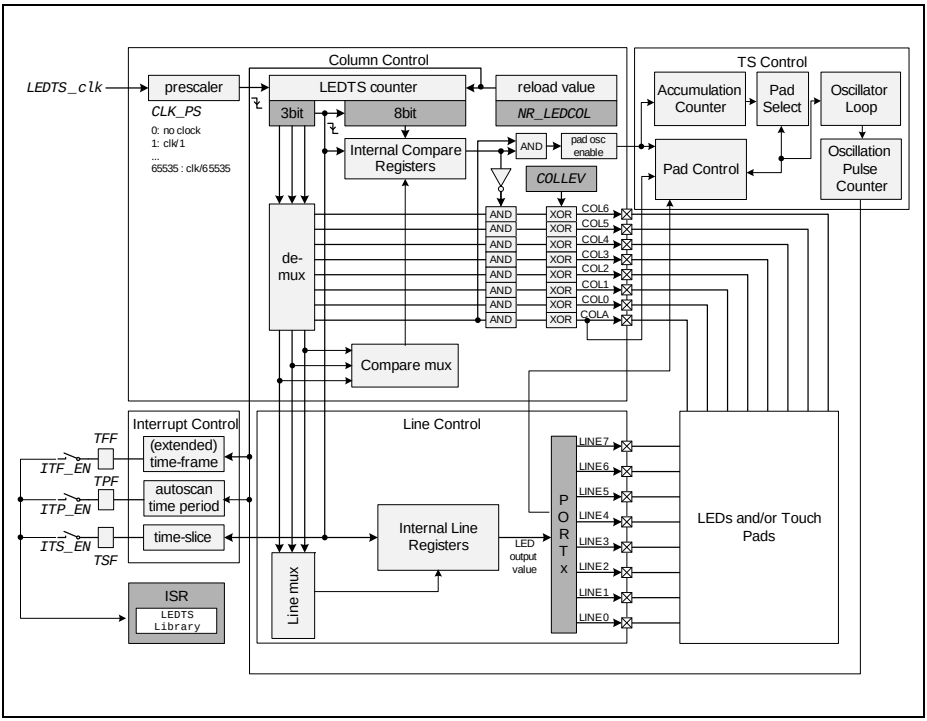


Figure 12-1 LEDTS Block Diagram

12.2 Functional Overview

The same pin can support LED & touch-sense functions in a time-multiplexed manner. LED mode or touch-sense mode can be enabled by hardware for respective function controls.

Time-division multiplexing is done by dividing the time domain into time slots. This basic time slot is called a **time slice**. In one time slice, one LED column is activated or the capacitance of one touch pad is measured.

A **time frame** is composed of 1 or more time slices, up to a maximum of eight. There is one time slice for every LED column enabled. If only LED function is enabled, a time frame can compose up to 8 LED time slices. However, if touch sense function is enabled, the last time slice in every time frame is reserved for touch-sense function. This reduces the maximum number of time slices that can be used for LED function in each time frame to 7. Only one time slice is used for touch sense function in every time frame. This is regardless of the number of touch pads enabled.

In each time slice used for LED function, only one LED column is enabled at a time. In the time slice reserved for touch-sense function, oscillations are enabled and measured on the pin which is activated. No LED column is active during this time slice. A touch pad input line (TSIN[x] pin) is active when its pad turn is enabled. If more than one touch pad input lines are enabled, the enabling and measurement on the touch pads in performed in a round robin manner. Only one touch pad is measured in every time frame.

The resolution of oscillation measurement can be increased by accumulating oscillation counts on each touch input line. When enabled by configuration of “Accumulate Count” (ACCCNT), the pad turn can be extended on consecutive time frames by up to 16 times.This also means that the same touch pad will be measured in consecutive time frames. This control will be handled by hardware. Otherwise it is also possible to enable for software control where the active pad turn is fully under user control.

The number of consecutive time frames, for which a pad turn has been extended, forms an **extended time frame**. When touch-sense function is enabled for automatic hardware pad turn control, several (extended) time frames make up one **autoscan time period** where all pad turns are completed. The time slice duration is configured centrally for the LED and/or touch-sense functions, using the LEDTS-counter. Refer to the description in **Section 12.3**, **Section 12.9.3** and **Figure 12-4**.

If enabled, a time slice interrupt is triggered on overflow of the 8LSBs of the LEDTS-counter for each new time slice started. The (extended) time frame interrupt may also be enabled. It is triggered on (the configured counts of) overflow of the whole LEDTS-counter. The autoscan time period interrupt may also be enabled. However, this interrupt will require that the hardware pad turn control is enabled. It is triggered when hardware completes the last pad turn on the highest enabled touch input line TSIN[NR_TSIN].

The column activation and pin oscillation duty cycles can be configured for each time slice. This allows the duration of activation of LED columns and/or touch-sense

oscillation counting to be flexible. This is also how the relative brightness of the LEDs can be controlled. In case of touch pads, the activation time is called the oscillation window.

Figure 12-2 shows an example for a LED matrix configuration with touch pads. The configuration in this example is 8 X 4 LED matrix with 4 touch input lines enabled in sequence by hardware. Here no pad turn is extended by ACCCNT, so four time frames complete an autoscan time period.

In the time slice interrupt, software can:

- set up line pattern for next time slice
- set up compare value for next time slice
- evaluate current function in time slice (especially for analysis/debugging)

Refer to **Section 12.9.1** for **Interpretation of Bit Field FNCOL** to determine the currently active time slice.

The (extended) time frame interrupt indicates one touch input line TSIN[x] has been sensed (once or number of times in consecutive frames), application-level software can, for example:

- start touch-sense processing (e.g. filtering) routines and update status
- update LED display data to SFR

In the autoscan time period interrupt which indicates all touch-sense input TSIN[x] have been scanned one round, application-level software can:

- evaluate touch detection result & action
- update LED display data to SFR

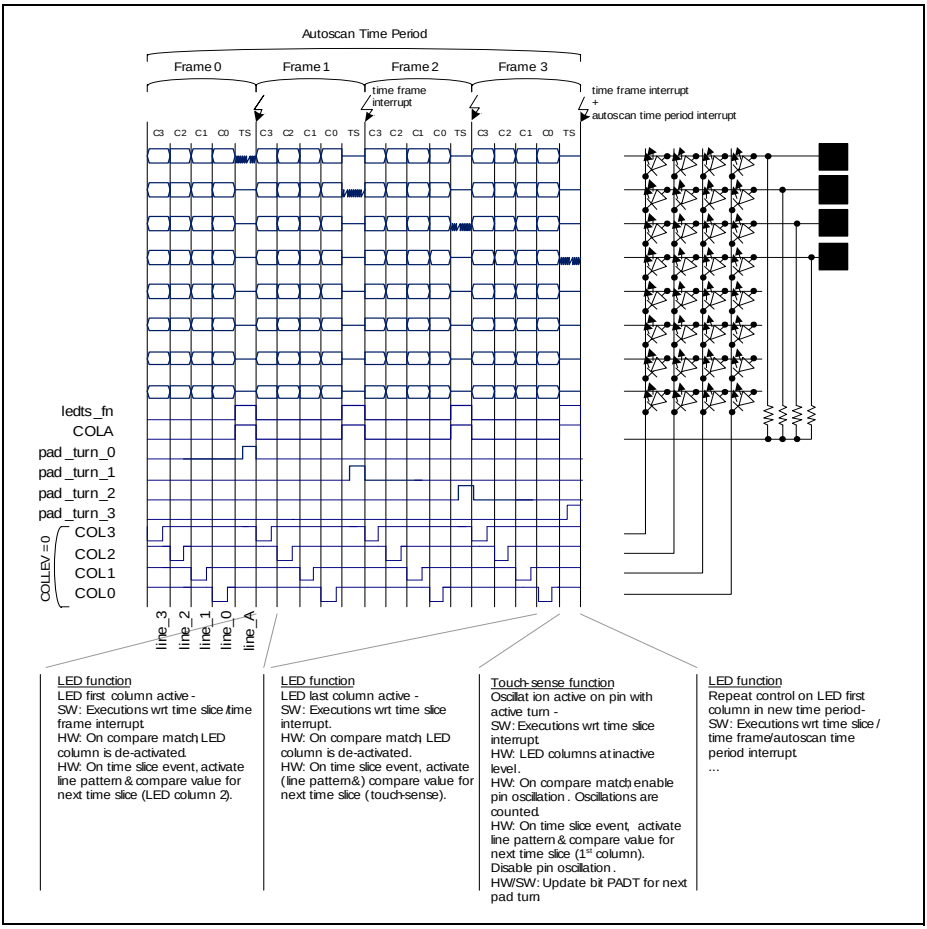


Figure 12-2 Time-Multiplexed LEDTS Functions on Pin (Example)

12.3 LED Drive Mode

LED driving is supported mainly for LED column selection and line control. At one time, only one column is active. The corresponding line level at high or low determines if the associated LED on column is lit or not. Up to eight columns are supported (if only LED function is enabled), and up to eight LEDs can be controlled per column.

With direct LED drive, adjustment of luminence for different types of LEDs with different forward voltages is supported. A compare register for LEDTS-counter is provided so that the duty cycle for column enabling per time slice can be adjusted. The LED column is enabled from the beginning of the time slice until compare match event. For 100% duty cycle for LED column enable in time slice, the compare value should be set to FF_H. If the compare value is set to 00_H, the LED column will stay at passive level during the time slice. The internal compare register for each time slice is updated by shadow transfer from their corresponding compare SFR. This takes place automatically at the beginning of each time slice, refer to **Figure 12-3**.

Updating of LED line pattern (LED enabling) per column (time slice) is performed via a similar shadow transfer mechanism, as illustrated in **Figure 12-3**. This shadow transfer of the corresponding line pattern to the internal line SFR takes place automatically at the beginning of each new time slice.

Note: Any write to any compare or line SFR within the time slice does not affect the internal latched configuration of current time slice.

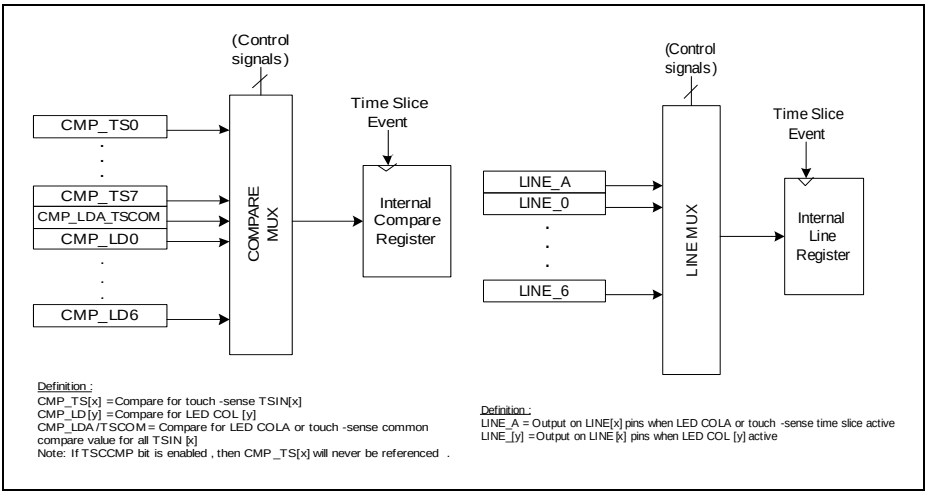


Figure 12-3 Activate Internal Compare/Line Register for New Time Slice

When the LEDTS-counter is first started (enable input clock by CLK_PS), a shadow transfer of line pattern and compare value is activated for the first time slice (column).

A time slice interrupt can be enabled. A new time slice starts on the overflow of the 8LSBs of the LEDTS-counter.

Figure 12-4 shows the LED function control circuit. This circuit also provides the control for enabling the pad oscillator. A 16-bit divider provides pre-scale possibilities to flexibly configure the internal LEDTS-counter count rate, which overflows in one time frame. During a time frame comprising a configurable number of time slices, the configured number of LED columns are activated in sequence. In the last time slice of the time frame, touch-sense function is activated if enabled.

The LEDTS-counter is started when bit CLK_PS is set to any value other than 0 and either the LED or touch-sense function is enabled. It does not run when both functions are disabled. To avoid over-write of function enable which disturbs the hardware control during LEDTS-counter running, the TS_EN and LD_EN bits can only be modified when bit CLK_PS = 0. It is nonetheless possible to set the bits TS_EN and LD_EN in one single write to SFR **GLOBCTL** when setting CLK_PS from 0 to 1, or from 1 to 0.

When started, the counter starts running from a reset/reload value based on enabled function(s): 1) the number of columns (bit-field NR_LEDCOL) when LED function is enabled, 2) add one time slice at end of time frame when touch-sense function is enabled. The counter always counts up and overflows from 7FFH to the reload value which is the same as the reset value. Within each time frame, the sequence of LED column enabling always starts from the most-significant enabled column (column with highest numbering).

To illustrate this point, in the case of four LED columns and one touch pad input enabled, the column enabling sequence will be in as follows:

- Start with COL3,
- followed by COL2,
- followed by COL1,
- followed by COL0,
- then COLA for touch sense function.

If the touch-sense function is not enabled, COLA will be available for LED function as the last LED column time slice of a time frame. The column enabling sequence will then be as follows:

- Start with COL2,
- followed by COL1,
- followed by COL0,
- then COLA.

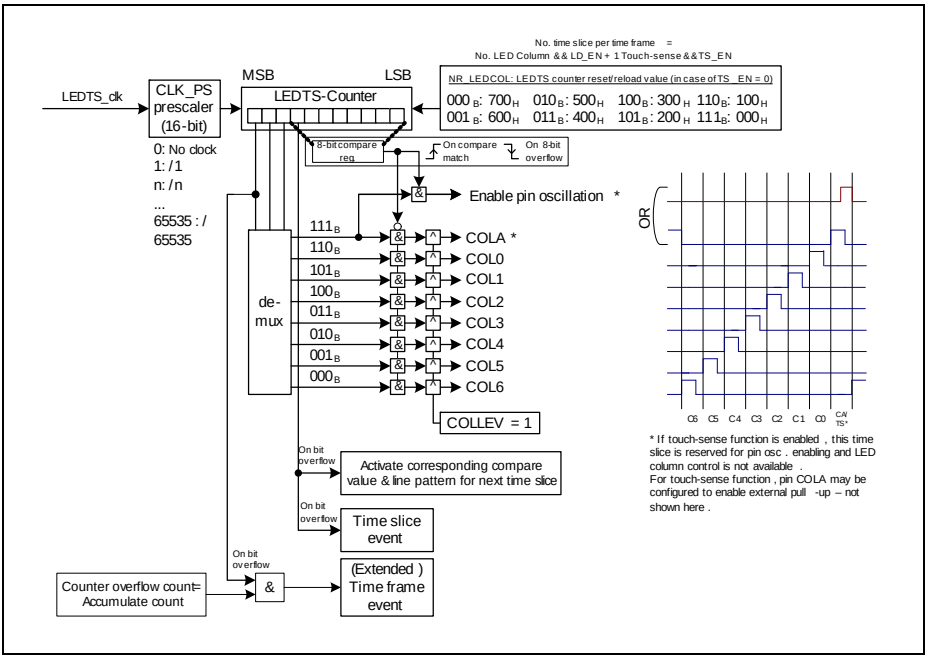


Figure 12-4 LED Function Control Circuit (also provides pad oscillator enable)

In **Section 12.9.3**, the time slice duration and formulations for LEDTS related timings are provided.

12.3.1 LED Pin Assignment and Current Capability

One LED column pin is enabled within each configured time slice duration to control up to eight LEDs at a time. The assignment of COL[x] to pins is configurable to provide options for application pin usage. The current capability of device pins is also a consideration factor for deciding pin assignment to LED function.

The product data-sheet provides data for all I/O parameters relevant to LED drive.

12.4 Touch-Sense Mode

Figure 12-5 shows the pin oscillation control unit, which is integrated with the standard PORTS pad. The active pad turn (pad_turn_x) for a touch input line is defined as the time duration within the touch-sense time slice (COL A) where the TS-counter is counting oscillations on the TSIN[x] pin. In the case of hardware pad turn control (default), the same TS-counter is connected sequentially on enabled touch input lines to execute a round-robin touch-sensing of TSIN[x] pins.

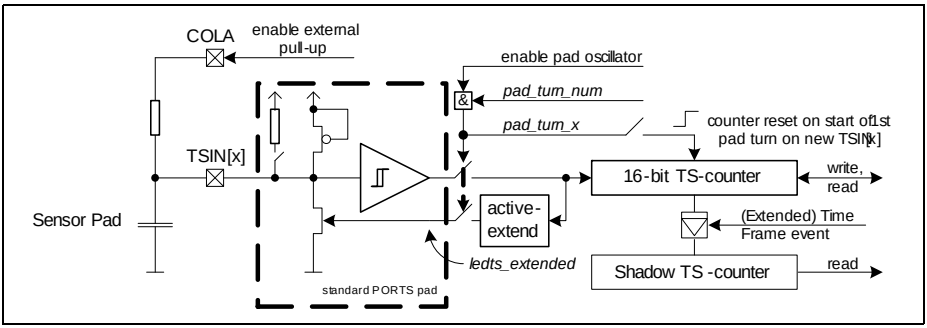


Figure 12-5 Touch-Sense Oscillator Control Circuit

In an example of four touch input lines enabled, the sequence order of touch-sense time slice (COL A) in sequential frames is as follows:

- Always starts with TSIN0,
- followed by TSIN1 in touch-sense time slice of next frame,
- followed by TSIN2 in next touch-sense time slice,
- then TSIN3, and repeat cycle.

It is possible to enable the touch-sense time slice on the same touch input line for consecutive frames (up to 16 times), to accumulate the oscillation count in TS-counter (see **Figure 12-6**). To illustrate this point, in the same case of four touch input lines enabled, and 2 accumulation counts configured, is as follows:

- Always starts with TSIN0,
- followed by TSIN0 again in touch-sense time slice of next frame,
- followed by TSIN1 in touch-sense time slice of next frame,
- followed by TSIN1 again in touch-sense time slice of next frame,
- followed by TSIN2 in touch-sense time slice of next frame,
- followed by TSIN2 again in touch-sense time slice of next frame,
- followed by TSIN3 in touch-sense time slice of next frame,
- then TSIN3 again, and repeat cycle.

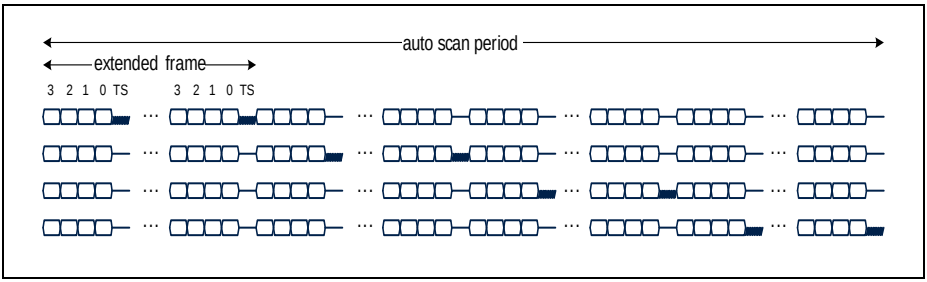


Figure 12-6 Hardware-Controlled Pad Turns for Autoscan of Four TSIN[x] with Extended Frames

There is a 16-bit TS-counter register and there is a 16-bit shadow TS-counter register. The former is both write- and read-accesible, while the latter is only read-accessible. The actual TS-counter counts the latched number of oscillations and can only be written when there is no active pad turn. The content of the TS-counter is latched to the shadow register on every (extended) time frame event. Reading from the shadow register therefore shows the latest valid oscillation count on one TSIN[x] input, ensuring for the application SW there is at least one time slice duration to get the valid oscillation count and meanwhile the actual TS-counter could continually update due to enabled pin oscillations in current time slice.

The TS-counter and shadow TS-counter have another user-enabled function on (extended) time frame event, which is to validate the counter value differences. When this function is enabled by the user and in case the counter values do not differ by 2^n LSB bits ('n' is configurable), the (extended) time frame interrupt request is gated (no interrupt) and the time frame event flag TFF is not set. This gating is on top of the time frame interrupt enable/disable control.

The TS-counter may be enabled for automatic reset (to 00_H) on the start of a new pad turn on the next TSIN[x], i.e. resets in the first touch-sense time slice of each (extended) time frame. Bit TSCTROVF indicates that the counter has overflowed. Alternatively, it can be configured such that the TS-counter stops counting in touch-sense time slice(s) of the same extended frame when the count value saturates, i.e. does not overflow & stops at $FFFF_H$. In this case, the TS-counter starts running again only in a new (extended) frame on the start of a new pad turn on the next TSIN[x].

A configurable pin-low-level active extension is provided for adjustment of oscillation per user system. The extension is active during the discharge phase of oscillation, and can be configured to be extended by a number of peripheral clocks. This function is very useful if there is a series resistor between the pin and the touch pad which makes the discharge slower. **Figure 12-7** illustrates this function.

The configuration of the active touch-sense pin TSIN[x] is over-ruled by hardware in the active duration to enable oscillations, reference **Section 12.9.5**. In particular, the weak

internal pull-up enable over-rule can be optionally de-activated (correspondingly internal pull-down disable over-rule is also de-activated; PORTS pin SFR setting for pull applies instead), such as when the user system utilize external resistor for pull-up instead. In the whole duration of the touch-sense time slice, COLA is activated high. This activates a pull-up via an external resistor connected to pin COLA. This configuration provides some flexibility to adjust the pin oscillation rate for adaptation to user system.

The touch-sense function is time-multiplexed with the LED function on enabled LINE[x]/TSIN[x] pins. During the touch-sense time slice for the other TSIN pins which are not on active pad turn, the corresponding LINE[x] output remains active. It is recommended that software takes care to set the line bits to 1 to avoid current sink from pin COLA.

The touch-sense function is active in the last time slice of a time frame. Refer to **Section 12.2**, and **Section 12.3** for more details on time slice allocation and configuration.

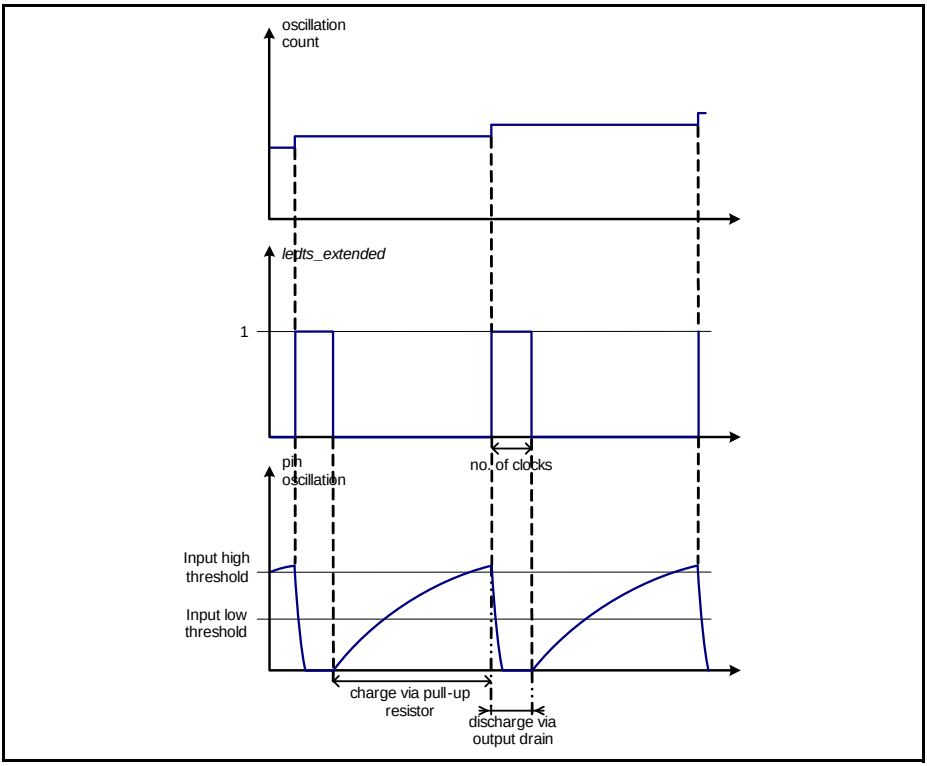


Figure 12-7 Pin-Low-Level Extension Function

The oscillation is enabled on the pin with valid turn for a configurable duration. A compare value provides the means to adjust the duty cycle within the time slice. The pin oscillation is enabled (TS-counter is counting) only on compare match until the end of the time slice. The time interval, in which the TS-counter is counting, is called the oscillation window. For a 100% duty cycle, the compare value has to be set to 00_H. In this case, the oscillation window fills in the entire time slice. Setting the compare value to FF_H results in no pin oscillation in time slice.

The time slice interrupt, (extended) time frame interrupt and/or autoscan time period interrupt may be enabled as required for touch-sense control.

12.4.1 Finger Sensing

When a finger is placed on the sensor pad, it increases the pin capacitance and frequency of oscillation on pin is reduced. The pin oscillation frequency without finger touch is typically expected to be in the range of 100 kHz to 5 MHz. Various factors affect the oscillation frequency including the size of touch pad, ground planes around and below the pad, the material and thickness of the overlay cover, the trace length and the individual pin itself (every pin has a different pull-up resistance).

In a real-world application, the printed circuit board (PCB) will not be touched directly. Instead, there is usually some sort of a transparent cover material, like a piece of plexiglass sheet, glued onto the PCB. In most of these applications, the oscillation frequency will change by about 2-10% when touched. This change in oscillation frequency can be considered to be very small, and therefore, further signal processing is necessary for reliable detection. Typically, this processing takes the form of a moving average calculation. It is never recommended to try to detect touches based on the raw oscillation count value.

As described in above section, some flexibility is provided to adjust the oscillation frequency in the user system: 1) Configurable pin low-level active extension, 2) Alternative enabling of external pull-up with resistance selectable by user. With a configurable time slice duration, the software can configure the duration of the active pad turn (adjustable within time slice using compare function) and set a count threshold for oscillations to detect if there is a finger touch or not.

To increase touch-sensing oscillation count accuracy, the input clock to LEDTS kernel should be set as high as possible.

12.5 Operating both LED Drive and Touch-Sense Modes

It is possible to enable both LED driving and touch-sense functions in a single time frame. If both functions are enabled, up to 7 time slices are configurable for the LED function, and the last time slice is reserved for touch-sensing function.

The touch-sense function is time-multiplexed with the LED function on enabled LINE[x]/TSIN[x] pins. During the touch-sense time slice (COLA), the corresponding LINE[s] output remains active for the other TSIN pins which are not on active pad turn. In a typical application, COLA is not used and the oscillation is generated by the internal pad structure only. The bits in LINE_A will determine whether the pads, that are not being measured in the given COLA time slice, have a floating or 0V value. This setting usually has a serious effect on the sensitivity and noise robustness of the touch pads.

Refer to **Section 12.2** and **Section 12.3** for more details on time slice allocation and configuration.

12.6 Service Request Processing

There are three interrupts triggered by LEDTS kernel, all assigned on same node: 1) time slice event, 2) (extended) time frame event, 3) autoscan time period event. The flags are set on event or when CLK_PS is set from 0 regardless of whether the corresponding interrupt is enabled or not. When enabled, the event (including setting of CLK_PS from 0) activates the SR0 interrupt request from the kernel.

Table 12-3 lists the interrupt event sources from the LEDTS, and the corresponding event interrupt enable bit and flag bit.

Table 12-3 LEDTS Interrupt Events

Event	Event Interrupt Enable Bit	Event Flag Bit
Start of Time Slice	GLOBCTL.ITS_EN	EVFR.TSF
Start of (Extended) Time Frame[1)]	GLOBCTL.ITF_EN	EVFR.TFF
Start of Autoscan Time Period	GLOBCTL.ITP_EN	EVFR.TPF

1) In case of consecutive pad turns enabled on same TSIN[x] pin by ACCCNT bit-field, interrupt is not triggered on a time frame – but on the extended time frame.

Table 12-4 shows the interrupt node assignment for each LEDTS interrupt source.

Table 12-4 LEDTS **Events' Interrupt Node Control**

Event	Interrupt Node Enable Bit	Interrupt Node Flag Bit	Node ID
Start of Time Slice	LEDTS0.SR0	LEDTS0.SR0	102
Start of (Extended) Time Frame			
Start of Autoscan Time Period			

12.7 Debug Behavior

The LEDTS timers/counters LEDTS-counter and TS-counter can be enabled (together) for suspend operation when debug mode becomes active (indicated by HALTED signal from CPU).

In debug suspend mode, write and read access are possible. Writing to bit fields that require **GLOBCTL**.CLK_PS = 0 will still be ignored.

At the onset of debug suspend, these counters stop counting (retains the last value) for the duration of the device in debug mode. The function that was active in current time slice on the onset of debug suspend, continues to be active. When debug suspend is revoked, the kernel would resume operation according to latest SFR settings.

XMC4[12]00

12.8 Power, Reset and Clock

The LEDTS kernel is clocked and accessible on the peripheral bus frequency. User should set up consistent time-slice durations for correct function by ensuring a constant frequency input clock when the kernel is in operation. It is recommended to set the input clock ledts_clk to highest frequency where possible, for optimal touch-sensing accuracy.

Kernel is in operation in active mode except power-down modes where touch-sensing and LED functions are not available.

12.9 Initialisation and System Dependencies

This section provides hints for enabling the LEDTS functions and using them.

12.9.1 Function Enabling

It is recommended to set up all configuration for the LEDTS in all SFRs before write **GLOBCTL** SFR to enable and start LED and/or touch-sense function(s).

*Note: SFR bits especially affecting the LEDTS-counter configuration for LED/touch-sense function can only be written when the counter is not running i.e. CLK_PS = 0. Refer to SFR bit description **Section 12.10**.*

Enable LED Function Only

To enable LED function only: set LD_EN, clear TS_EN.

Initialization after reset:

```
GLOBCTL = 0bXXXXXXXX XXXXXXXX XXX00000 0000XX10;
     //set LD_EN and start LEDTS-counter on prescaled clock
     //CLK_PS != 0)
```

Re-configuration during run-time:

```
GLOBCTL &= 0x0000X00X;//stop LEDTS-counter by clearing prescaler
GLOBCTL = 0bXXXXXXXX XXXXXXXX XXX00000 0000XX10;
```

Enable Touch-Sense Function Only

To enable touch-sense function only: clear LD_EN, set TS_EN.

Initialization after reset:

```
GLOBCTL = 0bXXXXXXXX XXXXXXXX XXX00000 0000XX01;
     //set TS_EN and start LEDTS-counter on prescaled clock
     //(CLK_PS != 0)
```

Re-configuration during run-time:

```
GLOBCTL &= 0x0000X00X;//stop LEDTS-counter by clearing prescaler
GLOBCTL = #0bXXXXXXXX XXXXXXXX XXX00000 0000XX01;
```

Enable Both LED and Touch-Sense Function

To enable both functions: set LD_EN, set TS_EN.

Initialization after reset:

```
GLOBCTL = 0bXXXXXXXX XXXXXXXX XXX00000 0000XX11;
     //set TS_EN and start LEDTS-counter on prescaled clock
     //(CLK_PS != 0)
```

Re-configuration during run-time:

```
GLOBCTL &= 0x0000X00X;//stop LEDTS-counter by clearing prescaler
GLOBCTL = 0bXXXXXXXX XXXXXXXX XXX00000 0000XX11;
```

12.9.2 Interpretation of Bit Field FNCOL

The interpretation of the FNCOL bit field can be handled by software. The following example where six time slices are enabled (per time frame), with five LED columns and touch-sensing enabled, illustrates this (**Table 12-5**).

The FNCOL bit field provides information on the function/column active in the previous time slice. With this information, software can determine the active function/column in

current time slice and prepare the necessary values (to be shadow-transferred) valid for the next time slice.

Referring to the example below, when the FNCOL bit field is 111_B, it can be derived that the touch-sensing function/column was active in the previous time slice and therefore the current active column is LED COL[4]. Hence, the software can update the shadow line and compare registers for LED COL[3] so that these changes will be reflected when LED COL[3] gets activated in the next time slice.

Table 12-5 Interpretation of FNCOL Bit Field

FNCOL	Active Function / Column in Current Time Slice	SW Prepare via “Shadow” Registers for Function / Column of Next Time Slice
111_H	LED COL[4]	LED COL[3]
010_H	LED COL[3]	LED COL[2]
011_H	LED COL[2]	LED COL[1]
100_H	LED COL[1]	LED COL[0]
101_H	LED COL[0]	Touch Input Line TSIN[PADT]
110_H	Touch Input Line TSIN[PADT]	LED COL[4]

12.9.3 LEDTS Timing Calculations

LEDTS main timing or duration formulation are provided in following.

Count-Rate (CR):

$$CR = (fCLK) \div (PREscaler) \tag{12.1}$$

where fCLK = LEDTS module input clock; PREscaler = **GLOBCTL**.CLK_PS

Time slice duration (TSD):

$$TSD = 2^8 \div (CR) \tag{12.2}$$

Time frame duration (TFD):

$$TFD = (\text{Number of time slices}) \times TSD \tag{12.3}$$

Extended TFD:

$$\text{ExtendedTFD} = ACCCNT \times TFD \tag{12.4}$$

where ACCCNT = **FNCTL**.ACCCNT

Autoscan time period duration (TPD):

$$TPD = (\text{Number of touch-sense inputs TSIN[x]}) \times TFD \tag{12.5}$$

LED drive active duration:

$$\text{LED Drive Active Duration} = \text{TSD} \times \text{Compare\_VALUE} \div 2^8 \quad (12.6)$$

Touch-sense drive active duration:

$$\text{Touch-sense Drive Active Duration} = \text{TSD} \times (2^8 - \text{Compare\_VALUE}) \div 2^8 \quad (12.7)$$

12.9.4 Time-Multiplexed LED and Touch-Sense Functions on Pin

Some hints are provided regarding the time-multiplexed usage of a pin for LED and touch-sense function:

- The maximum number of LED columns = 7 when touch-sense function is also enabled.
- If enabled by pin, COLA outputs HIGH to enable external R (resistor) as pull-up for touch-sense function.
- During touch-sense time slice, it is recommended to set LED lines to output LOW.
- During LED time slice, COLA outputs LOW and will sink current if connected lines output HIGH.
- The effective capacitance for each TSIN[x] depends largely on what is connected to the pin and the application board layout. All touch-pads for the application should be calibrated for robust touch-detection.

12.9.5 LEDTS Pin Control

The user may flexibly assign pins as provided by PORTS SFRs, for the LEDTS functions:

- COL[x] (for LED column control)
- LINE[x] (for LED line control)
- TSIN[x] (for touch-sensing)

Refer also to **Section 12.3** for more considerations with regards to which COL[x] and/or LINE[x]/TSIN[x] will be active based on user configuration.

User code must configure the assigned LED pin PORTS SFR alternate output selection for the LED function, see **Table 12-6** and **Figure 12-8**.

For the touch-sense function, it is also required to configure the PORTS SFRs to enable the hardware function on TSIN[x] pin (similarly alternate output COLA). However, the LEDTS will provide some pin over-rule controls to the assigned touch-sense pin with active pad turn, see **Table 12-6** and **Figure 12-8**.

Table 12-6 LEDTS Pin Control Signals

Function	ld/ts_en	ledts_fn	Pin	Control of Assigned Pin
LED column	LD_EN = 1	0 = LED	Enable COL[x]; Passive level on COL[the rest]. If TS_EN = 1, COLA = 0.	PORTS SFR setting
LED line	LD_EN = 1	0 = LED	LINE[x] = internal line register value latched from bit field LINE_[x]	PORTS SFR setting
Touch-sense	TS_EN = 1	1 = Touch-sense	Enable TSIN[x] for oscillation. All other TSIN pins output line value. Passive level on COL[the rest] except COLA = 1.	Hardware over-rule on pad_turn_x[1] for active duration: - Enable pull-up, disable pull-down (pull over-rule can be disabled by bit EPULL) - Set to output mode (both input, output stages enabled) - Enable open-drain

1) For the other pad inputs not on turn, there is no HW over-rule which means the PORTS SFR setting is active.

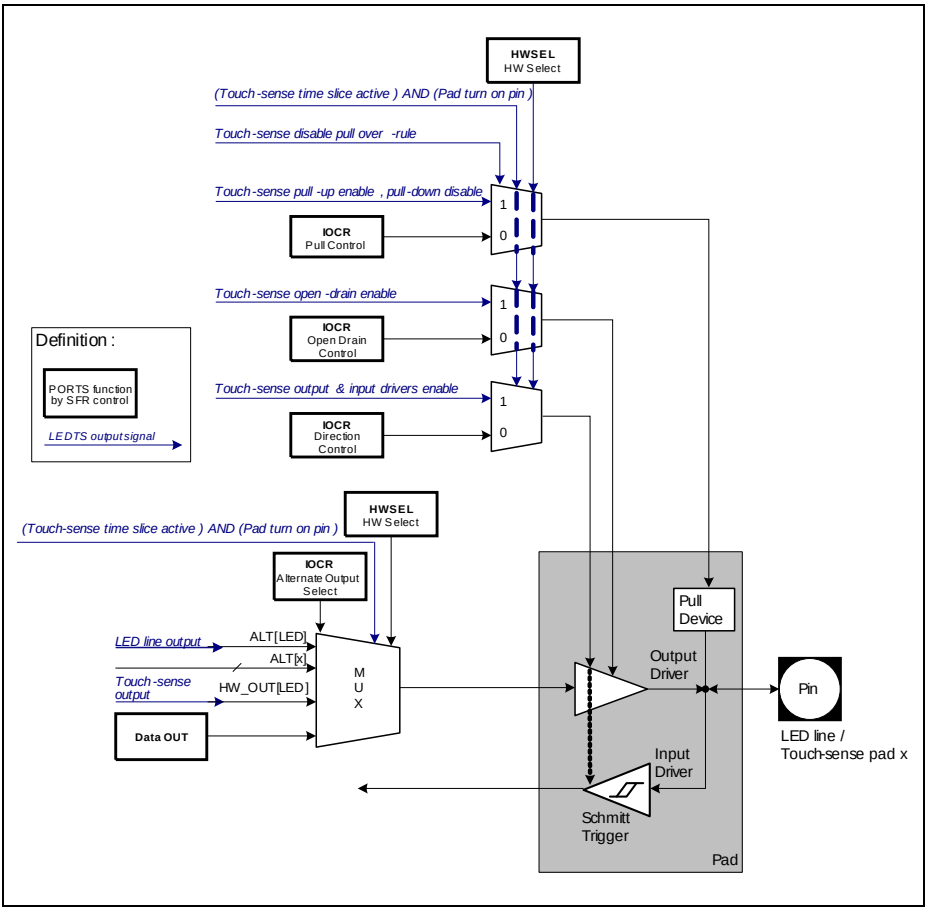


Figure 12-8 Over-rule Control on Pin for Touch-Sense Function

12.9.6 Software Hints

This section provides some useful software hints:

- Compare value 00_H enables oscillation for the full duration of the time slice, whereas FF_H disables oscillation.
- In order to maximize the resolution of the oscillation window, compare value should be selected to maximize the oscillation count without overflowing the TS-counter.
- Valid pad detection period (the time required to detect a valid touch on a pad) can be extended by:

  - enabling dummy LED columns (without assigning/setting the LED column pins)
  - selecting bigger pre-scale factor (**GLOBCTL**.CLK_PS)
  - accumulating the number of pad oscillations (**FNCTL**.ACCCNT)
- Valid pad detection period can be reduced by:
  - selecting smaller pre-scale factor (**GLOBCTL**.CLK_PS)
  - reducing the number of accumulations for pad oscillations (**FNCTL**.ACCCNT)

12.9.7 Hardware Design Hints

This section provides some hardware design hints:

- Touch button oscillation frequency changes when the value of the external pull-up resistor (connected to COLA pin) changes. This results in different sensitivity of the touch button as well as the crosstalk between the adjacent touch buttons.
  - A suitable pull-up resistor should be selected to balance the sensitivity of the touch button and the accuracy of the detection.
- The presence of LEDs modifies the equivalent capacitance for a touch pad. A larger number of LEDs connected to a touch pad will increase the self-capacitance of the pad. This makes the pad less sensitive.
- If possible, LEDs should be located near to the touch pads, to reduce the additional parasitic capacitance introduced by the traces.

12.10 Registers

Registers Overview

The absolute register address is calculated by adding:

Module Base Address + Offset Address

Table 12-7 Registers Address Space

Module	Base Address	End Address	Note
LEDTS0	4801 0000$_H$	4801 00FF$_H$	

The prefix "**LEDTSx**_" is added to the register names in this table to indicate they belong to the LEDTS kernel.

Note: Register bits marked "w" always deliver 0 when read.

Access rights within the address range of an LEDTS kernel:

- Read or write access to defined register addresses: U, PV
- Accesses to empty addresses: nBE

Table 12-8 Register Overview of LEDTS

Short Name	Description	Offset Addr.	Access Mode		Description See
			Read	Write	
ID	Module Identification Register	0000$_H$	U, PV	U, PV	Page 12-23
GLOBCTL	Global Control Register	0004$_H$	U, PV	U, PV	Page 12-24
FNCTL	Function Control Register	0008$_H$	U, PV	U, PV	Page 12-26
EVFR	Event Flag Register	000C$_H$	U, PV	U, PV	Page 12-30
TSVAL	Touch-Sense TS-Counter Value	0010$_H$	U, PV	U, PV	Page 12-31
LINE0	Line Pattern Register 0	0014$_H$	U, PV	U, PV	Page 12-32
LINE1	Line Pattern Register 1	0018$_H$	U, PV	U, PV	Page 12-33
LDCMP0	LED Compare Register 0	001C$_H$	U, PV	U, PV	Page 12-33
LDCMP1	LED Compare Register 1	0020$_H$	U, PV	U, PV	Page 12-34
TSCMP0	Touch-Sense Compare Register 0	0024$_H$	U, PV	U, PV	Page 12-35

Table 12-8 Register Overview of LEDTS

Short Name	Description	Offset Addr.	Access Mode		Description See
			Read	Write	
TSCMP1	Touch-Sense Compare Register 1	0028_H	U, PV	U, PV	Page 12-35
Reserved	Reserved	$002C_H$ - $1FFC_H$	nBE	nBE	

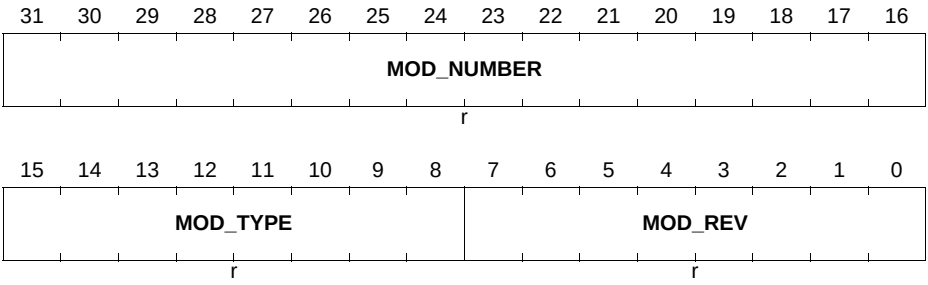
12.10.1 Registers Description

The LEDTS SFRs are organized into registers for global initialization control, functional control comprising TS-counter value, line pattern and compare value registers.

LEDTSx_ID

The module identification register indicate the function and the design step of the peripheral.

ID
Module Identification Register (00_H) Reset Value:00AB C0XX$_H$



Field	Bits	Type	Description
MOD_REV	[7:0]	r	**Module Revision Number** MOD_REV defines the revision number. The value of a module revision starts with a 00_H (first revision).
MOD_TYPE	[15:8]	r	**Module Type** This bit field is $C0_H$. It defines the module as a 32-bit module.

Field	Bits	Type	Description
MOD_NUMBER	[31:16]	r	**Module Number Value** This bit field defines the module identification number.

GLOBCTL

The GLOBCTL register is used to initialize the LEDTS global controls.

GLOBCTL
Global Control Register **(04_H)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CLK_PS															
rw															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ITP_EN	ITF_EN	ITS_EN	FENVAL	MASKVAL			SUSCFG	0				ENSYNC	CMTR	LD_EN	TS_EN
rw	rw	rw	rw	rw			rw	r				rw	rw	rw	rw

Field	Bits	Type	Description
TS_EN[1)]	0	rw	**Touch-Sense Function Enable** Set to enable the kernel for touch-sense function control when CLK_PS is set from 0.
LD_EN[1)]	1	rw	**LED Function Enable** Set to enable the kernel for LED function control when CLK_PS is set from 0.
CMTR[1)]	2	rw	**Clock Master Disable** 0_B Kernel generates its own clock for LEDTS-counter based on SFR setting 1_B LEDTS-counter takes its clock from another master kernel If this bit is set, the bit field **GLOBCTL**.CLK_PS needs to be set to a non-zero value to activate the kernel, even though it is taking its clock from the master kernel.

Field	Bits	Type	Description
ENSYNC[1)]	3	rw	**Enable Autoscan Time Period Synchronization** 0_B No synchronization 1_B Synchronization enabled on Kernel0 autoscan time period
SUSCFG	8	rw	**Suspend Request Configuration** 0_B Ignore suspend request 1_B Enable suspend according to request This bit is restored to default with Debug Reset.
MASKVAL	[11:9]	rw	**Mask Number of LSB Bits for Event Validation** This defines the number of LSB bits to mask for TS-counter and shadow TS-counter comparison when Time Frame validation is enabled. 0_D Mask LSB bit 1_D Mask 2 LSB bits ... 7_D Mask 8 LSB bits
FENVAL	12	rw	**Enable (Extended) Time Frame Validation** When enabled, TS-counter and shadow TS-counter values are compared to validate a Time Frame event for set flag and interrupt request. When validation fail, TFF flag does not get set and interrupt is not requested. 0_B Disable 1_B Enable
ITS_EN	13	rw	**Enable Time Slice Interrupt** 0_B Disable 1_B Enable
ITF_EN	14	rw	**Enable (Extended) Time Frame Interrupt** 0_B Disable 1_B Enable
ITP_EN	15	rw	**Enable Autoscan Time Period Interrupt** 0_B Disable 1_B Enable (valid only for case of hardware-enabled pad turn control)

Field	Bits	Type	Description
CLK_PS	[31:16]	rw	**LEDTS-Counter Clock Pre-Scale Factor** The constant clock input is prescaled according to setting. 0_D No clock 1_D Divide by 1 ... 65535_DDivide by 65535 This bit can only be set to any other value (from 0) provided at least one of touch-sense or LED function is enabled. The LEDTS-counter starts running on the input clock from reset/reload value based on enabled function(s) (and NR_LEDCOL). Refer **Section 12.3** for details. When this bit is clear to 0 from other value, the LEDTS-counter stops running and resets.
0	[7:4]	r	**Reserved** Read as 0; should be written with 0.

1) This bit can only be modified when bit CLK_PS = 0.

FNCTL

The FNCTL control register provides control bits for the LED and Touch Sense functions.

FNCTL
Function Control Register **(08_H)** **Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
NR_LEDCOL			COL LEV	NR_TSIN			TSC TRS AT	TSC TRR	TSOEXT		TSC CMP	ACCCNT			
rw			rw	rw			rw	rw	rw		rw	rw			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0								FNCOL			EPU LL	PAD TSW	PADT		
r								rh			rw	rw	rwh		

Field	Bits	Type	Description
PADT	[2:0]	rwh	**Touch-Sense TSIN Pad Turn** This is the TSIN[x] pin that is next or currently active in pad turn. When PADTSW = 0, the value is updated by hardware at the end of touch-sense time slice. Software write is always possible. However, PADT must not be written during an active pad turn. 0_D TSIN0 ... 7_D TSIN7
PADTSW[1)]	3	rw	**Software Control for Touch-Sense Pad Turn** 0_B The hardware automatically enables the touch-sense inputs in sequence round-robin, starting from TSIN0. 1_B Disable hardware control for software control only. The touch-sense input is configured in bit PADT.
EPULL	4	rw	**Enable External Pull-up Configuration on Pin COLA** When set, the internal pull-up over-rule on active touch-sense input pin is disabled. 0_B HW over-rule to enable internal pull-up is active on TSIN[x] for set duration in touch-sense time slice. With this setting, it is not specified to assign the COLA to any pin. 1_B Enable external pull-up: Output 1 on pin COLA for whole duration of touch-sense time slice. *Note: Independent of this setting, COLA always outputs 1 for whole duration of touch-sense time slice.*
FNCOL	[7:5]	rh	**Previous Active Function/LED Column Status** Shows the active function / LED column in the previous time slice. Updated on start of new time-slice when LEDTS-counter 8LSB over-flows. Controlled by latched value of the internal DE-MUX, see **Figure 12-4**.

Field	Bits	Type	Description
ACCCNT[1)]	[19:16]	rw	**Accumulate Count on Touch-Sense Input** Defines the number of times a touch-sense input/pin is enabled in touch-sense time slice of consecutive frames. This provides to accumulate oscillation count on the TSIN[x]. 0_D 1 time 1_D 2 times ... 15_D 16 times
TSCCMP	20	rw	**Common Compare Enable for Touch-Sense** 0_B Disable common compare for touch-sense 1_B Enable common compare for touch-sense
TSOEXT	[22:21]	rw	**Extension for Touch-Sense Output for Pin-Low-Level** 00_B Extend by 1 ledts_clk 01_B Extend by 4 ledts_clk 10_B Extend by 8 ledts_clk 11_B Extend by 16 ledts_clk
TSCTRR	23	rw	**TS-Counter Auto Reset** 0_B Disable TS-counter automatic reset 1_B Enable TS-counter automatic reset to 00H on the first pad turn of a new TSIN[x]. Triggered on compare match in time slice.
TSCTRSAT	24	rw	**Saturation of TS-Counter** 0_B Disable 1_B Enable. TS-counter stops counting in the touch-sense time slice(s) of the same (extended) frame when it reaches FFH. Counter starts to count again on the first pad turn of a new TSIN[x], triggered on compare match.
NR_TSIN[1)]	[27:25]	rw	**Number of Touch-Sense Input** Defines the number of touch-sense inputs TSIN[x]. Used for the hardware control of pad turn enabling. 0_D 1 ... 7_D 8

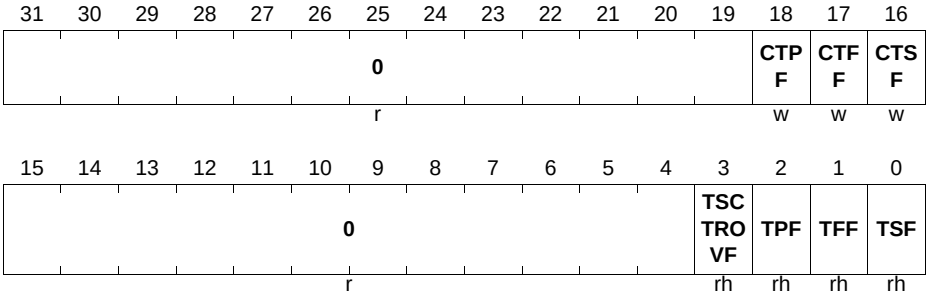
Field	Bits	Type	Description
COLLEV	28	rw	**Active Level of LED Column** 0_B Active low 1_B Active high
NR_LEDCOL[1)]	[31:29]	rw	**Number of LED Columns** Defines the number of LED columns. 000_B 1 LED column 001_B 2 LED columns 010_B 3 LED columns 011_B 4 LED columns 100_B 5 LED columns 101_B 6 LED columns 110_B 7 LED columns 111_B 8 LED columns (if bit TS_EN = 1, a maximum number of 7 LED columns is allowed. Combination of TS_EN = 1 and NR_LEDCOL = 111_B is disallowed) *Note: LED column is enabled in sequence starting from highest column number. If touch-sense function is not enabled, COLA is activated in last time slice.*
0	[15:8]	r	**Reserved** Read as 0; should be written with 0.

1) This bit can only be modified when bit CLK_PS = 0.

EVFR

The EVFR register contains the status flags for events and control bits for requesting clearance of event flags.

EVFR
Event Flag Register **($0C_H$)** **Reset Value: $0000\ 0000_H$**



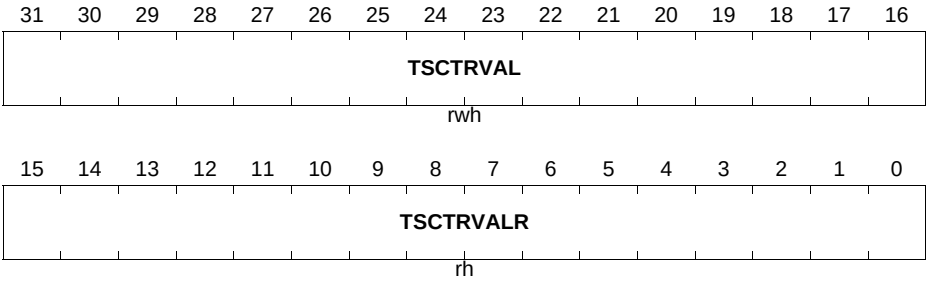
Field	Bits	Type	Description
TSF	0	rh	**Time Slice Interrupt Flag** Set on activation of each new time slice, including when bit CLK_PS is set from 0. To be cleared by software.
TFF	1	rh	**(Extended) Time Frame Interrupt Flag** Set on activation of each new (extended) time frame, including when bit CLK_PS is set from 0.
TPF	2	rh	**Autoscan Time Period Interrupt Flag** Set on activation of each new time period, including when bit CLK_PS is set from 0. This bit will never be set in case of hardware pad turn control is disabled (bit PADTSW = 1).
TSCTROVF	3	rh	**TS-Counter Overflow Indication** This bit indicates whether a TS-counter overflow has occurred. This bit is cleared on new pad turn, triggered on compare match. 0_B No overflow has occurred. 1_B The TS-counter has overflowed at least once.
CTSF	16	w	**Clear Time Slice Interrupt Flag** 0_B No action. 1_B Bit TSF is cleared. Read always as 0.

Field	Bits	Type	Description
CTFF	17	w	**Clear (Extended) Time Frame Interrupt Flag** 0_B No action. 1_B Bit TFF is cleared. Read always as 0.
CTPF	18	w	**Clear Autoscan Time Period Interrupt Flag** 0_B No action. 1_B Bit TPF is cleared. Read always as 0.
0	[31:19], [15:4]	r	**Reserved** Read as 0; should be written with 0.

TSVAL

The TSVAL register holds the current and shadow touch sense counter values.

TSVAL
Touch-sense TS-Counter Value (10_H) Reset Value: 0000 0000$_H$



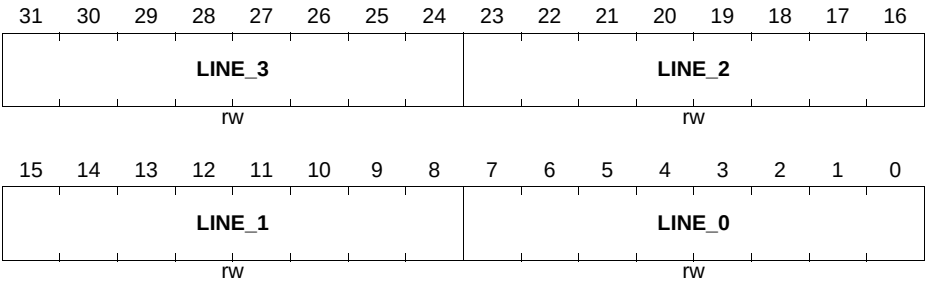
Field	Bits	Type	Description
TSCTRVALR	[15:0]	rh	**Shadow TS-Counter Value (Read)** This is the latched value of the TS-counter (on every extended time frame event). It shows the latest valid oscillation count from the last completed time slice.

Field	Bits	Type	Description
TSCTRVAL	[31:16]	rwh	**TS-Counter Value** This is the actual TS-counter value. It can only be written when no pad turn is active. The counter may be enabled for automatic reset once per (extended) frame on the start of a new pad turn on the next TSIN[x] pin.

LINEx (x = 0-1)

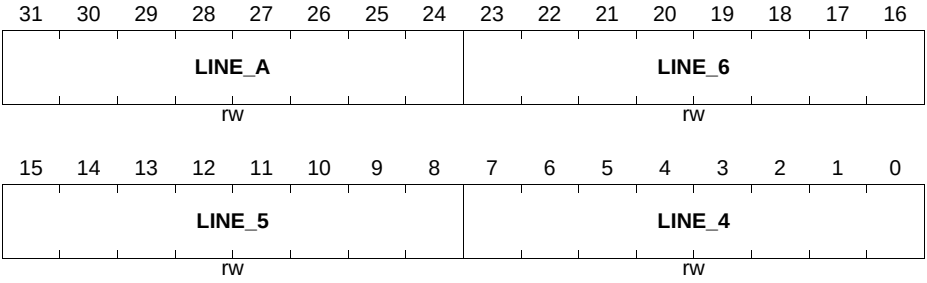
The LINEx registers hold the values that are output to the respective line pins during their active column period.

LINE0
Line Pattern Register 0 **(14_H)** **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
LINE_0, **LINE_1,** **LINE_2,** **LINE_3**	[7:0], [15:8], [23:16], [31:24]	rw	**Output on LINE[x]** This value is output on LINE[x] to pin when LED COL[x] is active.

LINE1
Line Pattern Register 1 **(18_H)** **Reset Value: 0000 0000$_H$**

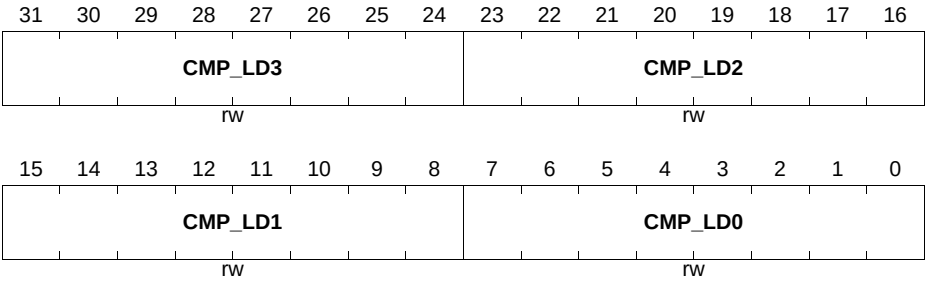


Field	Bits	Type	Description
LINE_4, LINE_5, LINE_6	[7:0], [15:8], [23:16]	rw	**Output on LINE[x]** This value is output on LINE[x] to pin when LED COL[x] is active.
LINE_A	[31:24]	rw	**Output on LINE[x]** This value is output on LINE[x] to pin when LED COLA or touch-sense time slice is active.

LDCMPx (x = 0-1)

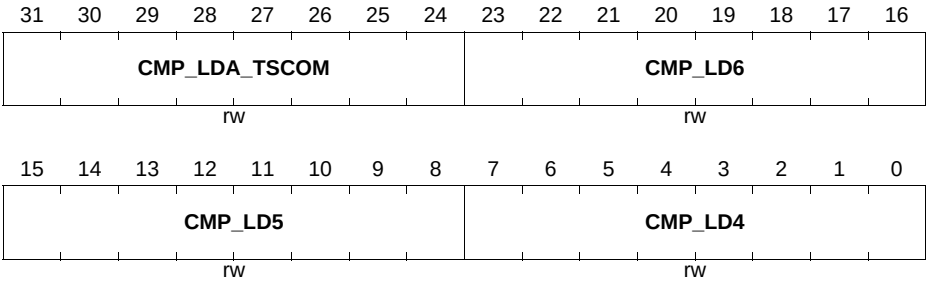
The LDCMPx registers hold the COMPARE values for their respective LED columns. These values are used for LED brightness control.

LDCMP0
LED Compare Register 0 **($1C_H$)** **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
CMP_LD0, CMP_LD1, CMP_LD2, CMP_LD3	[7:0], [15:8], [23:16], [31:24]	rw	**Compare Value for LED COL[x]**

LDCMP1
LED Compare Register 1 **(20_H)** **Reset Value: 0000 0000_H**

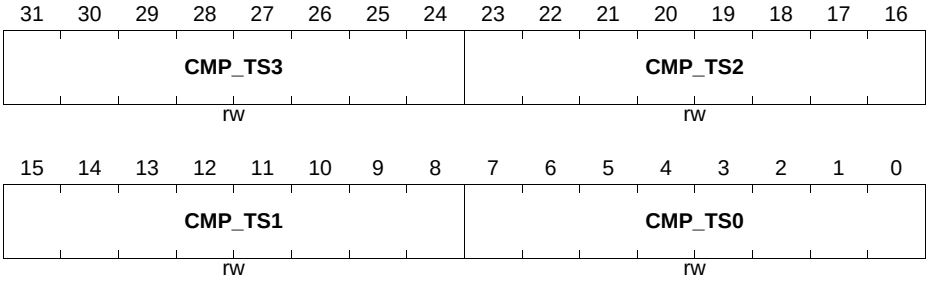


Field	Bits	Type	Description
CMP_LD4, CMP_LD5, CMP_LD6	[7:0], [15:8], [23:16]	rw	**Compare Value for LED COL[x]**
CMP_LDA_TSCOM	[31:24]	rw	**Compare Value for LED COLA / Common Compare Value for Touch-sense Pad Turns** LED function The compare value for LED COLA is only valid when touch-sense function is not enabled. Touch-sense function The common compare value for touch-sense pad turns is enabled by set TSCCMP bit. When enabled for common compare, settings in SFRs LEDTS_TSCMP0,1 are not referenced.

TSCMPx (x = 0-1)

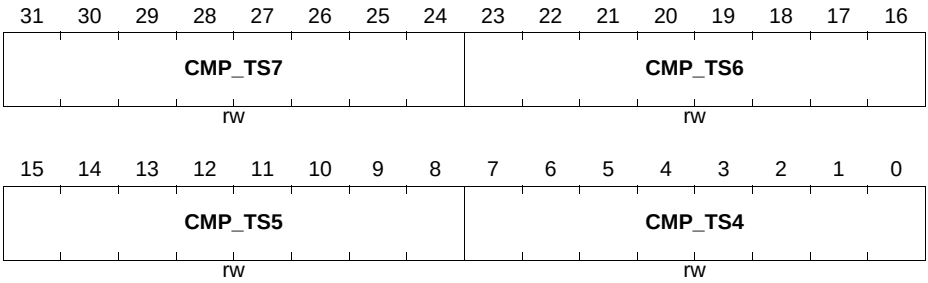
The TSCMPx registers hold the COMPARE values for their respective touch pad input lines. These values determine the size of the pad oscillation window for each pad input lines during their pad turn.

TSCMP0
Touch-sense Compare Register 0 (24_H) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
CMP_TS0, CMP_TS1, CMP_TS2, CMP_TS3	[7:0], [15:8], [23:16], [31:24]	rw	**Compare Value for Touch-Sense TSIN[x]**

TSCMP1
Touch-sense Compare Register 1 (28_H) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
CMP_TS4, CMP_TS5, CMP_TS6, CMP_TS7	[7:0], [15:8], [23:16], [31:24]	rw	**Compare Value for Touch-Sense TSIN[x]**

12.11 Interconnects

The LEDTS has interconnection to other peripherals enabling higher level of automation without requiring software. **Table 12-9** provides a list of the pin connections.

LEDTSx.FN is an output signal denoting LEDTS active function. This signal can be used as a source for VADC request gating and background gating. LEDTSx.SR0 is an output signal denoting LEDTS kernel service request. This signal can be used as an interrupt trigger .

Table 12-9 Pin Connection

Input/Output	I/O	Connected To	Description
LEDTS0.TSIN0	I	P2.2	LEDTS Touch-sense 0 input
LEDTS0.TSIN1	I	P2.3	LEDTS Touch-sense 1 input
LEDTS0.TSIN2	I	P2.4	LEDTS Touch-sense 2 input
LEDTS0.TSIN3	I	P2.5	LEDTS Touch-sense 3 input
LEDTS0.TSIN4	I	P2.8	LEDTS Touch-sense 4 input
LEDTS0.TSIN5	I	P2.9	LEDTS Touch-sense 5 input
LEDTS0.TSIN6	I	P2.15	LEDTS Touch-sense 6 input
LEDTS0.LINE0	O	P2.2	LEDTS Line 0 output
LEDTS0.LINE1	O	P2.3	LEDTS Line 1 output
LEDTS0.LINE2	O	P2.4	LEDTS Line 2 output
LEDTS0.LINE3	O	P2.5	LEDTS Line 3 output
LEDTS0.LINE4	O	P2.8	LEDTS Line 4 output
LEDTS0.LINE5	O	P2.9	LEDTS Line 5 output
LEDTS0.LINE6	O	P2.15	LEDTS Line 6 output
LEDTS0.COL0	O	P0.9 P2.1	LEDTS Column 0 output
LEDTS0.COL1	O	P0.10 P2.0	LEDTS Column 1 output
LEDTS0.COL2	O	P0.0 P2.7	LEDTS Column 2 output
LEDTS0.COL3	O	P0.1 P2.6	LEDTS Column 3 output
LEDTS0.EXTENDED0	O	P2.2.HW0	
LEDTS0.EXTENDED1	O	P2.3.HW0	
LEDTS0.EXTENDED2	O	P2.4.HW0	

Table 12-9 Pin Connection (cont'd)

Input/Output	I/O	Connected To	Description
LEDTS0.EXTENDED3	O	P2.5.HW0	
LEDTS0.EXTENDED4	O	P2.8.HW0	
LEDTS0.EXTENDED5	O	P2.9.HW0	
LEDTS0.EXTENDED6	O	P2.15.HW0	
LEDTS.FN	O	VADC.GxREQGTJ	1) VADC request gating
		VADC.BGREQGTJ	2) VADC background gating input J

13 Universal Serial Bus (USB)

The USB module is a device-only controller, fully compliant with the USB 2.0 Specification.

The USB core's USB 2.0 configurations support full-speed (12-Mbps) transfers.

The USB core is optimized for the following applications and systems:

- Portable electronic devices
- Point-to-point applications (direct connection to FS device)

References:

[10] USB 2.0 specification (April 27, 2000).

Table 13-1 Abbreviations

DWORD	32-bit Data Word
FS	Full Speed
MAC	Media Access Controller
PHY	Physical Layer
SOF	Start of Frame

13.1 Overview

This section describes the features and provides a block diagram of the USB module.

13.1.1 Features

The USB module includes the following features:

- Complies with the USB 2.0 Specification
- Self-powered USB Device
- Support for the Full-Speed (12-Mbps) mode
- Provides a USB FS PHY interface
- Supports up to 7 bidirectional endpoints, including control endpoint 0
- Supports SOFs in Full-Speed modes.
- Supports clock gating for power saving
- Supports USB suspend/resume
- Supports USB soft disconnect
- Supports DMA mode in:
  - Descriptor-Based Scatter/Gather DMA operation
  - Buffer DMA operation
- 2 Kbytes of RAM for data FIFO consisting of 512 DWORDs

- Dedicated transmit FIFO for each of the device IN endpoints in Slave and DMA modes. Each FIFO can hold multiple packets.
- Supports packet-based, Dynamic FIFO memory allocation for endpoints for small FIFOs and flexible, efficient use of RAM.
- Provides support to change an endpoint's FIFO memory size during transfers.
- Supports endpoint FIFO sizes that are not powers of 2, to allow the use of contiguous memory locations.

13.1.2 Block Diagram

Figure 13-1 shows the USB module block diagram.

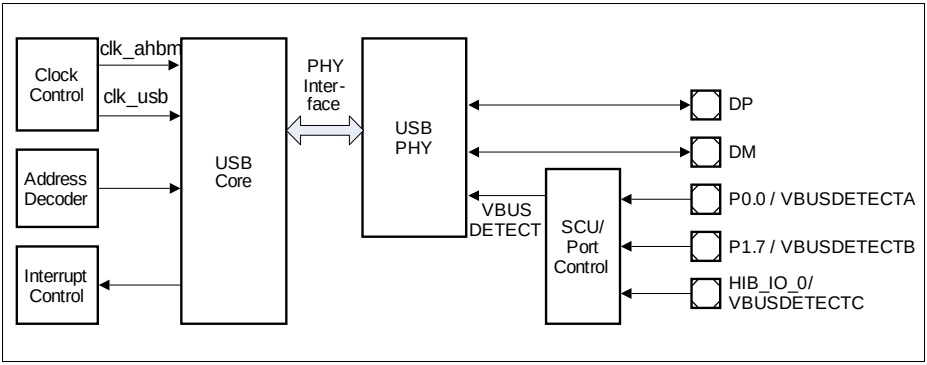


Figure 13-1 USB Module Block Diagram

13.2 Functional Description

This chapter describes the operation modes and the FIFO architecture of the USB module.

13.2.1 USB Device

The USB Device supports Control transfers through the bidirectional endpoint 0, and Bulk, Interrupt or Isochronous tranfers configurable from within the other 6 bidirectional endpoints. Being a self-powered device, it does not require an additional external voltage regulator. VBUSDETECT input signal is used to detect the removal of VBUS by the host.

Software can disconnect the USB Device from the USB through the DCTL.SftDiscon bit.

Figure 13-2 shows the connections of the USB Device.

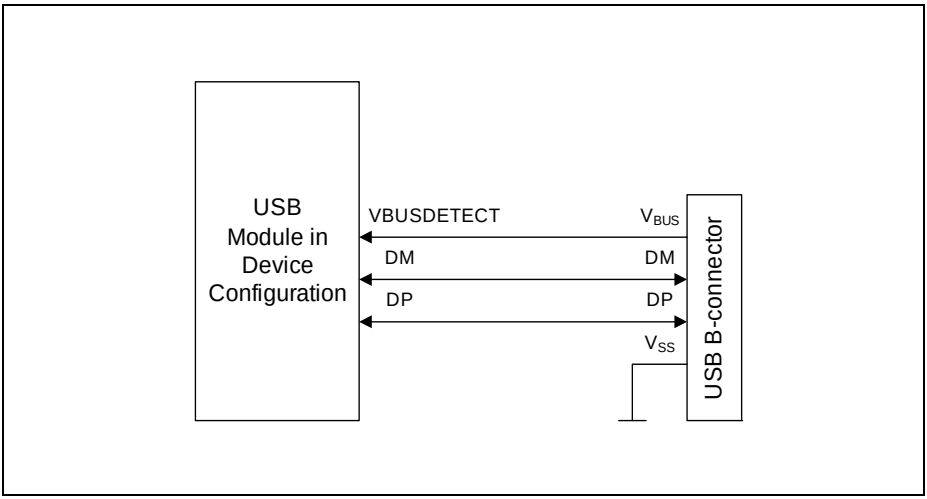


Figure 13-2 USB Device Connections

13.2.2 FIFO Architecture

This section describes the FIFO architecture in a USB Device.

13.2.2.1 Device FIFO Architecture

This section describes the USB device FIFO architecture.

Dedicated Transmit FIFO Operation

The core uses individual transmit FIFOs for each IN endpoint.

The core internally handles underrun condition during transmit and corrupts the packet (inverts the CRC) on the USB. If packet transmission results in an underrun condition (eventually resulting in packet corruption on the USB), the host can time out the endpoint after three consecutive errors.

Single Receive FIFO

The USB device uses a single receive FIFO to receive the data for all the OUT endpoints. The receive FIFO holds the status of the received data packet, such as byte count, data PID and the validity of the received data. The DMA or the application reads the data out of the receive FIFO as it is received.

13.3 Programming Overview

This section provides an overview of the programming options available in the USB core in different modes of operation.

13.3.1 Programming Options on DMA

The application can operate the core in either of two modes:

- In **DMA Mode** — the core fetches the data to be transmitted or updates the received data on the AHB.
- In **Slave Mode** — the application initiates the data transfers for data fetch and store.

The application cannot operate the core using a combination of DMA and Slave simultaneously. The application can operate the DMA in:

- Scatter/Gather mode (a Descriptor-Based mode).
- Buffer-pointer based mode.

13.3.1.1 DMA Mode

The AHB master uses the programmed DMA address (DIEPDMAx/DOEPDMAx register in device mode) to access the data buffers.

Transfer-Level Operation

In DMA mode, the application is interrupted only after the programmed transfer size is transmitted or received (provided the USB core detects no Timeout/CRC Error in Device mode). The application must handle all transaction errors. In Device mode with dedicated FIFOs, all the USB errors are handled by the core itself.

Transaction-Level Operation

This mode is similar to transfer-level operation with the programmed transfer size equal to one packet size (either maximum packet size, or a short packet size). When Scatter/Gather DMA is enabled, the transfer size is extracted from the descriptors.

13.3.1.2 Slave Mode

In Slave mode, the application can operate the USB core either in transaction-level (packet-level) operation or in pipelined transaction-level operation.

Transaction-Level Operation

The application handles one data packet at a time per endpoint in transaction-level operations. Based on the handshake response received on the USB, the application determines whether to retry the transaction or proceed with the next, until the end of the transfer. The application is interrupted on completion of every packet. The application

performs transaction-level operations for a endpoint for a transmission (device: IN) or reception (device: OUT) as shown in **Figure 13-3** and **Figure 13-4**.

Transaction-Level Operation: Device Mode

For an IN transaction, the application enables the endpoint, writes the data packet into the corresponding transmit FIFO, and waits for the packet completion interrupt from the core.

For an OUT transaction, the application enables the endpoint, waits for the packet received interrupt from the core, then empties the packet from the receive FIFO.

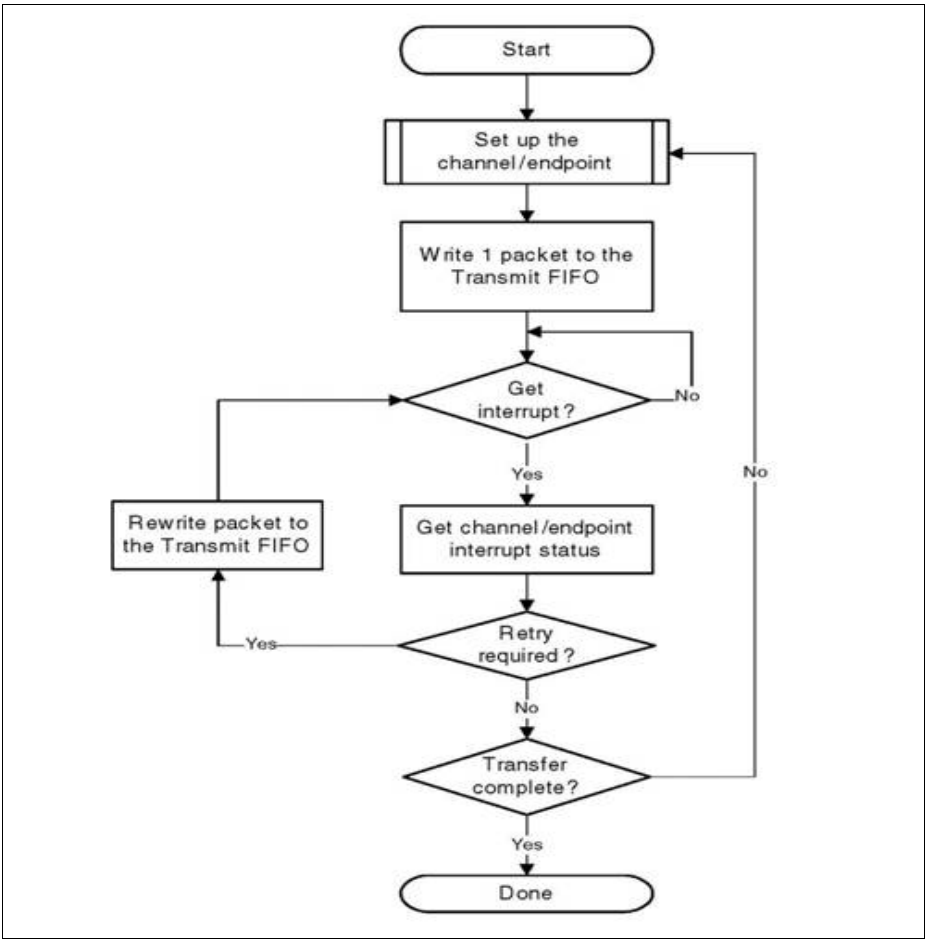


Figure 13-3 Transmit Transaction-Level Operation in Slave Mode

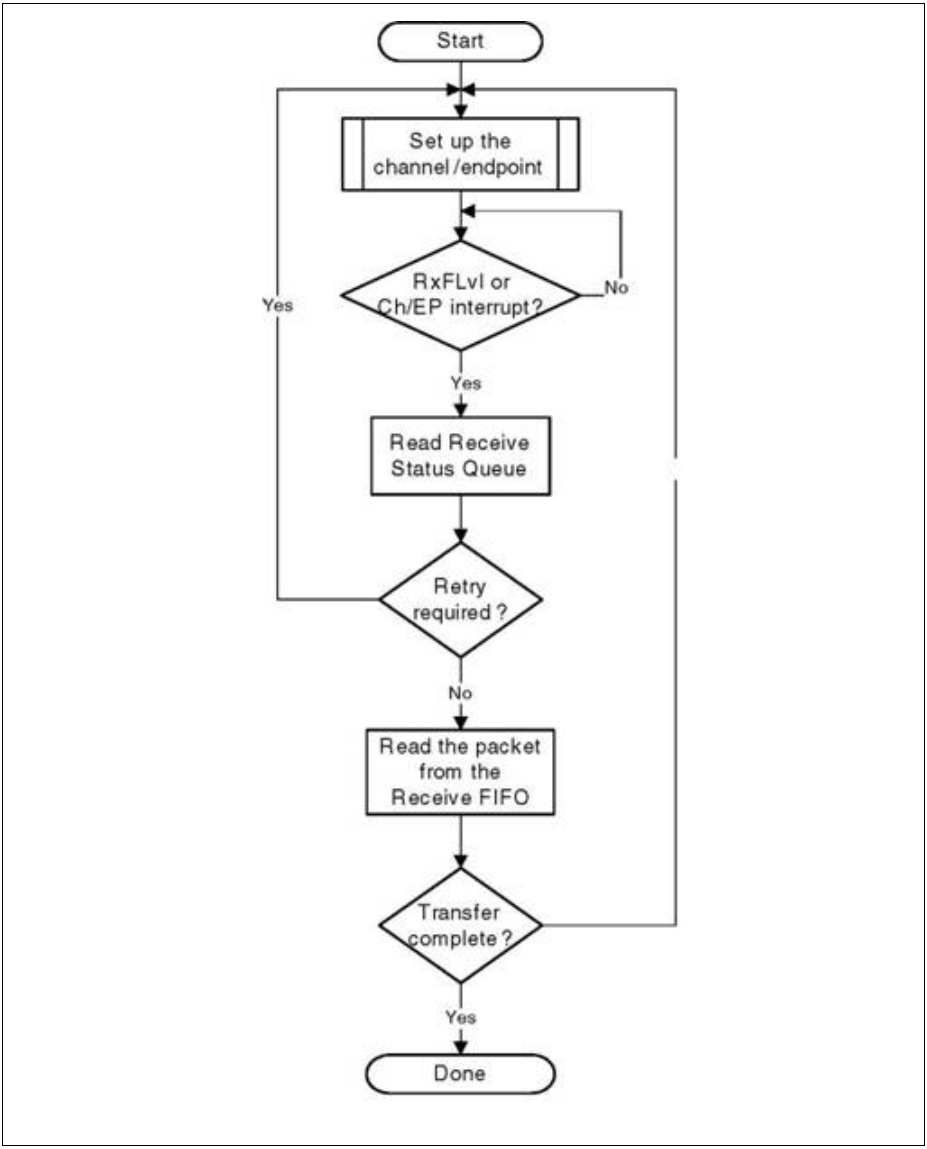


Figure 13-4 Receive Transaction-Level Operation in Slave Mode

Note: The application has to finish writing one complete packet before switching to a different endpoint FIFO. Violating this rule results in an error.

Pipelined Transaction-Level Operation

The application can pipeline more than one transaction (IN or OUT) with pipelined transaction-level operation, which is analogous to Transfer mode in DMA mode. In pipelined transaction-level operation, the application can program the core to perform multiple transactions. The advantage of this mode compared to transaction-level operation is that the application is not interrupted on a packet basis.

Pipelined Transaction-Level Operation: Device Mode

For an IN transaction, the application sets up a transfer and enables the endpoint. The application can write multiple packets back-to-back for the same endpoint into the transmit FIFO, based on available space. It can also pipeline IN transactions for multiple channels by writing into the DIEPCTLx register followed by a packet write to that endpoint. The core writes the endpoint number, along with the last DWORD write for the packet into the Request queue. The core transmits the data in the transmit FIFO when an IN token is received on the USB.

For an OUT transaction, the application sets up a transfer and enables the endpoint. The core receives the OUT data into the receive FIFO, when it has available space. As the packets are received into the FIFO, the application must empty data from it.

From this point on in this chapter, the terms "Pipelined Transaction mode" and "Transfer mode" are used interchangeably.

13.3.2 Core Initialization

If the cable is connected during power-up, the Current Mode of Operation bit in the Core Interrupt register (GINTSTS.CurMod) reflects the mode. The USB core enters Device mode when a "B" plug is connected.

This section explains the initialization of the USB core after power-on. The application must follow the initialization sequence.

1. Program the following fields in the Global AHB Configuration (GAHBCFG) register.
   a) DMA Mode bit
   b) AHB Burst Length field
   c) Global Interrupt Mask bit = 1
   d) RxFIFO Non-Empty (GINTSTS.RxFLvl) (applicable only when the core is operating in Slave mode)
2. Program the following fields in GUSBCFG register.
   a) FS Time-Out Calibration field
   b) USB Turnaround Time field
3. The software can read the GINTSTS.CurMod bit to determine whether the USB core is operating in Device mode. The software then follows **“Device Initialization” on Page 13-11** sequence.

Note: The core is designed to be interrupt-driven. Polling interrupt mechanism is not recommended: this may result in undefined resolutions.

Note: In device mode, just after Power On Reset or a Soft Reset, the GINTSTS.Sof bit is set to 1_B for debug purposes. This status must be cleared and can be ignored.

13.4 Device Programming Overview

This section discusses how to program the DWC_otg core when it is in Device mode.

13.4.1 Device Initialization

As prerequisites, the application must meet the following conditions to set up the device core to handle traffic:

- In Slave mode, GINTMSK.NPTxFEmpMsk, and GINTMSK.RxFLvlMsk must be unset.
- In DMA mode, the GINTMSK.NPTxFEmpMsk, and GINTMSK.RxFLvlMsk interrupts must be masked.

The application must perform the following steps to initialize the core at device on, power on, or after a mode change from Host to Device.

1. Program the following fields in DCFG register.
   a) DescDMA bit
   b) Device Speed
   c) NonZero Length Status OUT Handshake
   d) Periodic Frame Interval (If Periodic Endpoints are supported)
2. Clear the DCTL.SftDiscon bit. The core issues a connect after this bit is cleared.
3. Program the GINTMSK register to unmask the following interrupts.
   a) USB Reset
   b) Enumeration Done
   c) Early Suspend
   d) USB Suspend
   e) SOF
4. Wait for the GINTSTS.USBReset interrupt, which indicates a reset has been detected on the USB and lasts for about 10 ms. On receiving this interrupt, the application must perform the steps listed in **“Initialization on USB Reset” on Page 13-13**.
5. Wait for the GINTSTS.EnumerationDone interrupt. This interrupt indicates the end of reset on the USB. On receiving this interrupt, the application must read the DSTS register to determine the enumeration speed and perform the steps listed in **“Initialization on Enumeration Completion” on Page 13-14**.

At this point, the device is ready to accept SOF packets and perform control transfers on control endpoint 0.

13.4.2 Device Connection

The device connect process varies depending if the VBUS is on or off when the device is connected to the USB cable.

VBUS is on when the device is connected

The device connection flow is as follows:

1. The device triggers the GINTSTS.SessReqInt [bit 30] interrupt bit.
2. When the device application detects the GINTSTS.SessReqInt interrupt, it programs the required bits in the DCFG register.
3. When the host drives reset, the device triggers GINTSTS.USBRst [bit 12] on detecting the reset. The host then follows the USB 2.0 Enumeration sequence.

13.4.3 Device Disconnection

The device session ends when the USB cable is disconnected or if the VBUS is switched off by the host.

The device disconnect flow is as follows:

1. When the USB cable is unplugged or when the VBUS is switched off by the Host, the device core triggers GINTSTS.USBRst [bit 12] interrupt bit.
2. When the device application detects GINTSTS.USBRst, the application sets a time-out check for set address control transfer from host.
3. If application does not receive set address control transfer from host before the time-out period, it is treated as a device disconnection.

13.4.3.1 Device Soft Disconnection

The application can also perform a soft disconnect by setting the DCTL.SftDiscon bit.

Send/Receive USB Transfers -> Soft disconnect->Soft reset->USB Device Enumeration

Sequence of operations:

1. The application configures the device to send or receive transfers.
2. The application sets the Soft disconnect bit (SftDiscon) in the Device Control Register (DCTL).
3. The application sets the Soft Reset bit (CSftRst) in the Reset Register (GRSTCTL).
4. Poll the GRSTCTL register until the core clears the soft reset bit, which ensures the soft reset is completed properly.
5. Initialize the core according to the instructions in **“Device Initialization” on Page 13-11**.

Suspend-> Soft disconnect->Soft reset->USB Device Enumeration

Sequence of operations:

1. The core detects a USB suspend and generates a Suspend Detected interrupt.

2. The application sets the Stop PHY Clock bit (StopPclk) in the Power and Clock Gating Control register (PCGCCTL), the core asserts suspend_n to the PHY, and the PHY clock stops.
3. The application clears the StopPclk bit and waits for the PHY clock to come back. The core de-asserts suspend_n to the PHY, and the PHY clock comes back.
4. The application sets the Soft disconnect bit (SftDiscon) in Device Control Register (DCTL).
5. The application sets the Soft Reset bit (CSftRst) in the Reset Register (GRSTCTL).
6. Poll the GRSTCTL register until the core clears the soft reset bit, which ensures the soft reset is completed properly.
7. Initialize the core according to the instructions in **“Device Initialization” on Page 13-11**.

13.4.4 Endpoint Initialization

13.4.4.1 Initialization on USB Reset

1. Set the NAK bit for all OUT endpoints
   a) DOEPCTLx.SNAK = 1 (for all OUT endpoints)
2. Unmask the following interrupt bits:
   a) DAINTMSK.INEP0 = 1 (control 0 IN endpoint)
   b) DAINTMSK.OUTEPO = 1 (control 0 OUT endpoint)
   c) DOEPMSK.SETUP = 1
   d) DOEPMSK.XferCompl = 1
   e) DIEPMSK.XferCompl = 1
   f) DIEPMSK.TimeOut = 1
3. To transmit or receive data, the device must initialize more registers as specified in **“Device Initialization” on Page 13-11**
4. Set up the Data FIFO RAM for each of the FIFOs
   a) Program the GRXFSIZ Register, to be able to receive control OUT data and setup data. At a minimum, this must be equal to 1 max packet size of control endpoint 0 + 2 DWORDs (for the status of the control OUT data packet) + 10 DWORDs (for setup packets).
   b) Program the dedicated FIFO size register (depending on the FIFO number chosen) in Dedicated FIFO operation, to be able to transmit control IN data. At a minimum, this must be equal to 1 max packet size of control endpoint 0.
5. Reset the Device Address field in Device Configuration Register (DCFG).
6. (This step is not required if the Scatter/Gather DMA mode is used.) Program the following fields in the endpoint-specific registers for control OUT endpoint 0 to receive a SETUP packet

a) DOEPTSIZ0.SetUP Count = 3 (to receive up to 3 back-to-back SETUP packets)
b) In DMA mode, DOEPDMAO register with a memory address to store any SETUP packets received

At this point, all initialization required to receive SETUP packets is done, except for enabling control OUT endpoint 0 in DMA mode.

13.4.4.2 Initialization on Enumeration Completion

This section describes what the application must do when it detects an Enumeration Done interrupt.

1. On the Enumeration Done interrupt (GINTSTS.EnumDone, read the DSTS register to determine the enumeration speed.
2. Program the DIEPCTLO.MPS field to set the maximum packet size. This step configures control endpoint 0. The maximum packet size for a control endpoint depends on the enumeration speed.
3. In DMA mode, program the DOEPCTLO register to enable control OUT endpoint O, to receive a SETUP packet. In Scatter/Gather DMA mode, the descriptors must be set up in memory before enabling the endpoint.
   a) DOEPCTLO. EPEna = 1
4. Unmask the SOF interrupt.

At this point, the device is ready to receive SOF packets and is configured to perform control transfers on control endpoint 0.

13.4.4.3 Initialization on SetAddress Command

This section describes what the application must do when it receives a SetAddress command in a SETUP packet.

1. Program the DCFG register with the device address received in the SetAddress command
2. Program the core to send out a status IN packet.

13.4.4.4 Initialization on SetConfiguration/SetInterface Command

This section describes what the application must do when it receives a SetConfiguration or SetInterface command in a SETUP packet.

1. When a SetConfiguration command is received, the application must program the endpoint registers to configure them with the characteristics of the valid endpoints in the new configuration.
2. When a SetInterface command is received, the application must program the endpoint registers of the endpoints affected by this command.

3. Some endpoints that were active in the prior configuration or alternate setting are not valid in the new configuration or alternate setting. These invalid endpoints must be deactivated.
4. For details on a particular endpoint's activation or deactivation, see **“Endpoint Activation” on Page 13-15** and **“Endpoint Deactivation” on Page 13-15**.
5. Unmask the interrupt for each active endpoint and mask the interrupts for all inactive endpoints in the DAINTMSK register.
6. Set up the Data FIFO RAM for each FIFO. See **“Data FIFO RAM Allocation” on Page 13-154** for more detail.
7. After all required endpoints are configured, the application must program the core to send a status IN packet.

At this point, the device core is configured to receive and transmit any type of data packet.

13.4.4.5 Endpoint Activation

This section describes the steps required to activate a device endpoint or to configure an existing device endpoint to a new type.

1. Program the characteristics of the required endpoint into the following fields of the DIEPCTLx register (for IN or bidirectional endpoints) or the DOEPCTLx register (for OUT or bidirectional endpoints).
   a) Maximum Packet Size
   b) USB Active Endpoint = 1
   c) Set Endpoint Data Toggle bit to 0 (for interrupt and bulk endpoints)
   d) Endpoint Type
   e) TxFIFO Number
2. Once the endpoint is activated, the core starts decoding the tokens addressed to that endpoint and sends out a valid handshake for each valid token received for the endpoint.

13.4.4.6 Endpoint Deactivation

This section describes the steps required to deactivate an existing endpoint.

Before an endpoint can be de-activated, any pending transfers must first be stopped. For more information on stopping transfers, see **“Transfer Stop Programming for OUT Endpoints” on Page 13-17** or **“Transfer Stop Programming for IN Endpoints” on Page 13-21**.

1. In the endpoint to be deactivated, clear the USB Active Endpoint bit in the DIEPCTLx register (for IN or bidirectional endpoints) or the DOEPCTLx register (for OUT or bidirectional endpoints).
2. Once the endpoint is deactivated, the core ignores tokens addressed to that endpoint, resulting in a timeout on the USB.

13.4.5 Programming OUT Endpoint Features

13.4.5.1 Disabling an OUT Endpoint

The application must use this sequence to disable an OUT endpoint that it has enabled.

Application Programming Sequence

1. Before disabling any OUT endpoint, the application must enable Global OUT NAK mode in the core, as described in **"Setting the Global OUT NAK" on Page 13-17**.
   a) DCTL.DCTL.SGOUTNak = 1_B
2. Wait for the GINTSTS.GOUTNakEff interrupt
3. Disable the required OUT endpoint by programming the following fields.
   a) DOEPCTLx.EPDisable = 1_B
   b) DOEPCTLx.SNAK = 1_B
4. Wait for the DOEPINTx.EPDisabled interrupt, which indicates that the OUT endpoint is completely disabled. When the EPDisabled interrupt is asserted, the core also clears the following bits.
   a) DOEPCTLx.EPDisable = 0_B
   b) DOEPCTLx.EPEnable = 0_B
5. The application must clear the Global OUT NAK bit to start receiving data from other non-disabled OUT endpoints.
   a) DCTL.SGOUTNak = 0_B

13.4.5.2 Stalling a Non-Isochronous OUT Endpoint

This section describes how the application can stall a non-isochronous endpoint.

1. Put the core in the Global OUT NAK mode, as described in **"Setting the Global OUT NAK" on Page 13-17**.
2. Disable the required endpoint, as described in **"Disabling an OUT Endpoint" on Page 13-16**.
   a) When disabling the endpoint, instead of setting the DOEPCTL.SNAK bit, set DOEPCTL.STALL = 1.
   b) The Stall bit always takes precedence over the NAK bit.
3. When the application is ready to end the STALL handshake for the endpoint, the DOEPCTLx.STALL bit must be cleared.

If the application is setting or clearing a STALL for an endpoint due to a SetFeature.Endpoint Halt or ClearFeature.Endpoint Halt command, the Stall bit must be set or cleared before the application sets up the Status stage transfer on the control endpoint.

13.4.5.3 Setting the Global OUT NAK

Internal Data Flow

1. When the application sets the Global OUT NAK (DCTL.SGOUTNak), the core stops writing data, except SETUP packets, to the receive FIFO. Irrespective of the space availability in the receive FIFO, non-isochronous OUT tokens receive a NAK handshake response, and the core ignores isochronous OUT data packets
2. The core writes the Global OUT NAK pattern to the receive FIFO. The application must reserve enough receive FIFO space to write this data pattern. See **"Data FIFO RAM Allocation" on Page 13-154**.
3. When either the core (in DMA mode) or the application (in Slave mode) pops the Global OUT NAK pattern DWORD from the receive FIFO, the core sets the GINTSTS.GOUTNakEff interrupt.
4. Once the application detects this interrupt, it can assume that the core is in Global OUT NAK mode. The application can clear this interrupt by clearing the DCTL.SGOUTNak bit.

Application Programming Sequence

1. To stop receiving any kind of data in the receive FIFO, the application must set the Global OUT NAK bit by programming the following field.
   a) DCTL.SGOUTNak = 1_B
2. Wait for the assertion of the interrupt GINTSTS.GOUTNakEff. When asserted, this interrupt indicates that the core has stopped receiving any type of data except SETUP packets.
3. The application can receive valid OUT packets after it has set DCTL.SGOUTNak and before the core asserts the GINTSTS.GOUTNakEff interrupt.
4. The application can temporarily mask this interrupt by writing to the GINTMSK.GINNakEffMsk bit.
   a) GINTMSK.GINNakEffMsk = 0_B
5. Whenever the application is ready to exit the Global OUT NAK mode, it must clear the DCTL.SGOUTNak bit. This also clears the GINTSTS.GOUTNakEff interrupt.
   a) DCTL.CGOUTNak = 1_B
6. If the application has masked this interrupt earlier, it must be unmasked as follows:
   a) GINTMSK.GINNakEffMsk = 1_B

13.4.5.4 Transfer Stop Programming for OUT Endpoints

When the core is operating as a device, the following programing sequence can be used to stop any transfers (because of an interrupt from the host, typically a reset).

Note: The RxFIFO is common for OUT endpoints, therefore there is only one transfer stop programming flow for OUT endpoints.

Sequence of operations:

1. Enable all OUT endpoints by setting DOEPCTL.EPEna = 1_B.
2. Before disabling any OUT endpoint, the application must enable Global OUT NAK mode in the core, according to the instructions in **"Setting the Global OUT NAK" on Page 13-17**. This ensures that data in the RX FIFO is sent to the application successfully. Set DCTL.DCTL.SGOUTNak = 1_B.
3. Wait for the GINTSTS.GOUTNakEff interrupt.
4. Disable all active OUT endpoints by programming the following register bits:
   a) DOEPCTL.EPEna = 1_B
   b) DOEPCTLn.EPDisable = 1_B
   c) DOEPCTLn.SNAK = 1_B
5. Wait for the DOEPINTn.EPDisabled interrupt for each OUT endpoint programmed in the previous step. The DOEPINTn.EPDisabled interrupt indicates that the corresponding OUT endpoint is completely disabled. When the EPDisabled interrupt is asserted, the DWC_otg core clears the following bits:
   a) DOEPCTL.EPEna = 0_B
   b) DOEPCTLn.EPDisable = 0_B
   c) DOEPCTLn.EPEnable = 0_B

Note: The application must not flush the RxFIFO, as the Global out NAK effective interrupt earlier ensures that there is no data left in the RxFIFO.

13.4.6 Programming IN Endpoint Features

13.4.6.1 Setting IN Endpoint NAK

Internal Data Flow

1. When the application sets the IN NAK for a particular endpoint, the core stops transmitting data on the endpoint, irrespective of data availability in the endpoint's transmit FIFO.
2. Non-isochronous IN tokens receive a NAK handshake reply
   a) Isochronous IN tokens receive a zero-data-length packet reply
3. The core asserts the DIEPINTx.IN NAK Effective interrupt in response to the DIEPCTL.Set NAK bit.
4. Once this interrupt is seen by the application, the application can assume that the endpoint is in IN NAK mode. This interrupt can be cleared by the application by setting the DIEPCTLx. Clear NAK bit.

Application Programming Sequence

1. To stop transmitting any data on a particular IN endpoint, the application must set the IN NAK bit. To set this bit, the following field must be programmed.
   a) DIEPCTLx.SetNAK = 1_B
2. Wait for assertion of the DIEPINTx.NAK Effective interrupt. This interrupt indicates the core has stopped transmitting data on the endpoint.
3. The core can transmit valid IN data on the endpoint after the application has set the NAK bit, but before the assertion of the NAK Effective interrupt.
4. The application can mask this interrupt temporarily by writing to the DIEPMSK.NAK Effective bit.
   a) DIEPMSK.NAK Effective = 0_B
5. To exit Endpoint NAK mode, the application must clear the DIEPCTLx.NAK status. This also clears the DIEPINTx.NAK Effective interrupt.
   a) DIEPCTLx.ClearNAK = 1_B
6. If the application masked this interrupt earlier, it must be unmasked as follows:
   a) DIEPMSK.NAK Effective = 1_B

13.4.6.2 IN Endpoint Disable

Use the following sequence to disable a specific IN endpoint (periodic/non-periodic) that has been previously enabled in dedicated FIFO operation.

Application Programming Sequence:

1. In Slave mode, the application must stop writing data on the AHB, for the IN endpoint to be disabled.
2. The application must set the endpoint in NAK mode. See **“Setting IN Endpoint NAK” on Page 13-18**.
   a) DIEPCTLx.SetNAK = 1_B
3. Wait for DIEPINTx.NAK Effective interrupt.
4. Set the following bits in the DIEPCTLx register for the endpoint that must be disabled.
   a) DIEPCTLx.Endpoint Disable = 1
   b) DIEPCTLx.SetNAK = 1
5. Assertion of DIEPINTx.Endpoint Disabled interrupt indicates that the core has completely disabled the specified endpoint. Along with the assertion of the interrupt, the core also clears the following bits.
   a) DIEPCTLx.EPEnable = 0_B
   b) DIEPCTLx.EPDisable = 0_B
6. The application must read the DIEPTSIZx register for the periodic IN EP, to calculate how much data on the endpoint was transmitted on the USB.
7. The application must flush the data in the Endpoint transmit FIFO, by setting the following fields in the GRSTCTL register.
   a) GRSTCTL.TxFIFONum = Endpoint Transmit FIFO Number

b) GRSTCTL.TxFFlush = 1

The application must poll the GRSTCTL register, until the TxFFlush bit is cleared by the core, which indicates the end of flush operation. To transmit new data on this endpoint, the application can re-enable the endpoint at a later point.

13.4.6.3 Timeout for Control IN Endpoints

The application must treat the TIMEOUT interrupt received for the last IN transaction of a Control Transfers Data Stage separately. This is done to take into account the TIMEOUT due to lost ACK case (core did not receive the ACK send by the host). Application must unmask timeout interrupt for control IN transfers Data phase only. On getting the timeout interrupt for control endpoint data phase, the application must also enable the OUT control endpoint for the status phase.

If the timeout is due to a lost ACK, the host switches to the Data stage, and the application receives Transfer Complete interrupt for the OUT endpoint. The application can then flush the IN packet and disable both the IN and OUT endpoints. If the timeout was due to lost data, the host sends the IN token again, and the application receives a Transfer Complete interrupt for the IN endpoint. The application can thus keep the OUT endpoint enabled for the status phase.

13.4.6.4 Stalling Non-Isochronous IN Endpoints

This section describes how the application can stall a non-isochronous endpoint.

Application Programming Sequence

1. Disable the IN endpoint to be stalled. See **“IN Endpoint Disable” on Page 13-19** for more details. Set the Stall bit as well.
2. DIEPCTLx.Endpoint Disable = 1, when the endpoint is already enabled
   a) DIEPCTLx.STALL = 1
   b) The Stall bit always takes precedence over the NAK bit
3. Assertion of the DIEPINTx.Endpoint Disabled interrupt indicates to the application that the core has disabled the specified endpoint.
4. The application must flush the Non-periodic or Periodic Transmit FIFO, depending on the endpoint type. In case of a non-periodic endpoint, the application must re-enable the other non-periodic endpoints, which do not need to be stalled, to transmit data.
5. Whenever the application is ready to end the STALL handshake for the endpoint, the DIEPCTLx.STALL bit must be cleared.
6. If the application sets or clears a STALL for an endpoint due to a SetFeature.Endpoint Halt command or ClearFeature.Endpoint Halt command, the Stall bit must be set or cleared before the application sets up the Status stage transfer on the control endpoint.

Special Case: Stalling the Control IN/OUT Endpoint

The core must stall IN/OUT tokens if, during the Data stage of a control transfer, the host sends more IN/OUT tokens than are specified in the SETUP packet. In this case, the application must to enable DIEPINTx.INTknTXFEmp and DOEPINTx.OUTTknEPdis interrupts during the Data stage of the control transfer, after the core has transferred the amount of data specified in the SETUP packet. Then, when the application receives this interrupt, it must set the STALL bit in the corresponding endpoint control register, and clear this interrupt.

13.4.6.5 Transfer Stop Programming for IN Endpoints

When the core is operating as a device, the following programing sequence can be used to stop any transfers (because of an interrupt from the host, typically a reset).

Sequence of operations:

1. Disable the IN endpoint by programming DIEPCTLn.EPDis = 1_B.
2. Wait for the DIEPINTn.EPDisabled interrupt, which indicates that the IN endpoint is completely disabled. When the EPDisabled interrupt is asserted, the core clears the following bits:
   a) DIEPCTL.EPDisable = 0_B
   b) DIEPCTL.EPEnable = 0_B
3. Flush the TX FIFO by programming the following bits:
   a) GRSTCTL.TxFFlsh = 1_B
   b) GRSTCTL.TxFNum = <FIFO number specific to endpoint>
4. The application can start polling till GRSTCTL.TXFFlsh is cleared. When this bit is cleared, it ensures that there is no data left in the TX FIFO.

13.4.6.6 Non-Periodic IN Endpoint Sequencing

In DMA mode, the DIEPCTLx.NextEp value programmed controls the order in which the core fetches non- periodic data for IN endpoints.

If application requires the core to fetch data for the non-periodic IN endpoints in a certain endpoint order, it must program the DIEPCTLx.NextEP field accordingly before enabling the endpoints. To enable a single endpoint enabled at a time the application must set the DIEPCTLx.NextEP field to the endpoint number itself. The core uses the NextEP field irrespective of the DIEPCTLx.EPEna bit.

13.4.7 Worst-Case Response Time

When the USB core acts as a device, there is a worst case response time for any tokens that follow an isochronous OUT. This worst case response time depends on the AHB clock frequency.

The core registers are in the AHB domain, and the core does not accept another token before updating these register values. The worst case is for any token following an isochronous OUT, because for an isochronous transaction, there is no handshake and the next token could come sooner. This worst case value is 7 PHY clocks when the AHB clock is the same as the PHY clock. When AHB clock is faster, this value is smaller.

If this worst case condition occurs, the core responds to bulk/ interrupt tokens with a NAK and drops isochronous and SETUP tokens. The host interprets this as a timeout condition for SETUP and retries the SETUP packet. For isochronous transfers, the incomplISOCIN and incomplISOCOUT interrupts inform the application that isochronous IN/OUT packets were dropped.

13.4.8 Choosing the Value of GUSBCFG.USBTrdTim

The value in GUSBCFG.USBTrdTim is the time it takes for the Media Access Controller (MAC), in terms of PHY clocks after it has received an IN token, to get the FIFO status, and thus the first data. The MAC is the part of the USB core that handles USB transactions and protocols. This time involves the synchronization delay between the PHY and AHB clocks. The worst case delay for this is when the AHB clock is the same as the PHY clock. In this case, the delay is 5 clocks. If the PHY clock is running at 60 MHz and the AHB is running at 30 MHz, this value is 9 clocks.

If the AHB is running at a higher frequency than the PHY, the application can use a smaller value for GUSBCFG.USBTrdTim.

The application can use the following formula to calculate the value of GUSBCFG.USBTrdTim:

4 * AHB Clock + 1 PHY Clock
= (2 clock sync + 1 clock memory address + 1 clock memory data from sync RAM) + (1 PHY Clock (next PHY clock MAC can sample the 2-clock FIFO output)

13.4.9 Handling Babble Conditions

If USB core receives a packet that is larger than the maximum packet size for that endpoint, the core stops writing data to the Rx buffer and waits for the end of packet (EOP). When the core detects the EOP, it flushes the packet in the Rx buffer and does not send any response to the host.

If the core continues to receive data at the EOF2 (the end of frame 2, which is very close to SOF), the core generates an early_suspend interrupt (GINTSTS.ErlySusp). On receiving this interrupt, the application must check the erratic_error status bit (DSTS.ErrticErr). If this bit is set, the application must take it as a long babble and perform a soft reset.

13.4.10 Device Programming Operations in Buffer DMA or Slave Mode

Table 13-2 provides links to the programming sequence for different USB transaction types when the core is in Slave or Buffer DMA mode of operation.

For information on device programming operations when in Scatter/Gather DMA mode, see Section 13.7.

Table 13-2 Device Programming Operations

Device Mode	IN	SETUP	OUT
Control			
Slave	"Non-Periodic (Bulk and Control) IN Data Transfers" on Page 13-36	"OUT Data Transfers" on Page 13-32	"Non-Isochronous OUT Data Transfers" on Page 13-42
DMA	"Non-Periodic (Bulk and Control) IN Data Transfers" on Page 13-70	"OUT Data Transfers" on Page 13-67	"Non-Isochronous OUT Data Transfers" on Page 13-72
Bulk			
Slave	"Non-Periodic (Bulk and Control) IN Data Transfers" on Page 13-36	-	"Non-Isochronous OUT Data Transfers" on Page 13-42
DMA	"Non-Periodic (Bulk and Control) IN Data Transfers" on Page 13-70	-	"Non-Isochronous OUT Data Transfers" on Page 13-72

Table 13-2 Device Programming Operations (cont'd)

Device Mode	IN	SETUP	OUT
Interrupt			
Slave	**"Periodic IN (Interrupt and Isochronous) Data Transfers" on Page 13-55** **"Periodic IN Data Transfers Using the Periodic Transfer Interrupt" on Page 13-57**	-	**"Non-Isochronous OUT Data Transfers" on Page 13-42** **"Interrupt OUT Data Transfers Using Periodic Transfer Interrupt" on Page 13-63**
DMA	**"Periodic IN (Interrupt and Isochronous) Data Transfers" on Page 13-76** **"Periodic IN Data Transfers Using the Periodic Transfer Interrupt" on Page 13-78**	-	**"Non-Isochronous OUT Data Transfers" on Page 13-72** **"Interrupt OUT Data Transfers Using Periodic Transfer Interrupt" on Page 13-84**

Table 13-2 Device Programming Operations (cont'd)

Device Mode	IN	SETUP	OUT
Isochronous			
Slave	**"Periodic IN (Interrupt and Isochronous) Data Transfers" on Page 13-55** **"Periodic IN Data Transfers Using the Periodic Transfer Interrupt" on Page 13-57**	-	**"Control Read Transfers (SETUP, Data IN, Status OUT)" on Page 13-26** **"Incomplete Isochronous OUT Data Transfers" on Page 13-52**
DMA	**"Periodic IN (Interrupt and Isochronous) Data Transfers" on Page 13-76** **"Periodic IN Data Transfers Using the Periodic Transfer Interrupt" on Page 13-78**	-	**"Control Read Transfers (SETUP, Data IN, Status OUT)" on Page 13-66** **"Incomplete Isochronous OUT Data Transfers" on Page 13-74**

13.5 Device Programming in Slave Mode

This section discusses how to program the core when it is acting as a Device in the Slave mode of operation.

13.5.1 Control Transfers

This section describes the various types of control transfers.

13.5.1.1 Control Write Transfers (SETUP, Data OUT, Status IN)

This section describes control write transfers.

Application Programming Sequence

1. Assertion of the DOEPINTx.SETUP Packet interrupt indicates that a valid SETUP packet has been transferred to the application. See **"OUT Data Transfers" on**

Page 13-32 for more details. At the end of the Setup stage, the application must reprogram the DOEPTSIZx.SUPCnt field to 3 to receive the next SETUP packet.
2. If the last SETUP packet received before the assertion of the SETUP interrupt indicates a data OUT phase, program the core to perform a control OUT transfer as explained in **"Non-Isochronous OUT Data Transfers" on Page 13-42**.
3. In a single OUT data transfer on control endpoint 0, the application can receive up to 64 bytes. If the application is expecting more than 64 bytes in the Data OUT stage, the application must re-enable the endpoint to receive another 64 bytes, and must continue to do so until it has received all the data in the Data stage.
4. Assertion of the DOEPINTx.Transfer Compl interrupt on the last data OUT transfer indicates the completion of the data OUT phase of the control transfer.
5. On completion of the data OUT phase, the application must do the following.
   a) To transfer a new SETUP packet in DMA mode, the application must re-enable the control OUT endpoint as explained in section **"OUT Data Transfers" on Page 13-32**.
      - DOEPCTLx.EPEna = 1_B
   b) To execute the received Setup command, the application must program the required registers in the core. This step is optional, based on the type of Setup command received.
6. For the status IN phase, the application must program the core as described in **"Non-Periodic (Bulk and Control) IN Data Transfers" on Page 13-36** to perform a data IN transfer.
7. Assertion of the DIEPINTx.Transfer Compl interrupt indicates completion of the status IN phase of the control transfer.

13.5.1.2 Control Read Transfers (SETUP, Data IN, Status OUT)

This section describes control write transfers.

Application Programming Sequence

1. Assertion of the DOEPINTx.SETUP Packet interrupt indicates that a valid SETUP packet has been transferred to the application. See "**"OUT Data Transfers" on Page 13-32** for more details. At the end of the Setup stage, the application must reprogram the DOEPTSIZx.SUPCnt field to 3 to receive the next SETUP packet.
2. If the last SETUP packet received before the assertion of the SETUP interrupt indicates a data IN phase, program the core to perform a control IN transfer as explained in **"Non-Periodic (Bulk and Control) IN Data Transfers" on Page 13-36**.
3. On a single IN data transfer on control endpoint 0, the application can transmit up to 64 bytes. To transmit more than 64 bytes in the Data IN stage, the application must re-enable the endpoint to transmit another 64 bytes, and must continue to do so, until it has transmitted all the data in the Data stage.

4. The DIEPINTx.Transfer Compl interrupt on the last IN data transfer marks the completion of the control transfer's Data stage.
5. To perform a data OUT transfer in the status OUT phase, the application must program the core as described in "**“OUT Data Transfers” on Page 13-32**. The application must program the DCFG.NZStsOUTHShk handshake field to a proper setting before transmitting an data OUT transfer for the Status stage.
6. Assertion of the DOEPINTx.Transfer Compl interrupt indicates completion of the status OUT phase of the control transfer. This marks the successful completion of the control read transfer.

13.5.1.3 Two-Stage Control Transfers (SETUP/Status IN)

This section describes two-stage control transfers.

Application Programming Sequence

1. Assertion of the DOEPINTx.SetUp interrupt indicates that a valid SETUP packet has been transferred to the application. See "**“OUT Data Transfers” on Page 13-32** for more detail. To receive the next SETUP packet, the application must reprogram the DOEPTSIZx.SUPCnt field to 3 at the end of the Setup stage.
2. Decode the last SETUP packet received before the assertion of the SETUP interrupt. If the packet indicates a two-stage control command, the application must do the following.
   a) Set DOEPCTLx.EPEna = 1_B
   b) Depending on the type of Setup command received, the application can be required to program registers in the core to execute the received Setup command.
3. For the status IN phase, the application must program the core described in **“Non-Periodic (Bulk and Control) IN Data Transfers” on Page 13-36** to perform a data IN transfer.
4. Assertion of the DIEPINTx.Transfer Compl interrupt indicates the completion of the status IN phase of the control transfer.

Example: Two-Stage Control Transfer

These notes refer to **Figure 13-5**.

1. SETUP packet #1 is received on the USB and is written to the receive FIFO, and the core responds with an ACK handshake. This handshake is lost and the host detects a timeout.
2. The SETUP packet in the receive FIFO results in a GINTSTS.RxFLvl interrupt to the application, causing the application to empty the receive FIFO.
3. SETUP packet #2 on the USB is written to the receive FIFO, and the core responds with an ACK handshake.
4. The SETUP packet in the receive FIFO sends the application the GINTSTS.RxFLvl interrupt and the application empties the receive FIFO.

5. After the second SETUP packet, the host sends a control IN token for the status phase. The core issues a NAK response to this token, and writes a Setup Stage Done entry to the receive FIFO. This entry results in a GINTSTS.RxFLvl interrupt to the application, which empties the receive FIFO. After reading out the Setup Stage Done DWORD, the core asserts the DOEPINTx.SetUp packet interrupt to the application.
6. On this interrupt, the application processes SETUP Packet #2, decodes it to be a two-stage control command, and clears the control IN NAK bit.
   a) DIEPCTLx.CNAK = 1
7. When the application clears the IN NAK bit, the core interrupts the application with a DIEPINTx.INTknTXFEmp. On this interrupt, the application enables the control IN endpoint with a DIEPTSIZx.XferSize of 0 and a DIEPTSIZx.PktCnt of 1. This results in a zero-length data packet for the status IN token on the USB.
8. At the end of the status IN phase, the core interrupts the application with a DIEPINTx.XferCompl interrupt.

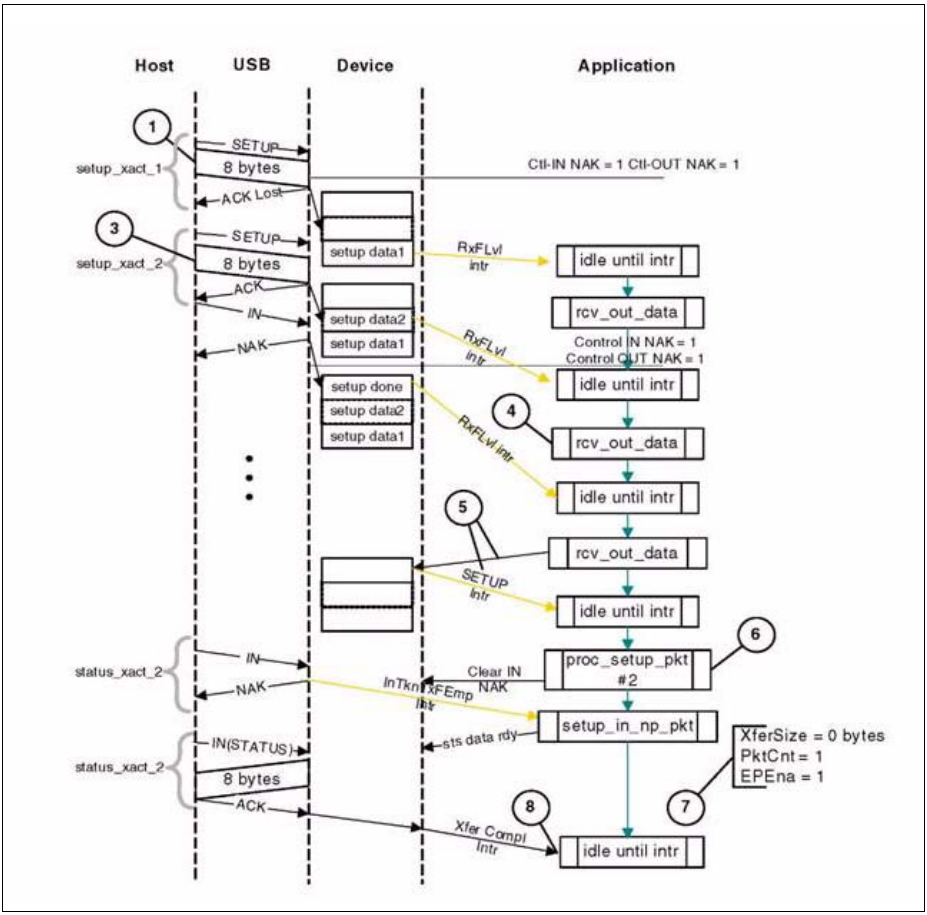


Figure 13-5 Two-Stage Control Transfer

13.5.1.4 Packet Read from FIFO

This section describes how to read packets (OUT data and SETUP packets) from the receive FIFO in Slave mode.

1. On catching a GINTSTS.RxFLvl interrupt, the application must read the Receive Status Pop register (GRXSTSP).
2. The application can mask the GINTSTS.RxFLvl interrupt by writing to GINTMSK.RxFLvl = 0_B, until it has read the packet from the receive FIFO.

3. If the received packet's byte count is not 0, the byte count amount of data is popped from the receive Data FIFO and stored in memory. If the received packet byte count is 0, no data is popped from the Receive Data FIFO.
4. The receive FIFO's packet status readout indicates one of the following.
5. Global OUT NAK Pattern: PktSts = Global OUT NAK, BCnt = 11'h000, EPNum = Dont Care (4'h0), DPID = Dont Care (00_B). This data indicates that the global OUT NAK bit has taken effect.
   a) SETUP Packet Pattern: PktSts = SETUP, BCnt = 11'h008, EPNum = Control EP Num, DPID = D0. This data indicates that a SETUP packet for the specified endpoint is now available for reading from the receive FIFO.
   b) Setup Stage Done Pattern: PktSts = Setup Stage Done, BCnt = 11'h0, EPNum = Control EP Num, DPID = Don't Care (00_B). This data indicates that the Setup stage for the specified endpoint has completed and the Data stage has started. After this entry is popped from the receive FIFO, the core asserts a Setup interrupt on the specified control OUT endpoint.
   c) Data OUT Packet Pattern: PktSts = DataOUT, BCnt = size of the Received data OUT packet (0 < BCnt <1,024), EPNum = EPNum on which the packet was received, DPID = Actual Data PID.
   d) Data Transfer Completed Pattern: PktSts = Data OUT Transfer Done, BCnt = 11'h0, EPNum = OUT EP Num on which the data transfer is complete, DPID = Dont Care (00_B). This data indicates that a OUT data transfer for the specified OUT endpoint has completed. After this entry is popped from the receive FIFO, the core asserts a Transfer Completed interrupt on the specified OUT endpoint.
   The encoding for the PktSts is listed in **“Receive Status Debug Read/Status Read and Pop Registers (GRXSTSR/GRXSTSP)” on Page 13-180**.
6. After the data payload is popped from the receive FIFO, the GINTSTS.RxFLvl interrupt must be unmasked.
7. Steps 1-5 are repeated every time the application detects assertion of the interrupt line due to GINTSTS.RxFLvl. Reading an empty receive FIFO can result in undefined core behavior.

Figure 13-6 provides a flow chart of this procedure.

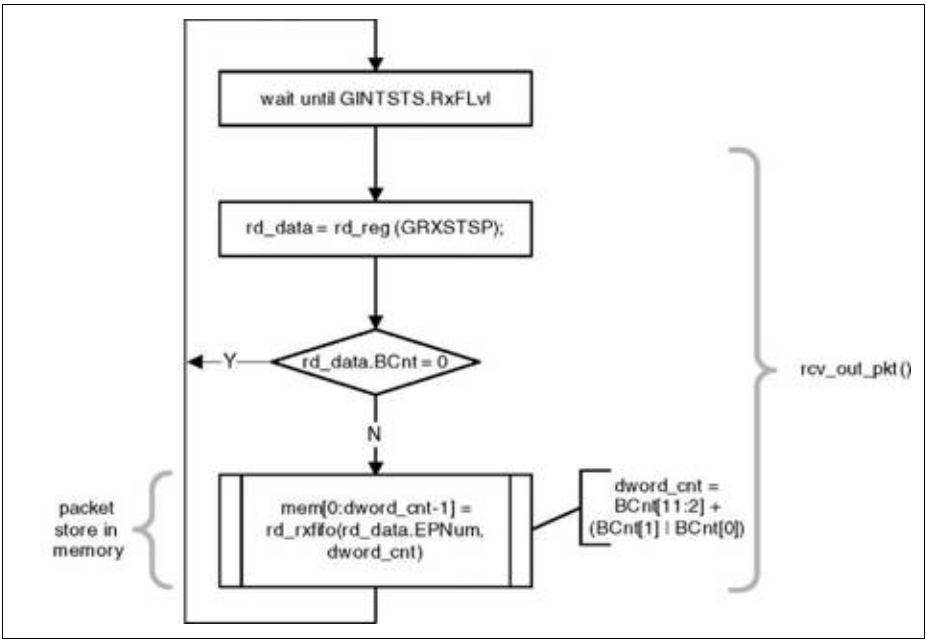


Figure 13-6 Receive FIFO Packet Read in Slave Mode

13.5.2 IN Data Transfers

This section describes the internal data flow and application-level operations during IN data transfers.

13.5.2.1 Packet Write

This section describes how the application writes data packets to the endpoint FIFO in Slave mode with dedicated transmit FIFOs.

1. The application can either choose polling or interrupt mode.
   a) In polling mode, application monitors the status of the endpoint transmit data FIFO, by reading the DTXFSTSx register, to determine, if there is enough space in the data FIFO.
   b) In interrupt mode, application waits for the DIEPINTx.TxFEmp interrupt and then reads the DTXFSTSx register, to determine, if there is enough space in the data FIFO.
   c) To write a single non-zero length data packet, there must be space to write the entire packet is the data FIFO.

d) For writing zero length packet, application must not look for FIFO space.

2. Using one of the above mentioned methods, when the application determines that there is enough space to write a transmit packet, the application must first write into the endpoint control register, before writing the data into the data FIFO. The application, typically must do a read modify write on the DIEPCTLx, to avoid modifying the contents of the register, except for setting the Endpoint Enable bit.

The application can write multiple packets for the same endpoint, into the transmit FIFO, if space is available. For periodic IN endpoints, application must write packets only for one frame. It can write packets for the next periodic transaction, only after getting transfer complete for the previous transaction.

13.5.3 OUT Data Transfers

This section describes the internal data flow and application-level operations during data OUT transfers and setup transactions.

13.5.3.1 Control Setup Transactions

This section describes how the core handles SETUP packets and the application's sequence for handling setup transactions. To initialize the core after power-on reset, the application must follow the sequence in **“Core Initialization” on Page 13-9**. Before it can communicate with the host, it must initialize an endpoint as described in **“Endpoint Initialization” on Page 13-13**. See **“Packet Read from FIFO” on Page 13-29**.

Application Requirements

1. To receive a SETUP packet, the DOEPTSIZx.SUPCnt field in a control OUT endpoint must be programmed to a non-zero value. When the application programs the SUPCnt field to a non-zero value, the core receives SETUP packets and writes them to the receive FIFO, irrespective of the DOEPCTLx.NAK status and DOEPCTLx.EPEna bit setting. The SUPCnt field is decremented every time the control endpoint receives a SETUP packet. If the SUPCnt field is not programmed to a proper value before receiving a SETUP packet, the core still receives the SETUP packet and decrements the SUPCnt field, but the application possibly is not be able to determine the correct number of SETUP packets received in the Setup stage of a control transfer.
   a) DOEPTSIZx.SUPCnt = 3
2. The application must always allocate some extra space in the Receive Data FIFO, to be able to receive up to three SETUP packets on a control endpoint.
   a) The space to be Reserved is 10 DWORDs. Three DWORDs are required for the first SETUP packet, 1 DWORD is required for the Setup Stage Done DWORD, and 6 DWORDs are required to store two extra SETUP packets among all control endpoints.

b) 3 DWORDs per SETUP packet are required to store 8 bytes of SETUP data and 4 bytes of SETUP status (Setup Packet Pattern). The core reserves this space in the receive data
c) FIFO to write SETUP data only, and never uses this space for data packets.

3. In Slave mode, the application must read the 2 DWORDs of the SETUP packet from the receive FIFO.
4. The application must read and discard the Setup Stage Done DWORD from the receive FIFO.

Internal Data Flow

1. When a SETUP packet is received, the core writes the received data to the receive FIFO, without checking for available space in the receive FIFO and irrespective of the endpoint's NAK and Stall bit settings.
   a) The core internally sets the IN NAK and OUT NAK bits for the control IN/OUT endpoints on which the SETUP packet was received.
2. For every SETUP packet received on the USB, 3 DWORDs of data is written to the receive FIFO, and the SUPCnt field is decremented by 1.
   a) The first DWORD contains control information used internally by the core
   b) The second DWORD contains the first 4 bytes of the SETUP command
   c) The third DWORD contains the last 4 bytes of the SETUP command
3. When the Setup stage changes to a Data IN/OUT stage, the core writes an entry (Setup Stage Done DWORD) to the receive FIFO, indicating the completion of the Setup stage.
4. On the AHB side, the application empties the SETUP packets.
5. When the application pops the Setup Stage Done DWORD from the receive FIFO, the core interrupts the application with a DOEPINTx.SETUP interrupt, indicating it can process the received SETUP packet.
6. The core clears the endpoint enable bit for control OUT endpoints.

Application Programming Sequence

1. Program the DOEPTSIZx register.
   a) DOEPTSIZx.SUPCnt = 3
2. Wait for the GINTSTS.RxFLvl interrupt and empty the data packets from the receive FIFO, as explained in **“Packet Read from FIFO” on Page 13-29**. This step can be repeated many times.
3. Assertion of the DOEPINTx.SETUP interrupt marks a successful completion of the SETUP Data Transfer. On this interrupt, the application must read the DOEPTSIZx register to determine the number of SETUP packets received and process the last received SETUP packet.

Note: If the application has not enabled EP0 before the host sends the SETUP packet, the core ACKs the SETUP packet and stores it in the FIFO, but does not write to the memory until EP0 is enabled. When the application enables the EP0 (first

enable) and clears the NAK bit at the same time the Host sends DATA OUT, the DATA OUT is stored in the RxFIFO. The USB core then writes the setup data to the memory and disables the endpoint. Though the application expects a Transfer Complete interrupt for the Data OUT phase, this does not occur, because the SETUP packet, rather than the DATA OUT packet, enables EP0 the first time. Thus, the DATA OUT packet is still in the RxFIFO until the application re-enables EP0. The application must enable EP0 one more time for the core to process the DATA OUT packet.

Figure 13-7 charts this flow.

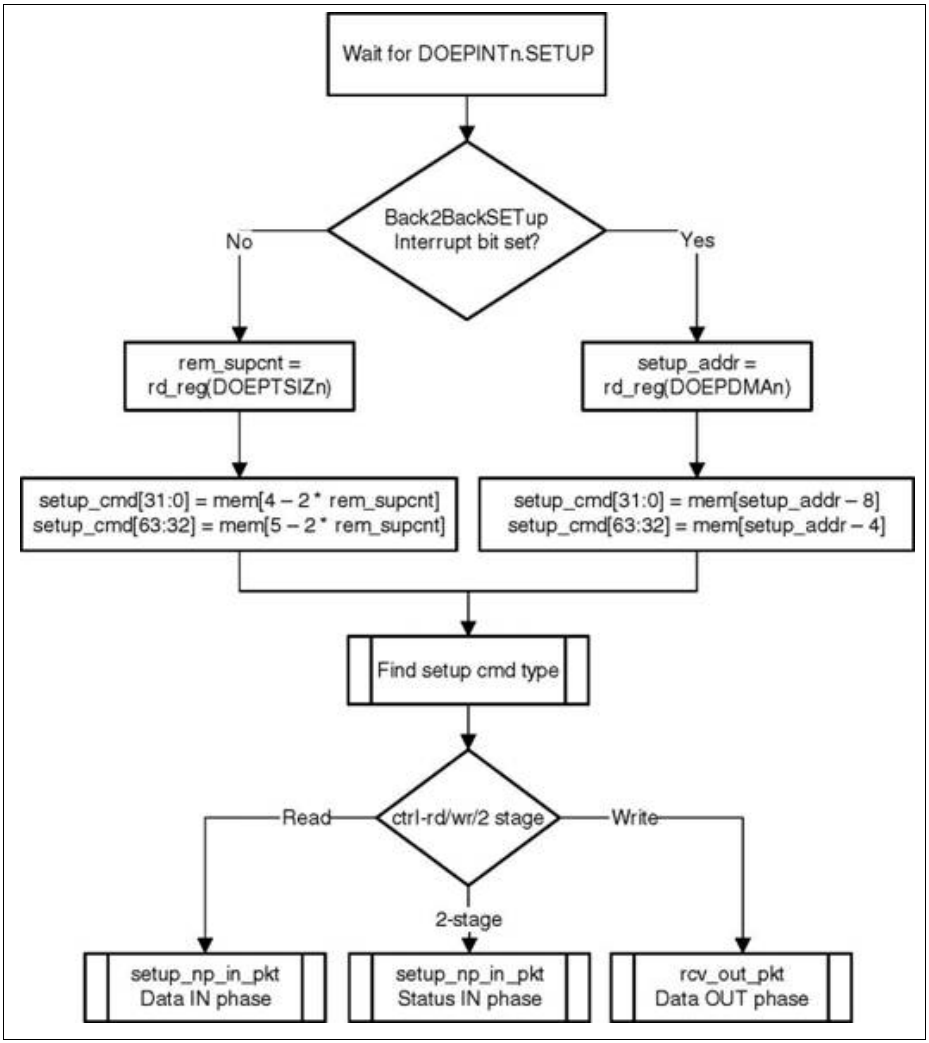


Figure 13-7 Processing a SETUP Packet

13.5.3.2 Handling More Than Three Back-to-Back SETUP Packets

According to the USB 2.0 specification, normally, during a SETUP packet error, a host does not send more than three back-to-back SETUP packets to the same endpoint. However, the USB 2.0 specification does not limit the number of back-to-back SETUP

packets a host can send to the same endpoint. When this condition occurs, the USB core generates an interrupt (DOEPINTx.Back2BackSETup).

13.5.4 Non-Periodic (Bulk and Control) IN Data Transfers

This section describes a regular non-periodic IN data transfer.

Application Requirements

1. For IN transfers, the Transfer Size field in the Endpoint Transfer Size register denotes a payload that constitutes multiple maximum-packet-size packets and a single short packet. This short packet is transmitted at the end of the transfer.
   a) To transmit a few maximum-packet-size packets and a short packet at the end of the transfer:
   b) Transfer size[epnum] = n * mps[epnum] + sp
   (where n is an integer > 0, and 0 <sp < mps[epnum])
   - If (sp > 0), then packet count[epnum] = n + 1.
   Otherwise, packet count[epnum] = n
   c) To transmit a single zero-length data packet:
   - Transfer size[epnum] = 0
   - Packet count[epnum] = 1
   d) To transmit a few maximum-packet-size packets and a zero-length data packet at the end of the transfer, the application must split the transfer in two parts. The first sends maximum-packet- size data packets and the second sends the zero-length data packet alone.
   - First transfer: transfer size[epnum] = n * mps[epnum]; packet count = n;
   - Second transfer: transfer size[epnum] = 0; packet count = 1;
2. Once an endpoint is enabled for data transfers, the core updates the Transfer Size register. At the end of IN transfer, which ended with an Endpoint Disabled interrupt, the application must read the Transfer Size register to determine how much data posted in the transmit FIFO was already sent on the USB.
3. Data fetched into transmit FIFO = Application-programmed initial transfer size - core-updated final transfer size
   a) Data transmitted on USB = (application-programmed initial packet count - Core updated final packet count) * mps[epnum]
   b) Data yet to be transmitted on USB = (Application-programmed initial transfer size - data transmitted on USB)

Internal Data Flow

1. The application must set the Transfer Size and Packet Count fields in the endpoint-specific registers and enable the endpoint to transmit the data.
2. In Slave mode, the application must also write the required data to the transmit FIFO for the endpoint.

3. Every time the application writes a packet into the transmit FIFO, the transfer size for that endpoint is decremented by the packet size. The data is fetched from the memory, until the transfer size for the endpoint becomes 0. After writing the data into the FIFO, the "number of packets in FIFO" count is incremented (this is a 3-bit count, internally maintained by the core for each IN endpoint transmit FIFO. The maximum number of packets maintained by the core at any time in an IN endpoint FIFO is eight). For zero-length packets, a separate flag is set for each FIFO, without any data in the FIFO.
4. Once the data is written to the transmit FIFO, the core reads it out upon receiving an IN token. For every non-isochronous IN data packet transmitted with an ACK handshake, the packet count for the endpoint is decremented by one, until the packet count is zero. The packet count is not decremented on a TIMEOUT.
5. For zero length packets (indicated by an internal zero length flag), the core sends out a zero-length packet for the IN token and decrements the Packet Count field.
6. If there is no data in the FIFO for a received IN token and the packet count field for that endpoint is zero, the core generates a IN Tkn Rcvd When FIFO Empty Interrupt for the endpoint, provided the endpoint NAK bit is not set. The core responds with a NAK handshake for non-isochronous endpoints on the USB.
7. In Dedicated FIFO operation, the core internally rewinds the FIFO pointers and no timeout interrupt is generated except for Control IN endpoint.
8. When the transfer size is 0 and the packet count is 0, the transfer complete interrupt for the endpoint is generated and the endpoint enable is cleared.

Application Programming Sequence

1. Program the DIEPTSIZx register with the transfer size and corresponding packet count.
2. Program the DIEPCTLx register with the endpoint characteristics and set the CNAK and Endpoint Enable bits.
3. When transmitting non-zero length data packet, the application must poll the DTXFSTSx register (where n is the FIFO number associated with that endpoint) to determine whether there is enough space in the data FIFO. The application can optionally use DIEPINTx.TxFEmp before writing the data.

13.5.4.1 Examples

Slave Mode Bulk IN Transaction

These notes refer to **Figure 13-8**.

1. The host attempts to read data (IN token) from an endpoint.
2. On receiving the IN token on the USB, the core returns a NAK handshake, because no data is available in the transmit FIFO.

3. To indicate to the application that there was no data to send, the core generates a DIEPINTx.IN Token Rcvd When TxFIFO Empty interrupt.
4. When data is ready, the application sets up the DIEPTSIZx register with the Transfer Size and Packet Count fields.
5. The application writes one maximum packet size or less of data to the Non-periodic TxFIFO.
6. The host reattempts the IN token.
7. Because data is now ready in the FIFO, the core now responds with the data and the host ACKs it.
8. Because the XferSize is now zero, the intended transfer is complete. The device core generates a DIEPINTx.XferCompl interrupt.
9. The application processes the interrupt and uses the setting of the DIEPINTx.XferCompl interrupt bit to determine that the intended transfer is complete.

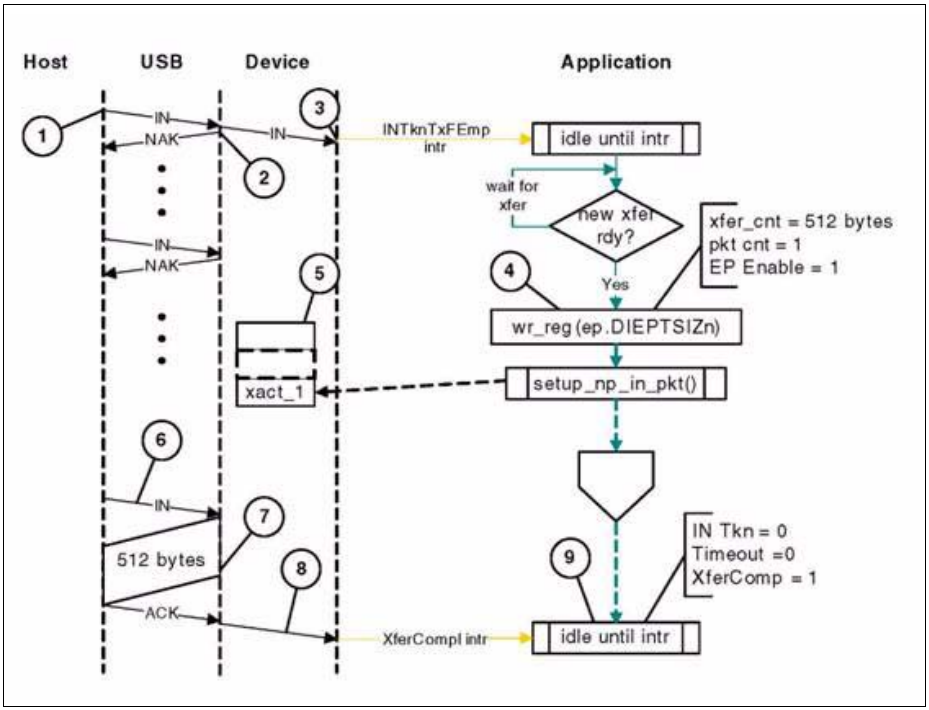


Figure 13-8 Slave Mode Bulk IN Transaction

Slave Mode Bulk IN Transfer (Pipelined Transaction)

These notes refer to **Figure 13-9**.

1. The host attempts to read data (IN token) from an endpoint.
2. On receiving the IN token on the USB, the core returns a NAK handshake, because no data is available in the transmit FIFO.
3. To indicate that there was no data to send, the core generates an DIEPINTx.InTkn Rcvd When TxFIFO Empty interrupt.
4. When data is ready, the application sets up the DIEPTSIZx register with the transfer size and packet count.
5. The application writes one maximum packet size or less of data to the Non-periodic TxFIFO.
6. The host reattempts the IN token.
7. Because data is now ready in the FIFO, the core responds with the data, and the host ACKs it.
8. When the TxFIFO level falls below the halfway mark, the core generates a GINTSTS.NonPeriodic TxFIFO Empty interrupt. This triggers the application to start writing additional data packets to the FIFO.
9. A data packet for the second transaction is ready in the TxFIFO.
10. A data packet for third transaction is ready in the TxFIFO while the data for the second packet is being sent on the bus.
11. The second data packet is sent to the host.
12. The last short packet is sent to the host.
13. Because the last packet is sent and XferSize is now zero, the intended transfer is complete. The core generates a DIEPINTx.XferCompl interrupt.
14. The application processes the interrupt and uses the setting of the DIEPINTx.XferCompl interrupt bit to determine that the intended transfer is complete

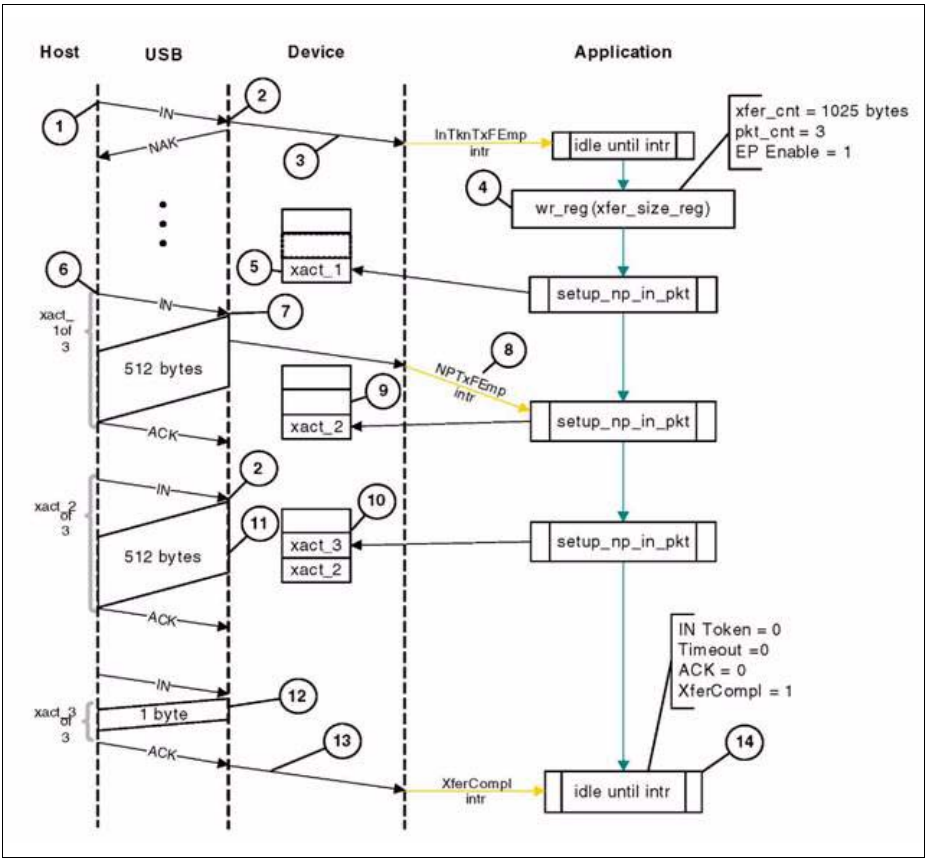


Figure 13-9 Slave Mode Bulk IN Transfer (Pipelined Transaction

Slave Mode Bulk IN Two-Endpoint Transfer

These notes refer to **Figure 13-10**.

1. The host attempts to read data (IN token) from endpoint 1.
2. On receiving the IN token on the USB, the core returns a NAK handshake, because no data is available in the transmit FIFO for endpoint 1, and generates a DIEPINTl.InTkn Rcvd When TxFIFO Empty interrupt.
3. The application processes the interrupt and initializes DIEPTSIZ1 register with the Transfer Size and Packet Count fields. The application starts writing the transaction data to the transmit FIFO.

4. The application writes one maximum packet size or less of data for endpoint 1 to the Non-periodic TxFIFO.
5. Meanwhile, the host attempts to read data (IN token) from endpoint 2.
6. On receiving the IN token on the USB, the core returns a NAK handshake, because no data is available in the transmit FIFO for endpoint 2, and the core generates a DIEPINT2.InTkn Rcvd When TxFIFO Empty interrupt.
7. Because the application has completed writing the packet for endpoint 1, it initializes the DIEPTSIZ2 register with the Transfer Size and Packet Count fields. The application starts writing the transaction data into the transmit FIFO for endpoint 2.
8. The host repeats its attempt to read data (IN token) from endpoint 1.
9. Because data is now ready in the TxFIFO, the core returns the data, which the host ACKs.
10. Meanwhile, the application has initialized the data for the next two packets in the TxFIFO (ep2.xact1 and ep1.xact2, in order).
11. The host repeats its attempt to read data (IN token) from endpoint 2.
12. Because endpoint 2's data is ready, the core responds with the data (ep2.xact_1), which the host ACKs.
13. Meanwhile, the application has initialized the data for the next two packets in the TxFIFO (ep2.xact2 and ep1.xact3, in order). The application has finished initializing data for the two endpoints involved in this scenario.
14. The host repeats its attempt to read data (IN token) from endpoint 1.
15. Because data is now ready in the FIFO, the core responds with the data, which the host ACKs.
16. The host repeats its attempt to read data (IN token) from endpoint 2.
17. With data now ready in the FIFO, the core responds with the data, which the host ACKs.
18. With the last packet for endpoint 2 sent and its XferSize now zero, the intended transfer is complete. The core generates a DIEPINT2.XferCompl interrupt for this endpoint.
19. The application processes the interrupt and uses the setting of the DIEPINT2.XferCompl interrupt bit to determine that the intended transfer on endpoint 2 is complete.
20. The host repeats its attempt to read data (IN token) from endpoint 1 (last transaction).
21. With data now ready in the FIFO, the core responds with the data, which the host ACKs.
22. Because the last endpoint one packet has been sent and XferSize is now zero, the intended transfer is complete. The core generates a DIEPINT1.XferCompl interrupt for this endpoint.
23. The application processes the interrupt and uses the setting of the DIEPINT1.XferCompl interrupt bit to determine that the intended transfer on endpoint 1 is complete.

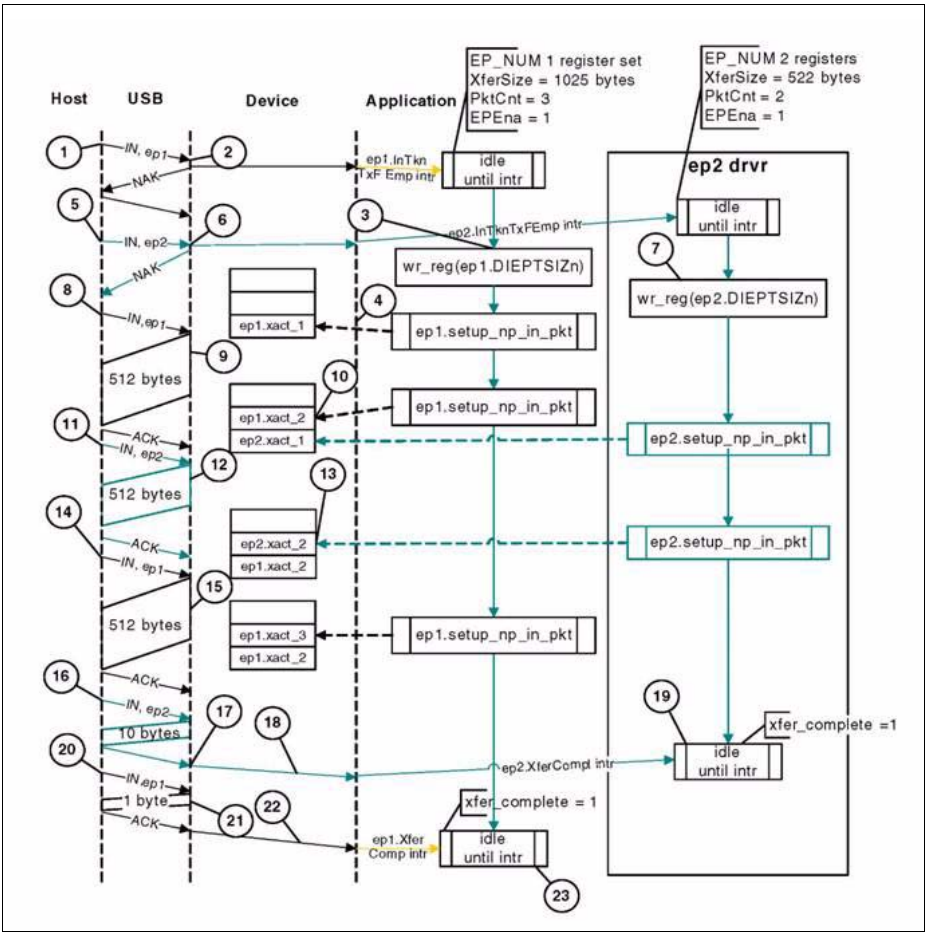


Figure 13-10 Slave Mode Bulk IN Two-Endpoint Transfer

13.5.5 Non-Isochronous OUT Data Transfers

This section describes a regular non-isochronous OUT data transfer (control, bulk, or interrupt).

Application Requirements

1. For OUT transfers, the Transfer Size field in the endpoint's Transfer Size register must be a multiple of the maximum packet size of the endpoint, adjusted to the

DWORD boundary.

```
if (mps[epnum] mod 4) == 0
transfer size[epnum] = n * (mps[epnum] //DWORD aligned
else
transfer size[epnum] = n * (mps[epnum] + 4 - (mps[epnum] mod 4))
//Non-DWORD aligned
packet count[epnum] = n
n > 0
```

2. On any OUT endpoint interrupt, the application must read the endpoint's Transfer Size register to calculate the size of the payload in the memory. The received payload size can be less than the programmed transfer size.
   a) Payload size in memory = application-programmed initial transfer size - core updated final transfer size
   b) Number of USB packets in which this payload was received = application-programmed initial packet count - core updated final packet count

Internal Data Flow

1. The application must set the Transfer Size and Packet Count fields in the endpoint-specific registers, clear the NAK bit, and enable the endpoint to receive the data.
2. Once the NAK bit is cleared, the core starts receiving data and writes it to the receive FIFO, as long as there is space in the receive FIFO. For every data packet received on the USB, the data packet and its status are written to the receive FIFO. Every packet (maximum packet size or short packet) written to the receive FIFO decrements the Packet Count field for that endpoint by 1.
   a) OUT data packets received with Bad Data CRC are flushed from the receive FIFO automatically.
   b) After sending an ACK for the packet on the USB, the core discards non-isochronous OUT data packets that the host, which cannot detect the ACK, re-sends. The application does not detect multiple back-to-back data OUT packets on the same endpoint with the same data PID. In this case the packet count is not decremented.
   c) If there is no space in the receive FIFO, isochronous or non-isochronous data packets are ignored and not written to the receive FIFO. Additionally, non-isochronous OUT tokens receive a NAK handshake reply.
   d) In all the above three cases, the packet count is not decremented because no data is written to the receive FIFO.
3. When the packet count becomes 0 or when a short packet is received on the endpoint, the NAK bit for that endpoint is set. Once the NAK bit is set, the isochronous or non-isochronous data packets are ignored and not written to the receive FIFO, and non-isochronous OUT tokens receive a NAK handshake reply.
4. After the data is written to the receive FIFO, the application reads the data from the receive FIFO and writes it to external memory, one packet at a time per endpoint.

5. At the end of every packet write on the AHB to external memory, the transfer size for the endpoint is decremented by the size of the written packet.
6. The OUT Data Transfer Completed pattern for an OUT endpoint is written to the receive FIFO on one of the following conditions.
   a) The transfer size is 0 and the packet count is 0
   b) The last OUT data packet written to the receive FIFO is a short packet (0 ^packet size < maximum packet size)
7. When he application pops this entry (OUT Data Transfer Completed), a Transfer Completed interrupt is generated for the endpoint and the endpoint enable is cleared.

Application Programming Sequence

1. Program the DOEPTSIZx register for the transfer size and the corresponding packet count.
2. Program the DOEPCTLx register with the endpoint characteristics, and set the Endpoint Enable and ClearNAK bits.
   a) DOEPCTLx.EPEna = 1
   b) DOEPCTLx.CNAK = 1
3. In Slave mode, wait for the GINTSTS.Rx StsQ level interrupt and empty the data packets from the receive FIFO as explained in **“Packet Read from FIFO” on Page 13-29**.
   a) This step can be repeated many times, depending on the transfer size.
4. Asserting the DOEPINTx.XferCompl interrupt marks a successful completion of the non- isochronous OUT data transfer.
5. Read the DOEPTSIZx register to determine the size of the received data payload.

Note: The XferSize is not decremented for the last packet.

Bulk OUT Transactions in Slave Mode

Figure 13-11 depicts the reception of a single bulk OUT data packet from the USB to the AHB and describes the events involved in the process.

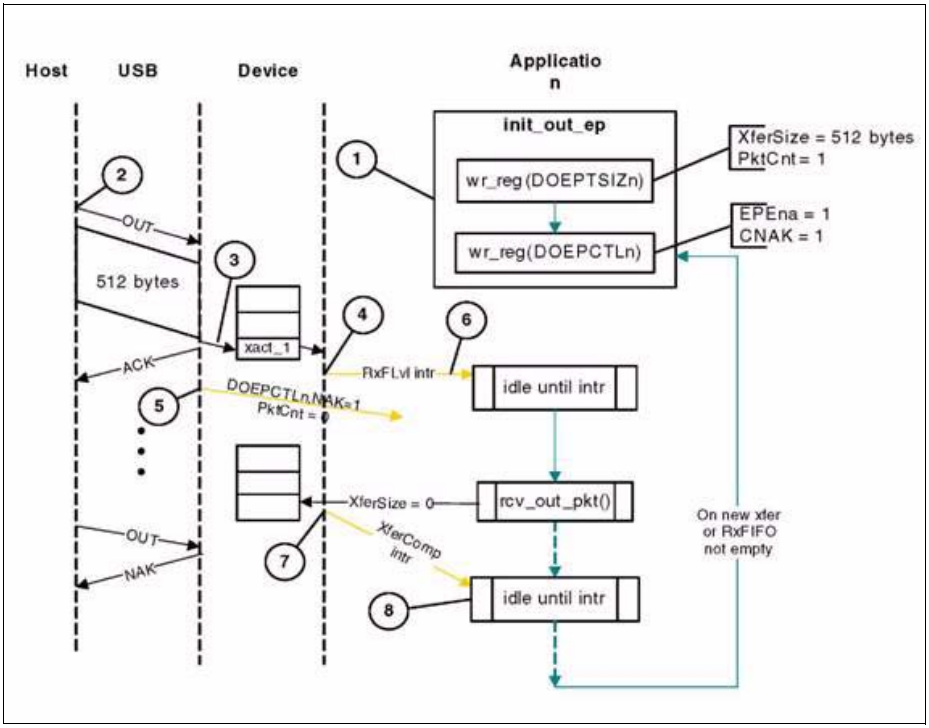


Figure 13-11 Slave Mode Bulk OUT Transaction

After a SetConfiguration/SetInterface command, the application initializes all OUT endpoints by setting DOEPCTLx.CNAK = 1 and DOEPCTLx.EPEna = 1, and setting a suitable XferSize and PktCnt in the DOEPTSIZx register.

1. Host attempts to send data (OUT token) to an endpoint.
2. When the core receives the OUT token on the USB, it stores the packet in the RxFIFO because space is available there.
3. After writing the complete packet in the RxFIFO, the core then asserts the GINTSTS.RxFLvl interrupt.
4. On receiving the PktCnt number of USB packets, the core sets the NAK bit for this endpoint internally to prevent it from receiving any more packets.
5. The application processes the interrupt and reads the data from the RxFIFO.

6. When the application has read all the data (equivalent to XferSize), the core generates a DOEPINTx.XferCompl interrupt.
7. The application processes the interrupt and uses the setting of the DOEPINTx.XferCompl interrupt bit to determine that the intended transfer is complete.

13.5.6 Isochronous OUT Data Transfers

This section describes a regular isochronous OUT data transfer.

Application Requirements

1. All the application requirements for non-isochronous OUT data transfers also apply to isochronous OUT data transfers
2. For isochronous OUT data transfers, the Transfer Size and Packet Count fields must always be set to the number of maximum-packet-size packets that can be received in a single frame and no more. Isochronous OUT data transfers cannot span more than 1 frame.
   a) 1 <= packet count[epnum] <= 3
3. When isochronous OUT endpoints are supported in the device, the application must read all isochronous OUT data packets from the receive FIFO (data and status) before the end of the periodic frame (GINTSTS.EOPF interrupt).
4. To receive data in the following frame, an isochronous OUT endpoint must be enabled after the GINTSTS.EOPF and before the GINTSTS.SOF.

Internal Data Flow

1. The internal data flow for isochronous OUT endpoints is the same as that for non-isochronous OUT endpoints, but for a few differences.
2. When an isochronous OUT endpoint is enabled by setting the Endpoint Enable and clearing the NAK bits, the Even/Odd frame bit must also be set appropriately. The core receives data on a isochronous OUT endpoint in a particular frame only if the following condition is met.
   a) DOEPCTLx.Even/Odd frame = DSTS.SOFFN[0]
3. When the application completely reads an isochronous OUT data packet (data and status) from the receive FIFO, the core updates the DOEPTSIZx.Received DPID field with the data PID of the last isochronous OUT data packet read from the receive FIFO.

Application Programming Sequence

1. Program the DOEPTSIZx register for the transfer size and the corresponding packet count.
2. Program the DOEPCTLx register with the endpoint characteristics and set the Endpoint Enable, ClearNAK, and Even/Odd frame bits.

a) Endpoint Enable = 1
b) CNAK=1
c) Even/Odd frame = (0: Even/1: Odd)

3. In Slave mode, wait for the GINTSTS.Rx StsQ level interrupt and empty the data packets from the receive FIFO as explained in **“Packet Read from FIFO” on Page 13-29**.
   a) This step can be repeated many times, depending on the transfer size.
4. The assertion of the DOEPINTx.XferCompl interrupt marks the completion of the isochronous OUT data transfer. This interrupt does not necessarily mean that the data in memory is good.
5. This interrupt can not always be detected for isochronous OUT transfers. Instead, the application can detect the GINTSTS.incomplete Isochronous OUT data interrupt. See **“Incomplete Isochronous OUT Data Transfers” on Page 13-52**, for more details
6. Read the DOEPTSIZx register to determine the size of the received transfer and to determine the validity of the data received in the frame. The application must treat the data received in memory as valid only if one of the following conditions is met.
   a) DOEPTSIZx.RxDPID = D0 and the number of USB packets in which this payload was received = 1
   b) DOEPTSIZx.RxDPID = D1 and the number of USB packets in which this payload was received = 2
   c) DOEPTSIZx.RxDPID = D2 and the number of USB packets in which this payload was received = 3
7. The number of USB packets in which this payload was received = App Programmed Initial Packet Count - Core Updated Final Packet Count
   The application can discard invalid data packets.

13.5.7 Isochronous OUT Data Transfers Using Periodic Transfer Interrupt

This section describes a regular isochronous OUT data transfer with the Periodic Transfer Interrupt feature.

Application Requirements

1. Before setting up ISOC OUT transfers spanned across multiple frames, the application must allocate buffer in the memory to accommodate all data to be received as part of the OUT transfers, then program that buffer's size and start address in the endpoint-specific registers.
   a) The application must mask the GINTSTS.incomp ISO OUT.
   b) The application must enable the DCTL.IgnrFrmNum
2. For ISOC transfers, the Transfer Size field in the DOEPTSIZx.XferSize register must be a multiple of the maximum packet size of the endpoint, adjusted to the DWORD boundary. The Transfer Size programmed can span across multiple frames based on

the periodicity after which the application wants to receive the DOEPINTx.XferCompl interrupt
a) transfer size[epnum] = n * (mps[epnum] + 4 - (mps[epnum] mod 4))
b) packet count[epnum] = n
c) n > 0 (Higher value of n reduces the periodicity of the DOEPINTx.XferCompl interrupt)
d) 1 =< packet count[epnum] =< n (Higher value of n reduces the periodicity of the DOEPINTx.XferCompl interrupt).

3. In DMA mode, the core stores a received data packet in the memory, always starting on a DWORD boundary. If the maximum packet size of the endpoint is not a multiple of 4, the core inserts byte pads at end of a maximum-packet-size packet up to the end of the DWORD. The application will not be informed about the frame number and the PID value on which a specific OUT packet has been received.
4. The assertion of the DOEPINTx.XferCompl interrupt marks the completion of the isochronous OUT data transfer. This interrupt does not necessarily mean that the data in memory is good.
   a) On DOEPINTx.XferCompl, the application must read the endpoint's Transfer Size register to calculate the size of the payload in the memory.
   b) Payload size in memory = application-programmed initial transfer size - core updated final transfer size
   c) Number of USB packets in which this payload was received = application-programmed initial packet count - core updated final packet count.
   d) If for some reason, the host stop sending tokens, there will be no interrupt to the application, and the application must timeout on its own.
5. The assertion of the DOEPINTx.XferCompl can also mark a packet drop on USB due to unavailability of space in the RxFifo or due to any packet errors.
   a) The application must read the DOEPINTx.PktDrpSts (DOEPINTx.Bit[11] is now used as the DOEPINTx.PktDrpSts) register to differentiate whether the DOEPINTx.XferCompl was generated due to the normal end of transfer or due to dropped packets. In case of packets being dropped on the USB due to unavailability of space in the RxFifo or due to any packet errors the endpoint enable bit is cleared.
   b) In case of packet drop on the USB application must re-enable the endpoint after recalculating the values DOEPTSIZx.XferSize and DOEPTSIZx.PktCnt.
   c) Payload size in memory = application-programmed initial transfer size - core updated final transfer size
   d) Number of USB packets in which this payload was received = application-programmed initial packet count - core updated final packet count.

Note: Due to application latencies it is possible that DOEPINT.XferComplete interrupt is generated without DOEPINT.PktDrpSts being set, This scenario is possible only if back-to-back packets are dropped for consecutive frames and the PktDrpSts is merged, but the XferSize and PktCnt values for the endpoint are nonzero. In this

case, the application must proceed further by programming the PktCnt and XferSize register for the next frame, as it would if PktDrpSts were being set.

Figure 13-12 gives the application flow for Isochronous OUT Periodic Transfer Interrupt feature.

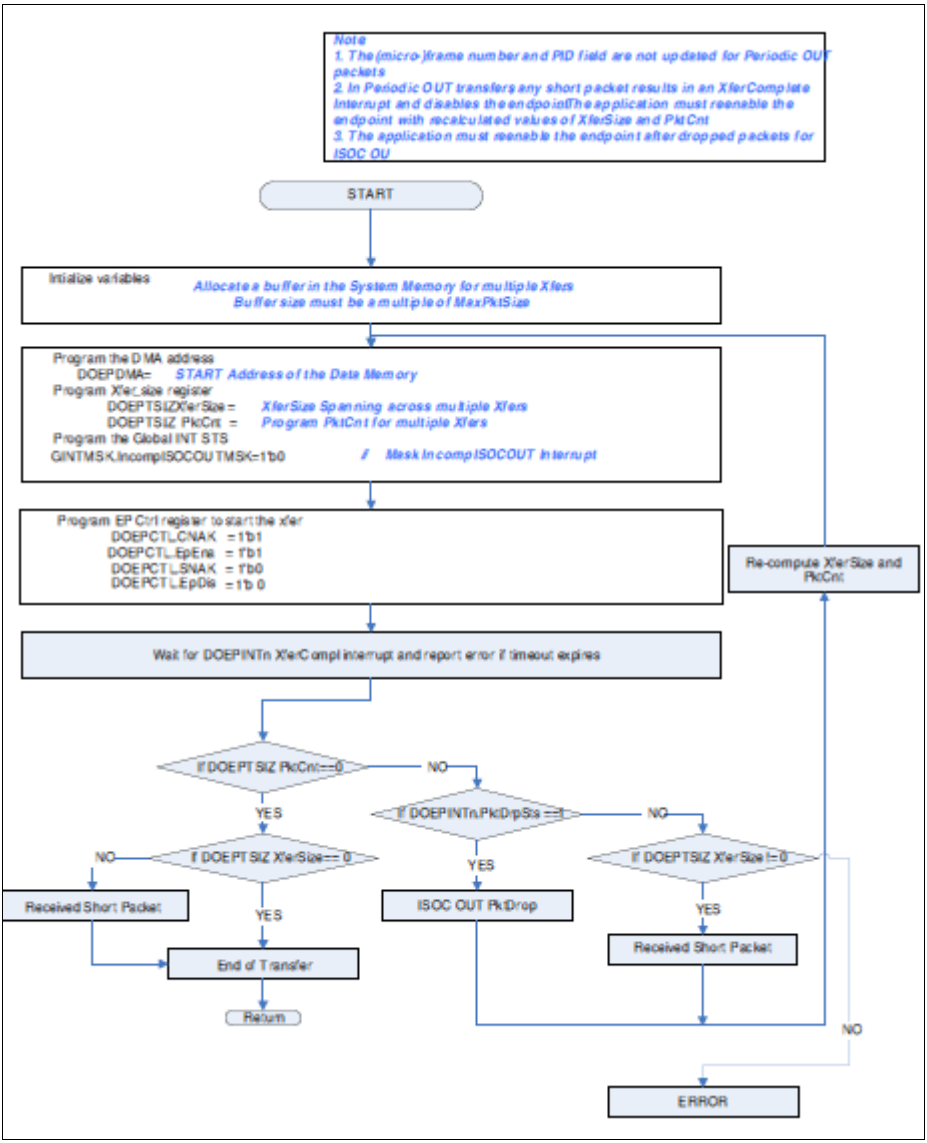


Figure 13-12 ISOC OUT Application Flow for Periodic Transfer Interrupt Feature

Internal Data Flow

1. The application must set the Transfer Size, Packets to be received in a frame and Packet Count Fields in the endpoint-specific registers, clear the NAK bit, and enable the endpoint to receive the data.
2. When an isochronous OUT endpoint is enabled by setting the Endpoint Enable and clearing the NAK bits, the Even/Odd frame will be ignored by the core.
3. Once the NAK bit is cleared, the core starts receiving data and writes it to the receive FIFO, as long as there is space in the receive FIFO. For every data packet received on the USB, the data packet and its status are written to the receive FIFO. Every packet (maximum packet size or short packet) written to the receive FIFO decrements the Packet Count field for that endpoint by 1.
4. When the packet count becomes 0 or when a short packet is received on the endpoint, the NAK bit for that endpoint is set. Once the NAK bit is set, the ISOC packets are ignored and not written to the receive FIFO.
5. After the data is written to the receive FIFO, the core's DMA engine, reads the data from the receive FIFO and writes it to external memory, one packet at a time per endpoint.
6. At the end of every packet write on the AHB to external memory, the transfer size for the endpoint is decremented by the size of the written packet.
7. The OUT Data Transfer Completed pattern for an OUT endpoint is written to the receive FIFO on one of the following conditions.
   a) The transfer size is 0 and the packet count is 0
   b) The last OUT data packet written to the receive FIFO is a short packet (0 < packet size < maximum packet size).
8. When the DMA pops this entry (OUT Data Transfer Completed), a Transfer Completed interrupt is generated for the endpoint or the endpoint enable is cleared.
9. OUT data packets received with Bad Data CRC or any packet error are flushed from the receive FIFO automatically.

In these two cases, the packet count and transfer size registers are not decremented because no data is written to the receive FIFO. **Figure 13-13** illustrates the internal data flow.

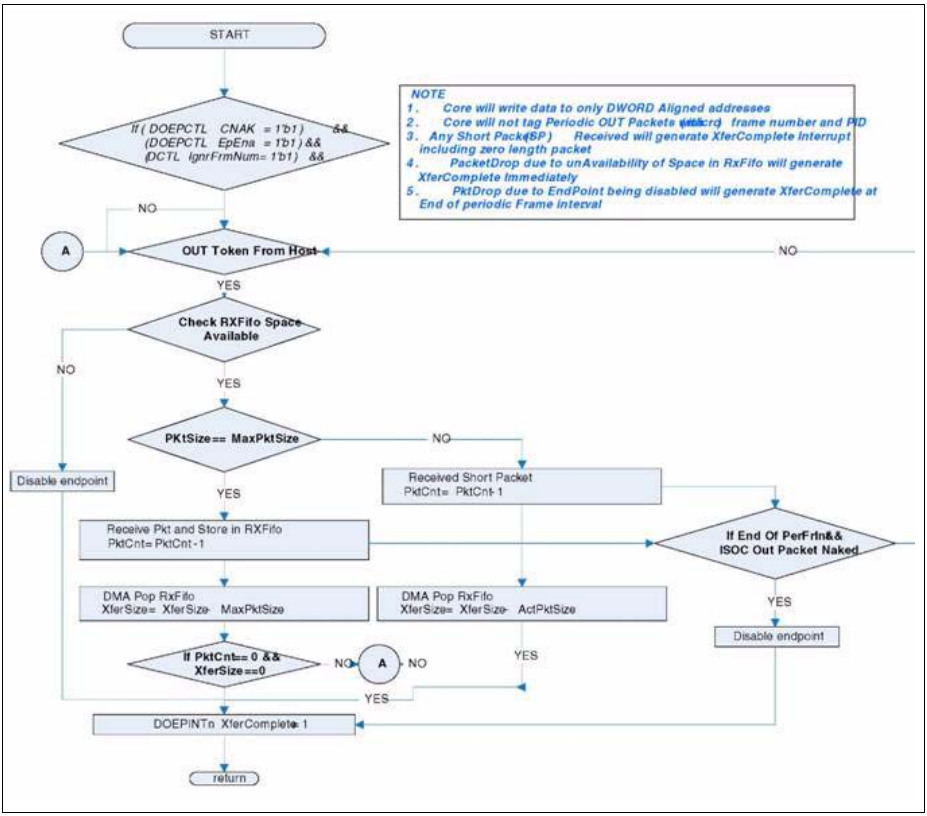


Figure 13-13 Isochronous OUT Core Internal Flow for Periodic Transfer Interrupt Feature

13.5.8 Incomplete Isochronous OUT Data Transfers

This section describes the application programming sequence when isochronous OUT data packets are dropped inside the core.

Internal Data Flow

1. For isochronous OUT endpoints, the DOEPINTx.XferCompl interrupt possibly is not always asserted. If the core drops isochronous OUT data packets, the application could fail to detect the DOEPINTx.XferCompl interrupt under the following circumstances.

a) When the receive FIFO cannot accommodate the complete ISO OUT data packet, the core drops the received ISO OUT data.
b) When the isochronous OUT data packet is received with CRC errors
c) When the isochronous OUT token received by the core is corrupted
d) When the application is very slow in reading the data from the receive FIFO

2. When the core detects an end of periodic frame before transfer completion to all isochronous OUT endpoints, it asserts the GINTSTS.incomplete Isochronous OUT data interrupt, indicating that a DOEPINTx.XferCompl interrupt is not asserted on at least one of the isochronous OUT endpoints. At this point, the endpoint with the incomplete transfer remains enabled, but no active transfers remains in progress on this endpoint on the USB.
3. This step is applicable only if the USB core is operating in slave mode.

Application Programming Sequence

1. Asserting the GINTSTS.incomplete Isochronous OUT data interrupt indicates that in the current frame, at least one isochronous OUT endpoint has an incomplete transfer.
2. If this occurs because isochronous OUT data is not completely emptied from the endpoint, the application must empty all isochronous OUT data (data and status) from the receive FIFO before proceeding.
   a) When all data is emptied from the receive FIFO, the application can detect the DOEPINTx.XferCompl interrupt. In this case, the application must re-enable the endpoint to receive isochronous OUT data in the next frame, as described in **“Isochronous OUT Data Transfers” on Page 13-46**.
3. When it receives a GINTSTS.incomplete Isochronous OUT data interrupt, the application must read the control registers of all isochronous OUT endpoints (DOEPCTLx) to determine which endpoints had an incomplete transfer in the current frame. An endpoint transfer is incomplete if both the following conditions are met.
   a) DOEPCTLx.Even/Odd frame bit = DSTS.SOFFN[0]
   b) DOEPCTLx.Endpoint Enable = 1
4. The previous step must be performed before the GINTSTS.SOF interrupt is detected, to ensure that the current frame number is not changed.
5. For isochronous OUT endpoints with incomplete transfers, the application must discard the data in the memory and disable the endpoint by setting the DOEPCTLx.Endpoint Disable bit.
6. Wait for the DOEPINTx.Endpoint Disabled interrupt and enable the endpoint to receive new data in the next frame as explained in **“Isochronous OUT Data Transfers” on Page 13-46**.
   Because the core can take some time to disable the endpoint, the application possibly is not able to receive the data in the next frame after receiving bad isochronous data.

13.5.9 Incomplete Isochronous IN Data Transfers

This section describes what the application must do on an incomplete isochronous IN data transfer.

Internal Data Flow

1. An isochronous IN transfer is treated as incomplete in one of the following conditions.
   a) The core receives a corrupted isochronous IN token on at least one isochronous IN endpoint. In this case, the application detects a GINTSTS.incomplete Isochronous IN Transfer interrupt.
   b) The application or DMA is slow to write the complete data payload to the transmit FIFO and an IN token is received before the complete data payload is written to the FIFO. In this case, the application detects a DIEPINTx.IN Tkn Rcvd When TxFIFO Empty interrupt. The application can ignore this interrupt, as it eventually results in a GINTSTS.incomplete Isochronous IN Transfer interrupt at the end of periodic frame.
   - The core transmits a zero-length data packet on the USB in response to the received IN token.
2. In either of the aforementioned cases, in Slave mode, the application must stop writing the data payload to the transmit FIFO as soon as possible.
3. The application must set the NAK bit and the disable bit for the endpoint.
4. The core disables the endpoint, clears the disable bit, and asserts the Endpoint Disable interrupt for the endpoint.

Application Programming Sequence

1. The application can ignore the DIEPINTx.IN Tkn Rcvd When TxFIFO empty interrupt on any isochronous IN endpoint, as it eventually results in a GINTSTS.incomplete Isochronous IN Transfer interrupt.
2. Assertion of the GINTSTS.incomplete Isochronous IN Transfer interrupt indicates an incomplete isochronous IN transfer on at least one of the isochronous IN endpoints.
3. The application must read the Endpoint Control register for all isochronous IN endpoints to detect endpoints with incomplete IN data transfers.
4. In Slave mode, the application must stop writing data to the Periodic Transmit FIFOs associated with these endpoints on the AHB.
5. In both modes of operation, program the following fields in the DIEPCTLx register to disable the endpoint. See **“IN Endpoint Disable” on Page 13-19** for more details.
   a) DIEPCTLx.SetNAK = 1
   b) DIEPCTLx.Endpoint Disable = 1
6. The DIEPINTx.Endpoint Disabled interrupt's assertion indicates that the core has disabled the endpoint.

7. At this point, the application must flush the data in the associated transmit FIFO or overwrite the existing data in the FIFO by enabling the endpoint for a new transfer in the next frame. To flush the data, the application must use the GRSTCTL register.

13.5.10 Periodic IN (Interrupt and Isochronous) Data Transfers

This section describes a typical periodic IN data transfer.

Application Requirements

1. Application requirements 1, 2, 3, and 4 of **"Non-Periodic (Bulk and Control) IN Data Transfers" on Page 13-36** also apply to periodic IN data transfers, except for a slight modification of Requirement 2.
   a) The application can only transmit multiples of maximum-packet-size data packets or multiples of maximum-packet-size packets, plus a short packet at the end. To transmit a few maximum- packet-size packets and a short packet at the end of the transfer, the following conditions must be met.
      - transfer size[epnum] = n * mps[epnum] + sp
        (where n is an integer > 0, and 0 <sp < mps[epnum])
      - If (sp > 0), packet count[epnum] = n + 1
        Otherwise, packet count[epnum] = n;
      - mc[epnum] = packet count[epnum]
   b) The application cannot transmit a zero-length data packet at the end of transfer. It can transmit a single zero-length data packet by it self. To transmit a single zero-length data packet,
   c) transfer size[epnum] = 0
      - packet count[epnum] = 1
      - mc[epnum] = packet count[epnum]
2. The application can only schedule data transfers 1 frame at a time.
   a) (DIEPTSIZx.MC - 1) * DIEPCTLx.MPS <DIEPTSIZx.XferSiz <DIEPTSIZx.MC * DIEPCTLx.MPS
   b) DIEPTSIZx. PktCnt = DIEPTSIZx.MC
   c) If DIEPTSIZx.XferSiz < DIEPTSIZx.MC * DIEPCTLx.MPS, the last data packet of the transfer is a short packet.
3. This step is not applicable for isochronous data transfers, only for interrupt transfers. The application can schedule data transfers for multiple frames, only if multiples of max packet sizes (up to 3 packets), must be transmitted every frame. This is can be done, only when the core is operating in DMA mode. This is not a recommended mode though.
   a) ((n*DIEPTSIZx.MC) - 1)*DIEPCTLx.MPS <= DIEPTSIZx.Transfer Size <= n*DIEPTSIZx.MC*DIEPCTLx.MPS
   b) DIEPTSIZx.Packet Count = n*DIEPTSIZx.MC
   c) n is the number of frames for which the data transfers are scheduled

Data Transmitted per frame in this case would be DIEPTSIZx.MC*DIEPCTLx.MPS, in all the frames except the last one. In the frame "n", the data transmitted would be (DIEPTSIZx.TransferSize - (n-1)*DIEPTSIZx.MC*DIEPCTLx.MPS)

4. For Periodic IN endpoints, the data must always be prefetched 1 frame ahead for transmission in the next frame. This can be done, by enabling the Periodic IN endpoint 1 frame ahead of the frame in which the data transfer is scheduled.
5. The complete data to be transmitted in the frame must be written into the transmit FIFO (either by the application or the DMA), before the Periodic IN token is received. Even when 1 DWORD of the data to be transmitted per frame is missing in the transmit FIFO when the Periodic IN token is received, the core behaves as when the FIFO was empty. When the transmit FIFO is empty, a zero data length packet would be transmitted on the USB for ISO IN endpoints. A NAK handshake is transmitted on the USB for INTR IN endpoints.
6. For a High Bandwidth IN endpoint with three packets in a frame, the application can program the endpoint FIFO size to be 2*max_pkt_size and have the third packet load in after the first packet has been transmitted on the USB.

Internal Data Flow

1. The application must set the Transfer Size and Packet Count fields in the endpoint-specific registers and enable the endpoint to transmit the data.
2. The application must also write the required data to the associated transmit FIFO for the endpoint.
3. Every time either the core's internal DMA (in DMA mode) or the application (in Slave mode) writes a packet to the transmit FIFO, the transfer size for that endpoint is decremented by the packet size. The data is fetched from application memory until the transfer size for the endpoint becomes 0.
4. When an IN token is received for an periodic endpoint, the core transmits the data in the FIFO, if available. If the complete data payload (complete packet, in dedicated FIFO mode) for the frame is not present in the FIFO, then the core generates an IN Tkn Rcvd When TxF Empty Interrupt for the endpoint.
   a) A zero-length data packet is transmitted on the USB for isochronous IN endpoints
   b) A NAK handshake is transmitted on the USB for interrupt IN endpoints
5. The packet count for the endpoint is decremented by 1 under the following conditions:
   a) For isochronous endpoints, when a zero- or non-zero-length data packet is transmitted
   b) For interrupt endpoints, when an ACK handshake is transmitted
   c) When the transfer size and packet count are both 0, the Transfer Completed interrupt for the endpoint is generated and the endpoint enable is cleared.

6. At the "Periodic frame Interval" (controlled by DCFG.PerFrint), when the core finds non-empty any of the isochronous IN endpoint FIFOs scheduled for the current frame non-empty, the core generates a GINTSTS.incompISOIN interrupt.

Application Programming Sequence (Transfer Per Frame)

1. Program the DIEPTSIZx register.
2. Program the DIEPCTLx register with the endpoint characteristics and set the CNAK and Endpoint Enable bits.
3. In Slave mode, write the data to be transmitted in the next frame to the transmit FIFO as described in **"Packet Write" on Page 13-31**.
4. Asserting the DIEPINTx.In Token Rcvd When TxF Empty interrupt indicates that the application has not yet written all data to be transmitted to the transmit FIFO.
5. If the interrupt endpoint is already enabled when this interrupt is detected, ignore the interrupt. If it is not enabled, enable the endpoint so that the data can be transmitted on the next IN token attempt.
   a) If the isochronous endpoint is already enabled when this interrupt is detected, see **"Incomplete Isochronous IN Data Transfers" on Page 13-54** for more details.
6. The core handles timeouts internally, without application intervention. The application, thus, never detects a DIEPINTn.TimeOUT interrupt for periodic interrupt IN endpoints.
7. Asserting the DIEPINTx.XferCompl interrupt with no DIEPINTx.In Tkn Rcvd When TxF Empty interrupt indicates the successful completion of an isochronous IN transfer. A read to the DIEPTSIZx register must indicate transfer size = 0 and packet count = 0, indicating all data is transmitted on the USB.
8. Asserting the DIEPINTx.XferCompl interrupt, with or without the DIEPINTx.In Tkn Rcvd When TxF Empty interrupt, indicates the successful completion of an interrupt IN transfer. A read to the DIEPTSIZx register must indicate transfer size = 0 and packet count = 0, indicating all data is transmitted on the USB.
9. Asserting the GINTSTS.incomplete Isochronous IN Transfer interrupt with none of the aforementioned interrupts indicates the core did not receive at least 1 periodic IN token in the current frame.

For isochronous IN endpoints, see **"Incomplete Isochronous IN Data Transfers" on Page 13-54**, for more details.

13.5.11 Periodic IN Data Transfers Using the Periodic Transfer Interrupt

This section describes a typical Periodic IN (ISOC / INTR) data transfer with the Periodic Transfer Interrupt feature.

1. For IN transfers, the Transfer Size field in the Endpoint Transfer Size register denotes a payload that constitutes multiple maximum-packet-size packets and a single short packet. This short packet is transmitted at the end of the transfer.

a) To transmit a few maximum-packet-size packets and a short packet at the end of the transfer:
   - Transfer size[epnum] = n * mps[epnum] + sp
   (where n is an integer > 0, and 0 < sp < mps[epnum]. A higher value of n reduces the periodicity of the DOEPINTx.XferCompl interrupt)
   - If (sp > 0), then packet count[epnum] = n + 1. Otherwise, packet count[epnum] = n
b) To transmit a single zero-length data packet:
   - Transfer size[epnum] = 0
   - Packet count[epnum] = 1
c) To transmit a few maximum-packet-size packets and a zero-length data packet at the end of the transfer, the application must split the transfer in two parts. The first sends maximum-packet- size data packets and the second sends the zero-length data packet alone.
   - First transfer: transfer size[epnum] = n * mps[epnum]; packet count = n;
   - Second transfer: transfer size[epnum] = 0; packet count = 1;
d) The application can only transmit multiples of maximum-packet-size data packets or multiples of maximum-packet-size packets, plus a short packet at the end. To transmit a few maximum- packet-size packets and a short packet at the end of the transfer, the following conditions must be met.
   - transfer size[epnum] = n * mps[epnum] + sp (where n is an integer > 0, and 0 < sp < mps[epnum])
   - If (sp > 0), packet count[epnum] = n + 1 Otherwise, packet count[epnum] = n;
   - mc[epnum] = number of packets to be sent out in a frame.
e) The application cannot transmit a zero-length data packet at the end of transfer. It can transmit a single zero-length data packet by itself. To transmit a single zero-length data packet,
   - transfer size[epnum] = 0
   - packet count[epnum] = 1
   - mc[epnum] = packet count[epnum]

2. Once an endpoint is enabled for data transfers, the core updates the Transfer Size register. At the end of IN transfer, which ended with an Endpoint Disabled interrupt, the application must read the Transfer Size register to determine how much data posted in the transmit FIFO was already sent on the USB.
   a) Data fetched into transmit FIFO = Application-programmed initial transfer size - core-updated final transfer size
   b) Data transmitted on USB = (application-programmed initial packet count - Core updated final packet count) * mps[epnum]
   c) Data yet to be transmitted on USB = (Application-programmed initial transfer size - data transmitted on USB)
3. For Periodic IN endpoints, the data must always be prefetched 1 frame ahead for transmission in the next frame. This can be done, by enabling the Periodic IN endpoint 1 frame ahead of the frame in which the data transfer is scheduled.

4. The complete data to be transmitted in the frame must be written into the transmit FIFO, before the Periodic IN token is received. Even when 1 DWORD of the data to be transmitted per frame is missing in the transmit FIFO when the Periodic IN token is received, the core behaves as when the FIFO was empty. When the transmit FIFO is empty,
   a) A zero data length packet would be transmitted on the USB for ISOC IN endpoints
   b) A NAK handshake would be transmitted on the USB for INTR IN endpoints
   c) DIEPTSIZx.PktCnt is not decremented in this case.

For a High Bandwidth IN endpoint with three packets in a frame, the application can program the endpoint FIFO size to be 2 * max_pkt_size and have the third packet load in after the first packet has been transmitted on the USB.

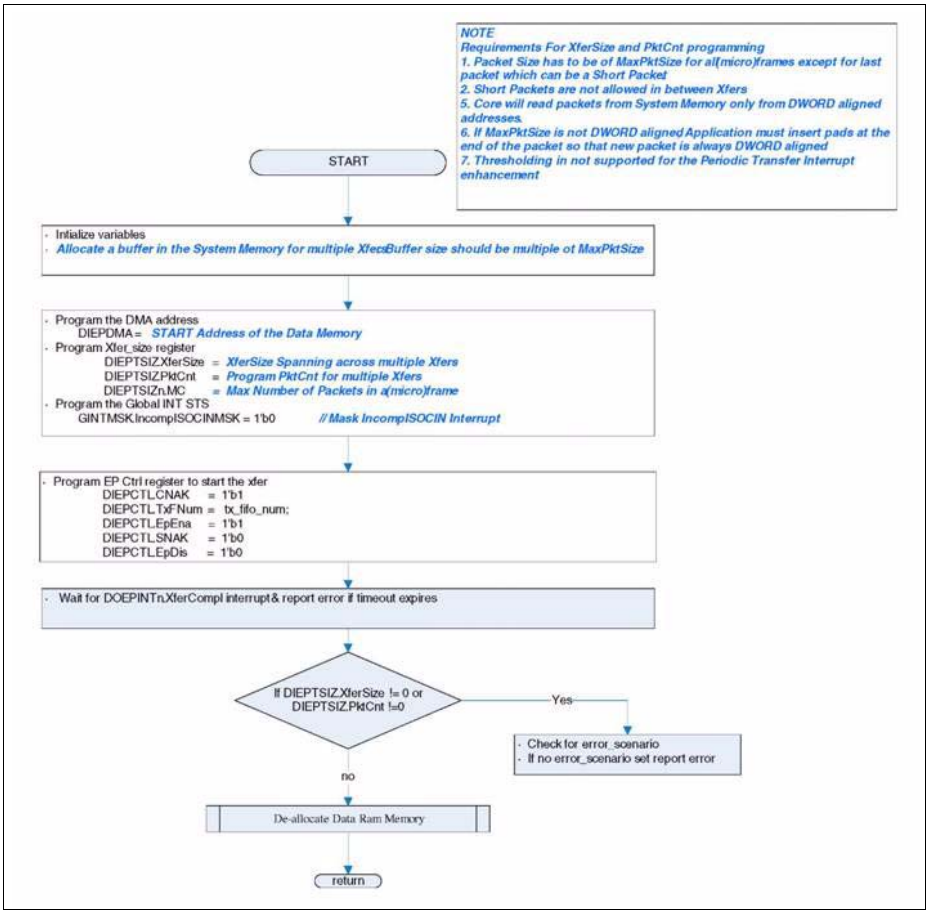


Figure 13-14 Periodic IN Application Flow for Periodic Transfer Interrupt Feature

Internal Data Flow

1. The application must set the Transfer Size and Packet Count fields in the endpoint-specific registers and enable the endpoint to transmit the data.
   a) The application must enable the DCTL.IgnrFrmNum
2. When an isochronous OUT endpoint is enabled by setting the Endpoint Enable and clearing the NAK bits, the Even/Odd frame will be ignored by the core. Subsequently the core updates the Even / Odd bit on its own.
3. Every time the application writes a packet to the transmit FIFO, the transfer size for that endpoint is decremented by the packet size. The data is fetched from application memory until the transfer size for the endpoint becomes 0.
4. When an IN token is received for a periodic endpoint, the core transmits the data in the FIFO, if available. If the complete packet for the frame is not present in the FIFO, then the core generates an IN Tkn Rcvd When TxFifo Empty Interrupt for the endpoint.
   a) A zero-length data packet is transmitted on the USB for isochronous IN endpoints
   b) A NAK handshake is transmitted on the USB for interrupt IN endpoints
5. If an IN token comes for an endpoint on the bus, and if the corresponding TxFIFO for that endpoint has at least 1 packet available, and if the DIEPCTLx.NAK bit is not set, and if the internally maintained even/odd bit match with the bit 0 of the current frame number, then the core will send this data out on the USB. The core will also decrement the packet count. Core also toggles the MultCount in DIEPCTLx register and based on the value of MultCount the next PID value is sent.
   a) If the IN token results in a timeout (core did not receive the handshake or handshake error), core rewind the FIFO pointers. Core does not decrement packet count. It does not toggle PID. DIEPINTx.TimeOUt interrupt will be set which the application could check.
   b) At the end of periodic frame interval (Based on the value programmed in the DCFG.PerFrint register, core will internally set the even/ odd internal bit to match the next frame.
6. The packet count for the endpoint is decremented by 1 under the following conditions:
   a) For isochronous endpoints, when a zero- or non-zero-length data packet is transmitted
   b) For interrupt endpoints, when an ACK handshake is transmitted
7. The data PID of the transmitted data packet is based on the value of DIEPTSIZx.MC programmed by the application. In case the DIEPTSIZx.MC value is set to 3 then, for a particular frame the core expects to receive 3 Isochronous IN token for the respective endpoint. The data PIDs transmitted will be D2 followed by D1 and D0 respectively for the tokens.
   a) If any of the tokens responded with a zero-length packet due to non-availability of data in the TxFIFO, the packet is sent in the next frame with the pending data PID. For example, in a frame, the first received token is responded to with data and data

PID value D2. If the second token is responded to with a zero-length packet, the host is expected not to send any more tokens for the respective endpoint in the current frame. When a token arrives in the next frame it will be responded to with the pending data PID value of D1.

b) Similarly the second token of the current frame gets responded with D0 PID. The host is expected to send only two tokens for this frame as the first token got responded with D1 PID.

8. When the transfer size and packet count are both 0, the Transfer Completed interrupt for the endpoint is generated and the endpoint enable is cleared.
9. The GINTSTS.incompISOIN will be masked by the application hence at the Periodic Frame interval (controlled by DCFG.PerFrint), even though the core finds non-empty any of the isochronous IN endpoint FIFOs, GINTSTS.incompISOIN interrupt will not be generated.

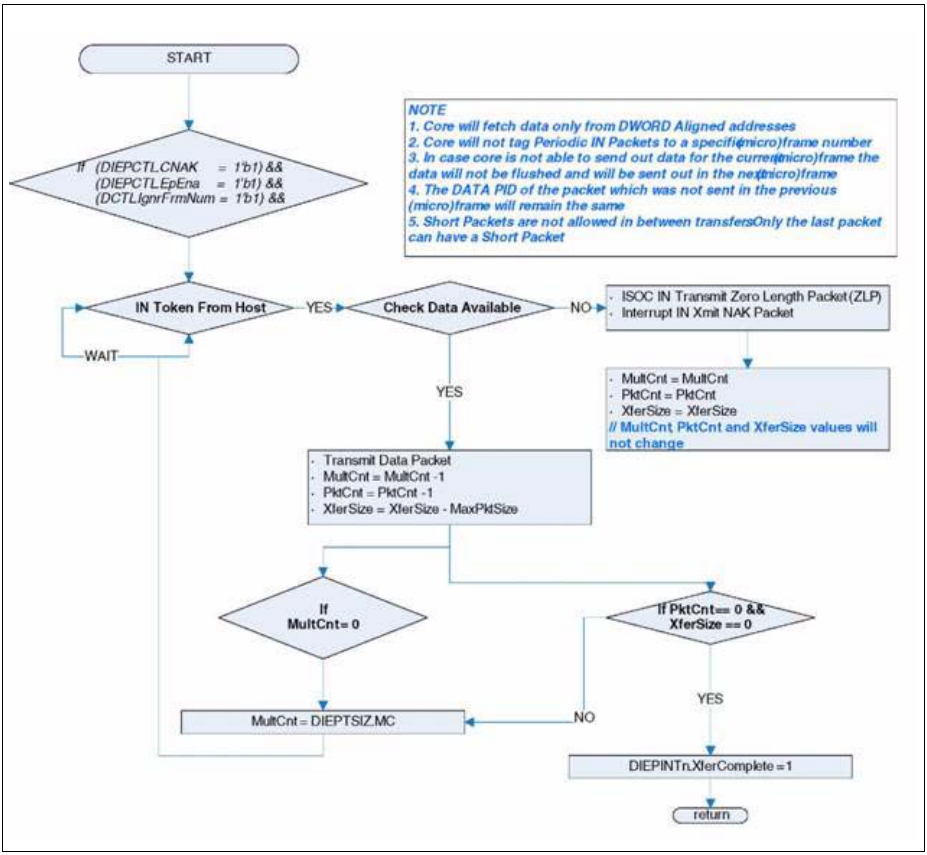


Figure 13-15 Periodic IN Core Internal Flow for Periodic Transfer Interrupt Feature

13.5.12 Interrupt OUT Data Transfers Using Periodic Transfer Interrupt

This section describes a regular INTR OUT data transfer with the Periodic Transfer Interrupt feature.

Application Requirements

1. Before setting up a periodic OUT transfer, the application must allocate a buffer in the memory to accommodate all data to be received as part of the OUT transfer, then program that buffer's size and start address in the endpoint-specific registers.
2. For Interrupt OUT transfers, the Transfer Size field in the endpoint's Transfer Size register must be a multiple of the maximum packet size of the endpoint, adjusted to

the DWORD boundary. The Transfer Size programmed can span across multiple frames based on the periodicity after which the application want to receive the DOEPINTx.XferCompl interrupt
a) transfer size[epnum] = n * (mps[epnum] + 4 - (mps[epnum] mod 4))
b) packet count[epnum] = n
c) n > 0 (Higher value of n reduces the periodicity of the DOEPINTx.XferCompl interrupt)
d) 1 < packet count[epnum] < n (Higher value of n reduces the periodicity of the DOEPINTx.XferCompl interrupt)

3. On DOEPINTx.XferCompl interrupt, the application must read the endpoint's Transfer Size register to calculate the size of the payload in the memory. The received payload size can be less than the programmed transfer size.
   a) Payload size in memory = application-programmed initial transfer size - core updated final transfer size
   b) Number of USB packets in which this payload was received = application-programmed initial packet count - core updated final packet count.
   c) If for some reason, the host stops sending tokens, there are no interrupts to the application, and the application must timeout on its own.
4. The assertion of the DOEPINTx.XferCompl interrupt marks the completion of the interrupt OUT data transfer. This interrupt does not necessarily mean that the data in memory is good.
5. Read the DOEPTSIZx register to determine the size of the received transfer and to determine the validity of the data received in the frame.

Internal Data Flow

1. The application must set the Transfer Size and Packet Count fields in the endpoint-specific registers, clear the NAK bit, and enable the endpoint to receive the data.
   a) The application must enable the DCTL.IgnrFrmNum
2. When an interrupt OUT endpoint is enabled by setting the Endpoint Enable and clearing the NAK bits, the Even/Odd frame will be ignored by the core.
3. Once the NAK bit is cleared, the core starts receiving data and writes it to the receive FIFO, as long as there is space in the receive FIFO. For every data packet received on the USB, the data packet and its status are written to the receive FIFO. Every packet (maximum packet size or short packet) written to the receive FIFO decrements the Packet Count field for that endpoint by 1.
   a) OUT data packets received with Bad Data CRC or any packet error are flushed from the receive FIFO automatically.
   b) Interrupt packets with PID errors are not passed to application. Core discards the packet, sends ACK and does not decrement packet count.
   c) If there is no space in the receive FIFO, interrupt data packets are ignored and not written to the receive FIFO. Additionally, interrupt OUT tokens receive a NAK handshake reply.

4. When the packet count becomes 0 or when a short packet is received on the endpoint, the NAK bit for that endpoint is set. Once the NAK bit is set, the isochronous or interrupt data packets are ignored and not written to the receive FIFO, and interrupt OUT tokens receive a NAK handshake reply.
5. After the data is written to the receive FIFO, the application reads the data from the receive FIFO and writes it to external memory, one packet at a time per endpoint.
6. At the end of every packet write on the AHB to external memory, the transfer size for the endpoint is decremented by the size of the written packet.
7. The OUT Data Transfer Completed pattern for an OUT endpoint is written to the receive FIFO on one of the following conditions.
   a) The transfer size is 0 and the packet count is 0.
   b) The last OUT data packet written to the receive FIFO is a short packet (0 < packet size < maximum packet size)
8. When the application pops this entry (OUT Data Transfer Completed), a Transfer Completed interrupt is generated for the endpoint and the endpoint enable is cleared.

13.6 Device Programming in Buffer DMA Mode

This section discusses how to program the core when it is acting as a Device in the Slave mode of operation.

13.6.1 Control Transfers

This section describes the various types of control transfers.

13.6.1.1 Control Write Transfers (SETUP, Data OUT, Status IN)

This section describes control write transfers.

Application Programming Sequence

1. Assertion of the DOEPINTx.SETUP Packet interrupt indicates that a valid SETUP packet has been transferred to the application. At the end of the Setup stage, the application must reprogram the DOEPTSIZx.SUPCnt field to 3 to receive the next SETUP packet.
2. If the last SETUP packet received before the assertion of the SETUP interrupt indicates a data OUT phase, program the core to perform a control OUT transfer as explained in **“Non-Isochronous OUT Data Transfers” on Page 13-72**.
   The application must reprogram the DOEPDMAx register to receive a control OUT data packet to a different memory location.
3. In a single OUT data transfer on control endpoint 0, the application can receive up to 64 bytes. If the application is expecting more than 64 bytes in the Data OUT stage, the application must re-enable the endpoint to receive another 64 bytes, and must continue to do so until it has received all the data in the Data stage.

4. Assertion of the DOEPINTx.Transfer Compl interrupt on the last data OUT transfer indicates the completion of the data OUT phase of the control transfer.
5. On completion of the data OUT phase, the application must do the following.
   a) To transfer a new SETUP packet in DMA mode, the application must re-enable the control OUT endpoint as explained in section in section "**“OUT Data Transfers” on Page 13-67**.
      - DOEPCTLx.EPEna = 1_B
   b) To execute the received Setup command, the application must program the required registers in the core. This step is optional, based on the type of Setup command received.
6. For the status IN phase, the application must program the core as described in **“Non-Periodic (Bulk and Control) IN Data Transfers” on Page 13-70** to perform a data IN transfer.
7. Assertion of the DIEPINTx.Transfer Compl interrupt indicates completion of the status IN phase of the control transfer.

13.6.1.2 Control Read Transfers (SETUP, Data IN, Status OUT)

This section describes control write transfers.

Application Programming Sequence

1. Assertion of the DOEPINTx.SETUP Packet interrupt indicates that a valid SETUP packet has been transferred to the application. At the end of the Setup stage, the application must reprogram the DOEPTSIZx.SUPCnt field to 3 to receive the next SETUP packet.
2. If the last SETUP packet received before the assertion of the SETUP interrupt indicates a data IN phase, program the core to perform a control IN transfer as explained in **“Non-Periodic (Bulk and Control) IN Data Transfers” on Page 13-70**.
3. On a single IN data transfer on control endpoint 0, the application can transmit up to 64 bytes. To transmit more than 64 bytes in the Data IN stage, the application must re-enable the endpoint to transmit another 64 bytes, and must continue to do so, until it has transmitted all the data in the Data stage.
4. The DIEPINTx.Transfer Compl interrupt on the last IN data transfer marks the completion of the control transfer's Data stage.
5. To perform a data OUT transfer in the status OUT phase, the application must program the core as described in "**“OUT Data Transfers” on Page 13-67**.
   a) The application must program the DCFG.NZStsOUTHShk handshake field to a proper setting before transmitting an data OUT transfer for the Status stage.
   b) The application must then reprogram the DOEPDMAn register to receive the control OUT data packet to a different memory location.

6. Assertion of the DOEPINTx.Transfer Compl interrupt indicates completion of the status OUT phase of the control transfer. This marks the successful completion of the control read transfer.
   a) To transfer a new SETUP packet, the application must re-enable the control OUT endpoint as explained in **“OUT Data Transfers” on Page 13-67**.
   b) DOEPCTLn.EPEna = 1_B

13.6.1.3 Two-Stage Control Transfers (SETUP/Status IN)

This section describes two-stage control transfers.

Application Programming Sequence

1. Assertion of the DOEPINTx.SetUp interrupt indicates that a valid SETUP packet has been transferred to the application. To receive the next SETUP packet, the application must reprogram the DOEPTSIZx.SUPCnt field to 3 at the end of the Setup stage.
2. Decode the last SETUP packet received before the assertion of the SETUP interrupt. If the packet indicates a two-stage control command, the application must do the following.
   a) To transfer a new SETUP packet in DMA mode, the application must re-enable the control OUT endpoint. For more information, see **“OUT Data Transfers” on Page 13-67**.
      - Set DOEPCTLx.EPEna = 1_B
   b) Depending on the type of Setup command received, the application can be required to program registers in the core to execute the received Setup command.
3. For the status IN phase, the application must program the core described in **“Non-Periodic (Bulk and Control) IN Data Transfers” on Page 13-70** to perform a data IN transfer.
4. Assertion of the DIEPINTx.Transfer Compl interrupt indicates the completion of the status IN phase of the control transfer.

13.6.2 OUT Data Transfers

This section describes the internal data flow and application-level operations during data OUT transfers and setup transactions.

13.6.2.1 Control Setup Transactions

This section describes how the core handles SETUP packets and the application's sequence for handling setup transactions. To initialize the core after power-on reset, the application must follow the sequence in **“Core Initialization” on Page 13-9**. Before it can communicate with the host, it must initialize an endpoint as described in **“Endpoint Initialization” on Page 13-13**. See **“Packet Read from FIFO” on Page 13-29**.

Application Requirements

1. To receive a SETUP packet, the DOEPTSIZx.SUPCnt field in a control OUT endpoint must be programmed to a non-zero value. When the application programs the SUPCnt field to a non-zero value, the core receives SETUP packets and writes them to the receive FIFO, irrespective of the DOEPCTLx.NAK status and DOEPCTLx.EPEna bit setting. The SUPCnt field is decremented every time the control endpoint receives a SETUP packet. If the SUPCnt field is not programmed to a proper value before receiving a SETUP packet, the core still receives the SETUP packet and decrements the SUPCnt field, but the application possibly is not be able to determine the correct number of SETUP packets received in the Setup stage of a control transfer.
   a) DOEPTSIZx.SUPCnt = 3
2. In DMA mode, the OUT endpoint must also be enabled, to transfer the received SETUP packet data from the internal receive FIFO to the external memory.
   a) DOEPCTLn.EPEna = 1_B
3. The application must always allocate some extra space in the Receive Data FIFO, to be able to receive up to three SETUP packets on a control endpoint.
   a) The space to be Reserved is 10 DWORDs. Three DWORDs are required for the first SETUP packet, 1 DWORD is required for the Setup Stage Done DWORD, and 6 DWORDs are required to store two extra SETUP packets among all control endpoints.
   b) 3 DWORDs per SETUP packet are required to store 8 bytes of SETUP data and 4 bytes of SETUP status (Setup Packet Pattern). The core reserves this space in the receive data
   c) FIFO to write SETUP data only, and never uses this space for data packets.
4. The core writes the 2 DWORDs of the SETUP data to the memory.
5. The application must read and discard the Setup Stage Done DWORD from the receive FIFO.

Internal Data Flow

1. When a SETUP packet is received, the core writes the received data to the receive FIFO, without checking for available space in the receive FIFO and irrespective of the endpoint's NAK and Stall bit settings.
   a) The core internally sets the IN NAK and OUT NAK bits for the control IN/OUT endpoints on which the SETUP packet was received.
2. For every SETUP packet received on the USB, 3 DWORDs of data is written to the receive FIFO, and the SUPCnt field is decremented by 1.
   a) The first DWORD contains control information used internally by the core
   b) The second DWORD contains the first 4 bytes of the SETUP command
   c) The third DWORD contains the last 4 bytes of the SETUP command

3. When the Setup stage changes to a Data IN/OUT stage, the core writes an entry (Setup Stage Done DWORD) to the receive FIFO, indicating the completion of the Setup stage.
4. On the AHB side, SETUP packets are emptied either by the DMA or the application. In DMA mode, the SETUP packets (2 DWORDs) are written to the memory location programmed in the DOEPDMAn register, only if the endpoint is enabled. If the endpoint is not enabled, the data remains in the receive FIFO until the enable bit is set.
5. When either the DMA or the application pops the Setup Stage Done DWORD from the receive FIFO, the core interrupts the application with a DOEPINTn.SETUP interrupt, indicating it can process the received SETUP packet.
6. The core clears the endpoint enable bit for control OUT endpoints.

Application Programming Sequence

1. Program the DOEPTSIZx register.
   a) DOEPTSIZx.SUPCnt = 3
2. Program the DOEPDMAn register and DOEPCTLn register with the endpoint characteristics and set the Endpoint Enable bit (DOEPCTLn.EPEna).
   a) Endpoint Enable = 1
3. Assertion of the DOEPINTx.SETUP interrupt marks a successful completion of the SETUP Data Transfer.
   a) On this interrupt, the application must read the DOEPTSIZx register to determine the number of SETUP packets received and process the last received SETUP packet.
   b) In DMA mode, the application must also determine if the interrupt bit DOEPINTn.Back2BackSETup is set. This bit is set if the core has received more than three back-to-back SETUP packets. If this is the case, the application must ignore the DOEPTSIZn.SUPCnt value and use the DOEPDMAn directly to read out the last SETUP packet received. DOEPDMAn8 provides the pointer to the last valid SETUP data.

Note: If the application has not enabled EP0 before the host sends the SETUP packet, the core ACKs the SETUP packet and stores it in the FIFO, but does not write to the memory until EP0 is enabled. When the application enables the EP0 (first enable) and clears the NAK bit at the same time the Host sends DATA OUT, the DATA OUT is stored in the RxFIFO. The USB core then writes the setup data to the memory and disables the endpoint. Though the application expects a Transfer Complete interrupt for the Data OUT phase, this does not occur, because the SETUP packet, rather than the DATA OUT packet, enables EP0 the first time. Thus, the DATA OUT packet is still in the RxFIFO until the application re-enables EP0. The application must enable EP0 one more time for the core to process the DATA OUT packet.

Figure 13-7 charts this flow.

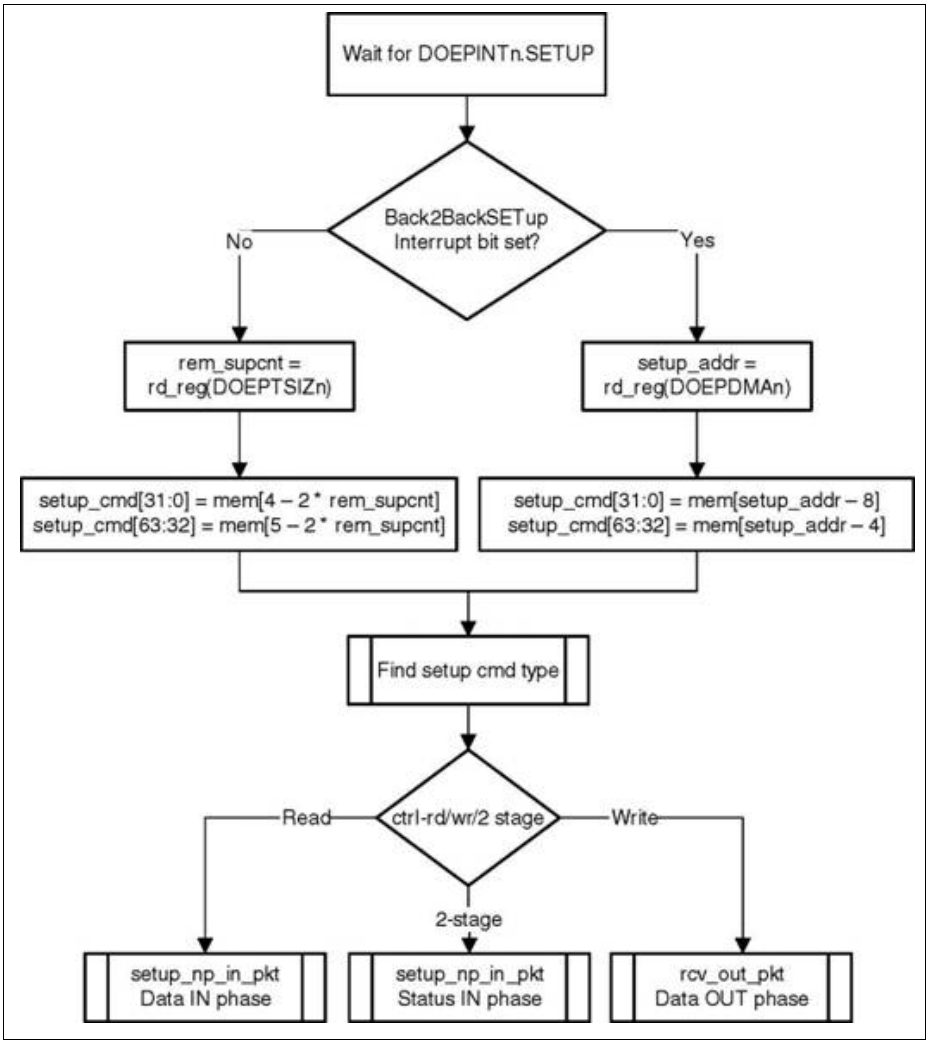


Figure 13-16 Processing a SETUP Packet

13.6.3 Non-Periodic (Bulk and Control) IN Data Transfers

This section describes a regular non-periodic IN data transfer.

Application Requirements

1. Before setting up an IN transfer, the application must ensure that all data to be transmitted as part of the IN transfer is part of a single buffer, and must program the size of that buffer and its start address (in DMA mode) to the endpoint-specific registers.
2. For IN transfers, the Transfer Size field in the Endpoint Transfer Size register denotes a payload that constitutes multiple maximum-packet-size packets and a single short packet. This short packet is transmitted at the end of the transfer.
   a) To transmit a few maximum-packet-size packets and a short packet at the end of the transfer:
   b) Transfer size[epnum] = n * mps[epnum] + sp
   (where n is an integer > 0, and 0 <sp < mps[epnum])
   - If (sp > 0), then packet count[epnum] = n + 1.
   Otherwise, packet count[epnum] = n
   c) To transmit a single zero-length data packet:
   - Transfer size[epnum] = 0
   - Packet count[epnum] = 1
   d) To transmit a few maximum-packet-size packets and a zero-length data packet at the end of the transfer, the application must split the transfer in two parts. The first sends maximum-packet- size data packets and the second sends the zero-length data packet alone.
   - First transfer: transfer size[epnum] = n * mps[epnum]; packet count = n;
   - Second transfer: transfer size[epnum] = 0; packet count = 1;
3. In DMA mode, the core fetches an IN data packet from the memory, always starting at a DWORD boundary. If the maximum packet size of the IN endpoint is not a multiple of 4, the application must arrange the data in the memory with pads inserted at the end of a maximum-packet-size packet so that a new packet always starts on a DWORD boundary.
4. Once an endpoint is enabled for data transfers, the core updates the Transfer Size register. At the end of IN transfer, which ended with an Endpoint Disabled interrupt, the application must read the Transfer Size register to determine how much data posted in the transmit FIFO was already sent on the USB.
5. Data fetched into transmit FIFO = Application-programmed initial transfer size - core-updated final transfer size
   a) Data transmitted on USB = (application-programmed initial packet count - Core updated final packet count) * mps[epnum]
   b) Data yet to be transmitted on USB = (Application-programmed initial transfer size - data transmitted on USB)

Internal Data Flow

1. The application must set the Transfer Size and Packet Count fields in the endpoint-specific registers and enable the endpoint to transmit the data.

2. The core fetches the data from memory according to the application setting for the endpoint.
3. Every time the core's internal DMA writes a packet into the transmit FIFO, the transfer size for that endpoint is decremented by the packet size. The data is fetched from the memory, until the transfer size for the endpoint becomes 0. After writing the data into the FIFO, the "number of packets in FIFO" count is incremented (this is a 3-bit count, internally maintained by the core for each IN endpoint transmit FIFO. The maximum number of packets maintained by the core at any time in an IN endpoint FIFO is eight). For zero-length packets, a separate flag is set for each FIFO, without any data in the FIFO.
4. Once the data is written to the transmit FIFO, the core reads it out upon receiving an IN token. For every non-isochronous IN data packet transmitted with an ACK handshake, the packet count for the endpoint is decremented by one, until the packet count is zero. The packet count is not decremented on a TIMEOUT.
5. For zero length packets (indicated by an internal zero length flag), the core sends out a zero-length packet for the IN token and decrements the Packet Count field.
6. If there is no data in the FIFO for a received IN token and the packet count field for that endpoint is zero, the core generates a IN Tkn Rcvd When FIFO Empty Interrupt for the endpoint, provided the endpoint NAK bit is not set. The core responds with a NAK handshake for non-isochronous endpoints on the USB.
7. In Dedicated FIFO operation, the core internally rewinds the FIFO pointers and no timeout interrupt is generated except for Control IN endpoint.
8. When the transfer size is 0 and the packet count is 0, the transfer complete interrupt for the endpoint is generated and the endpoint enable is cleared.

Application Programming Sequence

1. Program the DIEPTSIZx register with the transfer size and corresponding packet count. Program also the DIEPDMAx register.
2. Program the DIEPCTLx register with the endpoint characteristics and set the CNAK and Endpoint Enable bits.
3. In DMA mode, ensure that the NextEp field is programmed so that the core fetches the data for IN endpoints in the correct order. See **"Non-Periodic IN Endpoint Sequencing" on Page 13-21** for details.
   a) This step can be repeated multiple times, depending on the transfer size.

13.6.4 Non-Isochronous OUT Data Transfers

This section describes a regular non-isochronous OUT data transfer (control, bulk, or interrupt).

Application Requirements

1. Before setting up an OUT transfer, the application must allocate a buffer in the memory to accommodate all data to be received as part of the OUT transfer, then program that buffer's size and start address (in DMA mode) in the endpoint-specific registers.
2. For OUT transfers, the Transfer Size field in the endpoint's Transfer Size register must be a multiple of the maximum packet size of the endpoint, adjusted to the DWORD boundary.
   ```
   if (mps[epnum] mod 4) == 0
   transfer size[epnum] = n * (mps[epnum] //DWORD aligned
   else
   transfer size[epnum] = n * (mps[epnum] + 4 - (mps[epnum] mod 4))
   //Non-DWORD aligned
   packet count[epnum] = n
   n > 0
   ```
3. In DMA mode, the core stores a received data packet in the memory, always starting on a DWORD boundary. If the maximum packet size of the endpoint is not a multiple of 4, the core inserts byte pads at end of a maximum-packet-size packet up to the end of the DWORD.
4. On any OUT endpoint interrupt, the application must read the endpoint's Transfer Size register to calculate the size of the payload in the memory. The received payload size can be less than the programmed transfer size.
   a) Payload size in memory = application-programmed initial transfer size - core updated final transfer size
   b) Number of USB packets in which this payload was received = application-programmed initial packet count - core updated final packet count

Internal Data Flow

1. The application must set the Transfer Size and Packet Count fields in the endpoint-specific registers, clear the NAK bit, and enable the endpoint to receive the data.
2. Once the NAK bit is cleared, the core starts receiving data and writes it to the receive FIFO, as long as there is space in the receive FIFO. For every data packet received on the USB, the data packet and its status are written to the receive FIFO. Every packet (maximum packet size or short packet) written to the receive FIFO decrements the Packet Count field for that endpoint by 1.
   a) OUT data packets received with Bad Data CRC are flushed from the receive FIFO automatically.
   b) After sending an ACK for the packet on the USB, the core discards non-isochronous OUT data packets that the host, which cannot detect the ACK, re-sends. The application does not detect multiple back-to-back data OUT packets on the same endpoint with the same data PID. In this case the packet count is not decremented.

c) If there is no space in the receive FIFO, isochronous or non-isochronous data packets are ignored and not written to the receive FIFO. Additionally, non-isochronous OUT tokens receive a NAK handshake reply.
d) In all the above three cases, the packet count is not decremented because no data is written to the receive FIFO.

3. When the packet count becomes 0 or when a short packet is received on the endpoint, the NAK bit for that endpoint is set. Once the NAK bit is set, the isochronous or non-isochronous data packets are ignored and not written to the receive FIFO, and non-isochronous OUT tokens receive a NAK handshake reply.
4. After the data is written to the receive FIFO, the core's DMA engine reads the data from the receive FIFO and writes it to external memory, one packet at a time per endpoint.
5. At the end of every packet write on the AHB to external memory, the transfer size for the endpoint is decremented by the size of the written packet.
6. The OUT Data Transfer Completed pattern for an OUT endpoint is written to the receive FIFO on one of the following conditions.
   a) The transfer size is 0 and the packet count is 0
   b) The last OUT data packet written to the receive FIFO is a short packet (0 ^packet size < maximum packet size)
7. When either the application or the DMA pops this entry (OUT Data Transfer Completed), a Transfer Completed interrupt is generated for the endpoint and the endpoint enable is cleared.

Application Programming Sequence

1. Program the DOEPTSIZx register for the transfer size and the corresponding packet count. Additionally, in DMA mode, program the DOEPDMAx register.
2. Program the DOEPCTLx register with the endpoint characteristics, and set the Endpoint Enable and ClearNAK bits.
   a) DOEPCTLx.EPEna = 1
   b) DOEPCTLx.CNAK = 1
3. Asserting the DOEPINTx.XferCompl interrupt marks a successful completion of the non- isochronous OUT data transfer.
4. Read the DOEPTSIZx register to determine the size of the received data payload.

Note: The XferSize is not decremented for the last packet.

13.6.5 Incomplete Isochronous OUT Data Transfers

This section describes the application programming sequence when isochronous OUT data packets are dropped inside the core.

Internal Data Flow

1. For isochronous OUT endpoints, the DOEPINTx.XferCompl interrupt possibly is not always asserted. If the core drops isochronous OUT data packets, the application could fail to detect the DOEPINTx.XferCompl interrupt under the following circumstances.
   a) When the receive FIFO cannot accommodate the complete ISO OUT data packet, the core drops the received ISO OUT data.
   b) When the isochronous OUT data packet is received with CRC errors
   c) When the isochronous OUT token received by the core is corrupted
   d) When the application is very slow in reading the data from the receive FIFO
2. When the core detects an end of periodic frame before transfer completion to all isochronous OUT endpoints, it asserts the GINTSTS.incomplete Isochronous OUT data interrupt, indicating that a DOEPINTx.XferCompl interrupt is not asserted on at least one of the isochronous OUT endpoints. At this point, the endpoint with the incomplete transfer remains enabled, but no active transfers remains in progress on this endpoint on the USB.

Application Programming Sequence

1. Asserting the GINTSTS.incomplete Isochronous OUT data interrupt indicates that in the current frame, at least one isochronous OUT endpoint has an incomplete transfer.
2. If this occurs because isochronous OUT data is not completely emptied from the endpoint, the application must empty all isochronous OUT data (data and status) from the receive FIFO before proceeding.
   a) When all data is emptied from the receive FIFO, the application can detect the DOEPINTx.XferCompl interrupt. In this case, the application must re-enable the endpoint to receive isochronous OUT data in the next frame, as described in **“Control Read Transfers (SETUP, Data IN, Status OUT)” on Page 13-66**.
3. When it receives a GINTSTS.incomplete Isochronous OUT data interrupt, the application must read the control registers of all isochronous OUT endpoints (DOEPCTLx) to determine which endpoints had an incomplete transfer in the current frame. An endpoint transfer is incomplete if both the following conditions are met.
   a) DOEPCTLx.Even/Odd frame bit = DSTS.SOFFN[0]
   b) DOEPCTLx.Endpoint Enable = 1
4. The previous step must be performed before the GINTSTS.SOF interrupt is detected, to ensure that the current frame number is not changed.
5. For isochronous OUT endpoints with incomplete transfers, the application must discard the data in the memory and disable the endpoint by setting the DOEPCTLx.Endpoint Disable bit.
6. Wait for the DOEPINTx.Endpoint Disabled interrupt and enable the endpoint to receive new data in the next frame as explained in **“Control Read Transfers (SETUP, Data IN, Status OUT)” on Page 13-66**.

Because the core can take some time to disable the endpoint, the application possibly is not able to receive the data in the next frame after receiving bad isochronous data.

13.6.6 Periodic IN (Interrupt and Isochronous) Data Transfers

This section describes a typical periodic IN data transfer.

Application Requirements

1. Application requirements 1, 2, 3, and 4 of **"Non-Periodic (Bulk and Control) IN Data Transfers" on Page 13-70** also apply to periodic IN data transfers, except for a slight modification of Requirement 2.
   a) The application can only transmit multiples of maximum-packet-size data packets or multiples of maximum-packet-size packets, plus a short packet at the end. To transmit a few maximum- packet-size packets and a short packet at the end of the transfer, the following conditions must be met.
      - transfer size[epnum] = n * mps[epnum] + sp
        (where n is an integer > 0, and 0 <sp < mps[epnum])
      - If (sp > 0), packet count[epnum] = n + 1
        Otherwise, packet count[epnum] = n;
      - mc[epnum] = packet count[epnum]
   b) The application cannot transmit a zero-length data packet at the end of transfer. It can transmit a single zero-length data packet by it self. To transmit a single zero-length data packet,
   c) transfer size[epnum] = 0
      - packet count[epnum] = 1
      - mc[epnum] = packet count[epnum]
2. The application can only schedule data transfers 1 frame at a time.
   a) (DIEPTSIZx.MC - 1) * DIEPCTLx.MPS <DIEPTSIZx.XferSiz <DIEPTSIZx.MC * DIEPCTLx.MPS
   b) DIEPTSIZx. PktCnt = DIEPTSIZx.MC
   c) If DIEPTSIZx.XferSiz < DIEPTSIZx.MC * DIEPCTLx.MPS, the last data packet of the transfer is a short packet.
3. This step is not applicable for isochronous data transfers, only for interrupt transfers. The application can schedule data transfers for multiple frames, only if multiples of max packet sizes (up to 3 packets), must be transmitted every frame. This is can be done, only when the core is operating in DMA mode. This is not a recommended mode though.
   a) ((n*DIEPTSIZx.MC) - 1)*DIEPCTLx.MPS <= DIEPTSIZx.Transfer Size <= n*DIEPTSIZx.MC*DIEPCTLx.MPS
   b) DIEPTSIZx.Packet Count = n*DIEPTSIZx.MC
   c) n is the number of frames for which the data transfers are scheduled

Data Transmitted per frame in this case would be DIEPTSIZx.MC*DIEPCTLx.MPS, in all the frames except the last one. In the frame "n", the data transmitted would be (DIEPTSIZx.TransferSize - (n-1)*DIEPTSIZx.MC*DIEPCTLx.MPS)

4. For Periodic IN endpoints, the data must always be prefetched 1 frame ahead for transmission in the next frame. This can be done, by enabling the Periodic IN endpoint 1 frame ahead of the frame in which the data transfer is scheduled.
5. The complete data to be transmitted in the frame must be written into the transmit FIFO (either by the application or the DMA), before the Periodic IN token is received. Even when 1 DWORD of the data to be transmitted per frame is missing in the transmit FIFO when the Periodic IN token is received, the core behaves as when the FIFO was empty. When the transmit FIFO is empty, a zero data length packet would be transmitted on the USB for ISO IN endpoints. A NAK handshake is transmitted on the USB for INTR IN endpoints.
6. For a High Bandwidth IN endpoint with three packets in a frame, the application can program the endpoint FIFO size to be 2*max_pkt_size and have the third packet load in after the first packet has been transmitted on the USB.

Internal Data Flow

1. The application must set the Transfer Size and Packet Count fields in the endpoint-specific registers and enable the endpoint to transmit the data.
2. The core fetches the data for the endpoint from memory, according to the application setting.
3. Every time either the core's internal DMA writes a packet to the transmit FIFO, the transfer size for that endpoint is decremented by the packet size. The data is fetched from application memory until the transfer size for the endpoint becomes 0.
4. When an IN token is received for an periodic endpoint, the core transmits the data in the FIFO, if available. If the complete data payload (complete packet, in dedicated FIFO mode) for the frame is not present in the FIFO, then the core generates an IN Tkn Rcvd When TxF Empty Interrupt for the endpoint.
   a) A zero-length data packet is transmitted on the USB for isochronous IN endpoints
   b) A NAK handshake is transmitted on the USB for interrupt IN endpoints
5. The packet count for the endpoint is decremented by 1 under the following conditions:
   a) For isochronous endpoints, when a zero- or non-zero-length data packet is transmitted
   b) For interrupt endpoints, when an ACK handshake is transmitted
   c) When the transfer size and packet count are both 0, the Transfer Completed interrupt for the endpoint is generated and the endpoint enable is cleared.
6. At the "Periodic frame Interval" (controlled by DCFG.PerFrint), when the core finds non-empty any of the isochronous IN endpoint FIFOs scheduled for the current frame non-empty, the core generates a GINTSTS.incompISOIN interrupt.

Application Programming Sequence (Transfer Per Frame)

1. Program the DIEPTSIZx and DIEPDMAn registers.
2. Program the DIEPCTLx register with the endpoint characteristics and set the CNAK and Endpoint Enable bits.
3. Asserting the DIEPINTx.In Token Rcvd When TxF Empty interrupt indicates that the application has not yet written all data to be transmitted to the transmit FIFO.
4. If the interrupt endpoint is already enabled when this interrupt is detected, ignore the interrupt. If it is not enabled, enable the endpoint so that the data can be transmitted on the next IN token attempt.
   a) If the isochronous endpoint is already enabled when this interrupt is detected, see **“Incomplete Isochronous IN Data Transfers” on Page 13-54** for more details.
5. The core handles timeouts internally, without application intervention. The application, thus, never detects a DIEPINTn.TimeOUT interrupt for periodic interrupt IN endpoints.
6. Asserting the DIEPINTx.XferCompl interrupt with no DIEPINTx.In Tkn Rcvd When TxF Empty interrupt indicates the successful completion of an isochronous IN transfer. A read to the DIEPTSIZx register must indicate transfer size = 0 and packet count = 0, indicating all data is transmitted on the USB.
7. Asserting the DIEPINTx.XferCompl interrupt, with or without the DIEPINTx.In Tkn Rcvd When TxF Empty interrupt, indicates the successful completion of an interrupt IN transfer. A read to the DIEPTSIZx register must indicate transfer size = 0 and packet count = 0, indicating all data is transmitted on the USB.
8. Asserting the GINTSTS.incomplete Isochronous IN Transfer interrupt with none of the aforementioned interrupts indicates the core did not receive at least 1 periodic IN token in the current frame.

For isochronous IN endpoints, see **“Incomplete Isochronous IN Data Transfers” on Page 13-54**, for more details.

13.6.7 Periodic IN Data Transfers Using the Periodic Transfer Interrupt

This section describes a typical Periodic IN (ISOC / INTR) data transfer with the Periodic Transfer Interrupt feature.

1. Before setting up an IN transfer, the application must ensure that all data to be transmitted as part of the IN transfer is part of a single buffer, and must program the size of that buffer and its start address (in DMA mode) to the endpoint-specific registers.
2. For IN transfers, the Transfer Size field in the Endpoint Transfer Size register denotes a payload that constitutes multiple maximum-packet-size packets and a single short packet. This short packet is transmitted at the end of the transfer.
   a) To transmit a few maximum-packet-size packets and a short packet at the end of the transfer:
      - Transfer size[epnum] = n * mps[epnum] + sp

(where n is an integer > 0, and 0 < sp < mps[epnum]. A higher value of n reduces the periodicity of the DOEPINTx.XferCompl interrupt)
- If (sp > 0), then packet count[epnum] = n + 1. Otherwise, packet count[epnum] = n

b) To transmit a single zero-length data packet:
- Transfer size[epnum] = 0
- Packet count[epnum] = 1

c) To transmit a few maximum-packet-size packets and a zero-length data packet at the end of the transfer, the application must split the transfer in two parts. The first sends maximum-packet- size data packets and the second sends the zero-length data packet alone.
- First transfer: transfer size[epnum] = n * mps[epnum]; packet count = n;
- Second transfer: transfer size[epnum] = 0; packet count = 1;

d) The application can only transmit multiples of maximum-packet-size data packets or multiples of maximum-packet-size packets, plus a short packet at the end. To transmit a few maximum- packet-size packets and a short packet at the end of the transfer, the following conditions must be met.
- transfer size[epnum] = n * mps[epnum] + sp (where n is an integer > 0, and 0 < sp < mps[epnum])
- If (sp > 0), packet count[epnum] = n + 1 Otherwise, packet count[epnum] = n;
- mc[epnum] = number of packets to be sent out in a frame.

e) The application cannot transmit a zero-length data packet at the end of transfer. It can transmit a single zero-length data packet by itself. To transmit a single zero-length data packet,
- transfer size[epnum] = 0
- packet count[epnum] = 1
- mc[epnum] = packet count[epnum]

3. In DMA mode, the core fetches an IN data packet from the memory, always starting at a DWORD boundary. If the maximum packet size of the IN endpoint is not a multiple of 4, the application must arrange the data in the memory with pads inserted at the end of a maximum-packet-size packet so that a new packet always starts on a DWORD boundary.
4. Once an endpoint is enabled for data transfers, the core updates the Transfer Size register. At the end of IN transfer, which ended with an Endpoint Disabled interrupt, the application must read the Transfer Size register to determine how much data posted in the transmit FIFO was already sent on the USB.
   a) Data fetched into transmit FIFO = Application-programmed initial transfer size - core-updated final transfer size
   b) Data transmitted on USB = (application-programmed initial packet count - Core updated final packet count) * mps[epnum]
   c) Data yet to be transmitted on USB = (Application-programmed initial transfer size - data transmitted on USB)

5. The application can schedule data transfers for multiple frames, only if multiples of max packet sizes (up to 3 packets), must be transmitted every frame. This is can be done, only when the core is operating in DMA mode.
   a) ((n*DIEPTSIZn.MC) - 1)*DIEPCTLn.MPS <= DIEPTSIZn.Transfer Size <= n*DIEPTSIZn.MC*DIEPCTLn.MPS
   b) DIEPTSIZn.Packet Count = n*DIEPTSIZn.MC
   c) n is the number of frames for which the data transfers are scheduled. Data Transmitted per frame in this case is DIEPTSIZn.MC*DIEPCTLn.MPS in all frames except the last one. In frame n, the data transmitted is (DIEPTSIZn.TransferSize – (n – 1) * DIEPTSIZn.MC * DIEPCTLn.MPS)
6. For Periodic IN endpoints, the data must always be prefetched 1 frame ahead for transmission in the next frame. This can be done, by enabling the Periodic IN endpoint 1 frame ahead of the frame in which the data transfer is scheduled.
7. The complete data to be transmitted in the frame must be written into the transmit FIFO, before the Periodic IN token is received. Even when 1 DWORD of the data to be transmitted per frame is missing in the transmit FIFO when the Periodic IN token is received, the core behaves as when the FIFO was empty. When the transmit FIFO is empty,
   a) A zero data length packet would be transmitted on the USB for ISOC IN endpoints
   b) A NAK handshake would be transmitted on the USB for INTR IN endpoints
   c) DIEPTSIZx.PktCnt is not decremented in this case.

For a High Bandwidth IN endpoint with three packets in a frame, the application can program the endpoint FIFO size to be 2 * max_pkt_size and have the third packet load in after the first packet has been transmitted on the USB.

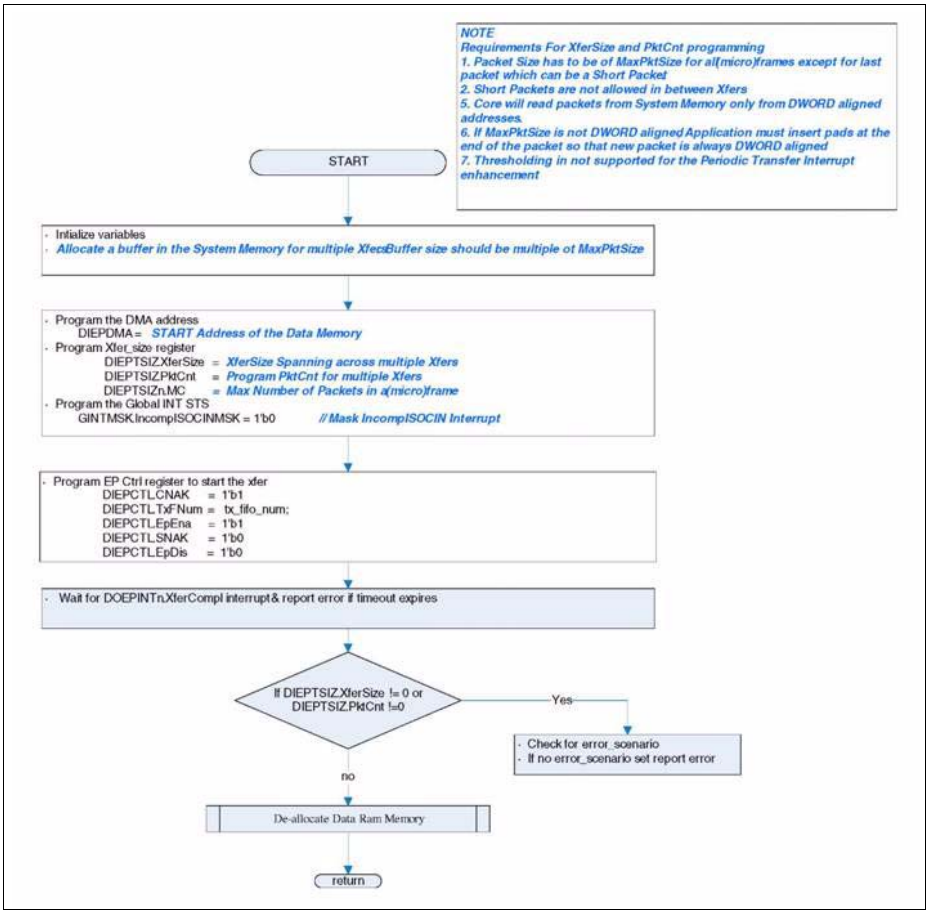


Figure 13-17 Periodic IN Application Flow for Periodic Transfer Interrupt Feature

Internal Data Flow

1. The application must set the Transfer Size and Packet Count fields in the endpoint-specific registers and enable the endpoint to transmit the data.
   a) The application must enable the DCTL.IgnrFrmNum
2. When an isochronous OUT endpoint is enabled by setting the Endpoint Enable and clearing the NAK bits, the Even/Odd frame will be ignored by the core. Subsequently the core updates the Even / Odd bit on its own.
3. Every time either the core's internal DMA writes a packet to the transmit FIFO, the transfer size for that endpoint is decremented by the packet size. The data is fetched from DMA or application memory until the transfer size for the endpoint becomes 0.
4. When an IN token is received for a periodic endpoint, the core transmits the data in the FIFO, if available. If the complete packet for the frame is not present in the FIFO, then the core generates an IN Tkn Rcvd When TxFifo Empty Interrupt for the endpoint.
   a) A zero-length data packet is transmitted on the USB for isochronous IN endpoints
   b) A NAK handshake is transmitted on the USB for interrupt IN endpoints
5. If an IN token comes for an endpoint on the bus, and if the corresponding TxFIFO for that endpoint has at least 1 packet available, and if the DIEPCTLx.NAK bit is not set, and if the internally maintained even/odd bit match with the bit 0 of the current frame number, then the core will send this data out on the USB. The core will also decrement the packet count. Core also toggles the MultCount in DIEPCTLx register and based on the value of MultCount the next PID value is sent.
   a) If the IN token results in a timeout (core did not receive the handshake or handshake error), core rewind the FIFO pointers. Core does not decrement packet count. It does not toggle PID. DIEPINTx.TimeOUt interrupt will be set which the application could check.
   b) At the end of periodic frame interval (Based on the value programmed in the DCFG.PerFrint register, core will internally set the even/ odd internal bit to match the next frame.
6. The packet count for the endpoint is decremented by 1 under the following conditions:
   a) For isochronous endpoints, when a zero- or non-zero-length data packet is transmitted
   b) For interrupt endpoints, when an ACK handshake is transmitted
7. The data PID of the transmitted data packet is based on the value of DIEPTSIZx.MC programmed by the application. In case the DIEPTSIZx.MC value is set to 3 then, for a particular frame the core expects to receive 3 Isochronous IN token for the respective endpoint. The data PIDs transmitted will be D2 followed by D1 and D0 respectively for the tokens.
   a) If any of the tokens responded with a zero-length packet due to non-availability of data in the TxFIFO, the packet is sent in the next frame with the pending data PID. For example, in a frame, the first received token is responded to with data and data

PID value D2. If the second token is responded to with a zero-length packet, the host is expected not to send any more tokens for the respective endpoint in the current frame. When a token arrives in the next frame it will be responded to with the pending data PID value of D1.

b) Similarly the second token of the current frame gets responded with D0 PID. The host is expected to send only two tokens for this frame as the first token got responded with D1 PID.

8. When the transfer size and packet count are both 0, the Transfer Completed interrupt for the endpoint is generated and the endpoint enable is cleared.
9. The GINTSTS.incompISOIN will be masked by the application hence at the Periodic Frame interval (controlled by DCFG.PerFrint), even though the core finds non-empty any of the isochronous IN endpoint FIFOs, GINTSTS.incompISOIN interrupt will not be generated.

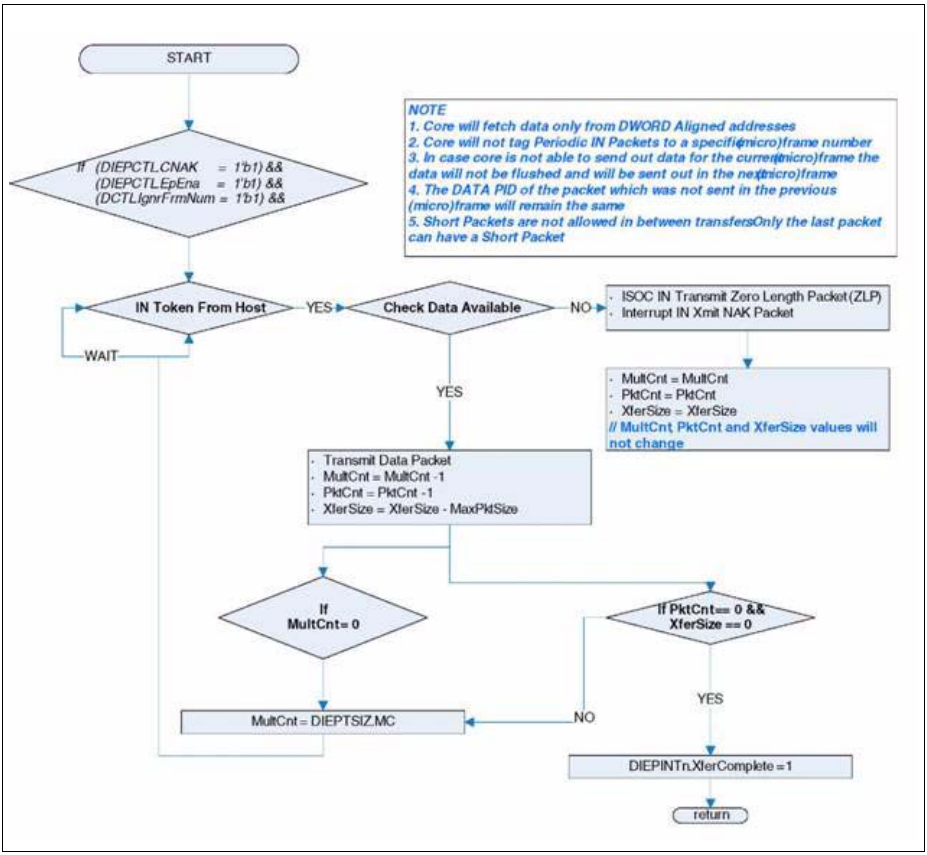


Figure 13-18 Periodic IN Core Internal Flow for Periodic Transfer Interrupt Feature

13.6.8 Interrupt OUT Data Transfers Using Periodic Transfer Interrupt

This section describes a regular INTR OUT data transfer with the Periodic Transfer Interrupt feature.

Application Requirements

1. Before setting up a periodic OUT transfer, the application must allocate a buffer in the memory to accommodate all data to be received as part of the OUT transfer, then program that buffer's size and start address in the endpoint-specific registers.
2. For Interrupt OUT transfers, the Transfer Size field in the endpoint's Transfer Size register must be a multiple of the maximum packet size of the endpoint, adjusted to

the DWORD boundary. The Transfer Size programmed can span across multiple frames based on the periodicity after which the application want to receive the DOEPINTx.XferCompl interrupt
a) transfer size[epnum] = n * (mps[epnum] + 4 - (mps[epnum] mod 4))
b) packet count[epnum] = n
c) n > 0 (Higher value of n reduces the periodicity of the DOEPINTx.XferCompl interrupt)
d) 1 < packet count[epnum] < n (Higher value of n reduces the periodicity of the DOEPINTx.XferCompl interrupt)

3. In DMA mode, the core stores a received data packet in the memory, always starting on a DWORD boundary. If the maximum packet size of the endpoint is not a multiple of 4, the core inserts byte pads at end of a maximum-packet-size packet up to the end of the DWORD. The application will not be informed about the (micro)frame number on which a specific packet has been received.
4. On DOEPINTx.XferCompl interrupt, the application must read the endpoint's Transfer Size register to calculate the size of the payload in the memory. The received payload size can be less than the programmed transfer size.
   a) Payload size in memory = application-programmed initial transfer size - core updated final transfer size
   b) Number of USB packets in which this payload was received = application-programmed initial packet count - core updated final packet count.
   c) If for some reason, the host stops sending tokens, there are no interrupts to the application, and the application must timeout on its own.
5. The assertion of the DOEPINTx.XferCompl interrupt marks the completion of the interrupt OUT data transfer. This interrupt does not necessarily mean that the data in memory is good.
6. Read the DOEPTSIZx register to determine the size of the received transfer and to determine the validity of the data received in the frame.

Internal Data Flow

1. The application must set the Transfer Size and Packet Count fields in the endpoint-specific registers, clear the NAK bit, and enable the endpoint to receive the data.
   a) The application must enable the DCTL.IgnrFrmNum
2. When an interrupt OUT endpoint is enabled by setting the Endpoint Enable and clearing the NAK bits, the Even/Odd frame will be ignored by the core.
3. Once the NAK bit is cleared, the core starts receiving data and writes it to the receive FIFO, as long as there is space in the receive FIFO. For every data packet received on the USB, the data packet and its status are written to the receive FIFO. Every packet (maximum packet size or short packet) written to the receive FIFO decrements the Packet Count field for that endpoint by 1.
   a) OUT data packets received with Bad Data CRC or any packet error are flushed from the receive FIFO automatically.

b) Interrupt packets with PID errors are not passed to application. Core discards the packet, sends ACK and does not decrement packet count.
c) If there is no space in the receive FIFO, interrupt data packets are ignored and not written to the receive FIFO. Additionally, interrupt OUT tokens receive a NAK handshake reply.

4. When the packet count becomes 0 or when a short packet is received on the endpoint, the NAK bit for that endpoint is set. Once the NAK bit is set, the isochronous or interrupt data packets are ignored and not written to the receive FIFO, and interrupt OUT tokens receive a NAK handshake reply.
5. After the data is written to the receive FIFO, the core's DMA engine reads the data from the receive FIFO and writes it to external memory, one packet at a time per endpoint.
6. At the end of every packet write on the AHB to external memory, the transfer size for the endpoint is decremented by the size of the written packet.
7. The OUT Data Transfer Completed pattern for an OUT endpoint is written to the receive FIFO on one of the following conditions.
   a) The transfer size is 0 and the packet count is 0.
   b) The last OUT data packet written to the receive FIFO is a short packet (0 < packet size < maximum packet size)
8. When either the application or the DMA pops this entry (OUT Data Transfer Completed), a Transfer Completed interrupt is generated for the endpoint and the endpoint enable is cleared.

13.7 Device Programming in Scatter-Gather DMA Mode

This chapter describes the programming requirements for the Device core operating in Scatter/Gather DMA mode. It describes how to initialize the channel and provides information on asynchronous transfers (bulk and control) and periodic transfers (isochronous and interrupt).

13.7.1 Programming Overview

When the Scatter/Gather DMA mode is enabled data buffers are presented through descriptor structures

1. The application prepares the descriptors, and sets the bit DIEPCTLx/DOEPCTLx.EPEna.
2. DMA fetches the corresponding descriptor (initially determined by DIEPDMAx/DOEPDMAx).
3. DMA internally sets the transfer size from descriptor back to DIEPTSIZx/DOEPTSIZx.
4. From this point, the current USB flow executes.
5. Once the transfer size data is moved by DMA, the DMA checks for further links in the descriptor chain.

6. If this is the last descriptor, the DMA sets the DIOEPINTn.XferCompl interrupt.
7. If there are further active links, the DMA continues to process them.

Note: The registers DIEPTSIZx/DOEPTSIZx must not be written by the application in Scatter/Gather DMA mode.

In Scatter/Gather DMA mode, the core implements a true scatter-gather memory distribution in which data buffers are scattered over the system memory. Each endpoint memory structure is implemented as a contiguous list of descriptors, in which each descriptor points to a data buffer of predefined size. In addition to the buffer pointer (1 DWORD), the descriptor also has a status quadlet (1 DWORD). When the list is implemented as a ring buffer, the list processor switches to the first element of the list when it encounters last bit. All endpoints (control, bulk, interrupt, and isochronous) implement these structures in memory.

Note: The descriptors are stored in continuos locations. For example descriptor 1 is stored in $0000'0000_H$, descriptor 2 is stored in $0000'0008_H$, descriptor 3 in $0000'0010_H$ and so on. The descriptors are always DWORD aligned.

13.7.2 SPRAM Requirements

For each endpoint the current descriptor pointer and descriptor status are cached to avoid additional requests to system memory. These are stored in SPRAM. In addition DIEPDMAx/DOEPDMAx registers are also implemented in SPRAM.

13.7.3 Descriptor Memory Structures

The descriptor memory structures are displayed in **Figure 13-19**.

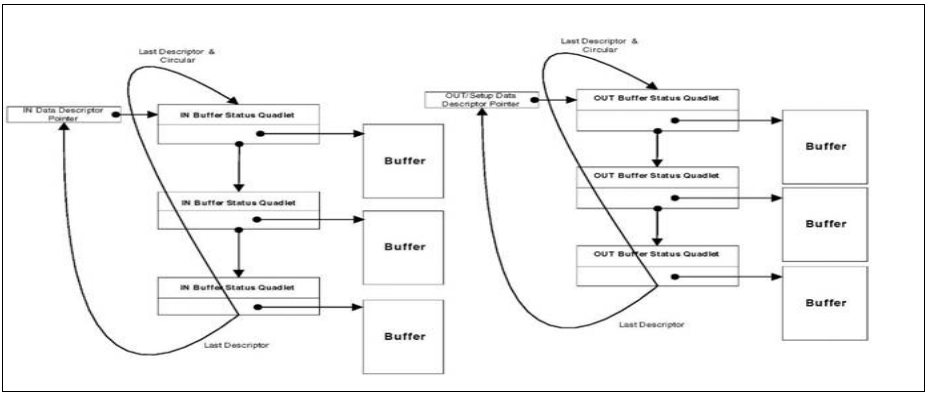


Figure 13-19 Descriptor Memory Structures

13.7.3.1 OUT Data Memory Structure

All endpoints that support OUT direction transactions (endpoints that receive data from the USB host), must implement a memory structure with the following characteristics:

- Each data buffer must have a descriptor associated with it to provide the status of the buffer. The buffer itself contains only raw data.
- Each buffer descriptor is two quadlets in length.

When the buffer status of the first descriptor is host Ready, the DMA fetches and processes its data buffer; otherwise the DMA optionally skips to the next descriptor until it reaches the end of the descriptor chain. The buffers to which the descriptor points hold packet data for non-isochronous endpoints and frame (FS)/µframe (FS) data for isochronous endpoints.

Host Ready — indicates that the descriptor is available for the DMA to process.

DMA Busy — indicates that the DMA is still processing the descriptor.

DMA Done—indicates that the buffer data transfer is complete.

Host Busy—indicates that the application is processing the descriptor.

The OUT data memory structure is shown in **Figure 13-20**, which shows the definition of status quadlet bits for non-ISO and ISO end points

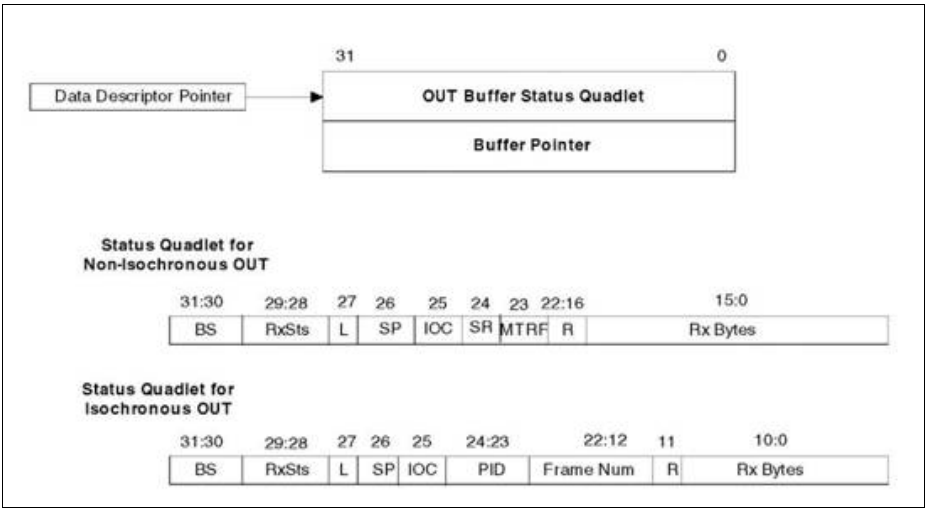


Figure 13-20 Out Data Memory Structure

The status quadlet interpretation depends on the end point type field (DOEPCTLx.EPType) for the corresponding end point. For example, if an end point is OUT and periodic, then the status quadlet is interpreted as Status Quadlet for Isochronous OUT.

Table 13-3 displays the OUT Data Memory Structure fields.

Note: Note that some fields change depending on the mode.

Table 13-3 OUT Data Memory Structure Values

Bit	Bit ID	Description
BS [31:30]	Buffer Status	This 2-bit value describes data buffer status. Possible options are: 00_B Host Ready 01_B DMA Busy 10_B DMA Done 11_BHost Busy Application sets to Host Ready if the descriptor is ready or to Host Busy if the descriptor is not ready. Core sets to DMA busy if the descriptor is being serviced or to DMA Done if the transfer finished associated with the descriptor. The application needs to make these bits as 00_B (Host Ready) as a last step after preparing the entire descriptor ready.Once the software makes these bits as Host Ready then it must not alter the descriptor until DMA completes
Rx Sts [29:28]	Receive Statu	This 2-bit value describes the status of the received data. Core updates this when the descriptor is closed. This reflects whether OUT data has been received correctly or with errors. BUFERR is set by the core when AHB error is encountered during buffer access. BUFERR is set by the core after asserting AHBErr for the corresponding end point.The possible combinations are: • 00_B Success, No AHB errors • 01_B Reserved • 10_B Reserved • 11_B BUFERR
L [27]	Last	Set by the application, this bit indicates that this descriptor is the last one in the chain. Note - L Bit is interpreted by the core even when BS value is other than Host ready. For example, BNA is set, the core keeps traversing all the descriptors until it encounters a descriptor whose L bit is set after which the core disables the corresponding endpoint.
SP[26]	Short Packet	Set by the Core, this bit indicates that this descriptor closed after short packet. When reset it indicates that the descriptor is closed after requested amount of data is received.

Table 13-3 OUT Data Memory Structure Values (cont'd)

Bit	Bit ID	Description	
IOC[25]	Interrupt On complete	Set by the application, this bit indicates that the core must generate a transfer complete interrupt(XferCompl) after this descriptor is finished.	
[24][1)]	Varies	**Non Isochronous Out** **Bit: SR[24]** **Bit ID:** Setup Packet Received Set by the Core, this bit indicates that this buffer holds 8 bytes of setup data. There is only one setup packet per descriptor. On reception of a setup packet, the descriptor is closed and the corresponding endpoint is disabled after SETUP_COMPLETE status is seen in the Rx fifo. The core puts a SETUP_COMPLETE status into the Rx FIFO when it sees the first IN/OUT token after the SETUP packet for that particular endpoint. However, if the L bit of the descriptor is set, the endpoint is disabled and the descriptor is closed irrespective of the SETUP_COMPLETE status. The application has to re-enable for receiving any OUT data for the control transfer. (It also need to reprogram the descriptor start address) Note - Because of the above behavior, the core can receive any number of back to back setup packets and one descriptor for every setup packet is used.	**Isochronous Out** **Bit: Reserved [24:23]** **Bit ID:** This field is reserved and the core writes 00_B.

Table 13-3 OUT Data Memory Structure Values (cont'd)

Bit	Bit ID	Description	
[23][1)]	Varies	**Non Isochronous Out** **Bit: MTRF[23]** **Bit ID:** Multiple Transfer Set by the application, this bit indicates the Core can continue processing the list after it encountered last descriptor.This is to support multiple transfers without application intervention. Reserved for ISO OUT and Control OUT endpoints.	See description for bit [24]
[22:16][1)]	Varies	**Non Isochronous Out** **Bit:** [22:12] **Bit ID:** R Reserved	**Isochronous Out** **Bit: Frame Number [22:12]** **Bit ID:** Frame number The 11-bit frame number corresponds to full speed frame number.

Table 13-3 OUT Data Memory Structure Values (cont'd)

Bit	Bit ID	Description	
[15:12][1)]	Varies	**Non Isochronous Out** **Bit:** Rx Bytes [15:0] **Bit ID:** Received number of bytes remaining This 16-bit value can take values from 0 to (64K-1) bytes, depending on the transfer size of data received from the USB host. The application programs the expected transfer size. When the descriptor is done this indicates remainder of the transfer size. Here, Rx Bytes must be in terms of multiple of MPS for the corresponding end point. The MPS for the various packet types are as follows: • Control – LS - 8 bytes – FS - 8,16,32,64 bytes • Bulk – FS - 8,16,32,64 bytes • Interrupt – LS - up to 8 bytes – FS - up to 64 bytes **Note**: In case of Interrupt packets, the MPS may not be a multiple of 4. If the MPS in an interrupt packet is not a multiple of 4, then a single interrupt packet corresponds to a single descriptor. If MPS is a multiple of 4 for an interrupt packet, then a single descriptor can have multiple MPS packets.	See description for bits [22:16]
[11][1)]	Varies		**Isochronous Out** **Bit:** 11 **Bit ID:** Reserved
[10:0][1)]	Varies		Isochronous Out Bit: Rx Bytes [10:0] Bit ID: Received number of bytes This 11 -bit value can take values from 0 to (2K-1) bytes, depending on the packet size of data received from the USB host.Application programs the expected transfer size. When the descriptor is done this indicates remainder of the transfer size. The maximum payload size of each ISO packet as per USB specification 2.0 is as follows. FS - up to 1023 bytes **Note**: A Value of 0 indicates zero bytes of data, 1 indicates 1 byte of data and so on.

1) The meaning of this field varies. See description.

Table 13-4 displays the matrix of L bit and MTRF bit options.

Table 13-4 OUT - L Bit and MTRF Bit

L Bit	MTRF bit	Functionality
1	1	Continue to process the list after the last descriptor encountered. Use DOEPDMAx as next descriptor. The Endpoint is not disabled.
1	0	For non-Isochronous endpoints, Stop processing list after last descriptor encountered. The application intervenes and programs the list pointer into DOEPDMAx register when a list is created in a new location otherwise enables the endpoint. Start processing when the endpoint is enabled again with DOEPDMAx register pointing to start of list. For Isochronous endpoints, the DMA engine always goes back to the base descriptor address after the last descriptor.
0	1	If a short packet is received or expected transfer is done, close the current descriptor, continue with the next descriptor. If a short packet or Zero length packet is received, the corresponding endpoint is not disabled.
0	0	After processing the current descriptor go to next descriptor. If a short packet OR zero length packet is received disable the endpoint and a transfer complete interrupt is generated irrespective of IOC bit setting for that descriptor.

Table 13-5 displays the out buffer pointer field description.

Note: For Bulk and Interrupt End Points, if MTRF bit is set for the last descriptor in a list, then all the descriptors in that list need to have their MTRF bit set.

Table 13-5 OUT Buffer Pointer

Buf Addr[31:0]	Buffer Address	The Buffer pointer field in the descriptor is 32 bits wide and contains the address where the received data is to be stored in the system memory. The starting buffer address must be DWORD aligned. The buffer size must be also DWORD aligned.

13.7.3.2 Isochronous OUT

- The application must create one descriptor per packet.
- End point is not disabled by the core based on L bit. The DMA always goes back to the base descriptor address after the last descriptor.
- The bit MTRF is not applicable.

13.7.3.3 Non-Isochronous OUT

- The core uses one descriptor per setup packet.
- The core closes the descriptor after receiving a short packet.
- Bit combinations for L and MTRF appear in **Table 13-4**.
- Multiple Interrupt packets in the same buffer is allowed only if the MPS is multiple of 4.

13.7.3.4 IN Data Memory Structure

All endpoints that support IN direction transactions (transmitting data to the USB host) must implement the following memory structure. Each buffer must have a descriptor associated with it. The application fills the data buffer, updates its status in the descriptor, and enables the endpoint. The DMA fetches this descriptor and processes it, moving on in this fashion until it reaches the end of the descriptor chain. The buffer to which the descriptor points to hold packet data for non-isochronous endpoints and frame data for isochronous endpoints.

The definition of status quadlet bits for non-periodic and periodic end points are as shown in the figure. The status quadlet interpretation depends on the end point type field (DIEPCTLx.EPType) for the corresponding end point. For example, if an end point is IN and periodic, then the status quadlet is interpreted as "Status Quadlet for Isochronous IN".

The IN data memory structure is shown in **Figure 13-21**.

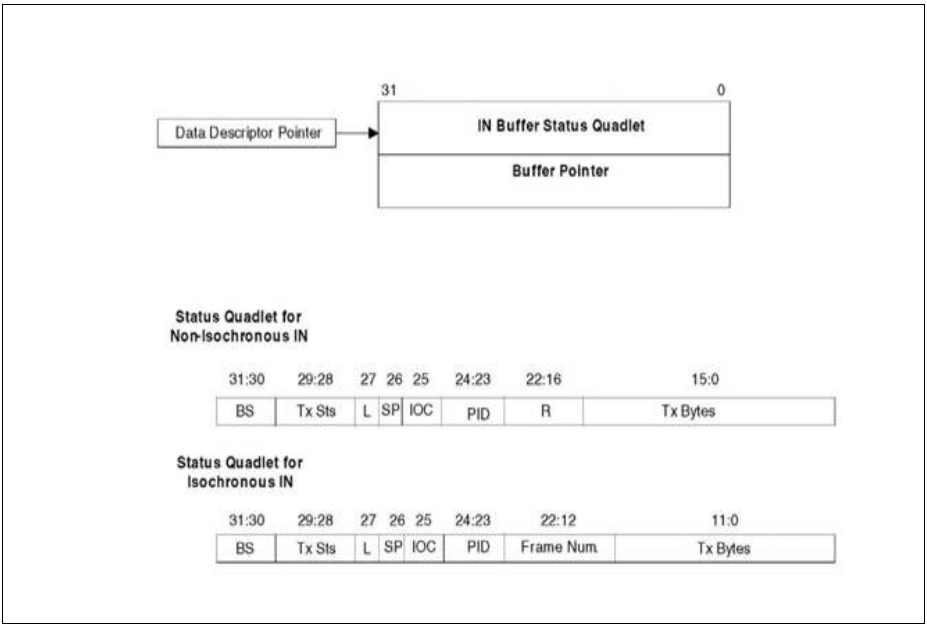


Figure 13-21 IN Data Memory Structure

Table 13-6 displays the IN Data Memory Structure fields.

Note: Some fields change depending on the mode

Table 13-6 IN Data Memory Structure Values

Bit	Bit ID	Description
BS [31:30]	Buffer Status	This 2-bit value describes the status of the data buffer. The possible options are: • 00_B Host ready • 01_B DMA busy • 10_B DMA done • 11_B Host busy The application needs to make these bits as 00_B (Host Ready) as a last step after preparing the entire descriptor ready.Once the software makes these bits as HostReady then it must not alter the descriptor until DMA done
Tx Sts [29:28]	Transmit Status	The status of the transmitted data. This reflects if the IN data has been transmitted correctly or with errors. BUFERR is set by core when there is a AHB error during buffer access. When iIgnrFrmNum is not set, BUFFLUSH is set by the core when • the core is fetching data pertaining to the current frame (N) and finds that the frame has incremented (N+1) during the data fetch • or • when it fetches a descriptor for which the frame number has already elapsed. The possible combinations are: • 00_B Success, No AHB errors • 01_B BUFFLUSH • 10_B Reserved • 11_B BUFERR
L [27]	Last	When set by the application, this bit indicates that this descriptor is the last one in the chain.
SP[26]	Short Packet	When set, this bit indicates that this descriptor points to a short packet or a zero length packet. If there is more than one packet in the descriptor, it indicates that the last packet is a short packet or a zero length packet.
IOC[25]	Interrupt On complete	When set by the application, this bit indicates that the core must generate a transfer complete interrupt after this descriptor is finished.

Table 13-6 IN Data Memory Structure Values (cont'd)

Bit	Bit ID	Description	
[24:23][1)]	Varies	**Non Isochronous In** **Bit:** Reserved[24:16] **Bit ID:** Reserved	**Isochronous In** **Bit:** Reserved[24:23] **Bit ID:** Reserved
[22:12][1)]	Varies		**Isochronous In** **Bit:** Frame Number [22:12] **Bit ID:** This field must correspond to the 11-bit full speed frame number.
[15:12][1)]	Varies	**Non Isochronous In** **Bit:** Tx bytes [15:0] **Bit ID:** Number of bytes to be transmitted This 16-bit value can take values from 0 to (64K-1) bytes, indicating the number of bytes of data to be transmitted to the USB host. **Note**: A Value of 0 indicates zero bytes of data, 1 indicates 1 byte of data and so on.	
[11:0][1)]	Varies		**Isochronous In** **Bit:** Tx bytes [11:0] **Bit ID:** Number of bytes to transmit Tx bytes [11:0] Number of bytes to be transmitted This 12-bit value can take values from 0 to (4K-1) bytes, indicating the number of bytes of data to be transmitted to the USB host. **Note**: A Value of 0 indicates zero bytes of data, 1 indicates 1 byte of data and so on.

1) The meaning of this field varies. See description.

Table 13-7 displays the matrix of IN - L Bit, SP Bit and Tx bytes options.

Table 13-7 IN - L Bit, SP Bit and Tx bytes

L Bit	SP bit	Tx Bytes	Functionality
0	1	Multiple of endpoint maximum packet size	Transmit a zero length packet after the last packet
0	1	Not multiple of maximum packet size	Send short packet at the end after normal packets are sent out. Then move onto next descriptor
0	1	0	Transmit zero length packet. Then move on to next descriptor.
0	0	Multiple of endpoint maximum packet size	Send normal packets and then move to next descriptor.
0	0	Not a multiple of maximum packet size	Transmit the normal packets and concatenate the remaining bytes with next buffer from the next descriptor. This combination is valid only for bulk end points.
0	0	0	Invalid. The behavior of the core is undefined.
1	1	Multiple of endpoint maximum packet size	Transmit a zero length packet after the last packet If this IN descriptor is for a ISO endpoint, then move onto the first descriptor in the list. If this IN descriptor is for a non-ISO endpoint, then stop processing this list and disable the corresponding end point.
1	1	Not multiple of maximum packet size	Send short packet after sending the normal packets If this IN descriptor is for a ISO endpoint, move onto the first descriptor in the list. If this IN descriptor is for a non-ISO endpoint, then stop processing this list and disable the corresponding end point.

Table 13-7 IN - L Bit, SP Bit and Tx bytes (cont'd)

L Bit	SP bit	Tx Bytes	Functionality
1	1	0	Transmit zero length packet If this IN descriptor is for a ISO endpoint, move onto the first descriptor in the list. If this IN descriptor is for a non-ISO endpoint, then stop processing this list and disable the corresponding end point.
1	0	Multiple of endpoint maximum packet size	Send normal packets If this IN descriptor is for a ISO endpoint, Move onto the first descriptor in the list after current transfer done. If this IN descriptor is for a non-ISO endpoint, then stop processing the list and disable the corresponding end point.
1	0	Not multiple of maximum packet size.	Invalid. The behavior of the core is undefined for these values.
1	0	0	invalid. The behavior of the core is undefined for these values.

The descriptions provided for the different combinations in **Table 13-7** depend on the previous descriptor L, SP, and Tx Bytes values. Consider **Table 13-8**. The MPS for this example is 512.

Table 13-8 IN - Buffer Pointer

DESC NO	L bit	SP bit	Txbytes	Description
1	0	0	520	Send a normal packet of size 512, and concatenate the remaining 8 bytes with the next descriptor's buffer data
2	0	1	512	For this combination of L,SP and TxBytes, as per the above table, we need to send a zero length packet instead of a short packet. However, a normal packet followed by a short packet of length 8-bytes is sent. This is to illustrate the context dependency based on previous descriptor L,SP and TxByte combinations.

Table 13-9 displays the IN buffer pointer field description.

Table 13-9 IN Buffer Pointer

Bit	Bit ID	Description
Buf Addr[31:0]	Buffer Address	The Buffer pointer field in the descriptor is 32 bits wide and contains the address where the transmit data is stored in the system memory. The address can be non- DWORD aligned.

13.7.3.5 Descriptor Update Interrupt Enable Modes

If IOC bit is set for a descriptor and if the corresponding Transfer Completed Interrupt Mask (XferComplMask) is unmasked, this interrupt (DIOEPINTn.XferCompl) is asserted while closing that descriptor.

13.7.3.6 DMA Arbitration in Scatter/Gather DMA Mode

The arbiter grants receive higher priority than transmit. Within transmit, the priority is as follows.

- The highest priority is given to periodic endpoints. The periodic endpoints are serviced in a round robin fashion.
- The non periodic endpoints are serviced after the periodic scheduling interval has elapsed. The duration of the periodic scheduling interval is programmable, as specified by register bits DCFG[25:24]. When the periodic interval is active, the periodic endpoints are given priority.
- Amongst the periodic endpoints, the priority is round robin.
- Amongst the non periodic endpoints, the Global Multi Count field in the Device Control Register (DCTL) specifies the number of packets that need to be serviced for that end point before moving to the next endpoint.

The arbiter disables an endpoint and moves on to the next endpoint in the following scenarios as well, for all the endpoint types:

- Descriptor Fetch and AHB Error occurs.
- Buffer Not Available (BNA), such as when buffer status is Host busy.
- AHB Error during Descriptor update stage and Data transfer stage.

13.7.3.7 Buffer Data Access on AHB in Scatter/Gather DMA Mode

The buffer address whose data needs to be accessed in the system memory can be non DWORD aligned for transmit.

For buffer data read, the core arranges the buffer data to form a quadlet internally before populating the TXFIFO within the core as per the following scenarios

- The packet starts in a non DWORD aligned address, the core does two reads on AHB before appending the relevant bytes to form a quadlet internally. Hence the core stores the bytes before pushing to the TXFIFO.
- The packet ends in a non DWORD aligned address and it is not the end of the buffer or expected transfer, the core may switch to service another end point and come back to service the initial end point. In this case, the core reads the same DWORD location again and then samples only the relevant bytes. This eliminates the storage of the bytes for the initial end point.

For buffer data write, the core always performs DWORD accesses.

13.7.4 Control Transfer Handling

Control transfers (3-Stage Control R/WR or 2-Stage), can be handled effectively in the Descriptor-Based Scatter/Gather DMA mode by following the procedure explained in this section. By following this procedure the application is able to handle all normal control transfer flow and any of the following abnormal cases.

- More than one SETUP packet (back to back) — Host could send any number of SETUP packets back to back, before sending any IN/OUT token. In this case, the application is suppose to take the last SETUP packet, and ignore the others.
- More OUT/IN tokens during data phase than what is specified in the wlength field — If the host sends more OUT/IN data tokens than what is specified in the wlength field of the SETUP data, then the device must STALL.
- Premature SETUP packet during data/status phase — Device application must be able to handle this SETUP packet and ignore the previous control transfer.
- Lost ACK for the last data packet of a Three-Stage Control Read Status Stage.

13.7.5 Interrupt Usage for Control Transfers

The application checks the following OUT interrupts status bits for the proper decoding of control transfers.

- DIEPINTx.XferCompl (Transfer complete, based on IOC bit in the descriptor)
- DIEPINTx.InTknTxfEmp (In token received when Tx FIFO is empty)
- DOEPINTx.XferCompl (Transfer complete, based on IOC bit in the descriptor)
- DOEPINTx.SetUp (Setup Complete interrupt, generated when the core receives IN/OUT token after a SETUP packet.
- DOEPINTx.StsPhseRcvd (Status phase received interrupt (Also called SI), generated when host has switched to status phase of a Control Write transfer).

The core performs some optimization of these interrupt settings, when it sees multiple interrupt bits need to be set for OUT endpoints. This reduces the number of valid combinations of interrupts and simplifies the application.

The core gives priority for DOEPINTx.XferCompl over DOEPINTx.SetUp and DOEPINTx.StsPhseRcvd (SI) interrupts. When setting the XferCompl interrupts, it clears the SetUP and SI interrupt bits.

- The core gives priority to DOEPINTx.SI interrupt over DOEPINTx.SetUp. When setting DOEPINTx.StsPhseRcvd (SI), the core clears DOEPINTx.SetUp interrupt bit.

Based on this, the application needs only to decode the combinations of interrupts for OUT endpoints shown in **Table 13-10**.

Table 13-10 Combinations of OUT Endpoint Interrupts for Control Transfer

StsPhse Rcvd (SI)	SetUp (SPD)	XferCompl (IOC)	Description	Template Used
0	0	1	Core has updated the OUT descriptor. Check the "SR" (Setup Received) bit in the descriptor to see if the data is for a SETUP or OUT transaction.	Case A
0	1	0	Setup Phase Done Interrupt for the previously decoded SETUP packet.	Case B
0	1	1	The core has updated the OUT descriptor for a SETUP packet, and the core is indicating a SETUP complete status also.	Case C
1	0	0	Host has switched to Status phase of a Control OUT transfer	Case D
1	0	1	Core has updated the OUT descriptor. Check the SR" (Setup Received) bit in the descriptor to see if the data is for a SETUP or OUT transaction. Also, the host has already switched to Control Write Status phase.	Case E

13.7.6 Application Programming Sequence

This section describes the application programming sequence to take care of normal and abnormal Control transfer scenarios.

All the control transfer cases can be handled by five separate descriptor lists. The descriptor lists are shown in **Figure 13-22**.

- Three lists are for SETUP. The SETUP descriptors also take data for the Status stage of Control Read.
- The first two (index 0 and 1) act in a ping-pong fashion.

- The third list is an empty list, linked to one of the OUT descriptors when premature SETUP comes during the data/ status phase.
- Two lists are for IN and OUT data respectively.
- **Figure 13-22** displays setup_index 0, 1, and 2 as elements of array of pointers called setup_index. The first two elements of this array point to SETUP descriptors. The third element of this array is initially a NULL pointer, but is eventually linked to a SETUP descriptor. These array elements could also point to a descriptor for Control Read Status phase.

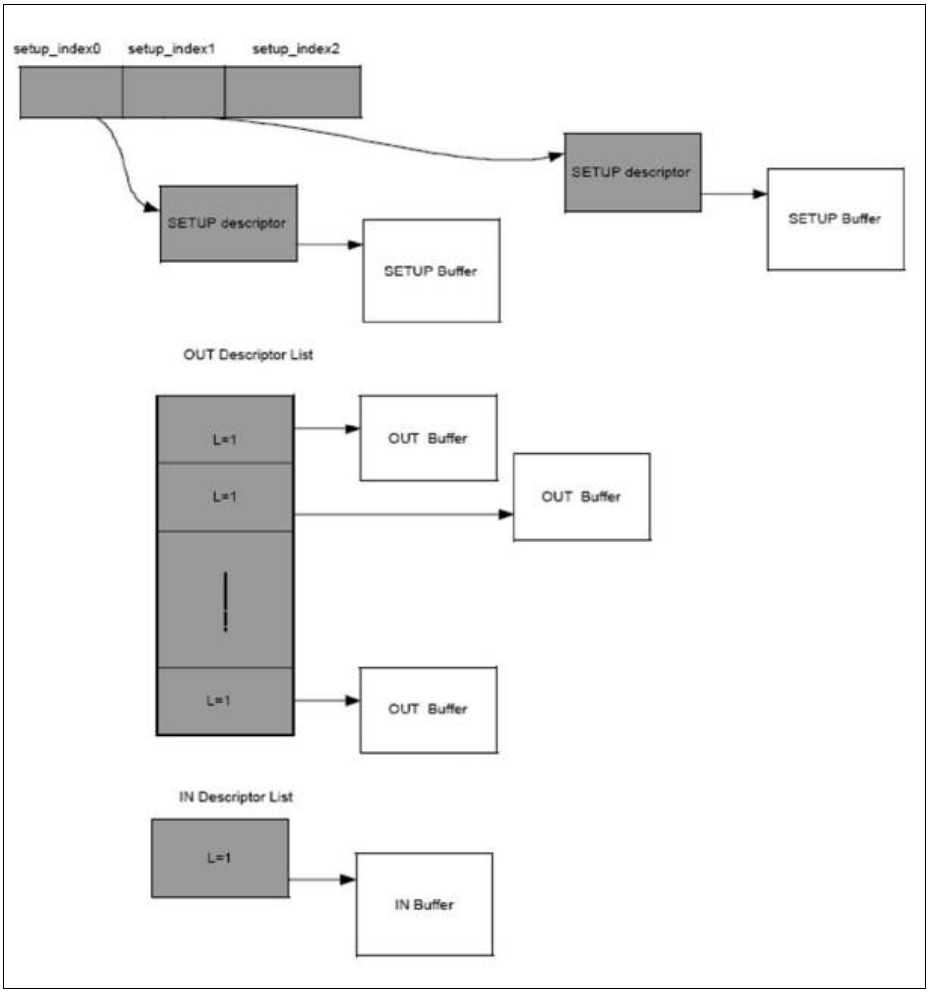


Figure 13-22 Descriptor Lists for Handling Control Transfers

The following are the steps that need to be followed by the application driver.

1. **Set up Desc for SETUP/Ctrl-Rd-Sts** — Setup 2 descriptor lists in memory for taking in SETUP packets. Each of this list must have only one descriptor, with the descriptor fields set to the following
   a) Rx_bytes — Set it to Max packet size of the control endpoint.
   b) IOC =1.
   c) MTRF=0.

d) L=1.

2. **Enable DMA**—If current setup_index =0, then setup_index=1. The application ping-pongs between these two descriptors. Program the address of the current setup descriptor (specified by setup_index) to DOEPDMAx. Write to DOEPCTLx with the following fields.
   a) DOEPCTL.MPS — Max Packet size of the endpoint
   b) DOEPCTL.EPEna — Set to 1 to enable the DMA for the endpoint.
3. **Wait for Interrupt**—Wait for OUT endpoint interrupt (GINTSTS.OEPInt). Then read the corresponding DOEPINT.
4. If Control Read Data Stage in progress
   a) Case A—Check SR bit (In this case SR bit is set, because the host cannot send OUT at this point. If it sends OUT it is NAKed. GOTO Step 24.
   b) Case B —GOTO Step 26.
   c) Case C:-Check SR bit (In this case SR bit is set because host cannot send OUT packets without SETUP at this stage). GOTO Step 24.
   d) Case D — Cannot happen at this stage because SI cannot come alone without a SETUP, at this stage.
   e) Case E — Indicates that host has switched to another SETUP (Three-Stage control write) and then has switched to status phase without and data phase (core clears SUP with SI in this case). Decode SETUP packet and if ok, GOTO Step 11.

   else If Ctrl Write Status Stage in progress OR Two-Stage Status Stage in progress

   f) Case A—Check SR bit (In this case SR bit is set, because the host cannot send OUT at this point. If it sends OUT it is NAKed.) GOTO Step 24.
   g) Case B — (Could happen for Two-Stage Ctrl Transfer.) GOTO Step 26.
   h) Case C—GOTO Step 24.
   i) Case D — Clear SI interrupt and wait Step 3.
   j) Case E — Cannot happen at this stage.

   else

   k) Case A—GOTO Check Desc.
   l) Case B — Normally, this does not occur at this stage. Either IOC comes first or IOC comes with SUP (Case C).
   m)Case C— GOTO Check Desc.
   n) Case D — Cannot happen at this point.
   o) Case E — If SR==1, Indicates Three stage control Transfer SETUP and that the host has switched to status phase. Decode the SETUP packet and Goto Step 11.
   p) Check Desc

   Read the Descriptor status quadlet corresponding to the setup_index and check the SR field. (Application might also want to check the BS and RxSts fields and

take necessary actions if there is any abnormalities). If SR field is 1 GOTO Step 5 (If Step Step 20 is active, terminate it). If SR field is 0 GOTO Step 22 (Control Rd Status phase) (This must also terminate Step 20).

5. Decode SETUP—Decode the SETUP packet. If it is a Three-Stage Control Write, GOTO Step 20. If it is a Three-Stage Control Read, GOTO Step 15. If it is a Two-Stage Control transfer, GOTO Step 11 (Same as Status stage for 3-Stage Control Write).
6. Desc list for Ctrl Wr data— Setup descriptor list for Control write data phase. This must be based on the Wlength field in the SETUP data. The descriptors in the list must be setup such that there must be one descriptor per packet. Each of these descriptors must have the control fields set as follows.
   a) Rx_Bytes — Set to the Max Packet Size of the control Endpoint.
   b) IOC = 1
   c) MTRF = 0.
   d) L=1.
   e) At this point we are not enabling and clearing the NAK for the IN endpoint for status phase.This is because, the status phase for Control Write can be ACKed only after decoding the complete data for the data phase.
   GOTO Step 7.
7. Enable DMA for Ctrl Wr Data—Write the start address of this list to DOEPDMAx. Program the DOEPCTLx with the following bits set
   a) DOEPCTL.MPS — Max Packet size of the endpoint
   b) DOEPCTL.EPEna — Set to 1 to enable the DMA for the endpoint. GOTO Step 8.
8. Wait for Ctrl Wr Data Interrupt—Wait for OUT endpoint interrupt (GINTSTS.OEPInt). Then read the corresponding DOEPINTx.
   a) Case A—check the SR field. Also clear DOEPINTx.XferCompl by writing to DOEPINTx.(Application might also want to check the BS and RxSts fields and take necessary actions if there is any abnormalities). If SR field is 0 GOTO Step 9.If SR field is 1, GOTO Step 23. (This indicates that the host has switched to a new control transfer).
   b) Case B —GOTO Step 25.
   c) Case C—GOTO Step 23. (This indicates that the host has switched to a new control transfer).
   d) Case D — Host has switched to status phase. Decode the data received so far. GOTO Step 10.
   e) Case E — Check SR bit. If SR==0, decode the data received so far. GOTO Step 10. If SR==1, decode the SETUP packet and Goto Step10.
9. Check Desc — If it's not the last packet of data phase, Re-enable the endpoint and clear the Nak. This is because the core sets NAK after receiving each OUT packet for control write data phase. This is to allow application to STALL in case the host sends more data than what is specified in the Wlength field. GOTO Step 8. Re-enabling and clearing the NK involves the following steps.
   a) Write to DOEPDMA with the new descriptor address.

b) Write to DOEPCTLx with the following fields.
- DOEPCTL.MPS — Max Packet size of the endpoint
- DOEPCTL.CNAK—Set to 1 to clear the NAK.
- DOEPCTL.EPEna — Set to 1 to enable the DMA for the endpoint.

If it is the last packet of the data phase, GOTO Step 10.

10. **STALL Extra Bytes**— Write to DOEPCTLx with Stall set so that the core could STALL any further OUT tokens from host.If the received Bytes so far is greater than what is specified in Wlength field OR is there were any unsupported commands in the data phase, then write to DIEPCTLx with the Stall bit set so that the Status phase could be Stalled.(The STALL bit is automatically cleared by the core with the next SETUP). GOTO Step 11.
11. **Disc list for Ctrl Wr Sts**— The following two process must run in parallel. This is because, we are preparing for the status phase (IN) of Control write but at the same time the host could send another SETUP. So IN and OUT descriptor list must be ready.
    a) Do Step 2— Step 5 (This is for handling SETUP or Ctrl Wr Status). If the OUT DMA is already enabled (OUT DMA was enabled for data phase of Three-Stage Control Write, but there was a premature status phase), GOTO Step 3.
    b) Setup descriptor list for Status phase IN, depending on the data in the status phase. Normally it is always a zero length packet.
    c) Tx_Bytes — Size of status phase,
    d) BS — Host Ready,
    e) L=1.
    f) IOC=1.
    g) SP=1 (Depending on the Tx_Bytes).
    h) Write to DIEPDMAx with the start address of the descriptor.Write to DIEPCTLx clear the NAK and enable the endpoint. Flush the corresponding TX FIFO.
    i) If SI has not been received in the data stage prior to the status stage, then wait for SI before clearing the NAK(DIEPCTLx.CNAK=1)
    j) DIEPCTLx.EpEna=1.
    k) GOTO Step 12.
12. **Wait for Interrupt**—Wait for IN endpoint interrupt (GINTSTS.IEPInt).
13. If IN endpoint INterrupt, and DIEPINTx.XferCompl, then GOTO Step 14.
14. **Check Desc** —Read the Status field of the descriptor. Check Tx_bytes in the descriptor.(Application might also want to check the BS and RxSts fields and take necessary actions if there is any abnormalities). This is end of Three-Stage Control Write OR Two-Stage Control transfer. We are now ready for the next control transfer (Already taken care by process "a" is Step 11.
15. **Desc for Ctrl Rd Data**—The following two steps must be run in parallel. This is because, we are preparing for Data phase of Control read, but at the same time, the host could abnormally abort this control transfer and send a SETUP, or switch to status phase.

16. Do Step 2— Step 5 (This is for handling SETUP and also Control Read Status phase.).
17. Setup descriptor list for Data phase IN, depending on the WLength field in the SETUP data. The setup can be for a single descriptor OR multiple descriptors. If it is multiple descriptors, ensure that IOC for the last descriptor is set.
    a) Tx_Bytes — Size of data phase (Wlength field).
    b) BS — Host Ready
    c) L=1.
    d) IOC=1. It is mandatory to set the IOC when it is the last descriptor.
    e) SP=1 (Depending on the Tx_Bytes).
    f) Write to DIEPDMAx with the start address of the descriptor list.
    g) Write to DIEPCTLx clear the NAK and enable the endpoint.
    h) Flush the corresponding TX FIFO.
    i) DIEPCTLx.MPS = Max_packet size of the endpoint,
    j) DIEPCTLx.CNAK=1 only if SPD already set (Case C in Step 3).
    k) Also set the DOEPCTLx.CNAK for the corresponding OUT endpoint after SPD because a premature status stage (OUT) can come which must be acked.
    l) DIEPCTLx.EpEna=1.
    m)GOTO Step 18.
18. **Wait for Interrupt**—Wait for IN endpoint interrupt (GINTSTS.IEPInt)
19. If IN endpoint interrupt, read the corresponding DIEPINTx and if XferCompl is set GOTO Step 20.
20. **Check_Desc**—Wait for the DIEPINTx.IOC interrupt. Go to Step 21.
21. **Set_Stall**—Write to DIEPCTLx with STALL bit set. (The STALL bit is automatically cleared by the core with the next SETUP). The function of this process initiated in step Step 15 is over, and must be terminated. The next control transfer is already taken care by the process that is running from Step 2.
22. **Ctrl Rd Sts Desc Check** — Read the descriptor to check the Rxbytes and also check the SP field. The Three-Stage control Read is complete here. GOTO Step 2, in preparation for the next SETUP.
23. The unexpected SETUP packet now received during the control write data phase, is sitting in the descriptor allocated for Data. Link this to the setup descriptor pointer. setup_desc_index = 2. Point setup_desc_index to the current OUT descriptor (which has the SETUP). GOTO Step 5.
24. Disable IN Endpoint DMA. Core flushes the corresponding Tx FIFO in order to flush the data that was meant for Control Write Status phase OR Control Read data phase. If Step 12 or Step 18 is active, terminate it. GOTO Step .
25. Read Modify write DOEPCTLx to clear the NAK. Then GOTO Step 8 again.
    a) DOEPCTLx.CNAK—Set to 1 to clear the NAK.
26. Read Modify write DIEPCTLx to clear the NAK. Then GOTO Step 3 again.
    a) DIEPCTLx.CNAK—Set to 1 to clear the NAK.
27. Read Modify write DIEPCTLx to clear the NAK. Then Step 12 again
    a) DIEPCTLx.CNAK—Set to 1 to clear the NAK.

b) DOEPCTLx.CNAK:Set to 1 to clear the NAK for the out endpoint. This clears the NAK to accept status stage data in case of control read.

13.7.7 Internal Data Flow

This section explains the cores internal data flow for control transfers.

13.7.7.1 Three-Stage Control Write

Figure 13-23 displays the core behavior for Three-Stage control write transfers.

1. On receiving SETUP, the data is pushed into the Rx FIFO and the core sets NAK on both IN and OUT endpoint of that control endpoint. Additionally, the clearing of the NAK bit is blocked by the core until the following SPD or SI is read by the application and cleared.
2. The DMA detects the RxFIFO as non-empty and does the following:
   a) Fetch the descriptor pointed by DOEPMA.
   b) Transfer the SETUP packet from the RxFIFO to the buffer pointed by the descriptor.
   c) Close the descriptor with DMA_DONE status.
3. On receiving the first data phase OUT token after the SETUP, the core push a SETUP_COMPLETE status into the RxFIFO.Core NAKs the data phase OUT tokens because of the NAK set on receiving the SETUP packet.
4. The core generates DOEPINT.XferCompl interrupt after having transferred the SETUP packet into memory (Step 2).
5. The core generates DOEPINT.SetUp interrupt after the DMA has popped the SETUP_COMPLETE status out of the RxFIFO.
6. Application clears NAK for the data phase, after receiving DOEPINTx.SetUp interrupt.
7. The core ACKs and the next OUT token because the NAK has been cleared (provided there is enough space in the RxFIFO.
8. DMA detects the OUT packet in RxFIFO and starts transferring the OUT packet to the system memory.
   a) Fetch the descriptor pointed by DOEPMA.
   b) Transfer the OUT packet from the RxFIFO to the buffer pointed by the descriptor.
   c) Close descriptor with DMA_DONE status.
9. The core NAKs the next OUT token because the core internally sets the NAK after every control write data phase packets. This is to allow application to Stall any extra tokens.
10. The core generates DOEPINT.XferCompl after closing the OUT descriptor (Step 8).
11. Application clear NAK on receiving DOEPINTx.XferCompl interrupt.
12. Host starts the Status phase by sending the IN token which is NAKed by the core. The core push DATA_PHASE_DONE status into the RxFIFO.

13. The core generates DOEPINTx.XferCompl for the last OUT packet transfer to system memory.
14. The core generates DOEPINT.StsPhsRcvd interrupt after the DMA has popped the DATA_PHASE_DONE status from the RxFIFO.
15. Application clears the NAK and enables the IN endpoint for status phase.
16. The core starts fetching the data for the Status phase
    a) Fetch the descriptor pointed by DIEPDMA.
    b) Fetch the packet (if size >0) to Tx fifo.
    c) Close the descriptor with DMA_DONE status
    d) The core generates DIEPINTx.XferCompl interrupt after closing the descriptor.
17. The core sends out data in response to the Status Phase IN token.

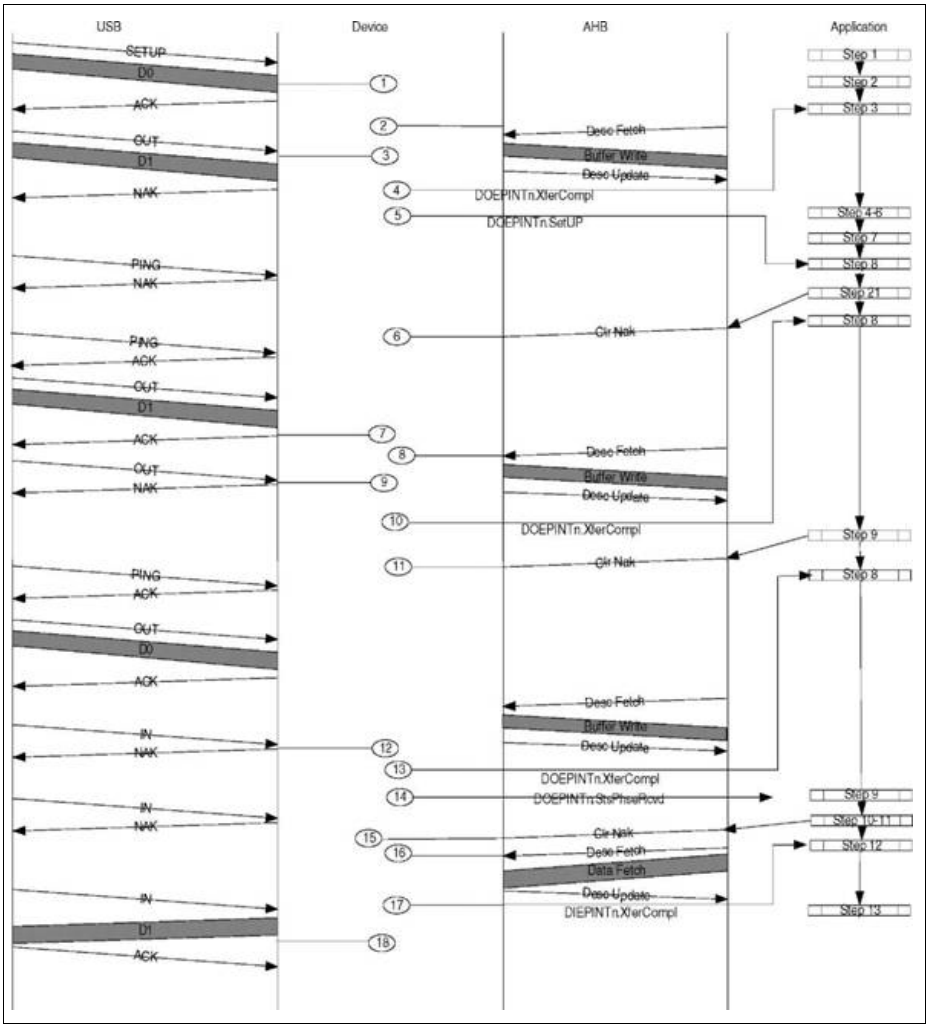


Figure 13-23 Three-Stage Control Write

13.7.7.2 Three-Stage Control Read

Figure 13-24 displays the core flow for three-stage control read transfers

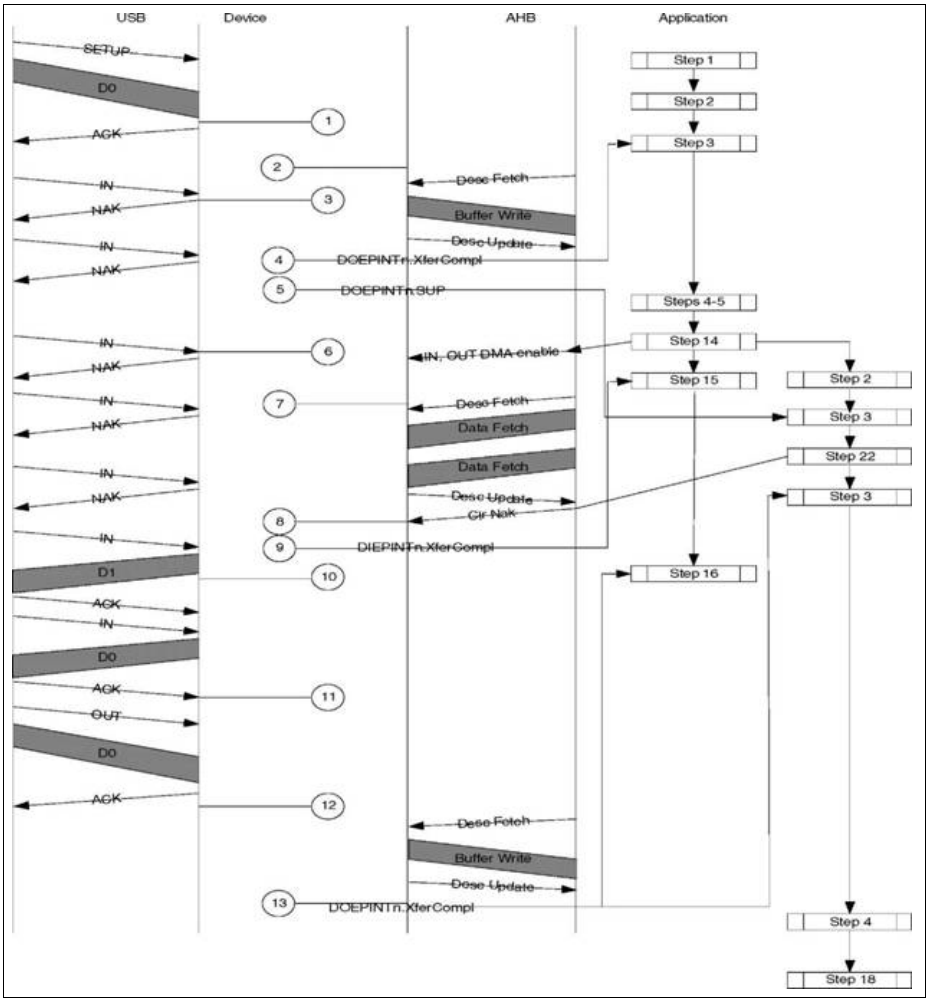


Figure 13-24 Three-Stage Control Read

In this example, it is assumed that the data phase consists of 2 packets, and the application allocates these two packets in a single buffer.

1. On receiving SETUP, the data is pushed into the Rx FIFO and the core sets NAK on both IN and OUT endpoint of that control endpoint.

2. The DMA detects the RxFIFO as non-empty and does the following.
   a) Fetch the descriptor pointed by DOEPMA.
   b) Transfer the SETUP packet from the RxFIFO to the buffer pointed by the descriptor.
   c) Close the descriptor with DMA_DONE status.
3. On receiving the first data phase OUT token after the SETUP, the core push a SETUP_COMPLETE status into the RxFIFO.Core NAKs the data phase OUT tokens because of the NAK set on receiving the SETUP packet.
4. The core generates DOEPINT.XferCompl interrupt after having transferred the SETUP packet into memory (Step 2).
5. The core generates DOEPINT.SetUp interrupt after the DMA has popped the SETUP_COMPLETE status out of the RxFIFO.
6. Data phase IN tokens are NAKed until this point because the NAK has not yet been cleared by the application.
7. The core starts fetching the IN data after the application enables IN DMA (In this example it is assumed that multiple packets are in the same buffer. But it could also be in different buffers). This involves the following steps
   a) Fetch the descriptor pointed by DIEPDMA.
   b) Fetch the data into the corresponding Tx FIFO.
   c) Close the descriptor with DMA_DONE status...
8. The application clears the NAK after receiving the setup complete (DOEPINT.SetUp) interrupt. The application also clears NAK of the OUT End point to accept the status phase.
9. After all the data has been fetched for the descriptor (Step 7), core generates DIEPINT.XferCompl interrupt.
10. The core sends data in response to the IN token for the data phase.
11. The core sends out the last packet of the IN data phase.
12. The core ACKs the status phase.
13. The core generates DOEPINTx.XferCompl interrupt after transferring the data received for the status phase to system memory.

13.7.7.3 Two-Stage Control Transfer

Figure 13-25 displays the core behavior for Two Stage control read transfers.

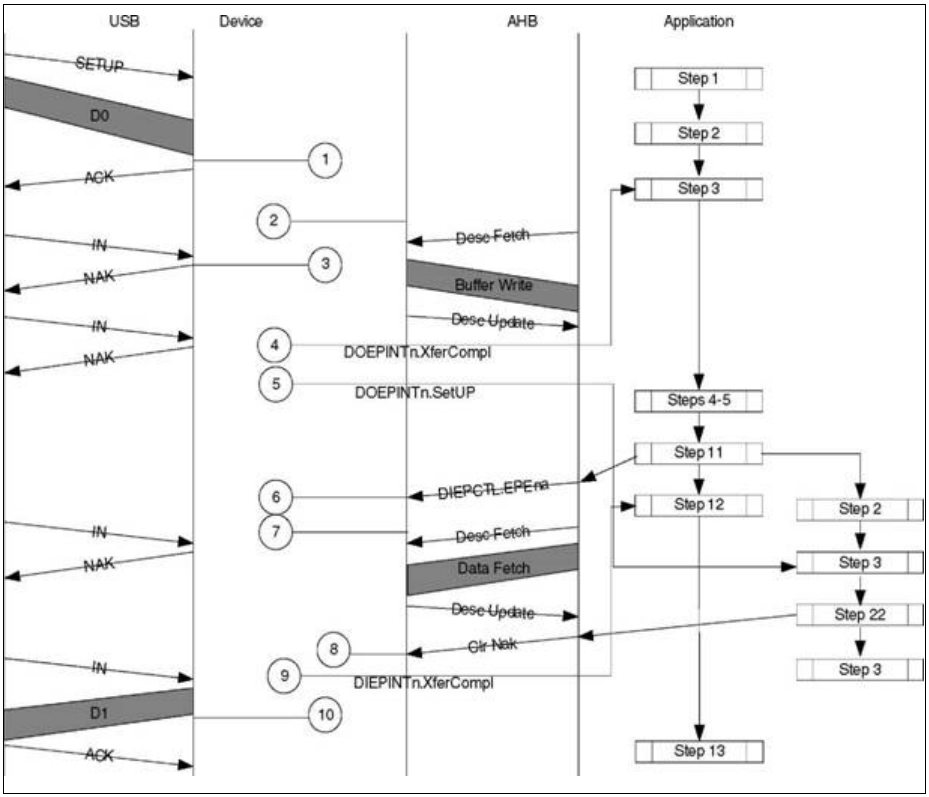


Figure 13-25 Two-Stage Control Transfer

This example shows the core behavior for a Two-Stage Control transfer.

1. On receiving SETUP, the data is pushed into the Rx FIFO and the core sets NAK on both IN and OUT endpoint of that control endpoint.
2. The DMA detects the RxFIFO as non-empty and does the following.
   a) Fetch the descriptor pointed by DOEPMA.
   b) Transfer the SETUP packet from the RxFIFO to the buffer pointed by the descriptor.
   c) Close the descriptor with DMA_DONE status.
   d) The core receives the status phase IN token, which it NAKs. Core also pushes SETUP_COMPLETE status into the Rx FIFO.

e) The core generates DOEPINT.XferCompl interrupt after having transferred the SETUP packet into memory (Step 2)
f) The core generates DOEPINT.SetUp interrupt after the DMA has popped the SETUP_COMPLETE status out of the RxFIFO.

3. Application enables the IN endpoint for status phase.
4. The core starts fetching the descriptor for IN endpoint.
5. Application clears the NAK for IN endpoint after getting the DOEPINTx.SetUP interrupt (Step 4).
6. The core generates DIEPINTx.XferCompl after updating the descriptor after IN data fetch.
7. The core sends out data for the status phase IN token from host.

13.7.7.4 Back to Back SETUP During Control Write

This example shows the core receiving 2 Back to Back SETUP tokens fo3 Three-Stage Control write.

1. On receiving SETUP, the data is pushed into the Rx FIFO and the core sets NAK on both IN and OUT endpoint of that control endpoint.
2. The DMA detects the RxFIFO as non-empty and does the following.
   a) Fetch the descriptor pointed by DOEPMA.
   b) Transfer the SETUP packet from the RxFIFO to the buffer pointed by the descriptor.
   c) Close the descriptor with DMA_DONE status.
   d) The core receives another SETUP, and pushes the data into the Rx FIFO, also sets the NAK.
   e) The core generates DOEPINT.XferCompl interrupt after having transferred the SETUP packet into memory (Step 2).
3. On receiving the first data phase OUT token after the SETUP, the core push a SETUP_COMPLETE status into the RxFIFO.Core NAKs the data phase OUT tokens because of the NAK set on receiving the SETUP packet.
4. The DMA detects the RxFIFO as non-empty (because of the 2nd SETUP packet) and does the following.
   a) Fetch the descriptor pointed by DOEPMA.
   b) Transfer the SETUP packet from the RxFIFO to the buffer pointed by the descriptor.
   c) Close the descriptor with DMA_DONE status.
   d) The core generates DOEPINT.XferCompl interrupt after having transferred the second SETUP packet into memory (Step 6)
   e) The core generates DOEPINT.SetUp interrupt after the DMA has popped the SETUP_COMPLETE status out of the RxFIFO.
5. Application clears NAK for the data phase, after receiving DOEPINTx.SetUp interrupt.

6. The core ACKs the next OUT/Ping token after the NAK has been cleared by the application.
7. The DMA detects the RxFIFO as non-empty (because of the OUT packet) and does the following.
   a) Fetch the descriptor pointed by DOEPMA.
   b) Transfer the SETUP packet from the RxFIFO to the buffer pointed by the descriptor.
   c) Close the descriptor with DMA_DONE status.
   d) The core generates DOEPINT.XferCompl interrupt after having transferred the OUT packet into memory (Step 11) and closing the descriptor.

The remaining steps are similar to Steps 11-18 of **“Application Programming Sequence” on Page 13-102**.

This example shows the core behavior for a Two-Stage Control transfer.

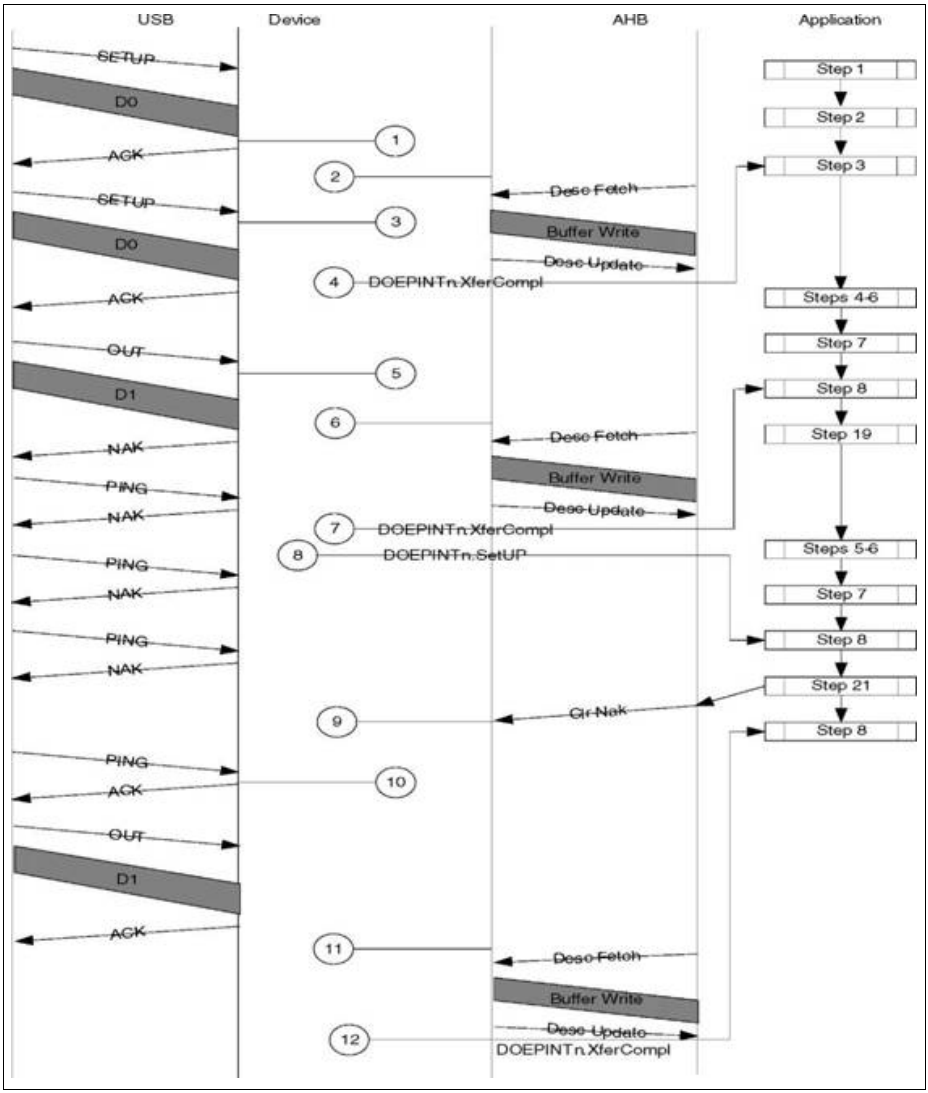


Figure 13-26 Back-to-Back SETUP Packet Handling During Control Write

13.7.7.5 Back-to-Back SETUPs During Control Read

1. On receiving SETUP, the data is pushed into the Rx FIFO and the core sets NAK on both IN and OUT endpoint of that control endpoint.
2. The DMA detects the RxFIFO as non-empty and does the following.
   a) Fetch the descriptor pointed by DOEPMA.
   b) Transfer the SETUP packet from the RxFIFO to the buffer pointed by the descriptor.
   c) Close the descriptor with DMA_DONE status.
   d) The core receives another SETUP, and pushes the data into the Rx FIFO, also sets the NAK.
3. The core generates DOEPINT.XferCompl interrupt after having transferred the SETUP packet into memory (Step 2).
4. Host sends IN token for the data phase which is NAKed by the core, because NAK is set in Setp3. The core pushes SETUP_COMPLETE status into RxFIFO.
5. After the application has re-enabled the OUT DMA (Application flow Step 2) core detects RxFIFO as non-empty because of the second SETUP packet and does the following.
   a) Fetch the descriptor pointed by DOEPMA.
   b) Transfer the SETUP packet from the RxFIFO to the buffer pointed by the descriptor.
   c) Close the descriptor with DMA_DONE status.
   d) The core generates DOEPINTx.XferCompl interrupt after having transferred the SETUP packet into memory (Step6).
   e) The core starts fetching data for IN endpoint because the IN endpoint was enabled by application in Step-14.
6. On seeing DOEPINTx.XferCompl (Step 7) and finding that it is a SETUP packet, application disables the endpoint in Step 20.
7. The core generates DOEPINTx.SetUP (Setup complete) interrupt after popping the SETUP_COMPLETE status from the RxFIFO.
8. The core generates endpoint disabled interrupt (as a result of application setting disable bit in step 9)
9. The core generates DIEPINTx.XferCompl after completing the IN data fetch and updating the descriptor.
10. application clears NAK after seeing setup_complete interrupt (generated in Step 10. The flow after this is same as steps 9 - 13 of **“Internal Data Flow” on Page 13-109**

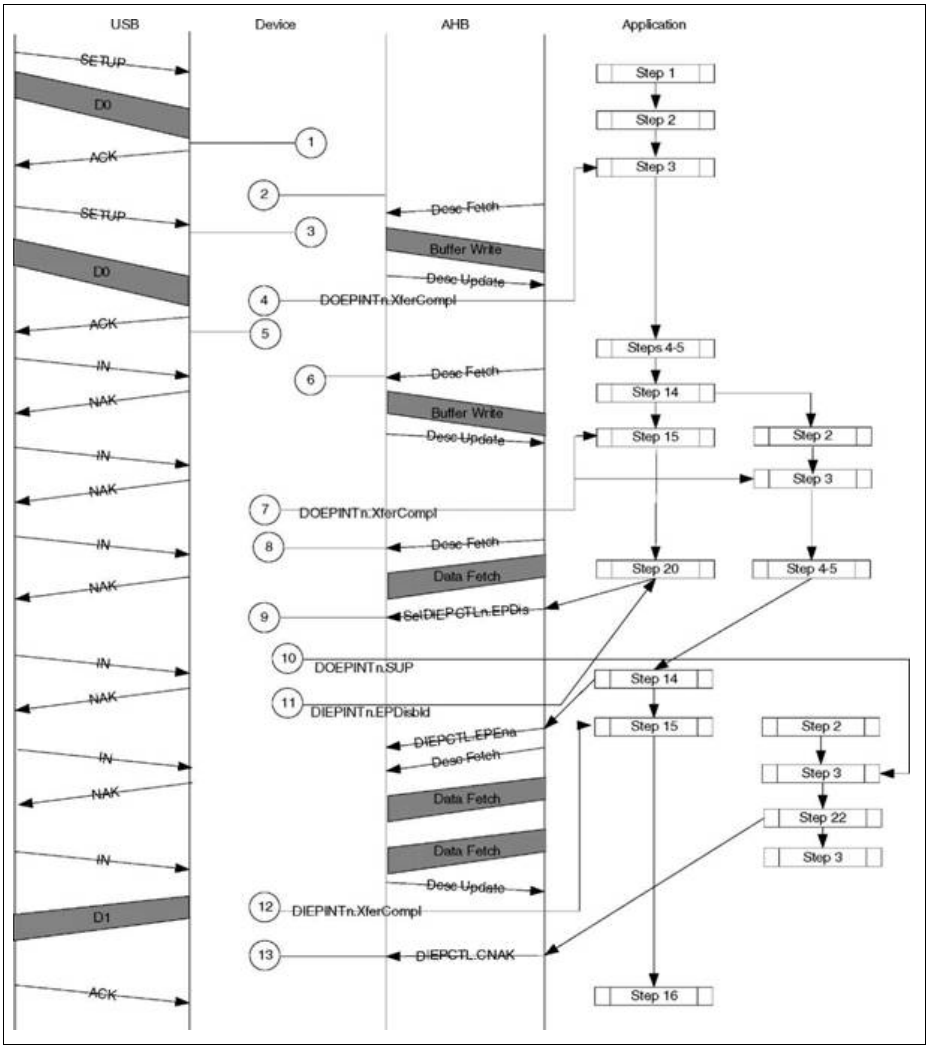


Figure 13-27 Back-to-Back SETUP During Control Read

13.7.7.6 Extra Tokens During Control Write Data Phase

This example assumes a three-stage control write transfer with only Wlength field in the SETUP indicating only 1 packet in the data phase. But the host sends an additional OUT packets which the core STALLs.

1. On receiving SETUP, the data is pushed into the Rx FIFO and the core sets NAK on both IN and OUT endpoint of that control endpoint.
2. The DMA detects the RxFIFO as non-empty and does the following.
   a) Fetch the descriptor pointed by DOEPMA.
   b) Transfer the SETUP packet from the RxFIFO to the buffer pointed by the descriptor.
   c) Close the descriptor with DMA_DONE status.
3. On receiving the first data phase OUT token after the SETUP, the core push a SETUP_COMPLETE status into the RxFIFO.Core NAKs the data phase OUT tokens because of the NAK set on receiving the SETUP packet.
4. The core generates DOEPINT.XferCompl interrupt after having transferred the SETUP packet into memory (Step 2).
5. The core generates DOEPINT.SetUp interrupt after the DMA has popped the SETUP_COMPLETE status out of the RxFIFO.
6. Application clears NAK for the data phase, after receiving DOEPINTx.SetUp interrupt.
7. The core ACKs and the next OUT token because the NAK has been cleared (provided there is enough space in the RxFIFO.
8. DMA starts transferring the OUT packet to the system memory.
   a) Fetch the descriptor pointed by DOEPMA.
   b) Transfer the OUT packet from the RxFIFO to the buffer pointed by the descriptor.
   c) Close descriptor with DMA_DONE status.
   d) The core generates DOEPINTx.XferCompl interrupt after having transferred the OUT packet to the system memory. Since there were only one packet in the data phase, the data phase is complete here.
   e) The core initially NAK's the extra tokens send by the host, because the core internally sets NAK after each OUT packet for the data phase of control write.
9. Application sets STALL to stall any extra tokens.
10. The core stalls the next OUT/PING token.
11. Host switches to next control transfer, core ACKs the SETUP. This SETUP packet is transferred to the system memory buffer originally allocated for Status phase.
12. The core generates DOEPINTx.XferCompl interrupt after transferring the SETUP packet to the system memory.

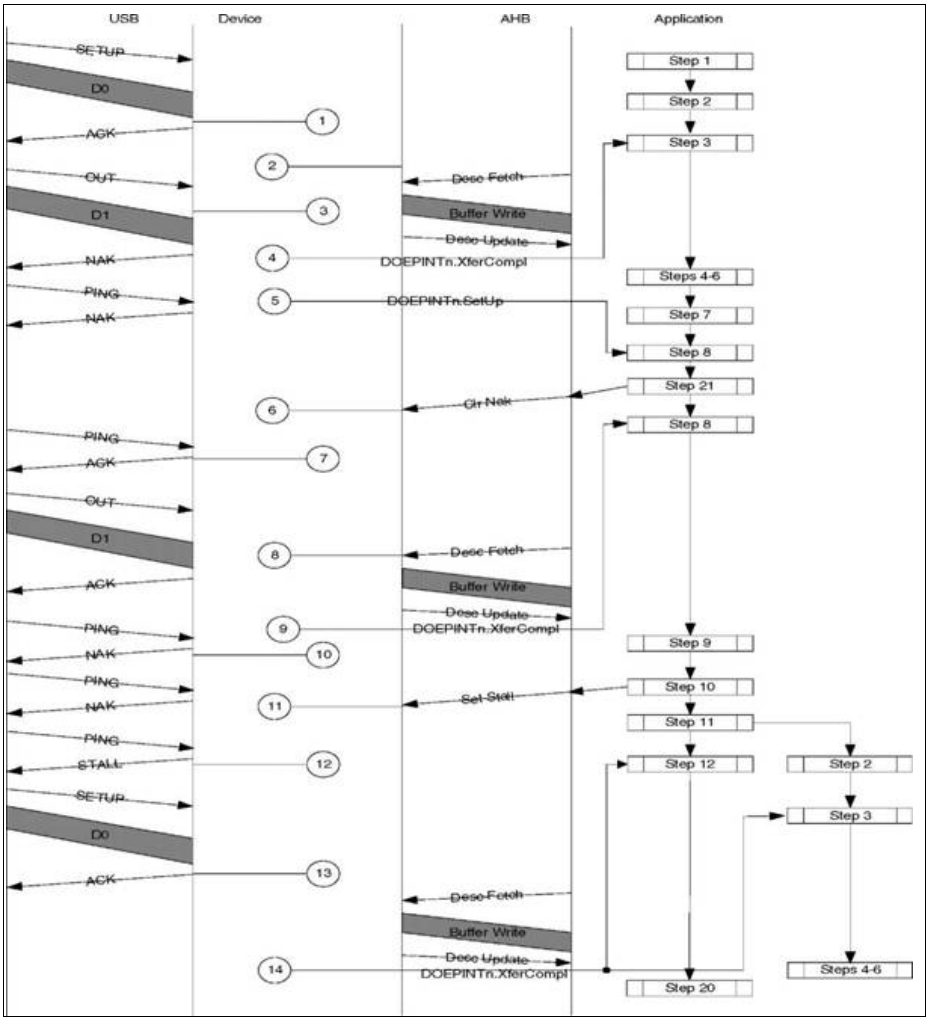


Figure 13-28 Extra Tokens During Control Write Data Phase

13.7.7.7 Extra Tokens During Control Read Data Phase

In this example, it is assumed that the data phase consists of 2 packets, and the application allocates these two packets in a single buffer. After the data phase is complete and the two packets have been transferred, the core sends an extra IN token and then the application sets Stall.

1. On receiving SETUP, the data is pushed into the Rx FIFO and the core sets NAK on both IN and OUT endpoint of that control endpoint.
2. The DMA detects the RxFIFO as non-empty and does the following.
   a) Fetch the descriptor pointed by DOEPMA.
   b) Transfer the SETUP packet from the RxFIFO to the buffer pointed by the descriptor.
   c) Close the descriptor with DMA_DONE status.
3. On receiving the first data phase OUT token after the SETUP, the core push a SETUP_COMPLETE status into the RxFIFO.Core NAKs the data phase OUT tokens because of the NAK set on receiving the SETUP packet.
4. The core generates DOEPINT.XferCompl interrupt after having transferred the SETUP packet into memory (Step 2).
5. The core generates DOEPINT.SetUp interrupt after the DMA has popped the SETUP_COMPLETE status out of the RxFIFO.
6. Data phase IN tokens are NAKed until this point because the NAK has not yet been cleared by the application.
7. The core starts fetching the IN data after the application enables IN DMA (In this example it is assumed that multiple packets are in the same buffer. But it could also be in different buffers). This involves the following steps
   a) Fetch the descriptor pointed by DIEPDMA.
   b) Fetch the data into the corresponding Tx FIFO.
   c) Close the descriptor with DMA_DONE status.
8. The application clear the NAK after receiving the setup complete (DOEPINT.SetUp) interrupt. Set the Stall bit after all the Data has been pushed in the FIFO
9. After all the data has been fetched for the descriptor (Step 7), core generates DIEPINT.XferCompl interrupt.
10. The core sends data in response to the IN token for the data phase.
11. The core sends out the last packet of the IN data phase.
12. Host sends an extra token.
13. The core Stalls the IN token and also automatically Stalls the Status phase if the Host switches to the Status phase.

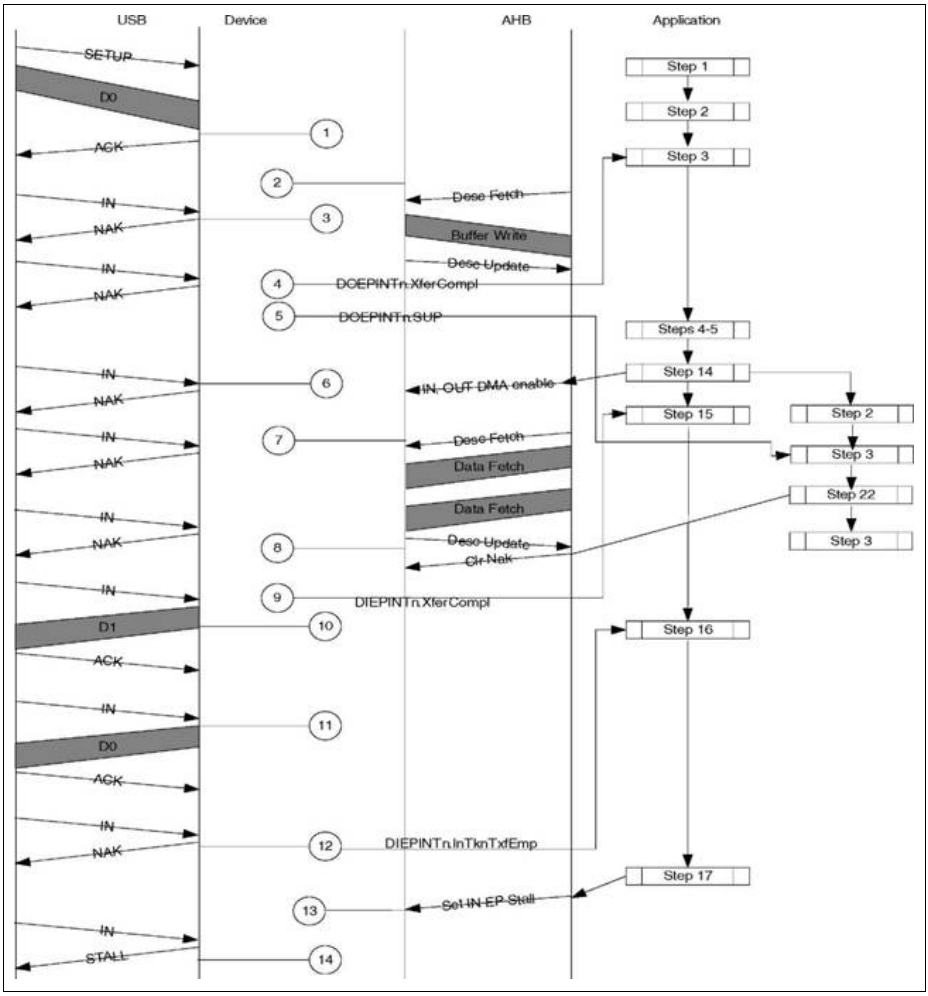


Figure 13-29 Extra IN Tokens During Control Read Data Phase

13.7.7.8 Premature SETUP During Control Write Data Phase

This example shows a Three-Stage Control Write transfer with host sending a premature Control Write SETUP packet during the data phase.

1. On receiving SETUP, the data is pushed into the Rx FIFO and the core sets NAK on both IN and OUT endpoint of that control endpoint.
2. The DMA detects the RxFIFO as non-empty and does the following.
   a) Fetch the descriptor pointed by DOEPMA.
   b) Transfer the SETUP packet from the RxFIFO to the buffer pointed by the descriptor.
   c) Close the descriptor with DMA_DONE status.
3. On receiving the first data phase OUT token after the SETUP, the core push a SETUP_COMPLETE status into the RxFIFO.Core NAKs the data phase OUT tokens because of the NAK set on receiving the SETUP packet.
4. The core generates DOEPINT.XferCompl interrupt after having transferred the SETUP packet into memory (Step 2).
5. The core generates DOEPINT.SetUp interrupt after the DMA has popped the SETUP_COMPLETE status out of the RxFIFO.
6. The core receives a SETUP packet during the data phase. This is an unexpected SETUP packet.On receiving this SETUP, the SETUP data is pushed into the RxFIFO and the core again sets NAK on both IN and OUT endpoints of the control endpoint (NAK was already set because of the first SETUP packet received).
7. Application decodes the previous DOEPINT.SetUp interrupt and clears the NAK, unaware of the fact that there is another SETUP packet sitting in the RxFIFO for the same control endpoint. On seeing this condition, core does not allow clearing of the NAK bit, and masks the clearing of NAK. The core takes this decision based on the fact that a SETUP_COMPLETE status is pending in the RXFIFO.
8. The DMA detects the RxFIFO as non-empty (because of the unexpected SETUP) and does following
   a) Fetch the descriptor pointed by DOEPMA.
   b) Transfer the SETUP packet from the RxFIFO to the buffer pointed by the descriptor.
   c) Close the descriptor with DMA_DONE status.
   d) The core NAKs the data phase OUT token because NAK bit clearing by the application did not take effect (as explained in Step 7).
   e) The core generates DOEPINT.XferCompl interrupt after having transferred the SETUP packet into memory (Step 8).
   f) The core generates DOEPINT.SetUp interrupt after the DMA has popped the SETUP_COMPLETE status (for the unexpected SETUP packet received) out of the RxFIFO.
9. Application clears the NAK after decoding the latest SETUP packet. This time, the core does not mask the clearing of the NAK because there are no more SETUP_COMPLETE status sitting in the RxFIFO.

10. The core ACKs the next OUT/PING token of the data phase.
11. DMA starts transferring the OUT packet to the system memory.
    a) Fetch the descriptor pointed by DOEPMA.
    b) Transfer the OUT packet from the RxFIFO to the buffer pointed by the descriptor.
    c) Close descriptor with DMA_DONE status.
    d) The core generates DOEPINTx.XferCompl interrupt after having transferred the OUT packet to the system memory.
    e) The remaining steps are similar to Steps 11-18 of **“Application Programming Sequence” on Page 13-102**

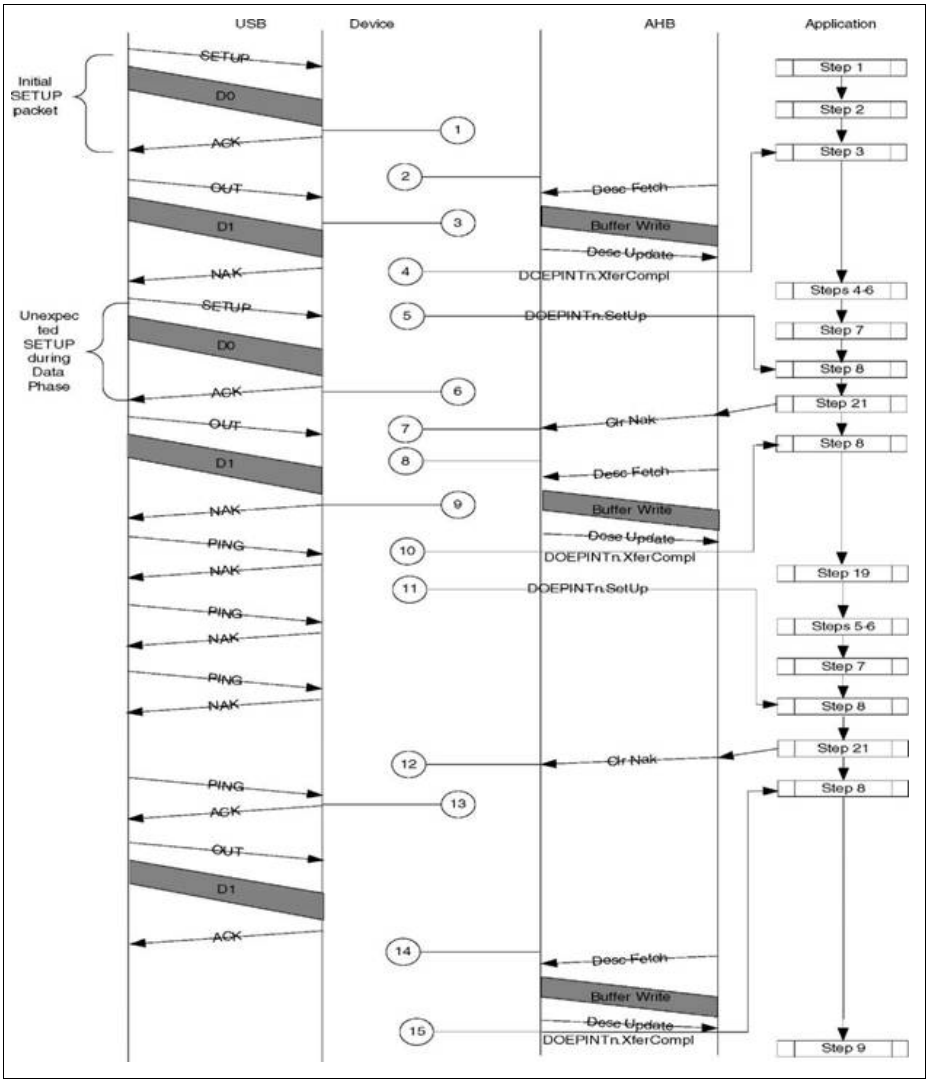


Figure 13-30 Premature SETUP During Control Write Data Phase

13.7.7.9 Premature SETUP During Control Read Data Phase

In this example, it is assumed that the data phase consists of 2 packets, and the application allocates these two packets in a single buffer. The host switches to a new control read command after having send two IN tokens during the data phase.

1. On receiving SETUP, the data is pushed into the Rx FIFO and the core sets NAK on both IN and OUT endpoint of that control endpoint.
2. The DMA detects the RxFIFO as non-empty and does the following.
   a) Fetch the descriptor pointed by DOEPMA.
   b) Transfer the SETUP packet from the RxFIFO to the buffer pointed by the descriptor.
   c) Close the descriptor with DMA_DONE status.
3. On receiving the first data phase IN token after the SETUP, the core push a SETUP_COMPLETE status into the RxFIFO.Core NAKs the data phase IN tokens because of the NAK set on receiving the SETUP packet.
4. The core generates DOEPINT.XferCompl interrupt after having transferred the SETUP packet into memory (Step 2).
5. The core generates DOEPINT.SetUp interrupt after the DMA has popped the SETUP_COMPLETE status out of the RxFIFO.
6. Host switches to a new Control transfer by sending a SETUP token. This is the premature SETUP packet. Core sets NAK on both IN and OUT control endpoints.
7. The core fetch the data for IN control endpoint after application enables the IN endpoint.
8. The core push SETUP_COMPLETE status into Rx FIFO on seeing the IN token for data phase.
9. Application clears NAK as a result of DOEPINT.SetUP (Setup complete) interrupt generated in Step 5. But core masks this clearing of setup_complete interrupt, because there is already one SETUP packet sitting in the Rx FIFO.
10. The core generates DIEPINTx.XFERCompl after closing the IN endpoint descriptor (for Step 7)
11. The core generates DOEPINTx.XferCompl after transferring the premature SETUP packet to system memory and closing the descriptor.
12. The core generates SETUP complete interrupt.
13. Application enables IN endpoint DMA for data phase.
14. The core fetches descriptor and data for IN endpoint.
15. Application clears IN endpoint NAK after receiving DOEPINTx.SetUP (Setup complete) interrupt. This time, the core does not mask the clearing of the Nak because NO SETUP packet is remaining in the Rx FIFO.
16. The core generates DIEPINTx.XferCompl interrupt after fetching the data and closing the descriptor. The remaining steps are same as steps 11 to 13 of **“Internal Data Flow” on Page 13-109**.

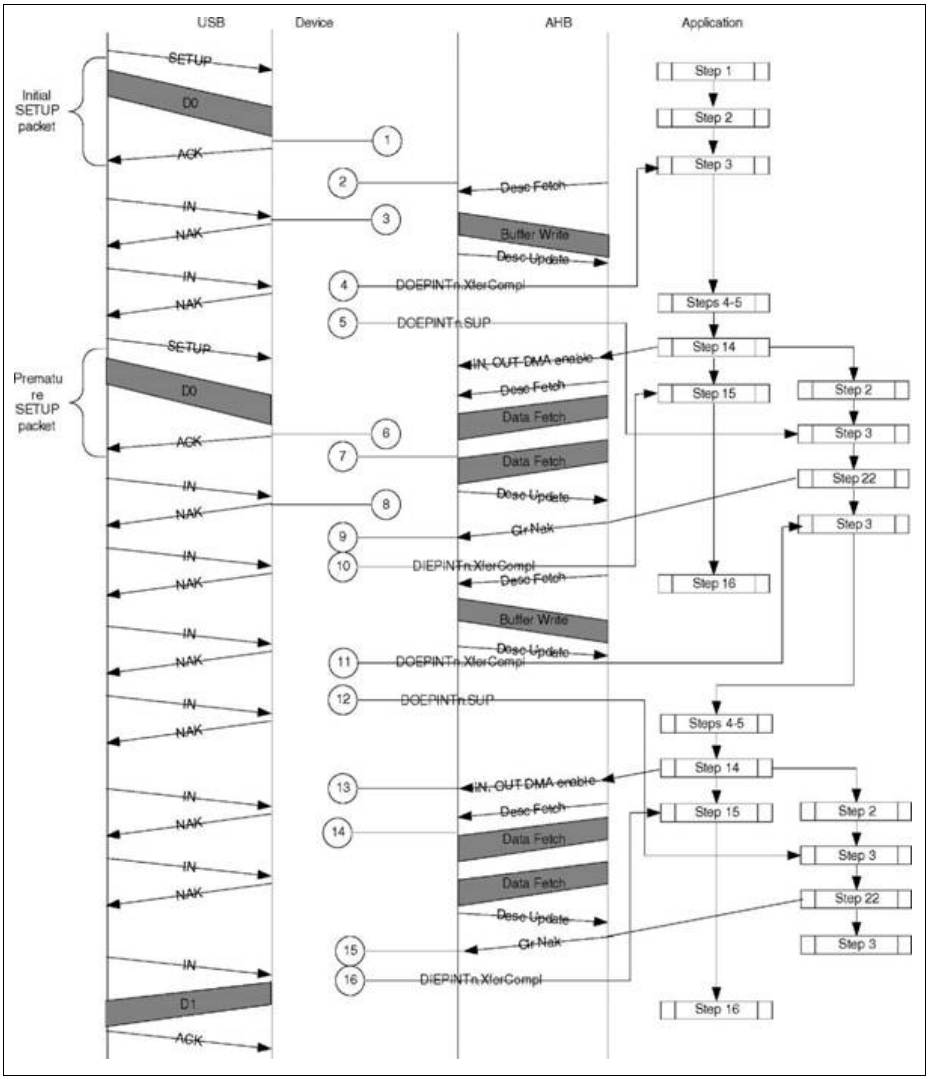


Figure 13-31 Premature SETUP During Control Read Data Phase

13.7.7.10 Premature Status During Control Write

This example assumes a Three-Stage control write transfer with only Wlength field in the SETUP indicating two packets in the data phase. But the host switch to data phase after the first packet of the data phase is complete.

1. On receiving SETUP, the data is pushed into the Rx FIFO and the core sets NAK on both IN and OUT endpoint of that control endpoint.
2. The DMA detects the RxFIFO as non-empty and does the following.
   a) Fetch the descriptor pointed by DOEPMA.
   b) Transfer the SETUP packet from the RxFIFO to the buffer pointed by the descriptor.
   c) Close the descriptor with DMA_DONE status.
3. On receiving the first data phase OUT token after the SETUP, the core push a SETUP_COMPLETE status into the RxFIFO.Core NAKs the data phase OUT tokens because of the NAK set on receiving the SETUP packet.
4. The core generates DOEPINT.XferCompl interrupt after having transferred the SETUP packet into memory (Step 2).
5. The core generates DOEPINT.SetUp interrupt after the DMA has popped the SETUP_COMPLETE status out of the RxFIFO.
6. Application clears NAK for the data phase, after receiving DOEPINTx.SetUp interrupt (Step 5).
7. The core ACKs and the next OUT token because the NAK has been cleared (provided there is enough space in the RxFIFO.
8. DMA sees TxFIFO non empty and starts transferring the OUT packet to the system memory.
   a) Fetch the descriptor pointed by DOEPMA.
   b) Transfer the OUT packet from the RxFIFO to the buffer pointed by the descriptor.
   c) Close descriptor with DMA_DONE status.
9. Host switch to status phase (IN token) without completing the data phase.
10. The core generates DOEPINTx.XferComp after closing the descriptor after the data fetch.
11. Application sets up descriptor, enables IN endpoint and clear NAK.
12. The core starts to fetch the descriptor and data for the status phase once application has enabled the IN endpoint.
13. The core generates DIEPINTx.XferCompl after doing the data fetch and the descriptor update (step 12)
14. The core sends data out in response to status phase IN token.

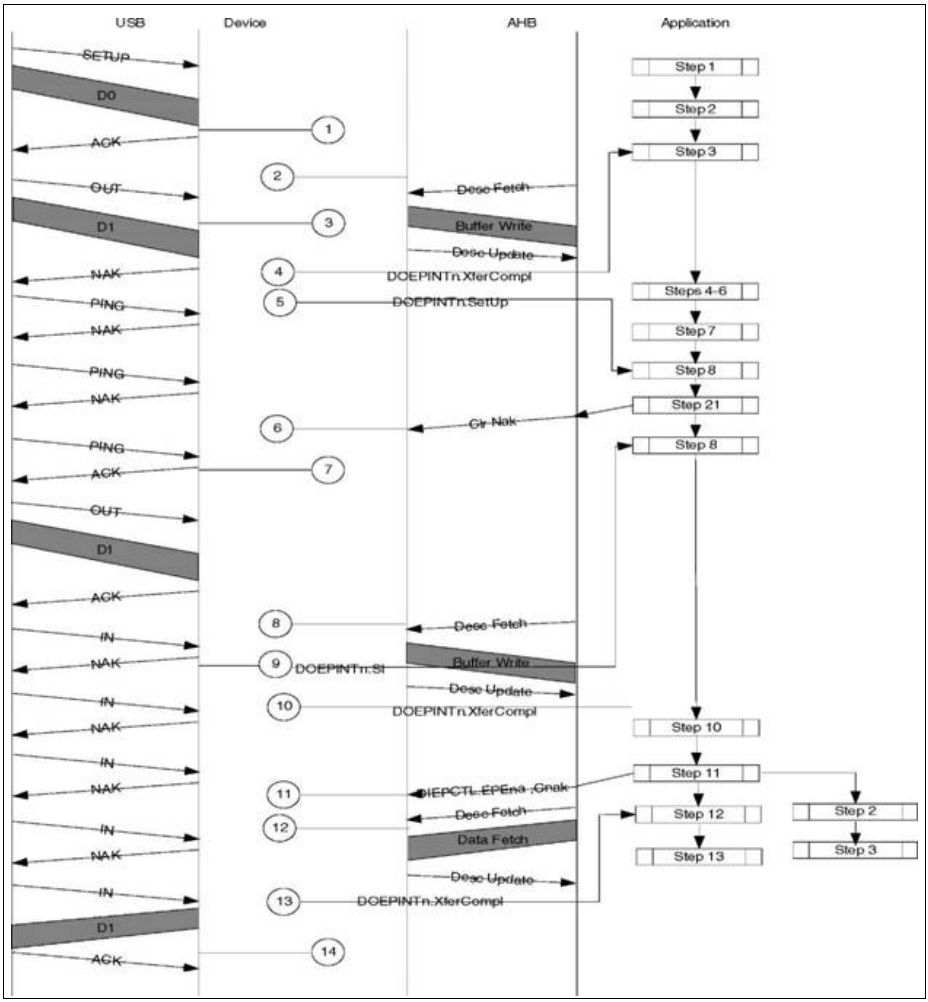


Figure 13-32 Premature Status Phase During Control Write

13.7.7.11 Premature Status During Control Read

In this example, it is assumed that the data phase consists of two packets, and the application allocates these two packets in a single buffer. After one packet in the data phase, host switches to status phase.

1. On receiving SETUP, the data is pushed into the Rx FIFO and the core sets NAK on both IN and OUT endpoint of that control endpoint.
2. The DMA detects the RxFIFO as non-empty and does the following.
   a) Fetch the descriptor pointed by DOEPMA.
   b) Transfer the SETUP packet from the RxFIFO to the buffer pointed by the descriptor.
   c) Close the descriptor with DMA_DONE status.
3. On receiving the first data phase IN token after the SETUP, the core push a SETUP_COMPLETE status into the RxFIFO.Core NAKs the data phase IN tokens because of the NAK set on receiving the SETUP packet.
4. The core generates DOEPINT.XferCompl interrupt after having transferred the SETUP packet into memory (Step 2).
5. The core generates DOEPINT.SetUp interrupt after the DMA has popped the SETUP_COMPLETE status out of the RxFIFO.
6. Data phase IN tokens are NAKed until this point because the NAK has not yet been cleared by the application.
7. The core starts fetching the IN data after the application enables IN DMA (In this example it is assumed that multiple packets are in the same buffer. But it could also be in different buffers). This involves the following steps
   a) Fetch the descriptor pointed by DIEPDMA.
   b) Fetch the data into the corresponding Tx FIFO.
   c) Close the descriptor with DMA_DONE status.
8. Application clear the NAK after receiving the setup complete (DOEPINT.SetUp) interrupt.
9. After all the data has been fetched for the descriptor (Step 7), core generates DIEPINT.XferCompl interrupt.
10. The core sends data in response to the IN token for the data phase.
11. Host switches to status phase and sends the status phase OUT token. Core ACks the OUT packet because the NAK has already been cleared.
12. The DMA detects the RxFIFO as non-empty (because of the status phase data) and does following.
    a) Fetch the descriptor pointed by DOEPMA.
    b) Transfer the SETUP packet from the RxFIFO to the buffer pointed by the descriptor.
    c) Close the descriptor with DMA_DONE status.
    d) The core generates DOEPINTx.XferCompl after transferring the status phase data to system memory and closing the descriptor.

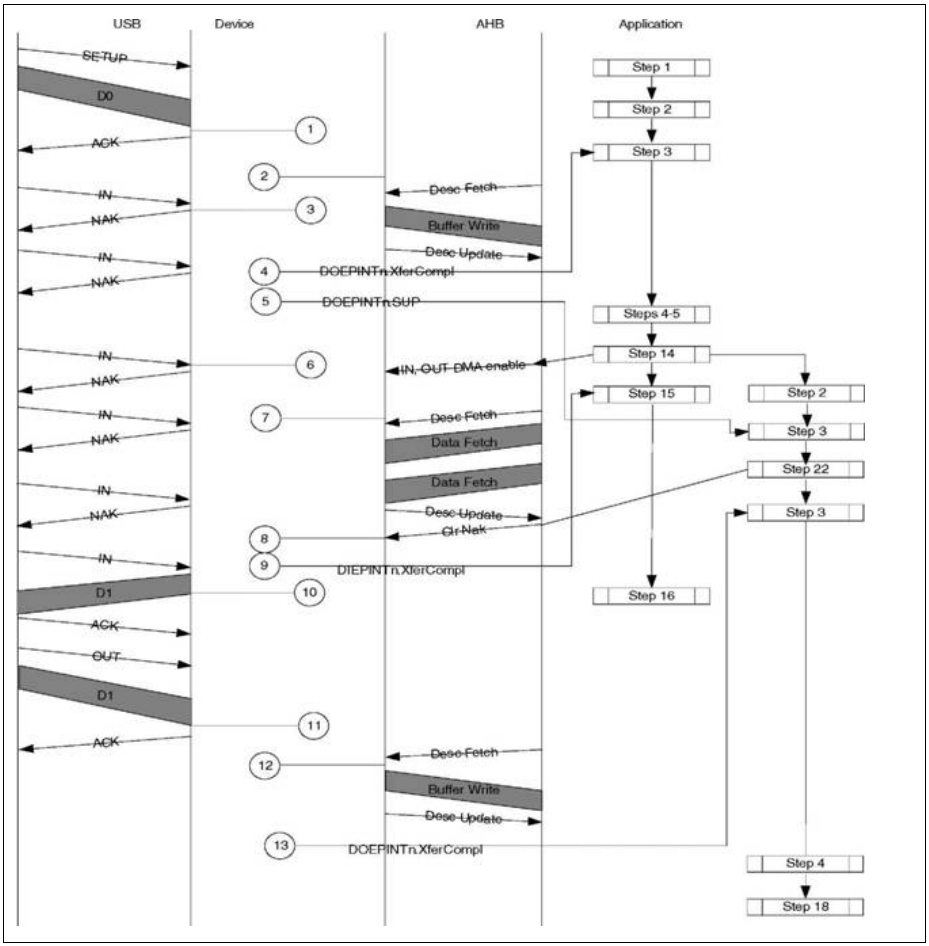


Figure 13-33 Premature Status Phase During Control Read

13.7.7.12 Lost ACK During Last Packet of Control Read

This is similar to the previous section. **Figure 13-34** shows this.

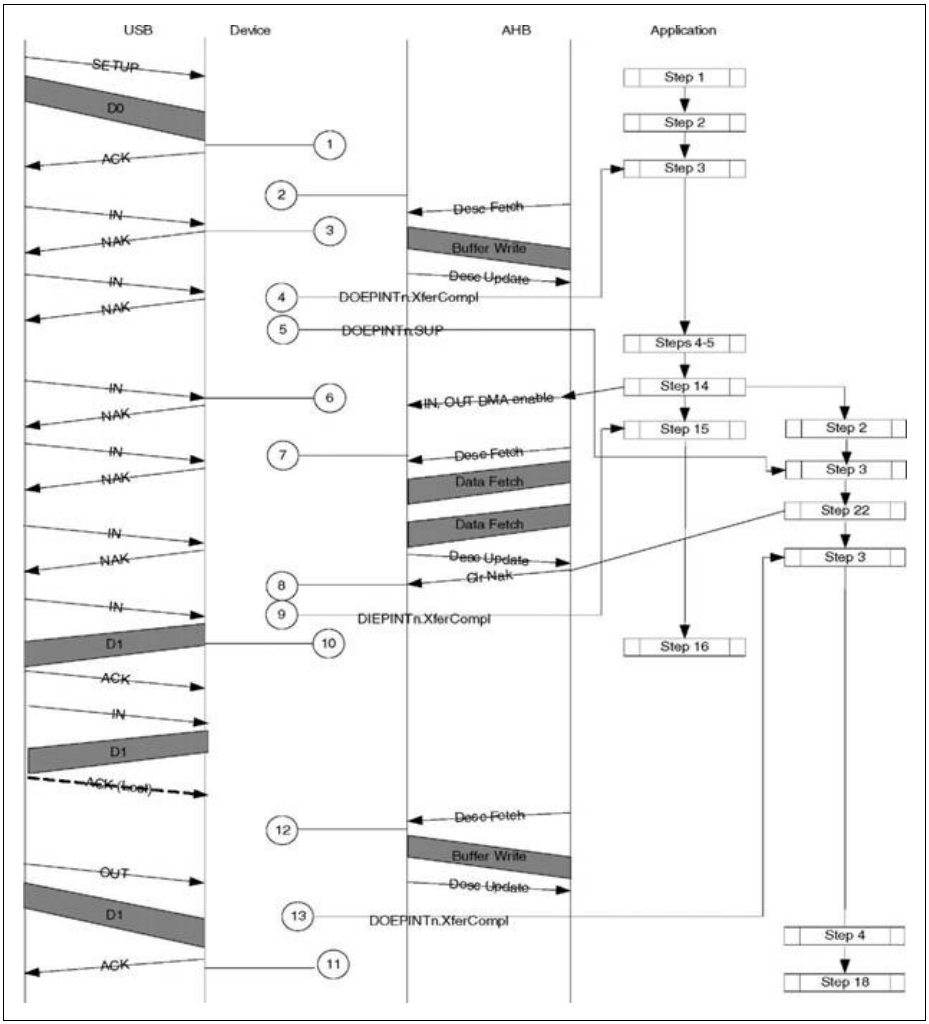


Figure 13-34 Lost ACK During Last Packet of Control Read

13.7.8 Bulk Transfer Handling in Scatter/Gather DMA Mode

13.7.8.1 Bulk IN Transfer in Scatter-Gather DMA Mode

Interrupt usage

The following interrupts are of relevance.

1. DIEPINTx.XferCompl (Transfer complete, based on IOC bit in the descriptor)
2. DIEPINTx.BNA (Buffer Not Available)

Application Programming Sequence

This section describes the application programming sequence for Bulk IN transfer scenarios.

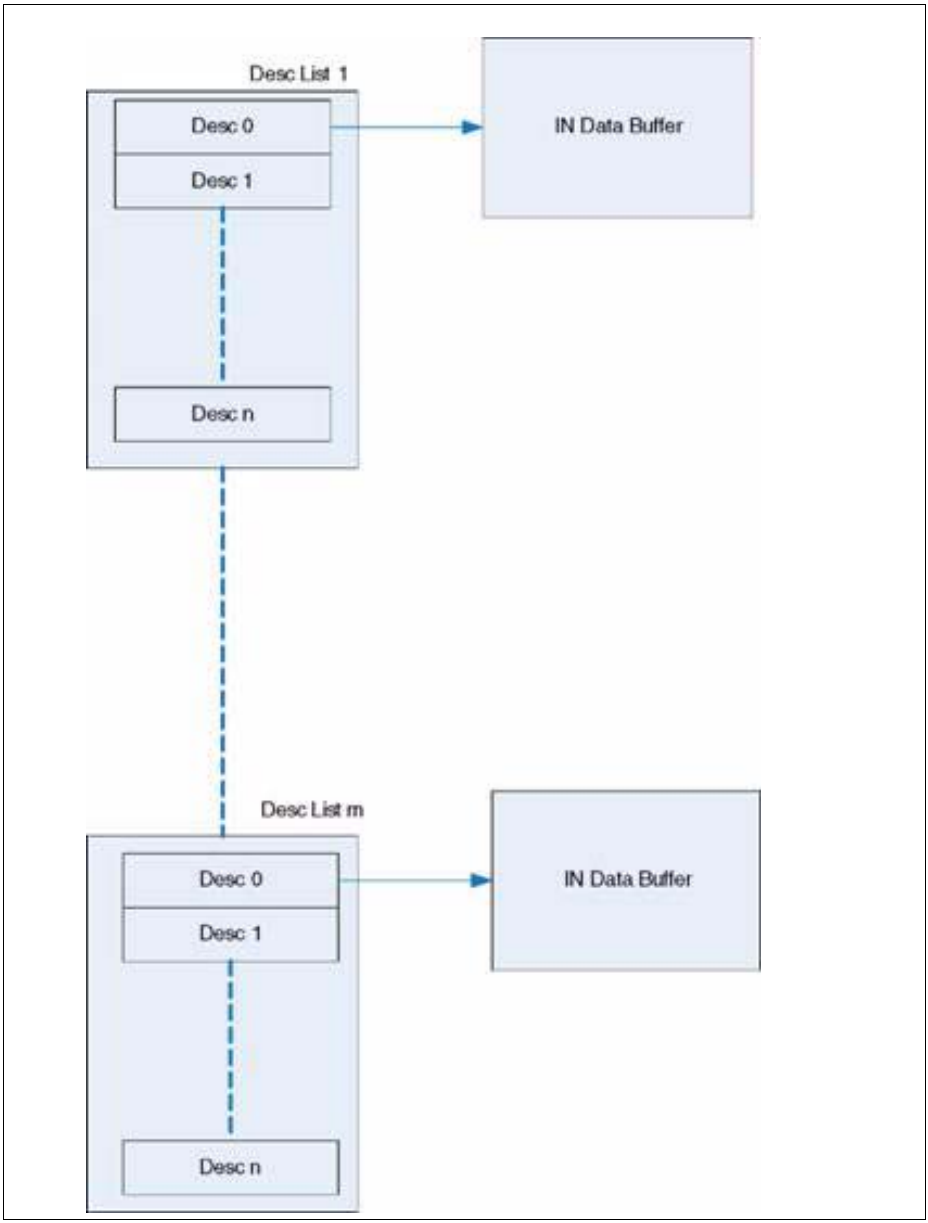


Figure 13-35 IN Descriptor List

1. Prepare Descriptor(s):
2. The application creates descriptor list(s) in the system memory pertaining to an Endpoint.
3. Each descriptor list may have up to n descriptors and there may be up to m descriptor lists.
4. Application may choose to set the IOC bit of the corresponding descriptor. If the IOC is set for the last descriptor of the list, the core generates DIEPINTx.XferCompl interrupt after the entire list is processed.
5. Program DIEPDMAx:
   a) Application programs the base address of the descriptor in the corresponding IN Endpoint DIEPDMAx register.
6. Enable DMA:
   a) Application programs the corresponding endpoint DIEPCTLx register with the following
      - DIEPCTLx.MPS — Max Packet size of the endpoint
      - DIEPCTLx.CNAK—Set to 1 to clear the NAK
      - DIEPCTLx.EPEna — Set to 1 to enable the DMA for the endpoint.
7. Wait for Interrupt:
   a) On reception of DIEPINTx.XferCompl, application must check the Buffer status and Tx Status field of the descriptor to ascertain that the descriptor closed normally.

DIEPINTx.BNA interrupt gets generated by the core when it encounters a descriptor in the list whose Buffer Status field is not Host Ready. In this case, the application is suppose to read the DIEPDMAx register to ascertain the address for which the BNA interrupt is asserted to take corrective action.

Internal Flow

Bulk IN Transfers

The core handles Bulk IN transfers internally as functionally depicted in **Figure 13-36** (Non ISO IN Descriptor/Data Processing). **Figure 13-37** depicts this flow.

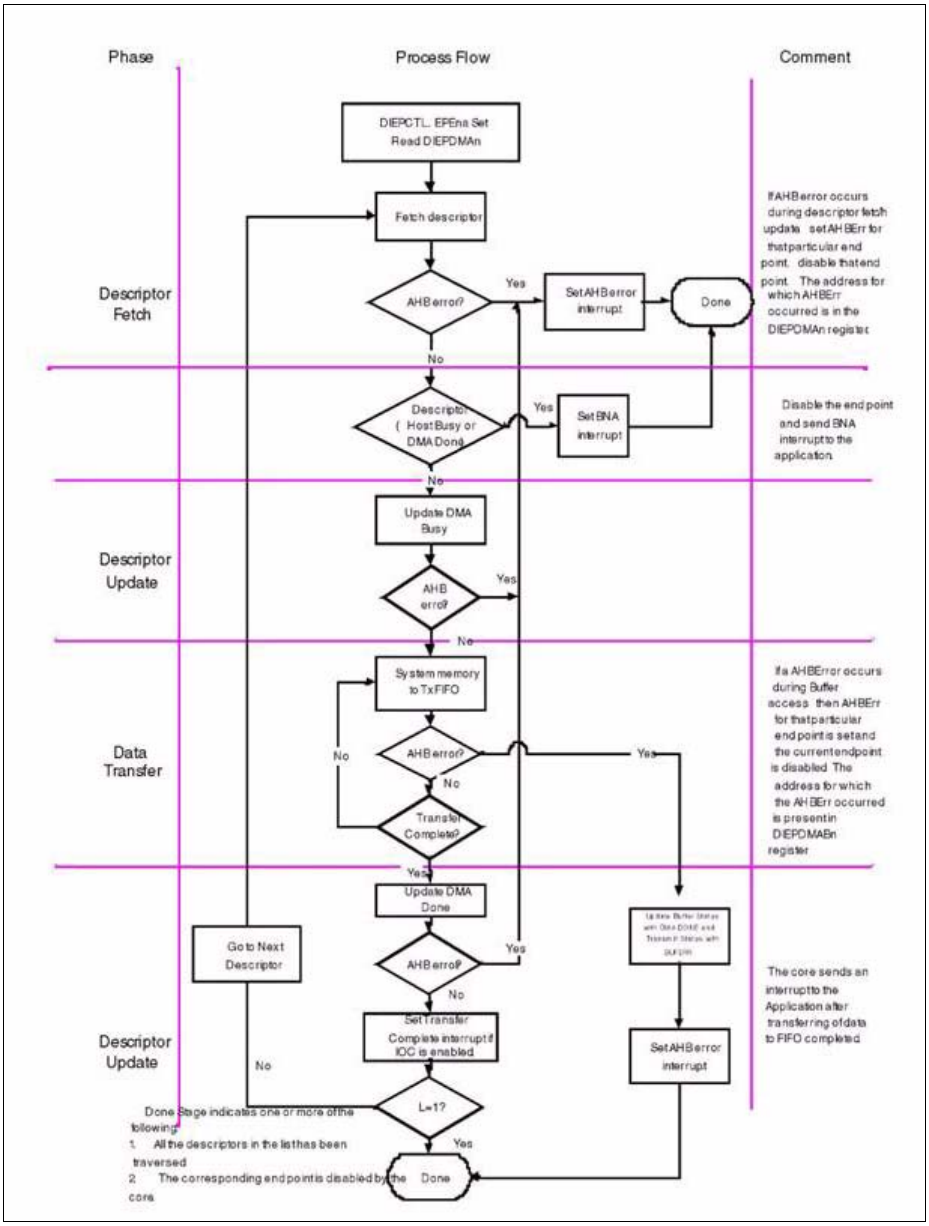


Figure 13-36 Non ISO IN Descriptor/Data Processing

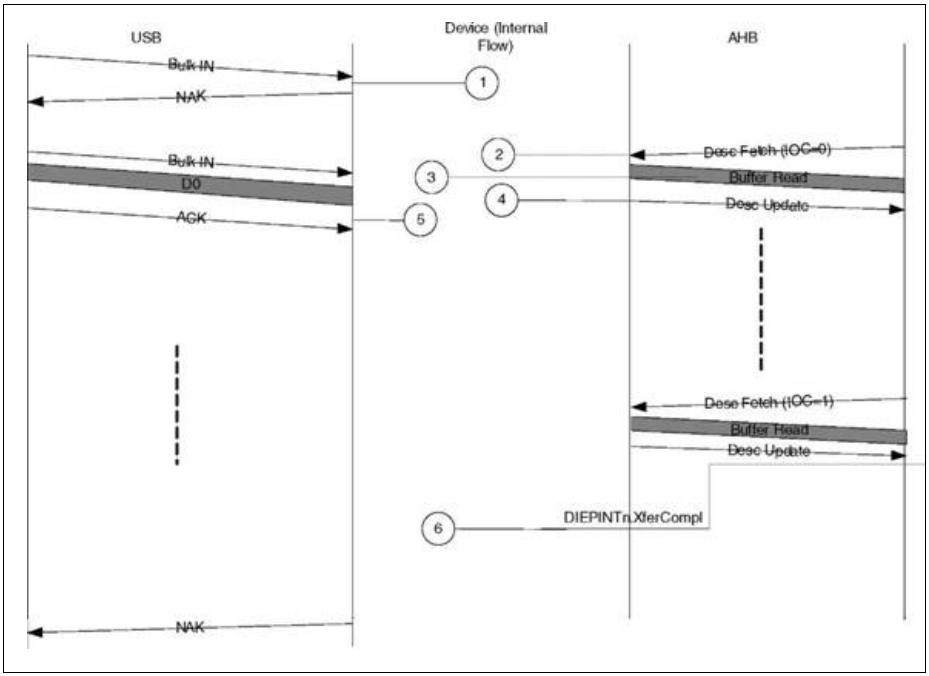


Figure 13-37 Bulk IN Transfers

1. When a BULK IN token is received on an end point before the corresponding DMA is enabled, (DIEPCTLx.EPEna = 0_B), it is NAKed on USB.
2. As a result of application enabling the DMA for the corresponding end point (DIEPCTLx.EPEna=1), the core fetches the descriptor and processes it.
3. The DMA fetches the data from the system memory and populates its internal FIFO with this data.
4. After fetching all the data from a descriptor, the core closes the descriptor with a DMA_DONE status.
5. On reception of BULK IN tokens on USB, data is sent to the USB Host.
6. After the last descriptor in the chain is processed, the core generates DIEPINTx.XferCompl interrupt provided the IOC bit for the last descriptor is set.

13.7.8.2 Bulk OUT Transfer in Scatter-Gather DMA Mode

Interrupt Usage

The following interrupts are of relevance.

1. DOEPINTx.XferCompl (Transfer complete, based on IOC bit in the descriptor)
2. DOEPINTx.BNA (Buffer Not Available)

Application Programming Sequence

This section describes the application programming sequence to take care of Bulk OUT transfer scenarios.

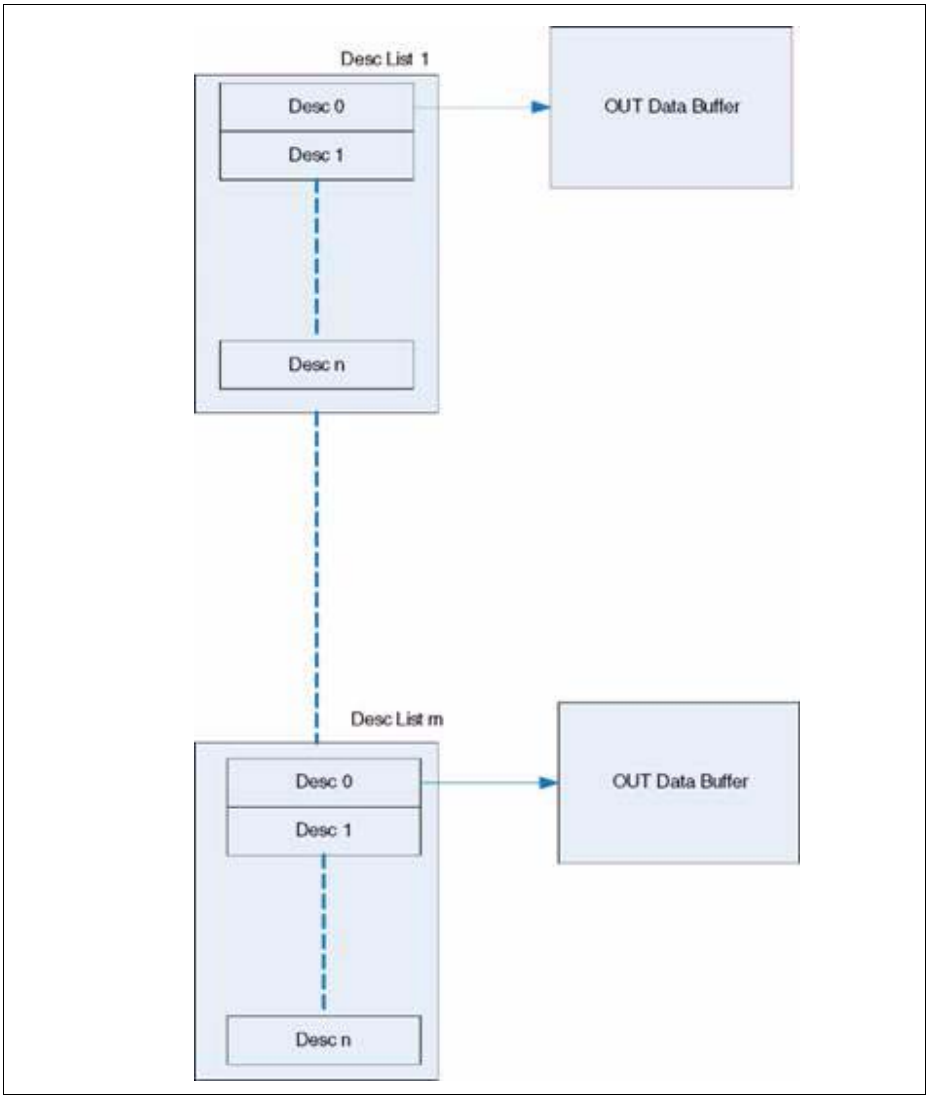


Figure 13-38 OUT Descriptor List

1. Prepare Descriptor(s):
2. The application creates descriptor list(s) in the system memory pertaining to an Endpoint.
3. Each descriptor list may have up to n descriptors and there may be up to m descriptor lists.
4. Application may choose to set the IOC bit of the corresponding descriptor. If the IOC is set for the last descriptor of the list, the core generates DOEPINTx.XferCompl interrupt after the entire list is processed.
   a) a. Based on L bit and MTRF bit combinations, the core may disable the end point. Refer to **Table 13-3 "OUT Data Memory Structure Values" on Page 13-89** for bit field descriptions.
   b) If the application programs the NAK bit, the core sets NAK for the endpoint after the descriptor is processed by the DMA. The application must set DIEPCTLn.CNAK to clear the NAK.
5. Program DOEPDMAx:
   a) Application programs the base address of the descriptor in the corresponding OUT Endpoint DOEPDMAx register.
6. Enable DMA:
   a) Application programs the corresponding endpoint DOEPCTLx register with the following:
      - DOEPCTL.MPS — Max Packet size of the endpoint
      - DOEPCTL.CNAK—Set to 1 to clear the NAK
      - DOEPCTL.EPEna — Set to 1 to enable the DMA for the endpoint.
7. Wait for Interrupt:
   a) On reception of DOEPINTx.XferCompl, application must check the Buffer status and Rx Status field of the descriptor to ascertain that the descriptor closed normally.
   b) On receiving DOEPINTn.BNA interrupt (generated by the core when it encounters a descriptor in the list whose Buffer Status field is not Host Ready), the application must read the DOEPDMAn register to ascertain the address for which the BNA interrupt is asserted to take one of the following corrective actions:
      - If the application has programmed DCTL.EnContOnBNA = 1_B, then the application must read the DOEPDMA register, identify which descriptor received BNA status, process the descriptor, and re-enable the endpoint. The core continues to process descriptors from the descriptor which received BNA status.
      - If the application has programmed DCTL.EnContOnBNA = 1_B, then every time the endpoint is disabled, the core automatically sets the NAK for the endpoint, the application must set DIEPCTLn.CNAK = 1 while enabling the endpoint. This ensures that the core does not accept packets when the DMA is disabled for an endpoint.
      - If the application has programmed DCTL.EnContOnBNA = 0_B, then the application must read the DOEPDMA register, identify which descriptor received BNA status, process the descriptor, and re-enable the endpoint. The core

continues to process descriptors from the start of the descriptor chain (that is, the descriptor programmed in the DOEPDMAn register).

Internal Flow

The core handles Bulk OUT transfers internally as depicted in **Figure 13-39**. **Figure 13-40** also diagrams this flow.

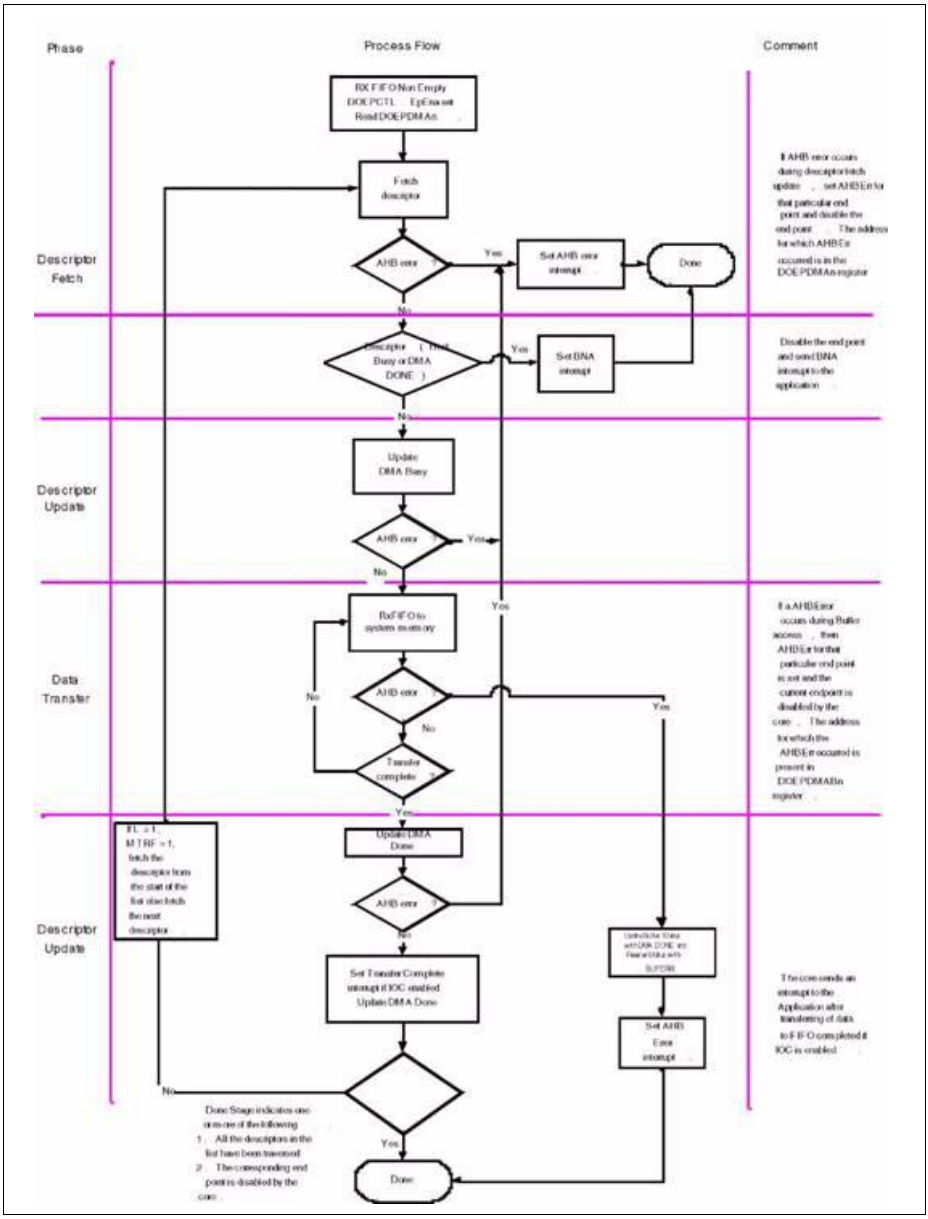


Figure 13-39 Non ISO OUT Descriptor/Data Buffer Processing

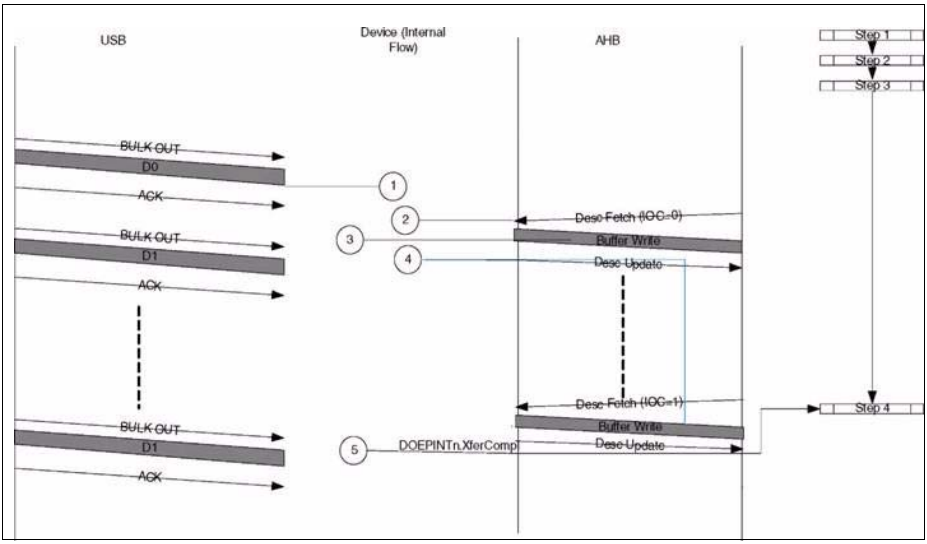


Figure 13-40 Bulk OUT Transfers

1. When a BULK OUT token is received on an end point, the core stores the received data internally in a FIFO.
2. As a result of application enabling the DMA for the corresponding end point (DOEPCTLx.EPEna=1), the core fetches the descriptor and processes it.
3. If the descriptor Buffer Status is HOST_READY:
   a) The DMA transfers the data from the internal FIFO to system memory.
   b) After transferring all the data from the FIFO, the core closes the descriptor with a DMA_DONE status.
   c) After the last descriptor in the chain is processed, the core generates DOEPINTn.XferCompl interrupt provided the IOC bit for the last descriptor is set.
4. If the descriptor Buffer Status is not HOST_READY:
   a) The DMA generates Buffer Not Available (BNA) interrupt
   b) After the application re-enables the corresponding endpoint (DOEPCTLn.EPEna=1), the core does the following:
   - Fetches the base descriptor from DOEPDMAn (DCTL.EnContOnBNA = 0_B).
   - Fetches the descriptor that received the BNA interrupt is (DCTL.EnContOnBNA = 1_B).

13.7.9 Interrupt Transfer Handling in Scatter/Gather DMA Mode

13.7.9.1 Interrupt IN Transfer in Scatter/Gather DMA Mode

Application programming for Interrupt IN transfers is as with the Bulk IN transfer sequence. The core handles Interrupt IN transfers internally in the same way it handles Bulk IN transfers

13.7.9.2 Interrupt OUT Transfer in Scatter/Gather DMA Mode

Application programming for Interrupt OUT transfers is as with the Bulk OUT transfer sequence. The core handles Interrupt OUT transfers internally in the same way it handles Bulk OUT Transfers

13.7.10 Isochronous Transfer Handling in Scatter/Gather DMA Mode

13.7.10.1 Isochronous IN Transfer in Scatter/Gather DMA Mode

The application programming for Isochronous IN transfers is in the same manner as Bulk IN transfer sequence.

The following behavior is of importance while working with Isochronous IN end points

DCTL.IgnrFrmNum = $\mathbf{1_B}$

The way the core handles Isochronous IN transfers internally in the same way as it handles Bulk IN Transfers.

DCTL.IgnrFrmNum = 0_B

The core closes the descriptor and clears the corresponding fetched data in the FIFO if the USB frame number to which the descriptor belongs is elapsed.

Isochronous Transfers in Scatter/Gather (Descriptor DMA) Mode

This topic includes descriptions of both isochronous IN and OUT transfers

Isochronous IN

In the case of ISO IN After descriptor is fetched, the frame number field M is compared with current USB frame number N.

If the frame number in the fetched descriptor is already elapsed (M<N) then the descriptor is closed with status changed to DMA Done.

If the frame number in the fetched descriptor is for future (N>M+1) then the descriptor is left untouched. The Core suspends and re-look at this descriptor contents in the next frame.

- If the frame number in the fetched descriptor is for current or next frame (N=M or M+1) then the descriptor is further processed as per the flow chart. At the end of data

transfer from memory to TxFIFO the above check must be performed. And if the data fetch finished in the subsequent frame, data must be flushed and descriptor must be closed (DMA Done) with BUFFLUSH status.

- For ISO IN, the application creates a series of descriptors (D,D+1,D+2) for a given periodic end point corresponding to successive frames (N,N+1,N+2).

Note: The series of descriptors does not correspond to the series of frames in the same order.

For example, D and D + 1 may correspond to N, D + 2 may correspond to N + 1 and so on except in the case where the application can create more than one descriptor for the same frame. The core fetches the descriptor and compares the frame/ ^frame number field with the current frame/ ^frame number.

If the fetched descriptor corresponds to a frame which has already elapsed, the core updates the descriptor with DMA Done Buffer status and proceeds to the next descriptor.

If the next descriptor fetched indicates that it corresponds to frame number N or N + 1, it services it. In the process of fetching the descriptors, if the core determines that the descriptor corresponds to a future frame/ ^.frame (> N + 1), it does not service the descriptor in that frame/^frame. Instead, it moves on to the next periodic endpoint or non-periodic endpoint without disabling the current periodic endpoint. It revisits this endpoint in the next frame/ ^.frame and repeats the process.

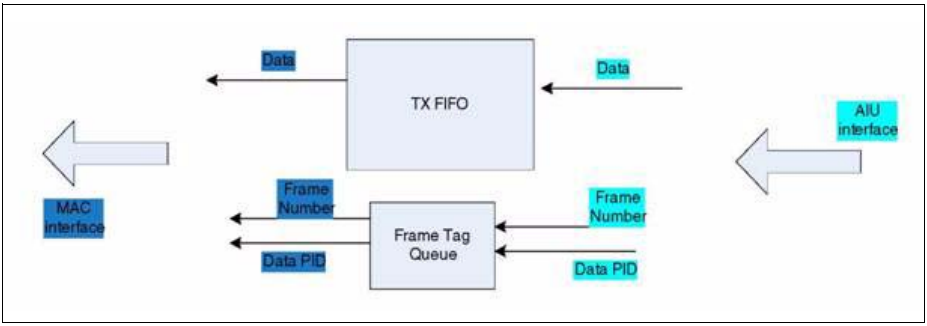


Figure 13-41 ISO IN Data Flow

Application Programming Sequence

This section describes the application programming sequence for Isochronous IN transfer scenarios.

Prepare Descriptor(s)

The application creates descriptor list(s) in the system memory pertaining to an Endpoint.

Each descriptor list may have up to n descriptors and there may be up to m descriptor lists.

Application may choose to set the IOC bit of the corresponding descriptor. If the IOC is set for the last descriptor of the list, the core generates DIEPINTx.XferCompl interrupt after the entire list is processed.

1. Program DIEPDMAx:
   a) Application programs the base address of the descriptor in the corresponding IN Endpoint DIEPDMAx register.
2. Enable DMA:
   a) Application programs the corresponding endpoint DIEPCTLx register with the following
      - DIEPCTLx.MPS — Max Packet size of the endpoint
      - DIEPCTLx.CNAK—Set to 1 to clear the NAK
      - DIEPCTLx.EPEna — Set to 1 to enable the DMA for the endpoint.
3. Wait for Interrupt:
   a) On reception of DIEPINTx.XferCompl, application must check the Buffer status and Tx Status field of the descriptor to ascertain that the descriptor closed normally.

DIEPINTx.BNA interrupt gets generated by the core when it encounters a descriptor in the list whose Buffer Status field is not Host Ready. In this case, the application is suppose to read the DIEPDMAx register to ascertain the address for which the BNA interrupt is asserted to take corrective action.

Internal Flow

The core handles isochronous IN transfers internally as functionally depicted in **Figure 13-42**. **Figure 13-43** also diagrams this flow.

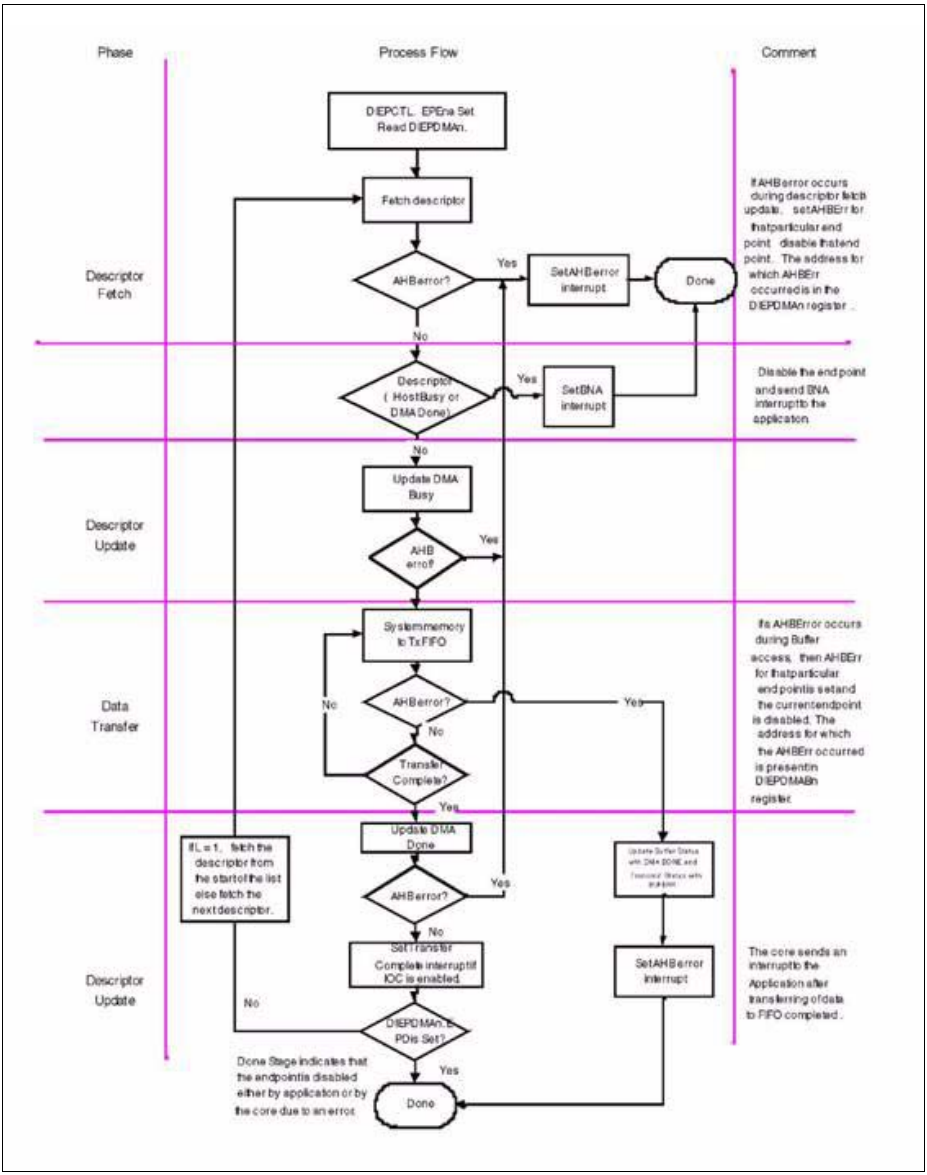


Figure 13-42 ISO IN Descriptor/Data Processing

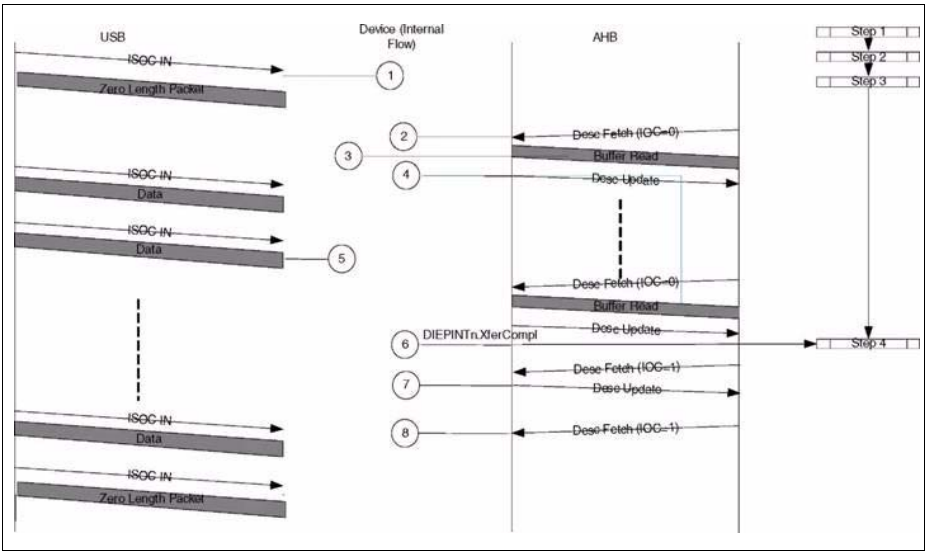


Figure 13-43 Isochronous IN Transfers

1. When an Isochronous IN token is received on an end point before the corresponding DMA is enabled, (DIEPCTLx.EPEna = 0_B), zero length packet is sent on USB.
2. As a result of application enabling the DMA for the corresponding end point (DIEPCTLx.EPEna=1), the core fetches the descriptor. If the descriptor belongs to the current or the next USB frame number, the core processes it.
3. The DMA fetches the data pointed by the above descriptor from the system memory and populates its internal FIFO with this data.
4. After fetching all the data, the core closes the descriptor with a DMA_DONE status.
5. On reception of Isochronous IN tokens on USB, data is sent to the USB Host.
6. After the last descriptor in the chain is processed, the core generates DIEPINTx.XferCompl interrupt provided the IOC bit for the last descriptor is set.
7. When the DMA fetches a descriptor whose USB frame number has been already elapsed, it closes that descriptor with a DMA_DONE status without fetching the data for that descriptor.
8. When the DMA fetches a descriptor which has a future USB frame number, it does not service it in the current context. It services it in the future.

13.7.10.2 Isochronous OUT Transfer in Scatter/Gather DMA Mode

The application programming for isochronous out transfers is in the same manner as Bulk OUT transfer sequence, except that the application creates only 1 packet per descriptor for an isochronous OUT endpoint. The core handles isochronous OUT transfers internally in the same way it handles Bulk OUT transfers, and as depicted in **Figure 13-44**.

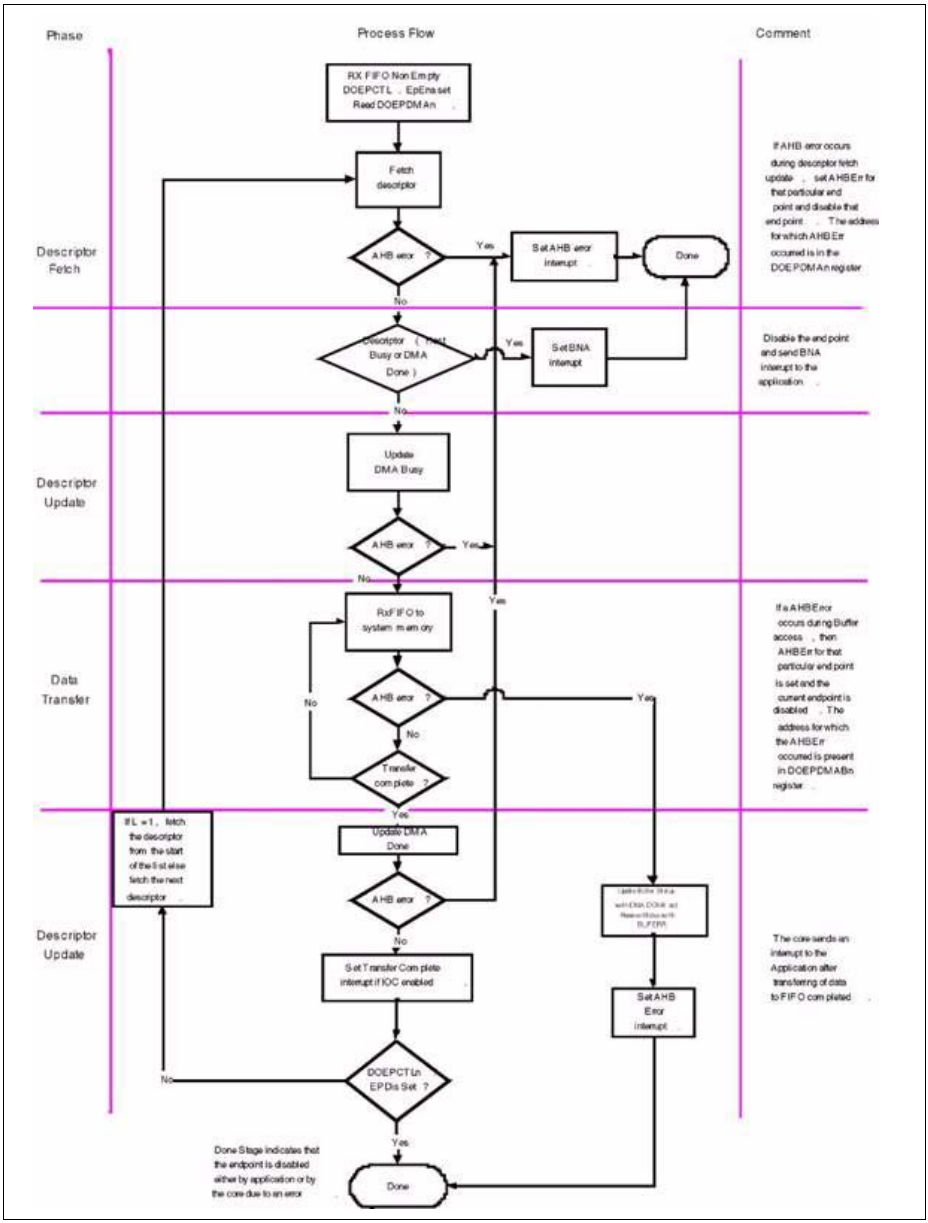


Figure 13-44 Isochronous OUT Descriptor/Data Buffer Processing

Isochronous OUT

For ISO OUT transactions, the core transfers the packets from the Rx FIFO to the system memory and updates the frame number field of the descriptor with the frame number in which the packet was received.The frame number for which data is received is extracted from the Receive Status queue and written back to the descriptor.

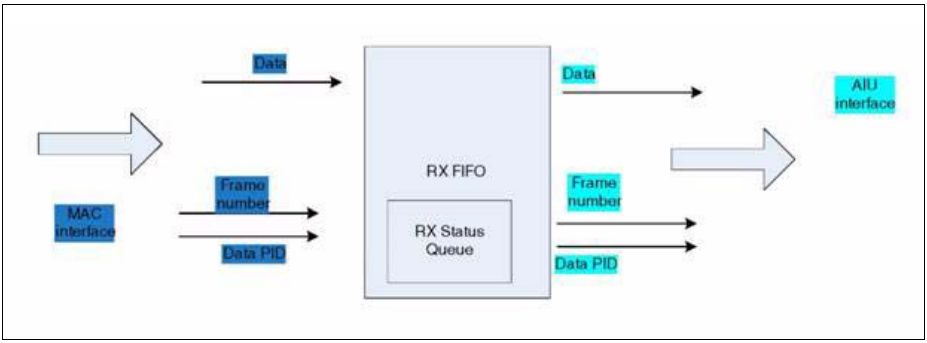


Figure 13-45 ISO Out Data Flow

Note: Incomplete Isochronous Interrupt (GINTSTS.incomplete) is not generated in Scatter/Gather DMA mode. Received isochronous packets are sent unmodified to the application memory, with the corresponding frame number updated in the descriptor status.

13.8 Clock Gating Programming Model

When the USB is suspended or the session is not valid, the PHY is driven into Suspend mode, and the PHY clock is stopped to reduce power consumption in the PHY and the USB core. The PHY clock is turned off for as long as the core asserts the suspend signal.

To further reduce power consumption, the USB core also supports AHB clock gating. The AHB clock to some of the USB internal modules can be gated by writing to the Gate Hclk bit in the Power and Clock Gating Control register.

The following sections show the procedures to use the clock gating feature.

13.8.1 Device Mode Suspend and Resume With Clock Gating

Sequence of operations:

1. The core detects a USB suspend and generates a Suspend Detected interrupt.
2. The application sets the Stop PHY Clock bit in the Power and Clock Gating Control register, the core asserts the suspend_n signal to the PHY, and the PHY clock stops. The application sets the Gate hclk bit in the Power and Clock Gating Control register, and the core gates the hclk (hclk_ctl) to AHB- domain modules other than the BIU.

3. The core remains in Suspend mode.
4. The Resume signaling from the host is detected. The core deasserts the suspend_n signal to the PHY to generate the PHY clock. A Resume Detected interrupt is generated.
5. The application clears the Gate hclk bit and the Stop PHY Clock bit.
6. The host finishes Resume signaling.
7. The core is in normal operating mode.

13.8.2 Device Mode Suspend and Remote Wakeup With Clock Gating

Sequence of operations:

1. The core detects a USB suspend and generates a Suspend Detected interrupt.
2. The application sets the Stop PHY Clock bit in the Power and Clock Gating Control register, the core asserts the suspend_n signal to the PHY, and the PHY clock stops. The application sets the Gate hclk bit in the Power and Clock Gating Control register, the core gates the hclk (hclk_ctl) to AHB-domain modules other than the BIU.
3. The core remains in Suspend mode.
4. The application clears the Gate hclk bit and the Stop PHY Clock bit.
5. The application sets the Remote Wakeup bit in the Device Control register, the core starts driving Remote Wakeup signaling.
6. The host drives Resume signaling.
7. The core is in normal operating mode.

13.8.3 Device Mode Session End and Start With Clock Gating

Sequence of operations:

1. The core detects a USB suspend, and generates a Suspend Detected interrupt. The host turns off VBUS.
2. The application sets the Stop PHY Clock bit in the Power and Clock Gating Control register, the core asserts the suspend_n signal to the PHY, and the PHY clock stops. The application sets the Gate hclk bit in the Power and Clock Gating Control register, and the core gates the hclk (hclk_ctl) to AHB- domain modules other than the BIU.
3. The core remains in Low-Power mode.
4. The new session is detected (bsessvld is high). The core deasserts the suspend_n signal to the PHY to generate the PHY clock. A New Session Detected interrupt is generated.
5. The application clears the Gate hclk and Stop PHY Clock bits.
6. The core detects USB reset.
7. The core is in normal operating mode

13.8.4 Device Mode Session End and SRP With Clock Gating

Sequence of operations:

1. The core detects a USB suspend, and generates a Suspend Detected interrupt. The host turns off VBUS.
2. The application sets the Stop PHY Clock bit in the Power and Clock Gating Control register, the core asserts the suspend_n signal to the PHY, and the PHY clock stops. The application sets the Gate hclk bit in the Power and Clock Gating Control register, and the core gates the hclk (hclk_ctl) to AHB- domain modules other than the BIU.
3. The core remains in Low-Power mode.
4. The application clears the Gate hclk and Stop PHY Clock bits.
5. The application sets the SRP Request bit, and the core drives data line and VBUS pulsing.
6. The host turns on Vbus, detects device connection, and drives a USB reset.
7. The core is in normal operating mode.

13.9 FIFO RAM Allocation

13.9.1 Data FIFO RAM Allocation

The RAM must be allocated among different FIFOs in the core before any transactions can start. The application must follow this procedure every time it changes core FIFO RAM allocation.

The application must allocate data RAM per FIFO based on the following criteria:

- AHB's operating frequency
- PHY Clock frequency
- Available AHB bandwidth
- Performance required on the USB

Based on the above criteria, the application must provide a table with RAM sizes for each FIFO in each mode.

USB core shares a single SPRAM between transmit FIFO(s) and receive FIFO.

In DMA mode — The SPRAM is also used for storing some register information:

- **In non Scatter Gather mode** — The Device mode Endpoint DMA address registers (DI/OEPDMAn) and Host mode Channel DMA registers (HCDMA) are stored in the SPRAM.
- **In Scatter Gather mode** — The Base descriptor address, the Current descriptor address, the current buffer address and the descriptor status quadlet information for each endpoint/channel are stored in the SPRAM.
  These register information are stored at the end of the SPRAM after the space allocated for receive and Transmit FIFO. These register space must also be taken into account when allocating the RAM among the different FIFOs.

In Slave mode — No registers are stored in the SPRAM. Therefore no additional space needs to be allocated in the SPRAM for register information.

The following rules apply while calculating how much RAM space must be allocated to store these registers.

For example in Scatter/Gather DMA mode, if there are five bidirectional endpoints, then the last forty SPRAM locations are reserved for storing these values.

13.9.1.1 Device Mode RAM Allocation

Considerations for allocating data RAM for Device Mode FIFOs are listed here:

1. Receive FIFO RAM Allocation:
   a) RAM for SETUP Packets: 10 locations must be reserved in the receive FIFO to receive up to n SETUP packets on the control endpoint. The core does not use these locations, which are reserved for SETUP packets, to write any other data.
   b) One location for Global OUT NAK
   c) Status information is written to the FIFO along with each received packet. Therefore, a minimum space of (Largest Packet Size / 4) + 1 must be allotted to receive packets. If a high-bandwidth endpoint is enabled, or multiple isochronous endpoints are enabled, then at least two (Largest Packet Size / 4) + 1 spaces must be allotted to receive back-to-back packets. Typically, two (Largest Packet Size / 4) + 1 spaces are recommended so that when the previous packet is being transferred to AHB, the USB can receive the subsequent packet. If AHB latency is high, enough space must be allocated to receive multiple packets. This is critical to prevent dropping any isochronous packets.
   d) Along with each endpoint's last packet, transfer complete status information is also pushed to the FIFO. Typically, one location for each OUT endpoint is recommended.
2. Transmit FIFO RAM Allocation:
   a) The minimum RAM space required for each IN Endpoint Transmit FIFO is the maximum packet size for that particular IN endpoint.
   b) More space allocated in the transmit IN Endpoint FIFO results in a better performance on the USB and can hide latencies on the AHB.

Table 13-11 FIFO Name - Data RAM Size

FIFO Name	Data RAM Size
Receive data FIFO	rx_fifo_size. This must include RAM for setup packets, OUT endpoint control information and data OUT packets.
Transmit FIFO 0	tx_fifo_size[0]
Transmit FIFO 1	tx_fifo_size[1]
Transmit FIFO 2	tx_fifo_size[2]
...	...
Transmit FIFO i	tx_fifo_size[i]

With this information, the following registers must be programmed as follows:

1. Receive FIFO Size Register (GRXFSIZ)
   GRXFSIZ.Receive FIFO Depth = rx_fifo_size;
2. Device IN Endpoint Transmit FIFO0 Size Register (GNPTXFSIZ)
   GNPTXFSIZ.non-periodic Transmit FIFO Depth = tx_fifo_size[0];
   GNPTXFSIZ.non-periodic Transmit RAM Start Address = rx_fifo_size;
3. Device IN Endpoint Transmit FIFO#1 Size Register (DIEPTXF1)
   DIEPTXF1. Transmit RAM Start Address = GNPTXFSIZ.FIFO0 Transmit RAM Start Address + tx_fifo_size[0];
4. Device IN Endpoint Transmit FIFO#2 Size Register (DIEPTXF2)
   DIEPTXF2.Transmit RAM Start Address = DIEPTXF1.Transmit RAM Start Address + tx_fifo_size[1];
5. Device IN Endpoint Transmit FIFO#i Size Register (DIEPTXFi)
   DIEPTXFm.Transmit RAM Start Address = DIEPTXFi-1.Transmit RAM Start Address + tx_fifo_size[i-1];
6. The transmit FIFOs and receive FIFO must be flushed after the RAM allocation is done, for the proper functioning of the FIFOs.
   a) GRSTCTL.TxFNum = 10_H
   b) GRSTCTL.TxFFlush = 1_B
   c) GRSTCTL.RxFFlush = 1_B
   d) The application must wait until the TxFFlush bit and the RxFFlush bits are cleared before performing any operation on the core.

See also **Figure 13-46**.

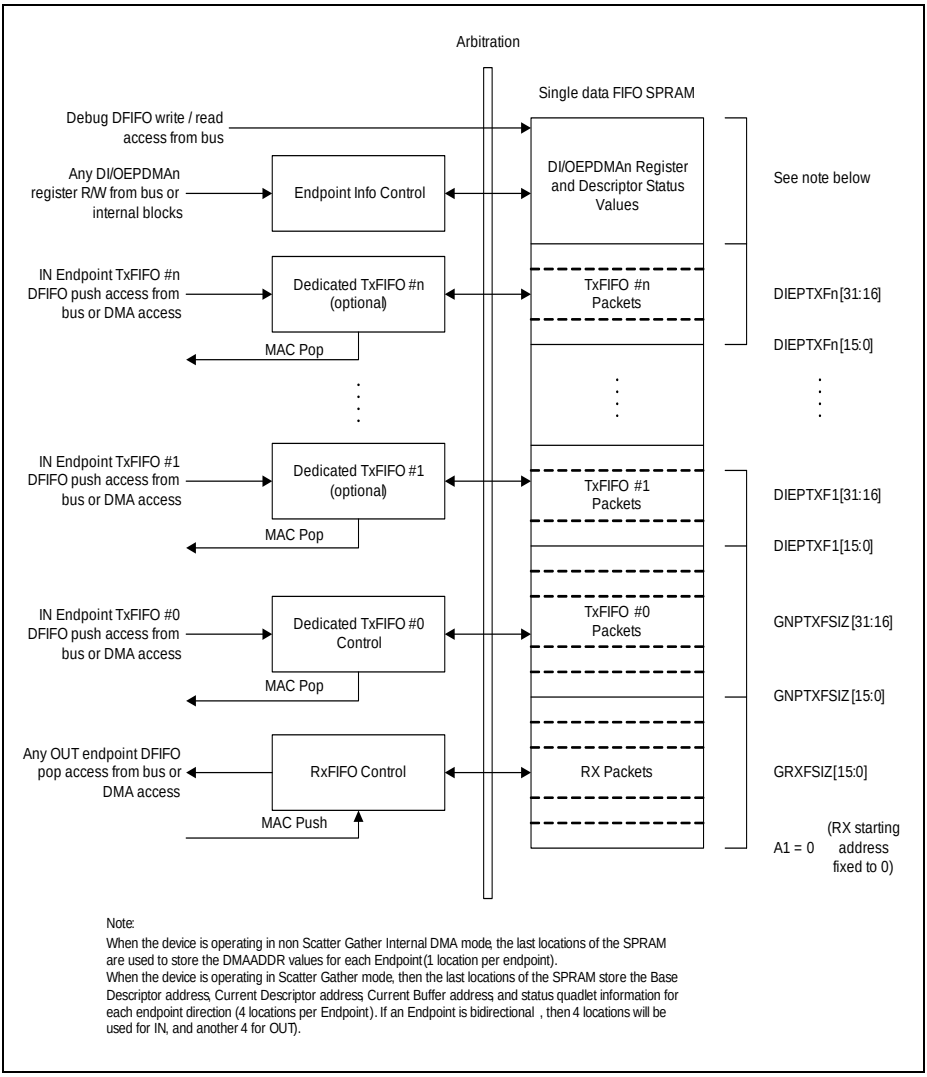


Figure 13-46 Device Mode FIFO Address Mapping

13.9.2 Dynamic FIFO Allocation

The application can change the RAM allocation for each FIFO during the operation of the core.

13.9.2.1 Dynamic FIFO Reallocation in Device Mode

Dynamic FIFO re-allocation in device mode occurs when there is Power On Reset or a USB Reset.

In Device mode, before changing FIFO data RAM allocation,

1. The application must determine the following:
   a) DIEPCTLn/DOEPCTLn.EPEna = 0_B
   b) DIEPCTLn/DOEPCTLn.NAKSts = 1_B
      If the bits are not set as above, follow the procedure in **"Transfer Stop Programming for OUT Endpoints" on Page 13-17** or **"Transfer Stop Programming for IN Endpoints" on Page 13-21** to ensure that all transfers on that endpoint are stopped.
2. Once these conditions are met, the application can reallocate FIFO data RAM as explained in **"Data FIFO RAM Allocation" on Page 13-154**.
3. Flush the TxFIFO in the core using the GRSTCTL.TxFIFO field. For more information on flushing TxFIFO, see **Flushing TxFIFOs in the Core**.

Note: The GlobalOUTNak process to disable OUT endpoints ensures that the RxFIFO does not have any data, so an RxFIFO flush is not required.

13.9.2.2 Flushing TxFIFOs in the Core

The application can flush all TxFIFOs in the core using GRSTCTL.TxFFlsh as follows:

1. Check that GINTSTS.GINNakEff=0. If this bit is cleared then set DCTL.SGNPInNak=1.
   NAK Effective interrupt = H indicating that the core is not reading from the FIFO.
2. Wait for GINTSTS.GINNakEff=1, which indicates the NAK setting has taken effect to all IN endpoints.
3. Poll GRSTCTL.AHBIdle until it is 1.
   AHBIdle = H indicates that the core is not writing anything to the FIFO.
4. Check that GRSTCTL.TxFFlsh =0. If it is 0, then write the TxFIFO number you want to flush to GRSTCTL.TxFNum.
5. Set GRSTCTL.TxFFlsh=1 and wait for it to clear.
6. Set the DCTL.GCNPInNak bit.

13.10 Service Request Generation

The USB module provides a single service request output connected to an interrupt node in the Nested Vectored Interrupt Controller (NVIC)

Figure 13-47 displays the USB core interrupt hierarchy.

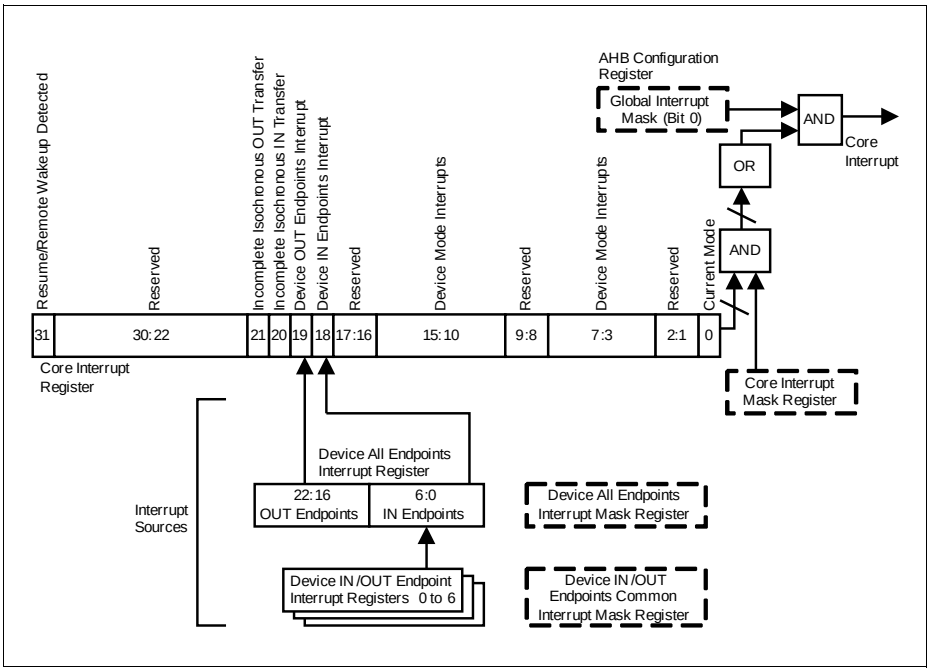


Figure 13-47 Interrupt Hierarchy

The Core Interrupt Handler

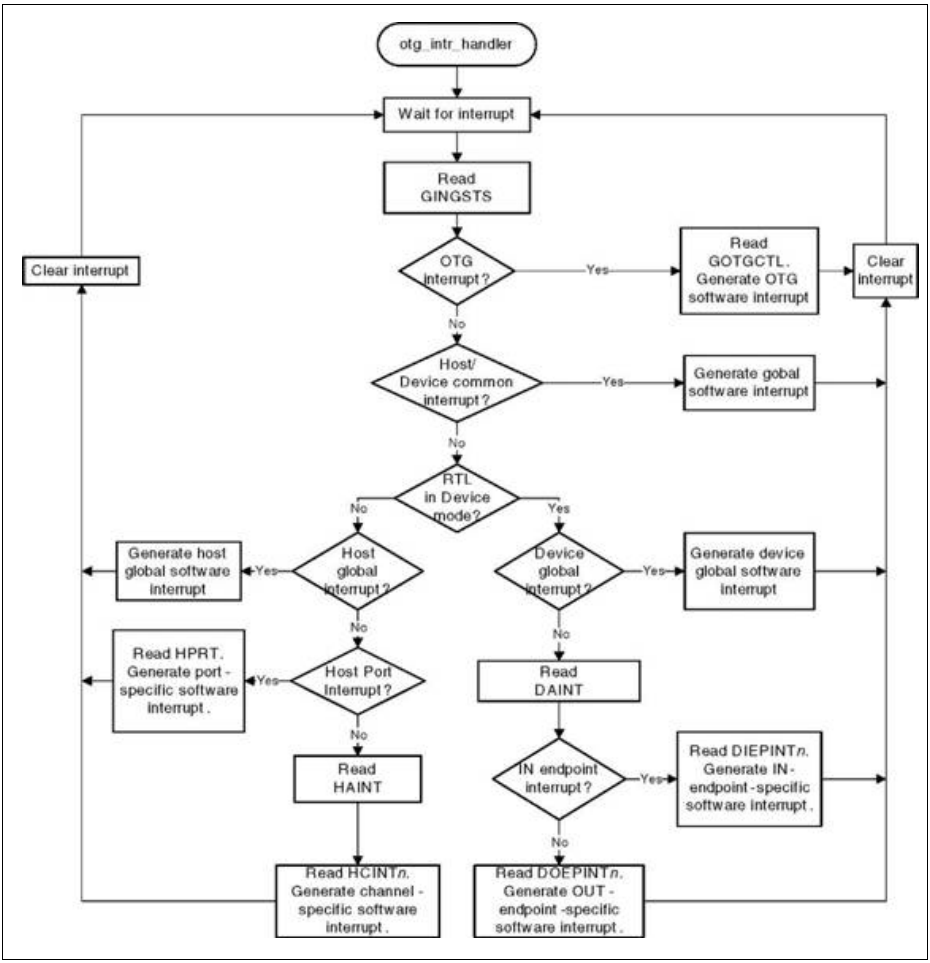


Figure 13-48 Core Interrupt Handler

13.11 Debug Behaviour

The USB module is not affected when the CPU enters HALT mode.

13.12 Power, Reset and Clock

The module, including all registers, can be reset to its default state by a system reset or a software reset triggered through the setting of corresponding bits in PRSETx registers.

The module has the following input clocks:

- clk_ahbm: the module clock, which is also referred to as hclk in this chapter
- clk_usb: the 48 MHz PHY clock., which is also referred to as phy_clk in this chapter

In addition, the module internally generates:

- hclk_gated: hclk gated for power optimization

13.13 Initialization and System Dependencies

The USB core is held in reset after a start-up from a system or software reset. The USB PHY is also by default in the power-down state. Therefore, the application has to apply the following initialization sequence before programming the USB core:

- Release reset of USB core by writing a 1 to the USBRS bit in SCU_PRCLR2 register
- Enable the 48 MHz PHY clock by configuring the USB PLL in SCU, see clock control section in SCU chapter
- Remove the USB PHY from power-down by writing a 1 to the USBPHYPDQ bits in SCU_PWRSET register

13.14 Registers

Register Overview

The application controls the USB core by reading from and writing to the Control and Status Registers (CSRs) through the AHB Slave interface. These registers are 32 bits wide and the addresses are 32-bit block aligned.

The absolute register address is calculated by adding:

Module Base Address + Offset Address

Table 13-12 Registers Address Space

Module	Base Address	End Address	Note
USB0	$5004\ 0000_H$	$5007\ FFFF_H$	

Figure 13-49 shows the CSR address map.

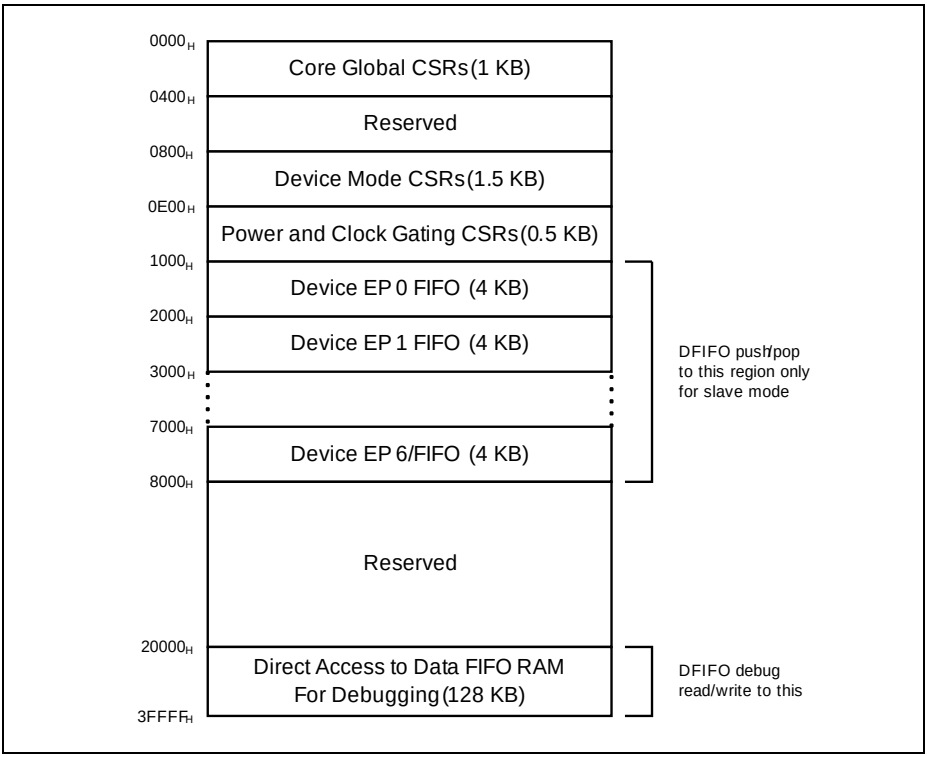


Figure 13-49 CSR Memory Map

The first letter of the register name is a prefix for the register type:

- G: Core Global
- D: Device mode

Note: FIFO size and FIFO depth are used interchangeably.

Table 13-13 Register Overview

Short Name	Description	Offset Addr.	Access Mode		Description See
			Read	Write	
USB Global Registers					
Reserved	Reserved	000_H-004_H	nBE	nBE	
GAHBCFG	AHB Configuration Register	008_H	U, PV	U, PV	**Page 13-168**

Table 13-13 Register Overview (cont'd)

Short Name	Description	Offset Addr.	Access Mode		Description See
			Read	Write	
GUSBCFG	USB Configuration Register	$00C_H$	U, PV	U, PV	Page 13-169
GRSTCTL	Reset Register	010_H	U, PV	U, PV	Page 13-171
GINTSTS	Interrupt Register	014_H	U, PV	U, PV	Page 13-175
GINTMSK	Interrupt Mask Register	018_H	U, PV	U, PV	Page 13-178
GRXSTSR	Receive Status Debug Read Register	$01C_H$	U, PV	U, PV	Page 13-180
GRXSTSP	Status Read and Pop Register	020_H	U, PV	U, PV	Page 13-180
GRXFSIZ	Receive FIFO Size Register	024_H	U, PV	U, PV	Page 13-182
GNPTXFSIZ	Non-Periodic Transmit FIFO Size Register	028_H	U, PV	U, PV	Page 13-182
Reserved	Reserved	$02CH$-038_H	nBE	nBE	
GUID	User ID Register	$03C_H$	U, PV	U, PV	Page 13-183
Reserved	Reserved	040_H-058_H	nBE	nBE	
GDFIFOCFG	DFIFO Software Config Register	$05C_H$	U, PV	U, PV	Page 13-183
Reserved	Reserved	060_H-100_H	nBE	nBE	
DIEPTXFn	Device IN Endpoint Transmit FIFO Size Register	104_H-124_H	U, PV	U, PV	Page 13-184
Reserved	Reserved	128_H-$7FF_H$	nBE	nBE	
Reserved	Reserved	504_H + n*20	nBE	nBE	
USB Device Mode Registers					
DCFG	Device Configuration Register	800_H	U, PV	U, PV	Page 13-186
DCTL	Device Control Register	804_H	U, PV	U, PV	Page 13-189
DSTS	Device Status Register	808_H	U, PV	U, PV	Page 13-193
Reserved	Reserved	$80C_H$	nBE	nBE	

Table 13-13 Register Overview (cont'd)

Short Name	Description	Offset Addr.	Access Mode		Description See
			Read	Write	
DIEPMSK	Device IN Endpoint Common Interrupt Mask Register	810_H	U, PV	U, PV	Page 13-195
DOEPMSK	Device OUT Endpoint Common Interrupt Mask Register	814_H	U, PV	U, PV	Page 13-196
DAINT	Device All Endpoints Interrupt Register	818_H	U, PV	U, PV	Page 13-197
DAINTMSK	Device All Endpoints Interrupt Mask Register	$81C_H$	U, PV	U, PV	Page 13-198
Reserved	Reserved	820_H-824_H	nBE	nBE	
DVBUSDIS	Device VBUS Discharge Time Register	828_H	U, PV	U, PV	Page 13-198
DVBUSPULSE	Device VBUS Pulsing Time Register	$82C_H$	U, PV	U, PV	Page 13-199
Reserved	Reserved	830_H	nBE	nBE	
DIEPEMPMSK	Device IN Endpoint FIFO Empty Interrupt Mask Register	834_H	U, PV	U, PV	Page 13-200
Reserved	Reserved	838_H-$8FC_H$	nBE	nBE	
DIEPCTL0	Device Control IN Endpoint 0 Control Register	900_H	U, PV	U, PV	Page 13-200
DIEPCTLx	Device Endpoint n Control Register	900_H + n*20_H	U, PV	U, PV	Page 13-205
Reserved	Reserved	904_H + n*20_H	nBE	nBE	
DIEPINTx	Device Endpoint-n Interrupt Register	908_H + n*20_H	U, PV	U, PV	Page 13-215
Reserved	Reserved	$90C_H$ + n*20_H	nBE	nBE	
DIEPTSIZ0	Device Endpoint 0 Transfer Size Register	910_H	U, PV	U, PV	Page 13-220

Table 13-13 Register Overview (cont'd)

Short Name	Description	Offset Addr.	Access Mode		Description See
			Read	Write	
DIEPTSIZx	Device Endpoint-n Transfer Size Register	910_H + n*20_H	U, PV	U, PV	Page 13-222
DIEPDMAx	Device Endpoint-n DMA Address Register	914_H + n*20_H	U, PV	U, PV	Page 13-226
DTXFSTSx	Device IN Endpoint Transmit FIFO Status Register	918_H + n*20_H	U, PV	U, PV	Page 13-228
DIEPDMABx	Device Endpoint-n DMA Buffer Address Register	$91C_H$ + n*20_H	U, PV	U, PV	Page 13-227
Reserved	Reserved	$9E0_H$-AFC_H	nBE	nBE	
DOEPCTL0	Device Control OUT Endpoint 0 Control Register	$B00_H$	U, PV	U, PV	Page 13-203
DOEPCTLx	Device Endpoint-n Control Register	$B00_H$ + n*20_H	U, PV	U, PV	Page 13-205
Reserved	Reserved	$B04_H$ + n*20_H	nBE	nBE	
DOEPINTx	Device Endpoint-n Interrupt Register	$B08_H$ + n*20_H	U, PV	U, PV	Page 13-215
Reserved	Reserved	$B0C_H$ + n*20_H	nBE	nBE	
DOEPTSIZ0	Device Endpoint 0 Transfer Size Register	$B10_H$	U, PV	U, PV	Page 13-220
DOEPTSIZx	Device Endpoint-n Transfer Size Register	$B10_H$ + n*20_H	U, PV	U, PV	Page 13-222
DOEPDMAx	Device Endpoint-n DMA Address Register	$B14_H$ + n*20_H	U, PV	U, PV	Page 13-226
Reserved	Reserved	$B18_H$ + n*20_H	nBE	nBE	
DOEPDMABx	Device Endpoint-n DMA Buffer Address Register	$B1C_H$ + n*20_H	U, PV	U, PV	Page 13-227
Reserved	Reserved	$BE0_H$-DFC_H	nBE	nBE	
USB Power and Gating Register					

Table 13-13 Register Overview (cont'd)

Short Name	Description	Offset Addr.	Access Mode		Description See
			Read	Write	
PCGCR	Power and Clock Gating Control Register	$E00_H$	U, PV	U, PV	Page 13-229
Reserved	Reserved	$E04_H$-FFC_H	nBE	nBE	

Data FIFO (DFIFO) Access Register Map

These registers are used to read or write the FIFO space for a specific endpoint in a given direction.

Table 13-14 Data FIFO (DFIFO) Access Register Map

FIFO Access Register Section	Address Range	Access
Device IN Endpoint 0: DFIFO Write Access Device OUT Endpoint 0: DFIFO Read Access	1000_H-$1FFC_H$	WO/RO
Device IN Endpoint 1: DFIFO Write Access Device OUT Endpoint 1: DFIFO Read Access	2000_H-$2FFC_H$	WO/RO
...	...	...
Device IN Endpoint 6: DFIFO Write Access Device OUT Endpoint 6: DFIFO Read Access	7000_H-$7FFC_H$	WO/RO

Access Restriction

Note: The USB registers are accessible only through word accesses. Half-word and byte accesses on USB registers will not generate a bus error. Write to unused address space will not cause an error but be ignored.

13.14.1 Register Description

This section describes Core Global, Device Mode, and Power and Clock Gating CSRs.

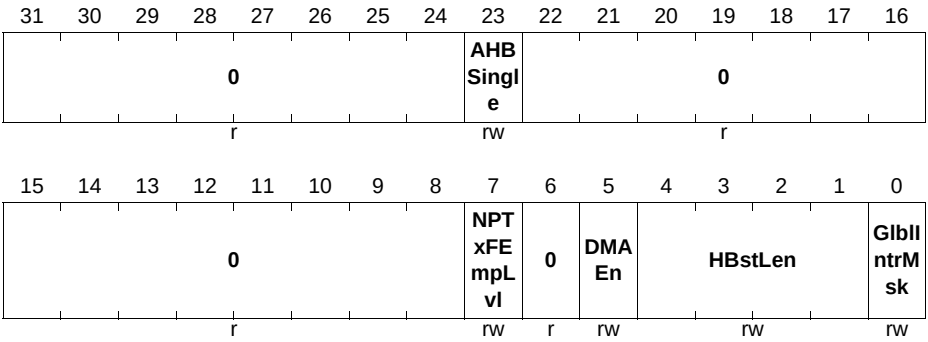
Note: Always program Reserved fields with 0s. Treat read values from Reserved fields as unknowns (Xs).

Global Registers

AHB Configuration Register (GAHBCFG)

This register can be used to configure the core after power-on or a change in mode. This register mainly contains AHB system-related configuration parameters. Do not change this register after the initial programming. The application must program this register before starting any transactions on either the AHB or the USB.

GAHBCFG
AHB Configuration Register **(008_H)** **Reset Value: 0000 0000_H**



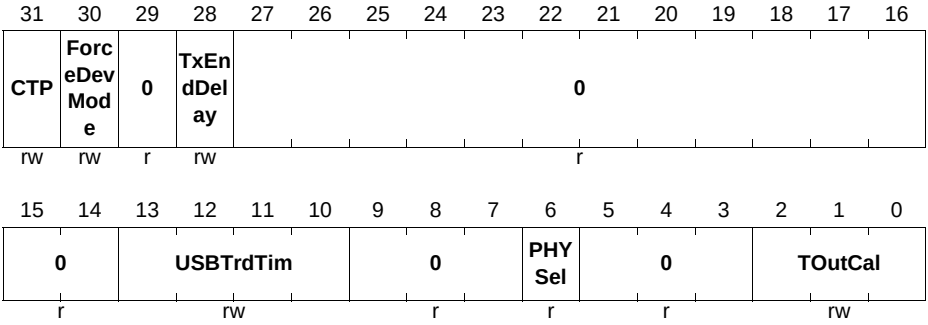
Field	Bits	Type	Description
GlblIntrMsk	0	rw	**Global Interrupt Mask** The application uses this bit to mask or unmask the interrupt line assertion to itself. Irrespective of this bit's setting, the interrupt status registers are updated by the core. 0_B Mask the interrupt assertion to the application. 1_B Unmask the interrupt assertion to the application.
HBstLen	[4:1]	rw	**Burst Length/Type** This field is used in DMA mode to indicate the AHB Master burst type. 0000_B Single 0001_B INCR 0011_B INCR4 0101_B INCR8 0111_B INCR16 Others: Reserved
DMAEn	5	rw	**DMA Enable** 0_B Core operates in Slave mode 1_B Core operates in a DMA mode

Field	Bits	Type	Description
NPTxFEmpLvl	7	rw	**Non-Periodic TxFIFO Empty Level** This bit indicates when IN endpoint Transmit FIFO empty interrupt (DIEPINTx.TxFEmp) is triggered. 0_B DIEPINTx.TxFEmp interrupt indicates that the IN Endpoint TxFIFO is half empty 1_B DIEPINTx.TxFEmp interrupt indicates that the IN Endpoint TxFIFO is completely empty This bit is used only in Device Slave mode.
AHBSingle	23	rw	**AHB Single Support** This bit when programmed supports single transfers for the remaining data in a transfer when the core is operating in DMA mode. 0_B The remaining data in a transfer is sent using INCR burst size. This is the default mode. 1_B The remaining data in a transfer is sent using single burst size.
0	[31:24], [22:8], 6	r	**Reserved** Read as 0; should be written with 0.

USB Configuration Register (GUSBCFG)

This register can be used to configure the core after power-on. It contains USB and USB-PHY related configuration parameters. The application must program this register before starting any transactions on either the AHB or the USB. Do not make changes to this register after the initial programming.

GUSBCFG
USB Configuration Register **($00C_H$)** **Reset Value: $0000\ 1440_H$**



Field	Bits	Type	Description
TOutCal	[2:0]	rw	**FS Timeout Calibration** The number of PHY clocks that the application programs in this field is added to the full-speed interpacket timeout duration in the core to account for any additional delays introduced by the PHY. This can be required, because the delay introduced by the PHY in generating the line state condition can vary from one PHY to another. The USB standard timeout value for full-speed operation is 16 to 18 (inclusive) bit times. The application must program this field based on the speed of enumeration. The number of bit times added per PHY clock is 0.25 bit times
PHYSel	6	r	**USB 1.1 Full-Speed Serial Transceiver Select** This bit is always read as 1 to indicate a full-speed transceiver is selected. 0_B Reserved 1_B USB 1.1 full-speed serial transceiver

Field	Bits	Type	Description
USBTrdTim	[13:10]	rw	**USB Turnaround Time** Sets the turnaround time in PHY clocks. Specifies the response time for a MAC request to the Packet FIFO Controller (PFC) to fetch data from the DFIFO (SPRAM). *Note: USB turnaround time is critical for certification where long cables and 5-Hubs are used. See "**"Choosing the Value of GUSBCFG.USBTrdTim" on Page 13-22**.* This bit is used in Device Only.
TxEndDelay	28	rw	**Tx End Delay** Writing a 1 to this bit enables the TxEndDelay timers in the core as per the section 4.1.5 on Opmode of the USB 2.0 Transceiver Macrocell Interface (UTMI) version 1.05. 0_B Normal mode 1_B Introduce Tx end delay timers This bit is used in Device Only.
ForceDevMode	30	rw	**Force Device Mode** Writing a 1 to this bit forces the core to device mode irrespective of connected plug. 0_B Normal Mode 1_B Force Device Mode After setting the force bit, the application must wait at least 25 ms before the change to take effect.
CTP	31	rw	**Corrupt Tx packet** This bit is for debug purposes only. Never set this bit to 1.
0	29, [27:16], [15:14], [9:7], [5:3]	r	**Reserved** Read as 0; should be written with 0.

Reset Register (GRSTCTL)

The application uses this register to reset various hardware features inside the core.

GRSTCTL
Reset Register **(010_H)** **Reset Value: $1000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AHBIdle	**DMAReq**	**0**													
r	r	r													

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0					**TxFNum**					**TxFFlsh**	**RxFFlsh**	**0**			**CSftRst**
r					rw					rwh	rwh	r			rwh

Field	Bits	Type	Description
CSftRst	0	rwh	**Core Soft Reset** Resets the hclk and phy_clock domains as follows: • Clears the interrupts and all the CSR registers except the following register bits: – PCGCCTL.GateHclk – GUSBCFG.PHYSel – DCFG.DevSpd – GGPIO • All module state machines (except the AHB Slave Unit) are reset to the IDLE state, and all the transmit FIFOs and the receive FIFO are flushed. • Any transactions on the AHB Master are terminated as soon as possible, after gracefully completing the last data phase of an AHB transfer. Any transactions on the USB are terminated immediately. The application can write to this bit any time it wants to reset the core. This is a self-clearing bit and the core clears this bit after all the necessary logic is reset in the core, which can take several clocks, depending on the current state of the core. Once this bit is cleared software must wait at least 3 PHY clocks before doing any access to the PHY domain (synchronization delay). Software must also must check that bit 31 of this register is 1 (AHB Master is IDLE) before starting any operation. Typically software reset is used during software development.
RxFFlsh	4	rwh	**RxFIFO Flush** The application can flush the entire RxFIFO using this bit, but must first ensure that the core is not in the middle of a transaction. The application must only write to this bit after checking that the core is neither reading from the RxFIFO nor writing to the RxFIFO. The application must wait until the bit is cleared before performing any other operations. This bit requires 8 clocks (slowest of PHY or AHB clock) to clear. This bit is set only by software and cleared only by hardware.

Field	Bits	Type	Description
TxFFlsh	5	rwh	**TxFIFO Flush** This bit selectively flushes a single or all transmit FIFOs, but cannot do so if the core is in the midst of a transaction. The application must write this bit only after checking that the core is neither writing to the TxFIFO nor reading from the TxFIFO. Verify using these registers: • Read—NAK Effective Interrupt ensures the core is not reading from the FIFO • Write—GRSTCTL.AHBIdle ensures the core is not writing anything to the FIFO. Flushing is normally recommended when FIFOs are re-configured. FIFO flushing is also recommended during device endpoint disable. The application must wait until the core clears this bit before performing any operations. This bit requires 8 clocks (slowest of PHY or AHB clock) to clear. This bit is set only by software and cleared only by hardware.
TxFNum	[10:6]	rw	**TxFIFO Number** This is the FIFO number that must be flushed using the TxFIFO Flush bit. This field must not be changed until the core clears the TxFIFO Flush bit. 00_H Tx FIFO 0 flush in device mode 01_H Tx FIFO 1 flush in device mode 02_H Tx FIFO 2 flush in device mode $\cdots_H$ ... $0F_H$ Tx FIFO 15 flush in device mode 10_H Flush all the transmit FIFOs in device or host mode.
DMAReq	30	r	**DMA Request Signal** Indicates that the DMA request is in progress. Used for debug.
AHBIdle	31	r	**AHB Master Idle** Indicates that the AHB Master State Machine is in the IDLE condition.
0	[29:11], [3:1]	r	**Reserved** Read as 0; should be written with 0.

Interrupt Register (GINTSTS)

This register interrupts the application for system-level events in the Device mode. It is shown in **Figure 13-47**.

Note: In the GINTSTS register, interrupt status bits with access type 'rwh' are set by hardware. To clear these bits, the application must write 1 into these bits.

The FIFO status interrupts are read only; once software reads from or writes to the FIFO while servicing these interrupts, FIFO interrupt conditions are cleared automatically.

The application must clear the GINTSTS register at initialization before unmasking the interrupt bit to avoid any interrupts generated prior to initialization.

GINTSTS
Interrupt Register **(014_H)** **Reset Value: 1400 0020$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
WkUpInt	**0**		**1**	**0**	**1**	**0**				**incompISOOUT**	**incompISOIN**	**OEPInt**	**IEPInt**	**0**	
rwh	r		r	r	r	r				rwh	rwh	r	r	r	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EOPF	**ISOOutDrop**	**EnumDone**	**USBRst**	**USBSusp**	**ErlySusp**	**0**		**GOUTNakEff**	**GINNakEff**	**1**	**RxFLvl**	**Sof**	**0**		**CurMod**
rwh	rwh	rwh	rwh	rwh	rwh	r		rh	rh	r	rh	rwh	r		rh

Field	Bits	Type	Description
CurMod	0	rh	**Current Mode of Operation** Indicates the current mode. 0_B Device mode 1_B Reserved
Sof	3	rwh	**Start of Frame** In Device mode, the core sets this bit to indicate that an SOF token has been received on the USB. The application can read the Device Status register to get the current frame number.
RxFLvl	4	rh	**RxFIFO Non-Empty** Indicates that there is at least one packet pending to be read from the RxFIFO.

Field	Bits	Type	Description
GINNakEff	6	rh	**Global IN Non-Periodic NAK Effective** Indicates that the Set Global Non-periodic IN NAK bit in the Device Control register (DCTL.SGNPInNak), set by the application, has taken effect in the core. That is, the core has sampled the Global IN NAK bit set by the application. This bit can be cleared by clearing the Clear Global Non-periodic IN NAK bit in the Device Control register (DCTL.CGNPInNak). This interrupt does not necessarily mean that a NAK handshake is sent out on the USB. The STALL bit takes precedence over the NAK bit.
GOUTNakEff	7	rh	**Global OUT NAK Effective** Indicates that the Set Global OUT NAK bit in the Device Control register (DCTL.SGOUTNak), set by the application, has taken effect in the core. This bit can be cleared by writing the Clear Global OUT NAK bit in the Device Control register (DCTL.CGOUTNak).
ErlySusp	10	rwh	**Early Suspend** The core sets this bit to indicate that an Idle state has been detected on the USB for 3 ms.
USBSusp	11	rwh	**USB Suspend** The core sets this bit to indicate that a suspend was detected on the USB. The core enters the Suspended state when there is no activity on the two USB data signals for an extended period of time.
USBRst	12	rwh	**USB Reset** The core sets this bit to indicate that a reset is detected on the USB.
EnumDone	13	rwh	**Enumeration Done** The core sets this bit to indicate that speed enumeration is complete. The application must read the Device Status (DSTS) register to obtain the enumerated speed.
ISOOutDrop	14	rwh	**Isochronous OUT Packet Dropped Interrupt** The core sets this bit when it fails to write an isochronous OUT packet into the RxFIFO because the RxFIFO does not have enough space to accommodate a maximum packet size packet for the isochronous OUT endpoint.

Field	Bits	Type	Description
EOPF	15	rwh	**End of Periodic Frame Interrupt** Indicates that the period specified in the Periodic Frame Interval field of the Device Configuration register (DCFG.PerFrInt) has been reached in the current frame.
IEPInt	18	r	**IN Endpoints Interrupt** The core sets this bit to indicate that an interrupt is pending on one of the IN endpoints of the core (in Device mode). The application must read the Device All Endpoints Interrupt (DAINT) register to determine the exact number of the IN endpoint on which the interrupt occurred, and then read the corresponding Device IN Endpoint-n Interrupt (DIEPINTx) register to determine the exact cause of the interrupt. The application must clear the appropriate status bit in the corresponding DIEPINTx register to clear this bit.
OEPInt	19	r	**OUT Endpoints Interrupt** The core sets this bit to indicate that an interrupt is pending on one of the OUT endpoints of the core (in Device mode). The application must read the Device All Endpoints Interrupt (DAINT) register to determine the exact number of the OUT endpoint on which the interrupt occurred, and then read the corresponding Device OUT Endpoint-n Interrupt (DOEPINTx) register to determine the exact cause of the interrupt. The application must clear the appropriate status bit in the corresponding DOEPINTx register to clear this bit.
incompISOIN	20	rwh	**Incomplete Isochronous IN Transfer** The core sets this interrupt to indicate that there is at least one isochronous IN endpoint on which the transfer is not completed in the current frame. This interrupt is asserted along with the End of Periodic Frame Interrupt (EOPF) bit in this register. *Note: This interrupt is not asserted in Scatter/Gather DMA mode.*

Field	Bits	Type	Description
incomplSOOUT	21	rwh	**Incomplete Isochronous OUT Transfer** In the Device mode, the core sets this interrupt to indicate that there is at least one isochronous OUT endpoint on which the transfer is not completed in the current frame. This interrupt is asserted along with the End of Periodic Frame Interrupt (EOPF) bit in this register.
WkUpInt	31	rwh	**Resume/Remote Wakeup Detected Interrupt** Wakeup Interrupt during Suspend state. • Device Mode - This interrupt is asserted only when Host Initiated Resume is detected on USB.
1	28, 26, 5	r	**Reserved** Read as 1; should be written with 1.
0	[30:29], 27, [25:22], [17:16], [9:8], [2:1]	r	**Reserved** Read as 0; should be written with 0.

Interrupt Mask Register (GINTMSK)

This register works with the Interrupt Register (**“Interrupt Register (GINTSTS)” on Page 13-175**) to interrupt the application. When an interrupt bit is masked, the interrupt associated with that bit is not generated. However, the GINTSTS register bit corresponding to that interrupt is still set.

- Mask interrupt: 0_B
- Unmask interrupt: 1_B

GINTMSK
Interrupt Mask Register **(018_H)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
WkUpIntMsk	0									incomplSOOUTMsk	incomplSOINMsk	OEPIntMsk	IEPIntMsk	0	
rw	r									rw	rw	rw	rw	r	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EOPFMsk	ISOOutDropMsk	EnumDoneMsk	USBRstMsk	USBSuspMsk	ErlySuspMsk	0		GOUTNakEffMsk	GINNakEffMsk	0	RxFLvlMsk	SofMsk	0		
rw	rw	rw	rw	rw	rw	r		rw	rw	r	rw	rw	r		

Field	Bits	Type	Description
SofMsk	3	rw	**Start of Frame Mask**
RxFLvlMsk	4	rw	**Receive FIFO Non-Empty Mask**
GINNakEffMsk	6	rw	**Global Non-periodic IN NAK Effective Mask**
GOUTNakEffMsk	7	rw	**Global OUT NAK Effective Mask**
ErlySuspMsk	10	rw	**Early Suspend Mask**
USBSuspMsk	11	rw	**USB Suspend Mask**
USBRstMsk	12	rw	**USB Reset Mask**
EnumDoneMsk	13	rw	**Enumeration Done Mask**
ISOOutDropMsk	14	rw	**Isochronous OUT Packet Dropped Interrupt Mask**
EOPFMsk	15	rw	**End of Periodic Frame Interrupt Mask** Mode: Device only Reset: 0_B
IEPIntMsk	18	rw	**IN Endpoints Interrupt Mask**
OEPIntMsk	19	rw	**OUT Endpoints Interrupt Mask**

Field	Bits	Type	Description
incomplSOINMsk	20	rw	**Incomplete Isochronous IN Transfer Mask**
incomplSOOUTMsk	21	rw	**Incomplete Isochronous OUT Transfer Mask**
WkUpIntMsk	31	rw	**Resume/Remote Wakeup Detected Interrupt Mask**
0	[30:22], [17:16], [9:8], 5, [2:0]	r	**Reserved** Read as 0; should be written with 0.

Receive Status Debug Read/Status Read and Pop Registers (GRXSTSR/GRXSTSP)

A read to the Receive Status Debug Read register returns the contents of the top of the Receive FIFO. A read to the Receive Status Read and Pop register additionally pops the top data entry out of the RxFIFO.

The core ignores the receive status pop/read when the receive FIFO is empty and returns a value of $0000'0000_H$. The application must only pop the Receive Status FIFO when the Receive FIFO Non-Empty bit of the Core Interrupt register (GINTSTS.RxFLvl) is asserted.

Notes

1. *Do not read this register's reset value before configuring the core because the read value is "X".*

Receive Status Debug Read/Status Read and Pop Registers in Device Mode

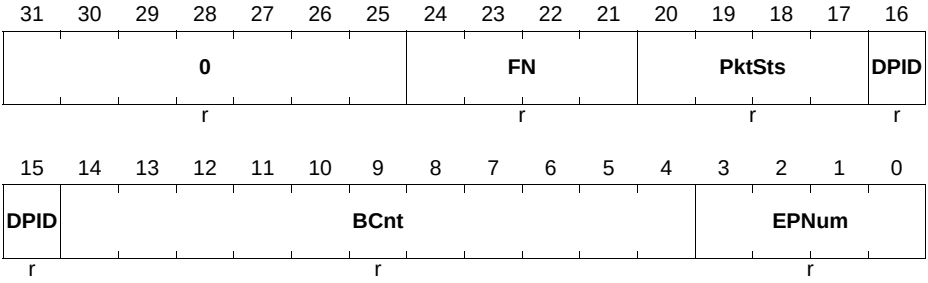
(GRXSTSR/GRXSTSP)

GRXSTSR
Receive Status Debug Read Register

($01C_H$) **Reset Value: $0000\ 0000_H$**

GRXSTSP
Receive Status Read and Pop Register

(020_H) **Reset Value: $0000\ 0000_H$**



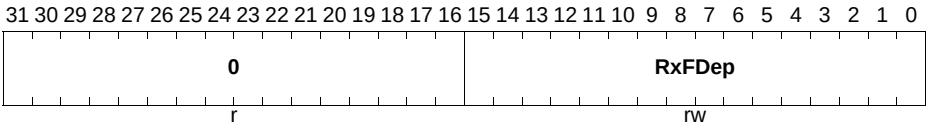
Field	Bits	Type	Description
EPNum	[3:0]	r	**Endpoint Number** Indicates the endpoint number to which the current received packet belongs.
BCnt	[14:4]	r	**Byte Count** Indicates the byte count of the received data packet.
DPID	[16:15]	r	**Data PID** Indicates the Data PID of the received OUT data packet 00_B DATA0 10_B DATA1 01_B DATA2 11_B MDATA
PktSts	[20:17]	r	**Packet Status** Indicates the status of the received packet 0001_B Global OUT NAK (triggers an interrupt) 0010_B OUT data packet received 0011_B OUT transfer completed (triggers an interrupt) 0100_B SETUP transaction completed (triggers an interrupt) 0110_B SETUP data packet received Others: Reserved

Field	Bits	Type	Description
FN	[24:21]	r	**Frame Number** This is the least significant 4 bits of the frame number in which the packet is received on the USB. This field is supported only when isochronous OUT endpoints are supported.
0	[31:25]	r	**Reserved** Read as 0; should be written with 0.

Receive FIFO Size Register (GRXFSIZ)

The application can program the RAM size that must be allocated to the RxFIFO.

GRXFSIZ
Receive FIFO Size Register **(024_H)** **Reset Value: 0000 011A$_H$**

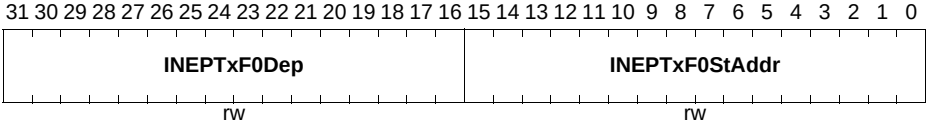


Field	Bits	Type	Description
RxFDep	[15:0]	rw	**RxFIFO Depth** This value is in terms of 32-bit words. • Minimum value is 16 • Maximum value is 282 Programmed values must not exceed the maximum value.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

Non-Periodic Transmit FIFO Size Register (GNPTXFSIZ)

The application can program the RAM size for IN Endpoint TxFIFO 0.

GNPTXFSIZ
Non-Periodic Transmit FIFO Size Register
(028_H) **Reset Value: 0010 011A$_H$**



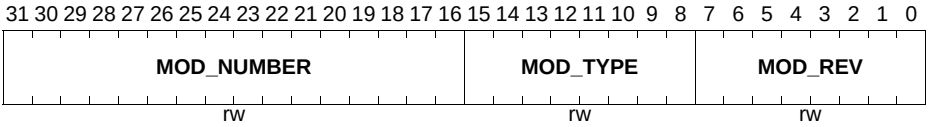
Field	Bits	Type	Description
INEPTxF0 StAddr	[15:0]	rw	**IN Endpoint FIFO0 Transmit RAM Start Address** This field contains the memory start address for IN Endpoint Transmit FIFO0. Programmed values must not exceed the power-on value.
INEPTxF0 Dep	[31:16]	rw	**IN Endpoint TxFIFO 0 Depth** This value is in terms of 32-bit words. • Minimum value is 16 • Maximum value is 16 Programmed values must not exceed the maximum value.

USB Module Identification Register (GUID)

This register contains the USB module version and revision and should not be overwritten by application.

GUID

USB Module Identification Register (03C$_H$) **Reset Value: 00B1 C0XX$_H$**



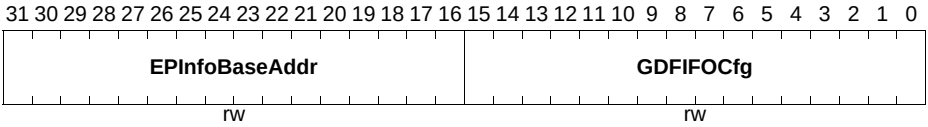
Field	Bits	Type	Description
MOD_REV	[7:0]	rw	**Module Revision** Indicates the revision number of the implementation. This information depends on the design step.
MOD_TYPE	[15:8]	rw	**Module Type** This internal marker is fixed to C0$_H$.
MOD_NUMBER	[31:16]	rw	**Module Number** Indicates the module identification number.

Global DFIFO Software Config Register (GDFIFOCFG)

This register needs to be configured only when DMA mode is used. In slave (non-DMA) mode, it can be ignored.

Note: The reset value of the register does not represent the implemented FIFO RAM size.

GDFIFOCFG
Global DFIFO Software Config Register(05C$_H$) **Reset Value: 027A 02B2$_H$**



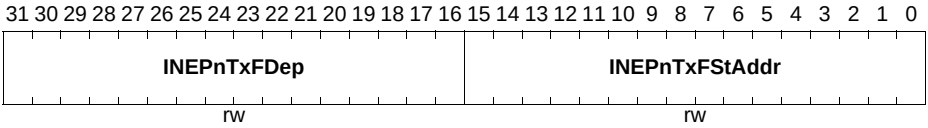
Field	Bits	Type	Description
GDFIFOCfg	[15:0]	rw	**GDFIFOCfg** This field is for dynamic programming of the DFIFO Size. This value takes effect only when the application programs a non zero value to this register. The value programmed must conform to the guidelines described in **"Data FIFO RAM Allocation" on Page 13-154**. The USB core does not have any corrective logic if the FIFO sizes are programmed incorrectly.
EPInfoBaseAddr	[31:16]	rw	**EPInfoBaseAddr** This field provides the start address of the RAM space allocated to store register information in DMA mode. See **"Data FIFO RAM Allocation" on Page 13-154**.

Device IN Endpoint Transmit FIFO Size Register (DIEPTXFn)

This register holds the size and memory start address of IN endpoint TxFIFOs implemented in Device mode. Each FIFO holds the data for one IN endpoint. This register is repeated for instantiated IN endpoint FIFOs 1 to 6. For IN endpoint FIFO 0 use GNPTXFSIZ register for programming the size and memory start address.

Note: The reset value of the register serves as a limit value and does not represent the implemented FIFO RAM size. The application must configure these registers in a way that the implemented FIFO RAM size is not exceeded.

DIEPTXF1
Device IN Endpoint 1 Transmit FIFO Size Register(104_H) Reset Value: 0100 $012A_H$
DIEPTXF2
Device IN Endpoint 2 Transmit FIFO Size Register(108_H) Reset Value: 0100 $022A_H$
DIEPTXF3
Device IN Endpoint 3 Transmit FIFO Size Register($10C_H$) Reset Value: 0100 $032A_H$
DIEPTXF4
Device IN Endpoint 4 Transmit FIFO Size Register(110_H) Reset Value: 0100 $042A_H$
DIEPTXF5
Device IN Endpoint 5 Transmit FIFO Size Register(114_H) Reset Value: 0100 $052A_H$
DIEPTXF6
Device IN Endpoint 6 Transmit FIFO Size Register(118_H) Reset Value: 0100 $062A_H$



Field	Bits	Type	Description
INEPnTxFStAddr	[15:0]	rw	**IN Endpoint FIFOn Transmit RAM Start Address** This field contains the memory start address for IN endpoint Transmit FIFOn (1 < n ≤ 6). Programmed values must not exceed the reset value.
INEPnTxFDep	[31:16]	rw	**IN Endpoint TxFIFO Depth** This value is in terms of 32-bit words. • Minimum value is 16 • Maximum value is 256 Programmed values must not exceed the maximum value.

Device Mode Registers

These registers are visible only in Device mode and must not be accessed in Host mode, as the results are unknown. Some of them affect all the endpoints uniformly, while others affect only a specific endpoint. Device Mode registers fall into two categories:

Device Logical IN Endpoint-Specific Registers

One set of endpoint registers is instantiated per logical endpoint. A logical endpoint is unidirectional: it can be either IN or OUT. To represent a bidirectional endpoint, two logical endpoints are required, one for the IN direction and the other for the OUT direction. This is also true for control endpoints.

The registers and register fields described in this section can pertain to IN or OUT endpoints, or both, or specific endpoint types as noted.

Device Configuration Register (DCFG)

This register configures the core in Device mode after power-on or after certain control commands or enumeration. Do not make changes to this register after initial programming.

DCFG
Device Configuration Register (800_H) Reset Value: $0820\ 0000_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0				1	0	PerSchIntvl		DescDMA	0	1	0			0	
r				r	r	rw		rw	r	r	r			r	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0			PerFrInt		DevAddr							0	NZStsOUTHShk	DevSpd	
r			rw		rw							r	rw	rw	

Field	Bits	Type	Description
DevSpd	[1:0]	rw	**Device Speed** Indicates the speed at which the application requires the core to enumerate, or the maximum speed the application can support. However, the actual bus speed is determined only after the chirp sequence is completed, and is based on the speed of the USB host to which the core is connected. See **“Device Initialization” on Page 13-11** for details. 00_B Reserved 01_B Reserved 10_B Reserved 11_B Full speed (USB 1.1 transceiver clock is 48 MHz)
NZStsOUTHShk	2	rw	**Non-Zero-Length Status OUT Handshake** The application can use this field to select the handshake the core sends on receiving a nonzero-length data packet during the OUT transaction of a control transfer's Status stage. 1_B Send a STALL handshake on a nonzero-length status OUT transaction and do not send the received OUT packet to the application. 0_B Send the received OUT packet to the application (zero-length or nonzero-length) and send a handshake based on the NAK and STALL bits for the endpoint in the Device Endpoint Control register.
DevAddr	[10:4]	rw	**Device Address** The application must program this field after every SetAddress control command.
PerFrInt	[12:11]	rw	**Periodic Frame Interval** Indicates the time within a frame at which the application must be notified using the End Of Periodic Frame Interrupt. This can be used to determine if all the isochronous traffic for that frame is complete. 00_B 80% of the frame interval 01_B 85% 10_B 90% 11_B 95%

Field	Bits	Type	Description
DescDMA	23	rw	**Enable Scatter/Gather DMA in Device mode.** The application can set this bit during initialization to enable the Scatter/Gather DMA operation. *Note: This bit must be modified only once after a reset.* The following combinations are available for programming: • GAHBCFG.DMAEn=0,DCFG.DescDMA=0 => Slave mode • GAHBCFG.DMAEn=0,DCFG.DescDMA=1 => Invalid • GAHBCFG.DMAEn=1 ,DCFG.DescDMA=0 => Buffered DMA mode • GAHBCFG.DMAEn=1 ,DCFG.DescDMA=1 => Scatter/Gather DMA mode
PerSchIntvl	[25:24]	rw	**Periodic Scheduling Interval** PerSchIntvl must be programmed only for Scatter/Gather DMA mode. Description: This field specifies the amount of time the Internal DMA engine must allocate for fetching periodic IN endpoint data. Based on the number of periodic endpoints, this value must be specified as 25, 50 or 75% of frame. • When any periodic endpoints are active, the internal DMA engine allocates the specified amount of time in fetching periodic IN endpoint data. • When no periodic endpoints are active, then the internal DMA engine services non-periodic endpoints, ignoring this field. • After the specified time within a frame, the DMA switches to fetching for non-periodic endpoints. 00_B 25% of frame. 01_B 50% of frame. 10_B 75% of frame. 11_B Reserved.

Field	Bits	Type	Description
1	27, 21	r	**Reserved** Read as 1; should be written with 1.
0	[31:28], 26, 22, [20:18], [17:13], 3	r	**Reserved** Read as 0; should be written with 0.

Device Control Register (DCTL)

DCTL
Device Control Register **(804_H)** **Reset Value: 0000 0002$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0														EnContOnBNA	NakOnBble
r														rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IgnrFrmNum	GMC		0		CGOUTNak	SGOUTNak	CGNPInNak	SGNPInNak	0			GOUTNakSts	GNPINNakSts	SftDiscon	RmtWkUpSig
rw	rw		r		w	w	w	w	r			rh	rh	rw	rw

Field	Bits	Type	Description
RmtWkUpSig	0	rw	**Remote Wakeup Signaling** When the application sets this bit, the core initiates remote signaling to wake the USB host. The application must set this bit to instruct the core to exit the Suspend state. As specified in the USB 2.0 specification, the application must clear this bit 1 to15 ms after setting it.
SftDiscon	1	rw	**Soft Disconnect** The application uses this bit to signal the USB core to do a soft disconnect. As long as this bit is set, the host does not see that the device is connected, and the device does not receive signals on the USB. The core stays in the disconnected state until the application clears this bit. The minimum duration for which the core must keep this bit set is specified in **Table 13-15**. 0_B Normal operation. When this bit is cleared after a soft disconnect, the core drives a device connect event to the USB host. When the device is reconnected, the USB host restarts device enumeration. 1_B The core drives a device disconnect event to the USB host.
GNPINNakSts	2	rh	**Global Non-periodic IN NAK Status** 0_B A handshake is sent out based on the data availability in the transmit FIFO. 1_B A NAK handshake is sent out on all non-periodic IN endpoints, irrespective of the data availability in the transmit FIFO.
GOUTNakSts	3	rh	**Global OUT NAK Status** 0_B A handshake is sent based on the FIFO Status and the NAK and STALL bit settings. 1_B No data is written to the RxFIFO, irrespective of space availability. Sends a NAK handshake on all packets, except on SETUP transactions. All isochronous OUT packets are dropped.

Field	Bits	Type	Description
SGNPInNak	7	w	**Set Global Non-periodic IN NAK** A write to this field sets the Global Non-periodic IN NAK.The application uses this bit to send a NAK handshake on all non-periodic IN endpoints. The core can also set this bit when a timeout condition is detected on a non-periodic endpoint in shared FIFO operation. The application must set this bit only after making sure that the Global IN NAK Effective bit in the Core Interrupt Register (GINTSTS.GINNakEff) is cleared.
CGNPInNak	8	w	**Clear Global Non-periodic IN NAK** A write to this field clears the Global Non-periodic IN NAK.
SGOUTNak	9	w	**Set Global OUT NAK** A write to this field sets the Global OUT NAK. The application uses this bit to send a NAK handshake on all OUT endpoints. The application must set the this bit only after making sure that the Global OUT NAK Effective bit in the Core Interrupt Register (GINTSTS.GOUTNakEff) is cleared.
CGOUTNak	10	w	**Clear Global OUT NAK** A write to this field clears the Global OUT NAK.
GMC	[14:13]	rw	**Global Multi Count** GMC must be programmed only once after initialization. Applicable only for Scatter/Gather DMA mode. This indicates the number of packets to be serviced for that end point before moving to the next end point. It is only for non-periodic end points. 00_B Invalid. 01_B 1 packet. 10_B 2 packets. 11_B 3 packets. When Scatter/Gather DMA mode is disabled, this field is reserved. and reads 00_B.

Field	Bits	Type	Description
IgnrFrmNum	15	rw	**Ignore frame number for isochronous endpoints in case of Scatter/Gather DMA** *Note: When this bit is enabled, there must be only one packet per descriptor.* In Scatter/Gather DMA mode, when this bit is enabled, the packets are not flushed when an ISOC IN token is received for an elapsed frame. When Scatter/Gather DMA mode is disabled, this field is used by the application to enable periodic transfer interrupt. The application can program periodic endpoint transfers for multiple frames. 0_B Scatter/Gather enabled: The core transmits the packets only in the frame number in which they are intended to be transmitted. Scatter/Gather disabled: Periodic transfer interrupt feature is disabled; the application must program transfers for periodic endpoints every frame 1_B Scatter/Gather enabled: The core ignores the frame number, sending packets immediately as the packets are ready. Scatter/Gather disabled: Periodic transfer interrupt feature is enabled; the application can program transfers for multiple frames for periodic endpoints. In non-Scatter/Gather DMA mode, the application receives transfer complete interrupt after transfers for multiple frames are completed.
NakOnBble	16	rw	**Set NAK automatically on babble** The core sets NAK automatically for the endpoint on which babble is received.

Field	Bits	Type	Description
EnContOnBNA	17	rw	**Enable continue on BNA** This bit enables the core to continue on BNA for Bulk OUT endpoints. With this feature enabled, when a Bulk OUT endpoint receives a BNA interrupt the core starts processing the descriptor that caused the BNA interrupt after the endpoint re-enables the endpoint. 0_B After receiving BNA interrupt, the core disables the endpoint. When the endpoint is re-enabled by the application, the core starts processing from the DOEPDMA descriptor. 1_B After receiving BNA interrupt, the core disables the endpoint. When the endpoint is re-enabled by the application, the core starts processing from the descriptor that received the BNA interrupt.
0	[31:18], [12:11], [6:4]	r	**Reserved** Read as 0; should be written with 0.

Table 13-15 lists the minimum duration under various conditions for which the Soft Disconnect (SftDiscon) bit must be set for the USB host to detect a device disconnect. To accommodate clock jitter, it is recommended that the application add some extra delay to the specified minimum duration.

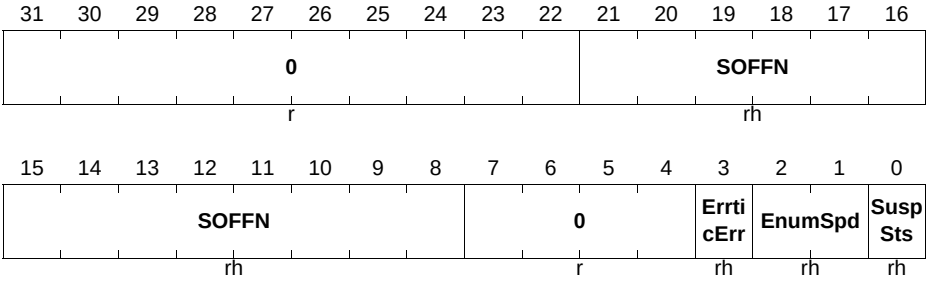
Table 13-15 Minimum Duration for Soft Disconnect

Operating Speed	Device State	Minimum Duration
Full speed	Suspended	1 ms + 2.5 µs
Full speed	Idle	2.5 µs
Full speed	Not Idle or Suspended (Performing transactions)	2.5 µs

Device Status Register (DSTS)

This register indicates the status of the core with respect to USB-related events. It must be read on interrupts from Device All Interrupts (DAINT) register.

DSTS
Device Status Register **(808_H)** **Reset Value: $0000\ 0002_H$**



Field	Bits	Type	Description
SuspSts	0	rh	**Suspend Status** In Device mode, this bit is set as long as a Suspend condition is detected on the USB. The core enters the Suspended state when there is no activity on the two USB data signals for an extended period of time. The core comes out of the suspend: • When there is any activity on the two USB data signals • When the application writes to the Remote Wakeup Signaling bit in the Device Control register (DCTL.RmtWkUpSig)
EnumSpd	[2:1]	rh	**Enumerated Speed** Indicates the speed at which the USB core has come up after speed detection through a chirp sequence. 00_B Reserved 01_B Reserved 10_B Reserved 11_B Full speed (PHY clock is running at 48 MHz)
ErrticErr	3	rh	**Erratic Error** The core sets this bit to report any erratic errors. Due to erratic errors, the USB core goes into Suspended state and an interrupt is generated to the application with Early Suspend bit of the Core Interrupt register (GINTSTS.ErlySusp). If the early suspend is asserted due to an erratic error, the application can only perform a soft disconnect recover.

Field	Bits	Type	Description
SOFFN	[21:8]	rh	**Frame Number of the Received SOF** When the core is operating at full speed, this field contains a frame number.
0	[31:22], [7:4]	r	**Reserved** Read as 0; should be written with 0.

Device IN Endpoint Common Interrupt Mask Register (DIEPMSK)

This register works with each of the Device IN Endpoint Interrupt (DIEPINTx) registers for all endpoints to generate an interrupt per IN endpoint. The IN endpoint interrupt for a specific status in the DIEPINTx register can be masked by writing to the corresponding bit in this register. Status bits are masked by default.

- Mask interrupt: 0_B
- Unmask interrupt: 1_B

DIEPMSK
Device IN Endpoint Common Interrupt Mask Register **(810_H)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		NAKMsk	0			BNAIntrMsk	TxfifoUndrnMsk	0	INEPNakEffMsk	0	INTknTXFEmpMsk	TimeOUTMsk	AHBErrMsk	EPDisbldMsk	XferComplMsk
r		rw	r			rw	rw	r	rw	r	rw	rw	rw	rw	rw

Field	Bits	Type	Description
XferComplMsk	0	rw	**Transfer Completed Interrupt Mask**
EPDisbldMsk	1	rw	**Endpoint Disabled Interrupt Mask**
AHBErrMsk	2	rw	**AHB Error Mask**
TimeOUTMsk	3	rw	**Timeout Condition Mask** (Non-isochronous endpoints)

Field	Bits	Type	Description
INTknTXFEmpMsk	4	rw	**IN Token Received When TxFIFO Empty Mask**
INEPNakEffMsk	6	rw	**IN Endpoint NAK Effective Mask**
TxfifoUndrnMsk	8	rw	**Fifo Underrun Mask**
BNAInIntrMsk	9	rw	**BNA Interrupt Mask**
NAKMsk	13	rw	**NAK interrupt Mask**
0	[31:14], [12:10], 7, 5	r	**Reserved** Read as 0; should be written with 0.

Device OUT Endpoint Common Interrupt Mask Register (DOEPMSK)

This register works with each of the Device OUT Endpoint Interrupt (DOEPINTx) registers for all endpoints to generate an interrupt per OUT endpoint. The OUT endpoint interrupt for a specific status in the DOEPINTx register can be masked by writing into the corresponding bit in this register. Status bits are masked by default.

- Mask interrupt: 0_B
- Unmask interrupt: 1_B

DOEPMSK
Device OUT Endpoint Common Interrupt Mask Register **(814_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	NYETMsk	NAKMsk	BbleErrMsk	0		BnaOutIntrMsk	OutPktErrMsk	0	Back2BackSETup	0	OUTTknEPdisMsk	SetUPMsk	AHBErrMsk	EPDisbldMsk	XferComplMsk
r	rw	rw	rw	r		rw	rw	r	rw	r	rw	rw	rw	rw	rw

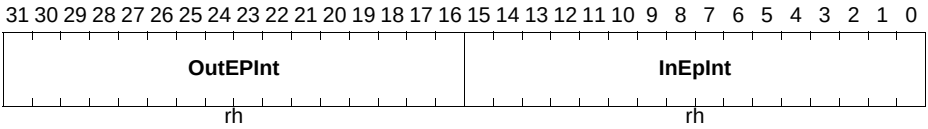
Field	Bits	Type	Description
XferComplMsk	0	rw	**Transfer Completed Interrupt Mask**
EPDisbldMsk	1	rw	**Endpoint Disabled Interrupt Mask**

Field	Bits	Type	Description
AHBErrMsk	2	rw	**AHB Error**
SetUPMsk	3	rw	**SETUP Phase Done Mask** Applies to control endpoints only.
OUTTknEPdisMsk	4	rw	**OUT Token Received when Endpoint Disabled Mask** Applies to control OUT endpoints only.
Back2BackSETup	6	rw	**Back-to-Back SETUP Packets Received Mask** Applies to control OUT endpoints only.
OutPktErrMsk	8	rw	**OUT Packet Error Mask**
BnaOutIntrMsk	9	rw	**BNA interrupt Mask**
BbleErrMsk	12	rw	**Babble Interrupt Mask**
NAKMsk	13	rw	**NAK Interrupt Mask**
NYETMsk	14	rw	**NYET Interrupt Mask**
0	[31:15], [11:10], 7, 5	r	**Reserved** Read as 0; should be written with 0.

Device All Endpoints Interrupt Register (DAINT)

When a significant event occurs on an endpoint, a Device All Endpoints Interrupt register interrupts the application using the Device OUT Endpoints Interrupt bit or Device IN Endpoints Interrupt bit of the Core Interrupt register (GINTSTS.OEPInt or GINTSTS.IEPInt, respectively). This is shown in **Figure 13-47 “Interrupt Hierarchy” on Page 13-159**. There is one interrupt bit per endpoint, up to a maximum of 7 bits for OUT endpoints and 7 bits for IN endpoints. For a bidirectional endpoint, the corresponding IN and OUT interrupt bits are used. Bits in this register are set and cleared when the application sets and clears bits in the corresponding Device Endpoint-n Interrupt register (DIEPINTx/DOEPINTx).

DAINT
Device All Endpoints Interrupt Register(818_H) **Reset Value: $0000\ 0000_H$**



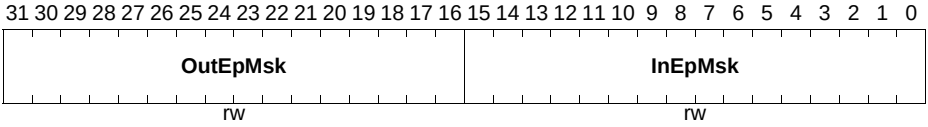
Field	Bits	Type	Description
InEpInt	[15:0]	rh	**IN Endpoint Interrupt Bits** One bit per IN Endpoint: Bit 0 for IN endpoint 0, bit 6 for endpoint 6. Bits [15:7] are not used.
OutEPInt	[31:16]	rh	**OUT Endpoint Interrupt Bits** One bit per OUT endpoint: Bit 16 for OUT endpoint 0, bit 22 for OUT endpoint 6. Bits [31:23] are not used.

Device All Endpoints Interrupt Mask Register (DAINTMSK)

The Device Endpoint Interrupt Mask register works with the Device Endpoint Interrupt register to interrupt the application when an event occurs on a device endpoint. However, the Device All Endpoints Interrupt (DAINT) register bit corresponding to that interrupt is still set.

- Mask Interrupt: 0_B
- Unmask Interrupt: 1_B

DAINTMSK
Device All Endpoints Interrupt Mask Register($81C_H$) **Reset Value: 0000 0000$_H$**

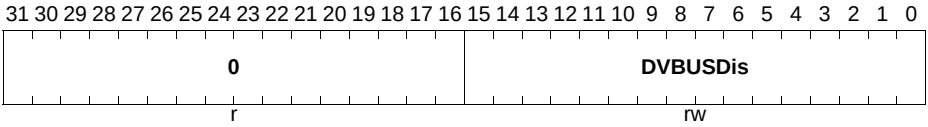


Field	Bits	Type	Description
InEpMsk	[15:0]	rw	**IN EP Interrupt Mask Bits** One bit per IN Endpoint: Bit 0 for IN EP 0, bit 6 for IN EP 6. Bits [15:7] are not used.
OutEpMsk	[31:16]	rw	**OUT EP Interrupt Mask Bits** One per OUT Endpoint: Bit 16 for OUT EP 0, bit 22 for OUT EP 6. Bits [31:23] are not used.

Device VBUS Discharge Time Register (DVBUSDIS)

This register specifies the VBUS discharge time after VBUS pulsing during SRP.

DVBUSDIS
Device VBUS Discharge Time Register(828_H) **Reset Value: 0000 17D7$_H$**

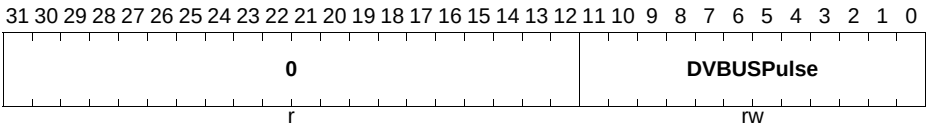


Field	Bits	Type	Description
DVBUSDis	[15:0]	rw	**Device Vbus Discharge Time** Specifies the Vbus discharge time after Vbus pulsing during SRP. This value equals: Vbus discharge time in PHY clocks / 1,024 The reset value is based on PHY operating at 60 MHz. Depending on the Vbus load, this value might need adjustment.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

Device VBUS Pulsing Time Register (DVBUSPULSE)

This register specifies the VBUS pulsing time during SRP.

DVBUSPULSE
Device VBUS Pulsing Time Register ($82C_H$) **Reset Value: 0000 05B8$_H$**



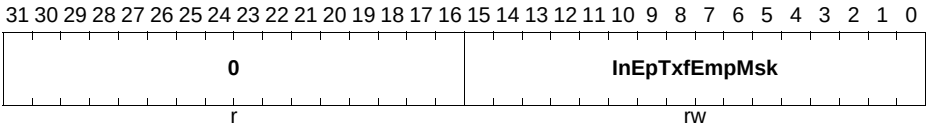
Field	Bits	Type	Description
DVBUSPulse	[11:0]	rw	**Device Vbus Pulsing Time** Specifies the Vbus pulsing time during SRP. This value equals: Vbus pulsing time in PHY clocks / 1,024 The reset value is based on PHY operating at 60 MHz.
0	[31:12]	r	**Reserved** Read as 0; should be written with 0.

Device IN Endpoint FIFO Empty Interrupt Mask Register (DIEPEMPMSK)

This register is used to control the IN endpoint FIFO empty interrupt generation (DIEPINTx.TxfEmp).

- Mask interrupt: 0_B
- Unmask interrupt: 1_B

DIEPEMPMSK
Device IN Endpoint FIFO Empty Interrupt Mask Register **(834_H)** **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
InEpTxfEmpMsk	[15:0]	rw	**IN EP Tx FIFO Empty Interrupt Mask Bits** These bits acts as mask bits for DIEPINTx.TxFEmp interrupt One bit per IN Endpoint: • Bit 0 for IN endpoint 0 • ... • Bit 6 for endpoint 6 Bits [15:7] are not used.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

Device Control IN Endpoint 0 Control Register (DIEPCTL0)

This section describes the Control IN Endpoint 0 Control register. Non-zero control endpoints use registers for endpoints 1-6.

DIEPCTL0

Device Control IN Endpoint 0 Control Register(900_H) **Reset Value: $0000\ 8000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EPEna	EPDis	0		SNAK	CNAK	TxFNum				Stall	0	EPType		NAKSts	0
rwh	rwh	r		w	w	rw				rwh	r	r		rh	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
USBActEP	0													MPS	
r	r													rw	

Field	Bits	Type	Description
MPS	[1:0]	rw	**Maximum Packet Size** Applies to IN and OUT endpoints. The application must program this field with the maximum packet size for the current logical endpoint. 00_B 64 bytes 01_B 32 bytes 10_B 16 bytes 11_B 8 bytes
USBActEP	15	r	**USB Active Endpoint** This bit is always set to 1, indicating that control endpoint 0 is always active in all configurations and interfaces.
NAKSts	17	rh	**NAK Status** Indicates the following: 0_B The core is transmitting non-NAK handshakes based on the FIFO status 1_B The core is transmitting NAK handshakes on this endpoint. When this bit is set, either by the application or core, the core stops transmitting data, even if there is data available in the TxFIFO. Irrespective of this bit's setting, the core always responds to SETUP data packets with an ACK handshake.
EPType	[19:18]	r	**Endpoint Type** Hardcoded to 00_B for control.

Field	Bits	Type	Description
Stall	21	rwh	**STALL Handshake** The application can only set this bit, and the core clears it, when a SETUP token is received for this endpoint. If a NAK bit, Global Non-periodic IN NAK, or Global OUT NAK is set along with this bit, the STALL bit takes priority. This bit is set only by software and cleared only by hardware.
TxFNum	[25:22]	rw	**TxFIFO Number** • This value is set to the FIFO number that is assigned to IN Endpoint 0.
CNAK	26	w	**Clear NAK** A write to this bit clears the NAK bit for the endpoint.
SNAK	27	w	**Set NAK** A write to this bit sets the NAK bit for the endpoint. Using this bit, the application can control the transmission of NAK handshakes on an endpoint. The core can also set this bit for an endpoint after a SETUP packet is received on that endpoint.
EPDis	30	rwh	**Endpoint Disable** The application sets this bit to stop transmitting data on an endpoint, even before the transfer for that endpoint is complete. The application must wait for the Endpoint Disabled interrupt before treating the endpoint as disabled. The core clears this bit before setting the Endpoint Disabled Interrupt. The application must set this bit only if Endpoint Enable is already set for this endpoint. This bit is set only by software and cleared only by hardware.

Field	Bits	Type	Description
EPEna	31	rwh	**Endpoint Enable** • When Scatter/Gather DMA mode is enabled, for IN endpoints this bit indicates that the descriptor structure and data buffer with data ready to transmit is setup. • When Scatter/Gather DMA mode is disabled—such as in buffer-pointer based DMA mode—this bit indicates that data is ready to be transmitted on the endpoint. The core clears this bit before setting the following interrupts on this endpoint: • Endpoint Disabled • Transfer Completed This bit is set only by software and cleared only by hardware.
0	[29:28], 20, 16, [14:2]	r	**Reserved** Read as 0; should be written with 0.

Device Control OUT Endpoint 0 Control Register (DOEPCTL0)

This section describes the Control OUT Endpoint 0 Control register. Non-zero control endpoints use registers for endpoints 1-6.

DOEPCTL0
Device Control OUT Endpoint 0 Control Register($B00_H$) Reset Value: $0000\ 8000_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EPEna	**EPDis**	**0**		**SNAK**	**CNAK**	**0**				**Stall**	**Snp**	**EPType**		**NAKSts**	**0**
rwh	r	r		w	w	r				rwh	rw	r		rh	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
USBActEP	**0**													**MPS**	
r	r													r	

Field	Bits	Type	Description
MPS	[1:0]	r	**Maximum Packet Size** The maximum packet size for control OUT endpoint 0 is the same as what is programmed in control IN Endpoint 0. 00_B 64 bytes 01_B 32 bytes 10_B 16 bytes 11_B 8 bytes
USBActEP	15	r	**USB Active Endpoint** This bit is always set to 1, indicating that a control endpoint 0 is always active in all configurations and interfaces.
NAKSts	17	rh	**NAK Status** Indicates the following: 0_B The core is transmitting non-NAK handshakes based on the FIFO status. 1_B The core is transmitting NAK handshakes on this endpoint. When either the application or the core sets this bit, the core stops receiving data, even if there is space in the RxFIFO to accommodate the incoming packet. Irrespective of this bit's setting, the core always responds to SETUP data packets with an ACK handshake.
EPType	[19:18]	r	**Endpoint Type** Hardcoded to 00 for control.
Snp	20	rw	**Snoop Mode** This bit configures the endpoint to Snoop mode. In Snoop mode, the core does not check the correctness of OUT packets before transferring them to application memory.
Stall	21	rwh	**STALL Handshake** The application can only set this bit, and the core clears it, when a SETUP token is received for this endpoint. If a NAK bit or Global OUT NAK is set along with this bit, the STALL bit takes priority. Irrespective of this bit's setting, the core always responds to SETUP data packets with an ACK handshake. This bit is set only by software and cleared only by hardware.
CNAK	26	w	**Clear NAK** A write to this bit clears the NAK bit for the endpoint.

Field	Bits	Type	Description
SNAK	27	w	**Set NAK** A write to this bit sets the NAK bit for the endpoint. Using this bit, the application can control the transmission of NAK handshakes on an endpoint. The core can also set bit on a Transfer Completed interrupt, or after a SETUP is received on the endpoint.
EPDis	30	r	**Endpoint Disable** The application cannot disable control OUT endpoint 0.
EPEna	31	rwh	**Endpoint Enable** When Scatter/Gather DMA mode is enabled, for OUT endpoints this bit indicates that the descriptor structure and data buffer to receive data is setup. • When Scatter/Gather DMA mode is disabled—(such as for buffer-pointer based DMA mode)—this bit indicates that the application has allocated the memory to start receiving data from the USB. The core clears this bit before setting any of the following interrupts on this endpoint: • SETUP Phase Done • Endpoint Disabled • Transfer Completed *Note: In DMA mode, this bit must be set for the core to transfer SETUP data packets into memory.* This bit is set only by software and cleared only by hardware.
0	[29:28], [25:22], 16, [14:2]	r	**Reserved** Read as 0; should be written with 0.

Device Endpoint-n Control Register (DIEPCTLx/DOEPCTLx)

The application uses this register to control the behavior of each logical endpoint other than endpoint 0.

Note: The fields of the DIEPCTLx/DOEPCTLx register change, depending on interrupt/bulk or isochronous/control endpoint.

DIEPCTLx (x=1-6)
Device Endpoint-x Control Register [INTBULK]
(900_H + x*20_H) **Reset Value: 0000 0000_H**

DOEPCTLx (x=1-6)
Device Endpoint-x Control Register [INTBULK]
($B00_H$ + x*20_H) **Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EPEna	**EPDis**	**SetD1PID**	**SetD0PID**	**SNAK**	**CNAK**	**TxFNum**				**Stall**	**Snp**	**EPType**		**NAKSts**	**DPID**
rwh	rwh	w	w	w	w	rw				rw	rw	rw		rh	rh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
USBActEP	**0**				**MPS**										
rwh	r				rw										

Field	Bits	Type	Description
MPS	[10:0]	rw	**Maximum Packet Size** Applies to IN and OUT endpoints. The application must program this field with the maximum packet size for the current logical endpoint. This value is in bytes.
USBActEP	15	rwh	**USB Active Endpoint** Applies to IN and OUT endpoints. Indicates whether this endpoint is active in the current configuration and interface. The core clears this bit for all endpoints (other than EP 0) after detecting a USB reset. After receiving the SetConfiguration and SetInterface commands, the application must program endpoint registers accordingly and set this bit. This bit is set only by software and can be cleared by hardware or a software write of 0 to the bit.

Field	Bits	Type	Description
DPID	16	rh	**Endpoint Data PID** Applies to interrupt/bulk IN and OUT endpoints only. Contains the PID of the packet to be received or transmitted on this endpoint. The application must program the PID of the first packet to be received or transmitted on this endpoint, after the endpoint is activated. The applications use the SetD1PID and SetD0PID fields of this register to program either DATA0 or DATA1 PID. 0_B DATA0 1_B DATA1 This field is applicable both for Scatter/Gather DMA mode and non-Scatter/Gather DMA mode.
NAKSts	17	rh	**NAK Status** Applies to IN and OUT endpoints. Indicates the following: 0_B The core is transmitting non-NAK handshakes based on the FIFO status. 1_B The core is transmitting NAK handshakes on this endpoint. When either the application or the core sets this bit: The core stops receiving any data on an OUT endpoint, even if there is space in the RxFIFO to accommodate the incoming packet. For non-isochronous IN endpoints: The core stops transmitting any data on an IN endpoint, even if there data is available in the TxFIFO. For isochronous IN endpoints: The core sends out a zero-length data packet, even if there data is available in the TxFIFO. Irrespective of this bit's setting, the core always responds to SETUP data packets with an ACK handshake.

Field	Bits	Type	Description
EPType	[19:18]	rw	**Endpoint Type** Applies to IN and OUT endpoints. This is the transfer type supported by this logical endpoint. 00_B Control 01_B Isochronous 10_B Bulk 11_B Interrupt
Snp	20	rw	**Snoop Mode** Applies to OUT endpoints only. This bit configures the endpoint to Snoop mode. In Snoop mode, the core does not check the correctness of OUT packets before transferring them to application memory.
Stall	21	rw	**STALL Handshake** Applies to non-control, non-isochronous IN and OUT endpoints only. The application sets this bit to stall all tokens from the USB host to this endpoint. If a NAK bit, Global Non-periodic IN NAK, or Global OUT NAK is set along with this bit, the STALL bit takes priority. Only the application can clear this bit, never the core.
TxFNum	[25:22]	rw	**TxFIFO Number** **T**hese bits specify the FIFO number associated with this endpoint. Each active IN endpoint must be programmed to a separate FIFO number. This field is valid only for IN endpoints.
CNAK	26	w	**Clear NAK** Applies to IN and OUT endpoints. A write to this bit clears the NAK bit for the endpoint.
SNAK	27	w	**Set NAK** Applies to IN and OUT endpoints.A write to this bit sets the NAK bit for the endpoint. Using this bit, the application can control the transmission of NAK handshakes on an endpoint. The core can also set this bit for OUT endpoints on a Transfer Completed interrupt, or after a SETUP is received on the endpoint.

Field	Bits	Type	Description
SetD0PID	28	w	**Set DATA0 PID** Applies to interrupt/bulk IN and OUT endpoints only. Writing to this field sets the Endpoint Data PID (DPID) field in this register to DATA0. This field is applicable both for Scatter/Gather DMA mode and non-Scatter/Gather DMA mode.
SetD1PID	29	w	**29 Set DATA1 PID** Applies to interrupt/bulk IN and OUT endpoints only. Writing to this field sets the Endpoint Data PID (DPID) field in this register to DATA1. This field is applicable both for Scatter/Gather DMA mode and non-Scatter/Gather DMA mode.
EPDis	30	rwh	**Endpoint Disable** Applies to IN and OUT endpoints. The application sets this bit to stop transmitting/receiving data on an endpoint, even before the transfer for that endpoint is complete. The application must wait for the Endpoint Disabled interrupt before treating the endpoint as disabled. The core clears this bit before setting the Endpoint Disabled interrupt. The application must set this bit only if Endpoint Enable is already set for this endpoint. This bit is set only by software and cleared only by hardware.

Field	Bits	Type	Description
EPEna	31	rwh	**Endpoint Enable** Applies to IN and OUT endpoints. • When Scatter/Gather DMA mode is enabled, • For IN endpoints this bit indicates that the descriptor structure and data buffer with data ready to transmit is setup. • For OUT endpoint it indicates that the descriptor structure and data buffer to receive data is setup. • When Scatter/Gather DMA mode is enabled—such as for buffer-pointer based DMA mode: – For IN endpoints, this bit indicates that data is ready to be transmitted on the endpoint. – For OUT endpoints, this bit indicates that the application has allocated the memory to start receiving data from the USB. – The core clears this bit before setting any of the following interrupts on this endpoint: • SETUP Phase Done • Endpoint Disabled • Transfer Completed *Note: For control endpoints in DMA mode, this bit must be set to be able to transfer SETUP data packets in memory.* This bit is set only by software and cleared only by hardware.
0	[14:11]	r	**Reserved** Read as 0; should be written with 0.

DIEPCTLx (x=1-6)
Device Endpoint-x Control Register [ISOCONT]
(900_H + x*20_H) **Reset Value: 0000 0000_H**

DOEPCTLx (x=1-6)
Device Endpoint-x Control Register [ISOCONT]
($B00_H$ + x*20_H) **Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EPEna	EPDis	SetOddFr	SetEvenFr	SNAK	CNAK	TxFNum				Stall	Snp	EPType		NAKSts	EO_FrNum
rwh	rwh	w	w	w	w	rw				rwh	rw	rw		rh	rh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
USBActEP	0				MPS										
rwh	r				rw										

Field	Bits	Type	Description
MPS	[10:0]	rw	**Maximum Packet Size** Applies to IN and OUT endpoints. The application must program this field with the maximum packet size for the current logical endpoint. This value is in bytes.
USBActEP	15	rwh	**USB Active Endpoint** Applies to IN and OUT endpoints. Indicates whether this endpoint is active in the current configuration and interface. The core clears this bit for all endpoints (other than EP 0) after detecting a USB reset. After receiving the SetConfiguration and SetInterface commands, the application must program endpoint registers accordingly and set this bit. This bit is set only by software and can be cleared by hardware or a software write of 0 to the bit.

Field	Bits	Type	Description
EO_FrNum	16	rh	**Even/Odd Frame** Applies to isochronous IN and OUT endpoints only. In non-Scatter/Gather DMA mode, the bit Indicates the frame number in which the core transmits/receives isochronous data for this endpoint. The application must program the even/odd frame number in which it intends to transmit/receive isochronous data for this endpoint using the SetEvnFr and SetOddFr fields in this register. 0_B Even frame 1_B Odd rame When Scatter/Gather DMA mode is enabled, this field is reserved. The frame number in which to send data is provided in the transmit descriptor structure. The frame in which data is received is updated in receive descriptor structure.
NAKSts	17	rh	**NAK Status** Applies to IN and OUT endpoints. Indicates the following: 0_B The core is transmitting non-NAK handshakes based on the FIFO status. 1_B The core is transmitting NAK handshakes on this endpoint. When either the application or the core sets this bit: The core stops receiving any data on an OUT endpoint, even if there is space in the RxFIFO to accommodate the incoming packet. For non-isochronous IN endpoints: The core stops transmitting any data on an IN endpoint, even if there data is available in the TxFIFO. For isochronous IN endpoints: The core sends out a zero-length data packet, even if there data is available in the TxFIFO. Irrespective of this bit's setting, the core always responds to SETUP data packets with an ACK handshake.

Field	Bits	Type	Description
EPType	[19:18]	rw	**Endpoint Type** Applies to IN and OUT endpoints. This is the transfer type supported by this logical endpoint. 00_B Control 01_B Isochronous 10_B Bulk 11_B Interrupt
Snp	20	rw	**Snoop Mode** Applies to OUT endpoints only. This bit configures the endpoint to Snoop mode. In Snoop mode, the core does not check the correctness of OUT packets before transferring them to application memory.
Stall	21	rwh	**STALL Handshake** Applies to control endpoints only. The application can only set this bit, and the core clears it, when a SETUP token is received for this endpoint. If a NAK bit, Global Non-periodic IN NAK, or Global OUT NAK is set along with this bit, the STALL bit takes priority. Irrespective of this bit's setting, the core always responds to SETUP data packets with an ACK handshake. This bit is set only by software and cleared only by hardware.
TxFNum	[25:22]	rw	**TxFIFO Number** **T**hese bits specify the FIFO number associated with this endpoint. Each active IN endpoint must be programmed to a separate FIFO number. This field is valid only for IN endpoints.
CNAK	26	w	**Clear NAK** Applies to IN and OUT endpoints. A write to this bit clears the NAK bit for the endpoint.

Field	Bits	Type	Description
SNAK	27	w	**Set NAK** Applies to IN and OUT endpoints.A write to this bit sets the NAK bit for the endpoint. Using this bit, the application can control the transmission of NAK handshakes on an endpoint. The core can also set this bit for OUT endpoints on a Transfer Completed interrupt, or after a SETUP is received on the endpoint.
SetEvenFr	28	w	**In non-Scatter/Gather DMA mode: Set Even frame** Applies to isochronous IN and OUT endpoints only. Writing to this field sets the Even/Odd frame (EO_FrNum) field to even frame. When Scatter/Gather DMA mode is enabled, this field is reserved. The frame number in which to send data is in the transmit descriptor structure. The frame in which to receive data is updated in receive descriptor structure.
SetOddFr	29	w	**Set Odd frame** Applies to isochronous IN and OUT endpoints only. Writing to this field sets the Even/Odd frame (EO_FrNum) field to odd frame. This field is not applicable for Scatter/Gather DMA mode.
EPDis	30	rwh	**Endpoint Disable** Applies to IN and OUT endpoints. The application sets this bit to stop transmitting/receiving data on an endpoint, even before the transfer for that endpoint is complete. The application must wait for the Endpoint Disabled interrupt before treating the endpoint as disabled. The core clears this bit before setting the Endpoint Disabled interrupt. The application must set this bit only if Endpoint Enable is already set for this endpoint. This bit is set only by software and cleared only by hardware.

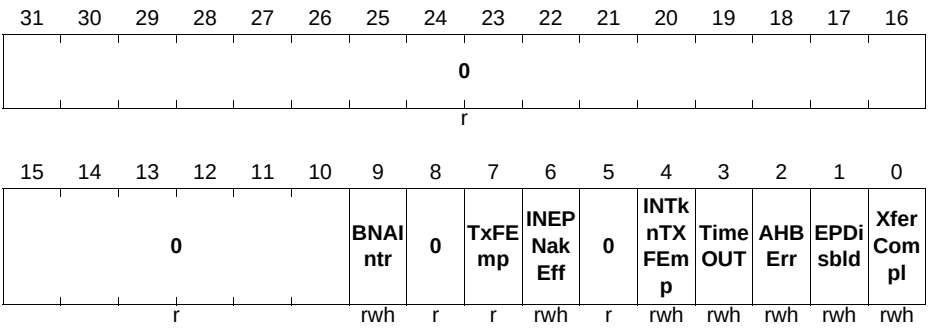
Field	Bits	Type	Description
EPEna	31	rwh	**Endpoint Enable** Applies to IN and OUT endpoints. • When Scatter/Gather DMA mode is enabled, • For IN endpoints this bit indicates that the descriptor structure and data buffer with data ready to transmit is setup. • For OUT endpoint it indicates that the descriptor structure and data buffer to receive data is setup. • When Scatter/Gather DMA mode is enabled—such as for buffer-pointer based DMA mode: – For IN endpoints, this bit indicates that data is ready to be transmitted on the endpoint. – For OUT endpoints, this bit indicates that the application has allocated the memory to start receiving data from the USB. – The core clears this bit before setting any of the following interrupts on this endpoint: • SETUP Phase Done • Endpoint Disabled • Transfer Completed *Note: For control endpoints in DMA mode, this bit must be set to be able to transfer SETUP data packets in memory.* This bit is set only by software and cleared only by hardware.
0	[14:11]	r	**Reserved** Read as 0; should be written with 0.

Device Endpoint-n Interrupt Register (DIEPINTx/DOEPINTx)

This register indicates the status of an endpoint with respect to USB- and AHB-related events. It is shown in **Figure 13-47 “Interrupt Hierarchy” on Page 13-159**. The application must read this register when the OUT Endpoints Interrupt bit or IN Endpoints Interrupt bit of the Core Interrupt register (GINTSTS.OEPInt or GINTSTS.IEPInt, respectively) is set. Before the application can read this register, it must first read the Device All Endpoints Interrupt (DAINT) register to get the exact endpoint number for the Device Endpoint-n Interrupt register. The application must clear the appropriate bit in this register to clear the corresponding bits in the DAINT and GINTSTS registers.

Note: In the DIEPINTx/DOEPINTx registers, status bits with access type ‘rwh’ are set by hardware. To clear these bits, the application must write 1 into these bits.

DIEPINTx (x=0-6)
Device Endpoint-x Interrupt Register (908_H + x*20_H) **Reset Value: 0000 0080_H**

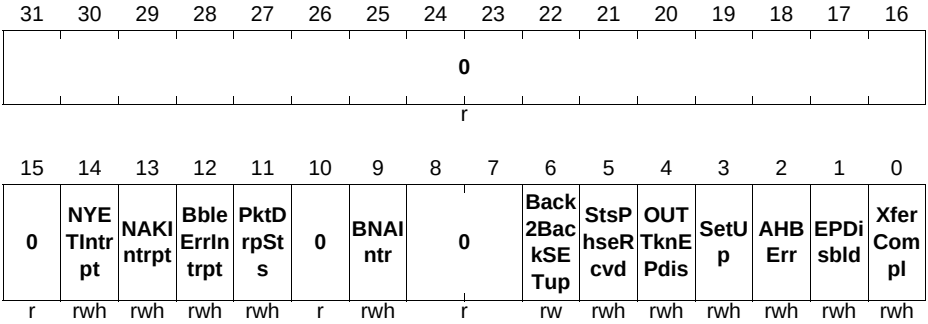


Field	Bits	Type	Description
XferCompl	0	rwh	**Transfer Completed Interrupt** Applies to IN and OUT endpoints. • When Scatter/Gather DMA mode is enabled • For IN endpoint this field indicates that the requested data from the descriptor is moved from external system memory to internal FIFO. • For OUT endpoint this field indicates that the requested data from the internal FIFO is moved to external system memory. This interrupt is generated only when the corresponding endpoint descriptor is closed, and the IOC bit for the corresponding descriptor is set. • When Scatter/Gather DMA mode is disabled, this field indicates that the programmed transfer is complete on the AHB as well as on the USB, for this endpoint.
EPDisbld	1	rwh	**Endpoint Disabled Interrupt** Applies to IN and OUT endpoints. This bit indicates that the endpoint is disabled per the application's request.
AHBErr	2	rwh	**AHB Error** Applies to IN and OUT endpoints. This is generated only in DMA mode when there is an AHB error during an AHB read/write. The application can read the corresponding endpoint DMA address register to get the error address.

Field	Bits	Type	Description
TimeOUT	3	rwh	**Timeout Condition** • Applies only to Control IN endpoints. • In Scatter/Gather DMA mode, the TimeOUT interrupt is not asserted. Indicates that the core has detected a timeout condition on the USB for the last IN token on this endpoint.
INTknTXFEmp	4	rwh	**IN Token Received When TxFIFO is Empty** Indicates that an IN token was received when the associated TxFIFO (periodic/non-periodic) was empty. This interrupt is asserted on the endpoint for which the IN token was received.
INEPNakEff	6	rwh	**IN Endpoint NAK Effective** Applies to periodic IN endpoints only. This bit can be cleared when the application clears the IN endpoint NAK by writing to DIEPCTLx.CNAK. This interrupt indicates that the core has sampled the NAK bit set (either by the application or by the core). The interrupt indicates that the IN endpoint NAK bit set by the application has taken effect in the core. This interrupt does not guarantee that a NAK handshake is sent on the USB. A STALL bit takes priority over a NAK bit. This bit is applicable only when the endpoint is enabled.
TxFEmp	7	r	**Transmit FIFO Empty** This bit is valid only for IN Endpoints This interrupt is asserted when the TxFIFO for this endpoint is either half or completely empty. The half or completely empty status is determined by the TxFIFO Empty Level bit in the Core AHB Configuration register (GAHBCFG.NPTxFEmpLvl)).
BNAIntr	9	rwh	**BNA (Buffer Not Available) Interrupt** The core generates this interrupt when the descriptor accessed is not ready for the Core to process, such as Host busy or DMA done This bit is valid only when Scatter/Gather DMA mode is enabled.
0	[31:10], 8, 5	r	**Reserved** Read as 0; should be written with 0.

DOEPINTx (x=0-6)

Device Endpoint-x Interrupt Register ($B08_H$ + x*20_H) **Reset Value: 0000 0080_H**



Field	Bits	Type	Description
XferCompl	0	rwh	**Transfer Completed Interrupt** Applies to IN and OUT endpoints. • When Scatter/Gather DMA mode is enabled • For IN endpoint this field indicates that the requested data from the descriptor is moved from external system memory to internal FIFO. • For OUT endpoint this field indicates that the requested data from the internal FIFO is moved to external system memory. This interrupt is generated only when the corresponding endpoint descriptor is closed, and the IOC bit for the corresponding descriptor is set. • When Scatter/Gather DMA mode is disabled, this field indicates that the programmed transfer is complete on the AHB as well as on the USB, for this endpoint.
EPDisbld	1	rwh	**Endpoint Disabled Interrupt** Applies to IN and OUT endpoints. This bit indicates that the endpoint is disabled per the application's request.
AHBErr	2	rwh	**AHB Error** Applies to IN and OUT endpoints. This is generated only in DMA mode when there is an AHB error during an AHB read/write. The application can read the corresponding endpoint DMA address register to get the error address.

Field	Bits	Type	Description
SetUp	3	rwh	**SETUP Phase Done** Applies to control OUT endpoints only. Indicates that the SETUP phase for the control endpoint is complete and no more back-to-back SETUP packets were received for the current control transfer. On this interrupt, the application can decode the received SETUP data packet.
OUTTknEPdis	4	rwh	**OUT Token Received When Endpoint Disabled** Indicates that an OUT token was received when the endpoint was not yet enabled. This interrupt is asserted on the endpoint for which the OUT token was received.
StsPhseRcvd	5	rwh	**Status Phase Received For Control Write** This interrupt is valid only for Control OUT endpoints and only in Scatter Gather DMA mode. This interrupt is generated only after the core has transferred all the data that the host has sent during the data phase of a control write transfer, to the system memory buffer. The interrupt indicates to the application that the host has switched from data phase to the status phase of a Control Write transfer. The application can use this interrupt to ACK or STALL the Status phase, after it has decoded the data phase. This is applicable only in case of Scatter Gather DMA mode.
Back2BackSETup	6	rw	**Back-to-Back SETUP Packets Received** Applies to Control OUT endpoints only. This bit indicates that the core has received more than three back-to-back SETUP packets for this particular endpoint. For information about handling this interrupt, see **“Handling More Than Three Back-to-Back SETUP Packets” on Page 13-35**.
BNAIntr	9	rwh	**BNA (Buffer Not Available) Interrupt** The core generates this interrupt when the descriptor accessed is not ready for the Core to process, such as Host busy or DMA done This bit is valid only when Scatter/Gather DMA mode is enabled.

Field	Bits	Type	Description
PktDrpSts	11	rwh	**Packet Dropped Status** This bit indicates to the application that an ISOC OUT packet has been dropped. This bit does not have an associated mask bit and does not generate an interrupt. This bit is valid in non Scatter/Gather DMA mode when periodic transfer interrupt feature is selected.
BbleErrIntrpt	12	rwh	**BbleErr (Babble Error) interrupt** The core generates this interrupt when babble is received for the endpoint.
NAKIntrpt	13	rwh	**NAK interrupt** The core generates this interrupt when a NAK is transmitted or received by the device.In case of isochronous IN endpoints the interrupt gets generated when a zero length packet is transmitted due to un-availability of data in the TXFIFO.
NYETIntrpt	14	rwh	**NYET interrupt** The core generates this interrupt when a NYET response is transmitted for a non isochronous OUT endpoint.
0	[31:15], 10, [8:7]	r	**Reserved** Read as 0; should be written with 0.

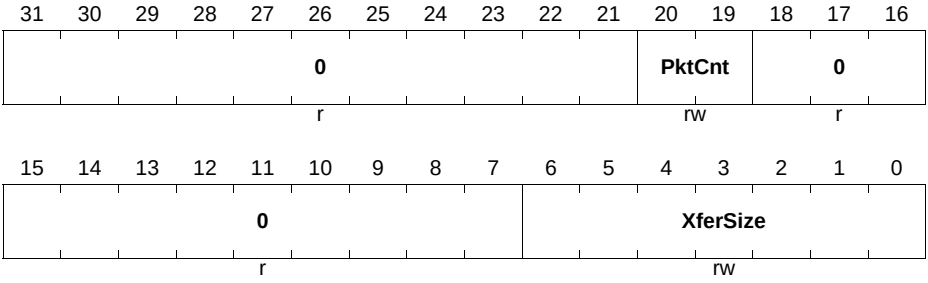
Device Endpoint 0 Transfer Size Register (DIEPTSIZ0/DOEPTSIZ0)

The application must modify this register before enabling endpoint 0. Once endpoint 0 is enabled using Endpoint Enable bit of the Device Control Endpoint 0 Control registers (DIEPCTL0.EPEna/DOEPCTL0.EPEna), the core modifies this register. The application can only read this register once the core has cleared the Endpoint Enable bit.

Non-zero endpoints use the registers for endpoints 1-6.

When Scatter/Gather DMA mode is enabled, this register must not be programmed by the application. If the application reads this register when Scatter/Gather DMA mode is enabled, the core returns all zeros.

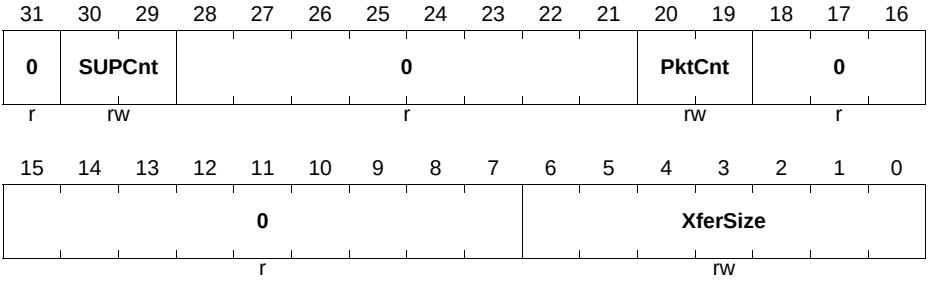
DIEPTSIZ0
Device IN Endpoint 0 Transfer Size Register(910_H) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
XferSize	[6:0]	rw	**Transfer Size** Indicates the transfer size in bytes for endpoint 0. The core interrupts the application only after it has exhausted the transfer size amount of data. The transfer size can be set to the maximum packet size of the endpoint, to be interrupted at the end of each packet. The core decrements this field every time a packet from the external memory is written to the TxFIFO.
PktCnt	[20:19]	rw	**Packet Count** Indicates the total number of USB packets that constitute the Transfer Size amount of data for endpoint 0. This field is decremented every time a packet (maximum size or short packet) is read from the TxFIFO.
0	[31:21], [18:7]	r	**Reserved** Read as 0; should be written with 0.

DOEPTSIZ0

Device OUT Endpoint 0 Transfer Size Register($B10_H$) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
XferSize	[6:0]	rw	**Transfer Size** Indicates the transfer size in bytes for endpoint 0. The core interrupts the application only after it has exhausted the transfer size amount of data. The transfer size can be set to the maximum packet size of the endpoint, to be interrupted at the end of each packet. The core decrements this field every time a packet is read from the RxFIFO and written to the external memory.
PktCnt	[20:19]	rw	**Packet Count** This field is decremented to zero after a packet is written into the RxFIFO.
SUPCnt	[30:29]	rw	**SETUP Packet Count** This field specifies the number of back-to-back SETUP data packets the endpoint can receive. 01_B 1 packet 10_B 2 packets 11_B 3 packets
0	31, [28:21], [18:7]	r	**Reserved** Read as 0; should be written with 0.

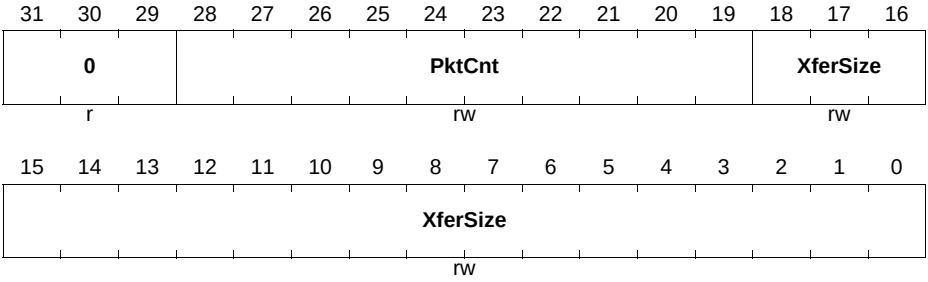
Device Endpoint-n Transfer Size Register (DIEPTSIZx/DOEPTSIZx)

The application must modify this register before enabling the endpoint. Once the endpoint is enabled using Endpoint Enable bit of the Device Endpoint-n Control registers (DIEPCTLx.EPEna/DOEPCTLx.EPEna), the core modifies this register. The application can only read this register once the core has cleared the Endpoint Enable bit.

This register is used only for endpoints other than Endpoint 0.

Note: When Scatter/Gather DMA mode is enabled, this register must not be programmed by the application. If the application reads this register when Scatter/Gather DMA mode is enabled, the core returns all zeros

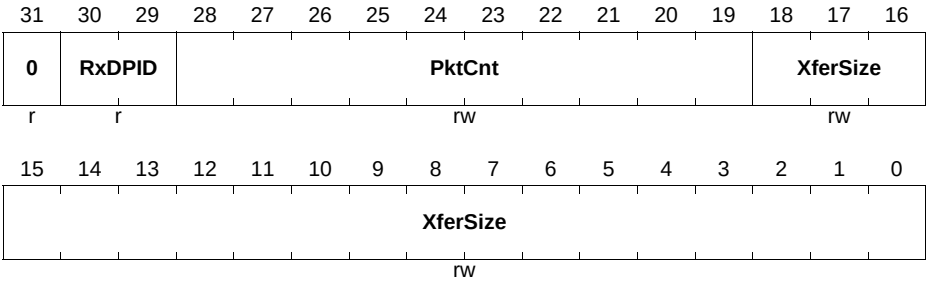
DIEPTSIZx (x=1-6)
Device Endpoint-x Transfer Size Register(910_H + x*20_H) Reset Value: 0000 0000_H



Field	Bits	Type	Description
XferSize	[18:0]	rw	**Transfer Size** This field contains the transfer size in bytes for the current endpoint. The core only interrupts the application after it has exhausted the transfer size amount of data. The transfer size can be set to the maximum packet size of the endpoint, to be interrupted at the end of each packet. • IN Endpoints: The core decrements this field every time a packet from the external memory is written to the TxFIFO.
PktCnt	[28:19]	rw	**Packet Count** Indicates the total number of USB packets that constitute the Transfer Size amount of data for this endpoint. • IN Endpoints: This field is decremented every time a packet (maximum size or short packet) is read from the TxFIFO..
0	[31:29]	r	**Reserved** Read as 0; should be written with 0.

Note: The fields of the DOEPTSIZx register change, depending on isochronous or control OUT endpoint.

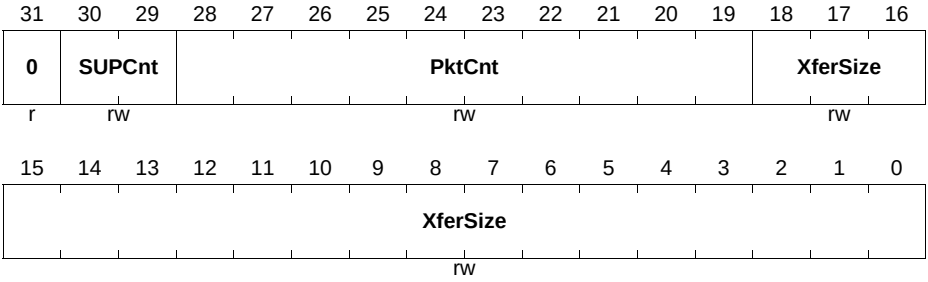
DOEPTSIZx (x=1-6)
Device Endpoint-x Transfer Size Register [ISO]
($B10_H$ + x*20_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
XferSize	[18:0]	rw	**Transfer Size** This field contains the transfer size in bytes for the current endpoint. The core only interrupts the application after it has exhausted the transfer size amount of data. The transfer size can be set to the maximum packet size of the endpoint, to be interrupted at the end of each packet. • OUT Endpoints: The core decrements this field every time a packet is read from the RxFIFO and written to the external memory.
PktCnt	[28:19]	rw	**Packet Count** Indicates the total number of USB packets that constitute the Transfer Size amount of data for this endpoint. • OUT Endpoints: This field is decremented every time a packet (maximum size or short packet) is written to the RxFIFO.

Field	Bits	Type	Description
RxDPID	[30:29]	r	**Received Data PID** Applies to isochronous OUT endpoints only. This is the data PID received in the last packet for this endpoint. 00_B DATA0 01_B DATA2 10_B DATA1 11_B MDATA
0	31	r	**Reserved** Read as 0; should be written with 0.

DOEPTSIZx (x=1-6)
Device Endpoint-x Transfer Size Register [CONT]
($B10_H$ + x*20_H) **Reset Value: 0000 0000_H**

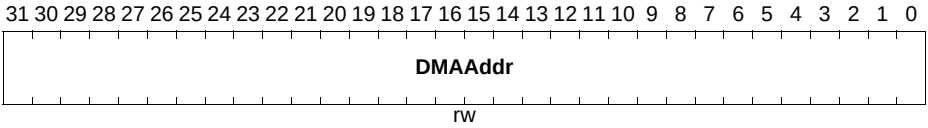


Field	Bits	Type	Description
XferSize	[18:0]	rw	**Transfer Size** This field contains the transfer size in bytes for the current endpoint. The core only interrupts the application after it has exhausted the transfer size amount of data. The transfer size can be set to the maximum packet size of the endpoint, to be interrupted at the end of each packet. • OUT Endpoints: The core decrements this field every time a packet is read from the RxFIFO and written to the external memory.
PktCnt	[28:19]	rw	**Packet Count** Indicates the total number of USB packets that constitute the Transfer Size amount of data for this endpoint. • OUT Endpoints: This field is decremented every time a packet (maximum size or short packet) is written to the RxFIFO.
SUPCnt	[30:29]	rw	**SETUP Packet Count** Applies to control OUT Endpoints only. This field specifies the number of back-to-back SETUP data packets the endpoint can receive. 01_B 1 packet 10_B 2 packets 11_B 3 packets
0	31	r	**Reserved** Read as 0; should be written with 0.

Device Endpoint-n DMA Address Register (DIEPDMAx/DOEPDMAx)

These registers are implemented in RAM.

DIEPDMAx (x=0-6)
Device Endpoint-x DMA Address Register(914_H + x*20_H) **Reset Value: XXXX XXXX$_H$**
DOEPDMAx (x=0-6)
Device Endpoint-x DMA Address Register($B14_H$ + x*20_H) **Reset Value: XXXX XXXX$_H$**

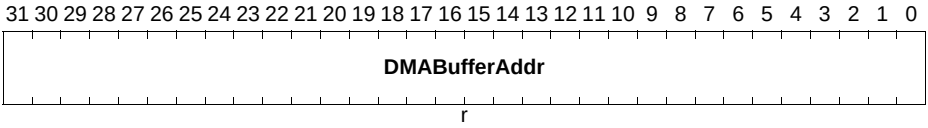


Field	Bits	Type	Description
DMAAddr	[31:0]	rw	**DMA Address** Holds the start address of the external memory for storing or fetching endpoint data. *Note: For control endpoints, this field stores control OUT data packets as well as SETUP transaction data packets. When more than three SETUP packets are received back-to-back, the SETUP data packet in the memory is overwritten.* This register is incremented on every AHB transaction. The application can give only a DWORD-aligned address. • When Scatter/Gather DMA mode is not enabled, the application programs the start address value in this field. • When Scatter/Gather DMA mode is enabled, this field indicates the base pointer for the descriptor list.

Device Endpoint-n DMA Buffer Address Register (DIEPDMABx/DOEPDMABx)

These fields are present only in case of Scatter/Gather DMA. These registers are implemented in RAM.

DIEPDMABx (x=0-6)
Device Endpoint-x DMA Buffer Address Register($91C_H$ + x*20_H) Reset Value: XXXX $XXXX_H$
DOEPDMABx (x=0-6)
Device Endpoint-x DMA Buffer Address Register($B1C_H$ + x*20_H) Reset Value: XXXX $XXXX_H$

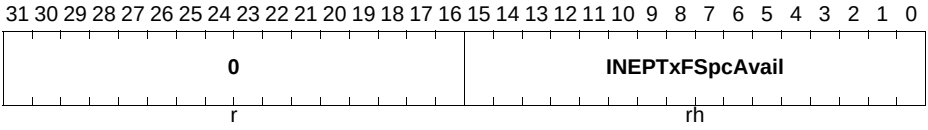


Field	Bits	Type	Description
DMABufferAddr	[31:0]	r	**DMA Buffer Address** Holds the current buffer address.This register is updated as and when the data transfer for the corresponding end point is in progress. This register is present only in Scatter/Gather DMA mode. Otherwise this field is reserved.

Device IN Endpoint Transmit FIFO Status Register (DTXFSTSx)

This read-only register contains the free space information for the Device IN endpoint TxFIFO.

DTXFSTSx (x=0-6)
Device IN Endpoint Transmit FIFO Status Register(918_H + x*20_H) Reset Value: 0000 0000_H



Field	Bits	Type	Description
INEPTxFSpcAvail	[15:0]	rh	**IN Endpoint TxFIFO Space Avail** Indicates the amount of free space available in the Endpoint TxFIFO. Values are in terms of 32-bit words. 0_H Endpoint TxFIFO is full 1_H 1 word available 2_H 2 words available Others: Up to n words can be selected (0 < n < 256); selections greater than n are reserved
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

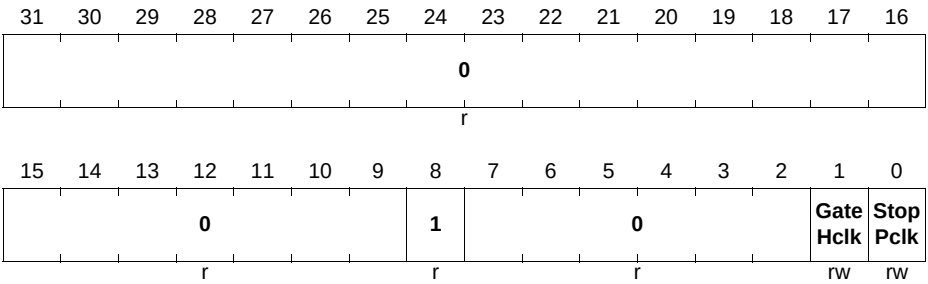
Power and Clock Gating Registers

There is a single register for power and clock gating.

Power and Clock Gating Control Register (PCGCCTL)

The application can use this register to control the core's clock gating features.

PCGCCTL
Power and Clock Gating Control Register($E00_H$) **Reset Value: 0000 0100$_H$**



Field	Bits	Type	Description
StopPclk	0	rw	**Stop Pclk** The application sets this bit to stop the PHY clock (phy_clk) when the USB is suspended, the session is not valid, or the device is disconnected. The application clears this bit when the USB is resumed or a new session starts.
GateHclk	1	rw	**Gate Hclk** The application sets this bit to gate hclk to modules other than the AHB Slave and Master and wakeup logic when the USB is suspended or the session is not valid. The application clears this bit when the USB is resumed or a new session starts.
1	8	r	**Reserved** Read as 1; should be written with 1.
0	[31:9], [7:2]	r	**Reserved** Read as 0; should be written with 0.

13.15 Interconnects

The interconnects section describes the connectivity of the module.

VBUSDETECT signal can be selected from one of the input pin functions VBUSDETECT[C:A] or controlled by software, through the SCU register USBHOSTDET.

Table 13-16 Pin Connections

Input/Output	I/O	Connected To	Description
USB0.D+	I/O	USB_DP	Data + signal
USB0.D-	I/O	USB_DM	Data - signal
USB0.VBUSDETECTA	I	P0.0	VBUSDETECT signal
USB0.VBUSDETECTB	I	P1.7	VBUSDETECT signal
USB0.VBUSDETECTC	I	HIB_IO_0	VBUSDETECT signal

14 Universal Serial Interface Channel (USIC)

The **U**niversal **S**erial **I**nterface **C**hannel module (USIC) is a flexible interface module covering several serial communication protocols. A USIC module contains two independent communication channels named USICx_CH0 and USICx_CH1, with x being the number of the USIC module (e.g. channel 0 of USIC module 0 is referenced as USIC0_CH0). The user can program during run-time which protocol will be handled by each communication channel and which pins are used.

References

The following documents are referenced for further information

[11] IIC Bus Specification (Philips Semiconductors v2.1)

[12] IIS Bus Specification (Philips Semiconductors June 5 1996 revision)

Table 14-1 Abbreviations

CTQ	Time Quanta Counter
DSU	Data Shift Unit
f_{PERIPH}	USIC module clock frequency
f_{PIN}	Input frequency to baud rate generator
MCLK	Master Clock
PPP	Protocol Pre-Processor
RSR	Receive Shift Register
SCLK	Shift Clock
TSR	Transmit Shift Register

14.1 Overview

This section gives an overview about the feature set of the USIC.

14.1.1 Features

Each USIC channel can be individually configured to match the application needs, e.g. the protocol can be selected or changed during run time without the need for a reset. The following protocols are supported:

- **UART** (ASC, asynchronous serial channel)
  - Module capability: receiver/transmitter with max. baud rate f_{PERIPH} / 4
  - Wide baud rate range down to single-digit baud rates
  - Number of data bits per data frame: 1 to 63
  - MSB or LSB first

- **LIN** Support by hardware (Local Interconnect Network)
  - Data transfers based on ASC protocol
  - Baud rate detection possible by built-in capture event of baud rate generator
  - Checksum generation under software control for higher flexibility
- **SSC/SPI** (synchronous serial channel with or without slave select lines)
  - Standard, Dual and Quad SPI format supported
  - Module capability: maximum baud rate f_{PERIPH} / 2, limited by loop delay
  - Number of data bits per data frame 1 to 63, more with explicit stop condition
  - Parity bit generation supported
  - MSB or LSB first
- **IIC** (Inter-IC Bus)
  - Application baud rate 100 kbit/s to 400 kbit/s
  - 7-bit and 10-bit addressing supported
  - Full master and slave device capability
- **IIS** (infotainment audio bus)
  - Module capability: maximum baud rate f_{PERIPH} / 2

Note: The real baud rates that can be achieved in a real application depend on the operating frequency of the device, timing parameters as described in the Data Sheet, signal delays on the PCB and timings of the peer device.

In addition to the flexible choice of the communication protocol, the USIC structure has been designed to reduce the system load (CPU load) allowing efficient data handling. The following aspects have been considered:

- **Data buffer capability**
  The standard buffer capability includes a double word buffer for receive data and a single word buffer for transmit data. This allows longer CPU reaction times (e.g. interrupt latency).
- **Additional FIFO buffer capability**
  In addition to the standard buffer capability, the received data and the data to be transmitted can be buffered in a FIFO buffer structure. The size of the receive and the transmit FIFO buffer can be programmed independently. Depending on the application needs, a total buffer capability of 64 data words can be assigned to the receive and transmit FIFO buffers of a USIC module (the two channels of the USIC module share the 64 data word buffer).
  In addition to the FIFO buffer, a bypass mechanism allows the introduction of high-priority data without flushing the FIFO buffer.
- **Transmit control information**
  For each data word to be transmitted, a 5-bit transmit control information has been added to automatically control some transmission parameters, such as word length, frame length, or the slave select control for the SPI protocol. The transmit control information is generated automatically by analyzing the address where the user software has written the data word to be transmitted (32 input locations = 2^5 = 5 bit transmit control information).

This feature allows individual handling of each data word, e.g. the transmit control information associated to the data words stored in a transmit FIFO can automatically modify the slave select outputs to select different communication targets (slave devices) without CPU load. Alternatively, it can be used to control the frame length.

- **Flexible frame length control**
  The number of bits to be transferred within a data frame is independent of the data word length and can be handled in two different ways. The first option allows automatic generation of frames up to 63 bits with a known length. The second option supports longer frames (even unlimited length) or frames with a dynamically controlled length.
- **Interrupt capability**
  The events of each USIC channel can be individually routed to one of 6 service request outputs SR[5:0] available for each USIC module, depending on the application needs. Furthermore, specific start and end of frame indications are supported in addition to protocol-specific events.
- **Flexible interface routing**
  Each USIC channel offers the choice between several possible input and output pins connections for the communications signals. This allows a flexible assignment of USIC signals to pins that can be changed without resetting the device.
- **Input conditioning**
  Each input signal is handled by a programmable input conditioning stage with programmable filtering and synchronization capability.
- **Baud rate generation**
  Each USIC channel contains its own baud rate generator. The baud rate generation can be based either on the internal module clock or on an external frequency input. This structure allows data transfers with a frequency that can not be generated internally, e.g. to synchronize several communication partners.
- **Transfer trigger capability**
  In master mode, data transfers can be triggered by events generated outside the USIC module, e.g. by an input pin or a timer unit (transmit data validation). This feature allows time base related data transmission.
- **Debugger support**
  The USIC offers specific addresses to read out received data without interaction with the FIFO buffer mechanism. This feature allows debugger accesses without the risk of a corrupted receive data sequence.

To reach a desired baud rate, two criteria have to be respected, the module capability and the application environment. The module capability is defined with respect to the module's input clock frequency, being the base for the module operation. Although the module's capability being much higher (depending on the module clock and the number of module clock cycles needed to represent a data bit), the reachable baud rate is generally limited by the application environment. In most cases, the application

environment limits the maximum reachable baud rate due to driver delays, signal propagation times, or due to EMI reasons.

Note: Depending on the selected additional functions (such as digital filters, input synchronization stages, sample point adjustment, data structure, etc.), the maximum reachable baud rate can be limited. Please also take care about additional delays, such as (internal or external) propagation delays and driver delays (e.g. for collision detection in ASC mode, for IIC, etc.).

A block diagram of the USIC module/channel structure is shown in **Figure 14-1**.

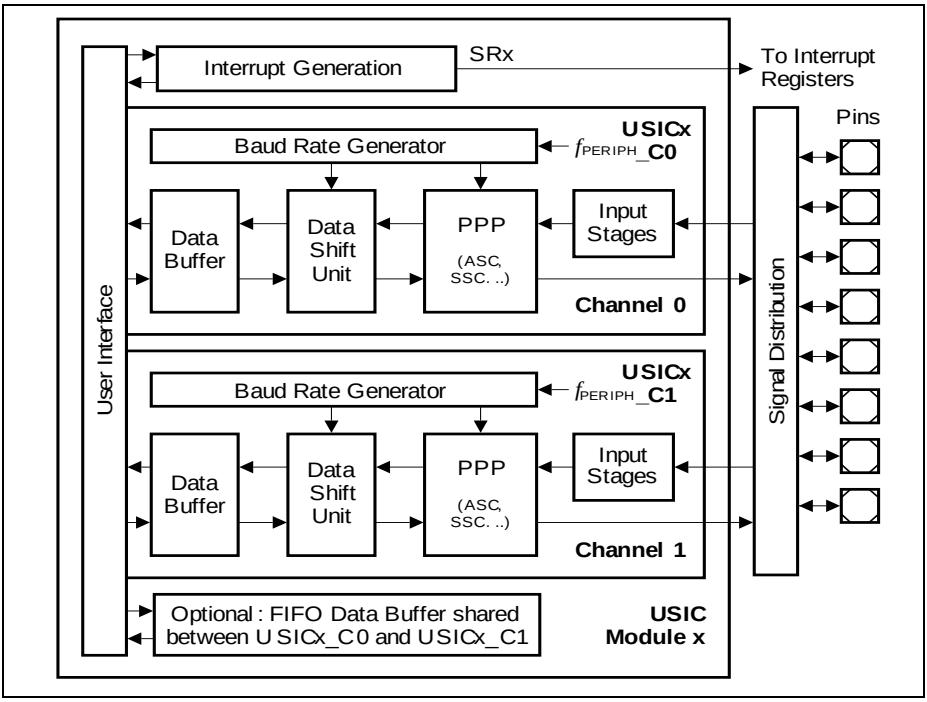


Figure 14-1 USIC Module/Channel Structure

14.2 Operating the USIC

This section describes how to operate the USIC communication channel.

14.2.1 USIC Structure Overview

This section introduces the USIC structure.

14.2.1.1 Channel Structure

The USIC module contains two independent communication channels, with a structure as shown in **Figure 14-1**.

The data shift unit and the data buffering of each channel support full-duplex data transfers. The protocol-specific actions are handled by the protocol pre-processors (PPP). In order to simplify data handling, an additional FIFO data buffer is optionally available for each USIC module to store transmit and receive data for each channel.

Due to the independent channel control and baud rate generation, the communication protocol, baud rate and the data format can be independently programmed for each communication channel.

14.2.1.2 Input Stages

For each protocol, the number of input signals used depends on the selected protocol. Each input signal is handled by an input stage (called DXn, where n=0-5) for signal conditioning, such as input selection, polarity control, or a digital input filter. They can be classified according to their meaning for the protocols, see **Table 14-2**.

The inputs marked as “optional” are not needed for the standard function of a protocol and may be used for enhancements. The descriptions of protocol-specific items are given in the related protocol chapters. For the external frequency input, please refer to the baud rate generator section, and for the transmit data validation, to the data handling section.

Table 14-2 Input Signals for Different Protocols

Selected Protocol	Shift Data Input(s) (handled by DX0, DX3, DX4 and DX5)[1)]	Shift Clock Input (handled by DX1)	Shift Control Input (handled by DX2)
ASC, LIN	RXD	optional: external frequency input or TXD collision detection	optional: transmit data validation
Standard SSC, SPI (Master)	DIN0 (MRST, MISO)	optional: external frequency input or delay compensation	optional: transmit data validation or delay compensation
Standard SSC, SPI (Slave)	DIN0 (MTSR, MOSI)	SCLKIN	SELIN
Dual-SSC, SPI (Master)	DIN[1:0] (MRST[1:0], MISO[1:0])	optional: external frequency input or delay compensation	optional: transmit data validation or delay compensation
Dual-SSC, SPI (Slave)	DIN[1:0] (MTSR[1:0], MOSI[1:0])	SCLKIN	SELIN
Quad-SSC, SPI (Master)	DIN[3:0] (MRST[3:0], MISO[3:0])	optional: external frequency input or delay compensation	optional: transmit data validation or delay compensation
Quad-SSC, SPI (Slave)	DIN[3:0] (MTSR[3:0], MOSI[3:0])	SCLKIN	SELIN
IIC	SDA	SCL	optional: transmit data validation
IIS (Master)	DIN0	optional: external frequency input or delay compensation	optional: transmit data validation or delay compensation
IIS (Slave)	DIN0	SCLKIN	WAIN

1) ASC, IIC, IIS and standard SSC protocols use only DX0 as the shift data input.

Note: To allow a certain flexibility in assigning required USIC input functions to port pins of the device, each input stage can select the desired input location among several possibilities.
The available USIC signals and their port locations are listed in the interconnects section, see ***Page 14-226****.*

14.2.1.3 Output Signals

For each protocol, up to 14 protocol-related output signals are available. The number of actually used outputs depends on the selected protocol. They can be classified according to their meaning for the protocols, see **Table 14-3**.

The outputs marked as “optional” are not needed for the standard function of a protocol and may be used for enhancements. The descriptions of protocol-specific items are given in the related protocol chapters. The MCLKOUT output signal has a stable frequency relation to the shift clock output (the frequency of MCLKOUT can be higher than for SCLKOUT) for synchronization purposes of a slave device to a master device. If the baud rate generator is not needed for a specific protocol (e.g. in SSC slave mode), the SCLKOUT and MCLKOUT signals can be used as clock outputs with 50% duty cycle with a frequency that can be independent from the communication baud rate.

Table 14-3 Output Signals for Different Protocols

Selected Protocol	Shift Data Output(s) DOUT[3:0][1)]	Shift Clock Output SCLKOUT	Shift Control Outputs SELO[7:0]	Master Clock Output MCLKOUT
ASC, LIN	TXD	not used	not used	optional: master time base
Standard SSC, SPI (Master)	DOUT0 (MTSR, MOSI)	master shift clock	slave select, chip select	optional: master time base
Standard SSC, SPI (Slave)	DOUT0 (MRST, MISO)	optional: independent clock output	not used	optional: independent clock output
Dual-SSC, SPI (Master)	DOUT[1:0] (MTSR[1:0], MOSI[1:0])	master shift clock	slave select, chip select	optional: master time base
Dual-SSC, SPI (Slave)	DOUT[1:0] (MRST[1:0], MISO[1:0])	optional: independent clock output	not used	optional: independent clock output

Table 14-3 Output Signals for Different Protocols (cont'd)

Selected Protocol	Shift Data Output(s) DOUT[3:0][1)]	Shift Clock Output SCLKOUT	Shift Control Outputs SELO[7:0]	Master Clock Output MCLKOUT
Quad-SSC, SPI (Master)	DOUT[3:0] (MTSR[3:0], MOSI[3:0])	master shift clock	slave select, chip select	optional: master time base
Quad-SSC, SPI (Slave)	DOUT[3:0] (MRST[3:0], MISO[3:0])	optional: independent clock output	not used	optional: independent clock output
IIC	SDA	SCL	not used	optional: master time base
IIS (master)	DOUT0	master shift clock	WA	optional: master time base
IIS (slave)	DOUT0	optional: independent clock output	not used	optional: independent clock output

1) ASC, IIC, IIS and standard SSC protocols use only DOUT0 as the shift data output.

Note: To allow a certain flexibility in assigning required USIC output functions to port pins of the device, most output signals are made available on several port pins. The port control itself defines pin-by-pin which signal is used as output signal for a port pin (see port chapter).
The available USIC signals and their port locations are listed in the interconnects section, see ***Page 14-226****.*

14.2.1.4 Baud Rate Generator

Each USIC Channel contains a baud rate generator structured as shown in **Figure 14-2**. It is based on coupled divider stages, providing the frequencies needed for the different protocols. It contains:

- A fractional divider to generate the input frequency $f_{PIN} = f_{FD}$ for baud rate generation based on the internal system frequency f_{PERIPH}.
- The DX1 input to generate the input frequency $f_{PIN} = f_{DX1}$ for baud rate generation based on an external signal.
- Two protocol-related counters: the divider mode counter to provide the master clock signal MCLK, the shift clock signal SCLK, and other protocol-related signals; and the capture mode timer for time interval measurement, e.g. baud rate detection.
- A time quanta counter associated to the protocol pre-processor defining protocol-specific timings, such shift control signals or bit timings, based on the input frequency f_{CTQIN}.

- The output signals MCLKOUT and SCLKOUT of the protocol-related divider that can be made available on pins. In order to adapt to different applications, some output characteristics of these signals can be configured.
  For device-specific details about availability of USIC signals on pins please refer to the interconnects section.

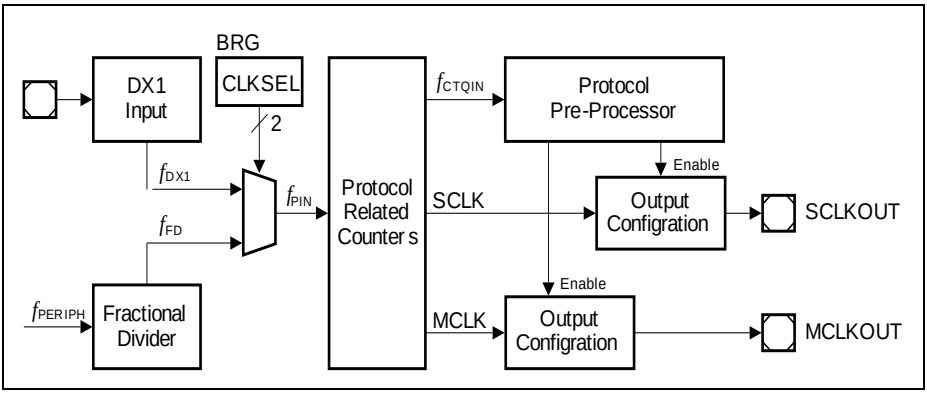


Figure 14-2 Baud Rate Generator

14.2.1.5 Channel Events and Interrupts

The notification of the user about events occurring during data traffic and data handling is based on:

- Data transfer events related to the transmission or reception of a data word, independent of the selected protocol.
- Protocol-specific events depending on the selected protocol.
- Data buffer events related to data handling by the optional FIFO data buffers.

14.2.1.6 Data Shifting and Handling

The data handling of the USIC module is based on an independent data shift unit (DSU) and a buffer structure that is similar for the supported protocols. The data shift and buffer registers are 16-bit wide (maximum data word length), but several data words can be concatenated to achieve longer data frames. The DSU inputs are the shift data (handled by input stage DX0, DX3, DX4 and DX5), the shift clock (handled by the input stage DX1), and the shift control (handled by the input stage DX2). The signal DOUT[3:0] represents the shift data outputs.

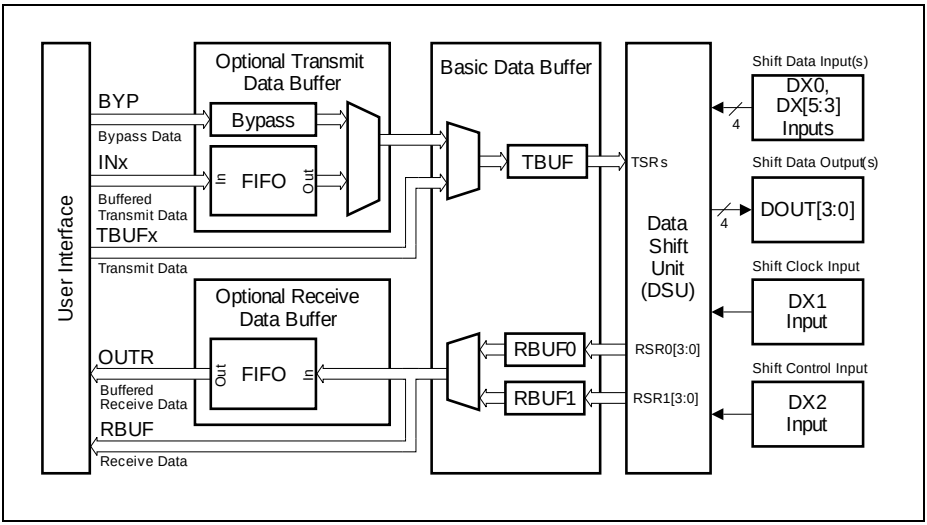


Figure 14-3 Principle of Data Buffering

The principle of data handling comprises:

- A transmitter with transmit shift registers (TSR and TSR[3:0]) in the DSU and a transmit data buffer (TBUF). A data validation scheme allows triggering and gating of data transfers by external events under certain conditions.
- A receiver with two alternating sets of receive shift registers (RSR0[3:0] and RSR1[3:0]) in the DSU and a double receive buffer structure (RBUF0, RBUF1). The alternating receive shift registers support the reception of data streams and data frames longer than one data word.
- A user interface to handle data, interrupts, and status and control information.

Basic Data Buffer Structure

The read access to received data and the write access of data to be transmitted can be handled by a basic data buffer structure.

The received data stored in the receiver buffers RBUF0/RBUF1 can be read directly from these registers. In this case, the user has to take care about the reception sequence to read these registers in the correct order. To simplify the use of the receive buffer structure, register RBUF has been introduced. A read action from this register delivers the data word received first (oldest data) to respect the reception sequence. With a read access from at least the low byte of RBUF, the data is automatically declared to be no longer new and the next received data word becomes visible in RBUF and can be read out next.

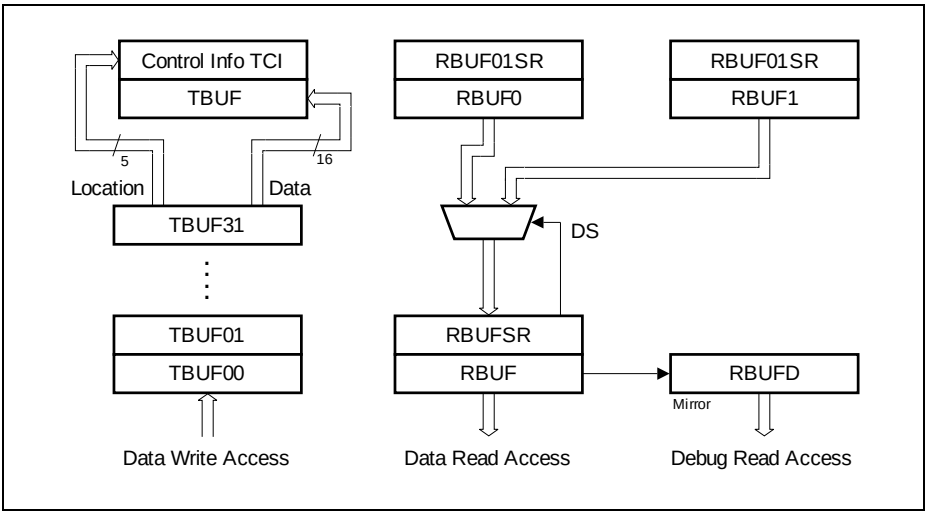


Figure 14-4 Data Access Structure without additional Data Buffer

It is recommended to read the received data words by accesses to RBUF and to avoid handling of RBUF0 and RBUF1. The USIC module also supports the use of debug accesses to receive data words. Debugger read accesses should not disturb the receive data sequence and, as a consequence, should not target RBUF. Therefore, register RBUFD has been introduced. It contains the same value as RBUF, but a read access from RBUFD does not change the status of the data (same data can be read several times). In addition to the received data, some additional status information about each received data word is available in the receiver buffer status register RBUF01SR (related to data in RBUF0 and RBUF1) and RBUFSR (related to data in RBUF).

Transmit data can be loaded to TBUF by software by writing to the transmit buffer input locations TBUFx (x = 00-31), consisting of 32 consecutive addresses. The data written to one of these input locations is stored in the transmit buffer TBUF. Additionally, the address of the written location is evaluated and can be used for additional control purposes. This 5-bit wide information (named **T**ransmit **C**ontrol **I**nformation TCI) can be used for different purposes in different protocols.

FIFO Buffer Structure

To allow easier data setup and handling, an additional data buffering mechanism can be optionally supported. The data buffer is based on the first-in-first-out principle (FIFO) that ensures that the sequence of transferred data words is respected.

If a FIFO buffer structure is used, the data handling scheme (data with associated control information) is similar to the one without FIFO. The additional FIFO buffer can be

independently enabled/disabled for transmission and reception (e.g. if data FIFO buffers are available for a specific USIC channel, it is possible to configure the transmit data path without and the receive data path with FIFO buffering).

The transmit FIFO buffer is addressed by using 32 consecutive address locations for INx instead of TBUFx (x=00-31) regardless of the FIFO depth. The 32 addresses are used to store the 5-bit TCI (together with the written data) associated with each FIFO entry.

The receive FIFO can be read out at two independent addresses, OUTR and OUTDR instead of RBUF and RBUFD. A read from the OUTR location triggers the next data packet to be available for the next read (general FIFO mechanism). In order to allow non-intrusive debugging (without risk of data loss), a second address location (OUTDR) has been introduced. A read at this location delivers the same value as OUTR, but without modifying the FIFO contents.

The transmit FIFO also has the capability to bypass the data stream and to load bypass data to TBUF. This can be used to generate high-priority messages or to send an emergency message if the transmit FIFO runs empty. The transmission control of the FIFO buffer can also use the transfer trigger and transfer gating scheme of the transmission logic for data validation (e.g. to trigger data transfers by events).

Note: The available size of a FIFO data buffer for a USIC channel depends on the specific device. Please refer to the implementation chapter for details about available FIFO buffer capability.

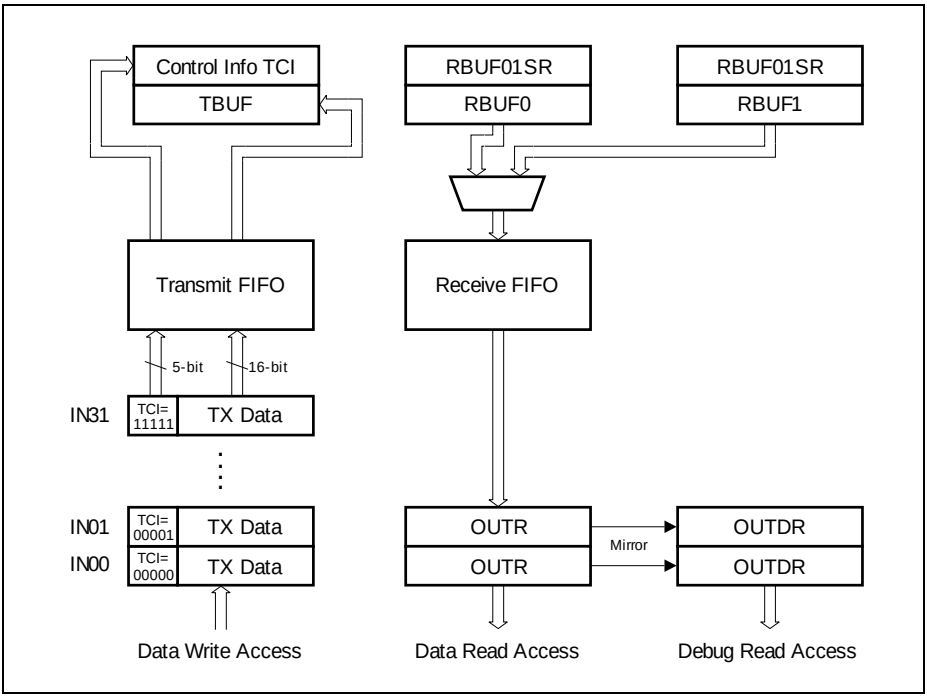


Figure 14-5 Data Access Structure with FIFO

14.2.2 Operating the USIC Communication Channel

This section describes how to operate a USIC communication channel, including protocol control and status, mode control and interrupt handling. The following aspects have to be taken into account:

- Enable the USIC module for operation and configure the behavior for the different device operation modes (see **Page 14-16**).
- Configure the pinning (refer to description in the corresponding protocol section).
- Configure the data structure (shift direction, word length, frame length, polarity, etc.).
- Configure the data buffer structure of the optional FIFO buffer area. A FIFO buffer can only be enabled if the related bit in register CCFG is set.
- Select a protocol by CCR.MODE. A protocol can only be selected if the related bit in register CCFG is set.

14.2.2.1 Protocol Control and Status

The protocol-related control and status information are located in the protocol control register PCR and in the protocol status register PSR. These registers are shared between the available protocols. As a consequence, the meaning of the bit positions in these registers is different within the protocols.

Use of PCR Bits

The signification of the bits in register PCR is indicated by the protocol-related alias names for the different protocols.

- PCR for the ASC protocol (see **Page 14-66**)
- PCR for the SSC protocol (see **Page 14-98**)
- PCR for the IIC protocol (see **Page 14-129**)
- PCR for the IIS protocol (see **Page 14-147**)

Use of PSR Flags

The signification of the flags in register PSR is indicated by the protocol-related alias names for the different protocols.

- PSR flags for the ASC protocol (see **Page 14-69**)
- PSR flags for the SSC protocol (see **Page 14-102**)
- PSR flags for the IIC protocol (see **Page 14-132**)
- PSR flags for the IIS protocol (see **Page 14-150**)

14.2.2.2 Mode Control

The mode control concept for system control tasks, such as suspend request for debugging, allows to program the module behavior under different device operating conditions. The behavior of a communication channel can be programmed for each of the device operating modes (normal operation, suspend mode). Therefore, each communication channel has an associated kernel state configuration register KSCFG defining its behavior in the following operating modes:

- **Normal operation**:
  This operating mode is the default operating mode when no suspend request is pending. The module clock is not switched off and the USIC registers can be read or written. The channel behavior is defined by KSCFG.NOMCFG.
- **Suspend mode**:
  This operating mode is requested when a suspend request is pending in the device. The module clock is not switched off and the USIC registers can be read or written. The channel behavior is defined by KSCFG.SUMCFG.

The four kernel modes defined by the register KSCFG are shown in **Table 14-4**.

Table 14-4 USIC Communication Channel Behavior

Kernel Mode	Channel Behavior	KSCFG. NOMCFG
Run mode 0	Channel operation as specified, no impact on data transfer	00_B
Run mode 1		01_B
Stop mode 0	Explicit stop condition as described in the protocol chapters	10_B
Stop mode 1		11_B

Generally, bit field KSCFG.NOMCFG should be configured for run mode 0 as default setting for standard operation. If a communication channel should not react to a suspend request (and to continue its operation as in normal mode), bit field KSCFG.SUMCFG has to be configured with the same value as KSCFG.NOMCFG. If the communication channel should show a different behavior and stop operation when a specific stop condition is reached, the code for stop mode 0 or stop mode 1 have to be written to KSCFG.SUMCFG.

The stop conditions are defined for the selected protocol (see mode control description in the protocol section).

Note: The stop mode selection strongly depends on the application needs and it is very unlikely that different stop modes are required in parallel in the same application. As a result, only one stop mode type (either 0 or 1) should be used in the bit fields in register KSCFG. Do not mix stop mode 0 and stop mode 1 and avoid transitions

from stop mode 0 to stop mode 1 (or vice versa) for the same communication channel.

14.2.2.3 General Channel Events and Interrupts

The general event and interrupt structure is shown in **Figure 14-6**. If a defined condition is met, an event is detected and an event indication flag becomes automatically set. The flag stays set until it is cleared by software. If enabled, an interrupt can be generated if an event is detected. The actual status of the event indication flag has no influence on the interrupt generation. As a consequence, the event indication flag does not need to be cleared to generate further interrupts.

Additionally, the service request output SRx of the USIC channel that becomes activated in case of an event condition can be selected by an interrupt node pointer. This structure allows to assign events to interrupts, e.g. depending on the application, several events can share the same interrupt routine (several events activate the same SRx output) or can be handled individually (only one event activates one SRx output).

The SRx outputs are connected to interrupt control registers to handle the CPU reaction to the service requests. This assignment is described in the implementation section on **Page 14-153**.

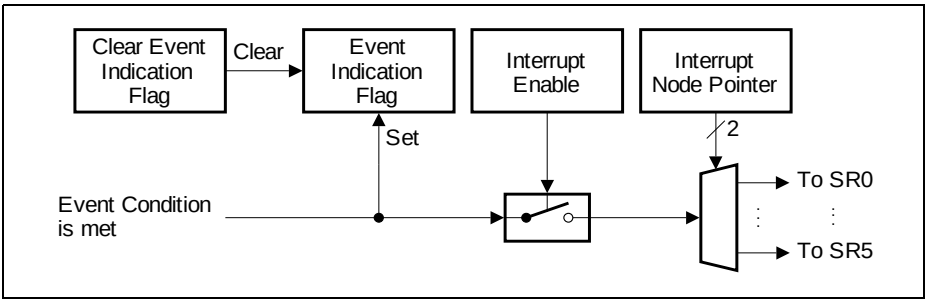


Figure 14-6 General Event and Interrupt Structure

14.2.2.4 Data Transfer Events and Interrupts

The data transfer events are based on the transmission or reception of a data word. The related indication flags are located in register PSR. All events can be individually enabled for interrupt generation.

- Receive event to indicate that a data word has been received:
  If a new received word becomes available in the receive buffer RBUF0 or RBUF1, either a receive event or an alternative receive event occurs.
  The receive event occurs if bit RBUFSR.PERR = 0. It is indicated by flag PSR.RIF and, if enabled, leads to receive interrupt.
- Receiver start event to indicate that a data word reception has started:
  When the receive clock edge that shifts in the first bit of a new data word is detected and reception is enabled, a receiver start event occurs. It is indicated by flag PSR.RSIF and, if enabled, leads to transmit buffer interrupt.
  In full duplex mode, this event follows half a shift clock cycle after the transmit buffer event and indicates when the shift control settings are internally “frozen” for the current data word reception and a new setting can be programmed.
  In SSC and IIS mode, the transmit data valid flag TCSR.TDV is cleared in single shot mode with the receiver start event.
- Alternative receive event to indicate that a specific data word has been received:
  If a new received word becomes available in the receive buffer RBUF0 or RBUF1, either a receive event or an alternative receive event occurs.
  The alternative receive event occurs if bit RBUFSR.PERR = 1. It is indicated by flag PSR.AIF and, if enabled, leads to alternative receive interrupt.
  Depending on the selected protocol, bit RBUFSR.PERR is set to indicate a parity error in ASC mode, the reception of the first byte of a new frame in IIC mode, and the WA information about right/left channel in IIS mode. In SSC mode, it is used as indication if the received word is the first data word, and is set if first and reset if not.
- Transmit shift event to indicate that a data word has been transmitted:
  A transmit shift event occurs with the last shift clock edge of a data word. It is indicated by flag PSR.TSIF and, if enabled, leads to transmit shift interrupt.
- Transmit buffer event to indicate that a data word transmission has been started:
  When a data word from the transmit buffer TBUF has been loaded to the shift register and a new data word can be written to TBUF, a transmit buffer event occurs. This happens with the transmit clock edge that shifts out the first bit of a new data word and transmission is enabled. It is indicated by flag PSR.TBIF and, if enabled, leads to transmit buffer interrupt.
  This event also indicates when the shift control settings (word length, shift direction, etc.) are internally “frozen” for the current data word transmission.
  In ASC and IIC mode, the transmit data valid flag TCSR.TDV is cleared in single shot mode with the transmit buffer event.
- Data lost event to indicate a loss of the oldest received data word:
  If the data word available in register RBUF (oldest data word from RBUF0 or RBUF1)

has not been read out before it becomes overwritten with new incoming data, this event occurs. It is indicated by flag PSR.DLIF and, if enabled, leads to a protocol interrupt.

Table 14-5 shows the registers, bits and bit fields indicating the data transfer events and controlling the interrupts of a USIC channel.

Table 14-5 Data Transfer Events and Interrupt Handling

Event	Indication Flag	Indication cleared by	Interrupt enabled by	SRx Output selected by
Standard receive event	PSR.RIF	PSCR.CRIF	CCR.RIEN	INPR.RINP
Receive start event	PSR.RSIF	PSCR.CRSIF	CCR.RSIEN	INPR.TBINP
Alternative receive event	PSR.AIF	PSCR.CAIF	CCR.AIEN	INPR.AINP
Transmit shift event	PSR.TSIF	PSCR.CTSIF	CCR.TSIEN	INPR.TSINP
Transmit buffer event	PSR.TBIF	PSCR.CTBIF	CCR.TBIEN	INPR.TBINP
Data lost event	PSR.DLIF	PSCR.CDLIF	CCR.DLIEN	INPR.PINP

Figure 14-7 shows the two transmit events and interrupts.

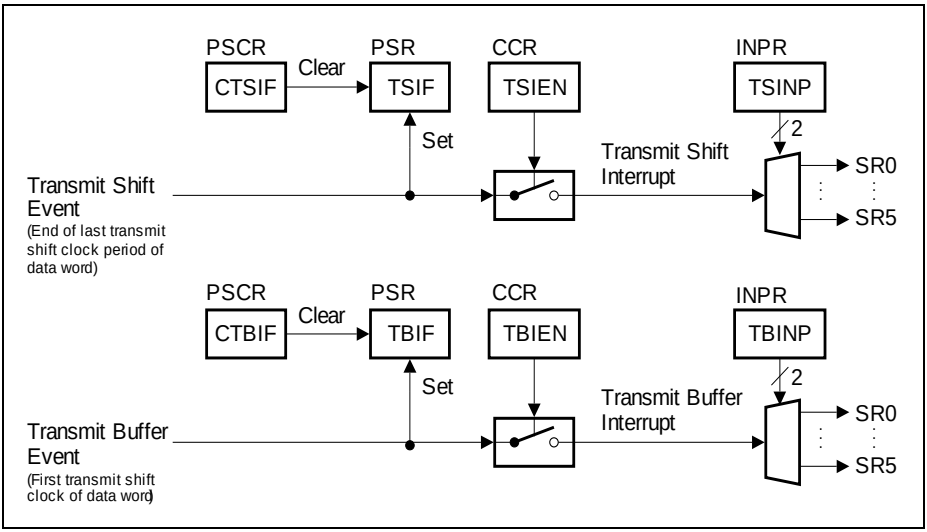


Figure 14-7 Transmit Events and Interrupts

Figure 14-8 shows the receive events and interrupts.

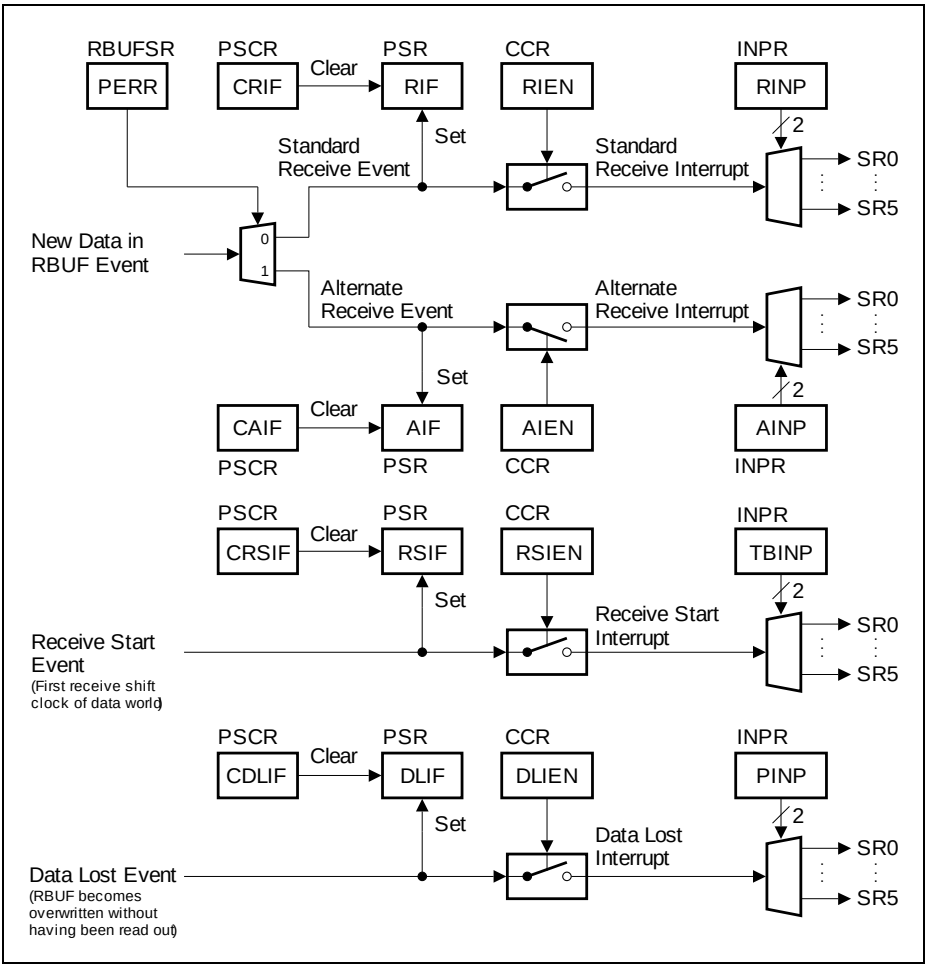


Figure 14-8 Receive Events and Interrupts

14.2.2.5 Baud Rate Generator Event and Interrupt

The baud rate generator event is based on the capture mode timer reaching its maximum value. It is indicated by flag PSR.BRGIF and, if enabled, leads to a protocol interrupt.

Table 14-6 shows the registers, bits and bit fields indicating the baud rate generator event and controlling the interrupt of a USIC channel.

Table 14-6 Baud Rate Generator Event and Interrupt Handling

Event	Indication Flag	Indication cleared by	Interrupt enabled by	SRx Output selected by
Baud rate generator event	PSR. BRGIF	PSCR. CBRGIF	CCR. BRGIEN	INPR.PINP

Figure 14-9 shows the baud rate generator event and interrupt.

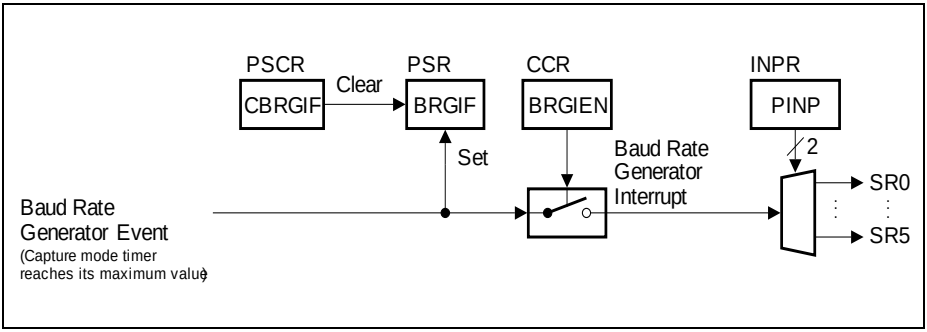


Figure 14-9 Baud Rate Generator Event and Interrupt

14.2.2.6 Protocol-specific Events and Interrupts

These events are related to protocol-specific actions that are described in the corresponding protocol chapters. The related indication flags are located in register PSR. All events can be individually enabled for the generation of the common protocol interrupt.

- Protocol-specific events in ASC mode:
  Synchronization break, data collision on the transmit line, receiver noise, format error in stop bits, receiver frame finished, transmitter frame finished
- Protocol-specific events in SSC mode:
  MSLS event (start-end of frame in master mode), DX2T event (start/end of frame in slave mode), both based on slave select signals, parity error
- Protocol-specific events in IIC mode:
  Wrong transmit code (error in frame sequence), start condition received, repeated start condition received, stop condition received, non-acknowledge received, arbitration lost, slave read request, other general errors
- Protocol-specific events in IIS mode:
  DX2T event (change on WA line), WA falling edge or rising edge detected, WA generation finished

Table 14-7 Protocol-specific Events and Interrupt Handling

Event	Indication Flag	Indication cleared by	Interrupt enabled by	SRx Output selected by
Protocol-specific events in ASC mode	PSR.ST[8:2]	PSCR.CST[8:2]	PCR.CTR[7:3]]	INPR.PINP
Protocol-specific events in SSC mode	PSR.ST[3:2]	PSCR.CST[3:2]	PCR.CTR[15:14]	INPR.PINP
Protocol-specific events in IIC mode	PSR.ST[8:1]	PSCR.CST[8:1]	PCR.CTR[24:18]	INPR.PINP
Protocol-specific events in IIS mode	PSR.ST[6:3]	PSCR.CST[6:3]	PCR.CTR[6:4], PCR.CTR[15]	INPR.PINP

14.2.3 Operating the Input Stages

All input stages offer the same feature set. They are used for all protocols, because the signal conditioning can be adapted in a very flexible way and the digital filters can be switched on and off separately.

14.2.3.1 General Input Structure

There are generally two types of input stages, one for the data input stages DX0, DX[5:3] and the other for non-data input stages DX[2:1], as shown in **Figure 14-10** and **Figure 14-11**. The difference is that for the data input stages, the input signal can be additionally selected from the port signal HWINn if hardware port control is enabled through CCR.HPCEN bit. All other enable/disable functions and selections are controlled independently for each input stage by bits in the registers DXnCR.

The desired input signal can be selected among the input lines DXnA to DXnG and a permanent 1-level by programming bit field DSEL (for the data input stages, hardware port control must be disabled for DSEL to take effect). Please refer to the interconnects section (**Section 14.12**) for the device-specific input signal assignment. Bit DPOL allows a polarity inversion of the selected input signal to adapt the input signal polarity to the internal polarity of the data shift unit and the protocol state machine. For some protocols, the input signals can be directly forwarded to the data shift unit for the data transfers (DSEN = 0, INSW = 1) without any further signal conditioning. In this case, the data path does not contain any delay due to synchronization or filtering.

In the case of noise on the input signals, there is the possibility to synchronize the input signal (signal DXnS is synchronized to f_{PERIPH}) and additionally to enable a digital noise filter in the signal path. The synchronized input signal (and optionally filtered if DFEN = 1) is taken into account by DSEN = 1. Please note that the synchronization leads to a delay in the signal path of 2-3 times the period of f_{PERIPH}.

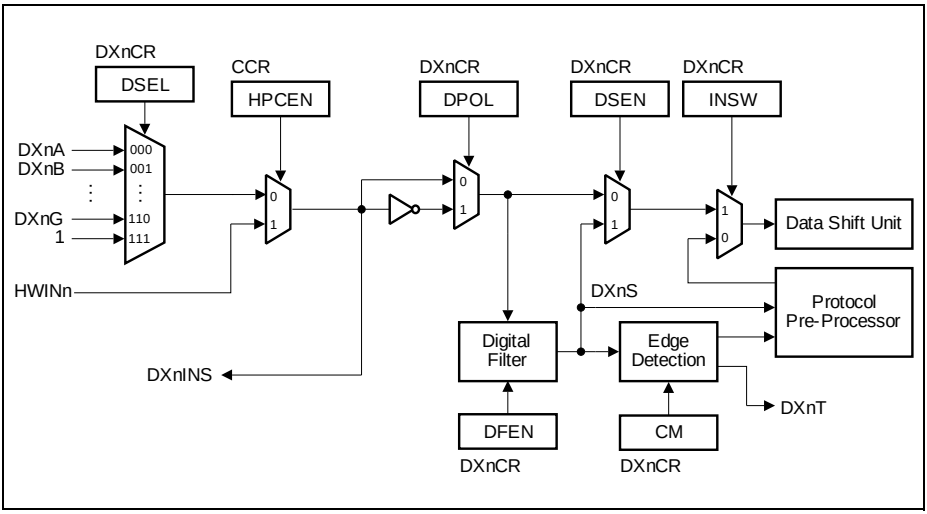


Figure 14-10 Input Conditioning for DX0 and DX[5:3]

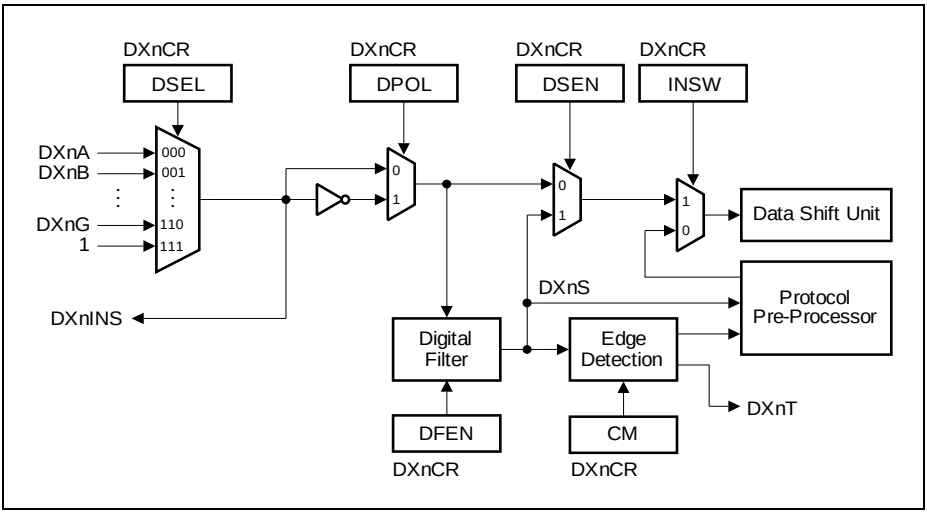


Figure 14-11 Input Conditioning for DX[2:1]

If the input signals are handled by a protocol pre-processor, the data shift unit is directly connected to the protocol pre-processor by INSW = 0. The protocol pre-processor is connected to the synchronized input signal DXnS and, depending on the selected protocol, also evaluates the edges.

To support delay compensation in SSC and IIS protocols, the DX1 input stage additionally allows the receive shift clock to be controlled independently from the transmit shift clock through the bit DCEN. When DCEN = 0, the shift clock source is selected by INSW and is the same for both receive and transmit. When DCEN = 1, the receive shift clock is derived from the selected input line as shown in **Figure 14-12**.

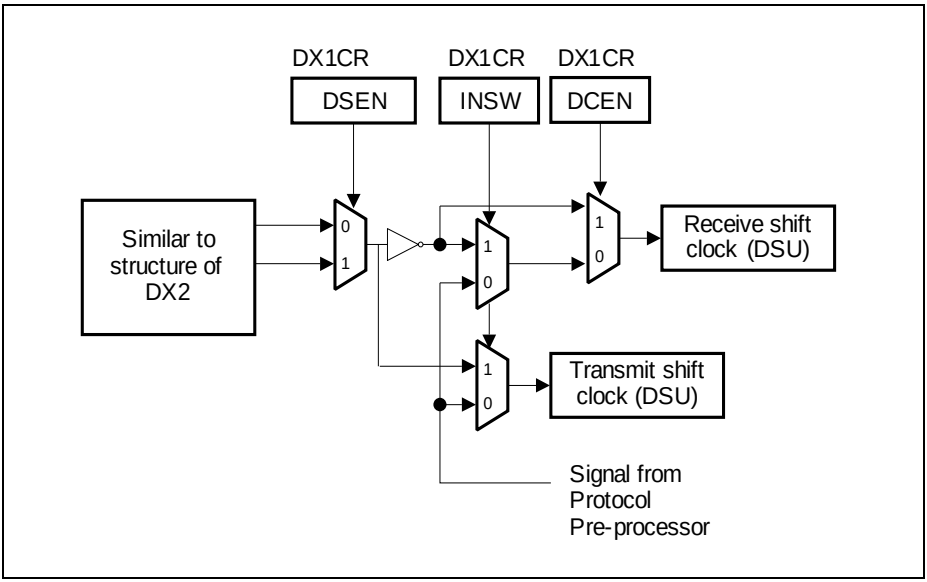


Figure 14-12 Delay Compensation Enable in DX1

14.2.3.2 Digital Filter

The digital filter can be enabled to reduce noise on the input signals. Before being filtered, the input signal becomes synchronized to f_{PERIPH}. If the filter is disabled, signal DXnS corresponds to the synchronized input signal. If the filter is enabled, pulses shorter than one filter sampling period are suppressed in signal DXnS. After an edge of the synchronized input signal, signal DXnS changes to the new value if two consecutive samples of the new value have been detected.

In order to adapt the filter sampling period to different applications, it can be programmed. The first possibility is the system frequency f_{PERIPH}. Longer pulses can be suppressed if the fractional divider output frequency f_{FD} is selected. This frequency is programmable in a wide range and can also be used to determine the baud rate of the data transfers.

In addition to the synchronization delay of 2-3 periods of f_{PERIPH}, an enabled filter adds a delay of up to two filter sampling periods between the selected input and signal DXnS.

14.2.3.3 Edge Detection

The synchronized (and optionally filtered) signal DXnS can be used as input to the data shift unit and is also an input to the selected protocol pre-processor. If the protocol pre-processor does not use the DXnS signal for protocol-specific handling, DXnS can be

used for other tasks, e.g. to control data transmissions in master mode (a data word can be tagged valid for transmission, see chapter about data buffering).

A programmable edge detection indicates that the desired event has occurred by activating the trigger signal DXnT (introducing a delay of one period of f_{PERIPH} before a reaction to this event can take place).

14.2.3.4 Selected Input Monitoring

The selected input signal of each input stage has been made available with the signals DXnINS. These signals can be used in the system to trigger other actions, e.g. to generate interrupts.

14.2.3.5 Loop Back Mode

The USIC transmitter output signals can be connected to the corresponding receiver inputs of the same communication channel in loop back mode. Therefore, the input “G” of the input stages that are needed for the selected protocol have to be selected. In this case, drivers for ASC, SSC, and IIS can be evaluated on-chip without the connections to port pins. Data transferred by the transmitter can be received by the receiver as if it would have been sent by another communication partner.

14.2.4 Operating the Baud Rate Generator

The following blocks can be configured to operate the baud rate generator, see also **Figure 14-2**.

14.2.4.1 Fractional Divider

The fractional divider generates its output frequency f_{FD} by either dividing the input frequency f_{PERIPH} by an integer factor n or by multiplication of n/1024. It has two operating modes:

- Normal divider mode (FDR.DM = 01_B):
  In this mode, the output frequency f_{FD} is derived from the input clock f_{PERIPH} by an integer division by a value between 1 and 1024. The division is based on a counter FDR.RESULT that is incremented by 1 with f_{PERIPH}. After reaching the value $3FF_H$, the counter is loaded with FDR.STEP and then continues counting. In order to achieve $f_{FD} = f_{PERIPH}$, the value of STEP has to be programmed with $3FF_H$.
  The output frequency in normal divider mode is defined by the equation:

(14.1)

$$f_{FD} = f_{PERIPH} \times \frac{1}{n} \qquad \text{with } n = 1024 - STEP$$

- Fractional divider mode (FDR.DM = 10_B):
  In this mode, the output frequency f_{FD} is derived from the input clock f_{PERIPH} by a fractional multiplication of n/1024 for a value of n between 0 and 1023. In general, the fractional divider mode allows to program the average output clock frequency with a finer granularity than in normal divider mode. Please note that in fractional divider mode f_{FD} can have a maximum period jitter of one f_{PERIPH} period. This jitter is not accumulated over several cycles.
  The frequency f_{FD} is generated by an addition of FDR.STEP to FDR.RESULT with f_{PERIPH}. The frequency f_{FD} is based on the overflow of the addition result over $3FF_H$.
  The output frequency in fractional divider mode is defined by the equation:

(14.2)

$$f_{FD} = f_{PERIPH} \times \frac{n}{1024} \qquad \text{with n = STEP}$$

The output frequency f_{FD} of the fractional divider is selected for baud rate generation by BRG.CLKSEL = 00_B.

14.2.4.2 External Frequency Input

The baud rate can be generated referring to an external frequency input (instead of to f_{PERIPH}) if in the selected protocol the input stage DX1 is not needed (DX1CTR.INSW = 0). In this case, an external frequency input signal at the DX1 input stage can be synchronized and sampled with the system frequency f_{PERIPH}. It can be optionally filtered by the digital filter in the input stage. This feature allows data transfers with frequencies that can not be generated by the device itself, e.g. for specific audio frequencies.

If BRG.CLKSEL = 10_B, the trigger signal DX1T determines f_{DX1}. In this mode, either the rising edge, the falling edge, or both edges of the input signal can be used for baud rate generation, depending on the configuration of the DX1T trigger event by bit field DX1CTR.CM. The signal MCLK toggles with each trigger event of DX1T.

If BRG.CLKSEL = 11_B, the rising edges of the input signal can be used for baud rate generation. The signal MCLK represents the synchronized input signal DX1S.

Both, the high time and the low time of external input signal must each have a length of minimum 2 periods of f_{PERIPH} to be used for baud rate generation.

14.2.4.3 Divider Mode Counter

The divider mode counter is used for an integer division delivering the output frequency f_{PDIV}. Additionally, two divider stages with a fixed division by 2 provide the output signals MCLK and SCLK with 50% duty cycle. If the fractional divider mode is used, the maximum fractional jitter of 1 period of f_{PERIPH} can also appear in these signals. The output frequencies of this divider is controlled by register BRG.

In order to define a frequency ratio between the master clock MCLK and the shift clock SCLK, the divider stage for MCLK is located in front of the divider by PDIV+1, whereas the divider stage for SCLK is located at the output of this divider.

$$f_{MCLK} = \frac{f_{PIN}}{2} \tag{14.3}$$

$$f_{SCLK} = \frac{f_{PDIV}}{2} \tag{14.4}$$

In the case that the master clock is used as reference for external devices (e.g. for IIS components) and a fixed phase relation to SCLK and other timing signals is required, it is recommended to use the MCLK signal as input for the PDIV divider. If the MCLK signal is not used or a fixed phase relation is not necessary, the faster frequency f_{PIN} can be selected as input frequency.

$$\begin{aligned} f_{PDIV} &= f_{PIN} \times \frac{1}{PDIV + 1} && \text{if PPPEN = 0} \\ f_{PDIV} &= f_{MCLK} \times \frac{1}{PDIV + 1} && \text{if PPPEN = 1} \end{aligned} \tag{14.5}$$

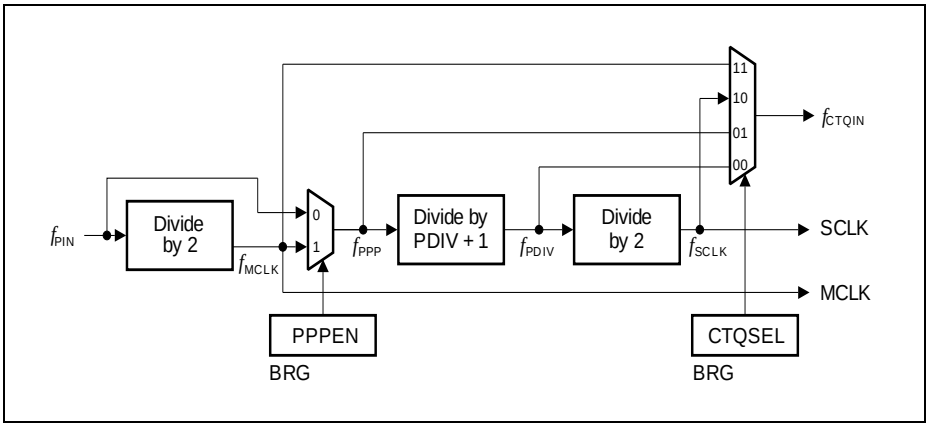


Figure 14-13 Divider Mode Counter

14.2.4.4 Capture Mode Timer

The capture mode timer is used for time interval measurement and is enabled by BRG.TMEN = 1. The timer works independently from the divider mode counter. Therefore, any serial data reception or transmission can continue while the timer is performing timing measurements. The timer counts f_{PPP} periods and stops counting

when it reaches its maximum value. Additionally, a baud rate generator interrupt event is generated (bit PSR.BRGIF becomes set).

If an event is indicated by DX0T or DX1T, the actual timer value is captured into bit field CMTR.CTV and the timer restarts from 0. Additionally, a transmit shift interrupt event is generated (bit PSR.TSIF becomes set).

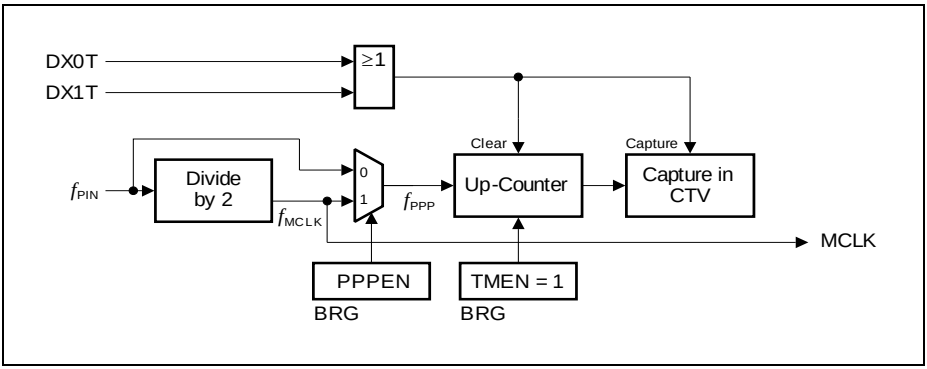


Figure 14-14 Protocol-Related Counter (Capture Mode)

The capture mode timer can be used to measure the baud rate in slave mode before starting or during data transfers, e.g. to measure the time between two edges of a data signal (by DX0T) or of a shift clock signal (by DX1T). The conditions to activate the DXnT trigger signals can be configured in each input stage.

14.2.4.5 Time Quanta Counter

The time quanta counter CTQ associated to the protocol pre-processor allows to generate time intervals for protocol-specific purposes. The length of a time quantum t_q is given by the selected input frequency f_{CTQIN} and the programmed pre-divider value.
The meaning of the time quanta depend on the selected protocol, please refer to the corresponding chapters for more protocol-specific information.

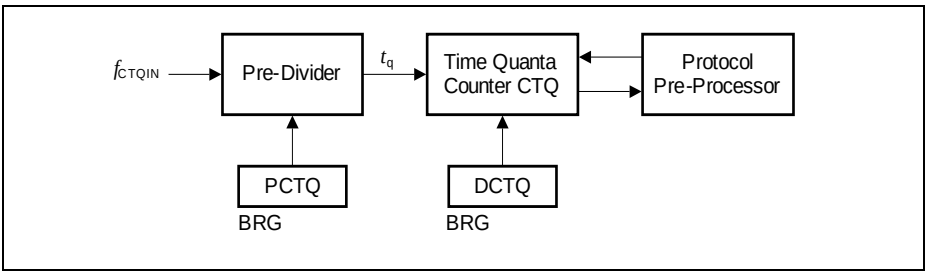


Figure 14-15 Time Quanta Counter

14.2.4.6 Master and Shift Clock Output Configuration

The master clock output signal MCLKOUT available at the corresponding output pin can be configured in polarity. The MCLK signal can be generated for each protocol in order to provide a kind of higher frequency time base compared to the shift clock.

The configuration mechanism of the master clock output signal MCLKOUT ensures that no shortened pulses can occur. Each MCLK period consists of two phases, an active phase, followed by a passive phase. The polarity of the MCLKOUT signal during the active phase is defined by the inverted level of bit BRG.MCLKCFG, evaluated at the start of the active phase. The polarity of the MCLKOUT signal during the passive phase is defined by bit BRG.MCLKCFG, evaluated at the start of the passive phase. If bit BRG.MCLKCFG is programmed with another value, the change is taken into account with the next change between the phases. This mechanism ensures that no shorter pulses than the length of a phase occur at the MCLKOUT output. In the example shown in **Figure 14-16**, the value of BRG.MCLKCFG is changed from 0 to 1 during the passive phase of MCLK period 2.

The generation of the MCLKOUT signal is enabled/disabled by the protocol pre-processor, based on bit PCR.MCLK. After this bit has become set, signal MCLKOUT is generated with the next active phase of the MCLK period. If PCR.MCLK = 0 (MCLKOUT generation disabled), the level for the passive phase is also applied for active phase.

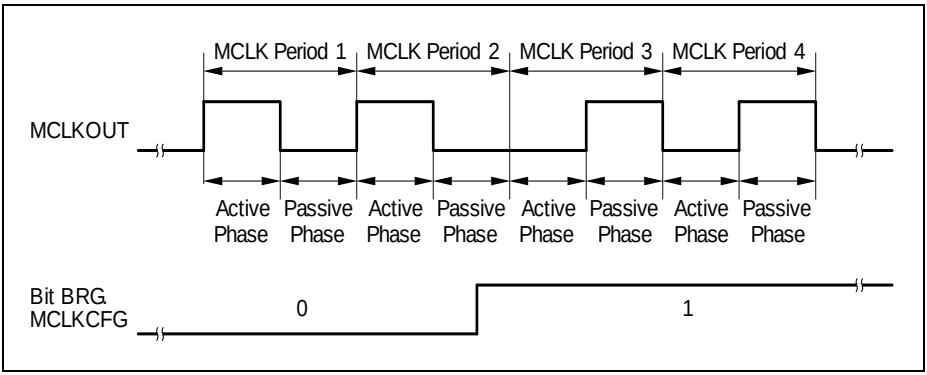


Figure 14-16 Master Clock Output Configuration

The shift clock output signal SCLKOUT available at the corresponding output pin can be configured in polarity and additionally, a delay of one period of f_{PDIV} (= half SCLK period) can be introduced. The delay allows to adapt the order of the shift clock edges to the application requirements. If the delay is used, it has to be taken into account for the calculation of the signal propagation times and loop delays.

The mechanism for the polarity control of the SCLKOUT signal is similar to the one for MCLKOUT, but based on bit field BRG.SCLKCFG. The generation of the SCLKOUT

signal is enabled/disabled by the protocol pre-processor. Depending on the selected protocol, the protocol pre-processor can control the generation of the SCLKOUT signal independently of the divider chain, e.g. for protocols without the need of a shift clock available at a pin, the SCLKOUT generation is disabled.

14.2.5 Operating the Transmit Data Path

The transmit data path is based on 16-bit wide transmit shift registers (TSR and TSR[3:0]) and a transmit buffer TBUF. The data transfer parameters like data word length, data frame length, or the shift direction are controlled commonly for transmission and reception by the shift control register SCTR. The transmit control and status register TCSR controls the transmit data handling and monitors the transmit status.

A change of the value of the data shift output signal DOUTx only happens at the corresponding edge of the shift clock input signal. The level of the last data bit of a data word/frame is held constant at DOUTx until the next data word begins with the next corresponding edge of the shift clock.

14.2.5.1 Transmit Buffering

The transmit shift registers can not be directly accessed by software, because they are automatically updated with the value stored in the transmit buffer TBUF if a currently transmitted data word is finished and new data is valid for transmission. Data words can be loaded directly into TBUF by writing to one of the transmit buffer input locations TBUFx (see **Page 14-33**) or, optionally, by a FIFO buffer stage (see **Page 14-39**).

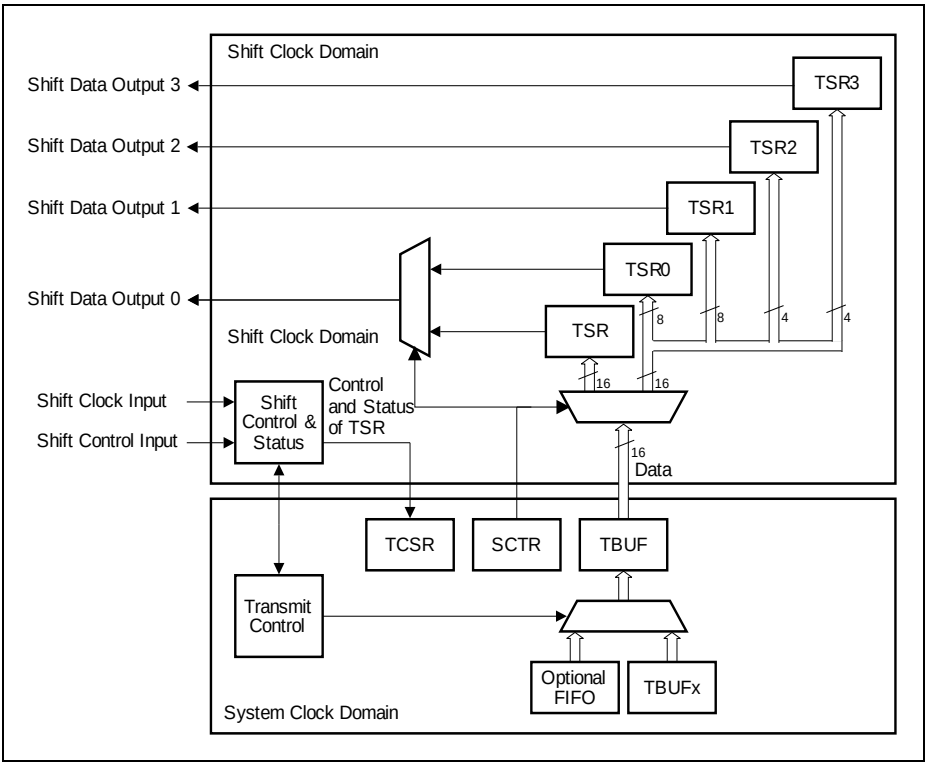


Figure 14-17 Transmit Data Path

14.2.5.2 Transmit Data Shift Mode

The transmit shift data can be selected to be shifted out one, two or four bits at a time through the corresponding number of output lines. This option allows the USIC to support protocols such as the Dual- and Quad-SSC. The selection is done through the bit field DSM in the shift control register SCTR.

Note: The bit field SCTR.DSM controls the data shift mode for both the transmit and receive paths to allow the transmission and reception of data through one to four data lines.

For the shift mode with two or four parallel data outputs, the data word and frame length must be in multiples of two or four respectively. The number of data shifts required to output a specific data word or data frame length is thus reduced by the factor of the number of parallel data output lines. For example, to transmit a 16-bit data word through four output lines, only four shifts are required.

Depending on the shift mode, different transmit shift registers with different bit composition are used as shown in **Table 14-8**. Note that the 'n' in the table denotes the shift number less one, i.e. for the first data shift n = 0, the second data shift n = 1 and continues until the total number of shifts less one is reached.

For all transmit shift registers, whether the first bit shifted out is the MSB or LSB depends on the setting of SCTR.SDIR.

Table 14-8 Transmit Shift Register Composition

Transmit Shift Registers	Single Data Output (SCTR.DSM = 00_B)	Two Data Outputs (SCTR.DSM = 10_B)	Four Data Outputs (SCTR.DSM = 11_B)
TSR	All data bits	Not used	Not used
TSR0	Not used	Bit n*2	Bit n*4
TSR1	Not used	Bit n*2 + 1	Bit n*4 + 1
TSR2	Not used	Not used	Bit n*4 + 2
TSR3	Not used	Not used	Bit n*4 + 3

14.2.5.3 Transmit Control Information

The transmit control information TCI is a 5-bit value derived from the address x of the written TBUFx or INx input location. For example, writing to TBUF31 generates a TCI of 11111_B.

The TCI can be used as an additional control parameter for data transfers to dynamically change the data word length, the data frame length, or other protocol-specific functions (for more details about this topic, please refer to the corresponding protocol chapters). The way how the TCI is used in different applications can be programmed by the bits WLEMD, FLEMD, SELMD, WAMD and HPCMD in register TCSR. Please note that not all possible settings lead to useful system behavior.

- Word length control:
  If TCSR.WLEMD = 1, bit field SCTR.WLE is updated with TCI[3:0] if a transmit buffer input location TBUFx is written. This function can be used in all protocols to dynamically change the data word length between 1 and 16 data bits per data word. Additionally, bit TCSR.EOF is updated with TCI[4]. This function can be used in SSC master mode to control the slave select generation to finish data frames. It is recommended to program TCSR.FLEMD = TCSR.SELMD = TCSR.WAMD = TCSR.HPCMD = 0.
- Frame length control:
  If TCSR.FLEMD = 1, bit field SCTR.FLE[4:0] is updated with TCI[4:0] and SCTR.FLE[5] becomes 0 if a transmit buffer input location TBUFx is written. This function can be used in all protocols to dynamically change the data frame length

between 1 and 32 data bits per data frame. It is recommended to program TCSR.SELMD = TCSR.WLEMD = TCSR.WAMD = TCSR.HPCMD = 0.

- Select output control:
  If TCSR.SELMD = 1, bit field PCR.CTR[20:16] is updated with TCI[4:0] and PCR.CTR[23:21] becomes 0 if a transmit buffer input location TBUFx is written. This function can be used in SSC master mode to define the targeted slave device(s). It is recommended to program TCSR.WLEMD = TCSR.FLEMD = TCSR.WAMD = TCSR.HPCMD = 0.
- Word address control:
  If TCSR.WAMD = 1, bit TCSR.WA is updated with TCI[4] if a transmit buffer input location TBUFx is written. This function can be used in IIS mode to define if the data word is transmitted on the right or the left channel. It is recommended to program TCSR.WLEMD = TCSR.FLEMD = TCSR.SELMD = TCSR.HPCMD = 0.
- Hardware Port control:
  If TCSR.HPCMD = 1, bit field SCTR.DSM is updated with TCI[1:0] if a transmit buffer input location TBUFx is written. This function can be used in SSC protocols to dynamically change the number of data input and output lines to set up for standard, dual and quad SSC formats.
  Additionally, bit TCSR.HPCDIR is updated with TCI[2]. This function can be used in SSC protocols to control the pin(s) direction when the hardware port control function is enabled through CCR.HPCEN = 1. It is recommended to program TCSR.FLEMD = TCSR.WLEMD = TCSR.SELMD = TCSR.WAMD = 0.

14.2.5.4 Transmit Data Validation

The data word in the transmit buffer TBUF can be tagged valid or invalid for transmission by bit TCSR.TDV (transmit data valid). A combination of data flow related and event related criteria define whether the data word is considered valid for transmission. A data validation logic checks the start conditions for each data word. Depending on the result of the check, the transmit shift register is loaded with different values, according to the following rules:

- If a USIC channel is the communication master (it defines the start of each data word transfer), a data word transfer can only be started with valid data in the transmit buffer TBUF. In this case, the transmit shift register is loaded with the content of TBUF, that is not changed due to this action.
- If a USIC channel is a communication slave (it can not define the start itself, but has to react), a data word transfer requested by the communication master has to be started independently of the status of the data word in TBUF. If a data word transfer is requested and started by the master, the transmit shift register is loaded at the first corresponding shift clock edge either with the data word in TBUF (if it is valid for transmission) or with the level defined by bit SCTR.PDL (if the content of TBUF has not been valid at the transmission start). In both cases, the content of TBUF is not changed.

The control and status bits for the data validation are located in register TCSR. The data validation is based on the logic blocks shown in **Figure 14-18**.

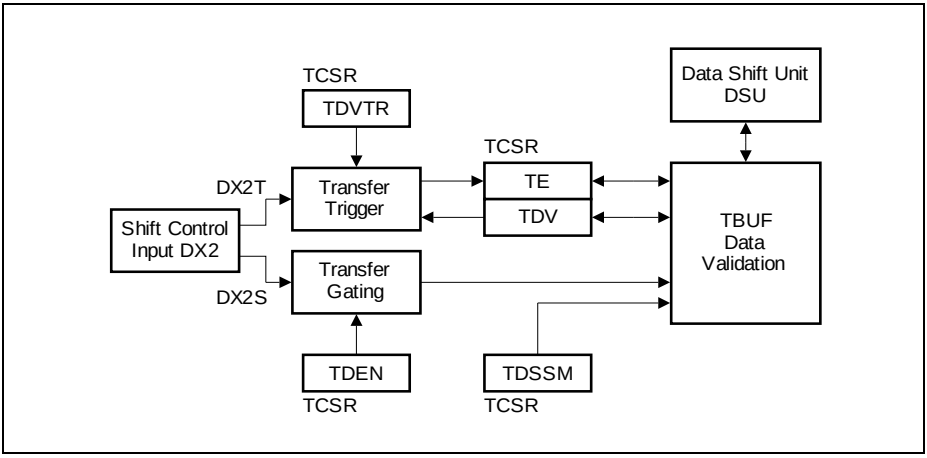


Figure 14-18 Transmit Data Validation

- A transfer gating logic enables or disables the data word transfer from TBUF under software or under hardware control. If the input stage DX2 is not needed for data shifting, signal DX2S can be used for gating purposes. The transfer gating logic is controlled by bit field TCSR.TDEN.
- A transfer trigger logic supports data word transfers related to events, e.g. timer based or related to an input pin. If the input stage DX2 is not needed for data shifting, signal DX2T can be used for trigger purposes. The transfer trigger logic is controlled by bit TCSR.TDVTR and the occurrence of a trigger event is indicated by bit TCSR.TE. For example, this can be used for triggering the data transfer upon receiving the Clear to Send (CTS) signal at DX2 in the RS-232 protocol.
- A data validation logic combining the inputs from the gating logic, the triggering logic and DSU signals. A transmission of the data word located in TBUF can only be started if the gating enables the start, bit TCSR.TDV = 1, and bit TCSR.TE = 1.
  The content of the transmit buffer TBUF should not be overwritten with new data while it is valid for transmission and a new transmission can start. If the content of TBUF has to be changed, it is recommended to clear bit TCSR.TDV by writing FMR.MTDV = 10_B before updating the data. Bit TCSR.TDV becomes automatically set when TBUF is updated with new data. Another possibility are the interrupts TBI (for ASC and IIC) or RSI (for SSC and IIS) indicating that a transmission has started. While a transmission is in progress, TBUF can be loaded with new data. In this case the user has to take care that an update of the TBUF content takes place before a new transmission starts.

With this structure, the following data transfer functionality can be achieved:

- If bit TCSR.TDSSM = 0, the content of the transmit buffer TBUF is always considered as valid for transmission. The transfer trigger mechanism can be used to start the transfer of the same data word based on the selected event (e.g. on a timer base or an edge at a pin) to realize a kind of life-sign mechanism. Furthermore, in slave mode, it is ensured that always a correct data word is transmitted instead of the passive data level.
- Bit TCSR.TDSSM = 1 has to be programmed to allow word-by-word data transmission with a kind of single-shot mechanism. After each transmission start, a new data word has to be loaded into the transmit buffer TBUF, either by software write actions to one of the transmit buffer input locations TBUFx or by an optional data buffer (e.g. FIFO buffer). To avoid that data words are sent out several times or to allow data handling with an additional data buffer (e.g. FIFO), bit TCSR.TDSSM has to be 1.
- Bit TCSR.TDV becoming automatically set when a new data word is loaded into the transmit buffer TBUF, a transmission start can be requested by a write action of the data to be transmitted to at least the low byte of one of the transmit buffer input locations TBUFx. The additional information TCI can be used to control the data word length or other parameters independently for each data word by a single write access.
- Bit field FMR.MTDV allows software driven modification (set or clear) of bit TCSR.TDV. Together with the gating control bit field TCSR.TDEN, the user can set up the transmit data word without starting the transmission. A possible program sequence could be: clear TCSR.TDEN = 00_B, write data to TBUFx, clear TCSR.TDV by writing FMR.MTDV = 10_B, re-enable the gating with TCSR.TDEN = 01_B and then set TCSR.TDV under software control by writing FMR.MTDV = 01_B.

14.2.6 Operating the Receive Data Path

The receive data path is based on two sets of 16-bit wide receive shift registers RSR0[3:0] and RSR1[3:0] and a receive buffer for each of the set (RBUF0 and RBUF1). The data transfer parameters like data word length, data frame length, or the shift direction are controlled commonly for transmission and reception by the shift control registers.

Register RBUF01SR monitors the status of RBUF0 and RBUF1.

14.2.6.1 Receive Buffering

The receive shift registers cannot be directly accessed by software, but their contents are automatically loaded into the receive buffer registers RBUF0 (or RBUF1 respectively) if a complete data word has been received or the frame is finished. The received data words in RBUF0 or RBUF1 can be read out in the correct order directly from register RBUF or, optionally, from a FIFO buffer stage (see **Page 14-39**).

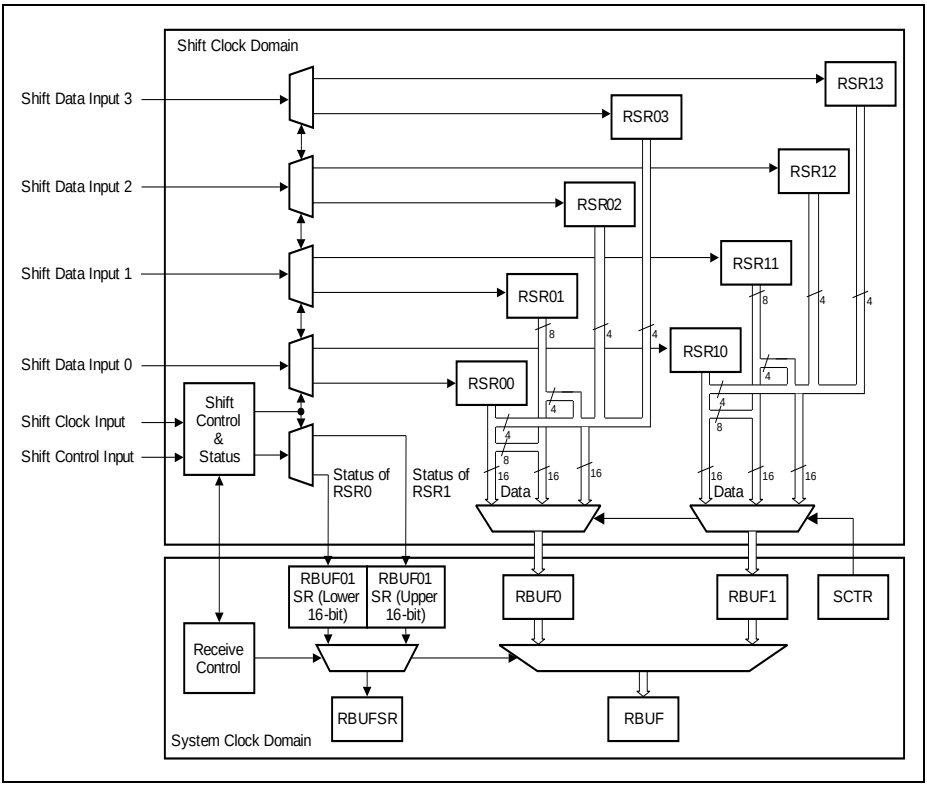


Figure 14-19 Receive Data Path

14.2.6.2 Receive Data Shift Mode

Receive data can be selected to be shifted in one, two or four bits at a time through the corresponding number of input stages and data input lines. This option allows the USIC to support protocols such as the Dual- and Quad-SSC. The selection is done through the bit field DSM in the shift control register SCTR.

Note: The bit field SCTR.DSM controls the data shift mode for both the transmit and receive paths to allow the transmission and reception of data through one to four data lines.

For the shift mode with two or four parallel data inputs, the data word and frame length must be in multiples of two or four respectively. The number of data shifts required to input a specific data word or data frame length is thus reduced by the factor of the

number of parallel data input lines. For example, to receive a 16-bit data word through four input lines, only four shifts are required.

Depending on the shift mode, different receive shift registers with different bit composition are used as shown in **Table 14-8**. Note that the 'n' in the table denotes the shift number less one, i.e. for the first data shift n = 0, the second data shift n = 1 and continues until the total number of shifts less one is reached.

For all receive shift registers, whether the first bit shifted in is the MSB or LSB depends on the setting of SCTR.SDIR.

Table 14-9 Receive Shift Register Composition

Receive Shift Registers	Input stage used	Single Data Input (SCTR.DSM = 00_B)	Two Data Inputs (SCTR.DSM = 10_B)	Four Data Inputs (SCTR.DSM = 11_B)
RSRx0	DX0	All data bits	Bit n*2	Bit n*4
RSRx1	DX3	Not used	Bit n*2 + 1	Bit n*4 + 1
RSRx2	DX4	Not used	Not used	Bit n*4 + 2
RSRx3	DX5	Not used	Not used	Bit n*4 + 3

14.2.6.3 Baud Rate Constraints

The following baud rate constraints have to be respected to ensure correct data reception and buffering. The user has to take care about these restrictions when selecting the baud rate and the data word length with respect to the module clock frequency f_{PERIPH}.

- A received data word in a receiver shift registers RSRx[3:0] must be held constant for at least 4 periods of f_{PERIPH} in order to ensure correct loading of the related receiver buffer register RBUFx.
- The shift control signal has to be constant inactive for at least 5 periods of f_{PERIPH} between two consecutive frames in order to correctly detect the end of a frame.
- The shift control signal has to be constant active for at least 1 period of f_{PERIPH} in order to correctly detect a frame (shortest frame).
- A minimum setup and hold time of the shift control signal with respect to the shift clock signal has to be ensured.

14.2.7 Hardware Port Control

Hardware port control is intended for SSC protocols with half-duplex configurations, where a single port pin is used for both input and output data functions, to control the pin direction through a dedicated hardware interface. All settings in Pn_IOCRy.PCx, except for the input pull device selection and output driver type (open drain or push-pull), are overruled by the hardware port control.

Input pull device selection is done through the Pn_IOCRy.PCx as before, while the output driver is fixed to push-pull-only in this mode.

One, two or four port pins can be selected with the hardware port control to support SSC protocols with multiple bi-directional data lines, such as dual- and quad-SSC. This selection and the enable/disable of the hardware port control is done through CCR.HPCEN. The direction of all selected pins is controlled through a single bit SCTR.HPCDIR.

SCTR.HPCDIR is automatically shadowed with the start of each data word to prevent changing of the pin direction in the middle of a data word transfer.

14.2.8 Operating the FIFO Data Buffer

The FIFO data buffers of a USIC module are built in a similar way, with transmit buffer and receive buffer capability for each channel. Depending on the device, the amount of available FIFO buffer area can vary. In the XMC4[12]00, totally 64 buffer entries can be distributed among the transmit or receive FIFO buffers of both channels of the USIC module.

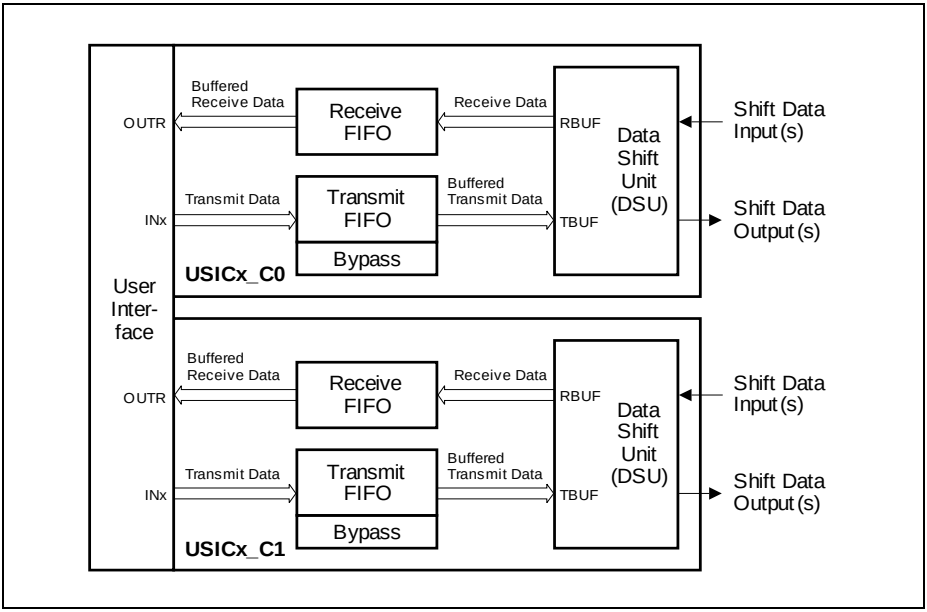


Figure 14-20 FIFO Buffer Overview

In order to operate the FIFO data buffers, the following issues have to be considered:

- FIFO buffer available and selected:
  The transmit FIFO buffer and the bypass structure are only available if CCFG.TB = 1, whereas the receive FIFO buffer is only available if CCFG.RB = 1.
  It is recommended to configure all buffer parameters while there is no data traffic for this USIC channel and the FIFO mechanism is disabled by TBCTR.SIZE = 0 (for transmit buffer) or RBCTR.SIZE = 0 (for receive buffer). The allocation of a buffer area by writing TBCTR or RBCTR has to be done while the corresponding FIFO buffer is disabled. The FIFO buffer interrupt control bits can be modified independently of data traffic.
- FIFO buffer setup:
  The total amount of available FIFO buffer entries limits the length of the transmit and receive buffers for each USIC channel.
- Bypass setup:
  In addition to the transmit FIFO buffer, a bypass can be configured as described on **Page 14-50**.

14.2.8.1 FIFO Buffer Partitioning

If available, the FIFO buffer area consists of a defined number of FIFO buffer entries, each containing a data part and the associated control information (RCI for receive data, TCI for transmit data). One FIFO buffer entry represents the finest granularity that can be allocated to a receive FIFO buffer or a transmit FIFO buffer. All available FIFO buffer entries of a USIC module are located one after the other in the FIFO buffer area. The overall counting starts with FIFO entry 0, followed by 1, 2, etc.

For each USIC module, a certain number of FIFO entries is available, that can be allocated to the channels of the same USIC module. It is not possible to assign FIFO buffer area to USIC channels that are not located within the same USIC module.
For each USIC channel, the size of the transmit and the receive FIFO buffer can be chosen independently. For example, it is possible to allocate the full amount of available FIFO entries as transmit buffer for one USIC channel. Some possible scenarios of FIFO buffer partitioning are shown in **Figure 14-21**.

Each FIFO buffer consists of a set of consecutive FIFO entries. The size of a FIFO data buffer can only be programmed as a power of 2, starting with 2 entries, then 4 entries, then 8 entries, etc. A FIFO data buffer can only start at a FIFO entry aligned to its size. For example, a FIFO buffer containing n entries can only start with FIFO entry 0, n, 2*n, 3*n, etc. and consists of the FIFO entries [x*n, (x+1)*n-1], with x being an integer number (incl. 0). It is not possible to have “holes” with unused FIFO entries within a FIFO buffer, whereas there can be unused FIFO entries between two FIFO buffers.

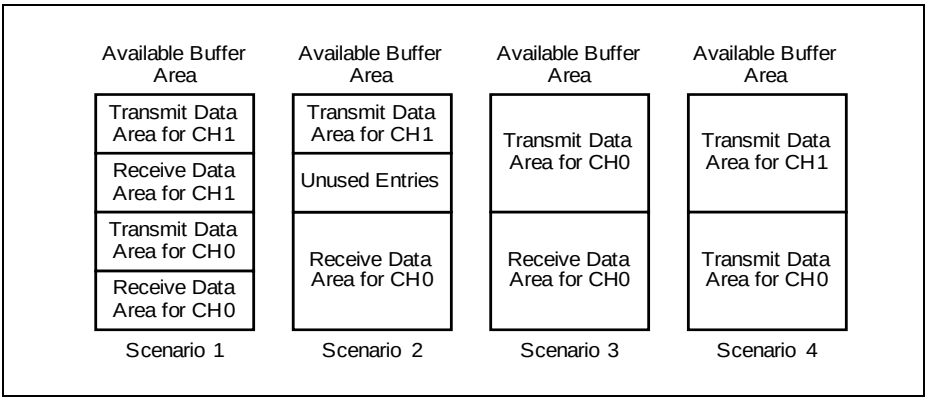


Figure 14-21 FIFO Buffer Partitioning

The data storage inside the FIFO buffers is based on pointers, that are internally updated whenever the data contents of the FIFO buffers have been modified. This happens automatically when new data is put into a FIFO buffer or the oldest data is taken from a FIFO buffer. As a consequence, the user program does not need to modify the pointers for data handling. Only during the initialization phase, the start entry of a FIFO buffer has to be defined by writing the number of the first FIFO buffer entry in the FIFO buffer to the corresponding bit field DPTR in register RBCTR (for a receive FIFO buffer) or TBCTR (for a transmit FIFO buffer) while the related bit field RBCTR.SIZE=0 (or TBCTR.SIZE = 0, respectively). The assignment of buffer entries to a FIFO buffer (regarding to size and pointers) must not be changed by software while the related USIC channel is taking part in data traffic.

14.2.8.2 Transmit Buffer Events and Interrupts

The transmit FIFO buffer mechanism detects the following events, that can lead to interrupts (if enabled):

- Standard transmit buffer event
- Transmit buffer error event

Standard Transmit Buffer Event

The standard transmit buffer event is triggered by the filling level of the transmit buffer (given by TRBSR.TBFLVL) exceeding (TBCTR.LOF = 1) or falling below (TBCTR.LOF = 0)[1)] a programmed limit (TBCTR.LIMIT).

1) If the standard transmit buffer event is used to indicate that new data has to be written to one of the INx locations, TBCTR.LOF = 0 should be programmed.

If the event trigger with TRBSR.STBT feature is disabled (TBCTR.STBTEN = 0), the trigger of the standard transmit buffer event is based on the transition of the fill level from equal to below or above the limit, not the fact of being below or above.

If TBCTR.STBTEN = 1, the transition of the fill level below or above the programmed limit additionally sets TRBSR.STBT. This bit triggers also the standard transmit buffer event whenever there is a transfer data to TBUF event or write data to INx event, depending on TBCTR.LOF setting.

The way TRBSR.STBT is cleared depends on the trigger mode (selected by TBCTR.STBTM). If TBCTR.STBTM = 0, TRBSR.STBT is cleared by hardware when the buffer fill level equals the programmed limit again (TRBSR.TBFLVL = TBCTR.LIMIT). If TBCTR.STBTM = 1, TRBSR.STBT is cleared by hardware when the buffer fill level equals the buffer size (TRBSR.TBFLVL = TBCTR.SIZE).

Note: The flag TRBSR.STBI is set only when the transmit buffer fill level exceeds or falls below the programmed limit (depending on TBCTR.LOF setting). Standard transmit buffer events triggered by TRBSR.STBT does not set the flag.

Figure 14-22 shows examples of the standard transmit buffer event with the different TBCTR.STBTEN and TBCTR.STBTM settings. These examples are meant to illustrate the hardware behaviour and might not always represent real application use cases.

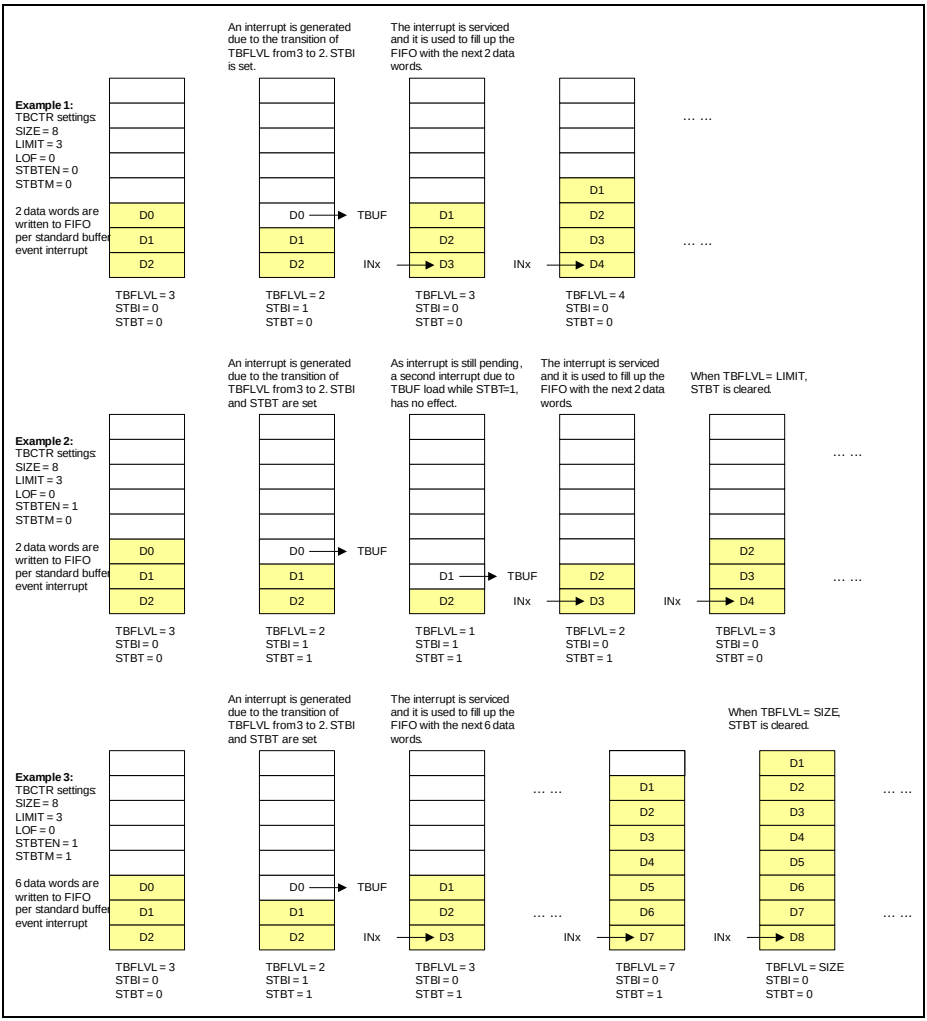


Figure 14-22 Standard Transmit Buffer Event Examples

Transmit Buffer Error Event

The transmit buffer error event is triggered when software has written to a full buffer. The written value is ignored.

Transmit Buffer Events and Interrupt Handling

Figure 14-23 shows the transmit buffer events and interrupts.

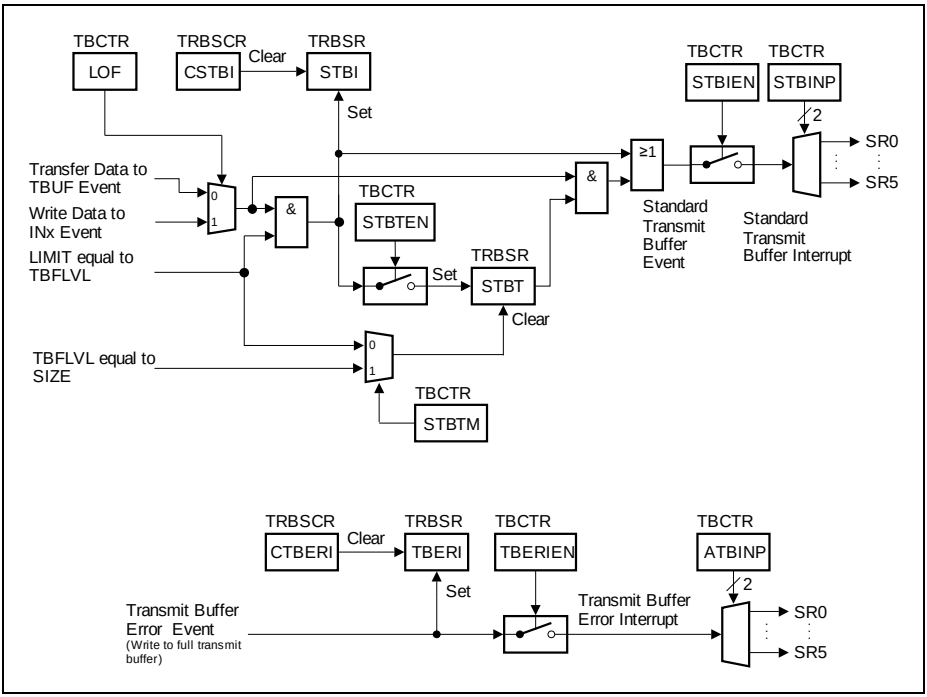


Figure 14-23 Transmit Buffer Events

Table 14-10 shows the registers, bits and bit fields to indicate the transmit buffer events and to control the interrupts related to the transmit FIFO buffers of a USIC channel.

Table 14-10 Transmit Buffer Events and Interrupt Handling

Event	Indication Flag	Indication cleared by	Interrupt enabled by	SRx Output selected by
Standard transmit buffer event	TRBSR.STBI	TRBSCR.CSTBI	TBCTR.STBIEN	TBCTR.STBINP
	TRBSR.STBT	Cleared by hardware		
Transmit buffer error event	TRBSR.TBERI	TRBSCR.CTBERI	TBCTR.TBERIEN	TBCTR.ATBINP

14.2.8.3 Receive Buffer Events and Interrupts

The receive FIFO buffer mechanism detects the following events, that can lead to an interrupt (if enabled):

- Standard receive buffer event
- Alternative receive buffer event
- Receive buffer error event

The standard receive buffer event and the alternative receive buffer event can be programmed to two different modes, one referring to the filling level of the receive buffer, the other one related to a bit position in the receive control information RCI of the data word that becomes available in OUTR.

If the interrupt generation refers to the filling level of the receive FIFO buffer, only the standard receive buffer event is used, whereas the alternative receive buffer event is not used. This mode can be selected to indicate that a certain amount of data has been received, without regarding the content of the associated RCI.

If the interrupt generation refers to RCI, the filling level is not taken into account. Each time a new data word becomes available in OUTR, an event is detected. If bit RCI[4] = 0, a standard receive buffer event is signaled, otherwise an alternative receive buffer device (RCI[4] = 1). Depending on the selected protocol and the setting of RBCTR.RCIM, the value of RCI[4] can hold different information that can be used for protocol-specific interrupt handling (see protocol sections for more details).

Standard Receive Buffer Event in Filling Level Mode

In filling level mode (RBCTR.RNM = 0), the standard receive buffer event is triggered by the filling level of the receive buffer (given by TRBSR.RBFLVL) exceeding (RBCTR.LOF = 1) or falling below (RBCTR.LOF = 0) a programmed limit (RBCTR.LIMIT).[1)]

If the event trigger with bit TRBSR.SRBT feature is disabled (RBCTR.SRBTEN = 0), the trigger of the standard receive buffer event is based on the transition of the fill level from equal to below or above the limit, not the fact of being below or above.

If RBCTR.SRBTEN = 1, the transition of the fill level below or above the programmed limit additionally sets the bit TRBSR.SRBT. This bit also triggers the standard receive buffer event each time there is a data read out event or new data received event, depending on RBCTR.LOF setting.

The way TRBSR.SRBT is cleared depends on the trigger mode (selected by RBCTR.SRBTM). If RBCTR.SRBTM = 0, TRBSR.SRBT is cleared by hardware when the buffer fill level equals the programmed limit again (TRBSR.RBFLVL =

1) If the standard receive buffer event is used to indicate that new data has to be read from OUTR, RBCTR.LOF = 1 should be programmed.

RBCTR.LIMIT). If RBCTR.SRBTM = 1, TRBSR.SRBT is cleared by hardware when the buffer fill level equals 0 (TRBSR.RBFLVL = 0).

Note: The flag TRBSR.SRBI is set only when the receive buffer fill level exceeds or falls below the programmed limit (depending on RBCTR.LOF setting). Standard receive buffer events triggered by TRBSR.SRBT does not set the flag.

Figure 14-24 shows examples of the standard receive buffer event with the different RBCTR.SRBTEN and RBCTR.SRBTM settings. These examples are meant to illustrate the hardware behaviour and might not always represent real application use cases.

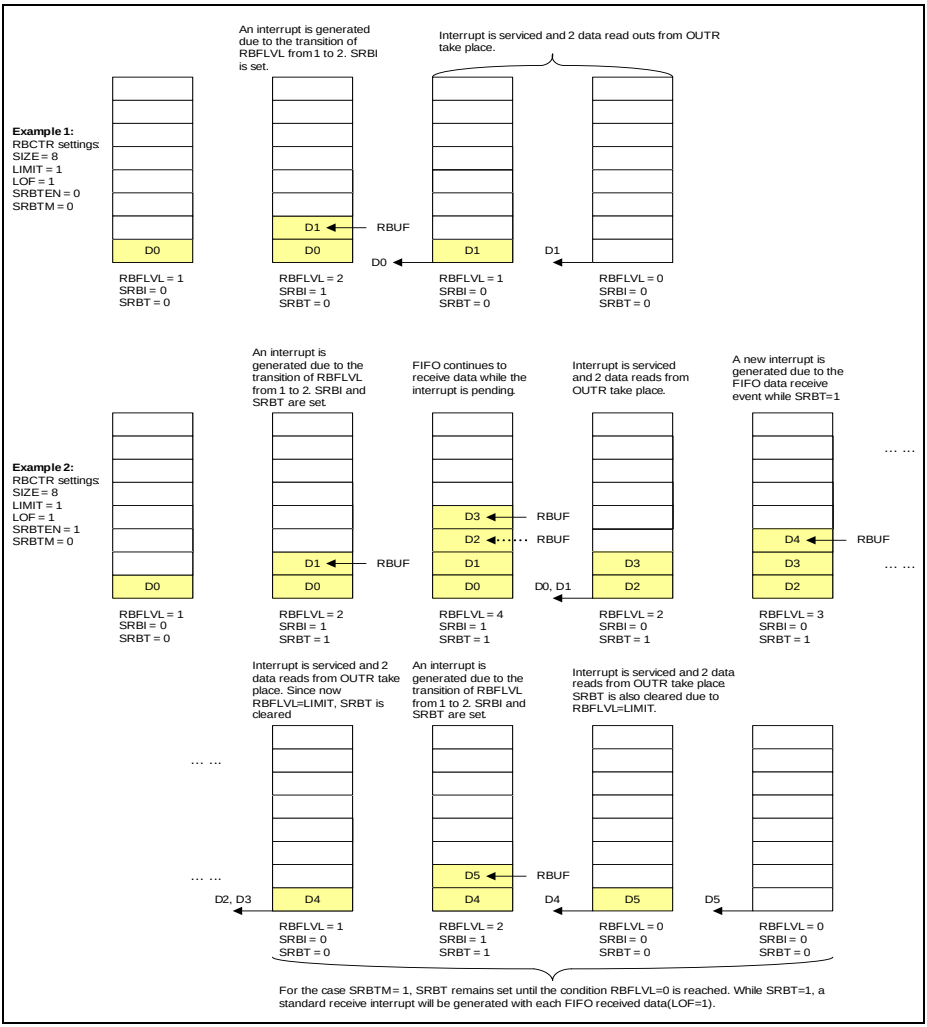


Figure 14-24 Standard Receive Buffer Event Examples

Standard and Alternate Receive Buffer Events in RCI Mode

In RCI mode (RBCTR.RNM = 1), the standard receive buffer event is triggered when the OUTR stage is updated with a new data value with RCI[4] = 0.

If the OUTR stage is updated with a new data value with RCI[4] = 1, an alternate receive buffer event is triggered instead.

Receive Buffer Error Event

The receive buffer error event is triggered if the software reads from an empty buffer, regardless of RBCTR.RNM value. The read data is invalid.

Receive Buffer Events and Interrupt Handling

Figure 14-25 shows the receiver buffer events and interrupts in filling level mode.

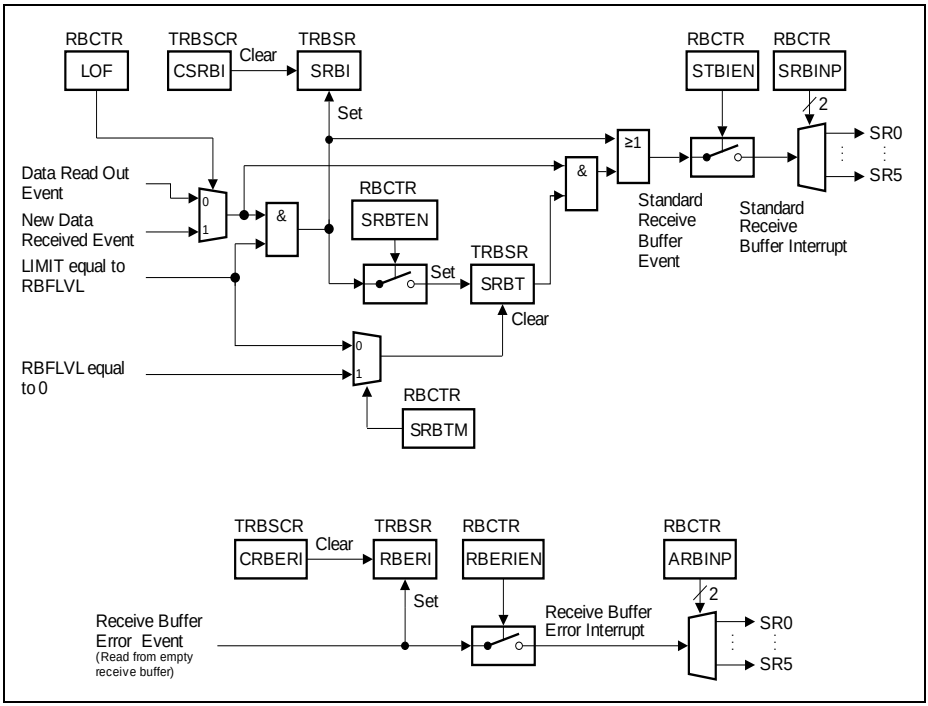


Figure 14-25 Receiver Buffer Events in Filling Level Mode

Figure 14-26 shows the receiver buffer events and interrupts in RCI mode.

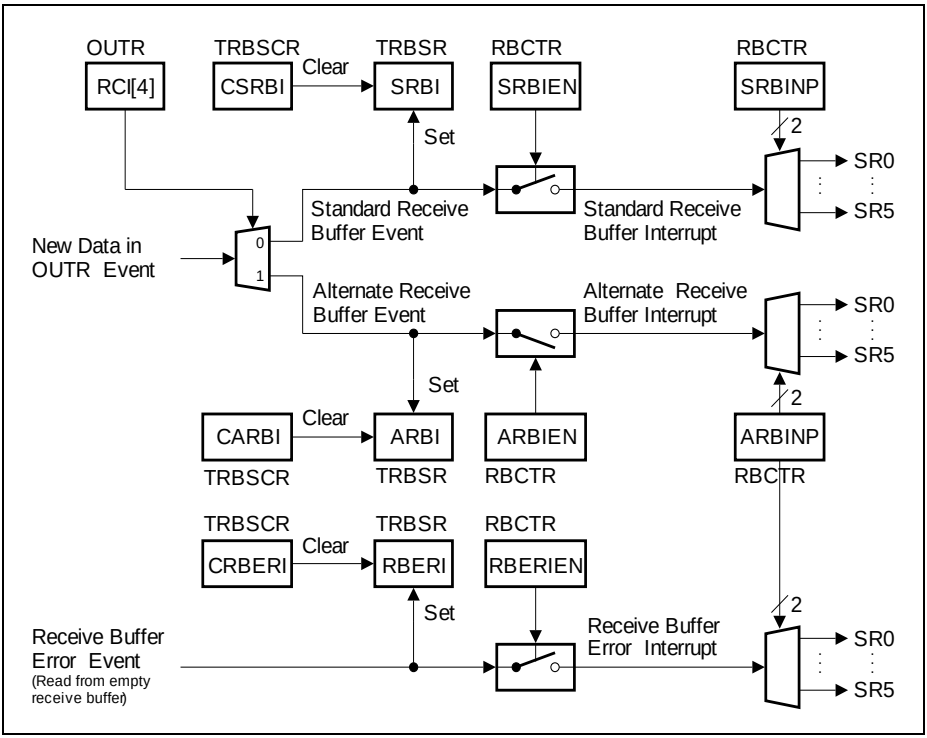


Figure 14-26 Receiver Buffer Events in RCI Mode

Table 14-11 shows the registers, bits and bit fields to indicate the receive buffer events and to control the interrupts related to the receive FIFO buffers of a USIC channel.

Table 14-11 Receive Buffer Events and Interrupt Handling

Event	Indication Flag	Indication cleared by	Interrupt enabled by	SRx Output selected by
Standard receive buffer event	TRBSR. SRBI	TRBSCR. CSRBI	RBCTR. SRBIEN	RBCTR. SRBINP
	TRBSR. SRBT	Cleared by hardware		

Table 14-11 Receive Buffer Events and Interrupt Handling (cont'd)

Event	Indication Flag	Indication cleared by	Interrupt enabled by	SRx Output selected by
Alternative receive buffer event	TRBSR. ARBI	TRBSCR. CARBI	RBCTR. ARBIEN	RBCTR. ARBINP
Receive buffer error event	TRBSR. RBERI	TRBSCR. CRBERI	RBCTR. RBERIEN	RBCTR. ARBINTXDP

14.2.8.4 FIFO Buffer Bypass

The data bypass mechanism is part of the transmit FIFO control block. It allows to introduce a data word in the data stream without modifying the transmit FIFO buffer contents, e.g. to send a high-priority message. The bypass structure consists of a bypass data word of maximum 16 bits in register BYP and some associated control information in register BYPCR. For example, these bits define the word length of the bypass data word and configure a transfer trigger and gating mechanism similar to the one for the transmit buffer TBUF.

The bypass data word can be tagged valid or invalid for transmission by bit BYRCR.BDV (bypass data valid). A combination of data flow related and event related criteria define whether the bypass data word is considered valid for transmission. A data validation logic checks the start conditions for this data word. Depending on the result of the check, the transmit buffer register TBUF is loaded with different values, according to the following rules:

- Data from the transmit FIFO buffer or the bypass data can only be transferred to TBUF if TCSR.TDV = 0 (TBUF is empty).
- Bypass data can only be transferred to TBUF if the bypass is enabled by BYPCR.BDEN or the selecting gating condition is met.
- If the bypass data is valid for transmission and has either a higher transmit priority than the FIFO data or if the transmit FIFO is empty, the bypass data is transferred to TBUF.
- If the bypass data is valid for transmission and has a lower transmit priority than the FIFO buffer that contains valid data, the oldest transmit FIFO data is transferred to TBUF.
- If the bypass data is not valid for transmission and the FIFO buffer contains valid data, the oldest FIFO data is transferred to TBUF.
- If neither the bypass data is valid for transmission nor the transmit FIFO buffer contains valid data, TBUF is unchanged.

The bypass data validation is based on the logic blocks shown in **Figure 14-27**.

- A transfer gating logic enables or disables the bypass data word transfer to TBUF under software or under hardware control. If the input stage DX2 is not needed for

data shifting, signal DX2S can be used for gating purposes. The transfer gating logic is controlled by bit field BYPCR.BDEN.

- A transfer trigger logic supports data word transfers related to events, e.g. timer based or related to an input pin. If the input stage DX2 is not needed for data shifting, signal DX2T can be used for trigger purposes. The transfer trigger logic is controlled by bit BYPCR.BDVTR.
- A bypass data validation logic combining the inputs from the gating logic, the triggering logic and TCSR.TDV.

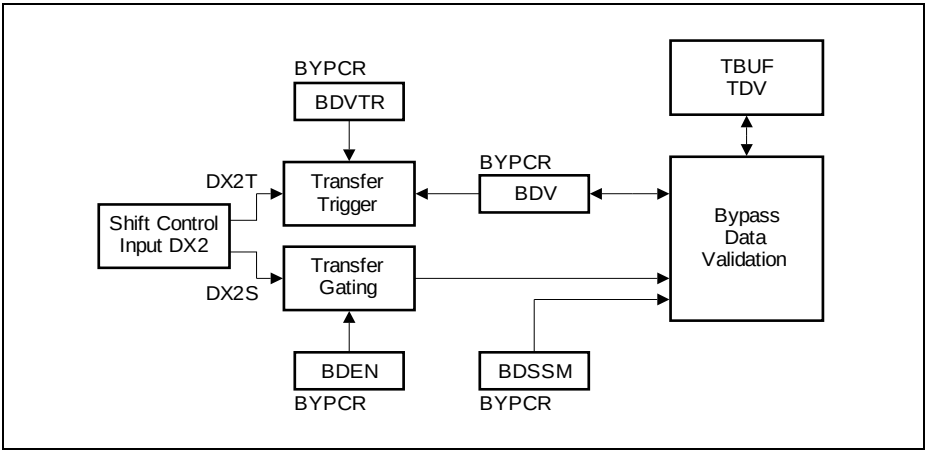


Figure 14-27 Bypass Data Validation

With this structure, the following bypass data transfer functionality can be achieved:

- Bit BYPCR.BDSSM = 1 has to be programmed for a single-shot mechanism. After each transfer of the bypass data word to TBUF, the bypass data word has to be tagged valid again. This can be achieved either by writing a new bypass data word to BYP or by DX2T if BDVTR = 1 (e.g. trigger on a timer base or an edge at a pin).
- Bit BYPCR.BDSSM = 0 has to be programmed if the bypass data is permanently valid for transmission (e.g. as alternative data if the data FIFO runs empty).

14.2.8.5 FIFO Access Constraints

The data in the shared FIFO buffer area is accessed by the hardware mechanisms for data transfer of each communication channel (for transmission and reception) and by software to read out received data or to write data to be transmitted. As a consequence, the data delivery rate can be limited by the FIFO mechanism. Each access by hardware to the FIFO buffer area has priority over a software access, that is delayed in case of an access collision.

In order to avoid data loss and stalling of the CPU due to delayed software accesses, the

baud rate, the word length and the software access mechanism have to be taken into account. Each access to the FIFO data buffer area by software or by hardware takes one period of f_{PERIPH}. Especially a continuous flow of very short, consecutive data words can lead to an access limitation.

14.2.8.6 Handling of FIFO Transmit Control Information

In addition to the transmit data, the transmit control information TCI can be transferred from the transmit FIFO or bypass structure to the USIC channel. Depending on the selected protocol and the enabled update mechanism, some settings of the USIC channel parameters can be modified. The modifications are based on the TCI of the FIFO data word loaded to TBUF or by the bypass control information if the bypass data is loaded into TBUF.

- TCSR.SELMD = 1: update of PCR.CTR[20:16] by FIFO TCI or BYPCR.BSELO with additional clear of PCR.CTR[23:21]
- TCSR.WLEMD = 1: update of SCTR.WLE and TCSR.EOF by FIFO TCI or BYPCR.BWLE (if the WLE information is overwritten by TCI or BWLE, the user has to take care that FLE is set accordingly)
- TCSR.FLEMD = 1: update of SCTR.FLE[4:0] by FIFO TCI or BYPCR.BWLE with additional clear of SCTR.FLE[5]
- TCSR.HPCMD = 1: update of SCTR.DSM and SCTR.HPCDIR by FIFO TCI or BYPCR.BHPC
- TCSR.WAMD = 1: update of TCSR.WA by FIFO TCI[4]

See **Section 14.2.5.3** for more details on TCI.

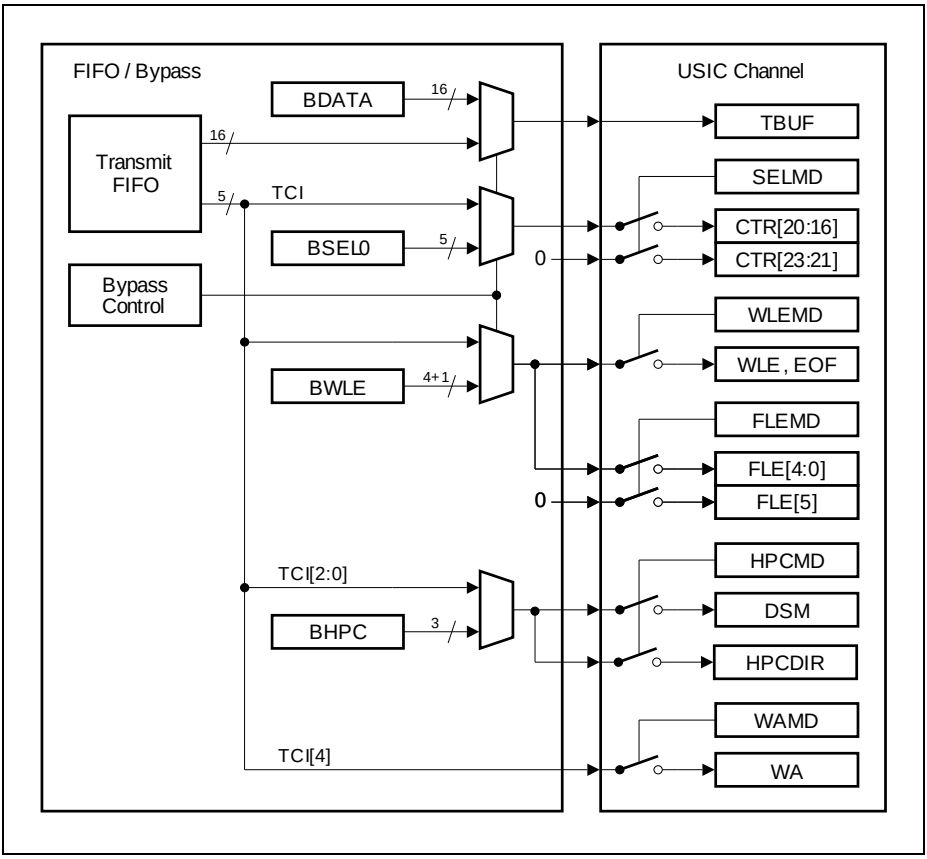


Figure 14-28 TCI Handling with FIFO / Bypass

14.3 Asynchronous Serial Channel (ASC = UART)

The asynchronous serial channel ASC covers the reception and the transmission of asynchronous data frames and provides a hardware LIN support. The receiver and transmitter being independent, frames can start at different points in time for transmission and reception. The ASC mode is selected by CCR.MODE = 0010_B with CCFG.ASC = 1 (ASC mode available).

14.3.1 Signal Description

An ASC connection is characterized by the use of a single connection line between a transmitter and a receiver. The receiver input RXD signal is handled by the input stage DX0.

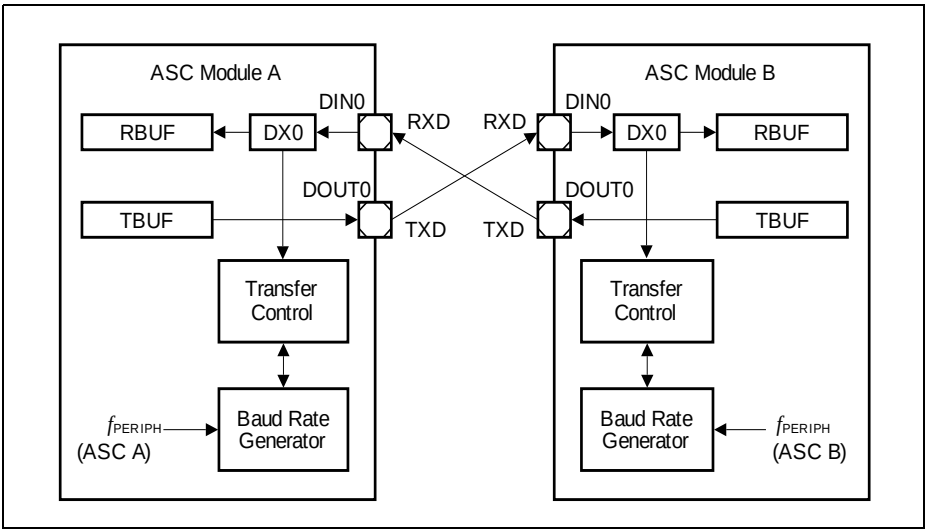


Figure 14-29 ASC Signal Connections for Full-Duplex Communication

For full-duplex communication, an independent communication line is needed for each transfer direction. **Figure 14-29** shows an example with a point-to-point full-duplex connection between two communication partners ASC A and ASC B.

For half-duplex or multi-transmitter communication, a single communication line is shared between the communication partners. **Figure 14-30** shows an example with a point-to-point half-duplex connection between ASC A and ASC B. In this case, the user has to take care that only one transmitter is active at a time. In order to support transmitter collision detection, the input stage DX1 can be used to monitor the level of the transmit line and to check if the line is in the idle state or if a collision occurred.
There are two possibilities to connect the receiver input DIN0 to the transmitter output

DOUT0. Communication partner ASC A uses an internal connection with only the transmit pin TXD, that is delivering its input value as RXD to the DX0 input stage for reception and to DX1 to check for transmitter collisions. Communication partner ASC B uses an external connection between the two pins TXD and RXD.

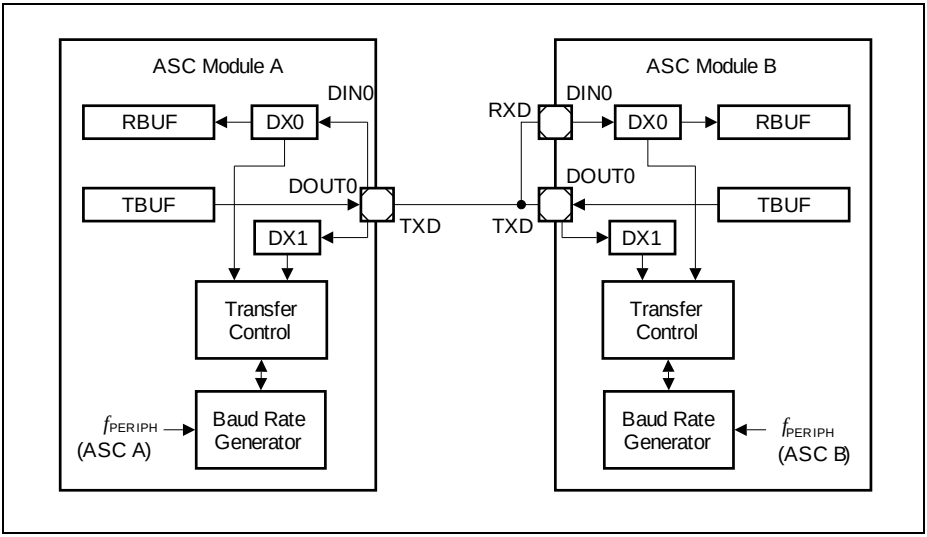


Figure 14-30 ASC Signal Connections for Half-Duplex Communication

14.3.2 Frame Format

A standard ASC frame is shown in **Figure 14-31**. It consists of:

- An idle time with the signal level 1.
- One start of frame bit (SOF) with the signal level 0.
- A data field containing a programmable number of data bits (1-63).
- A parity bit (P), programmable for either even or odd parity. It is optionally possible to handle frames without parity bit.
- One or two stop bits with the signal level 1.

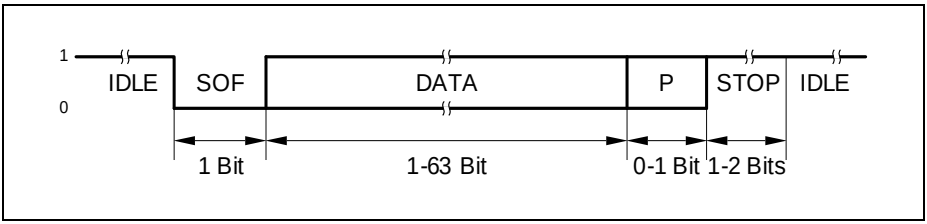


Figure 14-31 Standard ASC Frame Format

The protocol specific bits (SOF, P, STOP) are automatically handled by the ASC protocol state machine and do not appear in the data flow via the receive and transmit buffers.

14.3.2.1 Idle Time

The receiver and the transmitter independently check the respective data input lines (DX0, DX1) for being idle. The idle detection ensures that an SOF bit of a recently enabled ASC module does not collide with an already running frame of another ASC module.

In order to start the idle detection, the user software has to clear bits PSR.RXIDLE and/or PSR.TXIDLE, e.g. before selecting the ASC mode or during operation. If a bit is cleared by software while a data transfer is in progress, the currently running frame transfer is finished normally before starting the idle detection again. Frame reception is only possible if PSR.RXIDLE = 1 and frame transmission is only possible if PSR.TXIDLE = 1. The duration of the idle detection depends on the setting of bit PCR.IDM. In the case that a collision is not possible, the duration can be shortened and the bus can be declared as being idle by setting PCR.IDM = 0.

In the case that the complete idle detection is enabled by PCR.IDM = 1, the data input of DX0 is considered as idle (PSR.RXIDLE becomes set) if a certain number of consecutive passive bit times has been detected. The same scheme applies for the transmitter's data input of DX1. Here, bit PSR.TXIDLE becomes set if the idle condition of this input signal has been detected.

The duration of the complete idle detection is given by the number of programmed data bits per frame plus 2 (in the case without parity) or plus 3 (in the case with parity). The counting of consecutive bit times with 1 level restarts from the beginning each time an edge is found, after leaving a stop mode or if ASC mode becomes enabled.

If the idle detection bits PSR.RXIDLE and/or TXIDLE are cleared by software, the counting scheme is not stopped (no re-start from the beginning). As a result, the cleared bit(s) can become set immediately again if the respective input line still meets the idle criterion.

Please note that the idle time check is based on bit times, so the maximum time can be up to 1 bit time more than programmed value (but not less).

14.3.2.2 Start Bit Detection

The receiver input signal DIN0 (selected signal of input stage DX0) is checked for a falling edge. An SOF bit is detected when a falling edge occurs while the receiver is idle or after the sampling point of the last stop bit. To increase noise immunity, the SOF bit timing starts with the first falling edge that is detected. If the sampled bit value of the SOF is 1, the previous falling edge is considered to be due to noise and the receiver is considered to be idle again.

14.3.2.3 Data Field

The length of the data field (number of data bits) can be programmed by bit field SCTR.FLE. It can vary between 1 and 63 data bits, corresponding to values of SCTR.FLE = 0 to 62 (the value of 63 is reserved and must not be programmed in ASC mode).

The data field can consist of several data words, e.g. a transfer of 12 data bits can be composed of two 8-bit words, with the 12 bits being split into 8-bits of the first word and 4 bits of the second word. The user software has to take care that the transmit data is available in-time, once a frame has been started. If the transmit buffer runs empty during a running data frame, the passive data level (SCTR.PDL) is sent out.

The shift direction can be programmed by SCTR.SDIR. The standard setting for ASC frames with LSB first is achieved with the default setting SDIR = 0.

14.3.2.4 Parity Bit

The ASC allows parity generation for transmission and parity check for reception on frame base. The type of parity can be selected by bit field CCR.PM, common for transmission and reception (no parity, even or odd parity). If the parity handling is disabled, the ASC frame does not contain any parity bit. For consistency reasons, all communication partners have to be programmed to the same parity mode.

After the last data bit of the data field, the transmitter automatically sends out its calculated parity bit if parity generation has been enabled. The receiver interprets this bit as received parity and compares it to its internally calculated one. The received parity bit value and the result of the parity check are monitored in the receiver buffer status registers, RBUFSR and RBUF01SR, as receiver buffer status information. These registers contain bits to monitor a protocol-related argument (PAR) and protocol-related error indication (PERR).

14.3.2.5 Stop Bit(s)

Each ASC frame is completed by 1 or 2 of stop bits with the signal level 1 (same level as the idle level). The number of stop bits is programmable by bit PSR.STPB. A new start bit can be transferred directly after the last stop bit.

14.3.3 Operating the ASC

In order to operate the ASC protocol, the following issues have to be considered:

- Select ASC mode:
  It is recommended to configure all parameters of the ASC that do not change during run time while CCR.MODE = 0000_B. Bit field SCTR.TRM = 01_B has to be programmed. The configuration of the input stages has to be done while CCR.MODE = 0000_B to avoid unintended edges of the input signals and the ASC mode can be enabled by CCR.MODE = 0010_B afterwards.
- Pin connections:
  Establish a connection of input stage DX0 with the selected receive data input pin (signal DIN0) with DX0CR.INSW = 0 and configure a transmit data output pin (signal DOUT0). For collision or idle detection of the transmitter, the input stage DX1 has to be connected to the selected transmit output pin, also with DX1CR.INSW = 0. Additionally, program DX2CR.INSW = 0.
  Due to the handling of the input data stream by the synchronous protocol handler, the propagation delay of the synchronization in the input stage has to be considered.
  Note that the step to enable the alternate output port functions should only be done after the ASC mode is enabled, to avoided unintended spikes on the output.
- Bit timing configuration:
  The desired baud rate setting has to be selected, comprising the fractional divider, the baud rate generator and the bit timing. Please note that not all feature combinations can be supported by the application at the same time, e.g. due to propagation delays. For example, the length of a frame is limited by the frequency difference of the transmitter and the receiver device. Furthermore, in order to use the average of samples (SMD = 1), the sampling point has to be chosen to respect the signal settling and data propagation times.
- Data format configuration:
  The word length, the frame length, and the shift direction have to be set up according to the application requirements by programming the register SCTR. If required by the application, the data input and output signals can be inverted.
  Additionally, the parity mode has to be configured (CCR.PM).

14.3.3.1 Bit Timing

In ASC mode, each bit (incl. protocol bits) is divided into time quanta in order to provide granularity in the sub-bit range to adjust the sample point to the application requirements. The number of time quanta per bit is defined by bit fields BRG.DCTQ and the length of a time quantum is given by BRG.PCTQ.

In the example given in **Figure 14-32**, one bit time is composed of 16 time quanta (BRG.DCTQ = 15). It is not recommended to program less than 4 time quanta per bit time.

Bit field PCR.SP determines the position of the sampling point for the bit value. The value of PCR.SP must not be set to a value greater than BRG.DCTQ. It is possible to sample the bit value only once per bit time or to take the average of samples. Depending on bit PCR.SMD, either the current input value is directly sampled as bit value, or a majority decision over the input values sampled at the latest three time quanta is taken into account. The standard ASC bit timing consists of 16 time quanta with sampling after 8 or 9 time quanta with majority decision.

The bit timing setup (number of time quanta and the sampling point definition) is common for the transmitter and the receiver. Due to independent bit timing blocks, the receiver and the transmitter can be in different time quanta or bit positions inside their frames. The transmission of a frame is aligned to the time quanta generation.

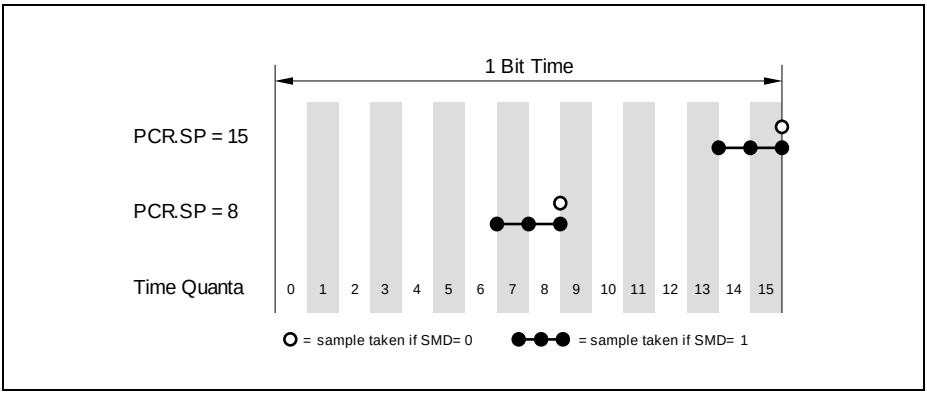


Figure 14-32 ASC Bit Timing

The sample point setting has to be adjusted carefully if collision or idle detection is enabled (via DX1 input signal), because the driver delay and some external delays have to be taken into account. The sample point for the transmit line has to be set to a value where the bit level is stable enough to be evaluated.
If the sample point is located late in the bit time, the signal itself has more time to become stable, but the robustness against differences in the clock frequency of transmitter and receiver decreases.

14.3.3.2 Baud Rate Generation

The baud rate f_{ASC} in ASC mode depends on the number of time quanta per bit time and their timing. The baud rate setting should only be changed while the transmitter and the receiver are idle. The bits in register BRG define the baud rate setting:

- BRG.CTQSEL
  to define the input frequency f_{CTQIN} for the time quanta generation

- BRG.PCTQ
  to define the length of a time quantum (division of f_{CTQIN} by 1, 2, 3, or 4)
- BRG.DCTQ
  to define the number of time quanta per bit time

The standard setting is given by CTQSEL = 00_B ($f_{CTQIN} = f_{PDIV}$) and PPPEN = 0 ($f_{PPP} = f_{PIN}$). Under these conditions, the baud rate is given by:

$$f_{ASC} = f_{PIN} \times \frac{1}{PDIV + 1} \times \frac{1}{PCTQ + 1} \times \frac{1}{DCTQ + 1} \quad (14.6)$$

In order to generate slower frequencies, two additional divide-by-2 stages can be selected by CTQSEL = 10_B ($f_{CTQIN} = f_{SCLK}$) and PPPEN = 1 ($f_{PPP} = f_{MCLK}$), leading to:

$$f_{ASC} = \frac{f_{PIN}}{2 \times 2} \times \frac{1}{PDIV + 1} \times \frac{1}{PCTQ + 1} \times \frac{1}{DCTQ + 1} \quad (14.7)$$

14.3.3.3 Noise Detection

The ASC receiver permanently checks the data input line of the DX0 stage for noise (the check is independent from the setting of bit PCR.SMD). Bit PSR.RNS (receiver noise) becomes set if the three input samples of the majority decision are not identical at the sample point for the bit value. The information about receiver noise gets accumulated over several bits in bit PSR.RNS (it has to be cleared by software) and can trigger a protocol interrupt each time noise is detected if enabled by PCR.RNIEN.

14.3.3.4 Collision Detection

In some applications, such as data transfer over a single data line shared by several sending devices (see **Figure 14-30**), several transmitters have the possibility to send on the same data output line TXD. In order to avoid collisions of transmitters being active at the same time or to allow a kind of arbitration, a collision detection has been implemented.

The data value read at the TXD input at the DX1 stage and the transmitted data bit value are compared after the sampling of each bit value. If enabled by PCR.CDEN = 1 and a bit sent is not equal to the bit read back, a collision is detected and bit PSR.COL is set. If enabled, bit PSR.COL = 1 disables the transmitter (the data output lines become 1) and generates a protocol interrupt. The content of the transmit shift register is considered as invalid, so the transmit buffer has to be programmed again.

14.3.3.5 Pulse Shaping

For some applications, the 0 level of transmitted bits with the bit value 0 is not applied at the transmit output during the complete bit time. Instead of driving the original 0 level,

only a 0 pulse is generated and the remaining time quanta of the bit time are driven with 1 level. The length of a bit time is not changed by the pulse shaping, only the signalling is changed.

In the standard ASC signalling scheme, the 0 level is signalled during the complete bit time with bit value 0 (ensured by programming PCR.PL = 000_B). In the case PCR.PL > 000_B, the transmit output signal becomes 0 for the number of time quanta defined by PCR.PL. In order to support correct reception with pulse shaping by the transmitter, the sample point has to be adjusted in the receiver according to the applied pulse length.

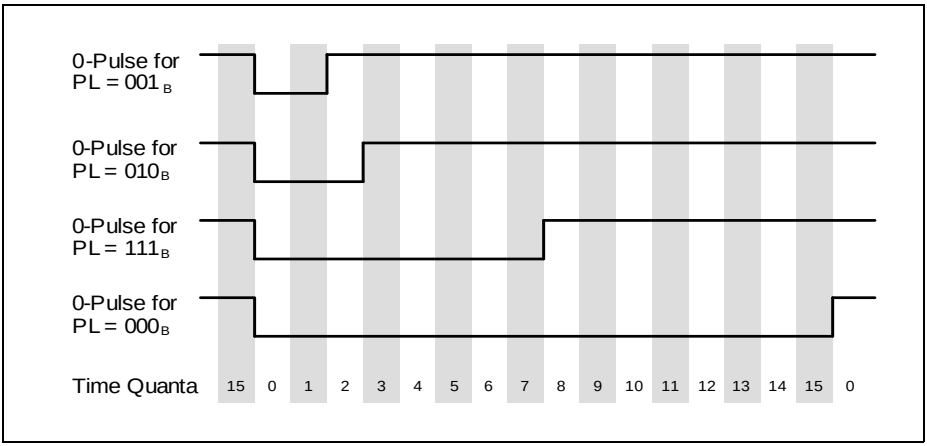


Figure 14-33 Transmitter Pulse Length Control

Figure 14-34 shows an example for the transmission of an 8-bit data word with LSB first and one stop bit (e.g. like for IrDA). The polarity of the transmit output signal has been inverted by SCTR.DOCFG = 01_B.

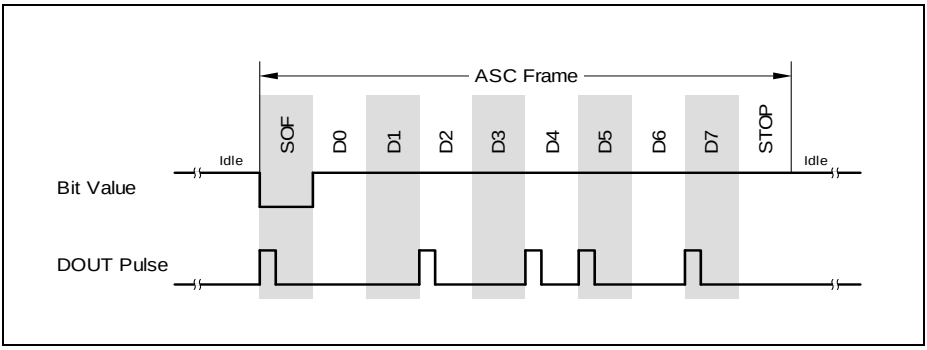


Figure 14-34 Pulse Output Example

14.3.3.6 Automatic Shadow Mechanism

The contents of the protocol control register PCR, as well as bit field SCTR.FLE are internally kept constant while a data frame is transferred by an automatic shadow mechanism (shadowing takes place with each frame start). The registers can be programmed all the time with new settings that are taken into account for the next data frame. During a data frame, the applied (shadowed) setting is not changed, although new values have been written after the start of the data frame.
Bit fields SCTR.WLE and SCTR.SDIR are shadowed automatically with the start of each data word. As a result, a data frame can consist of data words with a different length. It is recommended to change SCTR.SDIR only when no data frame is running to avoid interference between hardware and software.

Please note that the starting point of a data word can be different for a transmitter and a receiver. In order to ensure correct handling, it is recommended to modify SCTR.WLE only while transmitter and receiver are both idle. If the transmitter and the receiver are referring to the same data signal (e.g. in a LIN bus system), SCTR.WLE can be modified while a data transfer is in progress after the RSI event has been detected.

14.3.3.7 End of Frame Control

The number of bits per ASC frame is defined by bit field SCTR.FLE. In order to support different frame length settings for consecutively transmitted frames, this bit field can be modified by hardware. The automatic update mechanism is enabled by TCSR.FLEMD = 1 (in this case, bits TCSR.WLEMD, SELMD, WAMD and HPCMD have to be written with 0).

If enabled, the transmit control information TCI automatically overwrites the bit field TCSR.FLEMD when the ASC frame is started (leading to frames with 1 to 32 data bits). The TCI value represents the written address location of TBUFxx (without additional data

buffer) or INxx (with additional data buffer). With this mechanism, an ASC with 8 data bits is generated by writing a data word to TBUF07 (IN07, respectively).

14.3.3.8 Mode Control Behavior

In ASC mode, the following kernel modes are supported:

- Run Mode 0/1:
  Behavior as programmed, no impact on data transfers.
- Stop Mode 0:
  Bit PSR.TXIDLE is cleared. A new transmission is not started. A current transmission is finished normally. Bit PSR.RXIDLE is not modified. Reception is still possible.
  When leaving stop mode 0, bit TXIDLE is set according to PCR.IDM.
- Stop Mode 1:
  Bit PSR.TXIDLE is cleared. A new transmission is not started. A current transmission is finished normally. Bit PSR.RXIDLE is cleared. A new reception is not possible. A current reception is finished normally.
  When leaving stop mode 1, bits TXIDLE and RXIDLE are set according to PCR.IDM.

14.3.3.9 Disabling ASC Mode

In order to switch off ASC mode without any data corruption, the receiver and the transmitter have to be both idle. This is ensured by requesting Stop Mode 1 in register KSCFG. After waiting for the end of the frame, the ASC mode can be disabled.

14.3.3.10 Protocol Interrupt Events

The following protocol-related events are generated in ASC mode and can lead to a protocol interrupt. The collision detection and the transmitter frame finished events are related to the transmitter, whereas the receiver events are given by the synchronization break detection, the receiver noise detection, the format error checks and the end of the received frame.

Please note that the bits in register PSR are not automatically cleared by hardware and have to be cleared by software in order to monitor new incoming events.

- Collision detection:
  This interrupt indicates that the transmitted value (DOUT0) does not match with the input value of the DX1 input stage at the sample point of a bit. For more details refer to **Page 14-60**.
- Transmitter frame finished:
  This interrupt indicates that the transmitter has completely finished a frame. Bit PSR.TFF becomes set at the end of the last stop bit. The DOUT0 signal assignment to port pins can be changed while no transmission is in progress.
- Receiver frame finished:
  This interrupt indicates that the receiver has completely finished a frame. Bit

PSR.RFF becomes set at the end of the last stop bit. The DIN0 signal assignment to port pins can be changed while no reception is in progress.

- Synchronization break detection:
  This interrupt can be used in LIN networks to indicate the reception of the synchronization break symbol (at the beginning of a LIN frame).
- Receiver noise detection:
  This interrupt indicates that the input value at the sample point of a bit and at the two time quanta before are not identical.
- Format error:
  The bit value of the stop bit(s) is defined as 1 level for the ASC protocol. A format error is signalled if the sampled bit value of a stop bit is 0.

14.3.3.11 Data Transfer Interrupt Handling

The data transfer interrupts indicate events related to ASC frame handling.

- Transmit buffer interrupt TBI:
  Bit PSR.TBIF is set after the start of first data bit of a data word. This is the earliest point in time when a new data word can be written to TBUF.
  With this event, bit TCSR.TDV is cleared and new data can be loaded to the transmit buffer.
- Transmit shift interrupt TSI:
  Bit PSR.TSIF is set after the start of the last data bit of a data word.
- Receiver start interrupt RSI:
  Bit PSR.RSIF is set after the sample point of the first data bit of a data word.
- Receiver interrupt RI and alternative interrupt AI:
  Bit PSR.RIF is set after the sampling point of the last data bit of a data word if this data word is not directly followed by a parity bit (parity generation disabled or not the last word of a data frame).
  If the data word is directly followed by a parity bit (last data word of a data frame and parity generation enabled), bit PSR.RIF is set after the sampling point of the parity bit if no parity error has been detected. If a parity error has been detected, bit PSR.AIF is set instead of bit PSR.RIF.
  The first data word of a data frame is indicated by RBUFSR.SOF = 1 for the received word.
  Bit PSR.RIF is set for a receiver interrupt RI with WA = 0. Bit PSR.AIF is set for a alternative interrupt AI with WA = 1.

14.3.3.12 Baud Rate Generator Interrupt Handling

The baud rate generator interrupt indicate that the capture mode timer has reached its maximum value. With this event, the bit PSR.BRGIF is set.

14.3.3.13 Protocol-Related Argument and Error

The protocol-related argument (RBUFSR.PAR) and the protocol-related error (RBUFSR.PERR) are two flags that are assigned to each received data word in the corresponding receiver buffer status registers.

In ASC mode, the received parity bit is monitored by the protocol-related argument and the result of the parity check by the protocol-related error indication (0 = received parity bit equal to calculated parity value). This information being elaborated only for the last received data word of each data frame, both bit positions are 0 for data words that are not the last data word of a data frame or if the parity generation is disabled.

14.3.3.14 Receive Buffer Handling

If a receive FIFO buffer is available (CCFG.RB = 1) and enabled for data handling (RBCTR.SIZE > 0), it is recommended to set RBCTR.RCIM = 11_B in ASC mode. This leads to an indication that the data word has been the first data word of a new data frame if bit OUTR.RCI[0] = 1, a parity error is indicated by OUTR.RCI[4] = 1, and the received parity bit value is given by OUTR.RCI[3].

The standard receive buffer event and the alternative receive buffer event can be used for the following operations in RCI mode (RBCTR.RNM = 1):

- A standard receive buffer event indicates that a data word can be read from OUTR that has been received without parity error.
- An alternative receive buffer event indicates that a data word can be read from OUTR that has been received with parity error.

14.3.3.15 Sync-Break Detection

The receiver permanently checks the DIN0 signal for a certain number of consecutive bit times with 0 level. The number is given by the number of programmed bits per frame (SCTR.FLE) plus 2 (in the case without parity) or plus 3 (in the case with parity). If a 0 level is detected at a sample point of a bit after this event has been found, bit PSR.SBD is set and additionally, a protocol interrupt can be generated (if enabled by PCR.SBIEN = 1). The counting restarts from 0 each time a falling edge is found at input DIN0. This feature can be used for the detection of a synchronization break for slave devices in a LIN bus system (the master does not check for sync break).
For example, in a configuration for 8 data bits without parity generation, bit PCR.SBD is set after at the next sample point at 0 level after 10 complete bit times have elapsed (representing the sample point of the 11th bit time since the first falling edge).

14.3.3.16 Transfer Status Indication

The receiver status can be monitored by flag PSR[9] = BUSY if bit PCR.CTR[16] (receiver status enable RSTEN) is set. In this case, bit BUSY is set during a complete frame reception from the beginning of the start of frame bit to the end of the last stop bit.

The transmitter status can be monitored by flag PSR[9] = BUSY if bit PCR.CTR[17] (transmitter status enable TSTEN) is set. In this case, bit BUSY is set during a complete frame reception from the beginning of the start of frame bit to the end of the last stop bit. If both bits RSTEN and TSTEN are set, flag BUSY indicates the logical OR-combination of the receiver and the transmitter status. If both bits are cleared, flag BUSY is not modified depending on the transfer status (status changes are ignored).

14.3.4 ASC Protocol Registers

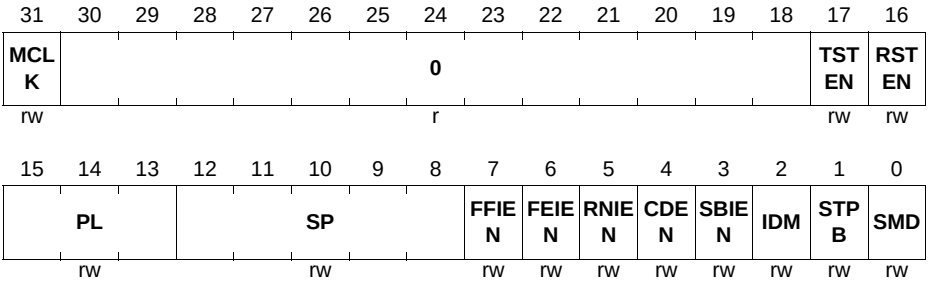
In ASC mode, the registers PCR and PSR handle ASC related information.

14.3.4.1 ASC Protocol Control Register

In ASC mode, the PCR register bits or bit fields are defined as described in this section.

PCR
Protocol Control Register [ASC Mode]

($3C_H$) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
SMD	0	rw	**Sample Mode** This bit field defines the sample mode of the ASC receiver. The selected data input signal can be sampled only once per bit time or three times (in consecutive time quanta). When sampling three times, the bit value shifted in the receiver shift register is given by a majority decision among the three sampled values. 0_B Only one sample is taken per bit time. The current input value is sampled. 1_B Three samples are taken per bit time and a majority decision is made.

Field	Bits	Type	Description
STPB	1	rw	**Stop Bits** This bit defines the number of stop bits in an ASC frame. 0_B The number of stop bits is 1. 1_B The number of stop bits is 2.
IDM	2	rw	**Idle Detection Mode** This bit defines if the idle detection is switched off or based on the frame length. 0_B The bus idle detection is switched off and bits PSR.TXIDLE and PSR.RXIDLE are set automatically to enable data transfers without checking the inputs before. 1_B The bus is considered as idle after a number of consecutive passive bit times defined by SCTR.FLE plus 2 (in the case without parity bit) or plus 3 (in the case with parity bit).
SBIEN	3	rw	**Synchronization Break Interrupt Enable** This bit enables the generation of a protocol interrupt if a synchronization break is detected. The automatic detection is always active, so bit SBD can be set independently of SBIEN. 0_B The interrupt generation is disabled. 1_B The interrupt generation is enabled.
CDEN	4	rw	**Collision Detection Enable** This bit enables the reaction of a transmitter to the collision detection. 0_B The collision detection is disabled. 1_B If a collision is detected, the transmitter stops its data transmission, outputs a 1, sets bit PSR.COL and generates a protocol interrupt. In order to allow data transmission again, PSR.COL has to be cleared by software.
RNIEN	5	rw	**Receiver Noise Detection Interrupt Enable** This bit enables the generation of a protocol interrupt if receiver noise is detected. The automatic detection is always active, so bit PSR.RNS can be set independently of PCR.RNIEN. 0_B The interrupt generation is disabled. 1_B The interrupt generation is enabled.

Field	Bits	Type	Description
FEIEN	6	rw	**Format Error Interrupt Enable** This bit enables the generation of a protocol interrupt if a format error is detected. The automatic detection is always active, so bits PSR.FER0/FER1 can be set independently of PCR.FEIEN. 0_B The interrupt generation is disabled. 1_B The interrupt generation is enabled.
FFIEN	7	rw	**Frame Finished Interrupt Enable** This bit enables the generation of a protocol interrupt if the receiver or the transmitter reach the end of a frame. The automatic detection is always active, so bits PSR.RFF or PSR.TFF can be set independently of PCR.FFIEN. 0_B The interrupt generation is disabled. 1_B The interrupt generation is enabled.
SP	[12:8]	rw	**Sample Point** This bit field defines the sample point of the bit value. The sample point must not be located outside the programmed bit timing (PCR.SP ≤ BRG.DCTQ).
PL	[15:13]	rw	**Pulse Length** This bit field defines the length of a 0 data bit, counted in time quanta, starting with the time quantum 0 of each bit time. Each bit value that is a 0 can lead to a 0 pulse that is shorter than a bit time, e.g. for IrDA applications. The length of a bit time is not changed by PL, only the length of the 0 at the output signal. The pulse length must not be longer than the programmed bit timing (PCR.PL ≤ BRG.DCTQ). This bit field is only taken into account by the transmitter and is ignored by the receiver. 000_B The pulse length is equal to the bit length (no shortened 0). 001_B The pulse length of a 0 bit is 2 time quanta. 010_B The pulse length of a 0 bit is 3 time quanta. ... 111_B The pulse length of a 0 bit is 8 time quanta.

Field	Bits	Type	Description
RSTEN	16	rw	**Receiver Status Enable** This bit enables the modification of flag PSR[9] = BUSY according to the receiver status. 0_B Flag PSR[9] is not modified depending on the receiver status. 1_B Flag PSR[9] is set during the complete reception of a frame.
TSTEN	17	rw	**Transmitter Status Enable** This bit enables the modification of flag PSR[9] = BUSY according to the transmitter status. 0_B Flag PSR[9] is not modified depending on the transmitter status. 1_B Flag PSR[9] is set during the complete transmission of a frame.
MCLK	31	rw	**Master Clock Enable** This bit enables the generation of the master clock MCLK. 0_B The MCLK generation is disabled and the MCLK signal is 0. 1_B The MCLK generation is enabled.
0	[30:18]	r	**Reserved** Returns 0 if read; should be written with 0.

14.3.4.2 ASC Protocol Status Register

In ASC mode, the PSR register bits or bit fields are defined as described in this section. The bits and bit fields in register PSR are not cleared by hardware.

The flags in the PSR register can be cleared by writing a 1 to the corresponding bit position in register PSCR. Writing a 1 to a bit position in PSR sets the corresponding flag, but does not lead to further actions (no interrupt generation). Writing a 0 has no effect. The PSR flags should be cleared by software before enabling a new protocol.

PSR
Protocol Status Register [ASC Mode] (48$_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															BRGIF
r															rwh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AIF	RIF	TBIF	TSIF	DLIF	RSIF	BUSY	TFF	RFF	FER1	FER0	RNS	COL	SBD	RXIDLE	TXIDLE
rwh	rwh	rwh	rwh	rwh	rwh	r	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
TXIDLE	0	rwh	**Transmission Idle** This bit shows if the transmit line (DX1) has been idle. A frame transmission can only be started if TXIDLE is set. 0_B The transmitter line has not yet been idle. 1_B The transmitter line has been idle and frame transmission is possible.
RXIDLE	1	rwh	**Reception Idle** This bit shows if the receive line (DX0) has been idle. A frame reception can only be started if RXIDLE is set. 0_B The receiver line has not yet been idle. 1_B The receiver line has been idle and frame reception is possible.
SBD	2	rwh	**Synchronization Break Detected**[1)] This bit is set if a programmed number of consecutive bit values with level 0 has been detected (called synchronization break, e.g. in a LIN bus system). 0_B A synchronization break has not yet been detected. 1_B A synchronization break has been detected.
COL	3	rwh	**Collision Detected**[1)] This bit is set if a collision has been detected (with PCR.CDEN = 1). 0_B A collision has not yet been detected and frame transmission is possible. 1_B A collision has been detected and frame transmission is not possible.

Field	Bits	Type	Description
RNS	4	rwh	**Receiver Noise Detected[1)]** This bit is set if receiver noise has been detected. 0_B Receiver noise has not been detected. 1_B Receiver noise has been detected.
FER0	5	rwh	**Format Error in Stop Bit 0[1)]** This bit is set if a 0 has been sampled in the stop bit 0 (called format error 0). 0_B A format error 0 has not been detected. 1_B A format error 0 has been detected.
FER1	6	rwh	**Format Error in Stop Bit 1[1)]** This bit is set if a 0 has been sampled in the stop bit 1 (called format error 1). 0_B A format error 1 has not been detected. 1_B A format error 1 has been detected.
RFF	7	rwh	**Receive Frame Finished[1)]** This bit is set if the receiver has finished the last stop bit. 0_B The received frame is not yet finished. 1_B The received frame is finished.
TFF	8	rwh	**Transmitter Frame Finished[1)]** This bit is set if the transmitter has finished the last stop bit. 0_B The transmitter frame is not yet finished. 1_B The transmitter frame is finished.
BUSY	9	r	**Transfer Status BUSY** This bit indicates the receiver status (if PCR.RSTEN = 1) or the transmitter status (if PCR.TSTEN = 1) or the logical OR combination of both (if PCR.RSTEN = PCR.TSTEN = 1). 0_B A data transfer does not take place. 1_B A data transfer currently takes place.
RSIF	10	rwh	**Receiver Start Indication Flag** 0_B A receiver start event has not occurred. 1_B A receiver start event has occurred.
DLIF	11	rwh	**Data Lost Indication Flag** 0_B A data lost event has not occurred. 1_B A data lost event has occurred.
TSIF	12	rwh	**Transmit Shift Indication Flag** 0_B A transmit shift event has not occurred. 1_B A transmit shift event has occurred.

Field	Bits	Type	Description
TBIF	13	rwh	**Transmit Buffer Indication Flag** 0_B A transmit buffer event has not occurred. 1_B A transmit buffer event has occurred.
RIF	14	rwh	**Receive Indication Flag** 0_B A receive event has not occurred. 1_B A receive event has occurred.
AIF	15	rwh	**Alternative Receive Indication Flag** 0_B An alternative receive event has not occurred. 1_B An alternative receive event has occurred.
BRGIF	16	rwh	**Baud Rate Generator Indication Flag** 0_B A baud rate generator event has not occurred. 1_B A baud rate generator event has occurred.
0	[31:17]	r	**Reserved** Returns 0 if read; should be written with 0.

1) This status bit can generate a protocol interrupt (see **Page 14-22**). The general interrupt status flags are described in the general interrupt chapter.

14.3.5 Hardware LIN Support

In order to support the LIN protocol, bit TCSR.FLEMD = 1 should be set for the master. For slave devices, it can be cleared and the fixed number of 8 data bits has to be set (SCTR.FLE = 7_H). For both, master and slave devices, the parity generation has to be switched off (CCR.PM = 00_B) and transfers take place with LSB first (SCTR.SDIR = 0) and 1 stop bit (PCR.STPB = 0).

The Local Interconnect Network (LIN) data exchange protocol contains several symbols that can all be handled in ASC mode. Each single LIN symbol represents a complete ASC frame. The LIN bus is a master-slave bus system with a single master and multiple slaves (for the exact definition please refer to the official LIN specification).

A complete LIN frame contains the following symbols:

- Synchronization break:
  The master sends a synchronization break to signal the beginning of a new frame. It contains at least 13 consecutive bit times at 0 level, followed by at least one bit time at 1 level (corresponding to 1 stop bit). Therefore, TBUF11 if the transmit buffer is used, (or IN11 if the FIFO buffer is used) has to be written with 0 (leading to a frame with SOF followed by 12 data bits at 0 level).
  A slave device shall detect 11 consecutive bit times at 0 level, which done by the synchronization break detection. Bit PSR.SBD is set if such an event is detected and a protocol interrupt can be generated. Additionally, the received data value of 0 appears in the receive buffer and a format error is signaled.

If the baud rate of the slave has to be adapted to the master, the baud rate measurement has to be enabled for falling edges by setting BRG.TMEN = 1, DX0CR.CM = 10_H and DX1CR.CM = 00_H before the next symbol starts.

- Synchronization byte:
  The master sends this symbol after writing the data value 55_H to TBUF07 (or IN07). A slave device can either receive this symbol without any further action (and can discard it) or it can use the falling edges for baud rate measurement. Bit PSR.TSIF = 1 (with optionally the corresponding interrupt) indicates the detection of a falling edge and the capturing of the elapsed time since the last falling edge in CMTR.CTV. Valid captured values can be read out after the second, third, fourth and fifth activation of TSIF. After the fifth activation of TSIF within this symbol, the baud rate detection can be disabled (BRG.TMEN = 0) and BRG.PDIV can be programmed with the captured CMTR.CTV value divided by twice the number of time quanta per bit (assuming BRG.PCTQ = 00_B).
- Other symbols:
  The other symbols of a LIN frame can be handled with ASC data frames without specific actions.

If LIN frames should be sent out on a frame base by the LIN master, the input DX2 can be connected to external timers to trigger the transmit actions (e.g. the synchronization break symbol has been prepared but is started if a trigger occurs). Please note that during the baud rate measurement of the ASC receiver, the ASC transmitter of the same USIC channel can still perform a transmission.

14.4 Synchronous Serial Channel (SSC)

The synchronous serial channel SSC covers the data transfer function of an SPI-like module. It can handle reception and transmission of synchronous data frames between a device operating in master mode and at least one device in slave mode. Besides the standard SSC protocol consisting of one input and one output data line, SSC protocols with two (Dual-SSC) or four (Quad-SSC) input/output data lines are also supported. The SSC mode is selected by CCR.MODE = 0001_B with CCFG.SSC = 1 (SSC mode is available).

14.4.1 Signal Description

A synchronous SSC data transfer is characterized by a simultaneous transfer of a shift clock signal together with the transmit and/or receive data signal(s) to determine when the data is valid (definition of transmit and sample point).

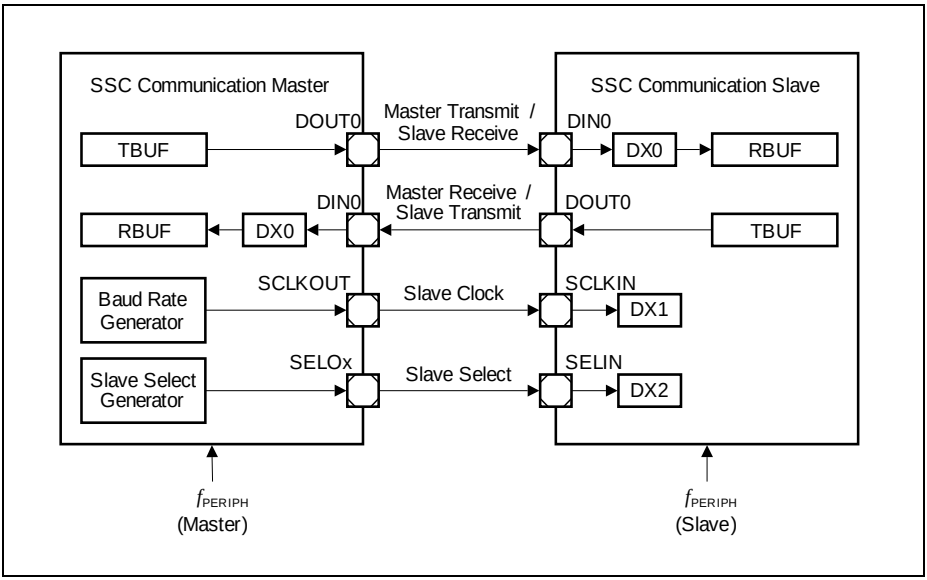


Figure 14-35 SSC Signals for Standard Full-Duplex Communication

In order to explicitly indicate the start and the end of a data transfer and to address more than one slave devices individually, the SSC module supports the handling of slave select signals. They are optional and are not necessarily needed for SSC data transfers. The SSC module supports up to 8 different slave select output signals for master mode operation (named SELOx, with x = 0-7) and 1 slave select input SELIN for slave mode. In most applications, the slave select signals are active low.

A device operating in master mode controls the start and end of a data frame, as well as the generation of the shift clock and slave select signals. This comprises the baud rate setting for the shift clock and the delays between the shift clock and the slave select output signals. If several SSC modules are connected together, there can be only one SSC master at a time, but several slaves. Slave devices receive the shift clock and optionally a slave select signal(s). For the programming of the input stages DXn please refer to **Page 14-22**.

Table 14-12 SSC Communication Signals

SSC Mode	Receive Data	Transmit Data	Shift Clock	Slave Select(s)
Standard SSC Master	MRST[1], input DIN0, handled by DX0	MTSR[2], Output DOUT0	Output SCLKOUT	Output(s) SELOx
Standard SSC Slave	MTSR, input DIN0, handled by DX0	MRST, Output DOUT0	Input SCLKIN, handled by DX1	input SELIN, handled by DX2
Dual-SSC Master	MRST[1:0], input DIN[1:0], handled by DX0 and DX3	MTSR[1:0], Output DOUT[1:0]	Output SCLKOUT	Output(s) SELOx
Dual-SSC Slave	MTSR[1:0], input DIN[1:0], handled by DX0 and DX3	MRST[1:0], Output DOUT[1:0]	Input SCLKIN, handled by DX1	input SELIN, handled by DX2
Quad-SSC Master	MRST[3:0], input DIN[3:0], handled by DX0, DX3, DX4 and DX5	MTSR[3:0], Output DOUT[3:0]	Output SCLKOUT	Output(s) SELOx
Quad-SSC Slave	MTSR[3:0], input DIN[3:0], handled by DX0, DX3, DX4 and DX5	MRST[3:0], Output DOUT[3:0]	Input SCLKIN, handled by DX1	input SELIN, handled by DX2

1) MRST = master receive slave transmit, also known as MISO = master in slave out

2) MTSR = master transmit slave receive, also known as MOSI = master out slave in

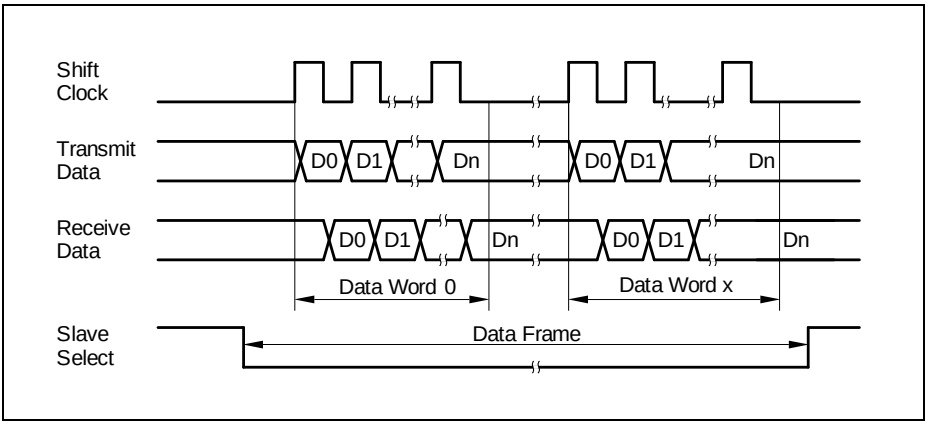


Figure 14-36 4-Wire SSC Standard Communication Signals

14.4.1.1 Transmit and Receive Data Signals

In standard SSC half-duplex mode, a single data line is used, either for data transfer from the master to a slave or from a slave to the master. In this case, MRST and MTSR are connected together, one signal as input, the other one as output, depending on the data direction. The user software has to take care about the data direction to avoid data collision (e.g. by preparing dummy data of all 1s for transmission in case of a wired AND connection with open-drain drivers, by enabling/disabling push/pull output drivers or by switching pin direction with hardware port control enabled). In full-duplex mode, data transfers take place in parallel between the master device and a slave device via two independent data signals MTSR and MRST, as shown in **Figure 14-35**.

The receive data input signal DIN0 is handled by the input stage DX0. In master mode (referring to MRST) as well as in slave mode (referring to MTSR), the data input signal DIN0 is taken from an input pin. The signal polarity of DOUT0 (data output) with respect to the data bit value can be configured in block DOCFG (data output configuration) by bit field SCTR.DOCFG.

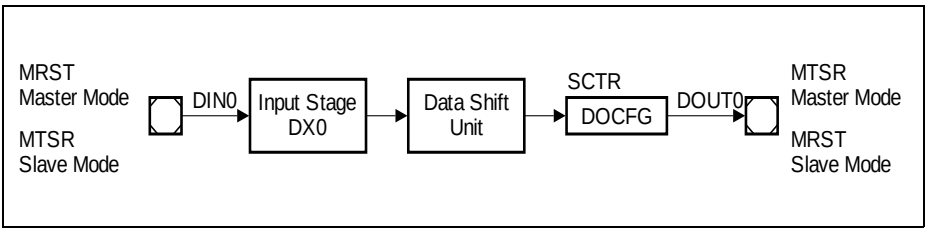


Figure 14-37 SSC Data Signals

For dual- and quad-SSC modes that require multiple input and output data lines to be used, additional input stages, DINx and DOUTx signals need to be set up.

14.4.1.2 Shift Clock Signals

The shift clock signal is handled by the input stage DX1. In slave mode, the signal SCLKIN is received from an external master, so the DX1 stage has to be connected to an input pin. The input stage can invert the received input signal to adapt to the polarity of SCLKIN to the function of the data shift unit (data transmission on rising edges, data reception on falling edges).

In master mode, the shift clock is generated by the internal baud rate generator. The output signal SCLK of the baud rate generator is taken as shift clock input for the data shift unit. The internal signal SCLK is made available for external slave devices by signal SCLKOUT. For complete closed loop delay compensation in a slave mode, SCLKOUT can also take the transmit shift clock from the input stage DX1. The selection is done through the bit BRG.SCLKOSEL. See **Section 14.4.6.3**.

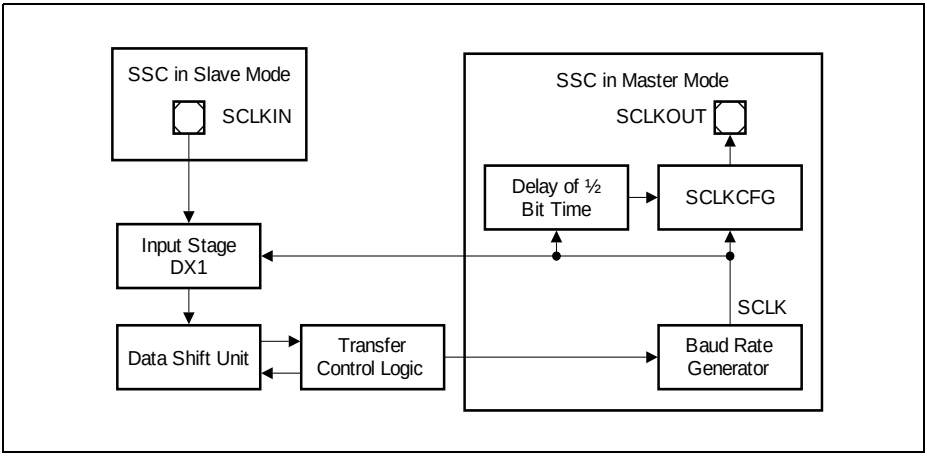


Figure 14-38 SSC Shift Clock Signals

Due to the multitude of different SSC applications, in master mode, there are different ways to configure the shift clock output signal SCLKOUT with respect to SCLK. This is done in the block SCLKCFG (shift clock configuration) by bit field BRG.SCLKCFG, allowing 4 possible settings, as shown in **Figure 14-39**.

- No delay, no polarity inversion (SCLKCFG = 00_B, SCLKOUT equals SCLK):
  The inactive level of SCLKOUT is 0, while no data frame is transferred. The first data bit of a new data frame is transmitted with the first rising edge of SCLKOUT and the first data bit is received in with the first falling edge of SCLKOUT. The last data bit of a data frame is transmitted with the last rising clock edge of SCLKOUT and the last

data bit is received in with the last falling edge of SCLKOUT. This setting can be used in master and in slave mode. It corresponds to the behavior of the internal data shift unit.

- No delay, polarity inversion (SCLKCFG = 01_B):
  The inactive level of SCLKOUT is 1, while no data frame is transferred. The first data bit of a new data frame is transmitted with the first falling clock edge of SCLKOUT and the first data bit is received with the first rising edge of SCLKOUT. The last data bit of a data frame is transmitted with the last falling edge of SCLKOUT and the last data bit is received with the last rising edge of SCLKOUT.
  This setting can be used in master and in slave mode. For slave mode, bit field DX1CR.DPOL = 1_B has to be programmed.
- SCLKOUT is delayed by 1/2 shift clock period, no polarity inversion (SCLKCFG = 10_B):
  The inactive level of SCLKOUT is 0, while no data frame is transferred.
  The first data bit of a new data frame is transmitted 1/2 shift clock period before the first rising clock edge of SCLKOUT. Due to the delay, the next data bits seem to be transmitted with the falling edges of SCLKOUT. The last data bit of a data frame is transmitted 1/2 period of SCLKOUT before the last rising clock edge of SCLKOUT. The first data bit is received 1/2 shift clock period before the first falling edge of SCLKOUT. Due to the delay, the next data bits seem to be received with the rising edges of SCLKOUT. The last data bit is received 1/2 period of SCLKOUT before the last falling clock edge of SCLKOUT.
  This setting can be used only in master mode and not in slave mode (the connected slave has to provide the first data bit before the first SCLKOUT edge, e.g. as soon as it is addressed by its slave select).
- SCLKOUT is delayed by 1/2 shift clock period, polarity inversion (SCLKCFG = 11_B):
  The inactive level of SCLKOUT is 1, while no data frame is transferred.
  The first data bit of a new data frame is transmitted 1/2 shift clock period before the first falling clock edge of SCLKOUT. Due to the delay, the next data bits seem to be transmitted with the rising edges of SCLKOUT. The last data bit of a data frame is transmitted 1/2 period of SCLKOUT before the last falling clock edge of SCLKOUT. The first data bit is received 1/2 shift clock period before the first rising edge of SCLKOUT. Due to the delay, the next data bits seem to be received with the falling edges of SCLKOUT. The last data bit is received 1/2 period of SCLKOUT before the last rising clock edge of SCLKOUT.
  This setting can be used only in master mode and not in slave mode (the connected slave has to provide the first data bit before the first SCLKOUT edge, e.g. as soon as it is addressed by its slave select).

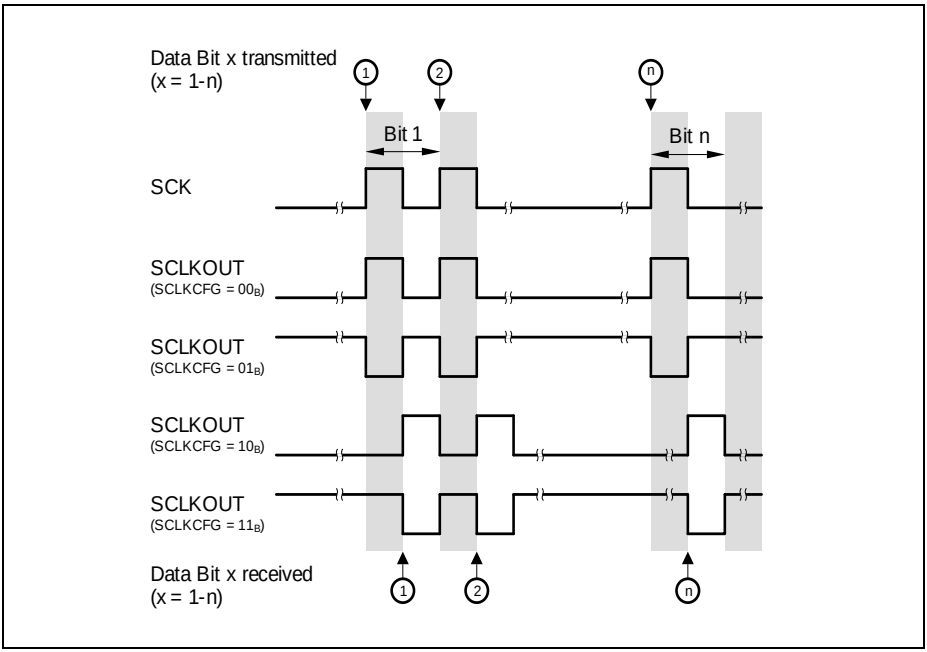


Figure 14-39 SCLKOUT Configuration in SSC Master Mode

Note: If a configuration with delay is selected and a slave select line is used, the slave select delays have to be set up accordingly.

In SSC slave mode, the bit PCR.SLPHSEL can be used to configure the clock phase of the data shift.

- When SLPHSEL = 0_B, the slave SSC transmits data bits with each leading edge of the selected shift clock input (SCLKIN) and receives data bits with each trailing edge of SCLKIN
- When SLPHSEL = 1_B, the slave SSC transmits the first data bit once the selected slave select input (SELIN) becomes active. If SELIN is not used, the DX2 stage has to deliver a 1-level to the data shift unit to shift out the first bit. Subsequent data bits are then transmitted with each trailing edge of SCLKIN. The SSC slave receives all data bits with each leading edge of SCLKIN.

For both settings, the clock polarity is determined by bit 0 of SCLKCFG.

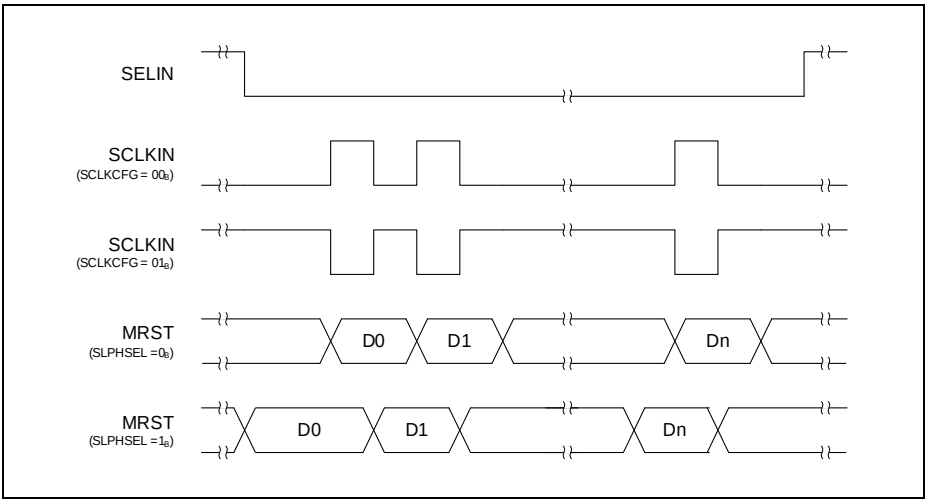


Figure 14-40 SLPHSEL Configuration in SSC Slave Mode

14.4.1.3 Slave Select Signals

The slave select signal is handled by the input stage DX2. In slave mode, the input signal SELIN is received from an external master via an input pin. The input stage can invert the received input signal to adapt the polarity of signal SELIN to the function of the data shift unit (the module internal signals are considered as high active, so a data transfer is only possible while the slave select input of the data shift unit is at 1-level, otherwise, shift clock pulses are ignored and do not lead to data transfers). If an input signal SELIN is low active, it should be inverted in the DX2 input stage.

In master mode, a master slave select signal MSLS is generated by the internal slave select generator. In order to address different external slave devices independently, the internal MSLS signal is made available externally via up to 8 SELOx output signals that can be configured by the block SELCFG (select configuration).

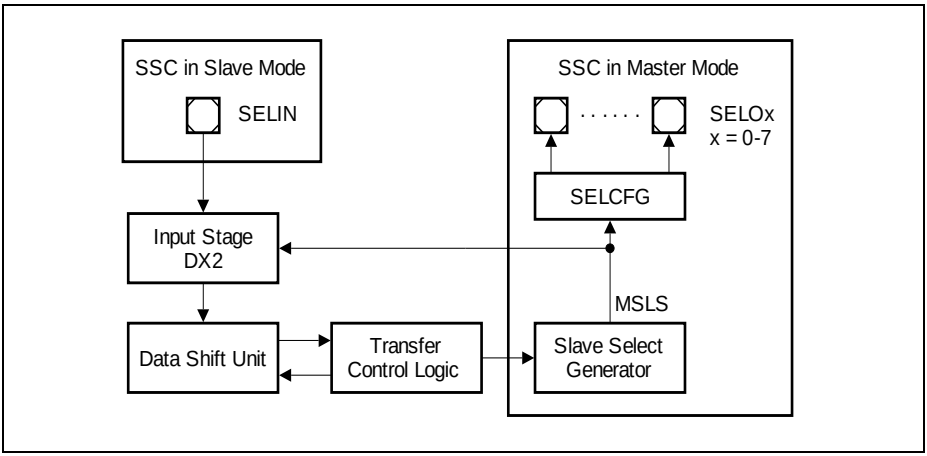


Figure 14-41 SSC Slave Select Signals

The control of the SELCFG block is based on protocol specific bits and bit fields in the protocol control register PCR. For the generation of the MSLS signal please refer to **Section 14.4.3.2**.

- PCR.SELCTR to chose between direct and coded select mode
- PCR.SELINV to invert the SELOx outputs
- PCR.SELO[7:0] as individual value for each SELOx line

The SELCFG block supports the following configurations of the SELOx output signals:

- Direct Select Mode (SELCTR = 1):
  Each SELOx line (with x = 0-7) can be directly connected to an external slave device. If bit x in bit field SELO is 0, the SELOx output is permanently inactive. A SELOx output becomes active while the internal signal MSLS is active (see **Section 14.4.3.2**) and bit x in bit field SELO is 1. Several external slave devices can be addressed in parallel if more than one bit in bit field SELO are set during a data frame. The number of external slave devices that can be addressed individually is limited to the number of available SELOx outputs.
- Coded Select Mode (SELCTR = 0):
  The SELOx lines (with x = 1-7) can be used as addresses for an external address decoder to increase the number of external slave devices. These lines only change with the start of a new data frame and have no other relation to MSLS. Signal SELO0 can be used as enable signal for the external address decoder. It is active while MSLS is active (during a data frame) and bit 0 in bit field SELO is 1. Furthermore, in coded select mode, this output line is delayed by one cycle of f_{PERIPH} compared to MSLS to allow the other SELOx lines to stabilize before enabling the address decoder.

14.4.2 Operating the SSC

This chapter contains SSC issues, that are of general interest and not directly linked to either master mode or slave mode.

14.4.2.1 Automatic Shadow Mechanism

The contents of the baud rate control register BRG, bit fields SCTR.FLE as well as the protocol control register PCR are internally kept constant while a data frame is transferred (= while MSLS is active) by an automatic shadow mechanism. The registers can be programmed all the time with new settings that are taken into account for the next data frame. During a data frame, the applied (shadowed) setting is not changed, although new values have been written after the start of the data frame.

Bit fields SCTR.WLE, SCTR.DSM, SCTR.HPCDIR and SCTR.SDIR are shadowed automatically with the start of each data word. As a result, a data frame can consist of data words with a different length, or data words that are transmitted or received through different number of data lines. It is recommended to change SCTR.SDIR only when no data frame is running to avoid interference between hardware and software.

Please note that the starting point of a data word are different for a transmitter (first bit transmitted) and a receiver (first bit received). In order to ensure correct handling, it is recommended to refer to the receive start interrupt RSI before modifying SCTR.WLE. If TCSR.WLEMD = 1, it is recommended to update TCSR and TBUFxx after the receiver start interrupt has been generated.

14.4.2.2 Mode Control Behavior

In SSC mode, the following kernel modes are supported:

- Run Mode 0/1:
  Behavior as programmed, no impact on data transfers.
- Stop Mode 0/1:
  The content of the transmit buffer is considered as not valid for transmission. Although being considered as 0, bit TCSR.TDV it is not modified by the stop mode condition.
  In master mode, a currently running word transfer is finished normally, but no new data word is started (the stop condition is not considered as end-of-frame condition).
  In slave mode, a currently running word transfer is finished normally. Passive data will be sent out instead of a valid data word if a data word transfer is started by the external master while the slave device is in stop mode. In order to avoid passive slave transmit data, it is recommended not to program stop mode for an SSC slave device if the master device does not respect the slave device's stop mode.

14.4.2.3 Disabling SSC Mode

In order to disable SSC mode without any data corruption, the receiver and the transmitter have to be both idle. This is ensured by requesting Stop Mode 1 in register KSCFG. After Stop Mode 1 has been acknowledged by KSCFG.2 = 1, the SSC mode can be disabled.

14.4.2.4 Data Frame Control

An SSC data frame can consist of several consecutive data words that may be separated by an inter-word delay. Without inter-word delay, the data words seem to form a longer data word, being equivalent to a data frame. The length of the data words are most commonly identical within a data frame, but may also differ from one word to another. The data word length information (defined by SCTR.WLE) is evaluated for each new data word, whereas the frame length information (defined by SCTR.FLE) is evaluated at the beginning at each start of a new frame.
The length of an SSC data frame can be defined in two different ways:

- By the number of bits per frame:
  If the number of bits per data frame is defined (frame length FLE), a slave select signal is not necessarily required to indicate the start and the end of a data frame.
  If the programmed number of bits per frame is reached within a data word, the frame is considered as finished and remaining data bits in the last data word are ignored and are not transferred.
  This method can be applied for data frames with up to 63 data bits.
- By the slave select signal:
  If the number of bits per data frame is not known, the start/end information of a data frame is given by a slave select signal. If a deactivation of the slave select signal is detected within a data word, the frame is considered as finished and remaining data bits in the last data word are ignored and are not transferred.
  This method has to be applied for frames with more than 63 data bits (programming limit of FLE). The advantage of slave select signals is the clearly defined start and end condition of data frames in a data stream. Furthermore, slave select signals allow to address slave devices individually.

14.4.2.5 Parity Mode

The SSC allows parity generation for transmission and parity check for reception on frame base. The type of parity can be selected by bit field CCR.PM, common for transmission and reception (no parity, even or odd parity). If the parity handling is disabled, the SSC frame does not contain any parity bit. For consistency reasons, all communication partners have to be programmed to the same parity mode.

If parity generation has been enabled, the transmitter automatically extends the clock by one cycle after the last data word of the data frame, and sends out its calculated parity bit in this cycle.

Figure 14-42 shows how a parity bit is added to the transmitted data bits of a frame. The number of the transmitted bits of a complete frame with parity is always one more than that without parity. The parity bit is transmitted as the last bit of a frame, following the data bits, independent of the shift direction (SCTR.SDIR).

Note: For dual and quad SSC protocols, the parity bit will be transmitted and received only on DOUT0 and DX0 respectively in the extended clock cycle.

Figure 14-42 Data Frames without/with Parity

Similarly, after the receiver receives the last word of a data frame as defined by FLE, it expects an additional one clock cycle, which will contain the parity bit. The receiver interprets this bit as received parity and separates it from the received data. The received parity bit value is instead monitored in the protocol-related argument (PAR) of the receiver buffer status registers as receiver buffer status information. The receiver compares the bit to its internally calculated parity and the result of the parity check is indicated by the flag PSR.PARERR. The parity error event generates a protocol interrupt if PCR.PARIEN = 1.

Parity bit generation and detection is not supported for the following cases:

- When frame length is 64 data bits or greater, i.e. FLE = 63_H;
- When in slave mode, the end of frame occurs before the number of data bits defined by FLE is reached.

14.4.2.6 Transfer Mode

In SSC mode, bit field SCTR.TRM = 01_B has to be programmed to allow data transfers. Setting SCTR.TRM = 00_B disables and stops the data transfer immediately.

14.4.2.7 Data Transfer Interrupt Handling

The data transfer interrupts indicate events related to SSC frame handling.

- Transmit buffer interrupt TBI:
  Bit PSR.TBIF is set after the start of first data bit of a data word.
- Transmit shift interrupt TSI:
  Bit PSR.TSIF is set after the start of the last data bit of a data word.
- Receiver start interrupt RSI:
  Bit PSR.RSIF is set after the reception of the first data bit of a data word.
  With this event, bit TCSR.TDV is cleared and new data can be loaded to the transmit buffer.
- Receiver interrupt RI:
  The reception of the second, third, and all subsequent words in a multi-word frame is always indicated by RBUFSR.SOF = 0. Bit PSR.RIF is set after the reception of the last data bit of a data word if RBUFSR.SOF = 0.
  Bit RBUFSR.SOF indicates whether the received data word has been the first data word of a multi-word frame or some subsequent word. In SSC mode, it decides if alternative interrupt or receive interrupt is generated.
- Alternative interrupt AI:
  The reception of the first word in a frame is always indicated by RBUFSR.SOF = 1. This is true both in case of reception of multi-word frames and single-word frames. In SSC mode, this results in setting PSR.AIF.

14.4.2.8 Baud Rate Generator Interrupt Handling

The baud rate generator interrupt indicate that the capture mode timer has reached its maximum value. With this event, the bit PSR.BRGIF is set.

14.4.2.9 Protocol-Related Argument and Error

The protocol-related argument (RBUFSR.PAR) and the protocol-related error (RBUFSR.PERR) are two flags that are assigned to each received data word in the corresponding receiver buffer status registers.

In SSC mode, the received parity bit is monitored by the protocol-related argument. The received start of frame indication is monitored by the protocol-related error indication (0 = received word is not the first word of a frame, 1 = received word is the first word of a new frame).

Note: For SSC, the parity error event indication bit is located in the PSR register.

14.4.2.10 Receive Buffer Handling

If a receive FIFO buffer is available (CCFG.RB = 1) and enabled for data handling (RBCTR.SIZE > 0), it is recommended to set RBCTR.RCIM = 01_B in SSC mode. This leads to an indication that the data word has been the first data word of a new data frame if bit OUTR.RCI[4] = 1, and the word length of the received data is given by OUTR.RCI[3:0].

The standard receive buffer event and the alternative receive buffer event can be used for the following operation in RCI mode (RBCTR.RNM = 1):

- A standard receive buffer event indicates that a data word can be read from OUTR that has not been the first word of a data frame.
- An alternative receive buffer event indicates that the first data word of a new data frame can be read from OUTR.

14.4.2.11 Multi-IO SSC Protocols

The SSC implements the following three features to support multiple data input/output SSC protocols, such as the dual- and quad-SSC:

1. Data Shift Mode (**Section 14.2.5.2**)
   Configures the data for transmission and reception using one, two or four data lines in parallel, through the bit field SCTR.DSM.
2. Hardware Port Control (**Section 14.2.7**)
   Sets up a dedicated hardware interface to control the direction of the pins overlaid with both DINx and DOUTx functions, through the bit SCTR.HPCDIR.
3. Transmit Control Information (**Section 14.2.5.3**)
   Allows the dynamic control of both the shift mode and pin direction during data transfers by writing to SCTR.DSM and SCTR.HPCDIR with TCI.

Figure 14-43 shows an example of a Quad-SSC protocol, which requires the master SSC to first transmit a command byte (to request a quad output read from the slave) and a dummy byte through a single data line. At the end of the dummy byte, both master and slave SSC switches to quad data lines, and with the roles of transmitter and receiver reversed. The master SSC then receives the data four bits per shift clock from the slave through the MRST[3:0] lines.

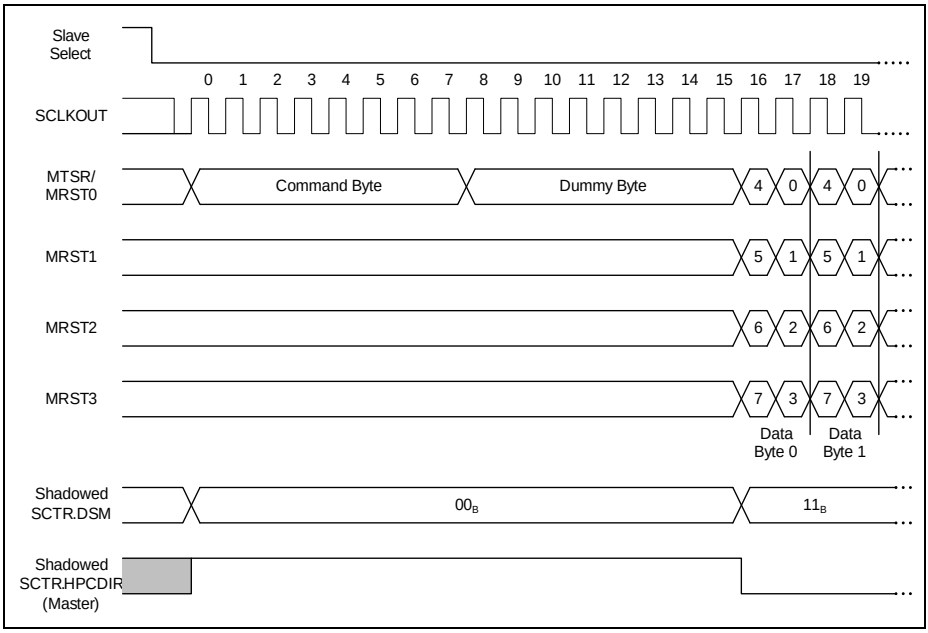


Figure 14-43 Quad-SSC Example

To work with the quad-SSC protocol in the given example, the following issues have to be additionally considered on top of those defined in **Section 14.4.3** and **Section 14.4.4**:

- During the initialization phase:
  - Set CCR.HPCEN to 11_B to enable the dedicated hardware interface to the DX0/DOUT0, DX3/DOUT1, DX4/DOUT2 and DX5/DOUT3 pins.
  - Set TCSR.[4:0] to 10_H to enable hardware port control in TCI
- To start the data transfer:
  - For the master SSC, write the command and dummy bytes into TBUF04 to select a single data line in output mode and initiate the data transfer.
  - For the slave SSC, dummy data can be preloaded into TBUF00 to select a single data line in input mode.
- To switch to quad data lines and pin direction:

– For the master SSC, write subsequent dummy data to TBUF03 to select quad data lines in input mode to read in valid slave data.
– For the slave SSC, write valid data to TBUF07 for transmission through quad data lines in output mode.

Note: If DOUT[1:0] or DOUT[3:0] pins are already enabled with CCR.HPCEN but SCTR.DSM does not use all pins, the unused DOUTx pins output the passive data level defined by SCTR.PDL.

Figure 14-44 shows the connections for the Quad-SSC example.

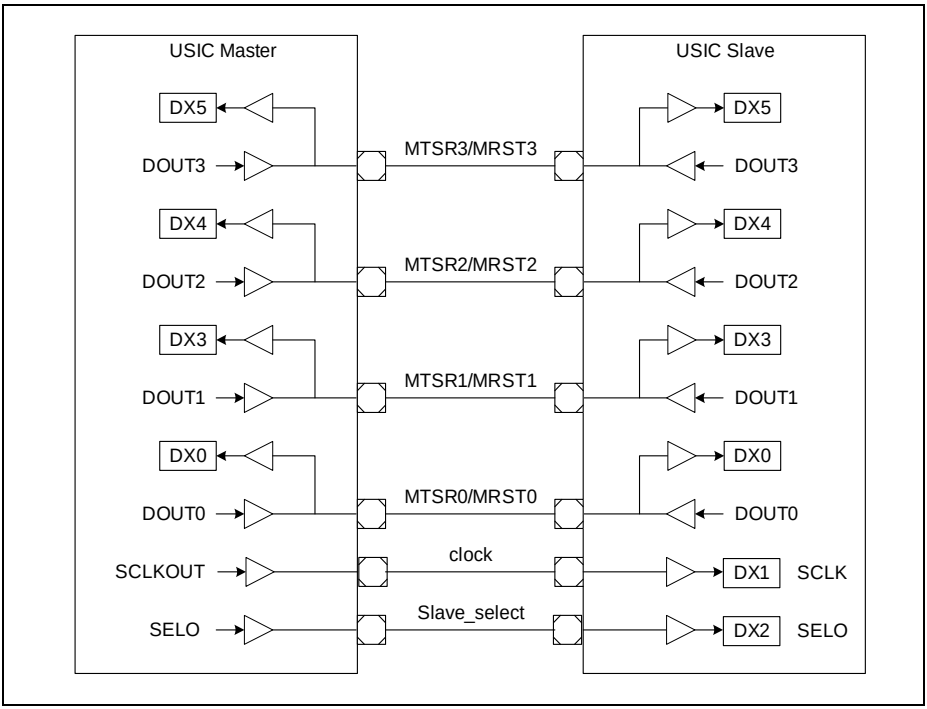


Figure 14-44 Connections for Quad-SSC Example

14.4.3 Operating the SSC in Master Mode

In order to operate the SSC in master mode, the following issues have to be considered:

- Select SSC mode:
  It is recommended to configure all parameters of the SSC that do not change during run time while CCR.MODE = 0000_B. Bit field SCTR.TRM = 01_B has to be programmed. The configuration of the input stages has to be done while

CCR.MODE = 0000_B to avoid unintended edges of the input signals and the SSC mode can be enabled by CCR.MODE = 0001_B afterwards.

- Pin connections:
  Establish a connection of the input stage (DX0, DX3, DX4, DX5) with the selected receive data input pin (DIN[3:0]) with DXnCR.INSW = 1 and configure the transmit data output pin (DOUT[3:0]). One, two or four such connections may be needed depending on the protocol. For half-duplex configurations, hardware port control can be also used to establish the required connections.
- Baud rate generation:
  The desired baud rate setting has to be selected, comprising the fractional divider and the baud rate generator. Bit DX1CR.INSW = 0 has to be programmed to use the baud rate generator output SCLK directly as input for the data shift unit. Configure a shift clock output pin (signal SCLKOUT).
- Slave select generation:
  The slave select delay generation has to be enabled by setting PCR.MSLSEN = 1 and the programming of the time quanta counter setting. Bit DX2CR.INSW = 0 has to be programmed to use the slave select generator output MSLS as input for the data shift unit. Configure slave select output pins (signals SELOx) if needed.
- Data format configuration:
  The word length, the frame length, the shift direction and shift mode have to be set up according to the application requirements by programming the register SCTR.

Note: The USIC can only receive in master mode if it is transmitting, because the master frame handling refers to bit TDV of the transmitter part.

Note: The step to enable the alternate output port functions should only be done after the SSC mode is enabled, to avoided unintended spikes on the output.

14.4.3.1 Baud Rate Generation

The baud rate (determining the length of one data bit) of the SSC is defined by the frequency of the SCLK signal (one period of f_{SCLK} represents one data bit). The SSC baud rate generation does not imply any time quanta counter.

In a standard SSC application, the phase relation between the optional MCLK output signal and SCLK is not relevant and can be disabled (BRG.PPPEN = 0). In this case, the SCLK signal directly derives from the protocol input frequency f_{PIN}. In the exceptional case that a fixed phase relation between the MCLK signal and SCLK is required (e.g. when using MCLK as clock reference for external devices), the additional divider by 2 stage has to be taken into account (BRG.PPPEN = 1).

The adjustable divider factor is defined by bit field BRG.PDIV.

$$f_{SCLK} = \frac{f_{PIN}}{2} \times \frac{1}{PDIV + 1} \quad \text{if PPPEN = 0}$$
$$f_{SCLK} = \frac{f_{PIN}}{2 \times 2} \times \frac{1}{PDIV + 1} \quad \text{if PPPEN = 1} \qquad (14.8)$$

14.4.3.2 MSLS Generation

The slave select signals indicate the start and the end of a data frame and are also used by the communication master to individually select the desired slave device. A slave select output of the communication master becomes active a programmable time before a data part of the frame is started (leading delay T_{ld}), necessary to prepare the slave device for the following communication. After the transfer of a data part of the frame, it becomes inactive again a programmable time after the end of the last bit (trailing delay T_{td}) to respect the slave hold time requirements. If data frames are transferred back-to-back one after the other, the minimum time between the deactivation of the slave select and the next activation of a slave select is programmable (next-frame delay T_{nf}). If a data frame consists of more than one data word, an optional delay between the data words can also be programmed (inter-word delay T_{iw}).

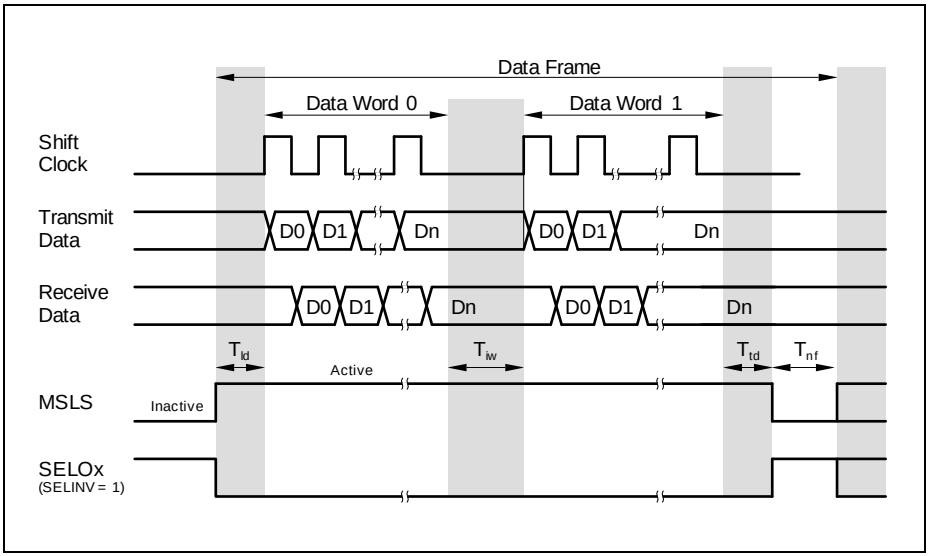


Figure 14-45 MSLS Generation in SSC Master Mode

In SSC master mode, the slave select delays are defined as follows:

- Leading delay T_{ld}:
  The leading delay starts if valid data is available for transmission. The internal signal MSLS becomes active with the start of the leading delay. The first shift clock edge (rising edge) of SCLK is generated by the baud rate generator after the leading delay has elapsed.
- Trailing delay T_{td}
  The trailing delay starts at the end of the last SCLK cycle of a data frame. The internal signal MSLS becomes inactive with the end of the trailing delay.
- Inter-word delay T_{iw}:
  This delay is optional and can be enabled/disabled by PCR.TIWEN. If the inter-word delay is disabled (TIWEN = 0), the last data bit of a data word is directly followed by the first data bit of the next data word of the same data frame. If enabled (TIWEN = 1), the inter-word delay starts at the end of the last SCLK cycle of a data word. The first SCLK cycle of the following data word of the same data frame is started when the inter-word delay has elapsed. During this time, no shift clock pulses are generated and signal MSLS stays active. The communication partner has time to “digest” the previous data word or to prepare for the next one.
- Next-frame delay T_{nf}:
  The next-frame delay starts at the end of the trailing delay. During this time, no shift clock pulses are generated and signal MSLS stays inactive. A frame is considered as finished after the next-frame delay has elapsed.

14.4.3.3 Automatic Slave Select Update

If the number of bits per SSC frame and the word length are defined by bit fields SCTR.FLE and SCTR.WLE, the transmit control information TCI can be used to update the slave select setting PCR.CTR[23:16] to control the SELOx select outputs. The automatic update mechanism is enabled by TCSR.SELMD = 1 (bits TCSR.WLEMD, FLEMD, and WAMD have to be cleared). In this case, the TCI of the first data word of a frame defines the slave select setting of the complete frame due to the automatic shadow mechanism (see **Page 14-62**).

14.4.3.4 Slave Select Delay Generation

The slave select delay generation is based on time quanta. The length of a time quantum (defined by the period of the f_{CTQIN}) and the number of time quanta per delay can be programmed.

In standard SSC applications, the leading delay T_{ld} and the trailing delay T_{td} are mainly used to ensure stability on the input and output lines as well as to respect setup and hold times of the input stages. These two delays have the same length (in most cases shorter than a bit time) and can be programmed with the same set of bit fields.

- BRG.CTQSEL
  to define the input frequency f_{CTQIN} for the time quanta generation for T_{ld} and T_{td}
- BRG.PCTQ
  to define the length of a time quantum (division of f_{CTQIN} by 1, 2, 3, or 4) for T_{ld} and T_{td}
- BRG.DCTQ
  to define the number of time quanta for the delay generation for T_{ld} and T_{td}

The inter-word delay T_{iw} and the next-frame delay T_{nf} are used to handle received data or to prepare data for the next word or frame. These two delays have the same length (in most cases in the bit time range) and can be programmed with a second, independent set of bit fields.

- PCR.CTQSEL1
  to define the input frequency f_{CTQIN} for the time quanta generation for T_{nf} and T_{iw}
- PCR.PCTQ1
  to define the length of a time quantum (division of f_{CTQIN} by 1, 2, 3, or 4) for T_{nf} and T_{iw}
- PCR.DCTQ1
  to define the number of time quanta for the delay generation for T_{nf} and T_{iw}
- PCR.TIWEN
  to enable/disable the inter-word delay T_{iw}

Each delay depends on the length of a time quantum and the programmed number of time quanta given by the bit fields CTQSEL/CTQSEL1, PCTQ/DCTQ and PCTQ1/DCTQ1 (the coding of CTQSEL1 is similar to CTQSEL, etc.). To provide a high flexibility in programming the delay length, the input frequencies can be selected between several possibilities (e.g. based on bit times or on the faster inputs of the protocol-related divider). The delay times are defined as follows:

$$T_{ld} = T_{td} = \frac{(PCTQ + 1) \times (DCTQ + 1)}{f_{CTQIN}}$$
$$T_{iw} = T_{nf} = \frac{(PCTQ1 + 1) \times (DCTQ1 + 1)}{f_{CTQIN}} \quad (14.9)$$

14.4.3.5 Protocol Interrupt Events

The following protocol-related events generated in SSC mode and can lead to a protocol interrupt. They are related to the start and the end of a data frame. After the start of a data frame a new setting could be programmed for the next data frame and after the end of a data frame the SSC connections to pins can be changed.

Please note that the bits in register PSR are not all automatically cleared by hardware and have to be cleared by software in order to monitor new incoming events.

- MSLS Interrupt:
  This interrupt indicates in master mode (MSLS generation enabled) that a data frame has started (activation of MSLS) and has been finished (deactivation of MSLS). Any change of the internal MSLS signal sets bit PSR.MSLSEV and additionally, a protocol interrupt can be generated if PCR.MSLSIEN = 1. The actual state of the internal MSLS signal can be read out at PSR.MSLS to take appropriate actions when this interrupt has been detected.
- DX2T Interrupt:
  This interrupt monitors edges of the input signal of the DX2 stage (although this signal is not used as slave select input for data transfers).
  A programmable edge detection for the DX2 input signal sets bit PSR.DX2TEV and additionally, a protocol interrupt can be generated if PCR.DX2TIEN = 1. The actual state of the selected input signal can be read out at PSR.DX2S to take appropriate actions when this interrupt has been detected.
- Parity Error Interrupt:
  This interrupt indicates that there is a mismatch in the received parity bit (in RBUFSR.PAR) with the calculated parity bit of the last received word of a data frame.

14.4.3.6 End-of-Frame Control

The information about the frame length is required for the MSLS generator of the master device. In addition to the mechanism based on the number of bits per frame (selected with SCTR.FLE < 63), the following alternative mechanisms for end of frame handling are supported. It is recommended to set SCTRFLE = 63 (if several end of frame mechanisms are activated in parallel, the first end condition being found finishes the frame).

- Software-based start of frame indication TCSR.SOF:
  This mechanism can be used if software handles the TBUF data without data FIFO. If bit SOF is set, a valid content of TBUF is considered as first word of a new frame. Bit SOF has to be set before the content of TBUF is transferred to the transmit shift register, so it is recommended to write it before writing data to TBUF. A current data word transfer is finished completely and the slave select delays T_{td} and T_{nf} are applied before starting a new data frame with T_{ld} and the content of TBUF.
  For software-handling of bit SOF, bit TCSR.WLEMD = 0 has to be programmed. In this case, all TBUF[31:0] address locations show an identical behavior (TCI not taken into account for data handling).
- Software-based end of frame indication TCSR.EOF:
  This mechanism can be used if software handles the TBUF data without data FIFO. If bit EOF is set, a valid content of TBUF is considered as last word of a new frame. Bit EOF has to be set before the content of TBUF is transferred to the transmit shift register, so it is recommended to write it before writing data to TBUF. The data word in TBUF is sent out completely and the slave select delays T_{td} and T_{nf} are applied. A new data frame can start with T_{ld} with the next valid TBUF value.
  For software-handling of bit EOF, bit TCSR.WLEMD = 0 has to be programmed. In this case, all TBUF[31:0] address locations show an identical behavior (TCI not taken into account for data handling).
- Software-based address related end of frame handling:
  This mechanism can be used if software handles the TBUF data without data FIFO. If bit TCSR.WLEMD = 1, the address of the written TBUF[31:0] is used as transmit control information TCI[4:0] to update SCTR.WLE (= TCI[3:0]) and TCSR.EOF (= TCI[4]) for each data word. The written TBUF[31:0] address location defines the word length and the end of a frame (locations TBUF[31:16] lead to a frame end).
  For example, writing transmit data to TBUF[07] results in a data word of 8-bit length without finishing the frame, whereas writing transmit data to TBUF[31] leads to a data word length of 16 bits, followed by T_{td}, the deactivation of MSLS and T_{nf}.
  If TCSR.WLEMD = 1, bits TCSR.EOF and SOF, as well as SCTR.WLE must not be written by software after writing data to a TBUF location. Furthermore, it is recommended to clear bits TCSR.SELMD, FLEMD and WAMD.
- FIFO-based address related end of frame handling:
  This mechanism can be used if a data FIFO is used to store the transmit data. The general behavior is similar to the software-based address related end of frame

handling, except that transmit data is not written to the locations TBUF[31:0], but to the FIFO input locations IN[31:0] instead. In this case, software must not write to any of the TBUF locations.

- TBUF related end of frame handling:
  If bit PCR.FEM = 0, an end of frame is assumed if the transmit buffer TBUF does not contain valid transmit data at the end of a data word transmission (TCSR.TDV = 0 or in Stop Mode). In this case, the software has to take care that TBUF does not run empty during a data frame in Run Mode. If bit PCR.FEM = 1, signal MSLS stays active while the transmit buffer is waiting for new data (TCSR.TDV = 1 again) or until Stop Mode is left.
- Explicit end of frame by software:
  The software can explicitly stop a frame by clearing bit PSR.MSLS by writing a 1 to the related bit position in register PSCR. This write action immediately clears bit PSR.MSLS, whereas the internal MSLS signal becomes inactive after finishing a currently running word transfer and respecting the slave select delays T_{td} and T_{nf}.

14.4.4 Operating the SSC in Slave Mode

In order to operate the SSC in slave mode, the following issues have to be considered:

- Select SSC mode:
  It is recommended to configure all parameters of the SSC that do not change during run time while CCR.MODE = 0000_B. Bit field SCTR.TRM = 01_B has to be programmed. The configuration of the input stages has to be done while CCR.MODE = 0000_B to avoid unintended edges of the input signals and the SSC mode can be enabled afterwards by CCR.MODE = 0001_B.
- Pin connections:
  Establish the connection of the input stage (DX0, DX3, DX4, DX5) with the selected receive data input pin (DIN[3:0]) with DXnCR.INSW = 1 and configure the transmit data output pin (DOUT[3:0]). One, two or four such connections may be needed depending on the protocol. For half-duplex configurations, hardware port control can be also used to establish the required connections.
  Establish a connection of input stage DX1 with the selected shift clock input pin (signal SCLKIN) with DX1CR.INSW = 1.
  Establish a connection of input stage DX2 with the selected slave select input pin (signal SELIN) with DX2CR.INSW = 1. If no slave select input signal is used, the DX2 stage has to deliver a 1-level to the data shift unit to allow data reception and transmission. If a slave device is not selected (DX2 stage delivers a 0 to the data shift unit) and a shift clock pulse are received, the incoming data is not received and the DOUTx signal outputs the passive data level defined by SCTR.PDL.
  Note that the step to enable the alternate output port functions should only be done after the SSC mode is enabled, to avoided unintended spikes on the output.
- Baud rate generation:
  The baud rate generator is not needed and can be switched off by the fractional divider.
- Data format configuration:
  If required, the shift mode can be set up for reception and/or transmission of two or four data bits at one time by programming the register SCTR.
- Slave select generation:
  The slave select delay generation is not needed and can be switched off. The bits and bit fields MSLSEN, SELCTR, SELINV, CTQSEL1, PCTQ1, DCTQ1, MSLSIEN, SELO[7:0], and TIWEN in register PCR are not necessary and can be programmed to 0.

14.4.4.1 Protocol Interrupts

The following protocol-related events generated in SSC mode and can lead to a protocol interrupt. They are related to the start and the end of a data frame. After the start of a data frame a new setting could be programmed for the next data frame and after the end of a data frame the SSC connections to pins can be changed.

Please note that the bits in register PSR are not all automatically cleared by hardware and have to be cleared by software in order to monitor new incoming events.

- MSLS event:
  The MSLS generation being switched off, this event is not available.
- DX2T event:
  The slave select input signal SELIN is handled by the DX2 stage and the edges of the selected signal can generate a protocol interrupt. This interrupt allows to indicate that a data frame has started and/or that a data frame has been completely finished. A programmable edge detection for the DX2 input signal activates DX2T, sets bit PSR.DX2TEV and additionally, a protocol interrupt can be generated if PCR.DX2TIEN = 1. The actual state of the selected input signal can be read out at PSR.DX2S to take appropriate actions when this interrupt has been detected.
- Parity Error Interrupt:
  This interrupt indicates that there is a mismatch in the received parity bit (in RBUFSR.PAR) with the calculated parity bit of the last received word of a data frame.

14.4.4.2 End-of-Frame Control

In slave mode, the following possibilities exist to determine the frame length. The slave device either has to refer to an external slave select signal, or to the number of received data bits.

- Frame length known in advance by the slave device, no slave select:
  In this case bit field SCTR.FLE can be programmed to the known value (if it does not exceed 63 bits). A currently running data word transfer is considered as finished if the programmed frame length is reached.
- Frame length not known by the slave, no slave select:
  In this case, the slave device's software has to decide on data word base if a frame is finished. Bit field SCTR.FLE can be either programmed to the word length SCTR.WLE, or to its maximum value to disable the slave internal frame length evaluation by counting received bits.
- Slave device addressed via slave select signal SELIN:
  If the slave device is addressed by a slave select signal delivered by the communication master, the frame start and end information are given by this signal. In this case, bit field SCTR.FLE should be programmed to its maximum value to disable the slave internal frame length evaluation.

14.4.5 SSC Protocol Registers

In SSC mode, the registers PCR and PSR handle SSC related information.

14.4.5.1 SSC Protocol Control Registers

In SSC mode, the PCR register bits or bit fields are defined as described in this section.

PCR
Protocol Control Register [SSC Mode]

($3C_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MCLK	0					SLPHSEL	TIWEN	SELO							
rw	rw					rw	rw	rw							

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DX2TIEN	MSLSIEN	PARIEN	DCTQ1					PCTQ1		CTQSEL1		FEM	SELINV	SELCTR	MSLSEN
rw	rw	rw	rw					rw		rw		rw	rw	rw	rw

Field	Bits	Type	Description
MSLSEN	0	rw	**MSLS Enable** This bit enables/disables the generation of the master slave select signal MSLS. If the SSC is a transfer slave, the SLS information is read from a pin and the internal generation is not needed. If the SSC is a transfer master, it has to provide the MSLS signal. 0_B The MSLS generation is disabled (MSLS = 0). This is the setting for SSC slave mode. 1_B The MSLS generation is enabled. This is the setting for SSC master mode.
SELCTR	1	rw	**Select Control** This bit selects the operating mode for the SELO[7:0] outputs. 0_B The coded select mode is enabled. 1_B The direct select mode is enabled.

Field	Bits	Type	Description
SELINV	2	rw	**Select Inversion** This bit defines if the polarity of the SELO[7:0] outputs in relation to the master slave select signal MSLS. 0_B The SELO outputs have the same polarity as the MSLS signal (active high). 1_B The SELO outputs have the inverted polarity to the MSLS signal (active low).
FEM	3	rw	**Frame End Mode** This bit defines if a transmit buffer content that is not valid for transmission is considered as an end of frame condition for the slave select generation. 0_B The current data frame is considered as finished when the last bit of a data word has been sent out and the transmit buffer TBUF does not contain new data (TDV = 0). 1_B The MSLS signal is kept active also while no new data is available and no other end of frame condition is reached. In this case, the software can accept delays in delivering the data without automatic deactivation of MSLS in multi-word data frames.
CTQSEL1	[5:4]	rw	**Input Frequency Selection** This bit field defines the input frequency f_{CTQIN} for the generation of the slave select delays T_{iw} and T_{nf}. 00_B $f_{CTQIN} = f_{PDIV}$ 01_B $f_{CTQIN} = f_{PPP}$ 10_B $f_{CTQIN} = f_{SCLK}$ 11_B $f_{CTQIN} = f_{MCLK}$
PCTQ1	[7:6]	rw	**Divider Factor PCTQ1 for T_{iw} and T_{nf}** This bit field represents the divider factor PCTQ1 (range = 0 - 3) for the generation of the inter-word delay and the next-frame delay. $T_{iw} = T_{nf} = 1/f_{CTQIN}$ x (PCTQ1 + 1) x (DCTQ1 + 1)
DCTQ1	[12:8]	rw	**Divider Factor DCTQ1 for T_{iw} and T_{nf}** This bit field represents the divider factor DCTQ1 (range = 0 - 31) for the generation of the inter-word delay and the next-frame delay. $T_{iw} = T_{nf} = 1/f_{CTQIN}$ x (PCTQ1 + 1) x (DCTQ1 + 1)

Field	Bits	Type	Description
PARIEN	13	rw	**Parity Error Interrupt Enable** This bit enables/disables the generation of a protocol interrupt with the detection of a parity error. 0_B A protocol interrupt is not generated with the detection of a parity error. 1_B A protocol interrupt is generated with the detection of a parity error.
MSLSIEN	14	rw	**MSLS Interrupt Enable** This bit enables/disables the generation of a protocol interrupt if the state of the MSLS signal changes (indicated by PSR.MSLSEV = 1). 0_B A protocol interrupt is not generated if a change of signal MSLS is detected. 1_B A protocol interrupt is generated if a change of signal MSLS is detected.
DX2TIEN	15	rw	**DX2T Interrupt Enable** This bit enables/disables the generation of a protocol interrupt if the DX2T signal becomes activated (indicated by PSR.DX2TEV = 1). 0_B A protocol interrupt is not generated if DX2T is activated. 1_B A protocol interrupt is generated if DX2T is activated.
SELO	[23:16]	rw	**Select Output** This bit field defines the setting of the SELO[7:0] output lines. 0_B The corresponding SELOx line cannot be activated. 1_B The corresponding SELOx line can be activated (according to the mode selected by SELCTR).
TIWEN	24	rw	**Enable Inter-Word Delay T_{iw}** This bit enables/disables the inter-word delay T_{iw} after the transmission of a data word. 0_B No delay between data words of the same frame. 1_B The inter-word delay T_{iw} is enabled and introduced between data words of the same frame.

Field	Bits	Type	Description
SLPHSEL	25	rw	**Slave Mode Clock Phase Select** This bit selects the clock phase for the data shifting in slave mode. 0_B Data bits are shifted out with the leading edge of the shift clock signal and latched in with the trailing edge. 1_B The first data bit is shifted out when the data shift unit receives a low to high transition from the DX2 stage. Subsequent bits are shifted out with the trailing edge of the shift clock signal. Data bits are always latched in with the leading edge.
MCLK	31	rw	**Master Clock Enable** This bit enables/disables the generation of the master clock output signal MCLK, independent from master or slave mode. 0_B The MCLK generation is disabled and output MCLK = 0. 1_B The MCLK generation is enabled.
0	[30:26]	rw	**Reserved** Returns 0 if read; should be written with 0.

14.4.5.2 SSC Protocol Status Register

In SSC mode, the PSR register bits or bit fields are defined as described in this section. The bits and bit fields in register PSR are not cleared by hardware.

The flags in the PSR register can be cleared by writing a 1 to the corresponding bit position in register PSCR. Writing a 1 to a bit position in PSR sets the corresponding flag, but does not lead to further actions (no interrupt generation). Writing a 0 has no effect. The PSR flags should be cleared by software before enabling a new protocol.

PSR
Protocol Status Register [SSC Mode] (48_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															BRGIF
r															rwh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AIF	RIF	TBIF	TSIF	DLIF	RSIF	0					PARERR	DX2TEV	MSLSEV	DX2S	MSLS
rwh	rwh	rwh	rwh	rwh	rwh	r					rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
MSLS	0	rwh	**MSLS Status** This bit indicates the current status of the MSLS signal. It must be cleared by software to stop a running frame. 0_B The internal signal MSLS is inactive (0). 1_B The internal signal MSLS is active (1).
DX2S	1	rwh	**DX2S Status** This bit indicates the current status of the DX2S signal that can be used as slave select input SELIN. 0_B DX2S is 0. 1_B DX2S is 1.
MSLSEV	2	rwh	**MSLS Event Detected**[1)] This bit indicates that the MSLS signal has changed its state since MSLSEV has been cleared. Together with the MSLS status bit, the activation/deactivation of the MSLS signal can be monitored. 0_B The MSLS signal has not changed its state. 1_B The MSLS signal has changed its state.

Field	Bits	Type	Description
DX2TEV	3	rwh	**DX2T Event Detected**[1)] This bit indicates that the DX2T trigger signal has been activated since DX2TEV has been cleared. 0_B The DX2T signal has not been activated. 1_B The DX2T signal has been activated.
PARERR	4	rwh	**Parity Error Event Detected**[1)] This bit indicates that there is a mismatch in the received parity bit (in RBUFSR.PAR) with the calculated parity bit of the last received word of the data frame. 0_B A parity error event has not been activated. 1_B A parity error event has been activated.
RSIF	10	rwh	**Receiver Start Indication Flag** 0_B A receiver start event has not occurred. 1_B A receiver start event has occurred.
DLIF	11	rwh	**Data Lost Indication Flag** 0_B A data lost event has not occurred. 1_B A data lost event has occurred.
TSIF	12	rwh	**Transmit Shift Indication Flag** 0_B A transmit shift event has not occurred. 1_B A transmit shift event has occurred.
TBIF	13	rwh	**Transmit Buffer Indication Flag** 0_B A transmit buffer event has not occurred. 1_B A transmit buffer event has occurred.
RIF	14	rwh	**Receive Indication Flag** 0_B A receive event has not occurred. 1_B A receive event has occurred.
AIF	15	rwh	**Alternative Receive Indication Flag** 0_B An alternative receive event has not occurred. 1_B An alternative receive event has occurred.
BRGIF	16	rwh	**Baud Rate Generator Indication Flag** 0_B A baud rate generator event has not occurred. 1_B A baud rate generator event has occurred.
0	[9:5], [31:17]	r	**Reserved** Returns 0 if read; not modified in SSC mode.

1) This status bit can generate a protocol interrupt in SSC mode (see **Page 14-22**). The general interrupt status flags are described in the general interrupt chapter.

14.4.6 SSC Timing Considerations

The input and output signals have to respect certain timings in order to ensure correct data reception and transmission. In addition to module internal timings (due to input filters, reaction times on events, etc.), also the timings from the input pin via the input stage (T_{in}) to the module and from the module via the output driver stage to the pin (T_{out}), as well as the signal propagation on the wires (T_{prop}) have to be taken into account.

Please note that there might be additional delays in the DXn input stages, because the digital filter and the synchronization stages lead to systematic delays, that have to be considered if these functions are used.

14.4.6.1 Closed-loop Delay

A system-inherent limiting factor for the baud rate of an SSC connection is the closed-loop delay. In a typical application setup, a communication master device is connected to a slave device in full-duplex mode with independent lines for transmit and receive data. In a general case, all transmitters refer to one shift clock edge for transmission and all receivers refer to the other shift clock edge for reception. The master device's SSC module sends out the transmit data, the shift clock and optionally the slave select signal. Therefore, the baud rate generation (BRG) and slave select generation (SSG) are part of the master device. The frame control is similar for SSC modules in master and slave mode, the main difference is the fact which module generates the shift clock and optionally, the slave select signals.

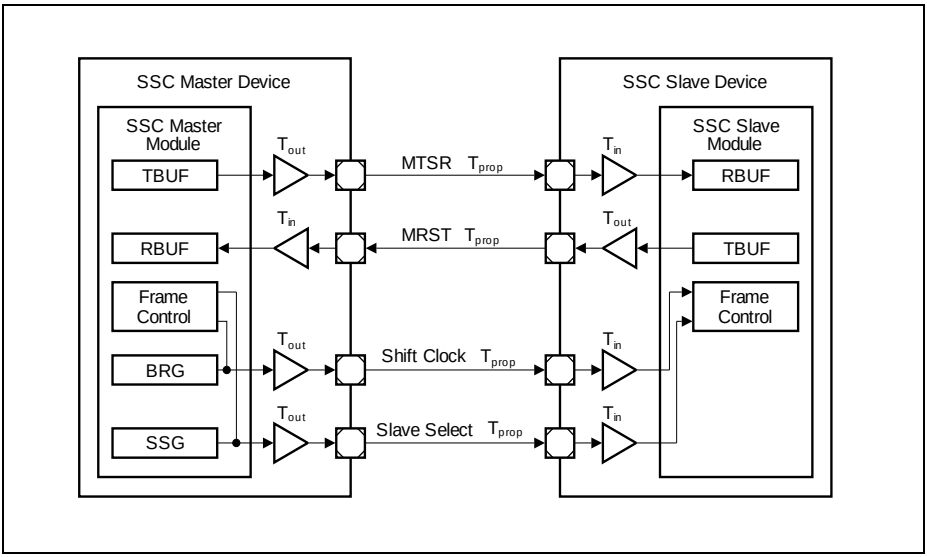


Figure 14-46 SSC Closed-loop Delay

The signal path between the SSC modules of the master and the slave device includes the master's output driver, the wiring to the slave device and the slave device's input stage. With the received shift clock edges, the slave device receives the master's transmit data and transmits its own data back to the master device, passing by a similar signal path in the other direction. The master module receives the slave's transmit data related to its internal shift clock edges. In order to ensure correct data reception in the master device, the slave's transmit data has to be stable (respecting setup and hold times) as master receive data with the next shift clock edge of the master (generally 1/2 shift clock period). To avoid data corruption, the accumulated delays of the input and output stages, the signal propagation on the wiring and the reaction times of the transmitter/receiver have to be carefully considered, especially at high baud rates.
In the given example, the time between the generation of the shift clock signal and the evaluation of the receive data by the master SSC module is given by the sum of T_{out_master} + 2 x T_{prop} + T_{in_slave} + T_{out_slave} + T_{in_master} + module reaction times + input setup times.
The input path is characterized by an input delay depending mainly on the input stage characteristics of the pads. The output path delay is determined by the output driver delay and its slew rate, the external load and current capability of the driver. The device specific values for the input/output driver are given in the Data Sheet.

Figure 14-47 describes graphically the closed-loop delay and the effect of two delay compensation options discussed in **Section 14.4.6.2** and **Section 14.4.6.3**.

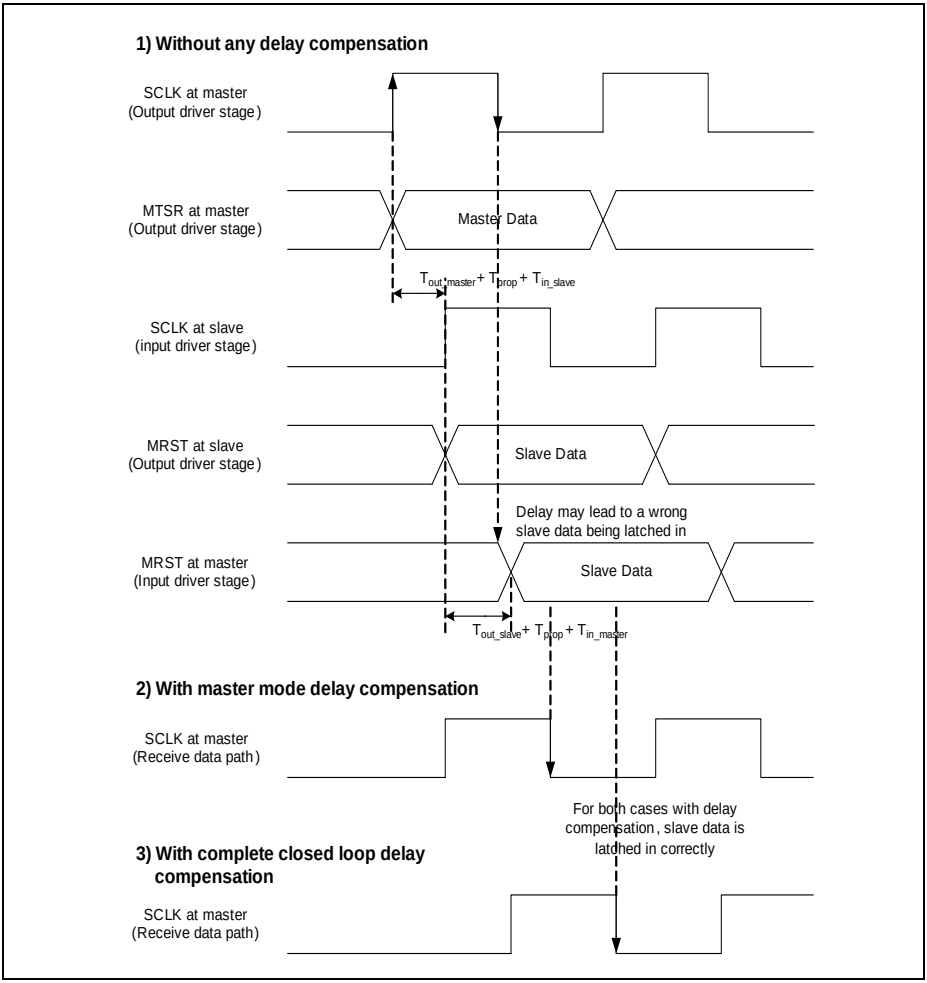


Figure 14-47 SSC Closed-loop Delay Timing Waveform

14.4.6.2 Delay Compensation in Master Mode

A higher baud rate can be reached by delay compensation in master mode. This compensation is possible if (at least) the shift clock pin is bidirectional.

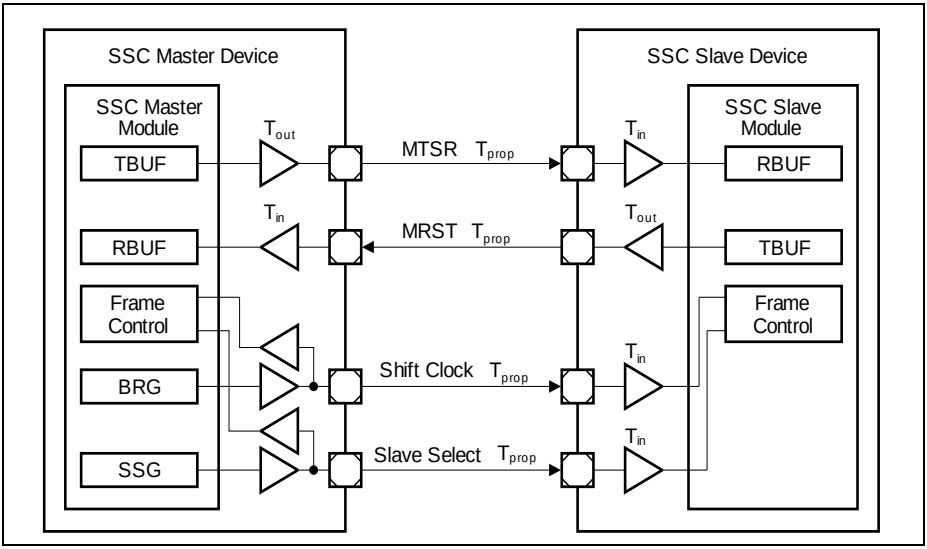


Figure 14-48 SSC Master Mode with Delay Compensation

If the receive shift clock signal in master mode is directly taken from the input function in parallel to the output signal, the output delay of the master device's shift clock output is compensated and only the difference between the input delays of the master and the slave devices have to be taken into account instead of the complete master's output delay and the slave's input delay of the shift clock path. The delay compensation is enabled with DX1CR.DCEN = 1 while DX1CR.INSW = 0 (transmit shift clock is taken from the baud-rate generator).

In the given example, the time between the evaluation of the shift clock signal and the receive data by the master SSC module is reduced by $T_{in\_master} + T_{out\_master}$.

Although being a master mode, the shift clock input and optionally the slave select signal are not directly connected internally to the data shift unit, but are taken as external signals from input pins. The delay compensation does not lead to additional pins for the SSC communication if the shift clock output pin (slave select output pin, respectively) is/are bidirectional. In this case, the input signal is decoupled from other internal signals, because it is related to the signal level at the pin itself.

14.4.6.3 Complete Closed-loop Delay Compensation

Alternatively, the complete closed-loop delay can be compensated by using one additional pin on both the SSC master and slave devices for the SSC communication.

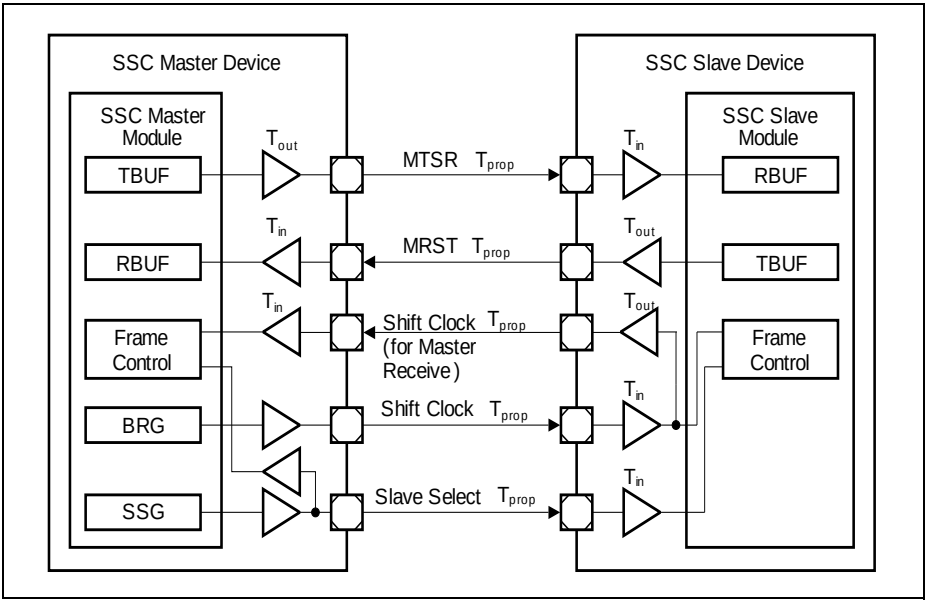


Figure 14-49 SSC Complete Closed-loop Delay Compensation

The principle behind this delay compensation method is to have the slave feedback the shift clock back to the master, which uses it as the receive shift clock. By going through a complete closed-loop signal path, the receive shift clock is thus fully compensated.

The slave has to setup the SCLKOUT pin function to output the shift clock by setting the bit BRG.SCLKOSEL to 1, while the master has to setup the DX1 pin function to receive the shift clock from the slave and enable the delay compensation with DX1CR.DCEN = 1 and DX1CR.INSW = 0.

14.5 Inter-IC Bus Protocol (IIC)

The IIC protocol of the USIC refers to the IIC bus specification **[11]**. Contrary to that specification, the USIC device assumes rise/fall times of the bus signals of max. 300 ns in all modes. Please refer to the pad characteristics in the AC/DC chapter for the driver capability. CBUS mode and HS mode are not supported.
The IIC mode is selected by CCR.MODE = 0100_B with CCFG.IIC = 1 (IIC mode available).

14.5.1 Introduction

USIC IIC Features:

- Two-wire interface, with one line for shift clock transfer and synchronization (shift clock SCL), the other one for the data transfer (shift data SDA)
- Communication in standard mode (100 kBit/s) or in fast mode (up to 400 kBit/s)
- Support of 7-bit addressing, as well as 10-bit addressing
- Master mode operation,
  where the IIC controls the bus transactions and provides the clock signal.
- Slave mode operation,
  where an external master controls the bus transactions and provides the clock signal.
- Multi-master mode operation,
  where several masters can be connected to the bus and bus arbitration can take place, i.e. the IIC module can be master or slave. The master/slave operation of an IIC bus participant can change from frame to frame.
- Efficient frame handling (low software effort), also allowing DMA transfers
- Powerful interrupt handling due to multitude of indication flags
- Compensation support for input delays

14.5.1.1 Signal Description

An IIC connection is characterized by two wires (SDA and SCL). The output drivers for these signals must have open-drain characteristics to allow the wired-AND connection of all SDA lines together and all SCL lines together to form the IIC bus system. Due to this structure, a high level driven by an output stage does not necessarily lead immediately to a high level at the corresponding input. Therefore, each SDA or SCL connection has to be input and output at the same time, because the input function always monitors the level of the signal, also while sending.

- Shift data SDA: input handled by DX0 stage, output signal DOUT0
- Shift clock SCL: input handled by DX1 stage, output signal SCLKOUT

Figure 14-29 shows a connection of two IIC bus participants (modules IIC A and IIC B) using the USIC. In this example, the pin assignment of module IIC A shows separate pins for the input and output signals for SDA and SCL. This assignment can be used if the application does not provide pins having DOUT0 and a DX0 stage input for the same pin

(similar for SCLKOUT and DX1). The pin assignment of module IIC B shows the connection of DOUT0 and a DX0 input at the same pin, also for SCLKOUT and a DX1 input.

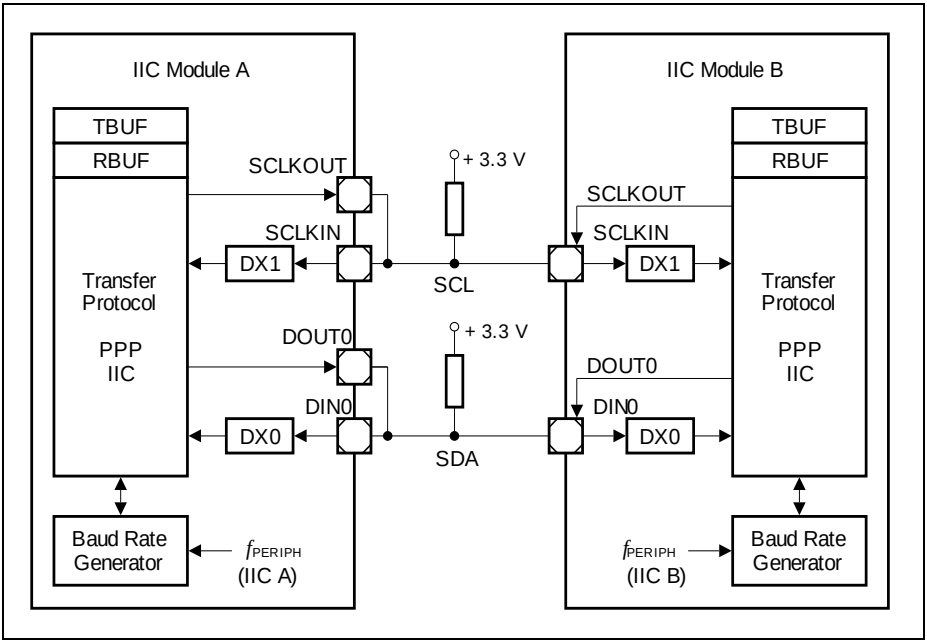


Figure 14-50 IIC Signal Connections

14.5.1.2 Symbols

A symbol is a sequence of edges on the lines SDA and SCL. Symbols contain 10 or 25 time quanta t_q, depending on the selected baud rate. The baud rate generator determines the length of the time quanta t_q, the sequence of edges in a symbol is handled by the IIC protocol pre-processor, and the sequence of symbols can be programmed by the user according to the application needs.

The following symbols are defined:

- Bus idle:
  SDA and SCL are high. No data transfer takes place currently.
- Data bit symbol:
  SDA stable during the high phase of SCL. SDA then represents the transferred bit value. There is one clock pulse on SCL for each transferred bit of data. During data transfers SDA may only change while SCL is low.

- Start symbol:
  Signal SDA being high followed by a falling edge of SDA while SCL is high indicates a start condition. This start condition initiates a data transfer over the IIC bus after the bus has been idle.
- Repeated start symbol:
  This start condition initiates a data transfer over the bus after a data symbol when the bus has not been idle. Therefore, SDA is set high and SCL low, followed by a start symbol.
- Stop symbol:
  A rising edge on SDA while SCL is high indicates a stop condition. This stop condition terminates a data transfer to release the bus to idle state. Between a start condition and a stop condition an arbitrary number of bytes may be transferred.

14.5.1.3 Frame Format

Data is transferred by the 2-line IIC bus (SDA, SCL) using a protocol that ensures reliable and efficient transfers. The sender of a (data) byte receives and checks the value of the following acknowledge field. The IIC being a wired-AND bus system, a 0 of at least one device leads to a 0 on the bus, which is received by all devices.

A data word consists of 8 data bit symbols for the data value, followed by another data bit symbol for the acknowledge bit. The data word can be interpreted as address information (after a start symbol) or as transferred data (after the address).

In order to be able to receive an acknowledge signal, the sender of the data bits has to release the SDA line by sending a 1 as acknowledge value. Depending on the internal state of the receiver, the acknowledge bit is either sent active or passive.

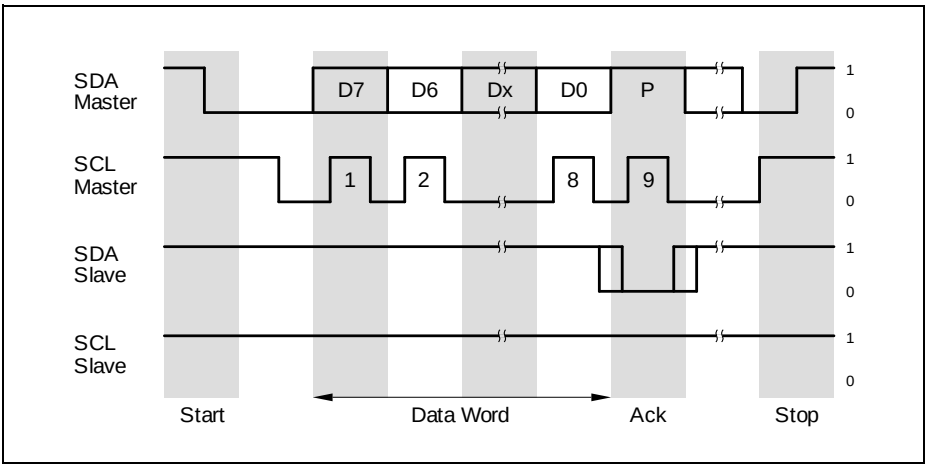


Figure 14-51 IIC Frame Example (simplified)

14.5.2 Operating the IIC

In order to operate the IIC protocol, the following issues have to be considered:

- Select IIC mode:
  It is recommended to configure all parameters of the IIC that do not change during run time while CCR.MODE = 0000_B. Bit field SCTR.TRM = 11_B should be programmed. The configuration of the input stages has to be done while CCR.MODE = 0000_B to avoid unintended edges of the input signals and the IIC mode can be enabled by CCR.MODE = 0100_B afterwards.
- Pin connections:
  Establish a connection of input stage DX0 (with DX0CR.DPOL = 0) to the selected shift data pin SDA (signal DIN0) with DX0CR.INSW = 0 and configure the transmit data output signal DOUT0 (with SCTR.DOCFG = 00_B) to the same pin. If available, this can be the same pin for input and output, or connect the selected input pin and the output pin to form the SDA line.
  The same mechanism applies for the shift clock line SCL. Here, signal SCLKOUT (with BRG.SCLKCFG = 00_B) and an input of the DX1 stage have to be connected (with DX1CR.DPOL = 0).
  The input stage DX2 is not used for the IIC protocol.
  If the digital input filters are enabled in the DX0/1 stages, their delays have to be taken into account for correct calculation of the signal timings.
  The pins used for SDA and SCL have to be set to open-drain mode to support the wired-AND structure of the IIC bus lines.
  Note that the step to enable the alternate output port functions should only be done after the IIC mode is enabled, to avoided unintended spikes on the output.
- Bit timing configuration:
  In standard mode (100 kBit/s) a minimum module frequency of 2 MHz is necessary, whereas in fast mode (400 kBit/s) a minimum of 10 MHz is required. Additionally, if the digital filter stage should be used to eliminate spikes up to 50 ns, a filter frequency of 20 MHz is necessary.
  There could be an uncertainty in the SCL high phase timing of maximum $1/f_{PPP}$ if another IIC participant lengthens the SCL low phase on the bus.
  More details are given in **Section 14.5.3**.
- Data format configuration:
  The data format has to be configured for 8 data bits (SCTR.WLE = 7), unlimited data flow (SCTR.FLE = $3F_H$), and MSB shifted first (SCTR.SDIR = 1). The parity generation has to be disabled (CCR.PM = 00_B).
- General hints:
  The IIC slave module becomes active (for reception or transmission) if it is selected by the address sent by the master. In the case that the slave sends data to the master, it uses the transmit path. So a master must not request to read data from the slave address defined for its own channel in order to avoid collisions.
  The built-in error detection mechanisms are only activated while the IIC module is

taking part in IIC bus traffic.
If the slave can not deal with too high frequencies, it can lengthen the low phase of the SCL signal.
For data transfers according to the IIC specification, the shift data line SDA shall only change while SCL = 0 (defined by IIC bus specification).

14.5.2.1 Transmission Chain

The IIC bus protocol requiring a kind of in-bit-response during the arbitration phase and while a slave is transmitting, the resulting loop delay of the transmission chain can limit the reachable maximal baud rate, strongly depending on the bus characteristics (bus load, module frequency, etc.).

Figure 14-50 shows the general signal path and the delays in the case of a slave transmission. The shift clock SCL is generated by the master device, output on the wire, then it passes through the input stage and the input filter. Now, the edges can be detected and the SDA data signal can be generated accordingly. The SDA signal passes through the output stage and the wire to the master receiver part. There, it passes through the input stage and the input filter before it is sampled.

This complete loop has to be finished (including all settling times to obtain stable signal levels) before the SCL signal changes again. The delays in this path have to be taken into account for the calculation of the baud rate as a function of f_{PERIPH} and f_{PPP}.

14.5.2.2 Byte Stretching

If a device is selected as transceiver and should transmit a data byte but the transmit buffer TBUF does not contain valid data to be transmitted, the device ties down SCL = 0 at the end of the previous acknowledge bit. The waiting period is finished if new valid data has been detected in TBUF.

14.5.2.3 Master Arbitration

During the address and data transmission, the master transmitter checks at the rising edge of SCL for each data bit if the value it is sending is equal to the value read on the SDA line. If yes, the next data bit values can be 0. If this is not the case (transmitted value = 1, value read = 0), the master has lost the transmit arbitration. This is indicated by status flag PSR.ARL and can generate a protocol interrupt if enabled by PCR.ARLIEN.
When the transmit arbitration has been lost, the software has to initialize the complete frame again, starting with the first address byte together with the start condition for a new master transmit attempt. Arbitration also takes place for the ACK bit.

14.5.2.4 Non-Acknowledge and Error Conditions

In case of a non-acknowledge or an error, the TCSR.TDV flag remains set, but no further transmission will take place. User software must invalidate the transmit buffer and disable transmissions (by writing FMRL.MTDV = 10_B), before configuring the transmission (by writing TBUF) again with appropriate values to react on the previous event. In the case the FIFO data buffer is used, additionally the FIFO buffer needs to be flushed and filled again.

14.5.2.5 Mode Control Behavior

In multi-master mode, only run mode 0 and stop mode 0 are supported, the other modes must not be programmed.

- Run Mode 0:
  Behavior as programmed. If TCSR.TDV = 0 (no new valid TBUF entry found) when a new TBUF entry needs to be processed, the IIC module waits for TDV becoming set to continue operation.
- Run Mode 1:
  Behavior as programmed. If in master mode, TCSR.TDV = 0 (no new valid TBUF entry found) when a new TBUF entry needs to be processed, the IIC module sends a stop condition to finish the frame. In slave mode, no difference to run mode 0.
- Stop Mode 0:
  Bit TCSR.TDV is internally considered as 0 (the bit itself is not modified by the stop mode). A currently running word is finished normally, but no new word is started in case of master mode (wait for TDV active).
  Bit TDV being considered as 0 for master and slave, the slave will force a wait state on the bus if read by an external master, too.
  Additionally, it is not possible to force the generation of a STOP condition out of the wait state. The reason is, that a master read transfer must be finished with a not-acknowledged followed by a STOP condition to allow the slave to release his SDA line. Otherwise the slave may force the SDA line to 0 (first data bit of next byte) making it impossible to generate the STOP condition (rising edge on SDA).
  To continue operation, the mode must be switched to run mode 0
- Stop Mode 1:
  Same as stop mode 0, but additionally, a master sends a STOP condition to finish the frame.
  If stop mode 1 is requested for a master device after the first byte of a 10 bit address, a stop condition will be sent out. In this case, a slave device will issue an error interrupt.

14.5.2.6 Data Transfer Interrupt Handling

The data transfer interrupts indicate events related to IIC frame handling. As the data input and output pins are the same in IIC protocol, a IIC transmitter also receives the

output data at its input pin. However, no receive related interrupts will be generated in this case.

- Transmit buffer event:
  The transmit buffer event indication flag PSR.TBIF is set when the content of the transmit buffer TBUF has been loaded to the transmit shift register, indicating that the action requested by the TBUF entry has started.
  With this event, bit TCSR.TDV is cleared. This interrupt can be used to write the next TBUF entry while the last one is in progress (handled by the transmitter part).
- Receive event:
  This receive event indication flag PSR.RIF indicates that a new data byte has been written to the receive buffer RBUF0/1 (except for the first data byte of a new frame, that is indicated by an alternative receive interrupt). The flag becomes set when the data byte is received (after the falling edge of SCL). This interrupt can be used to read out the received data while a new data byte can be in progress (handled by the receiver part).
- Alternate receive event:
  The alternative receive event indication flag AIF is based on bit RBUFSR[9] (same as RBUF[9]), indicating that the received data word has been the first data word of a new data frame.
- Transmit shift event:
  The transmit shift event indication flag TSIF is set after the start of the last data bit of a data byte.
- Receive start event:
  The receive start event indication flag RSIF is set after the sample point of the first data bit of a data byte.

Note: The transmit shift and receive start events can be ignored if the application does not require them during the IIC data transfer.

14.5.2.7 IIC Protocol Interrupt Events

The following protocol-related events are generated in IIC mode and can lead to a protocol interrupt.

Please note that the bits in register PSR are not all automatically cleared by hardware and have to be cleared by software in order to monitor new incoming events.

- start condition received at a correct position in a frame (PSR.SCR)
- repeated start condition received at a correct position in a frame (PSR.RSCR)
- stop condition transferred at a correct position in a frame (PSR.PCR)
- master arbitration lost (PSR.ARL)
- slave read requested (PSR.SRR)
- acknowledge received (PSR.ACK)
- non-acknowledge received (PSR.NACK)
- start condition not at the expected position in a frame (PSR.ERR)

- stop condition not at the expected position in a frame (PSR.ERR)
- as slave, 10-bit address interrupted by a stop condition after the first address byte (PSR.ERR)
- TDF slave code in master mode (PSR.WTDF)
- TDF master code in slave mode (PSR.WTDF)
- Reserved TDF code found (PSR.WDTF)
- Start condition code during a running frame in master mode (PSR.WTDF)
- Data byte transmission code after transfer direction has been changed to reception (master read) in master mode (PSR.WTDF)

If a wrong TDF code is found in TBUF, the error event is active until the TDF value is either corrected or invalidated. If the related interrupt is enabled, the interrupt handler should check PSR.WDTF first and correct or invalidate TBUF, before dealing with the other possible interrupt events.

14.5.2.8 Baud Rate Generator Interrupt Handling

The baud rate generator interrupt indicate that the capture mode timer has reached its maximum value. With this event, the bit PSR.BRGIF is set.

14.5.2.9 Receiver Address Acknowledge

After a (repeated) start condition, the master sends a slave address to identify the target device of the communication. The start address can comprise one or two address bytes (for 7 bit or for 10 bit addressing schemes). After an address byte, a slave sensitive to the transmitted address has to acknowledge the reception.

Therefore, the slave's address can be programmed in the device, where it is compared to the received address. In case of a match, the slave answers with an acknowledge (SDA = 0). Slaves that are not targeted answer with an non-acknowledge (SDA = 1).
In addition to the match of the programmed address, another address byte value has to be answered with an acknowledge if the slave is capable to handle the corresponding requests. The address byte 00_H indicates a general call address, that can be acknowledged. The value 01_H stands for a start byte generation, that is not acknowledged

In order to allow selective acknowledges for the different values of the address byte(s), the following control mechanism is implemented:

- The address byte 00_H is acknowledged if bit PCR.ACK00 is set.
- The address byte 01_H is not acknowledged.
- The first 7 bits of a received first address byte are compared to the programmed slave address (PCR.SLAD[15:9]). If these bits match, the slave sends an acknowledge. In addition to this, if the slave address is programmed to 1111 $0XX_B$, the slave device waits for a second address byte and compares it also to PCR.SLAD[7:0] and sends an acknowledge accordingly to cover the 10 bit addressing mode. The user has to

take care about reserved addresses (refer to IIC specification for more detailed description). Only the address 1111 $0XX_B$ is supported.

Under each of these conditions, bit PSR.SLSEL will be set when the addressing delivered a match. This bit is cleared automatically by a (repeated) start condition.

14.5.2.10 Receiver Handling

A selected slave receiver always acknowledges a received data byte. If the receive buffers RBUF0/1 are already full and can not accept more data, the respective register is overwritten (PSR.DLI becomes set in this case and a protocol interrupt can be generated).

An address reception also uses the registers RBUF0/1 to store the address before checking if the device is selected. The received addresses do not set RDV0/1, so the addresses are not handled like received data.

14.5.2.11 Receiver Status Information

In addition to the received data byte, some IIC protocol related information is stored in the 16-bit data word of the receive buffer. The received data byte is available at the bit positions RBUF[7:0], whereas the additional information is monitored at the bit positions RBUF[12:8]. This structure allows to identify the meaning of each received data byte without reading additional registers, also when using a FIFO data buffer.

- RBUF[8]:
  Value of the received acknowledge bit. This information is also available in RBUFSR[8] as protocol argument.
- RBUF[9]:
  A 1 at this bit position indicates that after a (repeated) start condition followed by the address reception the first data byte of a new frame has been received. A 0 at this bit position indicates further data bytes. This information is also available in RBUFSR[9], allowing different interrupt routines for the address and data handling.
- RBUF[10]:
  A 0 at this bit position indicates that the data byte has been received when the device has been in slave mode, whereas a 1 indicates a reception in master mode.
- RBUF[11]:
  A 1 at this bit position indicates an incomplete/erroneous data byte in the receive buffer caused by a wrong position of a START or STOP condition in the frame. The bit is not identical to the frame error status bit in PSR, because the bit in the PSR has to be cleared by software (“sticky” bit), whereas RBUF[11] is evaluated data byte by data byte. If RBUF[11] = 0, the received data byte has been correct, independent of former errors.
- RBUF[12]:
  A 0 at this bit position indicates that the programmed address has been received. A 1 indicates a general call address.

14.5.3 Symbol Timing

The symbol timing of the IIC is determined by the master stimulating the shift clock line SCL. It is different in each of the modes.

- 100 kBaud standard mode (PCR.STIM = 0):
  The symbol timing is based on 10 time quanta t_q per symbol. A minimum module clock frequency f_{PERIPH} = 2 MHz is required.
- 400 kBaud fast mode (PCR.STIM = 1):
  The symbol timing is based on 25 time quanta t_q per symbol. A minimum module clock frequency f_{PERIPH} = 10 MHz is required.

The baud rate setting should only be changed while the transmitter and the receiver are idle or CCR.MODE = 0. The bits in register BRG define the length of a time quantum t_q that is given by one period of f_{PCTQ}.

- BRG.CTQSEL
  to define the input frequency f_{CTQIN} for the time quanta generation
- BRG.PCTQ
  to define the length of a time quantum (division of f_{CTQIN} by 1, 2, 3, or 4)
- BRG.DCTQ
  to define the number of time quanta per symbol (number of t_q = DCTQ + 1)

The standard setting is given by CTQSEL = 00_B ($f_{CTQIN} = f_{PDIV}$) and PPPEN = 0 ($f_{PPP} = f_{IN}$). Under these conditions, the frequency f_{PCTQ} is given by:

$$f_{PCTQ} = f_{PIN} \times \frac{1}{PDIV + 1} \times \frac{1}{PCTQ + 1} \quad (14.10)$$

To respect the specified SDA hold time of 300 ns for standard mode and fast mode after a falling edge of signal SCL, a hold delay t_{HDEL} has been introduced. It also prevents an erroneous detection of a start or a stop condition. The length of this delay can be programmed by bit field PCR.HDEL. Taking into account the input sampling and output update, bit field HDEL can be programmed according to:

(14.11)

$$HDEL \geq 300\ ns \times f_{PPP} - \left(3 \times \frac{f_{PPP}}{f_{PERIPH}}\right) + 1 \quad \text{with digital filter and HDELmin} = 2$$

$$HDEL \geq 300\ ns \times f_{PPP} - \left(3 \times \frac{f_{PPP}}{f_{PERIPH}}\right) + 2 \quad \text{without digital filter and HDELmin} = 1$$

If the digital input filter is used, HDEL compensates the filter delay of 2 filter periods (f_{PPP} should be used) in case of a spike on the input signal. This ensures that a data bit on the SDA line changing just before the rising edge or behind the falling edge of SCL will not be treated as a start or stop condition.

14.5.3.1 Start Symbol

Figure 14-52 shows the general start symbol timing.

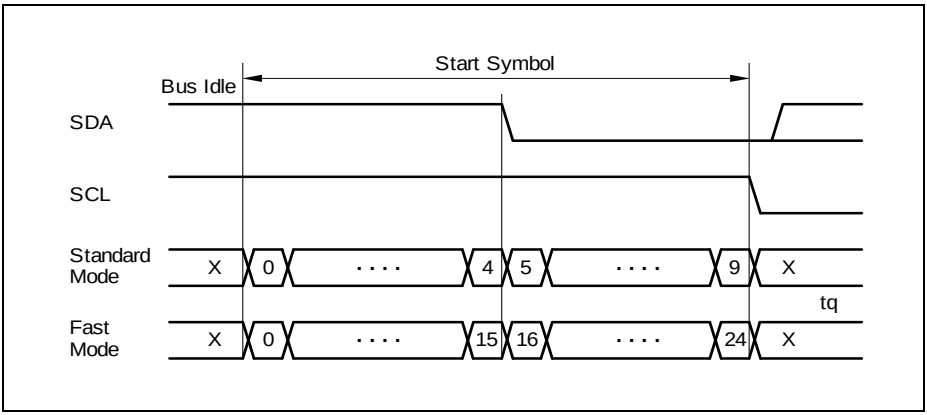


Figure 14-52 Start Symbol Timing

14.5.3.2 Repeated Start Symbol

During the first part of a repeated start symbol, an SCL low value is driven for the specified number of time quanta. Then a high value is output. After the detection of a rising edge at the SCL input, a normal start symbol is generated, as shown in **Figure 14-53**.

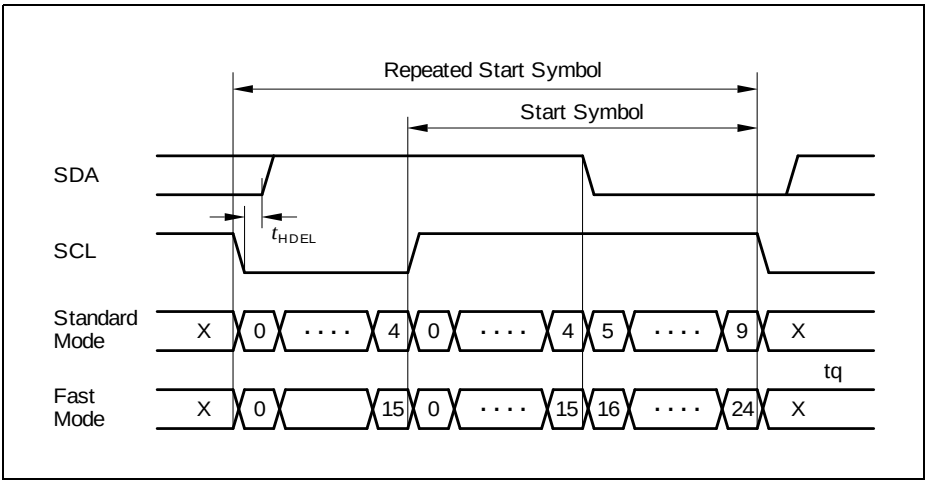


Figure 14-53 Repeated Start Symbol Timing

14.5.3.3 Stop Symbol

Figure 14-54 shows the stop symbol timing.

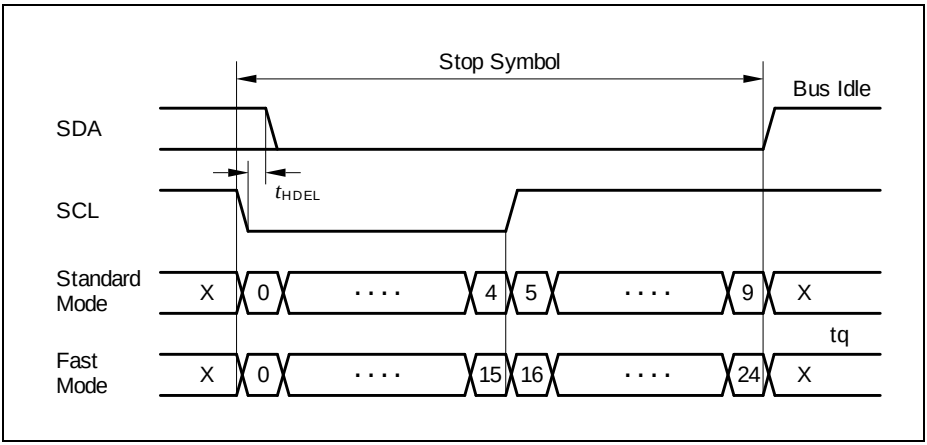


Figure 14-54 Stop Symbol Timing

14.5.3.4 Data Bit Symbol

Figure 14-55 shows the general data bit symbol timing.

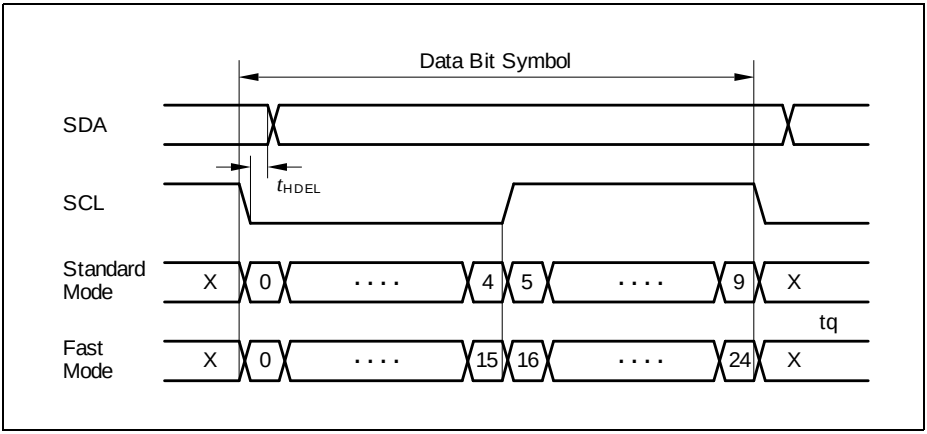


Figure 14-55 Data Bit Symbol

Output SDA changes after the time t_{HDEL} defined by PCR.HDEL has elapsed if a falling edge is detected at the SCL input to respect the SDA hold time. The value of PCR.HDEL allows compensation of the delay of the SCL input path (sampling, filtering).

In the case of an acknowledge transmission, the USIC IIC waits for the receiver indicating that a complete byte has been received. This adds an additional delay of 3 periods of f_{PERIPH} to the path. The minimum module input frequency has to be selected properly to ensure the SDA setup time to SCL rising edge.

14.5.4 Data Flow Handling

The handling of the data flow and the sequence of the symbols in an IIC frame is controlled by the IIC transmitter part of the USIC communication channel. The IIC bus protocol is byte-oriented, whereas a USIC data buffer word can contain up to 16 data bits. In addition to the data byte to be transmitted (located at TBUF[7:0]), bit field TDF (transmit data format) to control the IIC sequence is located at the bit positions TBUF[10:8]. The TDF code defines for each data byte how it should be transmitted (IIC master or IIC slave), and controls the transmission of (repeated) start and stop symbols. This structure allows the definition of a complete IIC frame for an IIC master device only by writing to TBUFx or by using a FIFO data buffer mechanism, because no other control registers have to be accessed. Alternatively, polling of the ACK and NACK bits in PSR register can be performed, and the next data byte is transmitted only after an ACK is received.

If a wrong or unexpected TDF code is encountered (e.g. due to a software error during setup of the transmit buffer), a stop condition will be sent out by the master. This leads to an abort of the currently running frame. A slave module waits for a valid TDF code and sets SCL = 0. The software then has to invalidate the unexpected TDF code and write a valid one.

Please note that during an arbitration phase in multi-master bus systems an unpredictable bus behavior may occur due to an unexpected stop condition.

14.5.4.1 Transmit Data Formats

The following transmit data formats are available in master mode:

Table 14-13 Master Transmit Data Formats

TDF Code	Description
000_B	**Send data byte as master** This format is used to transmit a data byte from the master to a slave. The transmitter sends its data byte (TBUF[7:0]), receives and checks the acknowledge bit sent by the slave.
010_B	**Receive data byte and send acknowledge** This format is used by the master to read a data byte from a slave. The master acknowledges the transfer with a 0-level to continue the transfer. The content of TBUF[7:0] is ignored.

Table 14-13 Master Transmit Data Formats (cont'd)

TDF Code	Description
011_B	**Receive data byte and send not-acknowledge** This format is used by the master to read a data byte from a slave. The master does not acknowledge the transfer with a 1-level to finish the transfer. The content of TBUF[7:0] is ignored.
100_B	**Send start condition** If TBUF contains this entry while the bus is idle, a start condition will be generated. The content of TBUF[7:0] is taken as first address byte for the transmission (bits TBUF[7:1] are the address, the LSB is the read/write control).
101_B	**Send repeated start condition** If TBUF contains this entry and SCL = 0 and a byte transfer is not in progress, a repeated start condition will be sent out if the device is the current master. The current master is defined as the device that has set the start condition (and also won the master arbitration) for the current message. The content of TBUF[7:0] is taken as first address byte for the transmission (bits TBUF[7:1] are the address, the LSB is the read/write control).
110_B	**Send stop condition** If the current master has finished its last byte transfer (including acknowledge), it sends a stop condition if this format is in TBUF. The content of TBUF[7:0] is ignored.
111_B	**Reserved** This code must not be programmed. No additional action except releasing the TBUF entry and setting the error bit in PSR (that can lead to a protocol interrupt).

The following transmit data format is available in slave mode (the symbols in a frame are controlled by the master and the slave only has to send data if it has been "asked" by the master):

Table 14-14 Slave Transmit Data Format

TDF Code	Description
001_B	**Send data byte as slave** This format is used to transmit a data byte from a slave to the master. The transmitter sends its data byte (TBUF[7:0]) plus the acknowledge bit as a 1.

14.5.4.2 Valid Master Transmit Data Formats

Due to the IIC frame format definitions, only some specific sequences of TDF codes are possible and valid. If the USIC IIC module detects a wrong TDF code in a running frame, the transfer is aborted and flag PCR.WTDF is set. Additionally, an interrupt can be generated if enabled by the user. In case of a wrong TDF code, the frame will be aborted immediately with a STOP condition if the USIC IIC master still owns the SDA line. But if the accessed slave owns the SDA line (read transfer), the master must perform a dummy read with a non-acknowledge so that the slave releases the SDA line before a STOP condition can be sent. The received data byte of the dummy read will be stored in RBUF0/1, but RDV0/1 will not be set. Therefore the dummy read will not generate a receive interrupt and the data byte will not be stored into the receive FIFO.

If the transfer direction has changed in the current frame (master read access), the transmit data request (TDF = 000_B) is not possible and won't be accepted (leading to a wrong TDF Code indication).

Table 14-15 Valid TDF Codes Overview

Frame Position	Valid TDF Codes
First TDF code (master idle)	Start (100_B)
Read transfer: second TDF code (after start or repeated start)	Receive with acknowledge (010_B) or receive with not-acknowledge (011_B)
Write transfer: second TDF code (after start or repeated start)	Transmit (000_B), repeated start (101_B), or stop (110_B)
Read transfer: third and subsequent TDF code after acknowledge	Receive with acknowledge (010_B) or receive with not-acknowledge (011_B)
Read transfer: third and subsequent TDF code after not-acknowledge	Repeated start (101_B) or stop (110_B)
Write transfer: third and subsequent TDF code	Transmit (000_B), repeated start (101_B), or stop (110_B)

- First TDF code:
  A master transfer starts with the TDF start code (100_B). All other codes are ignored, but no WTDF error will be indicated.
- TDF code after a start (100_B) or repeated start code (101_B) in case of a read access:
  If a master-read transfer is started (determined by the LSB of the address byte = 1), the transfer direction of SDA changes and the slave will actively drive the data line. In this case, only the codes 010_B and 011_B are valid. To abort the transfer in case of a wrong code, a dummy read must be performed by the master before the STOP condition can be generated.

- TDF code after a start (100_B) or repeated start code (101_B) in case of a write access:
  If a master-write transfer is started (determined by the LSB of the address byte = 0), the master still owns the SDA line. In this case, the transmit (000_B), repeated start (101_B) and stop (110_B) codes are valid. The other codes are considered as wrong. To abort the transfer in case of a wrong code, the STOP condition is generated immediately.
- TDF code of the third and subsequent command in case of a read access with acknowledged previous data byte:
  If a master-read transfer is started (determined by the LSB of the address byte), the transfer direction of SDA changes and the slave will actively drive the data line. To force the slave to release the SDA line, the master has to not-acknowledge a byte transfer. In this case, only the receive codes 010_B and 011_B are valid. To abort the transfer in case of a wrong code, a dummy read must be performed by the master before the STOP condition can be generated.
- TDF code of the third and subsequent command in case of a read access with a not-acknowledged previous data byte:
  If a master-read transfer is started (determined by the LSB of the address byte), the transfer direction of SDA changes and the slave will actively drive the data line. To force the slave to release the SDA line, the master has to not-acknowledge a byte transfer. In this case, only the restart (101_B) and stop code (110_B) are valid. To abort the transfer in case of a wrong code, the STOP condition is generated immediately.
- TDF code of the third and subsequent command in case of a write access:
  If a master-write transfer is started (determined by the LSB of the address byte), the master still owns the SDA line. In this case, the transmit (000_B), repeated start (101_B) and stop (110_B) codes are valid. The other codes are considered as wrong. To abort the transfer in case of a wrong code, the STOP condition is generated immediately.
- After a master device has received a non-acknowledge from a slave device, a stop condition will be sent out automatically, except if the following TDF code requests a repeated start condition. In this case, the TDF code is taken into account, whereas all other TDF codes are ignored.

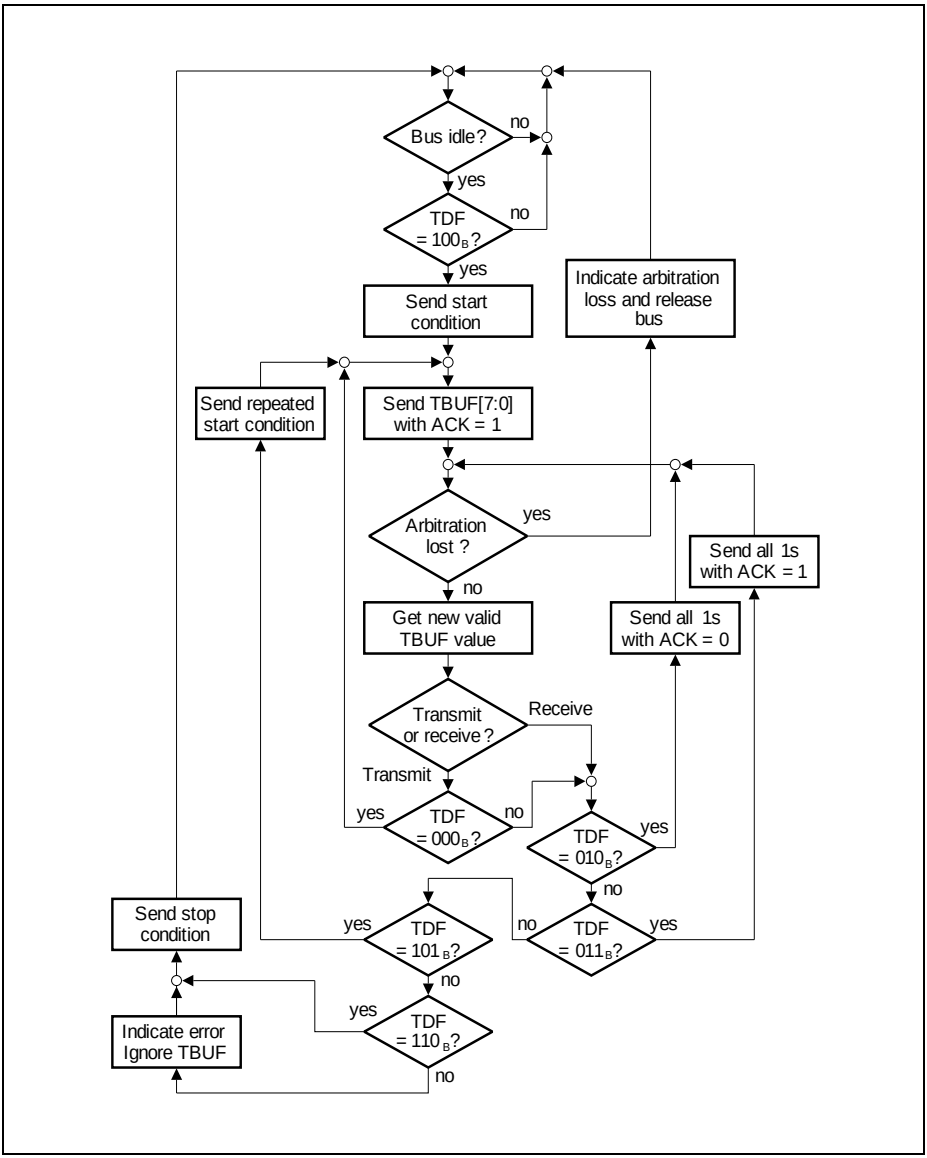


Figure 14-56 IIC Master Transmission

14.5.4.3 Master Transmit/Receive Modes

In master transmit mode, the IIC sends a number of data bytes to a slave receiver. The TDF code sequence for the master transmit mode is shown in **Table 14-16**.

Table 14-16 TDF Code Sequence for Master Transmit

TDF Code Sequence	TBUF[10:8] (TDF Code)	TBUF[7:0]	IIC Response	Interrupt Events
1st code	100_B	Slave address + write bit	Send START condition, slave address and write bit	SCR: Indicates a START condition is detected TBIF: Next word can be written to TBUF
2nd code	000_B	Data or 2nd slave address byte	Send data or 2nd slave address byte	TBIF: Next word can be written to TBUF
Subsequent codes for data transmit	000_B	Data	Send data	TBIF: Next word can be written to TBUF
Last code	110_B	Don't care	Send STOP condition	PCR: Indicates a STOP condition is detected

In master receive mode, the IIC receives a number of data bytes from a slave transmitter. The TDF code sequence for the master receive 7-bit and 10-bit addressing modes are shown in **Table 14-17** and **Table 14-18**.

Table 14-17 TDF Code Sequence for Master Receive (7-bit Addressing Mode)

TDF Code Sequence	TBUF[10:8] (TDF Code)	TBUF[7:0]	IIC Response	Interrupt Events
1st code	100_B	Slave address + read bit	Send START condition, slave address and read bit	SCR: Indicates a START condition is detected TBIF: Next word can be written to TBUF
2nd code	010_B	Don't care	Receive data and send ACK bit	TBIF: Next word can be written to TBUF AIF: First data received can be read

Table 14-17 TDF Code Sequence for Master Receive (7-bit Addressing Mode)

TDF Code Sequence	TBUF[10:8] (TDF Code)	TBUF[7:0]	IIC Response	Interrupt Events
Subsequent codes for data receive	010_B	Don't care	Receive data and send ACK bit	TBIF: Next word can be written to TBUF RIF: Subsequent data received can be read
Code for last data to be received	011_B	Don't care	Receive data and send NACK bit	TBIF: Next word can be written to TBUF RIF: Last data received can be read
Last code	110_B	Don't care	Send STOP condition	PCR: Indicates a STOP condition is detected

Table 14-18 TDF Code Sequence for Master Receive (10-bit Addressing Mode)

TDF Code Sequence	TBUF[10:8] (TDF Code)	TBUF[7:0]	IIC Response	Interrupt Events
1st code	100_B	Slave address (1st byte) + write bit	Send START condition, slave address (1st byte) and write bit	SCR: Indicates a START condition is detected TBIF: Next word can be written to TBUF
2nd code	000_B	Slave address (2nd byte)	Send address (2nd byte)	TBIF: Next word can be written to TBUF
3rd code	101_B	1st slave address + read bit	Send repeated START condition, slave address (1st byte) and read bit	RSCR: Indicates a repeated START condition is detected TBIF: Next word can be written to TBUF
4th code	010_B	Don't care	Receive data and send ACK bit	TBIF: Next word can be written to TBUF AIF: First data received can be read
Subsequent codes for data receive	010_B	Don't care	Receive data and send ACK bit	TBIF: Next word can be written to TBUF RIF: Subsequent data received can be read

Table 14-18 TDF Code Sequence for Master Receive (10-bit Addressing Mode)

TDF Code Sequence	TBUF[10:8] (TDF Code)	TBUF[7:0]	IIC Response	Interrupt Events
Code for last data to be received	011_B	Don't care	Receive data and send NACK bit	TBIF: Next word can be written to TBUF RIF: Last data received from slave can be read
Last code	110_B	Don't care	Send STOP condition	PCR: Indicates a STOP condition is detected

Figure 14-57 shows the interrupt events during the master transmit-slave receive and master receive/slave transmit sequences.

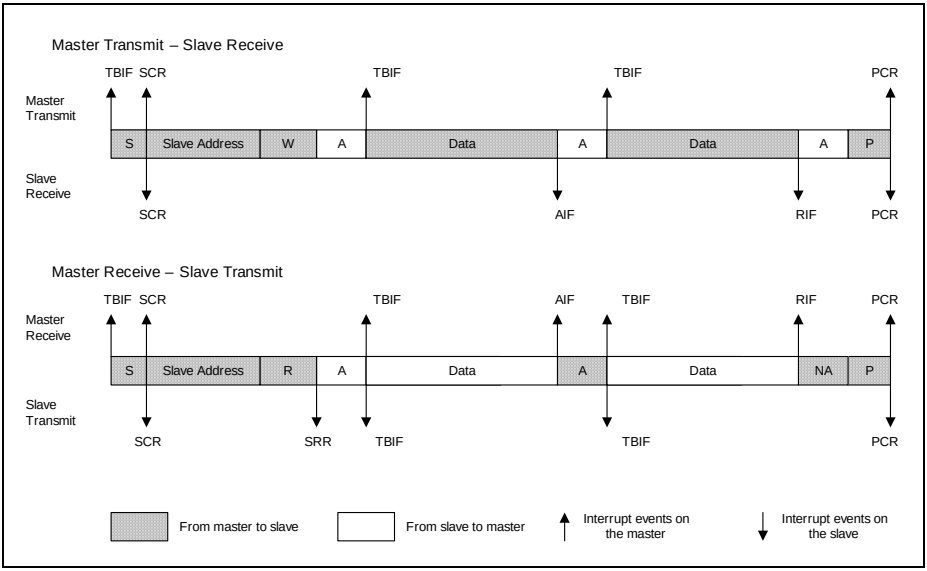


Figure 14-57 Interrupt Events on Data Transfers

14.5.4.4 Slave Transmit/Receive Modes

In slave receive mode, no TDF code needs to be written and data reception is indicated by the alternate receive (AIF) or receive (RIF) events.

In slave transmit mode, upon receiving its own slave address or general call address if this option is enabled, a slave read request event (SRR) will be triggered. The slave IIC then writes the TDF code 001_B and the requested data to TBUF to transmit the data to

the master. The slave does not check if the master reply with an ACK or NACK to the transmitted data.

In both cases, the data transfer is terminated by the master sending a STOP condition, which is indicated by a PCR event. See also **Figure 14-57**.

14.5.5 IIC Protocol Registers

In IIC mode, the registers PCR and PSR handle IIC related information.

14.5.5.1 IIC Protocol Control Registers

In IIC mode, the PCR register bits or bit fields are defined as described in this section.

PCR
Protocol Control Register [IIC Mode]

($3C_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MCLK	ACKIEN	HDEL				SACKDIS	ERRIEN	SRRIEN	ARLIEN	NACKIEN	PCRIEN	RSCRIEN	SCRIEN	STIM	ACK00
rw	rw	rw				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SLAD															
rw															

Field	Bits	Type	Description
SLAD	[15:0]	rw	**Slave Address** This bit field contains the programmed slave address. The corresponding bits in the first received address byte are compared to the bits SLAD[15:9] to check for address match. If SLAD[15:11] = 11110_B, then the second address byte is also compared to SLAD[7:0].
ACK00	16	rw	**Acknowledge 00_H** This bit defines if a slave device should be sensitive to the slave address 00_H. 0_B The slave device is not sensitive to this address. 1_B The slave device is sensitive to this address.

Field	Bits	Type	Description
STIM	17	rw	**Symbol Timing** This bit defines how many time quanta are used in a symbol. 0_B A symbol contains 10 time quanta. The timing is adapted for standard mode (100 kBaud). 1_B A symbol contains 25 time quanta. The timing is adapted for fast mode (400 kBaud).
SCRIEN	18	rw	**Start Condition Received Interrupt Enable** This bit enables the generation of a protocol interrupt if a start condition is detected. 0_B The start condition interrupt is disabled. 1_B The start condition interrupt is enabled.
RSCRIEN	19	rw	**Repeated Start Condition Received Interrupt Enable** This bit enables the generation of a protocol interrupt if a repeated start condition is detected. 0_B The repeated start condition interrupt is disabled. 1_B The repeated start condition interrupt is enabled.
PCRIEN	20	rw	**Stop Condition Received Interrupt Enable** This bit enables the generation of a protocol interrupt if a stop condition is detected. 0_B The stop condition interrupt is disabled. 1_B The stop condition interrupt is enabled.
NACKIEN	21	rw	**Non-Acknowledge Interrupt Enable** This bit enables the generation of a protocol interrupt if a non-acknowledge is detected by a master. 0_B The non-acknowledge interrupt is disabled. 1_B The non-acknowledge interrupt is enabled.
ARLIEN	22	rw	**Arbitration Lost Interrupt Enable** This bit enables the generation of a protocol interrupt if an arbitration lost event is detected. 0_B The arbitration lost interrupt is disabled. 1_B The arbitration lost interrupt is enabled.
SRRIEN	23	rw	**Slave Read Request Interrupt Enable** This bit enables the generation of a protocol interrupt if a slave read request is detected. 0_B The slave read request interrupt is disabled. 1_B The slave read request interrupt is enabled.

Field	Bits	Type	Description
ERRIEN	24	rw	**Error Interrupt Enable** This bit enables the generation of a protocol interrupt if an IIC error condition is detected (indicated by PSR.ERR or PSR.WTDF). 0_B The error interrupt is disabled. 1_B The error interrupt is enabled.
SACKDIS	25	rw	**Slave Acknowledge Disable** This bit disables the generation of an active acknowledge signal for a slave device (active acknowledge = 0 level). Once set by software, it is automatically cleared with each (repeated) start condition. If this bit is set after a byte has been received (indicated by an interrupt) but before the next acknowledge bit has started, the next acknowledge bit will be sent with passive level. This would indicate that the receiver does not accept more bytes. As a result, a minimum of 2 bytes will be received if the first receive interrupt is used to set this bit. 0_B The generation of an active slave acknowledge is enabled (slave acknowledge with 0 level = more bytes can be received). 1_B The generation of an active slave acknowledge is disabled (slave acknowledge with 1 level = reception stopped).
HDEL	[29:26]	rw	**Hardware Delay** This bit field defines the delay used to compensate the internal treatment of the SCL signal (see **Page 14-118**) in order to respect the SDA hold time specified for the IIC protocol.
ACKIEN	30	rw	**Acknowledge Interrupt Enable** This bit enables the generation of a protocol interrupt if an acknowledge is detected by a master. 0_B The acknowledge interrupt is disabled. 1_B The acknowledge interrupt is enabled.
MCLK	31	rw	**Master Clock Enable** This bit enables generation of the master clock MCLK (not directly used for IIC protocol, can be used as general frequency output). 0_B The MCLK generation is disabled and MCLK is 0. 1_B The MCLK generation is enabled.

14.5.5.2 IIC Protocol Status Register

The following PSR status bits or bit fields are available in IIC mode. Please note that the bits in register PSR are not cleared by hardware.

The flags in the PSR register can be cleared by writing a 1 to the corresponding bit position in register PSCR. Writing a 1 to a bit position in PSR sets the corresponding flag, but does not lead to further actions (no interrupt generation). Writing a 0 has no effect. These flags should be cleared by software before enabling a new protocol.

PSR
Protocol Status Register [IIC Mode] (48_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															BRGIF
r															rwh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AIF	RIF	TBIF	TSIF	DLIF	RSIF	ACK	ERR	SRR	ARL	NACK	PCR	RSCR	SCR	WTDF	SLSEL
rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
SLSEL	0	rwh	**Slave Select** This bit indicates that this device has been selected as slave. 0_B The device is not selected as slave. 1_B The device is selected as slave.
WTDF	1	rwh	**Wrong TDF Code Found**[1)] This bit indicates that an unexpected/wrong TDF code has been found. A protocol interrupt can be generated if PCR.ERRIEN = 1. 0_B A wrong TDF code has not been found. 1_B A wrong TDF code has been found.
SCR	2	rwh	**Start Condition Received**[1)] This bit indicates that a start condition has been detected on the IIC bus lines.A protocol interrupt can be generated if PCR.SCRIEN = 1. 0_B A start condition has not yet been detected. 1_B A start condition has been detected.

Field	Bits	Type	Description
RSCR	3	rwh	**Repeated Start Condition Received[1)]** This bit indicates that a repeated start condition has been detected on the IIC bus lines. A protocol interrupt can be generated if PCR.RSCRIEN = 1. 0_B A repeated start condition has not yet been detected. 1_B A repeated start condition has been detected.
PCR	4	rwh	**Stop Condition Received[1)]** This bit indicates that a stop condition has been detected on the IIC bus lines. A protocol interrupt can be generated if PCR.PCRIEN = 1. 0_B A stop condition has not yet been detected. 1_B A stop condition has been detected.
NACK	5	rwh	**Non-Acknowledge Received[1)]** This bit indicates that a non-acknowledge has been received in master mode. This bit is not set in slave mode. A protocol interrupt can be generated if PCR.NACKIEN = 1. 0_B A non-acknowledge has not been received. 1_B A non-acknowledge has been received.
ARL	6	rwh	**Arbitration Lost[1)]** This bit indicates that an arbitration has been lost. A protocol interrupt can be generated if PCR.ARLIEN = 1. 0_B An arbitration has not been lost. 1_B An arbitration has been lost.
SRR	7	rwh	**Slave Read Request[1)]** This bit indicates that a slave read request has been detected. It becomes active to request the first data byte to be made available in the transmit buffer. For further consecutive data bytes, the transmit buffer issues more interrupts. For the end of the transfer, the master transmitter sends a stop condition. A protocol interrupt can be generated if PCR.SRRIEN = 1. 0_B A slave read request has not been detected. 1_B A slave read request has been detected.

Field	Bits	Type	Description
ERR	8	rwh	**Error**[1)] This bit indicates that an IIC error (frame format or TDF code) has been detected. A protocol interrupt can be generated if PCR.ERRIEN = 1. 0_B An IIC error has not been detected. 1_B An IIC error has been detected.
ACK	9	rwh	**Acknowledge Received**[1)] This bit indicates that an acknowledge has been received in master mode. This bit is not set in slave mode. A protocol interrupt can be generated if PCR.ACKIEN = 1. 0_B An acknowledge has not been received. 1_B An acknowledge has been received.
RSIF	10	rwh	**Receiver Start Indication Flag** 0_B A receiver start event has not occurred. 1_B A receiver start event has occurred.
DLIF	11	rwh	**Data Lost Indication Flag** 0_B A data lost event has not occurred. 1_B A data lost event has occurred.
TSIF	12	rwh	**Transmit Shift Indication Flag** 0_B A transmit shift event has not occurred. 1_B A transmit shift event has occurred.
TBIF	13	rwh	**Transmit Buffer Indication Flag** 0_B A transmit buffer event has not occurred. 1_B A transmit buffer event has occurred.
RIF	14	rwh	**Receive Indication Flag** 0_B A receive event has not occurred. 1_B A receive event has occurred.
AIF	15	rwh	**Alternative Receive Indication Flag** 0_B An alternative receive event has not occurred. 1_B An alternative receive event has occurred.
BRGIF	16	rwh	**Baud Rate Generator Indication Flag** 0_B A baud rate generator event has not occurred. 1_B A baud rate generator event has occurred.
0	[31:17]	r	**Reserved** Returns 0 if read; not modified in IIC mode.

1) This status bit can generate a protocol interrupt (see **Page 14-22**). The general interrupt status flags are described in the general interrupt chapter.

14.6 Inter-IC Sound Bus Protocol (IIS)

This chapter describes how the USIC module handles the IIS protocol. This serial protocol can handle reception and transmission of synchronous data frames between a device operating in master mode and a device in slave mode. An IIS connection based on a USIC communication channel supports half-duplex and full-duplex data transfers. The IIS mode is selected by CCR.MODE = 0011_B with CCFG.IIS = 1 (IIS mode is available).

14.6.1 Introduction

The IIS protocol is a synchronous serial communication protocol mainly for audio and infotainment applications **[12]**.

14.6.1.1 Signal Description

A connection between an IIS master and an IIS slave is based on the following signals:

- A shift clock signal SCK, generated by the transfer master. It is permanently generated while an IIS connection is established, also while no valid data bits are transferred.
- A word address signal WA (also named WS), generated by the transfer master. It indicates the beginning of a new data word and the targeted audio channel (e.g. left/right). The word address output signal WA is available on all SELOx outputs if the WA generation is enabled (by PCR.WAGEN = 1 for the transfer master). The WA signal changes synchronously to the falling edges of the shift clock.
- If the transmitter is the IIS master device, it generates a master transmit slave receive data signal. The data changes synchronously to the falling edges of the shift clock.
- If the transmitter is the IIS slave device, it generates a master receive slave transmit data signal. The data changes synchronously to the falling edges of the shift clock.

The transmitter part and the receiver part of the USIC communication channel can be used together to establish a full-duplex data connection between an IIS master and a slave device.

Table 14-19 IIS IO Signals

IIS Mode	Receive Data	Transmit Data	Shift Clock	Word Address
Master	Input DIN0, handled by DX0	Output DOUT0	Output SCLKOUT	Output(s) SELOx
Slave	Input DIN0, handled by DX0	Output DOUT0	Input SCLKIN, handled by DX1	Input SELIN, handled by DX2

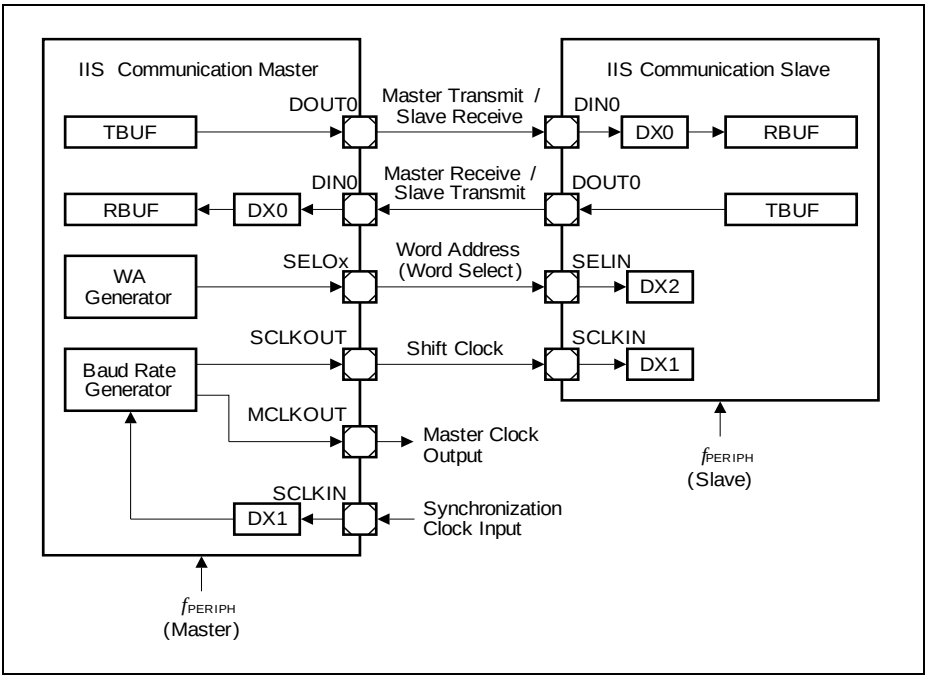


Figure 14-58 IIS Signals

Two additional signals are available for the USIC IIS communication master:

- A master clock output signal MCLKOUT with a fixed phase relation to the shift clock to support oversampling for audio components. It can also be used as master clock output of a communication network with synchronized IIS connections.
- A synchronization clock input SCLKIN for synchronization of the shift clock generation to an external frequency to support audio frequencies that can not be directly derived from the system clock f_{PERIPH} of the communication master. It can be used as master clock input of a communication network with synchronized IIS connections.

14.6.1.2 Protocol Overview

An IIS connection supports transfers for two different data frames via the same data line, e.g. a data frames for the left audio channel and a data frame for the right audio channel. The word address signal WA is used to distinguish between the different data frames. Each data frame can consist of several data words.

In a USIC communication channel, data words are tagged for being transmitted for the left or for the right channel. Also the received data words contain a tag identifying the WA state when the data has been received.

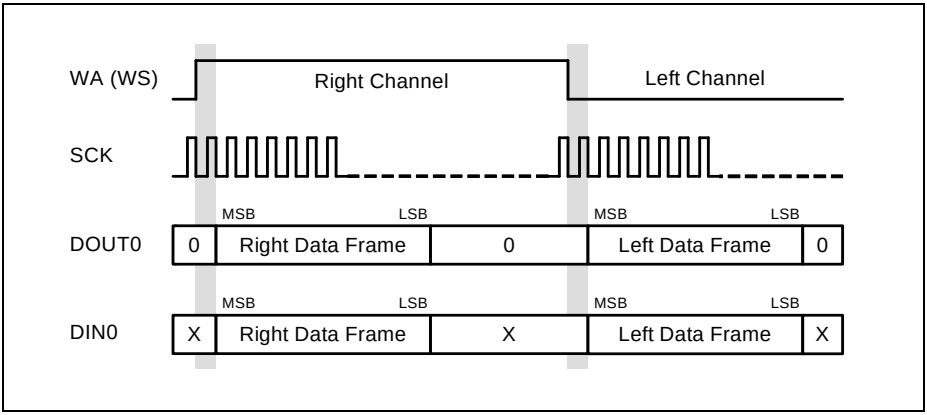


Figure 14-59 Protocol Overview

14.6.1.3 Transfer Delay

The transfer delay feature allows the transfer of data (transmission and reception) with a programmable delay (counted in shift clock periods).

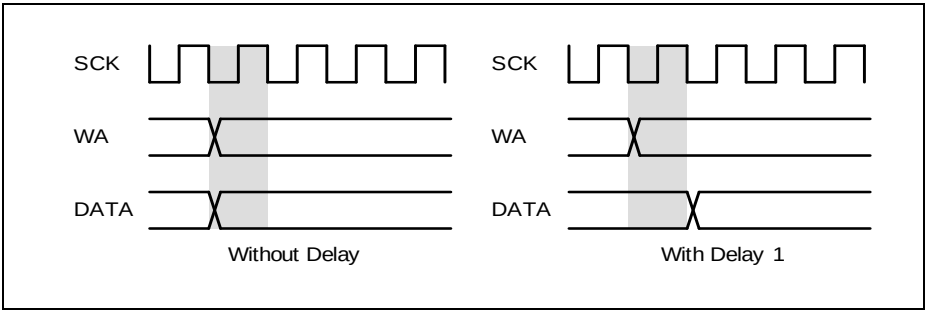


Figure 14-60 Transfer Delay for IIS

14.6.1.4 Connection of External Audio Components

The IIS signals can be used to communicate with external audio devices (such as Codecs) or other audio data sources/destinations.

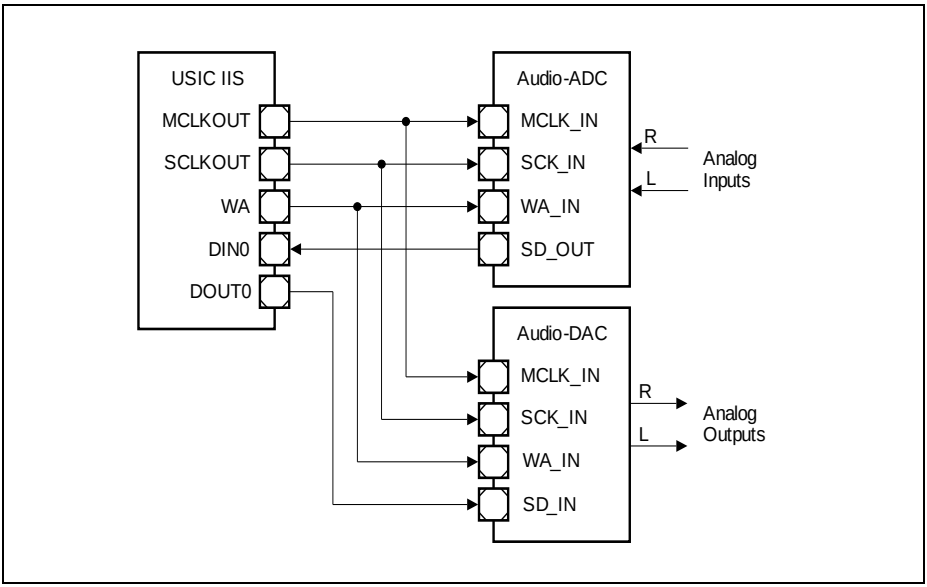


Figure 14-61 Connection of External Audio Devices

In some applications, especially for Audio-ADCs or Audio-DACs, a master clock signal is required with a fixed phase relation to the shift clock signal. The frequency of MCLKOUT is a multiple of the shift frequency SCLKOUT. This factor defines the oversampling factor of the external device (commonly used values: 256 or 384).

14.6.2 Operating the IIS

This chapter contains IIS issues, that are of general interest and not directly linked to master mode or slave mode.

14.6.2.1 Frame Length and Word Length Configuration

After each change of the WA signal, a complete data frame is intended to be transferred (frame length ≤ system word length). The number of data bits transferred after a change of signal WA is defined by SCTR.FLE. A data frame can consist of several data words with a data word length defined by SCTR.WLE. The changes of signal WA define the system word length as the number of SCLK cycles between two changes of WA (number of bits available for the right channel and same number available for the left channel).

If the system word length is longer than the frame length defined by SCTR.FLE, the additional bits are transmitted with passive data level (SCTR.PDL). If the system word

length is smaller than the device frame length, not all LSBs of the transmit data can be transferred.

It is recommended to program bits WLEMD, FLEMD and SELMD in register TCSR to 0.

14.6.2.2 Automatic Shadow Mechanism

The baud rate and shift control setting are internally kept constant while a data frame is transferred by an automatic shadow mechanism. The registers can be programmed all the time with new settings that are taken into account for the next data frame. During a data frame, the applied (shadowed) setting is not changed, although new values have been written after the start of the data frame. The setting is internally “frozen” with the start of each data frame.

Although this shadow mechanism being implemented, it is recommended to change the baud rate and shift control setting only while the IIS protocol is switched off.

14.6.2.3 Mode Control Behavior

In IIS mode, the following kernel modes are supported:

- Run Mode 0/1:
  Behavior as programmed, no impact on data transfers.
- Stop Mode 0/1:
  Bit PCR.WAGEN is internally considered as 0 (the bit itself is not changed). If WAGEN = 1, the WA generation is stopped, but PSR.END is not set. The complete data frame is finished before entering stop mode, including a possible delay due to PCR.TDEL.
  When leaving a stop mode with WAGEN = 1, the WA generation starts from the beginning.

14.6.2.4 Transfer Delay

The transfer delay can be used to synchronize a data transfer to an event (e.g. a change of the WA signal). This event has to be synchronously generated to the falling edge of the shift clock SCK (like the change of the transmit data), because the input signal for the event is directly sampled in the receiver (as a result, the transmitter can use the detection information with its next edge).

Event signals that are asynchronous to the shift clock while the shift clock is running must not be used. In the example in **Figure 14-60**, the event (change of signal WA) is generated by the transfer master and as a result, is synchronous to the shift clock SCK. With the rising edge of SCK, signal WA is sampled and checked for a change. If a change is detected, a transfer delay counter TDC is automatically loaded with its programmable reload value (PCR.TDEL), otherwise it is decremented with each rising edge of SCK until it reaches 0, where it stops. The transfer itself is started if the value of TDC has become 0. This can happen under two conditions:

- TDC is reloaded with a PCR.TDEL = 0 when the event is detected
- TDC has reached 0 while counting down

The transfer delay counter is internal to the IIS protocol pre-processor and can not be observed by software. The transfer delay in SCK cycles is given by PCR.TDEL+1.

In the example in **Figure 14-62**, the reload value PCR.TDEL for TDC is 0. When the samples taken on receiver side show the change of the WA signal, the counter TDC is reloaded. If the reload value is 0, the data transfer starts with 1 shift clock cycle delay compared to the change of WA.

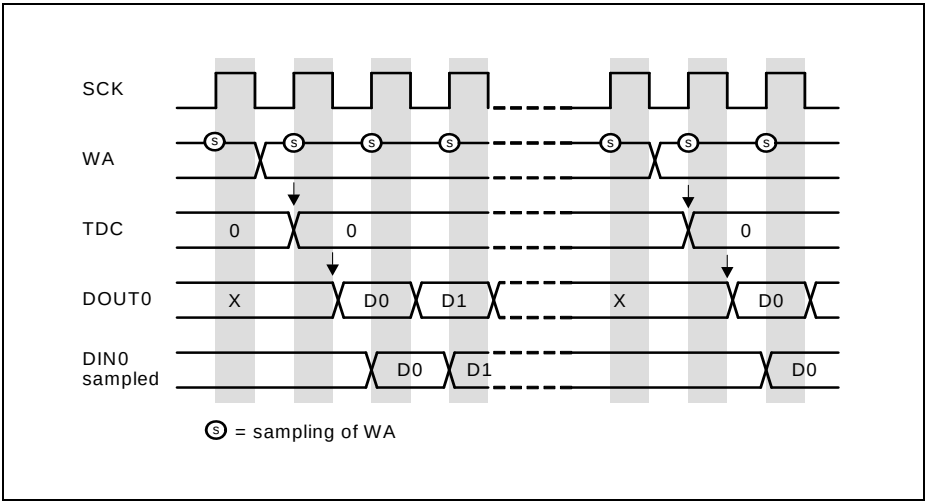


Figure 14-62 Transfer Delay with Delay 1

The ideal case without any transfer delay is shown in **Figure 14-63**. The WA signal changes and the data output value become valid at the same time. This implies that the transmitter “knows” in advance that the event signal will change with the next rising edge of TCLK. This is achieved by delaying the data transmission after the previously detected WA change the system word length minus 1.

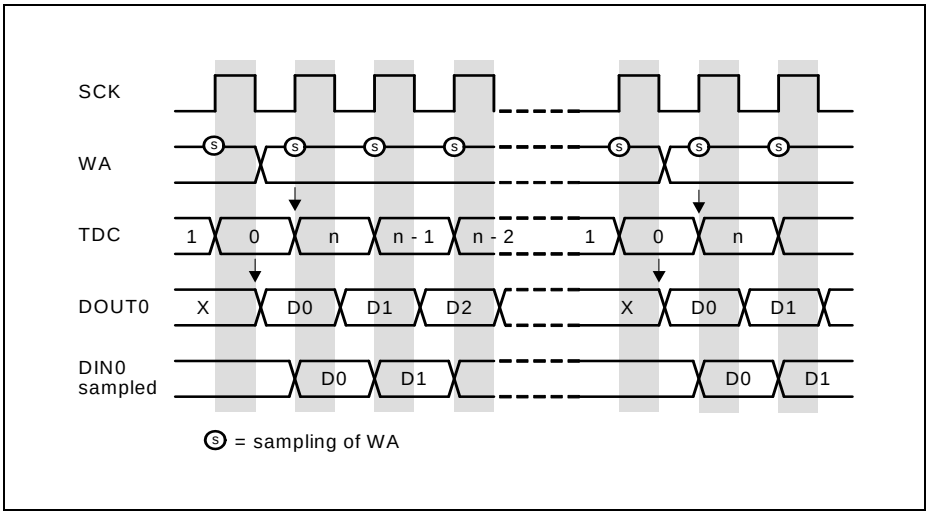


Figure 14-63 No Transfer Delay

If the end of the transfer delay is detected simultaneously to change of WA, the transfer is started and the delay counter is reloaded with PCR.TDEL. This allows to run the USIC as IIS device without any delay. In this case, internally the delay from the previous event elapses just at the moment when a new event occurs. If PCR.TDEL is set to a value bigger than the system word length, no transfer takes place.

14.6.2.5 Parity Mode

Parity generation is not supported in IIS mode and bit field CCR.PM = 00_B has to be programmed.

14.6.2.6 Transfer Mode

In IIS mode, bit field SCTR.TRM = 11_B has to be programmed to allow data transfers. Setting SCTR.TRM = 00_B disables and stops the data transfer immediately.

14.6.2.7 Data Transfer Interrupt Handling

The data transfer interrupts indicate events related to IIS frame handling.

- Transmit buffer interrupt TBI:
  Bit PSR.TBIF is set after the start of first data bit of a data word.
- Transmit shift interrupt TSI:
  Bit PSR.TSIF is set after the start of the last data bit of a data word.

- Receiver start interrupt RSI:
  Bit PSR.RSIF is set after the reception of the first data bit of a data word.
  With this event, bit TCSR.TDV is cleared and new data can be loaded to the transmit buffer.
- Receiver interrupt RI and alternative interrupt AI:
  Bit PSR.RIF is set at after the reception of the last data bit of a data word with WA = 0.
  Bit RBUFSR.SOF indicates whether the received data word has been the first data word of a new data frame.
  Bit PSR.AIF is set at after the reception of the last data bit of a data word with WA = 1.
  Bit RBUFSR.SOF indicates whether the received data word has been the first data word of a new data frame.

14.6.2.8 Baud Rate Generator Interrupt Handling

The baud rate generator interrupt indicate that the capture mode timer has reached its maximum value. With this event, the bit PSR.BRGIF is set.

14.6.2.9 Protocol-Related Argument and Error

In order to distinguish between data words received for the left or the right channel, the IIS protocol pre-processor samples the level of the WA input (just after the WA transition) and propagates it as protocol-related error (although it is not an error, but an indication) to the receive buffer status register at the bit position RBUFSR[9]. This bit position defines if either a standard receive interrupt (if RBUFSR[9] = 0) or an alternative receive interrupt (if RBUFSR[9] = 1) becomes activated when a new data word has been received. Incoming data can be handled by different interrupts or DMA mechanisms for the left and the right channel if the corresponding events are directed to different interrupt nodes. Flag PAR is always 0.

14.6.2.10 Transmit Data Handling

The IIS protocol pre-processor allows to distinguish between the left and the right channel for data transmission. Therefore, bit TCSR.WA indicates on which channel the data in the buffer will be transmitted. If TCSR.WA = 0, the data will be transmitted after a falling edge of WA. If TCSR.WA = 1, the data will be transmitted after a rising edge of WA. The WA value sampled after the WA transition is considered to distinguish between both channels (referring to PSR.WA).

Bit TCSR.WA can be automatically updated by the transmit control information TCI[4] for each data word if TCSR.WAMD = 1. In this case, data written to TBUF[15:0] (or IN[15:0] if a FIFO data buffer is used) is considered as left channel data, whereas data written to TBUF[31:16] (or IN[31:16] if a FIFO data buffer is used) is considered as right channel data.

14.6.2.11 Receive Buffer Handling

If a receive FIFO buffer is available (CCFG.RB = 1) and enabled for data handling (RBCTR.SIZE > 0), it is recommended to set RBCTR.RCIM = 11_B in IIS mode. This leads to an indication that the data word has been the first data word of a new data frame if bit OUTR.RCI[0] = 1, and the channel indication by the sampled WA value is given by OUTR.RCI[4].

The standard receive buffer event and the alternative receive buffer event can be used for the following operation in RCI mode (RBCTR.RNM = 1):

- A standard receive buffer event indicates that a data word can be read from OUTR that belongs to a data frame started when WA = 0.
- An alternative receive buffer event indicates that a data word can be read from OUTR that belongs to a data frame started when WA = 1.

14.6.2.12 Loop-Delay Compensation

The synchronous signaling mechanism of the IIS protocol being similar to the one of the SSC protocol, the closed-loop delay has to be taken into account for the application setup. In IIS mode, loop-delay compensation in master mode is also possible to achieve higher baud rates.

Please refer to the more detailed description in the SSC chapter.

14.6.3 Operating the IIS in Master Mode

In order to operate the IIS in master mode, the following issues have to be considered:

- Select IIS mode:
  It is recommended to configure all parameters of the IIS that do not change during run time while CCR.MODE = 0000_B. Bit field SCTR.TRM = 11_B has to be programmed. The configuration of the input stages has to be done while CCR.MODE = 0000_B to avoid unintended edges of the input signals and the IIS mode can be enabled by CCR.MODE = 0011_B afterwards.
- Pin connection for data transfer:
  Establish a connection of input stage DX0 with the selected receive data input pin (DIN0) with DX0CR.INSW = 1. Configure a transmit data output pin (DOUT0) for a transmitter.
  The data shift unit allowing full-duplex data transfers based on the same WA signal, the values delivered by the DX0 stage are considered as data bits (receive function can not be disabled independently from the transmitter). To receive IIS data, the transmitter does not necessarily need to be configured (no assignment of DOUT0 signal to a pin).
- Baud rate generation:
  The desired baud rate setting has to be selected, comprising the fractional divider

and the baud rate generator. Bit DX1CR.INSW = 0 has to be programmed to use the baud rate generator output SCLK directly as input for the data shift unit.

- Word address WA generation:
  The WA generation has to be enabled by setting PCR.WAGEN = 1 and the programming of the number of shift clock cycles between the changes of WA. Bit DX2CR.INSW = 0 has to be programmed to use the WA generator as input for the data shift unit. Configure WA output pin for signal SELOx if needed.
- Data format configuration:
  The word length, the frame length, and the shift direction have to be set up according to the application requirements by programming the register SCTR. Generally, the MSB is shifted first (SCTR.SDIR = 1).
  Bit TCSR.WAMD can be set to use the transmit control information TCI[4] to distinguish the data words for transmission while WA = 0 or while WA = 1.

Note: The step to enable the alternate output port functions should only be done after the IIS mode is enabled, to avoided unintended spikes on the output.

14.6.3.1 Baud Rate Generation

The baud rate is defined by the frequency of the SCLK signal (one period of f_{SCLK} represents one data bit).

If the fractional divider mode is used to generate f_{PIN}, there can be an uncertainty of one period of f_{PERIPH} for f_{PIN}. This uncertainty does not accumulate over several SCLK cycles. As a consequence, the average frequency is reached, whereas the duty cycle of 50% of the SCLK and MCLK signals can vary by one period of f_{PERIPH}.

In IIS applications, where the phase relation between the optional MCLK output signal and SCLK is not relevant, SCLK can be based on the frequency f_{PIN} (BRG.PPPEN = 0). In the case that a fixed phase relation between the MCLK signal and SCLK is required (e.g. when using MCLK as clock reference for external devices), the additional divider by 2 stage has to be taken into account (BRG.PPPEN = 1). This division is due to the fact that signal MCLK toggles with each cycle of f_{PIN}. Signal SCLK is then based on signal MCLK, see **Figure 14-64**.

The adjustable integer divider factor is defined by bit field BRG.PDIV.

$$f_{SCLK} = \frac{f_{PIN}}{2} \times \frac{1}{PDIV + 1} \quad \text{if PPPEN = 0}$$
$$f_{SCLK} = \frac{f_{PIN}}{2 \times 2} \times \frac{1}{PDIV + 1} \quad \text{if PPPEN = 1} \qquad (14.12)$$

Note: In the IIS protocol, the master (unit generating the shift clock and the WA signal) changes the status of its data and WA output line with the falling edge of SCK. The slave transmitter also has to transmit on falling edges. The sampling of the received data is done with the rising edges of SCLK. The input stage DX1 and the

SCLKOUT have to be programmed to invert the shift clock signal to fit to the internal signals.

14.6.3.2 WA Generation

The word address (or word select) line WA regularly toggles after N cycles of signal SCLK. The time between the changes of WA is called system word length and can be programmed by using the following bit fields.

In IIS master mode, the system word length is defined by:

- BRG.CTQSEL = 10_B
  to base the WA toggling on SCLK
- BRG.PCTQ
  to define the number N of SCLK cycles per system word length
- BRG.DCTQ
  to define the number N of SCLK cycles per system word length

$$N = (PCTQ + 1) \times (DCTQ + 1) \quad (14.13)$$

14.6.3.3 Master Clock Output

The master clock signal MCLK can be generated by the master of the IIS transfer (BRG.PPPEN = 1). It is used especially to connect external Codec devices. It can be configured by bit BRG.MCLKCFG in its polarity to become the output signal MCLKOUT.

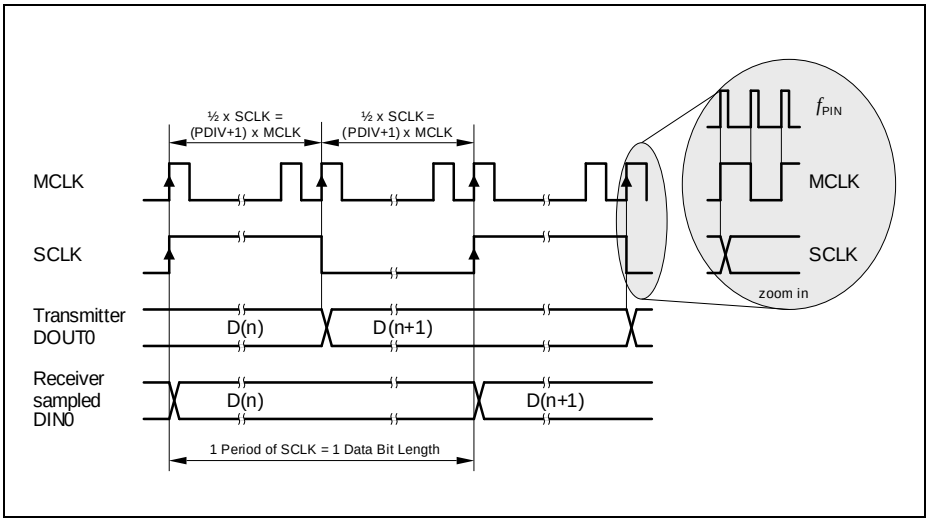


Figure 14-64 MCLK and SCLK for IIS

14.6.3.4 Protocol Interrupt Events

The following protocol-related events are generated in IIS mode and can lead to a protocol interrupt.

Please note that the bits in register PSR are not all automatically cleared by hardware and have to be cleared by software in order to monitor new incoming events.

- WA rising/falling edge events:
  The WA generation block indicates two events that are monitored in register PSR. Flag PSR.WAFE is set with the falling edge, flag PSR.WARE with the rising edge of the WA signal. A protocol interrupt can be generated if PCR.WAFEIEN = 1 for the falling edge, similar for PCR.WAREIEN = 1 for a rising edge.
- WA end event:
  The WA generation block also indicates when it has stopped the WA generation after it has been disabled by writing PCR.WAGEN = 0. A protocol interrupt can be generated if PCR.ENDIEN = 1.
- DX2T event:
  An activation of the trigger signal DX2T is indicated by PSR.DX2TEV = 1 and can generate a protocol interrupt if PCR.DX2TIEN = 1. This event can be evaluated instead of the WA rising/falling events if a delay compensation like in SSC mode (for details, refer to corresponding SSC section) is used.

14.6.4 Operating the IIS in Slave Mode

In order to operate the IIS in slave mode, the following issues have to be considered:

- Select IIS mode:
  It is recommended to configure all parameters of the IIS that do not change during run time while CCR.MODE = 0000_B. Bit field SCTR.TRM = 11_B has to be programmed. The configuration of the input stages has to be done while CCR.MODE = 0000_B to avoid unintended edges of the input signals and the IIS mode can be enabled by CCR.MODE = 0011_B afterwards.
- Pin connection for data transfer:
  Establish a connection of input stage DX0 with the selected receive data input pin (DIN0) with DX0CR.INSW = 1. Configure a transmit data output pin (DOUT0) for a transmitter.
  The data shift unit allowing full-duplex data transfers based on the same WA signal, the values delivered by the DX0 stage are considered as data bits (receive function can not be disabled independently from the transmitter). To receive IIS data, the transmitter does not necessarily need to be configured (no assignment of DOUT0 signal to a pin).
  Note that the step to enable the alternate output port functions should only be done after the IIS mode is enabled, to avoided unintended spikes on the output.

- Pin connection for shift clock:
  Establish a connection of input stage DX1 with the selected shift clock input pin (SCLKIN) with DX1CR.INSW = 1 and with inverted polarity (DX1CR.DPOL = 1).
- Pin connection for WA input:
  Establish a connection of input stage DX2 with the WA input pin (SELIN) with DX2CR.INSW = 1.
- Baud rate generation:
  The baud rate generator is not needed and can be switched off by the fractional divider.
- WA generation:
  The WA generation is not needed and can be switched off (PCR.WAGEN = 0).

14.6.4.1 Protocol Events and Interrupts

The following protocol-related event is generated in IIS mode and can lead to a protocol interrupt.

Please note that the bits in register PSR are not all automatically cleared by hardware and have to be cleared by software in order to monitor new incoming events.

- WA rising/falling/end events:
  The WA generation being switched off, these events are not available.
- DX2T event:
  An activation of the trigger signal DX2T is indicated by PSR.DX2TEV = 1 and can generate a protocol interrupt if PCR.DX2TIEN = 1.

14.6.5 IIS Protocol Registers

In IIS mode, the registers PCR and PSR handle IIS related information.

14.6.5.1 IIS Protocol Control Registers

In IIS mode, the PCR register bits or bit fields are defined as described in this section.

PCR
Protocol Control Register [IIS Mode]

($3C_H$) Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MCLK	0									TDEL					
rw	rw									rw					

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DX2TIEN	0								ENDIEN	WAREIEN	WAFEIEN	0	SELINV	DTEN	WAGEN
rw	rw								rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
WAGEN	0	rw	**WA Generation Enable** This bit enables/disables the generation of word address control output signal WA. 0_B The IIS can be used as slave. The generation of the word address signal is disabled. The output signal WA is 0. The MCLKO signal generation depends on PCR.MCLK. 1_B The IIS can be used as master. The generation of the word address signal is enabled. The signal starts with a 0 after being enabled. The generation of MCLK is enabled, independent of PCR.MCLK. After clearing WAGEN, the USIC module stops the generation of the WA signal within the next 4 WA periods.
DTEN	1	rw	**Data Transfers Enable** This bit enables/disables the transfer of IIS frames as a reaction to changes of the input word address control line WA. 0_B The changes of the WA input signal are ignored and no transfers take place. 1_B Transfers are enabled.

Field	Bits	Type	Description
SELINV	2	rw	**Select Inversion** This bit defines if the polarity of the SELOx outputs in relation to the internally generated word address signal WA. 0_B The SELOx outputs have the same polarity as the WA signal. 1_B The SELOx outputs have the inverted polarity to the WA signal.
WAFEIEN	4	rw	**WA Falling Edge Interrupt Enable** This bit enables/disables the activation of a protocol interrupt when a falling edge of WA has been generated. 0_B A protocol interrupt is not activated if a falling edge of WA is generated. 1_B A protocol interrupt is activated if a falling edge of WA is generated.
WAREIEN	5	rw	**WA Rising Edge Interrupt Enable** This bit enables/disables the activation of a protocol interrupt when a rising edge of WA has been generated. 0_B A protocol interrupt is not activated if a rising edge of WA is generated. 1_B A protocol interrupt is activated if a rising edge of WA is generated.
ENDIEN	6	rw	**END Interrupt Enable** This bit enables/disables the activation of a protocol interrupt when the WA generation stops after clearing PCR.WAGEN (complete system word length is processed before stopping). 0_B A protocol interrupt is not activated. 1_B A protocol interrupt is activated.
DX2TIEN	15	rw	**DX2T Interrupt Enable** This bit enables/disables the generation of a protocol interrupt if the DX2T signal becomes activated (indicated by PSR.DX2TEV = 1). 0_B A protocol interrupt is not generated if DX2T is active. 1_B A protocol interrupt is generated if DX2T is active.

Field	Bits	Type	Description
TDEL	[21:16]	rw	**Transfer Delay** This bit field defines the transfer delay when an event is detected. If bit field TDEL = 0, the additional delay functionality is switched off and a delay of one shift clock cycle is introduced.
MCLK	31	rw	**Master Clock Enable** This bit enables generation of the master clock MCLK (not directly used for IIC protocol, can be used as general frequency output). 0_B The MCLK generation is disabled and MCLK is 0. 1_B The MCLK generation is enabled.
0	3, [14:7], [30:22]	rw	**Reserved** Returns 0 if read; should be written with 0;

14.6.5.2 IIS Protocol Status Register

The following PSR status bits or bit fields are available in IIS mode. Please note that the bits in register PSR are not cleared by hardware.

The flags in the PSR register can be cleared by writing a 1 to the corresponding bit position in register PSCR. Writing a 1 to a bit position in PSR sets the corresponding flag, but does not lead to further actions (no interrupt generation). Writing a 0 has no effect. These flags should be cleared by software before enabling a new protocol.

PSR
Protocol Status Register [IIS Mode] (48_H) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
							0								**BRGIF**
							r								rwh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AIF	**RIF**	**TBIF**	**TSIF**	**DLIF**	**RSIF**		**0**		**END**	**WARE**	**WAFE**	**DX2TEV**	**0**	**DX2S**	**WA**
rwh	rwh	rwh	rwh	rwh	rwh		r		rwh	rwh	rwh	rwh	r	rwh	rwh

Field	Bits	Type	Description
WA	0	rwh	**Word Address** This bit indicates the status of the WA input signal, sampled after a transition of WA has been detected. This information is forwarded to the corresponding bit position RBUFSR[9] to distinguish between data received for the right and the left channel. 0_B WA has been sampled 0. 1_B WA has been sampled 1.
DX2S	1	rwh	**DX2S Status** This bit indicates the current status of the DX2S signal, which is used as word address signal WA. 0_B DX2S is 0. 1_B DX2S is 1.
DX2TEV	3	rwh	**DX2T Event Detected**[1)] This bit indicates that the DX2T signal has been activated. In IIS slave mode, an activation of DX2T generates a protocol interrupt if PCR.DX2TIEN = 1. 0_B The DX2T signal has not been activated. 1_B The DX2T signal has been activated.
WAFE	4	rwh	**WA Falling Edge Event**[1)] This bit indicates that a falling edge of the WA output signal has been generated. This event generates a protocol interrupt if PCR.WAFEIEN = 1. 0_B A WA falling edge has not been generated. 1_B A WA falling edge has been generated.
WARE	5	rwh	**WA Rising Edge Event**[1)] This bit indicates that a rising edge of the WA output signal has been generated. This event generates a protocol interrupt if PCR.WAREIEN = 1. 0_B A WA rising edge has not been generated. 1_B A WA rising edge has been generated.

Field	Bits	Type	Description
END	6	rwh	**WA Generation End**[1)] This bit indicates that the WA generation has ended after clearing PCR.WAGEN. This bit should be cleared by software before clearing WAGEN. 0_B The WA generation has not yet ended (if it is running and WAGEN has been cleared). 1_B The WA generation has ended (if it has been running).
RSIF	10	rwh	**Receiver Start Indication Flag** 0_B A receiver start event has not occurred. 1_B A receiver start event has occurred.
DLIF	11	rwh	**Data Lost Indication Flag** 0_B A data lost event has not occurred. 1_B A data lost event has occurred.
TSIF	12	rwh	**Transmit Shift Indication Flag** 0_B A transmit shift event has not occurred. 1_B A transmit shift event has occurred.
TBIF	13	rwh	**Transmit Buffer Indication Flag** 0_B A transmit buffer event has not occurred. 1_B A transmit buffer event has occurred.
RIF	14	rwh	**Receive Indication Flag** 0_B A receive event has not occurred. 1_B A receive event has occurred.
AIF	15	rwh	**Alternative Receive Indication Flag** 0_B An alternative receive event has not occurred. 1_B An alternative receive event has occurred.
BRGIF	16	rwh	**Baud Rate Generator Indication Flag** 0_B A baud rate generator event has not occurred. 1_B A baud rate generator event has occurred.
0	2, [9:7], [31:17]	r	**Reserved** Returns 0 if read; not modified in IIS mode.

1) This status bit can generate a protocol interrupt (see **Page 14-22**). The general interrupt status flags are described in the general interrupt chapter.

14.7 Service Request Generation

The USIC module provides 6 service request outputs SR[5:0] to be shared between two channels. The service request outputs SR[5:0] are connected to interrupt nodes in the Nested Vectored Interrupt Controller (NVIC). Additionally, the first 2 outputs, SR[1:0], are also connected to the General Purpose DMA (GPDMA) via the DMA Line Router (DLR). For details of the interrupt node and DMA line assignments, refer to the respective chapter in the specification.

Each USIC communication channel can be connected to up to 6 service request handlers (connected to USICx.SR[5:0], though 3 or 4 are normally used, e.g. one for transmission, one for reception, one or two for protocol or error handling, or for the alternative receive events).

14.8 Debug Behaviour

Each USIC communication channel can be pre-configured to enter one of four kernel modes, when the program execution of the CPU is halted by the debugger.

Refer to **Section 14.2.2.2** for details.

14.9 Power, Reset and Clock

The USIC module is located in the core power domain. The module, including all registers other than the bit field KSCFG.SUMCFG, can be reset to its default state by a system reset or a software reset triggered through the setting of corresponding bits in PRSETx registers. The bit field KSCFG.SUMCFG is reset to its default value only by a debug reset.

The USIC module is clocked by the Peripheral Bus clock (f_{PERIPH}). If the module clock is disabled by KSCFG.MODEN = 0, the module cannot be accessed by read or write operations (except register KSCFG that can always be accessed).

14.10 Initialization and System Dependencies

The USIC module is held in reset after a start-up from a system or software reset. Therefore, the application has to apply the following initialization sequence before operating the USIC module:

- Release reset of USIC module by writing a 1 to the USICxRS bit in SCU_PRCLR0 or SCU_PRCLR1 registers
- Enable the module by writing 1s to the MODEN and BPMODEN bits in KSCFG register.

14.11 Registers

Table 14-20 shows all registers which are required for programming a USIC channel, as well as the FIFO buffer. It summarizes the USIC communication channel registers and defines the relative addresses and the reset values.

Please note that all registers can be accessed with any access width (8-bit, 16-bit, 32-bit), independent of the described width.

All USIC registers (except bit field KSCFG.SUMCFG) are always reset by a system reset. Bit field KSCFG.SUMCFG is reset by a debug reset.

Note: The register bits marked "w" always deliver 0 when read. They are used to modify flip-flops in other registers or to trigger internal actions.

Figure 14-65 shows the register types of the USIC module registers and channel registers. In a specific microcontroller, module registers of USIC module "x" are marked by the module prefix "USICx_". Channel registers of USIC module "x" are marked by the channel prefix "USICx_CH0_" and "USICx_CH1_".

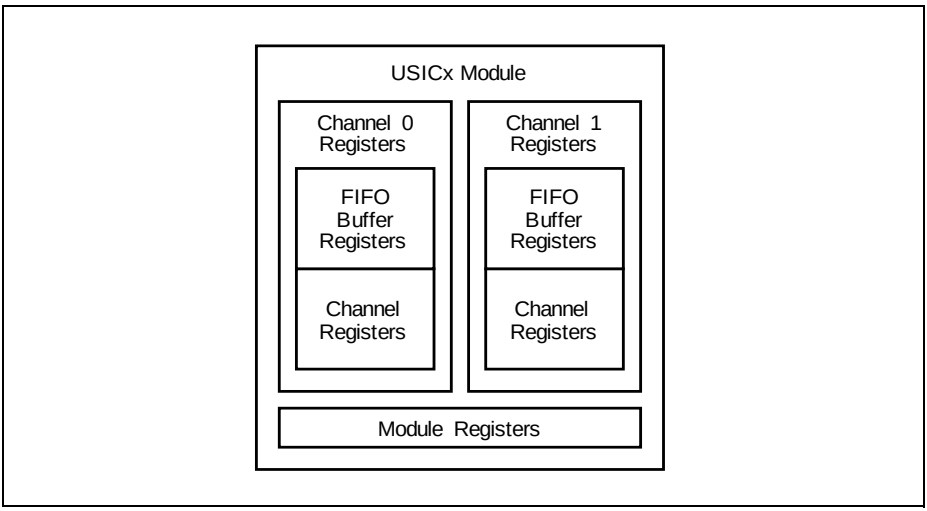


Figure 14-65 USIC Module and Channel Registers

Table 14-20 USIC Kernel-Related and Kernel Registers

Register Short Name	Register Long Name	Offset Addr.	Access Mode		Description see
			Read	Write	
Module Registers[1)]					
ID	Module Identification Register	008_H	U, PV	U, PV	**Page 14-158**

Table 14-20 USIC Kernel-Related and Kernel Registers (cont'd)

Register Short Name	Register Long Name	Offset Addr.	Access Mode		Description see
			Read	Write	
Channel Registers					
–	reserved	000_H	BE	BE	–
CCFG	Channel Configuration Register	004_H	U, PV	U, PV	**Page 14-163**
KSCFG	Kernel State Configuration Register	$00C_H$	U, PV	U, PV	**Page 14-164**
FDR	Fractional Divider Register	010_H	U, PV	PV	**Page 14-177**
BRG	Baud Rate Generator Register	014_H	U, PV	PV	**Page 14-178**
INPR	Interrupt Node Pointer Register	018_H	U, PV	U, PV	**Page 14-167**
DX0CR	Input Control Register 0	$01C_H$	U, PV	U, PV	**Page 14-172**
DX1CR	Input Control Register 1	020_H	U, PV	U, PV	**Page 14-174**
DX2CR	Input Control Register 2	024_H	U, PV	U, PV	**Page 14-172**
DX3CR	Input Control Register 3	028_H	U, PV	U, PV	
DX4CR	Input Control Register 4	$02C_H$	U, PV	U, PV	
DX5CR	Input Control Register 5	030_H	U, PV	U, PV	
SCTR	Shift Control Register	034_H	U, PV	U, PV	**Page 14-182**
TCSR	Transmit Control/Status Register	038_H	U, PV	U, PV	**Page 14-185**
PCR	Protocol Control Register	$03C_H$	U, PV	U, PV	**Page 14-168** 2)
			U, PV	U, PV	**Page 14-66** 3)
			U, PV	U, PV	**Page 14-98** 4)
			U, PV	U, PV	**Page 14-129** 5)
			U, PV	U, PV	**Page 14-148** 6)
CCR	Channel Control Register	040_H	U, PV	PV	**Page 14-159**
CMTR	Capture Mode Timer Register	044_H	U, PV	U, PV	**Page 14-181**

Table 14-20 USIC Kernel-Related and Kernel Registers (cont'd)

Register Short Name	Register Long Name	Offset Addr.	Access Mode		Description see
			Read	Write	
PSR	Protocol Status Register	048_H	U, PV	U, PV	Page 14-169 2)
			U, PV	U, PV	Page 14-70 3)
			U, PV	U, PV	Page 14-102 4)
			U, PV	U, PV	Page 14-132 5)
			U, PV	U, PV	Page 14-150 6)
PSCR	Protocol Status Clear Register	$04C_H$	U, PV	U, PV	Page 14-170
RBUFSR	Receiver Buffer Status Register	050_H	U, PV	U, PV	Page 14-203
RBUF	Receiver Buffer Register	054_H	U, PV	U, PV	Page 14-201
RBUFD	Receiver Buffer Register for Debugger	058_H	U, PV	U, PV	Page 14-202
RBUF0	Receiver Buffer Register 0	$05C_H$	U, PV	U, PV	Page 14-194
RBUF1	Receiver Buffer Register 1	060_H	U, PV	U, PV	Page 14-195
RBUF01SR	Receiver Buffer 01 Status Register	064_H	U, PV	U, PV	Page 14-196
FMR	Flag Modification Register	068_H	U, PV	U, PV	Page 14-192
–	reserved; do not access this location	$06C_H$	U, PV	BE	–
–	reserved	070_H - $07C_H$	BE	BE	–
TBUFx	Transmit Buffer Input Location x (x = 00-31)	080_H + x*4	U, PV	U, PV	Page 14-194
FIFO Buffer Registers					
BYP	Bypass Data Register	100_H	U, PV	U, PV	Page 14-204
BYPCR	Bypass Control Register	104_H	U, PV	U, PV	Page 14-205
TBCTR	Transmit Buffer Control Register	108_H	U, PV	U, PV	Page 14-213
RBCTR	Receive Buffer Control Register	$10C_H$	U, PV	U, PV	Page 14-217
TRBPTR	Transmit/Receive Buffer Pointer Register	110_H	U, PV	U, PV	Page 14-225

Table 14-20 USIC Kernel-Related and Kernel Registers (cont'd)

Register Short Name	Register Long Name	Offset Addr.	Access Mode		Description see
			Read	Write	
TRBSR	Transmit/Receive Buffer Status Register	114_H	U, PV	U, PV	**Page 14-208**
TRBSCR	Transmit/Receive Buffer Status Clear Register	118_H	U, PV	U, PV	**Page 14-212**
OUTR	Receive Buffer Output Register	$11C_H$	U, PV	U, PV	**Page 14-223**
OUTDR	Receive Buffer Output Register for Debugger	120_H	U, PV	U, PV	**Page 14-224**
–	reserved	124_H - $17C_H$	BE	BE	–
INx	Transmit FIFO Buffer Input Location x (x = 00-31)	180_H + x*4	U, PV	U, PV	**Page 14-222**

1) Details of the module identification registers are described in the implementation section (see **Page 14-158**).
2) This page shows the general register layout.
3) This page shows the register layout in ASC mode.
4) This page shows the register layout in SSC mode.
5) This page shows the register layout in IIC mode.
6) This page shows the register layout in IIS mode.

14.11.1 Address Map

The registers of the USIC communication channel are available at the following base addresses. The exact register address is given by the relative address of the register (given in **Table 14-20**) plus the channel base address (given in **Table 14-21**).

Table 14-21 Registers Address Space

Module	Base Address	End Address	Note
USIC0_CH0	40030000_H	$400301FF_H$	–
USIC0_CH1	40030200_H	$400303FF_H$	–
USIC1_CH0	48020000_H	$480201FF_H$	–
USIC1_CH1	48020200_H	$480203FF_H$	–

Table 14-22 FIFO and Reserved Address Space

Module	Base Address	End Address	Note
USIC0	40030400_H	$400307FF_H$	USIC0 RAM area, shared between USIC0_CH0 and USIC0_CH1
reserved	40030800_H	$40033FFF_H$	This address range is reserved
USIC1	48020400_H	$480207FF_H$	USIC1 RAM area, shared between USIC1_CH0 and USIC1_CH1
reserved	48020800_H	$48023FFF_H$	This address range is reserved

14.11.2 Module Identification Registers

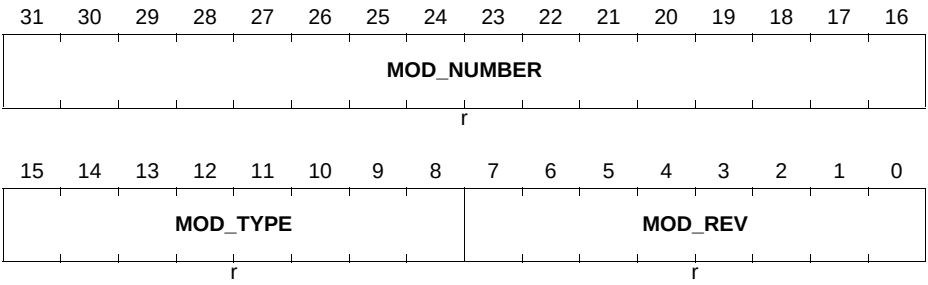
The module identification registers indicate the function and the design step of the USIC modules.

USIC0_ID
Module Identification Register

($4003\ 0008_H$) **Reset Value: $00AA\ C0XX_H$**

USIC1_ID
Module Identification Register

($4802\ 0008_H$) **Reset Value: $00AA\ C0XX_H$**



Field	Bits	Type	Description
MOD_REV	[7:0]	r	**Module Revision Number** MOD_REV defines the revision number. The value of a module revision starts with 01_H (first revision).
MOD_TYPE	[15:8]	r	**Module Type** This bit field is $C0_H$. It defines the module as a 32-bit module.
MOD_NUMBER	[31:16]	r	**Module Number Value** This bit field defines the USIC module identification number ($00AA_H$ = USIC).

14.11.3 Channel Control and Configuration Registers

14.11.3.1 Channel Control Register

The channel control register contains the enable/disable bits for hardware port control and interrupt generation on channel events, the control of the parity generation and the protocol selection of a USIC channel.

FDR can be written only with a privilege mode access.

CCR
Channel Control Register **(40_H)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															BRGIEN
r															rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AIEN	RIEN	TBIEN	TSIEN	DLIEN	RSIEN	PM		HPCEN		0		MODE			
rw	rw	rw	rw	rw	rw	rw		rw		r		rw			

Field	Bits	Type	Description
MODE	[3:0]	rw	**Operating Mode** This bit field selects the protocol for this USIC channel. Selecting a protocol that is not available (see register CCFG) or a reserved combination disables the USIC channel. When switching between two protocols, the USIC channel has to be disabled before selecting a new protocol. In this case, registers PCR and PSR have to be cleared or updated by software. 0_H The USIC channel is disabled. All protocol-related state machines are set to an idle state. 1_H The SSC (SPI) protocol is selected. 2_H The ASC (SCI, UART) protocol is selected. 3_H The IIS protocol is selected. 4_H The IIC protocol is selected. Other bit combinations are reserved.
HPCEN	[7:6]	rw	**Hardware Port Control Enable** This bit enables the hardware port control for the specified set of DX[3:0] and DOUT[3:0] pins. 00_B The hardware port control is disabled. 01_B The hardware port control is enabled for DX0 and DOUT0. 10_B The hardware port control is enabled for DX3, DX0 and DOUT[1:0]. 11_B The hardware port control is enabled for DX0, DX[5:3] and DOUT[3:0]. *Note: The hardware port control feature is useful only for SSC protocols in half-duplex configurations, such as dual- and quad-SSC. For all other protocols HPCEN must always be written with 00_B.*

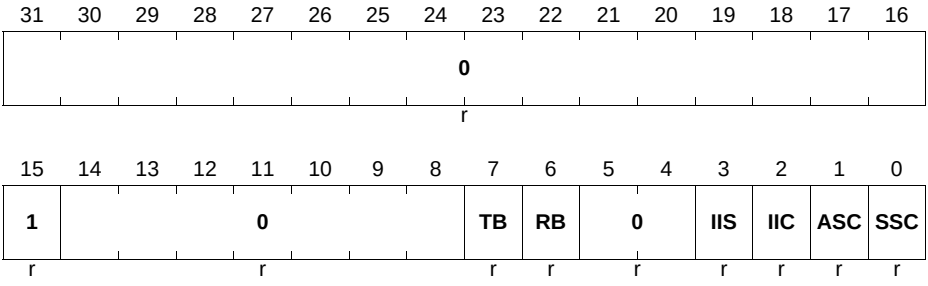
Field	Bits	Type	Description
PM	[9:8]	rw	**Parity Mode** This bit field defines the parity generation of the sampled input values. 00_B The parity generation is disabled. 01_B Reserved 10_B Even parity is selected (parity bit = 1 on odd number of 1s in data, parity bit = 0 on even number of 1s in data). 11_B Odd parity is selected (parity bit = 0 on odd number of 1s in data, parity bit = 1 on even number of 1s in data).
RSIEN	10	rw	**Receiver Start Interrupt Enable** This bit enables the interrupt generation in case of a receiver start event. 0_B The receiver start interrupt is disabled. 1_B The receiver start interrupt is enabled. In case of a receiver start event, the service request output SRx indicated by INPR.TBINP is activated.
DLIEN	11	rw	**Data Lost Interrupt Enable** This bit enables the interrupt generation in case of a data lost event (data received in RBUFx while RDVx = 1). 0_B The data lost interrupt is disabled. 1_B The data lost interrupt is enabled. In case of a data lost event, the service request output SRx indicated by INPR.PINP is activated.
TSIEN	12	rw	**Transmit Shift Interrupt Enable** This bit enables the interrupt generation in case of a transmit shift event. 0_B The transmit shift interrupt is disabled. 1_B The transmit shift interrupt is enabled. In case of a transmit shift interrupt event, the service request output SRx indicated by INPR.TSINP is activated.

Field	Bits	Type	Description
TBIEN	13	rw	**Transmit Buffer Interrupt Enable** This bit enables the interrupt generation in case of a transmit buffer event. 0_B The transmit buffer interrupt is disabled. 1_B The transmit buffer interrupt is enabled. In case of a transmit buffer event, the service request output SRx indicated by INPR.TBINP is activated.
RIEN	14	rw	**Receive Interrupt Enable** This bit enables the interrupt generation in case of a receive event. 0_B The receive interrupt is disabled. 1_B The receive interrupt is enabled. In case of a receive event, the service request output SRx indicated by INPR.RINP is activated.
AIEN	15	rw	**Alternative Receive Interrupt Enable** This bit enables the interrupt generation in case of a alternative receive event. 0_B The alternative receive interrupt is disabled. 1_B The alternative receive interrupt is enabled. In case of an alternative receive event, the service request output SRx indicated by INPR.AINP is activated.
BRGIEN	16	rw	**Baud Rate Generator Interrupt Enable** This bit enables the interrupt generation in case of a baud rate generator event. 0_B The baud rate generator interrupt is disabled. 1_B The baud rate generator interrupt is enabled. In case of a baud rate generator event, the service request output SRx indicated by INPR.PINP is activated.
0	[5:4], [31:17]	r	**Reserved** Read as 0; should be written with 0.

14.11.3.2 Channel Configuration Register

The channel configuration register contains indicates the functionality that is available in the USIC channel.

CCFG
Channel Configuration Register (04_H) Reset Value: 0000 80CF$_H$



Field	Bits	Type	Description
SSC	0	r	**SSC Protocol Available** This bit indicates if the SSC protocol is available. 0_B The SSC protocol is not available. 1_B The SSC protocol is available.
ASC	1	r	**ASC Protocol Available** This bit indicates if the ASC protocol is available. 0_B The ASC protocol is not available. 1_B The ASC protocol is available.
IIC	2	r	**IIC Protocol Available** This bit indicates if the IIC functionality is available. 0_B The IIC protocol is not available. 1_B The IIC protocol is available.
IIS	3	r	**IIS Protocol Available** This bit indicates if the IIS protocol is available. 0_B The IIS protocol is not available. 1_B The IIS protocol is available.
RB	6	r	**Receive FIFO Buffer Available** This bit indicates if an additional receive FIFO buffer is available. 0_B A receive FIFO buffer is not available. 1_B A receive FIFO buffer is available.

Field	Bits	Type	Description
TB	7	r	**Transmit FIFO Buffer Available** This bit indicates if an additional transmit FIFO buffer is available. 0_B A transmit FIFO buffer is not available. 1_B A transmit FIFO buffer is available.
1	15	r	**Reserved** Read as 1; should be written with 1.
0	[5:4], [14:8], [31:16]	r	**Reserved** Read as 0; should be written with 0.

14.11.3.3 Kernel State Configuration Register

The kernel state configuration register KSCFG allows the selection of the desired kernel modes for the different device operating modes.

KSCFG
Kernel State Configuration Register ($0C_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				BPSUM	0	SUMCFG		BPNOM	0	NOMCFG		0		BPMODEN	MODEN
r				w	r	rw		w	r	rw		r		w	rw

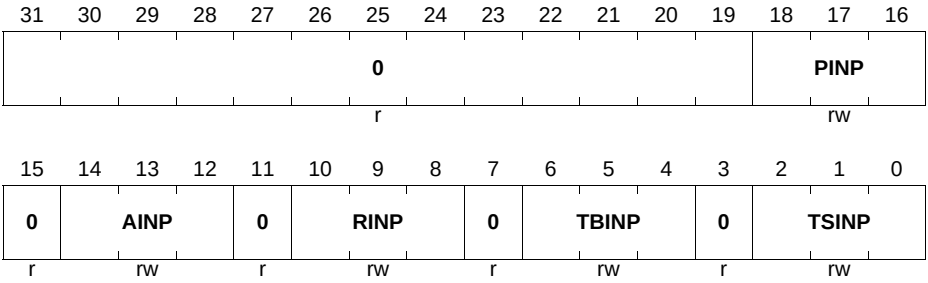
Field	Bits	Type	Description
MODEN	0	rw	**Module Enable** This bit enables the module kernel clock and the module functionality. 0_B The module is switched off immediately (without respecting a stop condition). It does not react on mode control actions and the module clock is switched off. The module does not react on read accesses and ignores write accesses (except to KSCFG). 1_B The module is switched on and can operate. After writing 1 to MODEN, it is recommended to read register KSCFG to avoid pipeline effects in the control block before accessing other USIC registers.
BPMODEN	1	w	**Bit Protection for MODEN** This bit enables the write access to the bit MODEN. It always reads 0. 0_B MODEN is not changed. 1_B MODEN is updated with the written value.
NOMCFG	[5:4]	rw	**Normal Operation Mode Configuration** This bit field defines the kernel mode applied in normal operation mode. 00_B Run mode 0 is selected. 01_B Run mode 1 is selected. 10_B Stop mode 0 is selected. 11_B Stop mode 1 is selected.
BPNOM	7	w	**Bit Protection for NOMCFG** This bit enables the write access to the bit field NOMCFG. It always reads 0. 0_B NOMCFG is not changed. 1_B NOMCFG is updated with the written value.
SUMCFG	[9:8]	rw	**Suspend Mode Configuration** This bit field defines the kernel mode applied in suspend mode. Coding like NOMCFG.

Field	Bits	Type	Description
BPSUM	11	w	**Bit Protection for SUMCFG** This bit enables the write access to the bit field SUMCFG. It always reads 0. 0_B SUMCFG is not changed. 1_B SUMCFG is updated with the written value.
0	[3:2], 6, 10, [31:12]	r	**Reserved** Read as 0; should be written with 0. Bit 2 can read as 1 after BootROM exit (but can be ignored).

14.11.3.4 Interrupt Node Pointer Register

The interrupt node pointer register defines the service request output SRx that is activated if the corresponding event occurs and interrupt generation is enabled.

INPR
Interrupt Node Pointer Register (18_H) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
TSINP	[2:0]	rw	**Transmit Shift Interrupt Node Pointer** This bit field defines which service request output SRx becomes activated in case of a transmit shift interrupt. 000_B Output SR0 becomes activated. 001_B Output SR1 becomes activated. 010_B Output SR2 becomes activated. 011_B Output SR3 becomes activated. 100_B Output SR4 becomes activated. 101_B Output SR5 becomes activated. *Note: All other settings of the bit field are reserved.*
TBINP	[6:4]	rw	**Transmit Buffer Interrupt Node Pointer** This bit field defines which service request output SRx will be activated in case of a transmit buffer interrupt or a receive start interrupt. Coding like TSINP.
RINP	[10:8]	rw	**Receive Interrupt Node Pointer** This bit field defines which service request output SRx will be activated in case of a receive interrupt. Coding like TSINP.

Field	Bits	Type	Description
AINP	[14:12]	rw	**Alternative Receive Interrupt Node Pointer** This bit field defines which service request output SRx will be activated in case of a alternative receive interrupt. Coding like TSINP.
PINP	[18:16]	rw	**Protocol Interrupt Node Pointer** This bit field defines which service request output SRx becomes activated in case of a protocol interrupt. Coding like TSINP.
0	3, 7, 11, 15, [31:19]	r	**Reserved** Read as 0; should be written with 0.

14.11.4 Protocol Related Registers

14.11.4.1 Protocol Control Registers

The bits in the protocol control register define protocol-specific functions. They have to be configured by software before enabling a new protocol. Only the bits used for the selected protocol are taken into account, whereas the other bit positions always read as 0. The protocol-specific meaning is described in the related protocol section.

PCR
Protocol Control Register **($3C_H$)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CTR 31	**CTR 30**	**CTR 29**	**CTR 28**	**CTR 27**	**CTR 26**	**CTR 25**	**CTR 24**	**CTR 23**	**CTR 22**	**CTR 21**	**CTR 20**	**CTR 19**	**CTR 18**	**CTR 17**	**CTR 16**
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CTR 15	**CTR 14**	**CTR 13**	**CTR 12**	**CTR 11**	**CTR 10**	**CTR 9**	**CTR 8**	**CTR 7**	**CTR 6**	**CTR 5**	**CTR 4**	**CTR 3**	**CTR 2**	**CTR 1**	**CTR 0**
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
CTRx (x = 0-31)	x	rw	**Protocol Control Bit x** This bit is a protocol control bit.

14.11.4.2 Protocol Status Register

The flags in the protocol status register can be cleared by writing a 1 to the corresponding bit position in register PSCR. Writing a 1 to a bit position in PSR sets the corresponding flag, but does not lead to further actions (no interrupt generation). Writing a 0 has no effect. These flags should be cleared by software before enabling a new protocol. The protocol-specific meaning is described in the related protocol section.

PSR
Protocol Status Register **(48_H)** **Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
							0								**BRGIF**
							r								rwh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AIF	**RIF**	**TBIF**	**TSIF**	**DLIF**	**RSIF**	**ST9**	**ST8**	**ST7**	**ST6**	**ST5**	**ST4**	**ST3**	**ST2**	**ST1**	**ST0**
rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
STx (x = 0-9)	x	rwh	**Protocol Status Flag x** See protocol specific description.
RSIF	10	rwh	**Receiver Start Indication Flag** 0_B A receiver start event has not occurred. 1_B A receiver start event has occurred.
DLIF	11	rwh	**Data Lost Indication Flag** 0_B A data lost event has not occurred. 1_B A data lost event has occurred.
TSIF	12	rwh	**Transmit Shift Indication Flag** 0_B A transmit shift event has not occurred. 1_B A transmit shift event has occurred.
TBIF	13	rwh	**Transmit Buffer Indication Flag** 0_B A transmit buffer event has not occurred. 1_B A transmit buffer event has occurred.
RIF	14	rwh	**Receive Indication Flag** 0_B A receive event has not occurred. 1_B A receive event has occurred.

Field	Bits	Type	Description
AIF	15	rwh	**Alternative Receive Indication Flag** 0_B An alternative receive event has not occurred. 1_B An alternative receive event has occurred.
BRGIF	16	rwh	**Baud Rate Generator Indication Flag** 0_B A baud rate generator event has not occurred. 1_B A baud rate generator event has occurred.
0	[31:17]	r	**Reserved;** read as 0; should be written with 0;

14.11.4.3 Protocol Status Clear Register

Read accesses to this register always deliver 0 at all bit positions.

PSCR
Protocol Status Clear Register **($4C_H$)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															CBRGIF
r															w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CAIF	CRIF	CTBIF	CTSIF	CDLIF	CRSIF	CST9	CST8	CST7	CST6	CST5	CST4	CST3	CST2	CST1	CST0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Field	Bits	Type	Description
CSTx (x = 0-9)	x	w	**Clear Status Flag x in PSR** 0_B No action 1_B Flag PSR.STx is cleared.
CRSIF	10	w	**Clear Receiver Start Indication Flag** 0_B No action 1_B Flag PSR.RSIF is cleared.
CDLIF	11	w	**Clear Data Lost Indication Flag** 0_B No action 1_B Flag PSR.DLIF is cleared.

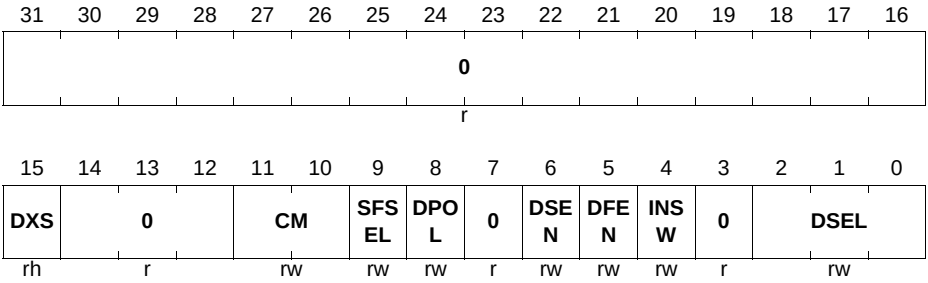
Field	Bits	Type	Description
CTSIF	12	w	**Clear Transmit Shift Indication Flag** 0_B No action 1_B Flag PSR.TSIF is cleared.
CTBIF	13	w	**Clear Transmit Buffer Indication Flag** 0_B No action 1_B Flag PSR.TBIF is cleared.
CRIF	14	w	**Clear Receive Indication Flag** 0_B No action 1_B Flag PSR.RIF is cleared.
CAIF	15	w	**Clear Alternative Receive Indication Flag** 0_B No action 1_B Flag PSR.AIF is cleared.
CBRGIF	16	w	**Clear Baud Rate Generator Indication Flag** 0_B No action 1_B Flag PSR.BRGIF is cleared.
0	[31:17]	r	**Reserved;** read as 0; should be written with 0;

14.11.5 Input Stage Register

14.11.5.1 Input Control Registers

The input control registers contain the bits to define the characteristics of the input stages (input stage DX0 is controlled by register DX0CR, etc.).

DX0CR
Input Control Register 0 **($1C_H$)** **Reset Value: 0000 0000$_H$**
DX2CR
Input Control Register 2 **(24_H)** **Reset Value: 0000 0000$_H$**
DX3CR
Input Control Register 3 **(28_H)** **Reset Value: 0000 0000$_H$**
DX4CR
Input Control Register 4 **($2C_H$)** **Reset Value: 0000 0000$_H$**
DX5CR
Input Control Register 5 **(30_H)** **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
DSEL	[2:0]	rw	**Data Selection for Input Signal** This bit field defines the input data signal for the corresponding input line for protocol pre-processor. The selection can be made from the input vector DXn[G:A]. 000_B The data input DXnA is selected. 001_B The data input DXnB is selected. 010_B The data input DXnC is selected. 011_B The data input DXnD is selected. 100_B The data input DXnE is selected. 101_B The data input DXnF is selected. 110_B The data input DXnG is selected. 111_B The data input is always 1.

Field	Bits	Type	Description
INSW	4	rw	**Input Switch** This bit defines if the data shift unit input is derived from the input data path DXn or from the selected protocol pre-processors. 0_B The input of the data shift unit is controlled by the protocol pre-processor. 1_B The input of the data shift unit is connected to the selected data input line. This setting is used if the signals are directly derived from an input pin without treatment by the protocol pre-processor.
DFEN	5	rw	**Digital Filter Enable** This bit enables/disables the digital filter for signal DXnS. 0_B The input signal is not digitally filtered. 1_B The input signal is digitally filtered.
DSEN	6	rw	**Data Synchronization Enable** This bit selects if the asynchronous input signal or the synchronized (and optionally filtered) signal DXnS can be used as input for the data shift unit. 0_B The un-synchronized signal can be taken as input for the data shift unit. 1_B The synchronized signal can be taken as input for the data shift unit.
DPOL	8	rw	**Data Polarity for DXn** This bit defines the signal polarity of the input signal. 0_B The input signal is not inverted. 1_B The input signal is inverted.
SFSEL	9	rw	**Sampling Frequency Selection** This bit defines the sampling frequency of the digital filter for the synchronized signal DXnS. 0_B The sampling frequency is f_{PERIPH}. 1_B The sampling frequency is f_{FD}.

Field	Bits	Type	Description
CM	[11:10]	rw	**Combination Mode** This bit field selects which edge of the synchronized (and optionally filtered) signal DXnS actives the trigger output DXnT of the input stage. 00_B The trigger activation is disabled. 01_B A rising edge activates DXnT. 10_B A falling edge activates DXnT. 11_B Both edges activate DXnT.
DXS	15	rh	**Synchronized Data Value** This bit indicates the value of the synchronized (and optionally filtered) input signal. 0_B The current value of DXnS is 0. 1_B The current value of DXnS is 1.
0	3, 7, [14:12], [31:16]	r	**Reserved** Read as 0; should be written with 0.

DX1CR
Input Control Register 1 **(20_H)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DXS	**0**			**CM**		**SFSEL**	**DPOL**	**0**	**DSEN**	**DFEN**	**INSW**	**DCEN**	**DSEL**		
rh	r			rw		rw	rw	r	rw	rw	rw	rw	rw		

Field	Bits	Type	Description
DSEL	[2:0]	rw	**Data Selection for Input Signal** This bit field defines the input data signal for the corresponding input line for protocol pre-processor. The selection can be made from the input vector DX1[G:A]. 000_B The data input DX1A is selected. 001_B The data input DX1B is selected. 010_B The data input DX1C is selected. 011_B The data input DX1D is selected. 100_B The data input DX1E is selected. 101_B The data input DX1F is selected. 110_B The data input DX1G is selected. 111_B The data input is always 1.
DCEN	3	rw	**Delay Compensation Enable** This bit selects if the receive shift clock is controlled by INSW or derived from the input data path DX1. 0_B The receive shift clock is dependent on INSW selection. 1_B The receive shift clock is connected to the selected data input line. This setting is used if delay compensation is required in SSC and IIS protocols, else DCEN should always be 0.
INSW	4	rw	**Input Switch** This bit defines if the data shift unit input is derived from the input data path DX1 or from the selected protocol pre-processors. 0_B The input of the data shift unit is controlled by the protocol pre-processor. 1_B The input of the data shift unit is connected to the selected data input line. This setting is used if the signals are directly derived from an input pin without treatment by the protocol pre-processor.
DFEN	5	rw	**Digital Filter Enable** This bit enables/disables the digital filter for signal DX1S. 0_B The input signal is not digitally filtered. 1_B The input signal is digitally filtered.

Field	Bits	Type	Description
DSEN	6	rw	**Data Synchronization Enable** This bit selects if the asynchronous input signal or the synchronized (and optionally filtered) signal DX1S can be used as input for the data shift unit. 0_B The un-synchronized signal can be taken as input for the data shift unit. 1_B The synchronized signal can be taken as input for the data shift unit.
DPOL	8	rw	**Data Polarity for DXn** This bit defines the signal polarity of the input signal. 0_B The input signal is not inverted. 1_B The input signal is inverted.
SFSEL	9	rw	**Sampling Frequency Selection** This bit defines the sampling frequency of the digital filter for the synchronized signal DX1S. 0_B The sampling frequency is f_{PERIPH}. 1_B The sampling frequency is f_{FD}.
CM	[11:10]	rw	**Combination Mode** This bit field selects which edge of the synchronized (and optionally filtered) signal DX1S actives the trigger output DX1T of the input stage. 00_B The trigger activation is disabled. 01_B A rising edge activates DX1T. 10_B A falling edge activates DX1T. 11_B Both edges activate DX1T.
DXS	15	rh	**Synchronized Data Value** This bit indicates the value of the synchronized (and optionally filtered) input signal. 0_B The current value of DX1S is 0. 1_B The current value of DX1S is 1.
0	7, [14:12], [31:16]	r	**Reserved** Read as 0; should be written with 0.

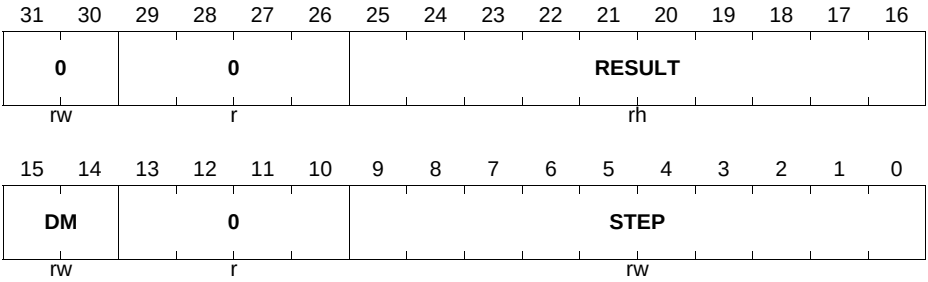
14.11.6 Baud Rate Generator Registers

14.11.6.1 Fractional Divider Register

The fractional divider register FDR allows the generation of the internal frequency f_{FD}, that is derived from the system clock f_{PERIPH}.

FDR can be written only with a privilege mode access.

FDR
Fractional Divider Register **(10_H)** **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
STEP	[9:0]	rw	**Step Value** In normal divider mode STEP contains the reload value for RESULT after RESULT has reached 3FF$_H$. In fractional divider mode STEP defines the value added to RESULT with each input clock cycle.
DM	[15:14]	rw	**Divider Mode** This bit fields defines the functionality of the fractional divider block. 00_B The divider is switched off, f_{FD} = 0. 01_B Normal divider mode selected. 10_B Fractional divider mode selected. 11_B The divider is switched off, f_{FD} = 0.

Field	Bits	Type	Description
RESULT	[25:16]	rh	**Result Value** In normal divider mode this bit field is updated with f_{PERIPH} according to: RESULT = RESULT + 1 In fractional divider mode this bit field is updated with f_{PERIPH} according to: RESULT = RESULT + STEP If bit field DM is written with 01_B or 10_B, RESULT is loaded with a start value of $3FF_H$.
0	[31:30]	rw	**Reserved for Future Use** Must be written with 0 to allow correct fractional divider operation.
0	[13:10], [29:26]	r	**Reserved** Read as 0; should be written with 0.

14.11.6.2 Baud Rate Generator Register

The protocol-related counters for baud rate generation and timing measurement are controlled by the register BRG.

FDR can be written only with a privilege mode access.

BRG
Baud Rate Generator Register (14_H) Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SCLKCFG		MCLKCFG	SCLKOSEL	0		PDIV									
rw		rw	rw	r		rw									

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	DCTQ					PCTQ		CTQSEL		0	PPPEN	TMEN	0	CLKSEL	
r	rw					rw		rw		r	rw	rw	r	rw	

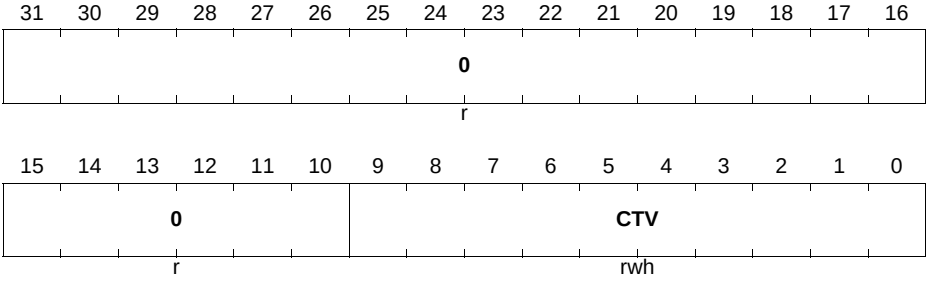
Field	Bits	Type	Description
CLKSEL	[1:0]	rw	**Clock Selection** This bit field defines the input frequency f_{PIN} 00_B The fractional divider frequency f_{FD} is selected. 01_B Reserved, no action 10_B The trigger signal DX1T defines f_{PIN}. Signal MCLK toggles with f_{PIN}. 11_B Signal MCLK corresponds to the DX1S signal and the frequency f_{PIN} is derived from the rising edges of DX1S.
TMEN	3	rw	**Timing Measurement Enable** This bit enables the timing measurement of the capture mode timer. 0_B Timing measurement is disabled: The trigger signals DX0T and DX1T are ignored. 1_B Timing measurement is enabled: The 10-bit counter is incremented by 1 with f_{PPP} and stops counting when reaching its maximum value. If one of the trigger signals DX0T or DX1T become active, the counter value is captured into bit field CTV, the counter is cleared and a transmit shift event is generated.
PPPEN	4	rw	**Enable 2:1 Divider for f_{PPP}** This bit defines the input frequency f_{PPP}. 0_B The 2:1 divider for f_{PPP} is disabled. $f_{PPP} = f_{PIN}$ 1_B The 2:1 divider for f_{PPP} is enabled. $f_{PPP} = f_{MCLK} = f_{PIN} / 2$.
CTQSEL	[7:6]	rw	**Input Selection for CTQ** This bit defines the length of a time quantum for the protocol pre-processor. 00_B $f_{CTQIN} = f_{PDIV}$ 01_B $f_{CTQIN} = f_{PPP}$ 10_B $f_{CTQIN} = f_{SCLK}$ 11_B $f_{CTQIN} = f_{MCLK}$

Field	Bits	Type	Description
PCTQ	[9:8]	rw	**Pre-Divider for Time Quanta Counter** This bit field defines length of a time quantum tq for the time quanta counter in the protocol pre-processor. t_Q = (PCTQ + 1) / f_{CTQIN}
DCTQ	[14:10]	rw	**Denominator for Time Quanta Counter** This bit field defines the number of time quanta t_q taken into account by the time quanta counter in the protocol pre-processor.
PDIV	[25:16]	rw	**Divider Mode: Divider Factor to Generate f_{PDIV}** This bit field defines the ratio between the input frequency f_{PPP} and the divider frequency f_{PDIV}.
SCLKOSEL	28	rw	**Shift Clock Output Select** This bit field selects the input source for the SCLKOUT signal. 0_B SCLK from the baud rate generator is selected as the SCLKOUT input source. 1_B The transmit shift clock from DX1 input stage is selected as the SCLKOUT input source. *Note: The setting SCLKOSEL = 1 is used only when complete closed loop delay compensation is required for a slave SSC/IIS. The default setting of SCLKOSEL = 0 should be always used for all other cases.*
MCLKCFG	29	rw	**Master Clock Configuration** This bit field defines the level of the passive phase of the MCLKOUT signal. 0_B The passive level is 0. 1_B The passive level is 1.
SCLKCFG	[31:30]	rw	**Shift Clock Output Configuration** This bit field defines the level of the passive phase of the SCLKOUT signal and enables/disables a delay of half of a SCLK period. 00_B The passive level is 0 and the delay is disabled. 01_B The passive level is 1 and the delay is disabled. 10_B The passive level is 0 and the delay is enabled. 11_B The passive level is 1 and the delay is enabled.
0	2, 5, 15, [27:26]	r	**Reserved** Read as 0; should be written with 0.

14.11.6.3 Capture Mode Timer Register

The captured timer value is provided by the register CMTR.

CMTR
Capture Mode Timer Register **(44_H)** **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
CTV	[9:0]	rwh	**Captured Timer Value** The value of the counter is captured into this bit field if one of the trigger signals DX0T or DX1T are activated by the corresponding input stage.
0	[31:10]	r	**Reserved** Read as 0; should be written with 0.

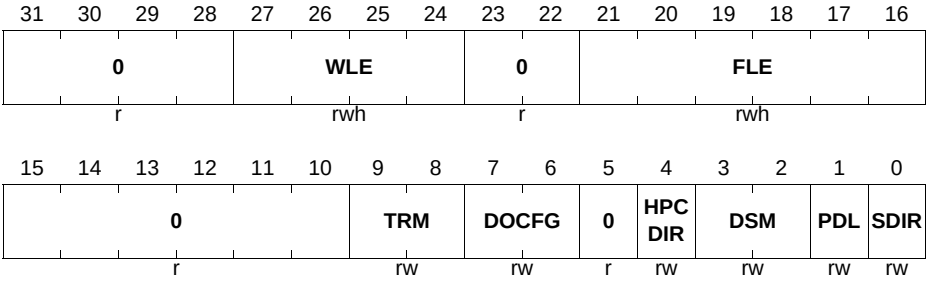
14.11.7 Transfer Control and Status Registers

14.11.7.1 Shift Control Register

The data shift unit is controlled by the register SCTR. The values in this register are applied for data transmission and reception.

Please note that the shift control settings SDIR, WLE, FLE, DSM and HPCDIR are shared between transmitter and receiver. They are internally “frozen” for a each data word transfer in the transmitter with the first transmit shift clock edge and with the first receive shift clock edge in the receiver. The software has to take care that updates of these bit fields by software are done coherently (e.g. refer to the receiver start event indication PSR.RSIF).

SCTR
Shift Control Register **(34_H)** **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
SDIR	0	rw	**Shift Direction** This bit defines the shift direction of the data words for transmission and reception. 0_B Shift LSB first. The first data bit of a data word is located at bit position 0. 1_B Shift MSB first. The first data bit of a data word is located at the bit position given by bit field SCTR.WLE.
PDL	1	rw	**Passive Data Level** This bit defines the output level at the shift data output signal when no data is available for transmission. The PDL level is output with the first relevant transmit shift clock edge of a data word. 0_B The passive data level is 0. 1_B The passive data level is 1.

Field	Bits	Type	Description
DSM	[3:2]	rw	**Data Shift Mode** This bit field describes how the receive and transmit data is shifted in and out. 00_B Receive and transmit data is shifted in and out one bit at a time through DX0 and DOUT0. 01_B Reserved. 10_B Receive and transmit data is shifted in and out two bits at a time through two input stages (DX0 and DX3) and DOUT[1:0] respectively. 11_B Receive and transmit data is shifted in and out four bits at a time through four input stages (DX0, DX[5:3]) and DOUT[3:0] respectively. *Note: Dual- and Quad-output modes are used only by the SSC protocol. For all other protocols DSM must always be written with 00_B.*
HPCDIR	4	rw	**Port Control Direction** This bit defines the direction of the port pin(s) which allows hardware pin control (CCR.PCEN = 1). 0_B The pin(s) with hardware pin control enabled are selected to be in input mode. 1_B The pin(s) with hardware pin control enabled are selected to be in output mode.
DOCFG	[7:6]	rw	**Data Output Configuration** This bit defines the relation between the internal shift data value and the data output signal DOUTx. $X0_B$ DOUTx = shift data value $X1_B$ DOUTx = inverted shift data value

Field	Bits	Type	Description
TRM	[9:8]	rw	**Transmission Mode** This bit field describes how the shift control signal is interpreted by the DSU. Data transfers are only possible while the shift control signal is active. 00_B The shift control signal is considered as inactive and data frame transfers are not possible. 01_B The shift control signal is considered active if it is at 1-level. This is the setting to be programmed to allow data transfers. 10_B The shift control signal is considered active if it is at 0-level. It is recommended to avoid this setting and to use the inversion in the DX2 stage in case of a low-active signal. 11_B The shift control signal is considered active without referring to the actual signal level. Data frame transfer is possible after each edge of the signal.
FLE	[21:16]	rwh	**Frame Length** This bit field defines how many bits are transferred within a data frame. A data frame can consist of several concatenated data words. If TCSR.FLEMD = 1, the value can be updated automatically by the data handler.
WLE	[27:24]	rwh	**Word Length** This bit field defines the data word length (amount of bits that are transferred in each data word) for reception and transmission. The data word is always right-aligned in the data buffer at the bit positions [WLE down to 0]. If TCSR.WLEMD = 1, the value can be updated automatically by the data handler. 0_H The data word contains 1 data bit located at bit position 0. 1_H The data word contains 2 data bits located at bit positions [1:0]. ... E_H The data word contains 15 data bits located at bit positions [14:0]. F_H The data word contains 16 data bits located at bit positions [15:0].

Field	Bits	Type	Description
0	5, [15:10], [23:22], [31:28]	r	**Reserved** Read as 0; should be written with 0.

14.11.7.2 Transmission Control and Status Register

The data transmission is controlled and monitored by register TCSR.

TCSR
Transmit Control/Status Register (38$_H$) Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0			**TE**	**TVC**	**TV**	**0**	**TSOF**	**0**							
r			rh	rh	rh	r	rh	r							

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		**WA**	**TDVTR**	**TDEN**		**0**	**TDSSM**	**TDV**	**EOF**	**SOF**	**HPCMD**	**WAMD**	**FLEMD**	**SELMD**	**WLEMD**
r		rwh	rw	rw		r	rw	rh	rwh	rwh	rw	rw	rw	rw	rw

Field	Bits	Type	Description
WLEMD	0	rw	**WLE Mode** This bit enables the data handler to automatically update the bit field SCTR.WLE by the transmit control information TCI[3:0] and bit TCSR.EOF by TCI[4] (see **Page 14-33**). If enabled, an automatic update takes place when new data is loaded to register TBUF, either by writing to one of the transmit buffer input locations TBUFx or by an optional data buffer. 0_B The automatic update of SCTR.WLE and TCSR.EOF is disabled. 1_B The automatic update of SCTR.WLE and TCSR.EOF is enabled.

Field	Bits	Type	Description
SELMD	1	rw	**Select Mode** This bit can be used mainly for the SSC protocol. It enables the data handler to automatically update bit field PCR.CTR[20:16] by the transmit control information TCI[4:0] and clear bit field PCR.CTR[23:21] (see **Page 14-33**). If enabled, an automatic update takes place when new data is loaded to register TBUF, either by writing to one of the transmit buffer input locations TBUFx or by an optional data buffer. 0_B The automatic update of PCR.CTR[23:16] is disabled. 1_B The automatic update of PCR.CTR[23:16] is disabled.
FLEMD	2	rw	**FLE Mode** This bit enables the data handler to automatically update bits SCTR.FLE[4:0] by the transmit control information TCI[4:0] and to clear bit SCTR.FLE[5] (see **Page 14-33**). If enabled, an automatic update takes place when new data is loaded to register TBUF, either by writing to one of the transmit buffer input locations TBUFx or by an optional data buffer. 0_B The automatic update of FLE is disabled. 1_B The automatic update of FLE is enabled.
WAMD	3	rw	**WA Mode** This bit can be used mainly for the IIS protocol. It enables the data handler to automatically update bit TCSR.WA by the transmit control information TCI[4] (see **Page 14-33**). If enabled, an automatic update takes place when new data is loaded to register TBUF, either by writing to one of the transmit buffer input locations TBUFx or by an optional data buffer. 0_B The automatic update of bit WA is disabled. 1_B The automatic update of bit WA is enabled.

Field	Bits	Type	Description
HPCMD	4	rw	**Hardware Port Control Mode** This bit can be used mainly for the dual and quad SSC protocol. It enables the data handler to automatically update bit SCTR.DSM by the transmit control information TCI[1:0] and bit SCTR.HPCDIR by TCI[2] (see **Page 14-33**). If enabled, an automatic update takes place when new data is loaded to register TBUF, either by writing to one of the transmit buffer input locations TBUFx or by an optional data buffer. 0_B The automatic update of bits SCTR.DSM and SCTR.HPCDIR is disabled. 1_B The automatic update of bits SCTR.DSM and SCTR.HPCDIR is enabled.
SOF	5	rwh	**Start Of Frame** This bit is only taken into account for the SSC protocol, otherwise it is ignored. It indicates that the data word in TBUF is considered as the first word of a new SSC frame if it is valid for transmission (TCSR.TDV = 1). This bit becomes cleared when the TBUF data word is transferred to the transmit shift register. 0_B The data word in TBUF is not considered as first word of a frame. 1_B The data word in TBUF is considered as first word of a frame. A currently running frame is finished and MSLS becomes deactivated (respecting the programmed delays).

Field	Bits	Type	Description
EOF	6	rwh	**End Of Frame** This bit is only taken into account for the SSC protocol, otherwise it is ignored. It can be modified automatically by the data handler if bit WLEMD = 1. It indicates that the data word in TBUF is considered as the last word of an SSC frame. If it is the last word, the MSLS signal becomes inactive after the transfer, respecting the programmed delays. This bit becomes cleared when the TBUF data word is transferred to the transmit shift register. 0_B The data word in TBUF is not considered as last word of an SSC frame. 1_B The data word in TBUF is considered as last word of an SSC frame.
TDV	7	rh	**Transmit Data Valid** This bit indicates that the data word in the transmit buffer TBUF can be considered as valid for transmission. The TBUF data word can only be sent out if TDV = 1. It is automatically set when data is moved to TBUF (by writing to one of the transmit buffer input locations TBUFx, or optionally, by the bypass or FIFO mechanism). 0_B The data word in TBUF is not valid for transmission. 1_B The data word in TBUF is valid for transmission and a transmission start is possible. New data should not be written to a TBUFx input location while TDV = 1.

Field	Bits	Type	Description
TDSSM	8	rw	**TBUF Data Single Shot Mode** This bit defines if the data word TBUF data is considered as permanently valid or if the data should only be transferred once. 0_B The data word in TBUF is not considered as invalid after it has been loaded into the transmit shift register. The loading of the TBUF data into the shift register does not clear TDV. 1_B The data word in TBUF is considered as invalid after it has been loaded into the shift register. In ASC and IIC mode, TDV is cleared with the TBI event, whereas in SSC and IIS mode, it is cleared with the RSI event. TDSSM = 1 has to be programmed if an optional data buffer is used.
TDEN	[11:10]	rw	**TBUF Data Enable** This bit field controls the gating of the transmission start of the data word in the transmit buffer TBUF. 00_B A transmission start of the data word in TBUF is disabled. If a transmission is started, the passive data level is sent out. 01_B A transmission of the data word in TBUF can be started if TDV = 1. 10_B A transmission of the data word in TBUF can be started if TDV = 1 while DX2S = 0. 11_B A transmission of the data word in TBUF can be started if TDV = 1 while DX2S = 1.
TDVTR	12	rw	**TBUF Data Valid Trigger** This bit enables the transfer trigger unit to set bit TCSR.TE if the trigger signal DX2T becomes active for event driven transfer starts, e.g. timer-based or depending on an event at an input pin. Bit TDVTR has to be 0 for protocols where the input stage DX2 is used for data shifting. 0_B Bit TCSR.TE is permanently set. 1_B Bit TCSR.TE is set if DX2T becomes active while TDV = 1.

Field	Bits	Type	Description
WA	13	rwh	**Word Address** This bit is only taken into account for the IIS protocol, otherwise it is ignored. It can be modified automatically by the data handler if bit WAMD = 1. Bit WA defines for which channel the data stored in TBUF will be transmitted. 0_B The data word in TBUF will be transmitted after a falling edge of WA has been detected (referring to PSR.WA). 1_B The data word in TBUF will be transmitted after a rising edge of WA has been detected (referring to PSR.WA).
TSOF	24	rh	**Transmitted Start Of Frame** This bit indicates if the latest start of a data word transmission has taken place for the first data word of a new data frame. This bit is updated with the transmission start of each data word. 0_B The latest data word transmission has not been started for the first word of a data frame. 1_B The latest data word transmission has been started for the first word of a data frame.
TV	26	rh	**Transmission Valid** This bit represents the transmit buffer underflow and indicates if the latest start of a data word transmission has taken place with a valid data word from the transmit buffer TBUF. This bit is updated with the transmission start of each data word. 0_B The latest start of a data word transmission has taken place while no valid data was available. As a result, the transmission of a data words with passive level (SCTR.PDL) has been started. 1_B The latest start of a data word transmission has taken place with valid data from TBUF.

Field	Bits	Type	Description
TVC	27	rh	**Transmission Valid Cumulated** This bit cumulates the transmit buffer underflow indication TV. It is cleared automatically together with bit TV and has to be set by writing FMR.ATVC = 1. 0_B Since TVC has been set, at least one data buffer underflow condition has occurred. 1_B Since TVC has been set, no data buffer underflow condition has occurred.
TE	28	rh	**Trigger Event** If the transfer trigger mechanism is enabled, this bit indicates that a trigger event has been detected (DX2T = 1) while TCSR.TDV = 1. If the event trigger mechanism is disabled, the bit TE is permanently set. It is cleared by writing FMR.MTDV = 10_B or when the data word located in TBUF is loaded into the shift register. 0_B The trigger event has not yet been detected. A transmission of the data word in TBUF can not be started. 1_B The trigger event has been detected (or the trigger mechanism is switched off) and a transmission of the data word in TBUF can be started.
0	9, [23:14], 25, [31:29]	r	**Reserved** Read as 0; should be written with 0.

14.11.7.3 Flag Modification Registers

The flag modification register FMR allows the modification of control and status flags related to data handling by using only write accesses. Read accesses to FMR always deliver 0 at all bit positions.

Additionally, the service request outputs of this USIC channel can be activated by software (the activation is triggered by the write access and is deactivated automatically).

FMR
Flag Modification Register **(68_H)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0										SIO5	SIO4	SIO3	SIO2	SIO1	SIO0
r										w	w	w	w	w	w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CRDV1	CRDV0	0									ATVC	0		MTDV	
w	w	r									w	r		w	

Field	Bits	Type	Description
MTDV	[1:0]	w	**Modify Transmit Data Valid** Writing to this bit field can modify bits TCSR.TDV and TCSR.TE to control the start of a data word transmission by software. 00_B No action. 01_B Bit TDV is set, TE is unchanged. 10_B Bits TDV and TE are cleared. 11_B Reserved
ATVC	4	w	**Activate Bit TVC** Writing to this bit can set bit TCSR.TVC to start a new cumulation of the transmit buffer underflow condition. 0_B No action. 1_B Bit TCSR.TVC is set.
CRDV0	14	w	**Clear Bits RDV for RBUF0** Writing 1 to this bit clears bits RBUF01SR.RDV00 and RBUF01SR.RDV10 to declare the received data in RBUF0 as no longer valid (to emulate a read action). 0_B No action. 1_B Bits RBUF01SR.RDV00 and RBUF01SR.RDV10 are cleared.

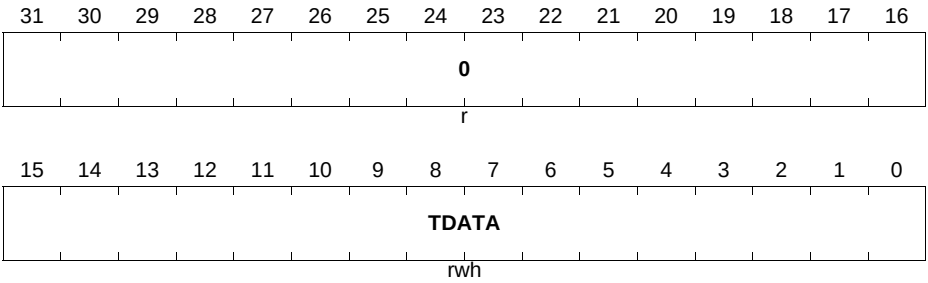
Field	Bits	Type	Description
CRDV1	15	w	**Clear Bit RDV for RBUF1** Writing 1 to this bit clears bits RBUF01SR.RDV01 and RBUF01SR.RDV11 to declare the received data in RBUF1 as no longer valid (to emulate a read action). 0_B No action. 1_B Bits RBUF01SR.RDV01 and RBUF01SR.RDV11 are cleared.
SIO0, **SIO1,** **SIO2,** **SIO3,** **SIO4,** **SIO5**	16, 17, 18, 19, 20, 21	w	**Set Interrupt Output SRx** Writing a 1 to this bit field activates the service request output SRx of this USIC channel. It has no impact on service request outputs of other USIC channels. 0_B No action. 1_B The service request output SRx is activated.
0	[3:2], [13:5], [31:22]	r	**Reserved** Read as 0; should be written with 0.

14.11.8 Data Buffer Registers

14.11.8.1 Transmit Buffer Locations

The 32 independent data input locations TBUF00 to TBUF31 are address locations that can be used as data entry locations for the transmit buffer. Data written to one of these locations will appear in a common register TBUF. Additionally, the 5 bit coding of the number [31:0] of the addressed data input location represents the transmit control information TCI (please refer to the protocol sections for more details).
The internal transmit buffer register TBUF contains the data that will be loaded to the transmit shift register for the next transmission of a data word. It can be read out at all TBUF00 to TBUF31 addresses.

TBUFx (x = 00-31)
Transmit Buffer Input Location x (80_H + x*4) Reset Value: 0000 0000_H

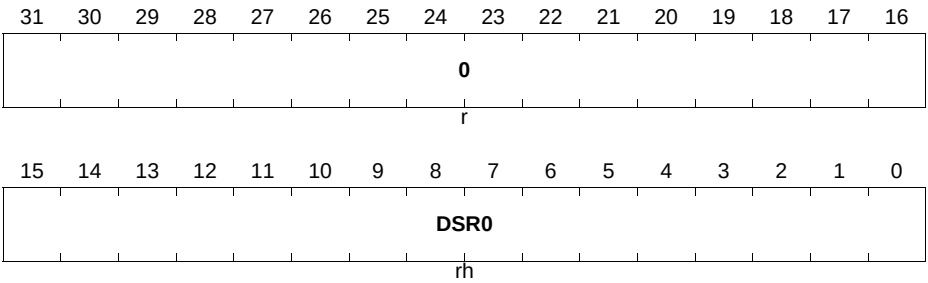


Field	Bits	Type	Description
TDATA	[15:0]	rwh	**Transmit Data** This bit field contains the data to be transmitted (read view). A data write action to at least the low byte of TDATA sets TCSR.TDV.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

14.11.8.2 Receive Buffer Registers RBUF0, RBUF1

The receive buffer register RBUF0 contains the data received from RSR0[3:0]. A read action does not change the status of the receive data from “not yet read = valid” to “already read = not valid”.

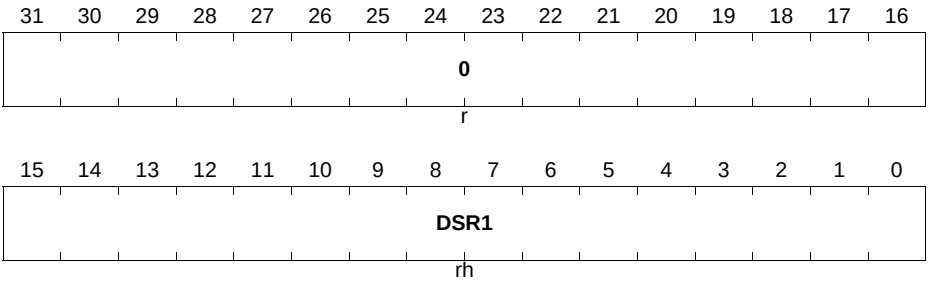
RBUF0
Receiver Buffer Register 0 ($5C_H$) Reset Value: 0000 0000_H



Field	Bits	Type	Description
DSR0	[15:0]	rh	**Data of Shift Registers 0[3:0]**
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

The receive buffer register RBUF1 contains the data received from RSR1[3:0]. A read action does not change the status of the receive data from “not yet read = valid” to “already read = not valid”.

RBUF1
Receiver Buffer Register 1 **(60_H)** **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
DSR1	[15:0]	rh	**Data of Shift Registers 1[3:0]**
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

The receive buffer status register RBUF01SR provides the status of the data in receive buffers RBUF0 and RBUF1.

RBUF01SR

Receiver Buffer 01 Status Register (64$_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DS1	RDV 11	RDV 10	0			PER R1	PAR 1	0	SOF 1	0		WLEN1			
rh	rh	rh	r			rh	rh	r	rh	r		rh			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DS0	RDV 01	RDV 00	0			PER R0	PAR 0	0	SOF 0	0		WLEN0			
rh	rh	rh	r			rh	rh	r	rh	r		rh			

Field	Bits	Type	Description
WLEN0	[3:0]	rh	**Received Data Word Length in RBUF0** This bit field indicates how many bits have been received within the last data word stored in RBUF0. This number indicates how many data bits have to be considered as receive data, whereas the other bits in RBUF0 have been cleared automatically. The received bits are always right-aligned. For all protocol modes besides dual and quad SSC, Received data word length = WLEN0 + 1 For dual SSC mode, Received data word length = WLEN0 + 2 For quad SSC mode, Received data word length = WLEN0 + 4
SOF0	6	rh	**Start of Frame in RBUF0** This bit indicates whether the data word in RBUF0 has been the first data word of a data frame. 0_B The data in RBUF0 has not been the first data word of a data frame. 1_B The data in RBUF0 has been the first data word of a data frame.

Field	Bits	Type	Description
PAR0	8	rh	**Protocol-Related Argument in RBUF0** This bit indicates the value of the protocol-related argument. This value is elaborated depending on the selected protocol and adds additional information to the data word in RBUF0. The meaning of this bit is described in the corresponding protocol chapter.
PERR0	9	rh	**Protocol-related Error in RBUF0** This bit indicates if the value of the protocol-related argument meets an expected value. This value is elaborated depending on the selected protocol and adds additional information to the data word in RBUF0. The meaning of this bit is described in the corresponding protocol chapter. 0_B The received protocol-related argument PAR matches the expected value. The reception of the data word sets bit PSR.RIF and can generate a receive interrupt. 1_B The received protocol-related argument PAR does not match the expected value. The reception of the data word sets bit PSR.AIF and can generate an alternative receive interrupt.
RDV00	13	rh	**Receive Data Valid in RBUF0** This bit indicates the status of the data content of register RBUF0. This bit is identical to bit RBUF01SR.RDV10 and allows consisting reading of information for the receive buffer registers. It is set when a new data word is stored in RBUF0 and automatically cleared if it is read out via RBUF. 0_B Register RBUF0 does not contain data that has not yet been read out. 1_B Register RBUF0 contains data that has not yet been read out.

Field	Bits	Type	Description
RDV01	14	rh	**Receive Data Valid in RBUF1** This bit indicates the status of the data content of register RBUF1. This bit is identical to bit RBUF01SR.RDV11 and allows consisting reading of information for the receive buffer registers. It is set when a new data word is stored in RBUF1 and automatically cleared if it is read out via RBUF. 0_B Register RBUF1 does not contain data that has not yet been read out. 1_B Register RBUF1 contains data that has not yet been read out.
DS0	15	rh	**Data Source** This bit indicates which receive buffer register (RBUF0 or RBUF1) is currently visible in registers RBUF(D) and in RBUFSR for the associated status information. It indicates which buffer contains the oldest data (the data that has been received first). This bit is identical to bit RBUF01SR.DS1 and allows consisting reading of information for the receive buffer registers. 0_B The register RBUF contains the data of RBUF0 (same for associated status information). 1_B The register RBUF contains the data of RBUF1 (same for associated status information).
WLEN1	[19:16]	rh	**Received Data Word Length in RBUF1** This bit field indicates how many bits have been received within the last data word stored in RBUF1. This number indicates how many data bits have to be considered as receive data, whereas the other bits in RBUF1 have been cleared automatically. The received bits are always right-aligned. For all protocol modes besides dual and quad SSC, Received data word length = WLEN1 + 1 For dual SSC mode, Received data word length = WLEN1 + 2 For quad SSC mode, Received data word length = WLEN1 + 4

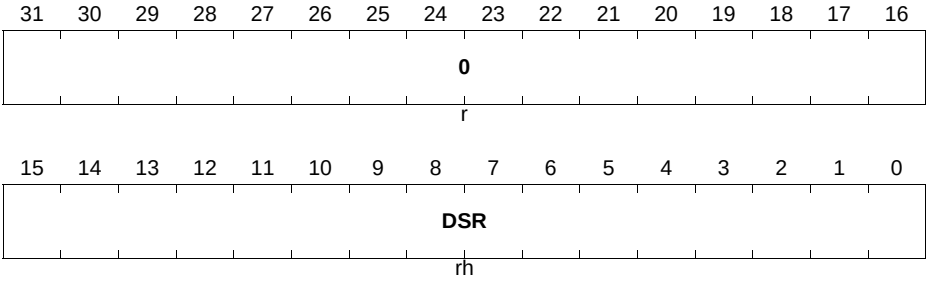
Field	Bits	Type	Description
SOF1	22	rh	**Start of Frame in RBUF1** This bit indicates whether the data word in RBUF1 has been the first data word of a data frame. 0_B The data in RBUF1 has not been the first data word of a data frame. 1_B The data in RBUF1 has been the first data word of a data frame.
PAR1	24	rh	**Protocol-Related Argument in RBUF1** This bit indicates the value of the protocol-related argument. This value is elaborated depending on the selected protocol and adds additional information to the data word in RBUF1. The meaning of this bit is described in the corresponding protocol chapter.
PERR1	25	rh	**Protocol-related Error in RBUF1** This bit indicates if the value of the protocol-related argument meets an expected value. This value is elaborated depending on the selected protocol and adds additional information to the data word in RBUF1. The meaning of this bit is described in the corresponding protocol chapter. 0_B The received protocol-related argument PAR matches the expected value. The reception of the data word sets bit PSR.RIF and can generate a receive interrupt. 1_B The received protocol-related argument PAR does not match the expected value. The reception of the data word sets bit PSR.AIF and can generate an alternative receive interrupt.
RDV10	29	rh	**Receive Data Valid in RBUF0** This bit indicates the status of the data content of register RBUF0. This bit is identical to bit RBUF01SR.RDV00 and allows consisting reading of information for the receive buffer registers. 0_B Register RBUF0 does not contain data that has not yet been read out. 1_B Register RBUF0 contains data that has not yet been read out.

Field	Bits	Type	Description
RDV11	30	rh	**Receive Data Valid in RBUF1** This bit indicates the status of the data content of register RBUF1. This bit is identical to bit RBUF01SR.RDV01 and allows consisting reading of information for the receive buffer registers. 0_B Register RBUF1 does not contain data that has not yet been read out. 1_B Register RBUF1 contains data that has not yet been read out.
DS1	31	rh	**Data Source** This bit indicates which receive buffer register (RBUF0 or RBUF1) is currently visible in registers RBUF(D) and in RBUFSR for the associated status information. It indicates which buffer contains the oldest data (the data that has been received first). This bit is identical to bit RBUF01SR.DS0 and allows consisting reading of information for the receive buffer registers. 0_B The register RBUF contains the data of RBUF0 (same for associated status information). 1_B The register RBUF contains the data of RBUF1 (same for associated status information).
0	[5:4], 7, [12:10], [21:20], 23, [28:26]	r	**Reserved** Read as 0; should be written with 0.

14.11.8.3 Receive Buffer Registers RBUF, RBUFD, RBUFSR

The receiver buffer register RBUF shows the content of the either RBUF0 or RBUF1, depending on the order of reception. Always the oldest data (the data word that has been received first) from both receive buffers can be read from RBUF. It is recommended to read out the received data from RBUF instead of RBUF0/1. With a read access of at least the low byte of RBUF, the status of the receive data is automatically changed from “not yet read = valid” to “already read = not valid”, the content of RBUF becomes updated, and the next received data word becomes visible in RBUF.

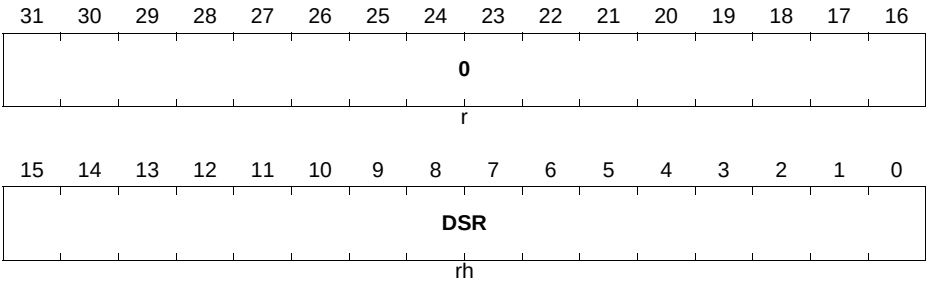
RBUF
Receiver Buffer Register **(54_H)** **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
DSR	[15:0]	rh	**Received Data** This bit field monitors the content of either RBUF0 or RBUF1, depending on the reception sequence.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

If a debugger should be used to monitor the received data, the automatic update mechanism has to be de-activated to guaranty data consistency. Therefore, the receiver buffer register for debugging RBUFD is available. It is similar to RBUF, but without the automatic update mechanism by a read action. So a debugger (or other monitoring function) can read RBUFD without disturbing the receive sequence.

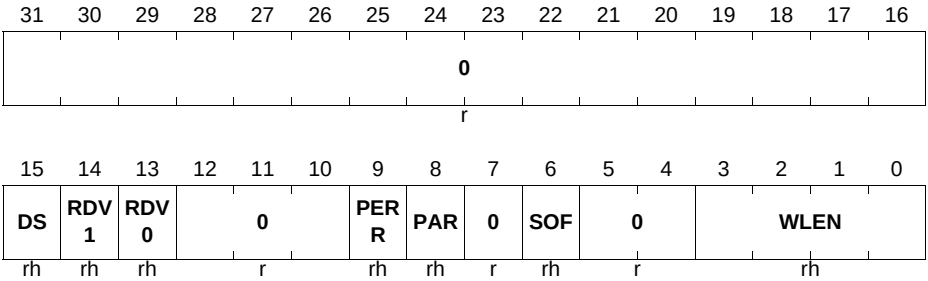
RBUFD
Receiver Buffer Register for Debugger(58_H) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
DSR	[15:0]	rh	**Data from Shift Register** Same as RBUF.DSR, but without releasing the buffer after a read action.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

The receive buffer status register RBUFSR provides the status of the data in receive buffers RBUF and RBUFD. If bits RBUF01SR.DS0 (or RBUF01SR.DS1) are 0, the lower 16-bit content of RBUF01SR is monitored in RBUFSR, otherwise the upper 16-bit content of RBUF01SR is shown.

RBUFSR
Receiver Buffer Status Register **(50_H)** **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
WLEN	[3:0]	rh	**Received Data Word Length in RBUF or RBUFD** Description see RBUF01SR.WLEN0 or RBUF01SR.WLEN1.
SOF	6	rh	**Start of Frame in RBUF or RBUFD** Description see RBUF01SR.SOF0 or RBUF01SR.SOF1.
PAR	8	rh	**Protocol-Related Argument in RBUF or RBUFD** Description see RBUF01SR.PAR0 or RBUF01SR.PAR1.
PERR	9	rh	**Protocol-related Error in RBUF or RBUFD** Description see RBUF01SR.PERR0 or RBUF01SR.PERR1.
RDV0	13	rh	**Receive Data Valid in RBUF or RBUFD** Description see RBUF01SR.RDV00 or RBUF01SR.RDV10.
RDV1	14	rh	**Receive Data Valid in RBUF or RBUFD** Description see RBUF01SR.RDV01 or RBUF01SR.RDV11.

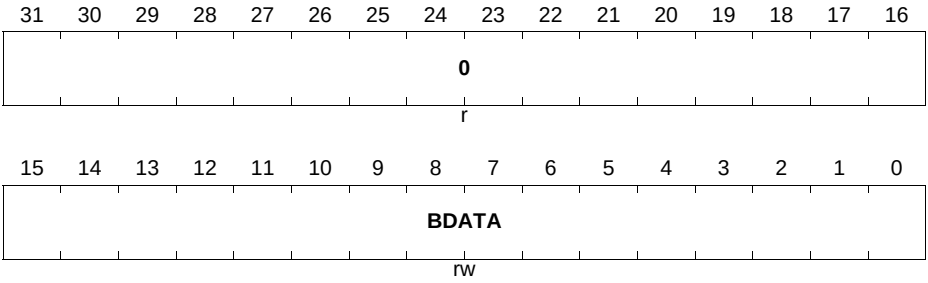
Field	Bits	Type	Description
DS	15	rh	**Data Source of RBUF or RBUFD** Description see RBUF01SR.DS0 or RBUF01SR.DS1.
0	[5:4], 7, [12:10], [31:16]	r	**Reserved** Read as 0; should be written with 0.

14.11.9 FIFO Buffer and Bypass Registers

14.11.9.1 Bypass Registers

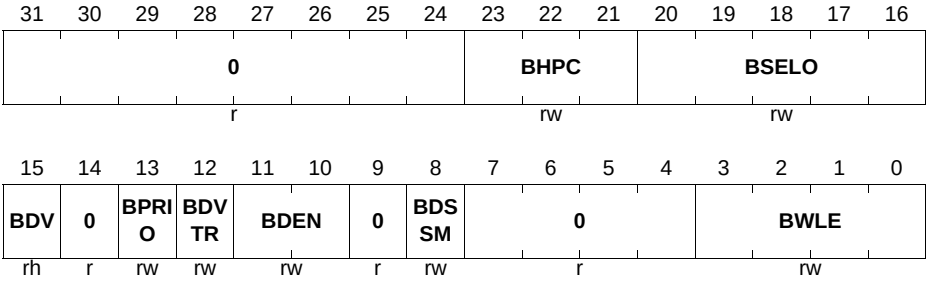
A write action to at least the low byte of the bypass data register sets BYPCR.BDV = 1 (bypass data tagged valid).

BYP
Bypass Data Register **(100_H)** **Reset Value: 0000 0000$_H$**



Bit (Field)	Width	Type	Description
BDATA	[15:0]	rw	**Bypass Data** This bit field contains the bypass data.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

BYPCR
Bypass Control Register **(104_H)** **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
BWLE	[3:0]	rw	**Bypass Word Length** This bit field defines the word length of the bypass data. The word length is given by BWLE + 1 with the data word being right-aligned in the data buffer at the bit positions [BWLE down to 0]. The bypass data word is always considered as an own frame with the length of BWLE. Same coding as SCTR.WLE.
BDSSM	8	rw	**Bypass Data Single Shot Mode** This bit defines if the bypass data is considered as permanently valid or if the bypass data is only transferred once (single shot mode). 0_B The bypass data is still considered as valid after it has been loaded into TBUF. The loading of the data into TBUF does not clear BDV. 1_B The bypass data is considered as invalid after it has been loaded into TBUF. The loading of the data into TBUF clears BDV.

Field	Bits	Type	Description
BDEN	[11:10]	rw	**Bypass Data Enable** This bit field defines if and how the transfer of bypass data to TBUF is enabled. 00_B The transfer of bypass data is disabled. 01_B The transfer of bypass data to TBUF is possible. Bypass data will be transferred to TBUF according to its priority if BDV = 1. 10_B Gated bypass data transfer is enabled. Bypass data will be transferred to TBUF according to its priority if BDV = 1 and while DX2S = 0. 11_B Gated bypass data transfer is enabled. Bypass data will be transferred to TBUF according to its priority if BDV = 1 and while DX2S = 1.
BDVTR	12	rw	**Bypass Data Valid Trigger** This bit enables the bypass data for being tagged valid when DX2T is active (for time framing or time-out purposes). 0_B Bit BDV is not influenced by DX2T. 1_B Bit BDV is set if DX2T is active.
BPRIO	13	rw	**Bypass Priority** This bit defines the priority between the bypass data and the transmit FIFO data. 0_B The transmit FIFO data has a higher priority than the bypass data. 1_B The bypass data has a higher priority than the transmit FIFO data.
BDV	15	rh	**Bypass Data Valid** This bit defines if the bypass data is valid for a transfer to TBUF. This bit is set automatically by a write access to at least the low-byte of register BYP. It can be cleared by software by writing TRBSCR.CBDV. 0_B The bypass data is not valid. 1_B The bypass data is valid.
BSELO	[20:16]	rw	**Bypass Select Outputs** This bit field contains the value that is written to PCR.CTR[20:16] if bypass data is transferred to TBUF while TCSR.SELMD = 1. In the SSC protocol, this bit field can be used to define which SELOx output line will be activated when bypass data is transmitted.

Field	Bits	Type	Description
BHPC	[23:21]	rw	**Bypass Hardware Port Control** This bit field contains the value that is written to SCTR[4:2] if bypass data is transferred to TBUF while TCSR.HPCMD = 1. In the SSC protocol, this bit field can be used to define the data shift mode and if hardware port control is enabled through CCR.HPCEN = 1, the pin direction when bypass data is transmitted.
0	[7:4], 9, 14, [31:24]	r	**Reserved** Read as 0; should be written with 0.

14.11.9.2 General FIFO Buffer Control Registers

The transmit and receive FIFO status information of USICx_CHy is given in registers USICx_CHy.TRBSR.

The bits related to the transmitter buffer in this register can only by written if the transmit buffer functionality is enabled by CCFG.TB = 1, otherwise write accesses are ignored. A similar behavior applies for the bits related to the receive buffer referring to CCFG.RB = 1.

The interrupt flags (event flags) in the transmit and receive FIFO status register TRBSR can be cleared by writing a 1 to the corresponding bit position in register TRBSCR, whereas writing a 0 has to effect on these bits. Writing a 1 by software to SRBI, RBERI, ARBI, STBI, or TBERI sets the corresponding bit to simulate the detection of a transmit/receive buffer event, but without activating any service request output (therefore, see FMR.SIOx).

Bits TBUS and RBUS have been implemented for testing purposes. They can be ignored by data handling software. Please note that a read action can deliver either a 0 or a 1 for these bits. It is recommended to treat them as “don’t care”.

TRBSR
Transmit/Receive Buffer Status Register

(114_H) **Reset Value: $0000\ 0808_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**TBFLVL**							**0**	**RBFLVL**						
r	rh							r	rh						

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**STBT**	**TBUS**	**TFULL**	**TEMPTY**	**0**	**TBERI**	**STBI**	**0**	**SRBT**	**RBUS**	**RFULL**	**REMPTY**	**ARBI**	**RBERI**	**SRBI**
r	rh	rh	rh	rh	r	rwh	rwh	r	rh	rh	rh	rh	rwh	rwh	rwh

Field	Bits	Type	Description
SRBI	0	rwh	**Standard Receive Buffer Event** This bit indicates that a standard receive buffer event has been detected. It is cleared by writing TRBSCR.CSRBI = 1. If enabled by RBCTR.SRBIEN, the service request output SRx selected by RBCTR.SRBINP becomes activated if a standard receive buffer event is detected. 0_B A standard receive buffer event has not been detected. 1_B A standard receive buffer event has been detected.
RBERI	1	rwh	**Receive Buffer Error Event** This bit indicates that a receive buffer error event has been detected. It is cleared by writing TRBSCR.CRBERI = 1. If enabled by RBCTR.RBERIEN, the service request output SRx selected by RBCTR.ARBINP becomes activated if a receive buffer error event is detected. 0_B A receive buffer error event has not been detected. 1_B A receive buffer error event has been detected.

Field	Bits	Type	Description
ARBI	2	rwh	**Alternative Receive Buffer Event** This bit indicates that an alternative receive buffer event has been detected. It is cleared by writing TRBSCR.CARBI = 1. If enabled by RBCTR.ARBIEN, the service request output SRx selected by RBCTR.ARBINP becomes activated if an alternative receive buffer event is detected. 0_B An alternative receive buffer event has not been detected. 1_B An alternative receive buffer event has been detected.
REMPTY	3	rh	**Receive Buffer Empty** This bit indicates whether the receive buffer is empty. 0_B The receive buffer is not empty. 1_B The receive buffer is empty.
RFULL	4	rh	**Receive Buffer Full** This bit indicates whether the receive buffer is full. 0_B The receive buffer is not full. 1_B The receive buffer is full.
RBUS	5	rh	**Receive Buffer Busy** This bit indicates whether the receive buffer is currently updated by the FIFO handler. 0_B The receive buffer information has been completely updated. 1_B The OUTR update from the FIFO memory is ongoing. A read from OUTR will be delayed. FIFO pointers from the previous read are not yet updated.
SRBT	6	rh	**Standard Receive Buffer Event Trigger** This bit triggers a standard receive buffer event when set. If enabled by RBCTR.SRBIEN, the service request output SRx selected by RBCTR.SRBINP becomes activated until the bit is cleared. 0_B A standard receive buffer event is not triggered using this bit. 1_B A standard receive buffer event is triggered using this bit.

Field	Bits	Type	Description
STBI	8	rwh	**Standard Transmit Buffer Event** This bit indicates that a standard transmit buffer event has been detected. It is cleared by writing TRBSCR.CSTBI = 1. If enabled by TBCTR.STBIEN, the service request output SRx selected by TBCTR.STBINP becomes activated if a standard transmit buffer event is detected. 0_B A standard transmit buffer event has not been detected. 1_B A standard transmit buffer event has been detected.
TBERI	9	rwh	**Transmit Buffer Error Event** This bit indicates that a transmit buffer error event has been detected. It is cleared by writing TRBSCR.CTBERI = 1. If enabled by TBCTR.TBERIEN, the service request output SRx selected by TBCTR.ATBINP becomes activated if a transmit buffer error event is detected. 0_B A transmit buffer error event has not been detected. 1_B A transmit buffer error event has been detected.
TEMPTY	11	rh	**Transmit Buffer Empty** This bit indicates whether the transmit buffer is empty. 0_B The transmit buffer is not empty. 1_B The transmit buffer is empty.
TFULL	12	rh	**Transmit Buffer Full** This bit indicates whether the transmit buffer is full. 0_B The transmit buffer is not full. 1_B The transmit buffer is full.

Field	Bits	Type	Description
TBUS	13	rh	**Transmit Buffer Busy** This bit indicates whether the transmit buffer is currently updated by the FIFO handler. 0_B The transmit buffer information has been completely updated. 1_B The FIFO memory update after write to INx is ongoing. A write to INx will be delayed. FIFO pointers from the previous INx write are not yet updated.
STBT	14	rh	**Standard Transmit Buffer Event Trigger** This bit triggers a standard transmit buffer event when set. If enabled by TBCTR.STBIEN, the service request output SRx selected by TBCTR.STBINP becomes activated until the bit is cleared. 0_B A standard transmit buffer event is not triggered using this bit. 1_B A standard transmit buffer event is triggered using this bit.
RBFLVL	[22:16]	rh	**Receive Buffer Filling Level** This bit field indicates the filling level of the receive buffer, starting with 0 for an empty buffer.
TBFLVL	[30:24]	rh	**Transmit Buffer Filling Level** This bit field indicates the filling level of the transmit buffer, starting with 0 for an empty buffer. *Note: The first data word written to Transmit FIFO will be loaded immediately to TBUF and removed from FIFO.*
0	7, 10, 15, 23, 31	r	**Reserved** Read as 0; should be written with 0.

The bits in register TRBSCR are used to clear the notification bits in register TRBSR or to clear the FIFO mechanism for the transmit or receive buffer. A read action always delivers 0.

TRBSCR
Transmit/Receive Buffer Status Clear Register
(118_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FLUSHTB	FLUSHRB	0			CBDV	CTBERI	CSTBI	0					CARBI	CRBERI	CSRBI
w	w	r			w	w	w	r					w	w	w

Field	Bits	Type	Description
CSRBI	0	w	**Clear Standard Receive Buffer Event** 0_B No effect. 1_B Clear TRBSR.SRBI.
CRBERI	1	w	**Clear Receive Buffer Error Event** 0_B No effect. 1_B Clear TRBSR.RBERI.
CARBI	2	w	**Clear Alternative Receive Buffer Event** 0_B No effect. 1_B Clear TRBSR.ARBI.
CSTBI	8	w	**Clear Standard Transmit Buffer Event** 0_B No effect. 1_B Clear TRBSR.STBI.
CTBERI	9	w	**Clear Transmit Buffer Error Event** 0_B No effect. 1_B Clear TRBSR.TBERI.
CBDV	10	w	**Clear Bypass Data Valid** 0_B No effect. 1_B Clear BYPCR.BDV.

Field	Bits	Type	Description
FLUSHRB	14	w	**Flush Receive Buffer** 0_B No effect. 1_B The receive FIFO buffer is cleared (filling level is cleared and output pointer is set to input pointer value). Should only be used while the FIFO buffer is not taking part in data traffic.
FLUSHTB	15	w	**Flush Transmit Buffer** 0_B No effect. 1_B The transmit FIFO buffer is cleared (filling level is cleared and output pointer is set to input pointer value). Should only be used while the FIFO buffer is not taking part in data traffic.
0	[7:3], [13:11], [31:16]	r	**Reserved** Read as 0; should be written with 0.

14.11.9.3 Transmit FIFO Buffer Control Registers

The transmit FIFO buffer is controlled by register TBCTR. TBCTR can only by written if the transmit buffer functionality is enabled by CCFG.TB = 1, otherwise write accesses are ignored.

TBCTR
Transmitter Buffer Control Register (108_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TBERIEN	STBIEN	0	LOF	0	SIZE			0		ATBINP			STBINP		
rw	rw	r	rw	r	rw			r		rw			rw		

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STBTEN	STBTM	LIMIT						0		DPTR					
rw	rw	rw						r		w					

Field	Bits	Type	Description
DPTR	[5:0]	w	**Data Pointer** This bit field defines the start value for the transmit buffer pointers when assigning the FIFO entries to the transmit FIFO buffer. A read always delivers 0. When writing DPTR while SIZE = 0, both transmitter pointers TDIPTR and RTDOPTR in register TRBPTR are updated with the written value and the buffer is considered as empty. A write access to DPTR while SIZE > 0 is ignored and does not modify the pointers.
LIMIT	[13:8]	rw	**Limit For Interrupt Generation** This bit field defines the target filling level of the transmit FIFO buffer that is used for the standard transmit buffer event detection.
STBTM	14	rw	**Standard Transmit Buffer Trigger Mode** This bit selects the standard transmit buffer event trigger mode. 0_B Trigger mode 0: While TRBSR.STBT=1, a standard buffer event will be generated whenever there is a data transfer to TBUF or data write to INx (depending on TBCTR.LOF setting). STBT is cleared when TRBSR.TBFLVL=TBCTR.LIMIT. 1_B Trigger mode 1: While TRBSR.STBT=1, a standard buffer event will be generated whenever there is a data transfer to TBUF or data write to INx (depending on TBCTR.LOF setting). STBT is cleared when TRBSR.TBFLVL=TBCTR.SIZE.
STBTEN	15	rw	**Standard Transmit Buffer Trigger Enable** This bit enables/disables triggering of the standard transmit buffer event through bit TRBSR.STBT. 0_B The standard transmit buffer event trigger through bit TRBSR.STBT is disabled. 1_B The standard transmit buffer event trigger through bit TRBSR.STBT is enabled.

Field	Bits	Type	Description
STBINP	[18:16]	rw	**Standard Transmit Buffer Interrupt Node Pointer** This bit field defines which service request output SRx becomes activated in case of a standard transmit buffer event. 000_B Output SR0 becomes activated. 001_B Output SR1 becomes activated. 010_B Output SR2 becomes activated. 011_B Output SR3 becomes activated. 100_B Output SR4 becomes activated. 101_B Output SR5 becomes activated. *Note: All other settings of the bit field are reserved.*
ATBINP	[21:19]	rw	**Alternative Transmit Buffer Interrupt Node Pointer** This bit field define which service request output SRx will be activated in case of a transmit buffer error event. 000_B Output SR0 becomes activated. 001_B Output SR1 becomes activated. 010_B Output SR2 becomes activated. 011_B Output SR3 becomes activated. 100_B Output SR4 becomes activated. 101_B Output SR5 becomes activated. *Note: All other settings of the bit field are reserved.*
SIZE	[26:24]	rw	**Buffer Size** This bit field defines the number of FIFO entries assigned to the transmit FIFO buffer. 000_B The FIFO mechanism is disabled. The buffer does not accept any request for data. 001_B The FIFO buffer contains 2 entries. 010_B The FIFO buffer contains 4 entries. 011_B The FIFO buffer contains 8 entries. 100_B The FIFO buffer contains 16 entries. 101_B The FIFO buffer contains 32 entries. 110_B The FIFO buffer contains 64 entries. 111_B Reserved

Field	Bits	Type	Description
LOF	28	rw	**Buffer Event on Limit Overflow** This bit defines which relation between filling level and programmed limit leads to a standard transmit buffer event. 0_B A standard transmit buffer event occurs when the filling level equals the limit value and gets lower due to transmission of a data word. 1_B A standard transmit buffer interrupt event occurs when the filling level equals the limit value and gets bigger due to a write access to a data input location INx.
STBIEN	30	rw	**Standard Transmit Buffer Interrupt Enable** This bit enables/disables the generation of a standard transmit buffer interrupt in case of a standard transmit buffer event. 0_B The standard transmit buffer interrupt generation is disabled. 1_B The standard transmit buffer interrupt generation is enabled.
TBERIEN	31	rw	**Transmit Buffer Error Interrupt Enable** This bit enables/disables the generation of a transmit buffer error interrupt in case of a transmit buffer error event (software writes to a full transmit buffer). 0_B The transmit buffer error interrupt generation is disabled. 1_B The transmit buffer error interrupt generation is enabled.
0	[7:6], [23:22], 27, 29	r	**Reserved** Read as 0; should be written with 0.

14.11.9.4 Receive FIFO Buffer Control Registers

The receive FIFO buffer is controlled by register RBCTR. This register can only be written if the receive buffer functionality is enabled by CCFG.RB = 1, otherwise write accesses are ignored.

RBCTR
Receiver Buffer Control Register ($10C_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RBERIEN	SRBIEN	ARBIEN	LOF	RNM	SIZE			RCIM		ARBINP			SRBINP		
rw	rw	rw	rw	rw	rw			rw		rw			rw		

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SRBTEN	SRBTM	LIMIT						0		DPTR					
rw	rw	rw						r		w					

Field	Bits	Type	Description
DPTR	[5:0]	w	**Data Pointer** This bit field defines the start value for the receive buffer pointers when assigning the FIFO entries to the receive FIFO buffer. A read always delivers 0. When writing DPTR while SIZE = 0, both receiver pointers RDIPTR and RDOPTR in register TRBPTR are updated with the written value and the buffer is considered as empty. A write access to DPTR while SIZE > 0 is ignored and does not modify the pointers.
LIMIT	[13:8]	rw	**Limit For Interrupt Generation** This bit field defines the target filling level of the receive FIFO buffer that is used for the standard receive buffer event detection.

Field	Bits	Type	Description
SRBTM	14	rw	**Standard Receive Buffer Trigger Mode** This bit selects the standard receive buffer event trigger mode. 0_B Trigger mode 0: While TRBSR.SRBT=1, a standard receive buffer event will be generated whenever there is a new data received or data read out (depending on RBCTR.LOF setting). SRBT is cleared when TRBSR.RBFLVL=RBCTR.LIMIT. 1_B Trigger mode 1: While TRBSR.SRBT=1, a standard receive buffer event will be generated whenever there is a new data received or data read out (depending on RBCTR.LOF setting). SRBT is cleared when TRBSR.RBFLVL=0.
SRBTEN	15	rw	**Standard Receive Buffer Trigger Enable** This bit enables/disables triggering of the standard receive buffer event through bit TRBSR.SRBT. 0_B The standard receive buffer event trigger through bit TRBSR.SRBT is disabled. 1_B The standard receive buffer event trigger through bit TRBSR.SRBT is enabled.
SRBINP	[18:16]	rw	**Standard Receive Buffer Interrupt Node Pointer** This bit field defines which service request output SRx becomes activated in case of a standard receive buffer event. 000_B Output SR0 becomes activated. 001_B Output SR1 becomes activated. 010_B Output SR2 becomes activated. 011_B Output SR3 becomes activated. 100_B Output SR4 becomes activated. 101_B Output SR5 becomes activated. *Note: All other settings of the bit field are reserved.*

Field	Bits	Type	Description
ARBINP	[21:19]	rw	**Alternative Receive Buffer Interrupt Node Pointer** This bit field defines which service request output SRx becomes activated in case of an alternative receive buffer event or a receive buffer error event. 000_B Output SR0 becomes activated. 001_B Output SR1 becomes activated. 010_B Output SR2 becomes activated. 011_B Output SR3 becomes activated. 100_B Output SR4 becomes activated. 101_B Output SR5 becomes activated. *Note: All other settings of the bit field are reserved.*
RCIM	[23:22]	rw	**Receiver Control Information Mode** This bit field defines which information from the receiver status register RBUFSR is propagated as 5 bit receiver control information RCI[4:0] to the receive FIFO buffer and can be read out in registers OUT(D)R. 00_B RCI[4] = PERR, RCI[3:0] = WLEN 01_B RCI[4] = SOF, RCI[3:0] = WLEN 10_B RCI[4] = 0, RCI[3:0] = WLEN 11_B RCI[4] = PERR, RCI[3] = PAR, RCI[2:1] = 00_B, RCI[0] = SOF
SIZE	[26:24]	rw	**Buffer Size** This bit field defines the number of FIFO entries assigned to the receive FIFO buffer. 000_B The FIFO mechanism is disabled. The buffer does not accept any request for data. 001_B The FIFO buffer contains 2 entries. 010_B The FIFO buffer contains 4 entries. 011_B The FIFO buffer contains 8 entries. 100_B The FIFO buffer contains 16 entries. 101_B The FIFO buffer contains 32 entries. 110_B The FIFO buffer contains 64 entries. 111_B Reserved

Field	Bits	Type	Description
RNM	27	rw	**Receiver Notification Mode** This bit defines the receive buffer event mode. The receive buffer error event is not affected by RNM. 0_B Filling level mode: A standard receive buffer event occurs when the filling level equals the limit value and changes, either due to a read access from OUTR (LOF = 0) or due to a new received data word (LOF = 1). 1_B RCI mode: A standard receive buffer event occurs when register OUTR is updated with a new value if the corresponding value in OUTR.RCI[4] = 0. If OUTR.RCI[4] = 1, an alternative receive buffer event occurs instead of the standard receive buffer event.
LOF	28	rw	**Buffer Event on Limit Overflow** This bit defines which relation between filling level and programmed limit leads to a standard receive buffer event in filling level mode (RNM = 0). In RCI mode (RNM = 1), bit fields LIMIT and LOF are ignored. 0_B A standard receive buffer event occurs when the filling level equals the limit value and gets lower due to a read access from OUTR. 1_B A standard receive buffer event occurs when the filling level equals the limit value and gets bigger due to the reception of a new data word.
ARBIEN	29	rw	**Alternative Receive Buffer Interrupt Enable** This bit enables/disables the generation of an alternative receive buffer interrupt in case of an alternative receive buffer event. 0_B The alternative receive buffer interrupt generation is disabled. 1_B The alternative receive buffer interrupt generation is enabled.

Field	Bits	Type	Description
SRBIEN	30	rw	**Standard Receive Buffer Interrupt Enable** This bit enables/disables the generation of a standard receive buffer interrupt in case of a standard receive buffer event. 0_B The standard receive buffer interrupt generation is disabled. 1_B The standard receive buffer interrupt generation is enabled.
RBERIEN	31	rw	**Receive Buffer Error Interrupt Enable** This bit enables/disables the generation of a receive buffer error interrupt in case of a receive buffer error event (the software reads from an empty receive buffer). 0_B The receive buffer error interrupt generation is disabled. 1_B The receive buffer error interrupt generation is enabled.
0	[7:6]	r	**Reserved** Read as 0; should be written with 0.

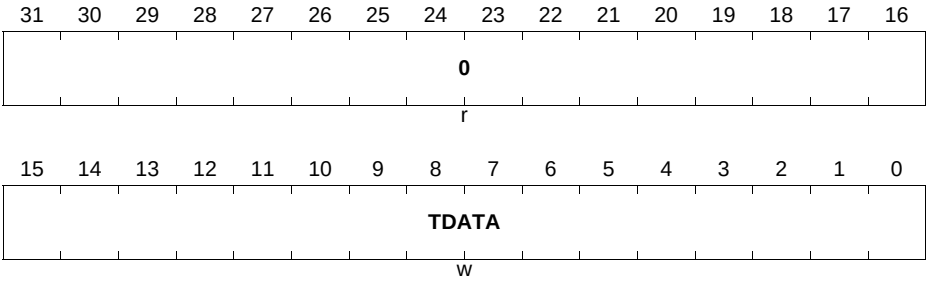
14.11.9.5 FIFO Buffer Data Registers

The 32 independent data input locations IN00 to IN31 are addresses that can be used as data entry locations for the transmit FIFO buffer. Data written to one of these locations will be stored in the transmit buffer FIFO. Additionally, the 5-bit coding of the number [31:0] of the addressed data input location represents the transmit control information TCI.

If the FIFO is already full and new data is written to it, the write access is ignored and a transmit buffer error event is signaled.

INx (x = 00-31)
Transmit FIFO Buffer Input Location x

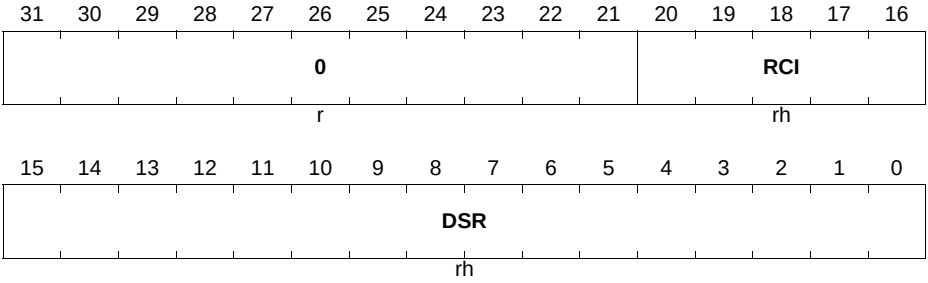
(180_H + x *4) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
TDATA	[15:0]	w	**Transmit Data** This bit field contains the data to be transmitted (write view), read actions deliver 0. A write action to at least the low byte of TDATA triggers the data storage in the FIFO.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

The receiver FIFO buffer output register OUTR shows the oldest received data word in the FIFO buffer and contains the receiver control information RCI containing the information selected by RBCTR.RCIM. A read action from this address location delivers the received data. With a read access of at least the low byte, the data is declared to be read and the next entry becomes visible. Write accesses to OUTR are ignored.

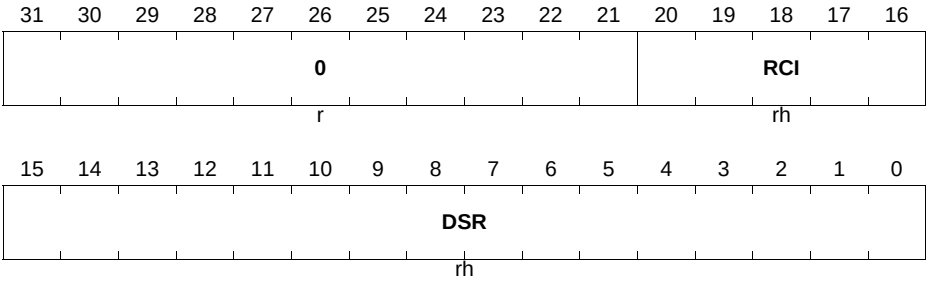
OUTR
Receiver Buffer Output Register (11C$_H$) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
DSR	[15:0]	rh	**Received Data** This bit field monitors the content of the oldest data word in the receive FIFO. Reading at least the low byte releases the buffer entry currently shown in DSR.
RCI	[20:16]	rh	**Receiver Control Information** This bit field monitors the receiver control information associated to DSR. The bit structure of RCI depends on bit field RBCTR.RCIM.
0	[31:21]	r	**Reserved** Read as 0; should be written with 0.

If a debugger should be used to monitor the received data in the FIFO buffer, the FIFO mechanism must not be activated in order to guaranty data consistency. Therefore, a second address set is available, named OUTDR (D like debugger), having the same bit fields like the original buffer output register OUTR, but without the FIFO mechanism. A debugger can read here (in order to monitor the receive data flow) without the risk of data corruption. Write accesses to OUTDR are ignored.

OUTDR
Receiver Buffer Output Register L for Debugger
(120_H) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
DSR	[15:0]	rh	**Data from Shift Register** Same as OUTR.DSR, but without releasing the buffer after a read action.
RCI	[20:16]	rh	**Receive Control Information from Shift Register** Same as OUTR.RCI.
0	[31:21]	r	**Reserved** Read as 0; should be written with 0.

14.11.9.6 FIFO Buffer Pointer Registers

The pointers for FIFO handling of the transmit and receive FIFO buffers are located in register TRBPTR. The pointers are automatically handled by the FIFO buffer mechanism and do not need to be modified by software. As a consequence, these registers can only be read by software (e.g. for verification purposes), whereas write accesses are ignored.

TRBPTR
Transmit/Receive Buffer Pointer Register

(110_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0		**RDOPTR**						**0**		**RDIPTR**					
r		rh						r		rh					

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		**TDOPTR**						**0**		**TDIPTR**					
r		rh						r		rh					

Field	Bits	Type	Description
TDIPTR	[5:0]	rh	**Transmitter Data Input Pointer** This bit field indicates the buffer entry that will be used for the next transmit data coming from the INx addresses.
TDOPTR	[13:8]	rh	**Transmitter Data Output Pointer** This bit field indicates the buffer entry that will be used for the next transmit data to be output to TBUF.
RDIPTR	[21:16]	rh	**Receiver Data Input Pointer** This bit field indicates the buffer entry that will be used for the next receive data coming from RBUF.
RDOPTR	[29:24]	rh	**Receiver Data Output Pointer** This bit field indicates the buffer entry that will be used for the next receive data to be output at the OUT(D)R addresses.
0	[7:6], [15:14], [23:22], [31:30]	r	**Reserved** Read as 0; should be written with 0.

14.12 Interconnects

The XMC4[12]00 device contains two USIC modules (USIC0 and USIC1) with 2 communication channels each.

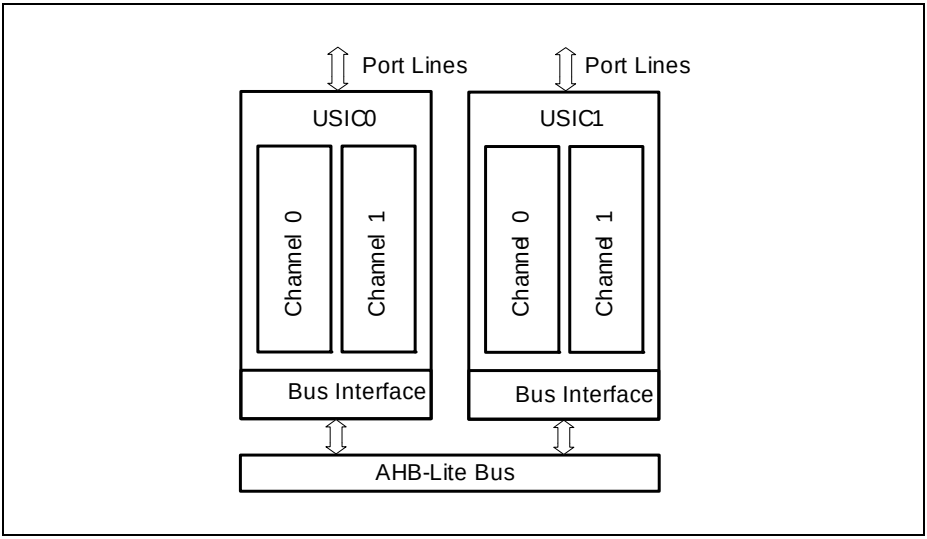


Figure 14-66 USIC Module Structure in XMC4[12]00

Figure 14-67 shows the I/O lines of one USIC channel. The tables in this section define the pin assignments and internal connections of the USIC channels I/O lines in the XMC4[12]00 device. Naming convention: USICx_CHy refers to USIC module x channel y.

The meaning of these pins with respect to each protocol is described in the protocols' respective chapters and summarized in **Table 14-2** and **Table 14-3**.

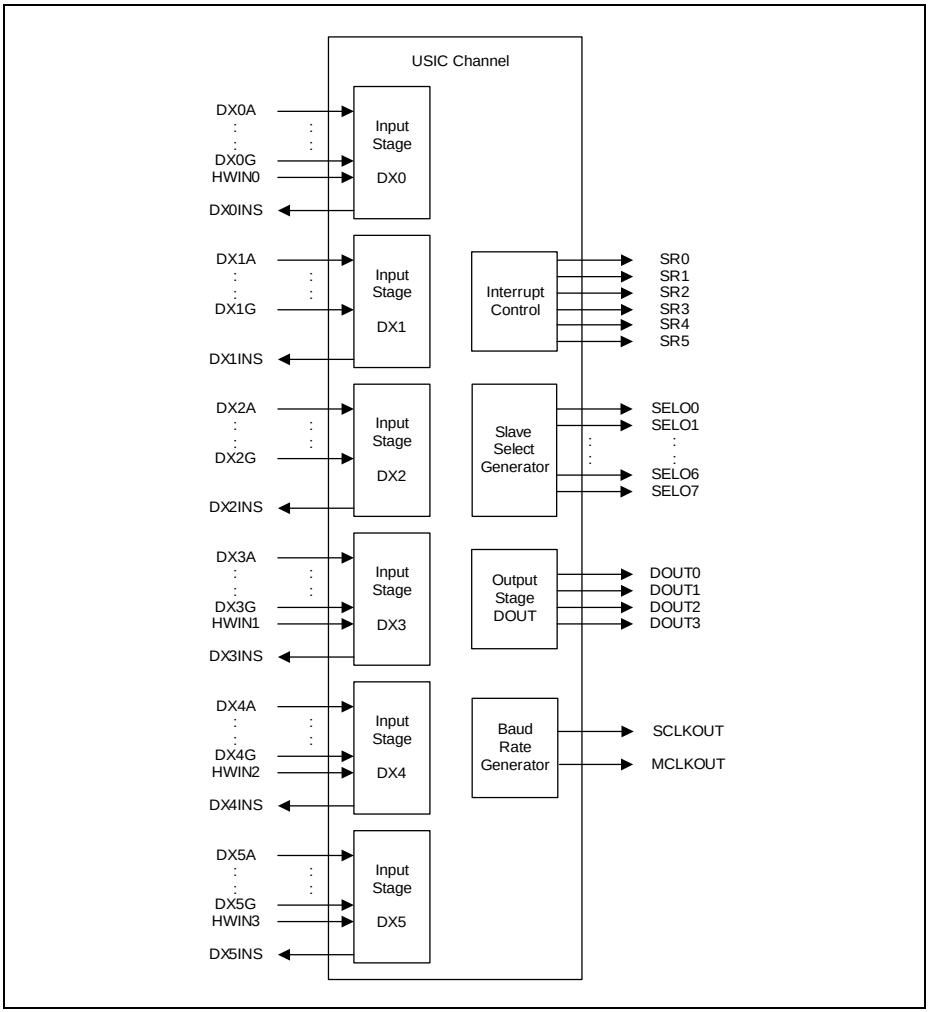


Figure 14-67 USIC Channel I/O Lines

The service request outputs SR[5:0] of one USIC channel is combined with those of the other channel with the module. Therefore, only 6 service request outputs are available per module.

14.12.1 USIC Module 0 Interconnects

The interconnects of USIC module 0 is grouped into the following categories:

- **USIC Module 0 Channel 0 Interconnects** (**Table 14-23**)
- **USIC Module 0 Channel 1 Interconnects** (**Table 14-24**)
- **USIC Module 0 Module Interconnects** (**Table 14-25**)

Table 14-23 USIC Module 0 Channel 0 Interconnects

Input/Output	I/O	Connected To	Description
Data Inputs (DX0)			
USIC0_CH0.DX0A	I	P1.5	Shift data input; used for: • ASC RXD • SSC MTSR/MRST • IIC SDA • IIS DIN
USIC0_CH0.DX0B	I	P1.4	
USIC0_CH0.DX0C	I	0	
USIC0_CH0.DX0D	I	0	
USIC0_CH0.DX0E	I	0	
USIC0_CH0.DX0F	I	XTAL1	
USIC0_CH0.DX0G	I	USIC0_CH0.DOUT0	Loop back shift data input
USIC0_CH0.HWIN0	I	P1.5	HW controlled shift data input
Clock Inputs			
USIC0_CH0.DX1A	I	P1.1	Shift clock input; used for: • SSC Slave SCLKIN • IIC SCL • IIS Slave SCLKIN • Optional for ASC, SSC Master and IIS Master
USIC0_CH0.DX1B	I	P0.8	
USIC0_CH0.DX1C	I	0	
USIC0_CH0.DX1D	I	0	
USIC0_CH0.DX1E	I	0	
USIC0_CH0.DX1F	I	USIC0_CH0.DX0INS	
USIC0_CH0.DX1G	I	USIC0_CH0.SCLKOUT	Loop back shift clock input
Control Inputs			
USIC0_CH0.DX2A	I	P1.0	Shift control input; used for: • SSC Slave SELIN • IIS Slave WAIN • Optional for ASC, SSC Master, IIC and IIS Master
USIC0_CH0.DX2B	I	P0.7	
USIC0_CH0.DX2C	I	0	
USIC0_CH0.DX2D	I	0	
USIC0_CH0.DX2E	I	CCU40.SR1	
USIC0_CH0.DX2F	I	CCU80.SR1	
USIC0_CH0.DX2G	I	USIC0_CH0.SELO0	Loop back shift control input
Data Inputs (DX3)			

Table 14-23 USIC Module 0 Channel 0 Interconnects (cont'd)

Input/Output	I/O	Connected To	Description
USIC0_CH0.DX3A	I	0	Shift data input; used for: • Dual and Quad SSC MTSR1/MRST1 • Can be ignored for all other protocols
USIC0_CH0.DX3B	I	0	
USIC0_CH0.DX3C	I	0	
USIC0_CH0.DX3D	I	0	
USIC0_CH0.DX3E	I	0	
USIC0_CH0.DX3F	I	0	
USIC0_CH0.DX3G	I	USIC0_CH0.DOUT1	Loop back shift data input
USIC0_CH0.HWIN1	I	P1.4	HW controlled shift data input
Data Inputs (DX4)			
USIC0_CH0.DX4A	I	0	Shift data input; used for: • Quad SSC MTSR2/MRST2 • Can be ignored for all other protocols
USIC0_CH0.DX4B	I	0	
USIC0_CH0.DX4C	I	0	
USIC0_CH0.DX4D	I	0	
USIC0_CH0.DX4E	I	0	
USIC0_CH0.DX4F	I	0	
USIC0_CH0.DX4G	I	USIC0_CH0.DOUT2	Loop back shift data input
USIC0_CH0.HWIN2	I	P1.3	HW controlled shift data input
Data Inputs (DX5)			
USIC0_CH0.DX5A	I	0	Shift data input; used for: • Quad SSC MTSR3/MRST3 • Can be ignored for all other protocols
USIC0_CH0.DX5B	I	0	
USIC0_CH0.DX5C	I	0	
USIC0_CH0.DX5D	I	0	
USIC0_CH0.DX5E	I	0	
USIC0_CH0.DX5F	I	0	
USIC0_CH0.DX5G	I	USIC0_CH0.DOUT3	Loop back shift data input
USIC0_CH0.HWIN3	I	P1.2	HW controlled shift data input
Data Outputs			

Table 14-23 USIC Module 0 Channel 0 Interconnects (cont'd)

Input/Output	I/O	Connected To	Description
USIC0_CH0.DOUT0	O	P1.5 P1.7 P1.5.HW0_OUT	Shift data output; used for: • ASC TXD • SSC MTSR/MRST • IIC SDA • IIS DOUT
USIC0_CH0.DOUT1	O	P1.4.HW0_OUT	Shift data output; used for: • Dual and Quad SSC MTSR1/MRST1 • Can be ignored for all other protocols
USIC0_CH0.DOUT2	O	P1.3.HW0_OUT	Shift data output; used for: • Quad SSC MTSR2/MRST2 • Can be ignored for all other protocols
USIC0_CH0.DOUT3	O	P1.2.HW0_OUT	Shift data output; used for: • Quad SSC MTSR3/MRST3 • Can be ignored for all other protocols
Clock Outputs			
USIC0_CH0.MCLKOUT	O	P1.3	Master clock output (optional for all protocols)
USIC0_CH0.SCLKOUT	O	P0.8 P1.1 P1.9	Shift clock output; used for: • Master SCLKOUT in SSC and IIS • IIC SCL • Can be ignored in ASC
Control Outputs			

Table 14-23 USIC Module 0 Channel 0 Interconnects (cont'd)

Input/Output	I/O	Connected To	Description
USIC0_CH0.SELO0	O	P0.7 P1.0	Shift control output; used for: • SSC Master SELO • IIS WA • Can be ignored for all other protocols
USIC0_CH0.SELO1	O	P1.8	
USIC0_CH0.SELO2	O	not connected	
USIC0_CH0.SELO3	O	not connected	
USIC0_CH0.SELO4	O	not connected	
USIC0_CH0.SELO5	O	not connected	
USIC0_CH0.SELO6	O	not connected	
USIC0_CH0.SELO7	O	not connected	
System Related Outputs			
USIC0_CH0.DX0INS	O	USIC0_CH0.DX1F	Selected DX0 input signal
USIC0_CH0.DX1INS	O	DAC.TRIGGER[6]	Selected DX1 input signal
USIC0_CH0.DX2INS	O	CCU40.IN0L	Selected DX2 input signal
USIC0_CH0.DX3INS	O	not connected	Selected DX3 input signal
USIC0_CH0.DX4INS	O	not connected	Selected DX4 input signal
USIC0_CH0.DX5INS	O	not connected	Selected DX5 input signal

Table 14-24 USIC Module 0 Channel 1 Interconnects

Input/Output	I/O	Connected To	Description
Data Inputs (DX0)			
USIC0_CH1.DX0A	I	P2.2	Shift data input; used for: • ASC RXD • SSC MTSR/MRST • IIC SDA • IIS DIN
USIC0_CH1.DX0B	I	P2.5	
USIC0_CH1.DX0C	I	0	
USIC0_CH1.DX0D	I	0	
USIC0_CH1.DX0E	I	0	
USIC0_CH1.DX0F	I	XTAL1	
USIC0_CH1.DX0G	I	USIC0_CH1.DOUT0	Loop back shift data input
USIC0_CH1.HWIN0	I	0	HW controlled shift data input
Clock Inputs			

Table 14-24 USIC Module 0 Channel 1 Interconnects (cont'd)

Input/Output	I/O	Connected To	Description
USIC0_CH1.DX1A	I	P2.4	Shift clock input; used for: • SSC Slave SCLKIN • IIC SCL • IIS Slave SCLKIN • Optional for ASC, SSC Master and IIS Master
USIC0_CH1.DX1B	I	P3.0	
USIC0_CH1.DX1C	I	0	
USIC0_CH1.DX1D	I	0	
USIC0_CH1.DX1E	I	0	
USIC0_CH1.DX1F	I	USIC0_CH1.DX0INS	
USIC0_CH1.DX1G	I	USIC0_CH1.SCLKOUT	Loop back shift clock input
Control Inputs			
USIC0_CH1.DX2A	I	P2.3	Shift control input; used for: • SSC Slave SELIN • IIS Slave WAIN • Optional for ASC, SSC Master, IIC and IIS Master
USIC0_CH1.DX2B	I	P3.1	
USIC0_CH1.DX2C	I	0	
USIC0_CH1.DX2D	I	0	
USIC0_CH1.DX2E	I	0	
USIC0_CH1.DX2F	I	CCU80.SR1	
USIC0_CH1.DX2G	I	USIC0_CH1.SELO0	Loop back shift control input
Data Inputs (DX3)			
USIC0_CH1.DX3A	I	0	Shift data input; used for: • Dual and Quad SSC MTSR1/MRST1 • Can be ignored for all other protocols
USIC0_CH1.DX3B	I	0	
USIC0_CH1.DX3C	I	0	
USIC0_CH1.DX3D	I	0	
USIC0_CH1.DX3E	I	0	
USIC0_CH1.DX3F	I	0	
USIC0_CH1.DX3G	I	USIC0_CH1.DOUT1	Loop back shift data input
USIC0_CH1.HWIN1	I	0	HW controlled shift data input
Data Inputs (DX4)			

Table 14-24 USIC Module 0 Channel 1 Interconnects (cont'd)

Input/Output	I/O	Connected To	Description
USIC0_CH1.DX4A	I	0	Shift data input; used for: • Quad SSC MTSR2/MRST2 • Can be ignored for all other protocols
USIC0_CH1.DX4B	I	0	
USIC0_CH1.DX4C	I	0	
USIC0_CH1.DX4D	I	0	
USIC0_CH1.DX4E	I	0	
USIC0_CH1.DX4F	I	0	
USIC0_CH1.DX4G	I	USIC0_CH1.DOUT2	Loop back shift data input
USIC0_CH1.HWIN2	I	0	HW controlled shift data input
Data Inputs (DX5)			
USIC0_CH1.DX5A	I	0	Shift data input; used for: • Quad SSC MTSR3/MRST3 • Can be ignored for all other protocols
USIC0_CH1.DX5B	I	0	
USIC0_CH1.DX5C	I	0	
USIC0_CH1.DX5D	I	0	
USIC0_CH1.DX5E	I	0	
USIC0_CH1.DX5F	I	0	
USIC0_CH1.DX5G	I	USIC0_CH1.DOUT3	Loop back shift data input
USIC0_CH1.HWIN3	I	0	HW controlled shift data input
Data Outputs			
USIC0_CH1.DOUT0	O	P2.5	Shift data output; used for: • ASC TXD • SSC MTSR/MRST • IIC SDA • IIS DOUT
USIC0_CH1.DOUT1	O	not connected	Shift data output; used for: • Dual and Quad SSC MTSR1/MRST1 • Can be ignored for all other protocols
USIC0_CH1.DOUT2	O	not connected	Shift data output; used for: • Quad SSC MTSR2/MRST2 • Can be ignored for all other protocols

Table 14-24 USIC Module 0 Channel 1 Interconnects (cont'd)

Input/Output	I/O	Connected To	Description
USIC0_CH1.DOUT3	O	not connected	Shift data output; used for: • Quad SSC MTSR3/MRST3 • Can be ignored for all other protocols
Clock Outputs			
USIC0_CH1.MCLKOUT	O	not connected	Master clock output (optional for all protocols)
USIC0_CH1.SCLKOUT	O	P2.4 P3.0	Shift clock output; used for: • Master SCLKOUT in SSC and IIS • IIC SCL • Can be ignored in ASC
Control Outputs			
USIC0_CH1.SELO0	O	P2.3 P3.1	Shift control output; used for: • SSC Master SELO • IIS WA • Can be ignored for all other protocols
USIC0_CH1.SELO1	O	not connected	
USIC0_CH1.SELO2	O	not connected	
USIC0_CH1.SELO3	O	not connected	
USIC0_CH1.SELO4	O	not connected	
USIC0_CH1.SELO5	O	not connected	
USIC0_CH1.SELO6	O	not connected	
USIC0_CH1.SELO7	O	not connected	
System Related Outputs			
USIC0_CH1.DX0INS	O	USIC0_CH1.DX1F	Selected DX0 input signal
USIC0_CH1.DX1INS	O	not connected	Selected DX1 input signal
USIC0_CH1.DX2INS	O	CCU40.IN2L	Selected DX2 input signal
USIC0_CH1.DX3INS	O	not connected	Selected DX3 input signal
USIC0_CH1.DX4INS	O	not connected	Selected DX4 input signal
USIC0_CH1.DX5INS	O	not connected	Selected DX5 input signal

Table 14-25 USIC Module 0 Module Interconnects

Input/Output	I/O	Connected To	Description
USIC0_SR[1:0]	O	NVIC GPDMA	interrupt output lines (service requests SRx)
USIC0_SR[5:2]	O	NVIC	interrupt output lines (service requests SRx)

14.12.2 USIC Module 1 Interconnects

The interconnects of USIC module 1 is grouped into the following three categories:

- **USIC Module 1 Channel 0 Interconnects** (**Table 14-26**)
- **USIC Module 1 Channel 1 Interconnects** (**Table 14-27**)
- **USIC Module 1 Module Interconnects** (**Table 14-28**)

Table 14-26 USIC Module 1 Channel 0 Interconnects

Input/Output	I/O	Connected To	Description
Data Inputs (DX0)			
USIC1_CH0.DX0A	I	P0.4	Shift data input; used for: • ASC RXD • SSC MTSR/MRST • IIC SDA • IIS DIN
USIC1_CH0.DX0B	I	P0.5	
USIC1_CH0.DX0C	I	P2.15	
USIC1_CH0.DX0D	I	P2.14	
USIC1_CH0.DX0E	I	0	
USIC1_CH0.DX0F	I	XTAL1	
USIC1_CH0.DX0G	I	USIC1_CH0.DOUT0	Loop back shift data input
USIC1_CH0.HWIN0	I	P0.5	HW controlled shift data input
Clock Inputs			
USIC1_CH0.DX1A	I	P0.11	Shift clock input; used for: • SSC Slave SCLKIN • IIC SCL • IIS Slave SCLKIN • Optional for ASC, SSC Master and IIS Master
USIC1_CH0.DX1B	I	0	
USIC1_CH0.DX1C	I	0	
USIC1_CH0.DX1D	I	0	
USIC1_CH0.DX1E	I	0	
USIC1_CH0.DX1F	I	USIC1_CH0.DX0INS	
USIC1_CH0.DX1G	I	USIC1_CH0.SCLKOUT	Loop back shift clock input
Control Inputs			

Table 14-26 USIC Module 1 Channel 0 Interconnects (cont'd)

Input/Output	I/O	Connected To	Description
USIC1_CH0.DX2A	I	P0.6	Shift control input; used for: • SSC Slave SELIN • IIS Slave WAIN • Optional for ASC, SSC Master, IIC and IIS Master
USIC1_CH0.DX2B	I	0	
USIC1_CH0.DX2C	I	0	
USIC1_CH0.DX2D	I	0	
USIC1_CH0.DX2E	I	CCU41.SR1	
USIC1_CH0.DX2F	I	CCU81.SR1	
USIC1_CH0.DX2G	I	USIC1_CH0.SELO0	Loop back shift control input
Data Inputs (DX3)			
USIC1_CH0.DX3A	I	0	Shift data input; used for: • Dual and Quad SSC MTSR1/MRST1 • Can be ignored for all other protocols
USIC1_CH0.DX3B	I	0	
USIC1_CH0.DX3C	I	0	
USIC1_CH0.DX3D	I	0	
USIC1_CH0.DX3E	I	0	
USIC1_CH0.DX3F	I	0	
USIC1_CH0.DX3G	I	USIC1_CH0.DOUT1	Loop back shift data input
USIC1_CH0.HWIN1	I	P0.4	HW controlled shift data input
Data Inputs (DX4)			
USIC1_CH0.DX4A	I	0	Shift data input; used for: • Quad SSC MTSR2/MRST2 • Can be ignored for all other protocols
USIC1_CH0.DX4B	I	0	
USIC1_CH0.DX4C	I	0	
USIC1_CH0.DX4D	I	0	
USIC1_CH0.DX4E	I	0	
USIC1_CH0.DX4F	I	0	
USIC1_CH0.DX4G	I	USIC1_CH0.DOUT2	Loop back shift data input
USIC1_CH0.HWIN2	I	P0.3	HW controlled shift data input
Data Inputs (DX5)			

Table 14-26 USIC Module 1 Channel 0 Interconnects (cont'd)

Input/Output	I/O	Connected To	Description
USIC1_CH0.DX5A	I	0	Shift data input; used for: • Quad SSC MTSR2/MRST2 • Can be ignored for all other protocols
USIC1_CH0.DX5B	I	0	
USIC1_CH0.DX5C	I	0	
USIC1_CH0.DX5D	I	0	
USIC1_CH0.DX5E	I	0	
USIC1_CH0.DX5F	I	0	
USIC1_CH0.DX5G	I	USIC1_CH0.DOUT3	Loop back shift data input
USIC1_CH0.HWIN3	I	P0.2	HW controlled shift data input
Data Outputs			
USIC1_CH0.DOUT0	O	P0.5 P1.15 P2.14 P0.5.HW0_OUT	Shift data output; used for: • ASC TXD • SSC MTSR/MRST • IIC SDA • IIS DOUT
USIC1_CH0.DOUT1	O	P0.4.HW0_OUT	Shift data output; used for: • Dual and Quad SSC MTSR1/MRST1 • Can be ignored for all other protocols
USIC1_CH0.DOUT2	O	P0.3.HW0_OUT	Shift data output; used for: • Quad SSC MTSR2/MRST2 • Can be ignored for all other protocols
USIC1_CH0.DOUT3	O	P0.2.HW0_OUT	Shift data output; used for: • Quad SSC MTSR3/MRST3 • Can be ignored for all other protocols
Clock Outputs			
USIC1_CH0.MCLKOUT	O	not connected	Master clock output (optional for all protocols)
USIC1_CH0.SCLKOUT	O	P0.11	Shift clock output; used for: • Master SCLKOUT in SSC and IIS • IIC SCL • Can be ignored in ASC

Table 14-26 USIC Module 1 Channel 0 Interconnects (cont'd)

Input/Output	I/O	Connected To	Description
Control Outputs			
USIC1_CH0.SELO0	O	P0.6	Shift control output; used for: • SSC Master SELO • IIS WA • Can be ignored for all other protocols
USIC1_CH0.SELO1	O	not connected	
USIC1_CH0.SELO2	O	not connected	
USIC1_CH0.SELO3	O	not connected	
USIC1_CH0.SELO4	O	not connected	
USIC1_CH0.SELO5	O	not connected	
USIC1_CH0.SELO6	O	not connected	
USIC1_CH0.SELO7	O	not connected	
System Related Outputs			
USIC1_CH0.DX0INS	O	USIC1_CH0.DX1F	Selected DX0 input signal
USIC1_CH0.DX1INS	O	DAC.TRIGGER[7]	Selected DX1 input signal
USIC1_CH0.DX2INS	O	CCU40.IN3L	Selected DX2 input signal
USIC1_CH0.DX3INS	O	not connected	Selected DX3 input signal
USIC1_CH0.DX4INS	O	not connected	Selected DX4 input signal
USIC1_CH0.DX5INS	O	not connected	Selected DX5 input signal

Table 14-27 USIC Module 1 Channel 1 Interconnects

Input/Output	I/O	Connected To	Description
Data Inputs (DX0)			
USIC1_CH1.DX0A	I	0	Shift data input; used for: • ASC RXD • SSC MTSR/MRST • IIC SDA • IIS DIN
USIC1_CH1.DX0B	I	0	
USIC1_CH1.DX0C	I	0	
USIC1_CH1.DX0D	I	P0.0	
USIC1_CH1.DX0E	I	CAN1INS	
USIC1_CH1.DX0F	I	XTAL1	
USIC1_CH1.DX0G	I	USIC1_CH1.DOUT0	Loop back shift data input
USIC1_CH1.HWIN0	I	0	HW controlled shift data input
Clock Inputs			

Table 14-27 USIC Module 1 Channel 1 Interconnects (cont'd)

Input/Output	I/O	Connected To	Description
USIC1_CH1.DX1A	I	P0.10	Shift clock input; used for: • SSC Slave SCLKIN • IIC SCL • IIS Slave SCLKIN • Optional for ASC, SSC Master and IIS Master
USIC1_CH1.DX1B	I	0	
USIC1_CH1.DX1C	I	0	
USIC1_CH1.DX1D	I	0	
USIC1_CH1.DX1E	I	0	
USIC1_CH1.DX1F	I	USIC1_CH1.DX0INS	
USIC1_CH1.DX1G	I	USIC1_CH1.SCLKOUT	Loop back shift clock input
Control Inputs			
USIC1_CH1.DX2A	I	P0.9	Shift control input; used for: • SSC Slave SELIN • IIS Slave WAIN • Optional for ASC, SSC Master, IIC and IIS Master
USIC1_CH1.DX2B	I	0	
USIC1_CH1.DX2C	I	0	
USIC1_CH1.DX2D	I	0	
USIC1_CH1.DX2E	I	0	
USIC1_CH1.DX2F	I	CCU81.SR1	
USIC1_CH1.DX2G	I	USIC1_CH1.SELO0	Loop back shift control input
Data Inputs (DX3)			
USIC1_CH1.DX3A	I	0	Shift data input; used for: • Dual and Quad SSC MTSR1/MRST1 • Can be ignored for all other protocols
USIC1_CH1.DX3B	I	0	
USIC1_CH1.DX3C	I	0	
USIC1_CH1.DX3D	I	0	
USIC1_CH1.DX3E	I	0	
USIC1_CH1.DX3F	I	0	
USIC1_CH1.DX3G	I	USIC1_CH1.DOUT1	Loop back shift data input
USIC1_CH1.HWIN1	I	0	HW controlled shift data input
Data Inputs (DX4)			

Table 14-27 USIC Module 1 Channel 1 Interconnects (cont'd)

Input/Output	I/O	Connected To	Description
USIC1_CH1.DX4A	I	0	Shift data input; used for: • Quad SSC MTSR2/MRST2 • Can be ignored for all other protocols
USIC1_CH1.DX4B	I	0	
USIC1_CH1.DX4C	I	0	
USIC1_CH1.DX4D	I	0	
USIC1_CH1.DX4E	I	0	
USIC1_CH1.DX4F	I	0	
USIC1_CH1.DX4G	I	USIC1_CH1.DOUT2	Loop back shift data input
USIC1_CH1.HWIN2	I	0	HW controlled shift data input
Data Inputs (DX5)			
USIC1_CH1.DX5A	I	0	Shift data input; used for: • Quad SSC MTSR3/MRST3 • Can be ignored for all other protocols
USIC1_CH1.DX5B	I	0	
USIC1_CH1.DX5C	I	0	
USIC1_CH1.DX5D	I	0	
USIC1_CH1.DX5E	I	0	
USIC1_CH1.DX5F	I	0	
USIC1_CH1.DX5G	I	USIC1_CH1.DOUT3	Loop back shift data input
USIC1_CH1.HWIN3	I	0	HW controlled shift data input
Data Outputs			
USIC1_CH1.DOUT0	O	P0.1 P1.9	Shift data output; used for: • ASC TXD • SSC MTSR/MRST • IIC SDA • IIS DOUT
USIC1_CH1.DOUT1	O	not connected	Shift data output; used for: • Dual and Quad SSC MTSR1/MRST1 • Can be ignored for all other protocols
USIC1_CH1.DOUT2	O	not connected	Shift data output; used for: • Quad SSC MTSR2/MRST2 • Can be ignored for all other protocols

Table 14-27 USIC Module 1 Channel 1 Interconnects (cont'd)

Input/Output	I/O	Connected To	Description
USIC1_CH1.DOUT3	O	not connected	Shift data output; used for: • Quad SSC MTSR3/MRST3 • Can be ignored for all other protocols
Clock Outputs			
USIC1_CH1.MCLKOUT	O	not connected	Master clock output (optional for all protocols)
USIC1_CH1.SCLKOUT	O	P0.10 P1.8	Shift clock output; used for: • Master SCLKOUT in SSC and IIS • IIC SCL • Can be ignored in ASC
Control Outputs			
USIC1_CH1.SELO0	O	P0.9	Shift control output; used for: • SSC Master SELO • IIS WA • Can be ignored for all other protocols
USIC1_CH1.SELO1	O	P0.2	
USIC1_CH1.SELO2	O	P1.7	
USIC1_CH1.SELO3	O	not connected	
USIC1_CH1.SELO4	O	not connected	
USIC1_CH1.SELO5	O	not connected	
USIC1_CH1.SELO6	O	not connected	
USIC1_CH1.SELO7	O	not connected	
System Related Outputs			
USIC1_CH1.DX0INS	O	USIC1_CH1.DX1F	Selected DX0 input signal
USIC1_CH1.DX1INS	O	not connected	Selected DX1 input signal
USIC1_CH1.DX2INS	O	not connected	Selected DX2 input signal
USIC1_CH1.DX3INS	O	not connected	Selected DX3 input signal
USIC1_CH1.DX4INS	O	not connected	Selected DX4 input signal
USIC1_CH1.DX5INS	O	not connected	Selected DX5 input signal

Table 14-28 USIC Module 1 Module Interconnects

Input/Output	I/O	Connected To	Description
USIC1_SR[1:0]	O	NVIC GPDMA	interrupt output lines (service requests SRx)
USIC1_SR[5:2]	O	NVIC	interrupt output lines (service requests SRx)

15 Controller Area Network Controller (MultiCAN)

This chapter describes the MultiCAN controller of the XMC4[12]00. It contains the following sections:

- CAN basics (see **Page 15-5**)
- Overview of the CAN Module in the XMC4[12]00 (see **Page 15-4**)
- Functional description of the MultiCAN Kernel (see **Page 15-14**)
- MultiCAN Kernel register description (see **Page 15-59**)
- XMC4[12]00 implementation-specific details are listed below,
  - Service Request Generation (see **Page 15-53**)
  - Debug Behavior (see **Page 15-55**)
  - Power, Reset and Clock (see **Page 15-56**)
  - Interconnects (see **Page 15-120**)

Note: The MultiCAN register names described in this chapter are referenced in the XMC4[12]00 Reference Manual by the module name prefix “CAN_”.

15.1 Overview

The MultiCAN module contains independently operating CAN nodes with Full-CAN functionality that are able to exchange Data and Remote Frames via a gateway function. Transmission and reception of CAN frames is handled in accordance with CAN specification V2.0 B (active). Each CAN node can receive and transmit standard frames with 11-bit identifiers as well as extended frames with 29-bit identifiers.

All CAN nodes share a common set of message objects. Each message object can be individually allocated to one of the CAN nodes. Besides serving as a storage container for incoming and outgoing frames, message objects can be combined to build gateways between the CAN nodes or to setup a FIFO buffer.

The message objects are organized in double-chained linked lists, where each CAN node has its own list of message objects. A CAN node stores frames only into message objects that are allocated to the message object list of the CAN node, and it transmits only messages belonging to this message object list. A powerful, command-driven list controller performs all message object list operations.

The bit timings for the CAN nodes are derived from the module timer clock (f_{CAN}), and are programmable up to a data rate of 1 Mbit/s. External bus transceivers are connected to a CAN node via a pair of receive and transmit pins.

15.1.1 Features

The MultiCAN module provides the following functionality:

- 2 independent CAN nodes and 64 message objects available.
- Compliant with ISO 11898
- CAN functionality according to CAN specification V2.0 B active
- Dedicated control registers for each CAN node
- Data transfer rates up to 1 Mbit/s
- Flexible and powerful message transfer control and error handling capabilities
- Advanced CAN bus bit timing analysis and baud rate detection for each CAN node via a frame counter
- Full-CAN functionality: A set of 64 message objects can be individually
  - Allocated (assigned) to any CAN node
  - Configured as transmit or receive object
  - Set up to handle frames with 11-bit or 29-bit identifier
  - Identified by a timestamp via a frame counter
  - Configured to remote monitoring mode
- Advanced acceptance filtering
  - Each message object provides an individual acceptance mask to filter incoming frames
  - A message object can be configured to accept standard or extended frames or to accept both standard and extended frames

  - Message objects can be grouped into four priority classes for transmission and reception
  - The selection of the message to be transmitted first can be based on frame identifier, IDE bit and RTR bit according to CAN arbitration rules, or according to its order in the list
- Advanced message object functionality
  - Message objects can be combined to build FIFO message buffers of arbitrary size, limited only by the total number of message objects
  - Message objects can be linked to form a gateway that automatically transfers frames between two different CAN buses. A single gateway can link any two CAN nodes. An arbitrary number of gateways can be defined.
- Advanced data management
  - The message objects are organized in double-chained lists
  - List reorganizations can be performed at any time, even during full operation of the CAN nodes
  - A powerful, command-driven list controller manages the organization of the list structure and ensures consistency of the list
  - Message FIFOs are based on the list structure and can easily be scaled in size during CAN operation
  - Static allocation commands offer compatibility with TwinCAN applications that are not list-based
- Advanced interrupt handling
  - Up to 8 interrupt output lines are available. Interrupt requests can be routed individually to one of the 8 interrupt output lines
  - Message post-processing notifications can be mapped flexibly using dedicated registers consisting of notification bits

15.1.2 Block Diagram

This section describes the serial communication module called MultiCAN (CAN = Controller Area Network) of the XMC4[12]00. A MultiCAN module can contain between two and eight independent CAN nodes, depending on the device, each representing one serial communication interface.

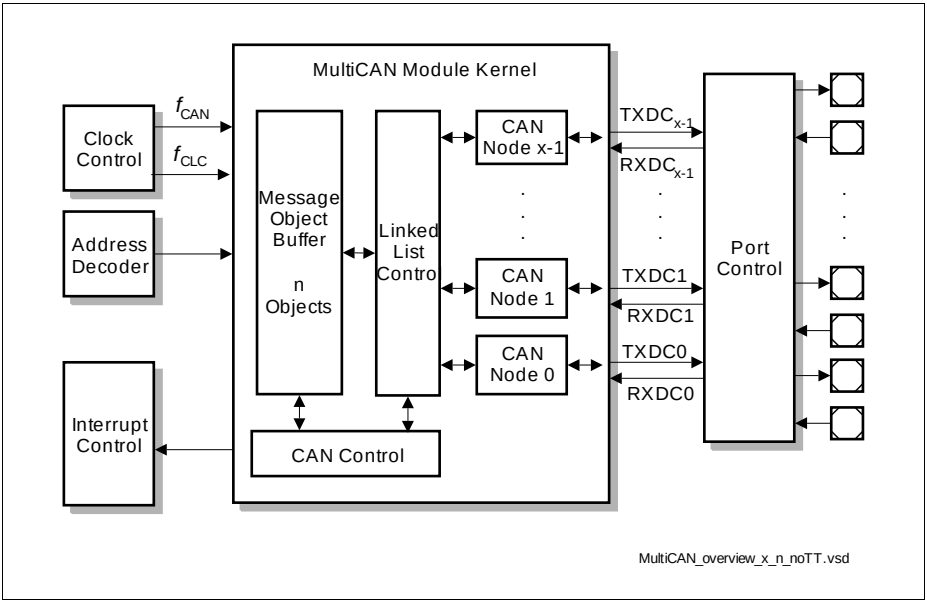


Figure 15-1 Overview of the MultiCAN Module

15.2 CAN Basics

CAN is an asynchronous serial bus system with one logical bus line. It has an open, linear bus structure with equal bus participants called nodes. A CAN bus consists of two or more nodes.

The bus logic corresponds to a “wired-AND” mechanism. Recessive bits (equivalent to the logic 1 level) are overwritten by dominant bits (logic 0 level). As long as no bus node is sending a dominant bit, the bus is in the recessive state. In this state, a dominant bit from any bus node generates a dominant bus state. The maximum CAN bus speed is, by definition, 1 Mbit/s. This speed limits the CAN bus to a length of up to 40 m. For bus lengths longer than 40 m, the bus speed must be reduced.

The binary data of a CAN frame is coded in NRZ code (Non-Return-to-Zero). To ensure re-synchronization of all bus nodes, bit stuffing is used. This means that during the transmission of a message, a maximum of five consecutive bits can have the same polarity. Whenever five consecutive bits of the same polarity have been transmitted, the transmitter will insert one additional bit (stuff bit) of the opposite polarity into the bit stream before transmitting further bits. The receiver also checks the number of bits with the same polarity and removes the stuff bits from the bit stream (= destuffing).

15.2.1 Addressing and Bus Arbitration

In the CAN protocol, address information is defined in the identifier field of a message. The identifier indicates the contents of the message and its priority. The lower the binary value of the identifier, the higher is the priority of the message.

For bus arbitration, CSMA/CD with NDA (Carrier Sense Multiple Access/Collision Detection with Non-Destructive Arbitration) is used. If bus node A attempts to transmit a message across the network, it first checks that the bus is in the idle state (“Carrier Sense”) i.e. no node is currently transmitting. If this is the case (and no other node wishes to start a transmission at the same moment), node A becomes the bus master and sends its message. All other nodes switch to receive mode during the first transmitted bit (Start-Of-Frame bit). After correct reception of the message (acknowledged by each node), each bus node checks the message identifier and stores the message, if required. Otherwise, the message is discarded.

If two or more bus nodes start their transmission at the same time (“Multiple Access”), bus collision of the messages is avoided by bit-wise arbitration (“Collision Detection / Non-Destructive Arbitration” together with the “Wired-AND” mechanism, dominant bits override recessive bits). Each node that sends also reads back the bus level. When a recessive bit is sent but a dominant one is read back, bus arbitration is lost and the transmitting node switches to receive mode. This condition occurs for example when the message identifier of a competing node has a lower binary value and therefore sends a message with a higher priority. In this way, the bus node with the highest priority message wins arbitration without losing time by having to repeat the message. Other nodes that lost arbitration will automatically try to repeat their transmission once the bus

returns to idle state. Therefore, the same identifier can be sent in a Data Frame only by one node in the system. There must not be more than one node programmed to send Data Frames with the same identifier.

Standard message identifier has a length of 11 bits. CAN specification 2.0B extends the message identifier lengths to 29 bits, i.e. the extended identifier.

15.2.2 CAN Frame Formats

There are three types of CAN frames:

- Data Frames
- Remote Frames
- Error Frames

A Data Frame contains a Data Field of 0 to 8 bytes in length. A Remote Frame contains no Data Field and is typically generated as a request for data (e.g. from a sensor). Data and Remote Frames can use an 11-bit “Standard” identifier or a 29-bit “Extended” identifier. An Error Frame can be generated by any node that detects a CAN bus error.

15.2.2.1 Data Frames

There are two types of Data Frames defined (see **Figure 15-2**):

- Standard Data Frame
- Extended Data Frame

Standard Data Frame

A Data Frame begins with the Start-Of-Frame bit (SOF = dominant level) for hard synchronization of all nodes. The SOF is followed by the Arbitration Field consisting of 12 bits, the 11-bit identifier (reflecting the contents and priority of the message), and the RTR (Remote Transmission Request) bit. With RTR at dominant level, the frame is marked as Data Frame. With RTR at recessive level, the frame is defined as a Remote Frame.

The next field is the Control Field consisting of 6 bits. The first bit of this field is the IDE (Identifier Extension) bit and is at dominant level for the Standard Data Frame. The following bit is reserved and defined as a dominant bit. The remaining 4 bits of the Control Field are the Data Length Code (DLC) that specifies the number of bytes in the Data Field. The Data Field can be 0 to 8 bytes wide. The Cyclic Redundancy (CRC) Field that follows the data bytes is used to detect possible transmission errors. It consists of a 15-bit CRC sequence, completed by a recessive CRC delimiter bit.

The final field is the Acknowledge Field. During the ACK Slot, the transmitting node sends out a recessive bit. Any node that has received an error free frame acknowledges the correct reception of the frame by sending back a dominant bit, regardless of whether or not the node is configured to accept that specific message. This behavior assigns the

CAN protocol to the “in-bit-response” group of protocols. The recessive ACK delimiter bit, which must not be overwritten by a dominant bit, completes the Acknowledge Field.

Seven recessive End-of-Frame (EOF) bits finish the Data Frame. Between any two consecutive frames, the bus must remain in the recessive state for at least 3 bit times (called Inter Frame Space). If after the Inter Frame Space, no other nodes attempt to transmit the bus remains in idle state with a recessive level.

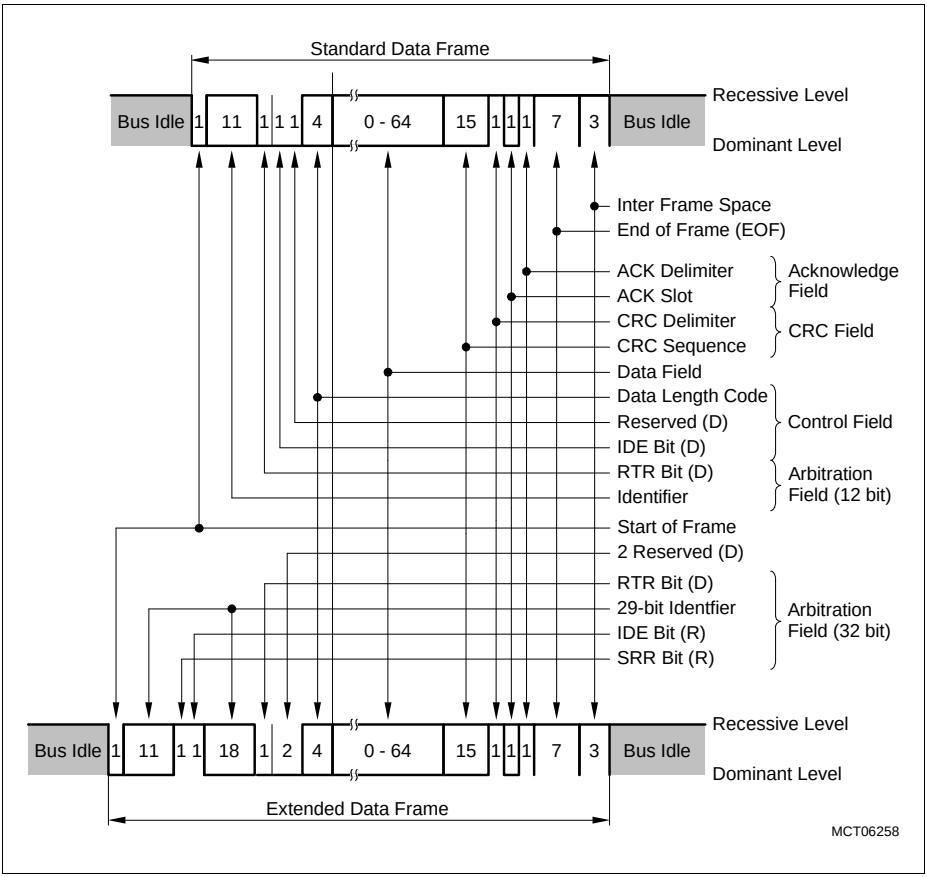


Figure 15-2 CAN Data Frame

Extended Data Frame

In the Extended CAN Data Frame, the message identifier of the standard frame has been extended to 29-bit. A split of the extended identifier into two parts, an 11-bit least

significant section (as in standard CAN frame) and an 18-bit most significant section, ensures that the Identifier Extension bit (IDE) can remain at the same bit position in both standard and extended frames.

In the Extended CAN Data Frame, the SOF bit is followed by the 32-bit Arbitration Field. The first 11 bits are the least significant bits of the 29-bit Identifier (“Base-ID”). These 11 bits are followed by the recessive Substitute Remote Request (SRR) bit. The SRR is further followed by the recessive IDE bit, which indicates the frame to be an Extended CAN frame. If arbitration remains unresolved after transmission of the first 11 bits of the identifier, and if one of the nodes involved in arbitration is sending a Standard CAN frame, then the Standard CAN frame will win arbitration due to the assertion of its dominant IDE bit. Therefore, the SRR bit in an Extended CAN frame is recessive to allow the assertion of a dominant RTR bit by a node that is sending a Standard CAN Remote Frame. The SRR and IDE bits are followed by the remaining 18 bits of the extended identifier and the RTR bit.

Control field and frame termination is identical to the Standard Data Frame.

15.2.2.2 Remote Frames

Normally, data transmission is performed on an autonomous basis with the data source node (e.g. a sensor) sending out a Data Frame. It is also possible, however, for a destination node (or nodes) to request the data from the source. For this purpose, the destination node sends a Remote Frame with an identifier that matches the identifier of the required Data Frame. The appropriate data source node will then send a Data Frame as a response to this remote request.

There are 2 differences between a Remote Frame and a Data Frame.

- The RTR bit is in the recessive state in a Remote Frame.
- There is no Data Field in a Remote Frame.

If a Data Frame and a Remote Frame with the same identifier are transmitted at the same time, the Data Frame wins arbitration due to the dominant RTR bit following the identifier. In this way, the node that transmitted the Remote Frame receives the requested data immediately. The format of a Standard and Extended Remote Frames is shown in **Figure 15-3**.

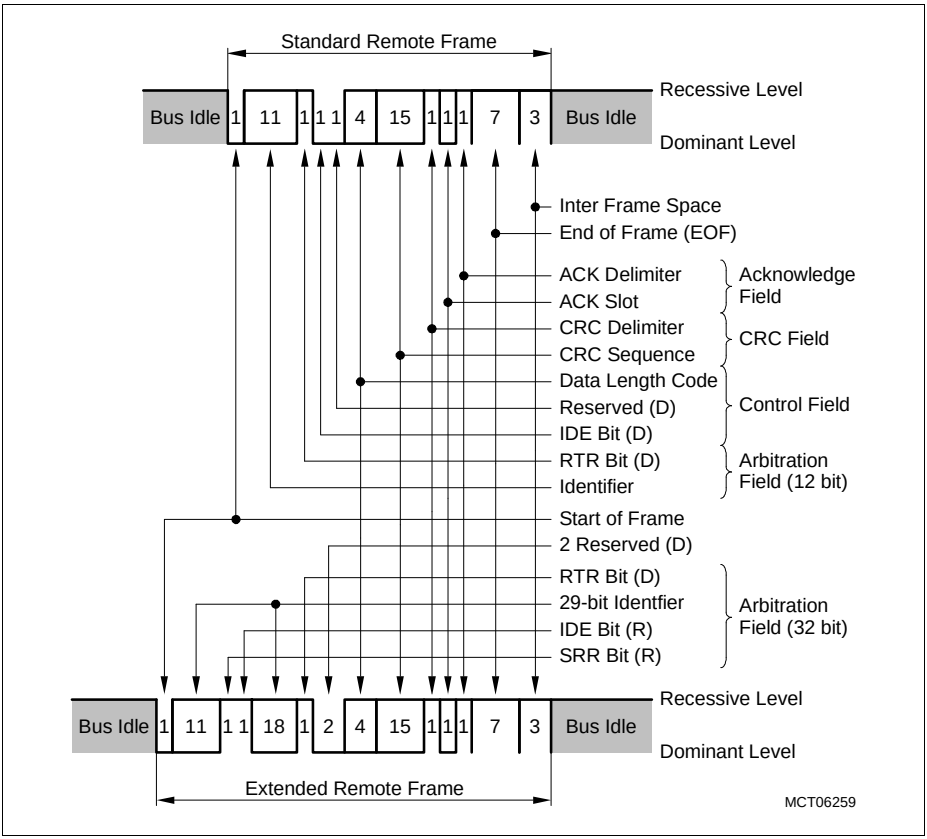


Figure 15-3 CAN Remote Frame

15.2.2.3 Error Frames

An Error Frame is generated by any node that detects a bus error. An Error Frame consists of two fields, an Error Flag field followed by an Error Delimiter field. The Error Delimiter Field consists of 8 recessive bits and allows the bus nodes to restart bus communications after an error. There are, however, two forms of Error Flag fields. The form of the Error Flag field depends on the error status of the node that detects the error.

When an error-active node detects a bus error, the node generates an Error Frame with an active-error flag. The error-active flag is composed of six consecutive dominant bits that actively violate the bit-stuffing rule. All other stations recognize a bit-stuffing error and generate Error Frames themselves. The resulting Error Flag field on the CAN bus therefore consists of six to twelve consecutive dominant bits (generated by one or more nodes). The Error Delimiter field completes the Error Frame. After completion of the Error Frame, bus activity returns to normal and the interrupted node attempts to re-send the aborted message.

If an error-passive node detects a bus error, the node transmits an error-passive flag followed, again, by the Error Delimiter field. The error-passive flag consists of six consecutive recessive bits, and therefore the Error Frame (for an error-passive node) consists of 14 recessive bits (i.e. no dominant bits). Therefore, the transmission of an Error Frame by an error-passive node will not affect any other node on the network, unless the bus error is detected by the node that is actually transmitting (i.e. the bus master). If the bus master node generates an error-passive flag, this may cause other nodes to generate Error Frames due to the resulting bit-stuffing violation. After transmission of an Error Frame an error-passive node must wait for 6 consecutive recessive bits on the bus before attempting to rejoin bus communications.

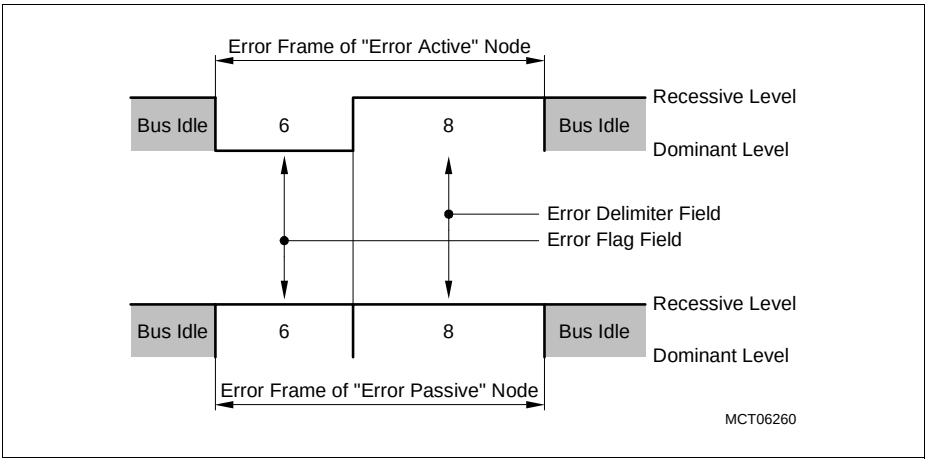


Figure 15-4 CAN Error Frames

15.2.3 The Nominal Bit Time

One bit cell (this means one high or low pulse of the NRZ code) is composed by four segments. Each segment is an integer multiple of Time Quanta t_Q. The Time Quanta is the smallest discrete timing resolution used by a CAN node. The nominal bit time definition with its segments is shown in **Figure 15-5**.

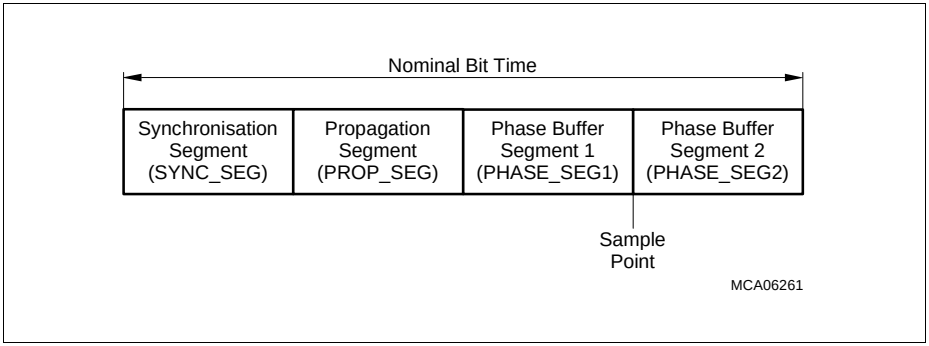


Figure 15-5 Partition of Nominal Bit Time

The Synchronization Segment (SYNC_SEG) is used to synchronize the various bus nodes. If there is a bit state change between the previous bit and the current bit, then the bus state change is expected to occur within this segment. The length of this segment is always 1 t_Q.

The Propagation Segment (PROP_SEG) is used to compensate for signal delays across the network. These delays are caused by signal propagation delay on the bus line and through the electronic interface circuits of the bus nodes.

The Phase Segments 1 and 2 (PHASE_SEG1, PHASE_SEG2) are used to compensate for edge phase errors. These segments can be lengthened or shortened by re-synchronization. PHASE_SEG2 is reserved for calculation of the subsequent bit level, and is ≥ 2 t_Q. At the sample point, the bus level is read and interpreted as the value of the bit cell. It occurs at the end of PHASE_SEG1.

The total number of t_Q in a bit time is between 8 and 25.

As a result of re-synchronization, PHASE_SEG1 can be lengthened or PHASE_SEG2 can be shortened. The amount of lengthening or shortening the phase buffer segments has an upper limit given by the re-synchronization jump width. The re-synchronization jump width may be between 1 and 4 t_Q, but it may not be longer than PHASE_SEG1.

15.2.4 Error Detection and Error Handling

The CAN protocol has sophisticated error detection mechanisms. The following errors can be detected:

- **Cyclic Redundancy Check (CRC) Error**
  With the Cyclic Redundancy Check, the transmitter calculates special check bits for the bit sequence from the start of a frame until the end of the Data Field. This CRC sequence is transmitted in the CRC Field. The receiving node also calculates the CRC sequence using the same formula, and performs a comparison to the received sequence. If a mismatch is detected, a CRC error has occurred and an Error Frame is generated. The message is repeated.
- **Acknowledge Error**
  In the Acknowledge Field of a message, the transmitter checks whether a dominant bit is read during the Acknowledge Slot (that is sent out as a recessive bit). If not, no other node has received the frame correctly, an Acknowledge Error has occurred, and the message must be repeated. No Error Frame is generated.
- **Form Error**
  If a transmitter detects a dominant bit in one of the four segments End of Frame, Interframe Space, Acknowledge Delimiter, or CRC Delimiter, a Form Error has occurred, and an Error Frame is generated. The message is repeated.
- **Bit Error**
  A Bit Error occurs if a) a transmitter sends a dominant bit and detects a recessive bit or b) if the transmitter sends a recessive bit and detects a dominant bit when monitoring the actual bus level and comparing it to the just transmitted bit. In case b), no error occurs during the Arbitration Field (ID, RTR, IDE) and the Acknowledge Slot.
- **Stuff Error**
  If between Start of Frame and CRC Delimiter, six consecutive bits with the same polarity are detected, the bit-stuffing rule has been violated. A stuff error occurs and an Error Frame is generated. The message is repeated.

Detected errors are made public to all other nodes via Error Frames (except Acknowledge Errors). The transmission of the erroneous message is aborted and the frame is repeated as soon as possible. Furthermore, each CAN node is in one of the three error states (error-active, error-passive or bus-off) according to the value of the internal error counters. The error-active state is the usual state where the bus node can transmit messages and active-error frames (made of dominant bits) without any restrictions. In the error-passive state, messages and passive-error frames (made of recessive bits) may be transmitted. The bus-off state makes it temporarily impossible for the node to participate in the bus communication. During this state, messages can be neither received nor transmitted.

Basic CAN, Full CAN

There is one more CAN characteristic that is related to the interface of a CAN module (controller) and the host CPU: Basic-CAN and Full-CAN functionality.

In Basic-CAN devices, only basic functions of the protocol are implemented in hardware, such as the generation and the check of the bit stream. The decision, whether a received message has to be stored or not (acceptance filtering), and the complete message management must be done by software. Normally, the CAN device also provides only one transmit buffer and one or two receive buffers. Therefore, the host CPU load is quite high when using Basic-CAN modules. The main advantage of Basic-CAN is a reduced chip size leading to low costs of these devices.

Full-CAN devices (this is the case for the MultiCAN controller as implemented in XMC4[12]00) manage the whole bus protocol in hardware, including the acceptance filtering and message management. Full-CAN devices contain message objects that handle autonomously the identifier, the data, the direction (receive or transmit) and the information of Standard CAN/Extended CAN operation. During the initialization of the device, the host CPU determines which messages are to be sent and which are to be received. The host CPU is informed by interrupt if the identifier of a received message matches with one of the programmed (receive-) message objects. The CPU load of Full-CAN devices is greatly reduced. When using Full-CAN devices, high baud rates and high bus loads with many messages can be handled.

15.3 MultiCAN Kernel Functional Description

This section describes the functionality of the MultiCAN module.

15.3.1 Module Structure

Figure 15-6 shows the general structure of the MultiCAN module.

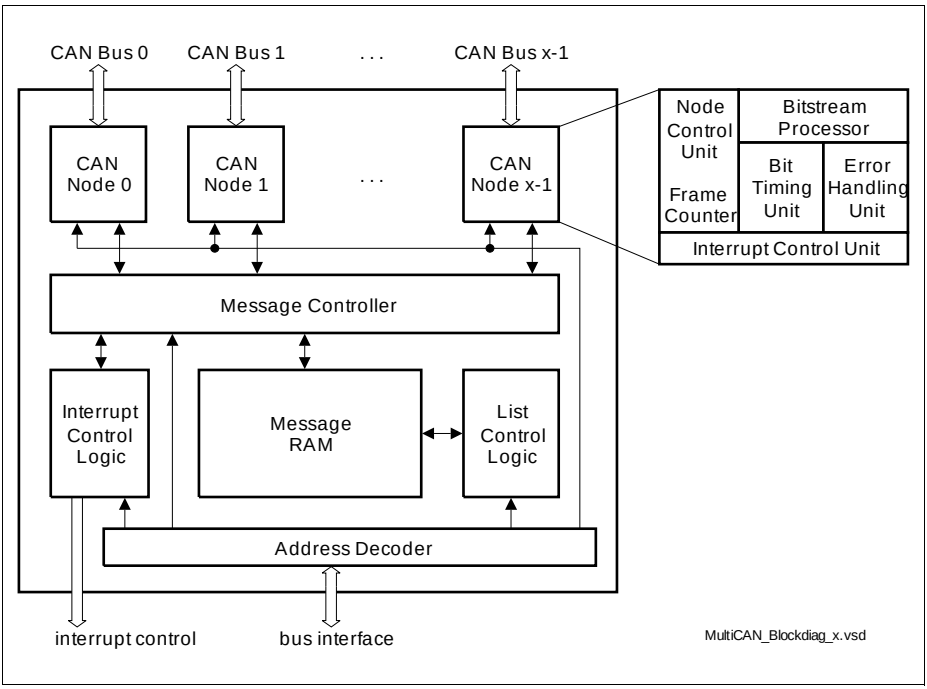


Figure 15-6 MultiCAN Block Diagram

CAN Nodes

Each CAN node consists of several sub-units.

- **Bitstream Processor**
  The Bitstream Processor performs data, remote, error and overload frame processing according to the ISO 11898 standard. This includes conversion between the serial data stream and the input/output registers.
- **Bit Timing Unit**
  The Bit Timing Unit determines the length of a bit time and the location of the sample point according to the user settings, taking into account propagation delays and phase shift errors. The Bit Timing Unit also performs resynchronization.

- **Error Handling Unit**
  The Error Handling Unit manages the receive and transmit error counter. Depending on the contents of both counters, the CAN node is set into an error-active, error passive or bus-off state.
- **Node Control Unit**
  The Node Control Unit coordinates the operation of the CAN node:
  – Enable/disable CAN transfer of the node
  – Enable/disable and generate node-specific events that lead to an interrupt request (CAN bus errors, successful frame transfers etc.)
  – Administration of the Frame Counter
- **Interrupt Control Unit**
  The Interrupt Control Unit in the CAN node controls the interrupt generation for the different conditions that can occur in the CAN node.

Message Controller

The Message Controller handles the exchange of CAN frames between the CAN nodes and the message objects that are stored in the Message RAM. The Message Controller performs several functions:

- Receive acceptance filtering to determine the correct message object for storing of a received CAN frame
- Transmit acceptance filtering to determine the message object to be transmitted first, individually for each CAN node
- Transfer contents between message objects and the CAN nodes, taking into account the status/control bits of the message objects
- Handling of the FIFO buffering and gateway functionality
- Aggregation of message-pending notification bits

List Controller

The List Controller performs all operations that lead to a modification of the double-chained message object lists. Only the list controller is allowed to modify the list structure. The allocation/deallocation or reallocation of a message object can be requested via a user command interface (command panel). The list controller state machine then performs the requested command autonomously.

Interrupt Control

The general interrupt structure is shown in **Figure 15-7**. The interrupt event can trigger the interrupt generation. The interrupt pulse is generated independently of the interrupt flag in the interrupt status register. The interrupt flag can be reset by software by writing a 0 to it.

If enabled by the related interrupt enable bit in the interrupt enable register, an interrupt pulse can be generated at one of the 16 interrupt output lines INT_Om of the MultiCAN

module. If more than one interrupt source is connected to the same interrupt node pointer (in the interrupt node pointer register), the requests are combined to one common line.

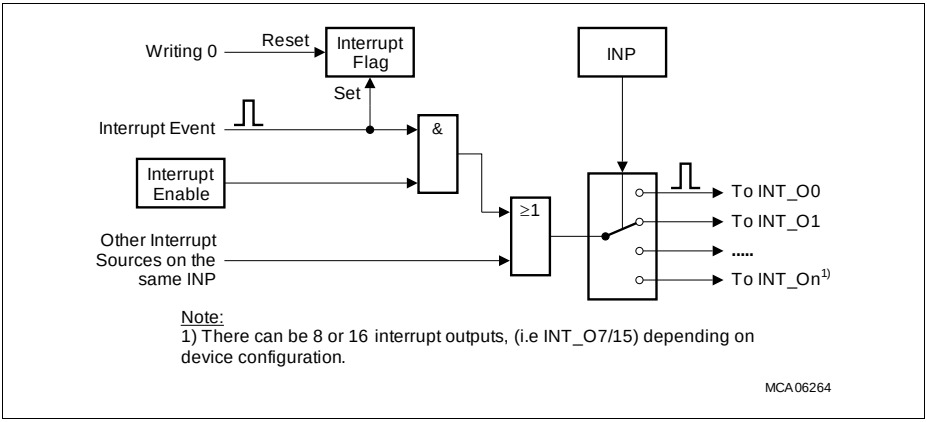


Figure 15-7 General Interrupt Structure

15.3.2 Port Input Control

It is possible to select the input lines for the RXDCx inputs for the CAN nodes. The selected input is connected to the CAN node and is also available to wake-up the system. More details are defined in **Section 15.8.2.1** on **Page 15-121**.

15.3.3 CAN Node Control

Each CAN node may be configured and run independently of the other CAN node. Each CAN node is equipped with its own node control logic to configure the global behavior and to provide status information.

Note: In the following descriptions, index “x” stands for the node number and index “n” represents the message object number.

Configuration Mode is activated when bit NCRx.CCE is set to 1. This mode allows CAN bit timing parameters and the error counter registers to be modified.

CAN Analyzer Mode is activated when bit NCRx.CALM is set to 1. In this operation mode, Data And Remote Frames are monitored without active participation in any CAN transfer (CAN transmit pin is held on recessive level). Incoming Remote Frames are stored in a corresponding transmit message object, while arriving data frames are saved in a matching receive message object.

In CAN Analyzer Mode, the entire configuration information of the received frame is stored in the corresponding message object, and can be evaluated by the CPU to determine their identifier, XTD bit information and data length code (ID and DLC optionally if the Remote Monitoring Mode is active, bit MOFCRn.RMM = 1). Incoming frames are not acknowledged, and no Error Frames are generated. If CAN Analyzer Mode is enabled, Remote Frames are not responded to by the corresponding Data Frame, and Data Frames cannot be transmitted by setting the transmit request bit MOSTATn.TXRQ. Receive interrupts are generated in CAN Analyzer Mode (if enabled) for all error free received frames.

The node-specific interrupt configuration is also defined by the Node Control Logic via the NCRx register bits TRIE, ALIE and LECIE:

- If control bit TRIE is set to 1, a transfer interrupt is generated when the NSRx register has been updated (after each successfully completed message transfer).
- If control bit ALIE is set to 1, an error interrupt is generated when a “bus-off” condition has been recognized or the Error Warning Level has been exceeded or under-run. Additionally, list or object errors lead to this type of interrupt.
- If control bit LECIE is set to 1, a last error code interrupt is generated when an error code > 0 is written into bit field NSRx.LEC by hardware.

The Node x Status Register NSRx provides an overview about the current state of the respective CAN node x, comprising information about CAN transfers, CAN node status, and error conditions.

The CAN frame counter can be used to check the transfer sequence of message objects or to obtain information about the instant a frame has been transmitted or received from the associated CAN bus. CAN frame counting is performed by a 16-bit counter, controlled by register NFCRx. Bit fields NFCRx.CFMOD and NFCRx.CFSEL determine the operation mode and the trigger event incrementing the frame counter.

15.3.3.1 Bit Timing Unit

According to the ISO 11898 standard, a CAN bit time is subdivided into different segments (**Figure 15-8**). Each segment consists of multiples of a time quantum t_q. The magnitude of t_q is adjusted by Node x Bit Timing Register bit fields NBTRx.BRP and NBTRx.DIV8, both controlling the baud rate prescaler (register NBTRx is described on **Page 15-85**). The baud rate prescaler is driven by the module timer clock f_{CAN} (generation and control of f_{CAN} is described on **Page 15-58**).

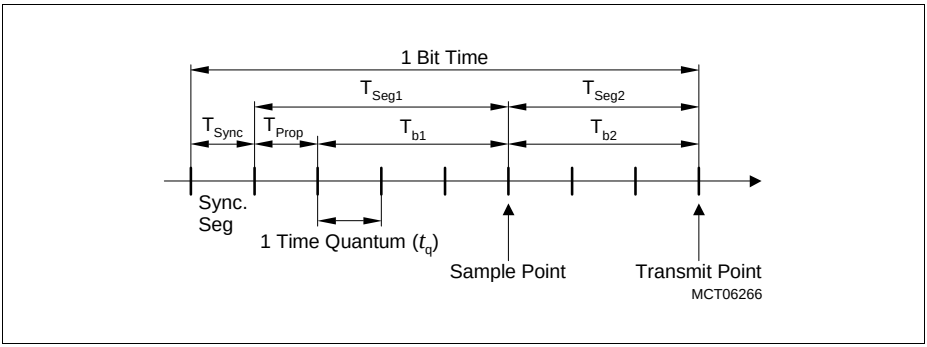


Figure 15-8 CAN Bus Bit Timing Standard

The Synchronization Segment (T_{Sync}) allows a phase synchronization between transmitter and receiver time base. The Synchronization Segment length is always one t_q. The Propagation Time Segment (T_{Prop}) takes into account the physical propagation delay in the transmitter output driver on the CAN bus line and in the transceiver circuit. For a working collision detection mechanism, T_{Prop} must be two times the sum of all propagation delay quantities rounded up to a multiple of t_q. The phase buffer segments 1 and 2 (T_{b1}, T_{b2}) before and after the signal sample point are used to compensate for a mismatch between transmitter and receiver clock phases detected in the synchronization segment.

The maximum number of time quanta allowed for re-synchronization is defined by bit field NBTRx.SJW. The Propagation Time Segment and the Phase Buffer Segment 1 are combined to parameter T_{Seg1}, which is defined by the value NBTRx.TSEG1. A minimum of 3 time quanta is demanded by the ISO standard. Parameter T_{Seg2}, which is defined by the value of NBTRx.TSEG2, covers the Phase Buffer Segment 2. A minimum of 2 time quanta is demanded by the ISO standard. According to ISO standard, a CAN bit time, calculated as the sum of T_{Sync}, T_{Seg1} and T_{Seg2}, must not fall below 8 time quanta.

Calculation of the bit time:

t_q	= (BRP + 1) / f_{CAN}	if DIV8 = 0
	= 8 × (BRP + 1) / f_{CAN}	if DIV8 = 1
T_{Sync}	= 1 × t_q	
T_{Seg1}	= (TSEG1 + 1) × t_q	(min. 3 t_q)
T_{Seg2}	= (TSEG2 + 1) × t_q	(min. 2 t_q)
bit time	= T_{Sync} + T_{Seg1} + T_{Seg2}	(min. 8 t_q)

To compensate phase shifts between clocks of different CAN controllers, the CAN controller must synchronize on any edge from the recessive to the dominant bus level. If the hard synchronization is enabled (at the start of frame), the bit time is restarted at the synchronization segment. Otherwise, the re-synchronization jump width T_{SJW} defines the maximum number of time quanta, a bit time may be shortened or lengthened by one re-synchronization. The value of SJW is defined by bit field NBTRx.SJW.

T_{SJW}	= (SJW + 1) × t_q
T_{Seg1}	≥ T_{SJW} + T_{prop}
T_{Seg2}	≥ T_{SJW}

The maximum relative tolerance for f_{CAN} depends on the Phase Buffer Segments and the re-synchronization jump width.

df_{CAN}	≤ min (T_{b1}, T_{b2}) / 2 × (13 × bit time - T_{b2})	AND
df_{CAN}	≤ T_{SJW} / 20 × bit time	

A valid CAN bit timing must be written to the CAN Node Bit Timing Register NBTR before resetting the INIT bit in the Node Control Register, i.e. before enabling the operation of the CAN node.

The Node Bit Timing Register may be written only if bit CCE (Configuration Change Enable) is set in the corresponding Node Control Register.

15.3.3.2 Bitstream Processor

Based on the message objects in the message buffer, the Bitstream Processor generates the remote and Data Frames to be transmitted via the CAN bus. It controls the CRC generator and adds the checksum information to the new remote or Data Frame. After including the SOF bit and the EOF field, the Bitstream Processor starts the CAN

bus arbitration procedure and continues with the frame transmission when the bus was found in idle state. While the data transmission is running, the Bitstream Processor continuously monitors the I/O line. If (outside the CAN bus arbitration phase or the acknowledge slot) a mismatch is detected between the voltage level on the I/O line and the logic state of the bit currently sent out by the transmit shift register, a CAN error interrupt request is generated, and the error code is indicated by the Node x Status Register bit field NSRx.LEC.

The data consistency of an incoming frame is verified by checking the associated CRC field. When an error has been detected, a CAN error interrupt request is generated and the associated error code is presented in the Node x Status Register NSRx. Furthermore, an Error Frame is generated and transmitted on the CAN bus. After decomposing a faultless frame into identifier and data portion, the received information is transferred to the message buffer executing remote and Data Frame handling, interrupt generation and status processing.

15.3.3.3 Error Handling Unit

The Error Handling Unit of a CAN node x is responsible for the fault confinement of the CAN device. Its two counters, the Receive Error Counter REC and the Transmit Error Counter TEC (bit fields of the Node x Error Counter Register NECNTx, see **Page 15-87**) are incremented and decremented by commands from the Bitstream Processor. If the Bitstream Processor itself detects an error while a transmit operation is running, the Transmit Error Counter is incremented by 8. An increment of 1 is used when the error condition was reported by an external CAN node via an Error Frame generation. For error analysis, the transfer direction of the disturbed message and the node that recognizes the transfer error are indicated for the respective CAN node x in register NECNTx. Depending on the values of the error counters, the CAN node is set into error-active, error-passive, or bus-off state.

The CAN node is in error-active state if both error counters are below the error-passive limit of 128. The CAN node is in error-passive state, if at least one of the error counters is equal to or greater than 128.

The bus-off state is activated if the Transmit Error Counter is equal to or greater than the bus-off limit of 256. This state is reported for CAN node x by the Node x Status Register flag NSRx.BOFF. The device remains in this state, until the “bus-off” recovery sequence is finished. Additionally, Node x Status Register flag NSRx.EWRN is set when at least one of the error counters is equal to or greater than the error warning limit defined by the Node x Error Count Register bit field NECNTx.EWRNLVL. Bit NSRx.EWRN is reset if both error counters fall below the error warning limit again (see **Page 15-78**).

15.3.3.4 CAN Frame Counter

Each CAN node is equipped with a frame counter that counts transmitted/received CAN frames or obtains information about the time when a frame has been started to transmit or be received by the CAN node. CAN frame counting/bit time counting is performed by a 16-bit counter that is controlled by Node x Frame Counter Register NFCRx (see **Page 15-89**). Bit field NFCRx.CFSEL determines the operation mode of the frame counter:

- **Frame Count Mode:**
  After the successful transmission and/or reception of a CAN frame, the frame counter is copied into the CFCVAL bit field of the MOIPRn register of the message object involved in the transfer. Afterwards, the frame counter is incremented.
- **Time Stamp Mode:**
  The frame counter is incremented with the beginning of a new bit time. When the transmission/reception of a frame starts, the value of the frame counter is captured and stored to the CFC bit field of the NFCRx register. After the successful transfer of the frame the captured value is copied to the CFCVAL bit field of the MOIPRn register of the message object involved in the transfer.
- **Bit Timing Mode:**
  Used for baud rate detection and analysis of the bit timing (**Chapter 15.3.5.3**).

15.3.3.5 CAN Node Interrupts

Each CAN node has four hardware triggered interrupt request types that are able to generate an interrupt request upon:

- The successful transmission or reception of a frame
- A CAN protocol error with a last error code
- An alert condition: Transmit/receive error counters reach the warning limit, bus-off state changes, a List Length Error occurs, or a List Object Error occurs
- An overflow of the frame counter

Besides the hardware generated interrupts, software initiated interrupts can be generated using the Module Interrupt Trigger Register MITR. Writing a 1 to bit n of bit field MITR.IT generates an interrupt request signal on the corresponding interrupt output line INT_On. When writing MITR.IT more than one bit can be set resulting in activation of multiple INT_On interrupt output lines at the same time. See also **“Service Request Generation” on Page 15-53** for further processing of the CAN node interrupts.

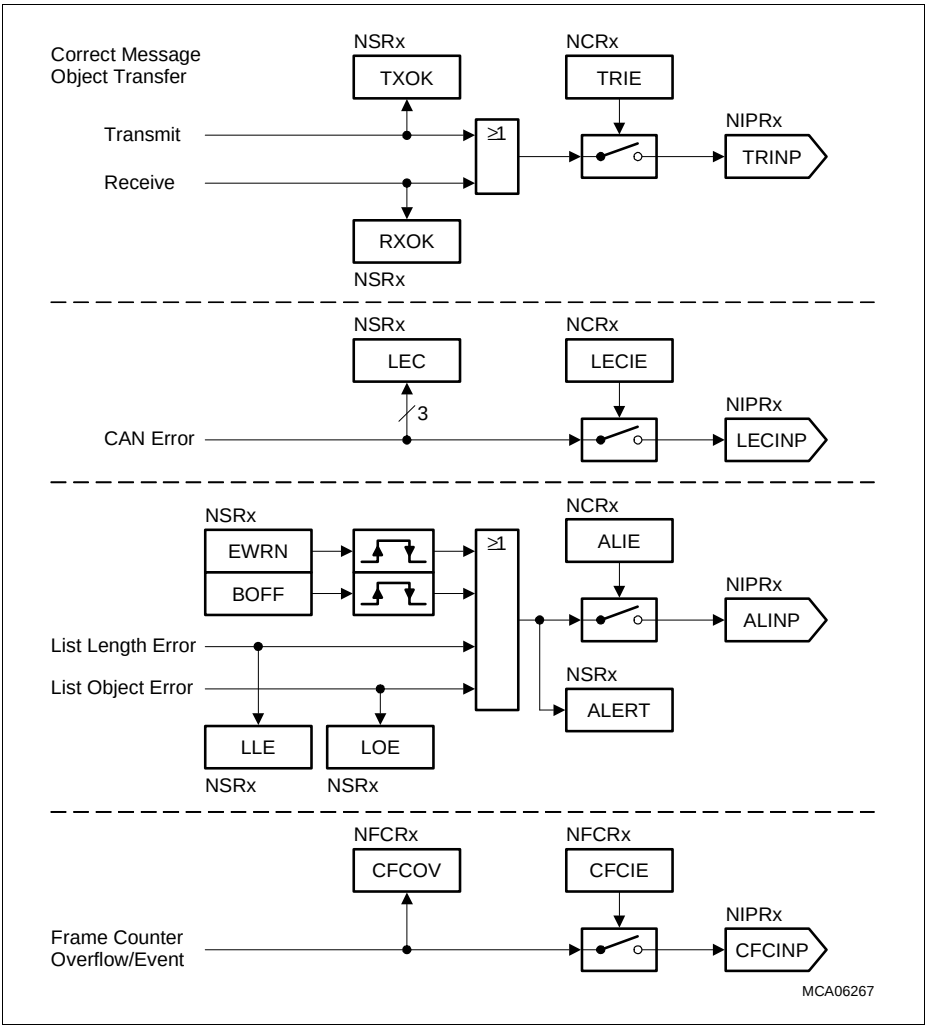


Figure 15-9 CAN Node Interrupts

15.3.4 Message Object List Structure

This section describes the structure of the message object lists in the MultiCAN module.

15.3.4.1 Basics

The message objects of the MultiCAN module are organized in double-chained lists, where each message object has a pointer to the previous message object in the list as well as a pointer to the next message object in the list. The MultiCAN module provides 8 lists. Each message object is allocated to one of these lists. In the example in **Figure 15-10**, the three message objects (3, 5, and 16) are allocated to the list with index 2 (List Register LIST2).

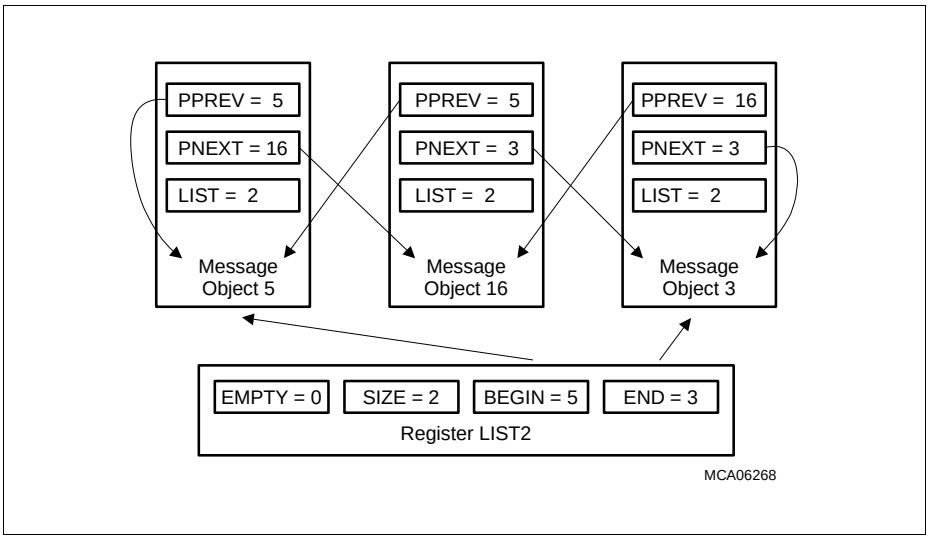


Figure 15-10 Example Allocation of Message Objects to a List

Bit field BEGIN in the List Register (for definition, see **Page 15-69**) points to the first element in the list (object 5 in the example), and bit field END points to the last element in the list (object 3 in the example). The number of elements in the list is indicated by bit field SIZE of the List Register (SIZE = number of list elements - 1, thus SIZE = 2 for the 3 elements in the example). The EMPTY bit of the List Register indicates whether or not a list is empty (EMPTY = 0 in the example, because list 2 is not empty).

Each message object n has a pointer PNEXT in its Message Object n Control Register MOCTRn (see **Page 15-93**) that points to the next message object in the list, and a pointer PPREV that points to the previous message object in the list. PPREV of the first message object points to the message object itself because the first message object has no predecessor (in the example message object 5 is the first message object in the list,

indicated by PPREV = 5). PNEXT of the last message object also points to the message object itself because the last message object has no successor (in the example, object 3 is the last message object in the list, indicated by PNEXT = 3).

Bit field MOCTRn.LIST indicates the list index number to which the message object is currently allocated. The message object of the example are allocated to list 2. Therefore, all LIST bit fields for the message objects assigned to list 2 are set to LIST = 2.

15.3.4.2 List of Unallocated Elements

The list with list index 0 has a special meaning: it is the list of all unallocated elements. An element is called unallocated if it belongs to list 0 (MOCTRn.LIST = 0). It is called allocated if it belongs to a list with an index not equal to 0 (MOCTRn.LIST > 0).

After reset, all message objects are unallocated. This means that they are assigned to the list of unallocated elements with MOCTRn.LIST = 0. After this initial allocation of the message objects caused by reset, the list of all unallocated message objects is ordered by message number (predecessor of message object n is object n-1, successor of object n is object n+1).

15.3.4.3 Connection to the CAN Nodes

Each CAN node is linked to one unique list of message objects. A CAN node performs message transfer only with the message objects that are allocated to the list of the CAN node. This is illustrated in **Figure 15-11**. Frames that are received on a CAN node may only be stored in one of the message objects that belongs to the CAN node; frames to be transmitted on a CAN node are selected only from the message objects that are allocated to that node, as indicated by the vertical arrows.

There are more lists (8) than CAN nodes (2). This means that some lists are not linked to one of the CAN nodes. A message object that is allocated to one of these unlinked lists cannot receive messages directly from a CAN node and it may not transmit messages.

FIFO and gateway mechanisms refer to message numbers and not directly to a specific list. The user must take care that the message objects targeted by FIFO/gateway belong to the desired list. The mechanisms make it possible to work with lists that do not belong to the CAN node.

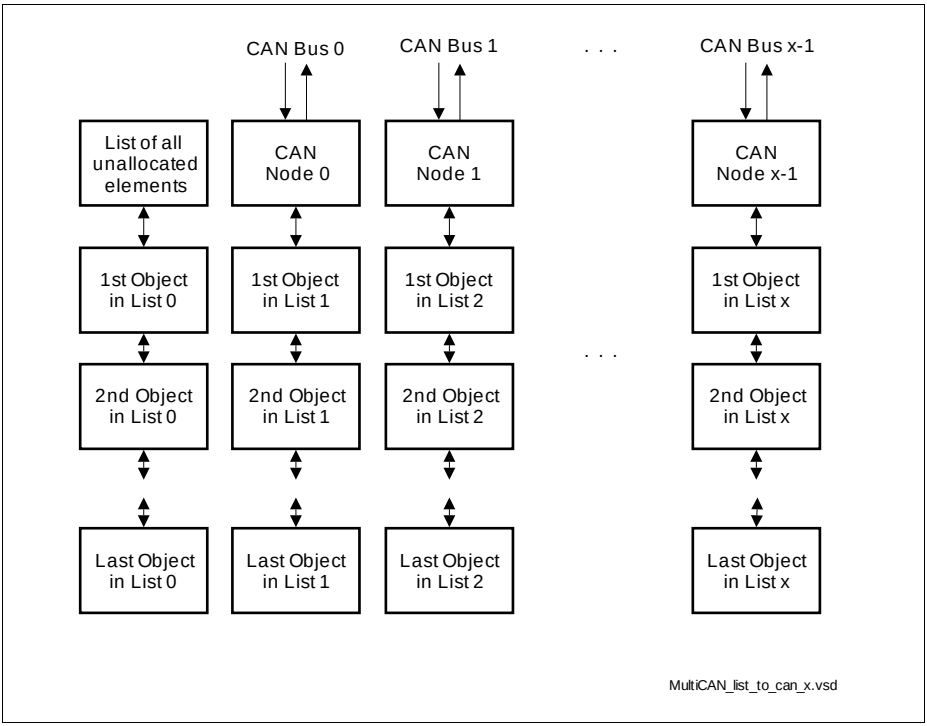


Figure 15-11 Message Objects Linked to CAN Nodes

15.3.4.4 List Command Panel

The list structure cannot be modified directly by write accesses to the LIST registers and the PPREV, PNEXT and LIST bit fields in the Message Object Control Registers, as they are read only. The list structure is managed by and limited to the list controller inside the MultiCAN module. The list controller is controlled via a command panel allowing the user to issue list allocation commands to the list controller. The list controller has two main purposes:

1. Ensure that all operations that modify the list structure result in a consistent list structure.
2. Present maximum ease of use and flexibility to the user.

The list controller and the associated command panel allows the programmer to concentrate on the final properties of the list, which are characterized by the allocation of message objects to a CAN node, and the ordering relation between objects that are allocated to the same list. The process of list (re-)building is done in the list controller.

Table 15-1 gives an overview on the available panel commands while Table 15-6 on Page 15-64 describes the panel commands in more detail.

Table 15-1 Panel Commands Overview

Command Name	Description
No Operation	No new command is started.
Initialize Lists	Run the initialization sequence to reset the CTRL and LIST field of all message objects.
Static Allocate	Allocate message object to a list.
Dynamic Allocate	Allocate the first message object of the list of unallocated objects to the selected list.
Static Insert Before	Remove a message object (source object) from the list that it currently belongs to, and insert it before a given destination object into the list structure of the destination object.
Dynamic Insert Before	Insert a new message object before a given destination object.
Static Insert Behind	Remove a message object (source object) from the list that it currently belongs to, and insert it behind a given destination object into the list structure of the destination object.
Dynamic Insert Behind	Insert a new message object behind a given destination object.

A panel command is started by writing the respective command code into the Panel Control Register bit field PANCTR.PANCMD (see Page 15-63). The corresponding command arguments must be written into bit fields PANCTR.PANAR1 and PANCTR.PANAR2 before writing the command code, or latest along with the command code in a single 32-bit write access to the Panel Control Register.

With the write operation of a valid command code, the PANCTR.BUSY flag is set and further write accesses to the Panel Control Register are ignored. The BUSY flag remains active and the control panel remains locked until the execution of the requested command has been completed. After a reset, the list controller builds up list 0. During this operation, BUSY is set and other accesses to the CAN RAM are forbidden. The CAN RAM can be accessed again when BUSY becomes inactive.

Note: The CAN RAM is automatically initialized after reset by the list controller in order to ensure correct list pointers in each message object. The end of this CAN RAM initialization is indicated by bit PANCTR.BUSY becoming inactive.

In case of a dynamic allocation command that takes an element from the list of unallocated objects, the PANCTR.RBUSY bit is also set along with the BUSY bit

(RBUSY = BUSY = 1). This indicates that bit fields PANAR1 and PANAR2 are going to be updated by the list controller in the following way:

1. The message number of the message object taken from the list of unallocated elements is written to PANAR1.
2. If ERR (bit 7 of PANAR2) is set to 1, the list of unallocated elements was empty and the command is aborted. If ERR is 0, the list was not empty and the command will be performed successfully.

The results of a dynamic allocation command are written before the list controller starts the actual allocation process. As soon as the results are available, RBUSY becomes inactive (RBUSY = 0) again, while BUSY still remains active until completion of the command. This allows the user to set up the new message object while it is still in the process of list allocation. The access to message objects is not limited during ongoing list operations. However, any access to a register resource located inside the RAM delays the ongoing allocation process by one access cycle.

As soon as the command is finished, the BUSY flag becomes inactive (BUSY = 0) and write accesses to the Panel Control Register are enabled again. Also, the “No Operation” command code is automatically written to the PANCTR.PANCMD field. A new command may be started any time when BUSY = 0.

All fields of the Panel Control Register PANCTR except BUSY and RBUSY may be written by the user. This makes it possible to save and restore the Panel Control Register if the Command Panel is used within independent (mutually interruptible) interrupt service routines. If this is the case, any task that uses the Command Panel and that may interrupt another task that also uses the Command Panel should poll the BUSY flag until it becomes inactive and save the whole PANCTR register to a memory location before issuing a command. At the end of the interrupt service routine, the task should restore PANCTR from the memory location.

Before a message object that is allocated to the list of an active CAN node shall be moved to another list or to another position within the same list, bit MOCTRn.MSGVAL (“Message Valid”) of message object n must be cleared.

15.3.5 CAN Node Analysis Features

The chapter describes the CAN node analysis capabilities of the MultiCAN module.

15.3.5.1 Analyzer Mode

The CAN Analyzer Mode makes it possible to monitor the CAN traffic for each CAN node individually without affecting the logical state of the CAN bus. The CAN Analyzer Mode for CAN node x is selected by setting Node x Control Register bit NCRx.CALM.

In CAN Analyzer Mode, the transmit pin of a CAN node is held at a recessive level permanently. The CAN node may receive frames (Data, Remote, and Error Frames) but is not allowed to transmit. Received Data/Remote Frames are not acknowledged (i.e. acknowledge slot is sent recessive) but will be received and stored in matching message objects as long as there is any other node that acknowledges the frame. The complete message object functionality is available, but no transmit request will be executed.

15.3.5.2 Loop-Back Mode

The MultiCAN module provides a Loop-Back Mode to enable an in-system test of the MultiCAN module as well as the development of CAN driver software without access to an external CAN bus.

The loop-back feature consists of an internal CAN bus (inside the MultiCAN module) and a bus select switch for each CAN node (see **Figure 15-12**). With the switch, each CAN node can be connected either to the internal CAN bus (Loop-Back Mode activated) or the external CAN bus, respectively to transmit and receive pins (normal operation). The CAN bus that is not currently selected is driven recessive; this means the transmit pin is held at 1, and the receive pin is ignored by the CAN nodes that are in Loop-Back Mode.

The Loop-Back Mode is selected for CAN node x by setting the Node x Port Control Register bit NPCRx.LBM. All CAN nodes that are in Loop-Back Mode may communicate together via the internal CAN bus without affecting the normal operation of the other CAN nodes that are not in Loop-Back Mode.

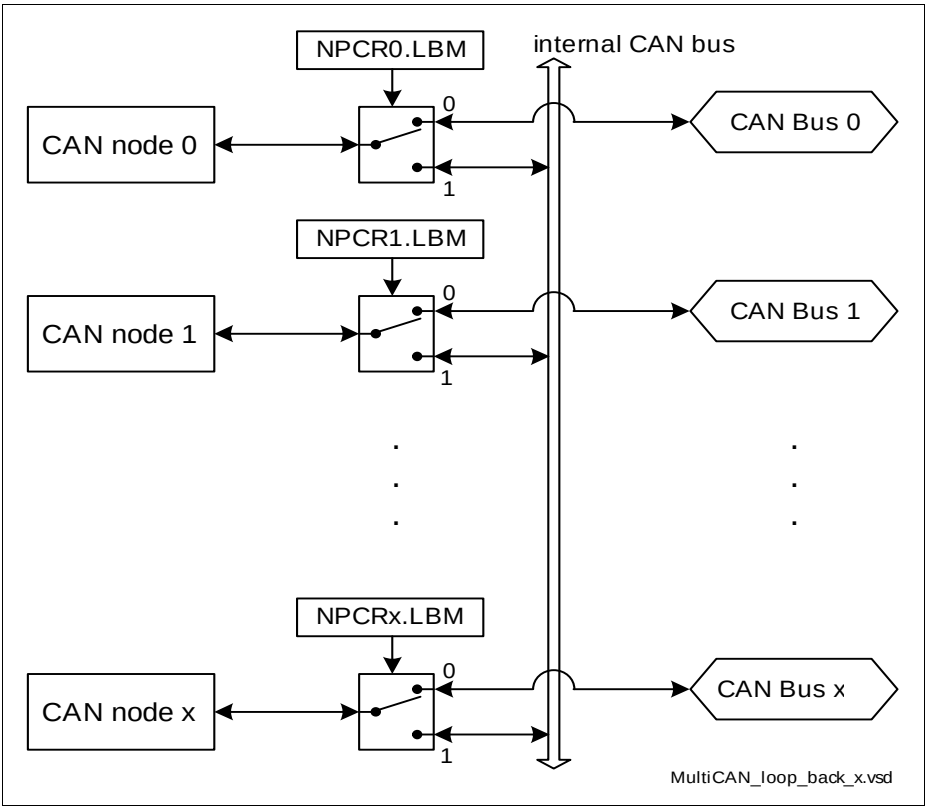


Figure 15-12 Loop-Back Mode

15.3.5.3 Bit Timing Analysis

Detailed analysis of the bit timing can be performed for each CAN node using the analysis modes of the CAN frame counter. The bit timing analysis functionality of the frame counter may be used for automatic detection of the CAN baud rate, as well as to analyze the timing of the CAN network.

Bit timing analysis for CAN node x is selected when bit field NFCRx.CFMOD = 10_B. Bit timing analysis does not affect the operation of the CAN node.

The bit timing measurement results are written into the NFCRx.CFC bit field. Whenever NFCRx.CFC is updated in bit timing analysis mode, bit NFCRx.CFCOV is also set to indicate the CFC update event. If NFCRx.CFCIE is set, an interrupt request can be generated (see **Figure 15-9**).

Automatic Baud Rate Detection

For automatic baud rate detection, the time between the observation of subsequent dominant edges on the CAN bus must be measured. This measurement is automatically performed if bit field NFCRx.CFSEL = 000_B. With each dominant edge monitored on the CAN receive input line, the time (measured in f_{CAN} clock cycles) between this edge and the most recent dominant edge is stored in the NFCRx.CFC bit field.

Synchronization Analysis

The bit time synchronization is monitored if NFCRx.CFSEL = 010_B. The time between the first dominant edge and the sample point is measured and stored in the NFCRx.CFC bit field. The bit timing synchronization offset may be derived from this time as the first edge after the sample point triggers synchronization and there is only one synchronization between consecutive sample points.

Synchronization analysis can be used, for example, for fine tuning of the baud rate during reception of the first CAN frame with the measured baud rate.

Driver Delay Measurement

The delay between a transmitted edge and the corresponding received edge is measured when NFCRx.CFSEL = 011_B (dominant to dominant) and NFCRx.CFSEL = 100_B (recessive to recessive). These delays indicate the time needed to represent a new bit value on the physical implementation of the CAN bus.

15.3.6 Message Acceptance Filtering

The chapter describes the Message Acceptance Filtering capabilities of the MultiCAN module.

15.3.6.1 Receive Acceptance Filtering

When a CAN frame is received by a CAN node, a unique message object is determined in which the received frame is stored after successful frame reception. A message object is qualified for reception of a frame if the following six conditions are met.

- The message object is allocated to the message object list of the CAN node by which the frame is received.
- Bit MOSTATn.MSGVAL in the Message Status Register (see **Page 15-96**) is set.
- Bit MOSTATn.RXEN is set.
- Bit MOSTATn.DIR is equal to bit RTR of the received frame.
  If bit MOSTATn.DIR = 1 (transmit object), the message object accepts only Remote Frames. If bit MOSTATn.DIR = 0 (receive object), the message object accepts only Data Frames.
- If bit MOAMRn.MIDE = 1, the IDE bit of the received frame becomes evaluated in the following way: If MOARn.IDE = 1, the IDE bit of the received frame must be set (indicates extended identifier). If MOARn.IDE = 0, the IDE bit of the received frame must be cleared (indicates standard identifier).
  If bit MOAMRn.MIDE = 0, the IDE bit of the received frame is “don’t care”. In this case, message objects with standard and extended frames are accepted.
- The identifier of the received frame matches the identifier stored in the Arbitration Register of the message object as qualified by the acceptance mask in the MOAMRn register. This means that each bit of the received message object identifier is equal to the bit field MOARn.ID, except those bits for which the corresponding acceptance mask bits in bit field MOAMRn.AM are cleared. These identifier bits are “don’t care” for reception. **Figure 15-13** illustrates this receive message identifier check.

Among all messages that fulfill all six qualifying criteria the message object with the highest receive priority wins receive acceptance filtering and becomes selected to store the received frame. All other message objects lose receive acceptance filtering.

The following priority scheme is defined for the message objects:

A message object a (MOa) has higher receive priority than a message object b (MOb) if the following two conditions are fulfilled (see **Page 15-110**):

1. MOa has a higher priority class than MOb. This means, the 2-bit priority bit field MOARa.PRI must be equal or less than bit field MOARb.PRI.
2. If both message objects have the same priority class (MOARa.PRI = MOARb.PRI), MOb is a list successor of MOa. This means that MOb can be reached by means of successively stepping forward in the list, starting from a.

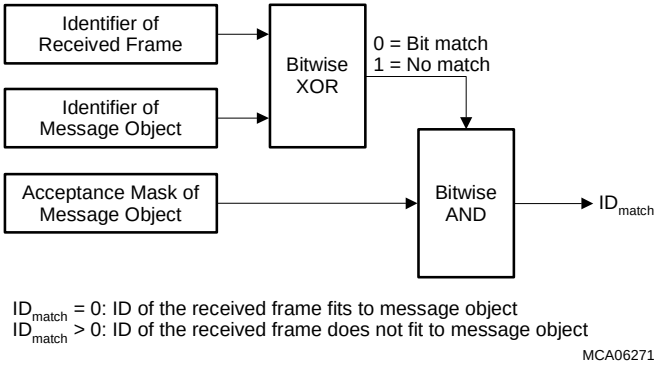


Figure 15-13 Received Message Identifier Acceptance Check

15.3.6.2 Transmit Acceptance Filtering

A message is requested for transmission by setting a transmit request in the message object that holds the message. If more than one message object have a valid transmit request for the same CAN node, one of these message objects is chosen for transmission, because only a single message object can be transmitted at one time on a CAN bus.

A message object is qualified for transmission on a CAN node if the following four conditions are met (see also **Figure 15-14**).

1. The message object is allocated to the message object list of the CAN node.
2. Bit MOSTATn.MSGVAL is set.
3. Bit MOSTATn.TXRQ is set.
4. Bit MOSTATn.TXEN0 and MOSTATn.TXEN1 are set.

A priority scheme determines which one of all qualifying message objects is transmitted first. It is assumed that message object a (MOa) and message object b (MOb) are two message objects qualified for transmission. MOb is a list successor of MOa. For both message objects, CAN messages CANa and CANb are defined (identifier, IDE, and RTR are taken from the message-specific bit fields and bits MOARn.ID, MOARn.IDE and MOCTRn.DIR).

If both message objects belong to the same priority class (identical PRI bit field in register MOARn), MOa has a higher transmit priority than MOb if one of the following conditions is fulfilled.

- PRI = 10_B and CAN message MOa has higher or equal priority than CAN message MOb with respect to CAN arbitration rules (see **Table 15-12** on **Page 15-111**).
- PRI = 01_B or PRI = 11_B (priority by list order).

The message object that is qualified for transmission and has highest transmit priority wins the transmit acceptance filtering, and will be transmitted first. All other message objects lose the current transmit acceptance filtering round. They get a new chance in subsequent acceptance filtering rounds.

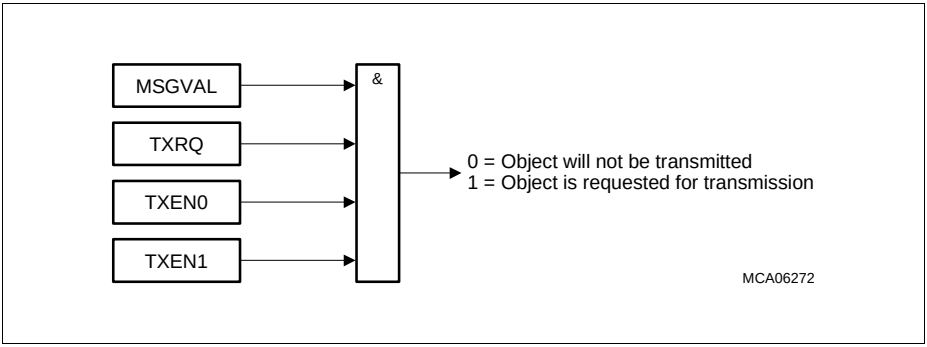


Figure 15-14 Effective Transmit Request of Message Object

15.3.7 Message Postprocessing

After a message object has successfully received or transmitted a frame, the CPU can be notified to perform a postprocessing on the message object. The postprocessing of the MultiCAN module consists of two elements:

1. Message interrupts to trigger postprocessing.
2. Message pending registers to collect pending message interrupts into a common structure for postprocessing.

15.3.7.1 Message Object Interrupts

When the storage of a received frame into a message object or the successful transmission of a frame is completed, a message interrupt can be issued. For each message object, a transmit and a receive interrupt can be generated and routed to one of the sixteen CAN interrupt output lines (see **Figure 15-15**). A receive interrupt occurs also after a frame storage event that has been induced by a FIFO or a gateway action. The status bits TXPND and RXPND in the Message Object n Status Register are always set after a successful transmission/reception, whether or not the respective message interrupt is enabled.

A third FIFO full interrupt condition of a message object is provided. If bit field MOFCRn.OVIE (Overflow Interrupt Enable) is set, the FIFO full interrupt will be activated depending on the actual message object type.

In case of a Receive FIFO Base Object (MOFCRn.MMC = 0001_B), the FIFO full interrupt is routed to the interrupt output line INT_Om as defined by the transmit interrupt node pointer MOIPRn.TXINP.

In case of a Transmit FIFO Base Object (MOFCRn.MMC = 0010_B), the FIFO full interrupt becomes routed to the interrupt output line INT_Om as defined by the receive interrupt node pointer MOIPRn.RXINP.

See also **“Service Request Generation” on Page 15-53** for further processing of the message object interrupts.

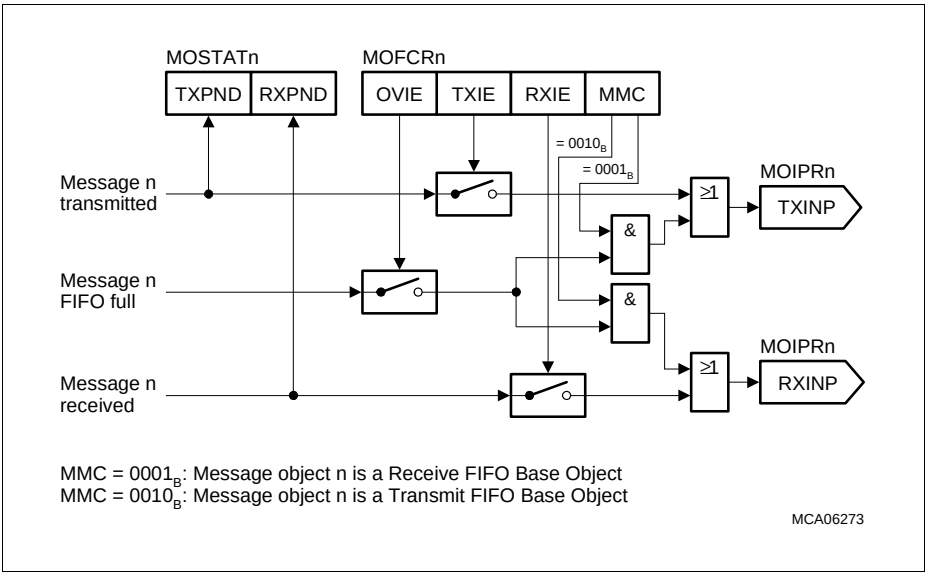


Figure 15-15 Message Interrupt Request Routing

15.3.7.2 Pending Messages

When a message interrupt request is generated, a message pending bit is set in one of the Message Pending Registers. There are 8 Message Pending Registers, MSPNDk (k = 0-7) with 32 pending bits available each. The general **Figure 15-16** shows the allocation of the message pending bits in case that the maximum possible number of eight Message Pending Registers are implemented and available on the chip.

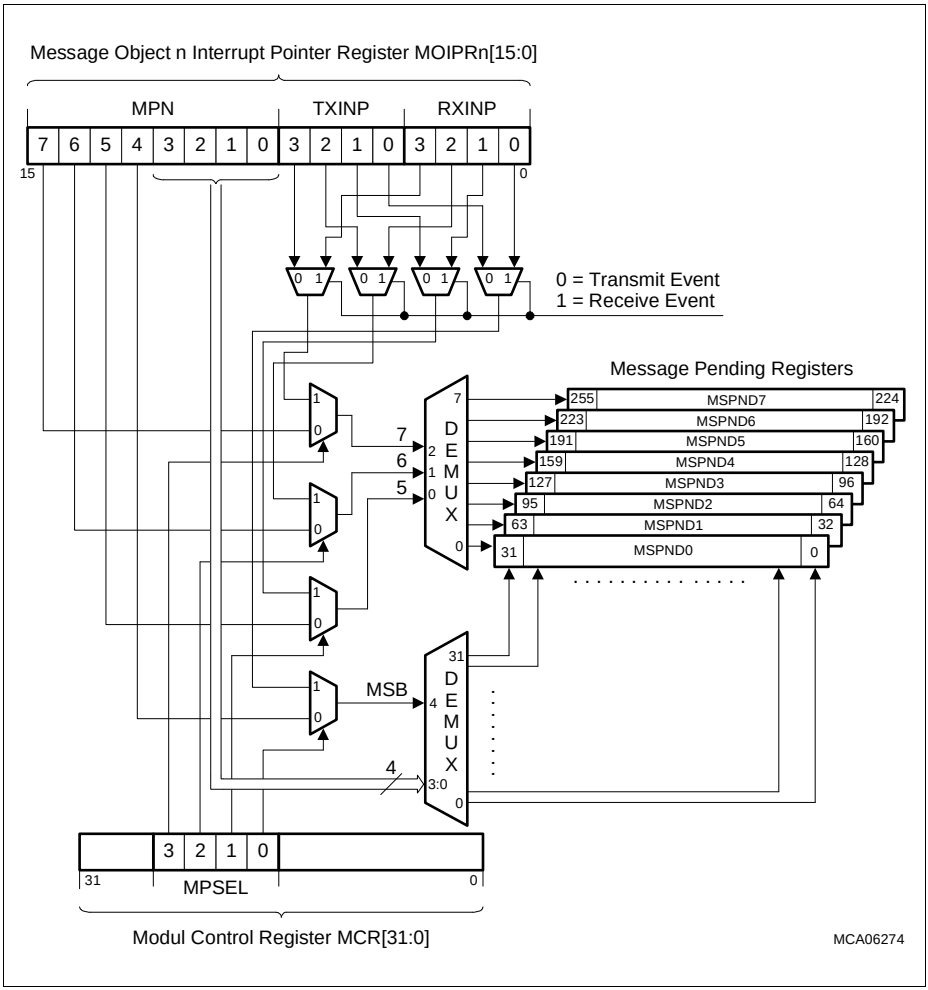


Figure 15-16 Message Pending Bit Allocation

The location of a pending bit is defined by two demultiplexers selecting the number k of the MSPNDk registers (3-bit demux), and the bit location within the corresponding MSPNDk register (5-bit demux).

Allocation Case 1

In this allocation case, bit field MCR.MPSEL = 0000_B (see **Page 15-67**). The location selection consists of 2 parts:

- The upper three bits of MOIPRn.MPN (MPN[7:5]) select the number k of a Message Pending Register MSPNDk in which the pending bit will be set.
- The lower five bits of MOIPRn.MPN (MPN[4:0]) select the bit position (0-31) in MSPNDk for the pending bit to be set.

Allocation Case 2

In this allocation case, bit field MCR.MPSEL is taken into account for pending bit allocation. Bit field MCR.MPSEL makes it possible to include the interrupt request node pointer for reception (MOIPRn.RXINP) or transmission (MOIPRn.TXINP) for pending bit allocation in such a way that different target locations for the pending bits are used in receive and transmit case. If MPSEL = 1111_B, the location selection operates in the following way:

- At a transmit event, the upper 3 bits of TXINP determine the number k of a Message Pending Register MSPNDk in which the pending bit will be set. At a receive event, the upper 3 bits of RXINP determine the number k.
- The bit position (0-31) in MSPNDk for the pending bit to be set is selected by the lowest bit of TXINP or RXINP (selects between low and high half-word of MSPNDk) and the four least significant bits of MPN.

General Hints

The Message Pending Registers MSPNDk can be written by software. Bits that are written with 1 are left unchanged, and bits which are written with 0 are cleared. This makes it possible to clear individual MSPNDk bits with a single register write access. Therefore, access conflicts are avoided when the MultiCAN module (hardware) sets another pending bit at the same time when software writes to the register.

Each Message Pending Register MSPNDk is associated with a Message Index Register MSIDk (see **Page 15-72**) which indicates the lowest bit position of all set (1) bits in Message Pending Register k. The MSIDk register is a read-only register that is updated immediately when a value in the corresponding Message Pending Register k is changed by software or hardware.

15.3.8 Message Object Data Handling

This chapter describes the handling capabilities for the Message Object Data of the MultiCAN module.

15.3.8.1 Frame Reception

After the reception of a message, it is stored in a message object according to the scheme shown in **Figure 15-17**. The MultiCAN module not only copies the received data into the message object, and it provides advanced features to enable consistent data exchange between MultiCAN and CPU.

MSGVAL

Bit MSGVAL (Message Valid) in the Message Object n Status Register MOSTATn is the main switch of the message object. During the frame reception, information is stored in the message object only when MSGVAL = 1. If bit MSGVAL is reset by the CPU, the MultiCAN module stops all ongoing write accesses to the message object. Now the message object can be re-configured by the CPU with subsequent write accesses to it without being disturbed by the MultiCAN.

RTSEL

When the CPU re-configures a message object during CAN operation (for example, clears MSGVAL, modifies the message object and sets MSGVAL again), the following scenario can occur:

1. The message object wins receive acceptance filtering.
2. The CPU clears MSGVAL to re-configure the message object.
3. The CPU sets MSGVAL again after re-configuration.
4. The end of the received frame is reached. As MSGVAL is set, the received data is stored in the message object, a message interrupt request is generated, gateway and FIFO actions are processed, etc.

After the re-configuration of the message object (after step 3 above) the storage of further received data may be undesirable. This can be achieved through bit MOCTRn.RTSEL (Receive/Transmit Selected) that makes it possible to disconnect a message object from an ongoing frame reception.

When a message object wins the receive acceptance filtering, its RTSEL bit is set by the MultiCAN module to indicate an upcoming frame delivery. The MultiCAN module checks RTSEL whether it is set on successful frame reception to verify that the object is still ready for receiving the frame. The received frame is then stored in the message object (along with all subsequent actions such as message interrupts, FIFO & gateway actions, flag updates) only if RTSEL = 1.

When a message object is invalidated during CAN operation (resetting bit MSGVAL), RTSEL should be cleared before setting MSGVAL again (latest with the same write

access that sets MSGVAL) to prevent the storage of a frame that belongs to the old context of the message object. Therefore, a message object re-configuration should consist of the following steps:

1. Clear MSGVAL bit
2. Re-configure the message object while MSGVAL = 0
3. Clear RTSEL bit and set MSGVAL again

RXEN

Bit MOSTATn.RXEN enables a message object for frame reception. A message object can receive CAN messages from the CAN bus only if RXEN = 1. The MultiCAN module evaluates RXEN only during receive acceptance filtering. After receive acceptance filtering, RXEN is ignored and has no further influence on the actual storage of a received message in a message object.

Bit RXEN enables the “soft phase out” of a message object: after clearing RXEN, a currently received CAN message for which the message object has won acceptance filtering is still stored in the message object but for subsequent messages the message object no longer wins receive acceptance filtering.

RXUPD, NEWDAT and MSGLST

An ongoing frame storage process is indicated by the RXUPD (Receive Updating) flag in the MOSTATn register. RXUPD is set with the start and cleared with the end of a message object update, which consists of frame storage as well as flag updates.

After storing the received frame (identifier, IDE bit, DLC; including the Data Field for Data Frames), the NEWDAT (New Data) bit of the message object is set. If NEWDAT was already set before it becomes set again, bit MSGLST (Message Lost) is set to indicate a data loss condition.

The RXUPD and NEWDAT flags can help to read consistent frame data from the message object during an ongoing CAN operation. The following steps are recommended to be executed:

1. Clear NEWDAT bit.
2. Read message content (identifier, data etc.) from the message object.
3. Check that both, NEWDAT and RXUPD, are cleared. If this is not the case, go back to step 1.
4. When step 3 was successful, the message object contents are consistent and has not been updated by the MultiCAN module while reading.

Bits RXUPD, NEWDAT and MSGLST have the same behavior for the reception of Data as well as Remote Frames.

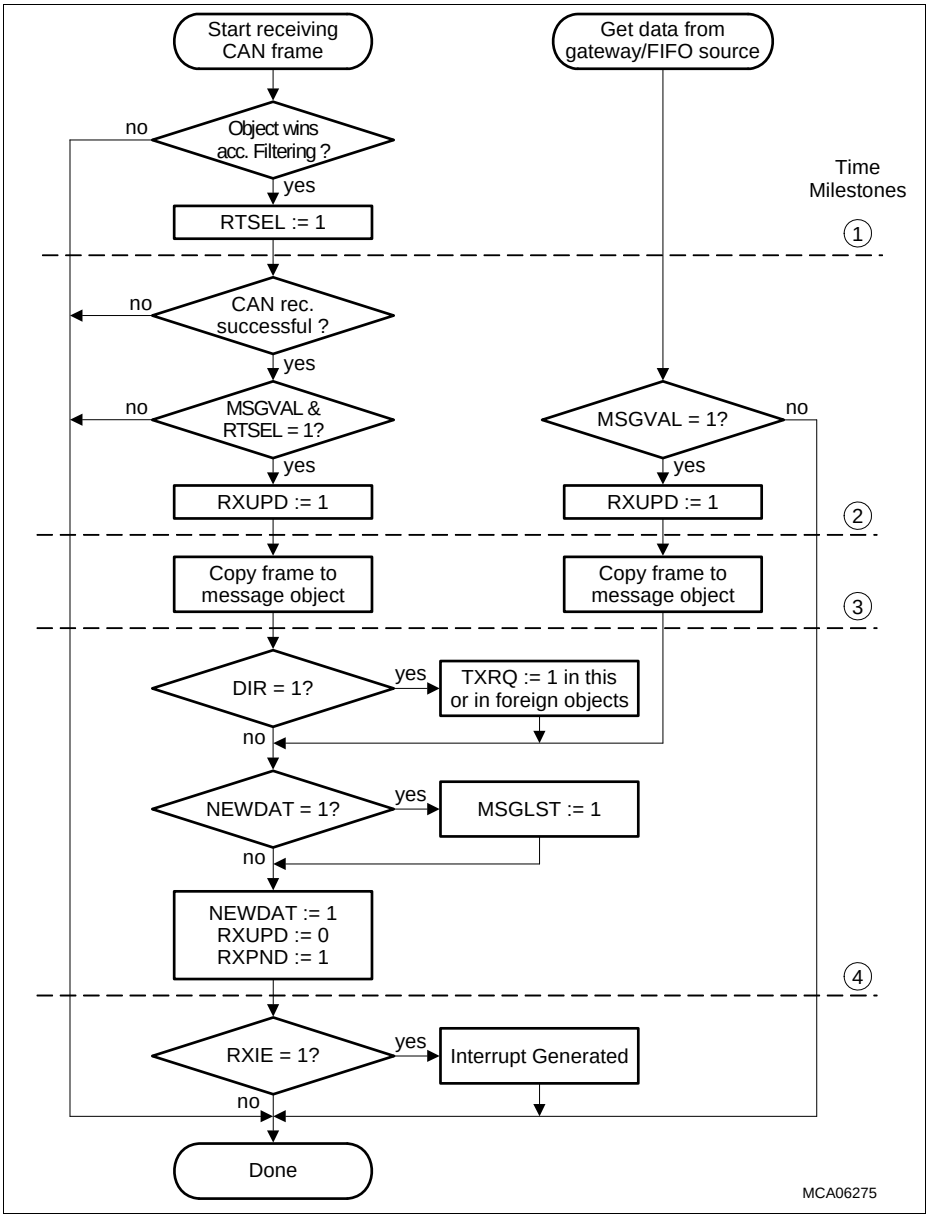


Figure 15-17 Reception of a Message Object

15.3.8.2 Frame Transmission

The process of a message object transmission is shown in **Figure 15-18**. Along with the copy of the message object content to be transmitted (identifier, IDE bit, RTR = DIR bit, DLC, including the Data Field for Data Frames) into the internal transmit buffer of the assigned CAN node, several status flags are also served and monitored to control consistent data handling.

The transmission process of a message object starting after the transmit acceptance filtering is identical for Remote and Data Frames.

MSGVAL, TXRQ, TXEN0, TXEN1

A message can only be transmitted if all four bits in registers MOSTATn, MSGVAL (Message Valid), TXRQ (Transmit Request), TXEN0 (Transmit Enable 0), TXEN1 (Transmit Enable 1) are set as shown in **Figure 15-14**. Although these bits are equivalent with respect to the transmission process, they have different semantics:

Table 15-2 Message Transmission Bit Definitions

Bit	Description
MSGVAL	**Message Valid** This is the main switch bit of the message object.
TXRQ	**Transmit Request** This is the standard transmit request bit. This bit must be set whenever a message object should be transmitted. TXRQ is cleared by hardware at the end of a successful transmission, except when there is new data (indicated by NEWDAT = 1) to be transmitted. When bit MOFCRn.STT (“Single Transmit Trial”) is set, TXRQ becomes already cleared when the contents of the message object are copied into the transmit frame buffer of the CAN node. A received remote request (after a Remote Frame reception) sets bit TXRQ to request the transmission of the requested data frame.
TXEN0	**Transmit Enable 0** This bit can be temporarily cleared by software to suppress the transmission of this message object when it writes new content to the Data Field. This avoids transmission of inconsistent frames that consist of a mixture of old and new data. Remote requests are still accepted when TXEN0 = 0, but transmission of the Data Frame is suspended until transmission is re-enabled by software (setting TXEN0).

Table 15-2 Message Transmission Bit Definitions (cont'd)

Bit	Description
TXEN1	**Transmit Enable 1** This bit is used in transmit FIFOs to select the message object that is transmit active within the FIFO structure. For message objects that are not transmit FIFO elements, TXEN1 can either be set permanently to 1 or can be used as a second independent transmission enable bit.

RTSEL

When a message object has been identified to be transmitted next after transmission acceptance filtering, bit MOCTRn.RTSEL (Receive/Transmit Selected) is set.

When the message object is copied into the internal transmit buffer, bit RTSEL is checked, and the message is transmitted only if RTSEL = 1. After the successful transmission of the message, bit RTSEL is checked again and the message postprocessing is only executed if RTSEL = 1.

For a complete re-configuration of a valid message object, the following steps should be executed:

1. Clear MSGVAL bit
2. Re-configure the message object while MSGVAL = 0
3. Clear RTSEL and set MSGVAL

Clearing of RTSEL ensures that the message object is disconnected from an ongoing/scheduled transmission and no message object processing (copying message to transmit buffer including clearing NEWDAT, clearing TXRQ, time stamp update, message interrupt, etc.) within the old context of the object can occur after the message object becomes valid again, but within a new context.

NEWDAT

When the contents of a message object have been transferred to the internal transmit buffer of the CAN node, bit MOSTATn.NEWDAT (New Data) is cleared by hardware to indicate that the transmit message object data is no longer new.

When the transmission of the frame is successful and NEWDAT is still cleared (if no new data has been copied into the message object meanwhile), TXRQ (Transmit Request) is cleared automatically by hardware.

If, however, the NEWDAT bit has been set again by the software (because a new frame should be transmitted), TXRQ is not cleared to enable the transmission of the new data.

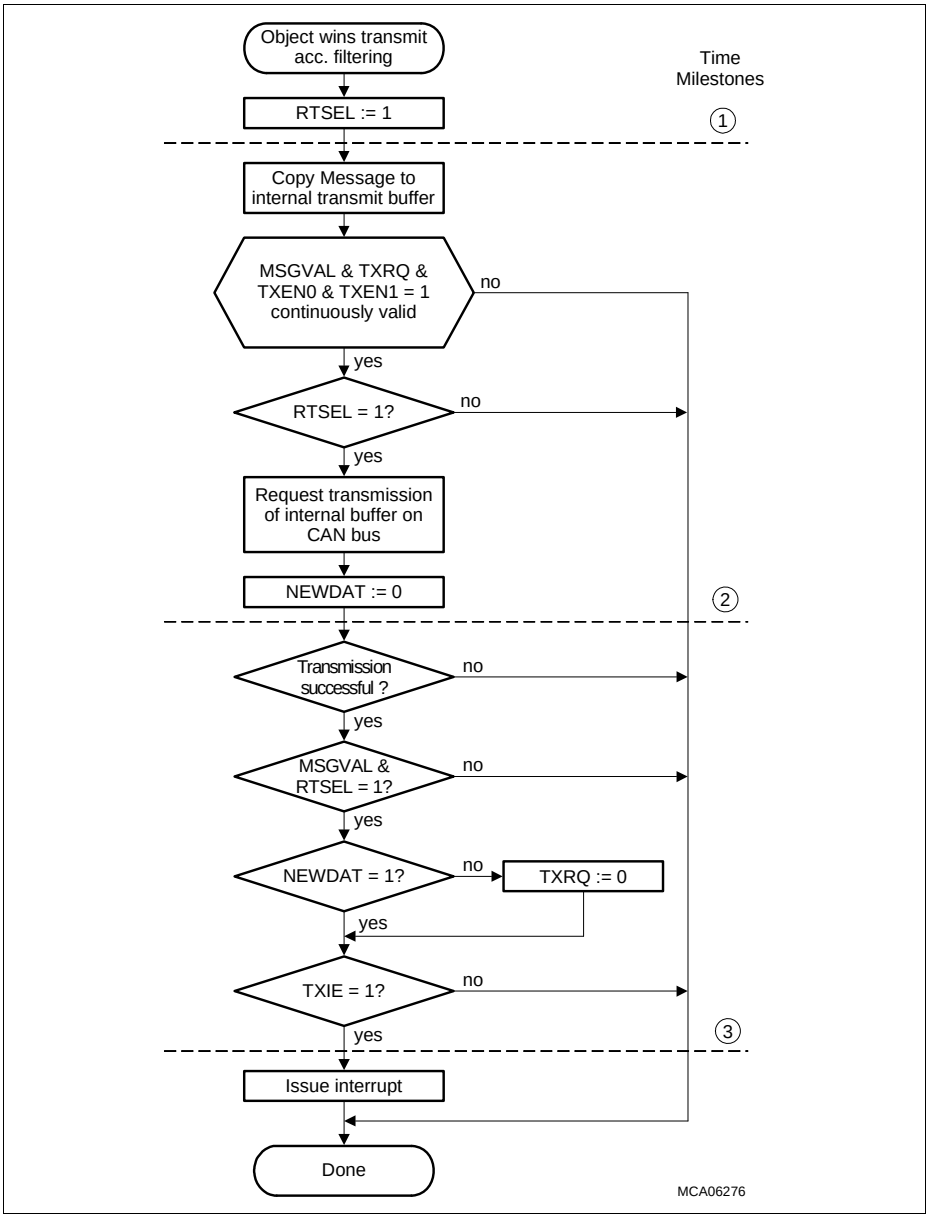


Figure 15-18 Transmission of a Message Object

15.3.9 Message Object Functionality

This chapter describes the functionality of the Message Objects in the MultiCAN module.

15.3.9.1 Standard Message Object

A message object is selected as standard message object when bit field MOFCRn.MMC = 0000_B (see **Page 15-103**). The standard message object can transmit and receive CAN frames according to the basic rules described in the previous sections. Additional services such as Single Data Transfer Mode or Single Transmit Trial (see following sections) are available and can be individually selected.

15.3.9.2 Single Data Transfer Mode

Single Data Transfer Mode is a useful feature in order to broadcast data over the CAN bus without unintended duplication of information. Single Data Transfer Mode is selected via bit MOFCRn.SDT.

Message Reception

When a received message stored in a message object is overwritten by a new received message, the contents of the first message are lost and replaced with the contents of the new received message (indicated by MSGLST = 1).

If SDT is set (Single Data Transfer Mode activated), bit MSGVAL of the message object is automatically cleared by hardware after the storage of a received Data Frame. This prevents the reception of further messages.

After the reception of a Remote Frame, bit MSGVAL is not automatically cleared.

Message Transmission

When a message object receives a series of multiple remote requests, it transmits several Data Frames in response to the remote requests. If the data within the message object has not been updated in the time between the transmissions, the same data can be sent more than once on the CAN bus.

In Single Data Transfer Mode (SDT = 1), this is avoided because MSGVAL is automatically cleared after the successful transmission of a Data Frame.

After the transmission of a Remote Frame, bit MSGVAL is not automatically cleared.

15.3.9.3 Single Transmit Trial

If the bit STT in the message object function register is set (STT = 1), the transmission request is cleared (TXRQ = 0) when the frame contents of the message object have been copied to the internal transmit buffer of the CAN node. Thus, the transmission of the message object is not tried again when it fails due to CAN bus errors.

15.3.9.4 Message Object FIFO Structure

In case of high CPU load it may be difficult to process a series of CAN frames in time. This may happen if multiple messages are received or must be transmitted in short time.

Therefore, a FIFO buffer structure is available to avoid loss of incoming messages and to minimize the setup time for outgoing messages. The FIFO structure can also be used to automate the reception or transmission of a series of CAN messages and to generate a single message interrupt when the whole CAN frame series is done.

There can be several FIFOs in parallel. The number of FIFOs and their size are limited only by the number of available message objects. A FIFO can be installed, resized and de-installed at any time, even during CAN operation.

The basic structure of a FIFO is shown in **Figure 15-19**. A FIFO consists of one base object and n slave objects. The slave objects are chained together in a list structure (similar as in message object lists). The base object may be allocated to any list. Although **Figure 15-19** shows the base object as a separate part beside the slave objects, it is also possible to integrate the base object at any place into the chain of slave objects. This means that the base object is slave object, too (not possible for gateways). The absolute object numbers of the message objects have no impact on the operation of the FIFO.

The base object does not need to be allocated to the same list as the slave objects. Only the slave object must be allocated to a common list (as they are chained together). Several pointers (BOT, CUR and TOP) that are located in the Message Object n FIFO/Gateway Pointer Register MOFGPRn link the base object to the slave objects, regardless whether the base object is allocated to the same or to another **list** than the slave objects.

The smallest FIFO would be a single message object which is both, FIFO base and FIFO slave (not very useful). The biggest possible FIFO structure would include all message objects of the MultiCAN module. Any FIFO sizes between these limits are possible.

In the FIFO base object, the FIFO boundaries are defined. Bit field MOFGPRn.BOT of the base object points to (includes the number of) the bottom slave object in the FIFO structure. The MOFGPRn.TOP bit field points to (includes the number of) the top slave object in the FIFO structure. The MOFGPRn.CUR bit field points to (includes the number of) the slave object that is actually selected by the MultiCAN module for message transfer. When a message transfer takes place with this object, CUR is set to the next message object in the list structure of the slave objects (CUR = PNEXT of current object). If CUR was equal to TOP (top of the FIFO reached), the next update of CUR will result in CUR = BOT (wrap-around from the top to the bottom of the FIFO). This scheme represents a circular FIFO structure where the bit fields BOT and TOP establish the link from the last to the first element.

Bit field MOFGPRn.SEL of the base object can be used for monitoring purposes. It makes it possible to define a slave object within the list at which a message interrupt is

generated whenever the CUR pointer reaches the value of the SEL pointer. Thus SEL makes it possible to detect the end of a predefined message transfer series or to issue a warning interrupt when the FIFO becomes full.

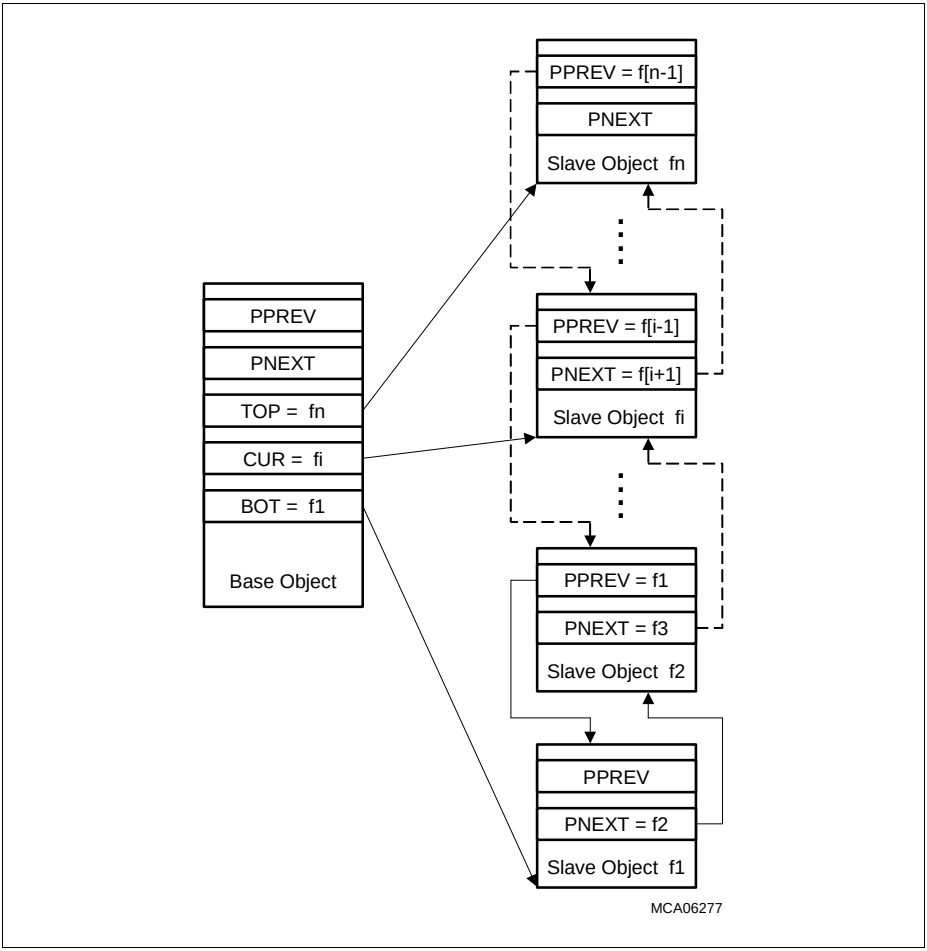


Figure 15-19 FIFO Structure with FIFO Base Object and n FIFO Slave Objects

15.3.9.5 Receive FIFO

The Receive FIFO structure is used to buffer incoming (received) Remote or Data Frames.

A Receive FIFO is selected by setting MOFCRn.MMC = 0001_B in the FIFO base object. This MMC code automatically designates a message object as FIFO base object. The message modes of the FIFO slave objects are not relevant for the operation of the Receive FIFO.

When the FIFO base object receives a frame from the CAN node it belongs to, the frame is not stored in the base object itself but in the message object that is selected by the base object's MOFGPRn.CUR pointer. This message object receives the CAN message as if it is the direct receiver of the message. However, MOFCRn.MMC = 0000_B is implicitly assumed for the FIFO slave object, and a standard message delivery is performed. The actual message mode (MMC setting) of the FIFO slave object is ignored. For the slave object, no acceptance filtering takes place that checks the received frame for a match with the identifier, IDE bit, and DIR bit.

With the reception of a CAN frame, the current pointer CUR of the base object is set to the number of the next message object in the FIFO structure. This message object will then be used to store the next incoming message.

If bit field MOFCRn.OVIE ("Overflow Interrupt Enable") of the FIFO base object is set and the current pointer MOFGPRn.CUR becomes equal to MOFGPRn.SEL, a FIFO overflow interrupt request is generated. This interrupt request is generated on interrupt node TXINP of the base object immediately after the storage of the received frame in the slave object. Transmit interrupts are still generated if TXIE is set.

A CAN message is stored in a FIFO slave only if MSGVAL = 1 in both FIFO base and slave object.

In order to avoid direct reception of a message by a slave message object, as if it was an independent message object and not a part of a FIFO, the bit RXEN of each slave object must be cleared. The setting of the bit RXEN is "don't care" only if the slave object is located in a list not assigned to a CAN node.

15.3.9.6 Transmit FIFO

The Transmit FIFO structure is used to buffer a series of Data or Remote Frames that must be transmitted. A transmit FIFO consists of one base message object and one or more slave message objects.

A Transmit FIFO is selected by setting MOFCRn.MMC = 0010_B in the FIFO base object. Unlike the Receive FIFO, slave objects assigned to the Transmit FIFO must explicitly set their bit fields MOFCRn.MMC = 0011_B. The CUR pointer in all slave objects must point back to the Transmit FIFO Base Object (to be initialized by software).

The MOSTATn.TXEN1 bits (Transmit Enable 1) of all message objects except the one which is selected by the CUR pointer of the base object must be cleared by software. TXEN1 of the message (slave) object selected by CUR must be set. CUR (of the base object) may be initialized to any FIFO slave object.

When tagging the message objects of the FIFO as valid to start the operation of the FIFO, then the base object must be tagged valid (MSGVAL = 1) first.

Before a Transmit FIFO becomes de-installed during operation, its slave objects must be tagged invalid (MSGVAL = 0).

The Transmit FIFO uses the TXEN1 bit in the Message Object Control Register of all FIFO elements to select the actual message for transmission. Transmit acceptance filtering evaluates TXEN1 for each message object and a message object can win transmit acceptance filtering only if its TXEN1 bit is set. When a FIFO object has transmitted a message, the hardware clears its TXEN1 bit in addition to standard transmit postprocessing (clear TXRQ, transmit interrupt etc.), and moves the CUR pointer in the next FIFO base object to be transmitted. TXEN1 is set automatically (by hardware) in the next message object. Thus, TXEN1 moves along the Transmit FIFO structure as a token that selects the active element.

If bit field MOFCRn.OVIE (“Overflow Interrupt Enable”) of the FIFO base object is set and the current pointer CUR becomes equal to MOFGPRn.SEL, a FIFO overflow interrupt request is generated. The interrupt request is generated on interrupt node RXINP of the base object after postprocessing of the received frame. Receive interrupts are still generated for the Transmit FIFO base object if bit RXIE is set.

15.3.9.7 Gateway Mode

The Gateway Mode makes it possible to establish an automatic information transfer between two independent CAN buses without CPU interaction.

The Gateway Mode operates on message object level. In Gateway mode, information is transferred between two message objects, resulting in an information transfer between the two CAN nodes to which the message objects are allocated. A gateway may be established with any pair of CAN nodes, and there can be as many gateways as there are message objects available to build the gateway structure.

Gateway Mode is selected by setting MOFCRs.MMC = 0100_B for the gateway source object s. The gateway destination object d is selected by the MOFGPRd.CUR pointer of the source object. The gateway destination object only needs to be valid (its MSGVAL = 1). All other settings are not relevant for the information transfer from the source object to the destination object.

Gateway source object behaves as a standard message object with the difference that some additional actions are performed by the MultiCAN module when a CAN frame has been received and stored in the source object (see **Figure 15-20**):

1. If bit MOFCRs.DLCC is set, the data length code MOFCRs.DLC is copied from the gateway source object to the gateway destination object.
2. If bit MOFCRs.IDC is set, the identifier MOARs.ID and the identifier extension MOARs.IDE are copied from the gateway source object to the gateway destination object.
3. If bit MOFCRs.DATC is set, the data bytes stored in the two data registers MODATALs and MODATAHs are copied from the gateway source object to the gateway destination object. All 8 data bytes are copied, even if MOFCRs.DLC indicates less than 8 data bytes.
4. If bit MOFCRs.GDFS is set, the transmit request flag MOSTATd.TXRQ is set in the gateway destination object.
5. The receive pending bit MOSTATd.RXPND and the new data bit MOSTATd.NEWDAT are set in the gateway destination object.
6. A message interrupt request is generated for the gateway destination object if its MOSTATd.RXIE is set.
7. The current object pointer MOFGPRs.CUR of the gateway source object is moved to the next destination object according to the FIFO rules as described on **Page 15-46**. A gateway with a single (static) destination object is obtained by setting MOFGPRs.TOP = MOFGPRs.BOT = MOFGPRs.CUR = destination object.

The link from the gateway source object to the gateway destination object works in the same way as the link from a FIFO base to a FIFO slave. This means that a gateway with an integrated destination FIFO may be created; in **Figure 15-19**, the object on the left is the gateway source object and the message object on the right side is the gateway destination objects.

The gateway operates equivalent for the reception of data frames (source object is receive object, i.e. DIR = 0) as well as for the reception of Remote Frames (source object is transmit object).

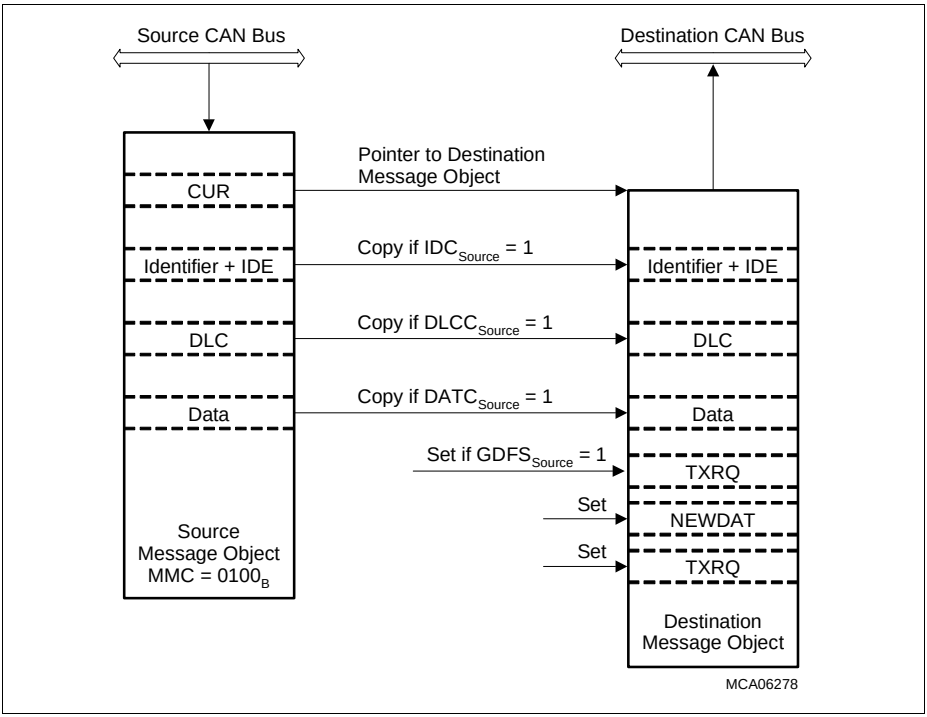


Figure 15-20 Gateway Transfer from Source to Destination

15.3.9.8 Foreign Remote Requests

When a Remote Frame has been received on a CAN node and is stored in a message object, a transmit request is set to trigger the answer (transmission of a Data Frame) to the request or to automatically issue a secondary request. If the Foreign Remote Request Enable bit MOFCRn.FRREN is cleared in the message object in which the remote request is stored, MOSTATn.TXRQ is set in the same message object.

If bit FRREN is set (FRREN = 1: foreign remote request enabled), TXRQ is set in the message object that is referenced by pointer MOFGPRn.CUR. The value of CUR is, however, not changed by this feature.

Although the foreign remote request feature works independently of the selected message mode, it is especially useful for gateways to issue a remote request on the source bus of a gateway after the reception of a remote request on the gateway destination bus. According to the setting of FRREN in the gateway destination object, there are two capabilities to handle remote requests that appear on the destination side (assuming that the source object is a receive object and the destination is a transmit object, i.e. DIR_{source} = 0 and $DIR_{destination}$ = 1):

FRREN = 0 in the Gateway Destination Object

1. A Remote Frame is received by gateway destination object.
2. TXRQ is set automatically in the gateway destination object.
3. A Data Frame with the current data stored in the destination object is transmitted on the destination bus.

FRREN = 1 in the Gateway Destination Object

1. A Remote Frame is received by gateway destination object.
2. TXRQ is set automatically in the gateway source object (must be referenced by CUR pointer of the destination object).
3. A remote request is transmitted by the source object (which is a receive object) on the source CAN bus.
4. The receiver of the remote request responds with a Data Frame on the source bus.
5. The Data Frame is stored in the source object.
6. The Data Frame is copied to the destination object (gateway action).
7. TXRQ is set in the destination object (assuming $GDFS_{source}$ = 1).
8. The new data stored in the destination object is transmitted on the destination bus, in response to the initial remote request on the destination bus.

15.4 Service Request Generation

The interrupt control logic in the MultiCAN module uses an interrupt compressing scheme that allows high flexibility in interrupt processing. There are 140 hardware interrupt sources and one software interrupt source available:

- CAN node interrupts:
  - Four different interrupt sources for each of the three CAN nodes = 12 interrupt sources
- Message object interrupts:
  - Two interrupt source for each message object = 128 interrupt sources
- One software initiated interrupt (register MITR)

Each of the 140 hardware initiated interrupt sources is controlled by a 4-bit interrupt pointer that directs the interrupt source to one of the 8 interrupt outputs INT_Om (m = 0-7). This makes it possible to connect more than one interrupt source (between one and all) to one interrupt output line. The interrupt wiring matrix shown in **Figure 15-21** is built up according to the following rules:

- Each output of the 4-bit interrupt pointer demultiplexer is connected to exactly one OR-gate input of the INT_Om line. The number “m” of the corresponding selected INT_Om interrupt output line is defined by the interrupt pointer value.
- Each INT_Om output line has an input OR gate which is connected to all interrupt pointer demultiplexer outputs which are selected by an identical 4-bit pointer value.

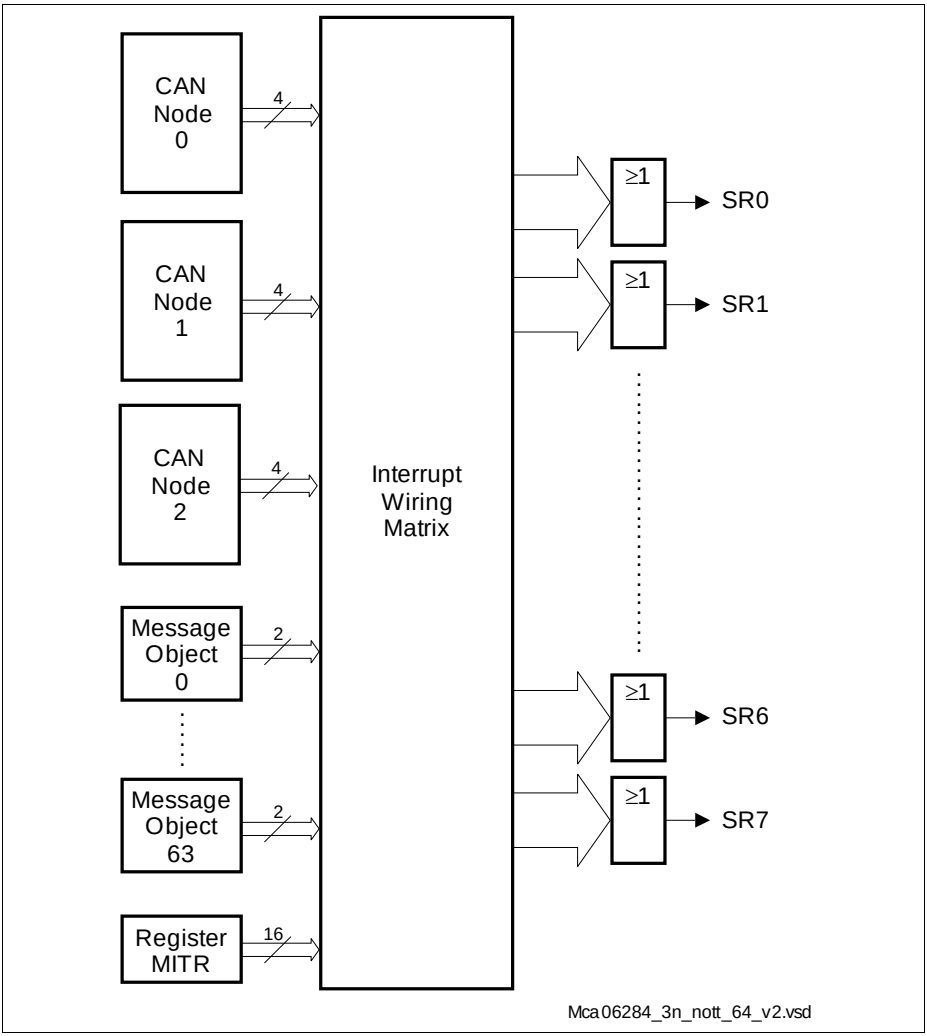


Figure 15-21 Interrupt Compressor

15.5 Debug behavior

The Suspend Mode can be triggered by the OCDS in order to freeze the state of the module and to permit access to the registers (at least for read actions). The MultiCAN module provides the following Suspend Modes:

- The current action is finished (Soft Suspend Mode):
  The module clock f_{CLC} keeps running. Module functions are stopped automatically after internal actions have been finished (for example, after a CAN frame has been sent out). The end of the internal actions is indicated to the fractional divider by a suspend mode acknowledged signal. Due to this behavior, the communication network is not blocked. Furthermore, all registers are accessible for read and write actions. As a result, the debugger can stop the module actions and modify registers. These modifications are taken into account after the Suspend Mode is left.

The Soft Suspend Mode can be individually enabled for each CAN node.

A CAN node that is not active can always be suspended. Refer to **CAN_CLC** and **CAN_FDR** registers for the corresponding OCDS control.

15.6 Power, Reset and Clock

The MultiCAN module is located in the core power domain. The module and all registers are reset to their default state by a system reset as shown on **Table 15-4** and **Table 15-13**.

15.6.1 Clock Control

The CAN module timer clock f_{CAN} of the functional blocks of the MultiCAN module is derived from the module control clock f_{CLC}. The Fractional Divider is used to generate f_{CAN} used for bit timing calculation, The frequency of f_{CAN} is identical for all CAN nodes. The register file operate with the module control clock f_{CLC}. See also **"Module Clock Generation" on Page 15-58**.

The output clock f_{CAN} of the Fractional Divider is based on the system clock f_{CLC}, but only every n-th clock pulse is taken. The suspend signal (coming as acknowledge from the MultiCAN module in response to a OCDS suspend request) freezes or resets the Fractional Divider.

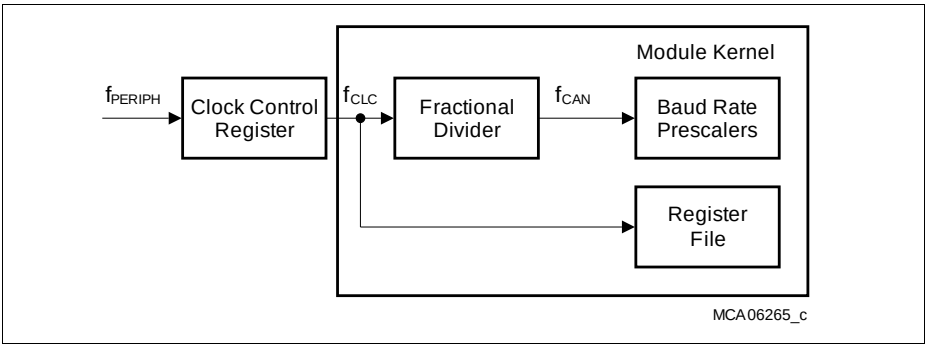


Figure 15-22 MultiCAN Clock Generation

Table 15-3 indicates the minimum operating frequencies in MHz for f_{CLC} that are required for a baud rate of 1 Mbit/s for the active CAN nodes. If a lower baud rate is desired, the values can be scaled linearly (e.g. for a maximum of 500 kbit/s, 50% of the indicated value are required).

The values imply that the CPU (or DMA) executes maximum accesses to the MultiCAN module. The values may contain rounding effects.

Table 15-3 Minimum Operating Frequencies [MHz]

Number of allocated message objects MO[1)]	1 CAN node active	2 CAN nodes active	3 CAN nodes active
16 MO	12	19	26
32 MO	15	23	30
64 MO	21	28	37

1) Only those message objects have to be taken into account that are allocated to a CAN node. The unallocated message objects have no influence on the minimum operating frequency.

The baud rate generation of the MultiCAN being based on f_{PERIPH}, this frequency has to be chosen carefully to allow correct CAN bit timing. The required value of f_{PERIPH} is given by an integer multiple (n) of the CAN baud rate multiplied by the number of time quanta per CAN bit time. For example, to reach 1 Mbit/s with 20 tq per bit time, possible values of f_{PERIPH} are given by formula [n × 20] MHz, with n being an integer value, starting at 1. In order to minimize jitter, it is not recommended to use the fractional divider mode for high baud rates.

15.6.2 Module Clock Generation

As shown in **Figure 15-23**, the clock signals for the MultiCAN module are generated and controlled by a clock control unit. This clock generation unit is responsible for the enable/disable control, the clock frequency adjustment, and the debug clock control. This unit includes two registers:

- CAN_CLC: generation of the module control clock f_{CLC}
- CAN_FDR: frequency control of the module timer clock f_{CAN}

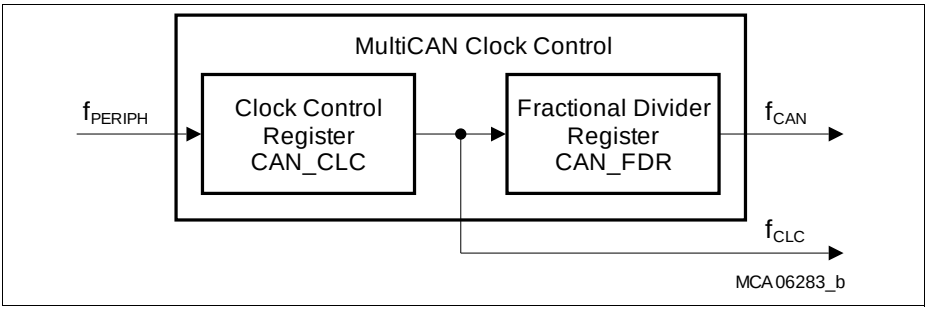


Figure 15-23 MultiCAN Module Clock Generation

The module control clock f_{CLC} is used inside the MultiCAN module for control purposes such as clocking of control logic and register operations. The frequency of f_{CLC} is identical to the system clock frequency f_{PERIPH}. The clock control register CAN_CLC makes it possible to enable/disable f_{CLC} under certain conditions.

The module timer clock f_{CAN} is used inside the MultiCAN module as input clock for all timing relevant operations (e.g. bit timing). The settings in the CAN_FDR register determine the frequency of the module timer clock f_{CAN} according the following two formulas:

$$f_{CAN} = f_{PB} \times \frac{1}{n} \quad \text{with n = 1024 - CAN\_FDR.STEP} \qquad (15.1)$$

$$f_{CAN} = f_{PB} \times \frac{n}{1024} \quad \text{with n = 0-1023} \qquad (15.2)$$

Equation (15.1) applies to normal divider mode (CAN_FDR.DM = 01_B) of the fractional divider. **Equation (15.2)** applies to fractional divider mode (CAN_FDR.DM = 10_B).

Note: The CAN module is disabled after reset. In general, after reset, the module control clock f_{CLC} must be switched on (writing to register CAN_CLC) before the frequency of the module timer clock f_{CAN} is defined (writing to register CAN_FDR).

15.7 Register Description

This section describes the kernel registers of the MultiCAN module. All MultiCAN kernel register names described in this section are also referenced in other parts of the XMC4[12]00 Reference Manual by the module name prefix “CAN_”.

MultiCAN Kernel Register Overview

The MultiCAN Kernel include three blocks of registers:

- Global Module Registers
- Node Registers, for each CAN node x
- Message Object Registers, for each message object n

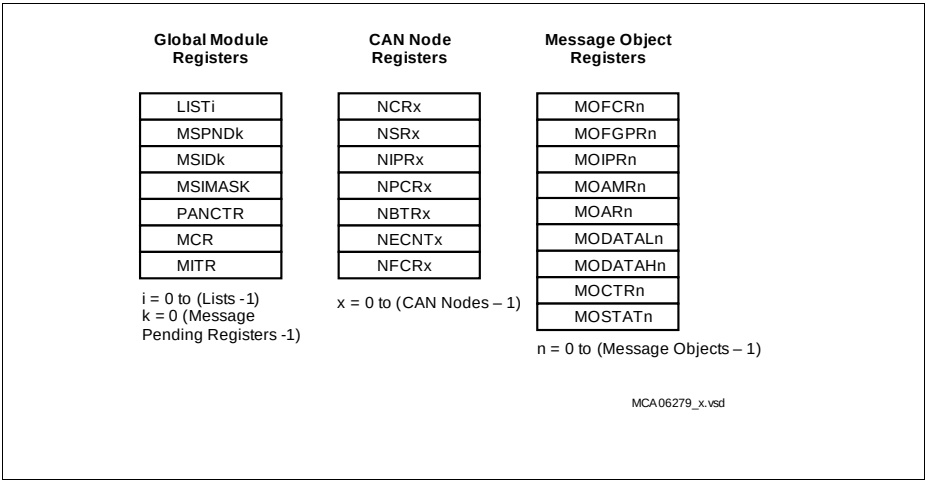


Figure 15-24 MultiCAN Kernel Registers

The registers of the MultiCAN module kernel are listed below.

Table 15-4 Registers Address Space - MultiCAN Kernel Registers

Module	Base Address	End Address	Note
CAN	$4801\ 4000_H$	$4801\ 7FFF_H$	-

Table 15-5 Registers Overview - MultiCAN Kernel Registers

Register Short Name	Register Long Name	Offset Address [1]	Access Mode [2]		Description see
			Read	Write	
Global Module Registers					
LISTi	List Register i	$0100_H + i \times 4_H$	U, PV	U, PV	Page 15-69
MSPNDk	Message Pending Register k	$0140_H + k \times 4_H$	U, PV	U, PV	Page 15-71
MSIDk	Message Index Register k	$0180_H + k \times 4_H$	U, PV	U, PV	Page 15-72
MSIMASK	Message Index Mask Register	$01C0_H$	U, PV	U, PV	Page 15-73
PANCTR	Panel Control Register	$01C4_H$	U, PV	U, PV	Page 15-63
MCR	Module Control Register	$01C8_H$	U, PV	U, PV	Page 15-67
MITR	Module Interrupt Trigger Reg.	$01CC_H$	U, PV	U, PV	Page 15-68
CAN Node Registers					
NCRx	Node x Control Register	$0200_H + x \times 100_H$	U, PV	U, PV	Page 15-74
NSRx	Node x Status Register	$0204_H + x \times 100_H$	U, PV	U, PV	Page 15-78
NIPRx	Node x Interrupt Pointer Reg.	$0208_H + x \times 100_H$	U, PV	U, PV	Page 15-82
NPCRx	Node x Port Control Register	$020C_H + x \times 100_H$	U, PV	U, PV	Page 15-84
NBTRx	Node x Bit Timing Register	$0210_H + x \times 100_H$	U, PV	U, PV	Page 15-85
NECNTx	Node x Error Counter Register	$0214_H + x \times 100_H$	U, PV	U, PV	Page 15-87
NFCRx	Node x Frame Counter Register	$0218_H + x \times 100_H$	U, PV	U, PV	Page 15-89
Message Object Registers					
MOFCRn	Message Object n Function Control Register	$1000_H + n \times 20_H$	U, PV	U, PV	Page 15-103

Table 15-5 Registers Overview - MultiCAN Kernel Registers (cont'd)

Register Short Name	Register Long Name	Offset Address [1]	Access Mode [2]		Description see
			Read	Write	
MOFGPRn	Message Object n FIFO/Gateway Pointer Register	$1004_H + n \times 20_H$	U, PV	U, PV	Page 15-107
MOIPRn	Message Object n Interrupt Pointer Register	$1008_H + n \times 20_H$	U, PV	U, PV	Page 15-101
MOAMRn	Message Object n Acceptance Mask Register	$100C_H + n \times 20_H$	U, PV	U, PV	Page 15-108
MODATALn	Message Object n Data Register Low	$1010_H + n \times 20_H$	U, PV	U, PV	Page 15-112
MODATAHn	Message Object n Data Register High	$1014_H + n \times 20_H$	U, PV	U, PV	Page 15-113
MOARn	Message Object n Arbitration Register	$1018_H + n \times 20_H$	U, PV	U, PV	Page 15-109
MOCTRn MOSTATn	Message Object n Control Reg. Message Object n Status Reg.	$101C_H + n \times 20_H$	U, PV	U, PV	Page 15-93 Page 15-96

1) The absolute register address is calculated as follows:
Module Base Address (**Table 15-4**) + Offset Address (shown in this column)
Further, the following ranges for parameters i, k, x, and n are valid: i = 0-7, k = 0-7, x = 0-1, n = 0-63.

2) Accesses to empty addresses: nBE

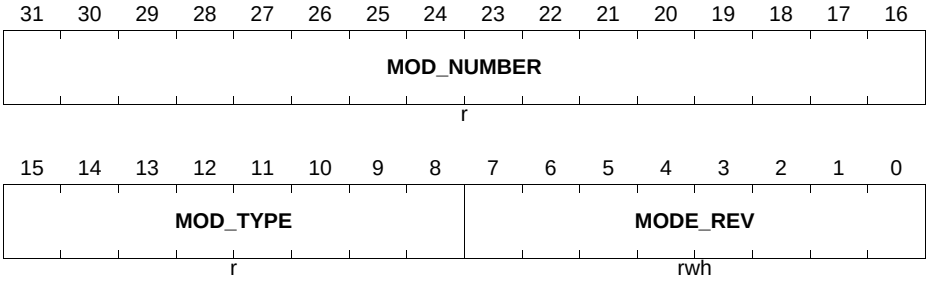
15.7.1 Global Module Registers

All list operations such as allocation, de-allocation and relocation of message objects within the list structure are performed via the Command Panel. It is not possible to modify the list structure directly by software by writing to the message objects and the LIST registers.

ID

The ID (Module Identification Register) defines the MultiCAN module identification number, module type and module revision number.

ID
Module Identification Register (008$_H$) Reset Value: 002B C0XX$_H$

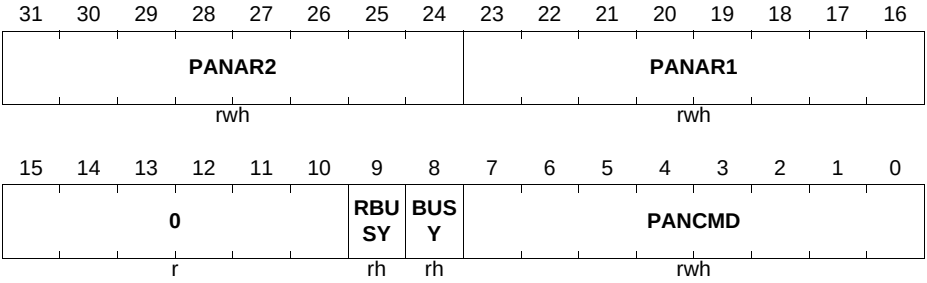


Field	Bits	Type	Description
MOD_REV	[7:0]	r	**Module Revision Number** MOD_REV defines the revision number. The value of a module revision starts with 01H (first revision).
MOD_TYPE	[15:8]	r	**Module Type** C0$_H$ Define the module as a 32-bit module.
MOD_NUMBER	[31:16]	r	**Module Number Value** This bit field defines the MultiCAN module identification number (=002BH)

PANCTR

The Panel Control Register PANCTR is used to start a new command by writing the command arguments and the command code into its bit fields.

PANCTR
Panel Control Register **($1C4_H$)** **Reset Value: $0000\ 0301_H$**



Field	Bits	Type	Description
PANCMD	[7:0]	rwh	**Panel Command** This bit field is used to start a new command by writing a panel command code into it. At the end of a panel command, the NOP (no operation) command code is automatically written into PANCMD. The coding of PANCMD is defined in **Table 15-6**.
BUSY	8	rh	**Panel Busy Flag** 0_B Panel has finished command and is ready to accept a new command. 1_B Panel operation is in progress.
RBUSY	9	rh	**Result Busy Flag** 0_B No update of PANAR1 and PANAR2 is scheduled by the list controller. 1_B A list command is running (BUSY = 1) that will write results to PANAR1 and PANAR2, but the results are not yet available.
PANAR1	[23:16]	rwh	**Panel Argument 1** See **Table 15-6**.
PANAR2	[31:24]	rwh	**Panel Argument 2** See **Table 15-6**.

Field	Bits	Type	Description
0	[15:10]	r	**Reserved** Read as 0; should be written with 0.

Panel Commands

A panel operation consists of a command code (PANCMD) and up to two panel arguments (PANAR1, PANAR2). Commands that have a return value deliver it to the PANAR1 bit field. Commands that return an error flag deliver it to bit 31 of the Panel Control Register, this means bit 7 of PANAR2.

Table 15-6 Panel Commands

PANCMD	PANAR2	PANAR1	Command Description
00_H	–	–	**No Operation** Writing 00_H to PANCMD has no effect. No new command is started.
01_H	**Result:** Bit 7: ERR Bit 6-0: undefined	–	**Initialize Lists** Run the initialization sequence to reset the CTRL and LIST fields of all message objects. List registers LIST[7:0] are set to their reset values. This results in the de-allocation of all message objects. The initialization command requires that bits NCRx.INIT and NCRx.CCE are set for all CAN nodes. Bit 7 of PANAR2 (ERR) reports the success of the operation: 0_B Initialization was successful 1_B Not all NCRx.INIT and NCRx.CCE bits are set. Therefore, no initialization is performed. The initialize lists command is automatically performed with each reset of the MultiCAN module, but with the exception that all message object registers are reset, too.

Table 15-6 Panel Commands (cont'd)

PANCMD	PANAR2	PANAR1	Command Description
02_H	**Argument:** List Index	**Argument:** Message Object Number	**Static Allocate** Allocate message object to a list. The message object is removed from the list that it currently belongs to, and appended to the end of the list, given by PANAR2. This command is also used to deallocate a message object. In this case, the target list is the list of unallocated elements (PANAR2 = 0).
03_H	**Argument:** List Index **Result:** Bit 7: ERR Bit 6-0: undefined	**Result:** Message Object Number	**Dynamic Allocate** Allocate the first message object of the list of unallocated objects to the selected list. The message object is appended to the end of the list. The message number of the message object is returned in PANAR1. An ERR bit (bit 7 of PANAR2) reports the success of the operation: 0_B Success. 1_B The operation has not been performed because the list of unallocated elements was empty.
04_H	**Argument:** Destination Object Number	**Argument:** Source Object Number	**Static Insert Before** Remove a message object (source object) from the list that it currently belongs to, and insert it before a given destination object into the list structure of the destination object. The source object thus becomes the predecessor of the destination object.

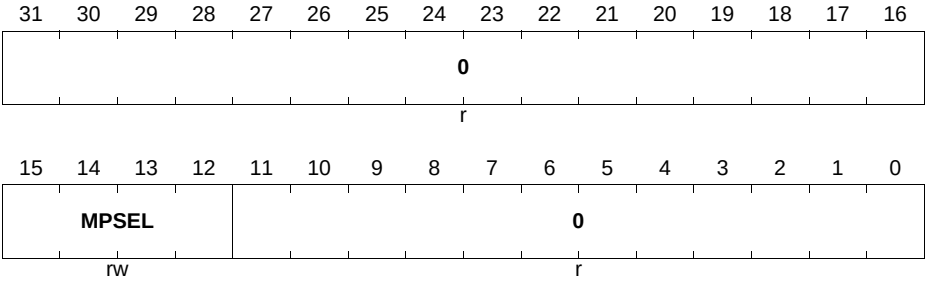
Table 15-6 Panel Commands (cont'd)

PANCMD	PANAR2	PANAR1	Command Description
05_H	**Argument:** Destination Object Number **Result:** Bit 7: ERR Bit 6-0: undefined	**Result:** Object Number of inserted object	**Dynamic Insert Before** Insert a new message object before a given destination object. The new object is taken from the list of unallocated elements (the first element is chosen). The number of the new object is delivered as a result to PANAR1. An ERR bit (bit 7 of PANAR2) reports the success of the operation: 0_B Success. 1_B The operation has not been performed because the list of unallocated elements was empty.
06_H	**Argument:** Destination Object Number	**Argument:** Source Object Number	**Static Insert Behind** Remove a message object (source object) from the list that it currently belongs to, and insert it behind a given destination object into the list structure of the destination object. The source object thus becomes the successor of the destination object.
07_H	**Argument:** Destination Object Number **Result:** Bit 7: ERR Bit 6-0: undefined	**Result:** Object Number of inserted object	**Dynamic Insert Behind** Insert a new message object behind a given destination object. The new object is taken from the list of unallocated elements (the first element is chosen). The number of the new object is delivered as result to PANAR1. An ERR bit (bit 7 of PANAR2) reports the success of the operation: 0_B Success. 1_B The operation has not been performed because the list of unallocated elements was empty.
08_H - FF_H	–	–	**Reserved**

MCR

The Module Control Register MCR contains basic settings that determine the operation of the MultiCAN module.

MCR
Module Control Register **($1C8_H$)** **Reset Value: $0000\ 0000_H$**

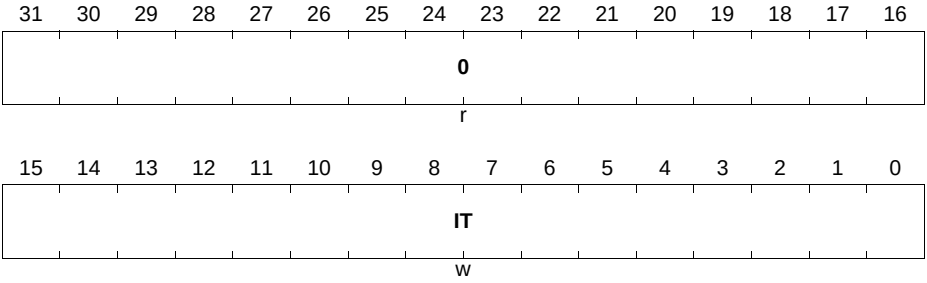


Field	Bits	Type	Description
MPSEL	[15:12]	rw	**Message Pending Selector** Bit field MPSEL makes it possible to select the bit position of the message pending bit after a message reception/transmission by a mixture of the MOIPRn register bit fields RXINP, TXINP, and MPN. Selection details are given in **Figure 15-16** on **Page 15-37**.
0	[31:16], [11:0]	r	**Reserved** Read as 0; should be written with 0.

MITR

The Interrupt Trigger Register ITR is used to trigger interrupt requests on each interrupt output line by software.

MITR
Module Interrupt Trigger Register (1CC$_H$) **Reset Value: 0000 0000$_H$**



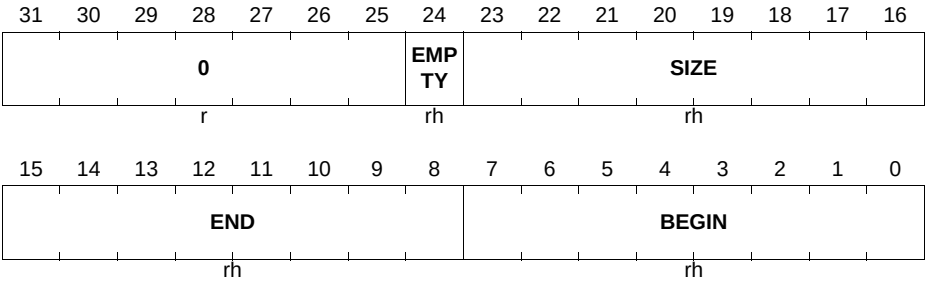
Field	Bits	Type	Description
IT	[7:0]	w	**Interrupt Trigger** Writing a 1 to IT[n] (n = 0-7) generates an interrupt request on interrupt output line INT_O[n]. Writing a 0 to IT[n] has no effect. Bit field IT is always read as 0. Multiple interrupt requests can be generated with a single write operation to MITR by writing a 1 to several bit positions of IT.
0	[31:8]	r	**Reserved** Read as 0; should be written with 0.

LIST

Each CAN node has a list that determines the allocated message objects. Additionally, a list of all unallocated objects is available. Furthermore, general purpose lists are available which are not associated to a CAN node. The List Registers are assigned in the following way:

- LIST0 provides the list of all unallocated objects
- LIST1 provides the list for CAN node 0
- LIST2 provides the list for CAN node 1
- LIST3 provides the list for CAN node 2
- LIST[7:4] are not associated to a CAN node (free lists)

LIST0
List Register 0 **(100_H)** **Reset Value: 003F 3F00$_H$**
LISTx (x = 1-7)
List Register x **(100_H+x*4_H)** **Reset Value: 0100 0000$_H$**



Field	Bits	Type	Description
BEGIN	[7:0]	rh	**List Begin** BEGIN indicates the number of the first message object in list i.
END	[15:8]	rh	**List End** END indicates the number of the last message object in list i.
SIZE	[23:16]	rh	**List Size** SIZE indicates the number of elements in the list i. SIZE = number of list elements - 1 SIZE = 0 indicates that list x is empty.

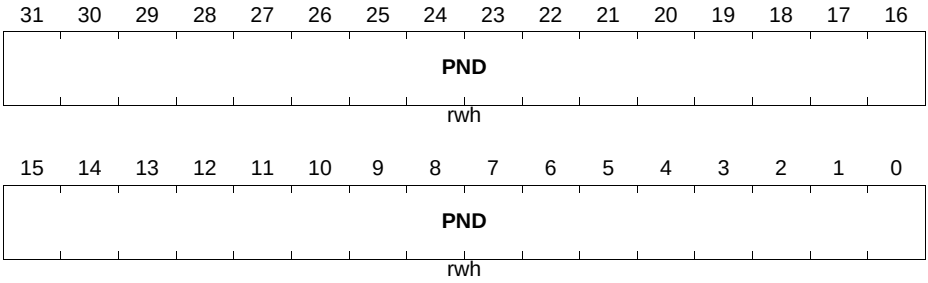
Field	Bits	Type	Description
EMPTY	24	rh	**List Empty Indication** 0_B At least one message object is allocated to list i. 1_B No message object is allocated to the list x. List x is empty.
0	[31:25]	r	**Reserved** Read as 0.

MSPNDk

When a message object n generates an interrupt request upon the transmission or reception of a message, then the request is routed to the interrupt output line selected by the bit field MOIPRn.TXINP or MOIPRn.RXINP of the message object n. As there are more message objects than interrupt output lines, an interrupt routine typically processes requests from more than one message object. Therefore, a priority selection mechanism is implemented in the MultiCAN module to select the highest priority object within a collection of message objects.

The Message Pending Register MSPNDk contains the pending interrupt notification of list i.

MSPNDk (k = 0-7)
Message Pending Register k **(140_H+k*4_H)** **Reset Value: 0000 0000_H**

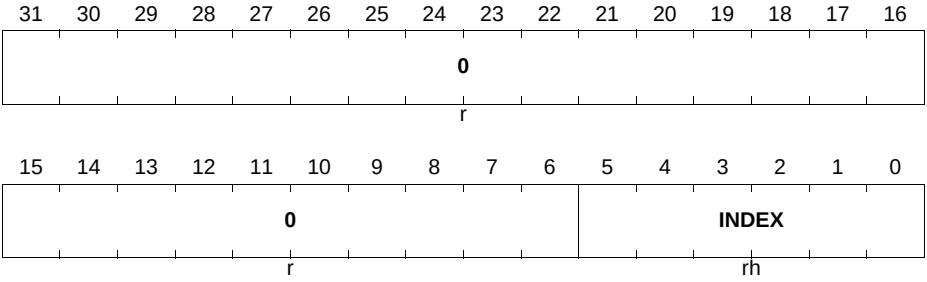


Field	Bits	Type	Description
PND	[31:0]	rwh	**Message Pending** When a message interrupt occurs, the message object sets a bit in one of the MSPND register, where the bit position is given by the MPN[4:0] field of the IPR register of the message object. The register selection n is given by the higher bits of MPN. The register bits can be cleared by software (write 0). Writing a 1 has no effect.

MSIDk

Each Message Pending Register has a Message Index Register MSIDk associated with it. The Message Index Register shows the active (set) pending bit with lowest bit position within groups of pending bits.

MSIDk (k = 0-7)
Message Index Register k **(180_H+k*4_H)** **Reset Value: 0000 0020$_H$**

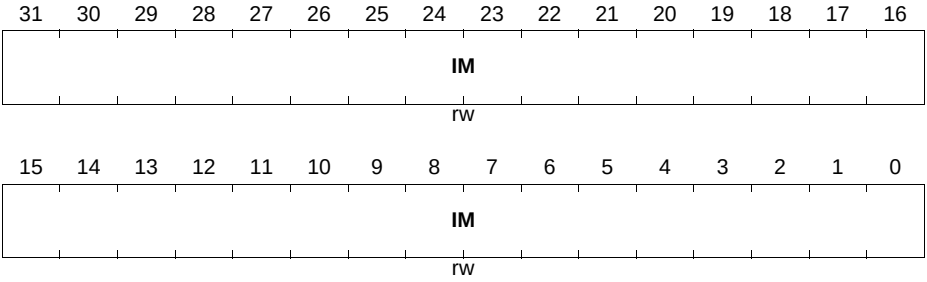


Field	Bits	Type	Description
INDEX	[5:0]	rh	**Message Pending Index** The value of INDEX is given by the bit position i of the pending bit of MSPNDk with the following properties: 1. MSPNDk[i] & IM[i] = 1 2. i = 0 or MSPNDk[i-1:0] & IM[i-1:0] = 0 If no bit of MSPNDk satisfies these conditions then INDEX reads 100000_B. Thus INDEX shows the position of the first pending bit of MSPNDk, in which only those bits of MSPNDk that are selected in the Message Index Mask Register are taken into account.
0	[31:6]	r	**Reserved** Read as 0; should be written with 0.

MSIMASK

The Message Index Mask Register MSIMASK selects individual bits for the calculation of the Message Pending Index. The Message Index Mask Register is used commonly for all Message Pending registers and their associated Message Index registers.

MSIMASK
Message Index Mask Register **($1C0_H$)** **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
IM	[31:0]	rw	**Message Index Mask** Only those bits in MSPNDk for which the corresponding Index Mask bits are set contribute to the calculation of the Message Index.

15.7.2 CAN Node Registers

The CAN node registers are built in for each CAN node of the MultiCAN module. They contain information that is directly related to the operation of the CAN nodes and are shared among the nodes.

NCR

The Node Control Register contains basic settings that determine the operation of the CAN node.

NCRx (x = 0-1)
Node x Control Register **(200_H+x*100_H)** **Reset Value: 0000 0001$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0							SUSEN	CALM	CCE	0	CANDIS	ALIE	LECIE	TRIE	INIT
r							rw	rw	rw	r	rw	rw	rw	rw	rwh

Field	Bits	Type	Description
INIT	0	rwh	**Node Initialization** 0_B Resetting bit INIT enables the participation of the node in the CAN traffic. If the CAN node is in the bus-off state, the ongoing bus-off recovery (which does not depend on the INIT bit) is continued. With the end of the bus-off recovery sequence the CAN node is allowed to take part in the CAN traffic. If the CAN node is not in the bus-off state, a sequence of 11 consecutive recessive bits must be detected before the node is allowed to take part in the CAN traffic. 1_B Setting this bit terminates the participation of this node in the CAN traffic. Any ongoing frame transfer is cancelled and the transmit line goes recessive. If the CAN node is in the bus-off state, then the running bus-off recovery sequence is continued. If the INIT bit is still set after the successful completion of the bus-off recovery sequence, i.e. after detecting 128 sequences of 11 consecutive recessive bits (11×1), then the CAN node leaves the bus-off state but remains inactive as long as INIT remains set. Bit INIT is automatically set when the CAN node enters the bus-off state (see **Page 15-20**).
TRIE	1	rw	**Transfer Interrupt Enable** TRIE enables the transfer interrupt of CAN node x. This interrupt is generated after the successful reception or transmission of a CAN frame in node x. 0_B Transfer interrupt is disabled. 1_B Transfer interrupt is enabled. Bit field NIPRx.TRINP selects the interrupt output line which becomes activated at this type of interrupt.

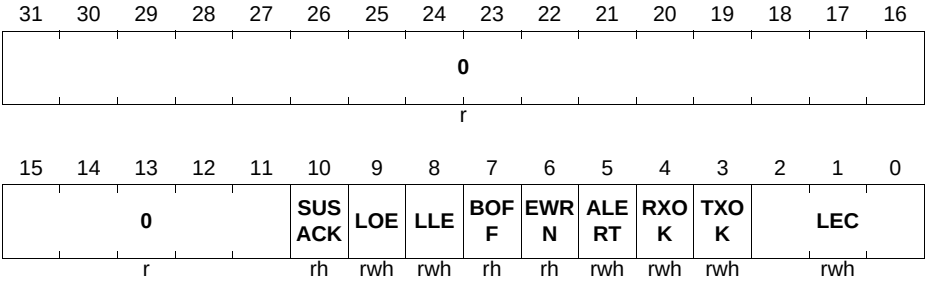
Field	Bits	Type	Description
LECIE	2	rw	**LEC Indicated Error Interrupt Enable** LECIE enables the last error code interrupt of CAN node x. This interrupt is generated with each update of bit field NSRx.LEC with LEC > 0 (CAN protocol error). 0_B Last error code interrupt is disabled. 1_B Last error code interrupt is enabled. Bit field NIPRx.LECINP selects the interrupt output line which becomes activated at this type of interrupt.
ALIE	3	rw	**Alert Interrupt Enable** ALIE enables the alert interrupt of CAN node x. This interrupt is generated by any one of the following events: • A change of bit NSRx.BOFF • A change of bit NSRx.EWRN • A List Length Error, which also sets bit NSRx.LLE • A List Object Error, which also sets bit NSRx.LOE • A Bit INIT is set by hardware 0_B Alert interrupt is disabled. 1_B Alert interrupt is enabled. Bit field NIPRx.ALINP selects the interrupt output line which becomes activated at this type of interrupt.
CANDIS	4	rw	**CAN Disable** Setting this bit disables the CAN node. The CAN node first waits until it is bus-idle or bus-off. Then bit INIT is automatically set, and an alert interrupt is generated if bit ALIE is set.
CCE	6	rw	**Configuration Change Enable** 0_B The Bit Timing Register, the Port Control Register, and the Error Counter Register may only be read. All attempts to modify them are ignored. 1_B The Bit Timing Register, the Port Control Register, and the Error Counter Register may be read and written.
CALM	7	rw	**CAN Analyzer Mode** If this bit is set, then the CAN node operates in Analyzer Mode. This means that messages may be received, but not transmitted. No acknowledge is sent on the CAN bus upon frame reception. Active-error flags are sent recessive instead of dominant. The transmit line is continuously held at recessive (1) level. Bit CALM can be written only while bit INIT is set.

Field	Bits	Type	Description
SUSEN	8	rw	**Suspend Enable** This bit makes it possible to set the CAN node into Suspend Mode via OCDS (on chip debug support): 0_B An OCDS suspend trigger is ignored by the CAN node. 1_B An OCDS suspend trigger disables the CAN node: As soon as the CAN node becomes bus-idle or bus-off, bit INIT is internally forced to 1 to disable the CAN node. The actual value of bit INIT remains unchanged. Bit SUSEN is reset via OCDS Reset.
0	[31:9], 5	r	**Reserved** Read as 0; should be written with 0.

NSR

The Node Status Register NSRx reports errors as well as successfully transferred CAN frames.

NSRx (x = 0-1)
Node x Status Register **(204_H+x*100_H)** **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
LEC	[2:0]	rwh	**Last Error Code** This bit field indicates the type of the last (most recent) CAN error. The encoding of this bit field is described in **Table 15-7**.
TXOK	3	rwh	**Message Transmitted Successfully** 0_B No successful transmission since last (most recent) flag reset. 1_B A message has been transmitted successfully (error-free and acknowledged by at least another node). TXOK must be reset by software (write 0). Writing 1 has no effect.
RXOK	4	rwh	**Message Received Successfully** 0_B No successful reception since last (most recent) flag reset. 1_B A message has been received successfully. RXOK must be reset by software (write 0). Writing 1 has no effect.

Field	Bits	Type	Description
ALERT	5	rwh	**Alert Warning** The ALERT bit is set upon the occurrence of one of the following events (the same events which also trigger an alert interrupt if ALIE is set): • A change of bit NSRx.BOFF • A change of bit NSRx.EWRN • A List Length Error, which also sets bit NSRx.LLE • A List Object Error, which also sets bit NSRx.LOE • Bit INIT has been set by hardware ALERT must be reset by software (write 0). Writing 1 has no effect.
EWRN	6	rh	**Error Warning Status** 0_B No warning limit exceeded. 1_B One of the error counters REC or TEC reached the warning limit EWRNLVL.
BOFF	7	rh	**Bus-off Status** 0_B CAN controller is not in the bus-off state. 1_B CAN controller is in the bus-off state.
LLE	8	rwh	**List Length Error** 0_B No List Length Error since last (most recent) flag reset. 1_B A List Length Error has been detected during message acceptance filtering. The number of elements in the list that belongs to this CAN node differs from the list SIZE given in the list termination pointer. LLE must be reset by software (write 0). Writing 1 has no effect.
LOE	9	rwh	**List Object Error** 0_B No List Object Error since last (most recent) flag reset. 1_B A List Object Error has been detected during message acceptance filtering. A message object with wrong LIST index entry in the Message Object Control Register has been detected. LOE must be reset by software (write 0). Writing 1 has no effect.

Field	Bits	Type	Description
SUSACK	10	rh	**Suspend Acknowledge** 0_B The CAN node is not in Suspend Mode or a suspend request is pending, but the CAN node has not yet reached bus-idle or bus-off. 1_B The CAN node is in Suspend Mode: The CAN node is inactive (bit NCR.INIT internally forced to 1) due to an OCDS suspend request.
0	[31:11]	r	**Reserved** Read as 0; should be written with 0.

Encoding of the LEC Bit Field

Table 15-7 Encoding of the LEC Bit Field

LEC Value	Signification
000_B	**No Error:** No error was detected for the last (most recent) message on the CAN bus.
001_B	**Stuff Error:** More than 5 equal bits in a sequence have occurred in a part of a received message where this is not allowed.
010_B	**Form Error:** A fixed format part of a received frame has the wrong format.
011_B	**Ack Error:** The transmitted message was not acknowledged by another node.
100_B	**Bit1 Error:** During a message transmission, the CAN node tried to send a recessive level (1) outside the arbitration field and the acknowledge slot, but the monitored bus value was dominant.
101_B	**Bit0 Error:** Two different conditions are signaled by this code: 1. During transmission of a message (or acknowledge bit, active-error flag, overload flag), the CAN node tried to send a dominant level (0), but the monitored bus value was recessive. 2. During bus-off recovery, this code is set each time a sequence of 11 recessive bits has been monitored. The CPU may use this code as indication that the bus is not continuously disturbed.

Table 15-7 Encoding of the LEC Bit Field (cont'd)

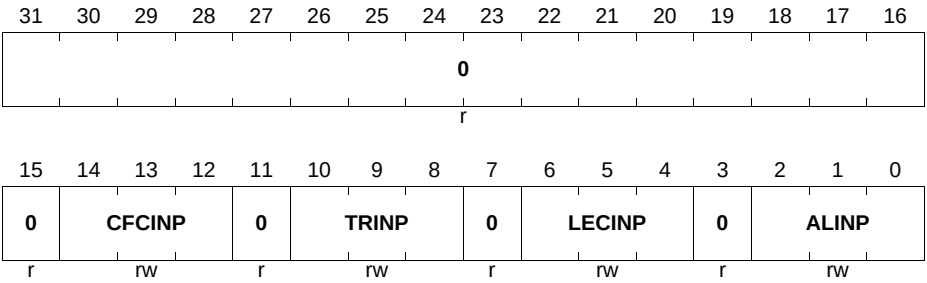
LEC Value	Signification
110_B	**CRC Error:** The CRC checksum of the received message was incorrect.
111_B	**CPU write to LEC:** Whenever the the CPU writes the value 111 to LEC, it takes the value 111. Whenever the CPU writes another value to LEC, the written LEC value is ignored.

NIPR

The four interrupt pointers in the Node Interrupt Pointer Register NIPRx select one out of the sixteen interrupt outputs individually for each type of CAN node interrupt. See also **Page 15-21** for more CAN node interrupt details.

NIPRx (x = 0-1)
Node x Interrupt Pointer Register

(208_H+x*100_H) **Reset Value: 0000 0000_H**



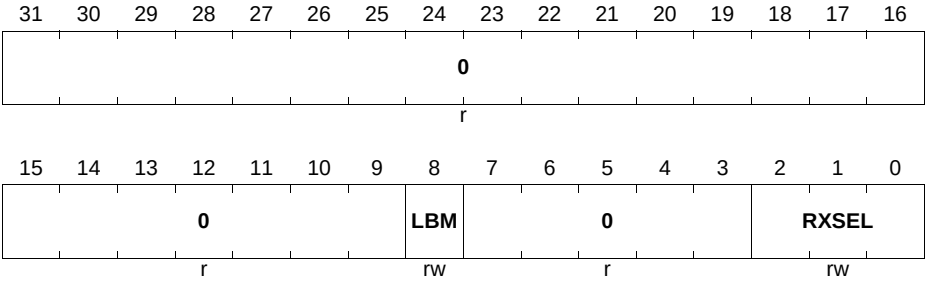
Field	Bits	Type	Description
ALINP	[2:0]	rw	**Alert Interrupt Node Pointer** ALINP selects the interrupt output line INT_Om (m = 0-7) for an alert interrupt of CAN Node x. 000_B Interrupt output line INT_O0 is selected. 001_B Interrupt output line INT_O1 is selected. $…_B$ … 111_B Interrupt output line INT_O7 is selected.
LECINP	[6:4]	rw	**Last Error Code Interrupt Node Pointer** LECINP selects the interrupt output line INT_Om (m = 0-7) for an LEC interrupt of CAN Node x. 000_B Interrupt output line INT_O0 is selected. 001_B Interrupt output line INT_O1 is selected. $…_B$ … 111_B Interrupt output line INT_O7 is selected.

Field	Bits	Type	Description
TRINP	[10:8]	rw	**Transfer OK Interrupt Node Pointer** TRINP selects the interrupt output line INT_Om (m = 0-7) for a transfer OK interrupt of CAN Node x. 000_B Interrupt output line INT_O0 is selected. 001_B Interrupt output line INT_O1 is selected. $\ldots_B$ … 111_B Interrupt output line INT_O7 is selected.
CFCINP	[14:12]	rw	**Frame Counter Interrupt Node Pointer** CFCINP selects the interrupt output line INT_Om (m = 0-7) for a transfer OK interrupt of CAN Node x. 000_B Interrupt output line INT_O0 is selected. 001_B Interrupt output line INT_O1 is selected. $\ldots_B$ … 111_B Interrupt output line INT_O7 is selected.
0	[31:15], 11, 7, 3	r	**Reserved** Read as 0; should be written with 0.

NPCR

The Node Port Control Register NPCRx configures the CAN bus transmit/receive ports. NPCRx can be written only if bit NCRx.CCE is set.

NPCRx (x = 0-1)
Node x Port Control Register **($20C_H$+x*100_H)** **Reset Value: 0000 0000_H**

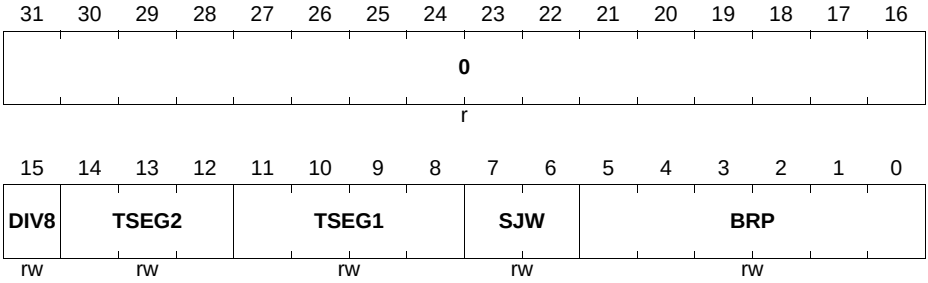


Field	Bits	Type	Description
RXSEL	[2:0]	rw	**Receive Select** RXSEL selects one out of 8 possible receive inputs. The CAN receive signal is performed only through the selected input. *Note: In XMC4[12]00, only specific combinations of RXSEL are available (see also **"MultiCAN I/O Control Selection and Setup" on Page 15-122**).*
LBM	8	rw	**Loop-Back Mode** 0_B Loop-Back Mode is disabled. 1_B Loop-Back Mode is enabled. This node is connected to an internal (virtual) loop-back CAN bus. All CAN nodes which are in Loop-Back Mode are connected to this virtual CAN bus so that they can communicate with each other internally. The external transmit line is forced recessive in Loop-Back Mode.
0	[7:3], [31:9]	r	**Reserved** Read as 0; should be written with 0.

NBTR

The Node Bit Timing Register NBTRx contains all parameters to set up the bit timing for the CAN transfer. NBTRx can be written only if bit NCRx.CCE is set.

NBTRx (x = 0-1)
Node x Bit Timing Register **(210_H+x*100_H)** **Reset Value: 0000 0000_H**



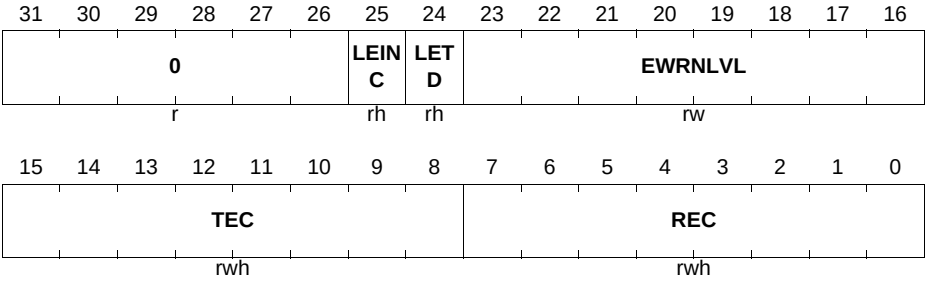
Field	Bits	Type	Description
BRP	[5:0]	rw	**Baud Rate Prescaler** The duration of one time quantum is given by (BRP + 1) clock cycles if DIV8 = 0. The duration of one time quantum is given by 8 × (BRP + 1) clock cycles if DIV8 = 1.
SJW	[7:6]	rw	**(Re) Synchronization Jump Width** (SJW + 1) time quanta are allowed for re-synchronization.
TSEG1	[11:8]	rw	**Time Segment Before Sample Point** (TSEG1 + 1) time quanta is the user-defined nominal time between the end of the synchronization segment and the sample point. It includes the propagation segment, which takes into account signal propagation delays. The time segment may be lengthened due to re-synchronization. Valid values for TSEG1 are 2 to 15.
TSEG2	[14:12]	rw	**Time Segment After Sample Point** (TSEG2 + 1) time quanta is the user-defined nominal time between the sample point and the start of the next synchronization segment. It may be shortened due to re-synchronization. Valid values for TSEG2 are 1 to 7.

Field	Bits	Type	Description
DIV8	15	rw	**Divide Prescaler Clock by 8** 0_B A time quantum lasts (BRP+1) clock cycles. 1_B A time quantum lasts 8 × (BRP+1) clock cycles.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

NECNT

The Node Error Counter Register NECNTx contains the CAN receive and transmit error counter as well as some additional bits to ease error analysis. NECNTx can be written only if bit NCRx.CCE is set.

NECNTx (x = 0-1)
Node x Error Counter Register (214_H+x*100_H) Reset Value: 0060 0000_H



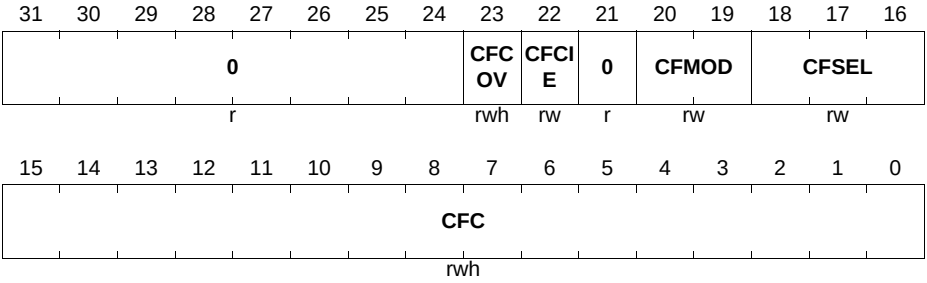
Field	Bits	Type	Description
REC	[7:0]	rwh	**Receive Error Counter** Bit field REC contains the value of the receive error counter of CAN node x.
TEC	[15:8]	rwh	**Transmit Error Counter** Bit field TEC contains the value of the transmit error counter of CAN node x.
EWRNLVL	[23:16]	rw	**Error Warning Level** Bit field EWRNLVL determines the threshold value (warning level, default 96) to be reached in order to set the corresponding error warning bit EWRN.
LETD	24	rh	**Last Error Transfer Direction** 0_B The last error occurred while the CAN node x was receiver (REC has been incremented). 1_B The last error occurred while the CAN node x was transmitter (TEC has been incremented).
LEINC	25	rh	**Last Error Increment** 0_B The last error led to an error counter increment of 1. 1_B The last error led to an error counter increment of 8.

Field	Bits	Type	Description
0	[31:26]	r	**Reserved** Read as 0; should be written with 0.

NFCR

The Node Frame Counter Register NFCRx contains the actual value of the frame counter as well as control and status bits of the frame counter.

NFCRx (x = 0-1)
Node x Frame Counter Register (218_H+x*100_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
CFC	[15:0]	rwh	**CAN Frame Counter** In Frame Count Mode (CFMOD = 00_B), this bit field contains the frame count value. In Time Stamp Mode (CFMOD = 01_B), this bit field contains the captured bit time count value, captured with the start of a new frame. In all Bit Timing Analysis Modes (CFMOD = 10_B), CFC always displays the number of f_{CLC} clock cycles (measurement result) minus 1. Example: a CFC value of 34 in measurement mode CFSEL = 000_B means that 35 f_{CLC} clock cycles have been elapsed between the most recent two dominant edges on the receive input.

Field	Bits	Type	Description
CFSEL	[18:16]	rw	**CAN Frame Count Selection** This bit field selects the function of the frame counter for the chosen frame count mode. **Frame Count Mode** Bit 0 If Bit 0 of CFSEL is set, then CFC is incremented each time a foreign frame (i.e. a frame not matching to a message object) has been received on the CAN bus. Bit 1 If Bit 1 of CFSEL is set, then CFC is incremented each time a frame matching to a message object has been received on the CAN bus. Bit 2 If Bit 2 of CFSEL is set, then CFC is incremented each time a frame has been transmitted successfully by the node. **Time Stamp Mode** 000_B The frame counter is incremented (internally) at the beginning of a new bit time. The value is sampled during the SOF bit of a new frame. The sampled value is visible in the CFC field. **Bit Timing Mode** The available bit timing measurement modes are shown in **Table 15-8**. If CFCIE is set, then an interrupt on request node x (where x is the CAN node number) is generated with a CFC update.
CFMOD	[20:19]	rw	**CAN Frame Counter Mode** This bit field determines the operation mode of the frame counter. 00_B Frame Count Mode: The frame counter is incremented upon the reception and transmission of frames. 01_B Time Stamp Mode: The frame counter is used to count bit times. 10_B Bit Timing Mode: The frame counter is used for analysis of the bit timing.

Field	Bits	Type	Description
CFCIE	22	rw	**CAN Frame Count Interrupt Enable** CFCIE enables the CAN frame counter overflow interrupt of CAN node x. 0_B CAN frame counter overflow interrupt is disabled. 1_B CAN frame counter overflow interrupt is enabled. Bit field NIPRx.CFCINP selects the interrupt output line that is activated at this type of interrupt.
CFCOV	23	rwh	**CAN Frame Counter Overflow Flag** Flag CFCOV is set upon a frame counter overflow (transition from $FFFF_H$ to 0000_H). In bit timing analysis mode, CFCOV is set upon an update of CFC. An interrupt request is generated if CFCIE = 1. 0_B No overflow has occurred since last flag reset. 1_B An overflow has occurred since last flag reset. CFCOV must be reset by software.
0	21, [31:24]	r	**Reserved** Read as 0; should be written with 0.

Bit Timing Analysis Modes

Table 15-8 Bit Timing Analysis Modes (CFMOD = 10)

CFSEL	Measurement
000_B	Whenever a dominant edge (transition from 1 to 0) is monitored on the receive input, the time (measured in clock cycles) between this edge and the most recent dominant edge is stored in CFC.
001_B	Whenever a recessive edge (transition from 0 to 1) is monitored on the receive input the time (measured in clock cycles) between this edge and the most recent dominant edge is stored in CFC.
010_B	Whenever a dominant edge is received as a result of a transmitted dominant edge, the time (clock cycles) between both edges is stored in CFC.
011_B	Whenever a recessive edge is received as a result of a transmitted recessive edge, the time (clock cycles) between both edges is stored in CFC.
100_B	Whenever a dominant edge that qualifies for synchronization is monitored on the receive input, the time (measured in clock cycles) between this edge and the most recent sample point is stored in CFC.

Table 15-8 Bit Timing Analysis Modes (CFMOD = 10) (cont'd)

CFSEL	Measurement
101_B	With each sample point, the time (measured in clock cycles) between the start of the new bit time and the start of the previous bit time is stored in CFC[11:0]. Additional information is written to CFC[15:12] at each sample point: CFC[15]: Transmit value of actual bit time CFC[14]: Receive sample value of actual bit time CFC[13:12]: CAN bus information (see **Table 15-9**)
110_B	Reserved, do not use this combination.
111_B	Reserved, do not use this combination.

Table 15-9 CAN Bus State Information

CFC[13:12]	CAN Bus State
00_B	**NoBit** The CAN bus is idle, performs bit (de-) stuffing or is in one of the following frame segments: SOF, SRR, CRC, delimiters, first 6 EOF bits, IFS.
01_B	**NewBit** This code represents the first bit of a new frame segment. The current bit is the first bit in one of the following frame segments: Bit 10 (MSB) of standard ID (transmit only), RTR, reserved bits, IDE, DLC(MSB), bit 7 (MSB) in each data byte and the first bit of the ID extension.
10_B	**Bit** This code represents a bit inside a frame segment with a length of more than one bit (not the first bit of those frame segments that is indicated by NewBit). The current bit is processed within one of the following frame segments: ID bits (except first bit of standard ID for transmission and first bit of ID extension), DLC (3 LSB) and bits 6-0 in each data byte.
11_B	**Done** The current bit is in one of the following frame segments: Acknowledge slot, last bit of EOF, active/passive-error frame, overload frame. Two or more directly consecutive Done codes signal an Error Frame.

15.7.3 Message Object Registers

MOCTR

The Message Object Control Register MOCTRn and the Message Object Status Register MOSTATn are located at the same address offset within a message object address block (offset address $1C_H$). The MOCTRn is a write-only register that makes it possible to set/reset CAN transfer related control bits through software.

MOCTR0
Message Object 0 Control Register ($101C_H$) **Reset Value: $0100\ 0000_H$**
MOCTRn (n = 1-62)
Message Object n Control Register
($101C_H+n*20_H$)
Reset Value: $((n+1)*01000000_H)+((n-1)*00010000_H)$

MOCTR63
Message Object 63 Control Register ($17FC_H$) **Reset Value: $3F3E\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0				SETDIR	SETTXEN1	SETTXEN0	SETTXRQ	SETRXEN	SETRTSEL	SETMSGVAL	SETMSGLST	SETNEWDAT	SETRXUPD	SETTXPND	SETRXPND
w				w	w	w	w	w	w	w	w	w	w	w	w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				RESDIR	RESTXEN1	RESTXEN0	RESTXRQ	RESRXEN	RESRTSEL	RESMSGVAL	RESMSGLST	RESNEWDAT	RESRXUPD	RESTXPND	RESRXPND
w				w	w	w	w	w	w	w	w	w	w	w	w

Field	Bits	Type	Description
RESRXPND, SETRXPND	0, 16	w	**Reset/Set Receive Pending** These bits control the set/reset condition for RXPND (see **Table 15-10**).
RESTXPND, SETTXPND	1, 17	w	**Reset/Set Transmit Pending** These bits control the set/reset condition for TXPND (see **Table 15-10**).
RESRXUPD, SETRXUPD	2, 18	w	**Reset/Set Receive Updating** These bits control the set/reset condition for RXUPD (see **Table 15-10**).

Field	Bits	Type	Description
RESNEWDAT, SETNEWDAT	3, 19	w	**Reset/Set New Data** These bits control the set/reset condition for NEWDAT (see **Table 15-10**).
RESMSGLST, SETMSGLST	4, 20	w	**Reset/Set Message Lost** These bits control the set/reset condition for MSGLST (see **Table 15-10**).
RESMSGVAL, SETMSGVAL	5, 21	w	**Reset/Set Message Valid** These bits control the set/reset condition for MSGVAL (see **Table 15-10**).
RESRTSEL, SETRTSEL	6, 22	w	**Reset/Set Receive/Transmit Selected** These bits control the set/reset condition for RTSEL (see **Table 15-10**).
RESRXEN, SETRXEN	7, 23	w	**Reset/Set Receive Enable** These bits control the set/reset condition for RXEN (see **Table 15-10**).
RESTXRQ, SETTXRQ	8, 24	w	**Reset/Set Transmit Request** These bits control the set/reset condition for TXRQ (see **Table 15-10**).
RESTXEN0, SETTXEN0	9, 25	w	**Reset/Set Transmit Enable 0** These bits control the set/reset condition for TXEN0 (see **Table 15-10**).
RESTXEN1, SETTXEN1	10, 26	w	**Reset/Set Transmit Enable 1** These bits control the set/reset condition for TXEN1 (see **Table 15-10**).
RESDIR, SETDIR	11, 27	w	**Reset/Set Message Direction** These bits control the set/reset condition for DIR (see **Table 15-10**).
0	[15:12], [31:28]	w	**Reserved** Should be written with 0.

Table 15-10 Reset/Set Conditions for Bits in Register MOCTRn

RESy Bit[1)]	SETy Bit	Action on Write
Write 0	Write 0	Leave element unchanged
	No write	
No write	Write 0	
Write 1	Write 1	
Write 1	Write 0	Reset element
	No write	
Write 0	Write 1	Set element
No write		

1) The parameter “y” stands for the second part of the bit name (“RXPND”, “TXPND”, … up to “DIR”).

MOSTAT

The MOSTATn is a read-only register that indicates message object list status information such as the number of the current message object predecessor and successor message object, as well as the list number to which the message object is assigned.

MOSTAT0
Message Object 0 Status Register (101C$_H$) Reset Value: 0100 0000$_H$
MOSTATn (n = 1-62)
Message Object n Status Register
(101C$_H$+n*20$_H$)
Rest Value: ((n+1)*01000000$_H$)+((n-1)*00010000$_H$)
MOSTAT63
Message Object 63 Status Register (17FC$_H$) Reset Value: 3F3E 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PNEXT								PPREV							
rh								rh							

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LIST				DIR	TX EN1	TX EN0	TX RQ	RX EN	RTS EL	MSG VAL	MSG LST	NEW DAT	RX UPD	TX PND	RX PND
rh				rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
RXPND	0	rh	**Receive Pending** 0_B No CAN message has been received. 1_B A CAN message has been received by the message object n, either directly or via gateway copy action. RXPND is set by hardware and must be reset by software.
TXPND	1	rh	**Transmit Pending** 0_B No CAN message has been transmitted. 1_B A CAN message from message object n has been transmitted successfully over the CAN bus. TXPND is set by hardware and must be reset by software.

Field	Bits	Type	Description
RXUPD	2	rh	**Receive Updating** 0_B No receive update ongoing. 1_B Message identifier, DLC, and data of the message object are currently updated.
NEWDAT	3	rh	**New Data** 0_B No update of the message object n since last flag reset. 1_B Message object n has been updated. NEWDAT is set by hardware after a received CAN frame has been stored in message object n. NEWDAT is cleared by hardware when a CAN transmission of message object n has been started. NEWDAT should be set by software after the new transmit data has been stored in message object n to prevent the automatic reset of TXRQ at the end of an ongoing transmission.
MSGLST	4	rh	**Message Lost** 0_B No CAN message is lost. 1_B A CAN message is lost because NEWDAT has become set again when it has already been set.
MSGVAL	5	rh	**Message Valid** 0_B Message object n is not valid. 1_B Message object n is valid. Only a valid message object takes part in CAN transfers.

Field	Bits	Type	Description
RTSEL	6	rh	**Receive/Transmit Selected** 0_B Message object n is not selected for receive or transmit operation. 1_B Message object n is selected for receive or transmit operation. **Frame Reception:** RTSEL is set by hardware when message object n has been identified for storage of a CAN frame that is currently received. Before a received frame becomes finally stored in message object n, a check is performed to determine if RTSEL is set. Thus the CPU can suppress a scheduled frame delivery to this message object n by clearing RTSEL by software. **Frame Transmission:** RTSEL is set by hardware when message object n has been identified to be transmitted next. A check is performed to determine if RTSEL is still set before message object n is actually set up for transmission and bit NEWDAT is cleared. It is also checked that RTSEL is still set before its message object n is verified due to the successful transmission of a frame. RTSEL needs to be checked only when the context of message object n changes, and a conflict with an ongoing frame transfer shall be avoided. In all other cases, RTSEL can be ignored. RTSEL has no impact on message acceptance filtering. RTSEL is not cleared by hardware.
RXEN	7	rh	**Receive Enable** 0_B Message object n is not enabled for frame reception. 1_B Message object n is enabled for frame reception. RXEN is evaluated for receive acceptance filtering only.

Field	Bits	Type	Description
TXRQ	8	rh	**Transmit Request** 0_B No transmission of message object n is requested. 1_B Transmission of message object n on the CAN bus is requested. The transmit request becomes valid only if TXRQ, TXEN0, TXEN1 and MSGVAL are set. TXRQ is set by hardware if a matching Remote Frame has been received correctly. TXRQ is reset by hardware if message object n has been transmitted successfully and NEWDAT is not set again by software.
TXEN0	9	rh	**Transmit Enable 0** 0_B Message object n is not enabled for frame transmission. 1_B Message object n is enabled for frame transmission. Message object n can be transmitted only if both bits, TXEN0 and TXEN1, are set. The user may clear TXEN0 in order to inhibit the transmission of a message that is currently updated, or to disable automatic response of Remote Frames.
TXEN1	10	rh	**Transmit Enable 1** 0_B Message object n is not enabled for frame transmission. 1_B Message object n is enabled for frame transmission. Message object n can be transmitted only if both bits, TXEN0 and TXEN1, are set. TXEN1 is used by the MultiCAN module for selecting the active message object in the Transmit FIFOs.

Field	Bits	Type	Description
DIR	11	rh	**Message Direction** 0_B Receive Object selected: With TXRQ = 1, a Remote Frame with the identifier of message object n is scheduled for transmission. On reception of a Data Frame with matching identifier, the message is stored in message object n. 1_B Transmit Object selected: If TXRQ = 1, message object n is scheduled for transmission of a Data Frame. On reception of a Remote Frame with matching identifier, bit TXRQ is set.
LIST	[15:12]	rh	**List Allocation** LIST indicates the number of the message list to which message object n is allocated. LIST is updated by hardware when the list allocation of the object is modified by a panel command.
PPREV	[23:16]	rh	**Pointer to Previous Message Object** PPREV holds the message object number of the previous message object in a message list structure.
PNEXT	[31:24]	rh	**Pointer to Next Message Object** PNEXT holds the message object number of the next message object in a message list structure.

Table 15-11 MOSTATn Reset Values

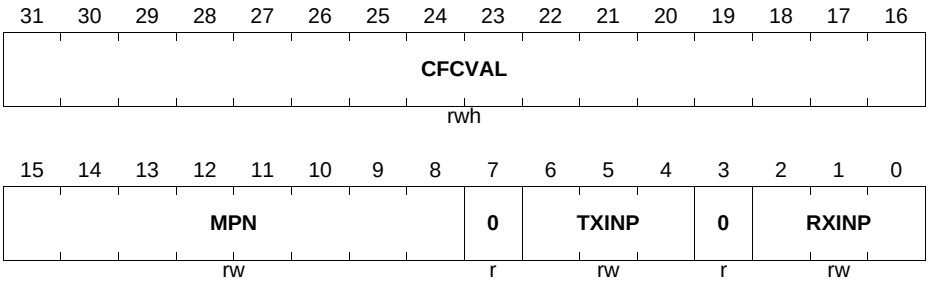
Message Object	PNEXT	PPREV	Reset Value
0	1	0	$0100\ 0000_H$
1	2	0	$0200\ 0000_H$
2	3	1	$0301\ 0000_H$
3	4	2	$0402\ 0000_H$
…	…	…	…
60	61	59	$3D3B\ 0000_H$
61	62	60	$3E3C\ 0000_H$
62	63	61	$3F3D\ 0000_H$
63	63	62	$3F3E\ 0000_H$

MOIPR

The Message Object Interrupt Pointer Register MOIPRn holds the message interrupt pointers, the message pending number, and the frame counter value of message object n.

MOIPRn (n = 0-63)
Message Object n Interrupt Pointer Register

(1008_H+n*20_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
RXINP	[2:0]	rw	**Receive Interrupt Node Pointer** RXINP selects the interrupt output line INT_Om (m = 0-7) for a receive interrupt event of message object n. RXINP can also be taken for message pending bit selection (see **Page 15-37**). 000_B Interrupt output line INT_O0 is selected. 001_B Interrupt output line INT_O1 is selected. $\cdots_B$ … 111_B Interrupt output line INT_O7 is selected.
TXINP	[6:4]	rw	**Transmit Interrupt Node Pointer** TXINP selects the interrupt output line INT_Om (m = 0-7) for a transmit interrupt event of message object n. TXINP can also be taken for message pending bit selection (see **Page 15-37**). 000_B Interrupt output line INT_O0 is selected. 001_B Interrupt output line INT_O1 is selected. $\cdots_B$ … 111_B Interrupt output line INT_O7 is selected.

Field	Bits	Type	Description
MPN	[15:8]	rw	**Message Pending Number** This bit field selects the bit position of the bit in the Message Pending Register that is set upon a message object n receive/transmit interrupt.
CFCVAL	[31:16]	rwh	**CAN Frame Counter Value** When a message is stored in message object n or message object n has been successfully transmitted, the CAN frame counter value NFCRx.CFC is then copied to CFCVAL.
0	7, 3	r	**Reserved** Read as 0; should be written with 0.

MOFCR

The Message Object Function Control Register MOFCRn contains bits that select and configure the function of the message object. It also holds the CAN data length code.

MOFCRn (n = 0-63)
Message Object n Function Control Register
(1000_H+n*20_H) **Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0				DLC				STT	SDT	RMM	FRR EN	0	OVIE	TXIE	RXIE
rw				rwh				rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				DATC	DLCC	IDC	GDFS	0				MMC			
rw				rw	rw	rw	rw	rw				rw			

Field	Bits	Type	Description
MMC	[3:0]	rw	**Message Mode Control** MMC controls the message mode of message object n. 0000_B Standard Message Object 0001_B Receive FIFO Base Object 0010_B Transmit FIFO Base Object 0011_B Transmit FIFO Slave Object 0100_B Gateway Source Object $..._B$ Reserved
GDFS	8	rw	**Gateway Data Frame Send** 0_B TXRQ is unchanged in the destination object. 1_B TXRQ is set in the gateway destination object after the internal transfer from the gateway source to the gateway destination object. Applicable only to a gateway source object; ignored in other nodes.

Field	Bits	Type	Description
IDC	9	rw	**Identifier Copy** 0_B The identifier of the gateway source object is not copied. 1_B The identifier of the gateway source object (after storing the received frame in the source) is copied to the gateway destination object. Applicable only to a gateway source object; ignored in other nodes.
DLCC	10	rw	**Data Length Code Copy** 0_B Data length code is not copied. 1_B Data length code of the gateway source object (after storing the received frame in the source) is copied to the gateway destination object. Applicable only to a gateway source object; ignored in other nodes.
DATC	11	rw	**Data Copy** 0_B Data fields are not copied. 1_B Data fields in registers MODATALn and MODATAHn of the gateway source object (after storing the received frame in the source) are copied to the gateway destination. Applicable only to a gateway source object; ignored in other nodes.
RXIE	16	rw	**Receive Interrupt Enable** RXIE enables the message receive interrupt of message object n. This interrupt is generated after reception of a CAN message (independent of whether the CAN message is received directly or indirectly via a gateway action). 0_B Message receive interrupt is disabled. 1_B Message receive interrupt is enabled. Bit field MOIPRn.RXINP selects the interrupt output line which becomes activated at this type of interrupt.

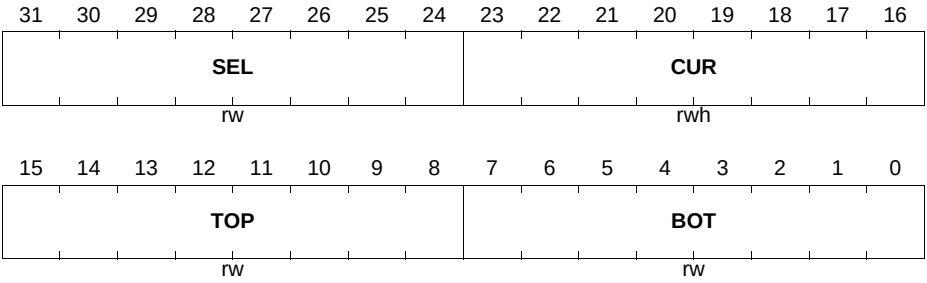
Field	Bits	Type	Description
TXIE	17	rw	**Transmit Interrupt Enable** TXIE enables the message transmit interrupt of message object n. This interrupt is generated after the transmission of a CAN message. 0_B Message transmit interrupt is disabled. 1_B Message transmit interrupt is enabled. Bit field MOIPRn.TXINP selects the interrupt output line which becomes activated at this type of interrupt.
OVIE	18	rw	**Overflow Interrupt Enable** OVIE enables the FIFO full interrupt of message object n. This interrupt is generated when the pointer to the current message object (CUR) reaches the value of SEL in the FIFO/Gateway Pointer Register. 0_B FIFO full interrupt is disabled. 1_B FIFO full interrupt is enabled. If message object n is a Receive FIFO base object, bit field MOIPRn.TXINP selects the interrupt output line which becomes activated at this type of interrupt. If message object n is a Transmit FIFO base object, bit field MOIPRn.RXINP selects the interrupt output line which becomes activated at this type of interrupt. For all other message object modes, bit OVIE has no effect.
FRREN	20	rw	**Foreign Remote Request Enable** Specifies whether the TXRQ bit is set in message object n or in a foreign message object referenced by the pointer CUR. 0_B TXRQ of message object n is set on reception of a matching Remote Frame. 1_B TXRQ of the message object referenced by the pointer CUR is set on reception of a matching Remote Frame.

Field	Bits	Type	Description
RMM	21	rw	**Transmit Object Remote Monitoring** 0_B Remote monitoring is disabled: Identifier, IDE bit, and DLC of message object n remain unchanged upon the reception of a matching Remote Frame. 1_B Remote monitoring is enabled: Identifier, IDE bit, and DLC of a matching Remote Frame are copied to transmit object n in order to monitor incoming Remote Frames. Bit RMM applies only to transmit objects and has no effect on receive objects.
SDT	22	rw	**Single Data Transfer** If SDT = 1 and message object n is not a FIFO base object, then MSGVAL is reset when this object has taken part in a successful data transfer (receive or transmit). If SDT = 1 and message object n is a FIFO base object, then MSGVAL is reset when the pointer to the current object CUR reaches the value of SEL in the FIFO/Gateway Pointer Register. With SDT = 0, bit MSGVAL is not affected.
STT	23	rw	**Single Transmit Trial** If this bit is set, then TXRQ is cleared on transmission start of message object n. Thus, no transmission retry is performed in case of transmission failure.
DLC	[27:24]	rwh	**Data Length Code** Bit field determines the number of data bytes for message object n. Valid values for DLC are 0 to 8. A value of DLC > 8 results in a data length of 8 data bytes. If a frame with DLC > 8 is received, the received value is stored in the message object.
0	[7:4], [15:12], 19	rw	**Reserved** Read as 0 after reset; value last written is read back; should be written with 0.

MOFGPR

The Message Object FIFO/Gateway Pointer register MOFGPRn contains a set of message object link pointers that are used for FIFO and gateway operations.

MOFGPRn (n = 0-63)
Message Object n FIFO/Gateway Pointer Register
(1004_H+n*20_H) **Reset Value: 0000 0000_H**

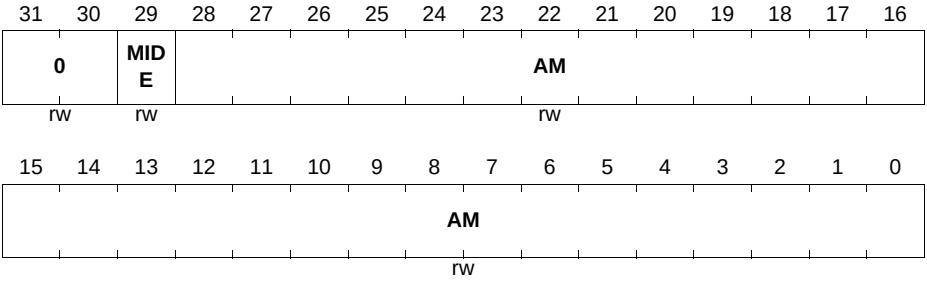


Field	Bits	Type	Description
BOT	[7:0]	rw	**Bottom Pointer** Bit field BOT points to the first element in a FIFO structure.
TOP	[15:8]	rw	**Top Pointer** Bit field TOP points to the last element in a FIFO structure.
CUR	[23:16]	rwh	**Current Object Pointer** Bit field CUR points to the actual target object within a FIFO/Gateway structure. After a FIFO/gateway operation CUR is updated with the message number of the next message object in the list structure (given by PNEXT of the message control register) until it reaches the FIFO top element (given by TOP) when it is reset to the bottom element (given by BOT).
SEL	[31:24]	rw	**Object Select Pointer** Bit field SEL is the second (software) pointer to complement the hardware pointer CUR in the FIFO structure. SEL is used for monitoring purposes (FIFO interrupt generation).

MOAMR

Message Object n Acceptance Mask Register MOAMRn contains the mask bits for the acceptance filtering of the message object n.

MOAMRn (n = 0-63)
Message Object n Acceptance Mask Register
($100C_H$+n*20_H) **Reset Value: 3FFF FFFF$_H$**



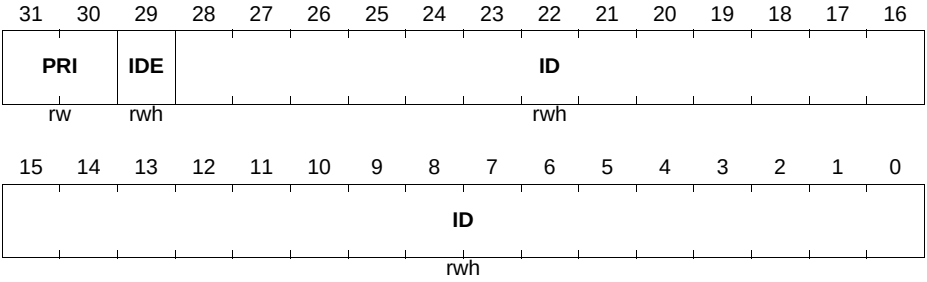
Field	Bits	Type	Description
AM	[28:0]	rw	**Acceptance Mask for Message Identifier** Bit field AM is the 29-bit mask for filtering incoming messages with standard identifiers (AM[28:18]) or extended identifiers (AM[28:0]). For standard identifiers, bits AM[17:0] are “don’t care”.
MIDE	29	rw	**Acceptance Mask Bit for Message IDE Bit** 0_B Message object n accepts the reception of both, standard and extended frames. 1_B Message object n receives frames only with matching IDE bit.
0	[31:30]	rw	**Reserved** Read as 0 after reset; value last written is read back; should be written with 0.

MOAR

Message Object n Arbitration Register MOARn contains the CAN identifier of the message object.

MOARn (n = 0-63)
Message Object n Arbitration Register

(1018_H+n*20_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
ID	[28:0]	rwh	**CAN Identifier of Message Object n** Identifier of a standard message (ID[28:18]) or an extended message (ID[28:0]). For standard identifiers, bits ID[17:0] are “don’t care”.
IDE	29	rwh	**Identifier Extension Bit of Message Object n** 0_B Message object n handles standard frames with 11-bit identifier. 1_B Message object n handles extended frames with 29-bit identifier.

Field	Bits	Type	Description
PRI	[31:30]	rw	**Priority Class** PRI assigns one of the four priority classes 0, 1, 2, 3 to message object n. A lower PRI number defines a higher priority. Message objects with lower PRI value always win acceptance filtering for frame reception and transmission over message objects with higher PRI value. Acceptance filtering based on identifier/mask and list position is performed only between message objects of the same priority class. PRI also determines the acceptance filtering method for transmission: 00_B Applicable only if TTCAN is available. 01_B Transmit acceptance filtering is based on the list order. This means that message object n is considered for transmission only if there is no other message object with valid transmit request (MSGVAL & TXEN0 & TXEN1 = 1) somewhere before this object in the list. 10_B Transmit acceptance filtering is based on the CAN identifier. This means, message object n is considered for transmission only if there is no other message object with higher priority identifier + IDE + DIR (with respect to CAN arbitration rules) somewhere in the list (see **Table 15-12**). 11_B Transmit acceptance filtering is based on the list order (as PRI = 01_B).

Transmit Priority of Msg. Objects based on CAN Arbitration Rules

Table 15-12 Transmit Priority of Msg. Objects Based on CAN Arbitration Rules

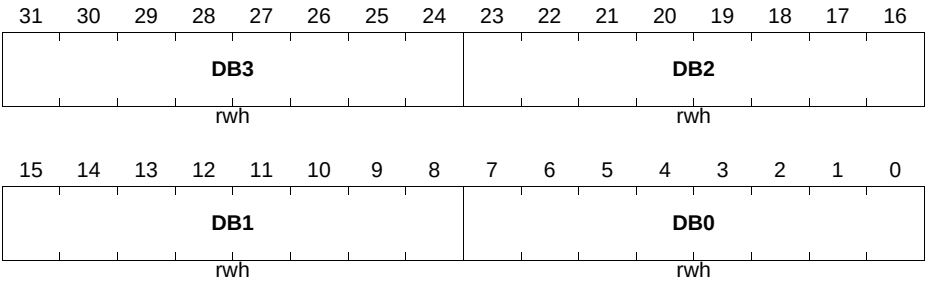
Settings of Arbitrarily Chosen Message Objects A and B, (A has higher transmit priority than B)	Comment
A.MOAR[28:18] < B.MOAR[28:18] (11-bit standard identifier of A less than 11-bit standard identifier of B)	Messages with lower standard identifier have higher priority than messages with higher standard identifier. MOAR[28] is the most significant bit (MSB) of the standard identifier. MOAR[18] is the least significant bit of the standard identifier.
A.MOAR[28:18] = B.MOAR[28:18] A.MOAR.IDE = 0 (send Standard Frame) B.MOAR.IDE = 1 (send Extended Frame)	Standard Frames have higher transmit priority than Extended Frames with equal standard identifier.
A.MOAR[28:18] = B.MOAR[28:18] A.MOAR.IDE = B.MOAR.IDE = 0 A.MOCTR.DIR = 1 (send Data Frame) B.MOCTR.DIR = 0 (send Remote Fame)	Standard Data Frames have higher transmit priority than standard Remote Frames with equal identifier.
A.MOAR[28:0] = B.MOAR[28:0] A.MOAR.IDE = B.MOAR.IDE = 1 A.MOCTR.DIR = 1 (send Data Frame) B.MOCTR.DIR = 0 (send Remote Frame)	Extended Data Frames have higher transmit priority than Extended Remote Frames with equal identifier.
A.MOAR[28:0] < B.MOAR[28:0] A.MOAR.IDE = B.MOAR.IDE = 1 (29-bit identifier)	Extended Frames with lower identifier have higher transmit priority than Extended Frames with higher identifier. MOAR[28] is the most significant bit (MSB) of the overall identifier (standard identifier MOAR[28:18] and identifier extension MOAR[17:0]). MOAR[0] is the least significant bit (LSB) of the overall identifier.

MODATAL

Message Object n Data Register Low MODATALn contains the lowest four data bytes of message object n. Unused data bytes are set to zero upon reception and ignored for transmission.

MODATALn (n = 0-63)
Message Object n Data Register Low

(1010_H+n*20_H) **Reset Value: 0000 0000_H**



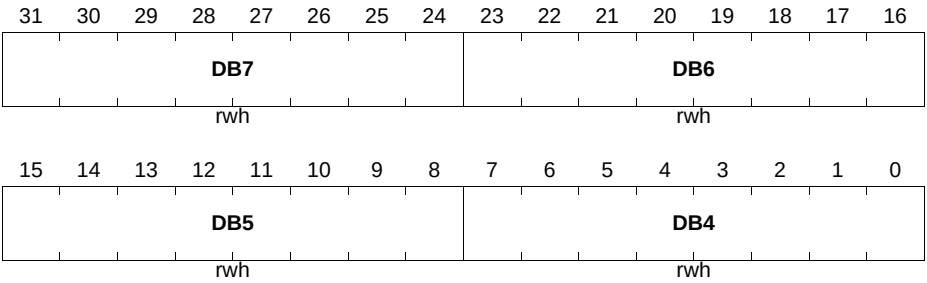
Field	Bits	Type	Description
DB0	[7:0]	rwh	**Data Byte 0 of Message Object n**
DB1	[15:8]	rwh	**Data Byte 1 of Message Object n**
DB2	[23:16]	rwh	**Data Byte 2 of Message Object n**
DB3	[31:24]	rwh	**Data Byte 3 of Message Object n**

MODATAH

Message Object n Data Register High MODATAH contains the highest four data bytes of message object n. Unused data bytes are set to zero upon reception and ignored for transmission.

MODATAHn (n = 0-63)
Message Object n Data Register High

(1014_H+n*20_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
DB4	[7:0]	rwh	**Data Byte 4 of Message Object n**
DB5	[15:8]	rwh	**Data Byte 5 of Message Object n**
DB6	[23:16]	rwh	**Data Byte 6 of Message Object n**
DB7	[31:24]	rwh	**Data Byte 7 of Message Object n**

15.7.4 MultiCAN Module External Registers

The registers listed in **Figure 15-25** must be programmed for proper operation of the MultiCAN module.

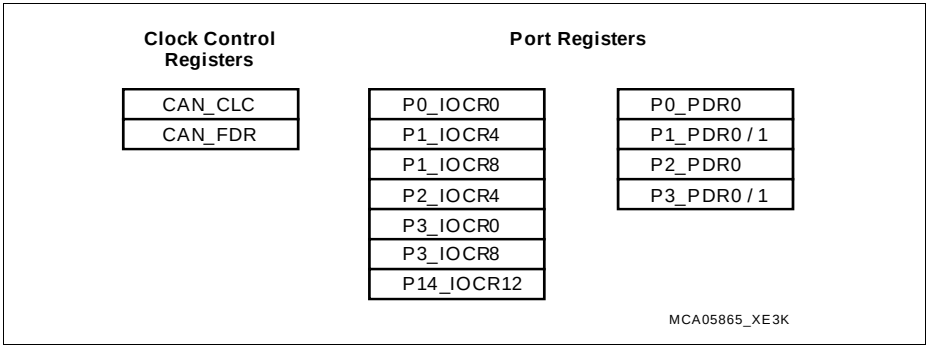


Figure 15-25 CAN Implementation-specific Special Function Registers

Table 15-13 MultiCAN Module External Registers

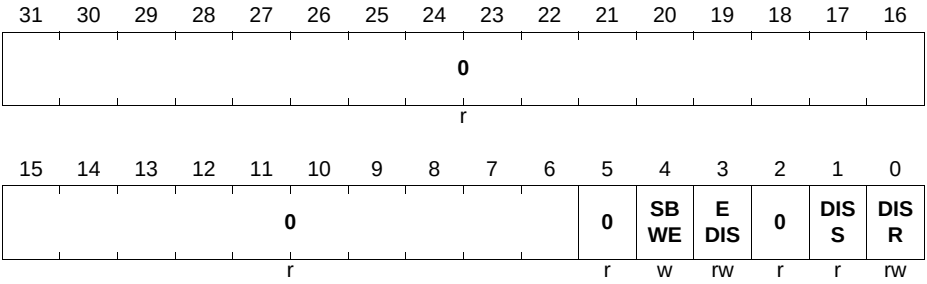
Short Name	Description	Offset Addr	Access Mode[1)]		Description see
			Read	Write	
Module Identification Registers					
ID	Module Identification Register	008_H	U, PV	U, PV	Page 15-62
Clock Control Registers					
CLC	Clock Control Register	000_H	U, PV	PV, E	Page 15-115
FDR	Fractional Divider Register	$00C_H$	U, PV	PV, E	Page 15-116

1) Accesses to empty addresses: nBE

CAN_CLC

The clock control registers makes it possible to control (enable/disable) the module control clock f_{CLC}.

CAN_CLC
CAN Clock Control Register **(000_H)** **Reset Value: 0000 0003$_H$**



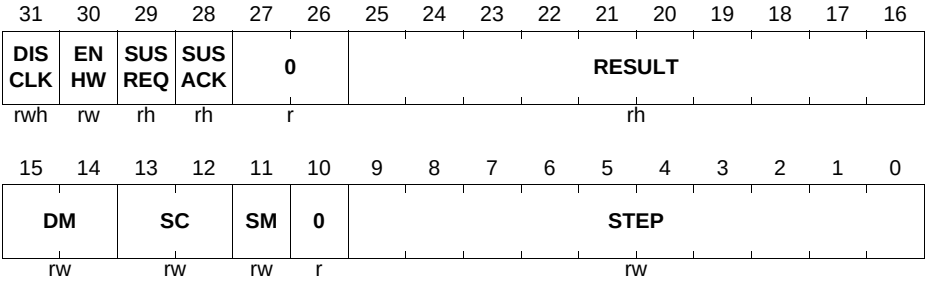
Field	Bits	Type	Description
DISR	0	rw	**Module Disable Request Bit** Used for enable/disable control of the module.
DISS	1	r	**Module Disable Status Bit** Bit indicates the current status of the module.
EDIS	3	rw	**Sleep Mode Enable Control** Used to control module's sleep mode.
SBWE	4	w	**Module Suspend Bit Write Enable for OCDS** Determines whether SPEN and FSOE are write-protected.
0	2, 5, [31:6]	r	**Reserved** Read as 0; should be written with 0.

Note: In disabled state, no registers of CAN module can be read or written except the CAN_CLC register.

CAN_FDR

The fractional divider register allows the programmer to control the clock rate of the module timer clock f_{CAN}.

CAN_FDR
CAN Fractional Divider Register ($00C_H$) Reset Value: $0000\ 0000_H$



Field	Bits	Type	Description
STEP	[9:0]	rw	**Step Value** Reload or addition value for RESULT.
SM	11	rw	**Suspend Mode** SM selects between granted or immediate Suspend Mode.
SC	[13:12]	rw	**Suspend Control** This bit field determines the behavior of the fractional divider in Suspend Mode.
DM	[15:14]	rw	**Divider Mode** This bit field selects normal divider mode, fractional divider mode, and off-state.
RESULT	[25:16]	rh	**Result Value** Bit field for the addition result.
SUSACK	28	rh	**Suspend Mode Acknowledge** Indicates state of $\overline{\text{SPNDACK}}$ signal.
SUSREQ	29	rh	**Suspend Mode Request** Indicates state of SPND signal.
ENHW	30	rw	**Enable Hardware Clock Control** Controls operation of ECEN input and DISCLK bit.

Field	Bits	Type	Description
DISCLK	31	rwh	**Disable Clock** Hardware controlled disable for f_{OUT} signal.
0	10, [27:26]	r	**Reserved** Read as 0; should be written with 0.

MultiCAN Module Register Address Map

The complete MultiCAN module register address map of **Figure 15-26** shows the general implementation-specific registers for clock control, module identification, and interrupt service request control and adds the absolute address information.

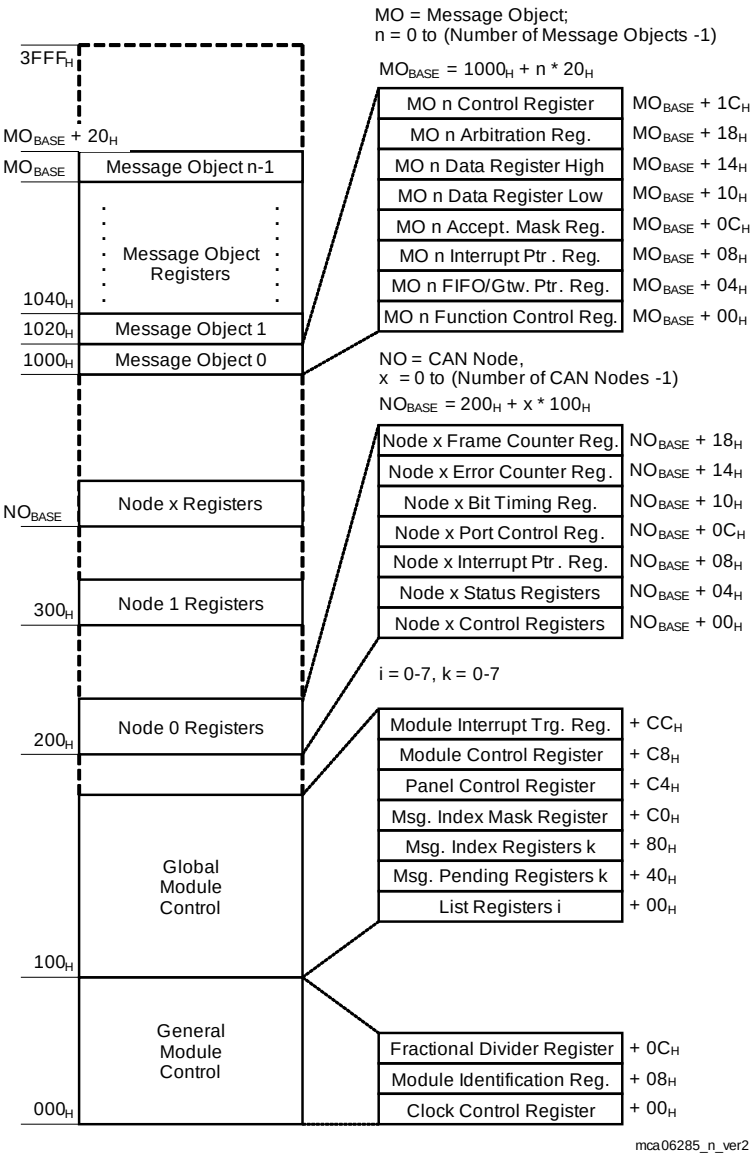


Figure 15-26 MultiCAN Module Register Map

15.8 Interconnects

This section describes CAN module interfaces with the clock control, port connections, and address decoding.

15.8.1 Interfaces of the MultiCAN Module

Figure 15-27 shows the XMC4[12]00 specific implementation details and interconnections of the MultiCAN module. The six I/O lines of the MultiCAN module (two I/O lines of each CAN node) are connected to I/O lines of Port 0,1,2,3,4 and 14. The MultiCAN module is also supplied by clock control, interrupt control, and address decoding logic. MultiCAN interrupts can be directed to the GPDMA controller and the CCU4 modules. CAN interrupts are able to trigger DMA transfers and CCU4 operations.

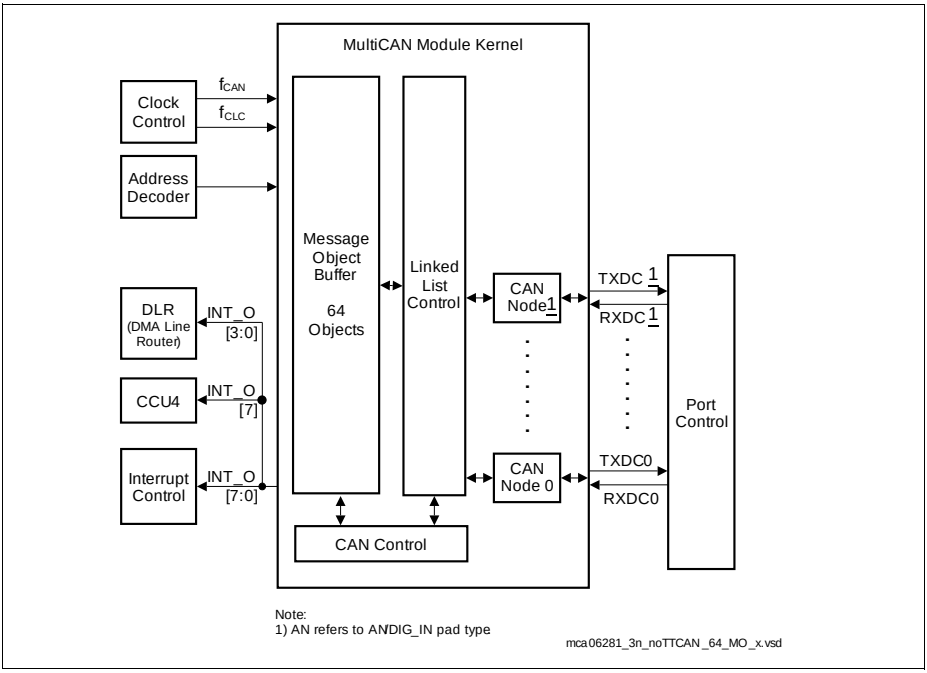


Figure 15-27 CAN module Implementation and Interconnections

15.8.2 Port and I/O Line Control

The interconnections between the MultiCAN module and the port I/O lines are controlled in the port logic. Additionally to the port input selection, the following port control operations must be executed:

- Input/output function selection (IOCR registers)
- Pad driver characteristics selection for the outputs (PDR registers)

15.8.2.1 Input/Output Function Selection in Ports

The port input/output control registers contain the bit fields that select the digital output and input driver characteristics such as pull-up/down devices, port direction (input/output), open-drain, and alternate output selections. The I/O lines for the MultiCAN module are controlled by the port input/output control registers Pn_IOCRy PCx defined in the GPIO chapter.

Additionally to the I/O control selection, as defined in **Table 15-14**, the selection of a CAN node's receive input line requires that bit field RXSEL in its node port control register NPCRx must be set.

The selected input signal (selected by bit field NPCRx.RXSEL) for each CAN node is made available by internal signal CANxINS (CAN node x input signal, with x = 0 - 1) as shown in **Figure 15-28**. The default setting after reset of a node's NPCRx.RXSEL bit field connect node x with RXDCx I/O line (x = 0-2).

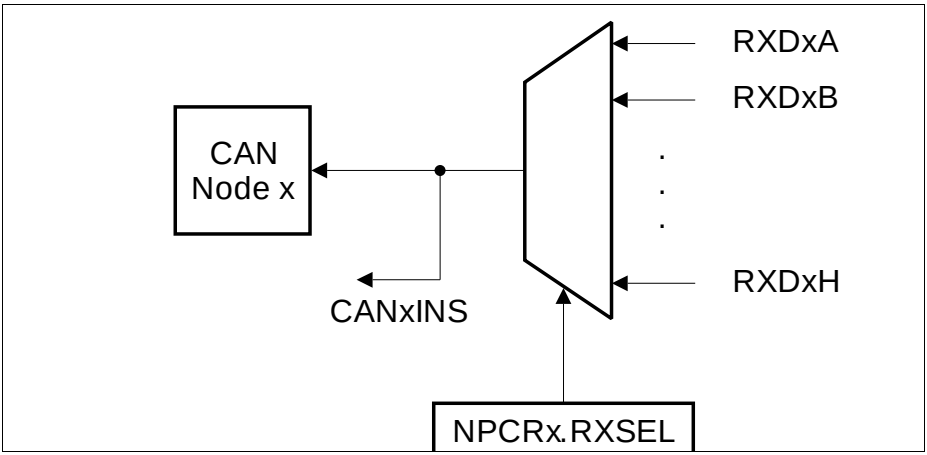


Figure 15-28 CAN Module Receive Input Selection

Table 15-14 shows how bits and bit fields must be programmed for the required I/O functionality of the CAN I/O lines.

Table 15-14 MultiCAN I/O Control Selection and Setup

Input/Output	I/O	Connected To	Description
Receive Inputs (Node 0)			
CAN.CAN0_RXDC0A	I	P1.5	CAN Receive Input
CAN.CAN0_RXDC0B	I	P14.3	CAN Receive Input
Receive Inputs (Node 1)			
CAN.CAN1_RXDC1A	I	P2.6	CAN Receive Input
CAN.CAN1_RXDC1D	I	P1.4	CAN Receive Input
CAN.CAN1_RXDC1F	I	CAN0INS	CAN Receive Input
Transmit Outputs (Node 0)			
CAN.CAN0_TXDC0	O	P0.0	CAN Transmit Output
CAN.CAN0_TXDC0	O	P1.4	CAN Transmit Output
CAN.CAN0_TXDC0	O	P3.2	CAN Transmit Output
CAN.CAN0_TXDC0	O	P2.0	CAN Transmit Output
Transmit Outputs (Node 1)			
CAN.CAN1_TXDC1	O	P2.7	CAN Transmit Output
CAN.CAN1_TXDC1	O	P1.5	CAN Transmit Output

15.8.2.2 MultiCAN Interrupt Output Connections

The interrupt outputs of the MultiCAN module are connected as shown in Table 15-15.

Table 15-15 CAN Interrupt Output Connections

Input/Output	I/O	Connected To	Description
System Related Outputs			
SR0	O	NVIC	Interrrupt Request
	O	DLR	DMA Request
SR1	O	CPU	CAN Interrrupt Output
	O	DLR	DMA Request
SR2	O	CPU	CAN Interrrupt Output
	O	DLR	DMA Request

Table 15-15 CAN Interrupt Output Connections (cont'd)

Input/Output	I/O	Connected To	Description
SR3	O	CPU	CAN Interrrupt Output
	O	DLR	DMA Request
SR4	O	NVIC	Interrrupt Request
SR5	O	NVIC	Interrrupt Request
SR6	O	NVIC	Interrrupt Request
SR7	O	NVIC	Interrrupt Request
	O	CCU4	CCU4 Trigger

15.8.2.3 Connections to USIC Inputs

The internal signal CAN1INS is connected to the USIC module, see **Table 15-16**.

Table 15-16 CAN-to-USIC Connections

Input/Output	I/O	Connected To	Description
System Related Outputs			
CAN.CAN1INS	O	U1C1_DX0E	CAN Receive Multiplexer Output

Analog Frontend Peripherals

16 Versatile Analog-to-Digital Converter (VADC)

The XMC4[12]00 provides a series of analog input channels connected to a cluster of Analog/Digital Converters using the Successive Approximation Register (SAR) principle to convert analog input values (voltages) to discrete digital values.
The number of analog input channels and ADCs depends on the chosen product type (please refer to **"Product-Specific Configuration" on Page 16-133**).

Table 16-1 Abbreviations used in ADC chapter

ADC	Analog to Digital Converter
DMA	Direct Memory Access (controller)
DNL	Differential Non-Linearity (error)
INL	Integral Non-Linearity (error)
LSB_n	Least Significant Bit: finest granularity of the analog value in digital format, represented by one least significant bit of the conversion result with n bits resolution (measurement range divided in 2^n equally distributed steps)
SCU	System Control Unit of the device
TUE	Total Unadjusted Error

16.1 Overview

Each converter of the ADC cluster can operate independent of the others, controlled by a dedicated set of registers and triggered by a dedicated group request source. The results of each channel can be stored in a dedicated channel-specific result register or in a group-specific result register.

A background request source can access all analog input channels that are not assigned to any group request source. These conversions are executed with low priority. The background request source can, therefore, be regarded as an additional background converter.

The Versatile Analog to Digital Converter module (VADC) of the XMC4[12]00 comprises a set of converter blocks that can be operated either independently or via a common request source that emulates a background converter. Each converter block is equipped with a dedicated input multiplexer and dedicated request sources, which together build separate groups.

This basic structure supports application-oriented programming and operating while still providing general access to all resources. The almost identical converter groups allow a flexible assignment of functions to channels.

The basic module clock f_{ADC} is connected to the system clock signal f_{PERIPH}.

Feature List

The following features describe the functionality of the ADC cluster:

- Nominal analog supply voltage 3.3 V
- Input voltage range from 0 V up to analog supply voltage
- Standard (V_{AREF}) and alternate (CH0) reference voltage source selectable for each channel to support ratiometric measurements and different signal scales
- Up to 2 independent converters with up to 8 analog input channels
- External analog multiplexer control, including adjusted sample time and scan support
- Conversion speed and sample time adjustable to adapt to sensors and reference
- Conversion time below 1 µs (depending on result width and sample time)
- Flexible source selection and arbitration
  - Programmable arbitrary conversion sequence (single or repeated)
  - Configurable auto scan conversion (single or repeated) on each converter
  - Configurable auto scan conversion (single or repeated) in the background (all converters)
  - Conversions triggered by software, timer events, or external events
  - Cancel-inject-restart mode for reduced conversion delay on priority channels
- Powerful result handling
  - Selectable result width of 8/10/12 bits
  - Fast Compare Mode
  - Independent result registers
  - Configurable limit checking against programmable border values
  - Data rate reduction through adding a selectable number of conversion results
  - FIR/IIR filter with selectable coefficients
- Flexible service request generation based on selectable events
- Built-in safety features
  - Broken wire detection with programmable default levels
  - Multiplexer test mode to verify signal path integrity
- Support of suspend and power saving modes

Note: Additional functions are available from the out of range comparator (see description in the SCU).

Table 16-2 VADC Applications

Use Case VADC	Application
Automatic scheduling of complex conversion sequences, including priorization of time-critical conversions	Motor control, Power conversion
Effective result handling for bursts of high-speed conversions	Highly dynamic input signals
Synchronous sampling of up to 4 input signals	Multi-phase current measurement

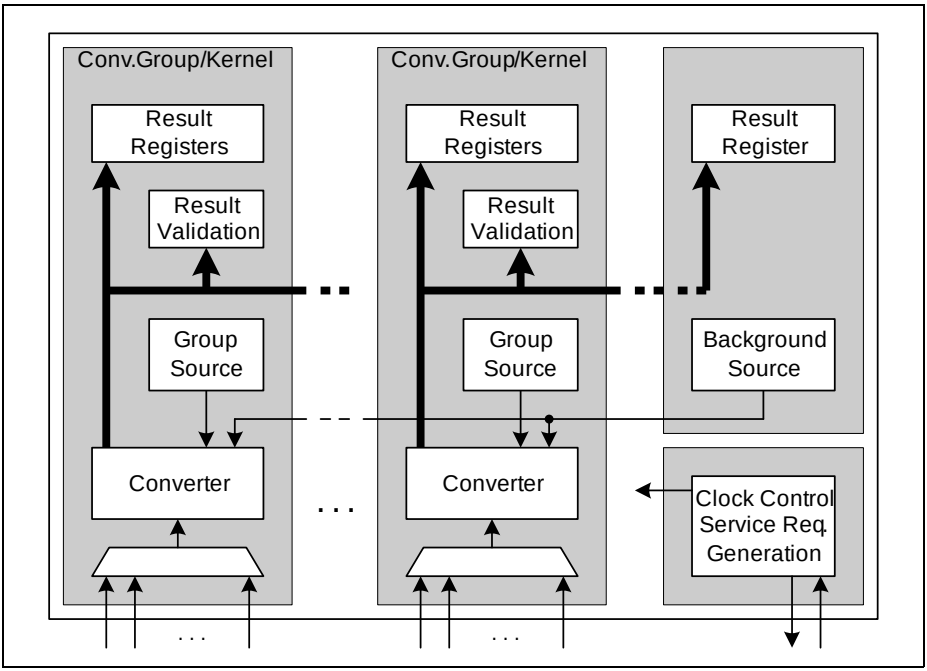


Figure 16-1 ADC Structure Overview

16.2 Introduction and Basic Structure

The Versatile Analog to Digital Converter module (VADC) of the XMC4[12]00 comprises a set of converter blocks that can be operated either independently or via a common request source that emulates a background converter. Each converter block is equipped with a dedicated input multiplexer and dedicated request sources, which together build separate groups.

This basic structure supports application-oriented programming and operating while still providing general access to all resources. The almost identical converter groups allow a flexible assignment of functions to channels.

A set of functional units can be configured according to the requirements of a given application. These units build a path from the input signals to the digital results.

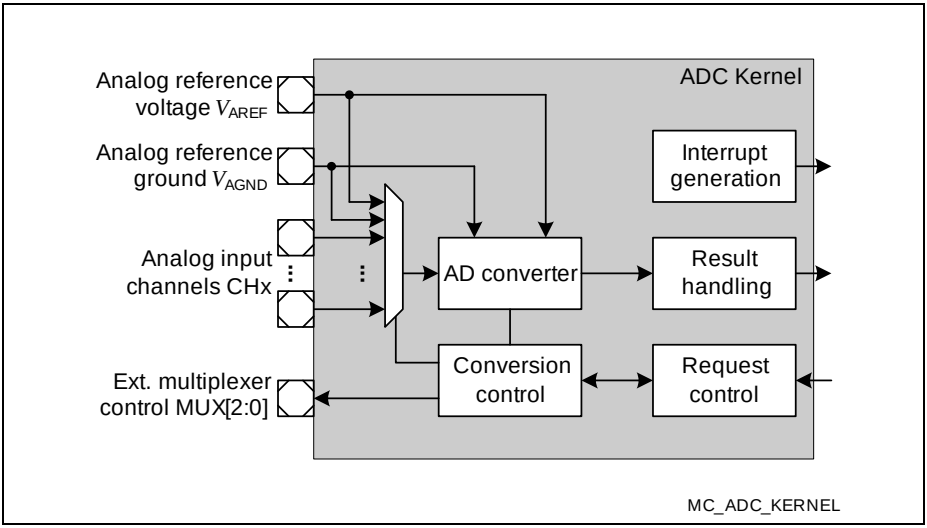


Figure 16-2 ADC Kernel Block Diagram

Conversion Modes and Request Sources

Analog/Digital conversions can be requested by several request sources (2 group request sources and the background request source) and can be executed in several conversion modes. The request sources can be enabled concurrently with configurable priorities.

- **Fixed Channel Conversion (single or continuous)**
  A specific channel source requests conversions of one selectable channel (once or repeatedly)
- **Auto Scan Conversion (single or continuous)**
  A channel scan source (request source 1 or 2) requests auto scan conversions of a configurable linear sequence of all available channels (once or repeatedly)
- **Channel Sequence Conversion (single or continuous)**
  A queued source (request source 0) requests a sequence of conversions of up to 8 arbitrarily selectable channels (once or repeatedly)

The conversion modes can be used concurrently by the available request sources, i.e. conversions in different modes can be enabled at the same time. Each source can be enabled separately and can be triggered by external events, such as edges of PWM or timer signals, or pin transitions.

Request Source Control

Because all request sources can be enabled at the same time, an arbiter resolves concurrent conversion requests from different sources. Each source can be triggered by external signals, by on-chip signals, or by software.

Requests with higher priority can either cancel a running lower-priority conversion (cancel-inject-repeat mode) or be converted immediately after the currently running conversion (wait-for-start mode). If the target result register has not been read, a conversion can be deferred (wait-for-read mode).

Certain channels can also be synchronized with other ADC kernels, so several signals can be converted in parallel.

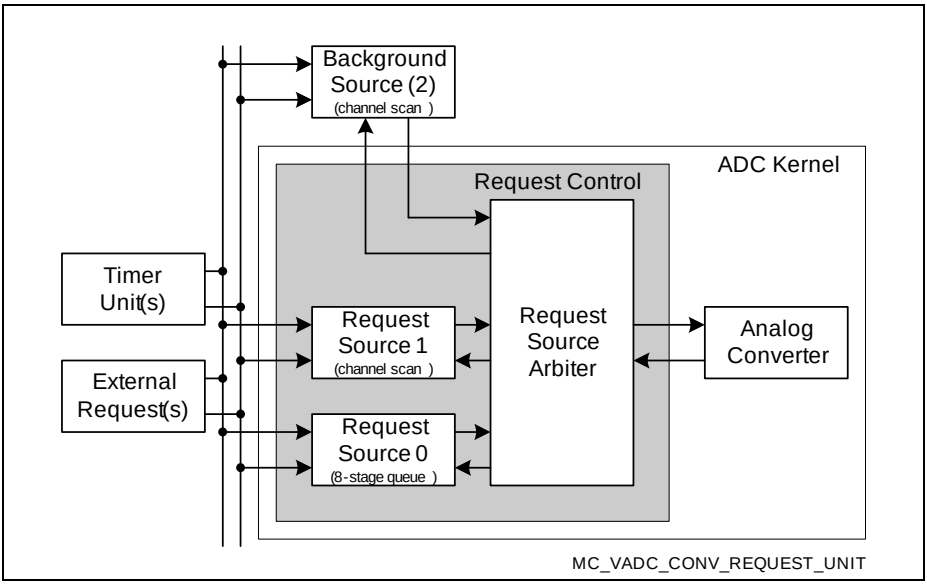


Figure 16-3 Conversion Request Unit

Input Channel Selection

The analog input multiplexer selects one of the available analog inputs (CH0 - CHx[1)]) to be converted. Three sources can select a linear sequence, an arbitrary sequence, or a specific channel. The priorities of these sources can be configured.

Additional external analog multiplexers can be controlled automatically, if more separate input channels are required than are built in.

*Note: Not all analog input channels are necessarily available in all packages, due to pin limitations. Please refer to the implementation description in **Section 16.14**.*

Conversion Control

Conversion parameters, such as sample phase duration, reference voltage, or result resolution can be configured for 4 input classes (2 group-specific classes, 2 global classes). Each channel can be individually assigned to one of these input classes.

The input channels can, thus, be adjusted to the type of sensor (or other analog sources) connected to the ADC.

1) The availablity of input channels depends on the package of the used product type. A summary can be found in **Section 16.14.2**.

This unit also controls the built-in multiplexer and external analog multiplexers, if selected.

Analog/Digital Converter

The selected input channel is converted to a digital value by first sampling the voltage on the selected input and then generating the selected number of result bits.

For 12-bit conversions, post-calibration is executed after converting the channel.

For broken wire detection (see **Section 16.10.1**), the converter network can be preloaded before sampling the selected input channel.

Result Handling

The conversion results of each analog input channel can be directed to one of 16 group-specific result registers and one global result register to be stored there. A result register can be used by a group of channels or by a single channel.

The wait-for-read mode avoids data loss due to result overwrite by blocking a conversion until the previous result has been read.

Data reduction (e.g. for digital anti-aliasing filtering) can automatically add up to 4 conversion results before issuing a service request.

Alternatively, an FIR or IIR filter can be enabled that preprocesses the conversion results before sending them to the result register.

Also, result registers can be concatenated to build FIFO structures that store a number of conversion results without overwriting previous data. This increases the allowed CPU latency for retrieving conversion data from the ADC.

Service Request Generation

Several ADC events can issue service requests to CPU or DMA:

- **Source events** indicate the completion of a conversion sequence in the corresponding request source. This event can be used to trigger the setup of a new sequence.
- **Channel events** indicate the completion of a conversion for a certain channel. This can be combined with limit checking, so interrupt are generated only if the result is within a defined range of values.
- **Result events** indicate the availability of new result data in the corresponding result register. If data reduction mode is active, events are generated only after a complete accumulation sequence.

Each event can be assigned to one of eight service request nodes. This allows grouping the requests according to the requirements of the application.

Safety Features

Safety-aware applications are supported with mechanisms that help to ensure the integrity of a signal path.

Broken-wire-detection (BWD) preloads the converter network with a selectable level before sampling the input channel. The result will then reflect the preload value if the input signal is no more connected. If buffer capacitors are used, a certain number of conversions may be required to reach the failure indication level.

Pull Down Diagnostics (PDD) connects an additional strong pull-down device to an input channel. A subsequent conversion can then confirm the expected modified signal level. This allows to check the proper connection of a signal source (sensor) to the multiplexer.

Multiplexer Diagnostics (MD) connects a weak pull-up or pull-down device to an input channel. A subsequent conversion can then confirm the expected modified signal level. This allows to check the proper operation of the multiplexer.

Note: These pull-up/pull-down devices are controlled via the port logic.

Converter Diagnostics (CD) connects an alternate signal to the converter. A subsequent conversion can then confirm the proper operation of the converter.

16.3 Configuration of General Functions

While many parameters can be selected individually for each channel, source, or group, some adjustments are valid for the whole ADC cluster:

- Clock control
- Kernel synchronization
- External multiplexer control
- Test functions

16.3.1 General Clocking Scheme and Control

The A/D Converters of the XMC4[12]00 are supplied with a global clock signal from the system f_{ADC}. Two clock signals are derived from this input and are distributed to all converters. The global configuration register defines common clock bases for all converters of the cluster. This ensures deterministic behavior of converters that shall operate in parallel.

The analog converter clock f_{ADCI} determines the performance of the converters and must be selected to comply with the specification given in the Data Sheet.

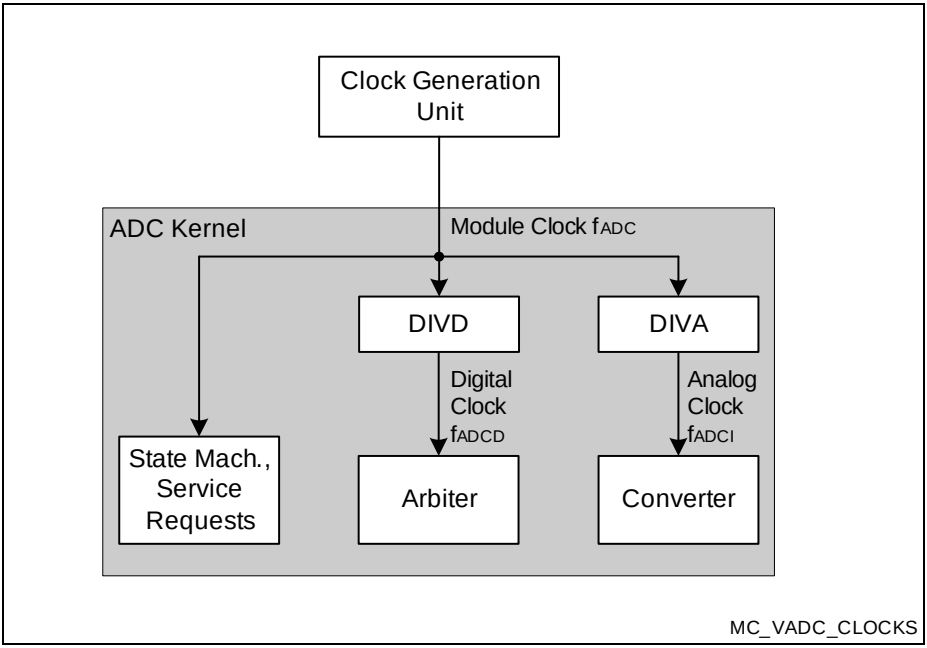


Figure 16-4 Clock Signal Summary

16.3.2 Priority Channel Assignment

Each channel of a group can be assigned to this group's request sources and is then regarded as a priority channel. An assigned priority channel can only be converted by its own group's request sources. A not assigned channel can also be converted by the background request source.

16.4 Module Activation and Power Saving

The analog converter of the ADC draws a permanent current during its operation. It can be deactivated between conversions to reduce the consumed overall energy.

The operating mode is determined by bitfield **GxARBCFG (x = 0 - 1)**.ANONS:

- ANONS = 11_B: **Normal Operation**
  The converter is active, conversions are started immediately.
  Requires no wakeup time.
- ANONS = 10_B or 01_B: **Reserved**
- ANONS = 00_B: **Converter switched Off** (default after reset)
  The converter is switched off. Furthermore, digital logic blocks are set to their initial state. If the arbiter is currently running, it completes the actual arbitration round and then stops.
  Before starting a conversion, select the active mode for ANONS.
  Requires the wakeup time (see below).

Wakeup Time from Analog Powerdown

When the converter is activated, it needs a certain wakeup time to settle before a conversion can be properly executed. This wakeup time can be established by waiting the required period before starting a conversion, or by adding it to the intended sample time.

The wakeup time is approximately 15 µs.
Exact numbers can be found in the respective Data Sheets.

Note: The wakeup time is also required after initially enabling the converter.

Calibration

Calibration automatically compensates deviations caused by process, temperature, and voltage variations. This ensures precise results throughout the operation time.

An initial start-up calibration is required once after a reset for all calibrated converters and is triggered globally. All calibrated converters must be enabled (ANONS = 11_B) before initiating the start-up calibration. Conversions may be started after the initial calibration sequence. This is indicated by bit CAL = 0_B.

After that, postcalibration cycles will compensate the effects of drifting parameters.

16.5 Conversion Request Generation

The conversion request unit of a group autonomously handles the generation of conversion requests. Three request sources (2 group-specific sources and the background source) can generate requests for the conversion of an analog channel. The arbiter resolves concurrent requests and selects the channel to be converted next.

Upon a trigger event, the request source requests the conversion of a certain analog input channel or a sequence of channels.

- **Software triggers**
  directly activate the respective request source.
- **External triggers**
  synchronize the request source activation with external events, such as a trigger pulse from a timer generating a PWM signal or from a port pin.

Application software selects the trigger, the channel(s) to be converted, and the request source priority. A request source can also be activated directly by software without requiring an external trigger.

The arbiter regularly scans the request sources for pending conversion requests and selects the conversion request with the highest priority. This conversion request is then forwarded to the converter to start the conversion of the requested channel.

Each request source can operate in single-shot or in continuous mode:

- **In single-shot mode**,
  the programmed conversion (sequence) is requested once after being triggered. A subsequent conversion (sequence) must be triggered again.
- **In continuous mode**,
  the programmed conversion (sequence) is automatically requested repeatedly after being triggered once.

For each request source, external triggers are generated from one of 16 selectable trigger inputs (REQTRx[P:A]) and from one of 16 selectable gating inputs (REQGTx[P:A]). The available trigger signals for the XMC4[12]00 are listed in **Section 16.14.3**.

*Note: **Figure 16-3 “Conversion Request Unit” on Page 16-6** summarizes the request sources.*

Two types of requests sources are available:

- **A queued source** can issue conversion requests for an arbitrary sequence of input channels. The channel numbers for this sequence can be freely programmed[1]. This supports application-specific conversion sequences that cannot be covered by a channel scan source. Also, multiple conversions of the same channel within a sequence are supported.
  A queued source converts a series of input channels permanently or on a regular time base. For example, if programmed with medium priority, some input channels can be converted upon a specified event (e.g. synchronized to a PWM). Conversions of lower priority sources are suspended in the meantime.
  Request source 0 is a group-specific 8-stage queued source.
- **A channel scan source** can issue conversion requests for a coherent sequence of input channels. This sequence begins with the highest enabled channel number and continues towards lower channel numbers. All available channels[1] can be enabled for the scan sequence. Each channel is converted once per sequence.
  A scan source converts a series of input channels permanently or on a regular time base. For example, if programmed with low priority, some input channels can be scanned in a background task to update information that is not time-critical.
  – Request source 1 is a group-specific channel scan source.
  – Request source 2 is a global channel scan source (background source).
  The background source can request all channels of all groups.

1) The availablity of input channels depends on the package of the used product type. A summary can be found in **Section 16.14.2**.
The background source can only request non-priority channels, i.e. channels that are not selected in registers GxCHASS. Priority channels are reserved for the group-specific request sources 0 and 1.

16.5.1 Queued Request Source Handling

A queued request source supports short conversion sequences (up to 8) of arbitrary channels (contrary to a scan request source with a fixed conversion order for the enabled channels). The programmed sequence is stored in a queue buffer (based on a FIFO mechanism). The requested channel numbers are entered via the queue input, while queue stage 0 defines the channel to be converted next.

A conversion request is only issued to the request source arbiter if a valid entry is stored in queue stage 0.

If the arbiter aborts a conversion triggered by a queued request source due to higher priority requests, the corresponding conversion parameters are automatically saved in the backup stage. This ensures that an aborted conversion is not lost but takes part in the next arbitration round (before stage 0).

The trigger and gating unit generates trigger events from the selected external (outside the ADC) trigger and gating signals. For example, a timer unit can issue a request signal to synchronize conversions to PWM events.

Trigger events start a queued sequence and can be generated either via software or via the selected hardware triggers. The occurrence of a trigger event is indicated by bit QSRx.EV. This flag is cleared when the corresponding conversion is started or by writing to bit QMRx.CEV.

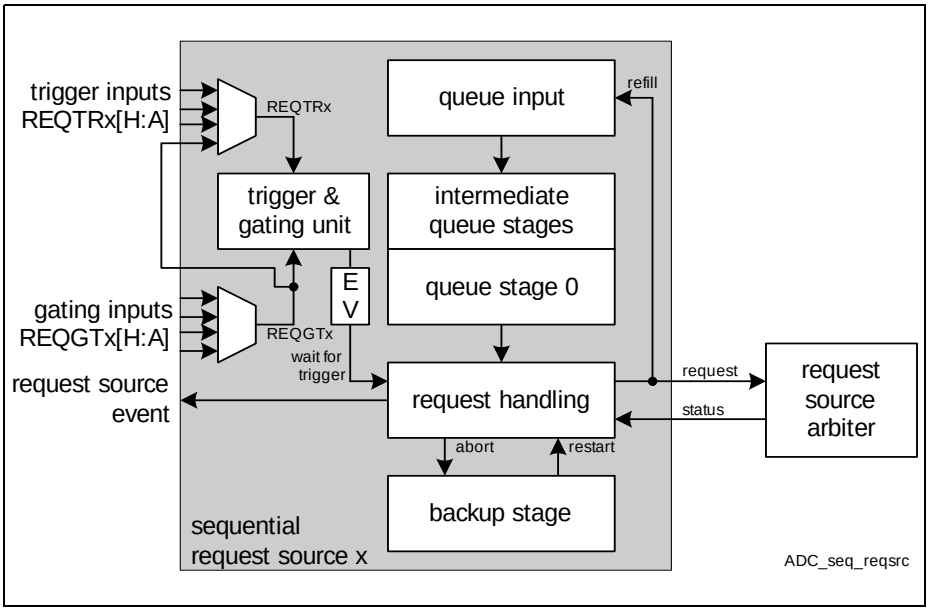


Figure 16-5 Queued Request Source

A sequence is defined by entering conversion requests into the queue input register (**GxQINR0 (x = 0 - 1)**). Each entry selects the channel to be converted and can enable an external trigger, generation of an interrupt, and an automatic refill (i.e. copy this entry to the top of the queue after conversion). The entries are stored in the queue buffer stages.

The content of stage 0 (**GxQ0R0 (x = 0 - 1)**) selects the channel to be converted next. When the requested conversion is started, the contents of this queue stage is invalidated and copied to the backup stage. Then the next queue entry can be handled (if available).

Note: The contents of the queue stages cannot be modified directly, but only by writing to the queue input or by flushing the queue.
*The current status of the queue is shown in register **GxQSR0 (x = 0 - 1)**.*
If all queue entries have automatic refill selected, the defined conversion sequence can be repeated without re-programming.

Properties of the Queued Request Source

Queued request source 0 provides 8 buffer stages and can handle sequences of up to 8 input channel entries. It supports short application-specific conversion sequences, especially for timing-critical sequences containing also multiple conversions of the same channel.

Queued Source Operation

Configure the queued request source by executing the following actions:

- Define the sequence by writing the entries to the queue input **GxQINR0 (x = 0 - 1)**. Initialize the complete sequence before enabling the request source, because with enabled refill feature, software writes to QINRx are not allowed.
- If hardware trigger or gating is desired, select the appropriate trigger and gating inputs and the proper transitions by programming **GxQCTRL0 (x = 0 - 1)**.
  Enable the trigger and select the gating mode by programming bitfield ENGT in register **GxQMR0 (x = 0 - 1)**.[1)]
- Enable the corresponding arbitration slot (0) to accept conversion requests from the queued source (see register **GxARBPR (x = 0 - 1)**).

Start a queued sequence by generating a trigger event:

- If a hardware trigger is selected and enabled, generate the configured transition at the selected input signal, e.g. from a timer or an input pin.
- Generate a software trigger event by setting GxQMR0.TREV = 1.

1) If PDOUT signals from the ERU are used, initialize the ERU accordingly before enabling the gate inputs to avoid un expected signal transitions.

- Write a new entry to the queue input of an empty queue. This leads to a (new) valid queue entry that is forwarded to queue stage 0 and starts a conversion request (if enabled by GxQMR0.ENGT and without waiting for an external trigger).

Note: If the refill mechanism is activated, a processed entry is automatically reloaded into the queue. This permanently repeats the respective sequence (autoscan). In this case, do not write to the queue input while the queued source is running. Write operations to a completely filled queue are ignored.

Stop or abort an ongoing queued sequence by executing the following actions:

- If external gating is enabled, switch the gating signal to the defined inactive level. This does not modify the queue entries, but only prevents issuing conversion requests to the arbiter.
- Disable the corresponding arbitration slot (0) in the arbiter. This does not modify the queue entries, but only prevents the arbiter from accepting requests from the request handling block.
- Disable the queued source by clearing bitfield ENGT = 00_B.
  - Invalidate the next pending queue entry by setting bit GxQMR0.CLRV = 1. If the backup stage contains a valid entry, this one is invalidated, otherwise stage 0 is invalidated.
  - Remove all entries from the queue by setting bit GxQMR0.FLUSH = 1.

Queue Request Source Events and Service Requests

A request source event of a queued source occurs when a conversion is finished. A source event service request can be generated based on a request source event according to the structure shown in **Figure 16-6**. If a request source event is detected, it sets the corresponding indication flag in register **GxSEFLAG (x = 0 - 1)**. These flags can also be set by writing a 1 to the corresponding bit position, whereas writing 0 has no effect. The indication flags can be cleared by SW by writing a 1 to the corresponding bit position in register **GxSEFCLR (x = 0 - 1)**.
The interrupt enable bit is taken from stage 0 for a normal sequential conversion, or from the backup stage for a repeated conversion after an abort.

The service request output line SRx that is selected by the request source event interrupt node pointer bitfields in register **GxSEVNP (x = 0 - 1)** becomes activated each time the related request source event is detected (and enabled by GxQ0R0.ENSI, or GxQBUR0.ENSI respectively) or the related bit position in register **GxSEFLAG (x = 0 - 1)** is written with a 1 (this write action simulates a request source event).

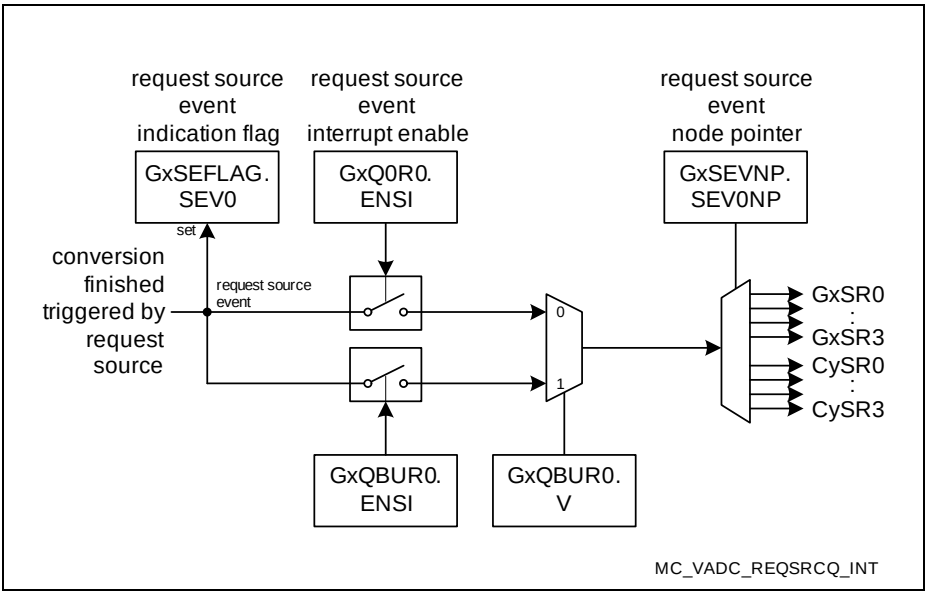


Figure 16-6 Interrupt Generation of a Queued Request Source

16.5.2 Channel Scan Request Source Handling

The VADC provides two types of channel scan sources:

- Source 1: Group scan source
  This scan source can request all channels of the corresponding group.
- Source 2: Background scan source
  This scan source can request all channels of all groups.
  Priority channels selected in registers **GxCHASS (x = 0 - 1)** cannot take part in background conversion sequences.

Both sources operate in the same way and provide the same register interface. The background source provides more request/pending bits because it can request all channels of all groups.

Each analog input channel can be included in or excluded from the scan sequence by setting or clearing the corresponding channel select bit in register **GxASSEL (x = 0 - 1)** or **BRSSELx (x = 0 - 1)**. The programmed register value remains unchanged by an ongoing scan sequence. The scan sequence starts with the highest enabled channel number and continues towards lower channel numbers.

Upon a load event, the request pattern is transferred to the pending bits in register **GxASPND (x = 0 - 1)** or **BRSPNDx (x = 0 - 1)**. The pending conversion requests indicate

which input channels are to be converted in an ongoing scan sequence. Each conversion start that was triggered by the scan request source, automatically clears the corresponding pending bit. If the last conversion triggered by the scan source is finished and all pending bits are cleared, the current scan sequence is considered finished and a request source event is generated.

A conversion request is only issued to the request source arbiter if at least one pending bit is set.

If the arbiter aborts a conversion triggered by the scan request source due to higher priority requests, the corresponding pending bit is automatically set. This ensures that an aborted conversion is not lost but takes part in the next arbitration round.

The trigger and gating unit generates load events from the selected external (outside the ADC) trigger and gating signals. For example, a timer unit can issue a request signal to synchronize conversions to PWM events.

Load events start a scan sequence and can be generated either via software or via the selected hardware triggers. The request source event can also generate an automatic load event, so the programmed sequence is automatically repeated.

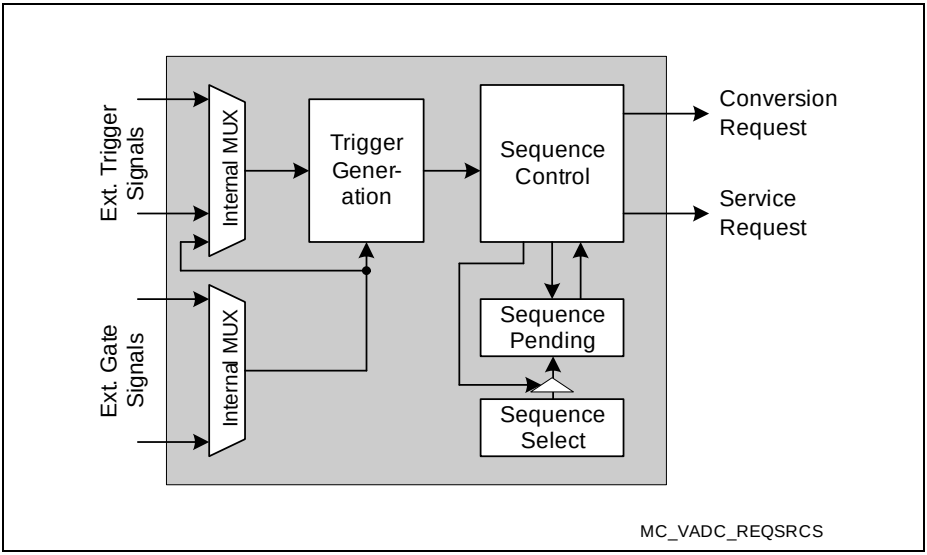


Figure 16-7 Scan Request Source

Scan Source Operation

Configure the scan request source by executing the following actions:

- Select the input channels for the sequence by programming **GxASSEL (x = 0 - 1)** or **BRSSELx (x = 0 - 1)**
- If hardware trigger or gating is desired, select the appropriate trigger and gating inputs and the proper signal transitions by programming **GxASCTRL (x = 0 - 1)** or **BRSCTRL**. Enable the trigger and select the gating mode by programming **GxASMR (x = 0 - 1)** or **BRSMR**.[1)]
- Define the load event operation (handling of pending bits, autoscan mode) by programming **GxASMR (x = 0 - 1)** or **BRSMR**.
  A load event with bit LDM = 0 copies the content of **GxASSEL (x = 0 - 1)** or **BRSSELx (x = 0 - 1)** to **GxASPND (x = 0 - 1)** or **BRSPNDx (x = 0 - 1)** (overwrite mode). This starts a new scan sequence and aborts any pending conversions from a previous scan sequence.
  A load event with bit LDM = 1 OR-combines the content of **GxASSEL (x = 0 - 1)** or **BRSSELx (x = 0 - 1)** to **GxASPND (x = 0 - 1)** or **BRSPNDx (x = 0 - 1)** (combine mode). This starts a scan sequence that includes pending conversions from a prvious scan sequence.
- Enable the corresponding arbitration slot (1) to accept conversion requests from the channel scan source (see register **GxARBPR (x = 0 - 1)**).

Start a channel scan sequence by generating a load event:

- If a hardware trigger is selected and enabled, generate the configured transition at the selected input signal, e.g. from a timer or an input pin.
- Generate a software load event by setting LDEV = 1 (**GxASMR (x = 0 - 1)** or **BRSMR**).
- Generate a load event by writing the scan pattern directly to the pending bits in **GxASPND (x = 0 - 1)** or **BRSPNDx (x = 0 - 1)**. The pattern is copied to **GxASSEL (x = 0 - 1)** or **BRSSELx (x = 0 - 1)** and a load event is generated automatically.
  In this case, a scan sequence can be defined and started with a single data write action, e.g. under PEC control (provided that the pattern fits into one register).

Note: If autoscan is enabled, a load event is generated automatically each time a request source event occurs when the scan sequence has finished. This permanently repeats the defined scan sequence (autoscan).

Stop or abort an ongoing scan sequence by executing the following actions:

- If external gating is enabled, switch the gating signal to the defined inactive level. This does not modify the conversion pending bits, but only prevents issuing conversion requests to the arbiter.

1) If PDOUT signals from the ERU are used, initialize the ERU accordingly before enabling the gate inputs to avoid un expected signal transitions.

- Disable the corresponding arbitration slot (1 or 2) in the arbiter. This does not modify the contents of the conversion pending bits, but only prevents the arbiter from accepting requests from the request handling block.
- Disable the channel scan source by clearing bitfield ENGT = 00_B. Clear the pending request bits by setting bit CLRPND = 1 (**GxASMR (x = 0 - 1)** or **BRSMR**).

Scan Request Source Events and Service Requests

A request source event of a scan source occurs if the last conversion of a scan sequence is finished (all pending bits = 0). A request source event interrupt can be generated based on a request source event. If a request source event is detected, it sets the corresponding indication flag in register **GxSEFLAG (x = 0 - 1)**. These flags can also be set by writing a 1 to the corresponding bit position, whereas writing 0 has no effect.

The service request output SRx that is selected by the request source event interrupt node pointer bitfields in register **GxSEVNP (x = 0 - 1)** becomes activated each time the related request source event is detected (and enabled by ENSI) or the related bit position in register **GxSEFLAG (x = 0 - 1)** is written with a 1 (this write action simulates a request source event).

The indication flags can be cleared by SW by writing a 1 to the corresponding bit position in register **GxSEFCLR (x = 0 - 1)**.[1)]

1) Please refer to **"Service Request Generation" on Page 16-56**.

16.6 Request Source Arbitration

The request source arbiter regularly polls the request sources, one after the other, for pending conversion requests. Each request source is assigned to a certain time slot within an arbitration round, called arbitration slot. The duration of an arbitration slot is user-configurable via register **GLOBCFG**.

The priority of each request source is user-configurable via register **GxARBPR (x = 0 - 1)**, so the arbiter can select the next channel to be converted, in the case of concurrent requests from multiple sources, according to the application requirements.

An unused arbitration slot is considered empty and does not take part in the arbitration. After reset, all slots are disabled and must be enabled (register **GxARBPR (x = 0 - 1)**) to take part in the arbitration process.

Figure 16-8 summarizes the arbitration sequence. An arbitration round consists of one arbitration slot for each available request source. The synchronization source is always evaluated in the last slot and has a higher priority than all other sources. At the end of each arbitration round, the arbiter has determined the highest priority conversion request.

If a conversion is started in an arbitration round, this arbitration round does not deliver an arbitration winner. In the XMC4[12]00, the following request sources are available:

- Arbitration slot 0: **Group Queued source**, 8-stage sequences in arbitrary order
- Arbitration slot 1: **Group Scan source**, sequences in defined order within group
- Arbitration slot 2: **Background Scan source**, sequences in defined order, all groups
- Last arbitration slot: **Synchronization source**, synchronized conversion requests from another ADC kernel (always handled with the highest priority in a synchronization slave kernel).

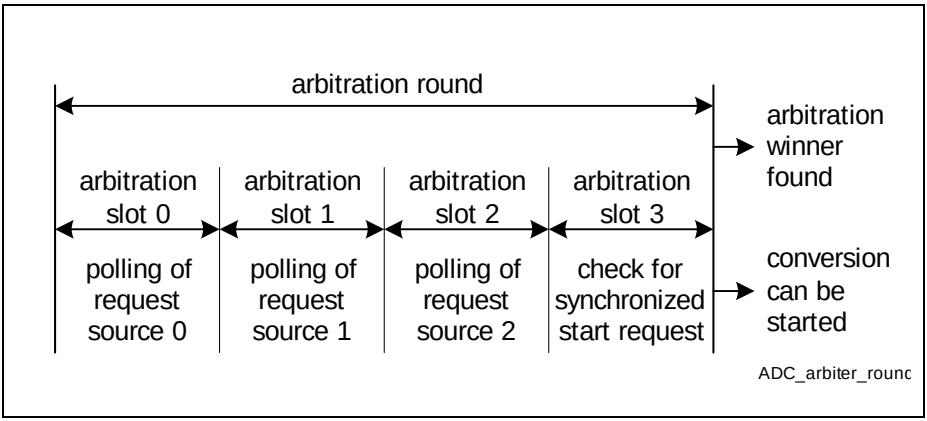


Figure 16-8 Arbitration Round with 4 Arbitration Slots

16.6.1 Arbiter Operation and Configuration

The timing of the arbiter (i.e. of an arbitration round) is determined by the number of arbitration slots within an arbitration round and by the duration of an arbitration slot.

An arbitration round consist of 4…20 arbitration slots (defined by bitfield **GxARBCFG (x = 0 - 1)**.ARBRND). 4 slots are sufficient for the XMC4[12]00, more can be programmed to obtain the same arbiter timing for different products.

The duration of an arbitration slot is configurable t_{Slot} = (DIVD+1) / f_{ADC}.
The duration of an arbitration round, therefore, is t_{ARB} = 4 × t_{Slot}.

The period of the arbitration round introduces a timing granularity to detect an incoming conversion request signal and the earliest point to start the related conversion. This granularity can introduce a jitter of maximum one arbitration round. The jitter can be reduced by minimizing the period of an arbitration round.

To achieve a reproducible reaction time (constant delay without jitter) between the trigger event of a conversion request (e.g. by a timer unit or due to an external event) and the start of the related conversion, mainly the following two options exist. For both options, the converter has to be idle and other conversion requests must not be pending for at least one arbiter round before the trigger event occurs:

- If bit **GxARBCFG (x = 0 - 1)**.ARBM = 0, the **arbiter runs permanently**. In this mode, synchronized conversions of more than one ADC kernel are possible.[1)]
  The trigger for a conversion request has to be generated synchronously to the arbiter timing. Incoming triggers should have exactly n-times the granularity of the arbiter (n = 1, 2, 3,...). In order to allow some flexibility, the duration of an arbitration slot can be programmed in cycles of f_{ADC}.
- If bit **GxARBCFG (x = 0 - 1)**.ARBM = 1, the **arbiter stops after an arbitration round** when no conversion request have been found pending any more. The arbiter is started again if at least one enabled request source indicates a pending conversion request. The trigger for a conversion request does not need to be synchronous to the arbiter timing.
  In this mode, parallel conversions are not possible for synchronization slave kernels.

Each request source has a configurable priority, so the arbiter can resolve concurrent conversion requests from different sources. The request with the highest priority is selected for conversion. These priorities can be adapted to the requirements of a given application (see register **GxARBPR (x = 0 - 1)**).

The **Conversion Start Mode** determines the handling of the conversion request that has won the arbitration.

1) For more information, please refer to **"Synchronization of Conversions" on Page 16-47**.

16.6.2 Conversion Start Mode

When the arbiter has selected the request to be converted next, the handling of this channel depends on the current activity of the converter:

- Converter is currently idle: the conversion of the arbitration winner is started immediately.
- Current conversion has same or higher priority: the current conversion is completed, the conversion of the arbitration winner is started after that.
- Current conversion has lower priority: the action is user-configurable:
  - **Wait-for-start mode**: the current conversion is completed, the conversion of the arbitration winner is started after that. This mode provides maximum throughput, but can produce a jitter for the higher priority conversion.
    Example in **Figure 16-9**:
    Conversion A is requested (t1) and started (t2). Conversion B is then requested (t3), but started only after completion of conversion A (t4).
  - **Cancel-inject-repeat mode**: the current conversion is aborted, the conversion of the arbitration winner is started after the abortion (3 f_{ADC} cycles).
    The aborted conversion request is restored in the corresponding request source and takes part again in the next arbitration round. This mode provides minimum jitter for the higher priority conversions, but reduces the overall throughput.
    Example in **Figure 16-9**:
    Conversion A is requested (t6) and started (t7). Conversion B is then requested (t8) and started (t9), while conversion A is aborted but requested again. When conversion B is complete (t10), conversion A is restarted.
    Exception: If both requests target the same result register with wait-for-read mode active (see **Section 16.8.3**), the current conversion cannot be aborted.

Note: A cancelled conversion can be repeated automatically in each case, or it can be discarded if it was cancelled. This is selected for each source by bit RPTDIS in the corresponding source's mode register.

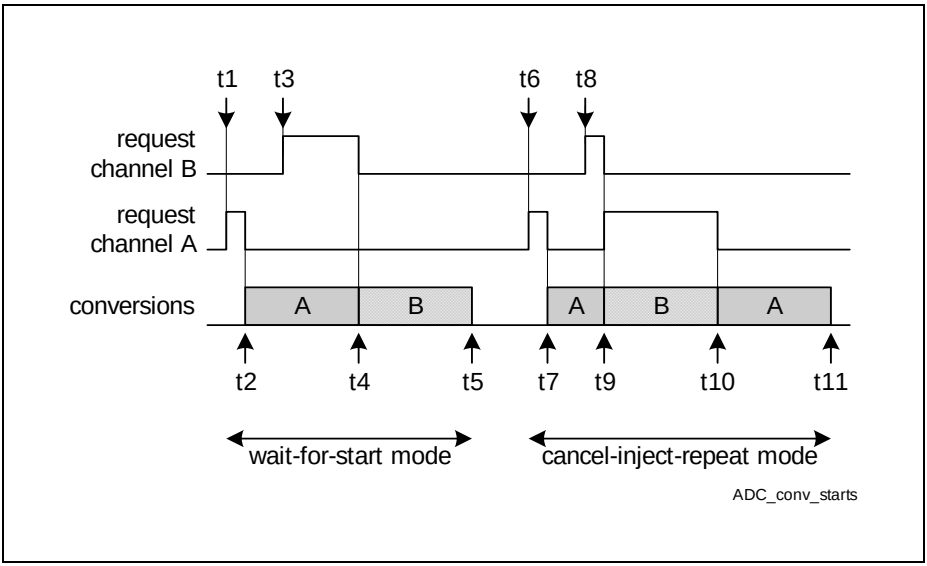


Figure 16-9 Conversion Start Modes

The conversion start mode can be individually programmed for each request source by bits in register **GxARBPR (x = 0 - 1)** and is applied to all channels requested by the source. In this example, channel A is issued by a request source with a lower priority than the request source requesting the conversion of channel B.

16.7 Analog Input Channel Configuration

For each analog input channel a number of parameters can be configured that control the conversion of this channel. The channel control registers define the following parameters:

- **Channel Parameters**: The sample time for this channel and the data width of the result are defined via input classes. Each channel can select one of two classes of its own group or one of two global classes.
- **Reference selection**: an alternate reference voltage can be selected for most channels (exceptions are marked in **Section 16.14.2**)
- **Result target**: The conversion result values are stored either in a group-specific result register or in the global result register. The group-specific result registers are selected in several ways:
  - channel-specific, selected by bitfield RESREG in register **G0CHCTRy (y = 0 - 7)** etc., with bitfield SRCRESREG = 0000_B
  - source-specific, selected by bitfield SRCRESREG in register **GxQCTRL0 (x = 0 - 1)**, **GxASCTRL (x = 0 - 1)** or **BRSCTRL** with bitfield SRCRESREG ≠ 0000_B
- **Result position**: The result values can be stored left-aligned or right-aligned. The exact position depends also on the configured result width and on the data accumulation mode.
  See also **Figure 16-16 “Result Storage Options” on Page 16-36**.
- **Compare with Standard Conversions (Limit Checking)**: Channel events can be generated whenever a new result value becomes available. Channel event generation can be restricted to values that lie inside or outside a user-configurable band.
  In Fast Compare Mode, channel events can be generated depending on the transitions of the (1-bit) result.
- **Broken Wire Detection**: This safety feature can detect a missing connection to an analog signal source (sensor).
- **Synchronization of Conversions**: Synchronized conversions are executed at the same time on several converters.

The **Alias Feature** redirects conversion requests for channels CH0 and/or CH1 to other channels.

16.7.1 Channel Parameters

Each analog input channel is configured by its associated channel control register.

*Note: For the safety feature “Broken Wire Detection”, refer to **Section 16.10.1**.*

The following features can be defined for each channel:

- The conversion class defines the result width and the sample time
- Generation of channel events and the result value band, if used
- Target of the result defining the target register and the position within the register

The group-specific input class registers define the sample time and data conversion mode for each channel of the respective group that selects them via bitfield ICLSEL in its channel control register GxCHCTRx.

The global input class registers define the sample time and data conversion mode for each channel of any group that selects them via bitfield ICLSEL in its channel control register GxCHCTRx.

16.7.2 Conversion Timing

The total time required for a conversion depends on several user-definable factors:

- The ADC conversion clock frequency, where $f_{ADCI} = f_{ADC} / (DIVA+1)$[1)]
- The selected sample time, where $t_S = (2 + STC) \times t_{ADCI}$ (STC = additional sample time, see also **Table 16-9**)
- The selected operating mode (normal conversion / fast compare mode)
- The result width N (8/10/12 bits) for normal conversions
- The post-calibration time PC, if selected (PC = 2, otherwise 0)
- The selected duration of the MSB conversion (DM = 0 or 1)
- Synchronization steps done at module clock speed

The conversion time is the sum of sample time, conversion steps, and synchronization. It can be computed with the following formulas:

Standard conversions: $t_{CN} = (2 + STC + N + DM + PC) \times t_{ADCI} + 2 \times t_{ADC}$

Fast compare mode: $t_{CN} = (2 + STC + 2) \times t_{ADCI} + 2 \times t_{ADC}$

The frequency at which conversions are triggered also depends on several configurable factors:

- The selected conversion time, according to the input class definitions. For conversions using an external multiplexer, also the extended sample times count.
- Delays induced by cancelled conversions that must be repeated.
- Delays due to equidistant sampling of other channels.
- The configured arbitration cycle time.
- The frequency of external trigger signals, if enabled.

Timing Examples

System assumptions:
f_{ADC} = 120 MHz i.e. t_{ADC} = 8.3 ns, DIVA = 3, f_{ADCI} = 30 MHz i.e. t_{ADCI} = 33.3 ns

According to the given formulas the following minimum conversion times can be achieved:

12-bit calibrated conversion:
$t_{CN12C} = (2 + 12 + 2) \times t_{ADCI} + 2 \times t_{ADC} = 16 \times 33.3\ ns + 2 \times 8.3\ ns = 550\ ns$

10-bit uncalibrated conversion:
$t_{CN10} = (2 + 10) \times t_{ADCI} + 2 \times t_{ADC} = 12 \times 33.3\ ns + 2 \times 8.3\ ns = 417\ ns$

Fast comparison:
$t_{FCM} = (2 + 2) \times t_{ADCI} + 2 \times t_{ADC} = 4 \times 33.3\ ns + 2 \times 8.3\ ns = 150\ ns$

1) The minimum prescaler factor for calibrated converters is 2.

16.7.3 Alias Feature

The Alias Feature redirects conversion requests for channels CH0 and/or CH1 to other channel numbers. This feature can be used to trigger conversions of the same input channel by independent events and to store the conversion results in different result registers.

- The same signal can be measured twice without the need to read out the conversion result to avoid data loss. This allows triggering both conversions quickly one after the other and being independent from CPU/DMA service request latency.
- The sensor signal is connected to only one analog input (instead of two analog inputs). This saves input pins in low-cost applications and only the leakage of one input has to be considered in the error calculation.
- Even if the analog input CH0 is used as alternative reference (see **Figure 16-10**), the internal trigger and data handling features for channel CH0 can be used.
- The channel settings for both conversions can be different (boundary values, service requests, etc.).

In typical low-cost AC-drive applications, only one common current sensor is used to determine the phase currents. Depending on the applied PWM pattern, the measured value has different meanings and the sample points have to be precisely located in the PWM period. **Figure 16-10** shows an example where the sensor signal is connected to one input channel (CHx) but two conversions are triggered for two different channels (CHx and CH0). With the alias feature, a conversion request for CH0 leads to a conversion of the analog input CHx instead of CH0, but taking into account the settings for CH0. Although the same analog input (CHx) has been measured, the conversion results can be stored and read out from the result registers RESx (conversion triggered for CHx) and RESy (conversion triggered for CH0). Additionally, different interrupts or limit boundaries can be selected, enabled or disabled.

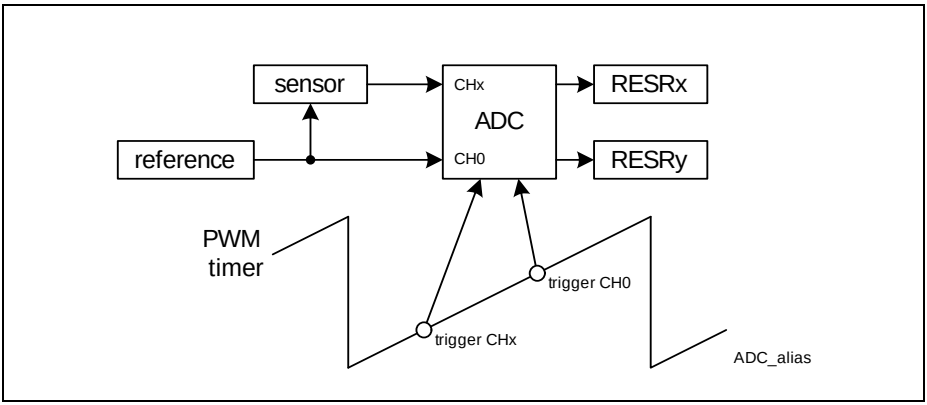


Figure 16-10 Alias Feature

16.7.4 Conversion Modes

A conversion can be executed in several ways. The conversion mode is selected according to the requested resolution of the digital result and according to the acceptable conversion time (**Section 16.7.2**).
Use bitfield CMS/CME in register **GxICLASS0 (x = 0 - 1)** etc. to select a mode.

Standard Conversions

A standard conversion returns a result value with a predefined resolution. 8-bit, 10-bit, and 12-bit resolution can be selected.

These result values can be accumulated, filtered, or used for digital limit checking and determination of extrema.

Note: The calibrated converters can operate with and without post-calibration.

Fast Compare Mode

In Fast Compare Mode, the selected input voltage is directly compared with a digital value that is stored in the corresponding result register. This compare operation returns a binary result indicating if the compared input voltage is above or below the given reference value. This result is generated quickly and thus supports monitoring of boundary values.

Fast Compare Mode uses a 10-bit compare value stored left-aligned at bit position 11. Separate positive and negative delta values define an arbitrary hysteresis band.

Selecting Compare Values

Values for digital or analog compare operations can be selected from several sources. The separate GxBOUND registers provide software-defined compare values. Compare values can also be taken from a result register where it can be provided by another channel building a reference.

In Fast Compare Mode, the result registers provide the compare value while the bitfields of local GxBOUND registers define positive and negative delta values.

16.7.5 Compare with Standard Conversions (Limit Checking)

The limit checking mechanism can automatically compare each digital conversion result to an upper and a lower boundary value. A channel event can then be generated when the result of a conversion/comparison is inside or outside a user-defined band (see bitfield CHEVMODE and **Figure 16-11**).

This feature supports automatic range monitoring and minimizes the CPU load by issuing service requests only under certain predefined conditions.

Note: Channel events can also be generated for each result value (ignoring the band) or they can be suppressed completely.

The boundary values to which results are compared can be selected from several sources (see register GxCHCTRy).

While bitfield BNDSELX = 0000_B, bitfields BNDSELU and BNDSELL select the valid upper/lower boundary value either from the group-specific boundary register **GxBOUND (x = 0 - 1)** or from the global boundary register **GLOBBOUND**. The group boundary register can be selected for each channel of the respective group, the global boundary register can be selected by each available channel.

Otherwise, the compare values are taken from result registers, where bitfield BNDSELX selects the upper boundary value (GxRES1 … GxRES15), the concatenated bitfields BNDSELU||BNDSELL select the lower boundary value (GxRES0 … GxRES15).

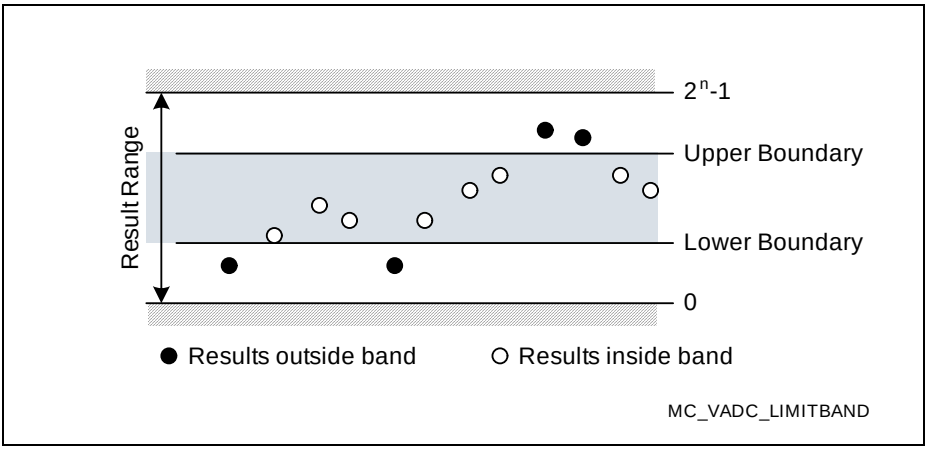


Figure 16-11 Result Monitoring through Limit Checking

A result value is considered inside the defined band when both of the following conditions are true:

- the value is less than or equal to the selected upper boundary
- the value is greater than or equal to the selected lower boundary

The result range can also be divided into two areas:
To select the lower part as valid band, set the lower boundary to the minimum value (000_H) and set the upper boundary to the highest intended value.
To select the upper part as valid band, set the upper boundary to the maximum value (FFF_H) and set the lower boundary to the lowest intended value.

Finding Extrema (Peak Detection)

The limit checking mechanism uses standard conversions and, therefore, always can provide the actual conversion result that was used for comparison. Combining this with a special FIFO mode, that only updates the corresponding FIFO stage if the result was above (or below) the current value of the stage, provides the usual conversion results and at the same time stores the highest (or lowest) result of a conversion sequence. For this operation the FIFO stage below the standard result must be selected as the compare value. Mode selection is done via bitfield FEN in register GxRCRy.

Before starting a peak detection sequence, write a reasonable start value to the result bitfield in the peak result register (e.g. 0000_H to find the maximum and $FFFF_H$ to find the minimum).

16.7.6 Utilizing Fast Compare Mode

In Fast Compare Mode, the input signal is directly compared to a value in the associated result register. This comparison just provides a binary result (above/below). If the exact result value is not required, this saves conversion time. A channel event can then be generated when the input signal becomes higher (or lower) than the compare value (see bitfield CHEVMODE and **Figure 16-12**).

The compare value in Fast Compare Mode is taken from the result register. Bitfields BOUNDARY1 and BOUNDARY0 in register **GxBOUND (x = 0 - 1)** define delta limits in this case. These deltas are added to (or subtracted from) the original compare value and allow defining an arbitrary hysteresis band.

The actual used compare value depends on the Fast Compare Result FCR (see registers **G0RESy (y = 0 - 15)**, etc.):

GxRESy.FCR = 0: reference value + upper delta
(GxRESy.RESULT + GxBOUND.BOUNDARY0)
GxRESy.FCR = 1: reference value - lower delta
(GxRESy.RESULT - GxBOUND.BOUNDARY1)

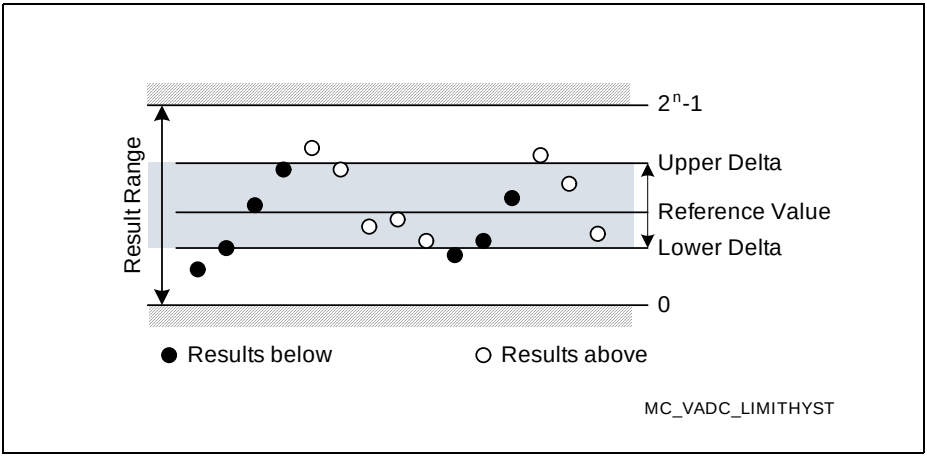


Figure 16-12 Result Monitoring through Compare with Hysteresis

16.7.7 Boundary Flag Control

Both limit checking mechanisms can be configured to automatically control the boundary flags. These boundary flags are also available as control signals for other modules. The flags can be set or cleared when the defined level is exceeded and the polarity of the output signal can be selected. A gate signal can be selected to enable the boundary flag operation while the gate is active.

Each boundary flag is available at group-specific output lines. Node pointers additionally route them to one of four boundary signals or one of the associated common service request lines, see register **GxBFLNP (x = 0 - 1)**.

For standard conversions, a boundary flag will be set/cleared when the conversion result is above the defined band, and will be cleared/set when the conversion result is below the defined band.

The band between the two boundary values defines a hysteresis for setting/clearing the boundary flags.

Using this feature on three channels that monitor linear hall elements can produce signals to feed the three hall position inputs of a unit that generates the corresponding PWM control signals.

In Fast Compare Mode, a boundary flag reflects the result of the comparisons, i.e. it will be set/cleared when the compared signal level is above the compare value, and will be cleared/set when the signal level is below the compare value. The delta values define a hysteresis band around the compare value.

Note: Clear register GxBOUND (i.e. the deltas) if a hysteresis is not wanted.

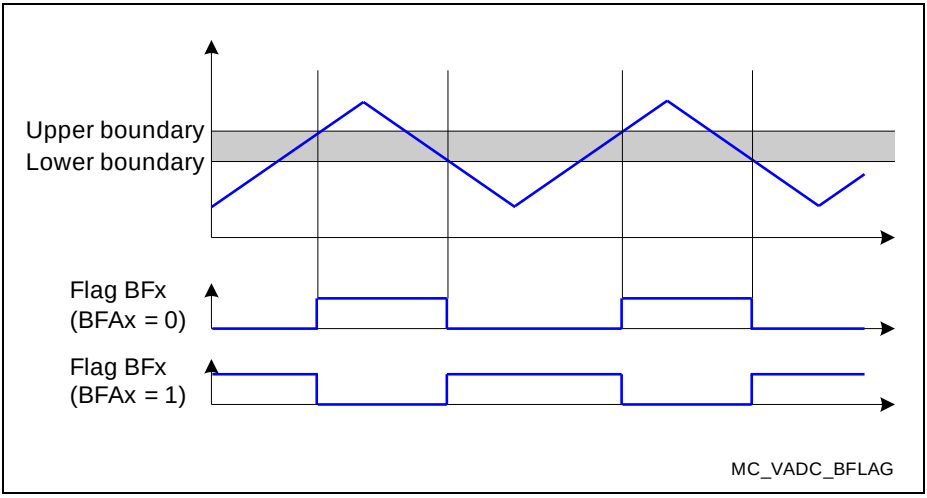


Figure 16-13 Boundary Flag Switching

Boundary flags can be switched by each compare operation, or the influence of compare operations can be restricted to the active phases of the corresponding request source gate signal (see **GxBFLC (x = 0 - 1)**).

Note: If a boundary flag is used together with Fast Compare Mode, it is recommended not to direct results from other channels to the corresponding result register.

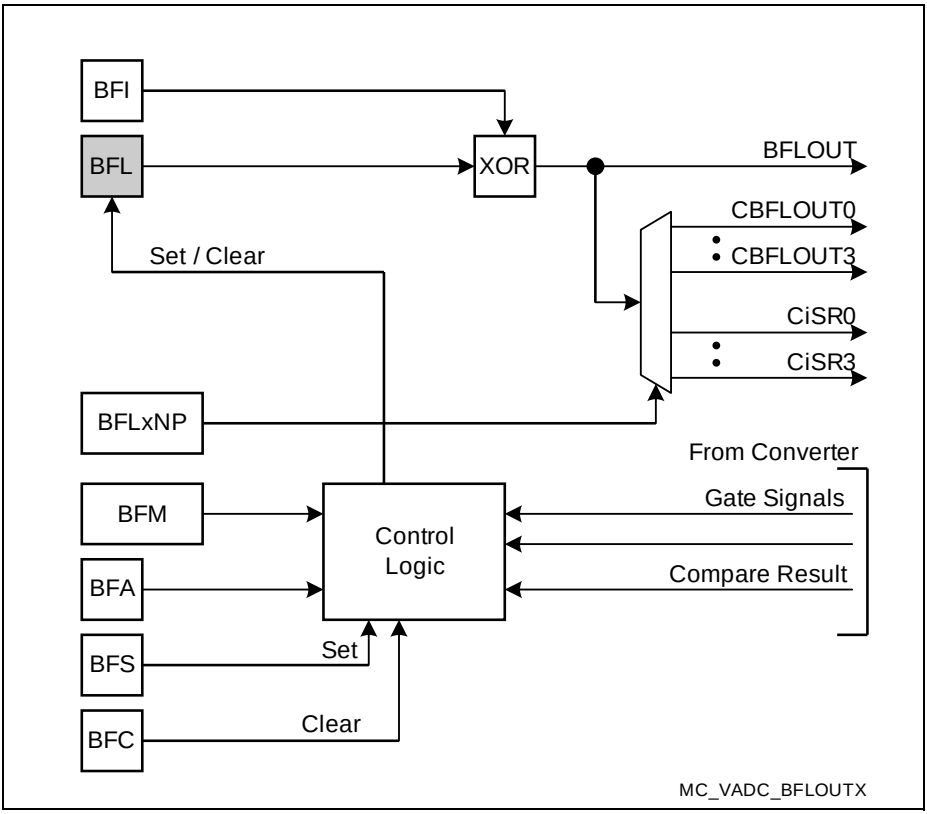


Figure 16-14 Boundary Flag Control

A boundary flag BFLy is assigned to result register GxRESy and thus to an arbitrary channel.

16.8 Conversion Result Handling

The A/D converters can preprocess the conversions result data to a certain extent before storing them for retrieval by the CPU or a DMA channel. This supports the subsequent handling of result data by the application software.

Conversion result handling comprises the following functions:

- **Storage of Conversion Results** to user-configurable registers
- **Data Alignment** according to result width and endianess
- **Wait-for-Read Mode** to avoid loss of data
- **Result Event Generation**
- Data reduction or anti-aliasing filtering (see **Section 16.8.6**)

16.8.1 Storage of Conversion Results

The conversion result values of a certain group can be stored in one of the 16 associated group result registers or in the common global result register (can be used, for example, for the channels of the background source (see **Selecting a Result Register**).

This structure provides different locations for the conversion results of different sets of channels. Depending on the application needs (data reduction, auto-scan, alias feature, etc.), the user can distribute the conversion results to minimize CPU load and/or optimize the performance of DMA transfers.

Each result register has an individual data valid flag (VF) associated with it. This flag indicates when “new” valid data has been stored in the corresponding result register and can be read out.

For standard conversions, result values are available in bitfield RESULT. Conversions in Fast Compare Mode use bitfield RESULT for the reference value, so the result of the operation is stored in bit FCR.

Result registers can be read via two different views. These views use different addresses but access the same register data:

- When a result register is read via the **application view**, the corresponding valid flag is automatically cleared when the result is read. This provides an easy handshake between result generation and retrieval. This also supports wait-for-read mode.
- When a result register is read via the **debug view**, the corresponding valid flag remains unchanged when the result is read. This supports debugging by delivering the result value without disturbing the handshake with the application.

The application can retrieve conversion results through several result registers:

- Group result register:
  Returns the result value and the channel number
- Global result register:
  Returns the result value and the channel number and the group number

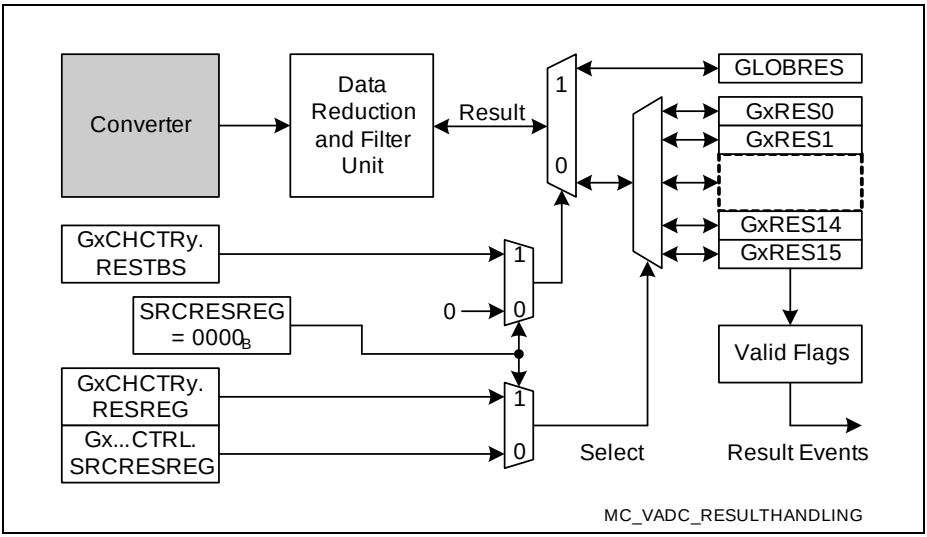


Figure 16-15 Conversion Result Storage

Selecting a Result Register

Conversion results are stored in result registers that can be assigned by the user according to the requirements of the application. The following bitfields direct the results to a register:

- SRCRESREG in register **GxQCTRL0 (x = 0 - 1)**, **GxASCTRL (x = 0 - 1)** or **BRSCTRL**
  Selects the group-specific result register GxRES1 … GxRES15
  when source-specific result registers are used
- RESTBS in register **G0CHCTRy (y = 0 - 7)** etc.
  Selects the global result register for results requested by the background source
- RESREG in register **G0CHCTRy (y = 0 - 7)** etc.
  Selects the group-specific result register GxRES0 … GxRES15
  when channel-specific result registers are used

Using source-specific result registers allows separating results from the same channel that are requested by different request sources. Usually these request sources are used by different tasks and are triggered at different times.

16.8.2 Data Alignment

The position of a conversion result value within the selected result register depends on 3 configurations (summary in **Figure 16-16**):

- The selected result width (12/10/8 bits, selected by the conversion mode)
- The selected result position (Left/Right-aligned
- The selected data accumulation mode (data reduction)

These options provide the conversion results in a way that minimizes data handling for the application software.

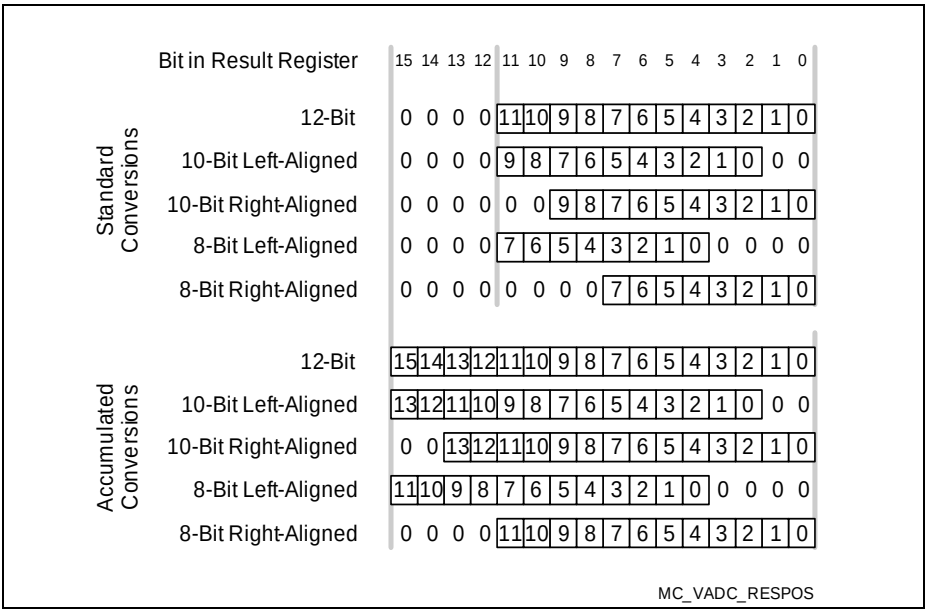


Figure 16-16 Result Storage Options

Bitfield RESULT can be written by software to provide the reference value for Fast Compare Mode. In this mode, bits 11-2 are evaluated, the other bits are ignored.

16.8.3 Wait-for-Read Mode

The wait-for-read mode prevents data loss due to overwriting a result register with a new conversion result before the CPU (or DMA) has read the previous data. For example, auto-scan conversion sequences or other sequences with “relaxed” timing requirements may use a common result register. However, the results come from different input channels, so an overwrite would destroy the result from the previous conversion[1)].

Wait-for-read mode automatically suspends the start of a conversion for this channel from this source until the current result has been read. So a conversion or a conversion sequence can be requested by a hardware or software trigger, while each conversion is only started after the result of the previous one has been read. This automatically aligns the conversion sequence with the CPU/DMA capability to read the formerly converted result (latency).

If wait-for-read mode is enabled for a result register (bit GxRCRy.WFR = 1), a request source does not generate a conversion request while the targeted result register contains valid data (indicated by the valid flag VF = 1) or if a currently running conversion targets the same result register.

If two request sources target the same result register with wait-for-read mode selected, a higher priority source cannot interrupt a lower priority conversion request started before the higher priority source has requested its conversion. Cancel-inject-repeat mode does not work in this case. In particular, this must be regarded if one of the involved sources is the background source (which usually has lowest priority). If the higher priority request targets a different result register, the lower priority conversion can be cancelled and repeated afterwards.

*Note: Wait-for-read mode is ignored for synchronized conversions of synchronization slaves (see **Section 16.9**).*

1) Repeated conversions of a single channel that use a separate result register will not destroy other results, but rather update their own previous result value. This way, always the actual signal data is available in the result register.

16.8.4 Result FIFO Buffer

Result registers can either be used as direct target for conversion results or they can be concatenated with other result registers of the same ADC group to form a result FIFO buffer (first-in-first-out buffer mechanism). A result FIFO stores several measurement results that can be read out later with a “relaxed” CPU response timing. It is possible to set up more than one FIFO buffer structure with the available result registers.

Result FIFO structures of two or more registers are built by concatenating result registers to their following “neighbor” result register (with next higher index, see **Figure 16-17**). This is enabled by setting bitfield GxRCRy.FEN = 01_B.
Conversion results are stored to the register with the highest index of a FIFO structure. Software reads the values from the FIFO register with the lowest index.

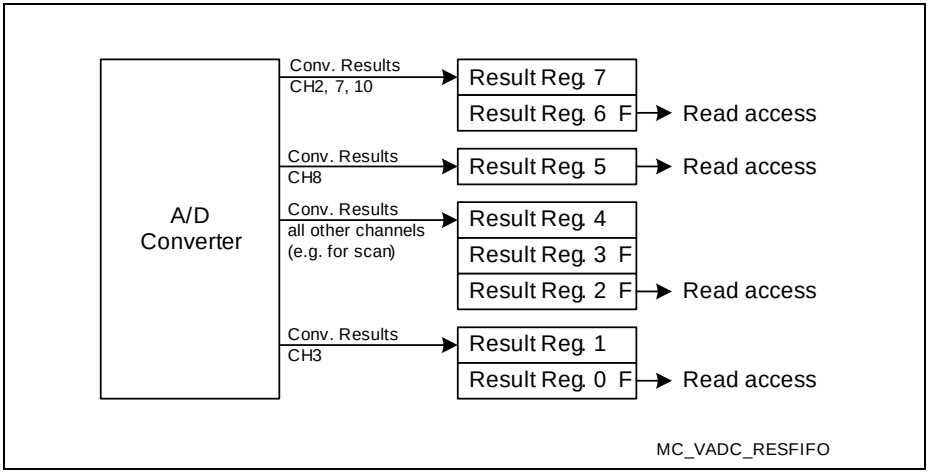


Figure 16-17 Result FIFO Buffers

In the example shown the result registers have been configured in the following way:

- 2-stage buffer consisting of result registers 7-6
- dedicated result register 5
- 3-stage buffer consisting of result registers 4-3-2
- 2-stage buffer consisting of result registers 1-0

Table 16-3 summarizes the required configuration of result registers if they are combined to build result FIFO buffers.

Table 16-3 Properties of Result FIFO Registers

Function	Input Stage	Intermed. Stage	Output Stage
Result target	YES	no	no
Application read	no	no	YES
Data reduction mode	YES	no	no
Wait-for-read mode	YES	no	no
Result event interrupt	no	no	YES
FIFO enable (FEN)	00_B	01_B	01_B
Registers in example	7, 4, 1	3	6, 2, 0

Note: If enabled, a result interrupt is generated for each data word in the FIFO.

16.8.5 Result Event Generation

A result event can be generated when a new value is stored to a result register. Result events can be restricted due to data accumulation and be generated only if the accumulation is complete.

Result events can also be suppressed completely.

16.8.6 Data Modification

The data resulting from conversions can be automatically modified before being used by an application. Several options can be selected (bitfield DMM in register **G0RCRy (y = 0 - 15)** etc.) which reduce the CPU/DMA load required to unload and/or process the conversion data.

- **Standard Data Reduction Mode (for GxRES0 … GxRES15):**
  Accumulates 2, 3, or 4 result values within each result register before generating a result interrupt. This can remove some noise from the input signal.
- **Result Filtering Mode (FIR, for GxRES7, GxRES15):**
  Applies a 3rd order Finite Impulse Response Filter (FIR) with selectable coefficients to the conversion results for the selected result register.
- **Result Filtering Mode (IIR, for GxRES7, GxRES15):**
  Applies a 1st order Infinite Impulse Response Filter (IIR) with selectable coefficients to the conversion results for the selected result register.
- **Difference Mode (for GxRES1 … GxRES15):**
  Subtracts the contents of result register GxRES0 from the conversion results for the selected result register. Bitfield DRCTR is not used in this mode.

Table 16-4 Function of Bitfield DRCTR

DRCTR	Standard Data Reduction Mode (DMM = 00_B)	DRCTR	Result Filtering Mode (DMM = 01_B)[1)]
0000_B	Data Reduction disabled	0000_B	FIR filter: a=2, b=1, c=0
0001_B	Accumulate 2 result values	0001_B	FIR filter: a=1, b=2, c=0
0010_B	Accumulate 3 result values	0010_B	FIR filter: a=2, b=0, c=1
0011_B	Accumulate 4 result values	0011_B	FIR filter: a=1, b=1, c=1
0100_B	Reserved	0100_B	FIR filter: a=1, b=0, c=2
0101_B	Reserved	0101_B	FIR filter: a=3, b=1, c=0
0110_B	Reserved	0110_B	FIR filter: a=2, b=2, c=0
0111_B	Reserved	0111_B	FIR filter: a=1, b=3, c=0
1000_B	Reserved	1000_B	FIR filter: a=3, b=0, c=1
1001_B	Reserved	1001_B	FIR filter: a=2, b=1, c=1
1010_B	Reserved	1010_B	FIR filter: a=1, b=2, c=1
1011_B	Reserved	1011_B	FIR filter: a=2, b=0, c=2
1100_B	Reserved	1100_B	FIR filter: a=1, b=1, c=2
1101_B	Reserved	1101_B	FIR filter: a=1, b=0, c=3

Table 16-4 Function of Bitfield DRCTR (cont'd)

DRCTR	Standard Data Reduction Mode (DMM = 00_B)	DRCTR	Result Filtering Mode (DMM = 01_B)[1]
1110_B	Reserved	1110_B	IIR filter: a=2, b=2
1111_B	Reserved	1111_B	IIR filter: a=3, b=4

1) The filter registers are cleared while bitfield DMM ≠ 01_B.

Standard Data Reduction Mode

The data reduction mode can be used as digital filter for anti-aliasing or decimation purposes. It accumulates a maximum of 4 conversion values to generate a final result.

Each result register can be individually enabled for data reduction, controlled by bitfield DRCTR in registers **G0CHCTRy (y = 0 - 7)**. The data reduction counter DRC indicates the actual status of the accumulation.

Note: Conversions for other result registers can be inserted between conversions to be accumulated.

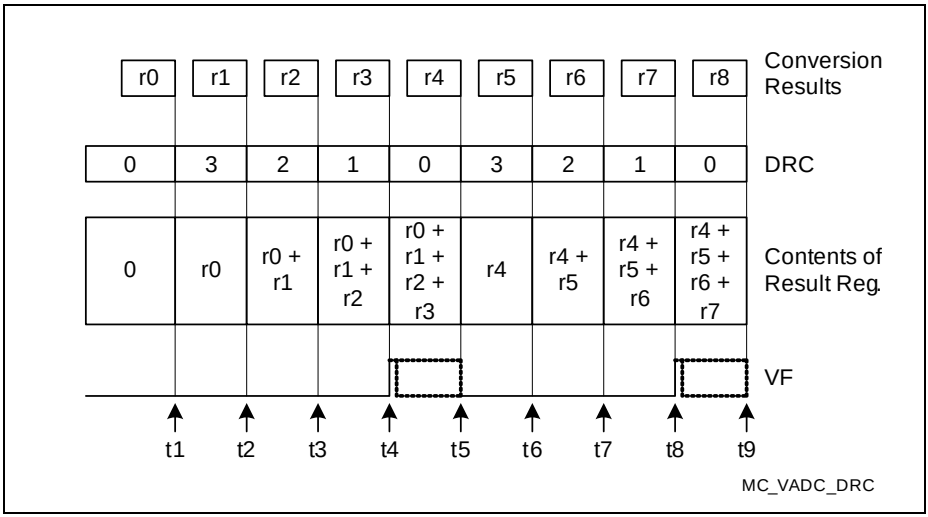


Figure 16-18 Standard Data Reduction Filter

This example shows a data reduction sequence of 4 accumulated conversion results. Eight conversion results (r0 … r7) are accumulated and produce 2 final results.

When a conversion is complete and stores data to a result register that has data reduction mode enabled, the data handling is controlled by the data reduction counter DRC:

- If DRC = 0 (t1, t5, t9 in the example), the conversion result is stored to the register. DRC is loaded with the contents of bitfield DRCTR (i.e. the accumulation begins).
- If DRC > 0 (t2, t3, t4 and t6, t7, t8 in the example), the conversion result is added to the value in the result register.
  DRC is decremented by 1.
- If DRC becomes 0, either decremented from 1 (t4 and t8 in the example) or loaded from DRCTR, the valid bit for the respective result register is set and a result register event occurs.

The final result must be read before the next data reduction sequence starts (before t5 or t9 in the example). This automatically clears the valid flag.

*Note: Software can clear the data reduction counter DRC by clearing the corresponding valid Flag (via **GxVFR (x = 0 - 1)**).*

The response time to read the final data reduction results can be increased by associating the adjacent result register to build a result FIFO (see **Figure 16-19**). In this case, the final result of a data reduction sequence is loaded to the adjacent register. The value can be read from this register until the next data reduction sequence is finished (t8 in the 2nd example).

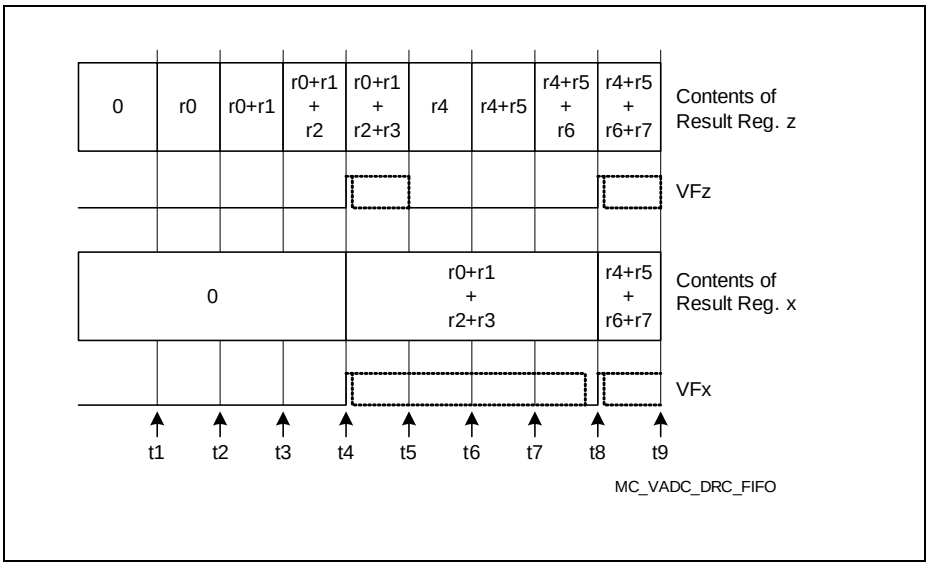


Figure 16-19 Standard Data Reduction Filter with FIFO Enabled

Finite Impulse Response Filter Mode (FIR)

The FIR filter (see **Figure 16-20**) provides 2 result buffers for intermediate results (RB1, RB2) and 3 configurable tap coefficients (a, b, c).

The conversion result and the intermediate result buffer values are added weighted with their respective coefficients to form the final value for the result register. Several predefined sets of coefficients can be selected via bitfield DRCTR (coding listed in **Table 16-4**) in registers **G0RESy (y = 0 - 15)** and **GLOBRES**. These coefficients lead to a gain of 3 or 4 to the ADC result producing a 14-bit value. The valid flag (VF) is activated for each sample after activation, i.e. for each sample generates a valid result.

Note: Conversions for other result registers can be inserted between conversions to be filtered.

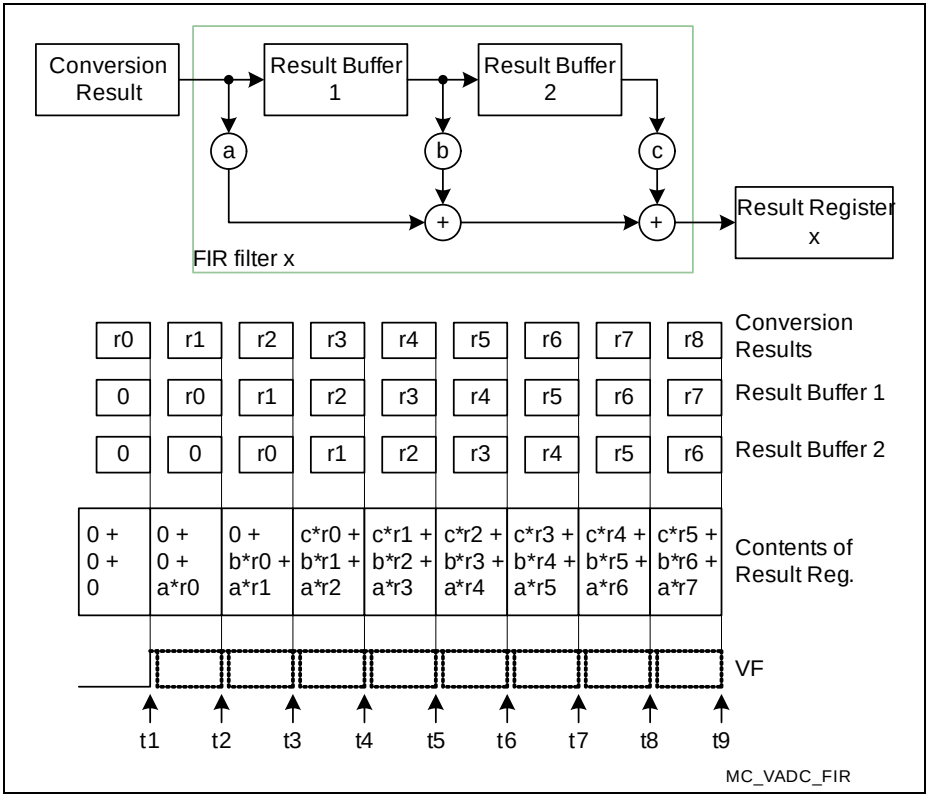


Figure 16-20 FIR Filter Structure

Note: The filter registers are cleared while bitfield DMM ≠ 01_B.

Infinite Impulse Response Filter Mode (IIR)

The IIR filter (see **Figure 16-21**) provides a result buffer (RB) and 2 configurable coefficients (a, b). It represents a first order low-pass filter.

The conversion result, weighted with the respective coefficient, and a fraction of the previous result are added to form the final value for the result register. Several predefined sets of coefficients can be selected via bitfield DRCTR (coding listed in **Table 16-4**) in registers **G0RESy (y = 0 - 15)** and **GLOBRES**. These coefficients lead to a gain of 4 to the ADC result producing a 14-bit value. The valid flag (VF) is activated for each sample after activation, i.e. for each sample generates a valid result.

Note: Conversions for other result registers can be inserted between conversions to be filtered.

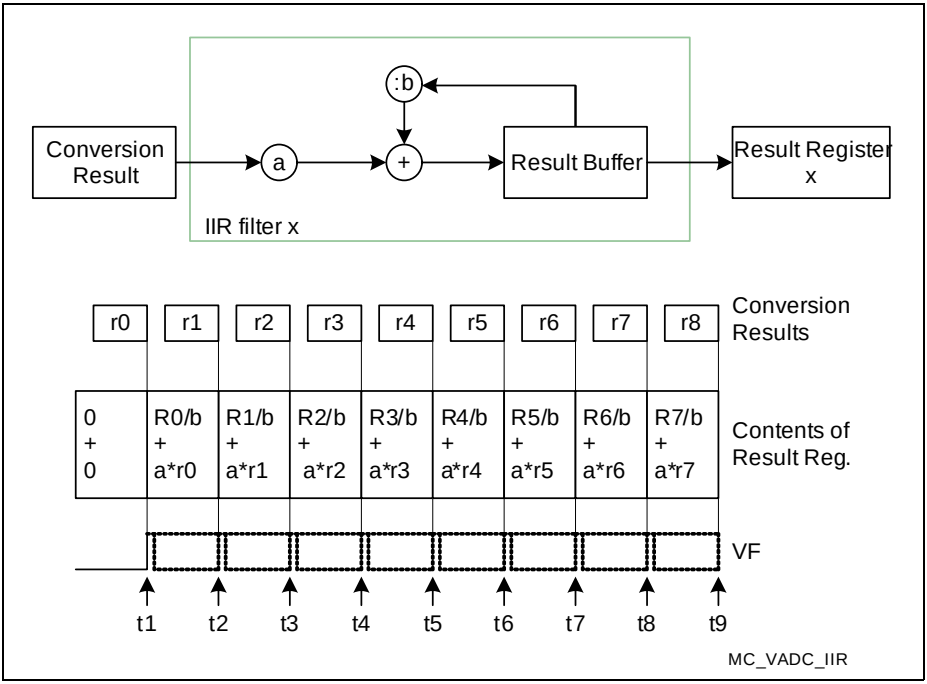


Figure 16-21 IIR Filter Structure

Note: The filter registers are cleared while bitfield DMM ≠ 01_B.

Difference Mode

Subtracting the contents of result register 0 from the actual result puts the results of the respective channel in relation to another signal. No software action is required.

The reference channel must store its result(s) into result register 0. The reference value can be determined once and then be used for a series of conversions, or it can be converted before each related conversion.

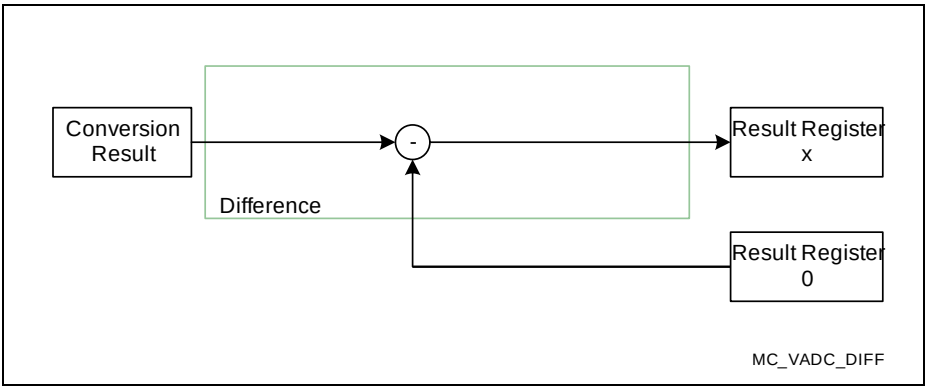


Figure 16-22 Result Difference

16.9 Synchronization of Conversions

The conversions of an ADC kernel can be scheduled either self-timed according to the kernel's configuration or triggered by external (outside the ADC) signals:

Synchronized Conversions for Parallel Sampling support parallel conversion of channels within a synchronization group[1]. This optimizes e.g. the control of electrical drives.

Equidistant Sampling supports conversions in a fixed raster with minimum jitter. This optimizes e.g. filter algorithms or audio applications.

16.9.1 Synchronized Conversions for Parallel Sampling

Several independent ADC kernels[1] implemented in the XMC4[12]00 can be synchronized for simultaneous measurements of analog input channels. While no parallel conversion is requested, the kernels can work independently.

The synchronization mechanism for parallel conversions ensures that the sample phases of the related channels start simultaneously. Synchronized kernels convert the same channel that is requested by the master. Different values for the resolution and the sample phase length of each kernel for a parallel conversion are supported.

A parallel conversion can be requested individually for each input channel (one or more). In the example shown in **Figure 16-23**, input channels CH3 of the ADC kernels 0 and 1 are converted synchronously, whereas other input channels do not lead to parallel conversions.

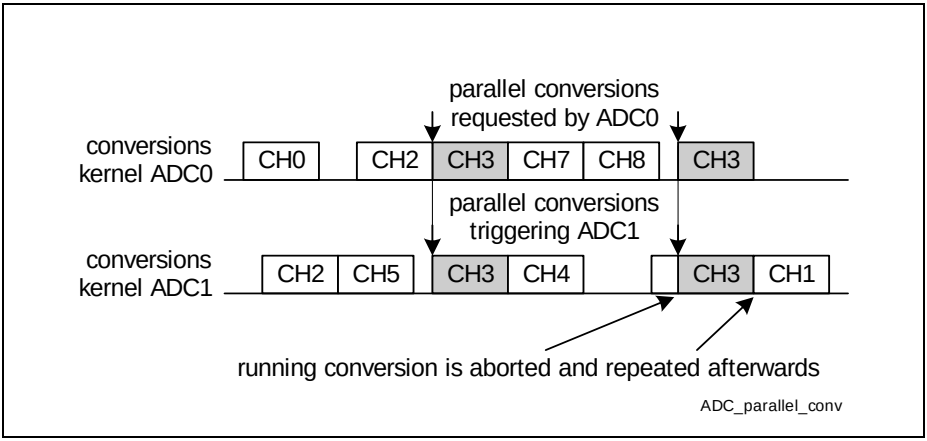


Figure 16-23 Parallel Conversions

1) For a summary, please refer to **"Synchronization Groups in the XMC4[12]00" on Page 16-134**.

One kernel operates as synchronization master, the other kernel(s) operate(s) as synchronization slave. Each kernel can play either role. The arbiters of all involved kernels must run synchronously. This is achieved by switching off all involved kernels before the initialization and switching on the master kernel at the end of the initialization sequence.

Master and slave kernels form a “conversion group” to control parallel sampling:

- The arbiters must run permanently (bits **GxARBCFG (x = 0 - 1)**.ARBM = 0). Initialize the slave before the master to have the arbiters run synchronously. Set the master’s GxARBCFG.ANONC at the end of the initialization.
- **The synchronization master** controls the slave(s) by providing the control information **GxARBCFG (x = 0 - 1)**.ANONS (see **Figure 16-24**) and the requested channel number.
  - Bitfield **GxSYNCTR (x = 0 - 1)**.STSEL = 00_B selects the master’s ANON information as the source of the ANON information for all kernels of the synchronization group.[1)]
  - The ready signals indicate when a slave kernel is ready to start the sample phase of a parallel conversion. Bit **GxSYNCTR (x = 0 - 1)**.EVALRy = 1 enables the control by the ready signal (in the example kernel 1 is the slave, so EVALR1 = 1).
  - The master requests a synchronized conversion of a certain channel (SYNC = 1 in the corresponding channel control register GxCHCTRy), which is also requested in the connected slave ADC kernel(s).
  - Wait-for-read mode is supported for the master.
- **The synchronization slave** reacts to incoming synchronized conversion requests from the master. While no synchronized conversions are requested, the slave kernel can execute “local” conversions.
  - Bitfield **GxSYNCTR (x = 0 - 1)**.STSEL = 01_B/10_B/11_B selects the master’s ANON information as the source of the ANON information for all kernels of the synchronization group[1)] (in the example kernel 0 is the master, so STSEL = 01_B).
  - The ready signals indicate when the master kernel and the other slave kernels are ready to start the sample phase of a parallel conversion. Bit **GxSYNCTR (x = 0 - 1)**.EVALRy = 1 enables the control by the ready signal (in the example kernel 0 is the master, so EVALR1 = 1).
  - The slave timing must be configured according to the master timing (ARBRND in register **GxARBCFG (x = 0 - 1)**) to enable parallel conversions.
  - A parallel conversion request is always handled with highest priority and cancel-inject-repeat mode.
  - Wait-for-read mode is ignored in the slave. Previous results may be overwritten, in particular, if the same result register is used by other conversions.

1) STSEL = 00_B selects the own ANON information. The other control inputs (STSEL = 01_B/10_B/11_B) are connected to the other kernels of a synchronization group in ascending order (see also **Table 16-11 “Synchronization Groups in the XMC4[12]00” on Page 16-134**).

- Once started, a parallel conversion cannot be aborted.

*Note: Synchronized conversions request the same channel number, defined by the master. Using the alias feature (see **Section 16.7.3**), analog signals from different input channels can be converted. This is advantageous if e.g. CH0 is used as alternate reference.*

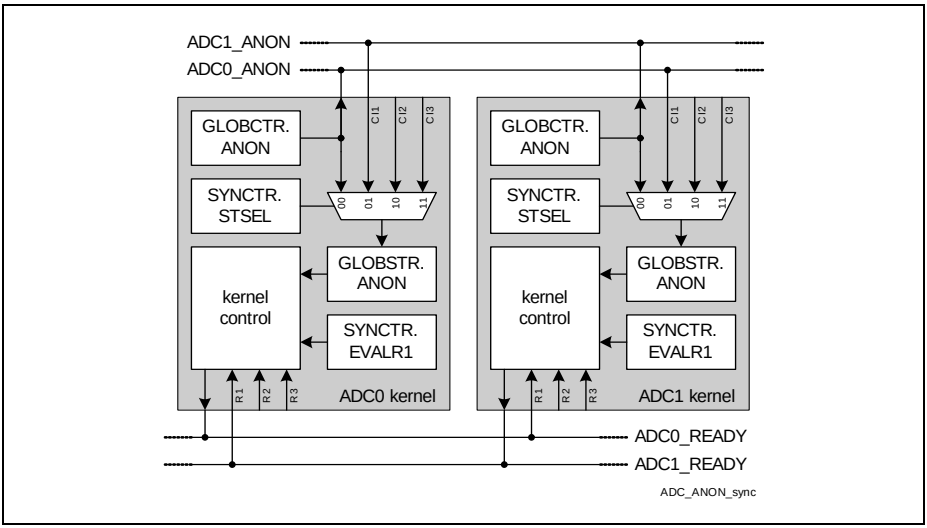


Figure 16-24 Synchronization via ANON and Ready Signals

16.9.2 Equidistant Sampling

To optimize the input data e.g. for filter or audio applications, conversions can be executed in a fixed timing raster. Conversions for equidistant sampling are triggered by an external signal (e.g. a timer). To generate the trigger signal synchronous to the arbiter, the ADC provides an output signal (ARBCNT) that is activated once per arbitration round and serves as timing base for the trigger timer. In this case, the arbiter must run permanently (**GxARBCFG (x = 0 - 1)**.ARBM = 0). If the timer has an independent time base, the arbiter can be stopped while no requests are pending. The preface time (see **Figure 16-25**) must be longer than one arbitration round and the highest possible conversion time.

Select timer mode (TMEN = 1 in register **GxQCTRL0 (x = 0 - 1)** or **GxASCTRL (x = 0 - 1)**) for the intended source of equidistant conversions. In timer mode, a request of this source is triggered and arbitrated, but only started when the trigger signal is removed (see **Figure 16-25**) and the converter is idle.

To ensure that the converter is idle and the start of conversion can be controlled by the trigger signal, the equidistant conversion requests must receive highest priority. The preface time between request trigger and conversion start must be long enough for a currently active conversion to finish.

The frequency of signal REQTRx defines the sampling rate and its high time defines the preface time interval where the corresponding request source takes part in the arbitration.

Depending on the used request source, equidistant sampling is also supported for a sequence of channels. It is also possible to do equidistant sampling for more than one request source in parallel if the preface times and the equidistant conversions do not overlap.

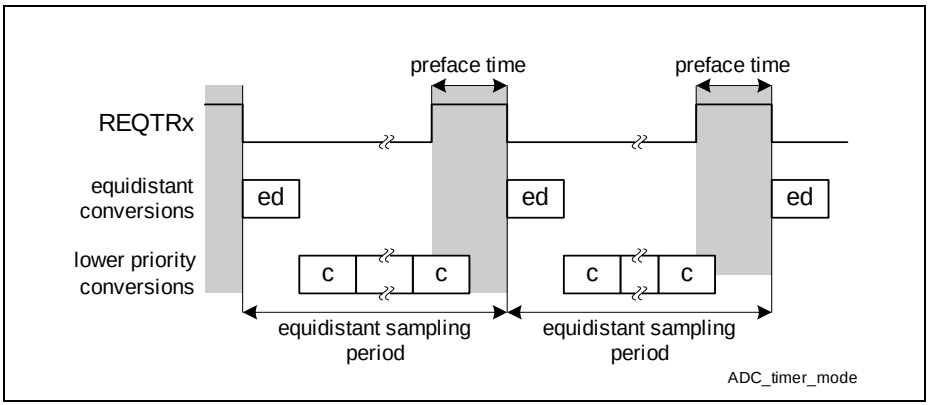


Figure 16-25 Timer Mode for Equidistant Sampling

16.10 Safety Features

Several test features can be enabled to verify the validity of the analog input signals of an application. These test features aim at different sections of the signal flow:

- **Broken Wire Detection** validates the connection from the sensor to the input pin,
- **Multiplexer Diagnostics** validates the operation of the internal analog input multiplexer,
- **Converter Diagnostics** validates the operation of the Analog/Digital converter itself.

16.10.1 Broken Wire Detection

To test the proper connection of an external analog sensor to its input pin, the converter's capacitor can be precharged to a selectable value before the regular sample phase. If the connection to the sensor is interrupted the subsequent conversion value will rather represent the precharged value than the expected sensor result. By using a precharge voltage outside the expected result range (broken wire detection preferably uses V_{AGND} and/or V_{AREF}) a valid measurement (sensor connected) can be distinguished from a failure (sensor detached).

While broken wire detection is disabled, the converter's capacitor is precharged to $V_{AREF}/2$.

Note: The duration of the complete conversion is increased by the preparation phase (same as the sample phase) if the broken wire detection is enabled. This influences the timing of conversion sequences.

Broken wire detection can be enabled for each channel separately by bitfield BWDEN in the corresponding channel control register (**G0CHCTRy (y = 0 - 7)**). This bitfield also selects the level for the preparation phase.

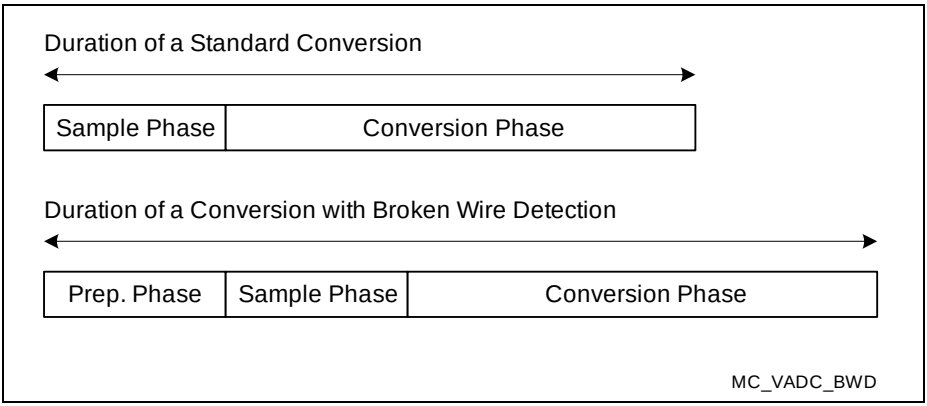


Figure 16-26 Broken Wire Detection

16.10.2 Signal Path Test Modes

Additional test structures can be activated to test the signal path from the sensor to the input pin and the internal signal path from the input pin through the multiplexer to the converter. These test structures apply additional loads to the signal path (see summary in **Figure 16-27**).

Multiplexer Diagnostics

To test the proper operation of the internal analog input multiplexer, additional pull-up and/or pull-down devices can be connected to a channel. In combination with a known external input signal this test function shows if the multiplexer connects any pin to the converter input and if this is the correct pin. These pull-up/pull-down devices are controlled via the port logic.

Pull-Down Diagnostics

One single input channel provides a further strong pull-down (R_{PDD}) that can be activated to verify the external connection to a sensor.

Converter Diagnostics

To test the proper operation of the converter itself, several signals can be connected to the converter input. The test signals can be connected to the converter input either instead of the standard input signal or in parallel to the standard input signal.

The test signal can be selected from four different signals as shown in **Figure 16-27**.

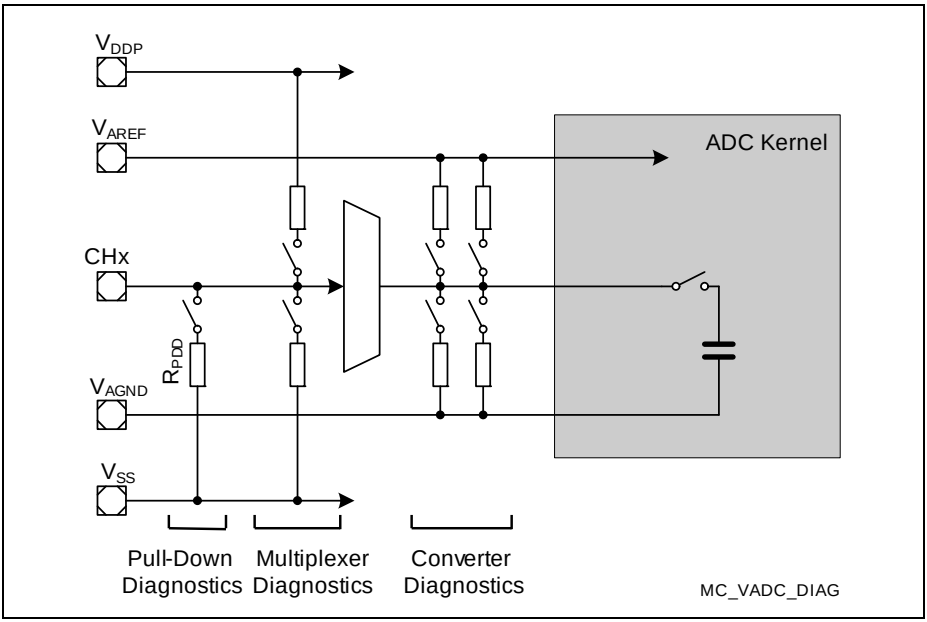


Figure 16-27 Signal Path Test

16.10.3 Configuration of Test Functions

The pull-up and pull-down devices for the test functions can be enabled individually under software control. Various test levels can be applied controlling the devices in an adequate way. Because these test functions interfere with the normal operation of the A/D Converters, they are controlled by a separate register set or by port registers.

Not all test options are available for each channel. Selecting an unavailable function has no effect.

16.11 External Multiplexer Control

The number of analog input channels can be increased by connecting external analog multiplexers to an input channel. The ADC can be configured to control these external multiplexers automatically.

For each available EMUX interface (see register **EMUXSEL**) one channel can be selected for this operating mode. The ADC supports 1-out-of-8 multiplexers with several control options:

- **Sequence mode** automatically converts all configured external channels when the selected channel is encountered. In the example in **Figure 16-28** the following conversions are done: --4-32-31-30-2-1-0--4-32-31-30-2-1-0--…
- **Single-step mode** converts one external channel of the configured sequence when the selected channel is encountered. In the example in **Figure 16-28** the following conversions are done: --4-32-2-1-0--4-31-2-1-0--4-30-2-1-0--4-32-… (Single-step mode works best with one channel)
- **Steady mode** converts the configured external channel when the selected channel is encountered. In the example in **Figure 16-28** the following conversions are done: --4-32-2-1-0--4-32-2-1-0--4-32-2-1-0--…

*Note: The example in **Figure 16-28** has an external multiplexer connected to channel CH3. The start selection value EMUXSET is assumed as 2.*

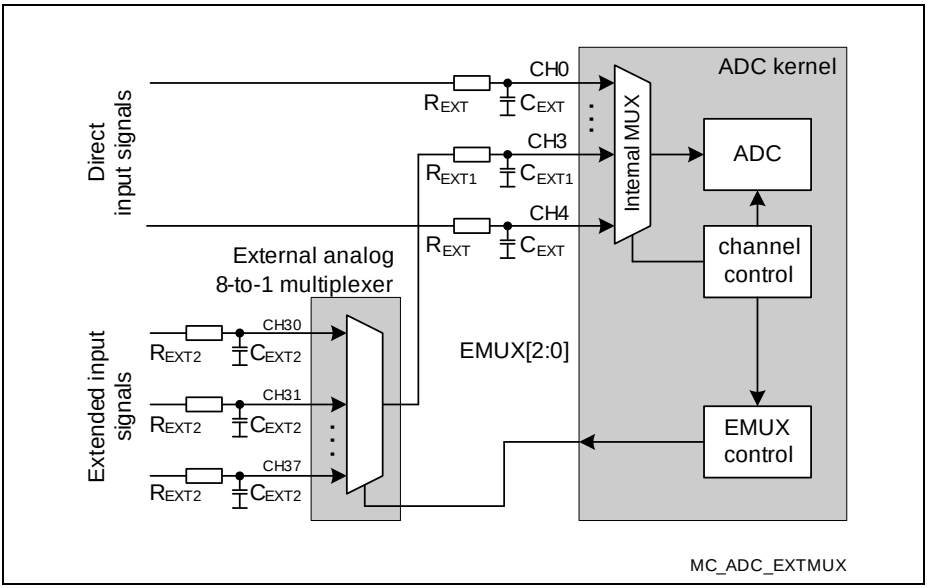


Figure 16-28 External Analog Multiplexer Example

Bitfield EMUXACT determines the control information sent to the external multiplexer. In single-step mode, EMUXACT is updated after each conversion of an enabled channel. If EMUXACT = 000_B it is reloaded from bitfield EMUXSET, otherwise it is decremented by 1.

Additional external channels may have different properties due to the modified signal path. Local filters may be used at the additional inputs (R_{EXT2}-C_{EXT2} on CH3x in **Figure 16-28**). For applications where the external multiplexer is located far from the ADC analog input, it is recommended to add an RC filter directly at the analog input of the ADC (R_{EXT1}-C_{EXT1} on CH3 in **Figure 16-28**).

Note: Each RC filter limits the bandwidth of the analog input signal.

Conversions for external channels, therefore, use the alternate conversion mode setting CME. This automatically selects a different conversion mode if required.

Switching the external multiplexer usually requires an additional settling time for the input signal. Therefore, the alternate sample time setting STCE is applied each time the external channel is changed. This automatically fulfills the different sampling time requirements in this case.

In each group an arbitrary channel can be assigned to external multiplexer control (register **GxEMUXCTR (x = 0 - 1)**). Each available port interface selects the group whose control lines are output (register **EMUXSEL**).

Control Signals

The external channel number that controls the external multiplexer can be output in standard binary format or Gray-coded. Gray code avoids intermediate multiplexer switching when selecting a sequence of channels, because only one bit changes at a time. **Table 16-5** indicates the resulting codes.

Table 16-5 EMUX Control Signal Coding

Channel	0	1	2	3	4	5	6	7
Binary	000_B	001_B	010_B	011_B	100_B	101_B	110_B	111_B
Gray	000	001	011	010	110	111	101	100

Operation Without External Multiplexer

If no external multiplexers are used in an application, the reset values of the control registers provide the appropriate setup.

EMUXMODE = 00_B disables the automatic EMUX control.

Since the control output signals are alternate port output signals, they are only visible at the respective pins if explicitly selected.

16.12 Service Request Generation

Each A/D Converter can activate up to 4 group-specific service request output signals and up to 4 shared service request output signals to issue an interrupt or to trigger a DMA channel. Two common service request groups are available, see **Table 16-10 "General Converter Configuration in the XMC4[12]00" on Page 16-133**.

Several events can be assigned to each service request output. Service requests can be generated by three types of events:

- **Request source events:** indicate that a request source completed the requested conversion sequence and the application software can initiate further actions.
  For a scan source (group or background), the event is generated when the complete defined set of channels (pending bits) has been converted.
  For a group queue source, the event is generated according to the programming, i.e. when a channel with enabled source interrupt has been converted or when an invalid entry is encountered.
- **Channel events:** indicate that a conversion is finished. Optionally, channel events can be restricted to result values within a programmable value range. This offloads the CPU/DMA from background tasks, i.e. a service request is only activated if the specified conversion result range is met or exceeded.
- **Result events:** indicate a new valid result in a result register. Usually, this triggers a read action by the CPU (or DMA). Optionally, result events can be generated only at a reduced rate if data reduction is active.
  For example, a single DMA channel can read the results for a complete auto-scan sequence, if all channels of the sequence target the same result register and the transfers are triggered by result events.

Each ADC event is indicated by a dedicated flag that can be cleared by software. If a service request is enabled for a certain event, the service request is generated for each event, independent of the status of the corresponding event indication flag. This ensures efficient DMA handling of ADC events (the ADC event can generate a service request without the need to clear the indication flag).

Event flag registers indicate all types of events that occur during the ADC's operation. Software can set each flag by writing a 1 to the respective position in register GxCEFLAG/GxRFLAG to trigger an event. Software can clear each flag by writing a 1 to the respective position in register GxCEFCLR/GxREFCLR. If enabled, service requests are generated for each occurrance of an event, even if the associated flag remains set.

Node Pointer Registers

Requests from each event source can be directed to a set of service request nodes via associated node pointers. Requests from several sources can be directed to the same node; in this case, they are ORed to the service request output signal.

Software Service Request Activation

Each service request can be activated via software by setting the corresponding bit in register **GxSRACT (x = 0 - 1)**. This can be used for evaluation and testing purposes.

*Note: For shared service request lines see common groups in **Table 16-10**.*

16.13 Registers

The Versatile ADC is built from a series of converter blocks that are controlled in an identical way. This makes programming versatile and scalable. The corresponding registers, therefore, have an individual offset assigned (see **Table 16-7**). The exact register location is obtained by adding the respective register offset to the base address (see **Table 16-6**) of the corresponding group.

Due to the regular group structure, several registers appear within each group. Other registers are provided for each channel. This is indicated in the register overview table by placeholders:

- X###$_H$ means: x × 0400$_H$ + 0###$_H$, for x = 0 - 1
- ###Y$_H$ means: ###0$_H$ + y × 0004$_H$, for y = 0 - N (depends on register type)

Table 16-6 Registers Address Space

Module	Base Address	End Address	Note
VADC	4000 4000$_H$	4000 7FFF$_H$	

Table 16-7 Registers Overview

Register Short Name	Register Long Name	Offset Addr.	Access Mode		Page Num.
			Read	Write	
ID	Module Identification Register	0008$_H$	U, PV	BE	16-61
CLC	Clock Control Register	0000$_H$	U, PV	PV	16-63
OCS	OCDS Control and Status Register	0028$_H$	U, PV	PV	16-64
GLOBCFG	Global Configuration Register	0080$_H$	U, PV	U, PV	16-66
GxARBCFG	Arbitration Configuration Register	X480$_H$	U, PV	U, PV	16-68
GxARBPR	Arbitration Priority Register	X484$_H$	U, PV	U, PV	16-70
GxCHASS	Channel Assignment Register, Group x	X488$_H$	U, PV	U, PV	16-67
GxQCTRL0	Queue 0 Source Control Register, Group x	X500$_H$	U, PV	U, PV	16-72
GxQMR0	Queue 0 Mode Register, Group x	X504$_H$	U, PV	U, PV	16-74
GxQSR0	Queue 0 Status Register, Group x	X508$_H$	U, PV	U, PV	16-76
GxQINR0	Queue 0 Input Register, Group x	X510$_H$	U, PV	U, PV	16-78
GxQ0R0	Queue 0 Register 0, Group x	X50C$_H$	U, PV	U, PV	16-80
GxQBUR0	Queue 0 Backup Register, Group x	X510$_H$	U, PV	U, PV	16-82

Table 16-7 Registers Overview (cont'd)

Register Short Name	Register Long Name	Offset Addr.	Access Mode		Page Num.
			Read	Write	
GxASCTRL	Autoscan Source Control Register, Group x	$X520_H$	U, PV	U, PV	16-84
GxASMR	Autoscan Source Mode Register, Group x	$X524_H$	U, PV	U, PV	16-86
GxASSEL	Autoscan Source Channel Select Register, Group x	$X528_H$	U, PV	U, PV	16-88
GxASPND	Autoscan Source Pending Register, Group x	$X52C_H$	U, PV	U, PV	16-89
BRSCTRL	Background Request Source Control Register	0200_H	U, PV	U, PV	16-90
BRSMR	Background Request Source Mode Register	0204_H	U, PV	U, PV	16-92
BRSSELx	Background Request Source Channel Select Register, Group x	$018Y_H$	U, PV	U, PV	16-94
BRSPNDx	Background Request Source Channel Pending Register, Group x	$01CY_H$	U, PV	U, PV	16-95
GxCHCTRy	Channel x Control Register	$X60Y_H$	U, PV	U, PV	16-96
GxICLASS0	Input Class Register 0, Group x	$X4A0_H$	U, PV	U, PV	16-98
GxICLASS1	Input Class Register 1, Group x	$X4A4_H$	U, PV	U, PV	16-98
GLOBICLASS0	Input Class Register 0, Global	$00A0_H$	U, PV	U, PV	16-98
GLOBICLASS1	Input Class Register 1, Global	$00A4_H$	U, PV	U, PV	16-98
GxALIAS	Alias Register	$X4B0_H$	U, PV	U, PV	16-109
GxBOUND	Boundary Select Register, Group x	$X4B8_H$	U, PV	U, PV	16-111
GLOBBOUND	Global Boundary Select Register	$00B8_H$	U, PV	U, PV	16-111
GxBFL	Boundary Flag Register, Group x	$X4C8_H$	U, PV	U, PV	16-112
GxBFLS	Boundary Flag Software Register, Group x	$X4CC_H$	U, SV	U, SVP	16-113
GxBFLC	Boundary Flag Control Register, Group x	$X4D0_H$	U, SV	U, SVP	16-114

Table 16-7 Registers Overview (cont'd)

Register Short Name	Register Long Name	Offset Addr.	Access Mode		Page Num.
			Read	Write	
GxBFLNP	Boundary Flag Node Pointer Register, Group x	$X4D4_H$	U, SV	U, SV P	16-115
GxRCRy	Group x Result Control Register y	$X68Y_H$	U, PV	U, PV	16-101
GxRESy	Group x Result Register y	$X70Y_H$	U, PV	U, PV	16-103
GxRESDy	Group x Result Register y (debug view)	$X78Y_H$	U, PV	U, PV	16-105
GLOBRCR	Global Result Control Register	0280_H	U, PV	U, PV	16-106
GLOBRES	Global Result Register	0300_H	U, PV	U, PV	16-107
GLOBRESD	Global Result Register (debug view)	0380_H	U, PV	U, PV	16-107
GxVFR	Valid Flag Register, Group x	$X5F8_H$	U, PV	U, PV	16-109
GxSYNCTR	Synchronization Control Register	$X4C0_H$	U, PV	U, PV	16-116
GLOBTF	Global Test Functions Register	0160_H	U, PV	U, PV	16-117
GxEMUXCTR	External Multiplexer Control Register, Group x	$X5F0_H$	U, PV	U, PV	16-118
EMUXSEL	External Multiplexer Select Register	$03F0_H$	U, PV	U, PV	16-121
GxSEFLAG	Source Event Flag Register, Group x	$X588_H$	U, PV	U, PV	16-121
GxCEFLAG	Channel Event Flag Register, Group x	$X580_H$	U, PV	U, PV	16-122
GxREFLAG	Result Event Flag Register, Group x	$X584_H$	U, PV	U, PV	16-123
GxSEFCLR	Source Event Flag Clear Register, Group x	$X598_H$	U, PV	U, PV	16-123

Table 16-7 Registers Overview (cont'd)

Register Short Name	Register Long Name	Offset Addr.	Access Mode		Page Num.
			Read	Write	
GxCEFCLR	Channel Event Flag Clear Register, Group x	$X590_H$	U, PV	U, PV	16-124
GxREFCLR	Result Event Flag Clear Register, Group x	$X594_H$	U, PV	U, PV	16-125
GLOBEFLAG	Global Event Flag Register	$00E0_H$	U, PV	U, PV	16-125
GxSEVNP	Source Event Node Pointer Register, Group x	$X5C0_H$	U, PV	U, PV	16-126
GxCEVNP0	Channel Event Node Pointer Register 0, Group x	$X5A0_H$	U, PV	U, PV	16-127
GxREVNP0	Result Event Node Pointer Register 0, Group x	$X5B0_H$	U, PV	U, PV	16-128
GxREVNP1	Result Event Node Pointer Register 1, Group x	$X5B4_H$	U, PV	U, PV	16-129
GLOBEVNP	Global Event Node Pointer Register	0140_H	U, PV	U, PV	16-130
GxSRACT	Service Request Software Activation Trigger, Group x	$X5C8_H$	U, PV	U, PV	16-132

16.13.1 Module Identification

The module identification register indicates the version of the ADC module that is used in the XMC4[12]00.

ID
Module Identification Register (0008_H) Reset Value: $00C5\ C0XX_H$

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16	15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0
MOD_NUMBER	MOD_TYPE	MOD_REV
r	r	r

Field	Bits	Type	Description
MOD_REV	[7:0]	r	**Module Revision** Indicates the revision number of the implementation. This information depends on the design step.
MOD_TYPE	[15:8]	r	**Module Type** This internal marker is fixed to $C0_H$.
MOD_NUMBER	[31:16]	r	**Module Number** Indicates the module identification number ($00C5_H$ = SARADC).

16.13.2 System Registers

A set of standardized registers provides general access to the module and controls basic system functions.

The Clock Control Register **CLC** allows the programmer to adapt the functionality and power consumption of the module to the requirements of the application. Register **CLC** controls the module clock signal and the reactivity to the sleep mode signal.

CLC
Clock Control Register **(0000_H)** **Reset Value: 0000 0003$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**EDIS**	**0**	**DISS**	**DISR**
r	r	r	r	r	r	r	r	r	r	r	r	rw	r	r	rw

Field	Bits	Type	Description
DISR	0	rw	**Module Disable Request Bit** Used for enable/disable control of the module. Also the analog section is disabled by clearing ANONS. 0_B On request: enable the module clock 1_B Off request: stop the module clock
DISS	1	r	**Module Disable Status Bit** 0_B Module clock is enabled 1_B Off: module is not clocked
0	2	r	**Reserved, write 0, read as 0**
EDIS	3	rw	**Sleep Mode Enable Control** Used to control module's reaction to sleep mode. 0_B Sleep mode request is enabled and functional 1_B Module disregards the sleep mode control signal
0	[31:4]	r	**Reserved, write 0, read as 0**

The OCDS control and status register OCS controls the module's behavior in suspend mode (used for debugging) and includes the module-related control bits for the OCDS Trigger Bus (OTGB).

The OCDS Control and Status (OCS) register is cleared by Debug Reset.
The OCS register can only be written when the OCDS is enabled.
If OCDS is being disabled, the OCS register value will not change.
When OCDS is disabled the OCS suspend control is ineffective.
Write access is 32 bit wide only and requires Supervisor Mode.

OCS

OCDS Control and Status Register (0028_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**SUS STA**	**SUS _P**	**SUS**				**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	rh	w	rw				r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**TG _P**	**TGB**	**TGS**	
r	r	r	r	r	r	r	r	r	r	r	r	w	rw	rw	

Field	Bits	Type	Description
TGS	[1:0]	rw	**Trigger Set for OTGB0/1** 00_B No Trigger Set output 01_B Trigger Set 1: TS16_SSIG, input sample signals 10_B Reserved 11_B Reserved
TGB	2	rw	**OTGB0/1 Bus Select** 0_B Trigger Set is output on OTGB0 1_B Trigger Set is output on OTGB1
TG_P	3	w	**TGS, TGB Write Protection** TGS and TGB are only written when TG_P is 1, otherwise unchanged. Read as 0.
0	[23:4]	r	**Reserved, write 0, read as 0**

Field	Bits	Type	Description
SUS	[27:24]	rw	**OCDS Suspend Control** Controls the sensitivity to the suspend signal coming from the OCDS Trigger Switch (OTGS) 0000_B Will not suspend 0001_B Hard suspend: Clock is switched off immediately. 0010_B Soft suspend mode 0: Stop conversions after the currently running one is completed and its result has been stored. No change for the arbiter. 0011_B Soft suspend mode 1: Stop conversions after the currently running one is completed and its result has been stored. Stop arbiter after the current arbitration round. others: Reserved
SUS_P	28	w	**SUS Write Protection** SUS is only written when SUS_P is 1, otherwise unchanged. Read as 0.
SUSSTA	29	rh	**Suspend State** 0_B Module is not (yet) suspended 1_B Module is suspended
0	[31:30]	r	**Reserved, write 0, read as 0**

Table 16-8 TS16_SSIG Trigger Set VADC

Bits	Name	Description
[3:0]	GxSAMPLE	Input signal sample phase of converter group x (x = 3-0)
[15:4]	0	Reserved

Note: The SAMPLE signals can be used as gate/trigger inputs for the adjacent groups. These outputs are enabled when bitfield TGS = 01_B and bit TGB = 0_B.

16.13.3 General Registers

The global configuration register provides global control and configuration options that are valid for all converters of the cluster.

GLOBCFG
Global Configuration Register (0080_H) Reset Value: $0000\ 000F_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SUCAL	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**DPCAL3**	**DPCAL2**	**DPCAL1**	**DPCAL0**
w	r	r	r	r	r	r	r	r	r	r	r	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DIVWC	**0**	**0**	**0**	**0**	**0**	**DIVD**		**DCMSB**	**0**	**0**	**DIVA**				
w	r	r	r	r	r	rw		rw	r	r	rw				

Field	Bits	Type	Description
DIVA	[4:0]	rw	**Divider Factor for the Analog Internal Clock** Defines the frequency of the basic converter clock f_{ADCI} (base clock for conversion and sample phase). 00_H $f_{ADCI} = f_{ADC} / 2$ 01_H $f_{ADCI} = f_{ADC} / 2$ 02_H $f_{ADCI} = f_{ADC} / 3$ ... $1F_H$ $f_{ADCI} = f_{ADC} / 32$
0	[6:5]	r	**Reserved, write 0, read as 0**
DCMSB	7	rw	**Double Clock for the MSB Conversion** Selects an additional clock cycle for the conversion step of the MSB.[1)] 0_B 1 clock cycles for the MSB (standard) 1_B Reserved
DIVD	[9:8]	rw	**Divider Factor for the Arbiter Clock** Defines the frequency of the arbiter clock f_{ADCD}. 00_B $f_{ADCD} = f_{ADC}$ 01_B $f_{ADCD} = f_{ADC} / 2$ 10_B $f_{ADCD} = f_{ADC} / 3$ 11_B $f_{ADCD} = f_{ADC} / 4$
0	[14:10]	r	**Reserved, write 0, read as 0**

Field	Bits	Type	Description
DIVWC	15	w	**Write Control for Divider Parameters** 0_B No write access to divider parameters 1_B Bitfields DIVA, DCMSB, DIVD can be written
DPCALx (x = 0 - 3)	x+16	rw	**Disable Post-Calibration** 0_B Automatic post-calibration after each conversion of group x 1_B No post-calibration
0	[30:20]	r	**Reserved, write 0, read as 0**
SUCAL	31	w	**Start-Up Calibration** The 0-1 transition of bit SUCAL initiates the start-up calibration phase of all calibrated analog converters. 0_B No action 1_B Initiate the start-up calibration phase (indication in bit GxARBCFG.CAL)

1) Please also refer to section **"Conversion Timing" on Page 16-26**.

GxCHASS (x = 0 - 1)
Channel Assignment Register, Group x

(x * 0400_H + 0488_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**ASSCH7**	**ASSCH6**	**ASSCH5**	**ASSCH4**	**ASSCH3**	**ASSCH2**	**ASSCH1**	**ASSCH0**
r	r	r	r	r	r	r	r	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
ASSCHy (y = 0 - 7)	y	rw	**Assignment for Channel y** 0_B Channel y can be a background channel converted with lowest priority 1_B Channel y is a priority channel within group x
0	[31:8]	r	**Reserved, write 0, read as 0**

16.13.4 Arbitration and Source Registers

The Arbitration Configuration Register selects the timing and the behavior of the arbiter.

GxARBCFG (x = 0 - 1)
Arbitration Configuration Register, Group x

($x * 0400_H + 0480_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SAMPLE	BUSY	0	CAL	0	0	0	0	0	0	0	0	0	0	ANONS	
rh	rh	r	rh	r	r	r	r	r	r	r	r	r	r	rh	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	ARBM	0	ARBRND		0	0	ANONC	
r	r	r	r	r	r	r	r	rw	r	rw		r	r	rw	

Field	Bits	Type	Description
ANONC	[1:0]	rw	**Analog Converter Control** Defines the value of bitfield ANONS in a stand-alone converter or a converter in master mode. Coding see ANONS or **Section 16.4**.
0	[3:2]	r	**Reserved, write 0, read as 0**
ARBRND	[5:4]	rw	**Arbitration Round Length** Defines the number of arbitration slots per arb. round (arbitration round length = t_{ARB}).[1)] 00_B 4 arbitration slots per round ($t_{ARB} = 4 / f_{ADCD}$) 01_B 8 arbitration slots per round ($t_{ARB} = 8 / f_{ADCD}$) 10_B 16 arbitration slots per round ($t_{ARB} = 16 / f_{ADCD}$) 11_B 20 arbitration slots per round ($t_{ARB} = 20 / f_{ADCD}$)
0	6	r	**Reserved, write 0, read as 0**

Field	Bits	Type	Description
ARBM	7	rw	**Arbitration Mode** 0_B The arbiter runs permanently. This setting is required for a synchronization slave (see **Section 16.9.1**) and for equidistant sampling using the signal ARBCNT (see **Section 16.9.2**). 1_B The arbiter only runs if at least one conversion request of an enabled request source is pending. This setting ensures a reproducible latency from an incoming request to the conversion start, if the converter is idle. Synchronized conversions are not supported.
0	[15:8]	r	**Reserved, write 0, read as 0**
ANONS	[17:16]	rh	**Analog Converter Control Status** Defined by bitfield ANONC in a stand-alone kernel or a kernel in master mode. In slave mode, this bitfield is defined by bitfield ANONC of the respective master kernel. See also **Section 16.4**. 00_B Analog converter off 01_B Reserved 10_B Reserved 11_B Normal operation (permanently on)
0	[27:18]	r	**Reserved, write 0, read as 0**
CAL	28	rh	**Start-Up Calibration Active Indication** Indicates the start-up calibration phase of the corresponding analog converter. 0_B Completed or not yet started 1_B Start-up calibration phase is active *Note: Start conversions only after the start-up calibration phase is complete.*
0	29	r	**Reserved, write 0, read as 0**
BUSY	30	rh	**Converter Busy Flag** 0_B Not busy 1_B Converter is busy with a conversion

Field	Bits	Type	Description
SAMPLE	31	rh	**Sample Phase Flag** 0_B Converting or idle 1_B Input signal is currently sampled

1) The default setting of 4 arbitration slots is sufficient for correct arbitration. The duration of an arbitration round can be increased if required to synchronize requests.

The Arbitration Priority Register defines the request source priority and the conversion start mode for each request source.

Note: Only change priority and conversion start mode settings of a request source while this request source is disabled, and a currently running conversion requested by this source is finished.

GxARBPR (x = 0 - 1)
Arbitration Priority Register, Group x

(x * 0400_H + 0484_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**ASEN2**	**ASEN1**	**ASEN0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	rw	rw	rw	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**CSM2**	**0**	**PRIO2**		**CSM1**	**0**	**PRIO1**		**CSM0**	**0**	**PRIO0**	
r	r	r	r	rw	r	rw		rw	r	rw		rw	r	rw	

Field	Bits	Type	Description
PRIO0, **PRIO1,** **PRIO2**	[1:0], [5:4], [9:8]	rw	**Priority of Request Source x** Arbitration priority of request source x (in slot x) 00_B Lowest priority is selected. ... 11_B Highest priority is selected.
CSM0, **CSM1,** **CSM2**	3, 7, 11	rw	**Conversion Start Mode of Request Source x** 0_B Wait-for-start mode 1_B Cancel-inject-repeat mode, i.e. this source can cancel conversion of other sources.

Field	Bits	Type	Description
0	2, 6, 10, [23:12]	r	**Reserved, write 0, read as 0**
ASENy (y = 0 - 2)	24 + y	rw	**Arbitration Slot y Enable** Enables the associated arbitration slot of an arbiter round. The request source bits are not modified by write actions to ASENR. 0_B The corresponding arbitration slot is disabled and considered as empty. Pending conversion requests from the associated request source are disregarded. 1_B The corresponding arbitration slot is enabled. Pending conversion requests from the associated request source are arbitrated.
0	[31:27]	r	**Reserved, write 0, read as 0**

The control register of the queue source selects the external gate and/or trigger signals. Write control bits allow separate control of each function with a simple write access.

GxQCTRL0 (x = 0 - 1)
Queue 0 Source Control Register, Group x
($x * 0400_H + 0500_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TM WC	**0**	**0**	**TM EN**	**0**	**0**	**0**	**0**	**GT WC**	**0**	**0**	**GT LVL**	**GT SEL**			
w	r	r	rw	r	r	r	r	w	r	r	rh	rw			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
XT WC	**XT MODE**		**XT LVL**	**XT SEL**				**0**	**0**	**0**	**0**	**SRCRESREG**			
w	rw		rh	rw				r	r	r	r	rw			

Field	Bits	Type	Description
SRCRESREG	[3:0]	rw	**Source-specific Result Register** 0000_B Use GxCHCTRy.RESREG to select a group result register 0001_B Store result in group result register GxRES1 … 1111_B Store result in group result register GxRES15
0	[7:4]	r	**Reserved, write 0, read as 0**
XTSEL	[11:8]	rw	**External Trigger Input Selection** The connected trigger input signals are listed in **Table 16-13 “Digital Connections in the XMC4[12]00” on Page 16-136** *Note: XTSEL = 1111_B uses the selected gate input as trigger source (ENGT must be $0X_B$).*
XTLVL	12	rh	**External Trigger Level** Current level of the selected trigger input
XTMODE	[14:13]	rw	**Trigger Operating Mode** 00_B No external trigger 01_B Trigger event upon a falling edge 10_B Trigger event upon a rising edge 11_B Trigger event upon any edge

Field	Bits	Type	Description
XTWC	15	w	**Write Control for Trigger Configuration** 0_B No write access to trigger configuration 1_B Bitfields XTMODE and XTSEL can be written
GTSEL	[19:16]	rw	**Gate Input Selection** The connected gate input signals are listed in **Table 16-13 “Digital Connections in the XMC4[12]00” on Page 16-136**
GTLVL	20	rh	**Gate Input Level** Current level of the selected gate input
0	[22:21]	r	**Reserved, write 0, read as 0**
GTWC	23	w	**Write Control for Gate Configuration** 0_B No write access to gate configuration 1_B Bitfield GTSEL can be written
0	[27:24]	r	**Reserved, write 0, read as 0**
TMEN	28	rw	**Timer Mode Enable** 0_B No timer mode: standard gating mechanism can be used 1_B Timer mode for equidistant sampling enabled: standard gating mechanism must be disabled
0	[30:29]	r	**Reserved, write 0, read as 0**
TMWC	31	w	**Write Control for Timer Mode** 0_B No write access to timer mode 1_B Bitfield TMEN can be written

The Queue Mode Register configures the operating mode of a queued request source.

GxQMR0 (x = 0 - 1)
Queue 0 Mode Register, Group x

($x * 0400_H + 0504_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**RPT DIS**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**CEV**	**FLU SH**	**TR EV**	**CLR V**	**0**	**0**	**0**	**0**	**0**	**EN TR**	**ENGT**	
r	r	r	r	w	w	w	w	r	r	r	r	r	rw	rw	

Field	Bits	Type	Description
ENGT	[1:0]	rw	**Enable Gate** Selects the gating functionality for source 0/2. 00_B No conversion requests are issued 01_B Conversion requests are issued if a valid conversion request is pending in the queue 0 register or in the backup register 10_B Conversion requests are issued if a valid conversion request is pending in the queue 0 register or in the backup register and REQGTx = 1 11_B Conversion requests are issued if a valid conversion request is pending in the queue 0 register or in the backup register and REQGTx = 0 *Note: REQGTx is the selected gating signal.*
ENTR	2	rw	**Enable External Trigger** 0_B External trigger disabled 1_B The selected edge at the selected trigger input signal REQTR generates the trigger event
0	[7:3]	r	**Reserved, write 0, read as 0**

Field	Bits	Type	Description
CLRV	8	w	**Clear Valid Bit** 0_B No action 1_B The next pending valid queue entry in the sequence and the event flag EV are cleared. If there is a valid entry in the queue backup register (QBUR.V = 1), this entry is cleared, otherwise the entry in queue register 0 is cleared.
TREV	9	w	**Trigger Event** 0_B No action 1_B Generate a trigger event by software
FLUSH	10	w	**Flush Queue** 0_B No action 1_B Clear all queue entries (including backup stage) and the event flag EV. The queue contains no more valid entry.
CEV	11	w	**Clear Event Flag** 0_B No action 1_B Clear bit EV
0	[15:12]	r	**Reserved, write 0, read as 0**
RPTDIS	16	rw	**Repeat Disable** 0_B A cancelled conversion is repeated 1_B A cancelled conversion is discarded
0	[31:17]	r	**Reserved, write 0, read as 0**

The Queue Status Register indicates the current status of the queued source. The filling level and the empty information refer to the queue intermediate stages (if available) and to the queue register 0. An aborted conversion stored in the backup stage is not indicated by these bits (therefore, see QBURx.V).

GxQSR0 (x = 0 - 1)
Queue 0 Status Register, Group x

($x * 0400_H + 0508_H$) **Reset Value: $0000\ 0020_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**EV**	**REQ GT**	**0**	**EMP TY**	**0**	**FILL**			
r	r	r	r	r	r	r	rh	rh	r	rh	r	rh			

Field	Bits	Type	Description
FILL	[3:0]	rh	**Filling Level for Queue 2** Indicates the number of valid queue entries. It is incremented each time a new entry is written to QINRx or by an enabled refill mechanism. It is decremented each time a requested conversion has been started. A new entry is ignored if the filling level has reached its maximum value. 0000_B There is 1 (if EMPTY = 0) or no (if EMPTY = 1) valid entry in the queue 0001_B There are 2 valid entries in the queue 0010_B There are 3 valid entries in the queue … 0111_B There are 8 valid entries in the queue others: Reserved
0	4	r	**Reserved, write 0, read as 0**
EMPTY	5	rh	**Queue Empty** 0_B There are valid entries in the queue (see FILL) 1_B No valid entries (queue is empty)
0	6	r	**Reserved, write 0, read as 0**

Field	Bits	Type	Description
REQGT	7	rh	**Request Gate Level** Monitors the level at the selected REQGT input. 0_B The gate input is low 1_B The gate input is high
EV	8	rh	**Event Detected** Indicates that an event has been detected while at least one valid entry has been in the queue (queue register 0 or backup stage). Once set, this bit is cleared automatically when the requested conversion is started. 0_B No trigger event 1_B A trigger event has been detected
0	[31:9]	r	**Reserved, write 0, read as 0**

The Queue Input Register is the entry point for conversion requests of a queued request source.

GxQINR0 (x = 0 - 1)
Queue 0 Input Register, Group x

($x * 0400_H + 0510_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**EXTR**	**ENSI**	**RF**	**REQCHNR**				
r	r	r	r	r	r	r	r	w	w	w			w		

Field	Bits	Type	Description
REQCHNR	[4:0]	w	**Request Channel Number** Defines the channel number to be converted
RF	5	w	**Refill** 0_B No refill: this queue entry is converted once and then invalidated 1_B Automatic refill: this queue entry is automatically reloaded into QINRx when the related conversion is started
ENSI	6	w	**Enable Source Interrupt** 0_B No request source interrupt 1_B A request source event interrupt is generated upon a request source event (related conversion is finished)
EXTR	7	w	**External Trigger** Enables the external trigger functionality. 0_B A valid queue entry immediately leads to a conversion request. 1_B A valid queue entry waits for a trigger event to occur before issuing a conversion request.
0	[31:8]	r	**Reserved, write 0, read as 0**

Note: Registers QINRx share addresses with registers QBURx.
Write operations target the control bits in register QINRx. Read operations return the status bits from register QBURx.

The queue registers 0 monitor the status of the pending request (queue stage 0).

GxQ0R0 (x = 0 - 1)
Queue 0 Register 0, Group x (x * 0400_H + $050C_H$) Reset Value: 0000 0000_H

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**V**	**EXTR**	**ENSI**	**RF**	**REQCHNR**				
r	r	r	r	r	r	r	rh	rh	rh	rh	rh				

Field	Bits	Type	Description
REQCHNR	[4:0]	rh	**Request Channel Number** Stores the channel number to be converted.
RF	5	rh	**Refill** Selects the handling of handled requests. 0_B The request is discarded after the conversion start. 1_B The request is automatically refilled into the queue after the conversion start.
ENSI	6	rh	**Enable Source Interrupt** 0_B No request source interrupt 1_B A request source event interrupt is generated upon a request source event (related conversion is finished)
EXTR	7	rh	**External Trigger** Enables external trigger events. 0_B A valid queue entry immediately leads to a conversion request 1_B The request handler waits for a trigger event
V	8	rh	**Request Channel Number Valid** Indicates a valid queue entry in queue register 0. 0_B No valid queue entry 1_B The queue entry is valid and leads to a conversion request

Field	Bits	Type	Description
0	[31:9]	r	**Reserved, write 0, read as 0**

The Queue Backup Registers monitor the status of an aborted queued request.

GxQBUR0 (x = 0 - 1)
Queue 0 Backup Register, Group x

($x * 0400_H + 0510_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**V**	**EXTR**	**ENSI**	**RF**	**REQCHNR**				
r	r	r	r	r	r	r	rh	rh	rh	rh	rh				

Field	Bits	Type	Description
REQCHNR	[4:0]	rh	**Request Channel Number** The channel number of the aborted conversion that has been requested by this request source
RF	5	rh	**Refill** The refill control bit of the aborted conversion
ENSI	6	rh	**Enable Source Interrupt** The enable source interrupt control bit of the aborted conversion
EXTR	7	rh	**External Trigger** The external trigger control bit of the aborted conversion
V	8	rh	**Request Channel Number Valid** Indicates if the entry (REQCHNR, RF, TR, ENSI) in the queue backup register is valid. Bit V is set when a running conversion (that has been requested by this request source) is aborted, it is cleared when the aborted conversion is restarted. 0_B Backup register not valid 1_B Backup register contains a valid entry. This will be requested before a valid entry in queue register 0 (stage 0) will be requested.
0	[31:9]	r	**Reserved, write 0, read as 0**

Note: Registers QBURx share addresses with registers QINRx.
Read operations return the status bits from register QBURx. Write operations target the control bits in register QINRx.

Registers of Group Scan Source

There is a separate register set for each group scan source. These sources can be operated independently.

The control register of the autoscan source selects the external gate and/or trigger signals.
Write control bits allow separate control of each function with a simple write access.

GxASCTRL (x = 0 - 1)
Autoscan Source Control Register, Group x
(x * 0400_H + 0520_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TM WC	0	0	TM EN	0	0	0	0	GT WC	0	0	GT LVL	GT SEL			
w	r	r	rw	r	r	r	r	w	r	r	rh	rw			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
XT WC	XT MODE		XT LVL	XT SEL				0	0	0	0	SRCRESREG			
w	rw		rh	rw				r	r	r	r	rw			

Field	Bits	Type	Description
SRCRESREG	[3:0]	rw	**Source-specific Result Register** 0000_B Use GxCHCTRy.RESREG to select a group result register 0001_B Store result in group result register GxRES1 … 1111_B Store result in group result register GxRES15
0	[7:4]	r	**Reserved, write 0, read as 0**
XTSEL	[11:8]	rw	**External Trigger Input Selection** The connected trigger input signals are listed in **Table 16-13 “Digital Connections in the XMC4[12]00” on Page 16-136** *Note: XTSEL = 1111_B uses the selected gate input as trigger source (ENGT must be $0X_B$).*
XTLVL	12	rh	**External Trigger Level** Current level of the selected trigger input

Field	Bits	Type	Description
XTMODE	[14:13]	rw	**Trigger Operating Mode** 00_B No external trigger 01_B Trigger event upon a falling edge 10_B Trigger event upon a rising edge 11_B Trigger event upon any edge
XTWC	15	w	**Write Control for Trigger Configuration** 0_B No write access to trigger configuration 1_B Bitfields XTMODE and XTSEL can be written
GTSEL	[19:16]	rw	**Gate Input Selection** The connected gate input signals are listed in **Table 16-13 “Digital Connections in the XMC4[12]00” on Page 16-136**
GTLVL	20	rh	**Gate Input Level** Current level of the selected gate input
0	[22:21]	r	**Reserved, write 0, read as 0**
GTWC	23	w	**Write Control for Gate Configuration** 0_B No write access to gate configuration 1_B Bitfield GTSEL can be written
0	[27:24]	r	**Reserved, write 0, read as 0**
TMEN	28	rw	**Timer Mode Enable** 0_B No timer mode: standard gating mechanism can be used 1_B Timer mode for equidistant sampling enabled: standard gating mechanism must be disabled
0	[30:29]	r	**Reserved, write 0, read as 0**
TMWC	31	w	**Write Control for Timer Mode** 0_B No write access to timer mode 1_B Bitfield TMEN can be written

The Conversion Request Mode Register configures the operating mode of the channel scan request source.

GxASMR (x = 0 - 1)
Autoscan Source Mode Register, Group x

($x * 0400_H + 0524_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**RPT DIS**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**LD EV**	**CLR PND**	**REQ GT**	**0**	**LDM**	**SCA N**	**EN SI**	**EN TR**	**ENGT**	
r	r	r	r	r	r	w	w	rh	r	rw	rw	rw	rw	rw	

Field	Bits	Type	Description
ENGT	[1:0]	rw	**Enable Gate** Selects the gating functionality for source 1. 00_B No conversion requests are issued 01_B Conversion requests are issued if at least one pending bit is set 10_B Conversion requests are issued if at least one pending bit is set and REQGTx = 1. 11_B Conversion requests are issued if at least one pending bit is set and REQGTx = 0. *Note: REQGTx is the selected gating signal.*
ENTR	2	rw	**Enable External Trigger** 0_B External trigger disabled 1_B The selected edge at the selected trigger input signal REQTR generates the load event
ENSI	3	rw	**Enable Source Interrupt** 0_B No request source interrupt 1_B A request source interrupt is generated upon a request source event (last pending conversion is finished)

Field	Bits	Type	Description
SCAN	4	rw	**Autoscan Enable** 0_B No autoscan 1_B Autoscan functionality enabled: a request source event automatically generates a load event
LDM	5	rw	**Autoscan Source Load Event Mode** 0_B Overwrite mode: Copy all bits from the select registers to the pending registers upon a load event 1_B Combine mode: Set all pending bits that are set in the select registers upon a load event (logic OR)
0	6	r	**Reserved, write 0, read as 0**
REQGT	7	rh	**Request Gate Level** Monitors the level at the selected REQGT input. 0_B The gate input is low 1_B The gate input is high
CLRPND	8	w	**Clear Pending Bits** 0_B No action 1_B The bits in register GxASPNDx are cleared
LDEV	9	w	**Generate Load Event** 0_B No action 1_B A load event is generated
0	[15:10]	r	**Reserved, write 0, read as 0**
RPTDIS	16	rw	**Repeat Disable** 0_B A cancelled conversion is repeated 1_B A cancelled conversion is discarded
0	[31:17]	r	**Reserved, write 0, read as 0**

The Channel Select Register selects the channels to be converted by the group scan request source. Its bits are used to update the pending register, when a load event occurs.

The number of valid channel bits depends on the channels available in the respective product type (please refer to **“Product-Specific Configuration” on Page 16-133**).

GxASSEL (x = 0 - 1)
Autoscan Source Channel Select Register, Group x
($x * 0400_H + 0528_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**CHSEL7**	**CHSEL6**	**CHSEL5**	**CHSEL4**	**CHSEL3**	**CHSEL2**	**CHSEL1**	**CHSEL0**
r	r	r	r	r	r	r	r	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
CHSELy (y = 0 - 7)	y	rwh	**Channel Selection** Each bit (when set) enables the corresponding input channel of the respective group to take part in the scan sequence. 0_B Ignore this channel 1_B This channel is part of the scan sequence
0	[31:8]	r	**Reserved, write 0, read as 0**

Note: Register GxASSEL is also updated when writing a pattern to register GxASPND.

The Channel Pending Register indicates the channels to be converted in the current conversion sequence. They are updated from the select register, when a load event occurs.

GxASPND (x = 0 - 1)
Autoscan Source Pending Register, Group x
(x * 0400_H + $052C_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**CHPND7**	**CHPND6**	**CHPND5**	**CHPND4**	**CHPND3**	**CHPND2**	**CHPND1**	**CHPND0**
r	r	r	r	r	r	r	r	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
CHPNDy (y = 0 - 7)	y	rwh	**Channels Pending** Each bit (when set) request the conversion of the corresponding input channel of the respective group. 0_B Ignore this channel 1_B Request conversion of this channel
0	[31:8]	r	**Reserved, write 0, read as 0**

Note: Writing to register GxASPND automatically updates register GxASSEL.

Registers of Background Scan Source

There is a single register set for the background scan source. This source is common for the complete VADC.

The control register of the background request source selects the external gate and/or trigger signals.
Write control bits allow separate control of each function with a simple write access.

BRSCTRL
Background Request Source Control Register
(0200_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	0	0	0	0	0	0	0	GTWC	0	0	GTLVL	GTSEL			
r	r	r	r	r	r	r	r	w	r	r	rh	rw			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
XTWC	XTMODE		XTLVL	XTSEL				0	0	0	0	SRCRESREG			
w	rw		rh	rw				r	r	r	r	rw			

Field	Bits	Type	Description
SRCRESREG	[3:0]	rw	**Source-specific Result Register** 0000_B Use GxCHCTRy.RESREG to select a group result register 0001_B Store result in group result register GxRES1 … 1111_B Store result in group result register GxRES15
0	[7:4]	r	**Reserved, write 0, read as 0**
XTSEL	[11:8]	rw	**External Trigger Input Selection** The connected trigger input signals are listed in **Table 16-13 “Digital Connections in the XMC4[12]00” on Page 16-136** *Note: XTSEL = 1111_B uses the selected gate input as trigger source (ENGT must be $0X_B$).*
XTLVL	12	rh	**External Trigger Level** Current level of the selected trigger input

Field	Bits	Type	Description
XTMODE	[14:13]	rw	**Trigger Operating Mode** 00_B No external trigger 01_B Trigger event upon a falling edge 10_B Trigger event upon a rising edge 11_B Trigger event upon any edge
XTWC	15	w	**Write Control for Trigger Configuration** 0_B No write access to trigger configuration 1_B Bitfields XTMODE and XTSEL can be written
GTSEL	[19:16]	rw	**Gate Input Selection** The connected gate input signals are listed in **Table 16-13 “Digital Connections in the XMC4[12]00” on Page 16-136**
GTLVL	20	rh	**Gate Input Level** Current level of the selected gate input
0	[22:21]	r	**Reserved, write 0, read as 0**
GTWC	23	w	**Write Control for Gate Configuration** 0_B No write access to gate configuration 1_B Bitfield GTSEL can be written
0	[31:24]	r	**Reserved, write 0, read as 0**

The Conversion Request Mode Register configures the operating mode of the background request source.

BRSMR
Background Request Source Mode Register
(0204_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	RPT DIS
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	LD EV	CLR PND	REQ GT	0	LDM	SCA N	EN SI	EN TR	ENGT	
r	r	r	r	r	r	w	w	rh	r	rw	rw	rw	rw	rw	

Field	Bits	Type	Description
ENGT	[1:0]	rw	**Enable Gate** Selects the gating functionality for source 1. 00_B No conversion requests are issued 01_B Conversion requests are issued if at least one pending bit is set 10_B Conversion requests are issued if at least one pending bit is set and REQGTx = 1. 11_B Conversion requests are issued if at least one pending bit is set and REQGTx = 0. *Note: REQGTx is the selected gating signal.*
ENTR	2	rw	**Enable External Trigger** 0_B External trigger disabled 1_B The selected edge at the selected trigger input signal REQTR generates the load event
ENSI	3	rw	**Enable Source Interrupt** 0_B No request source interrupt 1_B A request source interrupt is generated upon a request source event (last pending conversion is finished)

Field	Bits	Type	Description
SCAN	4	rw	**Autoscan Enable** 0_B No autoscan 1_B Autoscan functionality enabled: a request source event automatically generates a load event
LDM	5	rw	**Autoscan Source Load Event Mode** 0_B Overwrite mode: Copy all bits from the select registers to the pending registers upon a load event 1_B Combine mode: Set all pending bits that are set in the select registers upon a load event (logic OR)
0	6	r	**Reserved, write 0, read as 0**
REQGT	7	rh	**Request Gate Level** Monitors the level at the selected REQGT input. 0_B The gate input is low 1_B The gate input is high
CLRPND	8	w	**Clear Pending Bits** 0_B No action 1_B The bits in registers BRSPNDx are cleared
LDEV	9	w	**Generate Load Event** 0_B No action 1_B A load event is generated
0	[15:10]	r	**Reserved, write 0, read as 0**
RPTDIS	16	rw	**Repeat Disable** 0_B A cancelled conversion is repeated 1_B A cancelled conversion is discarded
0	[31:17]	r	**Reserved, write 0, read as 0**

The Channel Select Registers select the channels to be converted by the background request source (channel scan source). Its bits are used to update the pending registers, when a load event occurs.

The number of valid channel bits depends on the channels available in the respective product type (please refer to **"Product-Specific Configuration" on Page 16-133**).

*Note: Priority channels selected in registers **GxCHASS (x = 0 - 1)** will not be converted.*

BRSSELx (x = 0 - 1)
Background Request Source Channel Select Register, Group x
(0180_H + x * 0004_H) **Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**CHSELG7**	**CHSELG6**	**CHSELG5**	**CHSELG4**	**CHSELG3**	**CHSELG2**	**CHSELG1**	**CHSELG0**
r	r	r	r	r	r	r	r	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
CHSELGy (y = 0 - 7)	y	rwh	**Channel Selection Group x** Each bit (when set) enables the corresponding input channel of the respective group to take part in the background scan sequence. 0_B Ignore this channel 1_B This channel is part of the scan sequence
0	[31:8]	r	**Reserved, write 0, read as 0**

The Channel Pending Registers indicate the channels to be converted in the current conversion sequence. They are updated from the select registers, when a load event occurs.

BRSPNDx (x = 0 - 1)
Background Request Source Pending Register, Group x

($01C0_H$ + x * 0004_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**CHPNDG7**	**CHPNDG6**	**CHPNDG5**	**CHPNDG4**	**CHPNDG3**	**CHPNDG2**	**CHPNDG1**	**CHPNDG0**
r	r	r	r	r	r	r	r	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
CHPNDGy (y = 0 - 7)	y	rwh	**Channels Pending Group x** Each bit (when set) request the conversion of the corresponding input channel of the respective group. 0_B Ignore this channel 1_B Request conversion of this channel
0	[31:8]	r	**Reserved, write 0, read as 0**

Note: Writing to any of registers BRSPNDx automatically updates the corresponding register BRSSELx and generates a load event that copies all bits from all registers BRSSELx to BRSPNDx.
Use this shortcut only when writing the last word of the request pattern.

16.13.5 Channel Control Registers

G0CHCTRy (y = 0 - 7)
Group 0, Channel y Ctrl. Reg. (0600_H + y * 0004_H) Reset Value: 0000 0000_H
G1CHCTRy (y = 0 - 7)
Group 1, Channel y Ctrl. Reg. ($0A00_H$ + y * 0004_H) Reset Value: 0000 0000_H
G2CHCTRy (y = 0 - 7)
Group 2, Channel y Ctrl. Reg. ($0E00_H$ + y * 0004_H) Reset Value: 0000 0000_H
G3CHCTRy (y = 0 - 7)
Group 3, Channel y Ctrl. Reg. (1200_H + y * 0004_H) Reset Value: 0000 0000_H

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	BWDEN	BWDCH		0	0	0	0	0	0	RESPOS	RESTBS	RESREG			
r	rw	rw		r	r	r	r	r	r	rw	rw	rw			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	REFSEL	SYNC	CHEVMODE		BNDSELU		BNDSELL		0	0	ICLSEL	
r	r	r	r	rw	rw	rw		rw		rw		r	r	rw	

Field	Bits	Type	Description
ICLSEL	[1:0]	rw	**Input Class Select** 00_B Use group-specific class 0 01_B Use group-specific class 1 10_B Use global class 0 11_B Use global class 1
0	[3:2]	r	**Reserved, write 0, read as 0**
BNDSELL	[5:4]	rw	**Lower Boundary Select** 00_B Use group-specific boundary 0 01_B Use group-specific boundary 1 10_B Use global boundary 0 11_B Use global boundary 1
BNDSELU	[7:6]	rw	**Upper Boundary Select** 00_B Use group-specific boundary 0 01_B Use group-specific boundary 1 10_B Use global boundary 0 11_B Use global boundary 1

Field	Bits	Type	Description
CHEVMODE	[9:8]	rw	**Channel Event Mode** Generate a channel event either in normal compare mode (NCM) with limit checking[1)] or in Fast Compare Mode (FCM)[2)] 00_B Never 01_B NCM: If result is inside the boundary band FCM: If result becomes high (above cmp. val.) 10_B NCM: If result is outside the boundary band FCM: If result becomes low (below cmp. val.) 11_B NCM: Always (ignore band) FCM: If result switches to either level
SYNC	10	rw	**Synchronization Request** 0_B No synchroniz. request, standalone operation 1_B Request a synchronized conversion of this channel (only taken into account for a master)
REFSEL	11	rw	**Reference Input Selection** Defines the reference voltage input to be used for conversions on this channel. 0_B Standard reference input V_{AREF} 1_B Alternate reference input from CH0[3)]
0	[15:12]	rw	**Reserved, write 0, read as 0**
RESREG	[19:16]	rw	**Result Register** 0000_BStore result in group result register GxRES0 … 1111_BStore result in group result register GxRES15
RESTBS	20	rw	**Result Target for Background Source** 0_B Store results in the selected group result register 1_B Store results in the global result register
RESPOS	21	rw	**Result Position** 0_B Store results left-aligned 1_B Store results right-aligned
0	[27:22]	r	**Reserved, write 0, read as 0**
BWDCH	[29:28]	rw	**Broken Wire Detection Channel** 00_B Select V_{AGND} 01_B Select V_{AREF} 10_B Reserved 11_B Reserved

Field	Bits	Type	Description
BWDEN	30	rw	**Broken Wire Detection Enable** 0_B Normal operation 1_B Additional preparation phase is enabled
0	31	r	**Reserved, write 0, read as 0**

1) The boundary band is defined as the area where the result is less than or equal to the selected upper boundary and greater than or equal to the selected lower boundary, see **Section 16.7.5**.

2) The result is bit FCR in the selected result register.

3) Some channels cannot select an alternate reference.

GxICLASS0 (x = 0 - 1)
Input Class Register 0, Group x

($x * 0400_H + 04A0_H$) **Reset Value: 0000 0000$_H$**

GxICLASS1 (x = 0 - 1)
Input Class Register 1, Group x

($x * 0400_H + 04A4_H$) **Reset Value: 0000 0000$_H$**

GLOBICLASSy (y = 0 - 1)
Input Class Register y, Global

($00A0_H + y * 0004_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**CME**			**0**	**0**	**0**	**STCE**				
r	r	r	r	r	rw			r	r	r	rw				

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**CMS**			**0**	**0**	**0**	**STCS**				
r	r	r	r	r	rw			r	r	r	rw				

Field	Bits	Type	Description
STCS	[4:0]	rw	**Sample Time Control for Standard Conversions** Number of additional clock cycles to be added to the minimum sample phase of 2 analog clock cycles: Coding and resulting sample time see **Table 16-9**. For conversions of external channels, the value from bitfield STCE can be used.
0	[7:5]	r	**Reserved, write 0, read as 0**

Field	Bits	Type	Description
CMS	[10:8]	rw	**Conversion Mode for Standard Conversions** 000_B 12-bit conversion 001_B 10-bit conversion 010_B 8-bit conversion 011_B Reserved 100_B Reserved 101_B 10-bit fast compare mode 110_B Reserved 111_B Reserved
0	[15:11]	r	**Reserved, write 0, read as 0**
STCE	[20:16]	rw	**Sample Time Control for EMUX Conversions** Number of additional clock cycles to be added to the minimum sample phase of 2 analog clock cycles: Coding and resulting sample time see **Table 16-9**. For conversions of standard channels, the value from bitfield STCS is used.
0	[23:21]	r	**Reserved, write 0, read as 0**
CME	[26:24]	rw	**Conversion Mode for EMUX Conversions** 000_B 12-bit conversion 001_B 10-bit conversion 010_B 8-bit conversion 011_B Reserved 100_B Reserved 101_B 10-bit fast compare mode 110_B Reserved 111_B Reserved
0	[31:27]	r	**Reserved, write 0, read as 0**

Table 16-9 Sample Time Coding

STCS / STCE	Additional Clock Cycles	Sample Time
$0\ 0000_B$	0	2 / f_{ADCI}
$0\ 0001_B$	1	3 / f_{ADCI}
…	…	…
$0\ 1111_B$	15	17 / f_{ADCI}
$1\ 0000_B$	16	18 / f_{ADCI}
$1\ 0001_B$	32	34 / f_{ADCI}

Table 16-9 Sample Time Coding (cont'd)

STCS / STCE	Additional Clock Cycles	Sample Time
…	…	…
$1\ 1110_B$	240	242 / f_{ADCI}
$1\ 1111_B$	256	258 / f_{ADCI}

16.13.6 Result Registers

The group result control registers select the behavior of the result registers of a given group.

G0RCRy (y = 0 - 15)
Group 0 Result Control Reg. y (0680$_H$ + y * 0004$_H$) Reset Value: 0000 0000$_H$
G1RCRy (y = 0 - 15)
Group 1 Result Control Reg. y (0A80$_H$ + y * 0004$_H$) Reset Value: 0000 0000$_H$
G2RCRy (y = 0 - 15)
Group 2 Result Control Reg. y (0E80$_H$ + y * 0004$_H$) Reset Value: 0000 0000$_H$
G3RCRy (y = 0 - 15)
Group 3 Result Control Reg. y (1280$_H$ + y * 0004$_H$) Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SRG EN	**0**	**0**	**0**	**0**	**FEN**		**WFR**	**0**	**0**	**DMM**		**DRCTR**			
rw	r	r	r	r	rw		rw	r	r	rw		rw			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Field	Bits	Type	Description
0	[15:0]	r	**Reserved, write 0, read as 0**
DRCTR	[19:16]	rw	**Data Reduction Control** Defines how result values are stored/accumulated in this register for the final result. The data reduction counter DRC can be loaded from this bitfield. The function of bitfield DRCTR is determined by bitfield DMM.
DMM	[21:20]	rw	**Data Modification Mode** 00_B Standard data reduction (accumulation) 01_B Result filtering mode[1)] 10_B Difference mode 11_B Reserved See **"Data Modification" on Page 16-40**
0	[23:22]	r	**Reserved, write 0, read as 0**

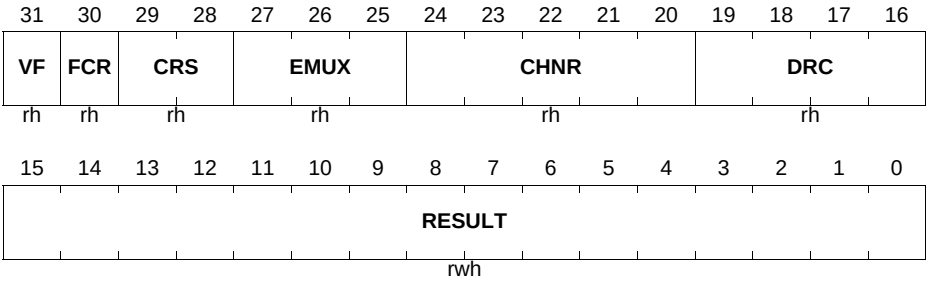
Field	Bits	Type	Description
WFR	24	rw	**Wait-for-Read Mode Enable** 0_B Overwrite mode 1_B Wait-for-read mode enabled for this register
FEN	[26:25]	rw	**FIFO Mode Enable** 00_B Separate result register 01_B Part of a FIFO structure: copy each new valid result 10_B Maximum mode: copy new result if bigger 11_B Minimum mode: copy new result if smaller
0	[30:27]	r	**Reserved, write 0, read as 0**
SRGEN	31	rw	**Service Request Generation Enable** 0_B No service request 1_B Service request after a result event

1) The filter registers are cleared while bitfield DMM ≠ 01_B.

The group result registers provide a selectable storage location for all channels of a given group.

Note: The preset value used in fast compare mode is written to the respective result register. The debug result registers are not writable.

G0RESy (y = 0 - 15)
Group 0 Result Register y **(0700_H + y * 0004_H)** **Reset Value: $0000\ 0000_H$**
G1RESy (y = 0 - 15)
Group 1 Result Register y **($0B00_H$ + y * 0004_H)** **Reset Value: $0000\ 0000_H$**
G2RESy (y = 0 - 15)
Group 2 Result Register y **($0F00_H$ + y * 0004_H)** **Reset Value: $0000\ 0000_H$**
G3RESy (y = 0 - 15)
Group 3 Result Register y **(1300_H + y * 0004_H)** **Reset Value: $0000\ 0000_H$**

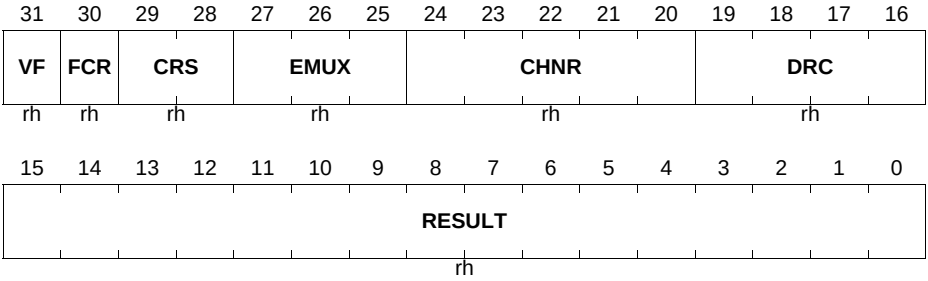


Field	Bits	Type	Description
RESULT	[15:0]	rwh	**Result of Most Recent Conversion** The position of the result bits within this bitfield depends on the configured operating mode. Please, refer to **Section 16.8.2**.
DRC	[19:16]	rh	**Data Reduction Counter** Indicates the number of values still to be accumulated for the final result. The final result is available and valid flag VF is set when bitfield DRC becomes zero (by decrementing or by reload). See **"Data Modification" on Page 16-40**
CHNR	[24:20]	rh	**Channel Number** Indicates the channel number corresponding to the value in bitfield RESULT.

Field	Bits	Type	Description
EMUX	[27:25]	rh	**External Multiplexer Setting** Indicates the setting of the external multiplexer, corresponding to the value in bitfield RESULT. *Note: Available in GxRES0 only. Use GxRES0 if EMUX information is required.*
CRS	[29:28]	rh	**Converted Request Source** Indicates the request source that as requested the conversion to which the result value in bitfield RESULT belongs. 00_B Request source 0 01_B Request source 1 10_B Request source 2 11_B Reserved
FCR	30	rh	**Fast Compare Result** Indicates the result of an operation in Fast Compare Mode. 0_B Signal level was below compare value 1_B Signal level was above compare value
VF	31	rh	**Valid Flag** Indicates a new result in bitfield RESULT or bit FCR. 0_B No new result available 1_B Bitfield RESULT has been updated with new result value and has not yet been read, or bit FCR has been updated

The debug view of the group result registers provides access to all result registers of a given group, however, without clearing the valid flag.

G0RESDy (y = 0 - 15)
Group 0 Result Reg. y, Debug (0780_H + y * 0004_H) Reset Value: $0000\ 0000_H$
G1RESDy (y = 0 - 15)
Group 1 Result Reg. y, Debug ($0B80_H$ + y * 0004_H) Reset Value: $0000\ 0000_H$
G2RESDy (y = 0 - 15)
Group 2 Result Reg. y, Debug ($0F80_H$ + y * 0004_H) Reset Value: $0000\ 0000_H$
G3RESDy (y = 0 - 15)
Group 3 Result Reg. y, Debug (1380_H + y * 0004_H) Reset Value: $0000\ 0000_H$



Field	Bits	Type	Description
RESULT	[15:0]	rh	**Result of Most Recent Conversion** The position of the result bits within this bitfield depends on the configured operating mode. Please, refer to **Section 16.8.2**.
DRC	[19:16]	rh	**Data Reduction Counter** Indicates the number of values still to be accumulated for the final result. The final result is available and valid flag VF is set when bitfield DRC becomes zero (by decrementing or by reload). See **"Data Modification" on Page 16-40**
CHNR	[24:20]	rh	**Channel Number** Indicates the channel number corresponding to the value in bitfield RESULT.
EMUX	[27:25]	rh	**External Multiplexer Setting** Indicates the setting of the external multiplexer, corresponding to the value in bitfield RESULT. *Note: Available in GxRESD0 only. Use GxRESD0 if EMUX information is required.*

Field	Bits	Type	Description
CRS	[29:28]	rh	**Converted Request Source** Indicates the request source that as requested the conversion to which the result value in bitfield RESULT belongs. 00_B Request source 0 01_B Request source 1 10_B Request source 2 11_B Reserved
FCR	30	rh	**Fast Compare Result** Indicates the result of an operation in Fast Compare Mode. 0_B Signal level was below compare value 1_B Signal level was above compare value
VF	31	rh	**Valid Flag** Indicates a new result in bitfield RESULT or bit FCR. 0_B No new result available 1_B Bitfield RESULT has been updated with new result value and has not yet been read, or bit FCR has been updated

The global result control register selects the behavior of the global result register.

GLOBRCR
Global Result Control Register (0280_H) Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SRGEN	**0**	**0**	**0**	**0**	**0**	**0**	**WFR**	**0**	**0**	**0**	**0**	**DRCTR**			
rw	r	r	r	r	r	r	rw	r	r	r	r	rw			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

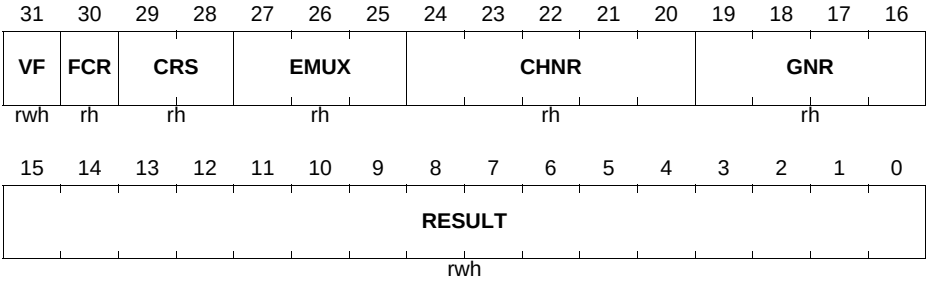
Field	Bits	Type	Description
0	[15:0]	r	**Reserved, write 0, read as 0**

Field	Bits	Type	Description
DRCTR	[19:16]	rw	**Data Reduction Control** Defines how result values are stored/accumulated in this register for the final result. The data reduction counter DRC can be loaded from this bitfield. 0000_BData reduction disabled others: see **"Function of Bitfield DRCTR" on Page 16-40**[1)]
0	[23:20]	r	**Reserved, write 0, read as 0**
WFR	24	rw	**Wait-for-Read Mode Enable** 0_B Overwrite mode 1_B Wait-for-read mode enabled for this register
0	[30:25]	r	**Reserved, write 0, read as 0**
SRGEN	31	rw	**Service Request Generation Enable** 0_B No service request 1_B Service request after a result event

1) Only standard data reduction is available for the global result register, i.e. DMM is assumed as 00_B.

The global result register provides a common storage location for all channels of all groups.

GLOBRES
Global Result Register **(0300_H)** **Reset Value: 0000 0000$_H$**
GLOBRESD
Global Result Register, Debug **(0380_H)** **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
RESULT	[15:0]	rwh	**Result of most recent conversion** The position of the result bits within this bitfield depends on the configured operating mode.[1)] Please, refer to **Section 16.8.2**.
GNR	[19:16]	rh	**Group Number** Indicates the group to which the channel number in bitfield CHNR refers.
CHNR	[24:20]	rh	**Channel Number** Indicates the channel number corresponding to the value in bitfield RESULT.
EMUX	[27:25]	rh	**External Multiplexer Setting** Indicates the setting of the external multiplexer, corresponding to the value in bitfield RESULT.
CRS	[29:28]	rh	**Converted Request Source** Indicates the request source that as requested the conversion to which the result value in bitfield RESULT belongs.
FCR	30	rh	**Fast Compare Result** Indicates the result of an operation in Fast Compare Mode. 0_B Signal level was below compare value 1_B Signal level was above compare value
VF	31	rwh	**Valid Flag** Indicates a new result in bitfield RESULT or bit FCR. 0_B Read access: No new valid data available Write access: No effect 1_B Read access: Bitfield RESULT contains valid data and has not yet been read, or bit FCR has been updated Write access: Clear this valid flag and the data reduction counter (overrides a hardware set action)[1)]

1) Only writable in register GLOBRES, not in register GLOBRESD.

The valid flag register summarizes the valid flags of all result registers.

GxVFR (x = 0 - 1)
Valid Flag Register, Group x **(x * 0400_H + $05F8_H$)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
VF15	**VF14**	**VF13**	**VF12**	**VF11**	**VF10**	**VF9**	**VF8**	**VF7**	**VF6**	**VF5**	**VF4**	**VF3**	**VF2**	**VF1**	**VF0**
rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
VFy **(y = 0 - 15)**	y	rwh	**Valid Flag of Result Register x** Indicates a new result in bitfield RESULT or in bit FCR. 0_B Read access: No new valid data available Write access: No effect 1_B Read access: Result register x contains valid data and has not yet been read, or bit FCR has been updated Write access: Clear this valid flag and bitfield DRC in register GxRESy (overrides a hardware set action)
0	[31:16]	r	**Reserved, write 0, read as 0**

16.13.7 Miscellaneous Registers

The alias register can replace the channel numbers of channels CH0 and CH1 with another channel number. The reset value disables this redirection.

GxALIAS (x = 0 - 1)
Alias Register, Group x **(x * 0400_H + $04B0_H$)** **Reset Value: $0000\ 0100_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**ALIAS1**					**0**	**0**	**0**	**ALIAS0**				
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	rw					r	r	r	rw				

Field	Bits	Type	Description
ALIAS0	[4:0]	rw	**Alias Value for CH0 Conversion Requests** Indicates the channel that is converted instead of channel CH0. The conversion is done with the settings defined for channel CH0.
0	[7:5]	r	**Reserved, write 0, read as 0**
ALIAS1	[12:8]	rw	**Alias Value for CH1 Conversion Requests** Indicates the channel that is converted instead of channel CH1. The conversion is done with the settings defined for channel CH1.
0	[31:13]	r	**Reserved, write 0, read as 0**

The local boundary register GxBOUND defines group-specific boundary values or delta limits for Fast Compare Mode.
The global boundary register GLOBBOUND defines general compare values for all channels.

Depending on the conversion width, the respective left 12/10/8 bits of a bitfield are used. For 10/8-bit results, the lower 2/4 bits must be zero!

GxBOUND (x = 0 - 1)
Boundary Select Register, Group x
(x * 0400_H + $04B8_H$) **Reset Value: $0000\ 0000_H$**

GLOBBOUND
Global Boundary Select Register ($00B8_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		0		0		0		BOUNDARY1								0		0		0		0		BOUNDARY0							
r		r		r		r		rw								r		r		r		r		rw							

Field	Bits	Type	Description
BOUNDARY0	[11:0]	rw	**Boundary Value 0 for Limit Checking** Standard Mode: This value is compared against the left-aligned conversion result. Fast Compare Mode: This value is added to the reference value (upper delta).
0	[15:12]	r	**Reserved, write 0, read as 0**
BOUNDARY1	[27:16]	rw	**Boundary Value 1 for Limit Checking** Standard Mode: This value is compared against the left-aligned conversion result. Fast Compare Mode: This value is subtracted from the reference value (lower delta).
0	[31:28]	r	**Reserved, write 0, read as 0**

The Boundary Flag Register holds the boundary flags themselves together with bits to select the activation condition and the output signal polarity for each flag.

GxBFL (x = 0 - 1)
Boundary Flag Register, Group x

(x * 0400_H + $04C8_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**BFI 3**	**BFI 2**	**BFI 1**	**BFI 0**
r	r	r	r	r	r	r	r	r	r	r	r	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**BFA 3**	**BFA 2**	**BFA 1**	**BFA 0**	**0**	**0**	**0**	**0**	**BFL 3**	**BFL 2**	**BFL 1**	**BFL 0**
r	r	r	r	rw	rw	rw	rw	r	r	r	r	rh	rh	rh	rh

Field	Bits	Type	Description
BFLz (z = 0 - 3)	z	rh	**Boundary Flag z** 0_B Passive state: result has not yet crossed the activation boundary (see bitfield BFAz), or selected gate signal is inactive, or this boundary flag is disabled 1_B Active state: result has crossed the activation boundary
0	[7:4]	r	**Reserved, write 0, read as 0**
BFAz (z = 0 - 3)	8 + z	rw	**Boundary Flag z Activation Select** 0_B Set boundary flag BFLz if result is above the defined band or compare value, clear if below 1_B Set boundary flag BFLz if result is below the defined band or compare value, clear if above
0	[15:12]	r	**Reserved, write 0, read as 0**
BFIz (z = 0 - 3)	16 + z	rw	**Boundary Flag z Inversion Control** 0_B Use BFLz directly 1_B Invert value and use $\overline{\text{BFLz}}$
0	[31:20]	r	**Reserved, write 0, read as 0**

The Boundary Flag Software Register provides means to set or clear each flag by software.

GxBFLS (x = 0 - 1)
Boundary Flag Software Register, Group x
($x * 0400_H + 04CC_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**BFS 3**	**BFS 2**	**BFS 1**	**BFS 0**
r	r	r	r	r	r	r	r	r	r	r	r	w	w	w	w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**BFC 3**	**BFC 2**	**BFC 1**	**BFC 0**
r	r	r	r	r	r	r	r	r	r	r	r	w	w	w	w

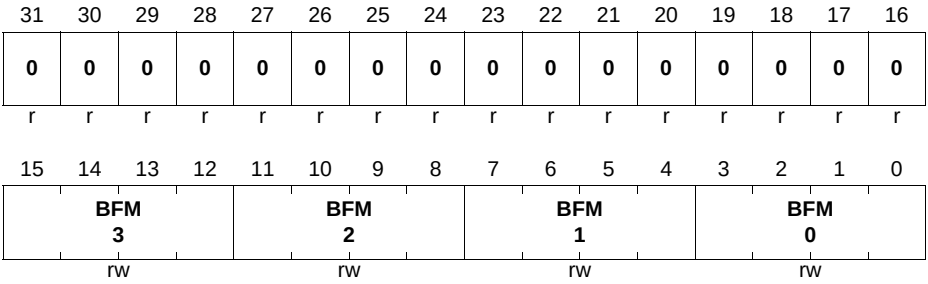
Field	Bits	Type	Description
BFCz (z = 0 - 3)	z	w	**Boundary Flag z Clear** 0_B No action 1_B Clear bit BFLz
0	[15:4]	r	**Reserved, write 0, read as 0**
BFSz (z = 0 - 3)	16 + z	w	**Boundary Flag z Set** 0_B No action 1_B Set bit BFLz
0	[31:20]	r	**Reserved, write 0, read as 0**

Note: If a boundary flag is used together with Fast Compare Mode, it is recommended not to direct results from other channels to the corresponding result register.

The Boundary Flag Control Register selects the basic operation of the boundary flags.

GxBFLC (x = 0 - 1)
Boundary Flag Control Register, Group x

($x * 0400_H + 04D0_H$) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
BFM0, BFM1, BFM2, BFM3	[3:0], [7:4], [11:8], [15:12]	rw	**Boundary Flag y Mode Control** 0000_B Disable boundary flag, BFLy is not changed 0001_B Always enable boundary flag (follow compare results) 0010_B Enable boundary flag while gate of source 0 is active, clear BFLy while gate is inactive 0011_B Enable boundary flag while gate of source 1 is active, clear BFLy while gate is inactive others: Reserved
0	[31:16]	r	**Reserved, write 0, read as 0**

The Boundary Flag Node Pointer Register directs signal VADCGxBFLy to alternate on-chip connections with other modules (in addition to the group-specific outputs). Possible targets are the corresponding common service request lines or the common boundary flag outputs (CBFLOUT0 … CBFLOUT3).

GxBFLNP (x = 0 - 1)
Boundary Flag Node Pointer Register, Group x
(x * 0400$_H$ + 04D4$_H$) **Reset Value: 0000 FFFF$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BFL3NP				**BFL2NP**				**BFL1NP**				**BFL0NP**			
rw				rw				rw				rw			

Field	Bits	Type	Description
BFL0NP, BFL1NP, BFL2NP, BFL3NP	[3:0], [7:4], [11:8], [15:12]	rw	**Boundary Flag y Node Pointer** 0000_BSelect common bondary flag output 0 … 0011_BSelect common bondary flag output 3 0100_BSelect shared service request line 0 … 0111_BSelect shared service request line 3 1111_BDisabled, no common output signal others: Reserved *Note: For shared service request lines see common groups in **Table 16-10**.*
0	[31:16]	r	**Reserved, write 0, read as 0**

The Synchronization Control Register controls the synchronization of kernels for parallel conversions.

Note: Program register GxSYNCTR only while bitfield GxARBCFG.ANONS = 00_B in all ADC kernels of the conversion group. Set the master's bitfield ANONC to 11_B afterwards.

GxSYNCTR (x = 0 - 1)
Synchronization Control Register, Group x

(x * 0400_H + $04C0_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**EVALR3**	**EVALR2**	**EVALR1**	**0**	**0**	**STSEL**	
r	r	r	r	r	r	r	r	r	rw	rw	rw	r	r	rw	

Field	Bits	Type	Description
STSEL	[1:0]	rw	**Start Selection** Controls the synchronization mechanism of the ADC kernel. 00_B Kernel is synchronization master: Use own bitfield GxARBCFG.ANONC 01_B Kernel is synchronization slave: Control information from input CI1 10_B Kernel is synchronization slave: Control information from input CI2 11_B Kernel is synchronization slave: Control information from input CI3 *Note: Control inputs CIx see **Figure 16-24**, connected kernels see **Table 16-11**.*
0	[3:2]	r	**Reserved, write 0, read as 0**

Field	Bits	Type	Description
EVALR1, EVALR2, EVALR3	4, 5, 6	rw	**Evaluate Ready Input Rx** Enables the ready input signal for a kernel of a conversion group. 0_B No ready input control 1_B Ready input Rx is considered for the start of a parallel conversion of this conversion group
0	[31:7]	r	**Reserved, write 0, read as 0**

GLOBTF
Global Test Functions Register (0160_H) Reset Value: $0000\ 0000_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**MDWC**	**0**	**0**	**0**	**0**	**0**	**0**	**PDD**
r	r	r	r	r	r	r	r	w	r	r	r	r	r	r	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CDWC	**0**	**0**	**0**	**0**	**CDSEL**		**CDEN**	**CDGR**				**0**	**0**	**0**	**0**
w	r	r	r	r	rw		rw	rw				r	r	r	r

Field	Bits	Type	Description
0	[3:0]	r	**Reserved, write 0, read as 0**
CDGR	[7:4]	rw	**Converter Diagnostics Group** Defines the group number to be used for converter diagnostics conversions.
CDEN	8	rw	**Converter Diagnostics Enable** 0_B All diagnostic pull devices are disconnected 1_B Diagnostic pull devices connected as selected by bitfield CDSEL
CDSEL	[10:9]	rw	**Converter Diagnostics Pull-Devices Select** 00_B Connected to VAREF 01_B Connected to VAGND 10_B Connected to 1/3rd VAREF 11_B Connected to 2/3rd VAREF
0	[14:11]	r	**Reserved, write 0, read as 0**

Field	Bits	Type	Description
CDWC	15	w	**Write Control for Conversion Diagnostics** 0_B No write access to parameters 1_B Bitfields CDSEL, CDEN, CDGR can be written
PDD	16	rw	**Pull-Down Diagnostics Enable** 0_B Disconnected 1_B The pull-down diagnostics device is active *Note: Channels with pull-down diagnostics device are marked in **Table 16-12**.*
0	[22:17]	r	**Reserved, write 0, read as 0**
MDWC	23	w	**Write Control for Multiplexer Diagnostics** 0_B No write access to parameters 1_B Bitfield PDD can be written
0	[31:24]	r	**Reserved, write 0, read as 0**

GxEMUXCTR (x = 0 - 1)
External Multiplexer Control Register, Group x

($x * 0400_H + 05F0_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EMX WC	**EMX CSS**	**EMX ST**	**EMX COD**	**EMUX MODE**						**EMUX CH**					
w	rw	rw	rw	rw						rw					

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**		**EMUX ACT**		**0**	**0**	**0**	**0**	**0**		**EMUX SET**	
r	r	r	r	r		rh		r	r	r	r	r		rw	

Field	Bits	Type	Description
EMUXSET	[2:0]	rw	**External Multiplexer Start Selection**[1)] Defines the initial setting for the external multiplexer.
0	[7:3]	r	**Reserved, write 0, read as 0**

Field	Bits	Type	Description
EMUXACT	[10:8]	rh	**External Multiplexer Actual Selection** Defines the current value for the external multiplexer selection. This bitfield is loaded from bitfield EMUXSET and modified according to the operating mode selected by bitfield EMUXMODE.
0	[15:11]	r	**Reserved, write 0, read as 0**
EMUXCH	[25:16]	rw	**External Multiplexer Channel Select** Defines the channel(s) to which the external multiplexer control is applied. **EMXCSS = 0: Channel number**, the lower 5 bits select an arbitrary channel (valid numbers are limited by the number of available channels, unused bits shall be 0) **EMXCSS = 1: Channel enable**, each bit enables the associated channel (multiple channels can be selected/enabled)
EMUXMODE	[27:26]	rw	**External Multiplexer Mode** 00_B Software control (no hardware action) 01_B Steady mode (use EMUXSET value) 10_B Single-step mode[1)2)] 11_B Sequence mode[1)]
EMXCOD	28	rw	**External Multiplexer Coding Scheme** 0_B Output the channel number in binary code 1_B Output the channel number in Gray code
EMXST	29	rw	**External Multiplexer Sample Time Control** 0_B Use STCE whenever the setting changes 1_B Use STCE for each conversion of an external channel
EMXCSS	30	r	**External Multiplexer Channel Selection Style** 0_B Channel number: Bitfield EMUXCH selects an arbitrary channel 1_B Channel enable: Each bit of bitfield EMUXCH selects the associated channel for EMUX control
EMXWC	31	w	**Write Control for EMUX Configuration** 0_B No write access to EMUX cfg. 1_B Bitfields EMXMODE, EMXCOD, EMXST, EMXCSS can be written

1) For single-step mode and sequence mode: Select the start value before selecting the respective mode.
2) Single-step mode modifies the EMUX channel number each time an EMUX-enabled channel is converted. Therefore, single-step mode works best with a single channel, because otherwise some external channels may be skipped.

Register EMUXSEL is a global register which assigns an arbitrary group to each of the EMUX interfaces.

EMUXSEL
External Multiplexer Select Register

(03F0$_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**		**EMUX GRP1**				**EMUX GRP0**		
r	r	r	r	r	r	r	r		rw				rw		

Field	Bits	Type	Description
EMUXGRP0, EMUXGRP1	[3:0], [7:4]	rw	**External Multiplexer Group for Interface x** Defines the group whose external multiplexer control signals are routed to EMUX interface x.[1)]
0	[31:8]	r	**Reserved, write 0, read as 0**

1) The pins that are associated with each EMUX interface are listed in **Table 16-13 "Digital Connections in the XMC4[12]00" on Page 16-136**.

16.13.8 Service Request Registers

GxSEFLAG (x = 0 - 1)
Source Event Flag Register, Group x

(x * 0400$_H$ + 0588$_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**SEV 1**	**SEV 0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	rwh	rwh

Field	Bits	Type	Description
SEV0, SEV1	0, 1	rwh	**Source Event 0/1** 0_B No source event 1_B A source event has occurred
0	[31:2]	r	**Reserved, write 0, read as 0**

Note: Software can set all flags in register GxSEFLAG and trigger the corresponding event by writing 1 to the respective bit. Writing 0 has no effect.
Software can clear all flags in register GxSEFLAG by writing 1 to the respective bit in register GxSEFCLR.

GxCEFLAG (x = 0 - 1)
Channel Event Flag Register, Group x
(x * 0400_H + 0580_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**CEV 7**	**CEV 6**	**CEV 5**	**CEV 4**	**CEV 3**	**CEV 2**	**CEV 1**	**CEV 0**
r	r	r	r	r	r	r	r	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
CEVy (y = 0 - 7)	y	rwh	**Channel Event for Channel y** 0_B No channel event 1_B A channel event has occurred
0	[31:8]	r	**Reserved, write 0, read as 0**

Note: Software can set all flags in register GxCEFLAG and trigger the corresponding event by writing 1 to the respective bit. Writing 0 has no effect.
Software can clear all flags in register GxCEFLAG by writing 1 to the respective bit in register GxCEFCLR.

GxREFLAG (x = 0 - 1)
Result Event Flag Register, Group x

($x * 0400_H + 0584_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
REV 15	**REV 14**	**REV 13**	**REV 12**	**REV 11**	**REV 10**	**REV 9**	**REV 8**	**REV 7**	**REV 6**	**REV 5**	**REV 4**	**REV 3**	**REV 2**	**REV 1**	**REV 0**
rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
REVy (y = 0 - 15)	y	rwh	**Result Event for Result Register y** 0_B No result event 1_B New result was stored in register GxRESy
0	[31:16]	r	**Reserved, write 0, read as 0**

Note: Software can set all flags in register GxREFLAG and trigger the corresponding event by writing 1 to the respective bit. Writing 0 has no effect.
Software can clear all flags in register GxREFLAG by writing 1 to the respective bit in register GxREFCLR.

GxSEFCLR (x = 0 - 1)
Source Event Flag Clear Register, Group x

($x * 0400_H + 0598_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**SEV 1**	**SEV 0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	w	w

Field	Bits	Type	Description
SEV0, SEV1	0, 1	w	**Clear Source Event 0/1** 0_B No action 1_B Clear the source event flag in GxSEFLAG
0	[31:2]	r	**Reserved, write 0, read as 0**

GxCEFCLR (x = 0 - 1)
Channel Event Flag Clear Register, Group x

(x * 0400_H + 0590_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**CEV 7**	**CEV 6**	**CEV 5**	**CEV 4**	**CEV 3**	**CEV 2**	**CEV 1**	**CEV 0**
r	r	r	r	r	r	r	r	w	w	w	w	w	w	w	w

Field	Bits	Type	Description
CEVy (y = 0 - 7)	y	w	**Clear Channel Event for Channel y** 0_B No action 1_B Clear the channel event flag in GxCEFLAG
0	[31:8]	r	**Reserved, write 0, read as 0**

GxREFCLR (x = 0 - 1)
Result Event Flag Clear Register, Group x

(x * 0400_H + 0594_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
REV 15	**REV 14**	**REV 13**	**REV 12**	**REV 11**	**REV 10**	**REV 9**	**REV 8**	**REV 7**	**REV 6**	**REV 5**	**REV 4**	**REV 3**	**REV 2**	**REV 1**	**REV 0**
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Field	Bits	Type	Description
REVy (y = 0 - 15)	y	w	**Clear Result Event for Result Register y** 0_B No action 1_B Clear the result event flag in GxREFLAG
0	[31:16]	r	**Reserved, write 0, read as 0**

GLOBEFLAG
Global Event Flag Register

($00E0_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**REV GLB CLR**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**SEV GLB CLR**
r	r	r	r	r	r	r	w	r	r	r	r	r	r	r	w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**REV GLB**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**SEV GLB**
r	r	r	r	r	r	r	rwh	r	r	r	r	r	r	r	rwh

Field	Bits	Type	Description
SEVGLB	0	rwh	**Source Event (Background)** 0_B No source event 1_B A source event has occurred

Field	Bits	Type	Description
0	[7:1]	r	**Reserved, write 0, read as 0**
REVGLB	8	rwh	**Global Result Event** 0_B No result event 1_B New result was stored in register GLOBRES
0	[15:9]	r	**Reserved, write 0, read as 0**
SEVGLBCLR	16	w	**Clear Source Event (Background)** 0_B No action 1_B Clear the source event flag SEVGLB
0	[23:17]	r	**Reserved, write 0, read as 0**
REVGLBCLR	24	w	**Clear Global Result Event** 0_B No action 1_B Clear the result event flag REVGLB
0	[31:25]	r	**Reserved, write 0, read as 0**

Note: Software can set flags REVGLB and SEVGLB and trigger the corresponding event by writing 1 to the respective bit. Writing 0 has no effect.
Software can clear these flags by writing 1 to bit REVGLBCLR and SECGLBCLR, respectively.

GxSEVNP (x = 0 - 1)
Source Event Node Pointer Register, Group x
($x * 0400_H + 05C0_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**SEV1NP**				**SEV0NP**			
r	r	r	r	r	r	r	r	rw				rw			

Field	Bits	Type	Description
SEV0NP, SEV1NP	[3:0], [7:4]	rw	**Service Request Node Pointer Source Event i[1)]** Routes the corresponding event trigger to one of the service request lines (nodes). 0000_B Select service request line 0 of group x … 0011_B Select service request line 3 of group x 0100_B Select shared service request line 0 … 0111_B Select shared service request line 3 $1xxx_B$ Reserved *Note: For shared service request lines see common groups in **Table 16-10**.*
0	[31:8]	r	**Reserved, write 0, read as 0**

1) Source 0 is an 8-stage queued source, source 1 is a channel scan source.

GxCEVNP0 (x = 0 - 1)
Channel Event Node Pointer Register 0, Group x
(x * 0400_H + $05A0_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CEV7NP				**CEV6NP**				**CEV5NP**				**CEV4NP**			
rw				rw				rw				rw			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CEV3NP				**CEV2NP**				**CEV1NP**				**CEV0NP**			
rw				rw				rw				rw			

Field	Bits	Type	Description
CEV0NP, **CEV1NP,** **CEV2NP,** **CEV3NP,** **CEV4NP,** **CEV5NP,** **CEV6NP,** **CEV7NP**	[3:0], [7:4], [11:8], [15:12], [19:16], [23:20], [27:24], [31:28]	rw	**Service Request Node Pointer Channel Event i** Routes the corresponding event trigger to one of the service request lines (nodes). 0000_B Select service request line 0 of group x … 0011_B Select service request line 3 of group x 0100_B Select shared service request line 0 … 0111_B Select shared service request line 3 $1xxx_B$ Reserved *Note: For shared service request lines see common groups in **Table 16-10**.*

GxREVNP0 (x = 0 - 1)
Result Event Node Pointer Register 0, Group x

(x * 0400_H + $05B0_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
REV7NP				**REV6NP**				**REV5NP**				**REV4NP**			
rw				rw				rw				rw			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
REV3NP				**REV2NP**				**REV1NP**				**REV0NP**			
rw				rw				rw				rw			

Field	Bits	Type	Description
REV0NP, **REV1NP,** **REV2NP,** **REV3NP,** **REV4NP,** **REV5NP,** **REV6NP,** **REV7NP**	[3:0], [7:4], [11:8], [15:12], [19:16], [23:20], [27:24], [31:28]	rw	**Service Request Node Pointer Result Event i** Routes the corresponding event trigger to one of the service request lines (nodes). 0000_B Select service request line 0 of group x … 0011_B Select service request line 3 of group x 0100_B Select shared service request line 0 … 0111_B Select shared service request line 3 $1xxx_B$ Reserved *Note: For shared service request lines see common groups in **Table 16-10**.*

GxREVNP1 (x = 0 - 1)
Result Event Node Pointer Register 1, Group x

(x * 0400_H + $05B4_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
REV15NP				**REV14NP**				**REV13NP**				**REV12NP**			
rw				rw				rw				rw			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
REV11NP				**REV10NP**				**REV9NP**				**REV8NP**			
rw				rw				rw				rw			

Field	Bits	Type	Description
REV8NP, **REV9NP,** **REV10NP,** **REV11NP,** **REV12NP,** **REV13NP,** **REV14NP,** **REV15NP**	[3:0], [7:4], [11:8], [15:12], [19:16], [23:20], [27:24], [31:28]	rw	**Service Request Node Pointer Result Event i** Routes the corresponding event trigger to one of the service request lines (nodes). 0000_B Select service request line 0 of group x … 0011_B Select service request line 3 of group x 0100_B Select shared service request line 0 … 0111_B Select shared service request line 3 $1xxx_B$ Reserved *Note: For shared service request lines see common groups in **Table 16-10**.*

GLOBEVNP
Global Event Node Pointer Register

(0140_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**REV0NP**			
r	r	r	r	r	r	r	r	r	r	r	r	rw			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**SEV0NP**			
r	r	r	r	r	r	r	r	r	r	r	r	rw			

Field	Bits	Type	Description
SEV0NP	[3:0]	rw	**Service Request Node Pointer Backgr. Source** Routes the corresponding event trigger to one of the service request lines (nodes). 0000_B Select shared service request line 0 of common service request group 0 … 0011_B Select shared service request line 3 of common service request group 0 0100_B Select shared service request line 0 of common service request group 1 … 0111_B Select shared service request line 3 of common service request group 1 $1xxx_B$ Reserved *Note: For shared service request lines see common groups in **Table 16-10**.*
0	[15:4]	r	**Reserved, write 0, read as 0**
REV0NP	[19:16]	rw	**Service Request Node Pointer Backgr. Result** Routes the corresponding event trigger to one of the service request lines (nodes). 0000_B Select shared service request line 0 of common service request group 0 … 0011_B Select shared service request line 3 of common service request group 0 0100_B Select shared service request line 0 of common service request group 1 … 0111_B Select shared service request line 3 of common service request group 1 $1xxx_B$ Reserved *Note: For shared service request lines see common groups in **Table 16-10**.*
0	[31:20]	r	**Reserved, write 0, read as 0**

GxSRACT (x = 0 - 1)
Service Request Software Activation Trigger, Group x
(x * 0400_H + $05C8_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**ASSR3**	**ASSR2**	**ASSR1**	**ASSR0**	**0**	**0**	**0**	**0**	**AGSR3**	**AGSR2**	**AGSR1**	**AGSR0**
r	r	r	r	w	w	w	w	r	r	r	r	w	w	w	w

Field	Bits	Type	Description
AGSRy (y = 0 - 3)	y	w	**Activate Group Service Request Node y** 0_B No action 1_B Activate the associated service request line
0	[7:4]	r	**Reserved, write 0, read as 0**
ASSRy (y = 0 - 3)	8 + y	w	**Activate Shared Service Request Node y** 0_B No action 1_B Activate the associated service request line
0	[31:12]	r	**Reserved, write 0, read as 0**

16.14 Interconnects

This section describes the actual implementation of the ADC module into the XMC4[12]00, i.e. the incorporation into the microcontroller system.

16.14.1 Product-Specific Configuration

The functional description describes the features and operating modes of the A/D Converters in a general way. This section summarizes the configuration that is available in this product (XMC4[12]00).

Each converter group is equipped with a separate analog converter module and a dedicated analog input multiplexer.

Table 16-10 General Converter Configuration in the XMC4[12]00

Converter Group	Input Channels	Channels with Alternate Reference	12-bit Performance	Common Service Request Group
G0	0 … 7	4	Calibrated	C0
G1	0 … 7	8	Calibrated	C0

Synchronization Groups in the XMC4[12]00

The converter kernels in the XMC4[12]00 can be connected to synchronization groups to achieve parallel conversion of several input channels.

Table 16-11 summarizes which kernels can be synchronized for parallel conversions.

Table 16-11 Synchronization Groups in the XMC4[12]00

ADC Kernel	Synchr. Group	Master selected by control input CIx[1]			
		CI0[2]	CI1	CI2	CI3
ADC00	A	ADC00	ADC01	-	-
ADC01	A	ADC01	ADC00	-	-

1) The control input is selected by bitfield STSEL in register **GxSYNCTR (x = 0 - 1)**.
Select the corresponding ready inputs accordingly by bits EVALRx.

2) Control input CI0 always selects the own control signals of the corresponding ADC kernel. This selection is meant for the synchronization master or for stand-alone operation.

16.14.2 Analog Module Connections in the XMC4[12]00

The VADC module accepts a number of analog input signals. The analog input multiplexers select the input channels to be converted from the signals available in this product.

The exact number of analog input channels and the available connection to port pins depend on the employed product type (see also **Table 16-10**). A summary of channels enclosing all versions of the XMC4[12]00 can be found in **Table 16-12**.

Input channels marked “PDD” provide a pull-down device for pull-down diagnostics.

Input channels marked “noAltref” cannot select the alternate reference voltage from channel 0 of the corresponding converter.

Table 16-12 Analog Connections in the XMC4[12]00

Signal	Dir.	Source/Destin.	Description
V_{AREF}	I	VDDA	positive analog reference
V_{AGND}	I	VSSA	negative analog reference
G0CH0	I	P14.0	analog input channel 0 of group 0
G0CH3	I	P14.3	analog input channel 3 of group 0
G0CH4	I	P14.4	analog input channel 4 of group 0
G0CH5	I	P14.5	analog input channel 5 of group 0
G0CH6	I	P14.6	analog input channel 6 of group 0
G0CH7 (PDD)	I	P14.7	analog input channel 7 of group 0
G1CH0	I	P14.8	analog input channel 0 of group 1
G1CH1	I	P14.9	analog input channel 1 of group 1
G1CH3	I	P14.3	analog input channel 3 of group 1
G1CH6	I	P14.14	analog input channel 6 of group 1

16.14.3 Digital Module Connections in the XMC4[12]00

The VADC module accepts a number of digital input signals and generates a number of output signals. This section summarizes the connection of these signals to other on-chip modules or to external resources via port pins.

Table 16-13 Digital Connections in the XMC4[12]00

Signal	Dir.	Source/Destin.	Description
Gate Inputs for Each Group			
VADC.GxREQGTA	I	CCU40.ST3	Gating input A
VADC.GxREQGTB	I	CCU41.ST3	Gating input B
VADC.GxREQGTC	I	CCU40.SR0	Gating input C
VADC.GxREQGTD	I	CCU41.SR1	Gating input D
VADC.GxREQGTE	I	CCU80.ST3A	Gating input E
VADC.GxREQGTF	I	CCU80.ST3B	Gating input F
VADC.GxREQGTG	I	-	Gating input G
VADC.GxREQGTH	I	-	Gating input H
VADC.G0REQGTI	I (s)	DAC0.SGN	Gating input I
VADC.G1REQGTI	I (s)	DAC1.SGN	Gating input I
VADC.GxREQGTJ	I (s)	LEDTS.FN	Gating input J
VADC.G0REQGTK	I (s)	VADC.G1BFLOUT0	Gating input K
VADC.G1REQGTK	I (s)	VADC.G0BFLOUT0	Gating input K
VADC.G0REQGTL	I (s)	VADC.G3SAMPLE[1)]	Gating input L
VADC.G1REQGTL	I (s)	VADC.G0SAMPLE	Gating input L
VADC.GxREQGTM	I	CCU80.SR0	Gating input M
VADC.GxREQGTN	I	CCU80.SR1	Gating input N
VADC.GxREQGTO	I	ERU1.PDOUT0	Gating input O
VADC.GxREQGTP	I	ERU1.PDOUT1	Gating input P
VADC.GxREQGTySEL	O	VADC.GxREQTRyP[2)]	Selected gating signal of the respective source
Gate Inputs for Global Background Source			
VADC.BGREQGTA	I	CCU40.ST3	Gating input A, background source
VADC.BGREQGTB	I	CCU41.ST3	Gating input B, background source
VADC.BGREQGTC	I	CCU40.SR0	Gating input C, background source

Table 16-13 Digital Connections in the XMC4[12]00 (cont'd)

Signal	Dir.	Source/Destin.	Description
VADC.BGREQGTD	I	CCU41.SR1	Gating input D, background source
VADC.BGREQGTE	I	CCU80.ST3A	Gating input E, background source
VADC.BGREQGTF	I	CCU80.ST3B	Gating input F, background source
VADC.BGREQGTG	I	-	Gating input G, background source
VADC.BGREQGTH	I	-	Gating input H, background source
VADC.BGREQGTI	I (s)	DAC0.SGN	Gating input I, background source
VADC.BGREQGTJ	I (s)	LEDTS.FN	Gating input J, background source
VADC.BGREQGTK	I (s)	VADC.G1BFLOUT0	Gating input K, background source
VADC.BGREQGTL	I (s)	-	Gating input L, background source
VADC.BGREQGTM	I	CCU80.SR0	Gating input M, background source
VADC.BGREQGTN	I	CCU80.SR1	Gating input N, background source
VADC.BGREQGTO	I	ERU1.PDOUT0	Gating input O, background source
VADC.BGREQGTP	I	ERU1.PDOUT1	Gating input P, background source
VADC.BGREQGTSEL	O	VADC.BGREQTRP[2]	Selected gating signal
Trigger Inputs for Each Group			
VADC.GxREQTRA	I	CCU40.SR2	Trigger input A
VADC.GxREQTRB	I	CCU40.SR3	Trigger input B
VADC.GxREQTRC	I	CCU41.SR2	Trigger input C
VADC.GxREQTRD	I	CCU41.SR3	Trigger input D
VADC.GxREQTRE	I	-	Trigger input E
VADC.GxREQTRF	I	-	Trigger input F
VADC.GxREQTRG	I	-	Trigger input G
VADC.GxREQTRH	I	-	Trigger input H
VADC.GxREQTRI	I (s)	CCU80.SR2	Trigger input I
VADC.GxREQTRJ	I (s)	CCU80.SR3	Trigger input J
VADC.GxREQTRK	I (s)	-	Trigger input K
VADC.GxREQTRL	I (s)	-	Trigger input L
VADC.GxREQTRM	I	ERU1.IOUT0	Trigger input M
VADC.G0REQTRN	I	ERU1.IOUT1	Trigger input N
VADC.G1REQTRN	I	ERU1.IOUT1	Trigger input N

Table 16-13 Digital Connections in the XMC4[12]00 (cont'd)

Signal	Dir.	Source/Destin.	Description
VADC.G0REQTRO	I	POSIF0.SR1	Trigger input O
VADC.G1REQTRO	I	-	Trigger input O
VADC.GxREQTRyP	I	VADC.GxREQGTySEL[2)]	Extend triggers to selected gating input of the respective source
VADC.GxREQTRySEL	O	-	Selected trigger signal of the respective source
Trigger Inputs for Global Background Source			
VADC.BGREQTRA	I	CCU40.SR2	Trigger input A, background source
VADC.BGREQTRB	I	CCU40.SR3	Trigger input B, background source
VADC.BGREQTRC	I	CCU41.SR2	Trigger input C, background source
VADC.BGREQTRD	I	CCU41.SR3	Trigger input D, background source
VADC.BGREQTRE	I	-	Trigger input E, background source
VADC.BGREQTRF	I	-	Trigger input F, background source
VADC.BGREQTRG	I	-	Trigger input G, background source
VADC.BGREQTRH	I	-	Trigger input H, background source
VADC.BGREQTRI	I (s)	CCU80.SR2	Trigger input I, background source
VADC.BGREQTRJ	I (s)	CCU80.SR3	Trigger input J, background source
VADC.BGREQTRK	I (s)	-	Trigger input K, background source
VADC.BGREQTRL	I (s)	-	Trigger input L, background source
VADC.BGREQTRM	I	ERU1.IOUT0	Trigger input M, background source
VADC.BGREQTRN	I	ERU1.IOUT1	Trigger input N, background source
VADC.BGREQTRO	I	POSIF0.SR1	Trigger input O, background source
VADC.BGREQTRP	I	VADC.BGREQGTSEL[2)]	Extend triggers to selected gating input of the background source
VADC.BGREQTRSEL	O	-	Selected trigger signal of the background source
System-Internal Connections			
VADC.G0SAMPLE[1)]	O	VADC.G1REQGTL	Indicates the input signal sample phase
VADC.G1SAMPLE	O	VADC.G2REQGTL	Indicates the input signal sample phase

Table 16-13 Digital Connections in the XMC4[12]00 (cont'd)

Signal	Dir.	Source/Destin.	Description
VADC.G0ARBCNT	O	CCU40.IN3G	Outputs a (count) pulse for each arbiter round
VADC.G1ARBCNT	O	CCU41.IN3G	Outputs a (count) pulse for each arbiter round
VADC.GxSR0	O	NVIC, GPDMA	Service request 0 of group x
VADC.GxSR1	O	NVIC, GPDMA	Service request 1of group x
VADC.GxSR2	O	NVIC, GPDMA	Service request 2 of group x
VADC.G0SR3	O	NVIC, GPDMA CCU80.IN0F	Service request 3 of group 0
VADC.G1SR3	O	NVIC, GPDMA	Service request 3 of group 1
VADC.C0SR0	O	NVIC, GPDMA ERU1.OGU01 POSIF0.IN2C	Service request 0 of common block 0
VADC.C0SR1	O	NVIC, GPDMA ERU1.OGU11	Service request 1 of common block 0
VADC.C0SR2	O	NVIC, GPDMA ERU1.OGU21	Service request 2 of common block 0
VADC.C0SR3	O	NVIC, GPDMA ERU1.OGU31	Service request 3 of common block 0
VADC.EMUX00	O	GPIO	Control of external analog multiplexer interface 0
VADC.EMUX01	O	GPIO	
VADC.EMUX02	O	GPIO	
VADC.EMUX10	O	-	Control of external analog multiplexer interface 1
VADC.EMUX11	O	GPIO	
VADC.EMUX12	O	GPIO	
CBFLOUT0	O	-	Common boundary flag output 0
CBFLOUT1	O	-	Common boundary flag output 1
CBFLOUT2	O	-	Common boundary flag output 2
CBFLOUT3	O	-	Common boundary flag output 3
VADC.G0BFLOUT0	O	VADC.G1REQGTK VADC.BGREQGTK CCU41.IN0L CCU80.IN0I	Boundary flag 0 output of group 0

Table 16-13 Digital Connections in the XMC4[12]00 (cont'd)

Signal	Dir.	Source/Destin.	Description
VADC.G1BFLOUT0	O	VADC.G0REQGTK POSIF0.IN0C	Boundary flag 0 output of group 1
VADC.GxBFL0	O	-	Boundary flag 0 level of group x
VADC.GxBFSEL0	I	0	Boundary flag 0 (group x) source select
VADC.GxBFDAT0	I	0	Boundary flag 0 (group x) alternate data
VADC.G0BFLOUT1	O	VADC.G0REQGTK CCU41.IN2L CCU80.IN1I	Boundary flag 1 output of group 0
VADC.G1BFLOUT1	O	POSIF0.IN1C	Boundary flag 1 output of group 1
VADC.GxBFL1	O	-	Boundary flag 1 level of group x
VADC.GxBFSEL1	I	0	Boundary flag 1 (group x) source select
VADC.GxBFDAT1	I	0	Boundary flag 1 (group x) alternate data
VADC.G0BFLOUT2	O	VADC.G3REQGTK CCU41.IN3L CCU80.IN2I	Boundary flag 2 output of group 0
VADC.G1BFLOUT2	O	POSIF0.EWHEA	Boundary flag 2 output of group 1
VADC.GxBFL2	O	-	Boundary flag 2 level of group x
VADC.GxBFSEL2	I	0	Boundary flag 2 (group x) source select
VADC.GxBFDAT2	I	0	Boundary flag 2 (group x) alternate data
VADC.G0BFLOUT3	O	VADC.G2REQGTK CCU80.IN3I ERU1.0B2 ERU1.2B2	Boundary flag 3 output of group 0
VADC.G1BFLOUT3	O	ERU1.1B2 ERU1.3B2	Boundary flag 3 output of group 1
VADC.GxBFL3	O	-	Boundary flag 3 level of group x

Table 16-13 Digital Connections in the XMC4[12]00 (cont'd)

Signal	Dir.	Source/Destin.	Description
VADC.GxBFSEL3	I	0	Boundary flag 3 (group x) source select
VADC.GxBFDAT3	I	0	Boundary flag 3 (group x) alternate data

1) To use the SAMPLE output signals, enable them via register **OCS**.

2) Internal signal connection.

17 Digital to Analog Converter (DAC)

This chapter describes the two Digital to Analog Converter (DAC) channels available in the module.

17.1 Overview

The module consists of two separate 12-bit digital to analog converters (DACs). It converts two digital input signals into two analog voltage signal outputs at a maximum conversion rate of 5 MHz. The available design structure is based on a current steering architecture with internal reference generation and provides buffered voltage outputs. In order to reduce power consumption during inactive periods, a power down mode is available.

A built-in wave generator mode allows the CPU free generation of a selectable choice of wave forms. Alternatively values can be feed via CPU or DMA directly to one or both DAC channels. Additionally an offset can be added and the amplitude can be scaled. Several time trigger sources are possible.

17.1.1 Features

Analog features

- DAC resolution 12 bit;
- Conversion rate up to 5 MHz with reduced accuracy;
- Conversion rate up to 2 MHz at full accuracy;
- Maximum settling time of 2 us for a full scale 12-bit input code transition;
- Buffered voltage output;
- Direct drive of 5 kOhm / 50 pF terminated load;
- Segmented current steering architecture;
- Low glitch energy;
- Power down mode;
- DAC output and ADC input share the same analog input pin. ADC measurement is possible in parallel to DAC usage.
- V_{DDA} analog supply;

Digital features

- One Advanced Microcontroller Bus Architecture (AMBA) 32-bit AHB-Lite bus interface for data transfer and control of both DACs;
- Self triggered Direct Memory Access (DMA) handling capability with independent or simultaneous data handling for the two DAC channels (see **Section 17.2.4**);
- First In First Out (FIFO) data buffers to allow a longer service request latency and to guarantee a continuous data transfer to the DACs (see **Section 17.2.4**);

- Pattern generators available with freely programmable waveforms for both DACs (see **Section 17.2.5**);
- Independent noise generators available for both DACs (see **Section 17.2.6**);
- Data scaling by shift operation (multiplication and division by 2, 4, 8,..., 128) of the DACs' input data;
- Data offset value addition to the DACs input data;
- 8 selectable external trigger inputs;
- Internal integer clock divider for DAC trigger generation;
- Software trigger option;
- V_{DDC} digital supply;

17.1.2 Block Diagram

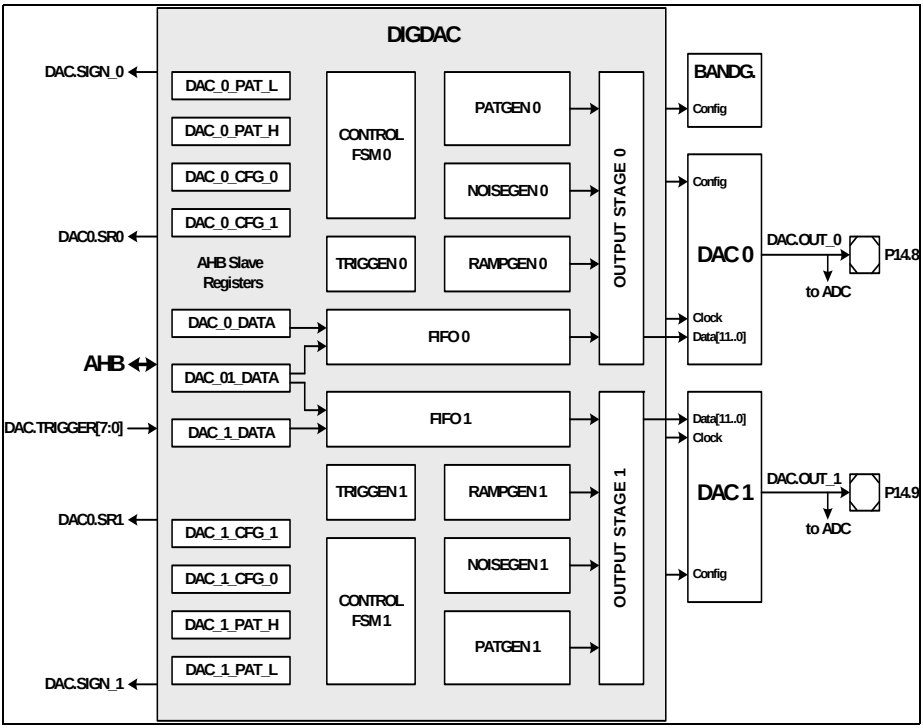


Figure 17-1 Block Diagram of DAC Module including Digdac Submodule

17.2 Operating Modes

The following chapters describe all the DAC's functional operating modes and how to use them. All used configuration parameters in this chapter are part of the registers described in **Section 17.6**.

17.2.1 Hardware features

To facilitate the understanding of the different operating modes, a brief description of the supporting hardware features is given here:

Control and Data Registers

All control and data registers shown on the left hand side of **Figure 17-1** are described in **Section 17.6** in detail. They are connected to the AHB-Lite bus via an AHB-Lite slave interface. The interface also handles the necessary error response generation without introducing any latency.

Control Logic - Finite State Machines (FSM)

The two control finite state machines control all data processing modes of the DAC (see **Figure 17-1**). This means that they are responsible for the start and stop operating sequence, the service request generation for DMA handling for the so called data-mode and the control of the data FIFOs, the pattern generators, the noise generators and the ramp generators. Both FSMs are equivalent in structure and both are able to operate fully independent for the two DAC channels. For the simultaneous data-mode both FSMs are active, but only the service request signal of channel 0 (DAC0.Service Request(SR)0) should be evaluated by the DMA controller. Of course in that simultaneous data-mode the trigger source has to be the same for both channels.

17.2.1.1 Trigger Generators (TG)

The block diagram in **Figure 17-2** shows one of the two DAC's trigger generators.

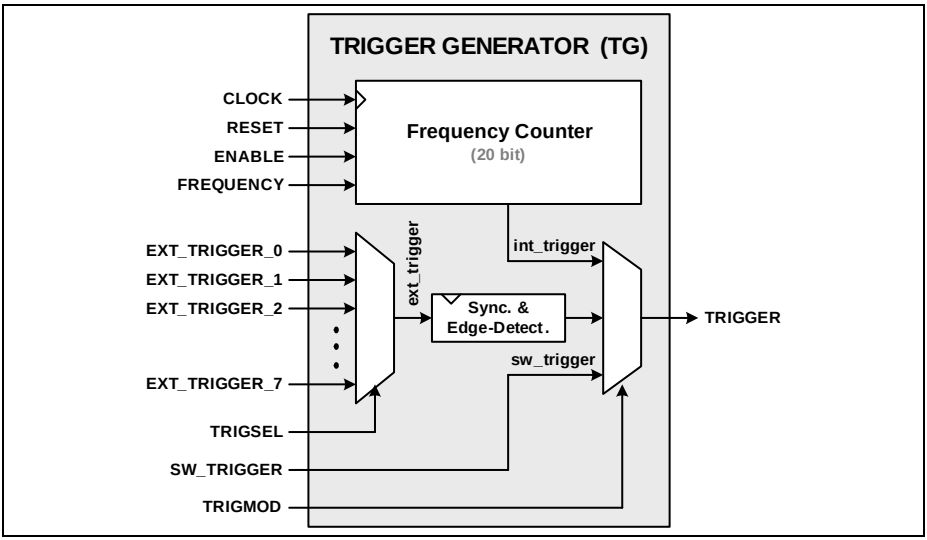


Figure 17-2 Trigger Generator Block Diagram

The TG consists of two multiplexers and one frequency counter. The first multiplexer selects between one of the eight external trigger sources whereas the second one enables switching between this selected external trigger source, an internally generated trigger and an additional software trigger input. The internal trigger is generated by the frequency counter which operates as a simple integer clock divider. So the internal trigger period can only be a multiple of the DACs system clock period. In order to guarantee correct operation of the analog part of the DAC the smallest frequency divider value is limited to 16 by hardware.

The external trigger inputs are synchronized to the system clock and only the rising edge is evaluated by the TG.

The software trigger input is connected to a register bit which is automatically cleared after it has been set to one. For this reason the output trigger of the TG is always a pulse with a pulse width of one system clock cycle for all three trigger possible modes.

17.2.1.2 Data FIFO buffer (FIFO)

The block diagram in **Figure 17-3** shows one of the two data FIFO buffers, one for each DAC.

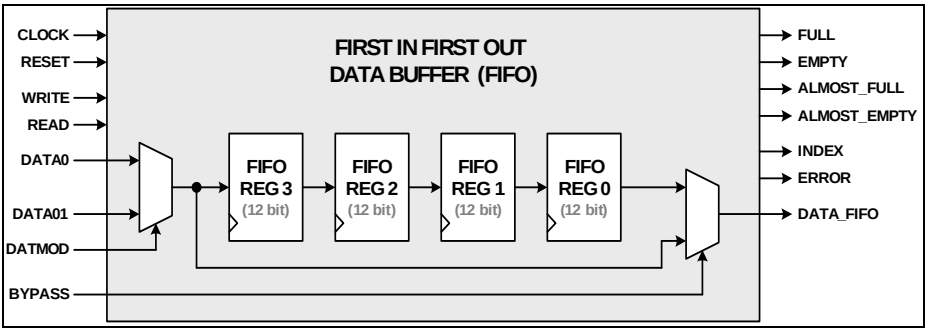


Figure 17-3 Data FIFO Block Diagram

This data FIFO buffer with four FIFO registers is introduced to allow a longer service request latency and to guarantee a continuous data processing for the DAC channels. It is used for DAC's so called data processing mode described in **Section 17.2.4**. All FIFO's read and write operations are controlled by the corresponding Control FSM. A read operation is triggered by the chosen trigger and a write operation is initiated by an Advanced High-performance Bus (AHB) write operation to the currently used data register. The FIFO's status outputs named full, empty and index can be read by the software. A FIFO bypass used for all other DACs operating modes is also available.

17.2.1.3 Data output stage

The block diagram in **Figure 17-4** shows one of the two DACs' data output stages.

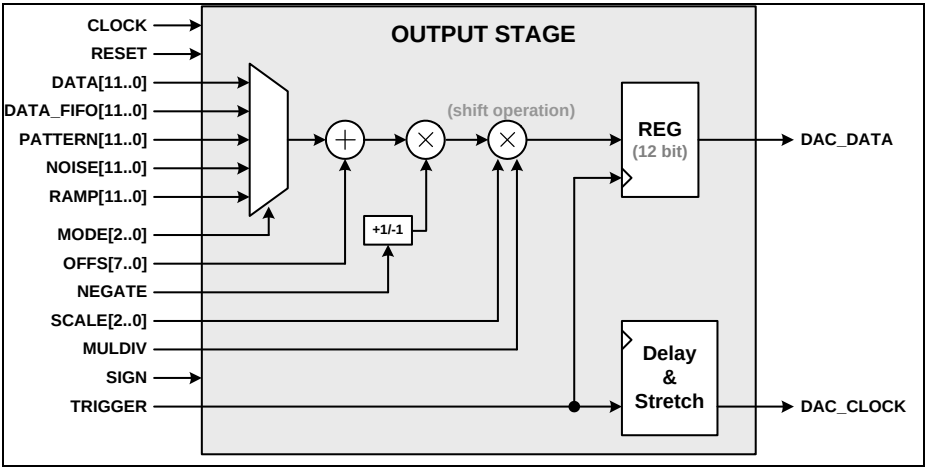


Figure 17-4 Data Output Stage Block Diagram

This output stage is the last element in the DAC's data path before the data is converted to the analog. It consists of a multiplexer, an adder, a multiplier, an output register and the generation of the DAC clock output.

The multiplexer selects between the five possible data sources and is programmed with the mode parameter. The adder stage gives the possibility to add an 8-bit offset value which is mainly needed for the PG mode in order to also process unsigned signal patterns to the DACs. In that case a certain offset value can be added to the signed output pattern values. The multiplier enables scaling by simple binary shifting of the data values. Therefore it allows multiplication and division by a programmed 2n scale value. Additionally the output value can be negated, i.e. converted to its two's complement value. These operations are possible in all functional operating modes. The output register contains the final sample delivered together with the corresponding trigger to the analog converter.

The clock output for operating the analog part of the DAC is generated using the DAC's trigger generator (TG). For that purpose the TG's trigger output is delayed by four system clock periods and stretched to a high-length of eight system clock periods.

17.2.1.4 Pattern Generators (PG) - Waveform Generator

The block diagram in **Figure 17-5** shows one of the two DAC's pattern generators.

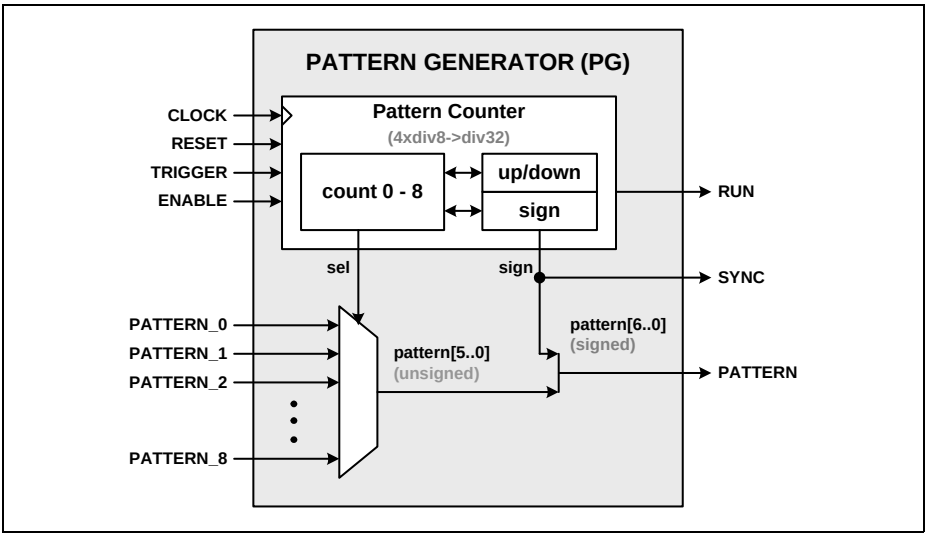


Figure 17-5 Pattern Generator Block Diagram

The nine pattern inputs on the left side of the PG block diagram are directly connected to the pattern registers described in **Section 17.6.3.4**. The pattern registers contain only

one quarter of the actual programmed periodic pattern. The output of the pattern counter in the PG is used to select one of the nine input patterns. This pattern counter is an up-down counter with an additional sign output which is inverted every time the counter reaches zero. Since the sign information is concatenated with the currently selected pattern, it is possible to generate a complete pattern sequence for a full period of any 2*π periodic waveform. For a detailed description how to operate the pattern generator please also refer to **Section 17.2.5**.

17.2.1.5 Noise Generators (NG) - Pseudo Random Number Generator

The block diagram in **Figure 17-6** shows one of the two DAC's noise generators.

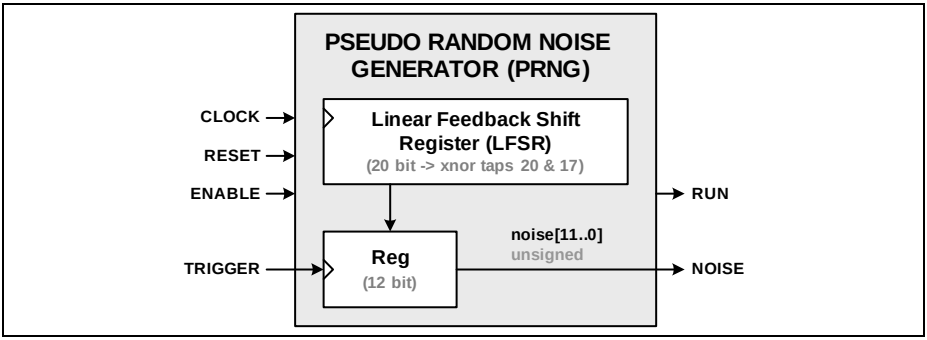


Figure 17-6 Noise Generator Block Diagram

The NG outputs a 12-bit pseudo random number. A 20-bit LFSR (linear feedback shift register) operating at the system clock frequency and a sample output register which is triggered by the NG's trigger inputs are used for this purpose. After enabling the NG, the LFSR is set back to its reset value and therefore it always starts outputting the same pseudo random number sequence.

17.2.1.6 Ramp Generators (RG)

The block diagram in **Figure 17-7** shows one of the two DAC's ramp generators.

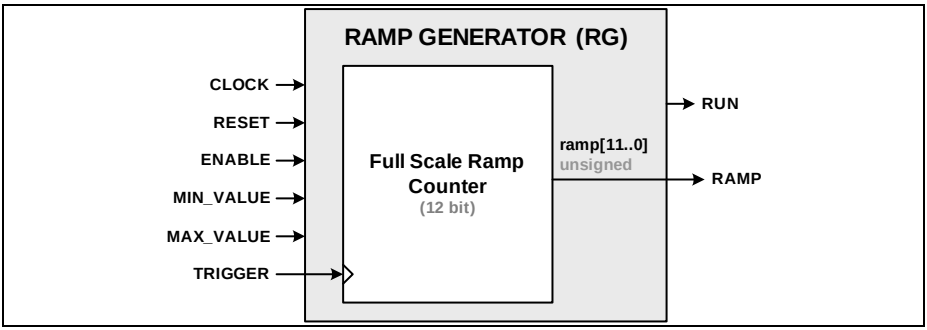


Figure 17-7 Ramp Generator Block Diagram

The ramp generator is basically an 12-bit up counter representing the full DAC range. If the RG is enabled it always starts at the programmed minimum value. The up-counting by ones is triggered by the selected trigger of the DAC channel. If the ramp counter reaches the programmed maximum value it restarts from the minimum value. This allows the generation of ramps within any desired value range and by variation of the trigger frequency also the ramp's slope can be modified.

17.2.2 Entering any Operating Mode

Before entering the desired operating mode with the **MODE** parameter, the corresponding analog DAC channel should be enabled with **ANAEN** and the startup time of the analog DAC channel should be considered.

Setting the **DATMOD** parameter to one enables simultaneous data processing. This means that both DAC channels use the same trigger source and both channels are always started and stopped synchronously. Hence the parameter setting of **MODE**, **TRIGMOD**, **TRIGSEL** and **FREQ** for DAC channel 0 is used for DAC channel 1 also.

17.2.3 Single Value Mode

By setting **MODE** to "single value mode", it is possible to convert only one single data value by the DACs. To start a conversion, a data value can be written to either **DATA0** or **DATA1** in **DAC0DATA** or **DAC1DATA** registers. This write operation itself then initiates a single trigger pulse and the value gets processed by DAC0 or DAC1. Only for this mode, no further external, internal or software trigger pulse is necessary. The DAC holds the processed value until a new value is written to **DATA0** or **DATA1**. This operating mode is intended for outputting static DAC output values. For processing sequential data streams in this "self triggered", single value mode, it is important not to exceed the maximum DAC data rate. If the **DATMOD** parameter is set to zero, data from the independent data registers **DAC0DATA** or **DAC1DATA** is processed. Whereas if

DATMOD is set to one, data from the simultaneous data register **DAC01DATA** is processed for both DACs.

17.2.4 Data Processing Mode

This operating mode is intended for continuous data processing from the system memory to DAC0 and/or DAC1. To enable it, the **MODE** parameter has to be set to data mode. Also, the desired trigger source has to be selected by setting **TRIGMOD**, **TRIGSEL** and **FREQ** in the configuration registers. The DAC can operate either with an internal generated trigger, one of the eight external trigger sources (see **Figure 17-2**) or the software trigger bit **SWTRIG**.

Simultaneous and independent Data Modes

With the **DATMOD** parameter, either the simultaneous or the independent “data mode” can be selected. In the simultaneous mode, both DACs receive their data from the same register **DAC01DATA**. In the independent “data mode” DAC0 gets its data from **DAC0DATA** and DAC1 from **DAC1DATA**. The two data paths are shown in **Figure 17-8** and also in **Figure 17-3**. Both DAC channels can activate a service request output signal to trigger a DMA channel, if they are configured in this mode and if the corresponding **SREN** bit is set to one. The service requests are DAC0.SR0 for the DAC0 channel and DAC0.SR1 for the DAC1 channel. For the simultaneous mode either DAC0.SR0 or DAC0.SR1 can be used by the DMA controller.

Start-Stop Operation

Before the DAC is started in “data mode”, all configuration registers DACx_CFG_x should be set according to the desired processing mode (see **Section 17.6.3.2**). Once this has been done, the DAC can be started by setting the **MODE** parameter to “data mode”. The control FSM will then start its operation until it reaches the run status indicated by the read parameter **RUN**. The run state can be left either by an operation error or by setting the **MODE** parameter to “disable DAC” again. An operation error can be a FIFO overflow or a FIFO underflow (see **Figure 17-3** and **Figure 17-8**).

17.2.4.1 FIFO Data Handling

Figure 17-8 shows the data handling for the FIFO of the DAC0 channel. Certainly the same structure also exists for the DAC1 channel (see **Figure 17-1** and **Figure 17-3** also). The data word **DATA0** from either data register **DAC0DATA** or **DAC01DATA** on the left hand side in **Figure 17-8** is loaded into one of the FIFO buffers registers. The load position in the FIFO depends on its actual filling level represented by the read parameters **FIFOIND**, **FIFOEMP** and **FIFOFUL**. If the FIFO is empty, **FIFOIND** = 0. If it is full, **FIFOIND** = 3. If a trigger occurs and the FIFO is not empty, the data is shifted to the next register (from left to right). At the same time, a service request (DAC0.SR0 or DAC0.SR1) is initiated in order to fill up the FIFO again. The service request should end

with a write operation to **DAC0DATA** or **DAC01DATA**. This write operation itself then triggers a write from the data registers to the FIFO buffer registers. If the FIFO stores only one last element (**FIFOIND** = 0 and **FIFOEMP** = 0) and a trigger has occurred, a service request is initiated and additionally **FIFOEMP** is set to 1. On the other hand, if there is only one last free register in the FIFO (**FIFOIND** = 2) and a write operation has been initiated, the **FIFOFUL** bit is set to 1. All the control signals for the FIFO handling are generated by the DAC's control FSM. This includes filling up the FIFO when "data mode" is entered and emptying the FIFO when leaving "data mode".

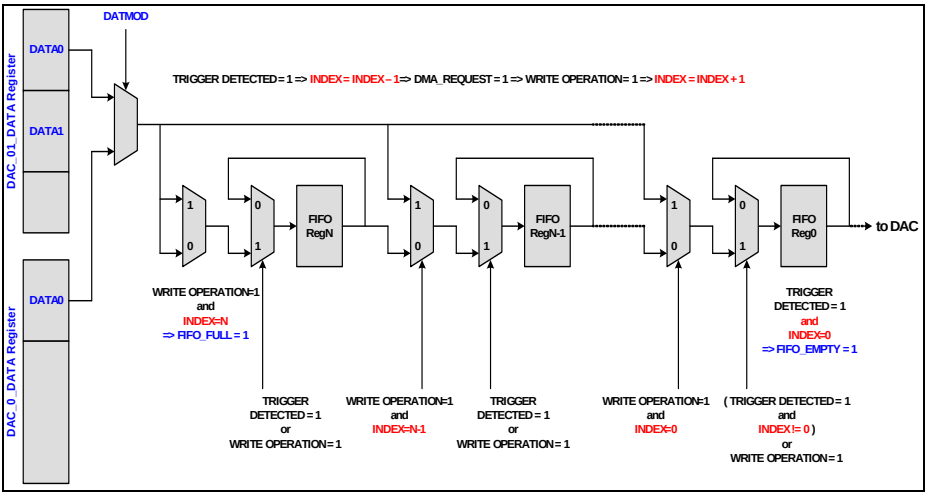


Figure 17-8 Data Handling for the FIFO Buffer

17.2.5 Pattern Generation Mode

This chapter describes the operation of the pattern generator. This mode is used to output a pattern or waveform to the DACs and it is activated by setting the **MODE** configuration parameter to "patgen mode".

Like in the "data mode" (see **Section 17.2.4**), the DAC in "patgen mode" can operate either with an internally generated trigger, one of the eight external trigger sources or a software trigger bit.

The desired pattern or waveform is freely programmable with the 5-bit parameters **PAT0** to **PAT8**. The pattern registers contain only one quadrant of the full waveform. The other quadrants of the 2*π periodic odd function are generated out of the first one with the help of a counter. The counter generates also the sign bit of the output pattern, therefore the 6-bit output signed values are in the range of -31 to +31.

In order to use the full range of the DAC in signed mode, a scaling by 32 by setting **SCALE** to "101" in the output stage is necessary . If the DAC should output a full range

pattern in unsigned mode, it is also possible to add an offset value programmed with **OFFS** to the output stage before doing the scaling. All the control signals for the pattern generators are generated by the DAC's control FSM.

address	lookup triangle	lookup sine	lookup rectangle
1	0	0	31
2	4	6	31
3	8	12	31
4	12	17	31
5	16	22	31
6	19	26	31
7	23	29	31
8	27	30	31
9	31	31	31

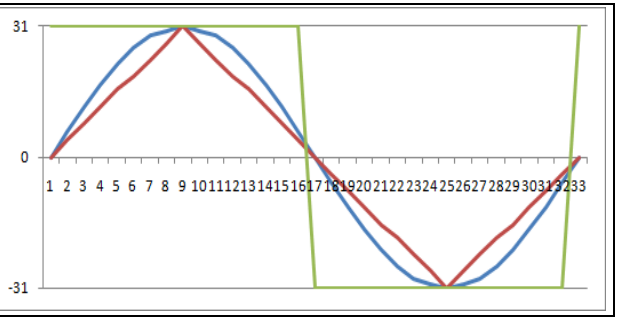


Figure 17-9 Example 5-bit Patterns and their corresponding Waveform Output

Figure 17-9 gives examples of lookup table entries for triangular, sine and rectangular pattern. These values can be programmed to **PAT0** to **PAT8** in order to get the corresponding waveforms at the DAC's output like shown in the chart on the right hand side. If the pattern generation is restarted / enabled again, it always starts with the first value of the first quarter of the actual programmed pattern (positive value and up-counting). The current sign information of the generated pattern is one of the DAC's system on chip outputs (see **Section 17.7.2.3**) and can be enabled using the parameter **SIGNEN**.

17.2.6 Noise Generation Mode

A 20-bit linear feedback shift register (LFSR) is used to produce a pseudo random number. In order to enable the noise generator, the **MODE** parameter has to be set to "noise mode". The LFSR itself runs with the system clock. The 12-bit random numbers is sampled with the preselected trigger source using **TRIGMOD**, **TRIGSEL** and optionally also **FREQ** or **SWTRIG** into an output register. The 12-bit values can be interpreted as signed or unsigned values. By setting the **MODE** parameter to "disable DAC" again, the noise generation stops and the DAC holds its last processed value. **Figure 17-10** shows an example pseudo random noise output. After restarting the noise generation mode, the LFSR is set back to its reset value and therefore it always starts outputting the same pseudo random number sequence.

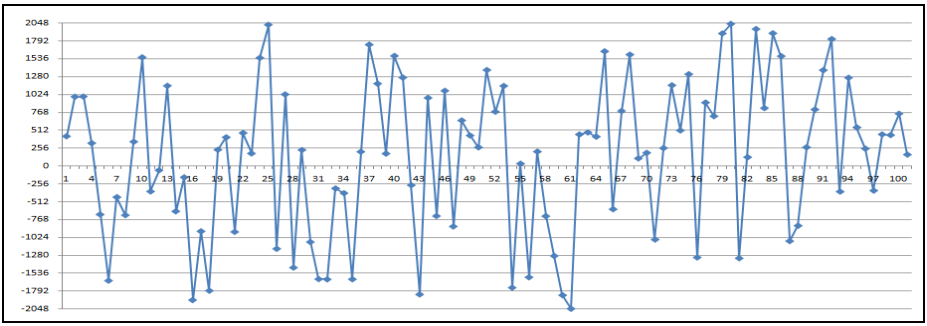


Figure 17-10 Signed 12-bit pseudo random Noise Example Output

17.2.7 Ramp Generation Mode

A 12-bit ramp counter is also part of the DAC. It is activated by setting the **MODE** parameter to “ramp mode”. The trigger source can be selected using **TRIGMOD**, **TRIGSEL** and optionally also **FREQ** or **SWTRIG**. The ramp counter starts at a programmed start value and each trigger pulse increments the counter by one. The start values are programmable via **DATA0** for DAC channel 0 and **DATA1** for DAC channel 1 of the independent data registers. The stop values are programmable via **DATA0** or **DATA1** of the simultaneous data register. If the ramp counter reaches its stop value, it restarts from the start value with the next trigger pulse. This allows the generation of ramps within any desired value range. The ramp’s slope can be modified by varying the trigger frequency. **Figure 17-11** shows two examples of ramp generation output waveforms.

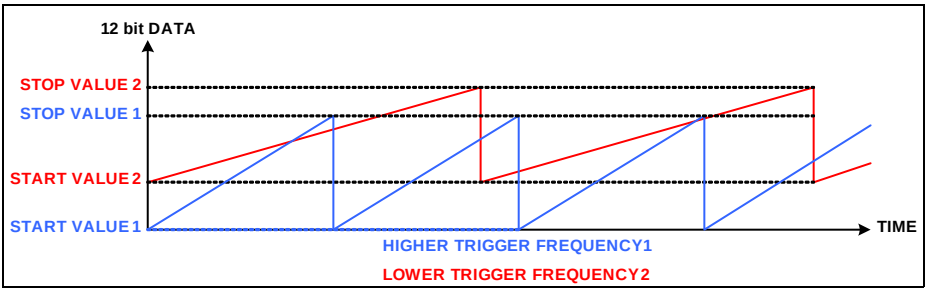


Figure 17-11 Unsigned 12-bit Ramp Generator Example Output

17.3 Service Request Generation

Service Requests are available in **Data Processing Mode** only.

17.4 Power, Reset and Clock

As long as **ANAEN** is set to default value “standby”, the corresponding DAC channel stays in power down mode, and its output is floating.

By setting **ANAEN** to “enable DAC”, the analog output is starting up; the max. possible startup time ($t_{STARTUP}$) is specified in the Data Sheet. Note, that in case the output is triggered before the startup time expired, the programed value is not passed to the output driver. The DAC output will then continue to drive the default value, till the first trigger event after elapsing the startup time. [1)]

The DAC module reset is shared with the peripheral bus reset line, as well as the module clock is shared with the peripheral bus clock line.

With **FREQ** the clock divider ratio of the internal trigger generator is set.

17.5 Initialization

A feasible initialization sequence of the DAC reads as follows:

1st Step: De-assert the reset of DAC module by setting DACRS bit in PRCLR1 register and disable gating by setting DAC bit in CGATCLR1 register.

2nd Step: Write the DACxCFG0 register values. Here you select the operating mode of the corresponding DAC channel by writing the **MODE** field, e.g. Patgen mode. By setting or clearing the **SIGN** bit, the choice between signed and unsigned input data format is made.

In the same step service request generation can be enabled with the **SREN** bit, as well as sign output with **SIGNEN** bit. Also the frequency divider of the internal trigger generator can be set up by writing the **FREQ** field.

3rd Step: Write the DACxCFG1 register values. Here you select the trigger source by writing the **TRIGMOD** field, e.g. software trigger. The DAC channel output is enabled by setting the **ANAEN** bit. You also need to choose now your values for **SCALE**, **MULDIV**, **OFFS**, and **DATMOD** fields.

4th Step: Configure the chosen data source. E.g. in case you selected Patgen mode, the pattern must be defined by programming DACxPATL and DACxPATH registers.

5th Step: E.g. in case software trigger is selected, the runtime code is responsible for generating the trigger signals by setting **SWTRIG** bit inside DACxCFG1 register. In this case, please mind the startup time ($t_{STARTUP}$) after setting **ANAEN** bit (see also **Section 17.4**).

C code example

The C code of this example is given below:

1) Please note, that the RUN status bit is set by hardware, before the startup time has expired.

```
void DAC_init()
{
//RESET MODULE
       SCU_RESET->PRCLR1 |= 0x00000020; //De-asserts reset for
VADC module
       SCU_CLK->CGATCLR1 |= SCU_CLK_CGATCLR1_DAC_Msk; //Disable
Gating
//DAC0 CONFIGURATION AS PATTERN GENERATOR
       DAC->DAC0CFG0=0x20300FFF;           //Pattern gen enable,
Signal enabled and Frequency
       DAC->DAC0CFG1=0x010405FD;           //Enable AN, Software
trigger
                                                  // Offset and
Scale divider set up.

         DAC->DAC0PATL=0x3568B0C0;              //Sinus  waveform
configuration
         DAC->DAC0PATH=0x00007FDD;              //Sinus  waveform
configuration
   //For triangle waveform use:
     //DAC->DAC0PATL=0x27062080;             //Triangle  waveform
configuration
     //DAC->DAC0PATH=0x00007F77;             //Triangle  waveform
configuration

//DAC1 CONFIGURATION AS RAMP GENERATOR
      DAC->DAC1CFG0=0x20500000;       //Ramp Mode, Signal enabled
and Frequency
       DAC->DAC1CFG1=0x01000000;    //Enable  AN.  No  offset  or
scaling. Internal Trigger

       DAC->DAC1DATA=0x00000003F;       //Start value
       DAC->DAC01DATA=0x0AFF0000;       //Stop value
}
-------------------------------------------------------------------------------------------
//For SW trigger in runtime code (after waiting startup time):

       DAC->DAC0CFG1|=0x00010000;      //Software Trigger of DAC0
```

17.6 Registers

17.6.1 Address Map

The DAC is available at the following base address:

Table 17-1 Registers Address Space

Module	Base Address	End Address	Note
DAC	4801 8000$_H$	4801 BFFF$_H$	16 kB

17.6.2 Register Overview

Table 17-2 shows all registers required for the operation of the DAC module:

Table 17-2 Register Overview of DAC

Short Name	Description	Offset Addr.[1)]	Access Mode		Description See
			Read	Write	
ID	Module Identification Register	000$_H$	U, PV, 32	U, PV, 32	Page 17-16
DAC0CFG0	DAC0 Configuration Register Number 0	004$_H$	U, PV, 32	U, PV, 32	Page 17-17
DAC0CFG1	DAC0 Configuration Register Number 1	008$_H$	U, PV, 32	U, PV, 32	Page 17-18
DAC1CFG0	DAC1 Configuration Register Number 0	00C$_H$	U, PV, 32	U, PV, 32	Page 17-20
DAC1CFG1	DAC1 Configuration Register Number 1	010$_H$	U, PV, 32	U, PV, 32	Page 17-22
DAC0DATA	Data Register for DAC0 for Independent Data Mode	014$_H$	U, PV, 32	U, PV, 32	Page 17-24
DAC1DATA	Data Register for DAC1 for Independent Data Mode	018$_H$	U, PV, 32	U, PV, 32	Page 17-25
DAC01DATA	Data Register for DAC0 and DAC1 for Simultaneous Data Mode	01C$_H$	U, PV, 32	U, PV, 32	Page 17-25
DAC0PATL	Lower Samples of Pattern for DAC0 PATGEN	020$_H$	U, PV, 32	U, PV, 32	Page 17-26

Table 17-2 Register Overview of DAC (cont'd)

Short Name	Description	Offset Addr.[1]	Access Mode		Description See
			Read	Write	
DAC0PATH	Higher Samples of Pattern for DAC0 PATGEN	024_H	U, PV, 32	U, PV, 32	Page 17-27
DAC1PATL	Lower Samples of Pattern for DAC1 PATGEN	028_H	U, PV, 32	U, PV, 32	Page 17-27
DAC1PATH	Higher Samples of Pattern for DAC1 PATGEN	$02C_H$	U, PV, 32	U, PV, 32	Page 17-28

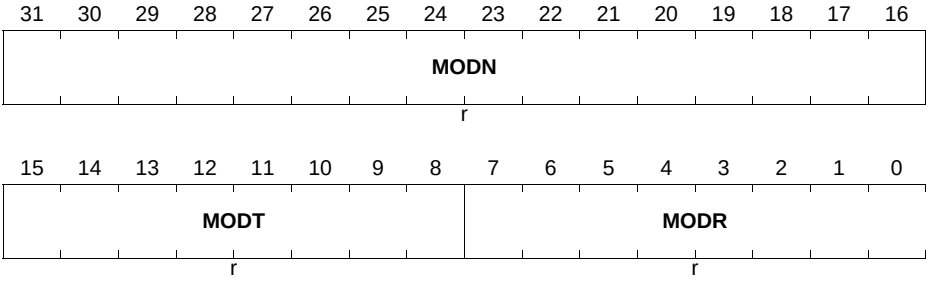
1) The absolute register address is calculated as follows:
Module Base Address + Offset Address (shown in this column)

17.6.3 Register Description

17.6.3.1 DAC_ID Register

The DAC module identification register contains the XMC4000 ID code.

DAC_ID
Module Identification Register **(000_H)** **Reset Value: 00A5 C0XX$_H$**



Field	Bits	Type	Description
MODR	[7:0]	r	**Module Revision** MOD_REV defines the module revision number. The value of a module revision starts with 01_H (first rev.).
MODT	[15:8]	r	**Module Type** This bit field is $C0_H$. It defines the module as a 32-bit module.

Field	Bits	Type	Description
MODN	[31:16]	r	**Module Number** For the DAC this bit field is $A5_H$

17.6.3.2 DAC Configuration Registers

The DAC configuration registers contain all the necessary bits to set DAC0 and DAC1 in the desired operating mode and to start and stop conversations.

DAC0CFG0
DAC0 Configuration Register 0 (004_H) Reset Value: 0000 0000_H

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RUN	SREN	SIGNEN	NEGATE	FIFOFUL	FIFOEMP	FIFOIND		SIGN	MODE			FREQ			
rh	rw	rw	r	rh	rh	rh		rw	rw			rw			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FREQ															
rw															

Field	Bits	Type	Description
FREQ	[19:0]	rw	**Integer Frequency Divider Value** • 0 to 16: divide by 16 • 16 to 2^20-1 divide by FREQ
MODE	[22:20]	rw	**Enables and Sets the Mode for DAC0** 000_B disable/switch-off DAC 001_B Single Value Mode 010_B Data Mode 011_B Patgen Mode 100_B Noise Mode 101_B Ramp Mode 110_B na 111_B na
SIGN	23	rw	**Selects Between Signed and Unsigned DAC0 Mode** 0_B DAC expects unsigned input data 1_B DAC expects signed input data
FIFOIND	[25:24]	rh	**Current write position inside the data FIFO**

Field	Bits	Type	Description
FIFOEMP	26	rh	**Indicate if the FIFO is empty** 0_B FIFO not empty 1_B FIFO empty
FIFOFUL	27	rh	**Indicate if the FIFO is full** 0_B FIFO not full 1_B FIFO full
NEGATE	28	rw	**Negates the DAC0 output** 0_B DAC output not negated 1_B DAC output negated Negation means the DAC value is converted to its two's complement value.
SIGNEN	29	rw	**Enable Sign Output of DAC0 Pattern Generator** 0_B Disable 1_B Enable
SREN	30	rw	**Enable DAC0 service request interrupt generation** 0_B disable 1_B enable
RUN	31	rh	**RUN indicates the current DAC0 operation status** 0_B DAC0 channel disabled 1_B DAC0 channel in operation RUN is set/cleared by hardware.

DAC0CFG1
DAC0 Configuration Register 1 **(008_H)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
REFCFGL				0			ANA EN		ANACFG				TRIGMOD		SWT RIG
rw				r			rw		rw				rw		rwh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DAT MOD	TRIGSEL			OFFS								MUL DIV	SCALE		
rw	rw			rw								rw	rw		

Field	Bits	Type	Description
SCALE	[2:0]	rw	**Scale value for up- or downscale of the DAC0 input data in steps by the power of 2 (=shift operation)** 000_B no shift = multiplication/division by 1 001_B shift by 1 = multiplication/division by 2 010_B shift by 2 = multiplication/division by 4 011_B shift left by 3 = multiplication/division by 8 100_B shift left by 4 = multiplication/division by 16 101_B shift left by 5 = multiplication/division by 32 110_B shift left by 6 = multiplication/division by 64 111_B shift left by 7 = multiplication/division by 128
MULDIV	3	rw	**Switch between up- and downscale of the DAC0 input data values** 0_B downscale = division (shift SCALE positions to the right) 1_B upscale = multiplication (shift SCALE positions to the left)
OFFS	[11:4]	rw	**8-bit offset value addition** e.g.: PATGEN output is a sine wave -31 to +31 and OFFS = 31 => the DAC0 input data will be a sine wave with an amplitude between 0 and 62. Depending on the SIGN bit this value is interpreted as signed or unsigned.
TRIGSEL	[14:12]	rw	**Selects one of the eight external trigger sources for DAC0**
DATMOD	15	rw	**Switch between independent or simultaneous DAC mode and select the input data register for DAC0 and DAC1** 0_B independent data handling - process data from DATA0 register (bits 11:0) to DAC0 and data from DATA1 register (bits 11:0) to DAC1 1_B simultaneous data handling - process data from DAC01 register to both DACs (bits 11:0 to DAC0 and bits 23:12 to DAC1). Trigger setting and MODE parameter for DAC0 is used for DAC1 also if DATMOD is set to 1 = simultaneous data mode!

Field	Bits	Type	Description
SWTRIG	16	rwh	**Software Trigger** Triggers DAC channel 0 if TRIGMOD is set to 10. Setting the bit to 1 generates one trigger pulse. The bit is cleared (set to 0) automatically. If DATMOD is set to simultaneous data mode this bit is used for both DAC channels (see DATMOD parameter).
TRIGMOD	[18:17]	rw	**Select the trigger source for channel 0** 00_B internal Trigger (integer divided clock - see FREQ parameter) 01_B external Trigger (preselected trigger by TRIGSEL parameter) 10_B software Trigger (see SWTRIG parameter) 11_B reserved
ANACFG	[23:19]	rw	**DAC0 analog configuration/calibration parameters** reserved for future use
ANAEN	24	rw	**Enable analog DAC for channel 0** 0_B DAC0 is set to standby (analog output only) 1_B enable DAC0 (analog output only)
0	[27:25]	r	**Reserved** Read as 0; Should be written with 0.
REFCFGL	[31:28]	rw	**Lower 4 band-gap configuration/calibration parameters** reserved for future use

DAC1CFG0
DAC1 Configuration Register 0 **($00C_H$)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RUN	**SREN**	**SIGNEN**	**NEGATE**	**FIFOFUL**	**FIFOEMP**	**FIFOIND**		**SIGN**	**MODE**			**FREQ**			
rh	rw	rw	r	rh	rh	rh		rw	rw			rw			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FREQ															
rw															

Field	Bits	Type	Description
FREQ	[19:0]	rw	**Integer Frequency Divider Value** • 0 to 16: divide by 16 • 16 to 2^20-1 divide by FREQ FREQ for DAC1 is not applicable if DATMOD is set to 1 = simultaneous data mode.
MODE	[22:20]	rw	**Enables and sets the Mode for DAC1** 000_B disable/switch-off DAC 001_B Single Value Mode 010_B Data Mode 011_B Patgen Mode 100_B Noise Mode 101_B Ramp Mode 110_B na 111_B na MODE for DAC1 is not applicable if DATMOD is set to 1 = simultaneous data mode.
SIGN	23	rw	**Selects between signed and unsigned DAC1 mode** 0_B DAC expects unsigned input data 1_B DAC expects signed input data
FIFOIND	[25:24]	rh	**Current write position inside the data FIFO**
FIFOEMP	26	rh	**Indicate if the FIFO is empty** 0_B FIFO not empty 1_B FIFO empty
FIFOFUL	27	rh	**Indicate if the FIFO is full** 0_B FIFO not full 1_B FIFO full
NEGATE	28	rw	**Negates the DAC1 output** 0_B DAC output not negated 1_B DAC output negated Negation means the DAC value is converted to its two’s complement value.
SIGNEN	29	rw	**Enable sign output of DAC1 pattern generator** 0_B disable 1_B enable

Field	Bits	Type	Description
SREN	30	rw	**Enable DAC1 service request interrupt generation** 0_B disable 1_B enable
RUN	31	rh	**RUN indicates the current DAC1 operation status** 0_B DAC1 channel disabled 1_B DAC1 channel in operation RUN is set/cleared by hardware.

DAC1CFG1
DAC1 Configuration Register 1 (010_H) Reset Value: $0000\ 0000_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
REFCFGH				0			ANA EN	ANACFG					TRIGMOD		SWT RIG
rw				r			rw	rw					rw		rwh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	TRIGSEL			OFFS								MUL DIV	SCALE		
r	rw			rw								rw	rw		

Field	Bits	Type	Description
SCALE	[2:0]	rw	**Scale value for up- or downscale of the DAC1 input data in steps by the power of 2 (=shift operation)** 000_B no shift = multiplication/division by 1 001_B shift by 1 = multiplication/division by 2 010_B shift by 2 = multiplication/division by 4 011_B shift left by 3 = multiplication/division by 8 100_B shift left by 4 = multiplication/division by 16 101_B shift left by 5 = multiplication/division by 32 110_B shift left by 6 = multiplication/division by 64 111_B shift left by 7 = multiplication/division by 128

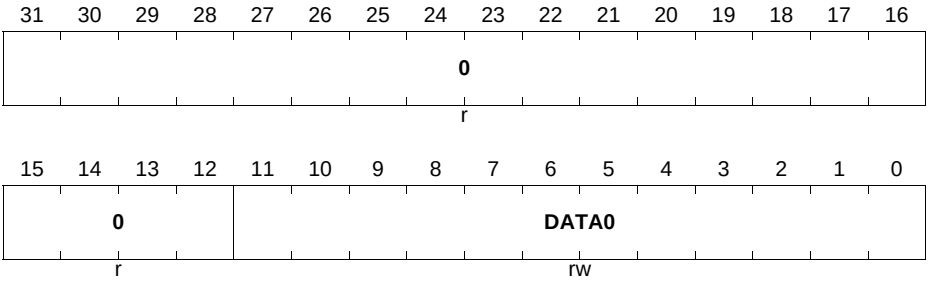
Field	Bits	Type	Description
MULDIV	3	rw	**Switch between up- and downscale of the DAC1 input data values** 0_B downscale = division (shift SCALE positions to the right) 1_B upscale = multiplication (shift SCALE positions to the left)
OFFS	[11:4]	rw	**8-bit offset value addition** e.g.: PATGEN output is a sine wave -31 to +31 and OFFS = 31 => the DAC1 input data will be a sine wave with an amplitude between 0 and 62. Depending on the SIGN bit this value is interpreted as signed or unsigned.
TRIGSEL	[14:12]	rw	**Selects one of the eight external trigger sources for DAC1** TRIGSEL for DAC1 is not applicable if DATMOD is set to 1 = simultaneous data mode.
0	15	r	**Reserved** Read as 0; Should be written with 0.
SWTRIG	16	rwh	**Software Trigger** Triggers DAC channel 1 if TRIGMOD is set to 10. Setting the bit to 1 generates one trigger pulse. The bit is cleared (set to 0) automatically. If DATMOD is set to simultaneous data mode (see DATMOD parameter) this bit is not applicable and the SWTRIG bit from channel 0 is used for channel 1 also.
TRIGMOD	[18:17]	rw	**Select the trigger source for channel 1** 00_B internal Trigger (integer divided clock - see FREQ parameter) 01_B external Trigger (preselected trigger by TRIGSEL parameter) 10_B software Trigger (see SWTRIG parameter) 11_B reserved
ANACFG	[23:19]	rw	**DAC1 analog configuration/calibration parameters** reserved for future use
ANAEN	24	rw	**Enable analog DAC for channel 1** 0_B DAC1 is set to standby (analog output only) 1_B enable DAC1 (analog output only)

Field	Bits	Type	Description
0	[27:25]	r	**Reserved** Read as 0; Should be written with 0.
REFCFGH	[31:28]	rw	**Higher 4 band-gap configuration/calibration parameters** reserved for future use

17.6.3.3 DAC Data Registers

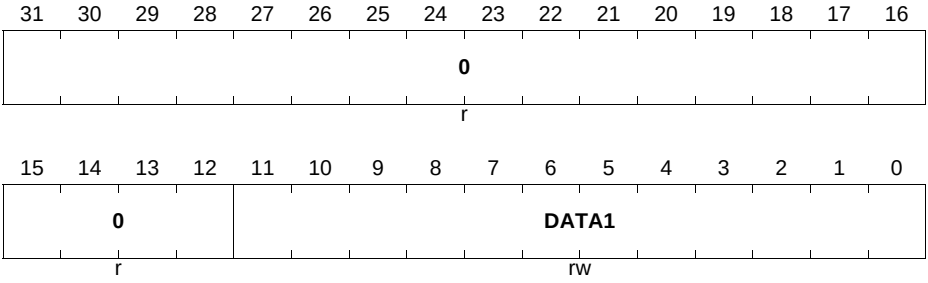
The DAC data registers contain the data provided to DAC0 and DAC1 either in simultaneous data mode (DAC01DATA) or in independent data mode (DAC0DATA and DAC1DATA).

DAC0DATA
DAC0 Data Register **(014_H)** **Reset Value: 0000 0000$_H$**



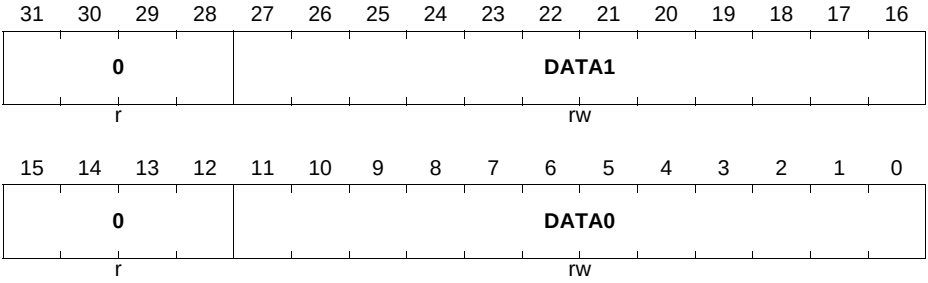
Field	Bits	Type	Description
DATA0	[11:0]	rw	**DAC0 Data Bits** Used as DAC0 data value and as counter start value in ramp generation mode
0	[31:12]	r	**Reserved** Read as 0; Should be written with 0.

DAC1DATA
DAC1 Data Register **(018_H)** **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
DATA1	[11:0]	rw	**DAC1 Data Bits** Used as DAC1 data value and as counter start value in ramp generation mode
0	[31:12]	r	**Reserved** Read as 0; Should be written with 0.

DAC01DATA
DAC01 Data Register **($01C_H$)** **Reset Value: 0000 0000$_H$**



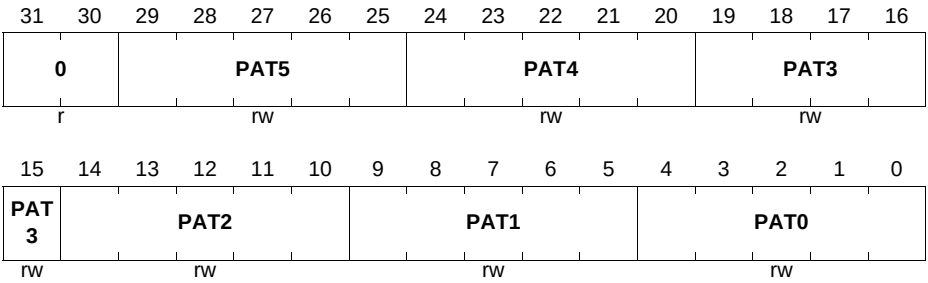
Field	Bits	Type	Description
DATA0	[11:0]	rw	**DAC0 Data Bits** Used as DAC0 data value and as counter stop value in ramp generation mode

Field	Bits	Type	Description
0	[15:12]	r	**Reserved** Read as 0; Should be written with 0.
DATA1	[27:16]	rw	**DAC1 Data Bits** Used as DAC1 data value and as counter stop value in ramp generation mode
0	[31:28]	r	**Reserved** Read as 0; Should be written with 0.

17.6.3.4 DAC Pattern Registers

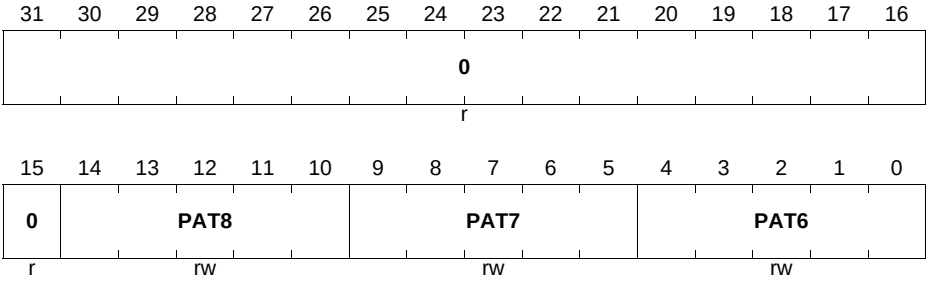
The DAC pattern registers contain the waveform patterns for the pattern generators of DAC0 and DAC1.

DAC0PATL
DAC0 Lower Pattern Register **(020_H)** **Reset Value: 3568 B0C0$_H$**



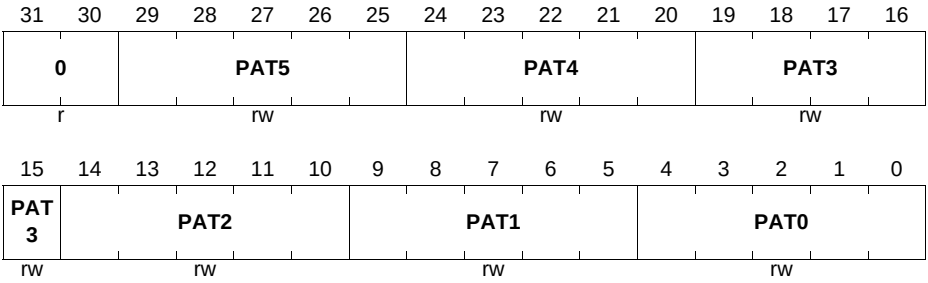
Field	Bits	Type	Description
PAT0	[4:0]	rw	**Pattern Number 0 for PATGEN of DAC0**
PAT1	[9:5]	rw	**Pattern Number 1 for PATGEN of DAC0**
PAT2	[14:10]	rw	**Pattern Number 2 for PATGEN of DAC0**
PAT3	[19:15]	rw	**Pattern Number 3 for PATGEN of DAC0**
PAT4	[24:20]	rw	**Pattern Number 4 for PATGEN of DAC0**
PAT5	[29:25]	rw	**Pattern Number 5 for PATGEN of DAC0**
0	[31:30]	r	**Reserved** Read as 0; Should be written with 0

DAC0PATH
DAC0 Higher Pattern Register **(024_H)** **Reset Value: 0000 7FDD$_H$**



Field	Bits	Type	Description
PAT6	[4:0]	rw	**Pattern Number 6 for PATGEN of DAC0**
PAT7	[9:5]	rw	**Pattern Number 7 for PATGEN of DAC0**
PAT8	[14:10]	rw	**Pattern Number 8 for PATGEN of DAC0**
0	[31:15]	r	**Reserved** Read as 0; Should be written with 0.

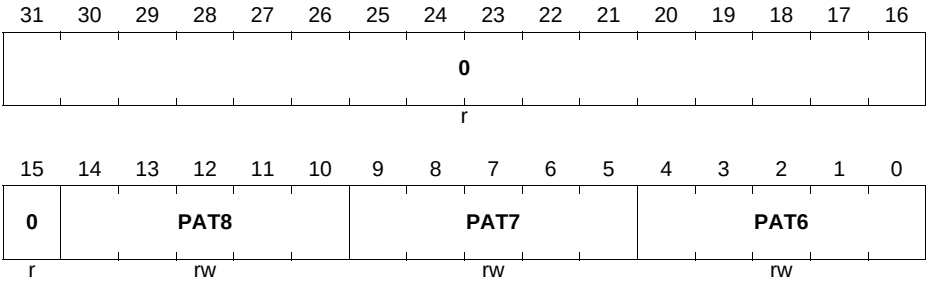
DAC1PATL
DAC1 Lower Pattern Register **(028_H)** **Reset Value: 3568 B0C0$_H$**



Field	Bits	Type	Description
PAT0	[4:0]	rw	**Pattern Number 0 for PATGEN of DAC1**
PAT1	[9:5]	rw	**Pattern Number 1 for PATGEN of DAC1**

Field	Bits	Type	Description
PAT2	[14:10]	rw	**Pattern Number 2 for PATGEN of DAC1**
PAT3	[19:15]	rw	**Pattern Number 3 for PATGEN of DAC1**
PAT4	[24:20]	rw	**Pattern Number 4 for PATGEN of DAC1**
PAT5	[29:25]	rw	**Pattern Number 5 for PATGEN of DAC1**
0	[31:30]	r	**Reserved** Read as 0; Should be written with 0.

DAC1PATH
DAC1 Higher Pattern Register (02C$_H$) Reset Value: 0000 7FDD$_H$



Field	Bits	Type	Description
PAT6	[4:0]	rw	**Pattern Number 6 for PATGEN of DAC1**
PAT7	[9:5]	rw	**Pattern Number 7 for PATGEN of DAC1**
PAT8	[14:10]	rw	**Pattern Number 8 for PATGEN of DAC1**
0	[31:15]	r	**Reserved** Read as 0; Should be written with 0.

17.7 Interconnects

17.7.1 Analog Connections

The analog interface lines of the DAC are listed below:

Table 17-3 Analog Connections

Input/Output	I/O	Connected To	Descriptions
DAC.OUT_0	O	P14.8	Analog output of channel 0
DAC.OUT_1	O	P14.9	Analog output of channel 1

17.7.2 Digital Connections

The DAC has the following system level connections to other modules:

17.7.2.1 Service Request Connections

Two service requests DAC.SR0 and DAC.SR1 are used for simultaneous and independent data mode. DAC.SR1 can be enabled on DMA channel 2 and DAC.SR0 can be enabled on DMA channel 3.

Table 17-4 Service Request Connections

Input/Output	I/O	Connected To	Descriptions
DAC.SR0	O	NVIC GPDMA	Service request
DAC.SR1	O	NVIC GPDMA	Service request

17.7.2.2 Trigger Connections

The eight trigger inputs are connected to the following sources:

Table 17-5 Trigger Connections

Input/Output	I/O	Connected To	Descriptions
DAC.TRIGGER[0]	I	CCU80.SR1	Trigger
DAC.TRIGGER[1]	I	reserved	Trigger
DAC.TRIGGER[2]	I	CCU40.SR1	Trigger
DAC.TRIGGER[3]	I	CCU41.SR1	Trigger
DAC.TRIGGER[4]	I	Port	Trigger
DAC.TRIGGER[5]	I	Port	Trigger
DAC.TRIGGER[6]	I	U0C0.DX1INS	Trigger
DAC.TRIGGER[7]	I	U1C0.DX1INS	Trigger

17.7.2.3 Synchronization Interface of the Pattern Generator

The interface consists of only two output signals called “DAC.SIGN_0” and “DAC.SIGN_1”. They are generated by the pattern generator of the two DAC channels and represent the actual sign information of the processed signal waveform converted by the DACs (see **Figure 17-5**).

Table 17-6 Pattern Generator Synchronization Connections

Input/Output	I/O	Connected To	Descriptions
DAC.SIGN_0	O	VADC.G0REQGTI VADC.BGREQGTI ERU1.0A3	
DAC.SIGN_1	O	VADC.G1REQGTI ERU1.2A3	

Industrial Control Peripherals

18 Capture/Compare Unit 4 (CCU4)

The CCU4 peripheral is a major component for systems that need general purpose timers for signal monitoring/conditioning and Pulse Width Modulation (PWM) signal generation. Power electronic control systems like switched mode power supplies or uninterruptible power supplies, can easily be implemented with the functions inside the CCU4 peripheral.

The internal modularity of CCU4, translates into a software friendly system for fast code development and portability between applications.

Table 18-1 Abbreviations table

PWM	Pulse Width Modulation
CCU4x	Capture/Compare Unit 4 module instance x
CC4y	Capture/Compare Unit 4 Timer Slice instance y
ADC	Analog to Digital Converter
POSIF	Position Interface peripheral
SCU	System Control Unit
f_{ccu4}	CCU4 module clock frequency
f_{tclk}	CC4y timer clock frequency

Note: A small “y” or “x” letter in a register indicates an index

18.1 Overview

Each CCU4 module is comprised of four identical 16 bit Capture/Compare Timer slices, CC4y. Each timer slice can work in compare mode or in capture mode. In compare mode one compare channel is available while in capture mode, up to four capture registers can be used in parallel.

Each CCU4 module has four service request lines and each timer slice contains a dedicated output signal, enabling the generation of up to four independent PWM signals.

Straightforward timer slice concatenation is also possible, enabling up to 64 bit timing operations. This offers a flexible frequency measurement, frequency multiplication and pulse width modulation scheme.

A programmable function input selector for each timer slice, that offers up to nine functions, discards the need of complete resource mapping due to input ports availability.

A built-in link between the CCU4 and several other modules enable flexible digital motor control loops implementation, e.g. with Hall Sensor monitoring or direct coupling with Encoders.

18.1.1 Features

CCU4 module features

Each CCU4 represents a combination of four timer slices, that can work independently in compare or capture mode. Each timer slice has a dedicated output for PWM signal generation.

All four CCU4 timer slices, CC4y, are identical in terms of available functions and operating modes. Avoiding this way the need of implementing different software routines, depending on which resource of CCU4 is used.

A built-in link between the four timer slices is also available, enabling this way a simplified timer concatenation and sequential operations.

General Features

- 16 bit timer cells
- capture and compare mode for each timer slice
  - four capture registers in capture mode
  - one compare channel in compare mode
- programmable low pass filter for the inputs
- built-in timer concatenation
  - 32, 48 or 64 bit width
- shadow transfer for the period and compare values
- programmable clock prescaler
- normal timer mode
- gated timer mode
- three counting schemes
  - center aligned
  - edge aligned
  - single shot
- PWM generation
- TRAP function
- start/stop can be controlled by external events
- counting external events
- four dedicated service request lines per CCU4

Additional features

- external modulation function
- load controlled by external events
- dithering PWM
- floating point pre scaler
- output state override by an external event
- suitable and flexible connectivity to several modules:
  - motor and power conversion applications
  - high number of signal conditioning possibilities

CCU4 features vs. applications

On **Table 18-2** a summary of the major features of the CCU4 unit mapped with the most common applications.

Table 18-2 Applications summary

Feature	Applications
Four independent timer cells	Independent PWM generation: • Multiple buck/boost converter control (with independent frequencies) • Different modes of operation for each timer, increasing the resource optimization • Up to 2 Half-Bridges control • multiple Zero Voltage Switch (ZVS) converter control with easy link to the ADC channels.
Concatenated timer cells	Easy to configure timer extension up to 64 bit: • High dynamic trigger capturing • High dynamic signal measurement
Dithering PWM	Generating a fractional PWM frequency or duty cycle: • To avoid big steps on frequency or duty cycle adjustment in slow control loop applications • Increase the PWM signal resolution over time
Floating prescaler	Automated control signal measurement: • decrease SW activity for monitoring signals with high or unknown dynamics • emulating more than a 16 bit timer for system control
Up to 9 functions via external signals for each timer	Flexible resource optimization: • The complete set of external functions is always available • Several arrangements can be done inside a CCU4, e.g., one timer working in capture mode and one working in compare

Table 18-2 Applications summary (cont'd)

Feature	Applications
4 dedicated service request lines	Specially developed for: • generating interrupts for the microprocessor • flexible connectivity between peripherals, e.g. ADC triggering.
Linking with other modules	Flexible profiles for: • Hall Sensor feedback/monitoring • Motor Encoders feedback/monitoring • PWM parallel modulation • Flexible signal conditioning

18.1.2 Block Diagram

Each CCU4 timer slice can operate independently from the other slices for all the available modes. Each timer slice contains a dedicated input selector for functions linked with external events and has a dedicated compare output signal, for PWM signal generation.

The built-in timer concatenation is only possible with adjacent slices, e.g. CC40/CC41. Combinations for slice concatenations like, CC40/CC42 or CC40/CC43 are not possible.

The individual service requests for each timer slice (four per slice) are multiplexed into four module service requests lines, **Figure 18-1**.

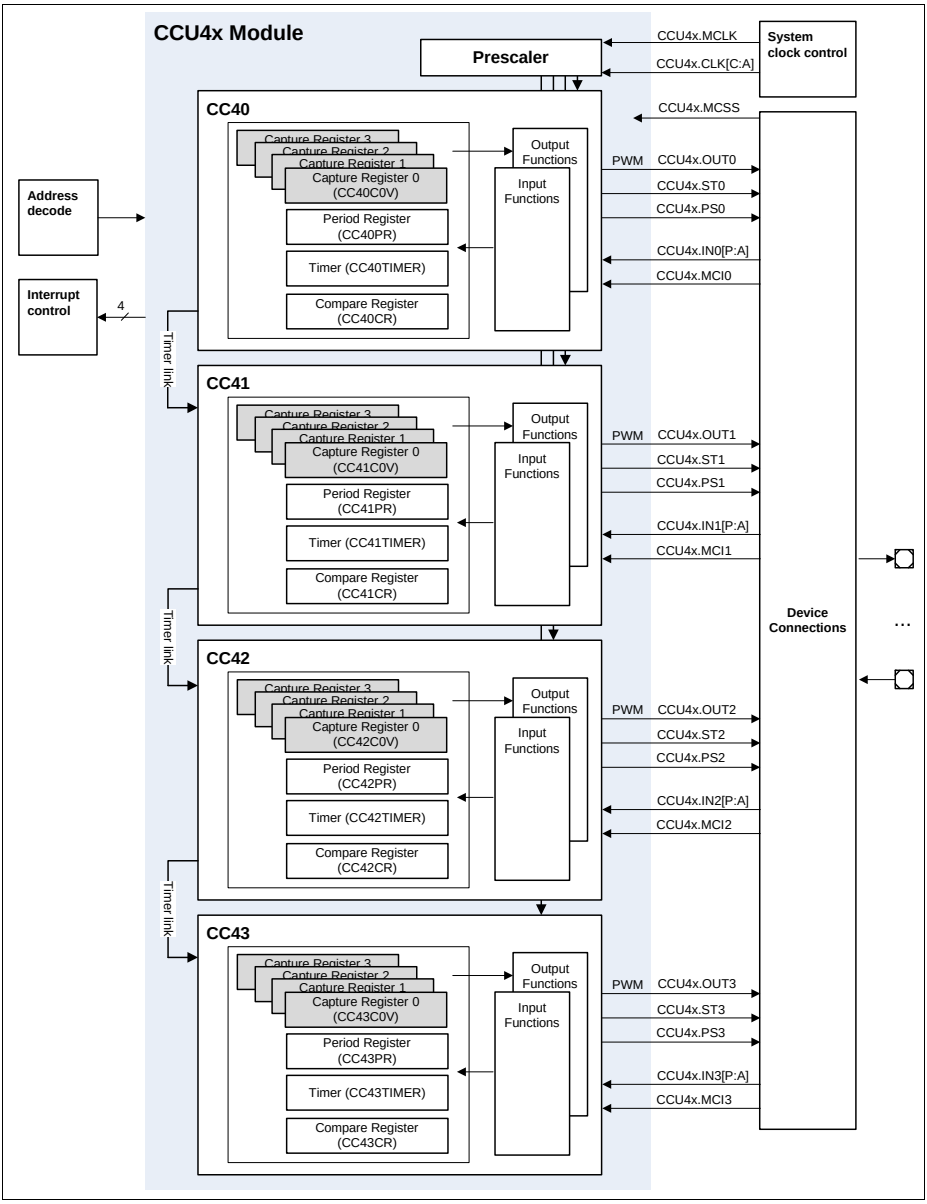


Figure 18-1 CCU4 block diagram

18.2 Functional Description

18.2.1 CC4y Overview

The input path of a CCU4 slice is comprised of a selector (**Section 18.2.2**) and a connection matrix unit (**Section 18.2.3**). The output path contains a service request control unit, a timer concatenation unit and two units that control directly the state of the output signal for each specific slice (for TRAP and modulation handling), see **Figure 18-2**.

The timer core is built of a 16 bit counter one period and one compare register in compare mode, or up to four capture registers in capture mode.

In compare mode the period register sets the maximum counting value while the compare channel is controlling the ACTIVE/PASSIVE state of the dedicated comparison slice output.

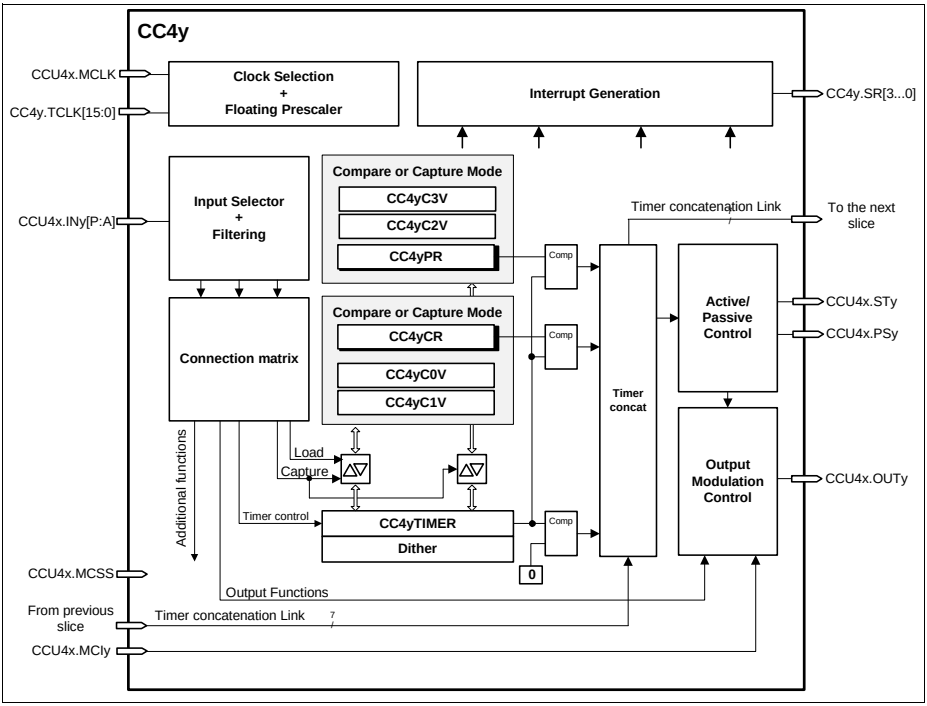


Figure 18-2 CCU4 slice block diagram

Each CCU4 slice, with the exception of the first, contains six dedicated inputs outputs that are used for the built-in timer concatenation functionality.

Inputs and outputs that are not seen at the CCU4 boundaries have a nomenclature of CC4y.<name>, whilst CCU4 module inputs and outputs are described as CCU4x.<signal_name>y (indicating the variable y the object slice).

Table 18-3 CCU4 slice pin description

Pin	I/O	Description
CCU4x.MCLK	I	Module clock
CC4y.TCLK[15:0]	I	Clocks from the pre scaler
CCU4x.INy[P:A]	I	Slice functional inputs (used to control the functionality throughout slice external events)
CCU4x.MCIy	I	Multi Channel mode input
CCU4x.MCSS	I	Multi Channel shadow transfer trigger
CC4y.SR[3...0]	O	Slice service request lines
CC4x.STy	O	Slice comparison status value
CCU4x.PSy	O	Multi channel pattern update trigger
CCU4x.OUTy	O	Slice dedicated output pin

Note:

2. *The status bit outputs of the Kernel, CCU4x.STy, are extended for one more kernel clock cycle.*
3. *The Service Request signals at the output of the kernel are extended for one more kernel clock cycle.*
4. *The maximum output signal frequency of the CCU4x.STy outputs is module clock divided by 4.*

The slice timer, can count up or down depending on the selected operating mode. A direction flag holds the actual counting direction.

The timer is connected to two stand alone comparators, one for the period match and one for a compare match. The registers used for period match and comparison match can be programmed to serve as capture registers enabling sequential capture capabilities on external events.

In normal edge aligned counting scheme, the counter is cleared to 0000_H each time that matches the period value defined in the period register. In center aligned mode, the counter direction changes from ‘up counting’ to ‘down counting’ after reaching the period

value. Both period and compare registers have an aggregated shadow register, which enables the update of the PWM period and duty cycle on the fly.

A single shot mode is also available, where the counter stops after it reaches the value set in the period register.

The start and stop of the counter can be controlled via software access or by a programmable input pin.

Functions like, load, counting direction (up/down), TRAP and output modulation can also be controlled with external events, see **Section 18.2.3**.

18.2.2 Input Selector

The first unit of the slice input path, is used to select which inputs are used to control the available external functions.

Inside this block the user also has the possibility to perform a low pass filtering of the signals and selecting the active edge(s) or level of the external event, see **Figure 18-3**.

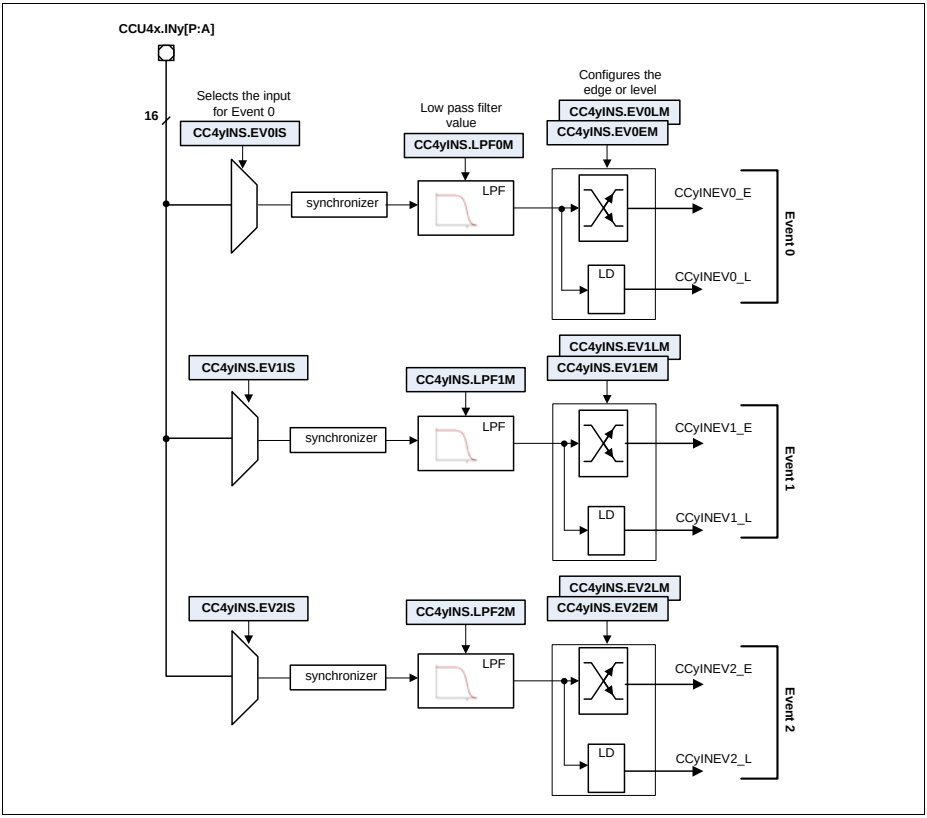


Figure 18-3 Slice input selector diagram

The user has the possibility of selecting any of the CCU4x.INy[P:A] inputs as the source of an event.

At the output of this unit we have a user selection of three events, that were configured to be active at rising, falling or both edges, or level active. These selected events can then be mapped to several functions.

Notice that each decoded event contains two outputs, one edge active and one level active, due to the fact that some functions like counting, capture or load are edge sensitive events while, timer gating or up down counting selection are level active.

18.2.3 Connection Matrix

The connection matrix maps the events coming from the input selector to several user configured functions, **Figure 18-4**.The following functions can be enabled on the connection matrix:

Table 18-4 Connection matrix available functions

Function	Brief description	Map to figure Figure 18-4
Start	Edge signal to start the timer	CCystrt
Stop	Edge signal to stop the timer	CCystp
Count	Edge signal used for counting events	CCycnt
Up/down	Level signal used to select up or down counting direction	CCyupd
Capture 0	Edge signal that triggers a capture into the capture registers 0 and 1	CCycapt0
Capture 1	Edge signal that triggers a capture into the capture register 2 and 3	CCycapt1
Gate	Level signal used to gate the timer clock	CCygate
Load	Edge signal that loads the timer with the value present at the compare register	CCyload
TRAP	Level signal used for fail-safe operation	CCytrap
Modulation	Level signal used to modulate/clear the output	CCymod
Status bit override	Status bit is going to be overridden with an input value	CCyoval for the value CCyoset for the trigger

Inside the connection matrix we also have a unit that performs the built-in timer concatenation. This concatenation enables a completely synchronized operation between the concatenated slices for timing operations and also for capture and load actions. The timer slice concatenation is done via the **CC4yCMC**.TCE bitfield. For a complete description of the concatenation function, please address **Section 18.2.9**.

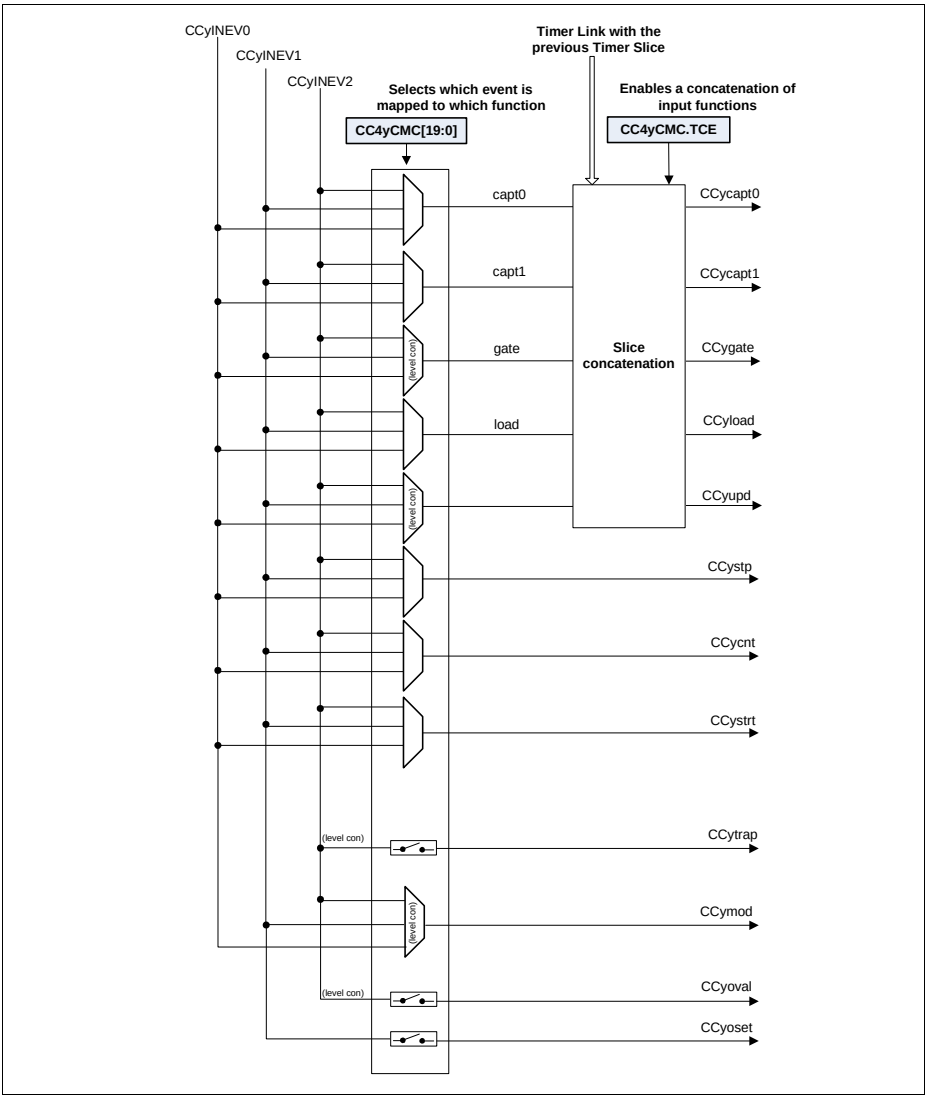


Figure 18-4 Slice connection matrix diagram

18.2.4 Starting/Stopping the Timer

Each timer slice contains a run bit register that indicates the actual status of the timer, **CC4yTCST**.TRB. The start and stop of the timer can be done via software access or can be controlled directly by external events, see **Figure 18-5**.

Selecting an external signal that acts as a start trigger does not force the user to use an external stop trigger and vice versa.

Selecting the single shot mode, imposes that after the counter reaches the period value the run bit, **CC4yTCST**.TRB, is going to be cleared and therefore the timer is stopped.

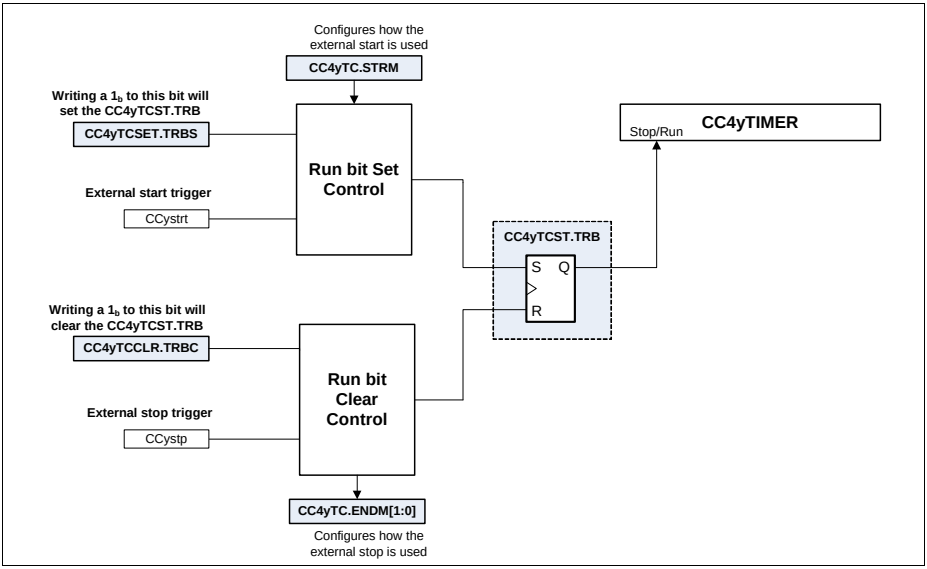


Figure 18-5 Timer start/stop control diagram

One can use the external stop signal to perform the following functions (configuration via **CC4yTC**.ENDM):

- Clear the run bit (stops the timer) - default
- Clear the timer (to 0000_H) but it does not clear the run bit (timer still running)
- Clear the timer and the run bit

One can use the external start to perform the following functions (configuration via **CC4yTC**.STRM):

- Start the timer (resume operation)
- Clear and start the timer

The set (start the timer) of the timer run bit, always has priority over a clear (stop the timer).

To start multiple CCU4 timers at the same time/synchronously one should use a dedicated input as external start (see **Section 18.2.7.1** for a description how to configure an input as start function). This input should be connected to all the Timers that need to started synchronously (see **Section 18.8** for a complete list of module connections), **Figure 18-6**.

For starting the timers synchronously via software there is a dedicated input signal, controlled by the SCU (System Control Unit), that is connected to all the CCU4 timers. This signal should then be configured as an external start signal (see **Section 18.2.7.1**) and then the software must write a $1_B/0_B$ (depending on the configuration of the external start signal) to the specific bitfield of the CCUCON register (this register is described on the SCU chapter).

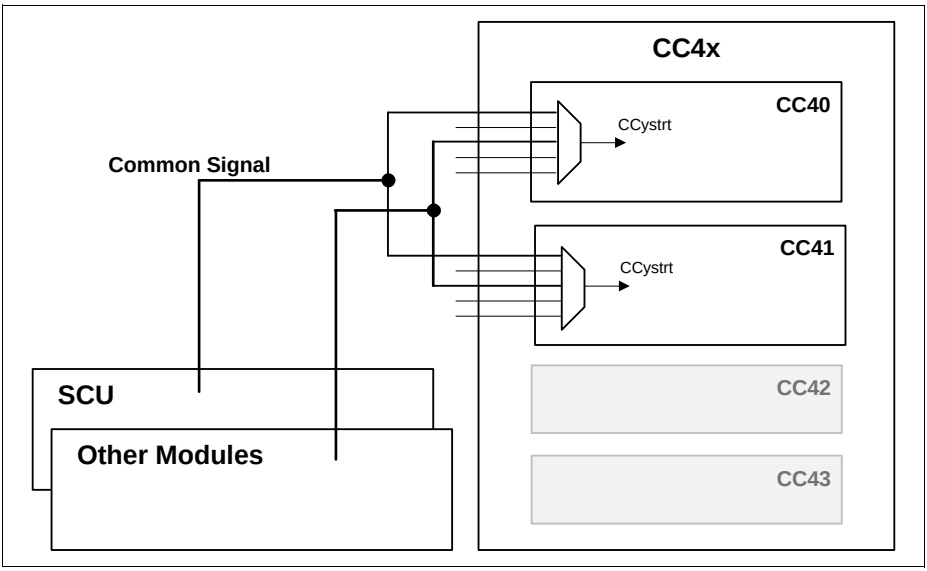


Figure 18-6 Starting multiple timers synchronously

18.2.5 Counting Modes

Each CC4y timer slice can be programmed into three different counting schemes:

- Edge aligned (default)
- Center aligned
- Single shot (can be edge or center aligned)

These three counting schemes can be used as stand alone without the need of selecting any inputs as external event sources. Nevertheless it is also possible to control the

counting operation via external events like, timer gating, counting trigger, external stop, external start, etc.

For all the counting modes, it is possible to update on the fly the values for the timer period and compare channel. This enables a cycle by cycle update of the PWM frequency and duty cycle.

The compare channel of each CC4y Timer Slice, has an associated Status Bit (**GCST**.CC4yST), that indicates the active or passive state of the channel, **Figure 18-7**. The set and clear of the status bit and the respective PWM signal generation is dictated by the timer period, compare value and the current counting mode. See the different counting mode descriptions, **Section 18.2.5.3** to **Section 18.2.5.5** to understand how this bit is set and cleared.

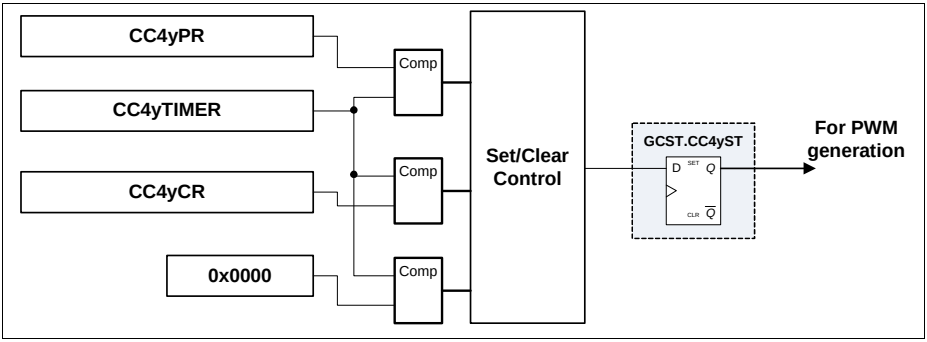


Figure 18-7 CC4y Status Bit

18.2.5.1 Calculating the PWM Period and Duty Cycle

The period of the timer is determined by the value in the period register, **CC4yPR** and by the timer mode.

The base for the PWM signal frequency and duty cycle, is always related to the clock frequency of the timer itself and not to the frequency of the module clock (due to the fact that the timer clock can be a scaled version of the module clock).

In Edge Aligned Mode, the timer period is:

$$T_{per} = \text{<Period-Value>} + 1; \text{ in } f_{tclk} \quad (18.1)$$

In Center Aligned Mode, the timer period is:

$$T_{per} = (\text{<Period-Value>} + 1) \times 2; \text{ in } f_{tclk} \quad (18.2)$$

For each of these counting schemes, the duty cycle of generated PWM signal is dictated by the value programmed into the **CC4yCR** register.

In Edge Aligned and Center Aligned Mode, the PWM duty cycle is:

DC= 1 - <Compare-Value>/(<Period-Value> + 1) (18.3)

Both **CC4yPR** and **CC4yCR** can be updated on the fly via software, enabling a glitch free transition between different period and duty cycle values for the generated PWM signal, **Section 18.2.5.2**

18.2.5.2 Updating the Period and Duty Cycle

Each CCU4 timer slice provides an associated shadow register for the period and compare values. This facilitates a concurrent update by software for these two parameters, with the objective of modifying during run time the PWM signal period and duty cycle.

In addition to the shadow registers for the period and compare values, one also has available shadow registers for the floating prescaler and dither functions, **CC4yFPCS** and **CC4yDITS** respectively (please address **Section 18.2.11** and **Section 18.2.10** for a complete description of these functions).

The structure of the shadow registers can be seen in **Figure 18-8**.

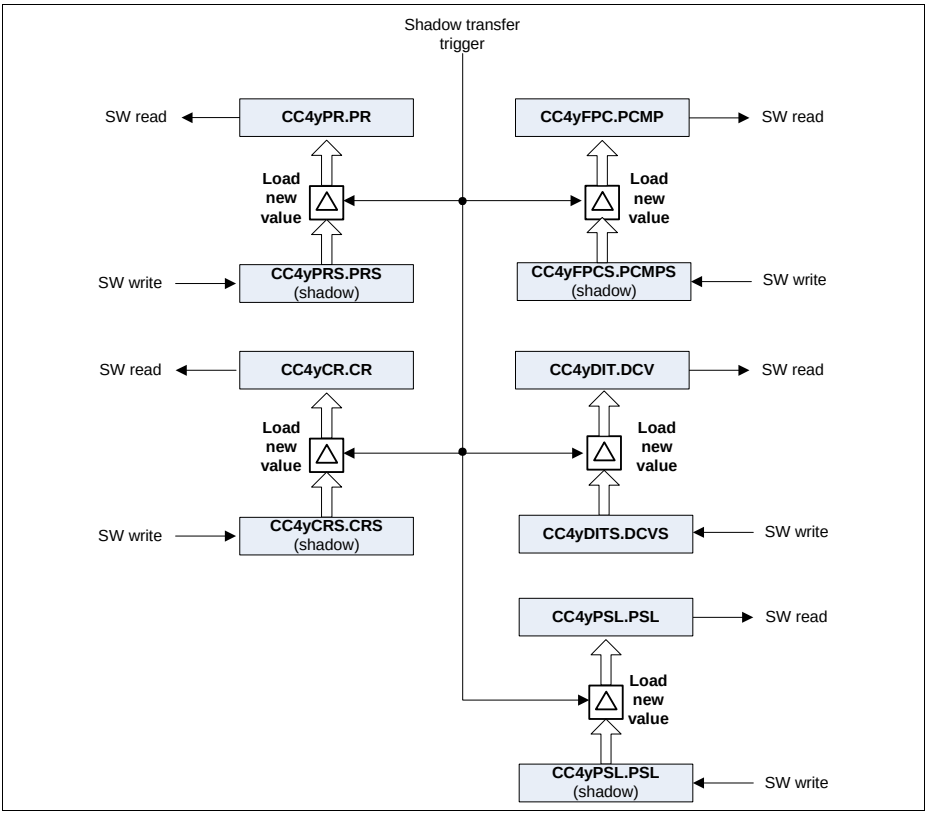


Figure 18-8 Shadow registers overview

The update of these registers can only be done by writing a new value into the associated shadow register and wait for a shadow transfer to occur.

Each group of shadow registers have an individual shadow transfer enable bit, **Figure 18-9**. The software must set this enable bit to 1_B, whenever an update of the values is needed. These bits are automatically cleared by the hardware, whenever an update of the values if finished. Therefore every time that an update of the registers is needed the software must set again the specific bit(s).

Nevertheless it is also possible to clear the enable bit via software. This can be used in the case that an update of the values needs to be cancelled (after the enable bit has already been set).

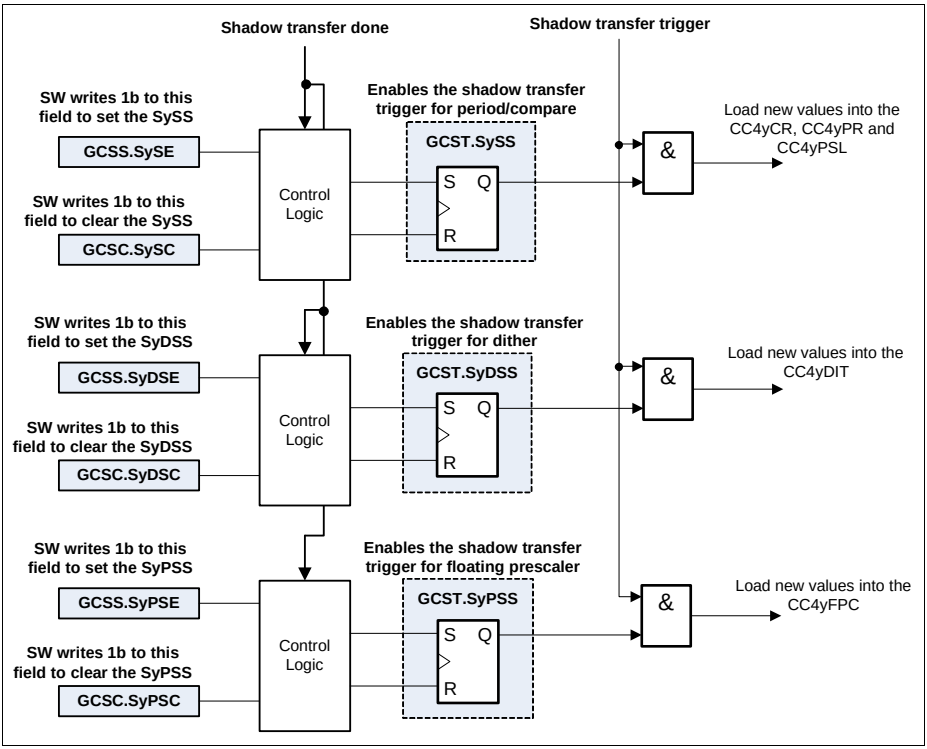


Figure 18-9 Shadow transfer enable logic

The shadow transfer operation is going to be done in the immediately next occurrence of a shadow transfer trigger, after the shadow transfer enable is set (**GCST**.SySS, **GCST**.SyDSS, **GCST**.SyPSS set to 1_B).

The occurrence of the shadow transfer trigger is imposed by the timer counting scheme (edge aligned or center aligned). Therefore the slots when the values are updated can be:

- in the next clock cycle after a Period Match while counting up
- in the next clock cycle after an One Match while counting down
- immediately, if the timer is stopped and the shadow transfer enable bit(s) is set

Figure 18-10 shows an example of the shadow transfer control when the timer slice has been configured into center aligned mode. For a complete description of all the timer slice counting modes, please address **Section 18.2.5.3**, **Section 18.2.5.4** and **Section 18.2.5.5**.

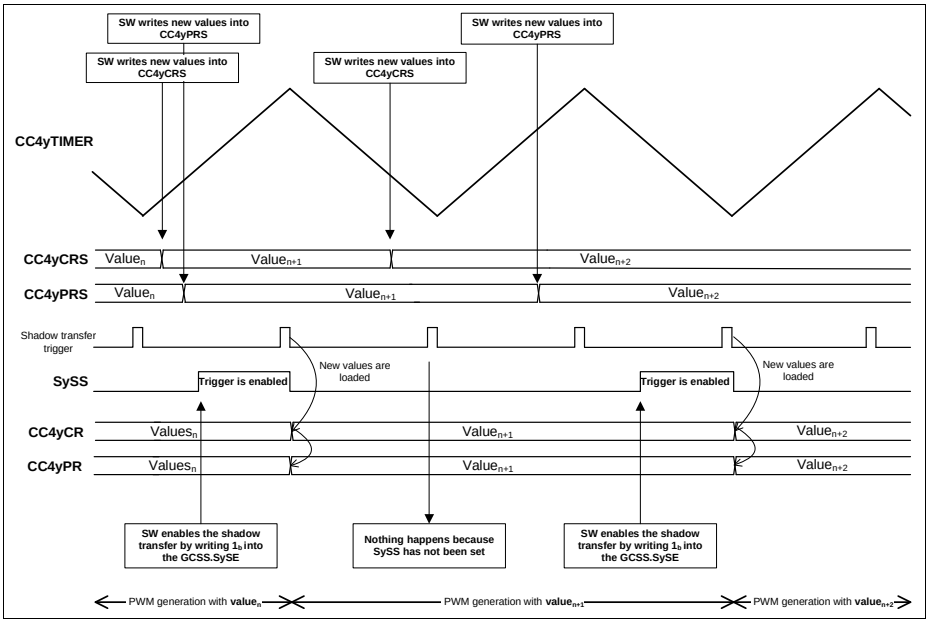


Figure 18-10 Shadow transfer timing example - center aligned mode

In some application cases it may be necessary to request shadow transfers not by software but by hardware. To perform this action each CCU4 contains a dedicated input that can be used to request a shadow transfer by hardware, the CCU4x.MCSS.

This input, when enabled, is used to set the shadow transfer enable bitfields (**GCST**.SySS, **GCST**.SyDSS and **GCST**.SyPSS) of the specific slice. It is possible to select which slice is using this input to perform the synchronization via the **GCTRL**.MSEy bit field. It is also possible to enable the usage of this signal for the three different shadow transfer signals: compare and period values, dither compare value and prescaler compare value. This can be configured on the **GCTRL**.MSDE field.

The structure for using the CCU4x.MCSS input signal can be seen in **Figure 18-9**. The usage of this signal is just an add on to the shadow transfer control and therefore all the previous described functions are still available.

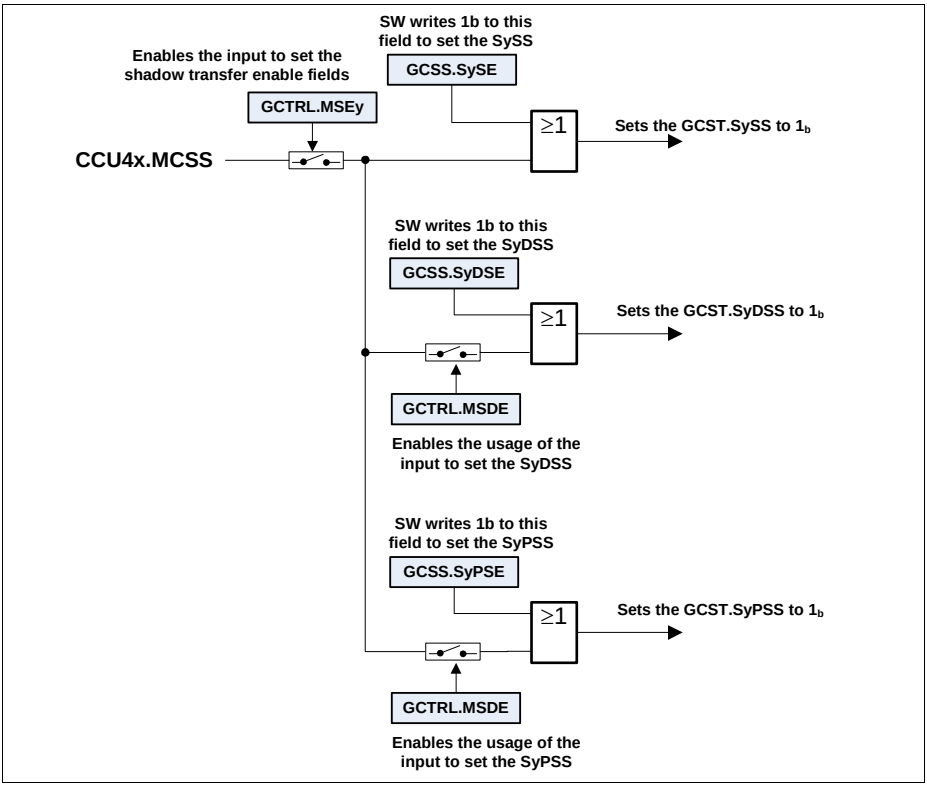


Figure 18-11 Usage of the CCU4x.MCSS input

18.2.5.3 Edge Aligned Mode

Edge aligned mode is the default counting scheme. In this mode, the timer is incremented until it matches the value programmed in the period register, **CC4yPR**. When period match is detected the timer is cleared to 0000_H and continues to be incremented.

In this mode, the value of the period register and compare register are updated with the value written by software into the correspondent shadow register, every time that an overflow occurs (period match), see **Figure 18-12**.

In edge aligned mode, the status bit of the comparison (CC4yST) is set one clock cycle after the timer hits the value programmed into the compare register. The clear of the status bit is done one clock cycle after the timer reaches 0000_H.

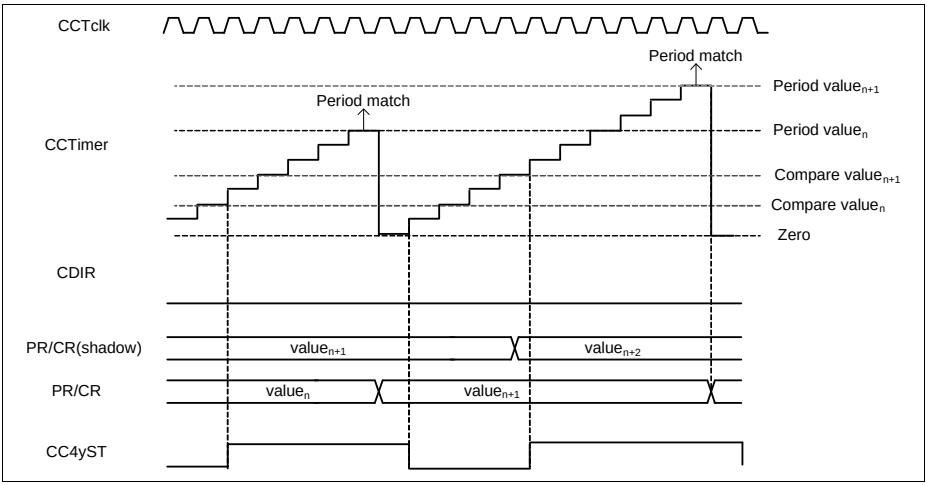


Figure 18-12 Edge aligned mode, CC4yTC.TCM = 0_B

18.2.5.4 Center Aligned Mode

In center aligned mode, the timer is counting up or down with respect to the following rules:

- The counter counts up while **CC4yTCST**.CDIR = 0_B and it counts down while **CC4yTCST**.CDIR = 1_B.
- Within the next clock cycle, the count direction is set to counting up (**CC4yTCST**.CDIR = 0_B) when the counter reaches 0001_H while counting down.
- Within the next clock cycle, the count direction is set to counting down (**CC4yTCST**.CDIR = 1_B), when the period match is detected while counting up.

The status bit (CC4yST) is always 1_B when the counter value is equal or greater than the compare value and 0_B otherwise.

While in edge aligned mode, the shadow transfer for compare and period registers is executed once per period. It is executed twice in center aligned mode as follows

- Within the next clock cycle after the counter reaches the period value, while counting up (**CC4yTCST**.CDIR = 0_B).
- Within the next clock cycle after the counter reaches 0001_H, while counting down (**CC4yTCST**.CDIR = 1_B).

*Note: Bit **CC4yTCST**.CDIR changes within the next timer clock after the one-match or the period-match, which means that the timer continues counting in the previous direction for one more cycle before changing the direction.*

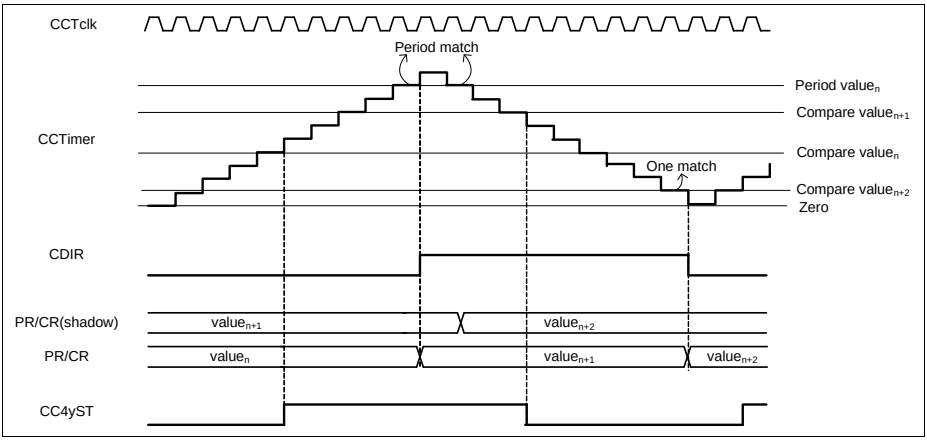


Figure 18-13 Center aligned mode, CC4yTC.TCM = 1_B

18.2.5.5 Single Shot Mode

In single shot mode, the timer is stopped after the current timer period is finished. This mode can be used with center or edge aligned scheme.

In edge aligned mode, **Figure 18-14**, the timer is stopped when it is cleared to 0000_H after having reached the period value. In center aligned mode, **Figure 18-15**, the period is finished when the timer has counted down to 0000_H.

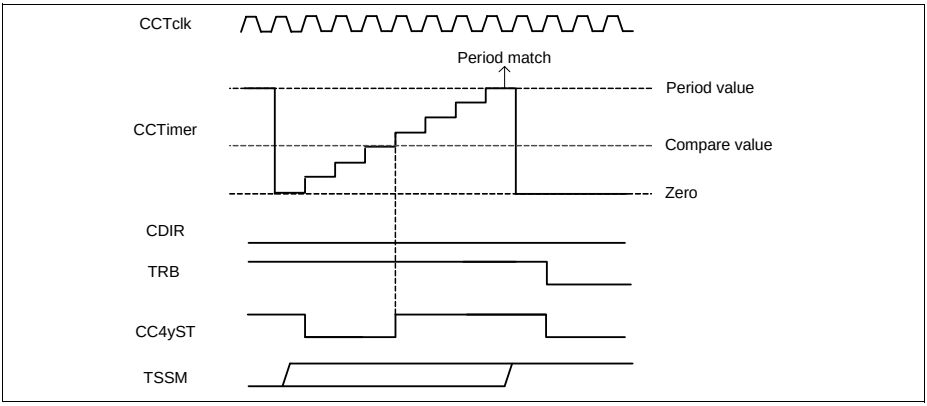


Figure 18-14 Single shot edge aligned - CC4yTC.TSSM = 1_B, CC4yTC.TCM = 0_B

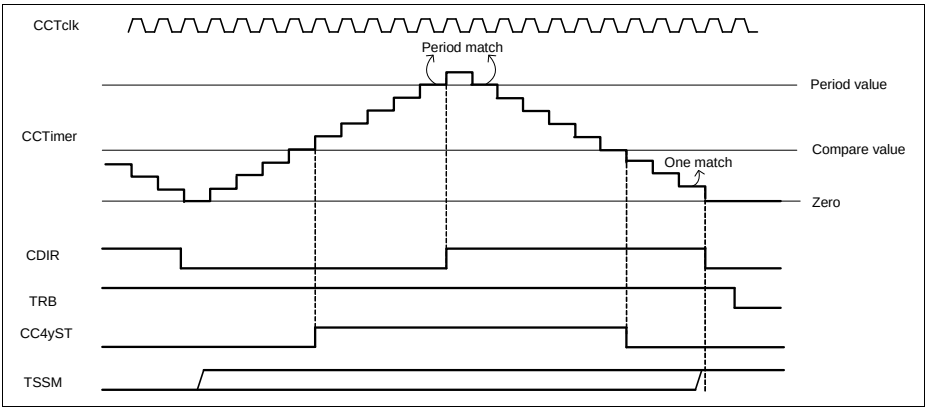


Figure 18-15 Single shot center aligned - CC4yTC.TSSM = 1_B, CC4yTC.TCM = 1_B

18.2.6 Active/Passive Rules

The general rules that set or clear the associated timer slice status bit (CC4yST), can be generalized independently of the timer counting mode.

The following events set the Status bit (CC4yST) to Active:

- in the next f_{tclk} cycle after a compare match while counting up
- in the next f_{tclk} cycle after a zero match while counting down

The following events set the Status bit (CC4yST) to Inactive:

- in the next f_{tclk} cycle after a zero match (and not compare match) while counting up
- in the next f_{tclk} cycle after a compare match while counting down

If external events are being used to control the timer operation, these rules are still applicable.

The status bit state can only be ‘override’ via software or by the external status bit override function, **Section 18.2.7.9**.

The software can at any time write a 1_B into the **GCSS**.SySTS bitfield, which will set the status bit **GCST**.CC4yST of the specific timer slice. Writing a 1_B into the **GCSC**.SySTC bitfield will clear the specific status bit.

18.2.7 External Events Control

Each CCU4 timer slice has the possibility of using up to three different input events, see **Section 18.2.2**. These three events can then be mapped to Timer Slice functions (the full set of available functions is described at **Section 18.2.3**)

These events can be mapped to any of the CCU4x.INy[P...A] inputs and there isn’t any imposition that an event cannot be used to perform several functions, or that an input

cannot be mapped to several events (e.g. input X triggers event 0 with rising edge and triggers event 1 with the falling edge).

18.2.7.1 External Start/Stop

To select an external start function, one should map one of the events (output of the input selector) to a specific input signal, by setting the required value in the **CC4yINS**.EVxIS field and indicating the active edge of the signal on the **CC4yINS**.EVxEM field.

This event should be then mapped to the start or stop functionality by setting the **CC4yCMC**.STRTS (for the start) or the **CC4yTC**.ENDM (for the stop) with the proper value.

Notice that both start and stop functions are edge and not level active and therefore the active/passive configuration is set only by the **CC4yINS**.EVxEM.

The external stop by default just clears the run bit (**CC4yTCST**.TRB), while the start functions does the opposite. Nevertheless one can select an extended subset of functions for the external start and stop. This subset is controlled by the registers **CC4yTC**.ENDM (for the stop) and **CC4yTC**.STRM (for the start).

For the start subset (**CC4yTC**.STRM):

- sets the run bit/starts the timer (resume operation)
- clears the timer, sets the run bit/starts the timer (flush and start)

For the stop subset (**CC4yTC**.ENDM):

- clears the run/stops the timer (stop)
- clears the timer (flush)
- clears the timer, clears the run bit/stops the timer (flush and stop)

If in conjunction with an external start/stop (configured also/only as flush) and external up/down signal is used, during the flush operation the timer is going to be set to 0000_H if the actual counting direction is up or set with the value of the period register if the counting direction is down.

Figure 18-16 to **Figure 18-19** shows the usage of two signals to perform the start/stop functions in all the previously mentioned subsets. External Signal(1) acts as an active HIGH start signal, while External Signal(2) is used as an active HIGH stop function.

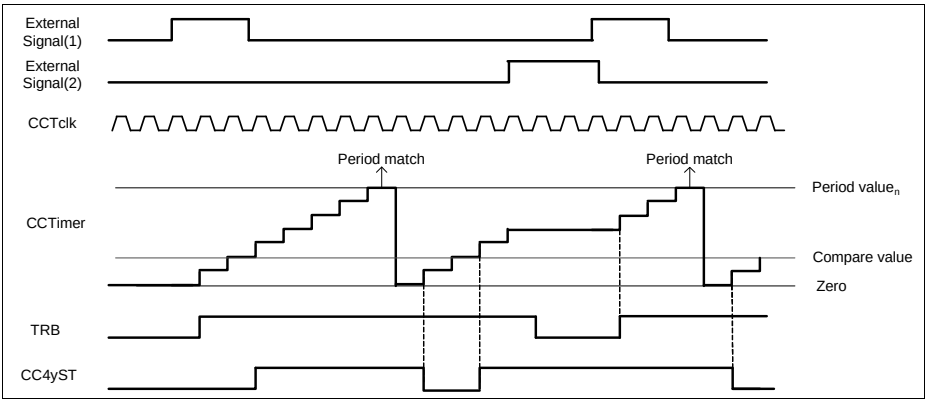


Figure 18-16 Start (as start)/ stop (as stop) - CC4yTC.STRM = 0_B, CC4yTC.ENDM = 00_B

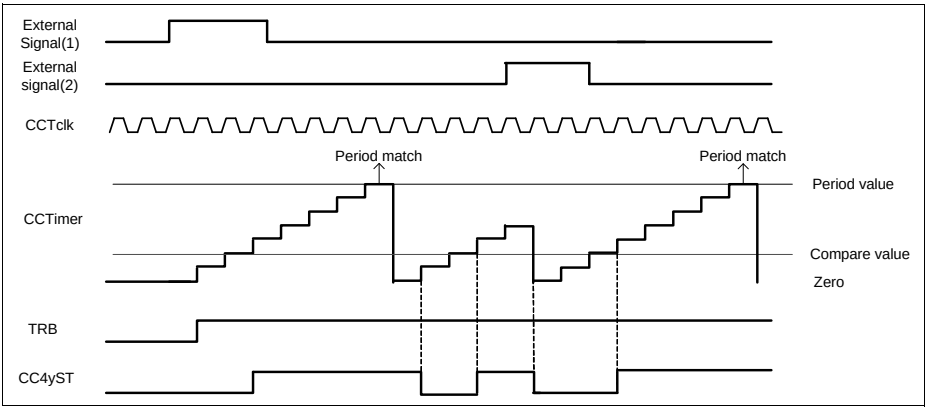


Figure 18-17 Start (as start)/ stop (as flush) - CC4yTC.STRM = 0_B, CC4yTC.ENDM = 01_B

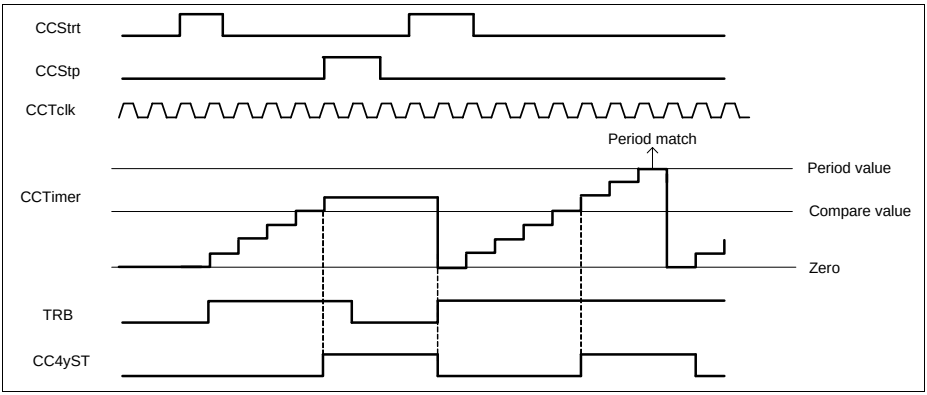


Figure 18-18 Start (as flush and start)/ stop (as stop) - CC4yTC.STRM = 1_B, CC4yTC.ENDM = 00_B

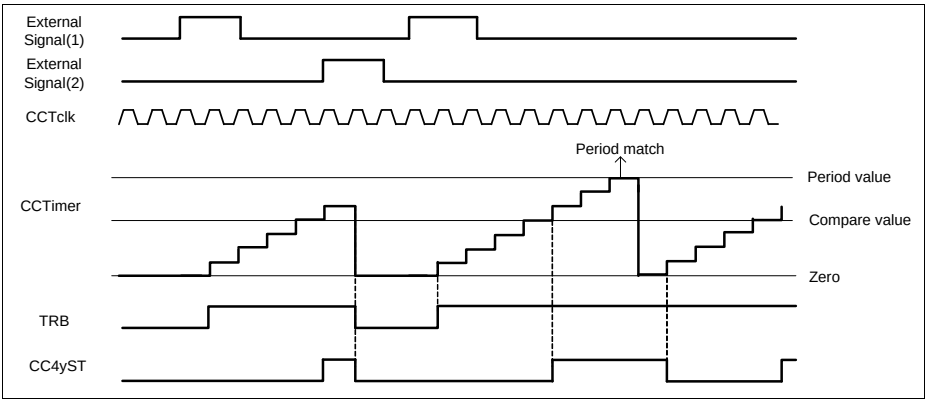


Figure 18-19 Start (as start)/ stop (as flush and stop) - CC4yTC.STRM = 0_B, CC4yTC.ENDM = 10_B

18.2.7.2 External Counting Direction

There is the possibility of selecting an input signal to act as increment/decrement control.

To select an external up/down control, one should map one of the events (output of the input selector) to a specific input signal, by setting the required value in the **CC4yINS**.EVxIS field and indicating the active level of the signal on the **CC4yINS**.EVxLM. This event should be then mapped to the up/down functionality by setting **CC4yCMC**.UDS with the proper value.

Notice that the up/down function is level active and therefore the active/passive configuration is set only by the **CC4yINS**.EVxLM.

The status bit of the slice (CC4yST) is always set when the timer value is equal or greater than the value stored in the compare register, see **Section 18.2.6**.

The update of the period and compare register values is done when:

- with the next clock after a period match, while counting up (**CC4yTCST**.CDIR = 0_B)
- with the next clock after a one match, while counting down (**CC4yTCST**.CDIR = 1_B)

The value of the **CC4yTCST**.CDIR register is updated accordingly with the changes on the decoded event. The Up/Down direction is always understood as **CC4yTCST**.CDIR = 1_B when counting down and **CC4yTCST**.CDIR = 0_B when counting up. Using an external signal to perform the up/down counting function and configuring the event as active HIGH means that the timer is counting up when the signal is HIGH and counting down when LOW.

Figure 18-20 shows an external signal being used to control the counting direction of the time. This signal was selected as active HIGH, which means that the timer is counting down while the signal is HIGH and counting up when the signal is LOW.

*Note: For a signal that should impose an increment when LOW and a decrement when HIGH, the user needs to set the **CC4yINS**.EVxLM = 0_B. When the operation is switched, then the user should set **CC4yINS**.EVxLM = 1_B.*

Note: Using an external counting direction control, sets the slice in edge aligned mode.

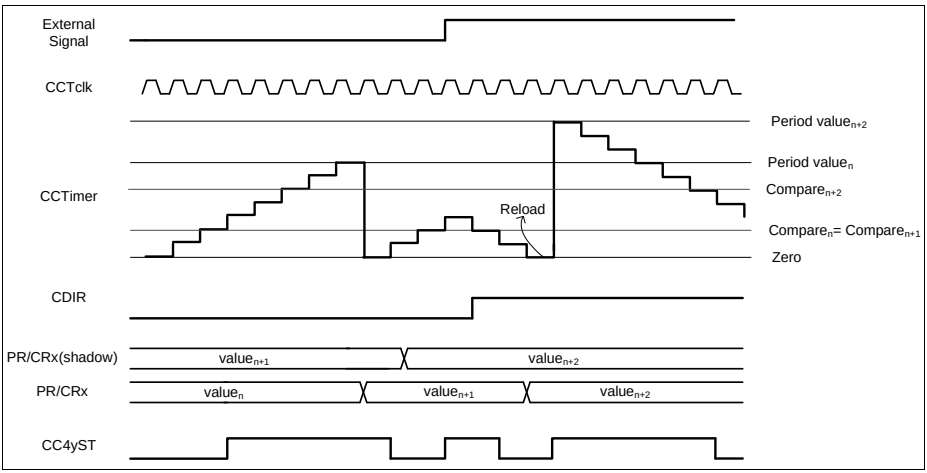


Figure 18-20 External counting direction

18.2.7.3 External Gating Signal

For pulse measurement, the user has the possibility of selecting an input signal that operates as counting gating.

To select an external gating control, one should map one of the events (output of the input selector) to a specific input signal, by setting the required value in the **CC4yINS**.EVxIS register and indicating the active level of the signal on the **CC4yINS**.EVxLM register. This event should be then mapped to the gating functionality by setting the **CC4yCMC**.GATES with the proper value.

Notice that the gating function is level active and therefore the active/passive configuration is set only by the **CC4yINS**.EVxLM.

The status bit during an external gating signal continues to be asserted when the compare value is reached and deasserted when the counter reaches 0000_H. One should note that the counter continues to use the period register to identify the wrap around condition. **Figure 18-21** shows the usage of an external signal for gating the slice counter. The signal was set as active LOW, which means the counter gating functionality is active when the external value is zero.

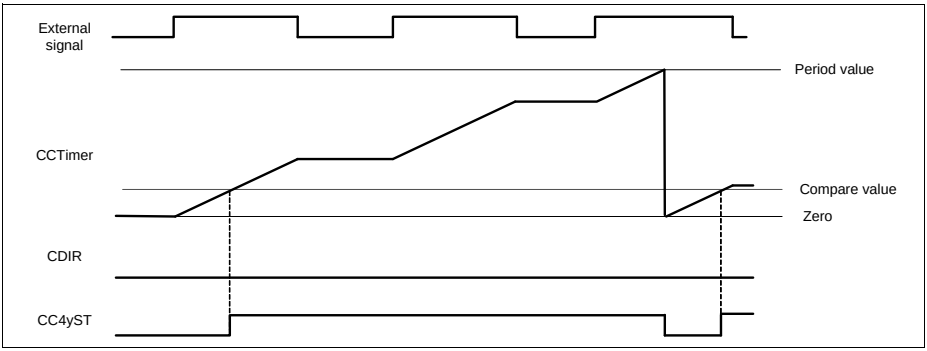


Figure 18-21 External gating

For any type of usage of the external gating function, the specific run bit of the Timer Slice, **CC4yTCST**.TRB, needs to be set. This can be done via an additional external signal or directly via software.

18.2.7.4 External Count Signal

There is also the possibility of selecting an external signal to act as the counting event.

To select an external counting, one should map one of the events (output of the input selector) to a specific input signal, by setting the required value in the **CC4yINS**.EVxIS register and indicating the active edge of the signal on the **CC4yINS**.EVxEM register.

This event should be then mapped to the counting functionality by setting the **CC4yCMC**.CNTS with the proper value.

Notice that the counting function is edge active and therefore the active/passive configuration is set only by the **CC4yINS**.EVxEM.

One can select just a the rising, falling or both edges to perform a count. On **Figure 18-22**, the external signal was selected as a counter event for both falling and rising edges. Wrap around condition is still applied with a comparison with the period register.

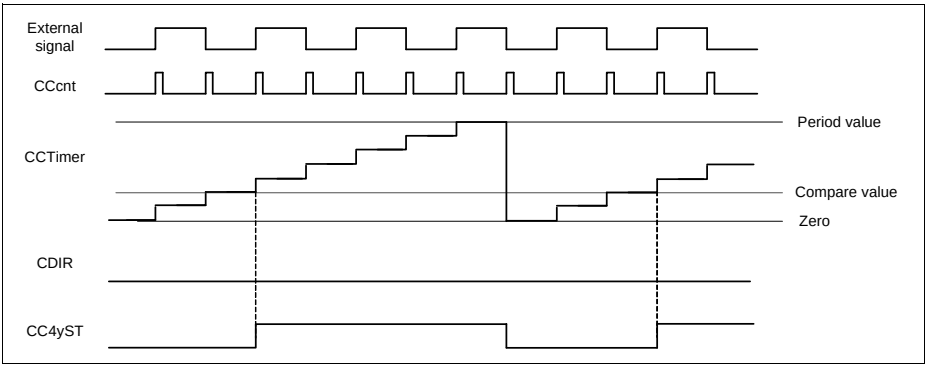


Figure 18-22 External count

For any type of usage of the external gating function, the specific run bit of the Timer Slice, **CC4yTCST**.TRB, needs to be set. This can be done via an additional external signal or directly via software.

18.2.7.5 External Load

Each slice of the CCU4 also has a functionality that enables the user to select an external signal as trigger for reloading the value of the timer with the current value of the compare register (if **CC4yTCST**.CDIR = 0_B) or with the value of the period register (if **CC4yTCST**.CDIR = 1_B).

To select an external load signal, one should map one of the events (output of the input selector) to a specific input signal, by setting the required value in the **CC4yINS**.EVxIS register and indicating the active edge of the signal on the **CC4yINS**.EVxEM register. This event should be then mapped to the load functionality by setting the **CC4yCMC**.LDS with the proper value.

Notice that the load function is edge active and therefore the active/passive configuration is set only by the **CC4yINS**.EVxEM.

On figure **Figure 18-23**, the external signal (1) was used to act as a load trigger, active on the rising edge. Every time that a rising edge on external signal (1) is detected, the

timer value is loaded with the value present on the compare register. If an external signal is being used to control the counting direction, up or down, the timer value can be loaded also with the value set in the period register. The External signal (2) represents the counting direction control (active HIGH). If at the moment that a load trigger is detected, the signal controlling the counting direction is imposing a decrement, then the value set in the timer is the period value.

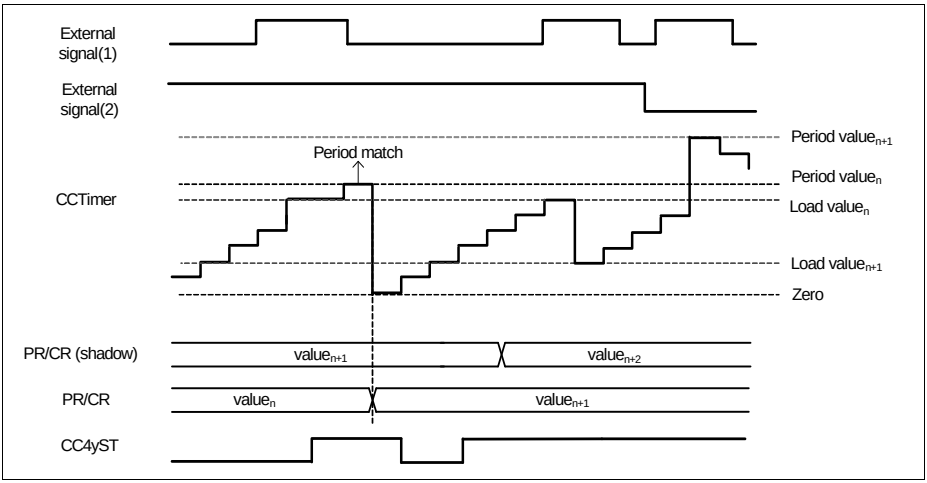


Figure 18-23 External load

18.2.7.6 External Capture

When selecting an external signal to be used as a capture trigger (if **CC4yCMC**.CAP0S or **CC4yCMC**.CAP1S are different from 0_H), the user is automatically setting the specific slice into capture mode.

In capture mode the user can have up to four capture registers, see **Figure 18-26**: capture register 0 (**CC4yC0V**), capture register 1 (**CC4yC1V**), capture register 2 (**CC4yC2V**) and capture register 3 (**CC4yC3V**).

These registers are shared between compare and capture modes which imposes:

- if **CC4yC0V** and **CC4yC1V** are used for capturing, the compare registers **CC4yCR** and **CC4yCRS** are not available (no compare channel)
- if **CC4yC2V** and **CC4yC3V** are used for capturing, the period registers **CC4yPR** and **CC4yPRS** are not available (no period control)

To select an external capture signal, one should map one of the events (output of the input selector) to a specific input signal, by setting the required value in the **CC4yINS**.EVxIS register and indicating the active edge of the signal on the

CC4yINS.EVxEM register. This event should be then mapped to the capture functionality by setting the **CC4yCMC**.CAP0S/**CC4yCMC**.CAP1S with the proper value.

Notice that the capture function is edge active and therefore the active/passive configuration is set only by the **CC4yINS**.EVxEM.

The user has the possibility of selecting the following capture schemes:

- Different capture events for **CC4yC0V**/**CC4yC1V** and **CC4yC2V**/**CC4yC3V**
- The same capture event for **CC4yC0V**/**CC4yC1V** and **CC4yC2V**/**CC4yC3V** with the same capture edge. For this capture scheme, only the CCcapt1 functionality needs to be programmed. To enable this scheme, the field **CC4yTC**.SCE needs to be set to 1.

Different Capture Events (SCE = 0_B)

Every time that a capture trigger 1 occurs, CCcapt1, the actual value of the timer is captured into the capture register 3 and the previous value stored in this register is transferred into capture register 2.

Every time that a capture trigger 0 occurs, CCcapt0, the actual value of the timer is captured into the capture register 1 and the previous value stored in this register is transferred into capture register 0.

Every time that a capture procedure into one of the registers occurs, the respective full flag is set. This flag is cleared automatically by HW when the SW reads back the value of the capture register.

The capture of a new value into a specific capture registers is dictated by the status of the full flag as follows:

$$CC4yC1V_{capt} = \text{NOT}(CC4yC1V_{full\_flag} \text{ AND } CC4yC0V_{full\_flag}) \quad (18.4)$$

$$CC4yC0V_{capt} = CC4yC1V_{full\_flag} \text{ AND NOT}(CC4yC0V_{full\_flag}) \quad (18.5)$$

It is also possible to disable the effect of the full flags reset by setting the **CC4yTC**.CCS = 1_B. This enables a continuous capturing independent if the values captured have been read or not.

*Note: When using the period registers for capturing, **CC4yCMC**.CAP1S different from 00_B, the counter always uses its full 16 bit width as period value.*

On **Figure 18-24**, an external signal was selected as an event for capturing the timer value into the **CC4yC0V**/**CC4yC1V** registers. The status bit, CC4yST, during capture mode is asserted whenever a capture trigger is detected and deasserted when the counter matches 0000_H.

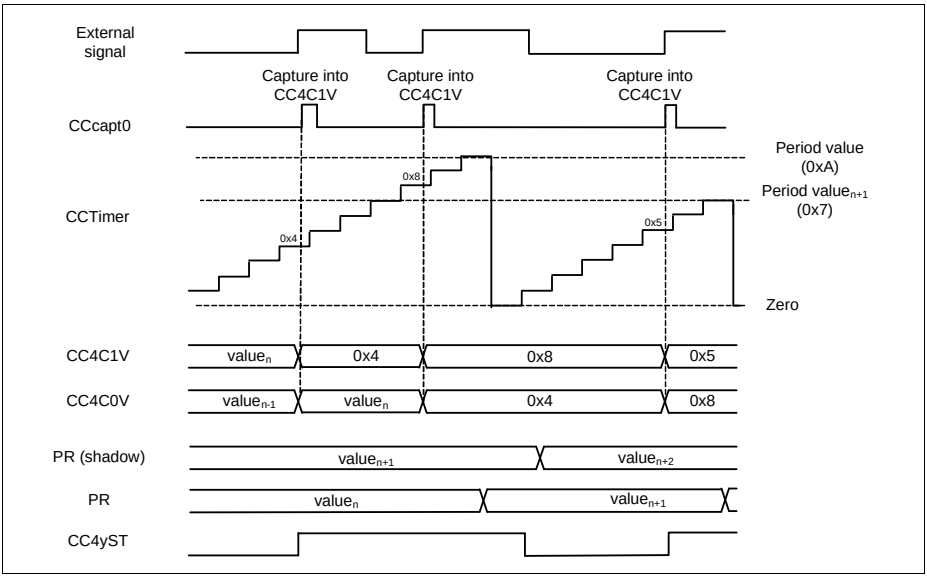


Figure 18-24 External capture - CC4yCMC.CAP0S != 00_B, CC4yCMC.CAP1S = 00_B

On **Figure 18-25**, two different signals were used as source for capturing the timer value into the **CC4yC0V**/**CC4yC1V** and **CC4yC2V**/**CC4yC3V** registers.

External signal(1) was selected as rising edge active capture source for **CC4yC0V**/**CC4yC1V**. External signal(2) was selected has the capture source for **CC4yC2V**/**CC4yC3V**, but as opposite to the external signal(1), the active edge was selected has falling.

See **Section 18.2.12.4**, for the complete capture mode usage description.

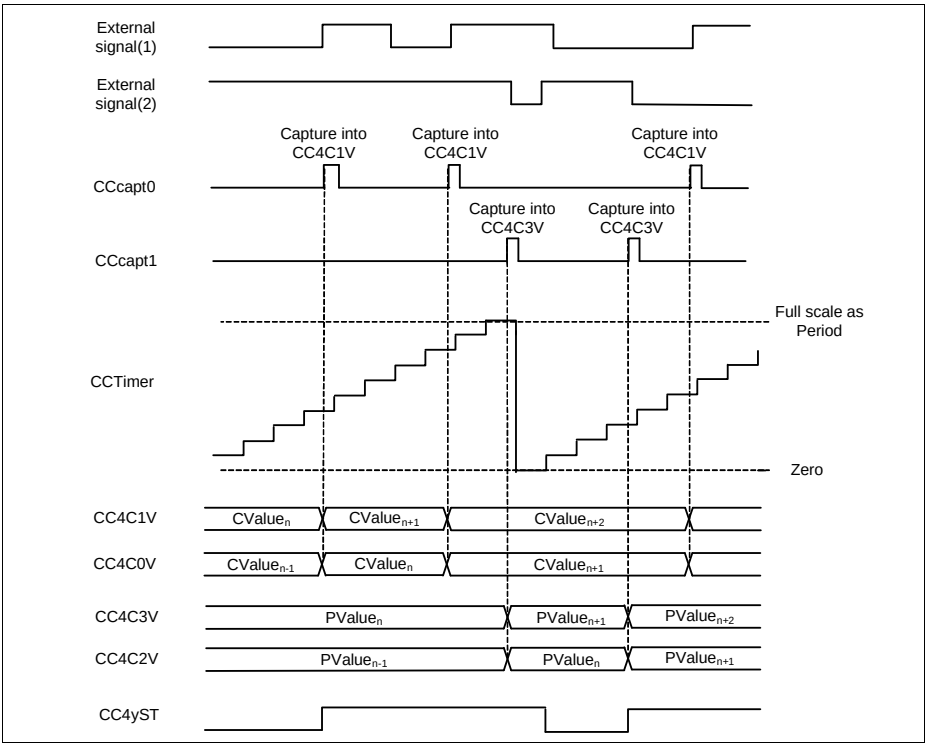


Figure 18-25 External capture - CC4yCMC.CAP0S != 00_B, CC4yCMC.CAP1S != 00_B

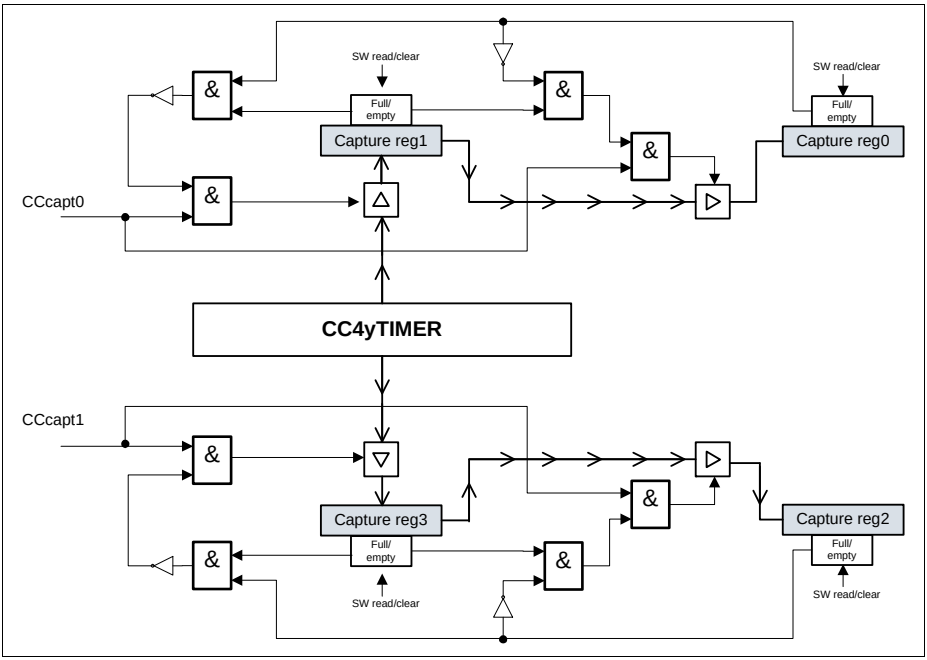


Figure 18-26 Slice capture logic

Same Capture Event (SCE = 1_B)

Setting the field **CC4yTC**.SCE = 1_B, enables the possibility of having 4 capture registers linked with the same capture event, **Figure 18-28**. The function that controls the capture is the CCcapt1.

The capture logic follows the same structure shown in **Figure 18-26** but extended to a four register chain, see **Figure 18-27**. The same full flag lock rules are applied to the four register chain (it also can be disabled by setting the **CC4yTC**.CCS = 1_B):

$$CC4yC3V_{capt} = \text{NOT}(CC4yC3V_{full\_flag} \text{ AND } CC4yC2V_{full\_flag} \text{ AND } CC4yC2V_{full\_flag} \text{ AND } CC4yC1V_{full\_flag}) \quad (18.6)$$

$$CC4yC2V_{capt} = CC4yC3V_{full\_flag} \text{ AND NOT}(CC4yC2V_{full\_flag} \text{ AND } CC4yC1V_{full\_flag} \text{ AND } CC4yC0V_{full\_flag}) \quad (18.7)$$

$$CC4yC1V_{capt} = CC4yC2V_{full\_flag} \text{ AND NOT}(CC4yC1V_{full\_flag} \text{ AND } CC4yC0V_{full\_flag}) \quad (18.8)$$

$$CC4yC0V_{capt} = CC4yC1V_{full\_flag} \text{ AND NOT}(CC4yC0V_{full\_flag}) \quad (18.9)$$

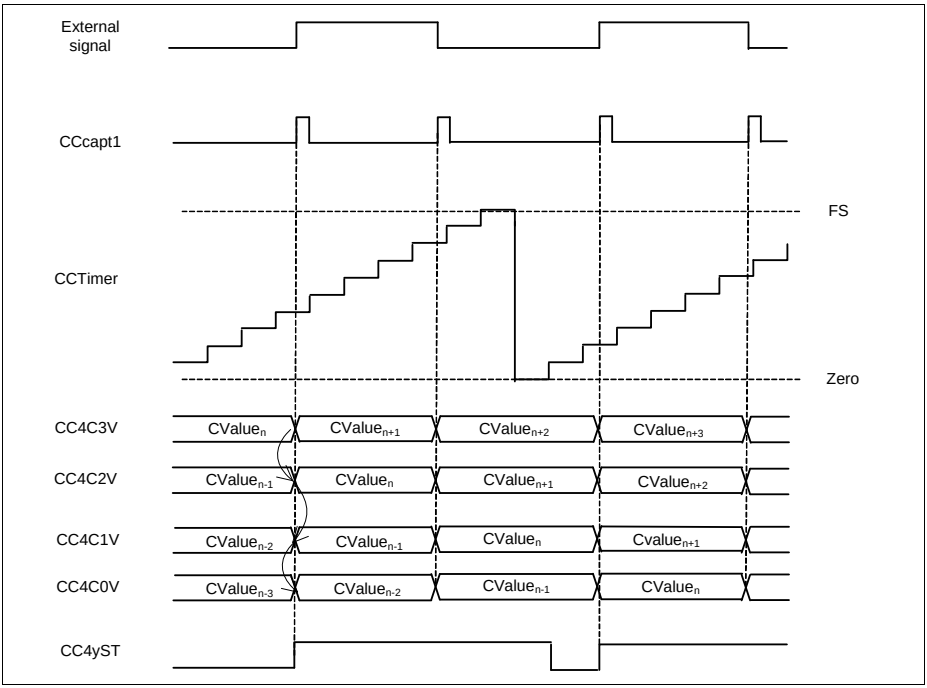


Figure 18-27 External Capture - CC4yTC.SCE = 1_B

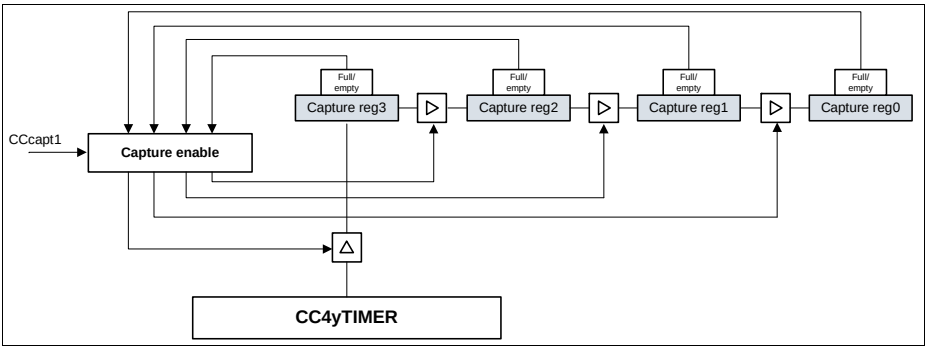


Figure 18-28 Slice Capture Logic - CC4yTC.SCE = 1_B

18.2.7.7 External Modulation

An external signal can be used to perform a modulation at the output of each timer slice.

To select an external modulation signal, one should map one of the input signals to one of the events, by setting the required value in the **CC4yINS**.EVxIS register and indicating the active level of the signal on the **CC4yINS**.EVxLM register. This event should be then mapped to the modulation functionality by setting the **CC4yCMC**.MOS = 01_B if event 0 is being used, **CC4yCMC**.MOS = 10_B if event 1 or **CC4yCMC**.MOS = 11_B if event 2.

Notice that the modulation function is level active and therefore the active/passive configuration is set only by the **CC4yINS**.EVxLM.

The modulation has two modes of operation:

- modulation event is used to clear the CC4yST bit - **CC4yTC**.EMT = 0_B
- modulation event is used to gate the outputs - **CC4yTC**.EMT = 1_B

On **Figure 18-29**, we have a external signal configured to act as modulation source that clears the CC4yST bit, **CC4yTC**.EMT = 0_B. It was programmed to be an active LOW event and therefore, when this signal is LOW the output value follows the normal ACTIVE/PASSIVE rules.

When the signal is HIGH (inactive state), then the CC4yST bit is cleared and the output is forced into the PASSIVE state. Notice that the values of the status bit, CC4yST and the specific output CCU4x.OUTy are not linked together. One can choose for the output to be active LOW or HIGH through the PSL bit.

The exit of the external modulation inactive state is synchronized with the PWM signal due to the fact that the CC4yST bit is cleared and cannot be set while the modulation signal is inactive.

The entering into inactive state also can be synchronized with the PWM signal, by setting **CC4yTC**.EMS = 1_B. With this all possible glitches at the output are avoided, see **Figure 18-30**.

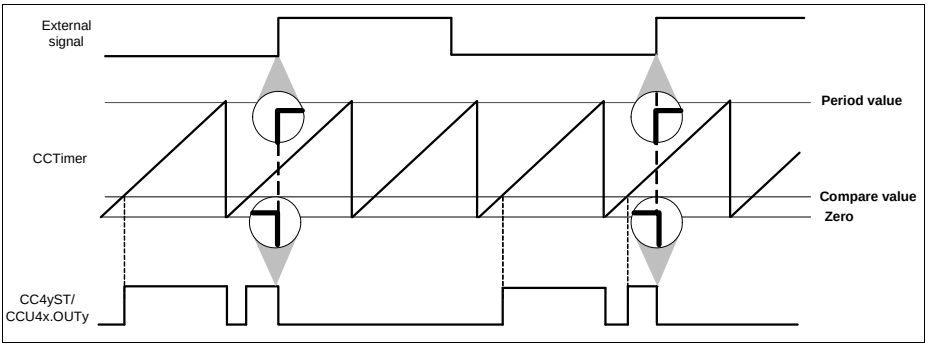


Figure 18-29 External modulation clearing the ST bit - CC4yTC.EMT = 0_B

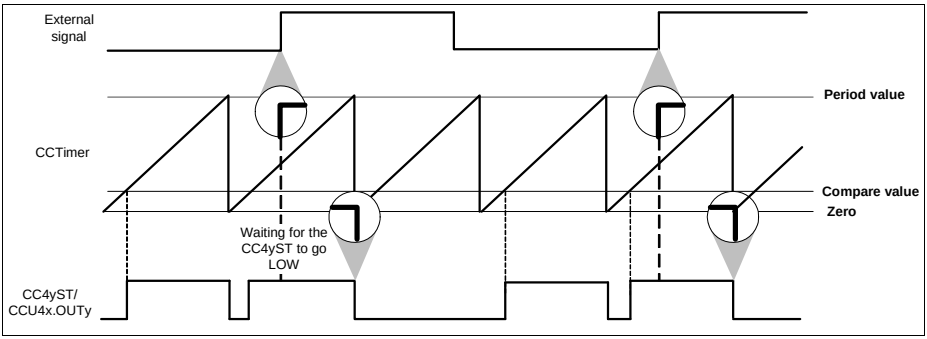


Figure 18-30 External modulation clearing the ST bit - CC4yTC.EMT = 0_B, CC4yTC.EMS = 1_B

On **Figure 18-31**, the external modulation event was used as gating signal of the outputs, **CC4yTC**.EMT = 1_B. The external signal was configured to be active HIGH, **CC4yINS**.EVxLM = 0_B, which means that when the external signal is HIGH the outputs are set to the PASSIVE state.In this mode, the gating event can also be synchronized with the PWM signal by setting the **CC4yTC**.EMS = 1_B.

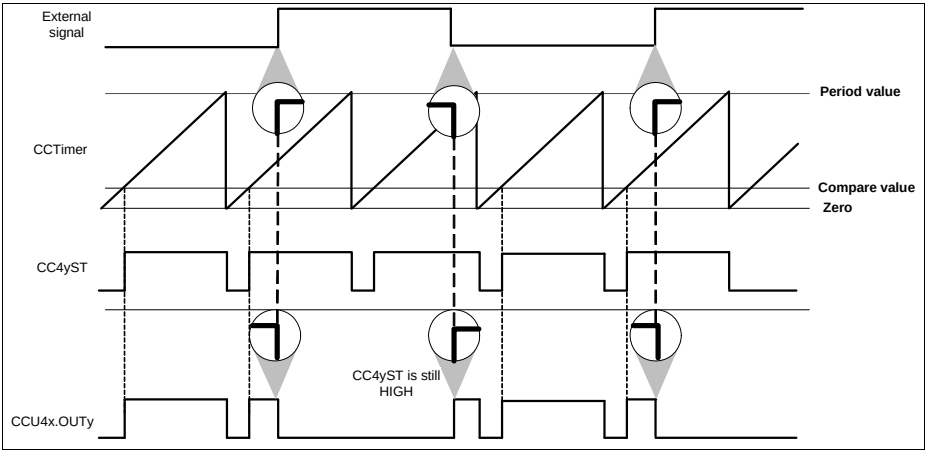


Figure 18-31 External modulation gating the output - CC4yTC.EMT = 1_B

18.2.7.8 TRAP Function

The TRAP functionality allows the PWM outputs to react on the state of an input pin. This functionality can be used to switch off the power devices if the TRAP input becomes active.

To select the TRAP functionality, one should map one of the input signals to event number 2, by setting the required value in the **CC4yINS**.EV2IS register and indicating the active level of the signal on the **CC4yINS**.EV2LM register. This event should be then mapped to the trap functionality by setting the **CC4yCMC**.TS = 1_B.

Notice that the trap function is level active and therefore the active/passive configuration is set only by the **CC4yINTS**.EV2LM.

There are two bitfields that can be monitored via software to crosscheck the TRAP function, **Figure 18-32**:

- The TRAP state bit, **CC4yINTS**.E2AS. This bitfield if the TRAP is currently active or not. This bitfield is therefore setting the specific Timer Slice output, into ACTIVE or PASSIVE state.
- The TRAP Flag, **CC4yINTS**.TRPF. This bitfield is used as a remainder in the case that the TRAP condition is cleared automatically via hardware. This field needs to be cleared by the software.

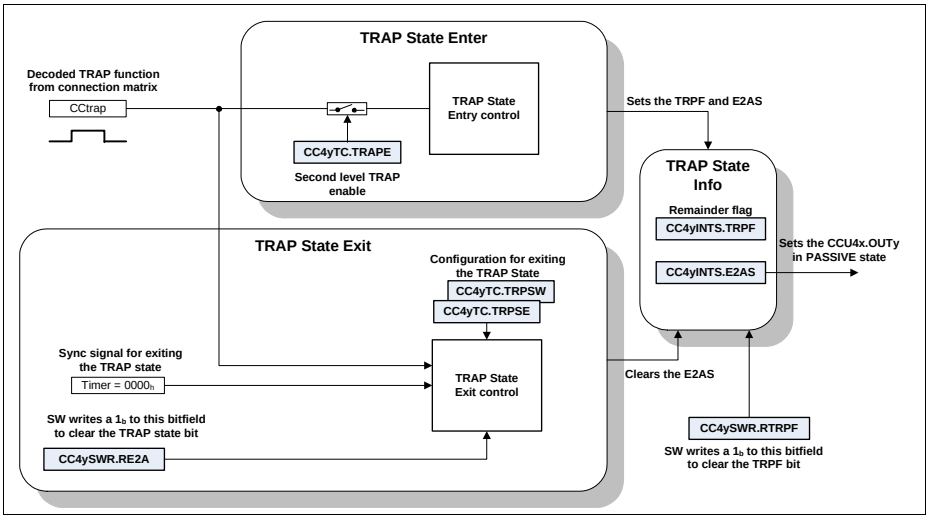


Figure 18-32 Trap control diagram

When a TRAP condition is detected at the selected input pin, both the Trap Flag and the Trap State bit are set to 1_B. The Trap State is entered immediately, by setting the CCU4xOUTy into the programmed PASSIVE state, **Figure 18-33**.

Exiting the Trap State can be done in two ways (**CC4yTC**.TRPSW register):

- automatically via HW, when the TRAP signal becomes inactive - **CC4yTC**.TRPSW = 0_B
- by SW only, by clearing the **CC4yINTS**.E2AS.The clearing is only possible if the input TRAP signal is in inactive state - **CC4yTC**.TRPSW = 1_B

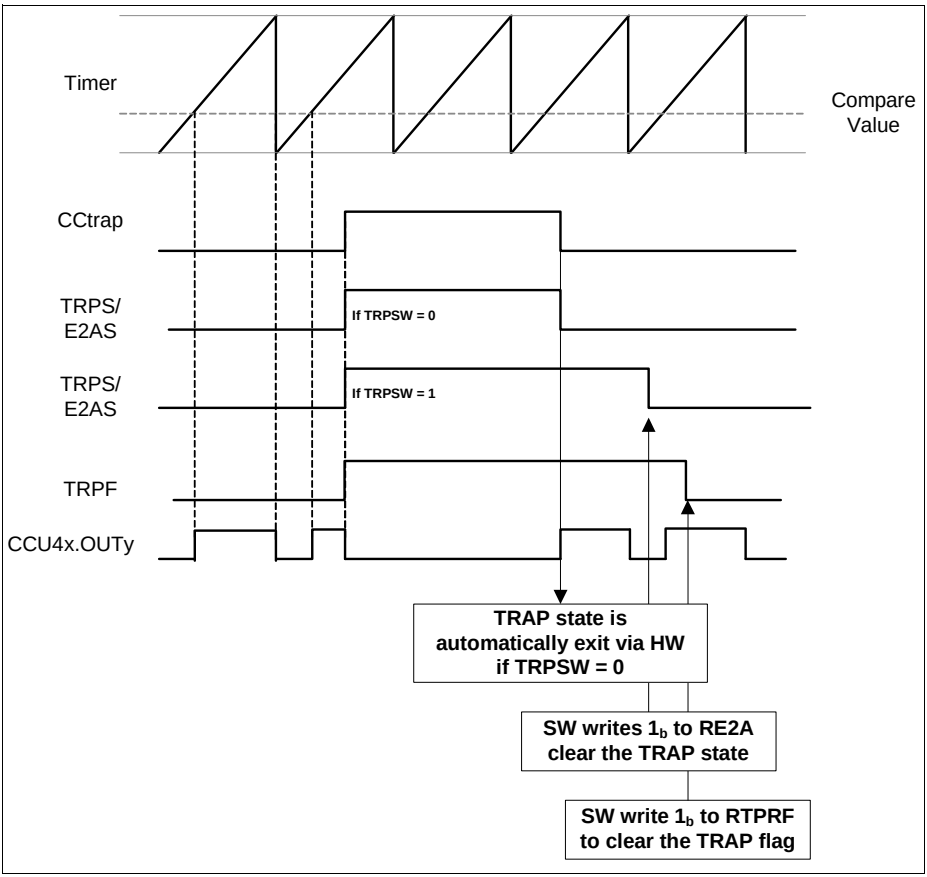


Figure 18-33 Trap timing diagram, CC4yPSL.PSL = 0_B (output passive level is 0_B)

It is also possible to synchronize the exiting of the TRAP state with the PWM signal, **Figure 18-34**. This function is enabled when the bitfield **CC4yTC**.TRPSE = 1_B.

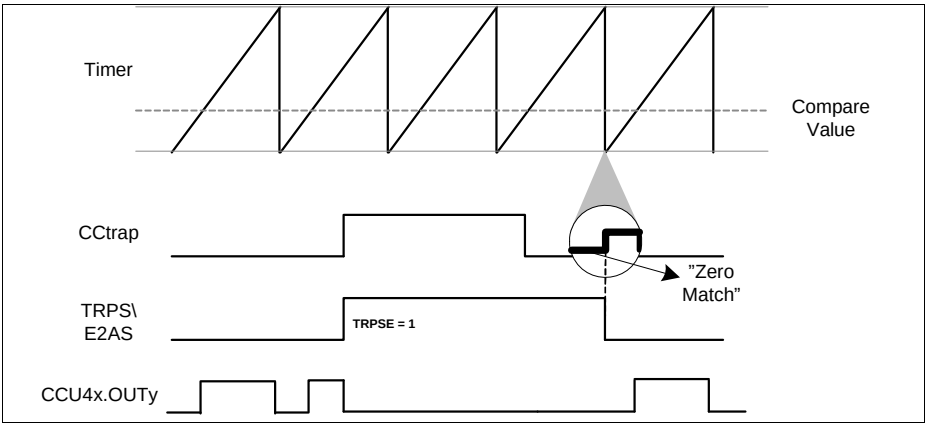


Figure 18-34 Trap synchronization with the PWM signal, CC4yTC.TRPSE = 1_B

18.2.7.9 Status Bit Override

For complex timed output control, each Timer Slice has a functionality that enables the override of the status bit (CC4yST) with a value passed trough an external signal.

The override of the status bit, can then lead to a change on the output pin, CCU4xOUTy (from inactive to active or vice versa).

To enable this functionality, two signals are needed:

- One signal that acts as a trigger to override the status bit (edge active)
- One signal that contains the value to be set in the status bit (level active)

To use the status bit override functionality, one should map the signal that acts as trigger to the event number 1, by setting the required value in the **CC4yINS**.EV1IS register and indicating the active edge of the signal on the **CC4yINS**.EV1EM register.

The signal that carries the value to be set on the status bit, needs to be mapped to the event number 2, by setting the required value in the **CC4yINS**.EV2IS register. The **CC4yINS**.EV2LM register should be set to 0_B if no inversion on the signal is needed and to 1_B otherwise.

The events should be then mapped to the status bit functionality by setting the **CC4yCMC**.OFS = 1_B.

Figure 18-35 shows the functionality of the status bit override, when the external signal(1) was selected as trigger source (rising edge active) and the external signal(2) was selected as override value.

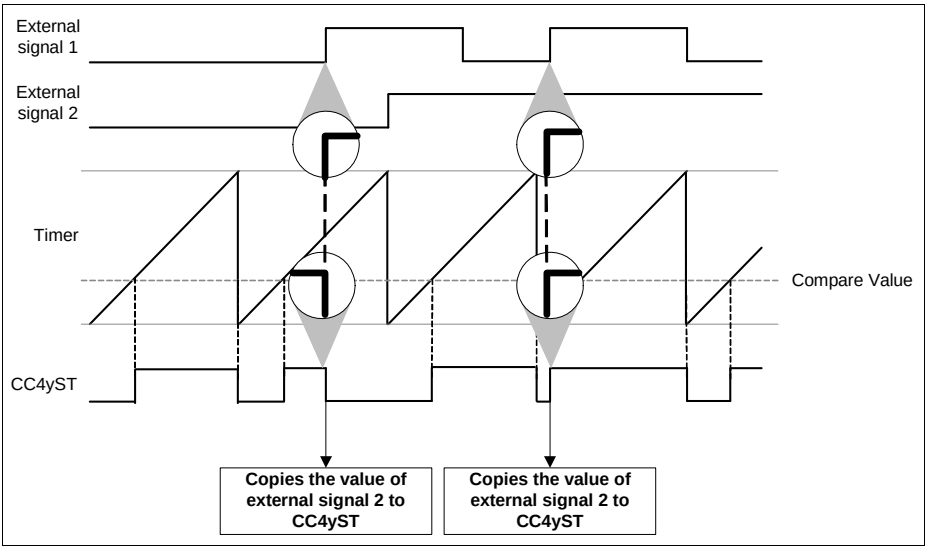


Figure 18-35 Status bit override

18.2.8 Multi-Channel Control

The multi channel control mode is selected individually in each slice by setting the **CC4yTC**.MCME = 1_B.

Within this mode, the output state of the Timer Slices (the ones set in multi channel mode) can be controlled in parallel by a single pattern.

The pattern is controlled via the CCU4 inputs, CCU4x.MCI0, CCU4x.MCI1, CCU4x.MCI2 and CCU4x.MCI3. Each of these inputs is connected directly to the associated slice input, e.g. CCU4x.MCI0 to CC40MCI, CCU4x.MCI1 to CC41MCI.

This pattern can be controlled directly by other module. The connectivity of each device may allow different control possibilities therefore one should address **Section 18.8** to check what modules are connected to these inputs.

When using the Multi Channel support of the CCU4, one can achieve a complete synchronicity between the output state update, CCU4x.OUTy, and the update of a new pattern, **Figure 18-36**. This synchronocity feature can be enabled by using some specific modules and therefore one should address **Section 18.8** to check which module is controlling the CCU4x.MCIy inputs.

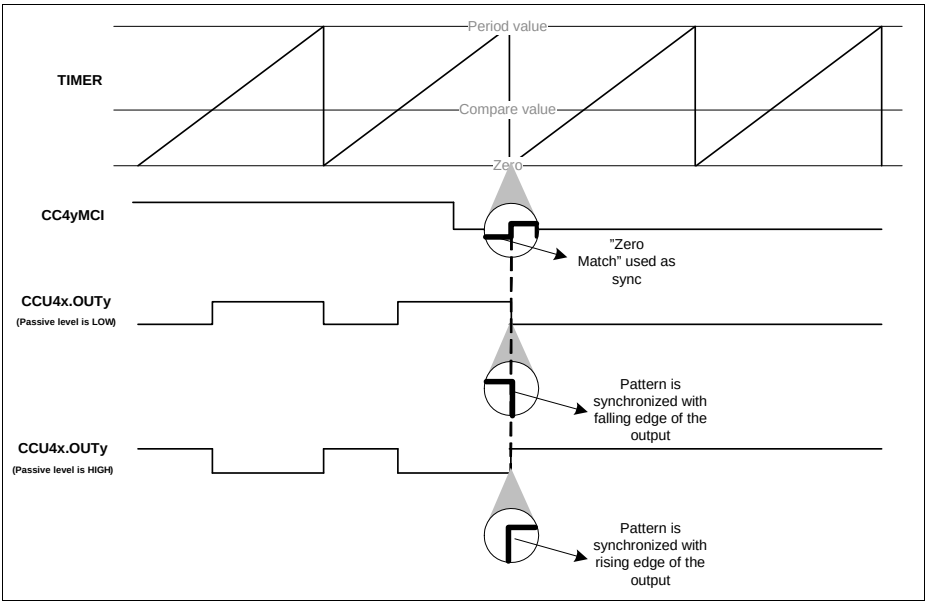


Figure 18-36 Multi channel pattern synchronization

Figure 18-37 shows the usage of the multi channel mode in conjunction with all four Timer Slices inside the CCU4.

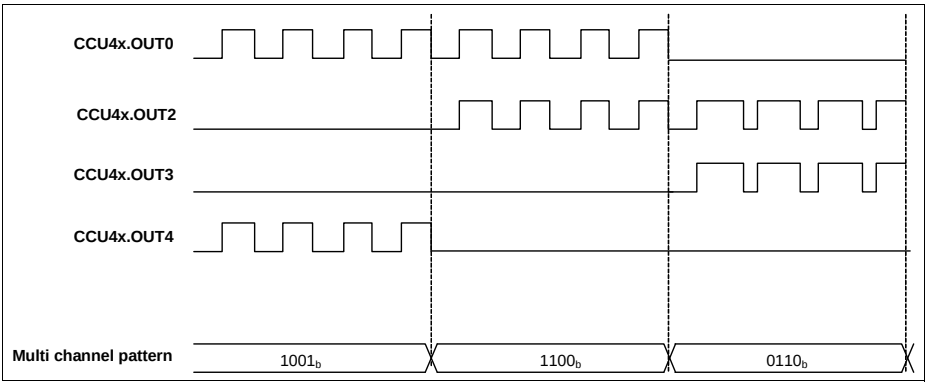


Figure 18-37 Multi Channel mode for multiple Timer Slices

The synchronization between the CCU4 and the module controlling the multi-channel pattern is achieved, by adding a 3 cycle delay on the output path of each Timer Slice

(between the status bit, CC4yST and the direct control of the output pin). This path is only selected when **CC4yTC**.MCME = 1_B, see **Figure 18-38**.

The multi pattern input synchronization can be seen on **Figure 18-39**. To achieve a synchronization between the update of the status bit, the sampling of a new multi channel pattern input is controlled by the period match or one match signal.

In a straightforward utilization of this synchronization feature, the module controlling the multi channel pattern signals, receives a sync signal from the CCU4, the CCU4x.PSy. This signal is then used by this module to update the multi-channel pattern. Due to the structure of the synchronization scheme inside the CCU4, the module controlling the multi-channel pattern needs to update this pattern, within 4 clock cycles after the CCU4x.PSy signal is asserted, **Figure 18-39**.

In a normal operation, where no external signal is used to control the counting direction, the signal used to enable the sampling of the pattern is always the period match when in edge aligned and the one match when in center aligned mode. When an external signal is used to control the counting direction, depending if the counter is counting up or counting down, the period match or the one match signal is used, respectively.

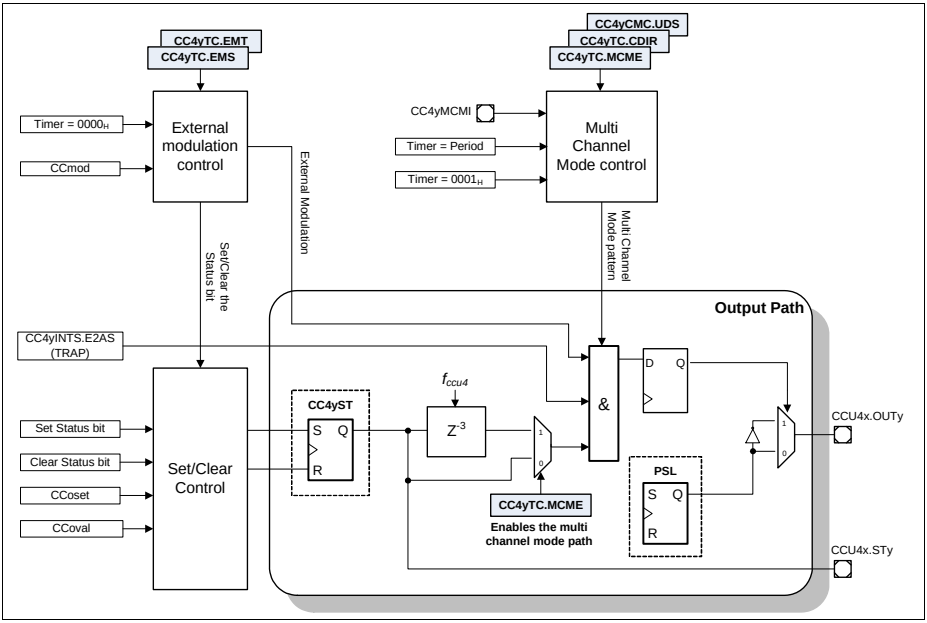


Figure 18-38 CC4y Status bit and Output Path

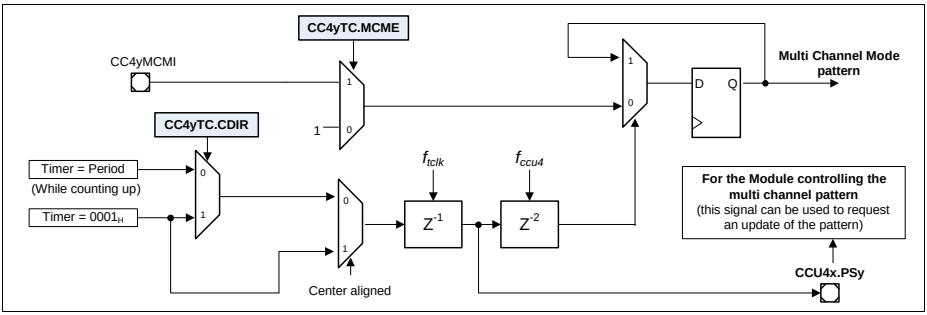


Figure 18-39 Multi Channel Mode Control Logic

18.2.9 Timer Concatenation

The CCU4 offers a very easy mechanism to perform a synchronous timer concatenation. This functionality can be used by setting the **CC4yCMC**.TCE = 1_B. By doing this the user is doing a concatenation of the actual CCU4 slice with the previous one, see **Figure 18-40**.

Notice that is not possible to perform concatenation with non adjacent slices and that timer concatenation automatically sets the slice mode into Edge Aligned. It is not possible to perform timer concatenation in Center Aligned mode.

To enable a 64 bit timer, one should set the **CC4yCMC**.TCE = 1_B in all the slices (with the exception of the CC40 due to the fact that it doesn't contain this control register).

To enable a 48 bit timer, one should set the **CC4yCMC**.TCE = 1_B in two adjacent slices and to enable a 32 bit timer, the **CC4yCMC**.TCE is set to 1_B in the slice containing the MSBs. Notice that the timer slice containing the LSBs should always have the TCE bitfield set to 0_B.

Several combinations for timer concatenation can be made inside a CCU4 module:

- one 64 bit timer
- one 48 bit timer plus a 16 timer
- two 32 bit timers
- one 32 bit timer plus two 16 bit timers

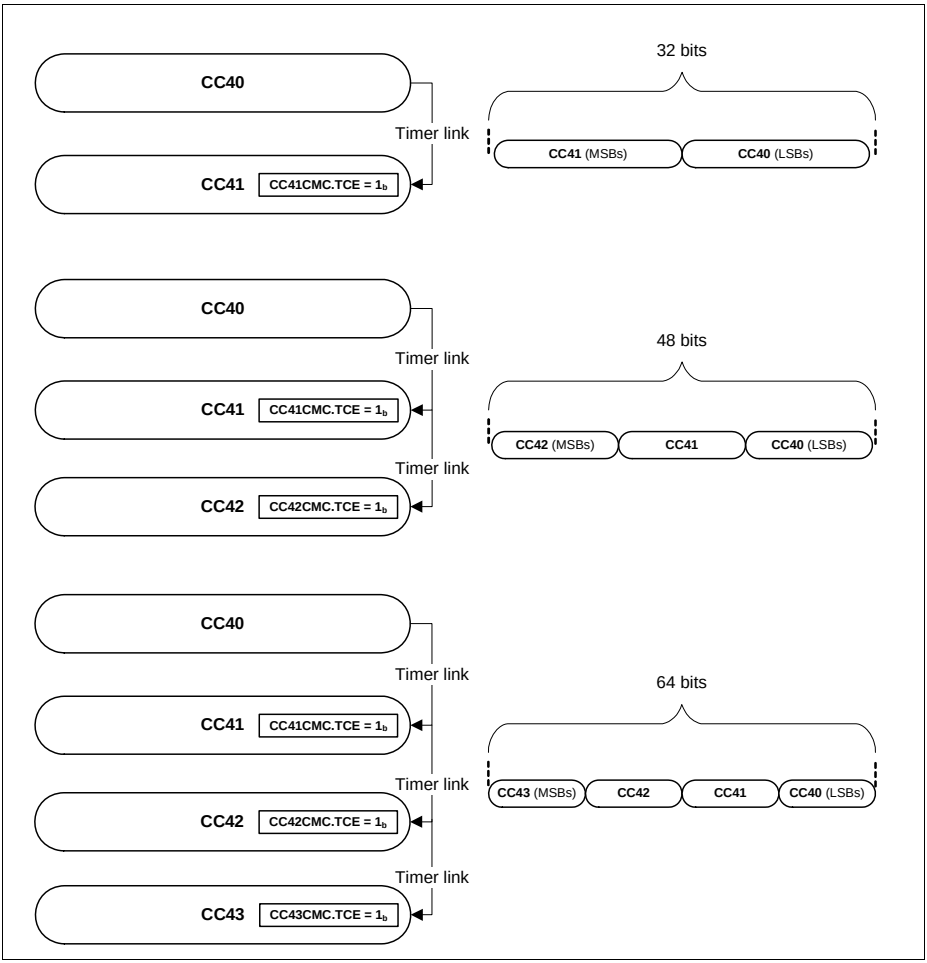


Figure 18-40 Timer Concatenation Example

Each Timer Slice is connected to the adjacent Timer Slices via a dedicated concatenation interface. This interface allows the concatenation of not only the Timer counting operation, but also a synchronous input trigger handling for capturing and loading operations, **Figure 18-41**.

Note: For all cases CC40 and CC43 are not considered adjacent slices

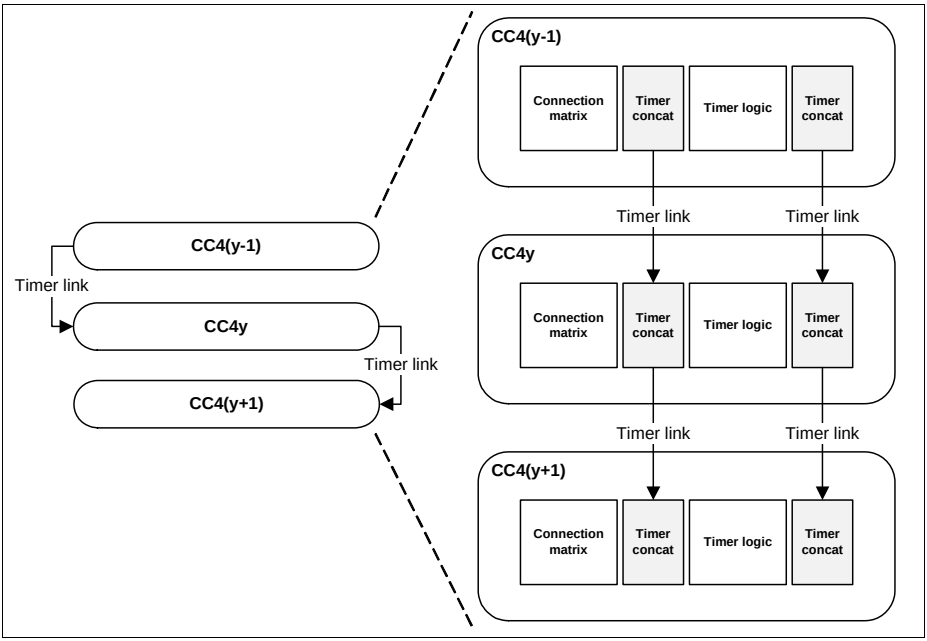


Figure 18-41 Timer Concatenation Link

Seven signals are present in the timer concatenation interface:

- Timer Period match (CC4yPM)
- Timer Zero match (CC4yZM)
- Timer Compare match (CC4yCM)
- Timer counting direction function (CCupd)
- Timer load function (CCload)
- Timer capture function for CC4yC0V and CC4yC1V registers (CCcap0)
- Timer capture function for CC4yC2V and CC4yC3V registers (CCcap1)

The first four signals are used to perform the synchronous timing concatenation at the output of the Timer Logic, like it is seen in **Figure 18-41**. With this link, the timer length can be easily adjusted to 32, 48 or 64 bits (counting up or counting down).

The last three signals are used to perform a synchronous link between the capture and load functions, for the concatenated timer system. This means that the user can have a capture or load function programmed in the first Timer Slice, and propagate this capture or load trigger synchronously from the LSBs until the MSBs, **Figure 18-42**.

The capture or load function only needs to be configured in the first Timer Slice (the one holding the LSBs). From the moment that **CC4yCMC**.TCE is set to 1_B, in the following Timer Slices, the link between these functions is done automatically by the hardware.

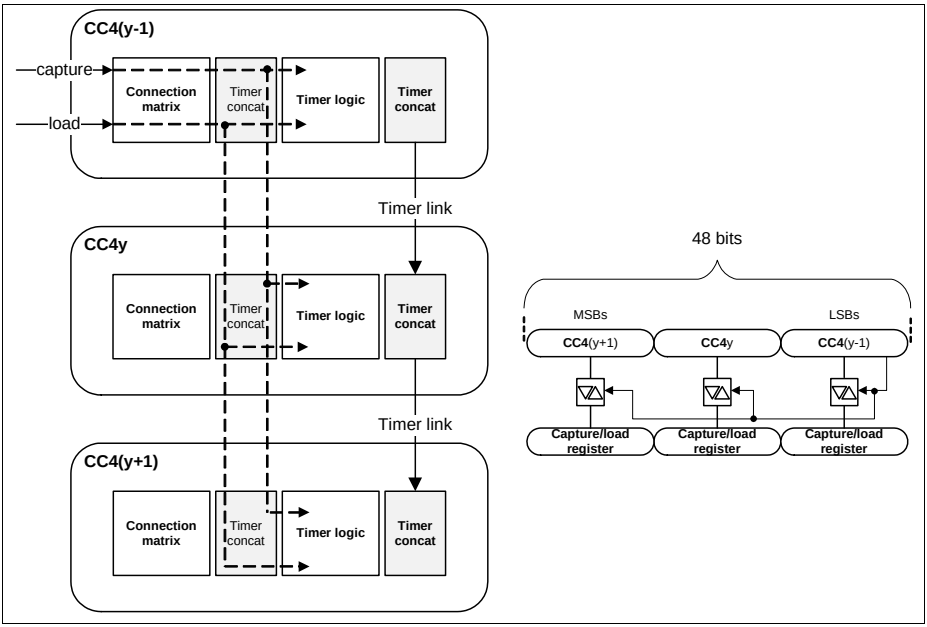


Figure 18-42 Capture/Load Timer Concatenation

The period match (CC4yPM) or zero match (CC4yZM) from the previous Timer Slice (with the immediately next lower index) are used in concatenated mode, as gating signal for the counter. This means that the counting operation of the MSBs only happens when a wrap around condition is detected, avoiding additional DSP operations to extract the counting value.

With the same methodology, the compare match (CC4yCM), zero match and period match are gated with the specific signals from the previous slice. This means that the timing information is propagated throughout all the slices, enabling a completely synchronous match between the LSB and MSB count, see **Figure 18-43**.

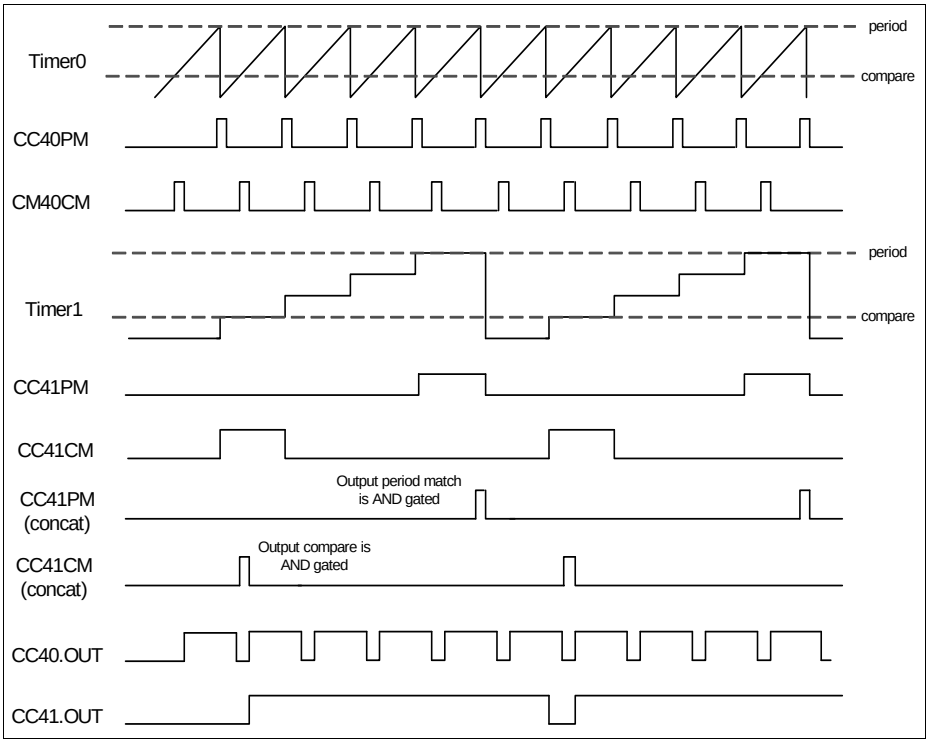


Figure 18-43 32 bit concatenation timing diagram

Note: The counting direction of the concatenated timer needs to be fixed. The timer can count up or count down, but the direction cannot be updated on the fly.

Figure 18-44 gives an overview of the timer concatenation logic. Notice that all the mechanism is controlled solely by the **CC4yCMC**.TCE bitfield.

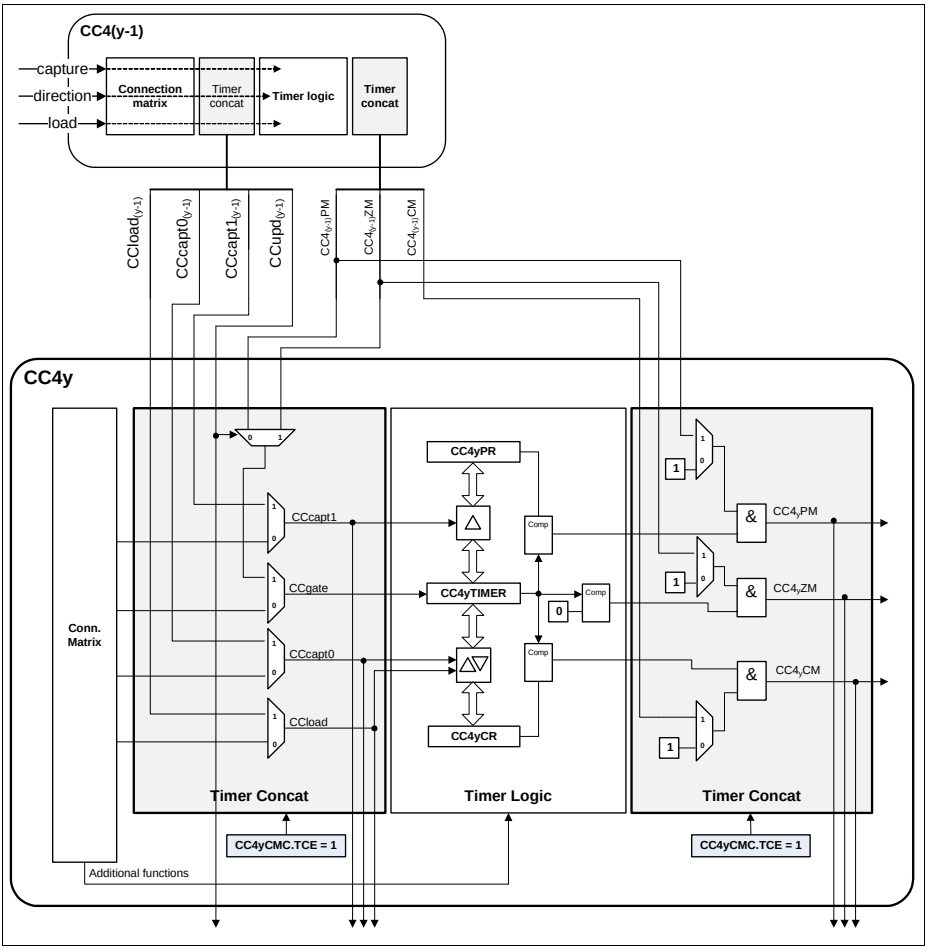


Figure 18-44 Timer concatenation control logic

18.2.10 PWM Dithering

The CCU4 has an automatic PWM dithering insertion function. This functionality can be used with very slow control loops that cannot update the period/compare values in a fast manner, and by that fact the loop can lose precision on long runs. By introducing dither on the PWM signal, the average frequency/duty cycle is then compensated against that error.

Each slice contains a dither control unit, see **Figure 18-45**.

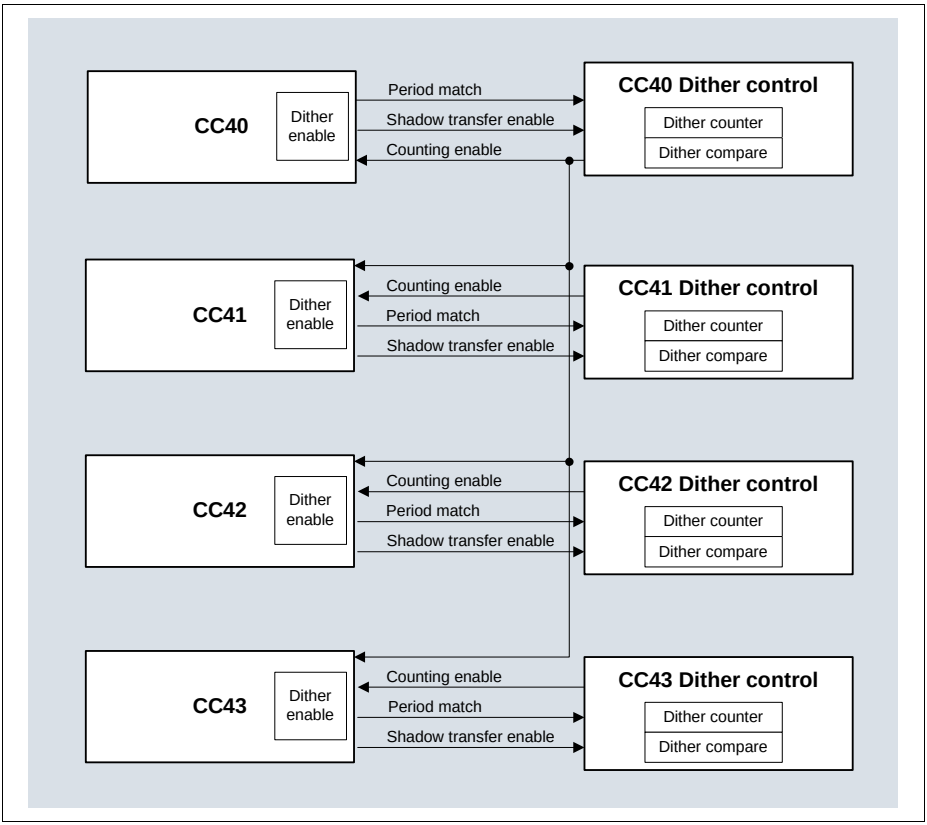


Figure 18-45 Dither structure overview

The dither control unit contains a 4 bit counter and a compare value. The four bit counter is incremented every time that a period match occurs. The counter works in a bit reverse mode so the distribution of increments stays uniform over 16 counter periods, see **Table 18-5**.

Table 18-5 Dither bit reverse counter

counter[3]	counter[2]	counter[1]	counter[0]
0	0	0	0
1	0	0	0
0	1	0	0
1	1	0	0
0	0	1	0
1	0	1	0
0	1	1	0
1	1	1	0
0	0	0	1
1	0	0	1
0	1	0	1
1	1	0	1
0	0	1	1
1	0	1	1
0	1	1	1
1	1	1	1

The counter is then compared against a programmed value, **CC4yDIT**.DCV. If the counter value is smaller than the programmed value, a gating signal is generated that can be used to extend the period, to delay the compare or both (controlled by the **CC4yTC**.DITHE field, see **Table 18-6**) for one clock cycle.

Table 18-6 Dither modes

DITHE[1]	DITH[0]	Mode
0	0	Dither is disabled
0	1	Period is increased by 1 cycle
1	0	Compare match is delayed by 1 cycle
1	1	Period is increased by 1 cycle and compare is delayed by 1 cycle

The dither compare value also has an associated shadow register that enables concurrent update with the period/compare register of CC4y. The control logic for the dithering unit is represented on **Figure 18-46**.

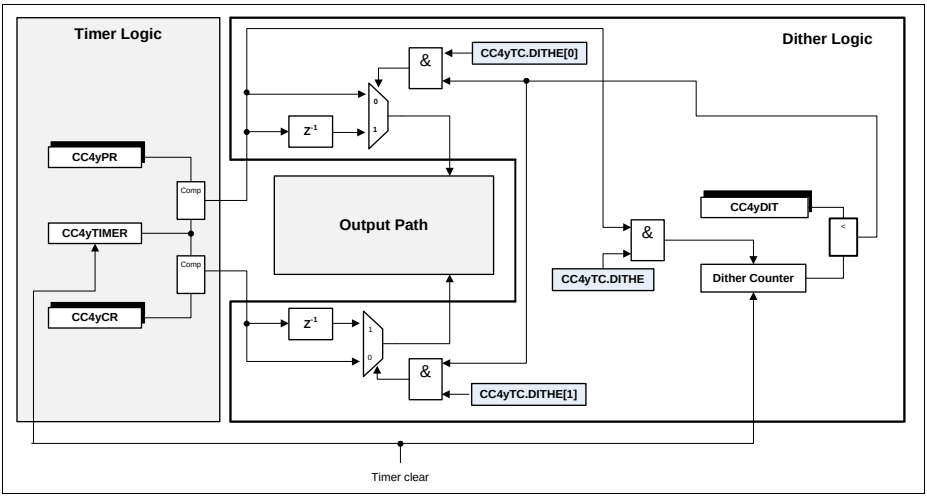


Figure 18-46 Dither control logic

Figure 18-47 to **Figure 18-52** show the effect of the different configurations for the dithering function, **CC4yTC**.DITHE, for both counting schemes, Edge and Center Aligned mode. In each figure, the bit reverse scheme is represented for the dither counter and the compare value was programmed with the value 8_H. In each figure, the variable T, represents the period of the counter, while the variable d indicates the duty cycle (status bit is set HIGH).

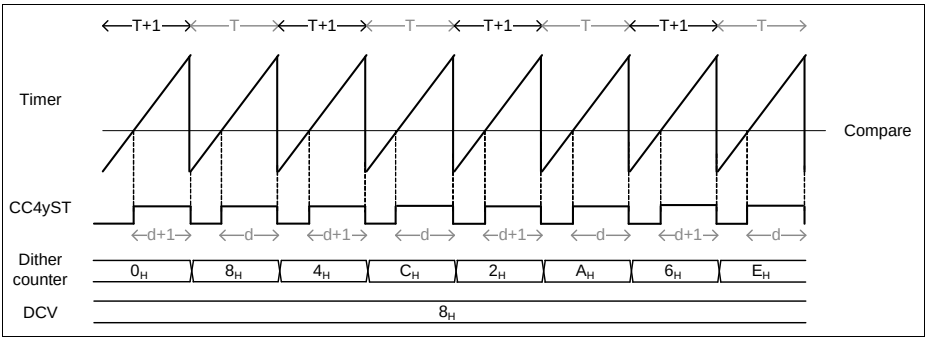


Figure 18-47 Dither timing diagram in edge aligned - CC4yTC.DITHE = 01_B

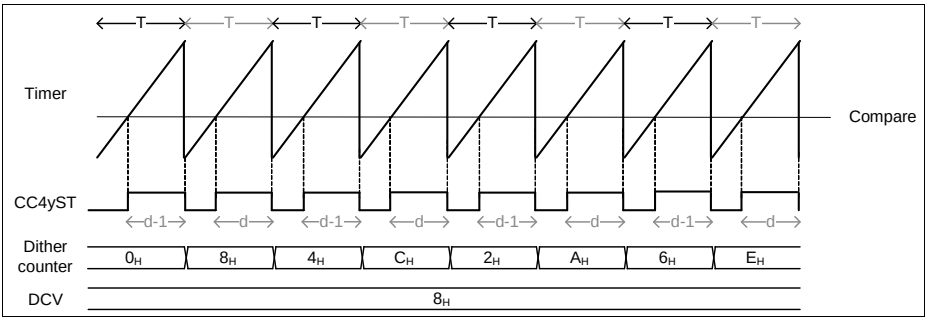


Figure 18-48 Dither timing diagram in edge aligned - CC4yTC.DITHE = 10_B

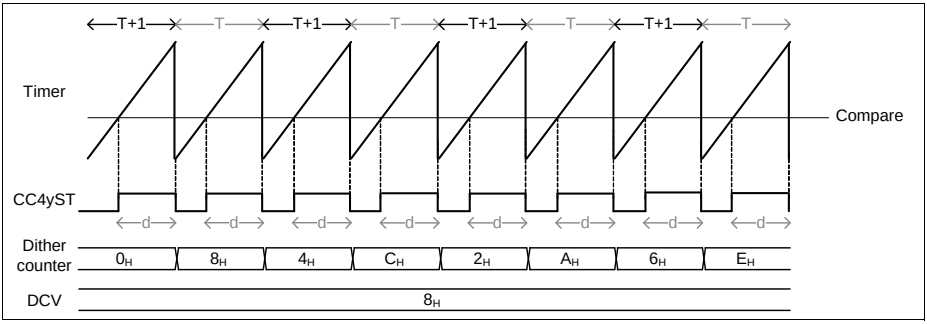


Figure 18-49 Dither timing diagram in edge aligned - CC4yTC.DITHE = 11_B

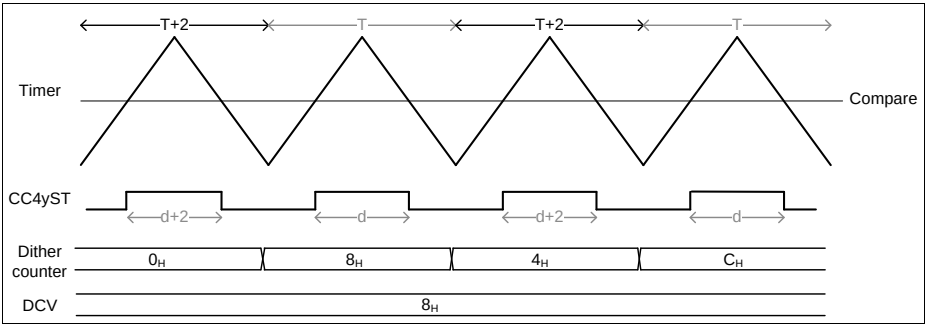


Figure 18-50 Dither timing diagram in center aligned - CC4yTC.DITHE = 01_B

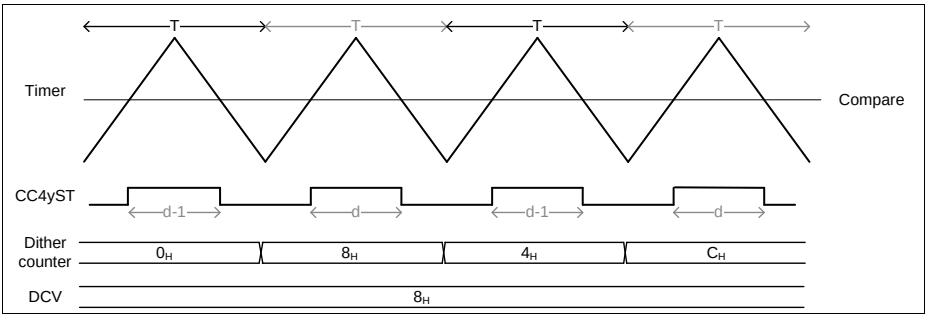


Figure 18-51 Dither timing diagram in center aligned - CC4yTC.DITHE = 10_B

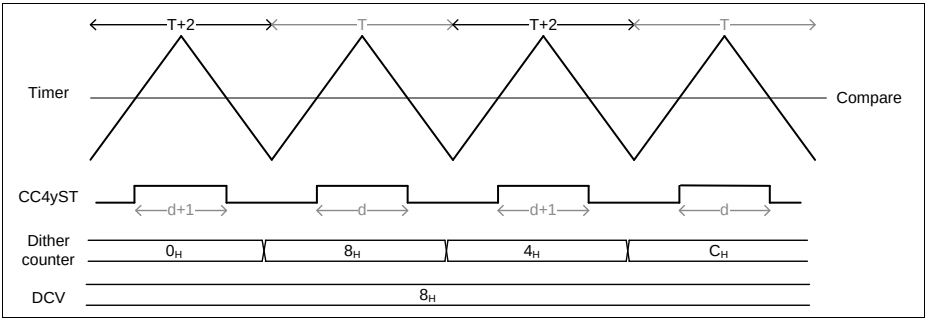


Figure 18-52 Dither timing diagram in center aligned - CC4yTC.DITHE = 11_B

Note: When using the dither, is not possible to select a period value of FS when in edge aligned mode. In center aligned mode, the period value must be at least FS - 2.

18.2.11 Prescaler

The CCU4 contains a 4 bit prescaler that can be used in two operating modes for each individual slice:

- normal prescaler mode
- floating prescaler mode

The run bit of the prescaler can be set/cleared by SW by writing into the registers, **GIDLC**.SPRB and **GIDLS**.CPRB respectively, and it can also be cleared by the run bit of a specific slice. With the last mechanism, the run bit of the prescaler is cleared one clock cycle after the clear of the run bit of the selected slice. To select which slice can perform this action, one should program the **GCTRL**.PRBC register.

18.2.11.1 Normal Prescaler Mode

In Normal prescaler mode the clock fed to the CC4y counter is a normal fixed division by N, accordingly to the value set in the **CC4yPSC**.PSIV register. The values for the possible division values are listed in **Table 18-7**. The **CC4yPSC**.PSIV value is only modified by a SW access. Notice that each slice has a dedicated prescaler value selector (**CC4yPSC**.PSIV), which means that the user can select different counter clocks for each Timer Slice (CC4y).

Table 18-7 Timer clock division options

CC4yPSC.PSIV	**Resulting clock**
0000_B	f_{ccu4}
0001_B	$f_{ccu4}/2$
0010_B	$f_{ccu4}/4$
0011_B	$f_{ccu4}/8$
0100_B	$f_{ccu4}/16$
0101_B	$f_{ccu4}/32$
0110_B	$f_{ccu4}/64$
0111_B	$f_{ccu4}/128$
1000_B	$f_{ccu4}/256$
1001_B	$f_{ccu4}/512$
1010_B	$f_{ccu4}/1024$
1011_B	$f_{ccu4}/2048$
1100_B	$f_{ccu4}/4096$
1101_B	$f_{ccu4}/8192$
1110_B	$f_{ccu4}/16384$
1111_B	$f_{ccu4}/32768$

18.2.11.2 Floating Prescaler Mode

The floating prescaler mode can be used individually in each slice by setting the register **CC4yTC**.FPE = 1_B. With this mode, the user can not only achieve a better precision on the counter clock for compare operations but also reduce the SW read access for the capture mode.

The floating prescaler mode contains additionally to the initial configuration value register, **CC4yPSC**.PSIV, a compare register, **CC4yFPC**.PCMP with an associated shadow register mechanism.

Figure 18-53 shows the structure of the prescaler in floating mode when the specific slice is in compare mode (no external signal is used for capture). In this mode, the value of the clock division is increment by 1_D every time that a timer overflow/underflow (overflow if in Edge Aligned Mode, underflow if in Center Aligned Mode) occurs.

In this mode, the Compare Match from the timer is AND gated with the Compare Match of the prescaler and every time that this event occurs, the value of the clock division is updated with the **CC4yPSC**.PSIV value in the immediately next timer overflow/underflow event.

The shadow transfer of the floating prescaler compare value, **CC4yFPC**.PCMP, is done following the same rules described on **Section 18.2.5.2**.

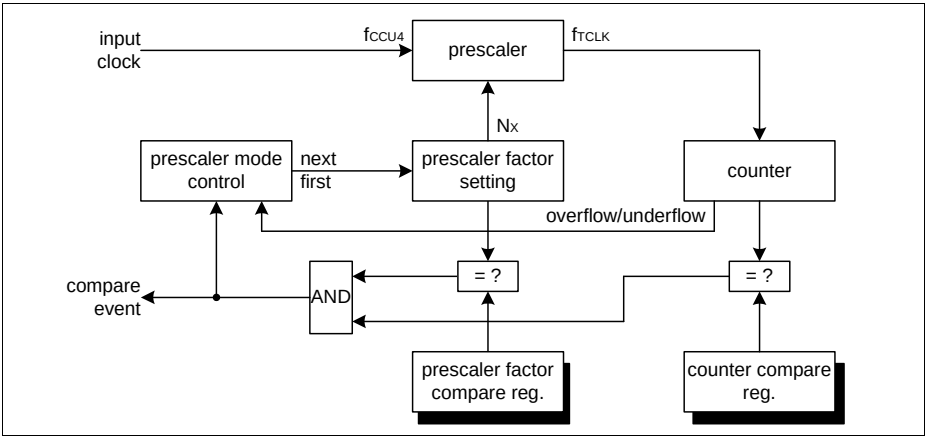


Figure 18-53 Floating prescaler in compare mode overview

When the specific CC4y is operating in capture mode (when at least one external signal is decoded as capture functionality), the actual value of the clock division also needs to be stored every time that a capture event occurs. The floating prescaler can have up to 4 capture registers (the maximum number of capture registers is dictated by the number of capture registers used in the specific slice).

The clock division value continues to be increment by 1_D every time that a timer overflow (in capture mode, the slice is always operating in Edge Aligned Mode) occurs and it is loaded with the PSIV value every time that a capture triggers is detected.

See the **Section 18.2.12.2** for a full description of the usage of the floating prescaler mode in conjunction with compare and capture modes.

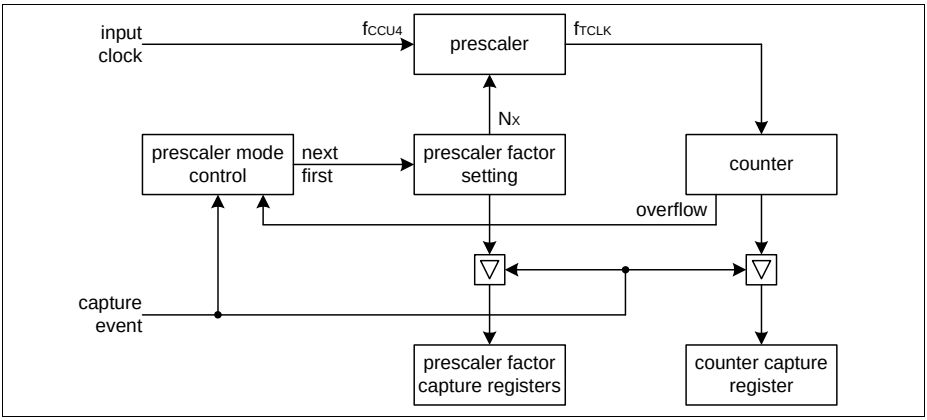


Figure 18-54 Floating Prescaler in capture mode overview

18.2.12 CCU4 Usage

18.2.12.1 PWM Signal Generation

The CCU4 offers a very flexible range in duty cycle configurations. This range is comprised between 0 to 100%.

To generate a PWM signal with a 100% duty cycle in Edge Aligned Mode, one should program the compare value, **CC4yCR**.CR, to 0000_H, see **Figure 18-55**.

In the same manner a 100% duty cycle signal can be generated in Center Aligned Mode, see **Figure 18-56**.

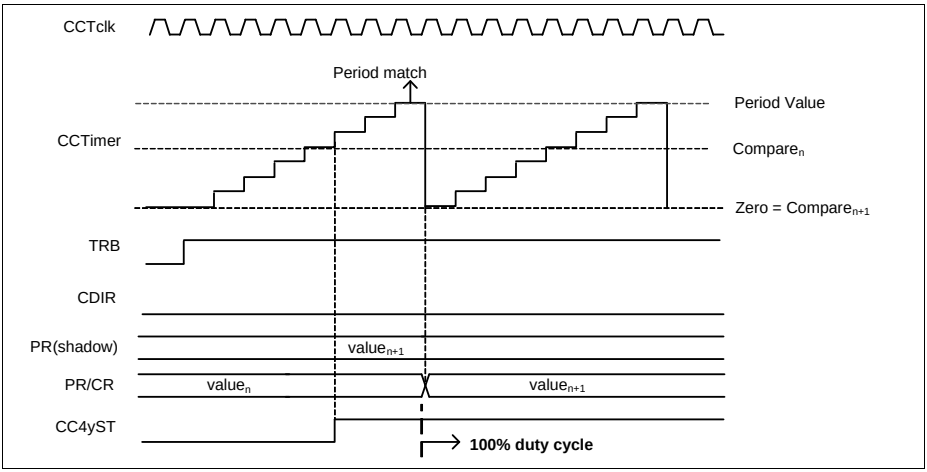


Figure 18-55 PWM with 100% duty cycle - Edge Aligned Mode

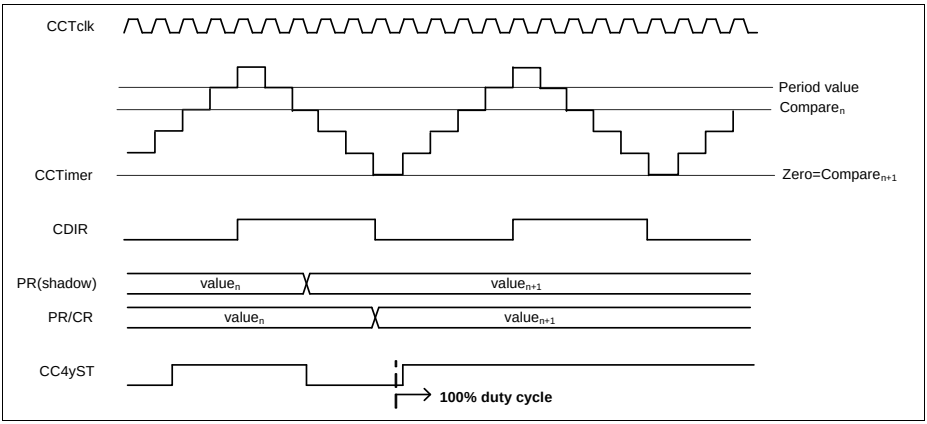


Figure 18-56 PWM with 100% duty cycle - Center Aligned Mode

To generate a PWM signal with 0% duty cycle in Edge Aligned Mode, the compare register should be set with the value programmed into the period value plus 1. In the case that the timer is being used with the full 16 bit capability (counting from 0 to 65535), setting a value bigger than the period value into the compare register is not possible and therefore the smallest duty cycle that can be achieved is 1/FS, see **Figure 18-57**.

In Center Aligned Mode, the counter is never running from 0 to 65535_D, due to the fact that it has to overshoot for one clock cycle the value set in the period register. Therefore the user never has a FS counter, which means that generating a 0% duty cycle signal is

always possible by setting a value in the compare register bigger than the one programmed into the period register, see **Figure 18-58**.

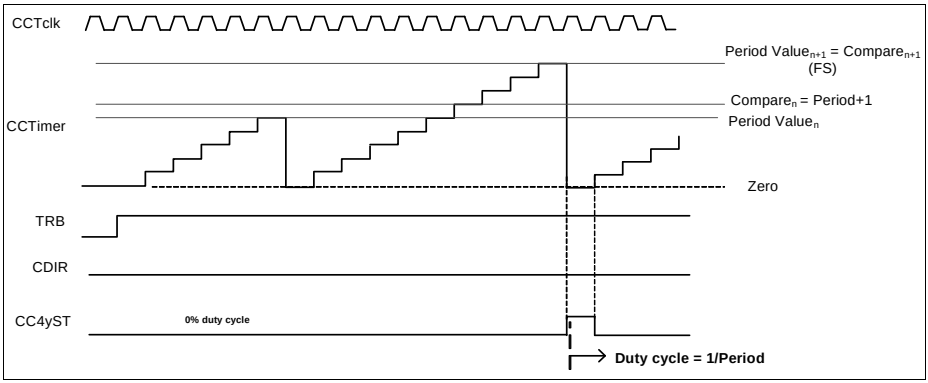


Figure 18-57 PWM with 0% duty cycle - Edge Aligned Mode

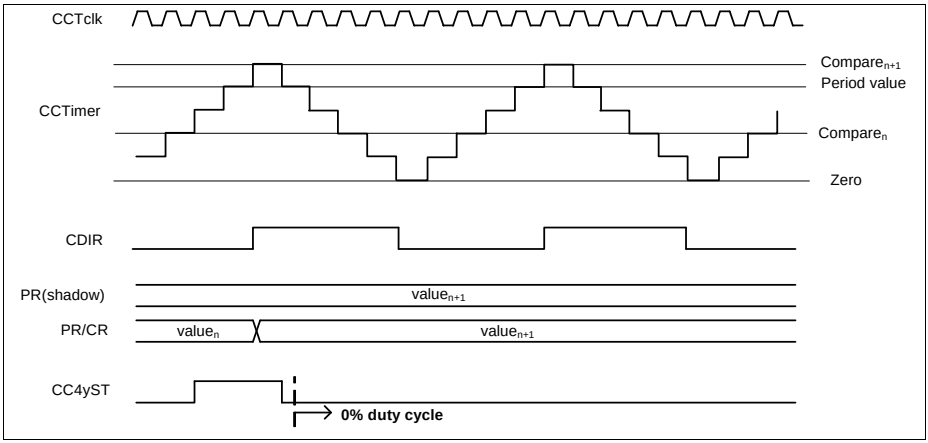


Figure 18-58 PWM with 0% duty cycle - Center Aligned Mode

18.2.12.2 Prescaler Usage

In Normal Prescaler Mode, the frequency of the f_{tclk} fed to the specific CC4y is chosen from the **Table 18-7**, by setting the **CC4yPSC**.PSIV with the required value.

In Floating Prescaler Mode, the frequency of the f_{tclk} can be modified over a selected timeframe, within the values specified in **Table 18-7**. This mechanism is specially useful if, when in capture mode, the dynamic of the capture triggers is very slow or unknown.

In Capture Mode, the Floating Prescaler value is incremented by 1 every time that a timer overflow happens and it is set with the initial programmed value when a capture event happens, see **Figure 18-59**.

When using the Floating Prescaler Mode in Capture Mode, the timer should be cleared each time that a capture event happens, **CC4yTC**.CAPC = 11_B. By operating the Capture mode in conjunction with the Floating Prescaler, even for capture signals that have a periodicity bigger that 16 bits, it is possible to use just a single CCU4 Timer Slice without monitoring the interrupt event of the timer overflow, cycle by cycle. For this the user just needs to know what is the timer captured value and the actual prescaler configuration at the time that the capture event occurred. These values are contained in each CC4yCxV register.

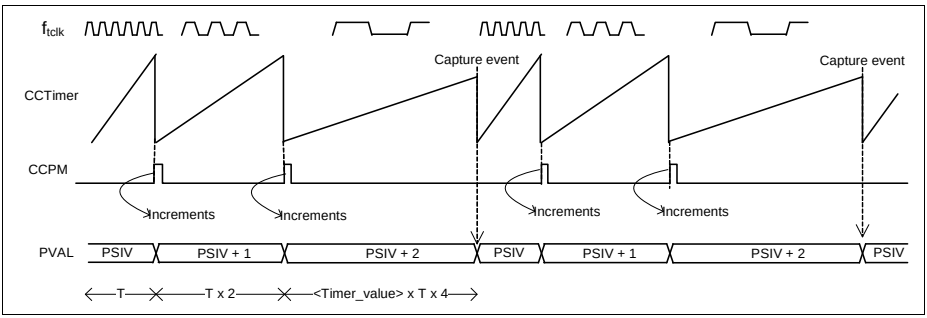


Figure 18-59 Floating Prescaler capture mode usage

When in Compare Mode, the Floating Prescaler function may be used to achieve a fractional PWM frequency or to perform some frequency modulation.

The same incrementing by 1_D mechanism is done every time that a overflow/underflow of the Timer occurs and the actual Prescaler value, doesn't match the one programmed into the **CC4yFPC**.PCMP register.

When a Compare Match from the Timer occurs and the actual Prescaler value is equal to the one programmed on the **CC4yFPC**.PCMP register, then the Prescaler value is set with the initial value, **CC4yPSC**.PSIV, when the next occurrence of a timer overflow/underflow.

In **Figure 18-60**, the Compare value of the Floating Prescaler was set to PSIV + 2. Every time that a timer overflow occurs, the value of the Prescaler is incremented by 1, which means that if we give f_{tclk} as the reference frequency for the **CC4yPSC**.PSIV value, we have $f_{tclk}/2$ for **CC4yPSC**.PSIV + 1 and $f_{tclk}/4$ for **CC4yPSC**.PSIV + 2.

The period over time of the counter becomes:

$$Period = (1/f_{tclk} + 2/f_{tclk} + 4/f_{tclk}) / 3 \quad (18.10)$$

The same mechanism is used in Center Aligned Mode, but to keep the rising arcade and falling arcade always symmetrical, instead of the overflow of the timer, the underflow is used, see **Figure 18-61**.

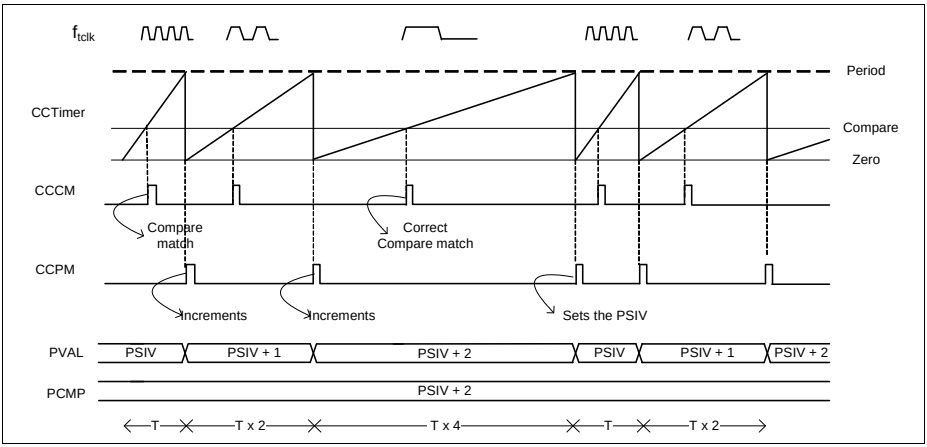


Figure 18-60 Floating Prescaler compare mode usage - Edge Aligned

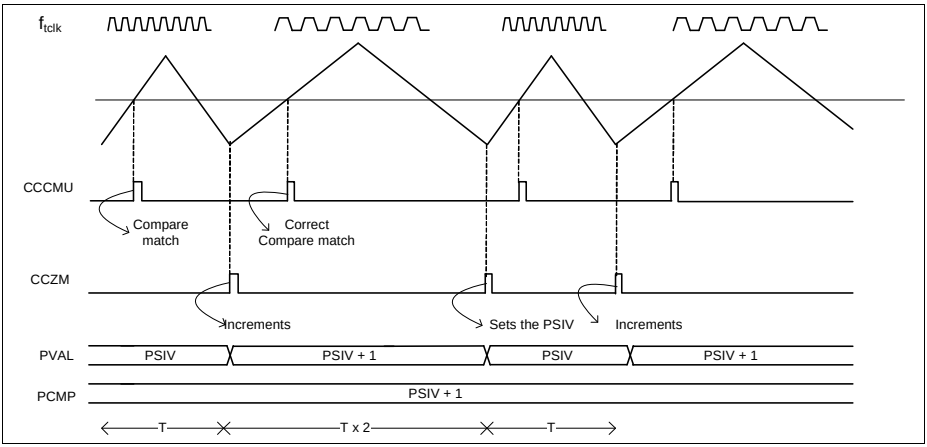


Figure 18-61 Floating Prescaler compare mode usage - Center Aligned

18.2.12.3 PWM Dither

The Dither functionality can be used to achieve a very fine precision on the periodicity of the output state in compare mode. The value set in the dither compare register, **CC4yDIT**.DCV is crosschecked against the actual value of the dither counter and every

time that the dither counter is smaller than the comparison value one of the follows actions is taken:

- The period is extended for 1 clock cycle - **CC4yTC**.DITHE = 01_B; in edge aligned mode
- The period is extended for 2 clock cycles - **CC4yTC**.DITHE = 01_B; in center aligned mode
- The comparison match while counting up (**CC4yTCST**.CDIR = 0_B) is delayed (this means that the status bit is going to stay in the SET state 1 cycle less) for 1 clock cycle - **CC4yTC**.DITHE = 10_B;
- The period is extended for 1 clock cycle and the comparison match while counting up is delayed for 1 clock cycle - **CC4yTC**.DITHE = 11_B; in edge aligned mode
- The period is extended for 2 clock cycles and the comparison match while counting up is delayed for 1 clock cycle; center aligned mode

The bit reverse counter distributes the number programmed in the **CC4yDIT**.DCV throughout a window of 16 timer periods.

Table 18-8 describes the bit reverse distribution versus the programmed value on the **CC4yDIT**.DCV field. The fields marked as '0' indicate that in that counter period, one of the above described actions, is going to be performed.

Table 18-8 Bit reverse distribution

	DCV															
Dither counter	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
4	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
C	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
2	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
A	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0
6	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
E	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0
5	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
D	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
3	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
B	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0

Table 18-8 Bit reverse distribution (cont'd)

	DCV															
Dither counter	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
7	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
F	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

The bit reverse distribution versus the programmed **CC4yDIT**.DCV value results in the following values for the Period and duty cycle:

DITHE = 01_B

Period = [(16 - DCV) x T + DCV x (T + 1)]/16; in Edge Aligned Mode (18.11)

Duty cycle = [(16 - DCV) x d/T + DCV x (d+1)/(T + 1)]/16; in Edge Aligned Mode (18.12)

Period = [(16 - DCV) x T + DCV x (T + 2)]/16; in Center Aligned Mode (18.13)

Duty cycle = [(16 - DCV) x d/T + DCV x (d+2)/(T + 2)]/16; in Center Aligned Mode (18.14)

DITHE = 10_B

Period = T ; in Edge Aligned Mode (18.15)

Duty cycle = [(16 - DCV) x d/T + DCV x (d-1)/T]/16; in Edge Aligned Mode (18.16)

Period = T ; in Center Aligned Mode (18.17)

Duty cycle = [(16 - DCV) x d/T + DCV x (d-1)/T]/16; in Center Aligned Mode (18.18)

DITHE = 11_B

Period = [(16 - DCV) x T + DCV x (T + 1)]/16; in Edge Aligned Mode (18.19)

Duty cycle = [(16 - DCV) x d/T + DCV x d/(T + 1)]/16; in Edge Aligned Mode (18.20)

Period = [(16 - DCV) x T + DCV x (T + 2)]/16; in Center Aligned Mode (18.21)

Duty cycle = [(16 - DCV) x d/T + DCV x (d+1)/(T + 2)]/16; in Center Aligned Mode (18.22)

where:

T - Original period of the signal, see **Section 18.2.5.1**

d - Original duty cycle of the signal, see **Section 18.2.5.1**

18.2.12.4 Capture Mode Usage

Each Timer Slice can make use of 2 or 4 capture registers. Using only 2 capture registers means that only 1 Event was linked to a captured trigger. To use the four capture registers, both capture triggers need to be mapped into an Event (it can be the same signal with different edges selected or two different signals) or the **CC4yTC**.SCE field needs to be set to 1, which enables the linking of the 4 capture registers.

The internal slice mechanism for capturing is the same for the capture trigger 1 or capture trigger 0.

Different Capture Events - SCE = 0_B

Capture trigger 1 (CCcapt1) is appointed to the capture register 2, **CC4yC2V** and capture register 3, **CC4yC3V**, while trigger 0 (CCcapt0) is appointed to capture register 1, **CC4yC1V** and 0, **CC4yC0V**.

In each CCcapt0 event, the timer value is stored into **CC4yC1V** and the value of the **CC4yC1V** is transferred into the **CC4yC0V**.

In each CCcapt1 event, the timer value is stored into capture register **CC4yC3V** and the value of the capture register **CC4yC3V** is transferred into **CC4yC2V**.

The capture/transfer mechanism only happens if the specific register is not full. A capture register becomes full when receives a new value and becomes empty after the SW has read back the value.

The full flag is cleared every time that the SW reads back the **CC4yC0V**, **CC4yC1V**, **CC4yC2V** or **CC4yC3V** register. The SW can be informed of a new capture trigger by enabling the interrupt source linked to the specific Event. This means that every time that a capture is made an interrupt pulse is generated.

In the case that the Floating Prescaler Mode is being used, the actual value of the clock division is also stored in the capture register (CC4yCxV).

Figure 18-62 shows an example of how the capture/transfer may be used in a Timer Slice that is using a external signal as count function (to measure the velocity of a rotating device), and an equidistant capture trigger that is used to dictate the timestamp for the velocity calculation (two Timer waveforms are plotted, one that exemplifies the clearing of the timer in each capture event and another without the clearing function active).

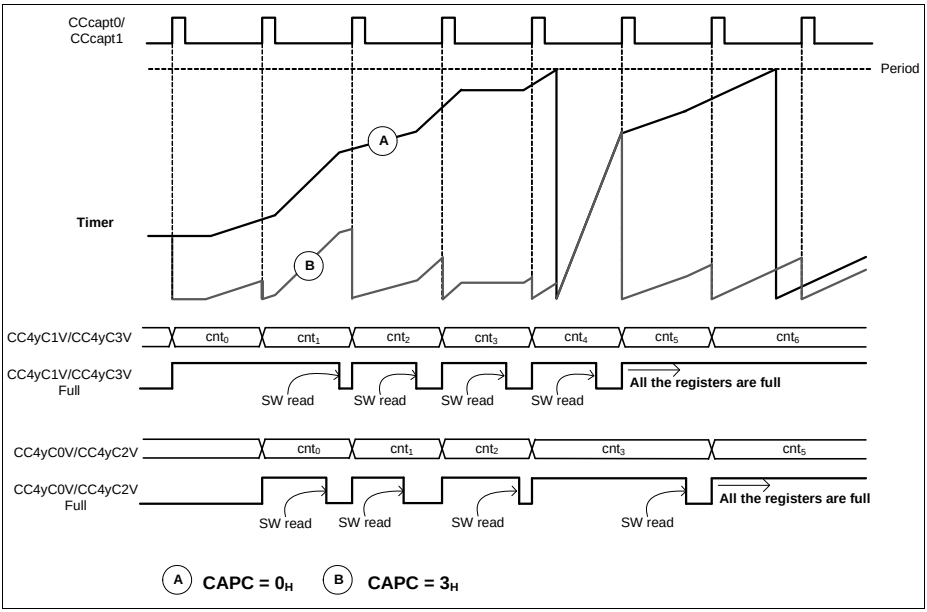


Figure 18-62 Capture mode usage - single channel

Same Capture Event - SCE = 1_B

If the **CC4yTC**.SCE is set to 1_B, all the four capture registers are chained together, emulating a fifo with a depth of 4. In this case, only the capture trigger 1, CCcapt1, is used to perform a capture event.

As an example for this mode, one can consider the case where one Timer Slice is being used in capture mode with SCE = 1_B, with another external signal that controls the counting. This timer slice can be incremented at different speeds, depending on the frequency of the counting signal.

An additional Timer Slice is used to control the capture trigger, dictating the time stamp for the capturing.

A simple scheme for this can be seen in **Figure 18-63**. The CC40ST output of slice 0 was used as capture trigger in the CC41 slice (active on rising and falling edge). The CC40ST output is used as known timebase marker, while the slice timer used for capture is being controlled by external events, e.g. external count.

Due to the fact that we have available 4 capture registers, every time that the SW reads back the complete set of values, 3 speed profiles can be measured.

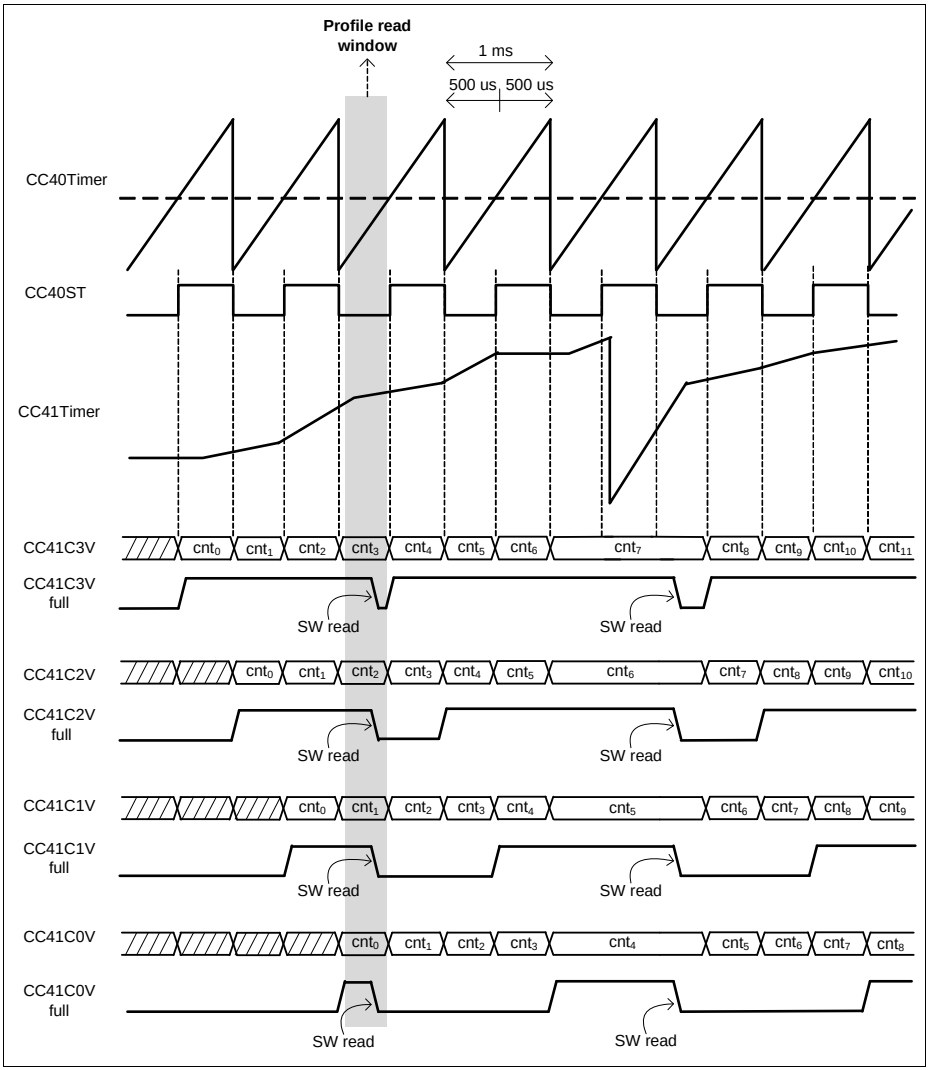


Figure 18-63 Three Capture profiles - CC4yTC.SCE = 1_B

To calculate the three different profiles in **Figure 18-63**, the 4 capture registers need to be read during the pointed read window. After that, the profile calculation is done:

Profile 1 = $CC41C1V_{info}$ - $CC41C0V_{info}$

Profile 2 = $CC41C2V_{info}$ - $CC41C1V_{info}$

Profile 3= $CC41C3V_{info}$ - $CC41C2V_{info}$

Note: This is an example and therefore several Timer Slice configurations and software loops can be implemented.

Extended Read Back Mode

When multiple Timer Slices need to be programmed into capture mode, it may not be suitable to distribute them over several CCU4 modules. This may be due to resource optimization or availability of Direct Memory Access (DMA) channels.

A simple way to overcome this issue, is to use the Extended Capture Read functionality of CCU4. This mode can be programmed independently for each and every Timer Slice via the **CC4yTC**.ECM bitfield.

The advantage of this mode is that there is only one associated read address for all the capture registers (note that the individual capture registers are still accessible), the **ECRD**. With this one can achieve a DMA channel compression and a better Timer Slice resource optimization through the entire device.

Figure 18-64 exemplifies the usage of the Extended Capture Read function. In this example we have three different Timer Slices that are used to monitor three different applications (in capture mode). An additional Timer Slice (notice that it doesn't need to be in the same CCU4 module) is used to trigger the DMA read of the capture registers.

The read back trigger periodicity can also be updated on the fly to adjust to different system states or operation modes.

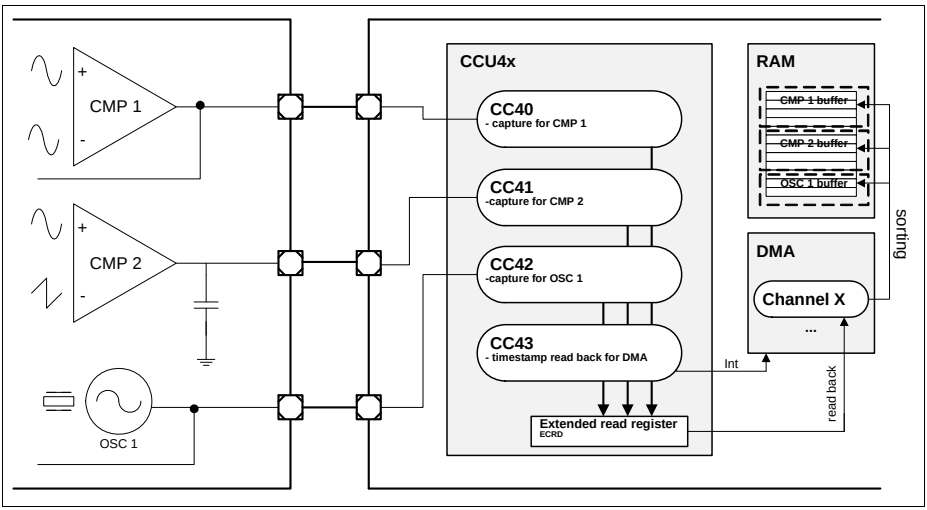


Figure 18-64 Extended read usage scheme example

Every time that the software reads back the **ECRD** register, the CCU4 returns the value of a specific capture register that contains new captured data. The read access of the capture registers follows a circular scheme that is maintained internally by the CCU4, **Figure 18-65**.

For the timer slices that are in capture mode but do not have **CC4yTC**.ECM = 1_B, their captured register values are also read back through the **ECRD**. However, the full flag of the capture registers is not cleared (it is only cleared via a read access to the specific CC4yCxV register).

Only the capture registers of the slices with **CC4yTC**.ECM = 1_B have their full flag cleared with a read access via **ECRD**.

On **Figure 18-66** an example time line is given, in which all the slices were programmed to use extended capture mode, **CC4yTC**.ECM = 1_B. In this example, one can see that the CCU4 doesn't keep memory of which was the first or last captured value between the Timer Slices. Like described on **Figure 18-66**, the read back pointer is incremented until a capture register that has the full flag set, is found.

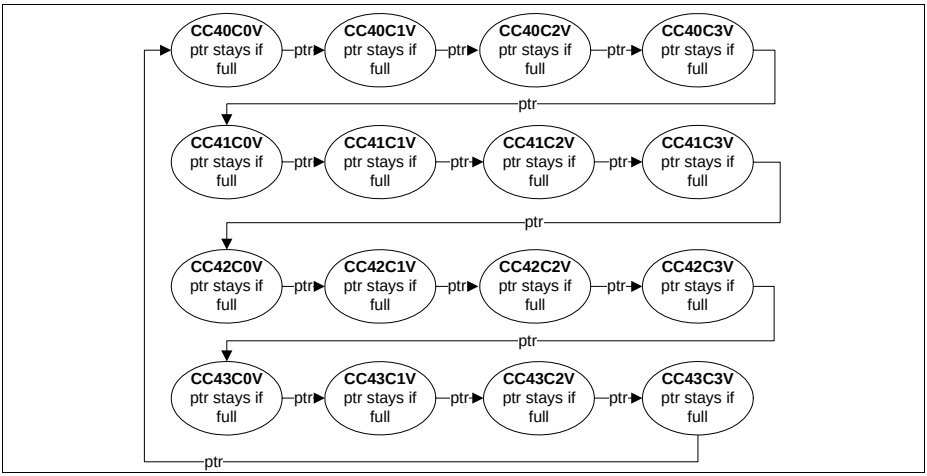


Figure 18-65 Extended Capture Read back

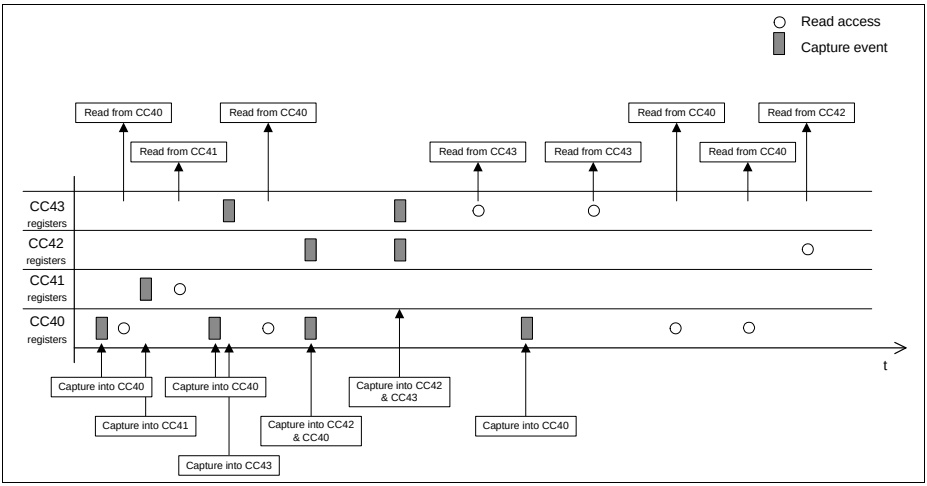


Figure 18-66 Extended Capture Access Example

18.3 Service Request Generation

Each CCU4 slice has an interrupt structure as the one in **Figure 18-67**. The register **CC4yINTS** is the status register for the interrupt sources. Each dedicated interrupt source can be set or cleared by SW, by writing into the specific bit in the **CC4ySWS** and **CC4ySWR** registers respectively.

Each interrupt source can be enabled/disabled via the **CC4yINTE** register. An enabled interrupt source will always generate a pulse on the service request line even if the specific status bit was not cleared. **Table 18-9** describes the interrupt sources of each CCU4 slice.

The interrupt sources, Period Match while counting up and one Match while counting down are ORed together. The same mechanism is applied to the Compare Match while counting up and Compare Match while counting down.

The interrupt sources for the external events are directly linked with the configuration set on the **CC4yINS**.EVxEM. If an event is programmed to be active on both edges, that means that service request pulse is going to be generated when any transition on the external signal is detected. If the event is linked with a level function, the **CC4yINS**.EVxEM still can be programmed to enable a service request pulse. The TRAP event doesn't need any of extra configuration for generating the service request pulse when the slice enters the TRAP state.

Table 18-9 Interrupt sources

Signal	Description
CCINEV0_E	Event 0 edge(s) information from event selector. Used when an external signal should trigger an interrupt.
CCINEV1_E	Event 1 edge(s) information from event selector. Used when an external signal should trigger an interrupt.
CCINEV2_E	Event 2 edge(s) information from event selector. Used when an external signal should trigger an interrupt.
CCPM_U	Period Match while counting up
CCCM_U	Compare Match while counting up
CCCM_D	Compare Match while counting down
CCOM_D	One Match while counting down
Trap state set	Entering Trap State. Will set the E2AS

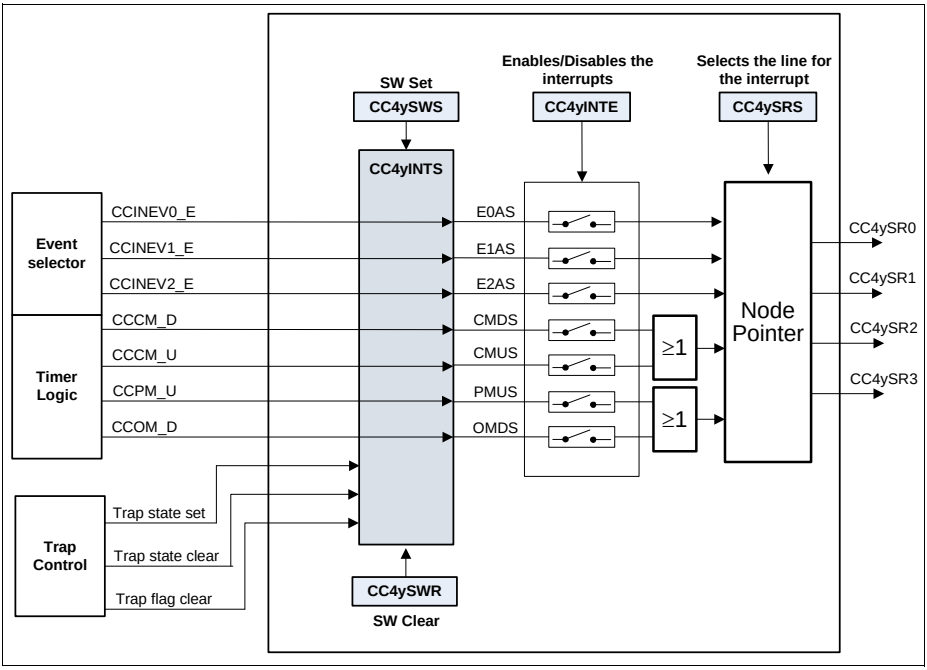


Figure 18-67 Slice interrupt structure overview

Each of the interrupt events can then be forwarded to one of the slice's four service request lines, **Figure 18-68**. The value set on the **CC4ySRS** controls which interrupt event is mapped into which service request line.

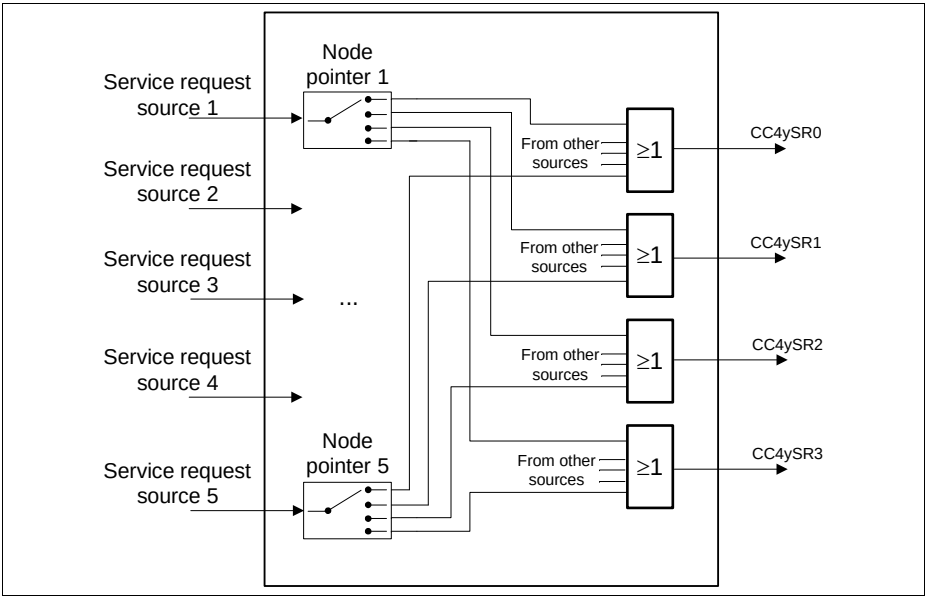


Figure 18-68 Slice Interrupt Node Pointer overview

The four service request lines of each slice are OR together inside the kernel of the CCU4, see **Figure 18-69**. This means that there are only four service request lines per CCU4, that can have in each line interrupt requests coming from different slices.

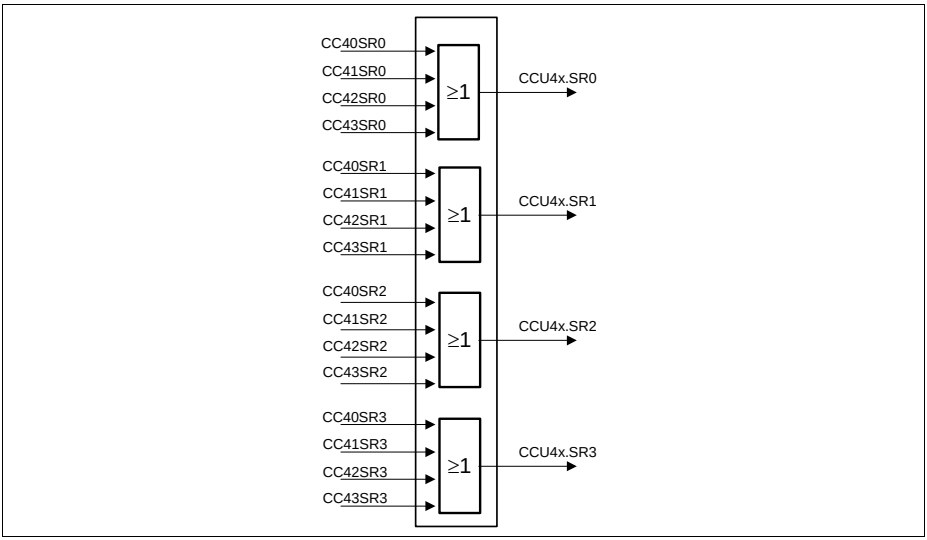


Figure 18-69 CCU4 service request overview

18.4 Debug Behavior

In suspend mode, the functional clocks for all slices as well the prescaler are stopped. The registers can still be accessed by the CPU (read only). This mode is useful for debugging purposes, e.g. where the current device status should be frozen in order to get a snapshot of the internal values. In suspend mode, all the slice timers are stopped. The suspend mode is non-intrusive concerning the register bits. This means register bits are not modified by hardware when entering or leaving the suspend mode.

Entry into suspend mode can be configured at the kernel level by means of the field **GCTRL**.SUSCFG.

The module is only functional after the suspend signal becomes inactive.

18.5 Power, Reset and Clock

The following sections describe the operating conditions, characteristics and timing requirements for the CCU4. All the timing information is related to the module clock, f_{ccu4}.

18.5.1 Clocks

Module Clock

The module clock of the CCU4 module is described in the SCU chapter as f_{CCU}.

The bus interface clock of the CCU4 module is described in the SCU chapter as f_{PERIPH}.

The module clock for the CCU4 is controlled via a specific control bit inside the SCU (System Control Unit), register CLKSET.

It is possible to disable the module clock for the CCU4 via the **GSTAT** register, nevertheless, there may be a dependency of the f_{ccu4} through the different CCU4 instances. One should address the SCU Chapter for a complete description of the product clock scheme.

If module clock dependencies exist through different IP instances, then one can disable the module clock internally inside the specific CCU4, by disabling the prescaler (**GSTAT**.PRB = 0_B).

External Clock

It is possible to use an external clock as source for the prescaler, and consequently for all the timer Slices, CC4y. This external source can be connected to one of the CCU4x.CLK[C...A] inputs.

This external source is nevertheless synchronized against f_{ccu4}.

Table 18-10 External clock operating conditions

Parameter	Symbol	Values			Unit	Note / Test Condition
		Min.	Typ.	Max.		
Frequency	f_{eclk}	–	–	$f_{ccu4}/4$	MHz	
ON time	ton_{eclk}	$2T_{ccu4}$ 1)2)	–	–	ns	
OFF time	$toff_{eclk}$	$2T_{ccu4}$ 1)2)	–	–	ns	Only the rising edge is used

1) Only valid if the signal was not previously synchronized/generated with the fccu4 clock (or a synchronous clock)

2) 50% duty cycle is not obligatory

18.5.2 Module Reset

Each CCU4 has one reset source. This reset source is handled at system level and it can be generated independently via a system control register, PRSET0/PRSET1 (address SCU chapter for a full description).

After reset release, the complete IP is set to default configuration. The default configuration for each register field is addressed on **Section 18.7**.

18.5.3 Power

The CCU4 is inside the power core domain, therefore no special considerations about power up or power down sequences need to be taken. For an explanation about the different power domains, please address the SCU (System Control Unit) chapter.

An internal power down mode for the CCU4, can be achieved by disabling the clock inside the CCU4 itself. For this one should set the **GSTAT** register with the default reset value (via the idle mode set register, **GIDLS**).

18.6 Initialization and System Dependencies

18.6.1 Initialization Sequence

The initialization sequence for an application that is using the CCU4, should be the following:

1st Step: Apply reset to the CCU4, via the specific SCU bitfield on the PRSET0/PRSET1 register.

2nd Step: Release reset of the CCU4, via the specific SCU bitfield on the PRCLR0/PRCLR1 register

3rd Step: Enable the CCU4 clock via the specific SCU register, CLKSET.

4th Step: Enable the prescaler block, by writing 1_B to the **GIDLC**.SPRB field.

5th Step: Configure the global CCU4 register **GCTRL**

6th Step: Configure all the registers related to the required Timer Slice(s) functions, including the interrupt/service request configuration.

7th Step: If needed, configure the startup value for a specific Compare Channel Status, of a Timer Slice, by writing 1_B to the specific **GCSS**.SyTS.

8th Step: Enable the specific timer slice(s), CC4y, by writing 1_B to the specific **GIDLC**.CSyI.

9th Step: For all the Timer Slices that should be started synchronously via SW, the specific system register localized in the SCU, CCUCON, that enables a synchronous timer start should be addressed. The SCU.GSC4x input signal needs to be configured previously as a start function, see **Section 18.2.7.1**.

18.6.2 System Dependencies

Each CCU4 may have different dependencies regarding module and bus clock frequencies. This dependencies should be addressed in the SCU and System Architecture Chapters.

Dependencies between several peripherals, regarding different clock operating frequencies may also exist. This should be addressed before configuring the connectivity between the CCU4 and some other peripheral.

The following topics must be taken into consideration for good CCU4 and system operation:

- CCU4 module clock must be at maximum two times faster than the module bus interface clock
- Module input triggers for the CCU4 must not exceed the module clock frequency (if the triggers are generated internally in the device)
- Module input triggers for the CCU4 must not exceed the frequency dictated in **Section 18.5.1**
- Frequency of the CCU4 outputs used as triggers/functions on other modules, must be crosschecked on the end point
- Applying and removing CCU4 from reset, can cause unwanted operations in other modules. This can occur if the modules are using CCU4 outputs as triggers/functions.

18.7 Registers

Registers Overview

The absolute register address is calculated by adding:

Module Base Address + Offset Address

Table 18-11 Registers Address Space

Module	Base Address	End Address	Note
CCU40	$4000C000_H$	$4000FFFF_H$	
CCU41	40010000_H	$40013FFF_H$	

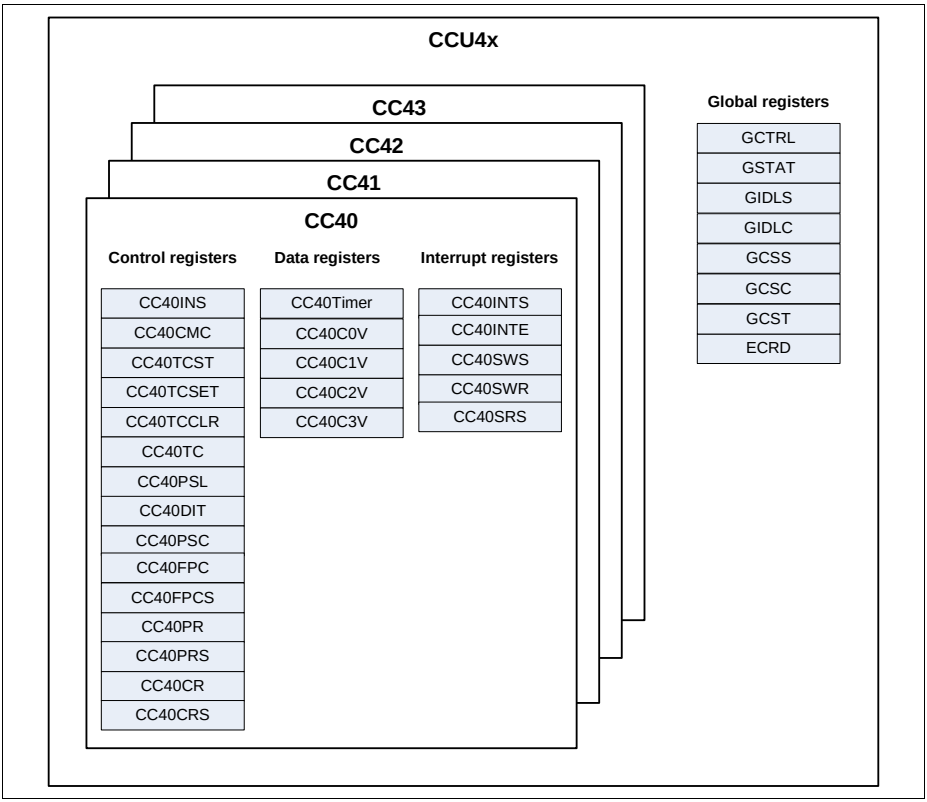


Figure 18-70 CCU4 registers overview

Table 18-12 Register Overview of CCU4

Short Name	Description	Offset Addr.[1)]	Access Mode		Description See
			Read	Write	
CCU4 Global Registers					
GCTRL	Module General Control Register	0000_H	U, PV	U, PV	Page 18-81
GSTAT	General Slice Status Register	0004_H	U, PV	BE	Page 18-84
GIDLS	General Idle Enable Register	0008_H	U, PV	U, PV	Page 18-85
GIDLC	General Idle Disable Register	$000C_H$	U, PV	U, PV	Page 18-87
GCSS	General Channel Set Register	0010_H	U, PV	U, PV	Page 18-88
GCSC	General Channel Clear Register	0014_H	U, PV	U, PV	Page 18-90
GCST	General Channel Status Register	0018_H	U, PV	BE	Page 18-93
ECRD	Extended Read Register	0050_H	U, PV	BE	Page 18-96
MIDR	Module Identification Register	0080_H	U, PV	BE	Page 18-97
CC40 Registers					
CC40INS	Input Selector Unit Configuration	0100_H	U, PV	U, PV	Page 18-98
CC40CMC	Connection Matrix Configuration	0104_H	U, PV	U, PV	Page 18-100
CC40TST	Timer Run Status	0108_H	U, PV	BE	Page 18-103
CC40TCSET	Timer Run Set	$010C_H$	U, PV	U,PV	Page 18-104
CC40TCCLR	Timer Run Clear	0110_H	U, PV	U, PV	Page 18-104
CC40TC	General Timer Configuration	0114_H	U, PV	U, PV	Page 18-105
CC40PSL	Output Passive Level Configuration	0118_H	U, PV	U, PV	Page 18-110
CC40DIT	Dither Configuration	$011C_H$	U, PV	BE	Page 18-111
CC40DITS	Dither Shadow Register	0120_H	U, PV	U, PV	Page 18-112
CC40PSC	Prescaler Configuration	0124_H	U, PV	U, PV	Page 18-112
CC40FPC	Prescaler Compare Value	0128_H	U, PV	U, PV	Page 18-113

Table 18-12 Register Overview of CCU4 (cont'd)

Short Name	Description	Offset Addr.[1)]	Access Mode		Description See
			Read	Write	
CC40FPCS	Prescaler Shadow Compare Value	$012C_H$	U, PV	U, PV	Page 18-114
CC40PR	Timer Period Value	0130_H	U, PV	BE	Page 18-115
CC40PRS	Timer Period Shadow Value	0134_H	U, PV	U, PV	Page 18-115
CC40CR	Timer Compare Value	0138_H	U, PV	BE	Page 18-116
CC40CRS	Timer Compare Shadow Value	$013C_H$	U, PV	U, PV	Page 18-117
CC40TIMER	Timer Current Value	0170_H	U, PV	U, PV	Page 18-118
CC40C0V	Capture Register 0 Value	0174_H	U, PV	BE	Page 18-118
CC40C1V	Capture Register 1Value	0178_H	U, PV	BE	Page 18-119
CC40C2V	Capture Register 2 Value	$017C_H$	U, PV	BE	Page 18-120
CC40C3V	Capture Register 3 Value	0180_H	U, PV	BE	Page 18-121
CC40INTS	Interrupt Status	$01A0_H$	U, PV	BE	Page 18-122
CC40INTE	Interrupt Enable	$01A4_H$	U, PV	U, PV	Page 18-124
CC40SRS	Interrupt Configuration	$01A8_H$	U, PV	U, PV	Page 18-126
CC40SWS	Interrupt Status Set	$01AC_H$	U, PV	U, PV	Page 18-127
CC40SWR	Interrupt Status Clear	$01B0_H$	U, PV	U, PV	Page 18-129
CC41 Registers					
CC41INS	Input Selector Unit Configuration	0200_H	U, PV	U, PV	Page 18-98
CC41CMC	Connection Matrix Configuration	0204_H	U, PV	U, PV	Page 18-100
CC41TST	Timer Run Status	0208_H	U, PV	BE	Page 18-103
CC41TCSET	Timer Run Set	$020C_H$	U, PV	U,PV	Page 18-104
CC41TCCLR	Timer Run Clear	0210_H	U, PV	U, PV	Page 18-104
CC41TC	General Timer Configuration	0214_H	U, PV	U, PV	Page 18-105
CC41PSL	Output Passive Level Configuration	0218_H	U, PV	U, PV	Page 18-110
CC41DIT	Dither Configuration	$021C_H$	U, PV	BE	Page 18-111
CC41DITS	Dither Shadow Register	0220_H	U, PV	U, PV	Page 18-112

Table 18-12 Register Overview of CCU4 (cont'd)

Short Name	Description	Offset Addr.[1)]	Access Mode		Description See
			Read	Write	
CC41PSC	Prescaler Configuration	0224_H	U, PV	U, PV	Page 18-112
CC41FPC	Prescaler Compare Value	0228_H	U, PV	U, PV	Page 18-113
CC41FPCS	Prescaler Shadow Compare Value	$022C_H$	U, PV	U, PV	Page 18-114
CC41PR	Timer Period Value	0230_H	U, PV	BE	Page 18-115
CC41PRS	Timer Period Shadow Value	0234_H	U, PV	U, PV	Page 18-115
CC41CR	Timer Compare Value	0238_H	U, PV	BE	Page 18-116
CC41CRS	Timer Compare Shadow Value	$023C_H$	U, PV	U, PV	Page 18-117
CC41TIMER	Timer Current Value	0270_H	U, PV	U, PV	Page 18-118
CC41C0V	Capture Register 0 Value	0274_H	U, PV	BE	Page 18-118
CC41C1V	Capture Register 1Value	0278_H	U, PV	BE	Page 18-119
CC41C2V	Capture Register 2 Value	$027C_H$	U, PV	BE	Page 18-120
CC41C3V	Capture Register 3 Value	0280_H	U, PV	BE	Page 18-121
CC41INTS	Interrupt Status	$02A0_H$	U, PV	BE	Page 18-122
CC41INTE	Interrupt Enable	$02A4_H$	U, PV	U, PV	Page 18-124
CC41SRS	Interrupt Configuration	$02A8_H$	U, PV	U, PV	Page 18-126
CC41SWS	Interrupt Status Set	$02AC_H$	U, PV	U, PV	Page 18-127
CC41SWR	Interrupt Status Clear	$02B0_H$	U, PV	U, PV	Page 18-129
CC42 Registers					
CC42INS	Input Selector Unit Configuration	0300_H	U, PV	U, PV	Page 18-98
CC42CMC	Connection Matrix Configuration	0304_H	U, PV	U, PV	Page 18-100
CC42TST	Timer Run Status	0308_H	U, PV	BE	Page 18-103
CC42TCSET	Timer Run Set	$030C_H$	U, PV	U,PV	Page 18-104
CC42TCCLR	Timer Run Clear	0310_H	U, PV	U, PV	Page 18-104
CC42TC	General Timer Configuration	0314_H	U, PV	U, PV	Page 18-105
CC42PSL	Output Passive Level Configuration	0318_H	U, PV	U, PV	Page 18-110

Table 18-12 Register Overview of CCU4 (cont'd)

Short Name	Description	Offset Addr.[1)]	Access Mode		Description See
			Read	Write	
CC42DIT	Dither Configuration	$031C_H$	U, PV	BE	Page 18-111
CC42DITS	Dither Shadow Register	0320_H	U, PV	U, PV	Page 18-112
CC42PSC	Prescaler Configuration	0324_H	U, PV	U, PV	Page 18-112
CC42FPC	Prescaler Compare Value	0328_H	U, PV	U, PV	Page 18-113
CC42FPCS	Prescaler Shadow Compare Value	$032C_H$	U, PV	U, PV	Page 18-114
CC42PR	Timer Period Value	0330_H	U, PV	BE	Page 18-115
CC42PRS	Timer Period Shadow Value	0334_H	U, PV	U, PV	Page 18-115
CC42CR	Timer Compare Value	0338_H	U, PV	BE	Page 18-116
CC42CRS	Timer Compare Shadow Value	$033C_H$	U, PV	U, PV	Page 18-117
CC42TIMER	Timer Current Value	0370_H	U, PV	U, PV	Page 18-118
CC42C0V	Capture Register 0 Value	0374_H	U, PV	BE	Page 18-118
CC42C1V	Capture Register 1Value	0378_H	U, PV	BE	Page 18-119
CC42C2V	Capture Register 2 Value	$037C_H$	U, PV	BE	Page 18-120
CC42C3V	Capture Register 3 Value	0380_H	U, PV	BE	Page 18-121
CC42INTS	Interrupt Status	$03A0_H$	U, PV	BE	Page 18-122
CC42INTE	Interrupt Enable	$03A4_H$	U, PV	U, PV	Page 18-124
CC42SRS	Interrupt Configuration	$03A8_H$	U, PV	U, PV	Page 18-126
CC42SWS	Interrupt Status Set	$03AC_H$	U, PV	U, PV	Page 18-127
CC42SWR	Interrupt Status Clear	$03B0_H$	U, PV	U, PV	Page 18-129
CC43 Registers					
CC43INS	Input Selector Unit Configuration	0400_H	U, PV	U, PV	Page 18-98
CC43CMC	Connection Matrix Configuration	0404_H	U, PV	U, PV	Page 18-100
CC43TST	Timer Run Status	0408_H	U, PV	BE	Page 18-103
CC43TCSET	Timer Run Set	$040C_H$	U, PV	U,PV	Page 18-104
CC43TCCLR	Timer Run Clear	0410_H	U, PV	U, PV	Page 18-104
CC43TC	General Timer Configuration	0414_H	U, PV	U, PV	Page 18-105

Table 18-12 Register Overview of CCU4 (cont'd)

Short Name	Description	Offset Addr.[1)]	Access Mode		Description See
			Read	Write	
CC43PSL	Output Passive Level Configuration	0418_H	U, PV	U, PV	Page 18-110
CC43DIT	Dither Configuration	$041C_H$	U, PV	BE	Page 18-111
CC43DITS	Dither Shadow Register	0420_H	U, PV	U, PV	Page 18-112
CC43PSC	Prescaler Configuration	0424_H	U, PV	U, PV	Page 18-112
CC43FPC	Prescaler Compare Value	0428_H	U, PV	U, PV	Page 18-113
CC43FPCS	Prescaler Shadow Compare Value	$042C_H$	U, PV	U, PV	Page 18-114
CC43PR	Timer Period Value	0430_H	U, PV	BE	Page 18-115
CC43PRS	Timer Period Shadow Value	0434_H	U, PV	U, PV	Page 18-115
CC43CR	Timer Compare Value	0438_H	U, PV	BE	Page 18-116
CC43CRS	Timer Compare Shadow Value	$043C_H$	U, PV	U, PV	Page 18-117
CC43TIMER	Timer Current Value	0470_H	U, PV	U, PV	Page 18-118
CC43C0V	Capture Register 0 Value	0474_H	U, PV	BE	Page 18-118
CC43C1V	Capture Register 1Value	0478_H	U, PV	BE	Page 18-119
CC43C2V	Capture Register 2 Value	$047C_H$	U, PV	BE	Page 18-120
CC43C3V	Capture Register 3 Value	0480_H	U, PV	BE	Page 18-121
CC43INTS	Interrupt Status	$04A0_H$	U, PV	BE	Page 18-122
CC43INTE	Interrupt Enable	$04A4_H$	U, PV	U, PV	Page 18-124
CC43SRS	Interrupt Configuration	$04A8_H$	U, PV	U, PV	Page 18-126
CC43SWS	Interrupt Status Set	$04AC_H$	U, PV	U, PV	Page 18-127
CC43SWR	Interrupt Status Clear	$04B0_H$	U, PV	U, PV	Page 18-129

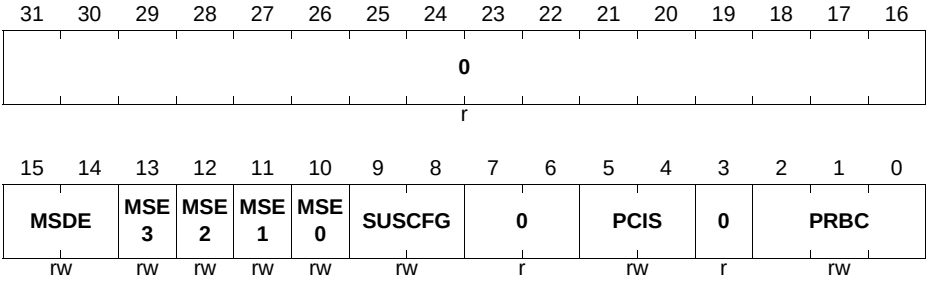
1) The absolute register address is calculated as follows:
Module Base Address + Offset Address (shown in this column)

18.7.1 Global Registers

GCTRL

The register contains the global configuration fields that affect all the timer slices inside CCU4.

GCTRL
Global Control Register **(0000_H)** **Reset Value: 00000000_H**



Field	Bits	Type	Description
PRBC	[2:0]	rw	**Prescaler Clear Configuration** This register controls how the prescaler Run Bit and internal registers are cleared. 000_B SW only 001_B **GSTAT**.PRB and prescaler registers are cleared when the Run Bit of CC40 is cleared. 010_B **GSTAT**.PRB and prescaler registers are cleared when the Run Bit of CC41 is cleared. 011_B **GSTAT**.PRB and prescaler registers are cleared when the Run Bit of CC42 is cleared. 100_B **GSTAT**.PRB and prescaler registers are cleared when the Run Bit of CC43 is cleared.
PCIS	[5:4]	rw	**Prescaler Input Clock Selection** 00_B Module clock 01_B CCU4x.ECLKA 10_B CCU4x.ECLKB 11_B CCU4x.ECLKC

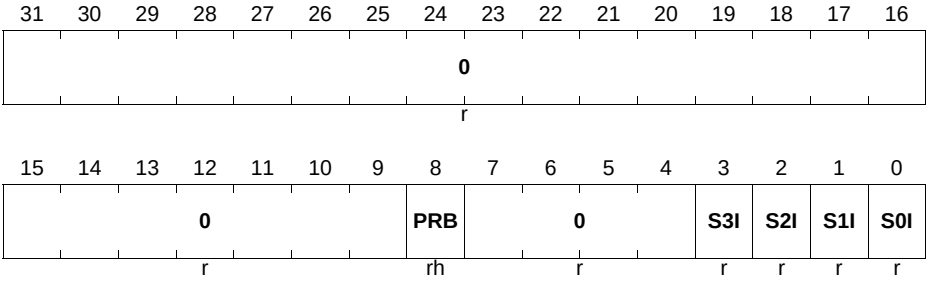
Field	Bits	Type	Description
SUSCFG	[9:8]	rw	**Suspend Mode Configuration** This field controls the entering in suspend mode for all the CAPCOM4 slices. 00_B Suspend request ignored. The module never enters in suspend 01_B Stops all the running slices immediately. Safe stop is not applied. 10_B Stops the block immediately and clamps all the outputs to PASSIVE state. Safe stop is applied. 11_B Waits for the roll over of each slice to stop and clamp the slices outputs. Safe stop is applied.
MSE0	10	rw	**Slice 0 Multi Channel shadow transfer enable** When this field is set, a shadow transfer of slice 0 can be requested not only by SW but also via the CCU4x.MCSS input. 0_B Shadow transfer can only be requested by SW 1_B Shadow transfer can be requested via SW and via the CCU4x.MCSS input.
MSE1	11	rw	**Slice 1 Multi Channel shadow transfer enable** When this field is set, a shadow transfer of slice 1 can be requested not only by SW but also via the CCU4x.MCSS input. 0_B Shadow transfer can only be requested by SW 1_B Shadow transfer can be requested via SW and via the CCU4x.MCSS input.
MSE2	12	rw	**Slice 2 Multi Channel shadow transfer enable** When this field is set, a shadow transfer of slice 2 can be requested not only by SW but also via the CCU4x.MCSS input. 0_B Shadow transfer can only be requested by SW 1_B Shadow transfer can be requested via SW and via the CCU4x.MCSS input.

Field	Bits	Type	Description
MSE3	13	rw	**Slice 3 Multi Channel shadow transfer enable** When this field is set, a shadow transfer of slice 3 can be requested not only by SW but also via the CCU4x.MCSS input. 0_B Shadow transfer can only be requested by SW 1_B Shadow transfer can be requested via SW and via the CCU4x.MCSS input.
MSDE	[15:14]	rw	**Multi Channel shadow transfer request configuration** This field configures the type of shadow transfer requested via the CCU4x.MCSS input. The field **CC4yTC**.MSEy needs to be set in order for this configuration to have any effect. 00_B Only the shadow transfer for period and compare values is requested 01_B Shadow transfer for the compare, period and prescaler compare values is requested 10_B Reserved 11_B Shadow transfer for the compare, period, prescaler and dither compare values is requested
0	3, [7:6], [31:16]	r	**Reserved** A read always returns 0.

GSTAT

The register contains the status of the prescaler and each timer slice (idle mode or running).

GSTAT
Global Status Register **(0004_H)** **Reset Value: $0000000F_H$**



Field	Bits	Type	Description
S0I	0	r	**CC40 IDLE status** This bit indicates if the CC40 slice is in IDLE mode or not. In IDLE mode the clocks for the CC40 slice are stopped. 0_B Running 1_B Idle
S1I	1	r	**CC41 IDLE status** This bit indicates if the CC41 slice is in IDLE mode or not. In IDLE mode the clocks for the CC41 slice are stopped. 0_B Running 1_B Idle
S2I	2	r	**CC42 IDLE status** This bit indicates if the CC42 slice is in IDLE mode or not. In IDLE mode the clocks for the CC42 slice are stopped. 0_B Running 1_B Idle
S3I	3	r	**CC43 IDLE status** This bit indicates if the CC43 slice is in IDLE mode or not. In IDLE mode the clocks for the CC43 slice are stopped. 0_B Running 1_B Idle

Field	Bits	Type	Description
PRB	8	rh	**Prescaler Run Bit** 0_B Prescaler is stopped 1_B Prescaler is running
0	[7:4], [31:9]	r	**Reserved** Read always returns 0.

GIDLS

Through this register one can set the prescaler and the specific timer slices into idle mode.

GIDLS
Global Idle Set **(0008_H)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0						PSIC	CPRB	0				SS3I	SS2I	SS1I	SS0I
r						w	w	r				w	w	w	w

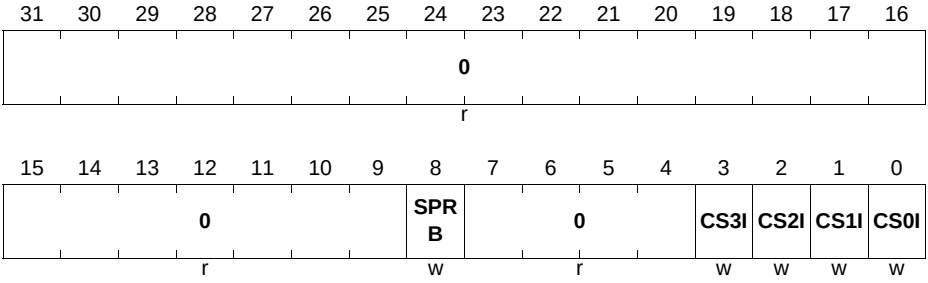
Field	Bits	Type	Description
SS0I	0	w	**CC40 IDLE mode set** Writing a 1_B to this bit sets the CC40 slice in IDLE mode. The clocks for the slice are stopped when in IDLE mode. When entering IDLE, the internal slice registers are not cleared. A read access always returns 0.
SS1I	1	w	**CC41 IDLE mode set** Writing a 1_B to this bit sets the CC41 slice in IDLE mode. The clocks for the slice are stopped when in IDLE mode. When entering IDLE, the internal slice registers are not cleared. A read access always returns 0.

Field	Bits	Type	Description
SS2I	2	w	**CC42 IDLE mode set** Writing a 1_B to this bit sets the CC42 slice in IDLE mode. The clocks for the slice are stopped when in IDLE mode. When entering IDLE, the internal slice registers are not cleared. A read access always returns 0.
SS3I	3	w	**CC43 IDLE mode set** Writing a 1_B to this bit sets the CC43 slice in IDLE mode. The clocks for the slice are stopped when in IDLE mode. When entering IDLE, the internal slice registers are not cleared. A read access always returns 0.
CPRB	8	w	**Prescaler Run Bit Clear** Writing a 1_B into this register clears the Run Bit of the prescaler. Prescaler internal registers are not cleared. A read always returns 0.
PSIC	9	w	**Prescaler clear** Writing a 1_B to this register clears the prescaler counter. It also loads the PSIV into the PVAL field for all Timer Slices. This performs a re alignment of the timer clock for all Slices. The Run Bit of the prescaler is not cleared. A read always returns 0.
0	[7:4], [31:10]	r	**Reserved** Read always returns 0.

GIDLC

Through this register one can remove the prescaler and the specific timer slices from idle mode.

GIDLC
Global Idle Clear **($000C_H$)** **Reset Value: 00000000_H**



Field	Bits	Type	Description
CS0I	0	w	**CC40 IDLE mode clear** Writing a 1_B to this bit removes the CC40 from IDLE mode. A read access always returns 0.
CS1I	1	w	**CC41 IDLE mode clear** Writing a 1_B to this bit removes the CC41 from IDLE mode. A read access always returns 0.
CS2I	2	w	**CC42 IDLE mode clear** Writing a 1_B to this bit removes the CC42 from IDLE mode. A read access always returns 0.
CS3I	3	w	**CC43 IDLE mode clear** Writing a 1_B to this bit removes the CC43 from IDLE mode. A read access always returns 0.
SPRB	8	w	**Prescaler Run Bit Set** Writing a 1_B into this register sets the Run Bit of the prescaler. A read always returns 0.
0	[7:4], [31:9]	r	**Reserved** Read always returns 0.

GCSS

Through this register one can request a shadow transfer for the specific timer slice(s) and set the status bit for each of the compare channels.

GCSS
Global Channel Set **(0010_H)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0												S3STS	S2STS	S1STS	S0STS
r												w	w	w	w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	S3PSE	S3DSE	S3SE	0	S2PSE	S2DSE	S2SE	0	S1PSE	S1DSE	S1SE	0	S0PSE	S0DSE	S0SE
r	w	w	w	r	w	w	w	r	w	w	w	r	w	w	w

Field	Bits	Type	Description
S0SE	0	w	**Slice 0 shadow transfer set enable** Writing a 1_B to this bit will set the **GCST**.S0SS field, enabling then a shadow transfer for the Period, Compare and Passive level values. A read always returns 0.
S0DSE	1	w	**Slice 0 Dither shadow transfer set enable** Writing a 1_B to this bit will set the **GCST**.S0DSS field, enabling then a shadow transfer for the Dither compare value. A read always returns 0.
S0PSE	2	w	**Slice 0 Prescaler shadow transfer set enable** Writing a 1_B to this bit will set the **GCST**.S0PSS field, enabling then a shadow transfer for the prescaler compare value. A read always returns 0.
S1SE	4	w	**Slice 1 shadow transfer set enable** Writing a 1_B to this bit will set the **GCST**.S1SS field, enabling then a shadow transfer for the Period, Compare and Passive level values. A read always returns 0.
S1DSE	5	w	**Slice 1 Dither shadow transfer set enable** Writing a 1_B to this bit will set the **GCST**.S1DSS field, enabling then a shadow transfer for the Dither compare value. A read always returns 0.

Field	Bits	Type	Description
S1PSE	6	w	**Slice 1 Prescaler shadow transfer set enable** Writing a 1_B to this bit will set the **GCST**.S1PSS field, enabling then a shadow transfer for the prescaler compare value. A read always returns 0.
S2SE	8	w	**Slice 2 shadow transfer set enable** Writing a 1_B to this bit will set the **GCST**.S2SS field, enabling then a shadow transfer for the Period, Compare and Passive level values. A read always returns 0.
S2DSE	9	w	**Slice 2 Dither shadow transfer set enable** Writing a 1_B to this bit will set the **GCST**.S2DSS field, enabling then a shadow transfer for the Dither compare value. A read always returns 0.
S2PSE	10	w	**Slice 2 Prescaler shadow transfer set enable** Writing a 1_B to this bit will set the **GCST**.S2PSS field, enabling then a shadow transfer for the prescaler compare value. A read always returns 0.
S3SE	12	w	**Slice 3 shadow transfer set enable** Writing a 1_B to this bit will set the **GCST**.S3SS field, enabling then a shadow transfer for the Period, Compare and Passive level values. A read always returns 0.
S3DSE	13	w	**Slice 3 Dither shadow transfer set enable** Writing a 1_B to this bit will set the **GCST**.S3DSS field, enabling then a shadow transfer for the Dither compare value. A read always returns 0.
S3PSE	14	w	**Slice 3 Prescaler shadow transfer set enable** Writing a 1_B to this bit will set the **GCST**.S3PSS field, enabling then a shadow transfer for the prescaler compare value. A read always returns 0.
S0STS	16	w	**Slice 0 status bit set** Writing a 1_B into this field sets the status bit of slice 0 (**GCST**.CC40ST) to 1_B. A read always returns 0.

Field	Bits	Type	Description
S1STS	17	w	**Slice 1 status bit set** Writing a 1_B into this field sets the status bit of slice 1 (**GCST**.CC41ST) to 1_B. A read always returns 0.
S2STS	18	w	**Slice 2 status bit set** Writing a 1_B into this field sets the status bit of slice 2 (**GCST**.CC42ST) to 1_B. A read always returns 0.
S3STS	19	w	**Slice 3 status bit set** Writing a 1_B into this field sets the status bit of slice 3 (**GCST**.CC43ST) to 1_B. A read always returns 0.
0	3, 7, 11, 15, [31:20]	r	**Reserved** Read always returns 0.

GCSC

Through this register one can reset a shadow transfer request for the specific timer slice and clear the status bit for each the compare channels.

GCSC
Global Channel Clear **(0014_H)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0												S3STC	S2STC	S1STC	S0STC
r												w	w	w	w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	S3PSC	S3DSC	S3SC	0	S2PSC	S2DSC	S2SC	0	S1PSC	S1DSC	S1SC	0	S0PSC	S0DSC	S0SC
r	w	w	w	r	w	w	w	r	w	w	w	r	w	w	w

Field	Bits	Type	Description
S0SC	0	w	**Slice 0 shadow transfer clear** Writing a 1_B to this bit will clear the **GCST**.S0SS field, canceling any pending shadow transfer for the Period, Compare and Passive level values. A read always returns 0.
S0DSC	1	w	**Slice 0 Dither shadow transfer clear** Writing a 1_B to this bit will clear the **GCST**.S0DSS field, canceling any pending shadow transfer for the Dither compare value. A read always returns 0.
S0PSC	2	w	**Slice 0 Prescaler shadow transfer clear** Writing a 1_B to this bit will clear the **GCST**.S0PSS field, canceling any pending shadow transfer for the prescaler compare value. A read always returns 0.
S1SC	4	w	**Slice 1 shadow transfer clear** Writing a 1_B to this bit will clear the **GCST**.S1SS field, canceling any pending shadow transfer for the Period, Compare and Passive level values. A read always returns 0.
S1DSC	5	w	**Slice 1 Dither shadow transfer clear** Writing a 1_B to this bit will clear the **GCST**.S1DSS field, canceling any pending shadow transfer for the Dither compare value. A read always returns 0.
S1PSC	6	w	**Slice 1 Prescaler shadow transfer clear** Writing a 1_B to this bit will clear the **GCST**.S1PSS field, canceling any pending shadow transfer for the prescaler compare value. A read always returns 0.
S2SC	8	w	**Slice 2 shadow transfer clear** Writing a 1_B to this bit will clear the **GCST**.S2SS field, canceling any pending shadow transfer for the Period, Compare and Passive level values. A read always returns 0.

Field	Bits	Type	Description
S2DSC	9	w	**Slice 2 Dither shadow transfer clear** Writing a 1_B to this bit will clear the **GCST**.S2DSS field, canceling any pending shadow transfer for the Dither compare value. A read always returns 0.
S2PSC	10	w	**Slice 2 Prescaler shadow transfer clear** Writing a 1_B to this bit will clear the **GCST**.S2PSS field, canceling any pending shadow transfer for the prescaler compare value. A read always returns 0.
S3SC	12	w	**Slice 3 shadow transfer clear** Writing a 1_B to this bit will clear the **GCST**.S3SS field, canceling any pending shadow transfer for the Period, Compare and Passive level values. A read always returns 0.
S3DSC	13	w	**Slice 3 Dither shadow transfer clear** Writing a 1_B to this bit will clear the **GCST**.S3DSS field, canceling any pending shadow transfer for the Dither compare value. A read always returns 0.
S3PSC	14	w	**Slice 3 Prescaler shadow transfer clear** Writing a 1_B to this bit will clear the **GCST**.S3PSS field, canceling any pending shadow transfer for the prescaler compare value. A read always returns 0.
S0STC	16	w	**Slice 0 status bit clear** Writing a 1_B into this field clears the status bit of slice 0 (**GCST**.CC40ST) to 0_B. A read always returns 0.
S1STC	17	w	**Slice 1 status bit clear** Writing a 1_B into this field clears the status bit of slice 1 (**GCST**.CC41ST) to 0_B. A read always returns 0.
S2STC	18	w	**Slice 2 status bit clear** Writing a 1_B into this field clears the status bit of slice 2 (**GCST**.CC42ST) to 0_B. A read always returns 0.

Field	Bits	Type	Description
S3STC	19	w	**Slice 3 status bit clear** Writing a 1_B into this field clears the status bit of slice 3 (**GCST**.CC43ST) to 0_B. A read always returns 0.
0	3, 7, 11, 15, [31:20]	r	**Reserved** Read always returns 0.

GCST

This register holds the information of the shadow transfer requests and of each timer slice status bit.

GCST
Global Channel Status **(0018_H)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0												CC43ST	CC42ST	CC41ST	CC40ST
r												rh	rh	rh	rh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	S3PSS	S3DSS	S3SS	0	S2PSS	S2DSS	S2SS	0	S1PSS	S1DSS	S1SS	0	S0PSS	S0DSS	S0SS
r	rh	rh	rh	r	rh	rh	rh	r	rh	rh	rh	r	rh	rh	rh

Field	Bits	Type	Description
S0SS	0	rh	**Slice 0 shadow transfer status** 0_B Shadow transfer has not been requested 1_B Shadow transfer has been requested This field is cleared by HW after the requested shadow transfer has been executed.
S0DSS	1	rh	**Slice 0 Dither shadow transfer status** 0_B Dither shadow transfer has not been requested 1_B Dither shadow transfer has been requested This field is cleared by HW after the requested shadow transfer has been executed.

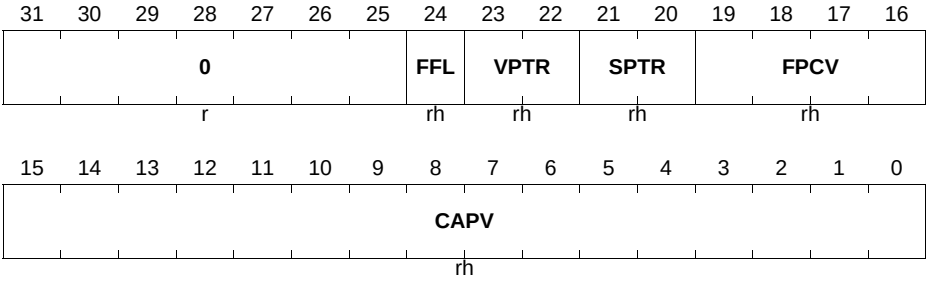
Field	Bits	Type	Description
S0PSS	2	rh	**Slice 0 Prescaler shadow transfer status** 0_B Prescaler shadow transfer has not been requested 1_B Prescaler shadow transfer has been requested This field is cleared by HW after the requested shadow transfer has been executed.
S1SS	4	rh	**Slice 1 shadow transfer status** 0_B Shadow transfer has not been requested 1_B Shadow transfer has been requested This field is cleared by HW after the requested shadow transfer has been executed.
S1DSS	5	rh	**Slice 1 Dither shadow transfer status** 0_B Dither shadow transfer has not been requested 1_B Dither shadow transfer has been requested This field is cleared by HW after the requested shadow transfer has been executed.
S1PSS	6	rh	**Slice 1 Prescaler shadow transfer status** 0_B Prescaler shadow transfer has not been requested 1_B Prescaler shadow transfer has been requested This field is cleared by HW after the requested shadow transfer has been executed.
S2SS	8	rh	**Slice 2 shadow transfer status** 0_B Shadow transfer has not been requested 1_B Shadow transfer has been requested This field is cleared by HW after the requested shadow transfer has been executed.
S2DSS	9	rh	**Slice 2 Dither shadow transfer status** 0_B Dither shadow transfer has not been requested 1_B Dither shadow transfer has been requested This field is cleared by HW after the requested shadow transfer has been executed.

Field	Bits	Type	Description
S2PSS	10	rh	**Slice 2 Prescaler shadow transfer status** 0_B Prescaler shadow transfer has not been requested 1_B Prescaler shadow transfer has been requested This field is cleared by HW after the requested shadow transfer has been executed.
S3SS	12	rh	**Slice 3 shadow transfer status** 0_B Shadow transfer has not been requested 1_B Shadow transfer has been requested This field is cleared by HW after the requested shadow transfer has been executed.
S3DSS	13	rh	**Slice 3 Dither shadow transfer status** 0_B Dither shadow transfer has not been requested 1_B Dither shadow transfer has been requested This field is cleared by HW after the requested shadow transfer has been executed.
S3PSS	14	rh	**Slice 3 Prescaler shadow transfer status** 0_B Prescaler shadow transfer has not been requested 1_B Prescaler shadow transfer has been requested This field is cleared by HW after the requested shadow transfer has been executed.
CC40ST	16	rh	**Slice 0 status bit**
CC41ST	17	rh	**Slice 1 status bit**
CC42ST	18	rh	**Slice 2 status bit**
CC43ST	19	rh	**Slice 3 status bit**
0	3, 7, 11, 15, [31:20]	r	**Reserved** Read always returns 0.

ECRD

This register holds the information related to the extended capture mode.

ECRD
Extended Capture Mode Read **(0050_H)** **Reset Value: 00000000_H**



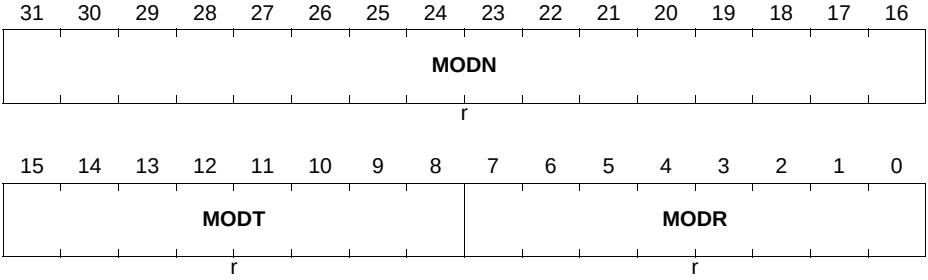
Field	Bits	Type	Description
CAPV	[15:0]	rh	**Timer Capture Value** This field contains the timer captured value
FPCV	[19:16]	rh	**Prescaler Capture value** This field contains the value of the prescaler clock division associated with the specific CAPV field
SPTR	[21:20]	rh	**Slice pointer** This field indicates the slice index in which the value was captured. 00_B CC40 01_B CC41 10_B CC42 11_B CC43
VPTR	[23:22]	rh	**Capture register pointer** This field indicates the capture register index in which the value was captured. 00_B Capture register 0 01_B Capture register 1 10_B Capture register 2 11_B Capture register 3
FFL	24	rh	**Full Flag** This bit indicates if the associated capture register contains a value. 0_B No new value was captured into this register 1_B A new value has been captured into this register

Field	Bits	Type	Description
0	[31:25]	r	**Reserved** Read always returns 0.

MIDR

This register contains the module identification number.

MIDR
Module Identification **(0080_H)** **Reset Value: $00A6C0XX_H$**



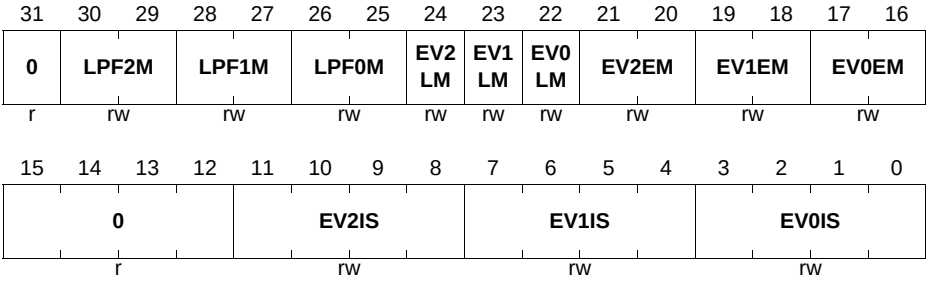
Field	Bits	Type	Description
MODR	[7:0]	r	**Module Revision** This bit field indicates the revision number of the module implementation (depending on the design step). The given value of 00_H is a placeholder for the actual number.
MODT	[15:8]	r	**Module Type**
MODN	[31:16]	r	**Module Number**

18.7.2 Slice (CC4y) Registers

CC4yINS

The register contains the configuration for the input selector.

CC4yINS (y = 0 - 3)
Input Selector Configuration **($0100_H + 0100_H * y$)** **Reset Value: 00000000_H**



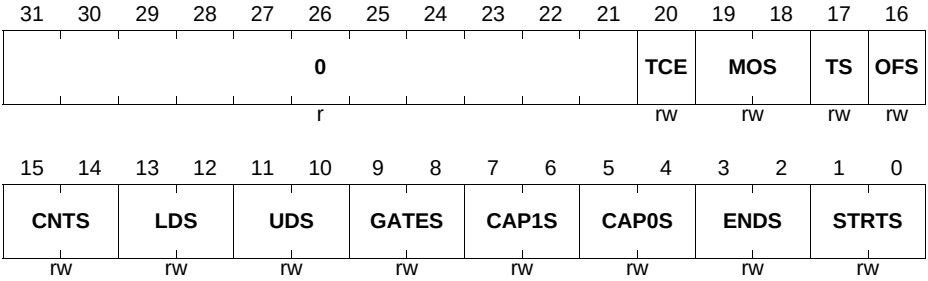
Field	Bits	Type	Description
EV0IS	[3:0]	rw	**Event 0 signal selection** This field selects which pins is used for the event 0. 0000_B CCU4x.INyA 0001_B CCU4x.INyB 0010_B CCU4x.INyC 0011_B CCU4x.INyD 0100_B CCU4x.INyE 0101_B CCU4x.INyF 0110_B CCU4x.INyG 0111_B CCU4x.INyH 1000_B CCU4x.INyI 1001_B CCU4x.INyJ 1010_B CCU4x.INyK 1011_B CCU4x.INyL 1100_B CCU4x.INyM 1101_B CCU4x.INyN 1110_B CCU4x.INyO 1111_B CCU4x.INyP
EV1IS	[7:4]	rw	**Event 1 signal selection** Same as EV0IS description
EV2IS	[11:8]	rw	**Event 2 signal selection** Same as EV0IS description

Field	Bits	Type	Description
EV0EM	[17:16]	rw	**Event 0 Edge Selection** 00_B No action 01_B Signal active on rising edge 10_B Signal active on falling edge 11_B Signal active on both edges
EV1EM	[19:18]	rw	**Event 1 Edge Selection** Same as EV0EM description
EV2EM	[21:20]	rw	**Event 2 Edge Selection** Same as EV0EM description
EV0LM	22	rw	**Event 0 Level Selection** 0_B Active on HIGH level 1_B Active on LOW level
EV1LM	23	rw	**Event 1 Level Selection** Same as EV0LM description
EV2LM	24	rw	**Event 2 Level Selection** Same as EV0LM description
LPF0M	[26:25]	rw	**Event 0 Low Pass Filter Configuration** This field sets the number of consecutive counts for the Low Pass Filter of Event 0. The input signal value needs to remain stable for this number of counts (f_{CCU4}), so that a level/transition is accepted. 00_B LPF is disabled 01_B 3 clock cycles of f_{CCU4} 10_B 5 clock cycles of f_{CCU4} 11_B 7 clock cycles of f_{CCU4}
LPF1M	[28:27]	rw	**Event 1 Low Pass Filter Configuration** Same description as LPF0M
LPF2M	[30:29]	rw	**Event 2 Low Pass Filter Configuration** Same description as LPF0M
0	[15:12], 31	r	**Reserved** Read always returns 0.

CC4yCMC

The register contains the configuration for the connection matrix.

CC4yCMC (y = 0 - 3)
Connection Matrix Control **($0104_H + 0100_H * y$)** **Reset Value: 00000000_H**



Field	Bits	Type	Description
STRTS	[1:0]	rw	**External Start Functionality Selector** Selects the Event that is going to be linked with the external start functionality. 00_B External Start Function deactivated 01_B External Start Function triggered by Event 0 10_B External Start Function triggered by Event 1 11_B External Start Function triggered by Event 2
ENDS	[3:2]	rw	**External Stop Functionality Selector** Selects the Event that is going to be linked with the external stop functionality. 00_B External Stop Function deactivated 01_B External Stop Function triggered by Event 0 10_B External Stop Function triggered by Event 1 11_B External Stop Function triggered by Event 2
CAP0S	[5:4]	rw	**External Capture 0 Functionality Selector** Selects the Event that is going to be linked with the external capture for capture registers number 1 and 0. 00_B External Capture 0 Function deactivated 01_B External Capture 0 Function triggered by Event 0 10_B External Capture 0 Function triggered by Event 1 11_B External Capture 0 Function triggered by Event 2

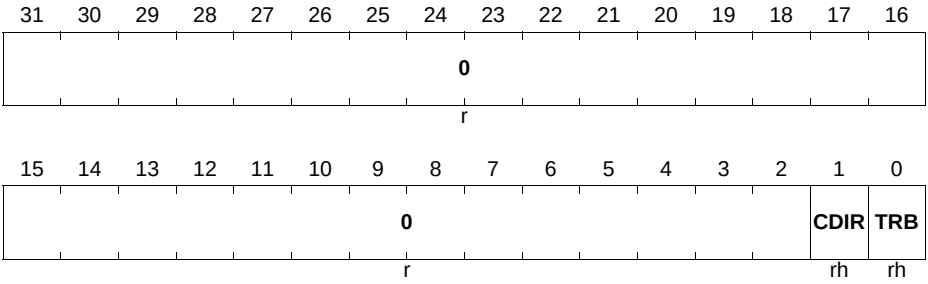
Field	Bits	Type	Description
CAP1S	[7:6]	rw	**External Capture 1 Functionality Selector** Selects the Event that is going to be linked with the external capture for capture registers number 3 and 2. 00_B External Capture 1 Function deactivated 01_B External Capture 1 Function triggered by Event 0 10_B External Capture 1 Function triggered by Event 1 11_B External Capture 1 Function triggered by Event 2
GATES	[9:8]	rw	**External Gate Functionality Selector** Selects the Event that is going to be linked with the counter gating function. This function is used to gate the timer increment/decrement procedure. 00_B External Gating Function deactivated 01_B External Gating Function triggered by Event 0 10_B External Gating Function triggered by Event 1 11_B External Gating Function triggered by Event 2
UDS	[11:10]	rw	**External Up/Down Functionality Selector** Selects the Event that is going to be linked with the Up/Down counting direction control. 00_B External Up/Down Function deactivated 01_B External Up/Down Function triggered by Event 0 10_B External Up/Down Function triggered by Event 1 11_B External Up/Down Function triggered by Event 2
LDS	[13:12]	rw	**External Timer Load Functionality Selector** Selects the Event that is going to be linked with the timer load function. 00_B - External Load Function deactivated 01_B - External Load Function triggered by Event 0 10_B - External Load Function triggered by Event 1 11_B - External Load Function triggered by Event 2

Field	Bits	Type	Description
CNTS	[15:14]	rw	**External Count Selector** Selects the Event that is going to be linked with the count function. The counter is going to be increment/decremented each time that a specific transition on the event is detected. 00_B External Count Function deactivated 01_B External Count Function triggered by Event 0 10_B External Count Function triggered by Event 1 11_B External Count Function triggered by Event 2
OFS	16	rw	**Override Function Selector** This field enables the ST bit override functionality. 0_B Override functionality disabled 1_B Status bit trigger override connected to Event 1; Status bit value override connected to Event 2
TS	17	rw	**Trap Function Selector** This field enables the trap functionality. 0_B Trap function disabled 1_B TRAP function connected to Event 2
MOS	[19:18]	rw	**External Modulation Functionality Selector** Selects the Event that is going to be linked with the external modulation function. 00_B - Modulation Function deactivated 01_B - Modulation Function triggered by Event 0 10_B - Modulation Function triggered by Event 1 11_B - Modulation Function triggered by Event 2
TCE	20	rw	**Timer Concatenation Enable** This bit enables the timer concatenation with the previous slice. 0_B Timer concatenation is disabled 1_B Timer concatenation is enabled *Note: In CC40 this field doesn't exist. This is a read only reserved field. Read access always returns 0.*
0	[31:21]	r	**Reserved** A read always returns 0

CC4yTCST

The register holds the status of the timer (running/stopped) and the information about the counting direction (up/down).

CC4yTCST (y = 0 - 3)
Slice Timer Status **($0108_H + 0100_H * y$)** **Reset Value: 00000000_H**

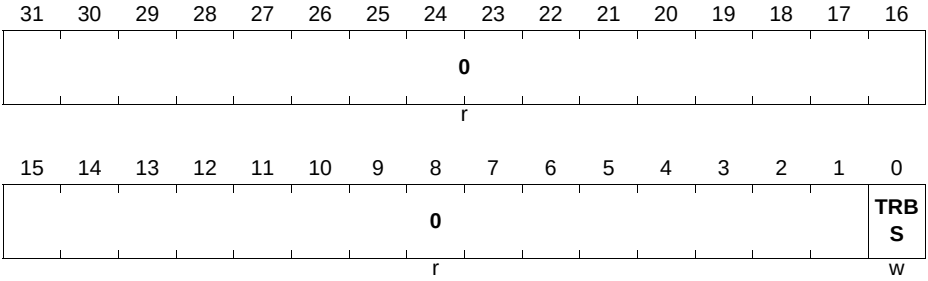


Field	Bits	Type	Description
TRB	0	rh	**Timer Run Bit** This field indicates if the timer is running. 0_B Timer is stopped 1_B Timer is running
CDIR	1	rh	**Timer Counting Direction** This filed indicates if the timer is being increment or decremented 0_B Timer is counting up 1_B Timer is counting down
0	[31:2]	r	**Reserved** Read always returns 0

CC4yTCSET

Through this register it is possible to start the timer.

CC4yTCSET (y = 0 - 3)
Slice Timer Run Set **($010C_H + 0100_H * y$)** **Reset Value: 00000000_H**

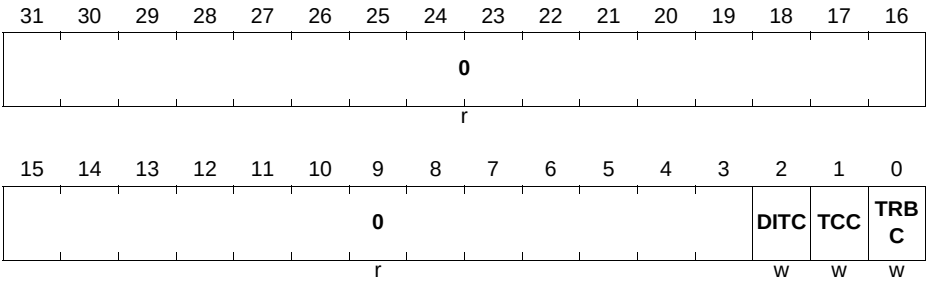


Field	Bits	Type	Description
TRBS	0	w	**Timer Run Bit set** Writing a 1_B into this field sets the run bit of the timer. Read always returns 0.
0	[31:1]	r	**Reserved** Read always returns 0

CC4yTCCLR

Through this register it is possible to stop and clear the timer, and clearing also the dither counter

CC4yTCCLR (y = 0 - 3)
Slice Timer Clear **($0110_H + 0100_H * y$)** **Reset Value: 00000000_H**



Field	Bits	Type	Description
TRBC	0	w	**Timer Run Bit Clear** Writing a 1_B into this field clears the run bit of the timer. The timer is not cleared. Read always returns 0.
TCC	1	w	**Timer Clear** Writing a 1_B into this field clears the timer to 0000_H. Read always returns 0.
DITC	2	w	**Dither Counter Clear** Writing a 1_B into this field clears the dither counter to 0_H. Read always returns 0.
0	[31:3]	r	**Reserved** Read always returns 0

CC4yTC

This register holds the several possible configurations for the timer operation.

CC4yTC (y = 0 - 3)
Slice Timer Control **($0114_H + 0100_H * y$)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0						MCME	EMT	EMS	TRPSW	TRPSE	0			TRAPE	FPE
r						rw	rw	rw	rw	rw	r			rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DIM	DITHE		CCS	SCE	STRM	ENDM		0	CAPC		ECM	CMOD	CLST	TSSM	TCM
rw	rw		rw	rw	rw	rw		r	rw		rw	rh	rw	rw	rw

Field	Bits	Type	Description
TCM	0	rw	**Timer Counting Mode** This field controls the actual counting scheme of the timer. 0_B Edge aligned mode 1_B Center aligned mode *Note: When using an external signal to control the counting direction, the counting scheme is always edge aligned.*
TSSM	1	rw	**Timer Single Shot Mode** This field controls the single shot mode. This is applicable in edge and center aligned modes. 0_B Single shot mode is disabled 1_B Single shot mode is enabled
CLST	2	rw	**Shadow Transfer on Clear** Setting this bit to 1_B enables a shadow transfer when a timer clearing action is performed. Notice that the shadow transfer enable bitfields on the **GCST** register still need to be set to 1_B via software.
CMOD	3	rh	**Capture Compare Mode** This field indicates in which mode the slice is operating. The default value is compare mode. The capture mode is automatically set by the HW when an external signal is mapped to a capture trigger. 0_B Compare Mode 1_B Capture Mode

Field	Bits	Type	Description
ECM	4	rw	**Extended Capture Mode** This field control the Capture mode of the specific slice. It only has effect if the CMOD bit is 1_B. 0_B Normal Capture Mode. Clear of the Full Flag of each capture register is done by accessing the registers individually only. 1_B Extended Capture Mode. Clear of the Full Flag of each capture register is done not only by accessing the individual registers but also by accessing the ECRD register. When reading the ECRD register, only the capture register register full flag pointed by the ECRD.VPTR is cleared.
CAPC	[6:5]	rw	**Clear on Capture Control** 00_B Timer is never cleared on a capture event 01_B Timer is cleared on a capture event into capture registers 2 and 3. (When SCE = 1_B, Timer is always cleared in a capture event) 10_B Timer is cleared on a capture event into capture registers 0 and 1. (When SCE = 1_B, Timer is always cleared in a capture event) 11_B Timer is always cleared in a capture event.
ENDM	[9:8]	rw	**Extended Stop Function Control** This field controls the extended functions of the external Stop signal. 00_B Clears the timer run bit only (default stop) 01_B Clears the timer only (flush) 10_B Clears the timer and run bit (flush/stop) 11_B Reserved *Note: When using an external up/down signal the flush operation sets the timer with zero if the counter is counting up and with the Period value if the counter is being decremented.*

Field	Bits	Type	Description
STRM	10	rw	**Extended Start Function Control** This field controls the extended functions of the external Start signal. 0_B Sets run bit only (default start) 1_B Clears the timer and sets run bit (flush/start) *Note: When using an external up/down signal the flush operation sets the timer with zero if the counter is being incremented and with the Period value if the counter is being decremented.*
SCE	11	rw	**Equal Capture Event enable** 0_B Capture into **CC4yC0V**/**CC4yC1V** registers control by CCycapt0 and capture into **CC4yC3V**/**CC4yC2V** control by CCycapt1 1_B Capture into **CC4yC0V**/**CC4yC1V** and **CC4yC3V**/**CC4yC2V** control by CCycapt1
CCS	12	rw	**Continuous Capture Enable** 0_B The capture into a specific capture register is done with the rules linked with the full flags, described at **Section 18.2.7.6**. 1_B The capture into the capture registers is always done regardless of the full flag status (even if the register has not been read back).
DITHE	[14:13]	rw	**Dither Enable** This field controls the dither mode for the slice. See **Section 18.2.10**. 00_B Dither is disabled 01_B Dither is applied to the Period 10_B Dither is applied to the Compare 11_B Dither is applied to the Period and Compare
DIM	15	rw	**Dither input selector** This fields selects if the dither control signal is connected to the dither logic of the specific slice of is connected to the dither logic of slice 0. Notice that even if this field is set to 1_B, the field DITHE still needs to be programmed. 0_B Slice is using its own dither unit 1_B Slice is connected to the dither unit of slice 0.

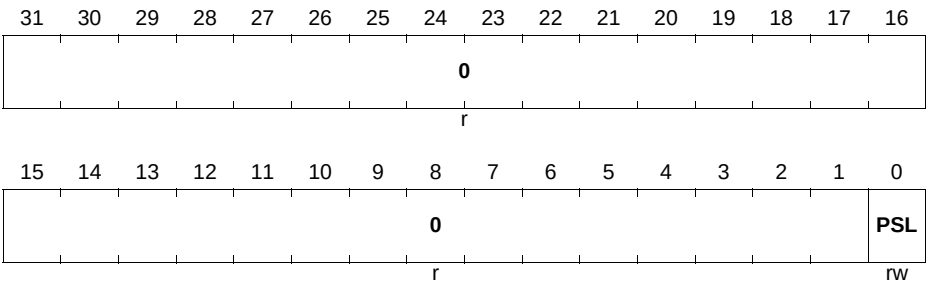
Field	Bits	Type	Description
FPE	16	rw	**Floating Prescaler enable** Setting this bit to 1_B enables the floating prescaler mode. 0_B Floating prescaler mode is disabled 1_B Floating prescaler mode is enabled
TRAPE	17	rw	**TRAP enable** Setting this bit to 1_B enables the TRAP action at the output pin. After mapping an external signal to the TRAP functionality, the user must set this field to 1_B to activate the effect of the TRAP on the output pin. Writing a 0_B into this field disables the effect of the TRAP function regardless of the state of the input signal. 0_B TRAP functionality has no effect on the output 1_B TRAP functionality affects the output
TRPSE	21	rw	**TRAP Synchronization Enable** Writing a 1_B into this bit enables a synchronous exiting with the PWM signal of the trap state. 0_B Exiting from TRAP state isn't synchronized with the PWM signal 1_B Exiting from TRAP state is synchronized with the PWM signal
TRPSW	22	rw	**TRAP State Clear Control** 0_B The slice exits the TRAP state automatically when the TRAP condition is not present 1_B The TRAP state can only be exited by a SW request.
EMS	23	rw	**External Modulation Synchronization** Setting this bit to 1_B enables the synchronization of the external modulation functionality with the PWM period. 0_B External Modulation functionality is not synchronized with the PWM signal 1_B External Modulation functionality is synchronized with the PWM signal

Field	Bits	Type	Description
EMT	24	rw	**External Modulation Type** This field selects if the external modulation event is clearing the CC4yST bit or if is gating the outputs. 0_B External Modulation functionality is clearing the CC4yST bit. 1_B External Modulation functionality is gating the outputs.
MCME	25	rw	**Multi Channel Mode Enable** 0_B Multi Channel Mode is disabled 1_B Multi Channel Mode is enabled
0	7, [20:18], [31:26]	r	**Reserved** Read always returns 0

CC4yPSL

This register holds the configuration for the output passive level control.

CC4yPSL (y = 0 - 3)
Passive Level Config **($0118_H + 0100_H * y$)** **Reset Value: 00000000_H**



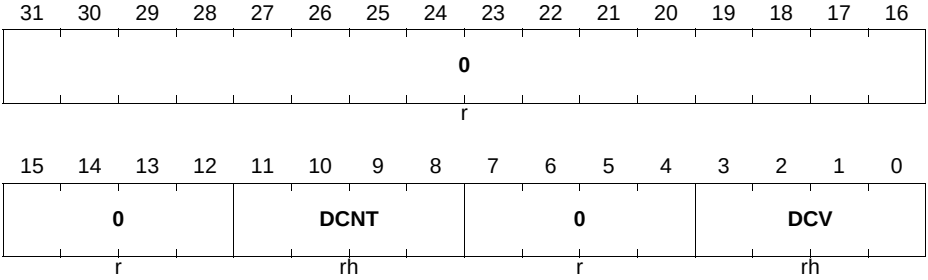
Field	Bits	Type	Description
PSL	0	rw	**Output Passive Level** This field controls the passive level of the output pin. 0_B Passive Level is LOW 1_B Passive Level is HIGH A write always addresses the shadow register, while a read always returns the current used value.

Field	Bits	Type	Description
0	[31:1]	r	**Reserved** A read access always returns 0

CC4yDIT

This register holds the current dither compare and dither counter values.

CC4yDIT (y = 0 - 3)
Dither Config **($011C_H + 0100_H * y$)** **Reset Value: 00000000_H**

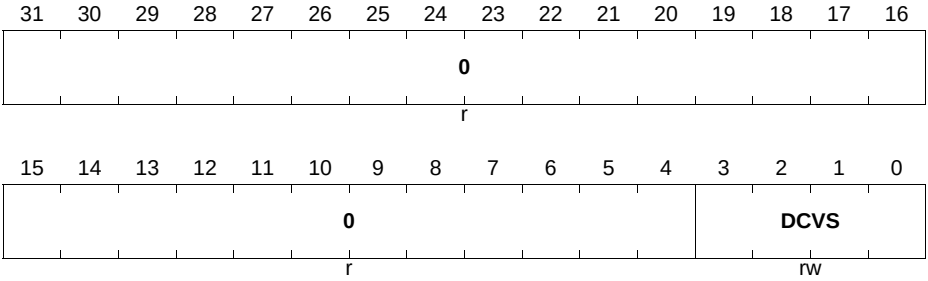


Field	Bits	Type	Description
DCV	[3:0]	rh	**Dither compare Value** This field contains the value used for the dither comparison. This value is updated when a shadow transfer occurs with the **CC4yDITS**.DCVS.
DCNT	[11:8]	rh	**Dither counter actual value**
0	[7:4], [31:12]	r	**Reserved** Read always returns 0.

CC4yDITS

This register contains the value that is going to be loaded into the **CC4yDIT**.DCV when the next shadow transfer occurs.

CC4yDITS (y = 0 - 3)
Dither Shadow Register **($0120_H + 0100_H * y$)** **Reset Value: 00000000_H**

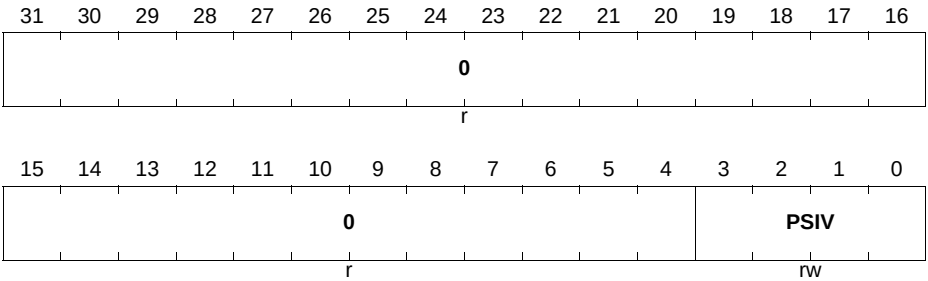


Field	Bits	Type	Description
DCVS	[3:0]	rw	**Dither Shadow Compare Value** This field contains the value that is going to be set on the dither compare value, **CC4yDIT**.DCV, within the next shadow transfer.
0	[31:4]	r	**Reserved** Read always returns 0.

CC4yPSC

This register contains the value that is loaded into the prescaler during restart.

CC4yPSC (y = 0 - 3)
Prescaler Control **($0124_H + 0100_H * y$)** **Reset Value: 00000000_H**

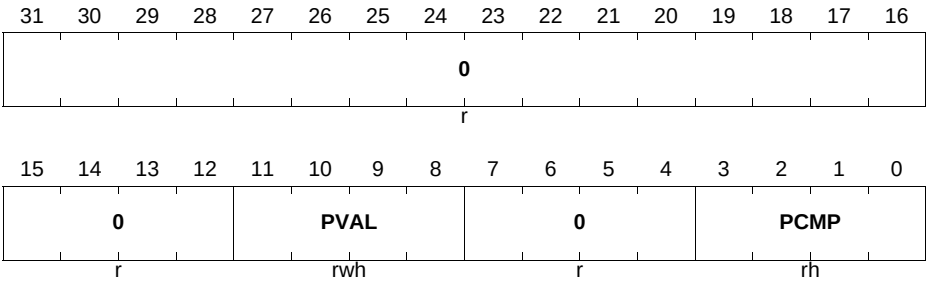


Field	Bits	Type	Description
PSIV	[3:0]	rw	**Prescaler Initial Value** This field contains the value that is applied to the Prescaler at startup. When floating prescaler mode is used, this value is applied when a timer compare match AND prescaler compare match occurs or when a capture event is triggered.
0	[31:4]	r	**Reserved** Read always returns 0.

CC4yFPC

This register contains the value used for the floating prescaler compare and the actual prescaler division value.

CC4yFPC (y = 0 - 3)
Floating Prescaler Control **(0128_H + 0100_H * y)** **Reset Value: 00000000_H**



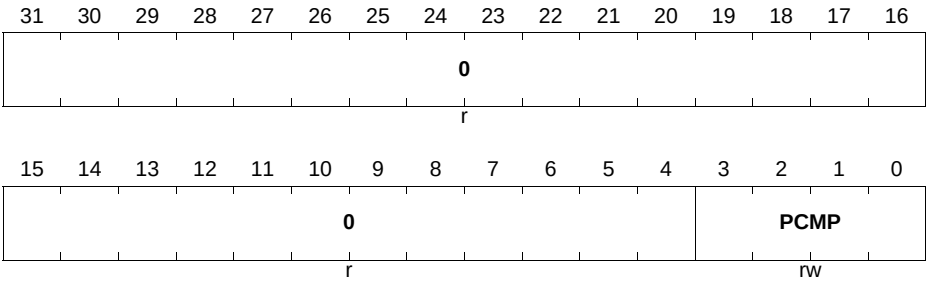
Field	Bits	Type	Description
PCMP	[3:0]	rh	**Floating Prescaler Compare Value** This field contains comparison value used in floating prescaler mode. The comparison is triggered by the Timer Compare match event. See **Section 18.2.11.2**.

Field	Bits	Type	Description
PVAL	[11:8]	rwh	**Actual Prescaler Value** See **Table 18-7**. Writing into this register is only possible when the prescaler is stopped. When the floating prescaler mode is not used, this value is equal to the **CC4yPSC**.PSIV.
0	[7:4], [15:12], [31:16]	r	**Reserved** Read always returns 0.

CC4yFPCS

This register contains the value that is going to be transferred to the **CC4yFPC**.PCMP field within the next shadow transfer update.

CC4yFPCS (y = 0 - 3)
Floating Prescaler Shadow **($012C_H + 0100_H * y$)** **Reset Value: 00000000_H**

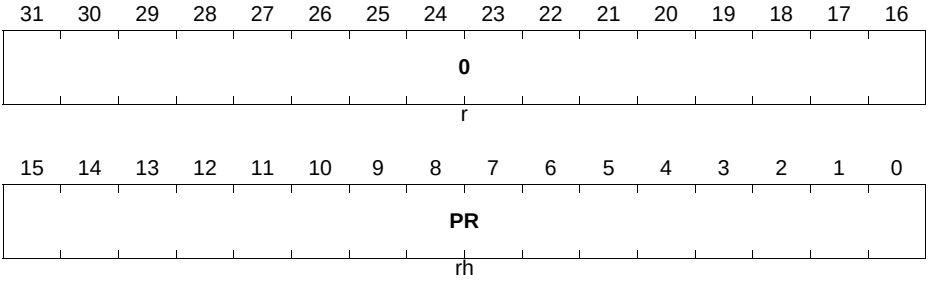


Field	Bits	Type	Description
PCMP	[3:0]	rw	**Floating Prescaler Shadow Compare Value** This field contains the value that is going to be set on the **CC4yFPC**.PCMP within the next shadow transfer. See **Table 18-7**.
0	[31:4]	r	**Reserved** Read always returns 0.

CC4yPR

This register contains the actual value for the timer period.

CC4yPR (y = 0 - 3)
Timer Period Value **($0130_H + 0100_H * y$)** **Reset Value: 00000000_H**

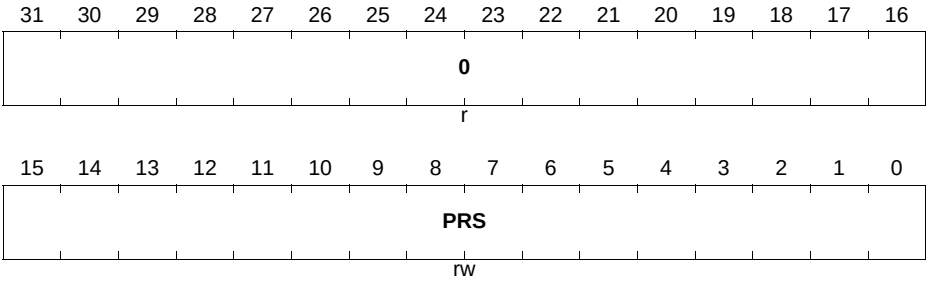


Field	Bits	Type	Description
PR	[15:0]	rh	**Period Register** Contains the value of the timer period. *Note: In Capture Mode when a external signal is selected for capturing the timer value into the capture registers 2 and 3, PR is not accessible for writing. A read always returns 0.*
0	[31:16]	r	**Reserved** A read always returns 0.

CC4yPRS

This register contains the value for the timer period that is going to be transferred into the **CC4yPR**.PR field when the next shadow transfer occurs.

CC4yPRS (y = 0 - 3)
Timer Shadow Period Value **($0134_H + 0100_H * y$)** **Reset Value: 00000000_H**

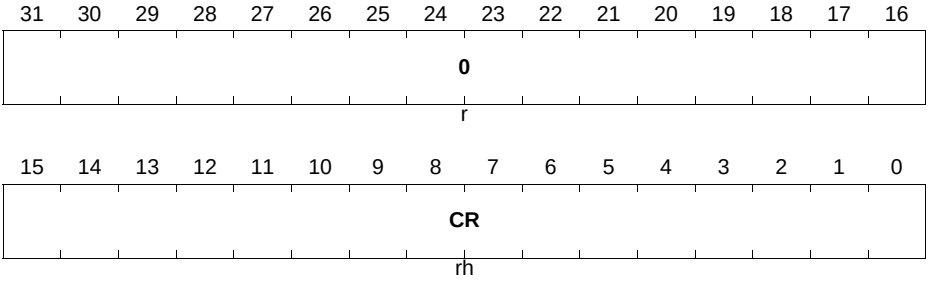


Field	Bits	Type	Description
PRS	[15:0]	rw	**Period Register** Contains the value of the timer period, that is going to be passed into the **CC4yPR**.PR field when the next shadow transfer occurs. *Note: In Capture Mode when a external signal is selected for capturing the timer value into the capture registers 2 and 3, the PRS is not accessible for writing. A read always returns 0.*
0	[31:16]	r	**Reserved** A read always returns 0.

CC4yCR

This register contains the value for the timer comparison.

CC4yCR (y = 0 - 3)
Timer Compare Value **(0138_H + 0100_H * y)** **Reset Value: 00000000_H**

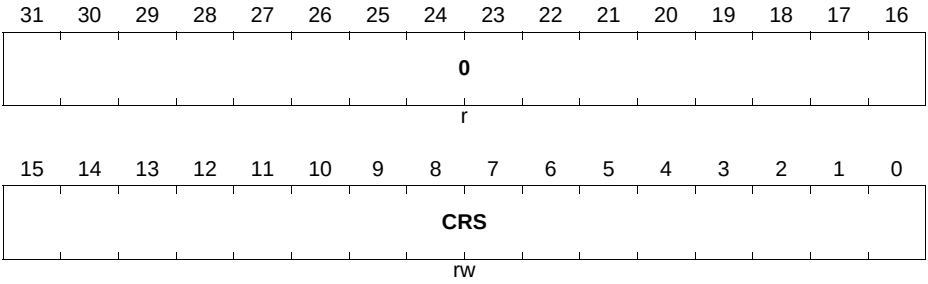


Field	Bits	Type	Description
CR	[15:0]	rh	**Compare Register** Contains the value for the timer comparison. *Note: In Capture Mode when a external signal is selected for capturing the timer value into the capture registers 0 and 1, a read always returns 0.*
0	[31:16]	r	**Reserved** A read always returns 0.

CC4yCRS

This register contains the value that is going to be loaded into the **CC4yCR**.CR field when the next shadow transfer occurs.

CC4yCRS (y = 0 - 3)
Timer Shadow Compare Value ($013C_H$ + 0100_H * y) **Reset Value: 00000000_H**

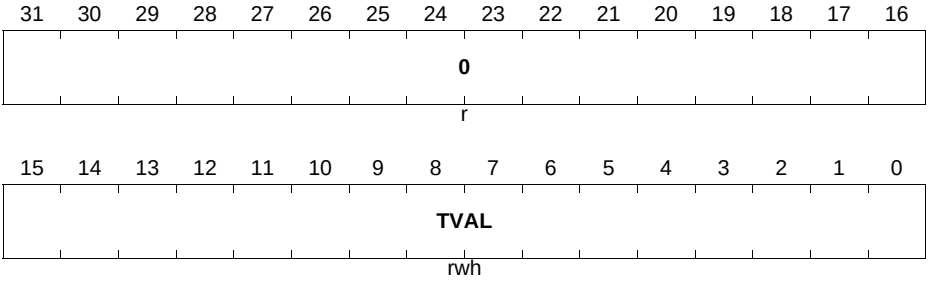


Field	Bits	Type	Description
CRS	[15:0]	rw	**Compare Register** Contains the value for the timer comparison, that is going to be passed into the **CC4yCR**.CR field when the next shadow transfer occurs. *Note: In Capture Mode when a external signal is selected for capturing the timer value into the capture registers 0 and 1, a read always returns 0.*
0	[31:16]	r	**Reserved** A read always returns 0.

CC4yTIMER

This register contains the current value of the timer.

CC4yTIMER (y = 0 - 3)
Timer Value **($0170_H + 0100_H * y$)** **Reset Value: 00000000_H**

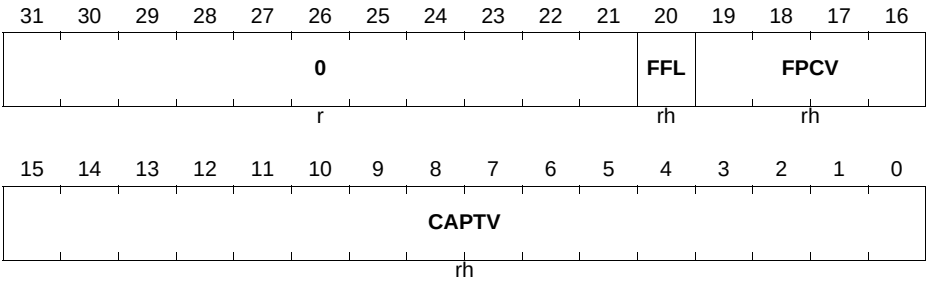


Field	Bits	Type	Description
TVAL	[15:0]	rwh	**Timer Value** This field contains the actual value of the timer. A write access is only possible when the timer is stopped.
0	[31:16]	r	**Reserved** A read access always returns 0

CC4yC0V

This register contains the values associated with the Capture 0 field.

CC4yC0V (y = 0 - 3)
Capture Register 0 **($0174_H + 0100_H * y$)** **Reset Value: 00000000_H**

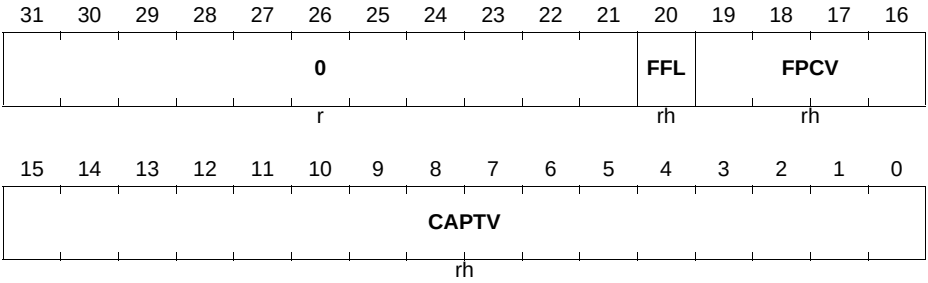


Field	Bits	Type	Description
CAPTV	[15:0]	rh	**Capture Value** This field contains the capture register 0 value. See **Figure 18-26**. In compare mode a read access always returns 0.
FPCV	[19:16]	rh	**Prescaler Value** This field contains the prescaler value at the time of the capture event into the capture register 0. In compare mode a read access always returns 0.
FFL	20	rh	**Full Flag** This bit indicates if a new value was capture into the capture register 0 after the last read access. See **Figure 18-26**. In compare mode a read access always returns 0. 0_B No new value was captured into the specific capture register 1_B A new value was captured into the specific register
0	[31:21]	r	**Reserved** A read always returns 0

CC4yC1V

This register contains the values associated with the Capture 1 field.

CC4yC1V (y = 0 - 3)
Capture Register 1 **($0178_H + 0100_H * y$)** **Reset Value: 00000000_H**

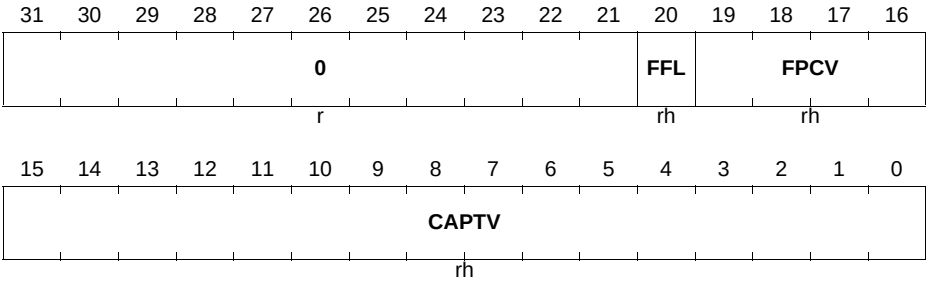


Field	Bits	Type	Description
CAPTV	[15:0]	rh	**Capture Value** This field contains the capture register 1 value. See **Figure 18-26**. In compare mode a read access always returns 0.
FPCV	[19:16]	rh	**Prescaler Value** This field contains the prescaler value at the time of the capture event into the capture register 1. In compare mode a read access always returns 0.
FFL	20	rh	**Full Flag** This bit indicates if a new value was capture into the capture register 1 after the last read access. See **Figure 18-26**. In compare mode a read access always returns 0. 0_B No new value was captured into the specific capture register 1_B A new value was captured into the specific register
0	[31:21]	r	**Reserved** A read always returns 0

CC4yC2V

This register contains the values associated with the Capture 2 field.

CC4yC2V (y = 0 - 3)
Capture Register 2 **($017C_H + 0100_H * y$)** **Reset Value: 00000000_H**

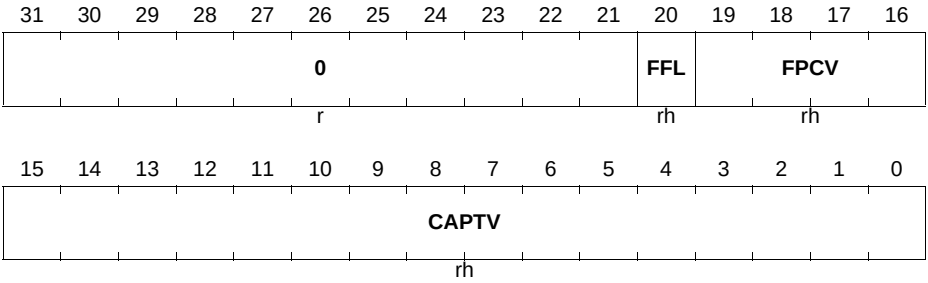


Field	Bits	Type	Description
CAPTV	[15:0]	rh	**Capture Value** This field contains the capture register 2 value. See **Figure 18-26**. In compare mode a read access always returns 0.
FPCV	[19:16]	rh	**Prescaler Value** This field contains the prescaler value at the time of the capture event into the capture register 2. In compare mode a read access always returns 0.
FFL	20	rh	**Full Flag** This bit indicates if a new value was capture into the capture register 2 after the last read access. See **Figure 18-26**. In compare mode a read access always returns 0. 0_B No new value was captured into the specific capture register 1_B A new value was captured into the specific register
0	[31:21]	r	**Reserved** A read always returns 0

CC4yC3V

This register contains the values associated with the Capture 3 field.

CC4yC3V (y = 0 - 3)
Capture Register 3 **($0180_H + 0100_H * y$)** **Reset Value: 00000000_H**



Field	Bits	Type	Description
CAPTV	[15:0]	rh	**Capture Value** This field contains the capture register 3 value. See **Figure 18-26**. In compare mode a read access always returns 0.
FPCV	[19:16]	rh	**Prescaler Value** This field contains the prescaler value at the time of the capture event into the capture register 3. In compare mode a read access always returns 0.
FFL	20	rh	**Full Flag** This bit indicates if a new value was capture into the capture register 3 after the last read access. See **Figure 18-26**. In compare mode a read access always returns 0. 0_B No new value was captured into the specific capture register 1_B A new value was captured into the specific register
0	[31:21]	r	**Reserved** A read always returns 0

CC4yINTS

This register contains the status of all interrupt sources.

CC4yINTS (y = 0 - 3)
Interrupt Status **($01A0_H + 0100_H * y$)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				TRPF	E2AS	E1AS	E0AS	0				CMDS	CMUS	OMDS	PMUS
r				rh	rh	rh	rh	r				rh	rh	rh	rh

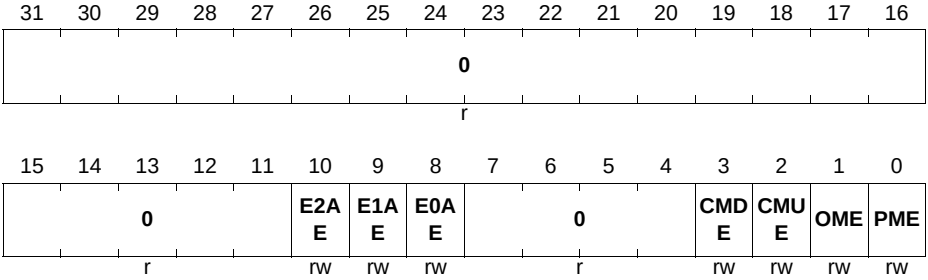
Field	Bits	Type	Description
PMUS	0	rh	**Period Match while Counting Up** 0_B Period match while counting up not detected 1_B Period match while counting up detected
OMDS	1	rh	**One Match while Counting Down** 0_B One match while counting down not detected 1_B One match while counting down detected
CMUS	2	rh	**Compare Match while Counting Up** 0_B Compare match while counting up not detected 1_B Compare match while counting up detected
CMDS	3	rh	**Compare Match while Counting Down** 0_B Compare match while counting down not detected 1_B Compare match while counting down detected
E0AS	8	rh	**Event 0 Detection Status** Depending on the user selection on the **CC4yINS**.EV0EM, this bit can be set when a rising, falling or both transitions are detected. 0_B Event 0 not detected 1_B Event 0 detected
E1AS	9	rh	**Event 1 Detection Status** Depending on the user selection on the **CC4yINS**.EV1EM, this bit can be set when a rising, falling or both transitions are detected. 0_B Event 1 not detected 1_B Event 1 detected
E2AS	10	rh	**Event 2 Detection Status** Depending on the user selection on the **CC4yINS**.EV1EM, this bit can be set when a rising, falling or both transitions are detected. 0_B Event 2 not detected 1_B Event 2 detected *Note: If this event is linked with the TRAP function, this field is automatically cleared when the slice exits the Trap State.*
TRPF	11	rh	**Trap Flag Status** This field contains the status of the Trap Flag.

Field	Bits	Type	Description
0	[7:4], [31:12]	r	**Reserved** A read always returns 0.

CC4yINTE

Through this register it is possible to enable or disable the specific interrupt source(s).

CC4yINTE (y = 0 - 3)
Interrupt Enable Control **($01A4_H + 0100_H * y$)** **Reset Value: 00000000_H**



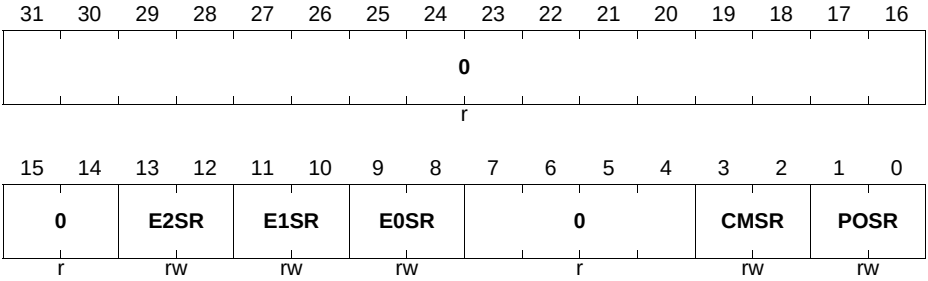
Field	Bits	Type	Description
PME	0	rw	**Period match while counting up enable** Setting this bit to 1_B enables the generation of an interrupt pulse every time a period match while counting up occurs. 0_B Period Match interrupt is disabled 1_B Period Match interrupt is enabled
OME	1	rw	**One match while counting down enable** Setting this bit to 1_B enables the generation of an interrupt pulse every time an one match while counting down occurs. 0_B One Match interrupt is disabled 1_B One Match interrupt is enabled

Field	Bits	Type	Description
CMUE	2	rw	**Compare match while counting up enable** Setting this bit to 1_B enables the generation of an interrupt pulse every time a compare match while counting up occurs. 0_B Compare Match while counting up interrupt is disabled 1_B Compare Match while counting up interrupt is enabled
CMDE	3	rw	**Compare match while counting down enable** Setting this bit to 1_B enables the generation of an interrupt pulse every time a compare match while counting down occurs. 0_B Compare Match while counting down interrupt is disabled 1_B Compare Match while counting down interrupt is enabled
E0AE	8	rw	**Event 0 interrupt enable** Setting this bit to 1_B enables the generation of an interrupt pulse every time that Event 0 is detected. 0_B Event 0 detection interrupt is disabled 1_B Event 0 detection interrupt is enabled
E1AE	9	rw	**Event 1 interrupt enable** Setting this bit to 1_B enables the generation of an interrupt pulse every time that Event 1 is detected. 0_B Event 1 detection interrupt is disabled 1_B Event 1 detection interrupt is enabled
E2AE	10	rw	**Event 2 interrupt enable** Setting this bit to 1_B enables the generation of an interrupt pulse every time that Event 2 is detected. 0_B Event 2 detection interrupt is disabled 1_B Event 2 detection interrupt is enabled
0	[7:4], [31:11]	r	**Reserved** A read always returns 0

CC4ySRS

Through this register it is possible to select to which service request line each interrupt source is forwarded.

CC4ySRS (y = 0 - 3)
Service Request Selector **($01A8_H + 0100_H * y$)** **Reset Value: 00000000_H**



Field	Bits	Type	Description
POSR	[1:0]	rw	**Period/One match Service request selector** This field selects to which slice Service request line, the interrupt(s) generated by the Period match while counting up and One match while counting down are going to be forward. 00_B Forward to CC4ySR0 01_B Forward to CC4ySR1 10_B Forward to CC4ySR2 11_B Forward to CC4ySR3
CMSR	[3:2]	rw	**Compare match Service request selector** This field selects to which slice Service request line, the interrupt(s) generated by the Compare match while counting up and Compare match while counting down are going to be forward. 00_B Forward to CC4ySR0 01_B Forward to CC4ySR1 10_B Forward to CC4ySR2 11_B Forward to CC4ySR3
E0SR	[9:8]	rw	**Event 0 Service request selector** This field selects to which slice Service request line, the interrupt generated by the Event 0 detection is going to be forward. 00_B Forward to CC4ySR0 01_B Forward to CC4ySR1 10_B Forward to CC4ySR2 11_B Forward to CC4ySR3

Field	Bits	Type	Description
E1SR	[11:10]	rw	**Event 1 Service request selector** This field selects to which slice Service request line, the interrupt generated by the Event 1detection is going to be forward. 00_B Forward to CC4ySR0 01_B Forward to CC4ySR1 10_B Forward to CC4ySR2 11_B Forward to CC4ySR3
E2SR	[13:12]	rw	**Event 2 Service request selector** This field selects to which slice Service request line, the interrupt generated by the Event 2 detection is going to be forward. 00_B Forward to CC4ySR0 01_B Forward to CC4ySR1 10_B Forward to CC4ySR2 11_B Forward to CC4ySR3
0	[7:4], [31:14]	r	**Reserved** Read always returns 0.

CC4ySWS

Through this register it is possible for the SW to set a specific interrupt status flag.

CC4ySWS (y = 0 - 3)
Interrupt Status Set **($01AC_H + 0100_H * y$)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

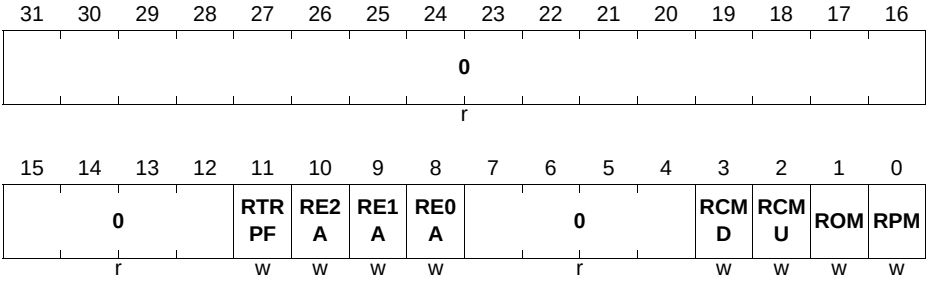
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				STRPF	SE2A	SE1A	SE0A	0				SCMD	SCMU	SOM	SPM
r				w	w	w	w	r				w	w	w	w

Field	Bits	Type	Description
SPM	0	w	**Period match while counting up set** Writing a 1_B into this field sets the **CC4yINTS**.PMUS bit. An interrupt pulse is generated if the source is enabled. A read always returns 0.
SOM	1	w	**One match while counting down set** Writing a 1_B into this bit sets the **CC4yINTS**.OMDS bit. An interrupt pulse is generated if the source is enabled. A read always returns 0.
SCMU	2	w	**Compare match while counting up set** Writing a 1_B into this field sets the **CC4yINTS**.CMUS bit. An interrupt pulse is generated if the source is enabled. A read always returns 0.
SCMD	3	w	**Compare match while counting down set** Writing a 1_B into this bit sets the **CC4yINTS**.CMDS bit. An interrupt pulse is generated if the source is enabled. A read always returns 0.
SE0A	8	w	**Event 0 detection set** Writing a 1_B into this bit sets the **CC4yINTS**.E0AS bit. An interrupt pulse is generated if the source is enabled. A read always returns 0.
SE1A	9	w	**Event 1 detection set** Writing a 1_B into this bit sets the **CC4yINTS**.E1AS bit. An interrupt pulse is generated if the source is enabled. A read always returns 0.
SE2A	10	w	**Event 2 detection set** Writing a 1_B into this bit sets the **CC4yINTS**.E2AS bit. An interrupt pulse is generated if the source is enabled. A read always returns 0.
STRPF	11	w	**Trap Flag status set** Writing a 1_B into this bit sets the **CC4yINTS**.TRPF bit. A read always returns 0.
0	[7:4], [31:12]	r	**Reserved** Read always returns 0

CC4ySWR

Through this register it is possible for the SW to clear a specific interrupt status flag.

CC4ySWR (y = 0 - 3)
Interrupt Status Clear **($01B0_H + 0100_H * y$)** **Reset Value: 00000000_H**



Field	Bits	Type	Description
RPM	0	w	**Period match while counting up clear** Writing a 1_B into this field clears the **CC4yINTS**.PMUS bit. A read always returns 0.
ROM	1	w	**One match while counting down clear** Writing a 1_B into this bit clears the **CC4yINTS**.OMDS bit. A read always returns 0.
RCMU	2	w	**Compare match while counting up clear** Writing a 1_B into this field clears the **CC4yINTS**.CMUS bit. A read always returns 0.
RCMD	3	w	**Compare match while counting down clear** Writing a 1_B into this bit clears the **CC4yINTS**.CMDS bit. A read always returns 0.
RE0A	8	w	**Event 0 detection clear** Writing a 1_B into this bit clears the **CC4yINTS**.E0AS bit. A read always returns 0.
RE1A	9	w	**Event 1 detection clear** Writing a 1_B into this bit clears the **CC4yINTS**.E1AS bit. A read always returns 0.
RE2A	10	w	**Event 2 detection clear** Writing a 1_B into this bit clears the **CC4yINTS**.E2AS bit. A read always returns 0.

Field	Bits	Type	Description
RTRPF	11	w	**Trap Flag status clear** Writing a 1_B into this bit clears the **CC4yINTS**.TRPF bit. Not valid if **CC4yTC**.TRPEN = 1_B and the Trap State is still active. A read always returns 0.
0	[7:4], [31:12]	r	**Reserved** Read always returns 0

18.8 Interconnects

The tables that refer to the “global pins” are the ones that contain the inputs/outputs of each module that are common to all slices.

The GPIO connections are available at the Ports chapter.

18.8.1 CCU40 pins

Table 18-13 CCU40 Pin Connections

Global Inputs/Outputs	I/O	Connected To	Description
CCU40.MCLK	I	SCU.CCUCLK	Kernel clock
CCU40.CLKA	I	ERU1.IOUT0	another count source for the prescaler
CCU40.CLKB	I	ERU1.IOUT1	another count source for the prescaler
CCU40.CLKC	I	0	another count source for the prescaler
CCU40.MCSS	I	0	Multi pattern sync with shadow transfer trigger
CCU40.SR0	O	NVIC; DMA; POSIF0.MSETA; VADC.G0REQGTC; VADC.G1REQGTC; VADC.BGREQGTC; HRPWM0.BL0K	Service request line

Table 18-13 CCU40 Pin Connections

Global Inputs/Outputs	I/O	Connected To	Description
CCU40.SR1	O	NVIC; DMA; DAC.TRIGGER[2]; U0C0.DX2E; HRPWM0.BL1K	Service request line
CCU40.SR2	O	NVIC; VADC.G0REQTRA; VADC.G1REQTRA; VADC.BGREQTRA; HRPWM0.BL2K	Service request line
CCU40.SR3	O	NVIC; CCU80.IGBTB; VADC.G0REQTRB; VADC.G1REQTRB; VADC.BGREQTRB; CCU80.IN0K;	Service request line

Table 18-14 CCU40 - CC40 Pin Connections

Input/Output	I/O	Connected To	Description
CCU40.IN0A	I	PORTS	General purpose function
CCU40.IN0B	I	PORTS	General purpose function
CCU40.IN0C	I	PORTS	General purpose function
CCU40.IN0D	I	ERU1.PDOUT1	General purpose function
CCU40.IN0E	I	POSIF0.OUT0	General purpose function
CCU40.IN0F	I	POSIF0.OUT1	General purpose function
CCU40.IN0G	I	POSIF0.OUT3	General purpose function
CCU40.IN0H	I	CAN.SR7	General purpose function
CCU40.IN0I	I	SCU.GSC40	General purpose function
CCU40.IN0J	I	ERU1.PDOUT0	General purpose function
CCU40.IN0K	I	ERU1.IOUT0	General purpose function
CCU40.IN0L	I	U0C0.DX2INS	General purpose function
CCU40.IN0M	I	CCU40.ST0	General purpose function
CCU40.IN0N	I	CCU40.ST1	General purpose function
CCU40.IN0O	I	CCU40.ST2	General purpose function

Table 18-14 CCU40 - CC40 Pin Connections

Input/Output	I/O	Connected To	Description
CCU40.IN0P	I	CCU40.ST3	General purpose function
CCU40.MCI0	I	0	Multi Channel pattern input
CCU40.OUT0	O	PORTS	Slice compare output
CCU40.GP00	O	NOT CONNECTED	Selected signal for event 0
CCU40.GP01	O	NOT CONNECTED	Selected signal for event 1
CCU40.GP02	O	NOT CONNECTED	Selected signal for event 2
CCU40.ST0	O	ERU1.0A2; ERU1.OGU02; POSIF0.HSDA; HRPWM0.BL0G	Slice status bit
CCU40.PS0	O	NOT CONNECTED	Multi channel pattern sync trigger: PM when counting UP (edge aligned) or OM when counting DOWN (center aligned)

Table 18-15 CCU40 - CC41 Pin Connections

Input/Output	I/O	Connected To	Description
CCU40.IN1A	I	PORTS	General purpose function
CCU40.IN1B	I	PORTS	General purpose function
CCU40.IN1C	I	PORTS	General purpose function
CCU40.IN1D	I	ERU1.PDOUT0	General purpose function
CCU40.IN1E	I	POSIF0.OUT0	General purpose function
CCU40.IN1F	I	POSIF0.OUT1	General purpose function
CCU40.IN1G	I	POSIF0.OUT3	General purpose function
CCU40.IN1H	I	POSIF0.OUT4	General purpose function
CCU40.IN1I	I	SCU.GSC40	General purpose function
CCU40.IN1J	I	ERU1.PDOUT1	General purpose function
CCU40.IN1K	I	ERU1.IOUT1	General purpose function
CCU40.IN1L	I	POSIF0.OUT2	General purpose function
CCU40.IN1M	I	CCU40.ST0	General purpose function
CCU40.IN1N	I	CCU40.ST1	General purpose function

Table 18-15 CCU40 - CC41 Pin Connections

Input/Output	I/O	Connected To	Description
CCU40.IN1O	I	CCU40.ST2	General purpose function
CCU40.IN1P	I	CCU40.ST3	General purpose function
CCU40.MCI1	I	0	Multi Channel pattern input
CCU40.OUT1	O	PORTS	Slice compare output
CCU40.GP10	O	NOT CONNECTED	Selected signal for event 0
CCU40.GP11	O	NOT CONNECTED	Selected signal for event 1
CCU40.GP12	O	NOT CONNECTED	Selected signal for event 2
CCU40.ST1	O	POSIF0.MSETB; ERU1.1A2; HRPWM0.BL1G	Slice status bit
CCU40.PS1	O	POSIF0.SYNCC	Multi channel pattern sync trigger: PM when counting UP (edge aligned) or OM when counting DOWN (center aligned)

Table 18-16 CCU40 - CC42 Pin Connections

Input/Output	I/O	Connected To	Description
CCU40.IN2A	I	PORTS	General purpose function
CCU40.IN2B	I	PORTS	General purpose function
CCU40.IN2C	I	PORTS	General purpose function
CCU40.IN2D	I	ERU1.PDOUT0	General purpose function
CCU40.IN2E	I	POSIF0.OUT0	General purpose function
CCU40.IN2F	I	POSIF0.OUT2	General purpose function
CCU40.IN2G	I	POSIF0.OUT3	General purpose function
CCU40.IN2H	I	POSIF0.OUT4	General purpose function
CCU40.IN2I	I	SCU.GSC40	General purpose function
CCU40.IN2J	I	ERU1.PDOUT2	General purpose function
CCU40.IN2K	I	ERU1.IOUT2	General purpose function
CCU40.IN2L	I	U0C1.DX2INS	General purpose function
CCU40.IN2M	I	CCU40.ST0	General purpose function
CCU40.IN2N	I	CCU40.ST1	General purpose function

Table 18-16 CCU40 - CC42 Pin Connections

Input/Output	I/O	Connected To	Description
CCU40.IN2O	I	CCU40.ST2	General purpose function
CCU40.IN2P	I	CCU40.ST3	General purpose function
CCU40.MCI2	I	0	Multi Channel pattern input
CCU40.OUT2	O	PORTS	Slice compare output
CCU40.GP20	O	NOT CONNECTED	Selected signal for event 0
CCU40.GP21	O	NOT CONNECTED	Selected signal for event 1
CCU40.GP22	O	NOT CONNECTED	Selected signal for event 2
CCU40.ST2	O	ERU1.2A2; HRPWM0.BL2G	Slice status bit
CCU40.PS2	O	NOT CONNECTED	Multi channel pattern sync trigger: PM when counting UP (edge aligned) or OM when counting DOWN (center aligned)

Table 18-17 CCU40 - CC43 Pin Connections

Input/Output	I/O	Connected To	Description
CCU40.IN3A	I	PORTS	General purpose function
CCU40.IN3B	I	PORTS	General purpose function
CCU40.IN3C	I	PORTS	General purpose function
CCU40.IN3D	I	ERU1.PDOUT0	General purpose function
CCU40.IN3E	I	POSIF0.OUT3	General purpose function
CCU40.IN3F	I	POSIF0.OUT5	General purpose function
CCU40.IN3G	I	VADC.G0ARBCNT	General purpose function
CCU40.IN3H	I	CCU80.IGBTO	General purpose function
CCU40.IN3I	I	SCU.GSC40	General purpose function
CCU40.IN3J	I	ERU1.PDOUT3	General purpose function
CCU40.IN3K	I	ERU1.IOUT3	General purpose function
CCU40.IN3L	I	U1C0.DX2INS	General purpose function
CCU40.IN3M	I	CCU40.ST0	General purpose function
CCU40.IN3N	I	CCU40.ST1	General purpose function

Table 18-17 CCU40 - CC43 Pin Connections

Input/Output	I/O	Connected To	Description
CCU40.IN3O	I	CCU40.ST2	General purpose function
CCU40.IN3P	I	CCU40.ST3	General purpose function
CCU40.MCI3	I	0	Multi Channel pattern input
CCU40.OUT3	O	PORTS	Slice compare output
CCU40.GP30	O	NOT CONNECTED	Selected signal for event 0
CCU40.GP31	O	NOT CONNECTED	Selected signal for event 1
CCU40.GP32	O	NOT CONNECTED	Selected signal for event 2
CCU40.ST3	O	VADC.G0REQGTA; VADC.G1REQGTA; VADC.BGREQGTA; ERU1.3A2; CCU80.IGBTA;	Slice status bit
CCU40.PS3	O	NOT CONNECTED	Multi channel pattern sync trigger: PM when counting UP (edge aligned) or OM when counting DOWN (center aligned)

18.8.2 CCU41 pins

Table 18-18 CCU41 Pin Connections

Global Inputs/Outputs	I/O	Connected To	Description
CCU41.MCLK	I	SCU.CCUCLK	Kernel clock
CCU41.CLKA	I	ERU1.IOUT0	another count source for the prescaler
CCU41.CLKB	I	ERU1.IOUT1	another count source for the prescaler
CCU41.CLKC	I	0	another count source for the prescaler
CCU41.MCSS	I	0	Multi pattern sync with shadow transfer trigger

Table 18-18 CCU41 Pin Connections

Global Inputs/Outputs	I/O	Connected To	Description
CCU41.SR0	O	NVIC; DMA; HRPWM0.BL0L	Service request line
CCU41.SR1	O	NVIC; DMA; DAC.TRIGGER[3]; VADC.G0REQGTD; VADC.G1REQGTD; VADC.BGREQGTD; U1C0.DX2E; HRPWM0.BL1L	Service request line
CCU41.SR2	O	NVIC; VADC.G0REQTRC; VADC.G1REQTRC; VADC.BGREQTRC; HRPWM0.BL2L	Service request line
CCU41.SR3	O	NVIC; VADC.G0REQTRD; VADC.G1REQTRD; VADC.BGREQTRD; CCU80.IN1K; HRPWM0.BL0J; HRPWM0.BL1J; HRPWM0.BL2J; HRPWM0.SC0IJ; HRPWM0.SC1IJ; HRPWM0.SC2IJ	Service request line

Table 18-19 CCU41 - CC40 Pin Connections

Input/Output	I/O	Connected To	Description
CCU41.IN0A	I	PORTS	General purpose function
CCU41.IN0B	I	PORTS	General purpose function
CCU41.IN0C	I	PORTS	General purpose function
CCU41.IN0D	I	ERU1.PDOUT1	General purpose function
CCU41.IN0E	I	HRPWM0.SR3	General purpose function
CCU41.IN0F	I	HRPWM0.QOUT0	General purpose function

Table 18-19 CCU41 - CC40 Pin Connections

Input/Output	I/O	Connected To	Description
CCU41.IN0G	I	HRPWM0.C0O	General purpose function
CCU41.IN0H	I	CAN.SR7	General purpose function
CCU41.IN0I	I	SCU.GSC40	General purpose function
CCU41.IN0J	I	ERU1.PDOUT0	General purpose function
CCU41.IN0K	I	ERU1.IOUT0	General purpose function
CCU41.IN0L	I	VADC.G0BFL0	General purpose function
CCU41.IN0M	I	CCU41.ST0	General purpose function
CCU41.IN0N	I	CCU41.ST1	General purpose function
CCU41.IN0O	I	CCU41.ST2	General purpose function
CCU41.IN0P	I	CCU41.ST3	General purpose function
CCU41.MCI0	I	0	Multi Channel pattern input
CCU41.OUT0	O	PORTS	Slice compare output
CCU41.GP00	O	NOT CONNECTED	Selected signal for event 0
CCU41.GP01	O	NOT CONNECTED	Selected signal for event 1
CCU41.GP02	O	NOT CONNECTED	Selected signal for event 2
CCU41.ST0	O	ERU1.OGU12; HRPWM0.ECLK0A; HRPWM0.ECLK1A; HRPWM0.ECLK2A; HRPWM0.BL0H;	Slice status bit
CCU41.PS0	O	NOT CONNECTED	Multi channel pattern sync trigger: PM when counting UP (edge aligned) or OM when counting DOWN (center aligned)

Table 18-20 CCU41 - CC41 Pin Connections

Input/Output	I/O	Connected To	Description
CCU41.IN1A	I	PORTS	General purpose function
CCU41.IN1B	I	PORTS	General purpose function
CCU41.IN1C	I	PORTS	General purpose function
CCU41.IN1D	I	ERU1.PDOUT0	General purpose function

Table 18-20 CCU41 - CC41 Pin Connections

Input/Output	I/O	Connected To	Description
CCU41.IN1E	I	HRPWM0.SR3	General purpose function
CCU41.IN1F	I	HRPWM0.QOUT1	General purpose function
CCU41.IN1G	I	HRPWM0.C1O	General purpose function
CCU41.IN1H	I	HRPWM0.QOUT0	General purpose function
CCU41.IN1I	I	SCU.GSC40	General purpose function
CCU41.IN1J	I	ERU1.PDOUT1	General purpose function
CCU41.IN1K	I	ERU1.IOUT1	General purpose function
CCU41.IN1L	I	0	General purpose function
CCU41.IN1M	I	CCU41.ST0	General purpose function
CCU41.IN1N	I	CCU41.ST1	General purpose function
CCU41.IN1O	I	CCU41.ST2	General purpose function
CCU41.IN1P	I	CCU41.ST3	General purpose function
CCU41.MCI1	I	0	Multi Channel pattern input
CCU41.OUT1	O	PORTS	Slice compare output
CCU41.GP10	O	NOT CONNECTED	Selected signal for event 0
CCU41.GP11	O	NOT CONNECTED	Selected signal for event 1
CCU41.GP12	O	NOT CONNECTED	Selected signal for event 2
CCU41.ST1	O	HRPWM0.BL1H	Slice status bit
CCU41.PS1	O	NOT CONNECTED	Multi channel pattern sync trigger: PM when counting UP (edge aligned) or OM when counting DOWN (center aligned)

Table 18-21 CCU41 - CC42 Pin Connections

Input/Output	I/O	Connected To	Description
CCU41.IN2A	I	PORTS	General purpose function
CCU41.IN2B	I	PORTS	General purpose function
CCU41.IN2C	I	PORTS	General purpose function
CCU41.IN2D	I	ERU1.PDOUT0	General purpose function
CCU41.IN2E	I	HRPWM0.SR3	General purpose function

Table 18-21 CCU41 - CC42 Pin Connections

Input/Output	I/O	Connected To	Description
CCU41.IN2F	I	HRPWM0.QOUT2	General purpose function
CCU41.IN2G	I	HRPWM0.C2O	General purpose function
CCU41.IN2H	I	HRPWM0.QOUT0	General purpose function
CCU41.IN2I	I	SCU.GSC40	General purpose function
CCU41.IN2J	I	ERU1.PDOUT2	General purpose function
CCU41.IN2K	I	ERU1.IOUT2	General purpose function
CCU41.IN2L	I	VADC.G0BFL1	General purpose function
CCU41.IN2M	I	CCU41.ST0	General purpose function
CCU41.IN2N	I	CCU41.ST1	General purpose function
CCU41.IN2O	I	CCU41.ST2	General purpose function
CCU41.IN2P	I	CCU41.ST3	General purpose function
CCU41.MCI2	I	0	Multi Channel pattern input
CCU41.OUT2	O	PORTS	Slice compare output
CCU41.GP20	O	NOT CONNECTED	Selected signal for event 0
CCU41.GP21	O	NOT CONNECTED	Selected signal for event 1
CCU41.GP22	O	NOT CONNECTED	Selected signal for event 2
CCU41.ST2	O	HRPWM0.BL2H	Slice status bit
CCU41.PS2	O	NOT CONNECTED	Multi channel pattern sync trigger: PM when counting UP (edge aligned) or OM when counting DOWN (center aligned)

Table 18-22 CCU41 - CC43 Pin Connections

Input/Output	I/O	Connected To	Description
CCU41.IN3A	I	PORTS	General purpose function
CCU41.IN3B	I	PORTS	General purpose function
CCU41.IN3C	I	PORTS	General purpose function
CCU41.IN3D	I	ERU1.PDOUT0	General purpose function
CCU41.IN3E	I	HRPWM0.SR3	General purpose function
CCU41.IN3F	I	HRPWM0.QOUT3	General purpose function

Table 18-22 CCU41 - CC43 Pin Connections

Input/Output	I/O	Connected To	Description
CCU41.IN3G	I	VADC.G1ARBCNT	General purpose function
CCU41.IN3H	I	HRPWM0.QOUT0	General purpose function
CCU41.IN3I	I	SCU.GSC40	General purpose function
CCU41.IN3J	I	ERU1.PDOUT3	General purpose function
CCU41.IN3K	I	ERU1.IOUT3	General purpose function
CCU41.IN3L	I	VADC.G0BFL2	General purpose function
CCU41.IN3M	I	CCU41.ST0	General purpose function
CCU41.IN3N	I	CCU41.ST1	General purpose function
CCU41.IN3O	I	CCU41.ST2	General purpose function
CCU41.IN3P	I	CCU41.ST3	General purpose function
CCU41.MCI3	I	0	Multi Channel pattern input
CCU41.OUT3	O	PORTS	Slice compare output
CCU41.GP30	O	NOT CONNECTED	Selected signal for event 0
CCU41.GP31	O	NOT CONNECTED	Selected signal for event 1
CCU41.GP32	O	NOT CONNECTED	Selected signal for event 2
CCU41.ST3	O	CCU81.IGBTA; VADC.G0REQGTB; VADC.G1REQGTB; VADC.BGREQGTB;	Slice status bit
CCU41.PS3	O	NOT CONNECTED	Multi channel pattern sync trigger: PM when counting UP (edge aligned) or OM when counting DOWN (center aligned)

19 Capture/Compare Unit 8 (CCU8)

The CCU8 peripheral functions play a major role in applications that need complex Pulse Width Modulation (PWM) signal generation, with complementary high side and low side switches, multi phase control or output parity checking. These functions in conjunction with a very flexible and programmable signal conditioning scheme, make the CCU8 the must have peripheral for state of the art motor control, multi phase and multi level power electronics systems.

The internal modularity of CCU8, translates into a software friendly system for fast code development and portability between applications.

Table 19-1 Abbreviations table

PWM	Pulse Width Modulation
CCU8x	Capture/Compare Unit 8 module instance x
CC8y	Capture/Compare Unit 8 Timer Slice instance y
ADC	Analog to Digital Converter
POSIF	Position Interface peripheral
SCU	System Control Unit
f_{ccu8}	CCU8 module clock frequency
f_{tclk}	CC8y timer clock frequency

Note: A small “y” or “x” letter in a register indicates an index

19.1 Overview

The CCU8 unit is comprised of four identical 16 bit Capture/Compare Timer slices, CC8y. Each Timer Slice can work in Compare or in Capture Mode. In Compare Mode, one has two dedicated compare channels that enable the generation of up to 4 PWM signals per Timer Slice (up to 16 PWM outputs per CCU8 unit), with dead time insertion to prevent short circuits in the switches. In Capture Mode a set of up to four capture registers is available.

Each CCU8 module has four service request lines that can be easily programmed to act as synchronized triggers between the PWM signal generation and an ADC conversion.

Straightforward timer slice concatenation is also possible, enabling up to 64 bit timing operations. This offers a flexible frequency measurement, frequency multiplication and pulse width modulation scheme.

A programmable function input selector for each timer slice, that offers up to nine functions, discards the need of complete resource mapping due to input ports availability.

A built-in link between the CCU8 and POSIF modules also enable a flexible digital motor control loop implementation, with direct coupling with Hall Sensors for Brushless DC Motor Control.

19.1.1 Features

CCU8 Module Features

Each CCU8 represents a combination of four Timer Slices, that can work independently in compare or capture mode. Each timer slice has 4 dedicated outputs for PWM signal generation.

All four CCU8 timer slices, CC8y, are identical in terms of available functions and operating modes. Avoiding this way the need of implementing different software routines, depending on which resource of CCU8 is used.

A built-in link between the four timer slices is also available, enabling this way a simplified timer concatenation and sequential operations.

General Features

- 16 bit timer cells
- programmable low pass filter for the inputs
- built-in timer concatenation
  - 32, 48 or 64 bit width
- shadow transfer for the period and compare channels
- four capture registers in capture mode
- programmable clock prescaler
- normal timer mode
- gated timer mode
- three counting schemes
  - center aligned
  - edge aligned
  - single shot
- PWM generation
- asymmetric PWM generation
- TRAP function
- dead time generation
- start/stop can be controlled by external events
- counting external events
- four dedicated service request lines per CCU8

Additional features

- external modulation function
- load controlled by external events

- dithering PWM
- floating point pre scaler
- output state override by an external event
- programmable output parity checker
- easy connection with POSIF unit for
  - hall sensor mode
  - rotary encoder mode
  - multi channel/multi phase control

CCU8 features vs. applications

On **Table 19-2** a summary of the major features of the CCU8 unit mapped with the most common applications.

Table 19-2 Applications summary

Feature	Applications
Four independent timer cells	Independent PWM generation: • Multiple buck/boost converter control (with independent frequencies) • Different mode of operation for each timer, increasing the resources optimization • Up to 2 H-Bridge control • multiple Zero Voltage Switch (ZVS) converter control with easy link to the ADC channels. • Multi Level Inverters
Two compare channels per Timer Slice	Linking between the two compare channels or linking between two Timer Slices: • Asymmetric PWM signal generation possibility decreases the number of current sensors • Linking between timer slices enable Phase Shift Full Bridge topologies control • Linking between slices enable N-Phase DC/DC converter control

Table 19-2 Applications summary (cont'd)

Feature	Applications
Two Dead Time Generators	Independent dead time values for rising and falling transitions and independent channel dead time counter: • Each channel can work stand alone with different dead time values. This enables the control of up to 2 Half-Bridges with different dead time values and the same frequency • Different dead time values for rising and falling transitions can be used to optimize the switching activity of the MOSFETs
Concatenated timer cells	Easy to configure timer extension up to 64 bit: • High dynamic trigger capturing • High dynamic signal measurement
Dithering PWM	Generating a fractional PWM frequency or duty cycle: • To avoid big steps on frequency or duty cycle adjustment in slow control loop applications • Increase the PWM signal resolution over time
Floating prescaler	Automated control signal measurement: • decrease SW activity for monitoring signals with high or unknown dynamics • generating a more than 16 bit timer for system control
Up to 9 functions via external signals for each timer	Flexible resource optimization: • The complete set of external functions is always available • Several arrangements can be done inside a CCU8, e.g., one timer working in capture mode and one working in compare
Output Parity Checker	Automated Mosfet signal monitoring: • parity checker can be used to monitor the output of the IGBTs and comparing them against the complete set of PWM outputs of CCU8. • Avoiding short circuits in a multi Mosfet system.

Table 19-2 Applications summary (cont'd)

Feature	Applications
4 dedicated service request lines	Specially developed for: • generating interrupts for the microprocessor • flexible connectivity between peripherals, e.g., ADC triggering.
Linking with POSIF	Flexible profiles for: • Rotary Encoder connection • Hall Sensor • Modulating the 4 timer outputs via SW

19.1.2 Block Diagram

Each CCU8 timer slice can operate independently from the other slices for all the available modes. Each timer slice contains a dedicated input selector for functions linked with external events and has 4 dedicated compare output signals, for PWM signal generation.

The built-in timer concatenation is only possible with adjacent slices, e.g. CC80/CC81. Combinations for slice concatenations like, CC80/CC82 or CC80/CC83 are not possible.

The individual service requests for each timer slice (four per slice) are multiplexed into four module service requests lines, **Figure 19-1**.

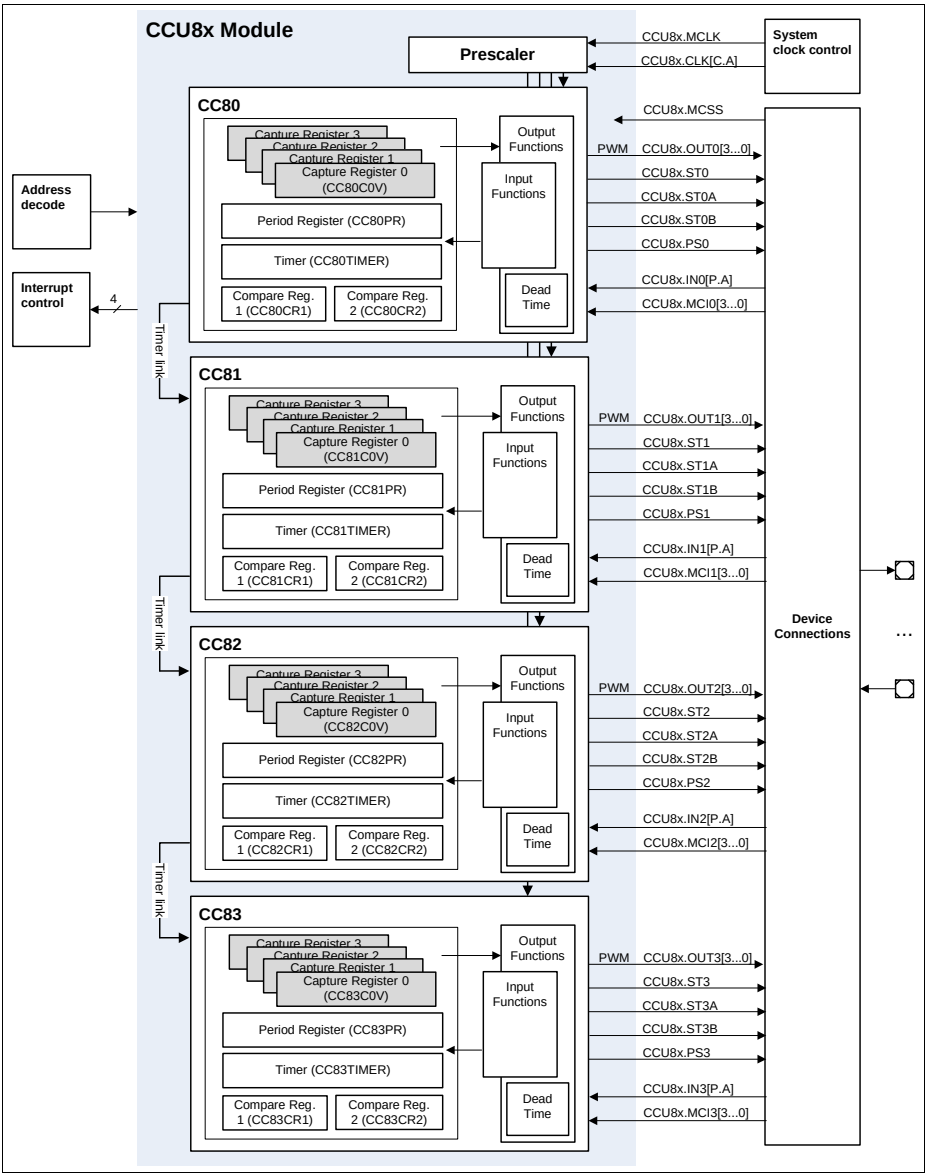


Figure 19-1 CCU8 block diagram

19.2 Functional Description

19.2.1 Overview

The input path of a CCU8 slice is comprised of a selector (**Section 19.2.2**) and a connection matrix unit (**Section 19.2.3**). The output path contains a service request control unit, a timer concatenation unit and two units that control directly the state of the output signal for each specific slice (for TRAP, dead time generation and modulation handling), see **Figure 19-2**.

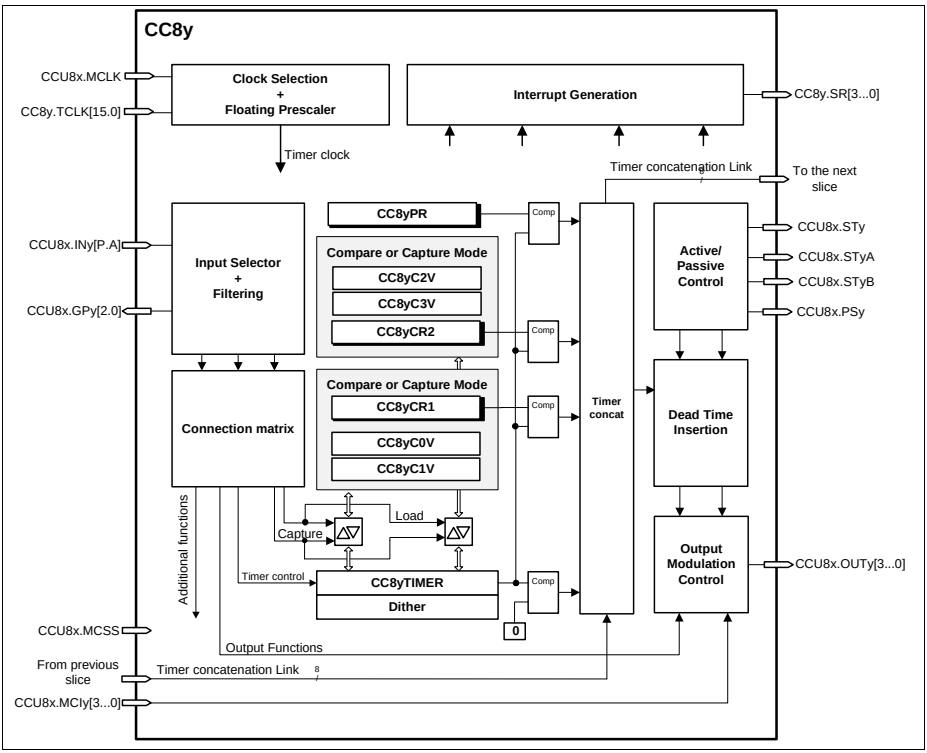


Figure 19-2 CCU8 slice block diagram

The timer core is built of 16 bit counter and one period register and two compare channels in compare mode, or four capture channels plus the period register in capture mode.

Individual timer clocks can be selected for each and every Timer Slice, enabling a very flexible resource organization inside each CCU8 module.

In compare mode the period sets the maximum counting value while the two compare registers are used to control the ACTIVE/PASSIVE state of the four dedicated comparison slice outputs.

Each CCU8 slice contains a dedicated timer link interface that is used to perform timer concatenation, up to 64 bits. This timer concatenation is controlled via a single bit field configuration.

Table 19-3 describes the inputs and outputs for each CCU8 Timer Slice.

Inputs and outputs that are not seen at the CCU8 module boundaries have a nomenclature of CC8y.<name>, whilst CCU8 module inputs and outputs are described as CCU8x.<signal_name>y (indicating the variable y the object slice).

Table 19-3 CCU8 slice pin description

Pin	I/O	Description
CCU8x.MCLK	I	Module clock
CC8y.TCLK		Clock from the pre scaler
CCU8x.INy[P:A]	I	Slice functional inputs (used to control the functionality throughout slice external events)
CCU8x.MCIy[3...0]	I	Multi Channel mode inputs
CCU8x.MCSS	I	Multi Channel shadow transfer trigger
CC8y.SR[3...0]	O	Slice service request lines
CCU8x.GPy[2...0]	O	Signals decoded from the input selector (used for the parity checker function)
CCU8x.STy	O	This signal can be the slice comparison status value of channel 1, channel 2 or a AND between both
CCU8x.STyA	O	Slice comparison status value of channel 1
CCU8x.STyB	O	Slice comparison status value of channel 2
CCU8x.PSy	O	Period match
CCU8x.OUTy[3...0]	O	Slice dedicated output pins

Note:

5. *The status bit outputs of the Kernel, CCU8x.STy, CCU8x.STyA and CCU8x.STyB are extended for one more kernel clock cycle.*
6. *The Service Request signals at the output of the kernel are extended for one more kernel clock cycle.*

7. *The maximum output signal frequency of the CCU8x.STy, CCU8x.STyA and CCU8x.STyA is module clock divided by 4.*

The slice timer, can count up or down depending on the selected operating mode. A direction flag contains the actual counting direction.

The timer is connected to three stand alone comparators, one for the period match and two for the compare match of each compare channel. The registers used for comparison match of both compare channels, can be programmed to serve as capture registers, enabling sequential capture capabilities on external events.

In normal edge aligned counting scheme, the counter is cleared to 0000_H each time it matches the period value defined in the period register. In center aligned mode, the counter direction changes from ‘up counting’ to ‘down counting’ after reaching the period value. Both period and compare registers have an aggregated shadow register, which enables the update of the PWM period and duty cycle on the fly.

A single shot mode is also available, where the counter stops after it reaches the value set in the period register.

The start and stop of the counter can be set/clear by software access or by a programmable input pin.

The dead time generator can be programmed with different values for the rising and falling edge of the output.

Functions like, load, counting direction (up/down), TRAP, output modulation can also be controlled with external events, see **Section 19.2.3**.

19.2.2 Input Selector

The first unit of the slice input path, is used to select from which are used to control the available external functions.

Inside this block the user also has the possibility to perform a low pass filtering of the signals and selecting the active edge(s) or level of the external event, see **Figure 19-3**.

The user has the possibility of selecting any of the CCU8x.INy[P:A] inputs has the source of an event.

At the output of this unit we have a user selection of three events, that were configured to be active at rising, falling or both edges, or level active. These selected events can then be mapped to several functions.

Notice that each decoded event contains two outputs, one edge active and one level active, due to the fact that some functions like counting, capture or load are edge sensitive events while, timer gating or up down counting selection are level active.

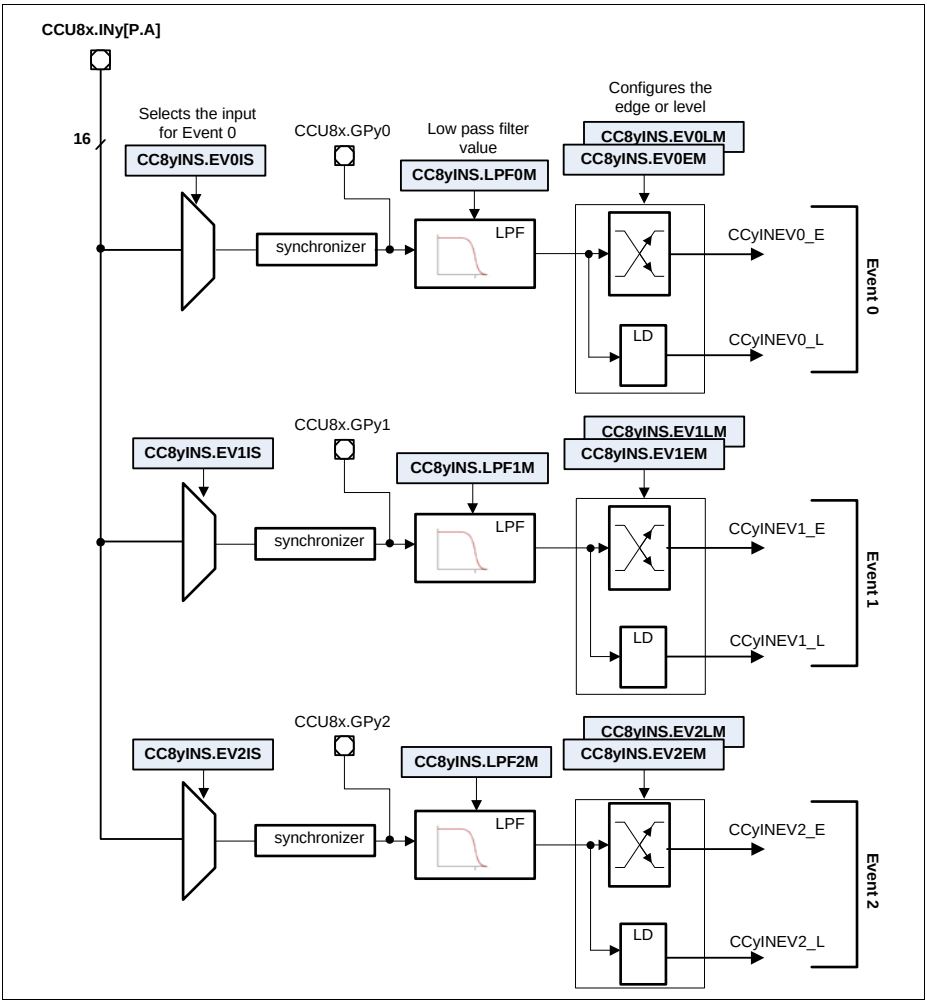


Figure 19-3 Slice input selector diagram

19.2.3 Connection Matrix

The connection matrix maps the events coming from the input selector to several user configured functions, **Figure 19-4**. The following functions can be enabled on the connection matrix:

Table 19-4 Connection matrix available functions

Function	Brief description	Map to figure Figure 19-4
Start	Edge signal to start the timer	CCystrt
Stop	Edge signal to stop the timer	CCystp
Count	Edge signal used for counting events	CCycnt
Up/down	Level signal used to select up or down counting direction	CCyupd
Capture 0	Edge signal that triggers a capture into the capture registers 0 and 1	CCycapt0
Capture 1	Edge signal that triggers a capture into the capture registers 2 and 3	CCycapt1
Gate	Level signal used to gate the timer clock	CCygate
Load	Edge signal that loads the timer with the value present at the compare register	CCyload
TRAP	Level signal used for fail-safe operation	CCytrap
Modulation	Level signal used to modulate/clear the output	CCymod
Status bit override	Status bit is going to be overridden with an input value	CCyoval for the value CCyoset for the trigger

Inside the connection matrix we also have a unit that performs the built-in timer concatenation. This concatenation enables a completely synchronized operation between the concatenated slices for timing operations and also for capture and load actions. The timer slice concatenation is done via the **CC8yCMC**.TCE bitfield. For a complete description of the concatenation function, please address **Section 19.2.10**.

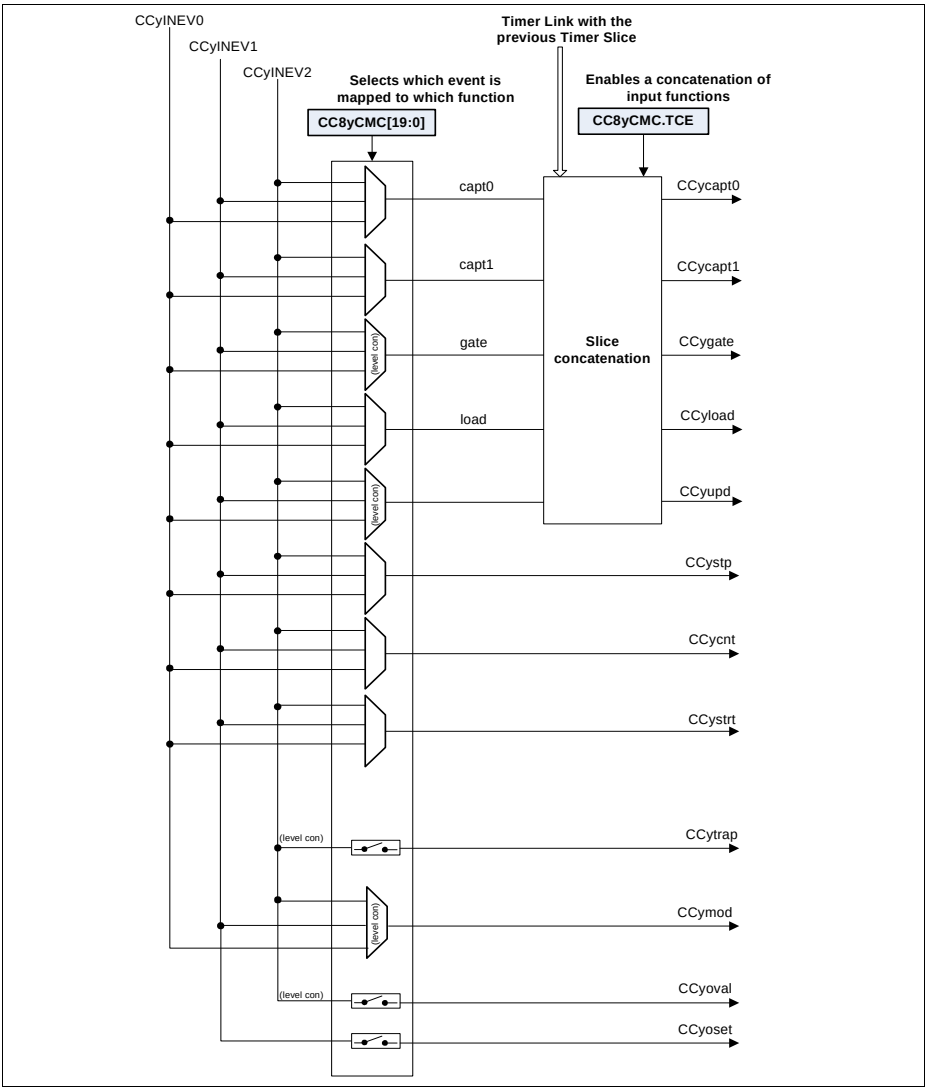


Figure 19-4 Slice connection matrix diagram

19.2.4 Start/Stop Control

Each slice contains a run bit register, that indicates the actual status of the timer, **CC8yTCST**.TRB. The start and stop of the timer can be done by software access or can be controlled directly by external events, see **Figure 19-5**.

Selecting an external signal that acts as a start trigger does not force the user to use an external stop trigger and vice versa.

Selecting the single shot mode, imposes that after the counter reaches the period value the run bit, **CC8yTCST**.TRB, is going to be cleared and therefore the timer is stopped.

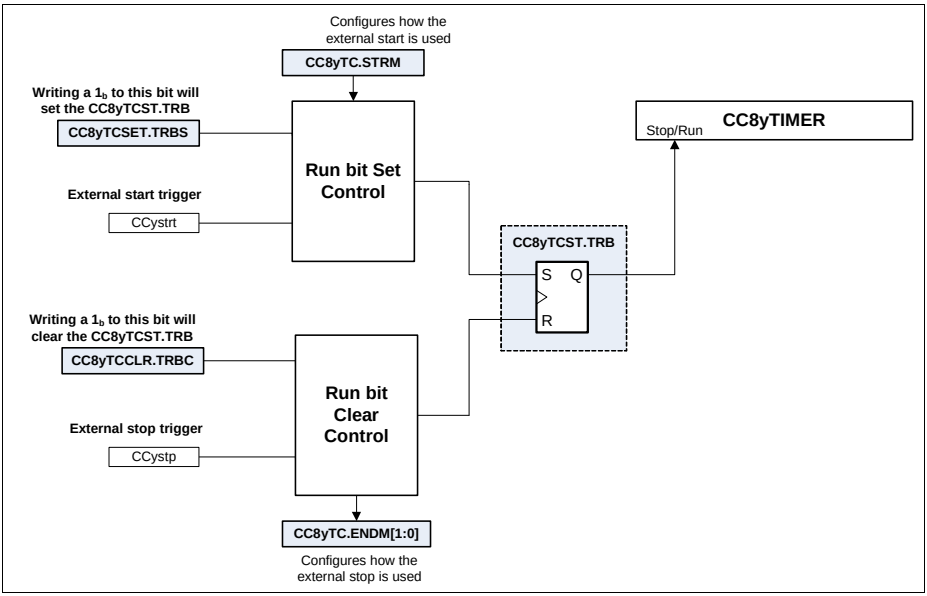


Figure 19-5 Timer start/stop control diagram

One can use the external stop signal to perform the following functions (configuration via **CC8yTC**.ENDM):

- Clear the run bit (stops the timer) - default
- Clear the timer (to 0000_H) but it does not clear the run bit (timer still running)
- Clear the timer and the run bit

One can use the external start to perform the following functions (configuration via **CC8yTC**.STRM):

- Start the timer (resume operation)
- Clear and starts the timer

The set (start the timer) of the timer run bit, always has priority over a clear (stop the timer).

To start multiple CCU8 timers at the same time/synchronously one should use a dedicated input as external start (see **Section 19.2.8.1** for a description how to configure an input as start function). This input should be connected to all the Timers that need to started synchronously (see **Section 19.8** for a complete list of module connections), **Figure 19-6**.

For starting the timers synchronously via software there is a dedicated input signal, controlled by the SCU (System Control Unit), that is connected to all the CCU8 timers. This signal should then be configured as an external start signal (see **Section 19.2.8.1**) and then the software must write $1_B/0_B$ (depending on the external start function configuration) to the specific bitfield of the CCUCON register (this register is described on the SCU chapter).

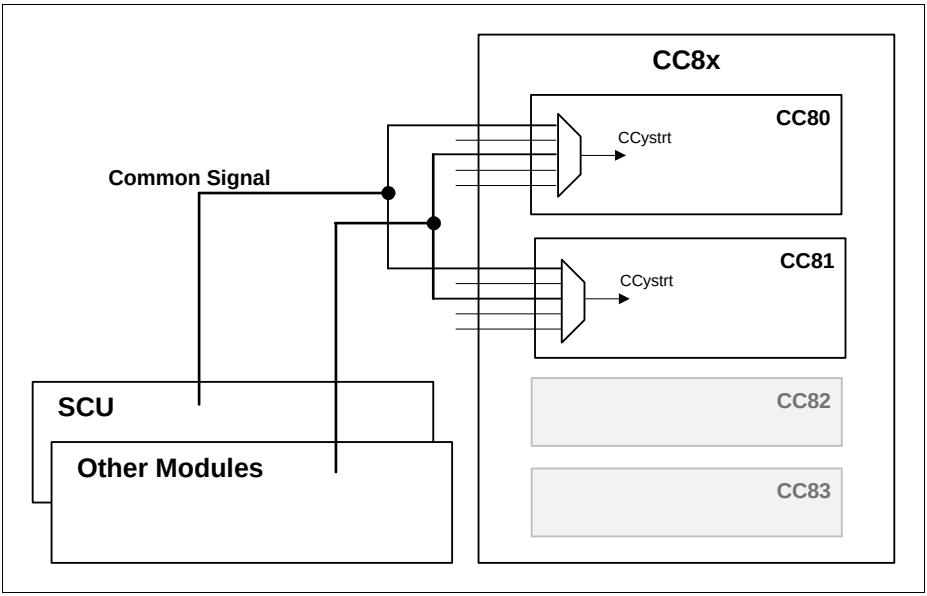


Figure 19-6 Start multiple timers synchronously

19.2.5 Counting Modes

Each CC8y timer can be programmed into three different counting schemes:

- Edge aligned (default)
- Center aligned
- Single shot (edge or center aligned)

These three counting schemes can be used as stand alone without the need of selecting any inputs as external event sources. Nevertheless it is also possible to control the counting operation via external events like, timer gating, counting trigger, external stop, external start, etc.

For all the counting modes, it is possible to update on the fly the values for the timer period and compare channel. This enables a cycle by cycle update of the PWM frequency and duty cycle.

Each compare channel of the CC8y Timer Slice has an associated Status Bit (**GCST**.CC8yST1 for compare channel 1 and **GCST**.CC8yST2 for compare channel 2), that indicates the active or passive state of the channel, **Figure 19-7**. The set and clear of the status bits and the respective PWM signal generation is dictated by the timer period, compare value and the current counting mode. See the different counting mode descriptions, **Section 19.2.5.3** to **Section 19.2.5.5** to understand how these bits are set and cleared.

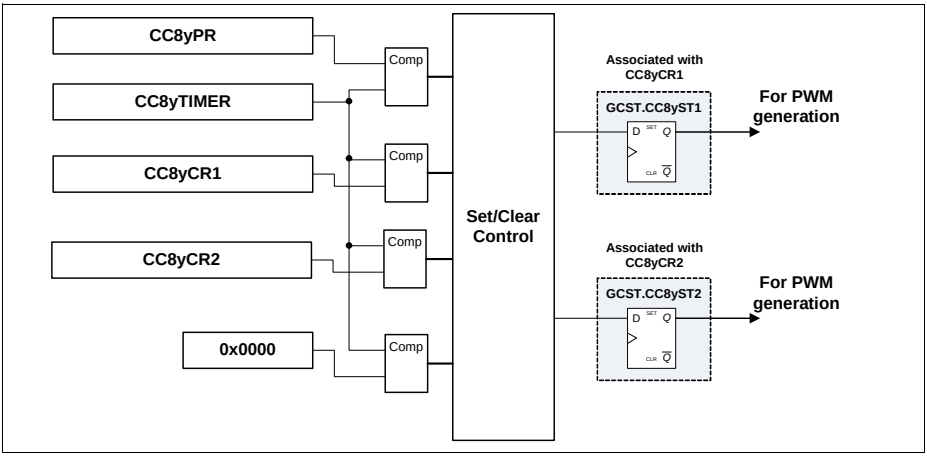


Figure 19-7 CC8y Status Bits

19.2.5.1 Calculating the PWM Period and Duty Cycle

The period of the timer is determined by the value in the period register, **CC8yPR** and by the timer mode.

The base for the PWM signal frequency and duty cycle, is always related to the clock frequency of the timer itself and not to the frequency of the module clock (due to the fact that the timer clock can be a scaled version of the module clock).

In Edge Aligned Mode, the timer period is:

T_{per}= <Period-Value> + 1; in f_{tclk} (19.1)

In Center Aligned Mode, the timer period is:

T_{per}= (<Period-Value> + 1) x 2; in f_{tclk} (19.2)

For each of these counting schemes, the duty cycle of generated PWM signal is dictated by the value programmed into the compare channel registers, **CC8yCR1** and **CC8yCR2**. Notice that one can have different duty cycle values for each of the compare channels.

In Edge Aligned and Center Aligned Mode, the PWM duty cycle is:

DC= 1 - <Compare-Value>/(<Period-Value> + 1) (19.3)

Both the period and compare registers, **CC8yPR**, **CC8yCR1** and **CC8yCR2** respectively, can be updated on the fly via software enabling a glitch free transition between different period and duty cycle values for the generated PWM signal, **Section 19.2.5.2**

19.2.5.2 Updating the Period and Duty Cycle

Standard Shadow Transfer Control

Each CCU8 timer slice provides an associated shadow register for the period and the two compare values. This facilitates a concurrent update by software for these three parameters, with the objective of modifying during run time the PWM signal period and duty cycle.

In addition to the shadow registers for the period and compare values, one also has available shadow registers for the floating prescaler, dither and passive level, **CC8yFPCS**, **CC8yDITS** and **CC8yPSL** respectively (please address **Section 19.2.13** and **Section 19.2.12** for a complete description of these functions).

The structure of the shadow registers can be seen in **Figure 19-8**.

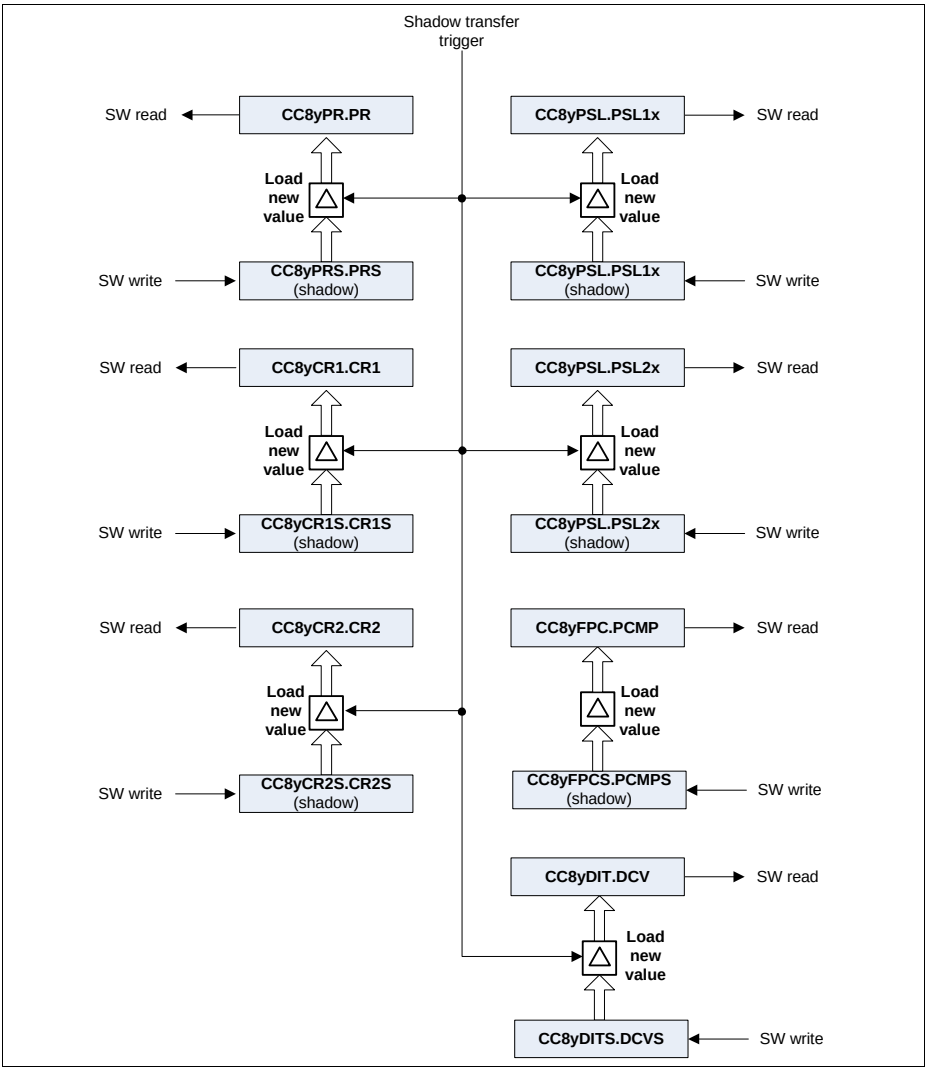


Figure 19-8 Shadow registers overview

The update of these registers can only be done by writing a new value into the associated shadow register and wait for a shadow transfer to occur.

Each group of shadow registers have an individual shadow transfer enable bit, **Figure 19-9**. The software must set this enable bit to 1_B, whenever an update of the

values is needed. These bits are automatically cleared by the hardware, whenever an update of the values if finished. Therefore every time that an update of the registers is needed the software must set again the specific bit(s).

Nevertheless it is also possible to clear the enable bit via software. This can be used in the case that an update of the values needs to be cancelled (after the enable bit has already been set).

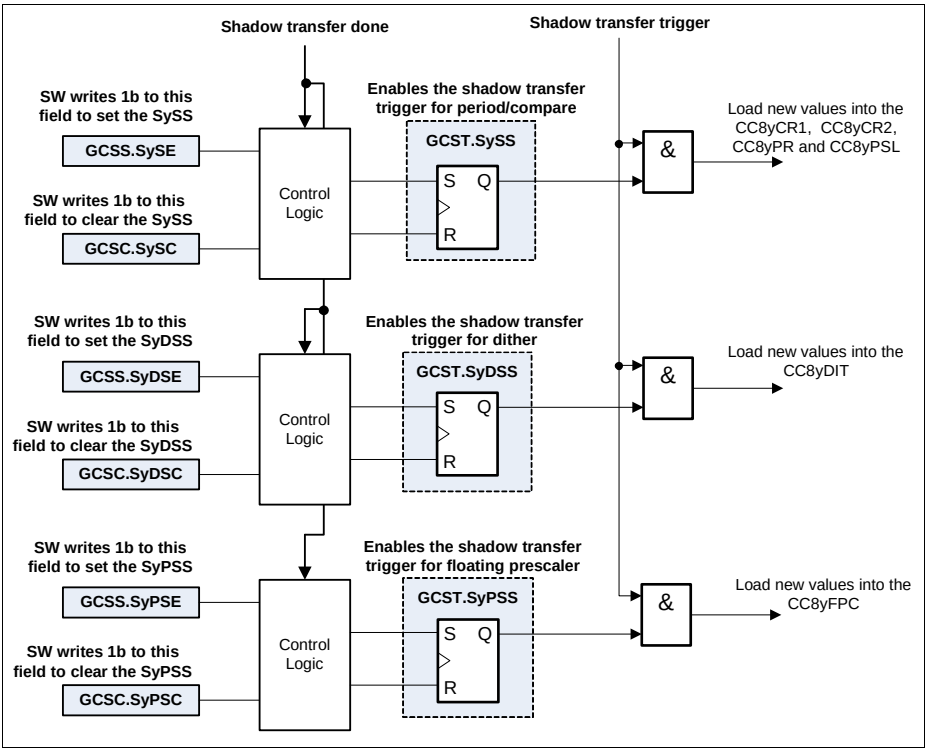


Figure 19-9 Shadow transfer enable logic

The shadow transfer operation is going to be done in the immediately next occurrence of a shadow transfer trigger, after the shadow transfer enable is set (**GCST**.SySS, **GCST**.SyDSS, **GCST**.SyPSS set to 1_B).

The occurrence of the shadow transfer trigger is imposed by the timer counting scheme (edge aligned or center aligned). Therefore the slots when the values are updated can be:

- in the next clock cycle after a Period Match while counting up
- in the next clock cycle after an One Match while counting down

- immediately, if the timer is stopped and the shadow transfer enable bit(s) is set

Figure 19-10 shows an example of the shadow transfer control when the timer slice has been configured into center aligned mode. For a complete description of all the timer slice counting modes, please address **Section 19.2.5.3**, **Section 19.2.5.4** and **Section 19.2.5.5**.

It is also possible to control in which slot the shadow transfer is done, via the STM field. This is only valid in Center Aligned Mode:

- **CC8ySTC**.STM = 00_B (default) - Shadow transfer is done at the Period Match and One match slot
- **CC8ySTC**.STM = 01_B - Shadow transfer is done only at the Period Match slot
- **CC8ySTC**.STM = 10_B - Shadow transfer is done only at the One Match slot

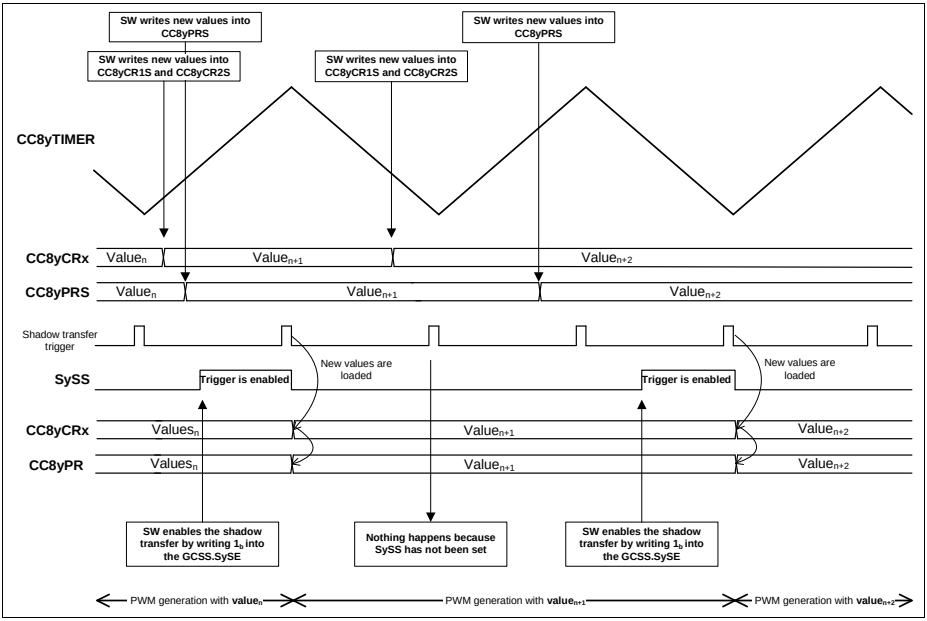


Figure 19-10 Shadow transfer timing example - center aligned mode

When using the CCU8 in conjunction with the POSIF to control the multi channel mode, it can be necessary in some cases, to perform the shadow transfers synchronously with the update of the multi channel pattern. To perform this action, each CCU8 contains a dedicated input that can be used to synchronize the two events, the CCU8x.MCSS.

This input, when enabled, is used to set the shadow transfer enable bitfields (**GCST**.SySS, **GCST**.SyDSS and **GCST**.SyPSS) of the specific slice. It is possible to select which slice is using this input to perform the synchronization via the **GCTRL**.MSEy

bit field. It is also possible to enable the usage of this signal for the three different shadow transfer signals: compare and period values, dither compare value and prescaler compare value. This can be configured on the **GCTRL**.MSDE field.

The structure for using the CCU8x.MCSS input signal can be seen in **Figure 19-11**. The usage of this signal is just an add on to the shadow transfer control and therefore all the previous described functions are still available.

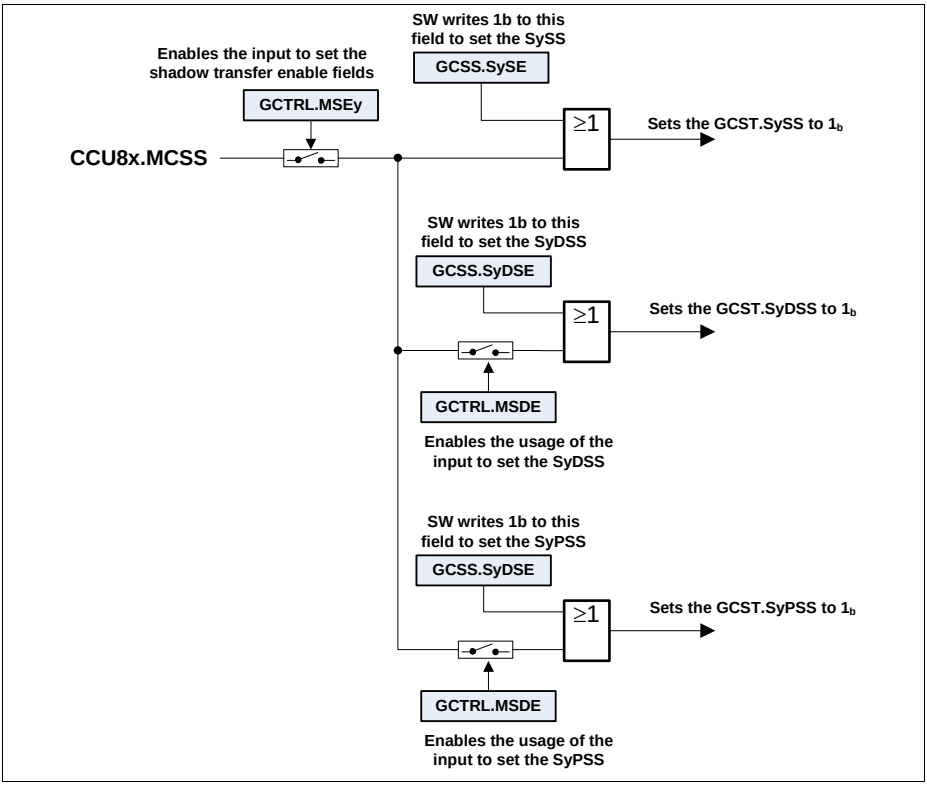


Figure 19-11 Usage of the CCU8x.MCSS input

Cascaded Shadow Transfer

It is possible to cascade the shadow transfer operation throughout the CCU8 timer slices. The specific shadow transfer of a timer slice is cascaded with the adjacent timer slices, **Figure 19-12**.

To enable the cascaded shadow transfer function, the bitfield **CC8ySTC**.CSE of the specific timer slice needs to be set to 1_B.

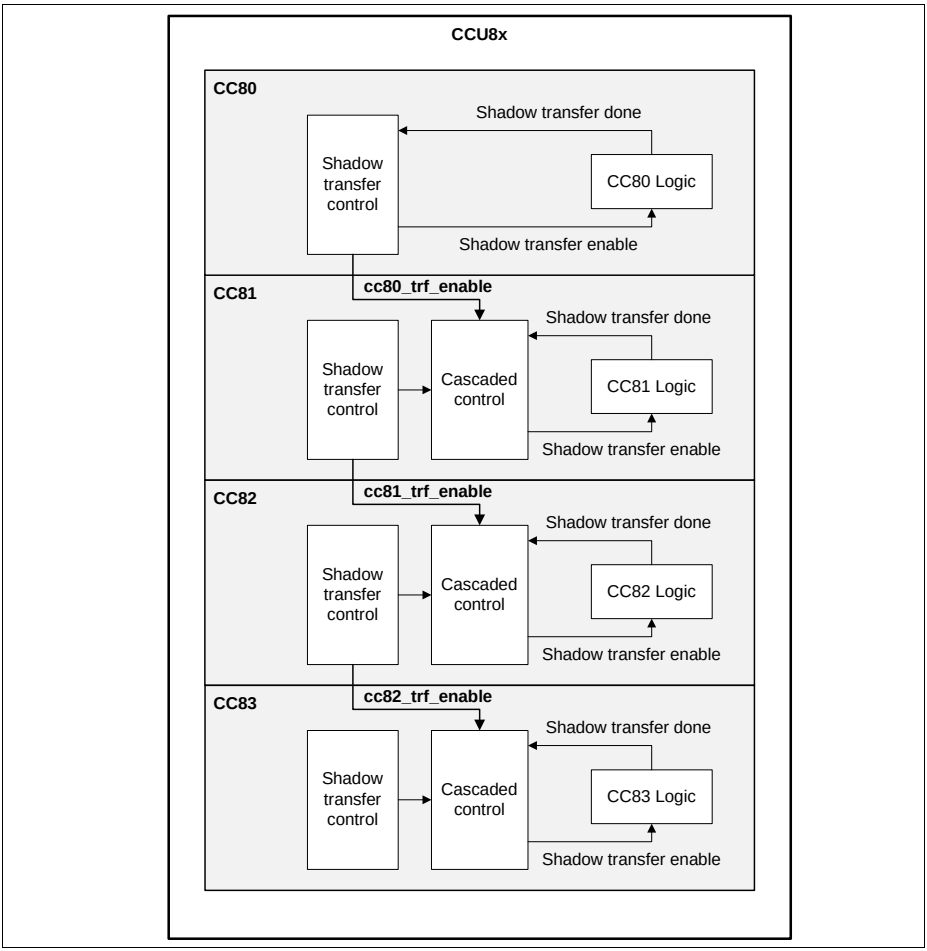


Figure 19-12 Cascaded shadow transfer linking

The shadow transfer enable bits, still need to be set via SW for each of the individual slices, **Figure 19-13**.

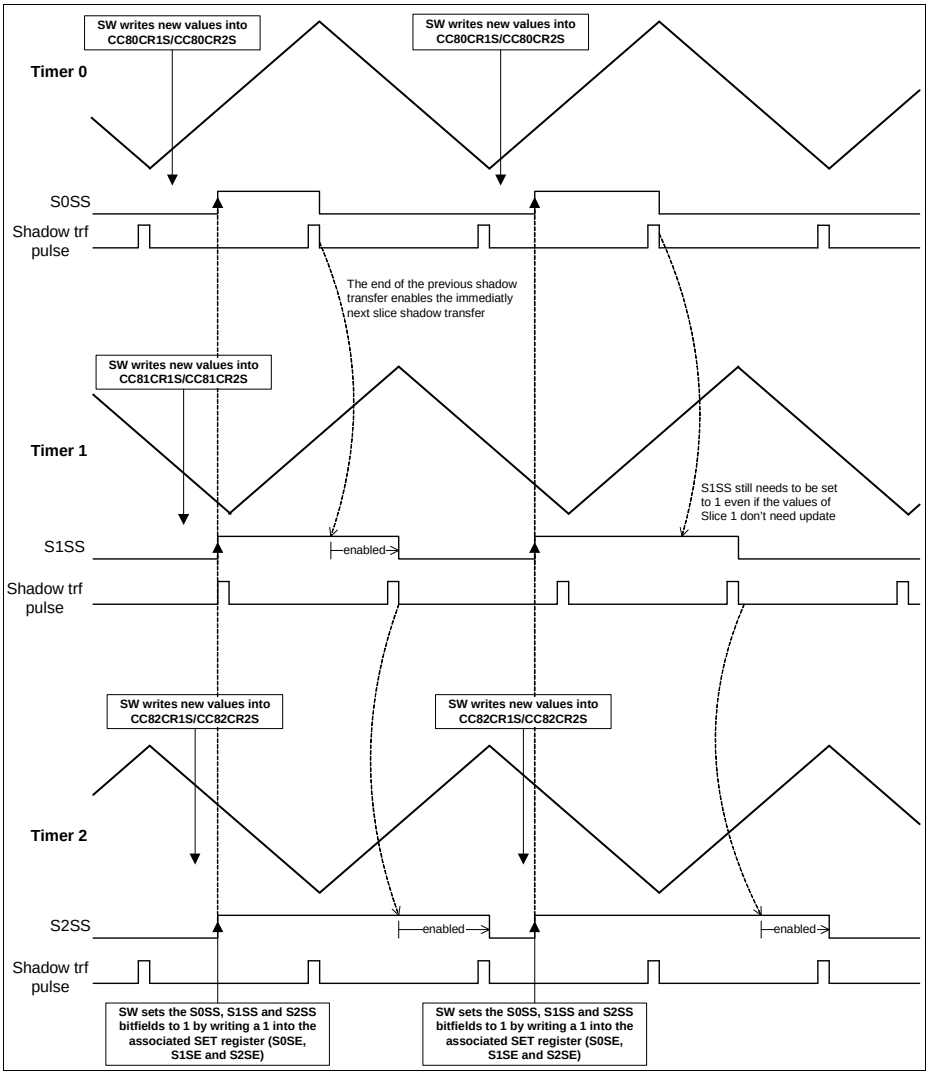


Figure 19-13 Cascade shadow transfer timing

Note: The shadow transfer enable bits, need to be set in all timer slices that are being used in the cascaded architecture, at the same time. The shadow transfer enable bits, also need to be set for all slices even if the shadow values of some slices were not updated.

19.2.5.3 Edge Aligned Mode

Edge aligned mode is the default counting scheme. In this mode, the timer is incremented until it matches the value programmed in the period register, **CC8yPR**. When period match is detected the timer is cleared to 0000_H and continues to be incremented.

In this mode, the value of the period register and compare registers are updated with the values written by software into the correspondent shadow register, every time that an overflow occurs (period match), see **Figure 19-14**.

In edge aligned mode, the status bit of the comparison (CC8ySTx) is set one clock cycle after the timer hits the value programmed into the compare register. The clear of the status bit is done one clock cycle after the timer reaches 0000_H.

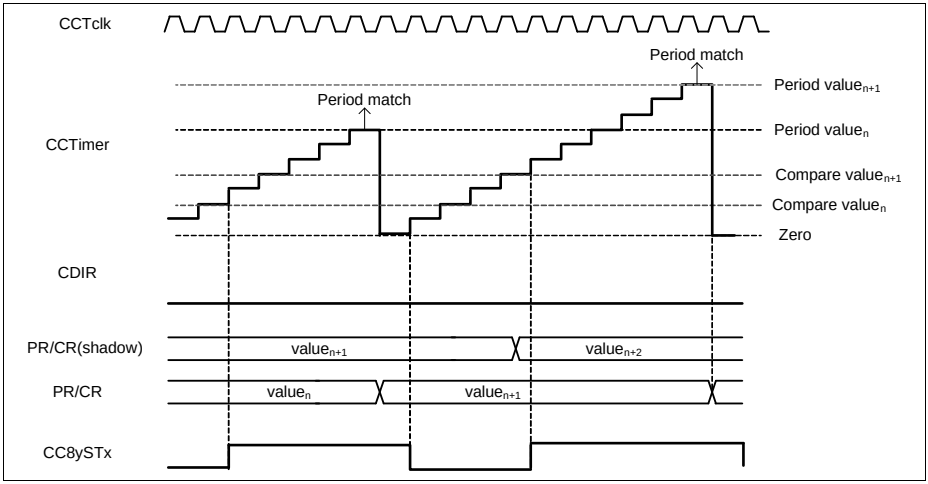


Figure 19-14 Edge aligned mode, CC8yTC.TCM = 0_B

19.2.5.4 Center Aligned Mode

In center aligned mode, the timer is counting up or down with respect to the following rules:

- The counter counts up while **CC8yTCST**.CDIR = 0_B and it counts down while **CC8yTCST**.CDIR = 1_B.
- Within the next clock cycle, the count direction is set to counting up (**CC8yTCST**.CDIR = 0_B) when the counter reaches 0001_H while counting down.
- Within the next clock cycle, the count direction is set to counting down (**CC8yTCST**.CDIR = 1_B), when the period match is detected while counting up.

The status bit (CC8ySTx) is always 1_B when the counter value is equal or greater than the compare value and 0_B otherwise.

While in edge aligned mode, the shadow transfer for compare and period registers is executed once per period. It is executed twice in center aligned mode as follows

- Within the next clock cycle after the counter reaches the period value, while counting up (**CC8yTCST**.CDIR = 0_B).
- Within the next clock cycle after the counter reaches 0001_H, while counting down (**CC8yTCST**.CDIR = 1_B).

*Note: Bit **CC8yTCST**.CDIR changes within the next timer clock after the one-match or the period-match, which means that the timer continues counting in the previous direction for one more cycle before changing the direction.*

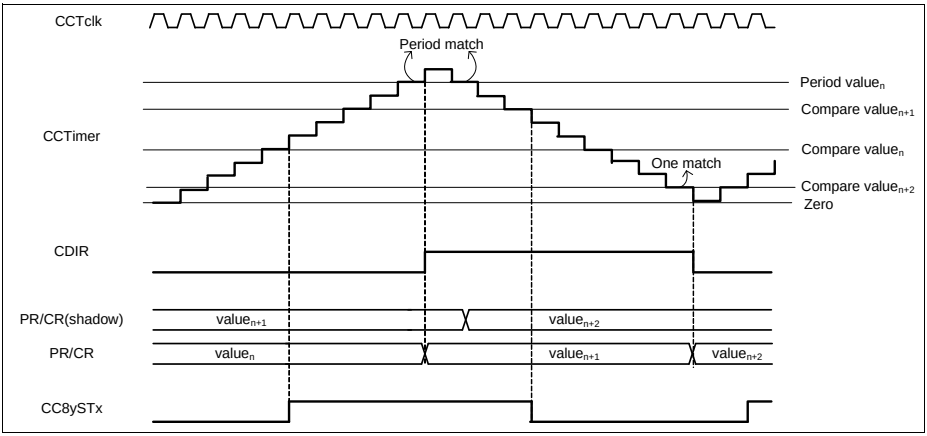


Figure 19-15 Center aligned mode, CC8yTC.TCM = 1_B

19.2.5.5 Single Shot Mode

In single shot mode, the timer is stopped after the current timer period is finished. This mode can be used with a center or edge aligned scheme.

In edge aligned mode, **Figure 19-16**, the timer is stopped when it is cleared to 0000_H after having reached the period value. In center aligned mode, **Figure 19-17**, the period is finished when the timer has counted down to 0000_H.

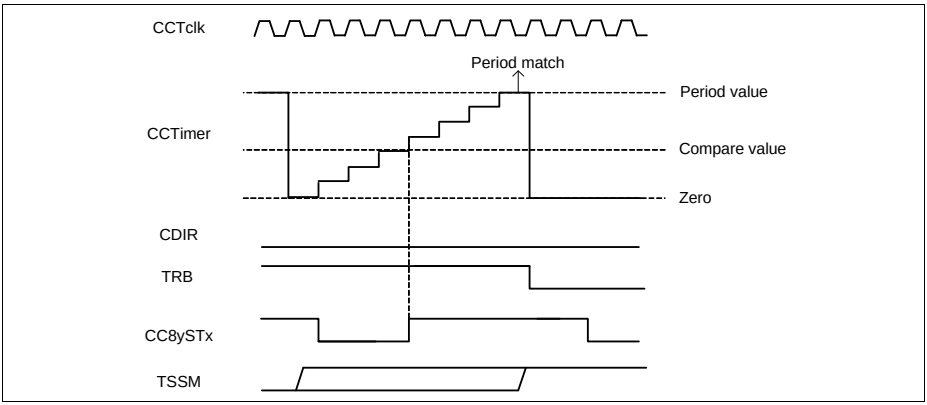


Figure 19-16 Single shot edge aligned - CC8yTC.TSSM = 1_B, CC8yTC.TCM = 0_B

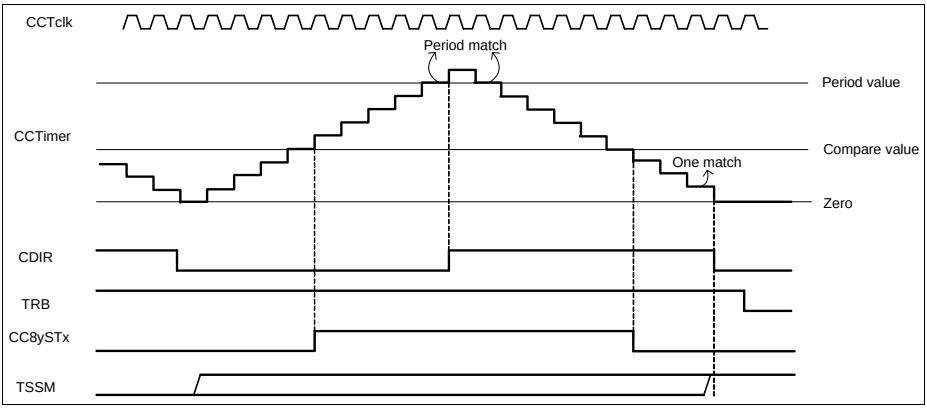


Figure 19-17 Single shot center aligned - CC8yTC.TSSM = 1_B, CC8yTC.TCM = 1_B

19.2.6 Active/Passive Rules

The general rules that set or clear the associated timer slice status bit, can be generalized independently of the timer counting mode.

The following events set the Status bit to Active:

- in the next f_{tclk} cycle after a compare match while counting up
- in the next f_{tclk} cycle after a zero match while counting down

The following events set the Status bit to Inactive:

- in the next f_{tclk} cycle after a zero match (and not compare match) while counting up
- in the next f_{tclk} cycle after a compare match while counting down

If external events are being used to control the timer operation, these rules are still applicable.

The status bit state can only be ‘override’ via software or by the external status bit override function, **Section 19.2.8.9**.

The software at any time can write a 1_B into the **GCSS**.SySTS bitfield, which will set the status bit **GCST**.CC8yST of the specific timer slice. By writing a 1_B into the **GCSC**.SySTC bitfield, the software imposes a clear of the specific status bit.

19.2.7 Compare Modes

Compare Channel Scheme

Each CCU8 slice has two compare channels and two dead time generators, one for each channel, see **Figure 19-18**. Each compare uses the information of the status bit, CC8ySTx, to generate two complementary outputs. All the outputs, CCU8x.OUTy0, CCU8x.OUTy1, CCU8x.OUTy2 and CCU8x.OUTy3, have a dedicated passive level control bit.

Each compare channel can work in an individual manner for both edge and center aligned modes. This means that two different complementary PWM signals can be generated by using the available compare channels. The PWM frequency is the same for both channels, but the duty cycle can be programmed independently for each channel.

It is also possible to select an asymmetric output scheme, by setting the field **CC8yCHC**.ASE = 1_B. In the asymmetric mode, the compare channels are grouped together to generate a single complementary PWM signal at the CCU8x.OUTy0 and CCU8x.OUTy1 pins.

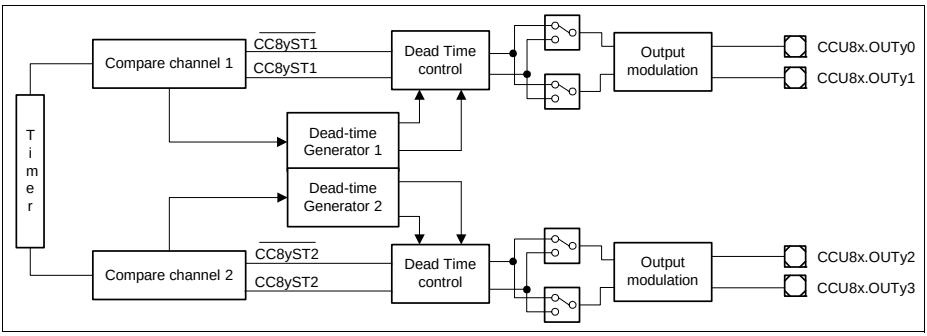


Figure 19-18 Compare channels diagram

Dead Time Generator

In most cases the switching behavior regarding the switch-on and switch-off times is not symmetrical, which can lead to a short circuit if the switch-on time is smaller than the switch-off time. To overcome this problem, each Timer Slice channel contains a dead time generator, which is able to delay the switching edges of the output signals, **Figure 19-19**.

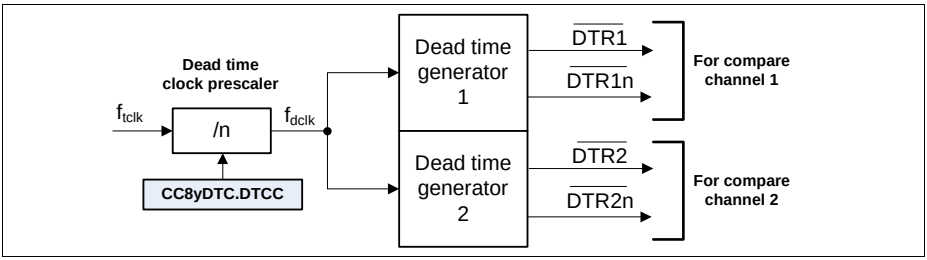


Figure 19-19 Dead Time scheme

Each dead time generator contains an eight bit counter with a different programmable reload value for rise and fall times. The dead time generators contain a programmable prescaler for the dead time counter clock, to enable large dead time insertion values, **Table 19-5**.

Table 19-5 Dead time prescaler values

CC8yDTC.DTCC[1:0]	Frequency
00_B	f_{tclk}
01_B	$f_{tclk}/2$
10_B	$f_{tclk}/4$
11_B	$f_{tclk}/8$

Any transition on the associated status bits, CC8ySTx, will trigger the start of the specific dead time generator, **Figure 19-20**.

When a SET (CC8ySTx passes from 0_B to 1_B) action for the CC8ySTx bit is detected, the dead time counter is reloaded with the value present on the **CC8yDC1R**.DT1R or **CC8yDC2R**.DT2R (depending on which channel we are addressing).

When a CLEAR action for the CC8ySTx bit is detected (CC8ySTx passes from 0_B to 1_B), the dead time counter is reloaded with the value present on the **CC8yDC1R**.DT1F or **CC8yDC2R**.DT2F (depending on which channel we are addressing).

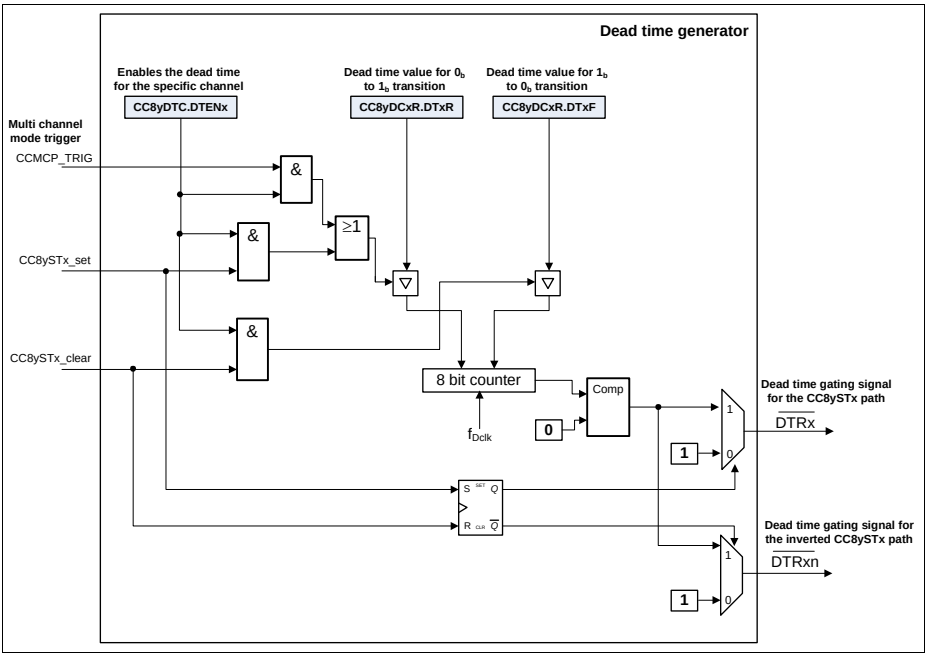


Figure 19-20 Dead Time generator scheme

Each dead time generator outputs two signals that are used to control the two complementary outputs (the CC8ySTx and the inverted CC8ySTx). The separation of the control signals enable a flexible enable/disable scheme inside of each compare channel, **Figure 19-21**. This means that the dead time generator can be enabled for one compare channel, but the dead time insertion can be discarded for one of the outputs.

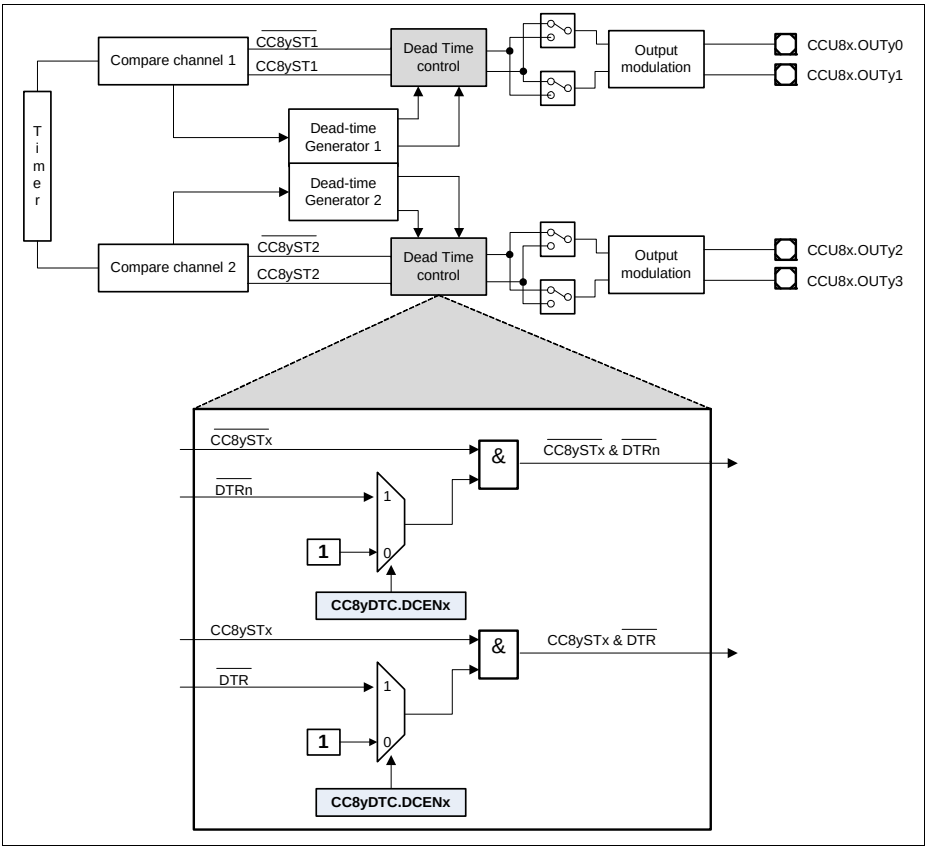


Figure 19-21 Dead Time control cell

When using the Multi Channel mode, **CC8yTC**.MCMEy = 1_B, there can be the scenario where the generated PWM signal has 100% duty cycle. This means that the respective status bit is always set and it is the Multi Channel pattern that is controlling the output modulation. In this case, we can have a transition from Inactive to Active state at the output, without having a transition on the specific status bit, creating a short on the switches due to the non existence of dead time insertion.

To overcome this possible short on the switches, a trigger from the multi channel control, CCMP_TRIG on **Figure 19-20**, is fed to the dead time generators. **Figure 19-22** shows the scheme for the generation of the CCMP_TRIG, where the signals, CCMCMx0 and CCMCMx1 represent the sampled multi channel pattern for a specific channel.

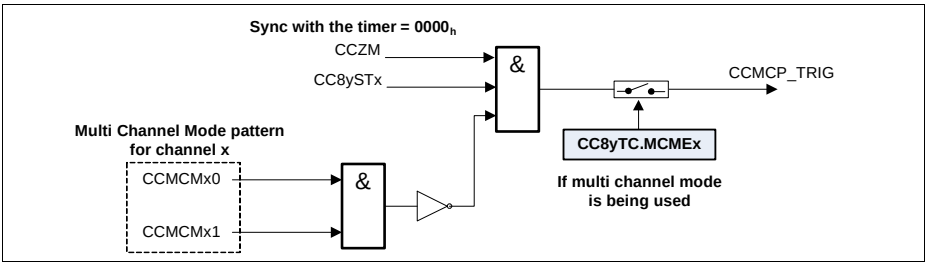


Figure 19-22 Dead Time trigger with the Multi Channel pattern

19.2.7.1 Edge Aligned Compare Modes

Standard Edge Aligned Mode

When the Timer Slice is programmed in edge aligned mode, the two channels can work independently, which means that the compare values can be programmed with different values (originating different duty cycles). In this scenario, each channel can output a pair of PWM signals used to control a high and low side switches, see **Figure 19-23**.

In this mode, for each channel the dead time for rise and fall transitions are controlled by the values programmed in the **CC8yDC1R**.DT1R and **CC8yDC2R**.DT2R, and **CC8yDC1R**.DT1F and **CC8yDC2R**.DT2F fields, respectively.

Figure 19-24 shows the timing diagrams for a specific slice when the compare values of each channel are different.

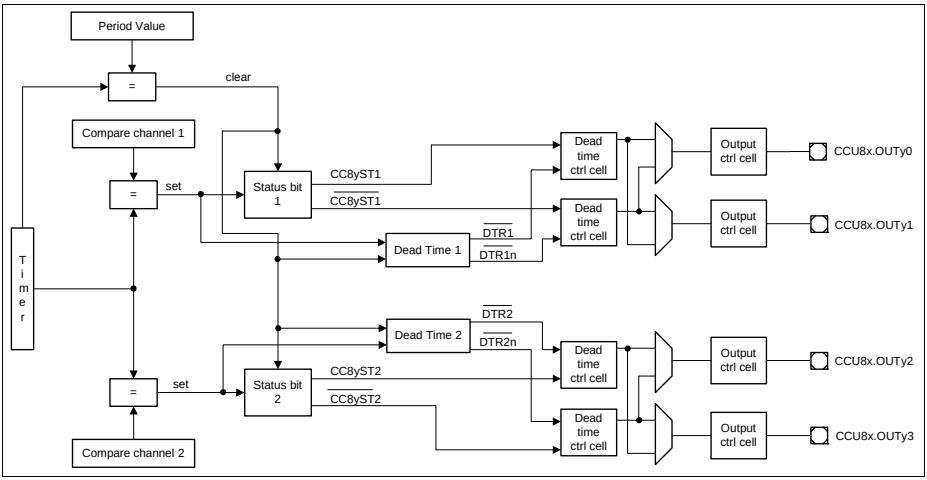


Figure 19-23 Edge Aligned with two independent channels scheme

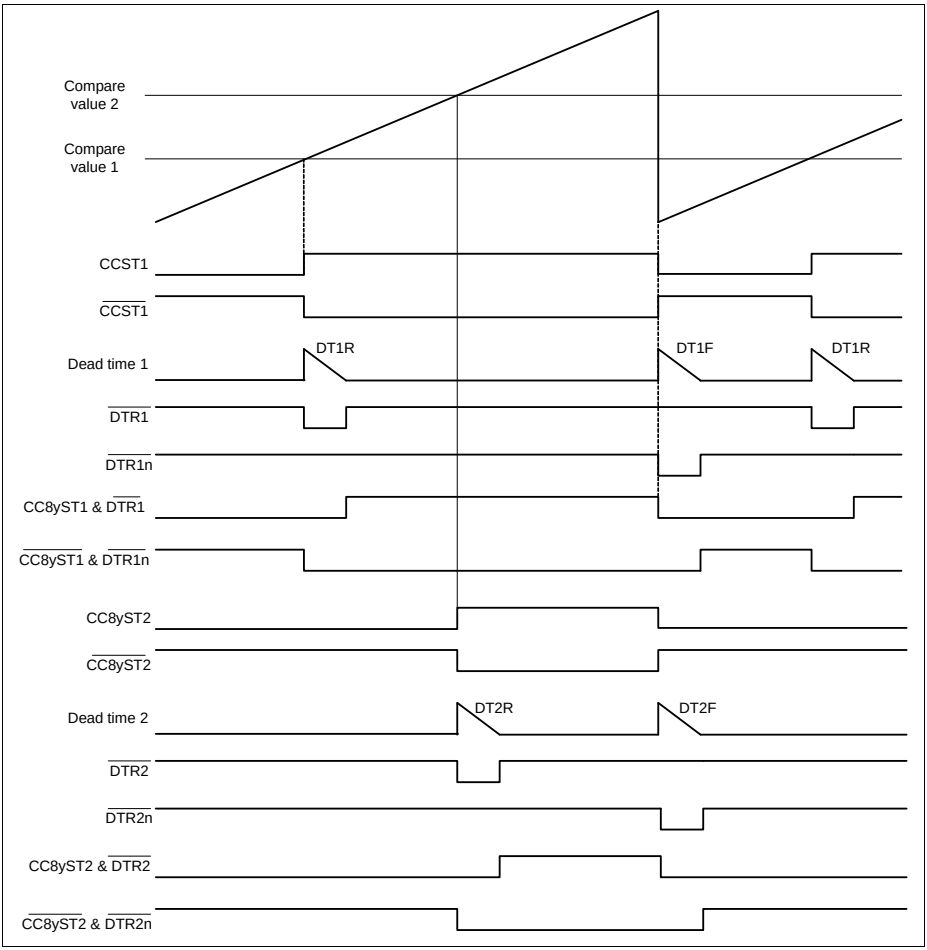


Figure 19-24 Edge Aligned - four outputs with dead time

Asymmetrical Edge Aligned Mode

There is also the possibility of using the two channels combined to generate an asymmetric PWM output. This mode is selected by setting the field **CC8yCHC**.ASE = 1_B.

In this mode, the compare channel 2 is disabled and therefore the outputs linked with this path are always in the passive state.

The status bit of the compare channel 1 is set when a compare match with the compare value 1 (field **CC8yCR1**.CR1) occurs and is cleared when a compare match with the compare value 2 (field **CC8yCR2**.CR2) occurs, see **Figure 19-25**.

When the **CC8yCR2**.CR2 is programmed with a value smaller than the one present in **CC8yCR1**.CR1, the CCST1 bit is always 0_B.

The dead time values for the rising and falling transitions are controlled by the fields **CC8yDC1R**.DT1R and **CC8yDC1R**.DT1F, respectively.

Figure 19-26 and **Figure 19-27** show the timing diagram for the Edge Aligned mode when the asymmetric scheme is active.

Note: When an external signal is used to control the counting direction, the asymmetric mode cannot be enabled.

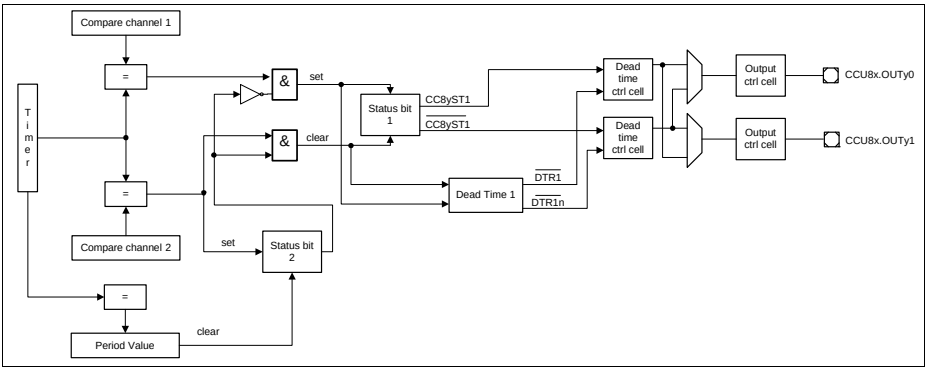


Figure 19-25 Edge Aligned with combined channels scheme

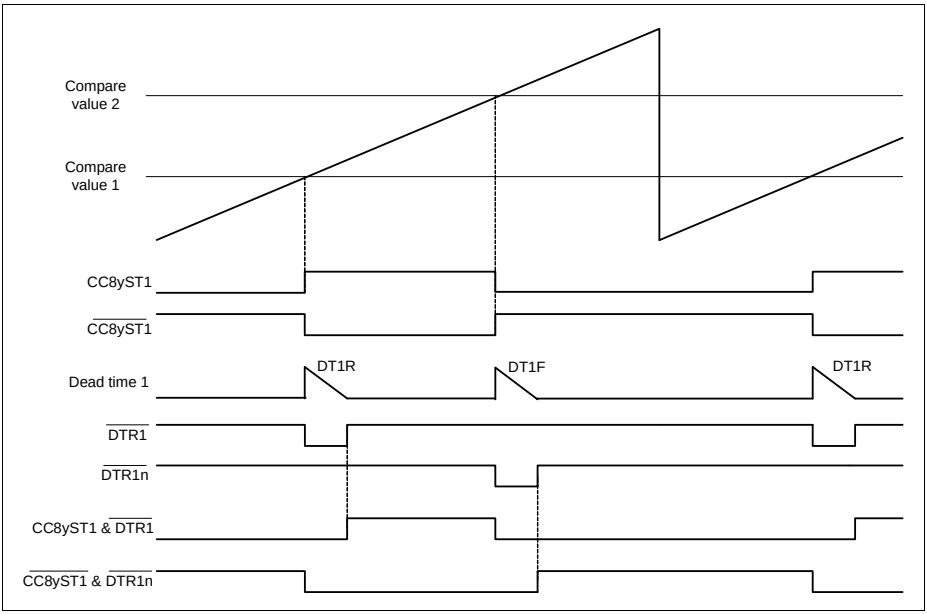


Figure 19-26 Edge Aligned - Asymmetric PWM timing, CC8yCR1.CR1 < CC8yCR2.CR2

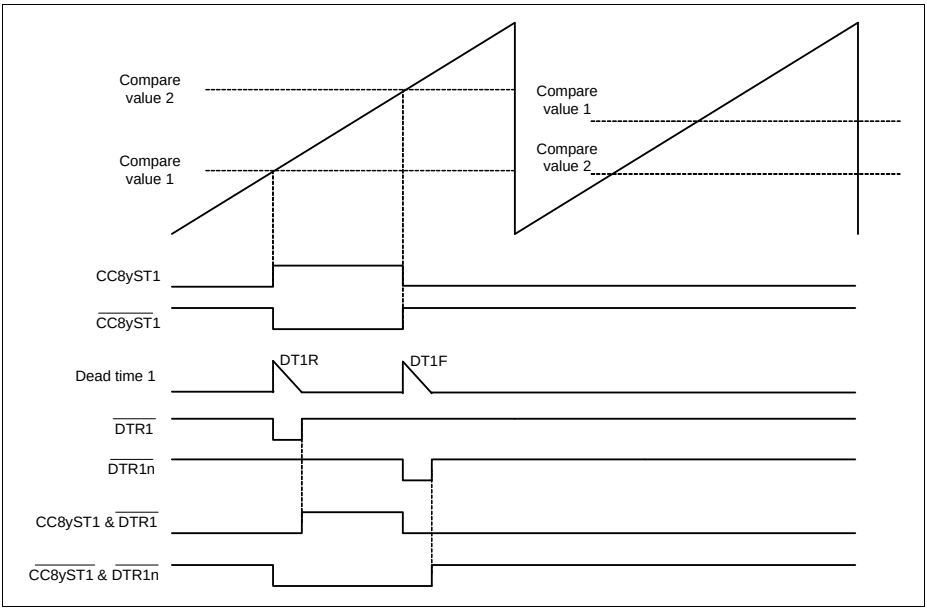


Figure 19-27 Edge Aligned - Asymmetric PWM timing, CC8yCR1.CR1 > CC8yCR2.CR2

19.2.7.2 Center Aligned Compare Modes

Standard Center Aligned Mode

In center aligned mode, like in edge aligned, it is possible to use the two compare channels independently. In this mode, each channel can generate a pair of PWM complementary signals with different duty cycle values, controlled via the **CC8yCR1** for channel 1 and **CC8yCR2** for channel 2.

For the dead time insertion, each channel as a pair of programmable values for the rise and fall transitions: **CC8yDC1R**.DT1R and **CC8yDC1R**.DT1F for channel 1; **CC8yDC2R**.DT2R and **CC8yDC2R**.DT2F for channel 2.

The major difference between the center and the edge aligned mode is directly linked to the set/clear logic of the status bit, see **Section 19.2.5**.

Figure 19-28 shows the scheme for both channels for this operating mode and **Figure 19-29** shows the timing diagrams for a specific channel.

Note: When an external signal is used to control the counting direction, the counting scheme is always edge aligned.

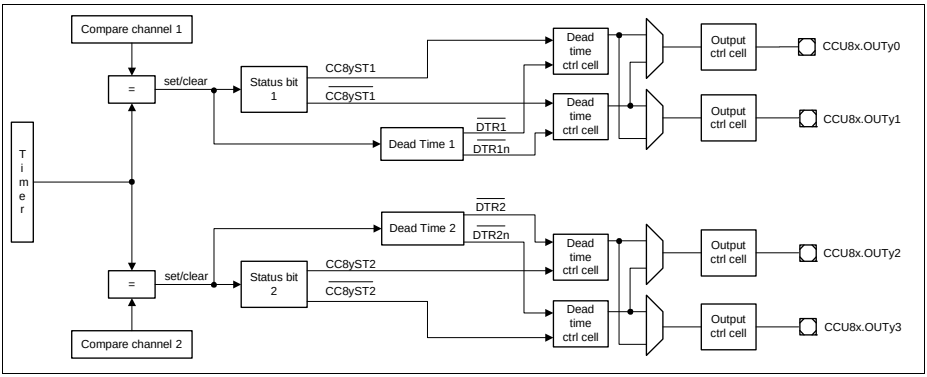


Figure 19-28 Center Aligned with two independent channels scheme

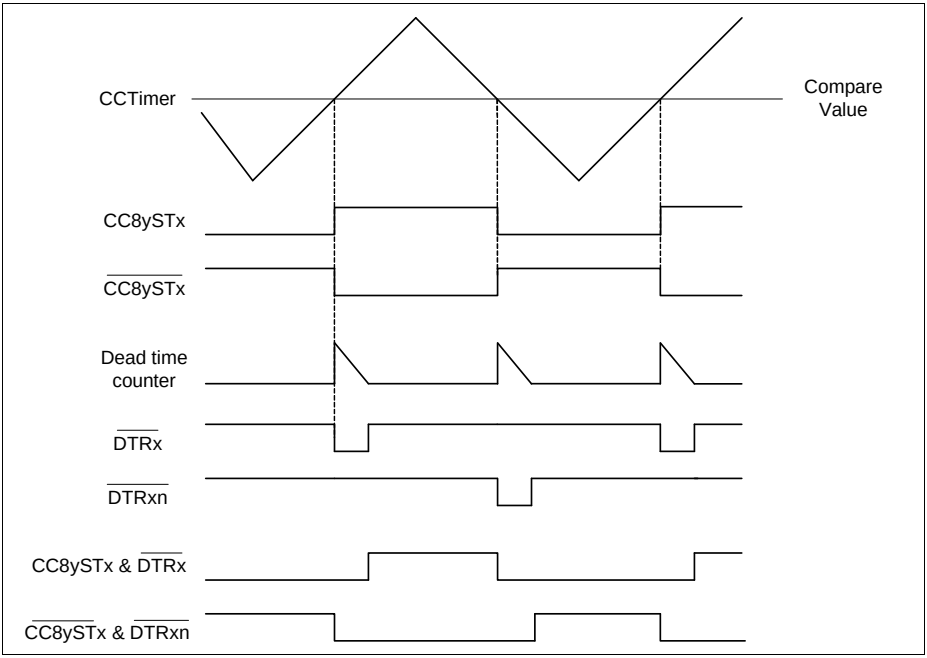


Figure 19-29 Center aligned - Independent channel with dead time

Asymmetrical Center Aligned Mode

The asymmetric mode is enabled in center aligned by setting the field **CC8yCHC**.ASE to 1_B.

In this mode, like in Edge Aligned, the outputs linked with the compare channel 2 are set to their passive levels.

The status bit, CC8yST1, is set when a compare match of channel 1 occurs while counting up, and is cleared when a compare match of channel 2 occurs while counting down, see **Figure 19-30**.

The dead time rise and fall times are controlled by the values programmed into the fields, **CC8yDC1R**.DT1R and **CC8yDC1R**.DT1F, respectively.

Figure 19-31 shows the timing diagram for the asymmetric mode. Notice that even in asymmetric mode the dead time can be disabled in each of the outputs independently.

Note: When an external signal is used to control the counting direction, the asymmetric mode cannot be enabled.

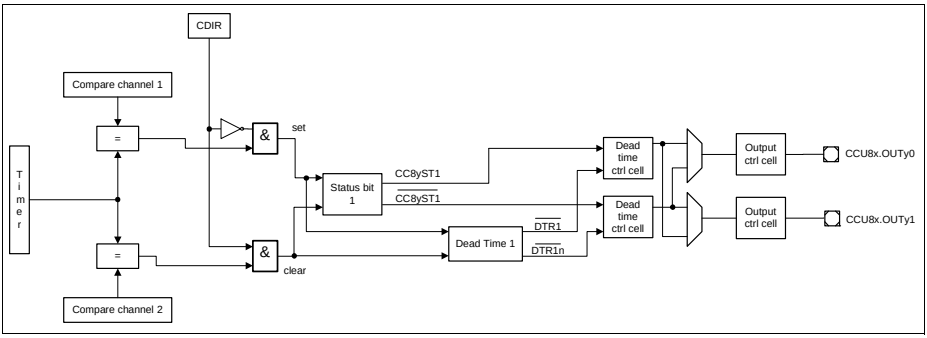


Figure 19-30 Center Aligned Asymmetric mode scheme

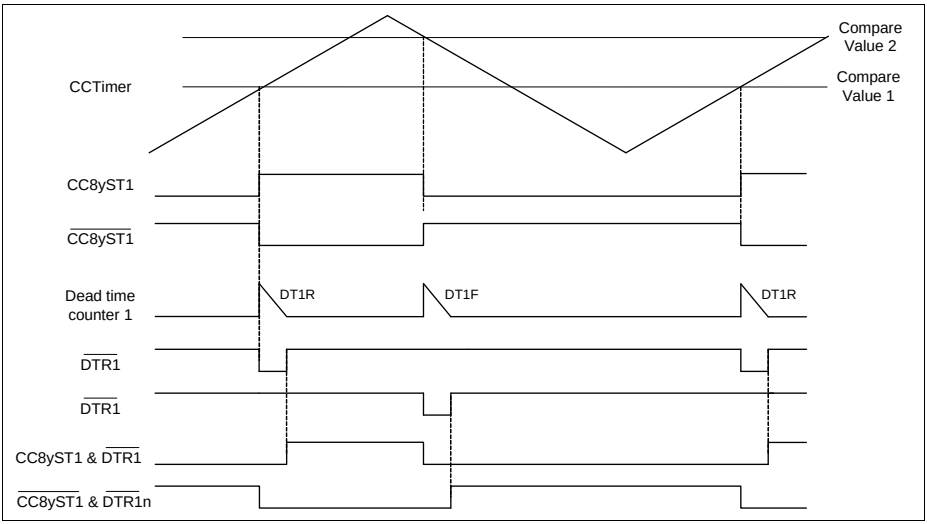


Figure 19-31 Asymmetric Center aligned mode with dead time

19.2.8 External Events Control

Each CCU8 slice has the possibility of using up to three different input events, see **Section 19.2.2**. These three events can then be mapped to Timer Slice functions (the full set of available functions is described at **Section 19.2.3**).

These events can be mapped to any of the CCU8x.INy[P...A] inputs and there isn't any imposition that an event cannot be used to perform several functions or, that an input cannot be mapped to several events (e.g. input X triggers event 0 with rising edge and triggers event 1 with the falling edge).

19.2.8.1 External Start/Stop

To select an external start function, one should map one of the events (output of the input selector) to a specific input signal, by setting the required value in the **CC8yINS**.EVxIS register and indicating the active edge of the signal on the **CC8yINS**.EVxEM register.

This event should be then mapped to the start or stop functionality by setting the **CC8yCMC**.STRTS (for the start) or the **CC8yTC**.ENDM (for the stop) with the proper value.

The same procedure is applicable to the stop functionality.

Notice that both start and stop functions are edge and not level active and therefore the active/passive configuration is set only by the **CC8yINS**.EVxEM.

The external stop by default just clears the run bit (**CC8yTCST**.TRB), while the start functions does the opposite. Nevertheless one can select an extended subset of functions for the external start and stop. This subset is controlled by the registers **CC8yTC**.ENDM (for the stop) and **CC8yTC**.STRM (for the start).

For the start subset (**CC8yTC**.STRM):

- sets the run bit/starts the timer (resume operation)
- clears the timer, sets the run bit/starts the timer (flush and start)

For the stop subset (**CC8yTC**.ENDM):

- clears the run/stops the timer (stop)
- clears the timer (flush)
- clears the timer, clears the run bit/stops the timer (flush and stop)

If in conjunction with an external start/stop (configured also/only as flush) and external up/down signal is used, during the flush operation the timer is going to be set to 0000_H if the actual counting direction is up or set with the value of the period register if the counting direction is down.

Figure 19-32 to **Figure 19-35** shows the usage of two signals to perform the start/stop functions in all the previously mentioned subsets. External Signal(1) acts as an active HIGH start signal, while External Signal(2) is used as an active HIGH stop function.

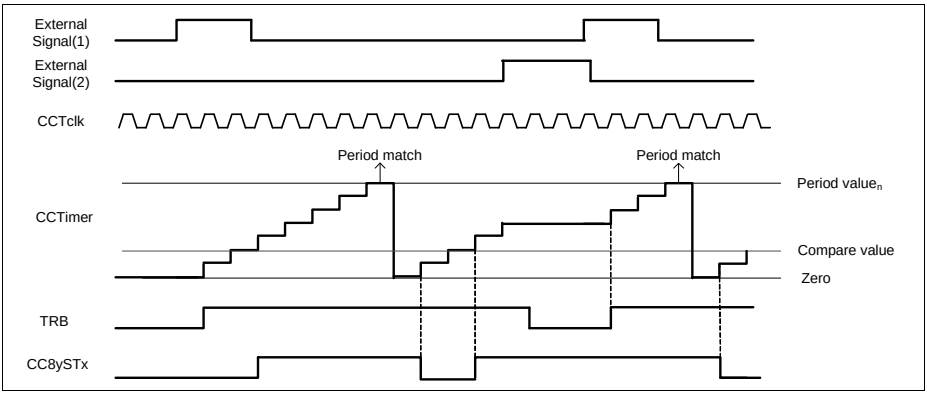


Figure 19-32 Start (as start)/ stop (as stop) - CC8yTC.STRM = 0_B, CC8yTC.ENDM = 00_B

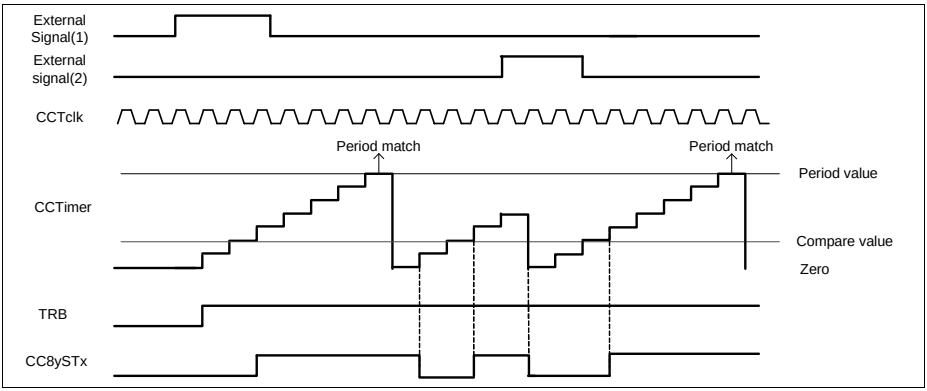


Figure 19-33 Start (as start)/ stop (as flush) - CC8yTC.STRM = 0_B, CC8yTC.ENDM = 01_B

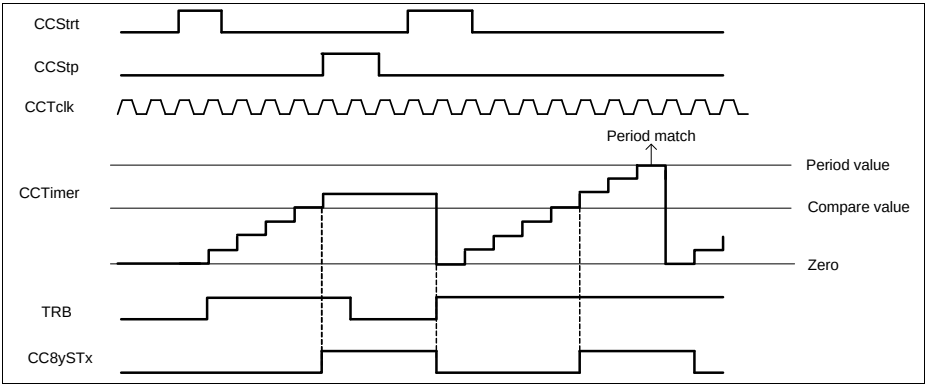


Figure 19-34 Start (as flush and start)/ stop (as stop) - CC8yTC.STRM = 1_B, CC8yTC.ENDM = 00_B

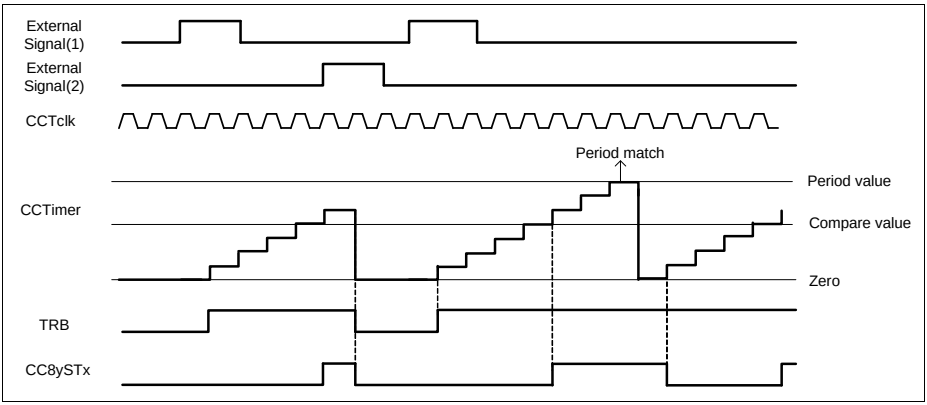


Figure 19-35 Start (as start)/ stop (as flush and stop) - CC8yTC.STRM = 0_B, CC8yTC.ENDM = 10_B

19.2.8.2 External Counting Direction

There is the possibility of selecting an input signal to act as counting up/counting down control.

To select an external up/down control, one should map one of the events (output of the input selector) to a specific input signal, by setting the required value in the **CC8yINS**.EVxIS register and indicating the active level of the signal on the **CC8yINS**.EVxLM register. This event should be then mapped to the up/down functionality by setting the **CC8yCMC**.UDS with the proper value.

Notice that the up/down function is level active and therefore the active/passive configuration is set only by the **CC8yINS**.EVxLM.

The status bit of the slice (CCSTx) is always set when the timer value is equal or greater than the value stored in the compare register, see **Section 19.2.6**.

The update of the period and compare register values is done when:

- with the next clock after a period match, while counting up (**CC8yTCST**.CDIR = 0_B)
- with the next clock after a one match, while counting down (**CC8yTCST**.CDIR = 1_B)

The value of the **CC8yTCST**.CDIR register is updated accordingly with the changes on the decoded event. The Up/Down direction is always understood as **CC8yTCST**.CDIR = 1 when counting down and **CC8yTCST**.CDIR = 0_B when counting up. Using an external signal to perform the up/down counting function and configuring the event as active HIGH means that the timer is counting up when the signal is HIGH and counting down when LOW.

Figure 19-36 shows an external signal being used to control the counting direction of the time. This signal was selected as active HIGH, which means that the timer is counting down while the signal is HIGH and counting up when the signal is LOW.

*Note: For a signal that should impose an increment when LOW and a decrement when HIGH, the user needs to set the **CC8yINS**.EVxLM = 0_B. When the operation is switched, then the user should set **CC8yINS**.EVxLM = 1_B.*

Note: Using an external counting direction control, sets the slice in edge aligned mode.

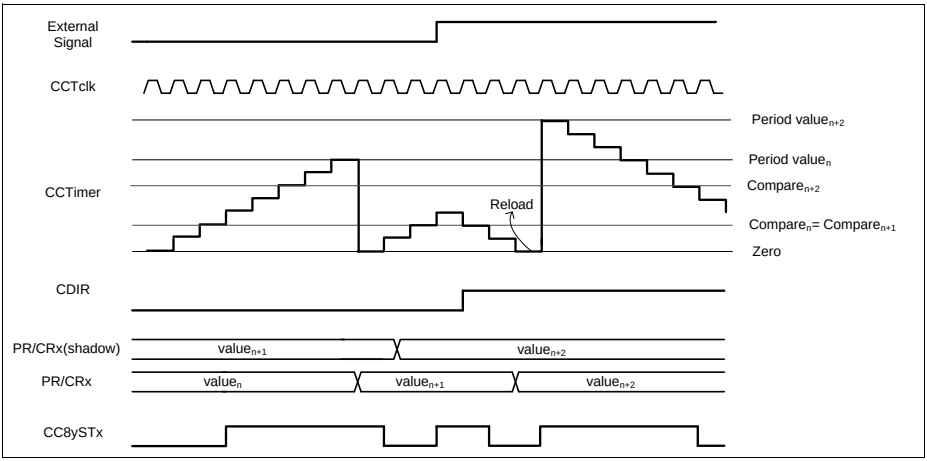


Figure 19-36 External counting direction

19.2.8.3 External Gating Signal

For pulse measurement, the user has the possibility of selecting an input signal that operates as counting gating.

To select an external gating control, one should map one of the events (output of the input selector) to a specific input signal, by setting the required value in the **CC8yINS**.EVxIS register and indicating the active level of the signal on the **CC8yINS**.EVxLM register. This event should be then mapped to the gating functionality by setting the **CC8yCMC**.GATES with the proper value.

Notice that the gating function is level active and therefore the active/passive configuration is set only by the **CC8yINS**.EVxLM.

The status bit during an external gating signal continues to be asserted when the compare value is reached and deasserted when the counter reaches 0000_H. One should note that the counter continues to use the period register to identify a wrap around condition. **Figure 19-37** shows the usage of an external signal for gating the slice

counter. The signal was set as active LOW, which means the counter gating functionality is active when the external value is zero.

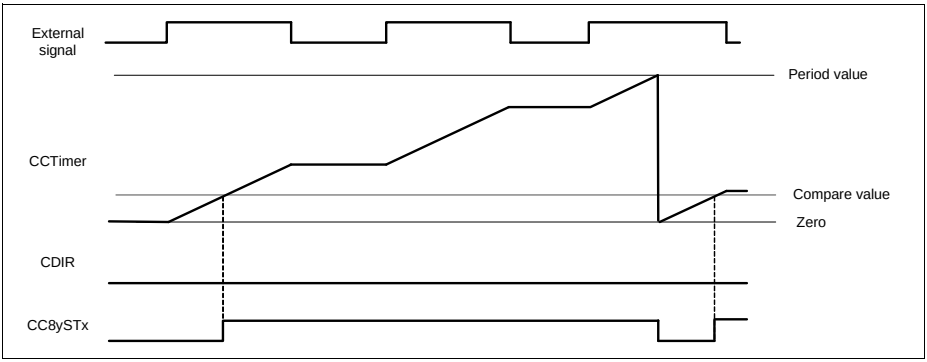


Figure 19-37 External gating

19.2.8.4 External Count Signal

There is also the possibility of selecting an external signal to act as the counting event.

To select an external counting, one should map one of the events (output of the input selector) to a specific input signal, by setting the required value in the **CC8yINS**.EVxIS register and indicating the active edge of the signal on the **CC8yINS**.EVxEM register. This event should be then mapped to the counting functionality by setting the **CC8yCMC**.CNTS with the proper value.

Notice that the counting function is edge active and therefore the active/passive configuration is set only by the **CC8yINS**.EVxEM.

One can select just a the rising, falling or both edges to perform a count. On **Figure 19-38**, the external signal was selected as a counter event for both falling and rising edges. Wrap around condition is still applied with a comparison with the period register.

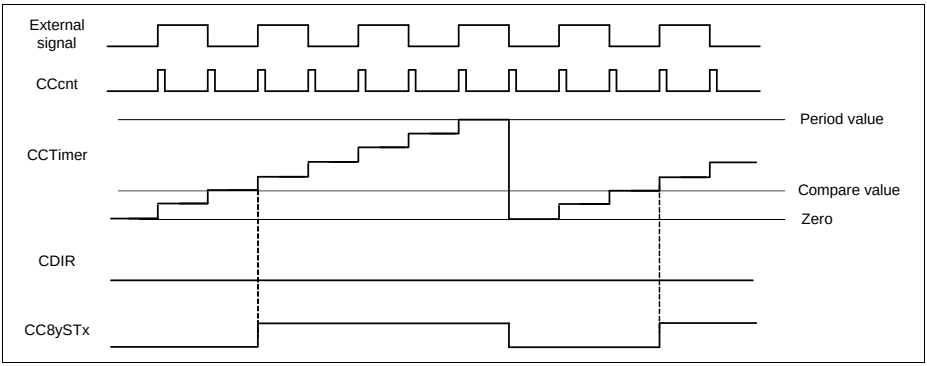


Figure 19-38 External count

19.2.8.5 External Load

Each slice of the CCU8 also has a functionality that enables the user to select an external signal as trigger for reloading the value of the timer with the current value of one compare register (if **CC8yTCST**.CDIR = 0_B) or with the value of the period register (if **CC8yTCST**.CDIR = 1_B).

The timer can be reloaded with the value from the compare channel 1 or compare channel 2 depending on the value set in the **CC8yTC**.TLS field, see **Figure 19-39**.

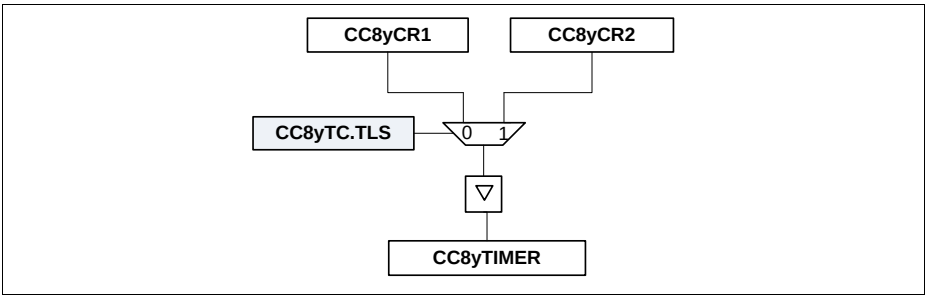


Figure 19-39 Timer load selection

To select an external load signal, one should map one of the events (output of the input selector) to a specific input signal, by setting the required value in the **CC8yINS**.EVxIS register and indicating the active edge of the signal on the **CC8yINS**.EVxEM register. This event should be then mapped to the load functionality by setting the **CC8yCMC**.LDS with the proper value.

Notice that the load function is edge active and therefore the active/passive configuration is set only by the **CC8yINS**.EVxEM.

On figure **Figure 19-40**, the external signal (1) was used to act as a load trigger, active on the rising edge. Every time that a rising edge on external signal (1) is detected, the timer value is loaded with the value present on the compare register. If an external signal is being used to control the counting direction, up or down, the timer value can be loaded also with the value set in the period register. The External signal (2) represents the counting direction control (active HIGH). If at the moment that a load trigger is detected, the signal controlling the counting direction is imposing a decrement, then the value set in the timer is the period value.

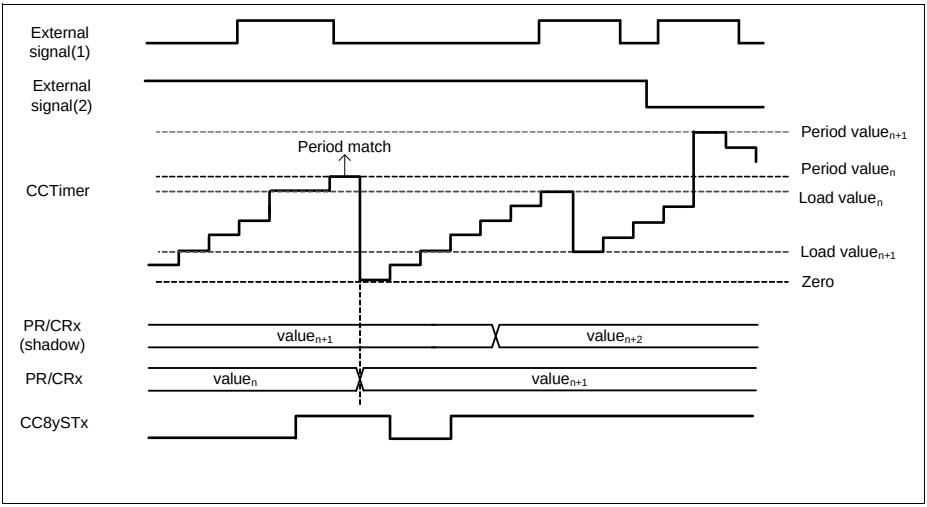


Figure 19-40 External load

19.2.8.6 External Capture

When selecting an external signal to be used as a capture trigger (if **CC8yCMC**.CAP0S or **CC8yCMC**.CAP1S are different from 0_H), the user is automatically setting the specific slice into capture mode.

In capture mode the user can have up to four capture registers, see **Figure 19-43**: capture register 0 (**CC8yC0V**), capture register 1 (**CC8yC1V**), capture register 2 (**CC8yC2V**) and capture register 3 (**CC8yC3V**).

These registers are shared between compare and capture modes, which imposes:

- if **CC8yC0V** and **CC8yC1V** are used for capturing, the compare registers **CC8yCR1** and **CC8yCR1S** are not available (compare channel 1 is not available)
- if **CC8yC2V** and **CC8yC3V** are used for capturing, the compare registers **CC8yCR2** and **CC8yCR2S** are not available (compare channel 2 is not available)

To select an external capture signal, one should map one of the events (output of the input selector) to a specific input signal, by setting the required value in the **CC8yINS**.EVxIS register and indicating the active edge of the signal on the **CC8yINS**.EVxEM register.

This event should be then mapped to the capture functionality by setting the **CC8yCMC**.CAP0S/**CC8yCMC**.CAP1S with the proper value.

Notice that the capture function is edge active and therefore the active/passive configuration is set only by the **CC8yINS**.EVxEM.

The user has the possibility of selecting the following capture schemes:

- Different capture events for **CC8yC0V**/**CC8yC1V** and **CC8yC2V**/**CC8yC3V**
- The same capture event for **CC8yC0V**/**CC8yC1V** and **CC8yC2V**/**CC8yC3V** with the same capture edge. For this capture scheme, only the CCcapt1 functionality needs to be programmed. To enable this scheme, the field **CC8yTC**.SCE needs to be set to 1_B.

Different Capture Events (SCE = 0_B)

Every time that a capture trigger 1 occurs, CCcapt1, the actual value of the timer is captured into the capture register 3 and the previous value stored in this register is transferred into capture register 2.

Every time that a capture trigger 0 occurs, CCcapt0, the actual value of the timer is captured into the capture register 1 and the previous value stored in this register is transferred into capture register 0.

Every time that a capture procedure into one of the registers occurs, the respective full flag is set. This flag is cleared automatically by HW when the SW reads back the value of the capture register.

The capture of a new value into a specific capture registers is dictated by the status of the full flag as follows:

$$CC8yC1V_{capt} = \text{NOT}(CC8yC1V_{full\_flag} \text{ AND } CC8yC0V_{full\_flag}) \quad (19.4)$$

$$CC8yC0V_{capt} = CC8yC1V_{full\_flag} \text{ AND NOT}(CC8yC0V_{full\_flag}) \quad (19.5)$$

It is also possible to disable the effect of the full flags by setting the **CC8yTC**.CCS = 1_B. This enables a continuous capturing independent if the values captured have been read or not.

On **Figure 19-41**, an external signal was selected as an event for capturing the timer value into the **CC8yC0V**/**CC8yC1V** registers. The status bit, CC8ySTx, during capture mode is asserted whenever a capture trigger is detected and de asserted when the counter matches 0000_H.

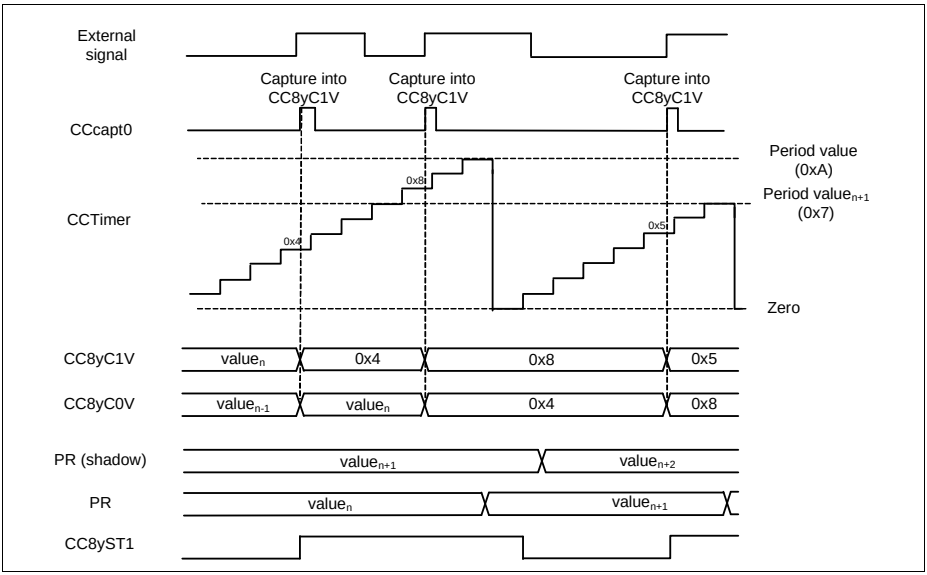


Figure 19-41 External capture - CC8yCMC.CAP0S != 00_B, CC8yCMC.CAP1S = 00_B

On **Figure 19-42**, two different signals were used as trigger sources for capturing the timer value into the **CC8yC0V**/**CC8yC1V** and **CC8yC2V**/**CC8yC3V** registers. External signal(1) was selected as an rising edge active source for the channel 1 capture trigger. External signal(2) was selected has the source for the channel 2 capture trigger, but as opposite to the external signal(1), the active edge was set as falling.

See **Section 19.2.14.4** for the complete capture mode usage description.

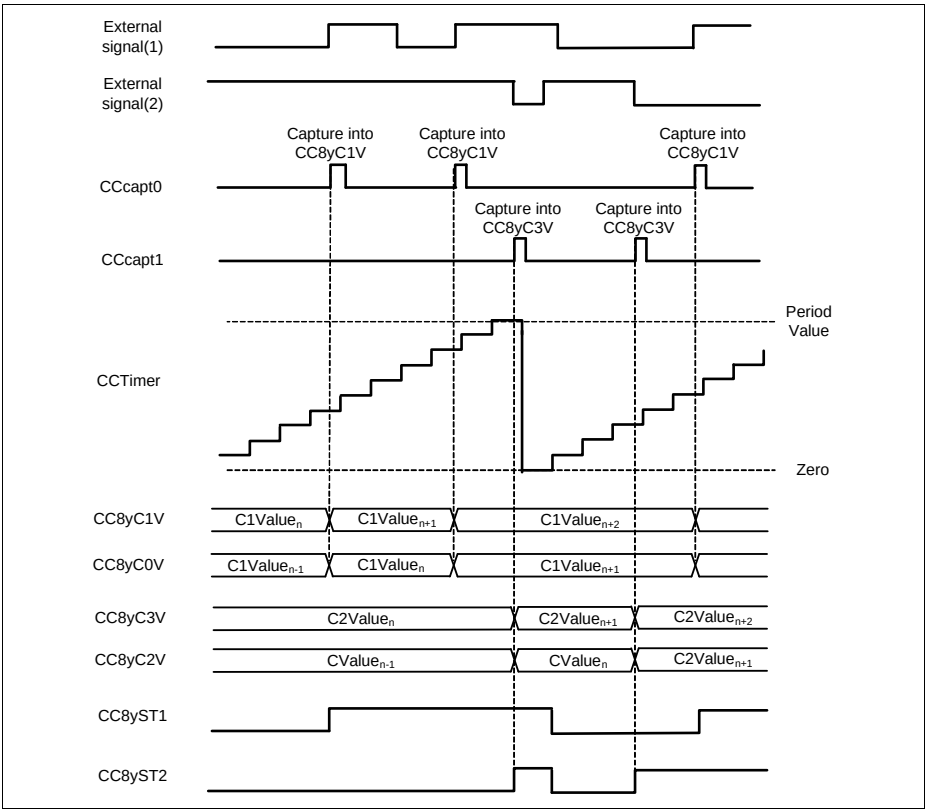


Figure 19-42 External capture - CC8yCMC.CAP0S != 00_B, CC8yCMC.CAP1S != 00_B

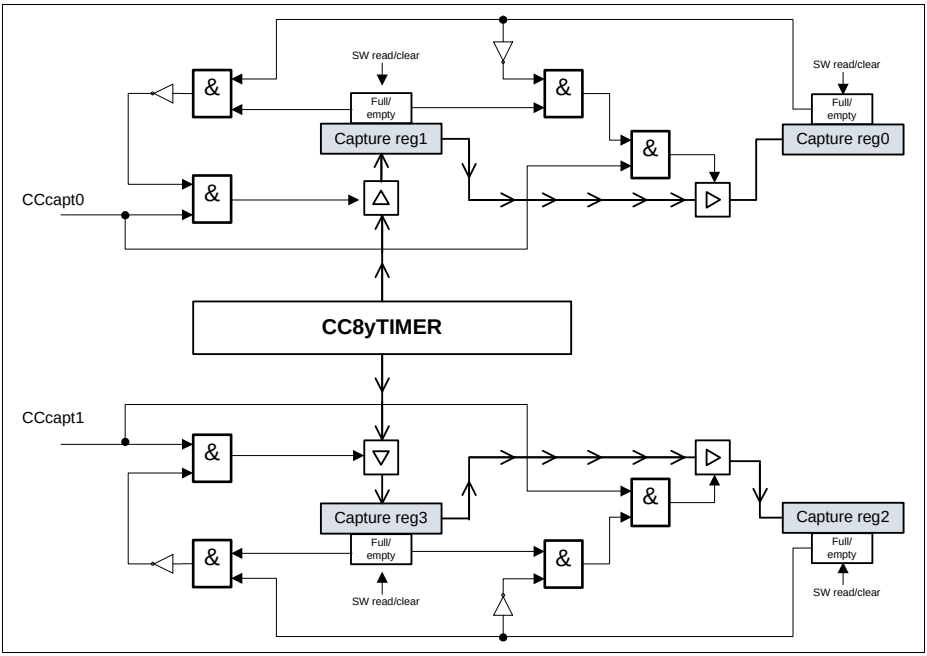


Figure 19-43 Slice capture logic

Same Capture Event (SCE = 1_B)

Setting the field **CC8yTC**.SCE = 1_B, enables the possibility of having 4 capture registers linked with the same capture event, **Figure 19-45**.The functionality that controls the capture is the CCcapt1.

The capture logic follows the same structure shown in **Figure 19-43** but extended to a four register chain, see **Figure 19-44**.The same full flag lock rules are applied to the four register chain (it also can be disabled by setting the **CC8yTC**.CCS = 1_B):

$$CC8yC3V_{capt} = \text{NOT}(CC8yC3V_{full\_flag} \text{ AND } CC8yC2V_{full\_flag} \text{ AND } CC8yC2V_{full\_flag} \text{ AND } CC8yC1V_{full\_flag}) \quad (19.6)$$

$$CC8yC2V_{capt} = CC8yC3V_{full\_flag} \text{ AND NOT}(CC8yC2V_{full\_flag} \text{ AND } CC8yC1V_{full\_flag} \text{ AND } CC8yC0V_{full\_flag}) \quad (19.7)$$

$$CC8yC1V_{capt} = CC8yC2V_{full\_flag} \text{ AND NOT}(CC8yC1V_{full\_flag} \text{ AND } CC8yC0V_{full\_flag}) \quad (19.8)$$

$$CC8yC0V_{capt} = CC8yC1V_{full\_flag} \text{ AND NOT}(CC8yC0V_{full\_flag}) \quad (19.9)$$

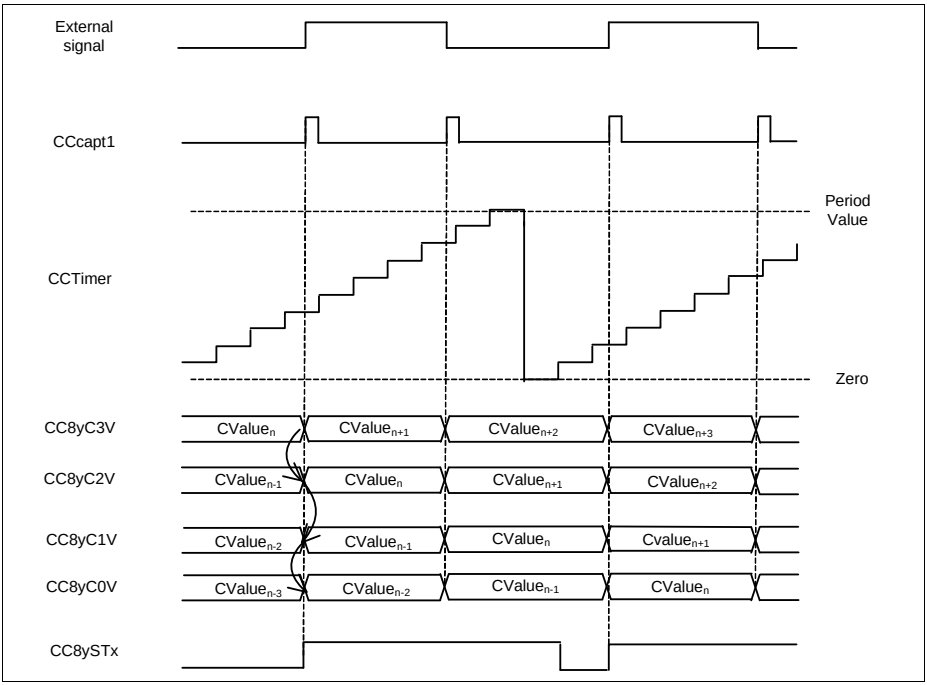


Figure 19-44 External Capture - CC8yTC.SCE = 1_B

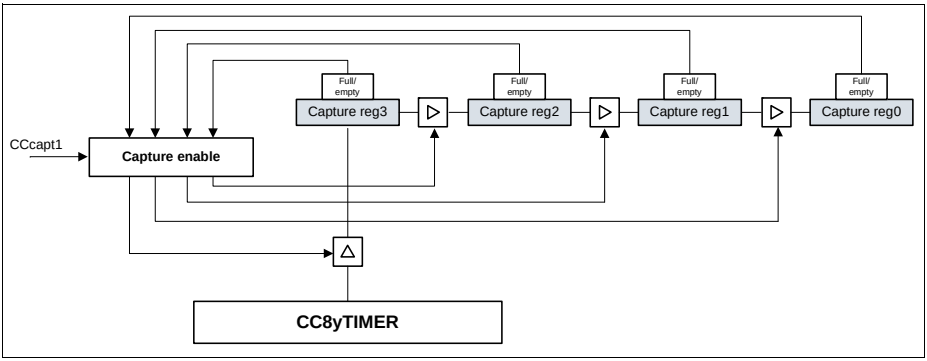


Figure 19-45 Slice Capture Logic - CC8yTC.SCE = 1_B

19.2.8.7 External Modulation

An external signal can be used also to perform a modulation at the output of each slice.

To select an external modulation signal, one should map one of the input signals to one of the events, by setting the required value in the **CC8yINS**.EVxIS register and indicating the active level of the signal on the **CC8yINS**.EVxLM register. This event should be then mapped to the modulation functionality by setting the **CC8yCMC**.MOS = 01_B if event 0 is being used, **CC8yCMC**.MOS = 10_B if event 1 or **CC8yCMC**.MOS = 11_B if event 2.

Notice that the modulation function is level active and therefore the active/passive configuration is set only by the **CC8yINS**.EVxLM.

The external modulation signal can be applied to each compare channel independently, or it can be applied to both channels, by setting **CC8yTC**.EME = 11_B.

The modulation has two modes of operation:

- modulation event is used to reset the CC8ySTx bit - **CC8yTC**.EMT = 0_B
- modulation event is used to gate the outputs - **CC8yTC**.EMT = 1_B

On **Figure 19-46**, we have a external signal configured to act as modulation source that clears the ST bit, **CC8yTC**.EMT = 0_B. It was programmed to be an active LOW event and therefore, when this signal is LOW the output value is following the normal ACTIVE/PASSIVE rules.

When the signal is HIGH (inactive state), then the CC8ySTx bit is cleared and the output is forced into the PASSIVE state. Notice that the values of the status bit, CC8ySTx and the specific output CCU8x.OUTy are not linked together. One can choose for the output to be active LOW through the **CC8yPSL**.PSLx bit.

The exit of the external modulation inactive state is synchronized with the PWM period due to the fact that the CC8ySTx bit is cleared and cannot be set while the modulation signal is inactive.

The entering into inactive state also can be synchronized with the PWM period, by setting **CC8yTC**.EMS = 1_B. With this all possible glitches at the output are avoided, see **Figure 19-47**.

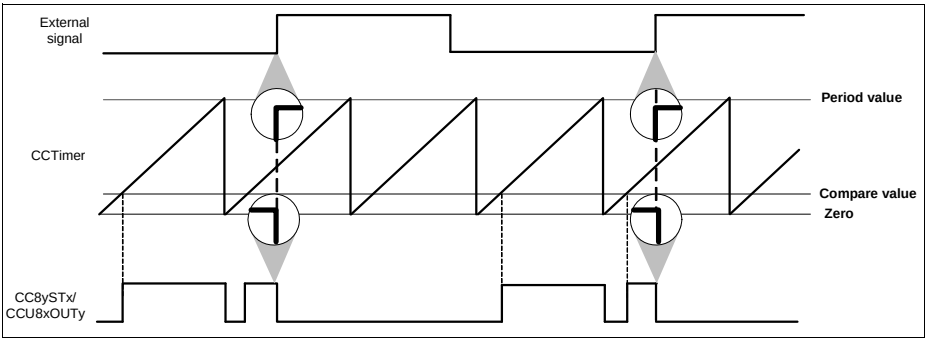


Figure 19-46 External modulation resets the ST bit - CC8yTC.EMS = 0_B

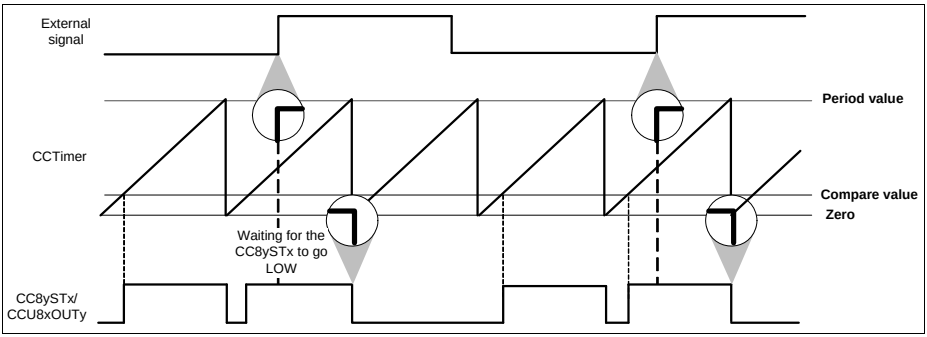


Figure 19-47 External modulation clearing the ST bit - CC8yTC.EMS = 1_B

On **Figure 19-48**, the external modulation event was used as gating signal of the outputs, **CC8yTC**.EMT = 1_B. The external signal was configured to be active HIGH, **CC8yINS**.EVxLM = 0_B, which means that when the external signal is HIGH the outputs are set to the PASSIVE state.In this mode, the gating event can also be synchronized with the PWM signal by setting the **CC8yTC**.EMS = 1_B.

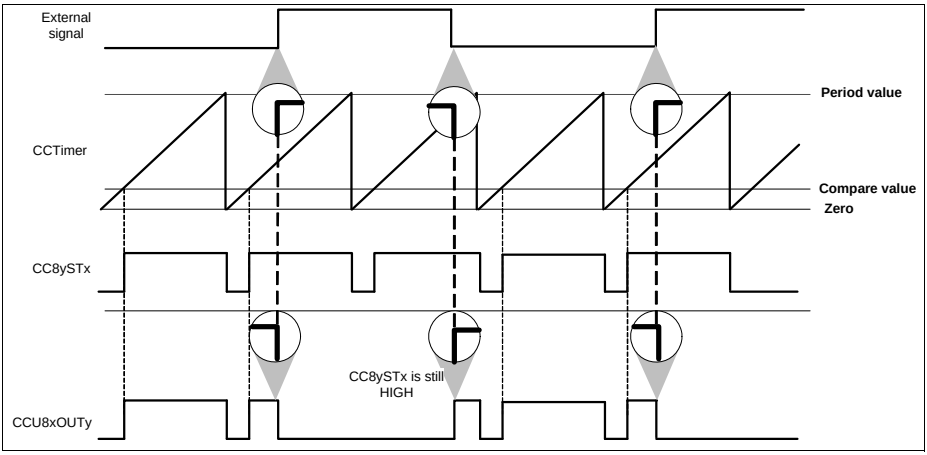


Figure 19-48 External modulation gating the output - CC8yTC.EMT = 1_B

19.2.8.8 Trap Function

The TRAP functionality allows the PWM outputs to react on the state of an input pin. This functionality can be used to switch off the power devices if the TRAP input becomes active.

To select the trap functionality, one should map one of the input signals to event number 2, by setting the required value in the **CC8yINS**.EV2IS register and indicating the active level of the signal on the **CC8yINS**.EV2LM register. This event should be then mapped to the trap functionality by setting the **CC8yCMC**.TS = 1_B.

Notice that the trap function is level active and therefore the active/passive configuration is set only by the **CC8yINS**.EV2LM.

There are two bitfields that can be monitored via software to crosscheck the TRAP function, **Figure 19-49**:

- The TRAP state bit, **CC8yINTS**.E2AS. This bitfield if the TRAP is currently active or not. This bitfield is therefore setting the specific Timer Slice output, into ACTIVE or PASSIVE state.
- The TRAP Flag, **CC8yINTS**.TRPF. This bitfield is used as a remainder in the case that the TRAP condition is cleared automatically via hardware. This field needs to be cleared by the software.

The E2AS can be configured to affect all of the CCU8 slice outputs, or a specific sub set of outputs via the **CC8yTC**.TRAPEy bit fields.

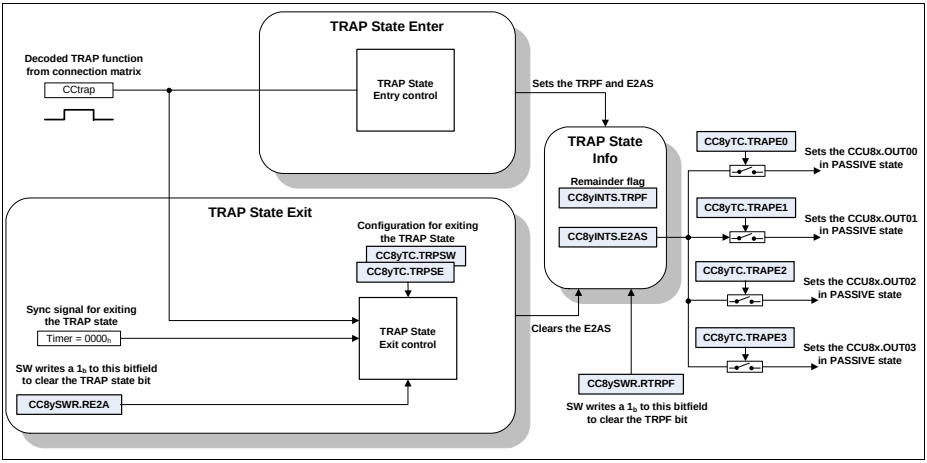


Figure 19-49 Trap control diagram

When a TRAP condition is detected at the selected input pin, both the Trap Flag and the Trap State bit are set to 1_B. The Trap State is entered immediately, by setting the CCU8xOUTy into the programmed PASSIVE state, **Figure 19-50**.

Exiting the Trap State can be done in two ways (**CC8yTC**.TRPSW register):

- automatically via HW, when the TRAP signal becomes inactive - **CC8yTC**.TRPSW = 0_B

- by SW only, by clearing the **CC8yINTS**.E2AS.The clearing is only possible if the input TRAP signal is in inactive state - **CC8yTC**.TRPSW = 1_B

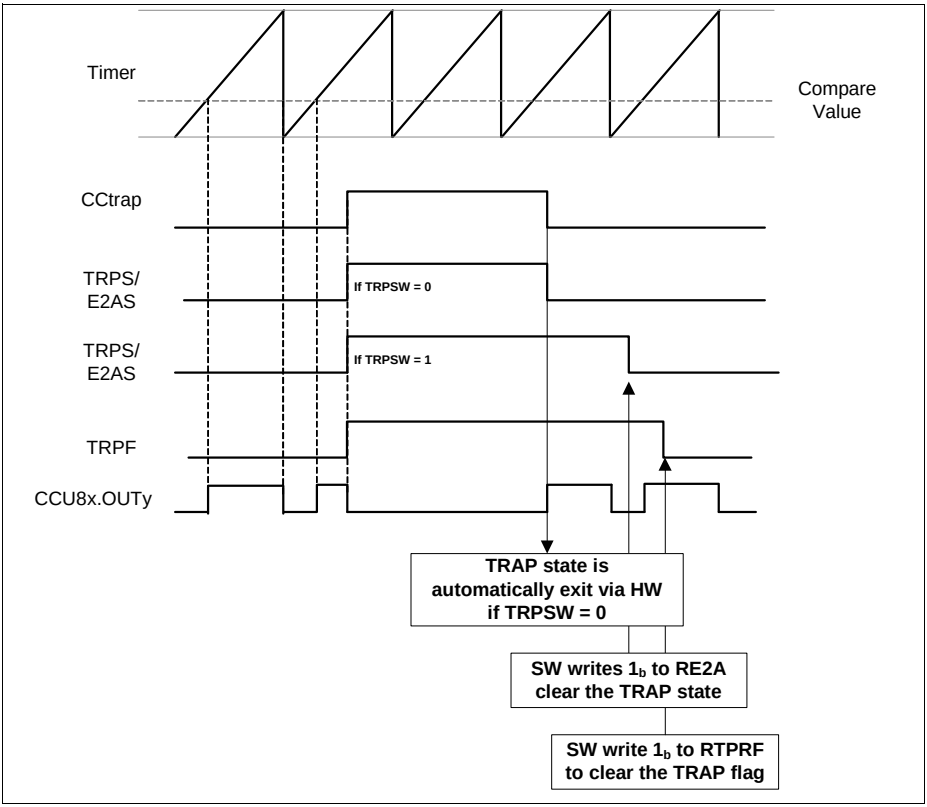


Figure 19-50 Trap timing diagram, CC8yTCST.CDIR = 0 CC8yPSL.PSL = 0

It is also possible to synchronize the exiting of the TRAP state with the PWM signal, **Figure 19-51**. This function is enabled when the bitfield **CC8yTC**.TRPSE = 1_B.

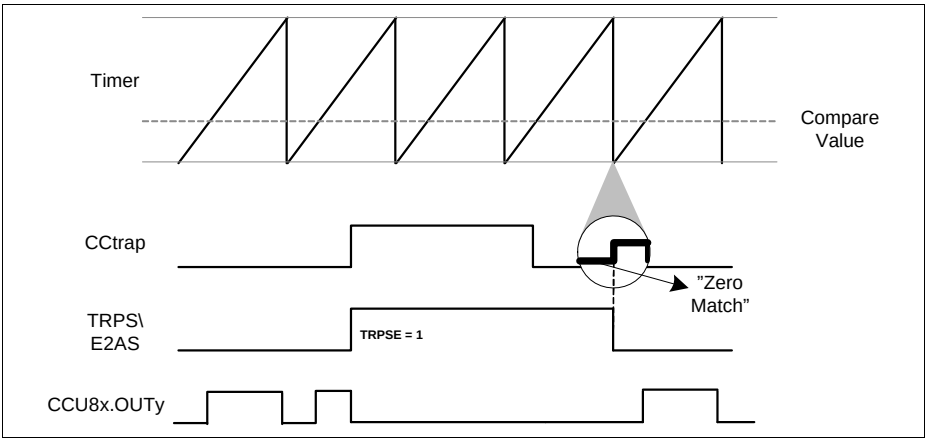


Figure 19-51 Trap synchronization with the PWM signal

19.2.8.9 Status Bit Override

For complex timed output control, each slice has a functionality that enables the override of the status bit of compare channel 1 (CC8yST1) with a value passed trough an external signal.

The override of the status bit, can then lead to a change on the output pins CCU8x.OUTy0 and CCU8x.OUTy1 (from inactive to active or vice versa).

To enable this functionality, two signals are needed:

- One signal that acts as a trigger to override the status bit (edge active)
- One signal that contains the value to be set in the status bit (level active)

To select the status bit override functionality, one should map the signal that acts as trigger to the event number 1, by setting the required value in the **CC8yINS**.EV1IS register and indicating the active edge of the signal on the **CC8yINS**.EV1EM register.

The signal that carries the value to be set on the status bit, needs to be mapped to the event number 2, by setting the required value in the **CC8yINS**.EV2IS register. The **CC8yINS**.EV2LM register should be set to 0_B if no inversion on the signal is needed and to 1_B otherwise.

The events should be then mapped to the status bit functionality by setting the **CC8yCMC**.OFS = 1_B.

Figure 19-52 shows the functionality of the status bit override, when the external signal(1) was selected as trigger source (rising edge active) and the external signal(2) was selected as override value.

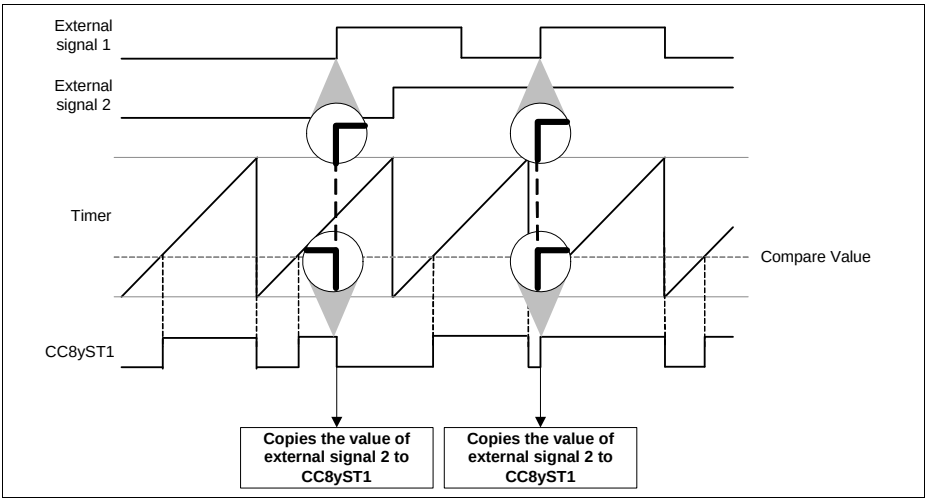


Figure 19-52 Status bit override

19.2.9 Multi-Channel Support

The multi channel control mode is selected individually in each slice by setting the **CC8yTC**.MCMEx = 1_B.

With this mode, the output state of the Timer Slices PWM signal(s) (the ones set in multichannel mode) can be controlled in parallel by a single pattern.

The pattern is controlled via the CCU8 inputs, CCU8x.MCIy[3:0]. Each group of these inputs is connected accordingly to the specific Timer Slice: for slice 0, CCU8xMCI1[3:0] for slice 1, CCU8xMCI2[3:0] for slice 2 and CCU8xMCI3[3:0] for slice 3.

This pattern can be controlled directly by one of the POSIF modules and be updated in parallel for all the Timer Slices.

Using the POSIF module in conjunction with the Multi Channel support of the CCU8, one can achieve a complete synchronicity between the output state update, CCU8x.OUTy and the update of a new pattern, **Figure 19-53**.

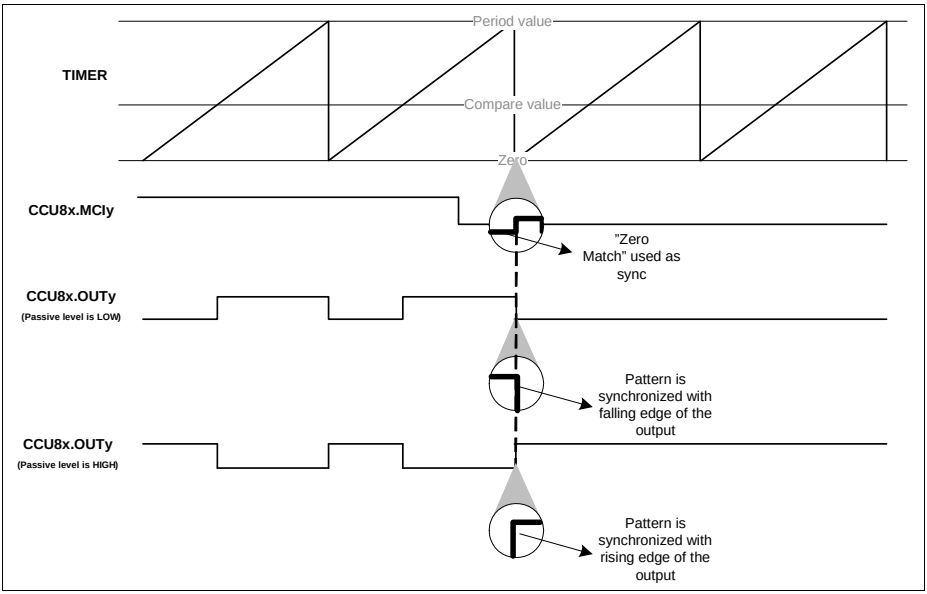


Figure 19-53 Multi channel pattern synchronization

These pattern inputs are going to be used in the output modulation control unit to put the specific PWM output into active or passive state: CCU8x.MCIy[0] has effect on the CC8yST1 path and therefore controls the CC8xOUT00 pin, CCU8x.MCy[1] is used in the same manner for the inverted CC8yST1 path, CCU8x.MCIy[2] and CCU8x.MCIy[3] are linked to the CC8yST2 and inverted CC8yST2 path respectively. **Figure 19-54** shows the simplified scheme for the multi channel control.

Figure 19-55, shows the usage of the multi channel mode in conjunction with two Timer Slices of the CCU8. The multi channel pattern is driven via the POSIF module, which enables a glitch free update of all the outputs of the CCU8.

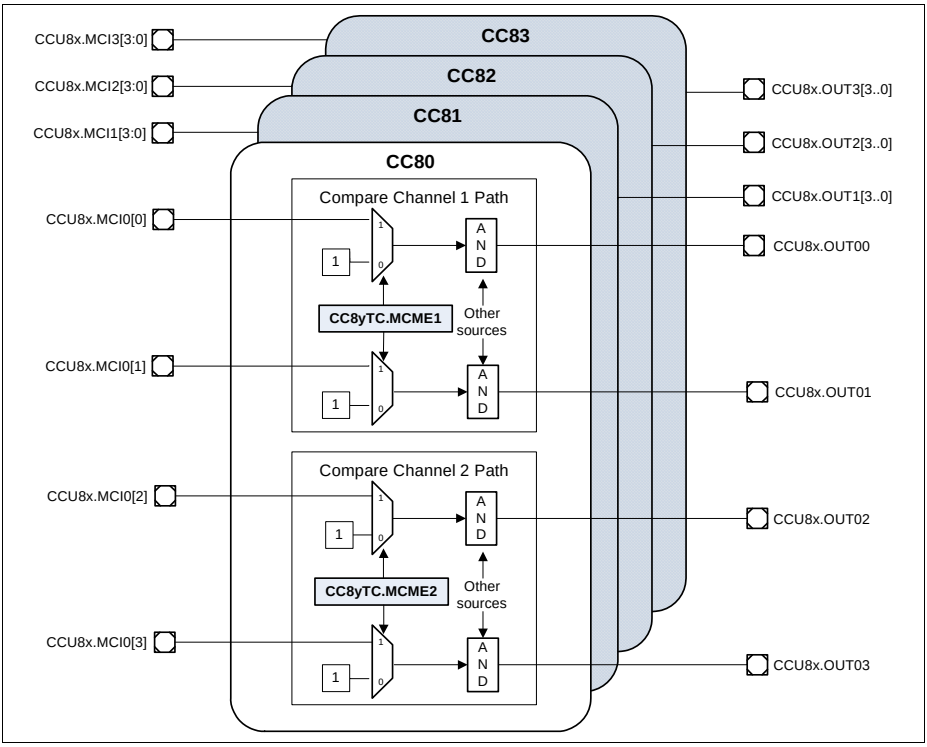


Figure 19-54 CCU8 Multi Channel overview

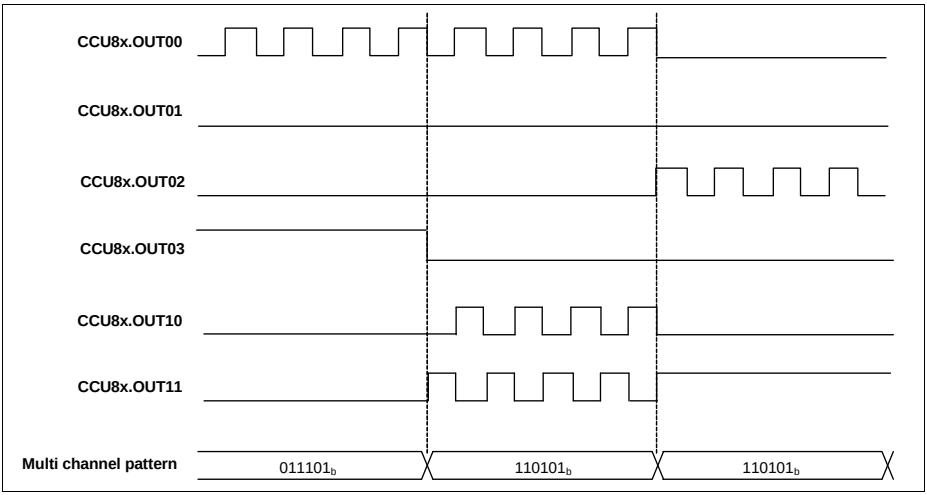


Figure 19-55 Multi Channel mode for multiple Timer Slices

The synchronization between the CCU8 and the POSIF is achieved, by adding a 3 cycle delay on the output path of each Timer Slice (between the status bit, CC8ySTx and the direct control of the output pin). This path is only selected when **CC8yTC**.MCMEx = 1_B.

On **Figure 19-56** the control of the CC8yST1 path is represented. The control of remaining paths follows the same mechanism (the multi channel is only enabled for the CC8yST2 path if **CC8yTC**.MCME2 = 1_B).

The multi pattern input synchronization can be seen on **Figure 19-57**. To achieve a synchronization between the update of the status bit, the sampling of a new multi channel pattern input is controlled by the period match or one match signal.

In a normal operation, where no external signal is used to control the counting direction, the signal used to enable the sampling of the pattern is always the period match when in edge aligned and the one match when in center aligned mode. When an external signal is used to control the counting direction, depending if the counter is counting up or counting down, the period match or the one match signal is used, respectively.

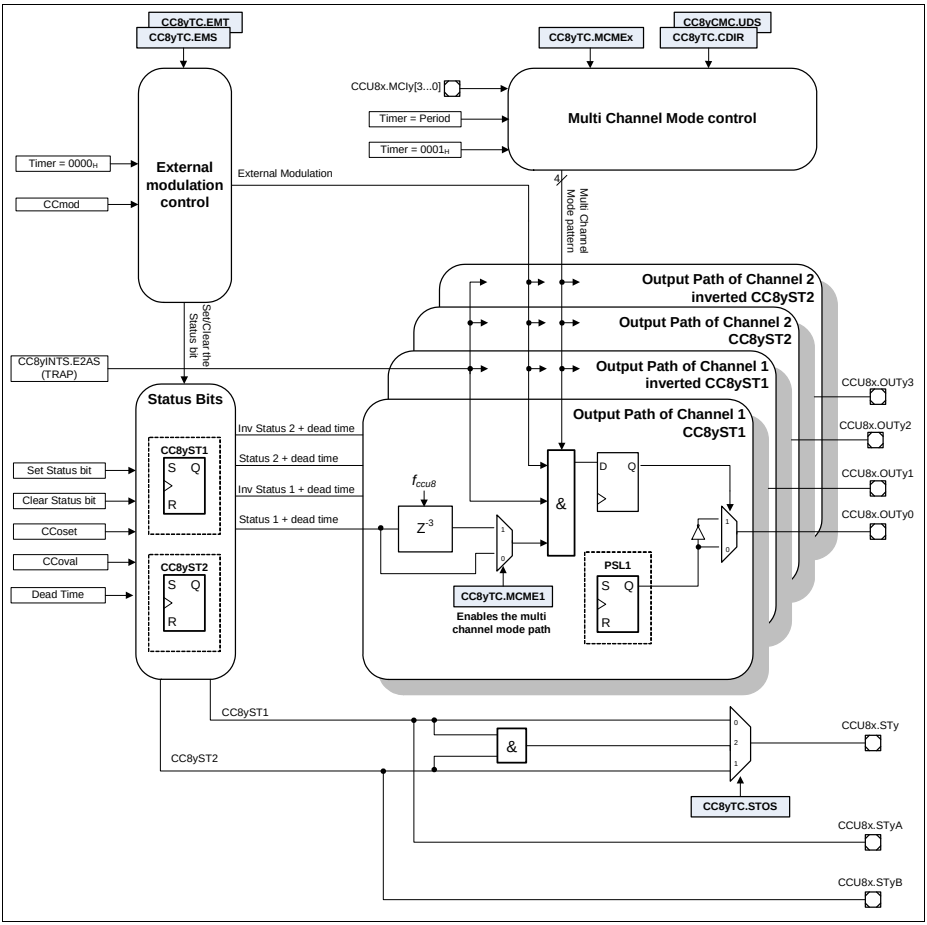


Figure 19-56 Output Control Diagram

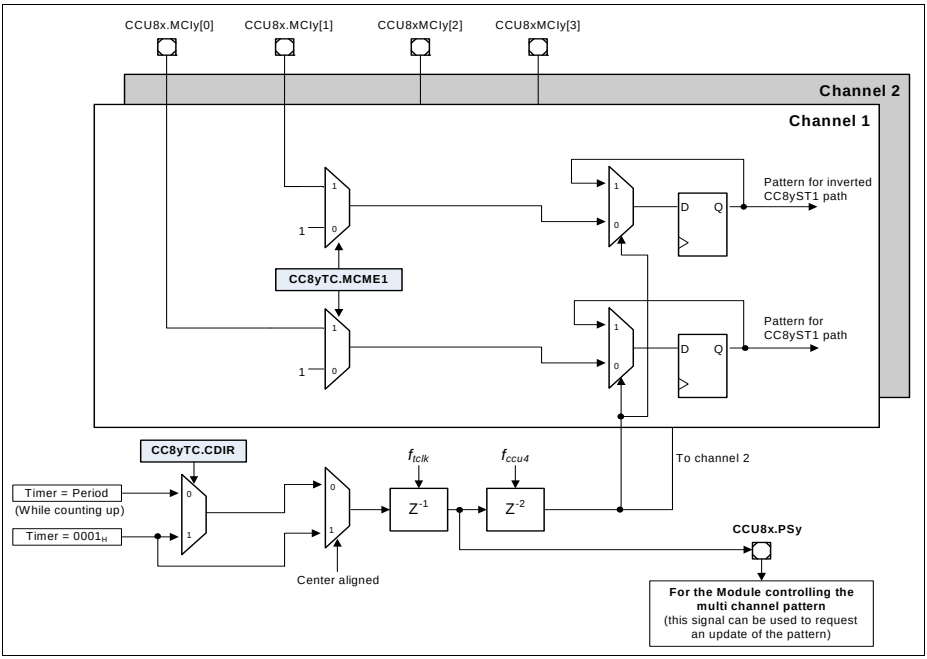


Figure 19-57 Multi Channel Pattern Synchronization Control

19.2.10 Timer Concatenation

The CCU8 offers a very easy mechanism to perform a synchronous timer concatenation. This functionality can be used by setting the **CC8yCMC**.TCE = 1_B. By doing this the user is doing a concatenation of the actual CCU8 slice with the previous one, see **Figure 19-58**.

Notice that is not possible to perform concatenation with non adjacent slices and that timer concatenation automatically sets the slice mode into Edge Aligned. It is not possible to perform timer concatenation in Center Aligned mode.

To enable a 64 bit timer, one should set the **CC8yCMC**.TCE = 1_B in all the slices (with the exception of the CC80 due to the fact that it doesn't contain this control field).

To enable a 48 bit timer, one should set the **CC8yCMC**.TCE = 1_B in two adjacent slices and to enable a 32 bit timer, the **CC8yCMC**.TCE is set to 1_B in the slice containing the MSBs. Notice that the timer slice containing the LSBs should always have the TCE bitfield set to 0_B.

Several combinations for timer concatenation can be made inside a CCU8 module:

- one 64 bit timer

- one 48 bit timer plus a 16 bit timer
- two 32 bit timers
- one 32 bit timer plus two 16 bit timers

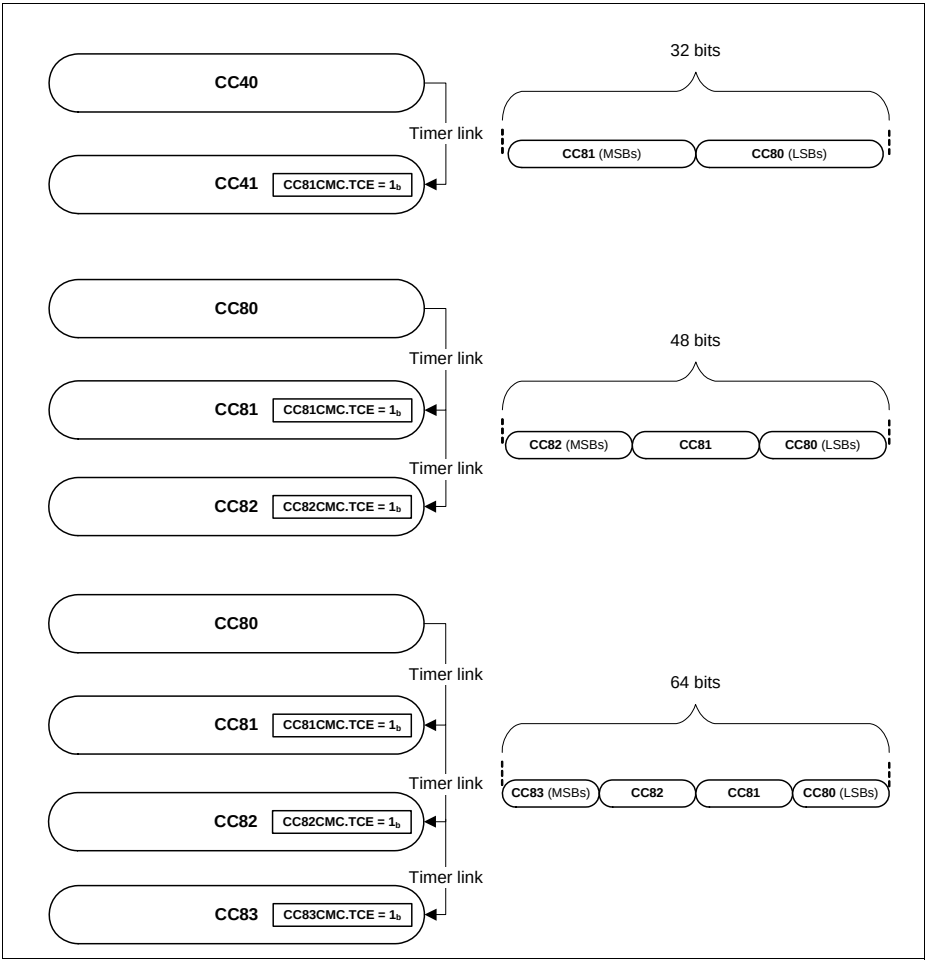


Figure 19-58 Timer concatenation example

Each Timer Slice is connected to the adjacent Timer Slices via a dedicated concatenation interface. This interface allows the concatenation of not only the Timer counting operation, but also a synchronous input trigger handling for capturing and loading operations, **Figure 19-59**.

Note: For all the cases, CC80 and CC83 are not considered adjacent slices

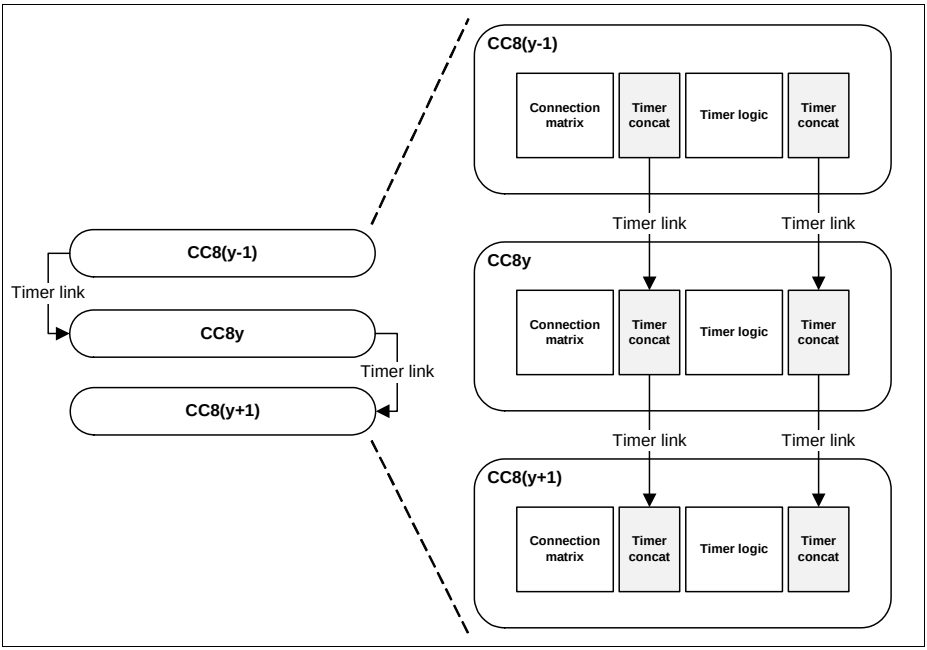


Figure 19-59 Timer concatenation link

Eight signals are present in the timer concatenation interface:

- Timer Period Match (CC8yPM)
- Timer Zero Match (CC8yZM)
- Timer Compare Match from channel 1 (CC8yCM1)
- Timer Compare Match from channel 2 (CC8yCM2)
- Timer counting direction function (CCupd)
- Timer load function (CCload)
- Timer capture function for CC8yC0V and CC8yC1V registers (CCcap0)
- Timer capture function for CC8yC2V and CC8yC3V registers (CCcap1)

The first five signals are used to perform the synchronous timing concatenation at the output of the Timer Logic, like it is seen in **Figure 19-59**. With this link, the timer length can be easily adjusted to 32, 48 or 64 bits (counting up or counting down)

The last three signals are used to perform a synchronous link between the capture and load functions, for the concatenated timer system. This means that the user can have a capture or load function programmed in the first Timer Slice, and propagate this capture or load trigger synchronously from the LSBs until the MSBs, **Figure 19-60**.

The capture or load function only needs to be configured in the first Timer Slice (the one holding the LSBs). From the moment that **CC8yCMC**.TCE is set to 1_B, in the following Timer Slices, the link between these functions is done automatically by the hardware.

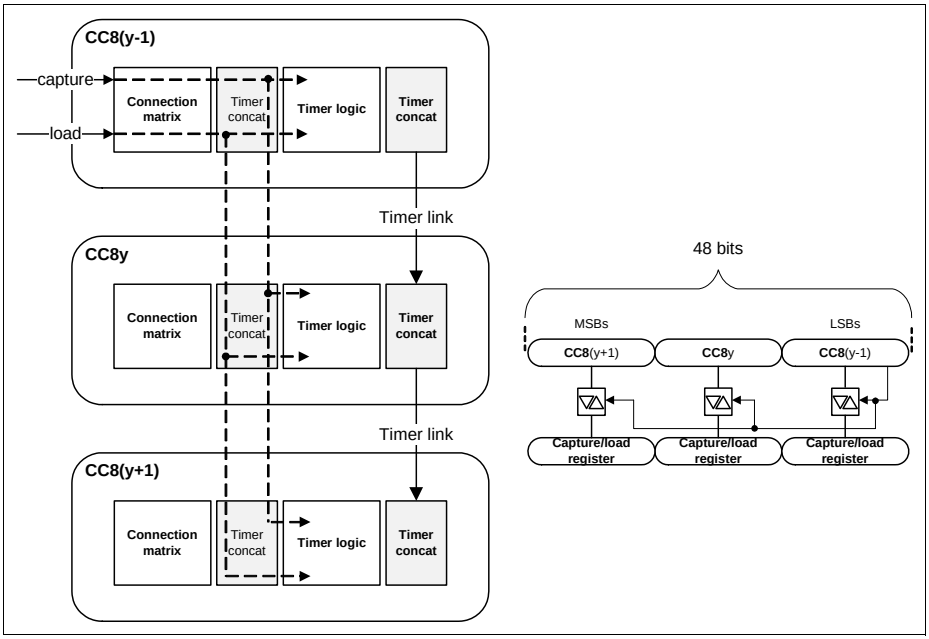


Figure 19-60 Capture/Load Timer Concatenation

the period match (CC8yPM) or zero match (CC8yZM) from the previous Timer Slice (with the immediately next lower index) are used in concatenated mode, as gating signal for the counter. This means that the counting operation of the MSBs only happens when a wrap around condition is detected (in the previous Timer Slice), avoiding additional DSP operations to extract the counting value.

With the same methodology, the compare match (CC8yCM1 and CC8yCM2), zero match and period match are gated with the specific signals from the previous Timer Slice. This means that the timing information is propagated throughout all the slices, enabling a completely synchronous match between LSB and MSB count, see **Figure 19-61**.

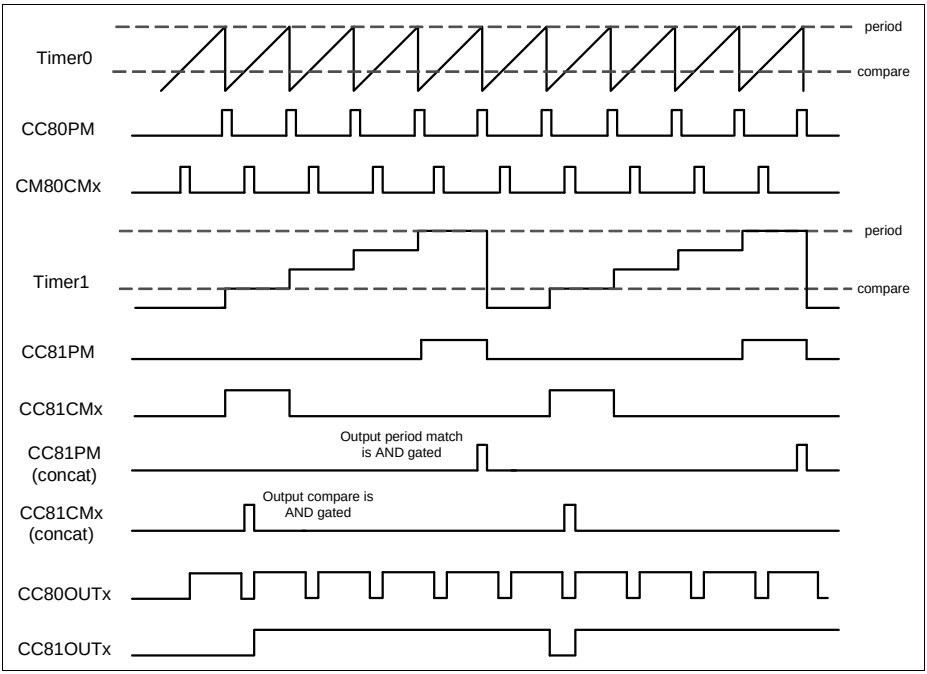


Figure 19-61 32 bit concatenation timing diagram

Note: the counting direction of the concatenated timer needs to be fixed. The timer can count up or count down, but the direction cannot be updated on the fly.

Figure 19-62 gives an overview of the timer concatenation logic. Notice that all the mechanism is controlled solely by the **CC8yCMC**.TCE bitfield.

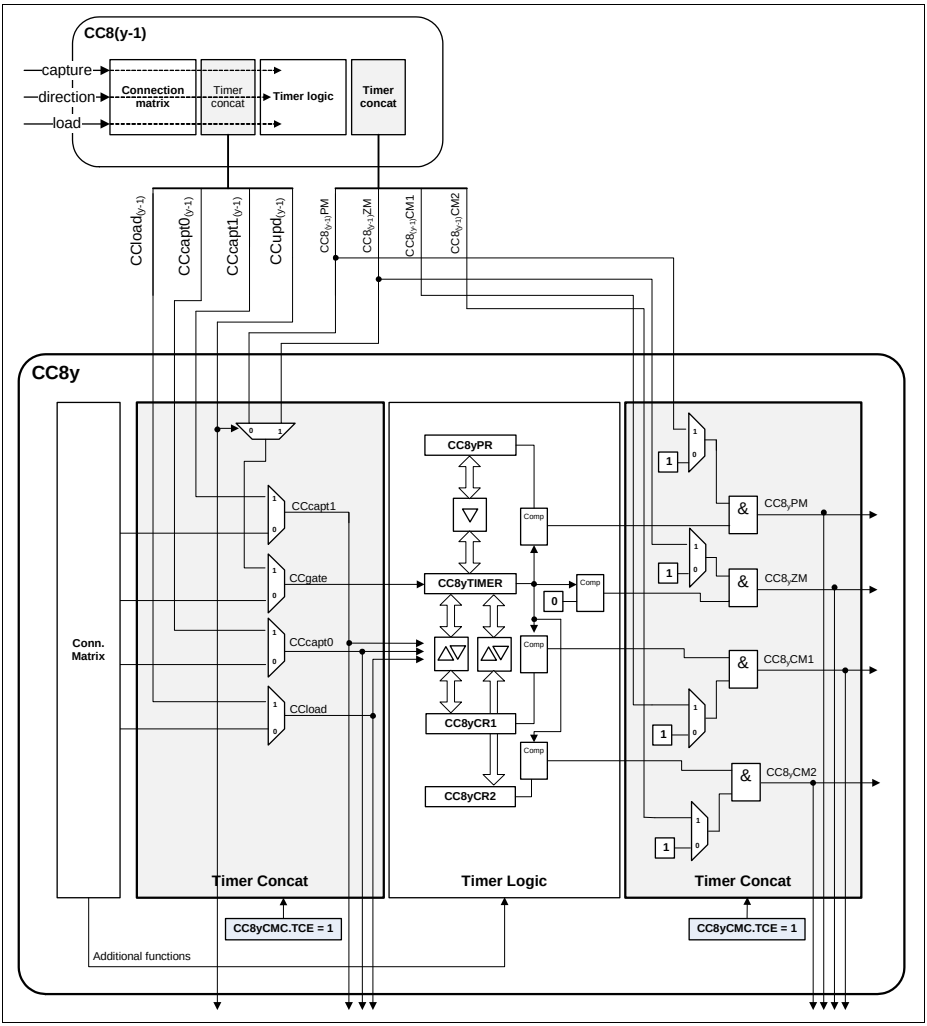


Figure 19-62 Timer concatenation control logic

19.2.11 Output Parity Checker

The parity checker function can be enabled by setting the **GIDLC**.PCH bit field to 1_B (parity checker is disabled while **GSTAT**.PCRB is 0_B).

This parity checker function, crosschecks the value at the output of the CCU8 module versus an input signal that should be connected to a driver XOR structure.

It is also possible to add a delay between the switching of the outputs and the evaluation of the input signal coming from the driver structure, and select which type of parity, even or odd (via the **GPCHK**.PCTS bit field). **Figure 19-63** shows the structure of the parity checker unit.

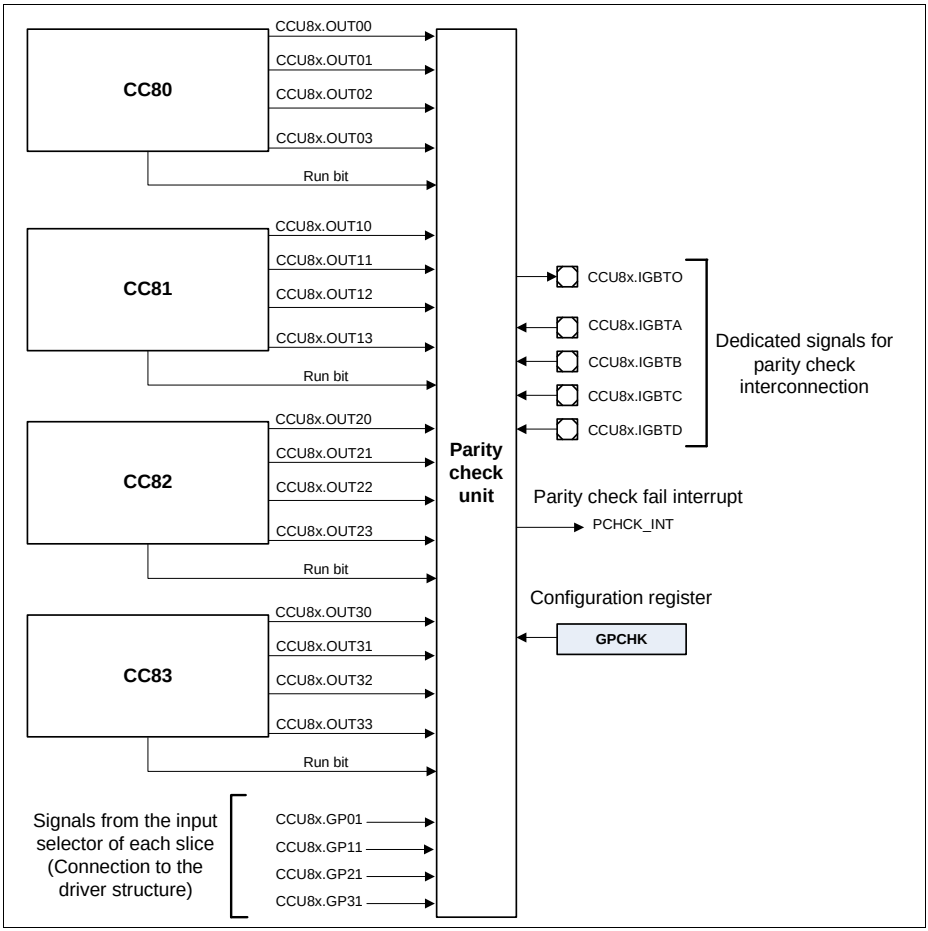


Figure 19-63 Parity checker structure

To use the parity check function, the user must select which signal is connected to the driver parity structure:

- The signal can be connected to any of the slices inputs
- The signal must be selected throughout the input selector mux of each slice. The signal must be mapped to the Event 1 of a slice.

Each of the CCU8 outputs can be individually selected to be part of the parity string and an interrupt is generated every time the input signal, coming from the driver structure, does not match the internally generated XOR result.

The interrupt is connected to the E1AS status bit of the slice where the driver parity output is connected:

- If **GPCHK**.PISEL = 00_B then the error status is in CC80INTS.E1AS
- If **GPCHK**.PISEL = 01_B then the error status is in CC81INTS.E1AS
- If **GPCHK**.PISEL = 10_B then the error status is in CC82INTS.E1AS
- If **GPCHK**-PISEL = 11_B then the error status is in CC83INTS.E1AS

The logic structure of the parity check is described on **Figure 19-64**. For a more detailed description of resource usage for the parity checker function, please address **Section 19.2.14.5**.

Configuration Example:

Driver parity output is connected to the input CCU8x.IN1B (where x = CCU8 unit). The input used to control the switching delay is the CCU8x.IGBTCCCU8. The driver is using 12 outputs coming from the first three slices with an even parity (PCTS field is in default).

The following registers should then be programmed:

CC8yINS.EV1IS = 0001_B; selects the input CCU8x.IN1B

GPCHK.PISEL = 01_B; selects the Event 1 coming from slice 1

GPCHK.PCDS = 10_B; selects the CCU8x.IGTBC input for delay control

GPCHK.PCSEL = $0FFF_H$; selects only the output signals of the first three slices for parity check

GIDLC.SPCH = 1_B; starts the parity function

When a mismatch between the driver output and the parity checker is detected, an interrupt is generated on Timer Slice 1. The interrupt status bit that stores the information is **CC8yINTS**.E1AS.

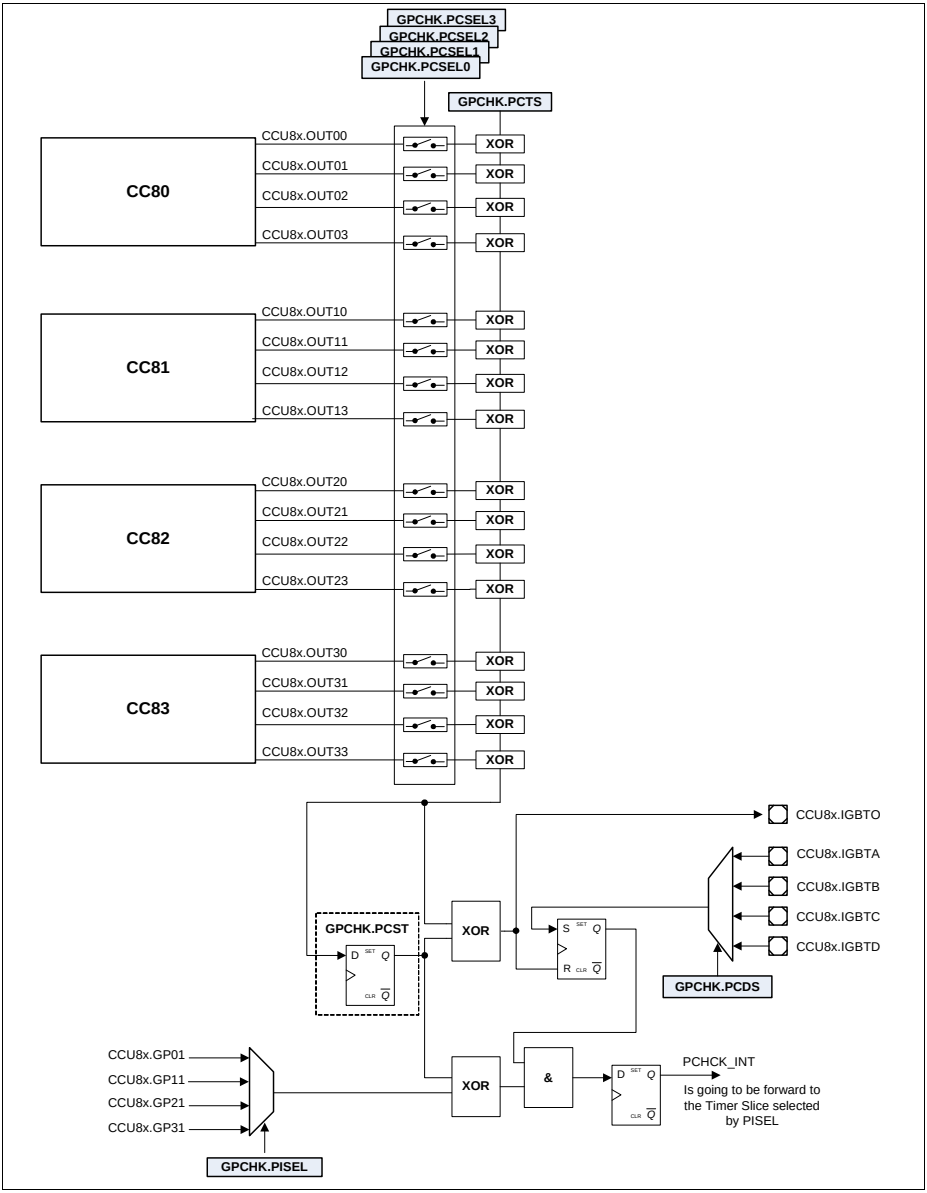


Figure 19-64 Parity checker logic

19.2.12 PWM Dithering

The CCU8 has an automatic PWM dithering insertion function. This functionality can be used with very slow control loops that cannot update the period/compare values in a fast manner, and by that fact the loop can lose precision on long runs. By introducing dither on the PWM signal, the average frequency/duty cycle is then compensated against that error.

Each slice contains a dither control unit, see **Figure 19-65**.

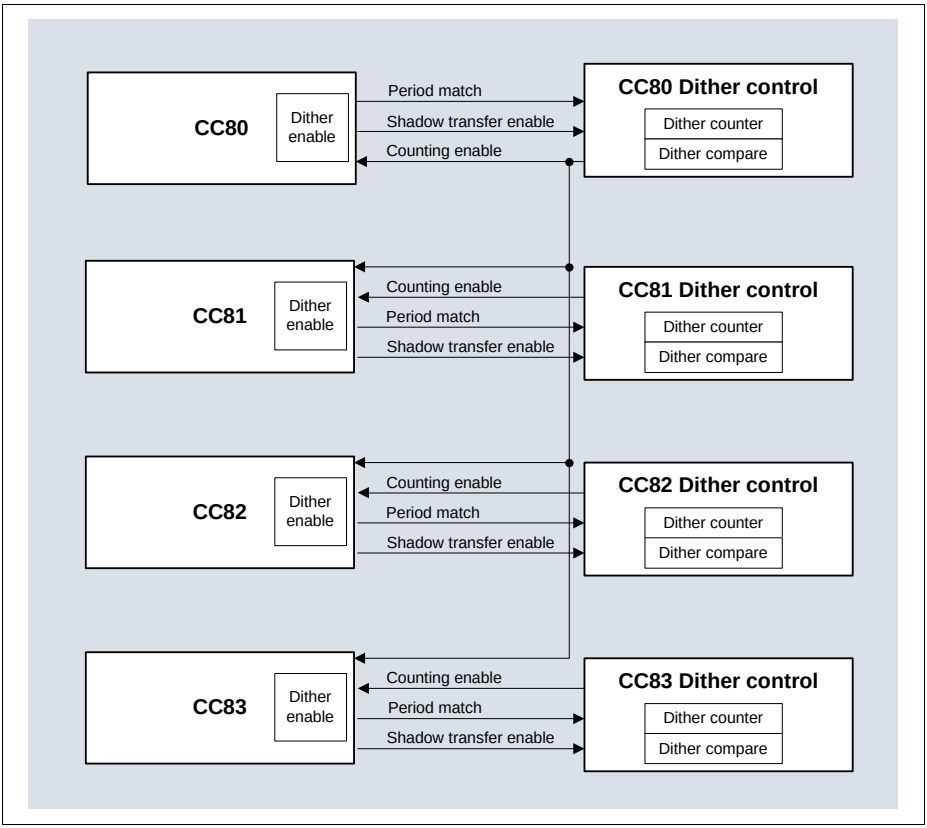


Figure 19-65 Dither structure overview

The dither control unit contains a 4 bit counter and a compare value. The four bit counter is incremented every time that a period match occurs. The counter works in a bit reverse mode so the distribution of increments stays uniform over 16 counter periods, see **Table 19-6**.

Table 19-6 Dither bit reverse counter

counter[3]	counter[2]	counter[1]	counter[0]
0	0	0	0
1	0	0	0
0	1	0	0
1	1	0	0
0	0	1	0
1	0	1	0
0	1	1	0
1	1	1	0
0	0	0	1
1	0	0	1
0	1	0	1
1	1	0	1
0	0	1	1
1	0	1	1
0	1	1	1
1	1	1	1

The counter is then compared against a programmed value, **CC8yDIT**.DCV. If the counter value is smaller than the programmed value, a gating signal is generated that can be used to extend the period, to delay the compare or both (controlled by the **CC8yTC**.DITHE field, see **Table 19-7**) for one clock cycle.

Table 19-7 Dither modes

DITHE[1]	DITH[0]	Mode
0	0	Dither is disabled
0	1	Period is increased by 1 cycle
1	0	Compare match is delayed by 1 cycle
1	1	Period is increased by 1 cycle and compare is delayed by 1 cycle

The dither compare value also has an associated shadow register that enables concurrent update with the period/compare registers of each CC8y. The control logic for the dithering unit is represented on **Figure 19-66**.

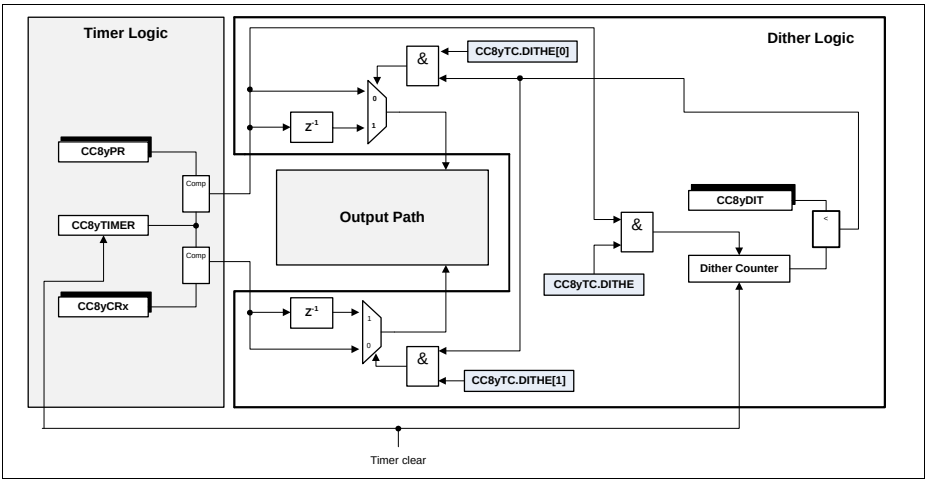


Figure 19-66 Dither control logic

Figure 19-72 to **Figure 19-72** show the effect of the different configurations of the dither, **CC8yTC**.DITHE, for both counting schemes, Edge and Center Aligned mode. In each figure, the bit reverse scheme is represented for the dither counter and the compare value was programmed with the value 8_H. In each figure, the variable T, represents the period of the counter, while the variable d indicates the duty cycle (status bit is set HIGH).

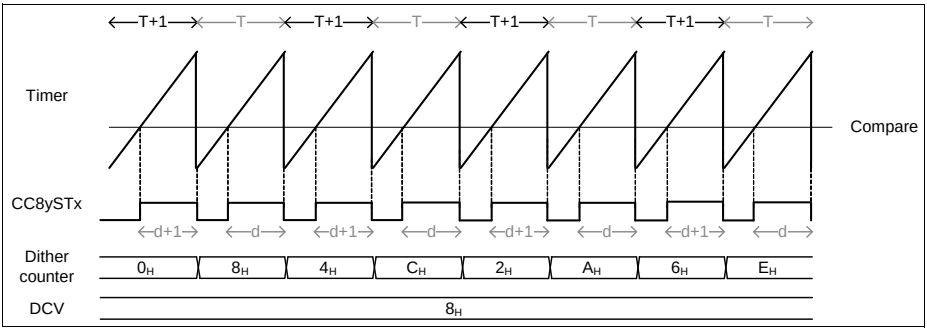


Figure 19-67 Dither timing diagram in edge aligned - CC8yTC.DITHE = 01_B

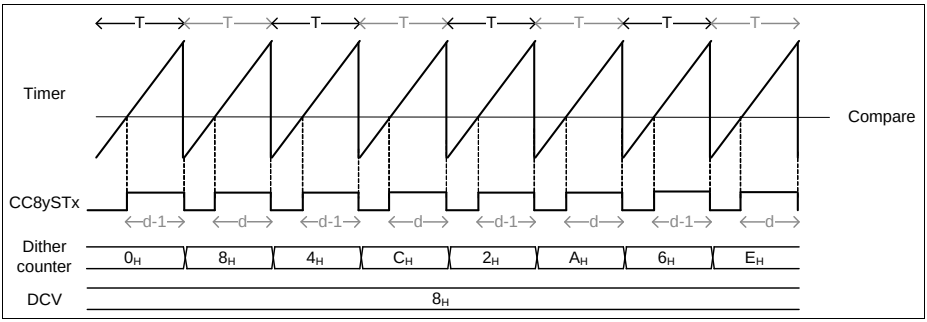


Figure 19-68 Dither timing diagram in edge aligned - CC8yTC.DITHE = 10_B

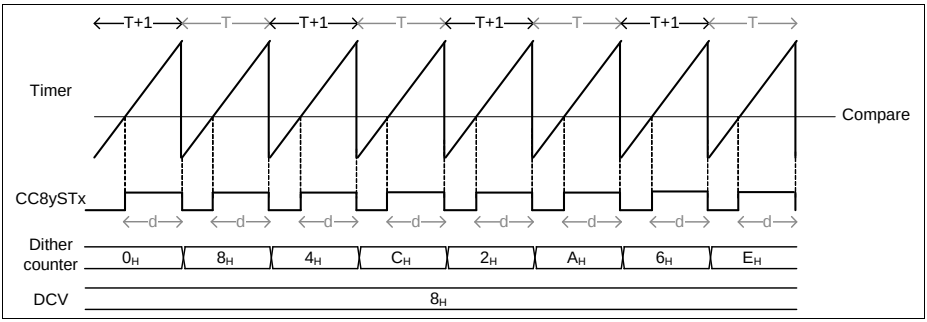


Figure 19-69 Dither timing diagram in edge aligned - CC8yTC.DITHE = 11_B

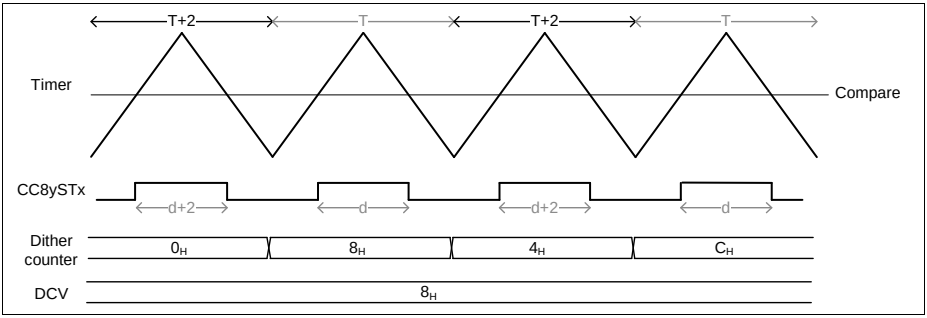


Figure 19-70 Dither timing diagram in center aligned - CC8yTC.DITHE = 01_B

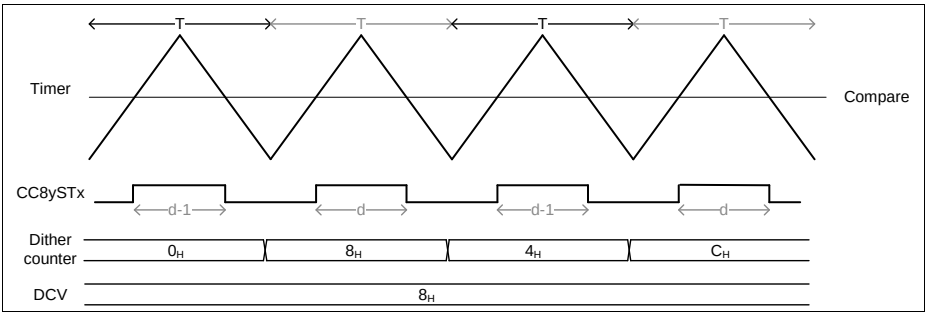


Figure 19-71 Dither timing diagram in center aligned - CC8yTC.DITHE = 10_B

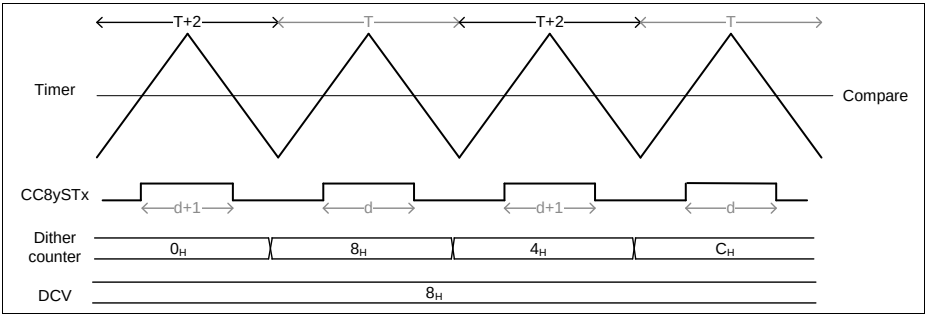


Figure 19-72 Dither timing diagram in edge aligned - CC8yTC.DITHE = 11_B

Note: When using the dither, is not possible to select a period value of FS when in edge aligned mode. In center aligned mode, the period value must be at least FS - 2.

19.2.13 Prescaler

The CCU8 contains a 4 bit prescaler that can be used in two operating modes for each individual slice:

- normal prescaler mode
- floating prescaler mode

The run bit of the prescaler can be set/cleared by SW by writing into the registers, **GIDLC**.SPRB and **GIDLS**.CPRB respectively or can also be cleared by the run bit of a specific slice. With the last mechanism, the run bit of the prescaler is cleared one clock cycle after the clear of the run bit of the selected sliced. To select which slice can perform this action, one should program the **GCTRL**.PRBC register.

19.2.13.1 Normal Prescaler Mode

In Normal prescaler mode the clock fed to the CC8y counter is a normal fixed division by N, accordingly to the value set in the **CC8yPSC**.PSIV register. The values for the possible division values are listed in **Table 19-8**. The **CC8yPSC**.PSIV value is only modified by a SW access. Notice that each slice has a dedicated prescaler value selector (**CC8yPSC**.PSIV), which means that the user can select different counter clocks for every and each Timer Slice (CC8y).

Table 19-8 Timer clock division options

CC8yPSC.PSIV	**Resulting clock**
0000_B	f_{ccu8}
0001_B	$f_{ccu8}/2$
0010_B	$f_{ccu8}/4$
0011_B	$f_{ccu8}/8$
0100_B	$f_{ccu8}/16$
0101_B	$f_{ccu8}/32$
0110_B	$f_{ccu8}/64$
0111_B	$f_{ccu8}/128$
1000_B	$f_{ccu8}/256$
1001_B	$f_{ccu8}/512$
1010_B	$f_{ccu8}/1024$
1011_B	$f_{ccu8}/2048$
1100_B	$f_{ccu8}/4096$
1101_B	$f_{ccu8}/8192$
1110_B	$f_{ccu8}/16384$
1111_B	$f_{ccu8}/32768$

19.2.13.2 Floating Prescaler Mode

The floating prescaler mode can be used individually in each slice by setting the register **CC8yTC**.FPE = 1_B. With this mode, the user can not only achieve a better precision on the counter clock for compare operations but also reduce the SW read access for the capture mode.

The floating prescaler mode contains additionally to the initial value register, **CC8yPSC**.PSIV, a compare register, **CC8yFPC**.PCMP with an associated shadow register.

Figure 19-73 shows the structure of the prescaler in floating mode when the specific slice is in compare mode (no external signal is used for capture). In this mode, the value of the clock division is incremented by 1_D every time that a timer overflow/underflow (overflow if in Edge Aligned Mode, underflow if in Center Aligned Mode) occurs.

In this mode, the Compare Match (both channels) from the timer is AND gated with the Compare Match of the prescaler and every time that this event occurs, the value of the clock division is updated with the **CC8yPSC**.PSIV value in the immediately next timer overflow/underflow event.

To use just one compare channel to control the floating prescaler, the other compare channel must be disabled. To due this, the compare value, **CC8yCR1** or **CC8yCR2** (depending on which channel is used) needs to be set with a value bigger than the period, **CC8yPR**. This means that in edge aligned more, the maximum value for the timer period is 65534_D, because the compare value of one channel needs to be set to 65535_D.

The shadow transfer of the floating prescaler compare value, **CC8yFPC**.PCMP, is done following the same rules described on **Section 19.2.5.2**.

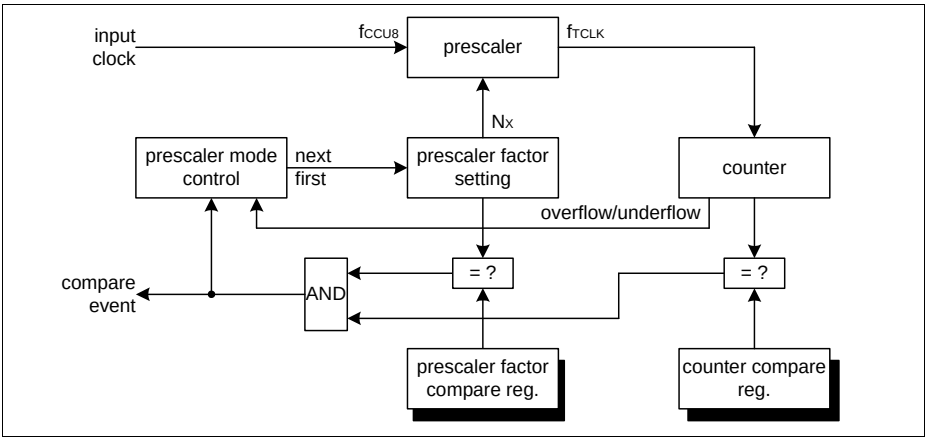


Figure 19-73 Floating prescaler in compare mode overview

When the specific CCU8 is operating in capture mode (when at least one external signal is decoded as capture functionality), the actual value of the clock division also needs to be stored every time that a capture event occurs. The floating prescaler can have up to 4 capture registers (the maximum number of capture registers is dictated by the number of capture registers used in the specific slice).

The clock division value continues to be increment by 1 every time that a timer overflow (in capture mode, the slice is always operating in Edge Aligned Mode) occurs and it is loaded with the PSIV value every time that a capture triggers is detected.

See the **Section 19.2.14** for a full description of the usage of the floating prescaler mode in conjunction with compare and capture modes.

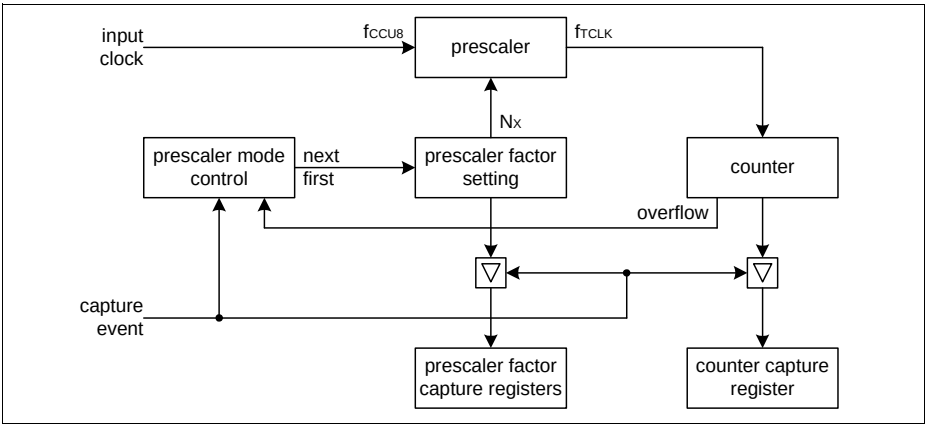


Figure 19-74 Floating Prescaler in capture mode overview

19.2.14 CCU8 Usage

19.2.14.1 PWM Signal Generation

The CCU8 offers a very flexible range in duty cycle configurations. This range is comprised between 0 to 100%.

To generate a PWM signal with a 100% duty cycle in Edge Aligned Mode, one should program the compare value, **CC8yCR1**.CR1/**CC8yCR2**.CR2, to 0000_H, **Figure 19-75**.

In the same manner a 100% duty cycle signal can be generated in Center Aligned Mode, **Figure 19-76**.

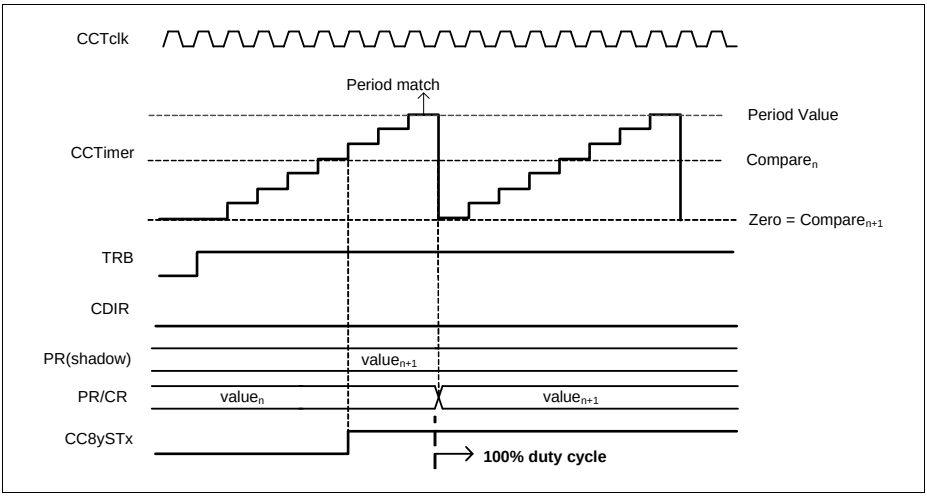


Figure 19-75 PWM with 100% duty cycle - Edge Aligned Mode

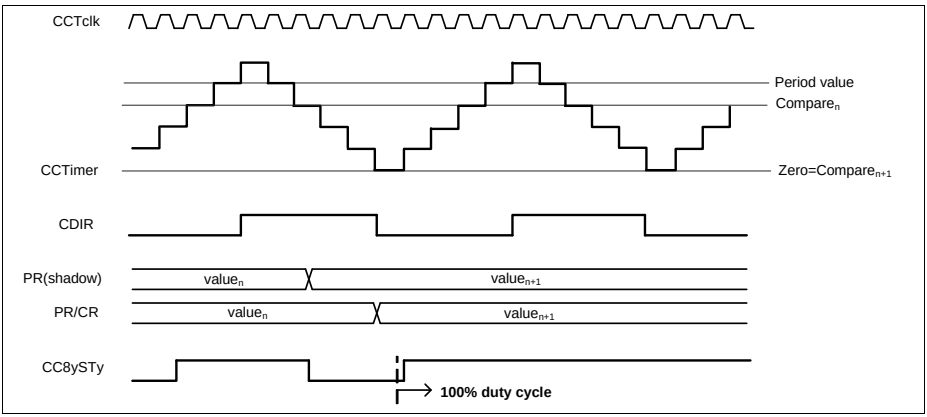


Figure 19-76 PWM with 100% duty cycle - Center Aligned Mode

To generate a PWM signal with 0% duty cycle in Edge Aligned Mode, the compare register should be set with the value programmed into the period value plus 1. In the case that the timer is being used with the full 16 bit capability (counting from 0 to 65535), setting a value bigger than the period value into the compare register is not possible and therefore the smallest duty cycle that can be achieved is 1/FS, see **Figure 19-77**.

In Center Aligned Mode, the counter is never running from 0_D to 65535_D, due to the fact that it has to overshoot for one clock cycle the value set in the period register. Therefore

the user never has a FS counter, which means that generating a 0% duty cycle signal is always possible by setting a value in the compare register bigger than the one programmed into the period register, see **Figure 19-78**.

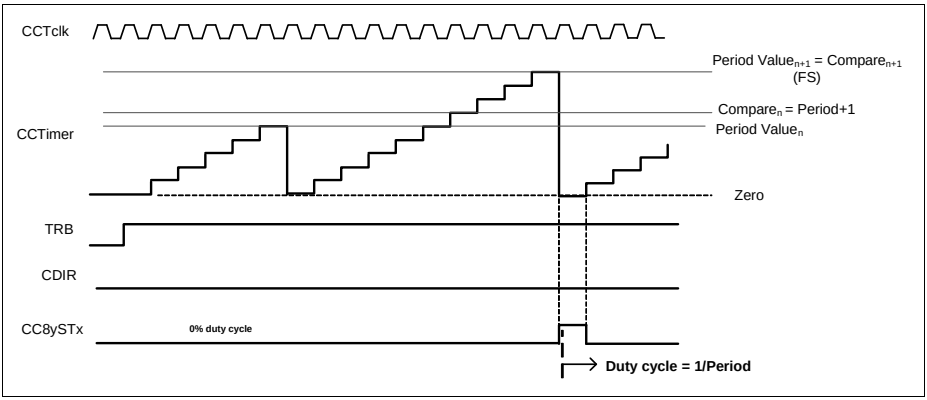


Figure 19-77 PWM with 0% duty cycle - Edge Aligned Mode

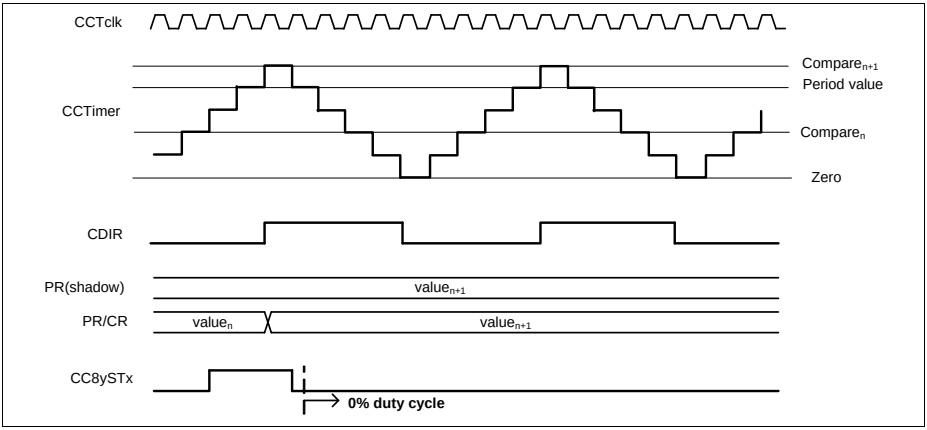


Figure 19-78 PWM with 0% duty cycle - Center Aligned Mode

19.2.14.2 Prescaler Usage

In Normal Prescaler Mode, the frequency of the f_{tclk} fed to the specific CC8y is chosen from the **Table 19-8**, by setting the **CC8yPSC**.PSIV with the required value.

In Floating Prescaler Mode, the frequency of the f_{tclk} can be modified over a selected timeframe, within the values specified in **Table 19-8**. This mechanism is specially useful if, in capture mode, the dynamic of the capture triggers is very slow or unknown.

In Capture Mode, the Floating Prescaler value is incremented by 1_D every time that a timer overflow happens and it is set with the initial programmed value when a capture event happens, see **Figure 19-79**.

When using the Floating Prescaler Mode in Capture Mode, the timer should be cleared each time that a capture event happens, **CC8yTC**.CAPC = 11_B. By operating the Capture mode in conjunction with the Floating Prescaler, even for capture signals that have a periodicity bigger that 16 bits, it is possible to use just a single CCU8 slice without monitoring the interrupt events triggered by the timer overflow. For this the user just needs to know what is the timer capture value and the actual prescaler configuration at the time that the capture event occurred. These values are contained in each CC8yCxV register.

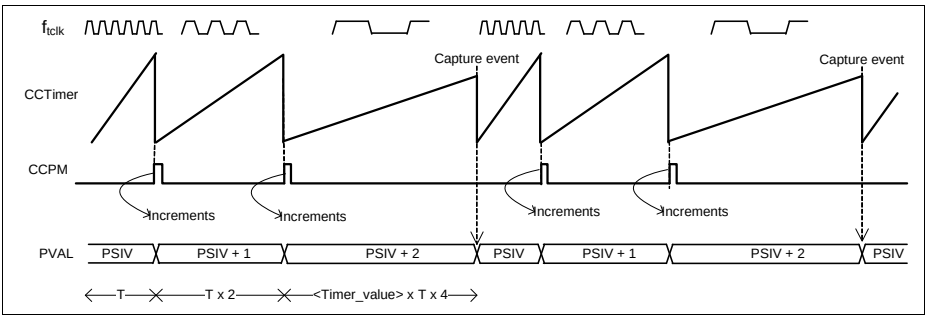


Figure 19-79 Floating Prescaler capture mode usage

When used in Compare Mode, the Floating Prescaler function may be used to achieve a fractional PWM frequency or to perform some frequency modulation.

The same incrementing by 1_D mechanism is done every time that a overflow/underflow of the Timer occurs (from any of the compare channels) and the actual Prescaler value doesn't match the one programmed into the **CC8yFPC**.PCMP register.

When a Compare Match from the Timer (from any of the compare channels) occurs and the actual Prescaler value is equal to the one programmed on the **CC8yFPC**.PCMP register, then the Prescaler value is set with the initial value, **CC8yPSC**.PSIV, in the immediately next occurrence of a timer overflow/underflow.

To use just one compare channel to control the floating prescaler, the other compare channel must be disabled. To due this, the compare value, **CC8yCR1** or **CC8yCR2** (depending on which channel is used) needs to be set with a value bigger than the period, **CC8yPR**. This means that in edge aligned more, the maximum value for the timer period is 65534_D, because the compare value of one channel needs to be set to 65535_D (so the compare match of the associated channel is disabled).

In **Figure 19-80**, the Compare value of the Floating Prescaler was set to PSIV + 2. Every time that a timer overflow occurs, the value of the Prescaler is incremented by 1, which

means that if we give f_{tclk} as the reference frequency for the **CC8yPSC**.PSIV value, we have $f_{tclk}/2$ for **CC8yPSC**.PSIV + 1 and $f_{tclk}/4$ for **CC8yPSC**.PSIV + 2. With the period overtime of the counter becomes:

Period = $(1/f_{tclk}+2/f_{tclk}+4/f_{tclk})/3$

The same mechanism is used in Center Aligned Mode, but to keep the rising arcade and falling arcade always symmetrical, instead of the overflow of the timer, the underflow is used, see **Figure 19-81**.

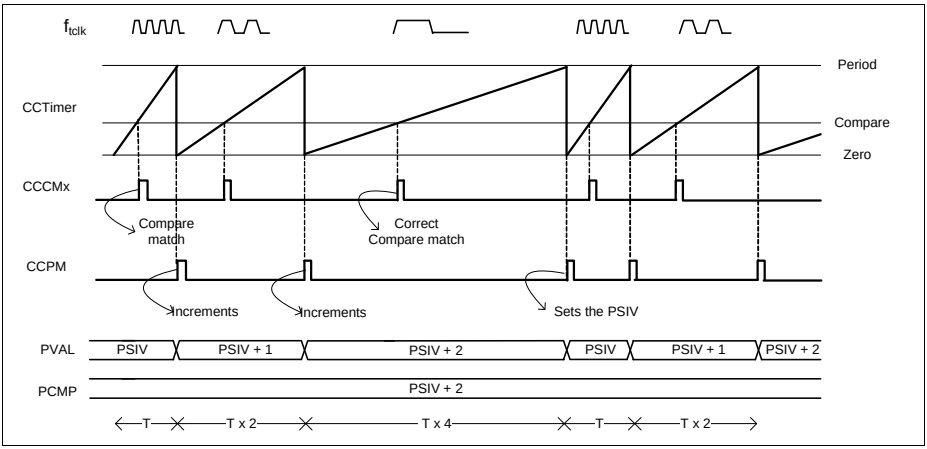


Figure 19-80 Floating Prescaler compare mode usage - Edge Aligned

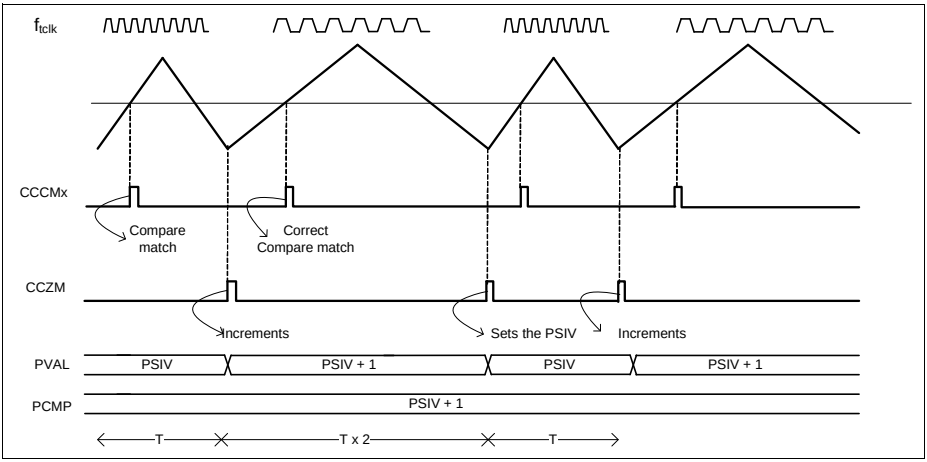


Figure 19-81 Floating Prescaler compare mode usage - Center Aligned

19.2.14.3 PWM Dither

The Dither function can be used to achieve a very fine precision on the periodicity of the output state in compare mode. The value set in the dither compare register, **CC8yDIT**.DCV is crosschecked against the actual value of the dither counter and every time that the dither counter is smaller than the comparison value one of the follows actions is taken:

- The period is extended for 1 clock cycle - **CC8yTC**.DITHE = 01_B; in edge aligned mode
- The period is extended for 2 clock cycles - **CC8yTC**.DITHE = 01_B; in center aligned mode
- The comparison match while counting up (**CC8yTCST**.CDIR = 0_B) is delayed (this means that the status bit is going to stay in the SET state 1 cycle less) for 1 clock cycle - **CC8yTC**.DITHE = 10_B;
- The period is extended for 1 clock cycle and the comparison match while counting up is delayed for 1 clock cycle - **CC8yTC**.DITHE = 11_B; in edge aligned mode
- The period is extended for 2 clock cycles and the comparison match while counting up is delayed for 1 clock cycle; center aligned mode

The bit reverse counter distributes the number programmed in the **CC8yDIT**.DCV throughout 16 timer periods.

Table 19-9, describes the bit reverse distribution versus the programmed value on the **CC8yDIT**.DCV field. The fields marked as '0' indicate that in that counter period, one of the above described actions, is going to be performed.

Table 19-9 Bit reverse distribution

	DCV															
Dither counter	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
4	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
C	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
2	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
A	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0
6	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
E	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0

Table 19-9 Bit reverse distribution (cont'd)

	DCV															
Dither counter	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
5	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
D	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
3	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
B	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
7	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
F	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

The bit reverse distribution versus the programmed **CC8yDIT**.DCV value results in the following values for the Period and duty cycle:

DITHE = 01_B

Period = [(16 - DCV) x T + DCV x (T + 1)]/16; in Edge Aligned Mode (19.10)

Duty cycle = [(16 - DCV) x d/T + DCV x (d+1)/(T + 1)]/16; in Edge Aligned Mode (19.11)

Period = [(16 - DCV) x T + DCV x (T + 2)]/16; in Center Aligned Mode (19.12)

Duty cycle = [(16 - DCV) x d/T + DCV x (d+2)/(T + 2)]/16; in Center Aligned Mode (19.13)

DITHE = 10_B

Period = T ; in Edge Aligned Mode (19.14)

Duty cycle = [(16 - DCV) x d/T + DCV x (d-1)/T]/16; in Edge Aligned Mode (19.15)

Period = T ; in Center Aligned Mode (19.16)

Duty cycle = [(16 - DCV) x d/T + DCV x (d-1)/T]/16; in Center Aligned Mode (19.17)

DITHE = 11_B

Period = [(16 - DCV) x T + DCV x (T + 1)]/16; in Edge Aligned Mode (19.18)

Duty cycle = [(16 - DCV) x d/T + DCV x d/(T + 1)]/16; in Edge Aligned Mode (19.19)

Period = [(16 - DCV) x T + DCV x (T + 2)]/16; in Center Aligned Mode (19.20)

Duty cycle = [(16 - DCV) x d/T + DCV x (d+1)/(T + 2)]/16; in Center Aligned Mode(19.21)

where:

T - Original period of the signal, see **Section 19.2.5.1**

d - Original duty cycle of the signal, see **Section 19.2.5.1**

19.2.14.4 Capture Mode Usage

Each slice has the possibility of using 2 or 4 capture registers. Using only 2 capture registers means that only 1 Event was linked to a captured trigger. To use the four capture registers, both capture triggers need to be mapped into an Event (it can be the same signal with different edges selected or two different signals) or the **CC8yTC**.SCE field needs to be set to 1_B, which enables the linking of the 4 capture registers.

The internal slice mechanism for capturing is the same for the capture trigger 1 or capture trigger 0.

Different Capture Events - SCE = 0_B

Capture trigger 1 (CCcapt1) is appointed to the capture register 2, **CC8yC2V** and capture register 3, **CC8yC3V**, while trigger 0 is appointed to capture register 1, **CC8yC1V** and 0, **CC8yC0V**.

In each CCcapt0 event, the timer value is stored into **CC8yC1V** and the value of the **CC8yC1V** is transferred into the **CC8yC0V**.

In each CCcapt1 event, the timer value is stored into capture register **CC8yC3V** and the value of the capture register **CC8yC3V** is transferred into **CC8yC2V**.

The previous capture/transfer mechanism only happens if the specific register is not full. A capture register becomes full when receives a new value and becomes empty after the SW has read back the value.

The full flag is cleared every time that the SW reads back the **CC8yC0V**, **CC8yC1V**, **CC8yC2V** or **CC8yC3V** register. The SW can be informed of a new capture trigger by enabling the interrupt source linked to the specific Event. This means that every time that a capture is made an interrupt pulse is generated.

In the case that the Floating Prescaler Mode is being used, the actual value of the clock division is also stored in the capture register (CC8yCxV).

Figure 19-82 shows an example of how the capture/transfer may be used in a Timer Slice that is using a external signal as count function (to measure the velocity of a rotating device), and an equidistant capture trigger that is used to dictate the timestamp

for the velocity calculation (two Timer waveforms are plotted, one that exemplifies the clearing of the timer in each capture event and another without the clearing function active).

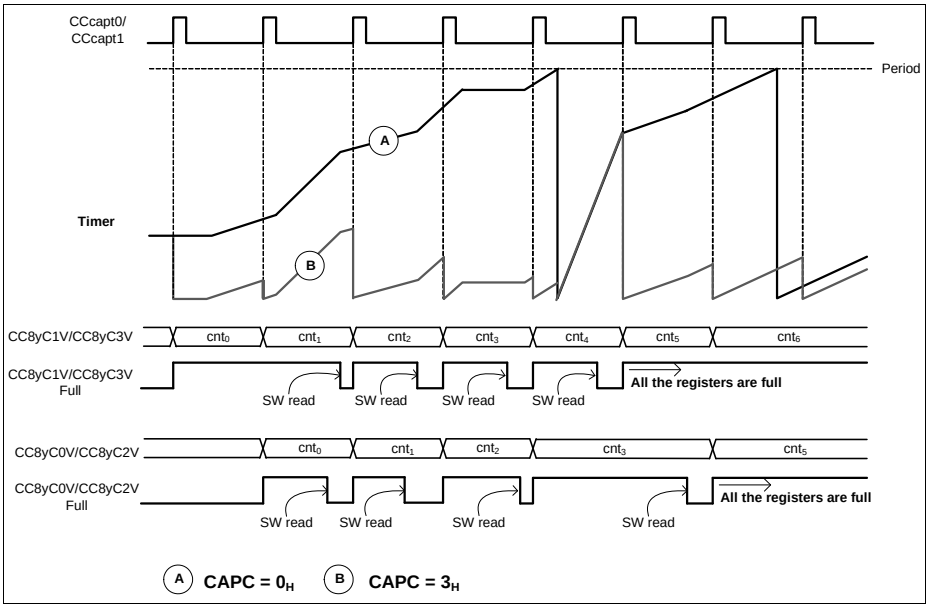


Figure 19-82 Capture mode usage - single channel

Same Capture Event - SCE = 1_B

If **CC8yTC**.SCE is set to 1_B, all the four capture registers are chained together, emulating a fifo with a depth of 4. In this case, only the capture trigger 1, CCcapt1, is used to perform a capture event.

As an example for this mode, one can consider the case where one Timer Slice is being used in capture mode with SCE = 1_B, with another external signal that controls the counting. This timer slice can be incremented at different speeds, depending on the frequency of the counting signal.

An additional Timer Slice is used to control the capture trigger, dictating the time stamp for the capturing.

A simple scheme for this can be seen in **Figure 19-83**. The CC80ST output of slice 0 was used as capture trigger in the CC81 slice (active on rising and falling edge). The CC80ST output is used as known timebase marker, while the slice timer used for capture is being controlled by external events, e.g. external count.

Due to the fact that we have 4 capture registers available, every time that the SW reads back the complete set of values, 3 speed profiles can be measured.

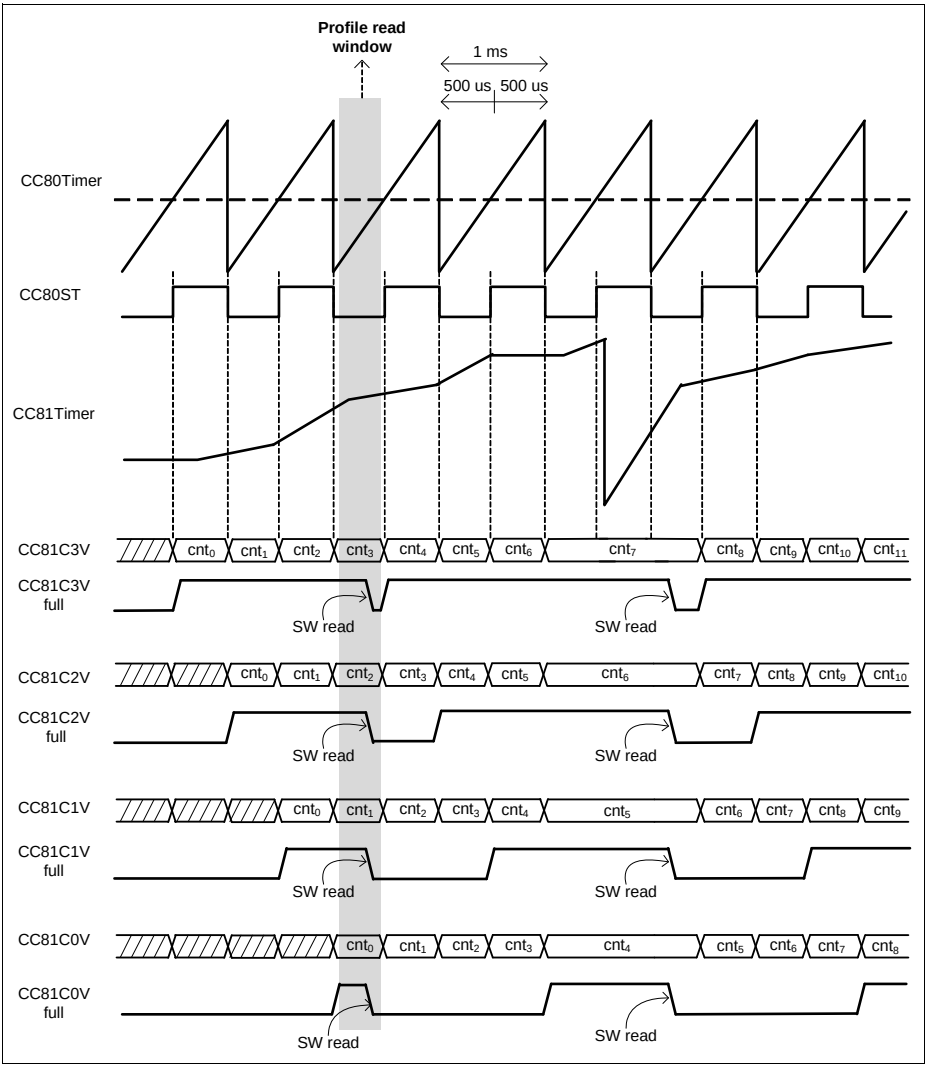


Figure 19-83 Three Capture profiles - CC8yTC.SCE = 1_B

To calculate the three different profiles in **Figure 19-83**, the 4 capture registers need to be read during the pointed read window. After that, the profile calculation is done:

Profile 1 = $CC81C1V_{info}$ - $CC81C0V_{info}$

Profile 2 = $CC81C2V_{info}$ - $CC81C1V_{info}$

Profile 3= $CC81C3V_{info}$ - $CC81C2V_{info}$

Note: This is an example and therefore several Timer Slice configurations and software loops can be implemented.

Extended Read Back Mode

When multiple Timer Slices need to be programmed into capture mode, it may not be suitable to distribute them over several CCU8 modules. This may be due to resource optimization or availability of Direct Memory Access (DMA) channels.

A simple way to overcome this issue, is to use the Extended Capture Read functionality of CCU8. This mode can be programmed independently for each and every Timer Slice via the **CC8yTC**.ECM bit field.

The advantage of this mode is that there is only one associated read address for all the capture registers (notice that the individual capture registers are still accessible), the **ECRD**. With this one can achieve DMA channel compression and a better Timer Slice resource optimization throughout the entire device.

Figure 19-84 exemplifies the usage of the Extended Capture Read function. In this example we have three different Timer Slices that are used to monitor three different applications (in capture mode). An additional Timer Slice (notice that it doesn't need to be in the same CCU8 module) is used to trigger the DMA read of the capture registers.

The read back trigger periodicity can also be updated on the fly to adjust to different system states or operation modes.

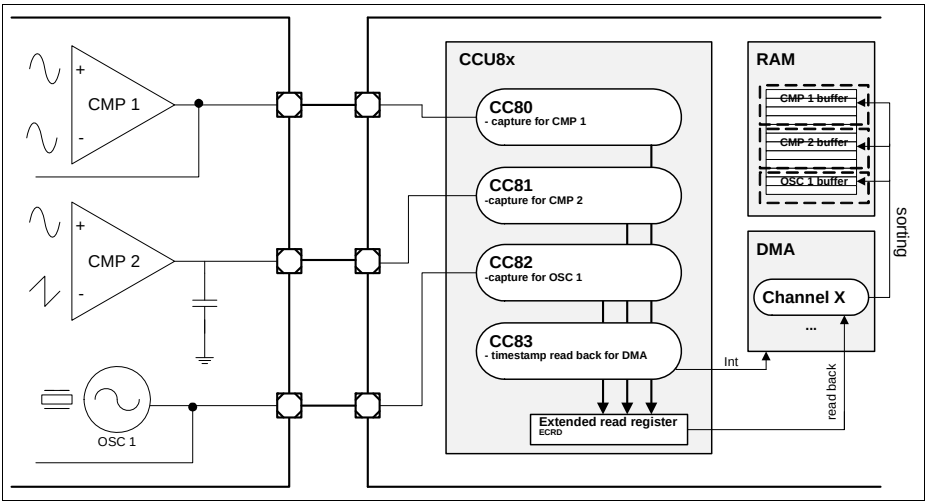


Figure 19-84 Extended read usage scheme example

Every time that the software reads back the **ECRD** register, the CCU8 returns the value of a specific capture register that contains new captured data. The read access of the capture registers follows a circular scheme that is maintained internally by the CCU8, **Figure 19-85**.

For the timer slices that are in capture mode but do not have **CC8yTC**.ECM = 1_B, their captured register values are also read back through the **ECRD**. However, the full flag of the capture registers is not cleared (it is only cleared via a read access to the specific CC8yCxV register).

Only the capture registers of the slices with **CC8yTC**.ECM = 1_B have their full flag cleared with a read access via **ECRD**.

On **Figure 19-86** an example time line is given, in which all the slices were programmed to use extended capture mode, **CC8yTC**.ECM = 1_B. In this example, one can see that the CCU8 doesn't keep memory of which was the first or last captured value between the Timer Slices. Like described on **Figure 19-85**, the read back pointer is incremented until a capture register that has the full flag set, is found.

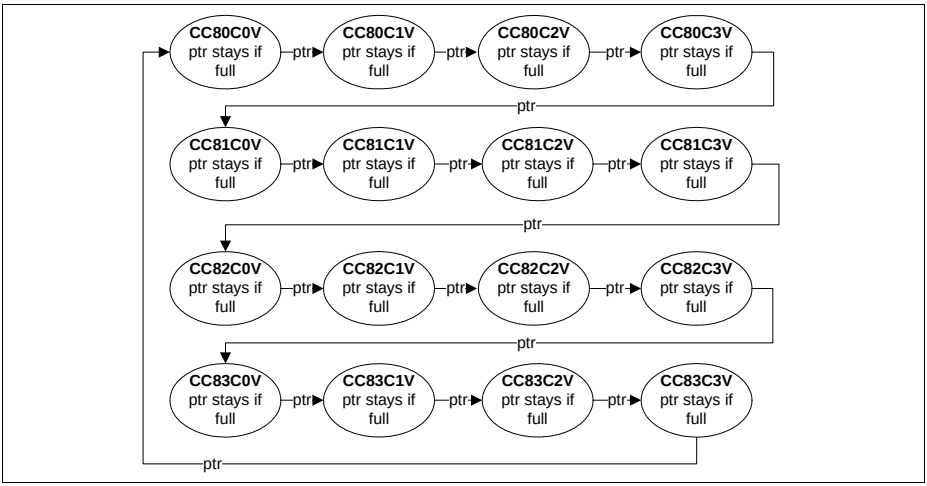


Figure 19-85 Extended Capture read back

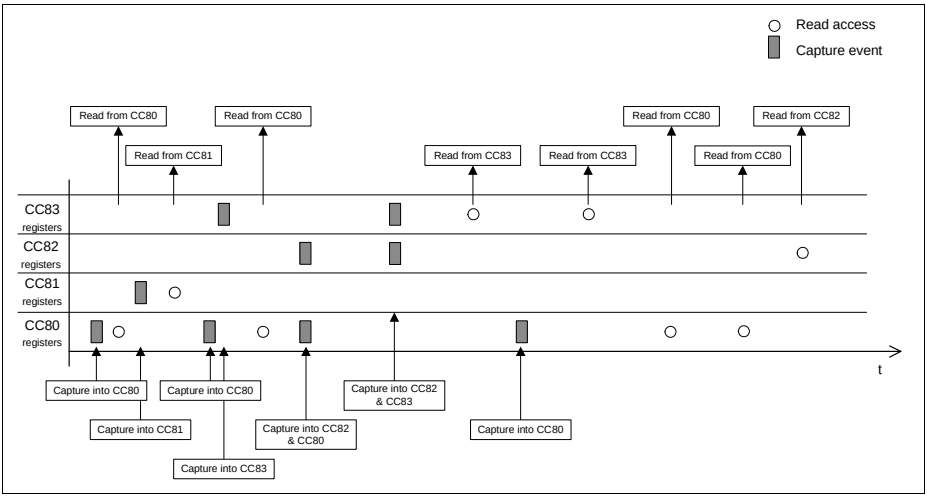


Figure 19-86 Extended Capture Access Example

19.2.14.5 Parity Checker Usage

The parity checker function available on the CCU8 uses of one CCU4 timer slice, to control the delay between the update of the outputs and the consequent update on the external switch/driver.

This CCU4 timer slice is configured to work in edge aligned mode, with single shot mode active and with a configured external flush & start function. This function is connected to the parity checker update output signal, CCU8x.IGBTO. The timer slice compare and period values need to be programmed accordingly to the delay that is foreseen between the outputs of the CCU8 and the consequent update on the driver/switch parity output. The connections between the CCU8, CCU4 and the external HW can be seen on **Figure 19-87**.

Figure 19-88 shows the timing waveforms for an usage example of the parity checker (case of even parity, **GPCHK**.PCTS field takes the default value). In this example only two outputs of CCU8 were considered to avoid extreme complexity on the diagram.

Every time an update of the selected outputs leads to a modification of the value **GPCHK**.PCST (parity checker status), or in other words, every time the result of the XOR chain changes, a trigger is generated on the CCU8x.IGBTO. This signal, as described previously, is used as a flush & start for a CCU4 slice.

When the CCU4 slice timer reaches the compare value, the specific status output, CCU4x.STy is asserted. After this elapsed delay, the value on the CCU8x.GPy1 (the parity value coming from the external HW) is crosschecked against the result of the internal XOR chain (**GPCHK**.PCST). If the values are different, a Service Request pulse can be generated, if it was previously enabled.

Notice that when an update of the CCU8 outputs leads to an equal result on the XOR chain, the CCU4 slice is not retriggered (the output parity from the external hardware remains with the same value).

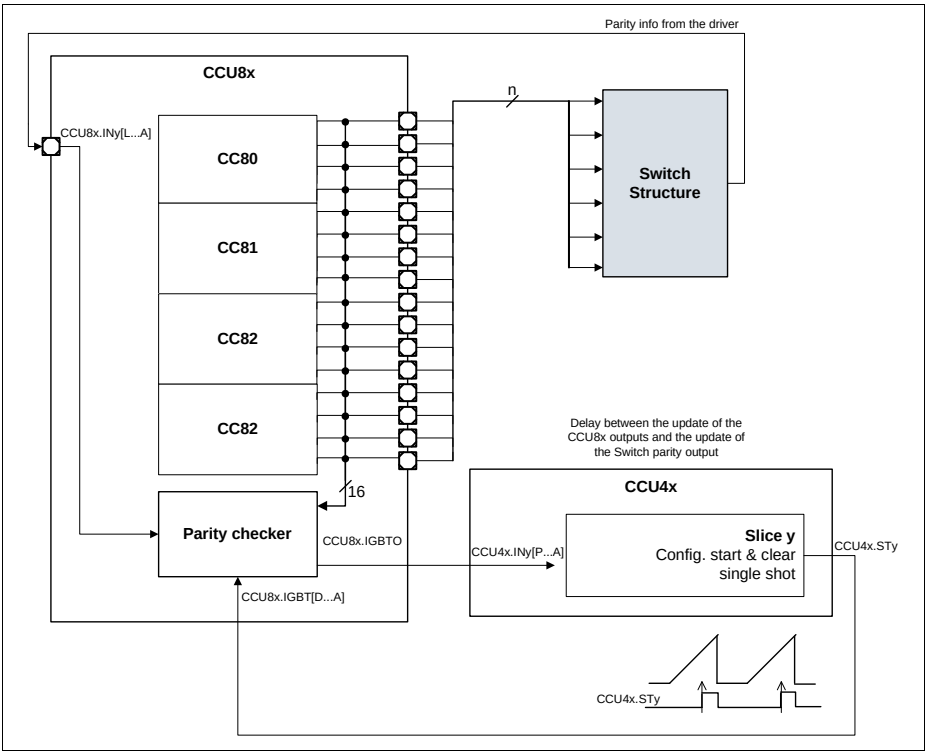


Figure 19-87 Parity Checker connections

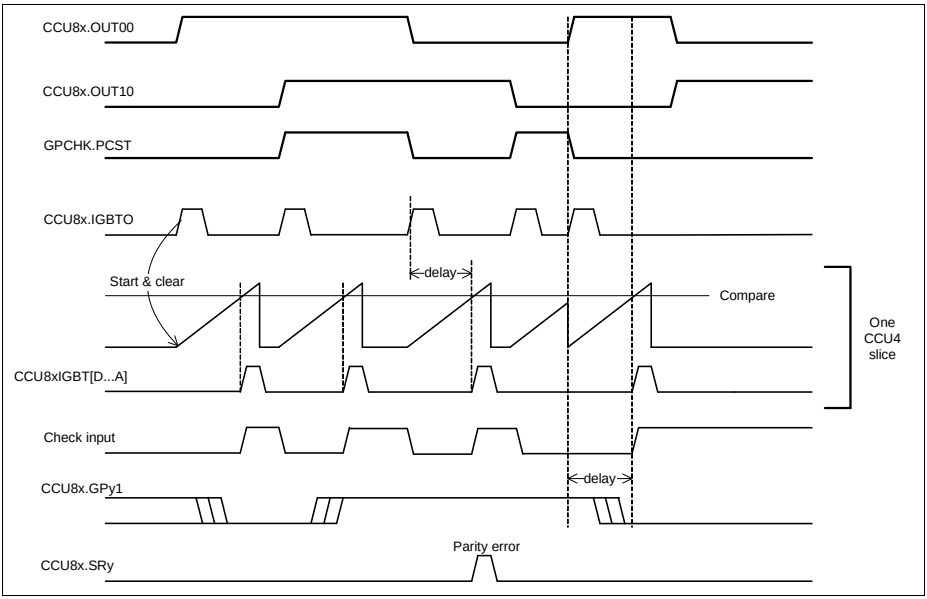


Figure 19-88 Parity Checker timing example

19.3 Service Request Generation

Each CCU8 slice has a interrupt structure as the one see in **Figure 19-89**. The register **CC8yINTS** is the status register for the interrupt sources. Each dedicated interrupt source can be set or cleared by SW, by writing into the specific bit in the **CC8ySWS** and **CC8ySWR** registers respectively.

Each interrupt source can be enabled/disabled via the **CC8yINTE** register. An enabled interrupt source will always generate a pulse on the service request line even if the specific status bit was not cleared. **Table 19-10** describes the interrupt sources of each CCU8 slice.

The interrupt sources, Period Match while counting up and one Match while counting down are ORed together. The same mechanism is applied to the Compare Match while counting up and Compare Match while counting down of both compare channels.

The interrupt sources for the external events are directly linked with the configuration set on the **CC8yINS**.EVxEM. If an event is programmed to be active on both edges, that means that service request pulse is going to be generated when any transition on the external signal is detected. If the event is linked with a level function, the **CC8yINS**.EVxEM still can be programmed to enable a service request pulse. The TRAP

event doesn't need any extra configuration for generating the service request pulse when the slice enters the TRAP state.

Table 19-10 Interrupt sources

Signal	Description
CCINEV0_E	Event 0 edge(s) information from event selector. Used when an external signal should trigger an interrupt.
CCINEV1_E	Event 1 edge(s) information from event selector. Used when an external signal should trigger an interrupt (It also can be the parity checker pattern fail information, if the parity checker was enabled).
CCINEV2_E	Event 2 edge(s) information from event selector. Used when an external signal should trigger an interrupt.
CCPM_U	Period Match while counting up.
CCCM1_U	Compare Match while counting up from compare channel 1.
CCCM1_D	Compare Match while counting down from compare channel 1.
CCCM2_U	Compare Match while counting up from compare channel 2.
CCCM2_D	Compare Match while counting down from compare channel 2.
CCOM_D	One Match while counting down.
Trap state set	Entering Trap State. Will set the E2AS.

Each of the interrupt events can then be forwarded to one, of the slice's four service request lines, **Figure 19-90**. The value set on the **CC8ySRS** controls which interrupt event is mapped into which service request line.

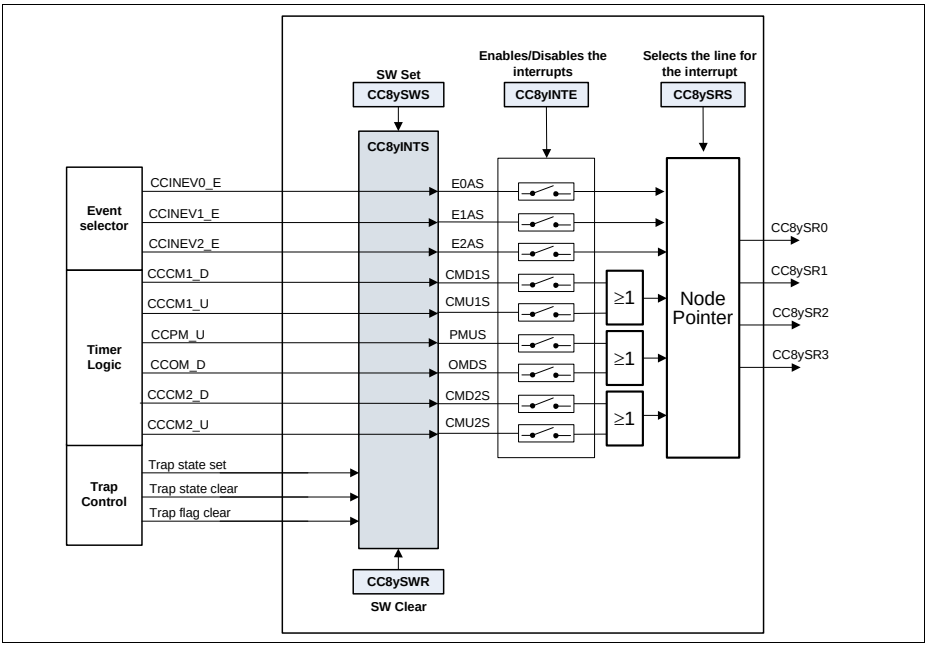


Figure 19-89 Slice interrupt node pointer overview

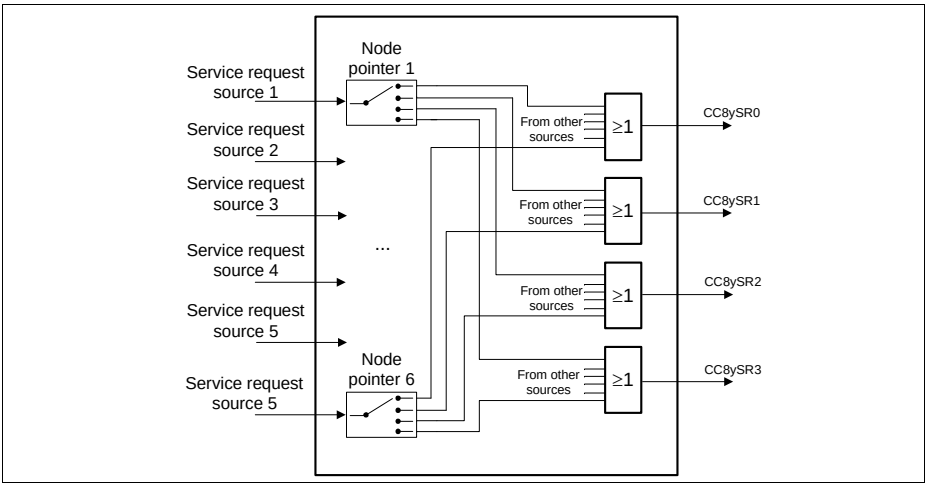


Figure 19-90 Slice interrupt selector overview

The four service request lines of each slice are OR together inside the kernel of the CCU8, see **Figure 19-91**. This means that there are only four service request lines per CCU8, that can have in each line interrupt requests coming from different slices.

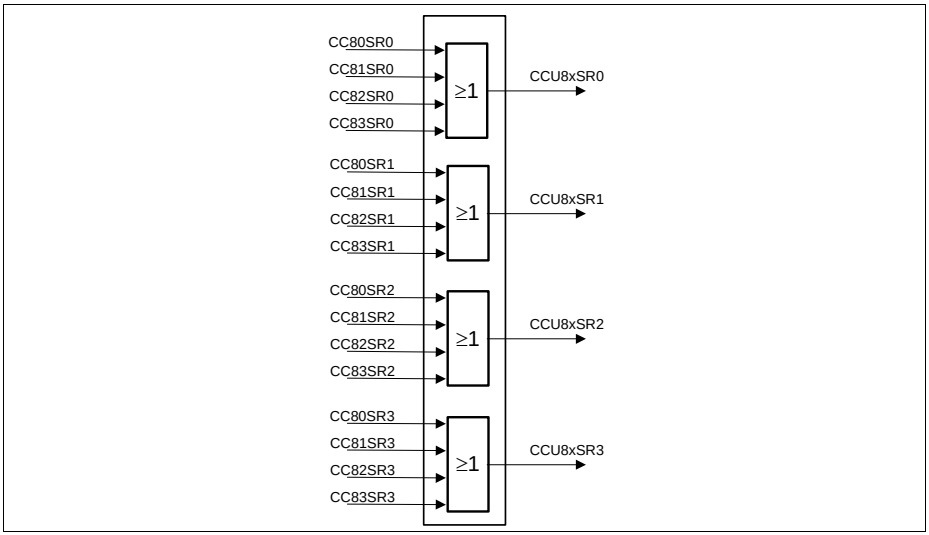


Figure 19-91 CCU8 service request overview

19.4 Debug Behavior

In suspend mode, the functional clocks for all slices as well the prescaler are stopped. The registers can still be accessed by the CPU (read only). This mode is useful for debugging purposes, e.g., where the current device status should be frozen in order to get a snapshot of the internal values. In suspend mode, all the slice counters are stopped. The suspend mode is non-intrusive concerning the register bits. This means register bits are not modified by hardware when entering or leaving the suspend mode.

Entry into suspend mode can be configured at the kernel level by means of the field **GCTRL**.SUSCFG.

The module is only functional after the suspend signal becomes inactive.

19.5 Power, Reset and Clock

The following sections describe the operating conditions, characteristics and timing requirements for the CCU8. All the timing information is related to the module clock, f_{ccu8}.

19.5.1 Clocks

Module Clock

The module clock of the CCU8 module is described in the SCU chapter as f_{CCU}.

The bus interface clock of the CCU8 module is described in the SCU chapter as f_{PERIPH}.

The module clock for the CCU8 is controlled via a specific control bit inside the SCU (System Control Unit), register CLKSET.

It is possible to disable the module clock for the CCU8 via the **GSTAT**, nevertheless, there may be a dependency of the f_{ccu8} through the different CCU8 instances. One should address the SCU Chapter for a complete description of the product clock scheme.

If module clock dependencies exist through different IP instances, then one can disable the module clock internally inside the specific CCU8, by disabling the prescaler (**GSTAT**.PRB = 0_B).

External Clock

It is possible to use an external clock as source for the prescaler, and consequently for all the timer Slices, CC8y. This external source can be connected to one of the CCU8x.CLK[C...A] inputs.

This external source is nevertheless synchronized against f_{ccu8}.

Table 19-11 External clock operating conditions

Parameter	Symbol	Values			Unit	Note / Test Condition
		Min.	Typ.	Max.		
Frequency	f_{eclk}	–	–	$f_{ccu8}/4$	MHz	
ON time	ton_{eclk}	$2T_{ccu8}$ [1)2)]	–	–	ns	
OFF time	$toff_{eclk}$	$2T_{ccu8}$ [1)2)]	–	–	ns	Only the rising edge is used

1) Only valid if the signal was not previously synchronized/generated with the fccu4 clock (or a synchronous clock)

2) 50% duty cycle is not obligatory

19.5.2 Module Reset

Each CCU8 has one reset source. This reset source is handled at system level and it can be generated independently via a system control register, PRSET0/PRSET1 (address SCU chapter for a full description).

After reset release, the complete IP is set to default configuration. The default configuration for each register field is addressed on **Section 19.7**.

19.5.3 Power

The CCU8 is inside the power core domain, therefore no special considerations about power up or power down sequences need to be taken. For a explanation about the different power domains, please address the SCU (System Control Unit) chapter.

An internal power down mode for the CCU8, can be achieved by disabling the clock inside the CCU8 itself. For this one should set the **GSTAT** register with the default reset value (via the idle mode set register, **GIDLS**).

19.6 Initialization and System Dependencies

19.6.1 Initialization Sequence

The initialization sequence for an application that is using the CCU8, should be the following:

1st Step: Apply reset to the CCU8, via the specific SCU bitfield on the PRSET0/PRSET1 register.

2nd Step: Release reset of the CCU8, via the specific SCU bitfield on the PRCLR0/PRCLR1 register.

3rd Step: Enable the CCU8 clock via the specific SCU register, CLKSET.

4th Step: Enable the prescaler block, by writing 1_B to the **GIDLC**.SPRB field.

5th Step: Configure the global CCU8 register **GCTRL**.

6th Step: Configure all the registers related to the required Timer Slice(s) functions, including the interrupt/service request configuration.

7th Step: If needed, configure the startup value for a specific Compare Channel Status, of a Timer Slice, by writing 1_B to the specific **GCSS**.SyTS.

8th Step: Configure the parity checker function if used, by programming the **GPCHK** register

9th Step: Enable the specific timer slice(s), CC8y, by writing 1_B to the specific **GIDLC**.CSyI.

10th Step: For all the Timer Slices that should be started synchronously via SW, the specific system register localized in the SCU, CCUCON, that enables a synchronous timer start should be addressed. The SCU.GSC8x input signal needs to be configured previously as a start function, see **Section 19.2.8.1**.

19.6.2 System Dependencies

Each CCU8 may have different dependencies regarding module and bus clock frequencies. This dependencies should be addressed in the SCU and System Architecture Chapters.

Dependencies between several peripherals, regarding different clock operating frequencies may also exist. This should be addressed before configuring the connectivity between the CCU8 and some other peripheral.

The following topics must be taken into consideration for good CCU8 and system operation:

- CCU8 module clock must be at maximum two times faster than the module bus interface clock
- Module input triggers for the CCU8 must not exceed the module clock frequency (if the triggers are generated internally in the device)
- Module input triggers for the CCU8 must not exceed the frequency dictated in **Section 19.5.1**
- Frequency of the CCU8 outputs used as triggers/functions on other modules, must be crosschecked on the end point
- Applying and removing CCU8 from reset, can cause unwanted operations in other modules. This can occur if the modules are using CCU8 outputs as triggers/functions.

19.7 Registers

Registers Overview

The absolute register address is calculated by adding:

Module Base Address + Offset Address

Table 19-12 Registers Address Space

Module	Base Address	End Address	Note
CCU80	40020000_H	$40023FFF_H$	

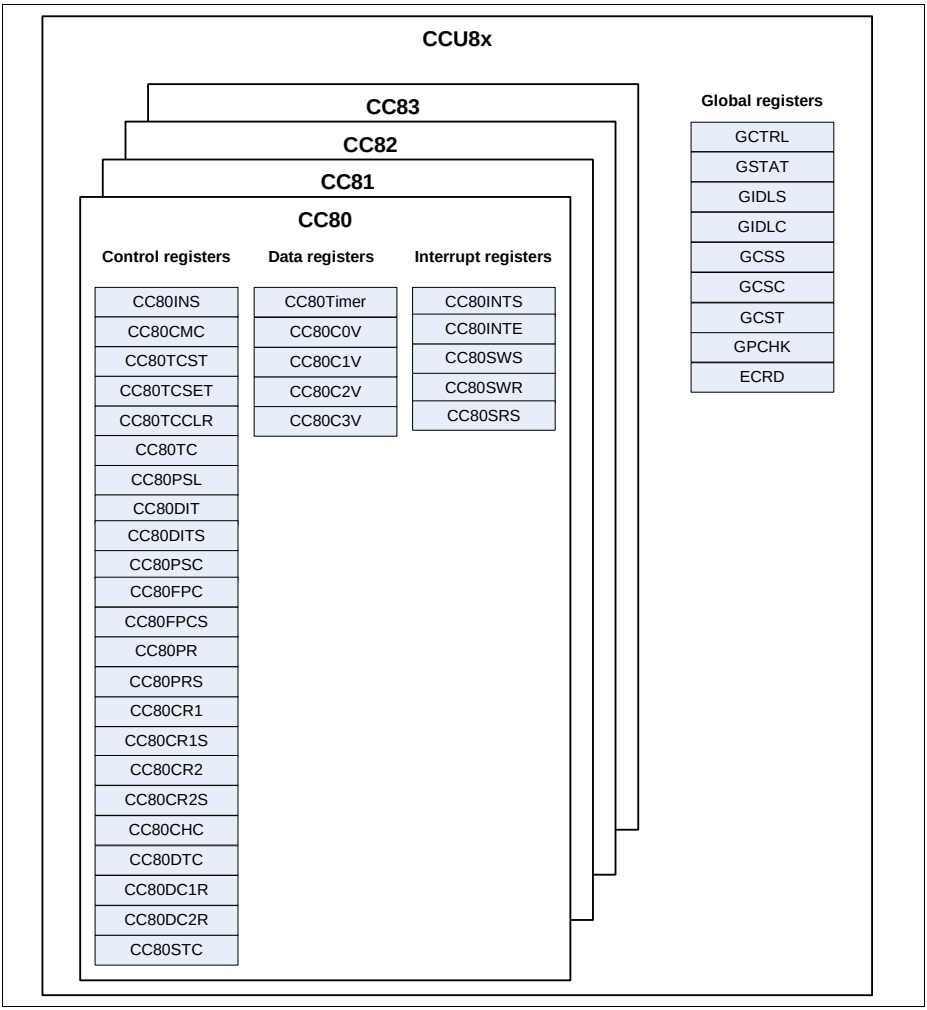


Figure 19-92 CCU8 registers overview

Table 19-13 Register Overview of CCU8

Short Name	Description	Offset Addr.[1]	Access Mode		Description See
			Read	Write	
CCU8 Global Registers					
GCTRL	Module General Control Register	0000_H	U, PV	U, PV	Page 19-106
GSTAT	General Slice Status Register	0004_H	U, PV	BE	Page 19-109
GIDLS	General Idle Enable Register	0008_H	U, PV	U, PV	Page 19-110
GIDLC	General Idle Disable Register	$000C_H$	U, PV	U, PV	Page 19-112
GCSS	General Channel Set Register	0010_H	U, PV	U, PV	Page 19-113
GCSC	General Channel Clear Register	0014_H	U, PV	U, PV	Page 19-116
GCST	General Channel Status Register	0018_H	U, PV	BE	Page 19-119
GPCHK	Parity Checker Configuration Register	$001C_H$	U, PV	U, PV	Page 19-122
ECRD	Extended Read Register	0050_H	U, PV	BE	Page 19-124
MIDR	Module Identification Register	0080_H	U, PV	BE	Page 19-126
CC80 Registers					
CC80INS	Input Selector Unit Configuration	0100_H	U, PV	U, PV	Page 19-126
CC80CMC	Connection Matrix Configuration	0104_H	U, PV	U, PV	Page 19-128
CC80TCST	Timer Run Status	0108_H	U, PV	BE	Page 19-132
CC80TCSET	Timer Run Set	$010C_H$	U, PV	U,PV	Page 19-133
CC80TCCLR	Timer Run Clear	0110_H	U, PV	U, PV	Page 19-134
CC80TC	General Timer Configuration	0114_H	U, PV	U, PV	Page 19-135
CC80PSL	Output Passive Level Configuration	0118_H	U, PV	U, PV	Page 19-140
CC80DIT	Dither Configuration	$011C_H$	U, PV	BE	Page 19-141
CC80DITS	Dither Shadow Register	0120_H	U, PV	U, PV	Page 19-142

Table 19-13 Register Overview of CCU8 (cont'd)

Short Name	Description	Offset Addr.[1)]	Access Mode		Description See
			Read	Write	
CC80PSC	Prescaler Configuration	0124_H	U, PV	U, PV	Page 19-143
CC80FPC	Prescaler Compare Value	0128_H	U, PV	U, PV	Page 19-143
CC80FPCS	Prescaler Shadow Compare Value	$012C_H$	U, PV	U, PV	Page 19-144
CC80PR	Timer Period Value	0130_H	U, PV	BE	Page 19-145
CC80PRS	Timer Period Shadow Value	0134_H	U, PV	U, PV	Page 19-146
CC80CR1	Timer Compare Value for Channel 1	0138_H	U, PV	BE	Page 19-146
CC80CR1S	Timer Compare Shadow Value for Channel 1	$013C_H$	U, PV	U, PV	Page 19-147
CC80CR2	Timer Compare Value for Channel 2	0140_H	U, PV	BE	Page 19-148
CC80CR2S	Timer Compare Shadow Value for Channel 2	0144_H	U, PV	U, PV	Page 19-149
CC80CHC	Channel Control	0148_H	U, PV	U, PV	Page 19-150
CC80DTC	Dead Time Control	$014C_H$	U, PV	U, PV	Page 19-152
CC80DC1R	Channel 1 Dead Time Counter Values	0150_H	U, PV	U, PV	Page 19-153
CC80DC2R	Channel 2 Dead Time Counter Values	0154_H	U, PV	U, PV	Page 19-154
CC80TIMER	Timer Current Value	0170_H	U, PV	U, PV	Page 19-154
CC80C0V	Capture Register 0 Value	0174_H	U, PV	BE	Page 19-155
CC80C1V	Capture Register 1 Value	0178_H	U, PV	BE	Page 19-156
CC80C2V	Capture Register 2 Value	$017C_H$	U, PV	BE	Page 19-157
CC80C3V	Capture Register 3 Value	0180_H	U, PV	BE	Page 19-158
CC80INTS	Interrupt Status	$01A0_H$	U, PV	BE	Page 19-159
CC80INTE	Interrupt Enable	$01A4_H$	U, PV	U, PV	Page 19-161
CC80SRS	Interrupt Configuration	$01A8_H$	U, PV	U, PV	Page 19-163
CC80SWS	Interrupt Status Set	$01AC_H$	U, PV	U, PV	Page 19-165
CC80SWR	Interrupt Status Clear	$01B0_H$	U, PV	U, PV	Page 19-167
CC80STC	Shadow Transfer Control	$01B4_H$	U, PV	U, PV	Page 19-169

Table 19-13 Register Overview of CCU8 (cont'd)

Short Name	Description	Offset Addr.[1)]	Access Mode		Description See
			Read	Write	
CC81 Registers					
CC81INS	Input Selector Unit Configuration	0200_H	U, PV	U, PV	Page 19-126
CC81CMC	Connection Matrix Configuration	0204_H	U, PV	U, PV	Page 19-128
CC81TCST	Timer Run Status	0208_H	U, PV	BE	Page 19-132
CC81TCSET	Timer Run Set	$020C_H$	U, PV	U,PV	Page 19-133
CC81TCCLR	Timer Run Clear	0210_H	U, PV	U, PV	Page 19-134
CC81TC	General Timer Configuration	0214_H	U, PV	U, PV	Page 19-135
CC81PSL	Output Passive Level Configuration	0218_H	U, PV	U, PV	Page 19-140
CC81DIT	Dither Configuration	$021C_H$	U, PV	BE	Page 19-141
CC81DITS	Dither Shadow Register	0220_H	U, PV	U, PV	Page 19-142
CC81PSC	Prescaler Configuration	0224_H	U, PV	U, PV	Page 19-143
CC81FPC	Prescaler Compare Value	0228_H	U, PV	U, PV	Page 19-143
CC81FPCS	Prescaler Shadow Compare Value	$022C_H$	U, PV	U, PV	Page 19-144
CC81PR	Timer Period Value	0230_H	U, PV	BE	Page 19-145
CC81PRS	Timer Period Shadow Value	0234_H	U, PV	U, PV	Page 19-146
CC81CR1	Timer Compare Value for Channel 1	0238_H	U, PV	BE	Page 19-146
CC81CR1S	Timer Compare Shadow Value for Channel 1	$023C_H$	U, PV	U, PV	Page 19-147
CC81CR2	Timer Compare Value for Channel 2	0240_H	U, PV	BE	Page 19-148
CC81CR2S	Timer Compare Shadow Value for Channel 2	0244_H	U, PV	U, PV	Page 19-149
CC81CHC	Channel Control	0248_H	U, PV	U, PV	Page 19-150
CC81DTC	Dead Time Control	$024C_H$	U, PV	U, PV	Page 19-152
CC81DC1R	Channel 1 Dead Time Counter Values	0250_H	U, PV	U, PV	Page 19-153

Table 19-13 Register Overview of CCU8 (cont'd)

Short Name	Description	Offset Addr.[1)]	Access Mode		Description See
			Read	Write	
CC81DC2R	Channel 2 Dead Time Counter Values	0254_H	U, PV	U, PV	Page 19-154
CC81TIMER	Timer Current Value	0270_H	U, PV	U, PV	Page 19-154
CC81C0V	Capture Register 0 Value	0274_H	U, PV	BE	Page 19-155
CC81C1V	Capture Register 1 Value	0278_H	U, PV	BE	Page 19-156
CC81C2V	Capture Register 2 Value	$027C_H$	U, PV	BE	Page 19-157
CC81C3V	Capture Register 3 Value	0280_H	U, PV	BE	Page 19-158
CC81INTS	Interrupt Status	$02A0_H$	U, PV	BE	Page 19-159
CC81INTE	Interrupt Enable	$02A4_H$	U, PV	U, PV	Page 19-161
CC81SRS	Interrupt Configuration	$02A8_H$	U, PV	U, PV	Page 19-163
CC81SWS	Interrupt Status Set	$02AC_H$	U, PV	U, PV	Page 19-165
CC81SWR	Interrupt Status Clear	$02B0_H$	U, PV	U, PV	Page 19-167
CC81STC	Shadow Transfer Control	$02B4_H$	U, PV	U, PV	Page 19-169
CC82 Registers					
CC82INS	Input Selector Unit Configuration	0300_H	U, PV	U, PV	Page 19-126
CC82CMC	Connection Matrix Configuration	0304_H	U, PV	U, PV	Page 19-128
CC82TCST	Timer Run Status	0308_H	U, PV	BE	Page 19-132
CC82TCSET	Timer Run Set	$030C_H$	U, PV	U,PV	Page 19-133
CC82TCCLR	Timer Run Clear	0310_H	U, PV	U, PV	Page 19-134
CC82TC	General Timer Configuration	0314_H	U, PV	U, PV	Page 19-135
CC82PSL	Output Passive Level Configuration	0318_H	U, PV	U, PV	Page 19-140
CC82DIT	Dither Configuration	$031C_H$	U, PV	BE	Page 19-141
CC82DITS	Dither Shadow Register	0320_H	U, PV	U, PV	Page 19-142
CC82PSC	Prescaler Configuration	0324_H	U, PV	U, PV	Page 19-143
CC82FPC	Prescaler Compare Value	0328_H	U, PV	U, PV	Page 19-143
CC82FPCS	Prescaler Shadow Compare Value	$032C_H$	U, PV	U, PV	Page 19-144

Table 19-13 Register Overview of CCU8 (cont'd)

Short Name	Description	Offset Addr.[1)]	Access Mode		Description See
			Read	Write	
CC82PR	Timer Period Value	0330_H	U, PV	BE	Page 19-145
CC82PRS	Timer Period Shadow Value	0334_H	U, PV	U, PV	Page 19-146
CC82CR1	Timer Compare Value for Channel 1	0338_H	U, PV	BE	Page 19-146
CC82CR1S	Timer Compare Shadow Value for Channel 1	$033C_H$	U, PV	U, PV	Page 19-147
CC82CR2	Timer Compare Value for Channel 2	0340_H	U, PV	BE	Page 19-148
CC82CR2S	Timer Compare Shadow Value for Channel 2	0344_H	U, PV	U, PV	Page 19-149
CC82CHC	Channel Control	0348_H	U, PV	U, PV	Page 19-150
CC82DTC	Dead Time Control	$034C_H$	U, PV	U, PV	Page 19-152
CC82DC1R	Channel 1 Dead Time Counter Values	0350_H	U, PV	U, PV	Page 19-153
CC82DC2R	Channel 2 Dead Time Counter Values	0354_H	U, PV	U, PV	Page 19-154
CC82TIMER	Timer Current Value	0370_H	U, PV	U, PV	Page 19-154
CC82C0V	Capture Register 0 Value	0374_H	U, PV	BE	Page 19-155
CC82C1V	Capture Register 1 Value	0378_H	U, PV	BE	Page 19-156
CC82C2V	Capture Register 2 Value	$037C_H$	U, PV	BE	Page 19-157
CC82C3V	Capture Register 3 Value	0380_H	U, PV	BE	Page 19-158
CC82INTS	Interrupt Status	$03A0_H$	U, PV	BE	Page 19-159
CC82INTE	Interrupt Enable	$03A4_H$	U, PV	U, PV	Page 19-161
CC82SRS	Interrupt Configuration	$03A8_H$	U, PV	U, PV	Page 19-163
CC82SWS	Interrupt Status Set	$03AC_H$	U, PV	U, PV	Page 19-165
CC82SWR	Interrupt Status Clear	$03B0_H$	U, PV	U, PV	Page 19-167
CC82STC	Shadow Transfer Control	$03B4_H$	U, PV	U, PV	Page 19-169
CC83 Registers					
CC83INS	Input Selector Unit Configuration	0400_H	U, PV	U, PV	Page 19-126

Table 19-13 Register Overview of CCU8 (cont'd)

Short Name	Description	Offset Addr.[1)]	Access Mode		Description See
			Read	Write	
CC83CMC	Connection Matrix Configuration	0404_H	U, PV	U, PV	Page 19-128
CC83TCST	Timer Run Status	0408_H	U, PV	BE	Page 19-132
CC83TCSET	Timer Run Set	$040C_H$	U, PV	U,PV	Page 19-133
CC83TCCLR	Timer Run Clear	0410_H	U, PV	U, PV	Page 19-134
CC83TC	General Timer Configuration	0414_H	U, PV	U, PV	Page 19-135
CC83PSL	Output Passive Level Configuration	0418_H	U, PV	U, PV	Page 19-140
CC83DIT	Dither Configuration	$041C_H$	U, PV	BE	Page 19-141
CC83DITS	Dither Shadow Register	0420_H	U, PV	U, PV	Page 19-142
CC83PSC	Prescaler Configuration	0424_H	U, PV	U, PV	Page 19-143
CC83FPC	Prescaler Compare Value	0428_H	U, PV	U, PV	Page 19-143
CC83FPCS	Prescaler Shadow Compare Value	$042C_H$	U, PV	U, PV	Page 19-144
CC83PR	Timer Period Value	0430_H	U, PV	BE	Page 19-145
CC83PRS	Timer Period Shadow Value	0434_H	U, PV	U, PV	Page 19-146
CC83CR1	Timer Compare Value for Channel 1	0438_H	U, PV	BE	Page 19-146
CC83CR1S	Timer Compare Shadow Value for Channel 1	$043C_H$	U, PV	U, PV	Page 19-147
CC83CR2	Timer Compare Value for Channel 2	0440_H	U, PV	BE	Page 19-148
CC83CR2S	Timer Compare Shadow Value for Channel 2	0444_H	U, PV	U, PV	Page 19-149
CC83CHC	Channel Control	0448_H	U, PV	U, PV	Page 19-150
CC83DTC	Dead Time Control	$044C_H$	U, PV	U, PV	Page 19-152
CC83DC1R	Channel 1 Dead Time Counter Values	0450_H	U, PV	U, PV	Page 19-153
CC83DC2R	Channel 2 Dead Time Counter Values	0454_H	U, PV	U, PV	Page 19-154
CC83TIMER	Timer Current Value	0470_H	U, PV	U, PV	Page 19-154
CC83C0V	Capture Register 0 Value	0474_H	U, PV	BE	Page 19-155

Table 19-13 Register Overview of CCU8 (cont'd)

Short Name	Description	Offset Addr.[1]	Access Mode		Description See
			Read	Write	
CC83C1V	Capture Register 1 Value	0478_H	U, PV	BE	Page 19-156
CC83C2V	Capture Register 2 Value	$047C_H$	U, PV	BE	Page 19-157
CC83C3V	Capture Register 3 Value	0480_H	U, PV	BE	Page 19-158
CC83INTS	Interrupt Status	$04A0_H$	U, PV	BE	Page 19-159
CC83INTE	Interrupt Enable	$04A4_H$	U, PV	U, PV	Page 19-161
CC83SRS	Interrupt Configuration	$04A8_H$	U, PV	U, PV	Page 19-163
CC83SWS	Interrupt Status Set	$04AC_H$	U, PV	U, PV	Page 19-165
CC83SWR	Interrupt Status Clear	$04B0_H$	U, PV	U, PV	Page 19-167
CC83STC	Shadow Transfer Control	$04B4_H$	U, PV	U, PV	Page 19-169

1) The absolute register address is calculated as follows:
Module Base Address + Offset Address (shown in this column)

19.7.1 Global Registers

GCTRL

The register contains the global configuration fields that affect all the timer slices inside CCU8.

GCTRL
Global Control Register **(0000_H)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSDE		MSE3	MSE2	MSE1	MSE0	SUSCFG		0		PCIS		0	PRBC		
rw		rw	rw	rw	rw	rw		r		rw		r	rw		

Field	Bits	Type	Description
PRBC	[2:0]	rw	**Prescaler Clear Configuration** This register controls how the prescaler Run Bit and internal registers are cleared. 000_B SW only 001_B **GSTAT**.PRB and prescaler registers are cleared when the Run Bit of CC80 is cleared. 010_B **GSTAT**.PRB and prescaler registers are cleared when the Run Bit of CC81 is cleared. 011_B **GSTAT**.PRB and prescaler registers are cleared when the Run Bit of CC82 is cleared. 100_B **GSTAT**.PRB and prescaler registers are cleared when the Run Bit of CC83 is cleared.
PCIS	[5:4]	rw	**Prescaler Input Clock Selection** 00_B Module clock 01_B CCU8x.ECLKA 10_B CCU8x.ECLKB 11_B CCU8x.ECLKC
SUSCFG	[9:8]	rw	**Suspend Mode Configuration** This field controls the entering in suspend mode for all the CCU8 slices. 00_B Suspend request ignored. The module never enters in suspend 01_B Stops all the running slices immediately. Safe stop is not applied. 10_B Stops the block immediately and clamps all the outputs to PASSIVE state. Safe stop is applied. 11_B Waits for the roll over of each slice to stop and clamp the slices outputs. Safe stop is applied.
MSE0	10	rw	**Slice 0 Multi Channel shadow transfer enable** When this field is set, a shadow transfer of slice 0 can be requested not only by SW but also via the CCU8x.MCSS input. 0_B Shadow transfer can only be requested by SW 1_B Shadow transfer can be requested via SW and via the CCU8x.MCSS input.

Field	Bits	Type	Description
MSE1	11	rw	**Slice 1 Multi Channel shadow transfer enable** When this field is set, a shadow transfer of slice 1 can be requested not only by SW but also via the CCU8x.MCSS input. 0_B Shadow transfer can only be requested by SW 1_B Shadow transfer can be requested via SW and via the CCU8x.MCSS input.
MSE2	12	rw	**Slice 2 Multi Channel shadow transfer enable** When this field is set, a shadow transfer of slice 2 can be requested not only by SW but also via the CCU8x.MCSS input. 0_B Shadow transfer can only be requested by SW 1_B Shadow transfer can be requested via SW and via the CCU8xMCSS input.
MSE3	13	rw	**Slice 3 Multi Channel shadow transfer enable** When this field is set, a shadow transfer of slice 3 can be requested not only by SW but also via the CCU8x.MCSS input. 0_B Shadow transfer can only be requested by SW 1_B Shadow transfer can be requested via SW and via the CCU8x.MCSS input.
MSDE	[15:14]	rw	**Multi Channel shadow transfer request configuration** This field configures the type of shadow transfer requested via the CCU8x.MCSS input. The field **CC8yTC**.MSEx needs to be set in order for this configuration to have any effect. 00_B Only the shadow transfer for period and compare values is requested 01_B Shadow transfer for the compare, period and prescaler compare values is requested 10_B Reserved 11_B Shadow transfer for the compare, period, prescaler and dither compare values is requested

Field	Bits	Type	Description
0	3, [7:6], [31:16]	r	**Reserved** A read always returns 0.

GSTAT

The register contains the status of the prescaler and each timer slice (idle mode or running).

GSTAT
Global Status Register **(0004_H)** **Reset Value: $0000000F_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0					PCRB	0	PRB	0				S3I	S2I	S1I	S0I
r					rh	r	rh	r				rh	rh	rh	rh

Field	Bits	Type	Description
S0I	0	rh	**CC80 IDLE status** This bit indicates if the CC80 slice is in IDLE mode or not. In IDLE mode the clocks for the CC80 slice are stopped. 0_B Running 1_B Idle
S1I	1	rh	**CC81 IDLE status** This bit indicates if the CC81 slice is in IDLE mode or not. In IDLE mode the clocks for the CC81 slice are stopped. 0_B Running 1_B Idle

Field	Bits	Type	Description
S2I	2	rh	**CC82 IDLE status** This bit indicates if the CC82 slice is in IDLE mode or not. In IDLE mode the clocks for the CC82 slice are stopped. 0_B Running 1_B Idle
S3I	3	rh	**CC83 IDLE status** This bit indicates if the CC83 slice is in IDLE mode or not. In IDLE mode the clocks for the CC83 slice are stopped. 0_B Running 1_B Idle
PRB	8	rh	**Prescaler Run Bit** 0_B Prescaler is stopped 1_B Prescaler is running
PCRB	10	rh	**Parity Checker Run Bit** 0_B Parity Checker is stopped 1_B Parity Checker is running
0	[7:4], 9, [31:11]	r	**Reserved** Read always returns 0.

GIDLS

Through this register one can set the prescaler and the specific timer slices into idle mode.

GIDLS
Global Idle Set **(0008_H)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0					CPCH	PSIC	CPRB	0				SS3I	SS2I	SS1I	SS0I
r					w	w	w	r				w	w	w	w

Field	Bits	Type	Description
SS0I	0	w	**CC80 IDLE mode set** Writing a 1_B to this bit sets the CC80 slice in IDLE mode. The clocks for the slice are stopped when in IDLE mode. When entering IDLE, the internal slice registers are not cleared. A read access always returns 0.
SS1I	1	w	**CC81 IDLE mode set** Writing a 1_B to this bit sets the CC81 slice in IDLE mode. The clocks for the slice are stopped when in IDLE mode. When entering IDLE, the internal slice registers are not cleared. A read access always returns 0.
SS2I	2	w	**CC82 IDLE mode set** Writing a 1_B to this bit sets the CC82 slice in IDLE mode. The clocks for the slice are stopped when in IDLE mode. When entering IDLE, the internal slice registers are not cleared. A read access always returns 0.
SS3I	3	w	**CC83 IDLE mode set** Writing a 1_B to this bit sets the CC83 slice in IDLE mode. The clocks for the slice are stopped when in IDLE mode. When entering IDLE, the internal slice registers are not cleared. A read access always returns 0.
CPRB	8	w	**Prescaler$_B$ Run Bit Clear** Writing a 1 into this register clears the Run Bit of the prescaler. Prescaler internal registers are not cleared. A read always returns 0.
PSIC	9	w	**Prescaler clear** Writing a 1_B to this register clears the prescaler counter. It also loads the PSIV into the PVAL field for all Timer Slices. This performs a re alignment of the timer clock for all Slices. The Run Bit of the prescaler is not cleared. A read always returns 0.

Field	Bits	Type	Description
CPCH	10	w	**Parity Checker Run bit clear** Writing a 1_B to this register clears the run bit of the parity checker. All the internal registers are cleared. The status bit value is kept, **GPCHK**.PCST. A read always returns 0.
0	[7:4], [31:11]	r	**Reserved** Read always returns 0.

GIDLC

Through this register one can remove the prescaler and the specific timer slices from idle mode.

GIDLC
Global Idle Clear **(000C$_H$)** **Reset Value: 00000000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0					SPCH	0	SPRB	0				CS3I	CS2I	CS1I	CS0I
r					w	r	w	r				w	w	w	w

Field	Bits	Type	Description
CS0I	0	w	**CC80 IDLE mode clear** Writing a 1_B to this bit removes the CC80 from IDLE mode. No clear to the internal slice register is done. A read access always returns 0.
CS1I	1	w	**CC81 IDLE mode clear** Writing a 1_B to this bit removes the CC81 from IDLE mode. No clear to the internal slice register is done. A read access always returns 0.
CS2I	2	w	**CC82 IDLE mode clear** Writing a 1_B to this bit removes the CC82 from IDLE mode. No clear to the internal slice register is done. A read access always returns 0.

Field	Bits	Type	Description
CS3I	3	w	**CC83 IDLE mode clear** Writing a 1_B to this bit removes the CC83 from IDLE mode. No clear to the internal slice register is done. A read access always returns 0.
SPRB	8	w	**Prescaler Run Bit Set** Writing a 1_B into this register sets the Run Bit of the prescaler. Prescaler internal registers are not cleared. A read always returns 0.
SPCH	10	w	**Parity Checker run bit set** Writing a 1_B into this register sets the Run Bit of the parity checker. A read always returns 0.
0	[7:4], 9, [31:11]	r	**Reserved** Read always returns 0.

GCSS

Through this register one can request a shadow transfer for the specific timer slice(s) and set the status bit for each of the compare channels.

GCSS
Global Channel Set **(0010_H)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0								**S3S T2S**	**S2S T2S**	**S1S T2S**	**S0S T2S**	**S3S T1S**	**S2S T1S**	**S1S T1S**	**S0S T1S**
r								w	w	w	w	w	w	w	w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**S3P SE**	**S3D SE**	**S3S E**	**0**	**S2P SE**	**S2D SE**	**S2S E**	**0**	**S1P SE**	**S1D SE**	**S1S E**	**0**	**S0P SE**	**S0D SE**	**S0S E**
r	w	w	w	r	w	w	w	r	w	w	w	r	w	w	w

Field	Bits	Type	Description
S0SE	0	w	**Slice 0 shadow transfer set enable** Writing a 1_B to this bit will set the **GCST**.S0SS field, enabling then a shadow transfer for the Period, Compare and Passive level values. A read always returns 0.

Field	Bits	Type	Description
S0DSE	1	w	**Slice 0 Dither shadow transfer set enable** Writing a 1_B to this bit will set the **GCST**.S0DSS field, enabling then a shadow transfer for the Dither compare value. A read always returns 0.
S0PSE	2	w	**Slice 0 Prescaler shadow transfer set enable** Writing a 1_B to this bit will set the **GCST**.S0PSS field, enabling then a shadow transfer for the prescaler compare value. A read always returns 0.
S1SE	4	w	**Slice 1 shadow transfer set enable** Writing a 1_B to this bit will set the **GCST**.S1SS field, enabling then a shadow transfer for the Period, Compare and Passive level values. A read always returns 0.
S1DSE	5	w	**Slice 1 Dither shadow transfer set enable** Writing a 1_B to this bit will set the **GCST**.S1DSS field, enabling then a shadow transfer for the Dither compare value. A read always returns 0.
S1PSE	6	w	**Slice 1 Prescaler shadow transfer set enable** Writing a 1_B to this bit will set the **GCST**.S1PSS field, enabling then a shadow transfer for the prescaler compare value. A read always returns 0.
S2SE	8	w	**Slice 2 shadow transfer set enable** Writing a 1_B to this bit will set the **GCST**.S2SS field, enabling then a shadow transfer for the Period, Compare and Passive level values. A read always returns 0.
S2DSE	9	w	**Slice 2 Dither shadow transfer set enable** Writing a 1_B to this bit will set the **GCST**.S2DSS field, enabling then a shadow transfer for the Dither compare value. A read always returns 0.

Field	Bits	Type	Description
S2PSE	10	w	**Slice 2 Prescaler shadow transfer set enable** Writing a 1_B to this bit will set the **GCST**.S2PSS field, enabling then a shadow transfer for the prescaler compare value. A read always returns 0.
S3SE	12	w	**Slice 3 shadow transfer set enable** Writing a 1_B to this bit will set the **GCST**.S3SS field, enabling then a shadow transfer for the Period, Compare and Passive level values. A read always returns 0.
S3DSE	13	w	**Slice 3 Dither shadow transfer set enable** Writing a 1_B to this bit will set the **GCST**.S3DSS field, enabling then a shadow transfer for the Dither compare value. A read always returns 0.
S3PSE	14	w	**Slice 3 Prescaler shadow transfer set enable** Writing a 1_B to this bit will set the **GCST**.S3PSS field, enabling then a shadow transfer for the prescaler compare value. A read always returns 0.
S0ST1S	16	w	**Slice 0 status bit 1 set** Writing a 1_B into this register sets the compare channel 1 status bit of slice 0 (**GCST**.CC80ST1). A read always returns 0.
S1ST1S	17	w	**Slice 1 status bit 1 set** Writing a 1_B into this register sets the compare channel 1 status bit of slice 1 (**GCST**.CC81ST1). A read always returns 0.
S2ST1S	18	w	**Slice 2 status bit 1 set** Writing a 1_B into this register sets the compare channel 1 status bit of slice 2 (**GCST**.CC82ST1). A read always returns 0.
S3ST1S	19	w	**Slice 3 status bit 1 set** Writing a 1_B into this register sets the compare channel 2 status bit of slice 3 (**GCST**.CC83ST1). A read always returns 0.

Field	Bits	Type	Description
S0ST2S	20	w	**Slice 0 status bit 2 set** Writing a 1_B into this register sets the compare channel 2 status bit of slice 0 (**GCST**.CC80ST2). A read always returns 0.
S1ST2S	21	w	**Slice 1 status bit 2 set** Writing a 1_B into this register sets the compare channel 2 status bit of slice 1 (**GCST**.CC81ST2). A read always returns 0.
S2ST2S	22	w	**Slice 2 status bit 2 set** Writing a 1_B into this register sets the compare channel 2 status bit of slice 2 (**GCST**.CC82ST2). A read always returns 0.
S3ST2S	23	w	**Slice 3 status bit 2 set** Writing a 1_B into this register sets the compare channel 2 status bit of slice 3 (**GCST**.CC83ST2). A read always returns 0.
0	3, 7, 11, 15, [31:24]	r	**Reserved** Read always returns 0.

GCSC

Through this register one can reset a shadow transfer request for the specific timer slice and clear the status bit for each the compare channels.

GCSC
Global Channel Clear **(0014_H)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0								S3ST2C	S2ST2C	S1ST2C	S0ST2C	S3ST1C	S2ST1C	S1ST1C	S0ST1C
r								w	w	w	w	w	w	w	w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	S3PSC	S3DSC	S3SC	0	S2PSC	S2DSC	S2SC	0	S1PSC	S1DSC	S1SC	0	S0PSC	S0DSC	S0SC
r	w	w	w	r	w	w	w	r	w	w	w	r	w	w	w

Field	Bits	Type	Description
S0SC	0	w	**Slice 0 shadow transfer request clear** Writing a 1_B to this bit will clear the **GCST**.S0SS field, canceling any pending shadow transfer for the Period, Compare and Passive level values. A read always returns 0.
S0DSC	1	w	**Slice 0 Dither shadow transfer clear** Writing a 1_B to this bit will clear the **GCST**.S0DSS field, canceling any pending shadow transfer for the Dither compare value. A read always returns 0.
S0PSC	2	w	**Slice 0 Prescaler shadow transfer clear** Writing a 1_B to this bit will clear the **GCST**.S0PSS field, canceling any pending shadow transfer for the prescaler compare value. A read always returns 0.
S1SC	4	w	**Slice 1 shadow transfer clear** Writing a 1_B to this bit will clear the **GCST**.S1SS field, canceling any pending shadow transfer for the Period, Compare and Passive level values. A read always returns 0.
S1DSC	5	w	**Slice 1 Dither shadow transfer clear** Writing a 1_B to this bit will clear the **GCST**.S1DSS field, canceling any pending shadow transfer for the Dither compare value. A read always returns 0.
S1PSC	6	w	**Slice 1 Prescaler shadow transfer clear** Writing a 1_B to this bit will clear the **GCST**.S1PSS field, canceling any pending shadow transfer for the prescaler compare value. A read always returns 0.
S2SC	8	w	**Slice 2 shadow transfer clear** Writing a 1_B to this bit will clear the **GCST**.S2SS field, canceling any pending shadow transfer for the Period, Compare and Passive level values. A read always returns 0.

Field	Bits	Type	Description
S2DSC	9	w	**Slice 2 Dither shadow transfer clear** Writing a 1_B to this bit will clear the **GCST**.S2DSS field, canceling any pending shadow transfer for the Dither compare value. A read always returns 0.
S2PSC	10	w	**Slice 2 Prescaler shadow transfer clear** Writing a 1_B to this bit will clear the **GCST**.S2PSS field, canceling any pending shadow transfer for the prescaler compare value. A read always returns 0.
S3SC	12	w	**Slice 3 shadow transfer clear** Writing a 1_B to this bit will clear the **GCST**.S3SS field, canceling any pending shadow transfer for the Period, Compare and Passive level values. A read always returns 0.
S3DSC	13	w	**Slice 3 Dither shadow transfer clear** Writing a 1_B to this bit will clear the **GCST**.S3DSS field, canceling any pending shadow transfer for the Dither compare value. A read always returns 0.
S3PSC	14	w	**Slice 3 Prescaler shadow transfer clear** Writing a 1_B to this bit will clear the **GCST**.S3PSS field, canceling any pending shadow transfer for the prescaler compare value. A read always returns 0.
S0ST1C	16	w	**Slice 0 status bit 1 clear** Writing a 1_B into this register clears the compare channel 1 status bit of slice 0 (**GCST**.CC80ST1). A read always returns 0.
S1ST1C	17	w	**Slice 1 status bit 1 clear** Writing a 1_B into this register clears the compare channel 1 status bit of slice 1 (**GCST**.CC81ST1). A read always returns 0.
S2ST1C	18	w	**Slice 2 status bit 1 clear** Writing a 1_B into this register clears the compare channel 1 status bit of slice 2 (**GCST**.CC82ST1). A read always returns 0.

Field	Bits	Type	Description
S3ST1C	19	w	**Slice 3 status bit 1 clear** Writing a 1_B into this register clears the compare channel 1 status bit of slice 3 (**GCST**.CC83ST1). A read always returns 0.
S0ST2C	20	w	**Slice 0 status bit 2 clear** Writing a 1_B into this register clears the compare channel 2 status bit of slice 0 (**GCST**.CC80ST2). A read always returns 0.
S1ST2C	21	w	**Slice 1 status bit 2 clear** Writing a 1_B into this register clears the compare channel 2 status bit of slice 1 (**GCST**.CC81ST2). A read always returns 0.
S2ST2C	22	w	**Slice 2 status bit 2 clear** Writing a 1_B into this register clears the compare channel 2 status bit of slice 2 (**GCST**.CC82ST2). A read always returns 0.
S3ST2C	23	w	**Slice 3 status bit 2 clear** Writing a 1_B into this register clears the compare channel 2 status bit of slice 3 (**GCST**.CC83ST2). A read always returns 0.
0	3, 7, 11, 15, [31:24]	r	**Reserved** Read always returns 0.

GCST

This register holds the information of the shadow transfer requests and of each timer slice status bit.

GCST
Global Channel status **(0018_H)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0								CC83ST2	CC82ST2	CC81ST2	CC80ST2	CC83ST1	CC82ST1	CC81ST1	CC80ST1
r								rh	rh	rh	rh	rh	rh	rh	rh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	S3PSS	S3DSS	S3SS	0	S2PSS	S2DSS	S2SS	0	S1PSS	S1DSS	S1SS	0	S0PSS	S0DSS	S0SS
r	rh	rh	rh	r	rh	rh	rh	r	rh	rh	rh	r	rh	rh	rh

Field	Bits	Type	Description
S0SS	0	rh	**Slice 0 shadow transfer status** 0_B Shadow transfer has not been requested 1_B Shadow transfer has been requested This field is cleared by HW after the requested shadow transfer has been executed.
S0DSS	1	rh	**Slice 0 Dither shadow transfer status** 0_B Dither shadow transfer has not been requested 1_B Dither shadow transfer has been requested This field is cleared by HW after the requested shadow transfer has been executed.
S0PSS	2	rh	**Slice 0 Prescaler shadow transfer status** 0_B Prescaler shadow transfer has not been requested 1_B Prescaler shadow transfer has been requested This field is cleared by HW after the requested shadow transfer has been executed.
S1SS	4	rh	**Slice 1 shadow transfer status** 0_B Shadow transfer has not been requested 1_B Shadow transfer has been requested This field is cleared by HW after the requested shadow transfer has been executed.

Field	Bits	Type	Description
S1DSS	5	rh	**Slice 1 Dither shadow transfer status** 0_B Dither shadow transfer has not been requested 1_B Dither shadow transfer has been requested This field is cleared by HW after the requested shadow transfer has been executed.
S1PSS	6	rh	**Slice 1 Prescaler shadow transfer status** 0_B Prescaler shadow transfer has not been requested 1_B Prescaler shadow transfer has been requested This field is cleared by HW after the requested shadow transfer has been executed.
S2SS	8	rh	**Slice 2 shadow transfer status** 0_B Shadow transfer has not been requested 1_B Shadow transfer has been requested This field is cleared by HW after the requested shadow transfer has been executed.
S2DSS	9	rh	**Slice 2 Dither shadow transfer status** 0_B Dither shadow transfer has not been requested 1_B Dither shadow transfer has been requested This field is cleared by HW after the requested shadow transfer has been executed.
S2PSS	10	rh	**Slice 2 Prescaler shadow transfer status** 0_B Prescaler shadow transfer has not been requested 1_B Prescaler shadow transfer has been requested This field is cleared by HW after the requested shadow transfer has been executed.
S3SS	12	rh	**Slice 3 shadow transfer status** 0_B Shadow transfer has not been requested 1_B Shadow transfer has been requested This field is cleared by HW after the requested shadow transfer has been executed.

Field	Bits	Type	Description
S3DSS	13	rh	**Slice 3 Dither shadow transfer status** 0_B Dither shadow transfer has not been requested 1_B Dither shadow transfer has been requested This field is cleared by HW after the requested shadow transfer has been executed.
S3PSS	14	rh	**Slice 3 Prescaler shadow transfer status** 0_B Prescaler shadow transfer has not been requested 1_B Prescaler shadow transfer has been requested This field is cleared by HW after the requested shadow transfer has been executed.
CC80ST1	16	rh	**Slice 0 compare channel 1 status bit**
CC81ST1	17	rh	**Slice 1 compare channel 1 status bit**
CC82ST1	18	rh	**Slice 2 compare channel 1 status bit**
CC83ST1	19	rh	**Slice 3 compare channel 1 status bit**
CC80ST2	20	rh	**Slice 0 compare channel 2 status bit**
CC81ST2	21	rh	**Slice 1 compare channel 2 status bit**
CC82ST2	22	rh	**Slice 2 compare channel 2 status bit**
CC83ST2	23	rh	**Slice 3 compare channel 2 status bit**
0	3, 7, 11, 15, [31:24]	r	**Reserved** Read always returns 0.

GPCHK

This register contains the configuration for the Parity Check function of CCU8.

GPCHK
Parity Checker Configuration **(001C$_H$)** **Reset Value: 00000000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PCSEL3				PCSEL2				PCSEL1				PCSEL0			
rw				rw				rw				rw			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PCST	0							PCTS	PCDS		PISEL		PACS		PASE
rh	r							rw	rw		rw		rw		rw

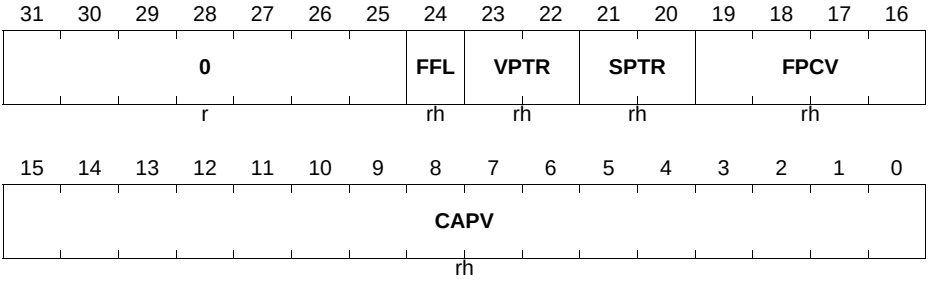
Field	Bits	Type	Description
PASE	0	rw	**Parity Checker Automatic start/stop** If this field is set, the parity checker run bit is automatically set, when the run bit of the selected slice is set and it is cleared when the run bit of the slice is cleared. The field PACS needs to be programmed accordingly.
PACS	[2:1]	rw	**Parity Checker Automatic start/stop selector** This fields selects to which slice the automatic start/stop of the parity checker is associated: 00_B CC80 01_B CC81 10_B CC82 11_B CC83
PISEL	[4:3]	rw	**Driver Input signal selector** This fields selects which signal contains the driver parity information: 00_B CC8x.GP01 - driver output is connected to event 1 of slice 0 01_B CC8x.GP11 - drive output is connected to event 1 of slice 1 10_B CC8x.GP21 - driver output is connected to event 1 of slice 2 11_B CC8x.GP31 - driver output is connected to event 1 of slice 3

Field	Bits	Type	Description
PCDS	[6:5]	rw	**Parity Checker Delay Input Selector** This fields selects which signal is controlling the delay between the change at the CCU8 outputs and effective change at the driver parity output: 00_B CCU8x.IGBTA 01_B CCU8x.IGBTB 10_B CCU8x.IGBTC 11_B CCU8x.IGBTD
PCTS	7	rw	**Parity Checker type selector** This fields selects if we have an odd or even parity: 0_B Even parity enabled 1_B Odd parity enabled
PCST	15	rh	**Parity Checker XOR status** This field contains the current value of the XOR chain.
PCSEL0	[19:16]	rw	**Parity Checker Slice 0 output selection** This fields selects which slice 0 outputs are going to be used to perform the parity check. The respective bit field needs to be set to 1_B to enable the output in the parity function. PCSEL0[0] - CCU8x.OUT00 PCSEL0[1] - CCU8x.OUT01 PCSEL0[2] - CCU8x.OUT02 PCSEL0[3] - CCU8x.OUT03
PCSEL1	[23:20]	rw	**Parity Checker Slice 1 output selection** Same description as PCSEL0.
PCSEL2	[27:24]	rw	**Parity Checker Slice 2 output selection** Same description as PCSEL0.
PCSEL3	[31:28]	rw	**Parity Checker Slice 3 output selection** Same description as PCSEL0.
0	[14:8]	r	**Reserved** Read always returns 0.

ECRD

This register holds the information related to the extended capture mode.

ECRD
Extended Capture Mode Read **(0050_H)** **Reset Value: 00000000_H**



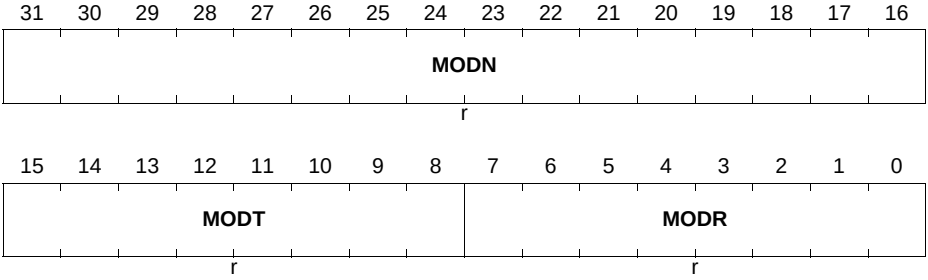
Field	Bits	Type	Description
CAPV	[15:0]	rh	**Timer Capture Value** This field contains the timer captured value
FPCV	[19:16]	rh	**Prescaler Capture value** This field contains the value of the prescaler clock division associated with the specific CAPV field
SPTR	[21:20]	rh	**Slice pointer** This field indicates the slice index in which the value was captured. 00_B CC80 01_B CC81 10_B CC82 11_B CC83
VPTR	[23:22]	rh	**Capture register pointer** This field indicates the capture register index in which the value was captured. 00_B Capture register 0 01_B Capture register 1 10_B Capture register 2 11_B Capture register 3
FFL	24	rh	**Full Flag** This bit indicates if the associated capture register contains a value. 0_B No new value was captured into this register 1_B A new value has been captured into this register

Field	Bits	Type	Description
0	[31:25]	r	**Reserved** Read always returns 0.

MIDR

This register contains the module identification number.

MIDR
Module Identification **(0080_H)** **Reset Value: $00A7C0XX_H$**



Field	Bits	Type	Description
MODR	[7:0]	r	**Module Revision** This bit field indicates the revision number of the module implementation (depending on the design step). The given value of 00_H is a placeholder for the actual number.
MODT	[15:8]	r	**Module Type**
MODN	[31:16]	r	**Module Number**

19.7.2 Slice (CC8y) Registers

CC8yINS

The register contains the configuration for the input selector.

CC8yINS (y = 0 - 3)
Input Selector Configuration **($0100_H + 0100_H * y$)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	LPF2M		LPF1M		LPF0M		EV2LM	EV1LM	EV0LM	EV2EM		EV1EM		EV0EM	
r	rw		rw		rw		rw	rw	rw	rw		rw		rw	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				EV2IS				EV1IS				EV0IS			
r				rw				rw				rw			

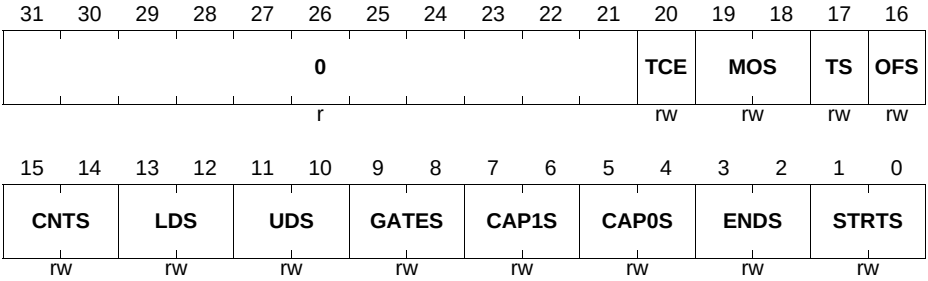
Field	Bits	Type	Description
EV0IS	[3:0]	rw	**Event 0 signal selection** This field selects which pins is used for the event 0. 0000_B CCU8x.INyA 0001_B CCU8x.INyB 0010_B CCU8x.INyC 0011_B CCU8x.INyD 0100_B CCU8x.INyE 0101_B CCU8x.INyF 0110_B CCU8x.INyG 0111_B CCU8x.INyH 1000_B CCU8x.INyI 1001_B CCU8x.INyJ 1010_B CCU8x.INyK 1011_B CCU8x.INyL 1100_B CCU8x.INyM 1101_B CCU8x.INyN 1110_B CCU8x.INyO 1111_B CCU8x.INyP
EV1IS	[7:4]	rw	**Event 1 signal selection** Same as EV0IS description
EV2IS	[11:8]	rw	**Event 2 signal selection** Same as EV0IS description

Field	Bits	Type	Description
EV0EM	[17:16]	rw	**Event 0 Edge Selection** 00_B No action 01_B Signal active on rising edge 10_B Signal active on falling edge 11_B Signal active on both edges
EV1EM	[19:18]	rw	**Event 1 Edge Selection** Same as EV0EM description
EV2EM	[21:20]	rw	**Event 2 Edge Selection** Same as EV0EM description
EV0LM	22	rw	**Event 0 Level Selection** 0_B Active on HIGH level 1_B Active on LOW level
EV1LM	23	rw	**Event 1 Level Selection** Same as EV0LM description
EV2LM	24	rw	**Event 2 Level Selection** Same as EV0LM description
LPF0M	[26:25]	rw	**Event 0 Low Pass Filter Configuration** This field sets the number of consecutive counts for the Low Pass Filter of Event 0. The input signal value needs to remain stable for this number of counts (f_{CCU8}), so that a level/transition is accepted. 00_B LPF is disabled 01_B 3 clock cycles of f_{CCU8} 10_B 5 clock cycles of f_{CCU8} 11_B 7 clock cycles of f_{CCU8}
LPF1M	[28:27]	rw	**Event 1 Low Pass Filter Configuration** Same description as LPF0M
LPF2M	[30:29]	rw	**Event 2 Low Pass Filter Configuration** Same description as LPF0M
0	[15:12], 31	r	**Reserved** Read always returns 0.

CC8yCMC

The register contains the configuration for the connection matrix.

CC8yCMC (y = 0 - 3)
Connection Matrix Control **($0104_H + 0100_H * y$)** **Reset Value: 00000000_H**



Field	Bits	Type	Description
STRTS	[1:0]	rw	**External Start Functionality Selector** Selects the Event that is going to be linked with the external start functionality. 00_B External Start Function deactivated 01_B External Start Function triggered by Event 0 10_B External Start Function triggered by Event 1 11_B External Start Function triggered by Event 2
ENDS	[3:2]	rw	**External Stop Functionality Selector** Selects the Event that is going to be linked with the external stop functionality. 00_B External Stop Function deactivated 01_B External Stop Function triggered by Event 0 10_B External Stop Function triggered by Event 1 11_B External Stop Function triggered by Event 2

Field	Bits	Type	Description
CAP0S	[5:4]	rw	**External Capture 0 Functionality Selector** Selects the Event that is going to be linked with the external capture for capture registers number 1 and 0. This function is used to capture the value of the timer into the capture registers 1 and 0. 00_B External Capture 0 Function deactivated 01_B External Capture 0 Function triggered by Event 0 10_B External Capture 0 Function triggered by Event 1 11_B External Capture 0 Function triggered by Event 2 *Note: If the field SCE is set, this functionality is deactivated.*
CAP1S	[7:6]	rw	**External Capture 1 Functionality Selector** Selects the Event that is going to be linked with the external capture for capture registers number 3 and 2. This function is used to capture the value of the timer into the capture registers 3 and 2. 00_B External Capture 1 Function deactivated 01_B External Capture 1 Function triggered by Event 0 10_B External Capture 1 Function triggered by Event 1 11_B External Capture 1 Function triggered by Event 2
GATES	[9:8]	rw	**External Gate Functionality Selector** Selects the Event that is going to be linked with the counter gating function. This function is used to gate the timer increment/decrement procedure. 00_B External Gating Function deactivated 01_B External Gating Function triggered by Event 0 10_B External Gating Function triggered by Event 1 11_B External Gating Function triggered by Event 2

Field	Bits	Type	Description
UDS	[11:10]	rw	**External Up/Down Functionality Selector** Selects the Event that is going to be linked with the Up/Down counting direction control. This function is used to control externally the timer increment/decrement operation. 00_B External Up/Down Function deactivated 01_B External Up/Down Function triggered by Event 0 10_B External Up/Down Function triggered by Event 1 11_B External Up/Down Function triggered by Event 2
LDS	[13:12]	rw	**External Timer Load Functionality Selector** Selects the Event that is going to be linked with the timer load function. The value present in the **CC8yCR1**/**CC8yCR2** (depending on the value of **CC8yTC**.TLS) is loaded into the specific slice timer. 00_B - External Load Function deactivated 01_B - External Load Function triggered by Event 0 10_B - External Load Function triggered by Event 1 11_B - External Load Function triggered by Event 2
CNTS	[15:14]	rw	**External Count Selector** Selects the Event that is going to be linked with the count function. The counter is going to be increment/decremented each time that a specific transition on the event is detected. 00_B External Count Function deactivated 01_B External Count Function triggered by Event 0 10_B External Count Function triggered by Event 1 11_B External Count Function triggered by Event 2 *Note: In CC40 this field doesn't exist. This is a read only reserved field. Read access always returns 0.*
OFS	16	rw	**Override Function Selector** This field enables the ST bit override functionality. 0_B Override functionality disabled 1_B Status bit trigger override connected to Event 1; Status bit value override connected to Event 2

Field	Bits	Type	Description
TS	17	rw	**Trap Function Selector** This field enables the trap functionality. 0_B Trap function disabled 1_B TRAP function connected to Event 2
MOS	[19:18]	rw	**External Modulation Functionality Selector** Selects the Event that is going to be linked with the external modulation function. 00_B - Modulation Function deactivated 01_B - Modulation Function triggered by Event 0 10_B - Modulation Function triggered by Event 1 11_B - Modulation Function triggered by Event 2
TCE	20	rw	**Timer Concatenation Enable** This bit enables the timer concatenation with the previous slice. 0_B Timer concatenation is disabled 1_B Timer concatenation is enabled *Note: In CC80 this field doesn't exist. This is a read only reserved field. Read access always returns 0.*
0	[31:21]	r	**Reserved** A read always returns 0

CC8yTCST

The register holds the status of the timer (running/stopped) and the information about the counting direction (up/down).

CC8yTCST (y = 0 - 3)
Slice Timer Status **($0108_H + 0100_H * y$)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

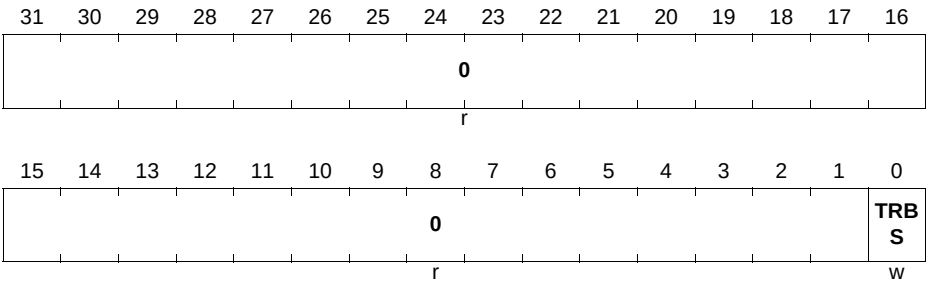
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0											DTR2	DTR1	0	CDIR	TRB
r											rh	rh	r	rh	rh

Field	Bits	Type	Description
TRB	0	rh	**Timer Run Bit** This field indicates if the timer is running. 0_B Timer is stopped 1_B Timer is running
CDIR	1	rh	**Timer Counting Direction** This filed indicates if the timer is being increment or decremented 0_B Timer is counting up 1_B Timer is counting down
DTR1	3	rh	**Dead Time Counter 1 Run bit** This field indicates if the dead time counter for linked with channel 1 is running. 0_B Dead Time counter is idle 1_B Dead Time counter is running
DTR2	4	rh	**Dead Time Counter 2 Run bit** This field indicates if the dead time counter for linked with channel 2 is running. 0_B Dead Time counter is idle 1_B Dead Time counter is running
0	2, [31:5]	r	**Reserved** Read always returns 0

CC8yTCSET

Through this register it is possible to start the timer.

CC8yTCSET (y = 0 - 3)
Slice Timer Run Set **($010C_H + 0100_H * y$)** **Reset Value: 00000000_H**



Field	Bits	Type	Description
TRBS	0	w	**Timer Run Bit set** Writing a 1_B into this field sets the run bit of the timer. The timer is not cleared. Read always returns 0.
0	[31:1]	r	**Reserved** Read always returns 0

CC8yTCCLR

Through this register it is possible to stop and clear the timer, and clearing also the dither counter

CC8yTCCLR (y = 0 - 3)
Slice Timer Clear **($0110_H + 0100_H * y$)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0											DTC 2C	DTC 1C	DITC	TCC	TRB C
r											w	w	w	w	w

Field	Bits	Type	Description
TRBC	0	w	**Timer Run Bit Clear** Writing a 1_B into this field clears the run bit of the timer. The timer is not cleared. Read always returns 0.
TCC	1	w	**Timer Clear** Writing a 1_B into this field clears the timer value. Read always returns 0.
DITC	2	w	**Dither Counter Clear** Writing a 1_B into this field clears the dither counter. Read always returns 0.

Field	Bits	Type	Description
DTC1C	3	w	**Dead Time Counter 1 Clear** Writing a 1_B into this field clears the channel 1 dead time counter. The counter is stopped until a new start trigger is detected. Read always returns 0.
DTC2C	4	w	**Dead Time Counter 2 Clear** Writing a 1_B into this field clears the channel 2 dead time counter. The counter is stopped until a new start trigger is detected. Read always returns 0.
0	[31:5]	r	**Reserved** Read always returns 0

CC8yTC

This register holds the several possible configurations for the timer operation.

CC8yTC (y = 0 - 3)
Slice Timer Control **($0114_H + 0100_H * y$)** **Reset Value: 18000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**STOS**		**EME**		**MCME2**	**MCME1**	**EMT**	**EMS**	**TRPSW**	**TRPSE**	**TRAPE3**	**TRAPE2**	**TRAPE1**	**TRAPE0**	**FPE**
r	rw		rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DIM	**DITHE**		**CCS**	**SCE**	**STRM**	**ENDM**		**TLS**	**CAPC**		**ECM**	**CMOD**	**CLST**	**TSSM**	**TCM**
rw	rw		rw	rw	rw	rw		rw	rw		rw	rh	rw	rw	rw

Field	Bits	Type	Description
TCM	0	rw	**Timer Counting Mode** This field controls the actual counting scheme of the timer. 0_B Edge aligned mode 1_B Center aligned mode *Note: When using an external signal to control the counting direction, the counting scheme is always edge aligned.*

Field	Bits	Type	Description
TSSM	1	rw	**Timer Single Shot Mode** This field controls the single shot mode. This is applicable in edge and center aligned modes. 0_B Single shot mode is disabled 1_B Single shot mode is enabled
CLST	2	rw	**Shadow Transfer on Clear** Setting this bit to 1_B enables a shadow transfer when a timer clearing action is done (by SW or by an external event). Notice that the shadow transfer enable bitfields on the **GCST** register still need to be set to 1_B via software.
CMOD	3	rh	**Capture Compare Mode** This field indicates in which mode the slice is operating. The default value is compare mode. The capture mode is automatically set by the HW when an external signal is mapped to a capture trigger. 0_B Compare Mode 1_B Capture Mode
ECM	4	rw	**Extended Capture Mode** This field control the Capture mode of the specific slice. It only has effect if the CMOD bit is 1_B. 0_B Normal Capture Mode. Clear of the Full Flag of each capture register is done by accessing the registers individually only. 1_B Extended Capture Mode. Clear of the Full Flag of each capture register is done not only by accessing the individual registers but also by accessing the ECRD register. When reading the ECRD register, only the capture register register full flag pointed by the VPTR is cleared
CAPC	[6:5]	rw	**Clear on Capture Control** 00_B Timer is never cleared on a capture event 01_B Timer is cleared on a capture event into capture registers 2 and 3. (When SCE = 1_B, Timer is always cleared in a capture event) 10_B Timer is cleared on a capture event into capture registers 0 and 1. (When SCE = 1_B, Timer is always cleared in a capture event) 11_B Timer is always cleared in a capture event.

Field	Bits	Type	Description
TLS	7	rw	**Timer Load selector** 0_B Timer is loaded with the value of CR1 1_B Timer is loaded with the value of CR2
ENDM	[9:8]	rw	**Extended Stop Function Control** This field controls the extended functions of the external Stop signal. 00_B Clears the timer run bit only (default stop) 01_B Clears the timer only (flush) 10_B Clears the timer and run bit (flush/stop) 11_B Reserved *Note: When using an external up/down signal the flush operation sets the timer with zero if the counter is counting up and with the Period value if the counter is being decremented.*
STRM	10	rw	**Extended Start Function Control** This field controls the extended functions of the external Start signal. 0_B Sets run bit only (default start) 1_B Clears the timer and sets run bit, if not set (flush/start) *Note: When using an external up/down signal the flush operation sets the timer with zero if the counter is being incremented and with the Period value if the counter is being decremented.*
SCE	11	rw	**Equal Capture Event enable** 0_B Capture into **CC8yC0V**/**CC8yC1V** registers control by CCycapt0 and capture into **CC8yC3V**/**CC8yC2V** control by CCycapt1 1_B Capture into **CC8yC0V**/**CC8yC1V** and **CC8yC3V**/**CC8yC2V** control by CCycapt1
CCS	12	rw	**Continuous Capture Enable** 0_B The capture into a specific capture register is done with the rules linked with the full flags, described at **Section 19.2.8.6**. 1_B The capture into the capture registers is always done regardless of the full flag status (even if the register has not been read back).

Field	Bits	Type	Description
DITHE	[14:13]	rw	**Dither Enable** This field controls the dither mode for the slice. See **Section 19.2.12**. 00_B Dither is disabled 01_B Dither is applied to the Period 10_B Dither is applied to the Compare 11_B Dither is applied to the Period and Compare
DIM	15	rw	**Dither input selector** This fields selects if the dither control signal is connected to the dither logic of the specific slice of is connected to the dither logic of slice 0. Notice that even if this field is set to 1_B, the field DITHE still needs to be programmed. 0_B Slice is using it own dither unit 1_B Slice is connected to the dither unit of slice 0.
FPE	16	rw	**Floating Prescaler enable** Setting this bit to 1_B enables the floating prescaler mode. 0_B Floating prescaler mode is disabled 1_B Floating prescaler mode is enabled
TRAPE0	17	rw	**TRAP enable for CCU8x.OUTy0** Setting this bit to 1 enables the TRAP action at the CCU8x.OUTy0 output pin. After mapping an external signal to the TRAP functionality, the user must set this field to 1 to activate the effect of the TRAP on the specific output pin. Writing a 0 into this field disables the effect of the TRAP function regardless of the state of the input signal. 0_B TRAP functionality has no effect on the CCU8x.OUTy0 output 1_B TRAP functionality affects the CCU8x.OUTy0 output
TRAPE1	18	rw	**TRAP enable for CCU8x.OUTy1** TRAP enable for the CCU8x.OUTy1. Same description as for the TRAPE0 field.
TRAPE2	19	rw	**TRAP enable for CCU8x.OUTy2** TRAP enable for the CCU8x.OUTy2. Same description as for the TRAPE0 field.

Field	Bits	Type	Description
TRAPE3	20	rw	**TRAP enable for CCU8x.OUTy3** TRAP enable for the CCU8x.OUTy3. Same description as for the TRAPE0 field.
TRPSE	21	rw	**TRAP Synchronization Enable** Writing a 1 into this bit enables a synchronous exiting with the PWM period of the trap state. 0_B Exiting from TRAP state isn't synchronized with the PWM signal 1_B Exiting from TRAP state is synchronized with the PWM signal
TRPSW	22	rw	**TRAP State Clear Control** 0_B The slice exits the TRAP state automatically when the TRAP condition is not present (Trap state cleared by HW and SW) 1_B The TRAP state can only be exited by a SW request.
EMS	23	rw	**External Modulation Synchronization** Setting this bit to 1 enables the synchronization of the external modulation functionality with the PWM period. 0_B External Modulation functionality is not synchronized with the PWM signal 1_B External Modulation functionality is synchronized with the PWM signal
EMT	24	rw	**External Modulation Type** This field selects if the external modulation event is clearing the CC8ySTx bits or if is gating the outputs. 0_B External Modulation functionality is clearing the CC8ySTx bits. 1_B External Modulation functionality is gating the outputs.
MCME1	25	rw	**Multi Channel Mode Enable for Channel 1** 0_B Multi Channel Mode in Channel 1 is disabled 1_B Multi Channel Mode in Channel 1 is enabled
MCME2	26	rw	**Multi Channel Mode Enable for Channel 2** 0_B Multi Channel Mode in Channel 2 is disabled 1_B Multi Channel Mode in Channel 2 is enabled

Field	Bits	Type	Description
EME	[28:27]	rw	**External Modulation Channel enable** This field controls in which channel, the modulation functionality has effect. The modulations functionality needs to be previously enabled by setting the **CC8yCMC**.MOS accordingly. 00_B External Modulation functionality doesn't affect any channel 01_B External Modulation only applied on channel 1 10_B External Modulation only applied on channel 2 11_B External Modulation applied on both channels
STOS	[30:29]	rw	**Status bit output selector** This field selects to which channel the output CC8ySTy is mapped. 00_B CC8yST1 forward to CCU8x.STy 01_B CC8yST2 forward to CCU8x.STy 10_B CC8yST1 AND CC8yST2 forward to CCU8x.STy 11_B Reserved
0	31	r	**Reserved** Read always returns 0.

CC8yPSL

This register holds the configuration for the output passive level control.

CC8yPSL (y = 0 - 3)
Passive Level Config **($0118_H + 0100_H * y$)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

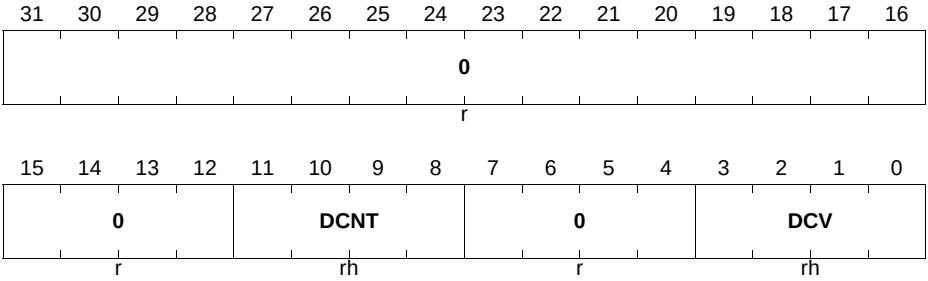
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0												PSL 22	PSL 21	PSL 12	PSL 11
r												rw	rw	rw	rw

Field	Bits	Type	Description
PSL11	0	rw	**Output Passive Level for CCU8x.OUTy0** This field controls the passive level of the CCU8x.OUTy0. 0_B Passive Level is LOW 1_B Passive Level is HIGH A write always addresses the shadow register, while a read always returns the current used value.
PSL12	1	rw	**Output Passive Level for CCU8x.OUTy1** This field controls the passive level of the CCU8x.OUTy1. 0_B Passive Level is LOW 1_B Passive Level is HIGH A write always addresses the shadow register, while a read always returns the current used value.
PSL21	2	rw	**Output Passive Level for CCU8x.OUTy2** This field controls the passive level of the CCU8x.OUTy2. 0_B Passive Level is LOW 1_B Passive Level is HIGH A write always addresses the shadow register, while a read always returns the current used value.
PSL22	3	rw	**Output Passive Level for CCU8x.OUTy3** This field controls the passive level of the CCU8x.OUTy3. 0_B Passive Level is LOW 1_B Passive Level is HIGH A write always addresses the shadow register, while a read always returns the current used value.
0	[31:4]	r	**Reserved** A read access always returns 0

CC8yDIT

This register holds the current dither compare and dither counter values.

CC8yDIT (y = 0 - 3)
Dither Config **($011C_H + 0100_H * y$)** **Reset Value: 00000000_H**

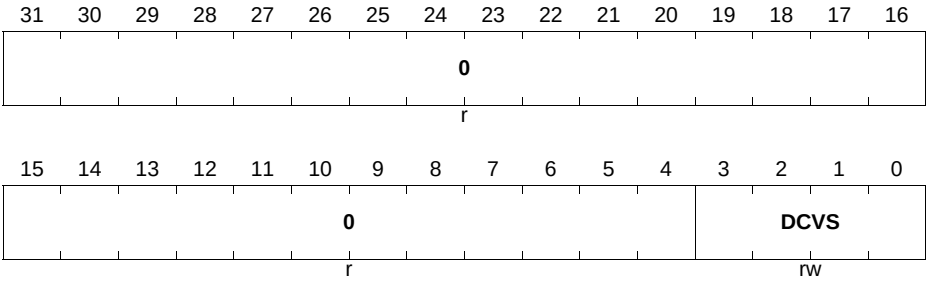


Field	Bits	Type	Description
DCV	[3:0]	rh	**Dither compare Value** This field contains the value used for the dither comparison. This value is updated when a shadow transfer occurs with the **CC8yDITS**.DCVS.
DCNT	[11:8]	rh	**Dither counter actual value**
0	[7:4], [31:12]	r	**Reserved** Read always returns 0.

CC8yDITS

This register contains the value that is going to be loaded into the **CC8yDIT**.DCV when the next shadow transfer occurs.

CC8yDITS (y = 0 - 3)
Dither Shadow Register **($0120_H + 0100_H * y$)** **Reset Value: 00000000_H**

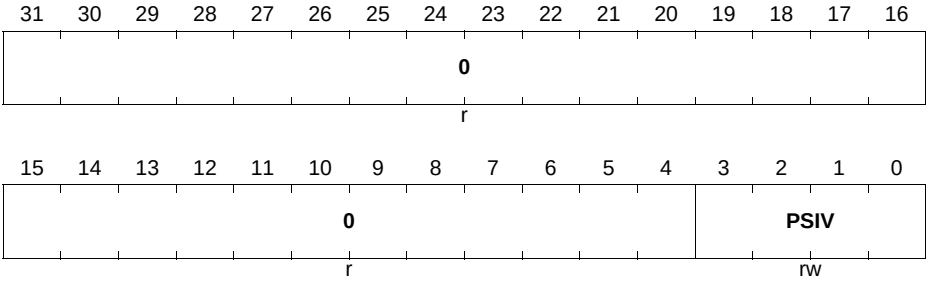


Field	Bits	Type	Description
DCVS	[3:0]	rw	**Dither Shadow Compare Value** This field contains the value that is going to be set on the dither compare value, **CC8yDIT**.DCV, within the next shadow transfer.
0	[31:4]	r	**Reserved** Read always returns 0.

CC8yPSC

This register contains the value that is loaded into the prescaler during restart.

CC8yPSC (y = 0 - 3)
Prescaler Control **(0124_H + 0100_H * y)** **Reset Value: 00000000_H**

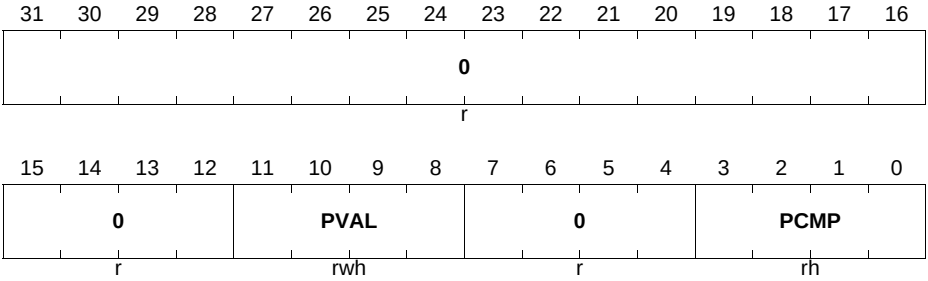


Field	Bits	Type	Description
PSIV	[3:0]	rw	**Prescaler Initial Value** This field contains the value that is applied to the Prescaler at startup. When floating prescaler mode is used, this value is applied when a timer compare match AND prescaler compare match occurs or when a capture event is triggered.
0	[31:4]	r	**Reserved** Read always returns 0.

CC8yFPC

This register contains the value used for the floating prescaler compare and the actual prescaler division value.

CC8yFPC (y = 0 - 3)
Floating Prescaler Control **(0128_H + 0100_H * y)** **Reset Value: 00000000_H**

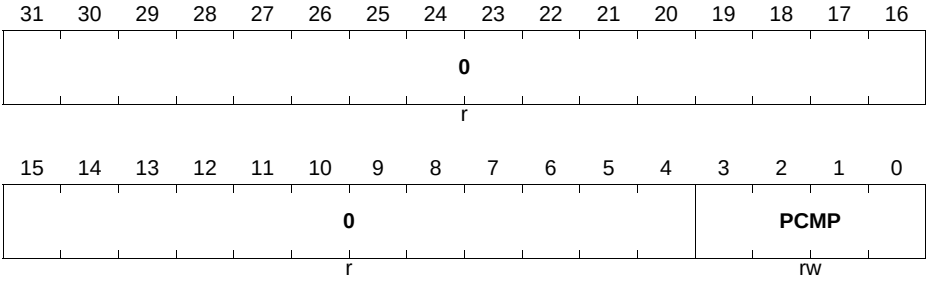


Field	Bits	Type	Description
PCMP	[3:0]	rh	**Floating Prescaler Compare Value** This field contains the value used to compare the actual prescaler value. The comparison is triggered by the Timer Compare match event. See **Section 19.2.13.2**.
PVAL	[11:8]	rwh	**Actual Prescaler Value** See **Table 19-8**. Writing into this register is only possible when the prescaler is stopped. When the floating prescaler mode is not used, this value is equal to the **CC8yPSC**.PSIV.
0	[7:4], [15:12], [31:16]	r	**Reserved** Read always returns 0.

CC8yFPCS

This register contains the value that is going to be transferred to the **CC8yFPC**.PCMP field within the next shadow transfer update.

CC8yFPCS (y = 0 - 3)
Floating Prescaler Shadow **($012C_H + 0100_H * y$)** **Reset Value: 00000000_H**

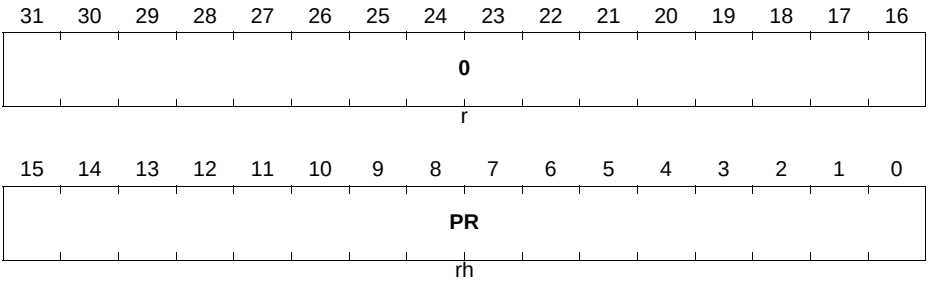


Field	Bits	Type	Description
PCMP	[3:0]	rw	**Floating Prescaler Shadow Compare Value** This field contains the value that is going to be set on the **CC8yFPC**.PCMP within the next shadow transfer. See **Table 19-8**.
0	[31:4]	r	**Reserved** Read always returns 0.

CC8yPR

This register contains the actual value for the timer period.

CC8yPR (y = 0 - 3)
Timer Period Value **($0130_H + 0100_H * y$)** **Reset Value: 00000000_H**

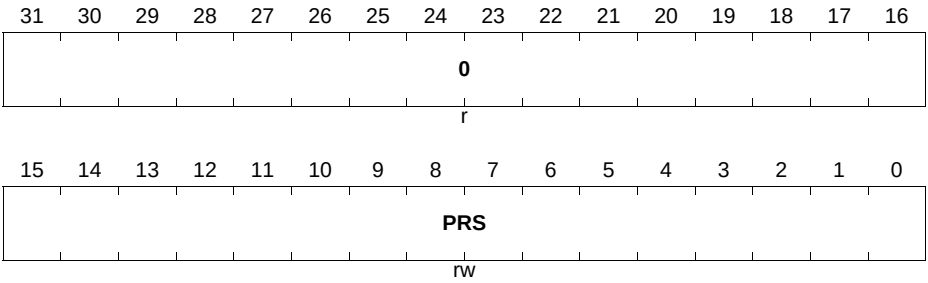


Field	Bits	Type	Description
PR	[15:0]	rh	**Period Register** Contains the value of the timer period.
0	[31:16]	r	**Reserved** A read always returns 0.

CC8yPRS

This register contains the value for the timer period that is going to be transferred into the **CC8yPR**.PR field when the next shadow transfer occurs.

CC8yPRS (y = 0 - 3)
Timer Shadow Period Value **($0134_H + 0100_H * y$)** **Reset Value: 00000000_H**

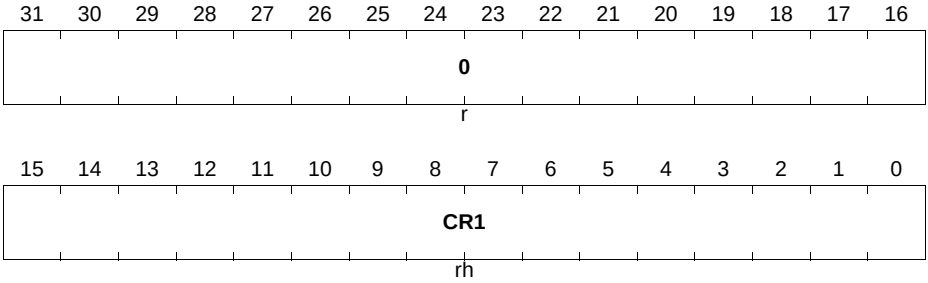


Field	Bits	Type	Description
PRS	[15:0]	rw	**Period Register** Contains the value of the timer period, that is going to be passed into the **CC8yPR**.PR field when the next shadow transfer occurs.
0	[31:16]	r	**Reserved** A read always returns 0.

CC8yCR1

This register contains the value for the timer comparison of channel 1.

CC8yCR1 (y = 0 - 3)
Channel 1 Compare Value **($0138_H + 0100_H * y$)** **Reset Value: 00000000_H**

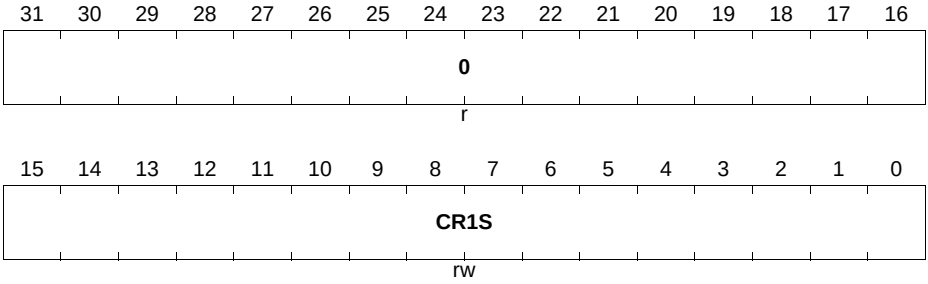


Field	Bits	Type	Description
CR1	[15:0]	rh	**Compare Register for Channel 1** Contains the value for the timer comparison. A write always addresses the shadow register, while a read returns the actual value. *Note: In Capture Mode when a external signal is selected for capturing the timer value into the capture registers 0 and 1, the CR is not accessible for writing. A read always returns 0.*
0	[31:16]	r	**Reserved** A read always returns 0.

CC8yCR1S

This register contains the value that is going to be loaded into the **CC8yCR1**.CR field when the next shadow transfer occurs.

CC8yCR1S (y = 0 - 3)
Channel 1 Compare Shadow Value($013C_H$ + 0100_H * y) **Reset Value: 00000000_H**

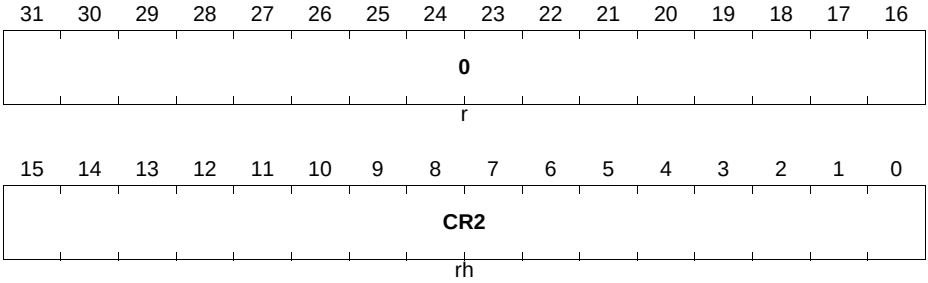


Field	Bits	Type	Description
CR1S	[15:0]	rw	**Shadow Compare Register for Channel 1** Contains the value for the timer comparison, that is going to be passed into the **CC8yCR1**.CR1 field when the next shadow transfer occurs. *Note: In Capture Mode when a external signal is selected for capturing the timer value into the capture registers 0 and 1, the CR is not accessible for writing. A read always returns 0.*
0	[31:16]	r	**Reserved** A read always returns 0.

CC8yCR2

This register contains the value for the timer comparison of channel 2.

CC8yCR2 (y = 0 - 3)
Channel 2 Compare Value **(0140_H + 0100_H * y)** **Reset Value: 00000000_H**

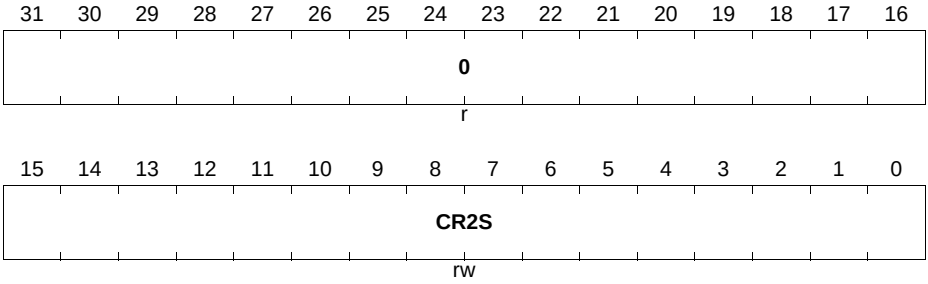


Field	Bits	Type	Description
CR2	[15:0]	rh	**Compare Register for Channel 2** Contains the value for the timer comparison. A write always addresses the shadow register, while a read returns the actual value. *Note: In Capture Mode when a external signal is selected for capturing the timer value into the capture registers 2 and 3, the CR is not accessible for writing. A read always returns 0.*
0	[31:16]	r	**Reserved** A read always returns 0.

CC8yCR2S

This register contains the value that is going to be loaded into the **CC8yCR2**.CR field when the next shadow transfer occurs.

CC8yCR2S (y = 0 - 3)
Channel 2 Compare Shadow Value(0144_H + 0100_H * y) **Reset Value: 00000000_H**

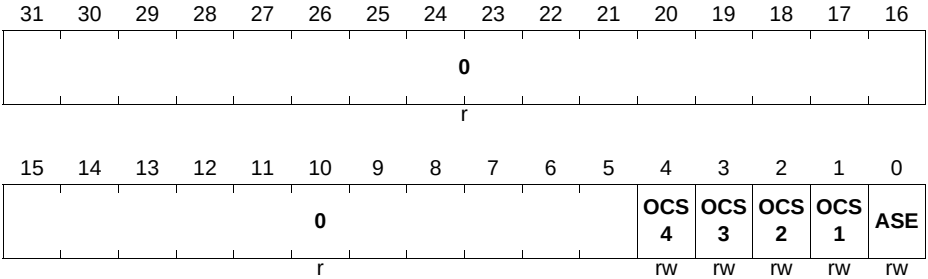


Field	Bits	Type	Description
CR2S	[15:0]	rw	**Shadow Compare Register for Channel 2** Contains the value for the timer comparison, that is going to be passed into the **CC8yCR2**.CR2 field when the next shadow transfer occurs. *Note: In Capture Mode when a external signal is selected for capturing the timer value into the capture registers 2 and 3, the CR is not accessible for writing. A read always returns 0.*
0	[31:16]	r	**Reserved** A read always returns 0.

CC8yCHC

This register contains the configuration for the output connections from the two compare channels and the enable for the asymmetric mode.

CC8yCHC (y = 0 - 3)
Channel Control **($0148_H + 0100_H * y$)** **Reset Value: 00000000_H**



Field	Bits	Type	Description
ASE	0	rw	**Asymmetric PWM mode Enable** 0_B Asymmetric PWM is disabled 1_B Asymmetric PWM is enabled
OCS1	1	rw	**Output selector for CCU8x.OUTy0** 0_B CC8yST1 signal path is connected to the CCU8x.OUTy0 1_B Inverted CC8yST1 signal path is connected to the CCU8x.OUTy0
OCS2	2	rw	**Output selector for CCU8x.OUTy1** 0_B Inverted CC8yST1 signal path is connected to the CCU8x.OUTy1 1_B CC8yST1 signal path is connected to the CCU8x.OUTy1
OCS3	3	rw	**Output selector for CCU8x.OUTy2** 0_B CC8yST2 signal path is connected to the CCU8x.OUTy2 1_B Inverted CCST2 signal path is connected to the CCU8x.OUTy2
OCS4	4	rw	**Output selector for CCU8x.OUTy3** 0_B Inverted CC8yST2 signal path is connected to the CCU8x.OUTy3 1_B CC8yST2 signal path is connected to the CCU8x.OUTy3
0	[31:5]	r	**Reserved** A read access always returns 0

CC8yDTC

This register contains the configuration for the dead time generator.

CC8yDTC (y = 0 - 3)
Dead Time Control **($014C_H + 0100_H * y$)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0								DTCC		DCEN4	DCEN3	DCEN2	DCEN1	DTE2	DTE1
r								rw		rw	rw	rw	rw	rw	rw

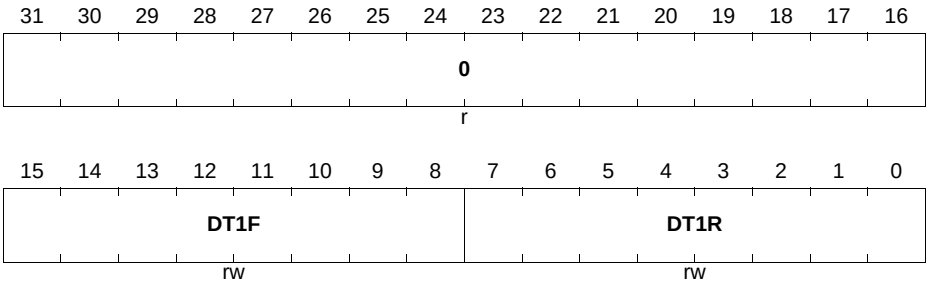
Field	Bits	Type	Description
DTE1	0	rw	**Dead Time Enable for Channel 1** This field enables the dead time counter for the compare channel 1. 0_B Dead Time for channel 1 is disabled 1_B Dead Time for channel 1 is enabled
DTE2	1	rw	**Dead Time Enable for Channel 2** This field enables the dead time counter for the compare channel 2. 0_B Dead Time for channel 2 is disabled 1_B Dead Time for channel 2 is enabled
DCEN1	2	rw	**Dead Time Enable for CC8yST1** 0_B Dead Time for CC8yST1 path is disabled 1_B Dead Time for CC8yST1 path is enabled
DCEN2	3	rw	**Dead Time Enable for inverted CC8yST1** 0_B Dead Time for inverted CC8yST1 path is disabled 1_B Dead Time for inverted CC8yST1 path is enabled
DCEN3	4	rw	**Dead Time Enable for CC8yST2** 0_B Dead Time for CC8yST2 path is disabled 1_B Dead Time for CC8yST2 path is enabled

Field	Bits	Type	Description
DCEN4	5	rw	**Dead Time Enable for inverted CC8yST2** 0_B Dead Time for inverted CC8yST2 path is disabled 1_B Dead Time for inverted CC8yST2 path is enabled
DTCC	[7:6]	rw	**Dead Time clock control** This field controls the prescaler clock configuration for the dead time counters. 00_B f_{tclk} 01_B $f_{tclk}/2$ 10_B $f_{tclk}/4$ 11_B $f_{tclk}/8$
0	[15:8], [31:16]	r	**Reserved** A read always returns 0.

CC8yDC1R

This register contains the dead time value for the compare channel 1.

CC8yDC1R (y = 0 - 3)
Channel 1 Dead Time Values **(0150_H + 0100_H * y)** **Reset Value: 00000000_H**



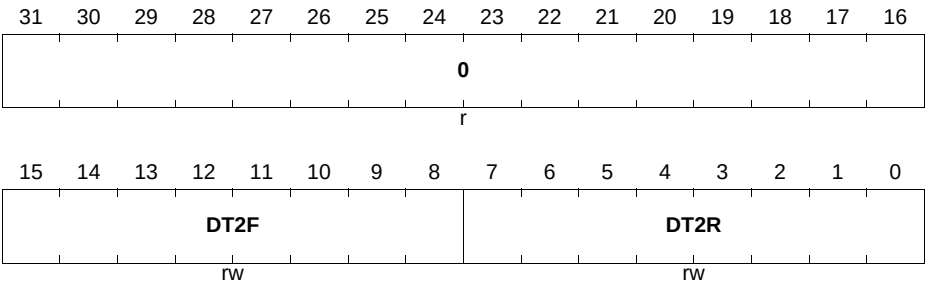
Field	Bits	Type	Description
DT1R	[7:0]	rw	**Rise Value for Dead Time of Channel 1** This field contains the delay value that is applied every time that a 0 to 1 transition occurs in the CC8yST1.

Field	Bits	Type	Description
DT1F	[15:8]	rw	**Fall Value for Dead Time of Channel 1** This field contains the delay value that is applied every time that a 1 to 0 transition occurs in the CC8yST1.
0	[31:16]	r	**Reserved** A read access always returns 0

CC8yDC2R

This register contains the dead time value for the compare channel 2.

CC8yDC2R (y = 0 - 3)
Channel 2 Dead Time Values **(0154_H + 0100_H * y)** **Reset Value: 00000000_H**

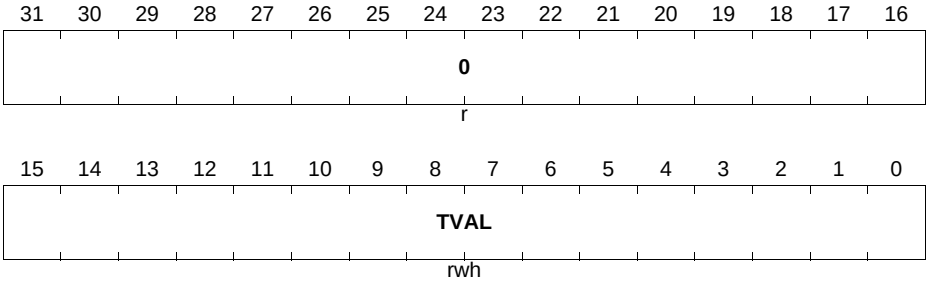


Field	Bits	Type	Description
DT2R	[7:0]	rw	**Rise Value for Dead Time of Channel 2** This field contains the delay value that is applied every time that a 0 to 1 transition occurs in the CC8yST2.
DT2F	[15:8]	rw	**Fall Value for Dead Time of Channel 2** This field contains the delay value that is applied every time that a 1 to 0 transition occurs in the CC8yST2.
0	[31:16]	r	**Reserved** A read access always returns 0

CC8yTIMER

This register contains the current value of the timer.

CC8yTIMER (y = 0 - 3)
Timer Value **($0170_H + 0100_H * y$)** **Reset Value: 00000000_H**

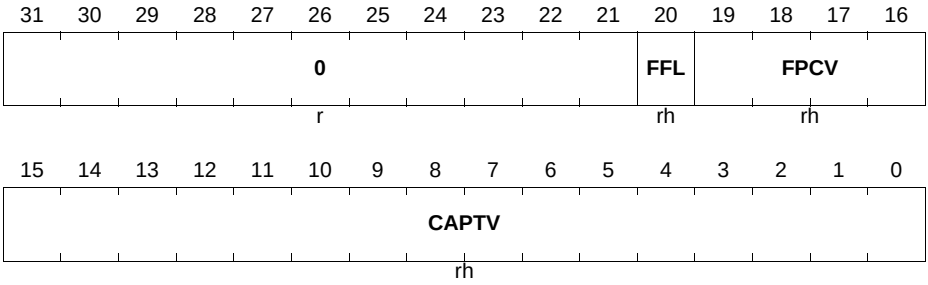


Field	Bits	Type	Description
TVAL	[15:0]	rwh	**Timer Value** This field contains the actual value of the timer. A write access is only possible when the timer is stopped.
0	[31:16]	r	**Reserved** A read access always returns 0

CC8yC0V

This register contains the values associated with the Capture 0 field.

CC8yC0V (y = 0 - 3)
Capture Register 0 **($0174_H + 0100_H * y$)** **Reset Value: 00000000_H**

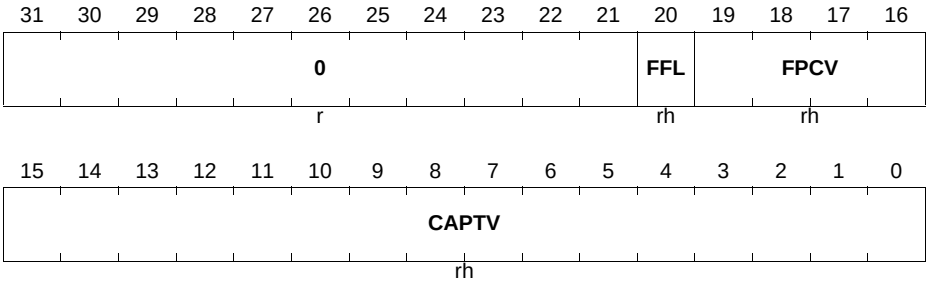


Field	Bits	Type	Description
CAPTV	[15:0]	rh	**Capture Value** This field contains the capture register 0 value. See **Figure 19-43**. In compare mode a read access always returns 0.
FPCV	[19:16]	rh	**Prescaler Value** This field contains the prescaler value when the time of the capture event into the capture register 0. In compare mode a read access always returns 0.
FFL	20	rh	**Full Flag** This bit indicates if a new value was capture into the capture register 0 after the last read access. See **Figure 19-43**. In compare mode a read access always returns 0. 0_B No new value was captured into the specific capture register 1_B A new value was captured into the specific register
0	[31:21]	r	**Reserved** A read always returns 0

CC8yC1V

This register contains the values associated with the Capture 1 field.

CC8yC1V (y = 0 - 3)
Capture Register 1 **($0178_H + 0100_H * y$)** **Reset Value: 00000000_H**

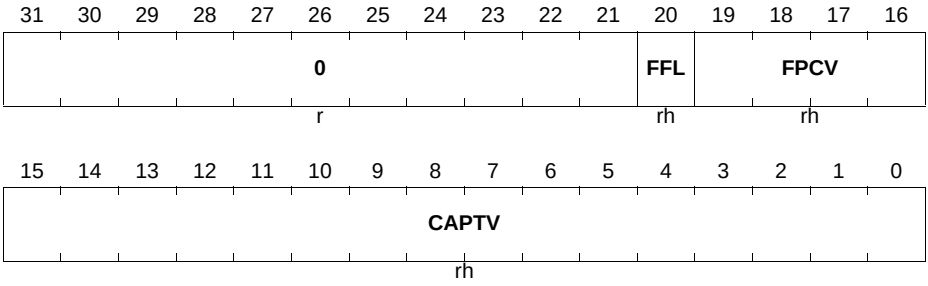


Field	Bits	Type	Description
CAPTV	[15:0]	rh	**Capture Value** This field contains the capture register 1 value. See **Figure 19-43**. In compare mode a read access always returns 0.
FPCV	[19:16]	rh	**Prescaler Value** This field contains the prescaler value when the time of the capture event into the capture register 1. In compare mode a read access always returns 0.
FFL	20	rh	**Full Flag** This bit indicates if a new value was capture into the capture register 1 after the last read access. See **Figure 19-43**. In compare mode a read access always returns 0. 0_B No new value was captured into the specific capture register 1_B A new value was captured into the specific register
0	[31:21]	r	**Reserved** A read always returns 0

CC8yC2V

This register contains the values associated with the Capture 2 field.

CC8yC2V (y = 0 - 3)
Capture Register 2 **($017C_H + 0100_H * y$)** **Reset Value: 00000000_H**

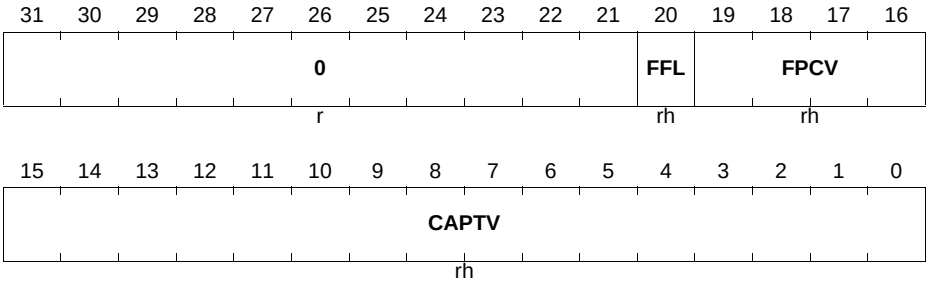


Field	Bits	Type	Description
CAPTV	[15:0]	rh	**Capture Value** This field contains the capture register 2 value. See **Figure 19-43**. In compare mode a read access always returns 0.
FPCV	[19:16]	rh	**Prescaler Value** This field contains the prescaler value when the time of the capture event into the capture register 2. In compare mode a read access always returns 0.
FFL	20	rh	**Full Flag** This bit indicates if a new value was capture into the capture register 2 after the last read access. See **Figure 19-43**. In compare mode a read access always returns 0. 0_B No new value was captured into the specific capture register 1_B A new value was captured into the specific register
0	[31:21]	r	**Reserved** A read always returns 0

CC8yC3V

This register contains the values associated with the Capture 3 field.

CC8yC3V (y = 0 - 3)
Capture Register 3 **($0180_H + 0100_H * y$)** **Reset Value: 00000000_H**



Field	Bits	Type	Description
CAPTV	[15:0]	rh	**Capture Value** This field contains the capture register 3 value. See **Figure 19-43**. In compare mode a read access always returns 0.
FPCV	[19:16]	rh	**Prescaler Value** This field contains the prescaler value when the time of the capture event into the capture register 3. In compare mode a read access always returns 0.
FFL	20	rh	**Full Flag** This bit indicates if a new value was capture into the capture register 3 after the last read access. See **Figure 19-43**. In compare mode a read access always returns 0. 0_B No new value was captured into the specific capture register 1_B A new value was captured into the specific register
0	[31:21]	r	**Reserved** A read always returns 0

CC8yINTS

This register contains the status of all interrupt sources.

CC8yINTS (y = 0 - 3)
Interrupt Status **($01A0_H + 0100_H * y$)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				TRPF	E2AS	E1AS	E0AS	0		CMD2S	CMU2S	CMD1S	CMU1S	OMDS	PMUS
r				rh	rh	rh	rh	r		rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
PMUS	0	rh	**Period Match while Counting Up** 0_B Period match while counting up not detected 1_B Period match while counting up detected
OMDS	1	rh	**One Match while Counting Down** 0_B One match while counting down not detected 1_B One match while counting down detected
CMU1S	2	rh	**Channel 1 Compare Match while Counting Up** 0_B Compare match while counting up not detected 1_B Compare match while counting up detected
CMD1S	3	rh	**Channel 1 Compare Match while Counting Down** 0_B Compare match while counting down not detected 1_B Compare match while counting down detected
CMU2S	4	rh	**Channel 2 Compare Match while Counting Up** 0_B Compare match while counting up not detected 1_B Compare match while counting up detected
CMD2S	5	rh	**Channel 2 Compare Match while Counting Down** 0_B Compare match while counting down not detected 1_B Compare match while counting down detected
E0AS	8	rh	**Event 0 Detection Status** Depending on the user selection on the **CC8yINS**.EV0EM, this bit can be set when a rising, falling or both transitions are detected. 0_B Event 0 not detected 1_B Event 0 detected
E1AS	9	rh	**Event 1 Detection Status** Depending on the user selection on the **CC8yINS**.EV1EM, this bit can be set when a rising, falling or both transitions are detected. 0_B Event 1 not detected 1_B Event 1 detected

Field	Bits	Type	Description
E2AS	10	rh	**Event 2 Detection Status** Depending on the user selection on the **CC8yINS**.EV1EM, this bit can be set when a rising, falling or both transitions are detected. 0_B Event 2 not detected 1_B Event 2 detected *Note: If this event is linked with the TRAP function, this field is automatically cleared when the slice exits the Trap State.*
TRPF	11	rh	**Trap Flag Status** This field contains the status of the Trap Flag.
0	[7:6], [31:12]	r	**Reserved** A read always returns 0.

CC8yINTE

Through this register it is possible to enable or disable the specific interrupt source(s).

CC8yINTE (y = 0 - 3)
Interrupt Enable Control **($01A4_H + 0100_H * y$)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0					E2AE	E1AE	E0AE	0		CMD2E	CMU2E	CMD1E	CMU1E	OME	PME
r					rw	rw	rw	r		rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
PME	0	rw	**Period match while counting up enable** Setting this bit to 1_B enables the generation of an interrupt pulse every time a period match while counting up occurs. 0_B Period Match interrupt is disabled 1_B Period Match interrupt is enabled

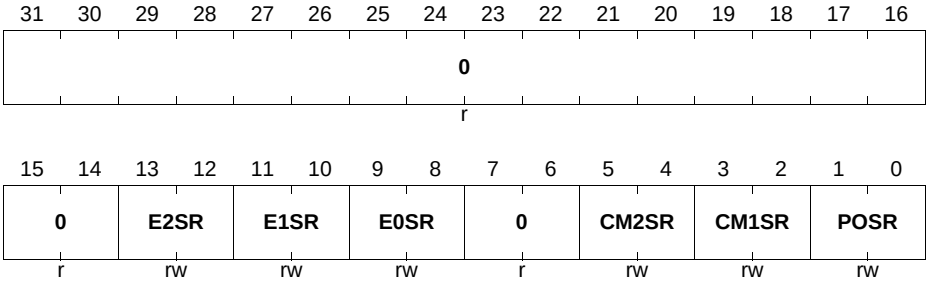
Field	Bits	Type	Description
OME	1	rw	**One match while counting down enable** Setting this bit to 1_B enables the generation of an interrupt pulse every time an one match while counting down occurs. 0_B One Match interrupt is disabled 1_B One Match interrupt is enabled
CMU1E	2	rw	**Channel 1 Compare match while counting up enable** Setting this bit to 1_B enables the generation of an interrupt pulse every time a compare match while counting up occurs. 0_B Compare Match while counting up interrupt is disabled 1_B Compare Match while counting up interrupt is enabled
CMD1E	3	rw	**Channel 1 Compare match while counting down enable** Setting this bit to 1_B enables the generation of an interrupt pulse every time a compare match while counting down occurs. 0_B Compare Match while counting down interrupt is disabled 1_B Compare Match while counting down interrupt is enabled
CMU2E	4	rw	**Channel 2 Compare match while counting up enable** Setting this bit to 1_B enables the generation of an interrupt pulse every time a compare match while counting up occurs. 0_B Compare Match while counting up interrupt is disabled 1_B Compare Match while counting up interrupt is enabled

Field	Bits	Type	Description
CMD2E	5	rw	**Channel 2 Compare match while counting down enable** Setting this bit to 1_B enables the generation of an interrupt pulse every time a compare match while counting down occurs. 0_B Compare Match while counting down interrupt is disabled 1_B Compare Match while counting down interrupt is enabled
E0AE	8	rw	**Event 0 interrupt enable** Setting this bit to 1_B enables the generation of an interrupt pulse every time that Event 0 is detected. 0_B Event 0 detection interrupt is disabled 1_B Event 0 detection interrupt is enabled
E1AE	9	rw	**Event 1 interrupt enable** Setting this bit to 1_B enables the generation of an interrupt pulse every time that Event 1 is detected. 0_B Event 1 detection interrupt is disabled 1_B Event 1 detection interrupt is enabled
E2AE	10	rw	**Event 2 interrupt enable** Setting this bit to 1_B enables the generation of an interrupt pulse every time that Event 2 is detected. 0_B Event 2 detection interrupt is disabled 1_B Event 2 detection interrupt is enabled
0	[7:6], [31:11]	r	**Reserved** A read always returns 0

CC8ySRS

Through this register it is possible to select to which service request line each interrupt source is forwarded.

CC8ySRS (y = 0 - 3)
Service Request Selector **($01A8_H$ + 0100_H * y)** **Reset Value: 00000000_H**



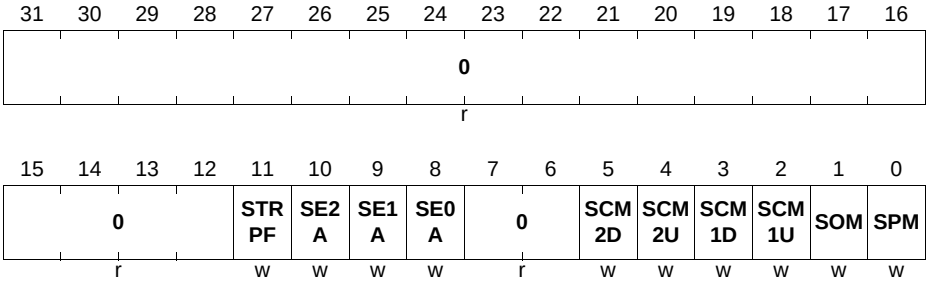
Field	Bits	Type	Description
POSR	[1:0]	rw	**Period/One match Service request selector** This field selects to which slice Service request line, the interrupt(s) generated by the Period match while counting up and One match while counting down are going to be forward. 00_B Forward to CC8ySR0 01_B Forward to CC8ySR1 10_B Forward to CC8ySR2 11_B Forward to CC8ySR3
CM1SR	[3:2]	rw	**Channel 1 Compare match Service request selector** This field selects to which slice Service request line, the interrupt(s) generated by the Compare match, of channel 1, while counting up and Compare match while counting down are going to be forward. 00_B Forward to CC8ySR0 01_B Forward to CC8ySR1 10_B Forward to CC8ySR2 11_B Forward to CC8ySR3

Field	Bits	Type	Description
CM2SR	[5:4]	rw	**Channel 2 Compare match Service request selector** This field selects to which slice Service request line, the interrupt(s) generated by the Compare match, of channel 2, while counting up and Compare match while counting down are going to be forward. 00_B Forward to CC8ySR0 01_B Forward to CC8ySR1 10_B Forward to CC8ySR2 11_B Forward to CC8ySR3
E0SR	[9:8]	rw	**Event 0 Service request selector** This field selects to which slice Service request line, the interrupt generated by the Event 0 detection are going to be forward. 00_B Forward to CCvySR0 01_B Forward to CC8ySR1 10_B Forward to CC8ySR2 11_B Forward to CC8ySR3
E1SR	[11:10]	rw	**Event 1 Service request selector** This field selects to which slice Service request line, the interrupt generated by the Event 1detection are going to be forward. 00_B Forward to CC8ySR0 01_B Forward to CC8ySR1 10_B Forward to CC8ySR2 11_B Forward to CC8ySR3
E2SR	[13:12]	rw	**Event 2 Service request selector** This field selects to which slice Service request line, the interrupt generated by the Event 2 detection are going to be forward. 00_B Forward to CC8ySR0 01_B Forward to CCvySR1 10_B Forward to CC8ySR2 11_B Forward to CC8ySR3
0	[7:6], [31:14]	r	**Reserved** Read always returns 0.

CC8ySWS

Through this register it is possible for the SW to set a specific interrupt status flag.

CC8ySWS (y = 0 - 3)
Interrupt Status Set **($01AC_H + 0100_H * y$)** **Reset Value: 00000000_H**



Field	Bits	Type	Description
SPM	0	w	**Period match while counting up set** Writing a 1_B into this field sets the **CC8yINTS**.PMUS bit. An interrupt pulse is generated if the source is enabled. A read always returns 0.
SOM	1	w	**One match while counting down set** Writing a 1_B into this bit sets the **CC8yINTS**.OMDS bit. An interrupt pulse is generated if the source is enabled. A read always returns 0.
SCM1U	2	w	**Channel 1 Compare match while counting up set** Writing a 1_B into this field sets the **CC8yINTS**.CMU1S bit. An interrupt pulse is generated if the source is enabled. A read always returns 0.
SCM1D	3	w	**Channel 1 Compare match while counting down set** Writing a 1_B into this bit sets the **CC8yINTS**.CMD1S bit. An interrupt pulse is generated if the source is enabled. A read always returns 0.
SCM2U	4	w	**Compare match while counting up set** Writing a 1_B into this field sets the **CC8yINTS**.CMU2S bit. An interrupt pulse is generated if the source is enabled. A read always returns 0.

Field	Bits	Type	Description
SCM2D	5	w	**Compare match while counting down set** Writing a 1_B into this bit sets the **CC8yINTS**.CMD2S bit. An interrupt pulse is generated if the source is enabled. A read always returns 0.
SE0A	8	w	**Event 0 detection set** Writing a 1_B into this bit sets the **CC8yINTS**.E0AS bit. An interrupt pulse is generated if the source is enabled. A read always returns 0.
SE1A	9	w	**Event 1 detection set** Writing a 1_B into this bit sets the **CC8yINTS**.E1AS bit. An interrupt pulse is generated if the source is enabled. A read always returns 0.
SE2A	10	w	**Event 2 detection set** Writing a 1_B into this bit sets the **CC8yINTS**.E2AS bit. An interrupt pulse is generated if the source is enabled. A read always returns 0.
STRPF	11	w	**Trap Flag status set** Writing a 1_B into this bit sets the **CC8yINTS**.TRPF bit. A read always returns 0.
0	[7:6], [31:12]	r	**Reserved** Read always returns 0

CC8ySWR

Through this register it is possible for the SW to clear a specific interrupt status flag.

CC8ySWR (y = 0 - 3)
Interrupt Status Clear **($01B0_H + 0100_H * y$)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				RTRPF	RE2A	RE1A	RE0A	0		RCM2D	RCM2U	RCM1D	RCM1U	ROM	RPM
r				w	w	w	w	r		w	w	w	w	w	w

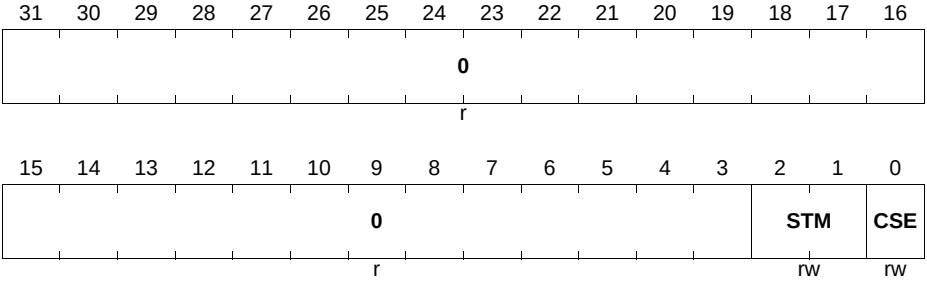
Field	Bits	Type	Description
RPM	0	w	**Period match while counting up clear** Writing a 1_B into this field clears the **CC8yINTS**.PMUS bit. A read always returns 0.
ROM	1	w	**One match while counting down clear** Writing a 1 into this bit clears the **CC8yINTS**.OMDS bit. A read always returns 0.
RCM1U	2	w	**Channel 1 Compare match while counting up clear** Writing a 1_B into this field clears the **CC8yINTS**.CMU1S bit. A read always returns 0.
RCM1D	3	w	**Channel 1 Compare match while counting down clear** Writing a 1_B into this bit clears the **CC8yINTS**.CMD1S bit. A read always returns 0.
RCM2U	4	w	**Channel 2 Compare match while counting up clear** Writing a 1_B into this field clears the **CC8yINTS**.CMU2S bit. A read always returns 0.
RCM2D	5	w	**Channel 2 Compare match while counting down clear** Writing a 1_B into this bit clears the **CC8yINTS**.CMD2S bit. A read always returns 0.
RE0A	8	w	**Event 0 detection clear** Writing a 1_B into this bit clears the **CC8yINTS**.E0AS bit. A read always returns 0.
RE1A	9	w	**Event 1 detection clear** Writing a 1_B into this bit clears the **CC8yINTS**.E1AS bit. A read always returns 0.
RE2A	10	w	**Event 2 detection clear** Writing a 1_B into this bit clears the **CC8yINTS**.E2AS bit. A read always returns 0.
RTRPF	11	w	**Trap Flag status clear** Writing a 1_B into this bit clears the **CC8yINTS**.TRPF bit. Not valid if **CC8yTC**.TRPEN = 1_B and the Trap State is still active. A read always returns 0.

Field	Bits	Type	Description
0	[7:6], [31:12]	r	**Reserved** Read always returns 0

CC8ySTC

Through this register it is possible to configure the extended options for the shadow transfer mechanism.

CC8ySTC (y = 0 - 3)
Shadow transfer control **($01B4_H + 0100_H * y$)** **Reset Value: 00000000_H**



Field	Bits	Type	Description
CSE	0	rw	**Cascaded shadow transfer enable** 0_B Cascaded shadow transfer disabled 1_B Cascaded shadow transfer enabled
STM	[2:1]	rw	**Shadow transfer mode** 00_B Shadow transfer is done in Period Match and One match. 01_B Shadow transfer is done only in Period Match. 10_B Shadow transfer is done only in One Match. 11_B Reserved *Note: This field only has effect if the timer is in Center Aligned Mode.*
0	[31:3]	r	**Reserved** Read always returns 0

19.8 Interconnects

The tables that refer to the “global pins” are the ones that contain the inputs/outputs of each module that are common to all slices.

The GPIO mapping is available at the Ports unit.

19.8.1 CCU80 Pins

Table 19-14 CCU80 Pin Connections

Global Inputs/Outputs	I/O	Connected To	Description
CCU80.MCLK	I	SCU.CCUCLK	Kernel clock
CCU80.CLKA	I	ERU1.IOUT0	another count source for the prescaler
CCU80.CLKB	I	ERU1.IOUT1	another count source for the prescaler
CCU80.CLKC	I	0	another count source for the prescaler
CCU80.MCSS	I	POSIF0.OUT6	Multi pattern sync with shadow transfer trigger
CCU80.IGBTA	I	CCU40.ST3;	Parity Checker delay finish trigger
CCU80.IGBTB	I	CCU40.SR3;	Parity Checker delay finish trigger
CCU80.IGBTC	I	0	Parity Checker delay finish trigger
CCU80.IGBTD	I	0	Parity Checker delay finish trigger
CCU80.IGBTO	O	CCU40.IN3H;	Parity Checker delay start trigger
CCU80.SR0	O	NVIC; DMA; VADC.G0REQGTM; VADC.G1REQGTM; VADC.BGREQGTM;	Service request line

Table 19-14 CCU80 Pin Connections

Global Inputs/Outputs	I/O	Connected To	Description
CCU80.SR1	O	NVIC; DMA; DAC.TRIGGER[0]; POSIF0.MSETE; VADC.G0REQGTN; VADC.G1REQGTN; U0C0.DX2F; U0C1.DX2F; VADC.BGREQGTN	Service request line
CCU80.SR2	O	NVIC; VADC.G0REQTRI; VADC.G1REQTRI; VADC.BGREQTRI;	Service request line
CCU80.SR3	O	NVIC; VADC.G0REQTRJ; VADC.G1REQTRJ; VADC.BGREQTRJ;	Service request line

Table 19-15 CCU80 - CC80 Pin Connections

Input/Output	I/O	Connected To	Description
CCU80.IN0A	I	PORTS	General purpose function
CCU80.IN0B	I	PORTS	General purpose function
CCU80.IN0C	I	PORTS	General purpose function
CCU80.IN0D	I	POSIF0.OUT2	General purpose function
CCU80.IN0E	I	POSIF0.OUT5	General purpose function
CCU80.IN0F	I	VADC.G0SR3	General purpose function
CCU80.IN0G	I	ERU1.IOUT0	General purpose function
CCU80.IN0H	I	SCU.GSC80	General purpose function
CCU80.IN0I	I	VADC.G0BFL0	General purpose function
CCU80.IN0J	I	ERU1.PDOUT0	General purpose function
CCU80.IN0K	I	CCU40.SR3	General purpose function
CCU80.IN0L	I	HRPWM0.SR0	General purpose function
CCU80.IN0M	I	HRPWM0.C0O	General purpose function
CCU80.IN0N	I	CCU80.ST1	General purpose function

Table 19-15 CCU80 - CC80 Pin Connections

Input/Output	I/O	Connected To	Description
CCU80.IN0O	I	CCU80.ST2	General purpose function
CCU80.IN0P	I	CCU80.ST3	General purpose function
CCU80.MCI00	I	POSIF0.MOUT[0]	Multi Channel pattern input for CCST1
CCU80.MCI01	I	POSIF0.MOUT[1]	Multi Channel pattern input for NOT(CCST1)
CCU80.MCI02	I	POSIF0.MOUT[2]	Multi Channel pattern input for CCST2
CCU80.MCI03	I	POSIF0.MOUT[3]	Multi Channel pattern input for NOT(CCST2)
CCU80.OUT00	O	PORTS	Slice compare output from channel 1. Can be the CCST1 or NOT(CCST1) path
CCU80.OUT01	O	PORTS	Slice compare output from channel 1. Can be the CCST1 or NOT(CCST1) path
CCU80.OUT02	O	PORTS	Slice compare output from channel 2. Can be the CCST2 or NOT(CCST2) path
CCU80.OUT03	O	PORTS	Slice compare output from channel 2. Can be the CCST2 or NOT(CCST2) path
CCU80.GP00	O	NOT CONNECTED	Selected signal for event 0
CCU80.GP01	O	NOT CONNECTED	Selected signal for event 1
CCU80.GP02	O	NOT CONNECTED	Selected signal for event 2
CCU80.ST0	O	ERU1.0B1; HRPWM0.BL0C; HRPWM0.BL1C; HRPWM0.BL2C	Output of the status bit multiplexer. It can be CCST1 or CCST2
CCU80.ST0A	O	NOT CONNECTED	Channel 1 status bit: CCST1

Table 19-15 CCU80 - CC80 Pin Connections

Input/Output	I/O	Connected To	Description
CCU80.ST0B	O	NOT CONNECTED	Channel 2 status bit: CCST2
CCU80.PS0	O	NOT CONNECTED	Multi channel pattern sync trigger: PM when counting UP (edge aligned) or OM when counting DOWN (center aligned)

Table 19-16 CCU80 - CC81 Pin Connections

Input/Output	I/O	Connected To	Description
CCU80.IN1A	I	PORTS	General purpose function
CCU80.IN1B	I	PORTS	General purpose function
CCU80.IN1C	I	PORTS	General purpose function
CCU80.IN1D	I	POSIF0.OUT2	General purpose function
CCU80.IN1E	I	POSIF0.OUT5	General purpose function
CCU80.IN1F	I	ERU1.PDOUT1	General purpose function
CCU80.IN1G	I	ERU1.IOUT1	General purpose function
CCU80.IN1H	I	SCU.GSC80	General purpose function
CCU80.IN1I	I	VADC.G0BFL1	General purpose function
CCU80.IN1J	I	ERU1.PDOUT0	General purpose function
CCU80.IN1K	I	CCU41.SR3	General purpose function
CCU80.IN1L	I	HRPWM0.SR1	General purpose function
CCU80.IN1M	I	CCU80.ST0	General purpose function
CCU80.IN1N	I	HRPWM0.C1O	General purpose function
CCU80.IN1O	I	CCU80.ST2	General purpose function
CCU80.IN1P	I	CCU80.ST3	General purpose function
CCU80.MCI10	I	POSIF0.MOUT[4]	Multi Channel pattern input for CCST1
CCU80.MCI11	I	POSIF0.MOUT[5]	Multi Channel pattern input for NOT(CCST1)
CCU80.MCI12	I	POSIF0.MOUT[6]	Multi Channel pattern input for CCST2

Table 19-16 CCU80 - CC81 Pin Connections

Input/Output	I/O	Connected To	Description
CCU80.MCI13	I	POSIF0.MOUT[7]	Multi Channel pattern input for NOT(CCST2)
CCU80.OUT10	O	PORTS	Slice compare output from channel 1. Can be the CCST1 or NOT(CCST1) path
CCU80.OUT11	O	PORTS	Slice compare output from channel 1. Can be the CCST1 or NOT(CCST1) path
CCU80.OUT12	O	PORTS	Slice compare output from channel 2. Can be the CCST2 or NOT(CCST2) path
CCU80.OUT13	O	PORTS	Slice compare output from channel 2. Can be the CCST2 or NOT(CCST2) path
CCU80.GP10	O	NOT CONNECTED	Selected signal for event 0
CCU80.GP11	O	NOT CONNECTED	Selected signal for event 1
CCU80.GP12	O	NOT CONNECTED	Selected signal for event 2
CCU80.ST1	O	ERU1.1B1; HRPWM0.BL0D; HRPWM0.BL1D; HRPWM0.BL2D	Output of the status bit multiplexer. It can be CCST1 or CCST2
CCU80.ST1A	O	NOT CONNECTED	Channel 1 status bit: CCST1
CCU80.ST1B	O	NOT CONNECTED	Channel 2 status bit: CCST2
CCU80.PS1	O	POSIF0.MSYNCA	Multi channel pattern sync trigger: PM when counting UP (edge aligned) or OM when counting DOWN (center aligned)

Table 19-17 CCU80 - CC82 Pin Connections

Input/Output	I/O	Connected To	Description
CCU80.IN2A	I	PORTS	General purpose function
CCU80.IN2B	I	PORTS	General purpose function
CCU80.IN2C	I	PORTS	General purpose function

Table 19-17 CCU80 - CC82 Pin Connections

Input/Output	I/O	Connected To	Description
CCU80.IN2D	I	POSIF0.OUT2	General purpose function
CCU80.IN2E	I	POSIF0.OUT5	General purpose function
CCU80.IN2F	I	ERU1.PDOUT2	General purpose function
CCU80.IN2G	I	ERU1.IOUT2	General purpose function
CCU80.IN2H	I	SCU.GSC80	General purpose function
CCU80.IN2I	I	VADC.G0BFL2	General purpose function
CCU80.IN2J	I	ERU1.PDOUT0	General purpose function
CCU80.IN2K	I	0	General purpose function
CCU80.IN2L	I	HRPWM0.SR2	General purpose function
CCU80.IN2M	I	CCU80.ST0	General purpose function
CCU80.IN2N	I	CCU80.ST1	General purpose function
CCU80.IN2O	I	HRPWM0.C2O	General purpose function
CCU80.IN2P	I	CCU80.ST3	General purpose function
CCU80.MCI20	I	POSIF0.MOUT[8]	Multi Channel pattern input for CCST1
CCU80.MCI21	I	POSIF0.MOUT[9]	Multi Channel pattern input for NOT(CCST1)
CCU80.MCI22	I	POSIF0.MOUT[10]	Multi Channel pattern input for CCST2
CCU80.MCI23	I	POSIF0.MOUT[11]	Multi Channel pattern input for NOT(CCST2)
CCU80.OUT20	O	PORTS	Slice compare output from channel 1. Can be the CCST1 or NOT(CCST1) path
CCU80.OUT21	O	PORTS	Slice compare output from channel 1. Can be the CCST1 or NOT(CCST1) path
CCU80.OUT22	O	PORTS	Slice compare output from channel 2. Can be the CCST2 or NOT(CCST2) path
CCU80.OUT23	O	PORTS	Slice compare output from channel 2. Can be the CCST2 or NOT(CCST2) path

Table 19-17 CCU80 - CC82 Pin Connections

Input/Output	I/O	Connected To	Description
CCU80.GP20	O	NOT CONNECTED	Selected signal for event 0
CCU80.GP21	O	NOT CONNECTED	Selected signal for event 1
CCU80.GP22	O	NOT CONNECTED	Selected signal for event 2
CCU80.ST2	O	ERU1.2B1; HRPWM0.BL0E; HRPWM0.BL1E; HRPWM0.BL2E	Output of the status bit multiplexer. It can be CCST1 or CCST2
CCU80.ST2A	O	NOT CONNECTED	Channel 1 status bit: CCST1
CCU80.ST2B	O	NOT CONNECTED	Channel 2 status bit: CCST2
CCU80.PS2	O	NOT CONNECTED	Multi channel pattern sync trigger: PM when counting UP (edge aligned) or OM when counting DOWN (center aligned)

Table 19-18 CCU80 - CC83 Pin Connections

Input/Output	I/O	Connected To	Description
CCU80.IN3A	I	PORTS	General purpose function
CCU80.IN3B	I	PORTS	General purpose function
CCU80.IN3C	I	PORTS	General purpose function
CCU80.IN3D	I	POSIF0.OUT2	General purpose function
CCU80.IN3E	I	POSIF0.OUT5	General purpose function
CCU80.IN3F	I	ERU1.PDOUT3	General purpose function
CCU80.IN3G	I	ERU1.IOUT3	General purpose function
CCU80.IN3H	I	SCU.GSC80	General purpose function
CCU80.IN3I	I	VADC.G0BFL3	General purpose function
CCU80.IN3J	I	ERU1.PDOUT0	General purpose function
CCU80.IN3K	I	0	General purpose function
CCU80.IN3L	I	HRPWM0.SR3	General purpose function
CCU80.IN3M	I	CCU80.ST0	General purpose function
CCU80.IN3N	I	CCU80.ST1	General purpose function
CCU80.IN3O	I	CCU80.ST2	General purpose function

Table 19-18 CCU80 - CC83 Pin Connections

Input/Output	I/O	Connected To	Description
CCU80.IN3P	I	HRPWM0.C0O	General purpose function
CCU80.MCI30	I	POSIF0.MOUT[12]	Multi Channel pattern input for CCST1
CCU80.MCI31	I	POSIF0.MOUT[13]	Multi Channel pattern input for NOT(CCST1)
CCU80.MCI32	I	POSIF0.MOUT[14]	Multi Channel pattern input for CCST2
CCU80.MCI33	I	POSIF0.MOUT[15]	Multi Channel pattern input for NOT(CCST2)
CCU80.OUT30	O	PORTS	Slice compare output from channel 1. Can be the CCST1 or NOT(CCST1) path
CCU80.OUT31	O	PORTS	Slice compare output from channel 1. Can be the CCST1 or NOT(CCST1) path
CCU80.OUT32	O	PORTS	Slice compare output from channel 2. Can be the CCST2 or NOT(CCST2) path
CCU80.OUT33	O	PORTS	Slice compare output from channel 2. Can be the CCST2 or NOT(CCST2) path
CCU80.GP30	O	NOT CONNECTED	Selected signal for event 0
CCU80.GP31	O	NOT CONNECTED	Selected signal for event 1
CCU80.GP32	O	NOT CONNECTED	Selected signal for event 2
CCU80.ST3	O	ERU1.3B1; HRPWM0.BL0F; HRPWM0.BL1F; HRPWM0.BL2F	Output of the status bit multiplexer. It can be CCST1 or CCST2
CCU80.ST3A	O	VADC.G0REQGTE; VADC.G1REQGTE; VADC.BGREQGTE;	Channel 1 status bit: CCST1

Table 19-18 CCU80 - CC83 Pin Connections

Input/Output	I/O	Connected To	Description
CCU80.ST3B	O	VADC.G0REQGTF; VADC.G1REQGTF; VADC.BGREQGTF;	Channel 2 status bit: CCST2
CCU80.PS3	O	POSIF0.MSYNCB	Multi channel pattern sync trigger: PM when counting UP (edge aligned) or OM when counting DOWN (center aligned)

20 High Resolution PWM Unit (HRPWM)

The HRPWM extends the capabilities of the associated Capture/Compare Unit with a suite of features especially tailored for Switched Mode Power Supply (SMPS) applications.

These features not only enable an easy and fast control loop (voltage or current control) development for SMPS, that can run up to 6 MHz (with more than 10 bit resolution), but also helps the application developer to have an optimized solution in terms of total number of external components, avoid susceptibility to environmental and process variations and reducing the size of the power supply.

The features of the HRPWM make it a must have module, for cutting edge and optimized SMPS applications development. Together with a powerful processor architecture, standard SMPS topologies are easily implemented and newly developed ones, no longer need to wait for the market to provide the proper control solution.

Table 20-1 Abbreviations table

PWM	Pulse Width Modulation
HRPWMx	High Resolution PWM module instance x
HRCy	High Resolution Channel unit instance y
CSGy	Comparator and Slope Generator unit instance y
CCU8x/CCU4x	Capture/Compare Unit 8/4 module instance x
CC8y/CC4y	Capture/Compare Unit 8/4 Timer Slice instance y
SCU	System Control Unit
f_{hrpwm}	HRPWM module clock frequency

Note:

8. A small “y” or “x” letter in a register translates into an index.
9. HRPWM is associated with Capture and Compare Unit 8 instance 0.

20.1 Overview

The control schemes of a typical SMPS converter are mainly divided into two basic methods:

- Voltage Mode Control
- Current Mode Control

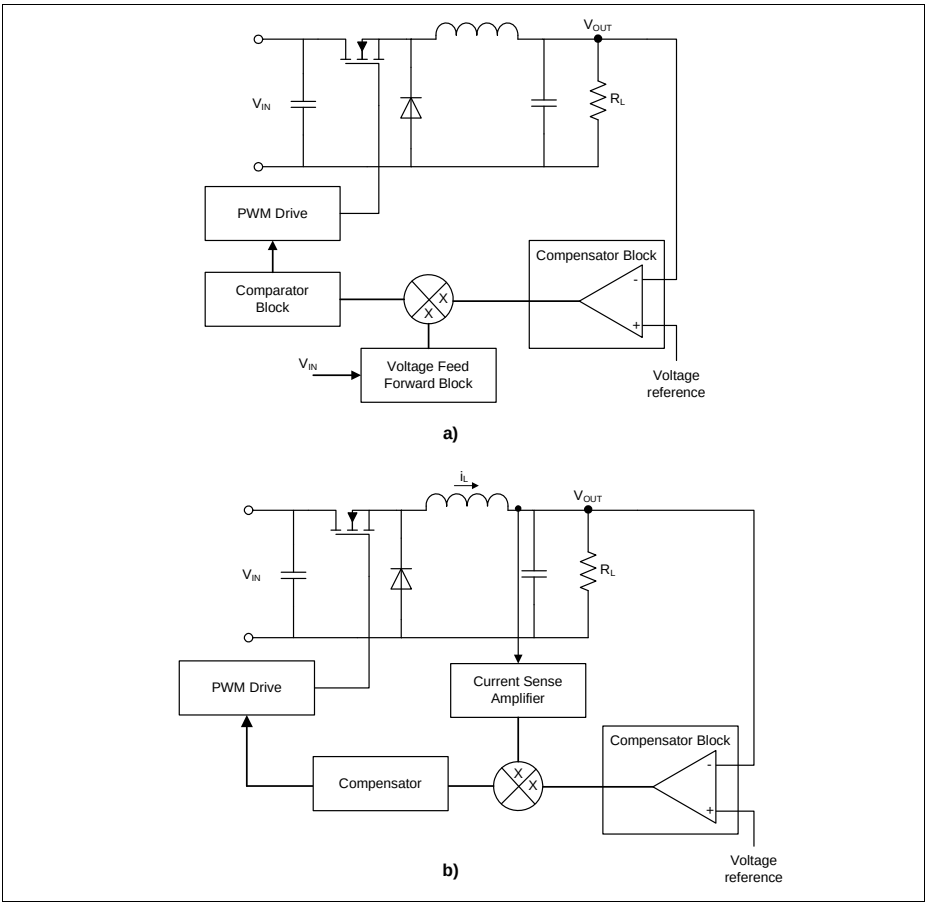


Figure 20-1 Different control methods: a) Voltage Mode Control, b) Current Mode Control

In a Voltage Mode Control, the generated output voltage is measured and compared against a reference value. The error between the two voltages is then processed via a compensation block, with the objective of generating a new duty cycle value for the PWM signal, **Figure 20-1**.

This method has only one control loop, making it simple to implement and to analyze. Nevertheless, with this method, any change on the line or load needs to be sensed as an output voltage change. Only after that, it can be corrected by the feedback loop which imposes a slow transient response.

The Current Control Method implements two feedback loops. One loop is sensing the voltage output at the load end while the other loop is sensing the output inductor current, **Figure 20-1**.

Like in the Voltage Control Method, the output voltage is compared against the wanted reference. This generated processed error is then used as reference for the current loop.

Due to the fact that in the Current Control Method the inductor current is monitored, any change in the input voltage or in the load current, can be corrected before it influences the output voltage.

The Current Control Method can be subdivided into different control techniques like: peak current control, average current control, hysteretic control, valley current control, or constant ON or OFF time control (**Figure 20-2**).

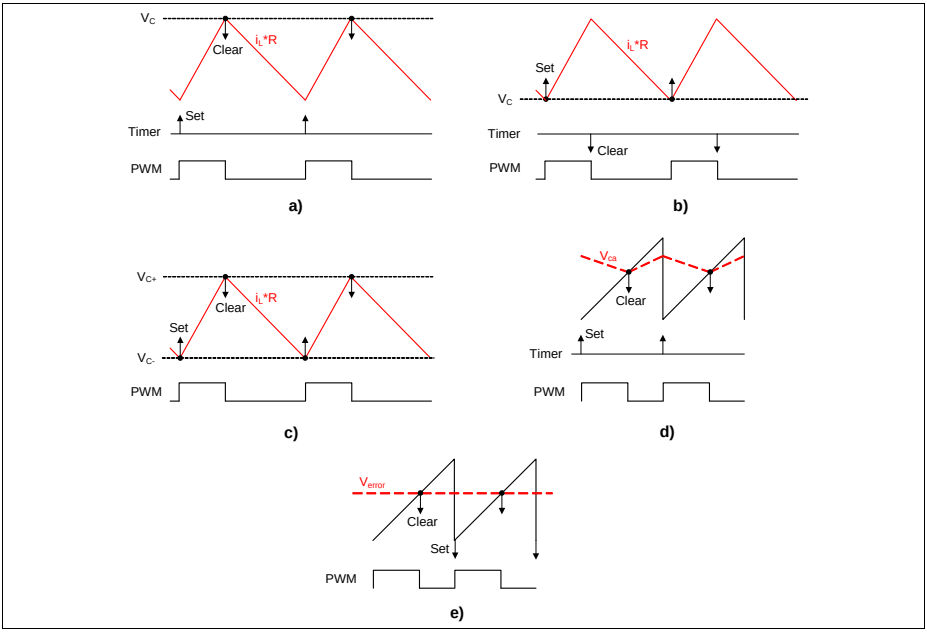


Figure 20-2 Different control techniques: a) peak current control mode, b) valley current control mode, c) hysteretic current control mode, d) average current control mode, e) voltage control mode

The selection of which technique to use, within the Current Control Method, can depend on several aspects that can go from the chosen SMPS topology, operating point of the converter (continuous conduction mode, discontinuous conduction mode or critical conduction mode), voltage conversion ratio, etc.

Reaching to the digital control of SMPS, by combining the HRPWM module with a powerful microcontroller system, is then a huge advantage for developing highly optimized power conversion systems. Systems that can be heavily immune to process and environmental variations, that contain fewer part count and that can be optimized via a tailored software control.

20.1.1 Features

The HRPWM is built of 3 CSG units, being each unit comprised of one High Speed Comparator, a dedicated 10 bit 30 MS/s DAC and one hardware controlled Slope Compensation module.

The HRPWM unit also upgrades up to 4 compare channels of a Capture/Compare unit, with the possibility of generating PWM signals with a maximum resolution of 150 ps. This function is especially useful for SMPS applications operating at a high switching frequency.

Table 20-2 PWM resolution with HRPWM

PWM frequency (kHz)	Normal resolution (bits)[1)]	High resolution (bits)[2)]
50	11.2	17.0
100	10.2	16.0
150	9.6	15.4
200	9.2	15.0
250	8.9	14.7
500	7.9	13.7
1000	6.9	12.7
2000	5.9	11.7
4000	4.9	10.7

Note:

10. 1) Assuming a 120 MHz clock for PWM generation unit
11. 2) Assuming the best case condition of 150 ps

A CSGy instance can be used independently or together with a Capture/Compare unit Timer Slice. The structure and the built-in link of each CSGy instance with a Timer Slice, is especially tailored for DC/DC applications using a current control loop.

The 30 MS/s 10 bit DAC and slope generation modules present inside each CSGy instance, enable a complete HW control mechanism for applications that need a slope compensation scheme to avoid sub harmonic oscillations, like peak current control. For

better control of the slope generation itself, a clock prescaler and a clock pulse swallower are also present.

Each DAC contains 2 programmable reference values that can be used in different scenarios: two value hysteretic control, slope compensation for peak current control, critical conduction mode, etc.

The combination of several CSGy instances, makes the control of a n-phase DC/DC converter a very simple procedure.

Each HRPWM contains a dedicated interrupt generation block, which makes it possible for the SW to monitor the state of each control loop.

General Features

- 3 High Speed Comparators, that can be use to compare an external signal against the DAC value
- 3 (30MS/s) 10 bit DAC
- 3 Slope generation blocks, that are used to generate the DAC input value
- up to 4 High Resolution channels (shared channels of a Capture/Compare Unit)
- different slope generations schemes, for a flexible and automated DAC voltage generation
- 2 DAC reference values, to allow flexible hysteretic mode control
- four dedicated service request lines per HRPWM
- Input multiplexer for the comparator input, allowing several analog inputs to be connected to each comparator and also dynamic input switching
- several power modes
  - high speed comparator mode
  - low speed comparator mode
  - power off

Additional features

- blanking compare mode, to avoid premature switch OFF due to noise
- a dedicated output per Comparator
- programmable clock prescaler
- programmable clock pulse swallower for slope linearization with uneven clock scale
- different slope generation schemes:
  - incrementing mode
  - decrementing mode
  - triangular mode
- shadow transfer for one DAC reference value
- external trigger for the DAC
- external control for the DAC reference values

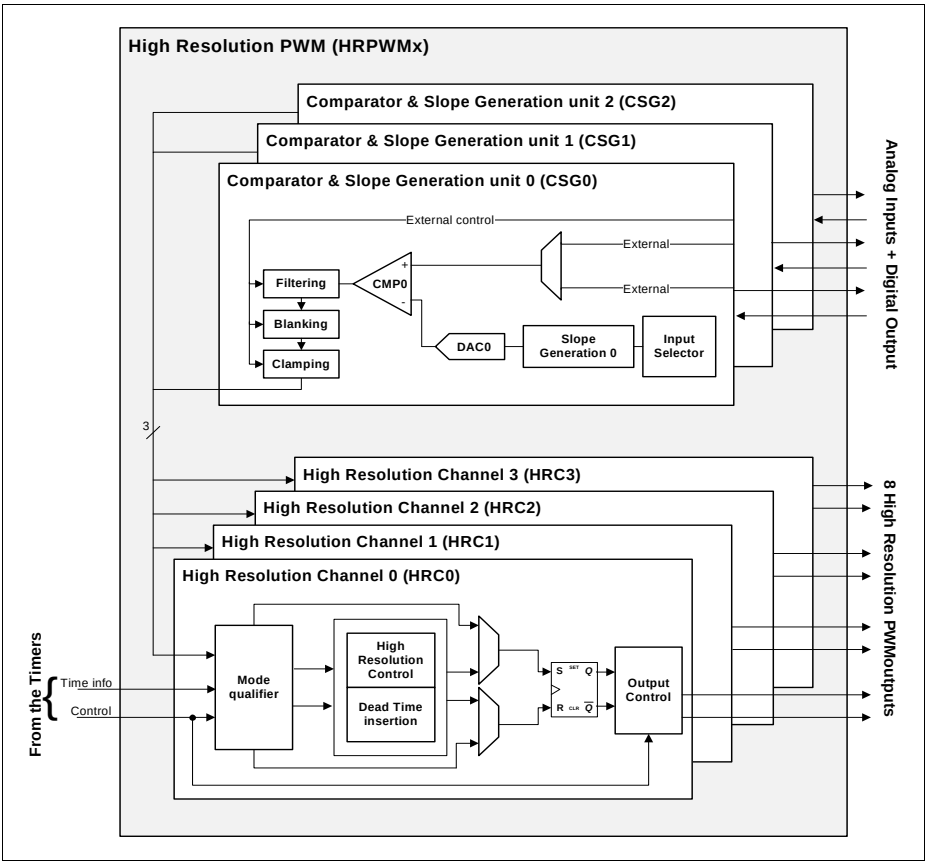


Figure 20-3 HRPWMx simplified function overview

20.1.1.1 Features vs. Applications

Table 20-3 lists the summary of the major features of the HRPWM unit, mapped with the most common applications.

Table 20-3 Applications summary

Feature	Applications
Dedicated comparators	Straightforward signal monitoring: • Current monitoring • Voltage monitoring • Reduce the number of external comparators • Fast response time vs. an ADC channel
Dedicated DACs with Slope generation	Peak Current mode control for DC/DC: • no need for external slope generator • can be linked with a timer to achieve a very accurate fixed frequency scheme
n Comparators + n DACs	Automated n-phase converter control: • 2 phase DC/DC • 3 phase DC/DC • interleaved PFC
Two dedicated reference values for the DAC	Automated control for several loop schemes: • Hysteretic current control • Peak current control • Valley current control • Voltage control
Different slope generation schemes + two reference DAC values	Straightforward built-in control for: • Critical conduction mode • Continuous conduction mode • Discontinuous conduction mode

Table 20-3 Applications summary (cont'd)

Feature	Applications
High Resolution Channels	Especial developed for: • Control up to 4 independent high switching frequency DC/DC • Control up to 3-phase Buck/Boost converter with high switching frequency per phase • Phase shift full bridges topologies (signal shift is possible) • Decreasing the output voltage ripple due to high PWM accuracy
Linking with Capture/Compare Unit	Control of different DC/DC topologies: • Buck/Boost • Half Bridge • Full Bridge • Interleaved or Phase Shift Full Bridge • LLC • Flyback

20.1.2 Block Diagram

Each HRPWM is comprised of three CSG (Comparator and Slope Generation) modules and the high resolution channels extension, HRC.

Inside each CSG module, one has a dedicated high speed comparator, a 30 MS/s 10 bit DAC and the correspondent digital control unit of the DAC. This digital control unit can be used to generate a slope at the inputs of the DAC.

For each HRPWM there are four programmable interrupts lines. These interrupts lines are not multiplexed with the interrupt lines of the attached Capture/Compare Unit.

The clock distribution and the bus access for the HRPWM unit is embedded inside the specific Capture/Compare Unit kernel, see **Figure 20-4**.

A built in linking of the CSG instances with the High Resolution Channel extension, HRCy, enables a direct control of the PWM output via the Comparator.

The utilization of the HRPWM together a Capture/Compare Unit can be divided into several modes:

- High resolution mode with HRCy controlled via the CSGy and the CC8y
- High resolution mode with HRCy controlled via the CC8y
- Low resolution mode with CC8y controlled via the CSGy
- Low resolution mode with HRCy controlled via the CSGy

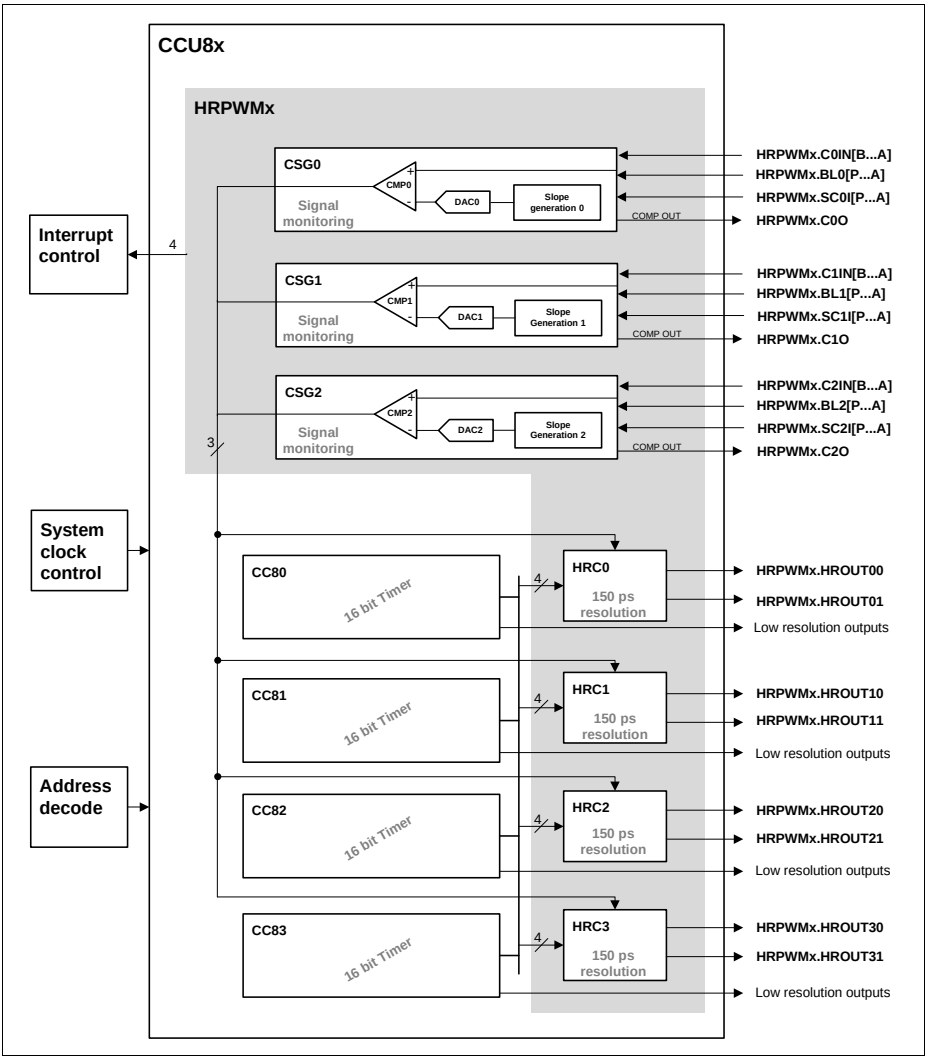


Figure 20-4 CCU8x/CCU4x with HRPWM (block diagram)

20.1.3 HRPWM Applications

Several applications can be targeted with the functions offered by the HRPWM and Capture/Compare Unit system. The following descriptions will help understanding the

amount of resources needed, for different type of control systems within the world of SMPS.

Non Synchronous Buck Converter Control

The non synchronous buck converter is a very common topology throughout several type of applications, **Figure 20-5**.

The peak current control algorithm is well know and most of the times used within this topology, with the objective of avoiding sub harmonic oscillations whenever the duty cycle value exceeds 50%.

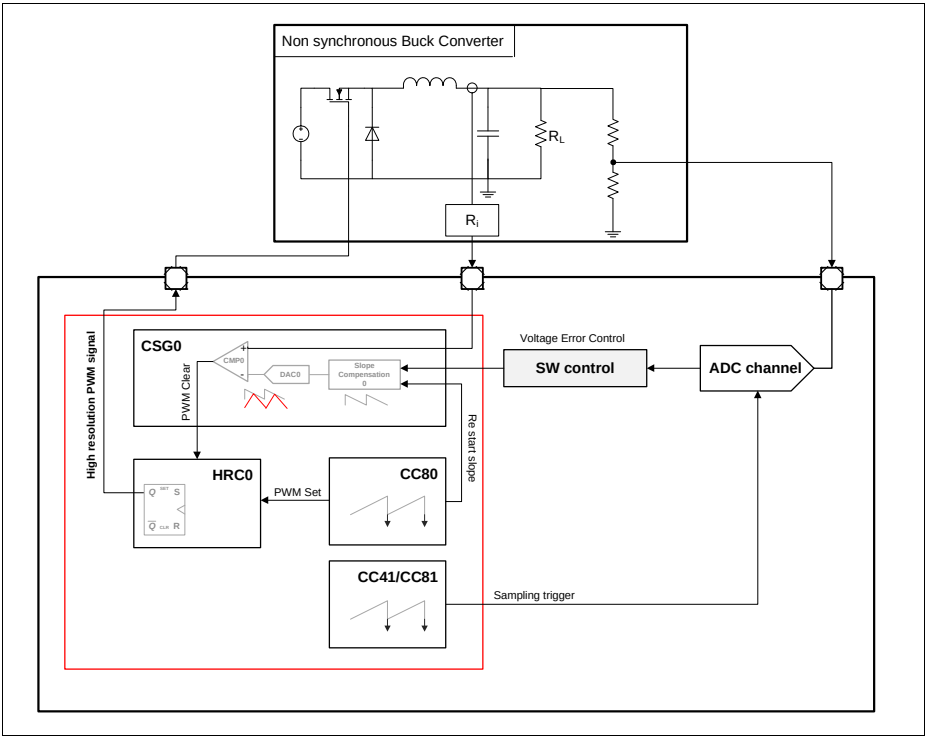


Figure 20-5 Digital peak current control

To implement the control of a non synchronous buck converter, a simple solution can be achieved with the functions available in the Capture/Compare Unit and the HRPWM. Using the HRPWM for this type of control reduces the number of external components and at the same time, high switching frequency systems can be addressed.

For a buck converter using a peak current control feedback, slope compensation is needed when the duty cycle for the PWM signal goes above 50% (this a theoretical value, while a more realistic value should be in the order of 36%), to avoid instability.

The slope compensation can be done inside the HRPWM, by using one CSG unit. The SW loop is monitoring the Voltage Error loop and it can update the reference for the slope compensation on-the-fly.

CC80 is used to control the SET of the PWM signal, while the Comparator inside the CSG unit is used to generate the CLEAR. This is a fixed frequency modulation and therefore CC80 can be a always ON timer, that triggers the restart of the CSG slope each time that the period is reached, **Figure 20-6**.

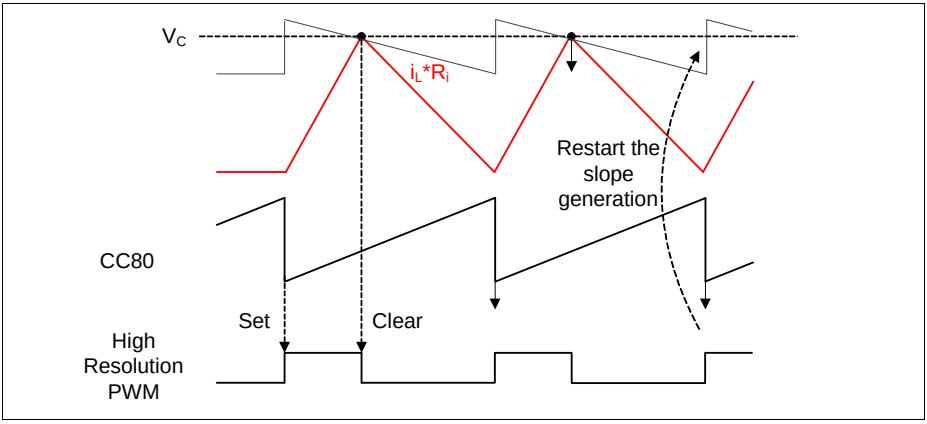


Figure 20-6 Peak current control

By replicating the scheme exemplified, it is possible to control up to 3 independent buck converters, using peak current control.

Multiple Half-Bridge Converters Control

The half bridge converter is well known inside the SMPS market due to its suitability for handling high voltage inputs, which is very common in off-line applications.

On **Figure 20-7**, is exemplified the control of a half bridge using a voltage feedback loop. In case that peak current control is needed, a CSG unit can be used to easily monitor the current feedback loop.

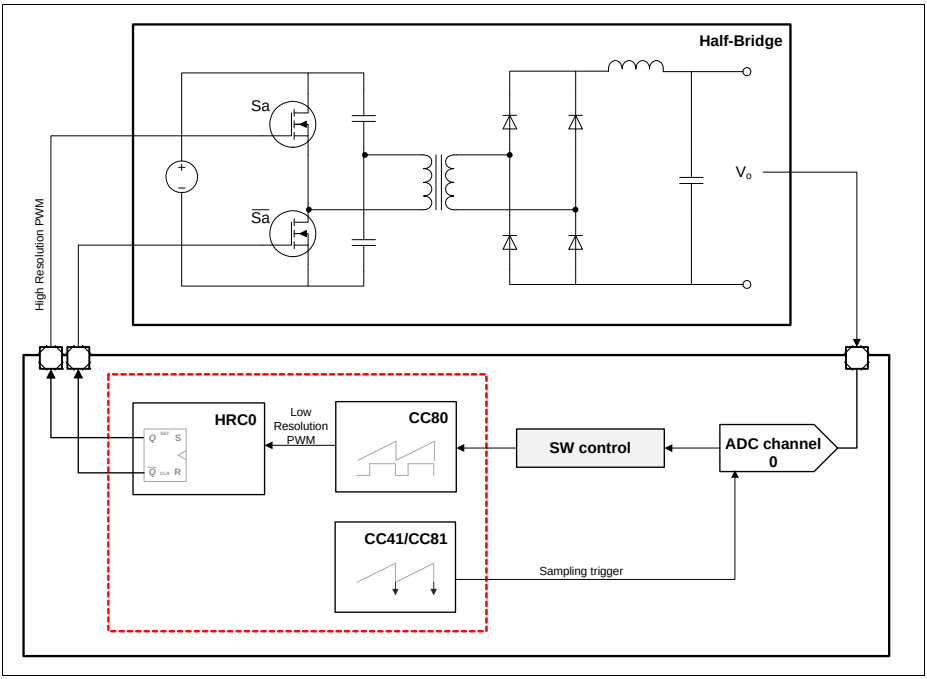


Figure 20-7 Half Bridge control

With the configuration above, it is possible to generate high resolution complementary gate signals to control the half-bridge, **Figure 20-8**. It is also possible to adjust the dead time values on-the-fly to adjust to different load conditions as well programming different dead time values, for the rising and falling transitions.

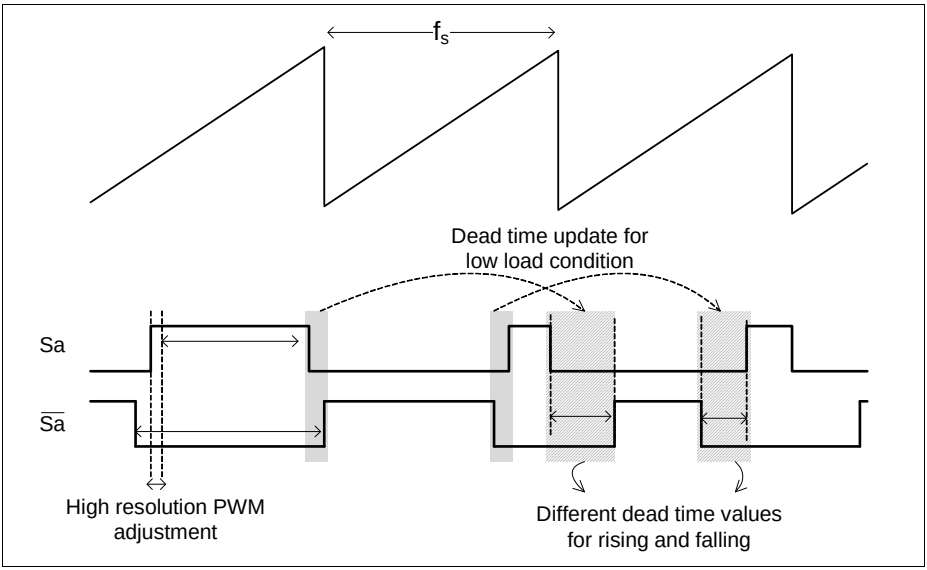


Figure 20-8 Half-bridge high resolution complementary signals

This system can be easily configured to achieve a control of 4 independent Half Bridges, **Figure 20-9**.

In this scheme timers from a Capture/Compare Unit 8 were used. By using this type of timers, one has available 2 Compare Channels per timer instance. With this solution, one compare channel can be used to control the output PWM signal, while the second compare channel can be used to trigger and ADC conversion in a cycle-by-cycle mechanism.

As a pure example, two ADC channels were used. Each channel performing the ADC conversions of 2 Half Bridges.

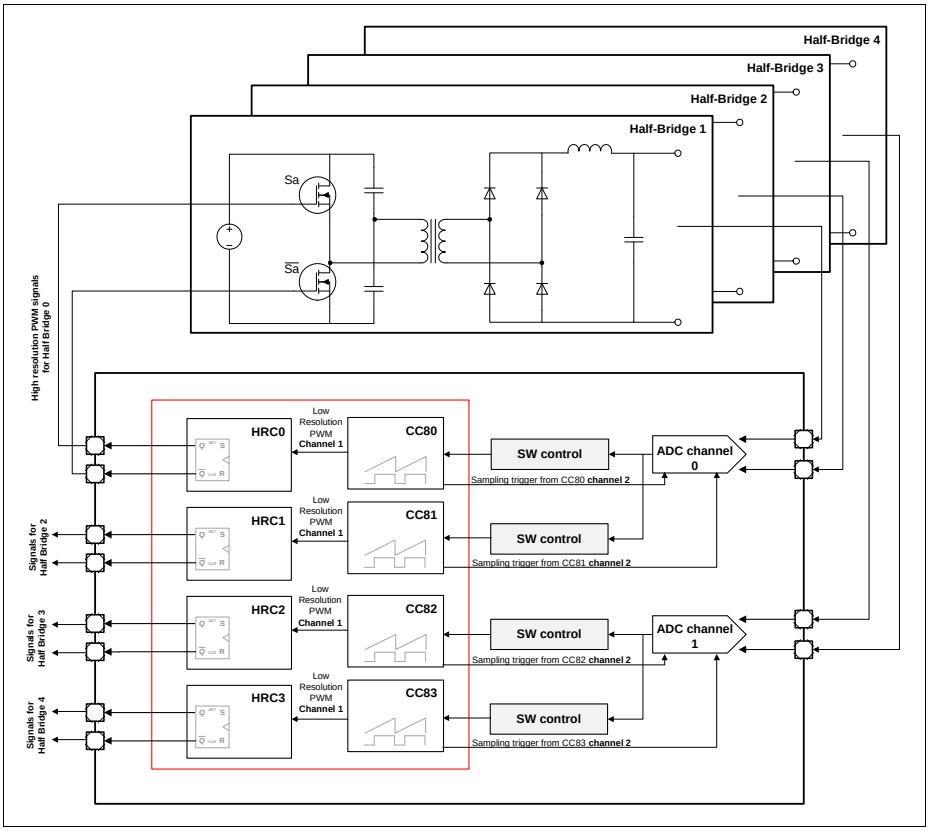


Figure 20-9 Control of 4 independent Half Bridges

Single Phase DC/DC with fail-safe control

It is possible with a simple arrangement of resources, to have a dedicated timer being used for maximum ON time control. With this mechanism it is possible to limit the maximum time where the switch is ON (overload situations due to wrong measurement), avoiding a premature burn of the switch.

This feature is possible due to the fact that each High Resolution Channel (HRC), can decode two pairs of SET & CLEAR signals, **Figure 20-10**. One pair can be used within the normal operating conditions, while the other pair is used to override and clear the PWM output to an inactive state.

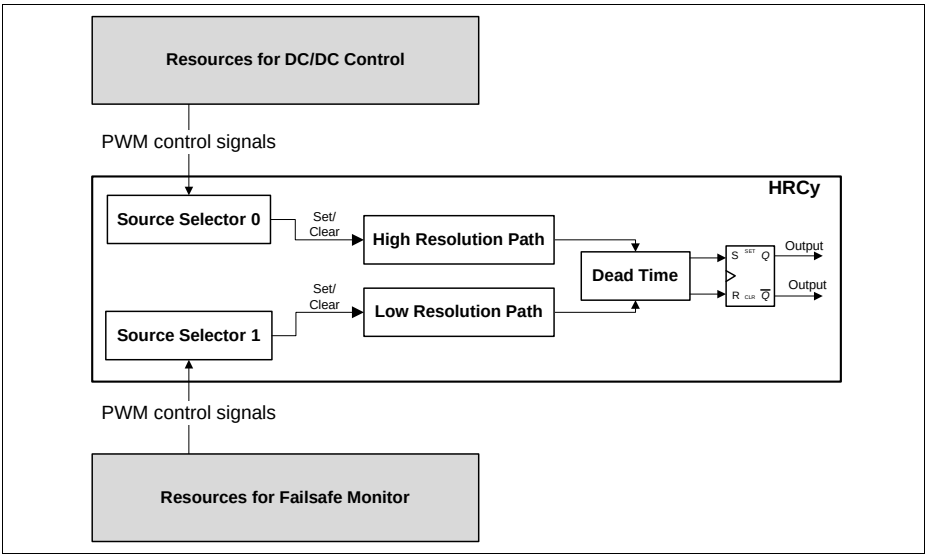


Figure 20-10 HRC dual Set/Clear decode

As an example **Figure 20-11** shows the control of a synchronous buck converter with this function. The buck converter loop implements a peak current control, being the Comparator and Slope generator inside the unit CSG0 used for this purpose. CC80 is used to maintain the fixed frequency modulation and the HRC0 unit, is generating the high resolution PWM signals.

CC81 is used as fail-safe timer while CC42/CC82 is used to trigger the ADC conversion of the output voltage.

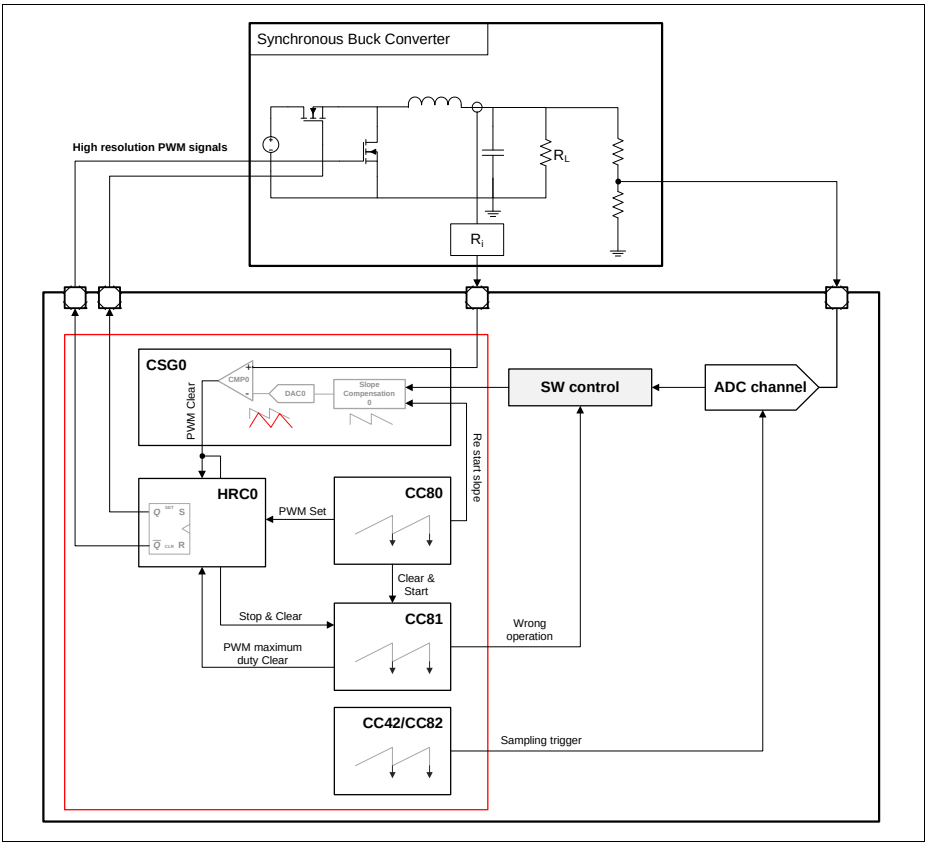


Figure 20-11 Synchronous buck converter with fail-safe timer

Figure 20-12 shows the timing operations between the several resources. CC81 is being started every time that the PWM signal is turned ON and consequently is cleared and stopped when the PWM is OFF. The moment that CC81 reaches the programmed maximum ON time it generates a CLEAR signal for the HRC0, overriding the output from ON to OFF.

It is also possible to use CC81 to cancel the operation of CC80 avoiding the next SET dictated by the fixed frequency modulation timer. To do this, the same signal that triggers the ISR can stop the CC80.

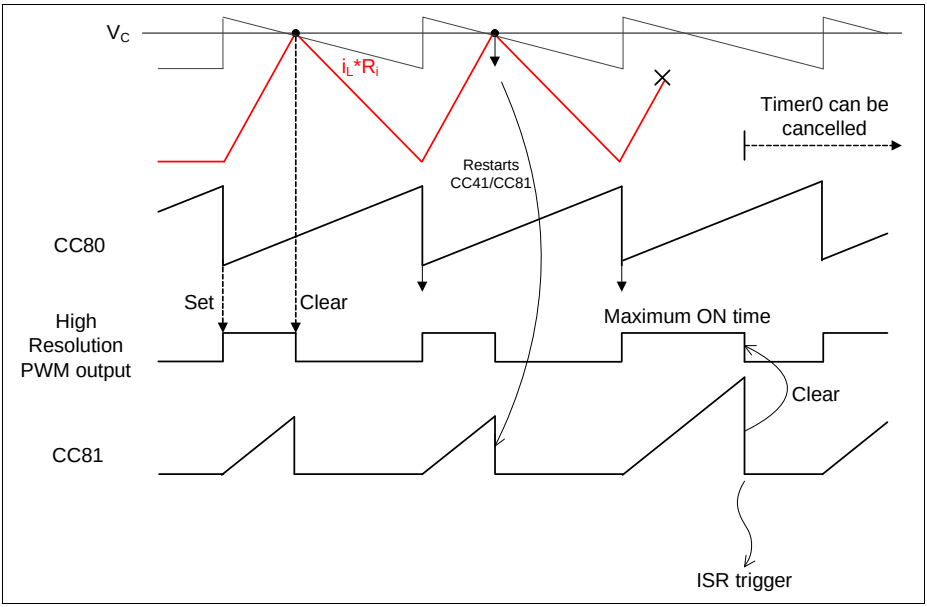


Figure 20-12 Maximum PWM ON time control

Phase Shift Full Bridge Control

Full bridge converters are commonly used in the SMPS world for applications that need a high output power level. In these type of topologies it is very important to employ a soft switching technique to reduce the switching losses.

A soft switching operation (or commonly known as Zero Voltage Switching - ZVS) can be achieved by using a modulation technique with fixed frequency and phase shift.

This topology can be easily controlled via the Capture/Compare Unit 8 and the HRPWM, **Figure 20-13**. Two timers are needed to control the two legs of the full bridge, CC80 and CC81. The second channel of CC81 can be used to trigger the ADC conversion and in the case that an A/D conversion is not needed in every PWM period, a third timer, CC82 can be used.

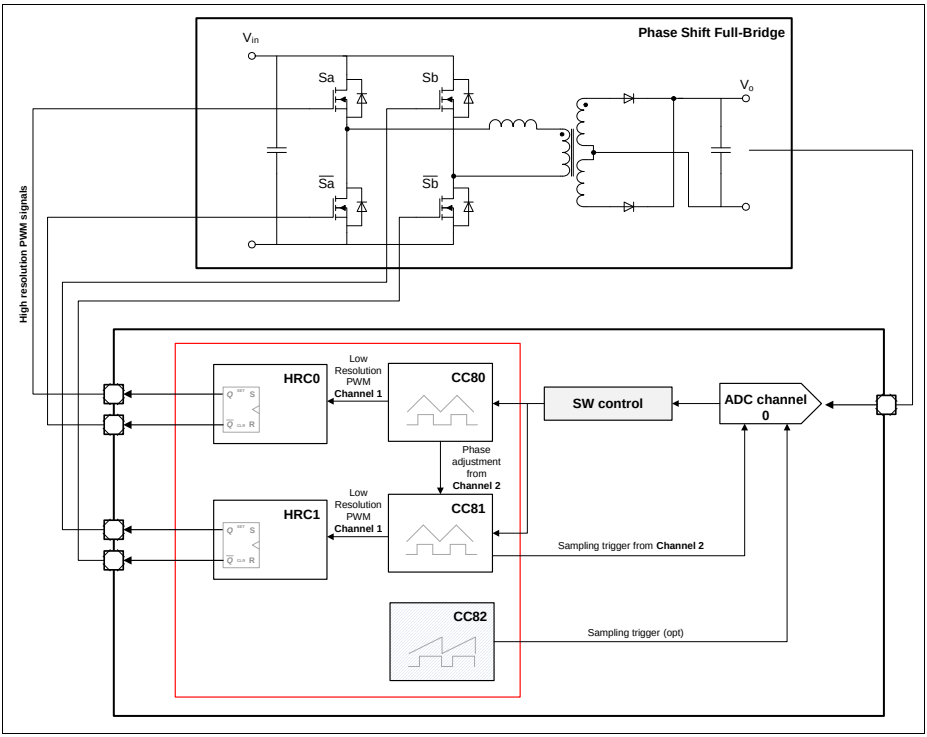


Figure 20-13 Phase Shift Full Bridge

CC80 is used as the leading timer, which means that the phase shift reference is the point where this timer starts. The second compare channel of CC80 is used to start CC81 (that is operating in single shot mode), controlling this way the phase difference between the two legs, **Figure 20-14**.

The coarse phase difference is controlled via the CC80, while a high resolution phase adjustment (150 ps) can be done via the HRC1 unit.

The phase difference phi, illustrated on **Figure 20-14**, controls the amount of energy applied to the inductor, hence the resulting DC output voltage.

With the flexible cascaded shadow transfer offered by the Capture/Compare Unit 8, it is also possible to perform all the adjustments for the phase shift on the fly, with just one software routine.

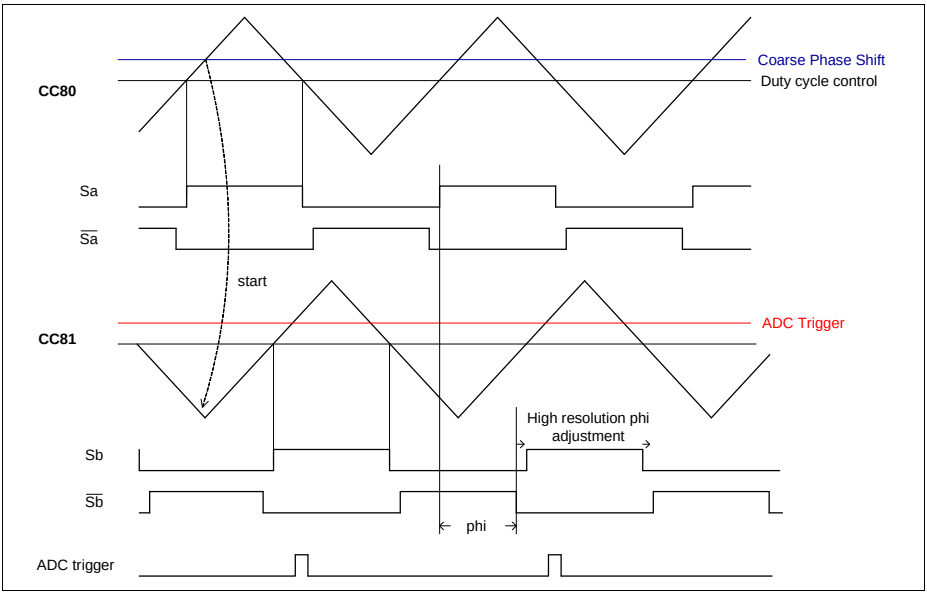


Figure 20-14 Phase Shift Full Bridge timings

3 Phase Buck Converter Control

Multiple phase buck converters can cover the trend that leads into lower supply voltages and bigger load current requirements. In a multiple phase converter, the output ripple can be substantially reduced due to the increase of the overall switching frequency. Increasing the switching frequency also reduces the size of the filter inductor, reducing this way the overall cost of the converter itself.

A 3 phase synchronous buck converter, can be controlled with just one HRPWM and a Capture/Compare Unit 8.

If a voltage feedback loop is implemented, then the control of a 3 phase buck converter can be implemented as demonstrated in **Figure 20-15**.

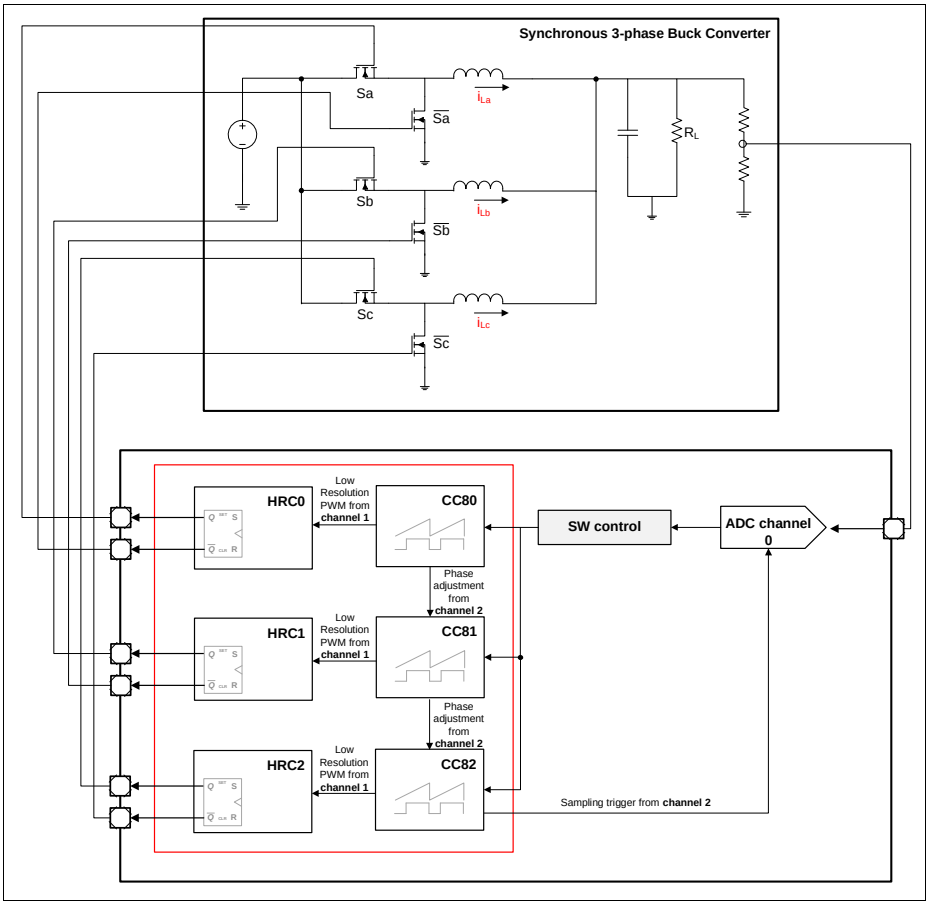


Figure 20-15 3 phase buck converter - voltage control

In the case that a current control method is implemented, then the resources available inside the CSG units, can be used to perform the cycle-by-cycle monitoring of the currents. As an example, a peak current control feedback loop is demonstrated in **Figure 20-16**.

The built in HW functions for slope generation inside each CSG unit, are especially developed for cycle-by-cycle current monitoring. By using this type of resources, the hardware developer can simplify the process of controlling a n-phase DC/DC converter.

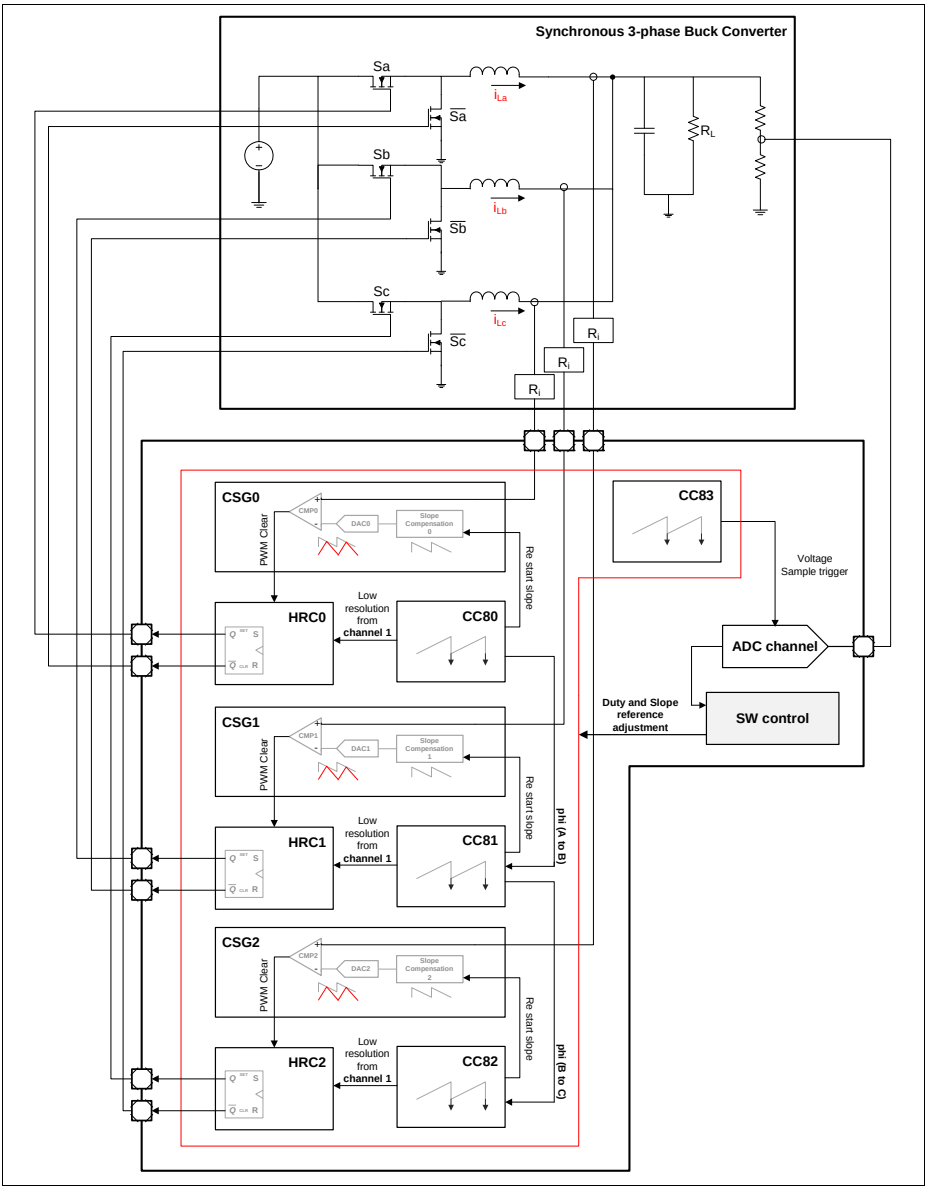


Figure 20-16 3-Phase buck converter - peak current control

The second compare channel of each Capture/Compare Unit 8 timer is used to control the phase difference between each buck converter leg. The value programmed into the second compare field is then used to trigger the start of the immediately next timer, **Figure 20-17**.

CC83 of Capture/Compare Unit 8 is used to control the trigger for the sampling of the output converter voltage. This timer can be programmed accordingly with the needs of the voltage output sampling process:

- one measurement per switching cycle
- n measurements by switching cycle
- one measurement in every n switching cycles

In case that a 2-phase buck converter implementation is needed, two adjustments need to be performed to this set up:

- one timer is removed, because only 2 PWM signals are needed
- the phase difference needs to be adjusted to 180° (instead of the 120° of a 3 phase converter)

Additionally the cascaded shadow transfer function of the Capture/Compare Unit 8 and HRPWM makes the update on-the-fly of the parameters easy and accurate to each switching period.

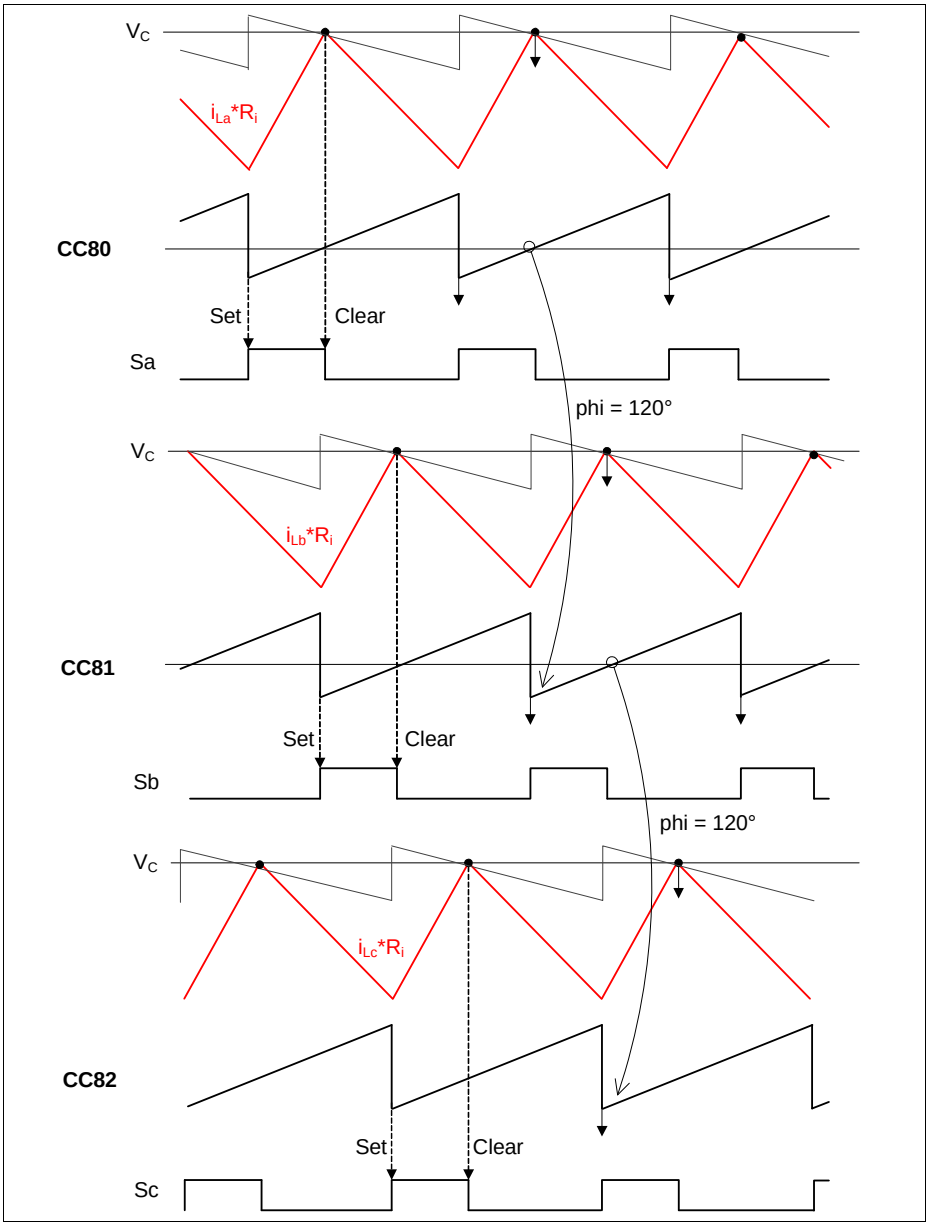


Figure 20-17 3-Phase buck converter control signals

Multi-Mode DC/DC Control

Modern front end high frequency DC/DC converters normally need to operate over dynamic load conditions, from high load up to low load operation points. For this type of applications having a multi-mode control method approach, can increase greatly the efficiency of the converter throughout the entire operating range.

One example is the usage of peak current mode control when comes to low load conditions. Due to the fact that the inductor current is sensed across the Mosfet (or High-Side Mosfet in a Half Bridge configuration), during the turn ON the node will exhibit a lot of noise. This implies that a blanking time after turn ON is implemented to avoid premature shut down.

This blanking period normally dictates the smaller value for the ON time (that can be in the order of 200/300 ns), imposing directly a limitation when the application has a small load, **Figure 20-18**.

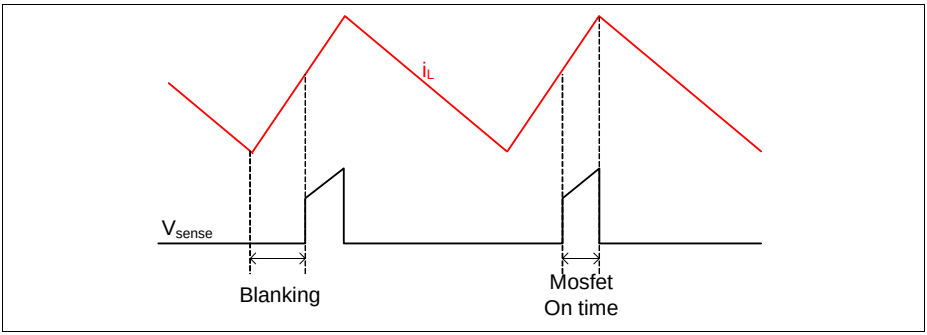


Figure 20-18 Blanking period in peak current mode

Several solutions can be implemented to overcome the different operating ranges (normal load versus low load conditions). These techniques can englobe different control modes like pulse skipping, pulse grouping or some proprietary solutions.

With the usage of the advanced functions of the HRPWM it is possible to control an automated transition between two operation modes, for a specific SMPS application.

Figure 20-19 shows the basic scheme of the multiple generation scheme for a PWM signal within each HRC unit. With this scheme, it is possible to have a set of resources being used with Operation Mode 1 (let's define it as Normal Load Operation Mode) and a complete different set of resources, to control Operation Mode 2 (let's consider it Low Load Operation Mode). The switch between the set resources is done via a mode switch bit, that can be updated by the SW on-the-fly.

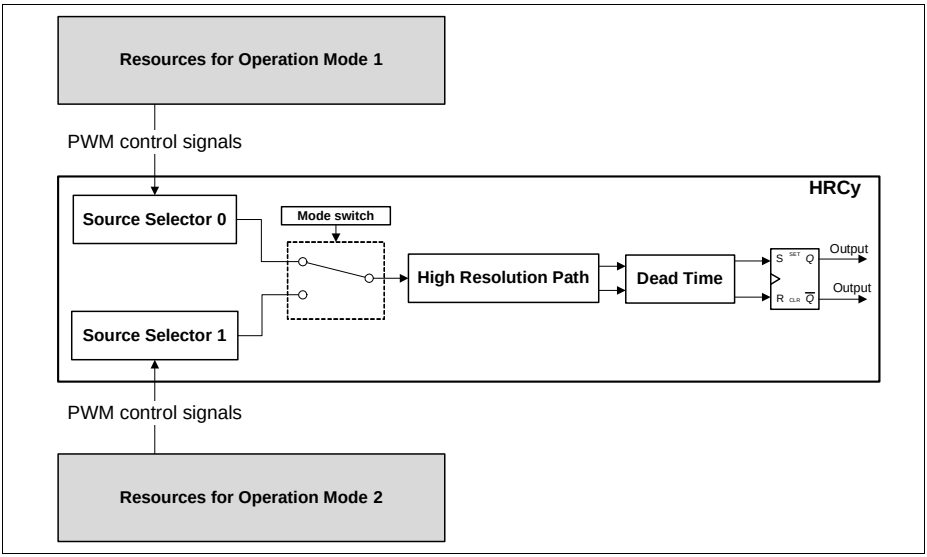


Figure 20-19 HRPWM multiple control scheme

This type of multiple control technique is a major advantage for state of the art DC/DC control. It will enable the usage of different control methods (applied for different operation modes) to increase the converter efficiency or reduce the front-end design costs.

Figure 20-21 exemplifies a resource organization for a Buck Converter using peak current control for Normal Load Conditions and a HW/SW control method for Low Load Conditions (that can be configured to several schemes: pulse skipping, pulse grouping, variable ON/OFF time, variable frequency, etc).

A comparator unit, CSG0 and a timer, CC80, are used to control the converter during normal operation (peak current control). The comparator dictates the CLEAR of the PWM signal while CC80 dictates the SET.

To implement a built-in low load control, an additional timer is used, CC81. This timer is only used when the SW enables the low load condition. And after this point, the SET and CLEAR of the PWM signal is no more dictated by the Comparator and CC80 but via CC81.

This control scheme is possible due to the dual control path available inside each HRC subunit of the HRPWM. The organization of resources for each operating mode can be freely chosen by the user. Therefore, **Figure 20-21** serves as a mere resource organization example.

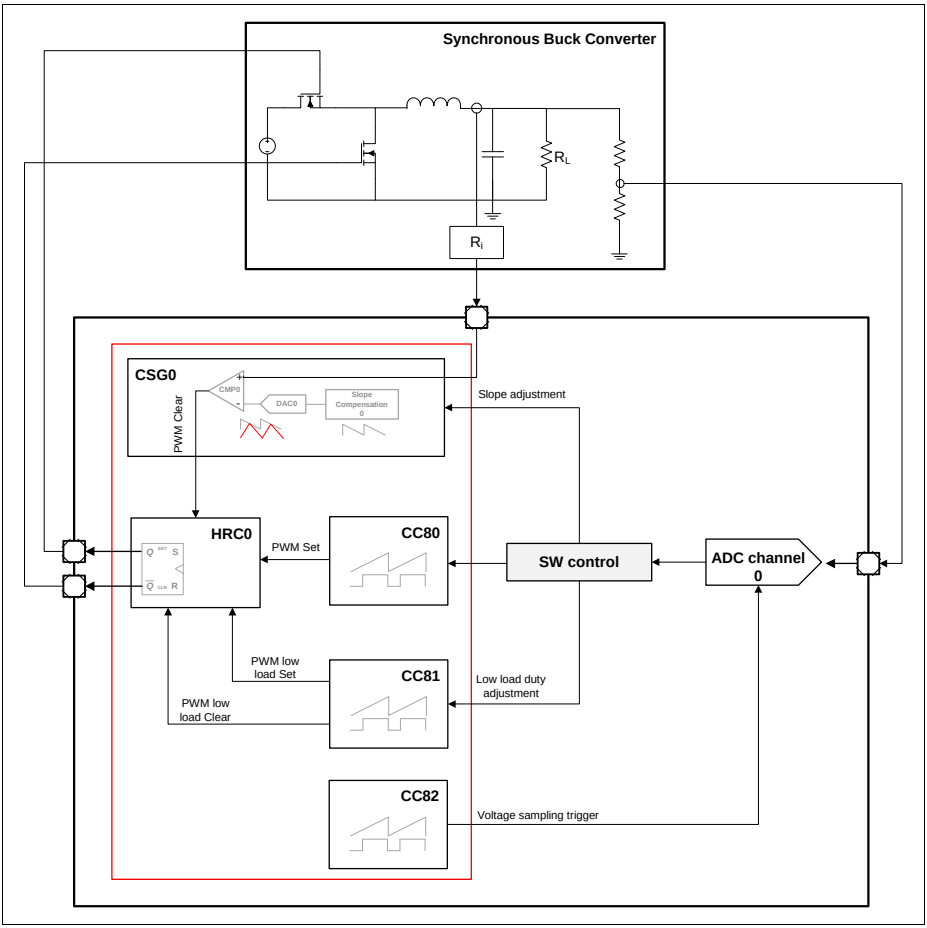


Figure 20-20 Low load ready buck converter

Figure 20-21 shows the transition between Normal Operation Mode and Low Load Mode.

Initially, the compare channel of CC80 is being used to dictate the SET of the PWM signal, while the Comparator is controlling the CLEAR.

When a switch between the modes is needed, the SW updates the mode switch control inside the specific HRC unit.

From this point onwards, it is possible to control the duty cycle via CC81 and with the functions inside the HRPWM, a PWM resolution of 150 ps can be achieved. All of this without re configuring the HW or losing the synchronicity between the two modes.

Note: The Set signals from CC80 and CC81 should distance at least one module clock cycle from each other.

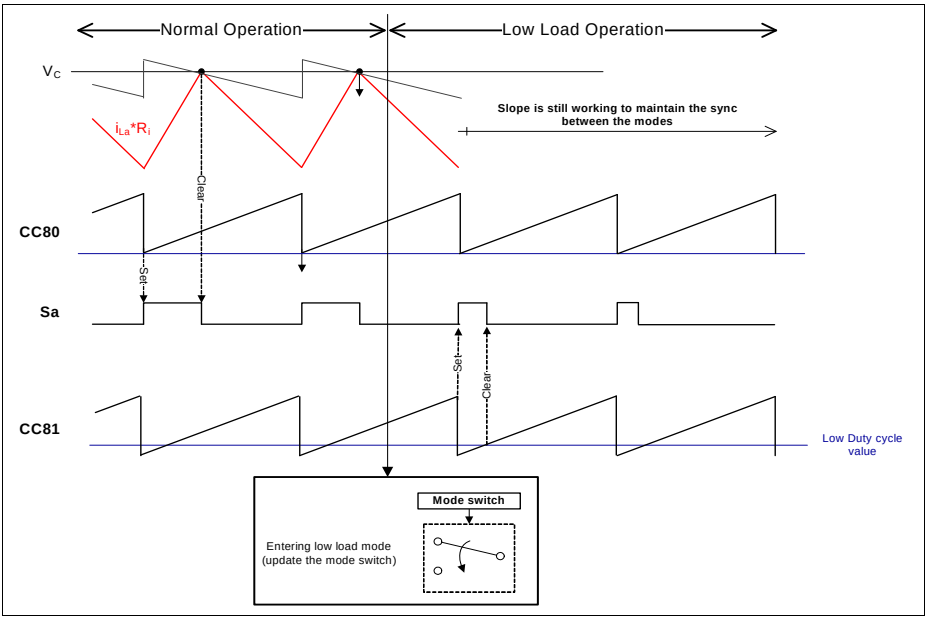


Figure 20-21 Entering low load mode

Within the Low Load Mode several techniques can then be implemented (depending on the resource organization). In **Figure 20-22** several control modes are exemplified.

The first control mode uses the second timer unit, to control a very accurate duty cycle value to maintain the proper output voltage. The duty cycle has a resolution of 150 ps and can be updated via SW, on a cycle-by-cycle manner dependent on the actual operating conditions.

In the second method, is considered that the ON time pulses of the PWM signal are too small to be transmitted to the switch. Therefore n pulses are grouped together and applied over a frequency n times smaller. These pulses can also be controlled cycle-by-cycle via SW with a resolution of 150 ps.

The third method exemplifies the usage of pulse skipping. For this method additional resources are needed:

- additional CSG unit to monitor the trip voltage for pulse skipping

- trip voltage calculation via SW or external Compensator block

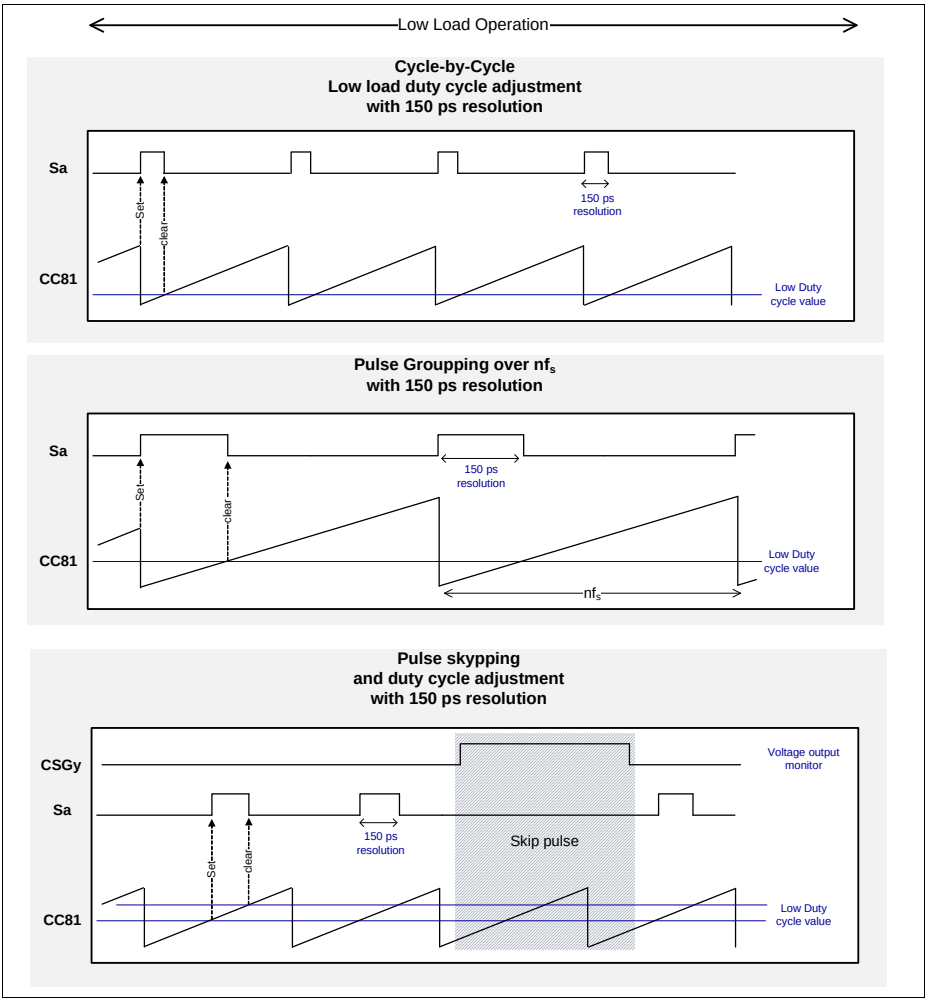


Figure 20-22 Low Load Mode control examples

Figure 20-23 shows the transition between Low Load Mode and Normal Operation Mode.

The synchronization between the modes switching is deeply linked with the type of control loops implemented. **Figure 20-23** description serves as a mere example for a specific scenario.

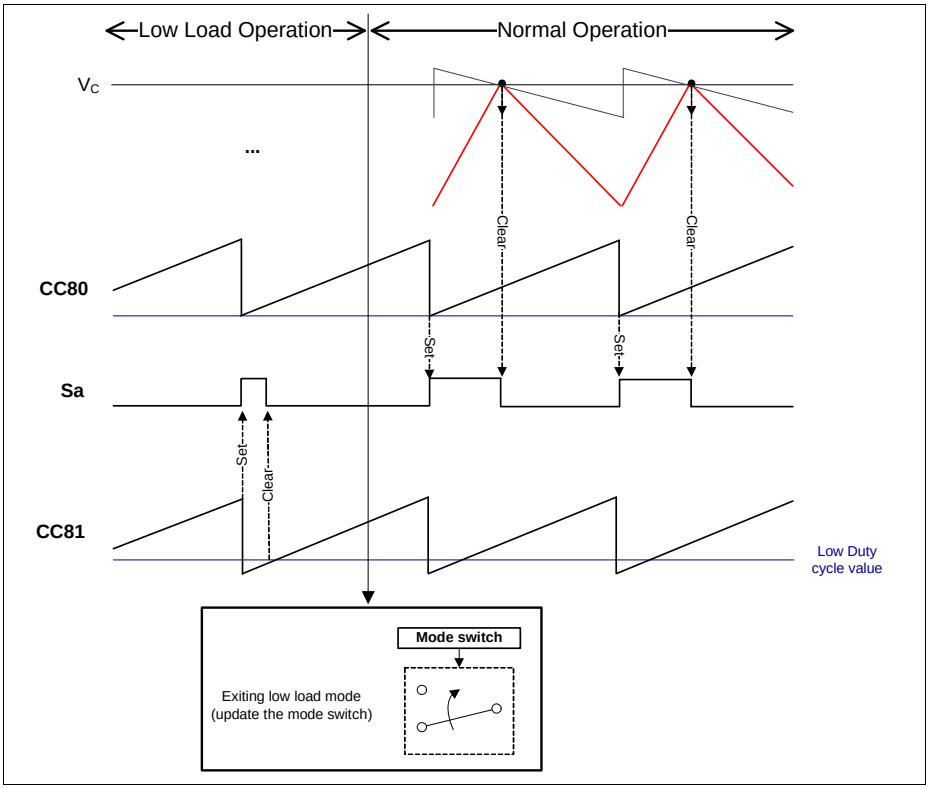


Figure 20-23 Exiting low load mode

Single Stage Critical Conduction Mode

Critical Conduction Mode (CRM) is widely used due to it's simplicity and low cost. This control method is especially used for Power Factor Correction (PFC) applications.

Inside the low power tier applications, up to the range of a few hundred watts, the boost topology is one of the most commonly used to implement PFC. Even for higher power output systems, some advantages can be achieved versus the most commonly used Continuous Conduction Mode (CCM):

- The "harsh" reverse recovery of the output diode and generation of EMI harmonics problem can be solved

- Complexity of the control algorithm for Continuous Conduction Mode can be heavily reduced
- No need for ramp compensation

With the functions available in the HRPWM it is easy to implement a PFC CRM loop, **Figure 20-25**. In this case, T_{ON} is assumed to be constant for a given reference signal ($k.V_{IN}$) while T_{OFF} will change throughout the cycle, imposing a variable frequency scheme.

This voltage mode CRM control scheme, will decrease the number of external components needed: the multiplier that monitors the input rectified ac line voltage; the error amplifier from V_{OUT}.

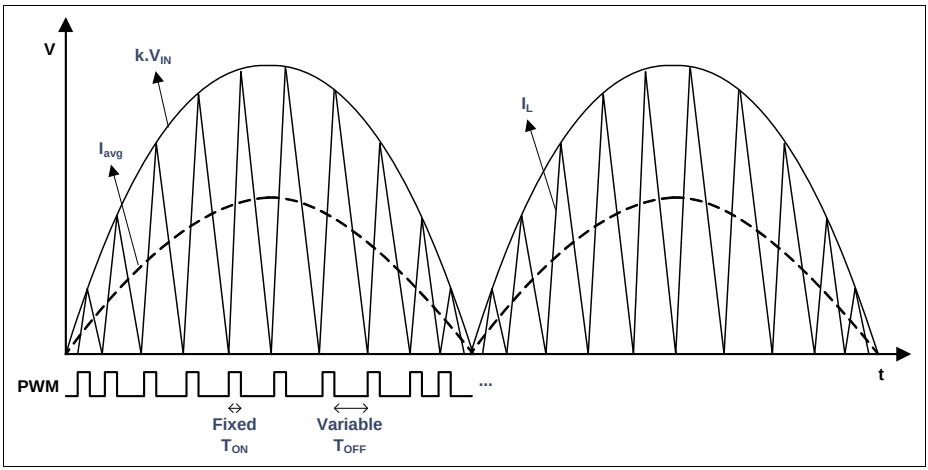


Figure 20-24 CRM PFC waveforms

The main CRM loop can then be built with one CSG, one HRC and one Timer, **Figure 20-25**. The CSG unit is used to generate the SET for the HRC output latch when the Zero Current Crossing is detected. The Timer is used to impose the fixed ON time PWM signal.

The switching waveforms for the PWM signal can be seen in **Figure 20-26**. The CSG0 is monitoring the threshold of the Zero Current Crossing. Every time that this threshold is reached, the PWM signal of the HRC0 unit is set HIGH. At the same time, the CC80 timer is started and when the ON period is over, the PWM signal is cleared.

In **Figure 20-26**, is also demonstrated that the switch ON signal sensing may have some noise. To avoid to turn ON the switch while the voltage is high (due to noise), which imposes losses on the converter, the filtering stage inside the CSG0 unit can be used.

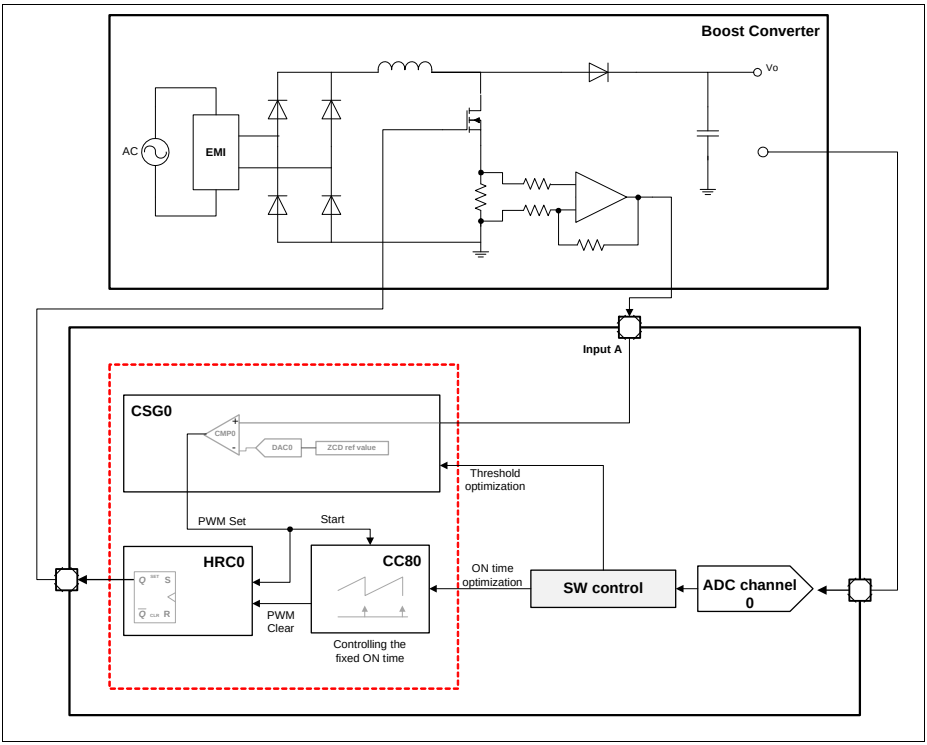


Figure 20-25 Critical Conduction Mode - Boost Converter

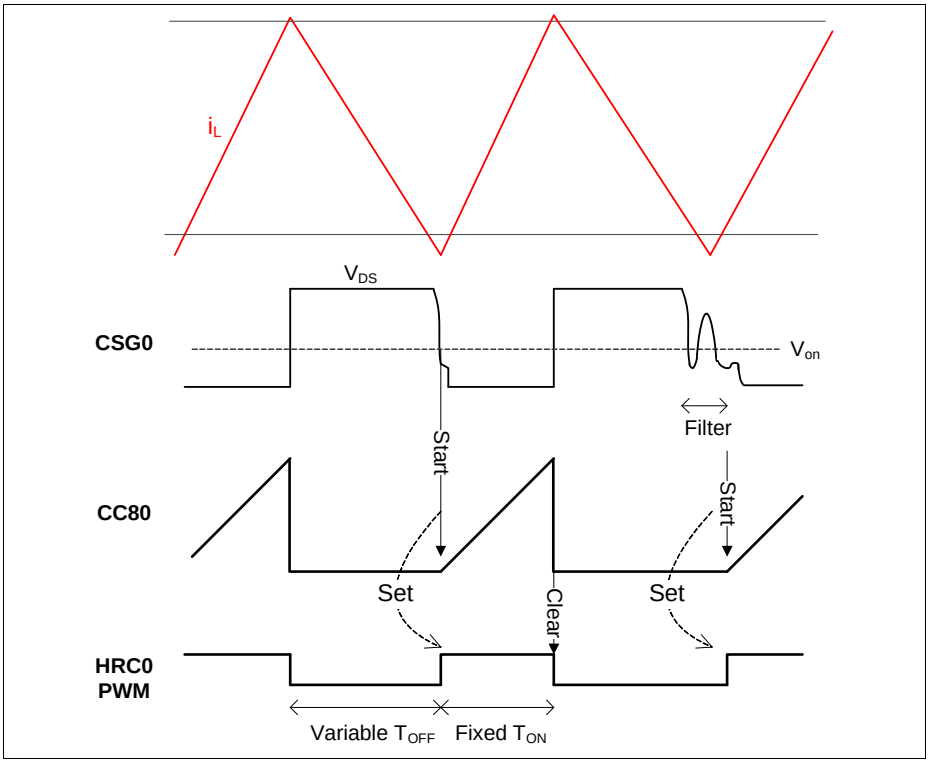


Figure 20-26 CRM switching waveforms

Hysteretic Stage Critical Conduction Mode

Additional control schemes or add-ons can be implemented to control a CRM loop. One straightforward modification to the CRM loop, is to use two comparison thresholds, one for the Zero Current Crossing and another for the Peak Current sensing (hysteretic CRM mode with variable frequency and variable T_{ON}).

A possible arrangement of resources is exemplified in **Figure 20-27**. The CSG is monitoring the two thresholds (turn ON and turn OFF) and the CC80 timer is used to pass the frequency and duty cycle information to the SW. As an add-on, the CC81 timer can be used to impose a minimum OFF time to the loop (forcing the converter to enter in Discontinuous Conduction Mode whenever the OFF time is too small).

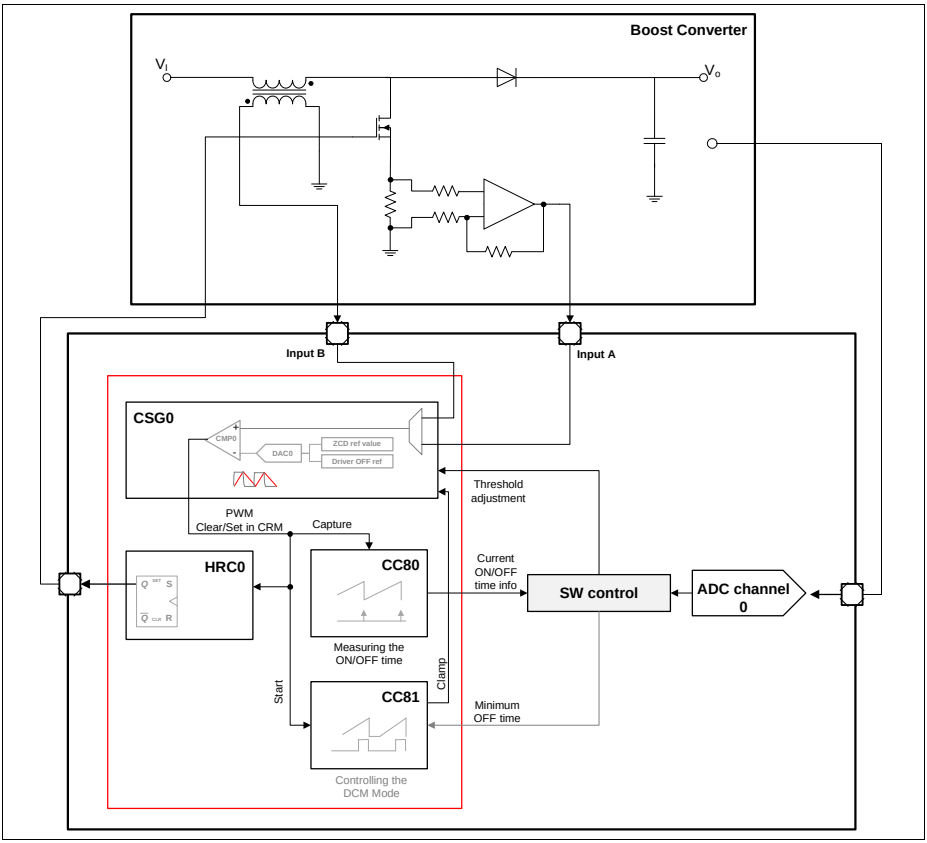


Figure 20-27 Hysteretic CRM resources

To control the hysteretic CRM, the function inside the CSG unit that allows the switch between the two analog inputs (one sensing the Zero Current Crossing and the other sensing the threshold for turning OFF the switch) must be used, **Figure 20-28**.

The CSG unit has a function that enables the switch between the two analog inputs, which makes possible to sense the OFF and the ON threshold. The 10 bit DAC also has two reference values that can be used for this mode.

In **Figure 20-28** two thresholds are being used to control the converter in the hysteretic CRM: one threshold for turning ON the switch, V_{on}, and another to turn OFF, V_{off}. These two values are then programmed into the two reference DAC values.

While the switch is conducting, the comparator input is connected to **Input A**. The moment that the threshold for switching OFF is detected, the PWM signal is deasserted

and the DAC reference value is switched to the V_{on} value. At the same time, the comparator input switches to **Input B**. When the threshold for switching ON is detected, the PWM is asserted and the Comparator input switches to **Input A** again.

All these operations are done automatically via HW.

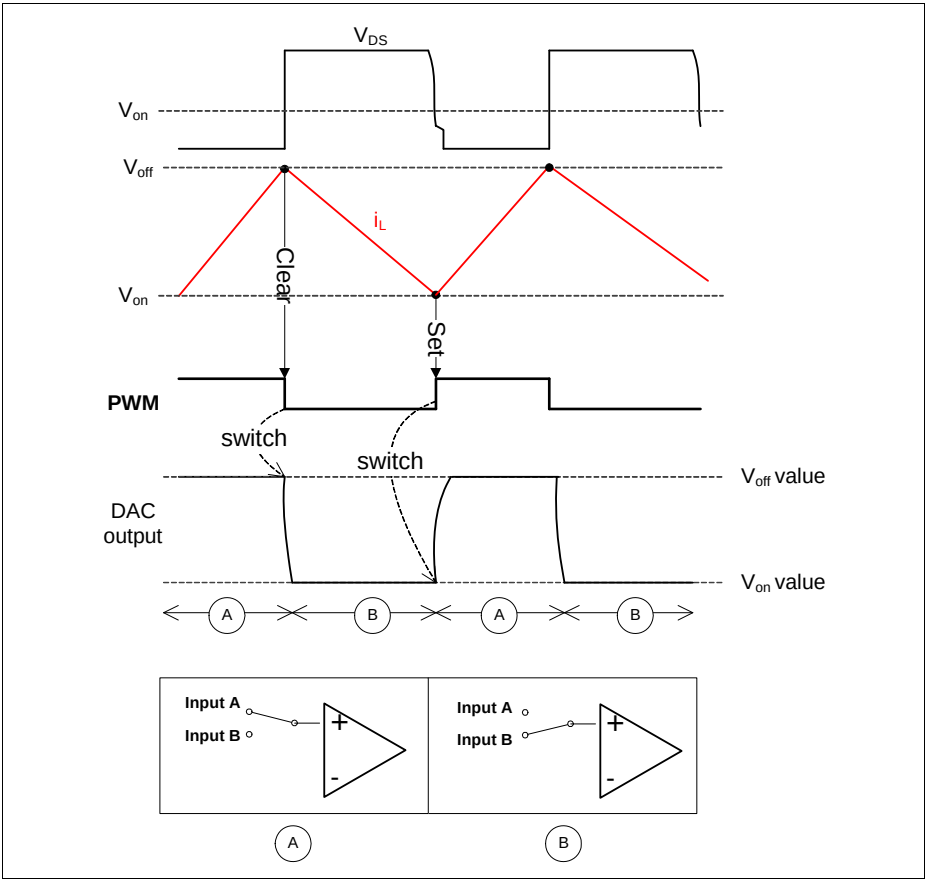


Figure 20-28 Critical Conduction Mode operation

Two additional timers, CC80 and CC81, are used to upgrade the CRM control with:

- Cycle-by-cycle monitoring of the ON time/frequency of the PWM signal for control loop readjustment - CC80
- Mode switch between Critical Conduction Code and Discontinuous Conduction Mode with fixed OFF time - CC81

The switching between Critical Conduction Mode (CRM) and Discontinuous Conduction Mode (DCM) can be implemented easily with the CC81 timer. By using CRM and DCM, one can have the following advantages:

- simple front-end EMI filter design due to the limit imposed by the DCM on the switching frequency
- limit the maximum currents on the diode MOSFET and inductor when in CRM

The CC81 timer is programmed into single shot mode and the trigger to start counting is a service request from the CSG unit. This trigger is generated every time that a transition from the V_{off} value to the V_{on} value is done by the CSG unit, **Figure 20-29**.

The value programmed into the CC81 Timer will then dictate the minimum OFF time wanted for the loop.

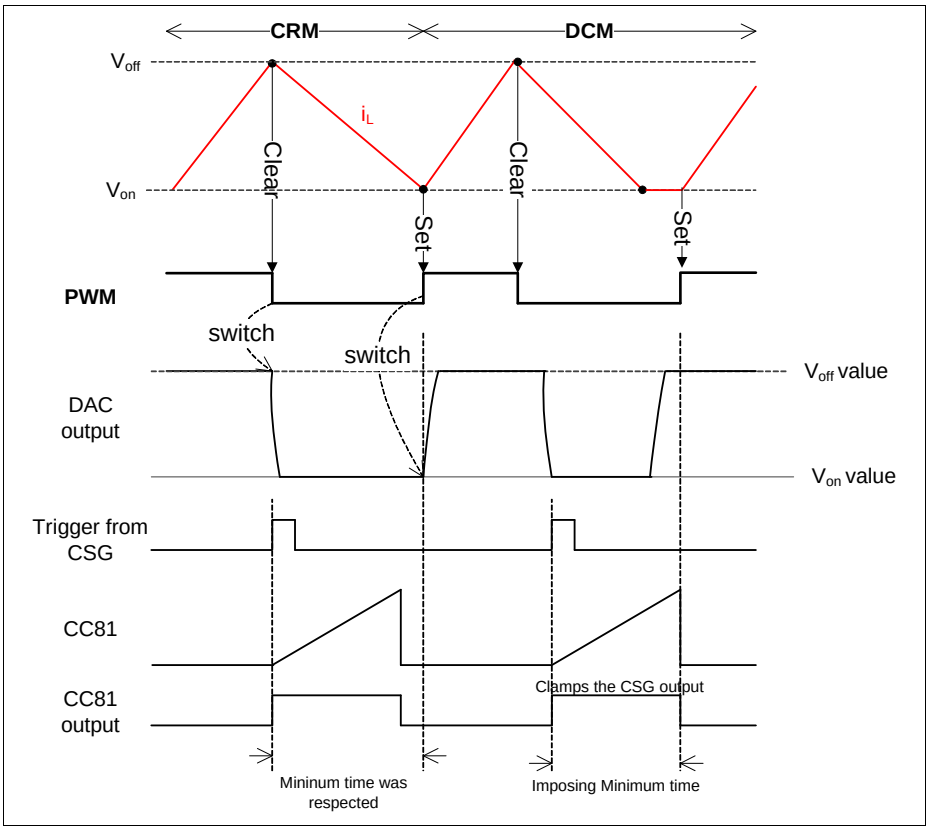


Figure 20-29 Transition from CRM to DCM

Migration to a Peak Current Control CRM is also available inside each CSG unit. For that, one just needs to enable the slope generation in hybrid mode, **Figure 20-30**.

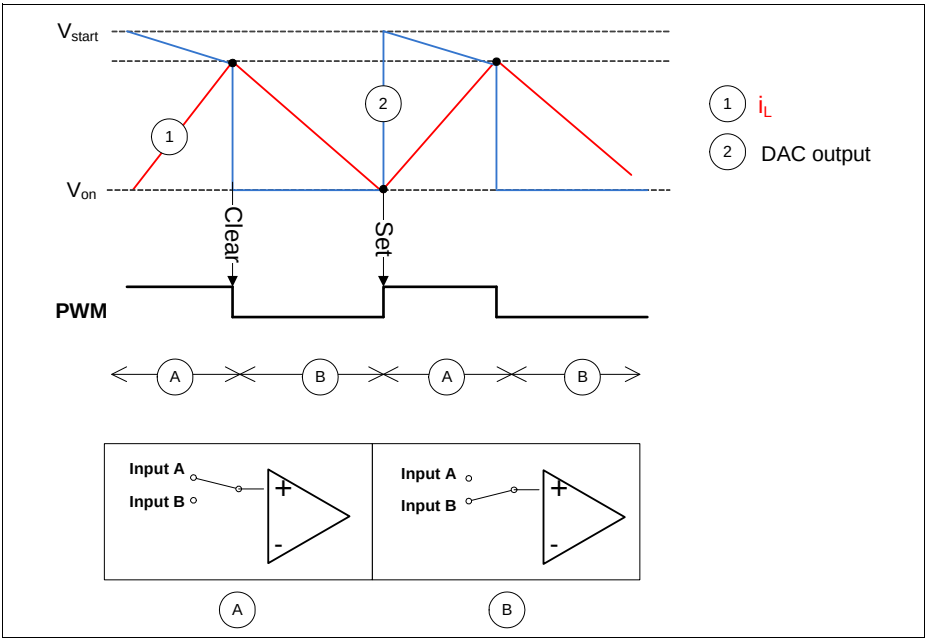


Figure 20-30 Peak Current Control with hysteretic CRM

LLC DC/DC Control

The LLC resonant converter is widely known in the world of DC/DC topologies. It carries several benefits that are most of the times needed for power conversion applications, such as:

- Zero Voltage Switching (ZVS) can be achieved over the entire operating range
- Smaller switching loss due to small turn off current
- Small stress voltage on the rectifiers

Controlling a LLC converter is a simple task with the HRPWM. A Comparator and Slope Generation unit, CSG, and one High Resolution channel, HRC, are sufficient to generate the proper PWM signal, **Figure 20-31**.

Additional resources may be needed in the case that the output voltage and the current duty cycle need to be monitored to perform threshold adjustments (to maintain the DC/DC efficiency):

- Monitoring the duty cycle - CC80

- Monitoring the output voltage - CC81 + ADC channel 0

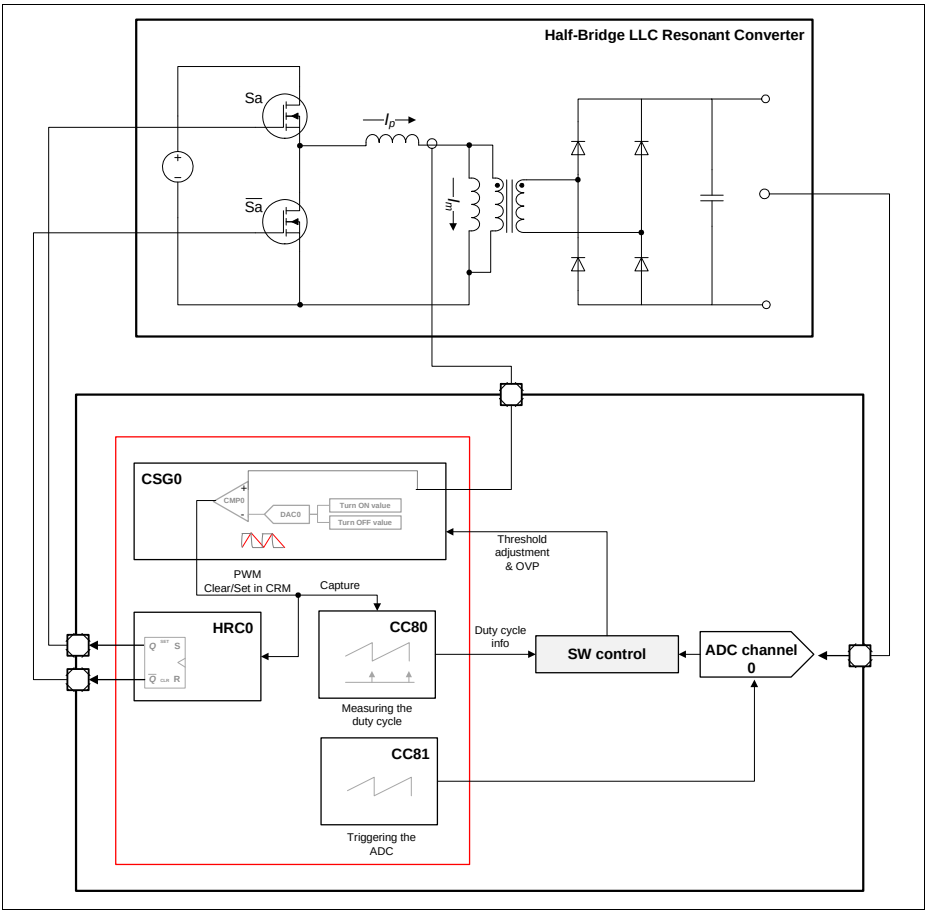


Figure 20-31 Half-Bridge LLC Resonant Converter

The control of the PWM signal is done via the Comparator present inside the CSG unit. This Comparator as an associated 10 bit DAC that contain two reference values, that can be programmed with the thresholds to SET and CLEAR the PWM signal.

The switch between the two thresholds fed to the DAC is done dynamically by HW, whenever the PWM signal passes from ON to OFF and vice-versa, **Figure 20-32**.

Dead time insertion between the complementary signals can be controlled inside the HRC unit. Each of these units has a dedicated dead time block, that is capable of

generating different dead time values for the two switching transitions, ON to OFF and OFF to ON.

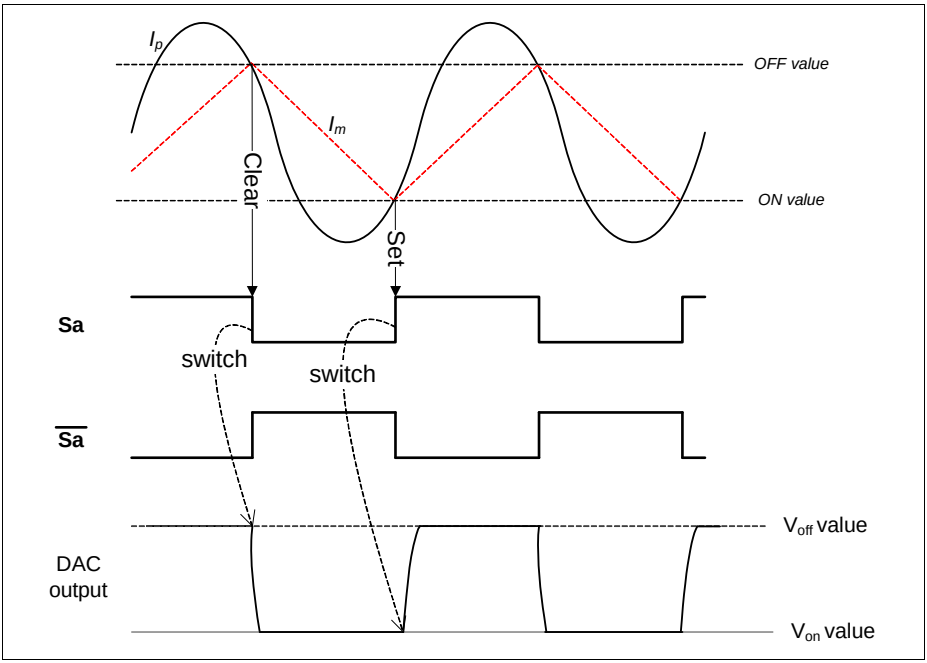


Figure 20-32 Half-Bridge LLC waveforms

20.2 Functional Description

20.2.1 Overview

The HRPWM unit is mainly comprised of two subunits: the CSGy and the HRCy, **Figure 20-33**.

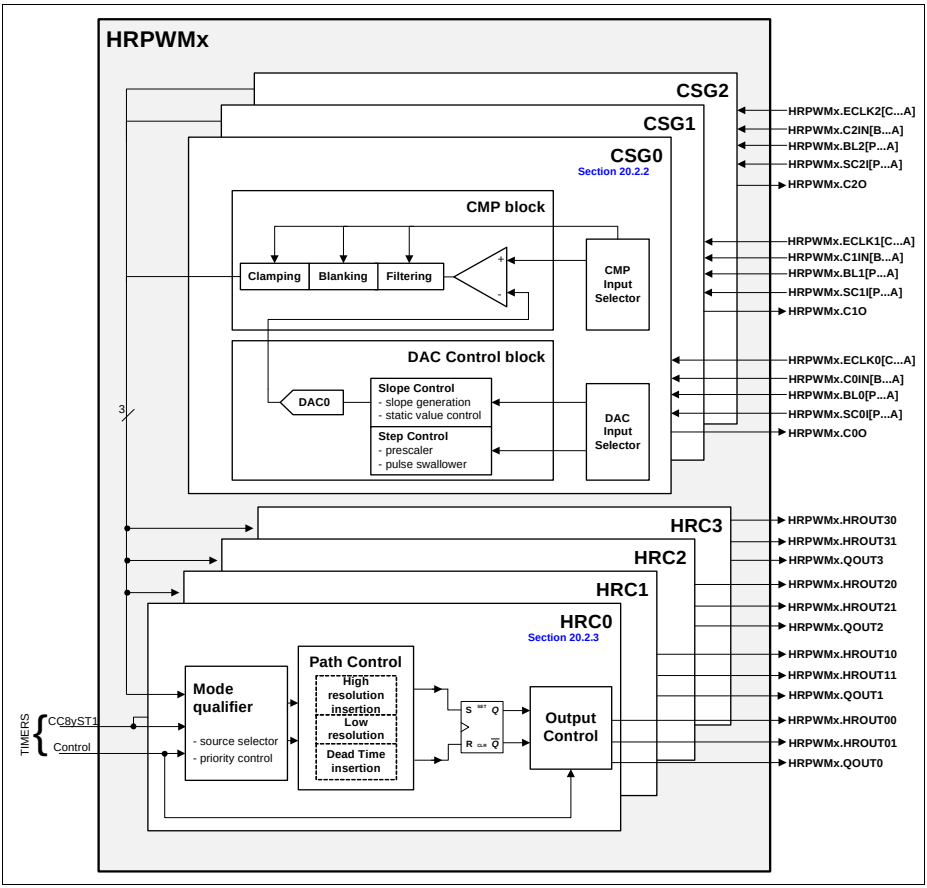


Figure 20-33 HRPWM logic overview

The two major function wise blocks inside a CSGy unit are:

- one High Speed Comparator block
- one DAC control block

Each CSGy unit has a set of 16 dedicated digital inputs, that can be decoded to perform several functions inside the DAC Control Block. The decoding of the external signals is done via the DAC Input Selector.

The Comparator Block also has a dedicated set of inputs, that can be used to perform several functions like start the blanking, clamping the comparator output, etc. The comparator input also has an associated multiplexer, that permits the user to choose one out of the two available analog inputs. The decoding of these signals is done inside the Comparator Input Selector.

Each comparator inside the CSGy has a dedicated digital output, that can be synchronized with the module clock and it can work in blanking or filtering mode. The blanking mode is especially useful to disable the comparator output at the moment that the external switch is being turned ON (avoiding a premature shut down of the switch).

The HRCy subunit works as an add on to one of the Capture/Compare unit channel. Each HRCy unit is able to generate two extra PWM outputs (complementary outputs), with a maximum resolution of 150 ps by placing the edges accordingly with a programmable value.

Each HRCy unit contains two programmable values that are used to place the rising and falling edge accordingly. The timer function inside the Capture/Compare Unit, is used to generate the coarse ON time for the PWM signal (with the specific timer clock frequency), while the HRCy unit is used to shorten or extend the signal with a 150 ps resolution. See **Section 20.2.3** for the description of this subunit.

The TRAP output control function present at the Capture/Compare Unit, is still usable in the High Resolution outputs.

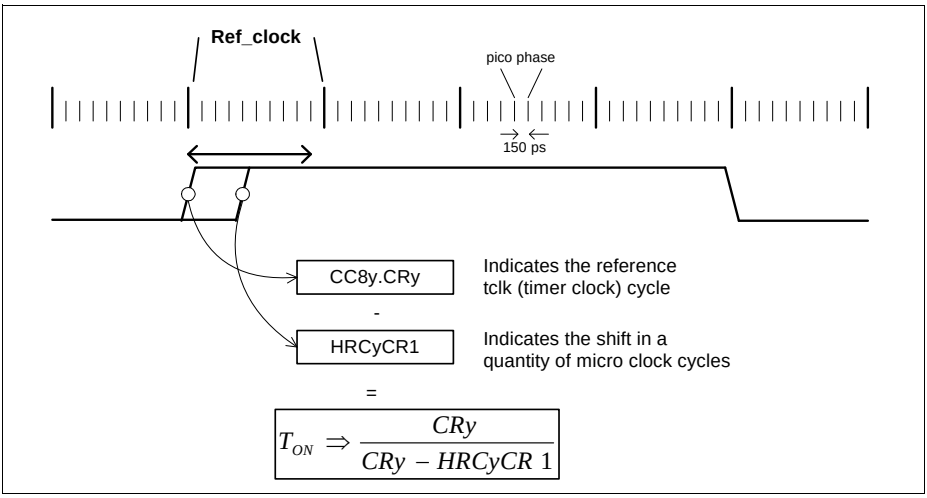


Figure 20-34 High Resolution signal generation (rising edge adjustment)

It is also possible with the built in linking of the CSGy with the HRCy subunits to control directly the PWM signal via the comparator block. With this, one can emulate a Comparator + Slope Generator/Oscillator + Latch control mechanism.

The table below summarizes the input and output signals for each HRPWM unit (internal connectivity with the Capture/Compare Unit is not listed):

Table 20-4 HRPWM pin description

Pin	I/O	Description
HRPWMx.CyIN[B...A]	I	Analog inputs for the high speed comparator
HRPWMx.BLy[P...A]	I	Digital inputs for the high speed comparator to control the blanking and output passive level
HRPWMx.SCyI[P...A]	I	Digital inputs used for controlling the slope/DAC control unit
HRPWMx.ECLKy[C...A]	I	Additional clock input for the slope generation
HRPWMx.CyO	O	Output of the high speed comparator
HRPWMx.HROUTy0	O	High resolution PWM output 0
HRPWMx.HROUTy1	O	High resolution PWM output 1
HRPWMx.QOUTy	O	High resolution PWM latch output (without dead time)
HRPWMx.SR[3...0]	O	Service request outputs

Note: The regular Capture/Compare Unit PWM outputs are still available even when the HRPWM extension is present.

Note: The Service Request signals at the output of the HRPWM are extended for one more module clock cycle.

20.2.2 Comparator and Slope Generation unit (CSGy)

The CSG units inside the HRPWM contain two major function wise blocks, **Figure 20-35**:

- Comparator Block
- DAC Control Block

Beside these two major blocks, inside a CSGy unit one can find the Interrupt Control and two Input Selectors (one for the Comparator Block and another for the DAC Control Block). The Input Selectors are used to map the external signals into internal functions.

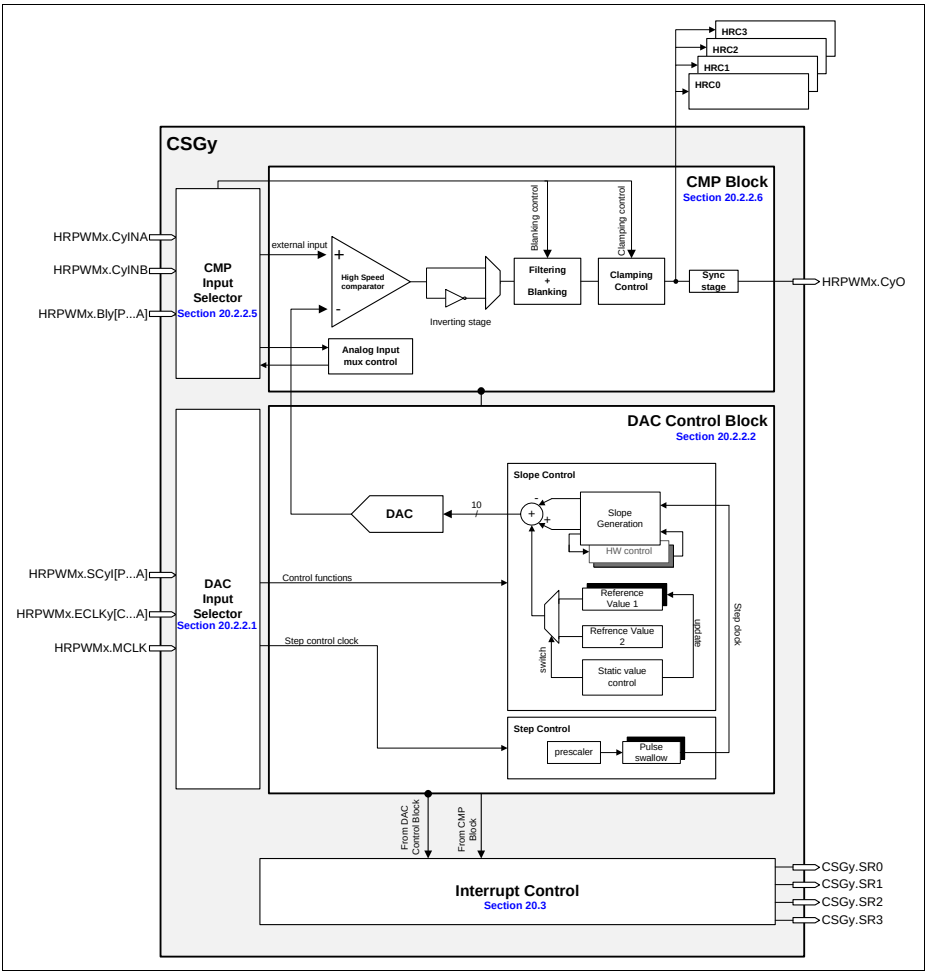


Figure 20-35 CSGy block diagram

Comparator Block

The Comparator Block, contains the associated analog high speed comparator and associated functions, output inversion, blanking and filtering stage and a clamping control. See **Section 20.2.2.6**, for a complete description of the stages and functions inside the Comparator Block.

The blanking control is very useful for SMPS applications, using trailing edge current control, to avoid the premature shut down of the switch due to noise on the line after a turn ON.

The Filtering Stage can be used to perform a filtering of a few clock cycles, of the comparator output signal to avoid some line noise or long settling time of the analog input.

After the Filtering/Blanking stage, a Clamping Control is available. This stage is used to clamp the output of the circuit into a passive level. The clamping of the output into passive level may be needed when an over current on the system is detected, to avoid the propagation into the HRC unit, or just to disable the comparator output during some period of time.

At the end of the signal path, one can still perform a synchronization of the comparator output (in case that the signal is not previously synchronized through the blanking/filtering stage). This synchronization may be needed if the comparator output is connected to several functions (one the same or different IPs) and the mismatch between the decoding of the signals, needs to be eliminated.

The Analog Input Mux Control, on **Figure 20-35**, is used to perform a dynamic switch between the two comparator input channels, HRPWMx.CyINA and HRPWMx.CyINB. The switch between this two analog inputs is useful to implement a hysteretic control loop. In this case one channel, e.g. HRPWMx.CyINA is used to sense the current zero crossing to turn on the switch, while the other channel, HRPWMx.CyINB is used to sense the peak current to shut off the switch, **Figure 20-36**.

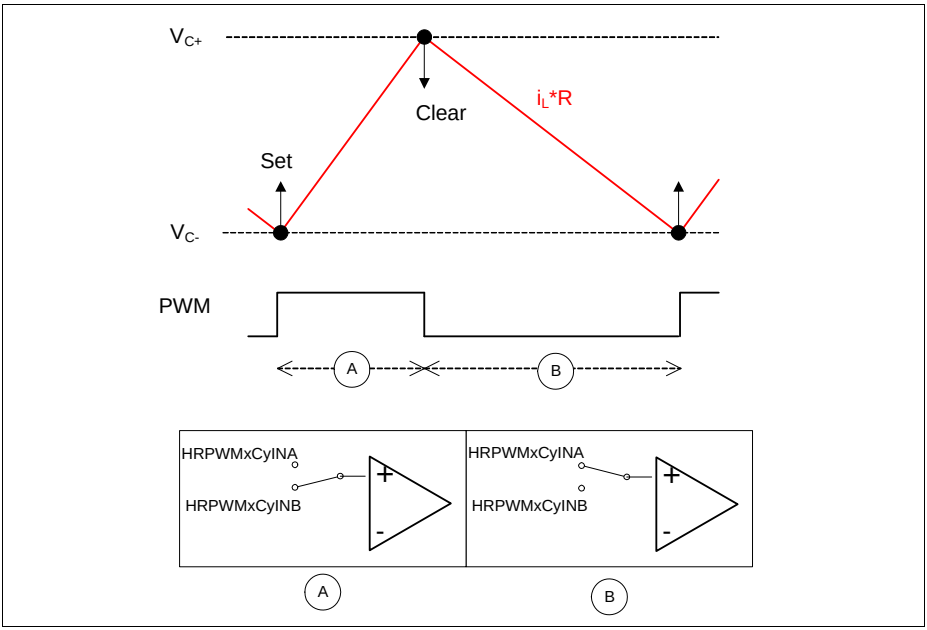


Figure 20-36 Dynamic Switch for Critical Conduction Mode

DAC Control Block

Inside the DAC Control Block, besides the 10 bit D/A Converter, one can find two major sub blocks: the Slope Control and the Step Control. Refer to **Section 20.2.2.2** for a complete description of the DAC Control block and associated stages and functions.

The Slope Control handles the several available slope generation modes. This means that it is inside of the sub block, that the value fed to DAC is controlled.

To generate a slope, the user has the availability of two reference values that can be used as slope upper or lower limit. This can emulate easily a slope with a given offset. One of this values, represented in **Figure 20-35** by Reference Value 1, has an associated shadow register enabling on-the-fly update.

In case that a slope is not needed, it is possible to use these two reference values in a complete static way. This means that the value fed to the DAC can either be Reference Value 1 or Reference Value 2.

It can also be the case that the control method is set to switch between these two values, depending on an external signal that acts as a trigger. This method is needed to complement the dynamic switch between the two analog input channels of the Comparator Block, **Figure 20-36**.

Figure 20-37 shows the different modes for the Slope Control.

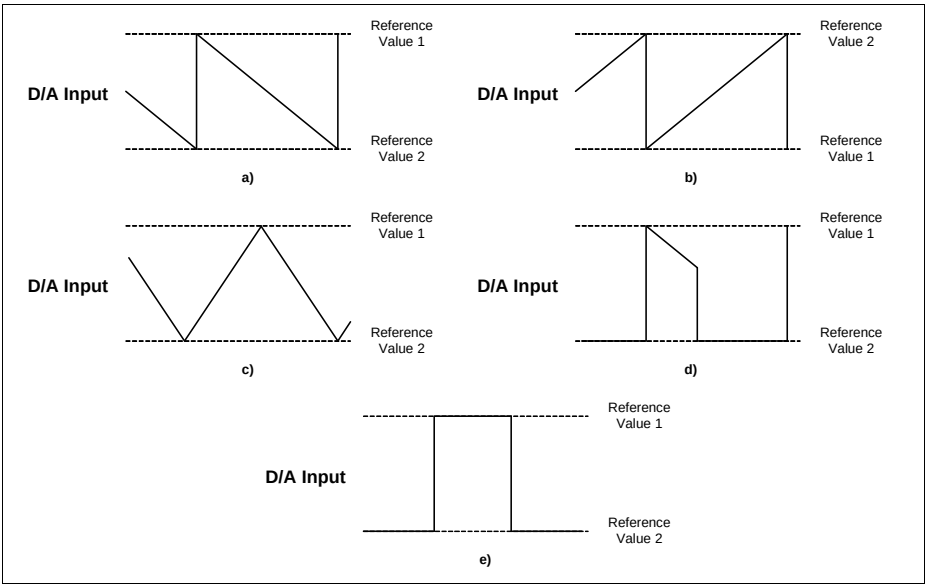


Figure 20-37 Slope Control - Modes: a) decrementing mode, b) incrementing mode, c) triangular mode, d) hybrid mode, e) static mode

The other sub block inside the DAC Control is the Step Control. This sub block generates the clock for the Slope Control.

The Step Control has a clock prescaler and a pulse swallow unit, that enable a very flexible clock generation and subsequently a very flexible slope generation.

Input Selectors

The two input selectors are used to decode the external signals into internal functions. They have a programmable level and edge detection stages and a full multiplexer stage for each function.

The combination of the full multiplexer and the edge/level detectors for each function, make the connectivity between the external components or other internal modules and the HRPWM very flexible.

Interrupt Control

The Interrupt Control block is the handling the generation of the interrupts from several number of sources. These sources are linked to the Comparator Block and the DAC Control Block.

The Interrupt Control contains a node pointer that enables the forwarding of a specific interrupt to one of the available four interrupt lines.

CSGy Connection Scheme

Figure 20-38 shows the connection scheme of each CSGy subunit. For a description of the mentioned signals, please address the specific chapter.

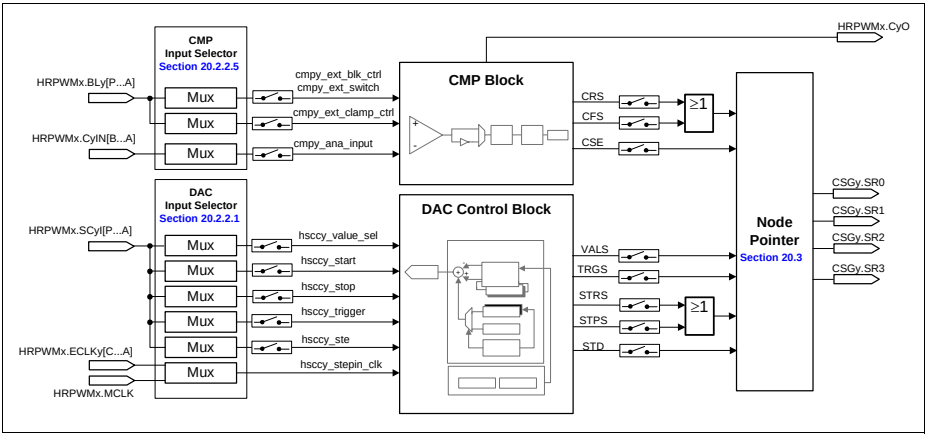


Figure 20-38 CSGy connection scheme

20.2.2.1 DAC Input Selector

On the DAC Input Selector module it is possible to program which inputs are going to be used for a specific function inside the CSGy unit, **Figure 20-39**.The functions that can be controlled directly via external pins are:

- Selection of which digital value is fed to the DAC (hsccy_value_sel)
- Start/Clear of the slope generation (hsccy_start)
- Stop/Clear of the slope generation (hsccy_stop)
- Conversion trigger for the DAC (hscc_trigger)
- Update enable for the programmable values (hsccy_ste)
- Clock used for the ramp generation/DAC (hsccy_stepin_clk)

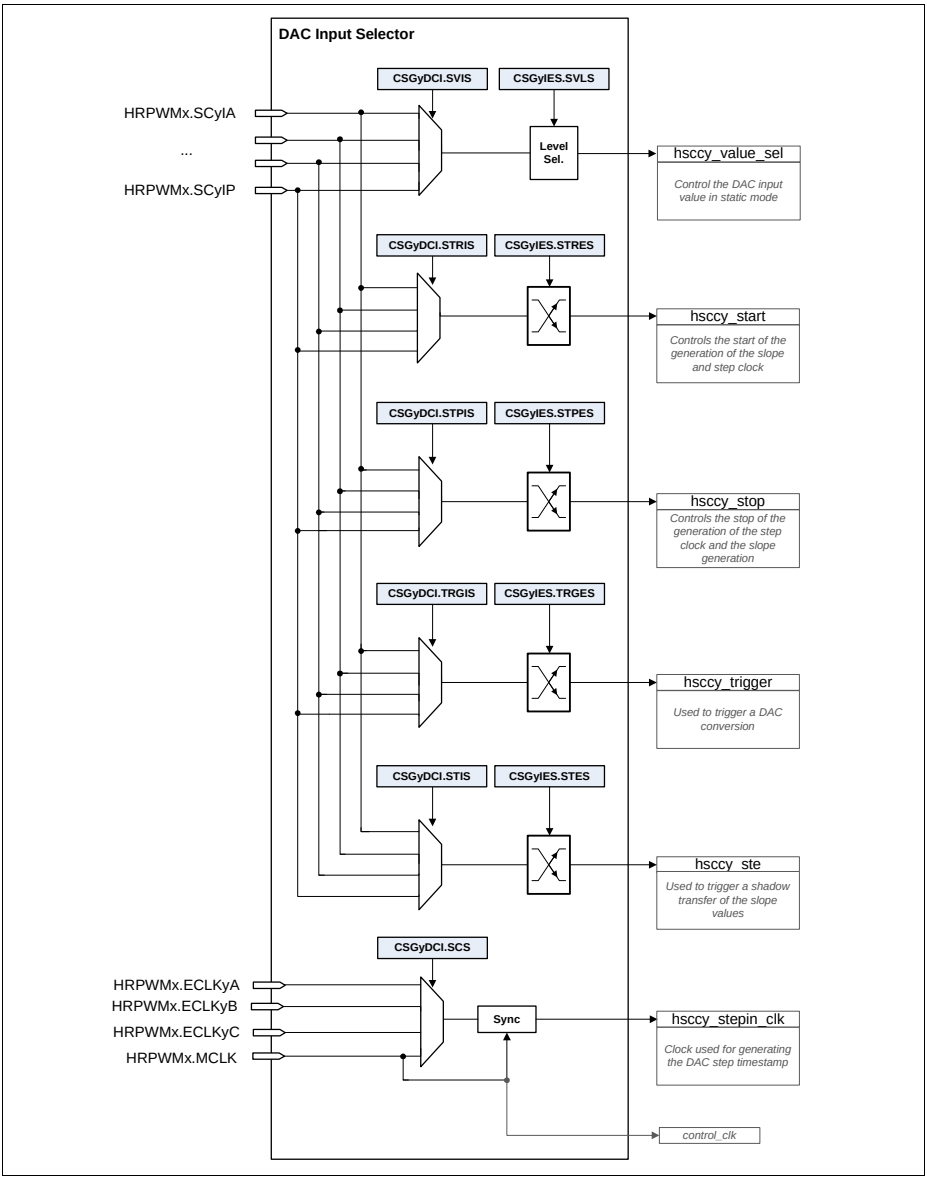


Figure 20-39 Input Selector Logic

Value Selector

The signal hsccy_value_sel is used to switch between the two programmable reference digital values (only if **CSGySC**.SCM = 0_H).

It has an associated programmable level selector, that makes possible for the user to select which is the ACTIVE level. The ACTIVE level can be mapped to the input logic level low or logic level high. When this signal is ACTIVE, the value fed to the DAC is always the one programmed into the **CSGyDSV1** register. When the signal is INACTIVE, then the value fed to the DAC is the **CSGyDSV2**, **Figure 20-40**.

When the Slope Control Block is not programmed into static mode (**CSGySC**.SCM different from 0_H), the hsccy_value_sel cannot be used.

The maximum frequency to switch between the two values is dictated by the latency time for the DAC and the propagation delay of the Comparator. Please address **Section 20.5.3** for a complete description of these values.

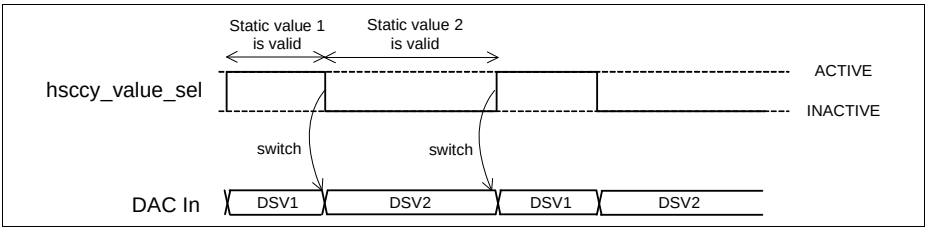


Figure 20-40 DAC value selection overview

Slope Start

Through the DAC Input Selector one is able to select an external signal that acts as a start for the slope generation, hsccy_start. This signal can not only be mapped to start the slope generation but also to start the generation of the step clock (the clock used inside the slope generation block).

This signal can also be used as a re-start condition or to clear the slope.

All these modes need to be programmed via the **CSGySC**.SSRM and **CSGySC**.PSRM, **Figure 20-41**. See a detailed description of these modes on **Section 20.2.2.2**.

It is also possible to select the active edge for the mapped input signal, rising edge, falling edge or both edges.

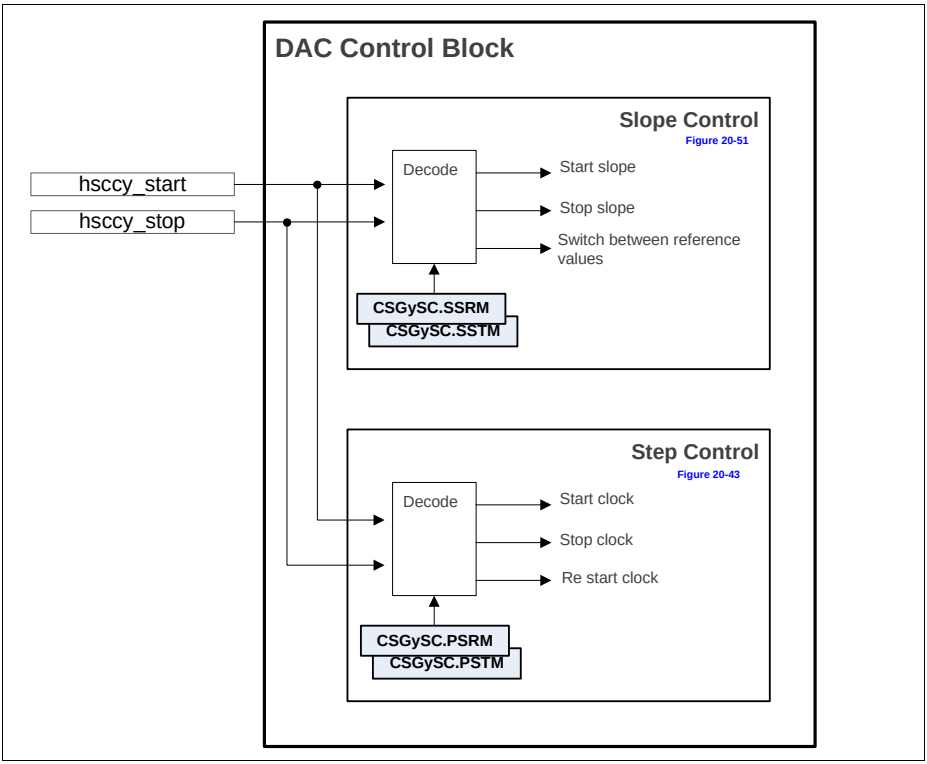


Figure 20-41 Start/Stop control decode overview

Slope Stop

The slope stop signal, hsccy_stop, has the inverse functionality of the start signal. It can not only stop the slope generation but also stop the generation of the slope clock. It can also be used as a simple slope clear.

These modes need to be programed accordingly on the **CSGySC**.SSTM and **CSGySC**.PSTM, **Figure 20-41**. See a detailed description of this modes on **Section 20.2.2.2**.

DAC trigger

When the slope generation unit is not enabled, it is possible to control the DAC conversion trigger via an external signal, hsccy_trigger. This is especially useful when

more than one comparator unit is being used in parallel and a delay is needed between two DAC conversions.

The dac trigger function is only available if the Slope Control Block mode is set to static, **CSGySC**.SCM = 0_H.

Shadow Transfer Enable

Each DAC Control Block contains one reference value, **CSGyDSV1**, that has an associated shadow register. Besides this reference value, the pulse swallow compare register, **CSGyPC**, also contains a shadow register.

These shadow registers allow an on-the-fly update of the control values, **Figure 20-42**. This is especially useful to perform adjustments on the slope value to achieve the best control loop performance.

When using more than one CSGy unit to control a n-phase DC/DC converter it may be needed to perform a cascaded update of these values, e.g, update the CSG0 values (CSG0DSV1, CSG0PC) and only after that the CSG1 values (CSG1DSV1, CSG1PC).

To enable this the user needs to map an external signal, that is going to be used to enable the shadow transfer (after the SW requested it) to the hsccy_ste signal.

The hsccy_ste gates the shadow transfer untill this signal becomes active. See **Section 20.2.2.4**, for a description of the shadow transfer logic.

Like for the other functions, the user can select the active edge of the hsccy_ste signal.

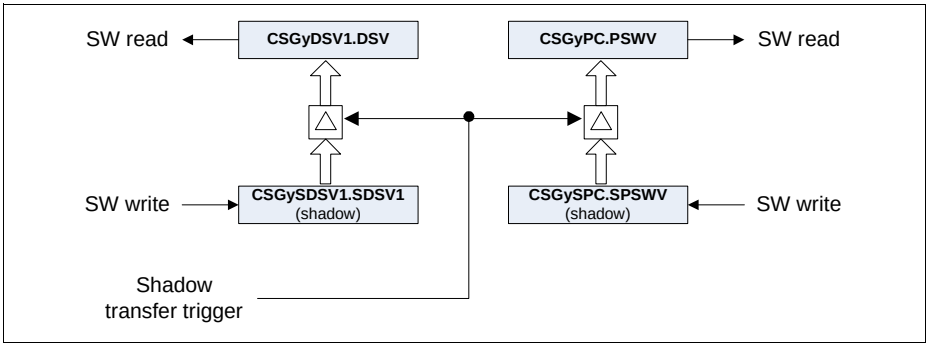


Figure 20-42 Shadow transfer overview

Clock for the Step Control

The clock used for the Step Control block is by default the module clock, nevertheless to give more flexibility for the slope generation it is possible to select another clock source from the inputs HRPWMx.ECLK[C...A]. This external clock source needs to respect the absolute maximum clock ratios described in **Section 20.5**.

20.2.2.2 DAC Control Block

The DAC Control Block contains two major functions: generating the frequency for the step clock (Step Control) and controlling the slope generation (Slope Control).

Step Control

Inside the Step Control block the clock for the Slope Control is generated, **Figure 20-43**. Inside this block one has two dedicated clock prescalers (one fixed prescaler and one programmable) and a clock pulse swallower.

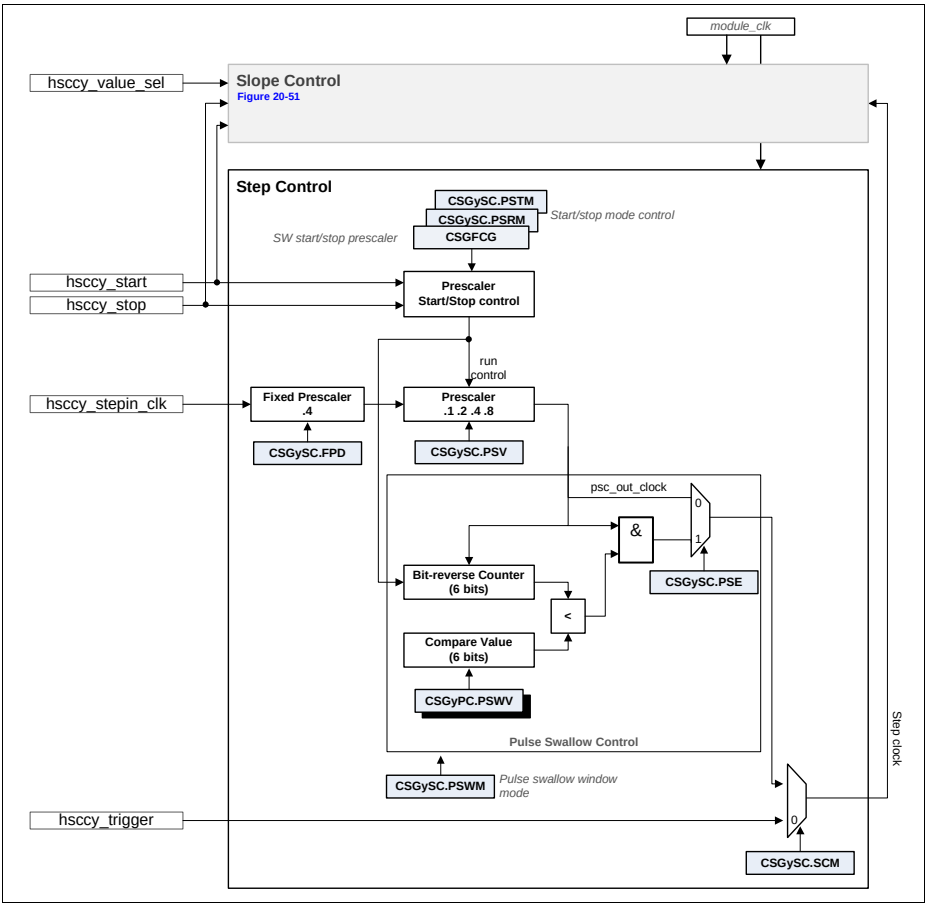


Figure 20-43 Step Control overview

With the two dedicated prescalers, one can achieve a very flexible step clock scheme that can cover a big range of switching frequencies. This is especially useful when using a peak current control method.

An external signal can also be used to generate the step clock for the Slope Control Block (and consequently the DAC conversion trigger). This external trigger, hsccy_trigger, can only be used when the Slope Control Block mode is set to static (**CSGySC**.SCM = 0_H).

The pulse swallower sub unit, is used to generate intermediate slope decay values. This is achieved by programming a certain number of clock pulses, within a timing window, that are going to be suppressed/swallowed.

Figure 20-44 shows example of the pulse swallower sub unit usage. In this example three slopes are present. This slopes are generated not by modifying the initial slope start value but by modifying the step clock frequency.

On example A, a slope is generated with a step clock of 30 MHz, nevertheless from the system point of view this is not the optimum solution due to the fact that the slope decay is too fast.

One possibility is to slow down the step clock to half, 15 MHz, (or to decrease the step size) but when this is done the slope is still not close to the proper value. Now the slope decay is to slow.

To achieve an intermediate solution, the pulse swallower can be programed to suppress a certain number of step clock pulses (while maintaining the step clock with a 30 MHz frequency). By doing this, one can achieve an intermediate slope decay between A and B.

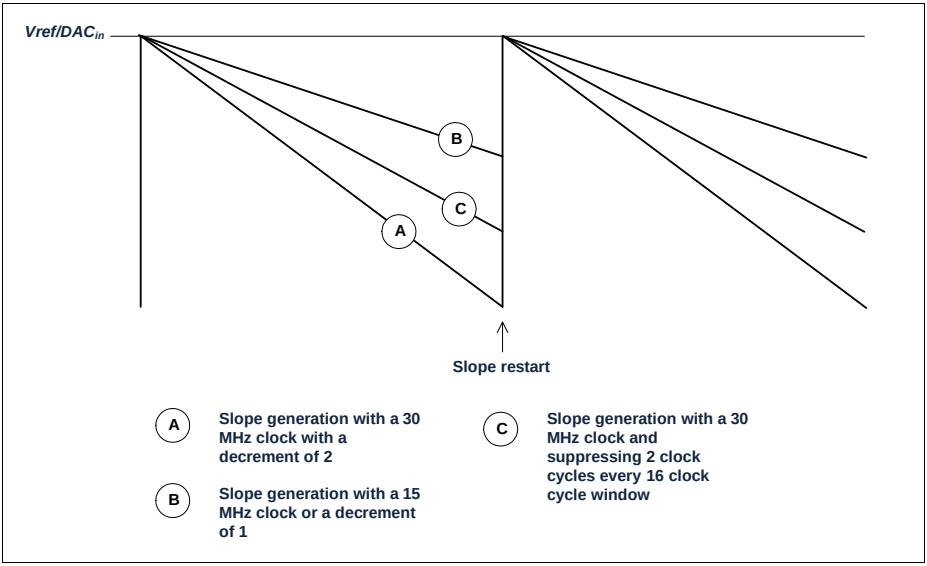


Figure 20-44 Pulse swallow application example

With the above example, the functionality of the pulse swallower can be mapped to **Figure 20-45**. By generating intermediate slope decay values, the comparison point between the inductor peak current can be finely adjusted. This emulates a very flexible ramp generator.

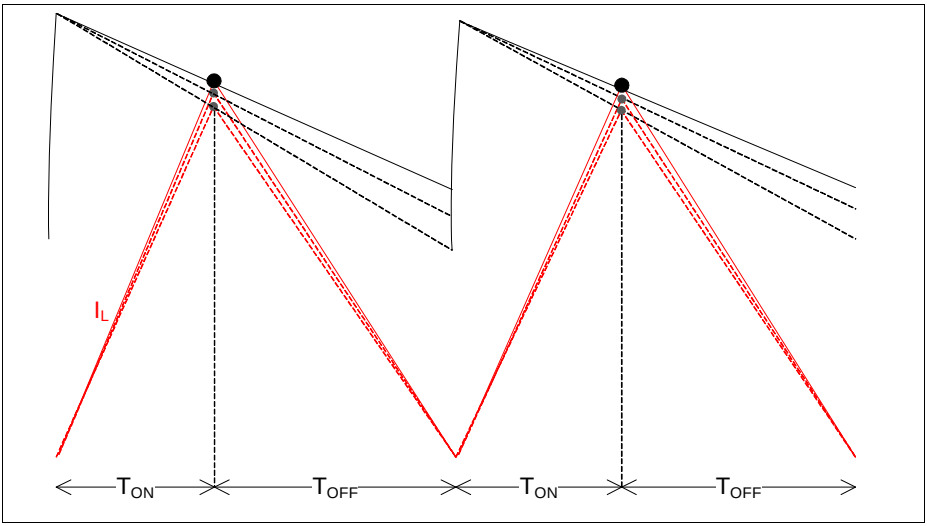


Figure 20-45 Intermediate ramp clock frequencies

The start and stop signals, hsccy_start and hsccy_stop, are used to start or stop the generation of the clock inside the step control block. They need to be programmed accordingly depending on the type of operation mode used for the slope generation. It is also possible to control the start and stop of the clock generation via SW.

The start signal, hsccy_start, can be used with the following configurations for the prescaler inside the step control block:

Table 20-5 Prescaler start function

CSGySC.PSRM	Function
00_B	Not used
01_B	Start
10_B	Clear
11_B	Clear & Start

In the same way, the stop signal can be mapped to the following functions for the prescaler:

Table 20-6 Prescaler stop function

CSGySC.PSTM	Function
00_B	Not used
01_B	Stop
10_B	Clear
11_B	Clear & Stop

Figure 20-46 shows the generation of the step clock when a frequency of module clock divided by 8 was chosen (fixed prescaler is enabled, dividing the clock by 4 and the programmable prescaler is post dividing this clock by 2). An external signal is used to start the step clock generation and a stop signal is used to freeze the slope (by gating the clock generation).

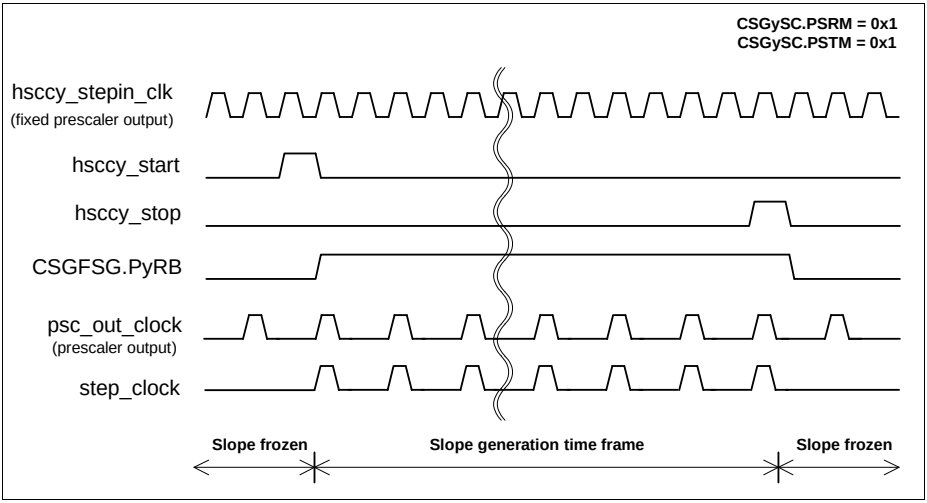


Figure 20-46 Step clock generation - Start & Stop

When the generation of the step clock is stopped and re started via external signals is possible to generate some jitter on the duration of the first value passed to the DAC. This is due to the fact that the generated clock can be a subdivision of the module clock, which means that re start signal may not be aligned with the prescaler count value, **Figure 20-47**.

This can lead to different time deviations at the first value. This is not a problem if the slope generation is controlling fully the PWM signal, with the Clear and Set being generated via the Comparator. Nevertheless when the switching frequency is dictated

via a Timer (of Capture/Compare Unit), not re synchronizing the slope can lead to a small deviation of the wanted charge/discharge time (especially for slow switching frequencies).

The slope can be synchronized with a Timer unit by using the start (or stop signal) as a start & clear. By selecting this configuration, the actual value for the prescaler counter is always cleared before starting the step clock generation. This imposes a fixed time for the first DAC step, **Figure 20-48**.

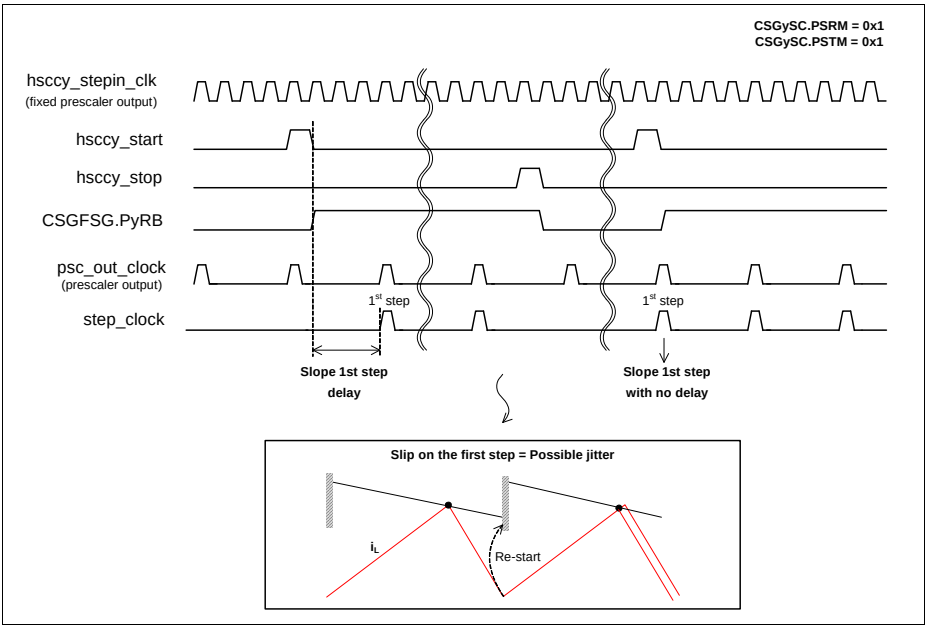


Figure 20-47 Step clock generation - first step jitter

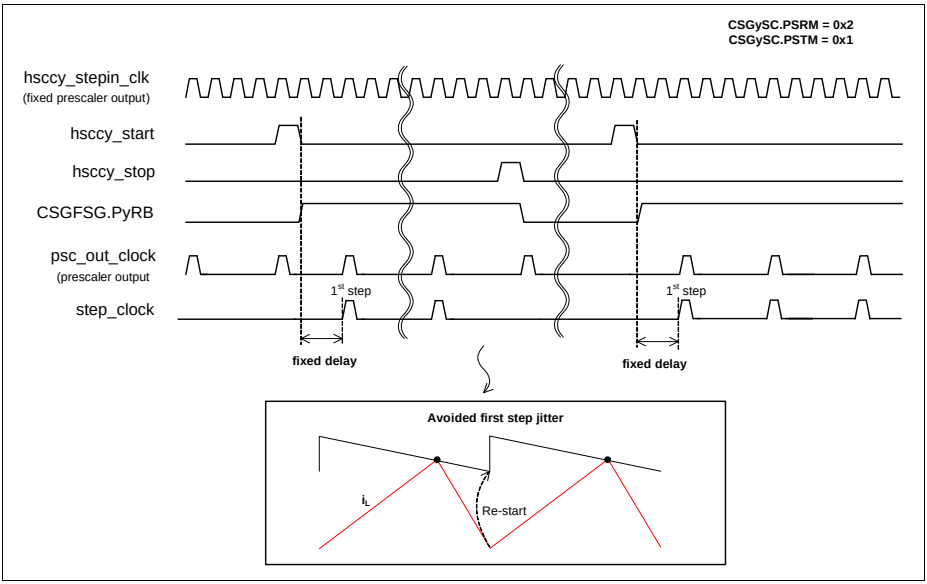


Figure 20-48 Step clock generation - Clear & Start

Inside the Step Control block there is also a pulse swallow unit. This unit contains a dedicated 6 bit compare value, **CSGyPC**.PSWV, that can be programmed accordingly with the number of pulses that need to be swallowed to achieve the wanted step clock distribution (and consequently the wanted slope decay).

Through the **CSGySC**.PSWM field, it is possible to adjust the pulse swallow window, **Table 20-7**.

Table 20-7 Pulse Swallow window

CSGySC.PSWM	Description
00_B	16 clock cycle window mode: The value of pulses programmed into the PSWV are swallowed over a 16 clock cycle window
01_B	32 clock cycle window mode: The value of pulses programmed into the PSWV are swallowed over a 32 clock cycle window
10_B	64 clock cycle window mode: The value of pulses programmed into the PSWV are swallowed over a 64 clock cycle window
11_B	Reserved

Figure 20-49 shows the operation of this mechanism when the pulse swallow value is set to 4 (**CSGyPC**.PSWV = 04_H) and **CSGySC**.PSWM is set to 00_B (16 clock cycle window mode).

Notice that the pulse swallow counter is a bit reverse counter, which means that the swallowed pulses are going to be distributed evenly over a 16 clock cycle period (clock at the output of the programmable prescaler, psc_out_clock).

With this mechanism one is able to generate intermediate slope decay values between two different clock division ratios.

Due to the fact that the pulse swallow counter operates with the prescaler output clock, when a external signal is used as start & clear for the prescaler, the pulse swallow counter is also cleared. Like the previous method for clearing the prescaler, some latency on the response is introduced, but the jitter imposed by different number of swallowed pulse during the first 16 clock cycles is avoided (after re starting), see **Figure 20-50**.

The counting scheme for a bit reverse counter is demonstrated in **Table 20-8**. The action of swallowing/suppressing the clock pulse for a **CSGyPC**.PSWV value of 4_D is exemplified.

Table 20-8 Bit reverse counter- CSGyPC.PSWV = 4_D

counter[3]	counter[2]	counter[1]	counter[0]	Action
0	0	0	0	Swallow
1	0	0	0	-
0	1	0	0	-
1	1	0	0	-
0	0	1	0	Swallow
1	0	1	0	-
0	1	1	0	-
1	1	1	0	-
0	0	0	1	Swallow
1	0	0	1	-
0	1	0	1	-
1	1	0	1	-
0	0	1	1	Swallow
1	0	1	1	-
0	1	1	1	-
1	1	1	1	-

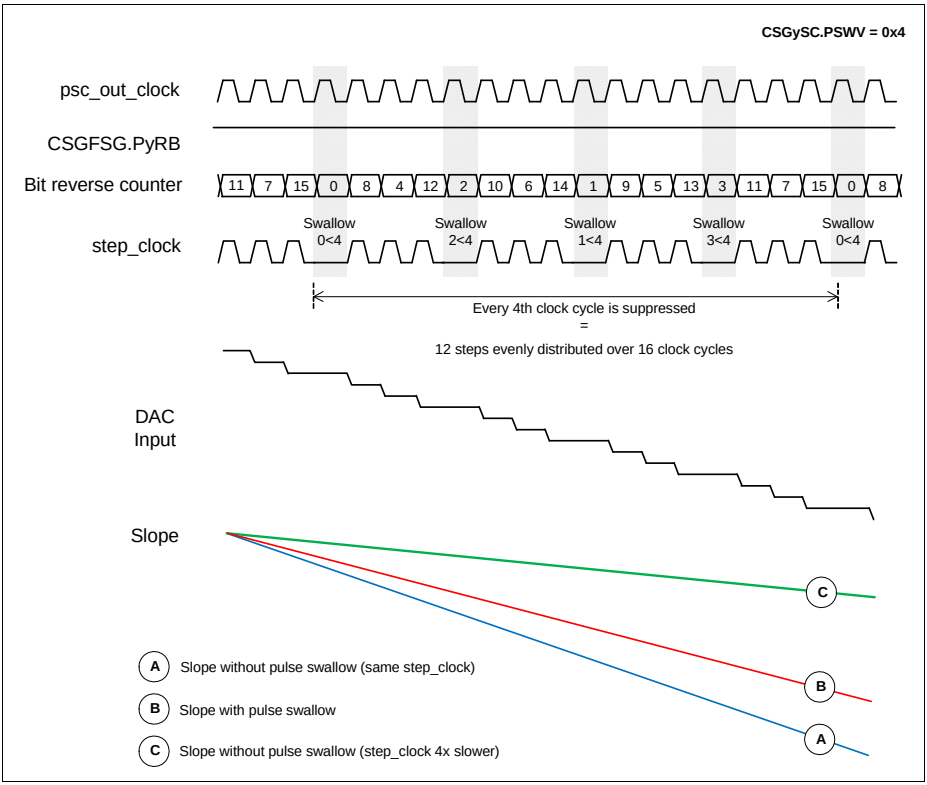


Figure 20-49 Pulse swallow operation

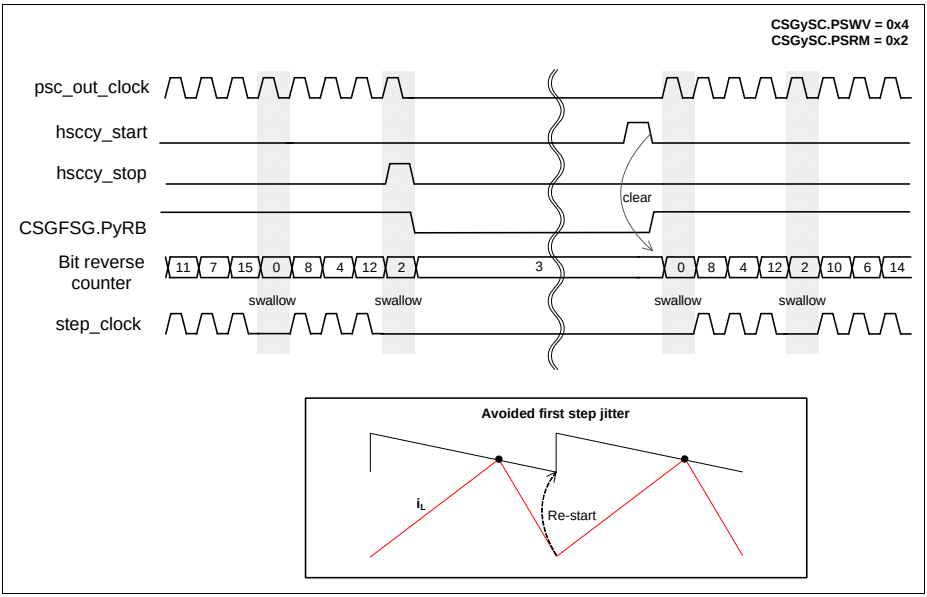


Figure 20-50 Pulse swallow with Start & Clear

Slope Control

Inside the Slope Control unit it is possible to generate several slope schemes for the DAC, that can be used in different SMPS control loop techniques.

Besides the generation of a decrementing, incrementing or triangular slope, it is also possible to have a static mode and a hybrid slope mode.

The hybrid mode is a mixed approach between an incrementing or decrementing slope with a time frame where the DAC is fed always with the same value.

Present inside the Slope Control block:

- a start/stop control unit that handles the modes for starting and stopping the slope generation
- a gain stage that is used to control the absolute step size for the slope generation
- a switch value control unit, that is used to switch between the two programmable reference values (for static mode)
- a 10 bit comparator, that is used to control the slope generation by monitoring the current DAC input value

Figure 20-51 shows all the previously described units.

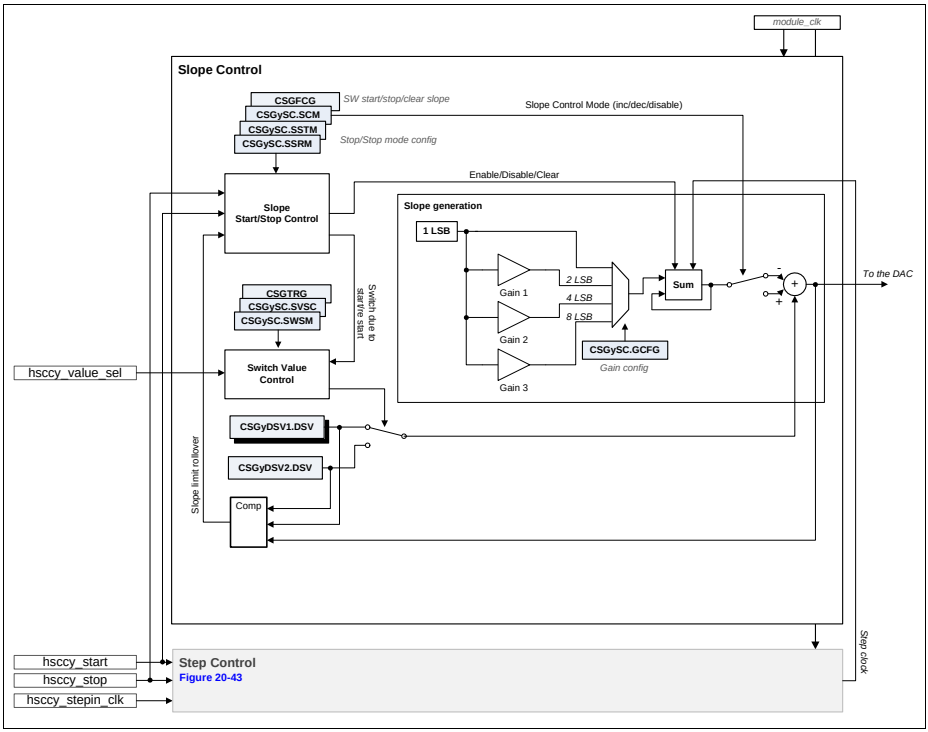


Figure 20-51 Slope Control block overview

The slope generation modes are dictated by two fields: **CSGySC**.SCM and **CSGySC**.SVSC.

The **CSGySC**.SCM field enables several slope generation schemes: incrementing, decrementing, triangular or disabled. By default, the slope generation is disabled (**CSGySC**.SCM = 00_B), which means that the value and the conversion DAC trigger can be controlled via external signals (hsccy_value_sel and hsccy_trigger) or via SW. See **Table 20-9** for the complete description of the slope generation modes.

Table 20-9 Slope control mode

CSGySC.SCM	Slope mode
00_B	Slope Generation is disabled. Value fed to the DAC is controlled via the hsccy_value_sel signal (or via SW) Usage of external DAC conversion trigger, hsccy_trigger is possible.
01_B	Slope Generation in decrementing mode. Top reference value is **CSGyDSV1**. Hybrid mode is possible by programming the **CSGySC**.SVSC field.
10_B	Slope generation in incrementing mode. Lower reference value is **CSGyDSV1** Hybrid mode is possible by programming the **CSGySC**.SVSC field.
11_B	Slope Generation in triangular mode. **CSGyDSV1** is the top reference value, **CSGyDSV2** is the low reference value.

The field **CSGySC**.SVSC controls the different schemes of how the programmed DAC reference values, **CSGyDSV1** and **CSGyDSV2**, are going to be used when the slope generation is enabled.

Notice that when the slope generation is disabled, **CSGySC**.SCM = 00_B, this additional functions are discarded. This is due to the fact that the reference value for the DAC needs to be controlled via the hsccy_value_sel or via SW.

See **Table 20-10** for a full description of how the two static values can be handled during the ramp generation.

Table 20-10 Slope reference control

CSGySC.SVSC	Reference value handling
00_B	Only **CSGyDSV1** value is used for the slope generation: • if slope is incrementing, **CSGyDSV1** is the bottom reference value from where the ramp starts • if slope is decrementing, **CSGyDSV1** is the upper reference value from where the ramp starts
01_B	The two programmable reference values are used: • **CSGyDSV1** is the low or high reference value from where the ramp starts (incrementing or decrementing respectively) • **CSGyDSV2** is used as a static value (the slope generation stops when a command is given to switch to this value) This enables the hybrid mode.

Table 20-10 Slope reference control (cont'd)

CSGySC.SVSC	Reference value handling
10_B	The two programmable reference values are used: • **CSGyDSV1** is the low or high reference value from where the slope starts (incrementing or decrementing respectively) • **CSGyDSV2** is used as an internal re start condition for the slope. This is used for a continuous slope generation.
11_B	Reserved

*Note: **CSGySC**.SVSC field needs to be programmed accordingly when the **CSGySC**.SCM field is different from 00_B or 11_B. When the slope is disabled or when a triangular slope is selected, the value programmed on the **CSGySC**.SVSC field is discarded.*

Slope control modes (SCM and SVSC)

There are two ways to control the inputs of the DAC:

- static mode, where the value fed to the DAC comes from one of the two programmable values, **CSGyDSV1** or **CSGyDSV2**, and the conversion trigger can also be controlled externally.
- dynamic mode, where the slope control block, is able to generate different types of slopes and at the same time control the switch between the **CSGyDSV1** and **CSGyDSV2** if in hybrid mode.

- **Static Mode**

In static mode, the user can map one external signal to control the switch between the two reference values, hsccy_value_sel, and another signal to control the DAC conversion trigger, hsccy_trigger.

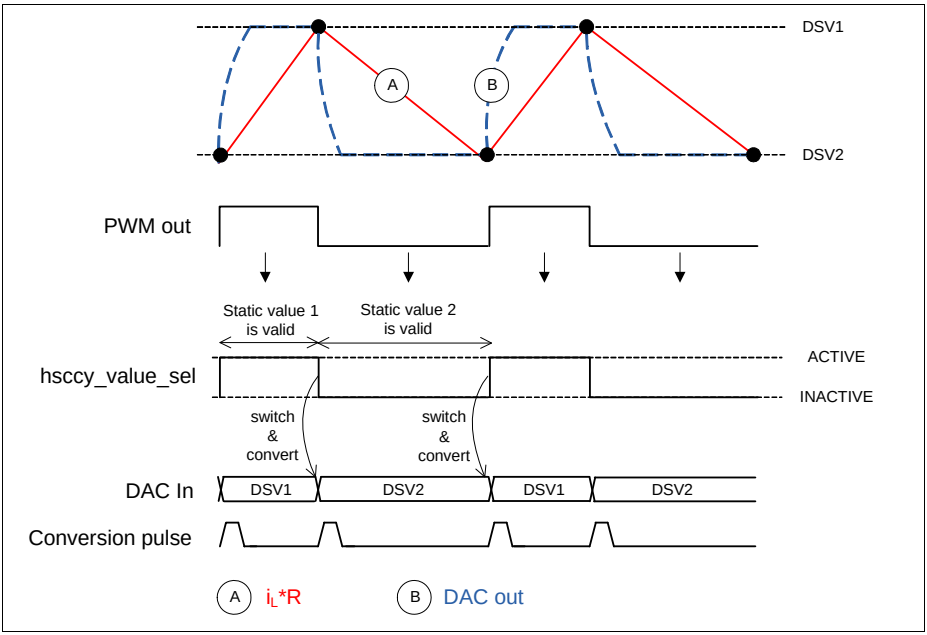


Figure 20-52 Static control mode - SCM = 00_B

Figure 20-52 shows the timing operation when the static control mode is being used. An external signal is configured to be used as value selector, hsccy_value_sel. When this function is ACTIVE, the value sent to the DAC is the one pre programmed in the **CSGyDSV1** register (see a description of how to program the ACTIVE level for the value selector on **Section 20.2.2.1**).

Every time that a switch between the values happens, the slope control block automatically generates a DAC conversion trigger.

This type of static control mode is suitable for a complete hysteretic mode (used together with the switching input function of the Comparator).

To control this mode, the value selector signal, hsccy_value_sel, can be decoded from the PWM signal itself (the output of the latch present in the HRCy unit) or it can come from a timer source.

For this static mode, it is also possible to use an external DAC conversion trigger, by mapping an external signal to the hsccy_trigger function (see a description of how to program the trigger on **Section 20.2.2.1**).

When an external conversion trigger is used, the switch between the values doesn't generate automatically the DAC trigger, **Figure 20-53**.

This type of control can be used to generate delays between the several CSGy units that are working in parallel.

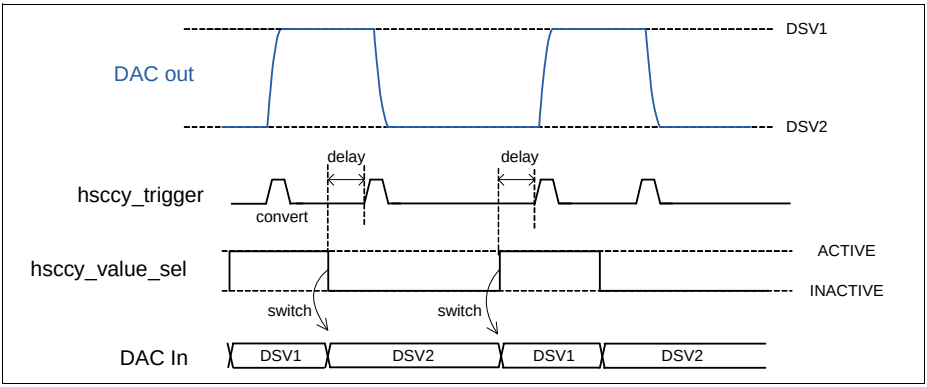


Figure 20-53 Static control mode with external trigger - SCM = 00_B

- **Dynamic Mode**

When the Slope Control Block is generating a slope, SCM configuration different from 00_B, the functions start and stop, hsccy_start and hsccy_stop, need to be configured accordingly with the wanted operation mode via the fields **CSGySC**.SSRM and **CSGySC**.SSTM.

The hsccy_start and hsccy_stop are the signals that control the start/restart or stop of the slope (it is also possible to control the start and stop of the ramp generation via SW).

Table 20-11 and **Table 20-12** list the available configurations for these signals.

Table 20-11 Start signal configuration (hsccy_start)

CSGySC.SSRM	**Signal function**
00_B	Not used. This configuration is used when in static mode.
01_B	Starts the slope generation. After the slope generation is already on going it works as a restart.
10_B	Resumes the slope. This doesn't restart the slope but resumes the slope from the point where is was stopped/halted.
11_B	Reserved

Table 20-12 Start signal configuration (hsccy_stop)

CSGySC.SSTM	Signal function
00_B	Not used. This configuration is used when in static mode control.
01_B	Stops/freezes the ramp generation.
10_B	Used in hybrid mode. It freezes the slope generation and feeds constantly the value programmed in **CSGyDSV2** to the DAC.
11_B	Reserved

Figure 20-53 shows the operation of the Slope Control Block when a decrementing slope was programmed, **CSGySC**.SCM = 01_B. Only the **CSGyDSV1** value is being used for the high start threshold of the slope, **CSGySC**.SVSC = 00_B.

To control the start of the ramp an external signal was mapped to the hsccy_start function, that was configured as start only. This means that every time that a start signal arrives, the ramp generation reloads the value fed to the DAC with the **CSGyDSV1** value and continues to decrement.

If the start is not detected until the value fed to the DAC reaches 000_H, the ramp generation is frozen at 000_H untill the start signal is again detected.

On **Figure 20-53**, the Step clock alignment indicates that the hsccy_start is also being used to clear and restart the step clock generation.

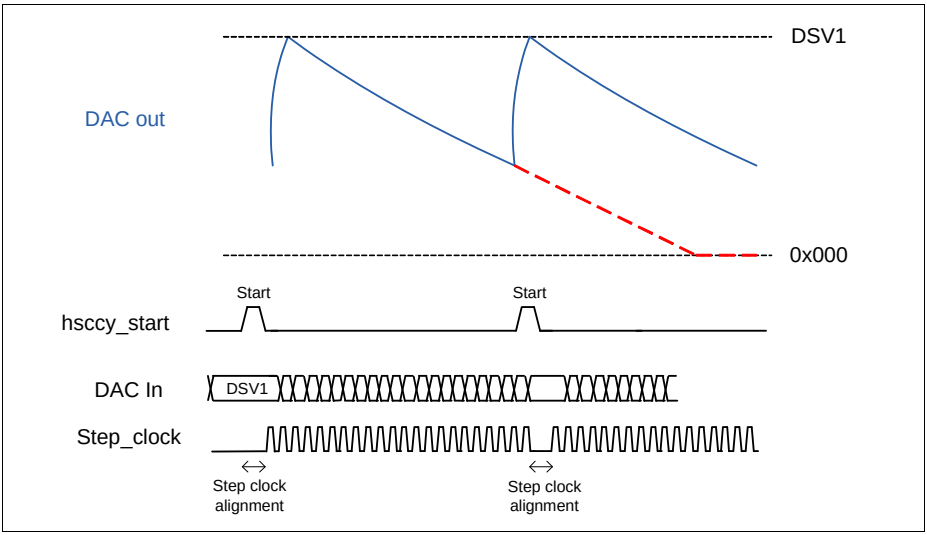


Figure 20-54 Ramp generation mode, decrementing - SCM= 01_B SVSC = 00_B

As described previously, it is also possible to have a hybrid mode for the slope generation.

The most useful application for this type of mode, it to control a peak current loop with the generated slope, while having also a small comparison threshold used for sensing the zero current crossing (using the **CSGyDSV2** value for this purpose), **Figure 20-36**.

Figure 20-55 describes the usage of this mode. The ramp generation was set to decrementing, **CSGySC**.SCM = 01_B, and we select that the two reference values are going to be used in hybrid mode, **CSGySC**.SVSC = 01_B

In this case, the **CSGyDSV1** is always the high threshold from which the slope starts to be decremented. The **CSGyDSV2** value is used as constant DAC input.

The **CSGyDSV2** value is constantly fed to the DAC after a slope stop trigger is detected.

With this slope operation mode (plus the dynamic switch of the comparator inputs), it becomes very easy to control a peak current control with zero current crossing detection.

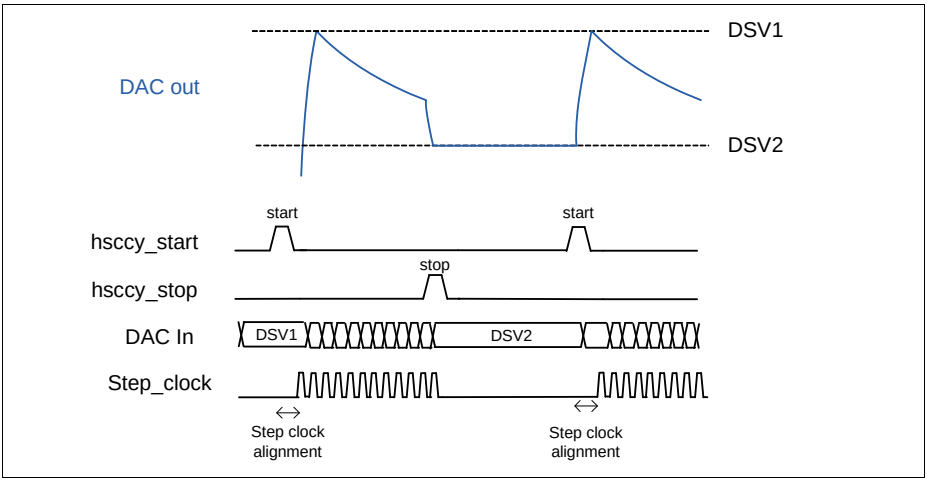


Figure 20-55 Ramp generation mode, decrementing - SCM = 01_B SVSC = 01_B

*Note: In **Figure 20-55** the step clock generation was also stopped when the ramp is stopped by programming the stop function for the prescaler accordingly. The start of the prescaler was also programmed as clear and start*

It is also possible to generate a continuous slope with an offset. For this the two reference values, **CSGyDSV1** and **CSGyDSV2**, must be used. The first one is used as the high threshold from where the slope starts, while the former is used as offset value, see **Figure 20-56**.

In this mode, the **CSGyDSV2** value is used as an internal restart signal for the ramp generation. This means that every time that the value fed to the DAC is equal or smaller (due to different step gains, it may be possible to undershoot the **CSGyDSV2** value), the ramp is re started from the value programmed into the **CSGyDSV1**.

It is still possible to use the start and stop functions mapped to external signals or to control the ramp generation via SW.

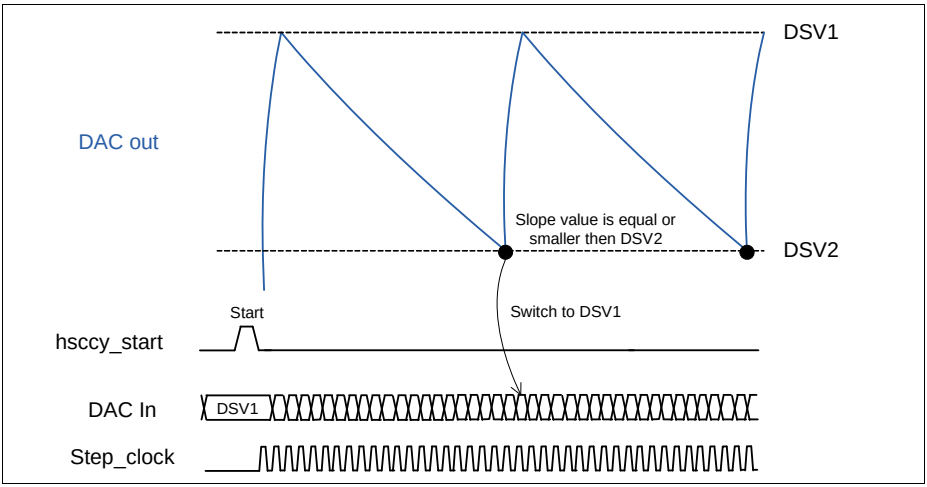


Figure 20-56 Ramp generation mode, offset ramp - SCM = 01_B SVSC = 10_B

When an incrementing slope is selected, **CSGySC**.SCM = 10_B, the threshold from where the incrementing starts is always dictated by the **CSGyDSV1** value, see the example on **Figure 20-57**.

For any type of slope, there isn't a fixed constraint that the **CSGyDSV1** value needs to be bigger or smaller than the one present in the **CSGyDSV2** register. The only constraint that exists, is that when an incrementing or decrementing slope is selected the start threshold for starting incrementing or decrementing is dictated by **CSGyDSV1** value. Therefore the value for the **CSGyDSV2** (if used) needs to be programmed accordingly by the user.

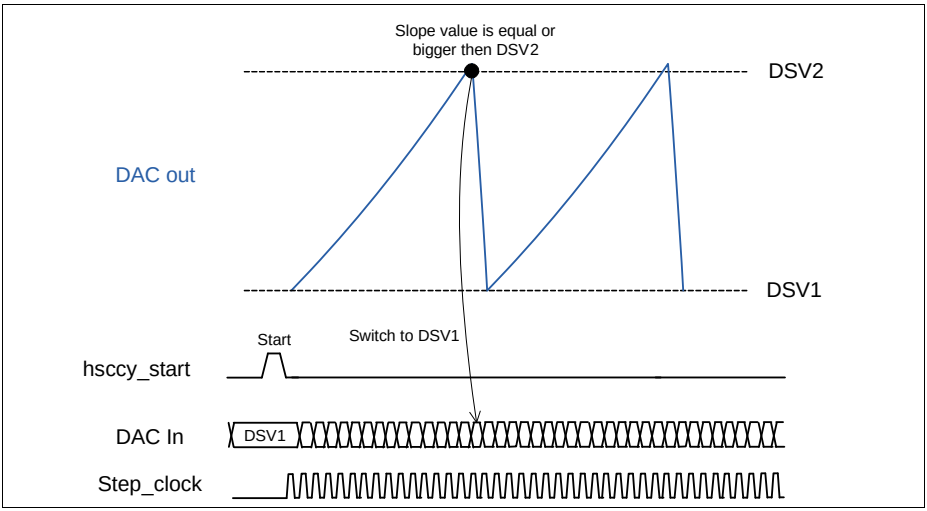


Figure 20-57 Ramp generation mode, incrementing - SCM = 10_B SVSC = 10_B

When **CSGySC**.SCM is programmed with the value 11_B, the user is selecting a triangular waveform slope generation at the DAC inputs.

In this case, the lower threshold is dictated by the **CSGyDSV2** value and the higher threshold dictated by the **CSGyDSV1** value, see **Figure 20-58**.

Notice that the default value at the DAC inputs before the triangular waveform generation starts, dictates if the control logic starts first incrementing or decrementing. See **Figure 20.2.2.3** for a full description of the initialization modes for the DAC control unit.

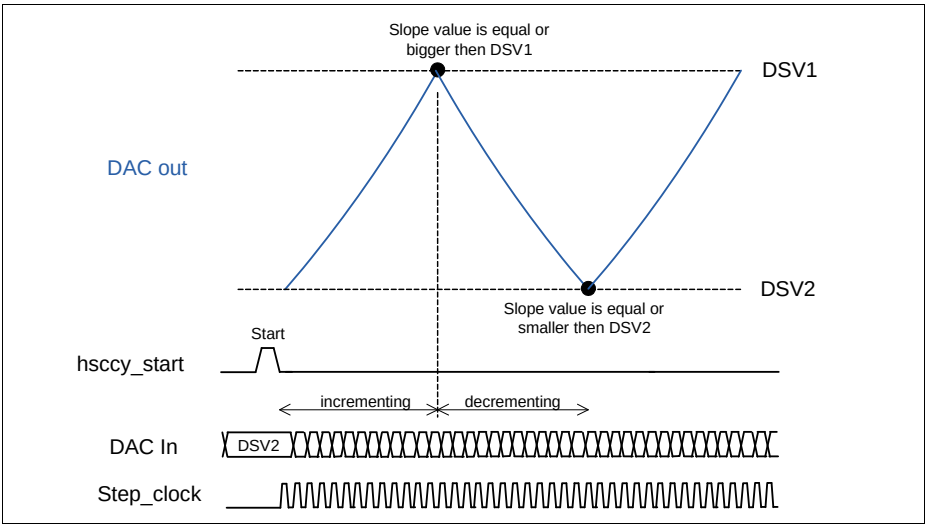


Figure 20-58 Ramp generation mode, triangular waveform - SCM = 11_B

20.2.2.3 Initialization of the DAC Control Unit

The DAC Control Unit contains a run bit that needs to be set via SW to enable the block (ramp generation or static control mode).

Due to the fact that two reference values can be used, **CSGyDSV1** and **CSGyDSV2**, a default value must be configured before setting the run bit of the DAC Control Block.

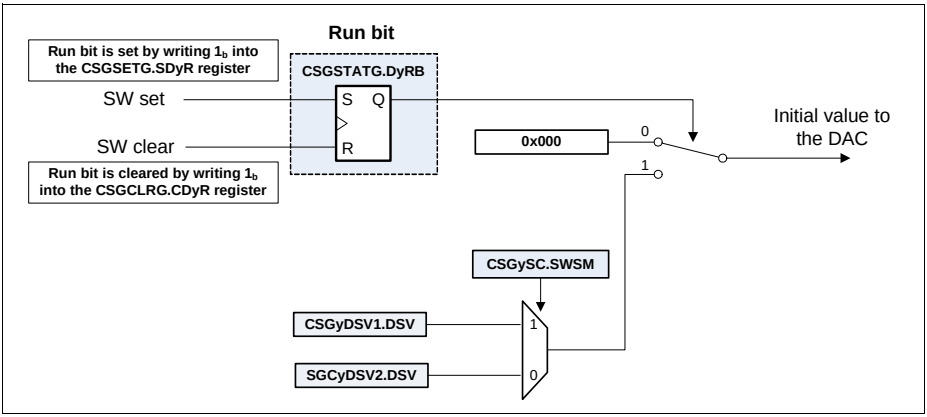


Figure 20-59 DAC Control Block run bit logic

After the run bit is set via SW, **CSGSTATG**.DyRB, it is possible to select what is the default value fed to the DAC, see **Figure 20-59**. The value at the DAC inputs when the run bit is not set is always 000_H.

After the run bit is set and depending on the configuration set in the **CSGySC**.SWSM field, it is possible to output to the DAC either the programmed value in the **CSGyDSV1** or **CSGyDSV2**.

This enables a very flexible mechanism that can accommodate all type of control operations (slope generation or static value control mode) within all types of initialization procedures for the external logic. All of this without doing a two step approach divided into initialization and after, a re programing of the DAC Control Block to perform the wanted control mode.

Another action possible at startup is to generate or not a DAC conversion trigger, see **Table 20-13**.

Table 20-13 Initialization modes

CSGySC.SWSM	**Action**
00_B	**CSGyDSV2** is fed to the DAC. Initial DAC conversion trigger is generated.
01_B	**CSGyDSV1** is fed to the DAC. Initial DAC conversion trigger is generated.
10_B	**CSGyDSV2** is fed to the DAC. Initial DAC conversion trigger is not generated.
11_B	**CSGyDSV1** is fed to the DAC. Initial conversion trigger is not generated

Figure 20-60 shows the different initialization scenarios, prior to the initialization of the slope generation (in dynamic mode). Before the start arrives, it is possible to already trigger a conversion on the DAC with one of the two reference values. This gives a high flexibility when the output of the comparator is being used to set or clear the output latch, that is controlling the PWM signal to the driver.

As an example on **Figure 20-60**, the Slope Control block was programmed to generate a decrementing slope, and therefore, after the start trigger is detected, the **CSGyDSV1** value is taken as starting point.

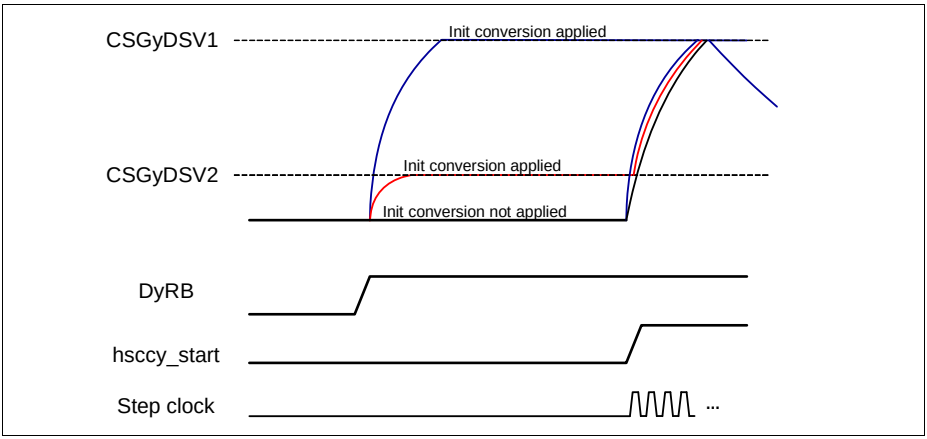


Figure 20-60 DAC Control initialization with slope generation mode

The same methodology can be used when the static control mode is preferred over the dynamic mode, **Figure 20-61**. In this case an initial conversion can be applied with the wanted reference value, before the first transition of the hsccy_value_sel signal.

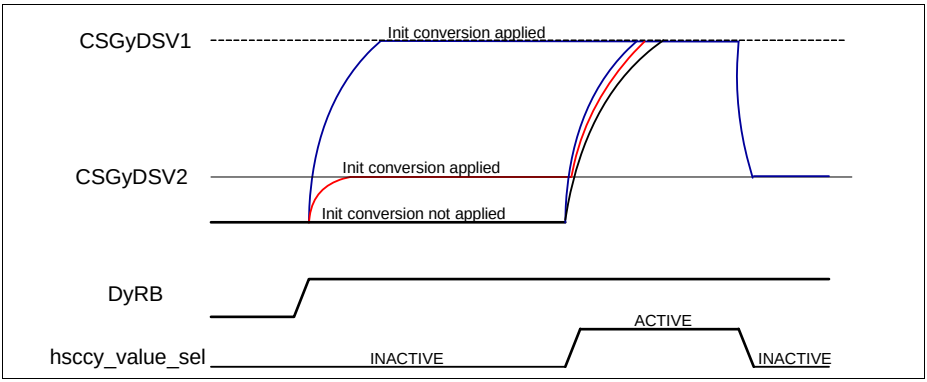


Figure 20-61 DAC Control initialization with static control mode

20.2.2.4 Shadow transfer for DAC Control unit

The **CSGyDSV1** and **CSGyPC** have an associated shadow transfer mechanism, **Figure 20-42**. This enables a clean update of these two fields under all the operation modes.

This functionality is especially useful for:

- if tuning is needed during continuous operation, to maximize the performance of a SMPS converter.
- to update the block with new values processed via the voltage loop. This is normally not done in a cycle-by-cycle operation.

The logic controlling the update of the **CSGyDSV1** and **CSGyPC** registers is seen at **Figure 20-62**.

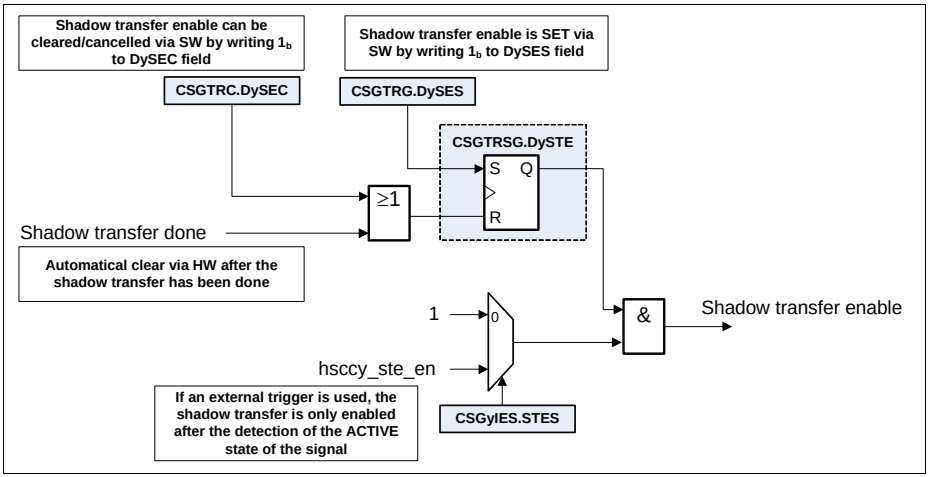


Figure 20-62 Shadow transfer logic

The request for a shadow transfer update is done via SW, with a write operation to the SET register linked with the **CSGTRSG**.DySTE field (writing a 1_B into the **CSGTRG**.DySES).

The **CSGTRSG**.DySTE field is automatically cleared when the shadow transfer is performed and remains 1_B while the shadow transfer is pending. Additionally the SW can also clear/cancel the enable by writing a 1_B into the **CSGTRC**.DySEC field.

After the DySTE is set to 1_B, the shadow transfer is enabled, and within the next trigger (controlled via HW), the specific values are going to be updated with the values written previously in the shadow registers,.

The update of the **CSGyDSV1** and **CSGyPC** register is always done when a switch between the **CSGyDSV2** to **CSGyDSV1** value occur if a triangular slope generation was not selected (**CSGySC**.SCM = 11_B selects a triangular waveform).

When a triangular slope was selected, the shadow transfer occurs when the waveform passes from decrementing to incrementing.

When the Run Bit of the DAC Control Unit is not set, the shadow transfer is done immediately.

Figure 20-63 shows the shadow update mechanism when the an external signal is being used to switch between the two DAC reference values, **CSGySC**.SCM = 00_B.

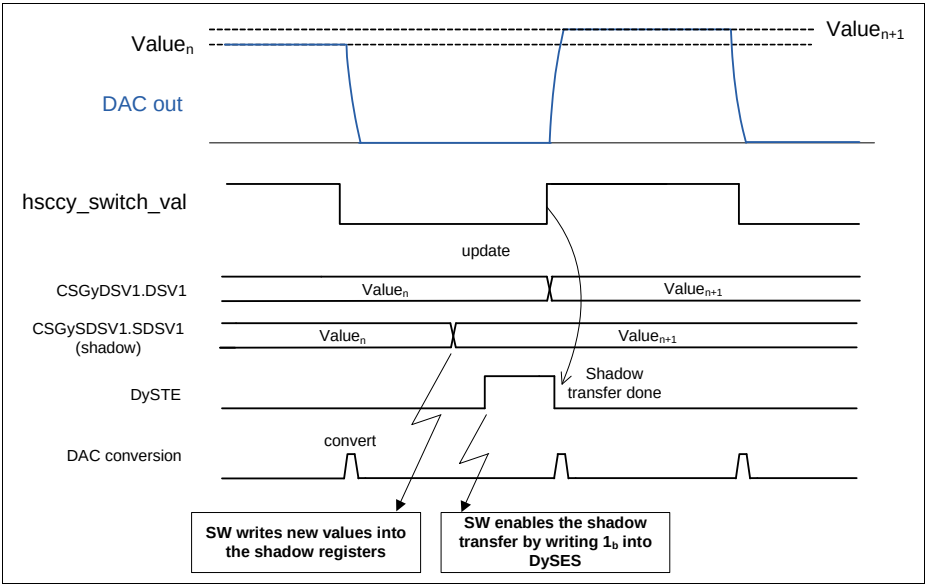


Figure 20-63 Shadow transfer timing - SCM = 00_B

When the slope generation is enabled (not is static mode), the shadow update of the values is always done before the beginning of a new generation cycle.

When an incrementing or decrementing slope is being generated, the shadow update occurs before the load and consequent conversion of the **CSGyDSV1** value, **Figure 20-64**.

When a triangular slope is being generated, the shadow update still occurs when the a generation cycle is over. This means that the update is performed when the slope transits from decrementing to incrementing, **Figure 20-65**.

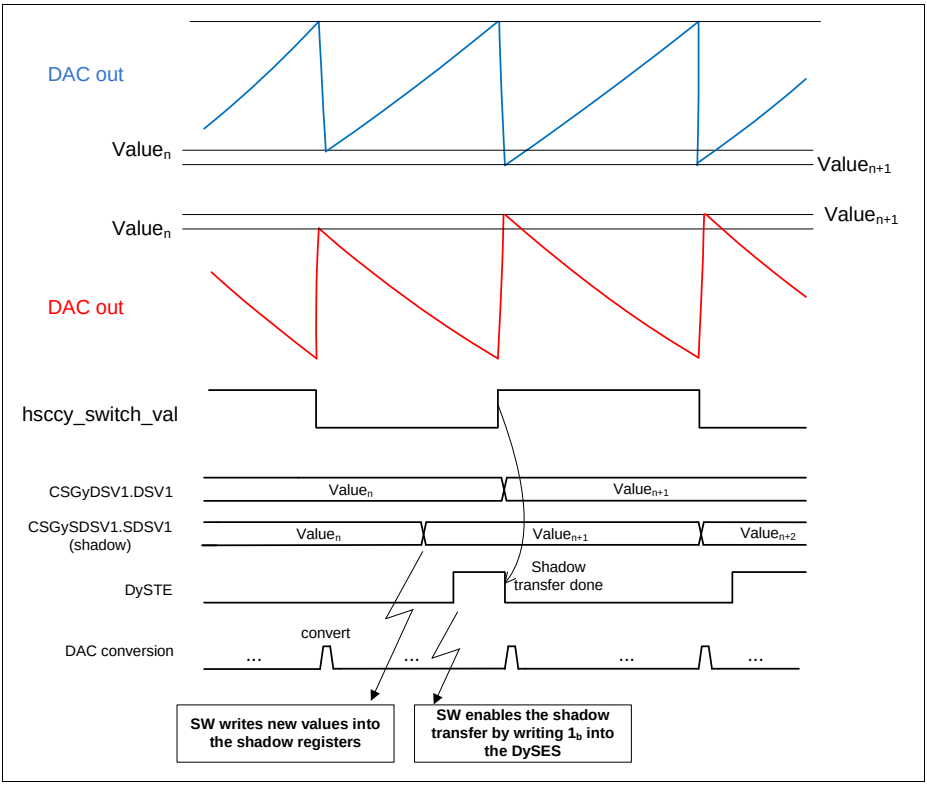


Figure 20-64 Shadow transfer timing - SCM = 01_B/10_B

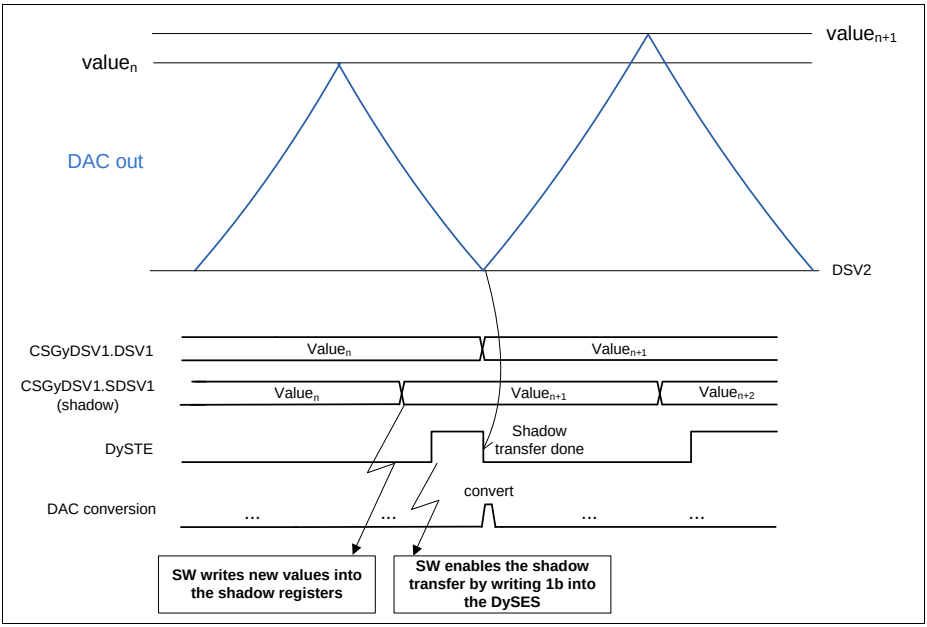


Figure 20-65 Shadow transfer timing - SCM = 11_B

An immediate shadow transfer operation can be enabled, when **CSGySC**.SCM = 00_B, by setting the bitfield **CSGySC**.IST = 1_B (if **CSGySC**.SCM is different from 00_B the value set on the **CSGySC**.IST is ignored).

The external trigger, hsccy_ste_en, can also be used to enable the shadow transfer. If this function is enabled, the shadow transfer trigger signal is only generated after the SW has set the **CSGTRSG**.DySTE bit and the proper transition on the external signal is detected.

The usage of an external signal for enabling the shadow transfer, is especially useful for performing fine tuning in topologies using n-phase converters, with individual phase current monitoring, without stressing the SW, **Figure 20-66** (the need for 3 independent ISR is avoided).

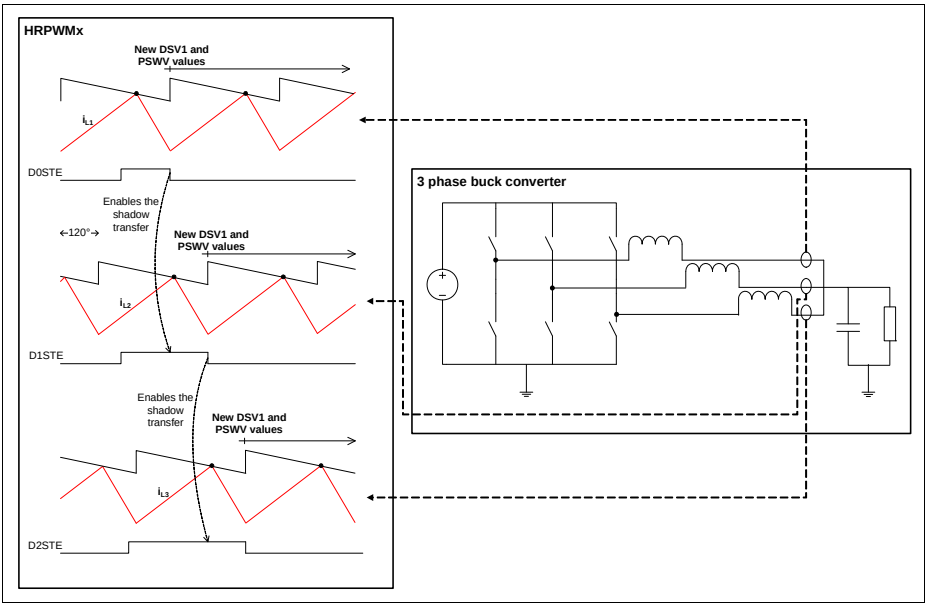


Figure 20-66 Three Phase Buck Converter shadow transfer

20.2.2.5 CMP Input Selector

Through the CMP Input Selector block the user has the possibility to select the inputs that are allocated to several functions, **Figure 20-67**.

From the CMP Input Selector, one can decode the following functions:

- External clamping of the comparator output
- External blanking period start
- Which analog input is connected to the Comparator

Clamping control

From the CMP Input Selector, it is possible to decode an external signal that sets the comparator output into clamped/passive level. This is a level signal, which means that while the signal is ACTIVE, the comparator output is gated to the passive level programmed into the **CSGyPLC**.PSL.

See **Section 20.2.2.6**, for a complete description of the features of the clamping control.

Blanking control

It is possible to use an external signal to start the blanking period. The external signal can be decoded from the set of inputs HRPWMx.BLy[P...A]. An edge detection logic is also present to give full flexibility on the connectivity.

The blanking unit can also be controlled internally via the comparator output. See **Section 20.2.2.6** for a complete description of the features of the blanking mode.

Analog input selection

The signal selected to act as blanking trigger, can also be used to switch between the two analog inputs (for the Comparator input). Due to the fact that the switch between these two analog inputs can be controlled dynamically by the HW, the external switching signal needs to be passed to the Input Mux Control Unit (it is inside this unit that the different modes that control the analog input multiplexer are handled).

See the Comparator chapter, **Section 20.2.2.6**, for a complete description of how to control which analog input is connected to the Comparator.

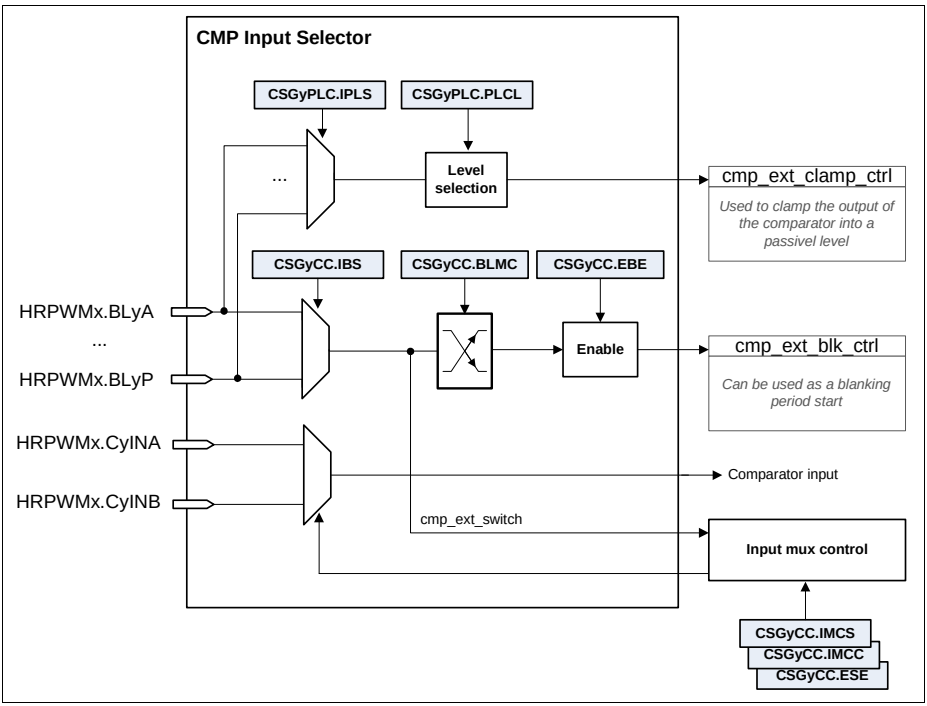


Figure 20-67 CMP Input Selector

20.2.2.6 Comparator Block (CMP)

Each CSGy unit contains a dedicated High Speed Comparator. Each comparator has a dedicated digital output, that can work in sync, blanking and filtered mode, see **Figure 20-68**.

The comparator unit also has also three different power modes:

- High Speed Mode
- Low Speed Mode
- Power OFF

*Note: For a full description of the power modes, please address **Section 20.5**.*

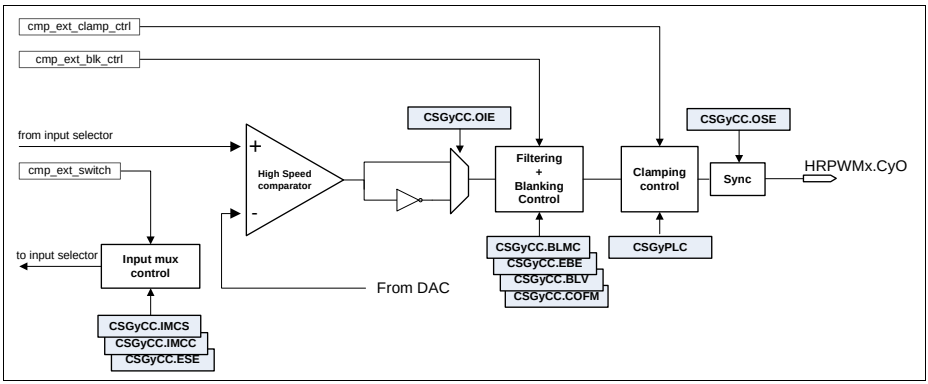


Figure 20-68 Comparator Block (CMP) overview

CMP Filtering & Blanking Control

It is possible to synchronize the output of the Comparator, by setting the field **CSGyCC**.OSE = 1_B. This is useful if the end block receiving this signal doesn't contain any input synchronizers or if this signal is distributed to several timer instances, avoiding with this some control loop jitter.

The output of the comparator can be the normal or inverted output, by setting the field **CSGyCC**.OIE accordingly.

The filtering block is placed after the comparator, making it possible to perform a small filtering operation for the comparator output. This imposes that the output of the comparator needs to be stable for a certain number of module clock cycles, so it can be passed to the output.

The filtering can be controlled by programming into the field **CSGyCC**.COFM, the wanted number of clock cycles for the filtering stage, **Table 20-14**.

Table 20-14 Comparator filtering modes

CSGyCC.COFM	Filtering window
0000_B	2 clock cycles filtering
0001_B	3 clock cycles filtering
0010_B	4 clock cycles filtering
0011_B	5 clock cycles filtering
...	...
1100_B	14 clock cycles filtering
1101_B	15 clock cycles filtering
1110_B	16 clock cycles filtering
1111_B	32 clock cycles filtering

Additional filtering options are available through the **CSGyCC**.COFC field. Through this field it is possible to select if the filtering operation is always performed (default value), only done when the DAC input is equal to the **CSGyDSV1** or when it is equal to **CSGyDSV2**.

This additional filtering options are only available when we have a static or hybrid mode programmed in the DAC Control Unit. In any of the other modes, the usage of the additional filtering options is not needed, due to the fact that the Comparator unit is only being used to generate the SET or the CLEAR for the PWM signal and never both.

It is also possible to program the comparator to work in a leading edge blanking mode (after the filter stage). This blanking mode is especially useful when the comparator is controlling the set/clear of the output latch present in the HRCy unit. This can avoid a premature shut down due to a glitch on the sensed induction current, see **Figure 20-69**.

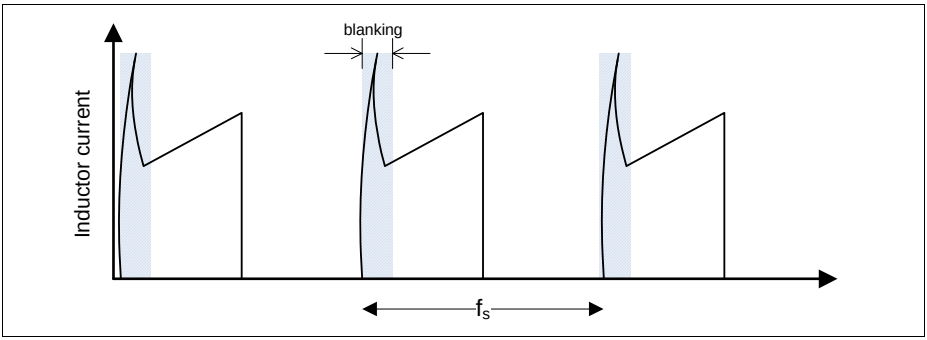


Figure 20-69 Comparator Blanking mode

The leading edge blanking start trigger can be the output of the comparator or an external signal (normally this is used when the comparator is used to clear the PWM signal while a timer is used to set the PWM signal).

To use an external signal as the leading edge for the blanking, one must set **CSGyCC**.EBE = 1_B, **Figure 20-70**. If this bitfield is 0_B, then the output of the comparator itself is used.

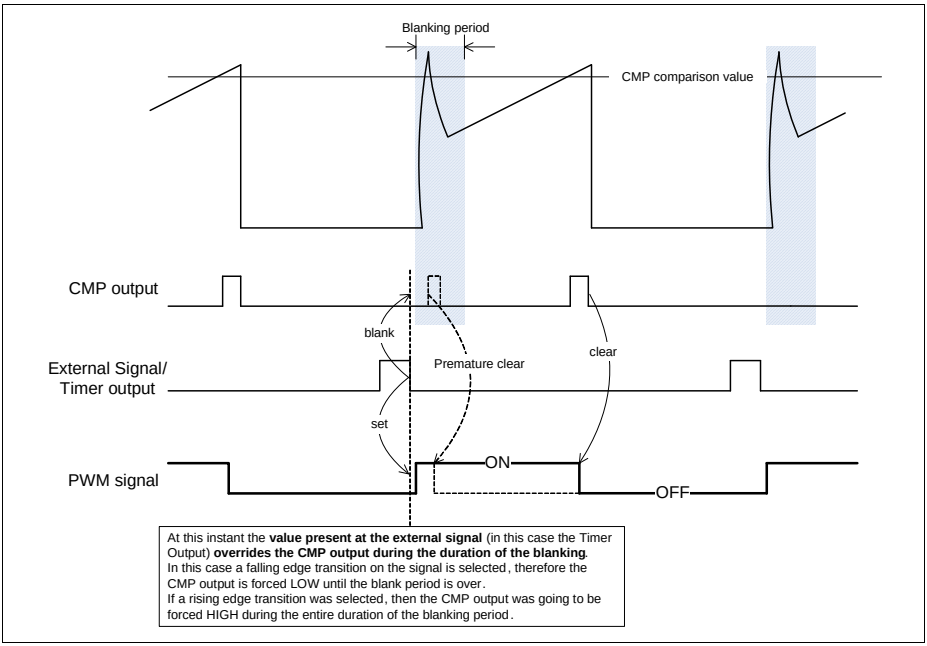


Figure 20-70 Comparator blanking using an external signal - EBE = 1_B, BLMC = 10_B

The blanking time is controlled via an 8 bit counter running on the module clock frequency. The blanking period value is set on the **CSGyBLV** register.

Updating the **CSGyBLV** during operation with a smaller value than the current one, stops immediately the blanking period. Programming it with a bigger value will extend the blanking period untill the new value is reached.

The value programmed into the **CSGyCC**.BLMC controls which is the leading edge for the blanking, from the comparator output or from the external signal, from LOW to HIGH, HIGH to LOW or both, see **Table 20-15**.

Table 20-15 Comparator blanking modes

CSGyCC.BLMC	Blanking mode
00_B	No blanking
01_B	Blanking after a LOW to HIGH transition
10_B	Blanking after a HIGH to LOW transition
11_B	Blanking after both transitions

Clamping Control

Each comparator has an clamping control stage that enables a programmable clamping action for the output. The output of the comparator can be set to a clamped level via SW (**CSGSETG**.SCyP) or via an external signal (from the HRPWMx.BLy[P...A]). The passive or clamping level (the value that is forced at the CSGy output) is defined via the **CSGyPLC**.PSL field, **Figure 20-71**.

While the **CSGSTATG**.PSLSy field is set to 1_B, the output of the comparator is always clamped to the value programmed into the **CSGyPLC**.PSL.

The bitfield **CSGyPLC**.PLSW indicates if the exit from the clamped state is controlled completely via SW or can also be controlled with the external signal.

Setting the **CSGyPLC**.PLSW to 1_B, imposes that the exit from the clamped state is only possible via SW (SW must clear the PSLSy field). If this bitfield is set to 0_B then the output exits from the clamped state, whenever the external signal (cmp_ext_clamp_ctrl) passes from active to inactive.

The active state for the external signal controlling the clamping (cmp_ext_clamp_ctrl) is programmable via the **CSGyPLC**.PLCL field.

When to enter and exit the Clamped state can be configured via the **CSGyPLC**.PLEC and **CSGyPLC**.PLXC fields. In default state, the clamped state is entered and exit immediately, nevertheless, these actions can be delayed until a certain action on the comparator output is sensed (e.g. output of the comparator passing from high to low or vice versa).

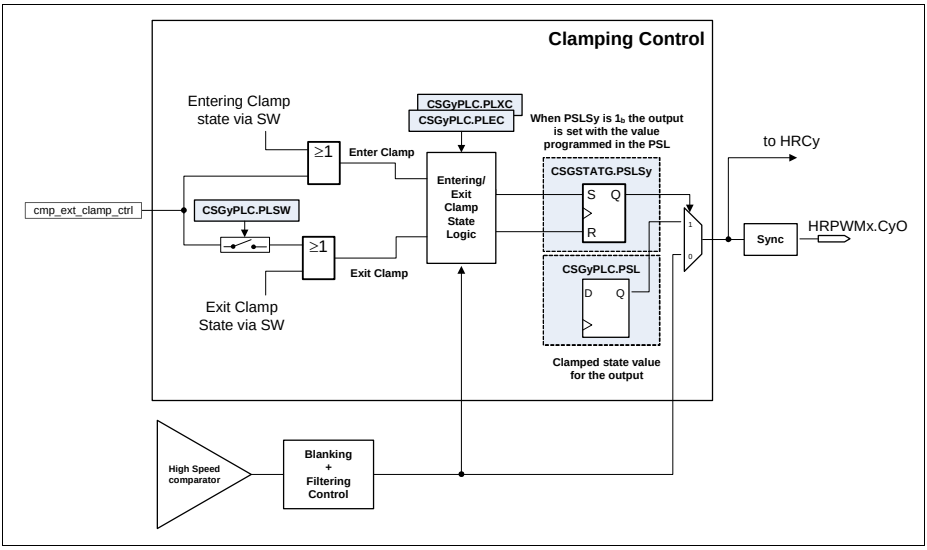


Figure 20-71 CMP clamping control

Input Multiplexer Control

The user has the possibility to select which signal connected to the Comparator input from the two available analog input signals.

In a straightforward usage of the comparator, the selection of which analog input is connected to the Comparator is controlled via the **CSGyCC**.IMCS field. This is a static value and cannot be update on-the-fly with guarantee that no wrong operation will be triggered.

When the Slope Control Unit is programmed into hybrid mode, where the two reference values are used, **CSGyDSV1** and **CSGyDSV2**, it is also needed to switch dynamically which analog signal is connected to the comparator.

This type of operation is useful when the comparator is being used for sensing the peak current to clear the output latch on the HRCy, and is also used to sense the voltage over the switch to set the latch, see **Section 20.1.3** for the description of the operation mode.

It is possible to dynamically switch the analog input that is connected to the comparator, with the comparator output (after the blanking block - default configuration) or with the external signal mapped for the blanking function (cmp_ext_switch in **Figure 20-67**). This mode is not the default and needs to be enabled by setting the field **CSGyCC**.ESE = 1_B.

The switching mode is controlled via the **CSGyCC**.IMCC field, **Table 20-16**.

Table 20-16 Comparator input switching configuration

CSGyCC.IMCC	Mode
00_B	Switching disable
01_B	Comparator input is connected to HRPWMx.CyINB when the control signal is HIGH
10_B	Comparator input is connected to HRPWMx.CyINA when the control signal is HIGH
11_B	Reserved

The switching between the two analog inputs is done dynamically every time that a transition on the control signal is detected. Like previously mentioned, this control signal can be the comparator output after the blanking logic, or the external signal used for the blanking (If **CSGyCC**.ESE = 0_B the comparator output is selected. If **CSGyCC**.ESE = 1_B then an external signal is used - the signal can be selected via the **CSGyCC**.BLMC field).

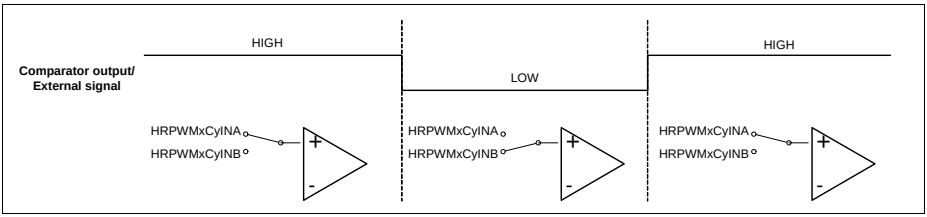


Figure 20-72 Dynamic input switching - IMCC = 10_B

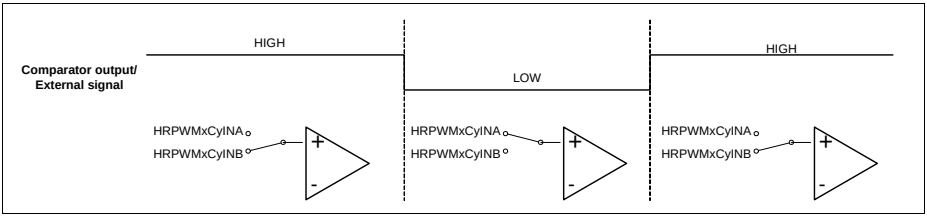


Figure 20-73 Dynamic input switching - IMCC = 01_B

Note: Selecting the external signal to control the input switching doesn't imply that this signal is also used for the blanking mechanism and vice-versa.

To avoid any wrong startup definitions between the DAC Control unit initial value, the initial value of the PWM signal and the signal connected to the comparator, the dedicated run bit for the comparator also controls the initial state of the switch, **Figure 20-74**.

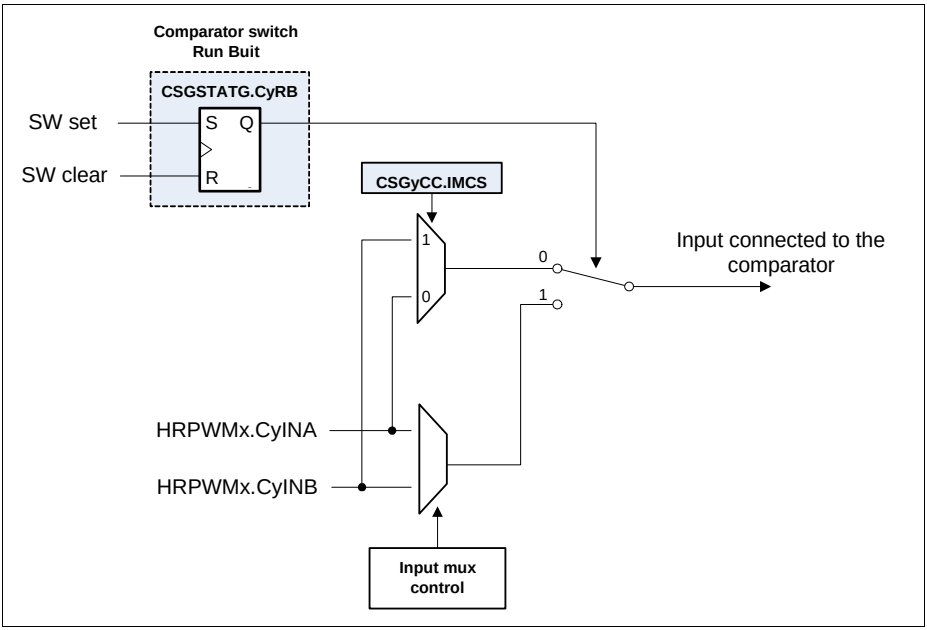


Figure 20-74 Dynamic input switching run bit

20.2.3 High Resolution Channel (HRCy)

Each HRPWM module contains a set of 4 High Resolution channel extensions. These extensions can be linked together with one or more specific Capture/Compare Unit Timer Slices.

The structure of the linking between the Timer Slice and the high resolution extension can be seen at **Figure 20-75**. Even when the high resolution extension is being used, the normal PWM outputs from the Capture/Compare unit are still available.

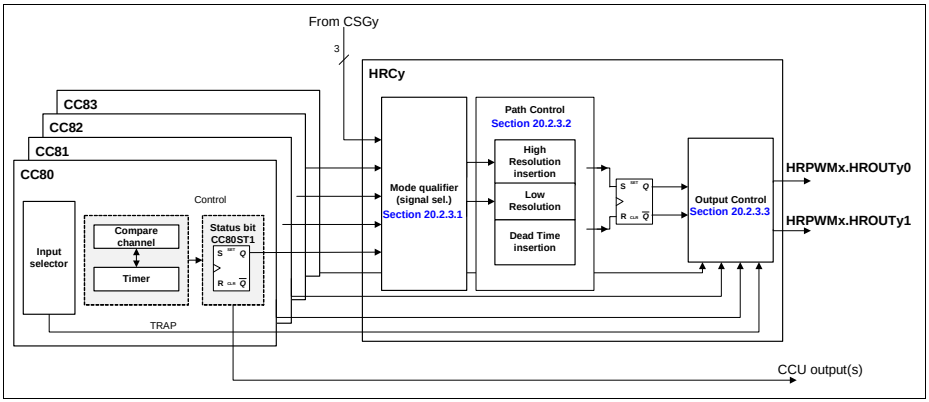


Figure 20-75 HRCy link with Capture/Compare Unit

Present inside each HRCy unit:

- Mode Qualifier block
- Path Control block
- Output Control block

The Mode Qualifier handles the different set and clear sources for the PWM signal, dependent on the programmed mode configuration, **Section 20.2.3.1**. The set and clear sources can have its origin on the Capture/Compare Timer Slice or in the CSGy unit.

The Path Control block, generates the high resolution signal and also controls the dead time generation for the two complementary output signals, **Section 20.2.3.2**.

The Output Control block, handles several functions passed from the Timer Slice, that can be used to set the output PWM signal into PASSIVE or ACTIVE state, **Section 20.2.3.3**. The output control function, TRAP (see the specific description for this features the Capture/Compare Unit chapter) can be also used for the HRCy.

A dual control scheme of the output PWM signal is implemented inside each HRCy unit. This dual control technique enables the control of the PWM signal, via two paths:

- a High Resolution Path
- a Low Resolution Path

These two paths can work in parallel and it is also possible to switch on-the-fly the signals linked to each of the two paths, **Figure 20-76**. Please address the following sections to understand the behavior of this dual control technique.

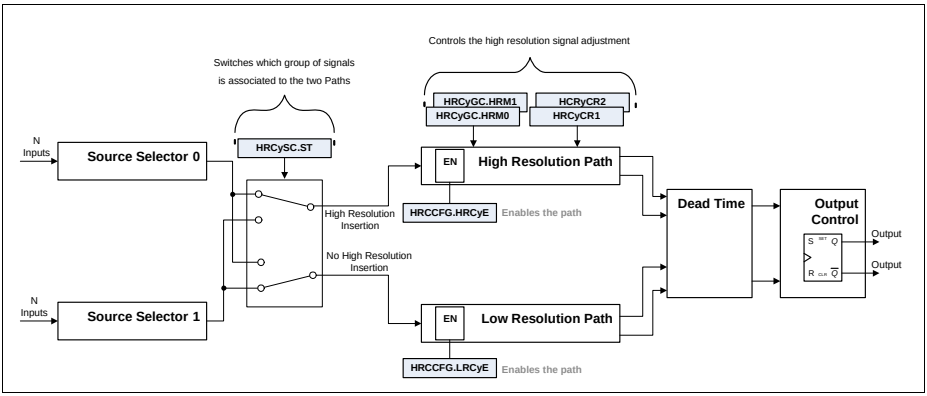


Figure 20-76 HRCy dual control scheme

HRCy Connection Scheme

Figure 20-77 shows the connection scheme of each HRCy subunit. For a description of each mentioned signals, please address the specific section.

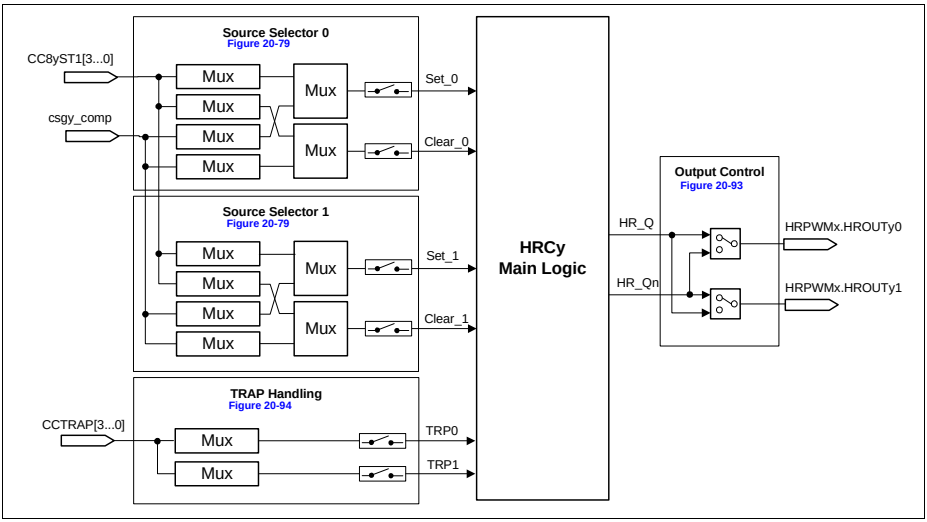


Figure 20-77 HRCy connection scheme

20.2.3.1 Mode Qualifier

The Mode Qualifier block is used to decode the set and clear sources for the output latch. The set and clear sources can come from the CSGy unit or from a Timer Slice.

The group of signals used to decode a set or clear for the output latch, will dictate at the end, the type of control loop that is used to handle the PWM output, **Figure 20-78**.

It is possible to decode two trigger sources for the set and clear signal of the output latch. This means, that one can have up to 2 set and 2 clear signals for the output latch.

This can be used for different situations:

- CSGy is used to control the PWM in normal conditions and a Timer Slice can be used to dictated the absolute maximum ON/OFF time, to overcome some wrong operation.
- To enable a control via two separate CSGy units. Where one is being used to monitor the inductor current and the other is being used to monitor some system error conditions.
- To enable a built-in control for very low load situations (without reconfigure the system): e.g., generate a pulse on the PWM signal in each 10 ms with a Timer Slice when the CSGy unit is not working.
- To migrate from Continuous Conduction Mode to Discontinuous Conduction Mode, without performing any extra configuration (one Comparator + Timer Slice used for Continuous Conduction Mode and another Timer Slice for Discontinuous Conduction Mode).
- To control the PWM signal via the Low Resolution Path under normal conditions and having the High Resolution Path, pre programmed with specific high resolution insertion values. The High Resolution Path can then be used on the fly, whenever the associated inputs start to be generated, without reconfigure the module.

Note: The above applications, serve merely as an example and should not be understood as the only possibilities for the function.

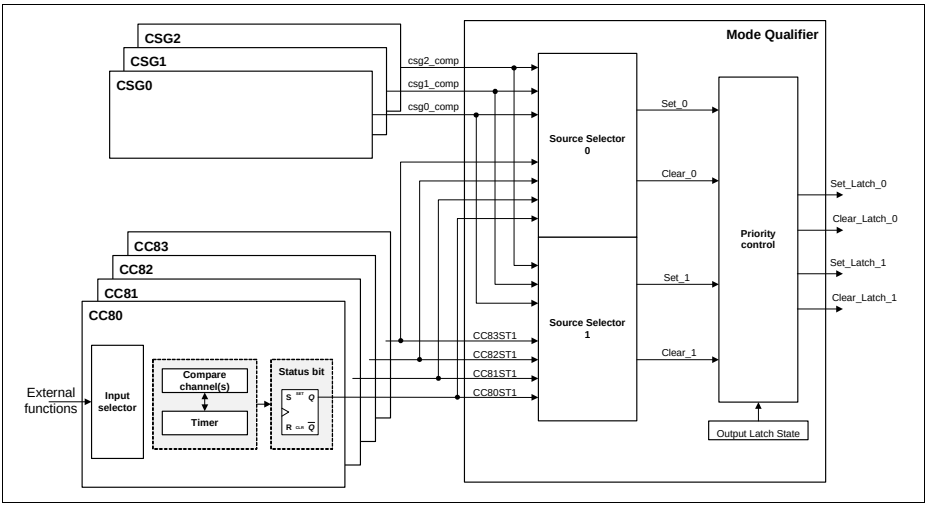


Figure 20-78 Mode Qualifier overview

The signals decoded from the source selector 0, have always priority over the ones decoded from the source selector 1. This procedure is handled inside the priority control block, **Figure 20-79**.

A set for the latch is only decoded if the output latch is in the PASSIVE state, while a clear is only decoded when the latch is ACTIVE.

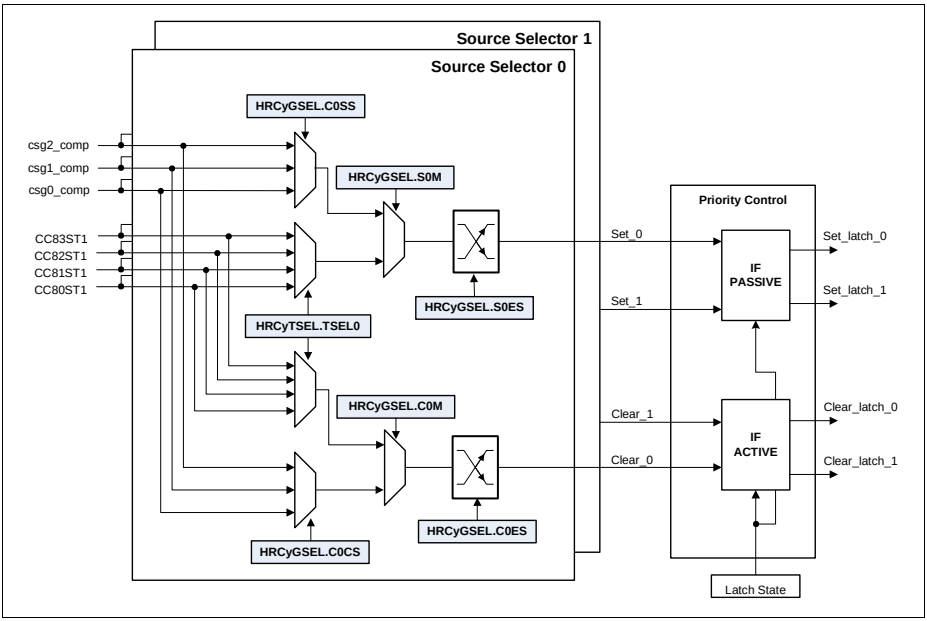


Figure 20-79 Mode qualifier logic

20.2.3.2 Path Control

This block handles the decoding of the signals into two separate paths, the High Resolution and the Low Resolution Path, and also handles the dead time insertion, **Figure 20-80**.

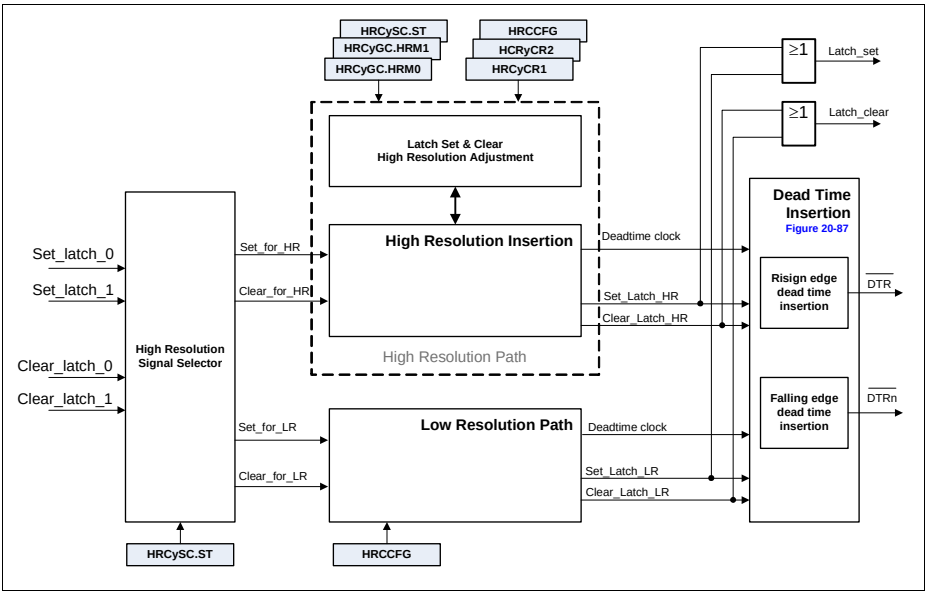


Figure 20-80 Path Control overview

High and Low Resolution Signal Path

The selection of which signal group (from Source Selector 0 or Source Selector 1) is connected to the high resolution path or the low resolution path, is handled inside the High Resolution Signal Selector block.

This selection is linked to the **HRCySC**.ST bitfield. If this bitfield is 0_B, then the signals from Source Selector 0 are linked to the High Resolution Path and the signals from the Source Selector 1 are linked to the Low Resolution Path. If this bitfield is 1_B, then we have the opposite condition, **Figure 20-81**.

The bitfield **HRCySC**.ST as an associated shadow register, **HRCySSC**.SST, that enables the switching of which group is connected to the High Resolution Path (and consequently the group that is connected to the Low Resolution Path). Please address **Section 20.2.3.4** for a complete description of how this update is handled.

Each of the paths as an associated enable bit: **HRCCFG**.HRCyE for the High Resolution Path and **HRCCFG**.LRCyE for the Low Resolution Path.

Note: Enabling/disabling on-the-fly the two available PWM generation paths, over the HRCyE and LRCyE bitfields, is not supported by the HW (wrong operation can occur).

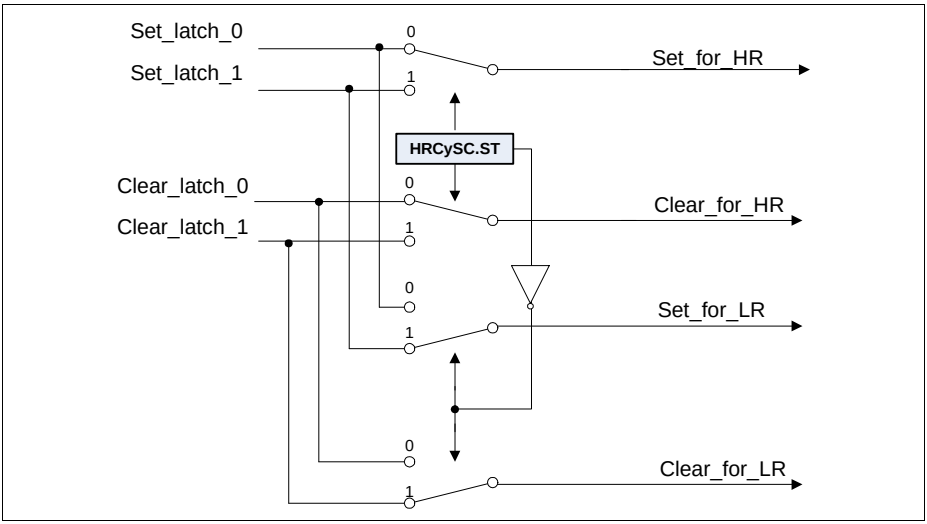


Figure 20-81 High resolution signal selector

If the High Resolution Path is enabled, **HRCCFG**.HRCyE = 1_B, the adjustment for the set and clear (within the High Resolution Path) is decoded depending on the value programmed into the **HRCyCR1** and **HRCyCR2** fields.

These two values are an eight bit control field that select the amount of 150 ps shifts, that are going to inserted into the output PWM signal.

The **HRCyCR1** is linked with the rising edge of the PWM signal while the **HRCyCR2** controls the delay for the falling edge of the PWM signal, **Figure 20-80**.

The final latch set and clear signals is a combination between the signals generated from the two paths (High Resolution and Low Resolution Path). This means that the two path functionality is not mutually exclusive.

If dead time was enabled, the proper clock for the dead time counter needs to be decoded. Therefore, each of the signal generation paths, outputs the associated dead time clock.

The Low Resolution Path, only generates one dead time counter clock, due to the fact that the signals generated via this path are always aligned with the reference clock (no high resolution adjustment).

The High Resolution Path, generates two dead time counter clocks. One clock aligned with the High Resolution Set and another aligned with the High Resolution Clear signal, **Figure 20-82**.

The decoding of which clock is used at which instant is automatically done inside the Dead Time Insertion block.

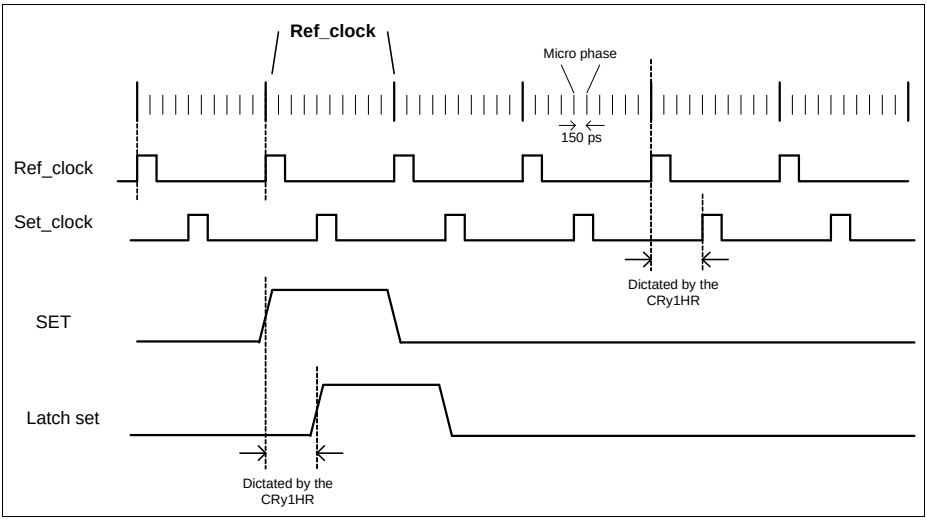


Figure 20-82 High resolution SET generation

The **HRCyGC**.HRM0 and **HRCyGC**.HRM1 fields, control the type of high resolution positioning for the PWM output signal, when this one is generated via the Source Selector 0 and Source Selector 1, respectively. The **HRCyGC**.HRM0 and **HRCyGC**.HRM1 fields are only taking into consideration inside the High Resolution Path, **Figure 20-83**.

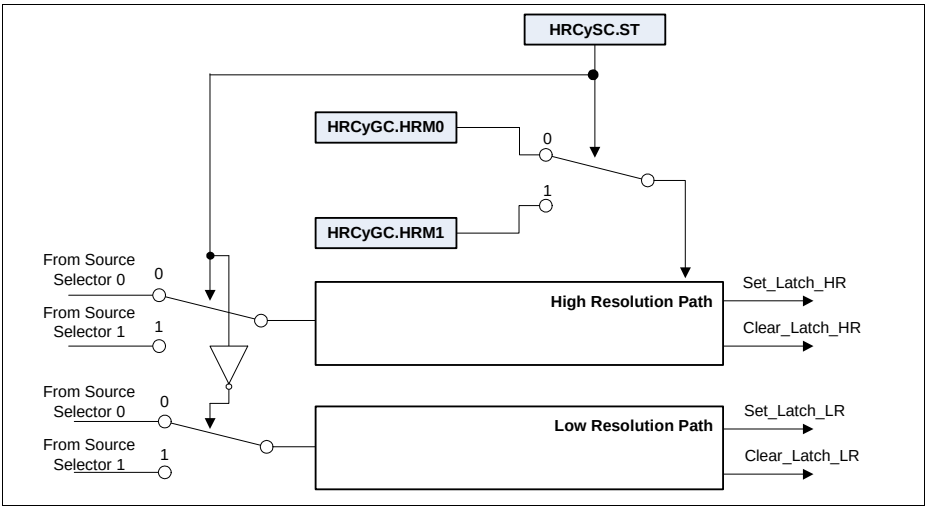


Figure 20-83 HRMn field usage inside the High Resolution Path

Figure 20-84 to **Figure 20-86** shows the different modes for the high resolution signal generation:

- high resolution rising edge PWM positioning: **HRCyGC**.HRMn = 00_B
- high resolution falling edge PWM positioning: **HRCyGC**.HRMn = 01_B
- high resolution both edges PWM positioning: **HRCyGC**.HRMyn= 10_B
- no high resolution positioning: **HRCyGC**.HRMn = 11_B. High Resolution Path is not performing any high resolution adjustment of the output PWM signal. This means that the High Resolution Path is generating signals with low resolution.

The fine adjustment of the PWM with high resolution is suitable when using origin signals that contain timing information. This is true for all the signals generated via a Capture/Compare Unit Timer Slice, but is not true for signals generated via the CSGy unit. Nevertheless this does not impose that the signals that are originated via the CSGy unit cannot be adjusted with high resolution precision.

Note: In the following examples, CC8yCRy are the compare registers from the specific Capture/Compare Unit Timer Slice, that dictate the coarse timing for the PWM signal.

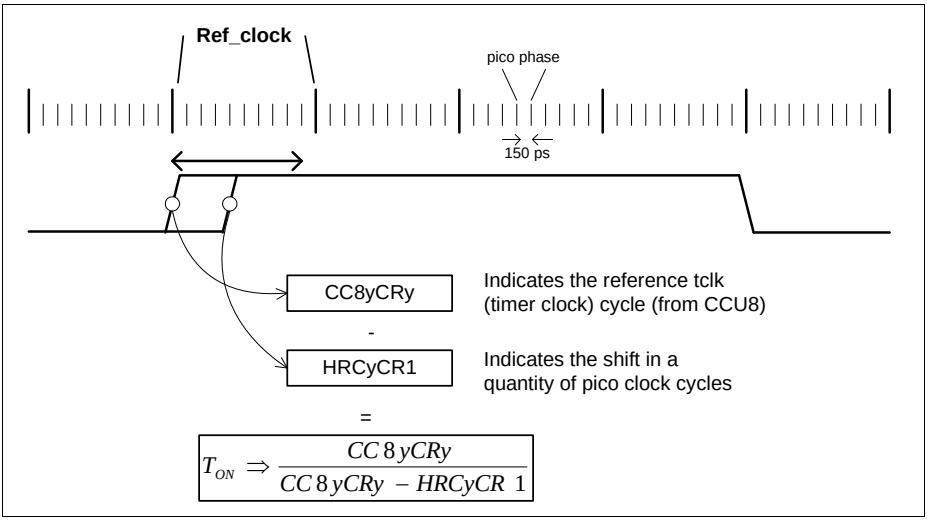


Figure 20-84 Rising edge adjustment

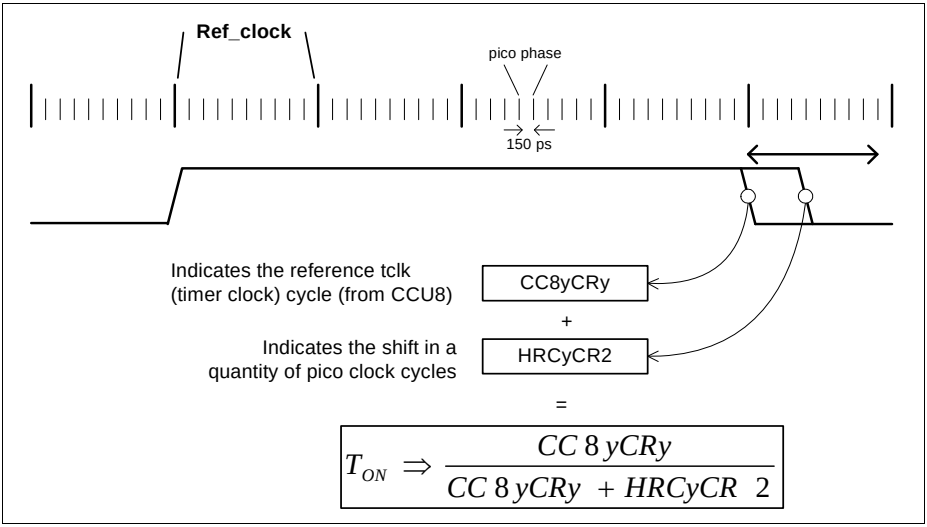


Figure 20-85 Falling edge adjustment

The mode where both edges can be finely placed is more indicated for phase shift topologies. Therefore it is suitable to use this mode when two compare channels are available on the Timer Slice (only Capture/Compare Unit 8 Timer Slices have two

compare channels). In this mode one can have two completely different references for rising and falling for the coarse positioning. This coarse positioning can then be finely tuned with the two values for the high resolution adjustment, **Figure 20-86**.

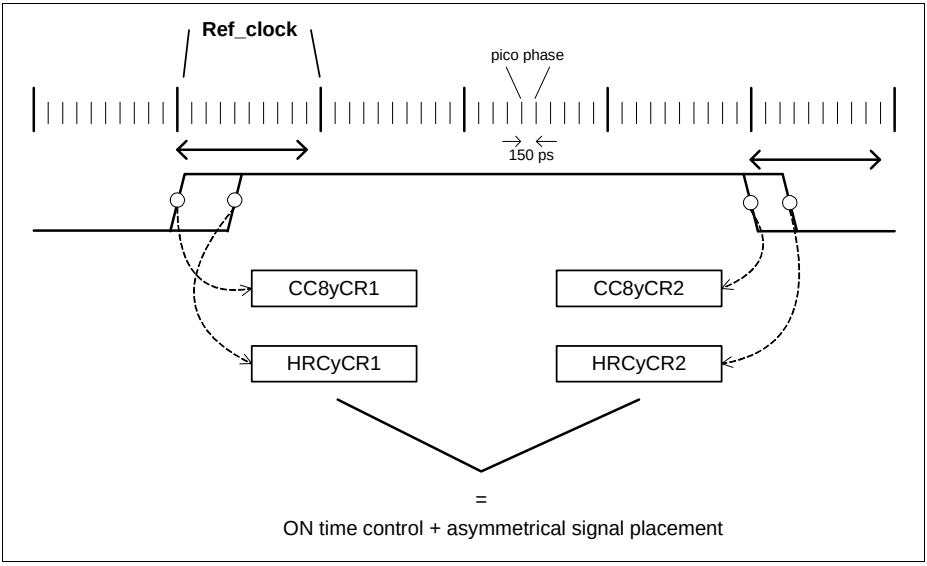


Figure 20-86 Both edges adjustment - asymmetric

Dead Time Insertion

Each HRCy has a dedicated dead time insertion block. For each dead time block, the user has two programmable fields that are used to control the rising and falling dead time value, **Figure 20-87**.

The value programmed into **HRCyDCR**.DTRV indicates the value for the dead time when the latch has a transition from PASSIVE to ACTIVE. The **HRCyDCF**.DTFV is used for the transition from ACTIVE to PASSIVE.

The resolution of the dead time counter is always linked with the reference clock, which means that there is no high resolution in the dead time counter itself.

Attention: To have a dead time value of 0, the field HRCyGC.DTE needs to be set to 0. The minimum dead time insertion is 2 module clock cycles.

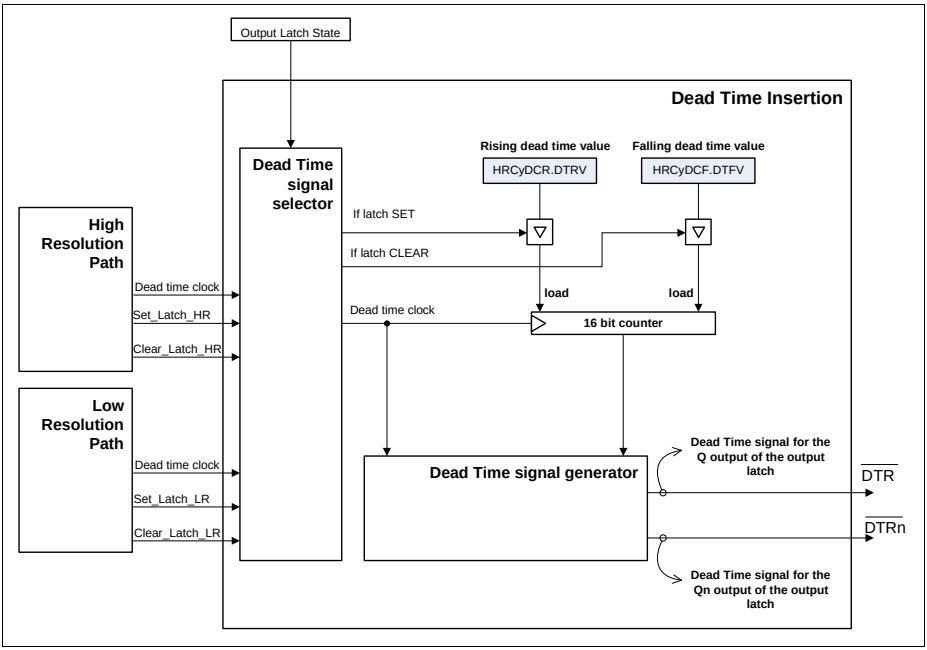


Figure 20-87 Dead Time overview

The signals generated from the dead time insertion block are going to be used in the output control to gate the output of the latch (one for each of the two complementary outputs).

Figure 20-88 and **Figure 20-89** show the alignment of the dead time counter with the high resolution, for the set and clear operations of the latch (PWM signal).

Attention: The independent dead time unit inside the Capture/Compare Unit 8 cannot be used to insert dead time at the outputs of the HRPWM block.

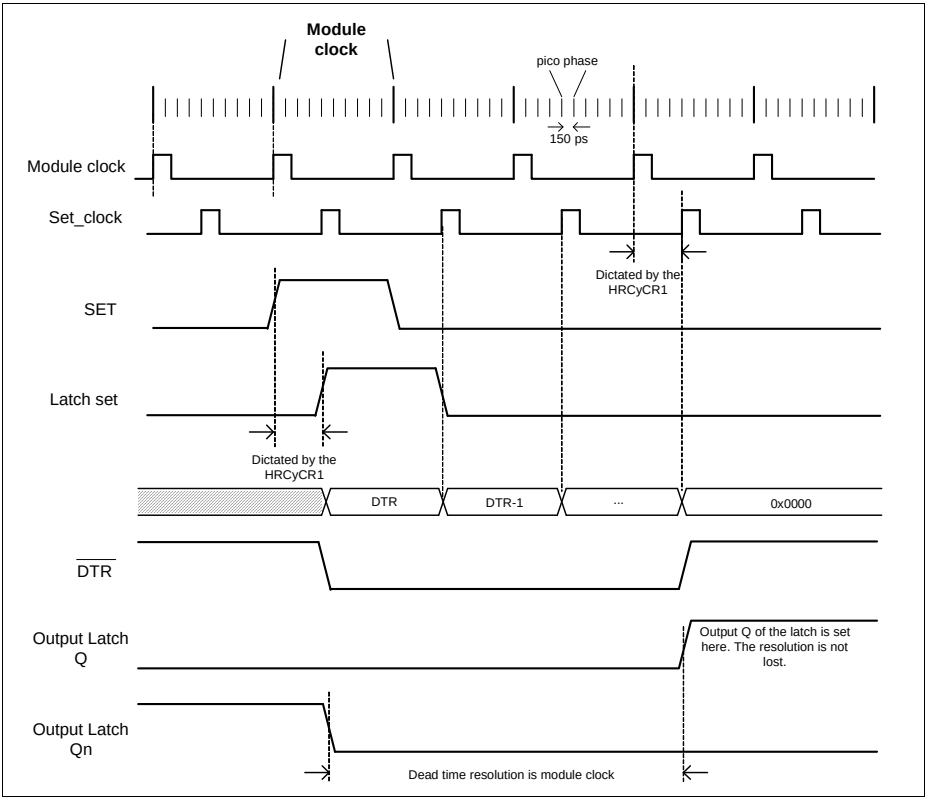


Figure 20-88 Dead time generation - Rise time

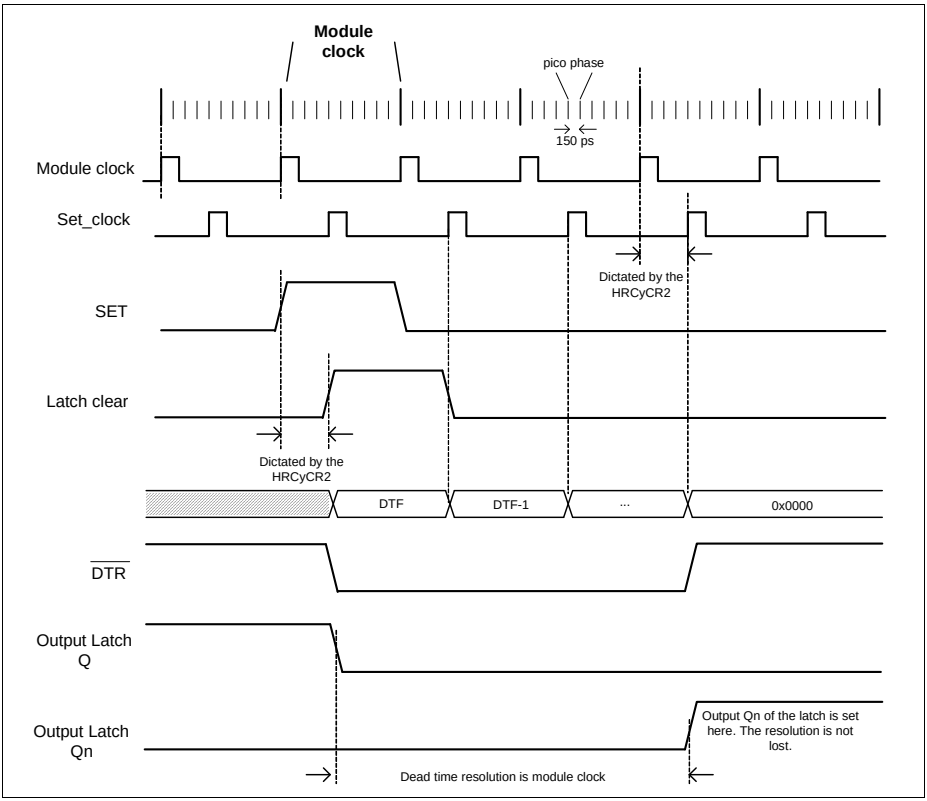


Figure 20-89 Dead time generation - Fall time

Figure 20-90 and **Figure 20-91** show the effect of the high resolution adjustment on the dead time generation, when the PWM signal is being generated by specific Capture/Compare Unit Timer, in Edge and Center Aligned Mode.

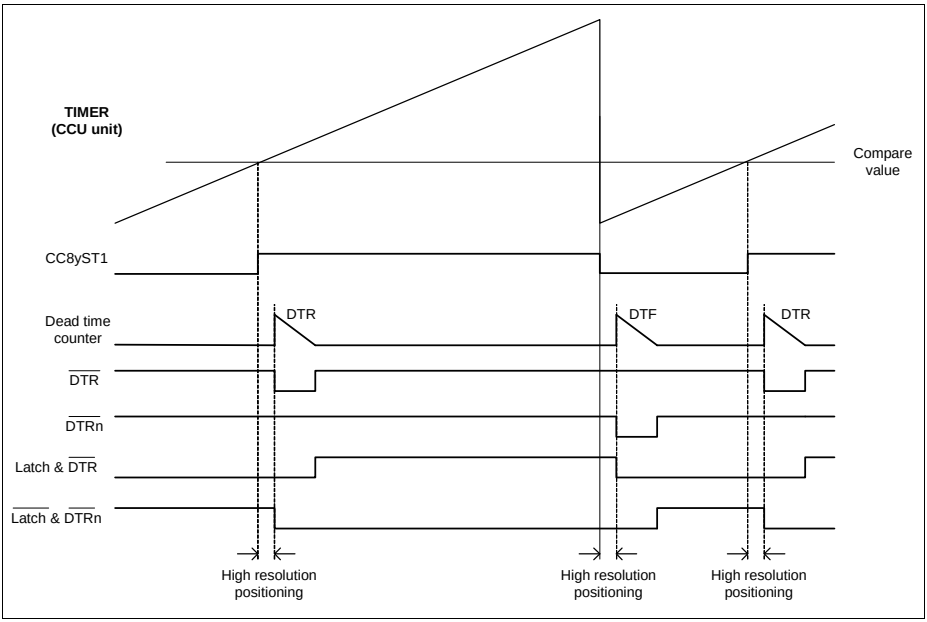


Figure 20-90 PWM with dead time generation - Edge Aligned Mode

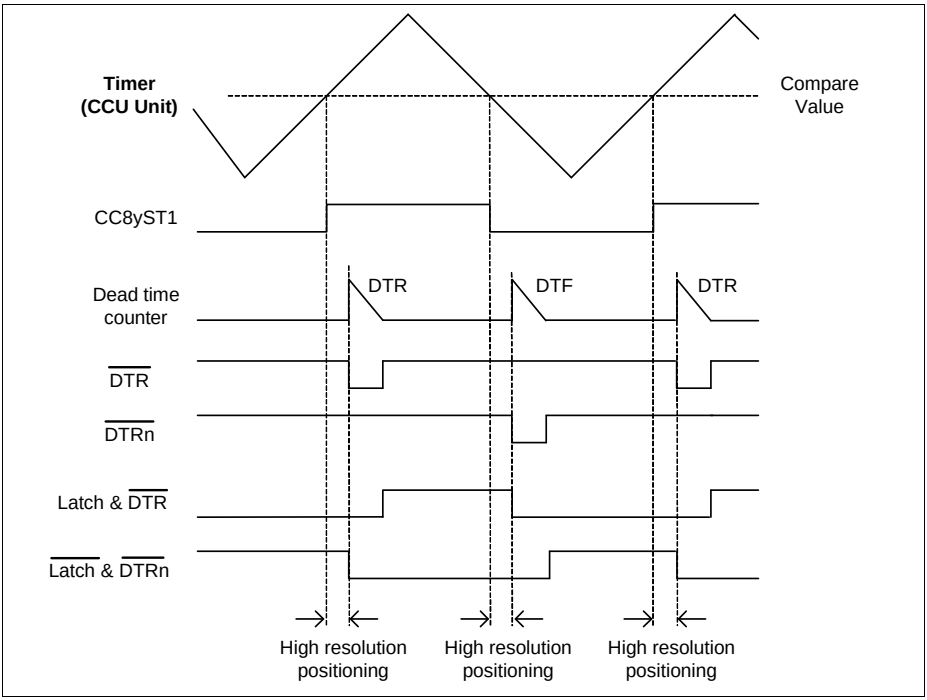


Figure 20-91 PWM with dead time generation - Center Aligned Mode

20.2.3.3 Output Control

The last block on the signal path is used to configure the ACTIVE and PASSIVE levels for the two output PWM signals. It is also inside this block that the TRAP signal is handled, **Figure 20-92**. This dedicated TRAP signal can be used to clamp both PWM outputs (to a passive state) whenever a fault condition occurs. This TRAP signal is fed from the associated Capture/Compare Unit.

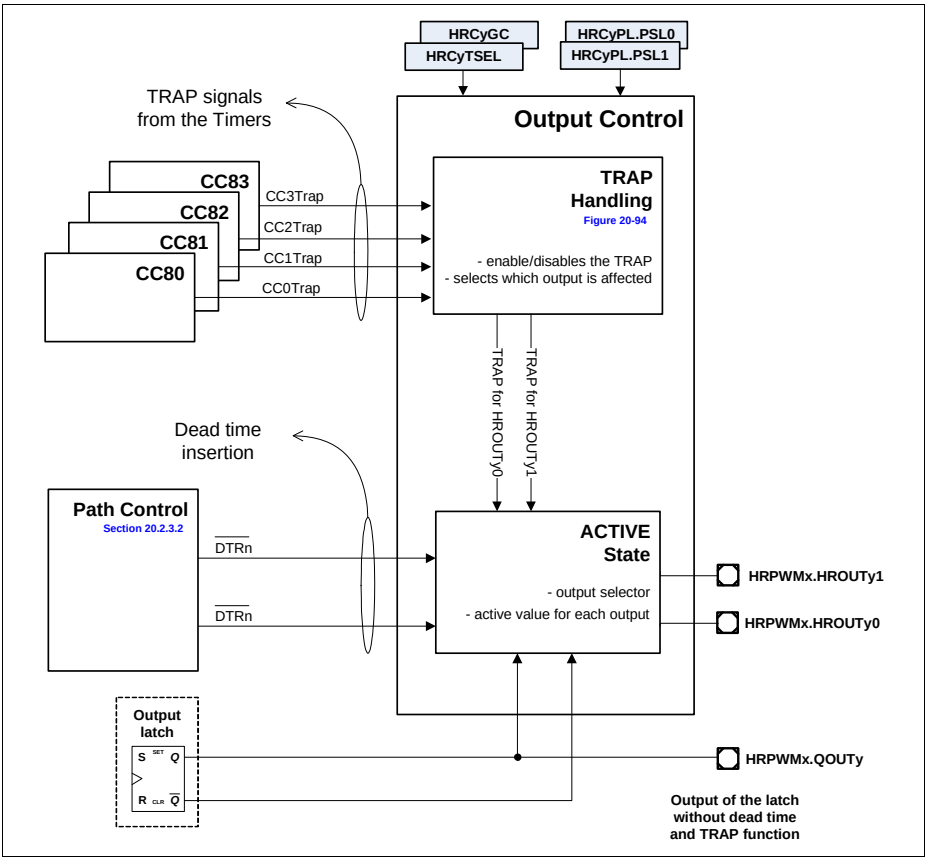


Figure 20-92 Output control overview

Like in the Capture/Compare Unit, it is possible to select the ACTIVE state for each output. This is done by programming **HRCyPL**.PSL0 and **HRCyPL**.PSL1 bitfields accordingly. The PSL0 field controls the ACTIVE state of the HRPWMx.HROUTy0 output while PSL1 controls the ACTIVE state of the HRPWMx.HROUTy1, **Figure 20-93**.

The state of both outputs after the block initialization is always the PASSIVE level dictated by the two bitfields, **HRCyPL**.PSL0 and **HRCyPL**.PSL1.

The selection of which signal from the output latch (Q or Qn) is connected to the two PWM outputs, HRPWMx.HROUTy0 and HRPWMx.HROUTy1 can be selected via the **HRCyGC**.OCS0 and **HRCyGC**.OCS1, respectively. The default configuration of these two fields, will drive a pair of complementary signals via the HRPWMx.HROUTy0 and HRPWMx.HROUTy1 outputs.

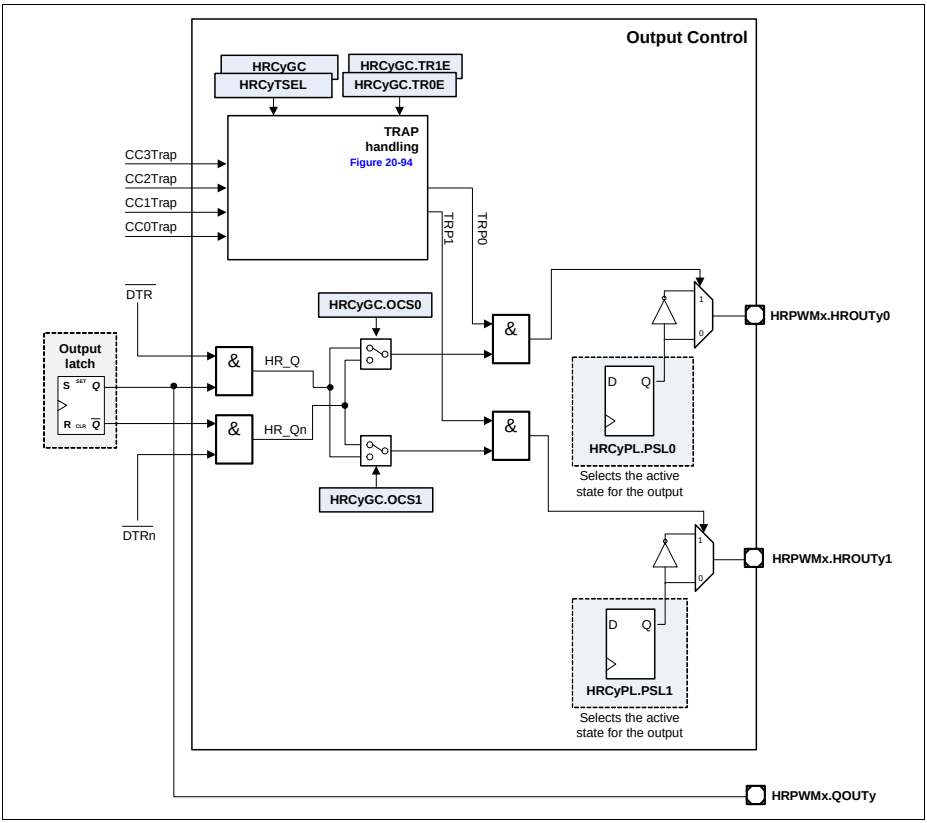


Figure 20-93 Output control logic

Each HRCy contains a dedicated TRAP handling mechanism that can be used to set the high resolution outputs in a predefined Passive level.

The internal TRAP control signal, can be generated from two different sources:

- Trap signal from the Capture/Compare Unit timer slice linked with the Source Selector 0
- Trap signal from the Capture/Compare Unit timer slice linked with the Source Selector 1

The TRAP signal coming from the Capture/Compare Timer can be used whenever the specific timer is selected inside the Source Selectors, **Figure 20-94**.

For a complete description of the Capture/Compare Unit TRAP detection and modes, please refer to the specific chapter inside the Capture/Compare Unit module.

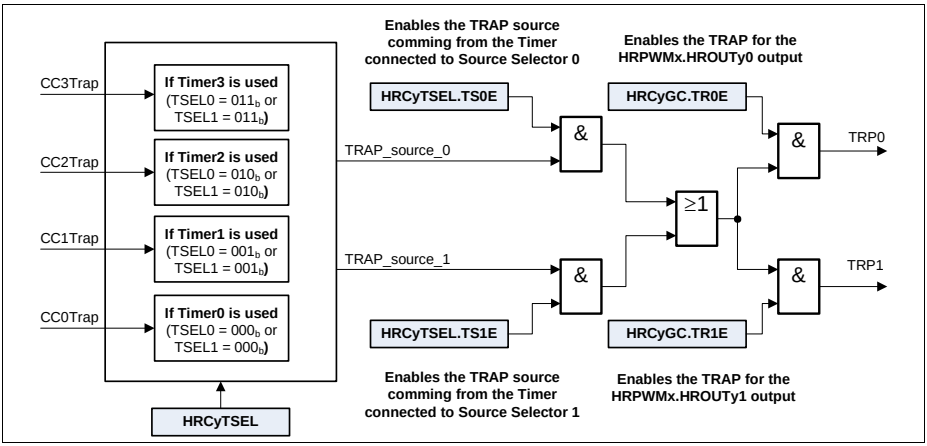


Figure 20-94 Trap handling logic

20.2.3.4 Shadow Transfer Control

Each HRCy unit has a shadow transfer mechanism that can be used to update the high resolution compare values, **HRCyCR1** and **HRCyCR2**, the dead time values, **HRCyDCR** and **HRCyDCF**, and the field that dictates which source selector is connected to the high resolution path, **HRCySC**, on-the-fly **Figure 20-95**.

The update of the high resolution compare values is needed to perform a cycle-by-cycle adjustment of the ON/OFF time of the PWM signal or the phase shift (that at the end control the amount of energy that is transmitted to the load).

The update of **HRCySC** register is needed every time that the software needs to switch the controll of the high resolution path, from the Source Selector 0 to Source Selector 1 and vice versa. The usage overview of this register is visible on **Figure 20-76**.

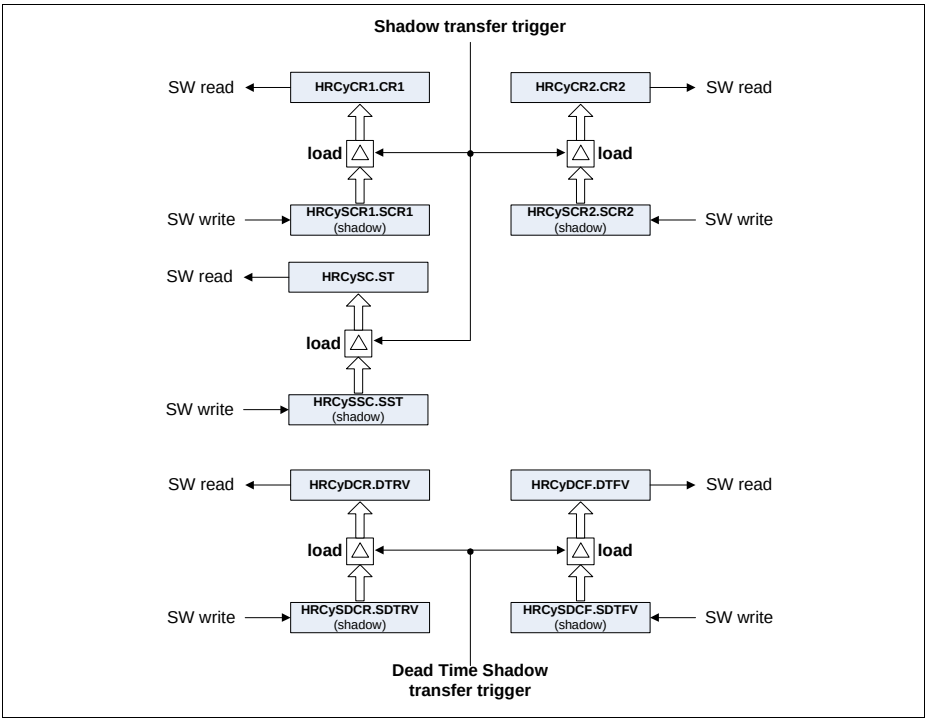


Figure 20-95 HRCy shadow transfer overview

The update of the dead time values is useful for applications that use:

- dead time adjustment dependent on the load conditions
- dead time adjustment dependent on temperature variations on the switches
- constant dead time compensation to minimize the distortion

Figure 20-96 shows an example of a synchronous buck converter, operating over a wide load range. The dead time should be optimized to achieve the best efficiency over the normal load conditions (ZVS in situation a)). When the load is light the discharge curve for C_1 is longer than in a normal load condition, and therefore the switch turns ON too early. This will impose a discharge of the capacitor C_1 to ground (instead of passing the energy to the load). If the load is too high then the discharge time is too fast, what imposes losses due to body diode conduction.

Conversion optimization can then be increased by mapping the operating range of the specific converter versus a set of dead time values that can be stored in a memory.

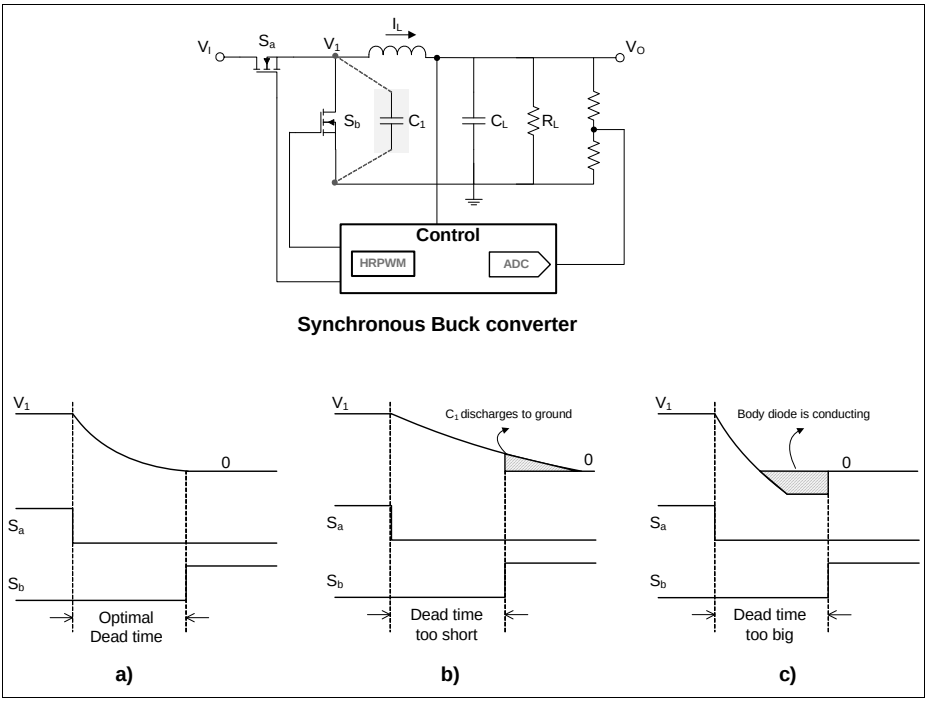


Figure 20-96 Dead Time vs. Load Conditions

HRCyCR1, HRCyCR2 and HRCySC Value Shadow Transfer

The shadow transfer mechanism for the **HRCyCR1** and **HRCyCR2** is needed to perform a cycle-by-cycle ON or OFF time adjustment of the PWM output. This adjustment can be done during normal operation without triggering any wrong behavior.

To make usage of the shadow transfer mechanism a Capture/Compare Unit Timer Slice is needed. This Timer Slice is used to dictate the time trigger for the shadow update of the HRCy registers.

Due to the fact that each HRC unit has a dual control mechanism to generate a PWM signal, made available from the two Source Selectors (Source Selector 0 and Source Selector 1), a flexible mechanism to keep the shadow transfer synchronized with the specific Timer is available, **Figure 20-97**.

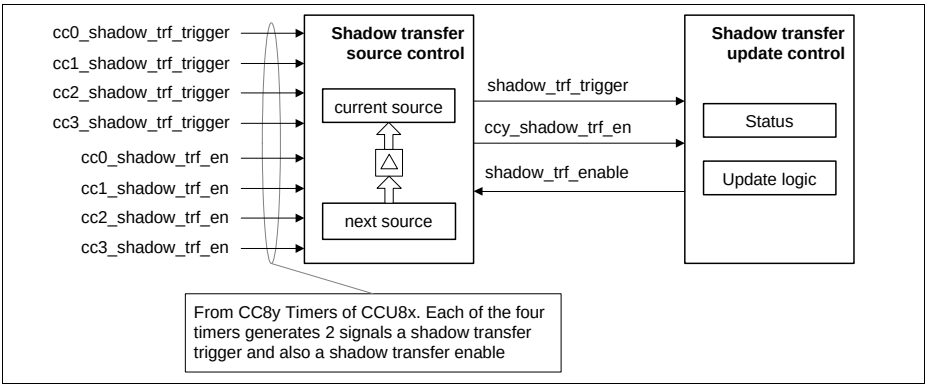


Figure 20-97 Shadow transfer control overview

Each HRC unit receives the shadow transfer enable and shadow transfer triggers, from the Capture/Compare Unit Timer Slices, CC8y. The selection of which Timer Slice is controlling the shadow transfer is dictated by the **HRCySC**.ST field.

Table 20-17 Shadow transfer control

HRCySC.ST	**Shadow transfer source**
0_B	Shadow transfer trigger and shadow transfer enable signals are linked to the Timer Slice (CC8y), connected to Source Selector 0.
1_B	Shadow transfer trigger and shadow transfer enable signals are linked to the Timer Slice (CC8y), connected to Source Selector 1.

Due to the fact that each HRCy contains two Source Selectors, it may be the case that connected to each Source Selector one has different timers, e.g. to Source Selector 0 one uses CC80 and in Source Selector 1 CC82 (this example is valid if **HRCyTSEL** = $0x10_H$).

In this case, every time that the **HRCySC**.ST is updated, the HW needs to update also which is the current timer that is being used to trigger a shadow transfer (the one connected to the High Resolution Path inside of each HRC unit), **Figure 20-98**.

This switch is done automatically via the HW every time that the **HRCySC**.ST field is updated and therefore no additional software routines are needed.

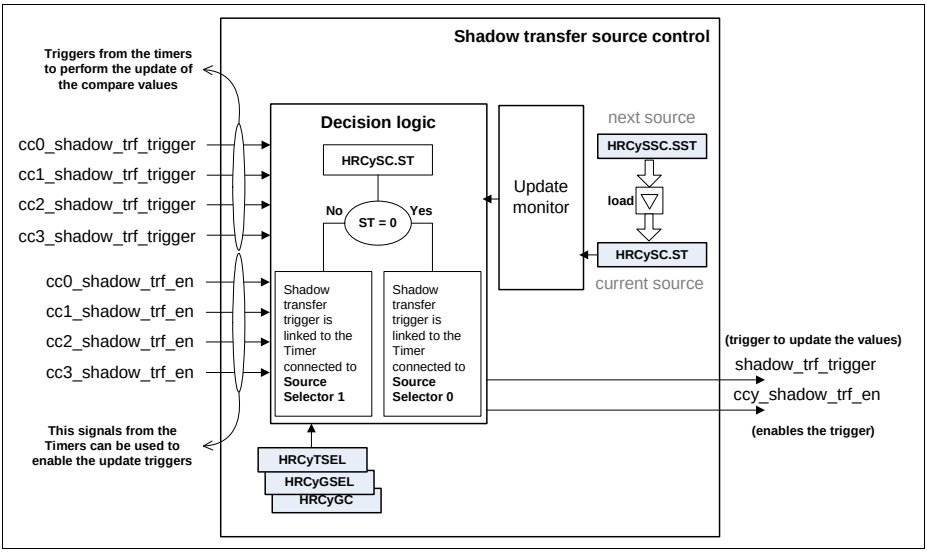


Figure 20-98 Shadow transfer source control logic

Figure 20-99 shows an example for this mechanism when the PWM output is controlled via two different group of resources:

- Resource group 1 controls the peak current control
- Resource group 2 controls a burst mode

The switch between the two resources/modes is done via software by updating the **HRCySC**.ST. The usage of the two modes can be dependent on different output load conditions.

Two distinct time frames can be seen in **Figure 20-99**: A and B.

During the A timeframe the PWM signal is being generated through a fixed frequency modulation with peak current control. The CSG0 Comparator is being used to Clear the PWM signal while the Capture/Compare Timer 0 is dictating the Set.

On timeframe B, the PWM signal is being generated entirely via the Capture/Compare Timer 1. Notice that in both time frames high resolution signal placement is possible.

The transition from A to B is dictated via the SW by updating the **HRCySC**.ST field.

By updating the **HRCySC**.ST field the software changes the current resource group that is controlling the PWM signal. At the same time the hardware updates automatically the source for the shadow transfer trigger. This will keep the synchronicity of the cycle-by-cycle update for the **HRCyCR1** and **HRCyCR2**, with the newly selected resource group.

Note: As an example during timeframe B the Comparator output is set to passive level.

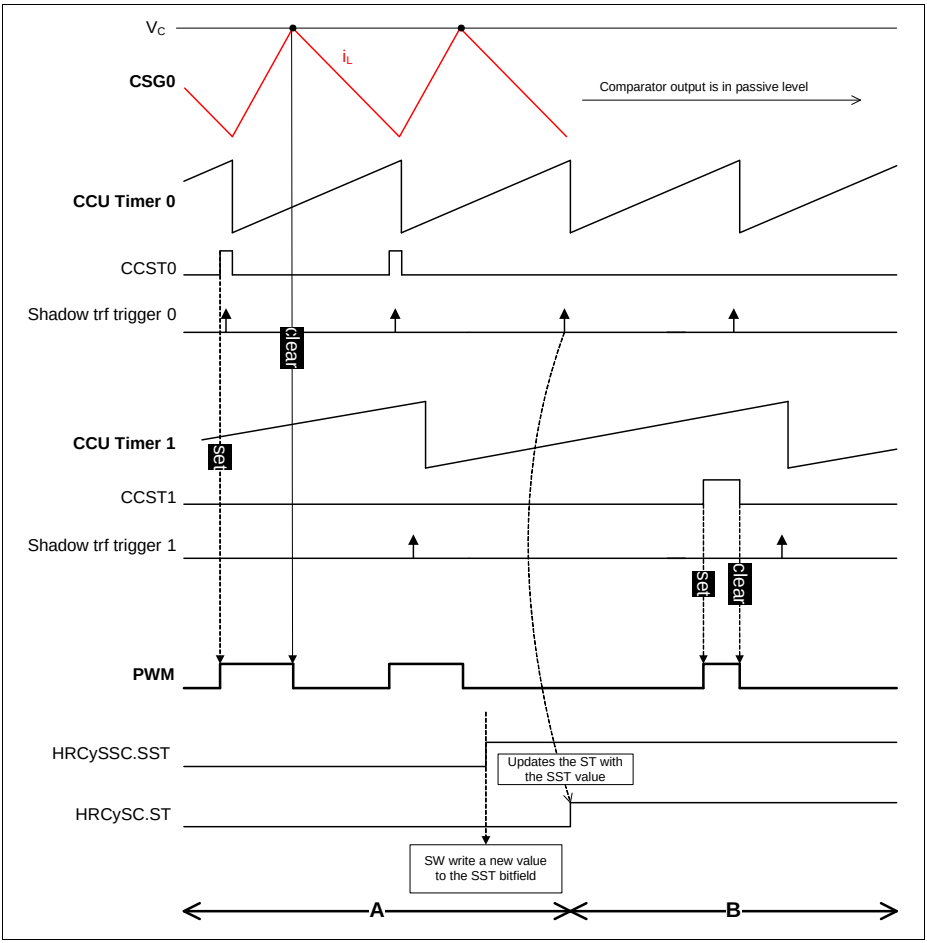


Figure 20-99 Shadow transfer Source Selector update

There are two possibilities for the software to request a shadow transfer for the high resolution compare values:

- Set the specific shadow transfer enable bit field: **HRCSTSG**.HySTE
- Use the shadow transfer enable bit field from the associated timer on the Capture/Compare Unit

The logic for controlling the shadow transfer enable signal is shown in **Figure 20-100**.

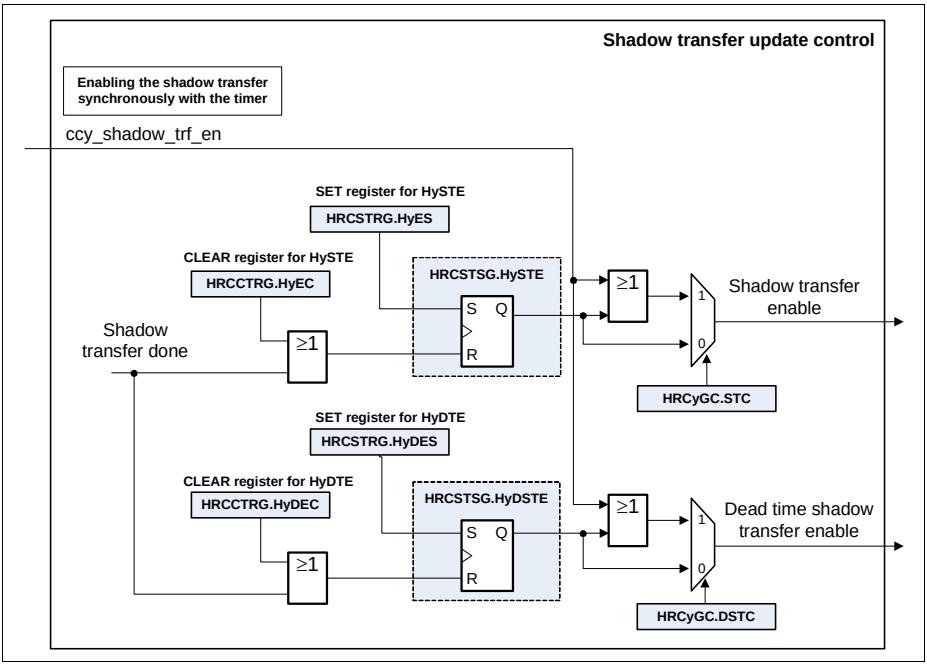


Figure 20-100Shadow transfer enable logic

Setting the **HRCSTSG**.HySTE field only enables the shadow transfer trigger and does not perform an update of the associated values. This means that after the shadow transfer is enabled, via the specific **HRCSTRG** field, the HRC channel needs to wait for a shadow update trigger from the Capture/Compare Unit Timer Slice.

Figure 20-101 shows the update of the **HRCyCR1** and **HRCyCR2** when the Capture/Compare Unit Timer Slice is programmed in edge aligned mode. Three timing examples can be seen:

- **HRCyGC**.HRMn = 00_B - PWM high resolution control on rising edge
- **HRCyGC**.HRMn = 01_B - PWM high resolution control on falling edge
- **HRCyGC**.HRMn = 10_B - PWM high resolution control on rising and falling edge

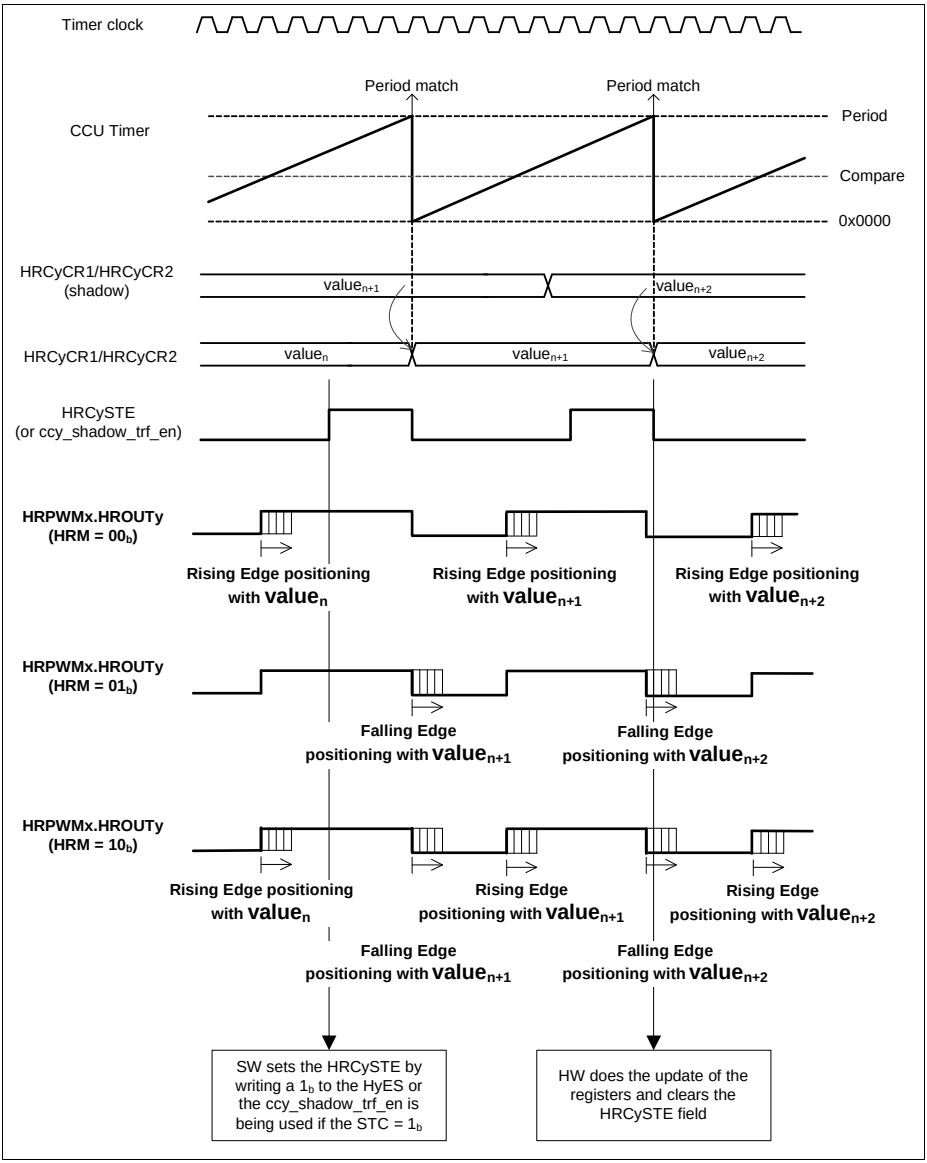


Figure 20-101Shadow transfer with Edge Aligned mode (CC8y)

When the Capture/Compare Unit Timer Slice is programmed to operate in center aligned mode, there are two time slots where the shadow transfer can be performed: when the timer hits the period while counting up and when the timer hits 1_D when counting down, **Figure 20-102**.

It may be helpful to control if both or only one timing slot is being used to perform the shadow transfer, when the Timer Slice is set to center aligned mode. This configuration needs to addressed on the Capture/Compare Unit Timer configuration (please address the Capture/Compare Unit 8 Chapter).

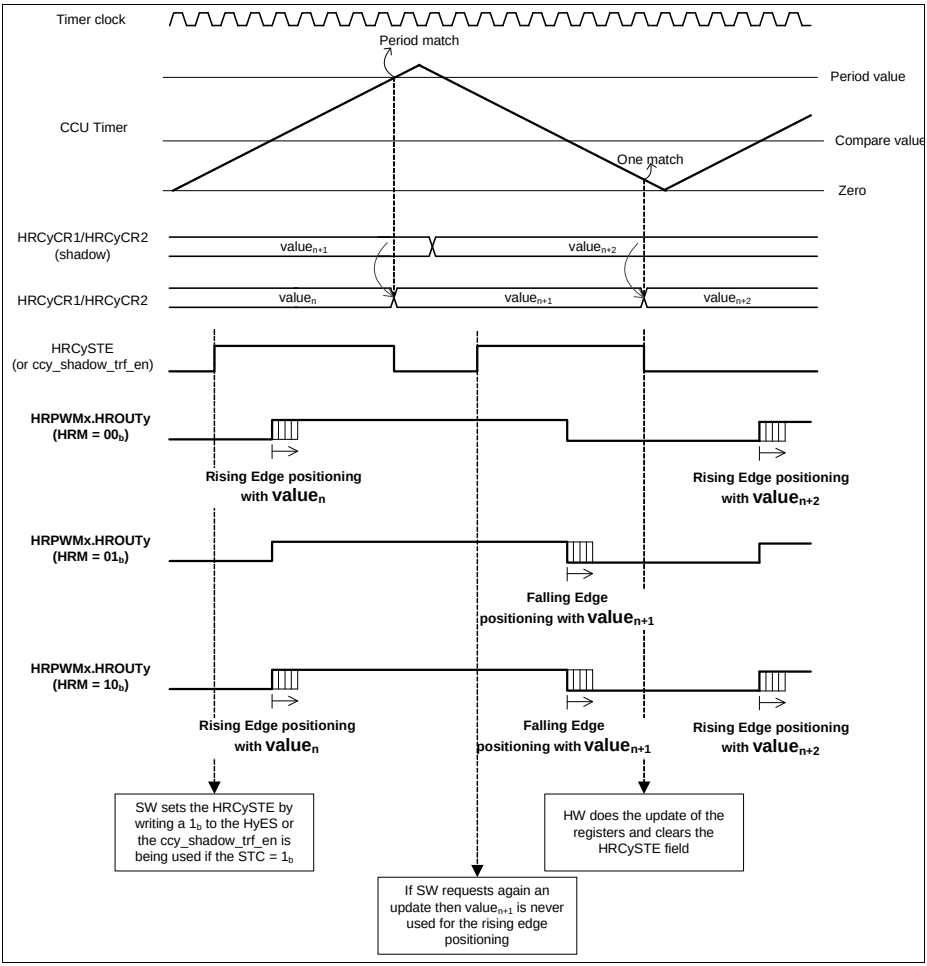


Figure 20-102Shadow transfer with Center Aligned mode (CC8y)

Dead Time Value Shadow Transfer

The dead time values update can be controlled via two different triggers: the same trigger used to update the high resolution compare values or an internal trigger that is generated when the dead time counter is not running, **Figure 20-103**.

The selection of which trigger is used to update the dead time values, is selected by the user by configuring the **HRCyGC**.DTUS bitfield accordingly.

When the **HRCyGC**.DTUS is set to 0_B, the dead time values are update at the same time as the high resolution compare values. When **HRCyGC**.DTUS is set to 1_B, then the dead time values are updated whenever the dead time counter is not running.

The configuration with **HRCyGC**.DTUS set to 0_B is normally used when the PWM signal is generated via a timer and high resolution is enabled. With this configuration, it is possible to synchronize the update of the dead time values and the high resolution compare values.

The configuration with **HRCyGC**.DTUS set to 1_B can be used when the PWM signal does not need high resolution. In this case the high resolution compare values are not used and a synchronization between these two updates is not needed (it may even not be possible if no Timer is being used to control the PWM output).

Enabling the shadow update of the dead time value, is done via the specific **HRCSTRG**.HyDES field. After the shadow update is enabled (**HRCSTSG**.HyDSTE = 1_B), the update trigger is unlocked and a shadow transfer is possible.

The field **HRCSTSG**.HyDSTE is automatically cleared by the HW when the shadow transfer has been performed, **Figure 20-100**.

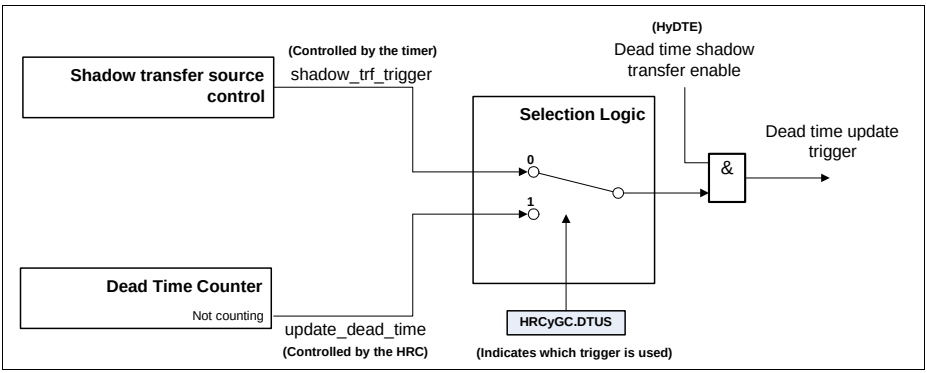


Figure 20-103Dead Time update trigger

Figure 20-104 shows the timing operation for the update of the dead time values when the output latch is being controlled only by a comparator. In this case, a set or clear operation that consequently triggers a dead time counting, always has priority over the update of the dead time values.

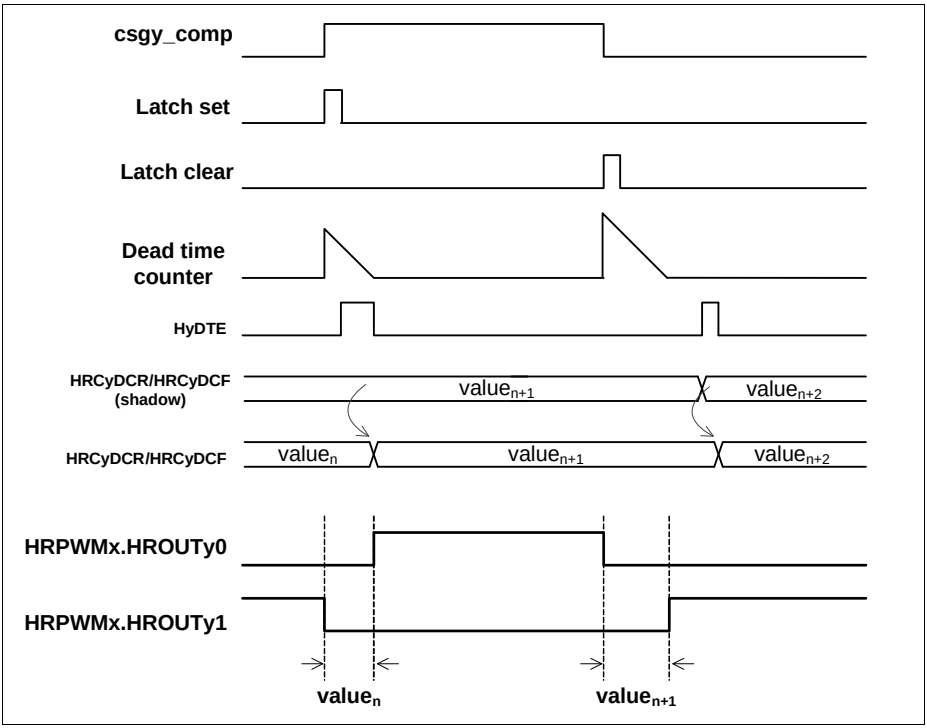


Figure 20-104Dead Time values update with comparator control

20.2.3.5 Constraints for the high resolution signal

For each HRCy unit some constraints exist for the generation of the output PWM signal. When using a Capture/Compare Unit Timer to generate the output PWM signal, these constraints are automatically checked and controlled via the hardware. The implemented software routines do not need to be adjusted due to this constraints.

Capture/Compare Unit Timer in Edge Aligned Mode

The fine adjustment of the high resolution PWM signal has the follow constraints when the Capture Compare Unit Timer Slice is set in edge aligned mode:

- The ON time cannot be smaller than 3 clock cycles (module clock)
- The OFF time cannot be smaller than 3 clock cycles (module clock)

Programming a PWM signal with an ON time smaller than 3 clock cycles, will result in a 0% duty cycle signal. Programming an OFF timer smaller than 3 clock cycles, will result in a 100% duty cycle signal.

Figure 20-105 to **Figure 20-107** show the constraints for the high resolution PWM signal generation, when the Timer Slice is in edge aligned mode. Notice that the compare value that dictates the set/clear of the CCSTy signal, is controlled on the Capture/Compare Unit. It is with the compare value of the Capture/Compare Unit that the reference for the edge positioning is generated.

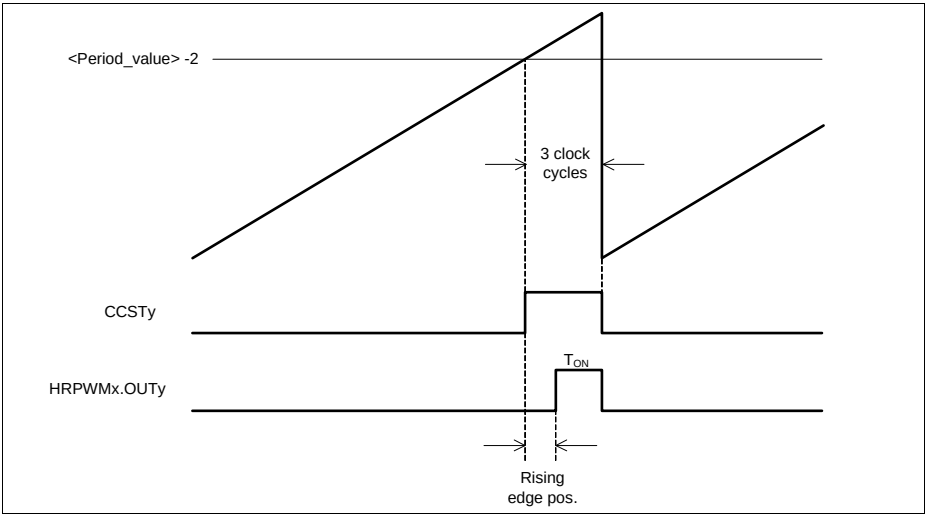


Figure 20-105High resolution PWM signal constraint - Edge Aligned, HRM = 00_B

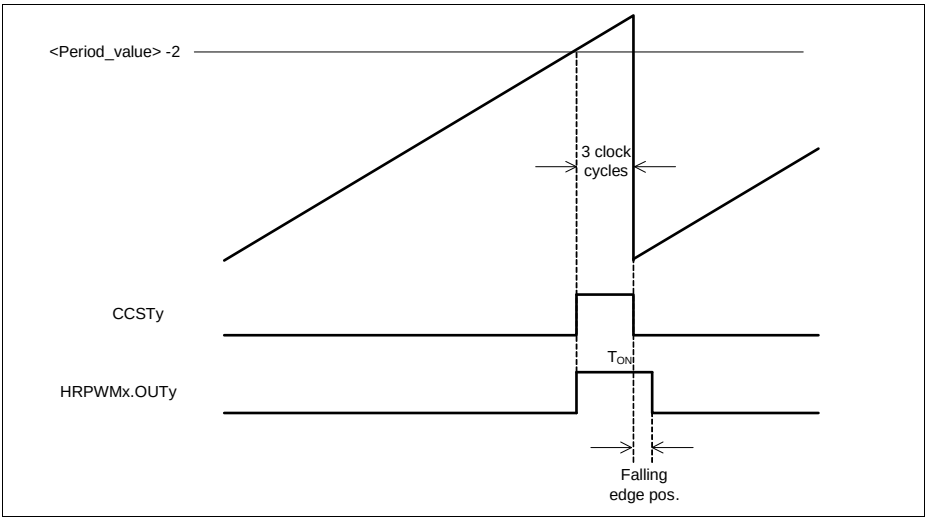


Figure 20-106High resolution PWM signal constraint - Edge Aligned, HRM = 01_B

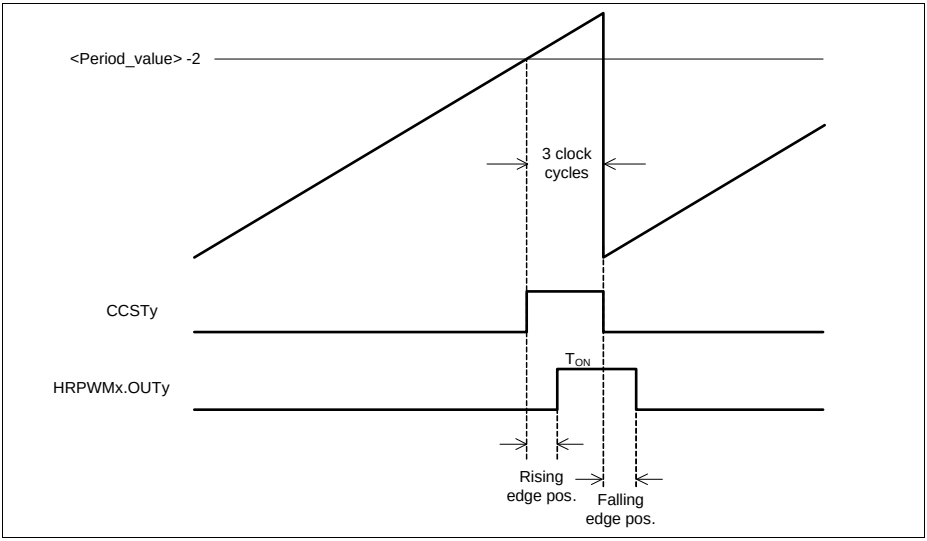


Figure 20-107High resolution PWM signal constraint - Edge Aligned, HRM = 11_B

Capture/Compare Unit Timer in Center Aligned Mode

The fine adjustment of the high resolution PWM signal has the follow constraints when the Capture Compare Unit Timer Slice is set in center aligned mode:

- The ON time cannot be smaller than 3 clock cycles (module clock)
- The OFF time cannot be smaller than 3 clock cycles (module clock)

Programming a PWM signal with an ON time smaller than 3 clock cycles, will result in a 0% duty cycle signal.

Programming a PWM signal with an OFF time smaller than 3 clock cycles, will result in a 100% duty cycle signal.

Figure 20-108 to **Figure 20-110** show the constraints for the high resolution PWM signal generation, when the Timer Slice is set in center aligned. Notice that the compare value that dictates the set/clear of the CCSTy signal, is controlled on the Capture/Compare Unit. It is with the compare value of the Capture/Compare Unit that the reference for the edge positioning is generated.

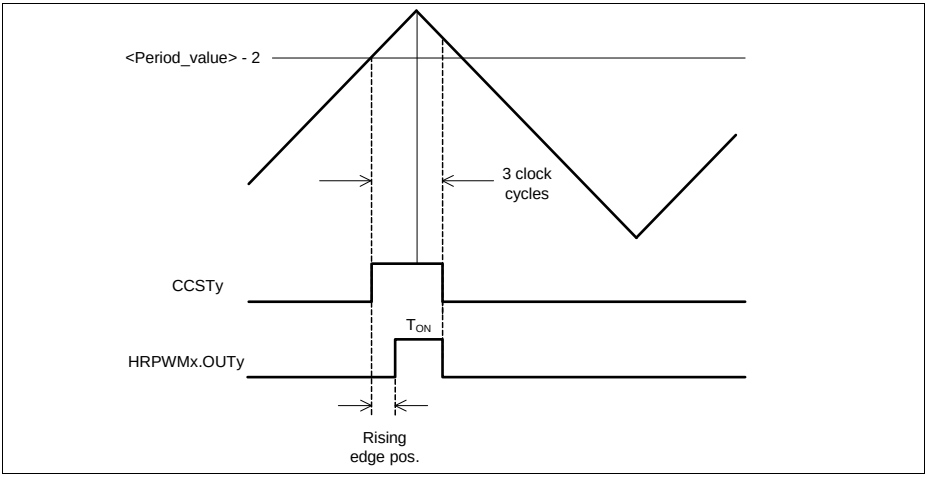


Figure 20-108HIgh resolution PWM signal constraints - Center Aligned, HRM = 00_B

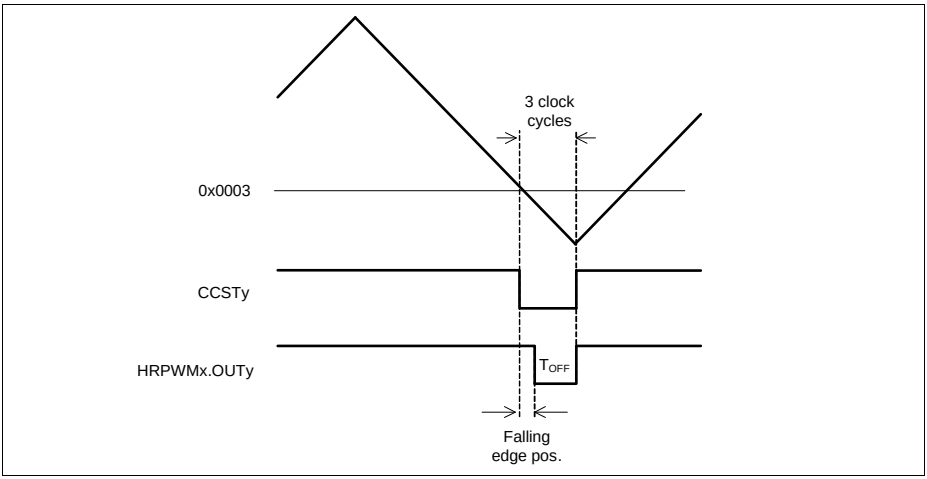


Figure 20-109High resolution PWM signal constraints - Center Aligned, HRM = 01_B

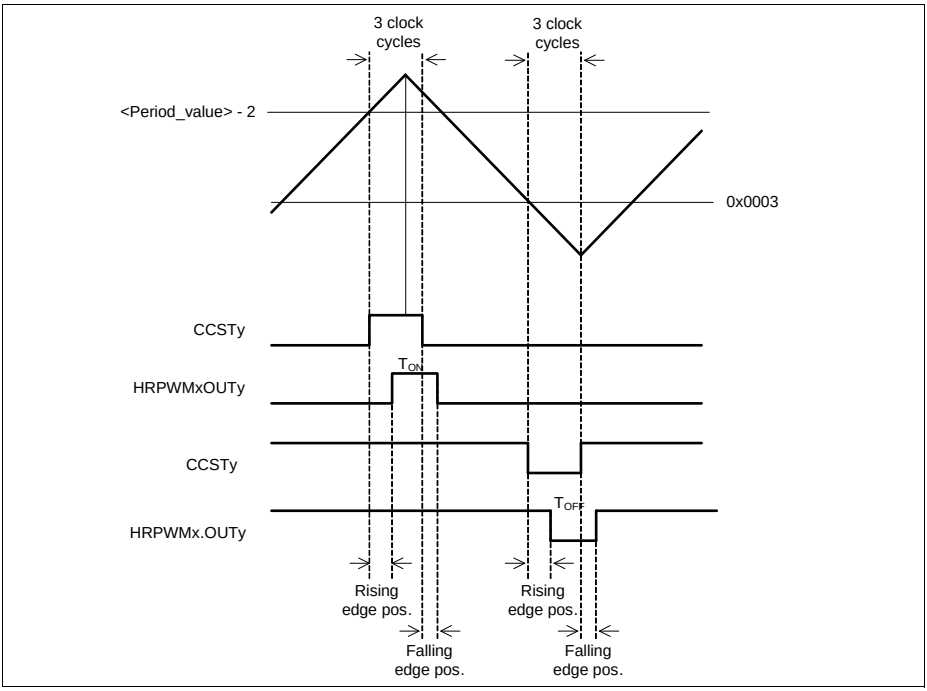


Figure 20-110High resolution PWM signal constraints - Center Aligned, HRM = 11_B

Capture/Compare Unit Timer in Asymmetrical Mode

When using both edges positioning mode, it is more suitable to use the asymmetrical mode (only available on Capture/Compare Unit 8) to be able to perform a phase shift on the PWM signal. This mode is especially useful for applications using phase shift topologies.

The generation of the high resolution signal has the follow constraints when the Timer Slice is set in asymmetrical edge aligned mode:

- The ON time cannot be smaller than 3 clock cycles (module clock)
- The maximum shift is only possible up to <Period_Value> - 2, on the Capture/Compare Unit

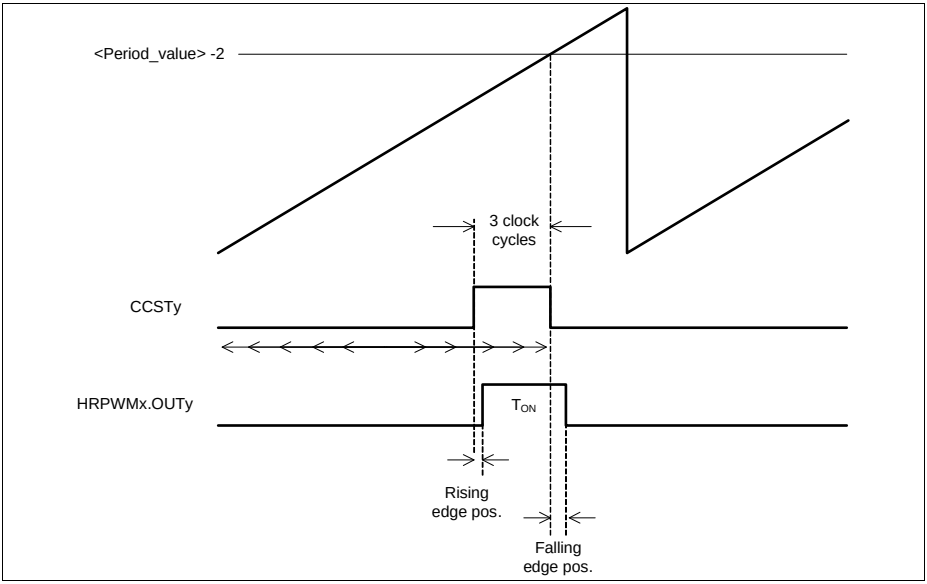


Figure 20-111High resolution signal constraints - Asymmetrical Edge mode

The generation of the high resolution signal has the follow constraints when the Timer Slice is set in asymmetrical center aligned mode:

- The ON time cannot be smaller than 3 clock cycles (module clock)
- The maximum shift is only possible up to within [0003_H:<Period_Value>-2], on the Capture/Compare Unit

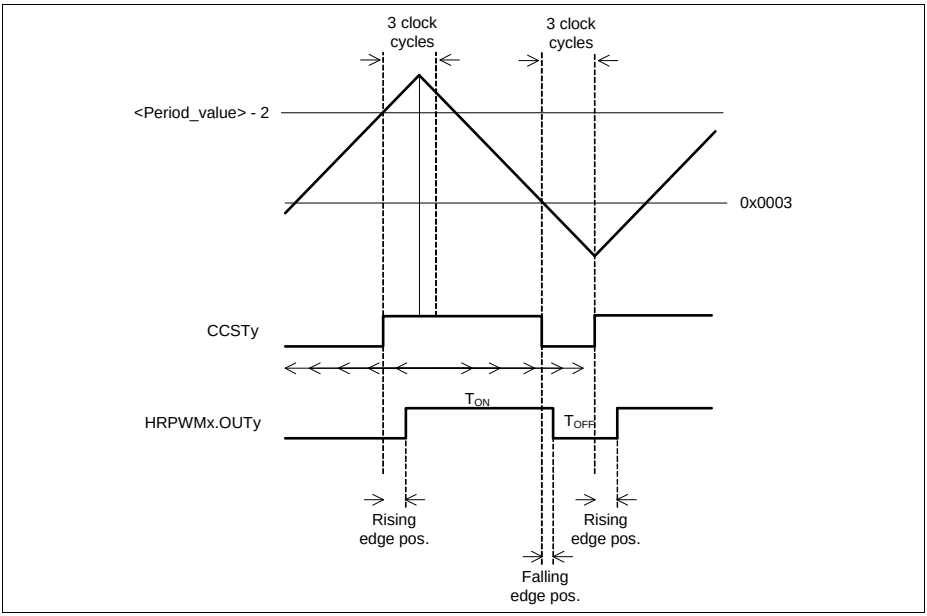


Figure 20-112High resolution signal constraints - Asymmetrical Center mode

Constraints versus Duty Cycle Adjustment

The limitation of 3 clock cycles for the minimum ON/OFF time, dictates the lower and upper boundary for the duty cycle adjustment with high resolution signal positioning.

Table 20-18 lists a set of typical frequencies and the correspondent boundaries.

Table 20-18 PWM frequency vs. minimum ON/OFF time1)

Switching Frequency (kHz)	Minimum DC (%)	Maximum DC (%)
250	0.625	99.375
500	1.25	98.75
1000	2.5	97.5
2000	5	95
4000	10	90

Note: 1)Assuming a reference clock frequency for the Capture/Compare Unit of 120 MHz (limit of 3 clock cycles is equal to 25 ns).

20.3 Service Request Generation

Each CSGy unit contains an Interrupt Control block that is able to generate service requests/interrupts from different sources (**Figure 20-113**):

- Switch from **CSGyDSV1** to **CSGyDSV2**:this indicates when the slope generation is disabled - static mode - a switch between the two reference values that are
  - when the slope control unit is set to static mode, this indicates that the value fed to the DAC passed from CSGyDSV1 to CSGyDSV2
  - when the slope control unit is set to triangular mode, this indicates a transition from the decrementing slope to the incrementing slope
- Switch from **CSGyDSV2** to **CSGyDSV1**
  - when the slope control unit is set to static mode, this indicates that the value fed to the DAC passed from CSGyDSV2 to CSGyDSV1
  - when the slope control unit is set to triangular mode, this indicates a transition from the incrementing slope to the decrementing slope
- Conversion trigger to the DAC has been generated
- Slope generation start
- Slope generation stop
- Shadow transfer of the **CSGyDSV1** and **CSGyPC** was performed
- Transition on the comparator output from HIGH to LOW
- Transition on the comparator output from LOW to HIGH
- Output of the comparator is set to the clamped state

Each service request source can be individually enabled via the **CSGySRE** register. The service request sources, slope_starts and slope_stops are ORed together. The same applies to the comp_out_rise and comp_out_fall signals.

A service request pulse is always generated (if the source is enabled) even if the status is already high. The status of the source can be set or cleared (via SW) even if the specific source is not enabled.

A node pointer is used to forward the enabled service request sources to one of the 4 specific service request lines (CSGy.SR0 to CSGy.SR3).

The SW has two dedicated registers where the set and clear of the service request sources can be handled.

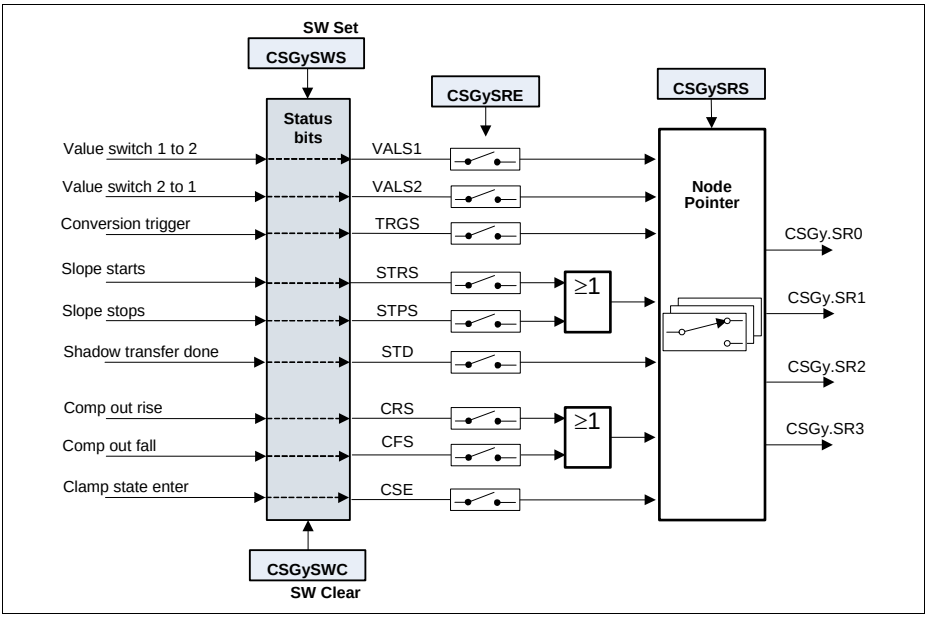


Figure 20-113CSGy Service request scheme

The service request lines from each CSGy unit are ORed together to generate the HRPWM output service request lines, **Figure 20-114**.

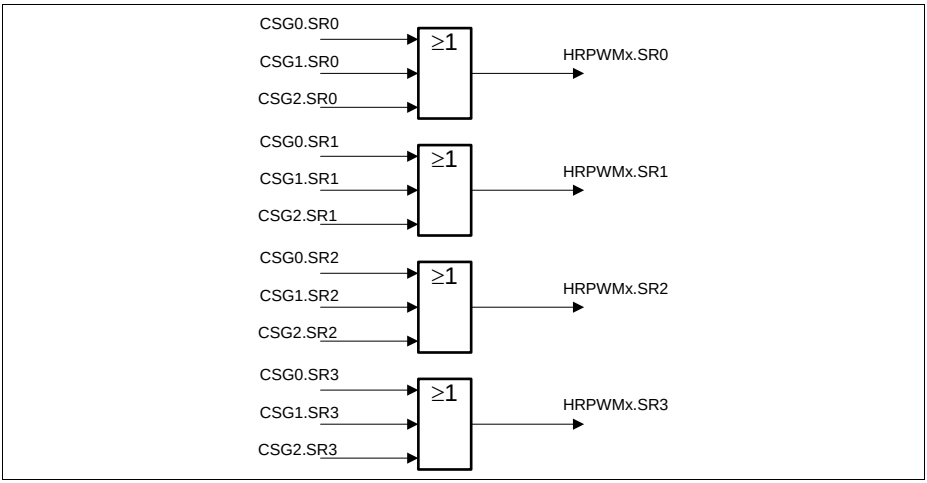


Figure 20-114HRPWM service request scheme

20.4 Debug Behavior

The debug suspend behavior for the HRPWM unit, is dictated by the value programmed into the **HRBSC**.SUSCFG field, **Table 20-19**.

The suspend mode is non intrusive, which means that the content of the registers is not altered when the module enters or exits the suspend mode. The registers can still be accessed by the CPU (read only).

This mode is useful for debugging purposes, e.g., where the current device status should be frozen in order to get a snapshot of the internal values.

To be able to use the module after the suspend mode has been released, a complete re initialization of the module is needed.

Table 20-19 SUSCFG configuration

HRBSC.SUSCFG	Description
000_B	Suspend is ignored. Module continues to work normally.
001_B	• CSGy units are halted. • HRCy units are halted.
010_B	• Comparator outputs, HRPWMx.CyO are clamped to passive level and the CSGy units are halted. • High resolution channels outputs, HRPWMx.HROUTy0 and HRPWMx.HROUTy1, are clamped to passive level and the HRCy units are halted.
011_B	• CSGy units are halted. • High resolution channels outputs, HRPWMx.HROUTy0 and HRPWMx.HROUTy1, are clamped to passive level and the HRCy units are halted.
111_B - 100_B	Reserved

20.5 Analog Behavior, Power, Reset and Clock

The following chapters describe the operating conditions, characteristics and timing requirements, for all the components inside the HRPWM module. Each description is given for just one sub unit, e.g., one CSG or one HRC.

All the timing information is related to the module clock, f_{hrpwm}.

20.5.1 Clocks

Module Clock

The module clock for the HRPWM, f_{hrpwm}, unit is linked with the clock of the Capture/Compare Unit, f_{ccu}. It is not possible to have the module clock of the HRPWM with a different frequency of the one used for the Capture/Compare Unit.

At system level, there is the possibility to disable the clock for the HRPWM, while the clock for the Capture/Compare unit is running, or vice-versa, **Figure 20-115**.

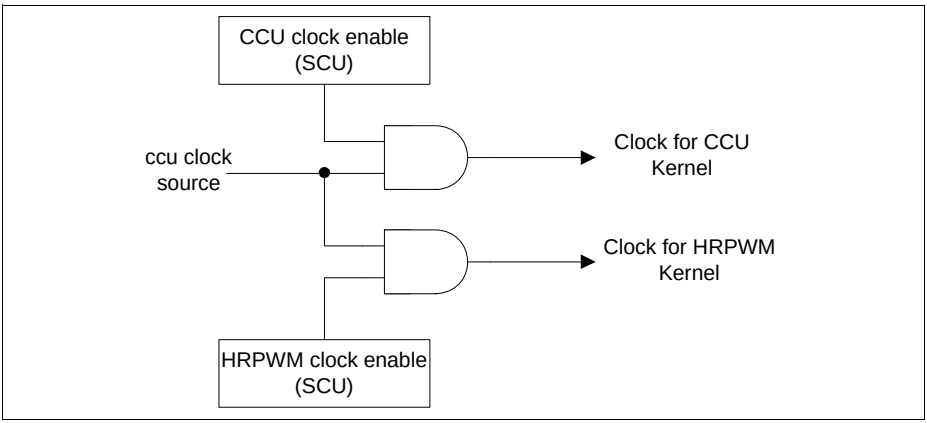


Figure 20-115HRPWM clock structure

Table 20-20 lists the thresholds for the module clock frequency. These thresholds guarantee an operation of all functions, including the generation of the high resolution signals.

Any values for the module clock, that fall outside the thresholds mentioned on **Table 20-20**, can lead to unpredictable results.

Table 20-20 Module clock operating conditions

Parameter	Symbol	Values			Unit	Note / Test Condition
		Min.	Typ.	Max.		
Frequency[1)]	f_{hrpwm}	80	–	180	MHz	

1) The value for the clock frequency that guarantees a 150 ps resolution are: 180, 120 and 80 MHz

CSG External Clock

It is possible to select an external source, that can be used as a clock for the slope generation, HRPWMx.ECLKy. This clock is synchronized internally with the module clock and therefore the external clock needs to meet the criterion described on **Table 20-21**.

Table 20-21 External clock operating conditions

Parameter	Symbol	Values			Unit	Note / Test Condition
		Min.	Typ.	Max.		
Frequency	f_{eclk}	–	–	$f_{hrpwm}/4$	MHz	
ON time	ton_{eclk}	$2T_{ccu}$ 1)2)	–	–	ns	
OFF time	$toff_{eclk}$	$2T_{ccu}$ 1)2)	–	–	ns	Only the rising edge is used

1) Only valid if the signal was not previously synchronized/generated with the fccu clock (or a synchronous clock)

2) 50% duty cycle is not obligatory

DAC Conversion Clock

The DAC conversion clock, can be generated internally or it can be controlled via a HRPWM module pin.

Table 20-22 External DAC conversion trigger operating conditions

Parameter	Symbol	Values			Unit	Note / Test Condition
		Min.	Typ.	Max.		
Frequency	f_{etrg}	–	–	30 2)	MHz	
ON time	ton_{etrg}	$2T_{ccu}$ 1)2)	–	–	ns	
OFF time	$toff_{etrg}$	$2T_{ccu}$ 1)2)	–	–	ns	

1) 50% duty cycle is not obligatory

2) Only valid if the signal was not previously synchronized/generated with the fccu clock (or a synchronous clock)

20.5.2 Module Reset

The HRPWM only has one reset source. This reset source is handled at system level and it can be generated independently via a system control register.

After reset release, all the analog blocks are in power down mode. The DAC Control Block inside the CSGy unit is not operating due to the fact that the run bit is not set.

The HRC channels, after reset release are in power down mode and therefore all the outputs are in their disabled state.

20.5.3 Analog Behavior

20.5.3.1 CSG unit

Whenever the reset of the HRPWM is asserted, the digital interface between the digital control and the analog blocks is initialized. If the analog part of the CSGy is not set in power down mode, the reset, acts as a “0” conversion for the DAC. The output of the comparator of the CSGy unit is not clamped after the reset is released.

Setting a CSGy unit in power down mode doesn’t guarantee that the output of the comparator is in a stable value, therefore before setting the unit in power down, the clamping stage should be activated.

20.5.3.2 HRC unit

After the reset is released, some time is needed for the HRC part used to generate the high resolution signals, to be initialized.

Setting the HRC units in power down, will disable the high resolution signal generation part, but the output latch is still controllable via the Capture/Compare Unit timer or via the specific comparator output(s).

Change the power mode, of the HRC units, on the fly it is not supported and it can lead to unpredictable results.

20.6 Initialization Sequence

The simplified initialization sequence for an application that is using the HRC, CSG and Capture/Compare Unit timer, should be the following:

1st Step: Apply reset to the HRPWM and CCU80 (Capture and Compare Unit 8 instance 0) via the specific SCU register, PRSET0/PRSET1.

2nd Step: Release reset of HRPWM, via the specific SCU register, PRCLR0.

3rd Step: Release reset of CCU80 (Capture and Compare Unit 8 instance 0), via the specific SCU register, PRCLR0/PRCLR1

4th Step: Enable the clock for the CCU80 and the HRPWM, via the specific SCU register, CLKSET.

5th Step: Write 0x00004A4E to the **GLBANA** register.

6th Step: Copy the data located at 0x20000084 (complete 32 bits) into the CSG0 memory location 0x40020A40.

Copy the data located at 0x20000088 (complete 32 bits) into the CSG1 memory location 0x40020B40.

Copy the data located at 0x2000008C (complete 32 bits) into the CSG0 memory location 0x40020C40.

Note: This step is only valid for qualified samples.

7th Step: If a CSGy subunit needs to run in the Low Speed Mode (**CSGCFG**.CyPM field needs to be set to 01_B), then (for High Speed Mode, this step is skipped):

Update the memory address 0x40020A40 bits [14:11] with 1100_B in case CSG0 unit is going to be set in Low Speed Mode. The rest of the 32 bit data should not be modified.

Update the memory address 0x40020B40 bits [14:11] with 1100_B in case CSG1 unit is going to be set in Low Speed Mode. The rest of the 32 bit data should not be modified.

Update the memory address 0x40020C40 bits [14:11] with 1100_B in case CSG2 unit is going to be set in Low Speed Mode. The rest of the 32 bit data should not be modified.

Note: This step is only valid for qualified samples.

8th Step: Enable the Bias Generator of the HRPWM, by setting the bitfield **HRBSC**.HRBE = 1_B. Wait for the bias generation initialization time (see t_{start} parameter in the CMP and 10-bit DAC characteristics table in the specific Data Sheet).

9th Step: Remove the CSGy unit(s) from power down mode, by setting the specific **CSGCFG**.CyPM. Wait for the CSGy startup time. (see t_{csgs} parameter in the CMP and 10-bit DAC characteristics table in the specific Data Sheet).

10th Step: Set the CSGy Comparator output to passive if the analog input value is not stable, by configuring **CSGyPLC**.PSL accordingly and setting **CSGSTATG**.PSLSy = 1_B.

Note: If the Comparator output is used for starting a CCU8x/CCU4x timer then the passive level should be set to a value that avoid a premature start. The same thing applies if the Comparator output is being used to SET the HRC latch.

11th Step: Configure the CSGy unit(s) SFRs. After configuring the SFRs a shadow transfer needs to be requested to load the **CSGyDSV1** and **CSGyPC** with the proper values. This is done by writing a 1_B into the **CSGTRG**.DySES field(s).

Note: The service requests used for interrupts can be configured but the SW should discard any request untill the initialization sequence is over.

12th Step: Configure the Capture/Compare Unit, CCU80, SFRs.

13th Step: Configure the HRC clock configuration field, **HRCCFG**.CLKC, with the clock frequency information.

14th Step: Release the HRC high resolution generation logic from power down, by setting **HRCCFG**.HRCPM = 1_B.

15th Step: After the HRCPM is set to 1_B a 2 us period should be inserted/elapsed. After this waiting period, the **GLBANA**.GHREN should be set to 1_B. Notice that the rest of the fields values need to be kept.

16th Step: After the startup time has been elapsed (see t_{start} parameter in the HRC characteristics table in the specific Data Sheet), poll the high resolution ready bitfield, **HRGHRS**.HRGR untill it is set to 1_B - this indicates that the High Resolution Logic is ready.

17th Step: Configure the HRCy unit(s) SFRs.

18th Step: Remove the Comparator output from passive level (if it was previously set to passive level), by writing a 1_B in the **CSGCLRG**.CCyP bitfield(s).

19th Step: Start setting the run bits (of the units) from an application point of view, e.g: if the first action should be a SET for the PWM signal, then set first the run first of the unit that is controlling the clear. The run bit of the DAC Control Unit, **CSGSTATG**.DyRB, should be set before the run bit of the Comparator, **CSGSTATG**.CyRB.

*Note: If the DAC of the CSGy is being used to generate a slope, then the prescaler also needs to be started (via SW or external signal) - **CSGFSG**.PyRB = 1_B.*

20.7 HRPWM Registers

Registers Overview

The absolute register address is calculated by adding:

Module Base Address + Offset Address

Table 20-23 Registers Address Space

Module	Base Address	End Address	Note
HRPWM0	40020000_H	$40023FFF_H$	

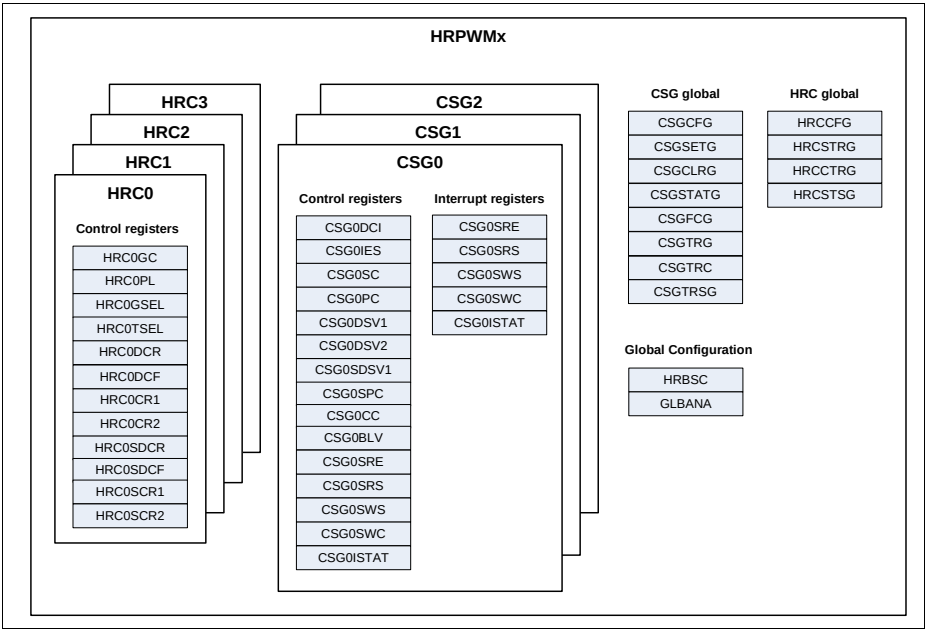


Figure 20-116HRPWM registers overview

Table 20-24 Register Overview

Short Name	Description	Offset Addr.	Access Mode		Description See
			Read	Write	
HRPWM Suspend Configuration					
HRBSC	Suspend & Bias config	0900_H	U, PV	U, PV	Page 20-138

Table 20-24 Register Overview (cont'd)

Short Name	Description	Offset Addr.	Access Mode		Description See
			Read	Write	
Reserved	Reserved	0904_H	BE	BE	
MIDR	Module identification register	0908_H	U, PV	BE	Page 20-139
Reserved	Reserved	$090C_H$ - 0910_H	BE	BE	
GLBANA	Global Analog Configuration Register	0914_H	U, PV	U, PV	Page 20-140
CSG Global Registers					
CSGCFG	Global CSG configuration	0920_H	U, PV	U, PV	Page 20-141
CSGSETG	Global CSG run bit set	0924_H	U, PV	U, PV	Page 20-143
CSGCLRG	Global CSG run bit clear	0928_H	U, PV	U, PV	Page 20-144
CSGSTATG	Global run bit status	$092C_H$	U, PV	BE	Page 20-146
CSGFCG	Global slope and prescaler control	0930_H	U, PV	U, PV	Page 20-147
CSGFSG	Global slope and prescaler run status	0934_H	U, PV	BE	Page 20-150
CSGTRG	Global shadow transfer and switch enable	0938_H	U, PV	U, PV	Page 20-151
CSGTRC	Global shadow transfer clear	$093C_H$	U, PV	U, PV	Page 20-152
CSGTRSG	Global shadow transfer and switch status	0940_H	U, PV	BE	Page 20-153
Reserved	Reserved	0944_H - $095C_H$	BE	BE	
CSG0 Registers					
CSG0DCI	External triggers input selector	$0A00_H$	U, PV	U, PV	Page 20-155
CSG0IES	External triggers edge/level configuration	$0A04_H$	U, PV	U, PV	Page 20-156
CSG0SC	Slope generation configuration	$0A08_H$	U, PV	U, PV	Page 20-158
CSG0PC	Pulse swallow control value	$0A0C_H$	U, PV	BE	Page 20-162
CSG0DSV1	DAC input reference value 1	$0A10_H$	U, PV	BE	Page 20-163
CSG0DSV2	DAC input reference value 2	$0A14_H$	U, PV	U, PV	Page 20-164

Table 20-24 Register Overview (cont'd)

Short Name	Description	Offset Addr.	Access Mode		Description See
			Read	Write	
CSG0SDSV1	Shadow DAC input reference value 1	$0A18_H$	U, PV	U, PV	Page 20-165
CSG0SPC	Shadow pulse swallow control value	$0A1C_H$	U, PV	U, PV	Page 20-165
CSG0CC	Comparator general configuration	$0A20_H$	U, PV	U, PV	Page 20-166
CSG0PLC	Comparator passive level configuration	$0A24_H$	U, PV	U, PV	Page 20-169
CSG0BLV	Blanking period value	$0A28_H$	U, PV	U, PV	Page 20-171
CSG0SRE	Service request enable	$0A2C_H$	U, PV	U, PV	Page 20-172
CSG0SRS	Service request output selection	$0A30_H$	U, PV	U, PV	Page 20-174
CSG0SWS	Service request set register	$0A34_H$	U, PV	U, PV	Page 20-176
CSG0SWC	Service request clear register	$0A38_H$	U, PV	U, PV	Page 20-177
CSG0ISTAT	Service request status register	$0A3C_H$	U, PV	BE	Page 20-179
Reserved	Reserved	$0A44_H$ - $0AFC_H$	BE	BE	
CSG1 Registers					
CSG1DCI	External triggers input selector	$0B00_H$	U, PV	U, PV	Page 20-155
CSG1IES	External triggers edge/level configuration	$0B04_H$	U, PV	U, PV	Page 20-156
CSG1SC	Slope generation configuration	$0B08_H$	U, PV	U, PV	Page 20-158
CSG1PC	Pulse swallow control value	$0B0C_H$	U, PV	BE	Page 20-162
CSG1DSV1	DAC input reference value 1	$0B10_H$	U, PV	BE	Page 20-163
CSG1DSV2	DAC input reference value 2	$0B14_H$	U, PV	U, PV	Page 20-164
CSG1SDSV1	Shadow DAC input reference value 1	$0B18_H$	U, PV	U, PV	Page 20-165
CSG1SPC	Shadow pulse swallow control value	$0B1C_H$	U, PV	U, PV	Page 20-165

Table 20-24 Register Overview (cont'd)

Short Name	Description	Offset Addr.	Access Mode		Description See
			Read	Write	
CSG1CC	Comparator configuration	$0B20_H$	U, PV	U, PV	Page 20-166
CSG1PLC	Comparator passive level configuration	$0B24_H$	U, PV	U, PV	Page 20-169
CSG1BLV	Blanking period value	$0B28_H$	U, PV	U, PV	Page 20-171
CSG1SRE	Service request enable	$0B2C_H$	U, PV	U, PV	Page 20-172
CSG1SRS	Service request output selection	$0B30_H$	U, PV	U, PV	Page 20-174
CSG1SWS	Service request set register	$0B34_H$	U, PV	U, PV	Page 20-176
CSG1SWC	Service request clear register	$0B38_H$	U, PV	U, PV	Page 20-177
CSG1ISTAT	Service request status register	$0B3C_H$	U, PV	BE	Page 20-179
Reserved	Reserved	$0B44_H$ - $0BFC_H$	BE	BE	
CSG2 Registers					
CSG2DCI	External triggers input selector	$0C00_H$	U, PV	U, PV	Page 20-155
CSG2IES	External triggers edge/level configuration	$0C04_H$	U, PV	U, PV	Page 20-156
CSG2SC	Slope generation configuration	$0C08_H$	U, PV	U, PV	Page 20-158
CSG2PC	Pulse swallow control value	$0C0C_H$	U, PV	BE	Page 20-162
CSG2DSV1	DAC input reference value 1	$0C10_H$	U, PV	BE	Page 20-163
CSG2DSV2	DAC input reference value 2	$0C14_H$	U, PV	U, PV	Page 20-164
CSG2SDSV1	Shadow DAC input reference value 1	$0C18_H$	U, PV	U, PV	Page 20-165
CSG2SPC	Shadow pulse swallow control value	$0C1C_H$	U, PV	U, PV	Page 20-165
CSG2CC	Comparator configuration	$0C20_H$	U, PV	U, PV	Page 20-166
CSG2PLC	Comparator passive level configuration	$0C24_H$	U, PV	U, PV	Page 20-169
CSG2BLV	Blanking period value	$0C28_H$	U, PV	U, PV	Page 20-171
CSG2SRE	Service request enable	$0C2C_H$	U, PV	U, PV	Page 20-172

Table 20-24 Register Overview (cont'd)

Short Name	Description	Offset Addr.	Access Mode		Description See
			Read	Write	
CSG2SRS	Service request output selection	$0C30_H$	U, PV	U, PV	Page 20-174
CSG2SWS	Service request set register	$0C34_H$	U, PV	U, PV	Page 20-176
CSG2SWC	Service request clear register	$0C38_H$	U, PV	U, PV	Page 20-177
CSG2ISTAT	Service request status register	$0C3C_H$	U, PV	BE	Page 20-179
Reserved	Reserved	$0C44_H$ - $0CFC_H$	BE	BE	
HRC Global Registers					
HRCCFG	Global clock and power configuration	0960_H	U, PV	U, PV	Page 20-181
HRCSTRG	Shadow transfer trigger enable	0964_H	U, PV	U, PV	Page 20-184
HRCCTRG	Shadow transfer trigger clear	0968_H	U, PV	U, PV	Page 20-187
HRCSTSG	Shadow transfer trigger status	$096C_H$	U, PV	BE	Page 20-189
HRGHRS	High Resolution Generation Status	0970_H	U, PV	BE	Page 20-191
Reserved	Reserved	0974_H - $09FC_H$	BE	BE	
HRC0 Registers					
HRC0GC	General channel configuration	1300_H	U, PV	U, PV	Page 20-192
HRC0PL	Output passive level configuration	1304_H	U, PV	U, PV	Page 20-195
HRC0GSEL	Global channel set and clear inputs configuration	1308_H	U, PV	U, PV	Page 20-196
HRC0TSEL	Timer selection	$130C_H$	U, PV	U, PV	Page 20-199
HRC0SC	Source Selector for the shadow transfer trigger selection	1310_H	U, PV	U, PV	Page 20-201
HRC0DCR	Rising dead time value	1314_H	U, PV	BE	Page 20-202
HRC0DCF	Falling dead time vale	1318_H	U, PV	BE	Page 20-203

Table 20-24 Register Overview (cont'd)

Short Name	Description	Offset Addr.	Access Mode		Description See
			Read	Write	
HRC0CR1	High resolution compare value 1	$131C_H$	U, PV	BE	Page 20-204
HRC0CR2	High resolution compare value 2	1320_H	U, PV	BE	Page 20-205
HRC0SSC	Shadow Source Selector for the shadow transfer trigger selection	1324_H	U, PV	U, PV	Page 20-206
HRC0SDCR	Shadow rising dead time value	1328_H	U, PV	U, PV	Page 20-207
HRC0SDCF	Shadow falling dead time value	$132C_H$	U, PV	U, PV	Page 20-208
HRC0SCR1	Shadow high resolution compare value 1	1330_H	U, PV	U, PV	Page 20-208
HRC0SCR2	Shadow high resolution compare value 2	1334_H	U, PV	U, PV	Page 20-210
Reserved	Reserved	$133C_H$ - $13FC_H$	BE	BE	
HRC1 Registers					
HRC1GC	General channel configuration	1400_H	U, PV	U, PV	Page 20-192
HRC1PL	Output passive level configuration	1404_H	U, PV	U, PV	Page 20-195
HRC1GSEL	Global channel set and clear inputs configuration	1408_H	U, PV	U, PV	Page 20-196
HRC1TSEL	Timer selection	$140C_H$	U, PV	U, PV	Page 20-199
HRC1SC	Source Selector for the shadow transfer trigger selection	1410_H	U, PV	U, PV	Page 20-201
HRC1DCR	Rising dead time value	1414_H	U, PV	BE	Page 20-202
HRC1DCF	Falling dead time vale	1418_H	U, PV	BE	Page 20-203
HRC1CR1	High resolution compare value 1	$141C_H$	U, PV	BE	Page 20-204

Table 20-24 Register Overview (cont'd)

Short Name	Description	Offset Addr.	Access Mode		Description See
			Read	Write	
HRC1CR2	High resolution compare value 2	1420_H	U, PV	BE	Page 20-205
HRC1SSC	Shadow Source Selector for the shadow transfer trigger selection	1424_H	U, PV	U, PV	Page 20-206
HRC1SDCR	Shadow rising dead time value	1428_H	U, PV	U, PV	Page 20-207
HRC1SDCF	Shadow falling dead time value	$142C_H$	U, PV	U, PV	Page 20-208
HRC1SCR1	Shadow high resolution compare value 1	1430_H	U, PV	U, PV	Page 20-208
HRC1SCR2	Shadow high resolution compare value 2	1434_H	U, PV	U, PV	Page 20-210
Reserved	Reserved	$143C_H$ - $14FC_H$	BE	BE	
HRC2 Registers					
HRC2GC	General channel configuration	1500_H	U, PV	U, PV	Page 20-192
HRC2PL	Output passive level configuration	1504_H	U, PV	U, PV	Page 20-195
HRC2GSEL	Global channel set and clear inputs configuration	1508_H	U, PV	U, PV	Page 20-196
HRC2TSEL	Timer selection	$150C_H$	U, PV	U, PV	Page 20-199
HRC2SC	Source Selector for the shadow transfer trigger selection	1510_H	U, PV	U, PV	Page 20-201
HRC2DCR	Rising dead time value	1514_H	U, PV	BE	Page 20-202
HRC2DCF	Falling dead time vale	1518_H	U, PV	BE	Page 20-203
HRC2CR1	High resolution compare value 1	$151C_H$	U, PV	BE	Page 20-204
HRC2CR2	High resolution compare value 2	1520_H	U, PV	BE	Page 20-205

Table 20-24 Register Overview (cont'd)

Short Name	Description	Offset Addr.	Access Mode		Description See
			Read	Write	
HRC2SSC	Shadow Source Selector for the shadow transfer trigger selection	1524_H	U, PV	U, PV	Page 20-206
HRC2SDCR	Shadow rising dead time value	1528_H	U, PV	U, PV	Page 20-207
HRC2SDCF	Shadow falling dead time value	$152C_H$	U, PV	U, PV	Page 20-208
HRC2SCR1	Shadow high resolution compare value 1	1530_H	U, PV	U, PV	Page 20-208
HRC2SCR2	Shadow high resolution compare value 2	1534_H	U, PV	U, PV	Page 20-210
Reserved	Reserved	$153C_H$ - $15FC_H$	BE	BE	
HRC3 Registers					
HRC3GC	General channel configuration	1600_H	U, PV	U, PV	Page 20-192
HRC3PL	Output passive level configuration	1604_H	U, PV	U, PV	Page 20-195
HRC3GSEL	Global channel set and clear inputs configuration	1608_H	U, PV	U, PV	Page 20-196
HRC3TSEL	Timer selection	$160C_H$	U, PV	U, PV	Page 20-199
HRC3SC	Source Selector for the shadow transfer trigger selection	1610_H	U, PV	U, PV	Page 20-201
HRC3DCR	Rising dead time value	1614_H	U, PV	BE	Page 20-202
HRC3DCF	Falling dead time vale	1618_H	U, PV	BE	Page 20-203
HRC3CR1	High resolution compare value 1	$161C_H$	U, PV	BE	Page 20-204
HRC3CR2	High resolution compare value 2	1620_H	U, PV	BE	Page 20-205
HRC3SSC	Shadow Source Selector for the shadow transfer trigger selection	1624_H	U, PV	U, PV	Page 20-206

Table 20-24 Register Overview (cont'd)

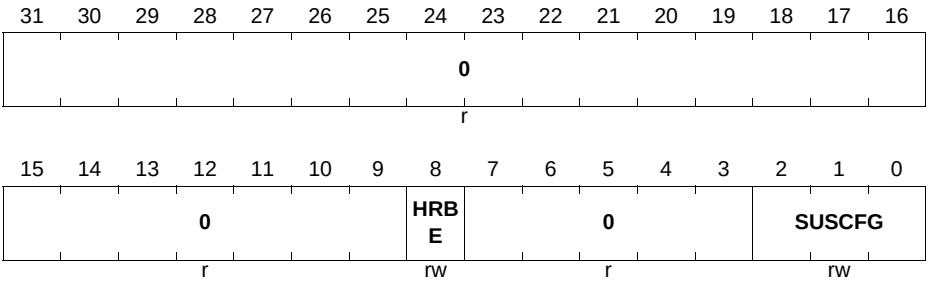
Short Name	Description	Offset Addr.	Access Mode		Description See
			Read	Write	
HRC3SDCR	Shadow rising dead time value	1628_H	U, PV	U, PV	Page 20-207
HRC3SDCF	Shadow falling dead time value	$162C_H$	U, PV	U, PV	Page 20-208
HRC3SCR1	Shadow high resolution compare value 1	1630_H	U, PV	U, PV	Page 20-208
HRC3SCR2	Shadow high resolution compare value 2	1634_H	U, PV	U, PV	Page 20-210
Reserved	Reserved	$163C_H$ - $16FC_H$	BE	BE	

20.7.1 Bias and Suspend Configuration Register

HRBSC

The register configures the suspend mode for the HRPWM unit and enables the bias generation.

HRBSC
Bias and suspend configuration (0900_H) Reset Value: 00000000_H

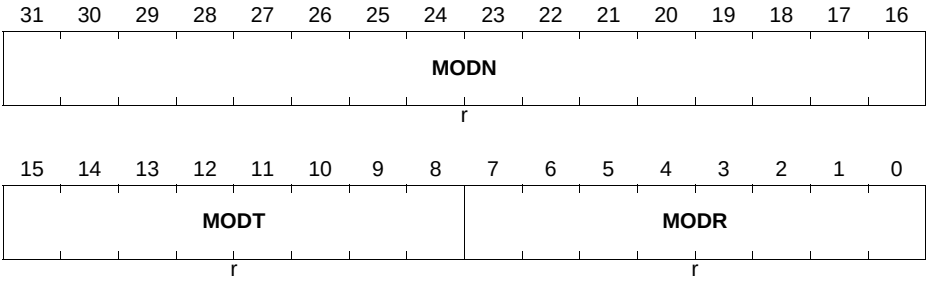


Field	Bits	Type	Description
SUSCFG	[2:0]	rw	**Suspend configuration** 000_B Suspend is ignored. 001_B CSGy and HRCy units are halted. 010_B Comparator outputs, HRPWMx.CyO are clamped to passive level and the CSGy units are halted. High resolution channel outputs, HRPWMx.HROUTy0 and HRPWMx.HROUTy1, are clamped to passive state and the HRCy units are halted. 011_B CSGy units are halted. High resolution channel outputs, HRPWMx.HROUTy0 and HRPWMx.HROUTy1, are clamped to passive state and the HRCy units are halted. 100_B Reserved. 101_B Reserved. 110_B Reserved. 111_B Reserved.
HRBE	8	rw	**HRPWM bias enable** Setting this field to 1_B enables the bias generation.
0	[31:9], [7:3]	r	**Reserved** Read always returns 0.

MIDR

This register contains the module identification number.

MIDR
Module identification register **(0908_H)** **Reset Value: $00A9C0XX_H$**

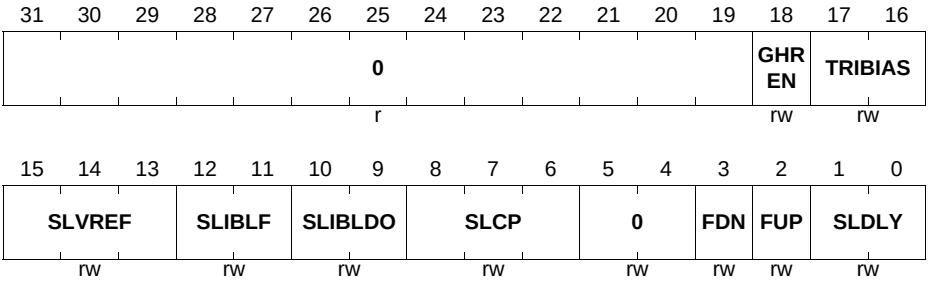


Field	Bits	Type	Description
MODR	[7:0]	r	**Module Revision** This bit field indicates the revision number of the module implementation (depending on the design step). The given value of 00_H is a placeholder for the actual number.
MODT	[15:8]	r	**Module Type**
MODN	[31:16]	r	**Module Number**

GLBANA

This register contains the global configuration for the analog units inside the HRPWM block. Only the GHREN bitfield needs to be addressed to bring up the high resolution generation logic - please address **Section 20.6** to understand if some other fields need to be written with values different from the ones imposed by reset.

GLBANA
Global Analog Configuration **(0914_H)** **Reset Value: $00004B8C_H$**



Field	Bits	Type	Description
SLDLY	[1:0]	rw	**Delay of lock detection** Please address **Section 20.6** to understand if this field needs to be update to a different value from the one imposed by reset.
FUP	2	rw	**Force chargepump up** No modification to the reset value should be done. If this register is accessed, this field should always be set to 1_B.

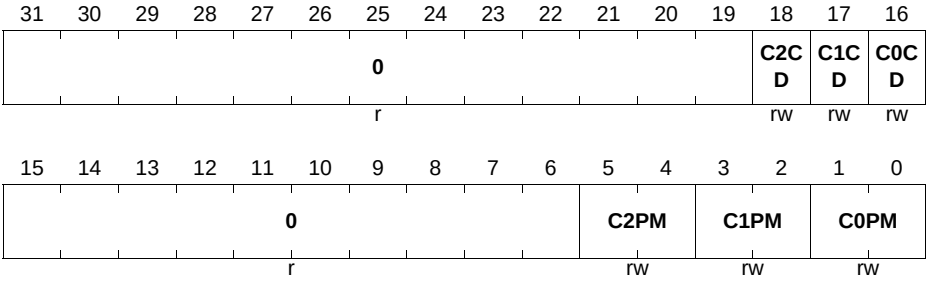
Field	Bits	Type	Description
FDN	3	rw	**Force chargepump down** No modification to the reset value should be done. If this register is accessed, this field should always be set to 1_B.
SLCP	[8:6]	rw	**HRCs chargepump current selection** Please address **Section 20.6** to understand if this field needs to be update to a different value from the one imposed by reset.
SLIBLDO	[10:9]	rw	**HRCs LDO bias current** No modification to the reset value should be done. If this register is accessed, this field should always be set to 01_B.
SLIBLF	[12:11]	rw	**HRCs loop filter bias current** No modification to the reset value should be done. If this register is accessed, this field should always be set to 01_B.
SLVREF	[15:13]	rw	**Reference voltage for chargepump and loop filter** No modification to the reset value should be done. If this register is accessed, this field should always be set to 010_B.
TRIBIAS	[17:16]	rw	**Bias trimming** No modification to the reset value should be done. If this register is accessed, this field should always be set to 00_B.
GHREN	18	rw	**Global High Resolution Enable** During the module initialization, this bitfield should be set to 1_B only after **HRCCFG**.HRCPM = 1_B, see **Section 20.6**. 0_B Global high resolution generation is disabled 1_B Global high resolution generation is enabled
0	[31:19], [5:4]	r	**Reserved** Read always returns 0

20.7.2 Global CSG Registers Description

CSGCFG

The register contains the global configuration for the power modes of all the CSGy units.

CSGCFG
Global CSG configuration **(0920_H)** **Reset Value: 00000000_H**



Field	Bits	Type	Description
C0PM	[1:0]	rw	**CSG0 Power Mode** This field configures the power mode for the subunit CSG0: 00_B CSG0 unit is powered OFF 01_B CSG0 unit is set in Low Speed Mode 10_B Reserved 11_B CSG0 unit is set in High Speed Mode
C1PM	[3:2]	rw	**CSG1 Power Mode** This field configures the power mode for the subunit CSG1: 00_B CSG1 unit is powered OFF 01_B CSG1 unit is set in Low Speed Mode 10_B Reserved 11_B CSG1 unit is set in High Speed Mode
C2PM	[5:4]	rw	**CSG2 Power Mode** This field configures the power mode for the subunit CSG2: 00_B CSG2 unit is powered OFF 01_B CSG2 unit is set in Low Speed Mode 10_B Reserved 11_B CSG2 unit is set in High Speed Mode
C0CD	16	rw	**CSG0 Clock disable** Setting this bitfield to 1_B disables the clock for the complete CSG0 subunit.

Field	Bits	Type	Description
C1CD	17	rw	**CSG1 Clock disable** Setting this bitfield to 1_B disables the clock for the complete CSG1 subunit.
C2CD	18	rw	**CSG2 Clock disable** Setting this bitfield to 1_B disables the clock for the complete CSG2 subunit.
0	[31:19], [15:6]	r	**Reserved** Read always returns 0

CSGSETG

The register contains the bit fields to set the run bits of each DAC control unit and comparator, as well the bit fields to set the comparator output into the clamped state.

CSGSETG
Global CSG run bit set **(0924_H)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0					**SC2P**	**SC2R**	**SD2R**	**0**	**SC1P**	**SC1R**	**SD1R**	**0**	**SC0P**	**SC0R**	**SD0R**
r					w	w	w	r	w	w	w	r	w	w	w

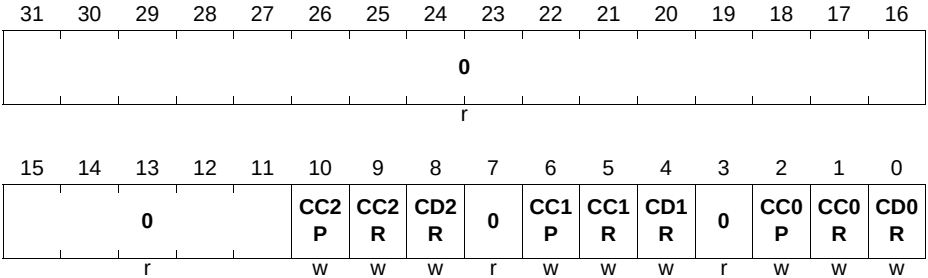
Field	Bits	Type	Description
SD0R	0	w	**DAC0 run bit set** Writing a 1_B to this bitfield sets the run bit for the DAC0. A read always returns 0_B.
SC0R	1	w	**CMP0 run bit set** Writing a 1_B to this bitfield sets the run bit for the CMP0. A read always returns 0_B.

Field	Bits	Type	Description
SC0P	2	w	**CMP0 passive level set** Writing a 1_B to this bitfield sets the output of CMP0 into the clamped state. A read always returns 0_B.
SD1R	4	w	**DAC1 run bit set** Writing a 1_B to this bitfield sets the run bit for the DAC1. A read always returns 0_B.
SC1R	5	w	**CMP1 run bit set** Writing a 1_B to this bitfield sets the run bit for the CMP1. A read always returns 0_B.
SC1P	6	w	**CMP1 passive level set** Writing a 1_B to this bitfield sets the output of CMP1 into the clamped state. A read always returns 0_B.
SD2R	8	w	**DAC2 run bit set** Writing a 1_B to this bitfield sets the run bit for the DAC2. A read always returns 0_B.
SC2R	9	w	**CMP2 run bit set** Writing a 1_B to this bitfield sets the run bit for the CMP2. A read always returns 0_B.
SC2P	10	w	**CMP2 passive level set** Writing a 1_B to this bitfield sets the output of CMP0 into the clamped state. A read always returns 0_B.
0	3, 7, [31:11]	r	**Reserved** Read always returns 0.

CSGCLRG

The register contains the bit fields to clear the run bits of each DAC control unit and comparator and also the bit fields to remove the comparator output from the clamped state.

CSGCLRG
Global CSG run bit clear **(0928_H)** **Reset Value: 00000000_H**



Field	Bits	Type	Description
CD0R	0	w	**DAC0 run bit clear** Writing a 1_B to this bitfield clears the run bit for the DAC0. A read always returns 0_B.
CC0R	1	w	**CMP0 run bit clear** Writing a 1_B to this bitfield clears the run bit for the CMP0. A read always returns 0_B.
CC0P	2	w	**CMP0 passive level clear** Writing a 1_B to this bitfield removes the output of CMP0 from the clamped state. A read always returns 0_B.
CD1R	4	w	**DAC1 run bit clear** Writing a 1_B to this bitfield clears the run bit for the DAC1. A read always returns 0_B.
CC1R	5	w	**CMP1 run bit clear** Writing a 1_B to this bitfield clears the run bit for the CMP1. A read always returns 0_B.
CC1P	6	w	**CMP1 passive level clear** Writing a 1_B to this bitfield removes the output of CMP1 from the clamped state. A read always returns 0_B.

Field	Bits	Type	Description
CD2R	8	w	**DAC2 run bit clear** Writing a 1_B to this bitfield clears the run bit for the DAC2. A read always returns 0_B.
CC2R	9	w	**CMP2 run bit clear** Writing a 1_B to this bitfield clears the run bit for the CMP2. A read always returns 0_B.
CC2P	10	w	**CMP2 passive level clear** Writing a 1_B to this bitfield removes the output of CMP2 from the clamped state. A read always returns 0_B.
0	3, 7, [31:11]	r	**Reserved** Read always returns 0.

CSGSTATG

The register contains the run bit status of each DAC control unit, the automatic Comparator input switching and the clamping status information.

CSGSTATG
Global CSG run bit status **(092C$_H$)** **Reset Value: 00000000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0					**PSLS2**	**C2RB**	**D2RB**	**0**	**PSLS1**	**C1RB**	**D1RB**	**0**	**PSLS0**	**C0RB**	**D0RB**
r					rh	rh	rh	r	rh	rh	rh	r	rh	rh	rh

Field	Bits	Type	Description
D0RB	0	rh	**DAC0 run bit status** 0_B DAC0 is not running (control logic is disabled) 1_B DAC0 is running

Field	Bits	Type	Description
C0RB	1	rh	**CMP0 run bit status** 0_B CMP0 functionality is disabled 1_B CMP0 functionality is enabled
PSLS0	2	rh	**CMP0 output passive status** 0_B CMP0 output is not clamped 1_B CMP0 output is clamped
D1RB	4	rh	**DAC1 run bit status** 0_B DAC1 is not running (control logic is disabled) 1_B DAC1 is running
C1RB	5	rh	**CMP1 run bit status** 0_B CMP1 functionality is disabled 1_B CMP1 functionality is enabled
PSLS1	6	rh	**CMP1 output passive status** 0_B CMP1 output is not clamped 1_B CMP1 output is clamped
D2RB	8	rh	**DAC2 run bit status** 0_B DAC2 is not running (control logic is disabled) 1_B DAC1 is running
C2RB	9	rh	**CMP2 run bit status** 0_B CMP2 functionality is disabled 1_B CMP2 functionality is enabled
PSLS2	10	rh	**CMP2 output passive status** 0_B CMP2 output is not clamped 1_B CMP2 output is clamped
0	3, 7, [31:11]	r	**Reserved** Read always returns 0.

CSGFCG

The register contains the bit fields for SW control of the stop/clear/start of the slope generation and associated prescaler of each CSGy subunit. This bitfields can be used to replace the functionality of the external control signals of each CSGy subunit.

CSGFCG
Global CSG slope/prescaler control (0930$_H$) **Reset Value: 00000000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0											PS2CLR	PS2STP	PS2STR	S2STP	S2STR
r											w	w	w	w	w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0			PS1CLR	PS1STP	PS1STR	S1STP	S1STR	0			PS0CLR	PS0STP	PS0STR	S0STP	S0STR
r			w	w	w	w	w	r			w	w	w	w	w

Field	Bits	Type	Description
S0STR	0	w	**Slope 0 start** Writing a 1_B to this bitfield starts (re start if generation is ongoing) of the slope generation for DAC0. A read always returns 0_B.
S0STP	1	w	**Slope 0 stop** Writing a 1_B to this bitfield stops the slope generation for DAC0. A read always returns 0_B.
PS0STR	2	w	**Prescaler 0 start** Writing a 1_B to this bitfield starts the prescaler inside the CSG0. A read always returns 0_B.
PS0STP	3	w	**Prescaler 0 stop** Writing a 1_B to this bitfield stops the prescaler inside the CSG0. A read always returns 0_B.
PS0CLR	4	w	**Prescaler 0 clear** Writing a 1_B to this bitfield clears the prescaler inside the CSG0 (the prescaler is not stopped). A read always returns 0_B.
S1STR	8	w	**Slope 1 start** Writing a 1_B to this bitfield starts (re start if generation is ongoing) of the slope generation for DAC1. A read always returns 0_B.

Field	Bits	Type	Description
S1STP	9	w	**Slope 1 stop** Writing a 1_B to this bitfield stops the slope generation for DAC1. A read always returns 0_B.
PS1STR	10	w	**Prescaler 1 start** Writing a 1_B to this bitfield starts the prescaler inside the CSG1. A read always returns 0_B.
PS1STP	11	w	**Prescaler 1 stop** Writing a 1_B to this bitfield stops the prescaler inside the CSG1. A read always returns 0_B.
PS1CLR	12	w	**Prescaler 1 clear** Writing a 1_B to this bitfield clears the prescaler inside the CSG1 (the prescaler is not stopped). A read always returns 0_B.
S2STR	16	w	**Slope 2 start** Writing a 1_B to this bitfield starts (re start if generation is ongoing) of the slope generation for DAC2. A read always returns 0_B.
S2STP	17	w	**Slope 2 stop** Writing a 1_B to this bitfield stops the slope generation for DAC2. A read always returns 0_B.
PS2STR	18	w	**Prescaler 2 start** Writing a 1_B to this bitfield starts the prescaler inside the CSG2. A read always returns 0_B.
PS2STP	19	w	**Prescaler 2 stop** Writing a 1_B to this bitfield stops the prescaler inside the CSG2. A read always returns 0_B.
PS2CLR	20	w	**Prescaler 2 clear** Writing a 1_B to this bitfield clears the prescaler inside the CSG2 (the prescaler is not stopped). A read always returns 0_B.

Field	Bits	Type	Description
0	[7:5], [15:13], [31:21]	r	**Reserved** Read always returns 0.

CSGFSG

The register contains the status of each slope generation and prescaler.

CSGFSG
Global CSG slope/prescaler status (0934$_H$) **Reset Value: 00000000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0														P2RB	S2RB
r														rh	rh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0						P1RB	S1RB	0						P0RB	S0RB
r						rh	rh	r						rh	rh

Field	Bits	Type	Description
S0RB	0	rh	**DAC0 slope generation status** 0_B Slope generation is stopped. 1_B Slope generation is running.
P0RB	1	rh	**CSG0 prescaler status** 0_B Prescaler is stopped. The clock used for the slope generation is halted and therefore the slope is frozen. 1_B Prescaler is running.
S1RB	8	rh	**DAC1 slope generation status** 0_B Slope generation is stopped. 1_B Slope generation is running.
P1RB	9	rh	**CSG1 prescaler status** 0_B Prescaler is stopped. The clock used for the slope generation is halted and therefore the slope is frozen. 1_B Prescaler is running.

Field	Bits	Type	Description
S2RB	16	rh	**DAC2 slope generation status** 0_B Slope generation is stopped. 1_B Slope generation is running.
P2RB	17	rh	**CSG2 prescaler status** 0_B Prescaler is stopped. The clock used for the slope generation is halted and therefore the slope is frozen. 1_B Prescaler is running.
0	[7:2], [15:10], [31:18]	r	**Reserved** Read always returns 0.

CSGTRG

The register contains the bitfields that are used to trigger a shadow transfer or a switch on the comparator input.

CSGTRG
Global CSG shadow/switch trigger (0938_H) **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0						D2SVS	D2SES	0		D1SVS	D1SES	0		D0SVS	D0SES
r						w	w	r		w	w	r		w	w

Field	Bits	Type	Description
D0SES	0	w	**DAC0 shadow transfer enable set** Writing a 1_B to this bitfield, enable a shadow transfer for the **CSGyDSV1** and **CSGyPC** registers of CSG0. Bitfield **CSGTRSG**.D0STE will be set to 1_B enabling this way the shadow transfer mechanism. A read always returns 0_B.

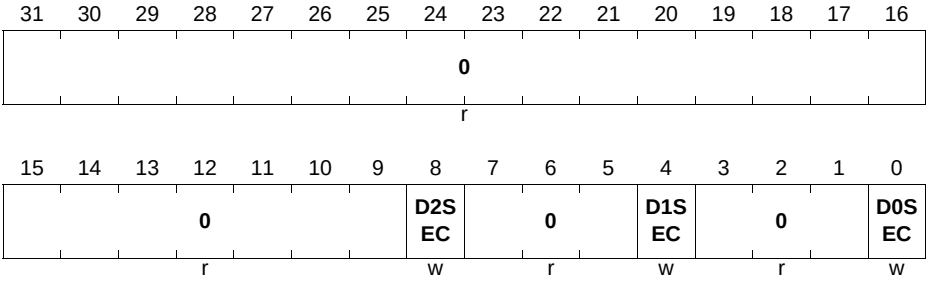
Field	Bits	Type	Description
D0SVS	1	w	**CMP0 input switch request** Writing a 1_B to this bitfield, requests a switch on the analog input connected to the input of the CMP0. This action only has effect if **CSGFSG**.S0RB = 0_B. A read always returns 0_B.
D1SES	4	w	**DAC1 shadow transfer enable set** Writing a 1_B to this bitfield, enable a shadow transfer for the **CSGyDSV1** and **CSGyPC** registers of CSG1. Bitfield **CSGTRSG**.D1STE will be set to 1_B enabling this way the shadow transfer mechanism. A read always returns 0_B.
D1SVS	5	w	**CMP1 input switch request** Writing a 1_B to this bitfield, requests a switch on the analog input connected to the input of the CMP1. This action only has effect if **CSGFSG**.S1RB = 0_B. A read always returns 0_B.
D2SES	8	w	**DAC2 shadow transfer enable set** Writing a 1_B to this bitfield, enable a shadow transfer for the **CSGyDSV1** and **CSGyPC** registers of CSG2. Bitfield **CSGTRSG**.D2STE will be set to 1_B enabling this way the shadow transfer mechanism. A read always returns 0_B.
D2SVS	9	w	**CMP2 input switch request** Writing a 1_B to this bitfield, requests a switch on the analog input connected to the input of the CMP2. This action only has effect if **CSGFSG**.S2RB = 0_B. A read always returns 0_B.
0	[3:2], [7:6], [31:10]	r	**Reserved** Read always returns 0.

CSGTRC

The register contains the bitfields that are used to trigger a shadow transfer or a switch on the comparator input.

CSGTRC

Global CSG shadow trigger clear (093C$_H$) **Reset Value: 00000000$_H$**

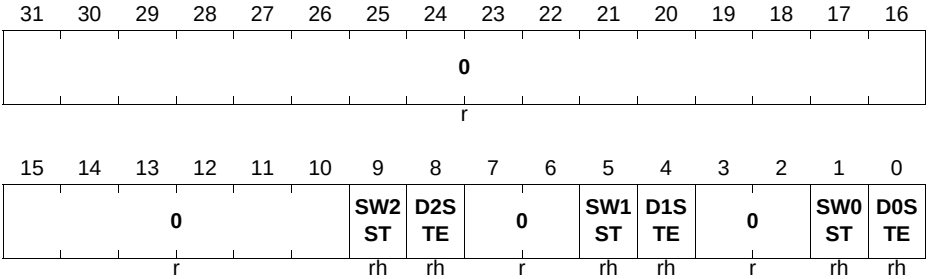


Field	Bits	Type	Description
D0SEC	0	w	**DAC0 shadow transfer enable clear** Writing a 1_B to this bitfield, clears the shadow transfer enable bitfield, **CSGTRSG**.D0STE. A read always returns 0_B.
D1SEC	4	w	**DAC1 shadow transfer enable clear** Writing a 1_B to this bitfield, clears the shadow transfer enable bitfield, **CSGTRSG**.D1STE. A read always returns 0_B.
D2SEC	8	w	**DAC2 shadow transfer enable clear** Writing a 1_B to this bitfield, clears the shadow transfer enable bitfield, **CSGTRSG**.D2STE. A read always returns 0_B.
0	[3:1], [7:5], [31:9]	r	**Reserved** Read always returns 0.

CSGTRSG

The register contains the status of the shadow transfer request and the status of the analog input connection.

CSGTRSG
Global CSG shadow/switch status (0940$_H$) **Reset Value: 00000000$_H$**



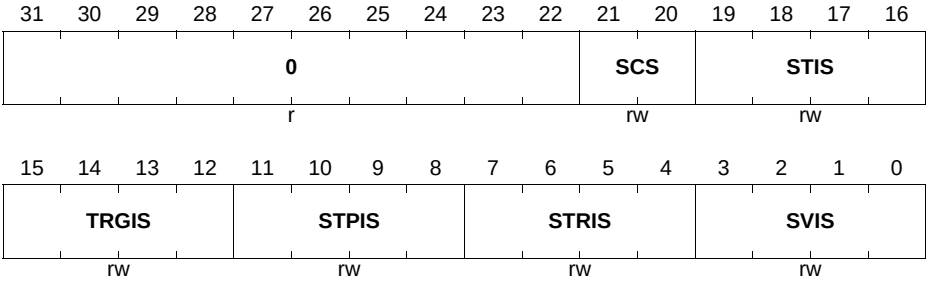
Field	Bits	Type	Description
D0STE	0	rh	**DAC0 shadow transfer enable** 0_B Shadow transfer has been performed. 1_B Shadow transfer has been requested but is still pending completion.
SW0ST	1	rh	**CMP0 input connection status** 0_B Input connected to HRPWMx.C0I[A] 1_B Input connected to HRPWMx.C0I[B]
D1STE	4	rh	**DAC1 shadow transfer enable** 0_B Shadow transfer has been performed. 1_B Shadow transfer has been requested but is still pending completion.
SW1ST	5	rh	**CMP1 input connection status** 0_B Input connected to HRPWMx.C1I[A] 1_B Input connected to HRPWMx.C1I[B]
D2STE	8	rh	**DAC2 shadow transfer enable** 0_B Shadow transfer has been performed. 1_B Shadow transfer has been requested but is still pending completion.
SW2ST	9	rh	**CMP2 input connection status** 0_B Input connected to HRPWMx.C2I[A] 1_B Input connected to HRPWMx.C2I[B]
0	[3:2], [7:6], [31:10]	r	**Reserved** Read always returns 0.

20.7.3 CSGy Registers Description

CSGyDCI

Within this register is possible to configure which external input is being used for each function (slope start, slope stop, shadow transfer request, etc).

CSGyDCI (y = 0 - 2)
External input selection **($0A00_H + 0100_H * y$)** **Reset Value: 00000000_H**



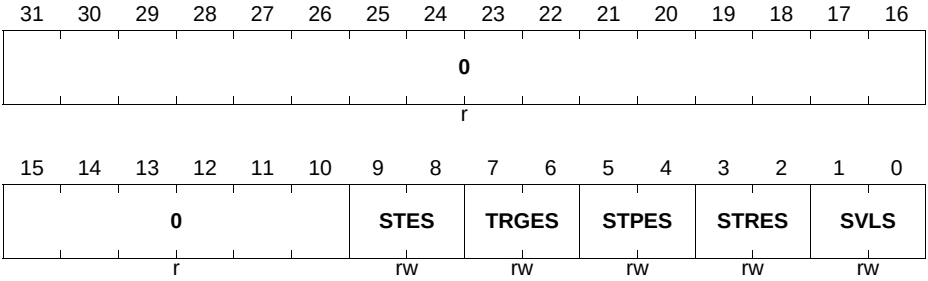
Field	Bits	Type	Description
SVIS	[3:0]	rw	**Value Selector input selection** This field selects which pin is used for the value selector function: 0000_B HRPWMx.SCyIA 0001_B HRPWMx.SCyIB 0010_B HRPWMx.SCyIC 0011_B HRPWMx.SCyID 0100_B HRPWMx.SCyIE 0101_B HRPWMx.SCyIF 0110_B HRPWMx.SCyIG 0111_B HRPWMx.SCyIH 1000_B HRPWMx.SCyII 1001_B HRPWMx.SCyIJ 1010_B HRPWMx.SCyIK 1011_B HRPWMx.SCyIL 1100_B HRPWMx.SCyIM 1101_B HRPWMx.SCyIN 1110_B HRPWMx.SCyIO 1111_B HRPWMx.SCyIP

Field	Bits	Type	Description
STRIS	[7:4]	rw	**Slope generation start control input selection** This field selects which pin is used as slope/prescaler start function. The input mapping codes are the same as for the SVIS field.
STPIS	[11:8]	rw	**Slope generation stop control input selection** This field selects which pin is used as slope/prescaler stop function. The input mapping codes are the same as for the SVIS field.
TRGIS	[15:12]	rw	**External conversion trigger input selection** This field selects which pin is used as conversion trigger function. The input mapping codes are the same as for the SVIS field.
STIS	[19:16]	rw	**External shadow request enable input selection** This field selects which pin is used as external shadow transfer enable function. The input mapping codes are the same as for the SVIS field.
SCS	[21:20]	rw	**Slope generation clock selection** This field selects which clock source is being used to generate the slope: 00_B HRPWMx.MCLK (Module clock is used) 01_B HRPWMx.ECLKA (External clock is used) 10_B HRPWMx.ECLKB (External clock is used) 11_B HRPWMx.ECLKC (External clock is used)
0	[31:22]	r	**Reserved** Read always returns 0.

CSGyIES

In this register one can configure the active edge(s) or level for all the external functions, linked with the pins HRPWMx.SyI[P...A].

CSGyIES (y = 0 - 2)
External input selection **($0A04_H + 0100_H * y$)** **Reset Value: 00000000_H**



Field	Bits	Type	Description
SVLS	[1:0]	rw	**External value switch function level selection** This field selects the active level to control which of the two reference values, **CSGyDSV1** and **CSGyDSV2** is fed to the DAC: 00_B Function disabled 01_B Active when input is HIGH 10_B Active when input is LOW 11_B Reserved
STRES	[3:2]	rw	**External start function edge selection** This field selects the active edge to control the start function for the prescaler/slope generation: 00_B Function disabled 01_B Active on rising edge 10_B Active on falling edge 11_B Active on both edges
STPES	[5:4]	rw	**External stop function edge selection** This field selects the active edge to control the stop function for the prescaler/slope generation: 00_B Function disabled 01_B Active on rising edge 10_B Active on falling edge 11_B Active on both edges

Field	Bits	Type	Description
TRGES	[7:6]	rw	**External trigger function edge selection** This field selects the active edge to control the DAC conversion trigger function: 00_B Function disabled 01_B Active on rising edge 10_B Active on falling edge 11_B Active on both edges
STES	[9:8]	rw	**External shadow transfer enable edge selection** This field selects the edge that is used to enable the shadow transfer operation (shadow transfer request still needs to be requested by SW): 00_B Function disabled 01_B Active on rising edge 10_B Active on falling edge 11_B Active on both edges
0	[31:10]	r	**Reserved** Read always returns 0.

CSGySC

The different modes for the slope generation can be configured through this register.

CSGySC (y = 0 - 2)
Slope generation control **($0A08_H + 0100_H * y$)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0						PSWM		0		PSE	IST	GCFG		SWSM	
r						rw		r		rw	rw	rw		rw	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SVSC		SSTM		SSRM		SCM		0	PSV		FPD	PSTM		PSRM	
rw		rw		rw		rw		r	rw		rw	rw		rw	

Field	Bits	Type	Description
PSRM	[1:0]	rw	**Prescaler external start configuration** This field controls the operation mode for the external start linked with the prescaler: 00_B External start trigger is ignored 01_B Start prescaler 10_B Clear prescaler 11_B Clear & Start prescaler
PSTM	[3:2]	rw	**Prescaler external stop configuration** This field controls the operation mode for the external stop linked with the prescaler: 00_B External stop trigger is ignored 01_B Stop prescaler 10_B Clear prescaler 11_B Clear & Stop prescaler
FPD	4	rw	**Fixed division disable** This field disables the fixed division by 4: 0_B Division by 4 enabled 1_B Division by 4 disabled
PSV	[6:5]	rw	**Prescaler division factor** This field configures the division factor for the prescaler: 00_B division by 1 01_B division by 2 10_B division by 4 11_B division by 8
SCM	[9:8]	rw	**Slope control mode** The different slope generations schemes can be selected via this field: 00_B Slope generation disabled. Used when the switch between the two reference values, **CSGyDSV1** and **CSGyDSV2** is done via external signal. 01_B Decrementing slope generation. 10_B Incrementing slope generation. 11_B Triangular slope generation.

Field	Bits	Type	Description
SSRM	[11:10]	rw	**Slope external start configuration** This field controls the operation mode for the external start linked with the slope generation: 00_B External start trigger is ignored 01_B Start/restart slope generation 10_B Resumes slope 11_B Reserved
SSTM	[13:12]	rw	**Slope external stop configuration** This field controls the operation mode for the external start linked with the slope generation: 00_B External stop trigger is ignored 01_B Stops/Halts the slope generation 10_B Used in hybrid mode. It freezes the slope generation and feeds constantly the value programmed in **CSGyDSV2** to the DAC. 11_B Reserved

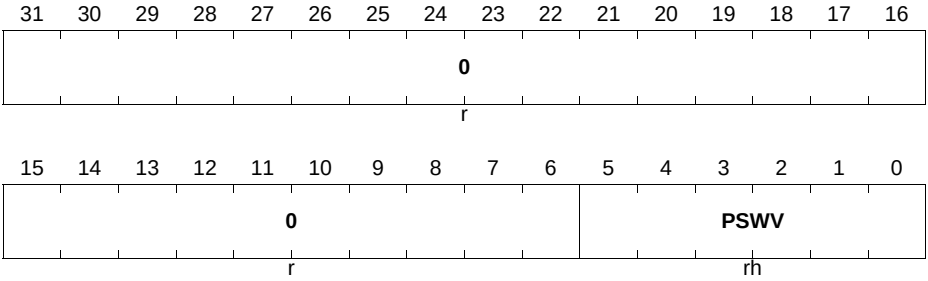
Field	Bits	Type	Description
SVSC	[15:14]	rw	**Slope reference value mode** This field controls the way that the two reference values, **CSGyDSV1** and **CSGyDSV2** are being used when the HW slope generation is enabled: 00_B Only **CSGyDSV1** value is used for the slope generation: if slope is incrementing, **CSGyDSV1** is the bottom reference value from where the ramp starts; if decrementing, then **CSGyDSV1** is the upper reference value from where the ramp starts. 01_B The two reference values are being used: **CSGyDSV1** is the low or high reference value from where the ramp starts (incrementing or decrementing respectively); **CSGyDSV2** is used as a static value (this value is constantly fed to the DAC after a stop trigger as been detected). 10_B The two reference values are used: **CSGyDSV1** is the low or high reference value from where the slope starts (incrementing or decrementing respectively); **CSGyDSV2** is used as an internal re start condition for the slope. 11_B Reserved *Note: This field needs to be programmed accordingly when **CSGySC**.SCM is different from 00_B and 11_B (if **CSGySC**.SCM is 00_B or 11_B the value on the SVSC is discarded).*
SWSM	[17:16]	rw	**Initial DAC start mode** This field controls the initial conversion of the DAC after the specific run bit is set (DyRB): 00_B **CSGyDSV2** is fed to the DAC and initial conversion trigger is generated. 01_B **CSGyDSV1** is fed to the DAC and initial conversion trigger is generated. 10_B **CSGyDSV2** is fed to the DAC but initial conversion trigger is not generated. 11_B **CSGyDSV1** is fed to the DAC but initial conversion trigger is not generated.

Field	Bits	Type	Description
GCFG	[19:18]	rw	**Slope step gain configuration** 00_B Each slope step has an increment/decrement of 1_D 01_B Each slope step has an increment/decrement of 2_D 10_B Each slope step has an increment/decrement of 4_D 11_B Each slope step has an increment/decrement of 8_D
IST	20	rw	**Immediate shadow transfer** Setting this bitfield to 1_B will enable a immediate shadow transfer. This means that the update of the specific registers will happen immediately after a shadow transfer has been requested. An update and conversion of the CSGyDSV1 value is done immediately by the specific DAC. This is only valid if **CSGySC**.SCM is equal to 00_B.
PSE	21	rw	**Pulse swallow enable** 0_B Pulse swallow disabled 1_B Pulse swallow enabled
PSWM	[25:24]	rw	**Pulse swallow window mode** 00_B 16 clock cycle window 01_B 32 clock cycle window 10_B 64 clock cycle window 11_B Reserved
0	7, [23:22], [31:26]	r	**Reserved** Read always returns 0.

CSGyPC

This register configures the number of clock pulses that should be swallowed in case that the bitfield PSE = 1_B.

CSGyPC (y = 0 - 2)
Pulse swallow configuration (0A0C$_H$ + 0100$_H$ * y) Reset Value: 00000000$_H$

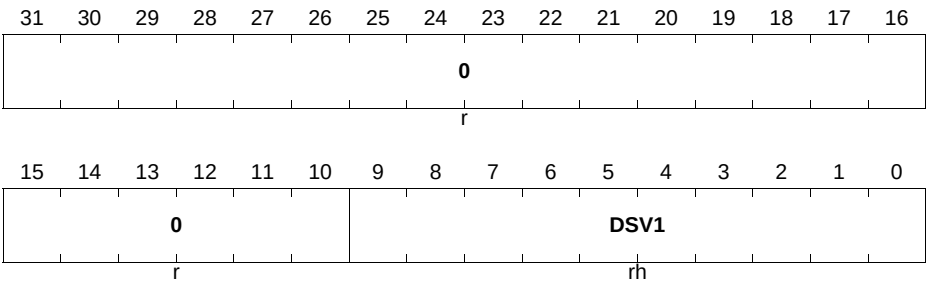


Field	Bits	Type	Description
PSWV	[5:0]	rh	**Pulse swallow configuration** The number of clock pulses to swallow is configured in this field. This has a full decimal mapping an therefore programming this field with the value n_D means that n clock cycles are going to be swallowed. The **CSGySC**.PSWM controls the number of bits that are being used from the PSWV: 00_B - 4 LSBs 01_B - 5 LSBs 10_B - 6 LSBs
0	[31:6]	r	**Reserved** Read always returns 0.

CSGyDSV1

This register contains the actual value used for the DSV1 reference.

CSGyDSV1 (y = 0 - 2)
DAC reference value 1 **($0A10_H + 0100_H * y$)** **Reset Value: 00000000_H**

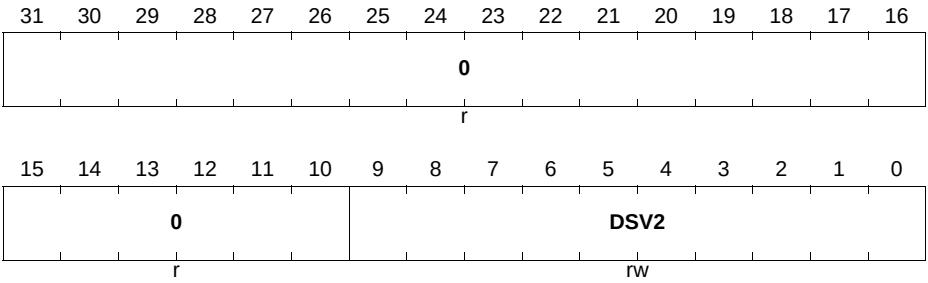


Field	Bits	Type	Description
DSV1	[9:0]	rh	**DAC reference value 1** This is the actual value used for the DAC reference value 1 (decimal mapping). 000_H is translated as minimum value and FFF_H as maximum value.
0	[31:10]	r	**Reserved** Read always returns 0.

CSGyDSV2

This register contains the actual value used for the DSV2 reference.

CSGyDSV2 (y = 0 - 2)
DAC reference value 1 **($0A14_H + 0100_H * y$)** **Reset Value: 00000000_H**

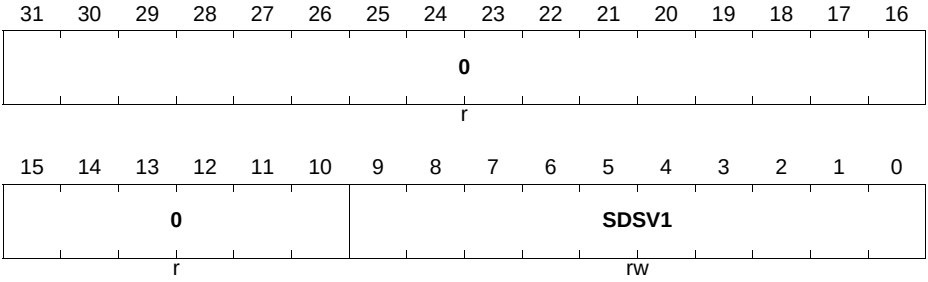


Field	Bits	Type	Description
DSV2	[9:0]	rw	**DAC reference value 2** This is the actual value used for the DAC reference value 2 (decimal mapping). 000_H is translated as minimum value and FFF_H as maximum value.
0	[31:10]	r	**Reserved** Read always returns 0.

CSGySDSV1

This is the shadow for the DSV1 reference. The update of the DSV1 field needs to be done via this address.

CSGySDSV1 (y = 0 - 2)
Shadow reference value 1 **($0A18_H + 0100_H * y$)** **Reset Value: 00000000_H**

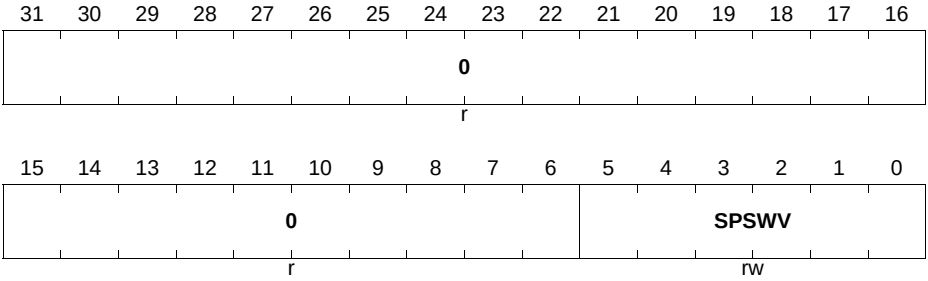


Field	Bits	Type	Description
SDSV1	[9:0]	rw	**Shadow DAC reference value 1** This is the value that is going to be loaded into the **CSGyDSV1**.DSV1 field after a shadow transfer has been performed. 000_H is translated as minimum value and FFF_H as maximum value.
0	[31:10]	r	**Reserved** Read always returns 0.

CSGySPC

This is the shadow register for the PSWV field. The update of the PSWV field needs to be done via this address.

CSGySPC (y = 0 - 2)
Shadow Pulse swallow value (0A1C$_H$ + 0100$_H$ * y) Reset Value: 00000000$_H$

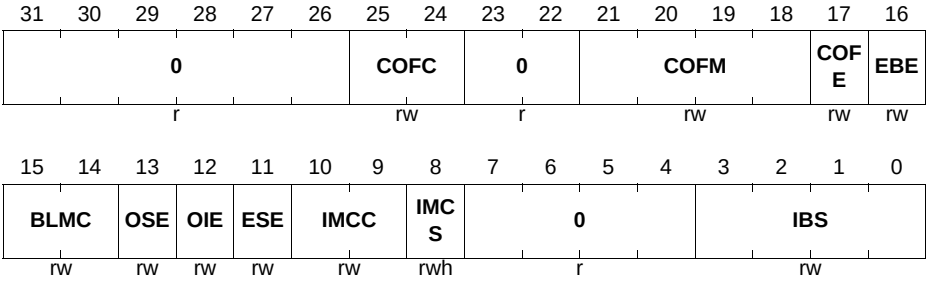


Field	Bits	Type	Description
SPSWV	[5:0]	rw	**Shadow pulse swallow value** This is the value that is going to be loaded into the **CSGyPC**.PSWV field after a shadow transfer has been performed.
0	[31:6]	r	**Reserved** Read always returns 0.

CSGyCC

This register contains the fields to configure the comparator operation.

CSGyCC (y = 0 - 2)
Comparator configuration (0A20$_H$ + 0100$_H$ * y) Reset Value: 00000000$_H$



Field	Bits	Type	Description
IBS	[3:0]	rw	**External blanking trigger selector** This configures which pin can be used as trigger for the blanking period (bitfield EBE needs to be programmed with 1_B for this configuration to be active): 0000_B HRPWMx.BLyA 0001_B HRPWMx.BLyB 0010_B HRPWMx.BLyC 0011_B HRPWMx.BLyD 0100_B HRPWMx.BLyE 0101_B HRPWMx.BLyF 0110_B HRPWMx.BLyG 0111_B HRPWMx.BLyH 1000_B HRPWMx.BLyI 1001_B HRPWMx.BLyJ 1010_B HRPWMx.BLyK 1011_B HRPWMx.BLyL 1100_B HRPWMx.BLyM 1101_B HRPWMx.BLyN 1110_B HRPWMx.BLyO 1111_B HRPWMx.BLyP
IMCS	8	rwh	**Comparator input selector** This configures which pin is connected to the input of the comparator: 0_B HRPWMx.CyINA 1_B HRPWMx.CyINB
IMCC	[10:9]	rw	**Comparator input switching configuration** Configuration for the dynamic comparator input switching: 00_B Dynamic switch disabled 01_B Comparator input is connected to HRPWMx.CyINB when the control signal is HIGH 10_B Comparator input is connected to HRPWMx.CyINA when the control signal is HIGH 11_B Reserved.

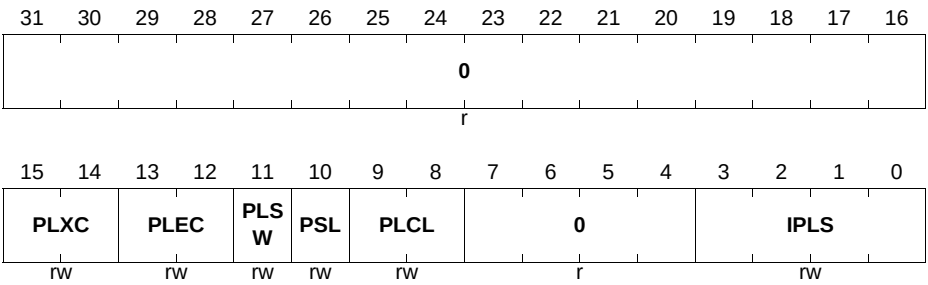
Field	Bits	Type	Description
ESE	11	rw	**External triggered switch enable** Setting this bitfield to 1_B enables the control of the switching between the two comparator inputs, with the external signal mapped to the blanking trigger. *Note: This doesn't impose that the EBE bitfield needs to be set to 1_B*
OIE	12	rw	**Comparator output inversion enable** Setting this bitfield to 1_B inverts the output of the comparator
OSE	13	rw	**Comparator output synchronization enable** When this bitfield is set to 1_B, the output of the comparator is synchronized with the module clock. Note that if the blanking is enabled, this bitfield becomes redundant.
BLMC	[15:14]	rw	**Blanking mode** This field controls when the blanking period is started (the blanking period can be started internally, from the comparator output or from a signal connected to HRPWMx.BLy[P...A]): 00_B Blanking disabled 01_B Blanking on a LOW to HIGH transition 10_B Blanking on a HIGH to LOW transition 11_B Blanking on both transitions
EBE	16	rw	**External blanking trigger enabled** Setting this bitfield to 1_B enables the usage of an external trigger to start the blanking period. The configuration of which external signal is used, must be programmed into the IBS field.
COFE	17	rw	**Comparator output filter enable** 0_B Filtering stage disabled 1_B Filtering stage enabled

Field	Bits	Type	Description
COFM	[21:18]	rw	**Comparator output filter window** 0000_B Comparator Output needs to be stable for 2 clock cycles 0001_B Comparator Output needs to be stable for 3 clock cycles 0010_B Comparator Output needs to be stable for 4 clock cycles 0011_B Comparator Output needs to be stable for 5 clock cycles ... 1100_B Comparator Output needs to be stable for 14 clock cycles 1101_B Comparator Output needs to be stable for 15 clock cycles 1110_B Comparator Output needs to be stable for 16 clock cycles 1111_B Comparator Output needs to be stable for 32 clock cycles
COFC	[25:24]	rw	**Comparator output filter control** 00_B Filtering is always done if enabled 01_B Filtering is only done when CSGyDSV1 value is currently fed to the DAC 10_B Filtering is only done when the CSGyDSV2 value is currently fed to the DAC 11_B Reserved *Note: This configuration only has effect when the Slope Control block is set to static or hybrid mode.*
0	[31:26], [23:22], [7:4]	r	**Reserved** Read always returns 0.

CSGyPLC

This register contains the fields to configure the comparator passive level operation.

CSGyPLC (y = 0 - 2)
Passive level configuration **($0A24_H + 0100_H * y$)** **Reset Value: 00000000_H**



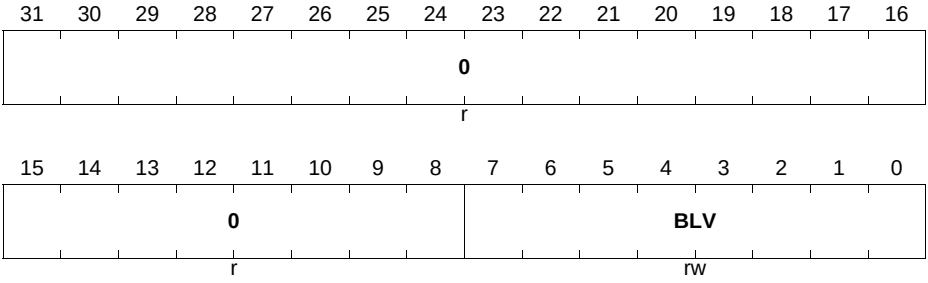
Field	Bits	Type	Description
IPLS	[3:0]	rw	**Clamping control signal selector** This configures which pin can be used to control/set the output in the clamped state: 0000_B HRPWMx.BLyA 0001_B HRPWMx.BLyB 0010_B HRPWMx.BLyC 0011_B HRPWMx.BLyD 0100_B HRPWMx.BLyE 0101_B HRPWMx.BLyF 0110_B HRPWMx.BLyG 0111_B HRPWMx.BLyH 1000_B HRPWMx.BLyI 1001_B HRPWMx.BLyJ 1010_B HRPWMx.BLyK 1011_B HRPWMx.BLyL 1100_B HRPWMx.BLyM 1101_B HRPWMx.BLyN 1110_B HRPWMx.BLyO 1111_B HRPWMx.BLyP
PLCL	[9:8]	rw	**Clamping control signal level selection** 00_B Clamping control disabled 01_B Output is set to clamped level when the control signal is HIGH 10_B Output is set to clamped level when the control signal is LOW 11_B Reserved

Field	Bits	Type	Description
PSL	10	rw	**Output passive level value** 0_B Output clamped level is LOW 1_B Output clamped level is HIGH
PLSW	11	rw	**Clamped state exit SW configuration** 0_B External signal and SW can remove the output from the clamped state 1_B Only SW can remove the output from the clamped state
PLEC	[13:12]	rw	**Passive level enter configuration** 00_B Passive level is entered immediately 01_B Passive level is entered only after the comparator output passes to LOW (output from the blanking stage) 10_B Passive level is entered only after the comparator output passes to HIGH (output from the blanking stage)
PLXC	[15:14]	rw	**Passive level exit configuration** 00_B Passive level is exit immediately 01_B Passive level is exit only after the comparator output passes to LOW (output from the blanking stage) 10_B Passive level is exit only after the comparator output passes to HIGH (output from the blanking stage)
0	[31:16], [7:4]	r	**Reserved** Read always returns 0.

CSGyBLV

This register contains the blanking period value.

CSGyBLV (y = 0 - 2)
Comparator blanking value **($0A28_H + 0100_H * y$)** **Reset Value: 00000000_H**

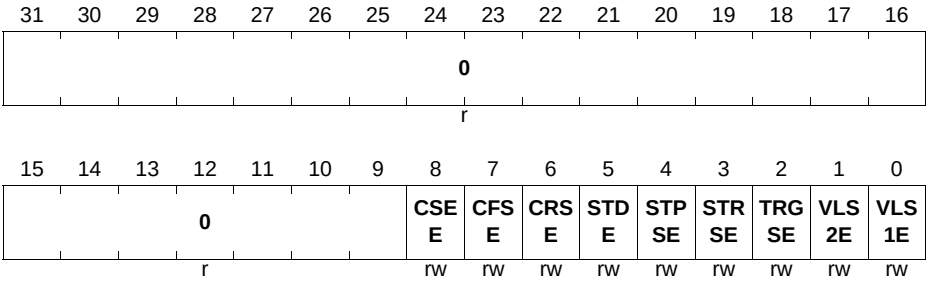


Field	Bits	Type	Description
BLV	[7:0]	rw	**Blanking value** Decimal value for the blanking period. The resolution of the blanking period is dictated by the frequency of the module clock: <blanking_value> = BLV x module_clock_period
0	[31:8]	r	**Reserved** Read always returns 0.

CSGySRE

In this register the different service request sources can be individually enabled/disabled.

CSGySRE (y = 0 - 2)
Service request enable **($0A2C_H + 0100_H * y$)** **Reset Value: 00000000_H**



Field	Bits	Type	Description
VLS1E	0	rw	**Value switch from CSGyDSV1 to CSGyDSV2 interrupt enable** Set this bitfield to 1_B enables the generation of an interrupt pulse whenever: • the value fed to the DAC switches from **CSGyDSV1** to **CSGyDSV2** • the slope generation transits from incrementing to decrementing (triangular waveform only)
VLS2E	1	rw	**Value switch from CSGyDSV2 to CSGyDSV1 interrupt enable** Set this bitfield to 1_B enables the generation of an interrupt pulse whenever: • the value fed to the DAC switches from **CSGyDSV2** to **CSGyDSV1** • the slope generation transits from decrementing to incrementing (triangular waveform only)
TRGSE	2	rw	**Conversion trigger interrupt enable** Setting this bitfield to 1_B enables the generation of an interrupt pulse whenever a DAC conversion trigger is generated.
STRSE	3	rw	**Start trigger interrupt enable** Setting this bitfield to 1_B enables the generation of an interrupt pulse every time that a slope generation start action is performed.
STPSE	4	rw	**Stop trigger interrupt enable** Setting this bitfield to 1_B enables the generation of an interrupt pulse every time that a slope generation stop action is performed.
STDE	5	rw	**Shadow transfer done interrupt enable** Setting this bitfield to 1_B enables the generation of an interrupt pulse every time that a shadow transfer has been performed.
CRSE	6	rw	**Comparator rise interrupt enable** Setting this bitfield to 1_B enables the generation of an interrupt pulse every time the comparator output changes from LOW to HIGH.

Field	Bits	Type	Description
CFSE	7	rw	**Comparator fall interrupt enable** Setting this bitfield to 1_B enables the generation of an interrupt pulse every time the comparator output changes from HIGH to LOW.
CSEE	8	rw	**Clamped state interrupt enable** Setting this bitfield to 1_B enables the generation of an interrupt pulse every time the comparator output is set into the clamped state.
0	[31:9]	r	**Reserved** Read always returns 0.

CSGySRS

In this register is possible to configure to which CSGy service request output (each CSGy has 4 outputs) each interrupt is forward.

CSGySRS (y = 0 - 2)
Service request line selector **($0A30_H$ + 0100_H * y)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		CSLS		CRFLS		STLS		SSLS		TRLS		VLS2S		VLS1S	
r		rw		rw		rw		rw		rw		rw		rw	

Field	Bits	Type	Description
VLS1S	[1:0]	rw	**Value switch from CSGyDSV1 to CSGyDSV2 interrupt line selection** This field programs to which CSGy service request line this specific interrupt is forward: 00_B CSGySR0 01_B CSGySR1 10_B CSGySR2 11_B CSGySR3

Field	Bits	Type	Description
VLS2S	[3:2]	rw	**Value switch from CSGyDSV2 to CSGyDSV1 interrupt line selection** This field programs to which CSGy service request line this specific interrupt is forward: 00_B CSGySR0 01_B CSGySR1 10_B CSGySR2 11_B CSGySR3
TRLS	[5:4]	rw	**Conversion trigger interrupt line selection** This field programs to which CSGy service request line this specific interrupt is forward: 00_B CSGySR0 01_B CSGySR1 10_B CSGySR2 11_B CSGySR3
SSLS	[7:6]	rw	**Start/Stop trigger interrupt line selection** This field programs to which CSGy service request line the combined start and stop interrupt pulses are forward: 00_B CSGySR0 01_B CSGySR1 10_B CSGySR2 11_B CSGySR3
STLS	[9:8]	rw	**Shadow transfer done interrupt line selection** This field programs to which CSGy service request line this specific interrupt is forwarded: 00_B CSGySR0 01_B CSGySR1 10_B CSGySR2 11_B CSGySR3
CRFLS	[11:10]	rw	**Comparator rise/fall interrupt line selection** This field programs to which CSGy service request line the combined comparator rise and comparator fall interrupt pulses are forward: 00_B CSGySR0 01_B CSGySR1 10_B CSGySR2 11_B CSGySR3

Field	Bits	Type	Description
CSLS	[13:12]	rw	**Comparator clamped state interrupt line selection** This field programs to which CSGy service request line the clamped state interrupt is forward: 00_B CSGySR0 01_B CSGySR1 10_B CSGySR2 11_B CSGySR3
0	[31:14]	r	**Reserved** Read always returns 0.

CSGySWS

This register permits the SW to set the status of the several service request flags, as well generate the accordingly service request pulse.

CSGySWS (y = 0 - 2)
Service request SW set **($0A34_H + 0100_H * y$)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0							SCSS	SCFS	SCRS	SSTD	SSTPS	SSTRS	STRGS	SVLS2	SVLS1
r							w	w	w	w	w	w	w	w	w

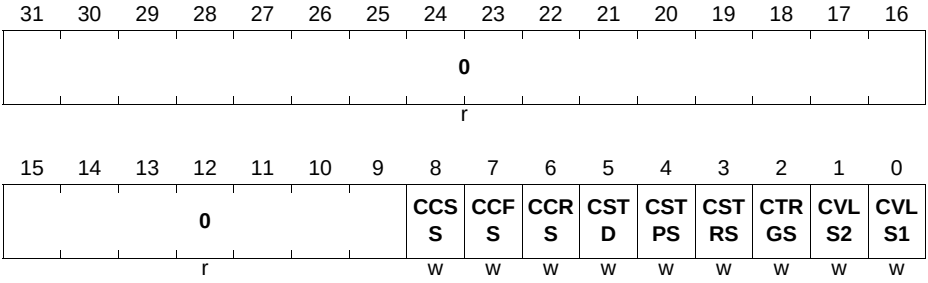
Field	Bits	Type	Description
SVLS1	0	w	**Value switch from CSGyDSV1 to CSGyDSV2 status set** Writing a 1_B to this field will set **CSGyISTAT**.VLS1S to 1_B and will trigger a service request pulse if enable. Read always returns 0_B.

Field	Bits	Type	Description
SVLS2	1	w	**Value switch from CSGyDSV2 to CSGyDSV1 status set** Writing a 1_B to this field will set **CSGyISTAT**.VLS2S to 1_B and will trigger a service request pulse if enable. Read always returns 0_B.
STRGS	2	w	**Conversion trigger status set** Writing a 1_B to this field will set **CSGyISTAT**.TRGSS to 1_B and will trigger a service request pulse if enable. Read always returns 0_B.
SSTRS	3	w	**Start trigger status set** Writing a 1_B to this field will set **CSGyISTAT**.STRSS to 1_B and will trigger a service request pulse if enable. Read always returns 0_B.
SSTPS	4	w	**Stop trigger status set** Writing a 1_B to this field will set **CSGyISTAT**.STPSS to 1_B and will trigger a service request pulse if enable. Read always returns 0_B.
SSTD	5	w	**Shadow transfer status set** Writing a 1_B to this field will set **CSGyISTAT**.STDS to 1_B and will trigger a service request pulse if enable. Read always returns 0_B.
SCRS	6	w	**Comparator rise status set** Writing a 1_B to this field will set **CSGyISTAT**.CRSS to 1_B and will trigger a service request pulse if enable. Read always returns 0_B.
SCFS	7	w	**Comparator fall status set** Writing a 1_B to this field will set **CSGyISTAT**.CFSS to 1_B and will trigger a service request pulse if enable. Read always returns 0_B.
SCSS	8	w	**Comparator clamped state status set** Writing a 1_B to this field will set **CSGyISTAT**.CSES to 1_B and will trigger a service request pulse if enable. Read always returns 0_B.
0	[31:9]	r	**Reserved** Read always returns 0.

CSGySWC

This register permits the SW to clear the status of the several service request flags.

CSGySWC (y = 0 - 2)
Service request SW clear **($0A38_H + 0100_H * y$)** **Reset Value: 00000000_H**



Field	Bits	Type	Description
CVLS1	0	w	**Value switch from CSGyDSV1 to CSGyDSV2 status clear** Writing a 1_B to this field will clear **CSGyISTAT**.VLS1S field. Read always returns 0_B.
CVLS2	1	w	**Value switch from CSGyDSV2 to CSGyDSV1 status clear** Writing a 1_B to this field will clear **CSGyISTAT**.VLS2S field. Read always returns 0_B.
CTRGS	2	w	**Conversion trigger status clear** Writing a 1_B to this field will clear **CSGyISTAT**.TRGSS field. Read always returns 0_B.
CSTRS	3	w	**Start trigger status clear** Writing a 1_B to this field will clear **CSGyISTAT**.STRSS field. Read always returns 0_B.
CSTPS	4	w	**Stop trigger status clear** Writing a 1_B to this field will clear **CSGyISTAT**.STPSS field. Read always returns 0_B.

Field	Bits	Type	Description
CSTD	5	w	**Shadow transfer status clear** Writing a 1_B to this field will clear **CSGyISTAT**.STDS field. Read always returns 0_B.
CCRS	6	w	**Comparator rise status clear** Writing a 1_B to this field will clear **CSGyISTAT**.CRSS field. Read always returns 0_B.
CCFS	7	w	**Comparator fall status clear** Writing a 1_B to this field will clear **CSGyISTAT**.CFSS field. Read always returns 0_B.
CCSS	8	w	**Comparator clamped status clear** Writing a 1_B to this field will clear **CSGyISTAT**.CSES field. Read always returns 0_B.
0	[31:9]	r	**Reserved** Read always returns 0.

CSGyISTAT

This register contains the status of all the service request flags. These flags are set by HW even if the specific service request in not enabled to be propagated to an output. the clear of the flags can only be done via SW, by addressing the **CSGySWC** register.

CSGyISTAT (y = 0 - 2)
Service request status **($0A3C_H + 0100_H * y$)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0							**CSES**	**CFSS**	**CRSS**	**STDS**	**STPSS**	**STRSS**	**TRGSS**	**VLS2S**	**VLS1S**
r							rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
VLS1S	0	rh	**Value switch from CSGyDSV1 to CSGyDSV2 interrupt status** 0_B Value switch not detected 1_B Value switch detected
VLS2S	1	rh	**Value switch from CSGyDSV2 to CSGyDSV1 interrupt status** 0_B Value switch not detected 1_B Value switch detected
TRGSS	2	rh	**Conversion trigger status** 0_B Conversion trigger was not generated 1_B Conversion trigger was generated
STRSS	3	rh	**Start trigger interrupt status** 0_B Start trigger not detected 1_B Start trigger detected
STPSS	4	rh	**Stop trigger interrupt status** 0_B Stop trigger not detected 1_B Stop trigger detected
STDS	5	rh	**Shadow transfer interrupt status** 0_B Shadow transfer was not performed 1_B Shadow transfer was performed
CRSS	6	rh	**Comparator rise interrupt status** 0_B Comparator output LOW to HIGH transition not detected 1_B Comparator output LOW to HIGH transition detected
CFSS	7	rh	**Comparator fall interrupt status** 0_B Comparator output HIGH to LOW transition not detected 1_B Comparator output HIGH to LOW transition detected
CSES	8	rh	**Comparator clamped interrupt status** 0_B Comparator output has been set to the clamped state 1_B Comparator output has not been set to the clamped state
0	[31:9]	r	**Reserved** Read always returns 0.

20.7.4 Global HRC Registers Description

HRCCFG

The register contains the global configuration for the HRCy units.

HRCCFG
Global HRC configuration **(0960_H)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0								LRC 3E	LRC 2E	LRC 1E	LRC 0E	0	CLKC		
r								rw	rw	rw	rw	r	rw		

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0								HRC 3E	HRC 2E	HRC 1E	HRC 0E	0			HRC PM
r								rw	rw	rw	rw		r		rw

Field	Bits	Type	Description
HRCPM	0	rw	**High resolution channels power mode** 0_B High resolution generation logic is OFF. It is not possible to generate high resolution signals throughout any of the high resolution channels, HRCy. 1_B High resolution generation logic is ON. In this mode it is possible to generate a high resolution signal placement with the HRCy subunits.
HRC0E	4	rw	**HRC0 high resolution enable** 0_B HRC0 High Resolution Path is disabled. In this mode, is not possible to use the High Resolution Path inside of HRC0 to generate an output PWM signal. 1_B HRC0 High Resolution Path is enabled. In this mode it is possible to generate a high resolution PWM signal if HRCPM = 1_B.

Field	Bits	Type	Description
HRC1E	5	rw	**HRC1 high resolution channel enable** 0_B HRC1 High Resolution Path is disabled. In this mode, is not possible to use the High Resolution Path inside of HRC1 to generate an output PWM signal. 1_B HRC1 High Resolution Path is enabled. In this mode it is possible to generate a high resolution PWM signal if HRCPM = 1_B.
HRC2E	6	rw	**HRC2 high resolution channel enable** 0_B HRC2 High Resolution Path is disabled. In this mode, is not possible to use the High Resolution Path inside of HRC2 to generate an output PWM signal. 1_B HRC2 High Resolution Path is enabled. In this mode it is possible to generate a high resolution PWM signal if HRCPM = 1_B.
HRC3E	7	rw	**HRC3 high resolution channel enable** 0_B HRC3 High Resolution Path is disabled. In this mode, is not possible to use the High Resolution Path inside of HRC3 to generate an output PWM signal. 1_B HRC3 High Resolution Path is enabled. In this mode it is possible to generate a high resolution PWM signal if HRCPM = 1_B.

Field	Bits	Type	Description
CLKC	[18:16]	rw	**Clock information control** This field needs to be configure accordingly with the module clock speed configured at system level. If the clock frequency is different from one of these three values, the code for the closest supported frequency should be programmed. 000_B No clock frequency is selected 001_B Module clock frequency is 180 MHz 010_B Module clock frequency is 120 MHz 011_B Module clock frequency is 80 MHz 100_B Reserved 101_B Reserved 110_B Reserved 111_B Reserved *Note: Address the specific HRPWM section on the device Data Sheet for a description of the high resolution accuracy versus the clock frequency.*
LRC0E	20	rw	**HRC0 low resolution channel enable** 0_B HRC0 Low Resolution Path is disabled. In this mode, is not possible to use the Low Resolution Path inside of HRC0 to generate an output PWM signal. 1_B HRC0 Low Resolution Path is enabled. In this mode it is possible to generate a an output PWM signal via the Low Resolution Path.
LRC1E	21	rw	**HRC1 low resolution channel enable** 0_B HRC1 Low Resolution Path is disabled. In this mode, is not possible to use the Low Resolution Path inside of HRC1 to generate an output PWM signal. 1_B HRC1 Low Resolution Path is enabled. In this mode it is possible to generate a an output PWM signal via the Low Resolution Path.

Field	Bits	Type	Description
LRC2E	22	rw	**HRC2 low resolution channel enable** 0_B HRC2 Low Resolution Path is disabled. In this mode, is not possible to use the Low Resolution Path inside of HRC2 to generate an output PWM signal. 1_B HRC2 Low Resolution Path is enabled. In this mode it is possible to generate a an output PWM signal via the Low Resolution Path.
LRC3E	23	rw	**HRC3 low resolution channel enable** 0_B HRC3 Low Resolution Path is disabled. In this mode, is not possible to use the Low Resolution Path inside of HRC3 to generate an output PWM signal. 1_B HRC3 Low Resolution Path is enabled. In this mode it is possible to generate a an output PWM signal via the Low Resolution Path.
0	[3:1], [15:8], 19, [31:24]	r	**Reserved** Read always returns 0

HRCSTRG

The register contains the bitfields that are used to enable a shadow transfer for the high resolution compare values, **HRCyCR1** and **HRCyCR2**, and dead time values, **HRCyDCR** and **HRCyDCF**.

HRCSTRG
Global HRC shadow trigger set **(0964_H)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		H3DES	H3ES	0		H2DES	H2ES	0		H1DES	H1ES	0		H0DES	H0ES
r		w	w	r		w	w	r		w	w	r		w	w

Field	Bits	Type	Description
H0ES	0	w	**HRC0 high resolution values shadow transfer Enable Set** Writing a 1_B to this bitfield, enables the shadow transfer for the high resolution compare values, **HRCyCR1** and **HRCyCR2**. Field **HRCSTSG**.H0STE is set to 1_B. The shadow transfer will be performed whenever the shadow transfer trigger is detected. A read always returns 0_B.
H0DES	1	w	**HRC0 dead time value shadow transfer enable set** Writing a 1_B to this bitfield, enables the shadow transfer for the dead time values, **HRCyDCR** and **HRCyDCF**. Field **HRCSTSG**.H0DSE is set to 1_B. The shadow transfer will be performed whenever the shadow transfer trigger is detected. A read always returns 0_B.
H1ES	4	w	**HRC1 high resolution values shadow transfer Enable Set** Writing a 1_B to this bitfield, enables the shadow transfer for the high resolution compare values, **HRCyCR1** and **HRCyCR2**. Field **HRCSTSG**.H1STE is set to 1_B. The shadow transfer will be performed whenever the shadow transfer trigger is detected. A read always returns 0_B.
H1DES	5	w	**HRC0 dead time value shadow transfer enable set** Writing a 1_B to this bitfield, enables the shadow transfer for the dead time values, **HRCyDCR** and **HRCyDCF**. Field **HRCSTSG**.H1DSE is set to 1_B. The shadow transfer will be performed whenever the shadow transfer trigger is detected. A read always returns 0_B.

Field	Bits	Type	Description
H2ES	8	w	**HRC2 high resolution values shadow transfer Enable Set** Writing a 1_B to this bitfield, enables the shadow transfer for the high resolution compare values, **HRCyCR1** and **HRCyCR2**. Field **HRCSTSG**.H2STE is set to 1_B. The shadow transfer will be performed whenever the shadow transfer trigger is detected. A read always returns 0_B.
H2DES	9	w	**HRC0 dead time value shadow transfer enable set** Writing a 1_B to this bitfield, enables the shadow transfer for the dead time values, **HRCyDCR** and **HRCyDCF**. Field **HRCSTSG**.H2DSE is set to 1_B. The shadow transfer will be performed whenever the shadow transfer trigger is detected. A read always returns 0_B.
H3ES	12	w	**HRC3 high resolution values shadow transfer Enable Set** Writing a 1_B to this bitfield, enables the shadow transfer for the high resolution compare values, **HRCyCR1** and **HRCyCR2**. Field **HRCSTSG**.H3STE is set to 1_B. The shadow transfer will be performed whenever the shadow transfer trigger is detected. A read always returns 0_B.
H3DES	13	w	**HRC0 dead time value shadow transfer enable set** Writing a 1_B to this bitfield, enables the shadow transfer for the dead time values, **HRCyDCR** and **HRCyDCF**. Field **HRCSTSG**.H3DSE is set to 1_B. The shadow transfer will be performed whenever the shadow transfer trigger is detected. A read always returns 0_B.
0	[3:2], [7:6], [11:10], [31:14]	r	**Reserved** Read always returns 0.

HRCCTRG

The register contains the bitfields that are used to clear the shadow transfer enable bit, for the high resolution compare values, **HRCyCR1** and **HRCyCR2**, and dead time values, **HRCyDCR** and **HRCyDCF**.

HRCCTRG
Global HRC shadow trigger clear **(0968_H)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		H3DEC	H3EC	0		H2DEC	H2CEC	0		H1DEC	H1EC	0		H0DEC	H0EC
r		w	w	r		w	w	r		w	w	r		w	w

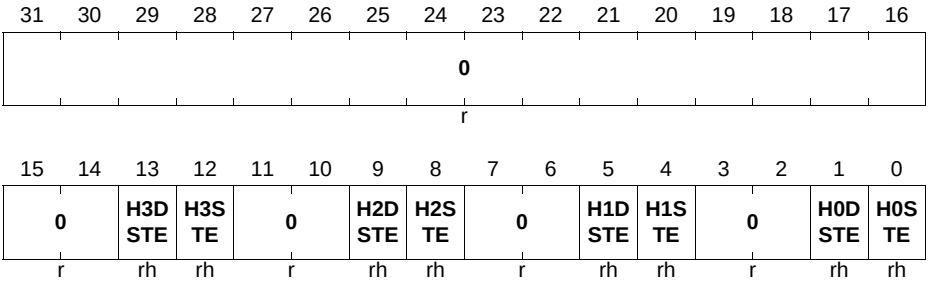
Field	Bits	Type	Description
H0EC	0	w	**HRC0 high resolution values shadow transfer Enable Clear** Writing a 1_B to this field, will clear the **HRCSTSG**.H0STE bit. This will disable an update of the high resolution compare values, **HRCyCR1** and **HRCyCR2**. A read always returns 0_B.
H0DEC	1	w	**HRC0 dead time value shadow transfer Enable Clear** Writing a 1_B to this field, will clear the **HRCSTSG**.H0DSTE bit. This will disable an update of the dead time values, **HRCyDCR** and **HRCyDCF**. A read always returns 0_B.
H1EC	4	w	**HRC1 high resolution values shadow transfer Enable Clear** Writing a 1_B to this field, will clear the **HRCSTSG**.H1STE bit. This will disable an update of the high resolution compare values, **HRCyCR1** and **HRCyCR2**. A read always returns 0_B.

Field	Bits	Type	Description
H1DEC	5	w	**HRC1 dead time value shadow transfer Enable Clear** Writing a 1_B to this field, will clear the **HRCSTSG**.H1DSTE bit. This will disable an update of the dead time values, **HRCyDCR** and **HRCyDCF**. A read always returns 0_B.
H2CEC	8	w	**HRC2 high resolution values shadow transfer Enable Clear** Writing a 1_B to this field, will clear the **HRCSTSG**.H2STE bit. This will disable an update of the high resolution compare values, **HRCyCR1** and **HRCyCR2**. A read always returns 0_B.
H2DEC	9	w	**HRC2 dead time value shadow transfer Enable Clear** Writing a 1_B to this field, will clear the **HRCSTSG**.H2DSTE bit. This will disable an update of the dead time values, **HRCyDCR** and **HRCyDCF**. A read always returns 0_B.
H3EC	12	w	**HRC3 high resolution values shadow transfer Enable Clear** Writing a 1_B to this field, will clear the **HRCSTSG**.H3STE bit. This will disable an update of the high resolution compare values, **HRCyCR1** and **HRCyCR2**. A read always returns 0_B.
H3DEC	13	w	**HRC3 dead time value shadow transfer Enable Clear** Writing a 1_B to this field, will clear the **HRCSTSG**.H3DSTE bit. This will disable an update of the dead time values, **HRCyDCR** and **HRCyDCF**. A read always returns 0_B.
0	[3:2], [7:6], [11:10], [31:14]	r	**Reserved** Read always returns 0.

HRCSTSG

The register contains the status of the shadow transfer request for the high resolution compare and dead time values.

HRCSTSG
Global HRC shadow transfer status (096C$_H$) **Reset Value: 00000000$_H$**



Field	Bits	Type	Description
H0STE	0	rh	**HRC0 high resolution values shadow transfer status** This field is automatically cleared by HW when the specific shadow transfer has been performed. 0_B No shadow transfer pending for **HRCyCR1** and **HRCyCR2** values 1_B Shadow transfer pending for **HRCyCR1** and **HRCyCR2** values
H0DSTE	1	rh	**HRC0 dead time value shadow transfer status** This field is automatically cleared by HW when the specific shadow transfer has been performed. 0_B No shadow transfer pending for **HRCyDCR** and **HRCyDCF** values 1_B Shadow transfer pending for **HRCyDCR** and **HRCyDCF** values

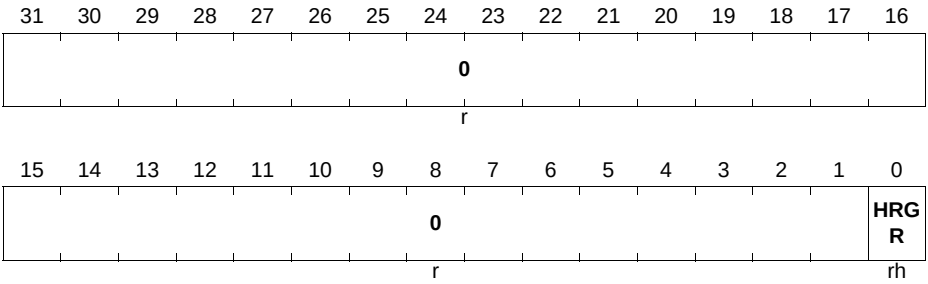
Field	Bits	Type	Description
H1STE	4	rh	**HRC1 high resolution values shadow transfer status** This field is automatically cleared by HW when the specific shadow transfer has been performed. 0_B No shadow transfer pending for **HRCyCR1** and **HRCyCR2** values 1_B Shadow transfer pending for **HRCyCR1** and **HRCyCR2** values
H1DSTE	5	rh	**HRC1 dead time value shadow transfer status** This field is automatically cleared by HW when the specific shadow transfer has been performed. 0_B No shadow transfer pending for **HRCyDCR** and **HRCyDCF** values 1_B Shadow transfer pending for **HRCyDCR** and **HRCyDCF** values
H2STE	8	rh	**HRC2 high resolution values shadow transfer status** This field is automatically cleared by HW when the specific shadow transfer has been performed. 0_B No shadow transfer pending for **HRCyCR1** and **HRCyCR2** values 1_B Shadow transfer pending for **HRCyCR1** and **HRCyCR2** values
H2DSTE	9	rh	**HRC2 dead time value shadow transfer status** This field is automatically cleared by HW when the specific shadow transfer has been performed. 0_B No shadow transfer pending for **HRCyDCR** and **HRCyDCF** values 1_B Shadow transfer pending for **HRCyDCR** and **HRCyDCF** values
H3STE	12	rh	**HRC3 high resolution values shadow transfer status** This field is automatically cleared by HW when the specific shadow transfer has been performed. 0_B No shadow transfer pending for **HRCyCR1** and **HRCyCR2** values 1_B Shadow transfer pending for **HRCyCR1** and **HRCyCR2** values

Field	Bits	Type	Description
H3DSTE	13	rh	**HRC3 dead time value shadow transfer status** This field is automatically cleared by HW when the specific shadow transfer has been performed. 0_B No shadow transfer pending for **HRCyDCR** and **HRCyDCF** values 1_B Shadow transfer pending for **HRCyDCR** and **HRCyDCF** values
0	[3:2], [7:6], [11:10], [31:14]	r	**Reserved** Read always returns 0.

HRGHRS

The register contains the status of the high resolution generation logic.

HRGHRS
High Resolution Generation Status (0970_H) **Reset Value: 00000000_H**



Field	Bits	Type	Description
HRGR	0	rh	**High Resolution Generation Ready** This field contains the status of the high resolution logic. When this bitfield is not set, it is not possible to generate a high resolution signal in any of the HRCy channels. 0_B High resolution logic is not working 1_B High resolution logic is working

Field	Bits	Type	Description
0	[31:1]	r	**Reserved** Read always returns 0.

20.7.5 HRCy Registers Description

HRCyGC

This register contains the general configuration for the HRCy operation.

HRCyGC (y = 0 - 3)
HRCy mode configuration **($1300_H + 0100_H * y$)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															**DTUS**
r															rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**OCS1**	**OCS0**	**DSTC**	**STC**	**TR1E**	**TR0E**	**DTE**	**0**				**HRM1**		**HRM0**	
r	rw	rw	rw	rw	rw	rw	rw	r				rw		rw	

Field	Bits	Type	Description
HRM0	[1:0]	rw	**HRCy high resolution mode configuration for source selector 0** This field controls the high resolution mode for the PWM signal, that is generated via the signals decoded from the Source Selector 0. 00_B Rising edge high resolution signal positioning enabled 01_B Falling edge high resolution signal positioning enabled 10_B Both edges high resolution signal positioning is enabled 11_B No high resolution positioning *Note: the configuration of the HRM0 is dependent on which signal is being used to control the output latch.*

Field	Bits	Type	Description
HRM1	[3:2]	rw	**HRCy high resolution mode configuration for source selector 1** This field controls the high resolution mode for the PWM signal, that is generated via the signals decoded from the Source Selector 1. 00_B Rising edge high resolution signal positioning enabled 01_B Falling edge high resolution signal positioning enabled 10_B Both edges high resolution signal positioning is enabled 11_B No high resolution positioning *Note: the configuration of the HRM1 is dependent on which signal is being used to control the output latch.*
DTE	8	rw	**HRCy dead time enable** 0_B Dead time insertion is disabled 1_B Dead time insertion is enabled
TR0E	9	rw	**HRCy trap enable** 0_B Trap function for HRPWMx.HROUTy0 is disabled 1_B Trap function for HRPWMx.HROUTy0 is enabled
TR1E	10	rw	**HRCy complementary trap enable** 0_B Trap function for HRPWMx.HROUTy1 is disabled 1_B Trap function for HRPWMx.HROUTy1 is enabled

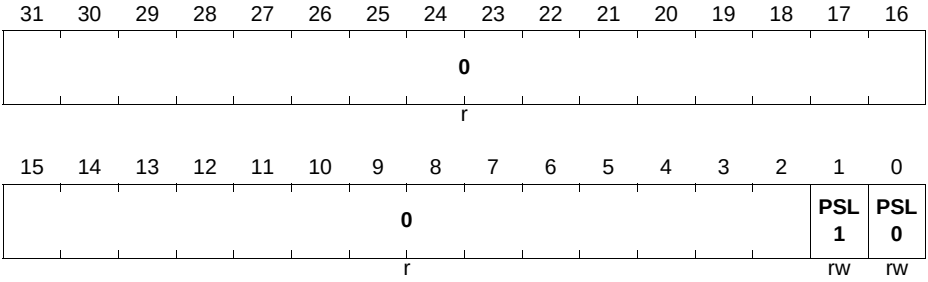
Field	Bits	Type	Description
STC	11	rw	**HRCy shadow transfer configuration** This field enables the link between the Capture/Compare Unit timer shadow transfer enable and the shadow transfer enable of the HRCy unit, for the **HRCyCR1** and **HRCyCR2** values. This means that the SW only needs to request a shadow transfer on the specific Capture/Compare Unit timer and doesn't need to address the HRCy unit individually. 0_B HRCy shadow transfer enable for **HRCyCR1** and **HRCyCR2** values is not linked with the specific Capture/Compare Unit timer. 1_B HRCy shadow transfer enable for **HRCyCR1** and **HRCyCR2** values is linked with the specific Capture/Compare Unit timer.
DSTC	12	rw	**HRCy dead time shadow transfer configuration** This field enables the link between the Capture/Compare Unit timer shadow transfer enable and the shadow transfer enable of the HRCy unit, for the **HRCyDCR** and **HRCyDCF** values. This means that the SW only needs to request a shadow transfer on the specific Capture/Compare Unit timer and doesn't need to address the HRCy unit individually. 0_B HRCy shadow transfer enable for **HRCyDCR** and **HRCyDCF** values is not linked with the specific Capture/Compare Unit timer. 1_B HRCy shadow transfer enable for **HRCyDCR** and **HRCyDCF** values is linked with the specific Capture/Compare Unit timer.
OCS0	13	rw	**HRPWMx.OUTy0 channel selector** 0_B HRPWMx.OUTy0 is connected to the latch Q channel 1_B HRPWMx.OUTy0 is connected to the latch Qn channel
OCS1	14	rw	**HRPWMx.OUTy1 channel selector** 0_B HRPWMx.OUTy1 is connected to the latch Qn channel 1_B HRPWMx.OUTy1 is connected to the latch Q channel

Field	Bits	Type	Description
DTUS	16	rw	**Dead Time update trigger selector** This field selects which trigger is used to update the dead time values (loading the value of **HRCySDCR** and **HRCySDCF** into **HRCyDCR** and **HRCyDCF**, respectively). 0_B The update of the values is done with the trigger generated by the timers. This is the same trigger that is used to update the **HRCyCR1** and **HRCyCR2**. 1_B The update of the dead time values is done when the dead time counter is not running, independently of the **HRCyCR1** and **HRCyCR2** registers.
0	[7:4], 15, [31:17]	r	**Reserved** Read always returns 0.

HRCyPL

This register configures the passive level for the two outputs of the HRCy unit.

HRCyPL (y = 0 - 3)
HRCy output passive level **($1304_H + 0100_H * y$)** **Reset Value: 00000000_H**



Field	Bits	Type	Description
PSL0	0	rw	**HRPWMx.OUTy0 passive level** 0_B HRPWMx.OUTy0 output passive level is set to LOW 1_B HRPWMx.OUTy0 output passive level is set to HIGH

Field	Bits	Type	Description
PSL1	1	rw	**HRPWMx.OUTy1 passive level** 0_B HRPWMx.OUTy1 output passive level is set to LOW 1_B HRPWMx.OUTy1 output passive level is set to HIGH
0	[31:2]	r	**Reserved** Read always returns 0.

HRCyGSEL

This register holds the configuration of which inputs are being used to generate the set and clear for the latch and therefore controlling the generation of the output PWM signal.

HRCyGSEL (y = 0 - 3)
HRCy global control selection ($1308_H + 0100_H * y$) **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0		**C1ES**		**S1ES**		**C1M**		**S1M**		**C1CS**			**C1SS**		
r		rw		rw		rw		rw		rw			rw		

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		**C0ES**		**S0ES**		**C0M**		**S0M**		**C0CS**			**C0SS**		
r		rw		rw		rw		rw		rw			rw		

Field	Bits	Type	Description
C0SS	[2:0]	rw	**Source selector 0 comparator set configuration** 000_B CMP output of CSG0 unit can be used as set source for the output latch 001_B CMP output of CSG1 unit can be used as set source for the output latch 010_B CMP output of CSG2 unit can be used as set source for the output latch 011_B Reserved 100_B Reserved 101_B Reserved 110_B Reserved 111_B Reserved

Field	Bits	Type	Description
C0CS	[5:3]	rw	**Source selector 0 comparator clear configuration** 000_B CMP output of CSG0 unit can be used as clear source for the output latch 001_B CMP output of CSG1 unit can be used as clear source for the output latch 010_B CMP output of CSG2 unit can be used as clear source for the output latch 011_B Reserved 100_B Reserved 101_B Reserved 110_B Reserved 111_B Reserved
S0M	[7:6]	rw	**Source selector 0 set configuration** 00_B Set from source selector 0 is controlled via the Capture/Compare Unit timer, CCSTy signal 01_B Set from source selector 0 is controlled via the CMP output from the CSGy unit. Which unit is being used is configured via the C0SS field. 10_B Reserved 11_B Reserved
C0M	[9:8]	rw	**Source selector 0 clear configuration** 00_B Clear from source selector 0 is controlled via the Capture/Compare Unit timer, CCSTy signal 01_B Clear from source selector 0 is controlled via the CMP output from the CSGy unit. Which unit is being used is configured via the C0CS field. 10_B Reserved 11_B Reserved
S0ES	[11:10]	rw	**Source selector 0 set edge configuration** 00_B Generation of the set signal is disabled 01_B Set signal is generated on a LOW to HIGH transition of the selected input 10_B Set signal is generated on a HIGH to LOW transition of the selected input 11_B Set signal is generated on both transitions of the selected input

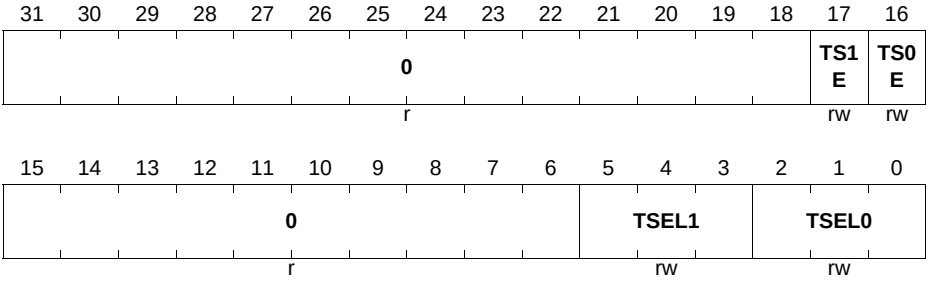
Field	Bits	Type	Description
C0ES	[13:12]	rw	**Source selector 0 clear edge configuration** 00_B Generation of the clear signal is disabled 01_B Clear signal is generated on a LOW to HIGH transition of the selected input 10_B Clear signal is generated on a HIGH to LOW transition of the selected input 11_B Clear signal is generated on both transitions of the selected input
C1SS	[18:16]	rw	**Source selector 1 comparator set configuration** 000_B CMP output of CSG0 unit can be used as set source for the output latch 001_B CMP output of CSG2 unit can be used as set source for the output latch 010_B CMP output of CSG2 unit can be used as set source for the output latch 11_B Reserved 100_B Reserved 101_B Reserved 110_B Reserved 111_B Reserved
C1CS	[21:19]	rw	**Source selector 1 comparator clear configuration** 000_B CMP output of CSG0 unit can be used as clear source for the output latch 001_B CMP output of CSG2 unit can be used as clear source for the output latch 010_B CMP output of CSG2 unit can be used as clear source for the output latch 011_B Reserved 100_B Reserved 101_B Reserved 110_B Reserved 111_B Reserved
S1M	[23:22]	rw	**Source selector 1 set configuration** 00_B Set from source selector 1 is controlled via the Capture/Compare Unit timer, CCSTy signal 01_B Set from source selector 1 is controlled via the CMP output from the CSGy unit. Which unit is being used is configured via the C1SS field. 10_B Reserved 11_B Reserved

Field	Bits	Type	Description
C1M	[25:24]	rw	**Source selector 1 clear configuration** 00_B Clear from source selector 1 is controlled via the Capture/Compare Unit timer, CCSTy signal 01_B Clear from source selector 1 is controlled via the CMP output from the CSGy unit. Which unit is being used is configured via the C1CS field. 10_B Reserved 11_B Reserved
S1ES	[27:26]	rw	**Source selector 1 set edge configuration** 00_B Generation of the set signal is disabled 01_B Set signal is generated on a LOW to HIGH transition of the selected input 10_B Set signal is generated on a HIGH to LOW transition of the selected input 11_B Set signal is generated on both transitions of the selected input
C1ES	[29:28]	rw	**Source selector 1 clear edge configuration** 00_B Generation of the clear signal is disabled 01_B Clear signal is generated on a LOW to HIGH transition of the selected input 10_B Clear signal is generated on a HIGH to LOW transition of the selected input 11_B Clear signal is generated on both transitions of the selected input
0	[15:14], [31:30]	r	**Reserved** Read always returns 0.

HRCyTSEL

This register holds the configuration for which timer from the Capture/Compare Unit is used for the Source Selector 0 and Source Selector 1.

HRCyTSEL (y = 0 - 3)
HRCy timer selection **($130C_H + 0100_H * y$)** **Reset Value: 00000000_H**



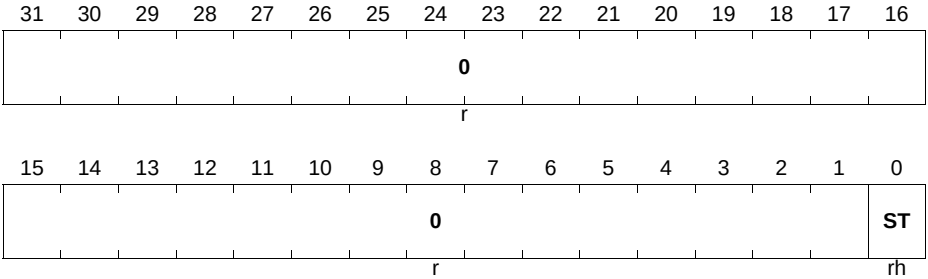
Field	Bits	Type	Description
TSEL0	[2:0]	rw	**Source Selector 0 Timer connection** 000_B Source Selector 0 is connected to Capture/Compare Unit Timer 0 (CCST0 can be used) 001_B Source Selector 0 is connected to Capture/Compare Unit Timer 1 (CCST1 can be used) 010_B Source Selector 0 is connected to Capture/Compare Unit Timer 2 (CCST2 can be used) 011_B Source Selector 0 is connected to Capture/Compare Unit Timer 3 (CCST3 can be used) 100_B Reserved 101_B Reserved 110_B Reserved 111_B Reserved

Field	Bits	Type	Description
TSEL1	[5:3]	rw	**Source Selector 1 Timer connection** 000_B Source Selector 1 is connected to Capture/Compare Unit Timer 0 (CCST0 can be used) 001_B Source Selector 1 is connected to Capture/Compare Unit Timer 1 (CCST1 can be used) 010_B Source Selector 1 is connected to Capture/Compare Unit Timer 2 (CCST2 can be used) 011_B Source Selector 1 is connected to Capture/Compare Unit Timer 3 (CCST3 can be used) 100_B Reserved 101_B Reserved 110_B Reserved 111_B Reserved
TS0E	16	rw	**Source selector 0 TRAP enable** 0_B TRAP signal generated from the Timer connected to Source Selector 0 is disabled. 1_B TRAP signal generated from the Timer connected to Source Selector 0 is enabled.
TS1E	17	rw	**Source selector 1 TRAP enable** 0_B TRAP signal generated from the Timer connected to Source Selector 1 is disabled. 1_B TRAP signal generated from the Timer connected to Source Selector 1 is enabled.
0	[15:6], [31:18]	r	**Reserved** Read always returns 0.

HRCySC

This register selects to which Source Selector, Source Selector 0 or Source Selector 1, the shadow transfer trigger and shadow transfer enable (from the Capture/Compare Unit Timer) is linked. Update of this register needs to be done via the **HRCySSC**.

HRCySC (y = 0 - 3)
HRCy current source for shadow(1310_H + 0100_H * y) **Reset Value: 00000000_H**

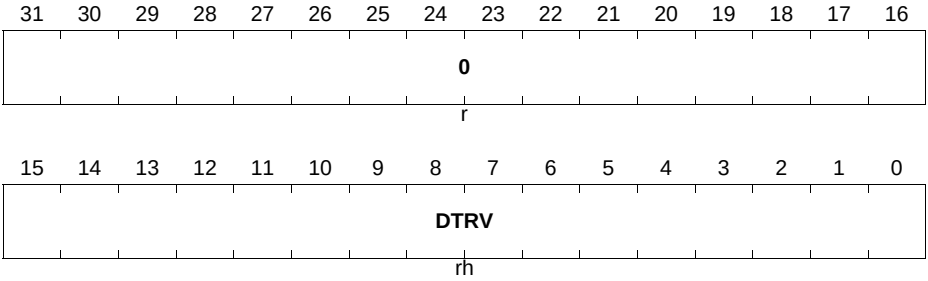


Field	Bits	Type	Description
ST	0	rh	**Source selector for the shadow transfer** 0_B Shadow transfer signals (shadow transfer trigger and shadow transfer enable) are linked with the timer CC8y connected to the Source Selector 0. 1_B Shadow transfer signals (shadow transfer trigger and shadow transfer enable) are linked with the timer CC8y connected to the Source Selector 1.
0	[31:1]	r	**Reserved** Read always returns 0.

HRCyDCR

This register holds the dead time value that is going to be inserted whenever a rising transition on the output latch is sensed.

HRCyDCR (y = 0 - 3)
HRCy dead time rising value **(1314_H + 0100_H * y)** **Reset Value: 00000001_H**

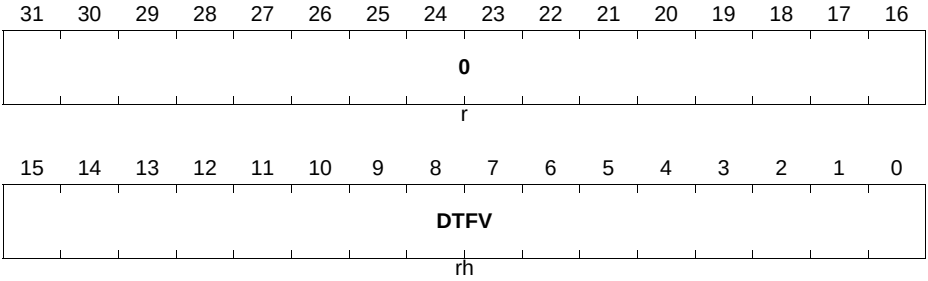


Field	Bits	Type	Description
DTRV	[15:0]	rh	**Dead time rising value** This field contains the dead time value that is going to be inserted in every transition from LOW to HIGH of the output latch. This is a full decimal mapping and the resolution is dictated by the module clock: Value = <DTRV> + 1 *Note: The update of this fields needs to be done via the associated shadow register, **HRCySDCR**.*
0	[31:16]	r	**Reserved** Read always returns 0.

HRCyDCF

This register holds the dead time value that is going to be inserted whenever a falling transition on the output latch is sensed.

HRCyDCF (y = 0 - 3)
HRCy dead time falling value **(1318_H + 0100_H * y)** **Reset Value: 00000001_H**

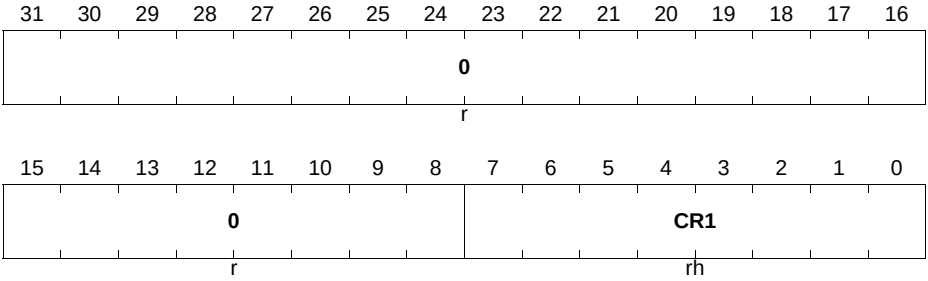


Field	Bits	Type	Description
DTFV	[15:0]	rh	**Dead time falling value** This field contains the dead time value that is going to be inserted in every transition from HIGH to LOW of the output latch. This is a full decimal mapping and the resolution is dictated by the module clock: Value = <DTFV> + 1 *Note: The update of this fields needs to be done via the associated shadow register, **HRCySDCF**.*
0	[31:16]	r	**Reserved** Read always returns 0.

HRCyCR1

This register holds the value for the rising edge high resolution signal placement. the update of this value should be done via the associated shadow register, **HRCySCR1**. The maximum value mentioned in the register description, indicates that if a bigger value is needed, then the software should set this value to 0_D and increase the MSB value on the specific CCU8 timer.

HRCyCR1 (y = 0 - 3)
HRCy rising edge value **($131C_H + 0100_H * y$)** **Reset Value: 00000000_H**

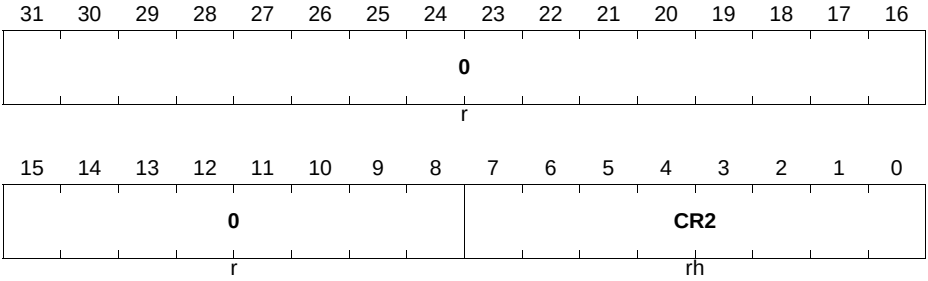


Field	Bits	Type	Description
CR1	[7:0]	rh	**High resolution rising edge value** This field contains the high resolution value that is going to affect the rising edge of the PWM signal. The resolution of the signal placement is 150 ps. This is a full decimal mapping which means that: <placement_value> = CR1 x 150 ps The maximum value for this field should be set accordingly with the used module clock frequency: 82_D for 80 MHz 54_D for 120 MHz 36_D for 180 MHz *Note: The update of this fields needs to be done via the associated shadow register, **HRCySCR1**.* *Note: Having an adjustment bigger than the maximum value, is done by increasing the MSBs.*
0	[31:8]	r	**Reserved** Read always returns 0.

HRCyCR2

This register holds the value for the falling edge high resolution signal placement. the update of this value should be done via the associated shadow register, **HRCySCR2**. The maximum value mentioned in the register description, indicates that if a bigger value is needed, then the software should set this value to 0_D and increase the MSB value on the specific CCU8 timer.

HRCyCR2 (y = 0 - 3)
HRCy falling edge value **($1320_H + 0100_H * y$)** **Reset Value: 00000000_H**

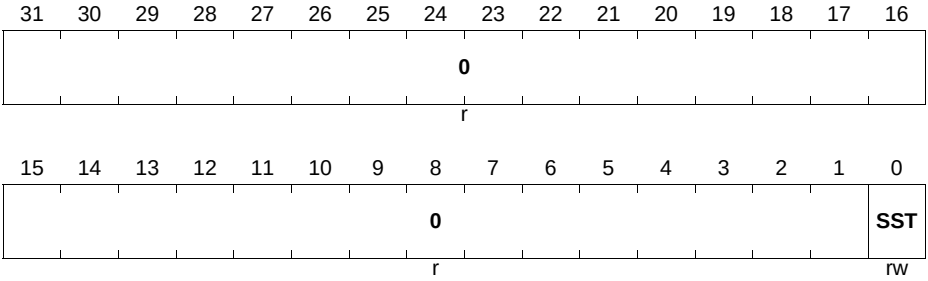


Field	Bits	Type	Description
CR2	[7:0]	rh	**High resolution falling edge value** This field contains the high resolution value that is going to affect the falling edge of the PWM signal. The resolution of the signal placement is 150 ps. This is a full decimal mapping which means that: <placement_value> = CR1 x 150 ps The maximum value for this field should be set accordingly with the used module clock frequency: 82_D for 80 MHz 54_D for 120 MHz 36_D for 180 MHz *Note: The update of this fields needs to be done via the associated shadow register, **HRCySCR2**.* *Note: Having an adjustment bigger than the maximum value, is done by increasing the MSBs.*
0	[31:8]	r	**Reserved** Read always returns 0.

HRCySSC

This register holds the next value that is going to be loaded into the **HRCySC** register.

HRCySSC (y = 0 - 3)
HRCy next source for shadow (1324_H + 0100_H * y) **Reset Value: 00000000_H**

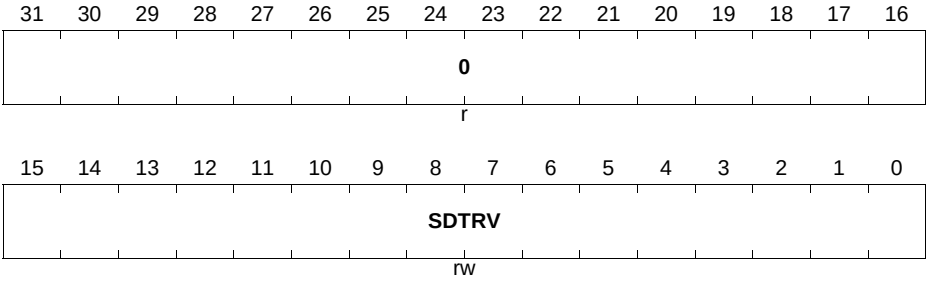


Field	Bits	Type	Description
SST	0	rw	**Source selector for the shadow transfer** 0_B Next shadow transfer signals (shadow transfer trigger and shadow transfer enable) are linked with the timer CC8y connected to the Source Selector 0. 1_B Next shadow transfer signals (shadow transfer trigger and shadow transfer enable) are linked with the timer CC8y connected to the Source Selector 1.
0	[31:1]	r	**Reserved** Read always returns 0.

HRCySDCR

This register contains the value that is going to be passed into the **HRCyDCR** field when a shadow transfer is performed.

HRCySDCR (y = 0 - 3)
HRCy shadow dead time rising (1328_H + 0100_H * y) **Reset Value: 00000001_H**

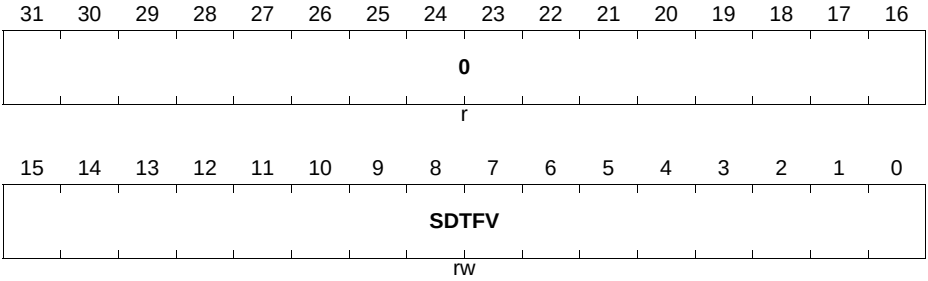


Field	Bits	Type	Description
SDTRV	[15:0]	rw	**Shadow dead time rising value** This field contains the dead time value that is going to be set into the **HRCyDCR**.DTRV field in the next shadow transfer update. Writing a value of 0000_H is not allowed and is mapped internally to 0001_H. This is a full decimal mapping and the dead time insertion value is dictated by the module clock: Value = <SDTRV> + 1
0	[31:16]	r	**Reserved** Read always returns 0.

HRCySDCF

This register contains the value that is going to be passed into the **HRCyDCF** field when a shadow transfer is performed.

HRCySDCF (y = 0 - 3)
HRCy shadow dead time falling($132C_H$ + 0100_H * y) **Reset Value: 00000001_H**

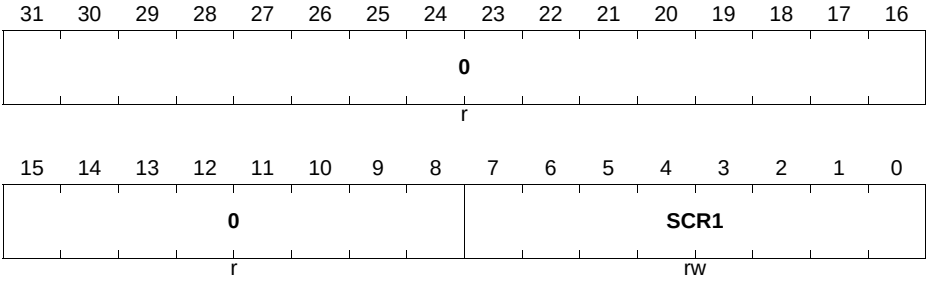


Field	Bits	Type	Description
SDTFV	[15:0]	rw	**Shadow dead time falling value** This field contains the dead time value that is going to be set into the **HRCyDCF**.DTFV field in the next shadow transfer update. Writing a value of 0000_H is not allowed and is mapped internally to 0001_H. This is a full decimal mapping and the dead time insertion value is dictated by the module clock: Value = <SDTFV> + 1
0	[31:16]	r	**Reserved** Read always returns 0.

HRCySCR1

This register holds the value that is going to be set into the **HRCyCR1** field in the next shadow transfer update. The maximum value mentioned in the register description, indicates that if a bigger value is needed, then the software should set this value to 0_D and increase the MSB value on the specific CCU8 timer.

HRCySCR1 (y = 0 - 3)
HRCy shadow rising edge value(1330_H + 0100_H * y) **Reset Value: 00000000_H**

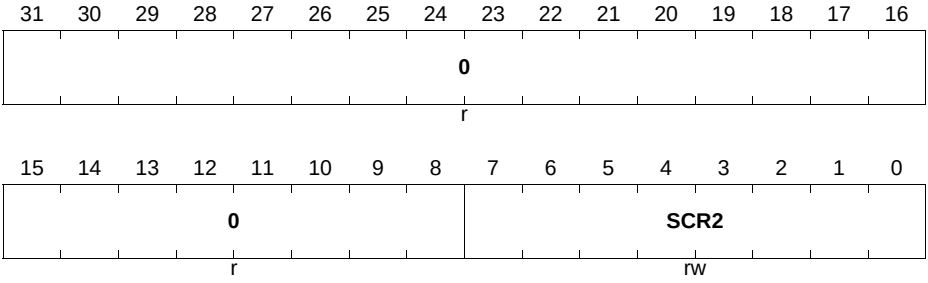


Field	Bits	Type	Description
SCR1	[7:0]	rw	**High resolution falling edge value** This field contains the value that is going to be set into the **HRCyCR1**.CR1 field in the next shadow transfer update. The maximum value for this field should be set accordingly with the used module clock frequency: 82_D for 80 MHz 54_D for 120 MHz 36_D for 180 MHz *Note: Having an adjustment bigger than the maximum value, is done by increasing the MSBs.*
0	[31:8]	r	**Reserved** Read always returns 0.

HRCySCR2

This register holds the value that is going to be set into the **HRCyCR2** field in the next shadow transfer update. The maximum value mentioned in the register description, indicates that if a bigger value is needed, then the software should set this value to 0_D and increase the MSB value on the specific CCU8 timer.

HRCySCR2 (y = 0 - 3)
HRCy shadow falling edge value(1334_H + 0100_H * y) **Reset Value: 00000000_H**



Field	Bits	Type	Description
SCR2	[7:0]	rw	**High resolution rising edge value** This field contains the value that is going to be set into the **HRCyCR2**.CR2 field in the next shadow transfer update. The maximum value for this field should be set accordingly with the used module clock frequency: 82_D for 80 MHz 54_D for 120 MHz 36_D for 180 MHz *Note: Having an adjustment bigger than the maximum value, is done by increasing the MSBs.*
0	[31:8]	r	**Reserved** Read always returns 0.

20.8 Interconnects

The connectivity tables are divided into functional groups: Global connections, CSGy connections and HRCy connections.

The Halt/Suspend input signal should be connected to the processor Halt signal for all the modules.

The GPIO mapping is available at the Ports unit.

20.8.1 HRPWM0 pins

Table 20-25 HRPWM0 Pin Connections

Global Inputs/Outputs	I/O	Connected To	Description
HRPWM0.MCLK	I	SCU.HRPWMCLK	Module clock. Equal to the ccu_clk but contains an independent enable in the SCU.
HRPWM0.SR0	O	NVIC; DMA; CCU80.IN0L; CCU80.IN1L; CCU80.IN2L; CCU80.IN3L	Service request line
HRPWM0.SR1	O	NVIC; DMA;	Service request line
HRPWM0.SR2	O	NVIC; HRPWM0.BL0M; HRPWM0.BL1M; HRPWM0.BL2M;	Service request line
HRPWM0.SR3	O	NVIC; HRPWM0.BL0N; HRPWM0.BL1N; HRPWM0.BL2N; CCU41.IN0E; CCU41.IN1E; CCU41.IN2E: CCU41.IN3E;	Service request line

Table 20-26 HRPWM0 - CSG0 Pin Connections

Input/Output	I/O	Connected To	Description
HRPWM0.C0INA	I	PORTS	Analog Input for the High Speed Comparator
HRPWM0.C0INB	I	PORTS	Analog Input for the High Speed Comparator
HRPWM0.ECLK0A	I	CCU41.ST0	External clock that can be used for the slope generator
HRPWM0.ECLK0B	I	ERU1.IOUT0	External clock that can be used for the slope generator

Table 20-26 HRPWM0 - CSG0 Pin Connections

Input/Output	I/O	Connected To	Description
HRPWM0.ECLK0C	I	ERU1.IOUT3	External clock that can be used for the slope generator
HRPWM0.BL0A	I	PORTS	General input that can be used to control the Blanking and Switch of the Comparator
HRPWM0.BL0B	I	SCU.GSHR0	General input that can be used to control the Blanking and Switch of the Comparator
HRPWM0.BL0C	I	CCU80.ST0	General input that can be used to control the Blanking and Switch of the Comparator
HRPWM0.BL0D	I	CCU80.ST1	General input that can be used to control the Blanking and Switch of the Comparator
HRPWM0.BL0E	I	CCU80.ST2	General input that can be used to control the Blanking and Switch of the Comparator
HRPWM0.BL0F	I	CCU80.ST3	General input that can be used to control the Blanking and Switch of the Comparator
HRPWM0.BL0G	I	CCU40.ST0	General input that can be used to control the Blanking and Switch of the Comparator
HRPWM0.BL0H	I	CCU41.ST0	General input that can be used to control the Blanking and Switch of the Comparator
HRPWM0.BL0I	I	HRPWM0.QOUT0	General input that can be used to control the Blanking and Switch of the Comparator
HRPWM0.BL0J	I	HRPWM0.QOUT1	General input that can be used to control the Blanking and Switch of the Comparator
HRPWM0.BL0K	I	CCU40.SR0	General input that can be used to control the Blanking and Switch of the Comparator

Table 20-26 HRPWM0 - CSG0 Pin Connections

Input/Output	I/O	Connected To	Description
HRPWM0.BL0L	I	CCU41.SR0	General input that can be used to control the Blanking and Switch of the Comparator
HRPWM0.BL0M	I	HRPWM0.SR2	General input that can be used to control the Blanking and Switch of the Comparator
HRPWM0.BL0N	I	HRPWM0.SR3	General input that can be used to control the Blanking and Switch of the Comparator
HRPWM0.BL0O	I	ERU1.IOUT0	General input that can be used to control the Blanking and Switch of the Comparator
HRPWM0.BL0P	I	ERU1.IOUT1	General input that can be used to control the Blanking and Switch of the Comparator
HRPWM0.SC0IA	I	NOT CONNECTED	General input that can be used to control the stat/stop/trigger for the Slope Control Logic
HRPWM0.SC0IB	I	SCU.GSHR0	General input that can be used to control the stat/stop/trigger for the Slope Control Logic
HRPWM0.SC0IC	I	CCU80.ST0	General input that can be used to control the stat/stop/trigger for the Slope Control Logic
HRPWM0.SC0ID	I	CCU80.ST1	General input that can be used to control the stat/stop/trigger for the Slope Control Logic
HRPWM0.SC0IE	I	CCU80.ST2	General input that can be used to control the stat/stop/trigger for the Slope Control Logic
HRPWM0.SC0IF	I	CCU80.ST3	General input that can be used to control the stat/stop/trigger for the Slope Control Logic
HRPWM0.SC0IG	I	CCU40.ST0	General input that can be used to control the stat/stop/trigger for the Slope Control Logic

Table 20-26 HRPWM0 - CSG0 Pin Connections

Input/Output	I/O	Connected To	Description
HRPWM0.SC0IH	I	CCU41.ST0	General input that can be used to control the stat/stop/trigger for the Slope Control Logic
HRPWM0.SC0II	I	HRPWM0.QOUT0	General input that can be used to control the stat/stop/trigger for the Slope Control Logic
HRPWM0.SC0IJ	I	HRPWM0.QOUT1	General input that can be used to control the stat/stop/trigger for the Slope Control Logic
HRPWM0.SC0IK	I	CCU40.SR0	General input that can be used to control the stat/stop/trigger for the Slope Control Logic
HRPWM0.SC0IL	I	CCU41.SR0	General input that can be used to control the stat/stop/trigger for the Slope Control Logic
HRPWM0.SC0IM	I	HRPWM0.C0O	General input that can be used to control the stat/stop/trigger for the Slope Control Logic
HRPWM0.SC0IN	I	HRPWM0.SR3	General input that can be used to control the stat/stop/trigger for the Slope Control Logic
HRPWM0.SC0IO	I	ERU1.IOUT0	General input that can be used to control the stat/stop/trigger for the Slope Control Logic
HRPWM0.SC0IP	I	ERU1.IOUT1	General input that can be used to control the stat/stop/trigger for the Slope Control Logic
HRPWM0.C0O	O	CCU80.IN0M; CCU80.IN3P; CCU41.IN0G; HRPWM0.SC0IM; ERU1.2B3	Comparator Output

Table 20-27 HRPWM0 - CSG1 Pin Connections

Input/Output	I/O	Connected To	Description
HRPWM0.C1INA	I	PORTS	Analog Input for the High Speed Comparator
HRPWM0.C1INB	I	PORTS	Analog Input for the High Speed Comparator
HRPWM0.ECLK1A	I	CCU41.ST0	External clock that can be used for the slope generator
HRPWM0.ECLK1B	I	ERU1.IOUT0	External clock that can be used for the slope generator
HRPWM0.ECLK1C	I	ERU1.IOUT3	External clock that can be used for the slope generator
HRPWM0.BL1A	I	PORTS	General input that can be used to control the Blanking and Switch of the Comparator
HRPWM0.BL1B	I	SCU.GSHR0	General input that can be used to control the Blanking and Switch of the Comparator
HRPWM0.BL1C	I	CCU80.ST0	General input that can be used to control the Blanking and Switch of the Comparator
HRPWM0.BL1D	I	CCU80.ST1	General input that can be used to control the Blanking and Switch of the Comparator
HRPWM0.BL1E	I	CCU80.ST2	General input that can be used to control the Blanking and Switch of the Comparator
HRPWM0.BL1F	I	CCU80.ST3	General input that can be used to control the Blanking and Switch of the Comparator
HRPWM0.BL1G	I	CCU40.ST1	General input that can be used to control the Blanking and Switch of the Comparator
HRPWM0.BL1H	I	CCU41.ST1	General input that can be used to control the Blanking and Switch of the Comparator

Table 20-27 HRPWM0 - CSG1 Pin Connections

Input/Output	I/O	Connected To	Description
HRPWM0.BL1I	I	HRPWM0.QOUT1	General input that can be used to control the Blanking and Switch of the Comparator
HRPWM0.BL1J	I	HRPWM0.QOUT2	General input that can be used to control the Blanking and Switch of the Comparator
HRPWM0.BL1K	I	CCU40.SR1	General input that can be used to control the Blanking and Switch of the Comparator
HRPWM0.BL1L	I	CCU41.SR1	General input that can be used to control the Blanking and Switch of the Comparator
HRPWM0.BL1M	I	HRPWM0.SR2	General input that can be used to control the Blanking and Switch of the Comparator
HRPWM0.BL1N	I	HRPWM0.SR3	General input that can be used to control the Blanking and Switch of the Comparator
HRPWM0.BL1O	I	ERU1.IOUT0	General input that can be used to control the Blanking and Switch of the Comparator
HRPWM0.BL1P	I	ERU1.IOUT1	General input that can be used to control the Blanking and Switch of the Comparator
HRPWM0.SC1IA	I	NOT CONNECTED	General input that can be used to control the stat/stop/trigger for the Slope Control Logic
HRPWM0.SC1IB	I	SCU.GSHR0	General input that can be used to control the stat/stop/trigger for the Slope Control Logic
HRPWM0.SC1IC	I	CCU80.ST0	General input that can be used to control the stat/stop/trigger for the Slope Control Logic
HRPWM0.SC1ID	I	CCU80.ST1	General input that can be used to control the stat/stop/trigger for the Slope Control Logic

Table 20-27 HRPWM0 - CSG1 Pin Connections

Input/Output	I/O	Connected To	Description
HRPWM0.SC1IE	I	CCU80.ST2	General input that can be used to control the stat/stop/trigger for the Slope Control Logic
HRPWM0.SC1IF	I	CCU80.ST3	General input that can be used to control the stat/stop/trigger for the Slope Control Logic
HRPWM0.SC1IG	I	CCU40.ST1	General input that can be used to control the stat/stop/trigger for the Slope Control Logic
HRPWM0.SC1IH	I	CCU41.ST1	General input that can be used to control the stat/stop/trigger for the Slope Control Logic
HRPWM0.SC1II	I	HRPWM0.QOUT1	General input that can be used to control the stat/stop/trigger for the Slope Control Logic
HRPWM0.SC1IJ	I	HRPWM0.QOUT2	General input that can be used to control the stat/stop/trigger for the Slope Control Logic
HRPWM0.SC1IK	I	CCU40.SR1	General input that can be used to control the stat/stop/trigger for the Slope Control Logic
HRPWM0.SC1IL	I	CCU41.SR1	General input that can be used to control the stat/stop/trigger for the Slope Control Logic
HRPWM0.SC1IM	I	HRPWM0.C1O	General input that can be used to control the stat/stop/trigger for the Slope Control Logic
HRPWM0.SC1IN	I	HRPWM0.SR3	General input that can be used to control the stat/stop/trigger for the Slope Control Logic
HRPWM0.SC1IO	I	ERU1.IOUT0	General input that can be used to control the stat/stop/trigger for the Slope Control Logic

Table 20-27 HRPWM0 - CSG1 Pin Connections

Input/Output	I/O	Connected To	Description
HRPWM0.SC1IP	I	ERU1.IOUT1	General input that can be used to control the stat/stop/trigger for the Slope Control Logic
HRPWM0.C1O	O	CCU80.IN1N; CCU41.IN1G; HRPWM0.SC1IM; ERU1.3A3	Comparator Output

Table 20-28 HRPWM0 - CSG2 Pin Connections

Input/Output	I/O	Connected To	Description
HRPWM0.C2INA	I	PORTS	Analog Input for the High Speed Comparator
HRPWM0.C2INB	I	PORTS	Analog Input for the High Speed Comparator
HRPWM0.ECLK2A	I	CCU41.ST0	External clock that can be used for the slope generator
HRPWM0.ECLK2B	I	ERU1.IOUT0	External clock that can be used for the slope generator
HRPWM0.ECLK2C	I	ERU1.IOUT3	External clock that can be used for the slope generator
HRPWM0.BL2A	I	PORTS	General input that can be used to control the Blanking and Switch of the Comparator
HRPWM0.BL2B	I	SCU.GSHR0	General input that can be used to control the Blanking and Switch of the Comparator
HRPWM0.BL2C	I	CCU80.ST0	General input that can be used to control the Blanking and Switch of the Comparator
HRPWM0.BL2D	I	CCU80.ST1	General input that can be used to control the Blanking and Switch of the Comparator
HRPWM0.BL2E	I	CCU80.ST2	General input that can be used to control the Blanking and Switch of the Comparator

Table 20-28 HRPWM0 - CSG2 Pin Connections

Input/Output	I/O	Connected To	Description
HRPWM0.BL2F	I	CCU80.ST3	General input that can be used to control the Blanking and Switch of the Comparator
HRPWM0.BL2G	I	CCU40.ST2	General input that can be used to control the Blanking and Switch of the Comparator
HRPWM0.BL2H	I	CCU41.ST2	General input that can be used to control the Blanking and Switch of the Comparator
HRPWM0.BL2I	I	HRPWM0.QOUT2	General input that can be used to control the Blanking and Switch of the Comparator
HRPWM0.BL2J	I	HRPWM0.QOUT3	General input that can be used to control the Blanking and Switch of the Comparator
HRPWM0.BL2K	I	CCU40.SR2	General input that can be used to control the Blanking and Switch of the Comparator
HRPWM0.BL2L	I	CCU41.SR2	General input that can be used to control the Blanking and Switch of the Comparator
HRPWM0.BL2M	I	HRPWM0.SR2	General input that can be used to control the Blanking and Switch of the Comparator
HRPWM0.BL2N	I	HRPWM0.SR3	General input that can be used to control the Blanking and Switch of the Comparator
HRPWM0.BL2O	I	ERU1.IOUT0	General input that can be used to control the Blanking and Switch of the Comparator
HRPWM0.BL2P	I	ERU1.IOUT1	General input that can be used to control the Blanking and Switch of the Comparator
HRPWM0.SC2IA	I	NOT CONNECTED	General input that can be used to control the stat/stop/trigger for the Slope Control Logic

Table 20-28 HRPWM0 - CSG2 Pin Connections

Input/Output	I/O	Connected To	Description
HRPWM0.SC2IB	I	SCU.GSHR0	General input that can be used to control the stat/stop/trigger for the Slope Control Logic
HRPWM0.SC2IC	I	CCU80.ST0	General input that can be used to control the stat/stop/trigger for the Slope Control Logic
HRPWM0.SC2ID	I	CCU80.ST1	General input that can be used to control the stat/stop/trigger for the Slope Control Logic
HRPWM0.SC2IE	I	CCU80.ST2	General input that can be used to control the stat/stop/trigger for the Slope Control Logic
HRPWM0.SC2IF	I	CCU80.ST3	General input that can be used to control the stat/stop/trigger for the Slope Control Logic
HRPWM0.SC2IG	I	CCU40.ST2	General input that can be used to control the stat/stop/trigger for the Slope Control Logic
HRPWM0.SC2IH	I	CCU41.ST2	General input that can be used to control the stat/stop/trigger for the Slope Control Logic
HRPWM0.SC2II	I	HRPWM0.QOUT2	General input that can be used to control the stat/stop/trigger for the Slope Control Logic
HRPWM0.SC2IJ	I	HRPWM0.QOUT3	General input that can be used to control the stat/stop/trigger for the Slope Control Logic
HRPWM0.SC2IK	I	CCU40.SR2	General input that can be used to control the stat/stop/trigger for the Slope Control Logic
HRPWM0.SC2IL	I	CCU41.SR2	General input that can be used to control the stat/stop/trigger for the Slope Control Logic
HRPWM0.SC2IM	I	HRPWM0.C2O	General input that can be used to control the stat/stop/trigger for the Slope Control Logic

Table 20-28 HRPWM0 - CSG2 Pin Connections

Input/Output	I/O	Connected To	Description
HRPWM0.SC2IN	I	HRPWM0.SR3	General input that can be used to control the stat/stop/trigger for the Slope Control Logic
HRPWM0.SC2IO	I	ERU1.IOUT0	General input that can be used to control the stat/stop/trigger for the Slope Control Logic
HRPWM0.SC2IP	I	ERU1.IOUT1	General input that can be used to control the stat/stop/trigger for the Slope Control Logic
HRPWM0.C2O	O	CCU80.IN2O; CCU41.IN2G; HRPWM0.SC2IM; ERU1.3B3	Comparator Output

Table 20-29 HRPWM0 - HRC0 Pin Connections

Input/Output	I/O	Connected To	Description
HRPWM0.HROUT00	O	PORTS	High resolution Output
HRPWM0.HROUT01	O	PORTS	High resolution Output
HRPWM0.QOUT0	O	CCU41.IN0F; CCU41.IN1H; CCU41.IN2H; CCU41.IN3H; HRPWM0.BL0I; HRPWM0.SC0I;	High resolution output without dead time. Used to measure the dutyc cycle of the PWM, to switch/start the a CSG unit or a timer, or for pattern generation.

Table 20-30 HRPWM0 - HRC1 Pin Connections

Input/Output	I/O	Connected To	Description
HRPWM0.HROUT10	O	PORTS	High resolution Output
HRPWM0.HROUT11	O	PORTS	High resolution Output
HRPWM0.QOUT1	O	CCU41.IN1F; HRPWM0.BL0J; HRPWM0.SC0J; HRPWM0.BL1I; HRPWM0.SC1I;	High resolution output without dead time. Used to measure the dutyc cycle of the PWM, to switch/start the a CSG unit or a timer, or for pattern generation.

Table 20-31 HRPWM0 - HRC2 Pin Connections

Input/Output	I/O	Connected To	Description
HRPWM0.HROUT20	O	PORTS	High resolution Output
HRPWM0.HROUT21	O	PORTS	High resolution Output
HRPWM0.QOUT2	O	CCU41.IN2F; HRPWM0.BL1J; HRPWM0.SC1J; HRPWM0.BL2I; HRPWM0.SC2I;	High resolution output without dead time. Used to measure the dutyc cycle of the PWM, to switch/start the a CSG unit or a timer, or for pattern generation.

Table 20-32 HRPWM0 - HRC3 Pin Connections

Input/Output	I/O	Connected To	Description
HRPWM0.HROUT30	O	PORTS	High resolution Output
HRPWM0.HROUT31	O	PORTS	High resolution Output
HRPWM0.QOUT3	O	CCU41.IN3F; HRPWM0.BL2J; HRPWM0.SC2J;	High resolution output without dead time. Used to measure the dutyc cycle of the PWM, to switch/start the a CSG unit or a timer, or for pattern generation.

21 Position Interface Unit (POSIF)

The POSIF unit is a flexible and powerful component for motor control systems that use Rotary Encoders or Hall Sensors as feedback loop. The several configuration schemes of the module, target a very large universe of motor control application requirements. This enables the build of simple and complex control feedback loops, for industrial and automotive motor applications, targeting high performance motion and position monitoring.

Table 21-1 Abbreviations table

PWM	Pulse Width Modulation
POSIFx	Position Interface module instance x
CCU8x	Capture/Compare Unit 8 module instance x
CCU4x	Capture/Compare Unit 4 module instance x
CC8y	Capture/Compare Unit 8 Timer Slice instance y
CC4y	Capture/Compare Unit 4 Timer Slice instance y
SCU	System Control Unit
f_{posif}	POSIF module clock frequency

Note: A small “y” or “x” letter in a register indicates an index

21.1 Overview

The POSIF module is comprised of three major sub units, the Quadrature Decoder unit, the Hall Sensor Control unit and the Multi-Channel Mode unit.

The Quadrature Decoder Unit is used for position control linked with a rotary incremental encoder.

The Hall Sensor Control Unit is used for direct control of brushless DC motors.

The Multi-Channel Mode unit is used in conjunction with the Hall Sensor mode to output the wanted motor control pattern but also can be used in a stand-alone manner to perform a simple multi-channel control of several control units.

The POSIF module is used in conjunction with a CCU4 or CCU8 which enables a very flexible resource arrangement and optimization for any type of application.

21.1.1 Features

POSIF module features

The POSIF is built of three dedicated control units that can operate in a stand-alone manner.

The Quadrature Decoder Unit contains an output interface that enables position and velocity measurements when linked with a CCU4 module. The Hall Sensor Unit enables the control of a multi-channel pattern, that can be linked with up 8 output control sources. The Multi-Channel unit offers a complete built-in interaction with the Hall Sensor Mode and an easy stand-alone control loop.

General Features

- Quadrature Decoder
  - interface for position measurement
  - interface for motor revolution measurement
  - interface for velocity measurement
  - interrupt sources for phase error, motor revolution, direction change and error on phase decoding
- Hall Sensor Mode
  - Simple build-in mode for brushless DC motor control
  - Shadow register for the multi-channel pattern
  - Complete synchronization with the PWM signals and the multi-channel pattern update
  - interrupt sources for Correct Hall Event detection, Wrong Hall Event Detection
- Multi-Channel Mode
  - Simple usage with Hall Sensor Mode
  - stand-alone Multi-Channel mode
  - Shadow register for the multi-channel pattern

Additional features

- Quadrature Decoder mode can be used in parallel with the stand-alone Multi-Channel mode
- Several profiles (via CCU4) to perform position and velocity measurement for the Quadrature Decoder mode
- Programmable delay times (via CCU4) for input pattern evaluation and new pattern value for the Hall Sensor Mode

POSIF features vs. applications

On **Table 21-2** a summary of the major features of the POSIF unit mapped with the most common applications.

Table 21-2 Applications summary

Feature	Applications
Quadrature Decoder Mode	Easy plug-in for rotary encoders: • with or without index/top marker signal • gear-slip or shaft winding compensation • separate outputs for position, velocity and revolution control - matching different system requirements • extended profile for position tracking - with revolution measurement and multiple position triggers for each revolution • support for high dynamic speed changes due to tick-to-tick and tick-to-sync capturing method
Hall Sensor Mode	Easy plug-in for Motor control using Hall Sensors: • 2 or 3 Hall sensor topologies • extended input filtering to avoid unwanted pattern switch due to noisy input signals • synchronization with the PWM signals of the Capture/Compare Unit • Active freewheeling/synchronous rectification with dead time support (link with Capture/Compare Unit 8) • easy velocity measurement function by using a Capture/Compare unit Timer Slice
Multi-Channel Mode	Modulating multiple PWM signals: • parallel modulation controlled via SW for N PWM signals - for systems with multiple power converters • generating proprietary PWM modulations • parallel and synchronous shut down of N PWM signals due to system feedback

21.1.2 Block Diagram

Each POSIF module can operate in three different modes, Quadrature Decoder, Hall Sensor and stand-alone Multi-channel Mode. To complete the control/measurement loop of all these three modes, the POSIF needs to be linked with a CCU4/CCU8 module.

In the case of the Quadrature Decoder mode, one CCU4 unit is needed, for the Hall Sensor Mode, one needs one slice of one CCU4 and a CCU8 unit. The connectivity between the stand-alone Multi-Channel mode and CCU4/CCU8 module(s) does not follow any usage constraints.

Each POSIF module contains 30 inputs and 25 outputs (including service requests), **Figure 21-1**, that are going to be mapped to available functions of the module, depending in which configuration it was selected by the user: Quadrature Decoder, Hall Sensor or stand-alone Multi-Channel.

The module also has two dedicated Service request Lines, see **Section 21.3**.

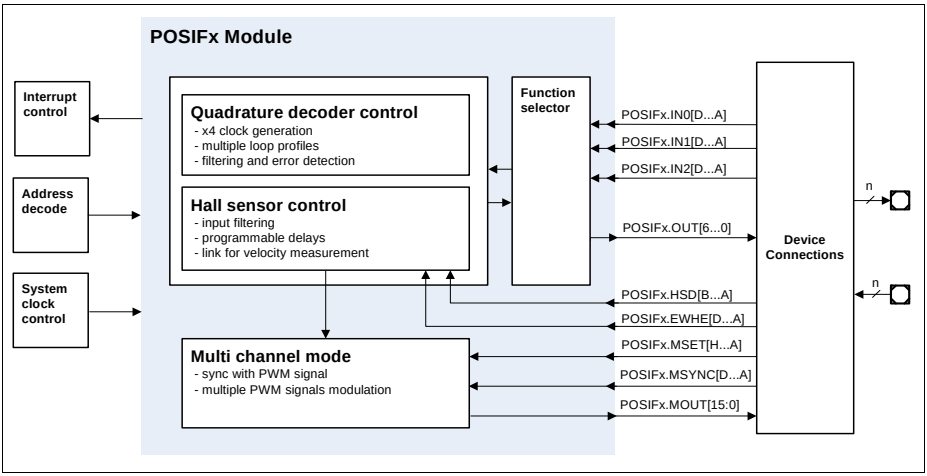


Figure 21-1 POSIF block diagram

21.2 Functional Description

21.2.1 Overview

The POSIF module contains a function selector unit, that is used in parallel by both Quadrature Decoder and Hall Sensor Control units. This block selects which input signals should be decoded for each control unit. The function selector is also decoding the outputs coming from these two modes (Quadrature and Hall Sensor Mode).

The POSIF module also contains a subset of inputs that are only used in Hall Sensor/Multi-Channel Mode. These inputs are connected to a timer structure (CCU4/CCU8) and are used to control the delays between pattern sampling, multi-channel pattern update and synchronization, etc.

The Multi-Channel unit contains 16 dedicated outputs, that contain the current multi-channel pattern and that can be connected to a CCU8/CCU4.

The POSIF control unit contain a dedicated Run bit, that can be set/clear by SW. It can also generate a Sync start signal that can be connected to the timer units (CCU4/CCU8).

For the Quadrature Decoder Mode, there is the possibility to define which of the signals is the leading phase and also the active level for each one.

The Quadrature Decoder Mode can also receive the clock and direction signals directly from an external source.

The Multi-Channel offers a shadow register, for the Multi-Channel pattern, enabling this way an update on the fly of these parameter. A write access always addresses the shadow register while a read, always returns the actual value of the Multi-Channel pattern.

Table 21-3 POSIF slice pin description

Pin	I/O	Hall Sensor Mode	Quadrature Decoder Mode	Multi-Channel Mode (stand-alone)
POSIFx.IN0[D...A]	I	Hall Input 1	Encoder Phase A or Clock	Not used
POSIFx.IN1[D...A]	I	Hall Input 2	Encoder Phase B or Direction	Not used
POSIFx.IN2[D...A]	I	Hall Input 3	Index/Zero marker	Not used
POSIFx.HSD[B...A]	I	Hall pattern sample delay	Not used	Not used
POSIFx.EWHE[D...A]	I	Wrong hall event emulation	Not used	Wrong hall event emulation
POSIFx.MSET[H...A]	I	Multi-Channel next pattern update set	Not used	Multi-Channel next pattern update set
POSIFx.MSYNC[D...A]	I	Multi-Channel pattern update synchronization	Not used	Multi-Channel pattern update synchronization
POSIFx.OUT0	O	Hall inputs edge detection trigger	Quadrature clock	Not used

Table 21-3 POSIF slice pin description (cont'd)

Pin	I/O	Hall Sensor Mode	Quadrature Decoder Mode	Multi-Channel Mode (stand-alone)
POSIFx.OUT1	O	Hall Correct event	Direction	Not used
POSIFx.OUT2	O	Idle/ wrong hall event	Period clock	Not used
POSIFx.OUT3	O	Stop	Clear/capt	Not used
POSIFx.OUT4	O	Multi-Channel pattern update	Index	Not used
POSIFx.OUT5	O	Sync start	Sync start	Not used
POSIFx.OUT6	O	Multi Pattern sync trigger	Not used	Multi Pattern sync trigger
POSIFx.MOUT[15:0]	O	Multi-Channel pattern	Not used	Multi-Channel pattern
POSIFx.SR0	O	Service request line 0	Service request line 0	Service request line 0
POSIFx.SR1	O	Service request line 1	Service request line 1	Service request line 1

Note: The Service Request signals at the output of the kernel are extended for one more kernel clock cycle.

21.2.2 Function Selector

The Function selector maps the input function signals to the selected operating mode, whether Quadrature or Hall Sensor Mode, **Figure 21-2**. The outputs are also decoded throughout this unit.

For each function input, POSI0, POSI1 and POSI2, the user can select one of the 4 input pins via the fields **PCONF**.INSEL0, **PCONF**.INSEL1 and **PCONF**.INSEL2. It is also possible to perform a low pass filtering on the three inputs, the field **PCONF**.LPC controls the low pass filters cut frequency.

The Hall Sensor Mode is the default function, **PCONF**.FSEL = 00_B. In this mode, the Function selector maps the POSI0 to the Hall Input 1, the POSI1 to the Hall input 2 and the POSI2 to the Hall input 3.

When the Quadrature Decoder mode is selected, **PCONF**.FSEL = 01_B, POSI0 is mapped to Phase A or Clock, POSI1 to the Phase B or Direction and POSI2 to the Index signals coming from the rotary motor encoder. Notice that it is also possible to select the

Quadrature Decoder and stand-alone Multi-Channel mode, by setting **PCONF**.FSEL = 11_B.

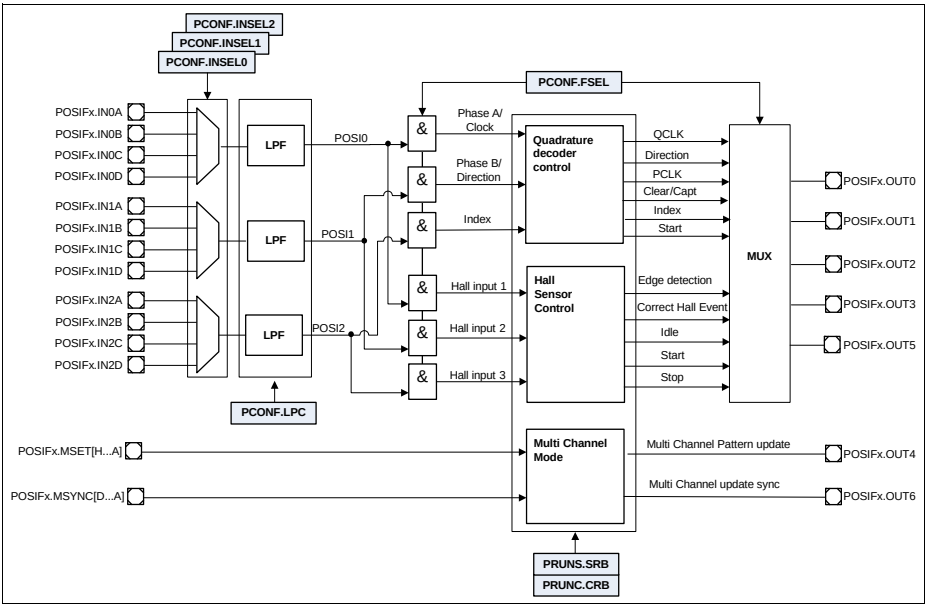


Figure 21-2 Function selector diagram

21.2.3 Hall Sensor Control

The Hall Sensor mode is divided in three major loops: detection of any update in the Hall inputs, delay between the detection and the sampling of the Hall inputs for comparison against the expected Hall Pattern and the update of the Multi-Channel pattern.

The Hall inputs are directly connected to an edge detection circuitry and with any modification in any of these three inputs, a signal is generated on the POSIFx.OUT0 pin, see **Figure 21-3**. This pin can be connected to one CCU4 slice that is controlling the delay between the edge detection and the next step on the Hall sensor mode, the sampling of the Hall Inputs.

The signal used to trigger the sample of the Hall inputs is selected via the **PCONF**.DSEL field, and this trigger can be active at the rising or falling edge (**PCONF**.SPES).

When the sampling trigger is sensed, the Hall inputs are sampled and compared against the Current Pattern, **HALP**.HCP and the Expected Hall Pattern, **HALP**.HEP (to evaluate if the input pattern match the HEP or HCP values).

The edge detection circuit generates an enable signal for sampling the hall inputs, PIFHSP and the sample logic generates a pulse every time that new values are captured, PIFHRDY, that is used inside the pattern compare logic.

When the sampled value matches the Expected Hall Pattern, a pulse is generated in the POSIFx.OUT1 pin to indicate a correct hall event. At the same time the next values programmed into the shadow registers are loaded. The **HALP**.HCP[LSB] is linked to the Hall input 1, and the **HALP**.HCP[MSB] is linked to the Hall input 3 (the same is applicable for the **HALP**.HEP register).

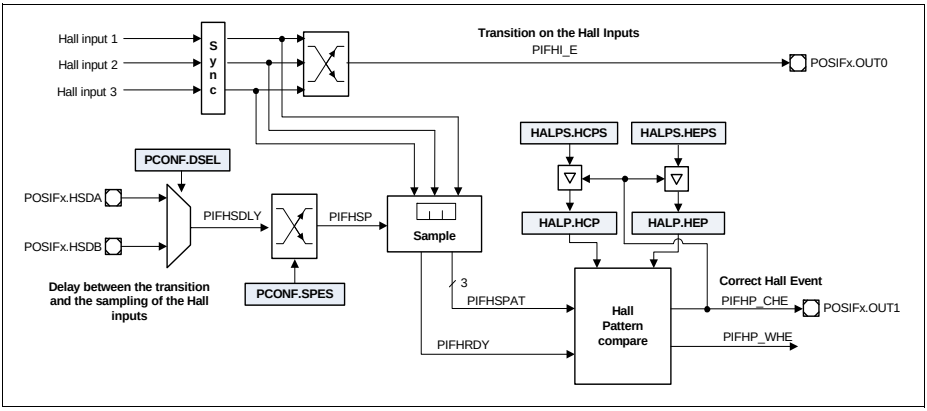


Figure 21-3 Hall Sensor Control Diagram

When the sampled value matches the Current Hall Pattern, a line glitch deemed to have occurred and no action is taken. When the sampled value does not match any of the values (the current and the expected pattern), a major error is deemed to have occurred and the Wrong Hall Event signal is generated. Every time that a sampled pattern leads to a wrong hall event or when it matches the current pattern a stop signal is generated throughout the POSIFx.OUT3 pin, **Figure 21-4**.

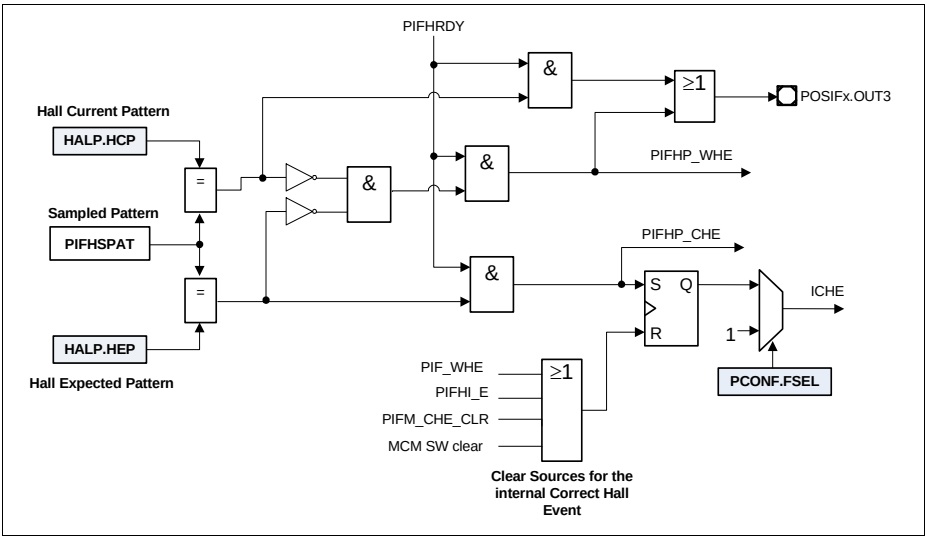


Figure 21-4 Hall Sensor Compare logic

A wrong hall event can generate an IDLE signal that is connected to the POSIFx.OUT2 and can be used to clear the run bit of the Hall Sensor Control unit.

The IDLE signal can also be connected to the PWM unit to perform a forced stop operation, **Figure 21-5**. The wrong hall event/idle function can also be controlled via a pin.

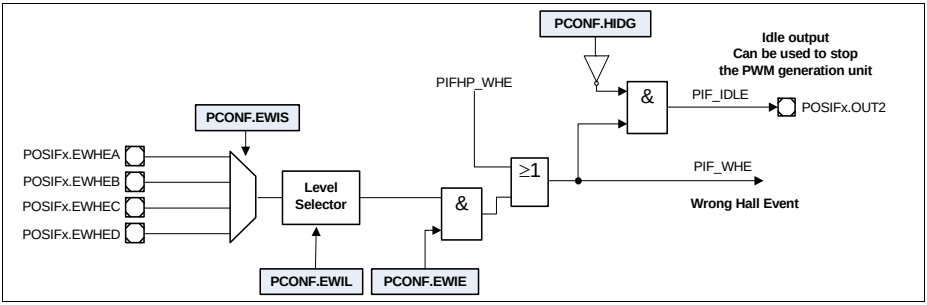


Figure 21-5 Wrong Hall Event/Idle logic

After the Correct Hall Event is detected, a delay can be generated between this detection and the update of the Multi-Channel pattern. On **Figure 21-6** it is demonstrated the control logic of the Multi-Channel mode.

The delay for the update of the Multi-Channel pattern can be controlled directly by a CCU4 slice. The trigger that indicates that the delay is finished, can be mapped to one of the input signals POSIFx.MSET[H...A] (**PCONF**.MSETS selects the input signal used for this purpose). One can also select the active edge for the trigger via **PCONF**.MSES.

The **PCONF**.MCUE field selects the source that enables an update of the Multi-Channel pattern. When set to 1_B, the Multi-Channel pattern can only be updated after the SW has written a 1_B into the **MCMS**.MNPS field.

After the update delay, the Multi-Channel pattern still needs to be synchronized with the PWM signal. For this, the user selects a signal from the POSIFx.MSYNC[D...A] inputs via **PCONF**.MSYNS field. When a falling edge is detected in this signal, then the new multi pattern is applied at the POSIFx.MOUT[15:0] outputs, with the **MCM**.MCMP[15] linked to the POSIFx.MOUT[15] and **MCM**.MCMP[0] to POSIFx.MOUT[0].

The POSIFxOUT6 pin is connected to the **MCMF**.MSS register field. This register field is enabling the Multi-Channel pattern update (that is done upon receiving the sync signal, PIFMSYNC) and can be used in conjunction with a CAPCOM module to perform a synchronous update of the Multi-Channel pattern and the compare values used inside of the CAPCOM.

When a wrong hall event is configured to set the Hall Sensor Control into IDLE, the Multi-Channel pattern is also cleared.

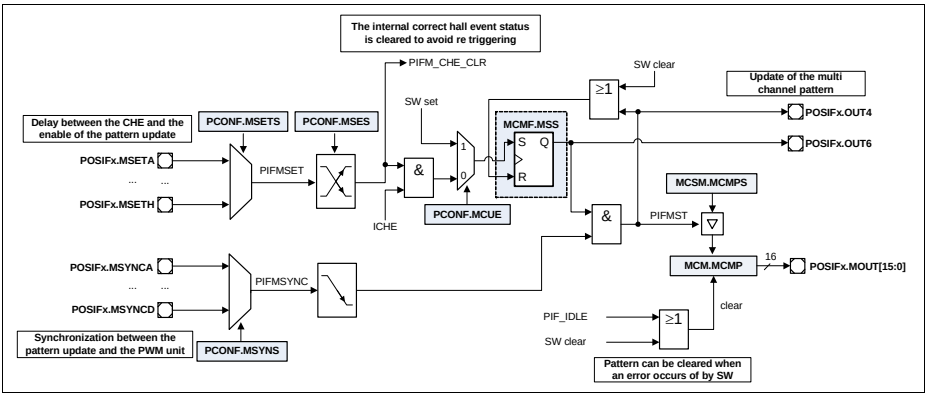


Figure 21-6 Multi-Channel Mode Diagram

Figure 21-7 shows all the previous described steps in the Hall Sensor Mode. Every time that a transition on a Hall input is detected, the pin POSIFx.OUT0 is asserted. This signal is used to start the timing delay between the transition detection and the sampling of the hall inputs.

In this scenario the status output (ST in **Figure 21-7**) of a timer cell was used to control the timing 1 (delay between the transition and the sampling of the hall inputs). With the

rising edge of the ST signal, the hall inputs are sampled and if they match the expected pattern, signal POSIFx.OUT1 is asserted.

A service request line (SR signal) mapped to the same timer cell is used to control the delay between a correct Hall Event and the update of the Multi-Channel pattern. This service request is asserted when a period match is detected.

Another timer cell is used to measure the time between each correct hall event. This timer cell is represented by the Timing 2 on **Figure 21-7** (the CR symbolizes the capture register of the timer cell).

After the delay controlled by the Timing 1 (SR signal) is over, the update of the Multi-Channel pattern needs to be synchronized with the PWM signal. This is done by using one of the pattern synchronization outputs of the Capture/Compare Unit that is used to generate the PWM signals (as an example a CCU8 was used).

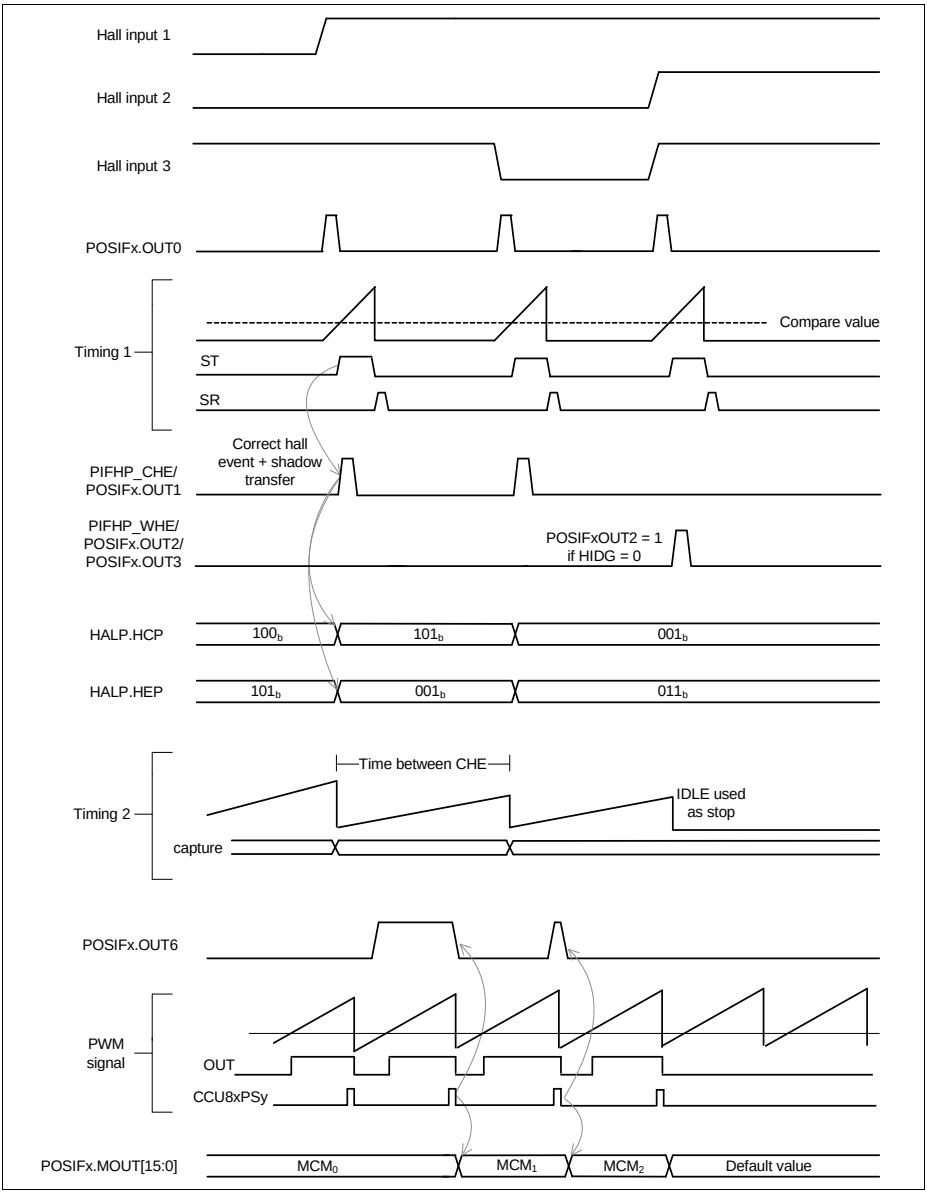


Figure 21-7 Hall Sensor timing diagram

21.2.4 Quadrature Decoder Control

The Quadrature Decoder Mode is selected by setting **PCONF**.FSEL = 01_B or **PCONF**.FSEL = 11_B (in this case the Multi-Channel mode is also enabled).

Inside the Quadrature Decoder Mode, two different subsets are available:

- standard Quadrature Mode
- Direction Count Mode

The standard mode is used when the external rotary encoder provides two phase signals and additionally and index/marker signal that is generated once per shaft revolution. The Direction Count Mode is used when the external encoder only provides a clock and a direction signal, **Figure 21-8**.

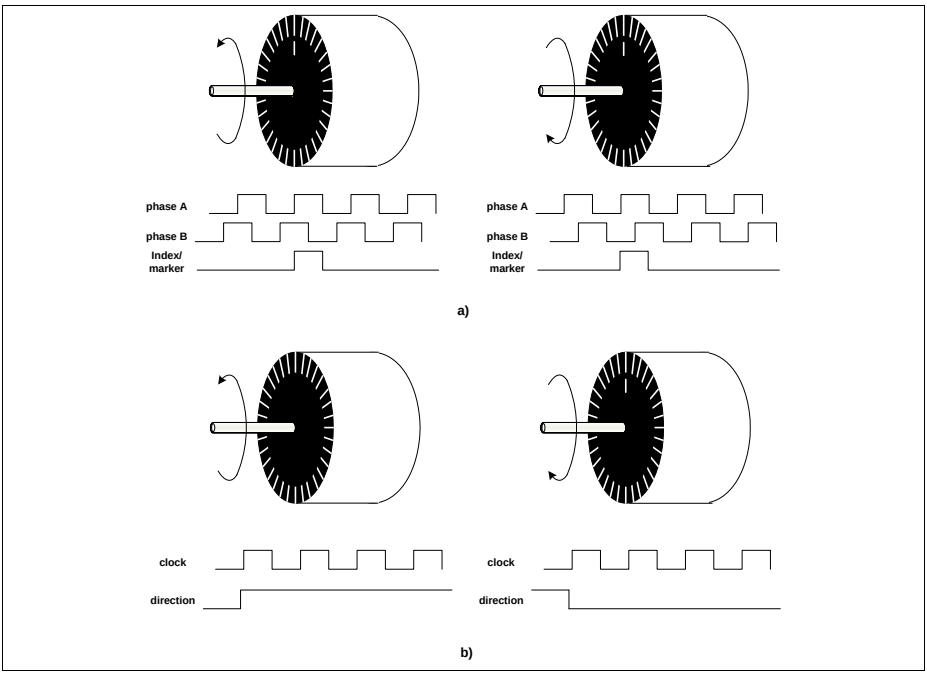


Figure 21-8 Rotary encoder types - a) standard two phase plus index signal; b) clock plus direction

Standard Quadrature Mode

The Quadrature Decoder unit offers a very flexible PhaseA/PhaseB configuration stage. Normally for a clockwise motor shaft rotation, Phase A should precede Phase B but nevertheless, the user can configure the leading phase as well the specific active state for each signal, **Figure 21-9**.

There are two major blocks that build the Quadrature Decoder Control unit: the block that decodes the quadrature clock and motor shaft direction and the block that handles the index (motor revolution) control.

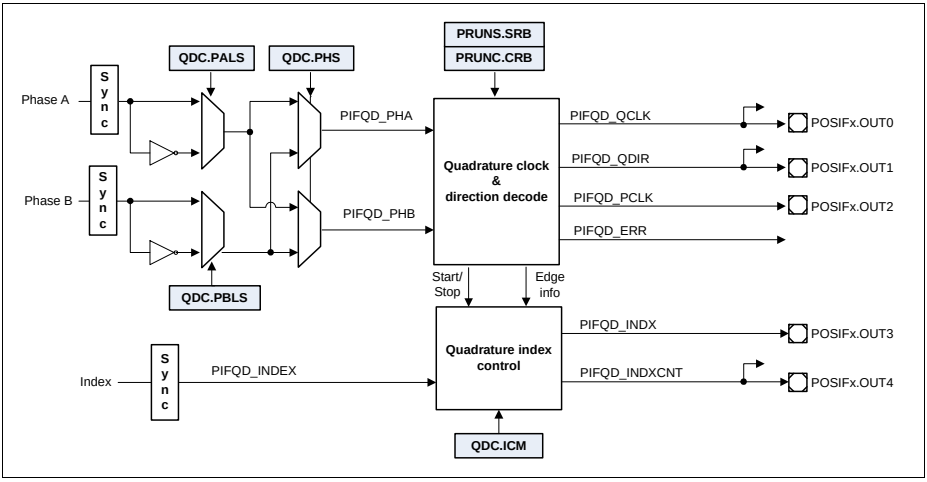


Figure 21-9 Quadrature Decoder Control Overview

The quadrature clock is connected to pin POSIFx.OUT0 and is used for position measurement. This clock is decoded from every edge of the phase signals and therefore there are 4 clock pulses per phase period.

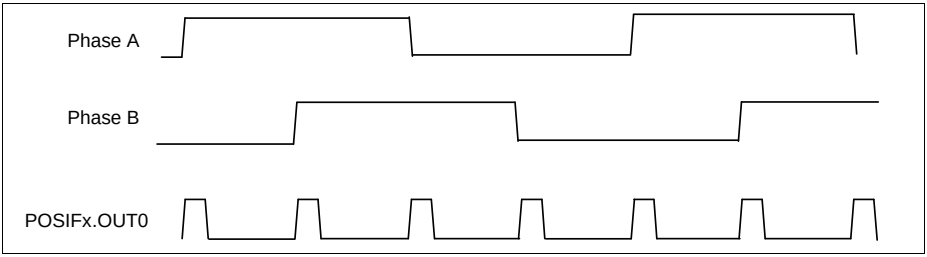


Figure 21-10 Quadrature clock generation

A period clock is also generated for velocity measurement operations.

The direction of the motor rotation is connected to the POSIFx.OUT1 pin and is asserted HIGH when the motor is rotating clockwise and LOW when it is turning in the counterclockwise direction.

The index control logic, memorizes which was the first edge after the assertion of the index signal, so the same quadrature transition is used for index event operations. It also

memorizes the direction of the first index, so that it can control when the revolution increment signal should be asserted.

An error signal, connected to a flag (and if enable an interrupt can be generated) also can be generated when a wrong phase pair is detected.

The Quadrature Decoder Control, uses the information of the current and previous phase pair to decode the direction and clocks. Both phase signals pass through a edge detection logic, from which the outputs are going to be used against the valid/invalid transitions stages, see **Figure 21-11**. There is the possibility to reset the decoder state machine (but not the flags and static configuration) by writing a 1_B into the **PRUNC**.CSM field.

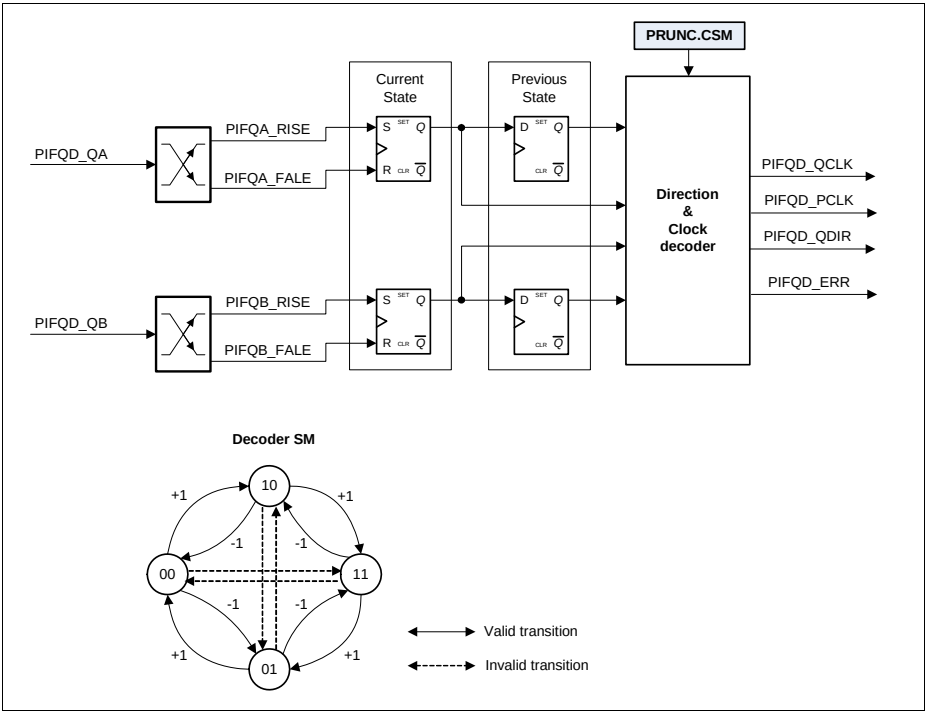


Figure 21-11 Quadrature Decoder States

Direction Count Mode

Some position encoders do not have the phase signals as outputs, instead, they provide two signals that contain the clock and direction information. This is know as the Direct Count Mode.

When using a position encoder of this type, the user should set the **PCONF**.QDCM bit field to 1_B (enabling the direction count mode). In this case, the signal selected from the POSIFx.IN0[D...A] is used as clock and the selected signal from the POSIFx.IN1[D...A] contains the direction information.

The input signals are synchronized with the POSIF module clock and sent to the respective outputs. In this case the inputs and outputs linked with the index/zero marker are not used. The POSIFx.OUT2 output is also inactive.

21.2.4.1 Quadrature Clock and Direction decoding

The Quadrature Decoder unit outputs two clocks. One clock is used for position control and therefore is generated in each edge transition of the phase signals. The second clock is used for velocity measurements and is immune to possible glitches on the phase signals that can be present at very slow rotation speeds. These glitches are not normal line noise, but are due to the slow movement of the engine.

The decoding of the direction signal is done following the rules on **Figure 21-11**. **Figure 21-12** shows all the valid transitions of Phase A and Phase B signals.

Figure 21-13 shows the difference between the two clock signals in the case that some glitches are present in the phase signals.

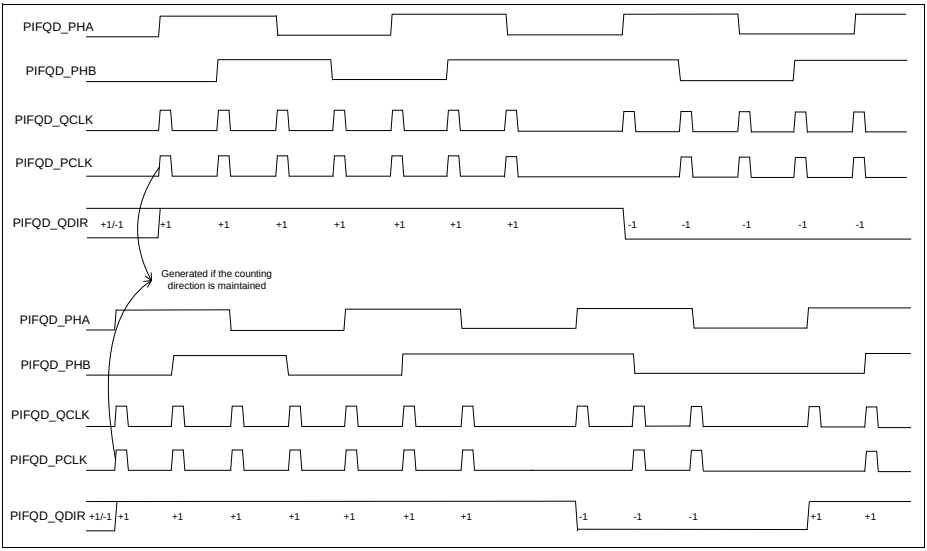


Figure 21-12 Quadrature clock and direction timings

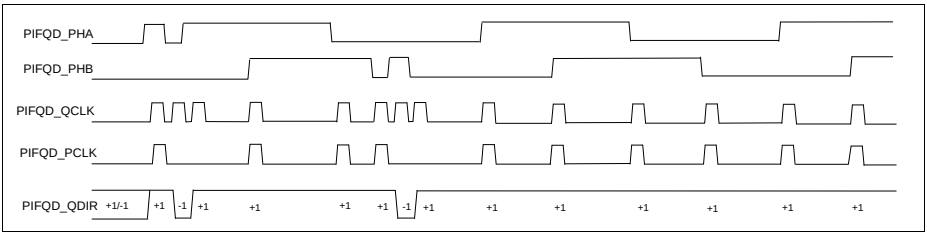


Figure 21-13 Quadrature clock with jitter

21.2.4.2 Index Control

The Index Control logic has two different outputs. One is asserted every time that an index signal is detected (and the motor shaft rotation is the same), POSIFx.OUT4, and therefore it can be used in a revolution counter. With this, the SW can monitor not only the position of the shaft but also the total number of revolutions that have occurred.

The activity of the other output, POSIFx.OUT3, can be programmed via the **QDC**.ICM field. Depending on the value set in the **QDC**.ICM, this signal can be generated:

- in every index signal occurrence
- only on the first index signal occurrence
- disabled - this outputs is never asserted

This is useful for applications that need to reset the counters on every index event or only at the first one.

The Index Control logic memorizes the immediately next phase edge that follows a index so the generated signals have always the same reference. If the first phase edge after the index is the rising edge of Phase B signal, then the index signals are going to be generated in the next index event with the rising edge of the Phase B signal if the direction is kept or with the falling edge of Phase B signal if the direction has changed.

Figure 21-14 shows the timing diagram for the generation of the index signals. In this case the **QDC**.ICM = 10_B, which means that the POSIFx.OUT3 index signal is going to be generated in every occurrence of the input index.

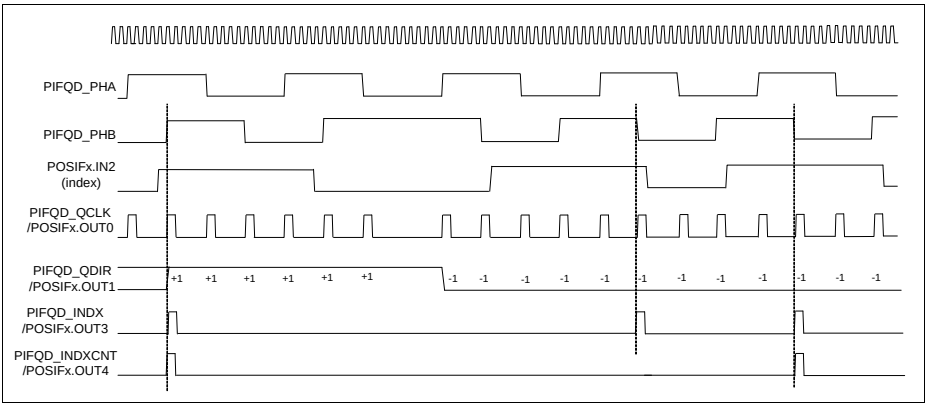


Figure 21-14 Index signals timing

21.2.5 Stand-Alone Multi-Channel Mode

The Multi-Channel mode (Multi-Channel Mode logic can be see on **Figure 21-6**) can be used without the Hall Sensor Control, by setting the **PCONF**.FSEL = 10_B or if the Quadrature Decoder Mode is also needed, by setting **PCONF**.FSEL = 11_B.

In stand-alone Multi-Channel Mode, the mechanism to update the multi-channel pattern is the same as the one described in **Section 21.2.3**.

The trigger for a pattern update can come from the SW, by writing 1_B to **MCMS**.MNPS, or it can come from an external signal like in Hall sensor Mode. This external signal should then be mapped to the PIFMSET function by selecting the appropriate value in the **PCONF**.MSETS field.

The synchronization between the update of the new Multi-Channel pattern and control signal still needs to be done. The user needs to map one input signal of the POSIFx.MSYNC[D...A] range to this function.

The SW has the possibility of clearing the actual Multi-Channel pattern, by writing 1 into the **MCMC**.MPC field.

21.2.6 Synchronous Start

The POSIF module has a synchronous start output, POSIFx.OUT5, that can be used together with the CAPCOM for a complete synchronous start of both modules. The synchronous start is linked with the run bit of the POSIF module, which means that every time that the run bit is set, a pulse is generated throughout the POSIFx.OUT5 pin.

By using the synchronous start output, the SW does not perform two independent accesses to start the POSIF and the CCU4/CCU8 and therefore is guaranteed that both modules start their operation at the exact same time.

21.2.7 Using the POSIF

The POSIF module needs to be linked with a CCU4/CCU8 module to perform the full set of functions in each of the possible modes (due to the fact that doesn't contain built-in counters/timers).

To operate the POSIF in the Quadrature Decoder Mode, one CCU4 module is needed. The Hall Sensor Mode, needs a CCU8 (at least 3 slices are need to control a brushless DC motor) and also (at least) two CCU4 or CCU8 slices. The stand-alone Multi-Channel Mode linking configuration depends heavily on the use cases and therefore the number of slices of CCU4 or CCU8 used is freely chosen by the user.

21.2.7.1 Hall Sensor Mode Usage

When using the Hall Sensor Mode of the POSIF, the Multi-Channel Mode is also working. Due to that fact, the CCU8 module used to perform the Multi-Channel Modulation, needs to be configured in Multi-Channel Mode.

Standard Hall Sensor Mode Usage

On **Figure 21-15**, the Hall Sensor Mode is used in conjunction with two CCU4 slices and one CCU8 module. The first slice of CCU4, slice 0 is being used to control the delays between the edge detection of the Hall Inputs and the actual sampling, and also to control the delay between a Correct Hall Event and the Multi-Channel Pattern update enable.

The rising edge of the CCU4x.ST0 is used as finish trigger for the first delay, while the Service request line is used for triggering the update of a new pattern after a Correct Hall Event. The service request is configured to be active on each period match hit of the specific slice.

Slice 0 is configured in single shot mode, so that the time delay can be re triggered every time that a request from the POSIF occurs.

The second slice of the CCU4 unit, Slice 1, is being used in Capture Mode, to capture the time between Correct Hall Events (storing this way the motor speed between two correct hall events). The POSIFx.OUT1 of the POSIF is used as capture trigger for the slice while the POSIFx.OUT3 is used as stop. The capture and stop triggers are configured in the specific timer slices as active on the rising edge.

The CCU8 is the module that is generating the PWM signals to control the motor and therefore, the Multi-Channel Pattern outputs POSIFx.MOUT[7:0] are linked to this unit.

To close the Multi-Channel loop, an output of the CCU8 needs to be connected to the POSIF module (one of the CCU8x.PSy signals), so that the Multi-Channel Pattern is updated synchronously with the PWM period.

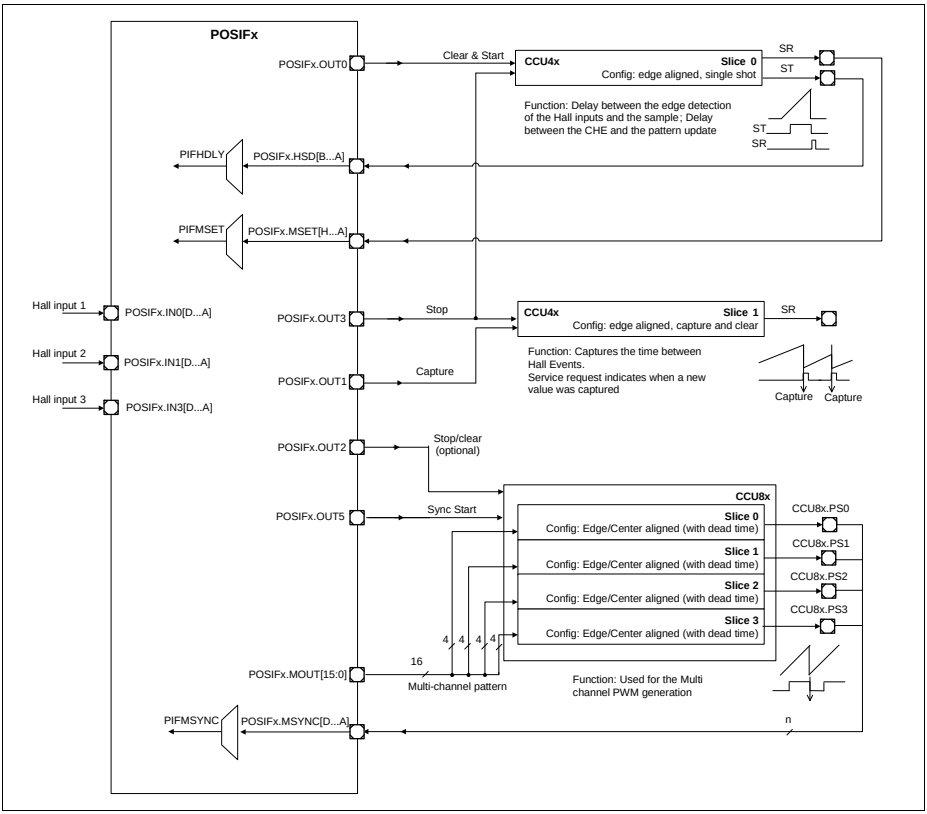


Figure 21-15 Hall Sensor Mode usage - profile 1

Hall Sensor Mode Usage - Flexible Time Control

On **Figure 21-16**, another profile is demonstrated. In this case instead of 2 CCU4 slices, the Hall Sensor Mode is using 3 CCU4 slices. This profile gives more flexibility in terms of delay configuration. It uses two timer slices to control independently the delay between the transition of the hall inputs and sampling, and the delay between a Correct Hall Event and the update of the Multi-Channel pattern. At the same time it also removes the need of using a service request to control the pattern update delay.

Slice 0 is used to control the delay between a transition at the hall inputs and the actual sampling. Slice 2 is used to control the delay between a Correct Hall Event and the update of the Multi-Channel pattern.

Slice 1 is used as keeper of the time stamp between Correct Hall events (the same function as Slice 1 in profile 1).

The synchronism between the PWM signal and the update of the Multi-Channel pattern is again done with one of the CCU8x.PSy outputs.

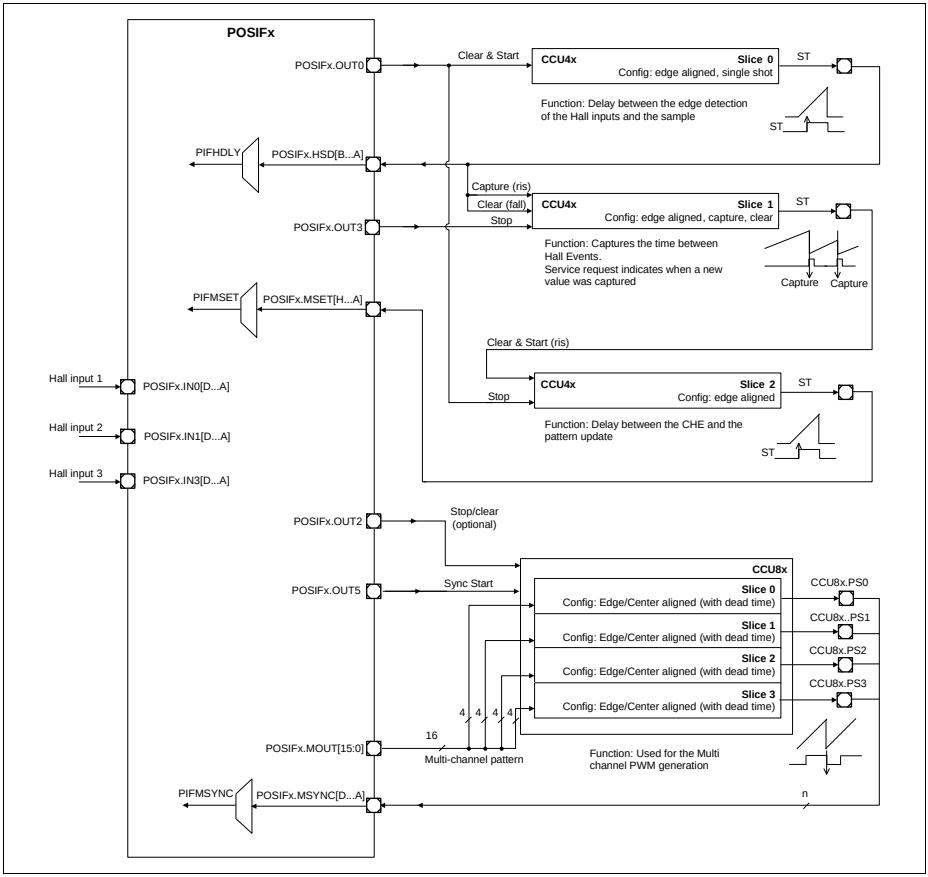


Figure 21-16 Hall Sensor Mode usage - profile 2

21.2.7.2 Quadrature Decoder Mode usage

The Quadrature Decoder Mode can be used in a very flexible way when connected with a CCU4 unit. The connection profile depends on the amount of functions that the user wants to perform in parallel: position control, revolution control and velocity measurement.

Quadrature Decoder Usage - Tick and Revolution Compare plus Velocity Between N Ticks

On **Figure 21-17**, the POSIF is linked with a CCU4, using all the module timer slices. In this profile, the user in Quadrature Decoder Mode has a slice being used for position control (comparison), one for revolutions counter and two aggregated slices used for velocity measurement.

Slice 0 is connected to the POSIFx.OUT0 and POSIFx.OUT1 outputs, which means that one has to configure POSIFx.OUT0 as counting functionality and POSIFx.OUT1 as Up/Down counting function. This slice is then used to track the actual position of the system and the compare channel can be configured to trigger the required actions, when the position reaches a certain value.

Slice 1 is connected to POSIFx.OUT4 pin, which means that it is used as a motor revolution counter. The compare channel can be programmed to trigger an interrupt every time that the motor performs N revolutions.

Slice 2 and Slice 3 are used to perform velocity measurements. Slice 2 receives the Quadrature Decoder period clock via POSIFx.OUT2, that is mapped to a counting functionality. The compare channel of this slice is then used to trigger a capture event in Slice3. The last one is using the module clock and therefore in every capture event, the actual system ticks are captured. This way the user knows how much time has elapsed between N phase periods and can calculate the corresponding velocity profiles.

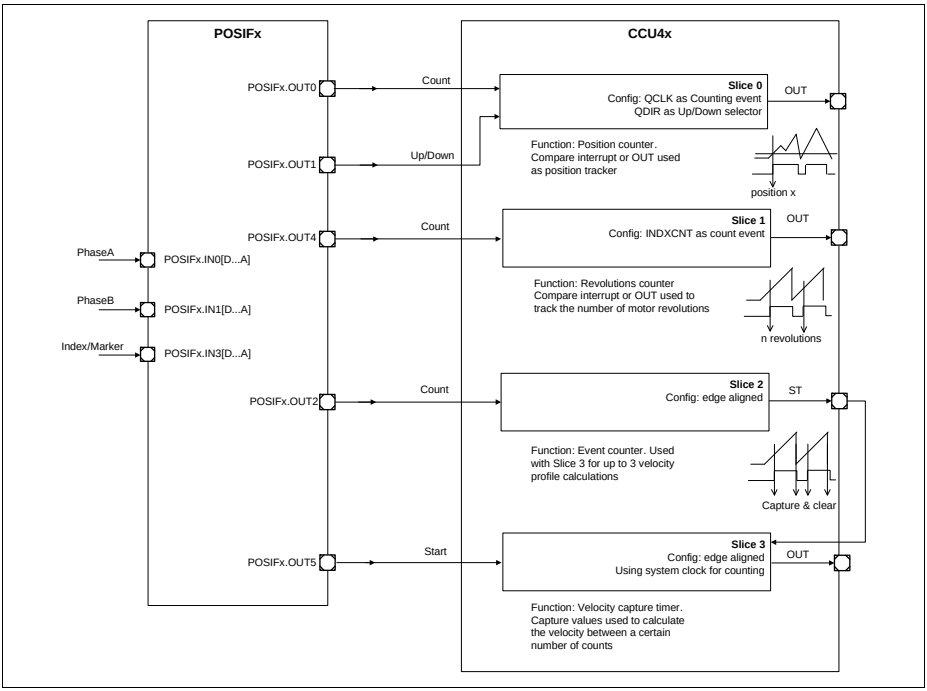


Figure 21-17 Quadrature Decoder Mode usage - profile 1

Quadrature Decoder Usage - Extended Tick Comparison plus Velocity Between N Ticks

The profile demonstrated in **Figure 21-18** enables the usage of 2 compare channels for the position control. This profile is especially useful for multi operations during just one motor revolution.

Slice 0 and Slice 1 are using the POSIFx.OUT0 as counting function and the POSIFxOUT1 as up/down control. The compare channels in each slice are then programmed with different compare values.

Slice 2 and Slice 3 are used in the same manner as profile 1, **Figure 21-18**.

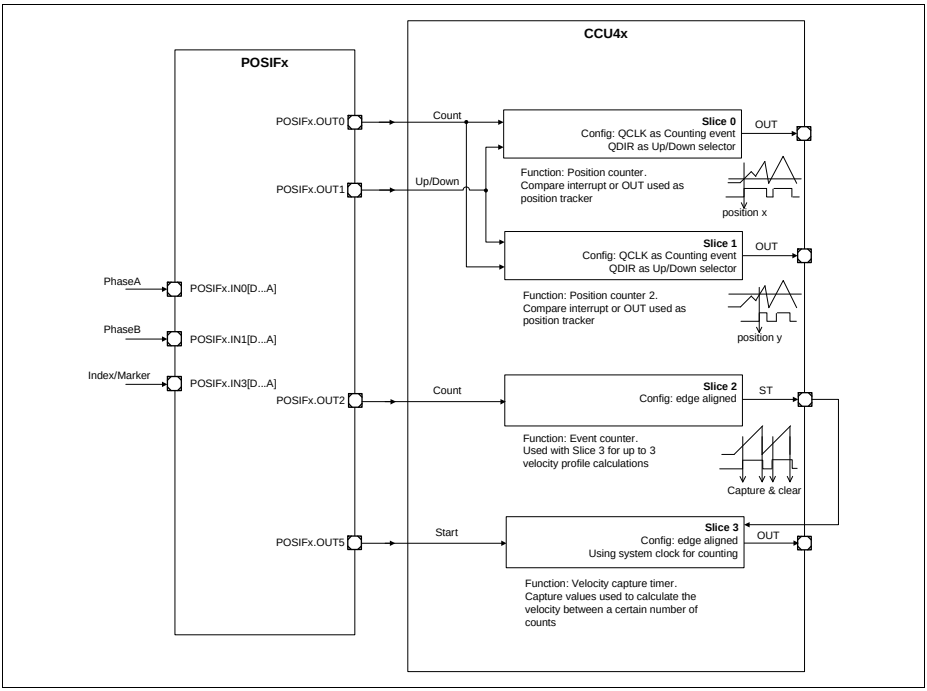


Figure 21-18 Quadrature Decoder Mode usage - profile 2

Quadrature Decoder Usage - Tick and Revolution Comparison with Index Clear plus Velocity Between N Ticks

In some applications it is useful to use the index marker as a clear signal for the position and velocity control. This clear action is linked with the POSIFx.OUT3 pin, that can be programmed to be asserted only at the first index marker or at all maker hits. This pin is then connected to the specific CCU4 slices and used as clear functionality. **Figure 21-19** shows the adaptation of profile 1 with the index marker signal used as clear signal.

The same procedure can be used in profile 2, so that we can have the 2 compare channels plus the velocity measurement and the index used as clear signal.

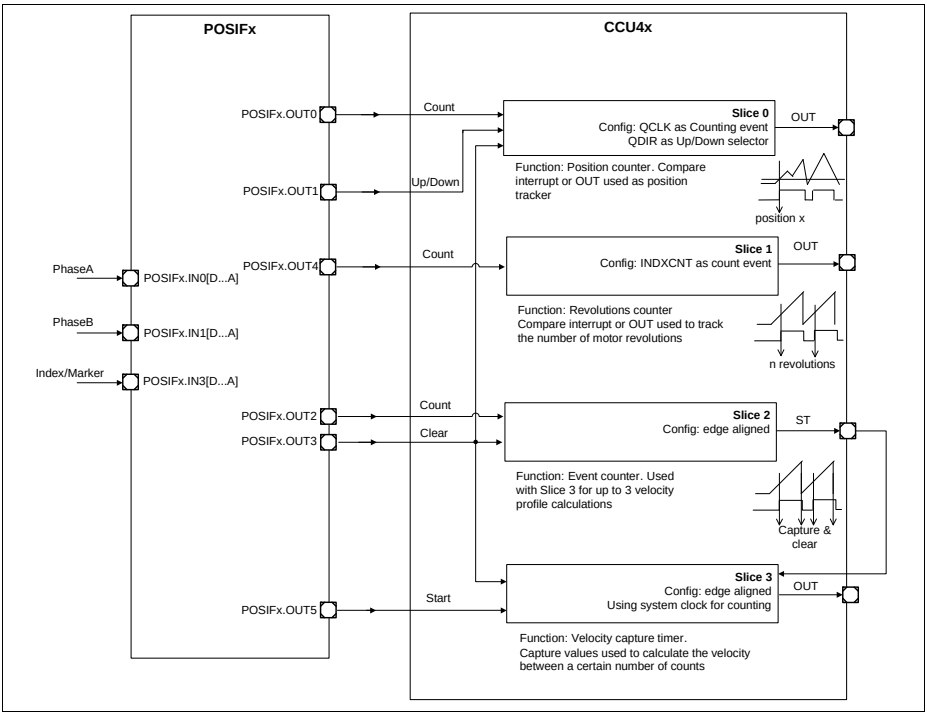


Figure 21-19 Quadrature Decoder Mode usage - profile 3

Quadrature Decoder Usage - Tick Comparison plus Micro Tick Velocity for Slow Rotating Speeds

When the motor rotating speed is slow, it may not be suitable to discard the time between each occurrence of a rotary encoder tick.

In a fast rotating system, the time between ticks is normally discarded and the velocity calculation can be done, by taking into account the number of ticks that have been elapsed since the last ISR. But in a slow rotating system, taking into account the number of ticks may not be enough (because of the associated error), and the software needs also to take into consideration, the time between the last tick and the actual ISR trigger.

Figure 21-20 shows a slow rotating system, where the ISR for the velocity calculation is triggered in a periodic way. In this case, because of the small amount of ticks between each ISR trigger, the software needs to know not only the amount of elapsed ticks but also the elapsed time between the last tick and the ISR occurrence.

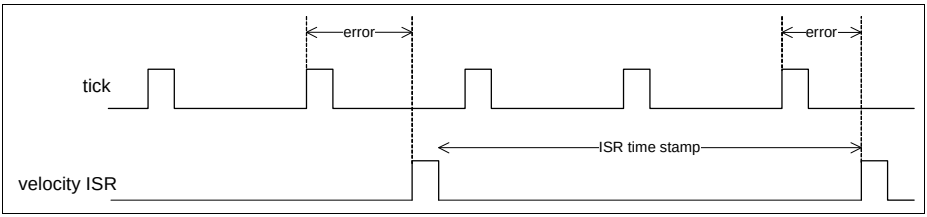


Figure 21-20 Slow rotating system example

It is possible to built a profile with the POSIF and one CCU4 module to perform a control loop that is immune to this slow velocity calculation pitfalls. The resource usage is exemplified on **Figure 21-21**.

One timer slice of a CCU4 module, Slice 0, is used to monitor the current position of the motor shaft.

Three additional timer slices are needed to built the slow velocity calculation loop: Slice 1, Slice 2 and Slice 3.

Timer Slice 1, is counting the number of ticks (PCLK) that are elapsed between each velocity ISR occurrence.

Timer Slice 2, is counting the time between each tick (PCLK) occurrence. This timer slice is cleared and started within every tick and therefore it always keeps the timing info between the last and the current tick.

Timer Slice 3, is controlling the periodicity of the velocity ISR. Every time that a velocity ISR is triggered, Slice 3 also triggers a capture in Slice 2 and a capture & clear in Slice 1.

With this mechanism, every time that the software reads back the captured values from Slice 1 and Slice 2, it can calculate the speed based on:

- the amount of ticks elapsed since the last ISR
- plus the amount of time elapsed between the last tick and the ISR

This control loop offers therefore a very accurate way to calculate the motor speed within systems with low velocity.

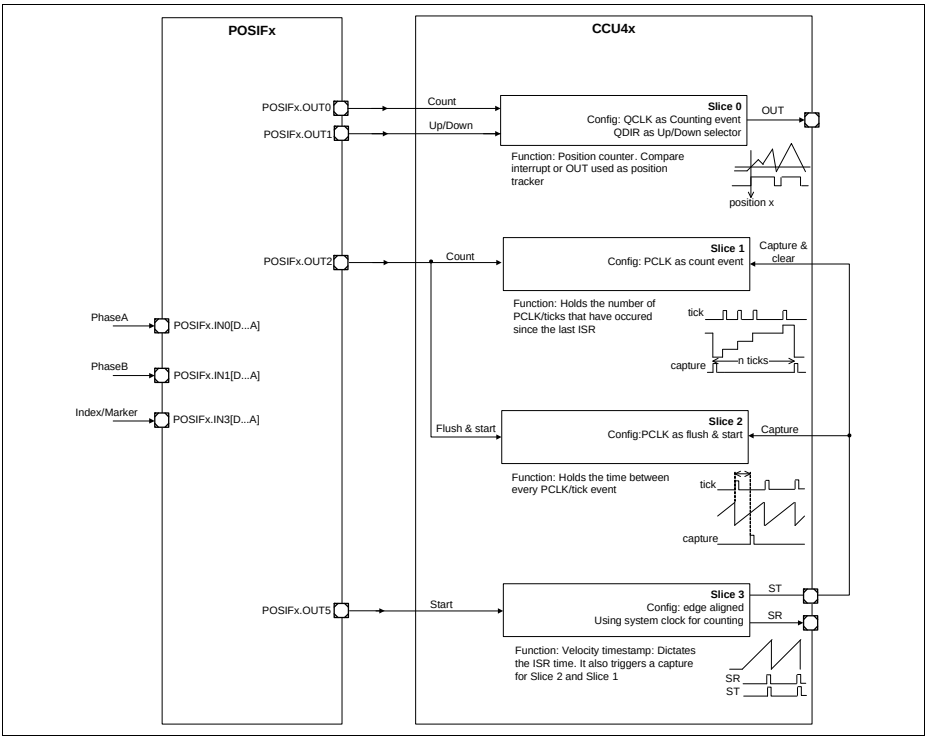


Figure 21-21 Quadrature Decoder Mode usage - profile 4

21.2.7.3 Stand-alone Multi-Channel Mode

The Multi-Channel Mode can be used in the POSIF module without being linked with the Hall Sensor Mode. This is especially useful for performing generic control loops that need to be perfectly synchronized.

The stand-alone Multi-Channel Mode is very generic and depends very much on the control loop specified by the user. A generic profile for the Multi-Channel mode is to use a CCU4/CCU8 module to perform the signal generation and a slice from a CCU4 module linked with an external pin that is used as trigger for updating the Multi-Channel pattern.

On **Figure 21-22**, a slice from a CCU4 unit is used to control the delay between the update of an external signal (e.g. external sensor) and the update of the Multi-Channel Pattern. Notice that this external signal can also be connected to the POSIF module if no timing delay is needed or it can be just controlled by SW, by writing an one into the **MCMS**.MNPS field.

One output can then be chosen from the CCU4/CCU8 unit that is generating the PWM signals, to act as synchronization trigger between the generated signal and the update of the Multi-Channel pattern (CCU4x.PSy in case of CCU4 and CCU8x.PSy in case of CCU8).

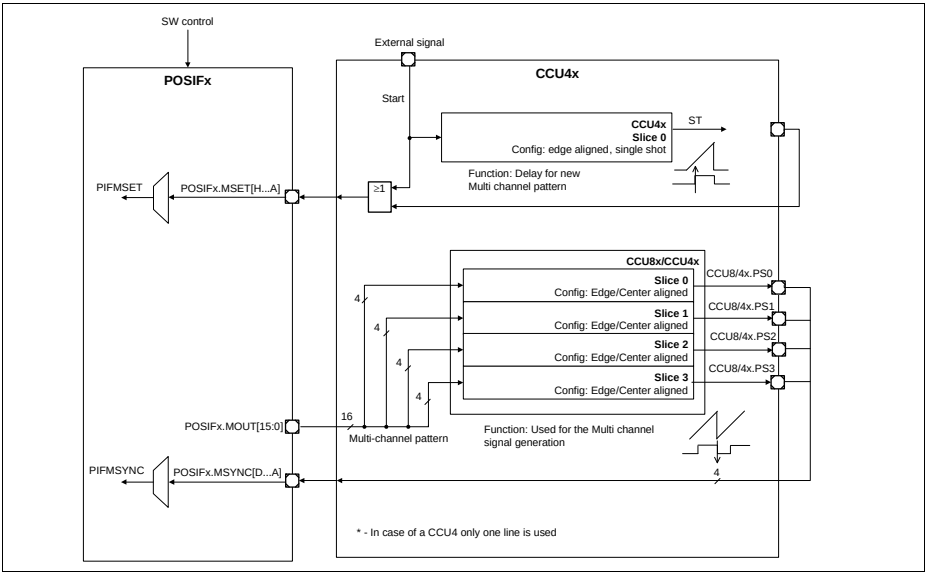


Figure 21-22 Stand-alone Multi-Channel Mode usage

21.3 Service Request Generation

The POSIF has several interrupt sources that are linked to the different operation modes: Hall Sensor Mode, Quadrature Decoder Mode and stand-alone Multi-Channel Mode.

The interrupt flags for the Hall Sensor Mode are described in **Section 21.3.1** while the interrupt flags of the Quadrature Decoder Mode are described in **Section 21.3.2**.

The flags associated with the Multi-Channel function are available in all the modes. This is due to the fact that the Multi-Channel Mode can operate in parallel with the Quadrature Decoder Mode and is needed whenever the Hall Sensor Mode is activated.

Each of the interrupt sources, can be routed to the POSIFx.SR0 or POSIFx.SR1 output, depending on the value programmed in the **PFLGE** register.

21.3.1 Hall Sensor Mode flags

The Hall Sensor Control contains four flags that can be configured to generate an interrupt request pulse, see **Figure 21-23**.

The four interrupt sources are:

- Transition at the Hall Inputs (**PFLG**.HIES)
- occurrence of a correct hall event (**PFLG**.CHES)
- occurrence of a wrong hall event (**PFLG**.WHES)
- shadow transfer of the Multi-Channel pattern (**PFLG**.MSTS)

The last one is triggered every time the Multi-Channel pattern is updated (PIFMST), which means that the POSIFx.MOUT[15:0] output was updated with a new value (see also **Figure 21-6**).

Each of this interrupt sources can be enabled/disabled individually. The SW also has the possibility to set and clear the specific flags by writing a 1_B into the specific fields of **SPFLG** and **RPFLG** registers. By enabling an interrupt source, an interrupt pulse is generated every time that a flag set operation occurs, independently if the flag is already set or not.

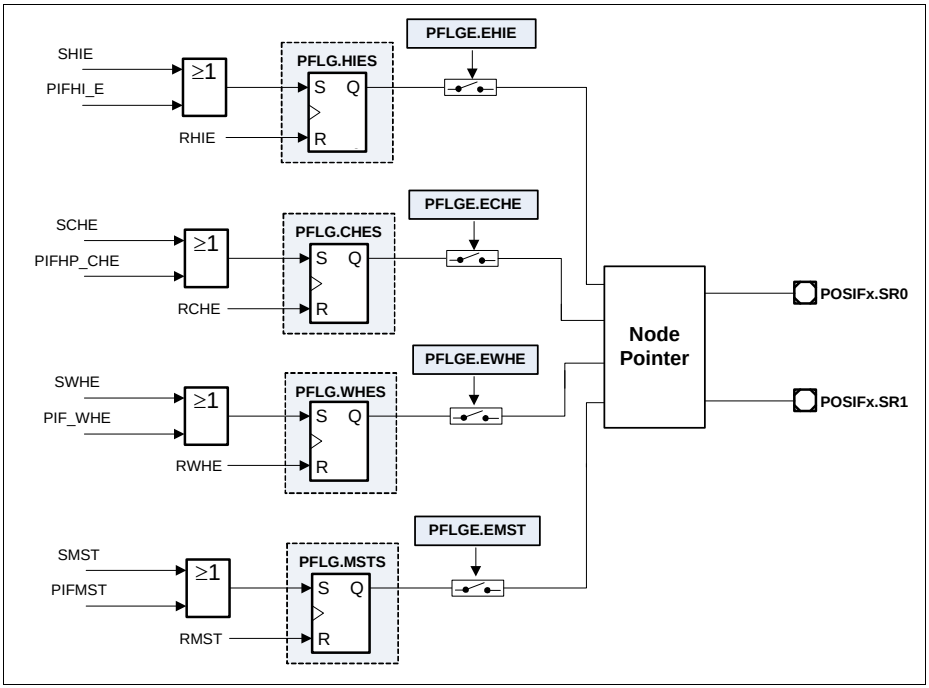


Figure 21-23 Hall Sensor Mode flags

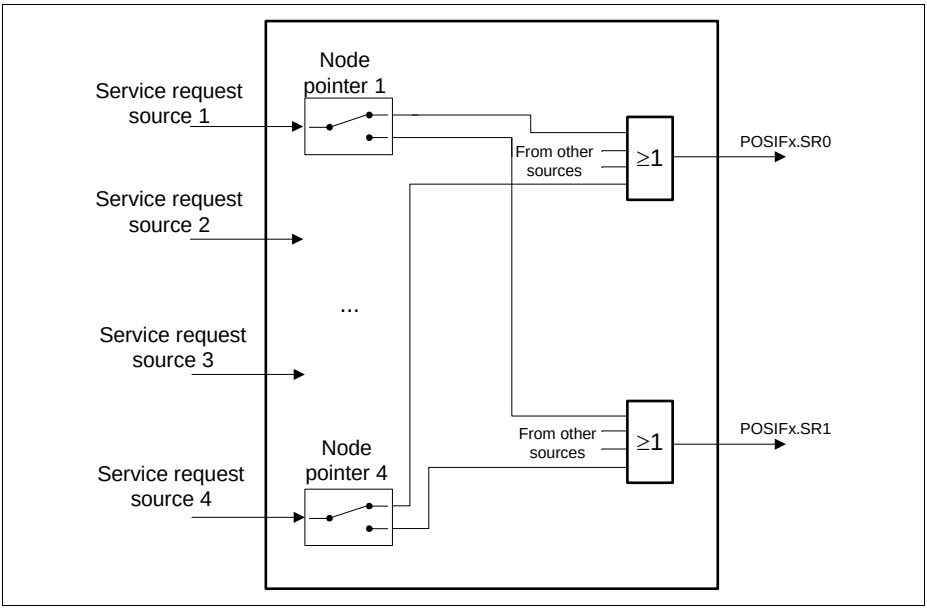


Figure 21-24 Interrupt node pointer overview - hall sensor mode

21.3.2 Quadrature Decoder Flags

The Quadrature Decoder Mode has five flags that can be enabled individually as interrupt sources (besides these five flags, the ones that are linked to the usage of Multi-Channel mode are also available: Multi-Channel Pattern update (**PFLG**.MSTS) and Wrong Hall event (**PFLG**.WHES). This can be useful if one selects the Quadrature Mode and the Multi-Channel stand-alone functionality.

These five flags are:

- Index event detection (**PFLG**.INDXS)
- phase detection error (**PFLG**.ERRS)
- quadrature clock generation (**PFLG**.CNTS)
- period clock generation (**PFLG**.PCLKS)
- direction change (**PFLG**.DIRS)

By enabling an interrupt source, an interrupt pulse is generated every time that a flag set operation occurs, independently if the flag is already set or not.

The index event detection flag is set every time that a index is detected.

The phase error flag is set when one invalid transition on the phase signals is detected, see **Figure 21-11**.

The quadrature clock and period clock flags are generated accordingly with the timings shown in **Figure 21-12**.

The direction change flag is set, every time that an inversion of the motor rotation happens.

Each flag can be set/cleared individually by SW, by writing into the specific field on the registers **SPFLG** and **RPFLG**, see **Figure 21-25**.

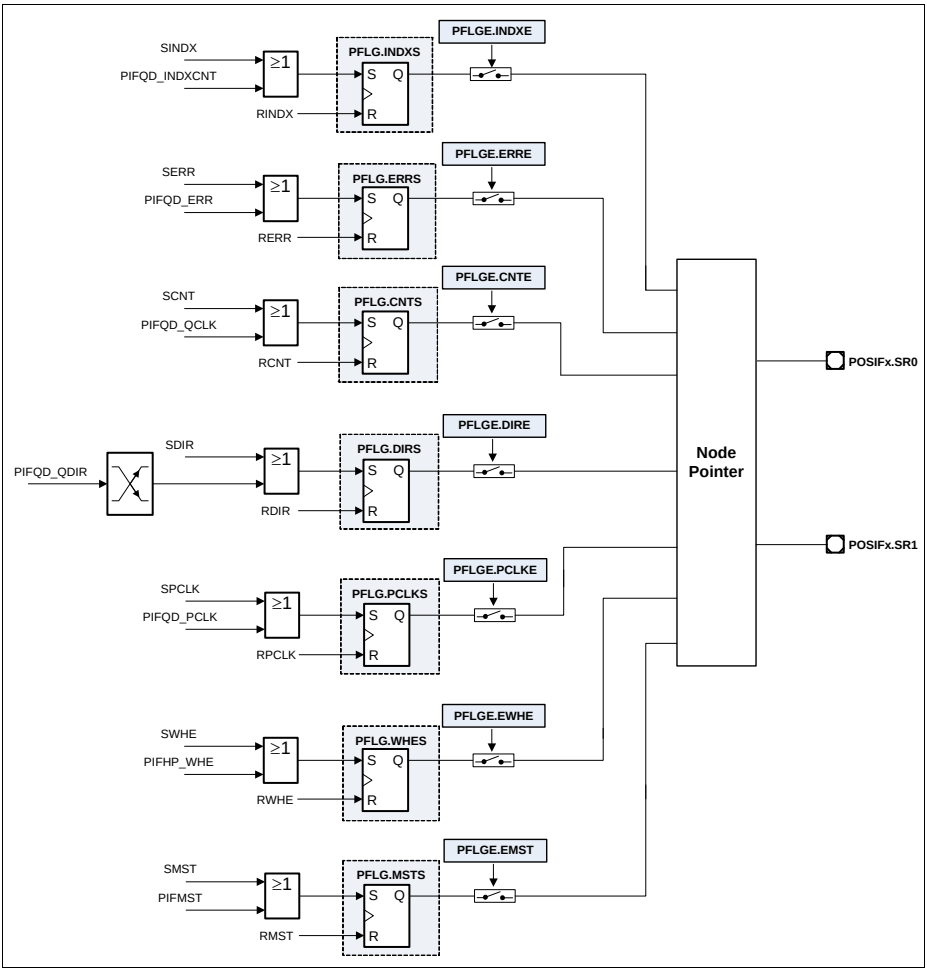


Figure 21-25 Quadrature Decoder flags

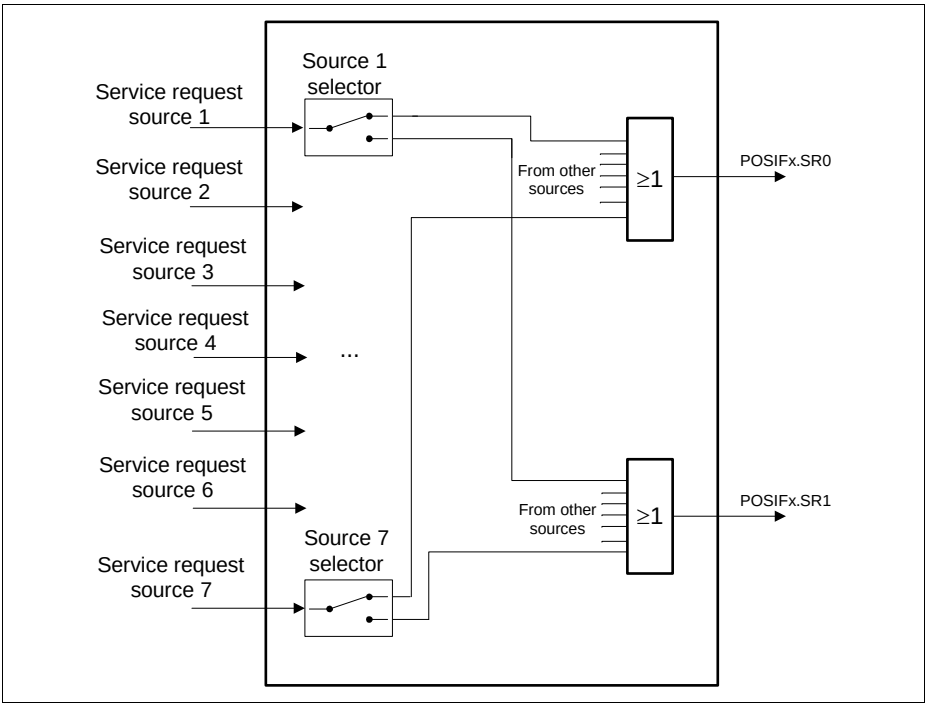


Figure 21-26 Interrupt node pointer overview - quadrature decoder mode

21.4 Debug Behavior

In suspend mode, the entire block is halted. The registers can still be accessed by the CPU (read only). This mode is useful for debugging purposes, e.g., where the current device status should be frozen in order to get a snapshot of the internal values. In suspend mode, all counters are stopped. The suspend mode is non-intrusive concerning the register bits. This means register bits are not modified by hardware when entering or leaving the suspend mode.

Entry into suspend mode can be configured at the kernel level by means of the register **PSUS**.

The module is only functional after the suspend signal becomes inactive.

21.5 Power, Reset and Clock

The following sections describe the operating conditions, characteristics and timing requirements for the POSIF. All the timing information is related to the module clock, f_{posif}.

21.5.1 Clocks

Module Clock

The module clock of the POSIF module is described in the SCU chapter as f_{CCU}.

The bus interface clock of the POSIF module is described in the SCU chapter as f_{PERIPH}.

The module clock for the POSIF is controlled via a specific control bit inside the SCU (System Control Unit), register CLKSET.

It is possible to disable the module clock for the POSIF, via a specific system control bit, nevertheless, there may be a dependency on the f_{posif} through the different POSIF instances or Capture/Compare Units. One should address the SCU Chapter for a complete description of the product clock scheme.

External Signals

The maximum frequency for the external signals, linked with the Hall Sensor and Rotary Encoder inputs can be seen in **Table 21-4**.

Table 21-4 External Hall/Rotary signals operating conditions

Parameter	Symbol	Values			Unit	Note / Test Condition
		Min.	Typ.	Max.		
Frequency	f_{esig}	–	–	$f_{posif}/4$	MHz	
ON time	ton_{esig}	$2T_{posif}$	–	–	ns	
OFF time	$toff_{esig}$	$2T_{posif}$	–	–	ns	

21.5.2 Module Reset

Each POSIF has one reset source. This reset source is handled at system level and it can be generated independently via a system control register, PRSET0 (address SCU chapter for a full description).

After reset release, the complete IP is set to default configuration. The default configuration for each register field is addressed on **Section 21.7**.

21.5.3 Power

The POSIF is inside the power core domain, therefore no special considerations about power up or power down sequences need to be taken. For a explanation about the different power domains, please address the SCU (System Control Unit) chapter.

21.6 Initialization and System Dependencies

21.6.1 Initialization

The initialization sequence for an application that is using the POSIF, should be the following:

1st Step: Apply reset to the POSIF, via the specific SCU register, PRSET0.

2nd Step: Release reset of the POSIF, via the specific SCU register, PRCLR0.

3rd Step: Enable the POSIF clock via the specific SCU register, CLKSET.

4th Step: Configure the POSIF operation mode, **PCONF** register.

5th Step: Configure the POSIF registers linked to the wanted operation mode:

- For Hall Sensor Mode: **HALPS**/**HALP**, **MCSM**/**MCM**
- For Quadrature Decoder Mode: **QDC**
- For stand-alone Multichannel Mode: **MCSM**/**MCM**

5th Step: Apply the pre programmed patterns into the **HALP** and **MCM** registers, by writing 1_B into the **MCMS**.STHR and **MCMS**.STMR fields.

6th Step: Configure the POSIF interrupts/service requests via the **PFLGE** register.

7th Step: Configure the Capture/Compare Units associated with the POSIF operation mode.

8th Step: If the synchronous start function of the POSIF is being used inside the associated Capture/Compare units, then one just needs to set the run bit of the module, by writing a 1_B into **PRUNS**.SRB.

9th Step: If the synchronous start function of the POSIF is not being used inside the associated Capture/Compare units, then start first the POSIF by writing a 1_B into **PRUNS**.SRB. After that, start the associated Capture/Compare Unit timer slices, by addressing the specific bit field inside each timer slice or by using the synchronous start function available on the SCU, CCUCON register.

Note: Due to different startup conditions of the motor system itself (as well SW control loop implementation) it is possible to receive some erroneous interrupt triggers before the SW and HW loop are completely locked. To overcome this, the SW can ignore the interrupts during start up or the interrupt sources can be enabled only after a proper lock between HW and SW has been sensed.

21.6.2 System Dependencies

Each POSIF may have different dependencies regarding module and bus clock frequencies. This dependencies should be addressed in the SCU and System Architecture Chapters.

Dependencies between several peripherals, regarding different clock operating frequencies may also exist. This should be addressed before configuring the connectivity between the POSIF and some other peripheral.

The following topics must be taken into consideration for good POSIF and system operation:

- POSIF module clock must be at maximum two times faster than the module bus interface clock
- Module input triggers for the POSIF must not exceed the module clock frequency (if the triggers are generated internally in the device)
- Module input triggers for the POSIF must not exceed the frequency dictated in **Section 21.5.1**
- Frequency of the POSIF outputs used as triggers/functions on other modules, must be crosschecked on the end point
- Applying and removing POSIF from reset, can cause unwanted operations in other modules. This can occur if the modules are using POSIF outputs as triggers/functions.

21.7 Registers

Registers Overview

The absolute register address is calculated by adding:

Module Base Address + Offset Address

Table 21-5 Registers Address Space

Module	Base Address	End Address	Note
POSIF0	40028000_H	$4002BFFF_H$	

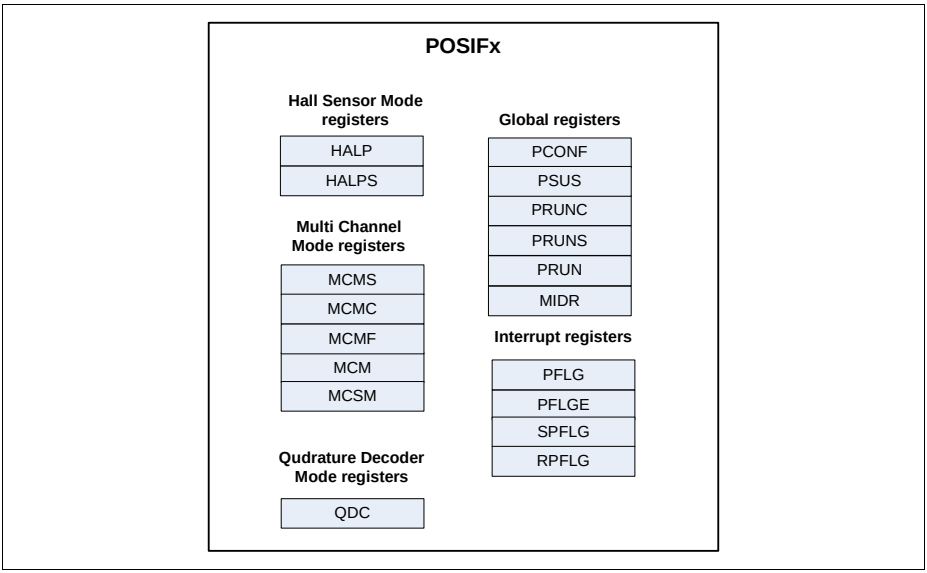


Figure 21-27 POSIF registers overview

Table 21-6 Register Overview of POSIF

Short Name	Description	Offset Addr.[1)]	Access Mode		Description See
			Read	Write	
POSIF Kernel Registers					
PCONF	Global control register	0000_H	U, PV	U, PV	Page 21-38

Table 21-6 Register Overview of POSIF (cont'd)

Short Name	Description	Offset Addr.[1)]	Access Mode		Description See
			Read	Write	
PSUS	Suspend Configuration	0004_H	U, PV	U, PV	Page 21-42
PRUNS	POSIF run bit set	0008_H	U, PV	U, PV	Page 21-43
PRUNC	POSIF run bit clear	$000C_H$	U, PV	U, PV	Page 21-43
PRUN	POSIF run bit status	0010_H	U, PV	BE	Page 21-44
PDBG	Debug Design Register	0100_H	U, PV	BE	Page 21-45
MIDR	Module Identification register	0020_H	U, PV	BE	Page 21-46
Hall Sensor Mode Registers					
HALP	Hall Current and Expected patterns	0030_H	U, PV	BE	Page 21-47
HALPS	Hall Current and Expected shadow patterns	0034_H	U, PV	U, PV	Page 21-48
Multi-Channel Mode Register					
MCM	Multi-Channel Mode Pattern	0040_H	U, PV	BE	Page 21-49
MCSM	Multi-Channel Mode shadow Pattern	0044_H	U, PV	U, PV	Page 21-50
MCMS	Multi-Channel Mode Control set	0048_H	U, PV	U, PV	Page 21-50
MCMC	Multi-Channel Mode Control clear	$004C_H$	U, PV	U, PV	Page 21-51
MCMF	Multi-Channel Mode flag status	0050_H	U, PV	BE	Page 21-52
Quadrature Decoder Mode Register					
QDC	Quadrature Decoder Configuration	0060_H	U, PV	U, PV	Page 21-53
Interrupt Registers					
PFLG	POSIF interrupt status	0070_H	U, PV	BE	Page 21-54
PFLGE	POSIF interrupt enable	0074_H	U, PV	U, PV	Page 21-56

Table 21-6 Register Overview of POSIF (cont'd)

Short Name	Description	Offset Addr.[1)]	Access Mode		Description See
			Read	Write	
SPFLG	Interrupt set register	0078_H	U, PV	U, PV	Page 21-59
RPFLG	Interrupt clear register	$007C_H$	U, PV	U, PV	Page 21-60

1) The absolute register address is calculated as follows:
Module Base Address + Offset Address (shown in this column)

21.7.1 Global registers

PCONF

The register contains the global configuration for the POSIF operation: operation mode, input selection, filter configuration.

PCONF
POSIF configuration **(0000_H)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0		LPC		EWIL	EWIE	EWIS		MSYNS		MSES	MSETS			SPES	DSEL
r		rw		rw	rw	rw		rw		rw	rw			rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		INSEL2		INSEL1		INSEL0		0		MCUE	HIDG	0	QDCM	FSEL	
r		rw		rw		rw		r		rw	rw	r	rw	rw	

Field	Bits	Type	Description
FSEL	[1:0]	rw	**Function Selector** 00_B Hall Sensor Mode enabled 01_B Quadrature Decoder Mode enabled 10_B stand-alone Multi-Channel Mode enabled 11_B Quadrature Decoder and stand-alone Multi-Channel Mode enabled

Field	Bits	Type	Description
QDCM	2	rw	**Position Decoder Mode selection** This field selects if the Position Decoder block is in Quadrature Mode or Direction Count Mode. In Quadrature mode, the position encoder is providing the phase signals, while in Direction Count Mode is providing a clock and a direction signal. 0_B Position encoder is in Quadrature Mode 1_B Position encoder is in Direction Count Mode.
HIDG	4	rw	**Idle generation enable** Setting this field to 1_B disables the generation of the IDLE signal that forces a clear on the Multi-Channel pattern and run bit.
MCUE	5	rw	**Multi-Channel Pattern SW update enable** 0_B Multi-Channel pattern update is controlled via HW 1_B Multi-Channel pattern update is controlled via SW
INSEL0	[9:8]	rw	**PhaseA/Hal input 1 selector** This fields selects which input is used for the Phase A or Hall input 1 function (dependent if the module is set for Quadrature Decoder or Hall Sensor Mode): 00_B POSIFx.IN0A 01_B POSIFx.IN0B 10_B POSIFx.IN0C 11_B POSIFx.IN0D
INSEL1	[11:10]	rw	**PhaseB/Hall input 2 selector** This fields selects which input is used for the Phase B or Hall input 2 function (dependent if the module is set for Quadrature Decoder or Hall Sensor Mode): 00_B POSIFx.IN1A 01_B POSIFx.IN1B 10_B POSIFx.IN1C 11_B POSIFx.IN1D

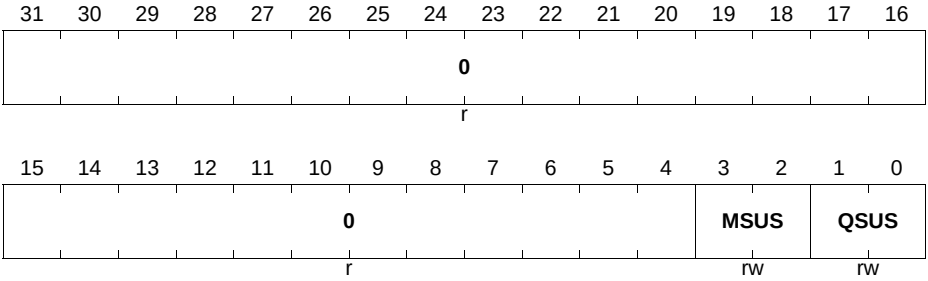
Field	Bits	Type	Description
INSEL2	[13:12]	rw	**Index/Hall input 3 selector** This fields selects which input is used for the Index or Hall input 3 function (dependent if the module is set for Quadrature Decoder or Hall Sensor Mode): 00_B POSIFx.IN2A 01_B POSIFx.IN2B 10_B POSIFx.IN2C 11_B POSIFx.IN2D
DSEL	16	rw	**Delay Pin selector** This field selects which input is used to trigger the end of the delay between the detection of an edge in the Hall inputs and the actual sample of the Hall inputs. 0_B POSIFx.HSDA 1_B POSIFx.HSDB
SPES	17	rw	**Edge selector for the sampling trigger** This field selects which edge is used of the selected POSIFx.HSD[B...A] signal to trigger a sample of the Hall inputs. 0_B Rising edge 1_B Falling edge
MSETS	[20:18]	rw	**Pattern update signal select** Selects the input signal that is used to enable a Multi-Channel pattern update. 000_B POSIFx.MSETA 001_B POSIFx.MSETB 010_B POSIFx.MSETC 011_B POSIFx.MSETD 100_B POSIFx.MSETE 101_B POSIFx.MSETF 110_B POSIFx.MSETG 111_B POSIFx.MSETH
MSES	21	rw	**Multi-Channel pattern update trigger edge** 0_B The signal used to enable a pattern update is active on the rising edge 1_B The signal used to enable a pattern update is active on the falling edge

Field	Bits	Type	Description
MSYNS	[23:22]	rw	**PWM synchronization signal selector** This fields selects which input is used as trigger for Multi-Channel pattern update (synchronization with the PWM signal). 00_B POSIFx.MSYNCA 01_B POSIFx.MSYNCB 10_B POSIFx.MSYNCC 11_B POSIFx.MSYNCD
EWIS	[25:24]	rw	**Wrong Hall Event selection** 00_B POSIFx.EWHEA 01_B POSIFx.EWHEB 10_B POSIFx.EWHEC 11_B POSIFx.EWHED
EWIE	26	rw	**External Wrong Hall Event enable** 0_B External wrong hall event emulation signal, POSIFx.EWHE[D...A], is disabled 1_B External wrong hall event emulation signal, POSIFx.EWHE[D...A], is enabled.
EWIL	27	rw	**External Wrong Hall Event active level** 0_B POSIFx.EWHE[D...A] signal is active HIGH 1_B POSIFx.EWHE[D...A] signal is active LOW
LPC	[30:28]	rw	**Low Pass Filters Configuration** 000_B Low pass filter disabled 001_B Low pass of 1 clock cycle 010_B Low pass of 2 clock cycles 011_B Low pass of 4 clock cycles 100_B Low pass of 8 clock cycles 101_B Low pass of 16 clock cycles 110_B Low pass of 32 clock cycles 111_B Low pass of 64 clock cycles
0	3, [7:6], [15:14], 31	r	**Reserved** Read always returns 0

PSUS

The register contains the suspend configuration for the POSIF.

PSUS
POSIF Suspend Config **(0004$_H$)** **Reset Value: 00000000$_H$**



Field	Bits	Type	Description
QSUS	[1:0]	rw	**Quadrature Mode Suspend Config** This field controls the entering in suspend for the quadrature decoder mode. 00_B Suspend request ignored 01_B Stop immediately 10_B Suspend in the next index occurrence 11_B Suspend in the next phase (PhaseA or PhaseB) occurrence
MSUS	[3:2]	rw	**Multi-Channel Mode Suspend Config** This field controls the entering in suspend for the Multi-Channel mode. The Hall sensor mode is also covered by this configuration. 00_B Suspend request ignored 01_B Stop immediately. Multi-Channel pattern is not set to the reset value. 10_B Stop immediately. Multi-Channel pattern is set to the reset value. 11_B Suspend with the synchronization of the PWM signal. Multi-Channel pattern is set to the reset value at the same time of the synchronization.
0	[31:4]	r	**Reserved** Read always returns 0

PRUNS

Via this register it is possible to set the run bit of the module.

PRUNS

POSIF Run Bit Set **(0008_H)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0															SRB
r															w

Field	Bits	Type	Description
SRB	0	w	**Set Run bit** Writing an 1_B into this bit sets the run bit of the module. Read always returns 0.
0	[31:1]	r	**Reserved** Read always returns 0

PRUNC

Via this register it is possible to clear the run bit and the internal state machines of the module.

PRUNC

POSIF Run Bit Clear **($000C_H$)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0														CSM	CRB
r														w	w

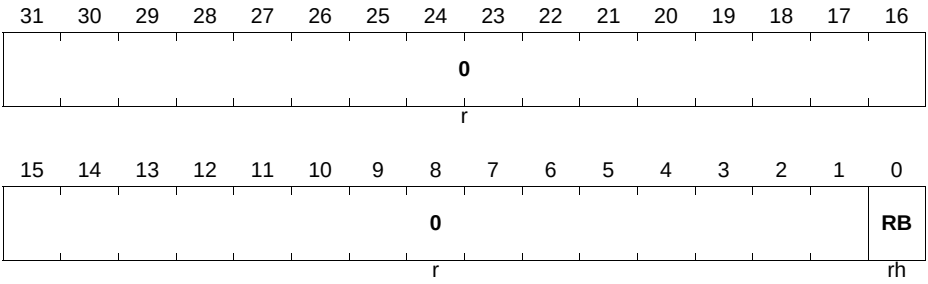
Field	Bits	Type	Description
CRB	0	w	**Clear Run bit** Writing an 1_B into this bit clears the run bit of the module. The module is stopped. Read always returns 0.
CSM	1	w	**Clear Current internal status** Writing an 1_B into this bit resets the state machine of the quadrature decoder and the current status of the Hall sensor and Multi-Channel mode registers. The flags and static configuration bit fields are not cleared. Read always returns 0.
0	[31:2]	r	**Reserved** Read always returns 0

PRUN

The register contains the run bit status of the POSIF.

PRUN

POSIF Run Bit Status **(0010_H)** **Reset Value: 00000000_H**



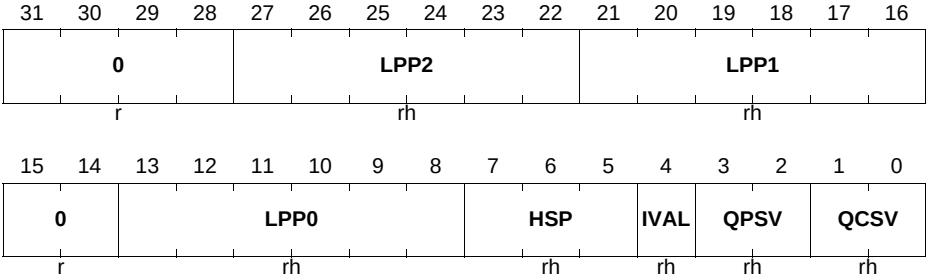
Field	Bits	Type	Description
RB	0	rh	**Run Bit** This field indicates if the module is in running or IDLE state. 0_B IDLE 1_B Running

Field	Bits	Type	Description
0	[31:1]	r	**Reserved** Read always returns 0

PDBG

Debug register for current state of the POSIF state machines and Hall Sampled values.

PDBG
POSIF Debug register **(0100_H)** **Reset Value: 00000000_H**

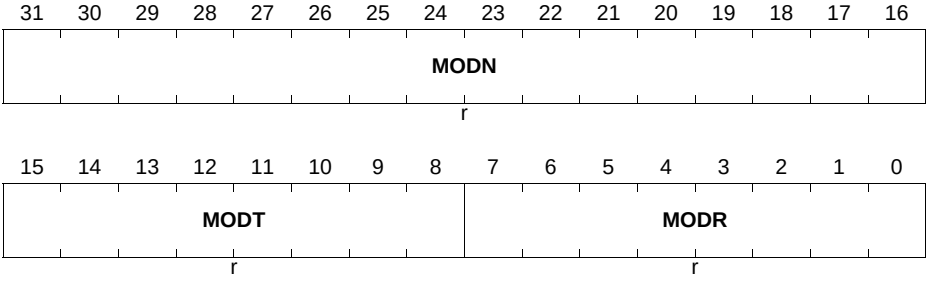


Field	Bits	Type	Description
QCSV	[1:0]	rh	**Quadrature Decoder Current state** QCSV[0] - Phase A QCSV[1] - Phase B
QPSV	[3:2]	rh	**Quadrature Decoder Previous state** QPSV[0] - Phase A QPSV[1] - Phase B
IVAL	4	rh	**Current Index Value**
HSP	[7:5]	rh	**Hall Current Sampled Pattern** HSP[0] - Hall Input 1 HSP[1] - Hall Input 2 HSP[2] - Hall Input 3
LPP0	[13:8]	rh	**Actual count of the Low Pass Filter for POSI0**
LPP1	[21:16]	rh	**Actual count of the Low Pass Filter for POSI1**
LPP2	[27:22]	rh	**Actual count of the Low Pass Filter for POSI2**
0	[31:28], [15:14]	r	**Reserved** A read always returns 0.

MIDR

This register contains the module identification number.

MIDR
Module Identification register (0020$_H$) Reset Value: 00A8C0XX$_H$



Field	Bits	Type	Description
MODR	[7:0]	r	**Module Revision** This bit field indicates the revision number of the module implementation (depending on the design step).
MODT	[15:8]	r	**Module Type**
MODN	[31:16]	r	**Module Number**

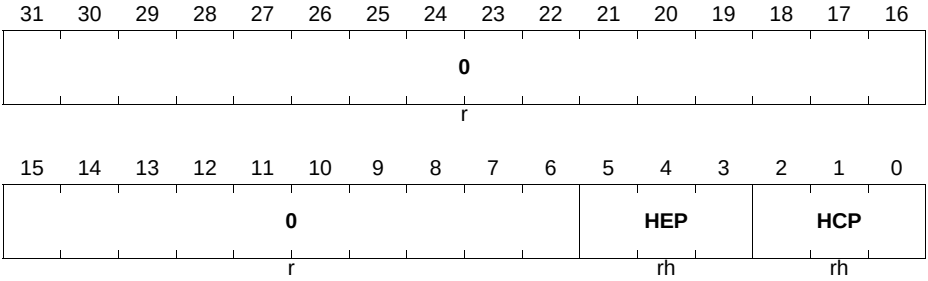
21.7.2 Hall Sensor Mode Registers

HALP

The register contains the values for the Hall Expected Pattern and Hall Current Pattern.

HALP

Hall Sensor Patterns **(0030_H)** **Reset Value: 00000000_H**

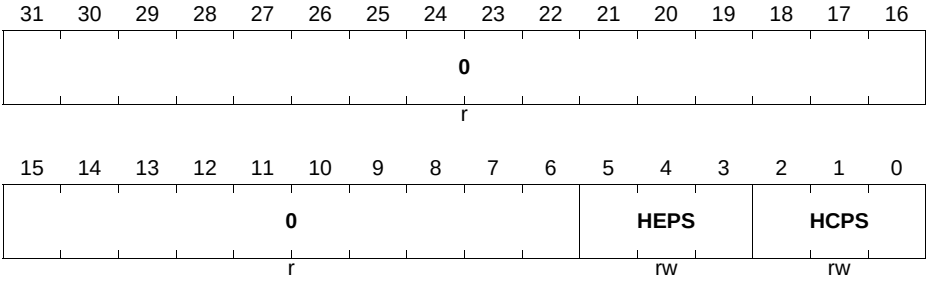


Field	Bits	Type	Description
HCP	[2:0]	rh	**Hall Current Pattern** This field contains the Hall Current pattern. This field is updated with the **HALPS**.HCPS value every time that a correct hall event occurs. HCP[0] - Hall Input 1 HCP[1] - Hall Input 2 HCP[2] - Hall Input 3
HEP	[5:3]	rh	**Hall Expected Pattern** This field contains the Hall Expected pattern. This field is updated with the **HALPS**.HEPS values every time that a correct hall event occurs. HEP[0] - Hall Input 1 HEP[1] - Hall Input 2 HEP[2] - Hall Input 3
0	[31:6]	r	**Reserved** Read always returns 0

HALPS

The register contains the values that are going to be loaded into the **HALP** register when the next correct hall event occurs.

HALPS
Hall Sensor Shadow Patterns (0034_H) Reset Value: 00000000_H



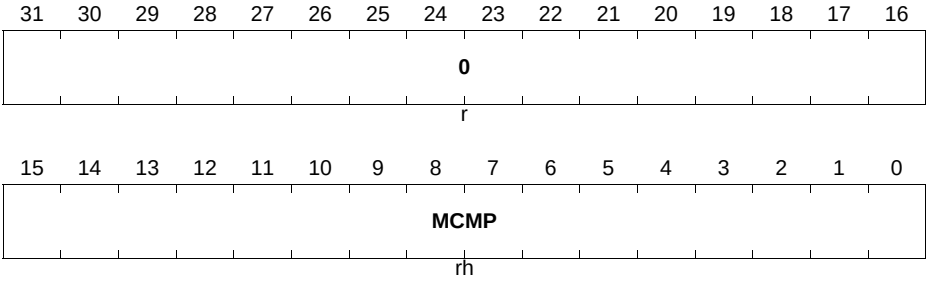
Field	Bits	Type	Description
HCPS	[2:0]	rw	**Shadow Hall Current Pattern** This field contains the next Hall Current pattern value. This field is set on the **HALP**.HCP field every time that a correct hall event occurs. HCPS[0] - Hall Input 1 HCPS[1] - Hall Input 2 HCPS[2] - Hall Input 3
HEPS	[5:3]	rw	**Shadow Hall expected Pattern** This field contains the next Hall Expected pattern. This field is set on the **HALP**.HEP field every time that a correct hall event occurs. HEPS[0] - Hall Input 1 HEPS[1] - Hall Input 2 HEPS[2] - Hall Input 3
0	[31:6]	r	**Reserved** Read always returns 0

21.7.3 Multi-Channel Mode Registers

MCM

The register contains the value of the Multi-Channel pattern that is applied to the outputs POSIFx.OUT[15:0].

MCM
Multi-Channel Pattern **(0040_H)** **Reset Value: 00000000_H**

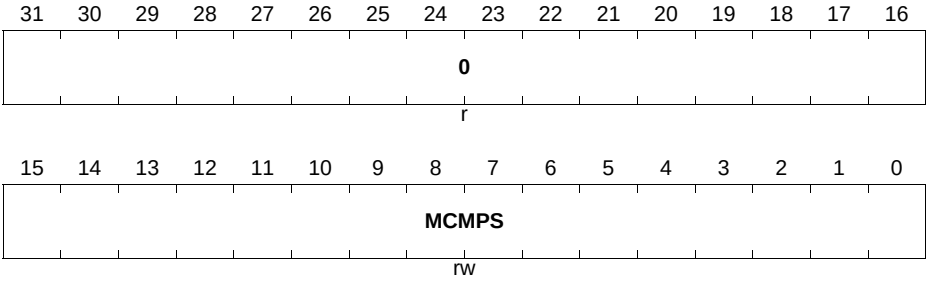


Field	Bits	Type	Description
MCMP	[15:0]	rh	**Multi-Channel Pattern** This field contains the Multi-Channel Pattern that is going to be applied to the Multi-Channel outputs, POSIFx.MOUT[15:0]. This field is updated with the value of the **MCSM**.MCMPS every time that a Multi-Channel pattern update is triggered.
0	[31:16]	r	**Reserved** Read always returns 0

MCSM

The register contains the value that is going to be loaded into the **MCM** register when the next Multi-Channel update trigger occurs.

MCSM
Multi-Channel Shadow Pattern **(0044_H)** **Reset Value: 00000000_H**

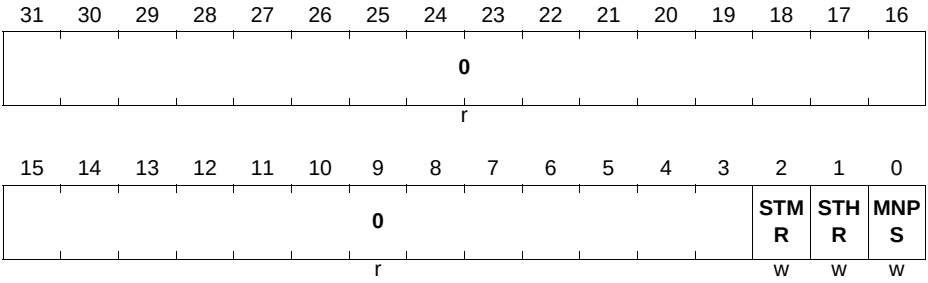


Field	Bits	Type	Description
MCMPS	[15:0]	rw	**Shadow Multi-Channel Pattern** This field contains the next Multi-Channel Pattern. Every time that a Multi-Channel pattern transfer is triggered, this value is passed into the field **MCM**.MCMP.
0	[31:16]	r	**Reserved** Read always returns 0

MCMS

Through this register it is possible to request a Multi-Channel pattern update. It is also possible through this register to request an immediate update of the Multi-Channel and Hall Sensor patterns without waiting for the hardware trigger.

MCMS
Multi-Channel Pattern Control set **(0048_H)** **Reset Value: 00000000_H**



Field	Bits	Type	Description
MNPS	0	w	**Multi-Channel Pattern Update Enable Set** Writing a 1_B into this field enables the Multi-Channel pattern update (sets the **MCMF**.MSS bit). The update is not done immediately due to the fact that the trigger that synchronizes the update with the PWM is still needed. A read always returns 0.
STHR	1	w	**Hall Pattern Shadow Transfer Request** Writing a 1_B into this field leads to an immediate update of the fields **HALP**.HCP and **HALP**.HEP. A read always returns 0.
STMR	2	w	**Multi-Channel Shadow Transfer Request** Writing a 1_B into this field leads to an immediate update of the field **MCM**.MCMP. A read always returns 0.
0	[31:3]	r	**Reserved** Read always returns 0

MCMC

Through this register is possible to cancel a Multi-Channel pattern update and to clear the Multi-Channel pattern to the default value.

MCMC
Multi-Channel Pattern Control clear ($004C_H$) **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

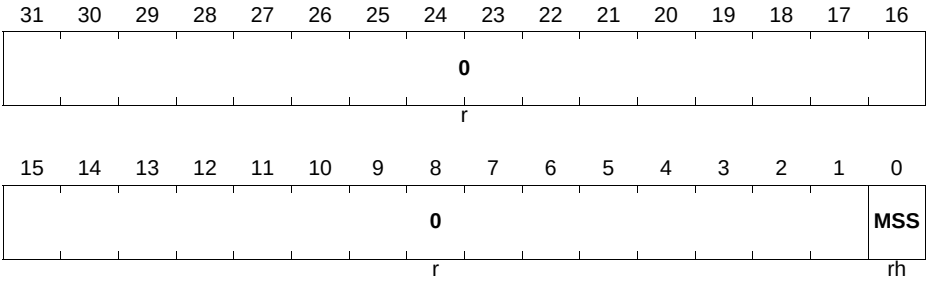
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0														MPC	MNPC
r														w	w

Field	Bits	Type	Description
MNPC	0	w	**Multi-Channel Pattern Update Enable Clear** Writing a 1_B into this field clears the **MCMF**.MSS bit. A read always returns 0.
MPC	1	w	**Multi-Channel Pattern clear** Writing a 1_B into this field clears the Multi-Channel Pattern value to 0000_H. A read always returns 0.
0	[31:2]	r	**Reserved** Read always returns 0

MCMF

The register contains the status of the Multi-Channel update request.

MCMF
Multi-Channel Pattern Control flag (0050_H) **Reset Value: 00000000_H**



Field	Bits	Type	Description
MSS	0	rh	**Multi-Channel Pattern update status** This field indicates if the Multi-Channel pattern is ready to be updated or not. When this field is set, the Multi-Channel pattern is updated when the triggering signal, selected from the POSIFx.MSYNC[D...A], becomes active. 0_B Update of the Multi-Channel pattern is set 1_B Update of the Multi-Channel pattern is not set
0	[31:1]	r	**Reserved** Read always returns 0

21.7.4 Quadrature Decoder Registers

QDC

The register contains the configuration for the operation of the Quadrature Decoder Mode.

QDC
Quadrature Decoder Control **(0060_H)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0							DVAL	0		ICM		0	PHS	PBLS	PALS
r							rh	r		rw		r	rw	rw	rw

Field	Bits	Type	Description
PALS	0	rw	**Phase A Level selector** 0_B Phase A is active HIGH 1_B Phase A is active LOW
PBLS	1	rw	**Phase B Level selector** 0_B Phase B is active HIGH 1_B Phase B is active LOW
PHS	2	rw	**Phase signals swap** 0_B Phase A is the leading signal for clockwise rotation 1_B Phase B is the leading signal for clockwise rotation

Field	Bits	Type	Description
ICM	[5:4]	rw	**Index Marker generations control** This field controls the generation of the index marker that is linked to the output pin POSIFx.OUT3. 00_B No index marker generation on POSIFx.OUT3 01_B Only first index occurrence generated on POSIFx.OUT3 10_B All index occurrences generated on POSIFx.OUT3 11_B Reserved
DVAL	8	rh	**Current rotation direction** 0_B Counterclockwise rotation 1_B Clockwise rotation
0	3, [7:6], [31:9]	r	**Reserved** Read always returns 0

21.7.5 Interrupt Registers

PFLG

The register contains the status of all the interrupt flags of the module.

PFLG
POSIF Interrupt Flags **(0070_H)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0			PCLKS	DIRS	CNTS	ERRS	INDXS	0			MSTS	0	HIES	WHES	CHES
r			rh	rh	rh	rh	rh	r			rh	r	rh	rh	rh

Field	Bits	Type	Description
CHES	0	rh	**Correct Hall Event Status** 0_B Correct Hall Event not detected 1_B Correct Hall Event detected

Field	Bits	Type	Description
WHES	1	rh	**Wrong Hall Event Status** 0_B Wrong Hall Event not detected 1_B Wrong Hall Event detected
HIES	2	rh	**Hall Inputs Update Status** 0_B Transition on the Hall Inputs not detected 1_B Transition on the Hall Inputs detected
MSTS	4	rh	**Multi-Channel pattern shadow transfer status** 0_B Shadow transfer not done 1_B Shadow transfer done
INDXS	8	rh	**Quadrature Index Status** 0_B Index event not detected 1_B Index event detected
ERRS	9	rh	**Quadrature Phase Error Status** 0_B Phase Error event not detected 1_B Phase Error event detected
CNTS	10	rh	**Quadrature CLK Status** 0_B Quadrature clock not generated 1_B Quadrature clock generated
DIRS	11	rh	**Quadrature Direction Change** 0_B Change on direction not detected 1_B Change on direction detected
PCLKS	12	rh	**Quadrature Period Clk Status** 0_B Period clock not generated 1_B Period clock generated
0	3, [7:5], [31:13]	r	**Reserved** Read always returns 0

PFLGE

Through this register it is possible to enable or disable each of the available interrupt sources. It is also possible to select to which service request line an interrupt is forward.

PFLGE
POSIF Interrupt Enable **(0074_H)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0			PCLSEL	DIRSEL	CNTSEL	ERRSEL	INDSEL	0			MSTSEL	0	HIESEL	WHESEL	CHESEL
r			rw	rw	rw	rw	rw	r			rw	r	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0			EPCLK	EDIR	ECNT	EERR	EINDX	0			EMST	0	EHIE	EWHE	ECHE
r			rw	rw	rw	rw	rw	r			rw	r	rw	rw	rw

Field	Bits	Type	Description
ECHE	0	rw	**Correct Hall Event Enable** 0_B Correct Hall Event interrupt disabled 1_B Correct Hall Event interrupt enabled
EWHE	1	rw	**Wrong Hall Event Enable** 0_B Wrong Hall Event interrupt disabled 1_B Wrong Hall Event interrupt enabled
EHIE	2	rw	**Hall Input Update Enable** 0_B Update of the Hall Inputs interrupt is disabled 1_B Update of the Hall Inputs interrupt is enabled
EMST	4	rw	**Multi-Channel pattern shadow transfer enable** 0_B Shadow transfer event interrupt disabled 1_B Shadow transfer event interrupt enabled
EINDX	8	rw	**Quadrature Index Event Enable** 0_B Index event interrupt disabled 1_B Index event interrupt enabled
EERR	9	rw	**Quadrature Phase Error Enable** 0_B Phase error event interrupt disabled 1_B Phase error event interrupt enabled
ECNT	10	rw	**Quadrature CLK interrupt Enable** 0_B Quadrature CLK event interrupt disabled 1_B Quadrature CLK event interrupt enabled
EDIR	11	rw	**Quadrature direction change interrupt Enable** 0_B Direction change event interrupt disabled 1_B Direction change event interrupt enabled

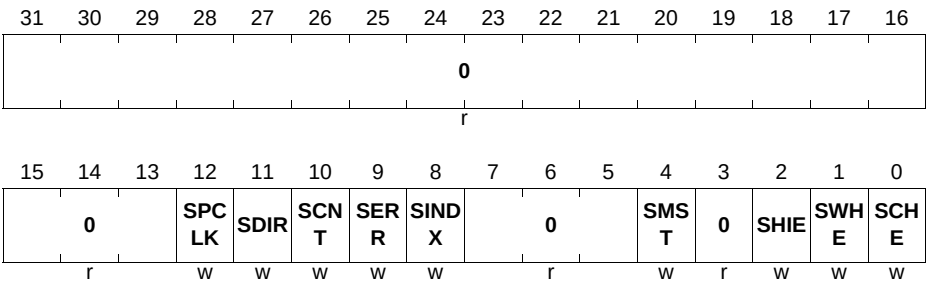
Field	Bits	Type	Description
EPCLK	12	rw	**Quadrature Period CLK interrupt Enable** 0_B Quadrature Period CLK event interrupt disabled 1_B Quadrature Period CLK event interrupt enabled
CHESEL	16	rw	**Correct Hall Event Service Request Selector** 0_B Correct Hall Event interrupt forward to POSIFx.SR0 1_B Correct Hall Event interrupt forward to POSIFx.SR1
WHESEL	17	rw	**Wrong Hall Event Service Request Selector** 0_B Wrong Hall Event interrupt forward to POSIFx.SR0 1_B Wrong Hall Event interrupt forward to POSIFx.SR1
HIESEL	18	rw	**Hall Inputs Update Event Service Request Selector** 0_B Hall Inputs Update Event interrupt forward to POSIFx.SR0 1_B Hall Inputs Update Event interrupt forward to POSIFx.SR1
MSTSEL	20	rw	**Multi-Channel pattern Update Event Service Request Selector** 0_B Multi-Channel pattern Update Event interrupt forward to POSIFx.SR0 1_B Multi-Channel pattern Update Event interrupt forward to POSIFx.SR1
INDSEL	24	rw	**Quadrature Index Event Service Request Selector** 0_B Quadrature Index Event interrupt forward to POSIFx.SR0 1_B Quadrature Index Event interrupt forward to POSIFx.SR1
ERRSEL	25	rw	**Quadrature Phase Error Event Service Request Selector** 0_B Quadrature Phase error Event interrupt forward to POSIFx.SR0 1_B Quadrature Phase error Event interrupt forward to POSIFx.SR1

Field	Bits	Type	Description
CNTSEL	26	rw	**Quadrature Clock Event Service Request Selector** 0_B Quadrature Clock Event interrupt forward to POSIFx.SR0 1_B Quadrature Clock Event interrupt forward to POSIFx.SR1
DIRSEL	27	rw	**Quadrature Direction Update Event Service Request Selector** 0_B Quadrature Direction Update Event interrupt forward to POSIFx.SR0 1_B Quadrature Direction Update Event interrupt forward to POSIFx.SR1
PCLSEL	28	rw	**Quadrature Period clock Event Service Request Selector** 0_B Quadrature Period clock Event interrupt forward to POSIFx.SR0 1_B Quadrature Period clock Event interrupt forward to POSIFx.SR1
0	3, [7:5], [15:13], 19, [23:21], [31:29]	r	**Reserved** Read always returns 0

SPFLG

Through this register it is possible for the SW to set a specific interrupt status flag.

SPFLG
POSIF Interrupt Set **(0078_H)** **Reset Value: 00000000_H**



Field	Bits	Type	Description
SCHE	0	w	**Correct Hall Event flag set** Writing a 1_B to this field sets the **PFLG**.CHES bit field. An interrupt pulse is generated. A read always returns 0.
SWHE	1	w	**Wrong Hall Event flag set** Writing a 1_B to this field sets the **PFLG**.WHES bit field. An interrupt pulse is generated. A read always returns 0.
SHIE	2	w	**Hall Inputs Update Event flag set** Writing a 1_B to this field sets the **PFLG**.HIES bit field. An interrupt pulse is generated. A read always returns 0.
SMST	4	w	**Multi-Channel Pattern shadow transfer flag set** Writing a 1_B to this field sets the **PFLG**.MSTS bit field. An interrupt pulse is generated. A read always returns 0.
SINDX	8	w	**Quadrature Index flag set** Writing a 1_B to this field sets the **PFLG**.INDXS bit field. An interrupt pulse is generated. A read always returns 0.
SERR	9	w	**Quadrature Phase Error flag set** Writing a 1_B to this field sets the **PFLG**.ERRS bit field. An interrupt pulse is generated. A read always returns 0.

Field	Bits	Type	Description
SCNT	10	w	**Quadrature CLK flag set** Writing a 1_B to this field sets the **PFLG**.CNTS bit field. An interrupt pulse is generated. A read always returns 0.
SDIR	11	w	**Quadrature Direction flag set** Writing a 1_B to this field sets the **PFLG**.DIRS bit field. An interrupt pulse is generated. A read always returns 0.
SPCLK	12	w	**Quadrature period clock flag set** Writing a 1_B to this field sets the **PFLG**.PCLKS bit field. An interrupt pulse is generated. A read always returns 0.
0	3, [7:5], [31:13]	r	**Reserved** Read always returns 0

RPFLG

Through this register it is possible for the SW to reset/clear a specific interrupt status flag.

RPFLG
POSIF Interrupt Clear **($007C_H$)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0			RPCLK	RDIR	RCNT	RERR	RINDX	0			RMST	0	RHIE	RWHE	RCHE
r			w	w	w	w	w	r			w	r	w	w	w

Field	Bits	Type	Description
RCHE	0	w	**Correct Hall Event flag clear** Writing a 1_B to this field clears the **PFLG**.CHES bit field. A read always returns 0.

Field	Bits	Type	Description
RWHE	1	w	**Wrong Hall Event flag clear** Writing a 1_B to this field clears the **PFLG**.WHES bit field. A read always returns 0.
RHIE	2	w	**Hall Inputs Update Event flag clear** Writing a 1_B to this field clears the **PFLG**.HIES bit field. A read always returns 0.
RMST	4	w	**Multi-Channel Pattern shadow transfer flag clear** Writing a 1_B to this field clears the **PFLG**.MSTS bit field. A read always returns 0.
RINDX	8	w	**Quadrature Index flag clear** Writing a 1_B to this field clears the **PFLG**.INDXS bit field. A read always returns 0.
RERR	9	w	**Quadrature Phase Error flag clear** Writing a 1_B to this field clears the **PFLG**.ERRS bit field. A read always returns 0.
RCNT	10	w	**Quadrature CLK flag clear** Writing a 1_B to this field clears the **PFLG**.CNTS bit field. A read always returns 0.
RDIR	11	w	**Quadrature Direction flag clear** Writing a 1_B to this field clears the **PFLG**.DIRS bit field. A read always returns 0.
RPCLK	12	w	**Quadrature period clock flag clear** Writing a 1_B to this field clears the **PFLG**.PCLKS bit field. A read always returns 0.
0	3, [7:5], [31:13]	r	**Reserved** Read always returns 0

21.8 Interconnects

The following tables, describe the connectivity present on the device for each POSIF module.

The GPIO connections are available at the Ports chapter.

21.8.1 POSIF0 pins

Table 21-7 POSIF0 Pin Connections

Global Inputs/Outputs	I/O	Connected To	Description
POSIF0.CLK	I	SCU.CCUCLK	Module clock is the same one used by the capcoms
POSIF0.IN0A	I	PORTS	Shared connection for rotary encoder and hall sensor
POSIF0.IN0B	I	PORTS	Shared connection for rotary encoder and hall sensor
POSIF0.IN0C	I	VADC.G1BFL0	Shared connection for rotary encoder and hall sensor
POSIF0.IN0D	I	ERU1.PDOUT0	Shared connection for rotary encoder and hall sensor
POSIF0.IN1A	I	PORTS	Shared connection for rotary encoder and hall sensor
POSIF0.IN1B	I	PORTS	Shared connection for rotary encoder and hall sensor
POSIF0.IN1C	I	VADC.G1BFL1	Shared connection for rotary encoder and hall sensor
POSIF0.IN1D	I	ERU1.PDOUT1	Shared connection for rotary encoder and hall sensor
POSIF0.IN2A	I	PORTS	Shared connection for rotary encoder and hall sensor
POSIF0.IN2B	I	PORTS	Shared connection for rotary encoder and hall sensor
POSIF0.IN2C	I	VADC.C0SR0	Shared connection for rotary encoder and hall sensor
POSIF0.IN2D	I	ERU1.PDOUT2	Shared connection for rotary encoder and hall sensor
POSIF0.HSDA	I	CCU40.ST0	Used for the Hall pattern sample delay. Like the dead time counter in capcom6

Table 21-7 POSIF0 Pin Connections

Global Inputs/Outputs	I/O	Connected To	Description
POSIF0.HSDB	I	0	Used for the Hall pattern sample delay. Like the dead time counter in capcom6
POSIF0.EWHEA	I	VADC.G1BFL2	Wrong Hall event emulation, Trap, etc
POSIF0.EWHEB	I	ERU1.IOUT0	Wrong Hall event emulation, Trap, etc
POSIF0.EWHEC	I	ERU1.IOUT1	Wrong Hall event emulation, Trap, etc
POSIF0.EWHED	I	0	Wrong Hall event emulation, Trap, etc
POSIF0.MSETA	I	CCU40.SR0	Multi Pattern updated set. Requests a new shadow transfer
POSIF0.MSETB	I	CCU40.ST1	Multi Pattern updated set. Requests a new shadow transfer
POSIF0.MSETC	I	0	Multi Pattern updated set. Requests a new shadow transfer
POSIF0.MSETD	I	0	Multi Pattern updated set. Requests a new shadow transfer
POSIF0.MSETE	I	CCU80.SR1	Multi Pattern updated set. Requests a new shadow transfer
POSIF0.MSETF	I	ERU1.IOUT2	Multi Pattern updated set. Requests a new shadow transfer
POSIF0.MSETG	I	0	Multi Pattern updated set. Requests a new shadow transfer
POSIF0.MSETH	I	0	Multi Pattern updated set. Requests a new shadow transfer

Table 21-7 POSIF0 Pin Connections

Global Inputs/Outputs	I/O	Connected To	Description
POSIF0.MSYNCA	I	CCU80.PS1	Sync for updating the multi channel pattern with the shadow transfer
POSIF0.MSYNCB	I	CCU80.PS3	Sync for updating the multi channel pattern with the shadow transfer
POSIF0.MSYNCC	I	CCU40.PS1	Sync for updating the multi channel pattern with the shadow transfer
POSIF0.MSYNCD	I	0	Sync for updating the multi channel pattern with the shadow transfer
POSIF0.OUT0	O	CCU40.IN0E; CCU40.IN1E; CCU40.IN2E	Quadrature mode: Quadrature clock; Hall sensor mode: Hall input edge detection
POSIF0.OUT1	O	CCU40.IN0F; CCU40.IN1F;	Quadrature mode: Shaft direction; Hall sensor mode: Correct Hall Event
POSIF0.OUT2	O	CCU40.IN1L; CCU40.IN2F; CCU80.IN0D; CCU80.IN1D; CCU80.IN2D; CCU80.IN3D;	Quadrature mode: Pclk for velocity; Hall sensor mode: Idle/wrong hall event
POSIF0.OUT3	O	CCU40.IN0G; CCU40.IN1G; CCU40.IN2G; CCU40.IN3E	Quadrature mode: Index event used for Clear/capt; Hall sensor mode: stop in Hall sensor mode
POSIF0.OUT4	O	CCU40.IN1H; CCU40.IN2H;	Quadrature mode:Index event; Hall sensor mode: Multi channel pattern update done

Table 21-7 POSIF0 Pin Connections

Global Inputs/Outputs	I/O	Connected To	Description
POSIF0.OUT5	O	CCU40.IN3F; CCU80.IN0E; CCU80.IN1E; CCU80.IN2E; CCU80.IN3E;	Sync start
POSIF0.OUT6	O	CCU80.MCSS;	Multi channel pattern update request
POSIF0.MOUT[0]	O	CCU80.MCI00;	Multi channel pattern
POSIF0.MOUT[1]	O	CCU80.MCI01;	Multi channel pattern
POSIF0.MOUT[2]	O	CCU80.MCI02;	Multi channel pattern
POSIF0.MOUT[3]	O	CCU80.MCI03;	Multi channel pattern
POSIF0.MOUT[4]	O	CCU80.MCI10;	Multi channel pattern
POSIF0.MOUT[5]	O	CCU80.MCI11;	Multi channel pattern
POSIF0.MOUT[6]	O	CCU80.MCI12;	Multi channel pattern
POSIF0.MOUT[7]	O	CCU80.MCI13;	Multi channel pattern
POSIF0.MOUT[8]	O	CCU80.MCI20;	Multi channel pattern
POSIF0.MOUT[9]	O	CCU80.MCI21;	Multi channel pattern
POSIF0.MOUT[10]	O	CCU80.MCI22;	Multi channel pattern
POSIF0.MOUT[11]	O	CCU80.MCI23;	Multi channel pattern
POSIF0.MOUT[12]	O	CCU80.MCI30;	Multi channel pattern
POSIF0.MOUT[13]	O	CCU80.MCI31;	Multi channel pattern
POSIF0.MOUT[14]	O	CCU80.MCI32;	Multi channel pattern
POSIF0.MOUT[15]	O	CCU80.MCI33;	Multi channel pattern
POSIF0.SR0	O	NVIC	Service request line 0
POSIF0.SR1	O	NVIC VADC.G0REQTRO; VADC.BGREQTRO; ERU1.0A1; ERU1.1A1;	Service request line 1

General Purpose I/O Ports

22 General Purpose I/O Ports (PORTS)

The XMC4[12]00 has many digital port pins which can be used as General Purpose I/Os (GPIO) and are connected to the on-chip peripheral units.

22.1 Overview

The PORTS provide a generic and flexible software and hardware interface for all standard digital I/Os. Each port slice has the same software interfaces for the operation as General Purpose I/O and it further provides the connectivity to the on-chip periphery and the control for the pad characteristics. **Table 22-1** gives an overview of the available PORTS and other pins in the different packages of the XMC4[12]00:

Table 22-1 Port/Pin Overview

Function	LQFP-64	VQFN-48	Note
P0	12	9	
P1	10	6	
P2	12	6	
P3	1	–	
P14	9	8	Analog/Digital Input only
Dedicated I/Os	10	10	HIB_IO, TMS, TCK, USB, XTAL, RTC_XTAL, PORST
Analog Supply, Reference and Ground	2	2	Shared VDDA and VAREF, shared VSSA and VAGND
Digital Supply	6	6	VDDP, VDDC, VBAT
Digital Ground	2	1	VSS, VSSO, must be connected to common V_{SS}
Exposed Die Pad	1	–	Must be connected to common V_{SS}

22.1.1 Features

This is a list of the main features of the PORTS:

- same generic register interface for each port pin, **Section 22.8**
- simple and robust software access for General Purpose I/O functionality, **Section 22.2**
- separate set and clear output control to avoid read-modify-write operations, **Section 22.8.5**
- direct input connections to on-chip peripherals, **Section 22.2.1**
- parallel input of the same pin to different peripherals possible, for example triggering a capture event in a CAPCOM unit and a Service Request via the ERU

- up to four alternate output connections from peripherals selectable, **Section 22.2.2**
- separate input and output path, which allows to evaluate the input while the output is active (feedback, plausability check)
- dedicated hardware-controlled interface for LEDTS and QSPI with select option, **Section 22.3**
- programmable open-drain or push-pull output driver stage, **Section 22.8.1**
- programmable driver strength and slew rate, **Page 22-6**
- programmable weak pull-up and pull-down devices, **Section 22.8.1**
- programmable input inverter, **Section 22.2.1**
- programmable power-save behavior in Deep Sleep mode, **Section 22.4**
- defined power-up/power-fail behavior, **Section 22.6**
- Privilege Mode restricted access to configuration registers to avoid accidential modification
- disabling of digital input stage on shared analog input ports, **Section 22.5**

22.1.2 Block Diagram

Below is a figure with the generic structure of a digital port pin, split into the port slice with the control logic and the pad with the pull devices and the input and output stages, **Figure 22-1**.

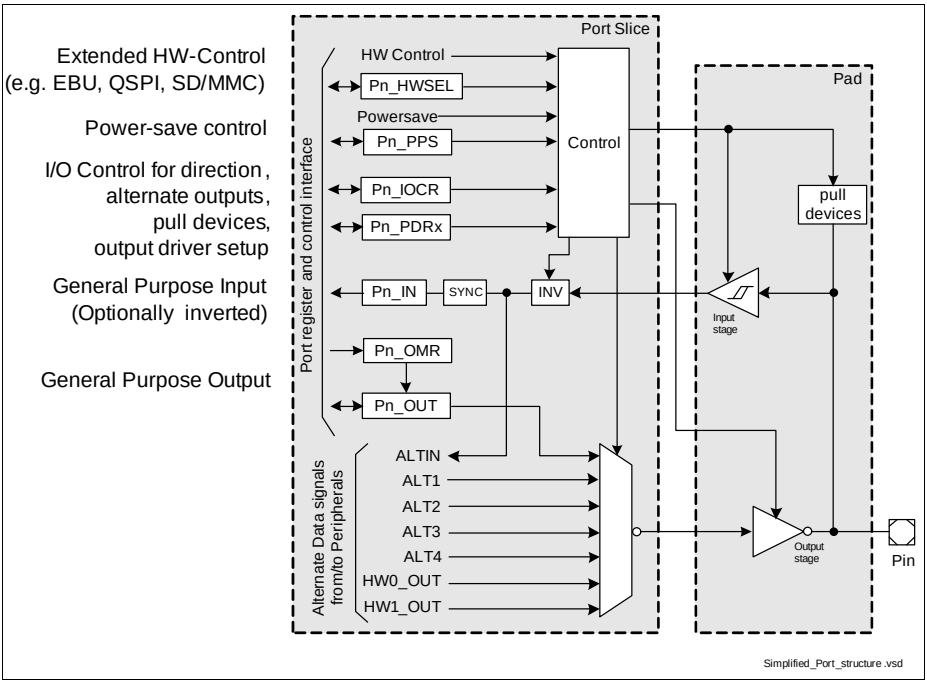


Figure 22-1 General Structure of a digital Port Pin

22.1.3 Definition of Terms

Some specific terms are used throughout this chaper:

- **Pin/Ball:** External connection of the device to the PCB.
- **Dedicated Pin:** A Pin with a dedicated function that is not under the control of the port logic (i.e. supply pins, $\overline{\text{PORST}}$).
- **Port Pin:** A pin under the control of the port logic (P0.1).
- **Port:** A group of up to 16 Port Pins sharing the same generic register set (P0).
- **Port Slice:** The “sum” of register bits and control logic used to control a port pin.
- **Pad:** Analog component containing the output driver, pull devices and input Schmitt-Trigger. Also interfaces the internal logic operating on V_{DDC} to the pad supply domain V_{DDP}.
- **GPIO:** General Purpose Input/Output. A port pin with the input and/or output function controlled by the application software.
- **Alternate Function:** Direct connection of a port pin with an on-chip peripheral.

22.2 GPIO and Alternate Function

The Ports can be operated as General Purpose Input/Outputs (GPIO) and with Alternate Functions of the on-chip periphery, configured by the Port Input/Output Control Register (Pn_IOCR, **Section 22.8.1**). It selects between

- Direct or Inverted Input
  - with or without pull device
- Push-pull or Open-Drain Output driven by
  - Pn_OUT (GPIO)
  - selected peripheral output connections.

As GPIO the port pin is controlled by the application software, reading the input value by the Port Input register Pn_IN (**Section 22.8.6**) and/or defining the output value by the Output Modification Register Pn_OMR (**Section 22.8.5**). Output modification by Pn_OMR is preferred over the direct change of the output value with the Output register Pn_OUT (**Section 22.8.4**), as Pn_OMR allows the manipulation of individual port pins in a single access without “disturbing” other pins controlled by the same Pn_OUT register. If an application uses a GPIO as a bi-directional I/O line, register Pn_IOCR has to be written to switch between input and output functionality.

For the operation with Alternate Functions, the port pins are directly connected to input or output functions of the on-chip periphery. This allows the peripheral to directly evaluate the input value or drive the output value of the port pin without further application software interaction after the initial configuration. The connection of alternate functions is used for control and communication interfaces, like a PWM from a CAPCOM unit or a SPI communication of a USIC channel. A detailed connectivity list of the peripherals to the port pins is given in the **Port I/O Functions** chapter. For specific functions, certain peripherals may also take direct control of “their” port pins, see **Hardware Controlled I/Os**.

22.2.1 Input Operation

As an input, the actual voltage level at the port pin is translated into a logical 0_B or 1_B via a Schmitt-Trigger device within the pad. The resulting input value can be optionally inverted. As general purpose input the signal is synchronized and can be read with the Input register (Pn_IN, **Section 22.8.6**). Alternatively, the input can be connected to multiple on-chip peripherals via the ALTIN signal. Where neccessary, these peripherals have internal controls to select the appropriate port pin with an input multiplexer stage, and will take care of synchronization and the further processing of the input signals (for more details on the input selection and handling see the respective peripheral chapters). With the Pn_IOCR register (**Section 22.8.1**) it is also possible to activate an internal weak pull-up or pull-down device in the pad.
The input register Pn_IN and the ALTIN signal always represent the state of the input, independent whether the port pin is configured as input or output. So, even if the port is

in output mode, the level of the pin can be read by software via Pn_IN and/or a peripheral can use the pin level as an input.

The ALTIN input signal of a port pin can be evaluated by multiple on-chip peripherals at the same time. For example, a pin used as slave select input of a USIC channel configured as SPI slave can also be used as trigger input of the ERU to trigger a service request or a wake-up event when the connected SPI master starts a communication.

22.2.2 Output Operation

In output mode, the output driver is activated and drives the value supplied through the output multiplexer to the port pin. Switching between input and output mode is accomplished through the Pn_IOCR register (**Section 22.8.1**), which

- enables or disables the output driver,
- selects between open-drain and push-pull mode,
- selects the general purpose or alternate function outputs.

The output multiplexer selects the signal source of the output with

- Pn_IOCR
  - general purpose output (Pn_OUT, **Section 22.8.4**)
  - alternate peripheral functions, ALT1..ALT4
- hardware control, Pn_HWSEL
  - HWO0
  - HWO1

Note: It is recommended to complete the Port and peripheral configuration with respect to driver strength, operating mode and inital values before the port pin is switched to output mode.

The output function is exclusive, meaning that always only exactly one peripheral has control of the output path.

Used as general purpose output, software can directly modify the content of Pn_OUT to define the output value on the pin. A write operation to Pn_OUT updates all port pins of that port (e.g. P0) that are configured as general purpose output. Updating just one or a selected few general purpose output pins via Pn_OUT requires a masked read-modify-write operation to avoid disturbing pins that shall not be changed. Direct writes to Pn_OUT will also affect Pn_OUT bits configured for use with the Pin Power-save function, **Section 22.4**.
Because of that, it is preferred to modify Pn_OUT bits by the Output Modification Register Pn_OMR (**Section 22.8.5**). The bits in Pn_OMR allow to individually set, clear or toggle the bits in the Pn_OUT register and only update the “addressed” Pn_OUT bits.

The data written by software into the output register Pn_OUT can also be used as input data to an on-chip peripheral. This enables, for example, peripheral tests and simulation via software without external circuitry.

Output lines of on-chip peripherals can directly control the output value of the output driver if selected via ALT1 to ALT4 as well as HW0_OUT and HW1_OUT. After initialization, this allows the connected peripherals to directly drive complex control and communication patterns without further software interaction with the ports.

The actual logic level at the pin can be examined through reading Pn_IN and compared against the applied output level (either applied by the output register Pn_OUT, or via an alternate output function of a peripheral unit). This can be used to detect some electrical failures at the pin caused through external circuitry. In addition, software-supported arbitration schemes between different “masters” can be implemented in this way, using the open-drain configuration and an external wired-AND circuitry. Collisions on the external communication lines can be detected when a high level (1_B) is output, but a low level (0_B) is seen when reading the pin value via the input register Pn_IN or directly by a peripheral (via ALTIN, for example a USIC channel in IIC mode).

Driver Mode

Before activating the push-pull driver, it is recommended to configure its driver strength and slew rate according to its pad class and the application need by the Pad Driver Mode register Pn_PDR (**Section 22.8.2**). Selecting the appropriate driver strength allows to optimize the outputs for the needed interface performance, can help to reduce power consumption, and limits noise, crosstalk and electromagnetic emissions.

There are three classes of GPIO output drivers:

- Class A1 pads (low speed 3.3V LVTTL outputs)
- Class A1+ pads (medium speed 3.3V LVTTL outputs)

Class A1 pins provide the choice between medium and weak output drivers.
Class A1+ pins provide the choice between strong/medium/weak output drivers. For the strong driver, the signal transition edge can be additionally selected as soft or slow.

The assignment of each port pin to one of these pad classes is listed in the **Package Pin Summary** table. Further details about pad properties in the XMC4[12]00 are summarized in the Data Sheet.

22.3 Hardware Controlled I/Os

Some ports pins are overlaid with peripheral functions for which the connected peripheral needs direct hardware control, e.g. for the direction of a bi-directional data bus. There is a dedicated hardware control interface for these functions. As multiple peripherals need access to this interface, the Pn_HWSEL register (**Section 22.8.8**) allows to select between the hardware “masters”. The assigned functions are listed in the columns HWO0/HWI0 in the **Port I/O Functions** table.

Depending on the operating mode, the peripheral can take control of various functions:

- Pin direction, input or output, e.g. for bi-directional signals
- Driver type, open-drain or push-pull

- Pull devices under peripheral control or under standard control via Pn_IOCR

Some configurations remain under control by the standard configuration interface, the output driver strength by Pn_PDR and the direct or inverted input path by Pn_IOCR.

Pn_HWSEL.HWx just pre-assigns the hardware-control of the pin to a certain peripheral, but the peripheral itself decides to actually take control over it. As long as the peripheral does not take control of a given pin via HWx_EN, the configuration of this pin is still defined by the configuration registers and it is available as GPIO or for other alternate functions. This might be because the selected peripheral has controls to just activate a subset of its pins, or because the peripheral is not active at all. E.g. unused address lines of the EBU are free for use as GPIO.

This mechanism can also be used to prohibit the hardware control of certain pins to a peripheral, in case the application does not need the respective functionality and the peripheral has no controls to disable the hardware control selectively.

If not specified differently, hardware outputs activate the push-pull output driver and the strength is defined by Pn_PDR. Similarly, the default hardware input configuration and the pull devices are controlled by Pn_IOCR.

If the JTAG interface is selected by Pn_HWSEL of the respective port pins, pull device and output driver configuration is overridden by hardware.

If configured accordingly, the LEDTS module can also control the internal pull devices and change between push-pull and open-drain output drivers.

Note: Do not enable the Pin Power Save function for pins configured for Hardware Control (Pn_HWSEL.HWx != 00_B). Doing so may result in an undefined behavior of the pin when the device enters the Deep Sleep state.

22.4 Power Saving Mode Operation

In Deep Sleep mode, the behavior of a pin depends on the setting of the Pin Power Save register Pn_PPS (**Section 22.8.7**). Basically, each pin can be configured to react to the Power Save Mode Request or to ignore it. In case a pin is configured to react to a Power Save Mode Request, the output driver is switched to tri-state, the input Schmitt-Trigger and the pull devices are switched off (see **Figure 22-2**). The input signal to the on-chip peripherals is optionally driven statically high or low, software-defined by a value stored in Pn_OUT or by the last input value sampled to the Pn_OUT register during normal operation. The actual reaction is configured with the Pn_IOCR register under power save conditions, see **Table 22-8**.

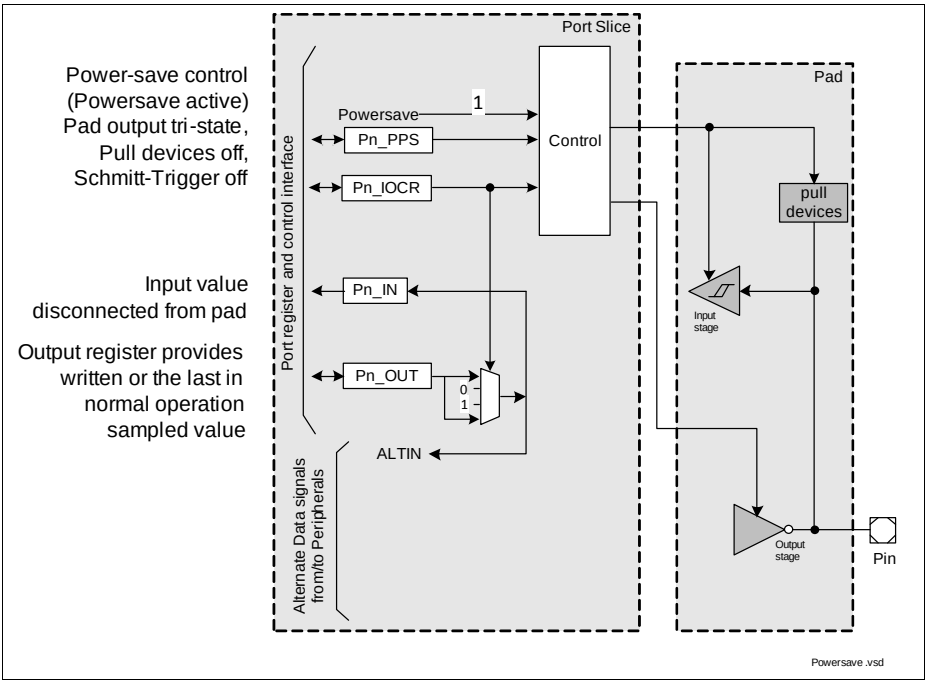


Figure 22-2 Port Pin in Power Save State

Note: Do not enable the Pin Power Save function for pins configured for Hardware Control (Pn_HWSEL.HWx != 00_B). Doing so may result in an undefined behavior of the pin when the device enters the Deep Sleep state.

22.5 Analog Ports

P14 and P15 are analog and digital input ports with a simplified port and pad structure, see **Figure 22-3**. The analog pads have no output drivers and the digital input Schmitt-Trigger can be controlled by the Pn_PDISC (**Section 22.8.3**) register. Accordingly, the port control interface is reduced in its functionality. The Pn_IOCR register controls the pull devices, the optional input inversion and the input source in power-save mode. The Pn_OUT has only its power-save functionality, as described in **Section 22.4**.

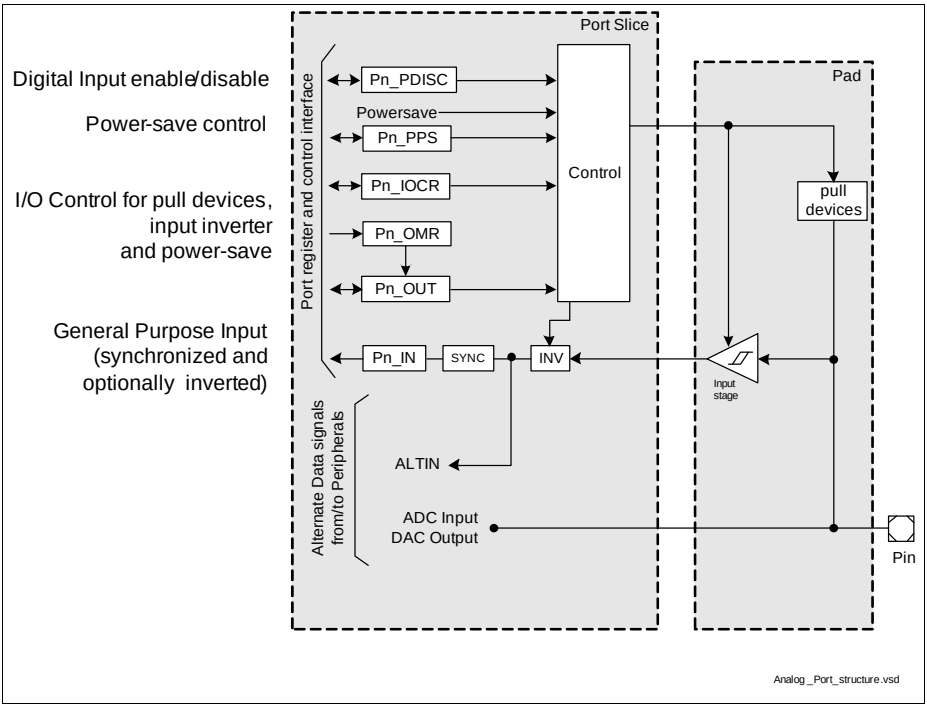


Figure 22-3 Analog Port Structure

22.6 Power, Reset and Clock

During power-up, until V_{DDC} and V_{DDP} voltage levels are stable and within defined limits, or during power-fail, while one/some voltage levels are outside the defined limits, the digital I/O pads are held in a defined state, which is tri-state input, output driver disabled and no pull devices active. The JTAG interface activates pull devices as default configuration.

The pins of the battery-buffered Hibernate Domain are independent to the supply monitoring of V_{DDC} and V_{DDP}, but are reset with the Standby Reset (see Reset Control chapter in the System Control Unit).

See the SCU Power Management chapter and the Data Sheet for details on the power-up, supply monitoring and voltage limits.

All Port registers are reset with the System Reset (see Reset Control chapter in the System Control Unit). The standard reset values are defined such that the port pins are configured as tri-state inputs, output driver disabled and no pull devices active.

Exceptions from these standard values are related to special interfaces (for example JTAG) or the analog input channels.

The Ports register interface is connected to Peripheral Bridge 1 (PBA1) and all registers of the Ports are clocked with f_{PERIPH}.

22.7 Initialization and System Dependencies

It is recommended to follow pre-defined routines for the initialization of the port pins.

Input

When a peripheral shall use a port pin as input, the actual pin levels may immediately trigger an unexpected peripheral event (e.g. clock edge at SPI). This can be avoided by forcing the "passive" level via pull-up/down programming.

The following steps are required to configure a port pin as an input:

- Pn_IOCR
  input configuration with pull device and/or power-save mode configuration
- Hardware Control (if applicable)
  - Pn_HWSEL
    switch hardware control to peripheral
- Pin Power Save (if applicable)
  - Pn_OMR/Pn_OUT
    default value in power save mode (if applicable)
  - Pn_PPS
    enable power save control

Output

When a port pin is configured as output for an on-chip peripheral, it is important that the peripheral is configured before the port switches the control to the peripheral in order to avoid spikes on the output.
The following steps are required to configure a port pin as an output:

- Pn_OMR/Pn_OUT
  Initial output value (as general purpose output)
- Pn_PDR
  Pad Driver Strength configuration
- GPIO or Alternate Output
  - Pn_IOCR
    Output multiplexer select
    Push-pull or open-drain output driver mode
    Activates the output driver!
- Hardware Control

– Pn_IOCR
  depending on the hardware function Pn_IOCR can enable the internal pull devices
– Pn_HWSEL
  Switch hardware control to peripheral

Transitions

If a port pin is used for different functions that require a reconfiguration of the port registers, it is recommended to do this transition via an intermediate “neutral” tri-state input configuration.

- Pn_HWSEL
  disable hardware selection; can be omitted if no hardware control is used on the port pin
- Pn_PPS
  disable power save mode control of the pin; can be ommited if no power save configuration is used on the port pin
- Pn_IOCR
  tri-state input and no pull device active

22.8 Registers

Registers Overview

The absolute register address is calculated by adding:

Module Base Address + Offset Address

Table 22-2 Registers Address Space

Module	Base Address	End Address	Note
P0	$4802\ 8000_H$	$4802\ 80FF_H$	
P1	$4802\ 8100_H$	$4802\ 81FF_H$	
P2	$4802\ 8200_H$	$4802\ 82FF_H$	
P3	$4802\ 8300_H$	$4802\ 83FF_H$	
P14	$4802\ 8E00_H$	$4802\ 8EFF_H$	Analog/Digital Input only

Table 22-3 Register Overview

Short Name	Description	Offset Addr.	Access Mode		Description See
			Read	Write	
Pn_OUT	Port n Output Register	0000_H	U, PV	U, PV	Page 22-24
Pn_OMR	Port n Output Modification Register	0004_H	U, PV	U, PV	Page 22-25
–	Reserved	0008_H-$000C_H$	BE	BE	–
Pn_IOCR0	Port n Input/Output Control Register 0	0010_H	U, PV	PV	Page 22-14
Pn_IOCR4	Port n Input/Output Control Register 4	0014_H	U, PV	PV	Page 22-15
Pn_IOCR8	Port n Input/Output Control Register 8	0018_H	U, PV	PV	Page 22-15
Pn_IOCR12	Port n Input/Output Control Register 12	$001C_H$	U, PV	PV	Page 22-16
–	Reserved	0020_H	BE	BE	–
Pn_IN	Port n Input Register	0024_H	U, PV	R	Page 22-26
–	Reserved	0028_H-$003C_H$	BE	BE	–
Pn_PDR0	Port n Pad Driver Mode 0 Register	0040_H	U, PV	PV	Page 22-20
Pn_PDR1	Port n Pad Driver Mode 1 Register	0044_H	U, PV	PV	Page 22-21
–	Reserved	0048_H-$005C_H$	BE	BE	–
Pn_PDISC	Port n Pin Function Decision Control Register (non-ADC ports)	0060_H	U, PV	BE	Page 22-22
P14_PDISC P15_PDISC	Port n Pin Function Decision Control Register (ADC ports)	0060_H	U, PV	PV	Page 22-23
–	Reserved	0064_H-$006C_H$	BE	BE	–
Pn_PPS	Port n Pin Power Save Register	0070_H	U, PV	PV	Page 22-27

Table 22-3 Register Overview (cont'd)

Short Name	Description	Offset Addr.	Access Mode		Description See
			Read	Write	
Pn_HWSEL	Port n Hardware Select Register	0074_H	U, PV	PV	Page 22-28
–	Reserved	0078_H-$00FC_H$	BE	BE	–

Table 22-4 Registers Access Rights and Reset Classes

Register Short Name	Access Rights		Reset Class
	Read	Write	
Pn_IN	U, PV	R	System Reset
Pn_OUT		U, PV	
Pn_OMR			
Pn_IOCR0		PV	
Pn_IOCR4			
Pn_IOCR8			
Pn_IOCR12			
Pn_PDISC (ADC ports)			
Pn_PDR0			
Pn_PDR1			
Pn_PPS			
Pn_HWSEL			
Pn_PDISC (non-ADC ports)		BE	

22.8.1 Port Input/Output Control Registers

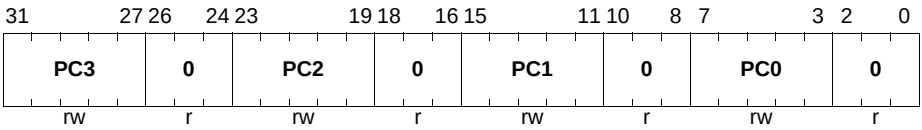
The port input/output control registers select the digital output and input driver functionality and characteristics of a GPIO port pin. Port direction (input or output), pull-up or pull-down devices for inputs, and push-pull or open-drain functionality for outputs can be selected by the corresponding bit fields PCx (x = 0-15). Each 32-bit wide port input/output control register controls four GPIO port lines:

Register Pn_IOCR0 controls the Pn.[3:0] port lines
Register Pn_IOCR4 controls the Pn.[7:4] port lines
Register Pn_IOCR8 controls the Pn.[11:8] port lines
Register Pn_IOCR12 controls the Pn.[15:12] port lines

The diagrams below show the register layouts of the port input/output control registers with the PCx bit fields. One PCx bit field controls exactly one port line Pn.x.

Pn_IOCR0 (n=0-3)
Port n Input/Output Control Register 0
(4802 8010$_H$ + n*100$_H$) **Reset Value: 0000 0000$_H$**

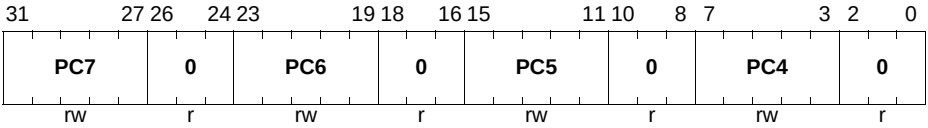
P14_IOCR0
Port 14 Input/Output Control Register 0
(0010$_H$) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
PC0, PC1, PC2, PC3	[7:3], [15:11], [23:19], [31:27]	rw	**Port Control for Port n Pin 0 to 3** This bit field determines the Port n line x functionality (x = 0-3) according to the coding table (see **Table 22-5**).
0	[2:0], [10:8], [18:16], [26:24]	r	**Reserved** Read as 0; should be written with 0.

Pn_IOCR4 (n=0-2)
Port n Input/Output Control Register 4
(4802 8014$_H$ + n*100$_H$) **Reset Value: 0000 0000$_H$**

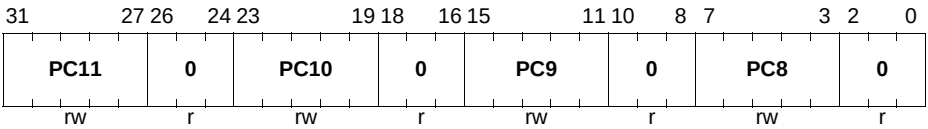
P14_IOCR4
Port 14 Input/Output Control Register 4
(0014$_H$) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
PC4, **PC5,** **PC6,** **PC7**	[7:3], [15:11], [23:19], [31:27]	rw	**Port Control for Port n Pin 4 to 7** This bit field determines the Port n line x functionality (x = 4-7) according to the coding table (see **Table 22-5**).
0	[2:0], [10:8], [18:16], [26:24]	r	**Reserved** Read as 0; should be written with 0.

Pn_IOCR8 (n=0-2)
Port n Input/Output Control Register 8
(4802 8018$_H$ + n*100$_H$) **Reset Value: 0000 0000$_H$**

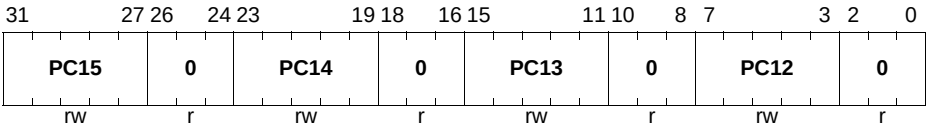
P14_IOCR8
Port n Input/Output Control Register 8
(0018$_H$) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
PC8, PC9, PC10, PC11	[7:3], [15:11], [23:19], [31:27]	rw	**Port Control for Port n Pin 8 to 11** This bit field determines the Port n line x functionality (x = 8-11) according to the coding table (see **Table 22-5**).
0	[2:0], [10:8], [18:16], [26:24]	r	**Reserved** Read as 0; should be written with 0.

Pn_IOCR12 (n=1-2)
Port n Input/Output Control Register 12
(4802 801C$_H$ + n*100$_H$) **Reset Value: 0000 0000$_H$**

P14_IOCR12
Port 14 Input/Output Control Register 12
(001C$_H$) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
PC12, PC13, PC14, PC15	[7:3], [15:11], [23:19], [31:27]	rw	**Port Control for Port n Pin 12 to 15** This bit field determines the Port n line x functionality (x = 12-15) according to the coding table (see **Table 22-5**).
0	[2:0], [10:8], [18:16], [26:24]	r	**Reserved** Read as 0; should be written with 0.

Depending on the GPIO port functionality (number of GPIO lines of a port), not all of the port input/output control registers are implemented.

The structure with one control bit field for each port pin located in different register bytes offers the possibility to configure the port pin functionality of a single pin with byte-oriented accesses without accessing the other PCx bit fields.

Port Control Coding

Table 22-5 describes the coding of the PCx bit fields that determine the port line functionality.

The Pn_IOCRy PCx bit field is also used to contol the pin behavior in Deep Sleep mode if the Pin Power Save option is enabled, see **Section 22.8.7**.

Table 22-5 Standard PCx Coding[1)]

PCx[4:0]	I/O	Output Characteristics	Selected Pull-up / Pull-down / Selected Output Function
$0X000_B$	Direct Input	–	No internal pull device active
$0X001_B$			Internal pull-down device active
$0X010_B$			Internal pull-up device active
$0X011_B$			No internal pull device active; Pn_OUTx continuously samples the input value
$0X100_B$	Inverted Input	–	No internal pull device active
$0X101_B$			Internal pull-down device active
$0X110_B$			Internal pull-up device active
$0X111_B$			No internal pull device active; Pn_OUTx continuously samples the input value

Table 22-5 Standard PCx Coding[1)] (cont'd)

PCx[4:0]	I/O	Output Characteristics	Selected Pull-up / Pull-down / Selected Output Function
10000_B	Output (Direct Input)	Push-pull	General-purpose output
10001_B			Alternate output function 1
10010_B			Alternate output function 2
10011_B			Alternate output function 3
10100_B			Alternate output function 4
10101_B			Reserved.
10110_B			Reserved.
10111_B			Reserved.
11000_B		Open-drain	General-purpose output
11001_B			Alternate output function 1
11010_B			Alternate output function 2
11011_B			Alternate output function 3
11100_B			Alternate output function 4
11101_B			Reserved.
11110_B			Reserved.
11111_B			Reserved.

1) For the analog and digital input ports P14 and P15 the combinations with PCx[4]=1_B are reserved.

22.8.2 Pad Driver Mode Register

The pad structure of the XMC4[12]00 GPIO lines offers the possibility to select the output driver strength and the slew rate. These two parameters are controlled by the bit fields in the pad driver mode registers Pn_PDR0/1, independently from input/output and pull-up/pull-down control functionality as programmed in the Pn_IOCR register. Pn_PDR0 and Pn_PDR1 registers are assigned to each port.

Depending on the assigned pad class, the 3-bit wide pad driver mode selection bit fields PDx in the pad driver mode registers Pn_PDR make it possible to select the port line functionality as shown in **Table 22-6**. Note that the pad driver mode registers are specific for each port.

Table 22-6 Pad Driver Mode Selection

Pad Class	PDx.2	PDx.1	PDx.0	Functionality
A1	X	X	0	Medium driver
			1	Weak driver
A1+	0	X	0	Strong driver soft edge
	0	X	1	Strong driver slow edge
	1	X	0	Medium driver
	1	X	1	Weak driver
A2	0	0	0	Strong driver, sharp edge
	0	0	1	Strong driver, medium edge
	0	1	0	Strong driver, soft edge
	0	1	1	Reserved
	1	0	0	Medium driver
	1	0	1	
	1	1	0	Reserved
	1	1	1	Weak driver

Note: The XMC4[12]00 Data Sheet describes the DC characteristics of all pad classes.

Pad Driver Mode Registers

This is the general description of the PDR registers. Each port contains its own specific PDR registers, described additionally at each port, that can contain between one and eight PDx fields for PDR0 and PDR1 registers, respectively. Each field controls 1 pin. For coding of PDx, see **Table 22-6**.

The analog and digital input ports P14 and P15 don’t have Pn_PDR registers.

Pn_PDR0 (n=0-3)

Port n Pad Driver Mode 0 Register(4802 8040$_H$ + n*100$_H$) Reset Value: 2222 2222$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**PD7**			**0**	**PD6**			**0**	**PD5**			**0**	**PD4**		
r	rw			r	rw			r	rw			r	rw		

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**PD3**			**0**	**PD2**			**0**	**PD1**			**0**	**PD0**		
r	rw			r	rw			r	rw			r	rw		

Field	Bits	Type	Description
PD0	[2:0]	rw	**Pad Driver Mode for Pn.0**
PD1	[6:4]	rw	**Pad Driver Mode for Pn.1**
PD2	[10:8]	rw	**Pad Driver Mode for Pn.2**
PD3	[14:12]	rw	**Pad Driver Mode for Pn.3**
PD4	[18:16]	rw	**Pad Driver Mode for Pn.4**
PD5	[22:20]	rw	**Pad Driver Mode for Pn.5**
PD6	[26:24]	rw	**Pad Driver Mode for Pn.6**
PD7	[30:28]	rw	**Pad Driver Mode for Pn.7**
0	3, 7, 11, 15, 19, 23, 27, 31	r	**Reserved** Read as 0; should be written with 0.

Pn_PDR1 (n=0-2)
Port n Pad Driver Mode 1 Register

(4802 8044$_H$ + n*100$_H$) **Reset Value: 2222 2222$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**PD15**			**0**	**PD14**			**0**	**PD13**			**0**	**PD12**		
r	rw			r	rw			r	rw			r	rw		

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**PD11**			**0**	**PD10**			**0**	**PD9**			**0**	**PD8**		
r	rw			r	rw			r	rw			r	rw		

Field	Bits	Type	Description
PD8	[2:0]	rw	**Pad Driver Mode for Pn.8**
PD9	[6:4]	rw	**Pad Driver Mode for Pn.9**
PD10	[10:8]	rw	**Pad Driver Mode for Pn.10**
PD11	[14:12]	rw	**Pad Driver Mode for Pn.11**
PD12	[18:16]	rw	**Pad Driver Mode for Pn.12**
PD13	[22:20]	rw	**Pad Driver Mode for Pn.13**
PD14	[26:24]	rw	**Pad Driver Mode for Pn.14**
PD15	[30:28]	rw	**Pad Driver Mode for Pn.15**
0	3, 7, 11, 15, 19, 23, 27, 31	r	**Reserved** Read as 0; should be written with 0.

22.8.3 Pin Function Decision Control Register

Pin Function Decision Control Register

The primary use for this register is to disable/enable the digital pad structure in shared analog and digital ports, see the dedicated description of the **Pn_PDISC (n=14-15)** register of the analog ports.

For “normal” digital I/O ports (P0-P6) this register is read-only and the read value corresponds to the available pins in the given package.

Pn_PDISC (n=0-3)
Port n Pin Function Decision Control Register
(4802 8060$_H$ + n*100$_H$) **Reset Value: 0000 XXXX$_H$[1)]**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PDIS 15	**PDIS 14**	**PDIS 13**	**PDIS 12**	**PDIS 11**	**PDIS 10**	**PDIS 9**	**PDIS 8**	**PDIS 7**	**PDIS 6**	**PDIS 5**	**PDIS 4**	**PDIS 3**	**PDIS 2**	**PDIS 1**	**PDIS 0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

1) The reset value is package dependent.

Field	Bits	Type	Description
PDISx (x = 0-15)	x	r	**Pad Disable for Port n Pin x** 0_B Pad Pn.x is enabled. 1_B Pad Pn.x is disabled.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

Pn_PDISC (n=14-15)
Port n Pin Function Decision Control Register

(4802 8060$_H$ + n*100$_H$) **Reset Value: 0000 XXXX$_H$[1)]**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PDIS 15	**PDIS 14**	**PDIS 13**	**PDIS 12**	**PDIS 11**	**PDIS 10**	**PDIS 9**	**PDIS 8**	**PDIS 7**	**PDIS 6**	**PDIS 5**	**PDIS 4**	**PDIS 3**	**PDIS 2**	**PDIS 1**	**PDIS 0**
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

1) The reset value is package dependent.

Field	Bits	Type	Description
PDISx (x = 0-15)	x	rw	**Pad Disable for Port n Pin x** This bit disables or enables the digital pad function. 0_B Digital Pad input is enabled. Analog and digital input path active. 1_B Digital Pad input is disabled. Analog input path active. (default)
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

22.8.4 Port Output Register

The port output register determines the value of a GPIO pin when it is selected by Pn_IOCRx as output. Writing a 0 to a Pn_OUT.Px (x = 0-15) bit position delivers a low level at the corresponding output pin. A high level is output when the corresponding bit is written with a 1. Note that the bits of Pn_OUT.Px can be individually set/reset by writing appropriate values into the port output modification register Pn_OMR, avoiding read-modify-write operations on the Pn_OUT, which might affect other pins of the port.

The Pn_OUT is also used to store/drive a defined value for the input in Deep Sleep mode. For details on this see the **Port Pin Power Save Register**. That is also the only use of the Pn_OUT register in the analog and digital input port P14.

Pn_OUT (n=0-3)
Port n Output Register **(4802 8000$_H$ + n*100$_H$)** **Reset Value: 0000 0000$_H$**
P14_OUT
Port 14 Output Register **(0000$_H$)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0
rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
Px **(x = 0-15)**	x	rwh	**Port n Output Bit x** This bit determines the level at the output pin Pn.x if the output is selected as GPIO output. 0_B The output level of Pn.x is 0. 1_B The output level of Pn.x is 1. Pn.x can also be set/reset by control bits of the Pn_OMR register.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

22.8.5 Port Output Modification Register

The port output modification register contains control bits that make it possible to individually set, reset, or toggle the logic state of a single port line by manipulating the output register.

Pn_OMR (n=0-3)
Port n Output Modification Register
(4802 8004$_H$ + n*100$_H$) **Reset Value: 0000 0000$_H$**

P14_OMR
Port 14 Output Modification Register
(0004$_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PR 15	**PR 14**	**PR 13**	**PR 12**	**PR 11**	**PR 10**	**PR 9**	**PR 8**	**PR 7**	**PR 6**	**PR 5**	**PR 4**	**PR 3**	**PR 2**	**PR 1**	**PR 0**
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PS 15	**PS 14**	**PS 13**	**PS 12**	**PS 11**	**PS 10**	**PS 9**	**PS 8**	**PS 7**	**PS 6**	**PS 5**	**PS 4**	**PS 3**	**PS 2**	**PS 1**	**PS 0**
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Field	Bits	Type	Description
PSx (x = 0-15)	x	w	**Port n Set Bit x** Setting this bit will set or toggle the corresponding bit in the port output register Pn_OUT. The function of this bit is shown in **Table 22-7**.
PRx (x = 0-15)	x + 16	w	**Port n Reset Bit x** Setting this bit will reset or toggle the corresponding bit in the port output register Pn_OUT. The function of this bit is shown in **Table 22-7**.

Note: Register Pn_OMR is virtual and does not contain any flip-flop. A read action delivers the value of 0. A 8 or 16-bits write behaves like a 32-bit write padded with zeros.

Table 22-7 Function of the Bits PRx and PSx

PRx	PSx	Function
0	0	Bit Pn_OUT.Px is not changed.
0	1	Bit Pn_OUT.Px is set.
1	0	Bit Pn_OUT.Px is reset.
1	1	Bit Pn_OUT.Px is toggled.

22.8.6 Port Input Register

The logic level of a GPIO pin can be read via the read-only port input register Pn_IN. Reading the Pn_IN register always returns the current logical value at the GPIO pin, synchronized to avoid meta-stabilities, independently whether the pin is selected as input or output.

Pn_IN (n=0-3)
Port n Input Register **(4802 8024$_H$ + n*100$_H$)** **Reset Value: 0000 XXXX$_H$**
P14_IN
Port 14 Input Register **(0024$_H$)** **Reset Value: 0000 XXXX$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0
rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
Px **(x = 0-15)**	x	rh	**Port n Input Bit x** This bit indicates the level at the input pin Pn.x. 0_B The input level of Pn.x is 0. 1_B The input level of Pn.x is 1.
0	[31:16]	r	**Reserved** Read as 0.

22.8.7 Port Pin Power Save Register

When the XMC4[12]00 enters Deep Sleep mode, pins with enabled Pin Power Save option are set to a defined state and the input Schmitt-Trigger as well as the output driver stage are switched off.

Note: Do not enable the Pin Power Save function for pins configured for Hardware Control (Pn_HWSEL.HWx != 00_B). Doing so may result in an undefined behavior of the pin when the device enters the Deep Sleep state.

Pn_PPS (n=0-3)
Port n Pin Power Save Register
($4802\ 8070_H + n*100_H$) **Reset Value: $0000\ 0000_H$**

P14_PPS
Port 14 Pin Power Save Register
(0070_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PPS 15	PPS 14	PPS 13	PPS 12	PPS 11	PPS 10	PPS 9	PPS 8	PPS 7	PPS 6	PPS 5	PPS 4	PPS 3	PPS 2	PPS 1	PPS 0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
PPSx (x = 0-15)	x	rw	**Port n Pin Power Save Bit x** 0_B Pin Power Save of Pn.x is disabled. 1_B Pin Power Save of Pn.x is enabled.
0	[31:16]	r	**Reserved** Read as 0.

Deep Sleep Pin Power Save behavior

The actual behavior in Deep Sleep mode with enabled Pin Power Save is controlled by the Pn_IOCRy.PCx bit field (**Page 22-14**) of the respective pin. **Table 22-8** shows the coding.

Table 22-8 PCx Coding in Deep Sleep mode

PCx[4:0]	I/O	Normal Operation or PPSx=0_B	Deep Sleep mode and PPSx=1_B
$0X000_B$	Direct Input	See **Table 22-5**	Input value=Pn_OUTx
$0X001_B$			Input value=0_B; pull-down deactivated
$0X010_B$			Input value=1_B; pull-up deactivated
$0X011_B$			Input value=Pn_OUTx, storing the last sampled input value
$0X100_B$	Inverted Input	See **Table 22-5**	Input value=$\overline{\text{Pn\_OUTx}}$
$0X101_B$			Input value=1_B; pull-down deactivated
$0X110_B$			Input value=0_B; pull-up deactivated
$0X111_B$			Input value=$\overline{\text{Pn\_OUTx}}$, storing the last sampled input value
$1XXXX_B$	Output	See **Table 22-5**	Output driver off, Input Schmitt-Trigger off, no pull device active, Input value=Pn_OUTx

22.8.8 Port Pin Hardware Select Register

Some peripherals require direct hardware control of their I/Os. As on some pins multiple such peripheral I/Os are mapped, the register Pn_HWSEL is used to select which peripheral has the control over the pin.

Note: Pn_HWSEL.HWx just pre-assigns the hardware-control of the pin to a certain peripheral, but the peripheral itself decides to actually take control over it. As long as the peripheral does not take control of a given pin via HWx_EN, the configuration of this pin is still defined by the configuration registers and it is available as GPIO or for other alternate functions. This might be because the selected peripheral has controls to just activate a subset of its pins, or because the peripheral is not active at all.

This mechanism can also be used to prohibit the hardware control of certain pins to a peripheral, in case the application does not need the respective functionality and the peripheral has no controls to disable the hardware control selectively.

The shared analog and digital input ports P14 and P15 do not support the hardware select feature.

P0_HWSEL
Port 0 Pin Hardware Select Register (0074_H) Reset Value: $0001\ 4000_H$
P1_HWSEL
Port 1 Pin Hardware Select Register (0074_H) Reset Value: $0000\ 0000_H$
P2_HWSEL
Port 2 Pin Hardware Select Register (0074_H) Reset Value: $0000\ 0004_H$
P3_HWSEL
Port 3 Pin Hardware Select Register (0074_H) Reset Value: $0000\ 0000_H$
P14_HWSEL
Port 14 Pin Hardware Select Register (0074_H) Reset Value: $0000\ 0000_H$

31 30	29 28	27 26	25 24	23 22	21 20	19 18	17 16
HW15	**HW14**	**HW13**	**HW12**	**HW11**	**HW10**	**HW9**	**HW8**
rw	rw	rw	rw	rw	rw	rw	rw

15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
HW7	**HW6**	**HW5**	**HW4**	**HW3**	**HW2**	**HW1**	**HW0**
rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
HWx (x = 0-15)	[2*x+1: 2*x]	rw	**Port n Pin Hardware Select Bit x** 00_B Software control only. 01_B HWI0/HWO0 control path can override the software configuration. 10_B HWI1/HWO1 control path can override the software configuration. 11_B Reserved.

22.9 Package Pin Summary

The following general building block is used to describe each pin:

Table 22-9 Package Pin Mapping Description

Function	Package A	Package B	...	Pad Type	Notes
Name	N	Ax	...	A1+	

The table is sorted by the “Function” column, starting with the regular Port pins (Px.y), followed by the dedicated pins (i.e. PORST) and supply pins.

The following columns, titled with the supported package variants, lists the package pin number to which the respective function is mapped in that package.

The “Pad Type” indicates the employed pad type (A1, A1+, special=special pad, In=input pad, AN/DIG_IN=analog and digital input, Power=power supply). Details about the pad properties are defined in the Data Sheet.

In the “Notes”, special information to the respective pin/function is given, i.e. deviations from the default configuration after reset. Per default the regular Port pins are configured as direct input with no internal pull device active.

Table 22-10 Package Pin Mapping

Function	LQFP-64	VQFN-48	Pad Type	Notes
P0.0	2	2	A1+	
P0.1	1	1	A1+	
P0.2	64	48	A1+	
P0.3	63	47	A1+	
P0.4	62	46	A1+	
P0.5	61	45	A1+	
P0.6	60	44	A1+	
P0.7	58	43	A1+	After a system reset, via HWSEL this pin selects the DB.TDI function.
P0.8	57	42	A1+	After a system reset, via HWSEL this pin selects the DB.TRST function, with a weak pull-down active.
P0.9	4	-	A1+	
P0.10	3	-	A1+	
P0.11	59	-	A1+	

Table 22-10 Package Pin Mapping (cont'd)

Function	LQFP-64	VQFN-48	Pad Type	Notes
P1.0	52	40	A1+	
P1.1	51	39	A1+	
P1.2	50	38	A1+	
P1.3	49	37	A1+	
P1.4	48	36	A1+	
P1.5	47	35	A1+	
P1.7	55	-	A1+	
P1.8	54	-	A1+	
P1.9	53	-	A1+	
P1.15	46	-	A1+	
P2.0	34	26	A1+	
P2.1	33	25	A1+	After a system reset, via HWSEL this pin selects the DB.TDO function.
P2.2	32	24	A1+	
P2.3	31	23	A1+	
P2.4	30	22	A1+	
P2.5	29	21	A1+	
P2.6	36	-	A1+	
P2.7	35	-	A1+	
P2.8	28	-	A1+	
P2.9	27	-	A1+	
P2.14	26	-	A1+	
P2.15	25	-	A1+	
P3.0	5	-	A1+	
P14.0	20	16	AN/DIG_IN	
P14.3	19	15	AN/DIG_IN	
P14.4	18	14	AN/DIG_IN	
P14.5	17	13	AN/DIG_IN	
P14.6	16	12	AN/DIG_IN	
P14.7	15	11	AN/DIG_IN	
P14.8	24	20	AN/DAC/DIG_IN	
P14.9	23	19	AN/DAC/DIG_IN	

Table 22-10 Package Pin Mapping (cont'd)

Function	LQFP-64	VQFN-48	Pad Type	Notes
P14.14	14	-	AN/DIG_IN	
USB_DP	7	4	special	
USB_DM	6	3	special	
HIB_IO_0	10	7	A1 special	At the first power-up and with every reset of the hibernate domain this pin is configured as open-drain output and drives "0". As output the medium driver mode is active.
TCK	45	34	A1	Weak pull-down active.
TMS	44	33	A1+	Weak pull-up active. As output the strong-soft driver mode is active.
$\overline{\text{PORST}}$	43	32	special	Strong pull-down controlled by EVR. Weak pull-up active while strong pull-down is not active.
XTAL1	39	29	clock_IN	
XTAL2	40	30	clock_O	
RTC_XTAL1	11	8	clock_IN	
RTC_XTAL2	12	9	clock_O	
VBAT	13	10	Power	When VDDP is supplied VBAT has to be supplied as well.
VDDA/VAREF	22	18	AN_Power/AN_Ref	Shared analog supply and reference voltage pin.
VSSA/VAGND	21	17	AN_Power/AN_Ref	Shared analog supply and reference ground pin.
VDDC	9	6	Power	
VDDC	42	31	Power	
VDDP	8	5	Power	
VDDP	38	28	Power	
VDDP	56	41	Power	
VSS	37	27	Power	

Table 22-10 Package Pin Mapping (cont'd)

Function	LQFP-64	VQFN-48	Pad Type	Notes
VSSO	41	-	Power	
VSS	Exp. Pad	Exp. Pad	Power	**Exposed Die Pad** The exposed die pad is connected internally to VSS. For proper operation, it is mandatory to connect the exposed pad directly to the common ground on the board. For thermal aspects, please refer to the Data Sheet. Board layout examples are given in an application note.

22.10 Port I/O Functions

The following general building block is used to describe each PORT pin:

Table 22-11 Port I/O Function Description

Function	Outputs			Inputs		
	ALT1	ALTn	HWO0	HWI0	Input	Input
P0.0		MODA.OUT	MODB.OUT	MODB.INA	MODC.INA	
Pn.y	MODA.OUT				MODA.INA	MODC.INB

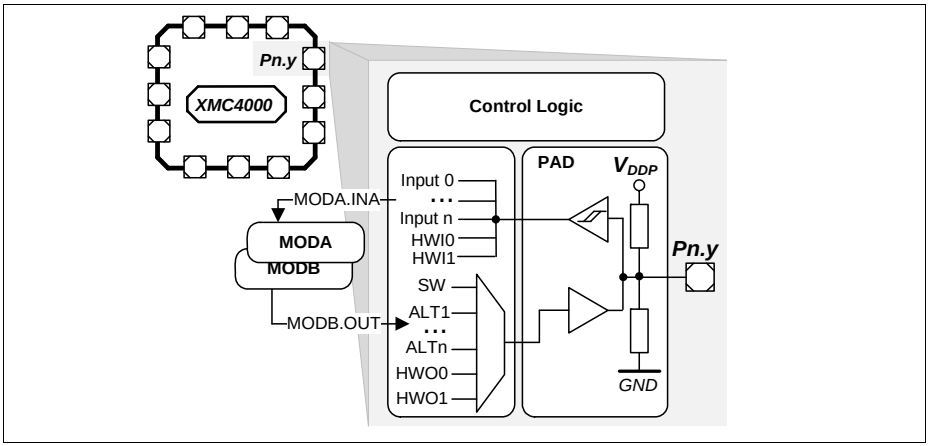


Figure 22-4 Simplified Port Structure

Pn.y is the port pin name, defining the control and data bits/registers associated with it. As GPIO, the port is under software control. Its input value is read via Pn_IN.y, Pn_OUT defines the output value.

Up to four alternate output functions (ALT1/2/3/4) can be mapped to a single port pin, selected by Pn_IOCR.PC. The output value is directly driven by the respective module, with the pin characteristics controlled by the port registers (within the limits of the connected pad).

The port pin input can be connected to multiple peripherals. Most peripherals have an input multiplexer to select between different possible input sources.
The input path is also active while the pin is configured as output. This allows to feedback an output to on-chip resources without wasting an additional external pin.

By Pn_HWSEL (**Section 22.8.8**) it is possible to select between different hardware “masters” (HWO0/HWI0). The selected peripheral can take control of the pin(s). Hardware control overrules settings in the respective port pin registers.

22.10.1 Port I/O Function Table

Table 22-12 Port I/O Functions

Function	Output					Input								
	ALT1	ALT2	ALT3	ALT4	HWO0	HWI0	Input	Input	Input	Input	Input	Input	Input	Input
P0.0		CAN. N0_TXD	CCU80. OUT21	LEDTS0. COL2			U1C1. DX0D		ERU0. 0B0	USB. VBUSDETECT A		HRPWM0. C1INB		
P0.1		U1C1. DOUT0	CCU80. OUT11	LEDTS0. COL3					ERU0. 0A0			HRPWM0. C2INB		
P0.2		U1C1. SELO1	CCU80. OUT01	HRPWM0. HROUT01	U1C0. DOUT3	U1C0. HWIN3			ERU0. 3B3					
P0.3			CCU80. OUT20	HRPWM0. HROUT20	U1C0. DOUT2	U1C0. HWIN2				ERU1. 3B0				
P0.4			CCU80. OUT10	HRPWM0. HROUT21	U1C0. DOUT1	U1C0. HWIN1		U1C0. DX0A	ERU0. 2B3					
P0.5		U1C0. DOUT0	CCU80. OUT00	HRPWM0. HROUT00	U1C0. DOUT0	U1C0. HWIN0		U1C0. DX0B		ERU1. 3A0				
P0.6		U1C0. SELO0	CCU80. OUT30	HRPWM0. HROUT30				U1C0. DX2A	ERU0. 3B2		CCU80. IN2B			
P0.7	WWDT. SERVICE_OUT	U0C0. SELO0		HRPWM0. HROUT11		DB. TDI	U0C0. DX2B		ERU0. 2B1		CCU80. IN0A	CCU80. IN1A	CCU80. IN2A	CCU80. IN3A
P0.8	SCU. EXTCLK	U0C0. SCLKOUT		HRPWM0. HROUT10		$\overline{\text{DB.}}$ $\overline{\text{TRST}}$	U0C0. DX1B		ERU0. 2A1		CCU80. IN1B			
P0.9	HRPWM0. HROUT31	U1C1. SELO0	CCU80. OUT12	LEDTS0. COL0			U1C1. DX2A		ERU0. 1B0					
P0.10		U1C1. SCLKOUT	CCU80. OUT02	LEDTS0. COL1			U1C1. DX1A		ERU0. 1A0					
P0.11		U1C0. SCLKOUT	CCU80. OUT31					U1C0. DX1A	ERU0. 3A2					
P1.0		U0C0. SELO0	CCU40. OUT3	ERU1. PDOUT3			U0C0. DX2A		ERU0. 3B0		CCU40. IN3A	HRPWM0. C0INA		
P1.1		U0C0. SCLKOUT	CCU40. OUT2	ERU1. PDOUT2			U0C0. DX1A	POSIF0. IN2A	ERU0. 3A0		CCU40. IN2A	HRPWM0. C1INA		
P1.2			CCU40. OUT1	ERU1. PDOUT1	U0C0. DOUT3	U0C0. HWIN3		POSIF0. IN1A		ERU1. 2B0	CCU40. IN1A	HRPWM0. C2INA		
P1.3		U0C0. MCLKOUT	CCU40. OUT0	ERU1. PDOUT0	U0C0. DOUT2	U0C0. HWIN2		POSIF0. IN0A		ERU1. 2A0	CCU40. IN0A	HRPWM0. C0INB		
P1.4	WWDT. SERVICE_OUT	CAN. N0_TXD	CCU80. OUT33		U0C0. DOUT1	U0C0. HWIN1	U0C0. DX0B	CAN. N1_RXDD	ERU0. 2B0		CCU41. IN0C	HRPWM0. BL0A		
P1.5	CAN. N1_TXD	U0C0. DOUT0	CCU80. OUT23		U0C0. DOUT0	U0C0. HWIN0	U0C0. DX0A	CAN. N0_RXDA	ERU0. 2A0	ERU1. 0A0	CCU41. IN1C			
P1.7		U0C0. DOUT0		U1C1. SELO2						USB. VBUSDETECT B				
P1.8		U0C0. SELO1		U1C1. SCLKOUT										

Table 22-12 Port I/O Functions (cont'd)

Function	Output					Input								
	ALT1	ALT2	ALT3	ALT4	HWO0	HWI0	Input	Input	Input	Input	Input	Input	Input	Input
P1.9	U0C0. SCLKOUT			U1C1. DOUT0										
P1.15	SCU. EXTCLK			U1C0. DOUT0						ERU1. 1A0				
P2.0	CAN. N0_TXD			LEDTS0. COL1					ERU0. 0B3		CCU40. IN1C			
P2.1				LEDTS0. COL0	DB.TDO/ TRACESWO					ERU1. 0B0	CCU40. IN0C			
P2.2	VADC. EMUX00		CCU41. OUT3	LEDTS0. LINE0	LEDTS0. EXTENDED0	LEDTS0. TSIN0A		U0C1. DX0A	ERU0. 1B2		CCU41. IN3A			
P2.3	VADC. EMUX01	U0C1. SELO0	CCU41. OUT2	LEDTS0. LINE1	LEDTS0. EXTENDED1	LEDTS0. TSIN1A		U0C1. DX2A	ERU0. 1A2		CCU41. IN2A			
P2.4	VADC. EMUX02	U0C1. SCLKOUT	CCU41. OUT1	LEDTS0. LINE2	LEDTS0. EXTENDED2	LEDTS0. TSIN2A		U0C1. DX1A	ERU0. 0B2		CCU41. IN1A	HRPWM0. BL1A		
P2.5		U0C1. DOUT0	CCU41. OUT0	LEDTS0. LINE3	LEDTS0. EXTENDED3	LEDTS0. TSIN3A		U0C1. DX0B	ERU0. 0A2		CCU41. IN0A	HRPWM0. BL2A		
P2.6			CCU80. OUT13	LEDTS0. COL3				CAN. N1_RXDA	ERU0. 1B3		CCU40. IN3C			
P2.7		CAN. N1_TXD	CCU80. OUT03	LEDTS0. COL2						ERU1. 1B0	CCU40. IN2C			
P2.8			CCU80. OUT32	LEDTS0. LINE4	LEDTS0. EXTENDED4	LEDTS0. TSIN4A	DAC. TRIGGER5				CCU40. IN0B	CCU40. IN1B	CCU40. IN2B	CCU40. IN3B
P2.9			CCU80. OUT22	LEDTS0. LINE5	LEDTS0. EXTENDED5	LEDTS0. TSIN5A	DAC. TRIGGER4				CCU41. IN0B	CCU41. IN1B	CCU41. IN2B	CCU41. IN3B
P2.14	VADC. EMUX11	U1C0. DOUT0	CCU80. OUT21					U1C0. DX0D						
P2.15	VADC. EMUX12		CCU80. OUT11	LEDTS0. LINE6	LEDTS0. EXTENDED6	LEDTS0. TSIN6A		U1C0. DX0C						
P3.0		U0C1. SCLKOUT					U0C1. DX1B				CCU80. IN2C			
P14.0							VADC. G0CH0							
P14.3							VADC. G0CH3	VADC. G1CH3			CAN. N0_RXDB			
P14.4							VADC. G0CH4							
P14.5							VADC. G0CH5				POSIF0. IN2B			
P14.6							VADC. G0CH6				POSIF0. IN1B		G0ORC6	
P14.7							VADC. G0CH7				POSIF0. IN0B			
P14.8					DAC. OUT_0			VADC. G1CH0						
P14.9					DAC. OUT_1			VADC. G1CH1						

Table 22-12 Port I/O Functions (cont'd)

Function	Output					Input								
	ALT1	ALT2	ALT3	ALT4	HWO0	HWI0	Input	Input	Input	Input	Input	Input	Input	Input
P14.14								VADC. G1CH6					G1ORC6	
USB_DP														
USB_DM														
HIB_IO_0	HIBOUT	WWDT. SERVICE_OUT					WAKEUPA			USB. VBUSDETECT C				
TCK						DB.TCK/ SWCLK								
TMS					DB.TMS/ SWDIO									
$\overline{\text{PORST}}$														
XTAL1							U0C0. DX0F	U0C1. DX0F	U1C0. DX0F	U1C1. DX0F				
XTAL2														
RTC_XTAL1									ERU0. 1B1					
RTC_XTAL2														

Startup Modes

23 Startup Modes

This chapter describes the various startup modes supported by the device along with actions that must be performed by the end user.

23.1 Overview

The on-chip firmware resides in a non volatile memory namely the BootROM and is the first software of any kind to be executed by the CPU right after reset.

The on-chip firmware has two parts:

- Startup Software (SSW) which provisions the various boot modes and is the main thread of execution
- Test Firmware (Testware, not described in this document) which deals with test related routines that can be invoked during chip test in test mode only

The terms startup mode and boot mode mean the same and are used interchangeably throughout this chapter.

23.1.1 Features

Supported boot modes are summarized briefly. Desired boot mode can be enabled by driving the boot mode pins (JTAG TCK and TMS) with appropriate logic levels and issuing a power on reset (PORST). Some boot modes can only be selected by configuring STCON.SWCON bit field (of SCU module) and applying a system reset (such as a watchdog reset or CPU software reset).

Normal Boot Mode

Startup type: ***Internal start***

Required Reset type: *PORST and System reset*

An application located at the start of flash is given control after SSW has finished its execution.

Alternative Boot Mode (ABM-0/ABM-1)

Startup type: *Internal start*

Required Reset type: *System reset*

An application located at user defined location on the flash is given control by SSW. The SSW after completing its execution evaluates a header hereafter known as ABM header kept at a well known address on the flash which in turn provides the location of application placed at user defined address. Two such applications can be programmed into the flash and thus two ABMs are supported. An invalid header results in the SSW aborting further execution and launching the CPU into safe mode. A PORST is required to exit the safe mode of operation.

Fallback ABM

Startup type: *Internal start*

Required Reset type: *System reset*

SSW evaluates the two ABM headers in succession. The first ABM header found valid by SSW results in application referenced by that header given control to. Should both the headers be found unusable, SSW aborts further execution and places the CPU into a safe mode. A PORST is required to exit the safe mode of operation.

PSRAM boot

Startup type: *Internal start*

Required Reset type: *System reset*

An application loaded into PSRAM is given control after SSW finishes its execution. The start address of this application is deduced from an ABM like header placed in the last 32 bytes of PSRAM.

ASC BSL

Startup type: *External start*

Required Reset type: *PORST and System reset*

An application can be downloaded into the start of PSRAM over the USIC ASC interface and executed. The size of the application downloaded is limited to the size of PSRAM on the device.

CAN BSL (CAN Bootstrap loading)

Startup type: *External start*

Required Reset type: *PORST and System reset*

This is same as ASC BSL mode except that the application is downloaded over CAN interface and executed.

Boot Mode Index (BMI)

Startup type: *Not applicable*

Required Reset type: *PORST and System reset*

A user defined bootmode is offered via following mechanism. With initial factory programming (at customer site), a so called BMI string can be programmed into user configuration block (UCB). The BMI string describes actions that must be performed by SSW before lending into one of the internal or external startup modes.

23.2 Startup Modes

This section describes the various device startup modes summarized in **Chapter 23.1.1**.

23.2.1 Reset Types and Corresponding Boot Modes

XMC4000 platforms supports 2 categories of reset namely Power On Reset (PORST) and System reset. Every cause of reset which is not a PORST is a system reset.

When the SSW executes after a PORST, it gets to choose from one of Normal boot mode, ASC BSL, CAN BSL and BMI based on what is read off the boot pins (JTAG TCK and TMS).

When the SSW executes after a system reset, it chooses one of the following boot modes - Normal, ASC BSL, BMI, CAN BSL, PSRAM boot, ABM-0, ABM-1 and Fallback ABM.

Following table enlists the boot mode pin encoding and associated boot modes.

Table 23-1 Boot Mode Pin Encoding for PORST

TCK	TMS	HWCON[1]	HWCON[0]	Boot mode
0	0	0	1	ASC BSL
0	1	0	0	Normal
1	0	1	1	CAN BSL
1	1	1	0	BMI

TCK and TMS are cached into HWCON[1:0] bit field of the SCU STCON register after PORST. HWCON bits are read by the SSW upon emergence from PORST.

SCU STCON.HWCON is mirrored by STCON.SWCON[1:0] after a PORST.

STCON contents can only be modified by PORST.

SWCON bitfield is read by firmware when the device emerges from a system reset. The following table enlists the encoding of the SWCON bitfield and boot modes. In the event when software does not explicitly program SWCON and a system reset is experienced, values on TCK and TMS decide the boot mode.

Table 23-2 System Reset Boot Modes

SWCON[3:0]	Boot mode
0000_B	Normal
0001_B	ASC BSL
0010_B	BMI
0011_B	CAN BSL

Table 23-2 System Reset Boot Modes (cont'd)

SWCON[3:0]	Boot mode
0100_B	PSRAM boot
1000_B	ABM-0
1100_B	ABM-1
1110_B	Fallback ABM

23.2.2 Initial Boot Sequence

There are several tasks which the SSW performs before it gets to the point where the user requested boot mode must be identified and launched. This is to ensure that user applications have a stable execution environment when program control is eventually ceded to them.

The SSW ensures that the flash subsystem has initialized before any user program can be executed out of flash memory.

It identifies the user requested boot mode and launches the same. It is important to state at this point that a few DSRAM1 locations are used by SSW for staging information read out of Flash configuration sector. The figure below depicts usage of DSRAM1 by SSW.

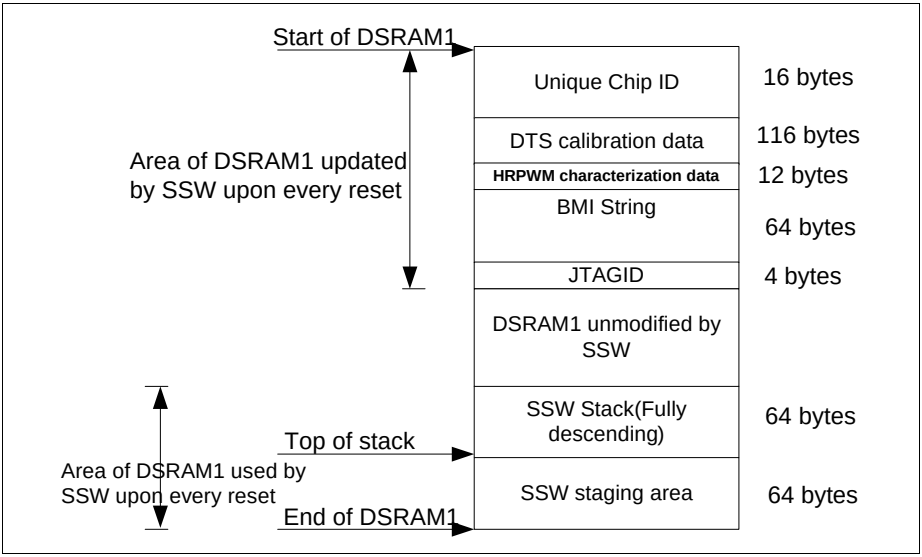


Figure 23-1 DSRAM1 usage by SSW

23.2.3 Boot Mode Selection

HWCON bit field is read only for PORST (Power ON Reset). For every other reset type (available in SCU_RSTSTAT) register, the SWCON field is assessed.

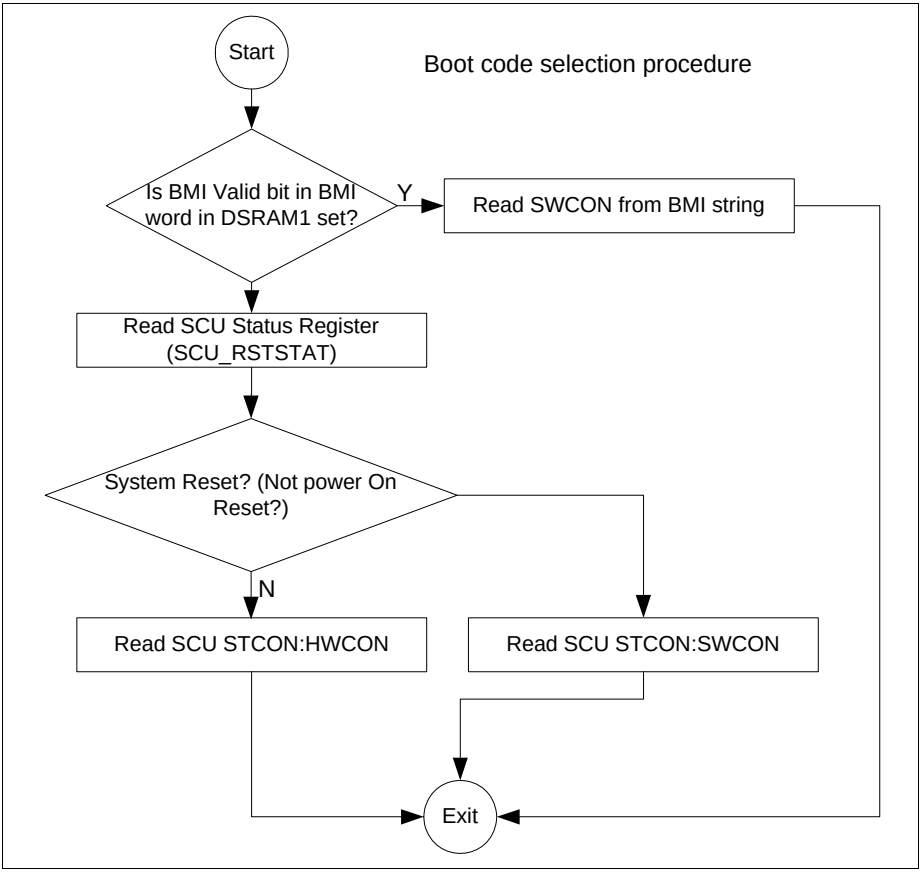


Figure 23-2 Reading Bootcode

Figure 23-3 depicts the decision tree that the SSW has to traverse in order to select the desired boot mode.

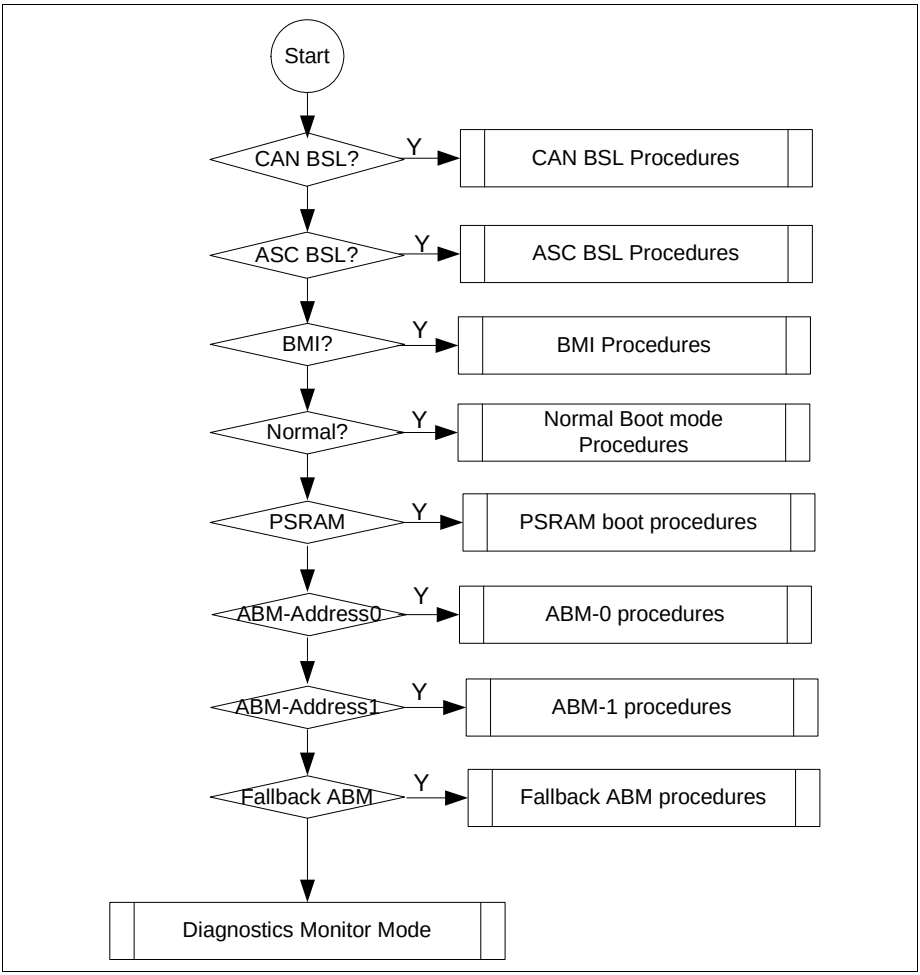


Figure 23-3 Boot Mode Identification

23.2.4 Normal Boot Mode

This is a boot mode in which user application available at the start of flash ($0C000000_H$) is given control to after SSW execution.

SSW enables access to coresight system before ceding control to the first user instruction. If the HALT after RESET feature were requested for by a connected hardware debugger, SSW configures a breakpoint on the first user instruction.

Debugger access is prohibited if the global flash read protection were found enabled. Before program control can be ceded to user application (described next), SSW turns on the Startup protection feature. This restricts access to certain registers (described in various chapters as startup protected registers) and memories such as the Flash Configuration Sector (FCS).

It is expected that the vector table of user application is available at the start of the flash. Firmware essentially reprograms the Cortex M4's SCB VTOR register with the start address of flash ($0C000000_H$) and cedes control to user application by programing register R15 (Program Counter) with reset vector contents. The reset vector contents point to a routine that could be in either the cached or the uncached address space of the flash.

$0C000000_H$ is the uncached start address of the flash.

Users have the option of linking their applications (Vector table, Text and Constant Data) to the cached address space (08000000_H - $080FFFFF_H$). Thus the cached address space becomes Virtual Memory Address (VMA). The Load Memory Address (LMA) for the cached space is the range $0C000000_H$ - $0C0FFFFF_H$.The linking mechanism of the software tool chain must ensure that the following equation is maintained.

VMA = LMA & 04000000_H

Users may also choose to link their applications to uncached space in which case the VMA and the LMA are the same.

When an application has been linked to cached address space, user code should reprogram Cortex M4's SCB VTOR register with the VMA of the vector table.

Figure 23-4, **Figure 23-5** and **Figure 23-6** depict commonly followed application memory layout.

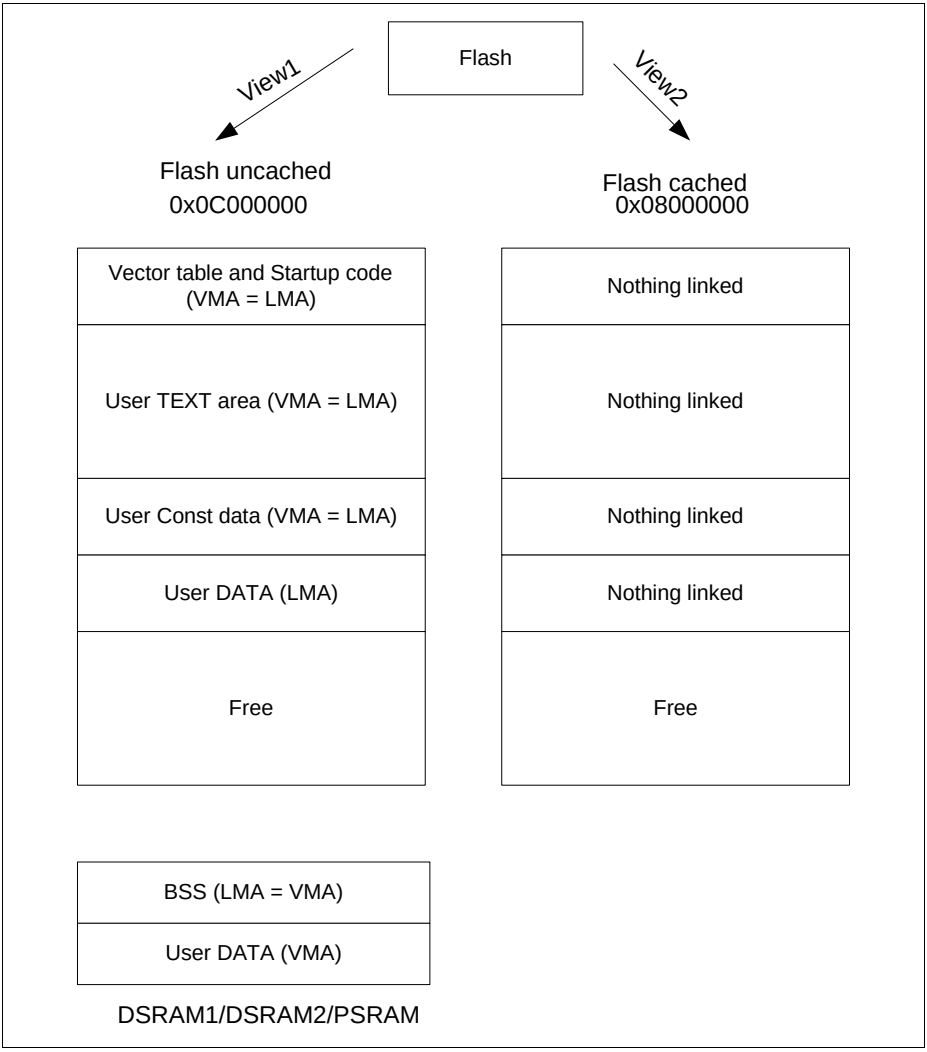


Figure 23-4 Memory Layout1

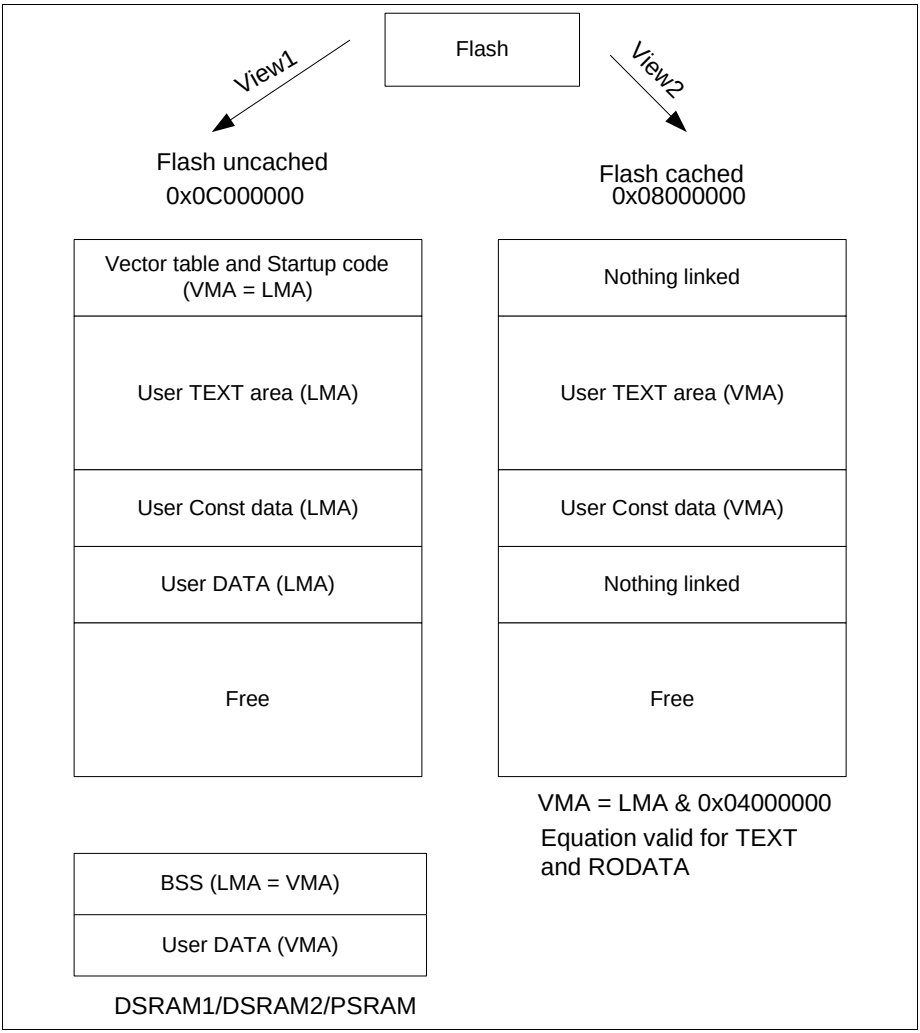


Figure 23-5 Memory Layout2

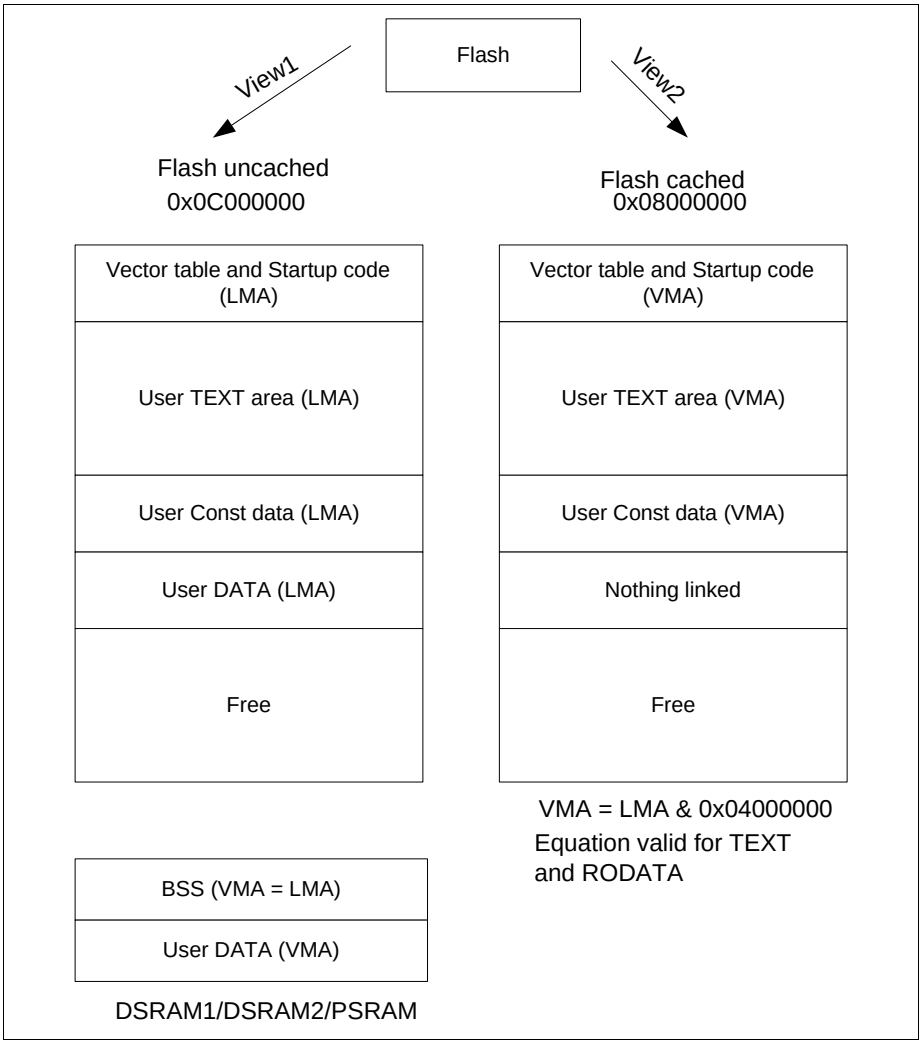


Figure 23-6 Memory Layout3

User Actions For Normal Boot Mode

User must:

- Flash the application at the start of flash
- Drive TCK and TMS as per **Table 23-1**

- Issue a PORST
- Alternatively, a currently running application can write to STCON.SWCON and issue a system reset (SWCON encoding available in **Table 23-2**)

23.2.5 Boot from PSRAM

This boot mode option requires user code to be downloaded into Program SRAM (PSRAM) first.

The SWCON bit field of the SCU STCON register is then expected to be programmed with the Boot from PSRAM boot code. This must be followed by initiation of any of the system resets.

PORST leaves SRAM contents undefined. Any other reset results in previous PSRAM contents retained intact. Hence in order for this boot mode to function as desired, the reset type simply cannot be PORST. Application initiated software reset or a watchdog reset are two examples of system reset.

For SSW to branch to user application in PSRAM, it must first be assured of integrity of user application. This is done by means of a magic key ($A5C3E10F_H$) and CRC audit. It is therefore required that the PSRAM boot header be placed at the last 32 bytes of the PSRAM. The layout of the header is depicted in **Figure 23-7**. Polynomial used for checksum calculation is of CRC-32 type ($04C11DB7_H$).

Magic Key (32 bits)
Start address of code (32 bits)
Length of the code (32 bits)
CRC-32 code for code range (32 bits)
CRC-32 code for above 4 (32 bits)

Figure 23-7 PSRAM Header Layout

This layout is reused in the ABM boot modes. A pictorial representation PSRAM usage for this boot mode is presented in **Figure 23-8**.

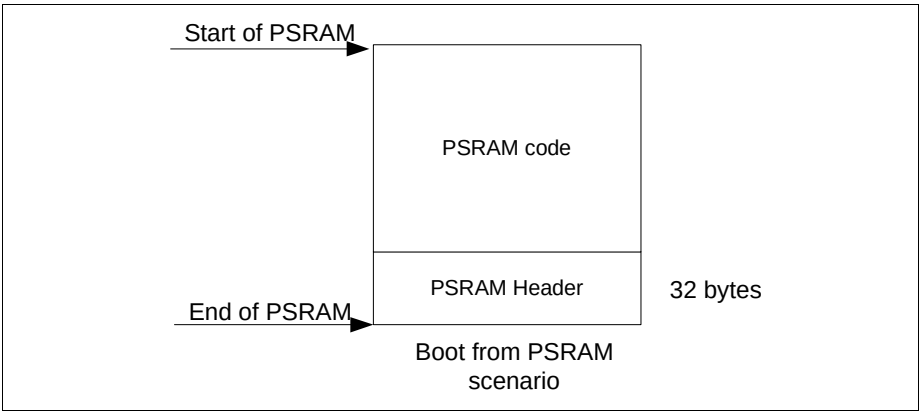


Figure 23-8 PSRAM Layout for PSRAM Boot

After audits confirm integrity of the code, SSW installs startup protection and cedes control to user application.

SCB VTOR is programmed with PSRAM start address and CPU register R15 with application reset vector. Note that PSRAM start address differs between device marking variants.

User Actions for PSRAM Boot Mode

User must:

- Download application (Vector table + code) into PSRAM (Currently running program launched by any of the other internal boot modes can do this)
- Create PSRAM header and program the last 32 bytes of PSRAM with this header (Currently running program can either download header from host or create a header after application has been downloaded)
- Program SWCON bit field in the SCU STCON register
- Issue a system reset

23.2.6 Alternative Boot Mode - Address0 (ABM-0)

SSW can cede control to user application residing at an user defined flash address. As described in **Figure 23-7**, an identical header is expected to be present however at a fixed location on the flash. This fixed address for the header is the last 32 bytes (Example $0C00FFE0_H$ on XMC4500) of the first 64 KB physical sector.

Should the SSW find a corrupted header, execution is aborted and a diagnostics monitor mode is entered into.

As a norm, the address range of an application is typically linear without any holes. As an exception, an application may be scattered on the flash (with the help of scatter linker

scripts) thus leaving holes on the flash. In such as cases, SSW only audits the ABM header and decides if control may be ceded to the reset vector. Distinction between linear and scattered application is made by evaluating application CRC and application length fields of the header. Both of these fields are set to $FFFFFFFF_H$ in the case of scattered application. Polynomial used for CRC calculations is CRC-32 ($04C11DB7_H$). A pictorial representation of this concept is presented in **Figure 23-9**.

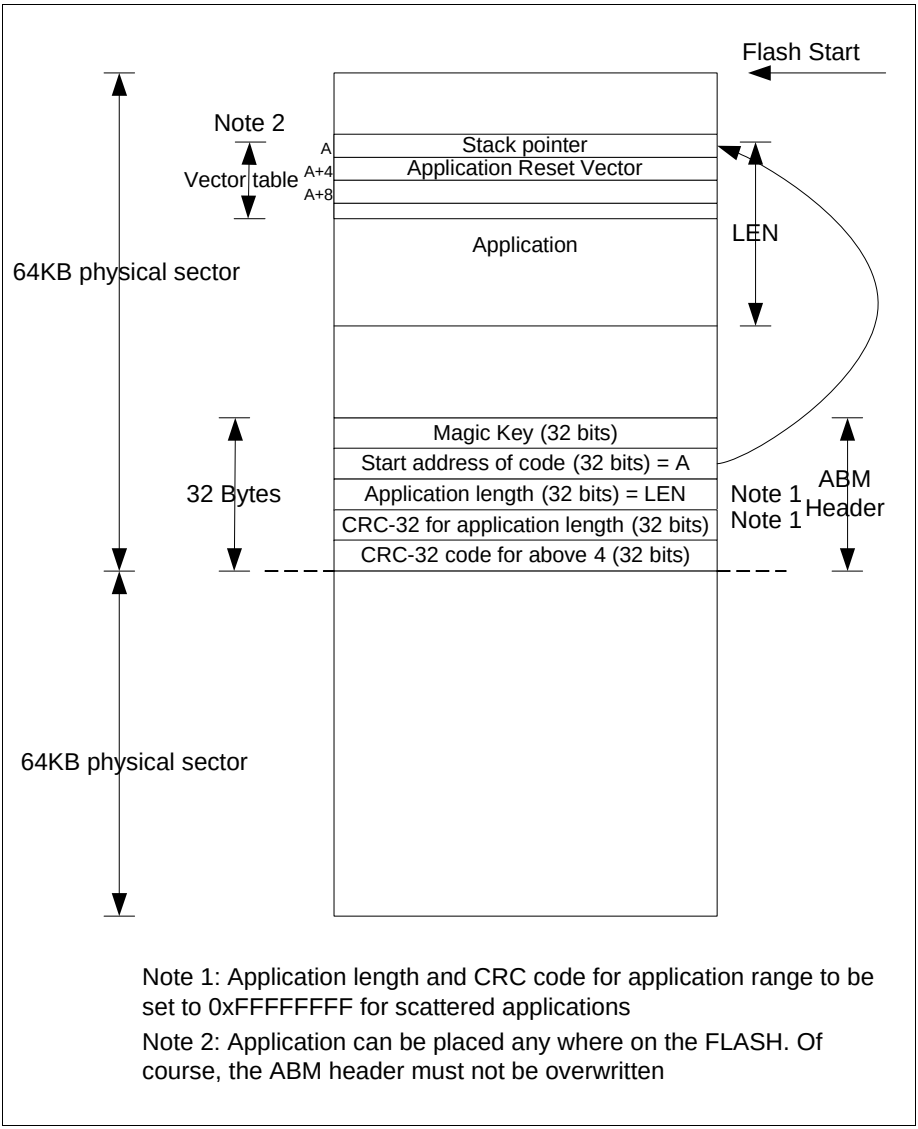


Figure 23-9 ABM Concept

User actions

User must:

- Program flash with application and ABM header
- Drive TCK and TMS to launch Normal boot mode
- Configure STCON.SWCON per **Table 23-2**
- Issue a system reset (Example: Watchdog reset, Software reset)
- Alternatively, program the flash with application and ABM header
- Encode the SWCON field in the BMI word with ABM boot code
- Drive TCK and TMS for BMI and issue PORST

23.2.7 Alternative Boot Mode - Address1 (ABM-1)

This is same as ABM-Address0. Address for the header is the last 32 bytes (Example $0C01FFE0_H$ for XMC4500) of the second 64 KB physical sector.

23.2.8 Fallback ABM

When this boot mode is selected, ABM Address-0 header is audited first. A positive audit results in SSW ceding control to user application pointed to by the header. A negative audit results in evaluation of ABM Address-1. Should the audit of ABM-Address1 header fail, SSW launched diagnostics monitor mode.

23.2.9 ASC BSL Mode

SSW supports bootstrap loader modes. The name though is a misnomer. When configured, any user application limited to the size of PSRAM on the device can be downloaded into PSRAM over the USIC channel (U0C0) and immediately executed.

As an example, this application may be a secondary flash loader that can download a larger application and write the latter into program flash.

Data and code fetches are disabled if the global flash read protection is found installed. Initial preparation and generic procedures are described next.

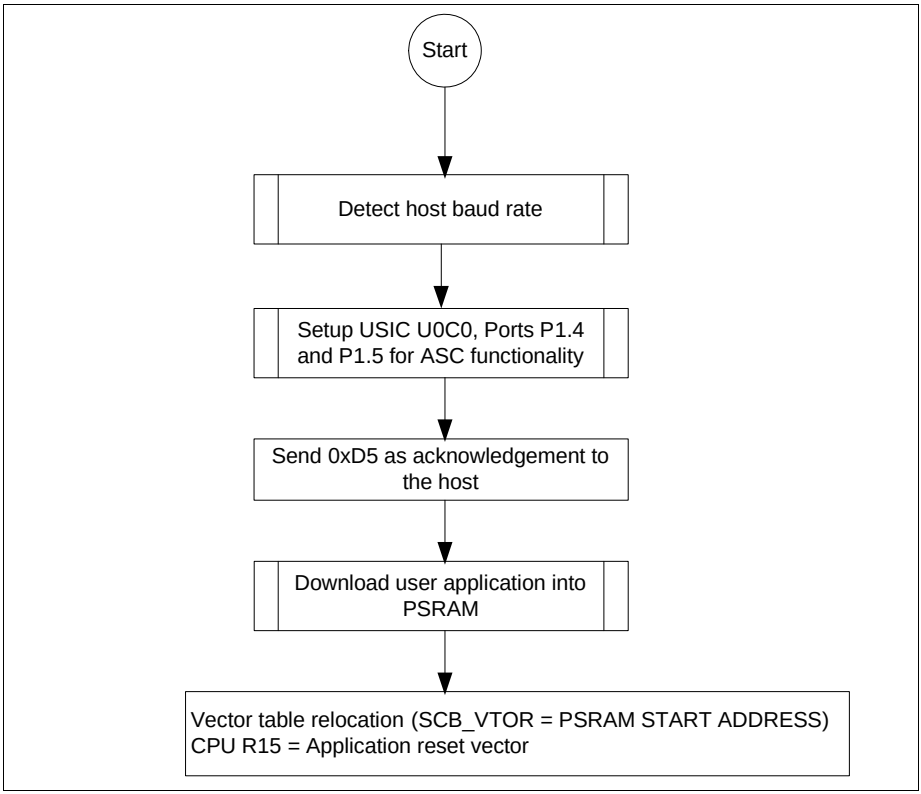


Figure 23-10 ASC BSL Mode Procedures

Full duplex ASC functionality of U0C0 is used for the BSL mode.

Port pins used are P1.4 (U0C0_DX0B) for USIC RX and P1.5 (U0C0_DOUT0) for USIC TX functionality.

The host starts by transmitting a zero byte to help the device detect the baud rate. After the baud rate has been detected by the device, a download protocol specified next helps download of application of any size only limited by the size of PSRAM on the device.

After the baud rate has been detected, SSW transmits an acknowledgement byte $D5_H$ back to the host. It then awaits 4 bytes describing the length of the application from the host. The least significant byte is received first.

If application length is found acceptable by SSW, an OK (1_H) byte is sent to the host following which the latter sends the byte stream of the application. After the byte stream has been received, SSW terminates the protocol by sending a final OK byte and then cedes control to the downloaded application.

If application length is found to be in error (application length greater than device PSRAM size), a N_OK byte is transmitted back to the host and the SSW resumes awaiting the length bytes.

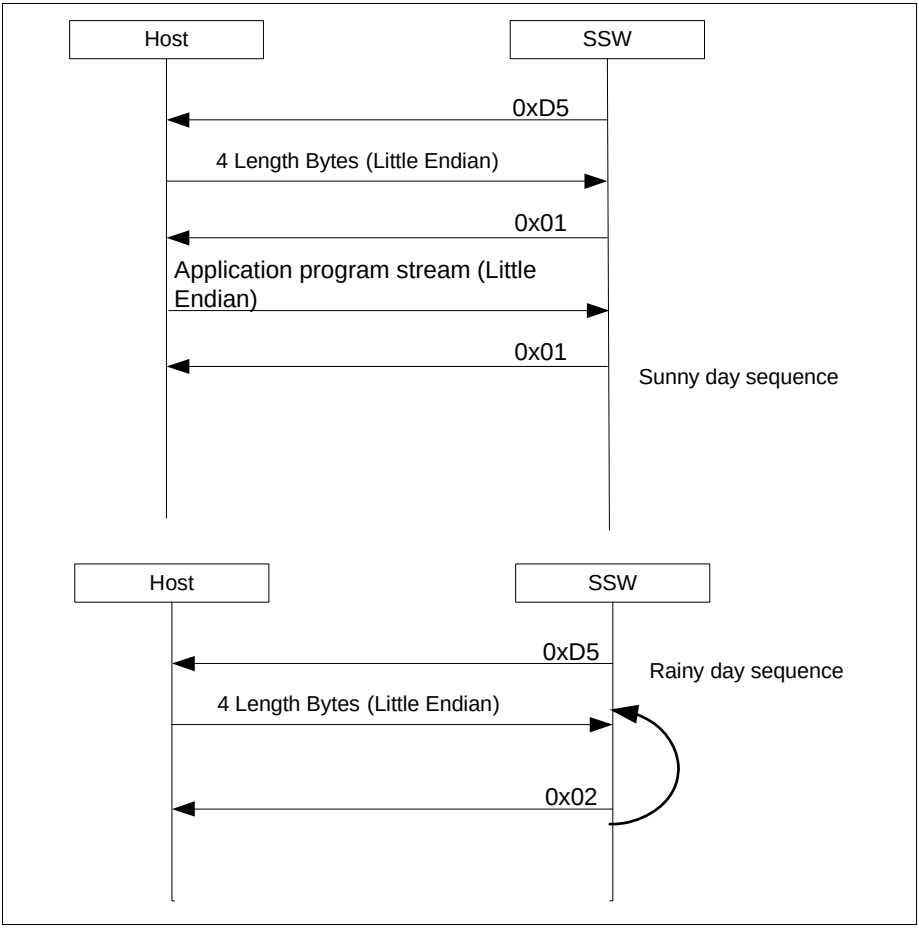


Figure 23-11 Application Download Protocol

User Actions for ASC BSL Mode

User must:

- Configure boot mode pins as described in **Table 23-1**
- Reset the device
- Ensure host sends 00_H to the device (to help device detect the baud rate)

- Ensure host receives $D5_H$ as acknowledgement from device
- Ensure host then sends data length of application and receive a positive acknowledgement
- Ensure host sends the complete application byte stream

23.2.10 CAN BSL Mode

The CAN bootstrap loader mode transfers user application via Node-0 or Node-1 of the CAN module into PSRAM.

A stable external cock is mandatory. SSW uses an iterative algorithm to determine the external clock frequency and switches to it. For transfer rates of 1 Mbps, it must be ensured by the end user that the external clock at least 10 MHz.

A protocol comprising three phases described next leads to user application downloaded into PSRAM. Size of downloaded application is only limited by the size of PSRAM on the device.

Figure 23-12 depicts aforementioned CAN BSL mode procedures.

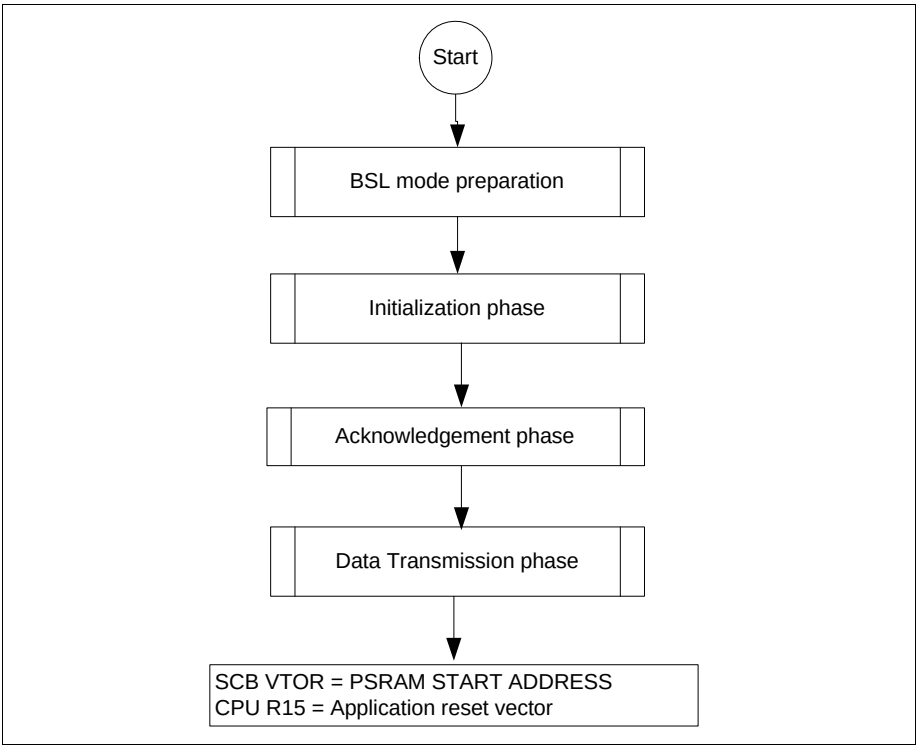


Figure 23-12 CAN BSL Procedures

Initialization Phase

The baud rate of the host is determined.

SSW switches to external clock source. PLL is brought out of power down mode and configured for pre-scalar mode of operation. The VCO is bypassed and powered down.

A standard CAN base frame comprising eight data bytes is transmitted continuously by the host. The 11 bit message ID of the frame (555_H) is used by SSW for baud rate detection. Data bytes 2 and 3 contain the acknowledgement identifier which the SSW must use which acknowledging the completion of initialization phase to the host. The next two bytes (4 and 5) indicate the number of eight byte data frames of user application which must be received by SSW and placed into PSRAM. The last two bytes (6 and 7) indicate the message identifier that would be used by the host while transmitting the user application.

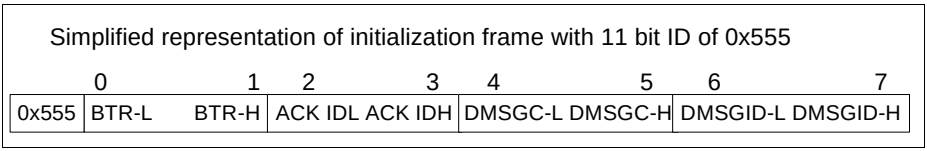


Figure 23-13 Data Field of CAN BSL Initialization Frame

Port pins used are P1.4 and P1.5. If the SSW detects initialization frame on P1.4, it would configure P1.5 for TX functionality. If SSW detects initialization frame on P1.5, it configures P1.4 for TX functionality.

Acknowledgement Phase

An acknowledgement frame is sent to the host indicating completion of initialization phase.

After SSW computes the baud rate of the host and reconfigures the NBTR register of node-0, it waits until the initialization frame is correctly and fully received. SSW signals its intent to use the bus by transmitting a dominant (0) bit in its ACK slot.

After the dominant bit has been transmitted, an acknowledgement frame using the ACK-ID extracted from the initialization frame is sent to the host. If the size of application intended to be downloaded is greater than the size of PSRAM on the device, a negative acknowledgement as depicted below is sent. SSW enters acknowledgement phase again. **Figure 23-14** depicts data part of acknowledgement frame.

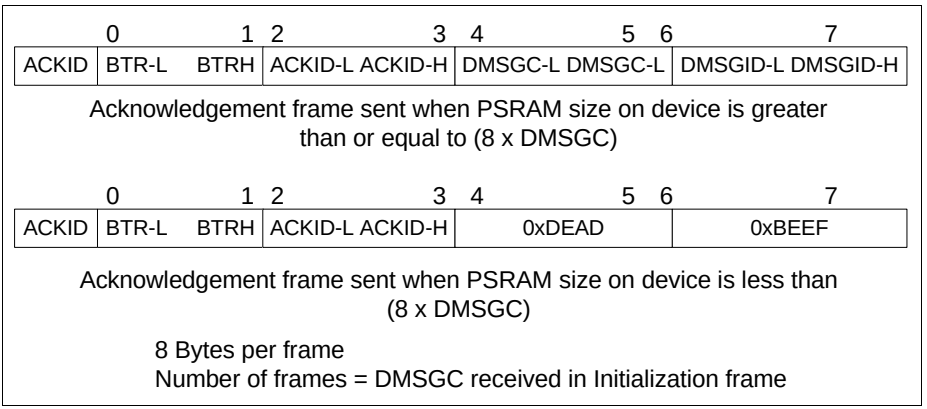


Figure 23-14 CAN Acknowledgement Frame

Data Transmission Phase

Host transmits user application in several CAN frames to the device.

After the SSW transmits the acknowledgement frame, it prepares to receive user application over several CAN frames. Each CAN frame carries eight data bytes and the number of CAN frames is limited to the value retrieved from the DMSGC (Data Message Count) field of the initialization frame. Message identifier is essentially the DMSGID extracted from the initialization frame.

Data received in a frame is placed into PSRAM sequentially.After all of the frames have been received, SSW cedes control to the first user instruction at the start of PSRAM.

User Actions for CAN BSL Mode

User must:

- Configure the boot mode pins as described in **Table 23-1**
- Reset the device
- Ensure CAN host continuously transmits initial frame described in **Table 23-2**
- Ensure host receives acknowledgement frame and transmits the application stream

23.2.11 Boot Mode Index (BMI)

BMI provides a provision for end user to customize boot sequence. A 32 bit BMI word describes a set of activities that must be performed by SSW. Such a BMI word is available in page-1 of User Configuration Block-2 (UCB2-Page1).

BMI word along with associated parameters is known as the BMI string.

The figure below shows details about BMI string composition.

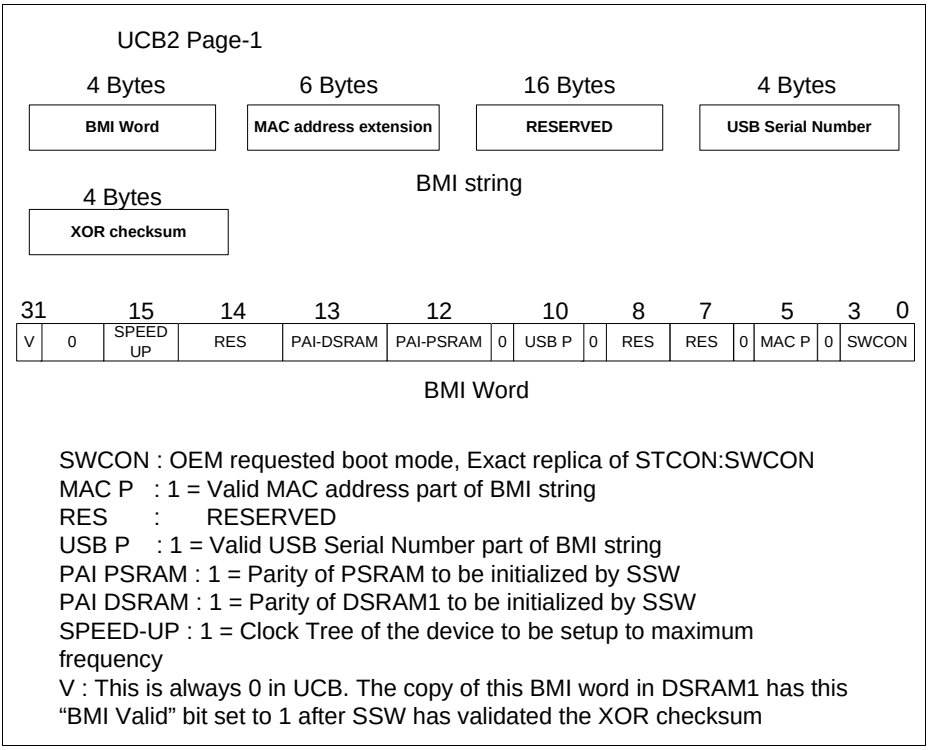


Figure 23-15 BMI String Layout

The BMI string is written into UCB2-Page1 by OEM. SSW upon a reset (and any reset) copies the BMI string into a 64 byte reserved location on DSRAM-1. A 4 byte XOR checksum is appended to the string.

Of the 64 bytes, the BMI string is stored starting from the lowest address with the BMI word stored first followed by the MAC address, IP address and USB serial number.

All elements of the BMI string and the XOR checksum are stored in little-endian format. SSW performs byte wise XOR checksum.

All elements of BMI string are stored linearly without any holes. Following table provides details of the layout.

Table 23-3 BMI String Layout Offset Table

Element	Offset
BMI Word (LSB)	0
MAC (LSB)	4
IP address (LSB)	10
IP address (MSB) for IPv4	13
IP address (MSB) for IPv6	25
USB serial number (LSB)	26
XOR checksum	33

Details of actions taken by SSW are listed in **Figure 23-16**.

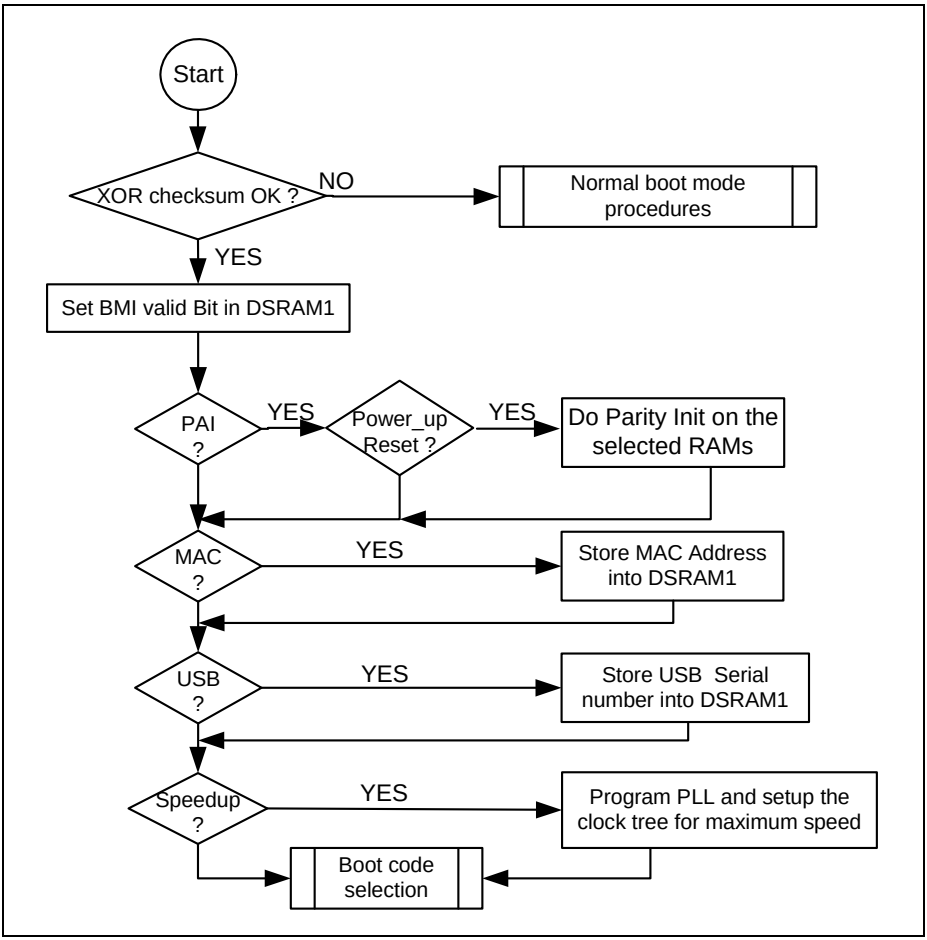


Figure 23-16 BMI Actions

Frequency scaling yields a frequency which will deviate significantly from maximum operational system frequency $f_{SYS(MAX)}$.This is due to the fact that the source clock provided as input to PLL is the internally generated clock f_{OFI} which shows significant variation with respect to its target value. SSW therefore programs PLL in a safe way so that f_{SYS} will be lower than $f_{SYS(MAX)}$.

User Actions for BMI

User must:

- Flash BMI string into the User Configuration Block (UCB2-Page1)
- Configure boot mode pins as described in **Table 23-1**
- Reset the device

23.3 Debug Behavior

This section describes the Halt after reset (HAR), flash protection, debugger access and diagnostics monitor mode features.

23.3.1 Boot Modes and Hardware Debugger Support

The SSW cannot be debugged. It is not possible to halt the CPU after a reset until the SSW has finished its execution. The SSW disables the coresight module thus inhibiting installation of breakpoints and watchpoints. All memory acceses that can otherwise be requested over the debug bus are also inhibited. The coresight module is enabled subject to certain conditions at the time when SSW has finished its activities and is about to cede control to user applications.

Halt After Reset Feature

Halt after reset feature results in halting of the CPU when the first instruction from the user program enters the CPU pipeline. This is implemented only for Normal boot mode, ABM-0 and ABM-1. It is further applicable only for a PORST case. A breakpoint is installed at the address retrieved from reset vector location of the application exception vector table. However for this to happen, the hardware debugger is expected to have registered its request with coresight for halting the CPU.

The CPU simply cannot be halted while the SSW executes. The hardware debugger can only register a halt request with the coresight while the SSW is executing. This request is picked up by SSW towards the end of its execution resulting in installation of breakpoint on the first user application instruction.

Debugger Support After SSW Execution

Debugger support is enabled after SSW execution. But this is subject to the state of flash protection. As long as the user flash is not protected, user programs can be debugged. Should the SSW detect that flash has been password protected, it disables the coresight (ARM debug) interface. Hence debugging is not possible on flash protected devices.

Flash Access After SSW Execution

Special attention must be paid to the table below. Flash access is unconditionally provided for Normal and ABM boot modes regardless of flash protection. CPU can fetch CODE and RO-DATA from the program flash for the two boot modes. Flash access is however only conditionally permitted for BSL and PSRAM boot modes. **Table 23-4** lists the criteria for flash and debugger accesses.

Table 23-4 Flash and Debug Access Policy

HWCON/SWCON	Flash protected?	Flash access	Debugger access
BSL	N	Y	Y
BSL	Y	N	N
Normal/ABM	N	Y	Y
Normal/ABM	Y	Y	N
Boot from PSRAM	N	Y	Y
Boot from PSRAM	Y	N	N

Empty Flash and Debugger Behavior

SSW executes a tight loop upon detecting empty flash. A value of 00000000_H at $0C000004_H$ (Reset vector) indicates empty flash. A hardware debugger can be attached subsequently and appropriate measures taken.

23.3.2 Failures and Handling

It is possible to find out the progress made by SSW during its execution and also the reason for boot failures.

Diagnostics Monitor Mode (DMM)

During its course of execution, SSW has either something normal to trace or an error to report. As access to the Coresight system is disabled during SSW execution, it is imperative that a backchannel mechanism is provisioned for aforesaid monologue from the SSW.

The SCU provides a dedicated register SCU_TSW0 which the SSW can write to. SCU_TSW0 is in fact mirrored by the TCU (Test Control Unit) and the latter's contents can be conveniently scanned out.

This register is known as the TRACE_ERROR_REG and the JTAG instruction for accessing it is the FW_TRACE_ERR (63_H). Please contact your Infineon representative for more details on TCU.

Errors classified as recoverable result in a watchdog reset. Fatal errors require a PORST. The power consumption is reduced to a minimum level.

Figure 23-17 represents the concept pictorially.

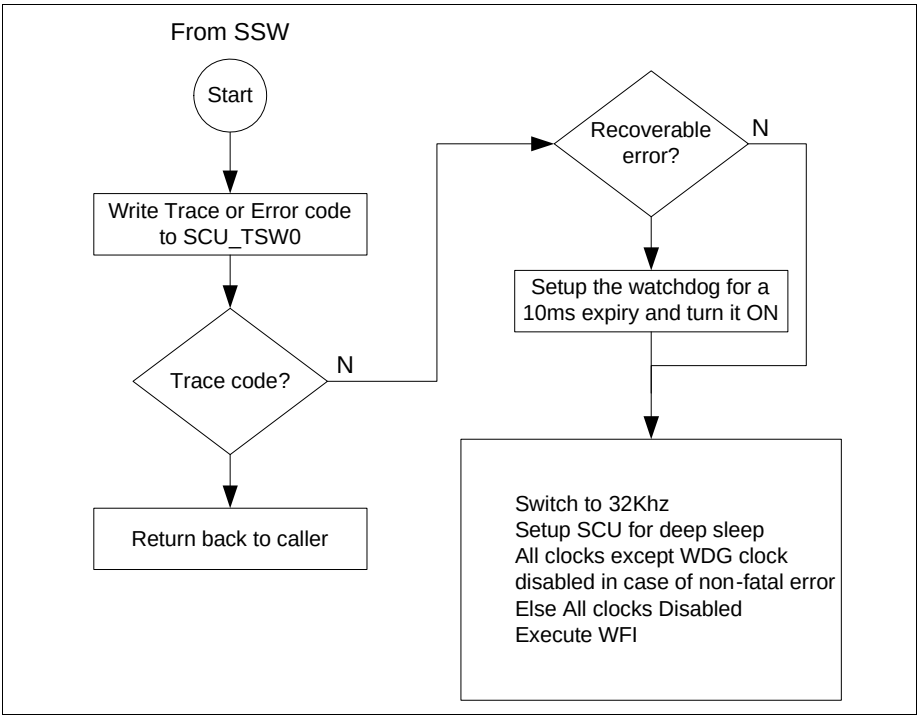


Figure 23-17 Diagnostics Monitor Mode

Encoding of the Trace Word

SSW can write a 32 bit word representing either a trace code or an error code. TSW0[15:0] represents the code. TSW0[17:16] represent the code type.

A code type of 00_B represents invalid code contents, 01_B represents a trace code, 10_B represents a fatal error and 11_B represents a non fatal error.

TSW0[31:18] bits are currently reserved.

List of Errors and Their Classification

Table 23-5 lists errors that lead to SSW launching DMM.

Table 23-5 Error events and codes

Event	Severity	16 bit Code
Flash rampup error	Fatal	1_H
FCS CRC mismatch	Fatal	2_H
Invalid SWCON	Fatal	3_H
MBIST error	Fatal	4_H
PSRAM boot header CRC mismatch	Fatal	5_H
ABM header CRC mismatch	Fatal	6_H
Invalid bootcode	Fatal	7_H
NMI exception	Fatal	8_H
Hardfault exception	Fatal	9_H
Memory management exception	Fatal	A_H
Busfault exception	Fatal	B_H
Usage fault exception	Fatal	C_H
ASC baudrate calculation error	Fatal	D_H
Invalid BMI password error	Fatal	E_H
EVR rampup error	Fatal	F_H
ASC BSL receive error	Fatal	10_H
CAN BSL errors	Fatal	11_H

23.4 Power, Reset and Clock

Startup software is controlling the device initialization only. Users can influence the behavior as described in earlier sections of this chapter. Several registers are changed from their hardware reset value by SSW as detailed below.

23.4.1 Clocking

SSW operates at 24 MHz nominal. The fast internal oscillator provides the clock frequency (f_{OFI}). Frequency scaling is conditionally performed when starting in BMI boot mode based on the BMI information.

23.4.2 Registers Modified by SSW

SSW during its course of execution modifies default register settings of a few registers. They are listed in **Table 23-6**.

Table 23-6 Registers Modified by SSW

Startup mode	Register	Bitfield	Value
All	CPU_CPACR	23:20	1111_B
All	CPU_SHCR	18:16	111_B
All	CPU_CCR	4:3	11_B
All	FCE_CLC	1:0	00_B
All	FCE_CFG0	10:8	000_B
All	FCE_IR0	31:0	Dependent on calculations
All	FCE_CRC0	31:0	Dependent on calculations
All	FCE_RES0	31:0	Dependent on calculations
All	P1_IOCR4	15:11	10010_B
All[1)]	SCU_CGATSTAT0	23 16 9 7 3:2 0	0_B 0_B 0_B 0_B 00_B 0_B
All[1)]	SCU_CGATSTAT1	7 5:3	0_B 000_B
All[1)]	SCU_CGATSTAT2	7 4 1	0_B 0_B 0_B
All	SCU_PRSTAT2	6	0_B
ASC BSL	SCU_PRSTAT0	11	0_B
ASC BSL	U0C0_KSCFG	0	1_B
ASC BSL	U0C0_BRG	14:10	11111_B
ASC BSL	U0C0_BRG	25:16	Dependent on detected baud rate
ASC BSL	U0C0_FDR	15:14	01_B

Table 23-6 Registers Modified by SSW (cont'd)

Startup mode	Register	Bitfield	Value
ASC BSL	U0C0_FDR	9:0	Dependent on detected baud rate
ASC BSL	U0C0_FDR	25:16	Dependent on detected baud rate
ASC BSL	U0C0_DX0CR	2:0 9	001_B 1_B
ASC BSL	U0C0_CCR	0 12:8	1_B 00111_B
ASC BSL	U0C0_SCTR	1 9:8 21:16 27:24	1_B 01_B 000111_B 0111_B
ASC BSL	U0C0_TCSR	11:10 8	01_B 1_B
ASC BSL	U0C0_CCR	9:8 3:0	01_B 0010_B
ASC BSL with Frequency scaling (FS)	SCU_OSCHPCTRL	5:4	00_B
ASC BSL with FS	SCU_PLLCON0	1:0 16	00_B 0_B
ASC BSL with FS	SCU_PLLCON1	22:16 14:8	1_B 1001_B
ASC BSL with FS	SCU_PLLCON2	0	0_B
ASC BSL with FS	SCU_SYSCLKCR	17:16	01_B
ASC BSL with FS	SCU_PLLSTAT	9:0	1110100100_B
CAN BSL	SCU_OSCHPCTRL	5:4 20:16	00_B Dependent on external clock frequency
CAN BSL	SCU_PLLCON0	17:16	00_B
CAN BSL	SCU_SYSCLKCR	17:16	01_B
CAN BSL	SCU_PLLSTAT	9:0	1110010101_B
CAN BSL	SCU_PRSTAT1	4	0_B

Table 23-6 Registers Modified by SSW (cont'd)

Startup mode	Register	Bitfield	Value
CAN BSL	CAN_CLC	1:0	00_B
CAN BSL	CAN_FDR	9:0 15:14 25:16	1111111111_B 01_B Dependent on baud rate
CAN BSL	CAN_NPCR1	2:0	011_B
CAN BSL	CAN_MOAR0/1	28:0	Dependent on data
CAN BSL	CAN_MOAMR0/1	27:0	$1FFFFFFF_H$
CAN BSL	CAN_MOFCR0/1	27:24	1000_B
CAN BSL	CAN_MODATAL0/1, CAN_MODATAH0/1	31:0	Dependent on data
CAN BSL	P1_IOCR4	7:3	10010_B
CAN BSL	CAN_NCR0/1	0	0_B
CAN BSL	CAN_NBTR0/1	5:0,14:8 7:6	Frequency/Baud rate dependent 11_B
CAN BSL	CAN_NFCR0/1	20:19	10_B
CAN BSL	CAN_LIST1/2	24 23:16 15:8	0_B 01_H 01_H
CAN BSL	P1_IOCR4	15:11	10001_B

1) If boot from Flash is requested but the reset vector address is not programmed ($C0000004_H = 0$) then the bits are not changed.

Debug and Trace System

24 Debug and Trace System (DBG)

The XMC4[12]00 Series Microcontrollers provide a large variety of debug, trace and test features. They are implemented with a standard configuration of the ARM CoreSight™ module together with a daisy chained standard TAP controller. Debug and trace functions are integrated into the ARM Cortex-M4. The debug system supports serial wire debug (SWD) and trace functions in addition to standard JTAG debug.

References

[13] Cortex-M4 Technical Reference Manual

[14] CoreSight™ Technology System Design Guide

[15] CoreSight™ Components Technical Reference Manual

[16] ARM Debug Interface v5 Architecture Specification

[17] Embedded Trace Macrocell Architecture Specification

[18] ARMv7-M Architecture Reference Manual

24.1 Overview

The Debug and Trace System implements ARM CoreSight™ debug and trace features with the objective of debugging the entire SoC. The CoreSight™ infrastructure includes a debug subsystem and a trace subsystem. The debug functionality includes processor halt, single-step, processor core register access, Vector Catch, unlimited software break points and full system memory access. The debug function includes a breakpoint unit supporting 2 literal comparators and 6 instruction comparators and a watchpoint unit supporting 4 watchpoints. CoreSight™ enables different trace sources to be enabled into one stream. The unique trace stream, marked with suitable identifiers and timestamps.Trace can be done usinga Serial Wire interface.

Features

The accurate Debug and Trace System provides the following functionality:

- Serial Wire Debug Port (SW-DP)
- JTAG Debug Port (SWJ-DP)
- Flash Patch Breakpoint (FPB)
- Data Watchpoint and Trace (DWT)
- Instrumentation Trace Macrocell (ITM)
- Trace Port Interface Unit (TPIU)
- Halt after reset (HAR)

Note: Please refer to ARM Reference Documentation Cortex-M4-r0p0 for more detailed information on the debug and trace functionality.

Application Mapping

Table 24-1 Debug System available features mapped to functions

SW-DP	Provides Serial Wire Debug, which allows to debug via 2 pins. Instrumentation Trace is provided via a third pin.
SWJ-DP	This debug port provides native JTAG debug capabilities.
FPB	The FPB implements hardware breakpoints and patches code and data from code space to system space.
DWT	Implemented watch points, trigger resources, and system profiling. The DWT contains four comparators that can be configured as a hardware watchpoint, an ETM trigger, a PC sampler event trigger or a data address sampler event trigger, data sampler, interrupt trace and CPU statistics
ITM	Application driven trace source, supports printf style debugging. The ITM generates trace information as packets out of four sources (Software Trace, Hardware Trace, Time Stamping and Global System Time Stamping).
TPIU	The TPIU encodes and provides trace information to the debugger. As port the single wire viewer (TRACESWO)can be used.
HAR	Allows to halt the CPU before application code is entered.

Block Diagram

The Debug and Trace system block diagram is shown in **Figure 24-1**.

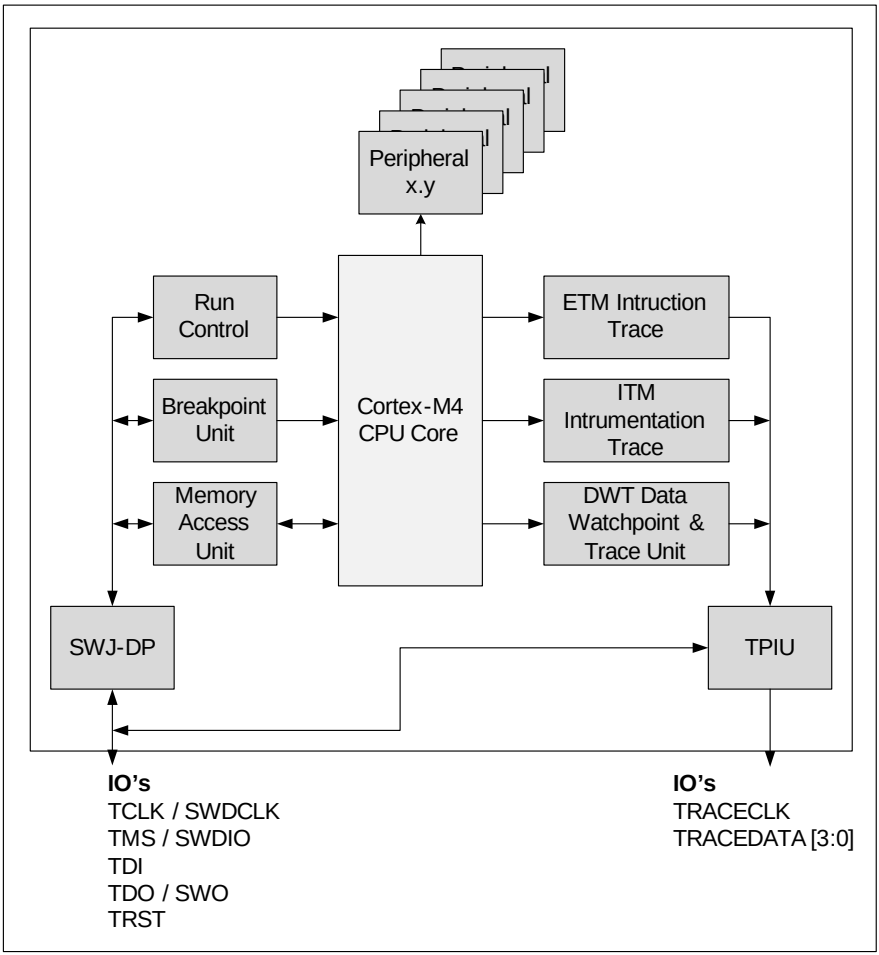


Figure 24-1 Debug and Trace System block diagram

Note: The figure shows ETM, TRACECLK and TRACEDATA[3:0], which are not available for the product.

24.2 Debug System Operation

The Debug System provides general debug options and additional trace functions. Debug options are based on break points and CPU halt. The trace capability supports data access trace.

24.2.1 Flash Patch Breakpoint (FPB)

The FPB implements hardware breakpoints. Six instruction comparators can be configured to generate a breakpoint instruction to the CPU on a match. The original M4 code patch function is not available.

24.2.2 Data Watchpoint and Trace (DWT)

The four DWT comparators can be configured to generate PC sampling packets at defined intervals, PC or Data watch point packets and a Watch point event to halt the CPU. To enable the features, the DWT provides counters with clock cycle, folded instructions, load store unit operation, sleep cycles, clock per instruction and interrupt overhead count.

24.2.3 Instrumentation Trace Macrocell (ITM)

The ITM supports printf style debugging and is an application trace source. The ITM is available to trace application software execution, and allows to emit diagnostic system information. Three different sources are supported to emit trace information as packet, which are software trace, hardware trace and time stamping. The software trace allows software to write directly to ITM stimulus register using printf function. For hardware trace the ITM emits packets generated by the DWT. Relative to packets the ITM emits timestamps, which are generated by a 48-bit counter. The packets emitted by the ITM are output to the trace port interface (TPIU). The TPIU formatter adds some extra packets and then outputs the complete packet sequence to the debugger. The ITM function can be activated by the TRCEN register bit, which is located in the Debug Exception and Monitor control register. ITM data can also be transferred using the Serial Wire interface. The ITM data are the only Trace source data which can be transferred via the Serial Wire interface.

24.2.4 Trace Port Interface Unit (TPIU)

The TPIU collects on-chip trace data from ITM sends this debug data to the external trace capture hardware.

24.3 Power, Reset and Clock

For requirements based on power, reset and clock signaling consult CoreSight™ Technology System Design Guide **[14]** for detailed information. Note, there is no power management implemented for the debug system.

24.3.1 Reset

Reset and implementation is provided according to the ARM reference documentation **[13]**.

24.3.1.1 CoreSight™ resets

The SWJ-DP and SW-DP register are in the power on reset domain. Besides this reset a tool controlled Debug reset can be generated based on a debug register configuration. Activating the reset request in the debug control and status register results in an activation of the CTRL/STAT.CDBGRSTREQ. The reset allows a debugger to reset the debug logic in a CoreSight™ system without affecting the functional behavior of the target system. The Debug logic is reset by system reset, if no tool is registered at the debug system.

System Reset (Warm Reset)

A System or warm reset initializes the majority of the processor, excluding NVIC and debug logic, (FPB, DWT, and ITM). The System reset affects the Debug system only, if the tool is not registered (CTRL/STAT.CDBGPWRUPREQ not set).

SWJ-DP reset

nTRST reset initializes the state of the JTAG SWJ-DP TAP controller. Normal processor operation is not affected.

SW-DP reset

Only the PORESETn reset initializes the SW-DP and the other debug logic.

Normal operation

During normal operation the resets PORESETn SYSRESETn and DAPRESETn are deasserted. If the SWJ-DP or SWDP ports are not in use, nTRST must be tied to 0 or 1.

24.3.1.2 Serial Wire interface driven system reset

The CTRL/STAT.SYSRESETREQ allows to reset the CPU core in Serial Wire interface mode. The CTRL/STAT.SYSRESETREQ is asserted when the

CTRL/STAT.SYSRESETREQ bit of the Application Interrupt and Reset Control register is set. This causes a reset, intended to force a large system reset of all major components, except for debug logic. **[15]**.

24.4 Initialization and System Dependencies

This chapter provides information about Debug access and Flash protection, Halt After Reset sequences, Halting Debug and Peripheral suspend, Timestamping for Trace and the available tool interfaces and how to enable the tool interface. Additionally the ID Codes and the ROM Table is presented, including the values a debug tool uses to identify the device and available debug functionality. The last section shows the JTAG Debug instruction code definition.

24.4.1 Debug accesses and Flash protection

The Flash has integral measures to protect its content from unauthorized access and modification, see the section Read and Write Protection in the PMU chapter and startup chapter. Special care is taken, that the debugger can't bypass this protection. Because of this, per default and after a system reset the debug interface is disabled. Depending on the boot scenario and the Flash protection setup, the Startup Software enables the debug interface. If it is left disabled, the user can define a protocol, e.g. a password-protected unlock sequence via an SPI port, to enable the debug interface.

24.4.2 Halt after reset

The XMC4[12]00 product supports two different possibilities of halt after reset. One after a power on reset ("Power-on Reset"), which is called Cold Reset Halt situation or Halt after reset (HAR) and the second one after a system reset, which is called Warm Reset Halt situation. For a HAR the debug tool has to register during the start up software (SSW) execution time. The SSW halts at the end of SSW on a successful tool registration, before application code is entered. The Warm Reset Halt is based on break point setting on the very first application code line and is entered by a system reset.

For security reasons it is required to prevent a debug access to the processor before and while the boot firmware code from ROM (SSW) is being executed. A bit DAPSA, (DAP has system access) in the SCU is implemented, allowing the access from CoreSight™ debug system to the processor core. The default value of this bit is disabled debug access. The register is reset by System Reset. The System Reset disables the debug access each time SSW is being executed. At the end of the SSW the DAPSA is enabled always (independent of any other register setting or signaling value), to allow debug access to the CPU. A tool accessing the SoC during the SSW execution time reads back a zero and a write is going to a virtual, none existing address.

In a HAR situation the system always comes from a "Power-on Reset". A tool can register for a HAR by sending a pattern enabling the CTRL/STAT.CDBGPWRUPREQ

and the DHCSR.C_DEBUGEN **[18]** register inside the CoreSight™ module. The registration has to be done after the "Power-on Reset" in a time interval smaller then the time the SSW is executed. This registration time is called tssw and the bits must be set before tssw is expired. The timing value for tssw needs to be handled by the debug tool software (no hardware timer available) and informs the tool software during a Cold Reset about the time frame available to set the HAR conditions.

The following figure (**Figure** HAR - Halt After Reset) shows the software flow based on the modules participating to the HAR.

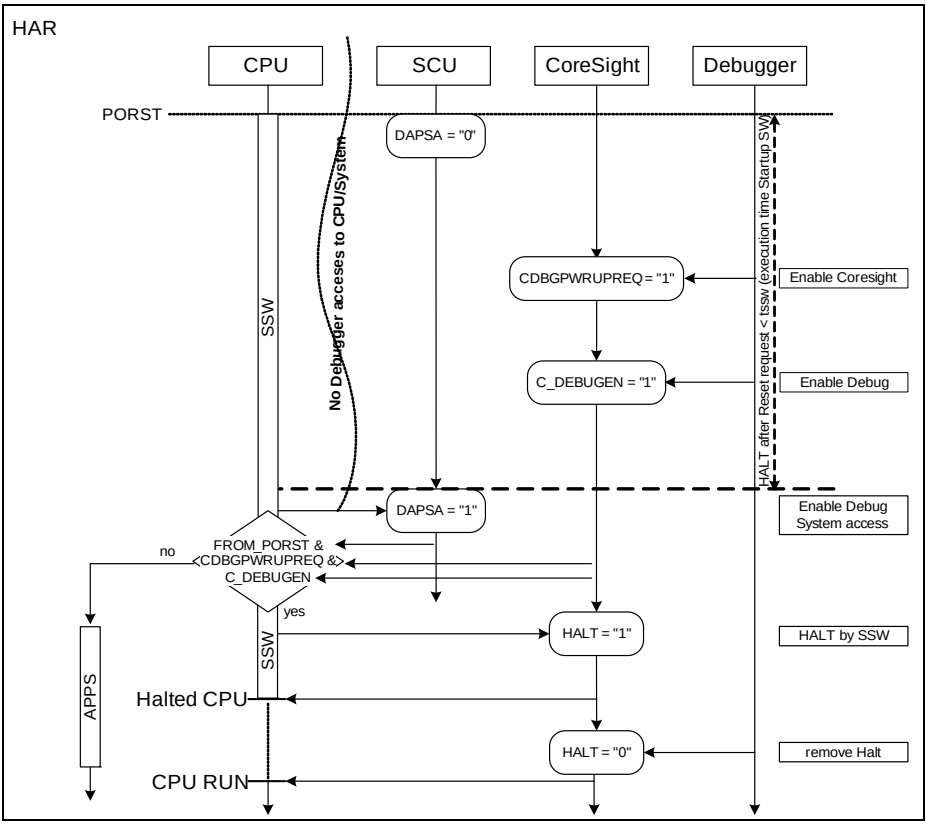


Figure 24-2 HAR - Halt After Reset

A Halt after system reset (Warm Reset) can be achieved by programming a break point at the first instruction of the application code. Before a Warm Reset the CTRL/STAT.CDBGPWRUPREQ and the DHCSR.C_DEBUGEN setting has to be

ensured. After a system reset, the HAR situation is not considered, as the reset is not coming from "Power-on Reset".

Note: The CTRL/STAT.*CDBGPWRUPREQ and* DHCSR.*C_DEBUGEN does not have to be set after a system reset, if they have already been set before.*

A tool hot plug condition allows to debug the system starting with a tool registration (setting CTRL/STAT.CDBGPWRUPREQ register) and a debug system enable (setting the DHCSR.C_DEBUGEN register). Afterwards break points can be set or the CPU can directly by HALTED by an enable of C_HALT.

The following Figure (**Figure** Hot Plug and Warm Reset) illustrates the debug tool HOT PLUG situation or the Halt after Warm Reset (system reset) and how to proceed to come to a Halt situation.

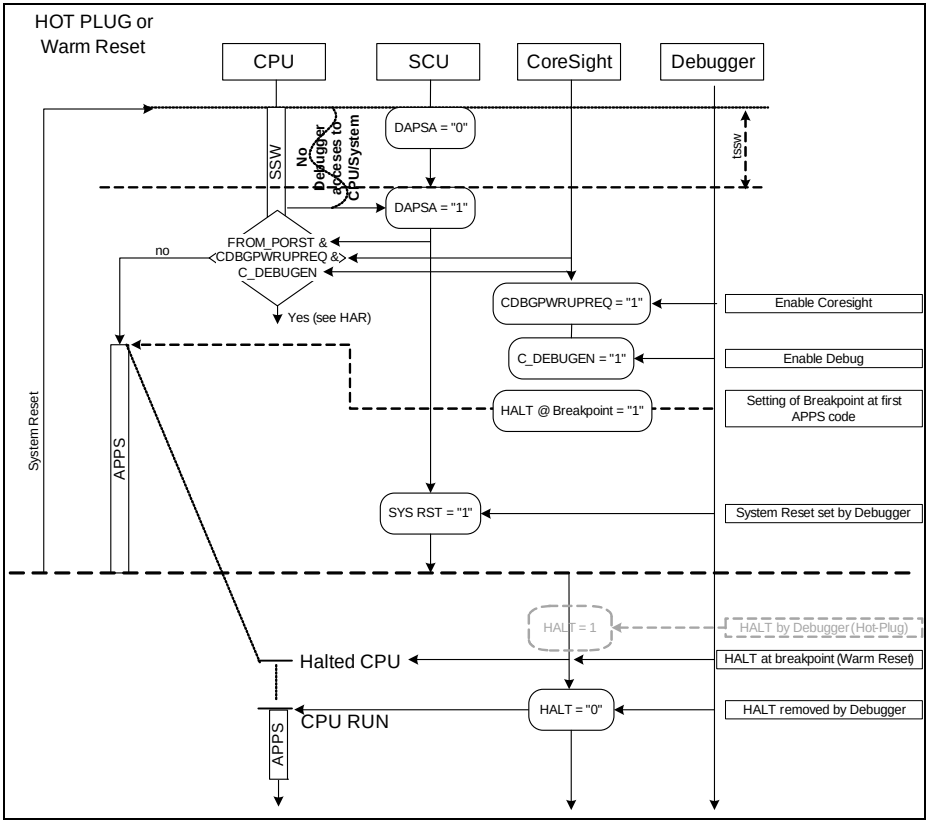


Figure 24-3 HOT PLUG or Warm Reset

24.4.3 Halting Debug and Peripheral Suspend

If the program execution of the CPU is stopped by the debugger, e.g. with a breakpoint, it is possible to suspend the peripherals as well. This allows to debug critical states of the whole microcontroller. It is particularly useful, e.g. to suspend the Watchdog Timer as it can't be serviced by a halted CPU.

In other cases it is important to keep some peripherals running, e.g. a PWM or a CAN node, to avoid system errors or even critical damage to the application. Because of this, the peripherals allow to configure how they behave when the CPU enters the halting debug mode.

It can be decided at the peripheral to support a Hard Suspend or a Soft Suspend. At a Hard Suspend situation the clock at the peripheral is switched off immediately, without waiting on acknowledge from the module. At a soft suspend the peripheral can decide when to suspend, usually at the end of the actual active transfer.

A Watchdog timer is only running when the suspend bus is not active. This is particularly useful as it can't be serviced by a halted CPU. A configuration option is available, which allows to enable the Watchdog timer also during suspend. This allows to debug Watchdog behavior, if a debugger is connected.

The user has to ensure, that always only those peripherals are sensitive to suspend, which are intended to be. To address this, each peripheral supporting suspend does have an enable register which allows to enable the suspend feature. The following table (**Table 24-2**) shows the peripherals, supporting or not supporting peripheral suspend or detailed information on the peripheral suspend behavior during soft suspend can be found at the respective peripheral chapter.

Table 24-2 Peripheral Suspend Support

Peripheral	Supported	Default mode	Hard Suspend	Soft Suspend	Suspend Reset
RTC	no	- - -	- - -	- - -	- - -
WDT[1)]	yes	active	yes	no	system reset
LEDTS	yes	not active	yes	no	debug reset
SDMMC	no	- - -	- - -	- - -	- - -
EBU	no	- - -	- - -	- - -	- - -
ETH	no	- - -	- - -	- - -	- - -
USB	no	- - -	- - -	- - -	- - -
USIC	yes	not active	no	yes	debug reset
MultiCAN	yes	not active	yes	yes	debug reset
VADC	yes	not active	yes	yes	debug reset

Table 24-2 Peripheral Suspend Support

Peripheral	Supported	Default mode	Hard Suspend	Soft Suspend	Suspend Reset
DAC	no	- - -	- - -	- - -	- - -
CCU4[1)]	yes	not active	yes	yes	system reset
CCU8[1)]	yes	not active	yes	yes	system reset
POSIF[1)]	yes	not active	yes	yes	system reset
HRPWM[1)]	yes	not active	yes	yes	system reset

1) A system reset results in a suspend configuration loss. If it is required to have suspend configuration available after system reset, a HW breakpoint has to be set at the first instruction of use code and reconfiguration of suspend behavior at the peripheral has to be performed again.

24.4.4 Timestamping

A 48-bit timestamping capability is required by the debug system in order to get accurate correlation between trace data from ITM and DWT. This is also named global timestamping. Timestamp packets encode timestamp information, generic control and synchronization information. The Timestamp counter is a free running global counter.

A synchronization packet is a timestamp packet control. It is emitted at each DWT trigger (DWT must be configured to trigger the ITM).

24.4.5 Debug tool interface access (SWJ-DP)

Debug capabilities can be accessed by a debug tool via Serial Wire (SW) or JTAG interface (JTAG - Debug Port). By default, the JTAG interface is active. The User might switch to SW interface as the full JTAG pins are not available to the user. To enable SW interface a dedicated JTAG sequence on TMS/SWDIO and TCK/SWCLK is required to to switch to the Serial Wire Debug interface. A successful sequence disables JTAG interface and enables SW interface.

The sequences to do this are described in **Section 24.4.5.1** and **Section 24.4.5.2**

24.4.5.1 Switch from JTAG to SWD

The sequence for switching from JTAG to SWD is:

- Send 50 or more TCK cycles with TMS = 1
- Send the 16-bit sequence on TMS = 1110011110011110 (0xE79E LSB first)
- Send 50 or more TCK cycles with TMS = 1

24.4.5.2 Switch from SWD to JTAG

The sequence for switching from SWD to JTAG is:

- Send 50 or more TCK cycles with TMS = 1
- Send the 16-bit sequence on TMS = 1110011100111100 (0xE73C LSB first)
- Send 50 or more TCK cycles with TMS = 1

24.4.6 ID Codes

Available ID Codes are used by a debug tool to identify the available debug components during tool setup.

Table 24-3 ARM CoreSight™ Component ID codes

ID	Value[1)]	Description
JTAG IDCODE	XXXX X083$_H$	The TAP JTAG IDCODE
SWJ-DP	4BA0 0477$_H$	The ARM JTAG ID
SW_DP	2BA0 1477$_H$	The ARM SW-DP ID

1) For “X” values please refer to Data Sheet

24.4.7 ROM Table

To identify Infineon as manufacturer and XMC4[12]00 as device, the ROM table has to be read out.

Table 24-4 Perpheral ID Values of XMC4[12]00 ROM Table

Name	Offset	Value[1)]	Bits	Reference	Infineon JTAG ID Code
PID0	FE0$_H$	XXXXXXXX$_B$	[7:0]	Part Number [7:0]	JTAG IDCODE [19:12]
PID1	FE4$_H$	0001XXXX$_B$	[7:4] [3:0]	JEP106 ID code [3:0] Part Number [11:8]	JTAG IDCODE [4:1] JTAG IDCODE [23:20]
PID2	FE8$_H$	XXXX1100$_B$	[7:4] [3] [2:0]	Revision JEDEC assigned ID fields JEP106 ID code [6:4]	JTAG IDCODE [31:28] ‘1’ JTAG IDCODE [7:5]
PID3	FEC$_H$	00000000$_B$	[7:4] [3:0]	RevAnd, minor revision field Customer-modified block (if non-zero)	
PID4	FD0$_H$	00000000$_B$	[7:4] [3:0]	4KB count JEP106 continuation code	4KB count JTAG IDCODE [11:8]

1) For “X” values please refer to Data Sheet

24.4.8 JTAG debug port

A standard JTAG IEEE1149 Boundary-Scan statemachine is implemented. It includes all mandatory instructions (SAMPLE/PRELOAD, EXTEST) and also some optional instructions (IDCODE, CLAMP, HIGHZ), and some custom/optional instructions are implemented. The optional RUNBIST instruction is not used and will be treated as BYPASS. No USERCODE-instruction is implemented, it will show only 0 values

Table 24-5 JTAG INSTRUCTIONS

Opcode	Range	Type	Instruction
0000 0000	00H	Reserved	
0000 0001	01H - 08H (8 instr.)	IEEE1149 Boundary-Scan	INTEST
0000 0010			SAMPLE/PRELOAD
0000 0011			RUNBIST
0000 0100			IDCODE
0000 0101			USERCODE
0000 0110			CLAMP
0000 0111			EXTEST
0000 1001- 0000 1111	Reserved		
0001 0001 - 0100 1111	11H - 4FH 63instr.		
1111 1111	FFH	IEEE 1149.1	BYPASS

JTAG Instruction Definition

BYPASS

The BYPASS instruction bypasses all serial register path, which are accessed over the JTAG TAP port. In the Capture-DR state the rising edge of the TCK clock sets the bypass register to zero. The bypass instruction is primarily used to allow a shorter access path to cascaded devices when the JTAG interface connects a number of devices in series. All unused instructions must connect the TAP controller bypass register output to the Test Access Port output TDO. The BYPASS instruction has no effect on the operation of the device.

24.5 Debug System Registers

For CoreSight™ register overview and detailed register definitions, please refer to ARM documentation CoreSight™ Components Technical Reference Manual **[15]** and ARMv7-M Architecture Reference Manual **[18]**.

24.6 Debug and Trace Signals

XMC4[12]00 MC Product family provides debug capability using ARM CoreSight™ Debug port SWJ-DP. SWJ-DP includes two debug interfaces namely JTAG Debug Port (JTAG-DP) and Serial Wire Debug Port (SW-DP).

The JTAG-DP interface has 4 (without Reset pin TRST for low pin package) or 5 Pins, see **Table 24-6**.

The serial wire Debug Port has 2 (Clock + Bidirectional data) or 3 pins (Clock + Bidirectional data + Asynchronous Trace output). They are overlaid on the JTAG-DP pins (TCK, TMS and TDO) for efficient use of package pins, see **Table 24-7**.

Sub chapters below additionally describe pull resistors to the IO and the suggested debug connector pin assignment.

Table 24-6 JTAG Debug signal description

Signal	Direction	Function
TCK	I	JTAG Test Clock. This pin is the clock for debug module when running in JTAG debug mode.
TMS	I	JTAG Test Mode Select. The TMS pin selects the next state in the TAP state machine.
TDI	I	JTAG Test Data In. This is the serial data input for the shift register
TDO	O	JTAG Test Data Output. This is the serial data output from the shift register. Data is shifted out of the device on the negative edge of the TCK signal
$\overline{\text{TRST}}$	I	JTAG Test reset.

Table 24-7 Serial Wire Debug signal description

Signal	Direction	Function
SWDCLK	I	Serial Wire Clock. This pin is the clock for debug module when running in Serial Wire debug mode.
SWDIO	I / O	Serial Wire debug data IO. Used by an external debug tool to communicate with and control the Cortex-M4 CPU.
SWO	O	Serial Wire Output. The SWO pin provides data from the ITM and/or ETM for an external debug tool to evaluate the instrumentation trace.

24.6.1 Internal pull-up and pull-down on JTAG pins

It is a requirement to ensure none floating JTAG input pins, as they are directly connected to flip-flops controlling the debug function. To avoid any uncontrolled I/O voltage levels internal pull-up and pull-downs on JTAG input pins are provided.

- TRST: Internal pull-down
- TMS/SWDIO: Internal pull-up
- TCK/SWCLK: Internal pull-down

24.6.2 Debug Connector

The suggested connector is the Cortex Debug and ETM connector, which is a 20-pin connector. The connector supports JTAG debug, Serial-Wire debug, Serial Wire viewer (via SWO connection when Serial Wire debug mode is used) and instruction trace operations. The following **Figure 24-4** shows the 20-pin connector for debug and trace.

There may be systems, which required HW to detect debugger presence. This can be enabled by using the GNDDetect pin of the Debug and Trace connector. The pin is driven low by the tool, when the tool connector is plugged into the connector at the PCB. GNDDetect for tool detection requires to have a weak pull-up on the PCB, connected to this pin. At the connector it is suggested not to have the KEY pin available as the KEY is used to identify the correct connector plug-in.

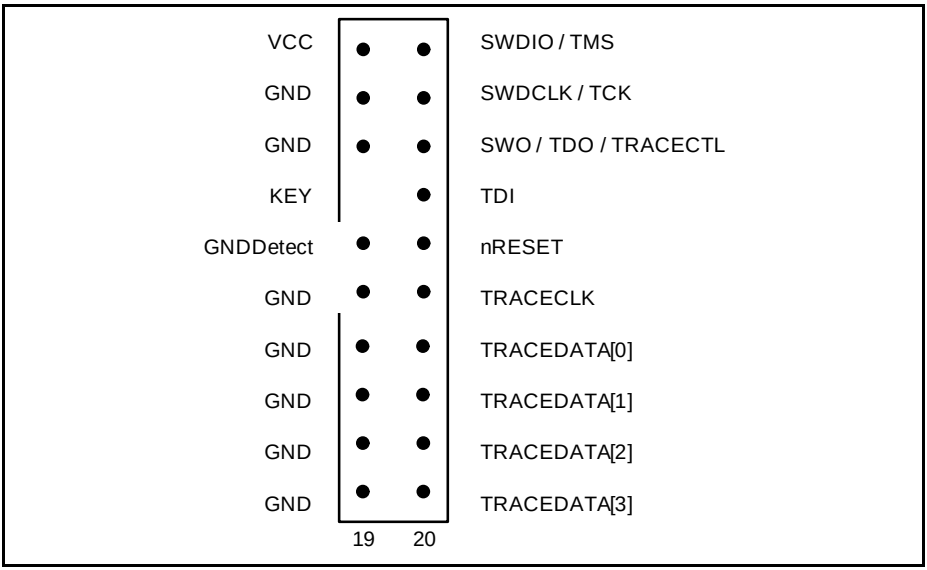


Figure 24-4 Debug and Trace connector

Note: The figure shows TRACECLK and TRACEDATA[3:0] ports, which are not available for the product, as ETM function is not available.

Lists of Figures and Tables

List of Figures

List of Figures

List of Figures

List of Figures

List of Figures

List of Figures

List of Figures

List of Figures

List of Figures

List of Figures

List of Figures

List of Figures

List of Figures

List of Figures

List of Tables

List of Tables

List of Tables

List of Tables

List of Tables

List of Tables

List of Tables

w w w . i n f i n e o n . c o m

Published by Infineon Technologies AG