PSOC™ Control C1 architecture

Reference manual

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 PSC1M2x/PSC1M3x series functional units and their interaction.

The manual describes the functionality of the superset device of the PSC1Mx microcontroller series. For the available functionality (features) of a specific PSC1Mx derivative (derivative device), please refer to the respective Datasheet. For simplicity, the various device types are referenced by the collective term PSC1Mx throughout this manual.

PSC1Mx Family User Documentation

The set of user documentation includes:

- **Reference Manual**
  - describes the functionality of the superset device.
- **Datasheets**
  - 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 Datasheets. 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/PSC1Mx to get access to the latest versions of those documents.

Related Documentations

The following documents are referenced:

- Arm® Cortex® M0
  - Technical Reference Manual
  - User Guide, Reference Material
- Arm®v6-M Architecture Reference Manual
- AMBA® 3 AHB-Lite Protocol Specification
- AMBA® 3 APB Protocol Specification

Copyright Notice

- Portions of CPU chapter Copyright © 2009, 2010 by Arm®, Ltd. All rights reserved. Used with permission.

Text Conventions

This document uses the following naming conventions:

- Functional units of the PSC1Mx 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 in general 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** = 1000 characters/bits per second
  - **MBaud, Mbit** = 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
PV (SV), U	Access permitted in Privileged (Supervisor) Mode. ***Note**: Arm® Cortex® M0 processor does not support different privilege levels. Only Privileged (Supervisor) Mode is supported in PSC1Mx Family. Symbol "U" and Symbol "PV" can be used to represent the access permitted in this mode.*
BP	Indicates that this register can only be access when the bit protection is disabled. See detailed description of Bit Protection Scheme in the Memory Organisation chapter.
32	Only 32-bit word accesses are permitted to this register/address range.
NC	No change, indicated register is not changed.
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 have to be written with 0 if the bit field is defined as rw.

These bit fields are reserved. The detailed description of these bit fields can be found in the register descriptions.

Abbreviations and Acronyms

The following acronyms and terms are used in this document:

ACMP	Analog Comparator
AHB	Advanced High-performance Bus
AMBA®	Advanced Microcontroller Bus Architecture
ANACTRL	Analog Control Unit
APB	Advanced Peripheral Bus
ASC	Asynchronous Serial Channel
BCCU	Brightness and Colour Control Unit
BMI	Boot Mode Index
CMSIS	Cortex® Microcontroller Software Interface Standard
CPU	Central Processing Unit
CCU4	Capture Compare Unit 4
CCU8	Capture Compare Unit 8
CRC	Cyclic Redundancy Code
DCO	Digitally Controlled Oscillator
ECC	Error Correction Code
ERU	Event Request Unit

EVR	Embedded Voltage Regulator
FPU	Floating Point Unit
GPIO	General Purpose Input/Output
HMI	Human-Machine Interface
IIC	Inter Integrated Circuit (also known as I2C)
IIS	Inter-IC Sound Interface
I/O	Input/Output
JTAG	Joint Test Action Group = IEEE1149.1
LED	Light Emitting Diode
MSB	Most Significant Bit
NC	Not Connected
NMI	Non-Maskable Interrupt
NVIC	Nested Vectored Interrupt Controller
ORC	Out of Range Comparator
PAU	Peripheral Access Unit
POSIF	Position Interface
PRNG	Pseudo Random Number Generator
ROM	Read-Only Memory
RAM	Random Access Memory
RTC	Real Time Clock
SCU	System Control Unit
SFR	Special Function Register
SHS	Sample and Hold Sequencer
SPI	Serial Peripheral Interface
SRAM	Static RAM
SR	Service Request
SSC	Synchronous Serial Channel
SSW	Start-up Software
TSE	Temperature Sensor
UART	Universal Asynchronous Receiver Transmitter
USIC	Universal Serial Interface Channel
VADC	Versatile Analog-to-Digital Converter
WDT	Watchdog Timer

Table of contents

About this document . . . 1
Table of contents . . . 5
1 Introduction . . . 28
1.1 Overview . . . 28
1.1.1 Architecture . . . 30
1.1.2 Device Overview . . . 30
1.2 CPU Subystem . . . 31
1.3 On-chip Memories . . . 32
1.4 Communication Peripherals . . . 32
1.5 Analog Front-End Peripherals . . . 33
1.6 Industrial Control Peripherals . . . 33
1.7 On-chip Debug Support . . . 34
2 Central Processing Unit (CPU) . . . 35
2.1 Overview . . . 35
2.1.1 Features . . . 36
2.1.2 Block Diagram . . . 36
2.2 Programmers Model . . . 38
2.2.1 Processor Mode . . . 38
2.2.2 Stacks . . . 38
2.2.3 Core Registers . . . 39
2.2.3.1 General-purpose registers . . . 40
2.2.3.2 Stack Pointer . . . 40
2.2.3.3 Link Register . . . 41
2.2.3.4 Program Counter . . . 41
2.2.3.5 Program Status Register . . . 42
2.2.3.6 Application Program Status Register . . . 42
2.2.3.7 Interrupt Program Status Register . . . 43
2.2.3.8 Execution Program Status Register . . . 44
2.2.3.9 Priority Mask Register . . . 45
2.2.3.10 CONTROL Register . . . 45
2.2.4 Exceptions and Interrupts . . . 46
2.2.5 Data Types . . . 46
2.2.6 The Cortex® Microcontroller Software Interface Standard . . . 46
2.2.7 CMSIS Functions . . . 47
2.3 Memory Model . . . 48
2.3.1 Memory Regions, Types and Attributes . . . 49
2.3.2 Memory System Ordering of Memory Accesses . . . 49
2.3.3 Behavior of Memory Accesses . . . 50
2.3.4 Software Ordering of Memory Accesses . . . 50

Table of contents

2.3.5 Memory Endianness . . . 51
2.3.5.1 Little-endian format . . . 51
2.4 Instruction Set . . . 51
2.4.1 Intrinsic Functions . . . 53
2.5 Exception Model . . . 53
2.5.1 Exception States . . . 54
2.5.2 Exception Types . . . 54
2.5.3 Exception Handlers . . . 55
2.5.4 Vector Table . . . 55
2.5.4.1 Vector Table Remap . . . 56
2.5.5 Exception Priorities . . . 57
2.5.6 Exception Entry and Return . . . 58
2.5.6.1 Exception entry . . . 59
2.5.6.2 Exception return . . . 59
2.6 Fault Handling . . . 60
2.6.1 Lockup . . . 60
2.7 Power Management . . . 61
2.7.1 Entering Sleep Mode . . . 61
2.7.2 Wakeup from Sleep Mode . . . 61
2.7.3 Power Management Programming Hints . . . 62
2.8 Private Peripherals . . . 62
2.8.1 About the Private Peripherals . . . 62
2.8.2 System control block . . . 62
2.8.2.1 System control block usage hints and tips . . . 62
2.8.3 System timer, SysTick . . . 63
2.8.3.1 SysTick usage hints and tips . . . 63
2.9 PPB Registers . . . 63

3 Bus System . . . 64
3.1 Bus Interfaces . . . 64

4 Service Request Processing . . . 65
4.1 Overview . . . 65
4.1.1 Features . . . 65
4.1.2 Block Diagram . . . 65
4.2 Service Request Distribution . . . 66

5 Interrupt Subsystem . . . 68
5.1 Nested Vectored Interrupt Controller (NVIC) . . . 68
5.1.1 Features . . . 68
5.1.2 Interrupt Node Assignment . . . 68
5.1.3 Interrupt Signal Generation . . . 70
5.1.4 NVIC design hints and tips . . . 70
5.1.5 Accessing CPU Registers using CMSIS . . . 71

Table of contents

5.1.6 Interrupt Priority . . . 71
5.1.7 Interrupt Latency . . . 72
5.2 General Module Interrupt Structure . . . 73
5.3 Registers . . . 74
5.4 Interrupt Request Source Overview . . . 74

6 Event Request Unit (ERU) . . . 88
6.1 Features . . . 88
6.2 Overview . . . 88
6.3 Event Request Select Unit (ERS) . . . 89
6.4 Event Trigger Logic (ETLx) . . . 89
6.5 Cross Connect Matrix . . . 90
6.6 Output Gating Unit (OGUy) . . . 91
6.7 Power, Reset and Clock . . . 93
6.8 Initialization and System Dependencies . . . 93
6.9 Registers . . . 94
6.10 Interconnects . . . 95
6.10.1 ERU0 Connections . . . 95
6.10.2 ERU1 Connections . . . 100

7 MATH Coprocessor (MATH) . . . 105
7.1 Overview . . . 105
7.1.1 Features . . . 105
7.1.2 Block Diagram . . . 105
7.2 Divider Unit (DIV) . . . 106
7.2.1 Features . . . 106
7.2.2 Division Operation . . . 106
7.2.2.1 Start Mode Selection . . . 107
7.2.2.2 Error Handling . . . 107
7.2.3 Operand/Result Pre-/Post-Processing . . . 109
7.3 CORDIC Coprocessor . . . 109
7.3.1 Overview . . . 109
7.3.1.1 Features . . . 110
7.3.1.2 Block Diagram . . . 111
7.3.2 Functional Overview . . . 111
7.3.2.1 Operation of the CORDIC Coprocessor . . . 112
7.3.2.2 Normalized Result Data . . . 112
7.3.3 CORDIC Coprocessor Operating Modes . . . 114
7.3.3.1 Domains of Convergence . . . 116
7.3.3.2 Overflow Considerations . . . 116
7.3.4 CORDIC Coprocessor Data Format . . . 117
7.3.5 Accuracy of CORDIC Coprocessor . . . 118
7.3.6 Performance of CORDIC Coprocessor . . . 120

Table of contents

7.3.7 CORDIC Coprocessor Look-Up Tables . . . 120
7.3.7.1 Arctangent and Hyperbolic Arctangent Look-Up Tables . . . 120
7.3.7.2 Linear Function Emulated Look-Up Table . . . 121
7.4 Global Functions . . . 122
7.4.1 Result Chaining . . . 122
7.4.1.1 Result Chaining when Start Mode = 0 . . . 122
7.4.1.2 Handling Busy Flags when Result Chaining is Enabled . . . 123
7.5 Service Request Generation . . . 123
7.6 Debug Behaviour . . . 124
7.7 Power, Reset and Clock . . . 125
7.8 Registers . . . 125
7.9 Interconnects . . . 126

8 Memory Organization . . . 127
8.1 Overview . . . 127
8.1.1 Features . . . 127
8.1.2 Cortex® -M0 Address Space . . . 127
8.2 Memory Regions . . . 128
8.3 Memory Map . . . 128
8.4 Memory Access . . . 132
8.4.1 Flash Memory Access . . . 132
8.4.2 SRAM Access . . . 132
8.4.3 ROM Access . . . 132
8.5 Memory Protection Strategy . . . 132
8.5.1 Intellectual Property (IP) Protection . . . 133
8.5.1.1 Blocking of Unauthorized External Access . . . 133
8.5.2 Memory Access Protection during Run time . . . 133
8.5.2.1 Bit Protection Scheme . . . 133
8.5.2.1.1 SCU_PASSWD . . . 134
8.5.2.2 Peripheral Privilege Access Control . . . 134
8.6 Service Request Generation . . . 135
8.7 Debug Behaviour . . . 135
8.8 Power, Reset and Clock . . . 135
8.9 Initialization and System Dependencies . . . 135

9 Prefetch Unit (PFU) . . . 136
9.1 Overview . . . 136
9.1.1 Block Diagram . . . 136
9.2 Operation Mode . . . 136

10 Flash Architecture . . . 137
10.1 Overview . . . 137
10.1.1 Features . . . 137
10.2 Definitions . . . 137

10.2.1 Logical and Physical States . . . 137
10.2.2 Data Portions . . . 138
10.2.3 Address Types . . . 138
10.2.4 Module-Specific Definitions . . . 139
10.3 Module Components . . . 139
10.3.1 Memory Cell Array . . . 139
10.3.1.1 Page . . . 140
10.3.1.2 Sector . . . 140
10.4 Functional Description . . . 140
10.4.1 SFR Accesses . . . 140
10.4.2 Memory Read . . . 140
10.4.3 Memory Write . . . 140
10.4.4 Memory Erase . . . 141
10.4.5 Sector Erase . . . 141
10.4.6 Verify . . . 142
10.4.7 Erase-Protection and Write-Protection . . . 142
10.5 Properties and Implementation of Error Correcting Code (ECC) . . . 142
10.6 Service Request Generation . . . 142
10.7 Power, Reset and Clock . . . 144
10.7.1 Power Supply . . . 144
10.7.2 Power Saving Modes . . . 144
10.7.2.1 NVM Idle Mode . . . 144
10.7.2.2 NVM Sleep Mode . . . 144
10.7.3 Reset . . . 144
10.7.4 Clock . . . 144
10.8 Registers . . . 145
10.9 Example Sequences . . . 145
10.9.1 Writing to Memory . . . 145
10.9.1.1 Writing a Single Block . . . 145
10.9.1.2 Writing Blocks . . . 146
10.9.2 Erasing Memory . . . 146
10.9.2.1 Erasing a Single Page . . . 146
10.9.2.2 Erasing Pages . . . 146
10.9.2.3 Erasing a Single Sector . . . 147
10.9.2.4 Erasing Sectors . . . 147
10.9.3 Verifying Memory . . . 147
10.9.3.1 Verifying a Single Block . . . 147
10.9.3.2 Verifying Blocks . . . 148
10.9.4 Writing to an Already Written Block . . . 148
10.9.5 Sleep Mode . . . 149
10.9.6 Timing . . . 150

11 Peripheral Access Unit (PAU) . . . 151

Table of contents

11.1 Overview . . . 151
11.1.1 Features . . . 151
11.2 Peripheral Privilege Access Control . . . 151
11.3 Peripheral Availability and Memory Size . . . 152
11.4 Service Request Generation . . . 153
11.5 Debug Behaviour . . . 153
11.6 Power, Reset and Clock . . . 153
11.7 Initialization and System Dependencies . . . 153
11.8 PAU Registers . . . 153

12 Window Watchdog Timer (WDT) . . . 155
12.1 Overview . . . 155
12.1.1 Features . . . 155
12.1.2 Block Diagram . . . 156
12.2 Time-Out Mode . . . 156
12.3 Pre-warning Mode . . . 157
12.4 Bad Service Operation . . . 157
12.5 Service Request Processing . . . 158
12.6 Debug Behavior . . . 158
12.7 Power, Reset and Clock . . . 158
12.8 Initialization and Control Sequence . . . 159
12.8.1 Initialization and Operation Start-Up . . . 159
12.8.2 Software Stop and Resume Operation . . . 159
12.8.3 Enter Sleep/Deep-Sleep & Resume Operation . . . 159
12.8.4 Pre-warning Alarm Handling . . . 160
12.9 WDT Registers . . . 161
12.10 Interconnects . . . 161

13 Real-Time Clock (RTC) . . . 162
13.1 Overview . . . 162
13.1.1 Features . . . 162
13.1.2 Block Diagram . . . 162
13.2 RTC Operation . . . 163
13.3 Register Access Operations . . . 164
13.4 Service Request Processing . . . 164
13.4.1 Periodic Service Request . . . 164
13.4.2 Timer Alarm Service Request . . . 164
13.5 Debug Behavior . . . 164
13.6 Power, Reset, and Clock . . . 164
13.7 Initialization and Control Sequence . . . 165
13.7.1 Initialization & Start of Operation . . . 165
13.7.2 Configure and Enable Periodic Event . . . 166
13.7.3 Configure and Enable Timer Event . . . 166

13.8 RTC Registers . . . 166
13.9 Interconnects . . . 167
14 System Control Unit (SCU) . . . 168
14.1 Overview . . . 168
14.1.1 Features . . . 168
14.1.2 Block Diagram . . . 169
14.2 Miscellaneous Control Functions (GCU) . . . 170
14.2.1 Service Requests Handling . . . 170
14.2.1.1 Service Request Sources . . . 171
14.2.2 SRAM Memory Content Protection . . . 172
14.2.3 Summary of ID . . . 172
14.2.4 Boot via Pins . . . 173
14.3 Power Management (PCU) . . . 173
14.3.1 Functional Description . . . 173
14.3.2 System States . . . 173
14.3.3 Embedded Voltage Regulator (EVR) . . . 174
14.3.4 Power-on Reset . . . 174
14.3.5 Power Validation . . . 175
14.3.6 Supply Voltage Monitoring . . . 175
14.3.7 V_{DDC} Response During Load Change . . . 175
14.3.8 Flash Power Control . . . 176
14.4 Reset Control (RCU) . . . 176
14.4.1 Functional Description . . . 176
14.4.2 Reset Status . . . 177
14.5 Clock Control (CCU) . . . 178
14.5.1 Features . . . 178
14.5.2 Clock System and Control . . . 178
14.5.2.1 DCO1 Oscillator Watchdog . . . 183
14.5.2.2 Loss of DCO1 Clock Detection and Recovery . . . 183
14.5.2.3 Standby Clock Failure . . . 183
14.5.2.4 XTAL Oscillator Watchdog . . . 183
14.5.2.5 Loss of external OSC_HP Clock Detection and Recovery . . . 184
14.5.2.6 Startup Control for System Clock . . . 184
14.5.2.7 DCLK input - External Clock via OSC_HP . . . 184
14.5.3 Clock Gating Control . . . 184
14.5.4 Calibrating DCO1 based on Temperature . . . 185
14.5.5 Automatic DCO1 Calibration based on External Reference . . . 185
14.6 Service Request Generation . . . 187
14.7 Debug Behavior . . . 187
14.8 Power, Reset and Clock . . . 187
14.9 Registers . . . 189

Table of contents

15 Pseudo Random Number Generator (PRNG) . . . 192
15.1 Overview . . . 192
15.1.1 Features . . . 192
15.2 Description of Operation Modes . . . 192
15.2.1 Key Loading Mode . . . 192
15.2.2 Streaming Mode . . . 192
15.2.3 Refreshing and Restarting a Random Bit Stream . . . 193
15.3 Service Request Generation . . . 193
15.4 Debug Behavior . . . 193
15.5 Power, Reset and Clock . . . 193
15.6 Initialization and System Dependencies . . . 194
15.7 Registers . . . 194
16 Universal Serial Interface Channel (USIC) . . . 195
16.1 Overview . . . 196
16.1.1 Features . . . 196
16.2 Operating the USIC . . . 199
16.2.1 USIC Structure Overview . . . 199
16.2.1.1 Channel Structure . . . 199
16.2.1.2 Input Stages . . . 199
16.2.1.3 Output Signals . . . 200
16.2.1.4 Baud Rate Generator . . . 201
16.2.1.5 Channel Events and Interrupts . . . 202
16.2.1.6 Data Shifting and Handling . . . 202
16.2.2 Operating the USIC Communication Channel . . . 204
16.2.2.1 Protocol Control and Status . . . 204
16.2.2.2 Mode Control . . . 205
16.2.3 Operating the Input Stages . . . 206
16.2.3.1 General Input Structure . . . 206
16.2.3.2 Input Selection . . . 207
16.2.3.3 Input Conditioning . . . 207
16.2.3.4 Digital Filter . . . 207
16.2.3.5 Edge Detection . . . 208
16.2.3.6 Selected Input Monitoring . . . 208
16.2.3.7 Loop Back Mode . . . 208
16.2.3.8 Delay Compensation in DX1 . . . 208
16.2.4 Operating the Baud Rate Generator . . . 209
16.2.4.1 Fractional Divider . . . 209
16.2.4.2 External Frequency Input . . . 209
16.2.4.3 Divider Mode Counter . . . 210
16.2.4.4 Capture Mode Timer . . . 210
16.2.4.5 Time Quanta Counter . . . 211

Table of contents

16.2.4.6 Master and Shift Clock Output Configuration . . . 211
16.2.5 Operating the Transmit Data Path . . . 212
16.2.5.1 Transmit Buffering . . . 212
16.2.5.2 Transmit Data Shift Mode . . . 213
16.2.5.3 Transmit Control Information . . . 214
16.2.5.4 Transmit Data Validation . . . 215
16.2.6 Operating the Receive Data Path . . . 216
16.2.6.1 Receive Buffering . . . 217
16.2.6.2 Receive Data Shift Mode . . . 217
16.2.6.3 Baud Rate Constraints . . . 218
16.2.7 Hardware Port Control . . . 218
16.2.8 Operating the FIFO Data Buffer . . . 218
16.2.8.1 FIFO Buffer Partitioning . . . 219
16.2.8.2 Transmit FIFO Buffer Modes . . . 220
16.2.8.3 Transmit Buffer Events and Interrupts . . . 221
16.2.8.4 Transmit FIFO Buffer Usage Example . . . 223
16.2.8.5 Receive FIFO Buffer Modes . . . 224
16.2.8.6 Receive Buffer Events and Interrupts . . . 226
16.2.8.7 Receive FIFO Buffer in Filling Level Mode Usage Example . . . 228
16.2.8.8 FIFO Buffer Bypass . . . 229
16.2.8.9 FIFO Access Constraints . . . 231
16.2.8.10 Handling of FIFO Transmit Control Information . . . 231
16.3 Asynchronous Serial Channel (ASC = UART) . . . 233
16.3.1 Signal Description . . . 233
16.3.1.1 ASC Full-Duplex Communication . . . 233
16.3.1.2 ASC Half-Duplex Communication . . . 234
16.3.2 Frame Format . . . 235
16.3.2.1 Idle Detection . . . 235
16.3.2.2 Start Bit Detection . . . 236
16.3.2.3 Data Field . . . 236
16.3.2.4 Parity Bit . . . 236
16.3.2.5 Stop Bit(s) . . . 237
16.3.3 Operating the ASC . . . 237
16.3.3.1 Bit Timing . . . 238
16.3.3.2 Baud Rate Generation . . . 239
16.3.3.3 Automatic Shadow Mechanism . . . 239
16.3.3.4 Mode Control Behavior . . . 240
16.3.3.5 Disabling ASC Mode . . . 240
16.3.3.6 Data Transfer Interrupt Handling . . . 240
16.3.3.7 Protocol Interrupt Events . . . 241
16.3.3.8 Baud Rate Generator Interrupt Handling . . . 242
16.3.3.9 Protocol-Related Argument and Error . . . 242

16.3.3.10 Receive FIFO Buffer Handling . . . 242
16.3.3.11 Data Flow Handling . . . 242
16.3.3.12 Initialization Code Example . . . 245
16.3.4 Additional Features . . . 246
16.3.4.1 Noise Detection . . . 246
16.3.4.2 Collision Detection . . . 246
16.3.4.3 Pulse Shaping . . . 246
16.3.4.4 End of Frame Control . . . 247
16.3.4.5 Sync-Break Detection . . . 247
16.3.4.6 Transfer Status Indication . . . 248
16.3.5 Hardware LIN Support . . . 248
16.4 Synchronous Serial Channel (SSC) . . . 249
16.4.1 Signal Description . . . 249
16.4.1.1 Transmit and Receive Data Signals . . . 252
16.4.1.2 Shift Clock Signals . . . 252
16.4.1.3 Slave Select Signals . . . 255
16.4.2 Operating the SSC . . . 256
16.4.2.1 Automatic Shadow Mechanism . . . 256
16.4.2.2 Mode Control Behavior . . . 257
16.4.2.3 Disabling SSC Mode . . . 257
16.4.2.4 Data Frame Control . . . 257
16.4.2.5 Parity Mode . . . 258
16.4.2.6 Data Transfer Interrupt Handling . . . 259
16.4.2.7 Protocol-Related Argument and Error . . . 259
16.4.2.8 Receive Buffer Handling . . . 260
16.4.3 Operating the SSC in Master Mode . . . 260
16.4.3.1 Baud Rate Generation . . . 261
16.4.3.2 MSLS Generation and Slave Select Delays . . . 261
16.4.3.3 Configuration of Slave Select Delays . . . 262
16.4.3.4 Automatic Slave Select Update . . . 263
16.4.3.5 Protocol Interrupt Events . . . 263
16.4.3.6 End-of-Frame Control . . . 265
16.4.3.7 Data Flow Handling . . . 266
16.4.3.8 Initialization Code Example . . . 268
16.4.4 Operating the SSC in Slave Mode . . . 269
16.4.4.1 Protocol Interrupts . . . 270
16.4.4.2 End-of-Frame Control . . . 270
16.4.4.3 Data Flow Handling . . . 271
16.4.4.4 Initialization Code Example . . . 273
16.4.5 Multi-IO SSC Protocols . . . 274
16.4.5.1 Operating the SSC in Multi-IO Modes . . . 276
16.4.5.2 Quad-SSC Example . . . 276

Table of contents

16.4.6 SSC Timing Considerations 278
16.4.6.1 Closed-loop Delay 278
16.4.6.2 Delay Compensation in Master Mode 279
16.4.6.3 Complete Closed-loop Delay Compensation 280
16.5 Inter-IC Bus Protocol (IIC) 281
16.5.1 Introduction 281
16.5.1.1 Signal Description 281
16.5.1.2 Symbols 283
16.5.1.3 Frame Format 284
16.5.2 Symbol Timing 284
16.5.2.1 Start Symbol 285
16.5.2.2 Repeated Start Symbol 285
16.5.2.3 Stop Symbol 286
16.5.2.4 Data Bit Symbol 286
16.5.3 Operating the IIC 287
16.5.3.1 Baud Rate Generation 288
16.5.3.2 Transmission Chain 289
16.5.3.3 Byte Stretching 289
16.5.3.4 Master Arbitration 289
16.5.3.5 Not Acknowledge and Error Conditions 289
16.5.3.6 Mode Control Behavior 290
16.5.3.7 Data Transfer Interrupt Handling 290
16.5.3.8 IIC Protocol Interrupt Events 291
16.5.3.9 Receiver Address Acknowledge 292
16.5.3.10 Receiver Handling 293
16.5.3.11 Receiver Status Information 293
16.5.3.12 IIC Initialization Code Example 294
16.5.4 Data Flow Handling 296
16.5.4.1 Transmit Data Formats 296
16.5.4.2 Valid Master Transmit Data Formats 297
16.5.4.3 Master Transmit/Receive Modes 299
16.5.4.4 Slave Transmit/Receive Modes 302
16.6 Inter-IC Sound Bus Protocol (IIS) 303
16.6.1 Introduction 303
16.6.1.1 Signal Description 303
16.6.1.2 Protocol Overview 306
16.6.1.3 Transfer Delay 308
16.6.1.4 Connection of External Audio Components 308
16.6.2 Operating the IIS 308
16.6.2.1 Frame Length and Word Length Configuration 308
16.6.2.2 Automatic Shadow Mechanism 309
16.6.2.3 Mode Control Behavior 309

16.6.2.4 Transfer Delay . . . 309
16.6.2.5 Data Transfer Interrupt Handling . . . 311
16.6.2.6 Protocol-Related Argument and Error . . . 311
16.6.2.7 Transmit Data Handling . . . 312
16.6.2.8 Receive Buffer Handling . . . 312
16.6.2.9 Loop-Delay Compensation . . . 312
16.6.3 Operating the IIS in Master Mode . . . 312
16.6.3.1 Baud Rate Generation . . . 313
16.6.3.2 WA Generation . . . 314
16.6.3.3 Master Clock Output . . . 314
16.6.3.4 Protocol Interrupt Events . . . 315
16.6.3.5 Initialization Code Example . . . 316
16.6.4 Operating the IIS in Slave Mode . . . 317
16.6.4.1 Protocol Interrupt Events . . . 318
16.6.4.2 Initialization Code Example . . . 319
16.7 Service Request Generation . . . 320
16.7.1 General Channel Events and Interrupts . . . 320
16.7.2 Data Transfer Events and Interrupts . . . 320
16.7.3 Baud Rate Generator Event and Interrupt . . . 322
16.7.4 Protocol-specific Events and Interrupts . . . 322
16.8 Debug Behaviour . . . 323
16.9 Power, Reset and Clock . . . 324
16.10 Initialization and System Dependencies . . . 325
16.11 Registers . . . 326
16.11.1 Address Map . . . 329
16.12 Interconnects . . . 330
16.12.1 USIC0 Module Interconnects . . . 330
16.12.1.1 USIC0 Channel 0 Interconnects . . . 330
16.12.1.2 USIC0 Channel 1 Interconnects . . . 335
16.12.1.3 USIC0 Global Interconnects . . . 340
16.12.2 USIC1 Module Interconnects . . . 340
16.12.2.1 USIC1 Channel 0 Interconnects . . . 340
16.12.2.2 USIC1 Channel 1 Interconnects . . . 345
16.12.2.3 USIC1 Global Interconnects . . . 349

17 Controller Area Network Controller (MultiCAN+) . . . 350
17.1 CAN Basics . . . 350
17.1.1 Addressing and Bus Arbitration . . . 350
17.1.2 CAN Frame Types . . . 351
17.1.2.1 Data Frames . . . 351
17.1.2.2 Remote Frames . . . 352
17.1.2.3 Error Frames . . . 353
17.1.3 The Nominal Bit Time . . . 353

17.1.4 Error Detection and Error Handling 354
17.2 Overview 355
17.2.1 Features List 356
17.3 MultiCAN+ Kernel Functional Description 358
17.3.1 Module Structure 358
17.3.2 Clock Control 360
17.3.3 Port Input Control 363
17.3.4 CAN Node Control 363
17.3.4.1 Bit Timing Unit 364
17.3.4.2 Bitstream Processor 365
17.3.4.3 Error Handling Unit 365
17.3.4.4 CAN Frame Counter 366
17.3.4.5 CAN Node Interrupts 366
17.3.5 Message Object List Structure 368
17.3.5.1 Basics 368
17.3.5.2 List of Unallocated Elements 369
17.3.5.3 Connection to the CAN Nodes 369
17.3.5.4 List Command Panel 370
17.3.6 CAN Node Analyzer Mode 372
17.3.6.1 Analyzer Mode 372
17.3.6.2 Loop-Back Mode 372
17.3.6.3 Bit Timing Analysis 373
17.3.7 Message Acceptance Filtering 374
17.3.7.1 Receive Acceptance Filtering 374
17.3.7.2 Transmit Acceptance Filtering 375
17.3.8 Message Postprocessing 376
17.3.8.1 Message Object Interrupts 376
17.3.8.2 Pending Messages 377
17.3.9 Message Object Data Handling 379
17.3.9.1 Frame Reception 379
17.3.9.2 Frame Transmission 382
17.3.10 Message Object Functionality 385
17.3.10.1 Standard Message Object 385
17.3.10.2 Single Data Transfer Mode 385
17.3.10.3 Single Transmit Trial 385
17.3.10.4 Message Object FIFO Structure 385
17.3.10.5 Receive FIFO 388
17.3.10.6 Transmit FIFO 388
17.3.10.7 Gateway Mode 389
17.3.10.8 Foreign Remote Requests 390
17.4 Use Case Example MultiCAN+ 391
17.5 MultiCAN+ Kernel Registers 395

17.5.1 Global Module Registers . . . 399
17.5.1.1 Panel Control Register . . . 399
17.5.2 CAN Node Registers . . . 402
17.5.2.1 Node x Status Register . . . 402
17.5.2.2 Node x Frame Counter Register . . . 403
17.5.3 Message Object Registers . . . 404
17.5.3.1 Message Object 31 Control Register . . . 404
17.5.3.2 Message Object 31 Status Register . . . 404
17.5.3.3 Message Object n Arbitration Register . . . 405
17.6 MultiCAN+ Module Implementation . . . 406
17.6.1 Interfaces of the MultiCAN+ Module . . . 406
17.6.2 MultiCAN+ Module External Registers . . . 406
17.6.3 Module Clock Generation . . . 408
17.6.3.1 Clock Selection . . . 408
17.6.3.2 Fractional Divider . . . 408
17.6.4 Port and I/O Line Control . . . 409
17.6.4.1 Input/Output Function Selection in Ports . . . 410
17.6.4.2 Node Receive Input Selection . . . 410
17.6.4.3 Connections to Interrupt Router Inputs . . . 411
17.6.4.4 Connections to ERU . . . 411
17.6.5 Interrupt Control . . . 412
17.6.6 MultiCAN+ Module Register Address Map . . . 414

18 Versatile Analog-to-Digital Converter (VADC) . . . 416
18.1 Overview . . . 416
18.2 Introduction and Basic Structure . . . 418
18.3 Electrical Models . . . 422
18.4 Configuration of General Functions . . . 424
18.4.1 General Clocking Scheme and Control . . . 424
18.4.2 Register Access Control . . . 425
18.4.3 Priority Channel and Result Register Assignment . . . 425
18.5 Analog Module Activation and Control . . . 426
18.5.1 Analog Converter Control . . . 426
18.5.2 Converter Handling in Deep Sleep Mode . . . 427
18.5.3 Calibration . . . 427
18.5.4 Reference Voltage Selection . . . 428
18.5.5 Sigma-Delta-Loop Function . . . 429
18.6 Conversion Request Generation . . . 429
18.6.1 Queued Request Source Handling . . . 430
18.6.2 Channel Scan Request Source Handling . . . 433
18.7 Request Source Arbitration . . . 435
18.7.1 Arbiter Operation and Configuration . . . 436
18.7.2 Conversion Start Mode . . . 437

Table of contents

18.8 Analog Input Channel Configuration . . . 438
18.8.1 Channel Parameters . . . 439
18.8.2 Alias Feature . . . 439
18.8.3 Conversion Modes . . . 440
18.8.4 Compare with Standard Conversions (Limit Checking) . . . 441
18.8.5 Utilizing Fast Compare Mode . . . 442
18.8.6 Boundary Flag Control . . . 442
18.9 Conversion Scheduling . . . 444
18.10 Conversion Timing . . . 445
18.10.1 Timing Definition . . . 445
18.10.2 Compatible Timing Mode . . . 447
18.11 Conversion Result Handling . . . 447
18.11.1 Storage of Conversion Results . . . 448
18.11.2 Data Alignment . . . 449
18.11.3 Wait-for-Read Mode . . . 450
18.11.4 Result FIFO Buffer . . . 451
18.11.5 Result Event Generation . . . 452
18.11.6 Data Modification . . . 452
18.12 Synchronization of Conversions . . . 458
18.12.1 Synchronized Conversions for Parallel Sampling . . . 458
18.12.2 Equidistant Sampling . . . 460
18.13 Safety Features . . . 461
18.13.1 Broken Wire Detection . . . 461
18.13.2 Multiplexer Diagnostics . . . 462
18.14 External Multiplexer Control . . . 462
18.15 Service Request Generation . . . 464
18.16 Registers . . . 465
18.17 Interconnects . . . 467
18.17.1 Product-Specific Configuration . . . 468
18.17.2 Analog Module Connections in the PSC1Mx . . . 468
18.17.3 Digital Module Connections in the PSC1Mx . . . 469

19 Analog Comparator (ACMP) and Out of Range Comparator (ORC) . . . 474
19.1 Overview . . . 474
19.1.1 Features . . . 474
19.2 Analog Comparator (ACMP) . . . 474
19.3 Out of Range Comparator (ORC) . . . 475
19.4 Service Request Generation . . . 476
19.5 Debug Behavior . . . 476
19.6 Registers . . . 476
19.7 Interconnects . . . 478

20 Temperature Sensor (DTS) . . . 481

20.1 General Description . . . 481
20.2 Service Request Generation . . . 481
20.3 Registers . . . 481
21 Capture/Compare Unit 4 (CCU4) . . . 482
21.1 Overview . . . 482
21.1.1 Features . . . 483
21.1.2 Block Diagram . . . 485
21.2 Functional Description . . . 486
21.2.1 Timer Slice Overview . . . 486
21.2.2 Timer Slice Input Selector . . . 488
21.2.3 Timer Slice Connection Matrix . . . 488
21.2.4 Timer Slice Core Functions . . . 489
21.2.4.1 Starting/Stopping the Timer . . . 490
21.2.4.2 Counting Modes Introduction . . . 491
21.2.4.3 Edge Aligned Mode . . . 491
21.2.4.4 Center Aligned Mode . . . 493
21.2.4.5 Single Shot Mode . . . 494
21.2.4.6 Calculating the PWM Period and Duty Cycle . . . 495
21.2.4.7 Updating the Period, Duty Cycle and other PWM conditions . . . 496
21.2.4.8 PWM Active/Passive Rules . . . 504
21.2.4.9 Output PWM Path . . . 505
21.2.5 Timer Slice External Functions . . . 505
21.2.5.1 External Start/Stop . . . 506
21.2.5.2 External Counting Direction . . . 507
21.2.5.3 External Gating Signal . . . 508
21.2.5.4 External Count Signal . . . 509
21.2.5.5 External Load . . . 509
21.2.5.6 External Capture . . . 510
21.2.5.7 TRAP Function . . . 514
21.2.5.8 Status Bit Override . . . 515
21.2.5.9 External Modulation . . . 516
21.2.6 Timer Slice Advanced Functions . . . 517
21.2.6.1 Multi-Channel Control . . . 518
21.2.6.2 Timer Concatenation . . . 519
21.2.6.3 PWM Dithering . . . 523
21.2.6.4 Capture Extended Read Back Mode . . . 527
21.2.7 Clock Prescaler . . . 529
21.2.7.1 Normal Prescaler Mode . . . 529
21.2.7.2 Floating Prescaler Mode . . . 530
21.2.8 CCU4 Usage . . . 531
21.2.8.1 PWM Signal Generation . . . 531
21.2.8.2 Prescaler Usage . . . 533

21.2.8.3 PWM Dither 534
21.2.8.4 Capture Mode Usage 536
21.3 Service Request Generation 541
21.4 Debug Behavior 543
21.5 Power, Reset and Clock 544
21.5.1 Clocks 544
21.5.2 Power 544
21.6 Initialization and System Dependencies 544
21.6.1 Initialization Sequence 544
21.6.2 System Dependencies 545
21.7 Registers 545
21.8 Interconnects 547
21.8.1 CCU40 Pins 548
21.8.2 CCU41 Pins 557

22 Capture/Compare Unit 8 (CCU8) 567
22.1 Overview 567
22.1.1 Features 568
22.1.2 Block Diagram 570
22.2 Functional Description 571
22.2.1 Timer Slice Overview 572
22.2.2 Timer Slice Input Selector 574
22.2.3 Timer Slice Connection Matrix 575
22.2.4 Timer Slice Core Functions 576
22.2.4.1 Starting/Stopping the Timer 576
22.2.4.2 Counting Modes Introduction 578
22.2.4.3 Edge Aligned Mode 579
22.2.4.4 Center Aligned Mode 580
22.2.4.5 Single Shot Mode 582
22.2.4.6 Dead Time Generation 583
22.2.4.7 Edge and Center Aligned Compare Modes 586
22.2.4.8 Calculating the PWM Period and Duty Cycle 591
22.2.4.9 Updating the Period, Duty Cycle and other PWM conditions 592
22.2.4.10 PWM Active/Passive Rules 599
22.2.4.11 Output PWM Path 600
22.2.5 Timer Slice External Functions 601
22.2.5.1 External Start/Stop 601
22.2.5.2 External Counting Direction 603
22.2.5.3 External Gating Signal 604
22.2.5.4 External Count Signal 605
22.2.5.5 External Load 605
22.2.5.6 External Capture 606
22.2.5.7 External Modulation 610

22.2.5.8 Trap Function 612
22.2.5.9 Status Bit Override 614
22.2.6 Timer Slice Advanced Functions 614
22.2.6.1 Multi-Channel Support 614
22.2.6.2 Timer Concatenation 617
22.2.6.3 PWM Dithering 621
22.2.6.4 Capture Extended Read Back Mode 625
22.2.6.5 Output Parity Checker 627
22.2.7 Clock Prescaler 629
22.2.7.1 Normal Prescaler Mode 630
22.2.7.2 Floating Prescaler Mode 630
22.2.8 CCU8 Usage 631
22.2.8.1 PWM Signal Generation 631
22.2.8.2 Prescaler Usage 633
22.2.8.3 PWM Dither 635
22.2.8.4 Capture Mode Usage 637
22.2.8.5 Parity Checker Usage 642
22.3 Service Request Generation 644
22.4 Debug Behavior 646
22.5 Power, Reset and Clock 646
22.5.1 Clocks 646
22.5.2 Power 647
22.6 Initialization and System Dependencies 648
22.6.1 Initialization Sequence 648
22.6.2 System Dependencies 648
22.7 Registers 648
22.8 Interconnects 651
22.8.1 CCU80 Pins 651
22.8.2 CCU81 Pins 662

23 Position Interface Unit (POSIF) 673
23.1 Overview 673
23.1.1 Features 673
23.1.2 Block Diagram 675
23.2 Functional Description 675
23.2.1 Overview 676
23.2.2 Function Selector 677
23.2.3 Hall Sensor Control 678
23.2.4 Quadrature Decoder Control 681
23.2.4.1 Quadrature Clock and Direction decoding 684
23.2.4.2 Index Control 684
23.2.5 Stand-Alone Multi-Channel Mode 685
23.2.6 Synchronous Start 685

23.2.7 Using the POSIF . . . 686
23.2.7.1 Hall Sensor Mode Usage . . . 686
23.2.7.2 Quadrature Decoder Mode usage . . . 688
23.2.7.3 Stand-alone Multi-Channel Mode . . . 691
23.3 Service Request Generation . . . 692
23.3.1 Hall Sensor Mode flags . . . 692
23.3.2 Quadrature Decoder Flags . . . 693
23.4 Debug Behavior . . . 695
23.5 Power, Reset and Clock . . . 695
23.5.1 Clocks . . . 695
23.5.2 Power . . . 696
23.6 Initialization and System Dependencies . . . 696
23.6.1 Initialization . . . 696
23.6.2 System Dependencies . . . 696
23.7 Registers . . . 696
23.8 Interconnects . . . 698
23.8.1 POSIF0 Pins . . . 698
23.8.2 POSIF1 Pins . . . 701

24 Brightness and Color Control Unit (BCCU) . . . 705
24.1 Overview . . . 705
24.1.1 Features . . . 705
24.2 Functional Description . . . 705
24.2.1 Channel Structure . . . 707
24.2.2 Exponential Dimming . . . 707
24.2.3 Linear Color Walk . . . 714
24.2.4 Sigma-Delta Modulator . . . 715
24.2.5 Packer . . . 715
24.2.6 Global Trigger Control . . . 717
24.2.7 Trap Control . . . 718
24.3 Power, Reset and Clock . . . 718
24.4 Service Request Generation . . . 719
24.5 Debug Behaviour . . . 720
24.6 Initialization . . . 720
24.7 Digital-to-Analog Converter . . . 721
24.8 Registers . . . 721
24.9 Interconnects . . . 722

25 General Purpose I/O Ports (Ports) . . . 727
25.1 Overview . . . 727
25.1.1 Features . . . 727
25.1.2 Block Diagram . . . 728
25.1.3 Definition of Terms . . . 728

Table of contents

25.2 GPIO and Alternate Functions . . . 729
25.2.1 Input Operation . . . 729
25.2.2 Output Operation . . . 730
25.3 Hardware Controlled I/Os . . . 731
25.4 Power Saving Mode Operation . . . 732
25.5 Analog Ports . . . 733
25.6 Power, Reset and Clock . . . 733
25.7 Initialization and System Dependencies . . . 733
25.8 Registers . . . 735
25.9 Package Pin Summary . . . 736
25.10 Port I/O Functions . . . 739
25.10.1 Port Pin for Boot Modes . . . 739
25.10.2 Port I/O Function Description . . . 740
25.10.3 Hardware Controlled I/O Function Description . . . 741
25.11 Pad Characteristics . . . 753
25.11.1 Input and Output Voltage . . . 753
25.11.2 Input Low and Input High Voltage . . . 753
25.11.3 Output Low and Output High Voltage . . . 754
25.11.4 Pin Reliability in Overload . . . 755
25.11.5 Internal Pull Device . . . 756

26 Boot and Startup . . . 758
26.1 Startup Sequence and System Dependencies . . . 758
26.1.1 Power-Up . . . 758
26.1.2 System Reset Release . . . 758
26.1.3 Startup Software (SSW) Execution . . . 759
26.1.3.1 Clock system handling by SSW . . . 759
26.1.4 Configuration of Special System Functions as part of User code initialization . . . 760
26.1.5 Configuration of Clock System and Miscellaneous Functions . . . 760
26.2 Start-up Modes . . . 761
26.2.1 Start-up modes in PSC1Mx . . . 761
26.2.1.1 User productive mode . . . 761
26.2.1.2 User mode with debug enabled . . . 762
26.2.1.3 User mode with debug enabled and Halt After Reset (HAR) . . . 762
26.2.1.4 Alternate Boot Mode (ABM) . . . 762
26.2.1.5 Standard Bootstrap Loader modes . . . 762
26.2.1.6 Bootstrap Loader modes with time-out . . . 763
26.2.2 Boot Mode Index (BMI) . . . 763
26.2.3 Start-up mode selection . . . 765
26.2.3.1 BMI handling by SSW . . . 765
26.2.3.2 Debug system handling . . . 766
26.3 Data in Flash for SSW and User SW . . . 766

Table of contents

27 Bootstrap Loaders (BSL) and User Routines . . . 768
27.1 ASC (UART) Bootstrap Loader . . . 768
27.1.1 Pin usage . . . 768
27.1.2 ASC BSL execution flow . . . 768
27.1.2.1 ASC BSL entry check sequence . . . 768
27.1.2.2 ASC BSL download sequence . . . 772
27.1.3 ASC BSL protocol data definitions . . . 773
27.2 SSC Bootstrap loader . . . 775
27.3 CAN BSL mode . . . 777
27.3.1 Initialization Phase . . . 779
27.3.2 Acknowledgement Phase . . . 779
27.3.3 Data Transmission Phase . . . 780
27.4 Firmware routines available for the user . . . 780
27.4.1 Erase Flash Page . . . 781
27.4.2 Erase, Program & Verify Flash Page . . . 781
27.4.3 Request BMI installation . . . 781
27.4.4 Calculate chip temperature . . . 781
27.4.5 DCO Calibration . . . 782
27.4.6 Erase Flash Sector . . . 782
27.4.7 Program & Verify Flash Block . . . 782
27.4.8 Calculate target level for temperature comparison . . . 783
27.5 Data in Flash used by the User Routines . . . 783

28 Debug System (DBG) . . . 784
28.1 Overview . . . 784
28.1.1 Features . . . 784
28.1.2 Block Diagram . . . 785
28.2 Debug System Operation . . . 785
28.2.1 System Control Space (SCS) . . . 785
28.2.2 Data Watchpoint and Trace (DWT) . . . 785
28.2.3 Break Point Unit (BPU) . . . 786
28.2.4 ROM Table . . . 786
28.2.5 Debug tool interface access - SWD . . . 786
28.2.5.1 SWD based transfers . . . 786
28.2.5.2 SWD based errors . . . 787
28.2.6 Debug tool interface access - Single Pin Debug (SPD) . . . 788
28.2.7 Debug accesses and Flash protection . . . 789
28.2.8 Halt after reset . . . 790
28.2.8.1 HAR . . . 790
28.2.8.2 Warm Reset . . . 792
28.2.9 Halting Debug and Peripheral Suspend . . . 793
28.2.10 Debug System based processor wake-up . . . 794

Table of contents

28.2.11 Debug Access Server (DAS) . . . 794
28.2.12 Debug Signals . . . 794
28.2.12.1 Internal pull-up and pull-down on SWD/SPD pins . . . 795
28.2.13 Reset . . . 795
28.3 Debug System Power Save Operation . . . 795
28.4 Service Request Generation . . . 795
28.5 Debug behavior . . . 795
28.6 Power, Reset and Clock . . . 796
28.6.1 Power management . . . 796
28.6.2 Debug System reset . . . 796
28.6.3 Debug System Clocks . . . 796
28.7 Initialization and System Dependencies . . . 796
28.7.1 ID Code . . . 796
28.7.2 ROM Table . . . 797
28.8 Debug System Registers . . . 798
28.8.1 DFSR - Debug Fault Status Register . . . 799
28.8.1.1 SCS_DFSR . . . 799
28.8.2 DHCSR - Debug Halting Control and Status Register . . . 800
28.8.2.1 SCS_DHCSR [Read Mode] . . . 800
28.8.2.2 SCS_DHCSR [Write Mode] . . . 803
28.8.3 DCRSR - Debug Core Register Selector Register . . . 804
28.8.3.1 SCS_DCRSR . . . 804
28.8.4 DCRDR - Debug Core Register Data Register . . . 805
28.8.4.1 SCS_DCRDR . . . 805
28.8.5 DEMCR - Debug Exception and Monitor Control Register . . . 806
28.8.5.1 SCS_DEMCR . . . 806
28.8.6 DWT_CTRL - Data Watchpoint Control Register . . . 807
28.8.6.1 DWT_CTRL . . . 807
28.8.7 DWT_PCSR - Program Counter Sample Register . . . 808
28.8.7.1 DWT_PCSR . . . 808
28.8.8 DWT_COMPx - DWT Comparator register . . . 808
28.8.8.1 DWT_COMPx . . . 808
28.8.9 DWT_MASKx - DWT Comparator Mask Register . . . 809
28.8.9.1 DWT_MASKx . . . 809
28.8.10 DWT_FUNCTIONx - Comparator Function Register . . . 810
28.8.10.1 DWT_FUNCTIONx . . . 810
28.8.11 BP_CTRL - Breakpoint Control Register . . . 811
28.8.11.1 BP_CTRL . . . 811
28.8.12 Breakpoint Comparator Registers . . . 811
28.8.12.1 BP_COMPx . . . 811

Revision history . . . 813

Table of contents

Trademarks 814

Disclaimer 815

1 Introduction

The PSC1M2x/PSC1M3x series belongs to the PSC1Mx Family of industrial microcontrollers based on the Arm® Cortex®-M0 processor core. The PSC1M2x/PSC1M3x series devices are optimized for motor control, power conversion and LED Lighting applications.

Increasing complexity and demand for computing power of embedded control applications requires microcontrollers to have a significant CPU performance, integrated peripheral functionality and rapid development environment enabling short time-to-market, without compromising cost efficiency. Nonetheless the architecture of the PSC1Mx microcontroller pursue successful hardware and software concepts, which have been established in Infineon microcontroller families.

1.1 Overview

The PSC1M2x/PSC1M3x series devices combine the extended functionality and performance of the Cortex®-M0 core with powerful on-chip peripheral subsystems and on-chip memory units. The following key features are available within the range of PSC1M2x/PSC1M3x series devices:

CPU Subsystem

- CPU Core
  - High Performance 32-bit Arm® Cortex®-M0 CPU Core
- Nested Vectored Interrupt Controller (NVIC)
- Event Request Unit (ERU) for event interconnections
- MATH Co-processor (MATH)
  - 24-bit trigonometric calculation (CORDIC)
  - 32-bit division unit
- Prefetch Unit (FPU) for code fetch access, to reduce flash latency

On-Chip Memories

- 8 KB on-chip ROM
- 16 KB SRAM (with parity)
- up to 200 KB Flash (with ECC)

Supply, Reset, Clock

- 1.8 V to 5.5 V supply with power-on reset and brownout detector
- On-chip clock monitor
- External crystal oscillator support (32 kHz and 4 to 20 MHz)
- Internal slow and fast oscillators

Communication Peripherals

- Two Universal Serial Interface Channels (USIC), usable as UART, double-SPI, quad-SPI, IIC, IIS, and LIN interfaces
- Controller Area Network interface (MultiCAN+), Full-CAN/Basic-CAN with two nodes, 32 message objects (MO)

Analog Front-End Peripherals

- A/D Converters, up to 12 analog inputs, includes two sample and hold stages and a fast 12-bit analog-to-digital converter (ADC) with adjustable gain
- Up to eight channels of out of range comparators (ORC)

- Up to three fast analog comparators (ACMP)
- Temperature Sensor (DTS)

Industrial Control Peripherals

- Capture/Compare Units 4 (CCU4) timers for signal monitoring and PWM
- Capture/Compare Units 8 (CCU8) timers for complex PWM, complementary high/low-side switches, and multi-phase control
- Position Interfaces (POSIF) for hall and quadrature encoders, and motor positioning
- Brightness and Colour Control Unit (BCCU) for LED lighting applications

System Control

- Window Watchdog Timer (WDT) for safety sensitive applications
- Real-Time Clock (RTC) module with alarm support
- System Control Unit (SCU) for system configuration and control
- Pseudo-random number generator (PRNG) provides random data with fast generation times

Input/Output Lines With Individual Bit Controllability

- Tri-stated in input mode
- Push/pull or open drain output mode
- Configurable pad hysteresis

Debug System

- Access through the standard Arm® serial wire debug (SWD) or the single pin debug (SPD) interface
- A breakpoint unit (BPU) supporting up to four hardware breakpoints
- A watchpoint unit (DWT) supporting up to two watchpoints

Packages Information

- PG-TSSOP-38
- PG-VQFN-40
- PG-VQFN-48

***Note**: For more information on package availability for a particular derivative, check the device datasheet.*

1.1.1 Architecture

Figure 1 shows the functional blocks and their basic connectivity within the PSC1Mx system.

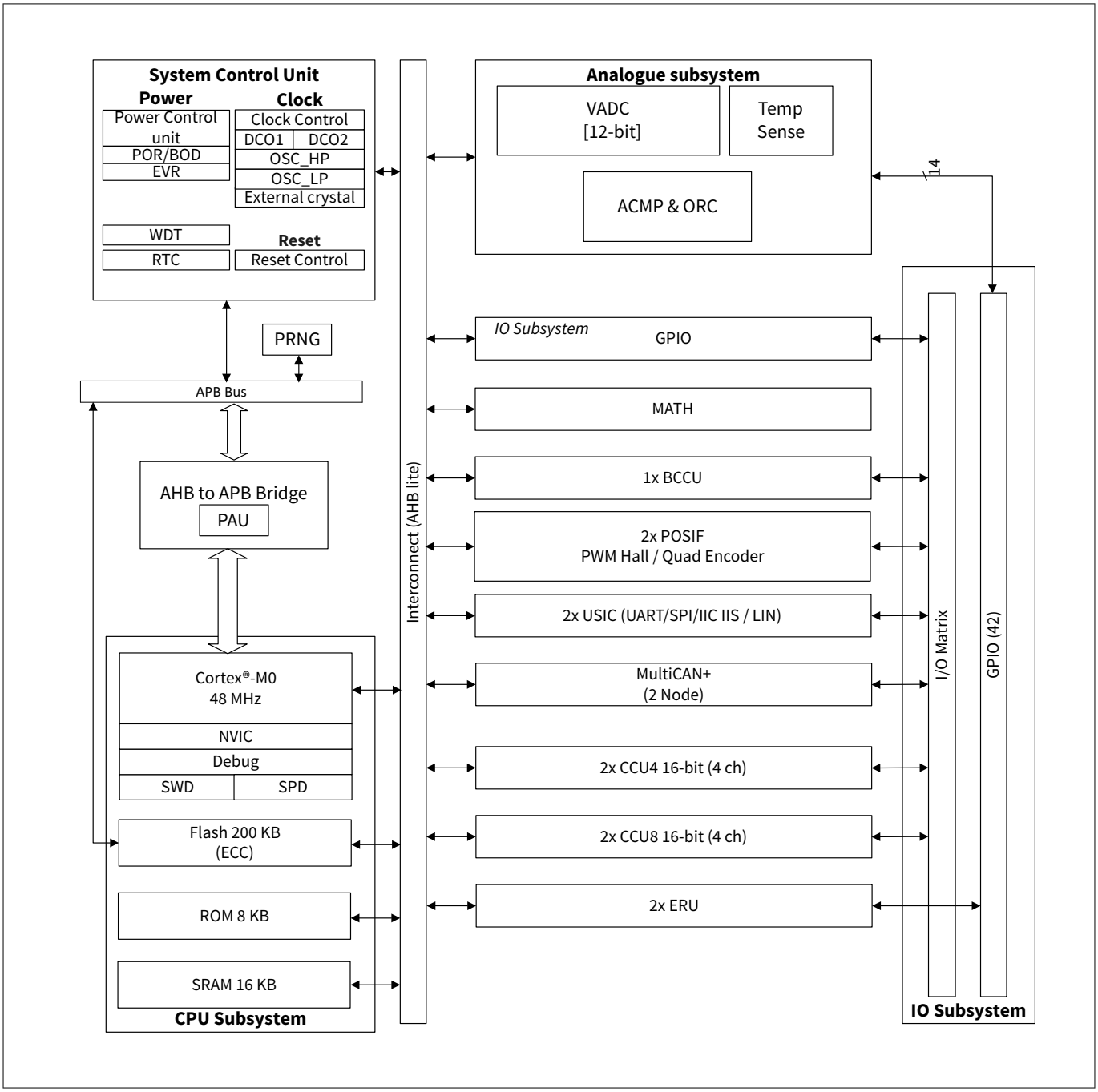


Figure 1 PSC1M2/3 Block Diagram

1.1.2 Device Overview

Table 3 lists the available features per device type for the PSC1M2x/PSC1M3x series.

Table 3 Features of PSC1Mx Device Types[1)]

Features		PSC1M2-T038	PSC1M2-Q040	PSC1M2-Q048	PSC1M2-F048	PSC1M3-Q048	PSC1M3-F048
CPU frequency		48 MHz					
Operating temperature		Ambient temperature -40 to 105°C					
Operating voltage		1.8 V to 5.5 V					
Flash options (Kbytes)		64, 128, 200				64, 128, 200	
SRAM (Kbytes)		16	16	16	16	16	16
MATH		1	1	1	1	1	1
Industrial Control	CCU4	2	2	2	2	2	2
	CCU8	2	2	2	2	2	2
	POSIF	1	1	2	2	2	2
	BCCU	1	1	1	1	1	1
Communication	USIC (modules / channels)	2/4	2/4	2/4	2/4	2/4	2/4
	MultiCAN+ (nodes /MOs)	-	-	-	-	2/32	
Analog	ADC (kernels/ analog inputs)	2/12	2/12	2/12	2/14	2/12	2 / 14
	ACMP	3	3	4	4	4	4
GPIOs		26	27	34	35	34	35
GPIs		8	8	8	8	8	8
Packages		TSSOP-38	VQFN-40	VQFN-48	LQFP-48	VQFN-48	LQFP-48

1) Features that are not included in this table are available in all the derivatives

1.2 CPU Subsystem

The PSC1Mx system core consists of the CPU and the memory interface blocks for memories.

Central Processing Unit (CPU)

The Arm® Cortex®-M0 processor is built on a highly area and power optimized 32-bit processor core, with a 3-stage pipeline von Neumann architecture. The processor delivers exceptional energy efficiency through a small but powerful instruction set and extensively optimized design, providing high-end processing hardware including a single cycle multiplier.

The instruction set is based on the 16-bit Arm® Thumb® instruction set and includes Thumb®-2 technology. This provides the exceptional performance expected of a modern 32-bit architecture, with a higher code density than other 8-bit and 16-bit microcontrollers.

Nested Vectored Interrupt Controller (NVIC)

The PSC1Mx provides separate interrupt nodes that may be assigned to 4 interrupt priority levels. Most interrupt sources are connected to a dedicated interrupt node. 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 subnode control registers.

Math Coprocessor (MATH)

The MATH Coprocessor (MATH) module comprises of two independent sub-blocks to support the CPU in math-intensive computations: a Divider Unit (DIV) for signed and unsigned 32-bit division operations and a Cordinate Rotation DIgital Computer (CORDIC) Coprocessor for computation of trigonometric, linear, or hyperbolic functions.

Prefetch Unit (PFU)

The PFU reduces the Flash latency gap at higher system frequencies to increase the instruction per cycle performance.

1.3 On-chip Memories

Various types of dedicated on-chip memories are available. 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 to fulfill application-specific requirements.

Boot ROM (BROM)

8 KB of ROM memory for boot code execution and exception vector table. The ROM contains system basic initialization sequence code and is executed immediately after reset release. The Bootstrap Loaders (BSL) and User Routines are also stored in the ROM.

Flash memory

Up to 200 KB of on-chip Flash memory store code or constant data. Dynamic error correction provides high read data security for all read accesses.

SRAM memory

16 KB of on-chip code RAM (SRAM) is provided to store the user code or data, as well as system variables such as system stack. The SRAM is accessed via the AHB and provides zero-waitstate access for CPU code execution.

1.4 Communication Peripherals

Communication features are key requirements in today’s industrial systems. The PSC1Mx offers a set of peripherals supporting advanced communication protocols. Besides CAN and the USIC, the PSC1Mx provides interfaces to various memories as well as a unit to realize a human-machine interface via LED and Touch Sense.

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 that 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 (MultiCAN)

The MultiCAN module contains two 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 to CAN specification ISO11898-1. Each CAN node can receive and transmit standard frames with 11-bit identifiers as well as extended frames with 29-bit identifiers.

All two 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 set up a FIFO buffer.

1.5 Analog Front-End Peripherals

The PSC1Mx hosts a number of interfaces to connect to the analog world.

Analog-to-Digital Converter (ADC)

The versatile ADC (VADC) module consists of an independent kernel that operates according to the successive approximation principle (SAR). The resolution is programmable from 8 to 12 bit.

The kernel provides a versatile state machine allowing complex measurement sequences. The kernel 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.

Analog Comparator (ACMP) and Out-of-Range Comparator (ORC)

The Analog Comparator is used to compare the voltage of two analog inputs and a digital output indicating that input voltage is higher. One of the input can either be the internal reference voltage or from an external pin. A low power mode is available to help to reduce the total power consumption.

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

Temperature Sensor (DTS)

The Temperature Sensor generates a measurement result that indicates directly the die temperature. It is also capable of generating interrupt requests when the temperature measurement crosses the selectable upper/ lower threshold value.

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.

Position Interface (POSIF) Unit

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.

Brightness and Color Control Unit (BCCU)

The BCCU is a dimming control peripheral for LED lighting applications that is capable of controlling multiple LED channels. A 1-bit sigma-delta bit stream is provided for every channel that determines the brightness. The brightness can be changed gradually along an exponential curve to appear natural to the human eye by using dedicated dimming engines. The module supports color control by adjusting the relative intensity of selected channels using a linear walk scheme for smooth color changes, and it also supports high-power multi-channel LED lamps by optionally “packing” the bitstream to provide a defined ON-time at the output.

1.7 On-chip Debug Support

The on-chip debug system based on the Arm® Cortex®-M0 debug system provides a broad range of debug and emulation features built into the PSC1Mx. The user software running on the PSC1Mx can thus be debugged within the target system environment.

The Debug unit is controlled by an external debugging tool via the debug interface. The debugger controls the Debug unit via a set of dedicated registers accessible via the debug interface. Additionally, the Debug unit can be controlled by the CPU, for example, by a monitor program.

Multiple breakpoints can be triggered by on-chip hardware or by software. Single stepping is supported as well as the injection of arbitrary instructions and read/write access to the complete internal address space. A breakpoint trigger can be answered with a CPU-halt, a monitor call or a data transfer.

The data transferred at a watchpoint (see above) can be obtained via the debug interface for increased performance.

2 Central Processing Unit (CPU)

PSC1Mx features the Arm® Cortex®-M0 processor. An entry-level 32-bit Arm® Cortex® processor designed for a broad range of embedded applications. This CPU offers significant benefits to users, including:

- a simple architecture that is easy to learn and program
- ultra-low power, energy-efficient operation
- excellent code density
- deterministic, high-performance interrupt handling
- upward compatibility with Cortex®-M processor family

References to Arm® Documentation

The following documents can be accessed through http://infocenter.arm.com
[1] Cortex®-M0 Devices, Generic User Guide (Arm® DUI 0467B)
[2] Arm®v6-M Architecture Reference Manual (Arm® DDI 0419)
[3] Cortex® Microcontroller Software Interface Standard (CMSIS)

References to Arm® Figures

[4] http://www.arm.com

2.1 Overview

The Cortex®-M0 processor is built on a highly area and power optimized 32-bit processor core, with a 3-stage pipeline von Neumann architecture. The processor delivers exceptional energy efficiency through a small but powerful instruction set and extensively optimized design, providing high-end processing hardware including a single-cycle multiplier.

The Cortex®-M0 processor implements the Arm®v6-M architecture, which is based on the 16-bit Thumb® instruction set and includes Thumb®-2 technology. The Cortex®-M0 instruction set provides exceptional performance expected of a modern 32-bit architecture, with a higher code density than other 8-bit and 16-bit microcontrollers.

The Cortex®-M0 processor closely integrates a configurable NVIC, to deliver industry leading interrupt performance. The NVIC provides 4 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 abandon and restart load-multiple and store-multiple operations. Interrupt handlers do not require any assembler wrapper code, removing any code overhead from the ISRs. Tail-chaining 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.

2.1.1 Features

The CPU provides the following functionality:

- Thumb® instruction set combines high code density with 32-bit performance
- integrated sleep modes for low power consumption
- fast code execution permits slower processor clock or increases sleep mode time
- single cycle 32-bit hardware multiplier
- high-performance interrupt handling for time-critical applications
- extensive debug capabilities:
  - Serial Wire Debug and Single Pin Debug reduce the number of pins required for debugging.

2.1.2 Block Diagram

The Cortex®-M0 core components comprise of:

Processor Core

The CPU provides most 16-bit Thumb® instruction set and subset of 32-bit Thumb®2 instruction set.

Nested Vectored Interrupt Controller

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

Debug Solution

The PSC1Mx implements a complete hardware debug solution.

- Single Pin Debug (SPD) or 2-pin Serial Wire Debug (SWD)
- Extensive hardware breakpoint and watchpoint options

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

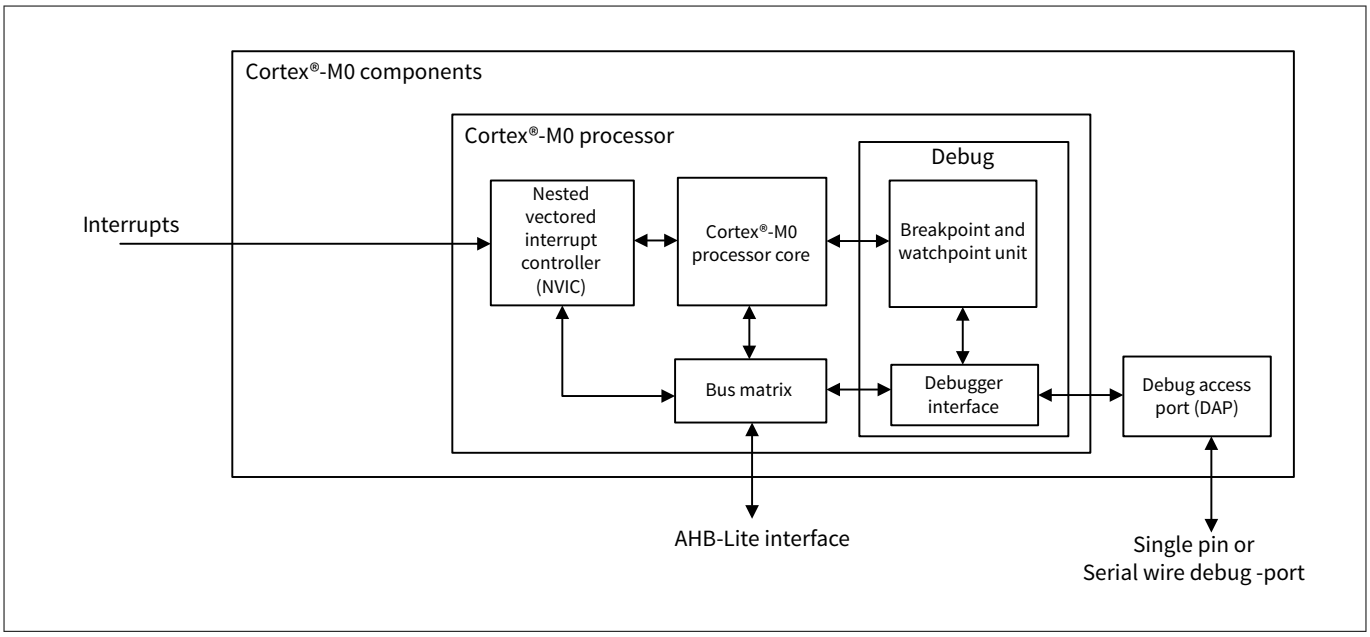


Figure 2 Cortex®-M0 Block Diagram

System Level Interface

The Cortex®-M0 processor provides a single system-level interface using AMBA® technology to provide high speed, low latency memory accesses.

2.2 Programmers Model

This section describes the Cortex®-M0 programmers model. In addition to the individual core register descriptions, it contains information about the processor modes and stacks.

2.2.1 Processor Mode

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.

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.

In Thread mode, the CONTROL register controls whether the processor uses the main stack or the process stack, see CONTROL Register. In Handler mode, the processor always uses the main stack. The options for processor operations are:

Table 4 Summary of processor mode, execution, and stack use options

Processor mode	Used to execute	Stack used
Thread	Applications	Main stack or process stack[1)]
Handler	Exception handlers	Main stack

1) See CONTROL Register.

2.2.3 Core Registers

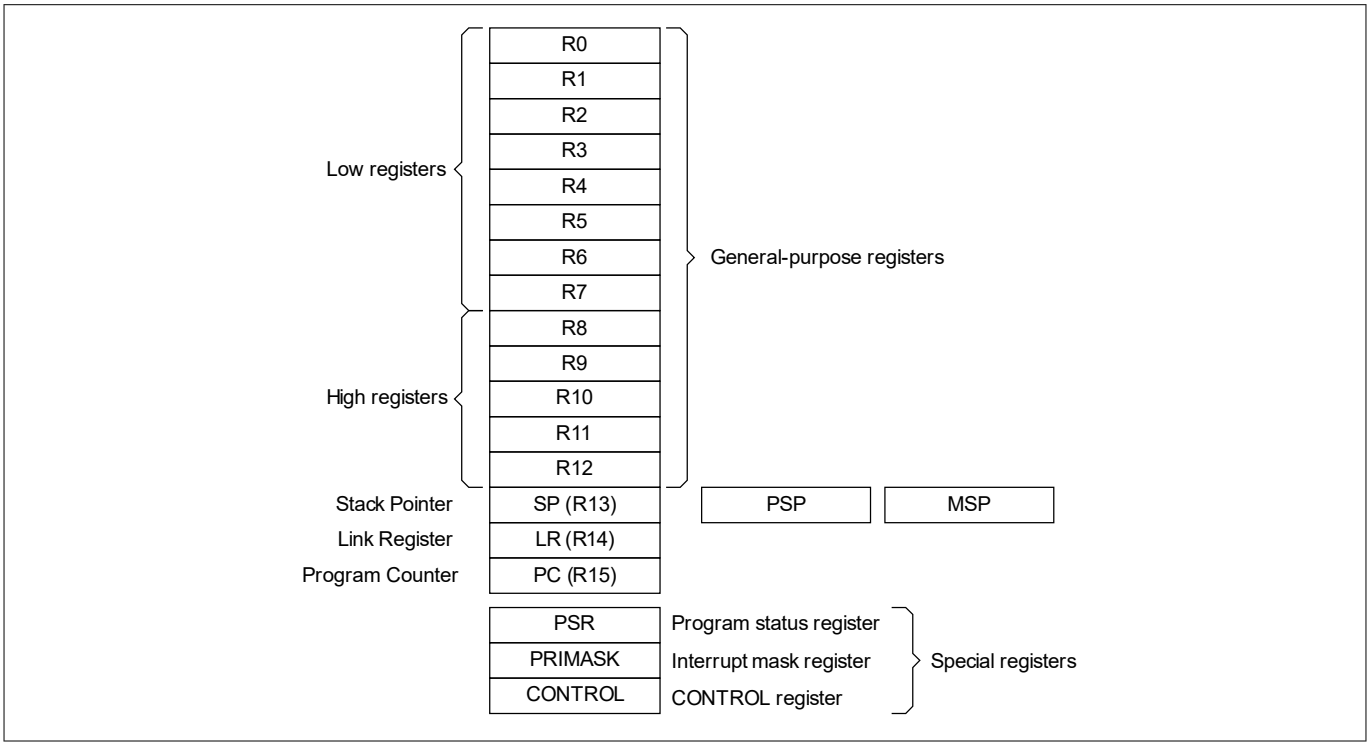


Figure 3 Core registers

The processor core registers are:

Table 5 Core register set summary

Name	Type[1)]	Reset value	Description
R0-R12	rw	Unknown	General-purpose registers
MSP	rw	See description	Stack Pointer
PSP	rw	Unknown	Stack Pointer
LR	rw	Unknown	Link Register
PC	rw	See description	Program Counter
PSR	rw	Unknown	Program Status Register
APSR	rw	Unknown	Application Program Status Register
IPSR	r	00000000_H	Interrupt Program Status Register
EPSR	r	Unknown	Execution Program Status Register
PRIMASK	rw	00000000_H	Priority Mask Register
CONTROL	rw	00000000_H	CONTROL Register

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

2.2.3.1 General-purpose registers

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

***Note**: For information on how to program the core registers, please refer to the Arm® v6-M Architecture Reference Manual [2].*

Rx (x=0-12)

General-purpose register Rx **Reset value:** XXXXXXXX_H

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

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Value															
rw															

Field	Bits	Type	Description
Value	31:0	rw	**Content of Register**

2.2.3.2 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.

***Note**: For information on how to program the core registers, please refer to the Arm® v6-M Architecture Reference Manual [2].*

SP

Stack Pointer **Reset value:** 00000000_H

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

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Value															
rw															

Field	Bits	Type	Description
Value	31:0	rw	**Content of Register**

2.2.3.3 Link Register

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

***Note**: For information on how to program the core registers, please refer to the Arm® v6-M Architecture Reference Manual [2].*

LR

Link Register **Reset value:** XXXXXXXX_H

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

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Value															
rw															

Field	Bits	Type	Description
Value	31:0	rw	**Content of Register**

2.2.3.4 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 00000004_H. Bit [0] of the value is loaded into the EPSR T-bit at reset and must be 1.

***Note**: For information on how to program the core registers, please refer to the Arm® v6-M Architecture Reference Manual [2].*

PC

Program Counter **Reset value:** 00000004_H

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

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Value															
rw															

Field	Bits	Type	Description
Value	31:0	rw	**Content of Register**

2.2.3.5 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, and V bits using APSR with the MSR instruction

The PSR combinations and attributes are:

Table 6 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

2.2.3.6 Application Program Status Register

The APSR contains the current state of the condition flags from previous instruction executions. See the register summary in Table 5 for its attributes.

APSR

Application Program Status Register **Reset value:** XXXXXXXX_H

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

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

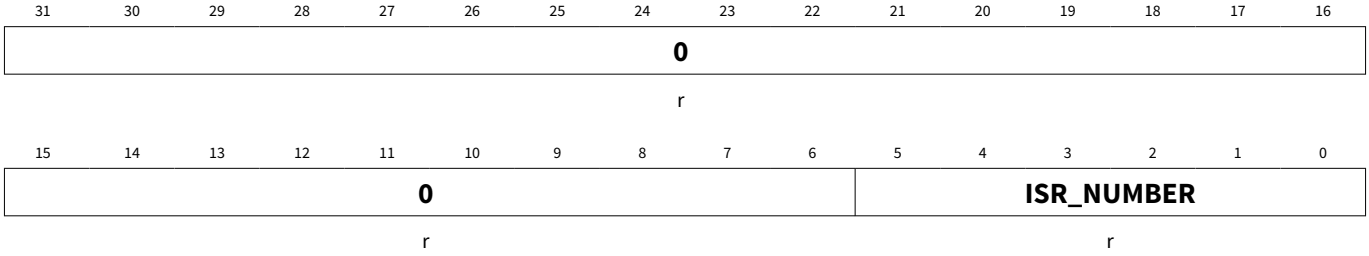
Field	Bits	Type	Description
N	31	rw	**Negative flag**
Z	30	rw	**Zero flag**
C	29	rw	**Carry or borrow flag**
V	28	rw	**Overflow flag**
0	27:0	r	**Reserved**

2.2.3.7 Interrupt Program Status Register

The IPSR contains the exception type number of the current Interrupt Service Routine (ISR). See the register summary in Table 5 for its attributes.

IPSR

Interrupt Program Status Register **Reset value:** 00000000_H



Field	Bits	Type	Description
0	31:6	r	**Reserved**
ISR_NUMBER	5:0	r	**Number of the current exception** 0_D Thread mode 1_D Reserved 2_D Reserved 3_D HardFault 4_D Reserved 5_D Reserved 6_D Reserved 7_D Reserved 8_D Reserved 9_D Reserved 10_D Reserved 11_D SVCall 12_D Reserved 13_D Reserved 14_D PendSV 15_D SysTick 16_D IRQ0 ... 47_D IRQ31 48_D-63_D Reserved See Exception Types for more information.

2.2.3.8 Execution Program Status Register

The EPSR contains the Thumb® state bit.

See the register summary in Table 5 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. Fault handlers can examine the EPSR value in the stacked PSR to determine the cause of the fault. See Exception Entry and Return.

EPSR

Execution Program Status Register **Reset value:** XXXXXXXX_H

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

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

Field	Bits	Type	Description
0	31:25	r	**Reserved**
T	24	r	**Thumb® state bit** See Thumb® state.
0	23:0	r	**Reserved**

Interruptible-restartable instructions

When an interrupt occurs during the execution of an LDM, STM, PUSH, POP instruction, the processor abandons execution of the instruction.

After servicing the interrupt, the processor restarts execution of the instruction from the beginning.

Thumb® state

The Cortex®-M0 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.

Attempting to execute instructions when the T bit is 0 results in a HardFault or lockup. See Lockup 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 or code sequences requiring atomicity.

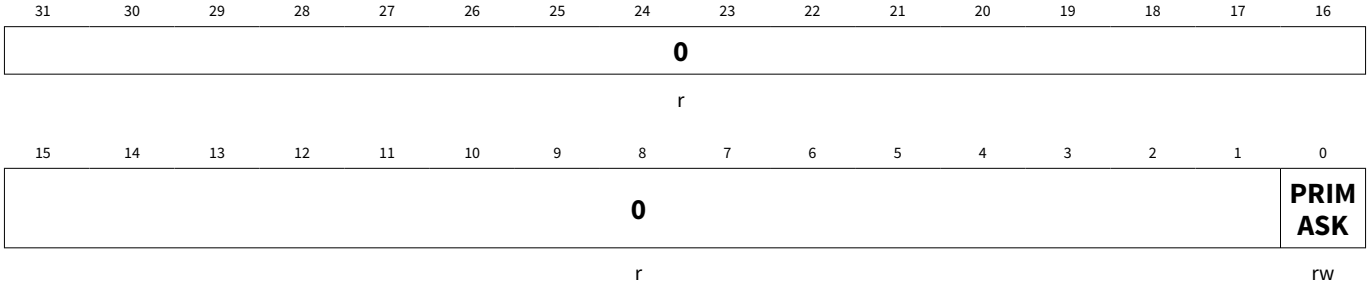
Exceptions can be disabled or re-enabled by the MSR and MRS instructions, or the CPS instruction, to change the value of PRIMASK or FAULTMASK.

2.2.3.9 Priority Mask Register

The PRIMASK register prevents activation of all exceptions with configurable priority. See the register summary in Table 5 for its attributes.

PRIMASK

Priority Mask Register **Reset value:** 00000000_H



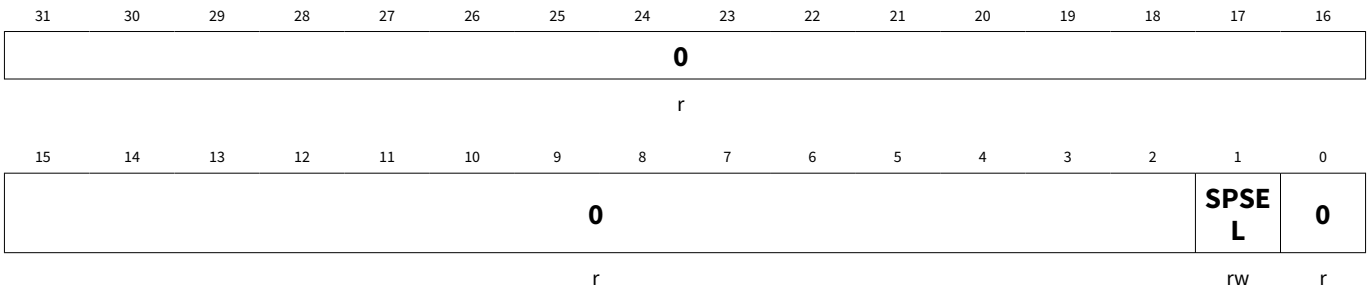
Field	Bits	Type	Description
0	31:1	r	**Reserved**
PRIMASK	0	rw	**Priority Mask** 0_B No effect. 1_B Prevents the activation of all exceptions with configurable priority.

2.2.3.10 CONTROL Register

The CONTROL register controls the stack used when the processor is in Thread mode. See the register summary in Table 5 for its attributes.

CONTROL

CONTROL Register **Reset value:** 00000000_H



Field	Bits	Type	Description
0	31:2	r	**Reserved**
SPSEL	1	rw	**Active stack pointer** This bit defines the current stack. In Handler mode, this bit reads as zero and ignores writes. 0_B MSP is the current stack pointer 1_B PSP is the current stack pointer
0	0	r	**Reserved**

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.

In an OS environment, it is recommended 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, use the MSR instruction to set the Active stack pointer bit to 1.

***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®-M0 processor supports interrupts and system exceptions. The processor and the NVIC prioritize and handle all exceptions. An interrupt or exception changes the normal flow of software control. The processor uses handler mode to handle all exceptions except for reset. See Exception entry and Exception return for more information.

The NVIC registers control interrupt handling. See Interrupt System chapter 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 for more information.

2.2.6 The Cortex® Microcontroller Software Interface Standard

For a Cortex®-M0 microcontroller system, the Cortex® Microcontroller Software Interface Standard (CMSIS) [3] 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.

The CMSIS includes address definitions and data structures for the core peripherals in the Cortex®-M0 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
- CMSIS Functions
- Accessing CPU Registers using CMSIS in Interrupt System chapter
- NVIC programming hints in Interrupt System chapter

For additional information please refer to https://www.arm.com/technologies/cmsis

2.2.7 CMSIS Functions

ISO/IEC C code cannot directly access some Cortex®-M0 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, an inline assembler may be used to access the relevant instruction.

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

Table 7 CMSIS functions to generate some Cortex®-M0 instructions

Instruction	CMSIS intrinsic function
CPSIE i	void __enable_irq (void)
CPSID i	void __disable_irq (void)
ISB	void __ISB (void)
DSB	void __DSB (void)
DMB	void __DMB (void)
NOP	void __NOP (void)
REV	uint32_t __REV (uint32_t int value)
REV16	uint32_t __REV16 (uint32_t int value)
REVSH	uint32_t __REVSH (uint32_t int value)
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 8 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)

(table continues...)

Table 8 (continued) CMSIS functions to access the special registers

Special register	Access	CMSIS function
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 TopOfMainStack)

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:

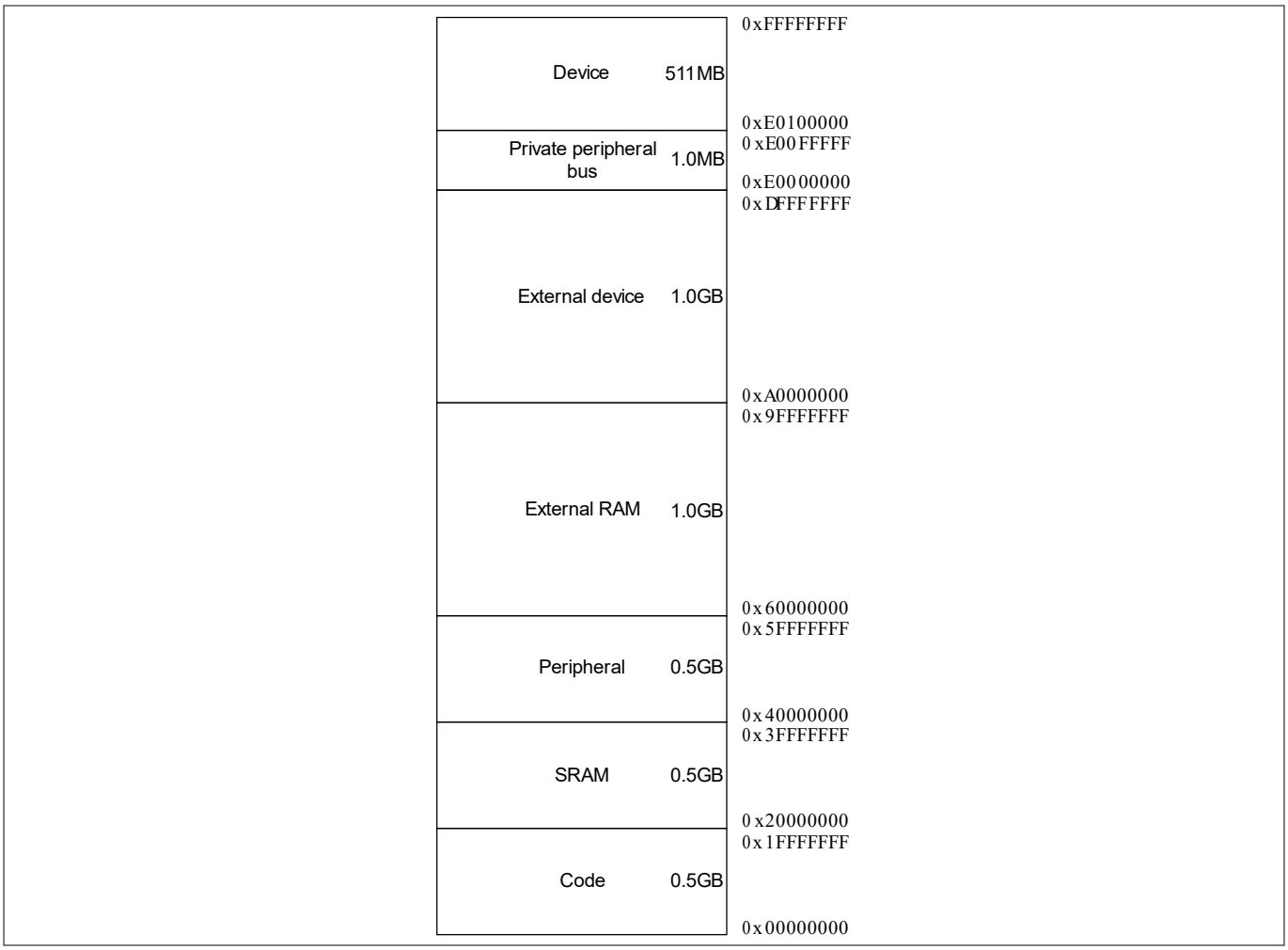


Figure 4 Memory map

The processor reserves regions of the Private peripheral bus (PPB) address range for core peripheral registers, see About the Private Peripherals.

2.3.1 Memory Regions, Types and Attributes

The memory map is split 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 HardFault exception is generated on execution of an instruction fetched from an XN region of memory.

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 any re-ordering 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.

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 described in Figure 5.

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

Figure 5 Memory system ordering

Where:

- “-” Means that the memory system does not guarentee 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 9 Memory access behavior

Address range	Memory region	Memory type[1)]	XN[1)]	Description
0x00000000-0x1FFFFFFF	Code	Normal	-	Executable region for program code. Data can be placed here.
0x20000000-0x3FFFFFFF	SRAM	Normal	-	Executable region for data. Code can be placed 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. Only word accesses can be used in this region.
0xE0100000-0xFFFFFFFF	Device	Device	XN	Vendor specific

1) See Memory Regions, Types and Attributes for more information.

The Code, SRAM, and external RAM regions can hold programs.

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.
- memory or devices in the memory map have different wait states
- some memory accesses are buffered or speculative.

Memory System Ordering of Memory Accesses 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.

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. Little-endian format describes how words of data are stored in memory.

2.3.5.1 Little-endian format

In little-endian format, the processor stores the least significant byte (lsbyte) of a word at the lowest-numbered byte, and the most significant byte (msbyte) at the highest-numbered byte. An example of the little-endian format is described in Figure 6.

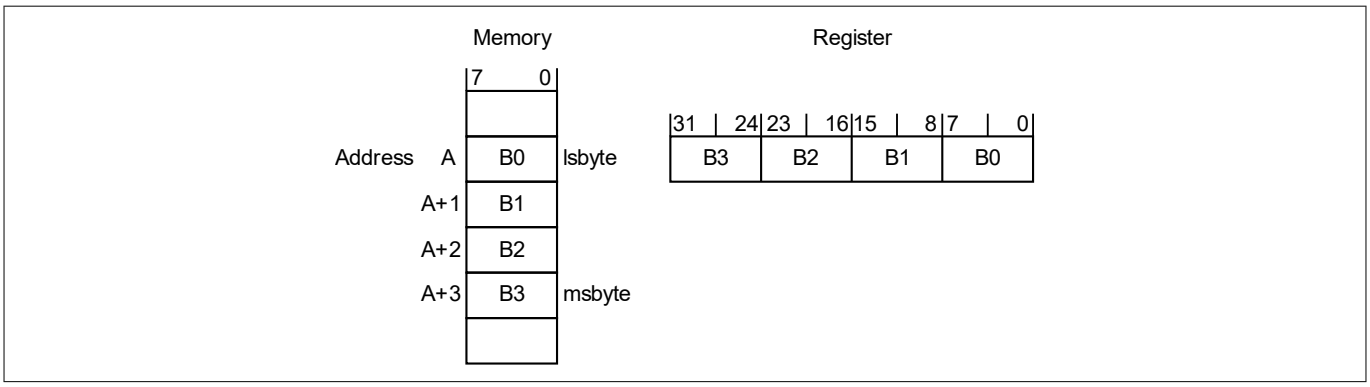


Figure 6 Little-endian format (Example)

2.4 Instruction Set

Table 10 lists the supported Cortex®-M0 instructions. For more information on the instructions and operands, please refer to the Cortex®-M0 Devices, Generic User Guide available through [1].

Table 10 Cortex®-M0 instructions

Mnemonic	Operands	Brief description	Flags
ADCS	{Rd,} Rn, Rm	Add with carry	N,Z,C,V
ADD{S}	{Rd,} Rn, <Rm|#imm>	Add	N,Z,C,V
ADR	Rd, label	PC-relative Address to Register	-
ANDS	{Rd,} Rn, Rm	Bitwise AND	N,Z
ASRS	{Rd,} Rm, <Rs|#imm>	Arithmetic Shift Right	N,Z,C
B{cc}	label	Branch {conditionally}	-
BICS	{Rd,} Rn, Rm	Bit Clear	N,Z
BKPT	#imm	Breakpoint	-
BL	label	Branch with Link	-
BLX	Rm	Branch indirect with Link	-
BX	Rm	Branch indirect	-
CMN	Rn, Rm	Compare Negative	N,Z,C,V
CMP	Rn, <Rm|#imm>	Compare	N,Z,C,V

(table continues...)

Table 10 (continued) Cortex®-M0 instructions

Mnemonic	Operands	Brief description	Flags
CPSID	i	Change Processor State, Disable Interrupts	-
CPSIE	i	Change Processor State, Enable Interrupts	-
DMB	-	Data Memory Barrier	-
DSB	-	Data Synchronization Barrier	-
EORS	{Rd,} Rn, Rm	Exclusive OR	N,Z
ISB	-	Instruction Synchronization Barrier	-
LDM	Rn{!}, reglist	Load Multiple registers, increment after	-
LDR	Rt, label	Load Register from PC- relative address	-
LDR	Rt, [Rn, <Rm|#imm>]	Load Register with word	-
LDRB	Rt, [Rn, <Rm|#imm>]	Load Reigster with byte	-
LDRH	Rt, [Rn, <Rm|#imm>]	Load Reigster with halfword	-
LDRSB	Rt, [Rn, <Rm|#imm>]	Load Reigster with signed byte	-
LDRSH	Rt, [Rn, <Rm|#imm>]	Load Reigster with signed halfword	-
LSLS	{Rd,} Rn, <Rs|#imm>	Logical Shift Left	N,Z,C
LSRS	{Rd,} Rn, <Rs|#imm>	Logical Shift Right	N,Z,C
MOV{S}	Rd, Rm	Move	N,Z
MRS	Rd, spec_reg	Move to general register from special register	-
MSR	spec_reg, Rm	Move to special register from general register	N,Z,C,V
MULS	Rd, Rn, Rm	Multiply, 32-bit result	N,Z
MVNS	Rd, Rm	Bitwise NOT	N,Z
NOP	-	No Operation	-
ORRS	{Rd,} Rn, Rm	Logical OR	N,Z
POP	reglist	Pop registers from stack	-
PUSH	reglist	Push registers onto stack	-
REV	Rd, Rm	Byte-Reverse word	-
REV16	Rd, Rm	Byte-Reverse packed halfwords	-
REVSH	Rd, Rm	Byte-Reverse signed halfword	-
RORS	{Rd,} Rn, Rs	Rotate Right	N,Z,C
RSBS	{Rd,} Rn, #0	Reverse Subtract	N,Z,C,V

(table continues...)

Table 10 (continued) Cortex®-M0 instructions

Mnemonic	Operands	Brief description	Flags
SBCS	{Rd,} Rn, Rm	Subtract with Carry	N,Z,C,V
STM	Rn!, reglist	Store Multiple registers, increment after	-
STR	Rt, [Rn, <Rm|#imm>]	Store Register as word	-
STRB	Rt, [Rn, <Rm|#imm>]	Store Register as byte	-
STRH	Rt, [Rn, <Rm|#imm>]	Store Register as halfword	-
SUB{S}	{Rd,} Rn, <Rm|#imm>	Subtract	N,Z,C,V
SVC	#imm	Supervisor Call	-
SXTB	Rd, Rm	Sign extend byte	-
SXTH	Rd, Rm	Sign extend halfword	-
TST	Rn, Rm	Logical AND based test	N,Z
UXTB	Rd, Rm	Zero extend a byte	-
UXTH	Rd, Rm	Zero extend a halfword	-
WFE	-	Wait for Event	-
WFI	-	Wait for Interrupt	-

2.4.1 Intrinsic Functions

ISO/IEC C code cannot directly access some Cortex®-M0 instructions. The intrinsic functions that can generate these instructions, provided by the CMSIS and might be provided by a C compiler are described in CMSIS Functions.

2.5 Exception Model

This section describes the exception model. It describes:

- Exception States
- Exception Types
- Exception Handlers
- Vector Table
- Exception Priorities
- Exception Entry and Return

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 described in Table 11.

Table 11 Exception types

Exception Types	Descriptions
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 in Thread mode.
HardFault	A HardFault is an exception that occurs because of an error during normal or exception processing. HardFaults have a fixed priority of -1, meaning they have higher priority than any exception with configurable priority.
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 12 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	-	Reserved	-	-	-

(table continues...)

Table 12 (continued) Properties of the different exception types

Exception number[1]	IRQ number[1]	Exception type	Priority	Vector address or offset[2]	Activation
3	-13	HardFault	-1	0x0000000C	Synchronous
4-10	-	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[3]	0x00000040 and above[4]	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.
2) See Vector Table for more information.
3) See Interrupt Priority Registers in Interrupt System chapter.
4) Increasing in steps of 4.

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

Software can disable the exceptions in Table 12 which have configurable priority, see Interrupt Clear-enable Register in Interrupt System chapter.

For more information about HardFaults, see Fault Handling.

2.5.3 Exception Handlers

The processor handles exceptions using:

Interrupt Service Routines (ISRs) Interrupts IRQ0 to IRQ31 are the exceptions handled by ISRs.

Fault handlers HardFault is the only exception handled by the fault handler.

System handlers PendSV, SVCall, SysTick, and the HardFault 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 7 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 written in Thumb® code.

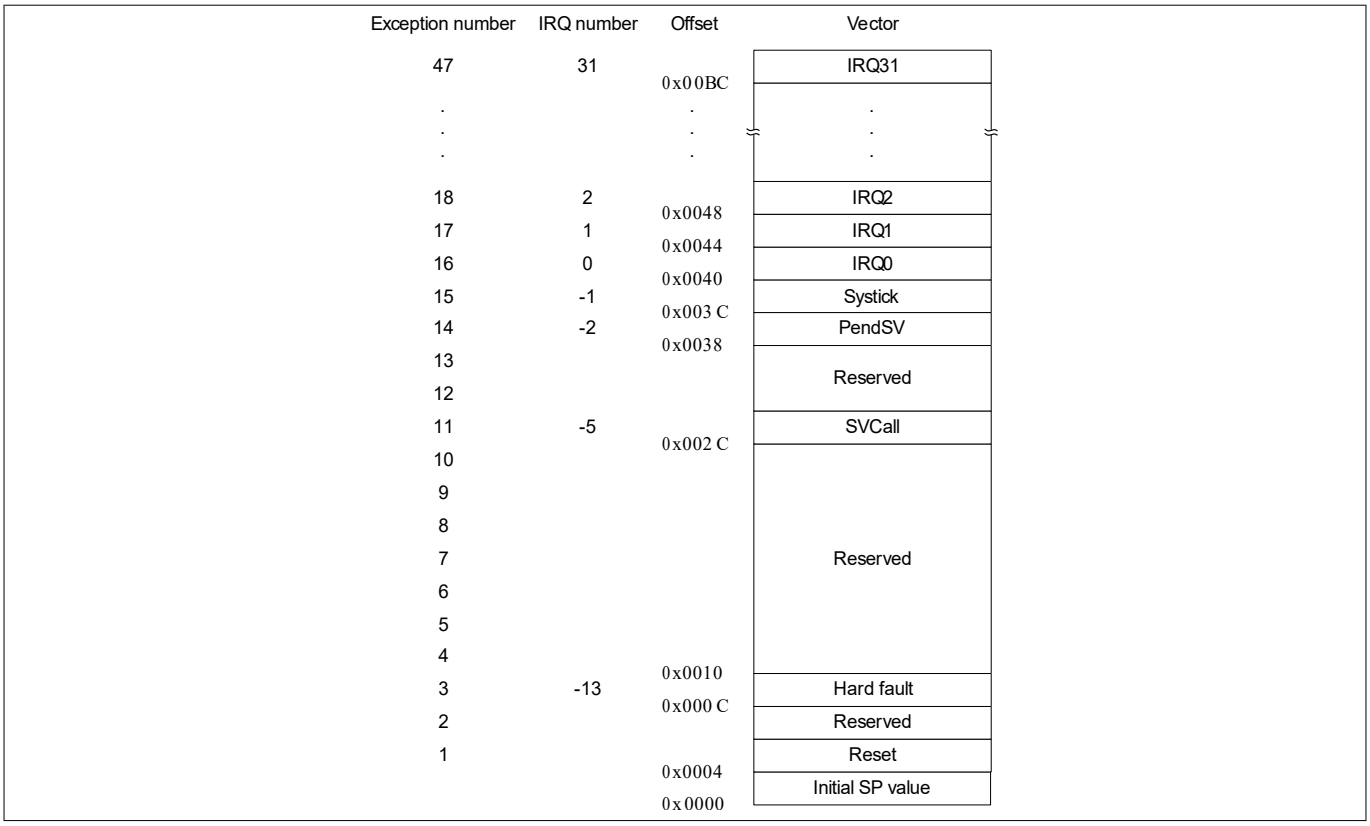


Figure 7 Vector table

The vector table is fixed at address 0x00000000.

2.5.4.1 Vector Table Remap

In PSC1Mx, the vector table is located inside the ROM. Therefore, the vector table is remapped to the SRAM based on the mapping shown in Table 13. The user application uses these locations as entry points for the actual exception and interrupt handlers. This is done by placing the code for these handlers or having the branch instruction to the handlers there.

For example, upon an exception entry due to IRQ0, the processor reads the intermediate handler start address $2000'0040_H$ (fixed in ROM) from the vector table and starts execution from there. If the actual handler is located in another address location due to size constraints, the address $2000'0040_H$ should trigger a load and a branch instruction to jump to this new location.

***Note**: The user application needs to reserve the SRAM addresses $2000'000C_H$ - $2000'00BF_H$ for the remapped vector table if all vectors are used.*

Table 13 Remapped Vector Table

Exception Number	IRQ Number	Vector	Default Vector Address	Remapped Vector Address
-	-	Initial SP Value	$0000'0000_H$	$1000'1000_H$
1	-	Reset	$0000'0004_H$	$1000'1004_H$ [1)]
3	-13	HardFault	$0000'000C_H$	$2000'000C_H$
11	-5	SVCall	$0000'002C_H$	$2000'002C_H$

(table continues...)

Table 13 (continued) Remapped Vector Table

Exception Number	IRQ Number	Vector	Default Vector Address	Remapped Vector Address
14	-2	PendSV	$0000'0038_H$	$2000'0038_H$
15	-1	SysTick	$0000'003C_H$	$2000'003C_H$
16-47	0-31	IRQn (n=0-31)	$0000'0040_H$ + (n*4)	$2000'0040_H$ + (n*4)

1) The remapped reset vector address refers to the location (start of the Flash memory) that the startup software jumps to upon exiting the startup sequence in user mode.

2.5.5 Exception Priorities

Table 12 shows that all exceptions have an associated priority, with:

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

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 SHPR2, SHPR3.
- Interrupt Priority Registers in Interrupt System chapter.

***Note**: Configurable priority values are in the range 0-192, in steps of 64. This means that the Reset and HardFault 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].

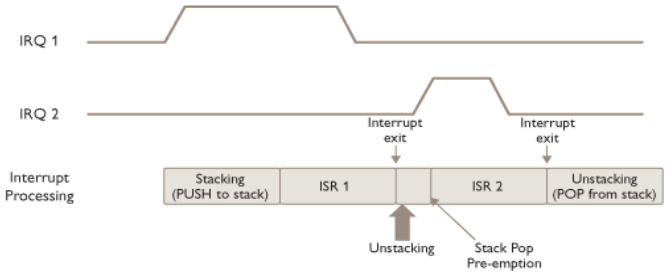
If multiple pending exceptions have the same priority, the pending exception with the lowest exception number takes precedence. For example, if both IRQ[0] and IRQ[1] are pending and have the same priority, then IRQ[0] is processed before IRQ[1].

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 Exception Entry and Return

Exception handling can be described using 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. When one exception preempts another, the exceptions are called nested exceptions. See Exception entry for more information.



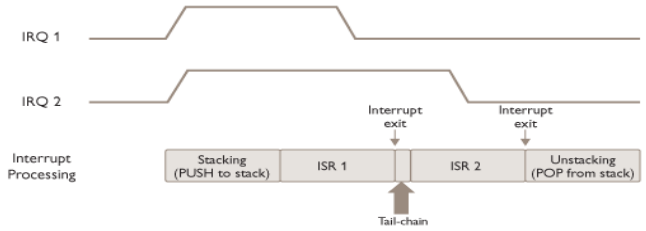
Source of figure [4].

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.

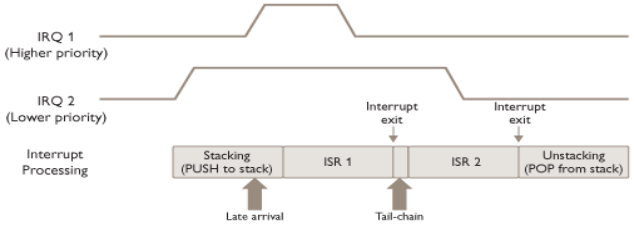
The processor pops the stack and restores the processor state to the state it had before the interrupt occurred. See Exception return for more information.

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 [4].

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. On return from the exception handler of the late-arriving exception, the normal tail-chaining rules apply.



Source of figure [4].

2.5.6.1 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 greater priority than any limits set by the mask register, see Exception mask registers. 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 a stack frame. The stack frame contains the following information, as illustrated in Figure 8.

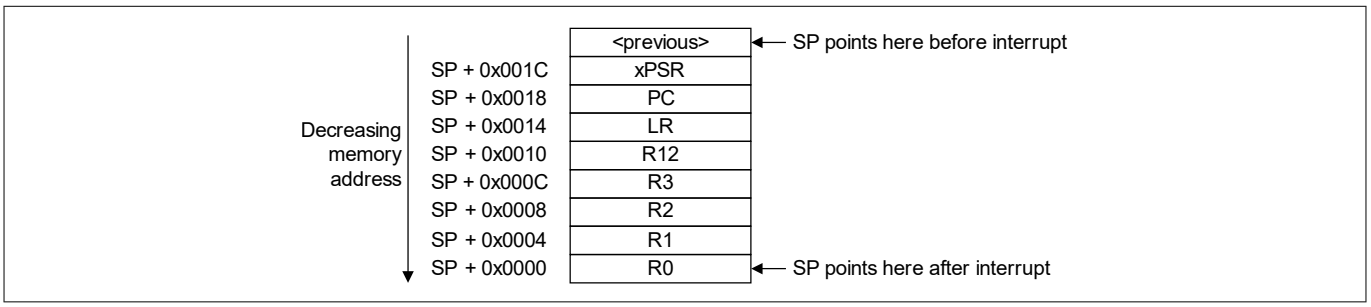


Figure 8 Exception stack frame

Immediately after stacking, the stack pointer indicates the lowest address in the stack frame. The stack frame is aligned to a double-word address.

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.

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.

2.5.6.2 Exception return

Exception return occurs when the processor is in Handler mode and execution of one of the following instructions attempts to set the PC to an EXC_RETURN value:

- a POP instruction that loads the PC
- a BX instruction using any register.

The processor saves an EXC_RETURN value to the LR on exception entry. The exception mechanism relies on this value to detect when the processor has completed an exception handler. Bits [31:4] of an EXC_RETURN value are set to 1. When this value is loaded into the PC, the processor detects that the exception is complete, and starts the exception return sequence. Bits [3:0] of the EXC_RETURN value indicate the required return stack and processor mode. Table 14 shows the EXC_RETURN values with description of the exception return behavior.

Table 14 Exception return behavior

EXC_RETURN[31:0]	Description
0xFFFFFFF1	Return to Handler mode. Exception return gets state from the main stack. Execution uses MSP after return.
0xFFFFFFF9	Return to Thread mode. Exception return gets state from MSP. Execution uses MSP after return.
0xFFFFFFFD	Return to Thread mode. Exception return gets state from the PSP. Execution uses PSP after return.
All other values	Reserved.

2.6 Fault Handling

Faults are a subset of the exceptions, see Exception model in Exception Model. All faults result in the HardFault exception being taken or cause lockup if they occur in the HardFault handler. The faults are:

- execution of an SVC instruction at a priority equal or higher than SVCall
- execution of a BKPT instruction without a debugger attached
- a system-generated bus error on a load or store
- execution of an instruction from an XN memory address
- execution of an instruction from a location for which the system generates a bus fault
- a system-generated bus error on a vector fetch
- execution of an undefined instruction
- execution of an instruction when not in Thumb®-State as a result of the T-bit being previously cleared to 0
- an attempted load or store to an unaligned address

***Note**: Only Reset can preempt the fixed priority HardFault handler. A HardFault can preempt any exception other than Reset, or another hard fault.*

2.6.1 Lockup

The processor enters a lockup state if a fault occurs when executing the HardFault handlers, or if the system generates a bus error when unstacking the PSR on an exception return using the MSP. When the processor is in lockup state it does not execute any instructions. The processor remains in lockup state until one of the following occurs:

- it is reset
- it is halted by a debugger

2.7 Power Management

The Cortex®-M0 processor sleep modes reduce power consumption:

- Sleep mode
- Deep sleep mode

The SLEEPDEEP bit of the SCR selects which sleep mode is used, see System Control Register SCR.

This 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. 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 is asserted. 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 an exception handler and returns to Thread mode, it immediately enters sleep mode. This mechanism is used in applications that only require the processor to run when an interrupt occurs.

2.7.2 Wakeup from Sleep Mode

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

Wakeup from WFI or Sleep-on-exit

The following events are WFI wake-up events:

- reset event
- debug event, if debug is enabled
- exception at a priority that would preempt any currently active exceptions, if PRIMASK was set to 0

Note: *If PRIMASK is set to 1, an interrupt or exception that has a higher priority than the current exception priority will cause the processor to wake up. Interrupt handler is not executed until the processor sets PRIMASK to 0. For more information about PRIMASK, see* Exception mask registers*.*

Wakeup from WFE

The following events are WFE wake-up events:

- reset event
- exception or interrupt with sufficient priority to cause exception entry
- exception or interrupt entering pending state, if SEVONPEND is set to 1
- debug event, if debug is enabled

***Note**:* *Wakeup by Event Register (see The Event Register in Arm® v6-M Architecture Reference Manual [2]) is not possible in PSC1Mx. Nevertheless, SEV instruction will set the event register.*

2.7.3 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®-M0 core peripherals.

2.8.1 About the Private Peripherals

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

Table 15 Core peripheral register regions

Address	Core peripheral	Description
0xE000E008-0xE000E00F	System Control Block	See System control block and SCS Registers
0xE000E010-0xE000E01F	System timer	See System timer, SysTick and SysTick Registers
0xE000E100-0xE000E4EF	Nested Vectored Interrupt Controller	See Interrupt System chapter
0xE000ED00-0xE000ED3F	System Control Block	See System control block and SCS Registers
0xE000EF00-0xE000EF03	Nested Vectored Interrupt Controller	See Interrupt System chapter

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.

2.8.2.1 System control block usage hints and tips

Ensure software uses aligned 32-bit word size transactions to access all the system control block registers.

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 cycle, then counts down on subsequent clock cycles.

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

2.8.3.1 SysTick usage hints and tips

The interrupt controller clock updates the SysTick count. When processor clock is selected and the clock signal is stopped for low power mode, the SysTick counter stops. When external clock is selected, the clock continues to run in low power mode and SysTick can be used as a wakeup source.

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

If 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.

2.9 PPB Registers

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

Table 16 Register Overview

Register Name	Description	Offset Address	Access Mode	
			Read	Write
System Control Space (SCS)				
CPUID	CPUID Base Register	$D00_H$	PV, 32	PV, 32
ICSR	Interrupt Control and State Register	$D04_H$	PV, 32	PV, 32
AIRCR	Application Interrupt and Reset Control Register	$D0C_H$	PV, 32	PV, 32
SCR	System Control Register	$D10_H$	PV, 32	PV, 32
CCR	Configuration and Control Register	$D14_H$	PV, 32	PV, 32
SHPR2	System Handler Priority Register 2	$D1C_H$	PV, 32	PV, 32
SHPR3	System Handler Priority Register 3	$D20_H$	PV, 32	PV, 32
SHCSR	System Handler Control and State Register	$D24_H$	PV, 32	PV, 32
System Timer (SysTick)				
SYST_CSR	SysTick Control and Status Register	010_H	PV, 32	PV, 32
SYST_RVR	SysTick Reload Value Register	014_H	PV, 32	PV, 32
SYST_CVR	SysTick Current Value Register	018_H	PV, 32	PV, 32
SYST_CALIB	SysTick Calibration Value Register	$01C_H$	PV, 32	-

3 Bus System

The single master bus system in PSC1Mx consists of a high-performance system bus based on the industry AMBA® 3 AHB-Lite Protocol standard for memories and high-bandwidth on-chip peripherals and a narrower APB for low-bandwidth on-chip peripherals.

3.1 Bus Interfaces

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

- Memory Interface
- Peripheral Interface

All on-chip modules implement Little Endian data organization.

Memory Interface

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

The memory interface data bus width is 32-bit. Flash memory supports only 32-bit accesses while SRAM allows 32-bit, 16-bit and 8-bit write accesses. Read accesses to SRAM is always 32-bit wide.

Peripheral Interface

Each slave on the AHB-Lite supports 32-bit accesses. Additionally:

- USIC0 supports 8-bit and 16-bit accesses
- MATH coprocessor supports 16-bit accesses for the Divider registers

Each slave on the APB supports only 16-bit accesses.

***Note**: Unaligned memory accesses to memory or peripheral slaves result in a HardFault exception.*

4 Service Request Processing

A hardware pulse is called Service Request (SR) in an PSC1Mx 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
- Peripheral action

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

Table 17 Abbreviations

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. PSC1Mx 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

4.1.2 Block Diagram

Figure 9 shows a representation of the interaction between the request sources and the request processing units.

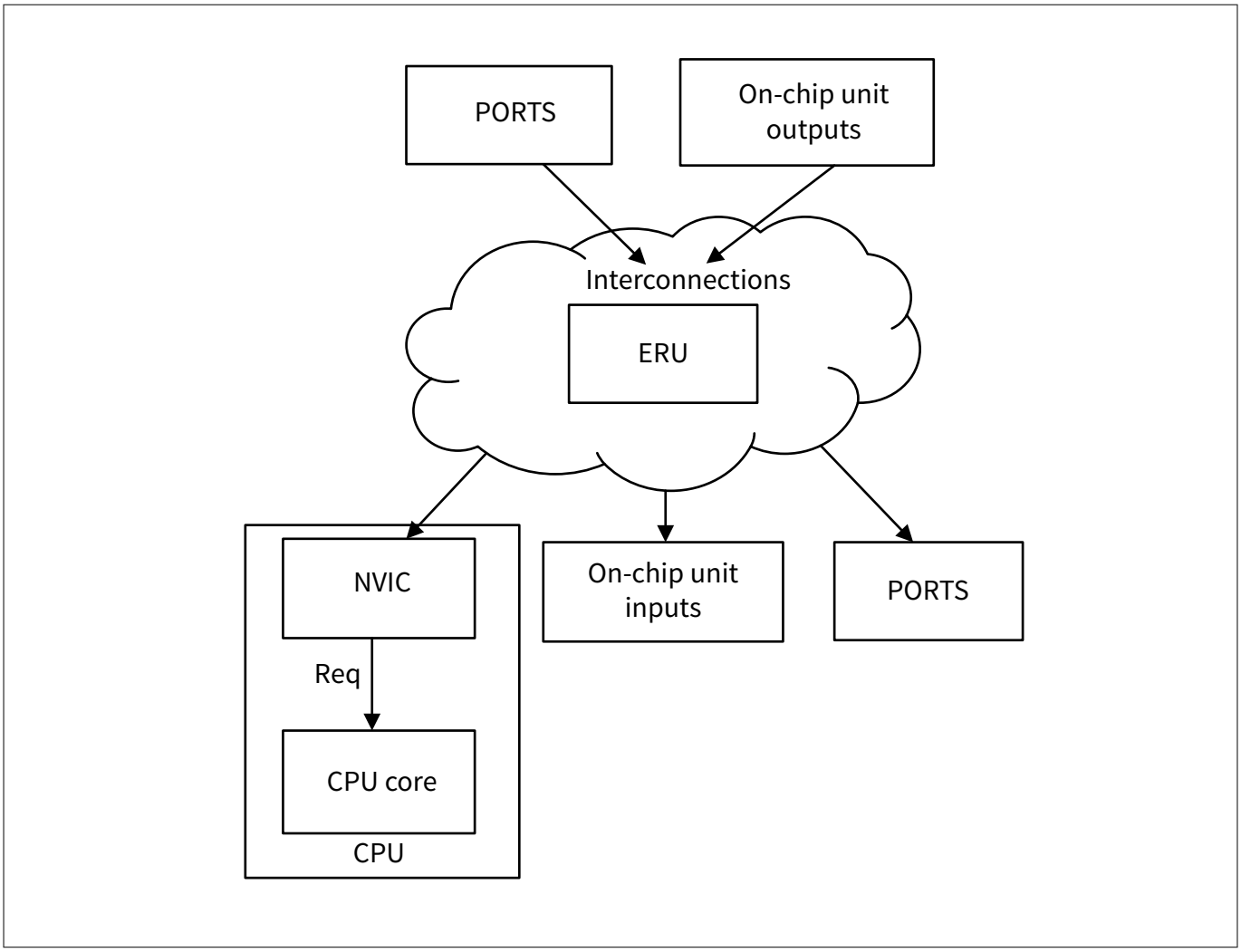


Figure 9 Block Diagram on Service Request Processing

4.2 Service Request Distribution

Figure 10 shows 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.

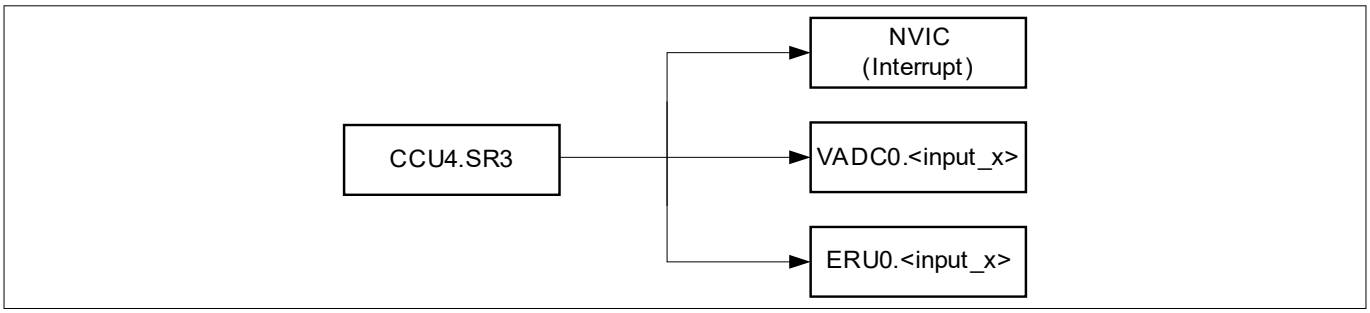


Figure 10 Example for Service Request Distribution

The units involved in Service Request distribution can be subdivided into

- Embedded real time services
- Interrupt services

Embedded real time services

Connectivity between On-Chip Units and PORTS is real time application and also chip package dependent. 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” chapter

Interrupt services

The following table gives an overview on the number of service requests per module and the number that can be assigned to the NVIC Interrupt service provider.

As there are more service requests than available NVIC nodes, a combination of OR logic and multiplexors are implemented for each NVIC node to allow flexible assignment of service requests. Refer to the Interrupt Node Assignment section in the Interrupt Subsystem chapter for details.

Service Requests are always of type “Pulse” in PSC1Mx.

Table 18 Interrupt services per module

Modules	Request Sources	NVIC	Type
VADC	12	6	Pulse
CCU40	4	4	Pulse
CCU41	4	4	Pulse
CCU80	4	2	Pulse
CCU81	4	2	Pulse
POSIF0	2	2	Pulse
POSIF1	2	2	Pulse
USIC0	6	6	Pulse
USIC1	6	6	Pulse
Reserved	3	3	Pulse
BCCU0	1	1	Pulse
MATH	1	1	Pulse
SCU	3	3	Pulse
ERU0	4	4	Pulse
ERU1	4	4	Pulse
MultiCAN+	8	4	Pulse
Total	68	54	-

5 Interrupt Subsystem

The interrupt Subsystem in PSC1Mx consists of the Nested Vectored Interrupt Controller (NVIC) and the respective modules' interrupt generation blocks.

***Note**: The CPU exception model is described in the CPU chapter.*

5.1 Nested Vectored Interrupt Controller (NVIC)

The NVIC is an integral part of the Cortex® M0 processor unit. Due to a tight coupling with the CPU, it provides the lowest interrupt latency and efficient processing of late arriving interrupts.

5.1.1 Features

The NVIC supports the following features:

- 32 interrupt nodes
- 4 programmable priority levels for each interrupt node
- Support for interrupt tail-chaining and late-arrival
- Software interrupt generation

5.1.2 Interrupt Node Assignment

Table 19 lists the service request sources per peripheral and their assignment to NVIC interrupt nodes. For calculation of the vector routine address, please refer to the section on Vector Table in the CPU chapter.

Each node can be assigned one of the following, as shown in Figure 11:

- Service Request Source A
- Service Request Source B
- Service Request Source C
- Service Request Source A OR B

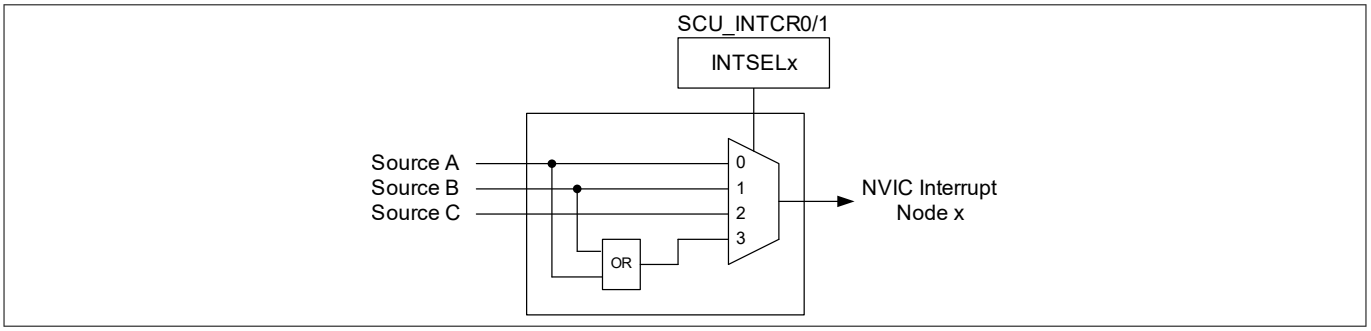


Figure 11 Source Selection per Interrupt Node

The selection is done through the bit field INTSELx (x=0-31) corresponding to the interrupt node in the SCU registers INTCR0 and INTCR1. Refer to the registers reference manual for the register description.

Table 19 Interrupt Node assignment

Node ID	Service Request Source A	Service Request Source B	Service Request Source C
0	SCU.SR0	CAN0.SR0	CCU40.SR0
1	SCU.SR1	CAN0.SR1	CCU80.SR0
2	SCU.SR2	CAN0.SR2	CCU80.SR1
3	ERU0.SR0	ERU1.SR0	CAN0.SR0
4	ERU0.SR1	ERU1.SR1	CAN0.SR1
5	ERU0.SR2	ERU1.SR2	CAN0.SR2
6	ERU0.SR3	ERU1.SR3	CAN0.SR3
7	MATH.SR0	CAN0.SR3	CCU40.SR1
8	Reserved	CCU40.SR0	CCU80.SR0
9	USIC0.SR0	USIC1.SR0	ERU0.SR0
10	USIC0.SR1	USIC1.SR1	ERU0.SR1
11	USIC0.SR2	USIC1.SR2	ERU0.SR2
12	USIC0.SR3	USIC1.SR3	ERU0.SR3
13	USIC0.SR4	USIC1.SR4	CCU80.SR1
14	USIC0.SR5	USIC1.SR5	POSIF0.SR0
15	VADC0.C0SR0	USIC0.SR0	POSIF0.SR1
16	VADC0.C0SR1	USIC0.SR1	CCU40.SR2
17	VADC0.G0SR0	USIC0.SR2	CAN0.SR0
18	VADC0.G0SR1	USIC0.SR3	CAN0.SR1
19	VADC0.G1SR0	USIC0.SR4	CAN0.SR2
20	VADC0.G1SR1	USIC0.SR5	CAN0.SR3
21	CCU40.SR0	CCU41.SR0	USIC0.SR0
22	CCU40.SR1	CCU41.SR1	USIC0.SR1
23	CCU40.SR2	CCU41.SR2	USIC0.SR2
24	CCU40.SR3	CCU41.SR3	USIC0.SR3
25	CCU80.SR0	CCU81.SR0	USIC0.SR4
26	CCU80.SR1	CCU81.SR1	USIC0.SR5
27	POSIF0.SR0	POSIF1.SR0	CCU40.SR3
28	POSIF0.SR1	POSIF1.SR1	ERU0.SR0
29	Reserved	CCU40.SR1	ERU0.SR1
30	Reserved	CCU40.SR2	ERU0.SR2
31	BCCU0.SR0	CCU40.SR3	ERU0.SR3

5.1.3 Interrupt Signal Generation

In PSC1Mx, all peripherals support only the generation of pulse interrupts. Pulse interrupts are also described as edge-triggered interrupts.

A pulse interrupt is an interrupt signal sampled synchronously on the rising edge of the processor clock (MCLK). To ensure the NVIC detects the interrupt, the peripheral asserts the interrupt signal for at least one MCLK 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 Hardware and software control of interrupts.

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.

Hardware and software control of interrupts

The Cortex®-M0 latches all interrupts. A peripheral interrupt becomes pending for one of the following reasons:

- the NVIC detects that the interrupt signal is active 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 Register NVIC_ISPR.

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:
  - 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, the state of the interrupt changes to inactive when the processor returns from the ISR.
- Software writes to the corresponding interrupt clear-pending register bit.
  - The state of the interrupt changes to inactive, if the state was pending; or active, if the state was active and pending.

5.1.4 NVIC design hints and tips

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

NVIC programming hints

Software uses the CPSIE i and CPSID i instructions to enable and disable interrupts. The CMSIS provides the following intrinsic functions for these instructions:

void __disable_irq(void) // Disable Interrupts

void __enable_irq(void) // Enable Interrupts

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

Table 20 CMSIS functions for NVIC control

CMSIS interrupt control function	Description
void NVIC_EnableIRQ (IRQn_t IRQn)	Enable IRQn
void NVIC_DisableIRQ (IRQn_t IRQn)	Disable IRQn
uint32_t NVIC_GetPendingIRQ (IRQn_t IRQn)	Return true (1) if IRQn is pending
void NVIC_SetPendingIRQ (IRQn_t IRQn)	Set IRQn pending

(table continues...)

Table 20 (continued) CMSIS functions for NVIC control

CMSIS interrupt control function	Description
void NVIC_ClearPendingIRQ (IRQn_t IRQn)	Clear IRQn pending status
void NVIC_SetPriority (IRQn_t IRQn, uint32_t priority)	Set priority for IRQn
uint32_t NVIC_GetPriority (IRQn_t IRQn)	Read priority of IRQn
void NVIC_SystemReset (void)	Reset the system

The input parameter IRQn is the IRQ number. For more information about these functions, please refer to the CMSIS documentation.

5.1.5 Accessing CPU Registers using CMSIS

CMSIS functions enable software portability between different Cortex®-M profile processors. To access the NVIC registers when using CMSIS, use the following functions:

Table 21 CMSIS access NVIC functions

CMSIS function	Description
void NVIC_EnableIRQ (IRQn_Type IRQn)[1)]	Enables an interrupt or exception.
void NVIC_DisableIRQ (IRQn_Type IRQn)[1)]	Disables an interrupt or exception.
void NVIC_SetPendingIRQ (IRQn_Type IRQn)[1)]	Sets the pending status of interrupt or exception to 1.
void NVIC_ClearPendingIRQ (IRQn_Type IRQn)[1)]	Clears the pending status of interrupt or exception to 0.
uint32_t NVIC_GetPendingIRQ (IRQn_Type IRQn)[1)]	Reads the pending status of interrupt or exception. This function returns non-zero value if the pending status is set to 1.
void NVIC_SetPriority(IRQn_Type IRQn, uint32_t priority)[1)]	Sets the priority of an interrupt or exception with configurable priority level to 1.
uint32_t NVIC_GetPriority(IRQn_Type IRQn)[1)]	Reads the priority of an interrupt or exception with configurable priority level. This function return the current priority level.

1) The input parameter IRQn is the IRQ number.

5.1.6 Interrupt Priority

An interrupt node can be assigned one of four priority levels. The levels are in steps of 64, from 0 to 192, and defined in an 8-bit priority field in the Interrupt Priority Register x (IPRx). A higher level corresponds to a lower priority, so level 0 is the highest interrupt priority.

Since there are four priority fields in each IPRx register and each field corresponds to one interrupt node, altogether 8 IPRx registers (IPR0...IPR7) are needed as shown in Figure 12.

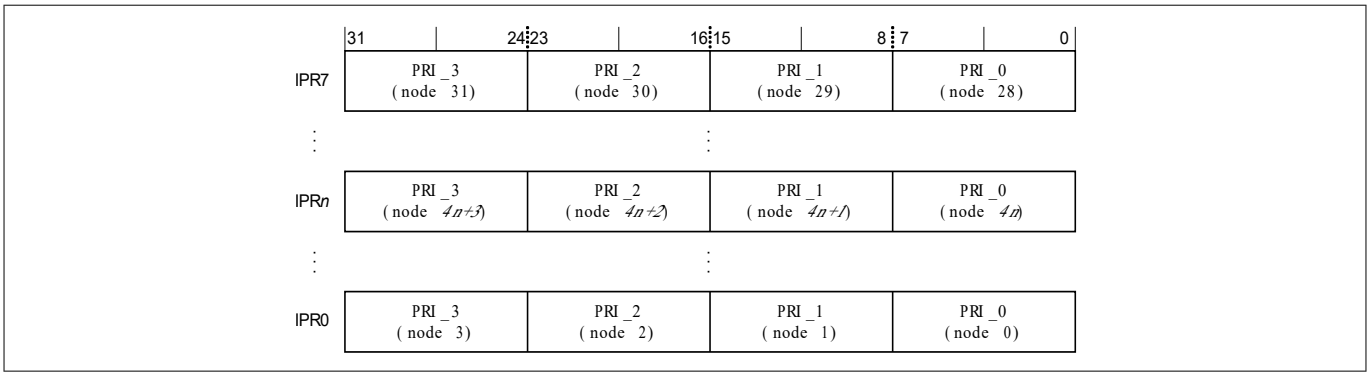


Figure 12 Interrupt Priority Register

The IPR number and byte offset for interrupt node m (0...31) can be found as follows:

- the corresponding IPR number n 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]
- for example, Priority field of interrupt node 21 is located at IPR5.[15:8], since
  - n = 21 DIV 4 = 5
  - byte offset = 21 MOD 4 = 1

***Note**: IPRx registers are only word-accessible.*

Refer to Table 20 for more information on the access to the interrupt node priority array, which provides the software view of the interrupt node priorities.

5.1.7 Interrupt Latency

The PSC1Mx interrupt latency, defined as the time from detection of the generated pulse and latching of the interrupt by NVIC to execution of the first instruction at the interrupt handler, is typically 21 MCLK cycles as shown in Figure 13.

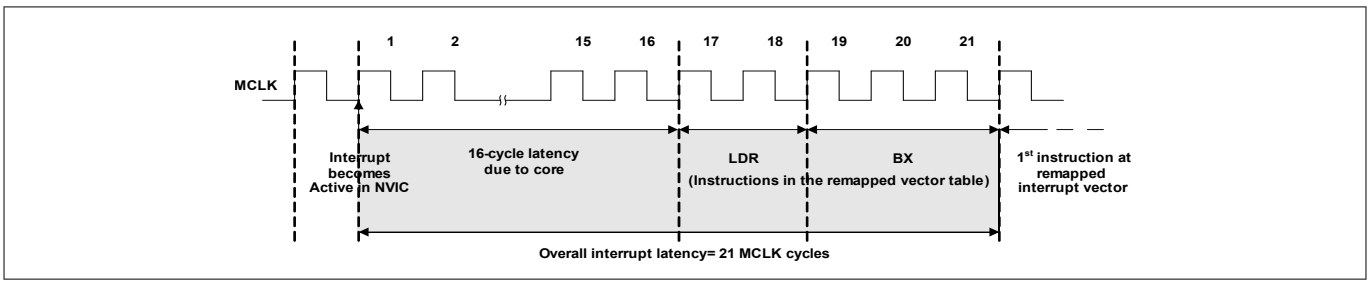


Figure 13 Typical PSC1Mx Interrupt Latency

This assumes the following conditions:

- Interrupt generation is enabled
- No occurrence of interrupt pre-emption, late-arrival or tail-chaining
- Delays due to memory wait states are not taken into account

5.2 General Module Interrupt Structure

A module might have multiple interrupt sources. Each interrupt source has typically the following structure (see Figure 14):

- An interrupt source status flag
- A set bit to allow software to set the flag to 1
- A clear bit to allow software to reset the flag to 0
- An enable bit to trigger interrupt when the hardware event occurs or status flag set bit is set (i.e. software triggered interrupt)

***Note**: If a flag set event (due to a peripheral HW event) occurs in the same clock cycle as a flag clear event (due to SW Setting of the Clear bit), the set has higher priority over the clear.*

***Note**: Setting of the status flag by a hardware event or software writing to status flag set bit, is independent of interrupt generation enabled/disabled. Similarly, interrupt generation is independent of the level of the status flag.*

Additionally, some modules might have more interrupt sources than interrupt lines. Therefore, they include an interrupt routing management block, which maps the interrupt sources to the interrupt lines.

For further details and exceptions to the above general structure, refer to the respective module chapters.

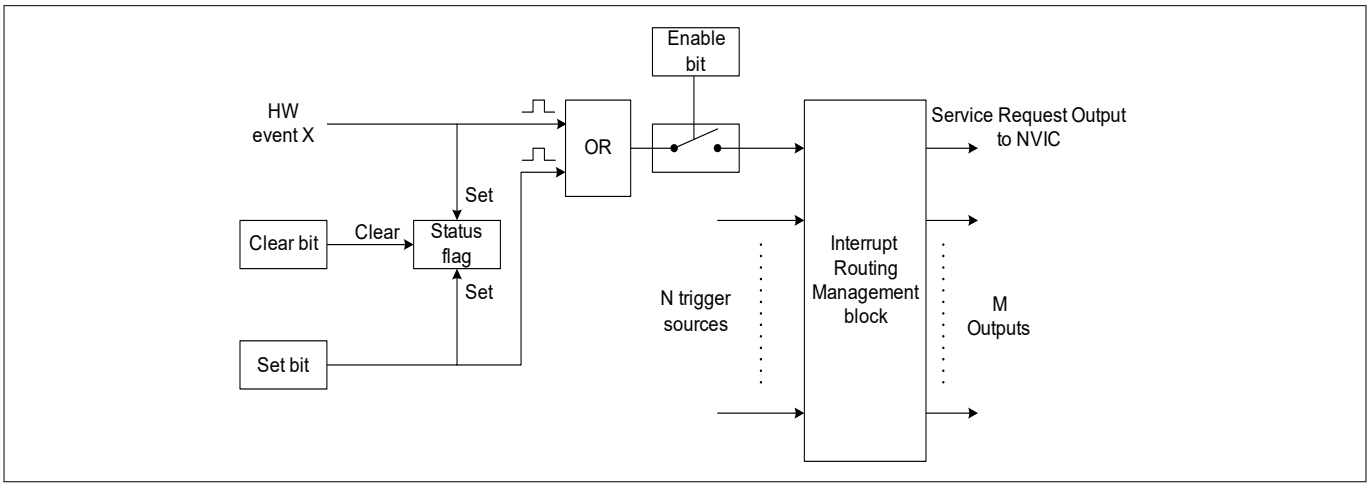


Figure 14 Typical Module Interrupt Structure

To enable a module HW event for interrupt generation, SW has to:

- Select the required module service request source for the interrupt node, through the SCU_INTCRx registers.
- Enable the interrupt node through the NVIC_ISER register.
- If the module has an interrupt routing management block, select an available service request output through which the interrupt will be generated to the NVIC. This is usually done by configuring a interrupt node pointer register in the module.
- Finally, set the interrupt enable bit of the module HW event for interrupt generation.

5.3 Registers

Table 22 Registers Address Space

Module	Base Address	End Address	Note
CPU PPB: System Control Space (SCS)	$E000E000_H$	$E000EFFF_H$	
SCU General Control Unit Registers	40010000_H	$40011FFF_H$	

Registers Overview

The absolute register address is calculated by adding:
Module Base Address + Offset Address

Table 23 Register Overview

Register Name	Description	Offset Address	Access Mode	
			Read	Write
Nested Vectored Interrupt Controller (NVIC)				
NVIC_ISER	Interrupt Set-enable Registers	100_H	U, PV	U, PV
NVIC_ICER	Interrupt Clear-enable Registers	180_H	U, PV	U, PV
NVIC_ISPR	Interrupt Set-pending Registers	200_H	U, PV	U, PV
NVIC_ICPR	Interrupt Clear-pending Registers	280_H	U, PV	U, PV
NVIC_IPRx (x=0-7)	Interrupt Priority Registers	400_H-$41C_H$	U, PV	U, PV
SCU Interrupt Related Registers				
SCU_INTCR0	Interrupt Control Register 0	$6C_H$	U, PV	U, PV
SCU_INTCR1	Interrupt Control Register 1	70_H	U, PV	U, PV

5.4 Interrupt Request Source Overview

An overview of all PSC1Mx interrupt sources and related register bits are shown in the next few pages. For interrupt node assignment, refer to Table 19.

Table 24 Interrupt Source Overview

Interrupt Source	Interrupt Event	Status Flag		Interrupt Enable		Set Flag		Clear Flag		Node Pointer	
		Register	Bit	Register	Bit	Register	Bit	Register	Bit	Register	Bit
SCU.SR0	Flash double bit ECC[1)]	SCU_SRRAW	FLECC2I	SCU_SRMSK	FLECC2I	SCU_SRSET	FLECC2I	SCU_SRCLR	FLECC2I	-	-
		NVM_NVMSTATUS	ECC2READ	-	-	-	-	NVM_NVMPROG	RSTECC	-	-
	Flash operation complete	SCU_SRRAW	FLCMPLTI	NVM_NVMCONF	INT_ON	SCU_SRSET	FLCMPLTI	SCU_SRCLR	FLCMPLTI	-	-
	SRAM parity error	SCU_SRRAW	PESRAMI	SCU_SRMSK	PESRAMI	SCU_SRSET	PESRAMI	SCU_SRCLR	PESRAMI	-	-
	USIC RAM parity error	SCU_SRRAW	PEU0I	SCU_SRMSK	PEU0I	SCU_SRSET	PEU0I	SCU_SRCLR	PEU0I	-	-
	Loss of clock	SCU_SRRAW	LOCI	SCU_SRMSK	LOCI	SCU_SRSET	LOCI	SCU_SRCLR	LOCI	-	-
SCU.SR1	Standby clock failure	SCU_SRRAW	SBYCLKFI	SCU_SRMSK	SBYCLKFI	SCU_SRSET	SBYCLKFI	SCU_SRCLR	SBYCLKFI	-	-
	VDDP pre-warning	SCU_SRRAW	VDDPI	SCU_SRMSK	VDDPI	SCU_SRSET	VDDPI	SCU_SRCLR	VDDPI	-	-
	VDDC drops below VDROP	SCU_SRRAW	VDROPI	SCU_SRMSK	VDROPI	SCU_SRSET	VDROPI	SCU_SRCLR	VDROPI	-	-
	VDDC rises above VCLIP	SCU_SRRAW	VCLIPI	SCU_SRMSK	VCLIPI	SCU_SRSET	VCLIPI	SCU_SRCLR	VCLIPI	-	-
	DTS done	SCU_SRRAW	TSE_DONE	SCU_SRMSK	TSE_DONE	SCU_SRSET	TSE_DONE	SCU_SRCLR	TSE_DONE	-	-

(table continues...)

Table 24 (continued) Interrupt Source Overview

Interrupt Source	Interrupt Event	Status Flag		Interrupt Enable		Set Flag		Clear Flag		Node Pointer	
		Register	Bit	Register	Bit	Register	Bit	Register	Bit	Register	Bit
	DTS compare high	SCU_SRRAW	TSE_HIGH	SCU_SRMSK	TSE_HIGH	SCU_SRSET	TSE_HIGH	SCU_SRCLR	TSE_HIGH	-	-
	DTS compare low	SCU_SRRAW	TSE_LOW	SCU_SRMSK	TSE_LOW	SCU_SRSET	TSE_LOW	SCU_SRCLR	TSE_LOW	-	-
	WDT pre-warning	SCU_SRRAW	PRWARN	SCU_SRMSK	PRWARN	SCU_SRSET	PRWARN	SCU_SRCLR	PRWARN	-	-
	RTC periodic event	SCU_SRRAW	PI	-	-	SCU_SRSET	PI	SCU_SRCLR	PI	-	-
	RTC alarm	SCU_SRRAW	AI	-	-	SCU_SRSET	AI	SCU_SRCLR	AI	-	-
	RTC CTR Mirror Register updated	SCU_SRRAW	RTC_CTR	SCU_SRMSK	RTC_CTR	SCU_SRSET	RTC_CTR	SCU_SRCLR	RTC_CTR	-	-
	RTC ATIM0 Mirror Register updated	SCU_SRRAW	RTC_ATIM0	SCU_SRMSK	RTC_ATIM0	SCU_SRSET	RTC_ATIM0	SCU_SRCLR	RTC_ATIM0	-	-
	RTC ATIM1 Mirror Register updated	SCU_SRRAW	RTC_ATIM1	SCU_SRMSK	RTC_ATIM1	SCU_SRSET	RTC_ATIM1	SCU_SRCLR	RTC_ATIM1	-	-
	RTC TIM0 Mirror Register updated	SCU_SRRAW	RTC_TIM0	SCU_SRMSK	RTC_TIM0	SCU_SRSET	RTC_TIM0	SCU_SRCLR	RTC_TIM0	-	-

(table continues...)

Table 24 (continued) Interrupt Source Overview

Interrupt Source	Interrupt Event	Status Flag		Interrupt Enable		Set Flag		Clear Flag		Node Pointer	
		Register	Bit	Register	Bit	Register	Bit	Register	Bit	Register	Bit
	RTC TIM1 Mirror Register updated	SCU_SRRAW	RTC_TIM1	SCU_SRMSK	RTC_TIM1	SCU_SRSET	RTC_TIM1	SCU_SRCLR	RTC_TIM1	-	-
SCU.SR2	Out of range comparator x event (x=0-7)	SCU_SRRAW	ORCxI	SCU_SRMSK	ORCxI	SCU_SRSET	ORCxI	SCU_SRCLR	ORCxI	-	-
	Analog comparator x event (x=0-2)	SCU_SRRAW	ACMPxI	SCU_SRMSK	ACMPxI	SCU_SRSET	ACMPxI	SCU_SRCLR	ACMPxI	-	-
ERUx.SR[3:0] (x=0-1)	ERUx_IOUTy (y=0-3)	See chapter on ERUx for details.									
MATH.SR0	CORDIC end of calculation	MATH_EVFR	CDEOC	MATH_EVIER	CDEOCIEN	MATH_EVFSR	CDEOCS	MATH_EVFCR	CDEOCC	-	-
	CORDIC error	MATH_EVFR	CDERR	MATH_EVIER	CDERRIEN	MATH_EVFSR	CDERRS	MATH_EVFCR	CDERRS	-	-
	DIV end of calculation	MATH_EVFR	DIVEOC	MATH_EVIER	DIVEOCIEN	MATH_EVFSR	DIVEOCS	MATH_EVFCR	DIVEOCC	-	-
	DIV error	MATH_EVFR	DIVERR	MATH_EVIER	DIVERRIEN	MATH_EVFSR	DIVERRS	MATH_EVFCR	DIVERRC	-	-
USICx_SR[5:0] (x=0-1)	USIC: Standard receive event	USICx_CHy_PSR	RIF	USICx_CHy_CCR	RIEN	-	-	USICx_CHy_PSCR	CRIF	USICx_CHy_INPR	RINP

(table continues...)

Table 24 (continued) Interrupt Source Overview

Interrupt Source	Interrupt Event	Status Flag		Interrupt Enable		Set Flag		Clear Flag		Node Pointer	
		Register	Bit	Register	Bit	Register	Bit	Register	Bit	Register	Bit
	USIC: Receive start event	USICx_CHy_PSR	RSIF	USICx_CHy_CCR	RSIEN	-	-	USICx_CHy_PSCR	CRSIF	USICx_CHy_INPR	TBINP
	USIC: Alternate receive event	USICx_CHy_PSR	AIF	USICx_CHy_CCR	AIEN	-	-	USICx_CHy_PSCR	CAIF	USICx_CHy_INPR	AINP
	USIC: Transmit shift event	USICx_CHy_PSR	TSIF	USICx_CHy_CCR	TSIEN	-	-	USICx_CHy_PSCR	CTSIF	USICx_CHy_INPR	TSINP
	USIC: Transmit buffer event	USICx_CHy_PSR	TBIF	USICx_CHy_CCR	TBIEN	-	-	USICx_CHy_PSCR	CTBIF	USICx_CHy_INPR	TBINP
	USIC: Data lost event	USICx_CHy_PSR	DLIF	USICx_CHy_CCR	DLIEN	-	-	USICx_CHy_PSCR	CDLIF	USICx_CHy_INPR	PINP
	USIC: BRG event	USICx_CHy_PSR	BRGIF	USICx_CHy_CCR	BRGIEN	-	-	USICx_CHy_PSCR	CBRGIF	USICx_CHy_INPR	PINP
USICx_SR[5:0] (x=0-1)	USIC: Standard transmit buffer event	USICx_CHy_TRBSR	STBI	USICx_CHy_TBCTR	STBIEN	-	-	USICx_CHy_TRBSCR	CSTBI	USICx_CHy_TBCTR	STBINP
	USIC: Standard transmit buffer event	USICx_CHy_TRBSR	STBT	USICx_CHy_TBCTR	STBIEN	-	-	-	-	USICx_CHy_TBCTR	STBINP
	USIC: Transmit Buffer error event	USICx_CHy_TRBSR	TBERI	USICx_CHy_TBCTR	TBERIEN	-	-	USICx_CHy_TRBSCR	CTBERI	USICx_CHy_TBCTR	ATBINP

(table continues...)

Table 24 (continued) Interrupt Source Overview

Interrupt Source	Interrupt Event	Status Flag		Interrupt Enable		Set Flag		Clear Flag		Node Pointer	
		Register	Bit	Register	Bit	Register	Bit	Register	Bit	Register	Bit
	USIC: Standard receive buffer event	USICx_CHy_TRBSR	SRBI	USICx_CHy_RBCTR	SRBIEN	-	-	USICx_CHy_TRBSCR	CSRBI	USICx_CHy_RBCTR	SRBINP
	USIC: Standard receive buffer event	USICx_CHy_TRBSR	SRBT	USICx_CHy_RBCTR	SRBIEN	-	-	-	-	USICx_CHy_RBCTR	SRBINP
	USIC: Alternate receive buffer event	USICx_CHy_TRBSR	ARBI	USICx_CHy_RBCTR	ARBIEN	-	-	USICx_CHy_TRBSCR	CARBI	USICx_CHy_RBCTR	ARBINP
	USIC: Receive buffer error event	USICx_CHy_TRBSR	RBERI	USICx_CHy_RBCTR	RBERIEN	-	-	USICx_CHy_TRBSCR	CRBERI	USICx_CHy_RBCTR	ARBINP
	ASC: Synchronisation break detected	USICx_CHy_PSR	SBD	USICx_CHy_PCR	SBDIEN	-	-	USICx_CHy_PSCR	CSBD	USICx_CHy_INPR	PINP
	ASC: Collision detected	USICx_CHy_PSR	COL	USICx_CHy_PCR	CDIEN	-	-	USICx_CHy_PSCR	CCOL	USICx_CHy_INPR	PINP
	ASC: Receiver noise detected	USICx_CHy_PSR	RNS	USICx_CHy_PCR	RNIEN	-	-	USICx_CHy_PSCR	CRNS	USICx_CHy_INPR	PINP
	ASC: Format error in stop bit 0	USICx_CHy_PSR	FER0	USICx_CHy_PCR	FEIEN	-	-	USICx_CHy_PSCR	CFER0	USICx_CHy_INPR	PINP

(table continues...)

Table 24 (continued) Interrupt Source Overview

Interrupt Source	Interrupt Event	Status Flag		Interrupt Enable		Set Flag		Clear Flag		Node Pointer	
		Register	Bit	Register	Bit	Register	Bit	Register	Bit	Register	Bit
	ASC: Format error in stop bit 1	USICx_CHy_PSR	FER1	USICx_CHy_PCR	FEIEN	-	-	USICx_CHy_PSCR	CFER1	USICx_CHy_INPR	PINP
	ASC: Receive frame finished	USICx_CHy_PSR	RFF	USICx_CHy_PCR	FFIEN	-	-	USICx_CHy_PSCR	CRFF	USICx_CHy_INPR	PINP
	ASC: Transmit frame finished	USICx_CHy_PSR	TFF	USICx_CHy_PCR	FFIEN	-	-	USICx_CHy_PSCR	CTFF	USICx_CHy_INPR	PINP
USIC0_SR[5:0] (x=0-1)	SSC: MSLS event detected	USICx_CHy_PSR	MSLSEV	USICx_CHy_PCR	MSLSIEN	-	-	USICx_CHy_PSCR	CMSLSEV	USICx_CHy_INPR	PINP
	SSC: Parity error detected	USICx_CHy_PSR	PAERR	USICx_CHy_PCR	PARIEN	-	-	USICx_CHy_PSCR	CPAERR	USICx_CHy_INPR	PINP
	SSC: DX2T event detected	USICx_CHy_PSR	DX2TEV	USICx_CHy_PCR	DX2TIEN	-	-	USICx_CHy_PSCR	CDX2TEV	USICx_CHy_INPR	PINP
	IIC: Wrong TDF code detected	USICx_CHy_PSR	WTDF	USICx_CHy_PCR	ERRIEN	-	-	USICx_CHy_PSCR	CWTDF	USICx_CHy_INPR	PINP
	IIC: Start condition received	USICx_CHy_PSR	SCR	USICx_CHy_PCR	SCRIEN	-	-	USICx_CHy_PSCR	CSCR	USICx_CHy_INPR	PINP

(table continues...)

Table 24 (continued) Interrupt Source Overview

Interrupt Source	Interrupt Event	Status Flag		Interrupt Enable		Set Flag		Clear Flag		Node Pointer	
		Register	Bit	Register	Bit	Register	Bit	Register	Bit	Register	Bit
	IIC: Repeated start condition received	USICx_CHy_PSR	RSCR	USICx_CHy_PCR	RSCRIEN	-	-	USICx_CHy_PSCR	CRSCR	USICx_CHy_INPR	PINP
	IIC: Stop condition received	USICx_CHy_PSR	PCR	USICx_CHy_PCR	PCRIEN	-	-	USICx_CHy_PSCR	CPCR	USICx_CHy_INPR	PINP
	IIC: NACK received	USICx_CHy_PSR	NACK	USICx_CHy_PCR	NACKIEN	-	-	USICx_CHy_PSCR	CNACK	USICx_CHy_INPR	PINP
	IIC: Arbitration lost	USICx_CHy_PSR	ARL	USICx_CHy_PCR	ARLIEN	-	-	USICx_CHy_PSCR	CARL	USICx_CHy_INPR	PINP
	IIC: Slave read request	USICx_CHy_PSR	SRR	USICx_CHy_PCR	SRRIEN	-	-	USICx_CHy_PSCR	CSRR	USICx_CHy_INPR	PINP
	IIC: Error detected	USICx_CHy_PSR	ERR	USICx_CHy_PCR	ERRIEN	-	-	USICx_CHy_PSCR	CERR	USICx_CHy_INPR	PINP
	IIC: ACK received	USICx_CHy_PSR	ACK	USICx_CHy_PCR	ACKIEN	-	-	USICx_CHy_PSCR	CACK	USICx_CHy_INPR	PINP
	IIS: DX2T event detected	USICx_CHy_PSR	DX2TEV	USICx_CHy_PCR	DX2TIEN	-	-	USICx_CHy_PSCR	CDX2TEV	USICx_CHy_INPR	PINP
	IIS: WA falling edge event	USICx_CHy_PSR	WAFE	USICx_CHy_PCR	WAFEIEN	-	-	USICx_CHy_PSCR	CWAFE	USICx_CHy_INPR	PINP
	IIS: WA rising edge event	USICx_CHy_PSR	WARE	USICx_CHy_PCR	WAREIEN	-	-	USICx_CHy_PSCR	CWARE	USICx_CHy_INPR	PINP

(table continues...)

Table 24 (continued) Interrupt Source Overview

Interrupt Source	Interrupt Event	Status Flag		Interrupt Enable		Set Flag		Clear Flag		Node Pointer	
		Register	Bit	Register	Bit	Register	Bit	Register	Bit	Register	Bit
	IIS: WA generation end	USICx_CHy_PSR	END	USICx_CHy_PCR	ENDIEN	-	-	USICx_CHy_PSCR	CEND	USICx_CHy_INPR	PINP
VADC0_C0SR[1:0] VADC0_GxSR[1:0] (x=0-1)	Source Event 0	VADC0_GxSEFLAG	SEV0	VADC0_GxQINR0	ENSI	VADC0_GxSEFLAG	SEV0	VADC0_GxSEFCLR	SEV0	VADC0_GxSEVNP	SEV0NP
	Source Event 1	VADC0_GxSEFLAG	SEV1	VADC0_GxASMR	ENSI	VADC0_GxSEFLAG	SEV1	VADC0_GxSEFCLR	SEV1	VADC0_GxSEVNP	SEV1NP
	Channel Event y (y=0-7)	VADC0_GxCEFLAG	CEVy	VADC0_GxCHCTRy	CHEVMODE	VADC0_GxCEFLAG	CEVy	VADC0_GxCEFCLR	CEVy	VADC0_GxCEVNP	CEVyINP
	Result Event y (y=0-7)	VADC0_GxREFLAG	REVy	VADC0_GxRCRy	SRGEN	VADC0_GxREFLAG	REVy	VADC0_GxREFCLR	REVy	VADC0_GxREVNP0	REVyNP
	Result Event y (y=8-15)	VADC0_GxREFLAG	REVy	VADC0_GxRCRy	SRGEN	VADC0_GxREFLAG	REVy	VADC0_GxREFCLR	REVy	VADC0_GxREVNP1	REVyNP
	Global Source Event	VADC0_GLOBEFLAG	SEVGLB	VADC0_BRSMR	ENSI	VADC0_GLOBEFLAG	SEVGLB	VADC0_GLOBEFLAG	SEVGLBCLR	VADC0_GLOBEVNP	REV0NP
	Global Result Event	VADC0_GLOBEFLAG	REVGLB	VADC0_GLOBERCR	SRGEN	VADC0_GLOBEFLAG	REVGLB	VADC0_GLOBEFLAG	REVGLBCLR	VADC0_GLOBEVNP	SEV0NP
CCU4x_SR[3:0] (x=0-1)	Event 0 edge(s) information from event selector	CCU4x_CC4yINTS	E0AS	CCU4x_CC4yINTE	E0AE	CCU4x_CC4ySWS	SE0A	CCU4x_CC4ySWR	RE0A	CCU4x_CC4ySRS	E0SR

(table continues...)

Table 24 (continued) Interrupt Source Overview

Interrupt Source	Interrupt Event	Status Flag		Interrupt Enable		Set Flag		Clear Flag		Node Pointer	
		Register	Bit	Register	Bit	Register	Bit	Register	Bit	Register	Bit
	Event 1 edge(s) information from event selector	CCU4x_ CC4yINTS	E1AS	CCU4x_ CC4yINTE	E1AE	CCU4x_ CC4ySWS	SE1A	CCU4x_ CC4ySWR	RE1A	CCU4x_ CC4ySRS	E1SR
	Event 2 edge(s) information from event selector	CCU4x_ CC4yINTS	E2AS	CCU4x_ CC4yINTE	E2AE	CCU4x_ CC4ySWS	SE2A	CCU4x_ CC4ySWR	RE2A	CCU4x_ CC4ySRS	E2SR
	Period Match while counting up	CCU4x_ CC4yINTS	PMUS	CCU4x_ CC4yINTE	PME	CCU4x_ CC4ySWS	SPM	CCU4x_ CC4ySWR	RPM	CCU4x_ CC4ySRS	POSR
	Compare Match while counting up	CCU4x_ CC4yINTS	CMUS	CCU4x_ CC4yINTE	CMUE	CCU4x_ CC4ySWS	SCMU	CCU4x_ CC4ySWR	RCMU	CCU4x_ CC4ySRS	CMSR
	Compare Match while counting down	CCU4x_ CC4yINTS	CMDS	CCU4x_ CC4yINTE	CMDE	CCU4x_ CC4ySWS	SCMD	CCU4x_ CC4ySWR	RCMD	CCU4x_ CC4ySRS	CMSR
	One Match while counting down	CCU4x_ CC4yINTS	OMDS	CCU4x_ CC4yINTE	OME	CCU4x_ CC4ySWS	SOM	CCU4x_ CC4ySWR	ROM	CCU4x_ CC4ySRS	POSR
	Entering Trap State	CCU4x_ CC4yINTS	TRPF	CCU4x_ CC4yINTE	E2AE	CCU4x_ CC4ySWS	STRPF	CCU4x_ CC4ySWR	RTRPF	CCU4x_ CC4ySRS	E2SR

(table continues...)

Table 24 (continued) Interrupt Source Overview

Interrupt Source	Interrupt Event	Status Flag		Interrupt Enable		Set Flag		Clear Flag		Node Pointer	
		Register	Bit	Register	Bit	Register	Bit	Register	Bit	Register	Bit
CCU8x_ SR[1:0] (x=0-1)	Event 0 edge(s) information from event selector	CCU8x_ CC8yINTS	E0AS	CCU8x_ CC8yINTE	E0AE	CCU8x_ CC8ySWS	SE0A	CCU8x_ CC8ySWR	RE0A	CCU8x_ CC8ySRS	E0SR
	Event 1 edge(s) information from event selector	CCU8x_ CC8yINTS	E1AS	CCU8x_ CC8yINTE	E1AE	CCU8x_ CC8ySWS	SE1A	CCU8x_ CC8ySWR	RE1A	CCU8x_ CC8ySRS	E1SR
	Event 2 edge(s) information from event selector	CCU8x_ CC8yINTS	E2AS	CCU8x_ CC8yINTE	E2AE	CCU8x_ CC8ySWS	SE2A	CCU8x_ CC8ySWR	RE2A	CCU8x_ CC8ySRS	E2SR
	Period Match while counting up	CCU8x_ CC8yINTS	PMUS	CCU8x_ CC8yINTE	PME	CCU8x_ CC8ySWS	SPM	CCU8x_ CC8ySWR	RPM	CCU8x_ CC8ySRS	POSR
	Compare Match while counting up from compare channel 1	CCU8x_ CC8yINTS	CMU1S	CCU8x_ CC8yINTE	CMU1E	CCU8x_ CC8ySWS	SCM1U	CCU8x_ CC8ySWR	RCM1U	CCU8x_ CC8ySRS	CM1SR

(table continues...)

Table 24 (continued) Interrupt Source Overview

Interrupt Source	Interrupt Event	Status Flag		Interrupt Enable		Set Flag		Clear Flag		Node Pointer	
		Register	Bit	Register	Bit	Register	Bit	Register	Bit	Register	Bit
	Compare Match while counting down from compare channel 1	CCU8x_ CC8yINTS	CMD1S	CCU8x_ CC8yINTE	CMD1E	CCU8x_ CC8ySWS	SCM1D	CCU8x_ CC8ySWR	RCM1D	CCU8x_ CC8ySRS	CM1SR
	Compare Match while counting up from compare channel 2	CCU8x_ CC8yINTS	CMU2S	CCU8x_ CC8yINTE	CMU2E	CCU8x_ CC8ySWS	SCM2U	CCU8x_ CC8ySWR	RCM2U	CCU8x_ CC8ySRS	CM2SR
	Compare Match while counting down from compare channel 2	CCU8x_ CC8yINTS	CMD2S	CCU8x_ CC8yINTE	CMD2E	CCU8x_ CC8ySWS	SCM2D	CCU8x_ CC8ySWR	RCM2D	CCU8x_ CC8ySRS	CM2SR
	One Match while counting down	CCU8x_ CC8yINTS	OMDS	CCU8x_ CC8yINTE	OME	CCU8x_ CC8ySWS	SOM	CCU8x_ CC8ySWR	ROM	CCU8x_ CC8ySRS	POSR
	Entering Trap State	CCU8x_ CC8yINTS	TRPF	CCU8x_ CC8yINTE	E2AE	CCU8x_ CC8ySWS	STRPF	CCU8x_ CC8ySWR	RTRPF	CCU8x_ CC8ySRS	E2SR
POSIFx. SR[1:0] (x=0-1)	Transition at Hall inputs	POSIFx_ PFLG	HIES	POSIFx_ PFLGE	EHIE	POSIFx_ SPFLG	SHIE	POSIFx_ RPFLG	RHIE	POSIFx_ PFLGE	HIESEL

(table continues...)

Table 24 (continued) Interrupt Source Overview

Interrupt Source	Interrupt Event	Status Flag		Interrupt Enable		Set Flag		Clear Flag		Node Pointer	
		Register	Bit	Register	Bit	Register	Bit	Register	Bit	Register	Bit
	Occurrence of correct Hall event	POSIFx_PFLG	CHES	POSIFx_PFLGE	ECHE	POSIFx_SPFLG	SCHE	POSIFx_RPFLG	RCHE	POSIFx_PFLGE	CHESEL
	Occurrence of wrong Hall event	POSIFx_PFLG	WHES	POSIFx_PFLGE	EWHE	POSIFx_SPFLG	SWHE	POSIFx_RPFLG	RWHE	POSIFx_PFLGE	WHESEL
	Shadow transfer of MCM pattern	POSIFx_PFLG	MSTS	POSIFx_PFLGE	EMST	POSIFx_SPFLG	SMST	POSIFx_RPFLG	RMST	POSIFx_PFLGE	MSTSEL
	Index event detection	POSIFx_PFLG	INDXS	POSIFx_PFLGE	EINDX	POSIFx_SPFLG	SINDX	POSIFx_RPFLG	RINDX	POSIFx_PFLGE	INDSEL
	Phase detection error	POSIFx_PFLG	ERRS	POSIFx_PFLGE	EERR	POSIFx_SPFLG	SERR	POSIFx_RPFLG	RERR	POSIFx_PFLGE	ERRSEL
	Quadrature clock generation	POSIFx_PFLG	CNTS	POSIFx_PFLGE	ECNT	POSIFx_SPFLG	SCNT	POSIFx_RPFLG	RCNT	POSIFx_PFLGE	CNTSEL
	Period clock generation	POSIFx_PFLG	PCLKS	POSIFx_PFLGE	EPCLK	POSIFx_SPFLG	SPCLK	POSIFx_RPFLG	RPCLK	POSIFx_PFLGE	PCLSEL
	Direction change	POSIFx_PFLG	DIRS	POSIFx_PFLGE	EDIR	POSIFx_SPFLG	SDIR	POSIFx_RPFLG	RDIR	POSIFx_PFLGE	DIRSEL
BCCU0.SR0	Trigger 0	BCCU0_EVFR	T0F	BCCU0_EVIER	T0IEN	BCCU0_EVFSR	T0FS	BCCU0_EVFCR	T0FC	-	-
	Trigger 1	BCCU0_EVFR	T1F	BCCU0_EVIER	T1IEN	BCCU0_EVFSR	T1FS	BCCU0_EVFCR	T1FC	-	-

(table continues...)

Table 24 (continued) Interrupt Source Overview

Interrupt Source	Interrupt Event	Status Flag		Interrupt Enable		Set Flag		Clear Flag		Node Pointer	
		Register	Bit	Register	Bit	Register	Bit	Register	Bit	Register	Bit
	FIFO Full	BCCU0_EVFR	FF	BCCU0_EVIER	FIEN	BCCU0_EVFSR	FFS	BCCU0_EVFCR	FFC	-	-
	FIFO Empty	BCCU0_EVFR	EF	BCCU0_EVIER	EIEN	BCCU0_EVFSR	EFS	BCCU0_EVFCR	EFC	-	-
	Trap	BCCU0_EVFR	TPF	BCCU0_EVIER	TPIEN	BCCU0_EVFSR	TPFS	BCCU0_EVFCR	TPFC	-	-
CAN0.SR[3:0]	Message Transmitted	CAN0_NSRx	TXOK	CAN0_NCRx	TRIE	-	-	-	-	CAN0_NIPRx	TRINP
	Message Received	CAN0_NSRx	RXOK	CAN0_NCRx	TRIE	-	-	-	-	CAN0_NIPRx	TRINP
	Last Error Code	CAN0_NSRx	LEC	CAN0_NCRx	LECIE	-	-	-	-	CAN0_NIPRx	LECINP
	Fast Last Error Code	CAN0_NSRx	FLEC	CAN0_NCRx	LECIE	-	-	-	-	CAN0_NIPRx	LECINP
	Alert Warning	CAN0_NSRx	ALERT	CAN0_NCRx	ALIE	-	-	-	-	CAN0_NIPRx	ALINP
	Receive Pending	CAN0_MOSTATn	RXPND	CAN0_MOFCRn	RXIE	-	-	CAN0_MOCTRn	RESRXPND	CAN0_MOIPRn	RXINP
	Transmit Pending	CAN0_MOSTATn	TXPND	CAN0_MOFCRn	TXIE	-	-	CAN0_MOCTRn	RESTXPND	CAN0_MOIPRn	TXINP
	FIFO Full	-	-	CAN0_MOFCRn	OVIE	-	-	-	-	CAN0_MOIPRn	TXINP/RXINP

1) Flash ECC double bit error has two status flags, each having its own clear bit. It is sufficient to use only one of the status flag and ignore the other.

6 Event Request Unit (ERU)

As described in the Service Request Processing chapter, PSC1Mx uses the Event Request Unit (ERU) to support the programmable interconnection for the processing of service requests. There are 2 instances of ERU in PSC1Mx, ERU0 and ERU1.

6.1 Features

The ERU supports these features:

- 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

6.2 Overview

The Event Request Unit (ERU) is a versatile multiple input event detection and processing unit.

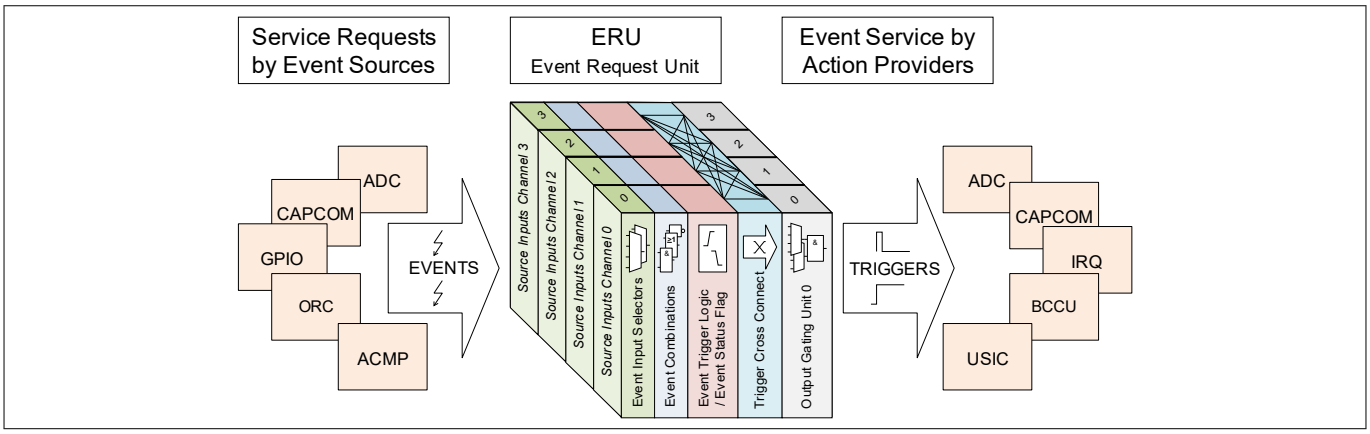


Figure 15 Event Request Unit Overview

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.*

6.3 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.

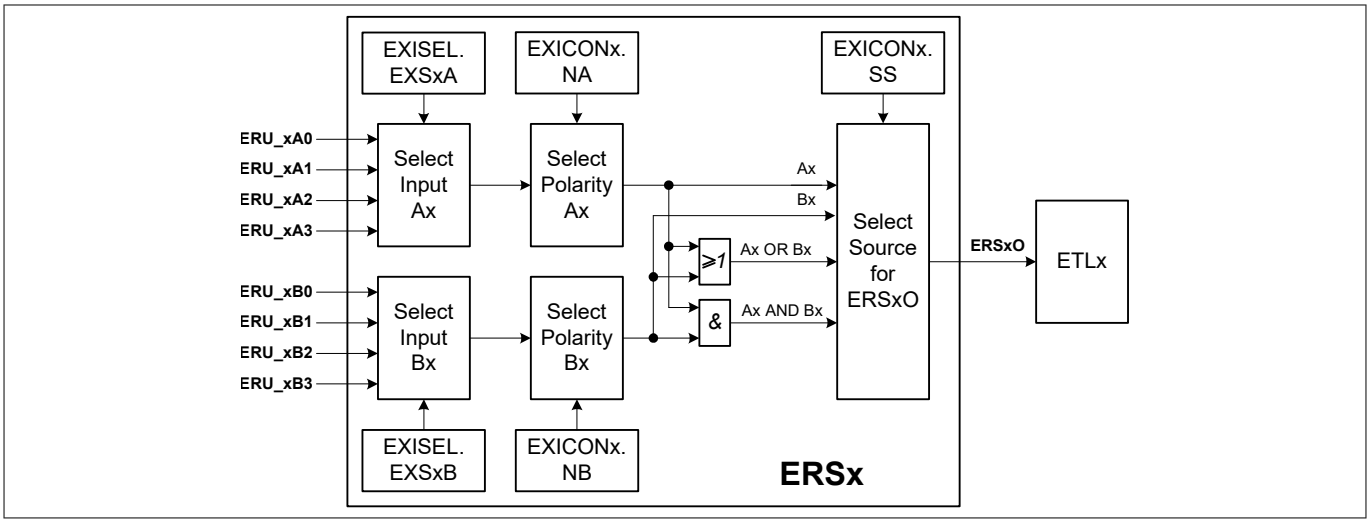


Figure 16 Event Request Select Unit Overview

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, for example, ERU0_EXICONx (x=0-3) for Input Channel 0).

6.4 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 (for example, 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, for example, ERU0_EXICONx (x=0-3) to control ERS0 and ETL0).

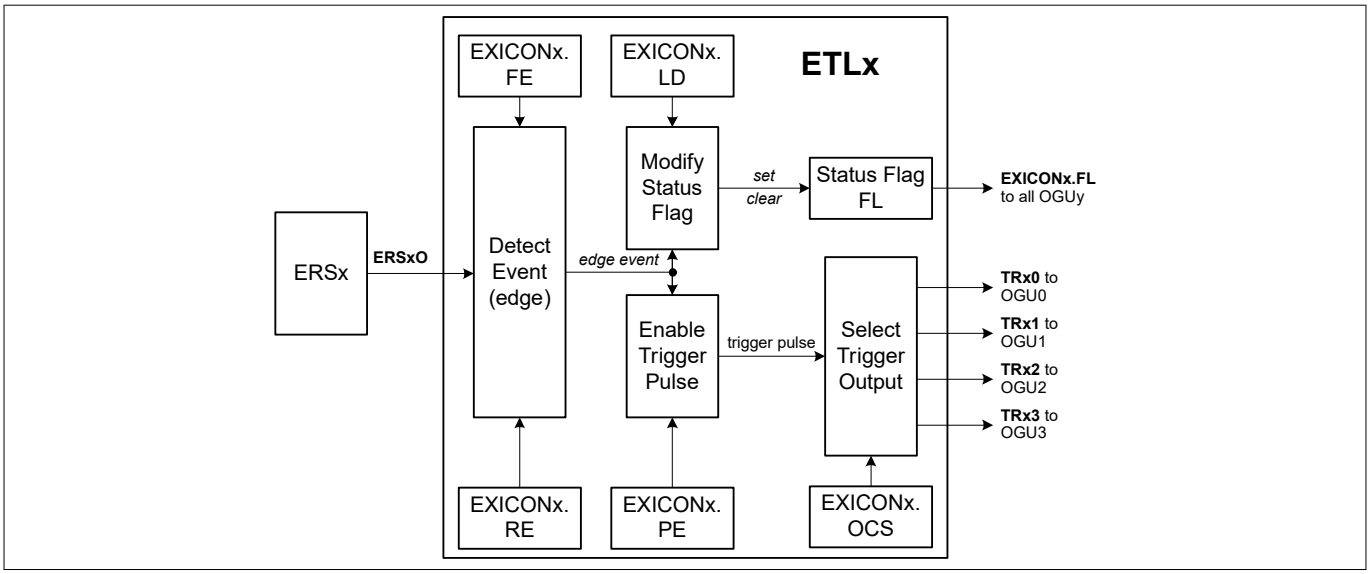


Figure 17 Event Trigger Logic Overview

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 18) 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.

6.5 Cross Connect Matrix

The matrix shown in Figure 18 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.

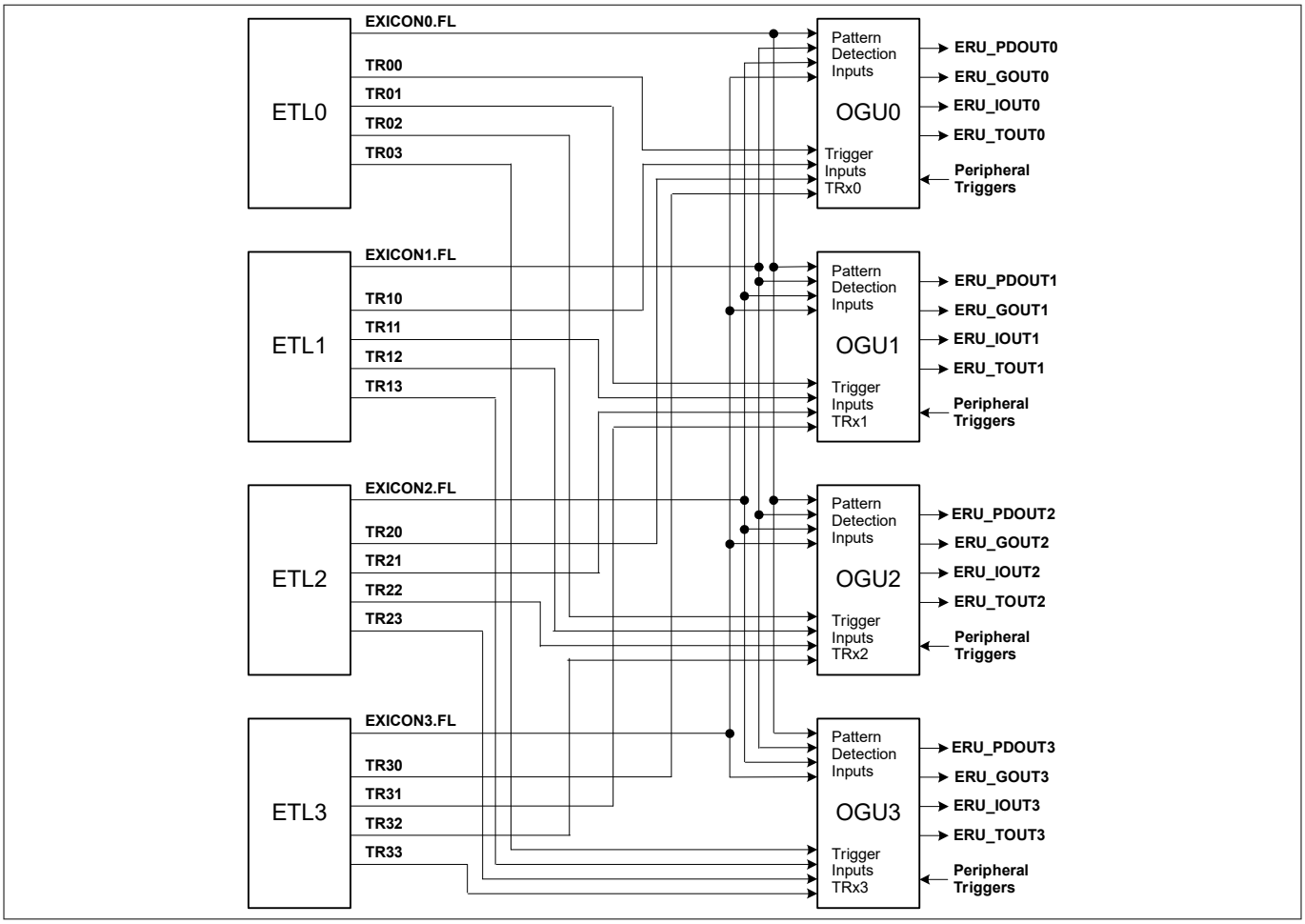


Figure 18 ERU Cross Connect Matrix

6.6 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 19 illustrates the logic blocks within an OGUy unit. All functions of an OGUy unit are controlled by its associated EXOCONy register, for example, 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.

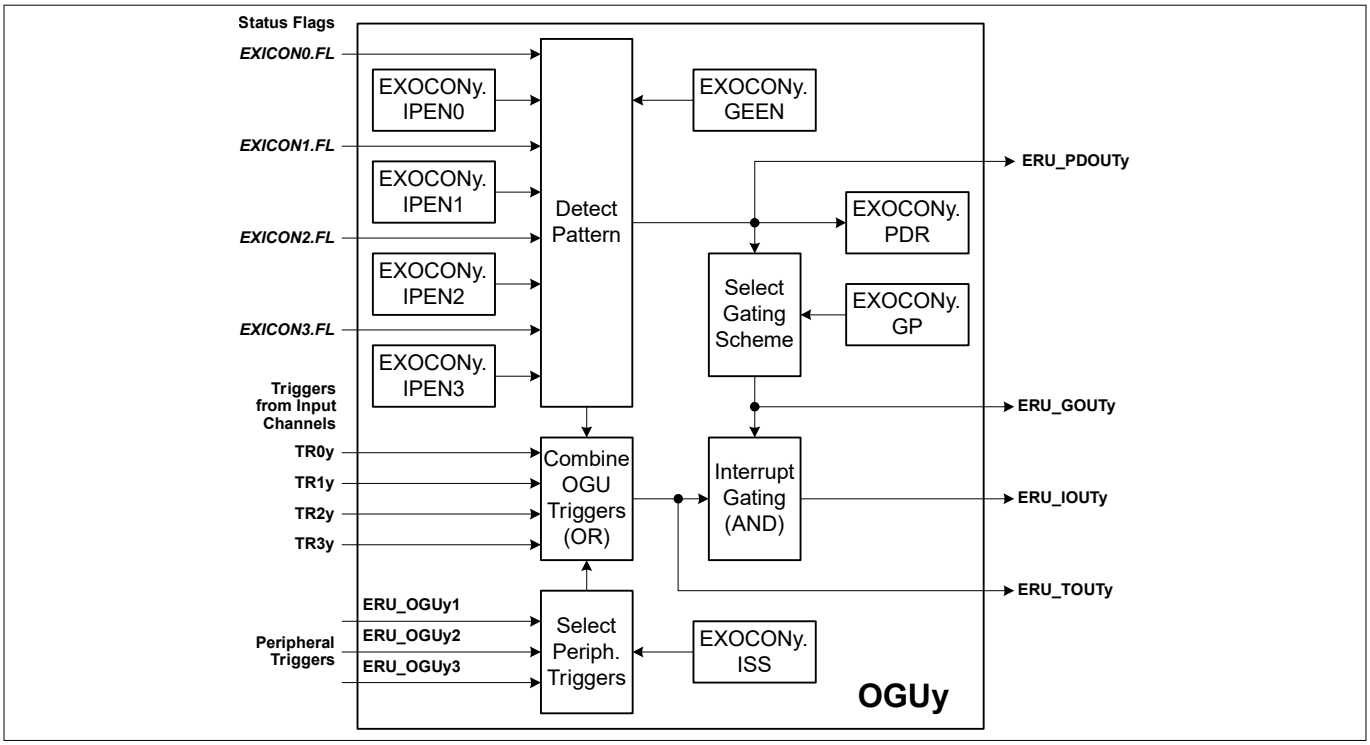


Figure 19 Output Gating Unit for Output Channel y

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 (for example, 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, for example, 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).

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 (for example, 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).

6.7 Power, Reset and Clock

ERU is running on the main clock, MCLK. It is consuming power in all operating modes as long as the MCLK continues to run.

6.8 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 ERUx 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 peripheral 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.*

6.9 Registers

Table 25 Registers Address Space

Module	Base Address	End Address	Note
ERU0	4001 0600$_H$	4001 062F$_H$	
ERU1	4001 0630$_H$	4001 06FF$_H$	

Registers Overview

The absolute register address is calculated by adding:
Module Base Address + Offset Address

Table 26 Register Overview

Register Name	Description	Offset Address	Access Mode	
			Read	Write
EXISEL	ERU External Input Control Selection	0000$_H$	U, PV	U, PV
EXICON0	ERU External Input Control Selection	0010$_H$	U, PV	U, PV
EXICON1	ERU External Input Control Selection	0014$_H$	U, PV	U, PV
EXICON2	ERU External Input Control Selection	0018$_H$	U, PV	U, PV
EXICON3	ERU External Input Control Selection	001C$_H$	U, PV	U, PV
EXOCON0	ERU Output Control Register	0020$_H$	U, PV	U, PV
EXOCON1	ERU Output Control Register	0024$_H$	U, PV	U, PV
EXOCON2	ERU Output Control Register	0028$_H$	U, PV	U, PV
EXOCON3	ERU Output Control Register	002C$_H$	U, PV	U, PV

6.10 Interconnects

This section describes how the ERU0 and ERU1 module is connected within the PSC1Mx system.

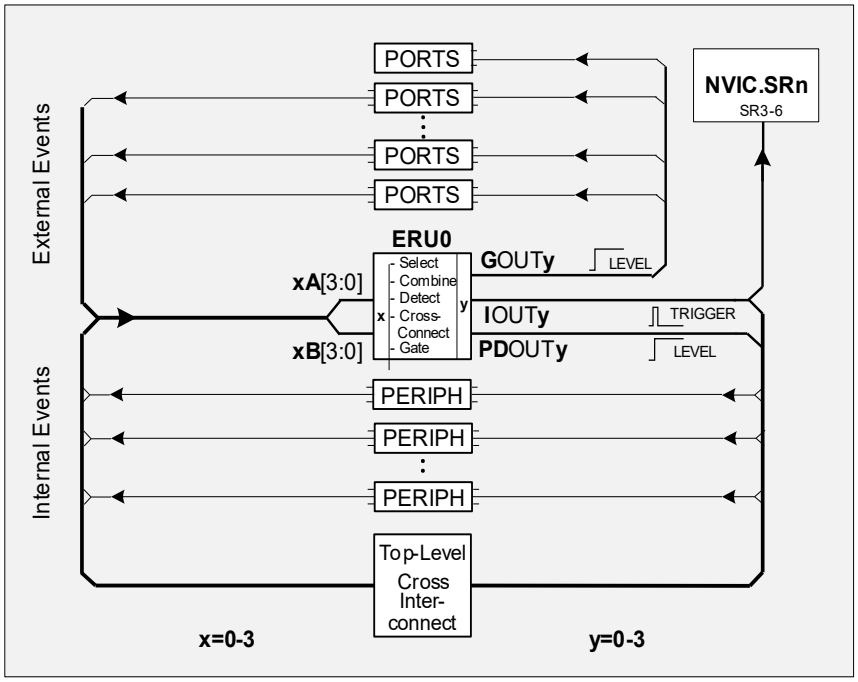


Figure 20 ERU Interconnects Overview

6.10.1 ERU0 Connections

The following table shows the ERU0 connections.

Table 27 ERU0 Pin Connections

Global Input/Output	I/O	Connected To	Description
ERU0.0A0	I	ACMP0.OUT	
ERU0.0A1	I	P2.4	
ERU0.0A2	I	ORC2.OUT	
ERU0.0A3	I	VADC0.G0BFLOUT0	
ERU0.0B0	I	P2.0	
ERU0.0B1	I	P2.2	
ERU0.0B2	I	ORC0.OUT	
ERU0.0B3	I	VADC0.G1BFLOUT0	
ERU0.1A0	I	ACMP1.OUT	
ERU0.1A1	I	P2.5	
ERU0.1A2	I	ORC3.OUT	
ERU0.1A3	I	VADC0.G0BFLOUT1	
ERU0.1B0	I	P2.1	

(table continues...)

Table 27 (continued) ERU0 Pin Connections

Global Input/Output	I/O	Connected To	Description
ERU0.1B1	I	P2.3	
ERU0.1B2	I	ORC1.OUT	
ERU0.1B3	I	VADC0.G1BFLOUT1	
ERU0.2A0	I	ACMP2.OUT	
ERU0.2A1	I	P2.6	
ERU0.2A2	I	ORC4.OUT	
ERU0.2A3	I	VADC0.G0BFLOUT2	
ERU0.2B0	I	P2.10	
ERU0.2B1	I	P2.11	
ERU0.2B2	I	ORC2.OUT	
ERU0.2B3	I	VADC0.G1BFLOUT2	
ERU0.3A0	I	ORC7.OUT	
ERU0.3A1	I	P2.7	
ERU0.3A2	I	ORC5.OUT	
ERU0.3A3	I	VADC0.G0BFLOUT3	
ERU0.3B0	I	P2.9	
ERU0.3B1	I	P2.8	
ERU0.3B2	I	ORC6.OUT	
ERU0.3B3	I	VADC0.G1BFLOUT3	
ERU0.OGU01	I	CCU40.SR0	
ERU0.OGU02	I	VADC0.C0SR2	
ERU0.OGU03	I	CCU80.SR2	
ERU0.OGU11	I	CCU40.SR1	
ERU0.OGU12	I	VADC0.C0SR2	
ERU0.OGU13	I	CCU80.SR2	
ERU0.OGU21	I	CCU40.SR2	
ERU0.OGU22	I	VADC0.C0SR3	
ERU0.OGU23	I	CCU80.SR3	
ERU0.OGU31	I	CCU40.SR3	
ERU0.OGU32	I	VADC0.C0SR3	
ERU0.OGU33	I	CCU80.SR3	

(table continues...)

Table 27 (continued) ERU0 Pin Connections

Global Input/Output	I/O	Connected To	Description
ERU0.PDOUT0	O	CCU80.IN0AF; CCU80.IN1AL; CCU80.IN2AL; CCU80.IN3AL; CCU81.IN0AF; CCU81.IN1AL; CCU81.IN2AL; CCU81.IN3AL; VADC0.BGREQGTO; VADC0.G0REQGTO; VADC0.G1REQGTO; CCU40.IN0AJ; CCU40.IN1AD; CCU41.IN0AJ; CCU41.IN1AD; USIC0_CH1.HWIN0; POSIF0.IN0D; P0.0; P2.11;	
ERU0.GOUT0	O	P0.0; P2.11;	
ERU0.TOUT0	O	not connected	
ERU0.IOUT0	O	NVIC; SCU.RTC_extclkin; VADC0.BGREQTRM; VADC0.G0REQTRM; VADC0.G1REQTRM; CCU40.CLKB; CCU40.IN0AK; CCU41.CLKA; CCU41.IN0AK; CCU80.CLKB; CCU80.IN0AG; CCU81.CLKA; CCU81.IN0AG; POSIF0.EWHEB; BCCU0.TRAPINE;	

(table continues...)

Table 27 (continued) ERU0 Pin Connections

Global Input/Output	I/O	Connected To	Description
ERU0.PDOUT1	O	CCU80.IN0AL; CCU80.IN1AF; CCU81.IN0AL; CCU81.IN1AF; VADC0.BGREQGTP; VADC0.G0REQGTP; VADC0.G1REQGTP; CCU40.IN0AD; CCU40.IN1AJ; CCU41.IN0AD; CCU41.IN1AJ; POSIF0.IN1D; USIC0_CH1.HWIN2; P0.1; P2.10;	
ERU0.GOUT1	O	P0.1;P2.10;	
ERU0.TOUT1	O	not connected	
ERU0.IOUT1	O	NVIC; VADC0.BGREQTRN; VADC0.G0REQTRN; VADC0.G1REQTRN; CCU40.CLKC; CCU40.IN1AK; CCU41.IN1AK; CCU80.CLKC; CCU80.IN1AG; CCU81.IN1AG; BCCU0.TRAPINF;	

(table continues...)

Table 27 (continued) ERU0 Pin Connections

Global Input/Output	I/O	Connected To	Description
ERU0.PDOUT2	O	VADC0.BGREQGTK; VADC0.G0REQGTK; VADC0.G1REQGTK; CCU40.IN3AD; CCU40.IN2AJ; CCU41.IN3AD; CCU41.IN2AJ; CCU80.IN2AF; CCU81.IN2AF; POSIF0.IN2D; P0.2; P2.1;	
ERU0.GOUT2	O	P0.2; P2.1;	
ERU0.TOUT2	O	not connected	
ERU0.IOUT2	O	NVIC; VADC0.BGREQTRG; VADC0.G0REQTRG; VADC0.G1REQTRG; CCU40.IN2AK; CCU41.IN2AK; CCU80.IN2AG; CCU81.IN2AG; POSIF0.MSETF; BCCU0.TRAPING;	
ERU0.PDOUT3	O	VADC0.BGREQGTL; VADC0.G0REQGTL; VADC0.G1REQGTL; CCU40.IN2AD; CCU40.IN3AJ; CCU41.IN2AD; CCU41.IN3AJ; CCU80.IN3AF; CCU81.IN3AF; P0.3; P2.0;	
ERU0.GOUT3	O	P0.3; P2.0;	

(table continues...)

Table 27 (continued) ERU0 Pin Connections

Global Input/Output	I/O	Connected To	Description
ERU0.TOUT3	O	not connected	
ERU0.IOUT3	O	NVIC; VADC0.BGREQTRH; VADC0.G0REQTRH; VADC0.G1REQTRH; CCU40.IN3AK; CCU41.IN3AK; CCU80.IN3AG; CCU81.IN3AG; POSIF0.EWHEC; POSIF0.MSETG; BCCU0.TRAPINH; POSIF1.EWHED;	

6.10.2 ERU1 Connections

The following table shows the ERU1 connections.

Table 28 ERU1 Pin Connections

Global Input/Output	I/O	Connected To	Description
ERU1.0A0	I	ACMP1.OUT	
ERU1.0A1	I	P3.0	
ERU1.0A2	I	P4.4	
ERU1.0A3	I	P4.7	
ERU1.0B0	I	CCU40.ST0	
ERU1.0B1	I	CCU41.ST0	
ERU1.0B2	I	CCU80.ST0	
ERU1.0B3	I	CCU81.ST0	
ERU1.1A0	I	ACMP3.OUT	
ERU1.1A1	I	–	
ERU1.1A2	I	P4.5	
ERU1.1A3	I	–	
ERU1.1B0	I	CCU40.ST1	
ERU1.1B1	I	CCU41.ST1	
ERU1.1B2	I	CCU80.ST3	
ERU1.1B3	I	CCU81.ST3	
ERU1.2A0	I	ACMP2.OUT	

(table continues...)

Table 28 (continued) ERU1 Pin Connections

Global Input/Output	I/O	Connected To	Description
ERU1.2A1	I	–	
ERU1.2A2	I	P4.6	
ERU1.2A3	I	–	
ERU1.2B0	I	CCU40.ST2	
ERU1.2B1	I	CCU41.ST2	
ERU1.2B2	I	CCU80.ST1	
ERU1.2B3	I	CCU81.ST1	
ERU1.3A0	I	ACMP0.OUT	
ERU1.3A1	I	POSIF1.SR1	
ERU1.3A2	I	P2.12	
ERU1.3A3	I	P2.13	
ERU1.3B0	I	CCU40.ST3	
ERU1.3B1	I	CCU41.ST3	
ERU1.3B2	I	CCU80.ST2	
ERU1.3B3	I	CCU81.ST2	
ERU1.OGU01	I	CCU41.SR0	
ERU1.OGU02	I	CAN0.SR4	
ERU1.OGU03	I	CCU81.SR2	
ERU1.OGU11	I	CCU41.SR1	
ERU1.OGU12	I	CAN0.SR5	
ERU1.OGU13	I	CCU81.SR2	
ERU1.OGU21	I	CCU41.SR2	
ERU1.OGU22	I	CAN0.SR6	
ERU1.OGU23	I	CCU81.SR3	
ERU1.OGU31	I	CCU41.SR3	
ERU1.OGU32	I	CAN0.SR7	
ERU1.OGU33	I	CCU81.SR3	

(table continues...)

Table 28 (continued) ERU1 Pin Connections

Global Input/Output	I/O	Connected To	Description
ERU1.PDOUT0	O	CCU40.IN0AW; CCU40.IN1AY; CCU41.IN0AW; CCU41.IN1AY; CCU80.IN0AV; CCU80.IN1AX; CCU80.IN2AX; CCU80.IN3AX; CCU81.IN0AV; CCU81.IN1AX; CCU81.IN2AX; CCU81.IN3AX; POSIF1.IN0D; USIC1_CH1.HWIN0; P1.4;	
ERU1.GOUT0	O	–	
ERU1.TOUT0	O	not connected	
ERU1.IOUT0	O	NVIC; VADC0.BGREQTRE; VADC0.G0REQTRE; VADC0.G1REQTRE; CCU40.CLKA; CCU40.IN0AX; CCU41.IN0AX; CCU41.CLKB; CCU80.CLKA; CCU80.IN0AW; CCU81.CLKB; CCU81.IN0AW; POSIF1.EWHEB; BCCU0.TRAPINJ;	

(table continues...)

Table 28 (continued) ERU1 Pin Connections

Global Input/Output	I/O	Connected To	Description
ERU1.PDOUT1	O	CCU40.IN0AY; CCU40.IN1AW; CCU41.IN0AY; CCU41.IN1AW; CCU80.IN0AX; CCU80.IN1AV; CCU81.IN0AX; CCU81.IN1AV; POSIF1.IN1D; USIC1_CH1.HWIN2; P1.1; P1.5; P4.1;	
ERU1.GOUT1	O	P4.1	
ERU1.TOUT1	O	not connected	
ERU1.IOUT1	O	NVIC; CCU41.CLKC; CCU40.IN1AX; CCU41.IN1AX; CCU80.IN1AW; CCU81.CLKC; CCU81.IN1AW; BCCU0.TRAPINK;	
ERU1.PDOUT2	O	CCU40.IN2AW; CCU40.IN3AY; CCU41.IN2AW; CCU41.IN3AY; CCU80.IN2AV; CCU81.IN2AV; POSIF1.IN2D; P1.2; P1.6; P4.2;	
ERU1.GOUT2	O	P4.2	
ERU1.TOUT2	O	not connected	

(table continues...)

Table 28 (continued) ERU1 Pin Connections

Global Input/Output	I/O	Connected To	Description
ERU1.IOUT2	O	NVIC; CCU40.IN2AX; CCU41.IN2AX; CCU80.IN2AW; CCU81.IN2AW; POSIF1.MSETF; BCCU0.TRAPINL;	
ERU1.PDOUT3	O	CCU40.IN3AW; CCU40.IN2AY; CCU41.IN2AY; CCU41.IN3AW; CCU80.IN3AV; CCU81.IN3AV; P1.3; P4.3;	
ERU1.GOUT3	O	P4.3	
ERU1.TOUT3	O	not connected	
ERU1.IOUT3	O	NVIC; CCU40.IN3AX; CCU41.IN3AX; CCU80.IN3AW; CCU81.IN3AW; POSIF1.EWHEC; POSIF1.MSETG; POSIF0.EWHED; BCCU0.TRAPINM;	

7 MATH Coprocessor (MATH)

7.1 Overview

The MATH Coprocessor (MATH) module comprises of two independent sub-blocks to support the CPU in math-intensive computations: a Divider Unit (DIV) for signed and unsigned 32-bit division operations and a CORDIC (COrdinate Rotation DIgital Computer) Coprocessor for computation of trigonometric, linear or hyperbolic functions.

7.1.1 Features

The MATH Coprocessor includes the following features:

- Divide function with operand pre-processing and result post-processing
- CORDIC Coprocessor for computation of trigonometric, hyperbolic and linear functions
- Supports kernel clock to interface clock ratio of 2:1 for faster execution
- Supports result chaining between the Divider Unit and CORDIC Coprocessor

7.1.2 Block Diagram

Figure 21 shows a block diagram of the MATH Coprocessor.

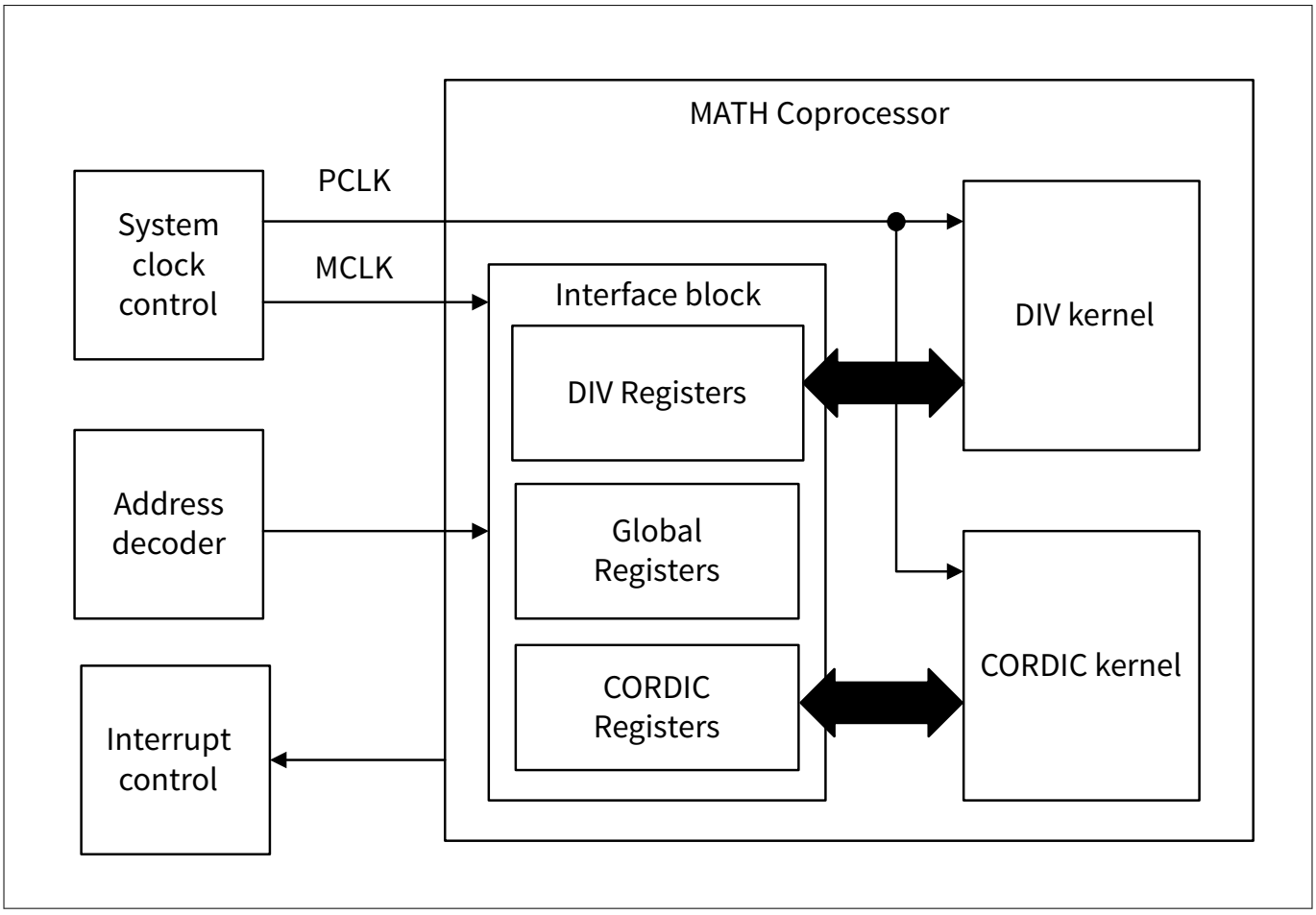


Figure 21 MATH Coprocessor Block Diagram

7.2 Divider Unit (DIV)

7.2.1 Features

The DIV supports the following features:

- Signed/unsigned division operation in 35 kernel clock cycles
- Three division modes:
  - 32-bit divide by 32-bit
  - 32-bit divide by 16-bit
  - 16-bit divide by 16-bit
- Operands pre-processing with configurable number of:
  - Left shifts for dividend
  - Right shifts for divisor
- Result post-processing with configurable number of shifts and shift direction

***Note**: The execution time of 35 kernel clock cycles for a division operation does not include the time to access the operand and result registers, which can take up a large part of the time for a division function in the application software.*

7.2.2 Division Operation

The DIV supports the truncated division operation, which is also the ISO C99 standard and the popular choice among modern processors. The division and modulus functions of the truncated division are related in the following way:

If q = D div d

and r = D mod d

then D = q * d + r

and | r | < | d |

where “D” is the dividend, “d” is the divisor, “q” is the quotient and “r” is the remainder. The truncated division rounds the quotient towards zero and the sign of its remainder is always the same as that of its dividend, i.e., sign (r) = sign (D).

To execute a divider operation with the DIV, it is first required to configure the division as follows:

- Signed or unsigned through the DIVCON.USIGN bit
- Division mode through the DIVCON.DIVMODE bits
- Start mode through the DIVCON.STMODE bit

The dividend and divisor values are then written into the DVD and DVS registers. The division is started with the write to DVS register or by setting the start bit DIVCON.ST, depending on the start mode. If the ST bit is used, the bit is automatically cleared in the next kernel clock cycle. The start of the division operation sets the busy flag, DIVST.BSY.

The division operation always takes 35 kernel clock cycles, and upon completion, the quotient and remainder values will be available at the QUOT and RMD registers, and the BSY flag will be cleared. The end of calculation event sets the Divider event flag EVFR.DIVEOC and can trigger an interrupt request to NVIC if enabled through the EVIER.DIVEOCIEN bit. The flag is cleared only by a software write to the EVFCR.DIVEOCC bit.

Figure 22 shows the timing diagram for a division operation.

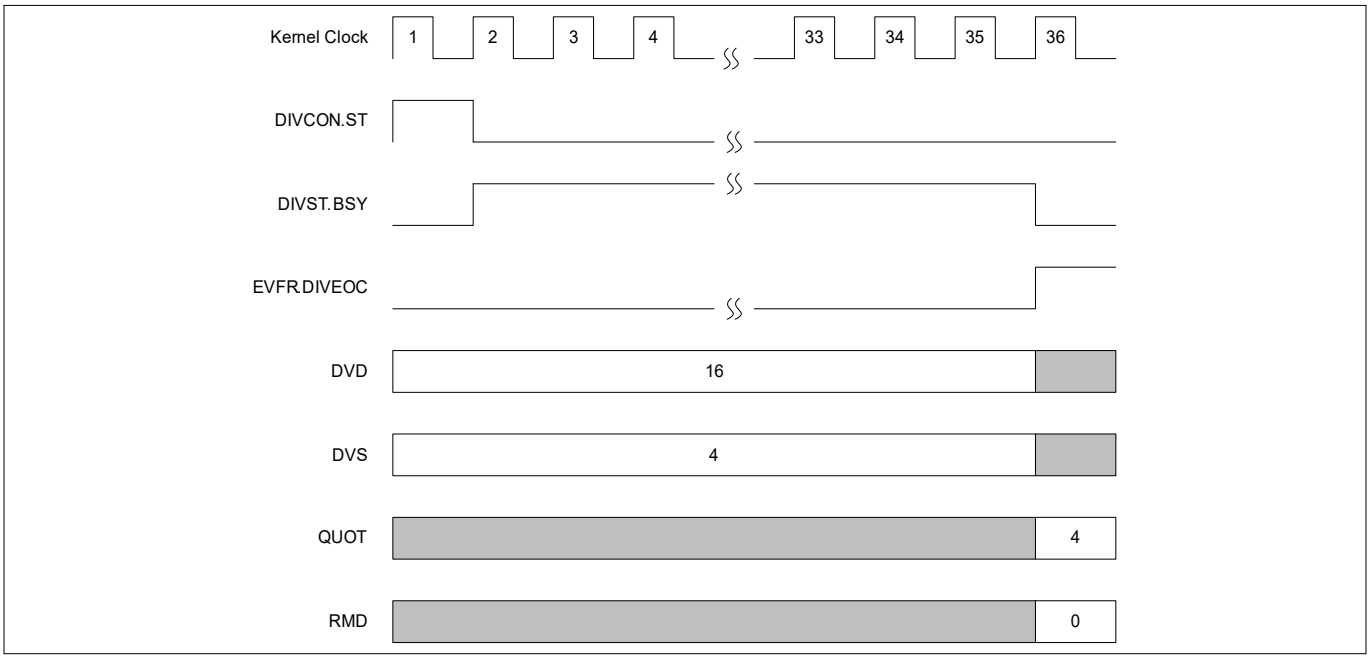


Figure 22 Timing Diagram for a Division Operation

***Note**: Reading the QUOT and RMD registers while BSY=1 will cause the DIV to insert wait states onto the bus until the active calculation is completed (BSY=0). This ensures that any read access on the result registers QUOT or RMD returns a valid result. However, the interrupt latency will be increased as the bus may be locked up for a number of kernel clock cycles.*

7.2.2.1 Start Mode Selection

The condition to start a division operation is selectable through the DIVCON.STMODE bit:

- When STMODE = 0, a division operation is started with a write to the DVS register. In this case, no further software set of the ST bit is necessary.
- When STMODE = 1, a division is started by setting the ST bit.

For both start modes, it must be ensured that the DIV is not performing any active calculation and the DIVST.BSY flag is 0 before starting the operation, else the start request is discarded though DVS will still be updated with the write value. Write access to DIVCON register is ignored while BSY = 1. It is recommended for the application to poll for this condition before starting the divider operation with a write to ST bit or DVS register.

7.2.2.2 Error Handling

The DIV supports two types of error detection:

- Divide by zero error
- Overflow error

In both cases, the error will be indicated by the EVFR.DIVERR flag. An interrupt request to NVIC can be generated if it is enabled through EVIER.DIVERRIEN.

The division operation will still proceed as normal and complete in 35 kernel clock cycles. The error flag becomes set at the same clock cycle as DIVEOC.

Divide by Zero Error

A divide by zero error occurs when a division operation is started with the divisor value in DVS register equal to 0.

An example of a divide by zero error is shown in Figure 23.

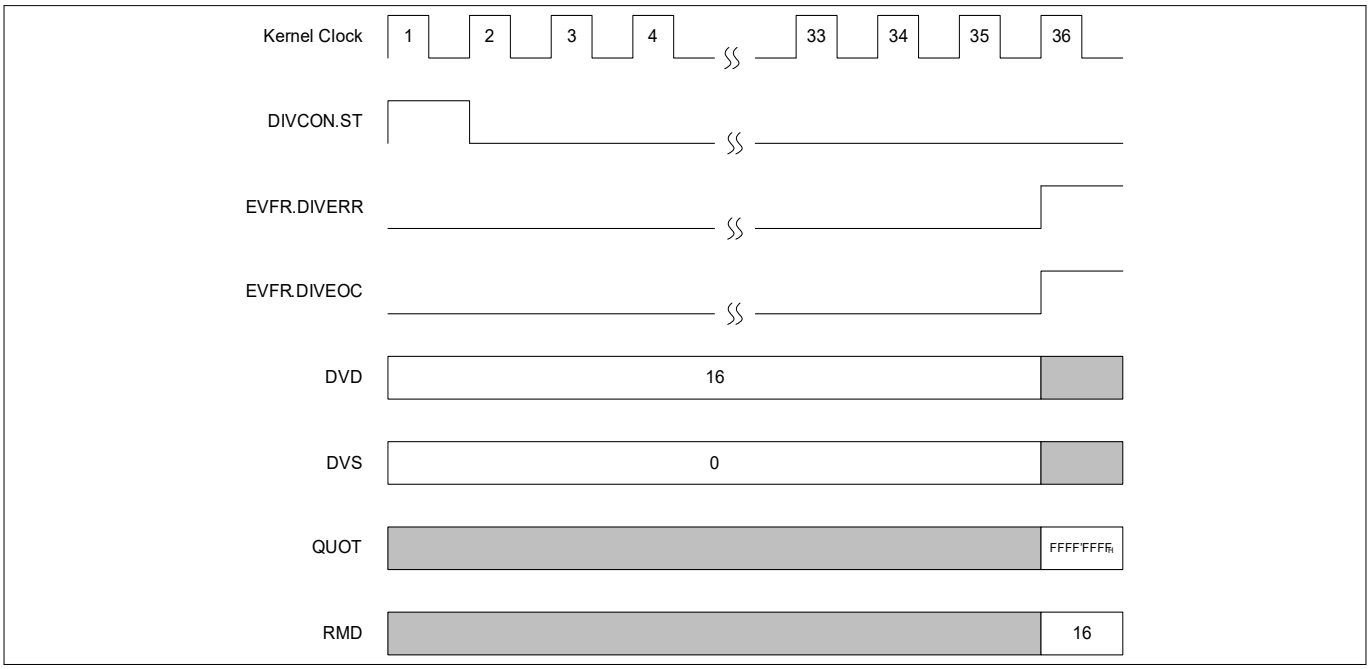


Figure 23 Timing Diagram with Divide by Zero Error

If DIVCON.USIGN is 0 (signed operation) and DVD is a negative value, the QUOT register will contain all zeros, else it will contain all ones. In either case, the RMD register will contain the value of the dividend. The flag is cleared only by a software write to the EVFCR.DIVERRC bit.

***Note**: If result post-processing (see Operand/Result Pre-/Post-Processing) is enabled, the value of the quotient (all zeros or all ones) will still be shifted according to DIVCON.QSCNT.*

Overflow Error

An overflow error occurs when one of the three conditions listed in Table 29 is detected.

Table 29 Overflow Error Conditions

DIVCON.DIVMODE	Dividend (Hex)	Dividend (Dec)	Divisor (Hex)	Divisor (Dec)
00_B	$8000'0000_H$	-2^{31}	$FFFF'FFFF_H$	-1
01_B	$8000'0000_H$	-2^{31}	$FFFF_H$	-1
10_B	8000_H	-32768	$FFFF_H$	-1

The DIV does not detect for overflow errors that is due to result post-processing. In this case, user must always ensure that the result after post-processing is still within the boundaries of DIV.

7.2.3 Operand/Result Pre-/Post-Processing

The DIV supports operand pre-processing and result post-processing by allowing the following:

- Left shift of the dividend value before the start of division
- Right shift of the divisor value before the start of division
- Left or right shift of the quotient value after the end of division

The number of shifts is determined by the respective 5-bit shift count bit fields in DIVCON register. Additionally for the quotient, the shift direction is defined by the bit DIVCON.QSDIR.

All shifts are arithmetic shifts. This means if shift left, zeros will be inserted at LSB, while if shift right, zeros (in unsigned mode) or the signed bit (in signed mode) will be inserted at MSB.

***Note**: For the case where left shifting of the operand causes the signed bit to change from 0 to 1, the divider handles the signed bit as the MSB of the data value and ignores the signed effect.*

If selected to be enabled by writing a non-zero value to the shift count bit fields, the shift operations will be triggered at the start and end of the division operation and do not consume any additional kernel clock cycles. The DIV event flag will be generated at the end of the 35-clock execution cycle.

During operand pre-processing, the shifts are performed only on the output of the DVD and DVS registers and not on the registers themselves. Therefore, the contents of these registers remain unchanged.

7.3 CORDIC Coprocessor

The CORDIC Coprocessor computes trigonometric, linear, hyperbolic and related functions.

7.3.1 Overview

This document describes the features of the CORDIC Coprocessor. It is not the purpose of this document to cover the theoretical background of the CORDIC algorithm.

The CORDIC algorithm is a useful convergence method for computing trigonometric, linear, hyperbolic and related functions. It allows performance of vector rotation not only in the Euclidian plane, but also in the Linear and Hyperbolic planes.

The CORDIC algorithm is an iterative process where truncation errors are inherent. Higher accuracy is achieved in the CORDIC Coprocessor with 27 iterations per calculation and kernel data width of at least 32 bits. The main advantage of using this algorithm is the fast calculation speed compared to software and high accuracy.

The generalized CORDIC algorithm has the following CORDIC equations. The factor *m* controls the vector rotation and selects the set of angles for the circular, linear and hyperbolic function:

$$x_{i+1} = x_i - m \cdot d_i \cdot y_i \cdot 2^{-i} \quad (1)$$

$$y_{i+1} = y_i + d_i \cdot x_i \cdot 2^{-i} \quad (2)$$

$$z_{i+1} = z_i - d_i \cdot e_i \quad (3)$$

Where

$m = 1$ Circular function (basic CORDIC) with $e_i = \text{atan}(2^{-i})$

$m = 0$ Linear function with $e_i = 2^{-i}$

$m = -1$ Hyperbolic function with $e_i = \text{atanh}(2^{-i})$

For clarity, the document uses the following terms for referencing CORDIC data:

- Result Data: Final result data at the end of CORDIC calculation (Bit BSY no longer active).
- Calculated Data: Intermediate or last data resulting from CORDIC iterations.
- Initial Data: Data used for the very first CORDIC iteration, is usually user-initialized data.

7.3.1.1 Features

The key features of the CORDIC Coprocessor are listed below:

- Modes of operation
  - Supports all CORDIC operating modes for solving circular (trigonometric), linear (multiply-add, divide-add) and hyperbolic functions
  - Integrated look-up tables (LUTs) for all operating modes
- Circular vectoring mode: Extended support for values of initial X and Y data up to full range of $[-2^{23},(2^{23}-1)]$ for solving angle and magnitude
- Circular rotation mode: Extended support for values of initial Z data up to full range of $[-2^{23},(2^{23}-1)]$, representing angles in the range $[-\pi,((2^{23}-1)/2^{23})\pi]$ for solving trigonometry
- 24-bit accessible data width
  - 32-bit kernel data width plus 2 sign extension bits and 1 overflow bit for X and Y each
  - 29-bit kernel data width plus 1 sign extension bit and 1 overflow bit for Z
  - With KEEP bit to retain the last value in the kernel register for a new calculation
- 27 iterations per calculation: 62 kernel clock cycles or less, from set of start (ST) bit to set of end-of-calculation flag, excluding time taken for write and read access of data bytes.
- Twos complement data processing
  - Only exception: X result data with user selectable option for unsigned result
- X and Y data generally accepted as integer or rational number; X and Y must be of the same data form
- Truncation Error
  - The result of a CORDIC calculation may return an approximation due to truncation of LSBs
  - Good accuracy of the CORDIC calculated result data, especially in circular mode
- Interrupt
  - On completion of a calculation
  - Interrupt enabling and corresponding flag

Table 30 CORDIC Applications

Use Case	Application
Angle computation	EPS, Motor control
Park transformation	Motor control
Y +XZ	Digital filtering, PI control loop

7.3.1.2 Block Diagram

Figure 24 shows a block diagram of the CORDIC Coprocessor kernel.

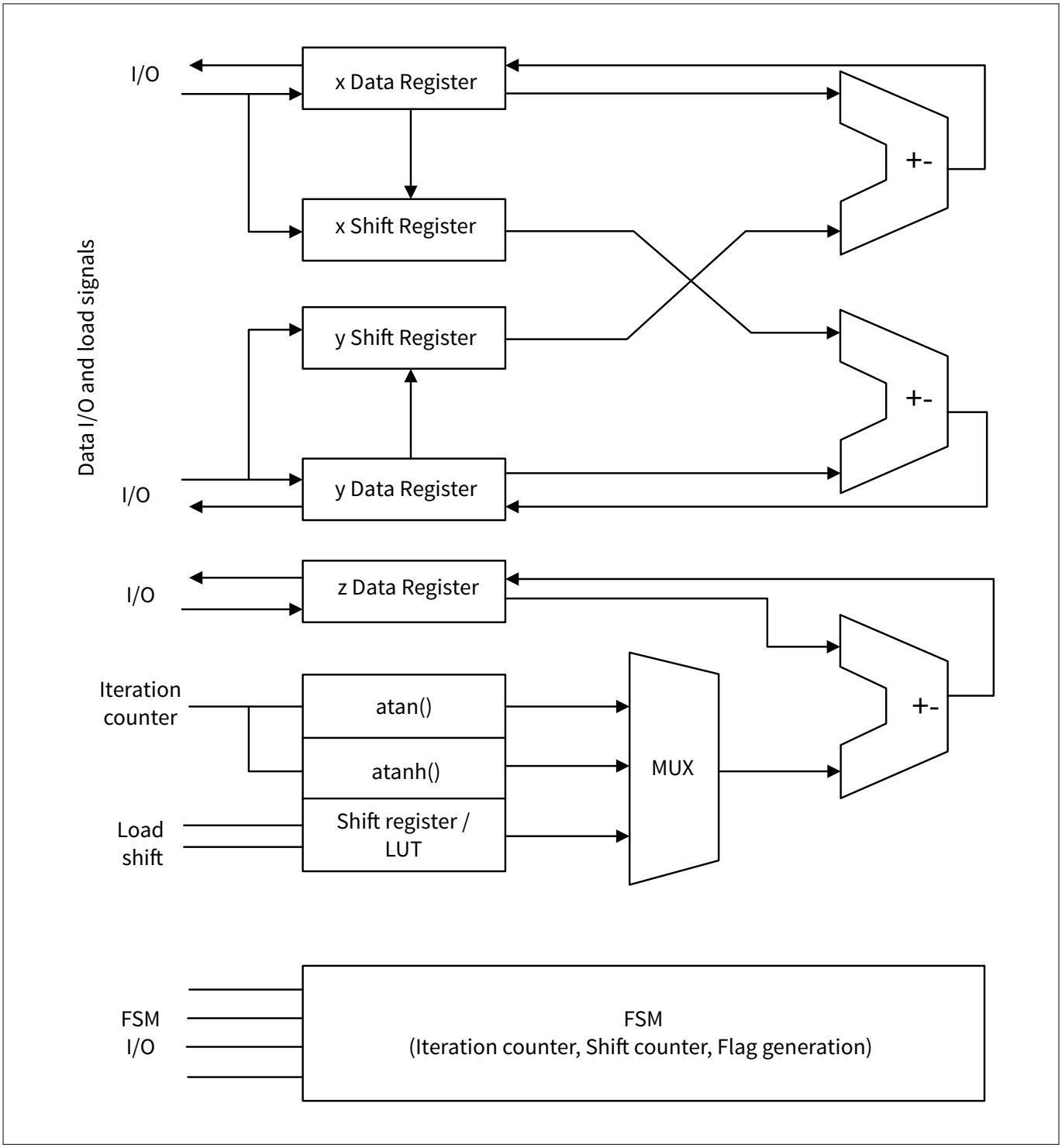


Figure 24 CORDIC Block Diagram

7.3.2 Functional Overview

The following sections describe the function of the CORDIC Coprocessor.

7.3.2.1 Operation of the CORDIC Coprocessor

The CORDIC Coprocessor can be used for the circular (trigonometric), linear (multiply-add, divide-add) or hyperbolic function, in either rotation or vectoring mode. The modes are selectable by software via the CON control register.

Initialization of the kernel data register is enabled by clearing respective STATC.KEEPx bits of the register STATC. If CON.ST_MODE = 1, writing 1 to bit CON.ST starts a new calculation. Otherwise, by default where CON.ST_MODE = 0, a new calculation starts after a write access to register CORDX. Each calculation involves a fixed number of 27 iterations. Bit STATC.BSY is set while a calculation is in progress to indicate busy status. It is cleared by hardware at the end of a calculation.

As the first step on starting a CORDIC calculation (provided the corresponding KEEP bits are not set), the initial data is loaded from the data registers CORDx to the internal kernel data registers. During the calculation, the kernel data registers always hold the latest intermediate data. On completion of the calculation, the result data will be loaded to the result registers CORRx.

The data registers CORDx function as shadow registers which can be written to without affecting an ongoing calculation. Values are transferred to the kernel data registers only on valid setting of bit CON.ST, or if CON.ST_MODE = 0, after write access to X data register CORDX (provided KEEP bit of corresponding data is not set). The result data can be read from the result registers CORRx at the end of calculation (BSY no longer active). The result data needs to be read before the start of the next calculation. Any read access on the result registers CORRx while STATC.BSY is set (kernel is still running a calculation) will cause the kernel to issue a bus wait until STATC.BSY is reset. This will ensure that any read access on the result registers CORRx returns a valid result.

At the end of each calculation, STATC.BSY returns to 0. The EVFR.CDEOC bit, which denotes the CORDIC end-of-calculation, is set. The interrupt request signal will be activated if interrupt is enabled by EVIER.CDEOCIEN = 1. The EVFR.CDEOC flag will have to be cleared manually via software by setting EVFCR.CDEOCC = 1. The result data in X, Y and Z are internally checked, and in case of data overflow, the EVFR.CDERR bit is set. An interrupt request signal will be activated if interrupt is enabled by EVIER.CDERRIEN = 1. The EVFR.CDERR flag will have to be cleared manually via software by setting EVFCR.CDERRC = 1.

Setting the bit CON.ST during an ongoing calculation (if CON.ST_MODE = 1) while STATC.BSY is set has no effect. In order to start a new calculation, bit CON.ST must be set again at a later time when STATC.BSY is no longer active. In the same manner, changing the operating mode during a running calculation (as indicated by STATC.BSY) has no effect.

In automatic start mode (CON.ST_MODE = 0), if write access to CORDX occurs while STATC.BSY is set, the new calculation will not be started, similar to the case with CON.ST_MODE = 1.

7.3.2.2 Normalized Result Data

In all operating modes, the CORDIC Coprocessor returns a normalized result data for X and Y, as shown in the following equation:

$$\text{X or Y Result Data} = \frac{\text{CORDIC Calculated Data}}{\text{MPS}} \quad (4)$$

On the other hand, the interpretation for Z result data differs, which is also dependent on the CORDIC function used:

For **linear** function, there is no additional processing of the CORDIC calculated Z data, as such it is taken directly as the result data. The accessible Z result data is a real number expressed as signed 4Q19.

For **circular** and **hyperbolic** functions, the accessible Z result data is a normalized integer value, angles in the range $[-\pi,((2^{23}-1)/2^{23})\pi]$ are represented by $[-2^{23},(2^{23}-1)]$. The CORDIC Coprocessor expects Z data to be interpreted with this scaling:

$$\text{Input Z Initial Data} = \text{Real Z Initial Value (in radians)} \times \frac{8388608}{\pi} \quad (5)$$

$$\text{Real Z Result Value (in radians)} = \text{Z Result Data} \times \frac{\pi}{8388608} \quad (6)$$

The CORDIC calculated data includes an inherent gain factor K resulting from the rotation or vectoring. The value K is different for each CORDIC function, as shown in Table 31.

Table 31 CORDIC Function Inherent Gain Factor for Result Data

Function	Approximated Gain K
Circular	1.646760258121
Hyperbolic	0.828159360960
Linear	1

7.3.3 CORDIC Coprocessor Operating Modes

Table 32 gives an overview of the CORDIC Coprocessor operating modes. In this table, X, Y and Z represent the initial data, while X_{final}, Y_{final} and Z_{final} represent the final result data when all processing is complete and BSY is no longer active.

The CORDIC equations are:

$$x_{i+1} = x_i - m \cdot d_i \cdot y_i \cdot 2^{-i} \quad (7)$$

$$y_{i+1} = y_i + d_i \cdot x_i \cdot 2^{-i} \quad (8)$$

$$z_{i+1} = z_i - d_i \cdot e_i \quad (9)$$

Table 32 CORDIC Coprocessor Operating Modes and Corresponding Result Data

Function	Rotation Mode	Vectoring Mode
	d_i = sign (z_i), $z_i \rightarrow 0$	d_i = -sign (y_i), $y_i \rightarrow 0$
Circular $m = 1$ $e_i = \text{atan}(2^{-i})$	$X_{final} = K[X \cos(Z) - Y \sin(Z)]$ / MPS $Y_{final} = K[Y \cos(Z) + X \sin(Z)]$ / MPS $Z_{final} = 0$ where K ≈ 1.646760258121	$X_{final} = K \text{ sqrt}(X^2+Y^2)$ / MPS $Y_{final} = 0$ $Z_{final} = Z + \text{atan}(Y / X)$ where K ≈ 1.646760258121
	For solving cos(Z) and sin(Z), set X = 1 / K, Y = 0. Useful domain: Full range of X, Y and Z supported due to pre-processing logic.	For solving magnitude of vector (sqrt(x^2+y^2)), set X = x / K, Y = y / K. Useful domain: Full range of X and Y supported due to pre- and post-processing logic. For solving atan(Y / X), set Z = 0. Useful domain: Full range of X and Y, except X = 0.
	Relationships: tan(v) = sin(v) / cos(v)	Relationships: acos(w) = atan[sqrt($1-w^2$) / w] asin(w) = atan[w / sqrt($1-w^2$)]
Linear $m = 0$ $e_i = 2^{-i}$	$X_{final} = X$ / MPS $Y_{final} = [Y + X Z]$ / MPS $Z_{final} = 0$	$X_{final} = X$ / MPS $Y_{final} = 0$ $Z_{final} = Z + Y / X$
	For solving $X \cdot Z$, set Y = 0. Useful domain: $\lvert Z \rvert \le 2$.	For solving ratio Y / X, set Z = 0. Useful domain: $\lvert Y / X \rvert \le 2$, $X > 0$.
Hyperbolic $m = -1$ $e_i = \text{atanh}(2^{-i})$	$X_{final} = k[X \cosh(Z) + Y \sinh(Z)]$ / MPS $Y_{final} = k[Y \cosh(Z) + X \sinh(Z)]$ / MPS $Z_{final} = 0$ where k ≈ 0.828159360960	$X_{final} = k \text{ sqrt}(X^2-Y^2)$ / MPS $Y_{final} = 0$ $Z_{final} = Z + \text{atanh}(Y / X)$ where k ≈ 0.828159360960

(table continues...)

Table 32 (continued) CORDIC Coprocessor Operating Modes and Corresponding Result Data

Function	Rotation Mode	Vectoring Mode
	For solving cosh(*Z*) and sinh(*Z*) and e^Z, set *X* = 1 / k, *Y* = 0. Useful domain: |*Z*| ≤ 1.11rad, *Y* = 0.	For solving sqrt(x^2-y^2), set *X* = x / k, *Y* = y / k. Useful domain: |y| < |x|, *X* > 0. For solving atanh(*Y* / *X*), set *Z* = 0. Useful domain: |atanh(*Y* / *X*)| ≤ 1.11rad, *X* > 0.
	Relationships: tanh(v) = sinh(v) / cosh(v) e^v = sinh(v) + cosh(v) $w^t = e^{t\ ln(w)}$	Relationships: ln(w) = 2 atanh[(w-1) / (w+1)] sqrt(w) = sqrt($(w+0.25)^2$-$(w-0.25)^2$) acosh(w) = ln[w+sqrt(1-w^2)] asinh(w) = ln[w+sqrt(1+w^2)]

Usage Notes

- For solving the respective functions, user must initialize the CORDIC data (X, Y and Z) with meaningful initial values within domain of convergence to ensure result convergence. The 'useful domain' listed in Table 32 covers the supported domain of convergence for the CORDIC algorithm and excludes the not-meaningful range(s) for the function. For details regarding the supported domain of convergence, refer to Domains of Convergence. For result data accuracy, refer to Accuracy of CORDIC Coprocessor.
- Function limitations must be considered, for example, setting initial X = 0 for atan(Y / X) is not meaningful. Violations of such function limitations may yield incoherent CORDIC result data.
- All data inputs are processed and handled as twos complement. Only exception is user-option for X result data (only) to be read as unsigned value.
- The only case where the result data is always positive and larger than the initial data is X result data (only) in circular vectoring mode; therefore, the user may want to use the MSB bit as data bit instead of sign bit. By setting X_USIGN = 1, X result data will be processed as unsigned data.
- For circular and hyperbolic functions, and due to the corresponding fixed LUT, the Z data is always handled as signed integer S28 (accessible as S23). The LUTs contain scaled integer values (S28) of atan(2^{-i}) for i = 0, 1, 2, ..., 27 and atanh(2^{-i}) for i = 1, 2, ..., 27, such that angles in the range [-π,((2^{23}-1)/2^{23})π] are represented by integer values ranging [-2^{23},(2^{23}-1)]. Therefore, Z data is limited (not considering domain of convergence) to represent angles [-π,((2^{23}-1)/2^{23})π] for these CORDIC functions. Any calculated value of Z outside of this range will result in overflow error.
- For linear function, the Z data is always handled as signed fraction S4.24 (accessible as S4.19 in the form signed 4Q19). The emulated LUT is actually a shift register that holds data in the form 1.23 which gives the real value of 2^{-i}. Therefore, regardless of the domain of convergence, Z data is logically only useful for values whose magnitude is smaller than 16. Overflow error is indicated by the EVFR.CDERR bit.
- The MPS setting has no effect on Z data. User must ensure proper initialization of Z initial data to prevent overflow and incorrect result data.
- The CORDIC Coprocessor is designed such that with correct user setting of MPS > 1, there is no internal overflow of the X and Y data and the read result data is complete. However, note that in these cases, the higher the MPS setting, the lower the resolution of the result data due to loss of LSB bit(s).
- The hyperbolic rotation mode is limited, in terms of result accuracy, in that initial Y data must be set to zero. In other words, the CORDIC Coprocessor is not able to return accurate result for cosh(Z)+/-sinh(Z) in a single calculation.

7.3.3.1 Domains of Convergence

It is not intended for this document to cover the theory of the inherent limits of CORDIC convergence, which varies with different CORDIC function in rotation or vectoring mode.

For convergence of result data, there are limitations to the magnitude or value of initial data and corresponding useful data form, depending on the operating mode used. The following are generally applicable regarding convergence of CORDIC result data.

Rotation Mode: Z data must converge towards 0. Initial Z data must be equal or smaller than $\Sigma d_i \cdot e_i$, where e_i is always decreasing for iteration i. In other words, |Z| ≤ Sum of LUT. In circular function, this means |Z| ≤ integer value representing 1.74 radians. For linear function, |Z| ≤ 2. In hyperbolic function, |Z| ≤ integer value representing 1.11 radians.

Vectoring Mode: Y data must converge towards 0. The values of initial X and Y are limited by the Z function which is dependent on the corresponding LUT. For circular function, this means |atan(Y / X)| ≤ 1.74 radians. For linear function, |Y / X| ≤ 2. For hyperbolic function, |atanh(Y / X)| ≤ 1.11 radians. In vectoring mode, the additional requirement is that X > 0.

While the operating modes of the CORDIC Coprocessor are generally bounded by these convergence limits, there are exceptions for the circular rotation and circular vectoring modes which use additional pre- (and post-)processing logic to support wider range of inputs.

Circular Rotation Mode: The full range of Z input $[-2^{23},(2^{23}-1)]$ representing angles $[-\pi,((2^{23}-1)/2^{23})\pi]$ is supported. No limitations on initial X and Y inputs, except for overflow considerations which can be overcome with MPS setting.

Circular Vectoring Mode: The full range of X and Y inputs $[-2^{23},(2^{23}-1)]$ are supported, while Z initial value should satisfy $|Z| \le \pi / 2$ to prevent possible Z result data overflow.

***Note**: Considerations should also be given to function limitations such as the meaning of the result data, for example, divide by zero is not meaningful. The 'useful domain' included within Table 32 for each of the main functions, attempts to cover both for CORDIC convergence and useful range of the function.*

***Note**: Input values may be within the domain of convergence, however, this does not guarantee a fixed level of accuracy of the CORDIC result data. Refer to Accuracy of CORDIC Coprocessor for details on accuracy of the CORDIC Coprocessor.*

7.3.3.2 Overflow Considerations

Besides considerations for domain of convergence, the limitations on the magnitude of input data must also be considered to prevent result data overflow.

Data overflow is handled by the CORDIC Coprocessor in the same way in all operating modes. Overflow for X and Y data can be prevented by correct setting by the user of the MPS bit, whose value is partly based on the CORDIC Coprocessor operating mode and the application data.

The MPS setting has no effect on the Z data. For circular and hyperbolic functions, any value of Z outside of the range $[-\pi,((2^{23}-1)/2^{23})\pi]$ cannot be represented and will result in Z data overflow error. Note that kernel data Z has values in the range $[-\pi,((2^{23}-1)/2^{23})\pi]$ scaled to the range $[-2^{23},(2^{23}-1)]$, so the written and read values of Z data are always normalized as such. For linear function, where Z is a real value, magnitude of Z must not exceed 4 integer bits.

7.3.4 CORDIC Coprocessor Data Format

The CORDIC Coprocessor accepts (initial) data X, Y and Z inputs in twos complement format. The result data are also in twos complement format.

The only exception is for the X result data in circular vectoring mode. The X result data has a default data format of twos complement, but the user can select via bit CON.X_USIGN = 1 for the X result data to be read as unsigned value. This option prevents a potential overflow of the X result data (taken together with the MPS setting), as the MSB bit is now a data bit. Note that setting bit CON.X_USIGN = 1 is only effective when operating in the circular vectoring mode, which always yields result data that is positive and larger than the initial data.

Generally, the input data for X and Y can be integer or rational number (fraction). However, in any calculation, the data form must be the same for both X and Y. Also, in case of fraction, X and Y must have the same number of bits for decimal place.

The Z data is always handled as integer, based on the normalization factor for circular or hyperbolic function. In case of linear function, accessible Z data is a real number with fixed input and result data form of S4.19 (signed 4Q19) which is a fraction with 19 decimal places.

Refer to Normalized Result Data for details on data normalization.

Table 33 summarizes the initial and result data format.

Table 33 Summary of X,Y and Z data format

Function	Data		Mode	
			Rotation	**Vectoring**
Circular	X	Initial	24-bit 2's complement	
		Result	24-bit 2's complement	24-bit 2's complement /24-bit unsigned data(CON.X_USIGN = 1)
	Y	Initial	24-bit 2's complement	
		Result		
	Z	Initial		
		Result		
Linear	X	Initial		
		Result		
	Y	Initial		
		Result		
	Z	Initial	S4.19	
		Result		
Hyperbolic	X	Initial	24-bit 2's complement	
		Result		
	Y	Initial		
		Result		
	Z	Initial		
		Result		

7.3.5 Accuracy of CORDIC Coprocessor

Each CORDIC calculation involves a fixed number of 27 CORDIC iterations starting from iteration 0. The hyperbolic function is special in this respect in that it starts from iteration 1 with repeat iterations at defined steps. The addressable data registers are 24 bits wide, while the internal kernel X and Y data registers used for calculation are each 35 bits wide (32 data bits plus 2 sign extension bits and 1 overflow bit) and internal kernel Z data register is 31 bits wide (29 data bits plus 1 sign extension bit and 1 overflow bit). For more details on the data form of the LUTs, refer to Arctangent and Hyperbolic Arctangent Look-Up Tables and Linear Function Emulated Look-Up Table.

For input data values within the specified useful domain (see Table 32), the result of each calculation of the CORDIC Coprocessor is guaranteed to converge, although the accuracy is not fixed per data form in each operating mode. The accuracy is a measure of the magnitude of the difference between the result data and the expected data from a high-accuracy calculator. “Normalized Deviation” (ND) is a generic term used to refer to the magnitude of deviation of the result data from the expected result. The deviation is calculated as if the input/result data is integer. In case the data is a rational number, the magnitude of deviation has to be interpreted. For example, Z for linear vectoring mode of the data form S4.19 - ND = 1 (01_B) means the difference from expected real data has magnitude of no more than $|2^{-19} + 2^{-19}|$; ND = 2 (10_B) means the difference is no more than $|2^{-18}+2^{-19}|$; ND = 3 (11_B) means the difference is no more than $|2^{-19}+2^{-18}+2^{-19}|$; ND = 4 ($100_B$) means the difference is no more than $|2^{-17}+2^{-19}|$, and so on. The value of 2^{-19} is always added to account for possible truncation error.

Table 34 lists the probability of Normalized Deviation in a single calculation, with approximately one million different input sets for each respective CORDIC Coprocessor operating mode, based on the input conditions specified (always within useful domain, possibly with additional conditions).

The accuracy of each mode can be easily increased, by working with rational numbers (fraction) instead of integers. This refers to X and Y data only (X and Y must always be of same data form), while the data form of Z is fixed per the respective LUT’s definition. It is obvious to expect that for a given input of X and Y (and Z), the calculated result will always return a constant value—regardless of whether X and Y are integers or rational numbers. The only difference is with regards to interpreting the input and result data, i.e., with no decimal place or how many decimal places. The deviation of the CORDIC result from the expected data is never smaller if X and Y are integers instead of rational numbers. Therefore, wherever possible, assign X and Y as rational numbers with carefully selected decimal place point, which could be based on the maximum ND of that mode.

Table 34 Normalized Deviation of a Calculation

Mode	X Normalized Deviation	Y or Z Normalized Deviation
Circular Vectoring	Input conditions: Useful Domain and [(1.64676/2)·sqrt(X^2+Y^2) ≥ 600]	
	0 : 97.8830%	0 : 69.6738%
	1 : 2.1150%	1 : 29.8307%
	2 : 0.0010%	2 : 0.4919%
	3 : 0.0010%	3 : 0.0036%
	ND for X ≤ 3	ND for Z ≤ 3
Circular Rotation	Input conditions: Useful Domain (Full range of X, Y and Z)	

(table continues...)

Table 34 (continued) Normalized Deviation of a Calculation

Mode	X Normalized Deviation	Y or Z Normalized Deviation
	0 : 47.8958% 1 : 50.3470% 2 : 1.7561% 3 : 0.0006% 4 : 0.0003% 5 : 0.0002% ND for X ≤ 5	0 : 47.8773% 1 : 50.3626% 2 : 1.7438% 3 : 0.0103% 4 : 0.0052% 5 : 0.0008% ND for Y ≤ 5
Linear Vectoring	Input conditions: Useful Domain (|Y / X| ≤ 2, X > 0)	
	0 : 99.5869% 1 : 0.4131% ND for X ≤ 1	0 : 47.6909% 1 : 52.2723% 2 : 0.0239% 3 : 0.0129% ND for Z ≤ 3
Linear Rotation	Input conditions: Useful Domain (|Z| ≤ 2)	
	0 : 100% ND for X ≤ 0	0 : 48.2181% 1 : 50.7801% 2 : 0.9996% 3 : 0.0022% ND for Y ≤ 3
Hyperbolic Vectoring	Input conditions: Useful Domain (|Y| < |X|, X > 0, |atanh(Y / X)| ≤ 1.11rad)	
	0 : 98.1092% 1 : 1.8855% 2 : 0.0017% 3 : 0.0019% 4 : 0.0017% ND for X ≤ 4	0 : 47.7010% 1 : 51.2954% 2 : 0.9938% 3 : 0.0098% ND for Z ≤ 3
Hyperbolic Rotation	Input conditions: Useful Domain (|Z| ≤ 1.11rad, Y = 0)	
	0 : 47.6358% 1 : 51.1029% 2 : 1.2578% 3 : 0.0012% 4 : 0.0007% 5 : 0.0006% 6 : 0.0005% 7 : 0.0002% 8 : 0.0003% ND for X ≤ 8	0 : 47.5419% 1 : 51.2064% 2 : 1.2478% 3 : 0.0013% 4 : 0.0010% 5 : 0.0007% 6 : 0.0006% 7 : 0.0003% ND for Y ≤ 7

***Note**: The accuracy/deviation as stated above for each mode is not guaranteed for the final result of multi-step calculations, for example, if an operation involves two CORDIC calculations, the second calculation uses the result data from the first calculation (enabled with corresponding KEEP bit set). This is due to accumulated approximations and errors.*

7.3.6 Performance of CORDIC Coprocessor

The CORDIC calculation time increases linearly with increased precision. Increased precision is achieved with greater number of iterations, which requires increased width of the data parameters.

The CORDIC Coprocessor uses barrel shifters for data shifting. For a fixed number of 27 iterations per calculation, the total time from the start of calculation to the instant the EVFR.CDEOC flag is set is approximately 62 kernel clock cycles. It should be noted that the EVFR.CDERR flag is valid only one kernel clock cycle after the onset of EVFR.CDEOC. This timing for one complete calculation is applicable also to those modes which involve additional data processing, and also to the hyperbolic modes which involve repeat iterations and an extra cycle for mode setup.

***Note**: The above timing exclude time taken for software loading of initial data and reading of the final result data, to and from the six data registers.*

7.3.7 CORDIC Coprocessor Look-Up Tables

This section provides the user with need-to-know information regarding the CORDIC Coprocessor look-up-tables.

This section describes the look-up tables used in the CORDIC Coprocessor kernel.

7.3.7.1 Arctangent and Hyperbolic Arctangent Look-Up Tables

The LUTs are 29 bits and 30 bits wide respectively, for the arctangent table (atan LUT) and hyperbolic arctangent table (atanh LUT). Each entry of the atan LUT is divided into 1 sign bit (MSB) followed by 28-bit integer part. For the atanh LUT, each entry has 1 repeater bit (MSB), followed by 1 sign bit, then 28-bit integer part.

The contents of the LUTs are:

- atan LUT with data form of S29, see Table 35

Table 35 Precomputed Scaled Values for atan(2^{-i})

Iteration No.	Scaled atan(2^{-i}) in hex	Iteration No.	Scaled atan(2^{-i}) in hex
i = 0	4000000	i = 14	145F
i = 1	25C80A4	i = 15	A30
i = 2	13F670B	i = 16	518
i = 3	A2223B	i = 17	28C
i = 4	5161A8	i = 18	146
i = 5	28BAFC	i = 19	A3
i = 6	145EC4	i = 20	51
i = 7	A2F8B	i = 21	29
i = 8	517CA	i = 22	14

(table continues...)

Table 35 (continued) Precomputed Scaled Values for atan(2^{-i})

Iteration No.	Scaled atan(2^{-i}) in hex	Iteration No.	Scaled atan(2^{-i}) in hex
i = 9	28BE6	i = 23	A
i = 10	145F3	i = 24	5
i = 11	A2FA	i = 25	3
i = 12	517D	i = 26	1
I = 13	28BE		

- atanh LUT with data form of S29, see Table 36

Table 36 Precomputed Scaled Values for atanh(2-i)

Iteration No.	Scaled atanh(2^{-i}) in hex	Iteration No.	Scaled atanh(2^{-i}) in hex
i = 0	-	i = 14	145F
i = 1	2CC2F12	i = 15	A30
i = 2	14D01AC	i = 16	518
i = 3	A3D4E0	i = 17	28C
i = 4	5197FC	i = 18	146
i = 5	28C1C7	i = 19	A3
i = 6	145F9D	i = 20	51
i = 7	A2FA6	i = 21	29
i = 8	517CE	i = 22	14
i = 9	28BE6	i = 23	A
i = 10	145F3	i = 24	5
i = 11	A2FA	i = 25	3
i = 12	517D	i = 26	1
I = 13	28BE		

The Z data is a normalized representation of the actual angle. The internal scaling is such that $[-\pi,((2^{23}-1)/2^{23})\pi]$ is equivalent to $[-2^{28},(2^{28}-1)]$. The last 5 LSB bits are truncated, as 24-bit data is transferred to the data bus when addressed. From user's point, the angles $[-\pi,((2^{23}-1)/2^{23})\pi]$ are therefore represented by the range $[-2^{23},(2^{23}-1)]$.

7.3.7.2 Linear Function Emulated Look-Up Table

The emulated LUT for linear function is actually a shift register. The emulated LUT has 1 integer bit (MSB) followed by 23-bit fractional part of the form 1Q23.

In linear function, where Z is a real number, the internal Z data is of the form signed 4Q19. The externally read data has the last 5 bits of the fractional part truncated, resulting in a sign bit followed by 4-bit integer part, and finally 19-bit fractional part.

7.4 Global Functions

Since the DIV and CORDIC have their own set of control and data registers, they can work independently from one another.

However, they share a common bus interface and some global functions.

7.4.1 Result Chaining

The MATH Coprocessor supports result chaining between the DIV and CORDIC.

For the DIV, this means that each of the operand registers, DVD and DVS, can be updated with the value from any one of the result registers (QUOT and RMD in DIV; CORR[Z:X] in CORDIC).

For the CORDIC, this means that each of the operand registers, CORD[Z:X], can be updated with the value from either of the DIV result registers, QUOT and RMD.

In both cases, the selection is done with the operand register result chaining bit fields (xRC) in GLBCON register.

***Note**: To update the CORDIC CORD[Z:X] registers with the value from the corresponding CORR[Z:X] registers, a separate control based on the KEEP[Z:X] bits in STATC register, has to be used.*

The update is done once the preceding DIV or CORDIC operation is completed, together with the update to the actual result register. Figure 25 shows an example of chaining the DIV quotient result to the CORDIC X-operand.

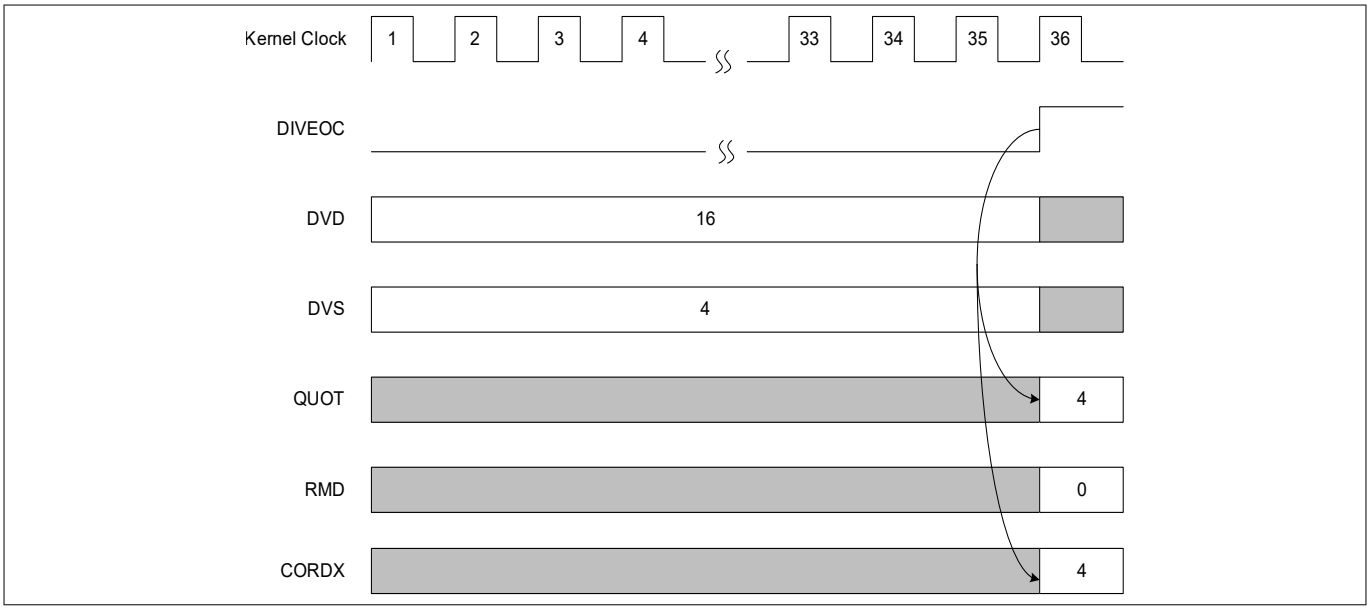


Figure 25 Division Operation with CORDXRC = 01_B

7.4.1.1 Result Chaining when Start Mode = 0

When the respective start mode is 0, a DIV and CORDIC operation can be started by having the application software write to the DVS and CORDX operand registers (see Start Mode Selection and Functional Overview). An update of these registers due to result chaining is equivalent to a hardware write and therefore, has the same effect.

To avoid any potential deadlock situation, application must take care to avoid the following two scenarios when using the result chaining feature:

1. DVS is chained to QUOT or RMD while DIVCON.STMODE = 0
2. DVS is chained to CORRx and CORDX is chained to QUOT or RMD, and at the same time both DIVCON.STMODE and CON.ST_MODE = 0

7.4.1.2 Handling Busy Flags when Result Chaining is Enabled

Using the example given in Figure 25, where a DIV result is chained to the CORDIC operand CORDX, if CON.ST_MODE = 0, starting the DIV calculation will set not just the DIV busy flag, but also that of the CORDIC. This is the case even though the CORDIC is initially not active, i.e. CORDIC calculation will start only after the DIV calculation is completed and either the QUOT or RMD value is written into CORDX. Similarly, the DIV, which becomes inactive after the DIV calculation is completed, does not clear its busy flag immediately. Instead, the busy flags of both DIV and CORDIC are cleared only after the CORDIC calculation is also completed.

The above applies also in the other direction, i.e. if a CORDIC result is chained to DIV DVS operand register and DIVCON.STMODE = 0.

While the busy flags are set:

- Reading the DIV or CORDIC result registers will cause wait states to be inserted onto the bus.
- While the DIV or CORDIC is active, starting a new DIV or CORDIC calculation or writing to their respective control registers DIVCON and CON will have no effect.
- However, if the DIV or CORDIC is still inactive or has just returned from an active state, starting a new DIV or CORDIC calculation or writing to their respective control registers will take effect.

A new calculation start or write to control register must be avoided while the DIV's or CORDIC's busy flag is set, otherwise calculation results might get corrupted.

7.5 Service Request Generation

If enabled by the respective interrupt enable bits in EVIER register, the DIV and CORDIC error and end of calculation events will trigger the interrupt service request to NVIC. The event is indicated by the event flag in EVFR register. The event flag can be cleared only by writing a 1 to the event flag clear bit in EVFCR register.

Writing a 1 to the event flag set bit in EVFSR register has the same effect of an end of calculation event.

Figure 26 shows the interrupt structure in the MATH Coprocessor.

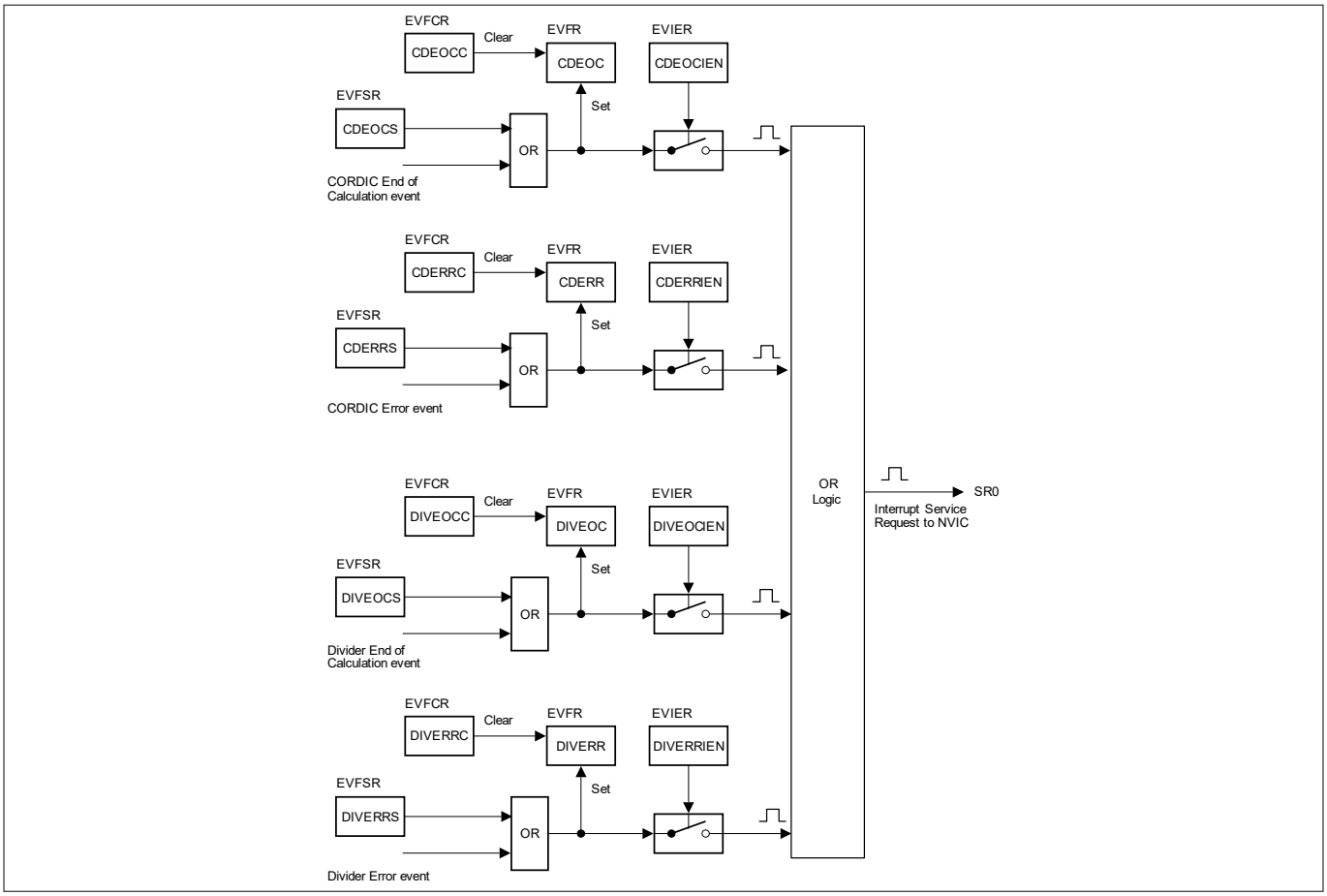


Figure 26 Interrupt Structure

7.6 Debug Behaviour

The MATH can be configured to enter a suspend mode when the program execution of the CPU is halted by the debugger (indicated by the assertion of the suspend signal).

Two suspend modes are supported and can be selected through the control bit field GLBCON.SUSCFG:

- Hard Suspend Mode
  - The kernel clock is immediately switched off, thereby stopping all calculations
- Soft Suspend Mode
  - Any active calculation is allowed to continue and only after it is completed, will the kernel clock be switched off

After the kernel clock is switched off, all registers become read-only. Writing to registers in this state has no effect. Suspend mode is exited and normal operations resumed when the suspend signal becomes deasserted.

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.

***Note**: In PSC1Mx, bit field GLBCON.SUSCFG is reset to its default value by any reset. If the suspend function is required during debugging, it is recommended that it is enabled in the user initialization code. Before programming the bit field, the interface clock has to be enabled and special care needs to be taken while enabling the interface clock as described in the CCU (Clock Gating Control) section of the SCU chapter.*

7.7 Power, Reset and Clock

The MATH Coprocessor is located in the core power domain. The module, including all registers, will be reset to its default state by a system reset.

The MATH Coprocessor requires two input clock signals, one for the kernel clock and one for the interface clock.The ratio of the kernel clock to the interface can be 2:1 or 1:1 but they must be at all times synchronous to each other.

Both clocks are disabled by default and can be enabled via the SCU_CGATCLR0 register. Enabling and disabling the two clocks could cause a load change and clock blanking could occur as described in the CCU (Clock Gating Control) section of the SCU chapter. It is strongly recommended to set up the two clocks in the user initialization code to avoid clock blanking during runtime.

7.8 Registers

Registers Overview

The absolute register address is calculated by adding:
Module Base Address + Offset Address

***Note**: The DIV registers shown in Table 38 supports both 16-bit and 32-bit bus access. All other registers (Global and CORDIC) supports only 32-bit access.*

Table 37 Registers Address Space

Module	Base Address	End Address	Note
MATH	$4003\ 0000_H$	$4003\ FFFF_H$	

Table 38 Register Overview

Register Name	Description	Offset Addr.	Access Mode	
			Read	Write
Global Registers				
Reserved	Reserved	0000_H	BE	BE
GLBCON	Global Control Register	0004_H	U, PV	U, PV
MATH_ID	Module Identification Register	0008_H	U, PV	BE
EVIER	Event Interrupt Enable Register	$000C_H$	U, PV	U, PV
EVFR	Event Flag Register	0010_H	U, PV	U, PV
EVFSR	Event Flag Set Register	0014_H	U, PV	U, PV
EVFCR	Event Flag Clear Register	0018_H	U, PV	U, PV
Reserved	Reserved	$001C_H$	BE	BE
Divider Registers				
DVD	Dividend Register	0020_H	U, PV	U, PV
DVS	Divisor Register	0024_H	U, PV	U, PV
QUOT	Quotient Register	0028_H	U, PV	BE
RMD	Dividend Register	$002C_H$	U, PV	BE
DIVST	Divider Status Register	0030_H	U, PV	BE

(table continues...)

Table 38 (continued) Register Overview

Register Name	Description	Offset Addr.	Access Mode	
			Read	Write
DIVCON	Divider Control Register	0034_H	U, PV	U, PV
Reserved	Reserved	0038_H - $003F_H$	BE	BE
CORDIC Registers				
STATC	Status and Data Control Register	0040_H	U, PV	U, PV
CON	Control Register	0044_H	U, PV	U, PV
CORDX	X Data Register	0048_H	U, PV	U, PV
CORDY	Y Data Register	$004C_H$	U, PV	U, PV
CORDZ	Z Data Register	0050_H	U, PV	U, PV
CORRX	X Result Register	0054_H	U, PV	BE
CORRY	Y Result Register	0058_H	U, PV	BE
CORRZ	Z Result Register	$005C_H$	U, PV	BE

7.9 Interconnects

Table 39 shows the module interconnects:

Table 39 Module Interconnects

Input/Output	I/O	Connected To	Description
SR0	O	NVIC	Interrupt service request output.

8 Memory Organization

This chapter provides description of the system memory organization, memory accesses, and memory protection strategy.

References

[5] Cortex®-M0 User Guide, Arm® DUI 0497A (ID112109)

8.1 Overview

The Memory Map in PSC1Mx is based on standard Arm® Cortex®-M0 system memory map.

8.1.1 Features

The Memory Map implements the following feature:

- Compatibility with standard Arm® Cortex®-M0 CPU [5]

8.1.2 Cortex®-M0 Address Space

The system memory map defines several regions. Address boundaries of each of the regions are determined by the Cortex®-M0 core architecture.

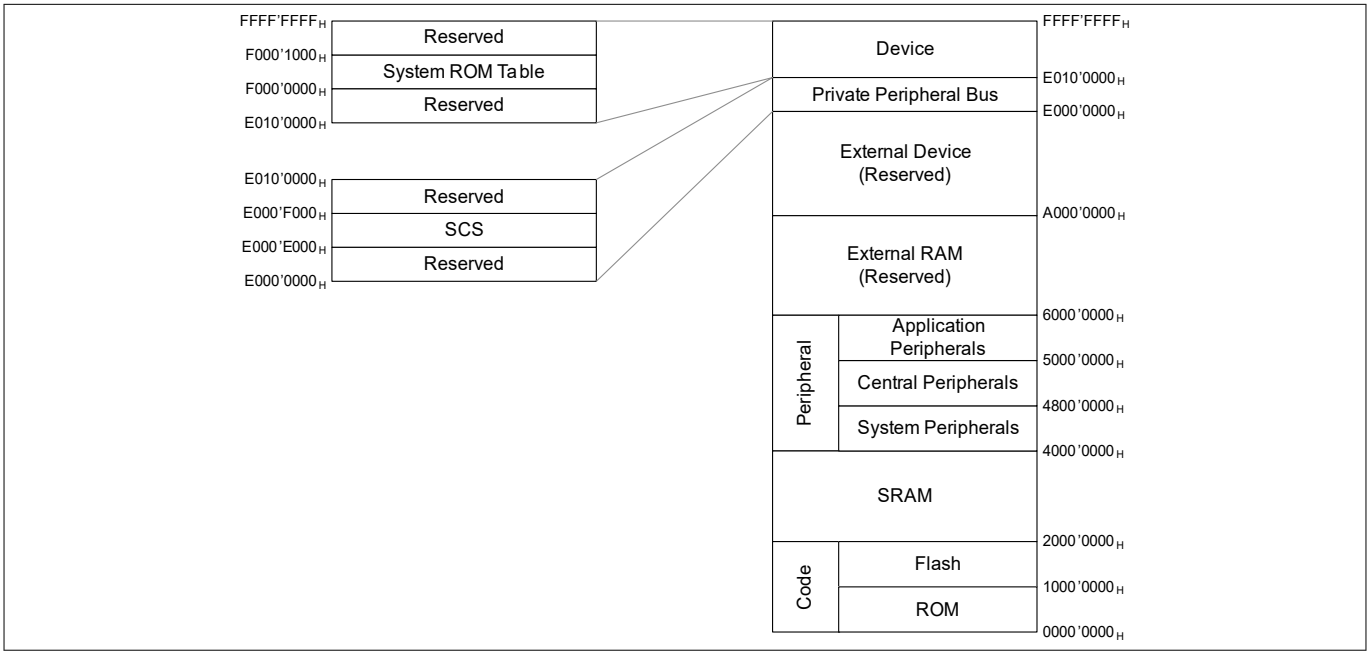


Figure 27 PSC1Mx Address Space

8.2 Memory Regions

The PSC1Mx device-specific address map contains on-chip memories and peripherals. The memory regions for PSC1Mx are described in Table 40.

Table 40 Memory Regions

Start (hex)	End (hex)	Size (hex)	Space name	Usage
00000000	1FFFFFFF	20000000	Code	ROM Firmware, Flash
20000000	3FFFFFFF	20000000	SRAM	Fast internal SRAM
40000000	47FFFFFF	08000000	Peripheral	System Peripherals group
48000000	4FFFFFFF	08000000	Peripheral	Central Peripherals group
50000000	57FFFFFF	08000000	Peripheral	Application Peripherals group
58000000	5FFFFFFF	08000000	Peripheral	reserved
60000000	9FFFFFFF	40000000	External SRAM	reserved
A0000000	DFFFFFFF	40000000	External Device	reserved
E0000000	E00FFFFF	00100000	Private Peripheral Bus	CPU
E0100000	EFFFFFFF	0FF00000	Vendor specific 1	reserved
F0000000	FFFFFFFF	10000000	Vendor specific 2	System ROM Table

8.3 Memory Map

Table 41 defines detailed system memory map of PSC1Mx where each individual peripheral or memory instance implement its own address spaces. For detailed register description of the system components and peripherals, refer to respective chapters of this document.

***Note**: Depending on the device variant, not all peripherals and memory address ranges may be available.*

Table 41 Memory Map

Address space	Address Range	Description	Access Type[1)]	
			Read	Write
Code	00000000_H - $00000CFF_H$	ROM (user-readable)	U, PV	nBE
	$00000D00_H$ - $00001FFF_H$	ROM (non-user-readable)	BE	BE
	00002000_H - $0FFFFFFF_H$	reserved	BE	BE
	10000000_H - $10000DFF_H$	Flash Sector 0 (non-user-readable)	nBE	nBE
	$10000E00_H$ - $10000FFF_H$	Flash Sector 0 (user-readable)	U, PV	nBE

(table continues...)

Table 41 (continued) Memory Map

Address space	Address Range	Description	Access Type[1]	
			Read	Write
Code (cont'd)	10001000_H - $10032FFF_H$	Flash (200 KB)	U, PV	U, PV
	10033000_H - $1FFFFFFF_H$	reserved	BE	BE
SRAM[2]	20000000_H - $20000FFF_H$	SRAM Block 0	U, PV	U, PV
	20001000_H - $20001FFF_H$	SRAM Block 1	U, PV	U, PV
	20002000_H - $20002FFF_H$	SRAM Block 2	U, PV	U, PV
	20003000_H - $20003FFF_H$	SRAM Block 3	U, PV	U, PV
	20004000_H - $3FFFFFFF_H$	reserved	BE	BE
System Peripherals	40000000_H - $400007FF_H$	Memory Control	U, PV	U, PV
	40000800_H - $4000FFFF_H$	reserved	BE	BE
	40010000_H - $40010FFF_H$	SCU (including RTC)	U, PV	U, PV
	40011000_H - $400110FF_H$	ANACTRL	U, PV	U, PV
	40011100_H - $4001FFFF_H$	reserved	BE	BE
	40020000_H - $4002001F_H$	WDT	U, PV	U, PV
	40020020_H - $4002FFFF_H$	reserved	BE	BE
	40030000_H - $4003003F_H$	MATH Global Registers and DIV	U, PV	U, PV
	40030040_H - $4003007F_H$	MATH CORDIC	U, PV	U, PV
	40030080_H - $4003FFFF_H$	reserved	BE	BE
	40040000_H - $4004007F_H$	Port 0	U, PV	U, PV
	40040080_H - $400400FF_H$	reserved	BE	BE
	40040100_H - $4004017F_H$	Port 1	U, PV	U, PV
	40040180_H - $400401FF_H$	reserved	BE	BE
	40040200_H - $4004027F_H$	Port 2	U, PV	U, PV
	40040280_H - $400402FF_H$	reserved	BE	BE
System Peripherals (cont'd)	40040300_H - $4004037F_H$	Port 3	U, PV	U, PV
	40040380_H - $400403FF_H$	reserved	BE	BE
	40040400_H - $4004047F_H$	Port 4	U, PV	U, PV
	40040480_H - $4004FFFF_H$	reserved	BE	BE
	40050000_H - $400500DF_H$	Flash Registers	U, PV	U, PV
	$400500E0_H$ - $47FFFFFF_H$	reserved	BE	BE
Central Peripherals	48000000_H - $480001FF_H$	USIC0 Channel 0	U, PV	U, PV
	48000200_H - $480003FF_H$	USIC0 Channel 1	U, PV	U, PV
	48000400_H - $480007FF_H$	USIC0 RAM	nBE	BE

(table continues...)

Table 41 (continued) Memory Map

Address space	Address Range	Description	Access Type[1)]	
			Read	Write
	48000800_H - $48003FFF_H$	reserved	BE	BE
	48004000_H - $480041FF_H$	USIC1 Channel 0	U, PV	U, PV
	48004200_H - $480043FF_H$	USIC1 Channel 1	U, PV	U, PV
	48004400_H - $480047FF_H$	USIC1 RAM	nBE	BE
	48004800_H - $4801FFFF_H$	reserved	BE	BE
	48020000_H - $4802000F_H$	PRNG	U, PV	U, PV
	48020010_H - $4802FFFF_H$	reserved	BE	BE
	48030000_H - $480303FF_H$	VADC0 General and Global Registers	U, PV	U, PV
	48030400_H - $480307FF_H$	VADC0 Group 0	U, PV	U, PV
	48030800_H - $48030BFF_H$	VADC0 Group 1	U, PV	U, PV
	$48030C00_H$ - $48033FFF_H$	reserved	BE	BE
	48034000_H - $480341FF_H$	SHS0	U, PV	U, PV
	48034200_H - $4803FFFF_H$	reserved	BE	BE
Central Peripherals (cont'd)	48040000_H - $480401FF_H$	CCU40 CC40 and Kernel Registers	U, PV	U, PV
	48040200_H - $480402FF_H$	CCU40 CC41	U, PV	U, PV
	48040300_H - $480403FF_H$	CCU40 CC42	U, PV	U, PV
	48040400_H - $480404FF_H$	CCU40 CC43	U, PV	U, PV
	48040500_H - $48043FFF_H$	reserved	BE	BE
	48044000_H - $480441FF_H$	CCU41 CC40 and Kernel Registers	U, PV	U, PV
	48044200_H - $480442FF_H$	CCU41 CC41	U, PV	U, PV
	48044300_H - $480443FF_H$	CCU41 CC42	U, PV	U, PV
	48044400_H - $480444FF_H$	CCU41 CC43	U, PV	U, PV
	48044500_H - $4FFFFFFF_H$	reserved	BE	BE
Application Peripherals	50000000_H - $500001FF_H$	CCU80 CC80 and Kernel Registers	U, PV	U, PV
	50000200_H - $500002FF_H$	CCU80 CC81	U, PV	U, PV
	50000300_H - $500003FF_H$	CCU80 CC81	U, PV	U, PV
	50000400_H - $500004FF_H$	CCU80 CC83	U, PV	U, PV
	50000500_H - $50003FFF_H$	reserved	BE	BE
	50004000_H - $500041FF_H$	CCU81 CC80 and Kernel Registers	U, PV	U, PV

(table continues...)

Table 41 (continued) Memory Map

Address space	Address Range	Description	Access Type[1]	
			Read	Write
	50004200_H - $500042FF_H$	CCU81 CC81	U, PV	U, PV
	50004300_H - $500043FF_H$	CCU81 CC81	U, PV	U, PV
	50004400_H - $500044FF_H$	CCU81 CC83	U, PV	U, PV
	50004500_H - $5000FFFF_H$	reserved	BE	BE
	50010000_H - $500101FF_H$	POSIF0	U, PV	U, PV
	50010200_H - $50013FFF_H$	reserved	BE	BE
	50014000_H - $500141FF_H$	POSIF1	U, PV	U, PV
	50014200_H - $5001FFFF_H$	reserved	BE	BE
Application Peripherals (cont'd)	50020000_H - $5002003F_H$	reserved	U, PV	U, PV
	50020040_H - $500203FF_H$	reserved	BE	BE
	50020400_H - $5002043F_H$	reserved	U, PV	U, PV
	50020440_H - $500207FF_H$	reserved	BE	BE
	50020800_H - $5002083F_H$	reserved	U, PV	U, PV
	50020840_H - $5002FFFF_H$	reserved	BE	BE
	50030000_H - $500301FF_H$	BCCU0	U, PV	U, PV
	50030200_H - $5003FFFF_H$	reserved	BE	BE
	50040000_H - $500402FF_H$	MultiCAN Node 0 and Global Registers	U, PV	U, PV
	50040300_H - $500403FF_H$	MultiCAN Node 1	U, PV	U, PV
	50040400_H - $50040FFF_H$	reserved	BE	BE
	50041000_H - $50043FFF_H$	MultiCAN Message Object Registers	U, PV	U, PV
	50044000_H - $5FFFFFFF_H$	reserved	BE	BE
External SRAM	60000000_H - $9FFFFFFF_H$	reserved	BE	BE
External Device	$A0000000_H$ - $DFFFFFFF_H$	reserved	BE	BE
Private Peripheral Bus	$E0000000_H$ - $E00FFFFF_H$	NVIC, System timer, System Control Block	U, PV	U, PV
Vendor specific 1	$E0100000_H$ - $EFFFFFFF_H$	reserved	BE	BE
Vendor specific 2	$F0000000_H$ - $F0000FFF_H$	System ROM Table	U, PV	nBE
	$F0001000_H$ - $FFFFFFFF_H$	reserved	BE	BE

1) For address ranges taken up by peripherals, the access type for each address in the range may differ from that shown in the table. Refer to respective chapters for details.
2) The address range $2000'0000_H$ to $2000'01FF_H$ will be overwritten by start-up software during device start-up.

8.4 Memory Access

This section describes the memory accesses to the different types of memories in the PSC1Mx.

8.4.1 Flash Memory Access

The PSC1Mx provides up to 200 KB of Flash memory for instruction code or constant data, starting at address $1000'1000_H$. This excludes Flash sector 0, which is used to store system information and is always read-only.

The Flash memory automatically inserts wait states during memory reads if needed.

For details of Flash memory access, refer to the Flash Architecture.

8.4.2 SRAM Access

The PSC1Mx provides 16 KB of SRAM for instruction code or constant data, as well as system variables such as the system stack, starting at address $2000'0000_H$.

The SRAM supports 8-bit, 16-bit, and 32-bit writes, and generates one parity bit for each 8 bits of written data. A read operation will check for parity errors on the 32-bit read data. Accesses to the SRAM require no wait states.

The 16 KB of SRAM is logically divided into four blocks of 4 KB each. Accesses to blocks 1, 2, and 3 can be disabled and re-enabled during run time with the peripheral privilege access scheme. See Peripheral Access Unit (PAU) for details.

***Note**: The address range $2000'0000_H$ to $2000'01FF_H$ will be overwritten by the start-up software during device start-up. Therefore, these addresses should not be targeted during the download of code or data by the bootstrap loader into the SRAM, nor should they store critical data that is still needed by the application after a system reset or SW master reset.*

8.4.3 ROM Access

The PSC1Mx provides 8 KB of ROM, which contains the startup software, vector table, and user routines.

Read accesses to the ROM require no wait states.

8.5 Memory Protection Strategy

Two aspects of memory protection are considered:

1. Intellectual Property (IP) Protection
2. Memory Access Protection during run time

The memory protection measures available in PSC1Mx are listed in Table 42.

Table 42 Memory Protection Measures

Protection Aspects	Protection Measures	Protection Target
IP protection	Blocking of unauthorized external access	Flash memory contents
Memory access protection	Bit protection scheme	Specific system-critical registers/bit fields
	Peripheral privilege access control	Specific address ranges; each range can be controlled independently

8.5.1 Intellectual Property (IP) Protection

IP protection refers to the prevention against unauthorized read out of critical data and user IP from Flash memory.

8.5.1.1 Blocking of Unauthorized External Access

In PSC1Mx, the Boot Mode Index (BMI) is used to control the boot options such that once the BMI is programmed to enter user mode (productive), it is not allowed to enter the other boot modes without an erase of the complete user Flash (including sector 0).

Therefore, boot options that load and execute external code, including unauthorized code that might read out the Flash memory contents, will be blocked and only user code originating from the Flash memory can be executed.

8.5.2 Memory Access Protection during Run time

Memory access protection refers to the prevention against unintended write access on a memory address space during run time.

8.5.2.1 Bit Protection Scheme

The bit protection scheme prevents direct software writing of selected register bits (protected bits) using the PASSWD register in the SCU module. When the bit field MODE is 11_B, writing 10011_B to the bit field PASS opens access to writing all protected bits, and writing 10101_B to the bit field PASS closes access to writing all protected bits. In both cases, the value of the bit field MODE is not changed, even if PASSWD register is written with 98_H or $A8_H$. It can only be changed when bit field PASS is written with 11000_B; for example, writing 0000'00C0$_H$ to PASSWD register disables the bit protection scheme.

Access is opened for maximum 32 MCLKs if the “close access” password is not written. If “open access” password is written again before the end of 32 MCLK cycles, there will be a recount of 32 MCLK cycles.

Table 43 shows the list of protected bit in PSC1Mx.

Table 43 List of Protected Register Bit Fields

Register	Bit Field
SCU_CLKCR, SCU_CLKCR1	FDIV, IDIV, PCLKSEL, RTCLKSEL, ADCCLKSEL, DCLKSEL
SCU_OSCCSR	DCO1PD
SCU_ANAOSCHPCTRL	All bits
SCU_ANASYNC1, SCU_ANASYNC2	All bits
SCU_ANAOFFSET	ADJL_OFFSET
SCU_CGATSET0	All bits
SCU_CGATCLR0	All bits
VADC0_ACCPROT0	All bits
VADC0_ACCPROT1	All bits

Writes to all protected registers except VADC0_ACCPROT[1:0], without opening access through bit field PASS, will be ignored. No bus error will be generated. User should read back the register value to ensure that the write has taken place.

Write to VADC0_ACCPROT[1:0], without opening access through bit field PASS, will trigger a hard fault. If an interrupt occurs immediately after the access is opened and the interrupt service routine requires more than 32

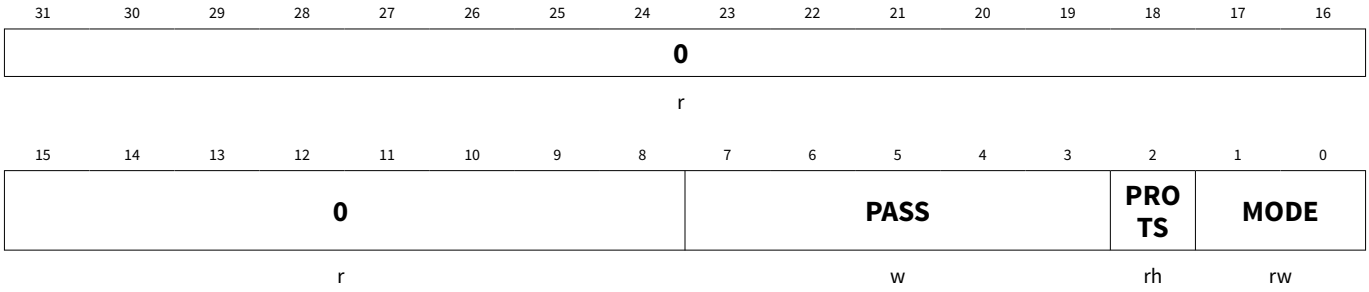
MCLK cycles, the write to the register will occur after the access is closed and thereby triggering a hard fault. To avoid this, interrupts should be disabled before writing to these two registers.

The PASSWD register is also described in the System Control Unit (SCU).

8.5.2.1.1 SCU_PASSWD

SCU_PASSWD **Address:** $4001\ 0024_H$

Password Register **Reset value:** $0000\ 0007_H$



Field	Bits	Type	Description
MODE	1:0	rw	**Bit Protection Scheme Control Bits** 00_B Scheme disabled - direct access to the protected bits is allowed. 11_B Scheme enabled - the bit field PASS has to be written with the passwords to open and close the access to the protected bits. (Default) Others: Scheme enabled, similar to the setting for MODE = 11_B. These two bits cannot be written directly. To change the value between 11_B and 00_B, the bit field PASS must be written with 11000_B. Only then will the MODE bit field be registered.
PROTS	2	rh	**Bit Protection Signal Status Bit** This bit shows the status of the protection. 0_B Software is able to write to all protected bits. 1_B Software is unable to write to any of the protected bits.
PASS	7:3	w	**Password Bits** This bit protection scheme only recognizes the following three passwords: 11000_B Enables writing of the bit field MODE. 10011_B Opens access to writing of all protected bits. 10101_B Closes access to writing of all protected bits.
0	31:8	r	**Reserved**

8.5.2.2 Peripheral Privilege Access Control

All CPU accesses are privileged accesses. In PSC1Mx, a separate scheme called Peripheral Privilege Access Control, which allows a peripheral's memory address space to be disabled to block any unintended write or read access, is provided. The address space can be re-enabled when necessary.

Refer to the PAU chapter for details.

8.6 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
- Data integrity Error (memories only)

Errors that cannot be indicated with bus errors get indicated with service requests that get propagated to the CPU as interrupts.

Unsupported Access Modes

Unsupported access modes can be classified in various ways and are usually specific to the module that access is performed to. Typical examples of unsupported access modes are write access to read-only type of address mapped resources, 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.

Data Integrity Error

Accesses to corrupted data in the memories result in either ECC (Error Correction Code) (ECC) or parity errors. The occurrence of such errors get signalized to the system through an SCU interrupt.

8.7 Debug Behaviour

The bus system in debug mode allows debug probe access to all system resources. No special handling of HALT mode is implemented and all interfaces respond with a valid bus response upon accesses.

8.8 Power, Reset and Clock

The bus system clocking scheme enables stable system operation and accesses to system resources for all valid system clock rates.

8.9 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 (for example, disable clock gating) before they can be accessed. For details please refer to individual peripheral chapters.

9 Prefetch Unit (PFU)

The purpose of the Prefetch unit is to reduce the Flash latency gap at higher system frequencies to increase the instruction per cycle performance.

9.1 Overview

The PFU consists mainly of:

- A one word prefetch buffer to store the prefetched Flash contents.
- A prefetch controller, which triggers a prefetch to the Flash if necessary and determines if the contents of the prefetch buffer can be used for a CPU instruction fetch.

9.1.1 Block Diagram

Figure 28 shows the block diagram of the PFU.

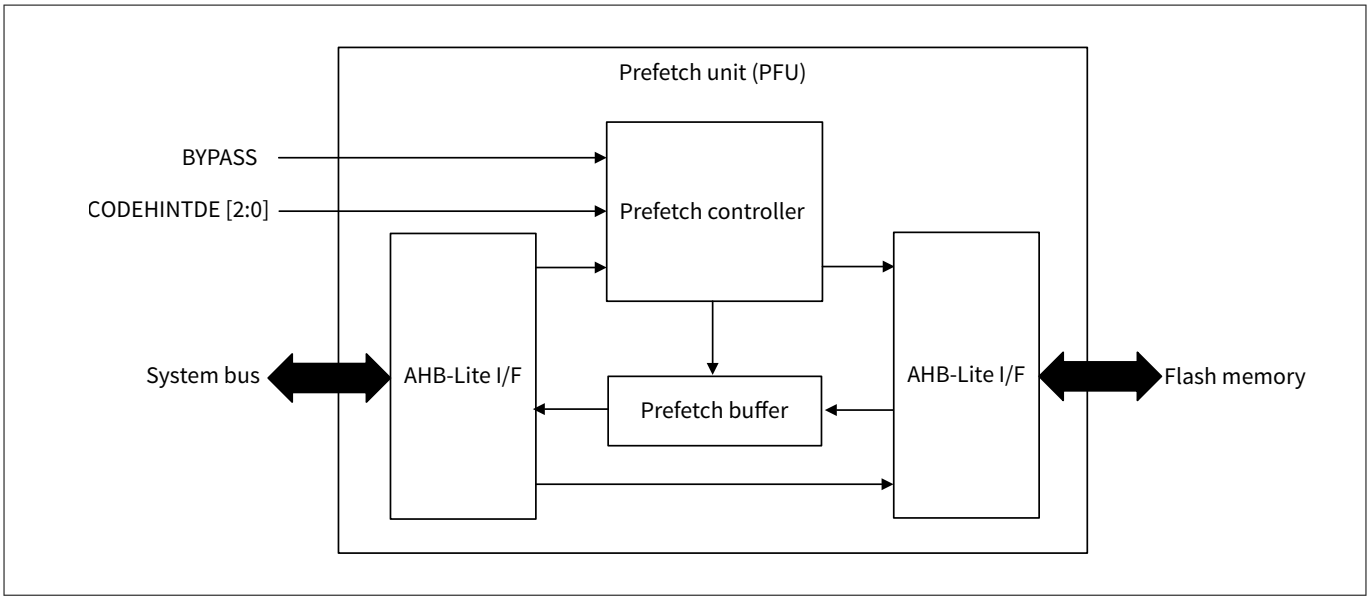


Figure 28 PFU Block Diagram

9.2 Operation Mode

The PFU is by default bypassed after a reset. The bit PFUBYP in SCU register PFUCR controls the bypass:

- Setting PFUBYP to '1' bypasses the PFU.
- Clearing PFUBYP to '0' removes the bypass and enables the PFU.

When the PFU is enabled, at the end of every instruction fetch by the CPU on the Flash, the PFU determines if a prefetch of the next address should be started based on the information from the CPU.

If yes, a read access to the next Flash address is immediately started, and the read data will be stored in the prefetch buffer when it becomes available.

Prefetch buffer hits are without any penalty, meaning single cycle access rate. This ensures a minimized latency.

***Note**: It is recommended to enable the PFU only when an application code consists of predominantly linear code. Otherwise, the performance penalty due to the execution of branch instructions may outweigh the performance gain of linear code execution.*

10 Flash Architecture

This chapter describes the non volatile memory (NVM) module.

10.1 Overview

The PSC1Mx has an embedded user-programmable NVM for storage of user code and data.

10.1.1 Features

The NVM has the following features:

- Reading by word, writing by block and erasing by page (256 B) or sector (4 KB).
- Fast personalization support.
- Automatic verify support.
- Up to 50,000 erase cycles per page.
- Minimum data retention of ten years in cells that were never previously programmed.
- Flash programming voltage generated on-chip.
- Configurable erase and write protection.
- Power saving sleep mode.
- Incremental write without erase for semaphores.

10.2 Definitions

Throughout the document the terms Non Volatile Memory (NVM) and Flash are used as synonyms, disregarding the fact that NVM describes a broader class of memories, where the described Flash is only a special case.

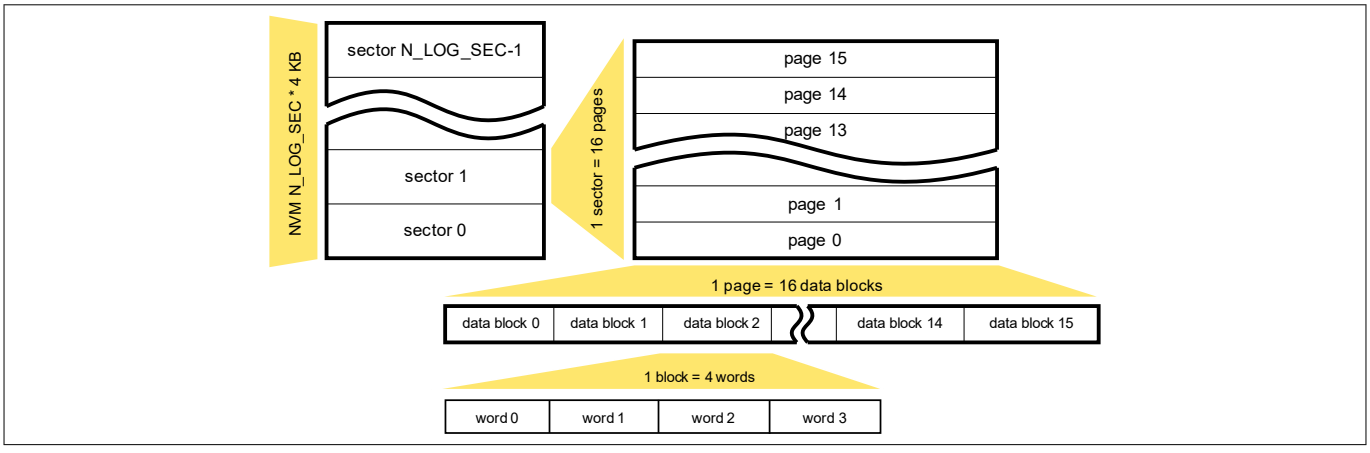


Figure 29 Logical structure of the NVM module

10.2.1 Logical and Physical States

Erasing

The erased state of a cell is '1'. Forcing an NVM cell to this state is called erasing. Erasing is possible with a granularity of a page.

Writing

The written state of a cell is '0'. Changing an erased cell to this state is called writing. Writing is possible with a granularity of a block.

Programming

The combination of erasing and writing is called programming. Programming often means also writing a previously erased page.

***Note**: The terms **write** and **writing** are also used for accessing special function registers. The meaning depends therefore on the context.*

10.2.2 Data Portions

Word

A word consists of 32-bits. A word represents the data size which is read from or written to the NVM module within one access cycle.

Block

A block consists of four words (128-bit data, extended by 4-bit parity, and 6-bit ECC). A block represents the smallest data portion that can be written.

Page

A page consists of 16 blocks.

Sector

A sector consists of 16 pages.

10.2.3 Address Types

Physical Address

Address of the CPU system.

Base Address or Memory Base Address

Physical address of the lower boundary for memory accesses of an NVM module.

Logical Address

Memory address offset inside the NVM module; If the memory is addressed, the logical address is the physical address subtracted by the base address of the module.

Sector Address

Module specific part of the logical address specifying the sector, for calculation see Memory Cell Array.

Page Address

Module specific part of the logical address, for calculation see Memory Cell Array.

10.2.4 Module-Specific Definitions

The following table defines NVM-specific constants used throughout this chapter.

Table 44 Module Specific Definitions

Module size [KB]	**204**	
Constant name	**Value**	**Comment**
N_BLOCKS	16	Number of blocks per page
N_PAGES	16	Number of pages per sector
N_LOG_SEC	51	Number of sectors ***Note**: For product variants with smaller Flash sizes, the number of sectors will be reduced accordingly.*

10.3 Module Components

10.3.1 Memory Cell Array

The non-volatile memory cells are organized in sectors, which consist of pages, which on their part are structured in blocks.

A logical memory address addr has the following structure:

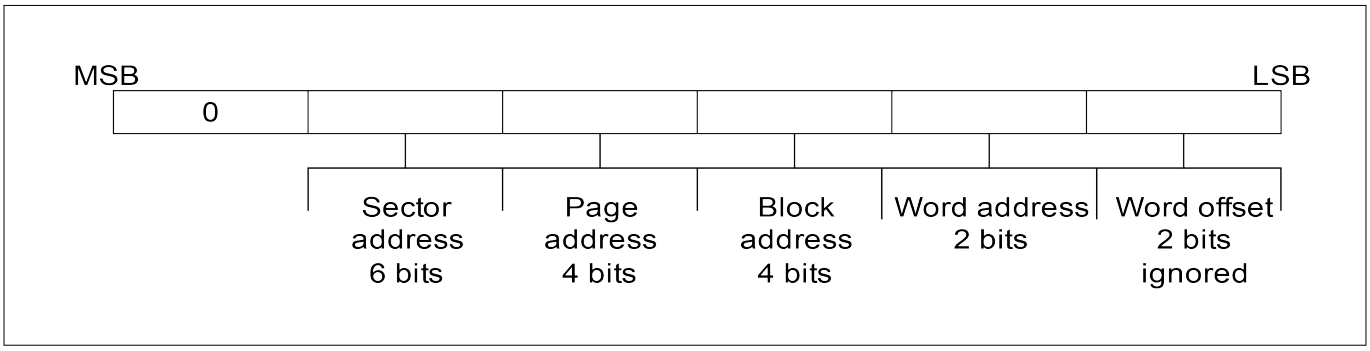


Figure 30 Decomposition of Logical Address

- addr[1:0] = word offset, for a memory address undefined and ignored
- addr[3:2] = word address
- addr[7:4] = block address
- addr[11:8] = page address
- addr[17:12] = sector address

The memory subsystem always accesses whole words, therefore the word offset is ignored by the NVM module.

10.3.1.1 Page

Each page consists of 16 data blocks of 138 bits each (including parity bits and ECC bits).

A page is the granularity of data that can be erased in the cell array.

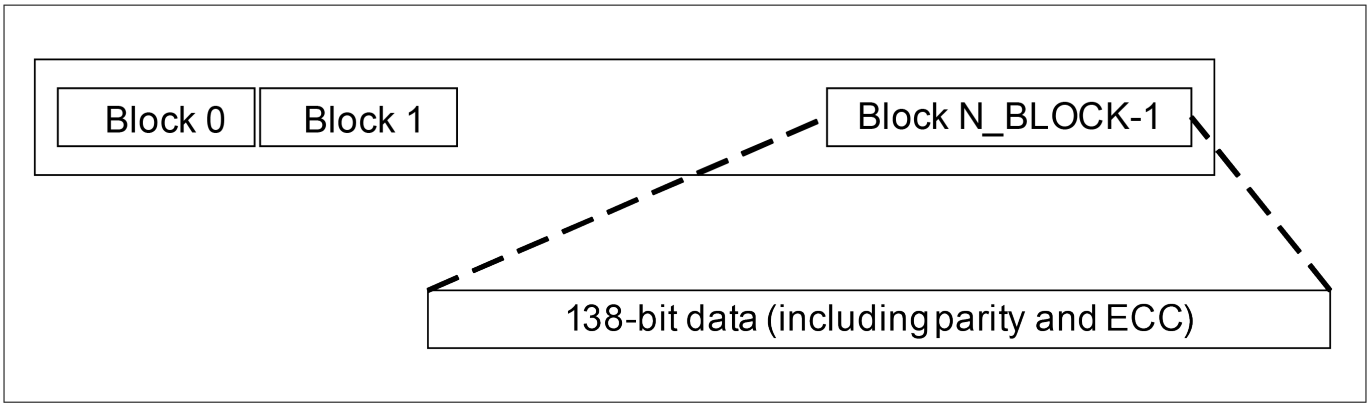


Figure 31 Structure of a Page

10.3.1.2 Sector

16 pages form a sector.

10.4 Functional Description

The NVM module supports read and write accesses to the memory and to the special function registers (SFRs). No read-modify-write mechanism is supported for SFRs.

The main tasks of the NVM module are reading and writing from/to the memory array.

10.4.1 SFR Accesses

Reading the special function registers is possible in every mode of the NVM module.

Register write is not possible while the state machine is busy. The write is stalled in this case. For other exceptions see NVM Sleep Mode.

10.4.2 Memory Read

The NVM memory can be read with a minimum granularity of a word provided that the word is in the memory address range of the module.

If the word is not within the memory address range of the NVM module, the module does not react at all and a different memory module may handle the access.

Memory read accesses are only possible while no FSM procedure (erase, write, verify, sleep, or wake-up) is in progress. A memory read access while the FSM is busy is stalled until the FSM is idle again. Then the access is carried out. Such a stall will also stop the CPU from executing code.

The NVM will insert waitstates during memory read automatically if needed.

10.4.3 Memory Write

From the user's viewpoint, data is written directly to the memory array. Internally, the module buffers one block and calculates the ECC bits. Then, a finite state machine is triggered to write both the ECC and data bits to the memory field.

Writing does not support wrap-around, that is writing of a block must always start with word 0 of the selected block and then automatically continues in ascending order up to word 3.

Since a block has to be written in four word writing steps, a special procedure has to be followed to transfer complete block data to the NVM; Writing of a block always has to start with word 0 of the addressed block. The following three writes need to address the other three words of the same block in ascending order. If this rule is not observed, the already provided words are discarded. If in this case the last provided word is addressing a word 0, this word is already accepted as the first word of a new block write procedure. Intermediate read operations do not influence the write data transfer. Intermediate read operations targeting the same address as the interrupted write operation are served from the memory array, i.e. do not read back the already transferred write data.

To write to the memory array, the NVM module has to be set to one-shot or continuous write mode by setting the SFR NVMPROG to the corresponding value. Data to be written can now be written to the desired address. It is not checked, if the addressed block was erased before. Writing to a non-erased block leads to corrupted data since writing can only clear bits but not set any bits. Reading such a block will lead most probably to an ECC fault.

A write access to the NVM when not in write mode does not trigger an exception or interrupt. The data is lost. Writes to the NVM are also used to trigger other operations which are described in the following subsections. Similarly, a write access to the NVM module has no effect and the data is lost when a protected sector is addressed (see Erase-Protection and Write-Protection).

Depending on the settings of NVMPROG, an automatic verify is performed (see Verify).

In the case of a one-shot write, the write mode is automatically exited. In the case of continuous write mode, further write operations to new addresses may follow until the write mode is explicitly exited. Continuous write operations may target blocks across the entire memory without any restriction regarding sector or page boundaries.

10.4.4 Memory Erase

Only whole pages can be physically erased, i.e. all bits of the addressed page are physically set to ‘1’.

To erase a page in the memory field, the NVM module has to be set to one-shot or continuous page erase mode by setting the SFR NVMPROG to the corresponding value. A write access to a memory location specifies the address of the page to be erased and triggers the erase.

A write access to the NVM when not in page erase mode does not trigger an exception or interrupt because they are also used to trigger other operations; as described in this section. Similarly, no erase is triggered when a protected sector is addressed (see Erase-Protection and Write-Protection).

In case of a one-shot page erase, page erase mode is automatically left. In case of continuous page erase mode, further page erase operations can follow, until the page erase mode is explicitly left.

10.4.5 Sector Erase

Additionally, whole sectors consisting of 16 pages can be physically erased in parallel, i.e. all bits of the addressed sector are physically set to ‘1’.

To erase a sector in the memory array, the NVM module has to be set to one-shot or continuous sector erase mode by setting the SFR NVMPROG to the corresponding value. Writing arbitrary data to a memory location specifies the address of the sector to be erased.

A write access to the NVM when not in sector erase mode does not trigger an exception or interrupt because it is also used to trigger other operations; as described in this section. Similarly, no erase is triggered when a protected sector is addressed (see Erase-Protection and Write-Protection).

In case of a one-shot sector erase, sector erase mode is automatically left. In case of continuous sector erase mode, further sector erase operations can follow, until the sector erase mode is explicitly left.

10.4.6 Verify

The data written as described in Memory Write can be verified automatically. The written data in the cell array can be automatically compared to the data still available in the module internal buffer. This is automatically performed two times with hardread written and hardread erased. These hardread levels provide some margin compared to the normal read level to ensure that the data is really programmed with suitably distinct levels for written and erased bits. The verification result can be read at SFR NVMSTATUS.

Stand-alone verify operations can also be started by setting the SFR NVMPROG to the corresponding value. In this case, data written to a memory location is compared with the content of the memory field at the specified address. The comparison here is performed just once with a read level chosen before from normal read, hardread written and hardread erased.

A write access to the NVM when not in verification mode does not trigger an exception or interrupt because it is also used to trigger other operations; as described in this section. Similarly, a write access to the NVM module has no effect, when a protected sector is addressed (see Erase-Protection and Write-Protection).

In case of a one-shot verify, verify mode is automatically left. In case of continuous verify mode, further verify operations can follow, until the verify mode is explicitly left.

***Note**: A continuous write operation without automatic verification, followed by two stand-alone verifications with 'hardread written' and 'hardread erased', is faster than a write operation with continuous automatic verification, since in the second case for every block write the hardread level has to be changed twice, whereas in the first case this change is performed only two times for the complete write data.*

On the other hand, for the continuous automatic verification the reference data for the verification is directly available within the NVM module, whereas for the stand-alone verification the reference data needs to be provided again by the CPU.

10.4.7 Erase-Protection and Write-Protection

By setting NVMCONF.SECPROT, a configurable number of sectors can be selected to be protected from any modification, that is a range of sectors starting with sector 0 can be defined to be erase-protected and write-protected.

10.5 Properties and Implementation of Error Correcting Code (ECC)

The error correcting code (ECC) for the data blocks is a one-bit error correction and a (partial) double-bit error detection for every data block of 128 bit, which uses 4 parity bits and 6 ECC bits in addition to the protected 128 data bits. The correct ECC bits for every block are generated automatically when the data is written.

10.6 Service Request Generation

The NVM module supports immediate error and status information to the user by interrupt generation.

A NVM interrupt can be issued due to the following events:

- Completion of:
  - NVM operations started through NVMPROG.ACTION (except for verify-only sequence)
  - NVM wake-up sequence
- NVM ECC double-bit error

The NVM interrupt request is generated to the CPU through the SCU. Therefore, related interrupt control bits are located in the SCU but with the following exceptions:

- There are two sets of NVM ECC double-bit error status flag and clear status bit:
  - NVMSTATUS.ECC2READ and NVMPROG.RSTECC in NVM module
  - SCU_RAWSR.FLECC2I and SCU_SRCLR.FLECC2I in SCU module
- It is sufficient to use just one of the two sets and ignore the other.
- The interrupt enable bit for the NVM operation complete interrupt is located only in the NVMCONF register.

10.7 Power, Reset and Clock

The following sections describe the power, reset and clock sources of the NVM module.

10.7.1 Power Supply

The NVM module sources all voltages required for read, program and erase operations from the on-chip Embedded Voltage Regulator (EVR).

The standard V_{DDC} is used for all digital control functions.

10.7.2 Power Saving Modes

The NVM module supports the following power saving modes. The modes differ in power consumption and in the time to restart the memory.

10.7.2.1 NVM Idle Mode

Idle mode is entered after reset after charging the pumps. In idle mode the power consumption is less than during a memory read access or write access.

Any operation of the NVM module (SFR access or memory access) can be started from idle mode without time delay.

10.7.2.2 NVM Sleep Mode

Entering and leaving NVM sleep mode requires special state machine sequences, which require some time. Therefore, besides SFR accesses, the NVM module cannot be used immediately after wake-up from sleep mode, as indicated by NVMSTATUS.BUSY.

NVM sleep mode is entered via WFE/WFI when Flash Power down is activated, see System Control Unit (SCU). Alternatively, sleep mode for the NVM module can be triggered by writing the respective bit in SFR NVMCONF.

In NVM sleep mode, only the SFR NVMCONF can be read and written; all other SFRs can only be read. Memory access is not possible in sleep mode.

10.7.3 Reset

The NVM module is reset with the system reset.

10.7.4 Clock

The NVM module is operating at the same clock speed as main clock, MCLK.

10.8 Registers

Table 46 shows a complete list of the NVM special function registers.

Table 45 Registers Address Space

Module	Base Address	End Address	Note
NVM	4005 0000$_H$	4005 00FF$_H$	

Table 46 Registers Overview

Register Name	Register Long Name	Offset Address
NVMSTATUS	NVM Status Register	0000$_H$
NVMPROG	NVM Programming Control Register	0004$_H$
NVMCONF	NVM Configuration Register	0008$_H$

The register is addressed wordwise.

Reading an SFR is not blocked, while the NVM module is busy. If the SFR value depends on the completion of an NVM sequence, NVMSTATUS.BUSY must be polled for 0B, before the SFR is read. Otherwise, reading the SFR might yield a value that is not updated yet.

10.9 Example Sequences

This section presents some low-level programming examples.

In all following operations, the protection defined by NVMCONF.SECPROT needs to be taken into account.

10.9.1 Writing to Memory

10.9.1.1 Writing a Single Block

This sequence requires that the target block is already erased.

Additional assumption: NVMPROG.ACTION = 00$_H$.

1. Start a one-shot write operation:
   Write NVMPROG.ACTION = 51$_H$ or 91$_H$, respectively, if an automatic verification of the written data is to be performed or not.
2. Write data for one block to the physical address.
3. Poll flag NVMSTATUS.BUSY until write sequence has finished, or wait for NVMready interrupt (if enabled).
4. If an automatic verification of the written data was requested in the first step, read NVMSTATUS.VERR for the verification result.

If no verification of the written data was requested in step 1, and thus no read access to the NVM SFR is required in step 4, step 3 may be omitted.

The next access to the NVM module or the next write access to an NVM SFR (which ever comes first) will be automatically stalled, until the operation started in step 2 is finished.

10.9.1.2 Writing Blocks

This sequence requires that the target blocks are already erased.

Additional assumption: NVMPROG.ACTION = 00_H.

1. Start a continuous write operation:

   Write NVMPROG.ACTION = 61_H or $A1_H$, respectively, if an automatic verification of the written data is to be performed or not.
2. Write data for one block to the physical address.
3. Poll flag NVMSTATUS.BUSY until write sequence has finished, or wait for NVMready interrupt (if enabled).

   Optionally, step 3 can be omitted; The next access to the NVM module or the next write access to an NVM SFR (which ever comes first) will be automatically stalled, until the operation started in step 2 is finished.
4. Jump to step 2 unless all data is written.
5. Stop continuous write operation:

   Write NVMPROG.ACTION = 00_H.
6. If an automatic verification of the written data was requested in the first step, read NVMSTATUS.VERR for the verification result.

10.9.2 Erasing Memory

10.9.2.1 Erasing a Single Page

Assumption: NVMPROG.ACTION = 00_H.

1. Start a one-shot page erase operation:

   Write NVMPROG.ACTION = 92_H.
2. Write dummy data for one word to one arbitrary word-aligned physical address of the page to be erased.
3. Poll flag NVMSTATUS.BUSY until page erase sequence has finished, or wait for NVMready interrupt (if enabled).

Optionally, step 3 can be omitted; The next access to the NVM module or the next write access to an NVM SFR (which ever comes first) will be automatically stalled, until the operation started in step 2 is finished.

10.9.2.2 Erasing Pages

Assumption: NVMPROG.ACTION = 00_H.

1. Start a continuous page erase operation:

   Write NVMPROG.ACTION = $A2_H$.
2. Write dummy data for one word to one arbitrary word-aligned physical address of the page to be erased.
3. Poll flag NVMSTATUS.BUSY until page erase sequence has finished, or wait for NVMready interrupt (if enabled).

   Optionally, step 3 can be omitted; The next access to the NVM module or the next write access to an NVM SFR (which ever comes first) will be automatically stalled, until the operation started in step 2 is finished.
4. Jump to step 2 unless all pages are erased.
5. Stop continuous page erase operation:

   Write NVMPROG.ACTION = 00_H.

10.9.2.3 Erasing a Single Sector

Assumption: NVMPROG.ACTION = 00_H.

1. Start a one-shot sector erase operation:

   Write NVMPROG.ACTION = 94_H.
2. Write dummy data for one word to one arbitrary word-aligned physical address of the sector to be erased.
3. Poll flag NVMSTATUS.BUSY until sector erase sequence has finished, or wait for NVMready interrupt (if enabled).

   Optionally, step 3 can be omitted; The next access to the NVM module or the next write access to an NVM SFR (which ever comes first) will be automatically stalled, until the operation started in step 2 is finished.

10.9.2.4 Erasing Sectors

Assumption: NVMPROG.ACTION = 00_H.

1. Start a continuous sector erase operation:

   Write NVMPROG.ACTION = $A4_H$.
2. Write dummy data for one word to one arbitrary word-aligned physical address of the sector to be erased.
3. Poll flag NVMSTATUS.BUSY until sector erase sequence has finished, or wait for NVMready interrupt (if enabled).

   Optionally, step 3 can be omitted; The next access to the NVM module or the next write access to an NVM SFR (which ever comes first) will be automatically stalled, until the operation started in step 2 is finished.
4. Jump to step 2 unless all sectors are erased.
5. Stop continuous sector erase operation:

   Write NVMPROG.ACTION = 00_H.

10.9.3 Verifying Memory

10.9.3.1 Verifying a Single Block

Assumption: NVMPROG.ACTION = 00_H.

1. Choose desired hardread level by setting NVMCONF.HRLEV.
2. Start a one-shot verify operation:

   Write NVMPROG.ACTION = $D0_H$.
3. Write reference data for one block to the physical address.
4. Poll flag NVMSTATUS.BUSY until verify sequence has finished.
5. Read NVMSTATUS.VERR for the verification result.

10.9.3.2 Verifying Blocks

Assumption: NVMPROG.ACTION = 00_H.

1. Choose desired hardread level by setting NVMCONF.HRLEV.
2. Start a continuous verify operation:
   Write NVMPROG.ACTION = $E0_H$.
3. Write reference data for one block to the physical address.
4. Poll flag NVMSTATUS.BUSY until verify sequence has finished.
   Optionally, step 4 can be omitted; The next access to the NVM module or the next write access to an NVM SFR (which ever comes first) will be automatically stalled, until the operation started in step 3 is finished.
5. Jump to step 3 unless all blocks are verified.
6. Stop continuous verify operation:
   Write NVMPROG.ACTION = 00_H.
7. Read NVMSTATUS.VERR for the verification result.

10.9.4 Writing to an Already Written Block

Normally, writing additional bits in an already written block is not feasible, since the already written ECC bits and the parity bits would need an update, too, which in most cases would require to erase some bits, which is not possible. The result would be an ECC-faulty block.

For special purposes a repeated update of specially constructed data is possible, for example, to store some marker bits for use in a power loss recovery SW implementation.

Starting with an erased block, and making use of the special structure of the ECC, it is possible to repeatedly add two newly written bits in identical bit positions to two arbitrary words of the block. This principle is shown in the exemplary writing scheme of Table 47, where a block is updated 32 times without corruption of the ECC, that is, the data stays ECC protected throughout the procedure.

Table 47 Incremental Update of a Block with Specially Constructed Data

Operation	Block Write Data	Resulting Block Content[1)]
Erase block		FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
Add 1st value	FFFFFFFF FFFFFFFC FFFFFFFF FFFFFFFC	FFFFFFFF FFFFFFFC FFFFFFFF FFFFFFFC
Add 2nd value	FFFFFFFF FFFFFFF3 FFFFFFFF FFFFFFF3	FFFFFFFF FFFFFFF0 FFFFFFFF FFFFFFF0
Add 3rd value	FFFFFFFF FFFFFFCF FFFFFFFF FFFFFFCF	FFFFFFFF FFFFFFC0 FFFFFFFF FFFFFFC0
Add 4th value	FFFFFFFF FFFFFF3F FFFFFFFF FFFFFF3F	FFFFFFFF FFFFFF00 FFFFFFFF FFFFFF00
Add 5th value	FFFFFFFF FFFFFCFF FFFFFFFF FFFFFCFF	FFFFFFFF FFFFFC00 FFFFFFFF FFFFFC00
Add 6th value	FFFFFFFF FFFFF3FF FFFFFFFF FFFFF3FF	FFFFFFFF FFFFF000 FFFFFFFF FFFFF000
...	...	...
15th value	FFFFFFFF CFFFFFFF FFFFFFFF CFFFFFFF	FFFFFFFF C0000000 FFFFFFFF C0000000
16th value	FFFFFFFF 3FFFFFFF FFFFFFFF 3FFFFFFF	FFFFFFFF 00000000 FFFFFFFF 00000000
17th value	FFFFFFFC FFFFFFFF FFFFFFFC FFFFFFFF	FFFFFFFC 00000000 FFFFFFFC 00000000
...	...	...
30th value	F3FFFFFF FFFFFFFF F3FFFFFF FFFFFFFF	F0000000 00000000 F0000000 00000000

(table continues...)

Table 47 (continued) Incremental Update of a Block with Specially Constructed Data

Operation	Block Write Data	Resulting Block Content[1)]
31st value	CFFFFFFF FFFFFFFF CFFFFFFF FFFFFFFF	C0000000 00000000 C0000000 00000000
32nd value	3FFFFFFF FFFFFFFF 3FFFFFFF FFFFFFFF	00000000 00000000 00000000 00000000
Add to invalidate written values[2)]	FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE	Same as before, but ECC2-faulty

1) ECC-clean and parity-clean (ECC bits and parity bits all stay erased), except where indicated otherwise.
2) This write data can be written on any of the states above (except the completely erased state), always resulting in an unchanged data content, but now with an ECC2 error (three ECC bits and one parity bit are written).

Other combinations of write values are possible, too, as long as the basic “add two identical bits in each of two words” is adhered to. In Table 47 it is also shown that an invalidation of the data by deliberately creating an ECC2 error is possible.

To minimize the write times and also to minimize the cycle load of the block, it is important to only write the new bits of the block as shown in Table 47. In principle it is possible to repeatedly write also the already written bits again, but this would unnecessarily increase the writing times and would also increase the cycle load.

10.9.5 Sleep Mode

Assumption: Active mode (NVMSTATUS.SLEEP = 0_B) and NVMSTATUS.BUSY = 0_B.

To Enter Sleep Mode

1. Execute WFE/WFI or NVMCONF.NVM_ON = 0_B.
2. NVMSTATUS.BUSY = 1_B and NVMSTATUS.SLEEP = 1_B until sleep mode is reached.
3. NVMSTATUS.BUSY = 0_B and NVMSTATUS.SLEEP = 1_B while in sleep mode.

To Wake-up from Sleep Mode

1. Any wake-up event and NVMCONF.NVM_ON = 1_B.
2. NVMSTATUS.BUSY = 1_B and NVMSTATUS.SLEEP = 1_B until active mode is reached.
3. Poll flag NVMSTATUS.BUSY until wake-up sequence has finished, or wait for NVMready interrupt (if enabled).
4. NVMSTATUS.BUSY = 0_B and NVMSTATUS.SLEEP = 0_B while in active mode.

A wake-up event while the goto sleep sequence has not finished yet, does not shorten this sequence, but only directly starts the wake-up sequence afterwards.

10.9.6 Timing

This state diagram shows the transitions and timings of all possible sequences.

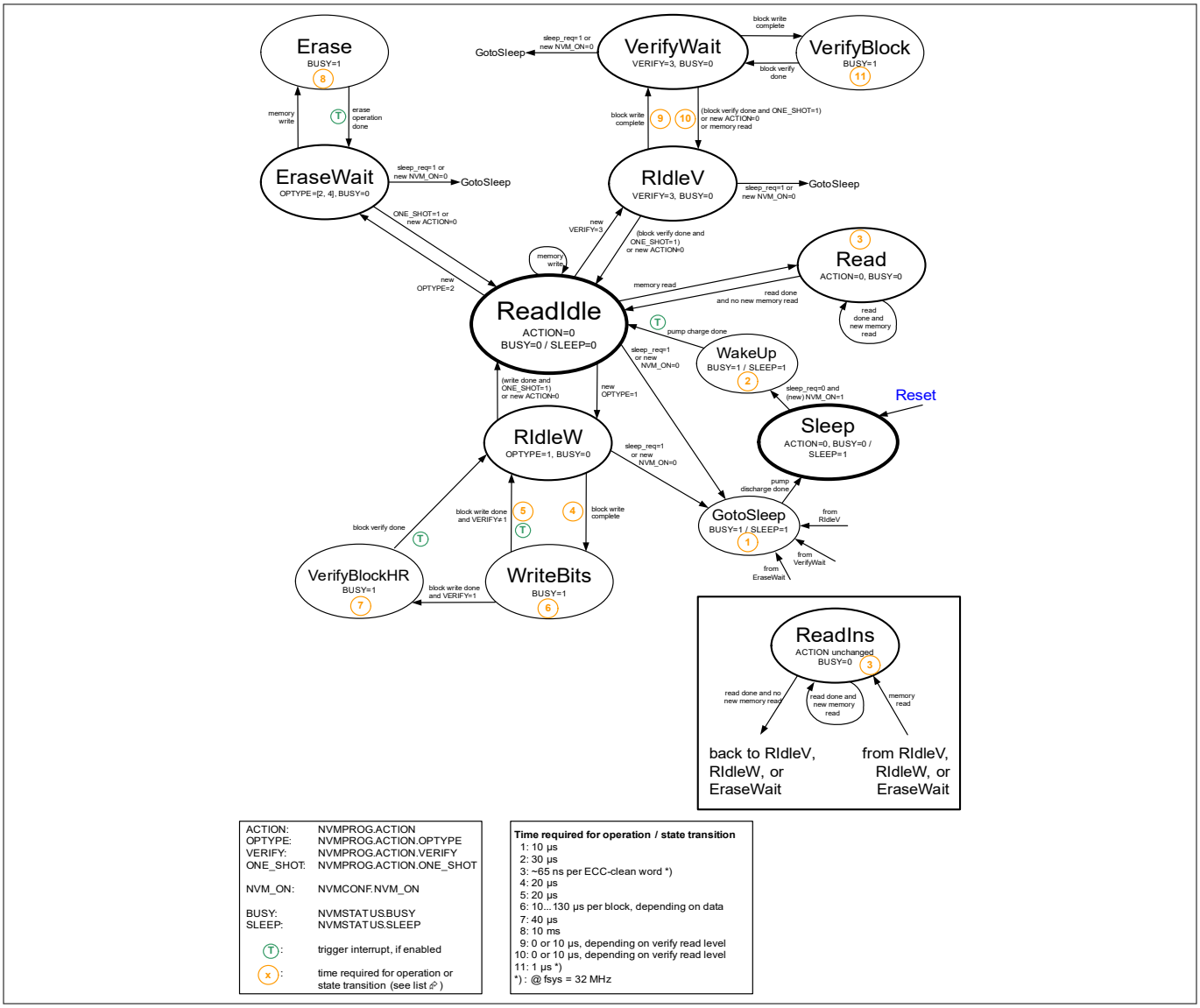


Figure 32 State Diagram of the NVM Module Timings are preliminary

11 Peripheral Access Unit (PAU)

This chapter describes Peripheral Access Unit (PAU) in PSC1Mx.

11.1 Overview

The PAU supports access control of memories and peripherals in a central place.

11.1.1 Features

The PAU provides the following features:

- Allows user application to enable/disable the access to the registers of a peripheral
- Generates a HardFault exception when there is an access to a disabled or unassigned address location
- Provides information on availability of peripherals and size of memories

11.2 Peripheral Privilege Access Control

The user application can use the Peripheral Privilege Access Registers, PRIVDISn, to disable access to a peripheral. When the PDISx bit corresponding to the peripheral is set, the memory address space mapped to the peripheral is rendered invalid. An access to such an invalid address causes a HardFault exception.

The application can clear the same bit to enable accesses to the peripheral again.

Table 48 shows the peripherals and their assigned PDISx bits. Peripherals without a PDISx bit are accessible at all times.

Table 48 Peripherals Availability and Privilege Access Control

Peripheral	Address Grouping	AVAILn.AVAILx bit	PRIVDIS.PDISx bit
Flash	Flash SFRs	-	PRIVDIS0.2
SRAM	RAM Block 1	AVAIL0.5	PRIVDIS0.5
	RAM Block 2	AVAIL0.6	PRIVDIS0.6
	RAM Block 3	AVAIL0.7	PRIVDIS0.7
WDT	WDT	-	PRIVDIS0.19
MATH	MATH Global SFRs and DIV	AVAIL0.20	PRIVDIS0.20
	MATH CORDIC	AVAIL0.21	PRIVDIS0.21
Ports	Port 0	AVAIL0.22	PRIVDIS0.22
	Port 1	AVAIL0.23	PRIVDIS0.23
	Port 2	AVAIL0.24	PRIVDIS0.24
	Port 3	AVAIL0.25	PRIVDIS0.25
	Port 4	AVAIL0.26	PRIVDIS0.26
USIC0	USIC0_CH0	AVAIL1.0	PRIVDIS1.0
	USIC0_CH1	AVAIL1.1	PRIVDIS1.1
PRNG	PRNG	AVAIL1.4	-
VADC0	VADC0 Basic SFRs	AVAIL1.5	PRIVDIS1.5
	VADC0 Group 0 SFRs	AVAIL1.6	PRIVDIS1.6

(table continues...)

Table 48 (continued) Peripherals Availability and Privilege Access Control

Peripheral	Address Grouping	AVAILn.AVAILx bit	PRIVDIS.PDISx bit
	VADC0 Group 1 SFRs	AVAIL1.7	PRIVDIS1.7
SHS0	SHS0	AVAIL1.8	PRIVDIS1.8
CCU40	CCU40_CC40 and CCU40 Kernel SFRs	AVAIL1.9	PRIVDIS1.9
	CCU40_CC41	AVAIL1.10	PRIVDIS1.10
	CCU40_CC42	AVAIL1.11	PRIVDIS1.11
	CCU40_CC43	AVAIL1.12	PRIVDIS1.12
USIC1	USIC1_CH0	AVAIL1.16	PRIVDIS1.16
	USIC1_CH1	AVAIL1.17	PRIVDIS1.17
CCU41	CCU41_CC40 and CCU41 Kernel SFRs	AVAIL1.25	PRIVDIS1.25
	CCU41_CC41	AVAIL1.26	PRIVDIS1.26
	CCU41_CC42	AVAIL1.27	PRIVDIS1.27
	CCU41_CC43	AVAIL1.28	PRIVDIS1.28
CCU80	CCU80_CC80 and CCU80 Kernel SFRs	AVAIL2.0	PRIVDIS2.0
	CCU80_CC81	AVAIL2.1	PRIVDIS2.1
	CCU80_CC82	AVAIL2.2	PRIVDIS2.2
	CCU80_CC83	AVAIL2.3	PRIVDIS2.3
POSIF0	POSIF0	AVAIL2.12	PRIVDIS2.12
BCCU0	BCCU0	AVAIL2.15	PRIVDIS2.15
CCU81	CCU81_CC81 and CCU80 Kernel SFRs	AVAIL2.16	PRIVDIS2.16
	CCU81_CC81	AVAIL2.17	PRIVDIS2.17
	CCU81_CC82	AVAIL2.18	PRIVDIS2.18
	CCU81_CC83	AVAIL2.19	PRIVDIS2.19
CAN0	MultiCAN Global and Node 0 SFRs	AVAIL2.20	PRIVDIS2.20
	MultiCAN Node 1 SFRs	AVAIL2.21	PRIVDIS2.21
	MultiCAN Message Objects	AVAIL2.23	PRIVDIS2.23
POSIF1	POSIF1	AVAIL2.28	PRIVDIS2.28

11.3 Peripheral Availability and Memory Size

The availability of peripherals and memory sizes varies according to product variants. The user application can read the Peripheral Availability Registers, AVAILn, to check for the availability of peripherals in a product variant. Refer to Table 48 for the bit assignment. Similarly, the Memory Size Registers (such as, FLSIZE for Flash memory) can be used to check for the size of the available memories.

11.4 Service Request Generation

The PAU generates a hard fault exception when there is an access to an invalid address.

11.5 Debug Behaviour

The PAU does not support a suspend mode while the system is halted by the debugger. That means that the PAU continues its operation during debug halt.

11.6 Power, Reset and Clock

The PAU is located in the core power domain. The module can be reset to its default state by a system reset.

The PAU module is clocked by the main clock, MCLK, from SCU.

11.7 Initialization and System Dependencies

The PAU is always available after reset.

11.8 PAU Registers

Registers Overview

The absolute register address is calculated by adding:

Module Base Address + Offset Address

Table 49 Registers Address Space

Module	Base Address	End Address	Note
PAU	4000 0000$_H$	4000 FFFF$_H$	

Table 50 Register Overview

Register Name	Description	Offset Addr.	Access Mode	
			Read	Write
Reserved	Reserved	0000$_H$ - 003C$_H$	nBE	nBE
AVAIL0	Peripheral Availability Register 0	0040$_H$	U, PV	BE
AVAIL1	Peripheral Availability Register 1	0044$_H$	U, PV	BE
AVAIL2	Peripheral Availability Register 2	0048$_H$	U, PV	BE
Reserved	Reserved	004C$_H$ - 007C$_H$	nBE	nBE
PRIVDIS0	Peripheral Privilege Access Register 0	0080$_H$	U, PV	U, PV
PRIVDIS1	Peripheral Privilege Access Register 1	0084$_H$	U, PV	U, PV
PRIVDIS2	Peripheral Privilege Access Register 2	0088$_H$	U, PV	U, PV
Reserved	Reserved	008C$_H$ - 03FC$_H$	nBE	nBE
ROMSIZE	ROM Size Register	0400$_H$	U, PV	BE

(table continues...)

Table 50 (continued) Register Overview

Register Name	Description	Offset Addr.	Access Mode	
			Read	Write
FLSIZE	Flash Size Register	0404_H	U, PV	BE
RAM0SIZE	RAM0 Size Register	0410_H	U, PV	BE
Reserved	Reserved	0414_H - $07FF_H$	nBE	nBE

12 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, for example, an 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 the unresponsive state into normal operation.

References

[6] Cortex®-M0 User Guide, Arm® DUI 0467B (ID081709)

12.1 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 and can be used to indicate 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, that is, between the lower and the upper boundary values. Correct servicing resets of the counter to 0_H. An incorrect service attempt ("Bad Service") results in a system reset request.

The timer block runs on the f_{WDT} clock, which is independent of the bus clock. The timer value is updated in the corresponding AHB register TIM. This mechanism enables immediate response on a read access from the bus.

12.1.1 Features

The watchdog timer (WDT) is an independent window watchdog timer.

WDT features include:

- Triggers system reset when not serviced on time or serviced in a wrong way
- Servicing restricted to be within boundaries of refresh window
- Can run from an independent clock
- Provides service indication to an external pin
- Can be suspended in halting mode
- Provides optional pre-warning alarm before reset

Table 51 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

12.1.2 Block Diagram

The WDT block diagram is shown in Figure 33.

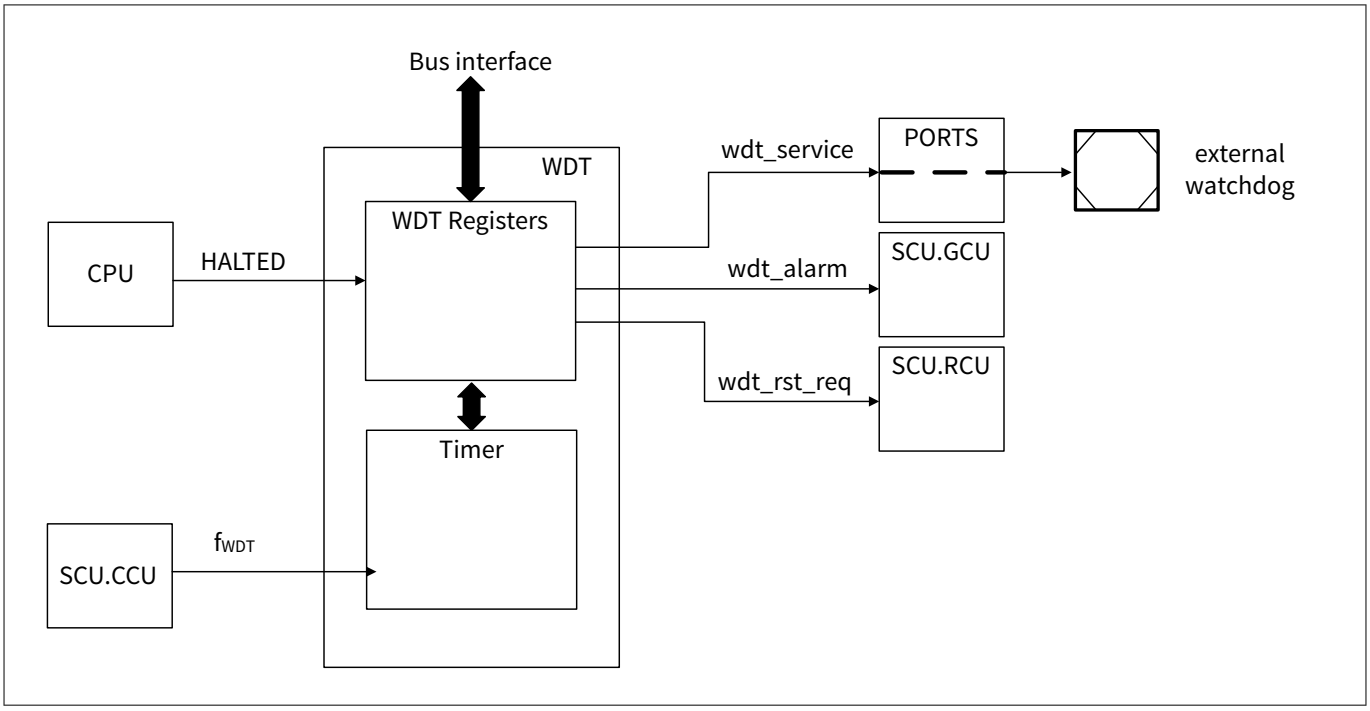


Figure 33 Watchdog Timer Block Diagram

12.2 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 boundary 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.

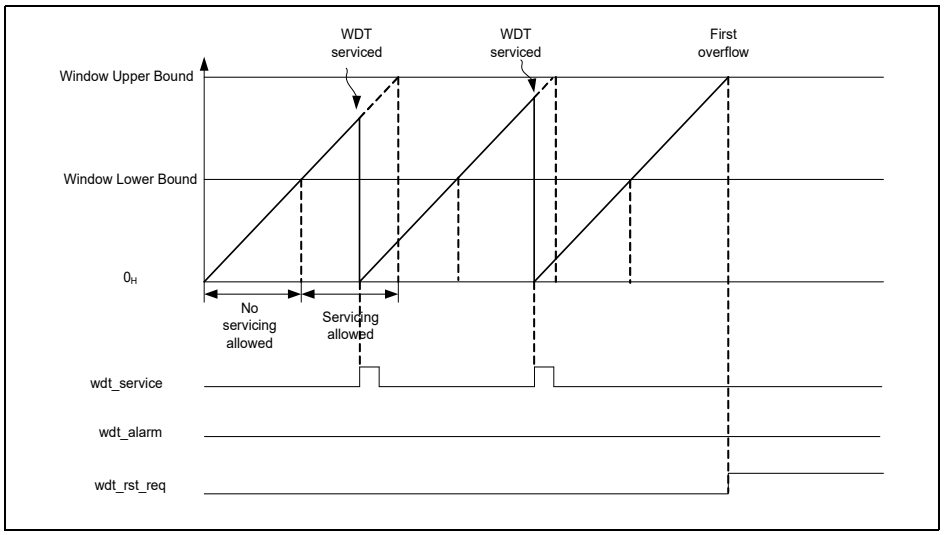


Figure 34 Reset without pre-warning

The example scenario depicted in Figure 34 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.

12.3 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.

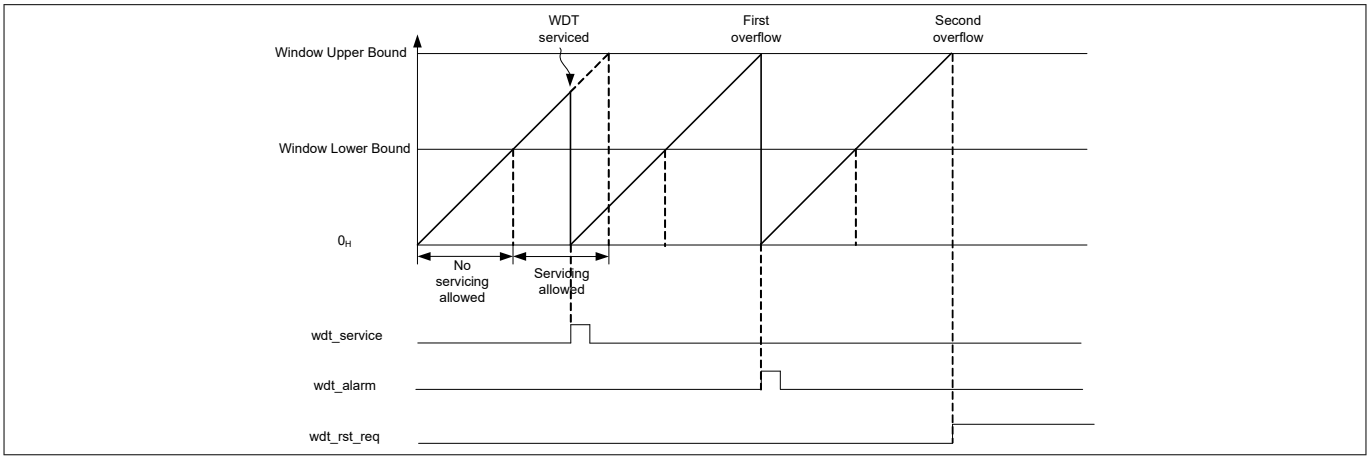


Figure 35 Reset after pre-warning

The example scenario depicted in Figure 35 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. 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_rstn upon the second missing service.

12.4 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.

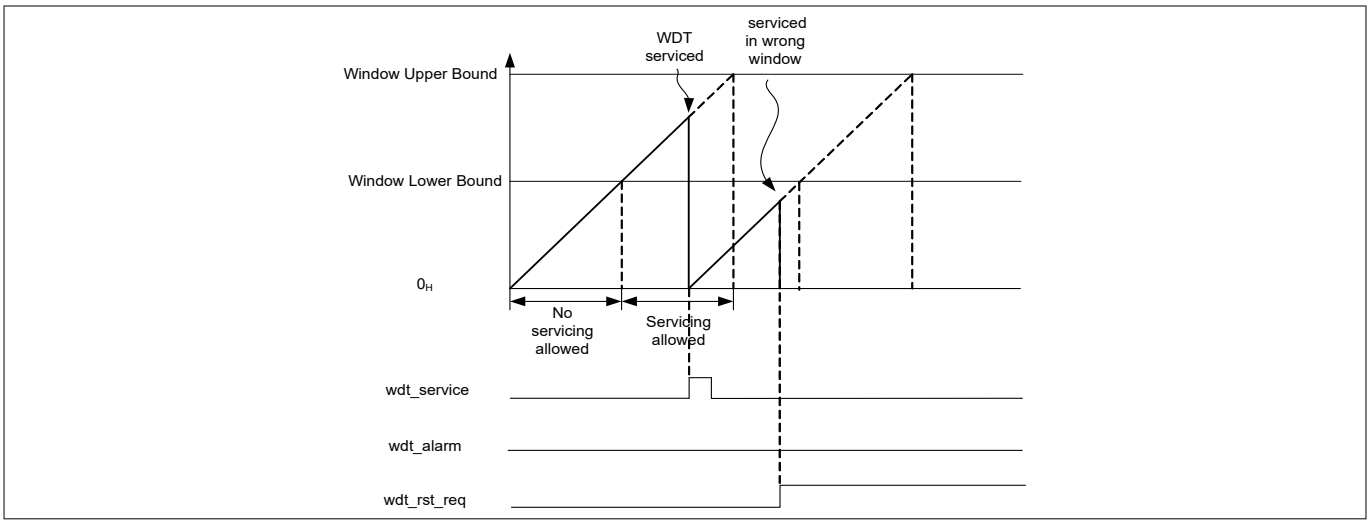


Figure 36 Reset upon servicing in a wrong window

The example in Figure 36 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.

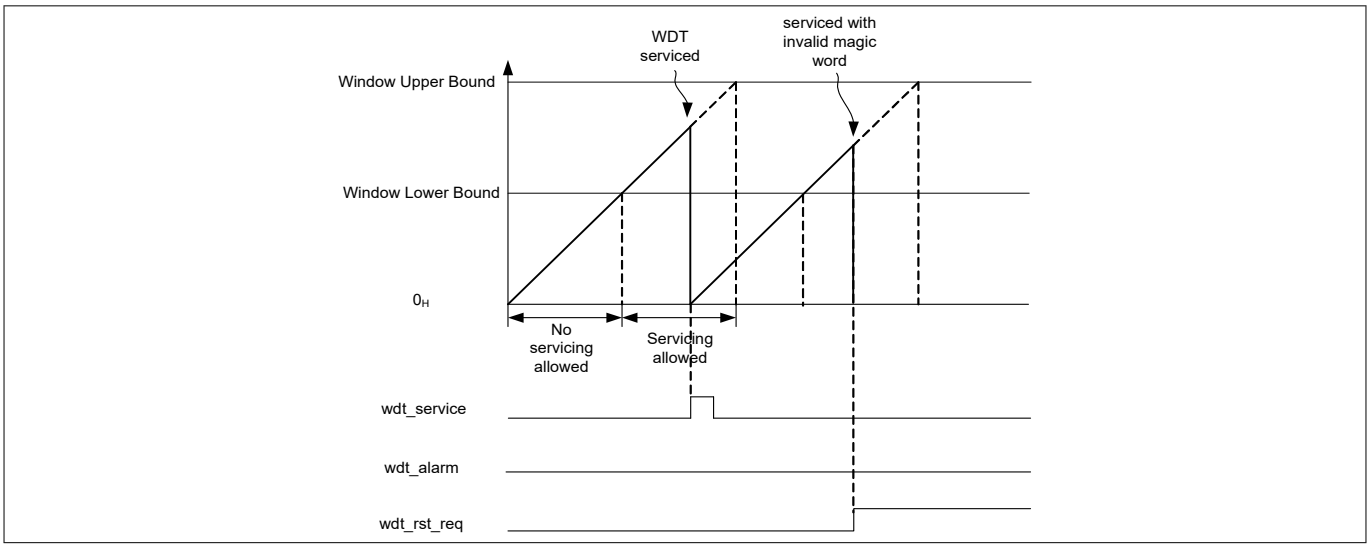


Figure 37 Reset upon servicing with a wrong magic word

The example in Figure 37 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.

12.5 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 is serviced in SCU.

Service requests can be disabled respectively by service request mask register in SCU.

12.6 Debug Behavior

The WDT function can be suspended when the CPU enters HALT mode. WDT debug function is controlled with DSP bit field in CTR register and it is set to be suspended by default.

12.7 Power, Reset and Clock

The WDT module is a part of the core domain and supplied with V_{DDC} voltage.

All WDT registers get reset with the system reset.

A sticky bit in the Reset Status Register, RSTSTAT, 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 is provided by the internal 32 kHz standby clock from SCU/CCU module, independently from the AHB interface clock.

The WDT module clock is disabled by default, and can be enabled via the SCU_CGATCLR0 register. Enabling and disabling the module clock could cause load change and clock blanking may occur as explained in the Clock Gating Control. It is strongly recommended to setup the module clock in the user initialisation code to avoid clock blanking during runtime.

12.8 Initialization and Control Sequence

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, refer to relevant chapters of this document.*

12.8.1 Initialization and Operation Start-Up

Complete WDT module initialization is required upon 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 and clear this bit using bit SCU_RSTCLR.RSCLR
  - perform appropriate operations dependent on the last system reset cause
- WDT software initialization sequence
  - enable WDT clock with SCU_CGATCLR0.WDT 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, see General Purpose I/O Ports (Ports))
  - enable interrupt for pre-warning alarm on system level with SCU_SRMSK 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
  - write magic word to WDT_SRV register within valid time window

12.8.2 Software Stop and Resume Operation

The WDT module can be stopped and re-started at any time, for example for debug purposes, using a 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
  - write magic word to WDT_SRV register within valid time window

12.8.3 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_CGATx register

- enter sleep/deep-sleep mode software sequence
  - select sleep or deep-sleep mode in CPU (for details please refer to Cortex®-M0 documentation [6])
  - enter selected mode (for details please refer to Cortex®-M0 documentation [6])
- wait for a wake-up event (no software interaction, CPU stopped)
- resume operation (CPU clock restarted automatically on an event)
- service the watchdog
  - write magic word to WDT_SRV register within valid time window

12.8.4 Pre-warning Alarm Handling

The WDT will fire pre-warning 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 (service request) clearing WDT_WDTSTAT register with WDT_WDTCLR register
- service the watchdog
  - write magic word to WDT_SRV register within valid time window

12.9 WDT Registers

Registers Overview

The absolute register address is calculated by adding:
Module Base Address + Offset Address

Table 52 Registers Address Space

Module	Base Address	End Address	Note
WDT	$4002\ 0000_H$	$4002\ FFFF_H$	Watchdog Timer Registers

Table 53 Register Overview

Register Name	Description	Offset Address	Access Mode	
			Read	Write
WDT Kernel Registers				
ID	Module ID Register	00_H	U, PV	PV
CTR	Control Register	04_H	U, PV	PV
SRV	Service Register	08_H	BE	PV
TIM	Timer Register	$0C_H$	U, PV	BE
WLB	Window Lower Bound	10_H	U, PV	PV
WUB	Window Upper Bound	14_H	U, PV	PV
WDTSTS	Watchdog Status Register	18_H	U, PV	PV
WDTCLR	Watchdog Status Clear Register	$1C_H$	U, PV	PV

12.10 Interconnects

Table 54 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	PORTS	service indication to external watchdog
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

13 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 computer systems.

13.1 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[1]. The clock of RTC is selectable via bit SCU_CLKCR.RTCCLKSEL.

The timer may remain operational during sleep or deep sleep mode.

13.1.1 Features

RTC features include:

- Real-time keeping with
  - 32.768 kHz external clock
  - 32.768 kHz internal clock
- Periodic time-based interrupt
- Programmable alarm interrupt on time match
- Supports wake-up mechanism from sleep or deep sleep mode

Table 55 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 sleep or deep sleep mode	Autonomous wakeups from sleep or deep sleep mode for system state control and maintenance routine operations

13.1.2 Block Diagram

The RTC block diagram is shown in Figure 38.

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 where dedicated fields represent a separate values for elapsing second, minutes, hours, days, days of week, months and years.

The RTC module is controlled directly from SCU module and shares system bus interface with other sub-modules of SCU.

Access to the RTC registers is performed via Register Mirror updated over serial interface.

1 The automatic leap year correction is performed when the year is divisible by 4.

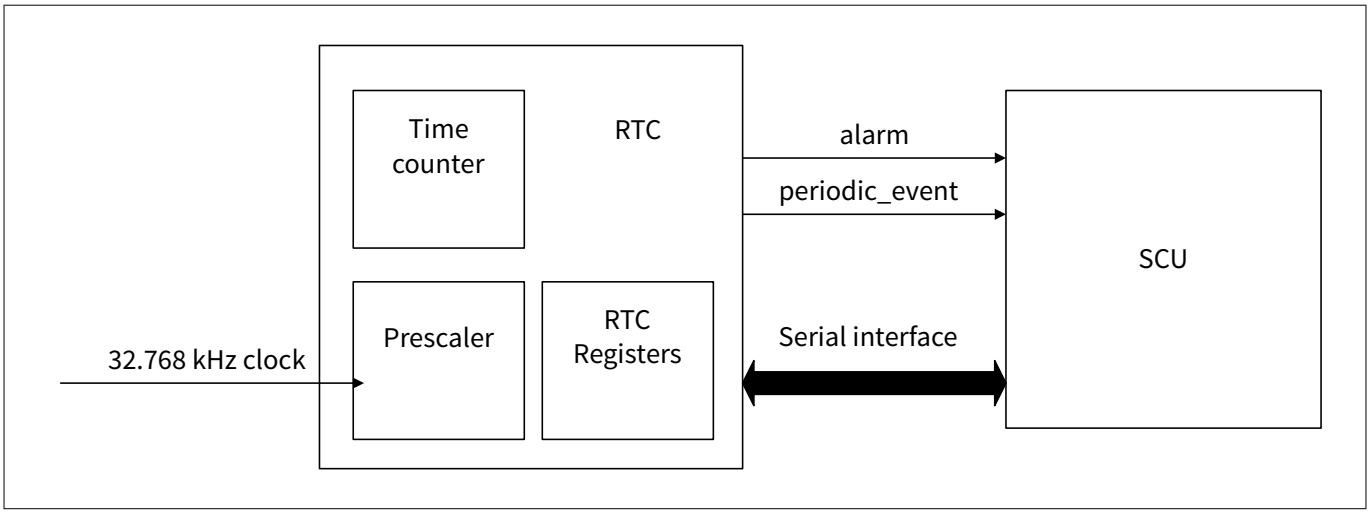


Figure 38 Real Time Clock Block Diagram Structure

13.2 RTC Operation

The RTC timer counts seconds, minutes, hours, days, days of week, months, and years, with the time in separate fields (see Figure 39). Individual bit fields of the RTC counter can be programmed and read with software over serial interface via mirror registers in SCU module.

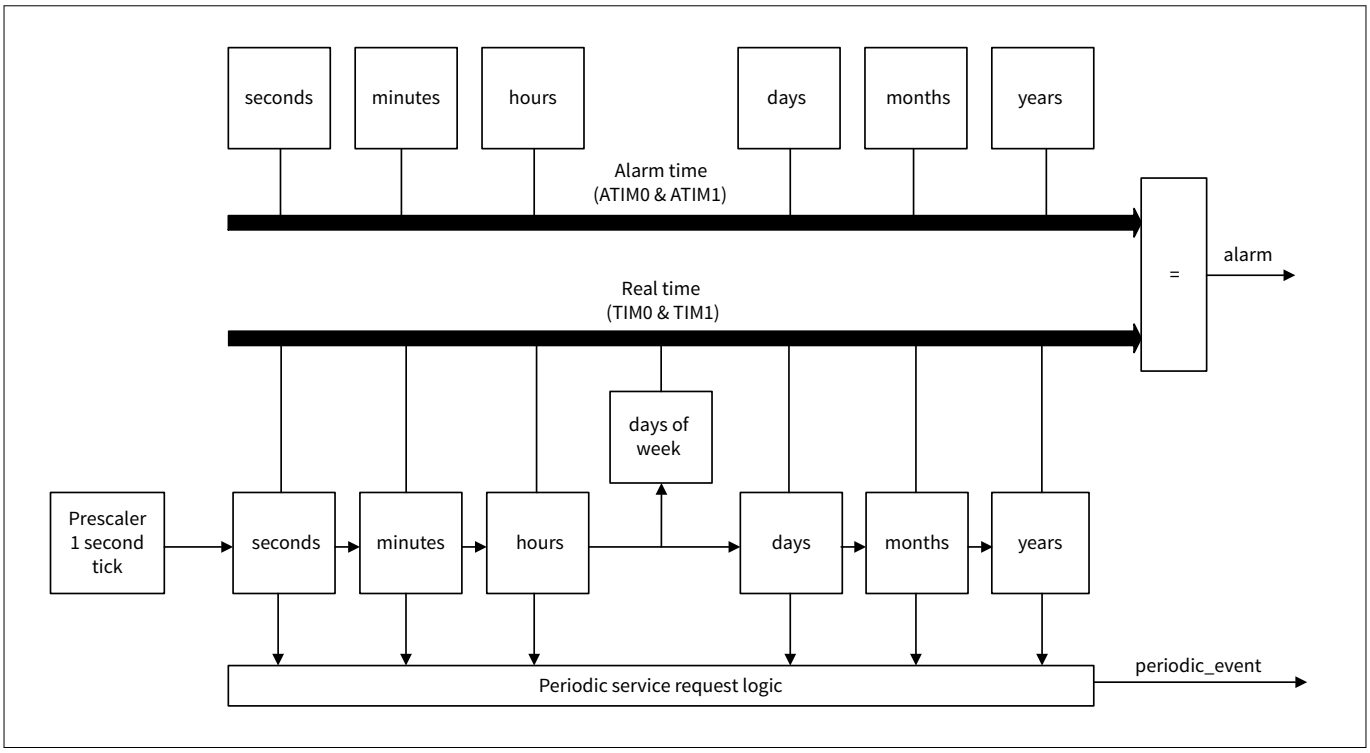


Figure 39 Block Diagram of RTC Time Counter

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.

13.3 Register Access Operations

The RTC module is a part of SCU from programming model perspective and shares register address space for configuration with other sub-modules of SCU. RTC registers are instantiated in the RTC module and mirrored in SCU. The registers get updated in both the clock domains over serial interface running at 32 kHz 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 refer to System Control Unit (SCU).

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.

For consistent read-out of the timer registers TIM0 and TIM1, the register TIM0 has to be read before the register TIM1. The value of TIM1 is stored in a shadow register upon each read of TIM0 before they get copied to the mirror register in core domain.

13.4 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 or as wake-up triggers from sleep or deep sleep mode.

13.4.1 Periodic Service Request

The periodic timer service request is raised whenever a non-masked field of the timer counter gets updated. Masking of the bits is performed using MSKSR register. Periodic Service requests can be disabled with MSKSR.

13.4.2 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/disabled with the MSKSR register.

13.5 Debug Behavior

The RTC function can be suspended when the CPU enters HALT mode. RTC debug function is controlled with SUS bit field in CTR register.

***Note**: In PSC1Mx, bit CTR.SUS is reset to its default value by any reset. Hence, if suspend function is required during debugging, it is recommended to enable the debug suspend function in the user initialization code. Before programming the register, the module clock has to be enabled and special care need to be handled while enabling the module clock as described in the Clock Gating Control.*

13.6 Power, Reset, and Clock

RTC can be programmed to remain powered up in sleep and deep sleep modes.

The RTC module remains in reset state after initial power up until reset released.

The RTC timer is running from an internal or external 32.768 kHz clock selectable with CLKCR.RTCCLKSEL control register of SCU/CCU module as shown in Figure 40. The prescaler setting of $7FFF_H$ results in an once per second update of the RTC timer.

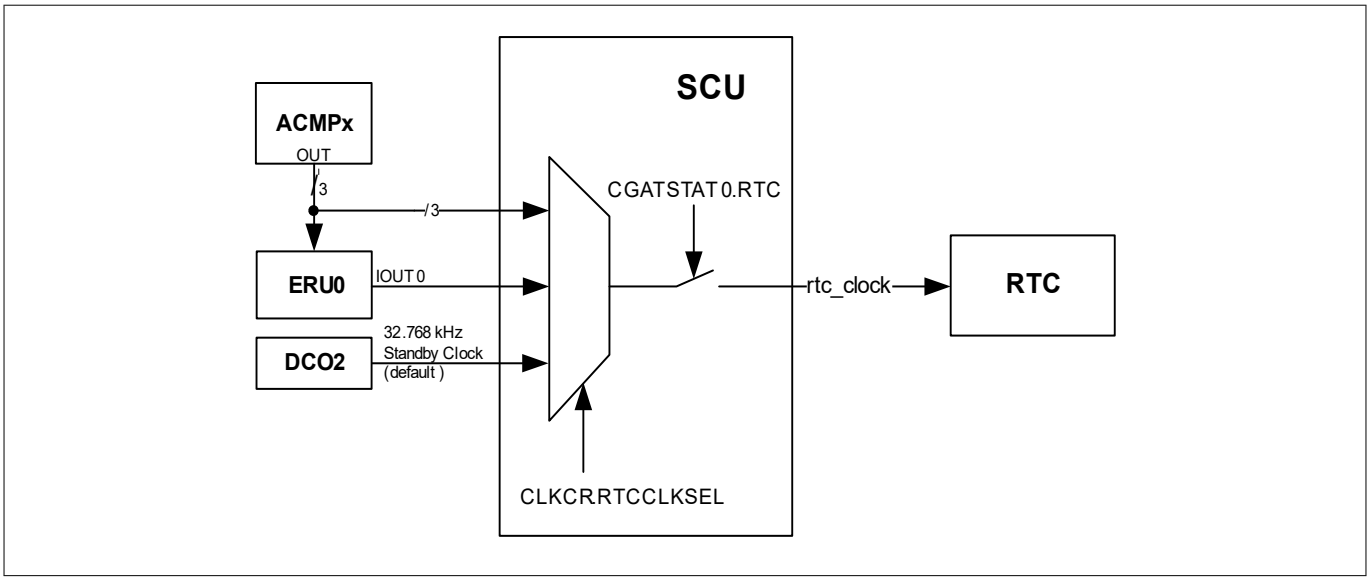


Figure 40 RTC Clock selection

The RTC module clock is disabled by default, and can be enabled via the SCU_CGATCLR0 register.

***Note**: Before changing RTC clock source via CLKCR.RTCCLKSEL, the RTC clock must be gated using bit CGATSET0.RTC.*

13.7 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, refer to relevant chapters of this document.*

13.7.1 Initialization & Start of Operation

Complete RTC module initialization is required upon reset. Accesses to RTC registers are performed via dedicated mirror registers (for more information, see System Control Unit (SCU))

- enable the clock to RTC module
  - write one to SCU_CGATCLR0.RTC
- 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 and RTC_TIM1 registers
  - write a new value to the RTC_TIM0 and RTC_TIM1 registers
- enable RTC module to start counting time
  - write one to RTC_CTR.ENB

***Note**: To ensure a successful transfer over serial interface, RTC_TIM0 and RTC_TIM1 can only be written once for each transfer. In addition, these registers must be written in 32-bit data width. Individual bit access will not start the serial transfer operation.*

13.7.2 Configure and Enable Periodic Event

The RTC periodic event configuration require programming in order to enable interrupt request generation out upon a change of value in the corresponding bit fields.

- enable service request for periodic timer events in RTC module
  - set respective bit field (MPSE, MPMI, MPHO, MPDA, MPMO, MPYE) in RTC_MSKSR register to enable the individual periodic timer events

13.7.3 Configure and Enable Timer Event

The RTC alarm event configuration require programming in order to enable interrupt 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 and RTC_ATIM1 registers
  - write to RTC_ATIM0 and RTC_ATIM1 registers
- enable service request for timer alarm events in RTC module
  - set MAI bit field of RTC_MSKSR register in order enable individual periodic timer events

***Note**: To ensure a successful transfer over serial interface, RTC_ATIM0 and RTC_ATIM1 can only be written once for each transfer. In addition, these registers must be written in 32-bit data width. Individual bit access will not start the serial transfer operation.*

13.8 RTC Registers

Registers Overview

The absolute register address is calculated by adding:
Module Base Address + Offset Address

Table 56 Registers Address Space

Module	Base Address	End Address	Note
RTC	$4001\ 0A00_H$	$4001\ 0AFF_H$	

Table 57 Register Overview

Register Name	Description	Offset Address	Access Mode	
			Read	Write
RTC Kernel Registers				
ID	ID Register	0000_H	U, PV	BE
CTR	Control Register	0004_H	U, PV	PV
RAWSTAT	Raw Service Request Register	0008_H	U, PV	BE
STSSR	Status Service Request Register	$000C_H$	U, PV	BE
MSKSR	Mask Service Request Register	0010_H	U, PV	PV
CLRSR	Clear Service Request Register	0014_H	U, PV	PV
ATIM0	Alarm Time Register 0	0018_H	U, PV	PV

(table continues...)

Table 57 (continued) Register Overview

Register Name	Description	Offset Address	Access Mode	
			Read	Write
ATIM1	Alarm Time Register 1	$001C_H$	U, PV	PV
TIM0	Time Register 0	0020_H	U, PV	PV
TIM1	Time Register 1	0024_H	U, PV	PV

13.9 Interconnects

Table 58 Pin Connections

Input/Output	I/O	Connected To	Description
Clock Signals			
f_{RTC}	I	SCU.CCU	32.768 kHz clock selected
Debug Signals			
HALTED	I	CPU	Indicates the processor is in debug state and halted
Service Request Connectivity			
periodic_event	O	SCU.GCU	Timer periodic service request
alarm	O	SCU.GCU	Alarm service request

14 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.

14.1 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, GCU Miscellaneous Control Functions (GCU)
- Power control, PCU Power Management (PCU)
- Reset operation, RCU Reset Control (RCU)
- Clock Control, CCU Clock Control (CCU)

14.1.1 Features

The following features are provided for monitoring and controlling the system:

- General Control
  - Start-up Software (SSW) and Boot Mode Support
  - Memory Content Protection
  - Interrupt Handling
- Power control
  - On-chip core supply generation via EVR
  - Power Validation
  - Supply Watchdog
  - Voltage Monitoring
  - Load Change Handling
- Reset Control
  - Reset assertion on various reset request sources
  - System Reset Generation
  - Inspection of reset sources after reset
- Clock Control
  - Clock Generation
  - Clock Supervision
  - Individual Peripheral Clock Gating
  - Clock Blanking Support
  - On-chip Oscillator Calibration
  - External Oscillator Controls

14.1.2 Block Diagram

Figure 41 shows the following sub-units:

- Power Control Unit (PCU)
- Reset Control Unit (RCU)
- Clock Control Unit (CCU)
- General Control Unit (GCU)

All the SFRs in the SCU module are accessible via the AHB and the 16-bit APB bus interface as shown in Figure 41.The APB bus interface is used to access the group of SFR called ANACTRL register. These registers are used to configure the analog modules in the system, namely, the embedded voltage regulator (EVR) and the digitally controlled oscillators (DCO1 and DCO2). The other group of SFRs, called the SCU registers, are accessible via the AHB bus interface.

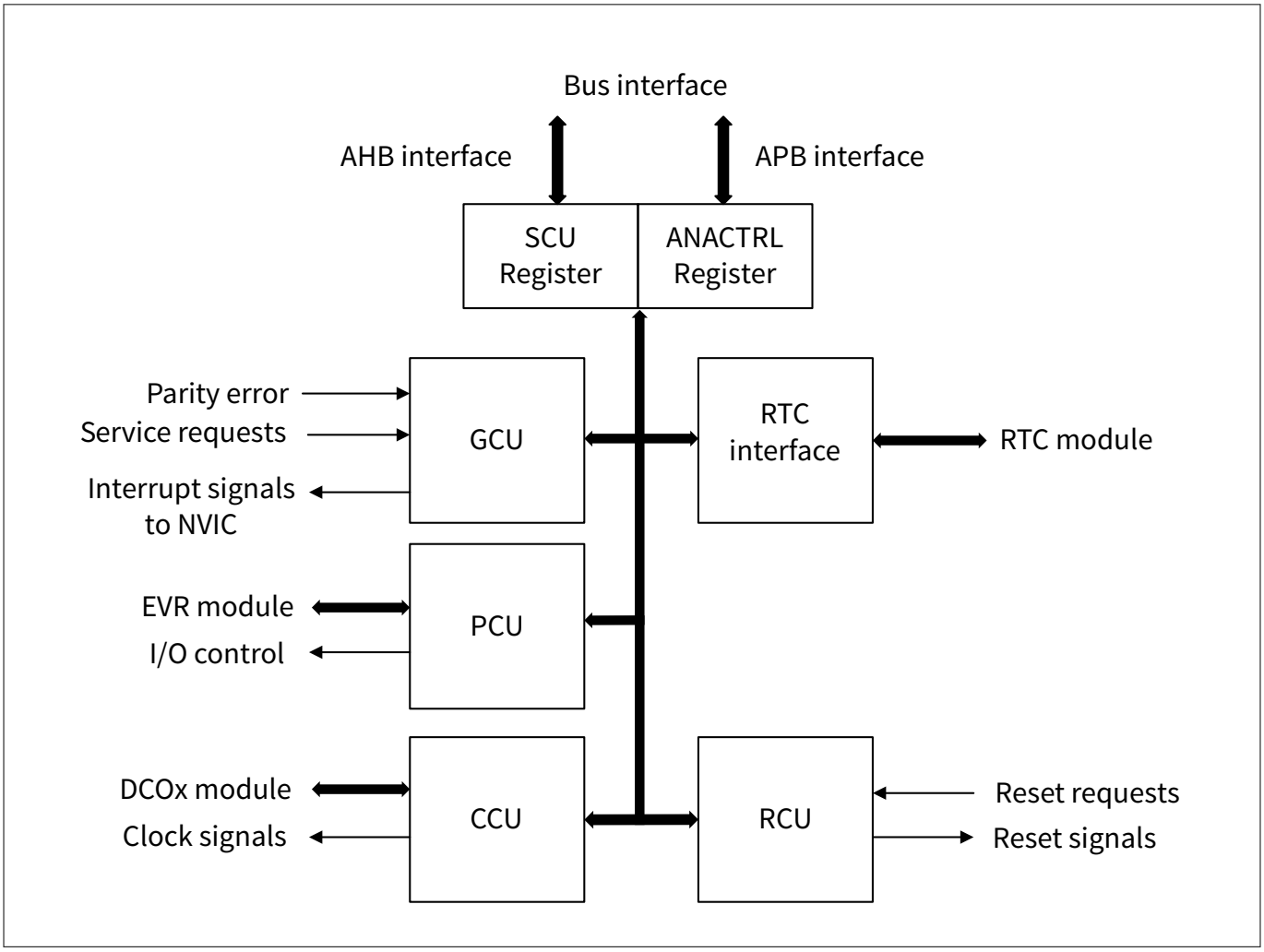


Figure 41 SCU Block Diagram

Interface of General Control Unit

The 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.

Interface of Power Control Unit

The PCU has an interface to the Embedded Voltage Regulator (EVR) and an interface to the CCU module. The PCU related signals are described in more detail in Power Management (PCU).

Interface of Reset Control Unit

The RCU has an interface to the Embedded Voltage Regulator (EVR). The RCU receives the power-on reset and the brownout reset information from the EVR. Reset requests are coming to the unit from the watchdog, the CPU, the GCU and CCU. The RCU provides reset signals to all other units of the chip in the core power domain. The RCU related signals are described in more detail in Reset Control (RCU).

Interface of Clock Control Unit

The CCU receives the clock source from the on-chip Digitally Controlled Oscillator (DCO). The CCU provides the clock signals to all other units of the chip.

Interface of RTC

Access to the RTC module is served over a serial interface. The interface provides mirror registers updated via the serial interface to the RTC module registers. Update of the mirror registers over the serial is controlled using MIRRSTS registers. End of update can also trigger service requests via SRRAW register. Refresh of the registers in the register mirror are performed continuously, as fast as possible in order to instantly reflect any register state change on both sides.

The RTC module functionality is described in Real-Time Clock (RTC).

14.2 Miscellaneous Control Functions (GCU)

System Control implements system management functions accessible via GCU registers. General system control including various auxillary function is performed in the General Control Unit (GCU).

14.2.1 Service Requests Handling

Service request events listed in Table 59 can result in assertion of an interrupt. Please refer to SRMSK/SRMSK1 register description.

The interrupt structure is shown in Figure 42. The interrupt request or the corresponding interrupt set bit (in register SRSET/SRSET1) can trigger the interrupt generation at the selected interrupt node x. The service request pulse is generated independently from the interrupt flag in register SRRAW/SRRAW1. The interrupt flag can be cleared by software by writing to the corresponding bit in register SRCLR/SRCLR1.

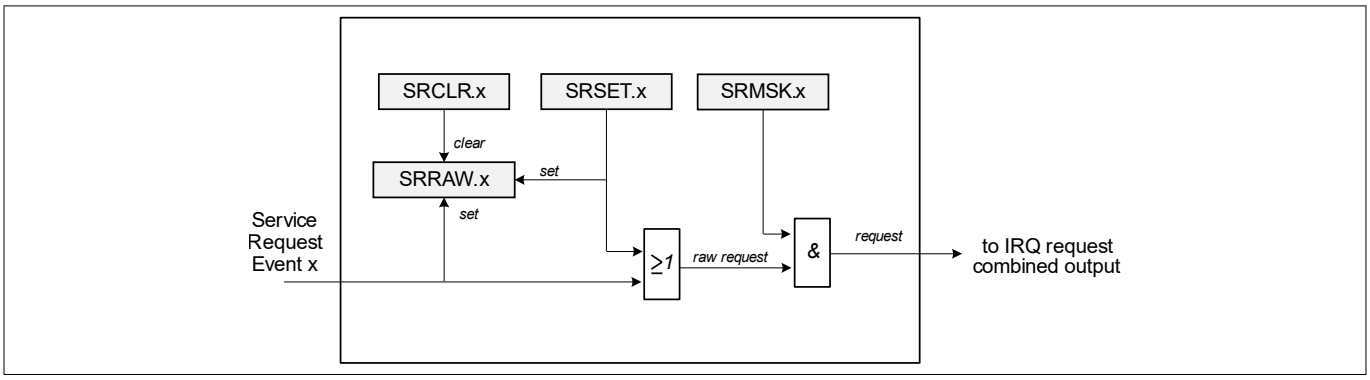


Figure 42 Service Request Handling

The flag in register SRRAW/SRRAW1 can be cleared by software by writing to the corresponding bit in register SRCLR/SRCLR1. All trap requests are combined to one common line and connected to a regular interrupt node of NVIC.

***Note**: When servicing an SCU service request, make sure that all related request flags are cleared after the identified request has been handled.*

14.2.1.1 Service Request Sources

The SCU supports service request sources listed in Table 59 and reflected in the SRRAW/SRRAW1, SRMSK/SRMSK1, SRCLR/SRCLR1 and SRSET/SRSET1 registers. The events that trigger these service requests are described in the respective module chapters or in the various sections within SCU.

Table 59 Service Requests

Modules	Service Request Name	Service Request Short Name	SCU.SRx
NVM	Flash Double Bit ECC Event	FLECC2I	SR0
	Flash Operation Complete Event	FLCMPLTI	
	16kbytes SRAM Parity Error Event	PESRAMI	
	USIC0 SRAM Parity Error Event	PEU0I	
	USIC1 SRAM Parity Error Event	PEU1I	
	MultiCAN SRAM Parity Error Event	PEMCI	
SCU:CCU	Loss of DCO1 Clock Event	LOCI	
	Loss of External OSC_HP Clock Event	LOECI	
	Standby Clock Failure Event	SBYCLKFI	SR1
	DCO1 out of synchronisation Event	DCO1OFSI	
SCU:PCU	VDDP Pre-warning Event	VDDPI	
	VDROP Event	VDROPI	
	VCLIP Event	VCLIPI	
SCU:GCU	Temperature Sensor Done Event	TSE_DONE	
	Temperature Sensor Compare High Event	TSE_HIGH	
	Temperature Sensor Compare Low Event	TSE_LOW	
WDT	WDT pre-warning	PRWARN	
RTC	RTC Periodic Event	PI	
	RTC Alarm	AI	
	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	
ORC	Out of Range Comparator Event	ORCxI [x = 0 - 7]	SR2
ACMP	Analog Comparator Event	ACMP0I, ACMP1I, ACMP2I. ACMP3I	

14.2.2 SRAM Memory Content Protection

For supervising the content of the on-chip SRAM memories, the following mechanism is provided:

All on-chip SRAMs provide protection of content 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 SRRAW/SRRAW1 status register. It is configurable via SRMSK/SRMSK1 whether a memory error should trigger an interrupt. It can also trigger a system reset when RSTCON.SPERSTEN, RSTCON.U0PERSTEN, RSTCON.U1PERSTEN or RSTCON.MPERSTEN is set to 1.

A parity control software test, such as to support in-system testing to fulfill Class B requirements, can be enabled with bit PMTSR.MTENS for the 16 kbytes SRAM memory individually. Once this bit is set, an inverted parity bit is generated during a write operation. When a read operation is performed on this SRAM address, a parity error shall be detected.

***Note**: Test software should be located in a different memory space.*

14.2.3 Summary of ID

This section describes the various IDs in PSC1Mx.

Module Identification

The module identification register indicates the function and the design step of each peripherals. Register SCU_ID is used for SCU module.

System ROM Table ID

The PID values in the system ROM table are defined in the Table 60. The PSC1Mx system ROM table is located at $F000\ 0000_H$. Cortex®-M0 ROM table is described in the Debug System (DBG).

Table 60 PID Values of PSC1Mx System ROM Table

Name	Offset	Reference	Values
PID0	$FE0_H$	PSC1Mx Part Number [7:0]	$C1_H$
PID1	$FE4_H$	bits [7:4] JEP106 ID code [3:0] bits [3:0] PSC1Mx Part Number [11:8]	90_H
PID2	$FE8_H$	bits [7:4] PSC1Mx Revision bit [3] == 1: JEDEC assigned ID fields bits [2:0] JEP106 ID code [6:4]	08_H
PID3	FEC_H	bits [7:4] RevAnd, minor revision field bits [3:0] if non-zero indicate a customer-modified block	00_H
PID4	$FD0_H$	bits [7:4] 4KB count bits [3:0] JEP106 continuation code	00_H

Chip Identification Number

The chip identification number is a eight word length number. It consists of the values in register DBGROMID, IDCHIP, register PAU_FLSIZE, register PAU_RAM0SIZE, and register PAU_AVAILn(n=0-2) respectively. This number is for easy identification of product variants information of the device, such as, package type, the temperature profile, flash size, RAM size, and the peripheral availability.

14.2.4 Boot via Pins

In PSC1Mx, 4 boot modes, namely ASC BSL, CAN BSL, User productive mode and User Boot mode can be entered via pins (P4.6 and P4.7). The input of boot pins are latched during the de-assertion of reset and stored in register bit STSTAT.HWCON. For more details about the selection of this mode, see Boot and Startup.

14.3 Power Management (PCU)

Power management control is performed in the Power Control Unit (PCU).

14.3.1 Functional Description

The PSC1Mx is running from a single external power supply of 1.8 - 5.5V (V_{DDP}). The main supply voltage is supervised by a supply watchdog.

The I/Os is 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.

14.3.2 System States

The system has the following general system states:

- Off
- Active
- Sleep
- Deep-Sleep

Figure 43 shows the state diagram and the transitions between the states.

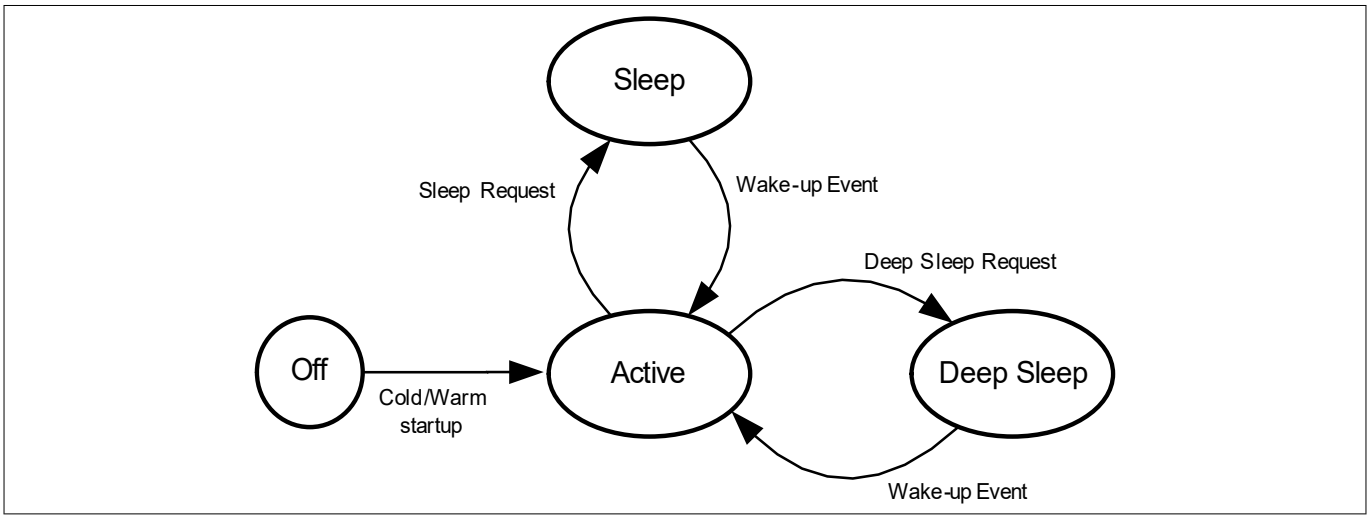


Figure 43 System States Diagram

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. Unused peripherals might be stopped by gating the clock to these peripherals.

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. To save power, the clock of the peripherals that are not needed during sleep state can be gated by register CGATSET0 before entering sleep state.

The Flash can be put into shutdown mode during active state to achieve a further power reduction before entering sleep state via bit NVMCONF.NVM_ON. However, user code would have to be executed in SRAM before entering sleep state and after waking up from sleep state.

To avoid the switching of code execution to SRAM due to the shutdown of flash, register PWRSVCR can be used. When FPD bit is set, flash is shutdown only when the device has entered sleep state. The shut down operation is performed after the core has executed WFI/WFE instruction. After a wake-up event is detected, the system will resumed to the previous state i.e. the flash is operable again before the CPU can continue to fetch and execute the code. In this case, user code can be executed in Flash and no switching to SRAM in needed. In his approach, the wake-up time is longer because of the time needed for flash to reach the active state.

Peripherals can continue to operate unaffected and eventually generate an event to wake-up the CPU. In User with Debug mode (UMD) or User with Debug mode and HAR (UMHAR), a Debug HALT request is also able to wake-up the CPU. Any accordingly configured interrupt will bring the CPU back to operation via the NVIC or the M0 debug system.

Deep-Sleep State

The deep-sleep state is entered on the same mechanism as the sleep state with the addition that user code has enabled the deep-sleep state in system control register. This state is similar to sleep state, except that in deep-sleep state, the PCLK and MCLK will be switched to a slow standby clock and DCO1 will be put into power-down mode.

The analog comparator could also be kept active during deep-sleep state. It can be switched to low power mode to save power.

The shutting down of flash via NVMCONF.NVM_ON or PWRSVCR.FPD as explained in above section is applicable for deep-sleep mode.

Peripherals that continue to operate will run using the slow standby clock and can eventually generate an event to wake-up the CPU. In User with Debug mode (UMD) or User with Debug mode and HAR (UMHAR), a Debug HALT request is also able to wake-up the CPU. Any accordingly configured interrupt will bring the CPU back to operation via the NVIC or the M0 debug system. The clock system is restored to the previous configuration for active state upon wake-up. Peripherals that are active will run with restored clock configuration.

The SRAM content is preserved in the deep-sleep state.

***Note**: It is recommended to slow down the PCLK and MCLK before entering deep sleep mode to prevent a sudden load change that could cause a brownout reset.*

***Note**: When DCO1 is used for the generation of MCLK/PCLK, it is recommended to disable the DCO1 calibration function before entering deep-sleep mode.*

14.3.3 Embedded Voltage Regulator (EVR)

The EVR generates the core voltage V_{DDC} out of the external supplied voltage V_{DDP}. The EVR provides two supply monitoring detectors for the input voltage V_{DDP}. The generated core voltage V_{DDC} is monitored by a power validation circuit (PV).

14.3.4 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.

14.3.5 Power Validation

A power validation circuit monitors the internal core supply voltage,V_{DDC}. It monitors that the core voltage is above the voltage threshold V_{DDCBO} which guarantees safe operation. Whenever the voltage falls below the threshold level, a brownout reset is generated.

14.3.6 Supply Voltage Monitoring

There are two detectors, namely, External voltage detector (VDEL) and External brownout detector (BDE) in the EVR that are used to monitor the V_{DDP}.

VDEL detector compares the supply voltage against a pre-warning threshold voltage. The threshold level is programmable via register ANAVDEL.VDEL_SELECT. An interrupt if enabled, will be triggered if a level below this threshold is detected and the flag, VDDPI, in SRRAW register bit is set. An indication bit, VDESR.VDDPPW, shows the output of the detector. During power-up, this indication bit can be polled to ensure that the external supply has reached the pre-warning voltage before starting the application code.

BDE detector is used to trigger a brownout reset when the V_{DDP} supply voltage drops below the defined threshold. Similarly, it is also used to ensure a proper startup during the power-up phase.

The Datasheet defines the nominal value and applied hysteresis.

14.3.7 V_{DDC} Response During Load Change

In PSC1Mx, the core voltage level, V_{DDC}, drops below the typical threshold when there is an increase in the load and rises when there is a decrease in the load. two detectors, VDROP, and VCLIP detectors are used to monitor the lower limits and the upper limits of the core voltage level respectively (detectors details are described in the next section). A VDROP event happens when VDDC drops below the VDROP threshold voltage. A VCLIP event happens when VDDC rises above the VCLIP threshold voltage. Each of these events can triggered it's dedicated interrupt if enabled via SRMSK register and the event status can be monitor via SRRAW register.

In PSC1Mx, the following scenarios may trigger a VDROP/VCLIP event due to load change:

- Changing the MCLK and PCLK frequency via CLKCR register
- Enabling/Gating the peripheral clock via the CGATSET0/CGATCLR0 registers

When a sudden load change occurs (> 4*baseload or < 0.25*baseload), regardless of an increase or decrease in load, the EVR needs time (15 µs) to regulate the core voltage back to a stable nominal voltage. During this period of time, the system is expected to maintain in the current load and no load change is allowed. Status bit VDDC2LOW and VDDC2HIGH in CRCLK register are used to indicate whether the voltage is stable.

***Note**: It is not recommended to increase the load more than 4 times of the baseload or decrease the load to less than 0.25 times of the baseload. If a bigger than the specified load change is required, the recommendation is to change the load in steps that each steps are within the limits. For example, a final load of 16 mA from the current baseload of 1 mA needs at least 2 steps. A step from 1 mA to 4 mA followed by another step from 4 mA to 16 mA.*

The VDDC2LOW and VDDC2HIGH status bit is generated by a 10-bit counter using 48 MHz clock as the clock input. It is implemented to count the 16 µs (default) that is needed to have a stable V_{DDC} after a VDROP or VCLIP event happens. The length of this counter can be changed depending on the amount of load change via bit CLKCR.CNTADJ. The larger is the load change, the more is the time that user need to wait for a stable clock. Refer to datasheet for a guideline of the current consumption of each modules.

Using the example above, after programming some configuration that causes a change in load from 1 mA to 4 mA, user can poll for VDDC2LOW (CNTADJ=300_H) to ensure the 16 µs (max) needed to regulate EVR. After VDDC2LOW is set to '0', another load change from 4 mA to 16 mA can be performed and the cycle repeats for each step of load change.

During a VDROP event, clock blanking happens and the detail description is documented in Clock Blanking. CPU clock and peripheral clocks continue to run during VCLIP event.

***Note**: When overflow event happens while the VDROP=1 (time to overflow based on the CNTADJ value is shorter than the time the device stays in a vdrop event), the 10-bit counter will automatically be restarted with the CNTADJ value.*

***Note**: VDROP and VCLIP detectors are disabled during deep sleep mode.*

14.3.8 Flash Power Control

The Flash module can be switched off to reduce static power. In sleep or deep-sleep state, it is dependent on the setting of register PWRSVCR whether the Flash module will be put to sleep or not in this state. The user has to evaluate the reduced leakage current against the longer startup time. In addition, the Flash can also be put to sleep using NVMCONF.NVM_ON before entering these power save modes.

14.4 Reset Control (RCU)

Reset Control Unit performs control of all reset related functionality including:

- Reset assertion on various reset request sources
- Inspection of reset sources after reset
- Selective reset of peripheral

14.4.1 Functional Description

The PSC1Mx has the following reset types for the system:

- Master Reset, $\overline{\text{MRESET}}$
- System Reset, $\overline{\text{SYSRESET}}$

Master Reset, $\overline{\text{MRESET}}$

Master reset is triggered by:

- Power-on reset (PORST)
- V_{DDP} or V_{DDC} undervoltage reset (also known as brownout reset)
- SW master reset via setting bit RSTCON.MRSTEN to 1

A complete reset to the device is executed by a master reset. Master reset is triggered by a power-on reset upon power-up. Whenever the supply V_{DDP} is ramped-up and crossing the V_{DDP} and V_{DDC} voltage threshold, the power-on reset is released. A power-on reset (also known as brown-out reset) is asserted again whenever the V_{DDP} voltage or the V_{DDC} voltage falls below reset thresholds. In addition, a master reset can be triggered by setting bit RSTCON.MRSTEN.

The sources that trigger a master reset also triggers a system reset $\overline{\text{SYSRESET}}$.

System Reset, $\overline{\text{SYSRESET}}$

System reset is triggered by:

- SW reset via Cortex® M0 Application Interrupt and Reset Control Register (AIRCR)
- Lockup signal from Cortex® M0 when enabled at RCU
- Watchdog reset
- Memory parity error when enabled
- Flash ECC double bit error when enabled

- Loss of DCO1 clock when enabled
- sources that trigger a master reset

A system reset affects almost all logics. The only exceptions are RCU registers and Debug system when debug probe is present. The reset gets extended to the length defined by implementation requirements.

The debug system is reset by System Reset in normal operation mode when debug probe is not present. When debug probe is present, System Reset must not affect the Debug system.

14.4.2 Reset Status

The EVR provides the cause of a power on reset to the RCU. The reset cause can be inspected after resuming operation by reading register RSTSTAT. This register also indicates the source of event that causes the triggering of a system reset.

All registers of the RCU undergo reset only by a master reset except for RSTSTAT and RSTCON register. RSTSTAT register is only reset by a power-on reset and RSTCON is reset by any reset type.

***Note**: Clearing of the reset status via register bit RSTCLR.RSCLR is strongly recommended to ensure a clear indication of the cause of next reset.*

Table 61 shows an overview of the reset signals their source and effects on the various parts of the system.

Table 61 Reset Overview

Module/Function	Power-on Reset	Master Reset by SW bit	System Reset
CPU Core	yes	yes	yes
SCU	yes	yes	yes, except reset indication bit
Peripherals	yes	yes	yes
Debug System	yes	yes	see footnote[1) 2)]
Port Control	yes	yes	yes
SRAM	Affected,unreliable	Not affected	Not affected
Flash	yes	yes	yes
EVR	yes	no[3)]	no
Clock System	yes	yes	yes

1) Debug system will be reset only if debug probe is not present.
2) Access to debug interface is disabled after every reset even when debug probe is present. See Warm Reset for more details.
3) The supply to EVR will not be affected and hence a complete reset to EVR is not possible. However, it will be partially affected by the reset in the ANACTRL module.

14.5 Clock Control (CCU)

14.5.1 Features

The clock control unit CCU has the following functionality:

- Dedicated RTC and standby clock
- Clock Supervisory
  - Oscillator watchdog
- Wide range of frequency scaling of system frequencies
- Individual peripheral clock gating
- Calibration of DCO based on external reference clock or the temperature sensor
- Controls for the external oscillator, such as crystal oscillators

14.5.2 Clock System and Control

Figure 44 shows the block diagram of the clock system in PSC1Mx. It consists of two on-chip oscillators (DCO1[2)] with synchronization unit and DCO2), two oscillators pads (OSC_HP and OSC_LP to drive external clock), a doubler, and a clock control unit (CCU). DCO1 has a clock output (dco1_clk), running at 48 MHz. DCO2 is used to generate the standby clock running at 32 kHz.

The main clock, MCLK, and fast peripheral clock, PCLK, are generated from DCLK (output of the doubler clock). Input to DCLK can be selected using bit CLKCR1.DCLKSEL to be either from the DCO1 clock source or the external clock source via the OSC_HP oscillator. After reset, DCLK is generated from DCO1 clock source. To select the external clock source, the sequence described in DCLK input - External Clock via OSC_HP is required.

PCLK is running in either the same frequency as MCLK or double the frequency of MCLK. It is selectable via CLKCR.PCLKSEL.

Figure 44 shows the list of peripherals that are running in the PCLK domain. The rest of the peripherals except RTC and WDT are clocked by MCLK which is the same as the core and bus system. WDT is running at a frequency of 32 kHz from the standby clock which is asynchronous to the MCLK and PCLK clock. RTC is running asynchronously from either the internal 32 kHz standby clock or the external 32.768 kHz XTAL (via ERU/ACMPx or the OSC_LP).

[2] The accuracy of the DCO1 clock output can be improved by calibrating it based on the die temperature given by the temperature sensor. Refer to Calibrating DCO1 based on Temperature and Automatic DCO1 Calibration based on External Reference for detailed description.

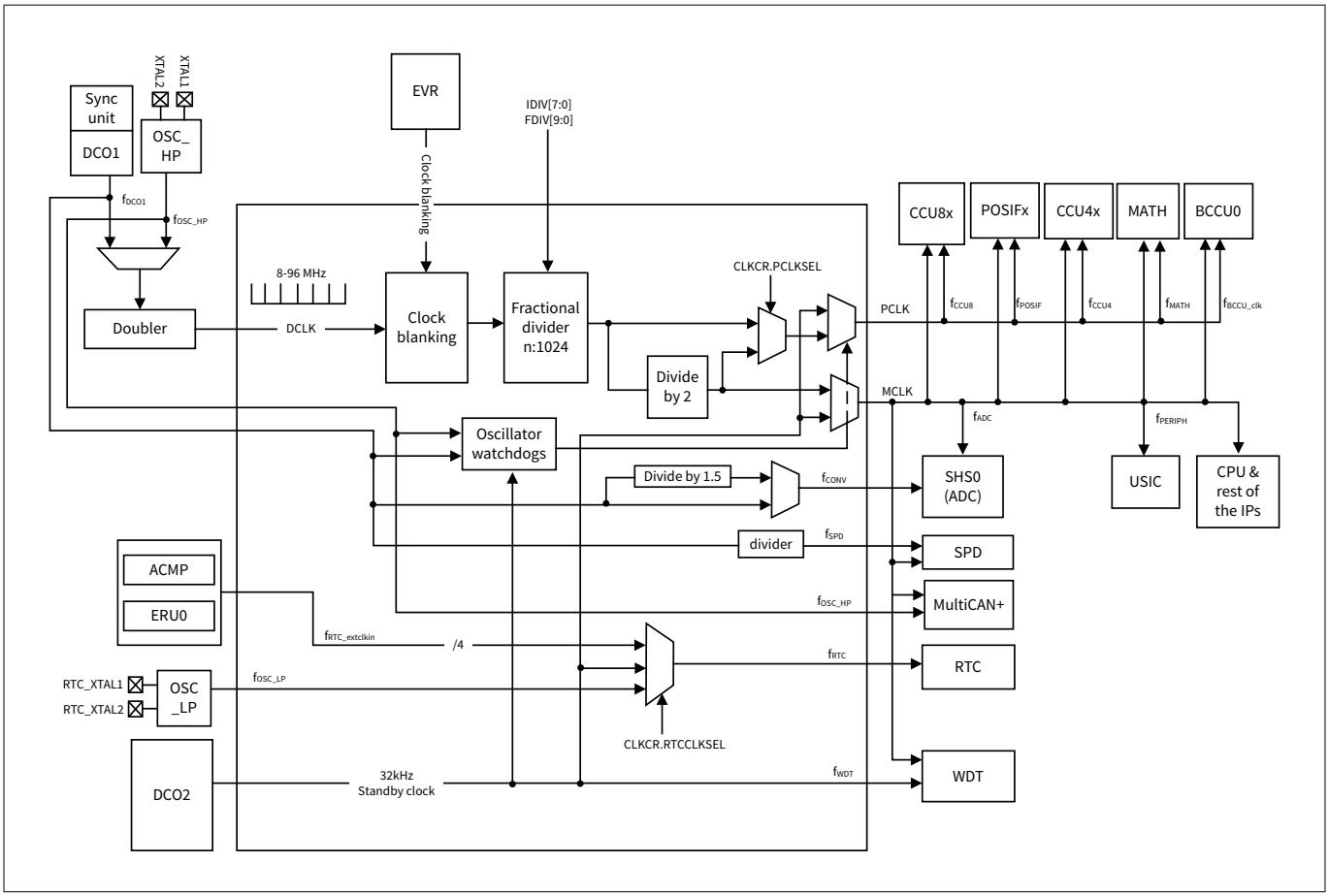


Figure 44 Clock System Block Diagram

***Note**: SHS(ADC) clock will not switched to standby clock source when there is a loss of DCO1 or external clock event.*

***Note**: DCO1 must be enabled (OSCCSR.DCO1PD = 0_B) when debug system via SPD or ADC is enabled.*

***Note**: RTC_extclkin are further described in Real-Time Clock (RTC).*

***Note**: MultiCAN+ clocks are further described in MultiCAN+ Clock.*

The following clock configuration option can be used for maximum performance and clock accuracy:

- Select DCO1 for MCLK (max 48 MHz) and PCLK (max 96 MHz)
- Select external crystal (20 MHz) for accurate CAN baud rate generation
- Calibrate DCO1 based on external crystal to achieve accurate MCLK/PCLK (see Automatic DCO1 Calibration based on External Reference)

Fractional Divider

The frequency of MCLK and PCLK are programmable through a fractional divider. The range of PCLK and MCLK frequencies based on different clock sources are shown in Table 62.

Table 62 PCLK and MCLK frequency range

DCLK clock source	DCLK frequency	PCLK frequency	MCLK frequency
DCO1 (48 MHz)	96 MHz	188 kHz - 96 MHz	188 kHz - 48 MHz
External oscillator via OSC_HP in osc mode (4 - 20 MHz)	8 MHz - 40 MHz	15.6 kHz - 40 MHz	15.6 kHz - 20 MHz
External clock via OSC_HP in direct-in mode (4 - 48 MHz)	8 MHz - 96 MHz	15.6 kHz - 96 MHz	15.6 kHz - 48 MHz

The following formula calculate the MCLK clock frequency.

$$\left(MCLK = \frac{DCLK}{\left(2\right) \times \left(IDIV + \frac{FDIV}{(1024)}\right)}\right) \quad \text{for} \quad IDIV > 0 \tag{10}$$

The following formula calculate the PCLK clock frequency when CLKCR.PCLKSEL is set to '1' and it is double the frequency of MCLK.

$$\left(PCLK = \frac{DCLK}{\left(IDIV + \frac{FDIV}{(1024)}\right)}\right) \quad \text{for} \quad IDIV > 0 \tag{11}$$

IDIV represents an unsigned 8-bit integer from the bit field CLKCR.IDIV and FDIV/1024 defines the fractional divider selection in CLKCR.FDIV. FDIV is an unsigned 10-bit integer from bit field CLKCR1[1:0] and CLKCR[7:0].

Table 63 Examples of MCLK frequency (FDIV=0)

DCLK frequency	IDIV = 1	IDIV = 2	IDIV = 4	IDIV = 8	IDIV = 128
96 MHz via DCO1	48 MHz	24 MHz	12 MHz	6 MHz	375 kHz
40 MHz via 20 MHz crystal oscillator	20 MHz	10 MHz	5 MHz	2.5 MHz	156 kHz

Below is the sequence to program the MCLK and PCLK frequency :

- Enable the access of protected bit IDIV and FDIV via register PASSWD
- Program the IDIV, FDIV value
  - write value to CLKCR1.FDIV (optional, could be skipped if no update is required)
  - write value to CLKCR.IDIV/FDIV (mandatory, IDIV/FDIV is valid only when register CLKCR is written)

Changing of the MCLK and PCLK can be done within two clock cycles.

***Note**: Changing the MCLK and PCLK frequency may result in a load change that causes clock blanking to happen. Refer to Clock Blanking and V_{DDC} Response During Load Change for more details.*

Clock Blanking

To prevent a brown-out reset in case of a sudden positive load change, the clock blanking circuitry is used. It freezes the clock input to the fractional divider to regulate the load change. The clock blanking only causes a small jitter if it is considered over a longer time period.

The clock blanking is activated when V_{DDC} is detected to be below the VDROP threshold. Once the V_{DDC} is above this threshold, the enabled clocking is resumed. This voltage drop detector is part of the EVR and it is activated by default upon any reset.

To monitor clock blanking activities, user can enable interrupt but can only enter ISR after the clock resumes.

In addition to the use of clock blanking function to prevent a brown-out reset, the system is also expected to maintain in the current load and no further load change is allowed for 16 µs (max). Detailed description of the core voltage behaviors during load change are described in V_{DDC} Response During Load Change .

As described in V_{DDC} Response During Load Change , the status bit CLKCR.VDDC2LOW is used to indicate to user that the core voltage, V_{DDC}, is below the nominal voltage and EVR is in the process of regulating it. During this period, the clock could be also not running in the selected speed and it is recommended to poll this bit before continuing with any large change to the current load.

***Note**: It is recommended to slow down the PCLK and MCLK before entering deep sleep mode to prevent a sudden load change that could cause a brownout reset when entering.*

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 45 and Figure 46 show the recommended external circuits for both operating modes, External Crystal Mode and External Input Clock Mode.

In external input clock mode, an external clock signal is supplied directly without using an external crystal and bypassing the amplifier of the oscillator. The input frequency must be in the range from 4 to 48 MHz. When using an external clock signal it must be connected to XTAL1. XTAL2 is left open (unconnected).

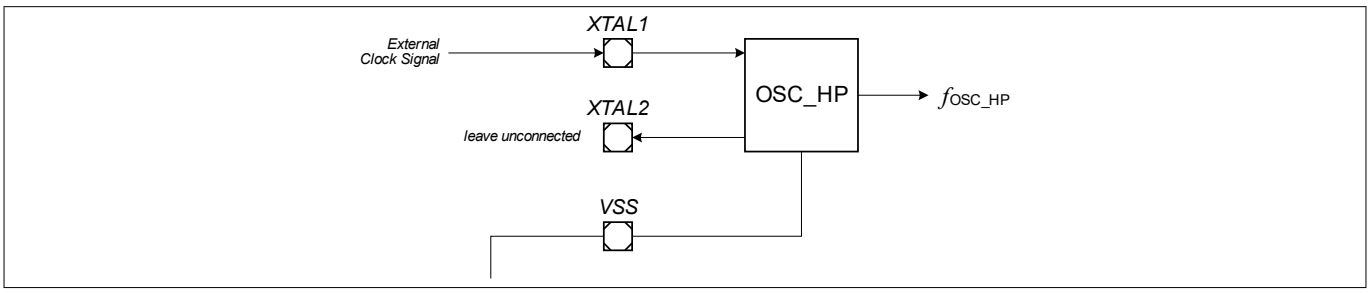


Figure 45 External Clock Input Mode for the High-Precision Oscillator

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.

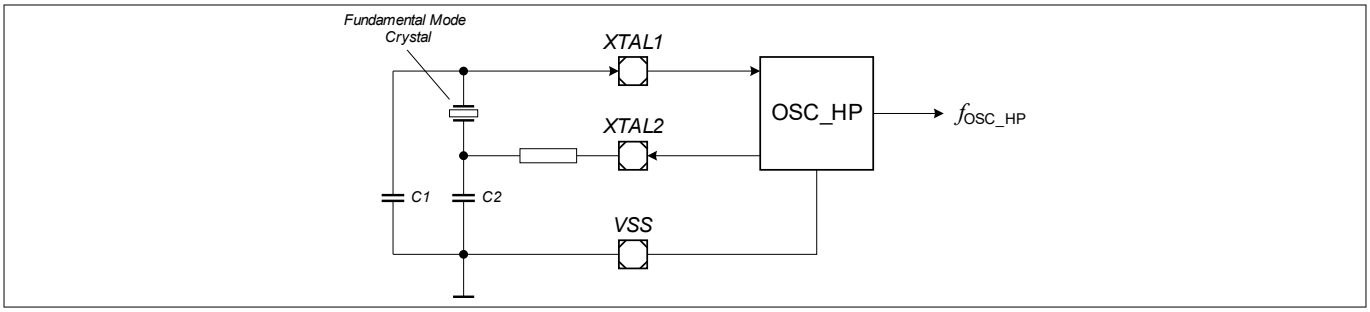


Figure 46 External Crystal Mode Circuitry for the High-Precision Oscillator

MultiCAN+ Clock

Figure 47 show the clock selection via the MultiCAN+ register bit CAN_MCR.CLKSEL. Refer to Controller Area Network Controller (MultiCAN+) for detailed description.

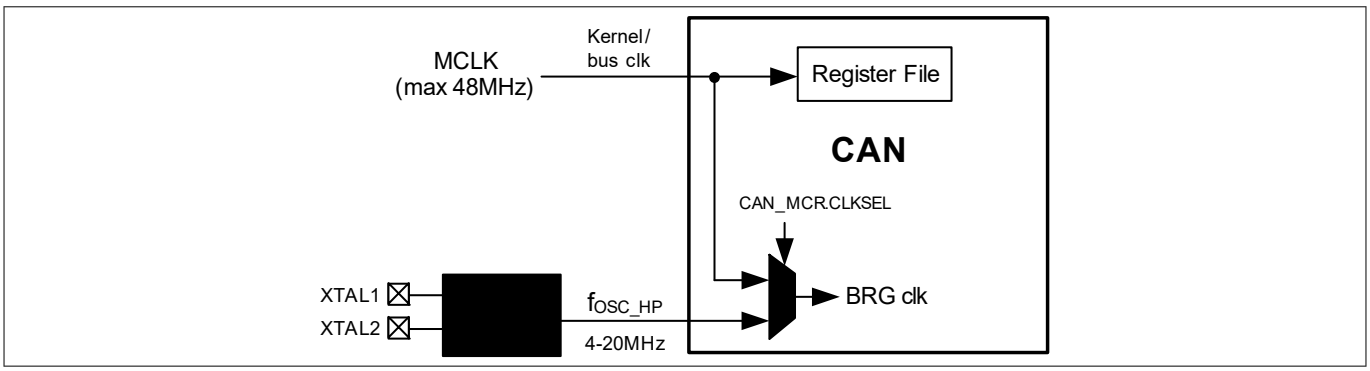


Figure 47 MultiCAN+ Clock

***Note**: Before changing MutliCAN+ clock source, the CAN clock must be gated using bit CGATSET0.MCAN0.*

***Note**: OSC_HP must be enabled (ANAOSCHPCTR.MODE) before changing the MutliCAN+ clock source.*

14.5.2.1 DCO1 Oscillator Watchdog

The oscillator watchdog (OWD) monitors the DCO1 clock frequency using the standby clock as the reference clock. It can be disabled via OSCCSR.OWDEN. By setting bit OSCCSR.OWDRES[3], the detection for DCO1 clock frequency can be restarted. The detection status output is only valid after some cycles of the standby clock frequency. When the OWD is disabled, the detection status will be reset and no detection is possible.

If the OWD is enabled before entering deep sleep mode, it will be disabled automatically by hardware when it enters deep sleep mode and re-enabled again after exiting deep sleep mode. The detection is reset and restarted after device wake-up from deep sleep mode.

14.5.2.2 Loss of DCO1 Clock Detection and Recovery

Loss of DCO1 clock happens when the oscillator watchdog (OWD) detects a DCO1 frequency that is less than 75 MHz or more than 105 MHz during normal operation. In this case, an interrupt will be generated if it is enabled. Concurrently, the oscillator status flag, OSCCSR.OSC2L or OSCCSR.OSC2H, is set to '1' and the system clock will be provided by the standby clock of 32 kHz if DCO1 is selected for the generation of PCLK/MCLK. Emergency routines can be executed to safely shut down the system. Beside triggering an interrupt when the loss of DCO1 clock happens, a system reset can also be triggered if it is enabled by RSTCON.LOCRSTEN.

***Note**: Switching to standby clock during loss of DCO1 clock event happens only if DCO2 is still running (>0 MHz). Bit SRRAW.SBYCLKFI indicates the fail status of the standby clock.*

The PSC1Mx remains in this loss of clock state until the next reset or after a successful clock recovery has been performed. A clock recovery could be carried out by restarting the detection by setting bit OSCCSR.OWDRES. Upon detecting a stable oscillator frequency of more than and lower than , OSC2L and OSC2H will be set to '0' and the MCLK/PCLK will switched automatically to the DCO1 clock source.

14.5.2.3 Standby Clock Failure

The standby clock failure event happens when the OWD detects a failure in standby clock where the clock stops running (~0 kHz). Bit SRRAW.SBYCLKFI is set to 1 and triggers an interrupt via SCU_SR1 service request if the interrupt is enabled in register SRMSK.

14.5.2.4 XTAL Oscillator Watchdog

The XTAL oscillator watchdog (XOWD) monitors the external OSC_HP clock frequency using the standby clock as the reference clock. It can be disabled via OSCCSR.XOWDEN. By setting bit OSCCSR.XOWDRES[4], the detection for the external OSC_HP clock frequency can be restarted. The detection status output is only valid after some cycles of the standby clock frequency. When the XOWD is disabled, the detection status will be reset and no detection is possible.

If the XOWD is enabled before entering deep sleep mode, it will be disabled automatically by hardware when it enters deep sleep mode and re-enabled again after exiting deep sleep mode. The detection is reset and restarted after device wake-up from deep sleep mode.

[3] It is recommended to clear the status bit SRRAW1.LOECI and SRRAW.SBYCLKFI before restarting the detection.
[4] It is recommended to clear the status bit SRRAW.LOCI and SRRAW.SBYCLKFI before restarting the detection.

***Note**: When OSC_HP is disabled in deep sleep mode, it is recommended to disable XOWD first before entering deep sleep to avoid a false detection after device wake-up.*

14.5.2.5 Loss of external OSC_HP Clock Detection and Recovery

Loss of external (OSC_HP) clock happens when the XTAL oscillator watchdog (XOWD) detects a failure in the external XTAL clock where the clock stops running (~0 kHz). In this case, bit SRRAW1.LOECI is set to 1 and triggers an interrupt via SCU_SR1 service request if the interrupt is enabled in register SRMSK1. Concurrently, the system clock will be provided by the standby clock of 32 kHz if external clock is used to generate the PCLK/MCLK. Emergency routines can be executed to safely shut down the system. Beside triggering an interrupt when the loss of external clock happens, a system reset can also be triggered if it is enabled by RSTCON.LOECRSTEN.

***Note**: Switching to standby clock during loss of external clock event happens only if DCO2 is still running (>0 MHz). Bit SRRAW.SBYCLKFI indicates the fail status of the standby clock.*

The PSC1Mx remains in this loss of clock state until the next reset or after a successful clock recovery has been performed. A clock recovery could be carried out by restarting the detection by setting bit OSCCSR.XOWDRES. Upon detecting a stable external clock frequency, MCLK/PCLK will switched automatically to the external clock source.

14.5.2.6 Startup Control for System Clock

When the PSC1Mx starts up after reset, system frequency is provided by the DCO1 oscillator. After reset, CPU runs in 8 MHz, default frequency. User can change the clock frequency that is used to execute the SSW and the user application code by defining it in the flash memory location 1000 1010_H. This feature allows the flexibility of device performance, for example, speed versus power consumption optimization during start-up. For details, please refer to the section Clock system handling by SSW.

14.5.2.7 DCLK input - External Clock via OSC_HP

The following sequence shall be used to select the external clock source via OSC_HP as the DCLK input.

- Enable OSC_HP with shaper not bypassed
  - set ANAOSCHPCTRL.MODE to 11_B
  - set ANAOSCHPCTRL.GAIN to
- Wait 10 ms for the external clock to stabilise
- Enable and reset XOWD
  - set OSCCSR.XOWDEN to 1_B
  - set OSCCSR.XOWDRES to 1_B
- Select external clock source as DCLK input (set bit CLKCR1.DCLKSEL)

14.5.3 Clock Gating Control

The clock to peripherals can be individually gated and parts of the system can be stopped by registers CGATSET0. After a master reset, only core, memories, SCU and PORT peripheral are not clock gated. The rest of the peripherals are default clock gated. User can select the clock of individual modules to be enabled by SSW after reset by defining it in the flash memory location 1000 1014_H. Refer to Boot and Startup for more details.

Load change during module clock enabling or gating

Enabling or gating the clock to peripherals could result in a load change that could cause clock blanking to happen. In addition, a load change of more than 4 times the current load would required the system to

maintain in the current load and no further load change is allowed for 16 µs (max). See V_{DDC} Response During Load Change for more details.

Module clock gating in Sleep and Deep-Sleep modes

It is recommended to gate the clock using registers CGATSET0 for module that is not needed during sleep mode or deep-sleep mode. These modules must be disabled before entering sleep or deep-sleep mode. In addition, the PCLK and MCLK will be switched to a slow standby clock and DCO1 will be put into power-down mode in deep-sleep mode.

14.5.4 Calibrating DCO1 based on Temperature

In PSC1Mx, DCO1 clock frequency can be calibrated during runtime to achieve a better accuracy. Based on the measured temperature using the on-chip temperature sensor (DTS), an offset value can be obtained using the formula as shown below:

$$\mathrm{OFFSET[steps]} = b + \frac{(a-b)(c-d)}{(e-d)} \tag{12}$$

where :

- OFFSET value ranges from 0 to 255
- c is the measured temperature [°C]
- a is constant DCO_ADJLO_T2
- b is constant DCO_ADJLO_T1
- d is constant ANA_TSE_T1
- e is constant ANA_TSE_T2

The four constants in the formula are stored in flash configuration page as shown in Table 64.

Table 64 DCO calibration data in Flash sector 0 (CS0)

Address	Length	Function
DCO calibration data:		
$1000'0F30_H$	1 B	ANA_TSE_T1
$1000'0F31_H$	1 B	ANA_TSE_T2
$1000'0F32_H$	1 B	DCO_ADJLO_T1
$1000'0F33_H$	1 B	DCO_ADJLO_T2

Input the offset value (rounded to the nearest integer) to register bit ANAOFFSET.ADJL_OFFSET to start the DCO1 calibration. The offset value must be within the range of 0 to 255. This bit field is bit protected. DCO1 will take about 5 µs (max) for clock to be adjusted to the new frequency.

An example code is provided in the Oscillator Handling Device Guide.

***Note**: Ensure bit ANASYNC1.SYNC_DCO_EN is set to 0 to disable the calibration via an external clock source so that the calibration via DTS can be carried out.*

14.5.5 Automatic DCO1 Calibration based on External Reference

In PSC1Mx, DCO1 clock frequency can be calibrated automatically during runtime to achieve better accuracy. To achieve an accuracy of < +/- 0.3%, calibration performed by a synchronization unit using high-precision clock as

reference is implemented. The high precision clock source for synchronization would be either one of the following and is selectable via bit ANASYNC1.XTAL_SEL:

- OSC_HP via XTAL1 and XTAL2 (4 - 20 MHz)
- OSC_LP via RTC_XTAL1 and RTC_XTAL2 (32.768 kHz)

Figure 48 shows the overview of the calibration scheme in PSC1Mx. It is enabled via bit ANASYNC1.SYNC_DCO_EN. ANAOFFSET.ADJL_OFFSET is taken as the initial DCO1 adjust value once the synchronization unit is enabled.

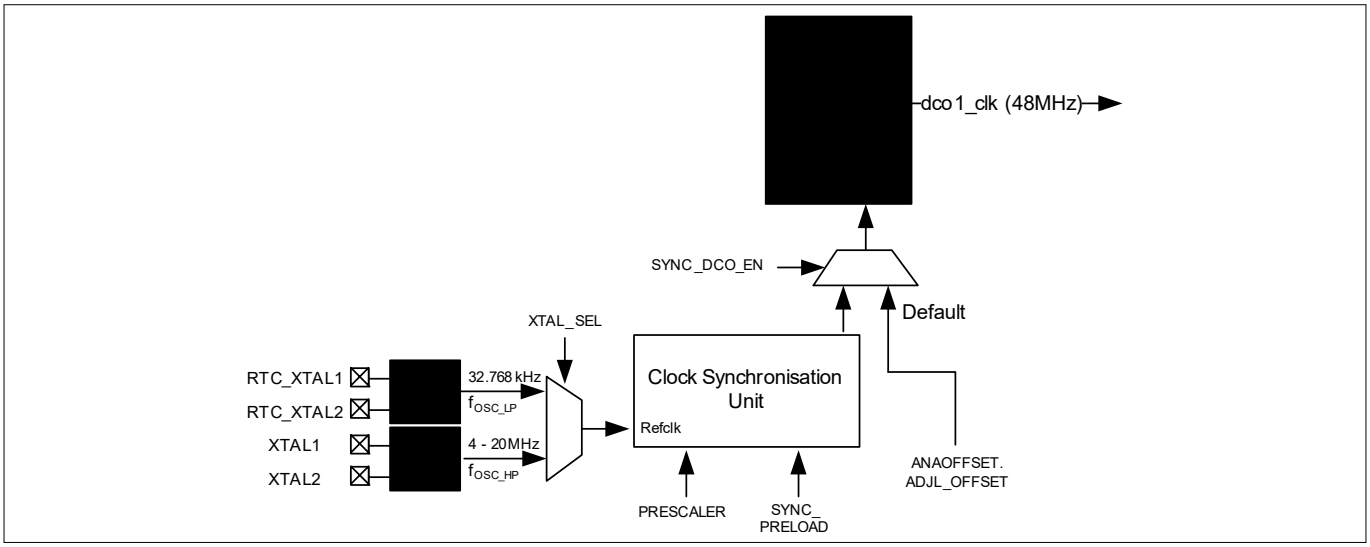


Figure 48 DCO1 Calibration

***Note**: The external clock must be enabled and stablised before selecting the automatic DCO1 calibration based on XTAL.*

***Note**: DCO1 is shut down when device enter Deep sleep mode. Hence, before entering Deep sleep mode, it is recommended to disable the automatic DCO1 calibration and shut down the XTAL function. If it is enabled, the DCO1 calibration will be disabled automatically during Deep sleep mode and resume after exiting this mode. Under this situation, the calibration values to ADJL inputs will be kept as the last values when the operation resume.*

Before enabling the DCO1 calibration, 2 values are needed to be programmed into bit ANASYNC2.PRESCALER and ANASYNC1.SYN_PRELOAD in order to generate a MCLK/PCLK frequency which is close to its target frequency. These parameters need to be setup once depending on the selected type of external clock source.

To trigger a reload of ANASYNC1.SYNC_PRELOAD and ANASYNC2.PRESCALER to the synchronization unit, write access to one of following:

- write access to register ANASYNC1
- write access to register ANASYNC2

Using OSC_LP as Reference for Calibration

Table 65 shows the recommended values if OSC_LP is used as the reference clock.

Table 65 Recommended values for a 32.768kHz clock via OSC_LP

Register name	HEX value
ANASYNC2.PRESCALER	6
ANASYNC1.SYNC_PRELOAD	2256

Using OSC_HP as Reference for Calibration

When OSC_HP is used, according to the wide range of selectable f_{OSC_HP} frequencies (such as, 4 - 20 MHz crystal oscillator), these two values have to be calculated once for a defined and constant crystal frequency based on the formula.

$$\text{PRESCALER} = \text{integer}\left(\frac{3000 \times f_{OSC_HP}[\text{MHz}]}{48}\right) \quad (13)$$

$$\text{SYN_PRELOAD} = \text{integer}\left(\frac{48 \times \text{PRESCALER}}{f_{OSC_HP}\,[\text{MHz}]}\right) \quad (14)$$

Table 66 shows some examples of PRESCALER and SYNC_PRELOAD values based on the frequency of the selected oscillator, f_{OSC_HP}.

Table 66 Examples of PRESCALER and SYNC_PRELOAD values

f_{OSC_HP}	ANASYNC2.PRESCALER	ANASYNC1.SYNC_PRELOAD
4	250	3000
8	500	3000
16	1000	3000
20	1250	3000

DCO1 Out of Synchronisation

The status of synchronisation can be monitored via bit ANASYN2.SYNC_READY. When the DCO1 synchronisation is not within the defined range, a DCO1 out of synchronisation (DCO1OFS) event occurs. An interrupt can be triggered via SCU SR1 request source during this event when is it enabled via the enable bit, DCO1OFSI in the SRMSK1 register.

As a precondition, the DCO1 frequency has to be synchronized once (by polling bit syn_ready in ANASYNC2 until it gets set). After that, it is then recommend to enable this interrupt if needed to detect the DCO1 out of synchronisation event. Status bit has to be cleared by using a bit in SRCLR1 register.

14.6 Service Request Generation

The SCU module provides 3 service request outputs SR[2:0]. SR0 is for system critical request, such as loss of clock event. SR1 is for the common SCU request such as DTS request. SR2 is for ACMP and ORC requests. The service request outputs SR[2:0] are connected to interrupt nodes in the Nested Vectored Interrupt Controller (NVIC).

Refer to Service Requests Handling for more details.

14.7 Debug Behavior

The SCU module does not get affected with the HALTED signal from SCU upon debug activities performed using external debug probe.

14.8 Power, Reset and Clock

The SCU module implements functions that involve various types of modules controlled directly or via dedicated interfaces that are instantiated in different power, clock and reset domains. These modules are

functionally considered parts of the SCU and therefore SCU is also considered a multi domain circuit in this sense.

Power domains:

Power domains get separated with appropriate power separation cells.

- Core domain supplied with V_{DDC} voltage
- Pad domain supplied with V_{DDP} voltage

Clock domains:

All cross-domain interfaces implement signal synchronization.

- internal SCU clock is MCLK, always identical to the CPU clock
- RTC and register mirror interface clock is 32.786 kHz clock generated from DCO2 oscillator

Reset domains:

All resets get internally synchronized to respective clocks.

- System Reset ($\overline{\text{SYSRESET}}$) resets most of the logics in SCU and can be triggered from various sources (refer to Reset Control (RCU) section for more details)
- Master Reset ($\overline{\text{MRESET}}$) contributes in generation of the System Reset and gets triggered upon power-up sequence of the Core domain

14.9 Registers

This section describes the registers of SCU which some of them resides in the ANACTRL module. Most of the registers are reset by $\overline{\text{SYSRESET}}$ reset signal but some of the registers can be reset only with power-on reset. SCU registers are accessible via the AHB-lite bus. ANACTRL registers are accessible via the APB bus. ANACTRL registers have name starting with “ANA”.

Table 67 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
CCU Registers	Offset address of Clock Control Unit	0300_H
RCU Registers	Offset address of Reset Control Unit	0400_H
RTC Registers	Offset address of Real Time Clock Module	$0A00_H$
ANACTRL Registers	Offset address of ANACTRL registers	1000_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 to SCU/ANACTRL SFRs result in an AHB/APB error response:

- Read or write access to undefined address
- Write access to read-only registers
- Write access to startup protected registers

Table 68 Registers Address Space

Module	Base Address	End Address	Note
SCU	$4001\ 0000_H$	$4001\ FFFF_H$	System Control Unit Registers

Table 69 Registers Overview

Register Name	Description	Offset Addr.[1]	Access Mode	
			Read	Write
PCU Registers (ANACTRL)				
ANAVDEL	Voltage Detector Control register	0050_H	U, PV	U, PV
PCU Registers (SCU)				
VDESR	Voltage Detector Status Register	0000_H	U, PV	U, PV
CCU Registers (SCU)				
CLKCR	Clock Control Register	0000_H	U, PV	U, PV, BP
CLKCR1	Clock Control Register 1	$001C_H$	U, PV	U, PV, BP
PWRSVCR	Power Save Control Register	0004_H	U, PV	U, PV
CGATSTAT0	Clock Gating Status for Peripherals 0	0008_H	U, PV	U, PV
CGATSET0	Clock Gating Set for Peripherals 0	$000C_H$	U, PV	U, PV, BP

(table continues...)

Table 69 (continued) Registers Overview

Register Name	Description	Offset Addr.[1)]	Access Mode	
			Read	Write
CGATCLR0/	Clock Gating Clear for Peripherals 0	0010_H	U, PV	U, PV, BP
OSCCSR	Oscillator Control and Status Register	0014_H	U, PV	U, PV
CCU Registers (ANACTRL)				
ANAOFFSET	DCO1 Offset Register	$106C_H$	U, PV	U, PV, BP
ANASYNC1	DCO1 Sync Control Register 1	1078_H	U, PV	U,PV
ANASYNC2	DCO1 Sync Control Register 2	$107C_H$	U, PV	U,PV
ANAOSCLPCTRL	OSC_LP Control Register	$108C_H$	U, PV	U,PV
ANAOSCHPCTRL	OSC_HP Control Register	1090_H	U, PV	U,PV.BP
RCU Registers (SCU)				
RSTSTAT	Reset Status	0000_H	U, PV	BE
RSTSET	Reset Set Register	0004_H	U, PV	U, PV
RSTCLR	Reset Clear Register	0008_H	U, PV	U, PV
RSTCON	Reset Control Register	$000C_H$	U, PV	U, PV
GCU Registers (SCU)				
ID	Module Identification Register	0008_H	U, PV	BE
IDCHIP	Chip ID	0004_H	U, PV	U, PV
DBGROMID	DBGROMID	0000_H	U, PV	U, PV
SSW0	SSW Support Register	0014_H	U, PV	U, PV
CCUCON	CCUx Global Start Control Register	0030_H	U, PV	U, PV
SRRAW	RAW Service Request Status	0038_H	U, PV	BE
SRMSK	Service Request Mask	$003C_H$	U, PV	U, PV
SRCLR	Service Request Clear	0040_H	U, PV	U, PV
SRSET	Service Request Set	0044_H	U, PV	U, PV
SRRAW1	RAW Service Request Status 1	0058_H	U, PV	BE
SRMSK1	Service Request Mask 1	$005C_H$	U, PV	U, PV
SRCLR1	Service Request Clear 1	0060_H	U, PV	U, PV
SRSET1	Service Request Set 1	0064_H	U, PV	U, PV
PASSWD	Bit protection Register 1	0024_H	U, PV	U, PV
MIRRSTS	Mirror Update Status Register	0048_H	U, PV	BE
PMTSR	Parity Memory Test Select Register	0054_H	U, PV	U, PV
PFUCR	Prefetch Unit Control Register	0068_H	U, PV	U, PV
INTCR0	Interrupt Control Register 0	$006C_H$	U, PV	U, PV

(table continues...)

Table 69 (continued) Registers Overview

Register Name	Description	Offset Addr.[1]	Access Mode	
			Read	Write
INTCR1	Interrupt Control Register 1	0070_H	U, PV	U, PV
STSTAT	Startup Status Register	0074_H	U, PV	U, PV

1) The absolute register address is calculated as follows:
Module Base Address + Sub-Module Offset Address + Offset Address (shown in this column)

15 Pseudo Random Number Generator (PRNG)

15.1 Overview

The pseudo random bit generator (PRNG) provides random data with fast generation times.

15.1.1 Features

The PRNG includes the following features:

- Fast random data generation times:
  - 17 to 18 clock cycles generation time for 16-bit random data
  - 9 to 10 clock cycles generation time for 8-bit random data
- Fulfills statistical tests according to NIST-800

15.2 Description of Operation Modes

15.2.1 Key Loading Mode

Before the PRNG can be used it has to be initialized by the user software.

The key (seed) k of the PRNG is a bit string $k = (k_{n-1}, k_{n-2}, ..., k_2, k_1, k_0)$ of length n. A key length of 80 bits is recommended, although smaller or larger key lengths are possible. It is recommended to use chip individual seed values.

The initialization of the PRNG consists of two basic phases:

1. Key loading
2. Warm-up

Key loading is initialized by setting the bit PRNG_CTRL.KLD to '1'. In the key loading mode, PRNG_WORD acts always as a 16-bit destination register. The p partial words W_i (0 <= i < p) of the key k = (W_{p-1}, ..., W_1, W_0), with W_i = (k_{15}, ..., k_1, k_0), are sequentially written to PRNG_WORD in the order W_0, W_1, ..., W_{p-1}. A useful seed size is 80 bits (i.e., 5 data words, p=5). Because the bits of the partial key word are sequentially loaded into the internal state of the PRNG, loading of a key word will take 16 clock cycles. The PRNG_CHK.RDV flag is set to '0' while loading is in progress. A flag value of '1' indicates that the next partial key word can be written to PRNG_WORD.

After the complete key has been loaded, the PRNG_CTRL.KLD flag must be set to '0' to prepare the following warm-up phase. This operation takes one clock cycle.

The warm-up phase provides a thorough diffusion of the key bits. For this purpose, the user must read and discard 64 random bits from the register PRNG_WORD. The random data output block must be set to either b = 8 or 16 bits. This is achieved by setting the corresponding value of the PRNG_CTRL.RDBS field.

The flag PRNG_CHK.RDV set to '1' indicates that the next random data block of width 'b' can be read from PRNG_WORD.

If, for any reason, PRNG_CTRL.RDBS is reset to the default value of 00_B, the PRNG must be initialized once more – that is, a key must be loaded and a warm-up phase carried out.

15.2.2 Streaming Mode

The flag PRNG_CHK.RDV set to 1_B indicates that the next random data block can be read from PRNG_WORD. After a word has been read the flag PRNG_CHK.RDV is reset to 0B by the hardware and generation of new random bits starts. The PRNG requires 17– 18 clock cycles to generate a 16 random bits and 9–10 clock cycles to generate eight random bits. From a software point of view, it is not necessary to poll the PRNG_CHK.RDV flag. Consecutive read accesses to PRNG_WORD will be delayed automatically by hardware as long as PRNG_CHK.RDV is '0'.

The width of the output data block is changed by setting the value of PRNG_CTRL.RDBS. This should be done before entering streaming mode, otherwise if the change is made during streaming, the new setting will not come into effect until the next generation cycle is started. The hardware checks that the selected number of bits are available and the flag PRNG_CHK.RDV is set when this condition is true.

In order to avoid reading a duplicate random byte when switching from 8-bit to 16-bit operation mode, the last byte generated in the 8-bit mode should first be discarded before switching to 16-bit mode.

***Note**: PRNG_WORD should be accessed as 16-bit register, the upper 16 bits (of a 32 bit access) are ignored on a write and zeroes on a read and therefore contains no random data.*

15.2.3 Refreshing and Restarting a Random Bit Stream

The random bit sequence can be refreshed with a new key. In this case the entropy contained in the last internal state is not cleared, but instead the new key is mixed into the current PRNG state. This is referred to as refreshing.

Refreshing is a means of introducing additional entropy into the generation of the random bit sequence. Without refreshing, the entire entropy rests in the initial key (or seed) that was used for initializing the PRNG during first key loading. Thereafter, the generation of the random bit sequence is purely deterministic. The deterministic process is broken whenever refreshing is performed. Refreshing implies that it is not possible to reproduce the same random result using the same key for data generation.

Since the internal state of the PRNG cannot be read and set directly, a sequence cannot be restored from any given state. A random bit sequence based on a certain initial key can, however, be continued. To this means, a segment s of the output sequence is stored and this segment is used as a initial key later on. The segment s = (s_{n-1}, s_{n-2}, ..., s_2, s_1, s_0) of length **n** should be fresh – that is, generated after the last bits used in the application. The length of **s** should be at least n = 80 bits. This way a pseudorandom bit sequence can be continued after a system reset without requiring new key material. This process is known as restarting.

***Note**: Integration in a multi-application/multitasking environment together with the requirement to obtain a reproducible pseudorandom bit sequence would necessitate a state saving and restoring feature. Hence the OS must be able to save the PRNG state before giving control to application 2 and restore the state before returning control to application 1. This is not possible with the PRNG module.*

15.3 Service Request Generation

The PRNG does not generate any service request.

15.4 Debug Behavior

The PRNG does not support a suspend mode while the system is halted by the debugger. That means that the PRNG continues its operation during debug halt.

15.5 Power, Reset and Clock

The PRNG module is located in the core power domain. The module can be reset to its default state by a system reset.

The PRNG module is clocked by the main clock, MCLK, from SCU.

15.6 Initialization and System Dependencies

The PRNG is always available after reset.

Refer to Key Loading Mode for the initialization steps.

15.7 Registers

The interface of the PRNG comprises the registers PRNG_WORD, PRNG_CHK and PRNG_CTRL.

Table 70 Registers Address Space

Module	Base Address	End Address	Note
PRNG	$4802\ 0000_H$	$4802\ 000F_H$	

Table 71 Registers Overview

Register Name	Register Long Name	Offset Address
Data Registers		
PRNG_WORD	PRNG word register	00_H
PRNG_CHK	PRNG status check register	04_H
Control Registers		
PRNG_CTRL	PRNG control register	$0C_H$

The register is addressed wordwise.

16 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 (for example, 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

[7] IIC Bus Specification (Philips Semiconductors v2.1)

[8] IIS Bus Specification (Philips Semiconductors June 5 1996 revision)

Table 72 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 ***Note**: In the USIC chapter, the module clock is referred to as f_{PERIPH}*
PPP	Protocol Pre-Processor
RSR	Receive Shift Register
SCLK	Shift Clock
TSR	Transmit Shift Register

16.1 Overview

This section gives an overview about the feature set of the USIC.

16.1.1 Features

Each USIC channel can be individually configured to match the application needs, for example, 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 Datasheet, 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 (for example, 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, for example, 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, for example, to synchronize several communication partners.

- **Transfer trigger capability**

  In master mode, data transfers can be triggered by events generated outside the USIC module, for example, 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 (for example, for collision detection in ASC mode, for IIC, etc.).*

A block diagram of the USIC module/channel structure is shown in Figure 49.

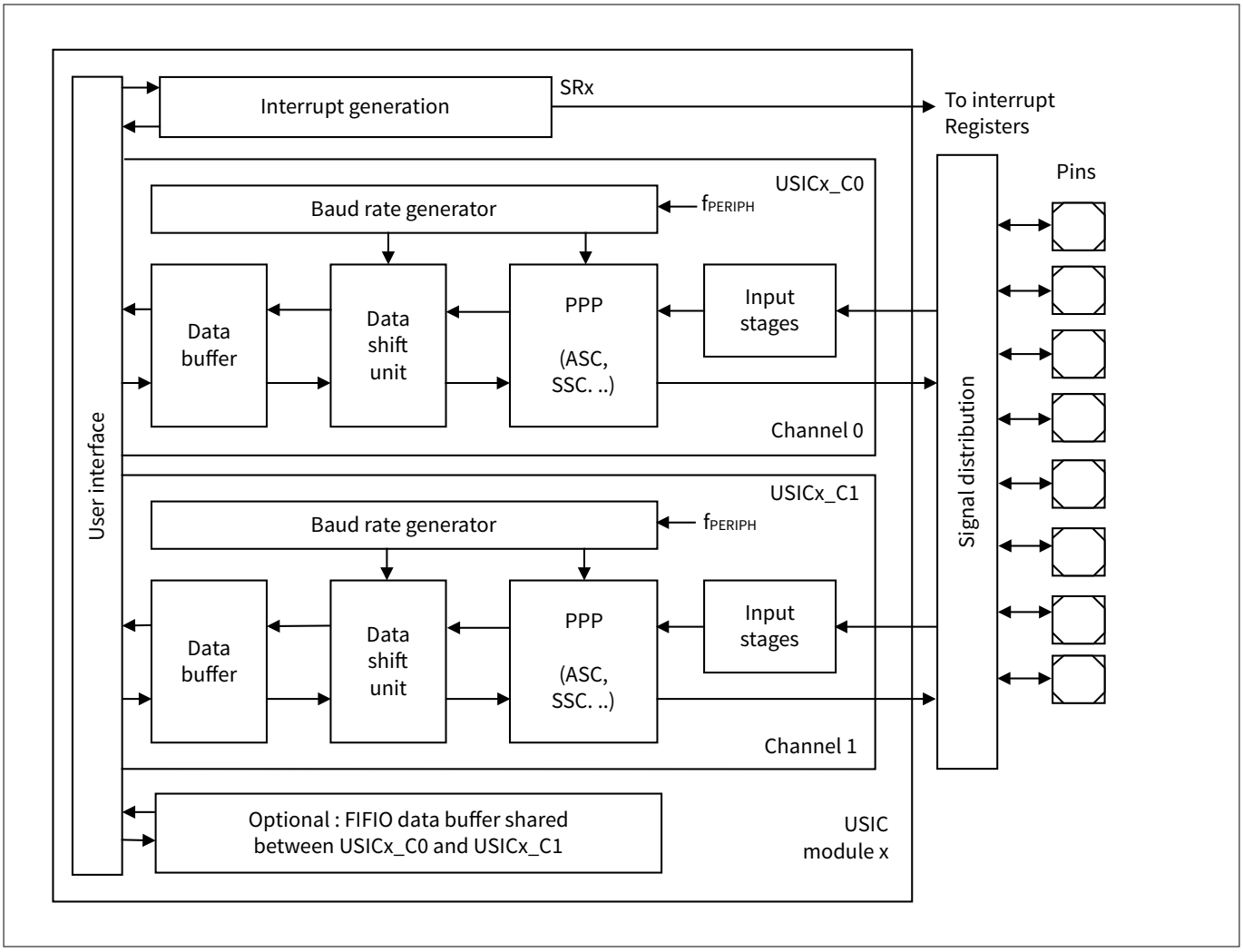


Figure 49 USIC Module/Channel Structure

16.2 Operating the USIC

This section describes how to operate the USIC communication channel.

16.2.1 USIC Structure Overview

This section introduces the USIC structure.

16.2.1.1 Channel Structure

The USIC module contains two independent communication channels, with a structure as shown in Figure 49.

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.

16.2.1.2 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 73.

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 73 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

(table continues...)

Table 73 (continued) 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)
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 Interconnects*.*

16.2.1.3 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 74.

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 (for example, 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 74 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

(table continues...)

Table 74 (continued) 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
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
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 Interconnects.

16.2.1.4 Baud Rate Generator

Each USIC Channel contains a baud rate generator structured as shown in Figure 50. 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, for example, 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 refer to the interconnects section.

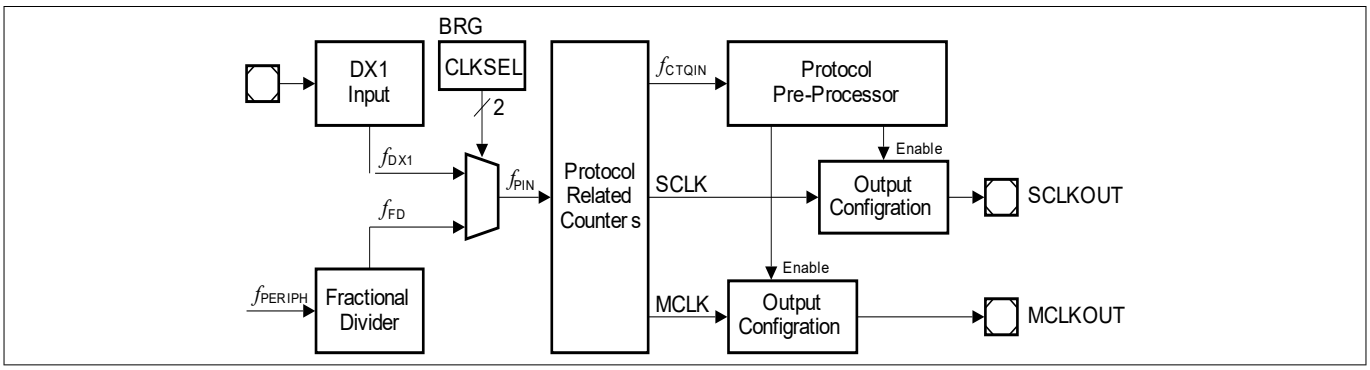


Figure 50 Baud Rate Generator

16.2.1.5 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.

16.2.1.6 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.

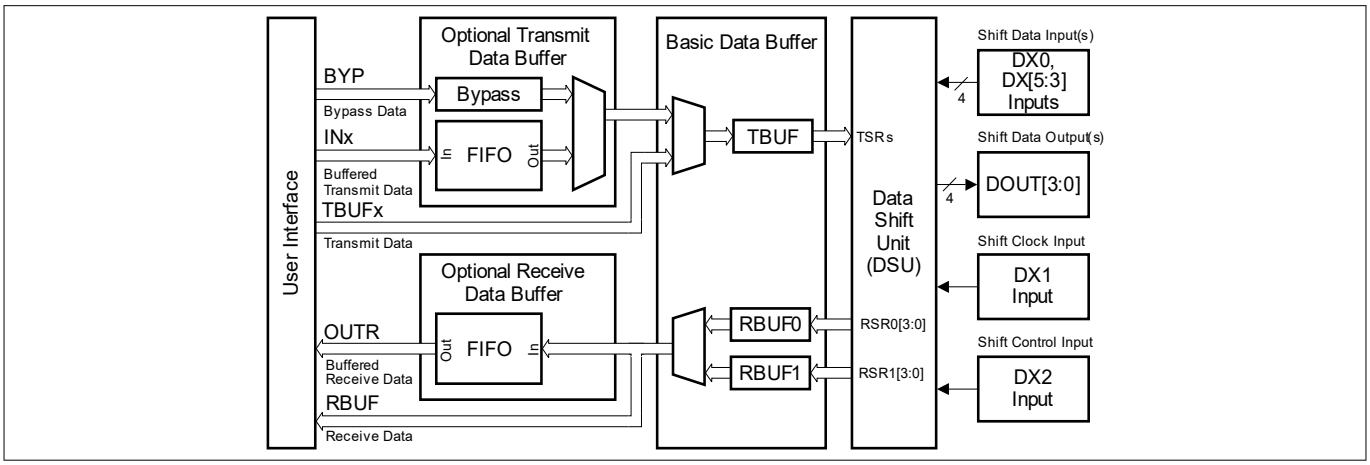


Figure 51 Principle of Data Buffering

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.

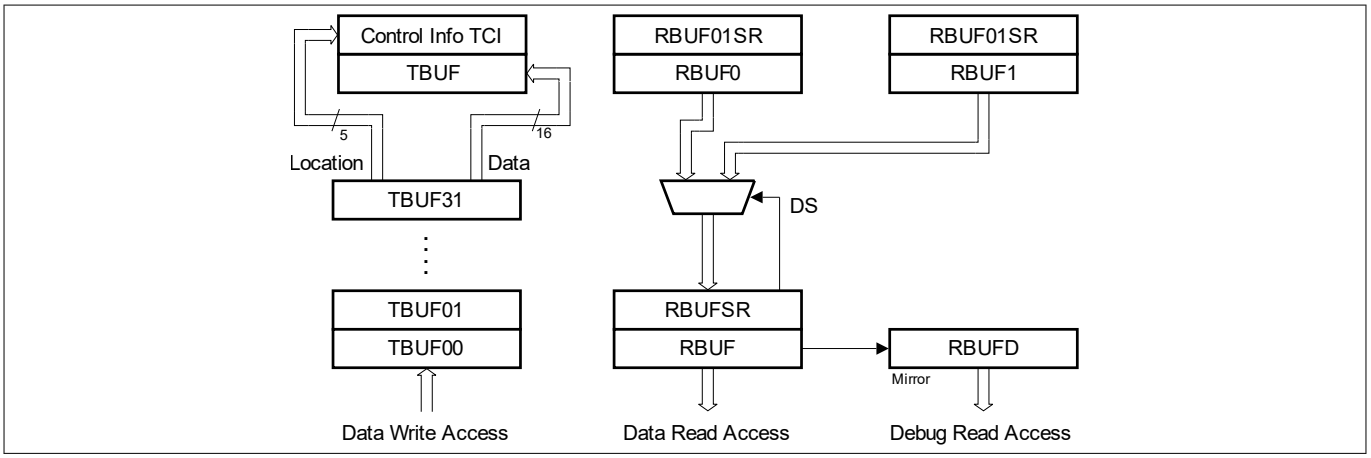


Figure 52 Data Access Structure without additional Data Buffer

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 Transmit Control Information 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 (for example, 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 (for example, to trigger data transfers by events).

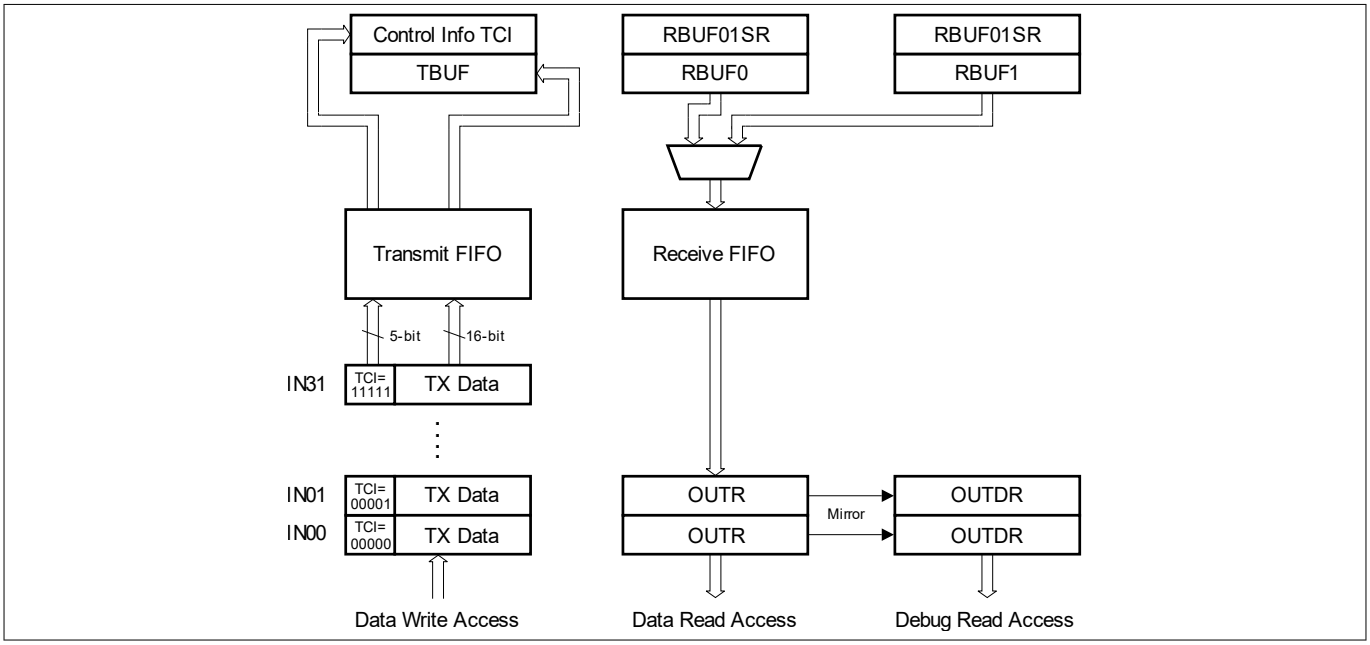


Figure 53 Data Access Structure with FIFO

16.2.2 Operating the USIC Communication Channel

This section describes how to operate a USIC communication channel, including protocol control and status, and mode control. 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 Mode Control).
- 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.
- Select a protocol by CCR.MODE.

16.2.2.1 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
- PCR for the SSC protocol
- PCR for the IIC protocol
- PCR for the IIS protocol

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
- PSR flags for the SSC protocol
- PSR flags for the IIC protocol
- PSR flags for the IIS protocol

16.2.2.2 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 75.

Table 75 USIC Communication Channel Behavior

Kernel Mode	Channel Behavior	KSCFG. NOMCFG /SUMCFG
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. The definition of each kernel mode may differ across protocols and is described in the respective protocol's section on mode control behaviour.

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.

***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.*

***Note**: In PSC1Mx, bit field KSCFG.SUMCFG is reset to its default value by any reset. If the suspend function is required during debugging, it is recommended that it is enabled in the user initialization code. Before programming the bit field, the module clock has to be enabled and special care needs to be taken while enabling the module clock as described in the CCU (Clock Gating Control) section of the SCU chapter.*

16.2.3 Operating the Input Stages

All input stages offer the same feature set, unless stated otherwise. 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.

16.2.3.1 General Input Structure

There are generally two types of input stages:

- One for the data input stages DX0, DX[5:3] (Figure 54)
- The other for non-data input stages DX[2:1] (Figure 55)

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.

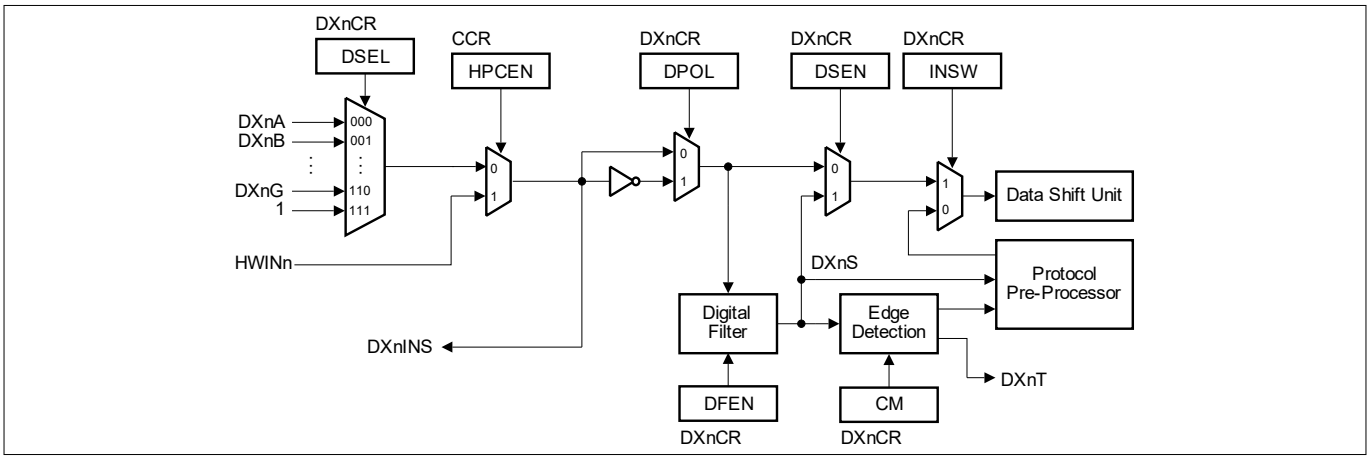


Figure 54 Input Conditioning for DX0 and DX[5:3]

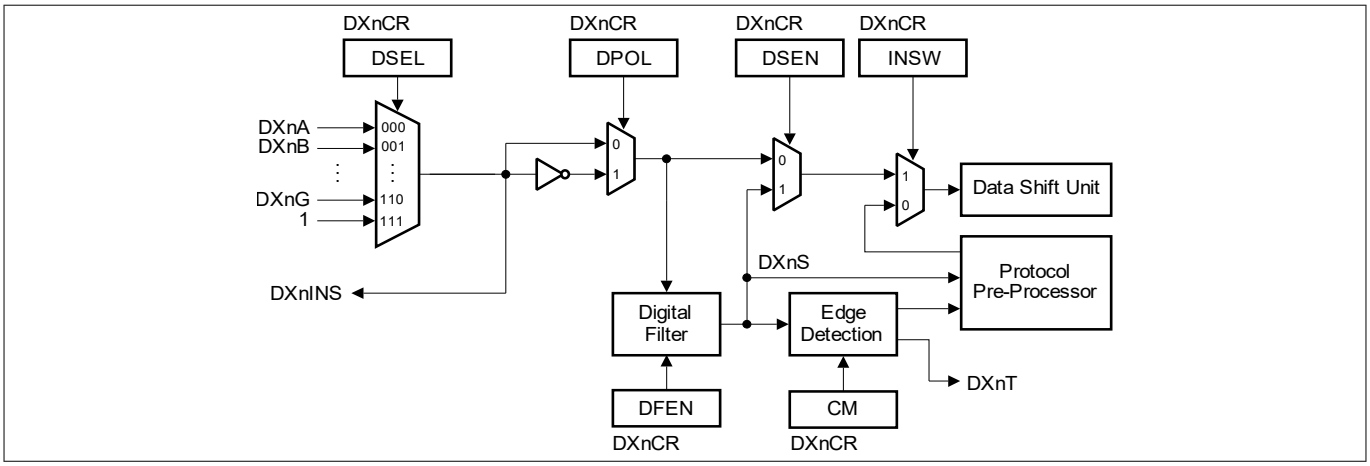


Figure 55 Input Conditioning for DX[2:1]

16.2.3.2 Input Selection

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).

Refer to the Interconnects section for the device-specific input signal assignment.

16.2.3.3 Input Conditioning

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}.

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.

16.2.3.4 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.

16.2.3.5 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, for example, 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).

16.2.3.6 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, for example, to generate interrupts.

16.2.3.7 Loop Back Mode

The USIC transmitter output signals can be connected to the receiver inputs of the same communication channel to form a loop back mode.

This can be done with an indirect connection through intermediate input stages (assuming these are not used by the application). For example, connecting the receiver input signal USIC0_CH0.DX0G to the transmitter output signal USIC0_CH0.DOUT0 through the DX3 input stage (DOUT0 -> DX3G -> DX3INS -> DX0G).

In the loop back mode, 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.

16.2.3.8 Delay Compensation in DX1

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, as shown in Figure 56.

- 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.

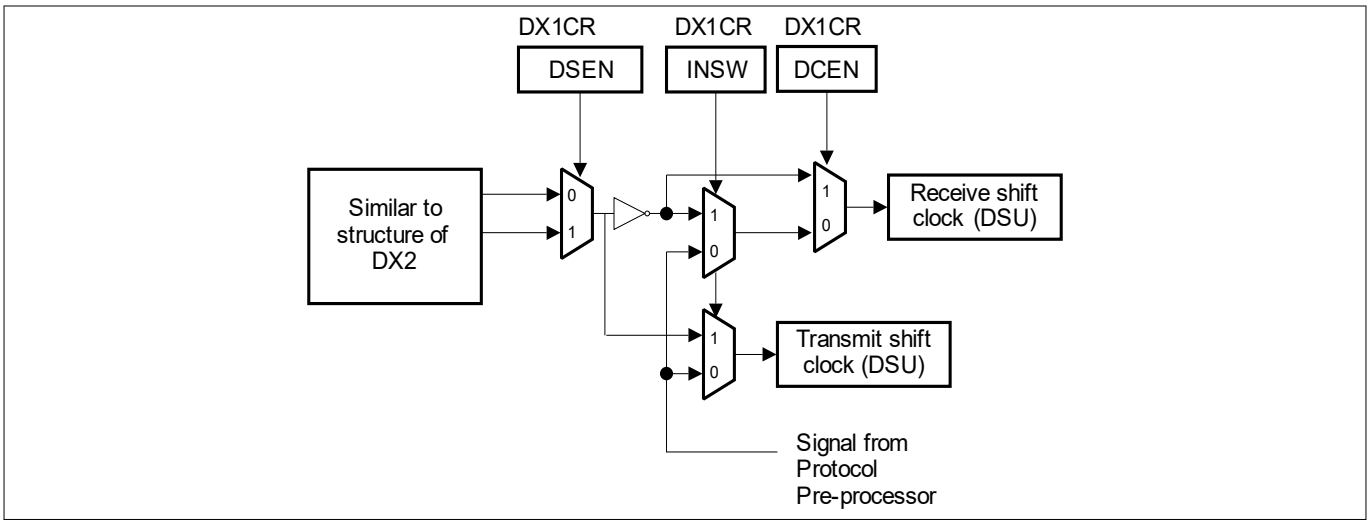


Figure 56 Delay Compensation Enable in DX1

16.2.4 Operating the Baud Rate Generator

The following blocks can be configured to operate the baud rate generator, see also Figure 50.

16.2.4.1 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:

$$f_{FD} = f_{PERIPH} \times \frac{1}{n} \quad \text{with } n = 1024 - STEP \tag{15}$$

- **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.
  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:

  ***Note**: Fractional divider mode f_{FD} can introduce a maximum period jitter of one f_{PERIPH} period. This jitter is not accumulated over several cycles.*

$$f_{FD} = f_{PERIPH} \times \frac{n}{1024} \quad \text{with } n = STEP \tag{16}$$

The output frequency f_{FD} of the fractional divider is selected for baud rate generation by BRG.CLKSEL = 00_B ($f_{PIN} = f_{FD}$).

16.2.4.2 External Frequency Input

The baud rate can be generated referring to an external frequency input (instead of to f_{PERIPH}) if the input stage DX1 is not needed (DX1CTR.INSW = 0) in the selected protocol.

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, for example, 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.

16.2.4.3 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.

$$f_{MCLK} = \frac{f_{PIN}}{2} \tag{17}$$

$$f_{SCLK} = \frac{f_{PDIV}}{2} \tag{18}$$

In the case that the master clock is used as reference for external devices (for example, 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 (PPPEN = 1). 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 (PPPEN = 0).

$$\begin{aligned} f_{PDIV} &= f_{PIN} \times \frac{1}{PDIV + 1} \quad \text{if PPPEN} = 0 \\ f_{PDIV} &= f_{MCLK} \times \frac{1}{PDIV + 1} \quad \text{if PPPEN} = 1 \end{aligned} \tag{19}$$

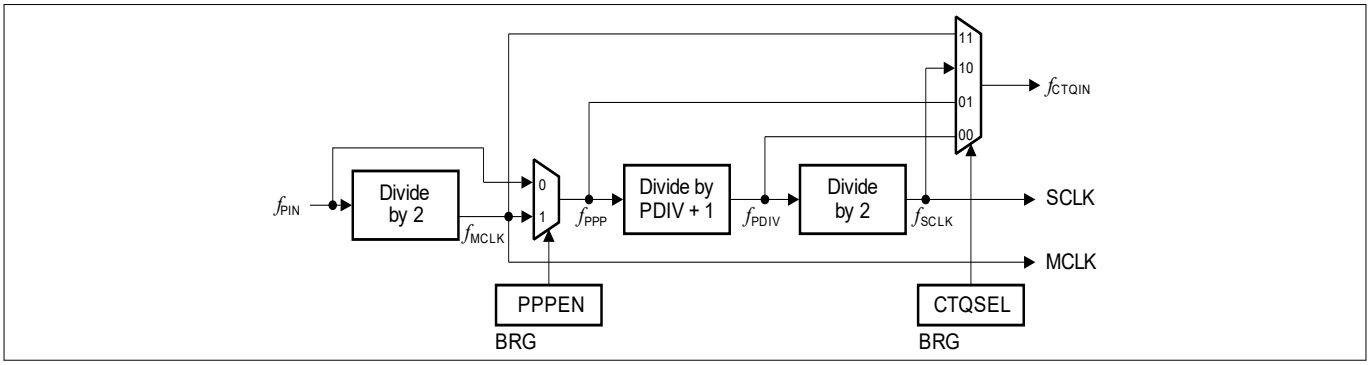


Figure 57 Divider Mode Counter

16.2.4.4 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).

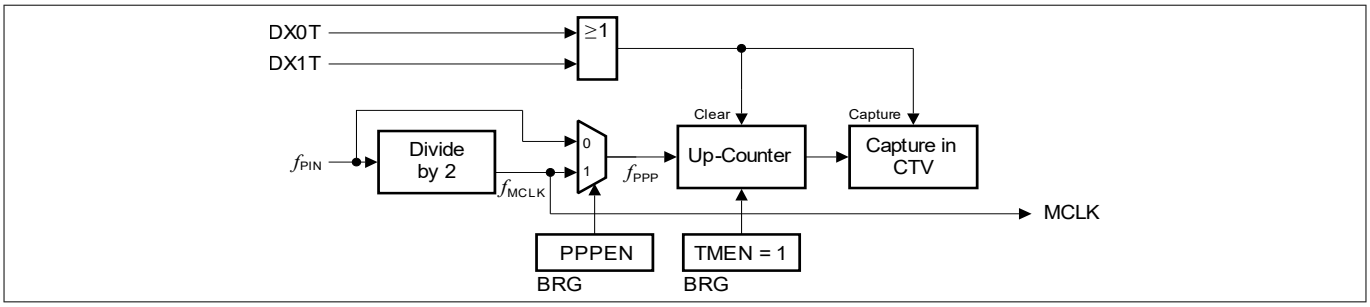


Figure 58 Protocol-Related Counter (Capture Mode)

The capture mode timer can be used to measure the baud rate in slave mode before starting or during data transfers, for example, 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.

16.2.4.5 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.

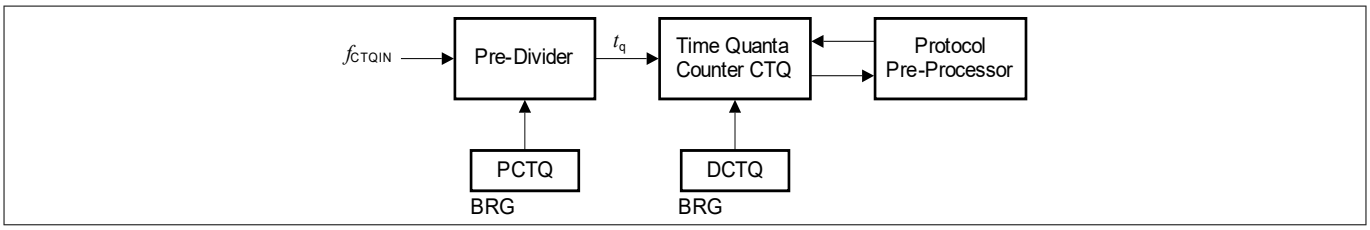


Figure 59 Time Quanta Counter

16.2.4.6 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 60, 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 preprocessor, 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.

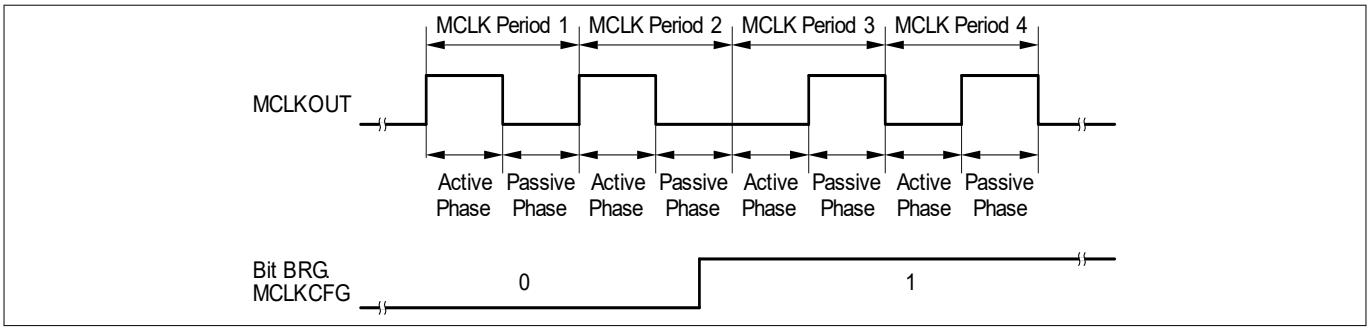


Figure 60 Master Clock Output Configuration

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, for example, for protocols without the need of a shift clock available at a pin, the SCLKOUT generation is disabled.

16.2.5 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.

16.2.5.1 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 Transmit Control Information) or, optionally, by a FIFO buffer stage (see Operating the FIFO Data Buffer).

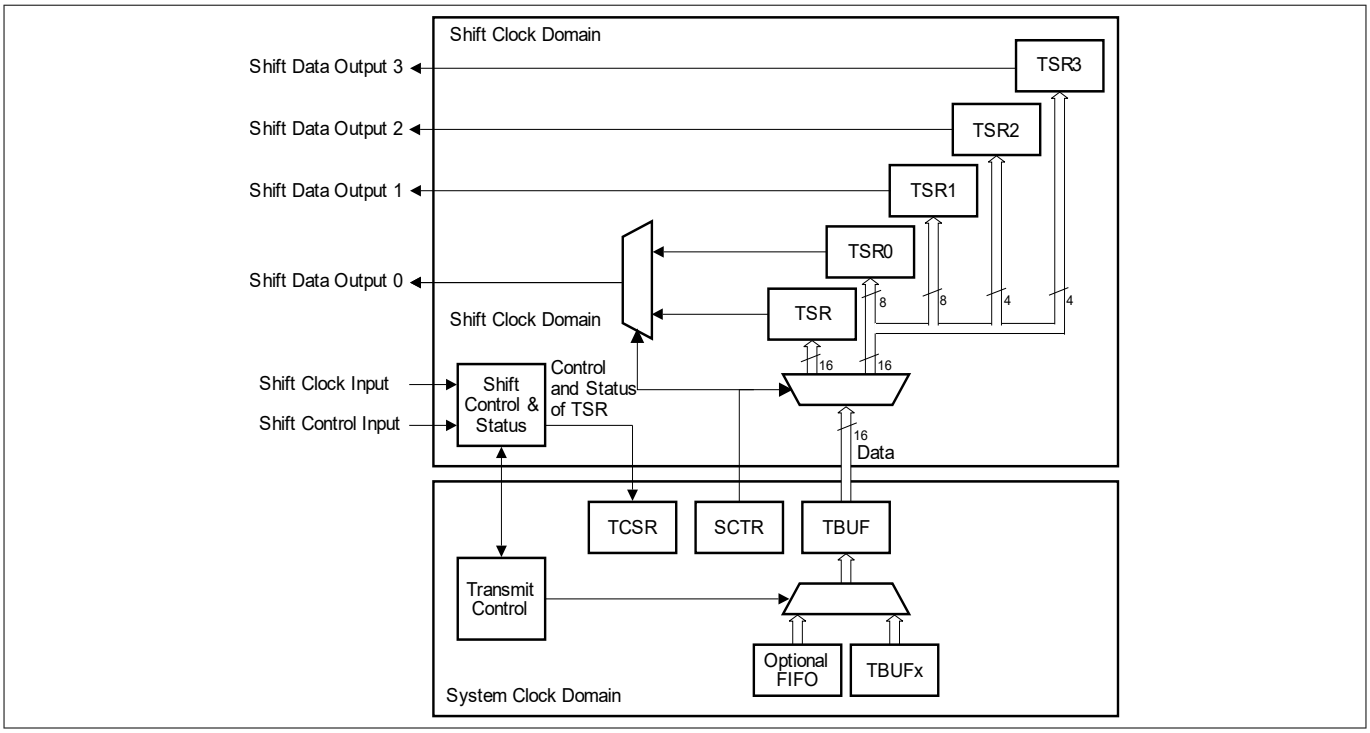


Figure 61 Transmit Data Path

16.2.5.2 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 76. 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 76 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

(table continues...)

Table 76 (continued) 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)
TSR3	Not used	Not used	Bit n*4 + 3

16.2.5.3 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, 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.

***Note**: There can be only one TCI mode enabled at any one time, i.e. one mode bit is programmed to 1 and all the rest to 0. Concurrent TCI modes are not supported.*

Table 77 TCI Modes

TCI Mode	Effects	Usage
Word length control (WLEMD = 1)	Bit field SCTR.WLE is updated with TCI[3:0] if a transmit buffer input location TBUFx is written	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]	Can be used in SSC master mode to control the slave select generation to finish data frames
Frame length control (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	Can be used in all protocols to dynamically change the data frame length between 1 and 32 data bits per data frame
Select output control (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	Can be used in SSC master mode to define the targeted slave device(s)
Word address control (WAMD = 1)	Bit TCSR.WA is updated with TCI[4] if a transmit buffer input location TBUFx is written	Can be used in IIS mode to define if the data word is transmitted on the right or the left channel
Hardware Port control (HPCMD = 1)	Bit field SCTR.DSM is updated with TCI[1:0] if a transmit buffer input location TBUFx is written	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

(table continues...)

Table 77 (continued) TCI Modes

TCI Mode	Effects	Usage
	Additionally, bit TCSR.HPCDIR is updated with TCI[2]	Can be used in SSC protocols to control the pin(s) direction when the hardware port control function is enabled through CCR.HPCEN = 1

16.2.5.4 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 62.

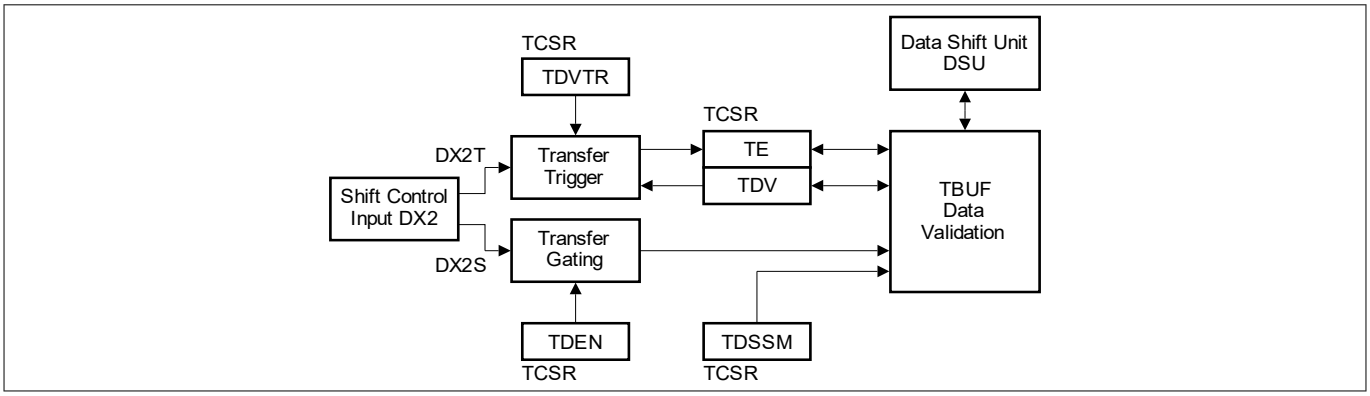


Figure 62 Transmit Data Validation

Transfer Gating

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.

Transfer Trigger

A transfer trigger logic supports data word transfers related to events, for example, 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. 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.

The transfer trigger logic is controlled by bit TCSR.TDVTR and the occurrence of a trigger event is indicated by bit TCSR.TE.

TBUF Data Validation

A data validation logic combines 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 (for example, 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 (for example, FIFO buffer). To avoid that data words are sent out several times or to allow data handling with an additional data buffer (for example, 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

16.2.6 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.

16.2.6.1 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 Operating the FIFO Data Buffer).

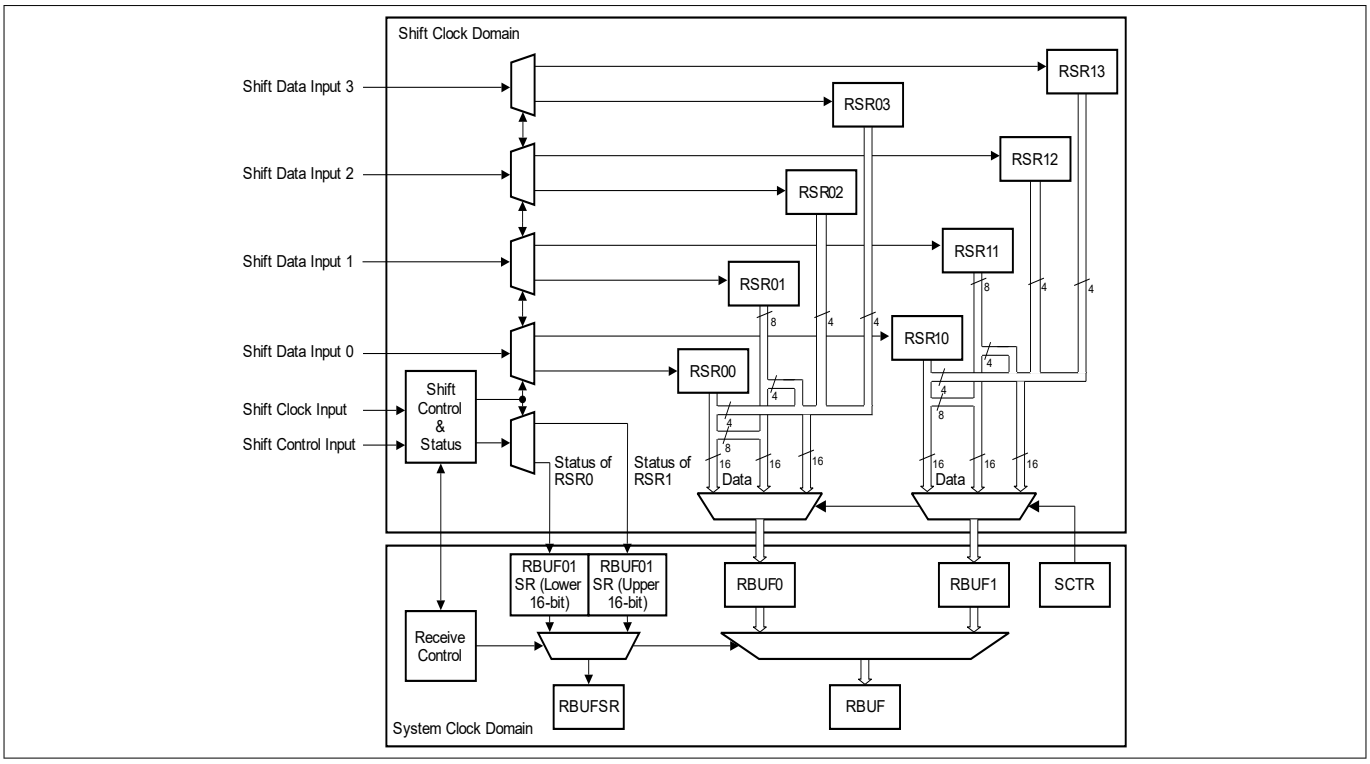


Figure 63 Receive Data Path

16.2.6.2 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 78. 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 78 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

16.2.6.3 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.

16.2.7 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.

***Note**: If the number of port pins enabled through CCR.HPCEN is less than the number requested through SCTR.DSM, the unused pins output the passive data level defined by SCTR.PDL.*

16.2.8 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.

Each USIC module has 64 buffer entries that can be distributed among the transmit or receive FIFO buffers of both channels of the module.

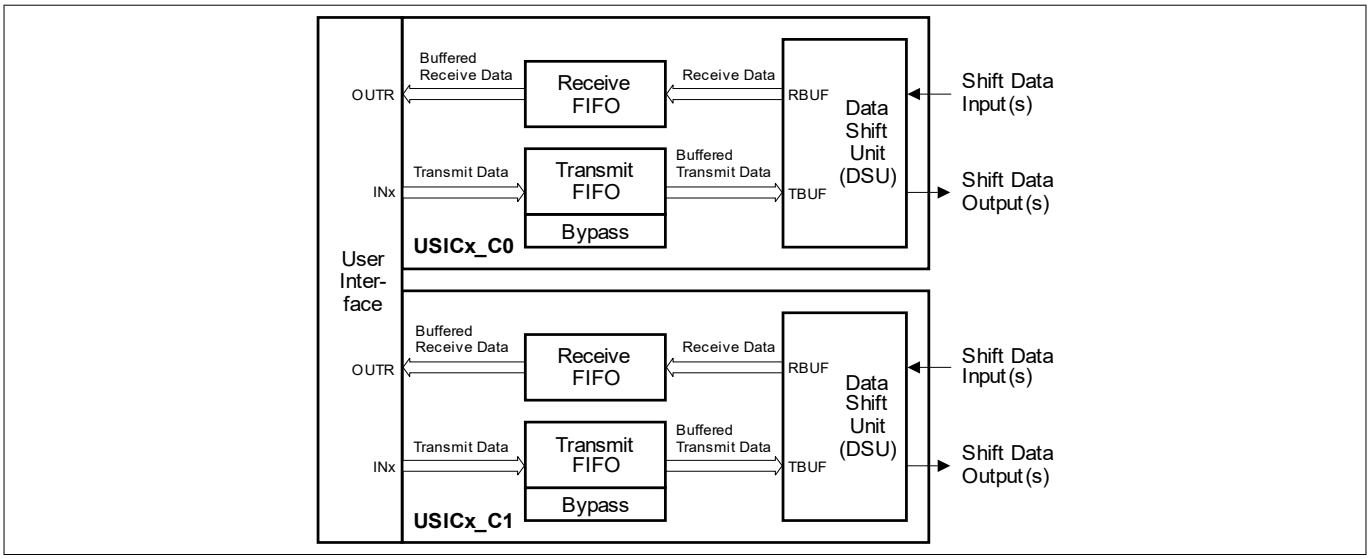


Figure 64 FIFO Buffer Overview

In order to operate the FIFO data buffers, the following issues have to be considered:

- FIFO buffer available and selected:

  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 FIFO Buffer Bypass.

16.2.8.1 FIFO Buffer Partitioning

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, 64 FIFO entries are 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 65.

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.

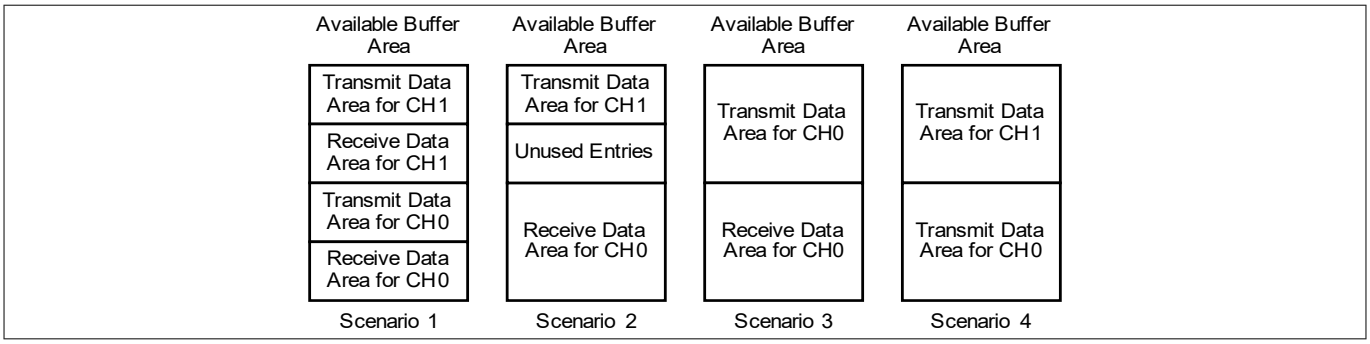


Figure 65 FIFO Buffer Partitioning

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.

16.2.8.2 Transmit FIFO Buffer Modes

The transmit FIFO buffer provides two operation modes with different triggering mechanisms for the standard transmit buffer events.

The modes are selectable through the bit field TBCTR.STBTEN.

STBTEN = 0 (Default)

When TBCTR.STBTEN = 0, a standard transmit buffer event is triggered only under the condition of the filling level of the transmit buffer (given by TRBSR.TBFLVL) transiting away from a programmed limit (TBCTR.LIMIT).

- When TBCTR.LOF = 0, the standard transmit buffer event is triggered by TBFLVL falling below LIMIT.
- When TBCTR.LOF = 1, the standard transmit buffer event is triggered by TBFLVL exceeding LIMIT.

The standard transmit buffer event is indicated by the flag STBI.

***Note**: The standard transmit buffer event indicated by TRBSR.STBI is triggered based on the transition of the fill level from equal to below or above the limit, not the fact of being below or above.*

STBTEN = 1

When TBCTR.STBTEN = 1, a standard transmit buffer event can be triggered in two ways:

1. Similar to the case where STBTEN = 0, with the difference that the flag TRBSR.STBT is also set in addition to STBI.
2. While STBT = 1, a standard transmit buffer event will be additionally triggered each time there is either a transfer data to TBUF event (for the case where LOF = 0) or write data to INx event (for the case where LOF = 1).

***Note**: Standard transmit buffer events triggered by TRBSR.STBT does not affect the TRBSR.STBI flag.*

The STBT flag is cleared depending on the setting of TBCTR.STBTM bit:

- When STBTM = 0, STBT is automatically cleared by hardware when the buffer fill level equals the programmed limit again (TRBSR.TBFLVL = TBCTR.LIMIT).
- When STBTM = 1, STBT is automatically cleared by hardware when the buffer fill level equals the buffer size (TRBSR.TBFLVL = TBCTR.SIZE).

Standard Transmit Buffer Event Examples

Figure 66 shows examples of the standard transmit buffer event generation 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.

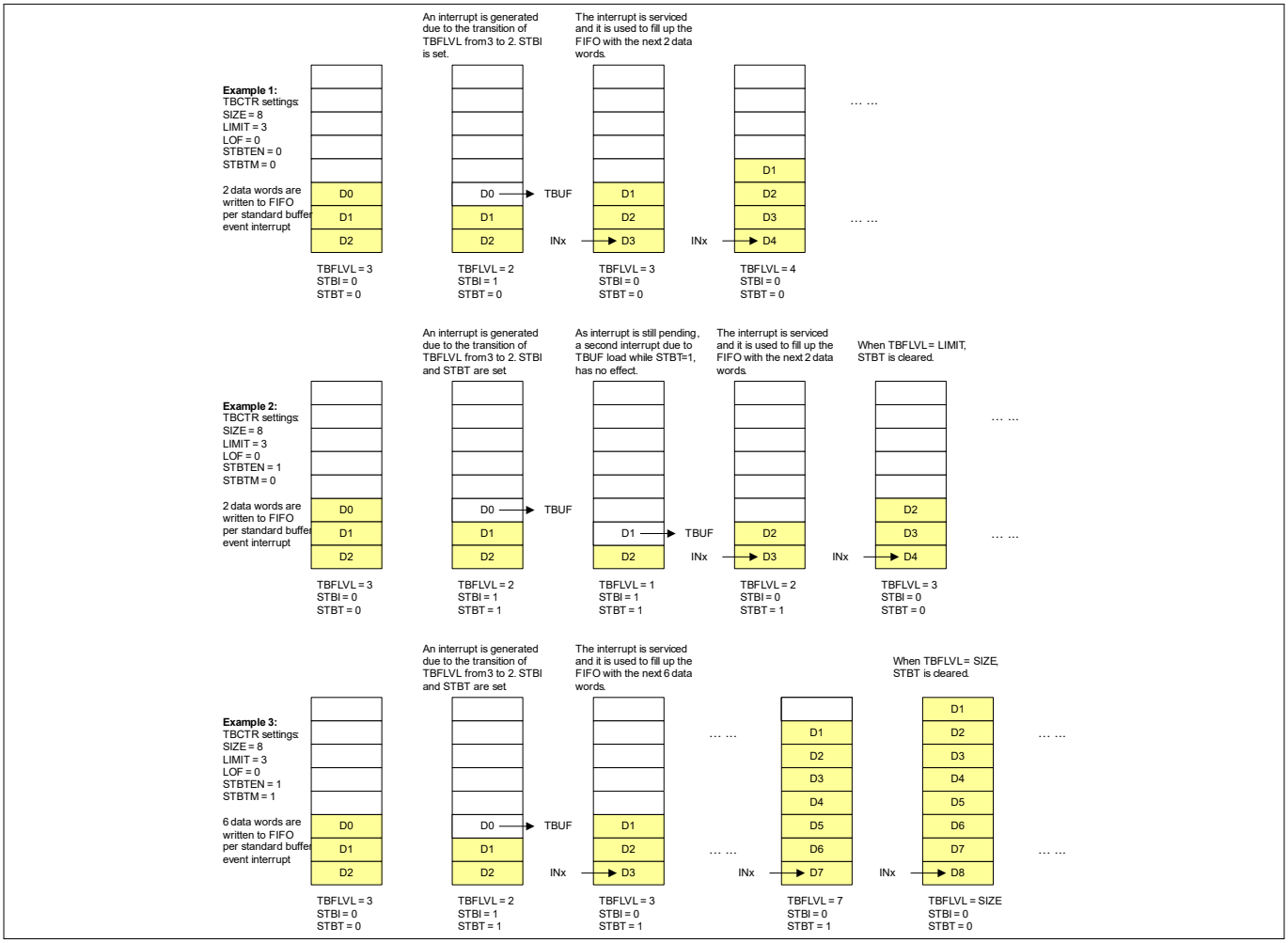


Figure 66 Standard Transmit Buffer Event Examples

16.2.8.3 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 either:

- The filling level of the transmit buffer (given by TRBSR.TBFLVL) exceeding (TBCTR.LOF = 1) or falling below (TBCTR.LOF = 0)[5] a programmed limit (TBCTR.LIMIT).
  - Applicable for all settings of TBCTR.STBTEN.
- A transfer data to TBUF event or write data to INx event, depending on TBCTR.LOF setting while TRBSR.STBT = 1.
  - Applicable only for TBCTR.STBTEN = 1.

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 67 shows the transmit buffer events and interrupts.

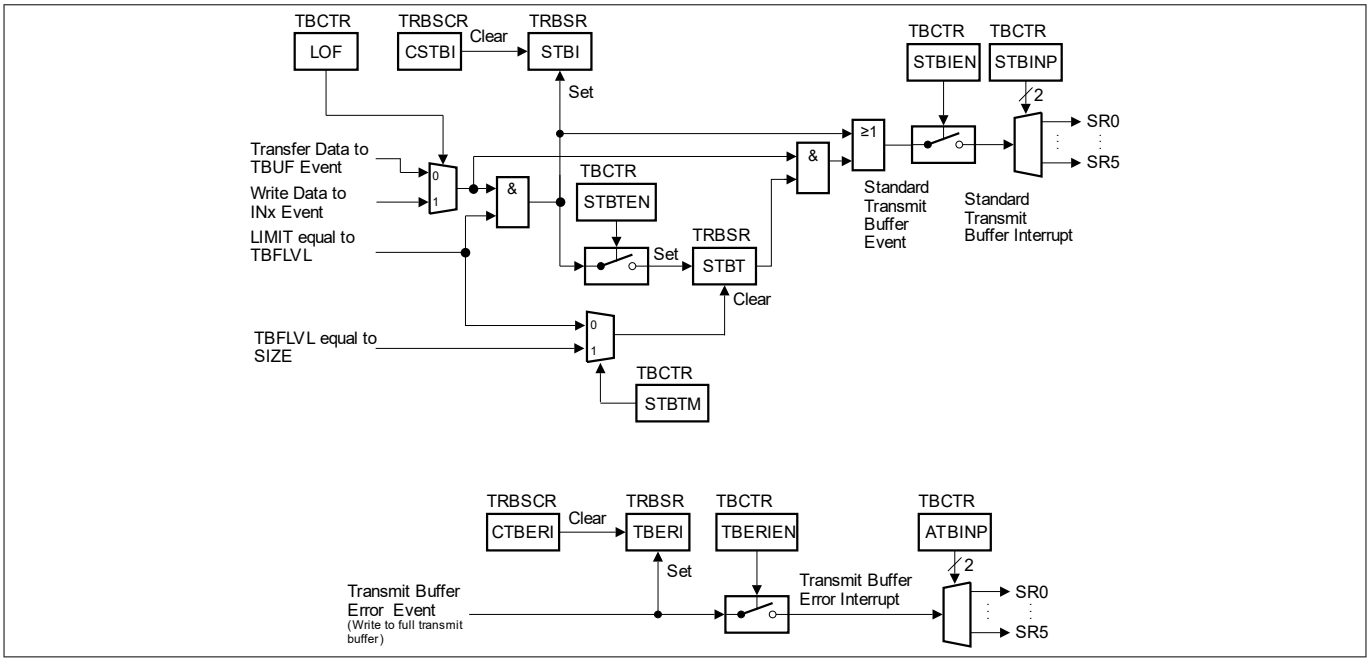


Figure 67 Transmit Buffer Events

Table 79 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 79 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

5 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.

16.2.8.4 Transmit FIFO Buffer Usage Example

To illustrate the usage of the transmit FIFO buffer, assume the following hypothetical application use case:

- A data set of 128 words is expected to be transmitted.
- 16 buffer entries are allocated as the channel's transmit data area.
- The target transmit buffer filling level is set at 9 to ensure that there is always transmit data in the buffer during the transmission of the complete data set.

TXFIFO Initialization

The following code initializes the transmit FIFO buffer in the given example:

```
/// ----------------------------------------------------------
/// Configure TXFIFO
/// - TXFIFO starts from FIFO buffer entry 0
/// - TXFIFO size of 16 words
/// - Standard transmit buffer event is triggered when fill level
/// falls below 9, i.e. transition from 9 to 8
/// ----------------------------------------------------------
// LOF SIZE STBTEN STBTM LIMIT DPTR
USIC0_CH1->TBCTR=(0<<28)|(4<<24)|(0<<15)|(0<<14)|(9<<8)|(0<<0);
```

TXFIFO Data Flow

Figure 68 shows a simplified data transmission flow for the given example.

Data transmission is started when the first data word is written to the transmit buffer input location INx. The application software continues to fill the buffer until all of the first 16 words are written to the buffer.

The buffer fill level is decremented with the transmission of each data word. When the fill level drops below the predefined limit of 9 to 8, a standard transmit buffer event is flagged through the bit TRBSR.STBI.

The standard transmit buffer event indicates to the application software that the next 8 data words should be written to the transmit buffer. A total of 14 iterations of the standard transmit buffer event is required to transmit all 128 words.

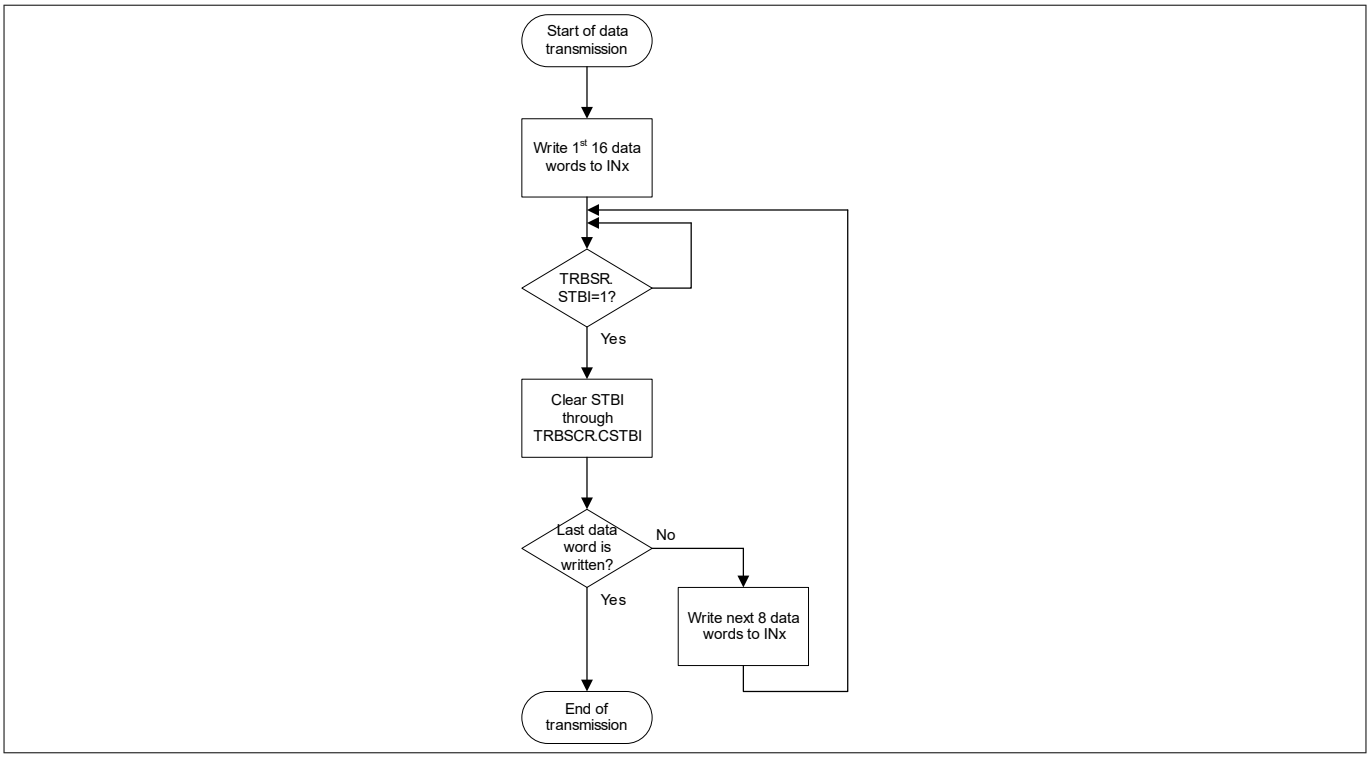


Figure 68 Simplified Data Transmission Flow With Transmit FIFO

16.2.8.5 Receive FIFO Buffer Modes

The receive FIFO buffer supports two main operation modes determined by the bit field RBCTR.RNM:

- Filling level mode (RNM = 0)
- Receive control information (RCI) mode (RNM = 1)

RCI Mode (RBCTR.RNM = 1)

In the RCI mode, each time a new data word becomes available in OUTR, an event is detected:

- If bit OUTR.RCI[4] = 0, a standard receive buffer event is signaled.
- If bit OUTR.RCI[4] = 1, an alternative receive buffer event is signaled.

The filling level of the receive buffer is not taken into account for the generation of the standard and alternate receive buffer events.

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).

Filling Level Mode (RBCTR.RNM = 0)

In the filling level mode, 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.

The receive FIFO buffer further provides two operation sub-modes with different triggering mechanisms for the standard receive buffer events, selectable through the bit field RBCTR.SRBTEN.

Filling Level Sub-Mode with SRBTEN = 0

When RBCTR.SRBTEN = 0, a standard receive buffer event is triggered only under the condition of the filling level of the receive buffer (given by TRBSR.RBFLVL) transiting away from a programmed limit (RBCTR.LIMIT).

- When RBCTR.LOF = 0, the standard transmit buffer event is triggered by RBFLVL falling below LIMIT.
- When RBCTR.LOF = 1, the standard transmit buffer event is triggered by RBFLVL exceeding LIMIT.

The standard receive buffer event is indicated by the flag SRBI.

***Note**: The standard receive buffer event indicated by TRBSR.SRBI is triggered based on the transition of the fill level from equal to below or above the limit, not the fact of being below or above.*

Filling Level Sub-Mode with SRBTEN = 1

When RBCTR.SRBTEN = 1, a standard receive buffer event can be triggered in two ways:

1. Similar to the case where SRBTEN = 0, with the difference that the flag TRBSR.SRBT is also set in addition to SRBI.
2. While SRBT = 1, a standard receive buffer event will be additionally triggered there is either a data read out event (for the case where LOF = 0) or new data received event (for the case where LOF = 1).

***Note**: Standard receive buffer events triggered by TRBSR.SRBT does not affect the TRBSR.SRBI flag.*

The SRBT flag is cleared depending on the setting of RBCTR.SRBTM bit:

- When SRBTM = 0, SRBT is automatically cleared by hardware when the buffer fill level equals the programmed limit again (TRBSR.RBFLVL = RBCTR.LIMIT).
- When SRBTM = 1, SRBT is automatically cleared by hardware when the buffer fill level equals 0 (TRBSR.RBFLVL = 0).

Standard Receive Buffer Event Examples in Filling Level Mode

Figure 69 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.

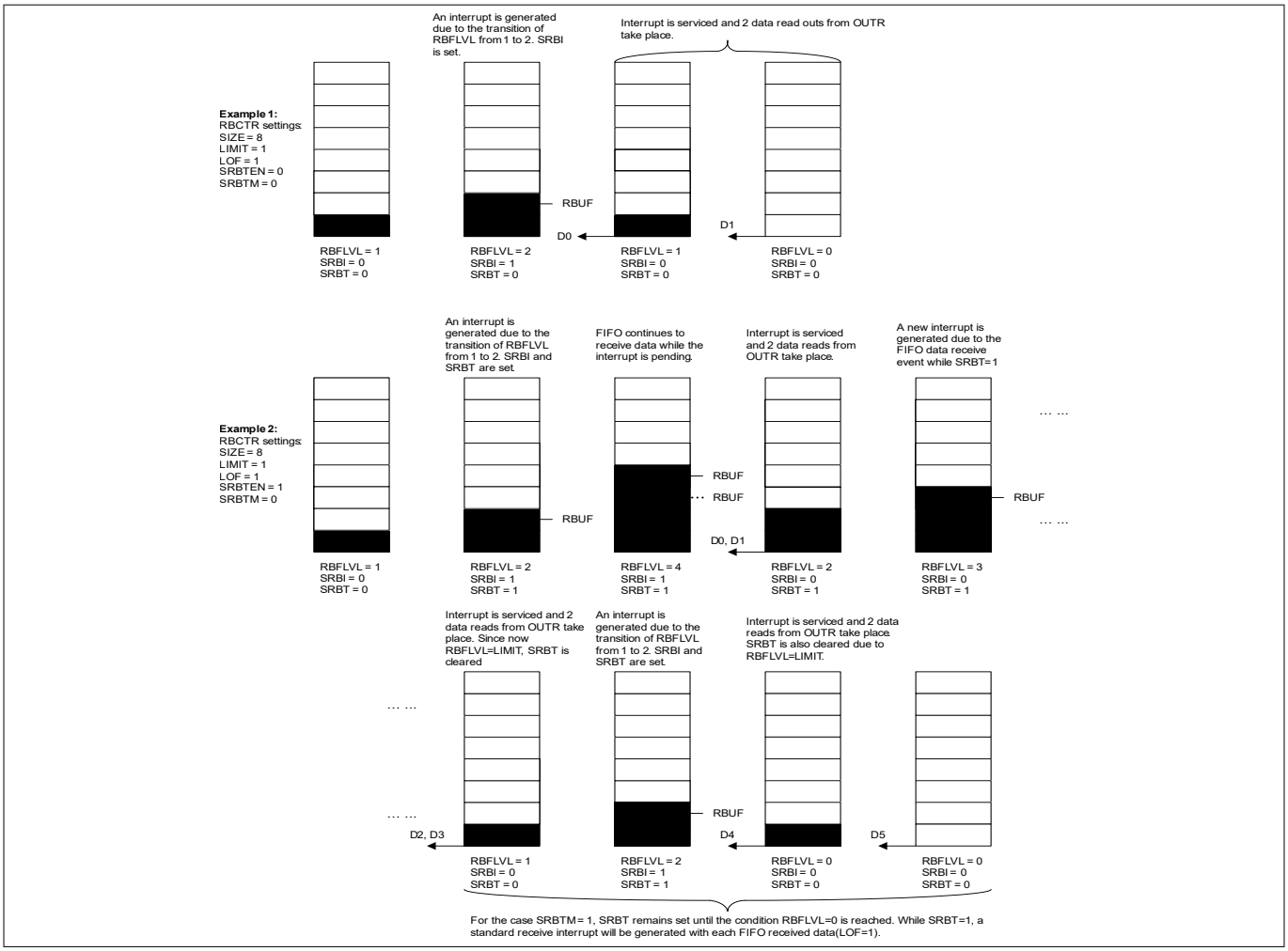


Figure 69 Standard Receive Buffer Event Examples

16.2.8.6 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

Standard Receive Buffer Event in Filling Level Mode

In filling level mode (RBCTR.RNM = 0), the standard transmit buffer event is triggered by either:

- 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)[6].
  - Applicable for all settings of RBCTR.SRBTEN.
- A data read out event or new data received event, depending on RBCTR.LOF setting while TRBSR.SRBT = 1.
  - Applicable only for RBCTR.SRBTEN = 1.

6 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.

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 70 shows the receiver buffer events and interrupts in filling level mode.

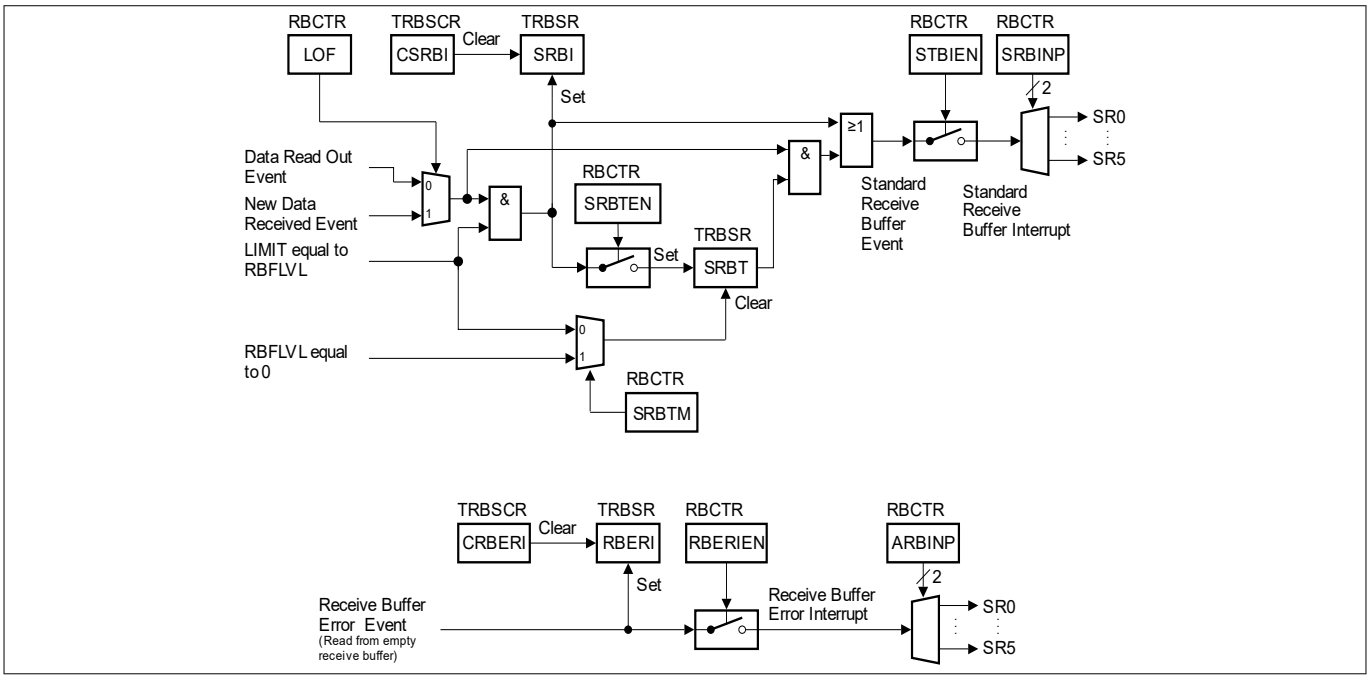


Figure 70 Receiver Buffer Events in Filling Level Mode

Figure 71 shows the receiver buffer events and interrupts in RCI mode.

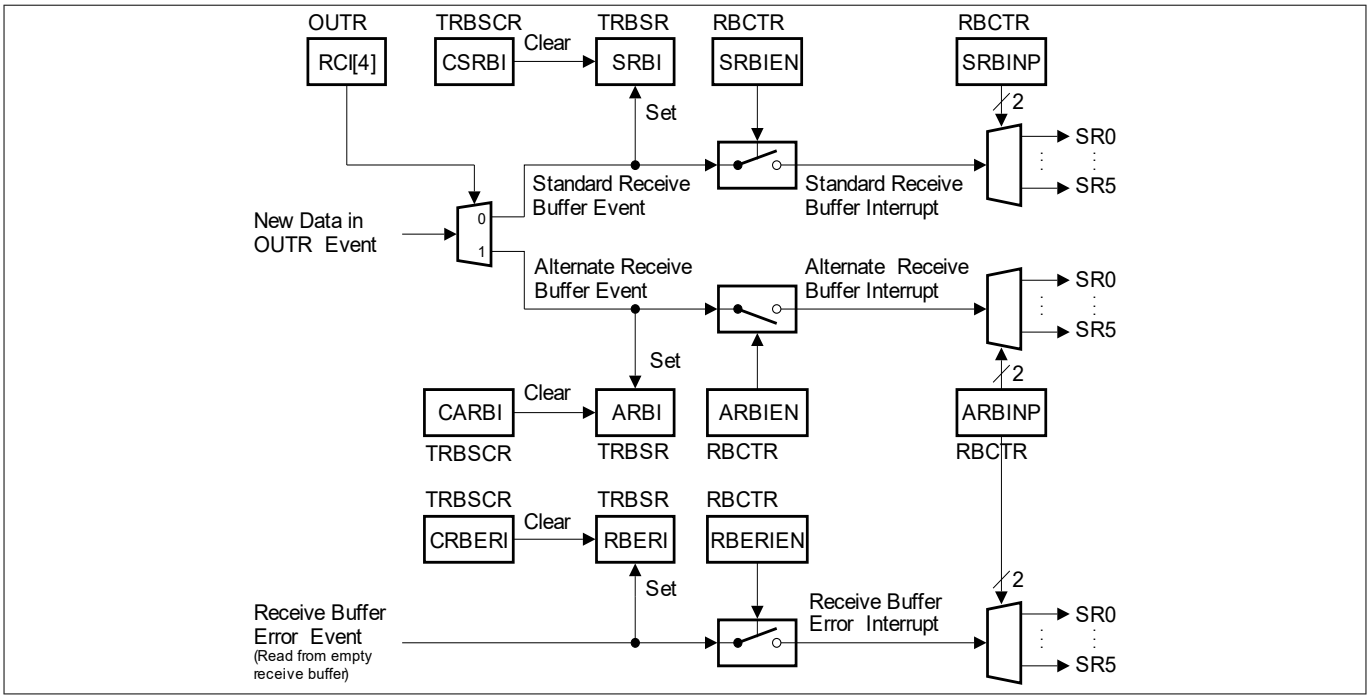


Figure 71 Receiver Buffer Events in RCI Mode

Table 80 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 80 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		
Alternative receive buffer event	TRBSR.ARBI	TRBSCR.CARBI	RBCTR.ARBIEN	RBCTR.ARBINP
Receive buffer error event	TRBSR.RBERI	TRBSCR.CRBERI	RBCTR.RBERIEN	RBCTR.ARBINP

16.2.8.7 Receive FIFO Buffer in Filling Level Mode Usage Example

To illustrate the usage of the receive FIFO buffer, assume the following hypothetical application use case:

- A data set of 128 words is expected to be received.
- 16 buffer entries are allocated as the channel's receive data area.
- The receive buffer filling level mode is selected and the standard receive buffer event is flagged each time the fill level reaches 8.

RXFIFO Initialization

The following code initializes the receive FIFO buffer in the given example:

```
/// ----------------------------------------------------------
/// Configure RXFIFO
/// - RXFIFO starts from FIFO buffer entry 32
/// - RXFIFO size of 16 data words
/// - Standard receive buffer event is triggered when fill level
/// exceeds 7, i.e. transition from 7 to 8
/// ----------------------------------------------------------
USIC0_CH1->RBCTR
=(1<<28)|(0<<27)|(4<<24)|(0<<15)|(0<<14)|(7<<8)|(32<<0);
// LOF RNM SIZE SRBTEN SRBTM LIMIT DPTR
```

RXFIFO Data Flow

Figure 72 shows a simplified data reception flow for the given example.

The buffer fill level is incremented with the reception of each data word. When the fill level exceeds the predefined limit of 7 and transits to 8, a standard receive buffer event is flagged through the bit TRBSR.SRBI.

The application software reads out the 8 data words in the buffer, through the receive buffer output register OUTR, with each standard receive buffer event. A total of 16 iterations of the standard receive buffer event is required to receive all 128 words.

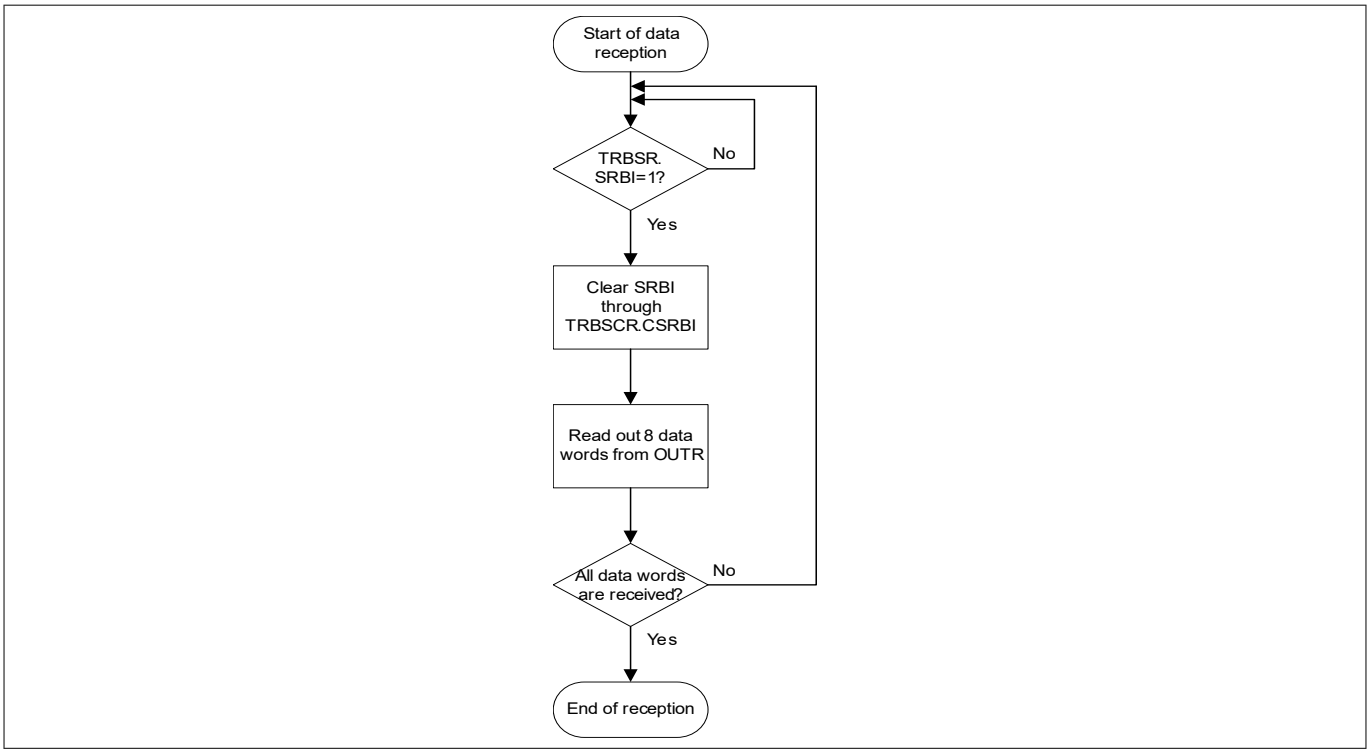


Figure 72 Simplified Data Reception Flow in Receive Buffer Filling Level Mode

16.2.8.8 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, for example, 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 73.

- 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, for example, 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.

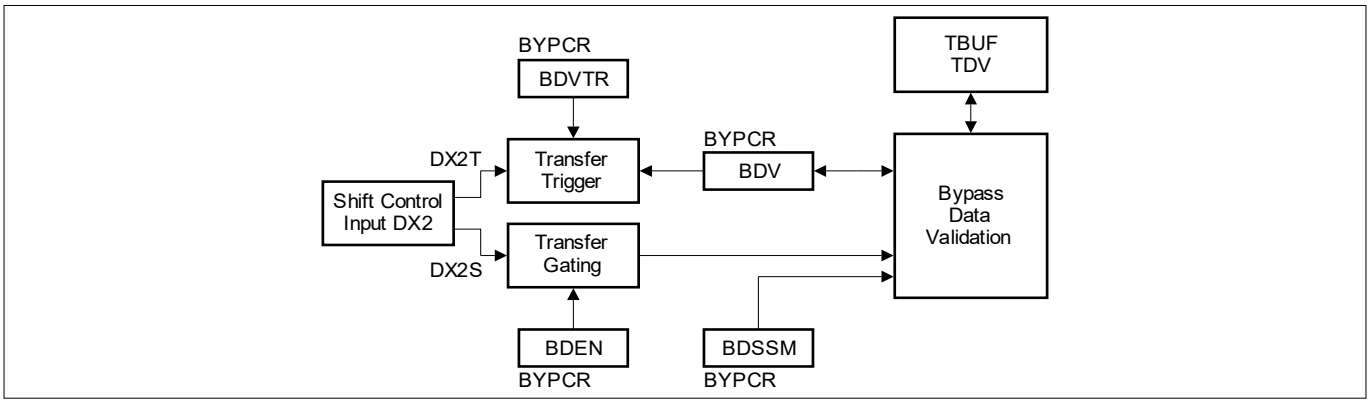


Figure 73 Bypass Data Validation

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 (for example, 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 (for example, as alternative data if the data FIFO runs empty).

16.2.8.9 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.

16.2.8.10 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, as shown in Table 81 and Figure 74.

See Transmit Control Information for more details on TCI.

Table 81 TCI Handling with FIFO / Bypass

TCSR Setting	Modifications of USIC Channel Parameters
SELMD = 1	• Update of PCR.CTR[20:16] by FIFO TCI or BYPCR.BSELO • Additional clear of PCR.CTR[23:21]
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)
FLEMD = 1	• Update of SCTR.FLE[4:0] by FIFO TCI or BYPCR.BWLE • Additional clear of SCTR.FLE[5]
HPCMD = 1	• Update of SCTR.DSM and SCTR.HPCDIR by FIFO TCI or BYPCR.BHPC
WAMD = 1	• Update of TCSR.WA by FIFO TCI[4]

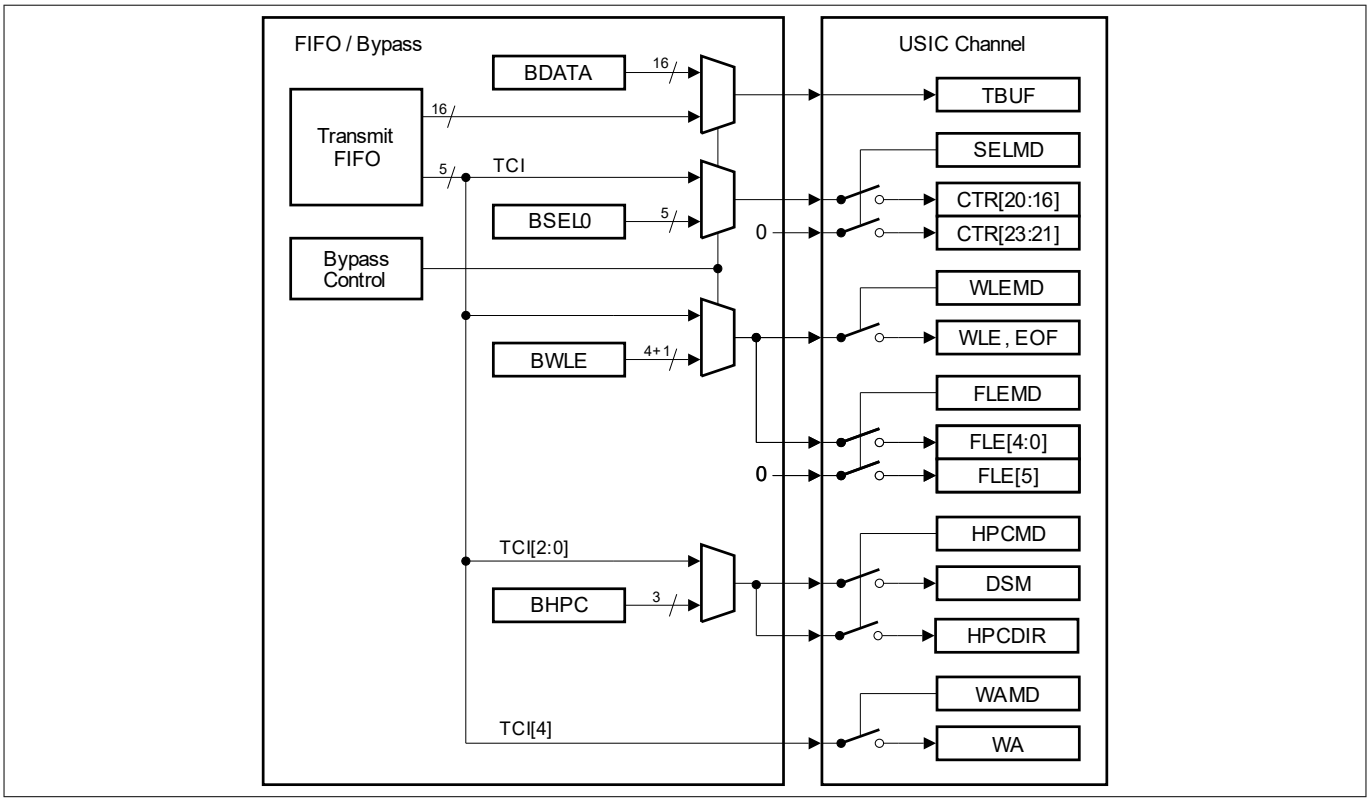


Figure 74 TCI Handling with FIFO / Bypass

16.3 Asynchronous Serial Channel (ASC = UART)

The asynchronous serial channel ASC covers the reception and the transmission of asynchronous data frames and provides 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.

16.3.1 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.

16.3.1.1 ASC Full-Duplex Communication

For full-duplex communication, an independent communication line is needed for each transfer direction.

Figure 75 shows an example with a point-to-point full-duplex connection between two communication partners ASC A and ASC B.

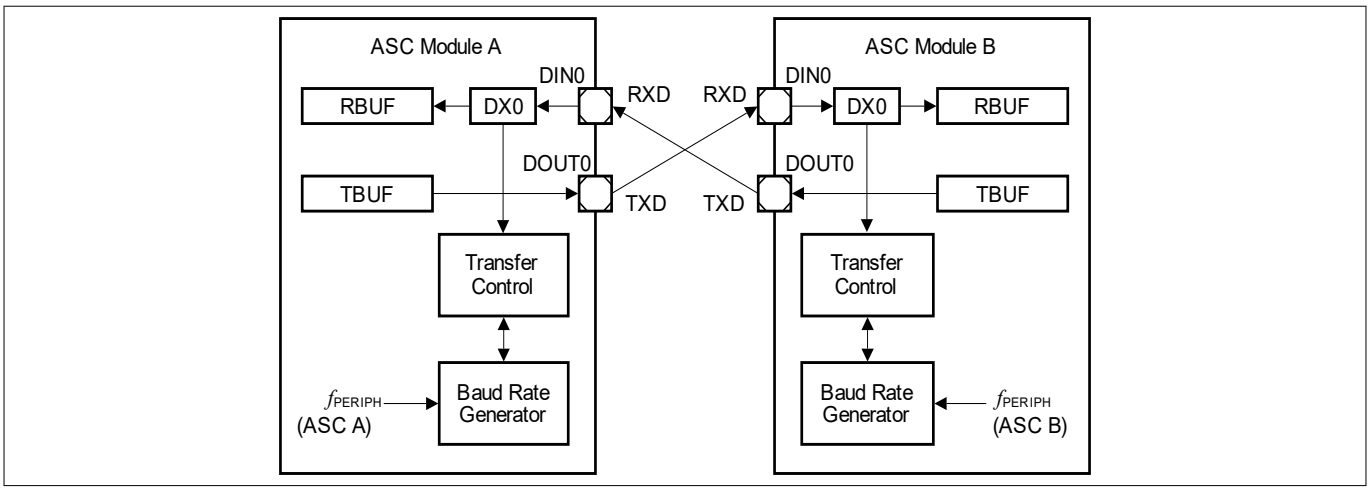


Figure 75 ASC Signal Connections for Full-Duplex Communication

Figure 76 shows the pins that support ASC full-duplex communication.

USIC0_CH0										USIC0_CH1									
Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40
P0.14	DX0A	57	45	39	P0.14	DOUT0	57	45	39	P1.3	DX0A	31	23	19	P1.3	DOUT0	31	23	19
P0.15	DX0B	58	46	40	P0.15	DOUT0	58	46	40	P1.2	DX0B	32	24	20	P1.2	DOUT0	32	24	20
P1.0	DX0C	34	26	22	P1.0	DOUT0	34	26	22	P0.6	DX0C	47	35	29	P0.6	DOUT0	47	35	29
P1.1	DX0D	33	25	21	P1.1	DOUT0	33	25	21	P0.7	DX0D	48	36	30	P0.7	DOUT0	48	36	30
P2.0	DX0E	9	3	1	P2.0	DOUT0	9	3	1	P2.11	DX0E	20	14	12	P2.11	DOUT0	20	14	12
P2.1	DX0F	10	4	2	P2.1	DOUT0	10	4	2	P2.10	DX0F	19	13	11	P2.10	DOUT0	19	13	11
					P1.5	DOUT0	29	21	17						P1.6	DOUT0	28	20	16

USIC1_CH0										USIC1_CH1									
Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40
P0.2	DX0A	43	31	25	P0.2	DOUT0	43	31	25	P0.0	DX0A	41	29	23	P0.0	DOUT0	41	29	23
P0.3	DX0B	44	32	26	P0.3	DOUT0	44	32	26	P0.1	DX0B	42	30	24	P0.1	DOUT0	42	30	24
P4.4	DX0C	63	47	-	P4.4	DOUT0	63	47	-	P2.12	DX0C	21	15	-	P2.12	DOUT0	21	15	-
P4.5	DX0D	64	48	-	P4.5	DOUT0	64	48	-	P2.13	DX0D	22	16	-	P2.13	DOUT0	22	16	-
-	DX0E	39	-	-	-	DOUT0	39	-	-	P3.0	DX0E	36	28	-	P3.0	DOUT0	36	28	-
-	DX0F	40	-	-	-	DOUT0	40	-	-	-	DX0F	37	-	-	-	DOUT0	37	-	-

Figure 76 Available Pins for ASC Full-Duplex Communication

16.3.1.2 ASC Half-Duplex Communication

For half-duplex or multi-transmitter communication, a single communication line is shared between the communication partners.

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.

Figure 77 shows an example with a point-to-point half-duplex connection between ASC A and ASC B.

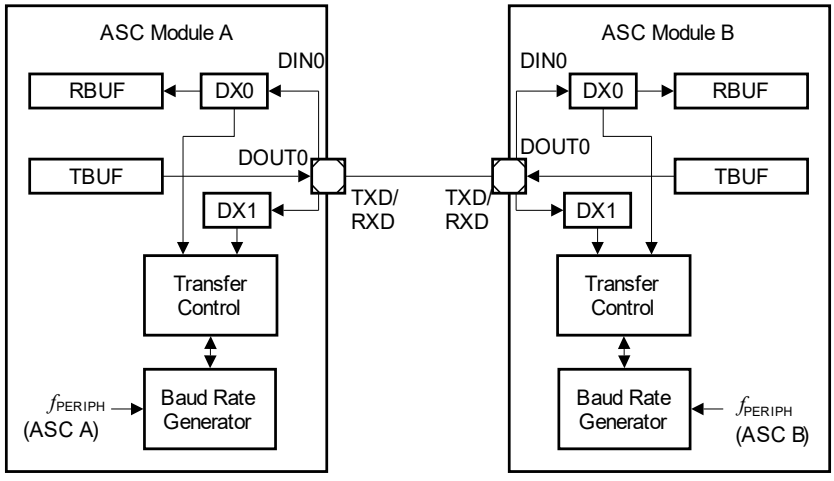


Figure 77 ASC Signal Connections for Half-Duplex Communication

Figure 78 shows the pins that support ASC half-duplex communication.

USIC0_CH0

Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40
P0.14	DX0A	57	45	39	P0.14	DOUT0	57	45	39
P0.15	DX0B	58	46	40	P0.15	DOUT0	58	46	40
P1.0	DX0C	34	26	22	P1.0	DOUT0	34	26	22
P1.1	DX0D	33	25	21	P1.1	DOUT0	33	25	21
P2.0	DX0E	9	3	1	P2.0	DOUT0	9	3	1
P2.1	DX0F	10	4	2	P2.1	DOUT0	10	4	2

USIC0_CH1

Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40
P1.3	DX0A	31	23	19	P1.3	DOUT0	31	23	19
P1.2	DX0B	32	24	20	P1.2	DOUT0	32	24	20
P0.6	DX0C	47	35	29	P0.6	DOUT0	47	35	29
P0.7	DX0D	48	36	30	P0.7	DOUT0	48	36	30
P2.11	DX0E	20	14	12	P2.11	DOUT0	20	14	12
P2.10	DX0F	19	13	11	P2.10	DOUT0	19	13	11

USIC1_CH0

Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40
P0.2	DX0A	43	31	25	P0.2	DOUT0	43	31	25
P0.3	DX0B	44	32	26	P0.3	DOUT0	44	32	26
P4.4	DX0C	63	47	-	P4.4	DOUT0	63	47	-
P4.5	DX0D	64	48	-	P4.5	DOUT0	64	48	-
-	DX0E	39	-	-	-	DOUT0	39	-	-
-	DX0F	40	-	-	-	DOUT0	40	-	-

USIC1_CH1

Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40
P0.0	DX0A	41	29	23	P0.0	DOUT0	41	29	23
P0.1	DX0B	42	30	24	P0.1	DOUT0	42	30	24
P2.12	DX0C	21	15	-	P2.12	DOUT0	21	15	-
P2.13	DX0D	22	16	-	P2.13	DOUT0	22	16	-
P3.0	DX0E	36	28	-	P3.0	DOUT0	36	28	-
-	DX0F	37	-	-	-	DOUT0	37	-	-

Figure 78 Available Pins for ASC Half-Duplex Communication

16.3.2 Frame Format

A standard ASC frame is shown in Figure 79. 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.

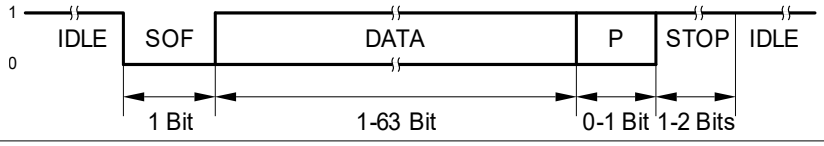


Figure 79 Standard ASC Frame Format

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.

16.3.2.1 Idle Detection

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, for example, 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).

16.3.2.2 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.

16.3.2.3 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, for example, 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.

16.3.2.4 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).

16.3.2.5 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.

16.3.3 Operating the ASC

In order to operate the ASC protocol, the USIC channel has to be first initialized.

It is recommended to configure all parameters of the ASC that do not change during run time while CCR.MODE = 0000_B, except stated otherwise.

The main initialization steps are outlined below:

- **Enable USIC channel**
  - Enable the module by writing 1s to MODEN and BPMODEN bits in KSCFG register.
- **Configure baud rate generator**
  - Select either fractional divider mode or normal divider mode with FDR.DM bit.
  - Configure the bit timing and baud rate setting through register BRG and FDR.STEP bit field.
  - **Configure input pins**
    - Establish a connection of input stage DX0 with the receive data input pin (signal DIN0) selected by DX0CR.DSEL bit field and with bit DX0CR.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.
    - 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.
- **Configure data format**
  - 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.
  - Write SCTR.TRM = 01_B to enable ASC data transfers.
  - If required by the application, the data input and output signals can be inverted with DX0CR.DPOL and SCTR.DOCFG respectively.
- **Configure data transfer parameters**
  - Write TCSR.TDSSM = 1 and TCSR.TDEN = 01_B to enable data transmission in single shot mode.
- **Configure protocol control parameters**
  - Select the idle detection mode, number of stop bits, sample mode and sample point through the register PCR.

- **Select ASC protocol**
  - Enable ASC mode with CCR.MODE = 0010_B.
  - Additionally, the parity mode can be configured with CCR.PM.
- **Configure output pins**
  - Configure the transmit data output pin (signal DOUT0) through the selected pin's port control register Pn_IOCR0/4/8/12. Refer to the port chapter.
  - The step to enable the output pin functions should only be done after the ASC mode is enabled with CCR.MODE, to avoided unintended spikes on the output.

Please note that not all feature combinations can be supported by the application at the same time, for example, 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.

An initialization code example is given in Initialization Code Example.

16.3.3.1 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 80, 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 (BRG.DCTQ = 15) with sampling after 8 or 9 time quanta (PCR.SP = 8 or 9) with majority decision (PCR.SMD = 1).

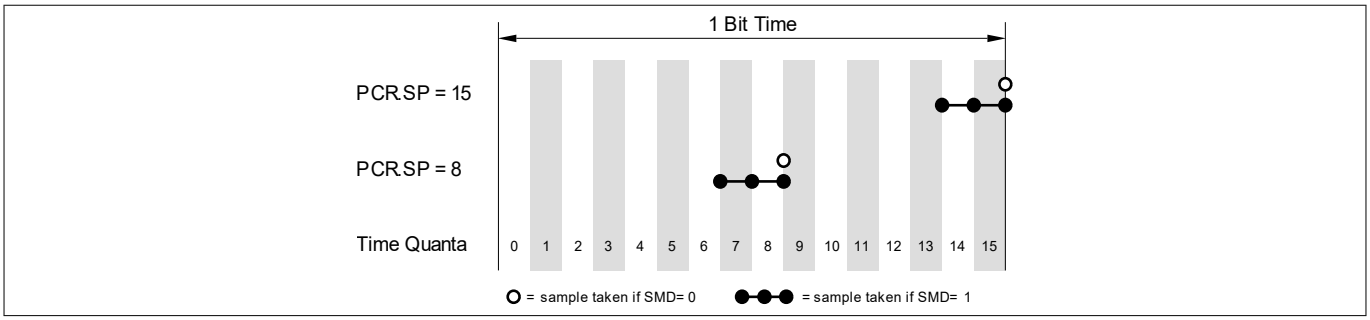


Figure 80 ASC Bit Timing

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.

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.

16.3.3.2 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 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 baud rate is given by:

$$f_{ASC} = f_{CTQIN} \times \frac{1}{PCTQ + 1} \times \frac{1}{DCTQ + 1} \quad (20)$$

The standard setting is given by CTQSEL = 00_B ($f_{CTQIN} = f_{PDIV}$), PPPEN = 0 ($f_{PPP} = f_{PIN}$) and CLKSEL = 00_B ($f_{PIN} = f_{FD}$). Under these conditions, the baud rate can further be equated to either (21) or (22), depending on the selected divider mode:

- In normal divider mode (FDR.DM = 01_B)

$$f_{ASC} = f_{PERIPH} \times \frac{1}{1024 - STEP} \times \frac{1}{PDIV + 1} \times \frac{1}{PCTQ + 1} \times \frac{1}{DCTQ + 1} \quad (21)$$

- In fractional divider mode (FDR.DM = 10_B)

$$f_{ASC} = f_{PERIPH} \times \frac{STEP}{1024} \times \frac{1}{PDIV + 1} \times \frac{1}{PCTQ + 1} \times \frac{1}{DCTQ + 1} \quad (22)$$

Table 82 shows examples of the baud rate calculation in fractional divider mode (FDR.DM = 10_B).

Table 82 ASC Baud Rate Calculation in Fractional Divider Mode

f_{PERIPH} (MHz)	FDR.STEP	BRG.PDIV	BRG.PCTQ	BRG.DCTQ	f_{ASC} (kbit/s)	Deviation Error
48	472	71	1	15	9.6	0.03%
48	472	35	1	15	19.2	0.03%
48	472	5	1	15	115.2	0.03%
48	512	2	0	7	1000	0.00%

The baud rate setting should only be changed while the transmitter and the receiver are idle.

16.3.3.3 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 transfer, 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 (for example, in a LIN bus system), SCTR.WLE can be modified while a data transfer is in progress after the RSI event has been detected.

16.3.3.4 Mode Control Behavior

In ASC mode, the following kernel modes are supported:

- Run Mode 0/1:

  Behavior as programmed, no impact on data transfers. Default is Run Mode 0.
- 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.

16.3.3.5 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.

16.3.3.6 Data Transfer Interrupt Handling

The data transfer interrupts indicate events related to ASC frame handling.

Table 83 ASC data transfer interrupt handling

Interrupt	Indicated by bit	Description
Transmit buffer interrupt	PSR.TBIF	Set after the start of 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. This is the earliest point in time when a new data word can be written to TBUF.
Transmit shift interrupt	PSR.TSIF	Set after the start of the last data bit of a data word.
Receiver start interrupt	PSR.RSIF	Set after the sample point of the first data bit of a data word.

(table continues...)

Table 83 (continued) ASC data transfer interrupt handling

Interrupt	Indicated by bit	Description
Receiver interrupt	PSR.RIF	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), 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, AIF is set instead of RIF. The first data word of a data frame is indicated by RBUFSR.SOF = 1 for the received word.
Alternative interrupt	PSR.AIF	
Data lost interrupt	PSR.DLIF	Set 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.

16.3.3.7 Protocol Interrupt Events

The following protocol-related events are generated in ASC mode and can lead to a protocol interrupt.

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.

Table 84 ASC protocol interrupt events

Events	Event flag	Description	Flag clear	Interrupt enable
Transmitter events				
Collision detection	PSR.COL	Transmitted value (DOUT0) does not match with the input value of the DX1 input stage at the sample point of a bit.	PSCR.CST3	PCR.CDEN
Transmitter frame finished	PSR.TFF	Transmitter has completely finished a frame. 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.	PSCR.CST8	PCR.FFIEN
Receiver events				
Receiver frame finished	PSR.RFF	Receiver has completely finished a frame. 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.	PSCR.CST7	PCR.FFIEN
Synchronization break detection	PSR.SBD	Used in LIN networks to indicate the reception of the synchronization break symbol (at the beginning of a LIN frame).	PSCR.CST2	PCR.SBIEN
Receiver noise detection	PSR.RNS	Input value at the sample point of a bit and at the two time quanta before are not identical.	PSCR.CST4	PCR.RNIEN
Format error	PSR.FER0/ FER1	A format error is signalled if the sampled bit value of a stop bit is 0.	PSCR.CST5/ CST6	PCR.FEIEN

16.3.3.8 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.

16.3.3.9 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 ASC mode, the received parity bit is monitored by the PAR flag and the result of the parity check by the PERR flag (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.

16.3.3.10 Receive FIFO Buffer Handling

If a receive FIFO buffer is enabled for data handling (RBCTR.SIZE > 0), it is recommended to set RBCTR.RCIM = 11_B in ASC mode. This leads to the following indications on the 5-bit field OUTR.RCI:

- RCI[0]: A 1 indicates that the data word has been the first data word of a new data frame
- RCI[3]: The bit contains the received parity bit value
- RCI[4]: A 1 indicates a parity error

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 (TRBSR.SRBI = 1) indicates that a data word can be read from OUTR that has been received without parity error.
- An alternative receive buffer event (TRBSR.ARBI = 1) indicates that a data word can be read from OUTR that has been received with parity error.

16.3.3.11 Data Flow Handling

This section describes the data flow during data transmission and reception.

Data Transmission

Data transmission is initiated by loading the transmit buffer TBUF with the data word to be transmitted, through one of the transmit buffer input locations (TBUFx). If the transmit FIFO is used, one of the transmit FIFO buffer input locations (INx) should be used instead of TBUFx.

If the transmitter is not busy transmitting a previous data word, the data word will be immediately updated to the transmit shift register. This is indicated by a transmit buffer interrupt event (PSR.TBIF = 1).

The transmit buffer interrupt event can be used to indicate that the next data word can now be loaded to TBUF.

For each data word, the start of transmission of the last data bit is indicated by a transmit shift interrupt event (PSR.TSIF = 1). The end of transmission of a complete data frame is indicated by the transmit frame finished event (PSR.TFF = 1).

Figure 81 shows a simplified data transmission flow with TBUFx. For an example with transmit FIFO, refer to Transmit FIFO Buffer Usage Example.

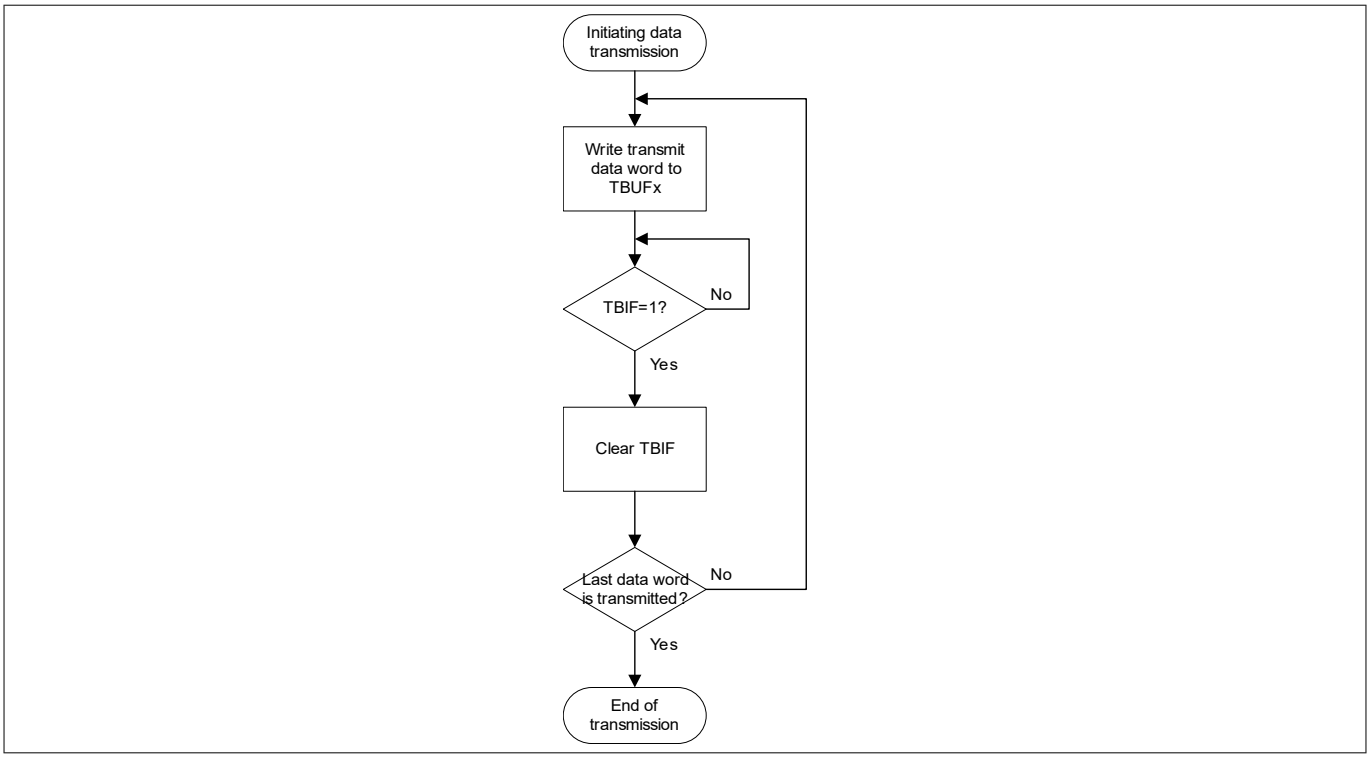


Figure 81 Simplified Data Transmission Flow

Data Reception

Data reception is initiated with the detection of a valid start bit on the receive input line.The receiver continues to sample the incoming data frame and shifts it into the available receive shift register.

For each data word, the sampling of the first data bit is indicated by a receive start interrupt event (PSR.RSIF = 1).

When the number of bits corresponding to the data word length is received, the contents of the receive shift register are transferred to the corresponding receive buffer (RBUF0 or RBUF1). This is indicated by a receive interrupt event (PSR.RIF) or an alternate receive interrupt event (PSR.AIF) if parity is enabled and a parity error is detected.

The reception of a complete data frame, after the last stop bit is received, is indicated by the receive frame finished event (PSR.RFF).

The user needs to read only the RBUF register for the received data word. If the receive FIFO buffer is used, the received data word should be read from register OUTR instead of RBUF.

Figure 82 shows a simplified data reception flow with RBUF. For an example with receive FIFO buffer, refer to Receive FIFO Buffer in Filling Level Mode Usage Example.

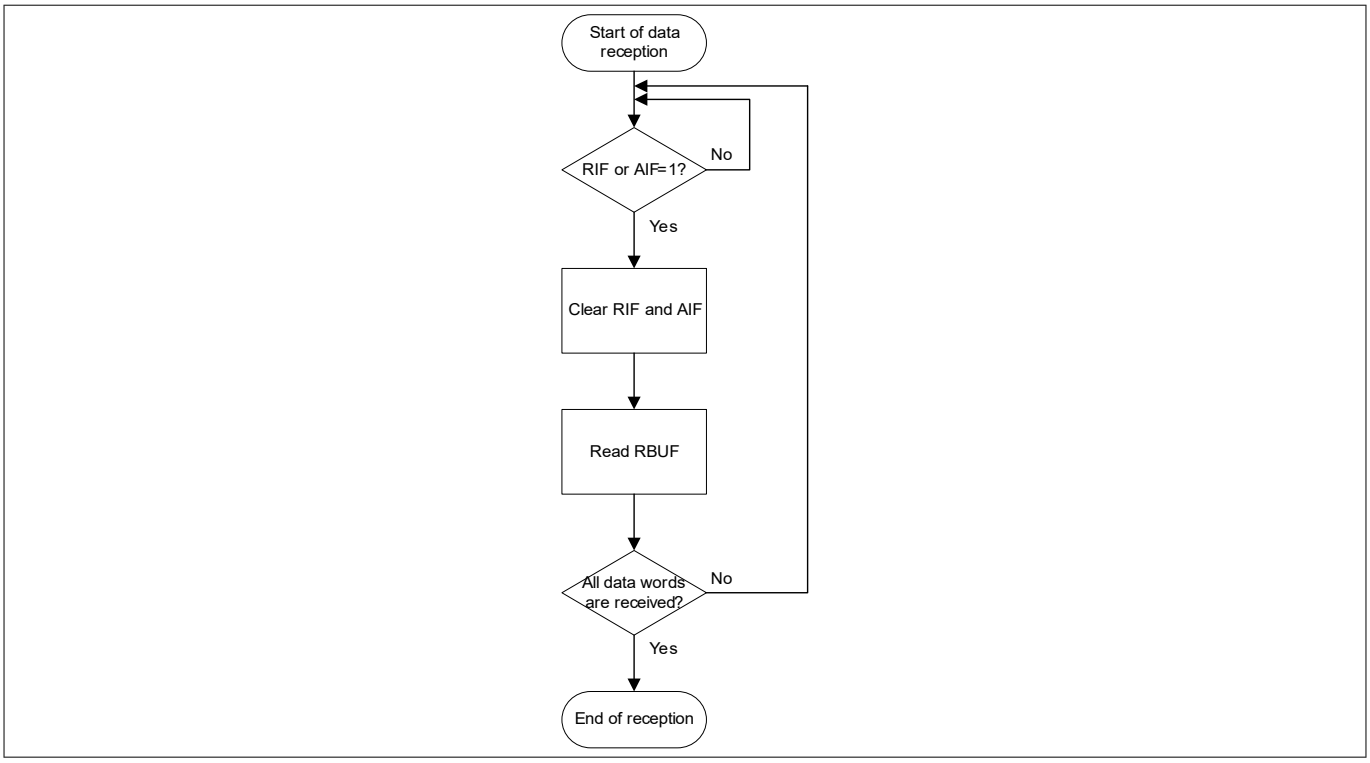


Figure 82 Simplified Data Reception Flow

Interrupt Events on Data Transfer

Figure 83 shows the interrupt events during an ASC data transfer for both the transmitter and the receiver.

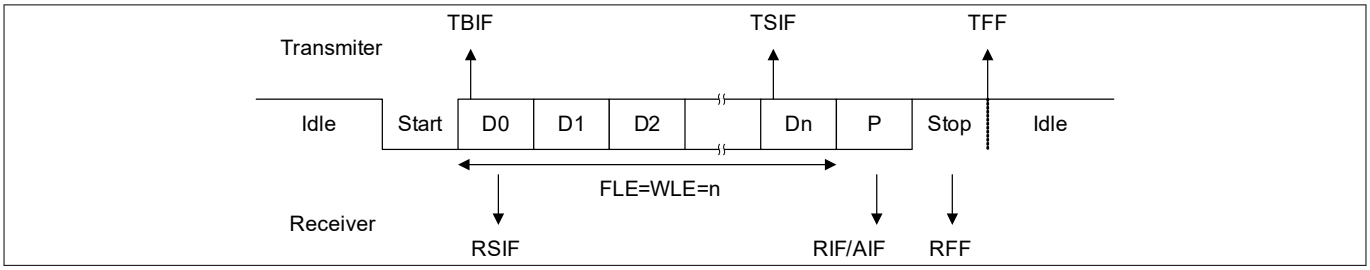


Figure 83 Interrupt Events on Data Transfer

16.3.3.12 Initialization Code Example

The following code shows an example of an ASC initialization sequence:

```
void USIC0_CH1_Init(void)
{
/// ----------------------------------------------------------
/// 1.Enable USIC0 channel 1
/// ----------------------------------------------------------
// BPMODEN MODEN
USIC0_CH1->KSCFG|= (1 << 1)| (1 << 0);
/// ----------------------------------------------------------
/// 2.Configure baud rate generator
/// - Fractional divider mode
/// - Baud rate = 19200 bit/s
/// - 16 time quanta per bit time
/// - Length of time quanta = 2 / f_CTQIN
/// ----------------------------------------------------------
// DM STEP
USIC0_CH1->FDR = (2 << 14) | (472 << 0);
// PDIV DCTQ PCTQ
USIC0_CH1->BRG = (35 << 16) | (15 << 10) | (1 << 8);
/// ----------------------------------------------------------
/// 3.Configure input stages
/// - Select input DX0C
/// - Protocol pre-processor to control input of data shift unit
/// ----------------------------------------------------------
// INSW DSEL
USIC0_CH1->DX0CR =(0 << 4) |(2 << 0);
/// ----------------------------------------------------------
/// 4.Configure data format
/// - Data word = data frame = 8 bits
/// - Data transfer allowed with passive level = 1 and LSB first
/// ----------------------------------------------------------
// WLE FLE TRM PDL SDIR
USIC0_CH1->SCTR =(7<<24)|(7<<16)|(1<<8)|(1<<1)|(0<<0);
/// ----------------------------------------------------------
/// 5.Configure data transfer parameters
/// - Single shot transmission of data word when a valid word
/// is available
/// ----------------------------------------------------------
// TDEN TDSSM
USIC0_CH1->TCSR =(1 << 10) |(1 << 8);
/// ----------------------------------------------------------
/// 6.Configure ASC protocol-specific parameters
/// - 1 stop bit
/// - Bit value is majority of the values sampled at the
/// latest 3 time quanta with respect to bit 8
/// ----------------------------------------------------------
// SP STPB SMD
USIC0_CH1->PCR =(8 << 8) |(0 << 1) | (1 <<0);
/// ----------------------------------------------------------
/// 7.Enable ASC protocol and select parity mode
```

```
/// - No parity is selected
/// ------------------------------------------------------------
// PM MODE
USIC0_CH1->CCR =(0 << 8) |(2 << 0);
/// ------------------------------------------------------------
/// 8.Configure ASC output function pins
/// - Assume P0.7 ALT7 function is assigned to DOUT0
/// ------------------------------------------------------------
// PC7
PORT0->IOCR4 |= (0x17 << 27);
}
```

16.3.4 Additional Features

The next sub-sections describe additional features supported in the ASC mode.

16.3.4.1 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.

16.3.4.2 Collision Detection

In some applications, such as data transfer over a single data line shared by several sending devices (see Figure 77), 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 RXD 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. The transmitter stops its data transmission (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.

16.3.4.3 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.

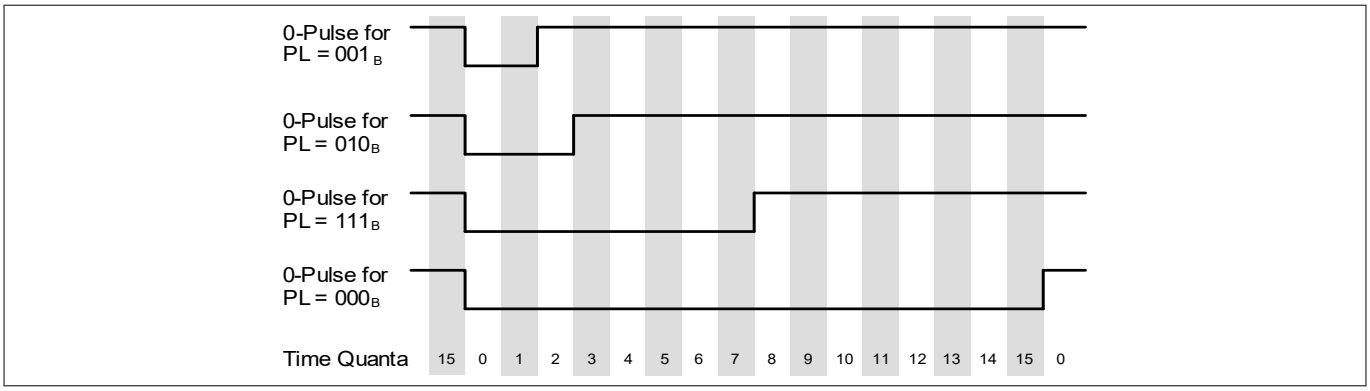


Figure 84 Transmitter Pulse Length Control

Figure 85 shows an example for the transmission of an 8-bit data word with LSB first and one stop bit (for example, like for IrDA). The polarity of the transmit output signal has been inverted by SCTR.DOCFG = 01_B.

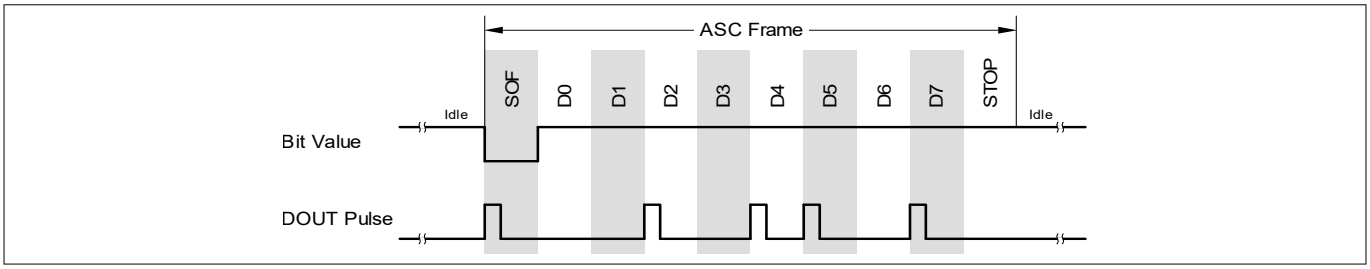


Figure 85 Pulse Output Example

16.3.4.4 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 SCTR.FLE when the ASC frame is started (leading to frames with 1 to 32 data bits).

The TCI value represents the written address location of TBUFx (if FIFO is not used) or INxx (if FIFO is used). With this mechanism, an ASC with 8 data bits is generated by writing a data word to TBUF07 (IN07, respectively).

16.3.4.5 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 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).

16.3.4.6 Transfer Status Indication

The receiver status can be monitored by flag PSR.BUSY if the receiver status enable bit PCR.RSTEN is set. In this case, 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 the same BUSY flag if the transmitter status enable bit PCR.TSTEN is set. In this case, bit BUSY is set during a complete frame transmission 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).

16.3.5 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 (for example, 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.

16.4 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.

16.4.1 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).

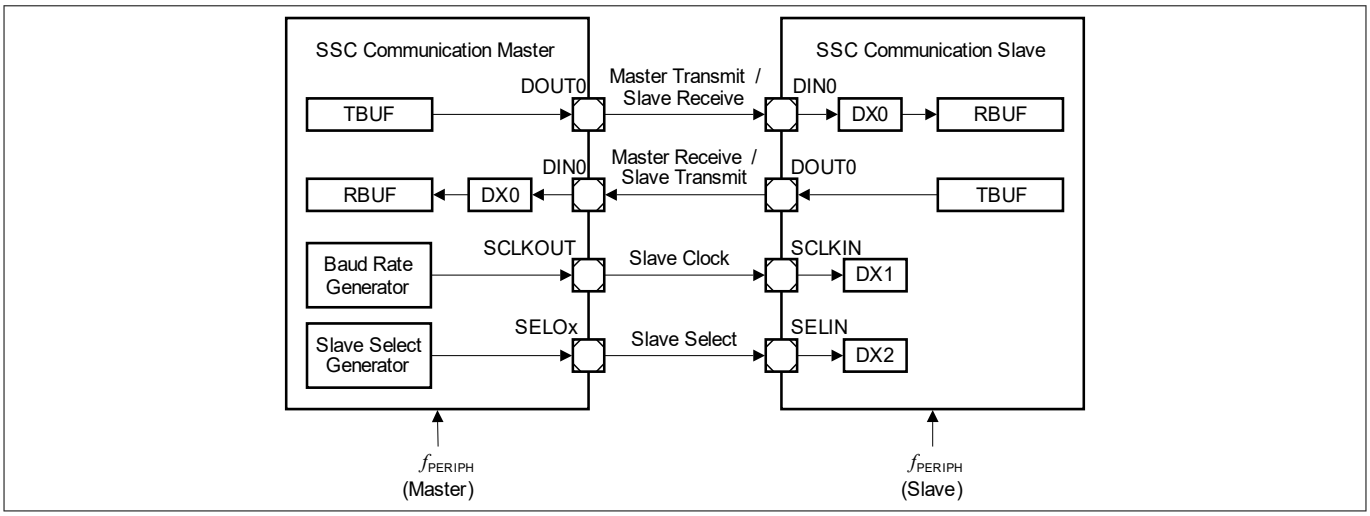


Figure 86 SSC Signals for Standard Full-Duplex Communication

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).

Figure 87 and Figure 88 show the pins that support standard SSC master and slave mode communications.

USIC0_CH0

Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40
P0.14	DX0A	57	45	39	P0.14	DOUT0	57	45	39	P0.7	SCLKOUT	48	36	30	P0.0	SELO0	41	29	23
P0.15	DX0B	58	46	40	P0.15	DOUT0	58	46	40	P0.8	SCLKOUT	51	39	33	P0.9	SELO0	52	40	34
P1.0	DX0C	34	26	22	P1.0	DOUT0	34	26	22	P0.14	SCLKOUT	57	45	39	P1.4	SELO0	30	22	18
P1.1	DX0D	33	25	21	P1.1	DOUT0	33	25	21	P1.6	SCLKOUT	28	20	16	P0.10	SELO1	53	41	35
P2.0	DX0E	9	3	1	P2.0	DOUT0	9	3	1	P2.0	SCLKOUT	9	3	1	P1.5	SELO1	29	21	17
P2.1	DX0F	10	4	2	P2.1	DOUT0	10	4	2						P0.11	SELO2	54	42	36
					P1.5	DOUT0	29	21	17						P1.6	SELO2	28	20	16
															P0.12	SELO3	55	43	37
															P0.13	SELO4	56	44	38

USIC0_CH1

Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40
P1.3	DX0A	31	23	19	P1.3	DOUT0	31	23	19	P0.8	SCLKOUT	51	39	33	P0.0	SELO0	41	29	23
P1.2	DX0B	32	24	20	P1.2	DOUT0	32	24	20	P1.3	SCLKOUT	31	23	19	P0.9	SELO0	52	40	34
P0.6	DX0C	47	35	29	P0.6	DOUT0	47	35	29	P1.4	SCLKOUT	30	22	18	P1.1	SELO0	33	25	21
P0.7	DX0D	48	36	30	P0.7	DOUT0	48	36	30	P2.1	SCLKOUT	10	4	2	P0.10	SELO1	53	41	35
P2.11	DX0E	20	14	12	P2.11	DOUT0	20	14	12	P2.11	SCLKOUT	20	14	12	P1.4	SELO1	30	22	18
P2.10	DX0F	19	13	11	P2.10	DOUT0	19	13	11						P0.11	SELO2	54	42	36
					P1.6	DOUT0	28	20	16						P1.5	SELO2	29	21	17
															P1.6	SELO3	28	20	16

USIC1_CH0

Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40
P0.2	DX0A	43	31	25	P0.2	DOUT0	43	31	25	P0.2	SCLKOUT	43	31	25	-	SELO0	37	-	-
P0.3	DX0B	44	32	26	P0.3	DOUT0	44	32	26	P2.12	SCLKOUT	21	15	-	P4.7	SELO0	4	2	-
P4.4	DX0C	63	47	-	P4.4	DOUT0	63	47	-	-	SCLKOUT	38	-	-	P3.0	SELO1	36	28	-
P4.5	DX0D	64	48	-	P4.5	DOUT0	64	48	-	-	SCLKOUT	40	-	-	P4.8	SELO1	5	-	-
-	DX0E	39	-	-	-	DOUT0	39	-	-	P4.3	SCLKOUT	62	-	-	P4.9	SELO2	6	-	-
-	DX0F	40	-	-	-	DOUT0	40	-	-	P4.5	SCLKOUT	64	48	-	-	SELO3	7	-	-
										P4.6	SCLKOUT	3	1	-	-	SELO4	8	-	-

USIC1_CH1

Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40
P0.0	DX0A	41	29	23	P0.0	DOUT0	41	29	23	P0.1	SCLKOUT	42	30	24	P0.4	SELO0	45	33	27
P0.1	DX0B	42	30	24	P0.1	DOUT0	42	30	24	P0.3	SCLKOUT	44	32	26	-	SELO0	39	-	-
P2.12	DX0C	21	15	-	P2.12	DOUT0	2	15	-	P1.8	SCLKOUT	26	-	-	-	SELO1	40	-	-
P2.13	DX0D	22	16	-	P2.13	DOUT0	22	16	-	P2.12	SCLKOUT	21	15	-	-	SELO1	59	-	-
P3.0	DX0E	36	28	-	P3.0	DOUT0	36	28	-	P3.0	SCLKOUT	36	28	-	P4.1	SELO2	60	-	-
-	DX0F	37	-	-	-	DOUT0	37	-	-	-	SCLKOUT	38	-	-	P4.2	SELO3	61	-	-

Figure 87 Available Pins for Standard SSC Master Mode Communication

USIC0_CH0

Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40
P0.14	DX0A	57	45	39	P0.14	DX1A	57	45	39	P0.0	DX2A	41	29	23	P0.14	DOUT0	57	45	39
P0.15	DX0B	58	46	40	P0.8	DX1B	51	39	33	P0.9	DX2B	52	40	34	P0.15	DOUT0	58	46	40
P1.0	DX0C	34	26	22	P0.7	DX1C	48	36	30	P0.10	DX2C	53	41	35	P1.0	DOUT0	34	26	22
P1.1	DX0D	33	25	21	P1.1	DX1D	33	25	21	P0.11	DX2D	54	42	36	P1.1	DOUT0	33	25	21
P2.0	DX0E	9	3	1	P2.0	DX1E	9	3	1	P0.12	DX2E	55	43	37	P2.0	DOUT0	9	3	1
P2.1	DX0F	10	4	2						P0.13	DX2F	56	44	38	P2.1	DOUT0	10	4	2
															P1.5	DOUT0	29	21	17

USIC0_CH1

Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40
P1.3	DX0A	31	23	19	P1.3	DX1A	31	23	19	P0.0	DX2A	41	29	23	P1.3	DOUT0	31	23	19
P1.2	DX0B	32	24	20	P0.8	DX1B	51	39	33	P0.9	DX2B	52	40	34	P1.2	DOUT0	32	24	20
P0.6	DX0C	47	35	29	P0.7	DX1C	48	36	30	P0.10	DX2C	53	41	35	P0.6	DOUT0	47	35	29
P0.7	DX0D	48	36	30	P2.11	DX1D	20	14	12	P0.11	DX2D	54	42	36	P0.7	DOUT0	48	36	30
P2.11	DX0E	20	14	12						P1.1	DX2E	33	25	21	P2.11	DOUT0	20	14	12
P2.10	DX0F	19	13	11						P2.0	DX2F	9	3	1	P2.10	DOUT0	19	13	11
															P1.6	DOUT0	28	20	16

USIC1_CH0

Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40
P0.2	DX0A	43	31	25	P0.2	DX1A	43	31	25	P4.7	DX2A	4	2	-	P0.2	DOUT0	43	31	25
P0.3	DX0B	44	32	26	P4.3	DX1B	62	-	-	P4.8	DX2B	5	-	-	P0.3	DOUT0	44	32	26
P4.4	DX0C	63	47	-	P4.5	DX1C	64	48	-	P4.9	DX2C	6	-	-	P4.4	DOUT0	63	47	-
P4.5	DX0D	64	48	-	P4.6	DX1D	3	1	-	-	DX2D	7	-	-	P4.5	DOUT0	64	48	-
-	DX0E	39	-	-	-	DX1E	40	-	-	-	DX2E	8	-	-	-	DOUT0	39	-	-
-	DX0F	40	-	-						-	DX2F	37	-	-	-	DOUT0	40	-	-

USIC1_CH1

Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40
P0.0	DX0A	41	29	23	P0.1	DX1A	42	30	24	-	DX2A	39	-	-	P0.0	DOUT0	41	29	23
P0.1	DX0B	42	30	24	P2.12	DX1B	21	15	-	-	DX2B	40	-	-	P0.1	DOUT0	42	30	24
P2.12	DX0C	21	15	-	-	DX1C	26	-	-	-	DX2C	27	-	-	P2.12	DOUT0	21	15	-
P2.13	DX0D	22	16	-	P3.0	DX1D	36	28	-						P2.13	DOUT0	22	16	-
P3.0	DX0E	36	28	-											P3.0	DOUT0	36	28	-
-	DX0F	37	-	-											-	DOUT0	37	-	-

Figure 88 Available Pins for Standard SSC Slave Mode Communication

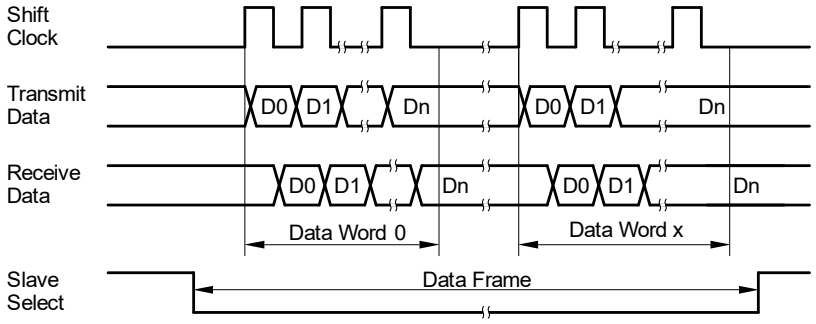


Figure 89 4-Wire SSC Standard Communication Signals

16.4.1.1 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 (for example, 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 86.

Figure 90 shows the SSC data signal paths for both Master and Slave modes.

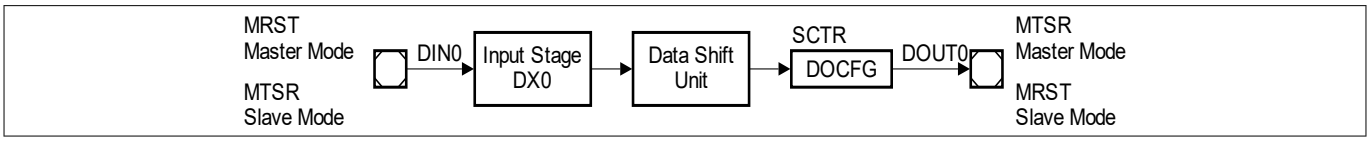


Figure 90 SSC Data Signals

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.

***Note**: 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.*

16.4.1.2 Shift Clock Signals

Figure 91 shows the SSC shift clock signal paths for both Master and Slave modes.

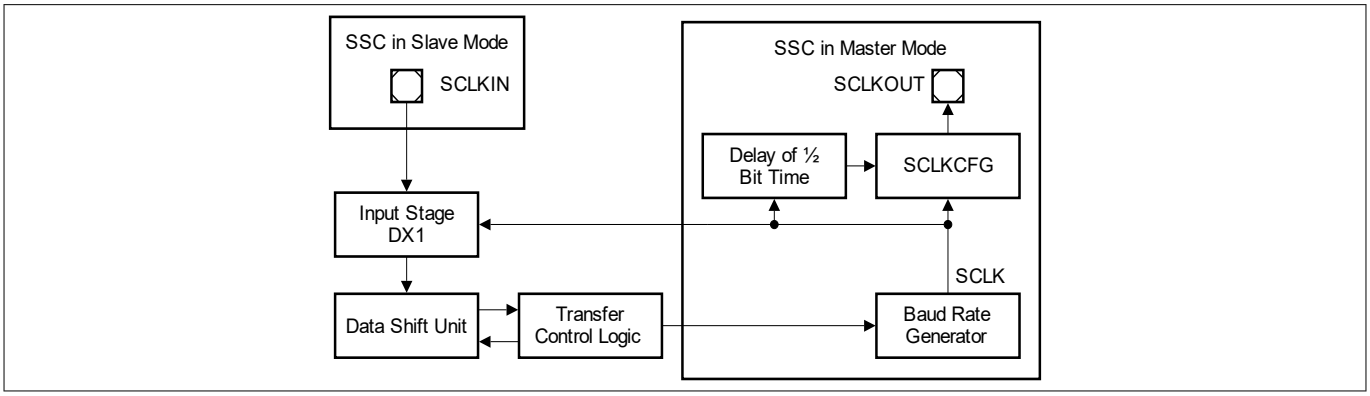


Figure 91 SSC Shift Clock Signals

Master Mode

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.

Shift Clock Configuration in Master Mode

Due to the multitude of different SSC applications, 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, which are described in Figure 92 and Table 85.

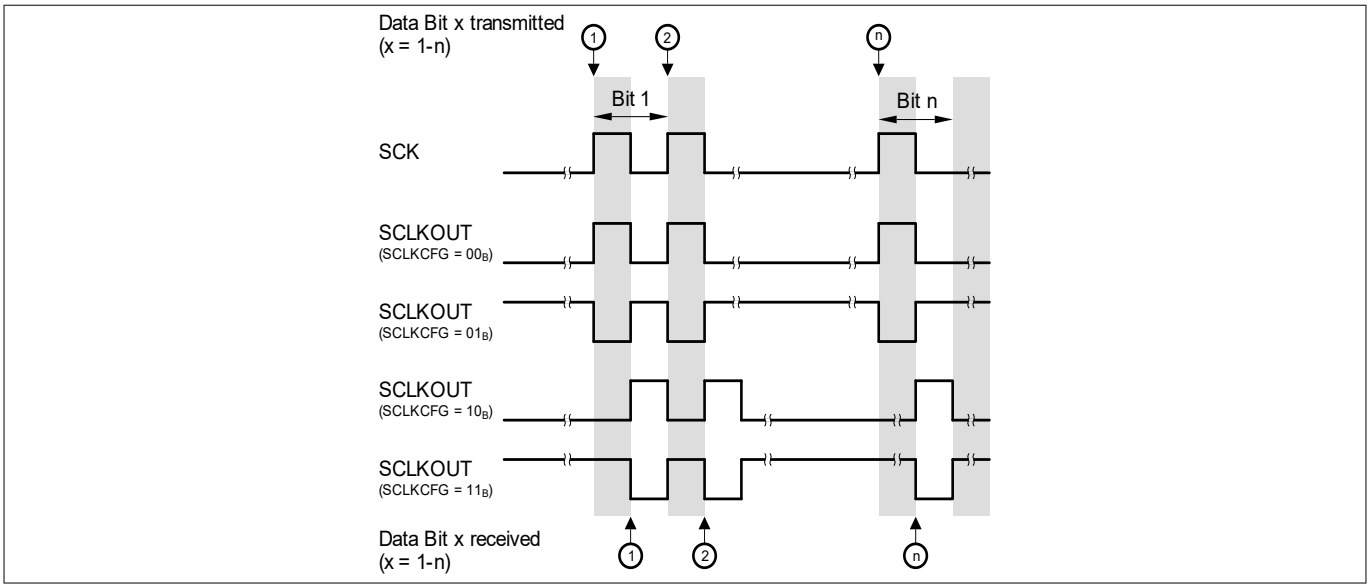


Figure 92 SCLKOUT Configuration in SSC Master Mode

Table 85 Shift Clock Configuration in Master Mode

SCLKCFG	Description
00_B	**No delay, no polarity inversion (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 corresponds to the behavior of the internal data shift unit.
01_B	**No delay, with polarity inversion** • 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.

(table continues...)

Table 85 (continued) Shift Clock Configuration in Master Mode

SCLKCFG	Description
10_B	**SCLKOUT delayed by 1/2 shift clock period, no polarity inversion** • 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. ***Note**: The connected slave has to provide the first data bit before the first SCLKOUT edge, for example, as soon as it is addressed by its slave select.*
11_B	**SCLKOUT delayed by 1/2 shift clock period, with polarity inversion** • 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. ***Note**: The connected slave has to provide the first data bit before the first SCLKOUT edge, for example, as soon as it is addressed by its slave select.*

Slave Mode

In **slave mode**, the shift clock signal is handled by the input stage DX1.

The signal SCLKIN is received from an external master, so the DX1 stage has to be connected to an input pin.

For complete closed loop delay compensation in slave mode, the transmit shift clock from the input stage DX1 can be additionally routed to the SCLKOUT signal, which is otherwise not used in the slave mode. The selection is done through the bit BRG.SCLKOSEL. See Complete Closed-loop Delay Compensation for details.

Supporting Different Shift Clock Configurations in Slave Mode

The DX1 input stage can invert the received input signal with bit DX1CR.DPOL 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 addition, 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.

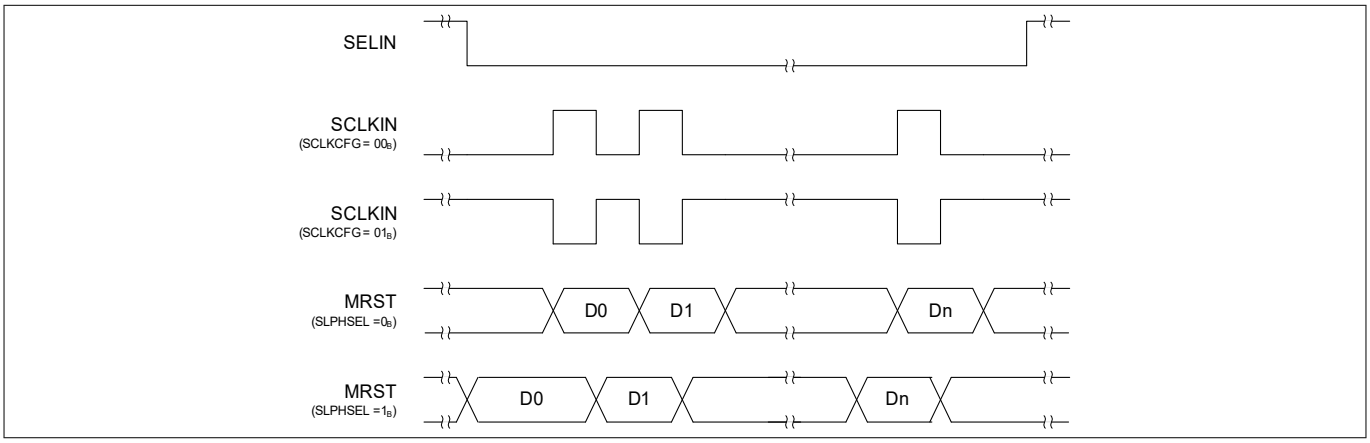


Figure 93 SLPHSEL Configuration in SSC Slave Mode

16.4.1.3 Slave Select Signals

Figure 94 shows the SSC slave select signal paths for both Master and Slave modes.

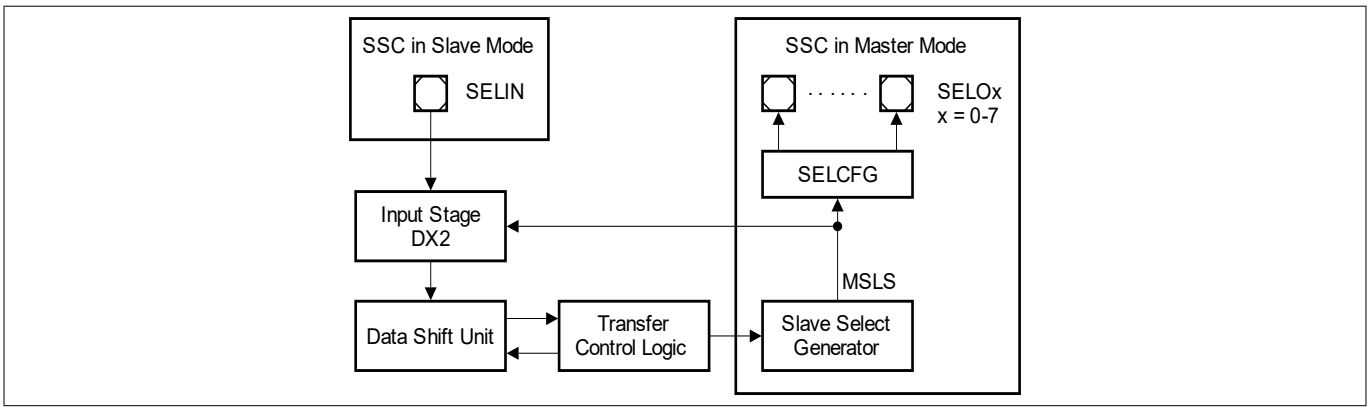


Figure 94 SSC Slave Select Signals

Master Mode

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).

The control of the SELCFG block is based on protocol specific bits and bit fields in the protocol control register PCR:

- 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

For the generation of the MSLS signal please refer to MSLS Generation and Slave Select Delays.

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 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 fPERIPH compared to MSLS to allow the other SELOx lines to stabilize before enabling the address decoder.

Slave Mode

In **slave mode**, the slave select signal is handled by the input stage DX2. 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.

16.4.2 Operating the SSC

This chapter contains SSC issues, that are of general interest and not directly linked to either master mode or slave mode.

16.4.2.1 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 TBUFx after the receiver start interrupt has been generated.

16.4.2.2 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.

16.4.2.3 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 and waiting for a currently running word transfer to finish.

16.4.2.4 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.

16.4.2.5 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 95 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.*

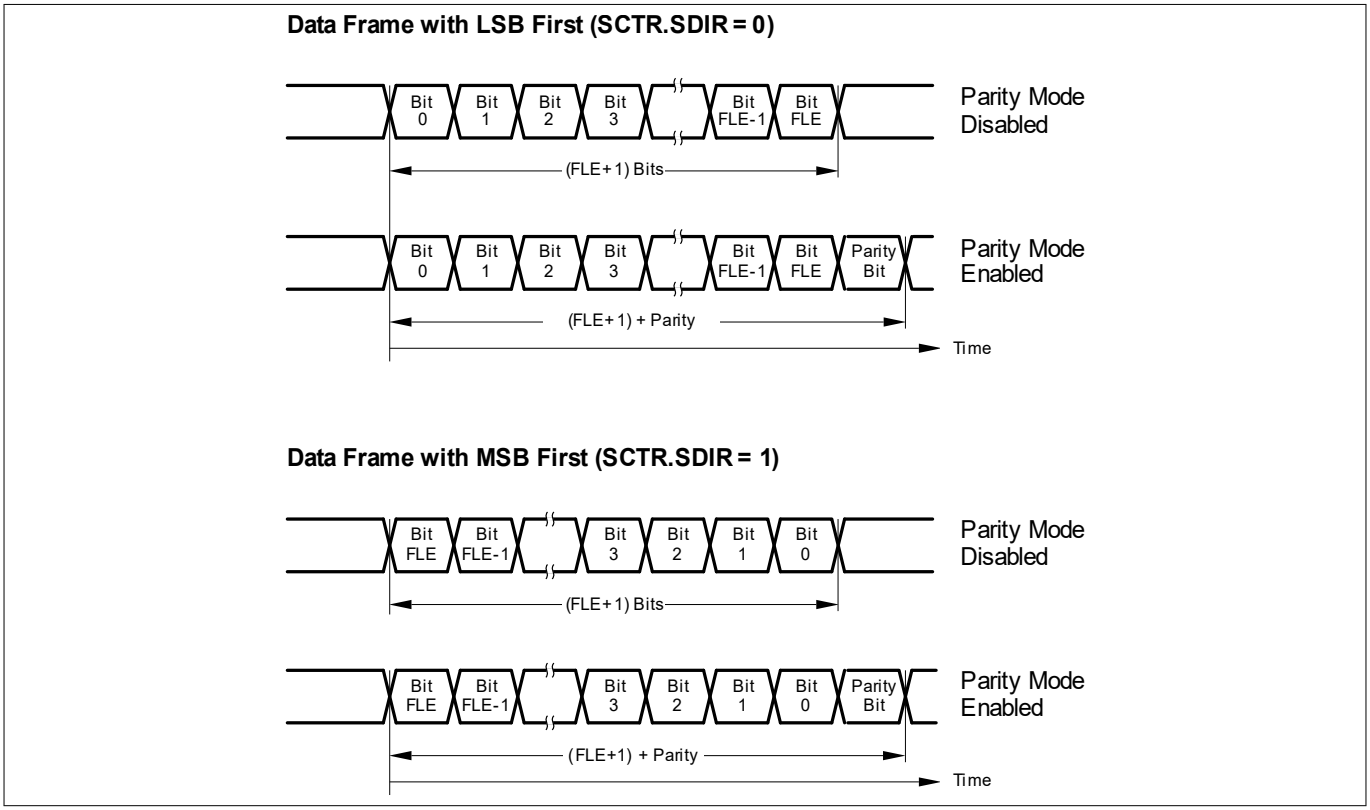


Figure 95 Data Frames without/with Parity

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.
- When in slave mode, the content of the TBUF is not valid at the transmission start. In this case, the slave outputs the level defined by SCTR.PDL including for the parity bit. This might result in the detection of a parity error at the master.

16.4.2.6 Data Transfer Interrupt Handling

The data transfer interrupts indicate events related to SSC frame handling.

Table 86 SSC Data transfer interrupt handling

Interrupt	Indicated by bit	Description
Transmit buffer interrupt	PSR.TBIF	Set after the start of first data bit of a data word.
Transmit shift interrupt	PSR.TSIF	Set after the start of the last data bit of a data word.
Receiver start interrupt	PSR.RSIF	Set after the reception of the 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.
Receiver interrupt	PSR.RIF	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	PSR.AIF	The reception of the first word in a frame is always indicated by RBUFSR.SOF = 1. Bit PSR.AIF is set after the reception of the last data bit of a data word if RBUFSR.SOF = 1. This is true both in case of reception of multi-word frames and single-word frames.
Data lost interrupt	PSR.DLIF	Set 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.

16.4.2.7 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.PARERR register bit field.*

16.4.2.8 Receive Buffer Handling

If the FIFO buffer is 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.

16.4.3 Operating the SSC in Master Mode

In order to operate the SSC in master mode, the USIC channel has to be first initialized.

It is recommended to configure all parameters of the SSC that do not change during run time while CCR.MODE = 0000_B, except stated otherwise.

The main initialization steps are outlined below:

- **Enable USIC channel**
  - Enable the module by writing 1s to MODEN and BPMODEN bits in KSCFG register.
- **Configure baud rate generator**
  - Select either fractional divider mode or normal divider mode with FDR.DM bit.
  - Configure the baud rate setting through register BRG and FDR.STEP bit field.
- **Configure input pins**
  - Establish a connection of input stage DX0 with the receive data input pin (signal DIN0) selected by DX0CR.DSEL bit field and with bit DX0CR.INSW = 1.
  - Program Bit DX1CR.INSW to use the baud rate generator output SCLK directly as input for the data shift unit.
  - Program Bit DX2CR.INSW to use the slave select generator output MSLS as input for the data shift unit.
- **Configure data format**
  - The word length, the frame length, the shift direction and the shift mode have to be set up according to the application requirements by programming the register SCTR.
  - Write SCTR.TRM = 01_B to enable SSC data transfers.
- **Configure data transfer parameters**
  - Write TCSR.TDSSM = 1 and TCSR.TDEN = 01_B to enable data transmission in single shot mode.
- **Configure protocol control parameters**
  - Enable slave select generation by setting bit field PCR.MSLSEN = 1.
  - Select the mode, the polarity and the active slave select output signal through the bit fields SELCTR, SELINV and SELO in PCR register respectively.
  - Configure the slave select delays if necessary, through registers BRG and PCR, see Configuration of Slave Select Delays.
  - Select the frame end mode with the bit field PCR.FEM.

- **Select SSC protocol**
  - Enable SSC mode with CCR.MODE = 0001_B.
- **Configure output pins**
  - Configure the transmit data (signal DOUT0), the shift clock (signal SCLKOUT) and slave select (signal SELOx) output pins through the selected pins' port control registers Pn_IOCR0/4/8/12. Refer to the port chapter.
  - The step to enable the output pin functions should only be done after the SSC mode is enabled with CCR.MODE, to avoided unintended spikes on the output.

An initialization code example is given in Initialization Code Example.

***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.*

16.4.3.1 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.

The standard setting is given by PPPEN = 0 ($f_{PPP} = f_{PIN}$) and CLKSEL = 00_B ($f_{PIN} = f_{FD}$). Under these conditions, the baud rate is given by either (23) or (24), depending on the selected divider mode:

- In normal divider mode (FDR.DM = 01_B)

$$f_{SCLK} = \frac{1}{2} \times f_{PERIPH} \times \frac{1}{1024 - STEP} \times \frac{1}{PDIV + 1} \tag{23}$$

- In fractional divider mode (FDR.DM = 10_B)

$$f_{SCLK} = \frac{1}{2} \times f_{PERIPH} \times \frac{STEP}{1024} \times \frac{1}{PDIV + 1} \tag{24}$$

Table 87 shows examples of the baud rate calculation in fractional divider mode (FDR.DM = 10_B).

Table 87 SSC Baud Rate Calculation in Fractional Divider Mode

f_{PERIPH} (MHz)	FDR.STEP	BRG.PDIV	f_{SCLK} (kbit/s)	Deviation Error
48	256	624	9.6	0.00%
48	512	23	500	0.00%
48	512	7	1500	0.00%
48	512	0	12000	0.00%

16.4.3.2 MSLS Generation and Slave Select Delays

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}**).

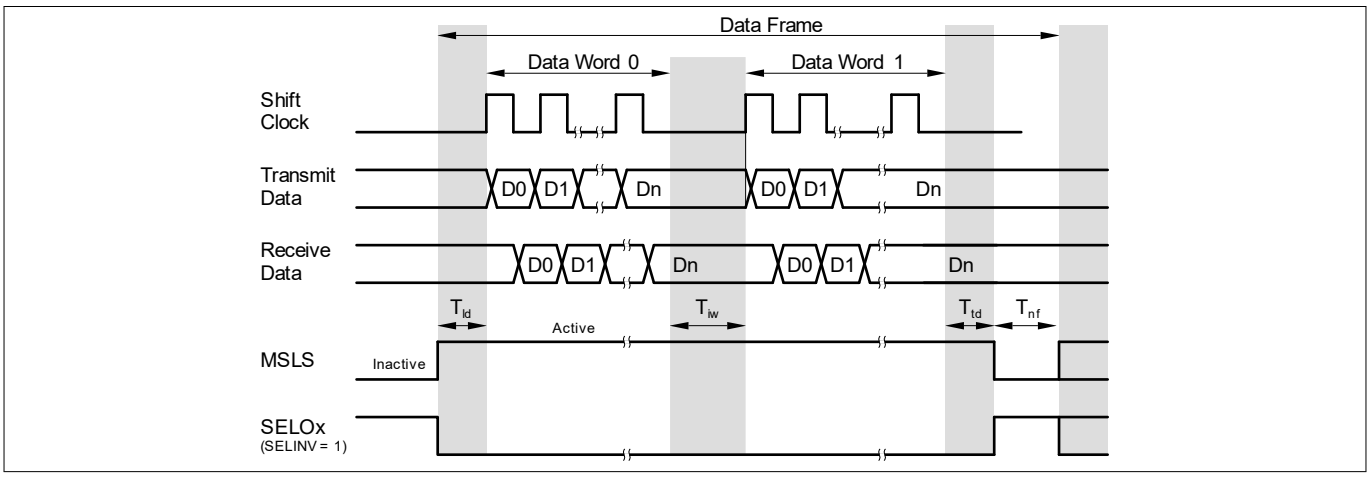


Figure 96 MSLS Generation in SSC Master Mode

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.

16.4.3.3 Configuration of Slave Select Delays

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 Ti_w 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 (for example, based on bit times or on the faster inputs of the protocol-related divider). The delay times are defined as follows:

$$\begin{aligned} T_{Id} = T_{td} &= \frac{(PCTQ + 1) \times (DCTQ + 1)}{f_{CTQIN}} \\ T_{iw} = T_{nf} &= \frac{(PCTQ1 + 1) \times (DCTQ1 + 1)}{f_{CTQIN}} \end{aligned} \tag{25}$$

16.4.3.4 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 Automatic Shadow Mechanism).

16.4.3.5 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.

Table 88 SSC master mode protocol interrupt events

Events	Event flag	Description	Flag clear	Interrupt enable
MSLS interrupt	PSR.MSLSEV	Set with any change of the internal MSLS signal in maser mode (MSLS generation enabled). This event indicates that a data frame has been started (activation of MSLS) or finished (deactivation of MSLS). 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.	PSCR.CST2	PCR.MSLSIEN
DX2T interrupt	PSR.DX2TEV	Set after an activation of the trigger signal DX2T. This event monitors edges of the input signal of the DX2 stage (although this signal is not used as slave select input for data transfers). 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.	PSCR.CST3	PCR.DX2TIEN
Parity error interrupt	PSR.PARERR	Set if there is a mismatch between the received parity bit (in RBUFSR.PAR) and the calculated parity bit, of the data frame.	PSCR.CST4	PCR.PARIEN

16.4.3.6 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 TBUFx (x = 00 to 31) 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 TBUFx 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 TBUFx 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 TBUFx address location defines the word length and the end of a frame (locations TBUF16 to TBUF31 lead to a frame end).

  For example, writing transmit data to TBUF07 results in a data word of 8-bit length without finishing the frame, whereas writing transmit data to TBUF31 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 TBUFx, but to the FIFO input locations INx (x = 00 to 31) 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}.

16.4.3.7 Data Flow Handling

Data transmission is initiated by loading the transmit buffer TBUF with the data word to be transmitted, through one of the transmit buffer input locations (TBUFx). If the transmit FIFO is used, one of the transmit FIFO buffer input locations (INx) should be used instead of TBUFx.

If the transmitter is not busy transmitting a previous data word, the MSLS signal becomes active to indicate a start of frame through the MSLS interrupt event (PSR.MSLSEV = 1).

The data word will then be updated to the transmit shift register. This is indicated by a transmit buffer interrupt event (PSR.TBIF = 1).

In the same clock cycle that the first data bit is placed onto the output line MTSR but in the opposite clock edge, the first data bit on the input line MRST is latched and shifted into the available receive shift register. This is indicated by a receive start interrupt event (PSR.RSIF = 1). The receive start interrupt event can be used to indicate that the next data word can now be loaded to TBUF.

For each data word, the start of transmission of the last data bit is indicated by a transmit shift interrupt event (PSR.TSIF = 1).

When the number of bits corresponding to the data word length is received, the contents of the receive shift register are transferred to the corresponding receive buffer (RBUF0 or RBUF1). This is indicated by an alternate receive interrupt event (PSR.AIF) if the received word is the first word of the frame, or by a receive interrupt event (PSR.RIF) if the received word represents subsequent words of the frame.

After the last word of the frame has been completely transmitted, the MSLS signal becomes inactive to indicate an end of frame. A MSLS interrupt event (PSR.MSLSEV = 1) is again triggered.

Figure 97 shows the simplified data transmission and reception flows with TBUFx and RBUF. For examples with FIFO buffer, refer to Transmit FIFO Buffer Usage Example and Receive FIFO Buffer in Filling Level Mode Usage Example.

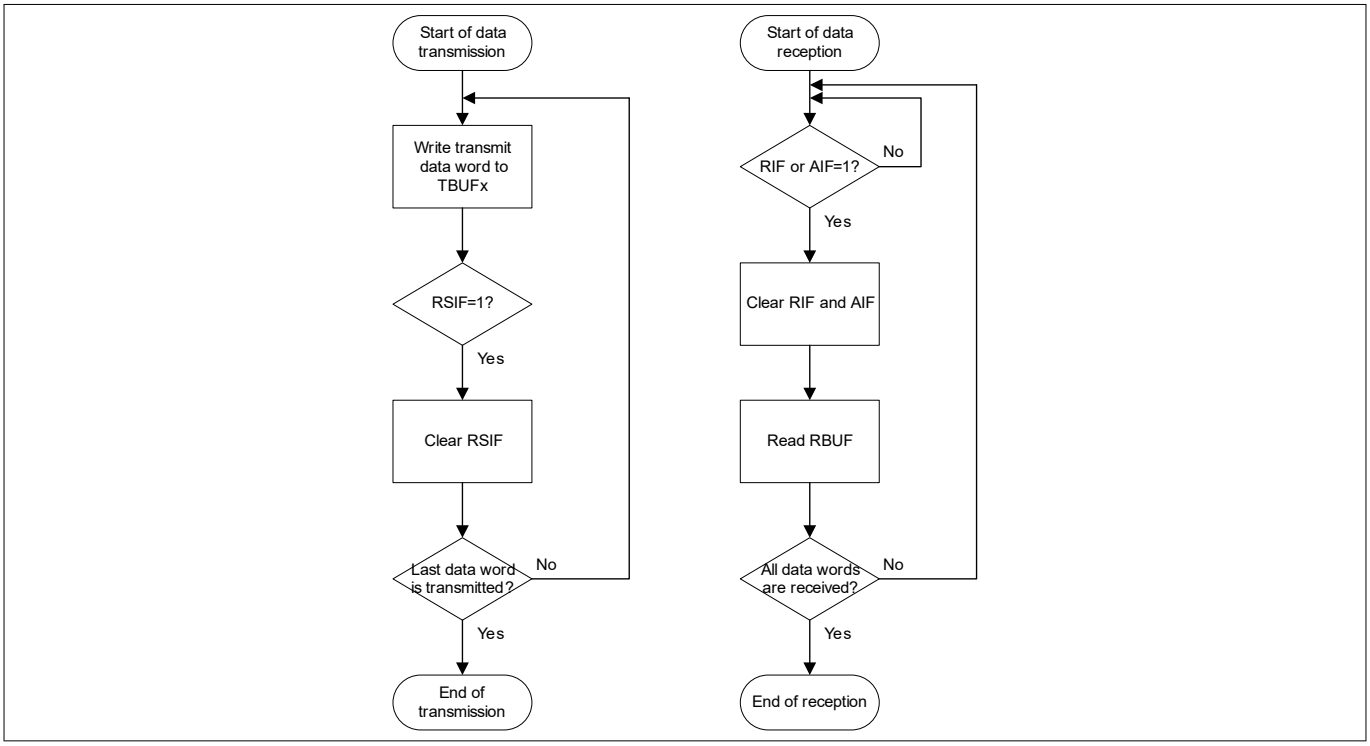


Figure 97 Simplified Data Transmission and Reception Flow

Figure 98 shows the interrupt events during a SSC master mode data transfer.

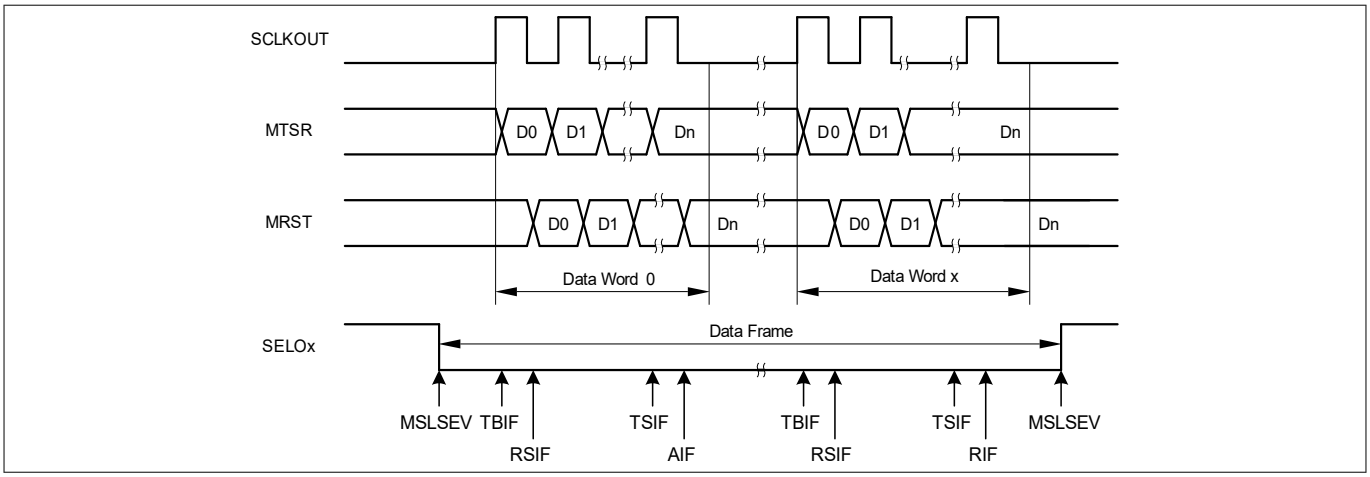


Figure 98 Interrupt Events on Data Transfer

16.4.3.8 Initialization Code Example

The following code example implements a function to initialize a USIC channel for SSC master mode data transfers with a baud rate of 1500 kbit/s (f_{PERIPH} = 48 MHz):

```
void USIC0_CH1_SSC_MasterMode_Init(void)
{
/// ----------------------------------------------------------
/// 1.Enable USIC0 channel 1
/// ----------------------------------------------------------
// BPMODEN MODEN
USIC0_CH1->KSCFG|= (1 << 1)| (1 << 0);
/// ----------------------------------------------------------
/// 2.Configure baud rate generator
/// - Fractional divider mode
/// - Baud rate = 1500 kbit/s
/// ----------------------------------------------------------
// DM STEP
USIC0_CH1->FDR = (2 << 14) | (512 << 0);
// PDIV
USIC0_CH1->BRG = (7 << 16);
/// ----------------------------------------------------------
/// 3.Configure input stages
/// - Select input DX0C
/// - Derive input of data shift unit directly from input pin
/// ----------------------------------------------------------
// INSW DSEL
USIC0_CH1->DX0CR =(1 << 4) |(2 << 0);
/// ----------------------------------------------------------
/// 4.Configure data format
/// - Data word = data frame = 15 bits
/// - Data transfer allowed with passive level = 1 and MSB first
/// ----------------------------------------------------------
// WLE FLE TRM PDL SDIR
USIC0_CH1->SCTR =(15<<24)|(15<<16)|(1<<8)|(1<<1)|(1<<0);
/// ----------------------------------------------------------
/// 5.Configure data transfer parameters
/// - Single shot transmission of data word when a valid word
/// is available
/// ----------------------------------------------------------
// TDEN TDSSM
USIC0_CH1->TCSR =(1 << 10) |(1 << 8);
/// ----------------------------------------------------------
/// 6.Configure SSC protocol-specific parameters
/// - Slave select generation is enabled
/// - Direct slave select mode with inversion is selected
/// - End of frame condition is required for the frame to be
/// considered as finished
/// - SELO0 is selected as the active select signal
/// ----------------------------------------------------------
// SELO FEM SELINV SELCTR MSLSEN
USIC0_CH1->PCR = (1<<16)|(1<<3)|(1<<2)|(1<<1)|(1<<0);
/// ----------------------------------------------------------
```

```
/// 7.Enable SSC protocol
/// ------------------------------------------------------------
// MODE
USIC0_CH1->CCR =(1 << 0);
/// ------------------------------------------------------------
/// 8.Configure SSC output function pins
/// - Assume
/// - P0.7 ALT7 function is assigned to DOUT0
/// - P0.8 ALT7 function is assigned to SCLKOUT
/// - P0.9 ALT7 function is assigned to SELO0
/// ------------------------------------------------------------
// PC7
PORT0->IOCR4 |= (0x27 << 26);
// PC9 PC8
PORT0->IOCR8 |= (0x27 << 10) | (0x27 << 2);
}
```

16.4.4 Operating the SSC in Slave Mode

In order to operate the SSC in slave mode, the USIC channel has to be first initialized.

It is recommended to configure all parameters of the SSC that do not change during run time while CCR.MODE = 0000_B, except stated otherwise.

The main initialization steps are outlined below:

- **Enable USIC channel**
  - Enable the module by writing 1s to MODEN and BPMODEN bits in KSCFG register.
- **Configure input pins**
  - Establish a connection of input stage DX0 with the receive data input pin (signal DIN0) selected by DX0CR.DSEL bit field and with bit DX0CR.INSW = 1.
  - Establish a connection of input stage DX1 with the shift clock input pin (signal SCLKIN) selected by DX1CR.DSEL bit field and with DX1CR.INSW = 1.
  - Establish a connection of input stage DX2 with the slave select input pin (signal SELIN) selected by DX2CR.DSEL bit field and 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.
- **Configure data format**
  - The word length, the frame length, the shift direction and the shift mode have to be set up according to the application requirements by programming the register SCTR.
  - Write SCTR.TRM = 01_B to enable SSC data transfers.
- **Configure data transfer parameters**
  - Write TCSR.TDSSM = 1 and TCSR.TDEN = 01_B to enable data transmission in single shot mode.

- **Select SSC protocol**
  - Enable SSC mode with CCR.MODE = 0001_B.
- **Configure output pins**
  - Configure the transmit data (signal DOUT0) output pin through the selected pin's port control register Pn_IOCR0/4/8/12. Refer to the port chapter.
  - The step to enable the output pin functions should only be done after the SSC mode is enabled with CCR.MODE, to avoided unintended spikes on the output.

An initialization code example is given in Initialization Code Example.

***Note**: In SSC slave mode, the baud rate generator and the slave select generation are not needed and can be switched off. All related bit fields (for example, FDR.DM, PCR.MSLSEN, etc.) can be programmed to 0.*

16.4.4.1 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.

Table 89 SSC slave mode protocol interrupt events

Events	Event flag	Description	Flag clear	Interrupt enable
DX2T interrupt	PSR.DX2TEV	Set after an activation of the trigger signal DX2T. This event monitors edges of the input signal of the DX2 stage (although this signal is not used as slave select input for data transfers). 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.	PSCR.CST3	PCR.DX2TIEN
Parity error interrupt	PSR.PARERR	Set if there is a mismatch between the received parity bit (in RBUFSR.PAR) and the calculated parity bit, of the data frame.	PSCR.CST4	PCR.PARIEN

16.4.4.2 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.

16.4.4.3 Data Flow Handling

To prepare the SSC slave for data transfer, the first data word is written to one of the transmit buffer input locations (TBUFx) by software. If the transmit FIFO is used, one of the transmit FIFO buffer input locations (INx) should be used instead of TBUFx.

Then slave then waits for the SSC master to initiate the data transfer. Once the data transfer is started, the data word will then be updated to the transmit shift register. This is indicated by a transmit buffer interrupt event (PSR.TBIF = 1).

In the same clock cycle that the first data bit is placed onto the output line MTSR but in the opposite clock edge, the first data bit on the input line MRST is latched and shifted into the available receive shift register. This is indicated by a receive start interrupt event (PSR.RSIF = 1). The receive start interrupt event can be used to indicate that the next data word can now be loaded to TBUF.

For each data word, the start of transmission of the last data bit is indicated by a transmit shift interrupt event (PSR.TSIF = 1).

When the number of bits corresponding to the data word length is received, the contents of the receive shift register are transferred to the corresponding receive buffer (RBUF0 or RBUF1). This is indicated by an alternate receive interrupt event (PSR.AIF) if the received word is the first word of the frame, or by a receive interrupt event (PSR.RIF) if the received word represents subsequent words of the frame.

Figure 99 shows the simplified data transmission and reception flows with TBUFx and RBUF. For examples with FIFO buffer, refer to Transmit FIFO Buffer Usage Example and Receive FIFO Buffer in Filling Level Mode Usage Example.

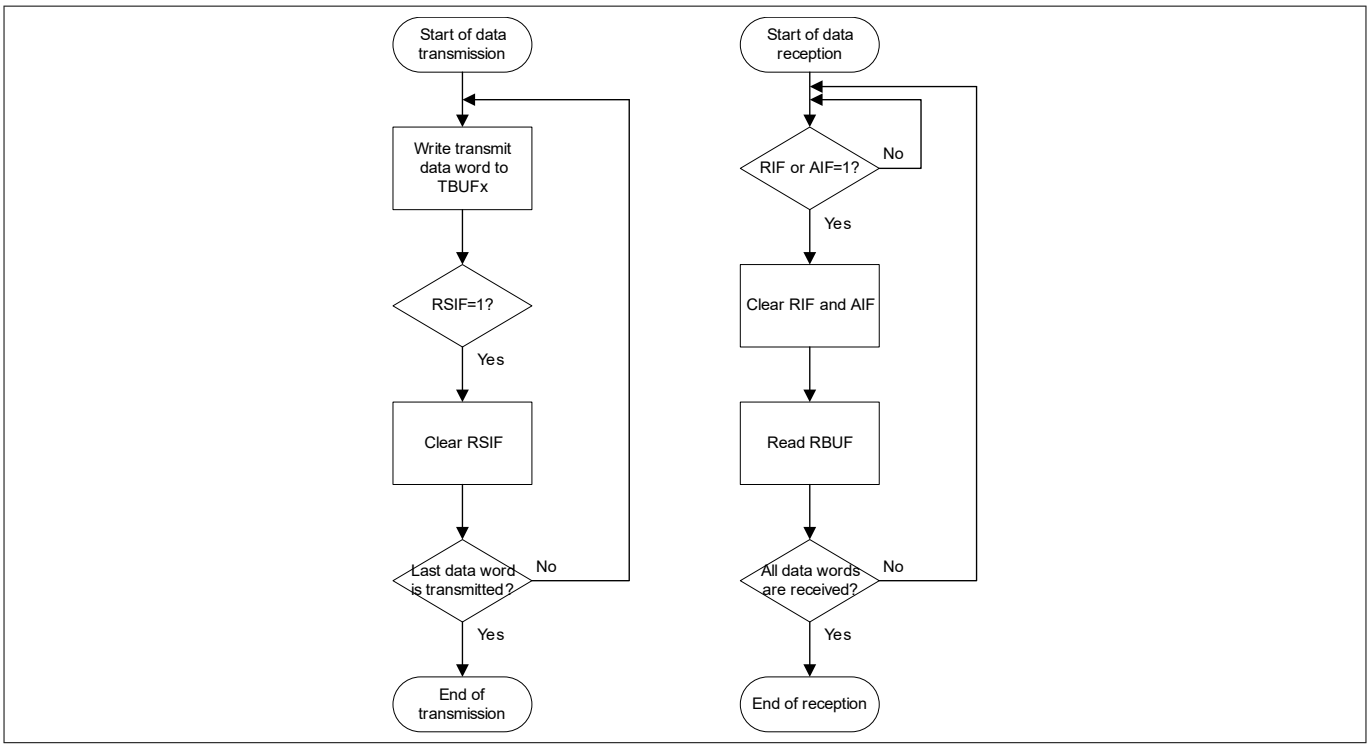


Figure 99 Simplified Data Transmission and Reception Flow

Figure 100 shows the interrupt events during a SSC slave mode data transfer.

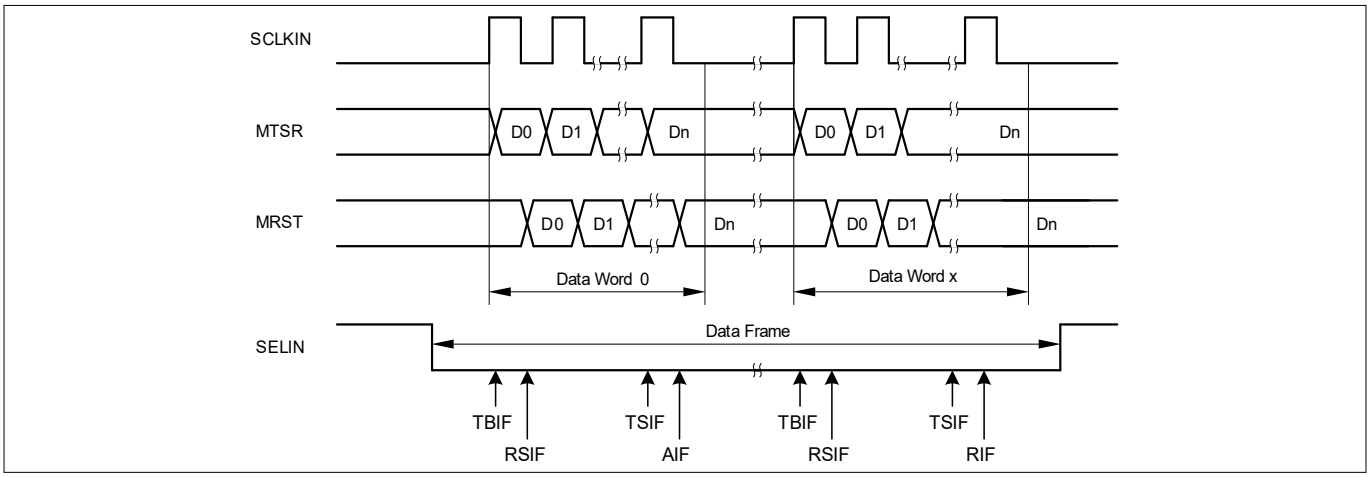


Figure 100 Interrupt Events on Data Transfer

16.4.4.4 Initialization Code Example

The following code example implements a function to initialize a USIC channel for SSC slave mode data transfers:

```
void USIC0_CH1_SSC_SlaveMode_Init(void)
{
/// ----------------------------------------------------------
/// 1.Enable USIC0 channel 1
/// ----------------------------------------------------------
// BPMODEN MODEN
USIC0_CH1->KSCFG|= (1 << 1)| (1 << 0);
/// ----------------------------------------------------------
/// 2.Configure input stages
/// - Select inputs DX0D, DX1B and DX2B
/// - Derive all input signals directly from input pins
/// ----------------------------------------------------------
// INSW DSEL
USIC0_CH1->DX0CR =(1 << 4) |(3 << 0);
// INSW DSEL
USIC0_CH1->DX1CR =(1 << 4) |(1 << 0);
// DPOL INSW DSEL
USIC0_CH1->DX2CR =(1 << 8) |(1 << 4) |(1 << 0);
/// ----------------------------------------------------------
/// 3.Configure data format
/// - Data word = data frame = 15 bits
/// - Data transfer allowed with passive level = 1 and MSB first
/// ----------------------------------------------------------
// WLE FLE TRM PDL SDIR
USIC0_CH1->SCTR =(15<<24)|(15<<16)|(1<<8)|(1<<1)|(1<<0);
/// ----------------------------------------------------------
/// 4.Configure data transfer parameters
/// - Single shot transmission of data word when a valid word
/// is available
/// ----------------------------------------------------------
// TDEN TDSSM
USIC0_CH1->TCSR =(1 << 10) |(1 << 8);
/// ----------------------------------------------------------
/// 5.Enable SSC protocol
/// ----------------------------------------------------------
// MODE
USIC0_CH1->CCR =(1 << 0);
/// ----------------------------------------------------------
/// 6.Configure SSC output function pins
/// - Assume P0.6 ALT7 function is assigned to DOUT0
/// ----------------------------------------------------------
// PC6
PORT0->IOCR4 |= (0x27 << 18);
}
```

16.4.5 Multi-IO SSC Protocols

Multi-IO SSC protocols, or specifically Dual-SSC and Quad-SSC, are extensions of the standard SSC protocol to double and quadruple the effective data transfer rates. This is achieved by transmitting and receiving data through two or four IO lines in parallel, in a half-duplex configuration.

Figure 101 shows the connections between a SSC master and a SSC slave, in Quad-SSC configurations. For Dual-SSC, the DX4/DOUT2 and DX5/DOUT3 pins are not used.

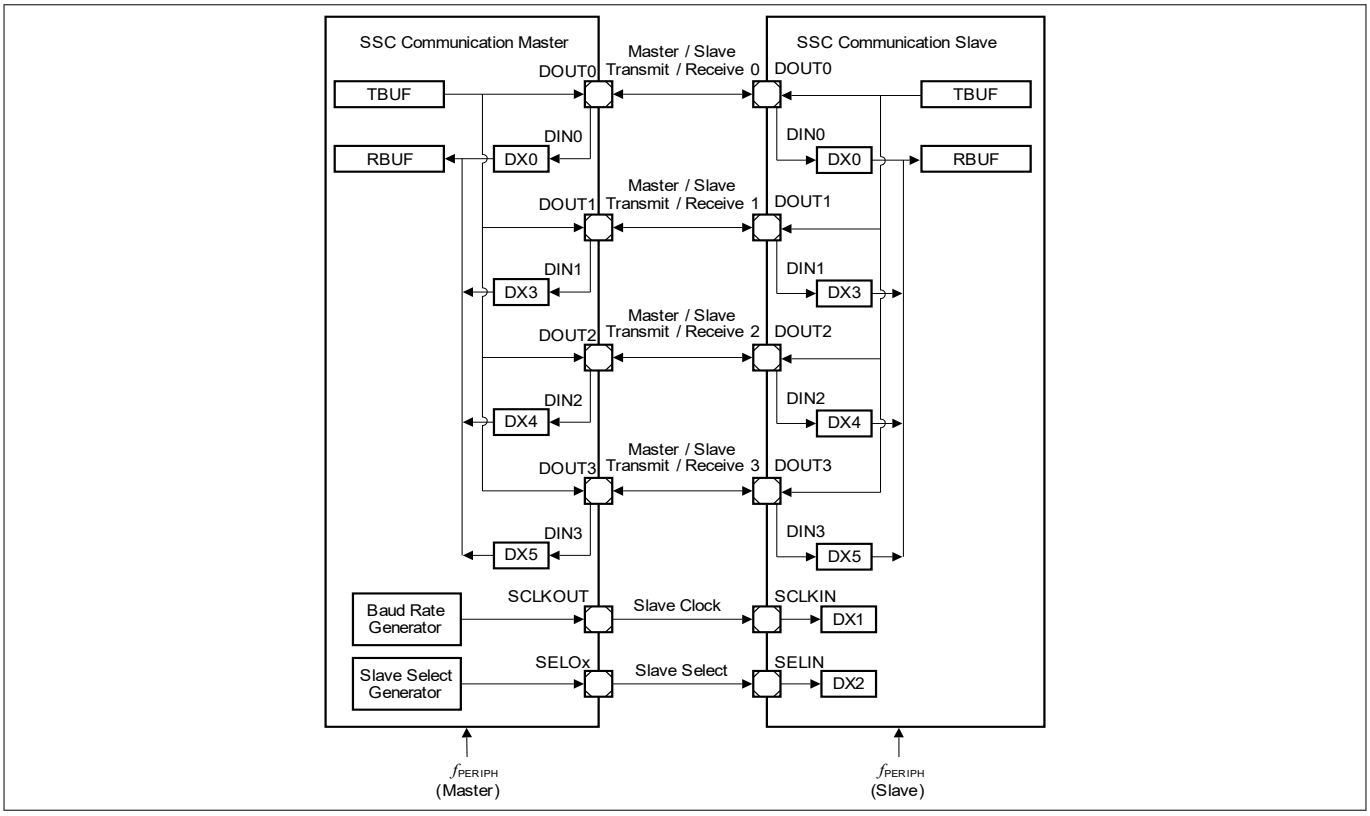


Figure 101 Quad-SSC Signal Connections

Figure 102 and Figure 103 show the pins that support the multi-IO SSC master and slave functions respectively.

USIC0_CH0

Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40
P1.0	HWIN0 (DX0)	34	26	22	P1.0	DOUT0	34	26	22	P0.7	SCLKOUT	48	36	30	P0.0	SELO0	41	29	23
P1.1	HWIN1 (DX3)	33	25	21	P1.1	DOUT1	33	25	21	P0.8	SCLKOUT	51	39	33	P0.9	SELO0	52	40	34
P1.2	HWIN2 (DX4)	32	24	20	P1.2	DOUT2	32	24	20	P0.14	SCLKOUT	57	45	39	P1.4	SELO0	30	22	18
P1.3	HWIN3 (DX5)	31	23	19	P1.3	DOUT3	31	23	19	P1.6	SCLKOUT	28	20	16	P0.10	SELO1	53	41	35
										P2.0	SCLKOUT	9	3	1	P1.5	SELO1	29	21	17
															P0.11	SELO2	54	42	36
															P1.6	SELO2	28	20	16
															P0.12	SELO3	55	43	37
															P0.13	SELO4	56	44	38

USIC1_CH0

Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40
-	HWIN0 (DX0)	40	-	-	-	DOUT0	40	-	-	P0.2	SCLKOUT	43	31	25	-	SELO0	37	-	-
-	HWIN1 (DX3)	39	-	-	-	DOUT1	39	-	-	P2.12	SCLKOUT	21	15	-	P4.7	SELO0	4	2	-
-	HWIN2 (DX4)	38	-	-	-	DOUT2	38	-	-	-	SCLKOUT	38	-	-	P3.0	SELO1	36	28	-
-	HWIN3 (DX5)	37	-	-	-	DOUT3	37	-	-	-	SCLKOUT	40	-	-	P4.8	SELO1	5	-	-
										P4.3	SCLKOUT	62	-	-	P4.9	SELO2	6	-	-
										P4.5	SCLKOUT	64	48	-	-	SELO3	7	-	-
										P4.6	SCLKOUT	3	1	-	-	SELO4	8	-	-

Figure 102 Available Pins for Multi-IO SSC Master Mode Communication

USIC0_CH0																	
Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40
P1.0	HWIN0 (DX0)	34	26	22	P0.14	DX1A	57	45	39	P0.0	DX2A	41	29	23	P1.0	DOUT0	34	26	22
P1.1	HWIN1 (DX3)	33	25	21	P0.8	DX1B	51	39	33	P0.9	DX2B	52	40	34	P1.1	DOUT1	33	25	21
P1.2	HWIN2 (DX4)	32	24	20	P0.7	DX1C	48	36	30	P0.10	DX2C	53	41	35	P1.2	DOUT2	32	24	20
P1.3	HWIN3 (DX5)	31	23	19	P1.1	DX1D	33	25	21	P0.11	DX2D	54	42	36	P1.3	DOUT3	31	23	19
					P2.0	DX1E	9	3	1	P0.12	DX2E	55	43	37					
										P0.13	DX2F	56	44	38					

USIC1_CH0																	
Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40
-	HWIN0 (DX0)	40	-	-	P0.2	DX1A	43	31	25	P4.7	DX2A	4	2	-	-	DOUT0	40	-	-	-
-	HWIN1 (DX3)	39	-	-	P4.3	DX1B	62	-	-	P4.8	DX2B	5	-	-	-	DOUT1	39	-	-	-
-	HWIN2 (DX4)	38	-	-	P4.5	DX1C	64	48	-	P4.9	DX2C	6	-	-	-	DOUT2	38	-	-	-
-	HWIN3 (DX5)	37	-	-	P4.6	DX1D	3	1	-	-	DX2D	7	-	-	-	DOUT3	37	-	-	-
					-	DX1E	40	-	-	-	DX2E	8	-	-					
										-	DX2F	37	-	-					

Figure 103 Available Pins for Multi-IO SSC Slave Mode Communication

16.4.5.1 Operating the SSC in Multi-IO Modes

In order to operate the multi-IO SSC in either master or slave mode, the following steps have to be taken in addition to the USIC channel initialization steps outlined in Operating the SSC in Master Mode and Operating the SSC in Slave Mode:

- **Enable hardware port control**
  - Enable the dedicated hardware port interface to the DX0/DOUT0 and DX3/DOUT1 pins (dual-SSC) and additionally, DX4/DOUT2 and DX5/DOUT3 pins (Quad-SSC) by configuring the bit field CCR.HPCEN.
- **Enable dynamic update of transfer parameters with TCI**
  - Enable the dynamic control of both the shift mode and pin direction during data transfers, i.e. SCTR.DSM and SCTR.HPCDIR bit fields are updated with TCI.
  - This is done by setting the bit TCSR.HPCMD to 1.

16.4.5.2 Quad-SSC Example

Figure 104 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.

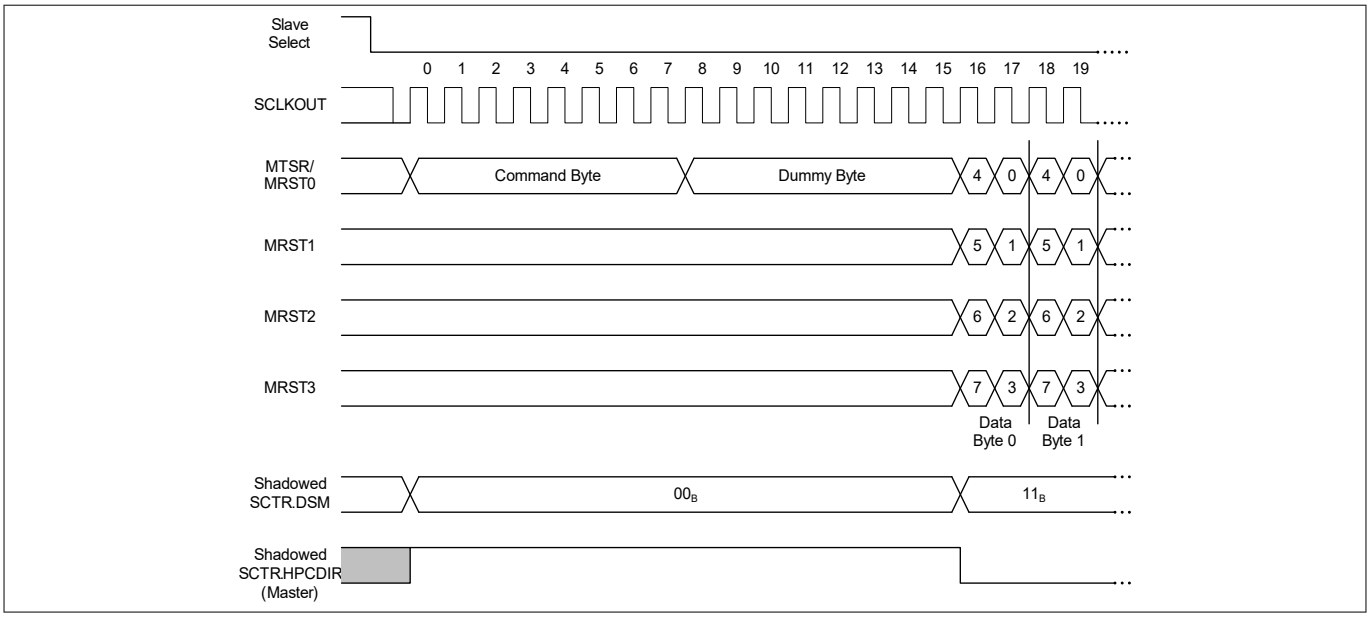


Figure 104 Quad-SSC Example

Initialization

The following code modifications can be made to the initialization code examples in Initialization Code Example or Initialization Code Example to configure the USIC channel as a Quad-SSC master or slave:

```
/// ----------------------------------------------------------
/// 5.Configure data transfer parameters
/// - Single shot transmission of data word when a valid word
/// is available
/// - Hardware port control TCI mode is enabled
/// ----------------------------------------------------------
// TDEN TDSSM HPCMD
USIC0_CH1->TCSR =(1 << 10)|(1 << 8)|(1 << 6);
/// ----------------------------------------------------------
/// 7.Enable SSC protocol and hardware port control
/// ----------------------------------------------------------
// HPCEN MODE
USIC0_CH1->CCR =(3 << 6)|(1 << 0);
```

Data Flow

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.

16.4.6 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.

16.4.6.1 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.

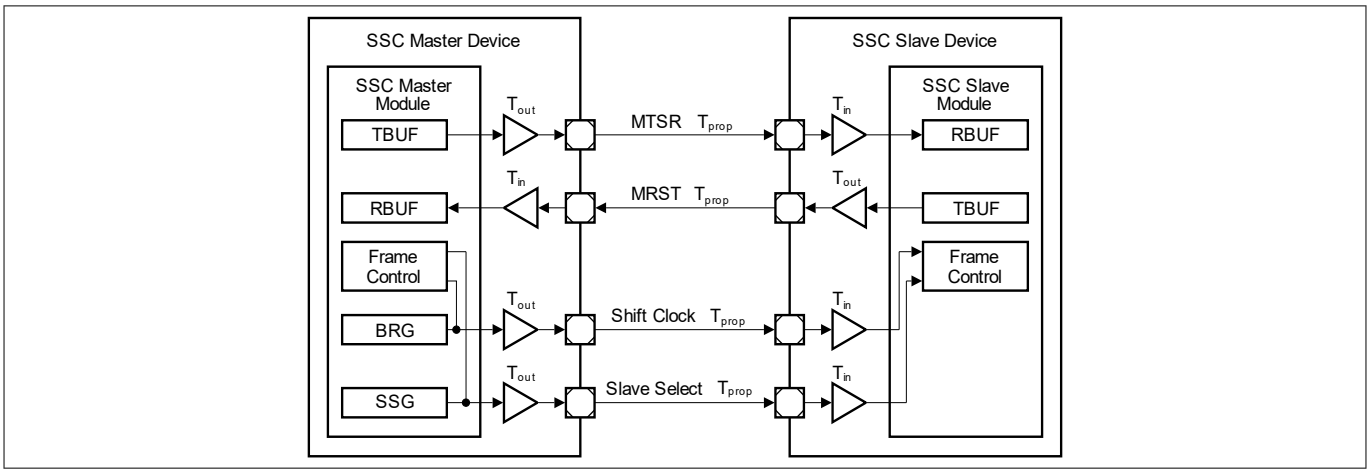


Figure 105 SSC Closed-loop Delay

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 $T_{out_master} + 2 \times T_{prop} + T_{in_slave} + T_{out_slave} + T_{in_master}$ + 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 Datasheet.

Figure 106 describes graphically the closed-loop delay and the effect of two delay compensation options discussed in Delay Compensation in Master Mode and Complete Closed-loop Delay Compensation.

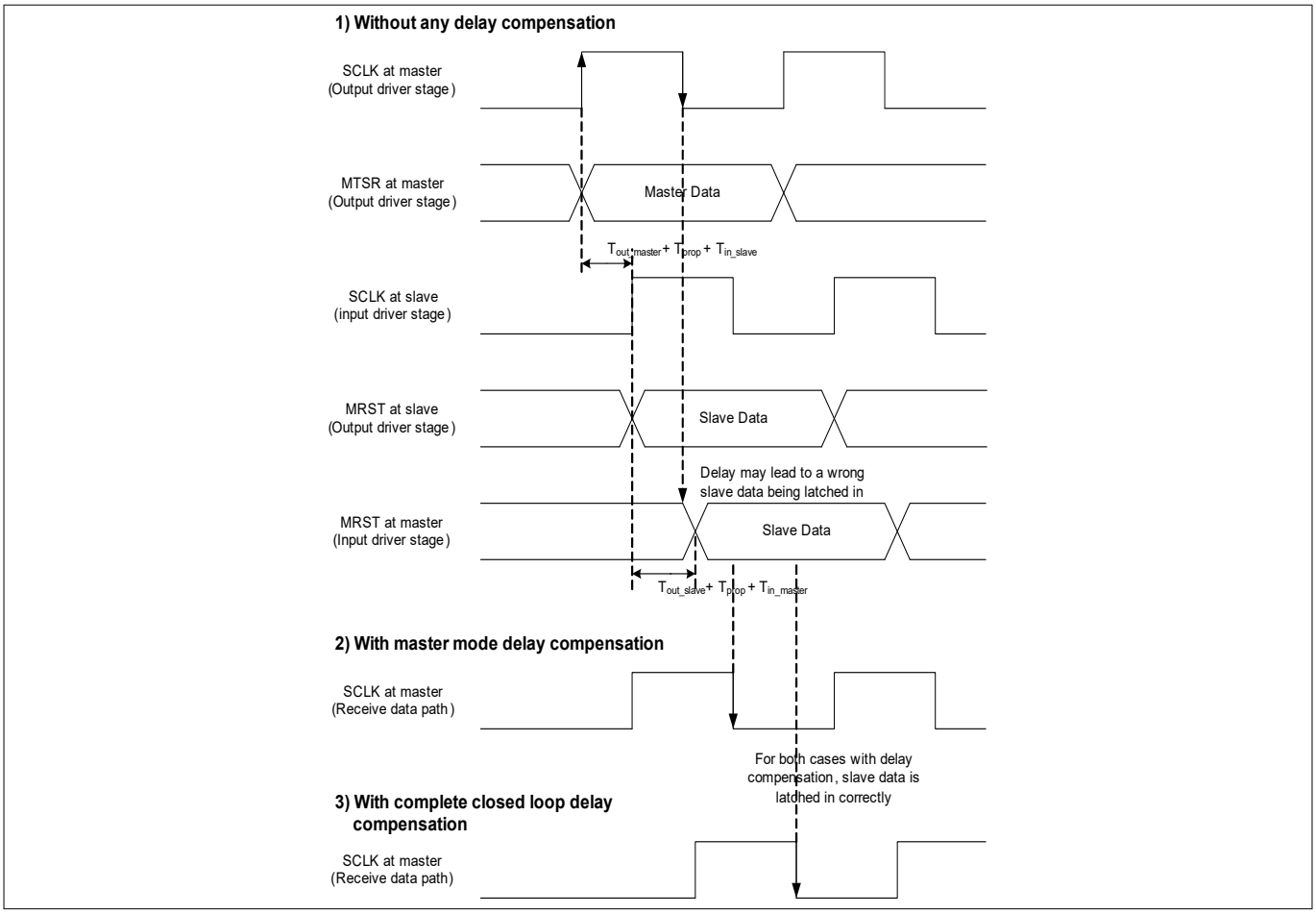


Figure 106 SSC Closed-loop Delay Timing Waveform

16.4.6.2 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.

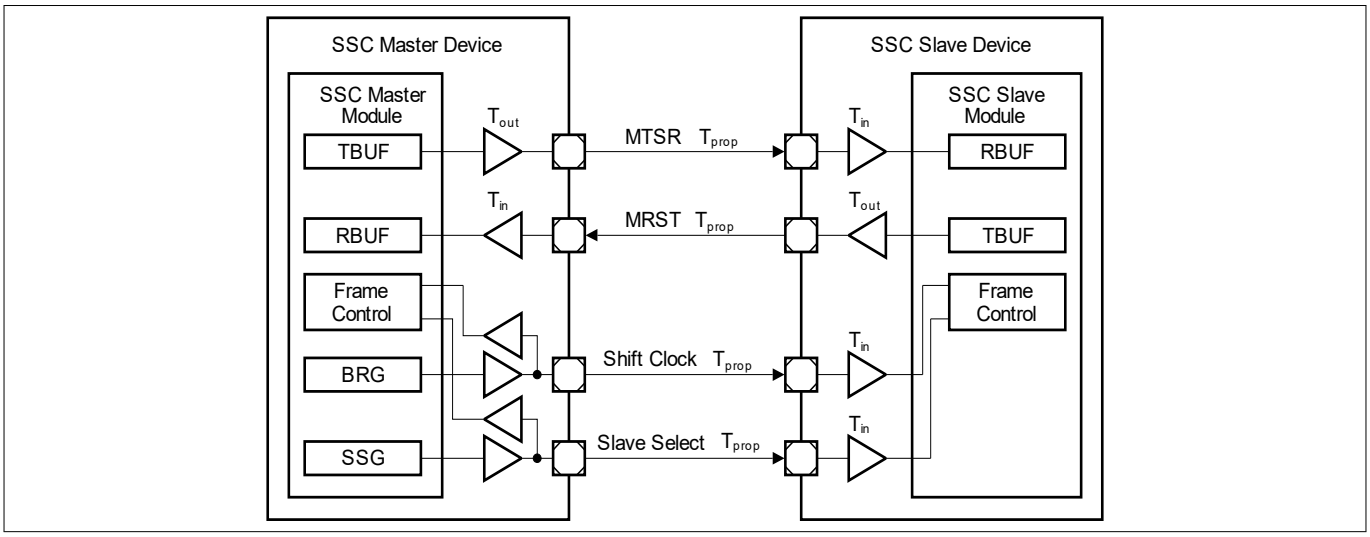


Figure 107 SSC Master Mode with Delay Compensation

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.

16.4.6.3 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.

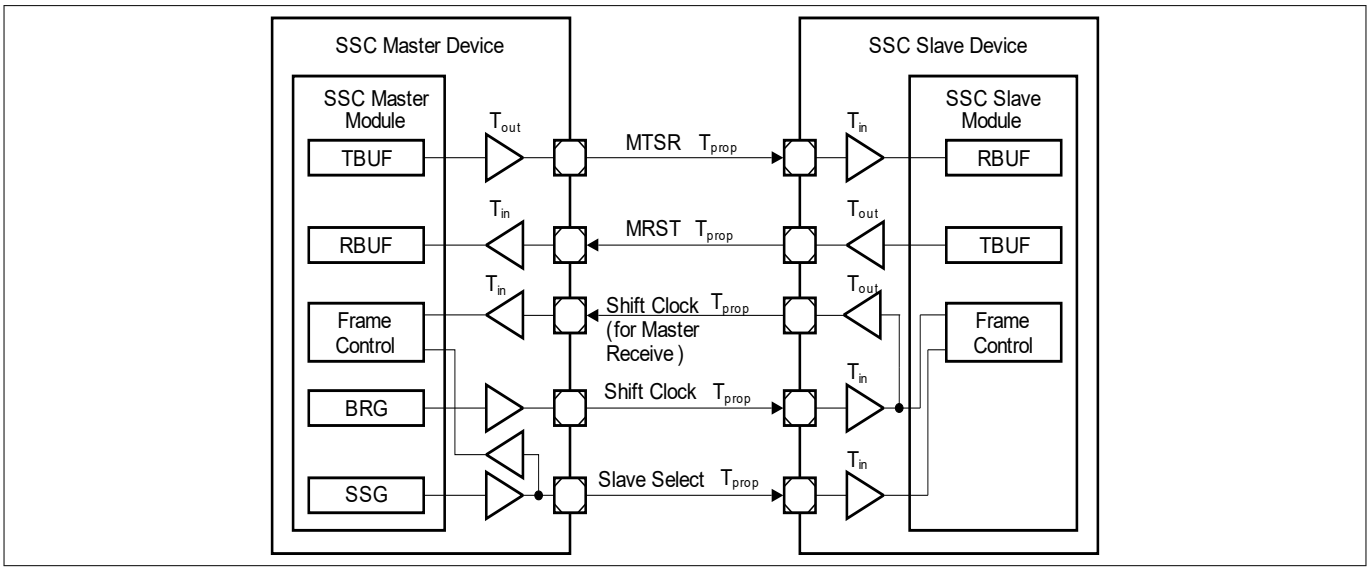


Figure 108 SSC Complete Closed-loop Delay Compensation

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.

16.5 Inter-IC Bus Protocol (IIC)

The IIC protocol of the USIC refers to the IIC bus specification [7].

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.

16.5.1 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)
- Powerful interrupt handling due to multitude of indication flags
- Compensation support for input delays

16.5.1.1 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 109 shows a connection of two IIC bus participants (modules IIC A and IIC B) using the USIC.

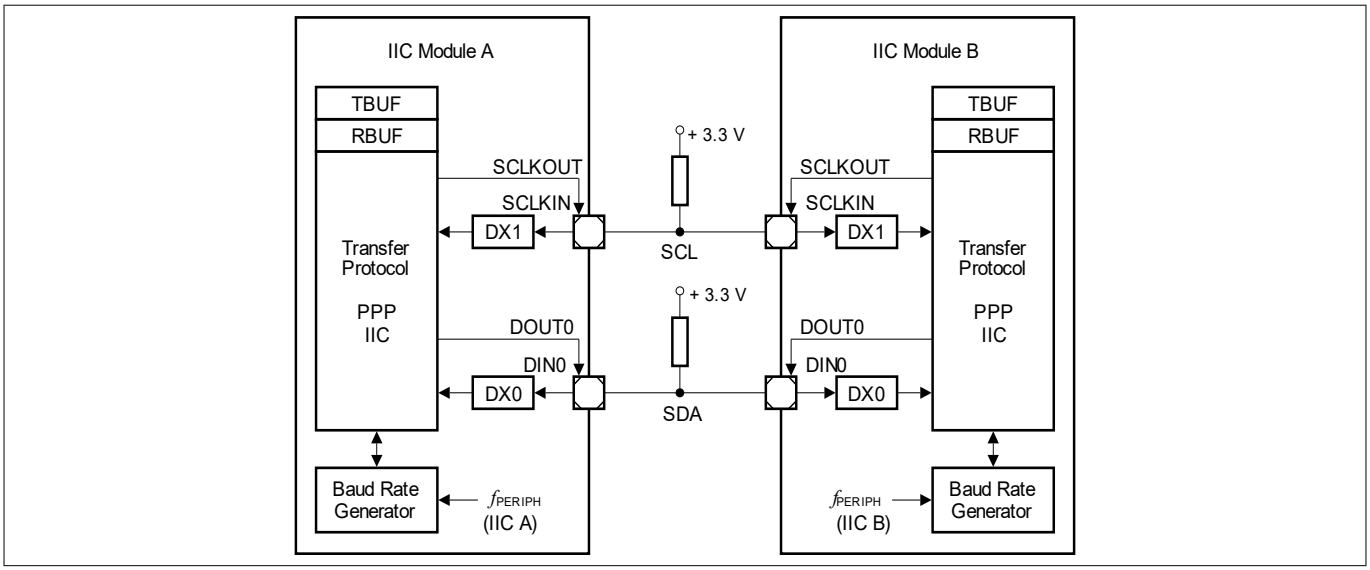


Figure 109 IIC Signal Connections

Figure 110 shows the pins that support the IIC functions.

USIC0_CH0

Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40
P0.14	DX0A	57	45	39	P0.14	DOUT0	57	45	39	P0.14	DX1A	57	45	39	P0.14	SCLKOUT	57	45	39
P0.15	DX0B	58	46	40	P0.15	DOUT0	58	46	40	P0.8	DX1B	51	39	33	P0.8	SCLKOUT	51	39	33
P1.0	DX0C	34	26	22	P1.0	DOUT0	34	26	22	P0.7	DX1C	48	36	30	P0.7	SCLKOUT	48	36	30
P1.1	DX0D	33	25	21	P1.1	DOUT0	33	25	21	P2.0	DX1E	9	3	1	P2.0	SCLKOUT	9	3	1
P2.0	DX0E	9	3	1	P2.0	DOUT0	9	3	1										
P2.1	DX0F	10	4	2	P2.1	DOUT0	10	4	2										

USIC0_CH1

Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40
P1.3	DX0A	31	23	19	P1.3	DOUT0	31	23	19	P1.3	DX1A	31	23	19	P1.3	SCLKOUT	31	23	19
P1.2	DX0B	32	24	20	P1.2	DOUT0	32	24	20	P0.8	DX1B	51	39	33	P0.8	SCLKOUT	51	39	33
P0.6	DX0C	47	35	29	P0.6	DOUT0	47	35	29	P2.11	DX1D	20	14	12	P2.11	SCLKOUT	20	14	12
P0.7	DX0D	48	36	30	P0.7	DOUT0	48	36	30										
P2.11	DX0E	20	14	12	P2.11	DOUT0	20	14	12										
P2.10	DX0F	19	13	11	P2.10	DOUT0	19	13	11										

USIC1_CH0

Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40
P0.2	DX0A	43	31	25	P0.2	DOUT0	43	31	25	P0.2	DX1A	43	31	25	P0.2	SCLKOUT	43	31	25
P0.3	DX0B	44	32	26	P0.3	DOUT0	44	32	26	P4.3	DX1B	62	-	-	P4.3	SCLKOUT	62	-	-
P4.4	DX0C	63	47	-	P4.4	DOUT0	63	47	-	P4.5	DX1C	64	48	-	P4.5	SCLKOUT	64	48	-
P4.5	DX0D	64	48	-	P4.5	DOUT0	64	48	-	P4.6	DX1D	3	1	-	P4.6	SCLKOUT	3	1	-
-	DX0E	39	-	-	-	DOUT0	39	-	-	-	DX1E	40	-	-	-	SCLKOUT	40	-	-
-	DX0F	40	-	-	-	DOUT0	40	-	-										

USIC1_CH1

Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40
P0.0	DX0A	41	29	23	P0.0	DOUT0	41	29	23	P0.1	DX1A	42	30	24	P0.1	SCLKOUT	42	30	24
P0.1	DX0B	42	30	24	P0.1	DOUT0	42	30	24	P2.12	DX1B	21	15	-	P2.12	SCLKOUT	21	15	-
P2.12	DX0C	21	15	-	P2.12	DOUT0	21	15	-	-	DX1C	26	-	-	-	SCLKOUT	26	-	-
P2.13	DX0D	22	16	-	P2.13	DOUT0	22	16	-	P3.0	DX1D	36	28	-	P3.0	SCLKOUT	36	28	-
P3.0	DX0E	36	28	-	P3.0	DOUT0	36	28	-										
-	DX0F	37	-	-	-	DOUT0	37	-	-										

Figure 110 Available Pins for IIC Communication

16.5.1.2 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.

Table 90 IIC Symbol Definition

Symbol	Definition
Bus idle	SDA and SCL are high. No data transfer takes place currently.
Data bit	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	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	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	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.

16.5.1.3 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.

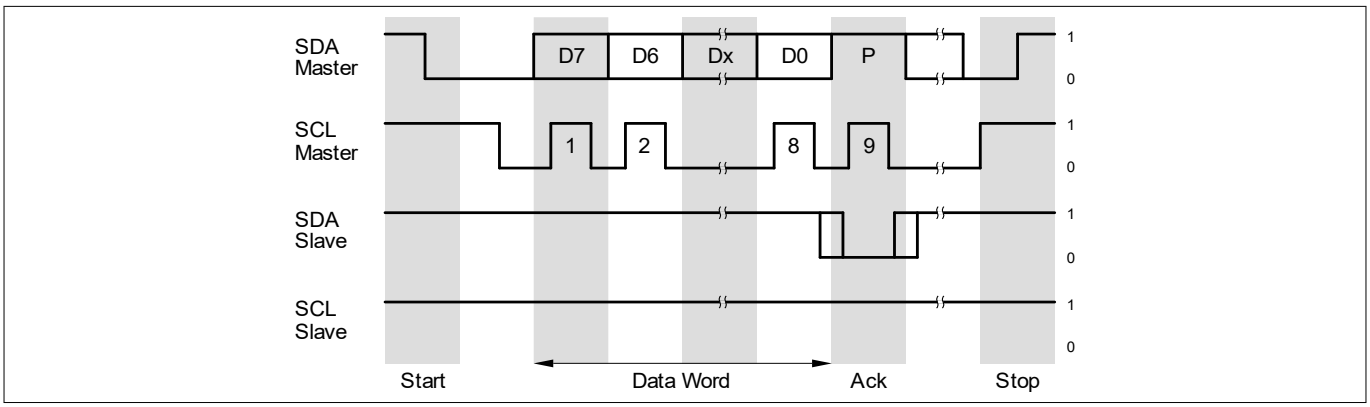


Figure 111 IIC Frame Example (simplified)

16.5.2 Symbol Timing

The symbol timing of the IIC is determined by the master generating 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.

  Additionally, if the digital filter stage should be used to eliminate spikes up to 50 ns, a filter frequency of 20 MHz is necessary.

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:

$$\begin{aligned} \text{HDEL} &\geq 300\ \text{ns} \times f_{PPP} - \left(3 \times \frac{f_{PPP}}{f_{PERIPH}}\right) + 1 \quad \text{with digital filter and HDELmin} = 2 \\ \text{HDEL} &\geq 300\ \text{ns} \times f_{PPP} - \left(3 \times \frac{f_{PPP}}{f_{PERIPH}}\right) + 2 \quad \text{without digital filter and HDELmin} = 1 \end{aligned} \tag{26}$$

For BRG.CLKSEL = 00_B and BRG.PPPEN = 0, $f_{PPP} = f_{FD}$, which is the output frequency of the fractional divider (see Fractional Divider).

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.

16.5.2.1 Start Symbol

Figure 112 shows the general start symbol timing.

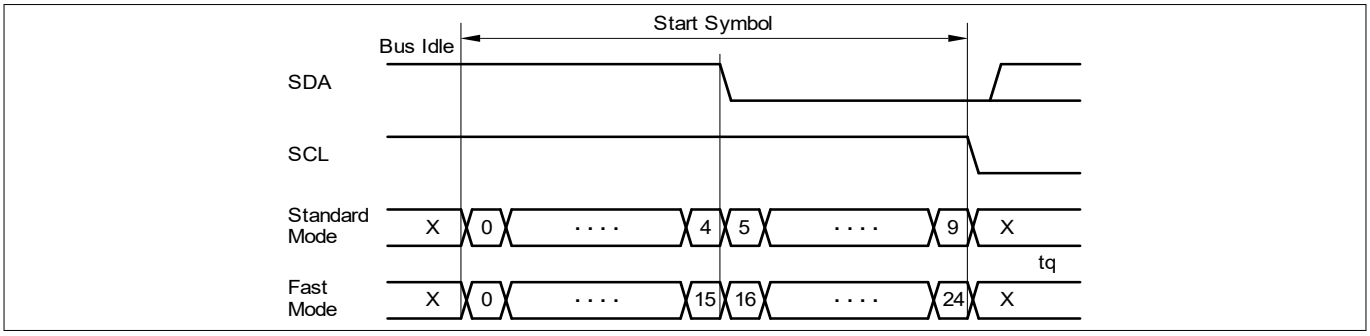


Figure 112 Start Symbol Timing

16.5.2.2 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 113.

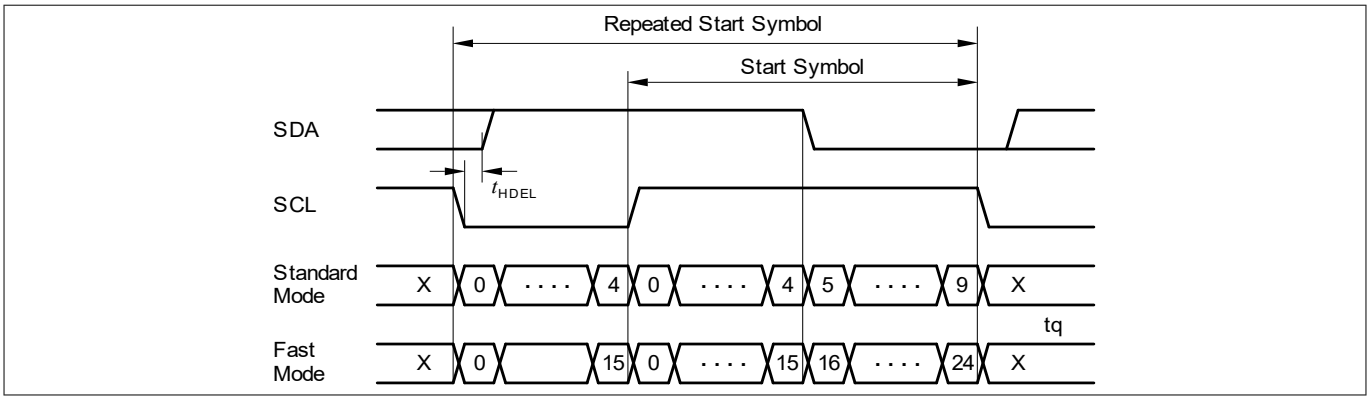


Figure 113 Repeated Start Symbol Timing

16.5.2.3 Stop Symbol

Figure 114 shows the stop symbol timing.

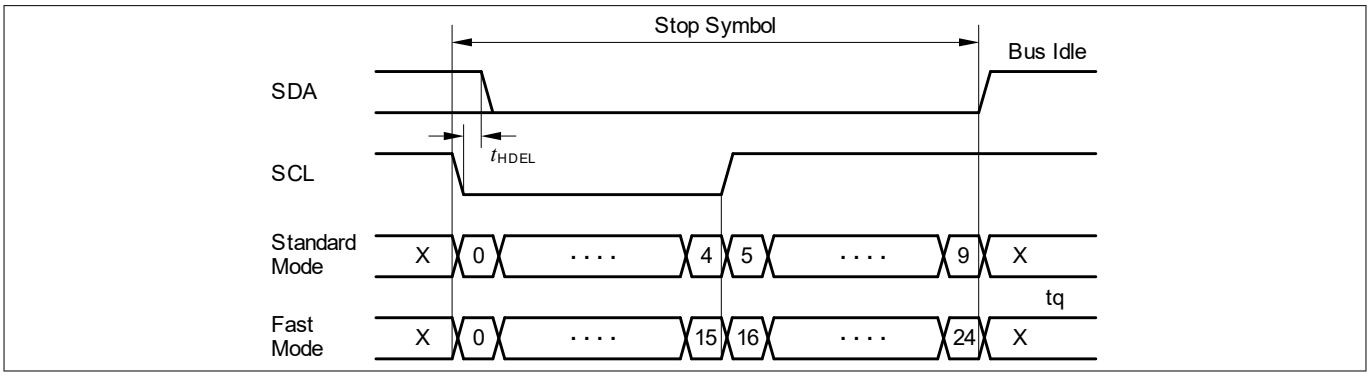


Figure 114 Stop Symbol Timing

16.5.2.4 Data Bit Symbol

Figure 115 shows the general data bit symbol timing.

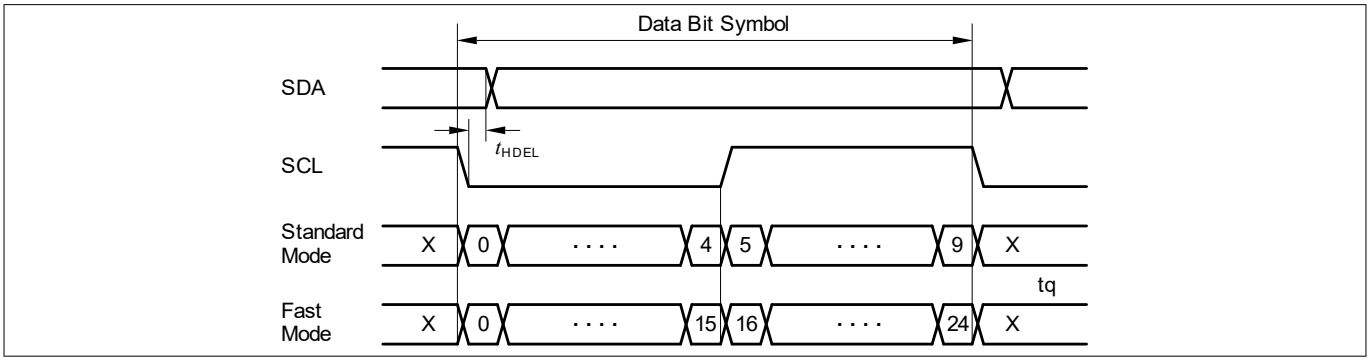


Figure 115 Data Bit Symbol

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.

16.5.3 Operating the IIC

In order to operate the IIC protocol, the USIC channel has to be first initialized.

It is recommended to configure all parameters of the IIC that do not change during run time while CCR.MODE = 0000_B, except stated otherwise.

The main initialization steps are outlined below:

- **Enable USIC channel**
  - Enable the module by writing 1s to MODEN and BPMODEN bits in KSCFG register.
- **Configure baud rate generator**
  - Select either fractional divider mode or normal divider mode with FDR.DM bit.
  - Configure the bit timing and baud rate setting through register bit fields BRG.PCTQ, BRG.DCTQ, BRG.PDIV and FDR.STEP, while BRG.SCLKCFG = 00_B.
  - 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.
- **Configure input pins**
  - Establish a connection of input stage DX0 to the shift data pin SDA (signal DIN0) selected by DX0CR.DSEL bit field, and with both bit DX0CR.INSW and bit DX0CR.DPOL = 0.
  - Similarly, establish a connection of input stage DX1 to the shift clock line SCL (signal SCLKIN) selected by DX1CR.DSEL bit field, and with both bit DX1CR.INSW and DX1CR.DPOL = 0.
  - 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.
- **Configure data format**
  - Configure the data format for 8 data bits (SCTR.WLE = 7), unlimited data flow (SCTR.FLE = $3F_H$) and MSB shifted first (SCTR.SDIR = 1).
  - Select the passive output level to be high (SCTR.PDL = 1) and for the data output to be without inversion (SCTR.DOCFG = 00_B).
  - Write SCTR.TRM = 11_B to enable IIC data transfers.
- **Configure data transfer parameters**
  - Write TCSR.TDSSM = 1 and TCSR.TDEN = 01_B to enable data transmission in single shot mode.
- **Configure protocol control parameters**
  - Select 25 time quanta per symbol for IIC fast mode (PCR.STIM = 1).
  - Configure the hardware delay (PCR.HDEL = 7) to ensure the SDA hold time after a falling SCL edge.
- **Select IIC protocol**
  - Enable IIC mode with CCR.MODE = 0100_B.
- **Configure output pins**
  - Select the transmit data SDA output pin (signal DOUT0) to be the same pin used for the SDA input.
  - Similarly, configure the shift clock SCL output pin (signal SCLKOUT) to be the same pin used for the SCL input.
  - 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.
  - Selection of the output pins and their behaviour are done through the respective pins' port control register Pn_IOCR0/4/8/12. Refer to the port chapter.
  - The step to enable the output pin functions should only be done after the IIC mode is enabled with CCR.MODE, to avoided unintended spikes on the output.

An initialization code example is given in IIC Initialization Code Example.

16.5.3.1 Baud Rate Generation

The baud rate f_{IIC} in IIC mode depends on the number of time quanta per bit time and their timing. 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 (DCTQ = 9_H for standard mode or DCTQ = 18_H for fast mode

The baud rate is given by:

$$f_{IIC} = f_{CTQIN} \times \frac{1}{PCTQ + 1} \times \frac{1}{DCTQ + 1} \quad (27)$$

The standard setting is given by CTQSEL = 00_B ($f_{CTQIN} = f_{PDIV}$), PPPEN = 0 ($f_{PPP} = f_{PIN}$) and CLKSEL = 00_B ($f_{PIN} = f_{FD}$). Under these conditions, the baud rate can further be equated to either (28) or (29), depending on the selected divider mode:

- In normal divider mode (FDR.DM = 01_B)

$$f_{IIC} = f_{PERIPH} \times \frac{1}{1024 - STEP} \times \frac{1}{PDIV + 1} \times \frac{1}{PCTQ + 1} \times \frac{1}{DCTQ + 1} \quad (28)$$

- In fractional divider mode (FDR.DM = 10_B)

$$f_{IIC} = f_{PERIPH} \times \frac{STEP}{1024} \times \frac{1}{PDIV + 1} \times \frac{1}{PCTQ + 1} \times \frac{1}{DCTQ + 1} \quad (29)$$

Table 91 shows examples of the baud rate calculation in fractional divider mode (FDR.DM = 10_B).

Table 91 IIC Baud Rate Calculation in Fractional Divider Mode

f_{PERIPH} (MHz)	FDR.STEP	BRG.PDIV	BRG.PCTQ	BRG.DCTQ	f_{IIC} (kbit/s)	Deviation Error
48	512	11	1	9	100	0.00%
48	427	0	1	24	400	0.08%

***Note**: The values shown in Table 91 should be used only as a reference. The actual IIC baud rates are highly influenced by external factors such as the pull-up resistance on the bus, the slave speed, etc.*

The baud rate setting should only be changed while the transmitter and the receiver are idle or CCR.MODE = 0.

16.5.3.2 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 109 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}.

16.5.3.3 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.

16.5.3.4 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.

16.5.3.5 Not Acknowledge and Error Conditions

In case of a not 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.

A software recovery sequence can include the following steps:

1. Flush the FIFO buffer (if FIFO buffer is used):
   a. Write 1_B to both TRBSCR.FLUSHTB and TRBSCR.FLUSHRB bits.
2. Invalidate the internal transmit buffer TBUF:
   a. Write 10_B to FMR.MTDV bit field.
3. Clear all status bits if necessary.

For the IIC slave, it might be additionally required to first disable the SDA and SCL output port functions (write 0 to the corresponding IOCRx.PCy bit fields). This ensures that the SDA and SCL lines are not held low infinitely by the slave. The port functions can be enabled again at the end of the sequence.

16.5.3.6 Mode Control Behavior

In IIC mode, the following kernel modes are supported:

- 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.

***Note**: In multi-master mode, only run mode 0 and stop mode 0 are supported, the other modes must not be programmed.*

16.5.3.7 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.

Table 92 IIC data transfer interrupt handling

Interrupt	Indicated by bit	Description
Transmit buffer interrupt	PSR.TBIF	Set after the content of the transmit buffer TBUF has been loaded to the transmit shift register. With this event, bit TCSR.TDV is cleared and new data can be loaded to the transmit buffer. This interrupt can be used to write the next TBUF entry while the last one is in progress (handled by the transmitter part).
Transmit shift interrupt	PSR.TSIF	Set after the start of the last data bit of a data word.
Receiver start interrupt	PSR.RSIF	Set after the sample point of the first data bit of a data word.

(table continues...)

Table 92 (continued) IIC data transfer interrupt handling

Interrupt	Indicated by bit	Description
Receiver interrupt	PSR.RIF	Set after a data byte, which is not the first byte of a frame, is received (after the falling edge of SCL) in the receive buffer RBUF0/1. 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).

Table 93 IIC data transfer interrupt handling

Interrupt	Indicated by bit	Description
Alternative interrupt	PSR.AIF	Similar to PSR.RIF except that PSR.AIF indicates that the data byte received is the first byte of a new frame. AIF is based on bit RBUFSR[9] (same as RBUF[9]).
Data lost interrupt	PSR.DLIF	Set 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.

***Note**: The transmit shift and receive start events can be ignored if the application does not require them during the IIC data transfer.*

16.5.3.8 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.

Table 94 IIC protocol interrupt events

Events	Event flag	Description	Flag clear	Interrupt enable
Start condition	PSR.SCR	Set after a valid start condition is detected.	PSCR.CST2	PCR.SCRIEN
Repeated start condition	PSR.RSCR	Set after a valid repeated start condition is detected.	PSCR.CST3	PCR.RSCRIEN
Stop condition	PSR.PCR	Set after a valid stop condition is detected.	PSCR.CST4	PCR.PCRIEN
Master arbitration lost	PSR.ARL	Set after the master arbitration is lost while in master mode.	PSCR.CST6	PCR.ARLIEN
Slave read requested	PSR.SRR	Set after a slave read is requested while in slave mode.	PSCR.CST7	PCR.SRRIEN
ACK received	PSR.ACK	Set after an acknowledge bit is received while in master mode.	PSCR.CST9	PCR.ACKIEN

(table continues...)

Table 94 (continued) IIC protocol interrupt events

Events	Event flag	Description	Flag clear	Interrupt enable
NACK received	PSR.NACK	Set after a not acknowledge bit is received while in master mode.	PSCR.CST5	PCR.NACKIEN
Error condition	PSR.ERR	Set after one of the following has occurred: • Start condition not at the expected position in a frame • Stop condition not at the expected position in a frame • 10-bit address interrupted by a stop condition after the first address byte in slave mode	PSCR.CST8	PCR.ERRIEN
TDF code error	PSR.WTDF	Set after one of the following has occurred: • TDF slave code in master mode • TDF master code in slave mode • Reserved TDF code found • Start condition code during a running frame in master mode • Data byte transmission code after transfer direction has been changed to reception (master read) in master mode	PSCR.CST1	

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.

16.5.3.9 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, the slave can also be configured to acknowledge the address byte 00_H, which indicates a general call address. This is done by setting the bit PCR.ACK00 to 1.

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 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.
- 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.

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.

***Note**: The address byte 01_H (START byte) and other reserved addresses (refer to IIC specification) are not acknowledged.*

16.5.3.10 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.DLIF 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.

16.5.3.11 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.

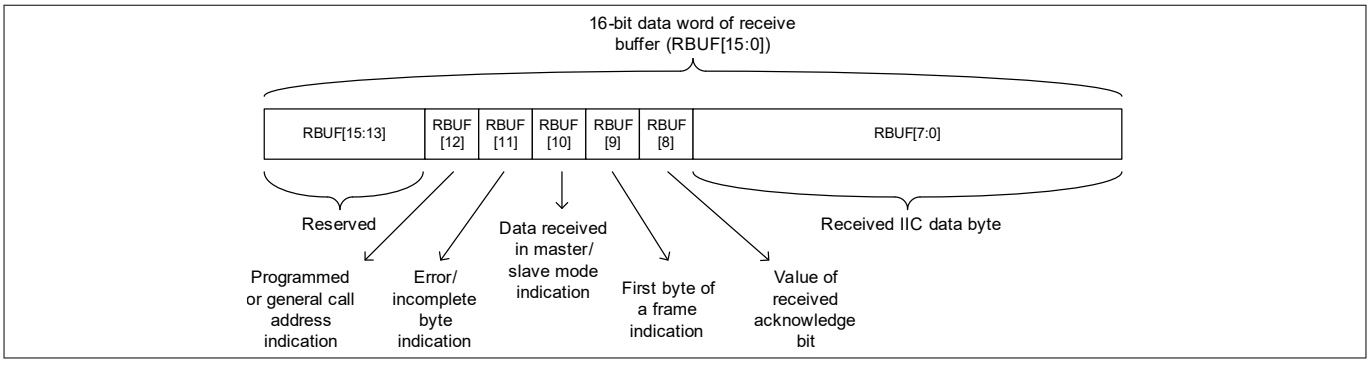


Figure 116 IIC received data word

Table 95 IIC protocol related information in RBUF register

Field	Description
RBUF[8]	Value of the received acknowledge bit. This information is also available in RBUFSR[8] as protocol argument.

(table continues...)

Table 95 (continued) IIC protocol related information in RBUF register

Field	Description
RBUF[9]	This bit indicates the data byte received after a (repeated) start condition and following the address reception: 0_B Subsequent data bytes of a frame has been received. 1_B The first data byte of a new frame has been received. This information is also available in RBUFSR[9], allowing different interrupt routines for the address and data handling.
RBUF[10]	0_B Data byte has been received while in slave mode. 1_B Data byte has been received while in master mode.
RBUF[11]	0_B Data byte is received correctly. 1_B 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.
RBUF[12]	0_B The programmed address has been received. 1_B A general call address has been received.

16.5.3.12 IIC Initialization Code Example

The following code example implements a function to initialize USIC0 channel 1 for IIC data transfers with a baud rate of 400 kbit/s (f_{PERIPH} = 48 MHz).

The same code can be used for master and slave mode operation. If the slave address is not needed (i.e. master only operation), the bit field PCR.SLAD can be programmed to 0.

```
void USIC0_CH1_IIC_Init(void)
{
/// ----------------------------------------------------------
/// 1.Enable USIC0 channel 1
/// ----------------------------------------------------------
// BPMODEN MODEN
USIC0_CH1->KSCFG|= (1 << 1)| (1 << 0);
/// ----------------------------------------------------------
/// 2.Configure baud rate generator
/// - Fractional divider mode
/// - Baud rate = 400 kbit/s
/// ----------------------------------------------------------
// DM STEP
USIC0_CH1->FDR = (2 << 14) | (427 << 0);
// PDIV DCTQ PCTQ
USIC0_CH1->BRG = (0 << 16)| (24 << 10)| (1 << 8);
/// ----------------------------------------------------------
/// 3.Configure input stages
/// - Select inputs DX0D and DX1B
/// - Protocol pre-processor to control data shift unit inputs
/// ----------------------------------------------------------
// INSW DSEL
USIC0_CH1->DX0CR =(0 << 4) |(3 << 0);
// INSW DSEL
USIC0_CH1->DX1CR =(0 << 4) |(1 << 0);
/// ----------------------------------------------------------
/// 4.Configure data format
/// - Data word = 8 bits with unlimited data flow
/// - Data transfer allowed with passive level = 1 and MSB first
/// ----------------------------------------------------------,
// WLE FLE TRM PDL SDIR
USIC0_CH1->SCTR =(7<<24)|(63<<16)|(3<<8)|(1<<1)|(1<<0);
/// ----------------------------------------------------------
/// 5.Configure data transfer parameters
/// - Single shot transmission of data word when a valid word
/// is available
/// ----------------------------------------------------------
// TDEN TDSSM
USIC0_CH1->TCSR =(1 << 10) |(1 << 8);
/// ----------------------------------------------------------
/// 6.Configure IIC protocol-specific parameters
/// - 1 symbol = 25 time quanta
/// - Configure HDEL for SDA hold time
/// - Configure slave address (assume 7-bit address = 0x5)
/// ----------------------------------------------------------
// HDEL STIM SLAD
USIC0_CH1->PCR = (7<<26)|(1<<17)|(5<<9);;
/// ----------------------------------------------------------
/// 7.Enable IIC protocol
/// ----------------------------------------------------------
```

```
// MODE
USIC0_CH1->CCR =(4 << 0);
/// ----------------------------------------------------------
/// 8.Configure IIC output function pins
/// - Assume
/// - P0.7 ALT7 function is assigned to DOUT0
/// - P0.8 ALT7 function is assigned to SCLKOUT
/// ----------------------------------------------------------
// PC7
PORT0->IOCR4 |= (0x27 << 26);
// PC8
PORT0->IOCR8 |= (0x27 << 2);
}
```

16.5.4 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 (for example, 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.

***Note**: During an arbitration phase in multi-master bus systems an unpredictable bus behavior may occur due to an unexpected stop condition.*

16.5.4.1 Transmit Data Formats

The following transmit data formats are available in master mode:

Table 96 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 continues...)

Table 96 (continued) Master Transmit Data Formats

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 97 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.

16.5.4.2 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 98 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.

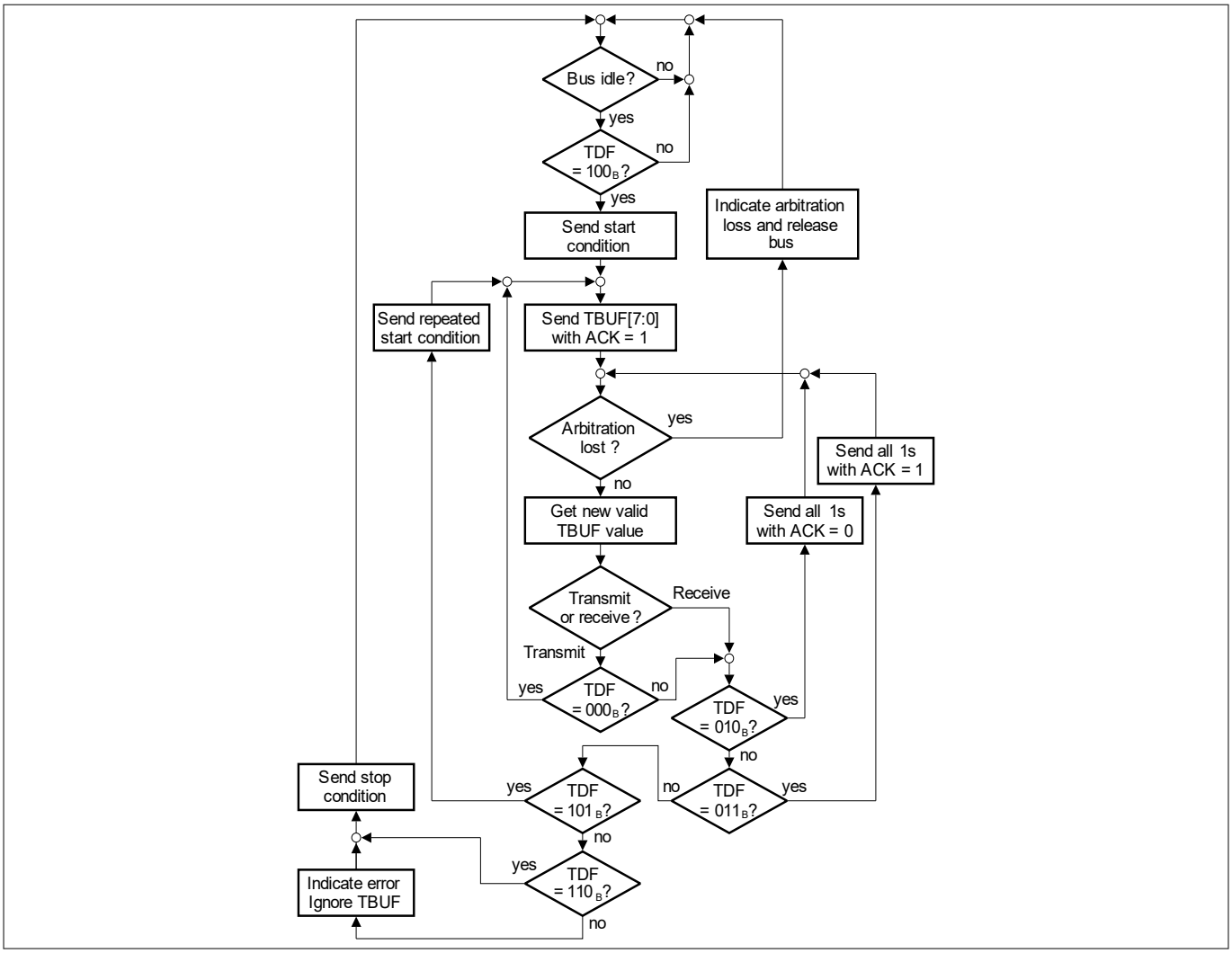


Figure 117 IIC Master Transmission

16.5.4.3 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 99.

Table 99 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

(table continues...)

Table 99 (continued) TDF Code Sequence for Master Transmit

TDF Code Sequence	TBUF[10:8] (TDF Code)	TBUF[7:0]	IIC Response	Interrupt Events
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 100 and Table 101.

Table 100 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
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 101 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

(table continues...)

Table 101 (continued) 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
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
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 118 shows the interrupt events during the master transmit-slave receive and master receive/slave transmit sequences.

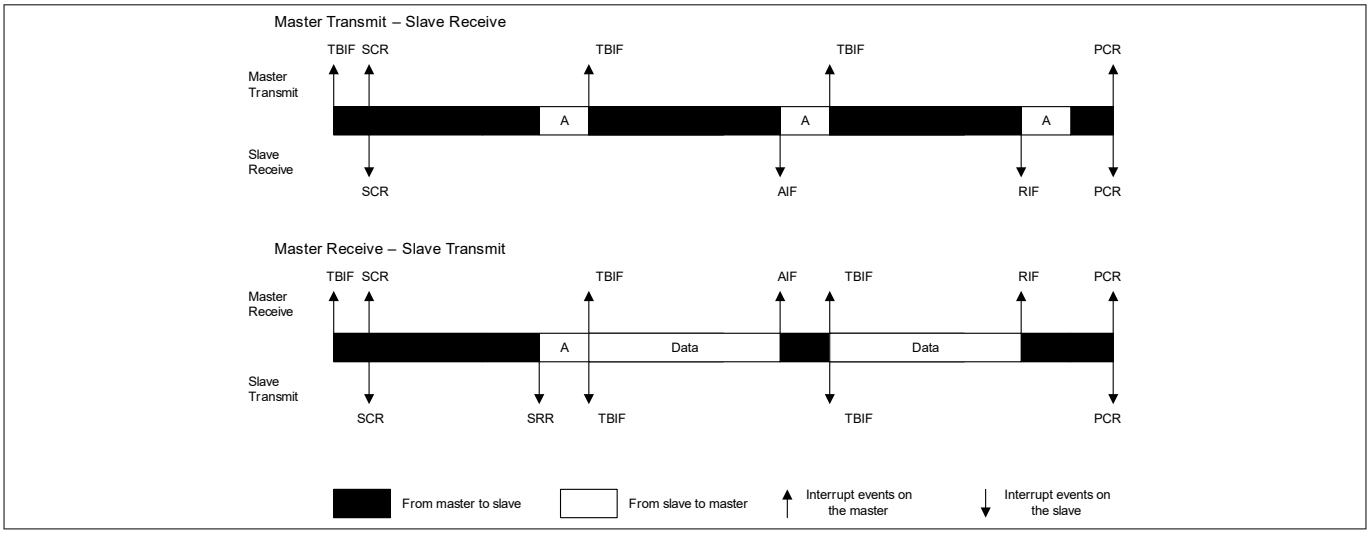


Figure 118 Interrupt Events on Data Transfers

16.5.4.4 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 118.

16.6 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.

16.6.1 Introduction

The IIS protocol is a synchronous serial communication protocol mainly for audio and infotainment applications [8].

16.6.1.1 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 (for example, 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 102 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

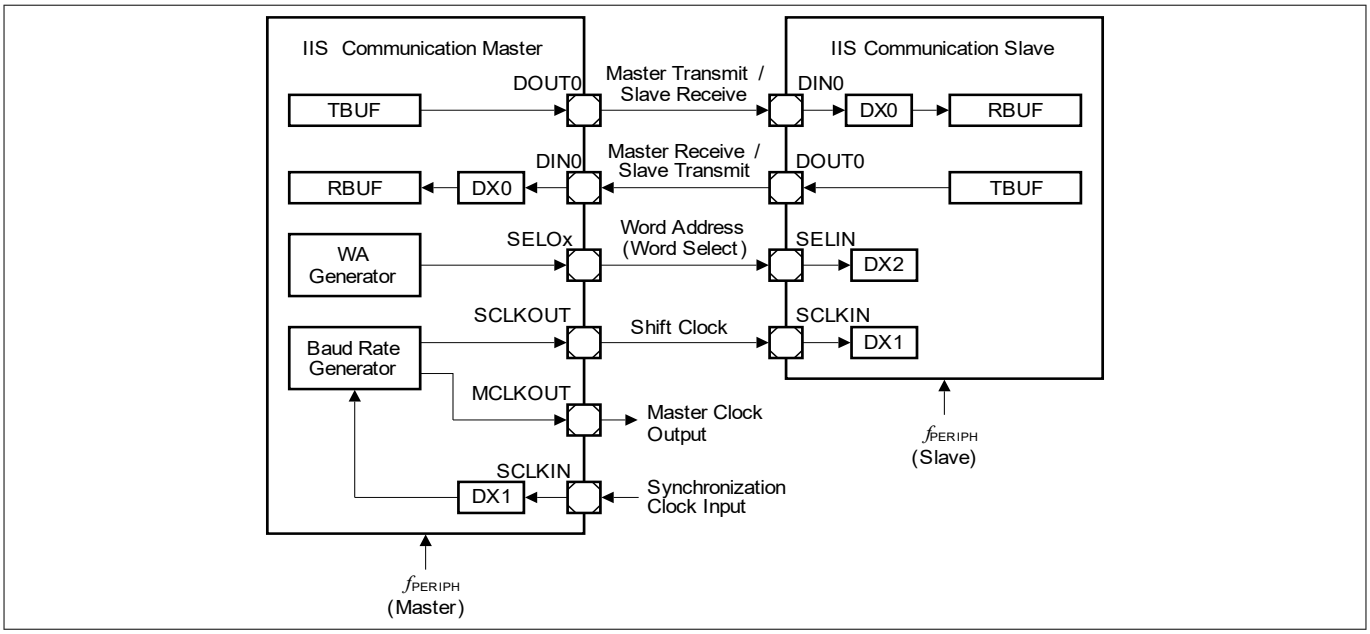


Figure 119 IIS Signals

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.

Figure 120 and Figure 121 show the pins that support IIS master and slave mode communications.

USIC0_CH0

Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40
P0.14	DX0A	57	45	39	P0.14	DOUT0	57	45	39	P0.7	SCLKOUT	48	36	30	P0.0	SELO0	41	29	23
P0.15	DX0B	58	46	40	P0.15	DOUT0	58	46	40	P0.8	SCLKOUT	51	39	33	P0.9	SELO0	52	40	34
P1.0	DX0C	34	26	22	P1.0	DOUT0	34	26	22	P0.14	SCLKOUT	57	45	39	P1.4	SELO0	30	22	18
P1.1	DX0D	33	25	21	P1.1	DOUT0	33	25	21	P1.6	SCLKOUT	28	20	16	P0.10	SELO1	53	41	35
P2.0	DX0E	9	3	1	P2.0	DOUT0	9	3	1	P2.0	SCLKOUT	9	3	1	P1.5	SELO1	29	21	17
P2.1	DX0F	10	4	2	P2.1	DOUT0	10	4	2						P0.11	SELO2	54	42	36
					P1.5	DOUT0	29	21	17						P1.6	SELO2	28	20	16
															P0.12	SELO3	55	43	37
															P0.13	SELO4	56	44	38

USIC0_CH1

Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40
P1.3	DX0A	31	23	19	P1.3	DOUT0	31	23	19	P0.8	SCLKOUT	51	39	33	P0.0	SELO0	41	29	23
P1.2	DX0B	32	24	20	P1.2	DOUT0	32	24	20	P1.3	SCLKOUT	31	23	19	P0.9	SELO0	52	40	34
P0.6	DX0C	47	35	29	P0.6	DOUT0	47	35	29	P1.4	SCLKOUT	30	22	18	P1.1	SELO0	33	25	21
P0.7	DX0D	48	36	30	P0.7	DOUT0	48	36	30	P2.1	SCLKOUT	10	4	2	P0.10	SELO1	53	41	35
P2.11	DX0E	20	14	12	P2.11	DOUT0	20	14	12	P2.11	SCLKOUT	20	14	12	P1.4	SELO1	30	22	18
P2.10	DX0F	19	13	11	P2.10	DOUT0	19	13	11						P0.11	SELO2	54	42	36
					P1.6	DOUT0	28	20	16						P1.5	SELO2	29	21	17
															P1.6	SELO3	28	20	16

USIC1_CH0

Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40
P0.2	DX0A	43	31	25	P0.2	DOUT0	43	31	25	P0.2	SCLKOUT	43	31	25	-	SELO0	37	-	-
P0.3	DX0B	44	32	26	P0.3	DOUT0	44	32	26	P2.12	SCLKOUT	21	15	-	P4.7	SELO0	4	2	-
P4.4	DX0C	63	47	-	P4.4	DOUT0	63	47	-	-	SCLKOUT	38	-	-	P3.0	SELO1	36	28	-
P4.5	DX0D	64	48	-	P4.5	DOUT0	64	48	-	-	SCLKOUT	40	-	-	P4.8	SELO1	5	-	-
-	DX0E	39	-	-	-	DOUT0	39	-	-	P4.3	SCLKOUT	62	-	-	P4.9	SELO2	6	-	-
-	DX0F	40	-	-	-	DOUT0	40	-	-	P4.5	SCLKOUT	64	48	-	-	SELO3	7	-	-
										P4.6	SCLKOUT	3	1	-	-	SELO4	8	-	-

USIC1_CH1

Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40
P0.0	DX0A	41	29	23	P0.0	DOUT0	41	29	23	P0.1	SCLKOUT	42	30	24	P0.4	SELO0	45	33	27
P0.1	DX0B	42	30	24	P0.1	DOUT0	42	30	24	P0.3	SCLKOUT	44	32	26	-	SELO0	39	-	-
P2.12	DX0C	21	15	-	P2.12	DOUT0	21	15	-	-	SCLKOUT	26	-	-	-	SELO1	40	-	-
P2.13	DX0D	22	16	-	P2.13	DOUT0	22	16	-	P2.12	SCLKOUT	21	15	-	-	SELO1	59	-	-
P3.0	DX0E	36	28	-	P3.0	DOUT0	36	28	-	P3.0	SCLKOUT	36	28	-	P4.1	SELO2	60	-	-
-	DX0F	37	-	-	-	DOUT0	37	-	-	-	SCLKOUT	38	-	-	P4.2	SELO3	61	-	-

Figure 120 Available Pins for IIS Master Mode Communication

USIC0_CH0

Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40
P0.14	DX0A	57	45	39	P0.14	DX1A	57	45	39	P0.0	DX2A	41	29	23	P0.14	DOUT0	57	45	39
P0.15	DX0B	58	46	40	P0.8	DX1B	51	39	33	P0.9	DX2B	52	40	34	P0.15	DOUT0	58	46	40
P1.0	DX0C	34	26	22	P0.7	DX1C	48	36	30	P0.10	DX2C	53	41	35	P1.0	DOUT0	34	26	22
P1.1	DX0D	33	25	21	P1.1	DX1D	33	25	21	P0.11	DX2D	54	42	36	P1.1	DOUT0	33	25	21
P2.0	DX0E	9	3	1	P2.0	DX1E	9	3	1	P0.12	DX2E	55	43	37	P2.0	DOUT0	9	3	1
P2.1	DX0F	10	4	2						P0.13	DX2F	56	44	38	P2.1	DOUT0	10	4	2
															P1.5	DOUT0	29	21	17

USIC0_CH1

Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40
P1.3	DX0A	31	23	19	P1.3	DX1A	31	23	19	P0.0	DX2A	41	29	23	P1.3	DOUT0	31	23	19
P1.2	DX0B	32	24	20	P0.8	DX1B	51	39	33	P0.9	DX2B	52	40	34	P1.2	DOUT0	32	24	20
P0.6	DX0C	47	35	29	P0.7	DX1C	48	36	30	P0.10	DX2C	53	41	35	P0.6	DOUT0	47	35	29
P0.7	DX0D	48	36	30	P2.11	DX1D	20	14	12	P0.11	DX2D	54	42	36	P0.7	DOUT0	48	36	30
P2.11	DX0E	20	14	12						P1.1	DX2E	33	25	21	P2.11	DOUT0	20	14	12
P2.10	DX0F	19	13	11						P2.0	DX2F	9	3	1	P2.10	DOUT0	19	13	11
															P1.6	DOUT0	28	20	16

USIC1_CH0

Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40
P0.2	DX0A	43	31	25	P0.2	DX1A	43	31	25	P4.7	DX2A	4	2	-	P0.2	DOUT0	43	31	25
P0.3	DX0B	44	32	26	P4.3	DX1B	62	-	-	P4.8	DX2B	5	-	-	P0.3	DOUT0	44	32	26
P4.4	DX0C	63	47	-	P4.5	DX1C	64	48	-	P4.9	DX2C	6	-	-	P4.4	DOUT0	63	47	-
P4.5	DX0D	64	48	-	P4.6	DX1D	3	1	-	-	DX2D	7	-	-	P4.5	DOUT0	64	48	-
-	DX0E	39	-	-	-	DX1E	40	-	-	-	DX2E	8	-	-	-	DOUT0	39	-	-
-	DX0F	40	-	-						-	DX2F	37	-	-	-	DOUT0	40	-	-

USIC1_CH1

Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40	Pin	Function	VQFN 64	VQFN 48	VQFN 40
P0.0	DX0A	41	29	23	P0.1	DX1A	42	30	24	-	DX2A	39	-	-	P0.0	DOUT0	41	29	23
P0.1	DX0B	42	30	24	P2.12	DX1B	21	15	-	-	DX2B	40	-	-	P0.1	DOUT0	42	30	24
P2.12	DX0C	21	15	-	-	DX1C	26	-	-	-	DX2C	27	-	-	P2.12	DOUT0	21	15	-
P2.13	DX0D	22	16	-	P3.0	DX1D	36	28	-						P2.13	DOUT0	22	16	-
P3.0	DX0E	36	28	-											P3.0	DOUT0	36	28	-
-	DX0F	37	-	-											-	DOUT0	37	-	-

Figure 121 Available Pins for IIS Slave Mode Communication

16.6.1.2 Protocol Overview

An IIS connection supports transfers for two different data frames via the same data line, for example, 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.

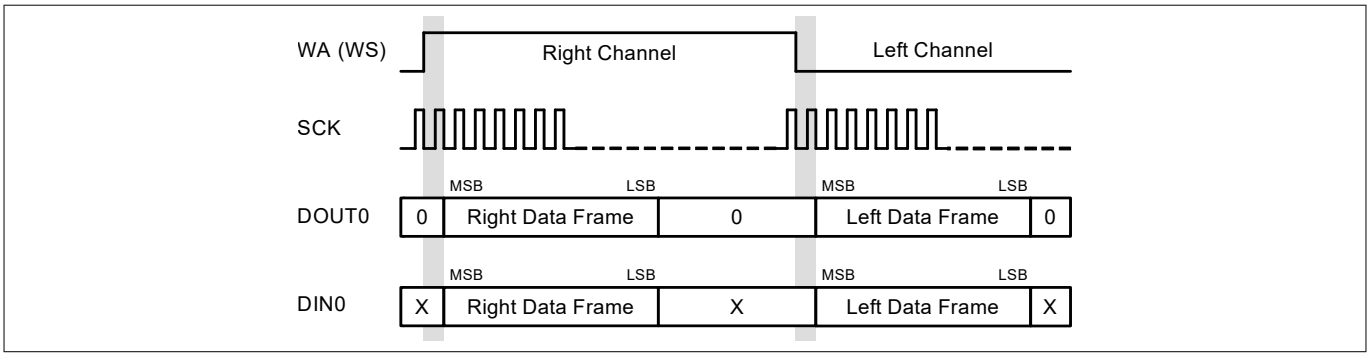


Figure 122 Protocol Overview

16.6.1.3 Transfer Delay

The transfer delay feature allows the transfer of data (transmission and reception) with a programmable delay (counted in shift clock periods).

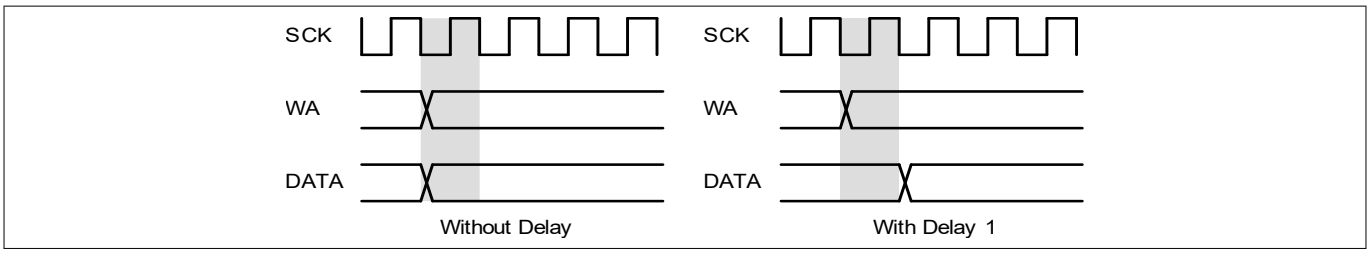


Figure 123 Transfer Delay for IIS

16.6.1.4 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.

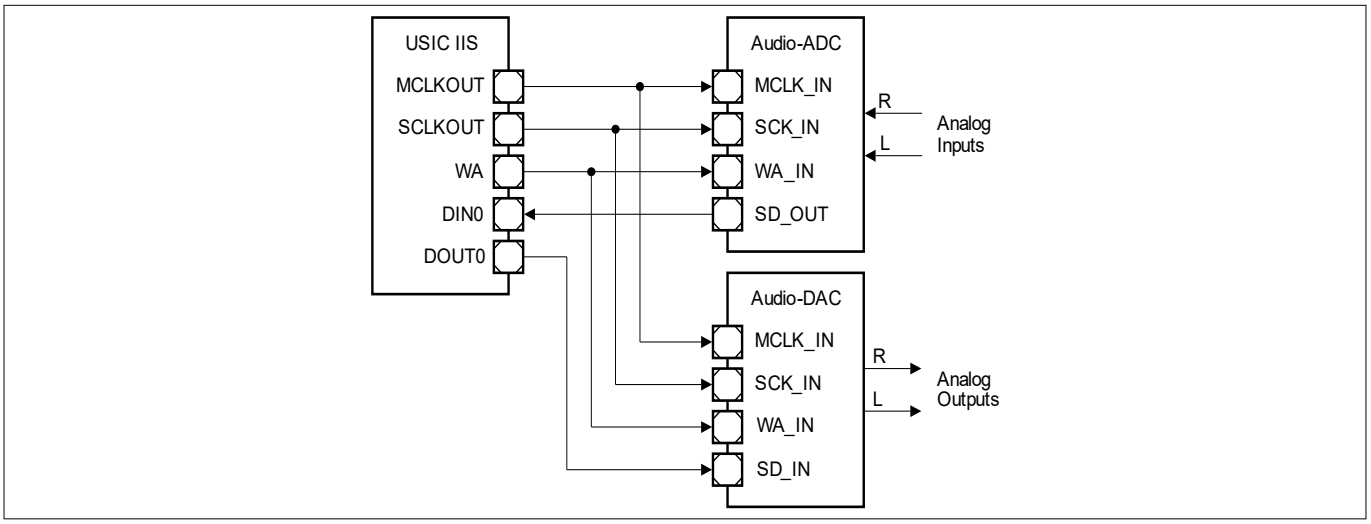


Figure 124 Connection of External Audio Devices

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).

16.6.2 Operating the IIS

This chapter contains IIS issues, that are of general interest and not directly linked to master mode or slave mode.

16.6.2.1 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.

16.6.2.2 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.

16.6.2.3 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.

16.6.2.4 Transfer Delay

The transfer delay can be used to synchronize a data transfer to an event (for example, 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 123, 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 125, 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.

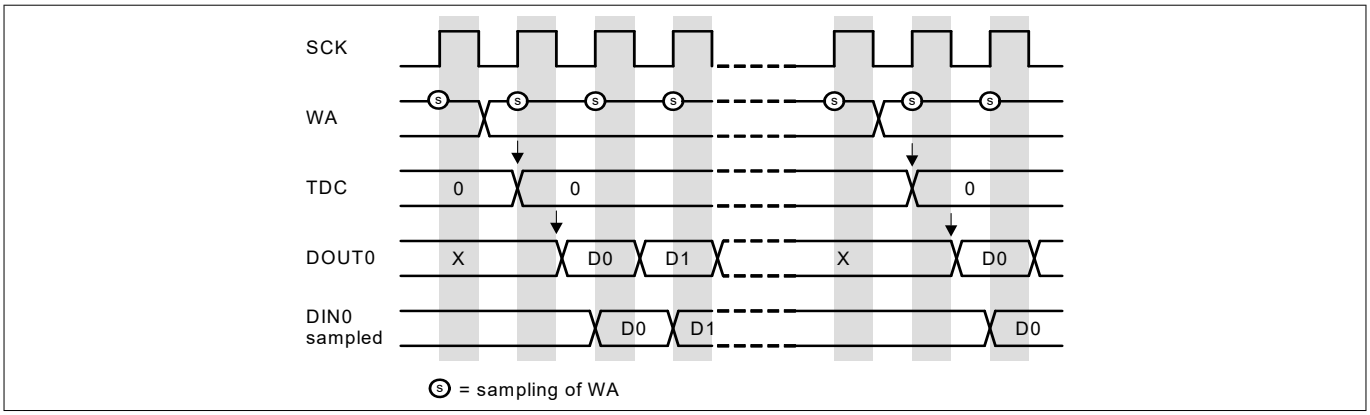


Figure 125 Transfer Delay with Delay 1

The ideal case without any transfer delay is shown in Figure 126. 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, by the system word length minus 1.

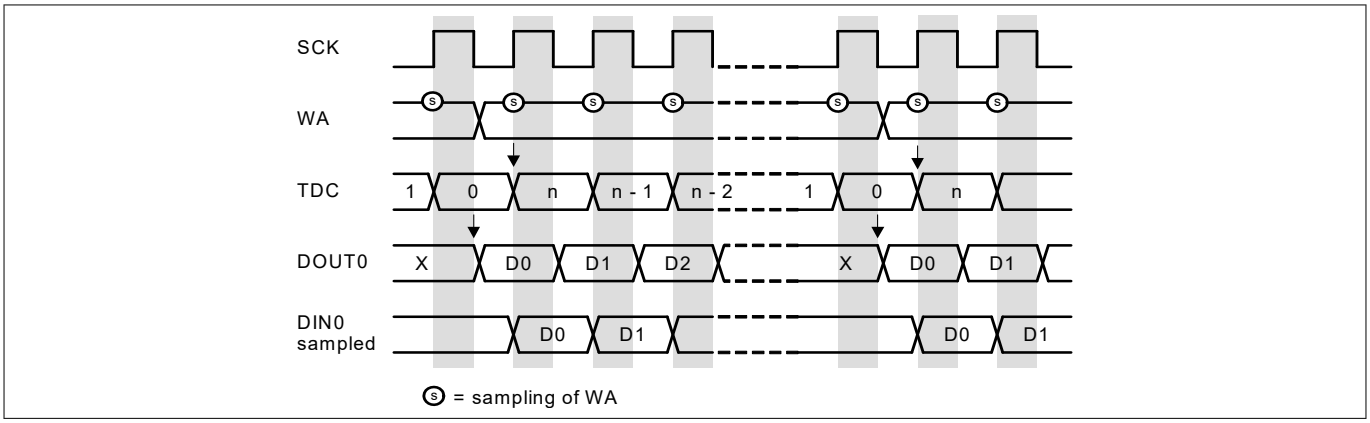


Figure 126 No Transfer Delay

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.

16.6.2.5 Data Transfer Interrupt Handling

The data transfer interrupts indicate events related to IIS frame handling.

Table 103 IIS Data Transfer Interrupt Handling

Interrupt	Indicated by bit	Description
Transmit buffer interrupt	PSR.TBIF	Set after the start of first data bit of a data word.
Transmit shift interrupt	PSR.TSIF	Set after the start of the last data bit of a data word.
Receiver start interrupt	PSR.RSIF	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. This is the earliest point in time when a new data word can be written to TBUF.
Receiver interrupt	PSR.RIF	Set 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.
Alternative interrupt	PSR.AIF	Set 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.
Data lost interrupt	PSR.DLIF	Set 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.

16.6.2.6 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 for the left and the right channel if the corresponding events are directed to different interrupt nodes. Flag PAR (RBUFSR[8]) is always 0.

16.6.2.7 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 any one of TBUF00 to TBUF15 (or IN00 to IN15 if a FIFO data buffer is used) is considered as left channel data, whereas data written to any one of TBUF16 to TBUF31 (or IN16 to IN31 if a FIFO data buffer is used) is considered as right channel data.

16.6.2.8 Receive Buffer Handling

If a receive FIFO buffer is 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.

16.6.2.9 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.

16.6.3 Operating the IIS in Master Mode

In order to operate the IIS in master mode, the USIC channel has to be first initialized.

It is recommended to configure all parameters of the IIS that do not change during run time while CCR.MODE = 0000_B, except stated otherwise.

The main initialization steps are outlined below:

- **Enable USIC channel**
  - Enable the module by writing 1s to MODEN and BPMODEN bits in KSCFG register.
- **Configure baud rate generator**
  - Select either fractional divider mode or normal divider mode with FDR.DM bit.
  - Configure the baud rate setting through register BRG and FDR.STEP bit field.
- **Configure input pins**
  - Establish a connection of input stage DX0 with the receive data input pin (signal DIN0) selected by DX0CR.DSEL bit field and with bit DX0CR.INSW = 1.

    The data shift unit allowing full-duplex data transfers based on the same word address (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).
  - Program Bit DX1CR.INSW = 0 to use the baud rate generator output SCLK directly as input for the data shift unit.
  - Program Bit DX2CR.INSW = 0 to use the WA generator output as input for the data shift unit.
- **Configure data format**
  - The word length, the frame length, the shift direction and the shift mode have to be set up according to the application requirements by programming the register SCTR. Generally, the MSB is shifted first (SCTR.SDIR = 1).
  - Write SCTR.TRM = 11_B to enable IIS data transfers.
- **Configure data transfer parameters**
  - Write TCSR.TDSSM = 1 and TCSR.TDEN = 01_B to enable data transmission in single shot mode.
  - 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.
- **Configure protocol control parameters**
  - Enable WA generation by setting bit field PCR.WAGEN = 1.
  - Configure the system word length through PCTQ and DCTQ bit fields in BRG register.
  - Write BRG.CTQSEL = 10_B to base WA toggling on SCLK signal.
- **Select IIS protocol**
  - Enable IIS mode with CCR.MODE = 0011_B. SCLK and WA generation are now started even though no valid data bits are transferred.
- **Configure output pins**
  - Configure the transmit data (signal DOUT0), the shift clock (signal SCLKOUT) and WA (signal SELOx) output pins through the selected pins' port control registers Pn_IOCR0/4/8/12. Refer to the port chapter.
  - The step to enable the output pin functions should only be done after the IIS mode is enabled with CCR.MODE, to avoided unintended spikes on the output.

An initialization code example is given in Initialization Code Example.

16.6.3.1 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}.

The standard setting is given by PPPEN = 0 ($f_{PPP} = f_{PIN}$) and CLKSEL = 00_B ($f_{PIN} = f_{FD}$). Under these conditions, the baud rate is given by either (30) or (31), depending on the selected divider mode:

- In normal divider mode (FDR.DM = 01_B)

$$f_{SCLK} = \frac{1}{2} \times f_{PERIPH} \times \frac{1}{1024 - STEP} \times \frac{1}{PDIV + 1} \quad (30)$$

- In fractional divider mode (FDR.DM = 10_B)

$$f_{SCLK} = \frac{1}{2} \times f_{PERIPH} \times \frac{STEP}{1024} \times \frac{1}{PDIV + 1} \quad (31)$$

Table 104 shows examples of the baud rate calculation in fractional divider mode (FDR.DM = 10_B).

Table 104 IIS Baud Rate Calculation in Fractional Divider Mode

f_{PERIPH} (MHz)	FDR.STEP	BRG.PDIV	f_{SCLK} (kbit/s)	Deviation Error
48	482	7	1411.2	0.06%
48	524	7	1536	-0.05%
48	482	3	2822.4	0.06%
48	524	3	3072	-0.05%

***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.*

16.6.3.2 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

$$N = (PCTQ + 1) \times (DCTQ + 1) \quad (32)$$

16.6.3.3 Master Clock Output

The master clock signal MCLK can be generated by the master of the IIS transfer (PCR.MCLK = 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.

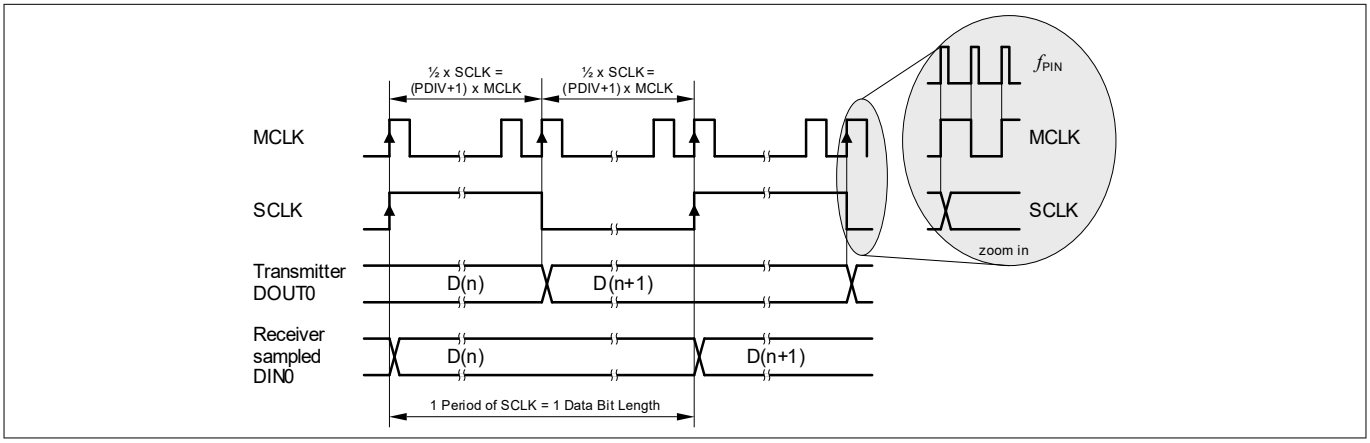


Figure 127 MCLK and SCLK for IIS

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 (for example, 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 as shown in Figure 127.

16.6.3.4 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.

Table 105 IIS master mode protocol interrupt events

Events	Event flag	Description	Flag clear	Interrupt enable
WA falling edge interrupt	PSR.WAFE	Set after a falling edge of the WA output signal has been generated.	PSCR.CST4	PCR.WAFEIEN
WA rising edge interrupt	PSR.WARE	Set after a rising edge of the WA output signal has been generated.	PSCR.CST5	PCR.WAREIEN
WA end interrupt	PSR.END	Set after the WA generation is stopped after it has been disabled by writing PCR.WAGEN = 0.	PSCR.CST6	PCR.ENDIEN
DX2T interrupt	PSR.DX2TEV	Set after an activation of the trigger signal DX2T. 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.	PSCR.CST3	PCR.DX2TIEN

16.6.3.5 Initialization Code Example

The following code example implements a function to initialize a USIC channel for IIS master mode data transfers with a baud rate of 1500 kbit/s (f_{PERIPH} = 48 MHz):

```
void USIC0_CH1_IIS_MasterMode_Init(void)
{
/// ----------------------------------------------------------
/// 1.Enable USIC0 channel 1
/// ----------------------------------------------------------
// BPMODEN MODEN
USIC0_CH1->KSCFG|=(1<<1)|(1<<0);
/// ----------------------------------------------------------
/// 2.Configure baud rate generator
/// - Fractional divider mode
/// - Baud rate = 1500 kbit/s
/// - System word length = 16
/// ----------------------------------------------------------
// DM STEP
USIC0_CH1->FDR=(2<<14)|(512<<0);
// PDIV DCTQ PCTQ CTQSEL
USIC0_CH1->BRG =(7<<16)|(3<<10)|(3<<8)|(2<<6);
/// ----------------------------------------------------------
/// 3.Configure input stages
/// - Select input DX0C
/// - Derive input of data shift unit directly from input pin
/// ----------------------------------------------------------
// INSW DSEL
USIC0_CH1->DX0CR=(1<<4)|(2<<0);
/// ----------------------------------------------------------
/// 4.Configure data format
/// - Data word = data frame = 15 bits
/// - Data transfer allowed with MSB first
/// ----------------------------------------------------------
// WLE FLE TRM SDIR
USIC0_CH1->SCTR=(15<<24)|(15<<16)|(3<<8)|(1<<0);
/// ----------------------------------------------------------
/// 5.Configure data transfer parameters
/// - Single shot transmission of data word when a valid word
/// is available
/// - Allow automatic upate of TCSR.WA
/// ----------------------------------------------------------
// TDEN TDSSM WAMD
USIC0_CH1->TCSR=(1<<10)|(1<<8)|(1<<3);
/// ----------------------------------------------------------
/// 6.Configure IIS protocol-specific parameters
/// - WA generation is enabled with inversion
/// - Data transfers are enabled
/// ----------------------------------------------------------
// SELINV DTEN WAGEN
USIC0_CH1->PCR=(1<<2)|(1<<1)|(1<<0);
/// ----------------------------------------------------------
/// 7.Enable IIS protocol
```

```
/// ----------------------------------------------------------
// MODE
USIC0_CH1->CCR=(3<<0);
/// ----------------------------------------------------------
/// 8.Configure IIS output function pins
/// - Assume
/// - P0.7 ALT7 function is assigned to DOUT0
/// - P0.8 ALT7 function is assigned to SCLKOUT
/// - P0.9 ALT7 function is assigned to WA
/// ----------------------------------------------------------
// PC7
PORT0->IOCR4|=(0x27<<26);
// PC9 PC8
PORT0->IOCR8|=(0x27<<10)|(0x27<<2);
}
```

16.6.4 Operating the IIS in Slave Mode

In order to operate the IIS in slave mode, the USIC channel has to be first initialized.

It is recommended to configure all parameters of the IIS that do not change during run time while CCR.MODE = 0000_B, except stated otherwise.

The main initialization steps are outlined below:

- **Enable USIC channel**
  - Enable the module by writing 1s to MODEN and BPMODEN bits in KSCFG register.
- **Configure input pins**
  - Establish a connection of input stage DX0 with the receive data input pin (DIN0) selected by DX0CR.DSEL bit field and with bit DX0CR.INSW = 1.

    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).
  - Establish a connection of input stage DX1 with the shift clock input pin (SCLKIN) selected by DX1CR.DSEL bit field and with DX1CR.INSW = 1 and DX1CR.DPOL = 1 (inverted polarity).
  - Establish a connection of input stage DX2 with the WA input pin (SELIN) selected by DX2CR.DSEL bit field and with DX2CR.INSW = 1.
- **Configure data format**
  - 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.
  - Write SCTR.TRM = 11_B to enable IIS data transfers.

- **Configure data transfer parameters**
  - Write TCSR.TDSSM = 1 and TCSR.TDEN = 01_B to enable data transmission in single shot mode.
  - 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.
- **Select IIS protocol**
  - Enable IIS mode with CCR.MODE = 0011_B.
- **Configure output pins**
  - Configure the transmit data (signal DOUT0) output pin through the selected pin's port control register Pn_IOCR0/4/8/12. Refer to the port chapter.
  - The step to enable the output pin functions should only be done after the IIS mode is enabled with CCR.MODE, to avoided unintended spikes on the output.

An initialization code example is given in Initialization Code Example.

***Note**: In IIS slave mode, the baud rate generator and the WA generation are not needed and can be switched off. All related bit fields (for example, FDR.DM, PCR.WAGEN, etc.) can be programmed to 0.*

16.6.4.1 Protocol Interrupt Events

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.

Table 106 IIS slave mode protocol interrupt events

Events	Event flag	Description	Flag clear	Interrupt enable
DX2T interrupt	PSR.DX2TEV	Set after an activation of the trigger signal DX2T.	PSCR.CST3	PCR.DX2TIEN

16.6.4.2 Initialization Code Example

The following code example implements a function to initialize a USIC channel for IIS slave mode data transfers:

```
void USIC0_CH1_IIS_SlaveMode_Init(void)
{
/// ----------------------------------------------------------
/// 1.Enable USIC0 channel 1
/// ----------------------------------------------------------
// BPMODEN MODEN
USIC0_CH1->KSCFG|= (1 << 1)| (1 << 0);
/// ----------------------------------------------------------
/// 2.Configure input stages
/// - Select inputs DX0D, DX1B and DX2B
/// - Derive all input signals directly from input pins
/// ----------------------------------------------------------
// INSW DSEL
USIC0_CH1->DX0CR =(1 << 4) |(3 << 0);
// DPOL INSW DSEL
USIC0_CH1->DX1CR =(1 << 8) |(1 << 4) |(1 << 0);
// INSW DSEL
USIC0_CH1->DX2CR =(1 << 4) |(1 << 0);
/// ----------------------------------------------------------
/// 3.Configure data format
/// - Data word = data frame = 15 bits
/// - Data transfer allowed with MSB first
/// ----------------------------------------------------------
// WLE FLE TRM SDIR
USIC0_CH1->SCTR =(15<<24)|(15<<16)|(3<<8)|(1<<0);
/// ----------------------------------------------------------
/// 4.Configure data transfer parameters
/// - Single shot transmission of data word when a valid word
/// is available
/// - Allow automatic upate of TCSR.WA
/// ----------------------------------------------------------
// TDEN TDSSM WAMD
USIC0_CH1->TCSR=(1<<10)|(1<<8)|(1<<3);
/// ----------------------------------------------------------
/// 5.Enable IIS protocol
/// ----------------------------------------------------------
// MODE
USIC0_CH1->CCR =(3 << 0);
/// ----------------------------------------------------------
/// 6.Configure IIS output function pins
/// - Assume P0.6 ALT7 function is assigned to DOUT0
/// ----------------------------------------------------------
// PC6
PORT0->IOCR4 |= (0x27 << 18);
}
```

16.7 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).

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, for example, one for transmission, one for reception, one or two for protocol or error handling, or for the alternative receive events).

16.7.1 General Channel Events and Interrupts

The general event and interrupt structure is shown in Figure 128. 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, for example, 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 the NVIC interrupt nodes for CPU processing of the service requests. This assignment is described in the interconnects section on Interconnects.

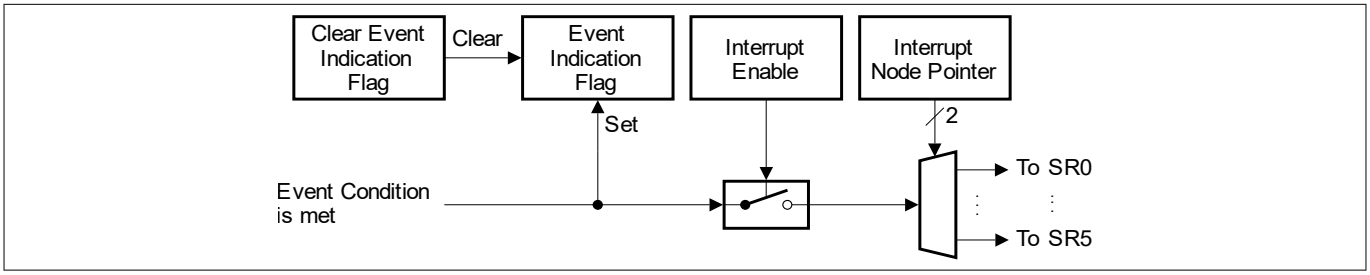


Figure 128 General Event and Interrupt Structure

16.7.2 Data Transfer Events and Interrupts

The data transfer events are based on the transmission or reception of a data word and are described in the corresponding protocol chapters.

All events can be individually enabled for interrupt generation.

Table 107 shows the registers, bits and bit fields indicating the data transfer events and controlling the interrupts of a USIC channel.

Table 107 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 129 shows the two transmit events and interrupts.

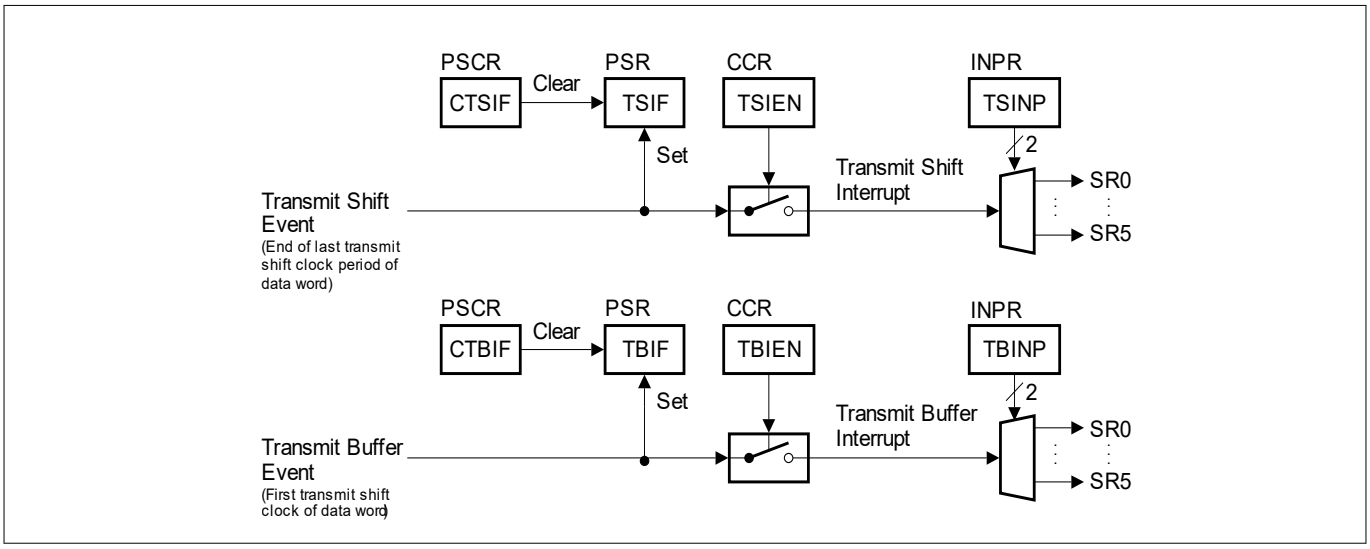


Figure 129 Transmit Events and Interrupts

Figure 130 shows the receive events and interrupts.

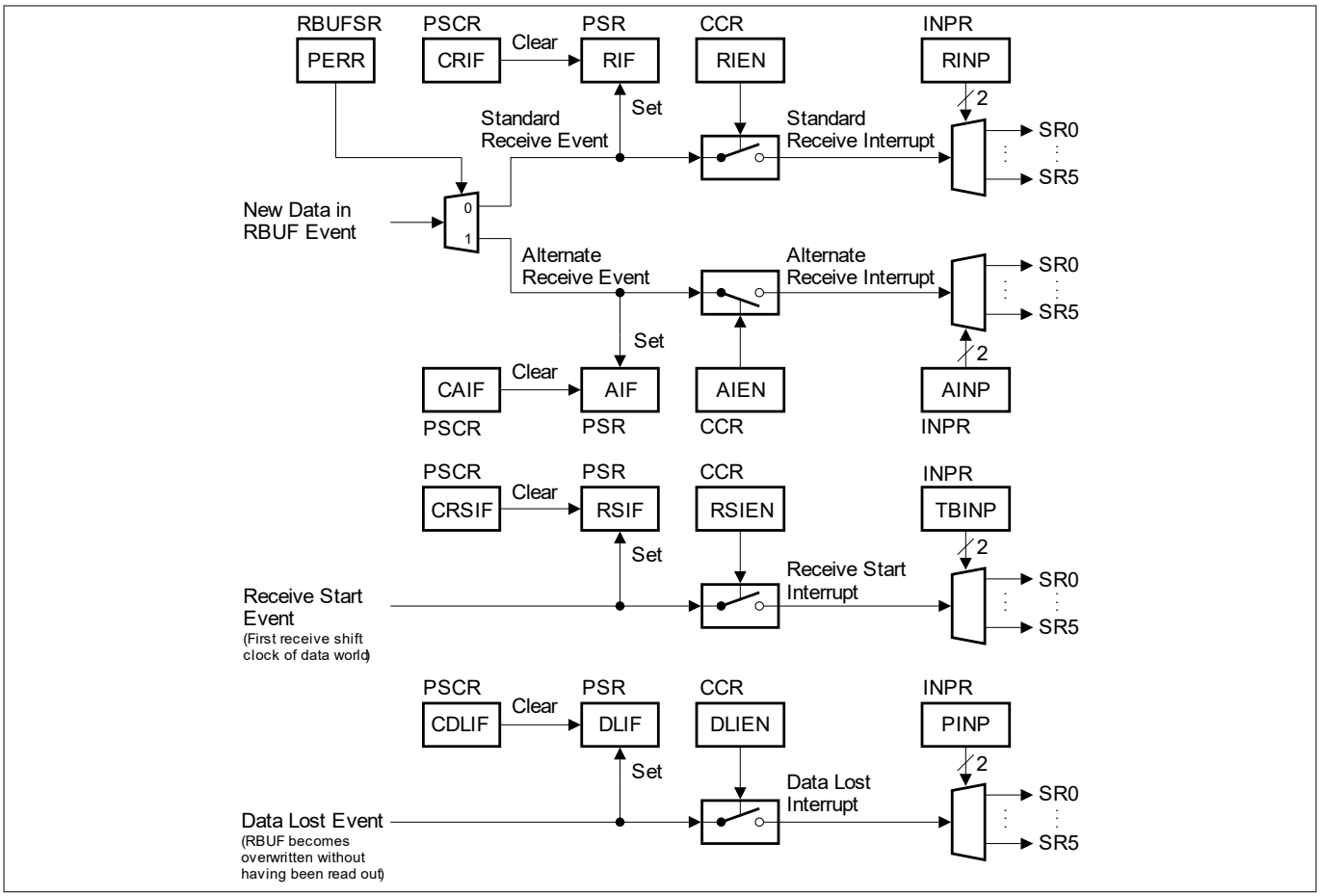


Figure 130 Receive Events and Interrupts

16.7.3 Baud Rate Generator Event and Interrupt

The baud rate generator event is based on the capture mode timer reaching its maximum value.

It can be enabled for the generation of a protocol interrupt.

Table 108 shows the registers, bits and bit fields indicating the baud rate generator event and controlling the interrupt of a USIC channel.

Table 108 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 131 shows the baud rate generator event and interrupt.

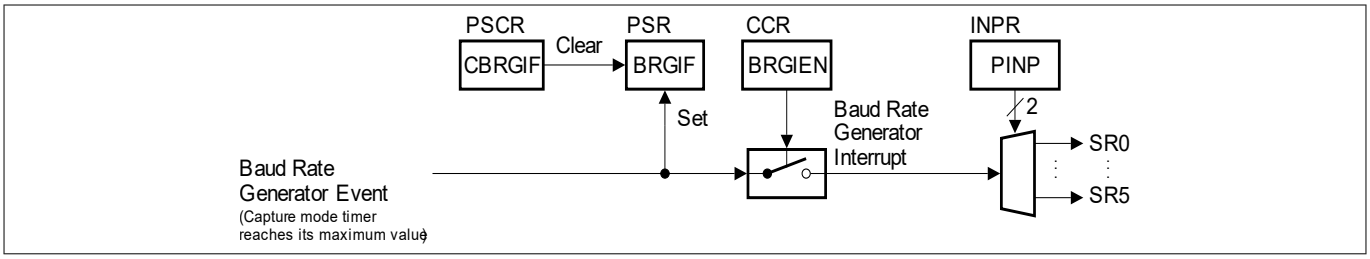


Figure 131 Baud Rate Generator Event and Interrupt

16.7.4 Protocol-specific Events and Interrupts

These events are related to protocol-specific actions that are described in the corresponding protocol chapters.

- ASC (Protocol Interrupt Events)
- SSC master mode (Protocol Interrupt Events)
- SSC slave mode (Protocol Interrupts)
- IIC (IIC Protocol Interrupt Events)
- IIS master mode (Protocol Interrupt Events)
- IIS slave mode (Protocol Interrupt Events)

All events can be individually enabled for the generation of the common protocol interrupt.

Table 109 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

16.8 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.

Selection is done through the bit field KSCFG.SUMCFG and the definition of the four kernel modes may differ across protocols. Refer to the protocol sections for details:

- ASC (Mode Control Behavior)
- SSC (Mode Control Behavior)
- IIC (Mode Control Behavior)
- IIS mode (Mode Control Behavior)

To avoid disturbing the receive data sequence and causing the loss of data, the debugger read accesses should not target the receive buffer RBUF (or OUTR if the receive FIFO buffer is used). Instead, the alternative address location at RBUFD (or OUTDR) should be used. A read at this location delivers the same value as RBUF (or OUTR) but does not affect the status of the read data.

16.9 Power, Reset and Clock

The USIC module is located in the core power domain. The module can be reset to its default state by a system reset.

The USIC module is clocked by the main clock, MCLK, from SCU. MCLK is disabled by default and can be enabled via the SCU_CGATCLR0 register. Enabling and disabling the module clock could cause a load change and clock blanking could occur as described in the CCU (Clock Gating Control) section of the SCU chapter. It is strongly recommended to set up the module clock in the user initialization code to avoid clock blanking during runtime.

***Note**: To differentiate from the USIC baud rate generator output, master clock (MCLK), the SCU MCLK is referenced throughout the USIC chapter as f_{PERIPH}.*

16.10 Initialization and System Dependencies

The application has to apply the following initialization sequence before operating the USIC module:

- Disable the system level clock gating of the USIC module(s) by writing one(s) to the USICx bits in the SCU register CGATCLR0.

For the initialization of the USIC channel for a specific protocol, refer to the protocol section:

- ASC (Operating the ASC)
- SSC master mode (Operating the SSC in Master Mode)
- SSC slave mode (Operating the SSC in Slave Mode)
- IIC (Operating the IIC)
- IIS master mode (Operating the IIS in Master Mode)
- IIS slave mode (Operating the IIS in Slave Mode)

16.11 Registers

Table 110 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.

All registers can be accessed with any access width (8-bit, 16-bit, 32-bit), independent of the described width.

***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 132 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_".

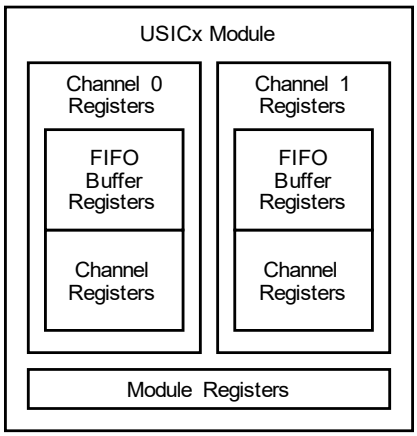


Figure 132 USIC Module and Channel Registers

Table 110 USIC Kernel-Related and Kernel Registers

Register Name	Register Long Name	Offset Addr.	Access Mode	
			Read	Write
Module Registers[1)]				
ID	Module Identification Register	008_H	U, PV	U, PV
Channel Registers				
–	reserved	000_H	BE	BE
CCFG	Channel Configuration Register	004_H	U, PV	U, PV
KSCFG	Kernel State Configuration Register	$00C_H$	U, PV	U, PV
FDR	Fractional Divider Register	010_H	U, PV	PV
BRG	Baud Rate Generator Register	014_H	U, PV	PV
INPR	Interrupt Node Pointer Register	018_H	U, PV	U, PV
DX0CR	Input Control Register 0	$01C_H$	U, PV	U, PV
DX1CR	Input Control Register 1	020_H	U, PV	U, PV
DX2CR	Input Control Register 2	024_H	U, PV	U, PV
DX3CR	Input Control Register 3	028_H	U, PV	U, PV
DX4CR	Input Control Register 4	$02C_H$	U, PV	U, PV

(table continues...)

Table 110 (continued) USIC Kernel-Related and Kernel Registers

Register Name	Register Long Name	Offset Addr.	Access Mode	
			Read	Write
DX5CR	Input Control Register 5	030_H	U, PV	U, PV
SCTR	Shift Control Register	034_H	U, PV	U, PV
TCSR	Transmit Control/Status Register	038_H	U, PV	U, PV
PCR	Protocol Control Register	$03C_H$	U, PV	U, PV[2]
			U, PV	U, PV[3]
			U, PV	U, PV[4]
			U, PV	U, PV[5]
			U, PV	U, PV[6]
CCR	Channel Control Register	040_H	U, PV	PV
CMTR	Capture Mode Timer Register	044_H	U, PV	U, PV
PSR	Protocol Status Register	048_H	U, PV	U, PV[2]
			U, PV	U, PV[3]
			U, PV	U, PV[4]
			U, PV	U, PV[5]
			U, PV	U, PV[6]
PSCR	Protocol Status Clear Register	$04C_H$	U, PV	U, PV
RBUFSR	Receiver Buffer Status Register	050_H	U, PV	U, PV
RBUF	Receiver Buffer Register	054_H	U, PV	U, PV
RBUFD	Receiver Buffer Register for Debugger	058_H	U, PV	U, PV
RBUF0	Receiver Buffer Register 0	$05C_H$	U, PV	U, PV
RBUF1	Receiver Buffer Register 1	060_H	U, PV	U, PV
RBUF01SR	Receiver Buffer 01 Status Register	064_H	U, PV	U, PV
FMR	Flag Modification Register	068_H	U, PV	U, PV
–	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
FIFO Buffer Registers				
BYP	Bypass Data Register	100_H	U, PV	U, PV
BYPCR	Bypass Control Register	104_H	U, PV	U, PV
TBCTR	Transmit Buffer Control Register	108_H	U, PV	U, PV
RBCTR	Receive Buffer Control Register	$10C_H$	U, PV	U, PV
TRBPTR	Transmit/Receive Buffer Pointer Register	110_H	U, PV	U, PV

(table continues...)

Table 110 (continued) USIC Kernel-Related and Kernel Registers

Register Name	Register Long Name	Offset Addr.	Access Mode	
			Read	Write
TRBSR	Transmit/Receive Buffer Status Register	114_H	U, PV	U, PV
TRBSCR	Transmit/Receive Buffer Status Clear Register	118_H	U, PV	U, PV
OUTR	Receive Buffer Output Register	$11C_H$	U, PV	U, PV
OUTDR	Receive Buffer Output Register for Debugger	120_H	U, PV	U, PV
–	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

1) Details of the module identification registers are described in the implementation section (see **USIC0_ID**).
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.

16.11.1 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 110) plus the channel base address (given in Table 111).

Table 111 Registers Address Space

Module	Base Address	End Address	Note
USIC0_CH0	48000000_H	$480001FF_H$	–
USIC0_CH1	48000200_H	$480003FF_H$	–
USIC1_CH0	48004000_H	$480041FF_H$	–
USIC1_CH1	48004200_H	$480043FF_H$	–

Table 112 FIFO and Reserved Address Space

Module	Base Address	End Address	Access Mode		Note
			Read	Write	
USIC0	48000400_H	$480007FF_H$	nBE	nBE if in direct RAM test mode; otherwise BE	USIC0 RAM area, shared between USIC0_CH0 and USIC0_CH1
reserved	48000800_H	$48003FFF_H$	BE	BE	–
USIC1	48004400_H	$480047FF_H$	nBE	nBE if in direct RAM test mode; otherwise BE	USIC1 RAM area, shared between USIC1_CH0 and USIC1_CH1
reserved	48004800_H	$48007FFF_H$	BE	BE	–

16.12 Interconnects

The PSC1Mx device contains two USIC modules (USIC0 and USIC1) with 2 communication channels each.

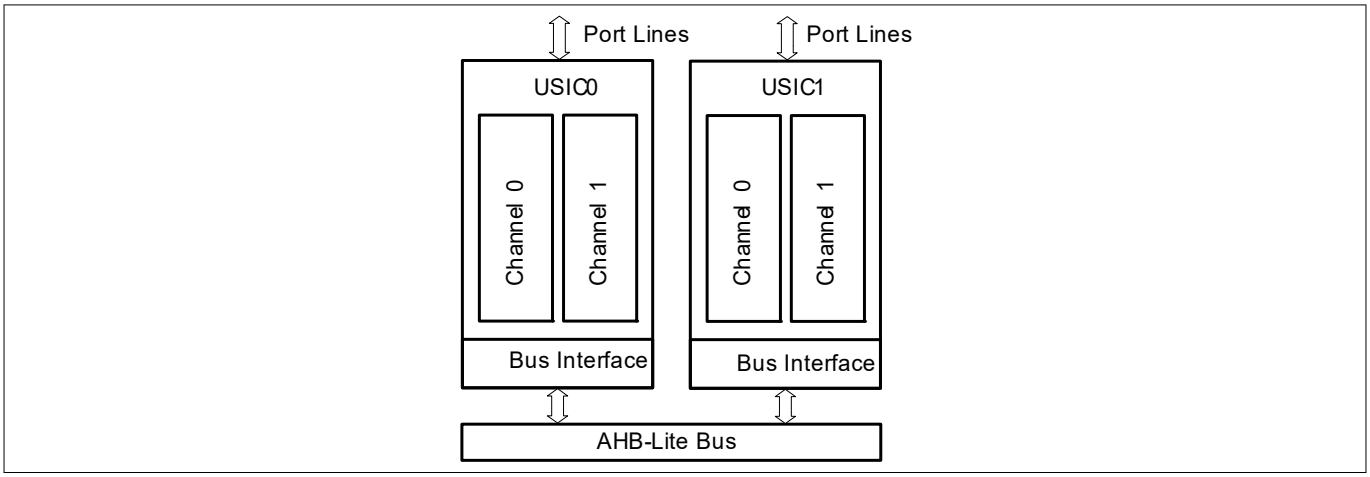


Figure 133 USIC Module Structure in PSC1Mx

The next sections define the pin assignments and internal connections of each USIC module and channel in the PSC1Mx device.

The meaning of these I/O lines with respect to each protocol is described in the protocols' respective chapters and summarized in Table 73 and Table 74.

Naming convention: USICx_CHy refers to USIC module x channel y.

16.12.1 USIC0 Module Interconnects

The interconnects of USIC0 module is grouped into the following categories:

- USIC0_CH0 Interconnects (USIC0 Channel 0 Interconnects)
- USIC0_CH1 Interconnects (USIC0 Channel 1 Interconnects)
- USIC0 Global Interconnects (USIC0 Global Interconnects)

16.12.1.1 USIC0 Channel 0 Interconnects

Table 113 shows the interconnects for USIC0 Channel 0.

Table 113 USIC0 Channel 0 Interconnects

Input/Output	I/O	Connected To	Description
Data Inputs (DX0)			
USIC0_CH0.DX0A	I	P0.14	Shift data input; used for: • ASC RXD • SSC MTSR/MRST • IIC SDA • IIS DIN
USIC0_CH0.DX0B	I	P0.15	
USIC0_CH0.DX0C	I	P1.0	
USIC0_CH0.DX0D	I	P1.1	
USIC0_CH0.DX0E	I	P2.0	
USIC0_CH0.DX0F	I	P2.1	
USIC0_CH0.DX0G	I	USIC0_CH0.DX3INS	

(table continues...)

Table 113 (continued) USIC0 Channel 0 Interconnects

Input/Output	I/O	Connected To	Description
USIC0_CH0.HWIN0	I	P1.0	HW controlled shift data input
Clock Inputs			
USIC0_CH0.DX1A	I	P0.14	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	P0.7	
USIC0_CH0.DX1D	I	P1.1	
USIC0_CH0.DX1E	I	P2.0	
USIC0_CH0.DX1F	I	USIC0_CH0.DX0INS	
USIC0_CH0.DX1G	I	USIC0_CH0.DX4INS	
Control Inputs			
USIC0_CH0.DX2A	I	P.0.0	Shift control input; used for: • SSC Slave SELIN • IIS Slave WAIN • Optional for ASC, SSC Master, IIC and IIS Master
USIC0_CH0.DX2B	I	P0.9	
USIC0_CH0.DX2C	I	P0.10	
USIC0_CH0.DX2D	I	P0.11	
USIC0_CH0.DX2E	I	P0.12	
USIC0_CH0.DX2F	I	P0.13	
USIC0_CH0.DX2G	I	USIC0_CH0.DX5INS	
Data Inputs (DX3)			
USIC0_CH0.DX3A	I	P2.2	Shift data input; used for: • Dual and Quad SSC MTSR1/MRST1 • Can be ignored for all other protocols
USIC0_CH0.DX3B	I	P2.4	
USIC0_CH0.DX3C	I	P2.10	
USIC0_CH0.DX3D	I	P2.8	
USIC0_CH0.DX3E	I	P2.6	
USIC0_CH0.DX3F	I	USIC0_CH0.DX5INS	
USIC0_CH0.DX3G	I	USIC0_CH0.DOUT0	
USIC0_CH0.HWIN1	I	P1.1	HW controlled shift data input
Data Inputs (DX4)			
USIC0_CH0.DX4A	I	P2.2	Shift data input; used for: • Quad SSC MTSR2/MRST2 • Can be ignored for all other protocols
USIC0_CH0.DX4B	I	P2.4	
USIC0_CH0.DX4C	I	P2.10	
USIC0_CH0.DX4D	I	P2.8	
USIC0_CH0.DX4E	I	P2.6	
USIC0_CH0.DX4F	I	USIC0_CH0.DX5INS	

(table continues...)

Table 113 (continued) USIC0 Channel 0 Interconnects

Input/Output	I/O	Connected To	Description
USIC0_CH0.DX4G	I	USIC0_CH0.SCLKOUT	
USIC0_CH0.HWIN2	I	P1.2	HW controlled shift data input
Data Inputs (DX5)			
USIC0_CH0.DX5A	I	P2.9	Shift data input; used for: • Quad SSC MTSR3/MRST3 • Can be ignored for all other protocols
USIC0_CH0.DX5B	I	P2.3	
USIC0_CH0.DX5C	I	P2.7	
USIC0_CH0.DX5D	I	P2.5	
USIC0_CH0.DX5E	I	P1.4	
USIC0_CH0.DX5F	I	P1.6	
USIC0_CH0.DX5G	I	USIC0_CH0.SELO0	
USIC0_CH0.HWIN3	I	P1.3	HW controlled shift data input
Data Outputs			
USIC0_CH0.DOUT0	O	P0.14; P0.15; P1.0; P1.1; P2.0; P2.1; P1.0 (HW1_OUT); USIC0_CH0.DX3G	Shift data output; used for: • ASC TXD • SSC MTSR/MRST • IIC SDA • IIS DOUT
USIC0_CH0.DOUT1	O	P1.1 (HW1_OUT)	Shift data output; used for: • Dual and Quad SSC MTSR1/MRST1 • Can be ignored for all other protocols
USIC0_CH0.DOUT2	O	P1.2 (HW1_OUT)	Shift data output; used for: • Quad SSC MTSR2/MRST2 • Can be ignored for all other protocols
USIC0_CH0.DOUT3	O	P1.3 (HW1_OUT)	Shift data output; used for: • Quad SSC MTSR3/MRST3 • Can be ignored for all other protocols
Clock Outputs			
USIC0_CH0.MCLKOUT	O	P0.11	Master clock output (optional for all protocols)

(table continues...)

Table 113 (continued) USIC0 Channel 0 Interconnects

Input/Output	I/O	Connected To	Description
USIC0_CH0.SCLKOUT	O	P0.7; P0.8; P0.14; P1.6; P2.0; USIC0_CH0.DX4G	Shift clock output; used for: • Master SCLKOUT in SSC and IIS • IIC SCL • Can be ignored in ASC
Control Outputs			
USIC0_CH0.SELO0	O	P0.0; P0.9; P1.4; USIC0_CH0.DX5G	Shift control output; used for: • SSC Master SELO • IIS WA • Can be ignored for all other protocols
USIC0_CH0.SELO1	O	P0.10; P1.5	
USIC0_CH0.SELO2	O	P0.11; P1.6	
USIC0_CH0.SELO3	O	P0.12	
USIC0_CH0.SELO4	O	P0.13	
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	not connected	Selected DX1 input signal
USIC0_CH0.DX2INS	O	CCU40.IN0AL	Selected DX2 input signal
USIC0_CH0.DX3INS	O	USIC0_CH0.DX0G	Selected DX3 input signal
USIC0_CH0.DX4INS	O	USIC0_CH0.DX1G	Selected DX4 input signal
USIC0_CH0.DX5INS	O	USIC0_CH0.DX2G; USIC0_CH0.DX3F; USIC0_CH0.DX4F	Selected DX5 input signal

Figure 134 shows a graphical representation of the USIC0 Channel 0 interconnects.

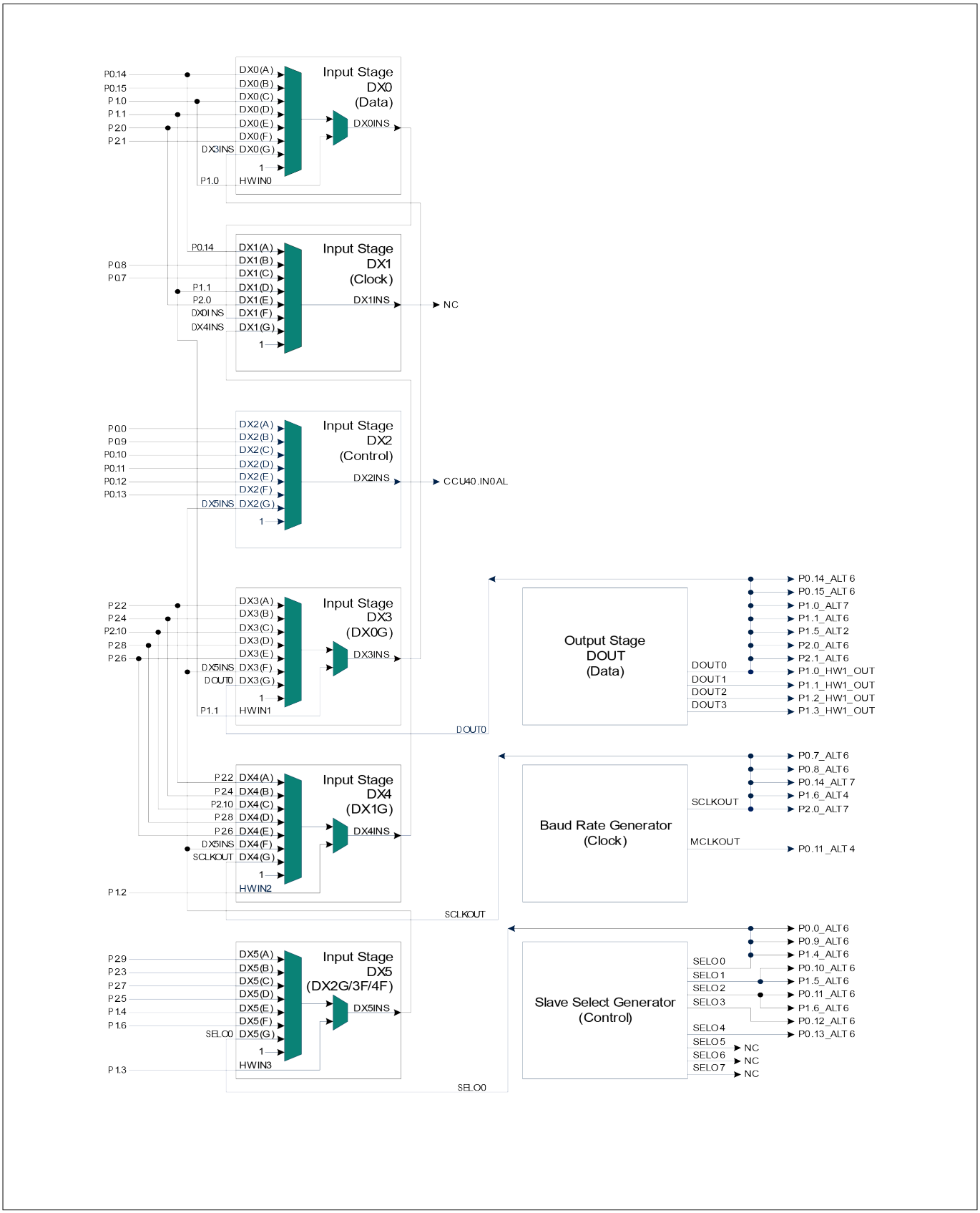


Figure 134 USIC0 Channel 0 Interconnects

16.12.1.2 USIC0 Channel 1 Interconnects

Table 114 shows the interconnects for USIC0 Channel 1.

Table 114 USIC0 Channel 1 Interconnects

Input/Output	I/O	Connected To	Description
Data Inputs (DX0)			
USIC0_CH1.DX0A	I	P1.3	Shift data input; used for: • ASC RXD • SSC MTSR/MRST • IIC SDA • IIS DIN
USIC0_CH1.DX0B	I	P1.2	
USIC0_CH1.DX0C	I	P0.6	
USIC0_CH1.DX0D	I	P0.7	
USIC0_CH1.DX0E	I	P2.11	
USIC0_CH1.DX0F	I	P2.10	
USIC0_CH1.DX0G	I	USIC0_CH1.DX3INS	
USIC0_CH1.HWIN0	I	ERU0.PDOUT0	HW controlled shift data input
Clock Inputs			
USIC0_CH1.DX1A	I	P1.3	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	P0.8	
USIC0_CH1.DX1C	I	P0.7	
USIC0_CH1.DX1D	I	0	
USIC0_CH1.DX1E	I	P2.11	
USIC0_CH1.DX1F	I	USIC0_CH1.DX0INS	
USIC0_CH1.DX1G	I	USIC0_CH1.DX4INS	
Control Inputs			
USIC0_CH1.DX2A	I	P0.0	Shift control input; used for: • SSC Slave SELIN • IIS Slave WAIN • Optional for ASC, SSC Master, IIC and IIS Master
USIC0_CH1.DX2B	I	P0.9	
USIC0_CH1.DX2C	I	P0.10	
USIC0_CH1.DX2D	I	P0.11	
USIC0_CH1.DX2E	I	P1.1	
USIC0_CH1.DX2F	I	P2.0	
USIC0_CH1.DX2G	I	USIC0_CH1.DX5INS	
Data Inputs (DX3)			
USIC0_CH1.DX3A	I	P2.1	Shift data input; used for: • Dual and Quad SSC MTSR1/MRST1 • Can be ignored for all other protocols
USIC0_CH1.DX3B	I	P2.9	
USIC0_CH1.DX3C	I	P2.3	
USIC0_CH1.DX3D	I	P2.7	
USIC0_CH1.DX3E	I	P2.5	

(table continues...)

Table 114 (continued) USIC0 Channel 1 Interconnects

Input/Output	I/O	Connected To	Description
USIC0_CH1.DX3F	I	USIC0_CH1.DX5INS	
USIC0_CH1.DX3G	I	USIC0_CH1.DOUT0	
USIC0_CH1.HWIN1	I	0	HW controlled shift data input
Data Inputs (DX4)			
USIC0_CH1.DX4A	I	P2.1	Shift data input; used for: • Quad SSC MTSR2/MRST2 • Can be ignored for all other protocols
USIC0_CH1.DX4B	I	P2.9	
USIC0_CH1.DX4C	I	P2.3	
USIC0_CH1.DX4D	I	P2.7	
USIC0_CH1.DX4E	I	P2.5	
USIC0_CH1.DX4F	I	USIC0_CH1.DX5INS	
USIC0_CH1.DX4G	I	USIC0_CH1.SCLKOU T	
USIC0_CH1.HWIN2	I	ERU0.PDOUT1	HW controlled shift data input
Data Inputs (DX5)			
USIC0_CH1.DX5A	I	P2.2	Shift data input; used for: • Quad SSC MTSR3/MRST3 • Can be ignored for all other protocols
USIC0_CH1.DX5B	I	P2.4	
USIC0_CH1.DX5C	I	P2.8	
USIC0_CH1.DX5D	I	P2.6	
USIC0_CH1.DX5E	I	P1.4	
USIC0_CH1.DX5F	I	P1.5	
USIC0_CH1.DX5G	I	USIC0.SR0	
USIC0_CH1.HWIN3	I	USIC0_CH1.DOUT0	HW controlled shift data input
Data Outputs			
USIC0_CH1.DOUT0	O	P0.6; P0.7; P1.2; P1.3; P1.6; P2.10; P2.11; USIC0_CH1.DX3G; USIC0_CH1.HWIN3	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

(table continues...)

Table 114 (continued) USIC0 Channel 1 Interconnects

Input/Output	I/O	Connected To	Description
USIC0_CH1.DOUT2	O	not connected	Shift data output; used for: • Quad SSC MTSR2/MRST2 • Can be ignored for all other protocols
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	P0.6; P0.15	Master clock output (optional for all protocols)
USIC0_CH1.SCLKOUT	O	P0.8; P1.3; P1.4; P2.1; P2.11; USIC0_CH1.DX4G	Shift clock output; used for: • Master SCLKOUT in SSC and IIS • IIC SCL • Can be ignored in ASC
Control Outputs			
USIC0_CH1.SELO0	O	P0.0; P0.9; P1.1	Shift control output; used for: • SSC Master SELO • IIS WA • Can be ignored for all other protocols
USIC0_CH1.SELO1	O	P0.10; P1.4	
USIC0_CH1.SELO2	O	P0.11; P1.5	
USIC0_CH1.SELO3	O	P1.6	
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.IN1AL	Selected DX2 input signal
USIC0_CH1.DX3INS	O	USIC0_CH1.DX0G	Selected DX3 input signal
USIC0_CH1.DX4INS	O	USIC0_CH1.DX1G	Selected DX4 input signal

(table continues...)

Table 114 (continued) USIC0 Channel 1 Interconnects

Input/Output	I/O	Connected To	Description
USIC0_CH1.DX5INS	O	USIC0_CH1.DX2G; USIC0_CH1.DX3F; USIC0_CH1.DX4F	Selected DX5 input signal

Figure 135 shows a graphical representation of the USIC0 Channel 1 interconnects.

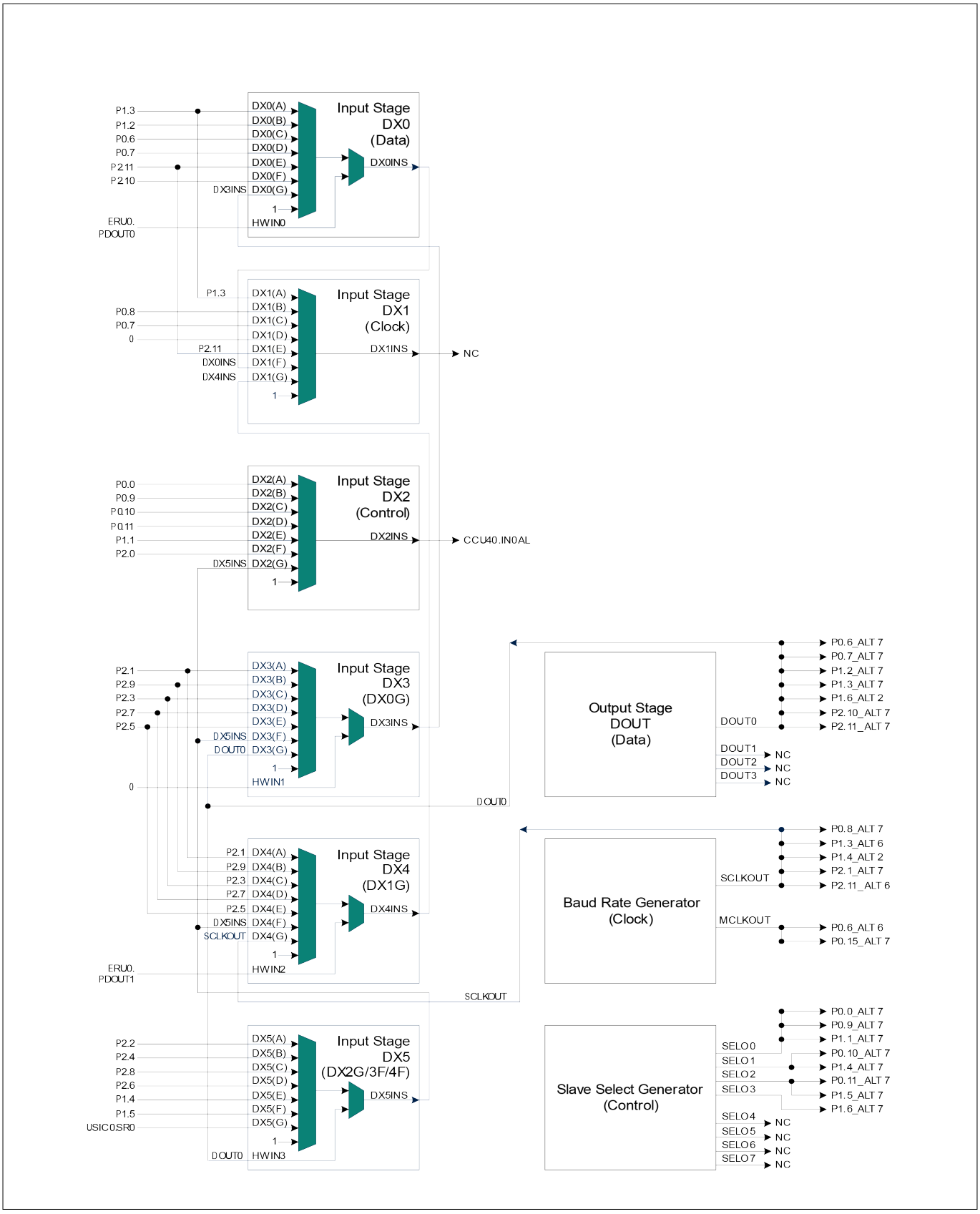


Figure 135 USIC0 Channel 1 Interconnects

16.12.1.3 USIC0 Global Interconnects

Table 115 shows the global interconnects for USIC0 module.

Table 115 USIC0 Global Interconnects

Input/Output	I/O	Connected To	Description
USIC0_SR0	O	NVIC; USIC0_CH1.DX5G	Service request Output 0
USIC0_SR1	O	NVIC	Service request Output 1
USIC0_SR2	O	NVIC	Service request Output 2
USIC0_SR3	O	NVIC	Service request Output 3
USIC0_SR4	O	NVIC	Service request Output 4
USIC0_SR5	O	NVIC	Service request Output 5

16.12.2 USIC1 Module Interconnects

The interconnects of USIC1 module is grouped into the following categories:

- USIC1_CH0 Interconnects (USIC1 Channel 0 Interconnects)
- USIC1_CH1 Interconnects (USIC1 Channel 1 Interconnects)
- USIC1 Global Interconnects (USIC1 Global Interconnects)

16.12.2.1 USIC1 Channel 0 Interconnects

Table 116 shows the interconnects for USIC1 Channel 0.

Table 116 USIC1 Channel 0 Interconnects

Input/Output	I/O	Connected To	Description
Data Inputs (DX0)			
USIC1_CH0.DX0A	I	P0.2	Shift data input; used for: • ASC RXD • SSC MTSR/MRST • IIC SDA • IIS DIN
USIC1_CH0.DX0B	I	P0.3	
USIC1_CH0.DX0C	I	P4.4	
USIC1_CH0.DX0D	I	P4.5	
USIC1_CH0.DX0E	I	–	
USIC1_CH0.DX0F	I	–	
USIC1_CH0.DX0G	I	USIC1_CH0.DX3INS	
USIC1_CH0.HWIN0	I	–	HW controlled shift data input
Clock Inputs			
USIC1_CH0.DX1A	I	P0.2	Shift clock input; used for: • SSC Slave SCLKIN • IIC SCL
USIC1_CH0.DX1B	I	P4.3	
USIC1_CH0.DX1C	I	P4.5	
USIC1_CH0.DX1D	I	P4.6	

(table continues...)

Table 116 (continued) USIC1 Channel 0 Interconnects

Input/Output	I/O	Connected To	Description
USIC1_CH0.DX1E	I	–	• IIS Slave SCLKIN • Optional for ASC, SSC Master and IIS Master
USIC1_CH0.DX1F	I	USIC1_CH0.DX0INS	
USIC1_CH0.DX1G	I	USIC1_CH0.DX4INS	
Control Inputs			
USIC1_CH0.DX2A	I	P4.7	Shift control input; used for: • SSC Slave SELIN • IIS Slave WAIN • Optional for ASC, SSC Master, IIC and IIS Master
USIC1_CH0.DX2B	I	P4.8	
USIC1_CH0.DX2C	I	P4.9	
USIC1_CH0.DX2D	I	–	
USIC1_CH0.DX2E	I	–	
USIC1_CH0.DX2F	I	–	
USIC1_CH0.DX2G	I	USIC1_CH0.DX5INS	
Data Inputs (DX3)			
USIC1_CH0.DX3A	I	P2.12	Shift data input; used for: • Dual and Quad SSC MTSR1/MRST1 • Can be ignored for all other protocols
USIC1_CH0.DX3B	I	–	
USIC1_CH0.DX3C	I	–	
USIC1_CH0.DX3D	I	–	
USIC1_CH0.DX3E	I	P2.3	
USIC1_CH0.DX3F	I	USIC1_CH0.DX5.INS	
USIC1_CH0.DX3G	I	USIC1_CH0.DOUT0	
USIC1_CH0.HWIN1	I	–	HW controlled shift data input
Data Inputs (DX4)			
USIC1_CH0.DX4A	I	P2.12	Shift data input; used for: • Quad SSC MTSR2/MRST2 • Can be ignored for all other protocols
USIC1_CH0.DX4B	I	–	
USIC1_CH0.DX4C	I	–	
USIC1_CH0.DX4D	I	–	
USIC1_CH0.DX4E	I	P2.3	
USIC1_CH0.DX4F	I	USIC1_CH0.DX5INS	
USIC1_CH0.DX4G	I	USIC1_CH0.SCLKOUT0	
USIC1_CH0.HWIN2	I	–	HW controlled shift data input
Data Inputs (DX5)			
USIC1_CH0.DX5A	I	P2.13	Shift data input; used for: • Quad SSC MTSR3/MRST3 • Can be ignored for all other protocols
USIC1_CH0.DX5B	I	–	
USIC1_CH0.DX5C	I	P4.1	

(table continues...)

Table 116 (continued) USIC1 Channel 0 Interconnects

Input/Output	I/O	Connected To	Description
USIC1_CH0.DX5D	I	P4.2	
USIC1_CH0.DX5E	I	P2.2	
USIC1_CH0.DX5F	I	P2.4	
USIC1_CH0.DX5G	I	USIC1_CH0.SELO0	
USIC1_CH0.HWIN3	I	–	HW controlled shift data input
Data Outputs			
USIC1_CH0.DOUT0	O	P0.2; P0.3; P4.4; P4.5; USIC1_CH0.DX3G	Shift data output; used for: • ASC TXD • SSC MTSR/MRST • IIC SDA • IIS DOUT
USIC1_CH0.DOUT1	O	–	Shift data output; used for: • Dual and Quad SSC MTSR1/MRST1 • Can be ignored for all other protocols
USIC1_CH0.DOUT2	O	–	Shift data output; used for: • Quad SSC MTSR2/MRST2 • Can be ignored for all other protocols
USIC1_CH0.DOUT3	O	–	Shift data output; used for: • Quad SSC MTSR3/MRST3 • Can be ignored for all other protocols
Clock Outputs			
USIC1_CH0.MCLKOUT	O	P2.13	Master clock output (optional for all protocols)
USIC1_CH0.SCLKOUT	O	P0.2; P2.12; P4.3; P4.5; P4.6; USIC1_CH0.DX4G	Shift clock output; used for: • Master SCLKOUT in SSC and IIS • IIC SCL • Can be ignored in ASC
Control Outputs			
USIC1_CH0.SELO0	O	P4.7; USIC1_CH0.DX5G	Shift control output; used for: • SSC Master SELO • IIS WA • Can be ignored for all other protocols
USIC1_CH0.SELO1	O	P3.0; P4.8	
USIC1_CH0.SELO2	O	P4.9	
USIC1_CH0.SELO3	O	–	

(table continues...)

Table 116 (continued) USIC1 Channel 0 Interconnects

Input/Output	I/O	Connected To	Description
USIC1_CH0.SELO4	O	–	
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	not connected	Selected DX1 input signal
USIC1_CH0.DX2INS	O	CCU41.IN0BC	Selected DX2 input signal
USIC1_CH0.DX3INS	O	USIC1_CH0.DX0G	Selected DX3 input signal
USIC1_CH0.DX4INS	O	USIC1_CH0.DX1G	Selected DX4 input signal
USIC1_CH0.DX5INS	O	USIC1_CH0.DX2G USIC1_CH0.DX3F; USIC1_CH0.DX4F	Selected DX5 input signal

Figure 136 shows a graphical representation of the USIC1 Channel 0 interconnects.

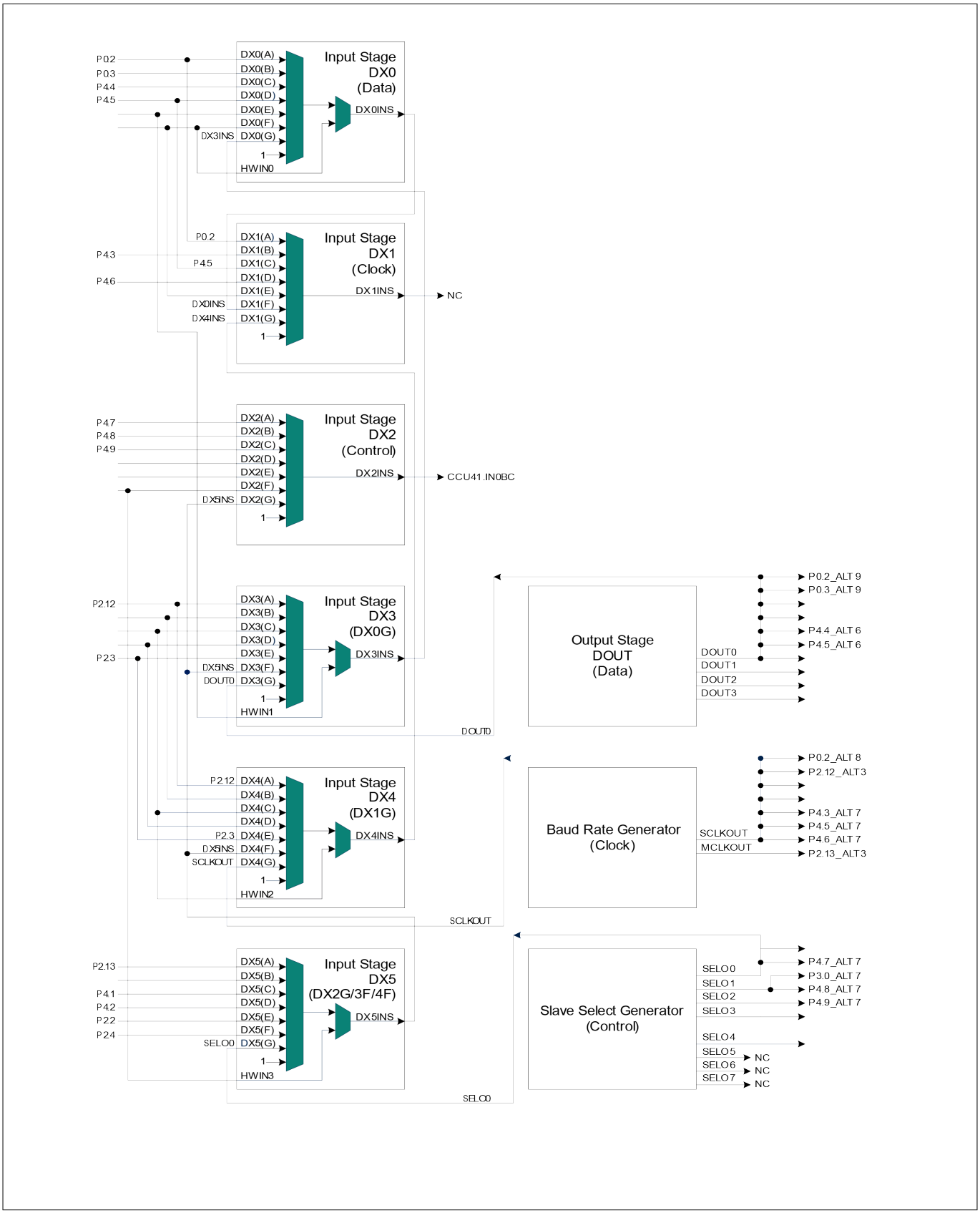


Figure 136 USIC1 Channel 0 Interconnects

16.12.2.2 USIC1 Channel 1 Interconnects

Table 117 shows the interconnects for USIC1 Channel 1.

Table 117 USIC1 Channel 1 Interconnects

Input/Output	I/O	Connected To	Description
Data Inputs (DX0)			
USIC1_CH1.DX0A	I	P0.0	Shift data input; used for: • ASC RXD • SSC MTSR/MRST • IIC SDA • IIS DIN
USIC1_CH1.DX0B	I	P0.1	
USIC1_CH1.DX0C	I	P2.12	
USIC1_CH1.DX0D	I	P2.13	
USIC1_CH1.DX0E	I	P3.0	
USIC1_CH1.DX0F	I	–	
USIC1_CH1.DX0G	I	USIC1_CH1.DX3INS	
USIC1_CH1.HWIN0	I	ERU1.PDOUT0	HW controlled shift data input
Clock Inputs			
USIC1_CH1.DX1A	I	P0.1	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	P2.12	
USIC1_CH1.DX1C	I	–	
USIC1_CH1.DX1D	I	P3.0	
USIC1_CH1.DX1E	I	0	
USIC1_CH1.DX1F	I	USIC1_CH1.DX0INS	
USIC1_CH1.DX1G	I	USIC1_CH1.DX4INS	
Control Inputs			
USIC1_CH1.DX2A	I	–	Shift control input; used for: • SSC Slave SELIN • IIS Slave WAIN • Optional for ASC, SSC Master, IIC and IIS Master
USIC1_CH1.DX2B	I	–	
USIC1_CH1.DX2C	I	–	
USIC1_CH1.DX2D	I	0	
USIC1_CH1.DX2E	I	0	
USIC1_CH1.DX2F	I	USIC.SR0	
USIC1_CH1.DX2G	I	USIC1_CH1.DX5.INS	
Data Inputs (DX3)			
USIC1_CH1.DX3A	I	–	Shift data input; used for: • Dual and Quad SSC MTSR1/MRST1 • Can be ignored for all other protocols
USIC1_CH1.DX3B	I	P0.15	
USIC1_CH1.DX3C	I	P2.4	
USIC1_CH1.DX3D	I	–	
USIC1_CH1.DX3E	I	P2.6	

(table continues...)

Table 117 (continued) USIC1 Channel 1 Interconnects

Input/Output	I/O	Connected To	Description
USIC1_CH1.DX3F	I	USIC1_CH1.DX5.INS	
USIC1_CH1.DX3G	I	USIC1_CH1.DOUT0	
USIC1_CH1.HWIN1	I	0	HW controlled shift data input
Data Inputs (DX4)			
USIC1_CH1.DX4A	I	–	Shift data input; used for: • Quad SSC MTSR2/MRST2 • Can be ignored for all other protocols
USIC1_CH1.DX4B	I	P0.15	
USIC1_CH1.DX4C	I	P2.4	
USIC1_CH1.DX4D	I	–	
USIC1_CH1.DX4E	I	P2.6	
USIC1_CH1.DX4F	I	USIC1_CH1.DX5INS	
USIC1_CH1.DX4G	I	USIC1_CH1.SCLKOUT0	
USIC1_CH1.HWIN2	I	ERU1.PDOUT1	HW controlled shift data input
Data Inputs (DX5)			
USIC1_CH1.DX5A	I	–	Shift data input; used for: • Quad SSC MTSR3/MRST3 • Can be ignored for all other protocols
USIC1_CH1.DX5B	I	P0.14	
USIC1_CH1.DX5C	I	P2.3	
USIC1_CH1.DX5D	I	P2.5	
USIC1_CH1.DX5E	I	P2.7	
USIC1_CH1.DX5F	I	P4.4	
USIC1_CH1.DX5G	I	USIC1_CH1.SELO0	
USIC1_CH1.HWIN3	I	USIC1_CH1.DOUT0	HW controlled shift data input
Data Outputs			
USIC1_CH1.DOUT0	O	P0.0; P0.1; P2.12; P2.13; P3.0; USIC1_CH1.DX3G; USIC1_CH1.HWIN3	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 continues...)

Table 117 (continued) USIC1 Channel 1 Interconnects

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	–	Master clock output (optional for all protocols)
USIC1_CH1.SCLKOUT	O	P0.1; P0.3; P2.12; P3.0; USIC1_CH1.DX4G	Shift clock output; used for: • Master SCLKOUT in SSC and IIS • IIC SCL • Can be ignored in ASC
Control Outputs			
USIC1_CH1.SELO0	O	P0.4; USIC1_CH1.DX5G	Shift control output; used for: • SSC Master SELO • IIS WA • Can be ignored for all other protocols
USIC1_CH1.SELO1	O	–	
USIC1_CH1.SELO2	O	P4.1	
USIC1_CH1.SELO3	O	P4.2	
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_CH0.DX1F	Selected DX0 input signal
USIC1_CH1.DX1INS	O	not connected	Selected DX1 input signal
USIC1_CH1.DX2INS	O	CCU41.IN0BC	Selected DX2 input signal
USIC1_CH1.DX3INS	O	USIC1_CH0.DX0G	Selected DX3 input signal
USIC1_CH1.DX4INS	O	USIC1_CH0.DX1G	Selected DX4 input signal
USIC1_CH1.DX5INS	O	USIC1_CH0.DX2G; USIC1_CH0.DX3F; USIC1_CH0.DX4F	Selected DX5 input signal

Figure 137 shows a graphical representation of the USIC1 Channel 1 interconnects.

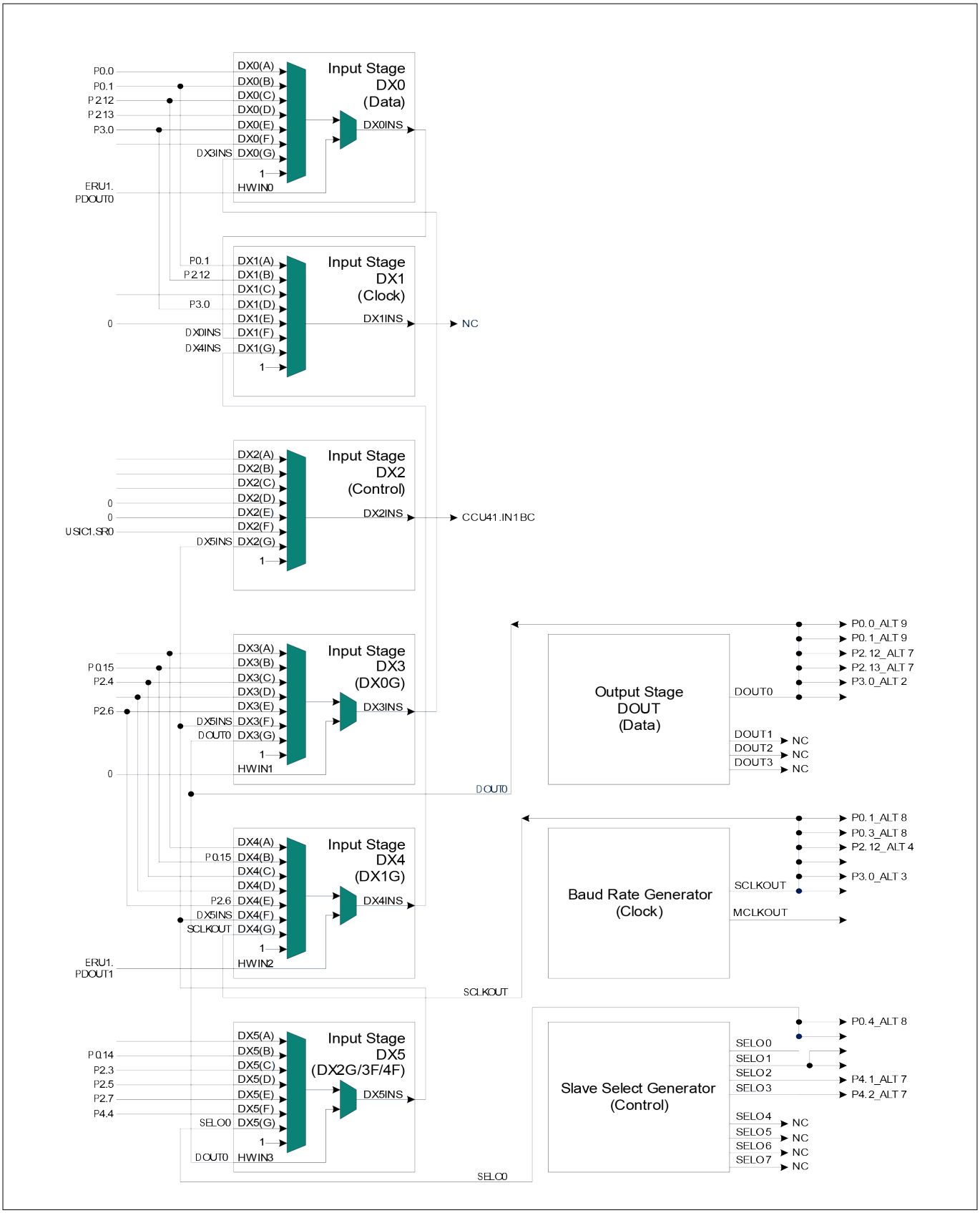


Figure 137 USIC1 Channel 1 Interconnects

16.12.2.3 USIC1 Global Interconnects

Table 118 shows the global interconnects for USIC1 module.

Table 118 USIC1 Global Interconnects

Input/Output	I/O	Connected To	Description
USIC1_SR0	O	NVIC; USIC1_CH1.DX2F	Service request Output 0
USIC1_SR1	O	NVIC	Service request Output 1
USIC1_SR2	O	NVIC	Service request Output 2
USIC1_SR3	O	NVIC	Service request Output 3
USIC1_SR4	O	NVIC	Service request Output 4
USIC1_SR5	O	NVIC	Service request Output 5

17 Controller Area Network Controller (MultiCAN+)

This chapter describes the MultiCAN+ controller of the PSC1Mx. It contains the following sections:

- CAN basics (see CAN Basics)
- Overview of the CAN Module in the PSC1Mx (see Overview)
- Functional description of the MultiCAN+ Kernel (see MultiCAN+ Kernel Functional Description)
- MultiCAN+ Kernel register description (see MultiCAN+ Kernel Registers)
- PSC1Mx implementation-specific details (port connections and control, interrupt control, address decoding, clock control, see MultiCAN+ Module Implementation).

Table 119 Fixed Module Constants

Constant	Description
n_objects	**Number of Message Objects available.**
n_interrupts	**Number of Interrupt Output Lines available.**
n_pendings **n_pendingregs**	**Number of Message Pending Bits available.** There are n_pendings/32 message pending registers.
n_lists	**Number of Lists available for allocation of Message Objects.**
n_nodes	**Number of CAN Nodes available** As each CAN node has its own list in addition to the list of unallocated elements, the relation n_nodes < n_lists is true.

17.1 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).

17.1.1 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 extended the message identifier lengths to 29 bits, i.e. the extended identifier. Both frame formats are part of the ISO 11898-1. The identifier is available for Classical CAN.

17.1.2 CAN Frame Types

There are three types of CAN frames:

- Data Frames
- Remote Frames
- Error Frames

A Data Frame for Classical CAN 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 (for example, 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.

17.1.2.1 Data Frames

There are two types of Data Frames defined (see Figure 138):

- 11bit ID Data Frame
- 29bit ID Data Frame

11-bit Data Frame (Classical CAN Format)

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 for Classical CAN) 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.

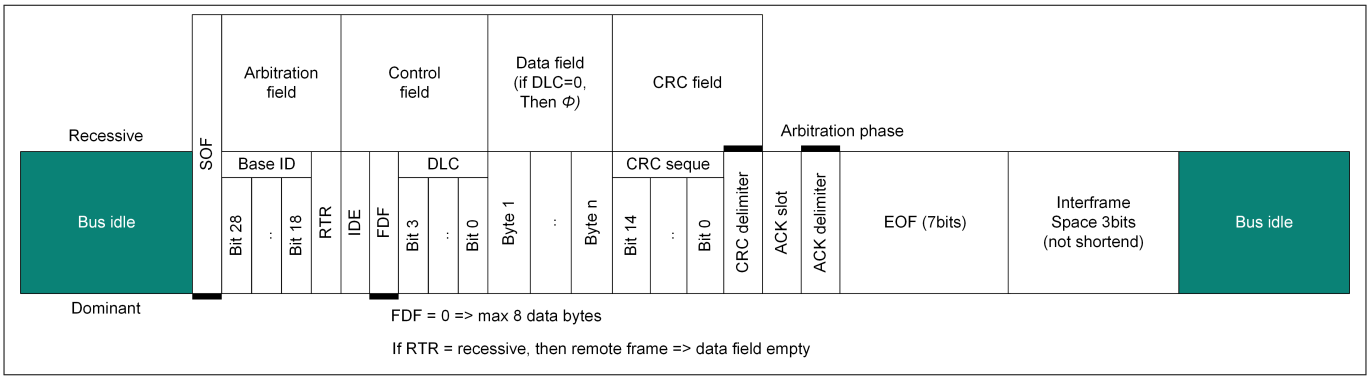


Figure 138 Classical11bit ID CAN Data Frame

Extended Data Frame (Classical CAN Format)

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 classical 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 classical CAN frame, then the 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 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 Classical Data Frame.

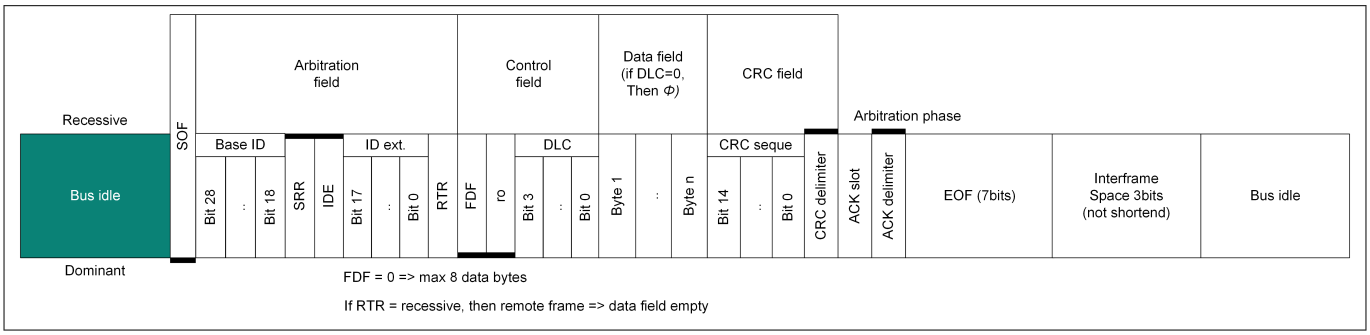


Figure 139 29 bit ID CAN Data Frame

17.1.2.2 Remote Frames

Normally, data transmission is performed on an autonomous basis with the data source node (for example, 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.

17.1.2.3 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.

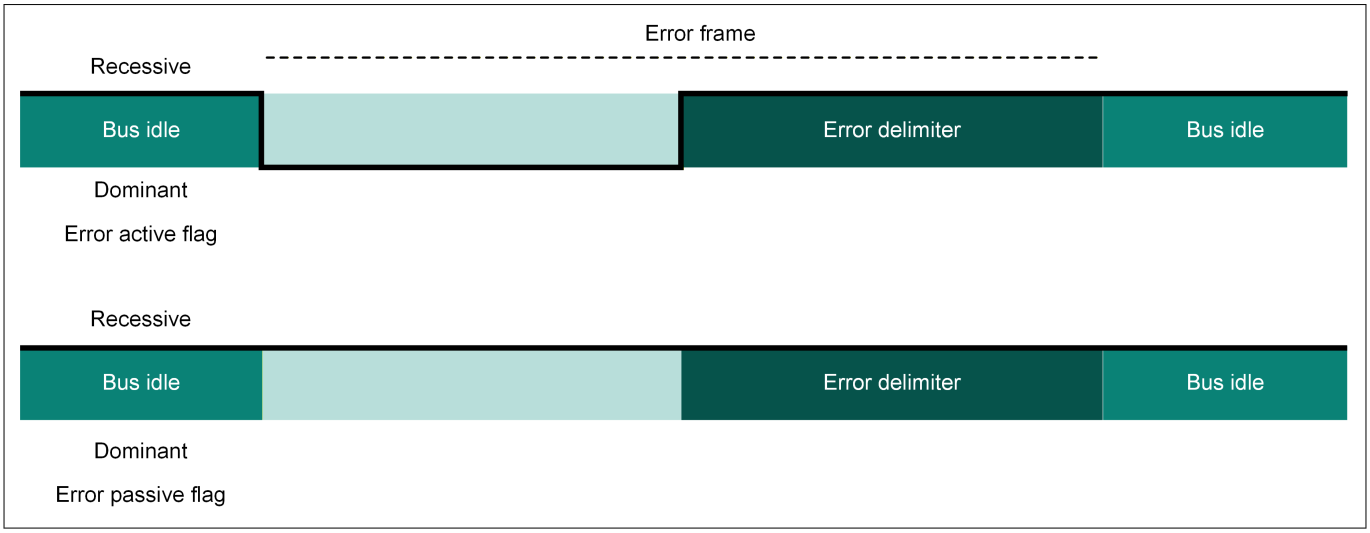


Figure 140 CAN Error Frames

17.1.3 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 141.

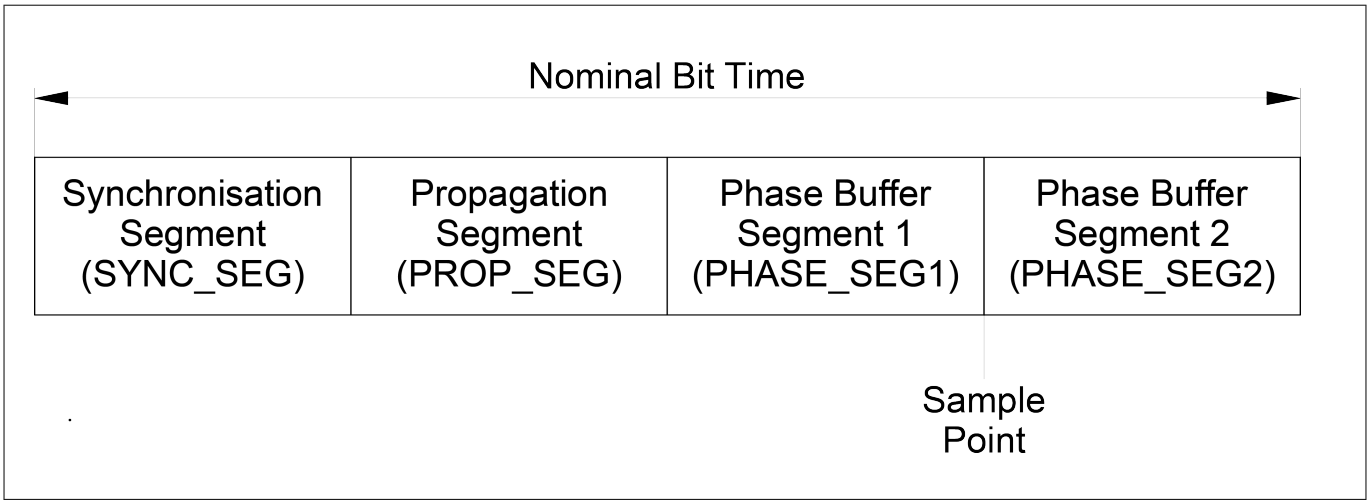


Figure 141 Partition of Nominal Bit Time

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.

17.1.4 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 CRC, 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.

Full-CAN devices (this is the case for the MultiCAN+ controller as implemented in PSC1Mx) 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 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.

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.

17.2 Overview

The MultiCAN+ module provides a communication interface which is fully compliant with CAN specification V2.0B (active), providing communications at up to 1 Mbit/s in Classical CAN (ISO 11898-1:2003(E) mode).

The MultiCAN+ module for the PSC1Mx consists of 1 module (i.e MultiCAN with 2 CAN nodes), representing 2 serial communication interfaces. Each CAN node communicates over two pins (TXD and RXD). The device ports which are used for TXD and RXD may be individually configured within the PORTS block. Several port configuration options are available to provide application-specific flexibility.

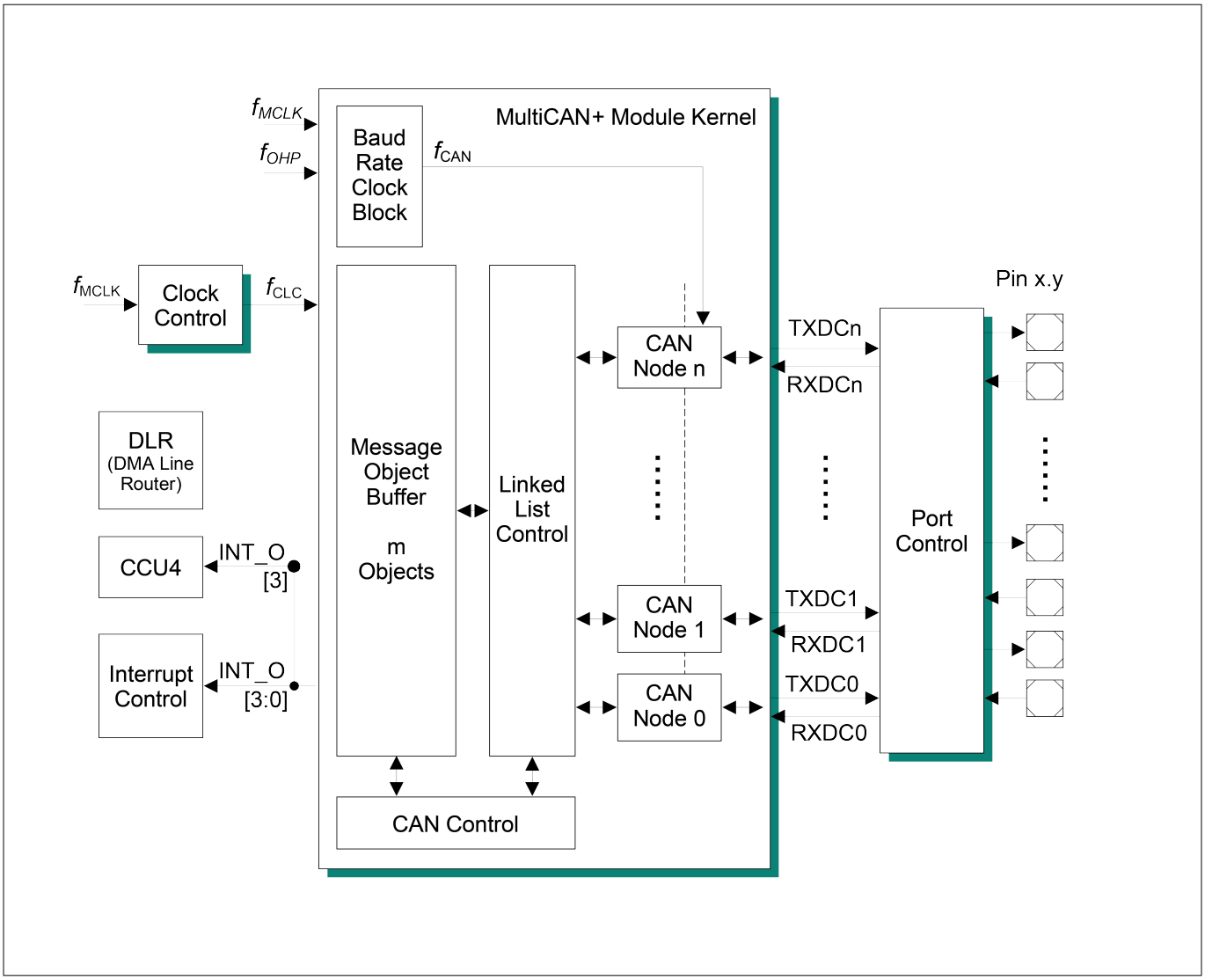


Figure 142 Overview of the MultiCAN+ Module. The module has 2 nodes and 32 objects. PSC1Mx

The MultiCAN+ module contains 2 independently operating CAN nodes with Full-CAN functionality that are able to exchange Data and Remote Frames via a gateway function. 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 32 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.

17.2.1 Features List

The MultiCAN+ module provides the following features:

- Compliant with ISO 11898 and SAE J 1939

- 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
- Support for asynchronous clock sources for baud rate generation by providing separate frequency domain and input:
  - System frequency clock f_{CLC}
  - Low-jitter E-Ray PLL clock
  - Direct oscillator clock (for example, from ceramic resonator)
- Frequency jitter calibration based on external CAN messages during runtime
- 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 32 message objects can be individually
  - Allocated (assigned) to any CAN node
  - Configured as transmit or receive object
  - Setup 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 different 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 on its order in the list
- Advanced CAN node features
  - Analyzer Mode supports monitoring of bus traffic without actively participating on the bus
  - Internal Loop-Back Mode is available for test purposes
  - Data transmission from a node can be stopped without affecting reception
  - Programmable minimum delay between two consecutive messages
- 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 2 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
- Advanced interrupt handling
  - Message interrupts, node interrupts can be generated

- Interrupt requests can be routed individually to one of the 8 interrupt output lines
- Message post-processing notifications can be combined flexibly into a dedicated register field of 256 notification bits

17.3 MultiCAN+ Kernel Functional Description

This section describes the functionality of the MultiCAN+ module.

17.3.1 Module Structure

Figure 143 shows the general structure of the MultiCAN+ module.

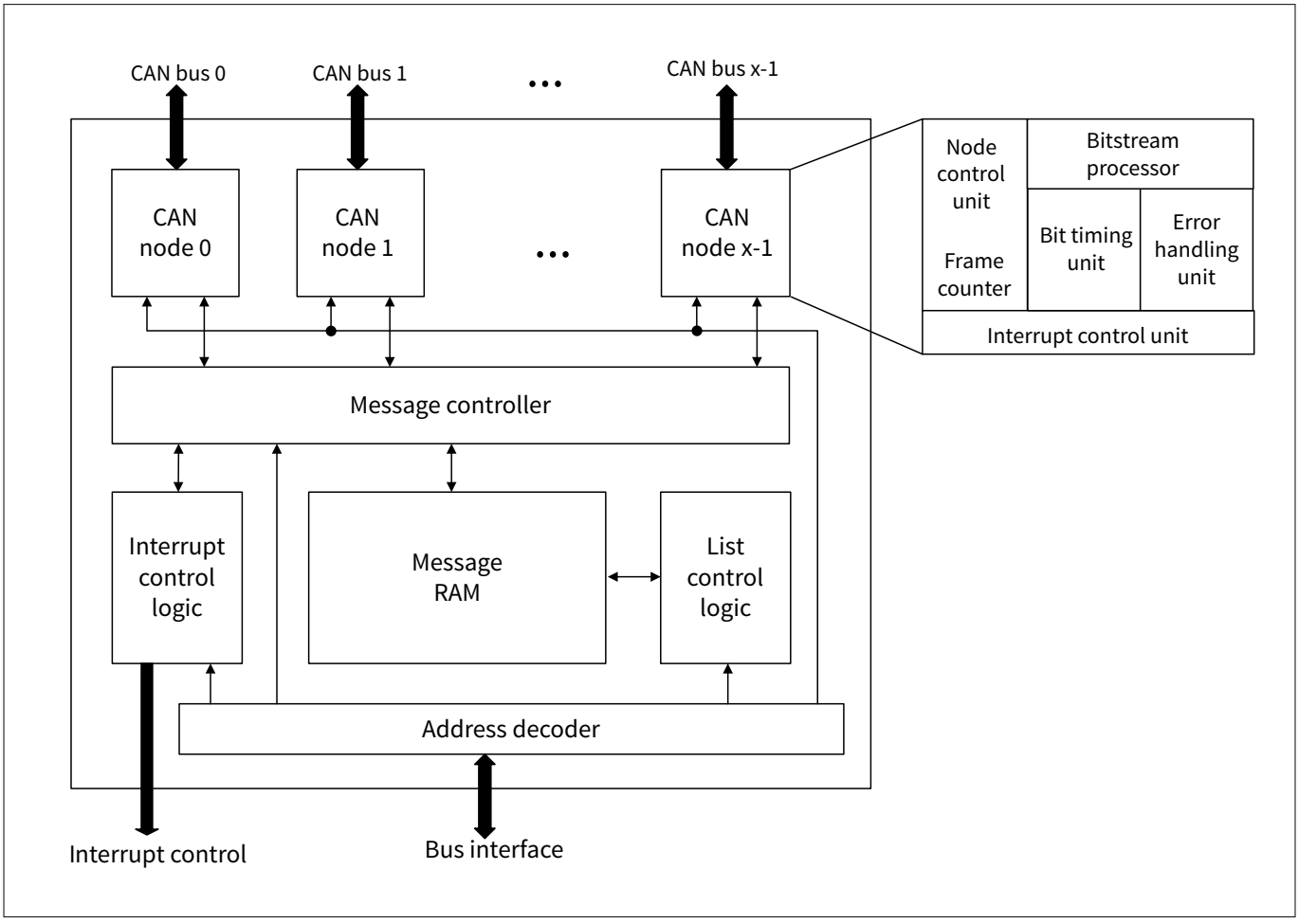


Figure 143 MultiCAN+ Block Diagram

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 144. 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 8 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.

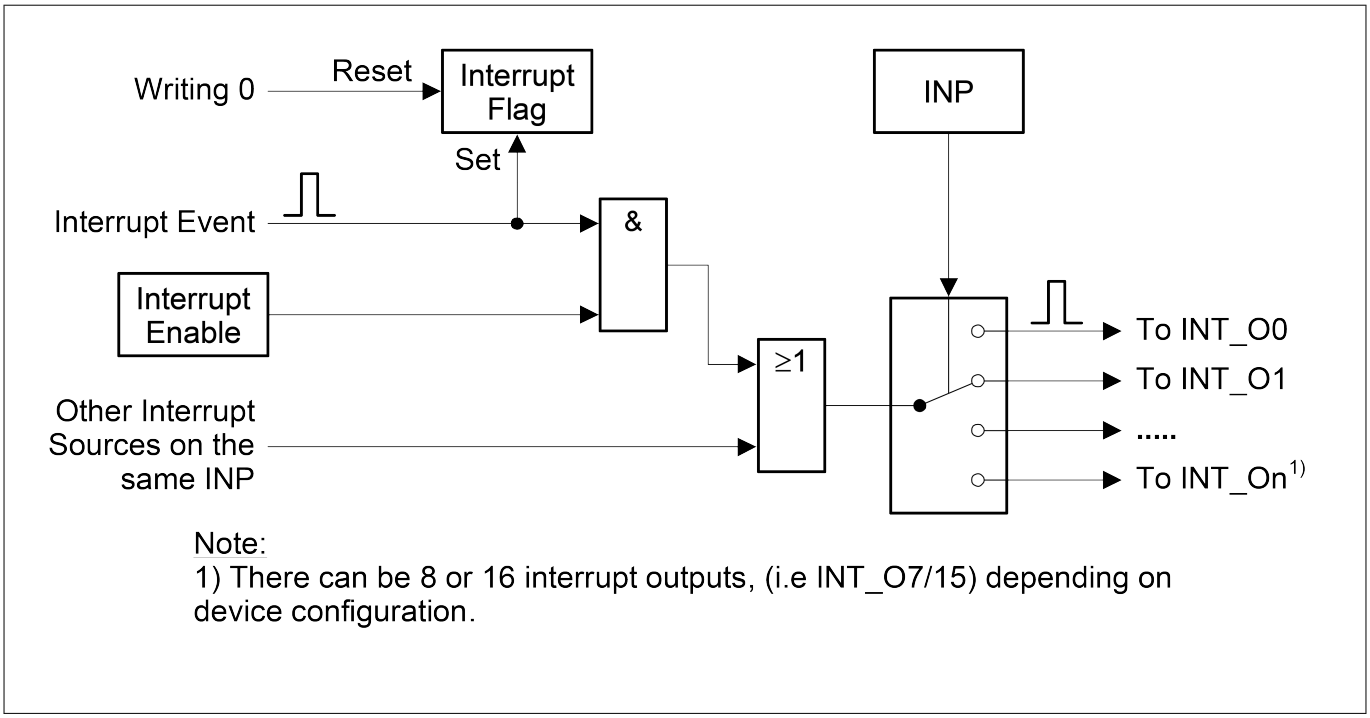


Figure 144 General Interrupt Structure

17.3.2 Clock Control

The CAN module timer clock f_{CAN} of the functional blocks of the MultiCAN+ module is derived from the synchronous clock source. 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 operates with the module control clock f_{CLC}. See also Figure 145.

The output clock f_{CAN} of the Fractional Divider is based on the clock f_A, but only every nth clock pulse is taken.

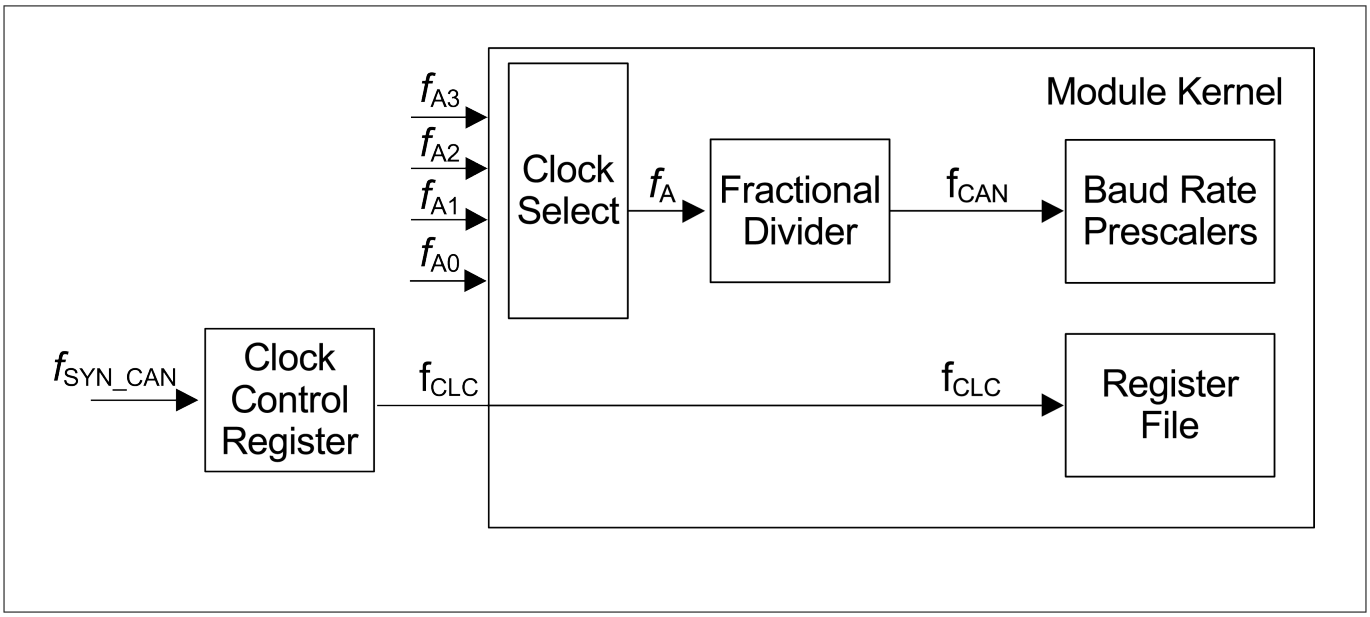


Figure 145 MultiCAN+ Clock Generation

The f_{SYN_CAN} is identical to f_{MCLK}. f_{Ai} is the asynchronous clock input.

Table 120 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 (for example, for a maximum of 500 kbit/s, 50% of the indicated value are required).

The values imply that the CPU executes maximum accesses to the MultiCAN+ module. The values may contain rounding effects.

Table 120 Minimum Operating Frequencies[1] [MHz]

Number of allocated message objects MO[2]	Number of Active CAN Nodes				
Number of Active CAN Nodes	**1**	**2**	**3**	**4**	**5**
16 MO	12	19	26	33	40
32 MO	15	23	30	37	44
64 MO	21	28	37	46	53
128 MO	40	45	50	55	61
256 MO	72	77	82	88	93

1) In the case of 15 time quanta, the minimum operating frequency required is 15 MHz.
2) 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_A, this frequency has to be chosen carefully to allow correct CAN bit timing. The required value of f_A 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_A are given by formula [n × 20] MHz, with n being an integer value, starting at 1.

It is not advised to use fractional divider mode.

Additionally, for correct operation of the MultiCAN, the following conditions have to be fulfilled,

$$\text{Baudrate}_{max} = [(8 \times T_{CAN}) + (8 \times T_{CLC}) + (4 \times \text{No. of active CAN nodes} \times T_{CLC})] \quad (33)$$

also

NBTR.SJW < NBTR.TSEG1

As an example, when f_{CLC} = 10 MHz, f_{CAN} = 20 MHz, No of active CAN nodes = 2, Baudrate_{max}=[(8 x 50 ns) + (8 x 100 ns) + (4 x 2 x 100 ns)] = 2000 ns = 500 kBaud

Table 121 below illustrates the minimum CAN module timer clock f_{CAN} and Module Control Clock f_{CLC} that's required to support a baudrate generation of 500 kBaud. If a higher baudrate is desired, the values need to be calculated as per Equation (33).

Table 121 Minimum Operating Frequencies [MHz] required for 500kBaud

No. of active CAN nodes	$f_{CAN} = f_{CLC}$ (MHz)	$f_{CAN} \neq f_{CLC}$ (MHz)	
		f_{CAN}	f_{CLC}
1	10	16	8
		20	8
		24	8
		80	7
2	12	16	11
		20	10
		24	10
		80	9
3	14	16	14

(table continues...)

Table 121 (continued) Minimum Operating Frequencies [MHz] required for 500kBaud

No. of active CAN nodes	$f_{CAN} = f_{CLC}$ (MHz)	$f_{CAN} \neq f_{CLC}$ (MHz)	
		f_{CAN}	f_{CLC}
		20	13
		24	12
		80	11

17.3.3 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 Node Receive Input Selection.

17.3.4 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

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 valid (including ACK) received frame is stored in the corresponding message object, and can be evaluated by the CPU to determine their identifier, IDE 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 alert 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.

Setting bit TXDIS in register NCRx stops the transmit activity of this node without affecting reception; bit CANDIS disables the node completely.

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.

17.3.4.1 Bit Timing Unit

According to the ISO 11898 standard, a CAN bit time is subdivided into different segments (Figure 146). 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 in **Node x Bit Timing Register**). The baud rate prescaler is driven by the module timer clock f_{CAN} (generation and control of f_{CAN} is described in Module Clock Generation).

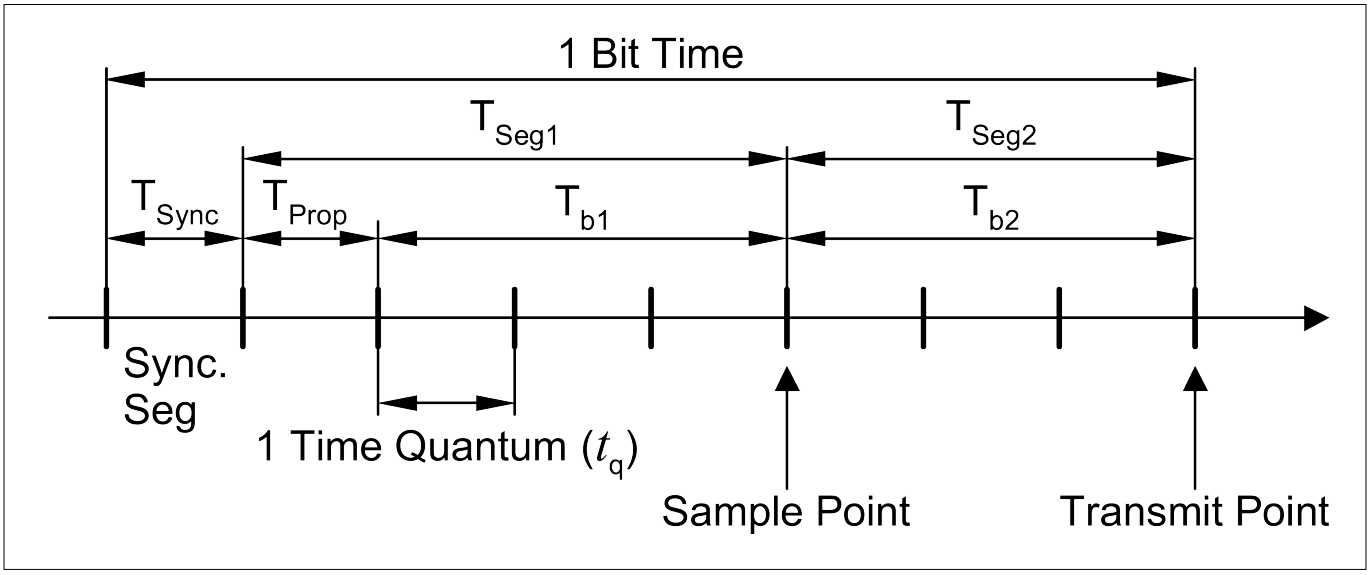


Figure 146 CAN Bus Bit Timing Standard

The Synchronization Segment (T_{Sync}) allows 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
$= 8 \times (BRP + 1) / f_{CAN}$ if DIV8 = 1
$T_{Sync} = 1 \times t_q$
$T_{Seg1} = (TSEG1 + 1) \times t_q$ (min. 3 t_q)
$T_{Seg2} = (TSEG2 + 1) \times t_q$ (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. 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.

$T_{SJW} = (SJW + 1) \times t_q$

$T_{Seg1} \geq T_{SJW} + T_{prop}$

$T_{Seg2} \geq TSJW$

The maximum relative tolerance for f_{CAN} depends on the Phase Buffer Segments, re-synchronization jump width and the bit time.

$df_{CAN} \leq \min(T_{b1}, T_{b2}) / [2 \times (13 \times \text{bit time} - T_{b2})]$ AND

$df_{CAN} \leq T_{SJW} / 20 \times \text{bit time}$

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.

17.3.4.2 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 LEC 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 LEC 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.

17.3.4.3 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 **Node x Error Counter Register**) 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 **Node x Control Register**).

17.3.4.4 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 **Node x Frame Counter Register**). 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 (internally) with the beginning of a new bit time. Its value is permanently sampled in the CFC field while the bus is idle. The value sampled just before the SOF bit of a new frame is detected is written to the corresponding message object. When the treatment of a message object is finished, the sampling continues.
- **Bit Timing Mode:**

  Used for baud rate detection and analysis of the bit timing (Bit Timing Analysis).
- **Error Count Mode:**

  The frame counter is incremented when an error frame is received or an error is detected by the node (001_B to 110_B) (see Table 127 for **Encoding of the LEC Bit field**). If the NFCRx.CFCIE interrupt bit is enabled, the NFCRx.CFCOV overflow flag will be set when the frame counter overflows. Configuration of CFSEL has no influence.

17.3.4.5 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 Interrupt Control for further processing of the CAN node interrupts.

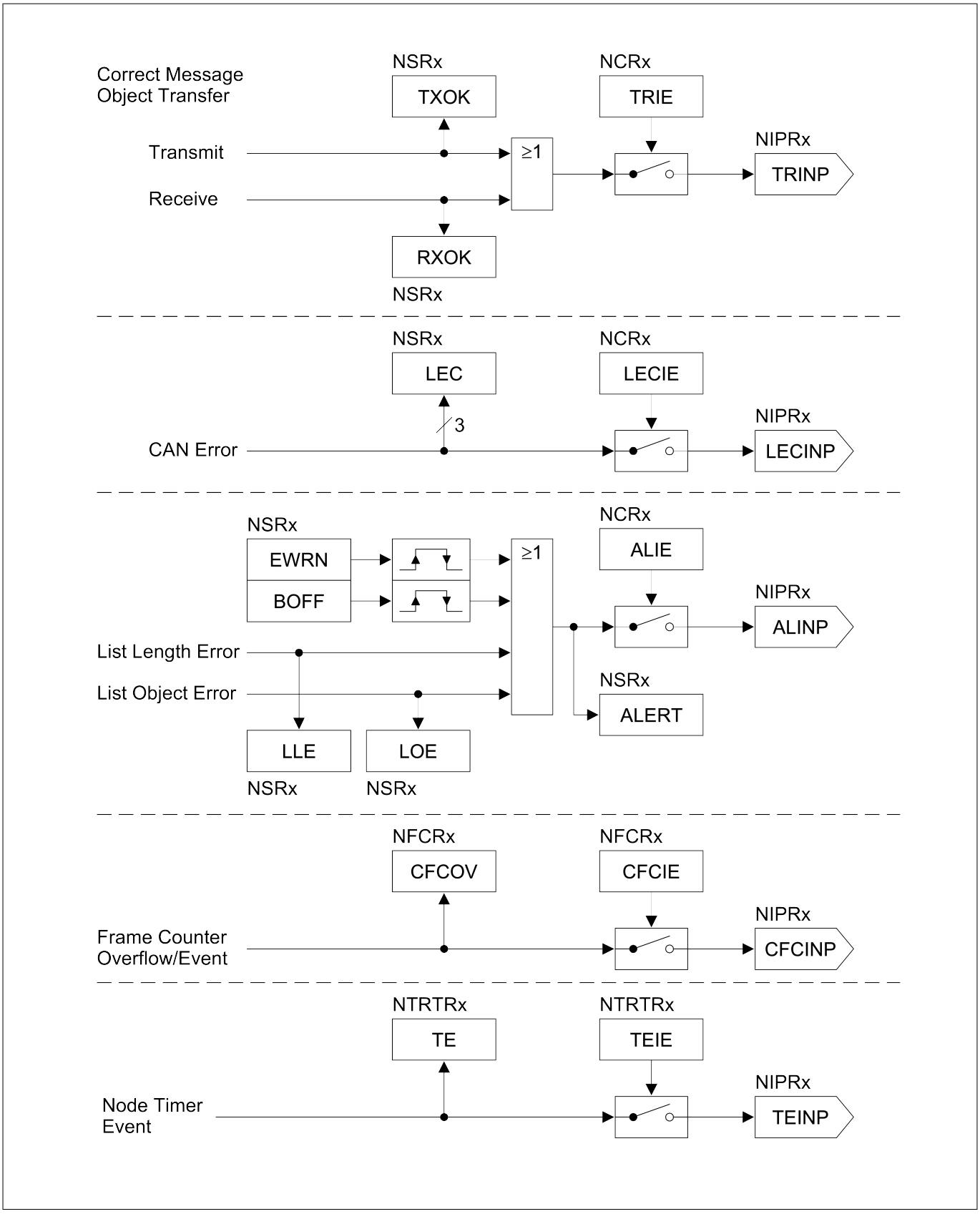


Figure 147 CAN Node Interrupts

17.3.5 Message Object List Structure

This section describes the structure of the message object lists in the MultiCAN+ module.

17.3.5.1 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 16 lists. Each message object is allocated to one of these lists. In the example in Figure 148, the three message objects (3, 5, and 16) are allocated to the list with index 2 (List Register LIST2).

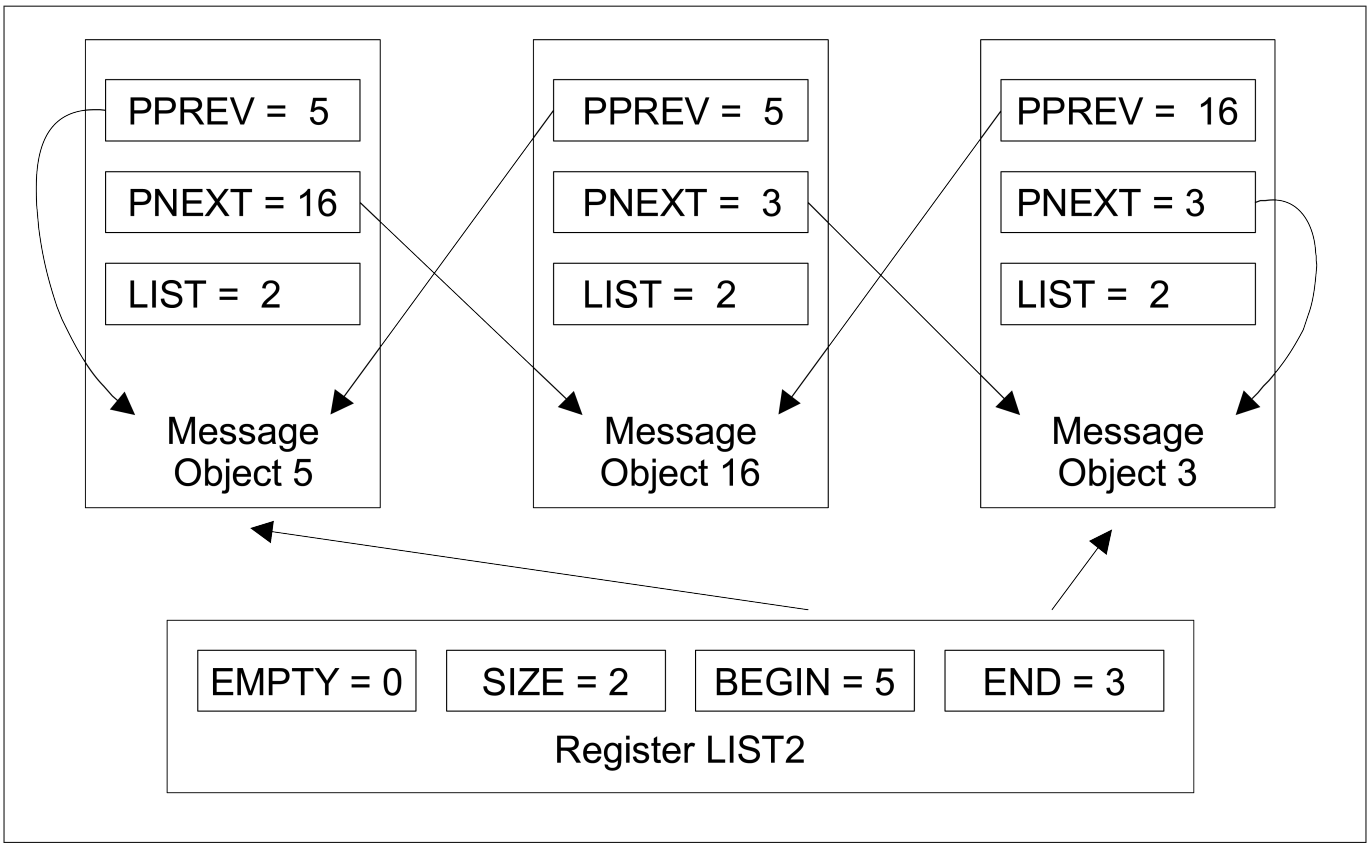


Figure 148 Example Allocation of Message Objects to a List

Bit field BEGIN in the List Register (for definition, see **LIST0**) 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 Status Register MOSTATn (see **CAN_MOSTAT0**) 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 MOSTATn.LIST indicates the list index number to which the message object is currently allocated to. The message objects 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.

17.3.5.2 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 (MOSTATn.LIST = 0). It is called allocated if it belongs to a list with an index not equal to 0 (MOSTATn.LIST > 0).

After reset, all message objects are unallocated. This means that they are assigned to the list of unallocated elements with MOSTATn.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).

17.3.5.3 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 149. 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 (16) 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.

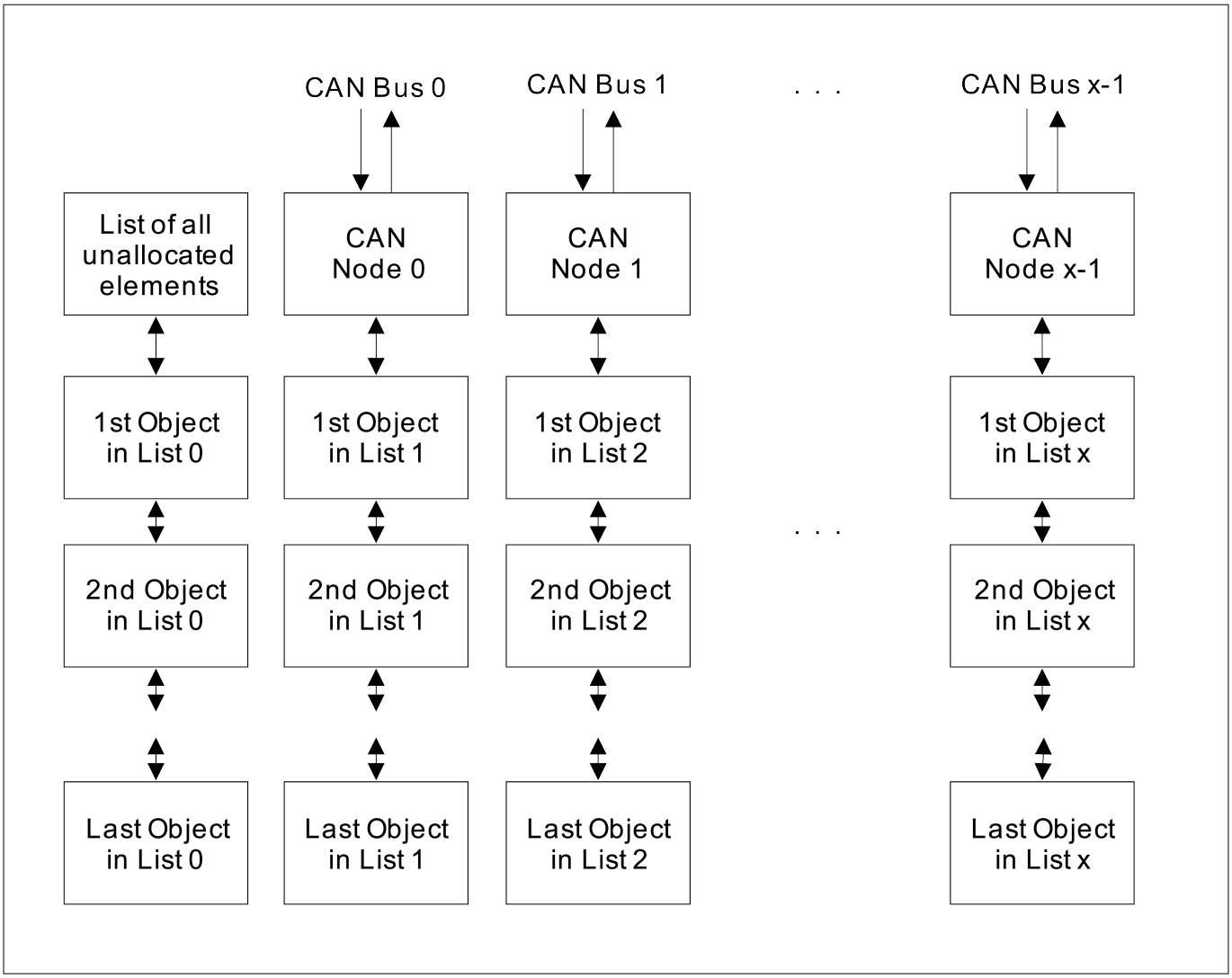


Figure 149 Message Objects Linked to CAN Nodes

17.3.5.4 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 Status 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 122 gives an overview on the available panel commands while Table 126 describes the panel commands in more detail.

Table 122 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 **Panel Control Register**). 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 and resetting the CLC.DISR (see **CAN Clock Control Register**) register, the list controller builds up list 0. Afterwards the BUSY bit is set, dependent on core speed this might be visible. During list controller initialization, 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 enabling the clocks of the module, by the list controller in order to ensure correct list pointers in each message object. The operation is indicated by automatically setting the BUSY bit. The end of this CAN RAM initialization is indicated by bit PANCTR.BUSY becoming inactive. It is advised to initialize some registers within the CAN controller before polling the PANCTR.BUSY the first time.*

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 MOSTATn.MSGVAL (“Message Valid”) of message object n must be cleared.

17.3.6 CAN Node Analyzer Mode

The chapter describes the CAN node analyzer capabilities of the MultiCAN+ module.

17.3.6.1 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.

17.3.6.2 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 150). 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.

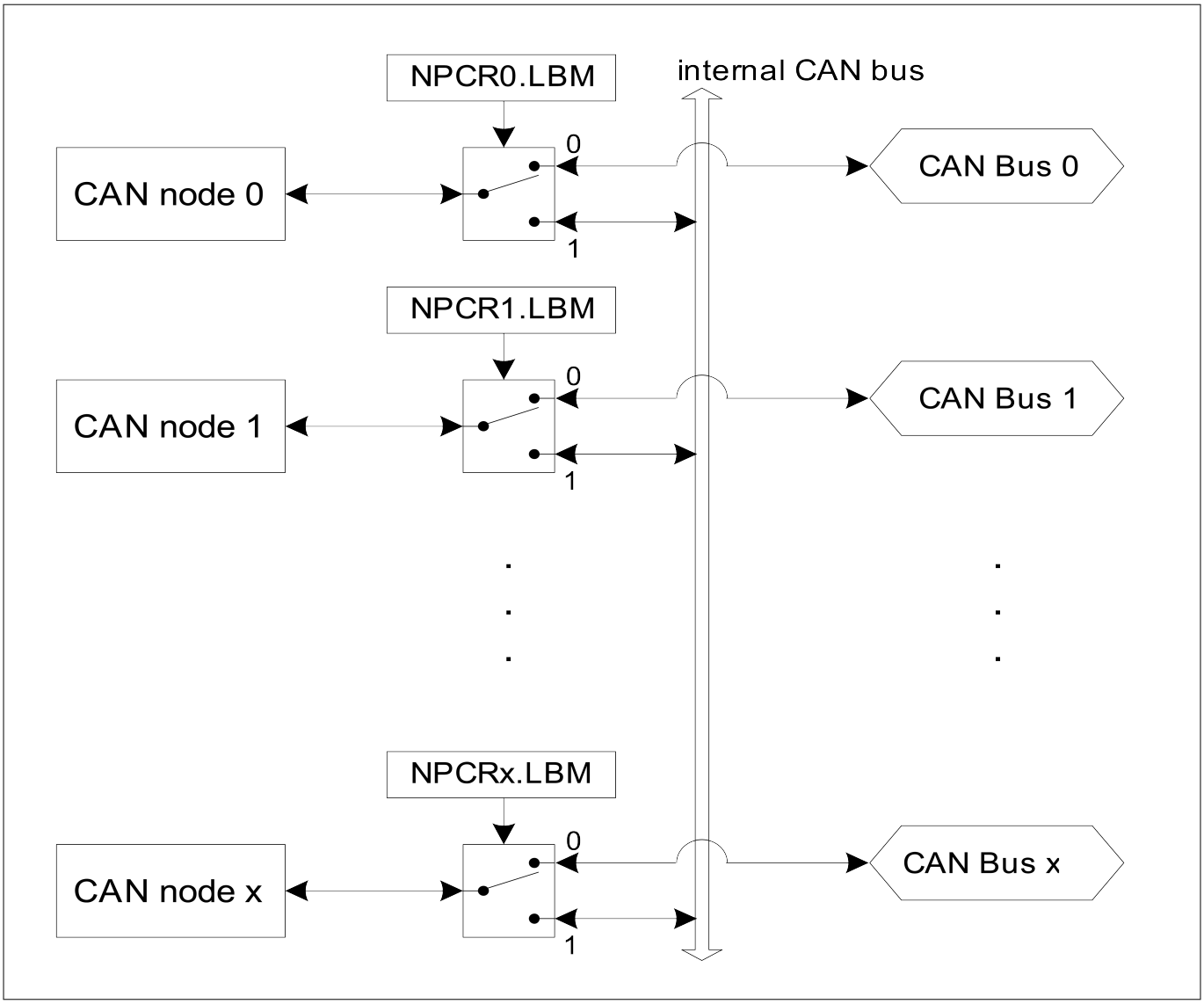


Figure 150 Loop-Back Mode

17.3.6.3 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. The value of NFCRx.CFC is valid one module cycle later when NFCRx.CFCOV is set. If NFCRx.CFCIE is set, an interrupt request can be generated (see Figure 147).

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_{CLC} 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.

17.3.7 Message Acceptance Filtering

The chapter describes the Message Acceptance Filtering capabilities of the MultiCAN+ module.

17.3.7.1 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 Object Status Register (see **Message Object n Status Register**) 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 151 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 **Message Object n Arbitration Register**):

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.

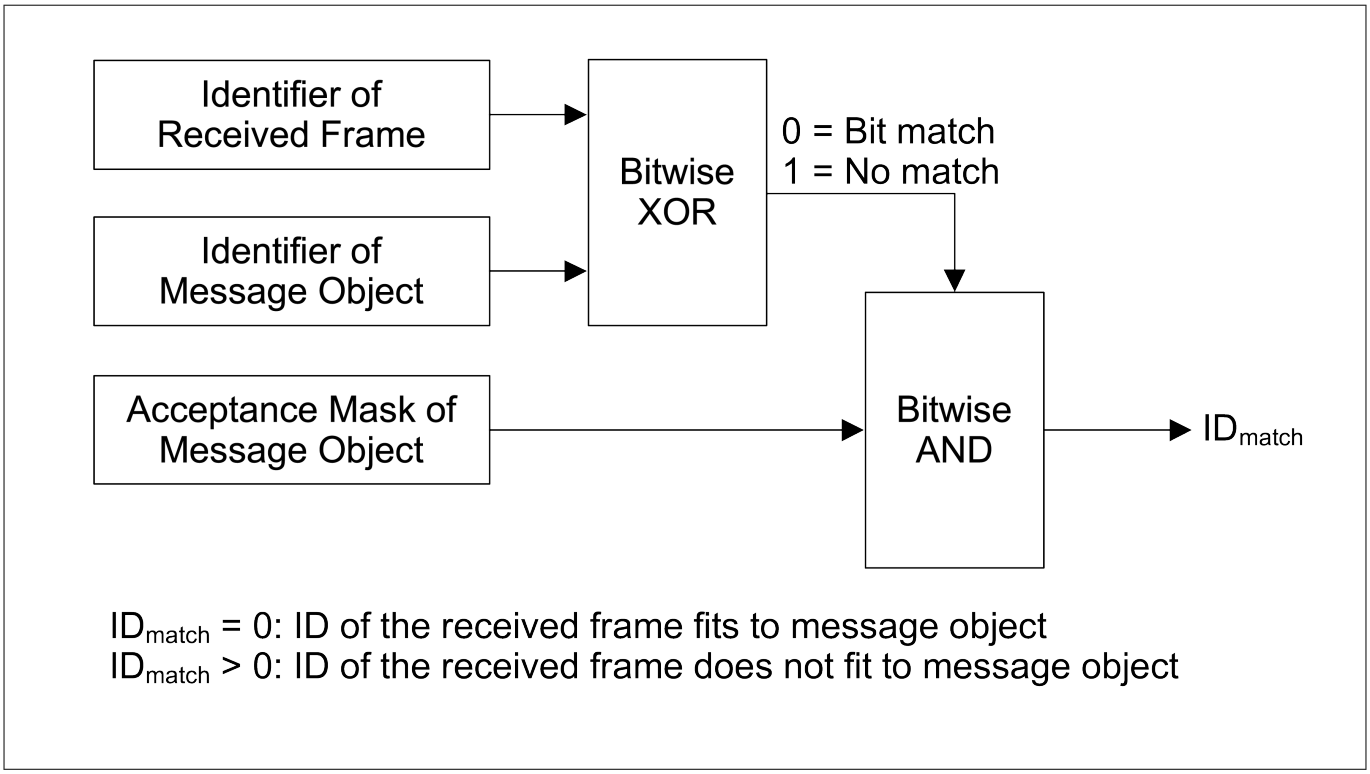


Figure 151 Received Message Identifier Acceptance Check

17.3.7.2 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 152).

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 MOSTATn.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 132).
- PRI = 01_B or PRI = 11_B (priority by list order).
- PRI = 00_B is reserved and makes the message object to have no function.

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.

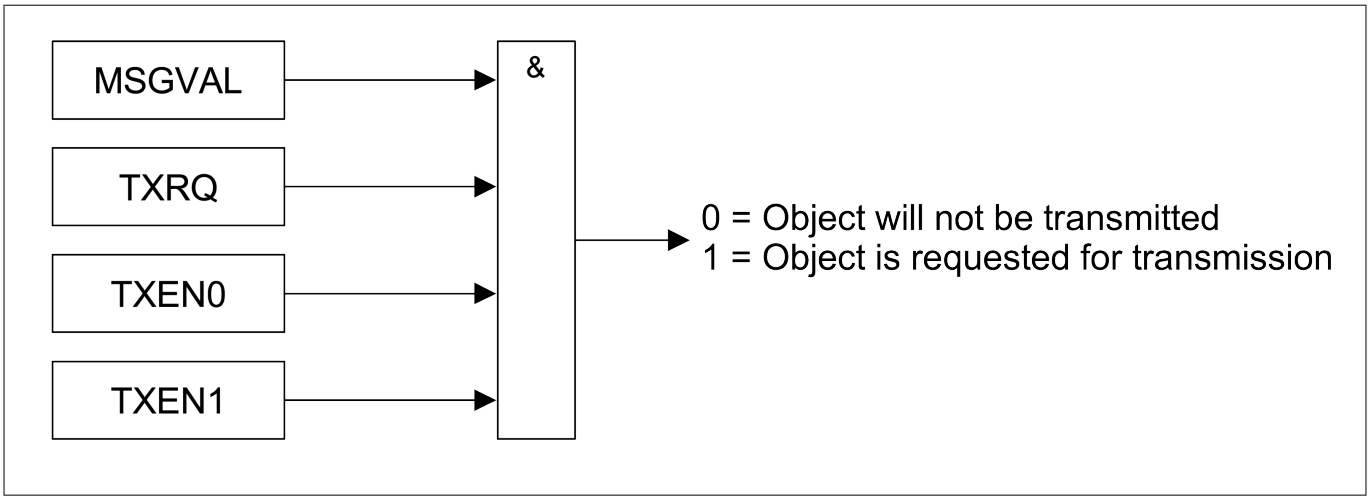


Figure 152 Effective Transmit Request of Message Object

17.3.8 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.

17.3.8.1 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 153). 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 Interrupt Control for further processing of the message object interrupts.

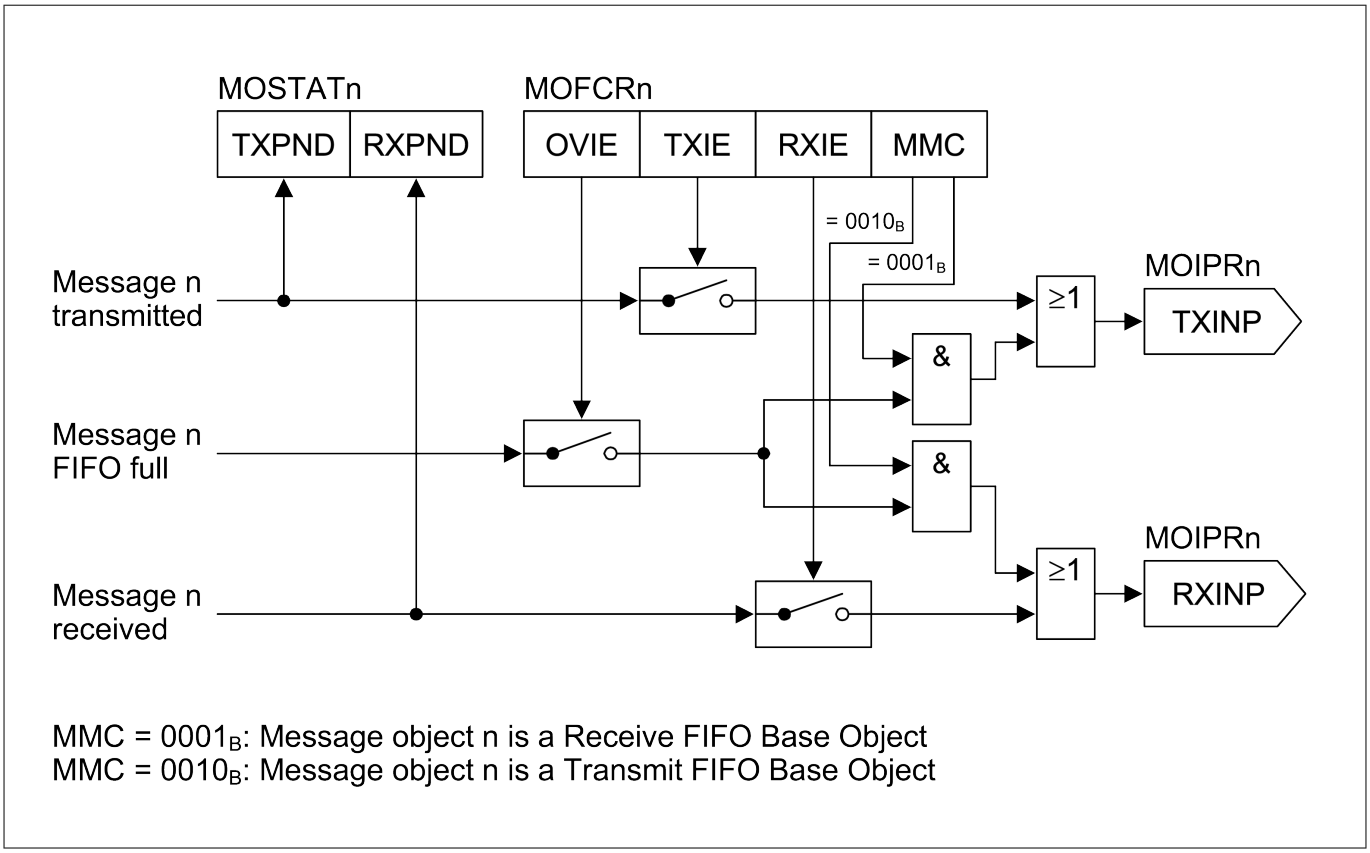


Figure 153 Message Interrupt Request Routing

17.3.8.2 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 154 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.

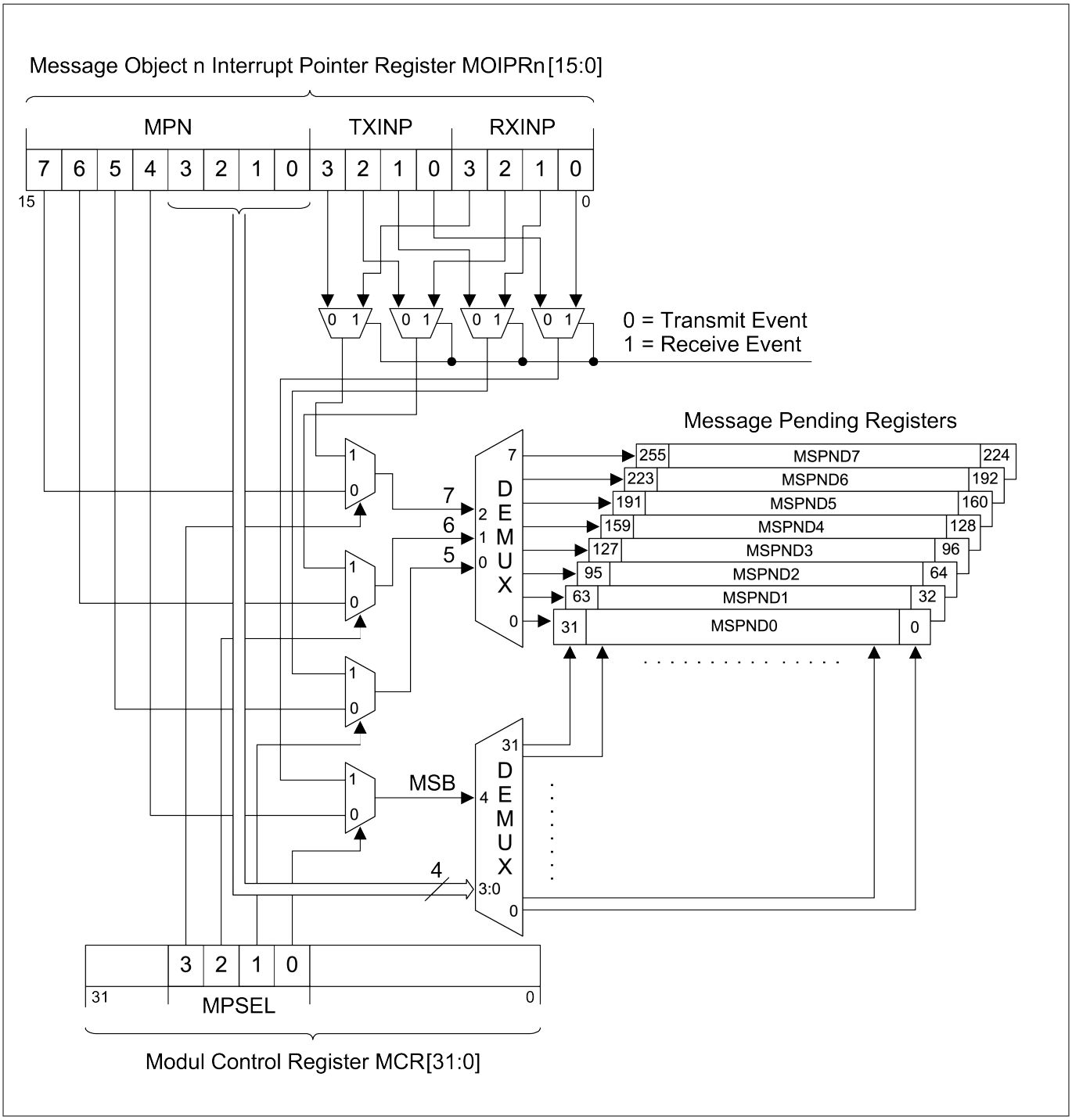


Figure 154 Message Pending Bit Allocation

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 **Module Control Register**). 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 **Message Index Register k**) 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.

17.3.9 Message Object Data Handling

This chapter describes the handling capabilities for the Message Object Data of the MultiCAN+ module.

17.3.9.1 Frame Reception

After the reception of a message, it is stored in a message object according to the scheme shown in Figure 155. 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 MOSTATn.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 is 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.

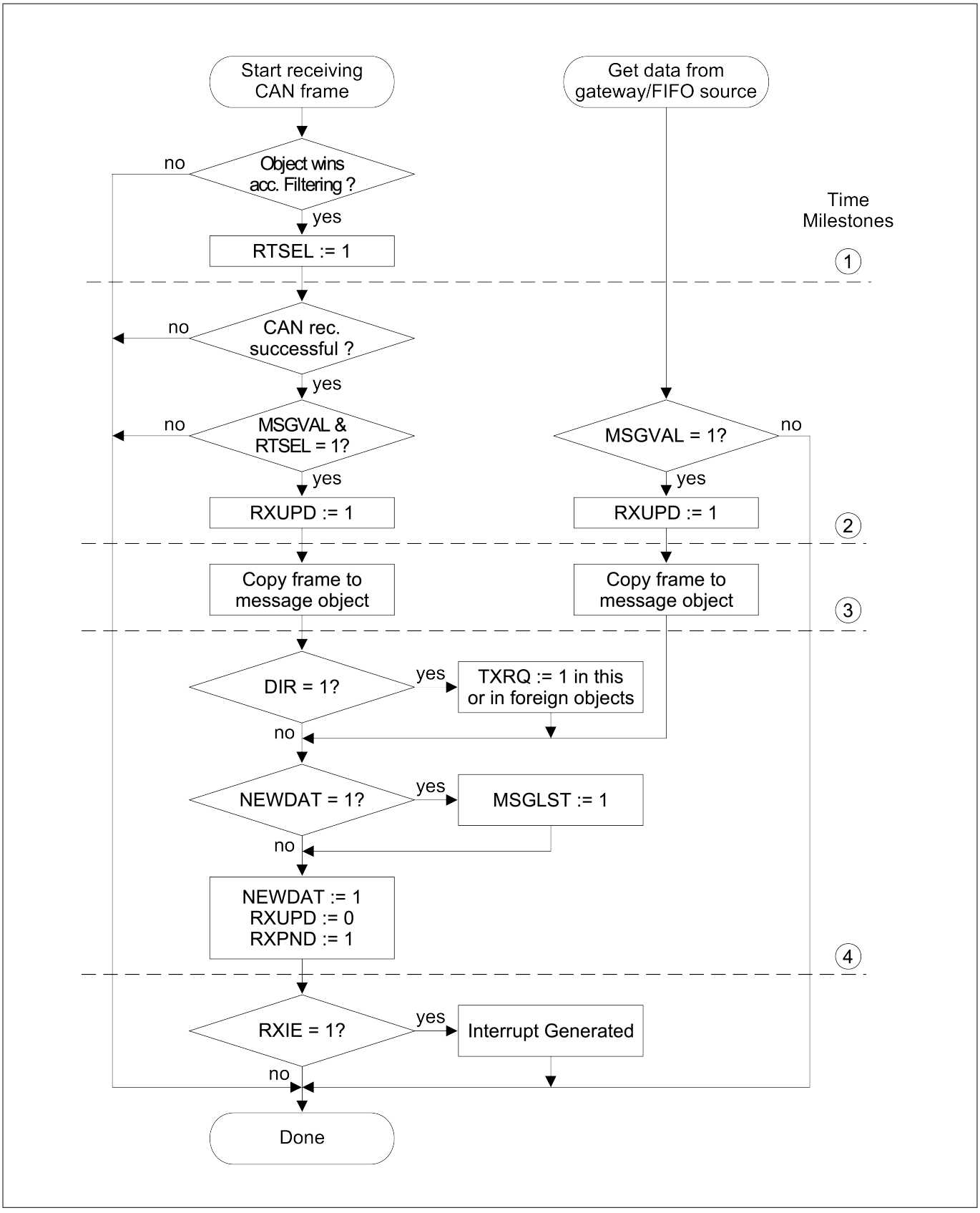


Figure 155 Reception of a Message Object

17.3.9.2 Frame Transmission

The process of a message object transmission is shown in Figure 156. 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 152. Although these bits are equivalent with respect to the transmission process, they have different semantics:

Table 123 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).
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 MOSTATn.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.

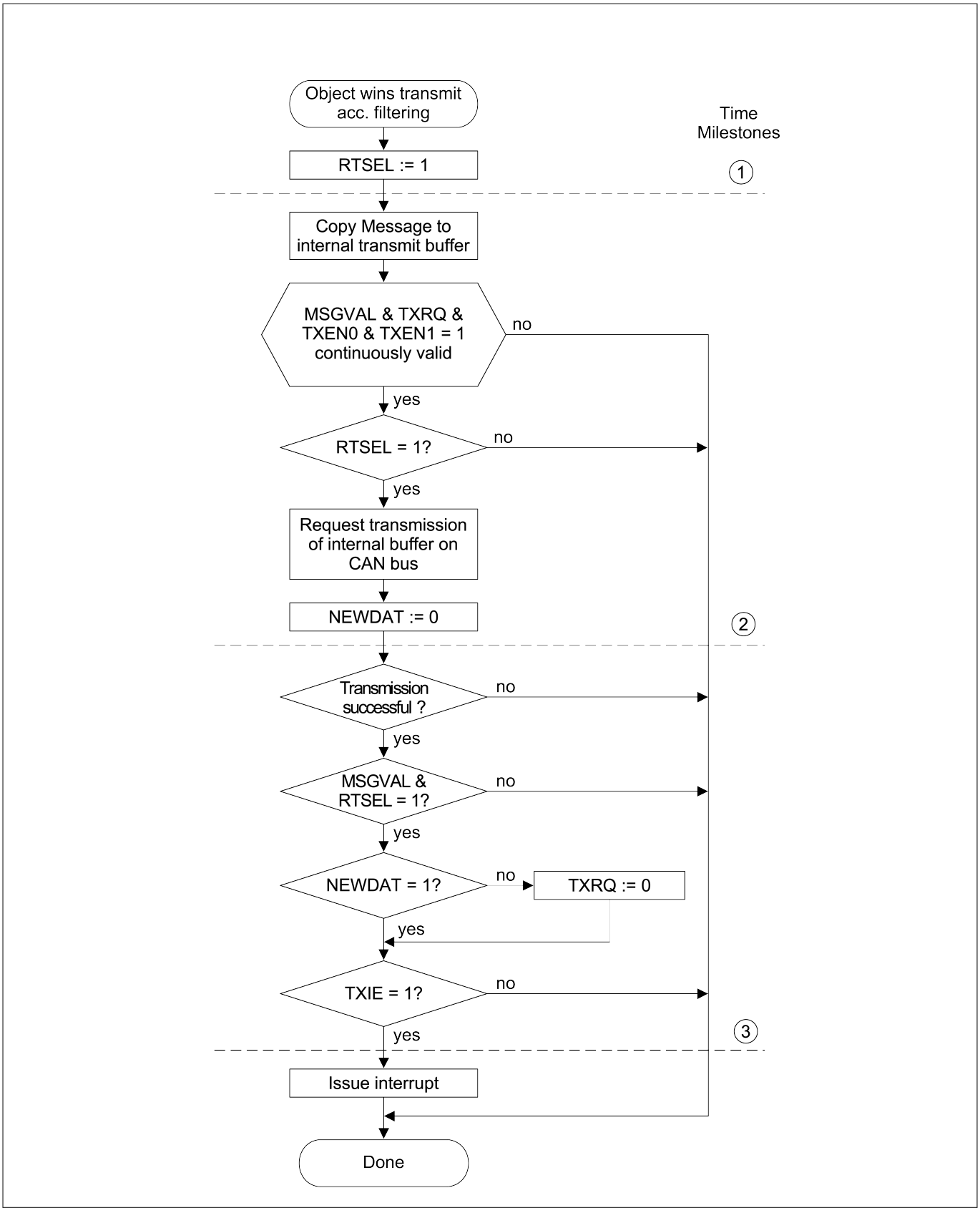


Figure 156 Transmission of a Message Object

17.3.10 Message Object Functionality

This chapter describes the functionality of the Message Objects in the MultiCAN+ module.

17.3.10.1 Standard Message Object

A message object is selected as standard message object when bit field MOFCRn.MMC = 0000_B (see **Node x Frame Counter Register**). 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.

17.3.10.2 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 or Remote Frame. This prevents the reception of further messages.

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 or Remote Frame.

17.3.10.3 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.

17.3.10.4 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 157. 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 157 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.

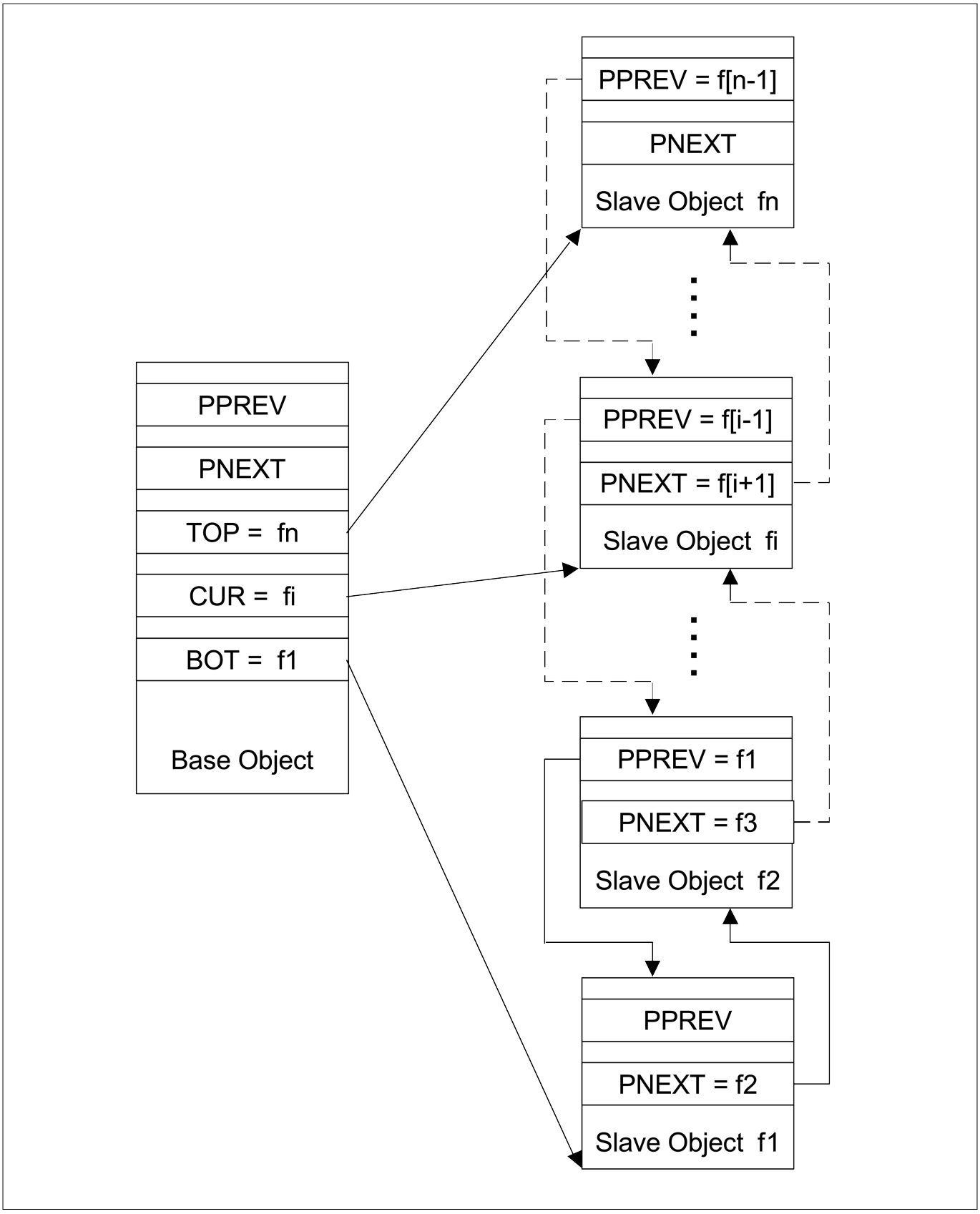


Figure 157 FIFO Structure with FIFO Base Object and n FIFO Slave Objects

17.3.10.5 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 FIFO base and slave object only if MSGVAL = 1.

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.

17.3.10.6 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 Status 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.

17.3.10.7 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 MOFGPRs.CUR=d 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 158):

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 in Message Object FIFO Structure. 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 157, 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).

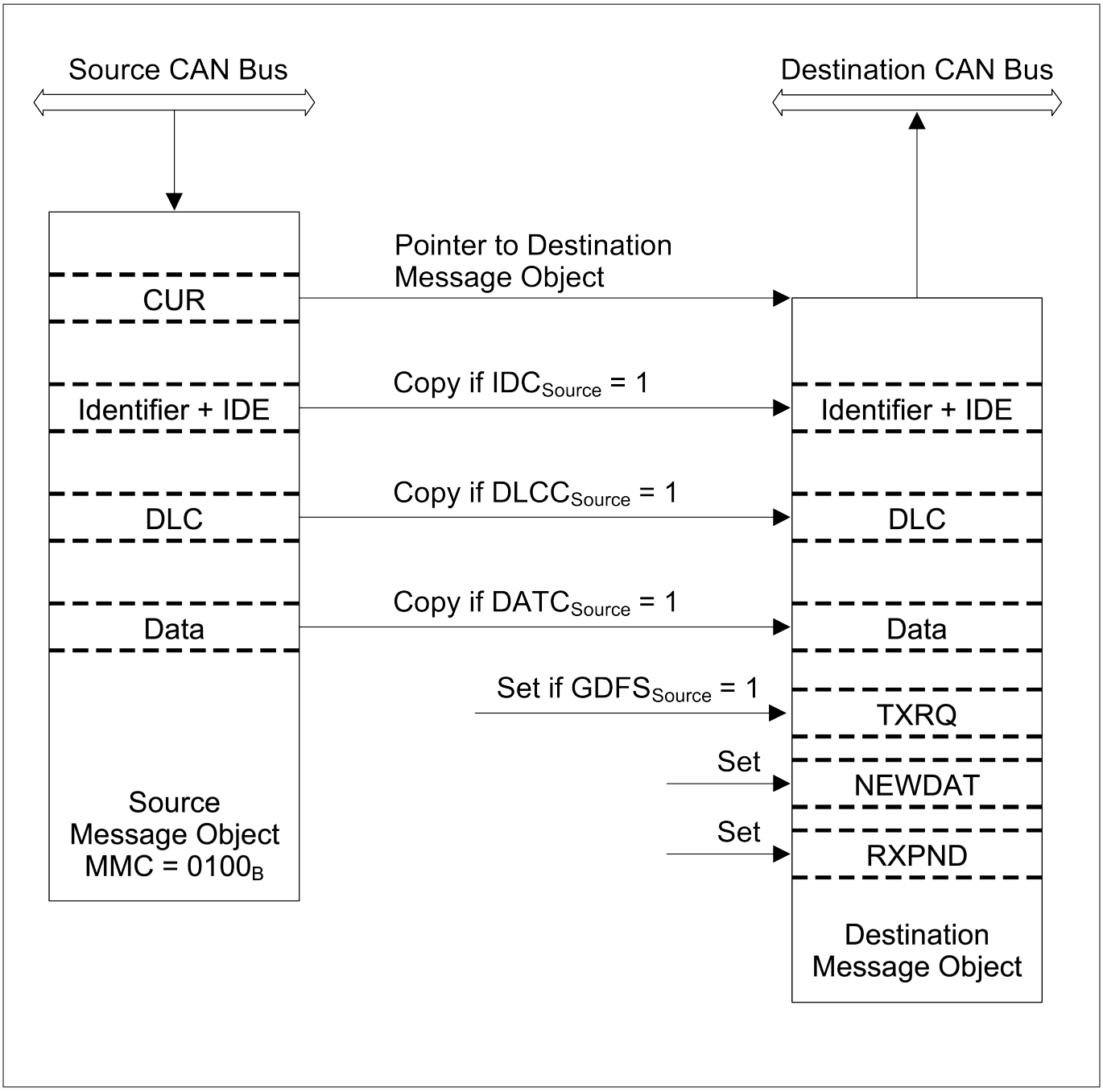


Figure 158 Gateway Transfer from Source to Destination

17.3.10.8 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.

17.4 Use Case Example MultiCAN+

This section explains the core functionality of the MultiCAN+ module with a code example. The example realizes sending a CAN message via internal CAN-bus from node 0 to node 1.

The MultiCAN+ module for the PSC1Mx consists of module (i.e MultiCAN with 2 CAN nodes), representing serial communication interfaces. Each CAN node can either be connected to a port or to the internal CAN bus (see Figure 161). So a maximum of 2 CAN channels can be realized with the MultiCAN+ module (For more information and further functions see Overview).

All CAN nodes share a common set of 32 message objects. A message object function like a container for a specific CAN message; both transmitting and receiving of CAN messages can be realized. The message objects are organized in double-chained lists, each list is dedicated to a certain CAN node (list 1 is node 0, list 2 is node 1, etc.). After a reset, all message objects are unallocated and therefore assigned to list 0, this list does not belong to a CAN node. During initialization it is possible to allocate the message objects individually to certain CAN node lists. Each allocated message object can be set up with all necessary information like the message ID etc. (see chapter 26.2 and following for further explanations of the CAN module).

In the use case example CAN node 0 will send a CAN message via internal CAN bus to CAN node 1. The transmitting message object (MO) will be MO 0 and the receiving message object will be MO 1. As soon as the CAN frame successfully receives at MO 1 a receive interrupt will occur. The initialization process follows the following order:

1. Load global MuliCAN+ module registers
2. Initialize the CAN nodes
3. Allocate the message objects to the CAN nodes
4. Initialize the message objects
5. Start the CAN nodes
6. Start transmit request for CAN message from node 0 to node 1.
7. Receive message at node 1, Rx interrupt occurs.

Step description to initialize the MultiCAN+ module:

(Line 2) enable control of the module, in clock control register CLC (**CLC**).

(Line 3) read back (dummy variable has to be defined). The reading process ensures that the write process from line 2 is done.

(Line 4) load fractional divider to $f_{CAN} = f_{MCLK}$ (see also Clock Control (Fractional Divider), f_{MCLK} see **MCR**).

(Line 5) read back (dummy variable to be defined).

(Line 6) set clk source to f_{MCLK} in module control register MCR (**MCR**).

***Note**: The reconfiguration of the clk source must be done by two writes and a certain delay between these. See MCR for more details.*

(Line 7) The setting of protection CCE[6] and INIT[0] activates the initialization and configuration mode for CAN node 0. This is necessary for the upcoming changes in CAN node 0. (see also node configuration register NCRn (**CAN_NCRx (x = 0-1)**))

(Line 8) This example uses the internal loop-back mode, so this line connects CAN node 0 to the internal CAN Bus. (see also Figure 150) and node port control register NPCR (**CAN_NPCRx (x = 0-1)**))

(Line 9) The CAN bit time is subdivided into different segments (see also Bit Timing Unit). Assuming f_{CAN} = 100 MHz this line then sets the baudrate to 500 kBaud and the sample point to 80% in the node bit timing register NBTR (**(CAN_NBTRx (x = 0-1)**).

(Line 10 - 12) same as line 7 - 9 but with node 1.

(Line 13) The allocation of the message objects to certain CAN node lists function via a list command panel. The panel control register PANCTR (**PANCTR**) can only be written if both busy flags are reset. Therefore this while loop waits until the register is ready.

(Line 14) This line allocates message object 0 to list 1 (belongs to CAN node 0).

(Line 15) see line 13.

(Line 16) This line allocates message object 1 to list 2 (belongs to CAN node 0).

(Line 17) Initialize MO 0 in the message object register MOCTR (**CAN_MOCTRz (z = 0-30)**) as transmit MO by setting bits [27:25]. Set also MSGVAL[21], that's the main switch bit of the MO.

(Line 18) this line sets the message data length of MO 0 to 8 bytes.(see also message object function control register MOFCR (**CAN_MOFCRn (n = 0-31)**).

(Line 19) the message ID of MO 0 gets set to standard message (11 bit length) with a message ID of FF_H in the Message object arbitration register MOAR (**(CAN_MOARn (n = 0-31)**).

(Line 20) Initialize MO 1 in the message object register MOCTR as receive MO by setting bit RXEN[23]. Set also MSGVAL[21], that's the main switch bit of the MO.

(Line 21) the receive interrupt gets enabled in the message object function control register MOFCR.

(Line 22) same as Line 19 but for MO 1. The same message ID is necessary to receive CAN messages from the bus with this specific ID.

(Line 23) the Rx interrupt gets connected to the CAN interrupt output line 1.(see also MOIPR (**CAN_MOIPRn (n = 0-31)**))

(Line 24) This line enables the Rx interrupt in the service request control register SRC_CANINT1 and sets the interrupt priority to CAN_SR1INT (1...255)

(Line 25) This function sets the interrupt priority of the receive interrupt.

(Line 26) The 4 lower data bytes are getting loaded in the MODATAL (**(CAN_MODATALn (n = 0-31)**) register. the data itself is here just an example.

(Line 27) The 4 higher data bytes are getting loaded in the MODATAH (**CAN_MODATAHn (n = 0-31)**) register. (just an data example as well)

(Line 28) This line resets INIT[0] and CCE[6] bit in the node control register from node 0. The node is now synchronizing itself to the bus.

(Line 29) same as line 28, but with node 1.

(Line 30) The NEWDATA bit gets set in the MOCTR register, this should always be done after a write process in the data registers from line 26/27. The transmit request bit TXRQ gets set, this starts the transmission of the CAN message. The RTSEL gets reset to ensure the transmission.

(Line 31) If the message is received the MOCTR.NEWDAT[19] bit in the receive MO 1 is set. This line just waits for the reception. The occurrence of the Rx interrupt can also be used as a proof that the message arrived.

(Line 32/33) Access the data bytes without clearing NEWDAT.

***Note**: The Rx ISR function prototype would be: void CAN1_Rx_irq (void); here could be your ISR code;*

***Note**: Line 1 does not apply for Arm® products, therefore this line does not exist.*

Initialization of the MultiCAN+ module:

```
// Load global MultiCAN+ registers:
(2)   CAN_CLC = 0x000;          // enable module control
(3)   dummy = CAN_CLC;          //read to check the made changes
(4)   CAN_FDR = (1 << 14)| 0x3FF;       //DM=1,STEP=1023

(5)   dummy = CAN_FDR;          //read to check the made changes
(6)   CAN_MCR = 0x1;            // CLKSEL=1
//Init CAN nodes 0 and 1:
(7)   CAN_NCR0 = (1 << 6)| 1;   // CCE=1,INIT=1
(8)   CAN_NPCR0 = (1 << 8);     // LBM=1
(9)   CAN_NBTR0 = 0x3EC9;       //set bit segments
(10)  CAN_NCR1 = (1 << 6)| 1;   // CCE=1,INIT=1
(11)  CAN_NPCR1 = (1 << 8);     // LBM=1
(12)  CAN_NBTR1 = 0x3EC9;       //set bit segments
//Allocate message objects to CAN nodes:
(13)  while(CAN_PANCTR.U & (0x00000100 | 0x00000200)); // busy?
(14)  CAN_PANCTR = (1 << 24)|(0 << 16)| 2; // MO 0 to list 1
(15)  while(CAN_PANCTR.U & (0x00000100 | 0x00000200)); // busy?
(16)  CAN_PANCTR = (2 << 24)|(1 << 16)| 2; // MO 1 to list 2
//Init MO_0 (list 1, node 0)
(17)  CAN_MOCTR0 = (1<<27)|(1<<26)|(1<<25)|(1<<21); // set to Tx
(18)  CAN_MOFCR0 = (8 << 24);               // DLC=8
(19)  CAN_MOAR0 = (1 << 30)|(0xFF << 18);   // PRI=01,ID=0xFF
//Init MO_1 (list 2, node 1)
(20)  CAN_MOCTR1 = (1 << 23)|(1 << 21);    // set to Rx
(21)  CAN_MOFCR1 = (1 << 16);              // RXIE=1
(22)  CAN_MOAR1 = (1 << 30)|(0xFF << 18);     // PRI=01, ID=0xFF
(23)  CAN_MOIPR1 = 0x1;        // RXINP=1 -> select INT_O1
(24)  SRC_CANINT1 = (1 << 10) | CAN_SRN1INT; // enable INT_01
(25)  NVIC_SetPriority (CAN_SRN1INT, & CAN1_Rx_irq);
// Load Data, Start CAN nodes
(26)  CAN_MODATAL0 = 0x0D0D0D0D;  // load data, lower 4 Bytes
(27)  CAN_MODATAH0 = 0x0E0E0E0E; // load data, higher 4 Bytes
(28)  CAN_NCR0 &= ~((1<<6)| 1);  // reset CCE and INIT
(29)  CAN_NCR1 &= ~((1<<6)| 1);  // reset CCE and INIT
// set transmit request to send message
(30)  CAN_MOCTR0 = (1<<24)|(1<<19)|(1<<6);//TXRQ=1,NEWDAT=1
(31)  while((CAN_MOSTAT1.B.NEWDAT) != 1);        //check if Rx
(32)  readfirstfourdatabytes1=CAN_MODATAL1;//read_receive
(33)  readnextfourdatabytes1=CAN_MODATAH1; //read_receive
```

17.5 MultiCAN+ Kernel Registers

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 PSC1Mx 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

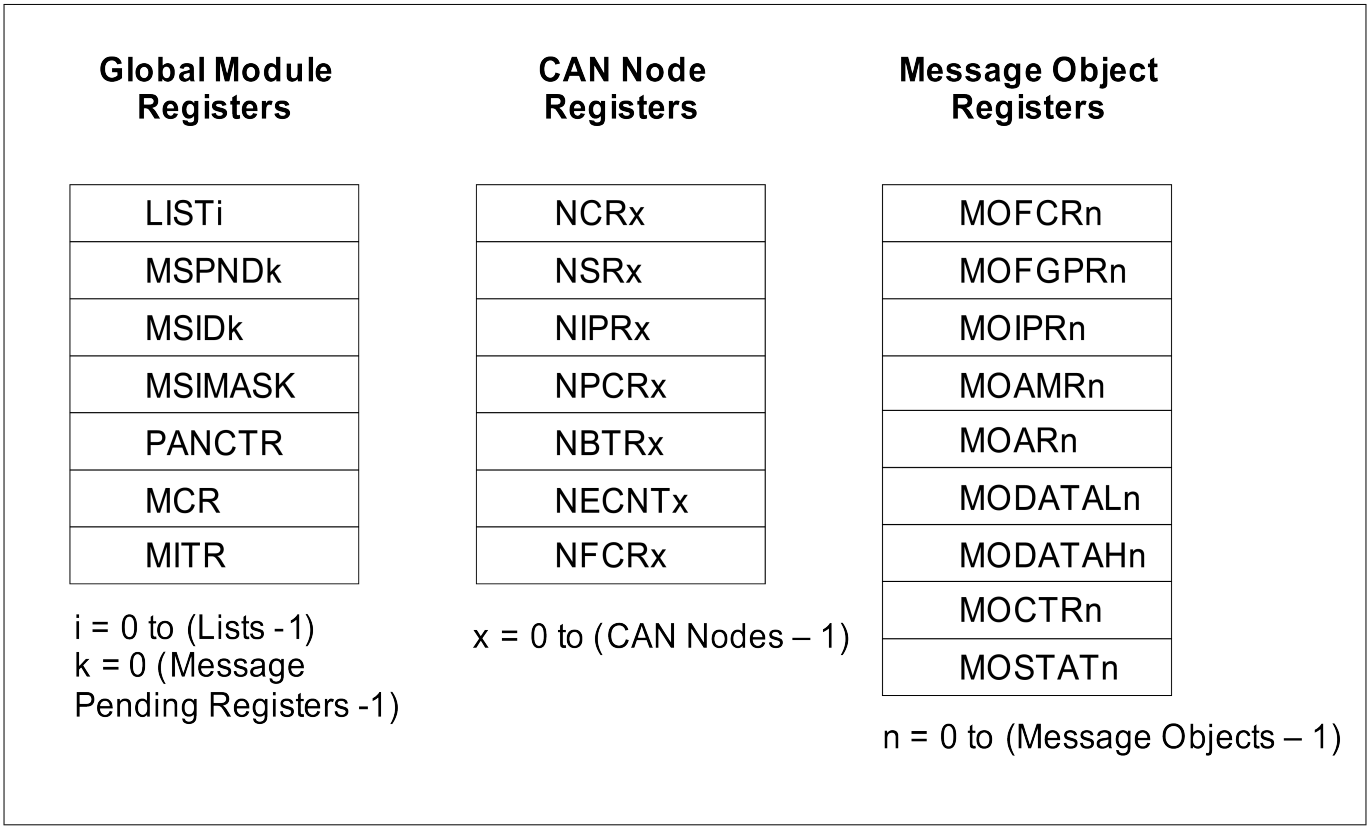


Figure 159 MultiCAN+ Kernel Registers

The registers of the MultiCAN+ module kernel are listed below.

Table 124 Registers Address Space - MultiCAN+ Kernel Registers

Module	Base Address	End Address	Note
CAN	$5004\ 0000_H$	$5004\ 3FFF_H$	-

Table 125 Registers Overview - MultiCAN+ Kernel Registers

Register Name	Description	Offset Addr[1]	Access Mode[2]		Reset
			Read	Write	
Global Module Registers					
LISTi	List Register i	$0100_H + i \times 4_H$	U, PV	U, PV	Application Reset
MSPNDk	Message Pending Register k	$0140_H + k \times 4_H$	U, PV	U, PV	Application Reset
MSIDk	Message Index Register k	$0180_H + k \times 4_H$	U, PV	U, PV	Application Reset
MSIMASK	Message Index Mask Register	$01C0_H$	U, PV	U, PV	Application Reset
PANCTR	Panel Control Register	$01C4_H$	U, PV	U, PV	Application Reset
MCR	Module Control Register	$01C8_H$	U, PV	U, PV	Application Reset
MITR	Module Interrupt Trigger Reg.	$01CC_H$	U, PV	U, PV	Application Reset
CAN Node Registers					
NCRx	Node x Control Register	$0200_H + x \times 100_H$	U, PV	U, PV	Application Reset
NSRx	Node x Status Register	$0204_H + x \times 100_H$	U, PV	U, PV	Application Reset
NIPRx	Node x Interrupt Pointer Reg.	$0208_H + x \times 100_H$	U, PV	U, PV	Application Reset
NPCRx	Node x Port Control Register	$020C_H + x \times 100_H$	U, PV	U, PV	Application Reset
NBTRx	Node x Bit Timing Register	$0210_H + x \times 100_H$	U, PV	U, PV	Application Reset
NECNTx	Node x Error Counter Register	$0214_H + x \times 100_H$	U, PV	U, PV	Application Reset
NFCRx	Node x Frame Counter Register	$0218_H + x \times 100_H$	U, PV	U, PV	Application Reset
Message Object Registers					
MOFCRn	Message Object n Function Control Register	$1000_H + n \times 20_H$	U, PV	U, PV	Application Reset
MOFGPRn	Message Object n FIFO/ Gateway Pointer Register	$1004_H + n \times 20_H$	U, PV	U, PV	Application Reset
MOIPRn	Message Object n Interrupt Pointer Register	$1008_H + n \times 20_H$	U, PV	U, PV	Application Reset

(table continues...)

Table 125 (continued) Registers Overview - MultiCAN+ Kernel Registers

Register Name	Description	Offset Addr[1]	Access Mode[2]		Reset
			Read	Write	
MOAMRn	Message Object n Acceptance Mask Register	$100C_H + n \times 20_H$	U, PV	U, PV	Application Reset
MODATALn	Message Object n Data Register Low	$1010_H + n \times 20_H$	U, PV	U, PV	Application Reset
MODATAHn	Message Object n Data Register High	$1014_H + n \times 20_H$	U, PV	U, PV	Application Reset
MOARn	Message Object n Arbitration Register	$1018_H + n \times 20_H$	U, PV	U, PV	Application Reset
MOCTRn MOSTATn	Message Object n Control Reg. Message Object n Status Reg.	$101C_H + n \times 20_H$	U, PV	U, PV	Application Reset

1) The absolute register address is calculated as follows:
Module Base Address (Table 124) + Offset Address (shown in this column)
Further, the following ranges for parameters i, k, x, and n are valid: i = 0-15, k = 0-7, x = 0-1, n = 0-31.
2) Accesses to empty addresses: nBE

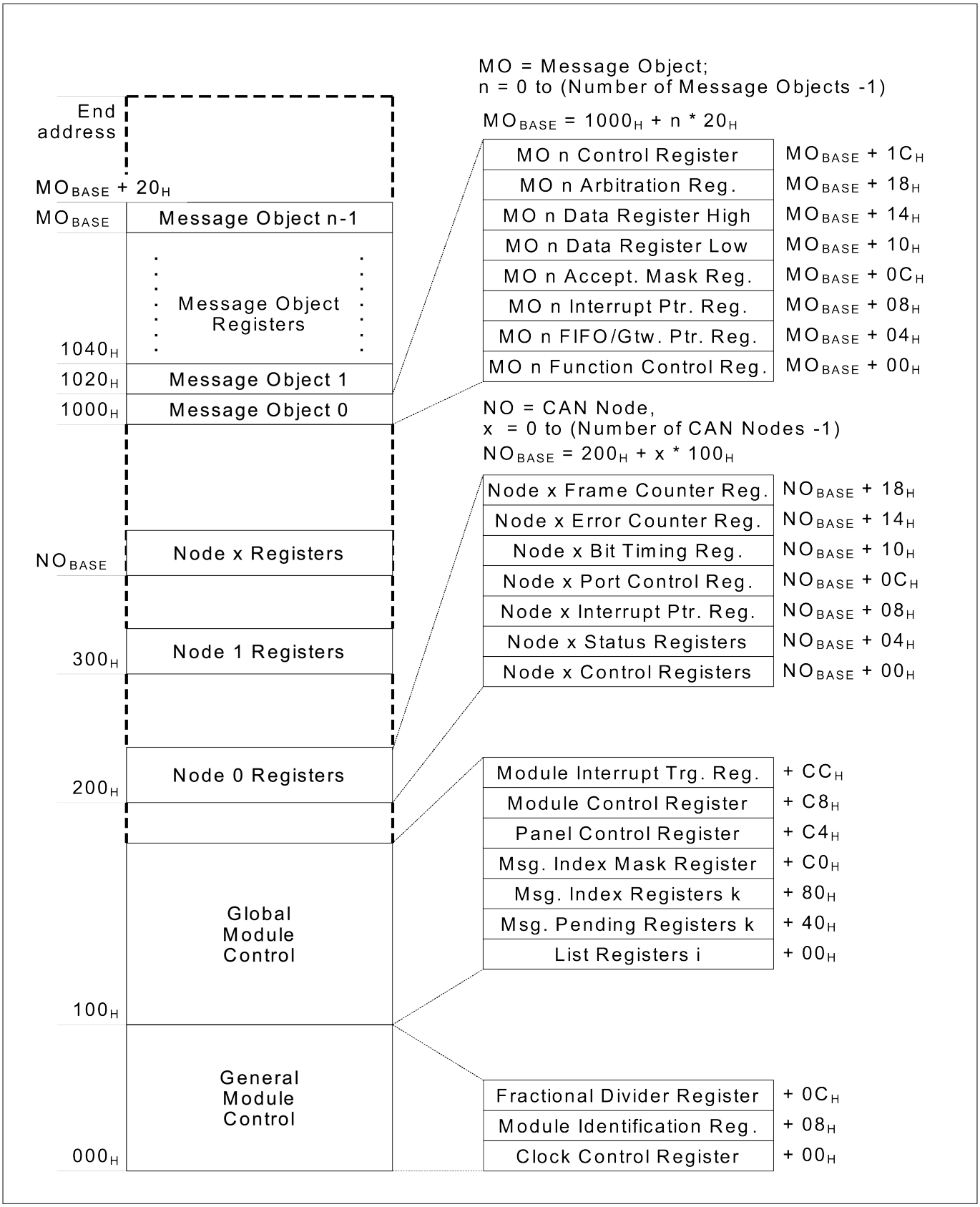


Figure 160 MultiCAN+ Register Address Map PSC1Mx

17.5.1 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.

17.5.1.1 Panel Control Register

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 126 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 deallocation 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.
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).

(table continues...)

Table 126 (continued) Panel Commands

PANCMD	PANAR2	PANAR1	Command Description
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.
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.

(table continues...)

Table 126 (continued) Panel Commands

PANCMD	PANAR2	PANAR1	Command Description
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**

The Module Control Register MCR contains basic settings that determine the operation of the MultiCAN+ module.

The write access to the lowest byte of the MCR register is possible only if the CCE bits of all CAN nodes are set (NCRx.CCE bits). The NCRx.INIT bits will be automatically set when the lowest byte of the MCR register is written, independent of the setting of the CCE bits. The INIT bits have to be reset by software in order to activate the CAN nodes.

The reconfiguration of the clock source has to be done by using two writes: first a write of zero to the CLKSEL bit field, and then a second write defining the new clock source. Between the first and the second write a delay of $4 / f_A + 2 / f_{CAN}$ number of cycles must be inserted by software, where f_A is the frequency being switched off with the first write. Exception: in case that f_{MCLK} is selected as the baud rate logic clock (MCR.CLKSEL = 1), no delay cycles between the writes are necessary. In both cases, simply using one write defining the new clock source is not allowed.

***Note**: If the baud rate logic is supplied from an unstable clock source, or no clock at all, the CAN functionality is not guaranteed.*

17.5.2 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.

The Node Control Register contains basic settings that determine the operation of the CAN node.

17.5.2.1 Node x Status Register

Encoding of the LEC Bit field

Table 127 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.
110_B	**CRC Error:** The CRC checksum of the received message was incorrect.
111_B	**CPU write to LEC:** Whenever 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.

17.5.2.2 Node x Frame Counter Register

Bit Timing Analysis Modes

Table 128 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.
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 129)
110_B	Reserved, do not use this combination.
111_B	Reserved, do not use this combination.

Table 129 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.

(table continues...)

Table 129 (continued) CAN Bus State Information

CFC[13:12]	CAN Bus State
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.

17.5.3 Message Object Registers

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. Therefore the reset value is written as 0_H, even though the read part of the register has a different reset value.

17.5.3.1 Message Object 31 Control Register

Table 130 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”).

17.5.3.2 Message Object 31 Status Register

Table 131 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	$0402\ 0000_H$
…	…	…	…
31	31	30	$1F1E\ 0000_H$

17.5.3.3 Message Object n Arbitration Register

Transmit Priority of Msg. Objects based on CAN Arbitration Rules

Table 132 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.MOSTAT.DIR = 1 (send Data Frame) B.MOSTAT.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.MOSTAT.DIR = 1 (send Data Frame) B.MOSTAT.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.

17.6 MultiCAN+ Module Implementation

This section describes CAN module interfaces with the clock control, port connections, interrupt control, and address decoding.

17.6.1 Interfaces of the MultiCAN+ Module

Figure 161 shows the PSC1Mx specific implementation details and interconnections of the MultiCAN+ module. The I/O lines of the MultiCAN+ module (two I/O lines of each CAN node) are connected to the Ports listed in Table 134. The MultiCAN+ module is also supplied by clock control, interrupt control, and address decoding logic. MultiCAN+ interrupts can be directed to the CPU, CCU4 modules which are able to trigger CCU4, CPU operations.

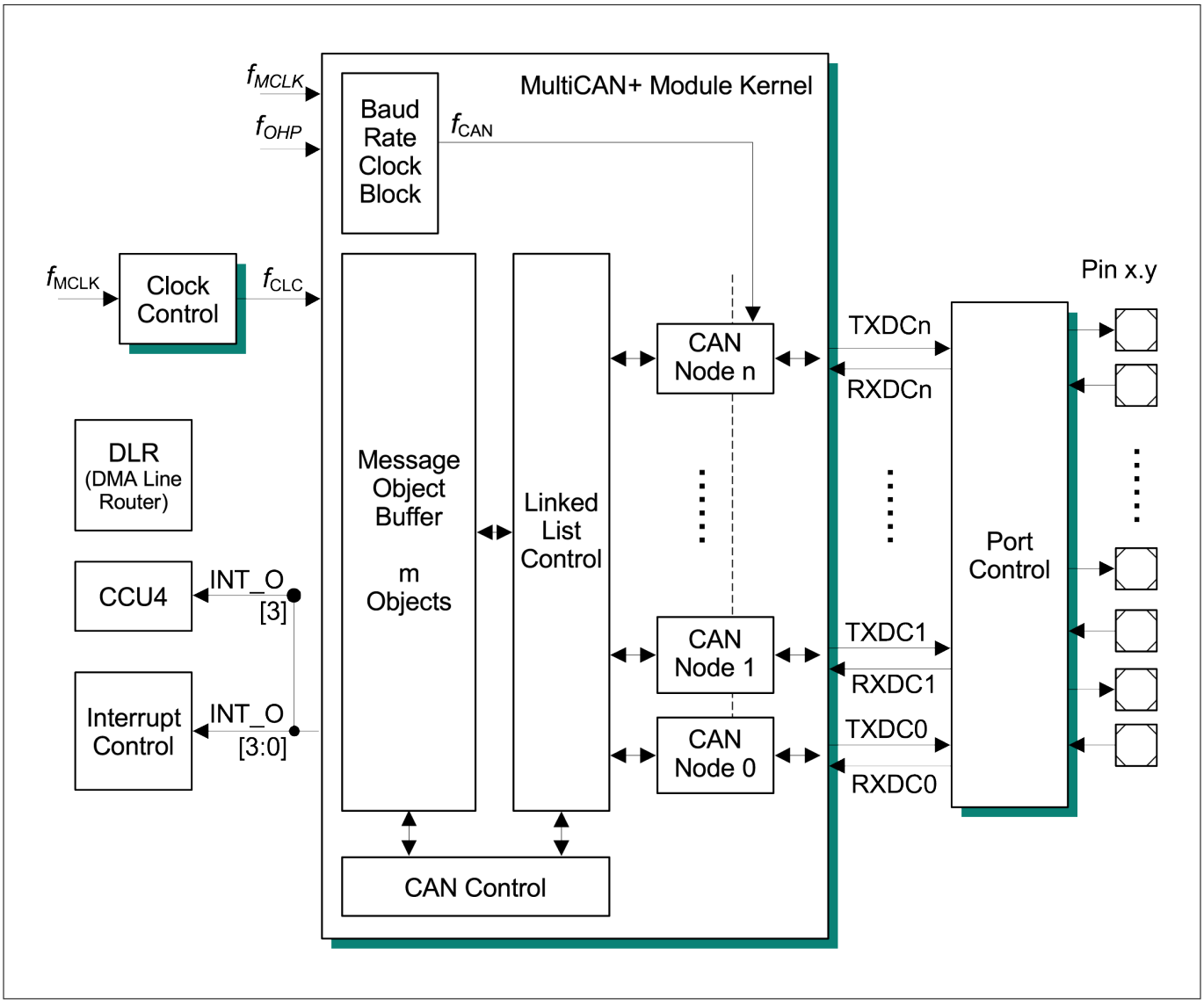


Figure 161 MultiCAN+ Module Implementation and Interconnections with n := 1 and m := 32 for PSC1Mx

17.6.2 MultiCAN+ Module External Registers

The registers listed in Figure 162 are not included in the MultiCAN+ module kernel, some registers must be programmed for proper operation of the MultiCAN+ module.

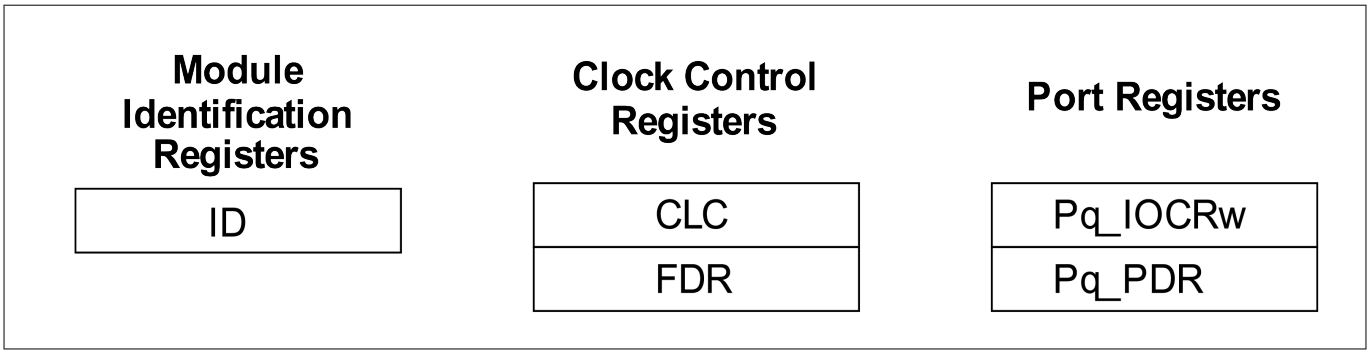


Figure 162 CAN Implementation-specific Special Function Registers

Table 133 MultiCAN+ Module External Registers

Register Name	Description	Offset Addr	Access Mode		Reset Class
			Read	Write	
Module Identification Registers					
ID	Module Identification Register	008_H	U, PV	nBE	Application Reset
Clock Control Registers					
CLC	Clock Control Register	000_H	U, PV	PV	Application Reset
FDR	Fractional Divider Register	$00C_H$	U, PV	PV	Application Reset

17.6.3 Module Clock Generation

This chapter describes the way the module get's its clock.

17.6.3.1 Clock Selection

The bit timing machine and the rest of the MultiCAN+ module are separate frequency domains and can be driven by separate independent frequencies. The bit timing unit can be driven by the AHB bus clock or with the direct oscillator clock, and the rest of the chip is driven only by the AHB bus clock.

The purpose of supplying the bit timing unit with a direct oscillator clock is to avoid the clock jitter added by the PLL, necessary when the chip is driven by a low cost ceramic resonator instead of by high precision quartz crystal.

Selecting the clock source for the bit timing unit is done by programming the bit-field MCR.CLKSEL.

Enabling and disabling the clock of the module by using CLC.DISR affects always both frequency domains, so that when f_{CLC} is switched off, f_A is also switched off.

17.6.3.2 Fractional Divider

As shown in Figure 163, 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}

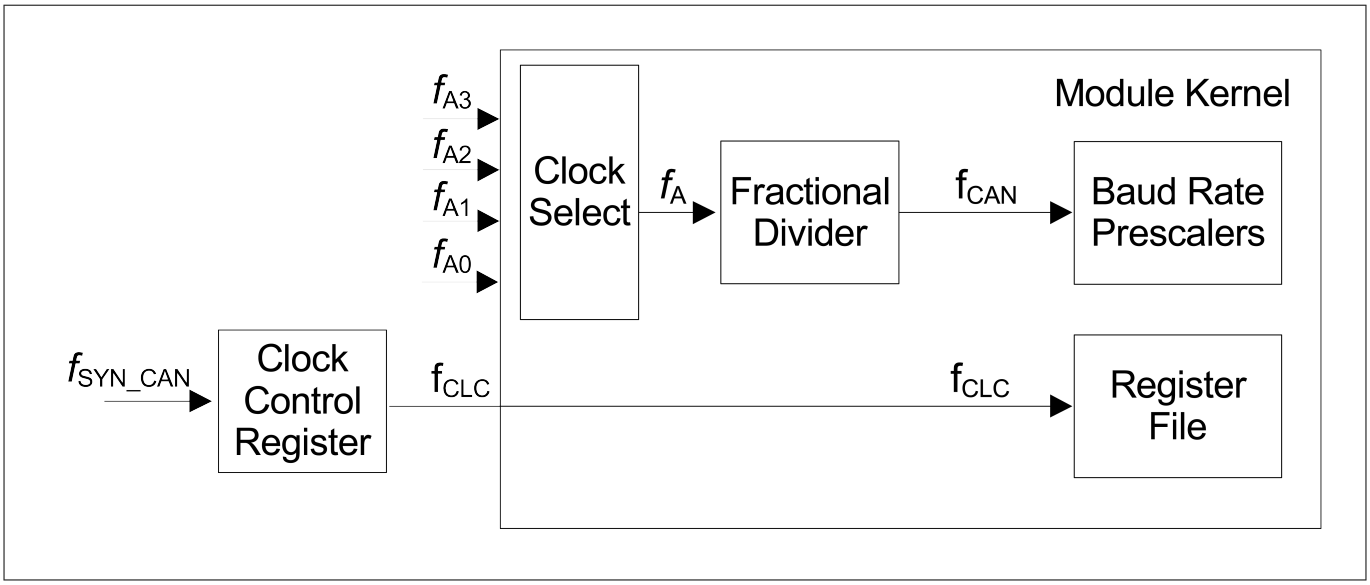


Figure 163 MultiCAN+ Module Clock Generation

The f_{SYN_CAN} is identical to f_{MCLK}.

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_{MCLK}. 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 (for example, bit timing). The settings in the CAN_FDR register determine the frequency of the module timer clock f_{CAN} according the following two formulas:

$$f_{CAN} = f_A \times \frac{1}{n} \text{ with } n = 1024 - \text{CAN_FDR.STEP} \quad (34)$$

$$f_{CAN} = f_A \times \frac{n}{1024} \text{ with } n = 0 - 1023 \quad (35)$$

Equation (34) applies to normal divider mode (CAN_FDR.DM = 01_B) of the fractional divider. Equation (35) 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).*

17.6.4 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)

17.6.4.1 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, which are described in the datasheet. In case of discrepancies between datasheet and CAN chapter, the description in the datasheet is correct.

Table 134 shows the corresponding pins, sorted by node. Even though the table is pair wise, it is possible to select a different pairing for RXD and TXD. In addition the RXSEL value to be programmed for the Receive Pins is part of this table. For more information on RXSEL please see Node Receive Input Selection.

Table 134 MultiCAN+ I/O Control Selection and Setup for PSC1Mx

Node	RXD	NPCRx.RXSEL	TXD
CAN0	P0.4 / RXDA	000_B	P0.5 / TXD
	P0.5 / RXDB	001_B	P0.4 / TXD
	P0.14 / RXDC	010_B	P0.15 / TXD
	P0.15 / RXDD	011_B	P0.14 / TXD
	P2.0 / RXDE	100_B	P2.1 / TXD
	P2.1 / RXDF	101_B	P2.0 / TXD
	P1.0 / RXDG	110_B	P1.1 / TXD
	P1.1 / RXDH	111_B	P1.0 / TXD
CAN1	P0.12 / RXDA	000_B	P0.13 / TXD
	P0.13 / RXDB	001_B	P0.12 / TXD
	P4.8 / RXDC	010_B	P4.9 / TXD
	P4.9 / RXDD	011_B	P4.8 / TXD
	P2.10 / RXDE	100_B	P2.11 / TXD
	P2.11 / RXDF	101_B	P2.12 / TXD
	P1.2 / RXDG	110_B	P1.3 / TXD
	P1.3 / RXDH	111_B	P1.2 / TXD

17.6.4.2 Node Receive Input Selection

Additionally to the I/O control selection, as defined in the datasheet, 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 according to Table 134. Values for NPCRx.RXSEL other than those of the table mentioned above will result in a recessive receive input for node x. As a hint A results in 0_H, B in 1_H until H resulting in the value of 7_H.

This feature allows, for example, a CAN node which operates in analyzer mode to monitor the receive operations of its neighbor CAN node.

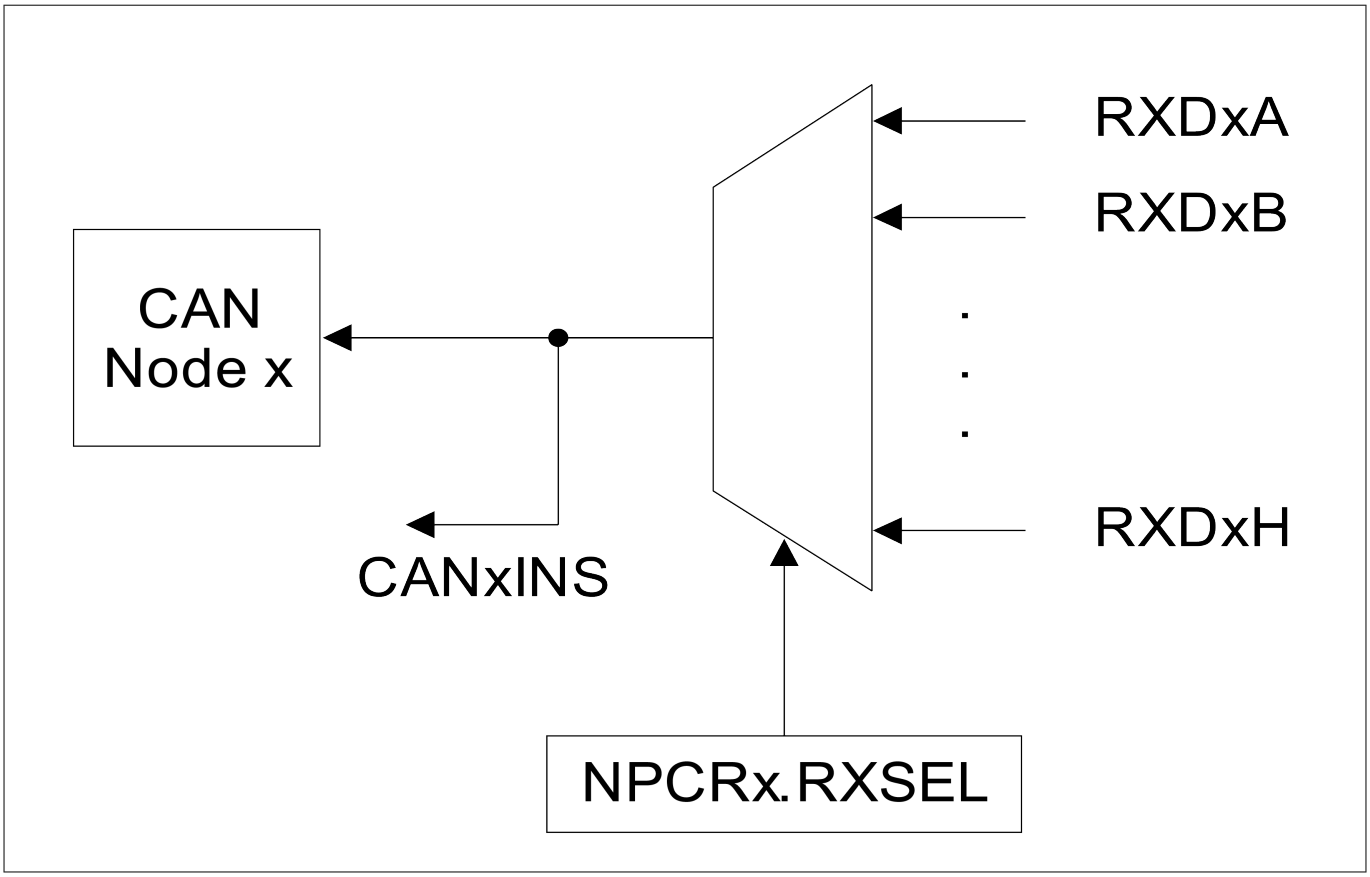


Figure 164 CAN Module Receive Input Selection

17.6.4.3 Connections to Interrupt Router Inputs

The interrupt output lines INT_O0-7 are connected to the Interrupt Router module, see Table 135.

Table 135 Interrupt Router Inputs

Interrupt Router Input	Connected to CAN Interrupt Output
SRC_CANINT0	INT_O0
SRC_CANINT1	INT_O1
SRC_CANINT2	INT_O2
SRC_CANINT3	INT_O3
SRC_CANINT4	INT_O4
SRC_CANINT5	INT_O5
SRC_CANINT6	INT_O6
SRC_CANINT7	INT_O7

17.6.4.4 Connections to ERU

On PSC1Mx the following connections to the ERU exist:

Table 136 CAN0 Pin Connections PSC1Mx

Global Input/Output	I/O	Connected To	Description
CAN0.SRC_CANINT0	O	NVIC	Service Request Output 0
CAN0.SRC_CANINT1	O	NVIC	Service Request Output 1
CAN0.SRC_CANINT2	O	NVIC	Service Request Output 2
CAN0.SRC_CANINT3	O	NVIC	Service Request Output 3
CAN0.SRC_CANINT4	O	ERU1.OGU02	Service Request Output 4
CAN0.SRC_CANINT5	O	ERU1.OGU12	Service Request Output 5
CAN0.SRC_CANINT6	O	ERU1.OGU22	Service Request Output 6
CAN0.SRC_CANINT7	O	ERU1.OGU32	Service Request Output 7

17.6.5 Interrupt Control

The interrupt control logic in the MultiCAN+ module uses an interrupt compressing scheme that allows high flexibility in interrupt processing. There are hardware and software interrupt sources available:

- CAN node interrupts:
  - Five different interrupt sources for each of the 2 CAN nodes = 5 * 2 interrupt sources
- Message object interrupts:
  - Two interrupt source for each message object = 2 * 32 interrupt sources
- One register (MITR) to initiate 16 interrupts via software

Each of the 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 165 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.

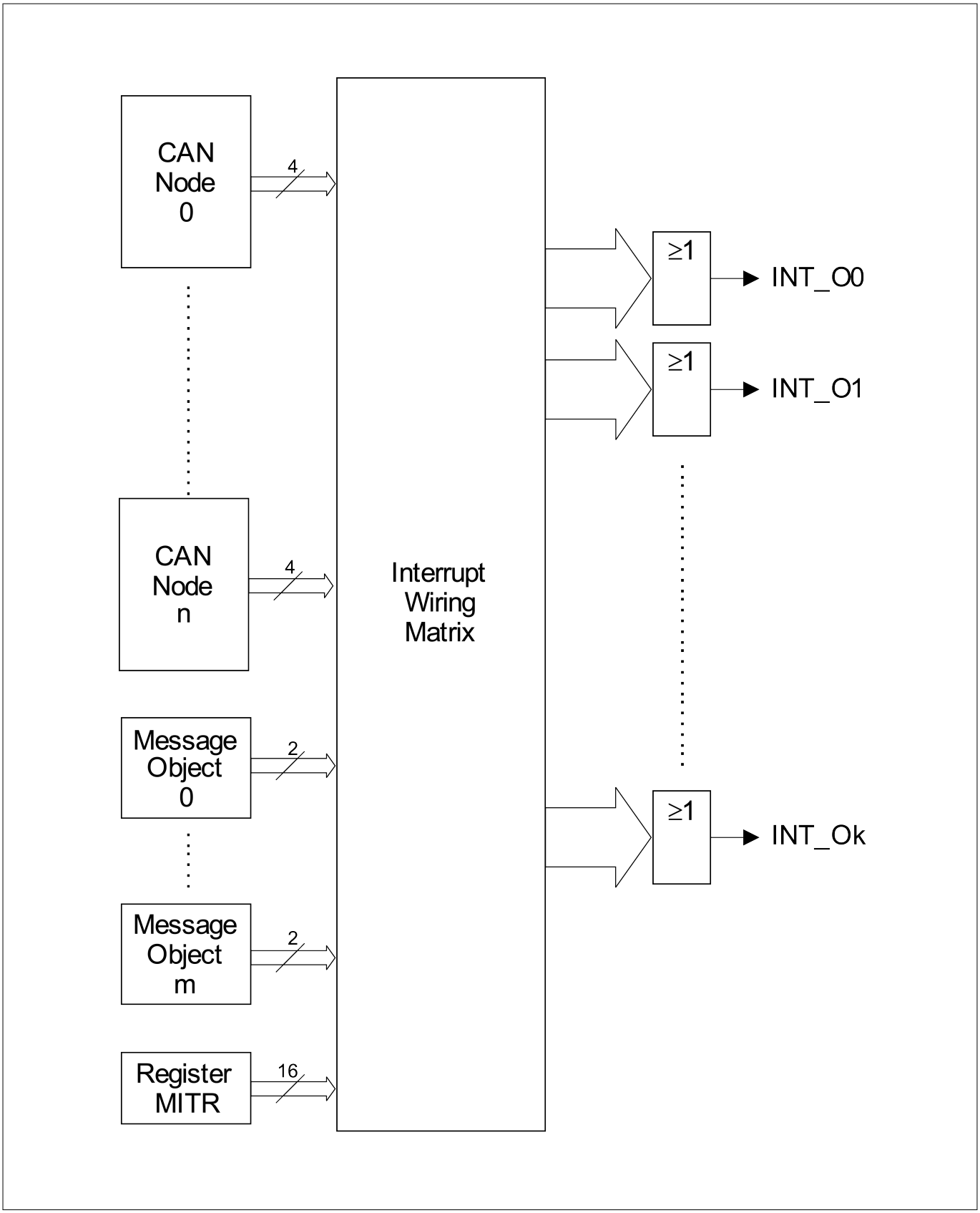


Figure 165 Interrupt Compressor n = 2, m = 32 and k = 8

17.6.6 MultiCAN+ Module Register Address Map

The complete MultiCAN+ module register address map of Figure 166 also shows the general implementation-specific registers for clock control, module identification, interrupt service request control and the absolute address information.

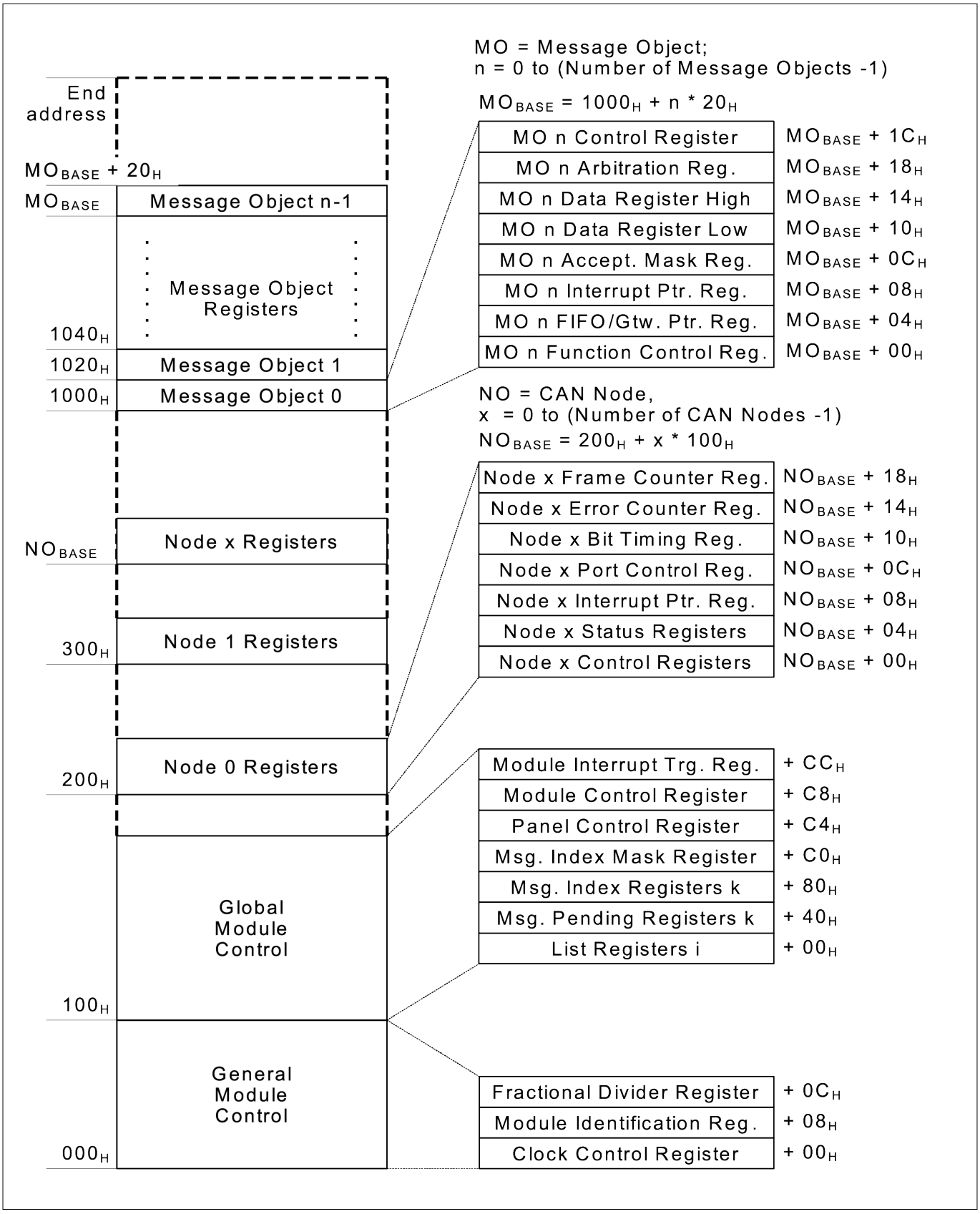


Figure 166 MultiCAN+ Register Address Map PSC1Mx

18 Versatile Analog-to-Digital Converter (VADC)

The PSC1Mx 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 PSC1Mx is based on Sample&Hold converters, where a cluster contains 2 Sample&Hold units which share a common converter.

The number of analog input channels and ADCs depends on the chosen product type (please refer to Product-Specific Configuration on page 468).

Table 137 Abbreviations used in ADC chapter

Abbreviation Meaning	Meaning
ADC	Analog to Digital Converter
ADC Kernel	also known as ADC Group
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)
S&H	Sample and Hold
SCU	System Control Unit of the device
SHS	Sample and Hold Sequencer
TUE	Total Unadjusted Error

18.1 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 PSC1Mx 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.

Feature List

The following features describe the functionality of the ADC cluster:

- Input voltage range from 0 V up to analog supply voltage
- Standard (V_{AREF}) reference voltage source and alternate Ground (CH0) selectable for each channel to support ratiometric measurements and different signal scales

- Internal reference voltage source for low voltage range operation
- Two independent sample and hold stages with 8 analog input channels each
- Two active sigma delta hold blocks provided supporting oversampling
- 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) in each group
  - Configurable auto scan conversion (single or repeated) in the background (all channels)
  - 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 (2 group-specific, 2 shared interrupts)
- Built-in safety feature: Broken wire detection with programmable default levels
- Support of debug suspend and power saving modes

***Note**: Additional functions are available from the out of range comparator (see description in the SCU).*

Table 138 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 2 input signals	Multi-phase current measurement

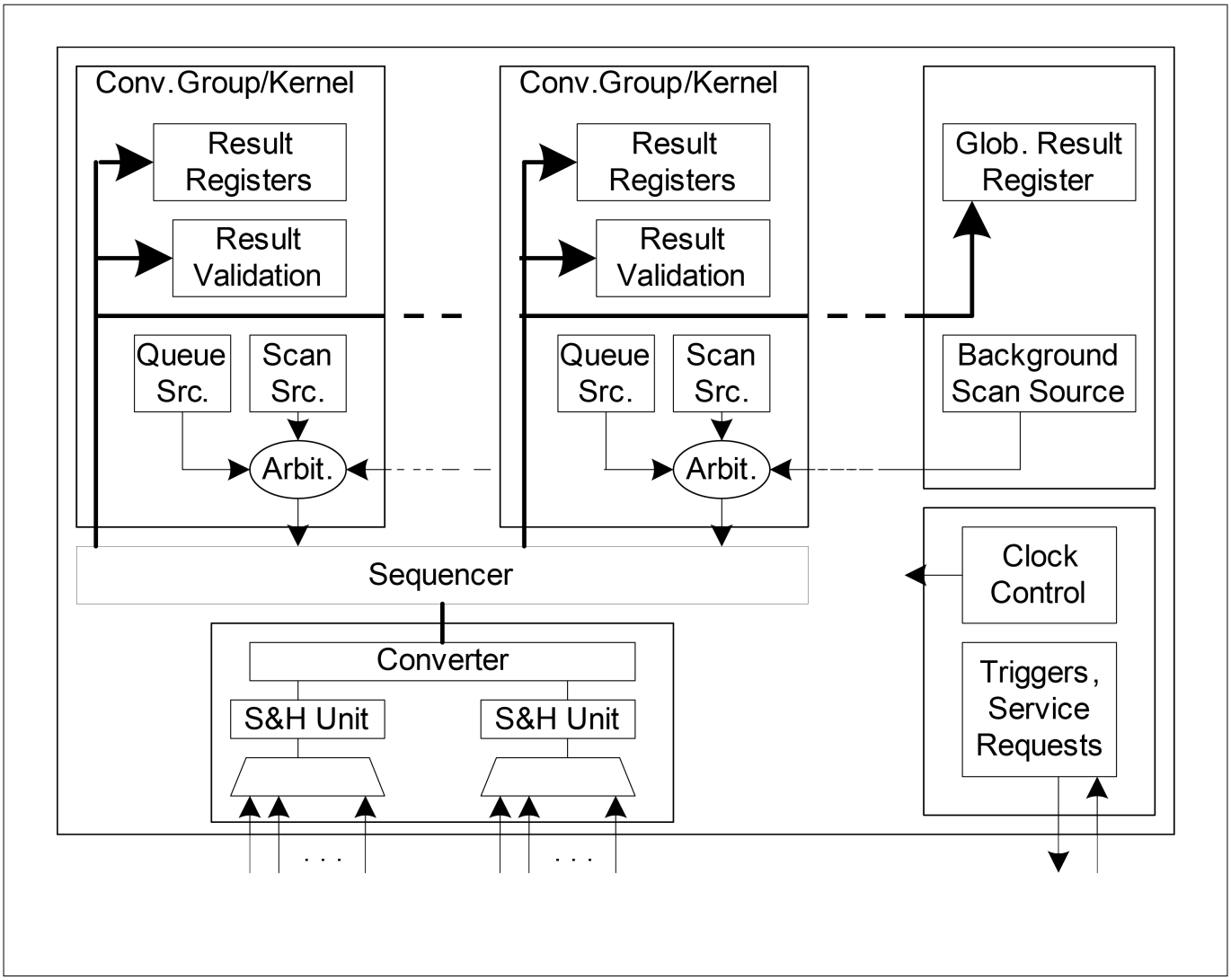


Figure 167 ADC Structure Overview

18.2 Introduction and Basic Structure

The Versatile Analog to Digital Converter module (VADC) of the PSC1Mx 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.

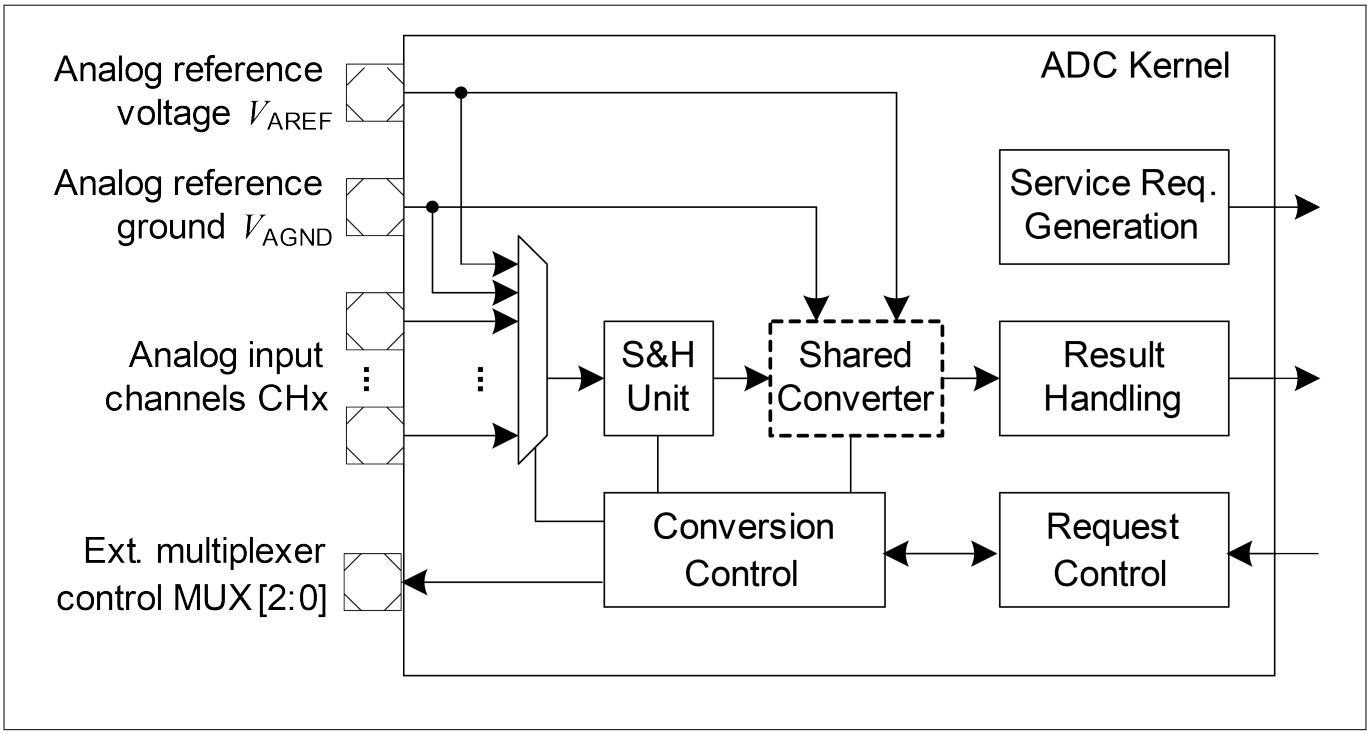


Figure 168 ADC Kernel Block Diagram

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.

Each kernel provides a dedicated Sample&Hold unit connected to the input multiplexer. Several kernels build a cluster and share a common high-speed converter, whose performance can be assigned to the connected kernels.

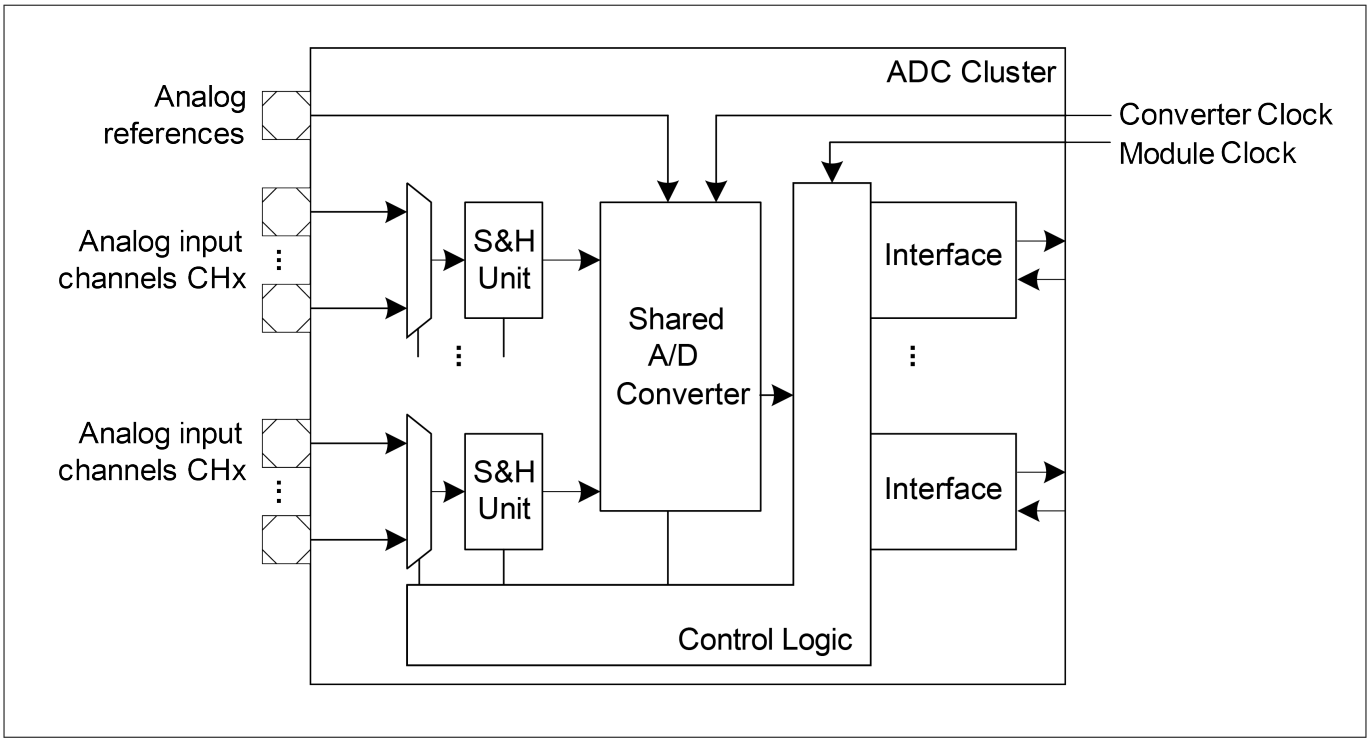


Figure 169 ADC Cluster Structure

The basic module clock f_{ADC} is connected to the system clock signal f_{MCLK}.

The converter clock f_{CONV} is connected to a 32-MHz or 48-MHz clock signal. The default frequency is 32 MHz.

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 2 signals can be converted in parallel.

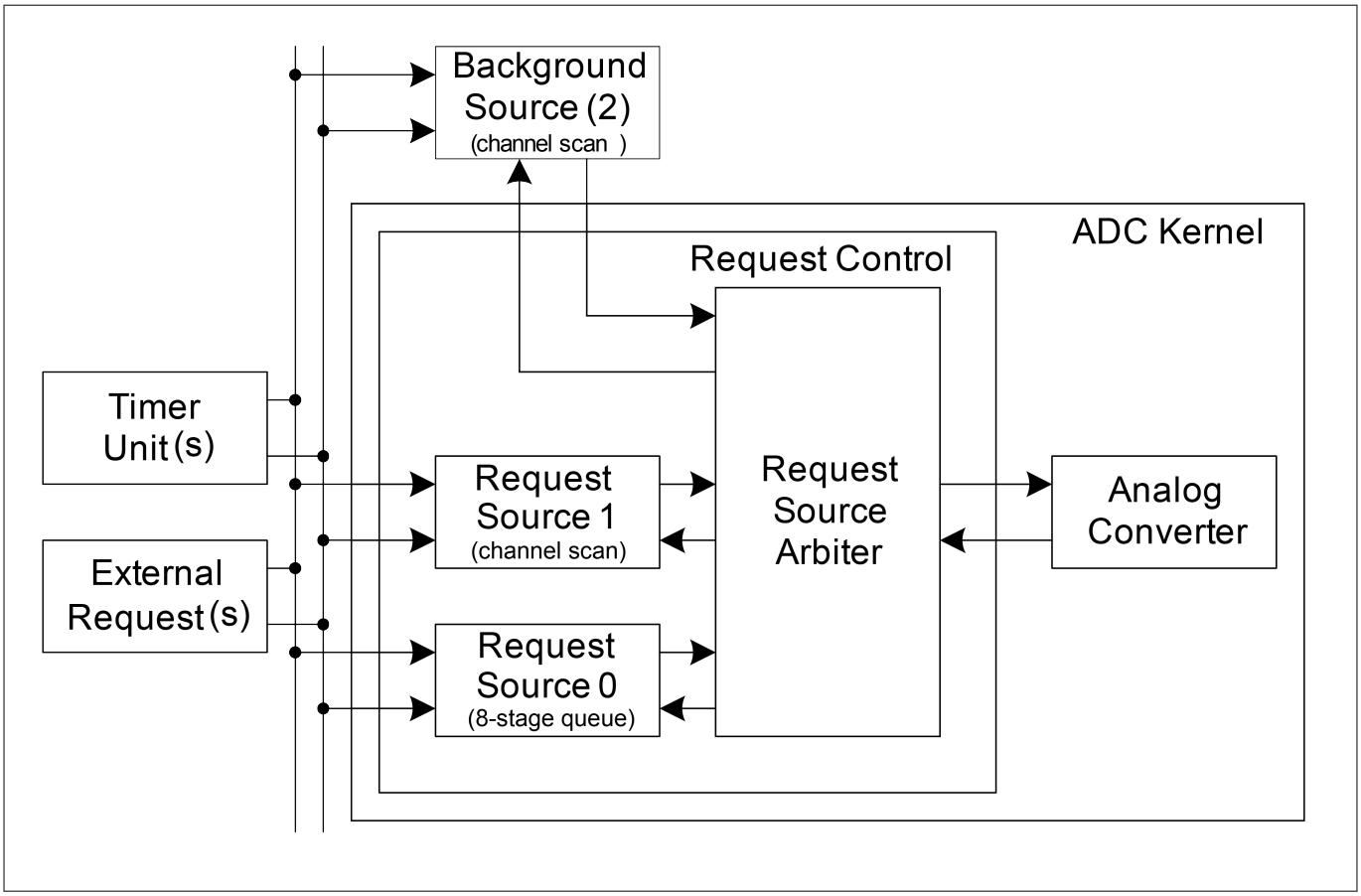


Figure 170 Conversion Request Unit

Input Channel Selection

The analog input multiplexer selects one of the available analog inputs (CH0 - CHx[7]) 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* Interconnects*.*

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.

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.

[7] The availablity of dedicated input channels depends on the package of the used product type. A summary can be found in Analog Module Connections in the PSC1Mx

The sample&hold units of a cluster share a common high-speed converter. This converter sequentially generates result values for all sample&hold units. Calibration cycles are executed automatically.

For broken wire detection (see Broken Wire Detection), the converter network can be precharged to a selected voltage 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 (for example, 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.

Interrupt Service Request Generation

Several ADC events can issue service requests to the CPU:

- **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, as a result the interrupts 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 outputs. This allows grouping the requests according to the requirements of the application.

Refer to Service Request Generation on Page 464 for connection details.

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.

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.

The pull devices are controlled via the standard port control registers.

18.3 Electrical Models

Each conversion of an analog input voltage to a digital value consists of two consecutive phases:

- During the sample phase, the input voltage is sampled and stored. The input signal path is a simplified model for this.
- During the conversion phase the stored voltage is converted to a digital result.

Input Signal Path

The ADC of the PSC1Mx uses a switched capacitor field represented by CAINSW (small parasitic capacitances are present at each input pin). During the sample phase, the capacitor field CAINSW is connected to the selected analog input CHx via the input multiplexer (modeled by ideal switches and series resistors RAIN).

The switch to CHx is closed during the sample phase and connects the capacitor field to the input voltage VAINx.

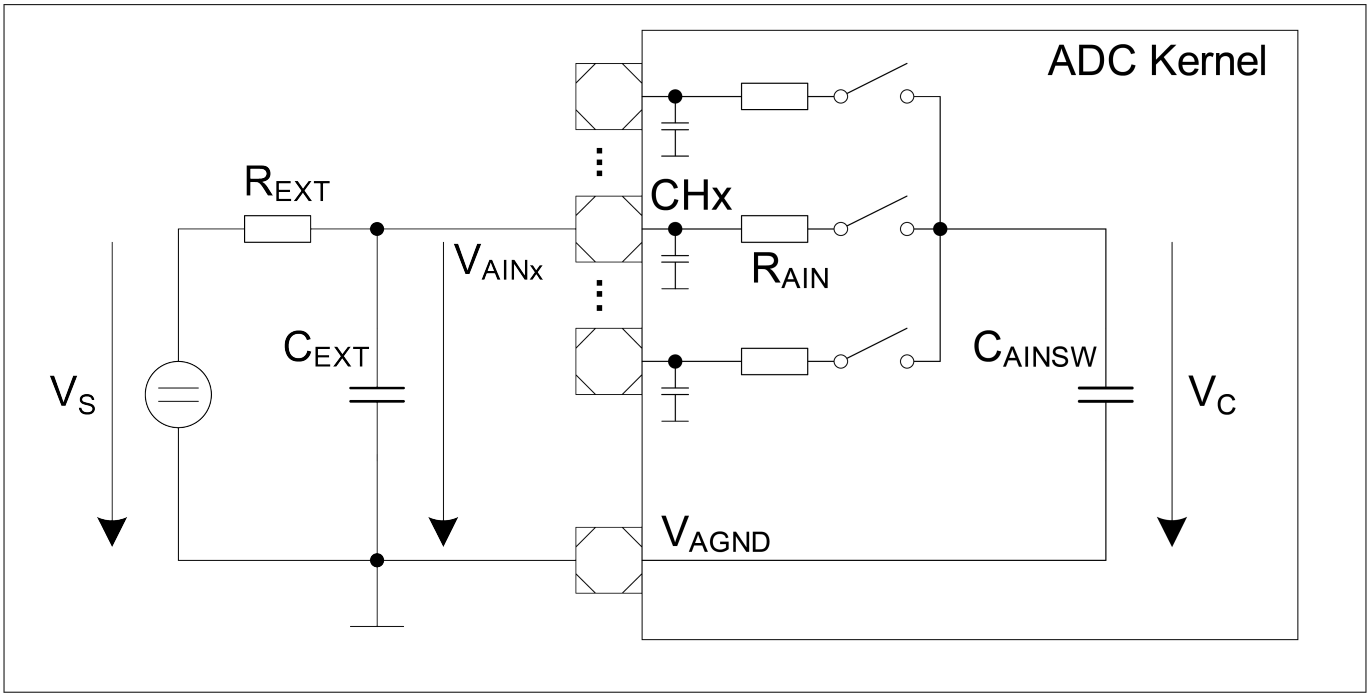


Figure 171 Signal Path Model

A simplified model for the analog input signal path is given in Figure 171. An analog voltage source (value V_S) with an internal impedance of R_{EXT} delivers the analog input that should be converted.

During the sample phase the corresponding switch is closed and the capacitor field C_{AINSW} is charged. Due to the low-pass behavior of the resulting RC combination, the voltage V_C to be actually converted does not immediately follow V_S. The value R_{EXT} of the analog voltage source and the desired precision of the conversion strongly define the required length of the sample phase.

To reduce the influence of R_{EXT} and to filter input noise, it is recommended to introduce a fast external blocking capacitor CEXT at the analog input pin of the ADC. Like this, mainly CEXT delivers the charge during the sample phase. This structure allows a significantly shorter sample phase than without a blocking capacitor, because the low-pass time constant defining the sample time is mainly given by the values of R_{AIN} and CAINSW.

The resulting low-pass filter of R_{EXT} (usually a parameter of the signal source) and CEXT should be dimensioned according to the application's requirements:

- For quickly changing dynamic signals, a smaller capacitor allows V_{AINx} to follow V_S between two sample phases of the same analog input channel.
- For high-precision conversions, an external blocking capacitor C_{EXT} in the range of at least $2^n \times C_{AINSW}$ keeps the voltage change of V_{AINx} during the sample phase below 1L SB_n. This voltage change is due to the charge redistribution between C_{EXT} and C_{AINSW}.

Leakage current through the analog input structure of the ADC can generate a voltage drop over R_{EXT}, introducing an error. The ADC input leakage current increases at high temperature and if the input voltage level is close to the analog supply ground V_{SS} or to the analog power supply V_{DDPA}. The input leakage current of an ADC channel can be reduced by avoiding input voltages close to the supplies.

An overload condition (input voltage exceeds the supply range) at adjacent analog inputs injects an additional leakage current (defined by a coupling factor) .

The capacitor C_{AINSW} is automatically discharged at the beginning of a sample phase.

Transfer Characteristics and Error Definitions

The transfer characteristic of the ADC describes the association of analog input voltages to the 2^n discrete digital result values (n bits resolution). Each digital result value (in the range of 0 to 2^n-1) represents an input voltage range defined by the reference voltage range divided by 2^n. This range (called quantization step or code width) represents the granularity (called LSB_n) of the ADC. The discrete character of the digital result generates a system-inherent quantization uncertainty of ±0.5 LSB_n for each conversion result.

The ideal transfer curve has the first digital transition (between 0 and 1) when the analog input reaches 0.5 LSB_n. The quantization steps are equally distributed over the input voltage range.

Analog input voltages below or above the reference voltage limits lead to a saturation of the digital result at 0 or 2^n-1.

The real transfer curve can exhibit certain deviations from the ideal transfer curve:

- The **offset error** is the deviation of the real transfer line from the ideal transfer line at the lowest code. This refers to best-fit lines through all possible codes, for both cases.
- The **gain error** is the deviation of the slope of the real transfer line from the slope of the ideal transfer line. This refers to best-fit lines through all possible codes, for both cases.
- The **differential non-linearity error** (DNL) is the deviation of the real code width (variation of the analog input voltage between two adjacent digital conversion results) from the ideal code width.
- The **integral non-linearity error** (INL) is the deviation of the real transfer curve from an adjusted ideal transfer curve (same offset and gain error as the real curve, but equal code widths).
- The **total unadjusted error** (TUE) describes the maximum deviation between a real conversion result and the ideal transfer characteristics over a given measurement range. Since some of these errors noted above can compensate each other, the TUE value generally is much less than the sum of the individual errors.

  The TUE also covers production process variations and internal noise effects (if switching noise is generated by the system, this generally leads to an increased TUE value).

18.4 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

18.4.1 General Clocking Scheme and Control

The A/D Converters of the PSC1Mx are supplied with a global clock signal from the system f_{ADC}. This clock signal controls all functions of all logic blocks and determines the overall timing. 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 converter clock f_{CONV} determines the performance of the converters themselves.

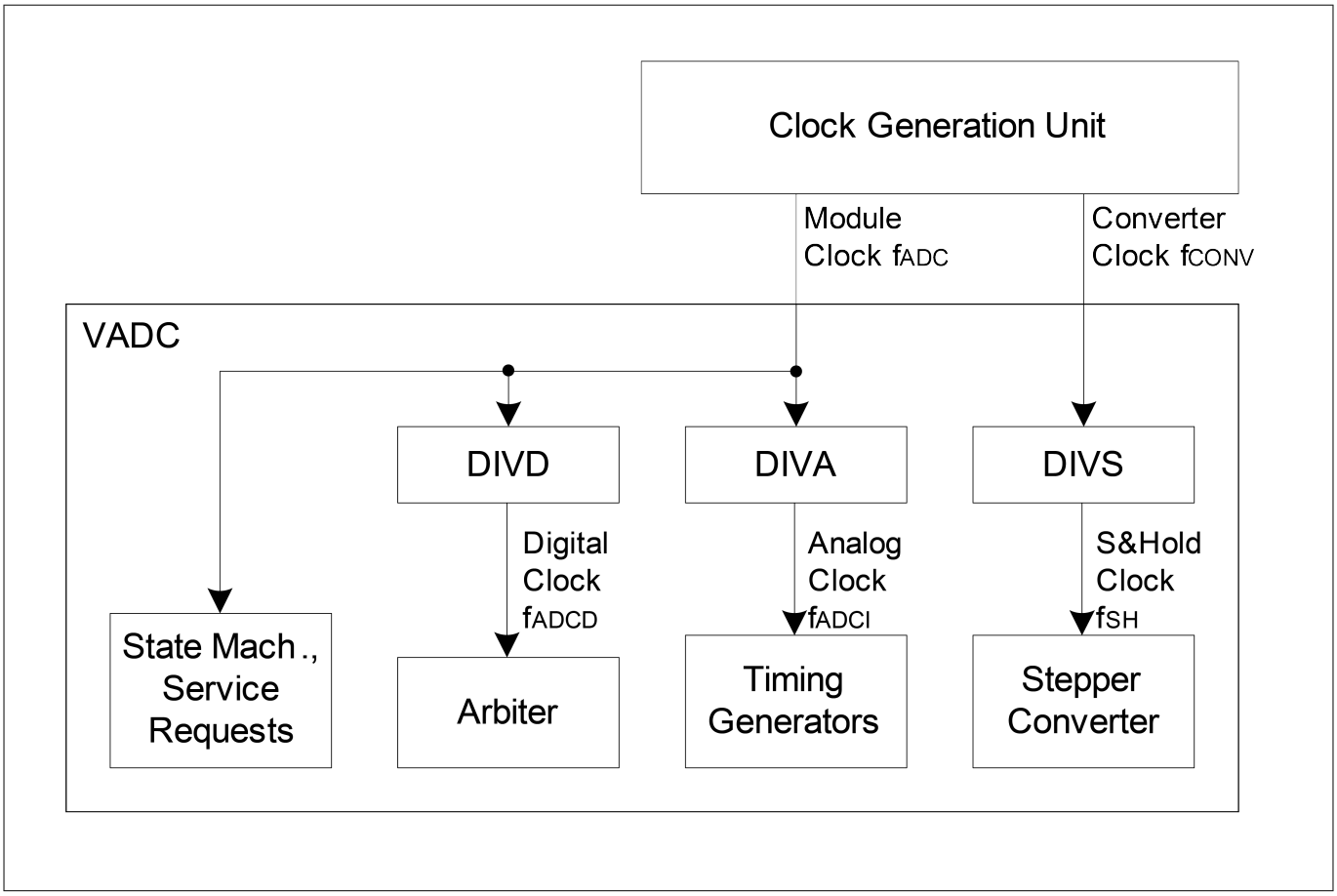


Figure 172 Clock Signal Summary

18.4.2 Register Access Control

Several protection schemes are provided to prevent unintended write access to control bitfields of the VADC.

- A specific register access control scheme provides a versatile protection scheme against unintended corruption of register contents. Registers ACCPROT0 and ACCPROT1 allow the restriction of write accesses for several groups of registers. The registers to be protected can be selected by the user. Table Register Protection Groups lists the registers that belong to each register group.

  Registers ACCPROT0/1 themselves are protected by the bit protection scheme described SCU chapter.
- Groups of bitfields within a register may also be protected by an associated write control bit. This write control bit (xxWC) must be written with 1 along with the write access to the intended bitfield(s).

18.4.3 Priority Channel and Result Register Assignment

Each channel and result register of a group can be assigned to this group's request sources and is then regarded as a priority channel or result register.

Priority Channel Assignment

An assigned priority channel can only be converted by its own group's request sources. An unassigned channel can also be converted by the background request source.

Priority Result Register Assignment

An assigned result register can only be written by its own group's request sources. A not assigned result register can also be written by the background request source.

18.5 Analog Module Activation and Control

The analog converter of the ADC is the functional block that generates the digital result values from the selected input voltage. It draws a permanent current during its operation and can be deactivated between conversions to reduce the consumed overall energy.

***Note**: After reset the analog converters are off. They must be enabled before triggering any action involving a converter.*

The accuracy of the conversions is improved by calibrating the analog converter to compensate process, temperature, and voltage variations.

18.5.1 Analog Converter Control

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: **Reserved**
- ANONS = 01_B: **Slow Standby mode**
  The converter enters a power save mode while no activity is required. It automatically returns to normal operation if a conversion is requested. Slow standby mode enables the lowest overall power consumption for the ADC supply.
  Requires the wakeup time (see below).
- ANONS = 00_B: **Converter switched Off** (default after reset)
  The converter is switched off if all groups of the corresponding cluster are 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 extended wakeup time (see below).

Bit ANOFF in register SHSCFG forces the analog part into power-down mode, without changing the configuration of the associated groups. While ANOFF = 0, the analog part is controlled automatically as selected by bitfields ANONS in the digital part. Since several groups (with separate ANONS bitfields) are serviced by the same cluster, the analog part is controlled by the following scheme:

Table 139 Analog Part Power-Down Control Options

ANON Settings[1)]	Power-Down Mode
All ANON signals are 00_B	Analog part off
ANON signals are 00_B or 01_B	Analog part automatically switched off while no sample phase and no conversion is active
One or more ANON signals are 11_B	Analog part always active

1) Taken into account while bit ANOFF = 0.

***Note**: Since switching off the analog part requires a wake-up phase, the optimum configuration depends on the actual application.*

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.*

18.5.2 Converter Handling in Deep Sleep Mode

During deep sleep mode the converter clocks are reduced or switched off. This also influences the converter's state machines.

To ensure proper operation after a deep sleep phase, the following sequence must be respected:

- Complete running conversions or sequences
- Disable the converters by clearing ANONC = 00_B
- Enter deep sleep mode
- - - - deep sleep phase[8] - - -
- Exit deep sleep mode
- Enable converters by setting ANONC = 01_B or 11_B
- Start new conversions or sequences

18.5.3 Calibration

Calibration automatically compensates deviations caused by process, temperature, and voltage variations. This ensures precise results throughout the operation time.

Several different calibration cycles are executed for offset and gain calibration. To minimize the extra time consumed by calibration, offset and gain calibration cycles are executed alternating.

An initial start-up calibration is required once after a reset for all converters. All converters must be enabled (ANONS = 11_B). The start-up calibration is initiated globally by setting bit SUCAL in register GLOBCFG. By setting bit SUCAL in register CALCTR the start-up calibration can be initiated for the corresponding cluster converter. Conversions may be started after the initial calibration sequence. This is indicated by bit CALS = 1_B AND bit CAL = 0_B.

The start-up calibration phase takes 1920 cycles.

At 32 MHz this is 1920 × 31.25 ns = 60 µs.

At 48 MHz this is 1920 × 20.83 ns = 40 µs.

After that, calibration cycles will compensate the effects of drifting parameters. The calibration cycles can be executed before or after (user configuration) a conversion sequence or can be disabled.

8 To reduce power consumption, the VADC module clock can be disabled during the deep sleep phase.

***Note**: Before initiating the start-up calibration without preceding reset, the converters must be disabled (ANONS = 00_B) and then enabled (ANONS = 11_B).*
After a start-up calibration in the lower supply range a settling time of 20 µs is required to ensure proper conversion results.

A calibration timer automatically triggers calibration cycles during idle phases of the ADC. This ensures the specified performance even after an idle phase.

***Note**: The ADC error depends on the temperature. Therefore, the calibration must be repeated periodically.*

Maximum Calibration Interval

While no conversions are requested, the stepper remains in its idle state. In order not to loose accuracy, a counter counts the time since the last executed calibration. This timer is loaded after each calibration with the start value from bitfield CALCTR.CALMAX and counts 128 × (CALMAX+1) clocks. If it expires it generates a calibration request which starts the stepper to execute a calibration cycle.

18.5.4 Reference Voltage Selection

The reference voltage that is required for conversions (V_{REF}) is generated internally. It is trimmed according to a calibration voltage which can be supplied on-chip (V_{REFINT}) or can be supplied externally (V_{AREF}).

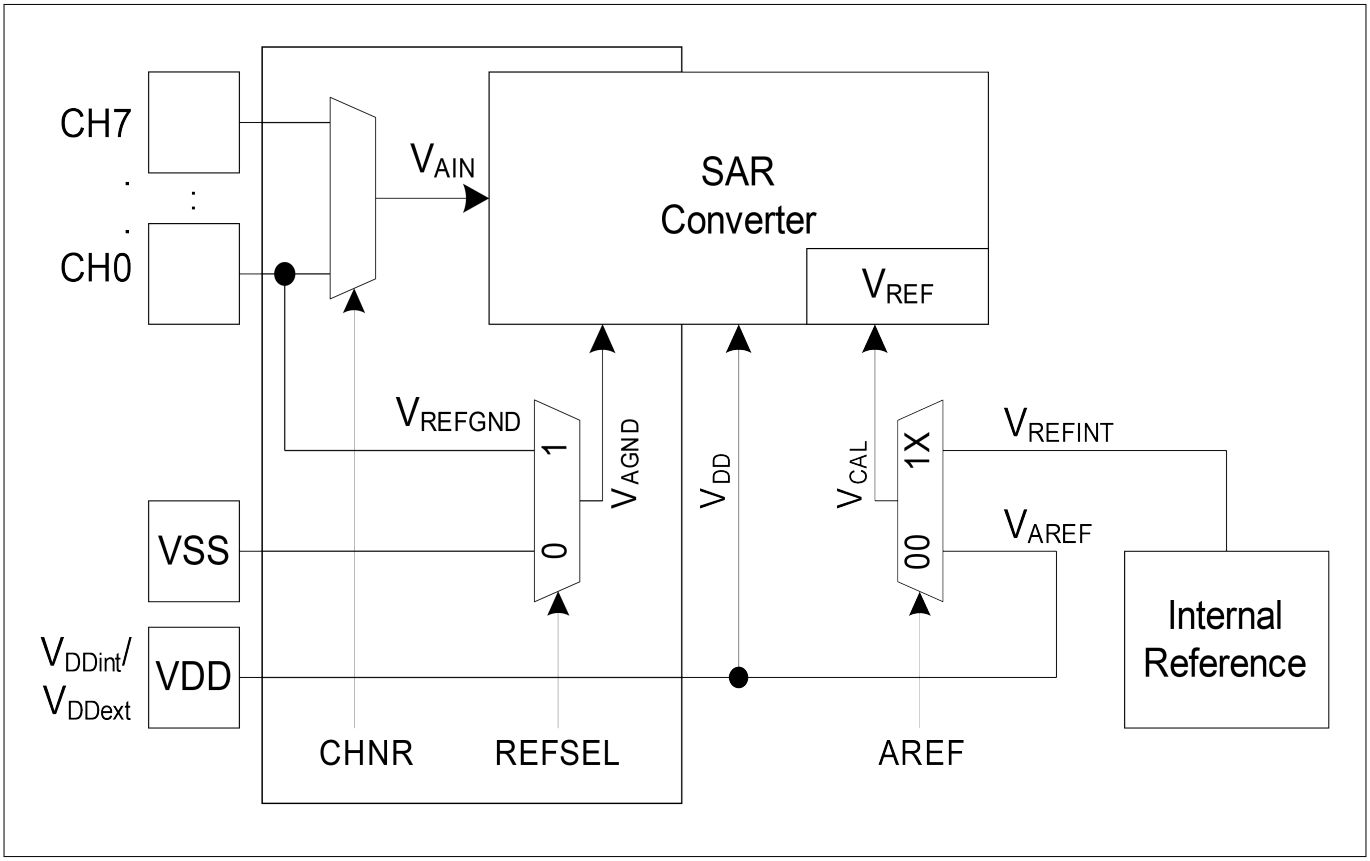


Figure 173 Calibration/Reference Voltage Sources

Bitfield AREF in register SHS0_SHSCFG selects the intended mode. For internal calibration voltage generation the operating voltage range (upper / lower) must be selected, external calibration voltage mode can only be used in upper voltage range.

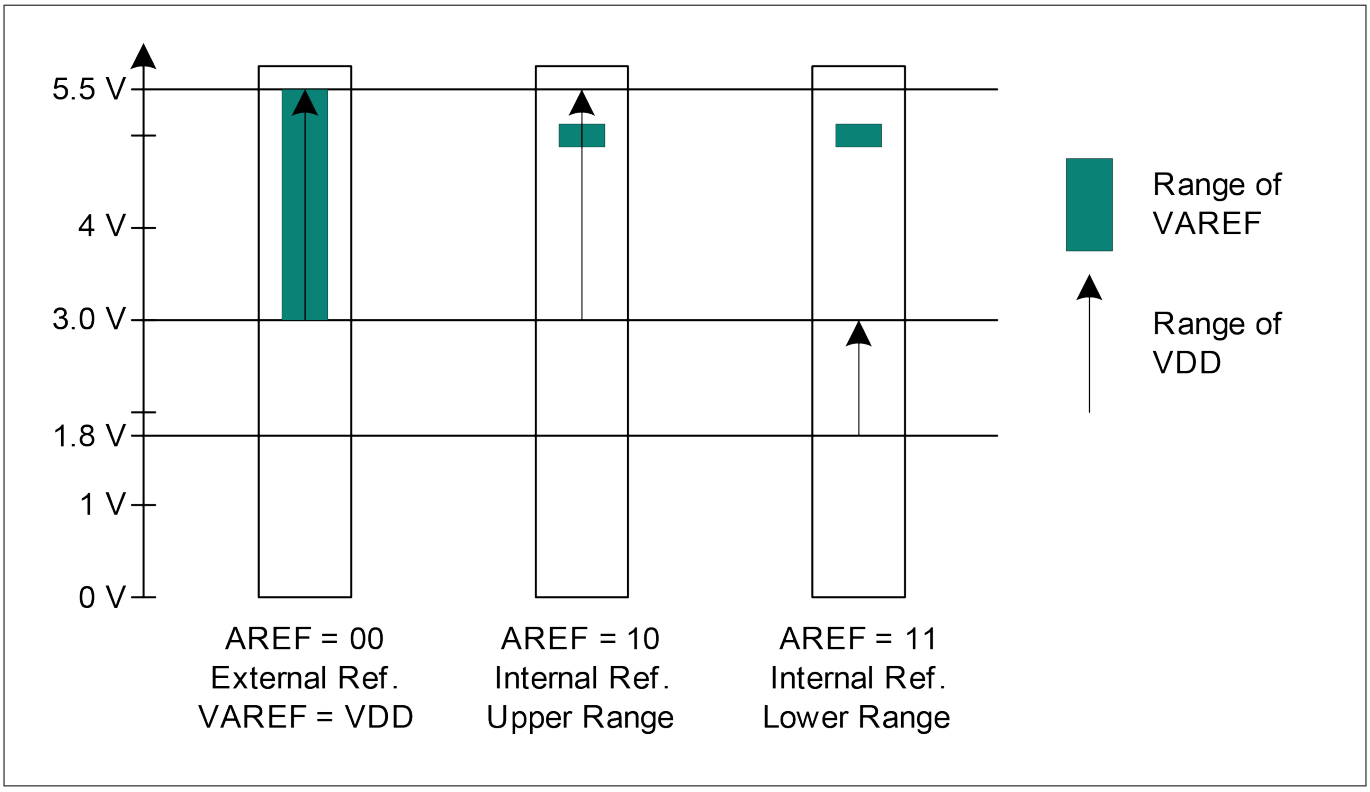


Figure 174 Reference Voltage Ranges

The internal reference voltage V_{REF} allows input voltages up to 5 V. However, the analog input voltage range is limited by the actual supply voltage V_{DD}. The full result range can be utilized by selecting higher gain factors according to the levels of the input signals.

***Note**: Operation in the lower supply voltage range requires longer sampling times for conversions and calibration.*

18.5.5 Sigma-Delta-Loop Function

Each standard analog-digital conversion incurs a quantization error due to the limited number of steps i.e. discrete result values. By activating the sigma-delta-loop function this residual quantization error can be forwarded to the next conversion. The residual charge is stored and added to the next sampled charge. Averaging a series of conversion results (with the loop enabled) can, therefore, reduce the induced quantization error.

***Note**: The data accumulation function of the VADCDIG already supports this averaging.*

The loop control bitfields are available in pairs in register SHS0_LOOP.

18.6 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 type and source, 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 sampling and 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 PSC1Mx are listed in Digital Module Connections in the PSC1Mx.

***Note**:* Figure 170 *on Page 421 summarizes the request sources.*

Two types of request 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[9]. 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 continuously or on a regular time base. For example, if programmed with medium priority, some input channels can be converted upon a specified event (for example, 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[9] can be enabled for the scan sequence. Each channel is converted once per sequence.

  A scan source converts a series of input channels continuously 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 conversions of all channels of all groups.

18.6.1 Queued Request Source Handling

A queued request source supports short conversion sequences (up to 8) of arbitrary channels of the same group (contrary to a scan request source with a fixed conversion order for the enabled channels). The programmed

[9] The availablity of dedicated input channels depends on the package of the used product type. A summary can be found in Introduction and Basic Structure. 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.

sequence is stored in a queue buffer (based on a FIFO mechanism). The requested channel numbers are entered via the queue input register GxQINR0 (x = 0 - 1), 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.

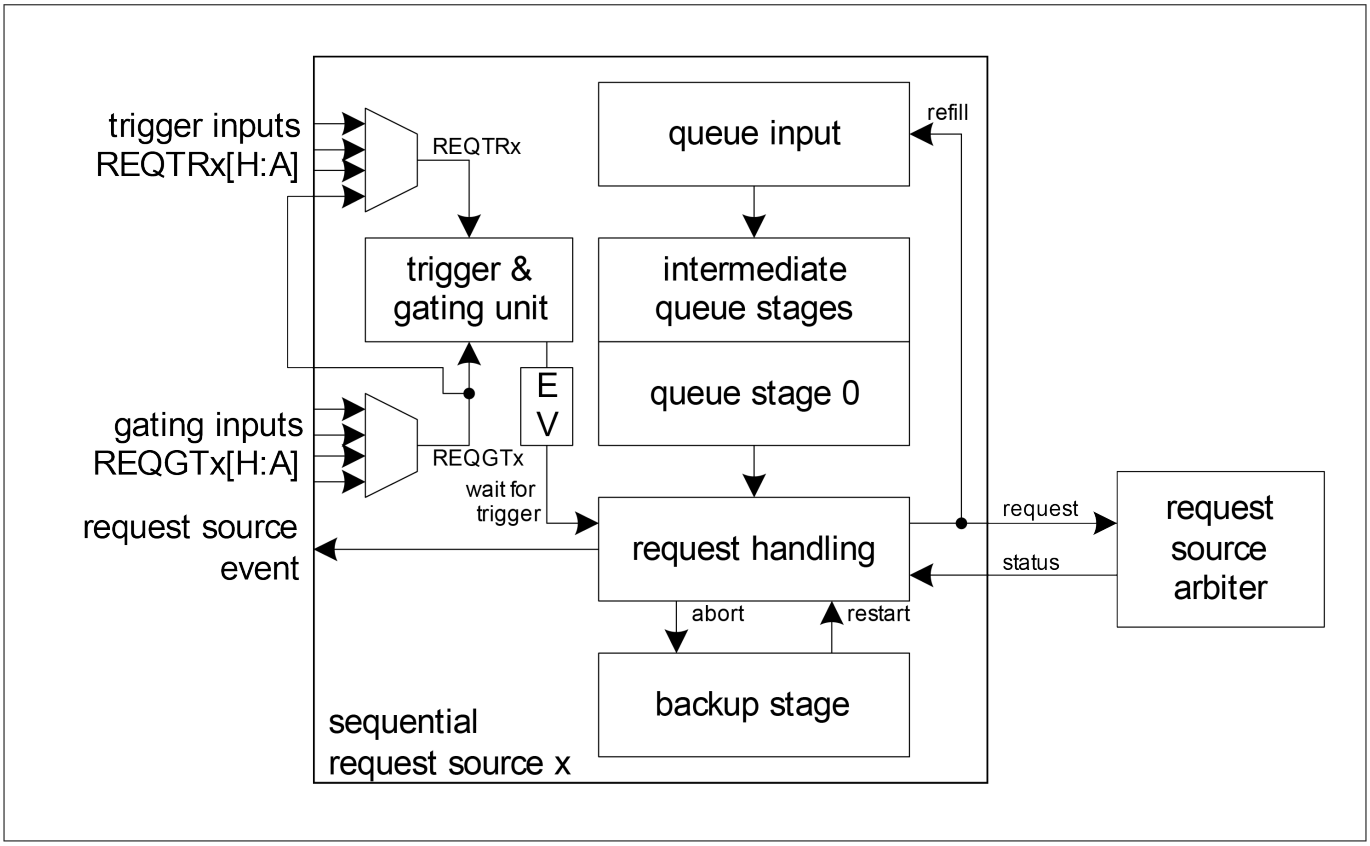


Figure 175 Queued Request Source

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).[10)]
- 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, for example, from a timer or an input pin.
- Generate a software trigger event by setting GxQMR0.TREV = 1.
- 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 one of 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 176. 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.

10 If PDOUT signals from the ERU are used, initialize the ERU accordingly before enabling the gate inputs to avoid un expected signal transitions.

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).

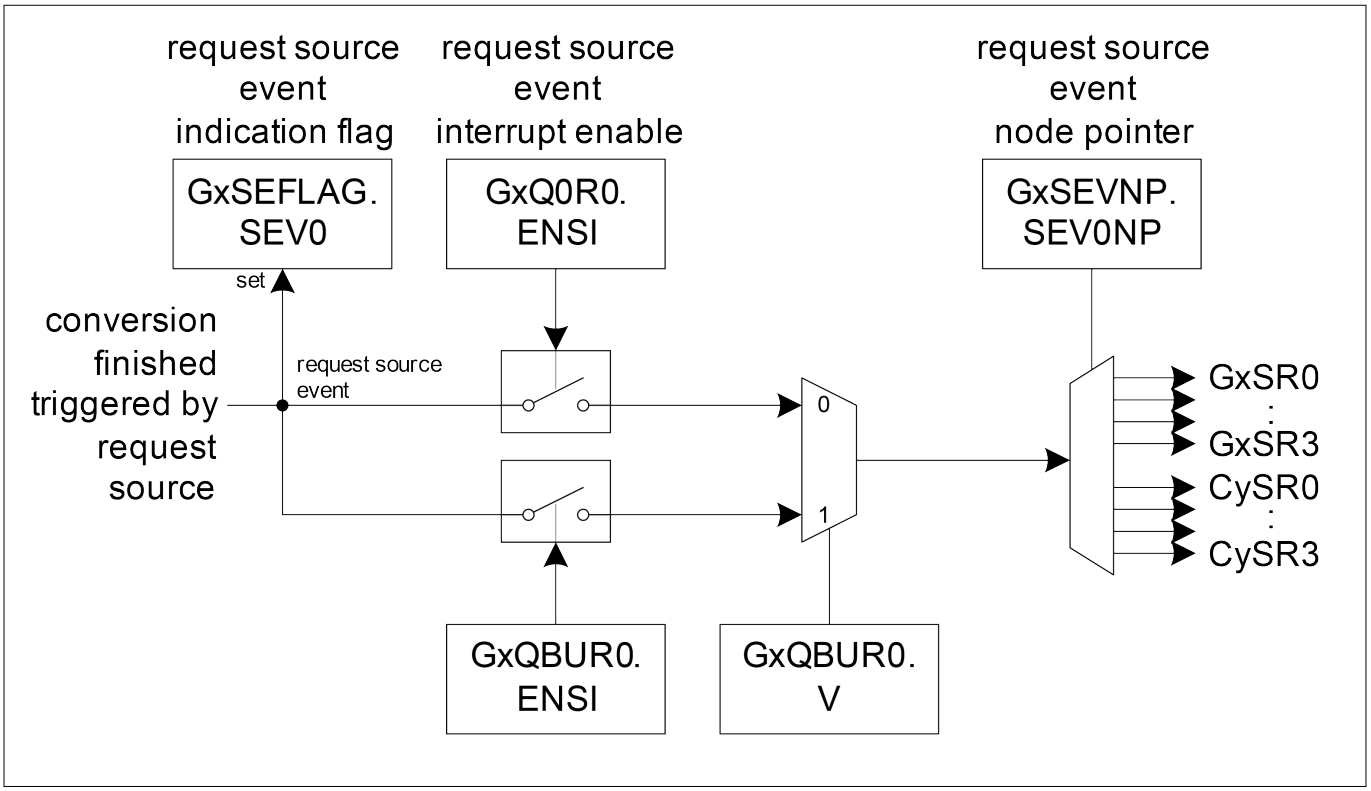


Figure 176 Interrupt Generation of a Queued Request Source

18.6.2 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.

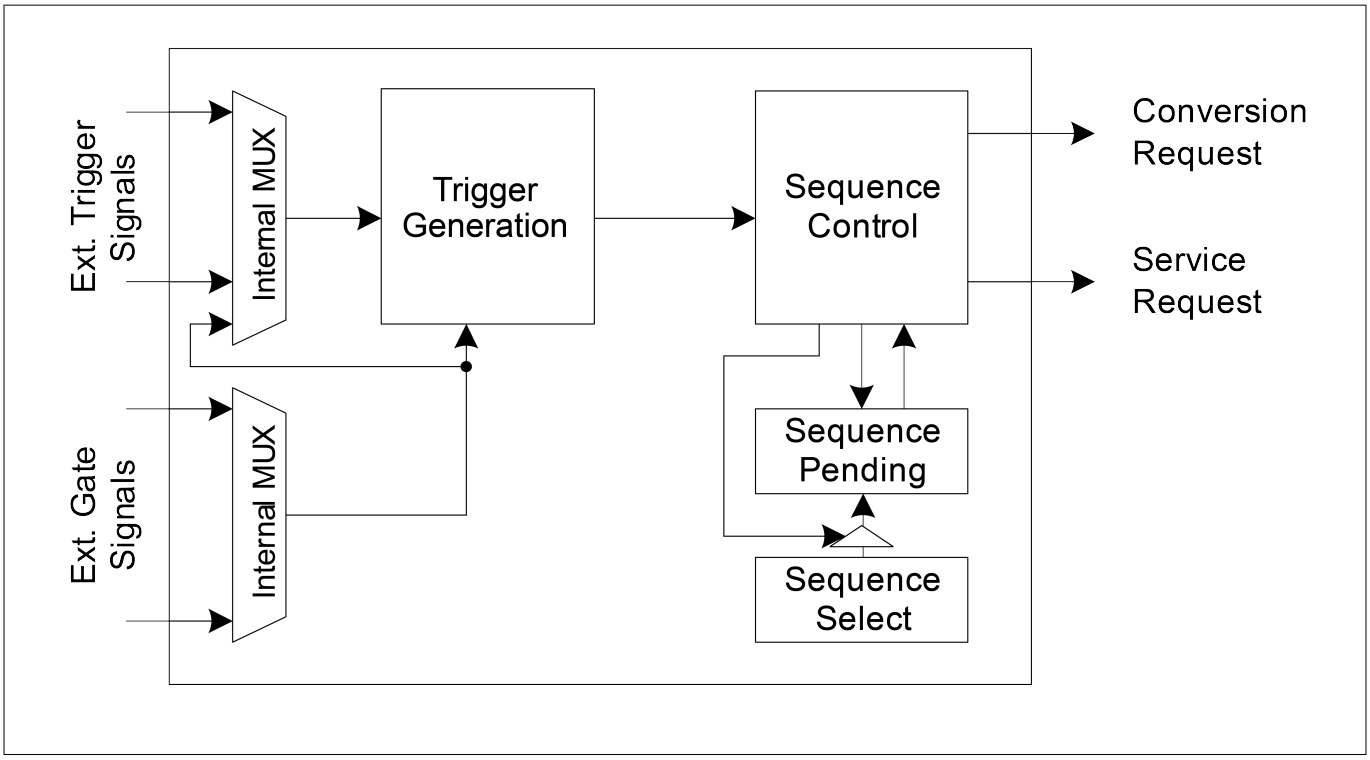


Figure 177 Scan Request Source

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.[11)]
- 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.

11 If PDOUT signals from the ERU are used, initialize the ERU accordingly before enabling the gate inputs to avoid un expected signal transitions.

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 previous 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, for example, 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, for example, 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 one of 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.
- 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 Interrupt 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).[12)]

18.7 Request Source Arbitration

The request source arbiter regularly polls the request sources of the respective group, 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.

12 Please refer to Service Request Generation on page 464.

Figure 19-12 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 PSC1Mx, 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 group (always handled with the highest priority in a synchronization slave group).

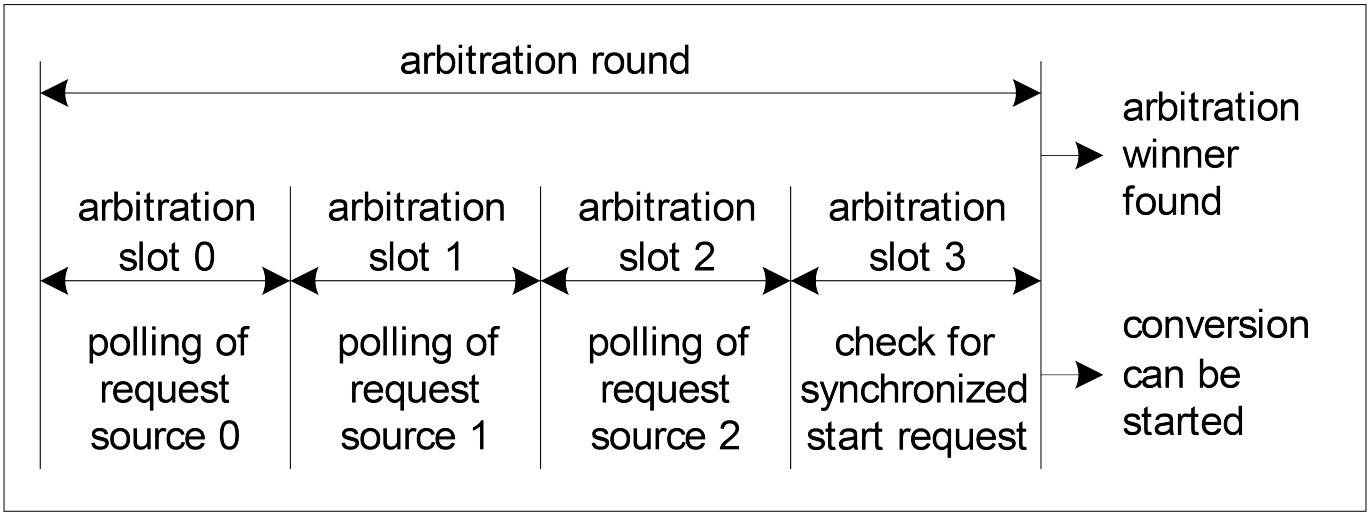


Figure 178 Arbitration Round with 4 Arbitration Slots

18.7.1 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 PSC1Mx, 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 $t_{ARB} = 4 \times t_{Slot}$.

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 (for example, 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.[13)]

13 For more information, please refer to Synchronization of Conversions on page 458

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 groups.

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.

18.7.2 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 179:

    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 179:

    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 Wait-for-Read Mode), 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.*

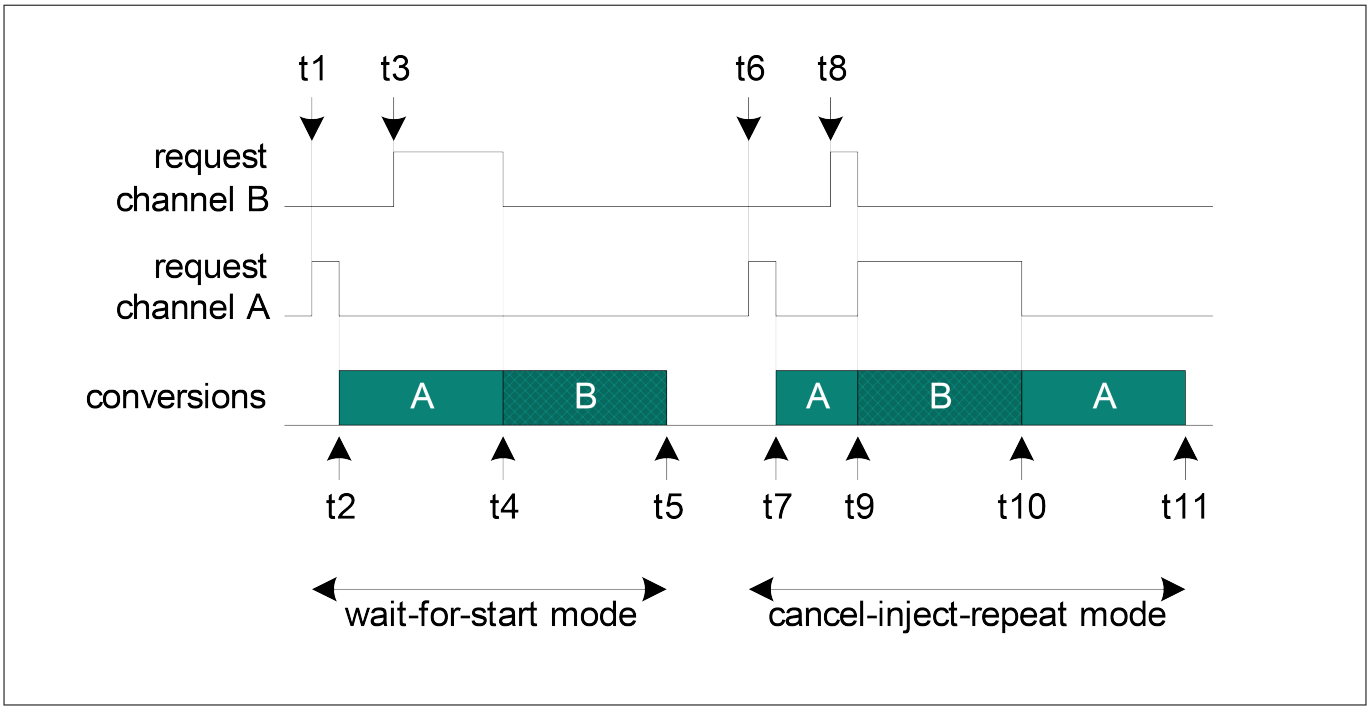


Figure 179 Conversion Start Modes

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

18.8 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 ground can be selected
- **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 channel-specific, selected by bitfield RESREG in register G0CHCTRy (y = 0 - 7) etc.
- **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 186 on Page 450.
- 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 defined at channels CH0 and/or CH1 to other analog input channels.

18.8.1 Channel Parameters

Each analog input channel is configured by its associated channel control register.

***Note**: For the safety feature "Broken Wire Detection", refer to* Broken Wire Detection*.*

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 GxCHCTRy.

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 GxCHCTRy.

18.8.2 Alias Feature

The Alias Feature redirects conversion requests defined at channels CH0 and/or CH1 to other analog input channels (pins). This feature can be used to trigger conversions of the same input pin 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 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 180), 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 180 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.

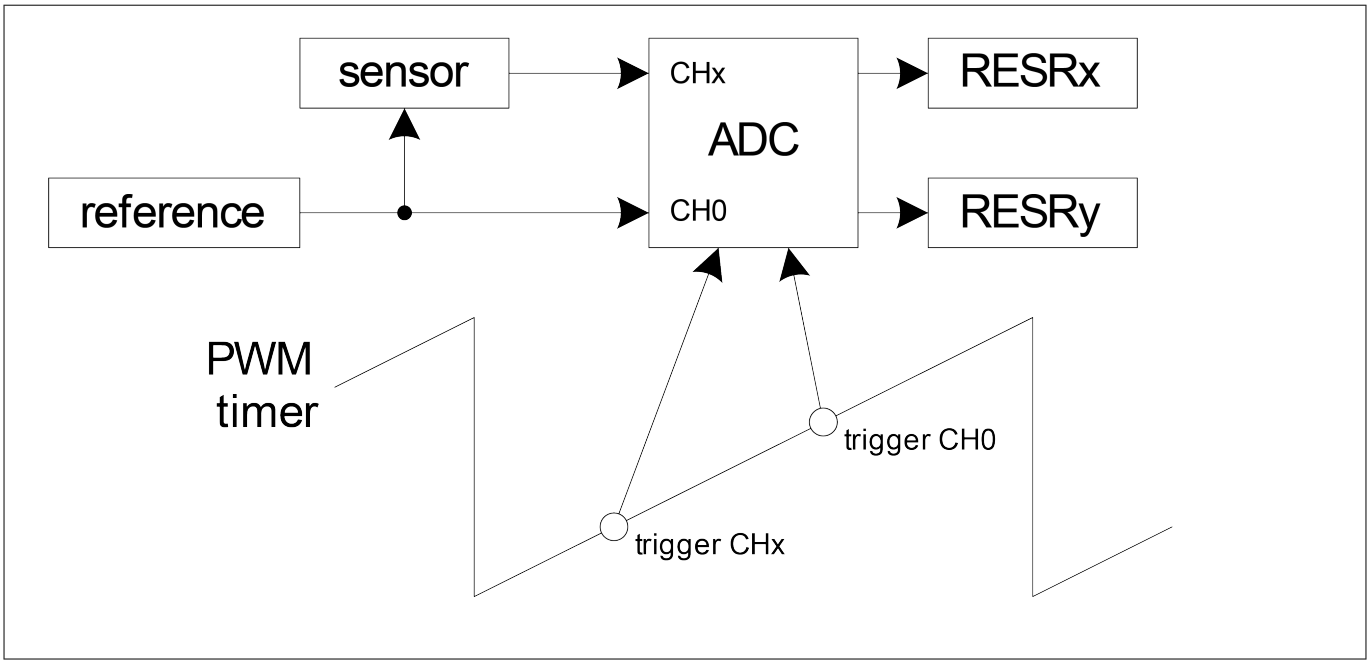


Figure 180 Alias Feature

18.8.3 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 (Conversion Timing).

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 converters can operate with and without calibration steps.*

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.

In Fast Compare Mode, the result registers provide the compare value while the bitfields of local GxBOUND registers define positive and negative delta values.

18.8.4 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 181).

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).

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.

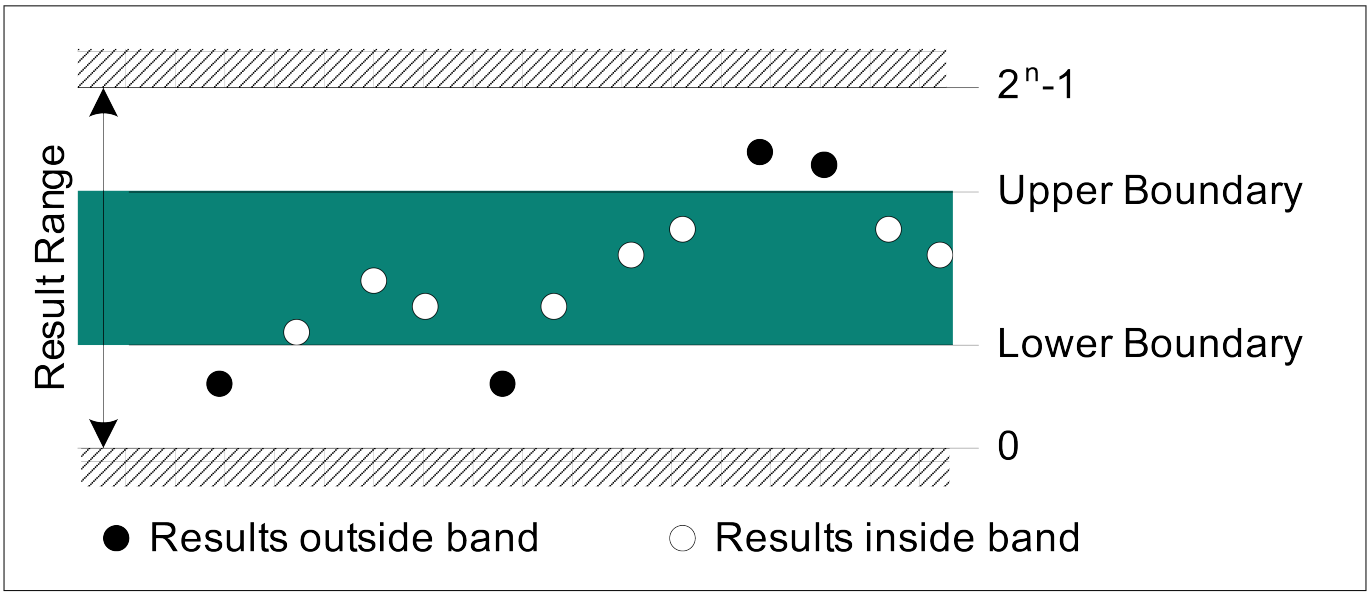


Figure 181 Result Monitoring through Limit Checking

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 (for example, 0000_H to find the maximum and $FFFF_H$ to find the minimum).

18.8.5 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 182).

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)

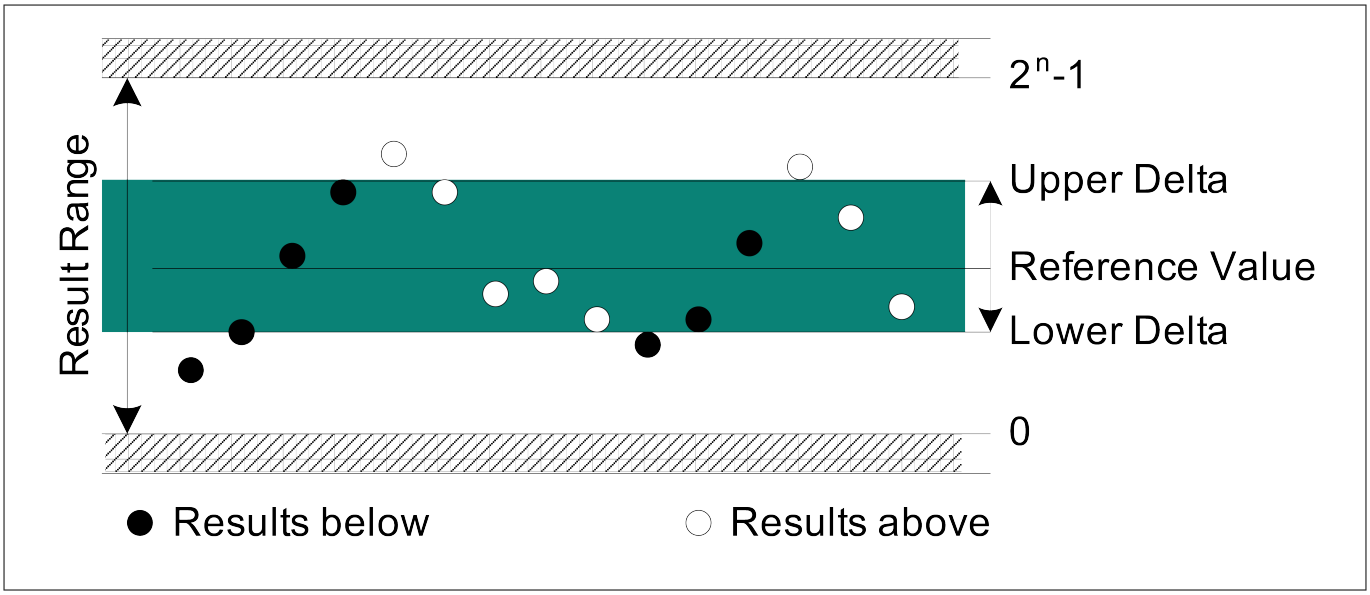


Figure 182 Result Monitoring through Compare with Hysteresis

18.8.6 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 when the conversion result is above the defined band, and will be cleared 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 inputs of a position interface.

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.*

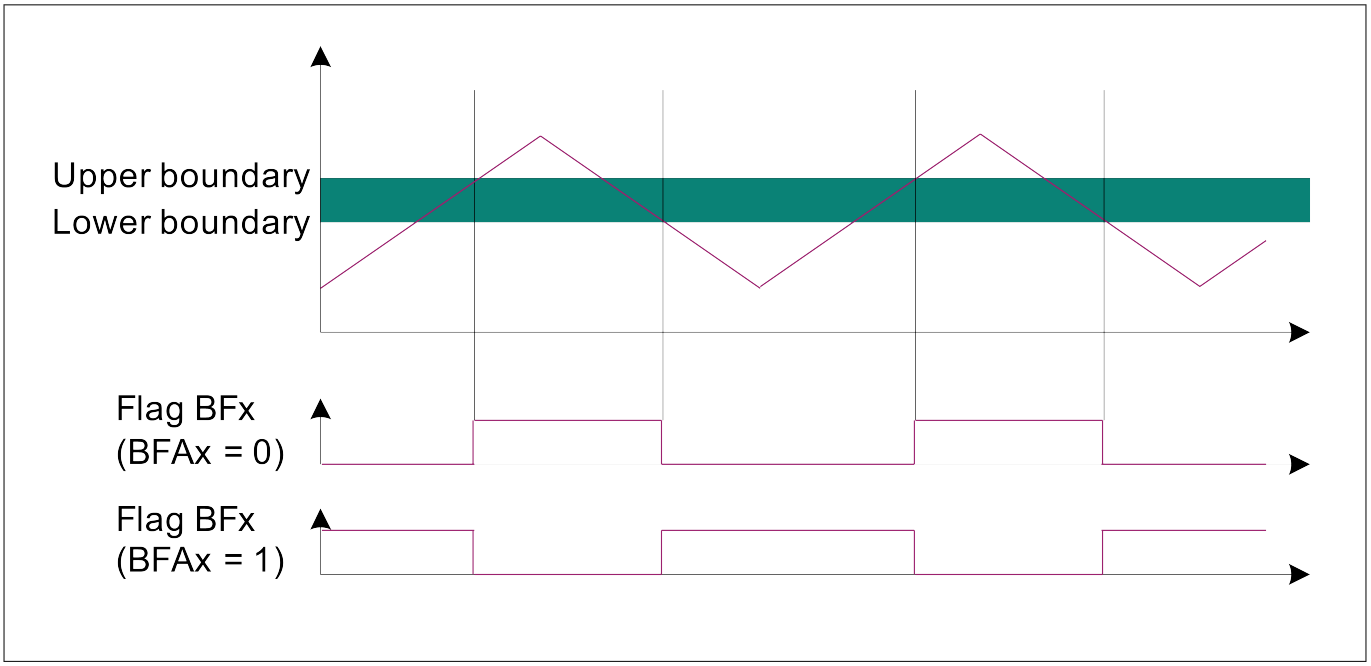


Figure 183 Boundary Flag Switching

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.*

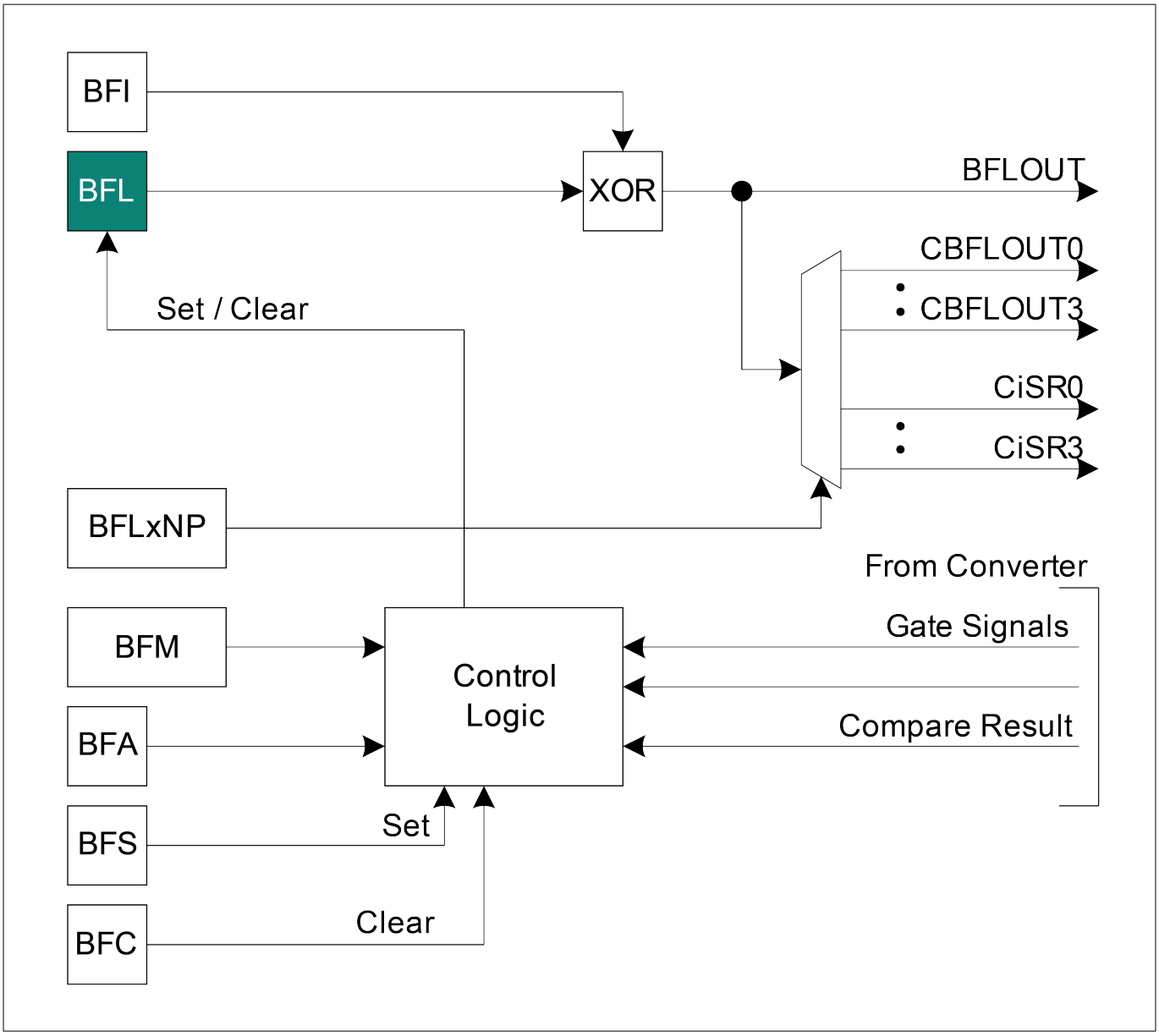


Figure 184 Boundary Flag Control

A boundary flag BFLy is assigned to result register GxRESy and thus to an arbitrary channel.

18.9 Conversion Scheduling

The converter clusters of the VADC combine one high-speed converter with several sample&hold units which can operate in parallel.

The PSC1Mx comprises 1 cluster with 2 S&H units each.

Since all S&H units use one converter, the actual conversion must be associated with certain S&H units. An 8-stage stepper continuously scans the conversion requests of the S&H units of a cluster and starts a conversion if one of them is found active.

Each of the 8 steps can be assigned to one of the available groups (via bitfield KSELx in register SHS0_STEPCFG) and can be enabled to take part in the round-robin scheme. The default programming after reset establishes a linear round-robin scheme that enables and scans all interfaces one after the other. In this case, all interfaces are handled in the same way. The other steps are disabled.

By selecting certain groups multiple times within the stepper sequence the associated S&H units can be scanned more often. In particular in accelerated timing mode this leads to a higher conversion rate on these kernels. This additional functionality makes the enhanced performance of the SHADC available to the application.

Example:

Default: 0 - 1 (2) (3) (4) (5) (6) (7) [2 S&H units: steps 2-7 disabled]

Boost1: 0 - 1 - 1 (3) (4) (5) (6) (7) [steps 3-7 disabled]

***Note**: The default configuration of the stepper provides an upward-compatible operation.*

Empty steps, i.e. steps where the associated S&H unit has no conversion request, are disregarded and the stepper proceeds with the next enabled step.

If all steps are empty the stepper enters its idle state until a conversion request or a calibration request (see Calibration) is activated.

General Timing Behavior

Conversion results are returned to the digital logic after the time interval defined by the parameters fADCI, sample time, conversion width, etc. This compatible timing mode provides the same behavior as predecessor modules. This supports existing applications that are ported to the PSC1Mx.

To exploit the performance of the high-speed converter, an accelerated timing mode is enabled by setting bit AT in register TIMCFG. In this case, each result value is returned as soon as it is converted. Combined with boosting certain groups (see above) this can deliver a high conversion rate.

Also the sample time can be reduced (bitfield SST) to achieve higher conversion rates.

18.10 Conversion Timing

The total time required for a conversion comprises the time from the start of the sample phase[14)] until the availability of the result.

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.

***Note**: In compatible timing mode, conversion timing is defined in the same way as known from predecesor modules. Existing applications find a smooth upgrade path.*

18.10.1 Timing Definition

The actual response time of a conversion depends on the position of the conversion within the cluster's conversion round

The conversion timing depends on the following factors (partly user-definable):

- The converter clock frequency, where $f_{SH} = f_{CONV}/(DIVS+1)$
- The selected sample time, where $t_S = SST \times t_{ADC}$ (short sample time)

14 The time from the trigger event that requests the corresponding conversion until the start of the sample phase depends on the arbitration and can, therefore, only be determined when the system setup is known.

- The result width N (8/10/12 bits)
- The post-calibration time, if selected (after a conversion round)
- The position of the conversion within the stepper sequence
- 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 formula:

$t_{CN} = SST \times t_{ADC} + (4 \times t_{SH})$[15] $+ (N + 8) \times t_{SH} + (5 \times t_{ADC})$[16]

A conversion round comprises 2 conversions (for 2 sample&hold units) and a calibration step. Calibration steps are executed after each conversion round (up to $12 \times t_{SH}$)[17], so the maximum complete conversion round including all 2 S&H units will take:

$t_{CR} = SST \times t_{ADC} + 4 \times t_{SH} + (N+8) \times t_{SH} + (N+8) \times t_{SH} + 5 \times t_{ADC} + 12 \times t_{SH}$[17]

Timing Examples

System assumptions:

f_{CONV} = 32 MHz, DIVS = 0, i.e. $_{tSH} = t_{CONV}$ = 31.25 ns,

f_{ADC} = 32 MHz, i.e. t_{ADC} = 31.25 ns,

SST = 3, i.e. t_S = 94 ns

According to the given formula the following minimum conversion times can be achieved:

(The numbers in parentheses apply for the 2nd conversion of a round)

12-bit conversion:

$t_{CN12} = 3 \times t_{ADC} + 4 \times t_{SH} + 20 \times t_{SH} + 5 \times t_{ADC}$ = 32 × 31.25 ns = 1 000 ns (1 625 ns)

10-bit conversion:

$t_{CN10} = 3 \times t_{ADC} + 4 \times t_{SH} + 18 \times t_{SH} + 5 \times t_{ADC}$ = 30 × 31.25 ns = 937.5 ns (1 500 ns)

An average conversion round including all 2 S&H units and calibration will take up to:

t_{CR} = 1 625 ns + 10.5 × 31.25 ns = 1 954 ns (12-bit conversion)[18]

The maximum time for an isolated conversion will take up to:

t_{C1} = 1 000 ns + 12 × 31.25 ns = 1 375 ns (12-bit conversion)[19]

48-MHz Clock Signal

System assumptions:

f_{CONV} = 48 MHz, DIVS = 0, i.e. $t_{SH} = t_{CONV}$ = 20.84 ns,

f_{ADC} = 48 MHz, i.e. t_{ADC} = 20.84 ns,

SST = 5, i.e. t_S = 104 ns

According to the given formula the following minimum conversion times can be achieved:

(The numbers in parentheses apply for the 2nd conversion of a round)

12-bit conversion:

$t_{CN12} = 3 \times t_{ADC} + 4 \times t_{SH} + 20 \times t_{SH} + 5 \times t_{ADC}$ = 32 × 20.84 ns = 667 ns (1 084 ns)

10-bit conversion:

$t_{CN10} = 3 \times t_{ADC} + 4 \times t_{SH} + 18 \times t_{SH} + 5 \times t_{ADC}$ = 30 × 20.84 ns = 625 ns (1 000 ns)

15 These clock cycles are required between the end of the sample phase and the beginning of the conversion. They can be excluded if this phase occurs during another conversion.

16 These clock cycles are required to transport the converted result value to the digital result register. They only need to be counted when computing the total conversion performance of a cluster.

17 Offset and gain calibration steps are executed alternating, where gain calibration takes more time than offset calibration (9 cycles).

18 10.5 calibration clock cycles are the average resulting from alternating offset and gain calibration steps.

19 12 calibration clock cycles are the maximum resulting from a gain calibration step.

An average conversion round including all 2 S&H units and calibration will take up to:
t_{CR} = 1 084 ns + 10.5 × 20.84 ns = 1 303 ns (12-bit conversion)[18)]
The maximum time for an isolated conversion will take up to:
t_{C1} = 667 ns + 12 × 20.84 ns = 917 ns (12-bit conversion)[19)]

18.10.2 Compatible Timing Mode

The conversion timing depends on the following factors (partly user-definable):

- The ADC conversion clock frequency, where $f_{ADCI} = f_{ADC}/(DIVA+1)$[20)]
- The selected sample time, where $t_S = (2 + STC) \times t_{ADCI}$[21)]
- (STC = additional sample time, see also Table Sample Time Coding)
- The result width N (8/10/12 bits)
- The post-calibration time PC, if selected (PC = 2, otherwise 0)
- 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 formula:
$t_{CN} = (2 + STC + N + PC) \times t_{ADCI} + 2 \times t_{ADC}$

Timing Examples Compatible Mode

According to the given formula the following minimum conversion times can be achieved:
System assumptions for 12-bit calibrated conversions:
f_{ADC} = 32 MHz i.e. t_{ADC} = 31.25 ns, DIVA = 3[20)], f_{ADCI} = 8 MHz i.e. t_{ADCI} = 125 ns
$t_{CN12C} = (2 + 12 + 2) \times t_{ADCI} + 2 \times t_{ADC}$ = 16 × 125 ns + 2 × 31.25 ns = 2 063 ns
System assumptions for 12-bit uncalibrated conversions:
f_{ADC} = 32 MHz i.e. t_{ADC} = 31.25 ns, DIVA = 3[20)], fADCI = 8 MHz i.e. tADCI = 125 ns
$t_{CN12U} = (2 + 12) \times t_{ADCI} + 2 \times t_{ADC}$ = 14 × 125 ns + 2 × 31.25 ns = 1 813 ns
System assumptions for 10-bit conversions:
f_{ADC} = 32 MHz i.e. t_{ADC} = 31.25 ns, DIVA = 3[20)], f_{ADCI} = 8 MHz i.e. t_{ADCI} = 125 ns
$t_{CN10} = (2 + 10) \times t_{ADCI} + 2 \times t_{ADC}$ = 12 × 125 ns + 2 × 31.25 ns = 1 563 ns

48-MHz Clock Signal

System assumptions for 12-bit calibrated conversions:
f_{ADC} = 48 MHz i.e. t_{ADC} = 20.84 ns, DIVA = 3[20)], f_{ADCI} = 12 MHz i.e. t_{ADCI} = 83.3 ns $t_{CN12C} = (2 + 12 + 2) \times t_{ADCI} + 2 \times t_{ADC}$ = 16 × 83.3 ns + 2 × 20.84 ns = 1 375 ns System assumptions for 12-bit uncalibrated conversions: f_{ADC} = 48 MHz i.e. t_{ADC} = 20.84 ns, DIVA = 3[20)], f_{ADCI} = 12 MHz i.e. t_{ADCI} = 83.3 ns $t_{CN12U} = (2 + 12) \times t_{ADCI} + 2 \times t_{ADC}$ = 14 × 83.3 ns + 2 × 20.84 ns = 1 209 ns System assumptions for 10-bit conversions:
f_{ADC} = 48 MHz i.e. t_{ADC} = 20.84 ns, DIVA = 3[20)], f_{ADCI} = 12 MHz i.e. t_{ADCI} = 83.3 ns
$t_{CN10} = (2 + 10) \times t_{ADCI} + 2 \times t_{ADC}$ = 12 × 83.3 ns + 2 × 20.84 ns = 1 042 ns

18.11 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. This supports the subsequent handling of result data by the application software.

18 10.5 calibration clock cycles are the average resulting from alternating offset and gain calibration steps.
19 12 calibration clock cycles are the maximum resulting from a gain calibration step.
20 f_{ADCI} is a virtual frequency which is used to control the conversion timing in compatible timing mode. DIVA must be 3 or higher for conversions in compatible mode.
21 $t_S = (2 + (STC-15) \times 16)) \times t_{ADCI}$ for STC > 0FH.

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 Data Modification)

18.11.1 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.

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

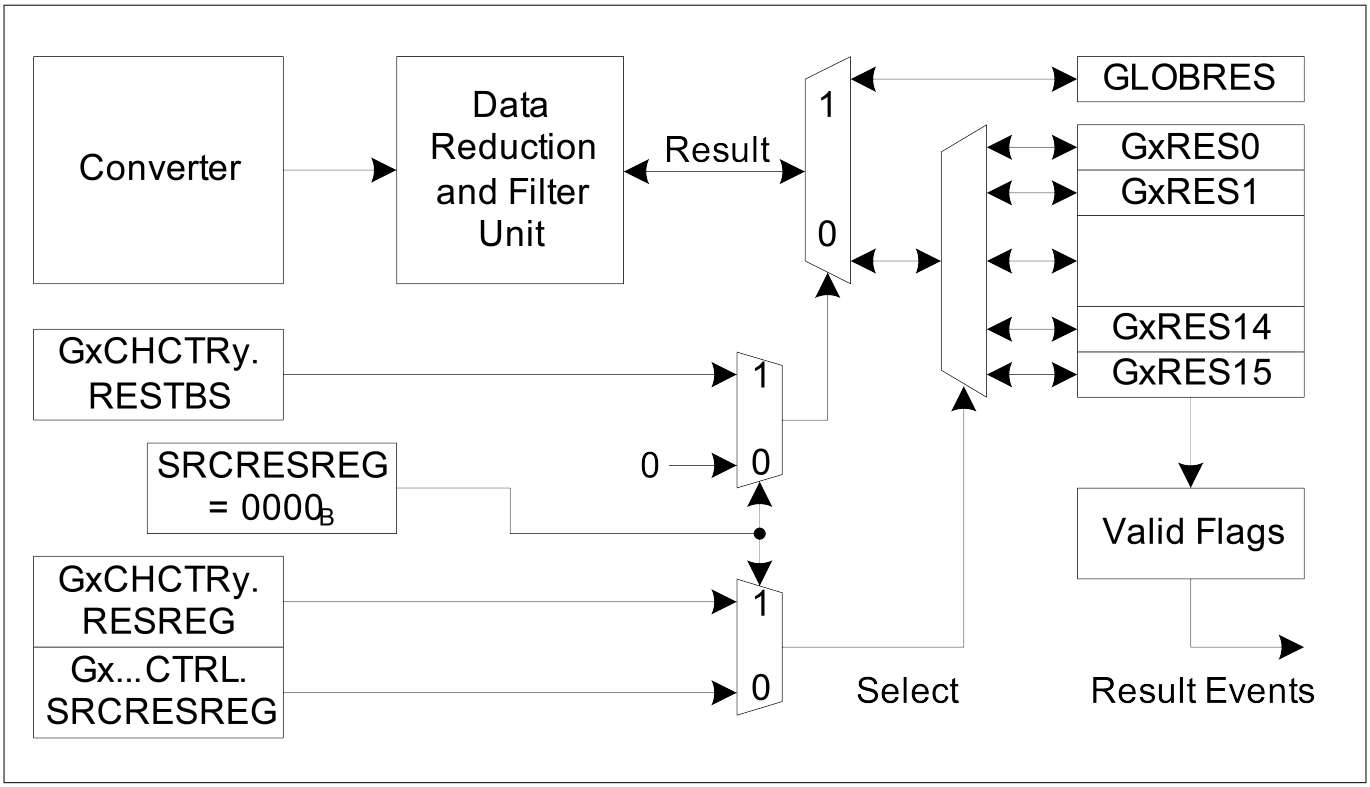


Figure 185 Conversion Result Storage

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.

18.11.2 Data Alignment

The position of a conversion result value within the selected result register depends on 3 configurations (summary in Figure 186):

- 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.

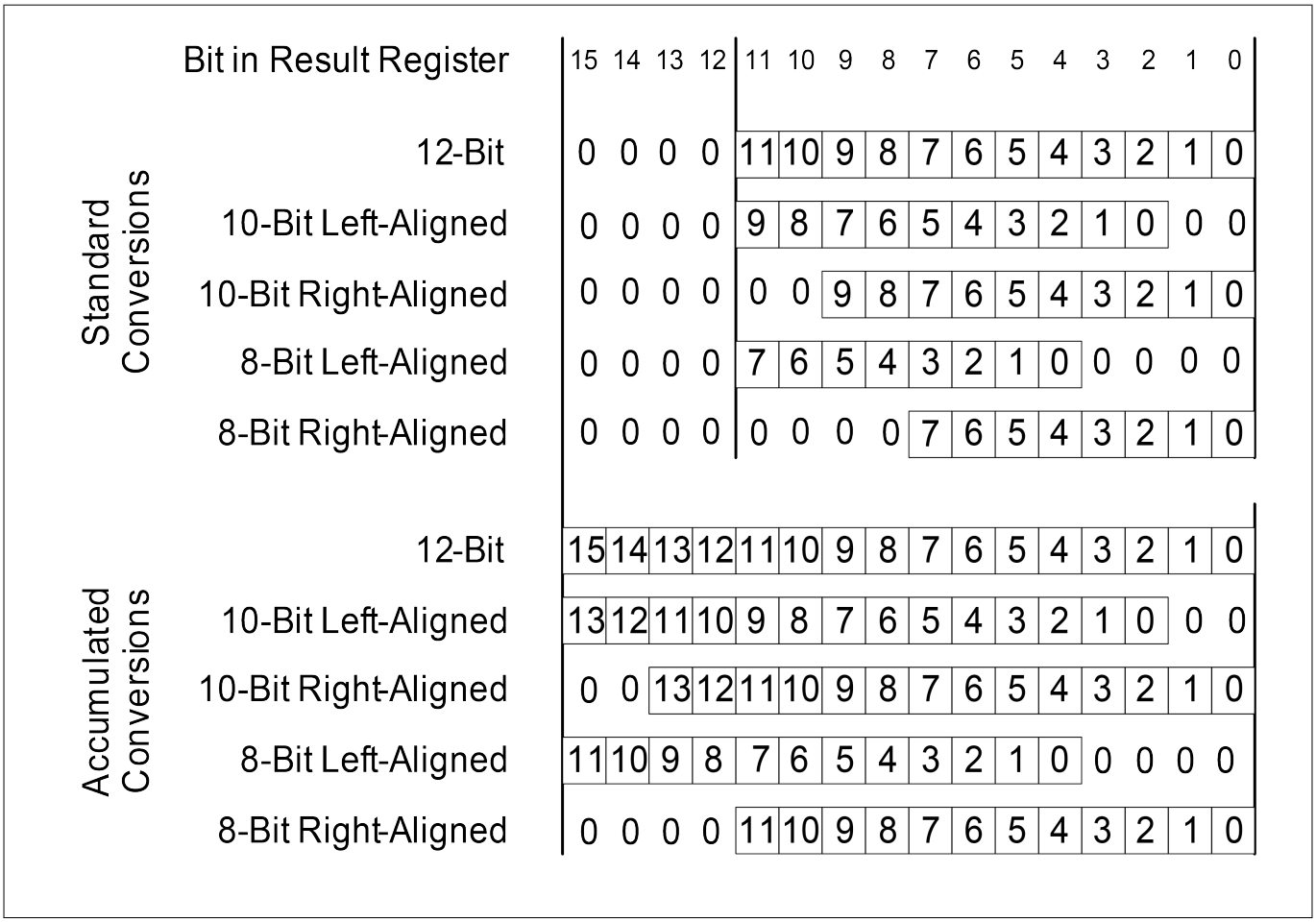


Figure 186 Result Storage Options

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.

18.11.3 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 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[22)].

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 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

22 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.

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 Synchronization of Conversions).*

18.11.4 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 187). 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.

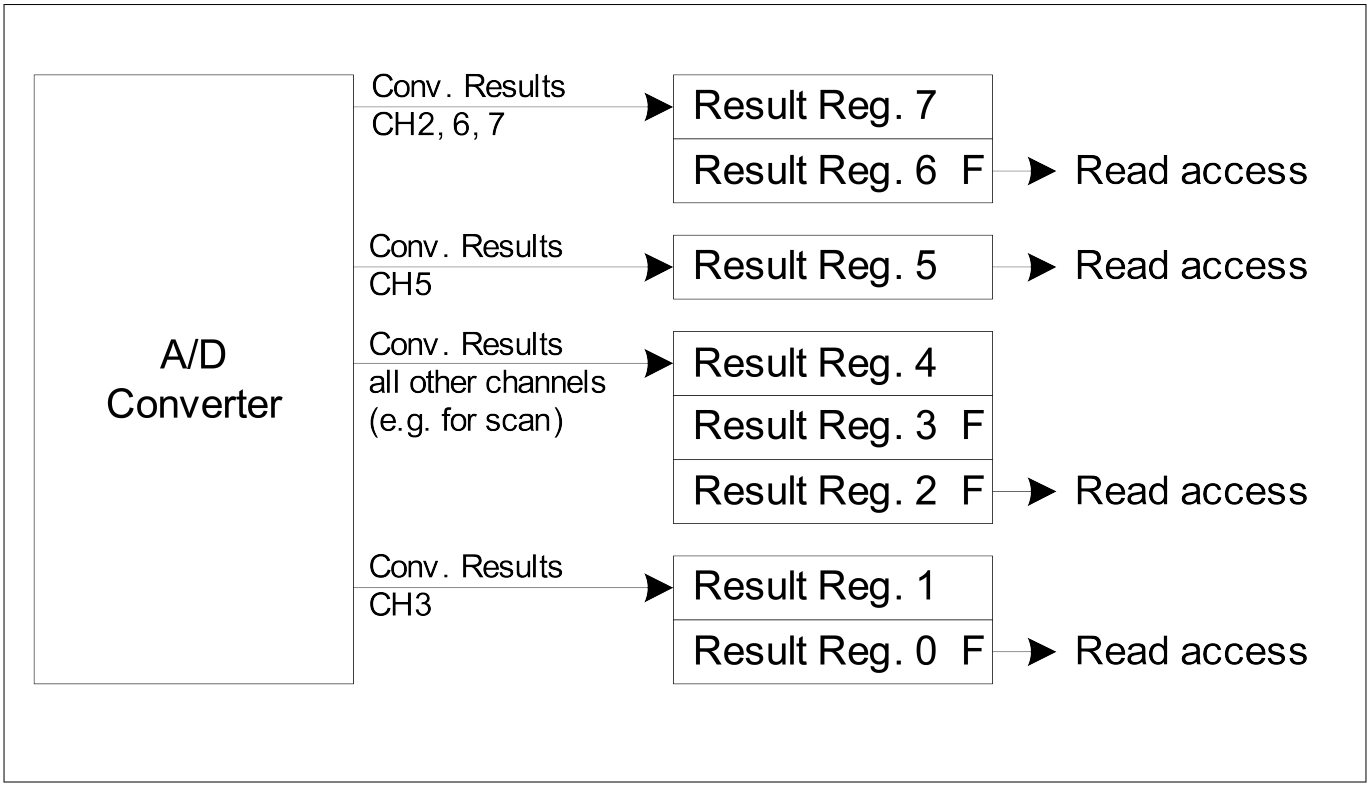


Figure 187 Result FIFO Buffers

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 140 summarizes the required configuration of result registers if they are combined to build result FIFO buffers.

Table 140 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.*

18.11.5 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.

18.11.6 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 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 141 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

(table continues...)

Table 141 (continued) Function of Bitfield DRCTR

DRCTR	Standard Data Reduction Mode (DMM = 00_B)	DRCTR	Result Filtering Mode (DMM = 01_B)[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
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.*

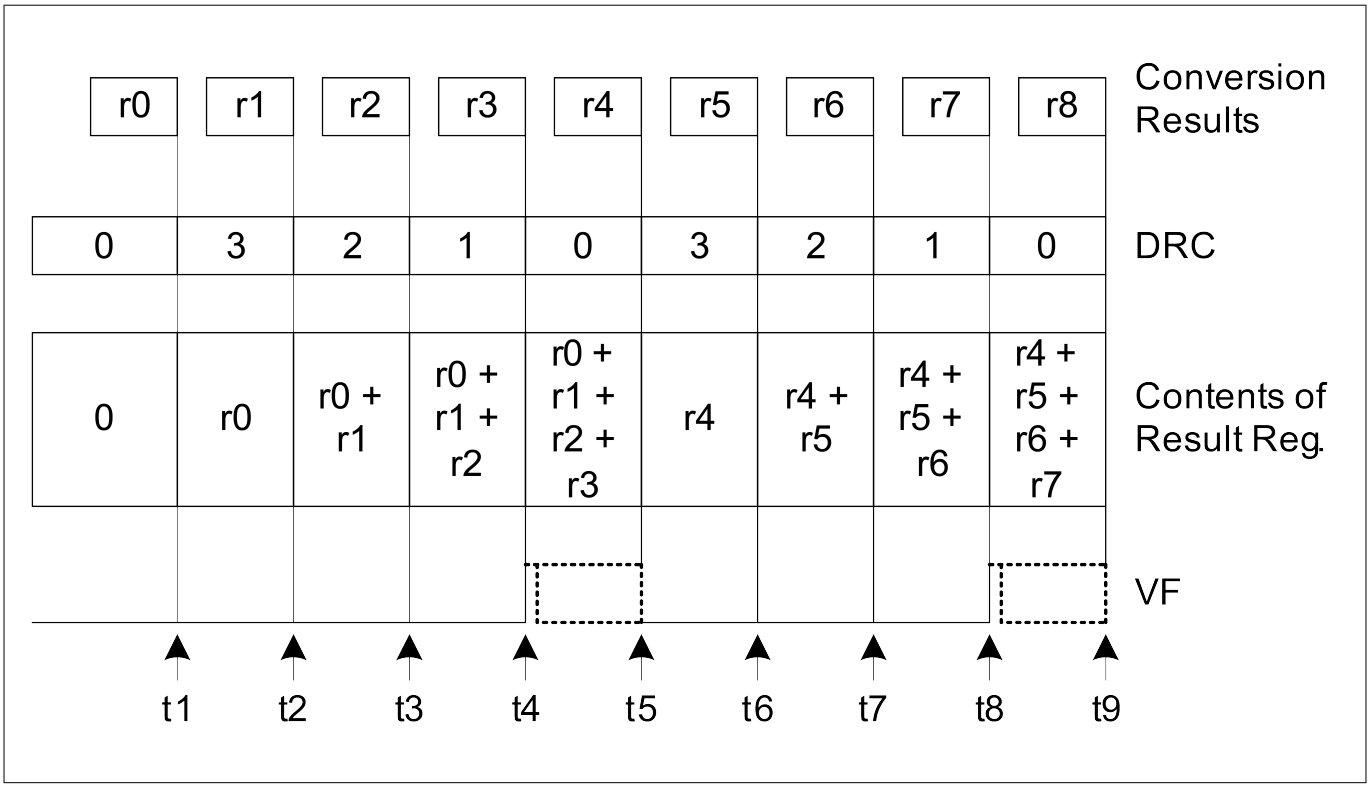


Figure 188 Standard Data Reduction Filter

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 189). 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).

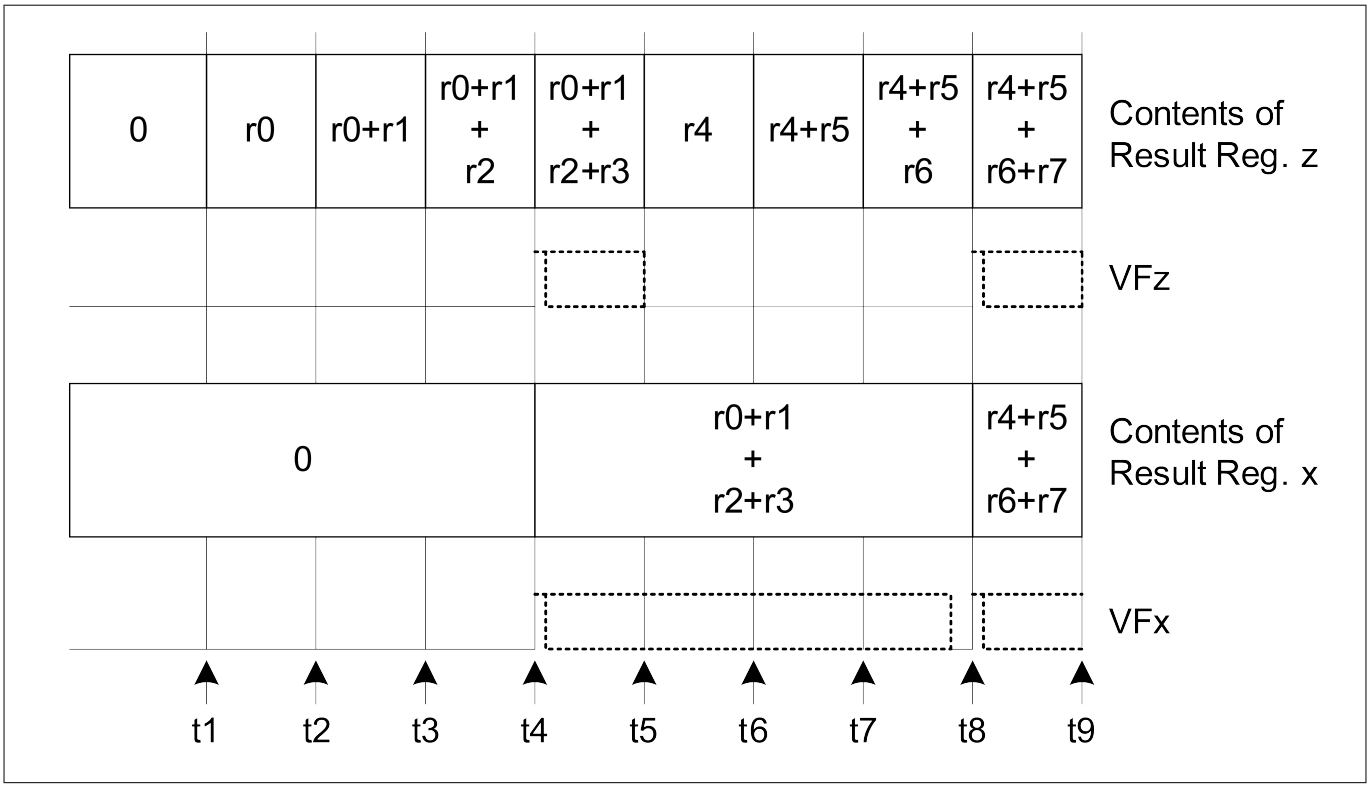


Figure 189 Standard Data Reduction Filter with FIFO Enabled

Finite Impulse Response Filter Mode (FIR)

The FIR filter (see Figure 190) 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 141) 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 it generates a valid result.

***Note**: Conversions for other result registers can be inserted between conversions to be filtered.*

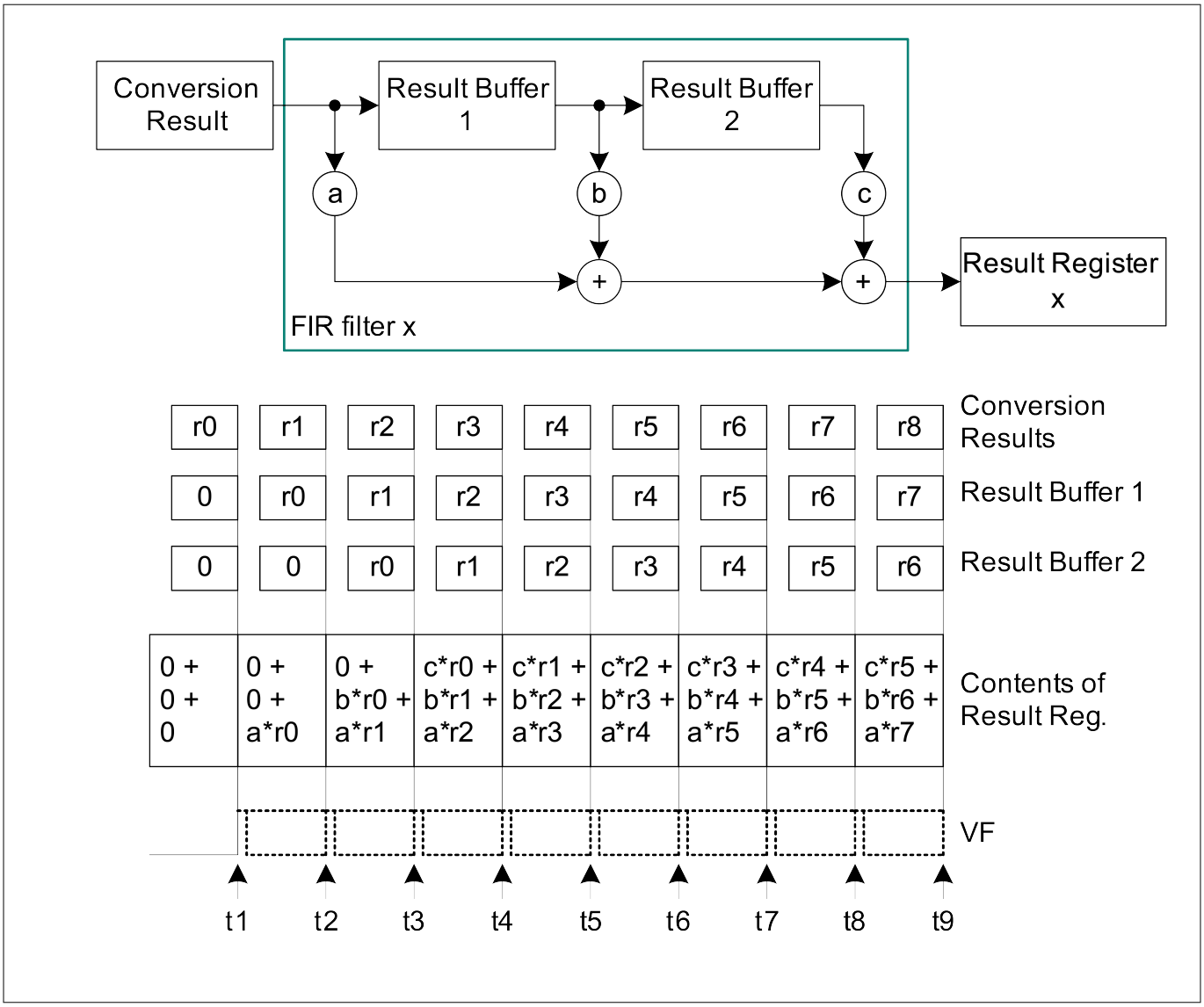


Figure 190 FIR Filter Structure

***Note**: The filter registers are cleared while bitfield DMM ≠ 01_B.*

Infinite Impulse Response Filter Mode (IIR)

The IIR filter (see Figure 191) 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 141) 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.*

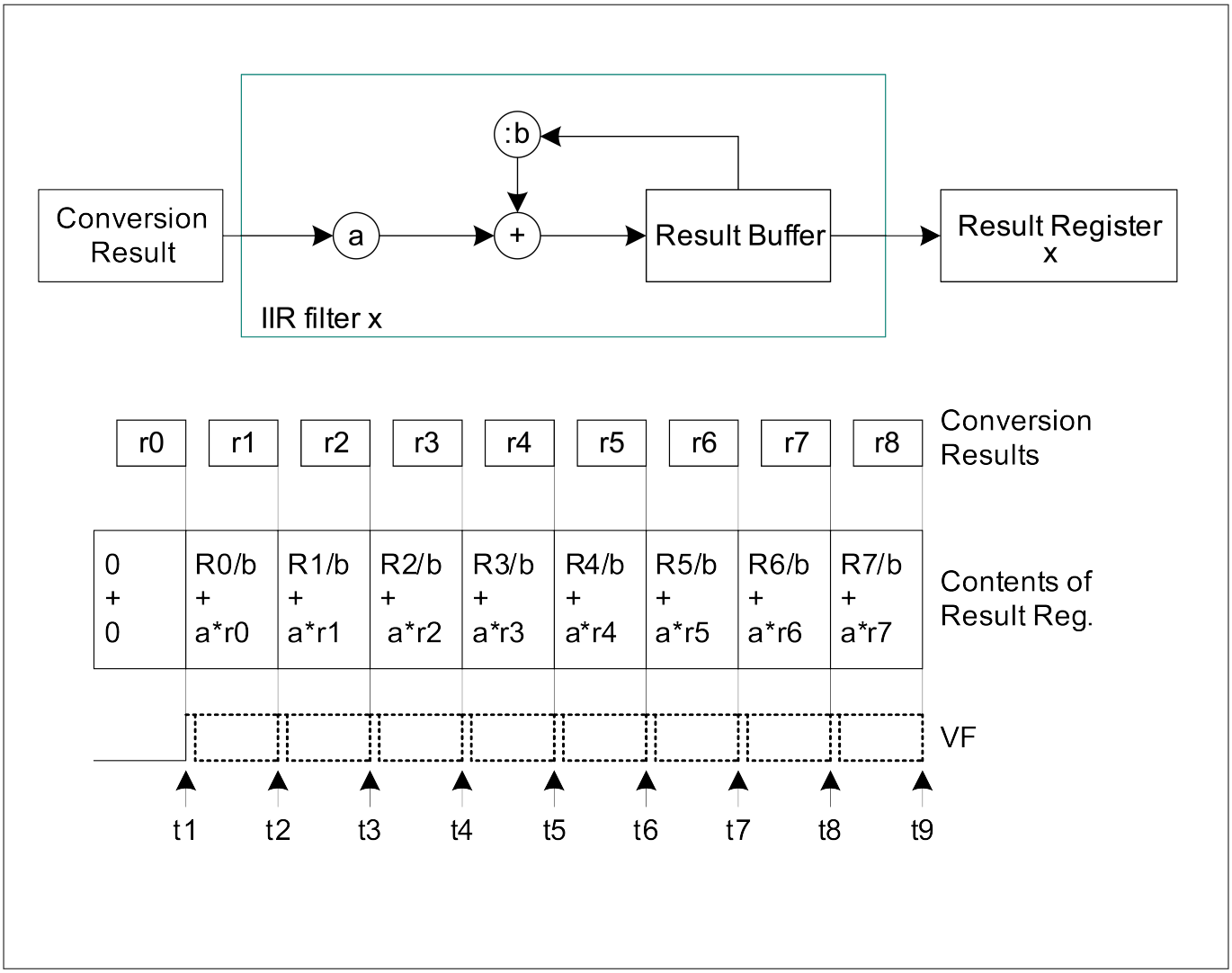


Figure 191 IIR Filter Structure

***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.

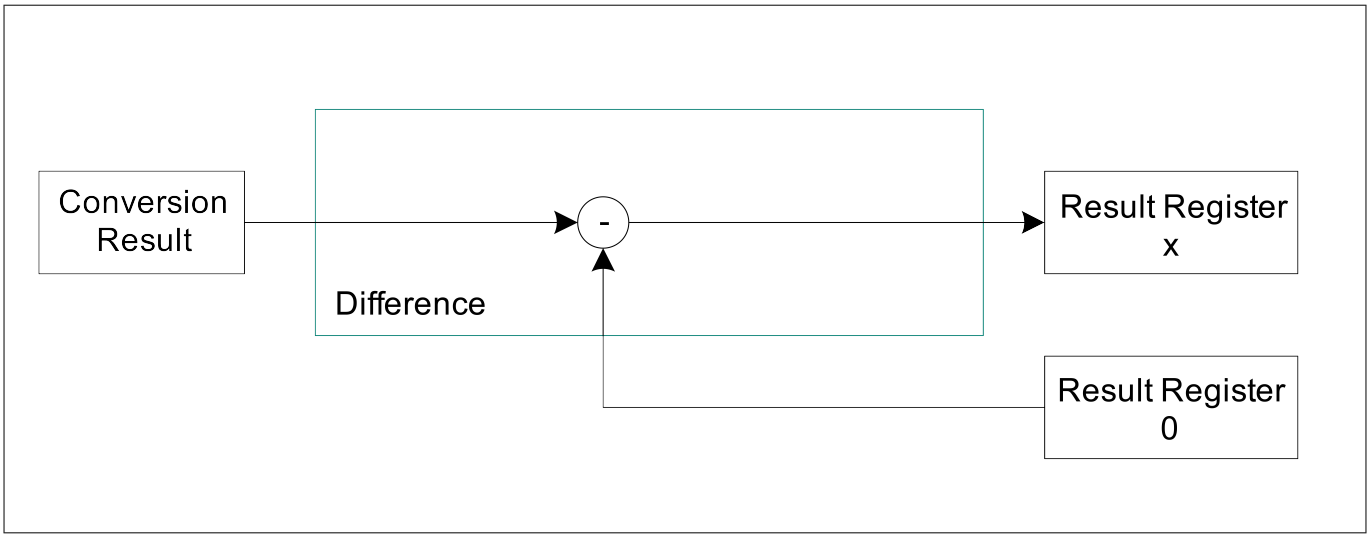


Figure 192 Result Difference

18.12 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 support parallel conversion of channels within a synchronization group[23]. This optimizes, for example, the control of electrical drives.

Equidistant sampling supports conversions in a fixed raster with minimum jitter. This optimizes, for example, filter algorithms or audio applications.

18.12.1 Synchronized Conversions for Parallel Sampling

Several independent ADC kernels[23] implemented in the PSC1Mx 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 193, input channels CH3 of the ADC kernels 0 and 1 are converted synchronously, whereas other input channels do not lead to parallel conversions.

One kernel operates as synchronization master, the other kernel(s) operate(s) as synchronization slave. Each kernel can play either role. Master and slave kernels form a "conversion group" to control parallel sampling:

- **The synchronization master** ADC kernel can request a synchronized conversion of a certain channel (SYNC = 1 in the corresponding channel control register G0CHCTRy (y = 0 - 7) etc.), which is also requested in the connected slave ADC kernel.

  Wait-for-read mode is supported for the master.
- **The synchronization slave** ADC kernel reacts to incoming synchronized conversion requests from the master. While no synchronized conversions are requested, the slave kernel can execute "local" conversions.
  - The slave timing must be configured according to the master timing (ARBRND in register GxARBPR (x = 0 - 1)) to enable parallel conversions.

[23] For a summary, please refer to Synchronization Groups in the PSC1Mx

- 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.

- The arbiter must run permanently (bit GxARBCFG (x = 0 - 1).ARBM = 0) for the synchronization slave. Initialize the slave before the master to have the arbiters run synchronously.
- Once started, a parallel conversion cannot be aborted.

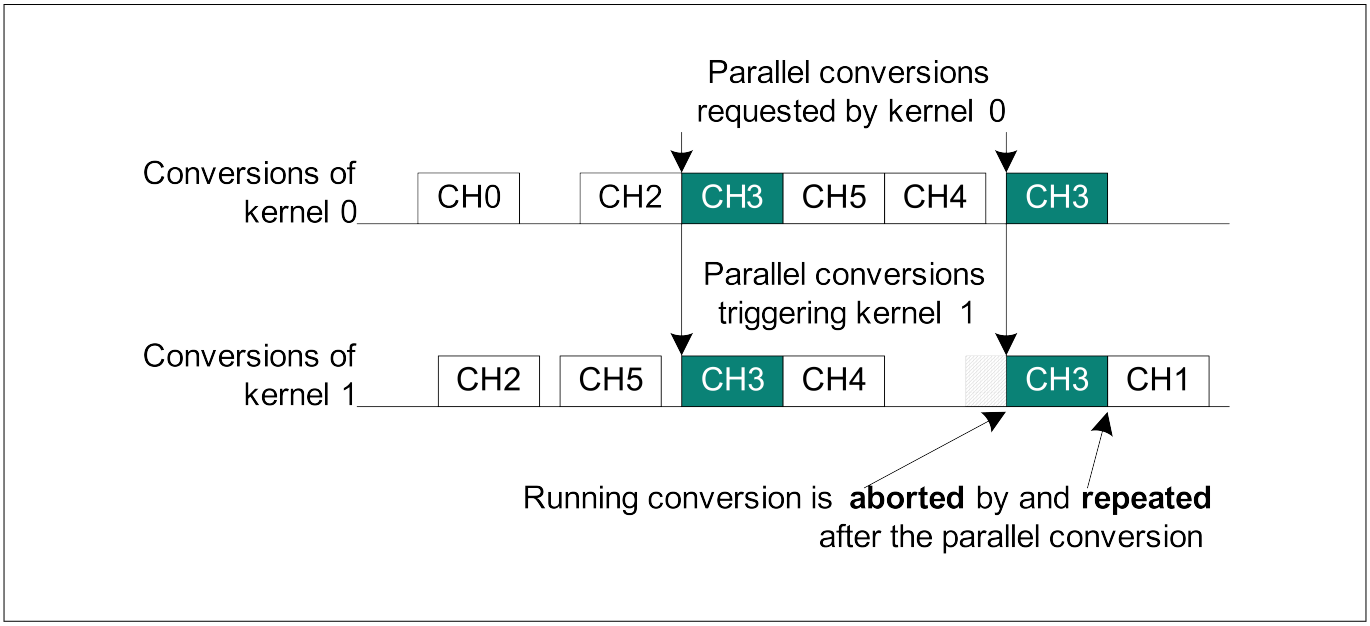


Figure 193 Parallel Conversions

The shown example uses synchronized conversions for channel CH3. The other channel conversions are controlled by their own kernels. ADC0 is the master, ADC1 is the slave.

The synchronization master controls the slave by providing the control information GxARBCFG (x = 0 - 1).ANONS (see Figure 194) and the requested channel number. Bitfields GxSYNCTR (x = 0 - 1).STSEL select the source of the ANON information for the master (00_B) and the slave(s) (01_B/10_B/11_B).

STSEL = 00_B always selects the own ANON information, and is, therefore, meant for the synchronization master or for stand-alone operation. 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 146 on Page 468).

The ready signals indicate when a slave kernel is ready to start the sample phase of a parallel conversion. Bit GxSYNCTR (x = 0 - 1).EVALR1 = 1 enables the control by the ready signal.

***Note**: Synchronized conversions request the same channel number, defined by the master. Using the alias feature (see Alias Feature), analog signals from different input channels can be converted. This is advantageous if, for example, CH0 is used as alternate reference*

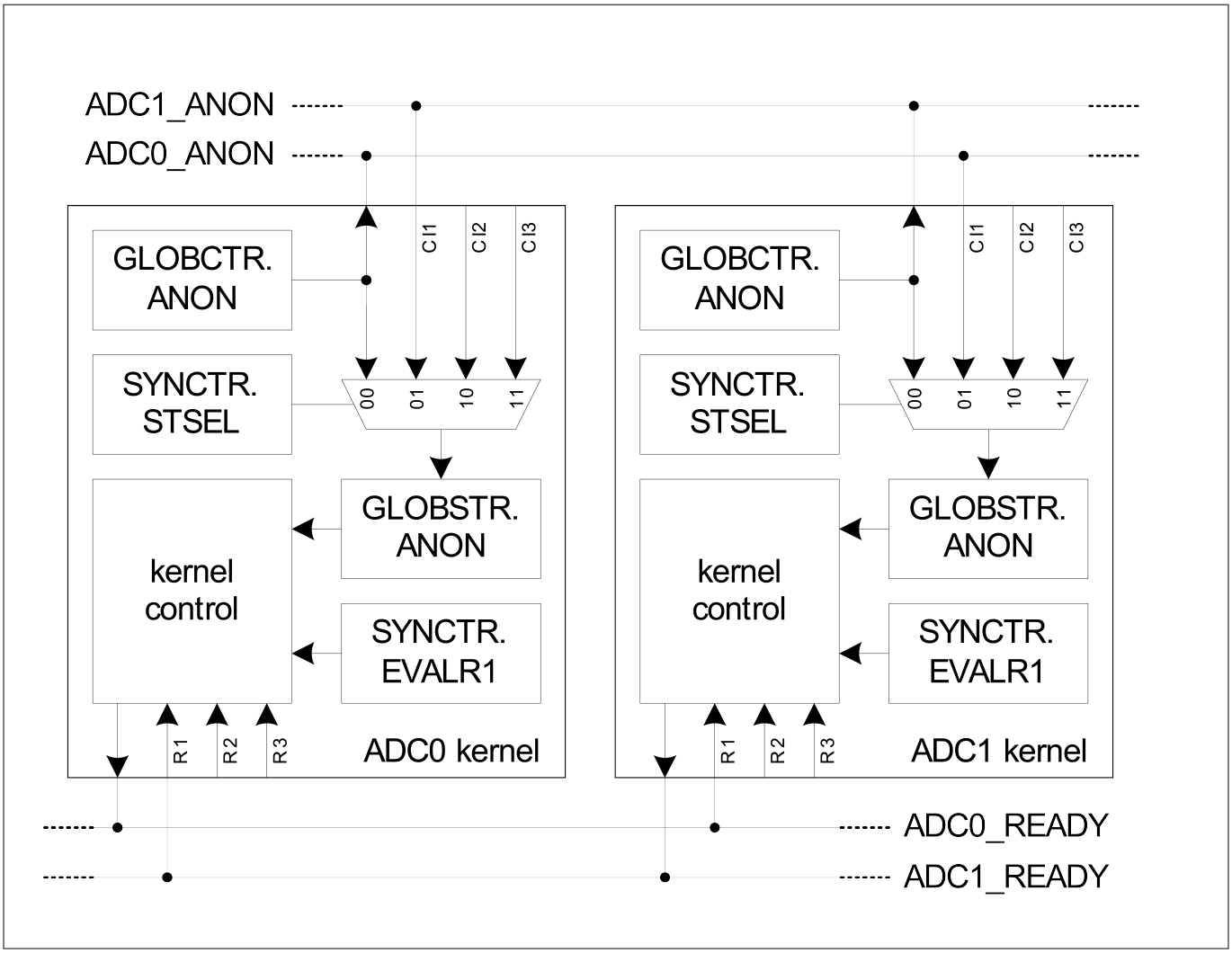


Figure 194 Synchronization via ANON and Ready Signals

18.12.2 Equidistant Sampling

To optimize the input data, for example, for filter or audio applications, conversions can be executed in a fixed timing raster. Conversions for equidistant sampling are triggered by an external signal (for example, 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 195) 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 195) 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.

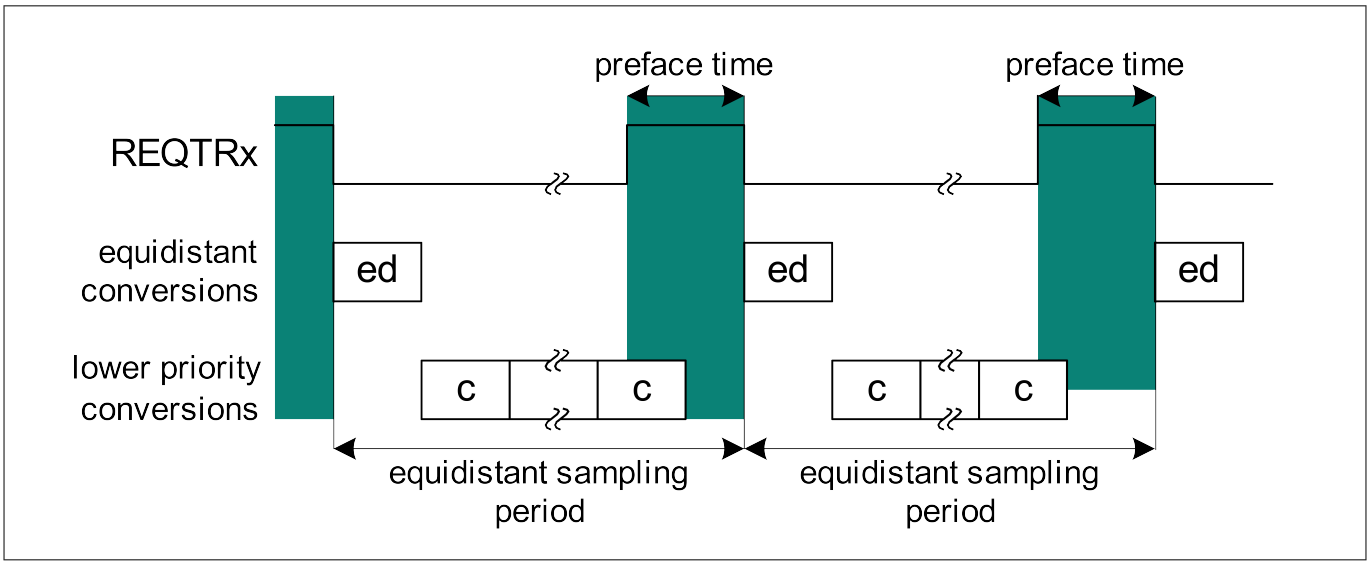


Figure 195 Timer Mode for Equidistant Sampling

18.13 Safety Features

Several test features can be enabled to verify the validity of the analog input signals of an application.

- Broken Wire Detection validates the connection from the sensor to the input pin,
- Multiplexer Diagnostics validates the operation of the internal analog input multiplexer,

18.13.1 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 discharged.

***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.

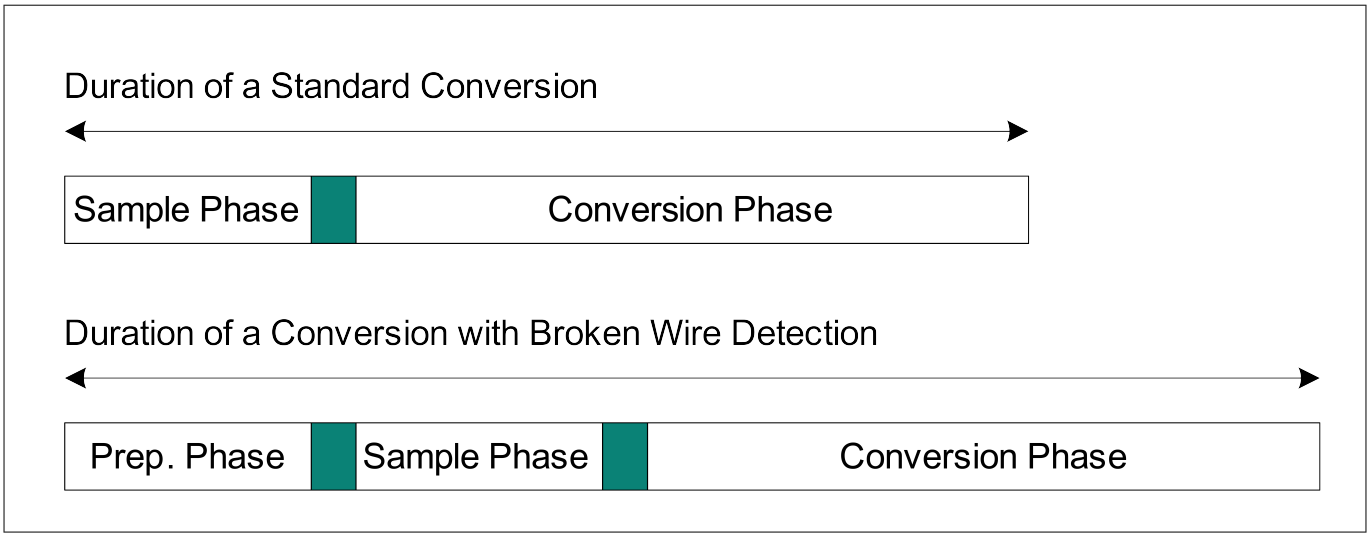


Figure 196 Broken Wire Detection

18.13.2 Multiplexer Diagnostics

Additional test structures (pull-ups, pull-downs) 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 pull devices apply additional loads to the signal path.

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. The pull devices are activated via the standard port control registers.

18.14 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 197 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 197 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 197 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 197 has an external multiplexer connected to channel CH3. The start selection value EMUXSET is assumed as 2.*

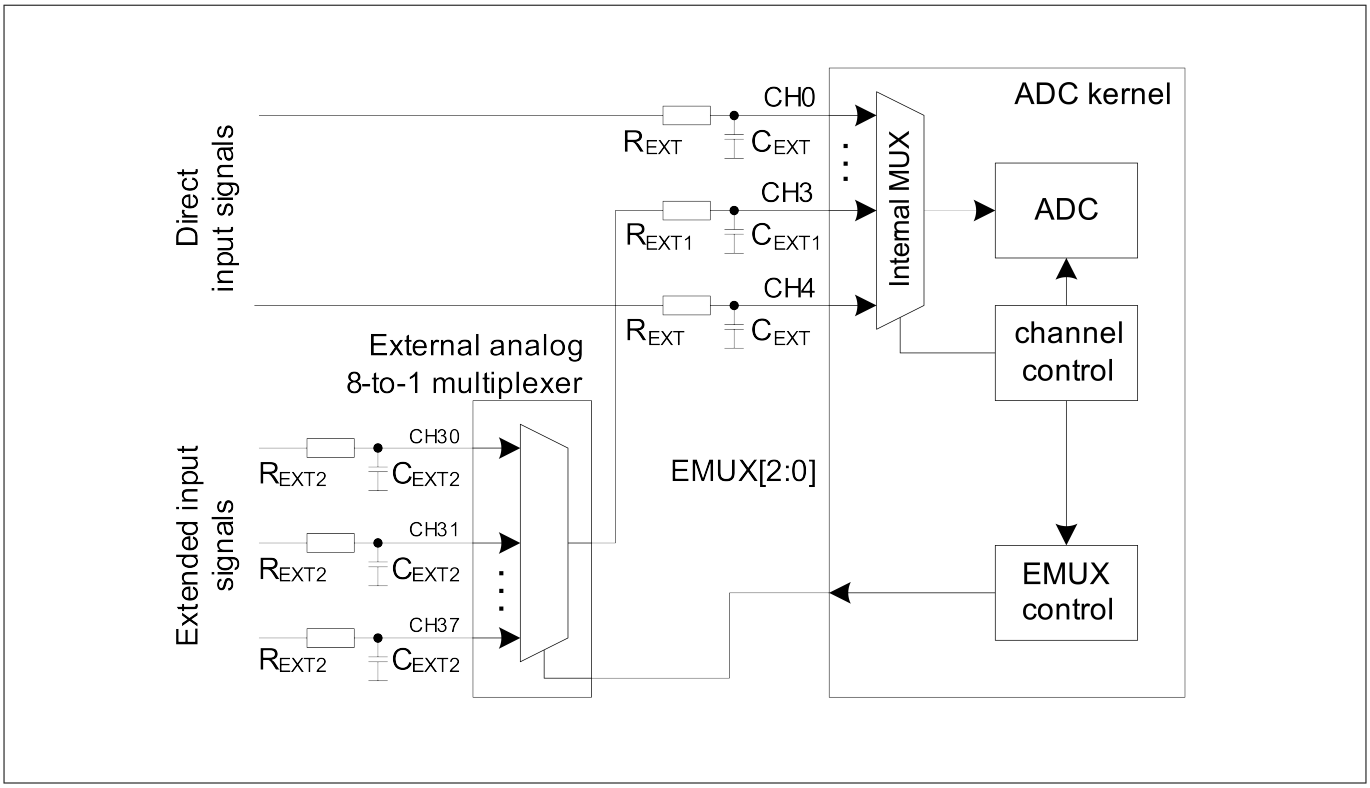


Figure 197 External Analog Multiplexer Example

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 197). 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 197).

***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 19-6 indicates the resulting codes.

Table 142 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

(table continues...)

Table 142 (continued) EMUX Control Signal Coding

Channel	0	1	2	3	4	5	6	7
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.

18.15 Service Request Generation

Each A/D Converter group can activate up to 4 group-specific service request output signals and up to 4 shared service request output signals. 2 group-specific and 2 shared request signals can issue an interrupt (see Table 148 on Page 469).

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 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. Optionally, result events can be generated only at a reduced rate if data reduction is active.

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 that 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 145.*

18.16 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 19-8). The exact register location is obtained by adding the respective register offset to the base address (see Table 19-7) 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
- ###YH means: ###0H + y × 0004_H, for y = 0 - N (depends on register type)
- Z###H means: 4803 4000_H + 0###H, (SHS0)

Table 143 Registers Address Space

Module	Base Address	End Address	Note
VADC0	4803 0000_H	4803 $0BFF_H$	

Table 144 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	
CLC	Clock Control Register	0000_H	U, PV	PV	
OCS	OCDS Control and Status Register	0028_H	U, PV	U, PV	
GLOBCFG	Global Configuration Register	0080_H	U, PV	U, PV	
ACCPROT0	Access Protection Register 0	0088_H	U, PV	U, PV[1)]	
ACCPROT1	Access Protection Register 1	008C	U, PV	U, P[1)] V	
GxARBCFG	Arbitration Configuration Register	$X480_H$	U, PV	U, PV	
GxARBPR	Arbitration Priority Register	$X484_H$	U, PV	U, PV	
GxCHASS	Channel Assignment Register, Group x	$X488_H$	U, PV	U, PV	
GxRRASS	Result Assignment Register, Group x	$X48C_H$	U, SV	U, PV	
GxQCTRL0	Queue 0 Source Control Register, Group x	$X500_H$	U, PV	U, PV	
GxQMR0	Queue 0 Mode Register, Group x	$X504_H$	U, PV	U, PV	
GxQSR0	Queue 0 Status Register, Group x	$X508_H$	U, PV	U, PV	
GxQINR0	Queue 0 Input Register, Group x	$X510_H$	U, PV	U, PV	
GxQ0R0	Queue 0 Register 0, Group x	$X50C_H$	U, PV	U, PV	
GxQBUR0	Queue 0 Backup Register, Group x	$X510_H$	U, PV	U, PV	
GxASCTRL	Autoscan Source Control Register, Group x	$X520_H$	U, PV	U, PV	
GxASMR	Autoscan Source Mode Register, Group x	$X524_H$	U, PV	U, PV	
GxASSEL	Autoscan Source Channel Select Register, Group x	$X528_H$	U, PV	U, PV	

(table continues...)

Table 144 (continued) Registers Overview

Register Short Name	Register Long Name	Offset Addr.	Access Mode		Page Num.
			Read	Write	
GxASPND	Autoscan Source Pending Register, Group x	$X52C_H$	U, PV	U, PV	
BRSCTRL	Background Request Source Control Register	0200_H	U, PV	U, PV	
BRSMR	Background Request Source Mode Register	0204_H	U, PV	U, PV	
BRSSELx	Background Request Source Channel Select Register, Group x	$018Y_H$	U, PV	U, PV	
BRSPNDx	Background Request Source Channel Pending Register, Group x	$01CY_H$	U, PV	U, PV	
GxCHCTRy	Channel x Control Register	$X60Y_H$	U, PV	U, PV	
GxICLASS0	Input Class Register 0, Group x	$X4A0_H$	U, PV	U, PV	
GxICLASS1	Input Class Register 1, Group x	$X4A4_H$	U, PV	U, PV	
GLOBICLASS0	Input Class Register 0, Global	$00A0_H$	U, PV	U, PV	
GLOBICLASS1	Input Class Register 1, Global	$00A4_H$	U, PV	U, PV	
GxALIAS	Alias Register	$X4B0_H$	U, PV	U, PV	
GxBOUND	Boundary Select Register, Group x	$X4B8_H$	U, PV	U, PV	
GLOBBOUND	Global Boundary Select Register	$00B8_H$	U, PV	U, PV	
GxBFL	Boundary Flag Register, Group x	$X4C8_H$	U, PV	U, PV	
GxBFLS	Boundary Flag Software Register, Group x	$X4CC_H$	U, PV	U, PV	
GxBFLC	Boundary Flag Control Register, Group x	$X4D0_H$	U, PV	U, PV	
GxBFLNP	Boundary Flag Node Pointer Register, Group x	$X4D4_H$	U, PV	U, PV	
GxRCRy	Group x Result Control Register y	$X68Y_H$	U, PV	U, PV	
GxRESy	Group x Result Register y	$X70Y_H$	U, PV	U, PV	
GxRESDy	Group x Result Register y (debug view)	$X78Y_H$	U, PV	U, PV	
GLOBRCR	Global Result Control Register	0280_H	U, PV	U, PV	
GLOBRES	Global Result Register	0300_H	U, PV	U, PV	
GLOBRESD	Global Result Register (debug view)	0380_H	U, PV	U, PV	
GxVFR	Valid Flag Register, Group x	$X5F8_H$	U, PV	U, PV	
GxSYNCTR	Synchronization Control Register	$X4C0_H$	U, PV	U, PV	
GxEMUXCTR	External Multiplexer Control Register, Group x	$X5F0_H$	U, PV	U, PV	
EMUXSEL	External Multiplexer Select Register	$03F0_H$	U, PV	U, PV	
GxSEFLAG	Source Event Flag Register, Group x	$X588_H$	U, PV	U, PV	

(table continues...)

Table 144 (continued) Registers Overview

Register Short Name	Register Long Name	Offset Addr.	Access Mode		Page Num.
			Read	Write	
GxCEFLAG	Channel Event Flag Register, Group	$X580_H$	U, PV	U, PV	
GxREFLAG	Result Event Flag Register, Group x	$X584_H$	U, PV	U, PV	
GxSEFCLR	Source Event Flag Clear Register, Group x	$X598_H$	U, PV	U, PV	
GxCEFCLR	Channel Event Flag Clear Register, Group x	$X590_H$	U, PV	U, PV	
GxREFCLR	Result Event Flag Clear Register, Group x	$X594_H$	U, PV	U, PV	
GLOBEFLAG	Global Event Flag Register	$00E0_H$	U, PV	U, PV	
GxSEVNP	Source Event Node Pointer Register, Group x	$X5C0_H$	U, PV	U, PV	
GxCEVNP0	Channel Event Node Pointer Register 0, Group x	$X5A0_H$	U, PV	U, PV	
GxREVNP0	Result Event Node Pointer Register 0, Group x	$X5B0_H$	U, PV	U, PV	
GxREVNP1	Result Event Node Pointer Register 1, Group x	$X5B4_H$	U, PV	U, PV	
GLOBEVNP	Global Event Node Pointer Register	0140_H	U, PV	U, PV	
GxSRACT	Service Request Software Activation Trigger, Group x	$X5C8_H$	U, PV	U, PV	
ID	SHS Module Identification Register	$Z008_H$	U, SV	U, SV	
SHSCFG	SHS Configuration Register	$Z040_H$	U, SV	U, SV	
CALCTR	SHS Calibration Control Register	$Z0BC_H$	U, SV	U, SV	
CALGC0	SHS Gain Calibration Control Reg.	$Z0C0_H$	U, SV	U, SV	
CALGC1	SHS Gain Calibration Control Reg.	$Z0C4_H$	U, SV	U, SV	
GNCTR00	Gain Control Register 0, S&H unit 0	$Z180_H$	U, SV	U, SV	
GNCTR10	Gain Control Register 0, S&H unit 1	$Z190_H$	U, SV	U, SV	
STEPCFG	SHS Stepper Configuration Register	$Z044_H$	U, SV	U, SV	
TIMCFG0	SHS Timing Configuration Register	$Z080_H$	U, SV	U, SV	
TIMCFG1	SHS Timing Configuration Register	$Z084_H$	U, SV	U, SV	
LOOP	SHS SD Loop Control Register	$Z050_H$	U, SV	U, SV	

1) Access to these registers is protected by bit protection scheme described in SCU chapter.

18.17 Interconnects

This section describes the actual implementation of the VADC module into the PSC1Mx, i.e. the incorporation into the microcontroller system.

18.17.1 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 (PSC1Mx).

Each converter group is equipped with a separate sample and hold stage and a dedicated analog input multiplexer.

Table 145 General Converter Configuration in the PSC1Mx

Converter Group	Input Channels	Channels with Alternate GND	12-bit Performance	Common Service Request Group
G0	0 ... 7	8	Calibrated S&H-ADC	C0
G1	0 ... 7	8		C0

Synchronization Groups in the PSC1Mx

The converter kernels in the PSC1Mx can be connected to synchronization groups to achieve parallel conversion of several input channels.

Table 146 summarizes which kernels can be synchronized for parallel conversions.

Table 146 Synchronization Groups in the PSC1Mx

ADC Kernel	Synchr. Group	Master selected by control input CIx[1]	
		CI0[2]	CI1
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.

18.17.2 Analog Module Connections in the PSC1Mx

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 145). A summary of channels enclosing all versions of the PSC1Mx can be found in Table 147.

Table 147 Analog Connections in the PSC1Mx

Signal	Dir.	Source/Destin.	Description
Supply Voltage and Reference Voltage			
VDDM[1]	I	VDD	positive supply voltage
V_{SSM}	I	VSS	negative supply voltage
V_{AREF}	I	VDD	positive analog reference
V_{AGND}	I	VSS	negative analog reference

(table continues...)

Table 147 (continued) Analog Connections in the PSC1Mx

Signal	Dir.	Source/Destin.	Description
Group 0 Analog Inputs			
G0CH0	I	P2.6	analog input channel 0 of group 0
G0CH1	I	P2.8	analog input channel 1 of group 0
G0CH2	I	P2.9	analog input channel 2 of group 0
G0CH3	I	P2.10	analog input channel 3 of group 0
G0CH4	I	P2.11	analog input channel 4 of group 0
G0CH5	I	P2.0	analog input channel 5 of group 0
G0CH6	I	P2.1	analog input channel 6 of group 0
G0CH7	I	P2.2	analog input channel 7 of group 0
Group 1 Analog Inputs			
G1CH0	I	P2.8	analog input channel 0 of group 1
G1CH1	I	P2.7	analog input channel 1 of group 1
G1CH2	I	P2.10	analog input channel 2 of group 1
G1CH3	I	P2.11	analog input channel 3 of group 1
G1CH4	I	P2.9	analog input channel 4 of group 1
G1CH5	I	P2.3	analog input channel 5 of group 1
G1CH6	I	P2.4	analog input channel 6 of group 1
G1CH7	I	P2.5	analog input channel 7 of group 1

1) For the VQFN-40 package, V_{DDM} must not deviate from V_{DDP}

18.17.3 Digital Module Connections in the PSC1Mx

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.

***Note**: The control bitfields for triggers and gates select the corresponding multiplexer input. Values 0000B ... 1111_B select inputs with suffix -A ... -P.*

Table 148 Digital Connections in the PSC1Mx

Signal	Dir.	Source/Destin.	Description
Gate Inputs for Each Group			
GxREQGTA	I	CCU40.ST3	Gating input A (GTSEL = 0000_B)
GxREQGTB	I	CCU40.ST2	Gating input B (GTSEL = 0001_B)
GxREQGTC	I	CCU40.ST1	Gating input C (GTSEL = 0010_B)
GxREQGTD	I	CCU40.ST0	Gating input D (GTSEL = 0011_B)

(table continues...)

Table 148 (continued) Digital Connections in the PSC1Mx

Signal	Dir.	Source/Destin.	Description
GxREQGTE	I	CCU80.ST3A	Gating input E (GTSEL = 0100_B)
GxREQGTF	I	CCU80.ST3	Gating input F (GTSEL = 0101_B)
GxREQGTG	I	0	Gating input G (GTSEL = 0110_B)
GxREQGTH	I	0	Gating input H (GTSEL = 0111_B)
GxREQGTI	I (s)	Reserved	Reserved
GxREQGTJ	I (s)	Reserved	Reserved
GxREQGTK	I (s)	ERU0.PDOUT2	Gating input K (GTSEL = 1010_B)
GxREQGTL	I (s)	ERU0.PDOUT3	Gating input L (GTSEL = 1011_B)
GxREQGTM	I	CCU80.ST0	Gating input M (GTSEL = 1100_B)
GxREQGTN	I	CCU80.ST1	Gating input N (GTSEL = 1101_B)
GxREQGTO	I	ERU0.PDOUT0	Gating input O (GTSEL = 1110_B)
GxREQGTP	I	ERU0.PDOUT1	Gating input P (GTSEL = 1111_B)
GxREQGTSEL	O	GxREQTRP[1)]	Selected gating signal of the respective source
Gate Inputs for Global Background Source			
BGREQGTA	I	CCU40.ST3	Gating input A (GTSEL = 0000_B)
BGREQGTB	I	CCU40.ST2	Gating input B (GTSEL = 0001_B)
BGREQGTC	I	CCU40.ST1	Gating input C (GTSEL = 0010_B)
BGREQGTD	I	CCU40.ST0	Gating input D (GTSEL = 0011_B)
BGREQGTE	I	CCU80.ST3A	Gating input E (GTSEL = 0100_B)
BGREQGTF	I	CCU80.ST3	Gating input F (GTSEL = 0101_B)
BGREQGTG	I	0	Gating input G (GTSEL = 0110_B)
BGREQGTH	I	0	Gating input H (GTSEL = 0111_B)
BGREQGTI	I (s)	reserved	reserved
BGREQGTJ	I (s)	reserved	reserved
BGREQGTK	I (s)	ERU0.PDOUT2	Gating input K (GTSEL = 1010_B)
BGREQGTL	I (s)	ERU0.PDOUT3	Gating input L (GTSEL = 1011_B)
BGREQGTM	I	CCU80.ST0	Gating input M (GTSEL = 1100_B)
BGREQGTN	I	CCU80.ST1	Gating input N (GTSEL = 1101_B)
BGREQGTO	I	ERU0.PDOUT0	Gating input O (GTSEL = 1110_B)
BGREQGTP	I	ERU0.PDOUT1	Gating input P (GTSEL = 1111_B)
BGREQGTSEL	O	BGREQTRP[1)]	Selected gating signal
Trigger Inputs for Each Group			
GxREQTRA	I	CCU40.SR2	Trigger input A (XTSEL = 0000_B)

(table continues...)

Table 148 (continued) Digital Connections in the PSC1Mx

Signal	Dir.	Source/Destin.	Description
GxREQTRB	I	CCU40.SR3	Trigger input B (XTSEL = 0001_B)
GxREQTRC	I	0	Trigger input C (XTSEL = 0010_B)
GxREQTRD	I	0	Trigger input D (XTSEL = 0011_B)
GxREQTRE	I	0	Trigger input E (XTSEL = 0100_B)
G0REQTRF	I	BCCU0.TRIGOU T0	Trigger input F (XTSEL = 0101_B)
G0REQTRF	I	BCCU0.TRIGOU T1	Trigger input F (XTSEL = 0101_B)
GxREQTRG	I	ERU0.IOUT2	Trigger input G (XTSEL = 0110_B)
GxREQTRH	I	ERU0.IOUT3	Trigger input H (XTSEL = 0111_B)
GxREQTRI	I (s)	CCU80.SR2	Trigger input I (XTSEL = 1000_B)
GxREQTRJ	I (s)	CCU80.SR3	Trigger input J (XTSEL = 1001_B)
GxREQTRK	I (s)	0	Trigger input K (XTSEL = 1010_B)
GxREQTRL	I (s)	0	Trigger input L (XTSEL = 1011_B)
GxREQTRM	I	ERU0.IOUT0	Trigger input M (XTSEL = 1100_B)
GxREQTRN	I	ERU0.IOUT1	Trigger input N (XTSEL = 1101_B)
GxREQTRO	I	POSIF0.SR1	Trigger input O (XTSEL = 1110_B)
GxREQTRP	I	GxREQGTSEL[1)]	Extend triggers to selected gating input of the respective source (XTSEL = 1111_B)
GxREQTRSEL	O	-	Selected trigger signal of the respective source
Trigger Inputs for Global Background Source			
BGREQTRA	I	CCU40.SR2	Trigger input A (XTSEL = 0000_B)
BGREQTRB	I	CCU40.SR3	Trigger input B (XTSEL = 0001_B)
BGREQTRC	I	0	Trigger input C (XTSEL = 0010_B)
BGREQTRD	I	0	Trigger input D (XTSEL = 0011_B)
BGREQTRE	I	0	Trigger input E (XTSEL = 0100_B)
BGREQTRF	I	BCCU0.TRIGOUT0	Trigger input F (XTSEL = 0101_B)
BGREQTRG	I	ERU0.IOUT2	Trigger input G (XTSEL = 0110_B)
BGREQTRH	I	ERU0.IOUT3	Trigger input H (XTSEL = 0111_B)
BGREQTRI	I (s)	CCU80.SR2	Trigger input I (XTSEL = 1000_B)
BGREQTRJ	I (s)	CCU80.SR3	Trigger input J (XTSEL = 1001_B)
BGREQTRK	I (s)	0	Trigger input K (XTSEL = 1010_B)
BGREQTRL	I (s)	0	Trigger input L (XTSEL = 1011_B)
BGREQTRM	I	ERU0.IOUT0	Trigger input M (XTSEL = 1100_B)
BGREQTRN	I	ERU0.IOUT1	Trigger input N (XTSEL = 1101_B)
BGREQTRO	I	POSIF0.SR1	Trigger input O (XTSEL = 1110_B)

(table continues...)

Table 148 (continued) Digital Connections in the PSC1Mx

Signal	Dir.	Source/Destin.	Description
BGREQTRP	I	BGREQGTSEL[1)]	Extend triggers to selected gating input of the background source (XTSEL = 1111_B)
BGREQTRSEL	O	-	Selected trigger signal of the background source
System-Internal Connections			
G0SAMPLE	O	-	Indicates the input signal sample phase
G1SAMPLE	O	-	Indicates the input signal sample phase
G0ARBCNT	O	CCU40.IN3G	Outputs a (count) pulse for each arbiter round
G1ARBCNT	O	-	Outputs a (count) pulse for each arbiter round
G0SR0	O	NVIC (17)	Service request 0 of group 0
G0SR1	O	NVIC (18)	Service request 1of group 0
G0SR2	O	-	Service request 2 of group 0
G0SR3	O	-	Service request 3 of group 0
G1SR0	O	NVIC (19)	Service request 0 of group 1
G1SR1	O	NVIC (20)	Service request 1of group 1
G1SR2	O	-	Service request 2 of group 1
G1SR3	O	-	Service request 3 of group 1
C0SR0	O	NVIC (15)	Service request 0 of common block 0
C0SR1	O	NVIC (16)	Service request 1 of common block 0
C0SR2	O	ERU0.OGU02 ERU0.OGU12	Service request 2 of common block 0
C0SR3	O	ERU0.OGU22 ERU0.OGU32	Service request 3 of common block 0
EMUX00	O	P0.4P2.12	Control of external analog multiplexer interface 0
EMUX01	O	P0.3P2.13	
EMUX02	O	P0.2	
EMUX10	O	P0.5	Control of external analog multiplexer interface 1
EMUX11	O	P0.6	
EMUX12	O	P0.7	
CBFLOUT0	O	POSIF0.IN0C	Common boundary flag output 0
CBFLOUT1	O	POSIF0.IN1C	Common boundary flag output 1
CBFLOUT2	O	POSIF0.IN2C	Common boundary flag output 2
CBFLOUT3	O	POSIF0.EWHEA	Common boundary flag output 3
G0BFLOUT0	O	ERU0.0A3	Boundary flag 0 output of group 0
G1BFLOUT0	O	ERU0.0B3	Boundary flag 0 output of group 1

(table continues...)

Table 148 (continued) Digital Connections in the PSC1Mx

Signal	Dir.	Source/Destin.	Description
G0BFLOUT1	O	ERU0.1A3	Boundary flag 1 output of group 0
G1BFLOUT1	O	ERU0.1B3	Boundary flag 1 output of group 1
G0BFLOUT2	O	ERU0.2A3	Boundary flag 2 output of group 0
G1BFLOUT2	O	ERU0.2B3	Boundary flag 2 output of group 1
G0BFLOUT3	O	ERU0.3A3	Boundary flag 3 output of group 0
G1BFLOUT3	O	ERU0.3B3	Boundary flag 3 output of group 1

1) Internal signal connection.

19 Analog Comparator (ACMP) and Out of Range Comparator (ORC)

This chapter describes the controls for analog comparator (ACMP) and out of range comparator (ORC).

19.1 Overview

This section gives an overview about the feature set of the Analog comparator and the Out of Range Comparator. In PSC1Mx, there are up to 4 Analog Comparators (ACMPx [x=0-3]) and 8 Out of Range Comparator (ORCx [x=0-7]).

19.1.1 Features

The following features are provided:

- Out of Range Comparator (ORC)
  - Over-voltage monitoring for the analog inputs pins of the ADC module
- Analog Comparator (ACMP)
  - Monitor external voltage level
  - Selectable low power mode
  - Inverted output option

19.2 Analog Comparator (ACMP)

Figure 198 shows the block diagram of a Analog Comparator unit.

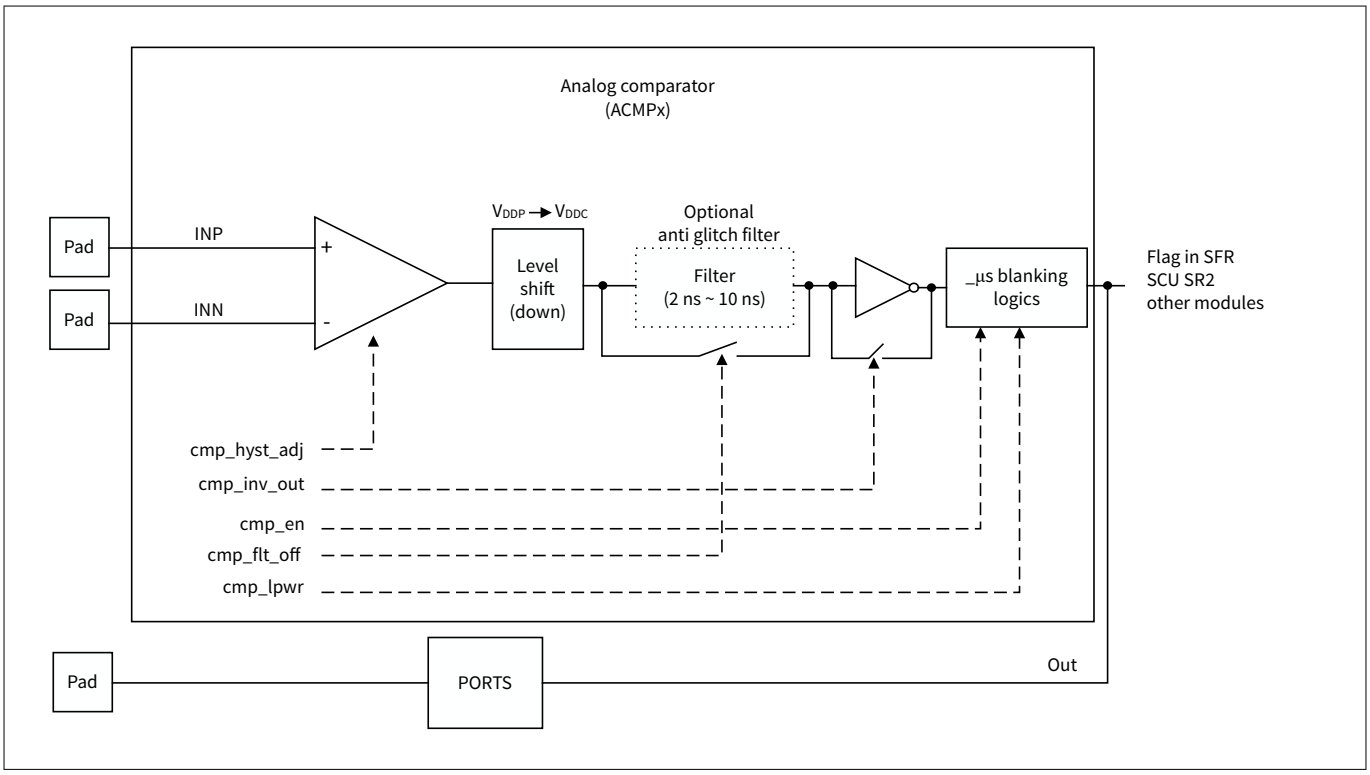


Figure 198 Block diagram of Analog Comparator

Input INP is compared with input INN in the pad voltage domain. The digital comparator output signal is shifted down from V_{DDP} power supply voltage level to V_{DDC} core voltage level. A filter (if enabled) absorbs generated spikes and can be controlled via bit ANACMPx.CMP_FLT_OFF. To prevent undefined states immediately after

comparator enabling, a blanking module is added to ensure a stable output during transition state. The blanking time is in a range of few usec. With the help of bit ANACMPx.CMP_INV_OUT, the comparator output is inverted. The output of the comparator could be used to wake-up the system from power save mode. It can also be observed from a pin.

The analog comparator can be switched off via bit CMP_EN bit in the ANACMPx register to save power.

Divided Reference Voltage

A resistor chain (in the order of 500 kOhm) divides the reference voltage value from ACMP.REF input if bit ANACMP1.REF_DIV_EN is set. According to Figure 199, the divided voltage is delivered to pin ACMP1.INP. With bit ANACMP0.ACMP0_SEL, ANACMP2.ACMP2_SEL and ANACMP3.ACMP3_SEL all other comparators can be supplied by this divided reference voltage.

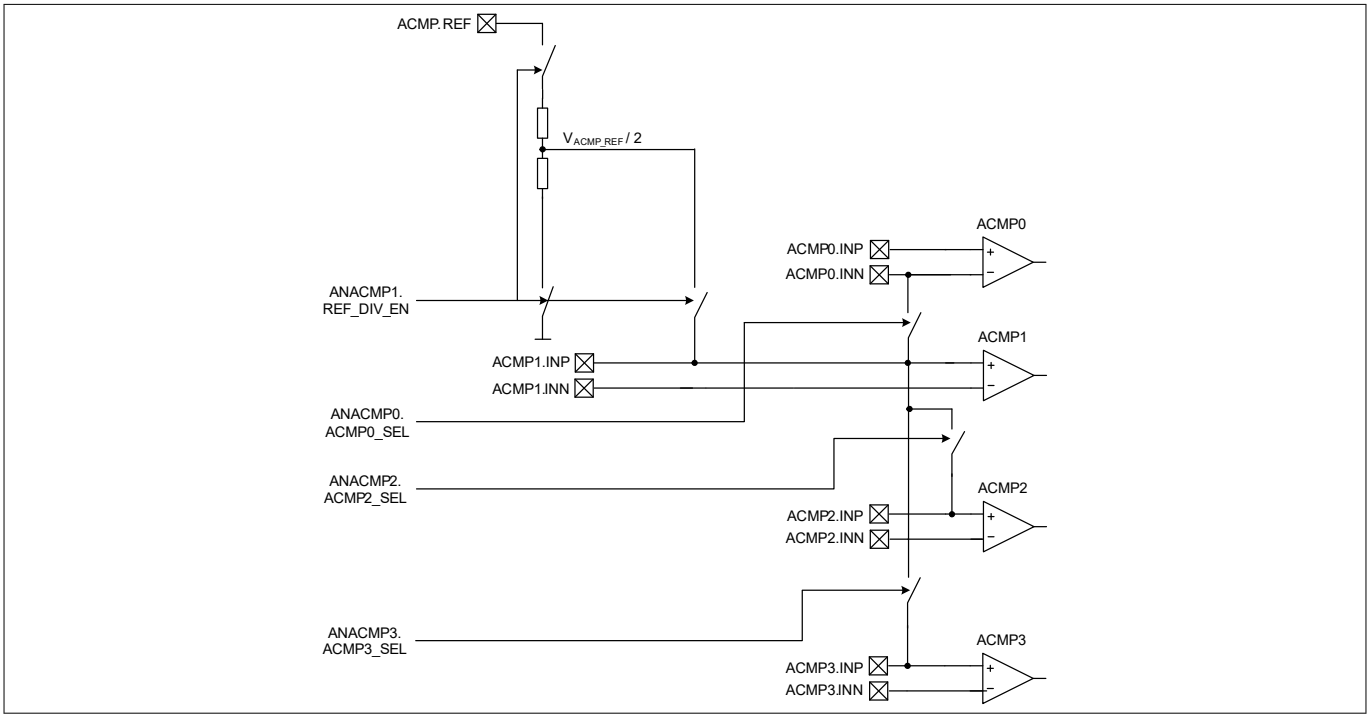


Figure 199 Analog Comparator Reference Divider Function

Low Power Mode

A low power state helps to reduce the total power consumption, for example, during sleep mode. It can be enabled by setting ANACMPx.LPWR to HIGH. In this mode, the analog comparators may shows a reduced performance. When switching back to normal mode, the blanking time applies to ensure a stable output.

19.3 Out of Range Comparator (ORC)

Out of range comparator (ORC) is build into every ADC channel which will trigger other modules or an interrupt when voltage out of range condition occurs. This happens when voltage at the input channel rises to above V_{DDP} level or when the input channel falls to a voltage below V_{DDP} level.

The out of range comparator is connected to ERU0 modules. All out of range comparator events is assigned to one interrupt node.

Before using the out of range comparator, it has to be enabled by setting the bits at ORCCTRL.ENORCx and configuring it to detect voltage higher or lower than V_{DDP} by setting bits ORCCTRL.CNFx (See Figure 200).

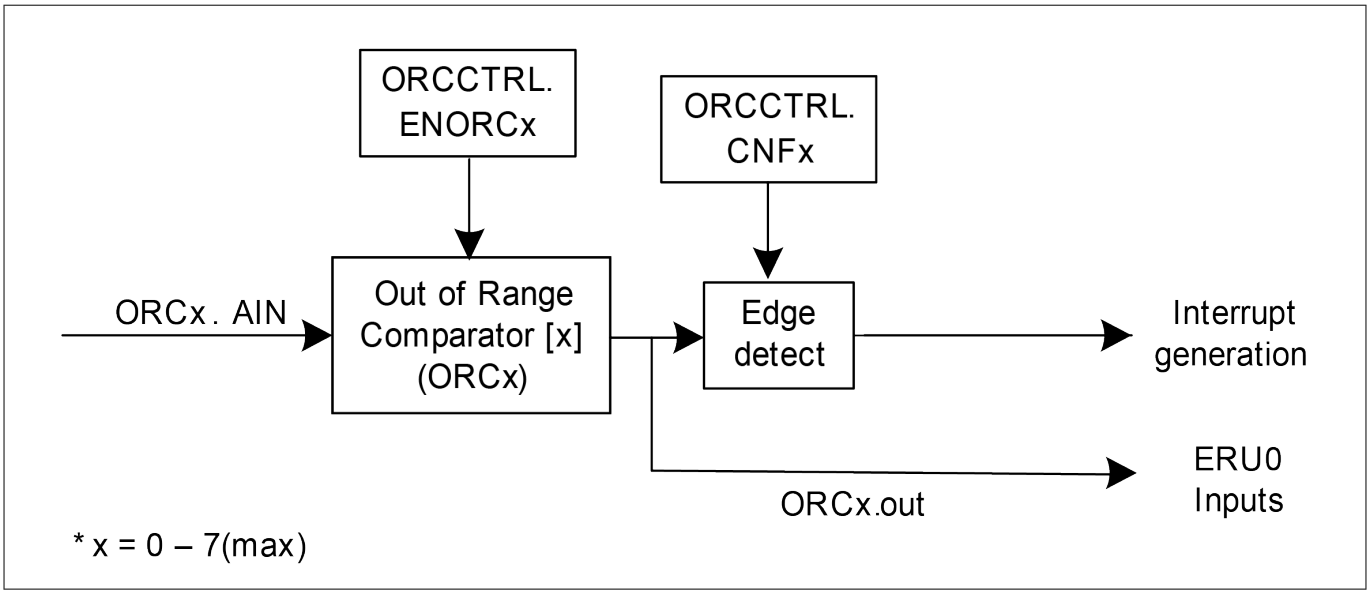


Figure 200 Out of range comparator

When a voltage out of range event occurs, the event sets an interrupt request through SCU.SR2 and triggers ERU0 through output of ORC (ORCx.out).

In PSC1Mx, ORCx.AIN[x=0-7] are connected to P2.2 - P2.9 respectively.

19.4 Service Request Generation

Both the ACMP and ORC has a service request output each. They are combined into a single output and connected to one of the interrupt node in the Nested Vectored Interrupt Controller (NVIC) via SCU.SR2.

The service request handling is part of the GCU block in SCU module. Refer to the “Service Request” description in the GCU section which is part of the SCU chapter for more details.

19.5 Debug Behavior

ACMPx and ORCx are not affected by the debug activities performed using external debug probe.

19.6 Registers

ACMP registers are accessible via the APB bus. ORC registers are accessible via the AHB bus. The absolute register address is calculated by adding:

Module Base Address + Offset Address

Following access to ACMP/ORC SFRs result in an AHB/APB error response:

- Read or write access to undefined address
- Write access to read-only registers

Table 149 Registers Address Space

Module	Base Address	End Address	Note
COMPARATOR	$4001\ 0000_H$	$4001\ FFFF_H$	System Control Unit Registers

Table 150 Registers Overview

Register Name	Description	Offset Addr.[1]	Access Mode	
			Read	Write
ORCx Registers				
ORCCTRL	Out of Range Comparator Control Register	0500_H	U, PV 32	U, PV 32
ACMPx Registers				
ANACMP0	Analog Comparator 0 Control Register	$105C_H$	U, PV	U, PV
ANACMP1	Analog Comparator 1 Control Register	1060_H	U, PV	U, PV
ANACMP2	Analog Comparator 2 Control Register	1064_H	U, PV	U, PV
ANACMP3	Analog Comparator 3 Control Register	1068_H	U, PV	U, PV

1) The absolute register address is calculated as follows:
Module Base Address + Offset Address (shown in this column)

19.7 Interconnects

ACMP and ORC are connected to the ports, the ERUx, CCU4x, CCU8x and BCCU0.

Table 151 Analog Comparator Pin Connections

Input/Output	I/O	Connected To	Description
ACMP0.INN	I	P2.8	“-” input of ACMP0
ACMP0.INP	I	P2.9	“+” input of ACMP0
ACMP1.INN	I	P2.6	“-” input of ACMP1
ACMP1.INP	I	P2.7	“+” input of ACMP1
ACMP2.INN	I	P2.2	“-” input of ACMP2
ACMP2.INP	I	P2.1	“+” input of ACMP2
ACMP3.INN	I	P2.12	“-” input of ACMP3
ACMP3.INP	I	P2.13	“+” input of ACMP3
ACMP.REF	I	P2.11	Reference input of ACMP
ACMP0.OUT	O	P0.10 P2.10 P4.3 ERU0.0A0 BCCU0.IN5 BCCU0.IN6 CCU40.IN0AS CCU40.IN3AR CCU80.IN2AR CCU80.IN3AS ERU1.3A0 CCU41.IN0AS CCU41.IN3AR CCU81.IN0AS CCU81.IN2AR	output of ACMP0

(table continues...)

Table 151 (continued) Analog Comparator Pin Connections

Input/Output	I/O	Connected To	Description
ACMP1.OUT	O	P1.0 P3.0 ERU0.1A0 BCCU0.IN0 BCCU0.IN7 CCU40.IN0AR CCU40.IN2AS CCU80.IN0AR CCU80.IN1AS CCU81.IN1AS P2.5.HW0 pull control ERU1.0A0 CCU41.IN0AR CCU41.IN2AS CCU81.IN0AR	output of ACMP1
ACMP2.OUT	O	P0.5 P1.2 P2.12 P4.2 ERU0.2A0 BCCU0.IN2 BCCU0.IN3 BCCU0.IN4 CCU40.IN1AS CCU40.IN2AR CCU80.IN0AS CCU80.IN1AR P2.3.HW0 pull control ERU1.2A0 CCU41.IN1AS CCU41.IN2AR CCU81.IN1AR CCU81.IN3AS	output of ACMP2

(table continues...)

Table 151 (continued) Analog Comparator Pin Connections

Input/Output	I/O	Connected To	Description
ACMP3.OUT	O	P4.1 BCCU0.IN1 BCCU0.IN8 CCU40.IN1AR CCU40.IN3AS CCU80.IN2AS CCU80.IN3AR ERU1.1A0 CCU41.IN1AR CCU41.IN3AS CCU81.IN2AS CCU81.IN3AR	output of ACMP3

Table 152 Out of Range Comparator Pin Connections

Input/Output	I/O	Connected To	Description
ORC0.AIN	I	P2.2	analog input of ORC0
ORC1.AIN	I	P2.3	analog input of ORC1
ORC2.AIN	I	P2.4	analog input of ORC2
ORC3.AIN	I	P2.5	analog input of ORC3
ORC4.AIN	I	P2.6	analog input of ORC4
ORC5.AIN	I	P2.7	analog input of ORC5
ORC6.AIN	I	P2.8	analog input of ORC6
ORC7.AIN	I	P2.9	analog input of ORC7
ORC0.OUT	O	ERU0.0B2	output of ORC0
ORC1.OUT	O	ERU0.1B2	output of ORC1
ORC2.OUT	O	ERU0.0A2 ERU0.2B2	output of ORC2
ORC3.OUT	O	ERU0.1A2	output of ORC3
ORC4.OUT	O	ERU0.2A2	output of ORC4
ORC5.OUT	O	ERU0.3A2	output of ORC5
ORC6.OUT	O	ERU0.3B2	output of ORC6
ORC7.OUT	O	ERU0.3A0	output of ORC7

20 Temperature Sensor (DTS)

This chapter describes the controls for Temperature Sensor (DTS). It is also known as “TSE”.

20.1 General Description

The Temperature Sensor (DTS) generates a measurement result that indicates directly the current temperature. The result of the measurement is displayed via bit field ANATSEMON.TSE_MON. The user routines (_CalcTemperature) inside ROM takes this value and calulate the actual silicon temperature. This routine can be called by the application software and more details are described in Chapter “Bootstrap Loaders and User Routines”. Library functions are also available and described in the Temperature Sensor device guide. The temperature sensor has to be enabled before it can be used via bit ANATSECTRL.TSE_EN.

The measurement result is ready when the SRRAW.TSE_DONE flag is set. An interrupt can be triggered when it is enabled via SRMSK.TSE_DONE bit. After the measurement is completed and result is stored in TSE_MON, temperature sensor continue with next measurement. Measurement are disabled when TSE_EN is cleared to 0. Hence, reading the TSE_MON value will provide you the latest temperature.

The temperature sensor is capable of generating interrupt requests when DTS temperature measurement in result crosses upper and/or lower threshold value configured in bit ANATSEIH.TSE_IH and ANATSEIL.TSE_IL respectively. Digital comparators are implemented to compare the actual measurement result against configured limits and triggers interrupt if the value fall outside of valid range. Result of comparison is shown as the SRRAW.TSE_HIGH and TSE_LOW flag. The user routines (_CalcTSEVAR) can be used to obtain the upper and lower temperature limits needed to program the bit TSE_IH and TSE_IL and more details are described in Chapter “Bootstrap Loaders and User Routines”. Library functions are also available and described in the Temperature Sensor device guide.

20.2 Service Request Generation

The temperature sensor has a service request output for the TSE_DONE, TSE_HIGH and TSE_LOW event. They are combined with events from other modules into a single output and connected to one of the interrupt node in the Nested Vectored Interrupt Controller (NVIC) via SCU.SR1.

The service request handling is part of the GCU block in SCU module. Refer to the “Service Request” description in the GCU section which is part of the SCU chapter for more details.

20.3 Registers

DTS registers are accessible via the APB bus. The absolute register address is calculated by adding:

Module Base Address + Offset Address

Following access to DTS SFRs result in an AHB/APB error response:

- Read or write access to undefined address
- Write access to read-only registers

Table 153 Registers Address Space

Module	Base Address	End Address	Note
DTS	$4001\ 0000_H$	$4001\ FFFF_H$	System Control Unit Registers

Table 154 Registers Overview

Register Name	Description	Offset Addr.[1)]	Access Mode	
			Read	Write
ANATSECTRL	Temperature Sensor Control Register	0024_H	U, PV	U, PV
ANATSEIH	Temperature Sensor High Interrupt Register	0030_H	U, PV	U, PV
ANATSEIL	Temperature Sensor Low Interrupt Register	0034_H	U, PV	U, PV
ANATSEMON	Temperature Sensor Counter2 Monitor Register	0040_H	U, PV	U, PV

1) The absolute register address is calculated as follows:
Module Base Address + Offset Address (shown in this column)

21 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 155 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*

21.1 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, for example, with Hall Sensor monitoring or direct coupling with Encoders.

21.1.1 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 and 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

- flexible coherent and non coherent update mechanism
- external modulation function
- load controlled by external events
- dithering PWM
- floating point prescaler
- 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 156 a summary of the major features of the CCU4 unit mapped with the most common applications.

Table 156 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, for example,, one timer working in capture mode and one working in compare
4 dedicated service request lines	Specially developed for: • generating interrupts for the microprocessor • flexible connectivity between peripherals, for example, ADC triggering.
Linking with other modules	Flexible profiles for: • Hall Sensor feedback/monitoring • Motor Encoders feedback/monitoring • PWM parallel modulation • Flexible signal conditioning
Flexible coherent and non coherent update schemes	Flexible profiles that can be used for: • fuel injection control • multiple observation points for motor shaft rotation • on-the-fly compensation linked to real-time operation condition sensing • cpu load optimization for fast control loops

21.1.2 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, for example, 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 201.

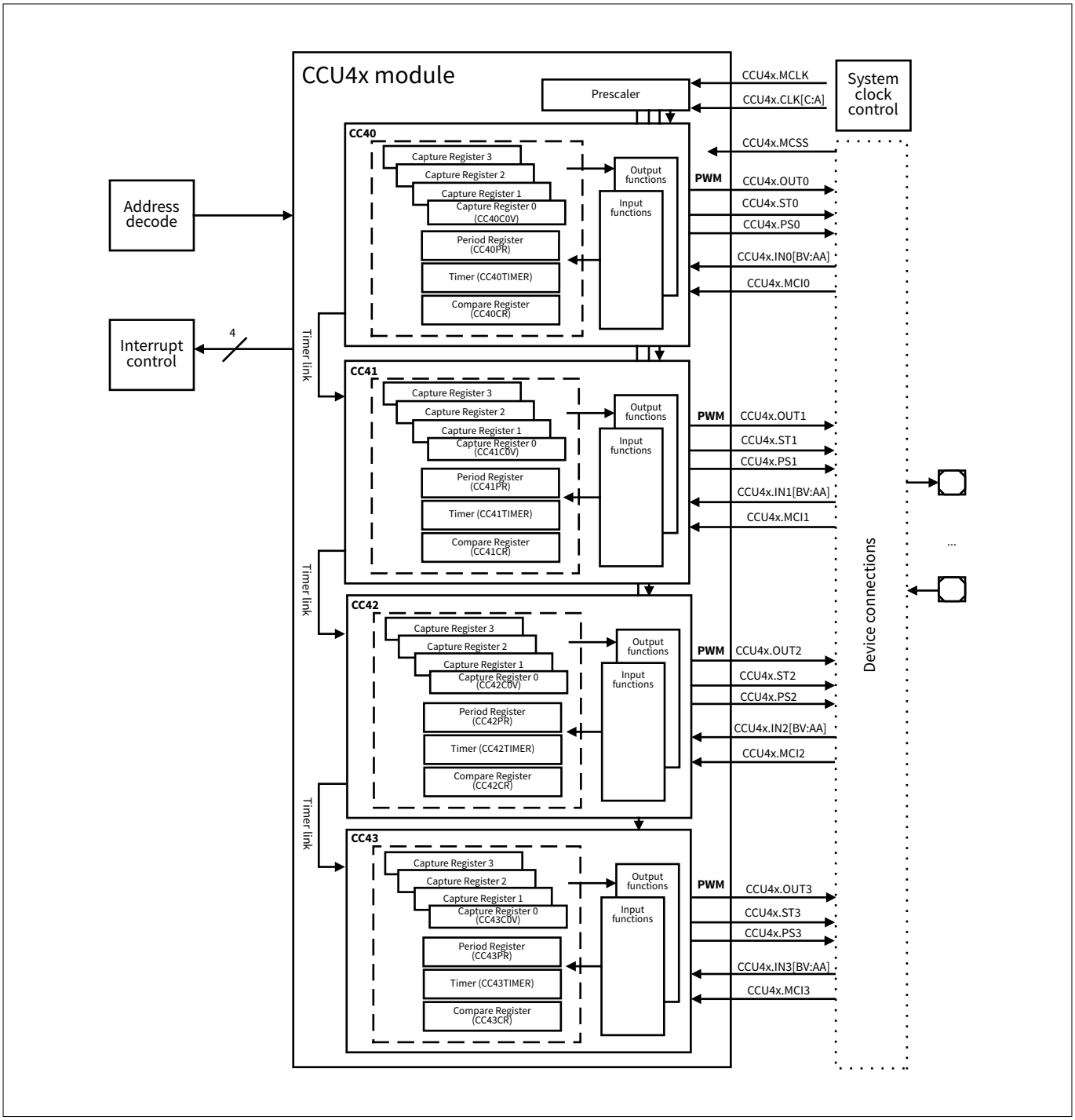


Figure 201 CCU4 block diagram

21.2 Functional Description

The following sections describe the complete set of functions and usability of the CCU4 peripheral.

In each figure several registers may be depicted to indicate controlability or configurability. These registers follow the description given in Figure 202. One should also note that indexing in a register can be done via the non capital y, x or n.

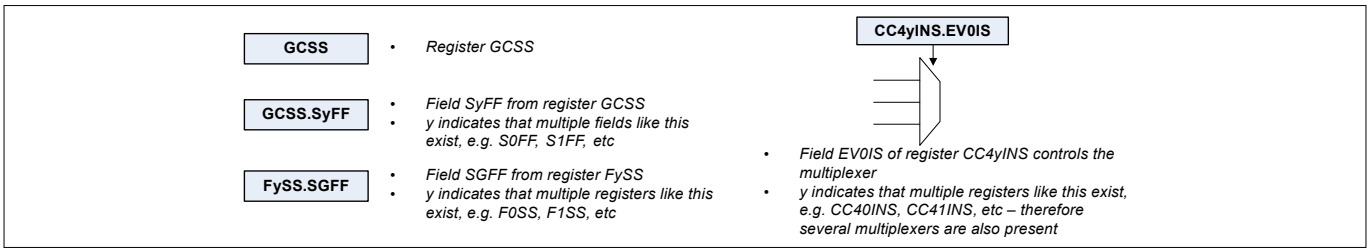


Figure 202 Register description in figures (example)

21.2.1 Timer Slice Overview

The input path of a CCU4 timer slice (a timer slice has a nomenclature of CC4y) is comprised of a selector (Timer Slice Input Selector) and a connection matrix unit (Timer Slice Connection Matrix). Via the input selector and connection matrix, the user can select several signals - that are connected to ports or peripherals - to control several available functions inside the timer kernel, for example, start, stop, count, etc. 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 203.

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.

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 Timer Slice Connection Matrix.

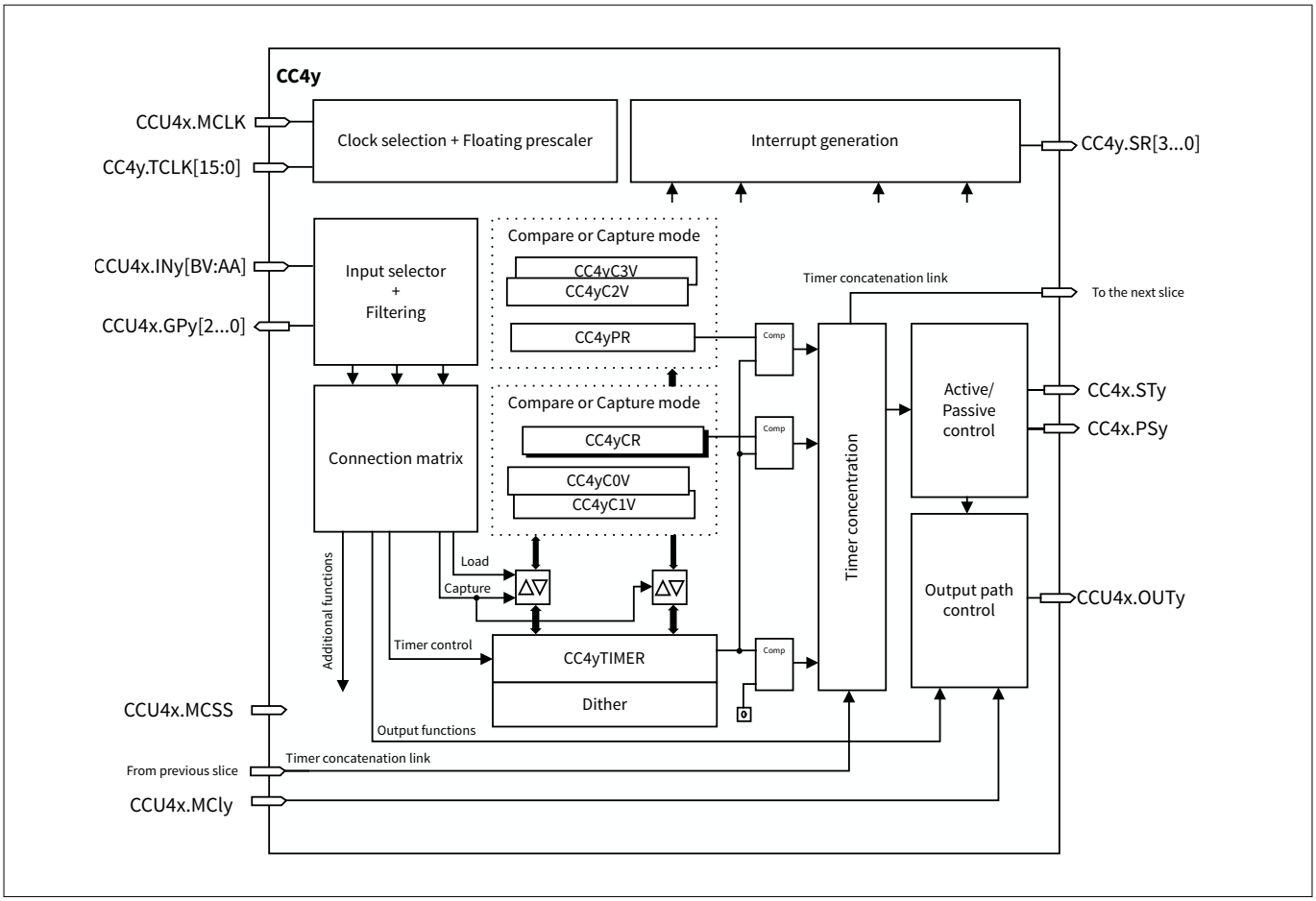


Figure 203 CCU4 slice block diagram

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 timer slice).

Table 157 CCU4 slice pin description

Pin	I/O	Description
CCU4x.MCLK	I	Module clock
CC4y.TCLK[15:0]	I	Clocks from the prescaler
CCU4x.INy[BV:AA]	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
CCU4x.GPy[2...0]	O	Signals decoded from the input selector
CC4x.STy	O	Slice comparison status value
CCU4x.PSy	O	Multi-Channel pattern update trigger
CCU4x.OUTy	O	Slice dedicated output pin

***Notes**:*

1. *The status bit outputs of the Kernel, CCU4x.STy, are extended for one more kernel clock cycle.*
2. *The Service Request signals at the output of the kernel are extended for one more kernel clock cycle.*
3. *The maximum output signal frequency of the CCU4x.STy outputs is module clock divided by 4.*

21.2.2 Timer Slice 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 204. The user has the possibility of selecting any of the CCU4x.INy[BV:AA] 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.

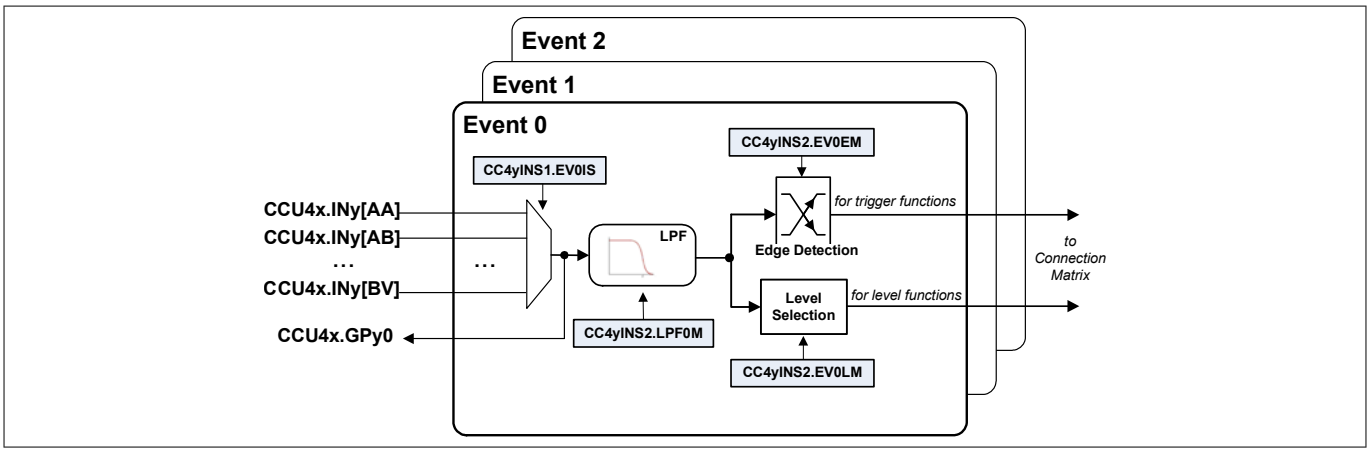


Figure 204 Slice input selector diagram

21.2.3 Timer Slice Connection Matrix

The connection matrix maps the events coming from the input selector to several user configured functions, Figure 205. The following functions can be enabled on the connection matrix:

Table 158 Connection matrix available functions

Function	Brief description	Map to figure Figure 205
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

(table continues...)

Table 158 (continued) Connection matrix available functions

Function	Brief description	Map to figure Figure 205
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 Timer Concatenation.

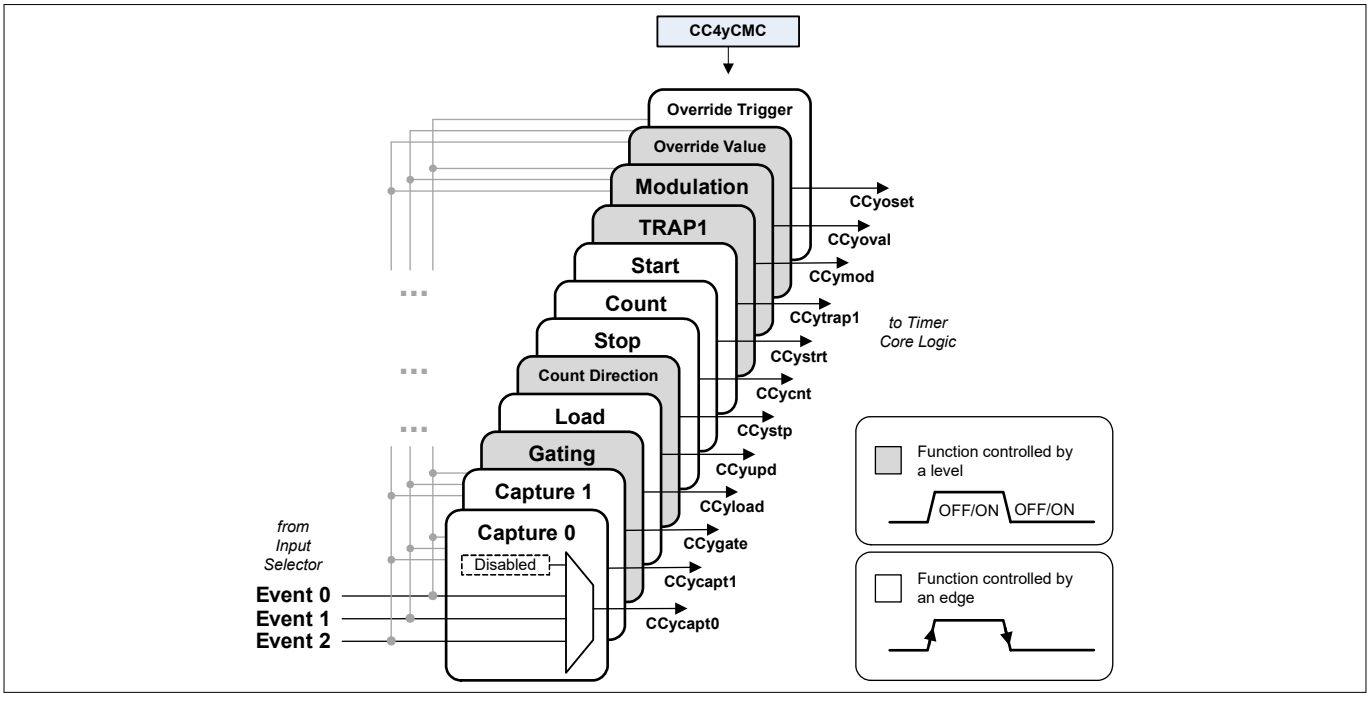


Figure 205 Slice connection matrix diagram

21.2.4 Timer Slice Core Functions

In the following sub sections one can find the description of the main functionalities of each CCU4 Timer Slice core. This functions include the different counting schemes, how to start and stop a timer or how to calculate and update the duty cycle of each Timer Slice.

Besides these functions, there are several others that can be controlled via external signals, for example, port pins or other peripherals - these functions are described in Timer Slice External Functions.

21.2.4.1 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 206.

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.

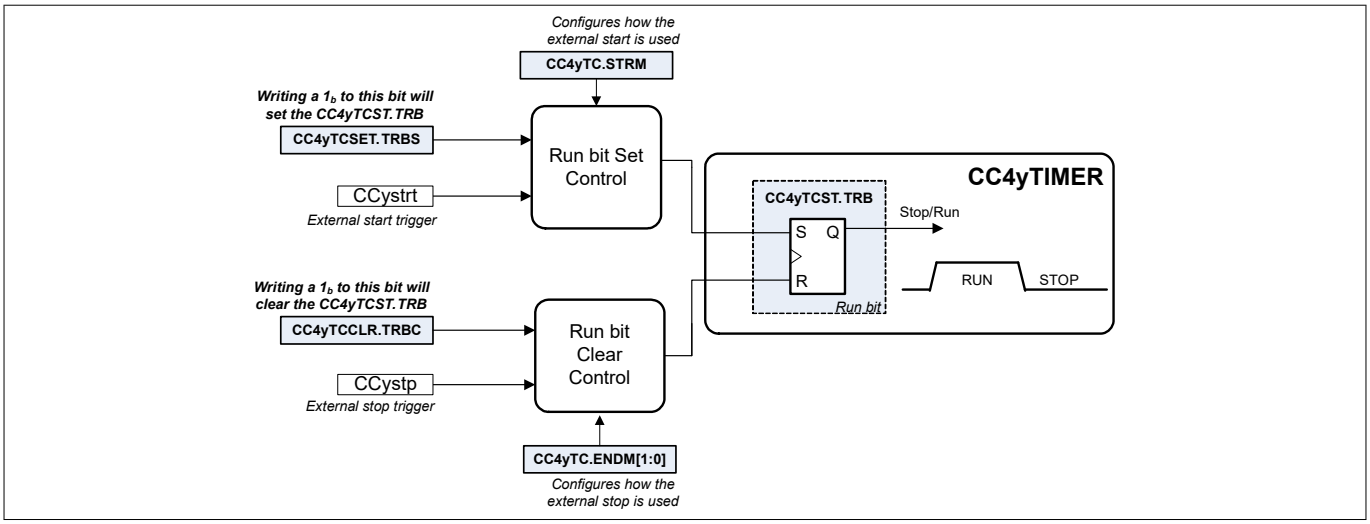


Figure 206 Timer start/stop control diagram

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 External Start/Stop for a description how to configure an input as start function). This input should be connected to all the Timers that need to be started synchronously (see Interconnects for a complete list of module connections), Figure 207.

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 External Start/Stop) and then the software must write $1_B/0_B$ (depending on the configuration of the external start function) to the specific bitfield of the CCUCON register (this register is described on the SCU chapter).

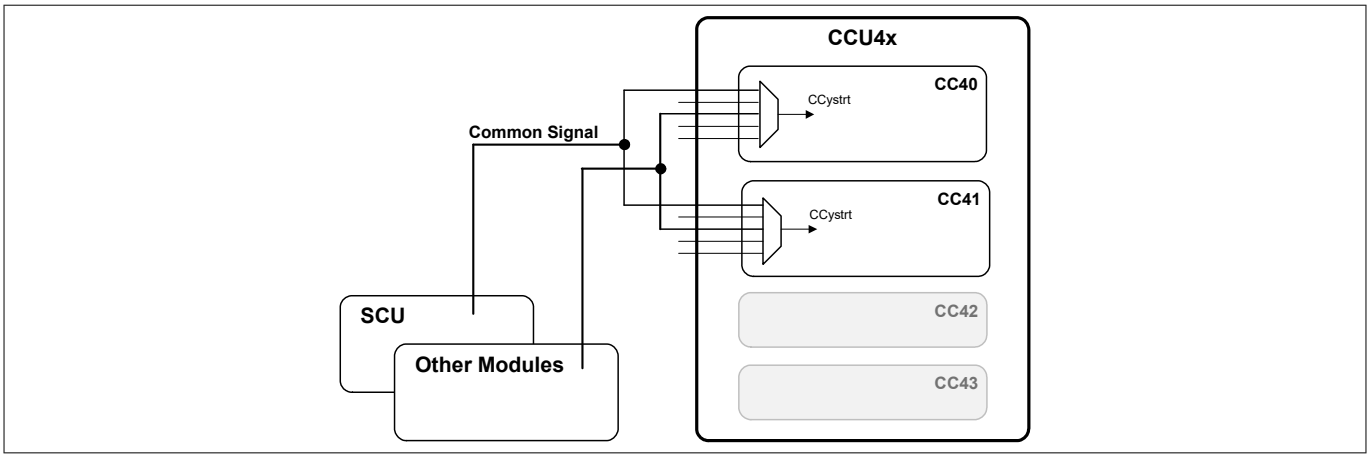


Figure 207 Starting multiple timers synchronously

21.2.4.2 Counting Modes Introduction

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.

Each compare channel of the CC4y Timer Slice has an associated Status Bit (GCST.CC4yST), that indicates the active or passive state of the channel, Figure 208. 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. Please address the different counting mode descriptions - Edge Aligned Mode to Single Shot Mode - to understand how this bitfield is set and cleared, and to have a full description of the timer behavior.

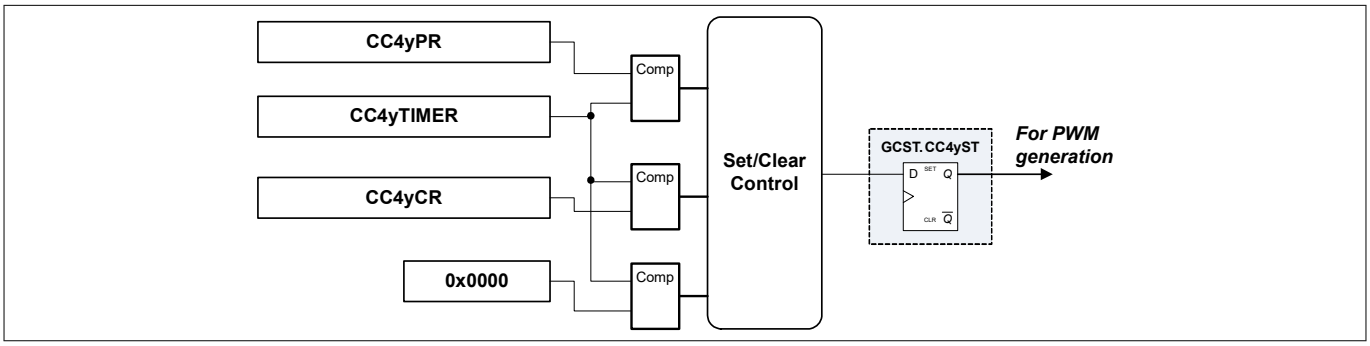


Figure 208 Status Bit

21.2.4.3 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 - Figure 209.

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.

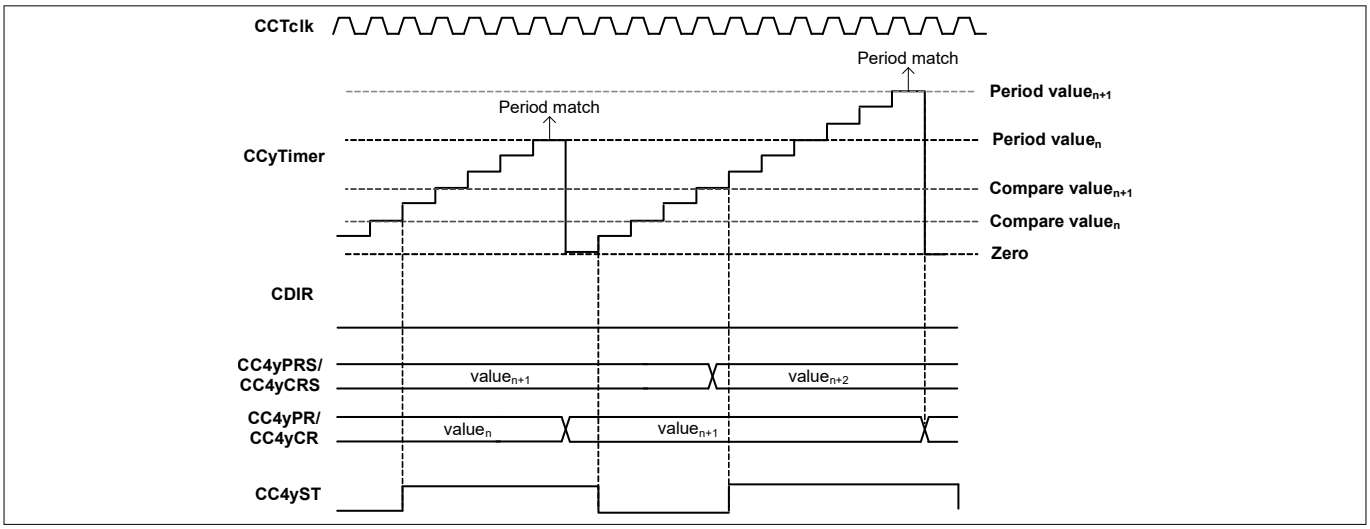


Figure 209 Edge aligned mode, CC4yTC.TCM = 0_B

In edge aligned mode, the value of the period register and compare register are updated with the value written by software into the corresponding shadow register (CC4yPRS and CC4yCRS), every time that an overflow occurs (period match) - Figure 209.

If an immediate update of the compare value and/or period is needed for the application, then the CC4ySTC.IRCC and CC4ySTC.IRPC need to be configure to 1_B - Figure 210 and Figure 211.

A complete description how to update the compare and period values is given in Updating the Period, Duty Cycle and other PWM conditions.

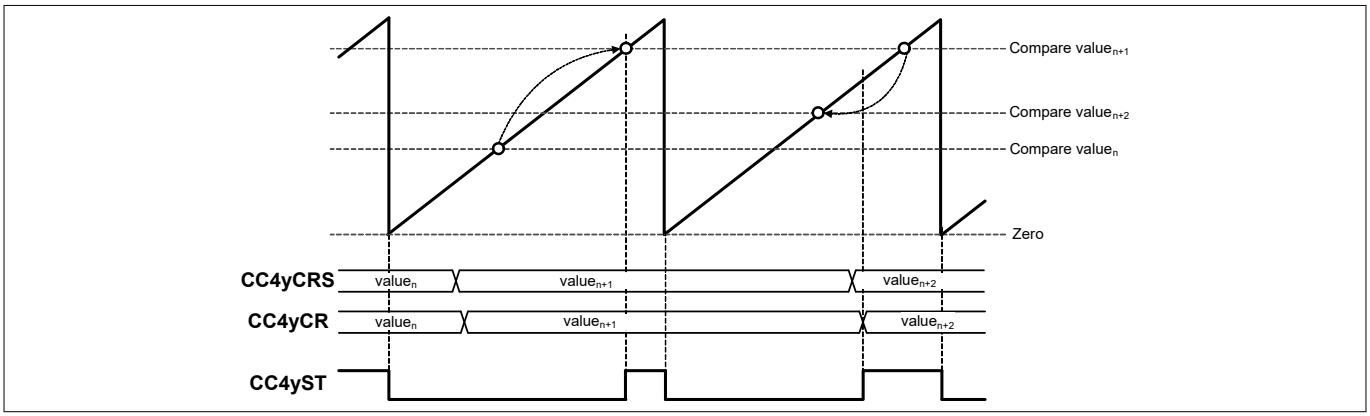


Figure 210 Edge aligned mode, immediate compare value update

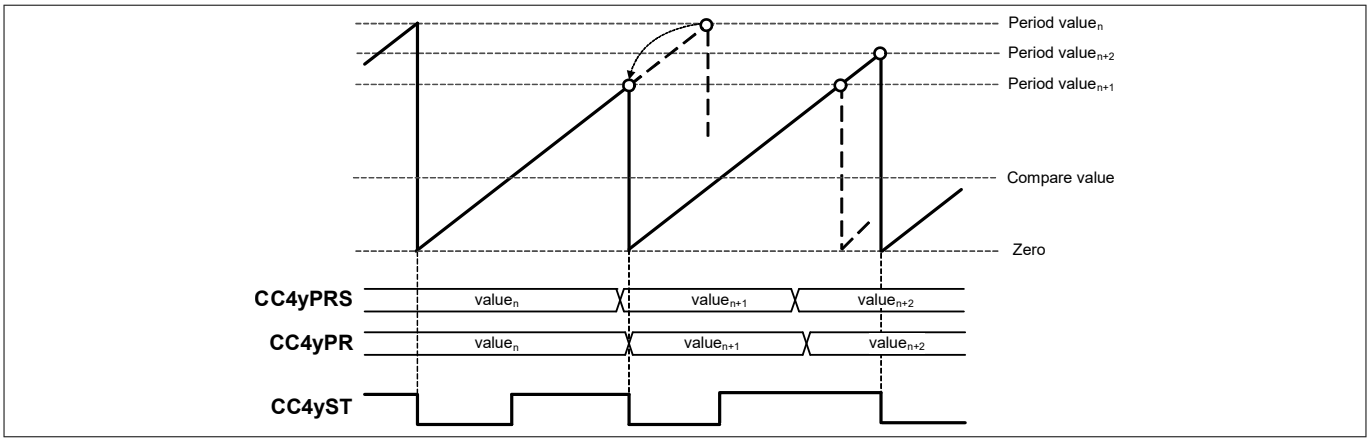


Figure 211 Edge aligned mode, immediate period value update

21.2.4.4 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.

***Note**: the bitfield 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.*

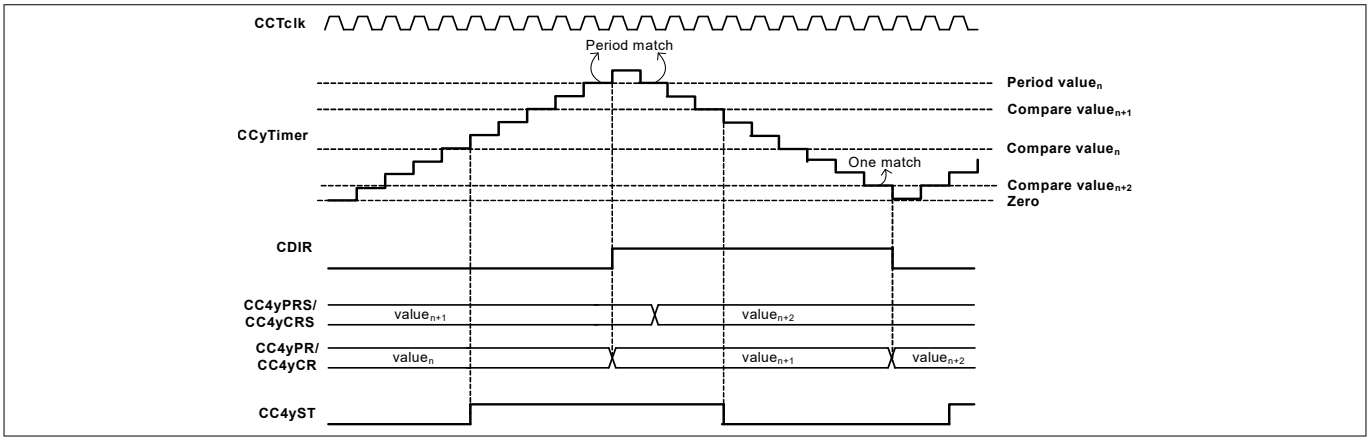


Figure 212 Center aligned mode, CC4yTC.TCM = 1_B

While in edge aligned mode, the shadow transfer for compare and period registers is executed once per switching cycle, in center aligned mode it can be executed twice:

- 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).

It is also possible to select in which instant the update is done. This is done by configuring the CC4ySTC.STM field accordingly.

If an immediate update of the compare value and/or period is needed for the application, then the CC4ySTC.IRCC and CC4ySTC.IRPC need to be configure to 1_B - Figure 213 and Figure 214.

A complete description how to update the compare and period values is given in Updating the Period, Duty Cycle and other PWM conditions.

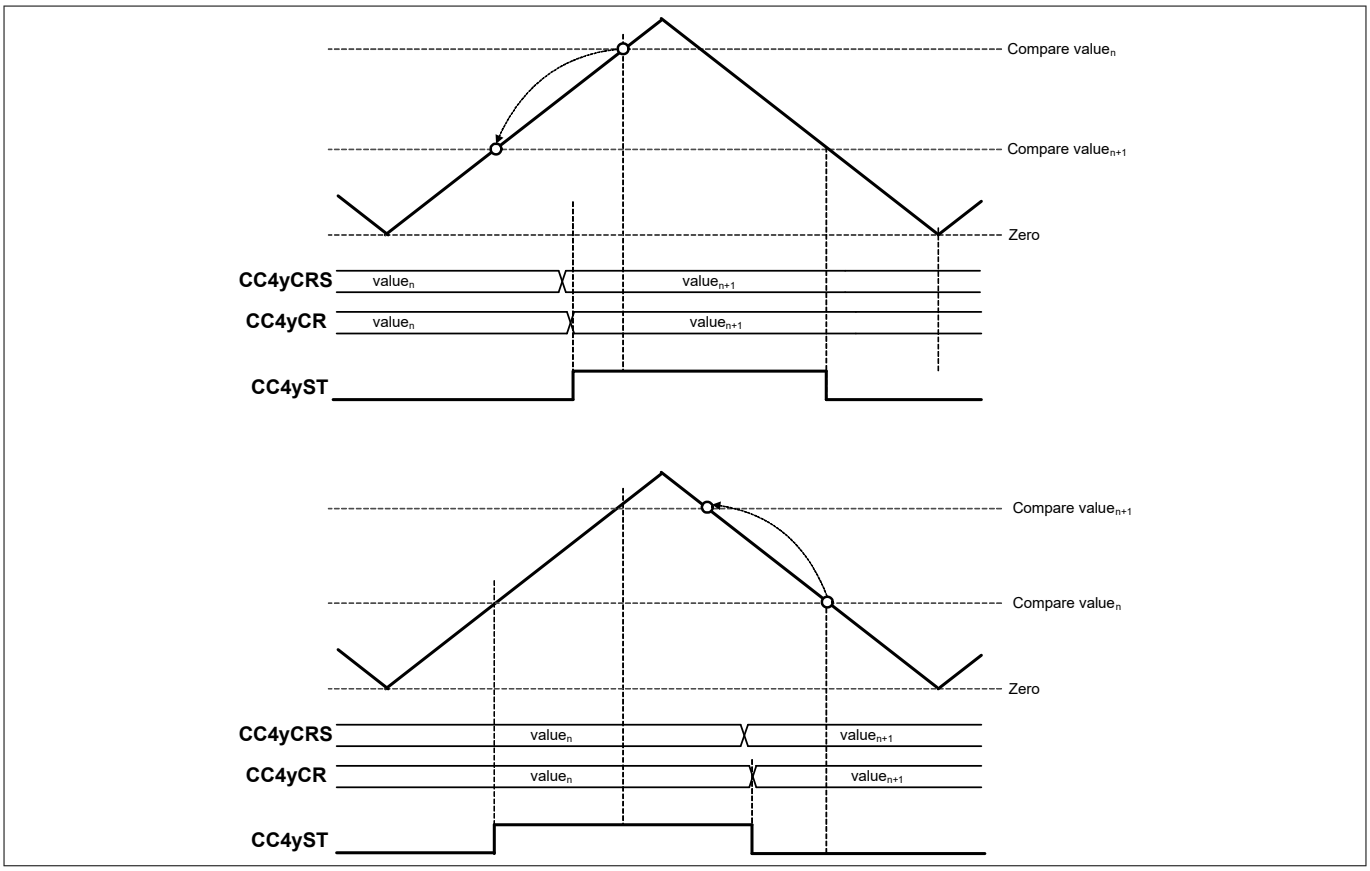


Figure 213 Center aligned mode, immediate compare value update

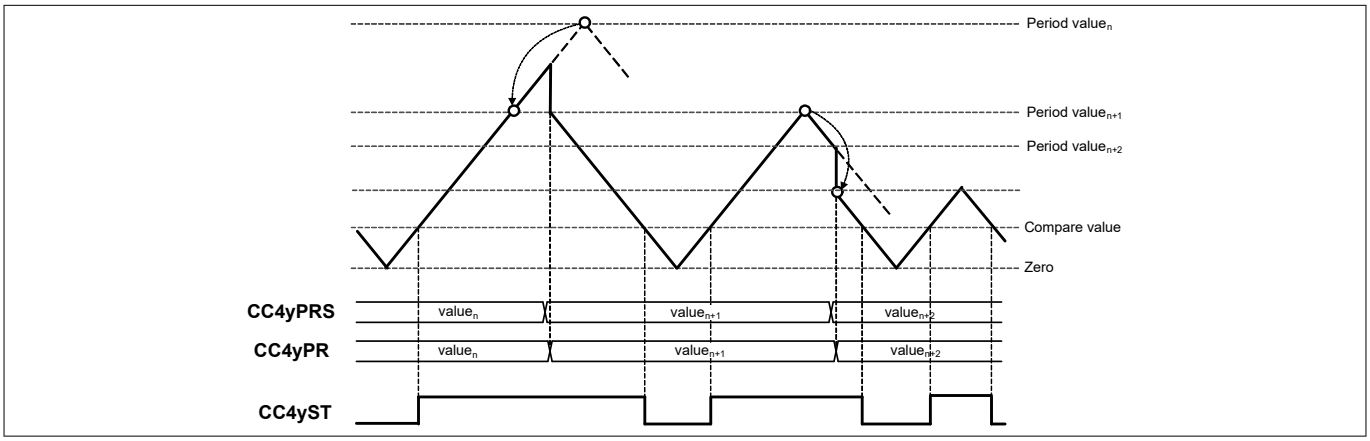


Figure 214 Center aligned mode, immediate period value update

21.2.4.5 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 215, the timer is stopped when it is cleared to 0000_H after having reached the period value. In center aligned mode, Figure 216, the period is finished when the timer has counted down to 0000_H.

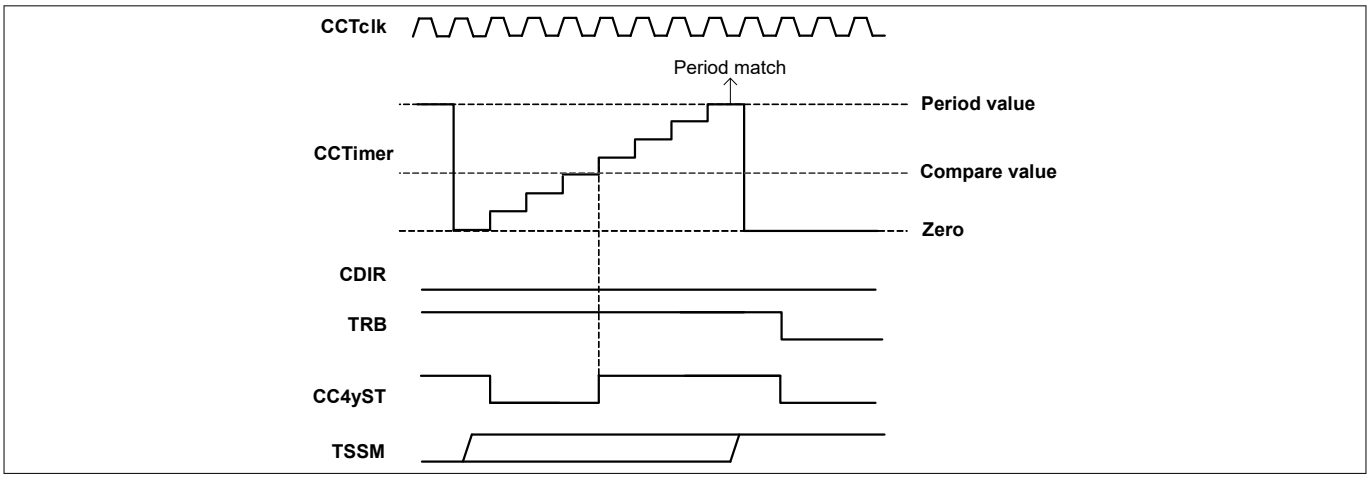


Figure 215 Single shot center aligned - CC4yTC.TSSM = 1_B, CC4yTC.TCM = 0_B

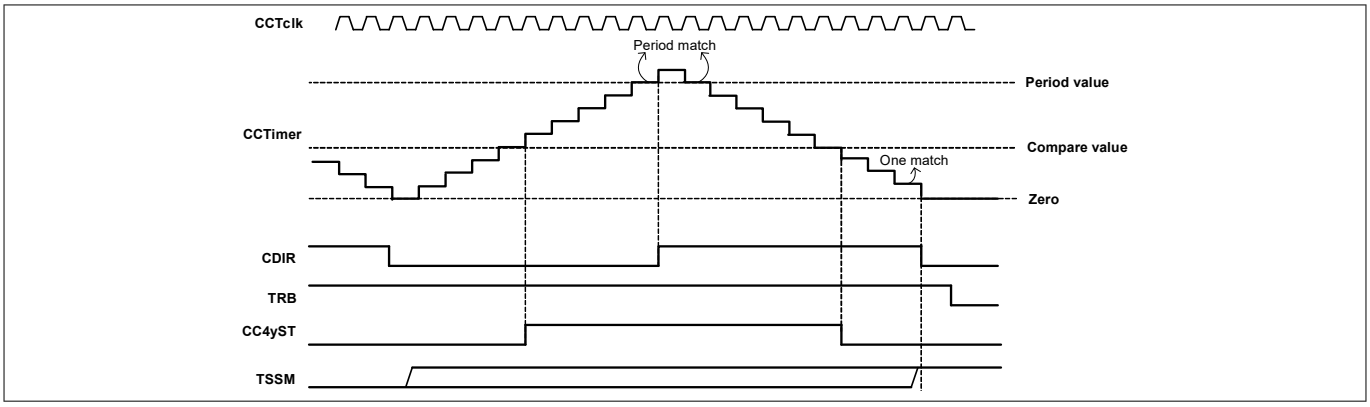


Figure 216 Single shot center aligned - CC4yTC.TSSM = 1_B, CC4yTC.TCM = 1_B

21.2.4.6 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:

$$T_{per} = < Period \quad Value > \quad + \quad 1; \quad in \quad f_{tclk} \tag{36}$$

In Center Aligned Mode, the timer period is:

$$T_{per} = \left(< Period \quad Value > \quad + \quad 1 \right) \quad \times \quad 2; \quad in \quad f_{tclk} \tag{37}$$

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:

$$DC = 1 - < Compare \quad Value > /(< Period \quad Value > + 1) \tag{38}$$

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, Updating the Period, Duty Cycle and other PWM conditions

21.2.4.7 Updating the Period, Duty Cycle and other PWM conditions

Every CCU4 timer slice contains several registers that can be updated on-the-fly, via software, with the intent of modifying certain pwm generation conditions. The most common parameters that need to be updated on-the-fly are normally the period and compare values - that will directly control the duty cycle and switching frequency. Besides these parameters, each timer slice also permits the user to update on-the-fly the floating prescaler, dither and even the passivel level of the PWM signal. The following text descriptions, will give an overview of the registers/parameters that can be updated on-the-fly, and also the options available to control this update.

Shadow Registers Overview

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 Clock Prescaler and PWM Dithering for a complete description of these functions).

The structure of the shadow registers can be seen in Figure 217.

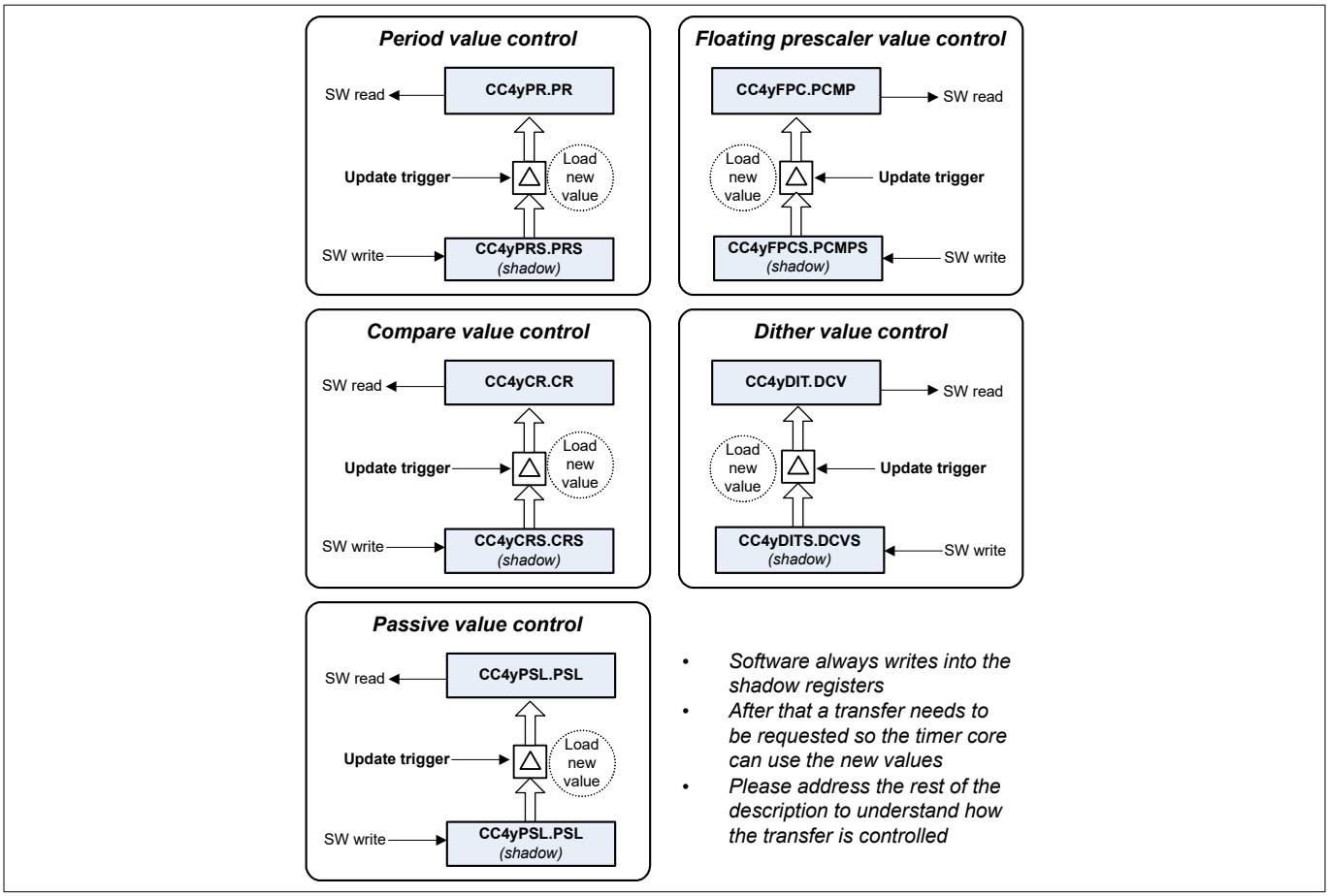


Figure 217 Shadow registers overview

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 - Figure 218. Each group of shadow registers have an individual shadow transfer enable bit. 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, when the update is done. 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).

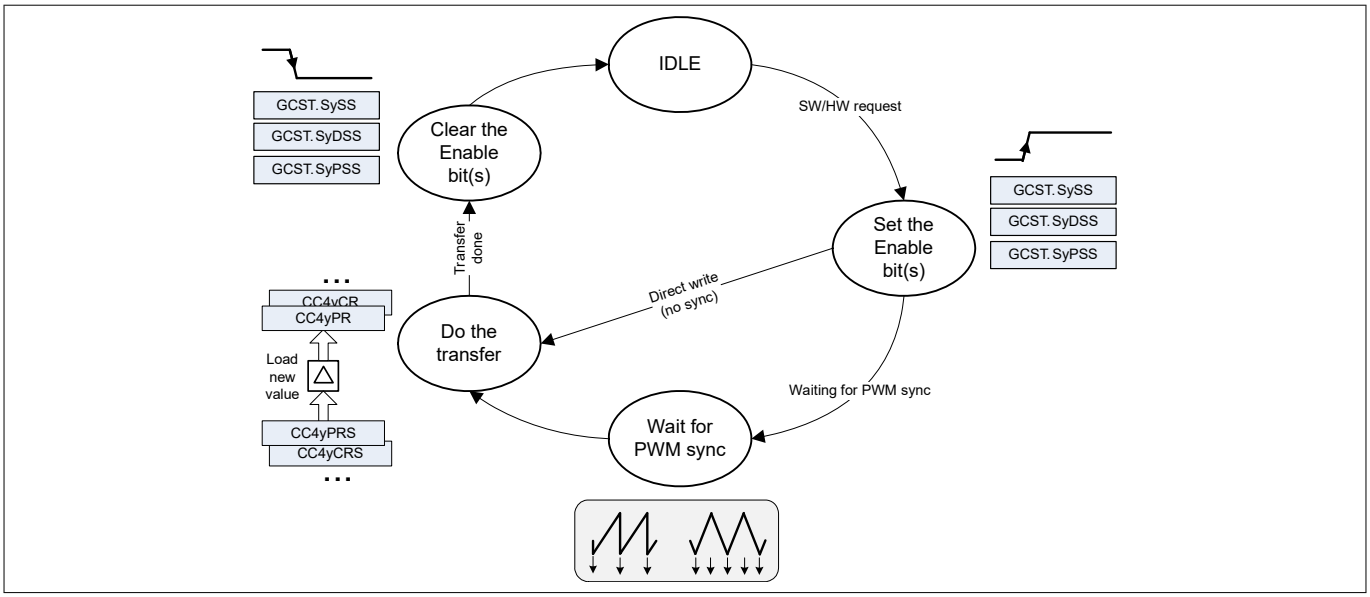


Figure 218 Shadow transfer state machine

The structure of the shadow transfer logic is depicted in Figure 219. In this figure it can be seen the logic structure associated with each of the stages of the state machine - Figure 218:

- In the request sources one can see that the shadow transfer can be requested by:
  - **software** by writing a 1_B into the specific bitfield in the GCST register
  - **an automatic trigger** that is configured in the CC4ySTC register, for example, request an update every time that the period shadow register is modified
  - **a hardware pin** linked to the CCU4x.MCSS input (this is configured via the GCTRL.MSEy and GCTRL.MSDE fields)
- In the control logic is visible that after a shadow transfer request is issued, the associated bitfield is set, indicating that an update is pending
- There is also a stage where the user can configure how the update of the values is done (this is configured also via the CC4ySTC register):
- - **synchronously with the PWM** switching cycle
  - done **immediately**

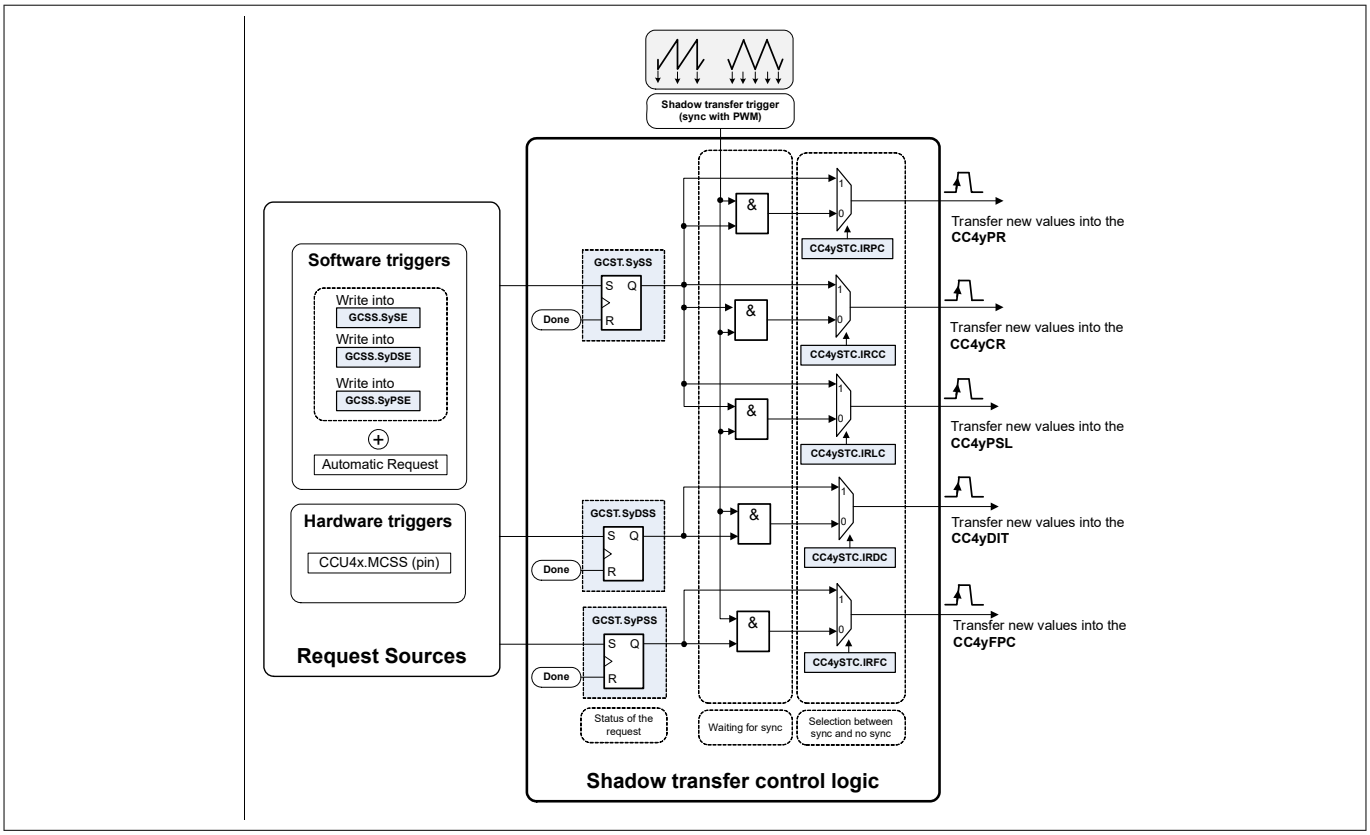


Figure 219 Shadow transfer enable logic

Updating the Shadow Registers

There are two major modes and two sub modes to update the shadow registers:

- Synchronously with the PWM switching cycle - major mode
- Immediate after an update request has been issued - major mode
- Automatic update request - sub mode
- Cascaded shadow transfer - sub mode

The following text will focus and describe each of these modes by the same order mentioned above. It is understood that the a sub mode can be used with any of the major modes.

- Synchronously with the PWM switching cycle

When the update of the registers is done synchronously with the PWM switching cycle, 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 - center aligned
- in the next clock cycle after an One Match while counting down - center aligned and edge aligned
- immediately,if the timer is stopped and the shadow transfer enable bit(s)is set - center aligned and edge aligned.

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:

- CC4ySTC.STM = 00_B (default) - Shadow transfer is done at the Period Match and One match slot
- CC4ySTC.STM = 01_B - Shadow transfer is done only at the Period Match slot
- CC4ySTC.STM = 10_B - Shadow transfer is done only at the One Match slot

Figure 220 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 Edge Aligned Mode, Center Aligned Mode and Single Shot Mode.

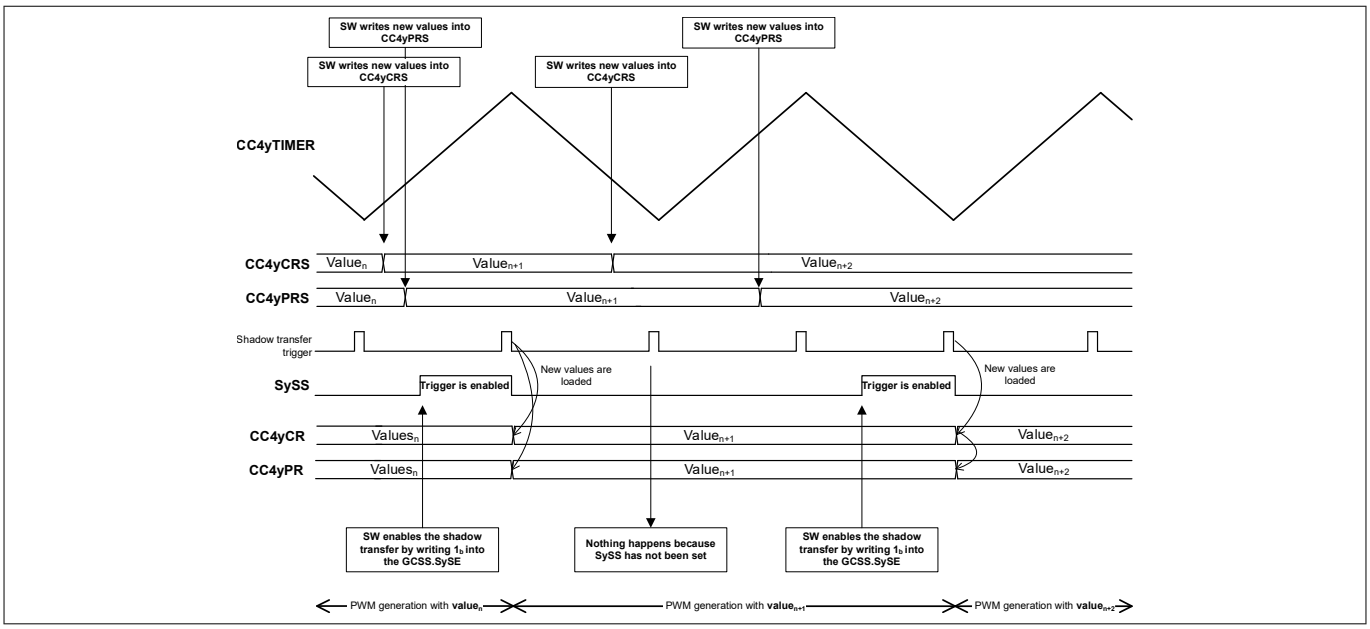


Figure 220 Shadow transfer timing example - center aligned mode with sync

- Immediate after an update request has been issued

In some applications it may be necessary to update the PWM conditions without waiting for a new switching cycle. When this is required, the user can configure via the CC4ySTC register, which fields/registers should be updated without waiting for a PWM synchronization (individual configuration fields exist for each parameter, for example, period, compare, etc).

In Figure 221 an example is depicted, of how the update of the period and compare registers can be done, without waiting for the normal switching cycle synchronization - CC4ySTC.IRPC and CC4ySTC.IRCC fields are both configured with 1_B.

The software writes new values into the period and compare shadow registers (CC4yPRS and CC4yCRS respectively). After doing that, the software requests an update of the current values being used by the timer kernel - by writing 1_B into the GCSS.SySE bitfield. Due to the fact that the software has configured the CC4ySTC register to perform an immediate update, after the GCSS.SySE is written with 1_B, the hardware will automatically load and use the new values.

The request bit - SySS - is then cleared when the transfer is done. One should notice that all the depicted CC4ySTC configuration fields need to be set to 1_B for the request bit (SySS) to be cleared immediately. If one of these CC4ySTC configuration bitfields is configured with 0B, then the associated value is updated coherently with the PWM signal - Figure 222.

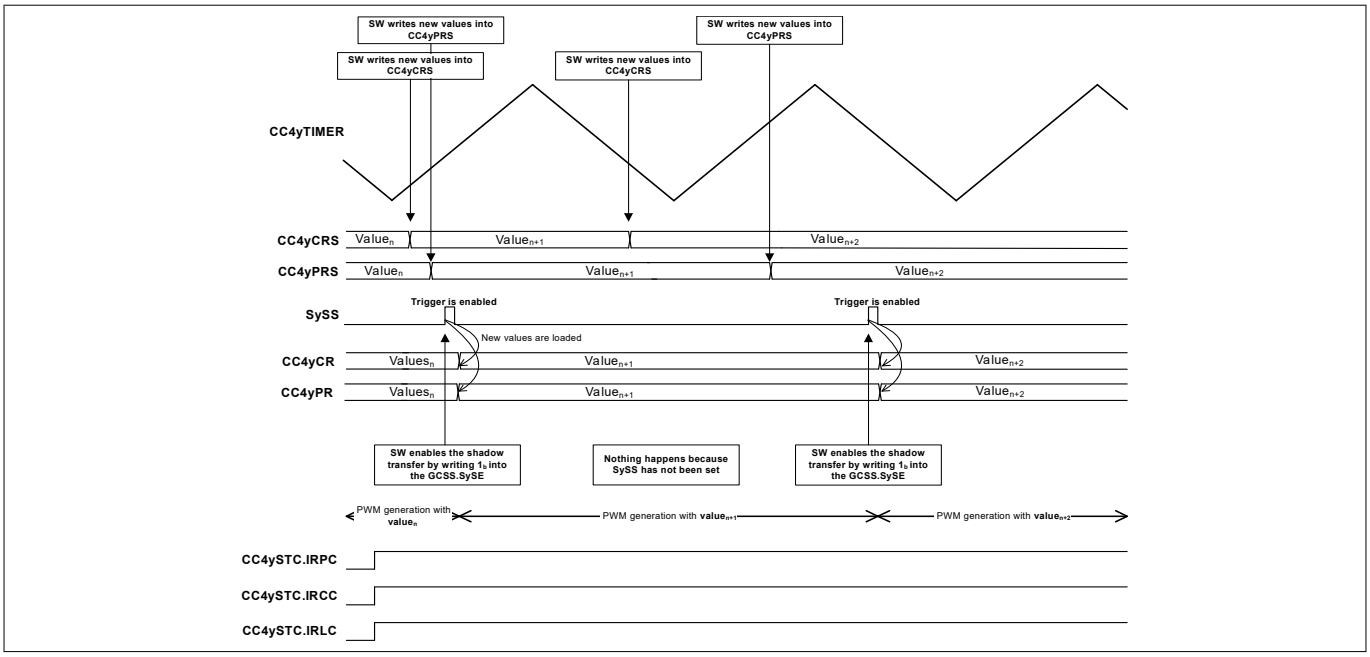


Figure 221 Shadow transfer timing example - center aligned mode without sync

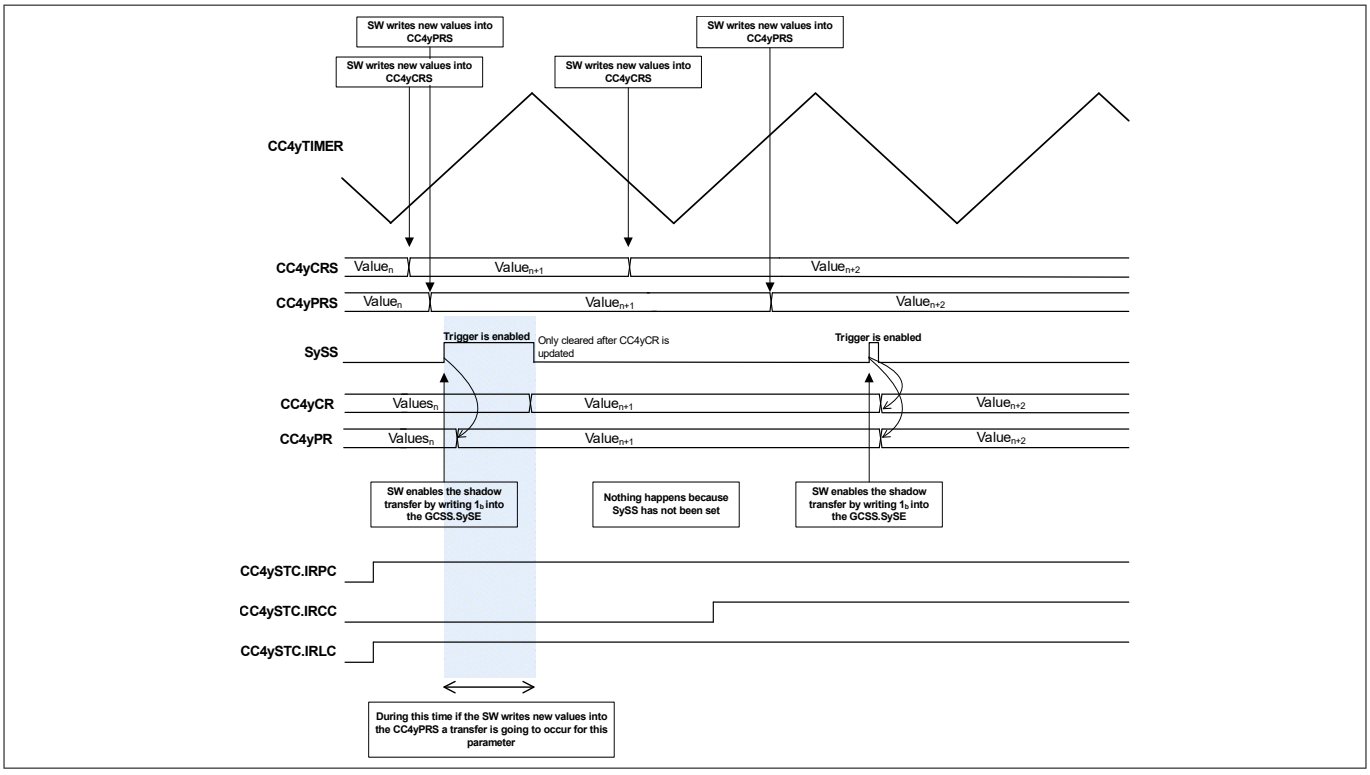


Figure 222 Shadow transfer timing example - dual configuration

- Automatic update request

Each timer slice offer also a possibility of enabling an automatic transfer request. This is configured in the CC4ySTC register (via the ASPC, ASCC, ASLC, ASDC fields). Enabling an automatic transfer request, signifies that after the software writes a value into a specific shadow register, the update of the current value is going to be automatically enabled by hardware. This means for example, that the software does not need to write a 1_B to the GCSS.SySE field after a new period value is loaded into the shadow register.

The operation of this sub mode can be seen in Figure 223 - initially the software had configured the timer slice with an automatic request enable uppon the update of the period shadow register (ASPC is set to 1_B). One can see that after the update of the CC4yPRS (period shadow register) the SySS bitfield is automatically set by hardware, enabling a shadow transfer (in the second time slot the field ASCC is set to 1_B).

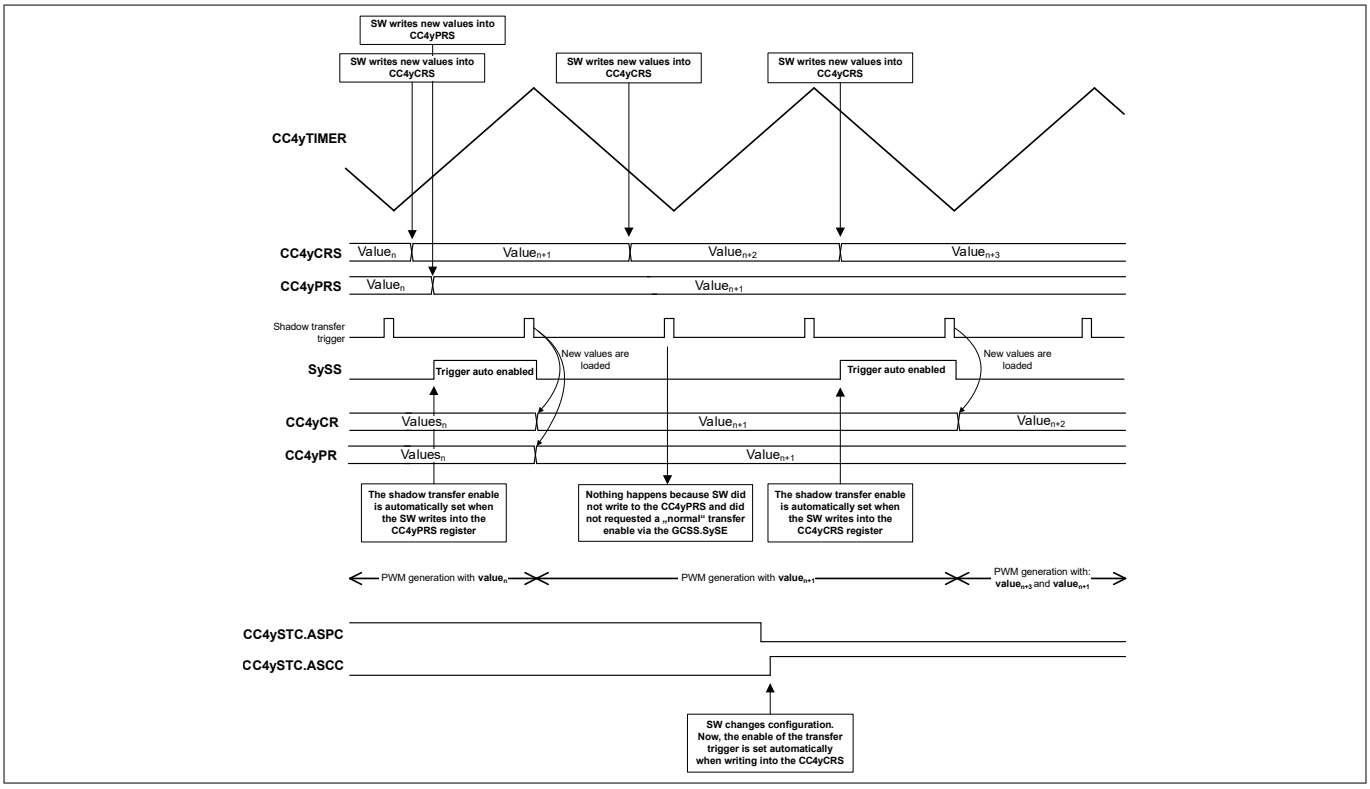


Figure 223 Shadow transfer timing example - center aligned mode with automatic request

- Cascaded Shadow Transfer

It is possible to cascade the shadow transfer operation throughout the CCU4 timer slices. The specific shadow transfer of a timer slice is cascaded with the adjacent timer slices, Figure 224.

To enable the cascaded shadow transfer function, the bitfield CC4ySTC.CSE of the specific timer slice needs to be set to 1_B.

The shadow transfer enable bits, still need to be set via SW for each of the individual slices, Figure 225.

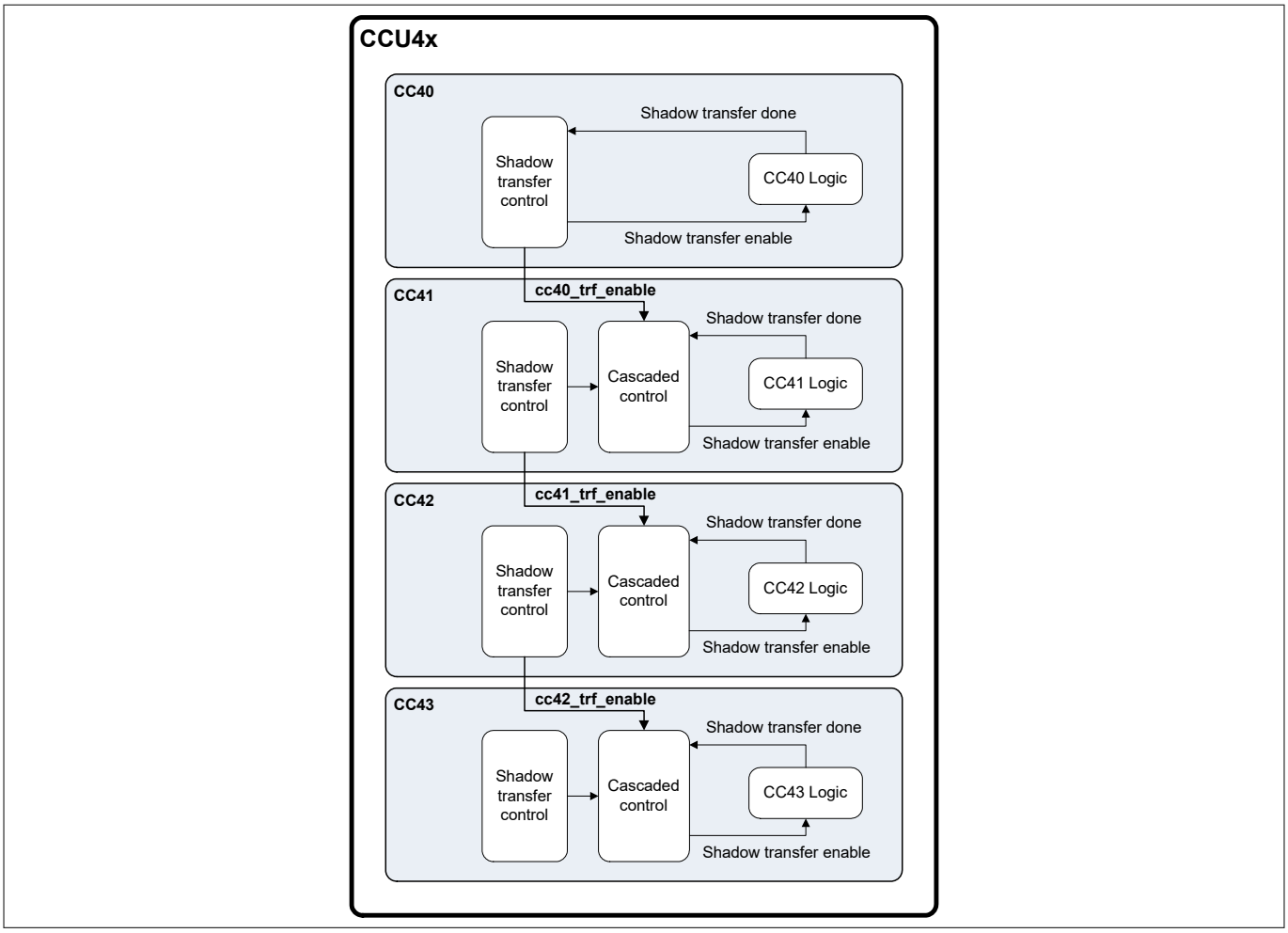


Figure 224 Cascaded shadow transfer linking

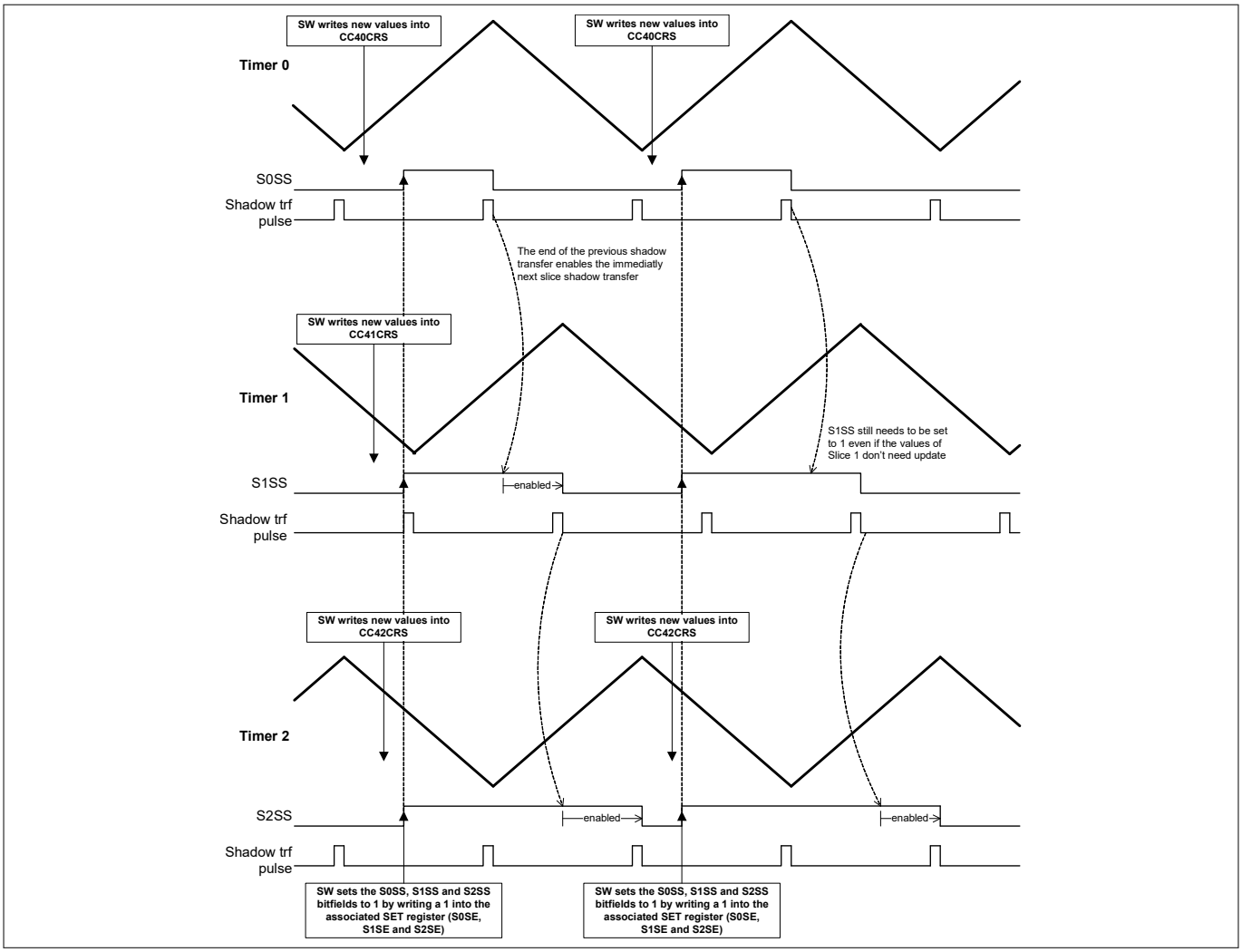


Figure 225 Cascade shadow transfer timing

***Note**: The shadow transfer enable bits - SySS, 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.*

21.2.4.8 PWM Active/Passive Rules

Like previously mentioned on Counting Modes Introduction, each CCU4 timer slice has a status bit (CC4yST) that contains the current state of the timer comparison logic that is used to generate the output PWM signal.

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, External Load.

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.

21.2.4.9 Output PWM Path

Each Timer Slice contains an output path, where the PWM output signal can be conditioned after the PWM status bit - CC4yST - Figure 226. Inside the output path, the polarity of the PWM signal can be inverted in relation to the status bit. Each output path offers two types of outputs:

- CCU4x.OUTy - PWM signal from Timer Slice y that can be conditioning or inverted
- CCU4x.STy - PWM signal from Timer Slice y that mirrors the PWM status bit - CC4yST.

The conditioning of the output path has three sources:

- External Modulation - conditioning controlled by an external signal
- Multi-Channel Mode - conditioning controlled by a dedicated timer slice input that normally is used in Brushless DC Motor Control patterns
- TRAP State - highest level of conditioning that is controlled by an external function and is normally linked to a system fault, for example, overcurrent protection

Each of these conditioning sources are explained in detailed in their dedicated section.

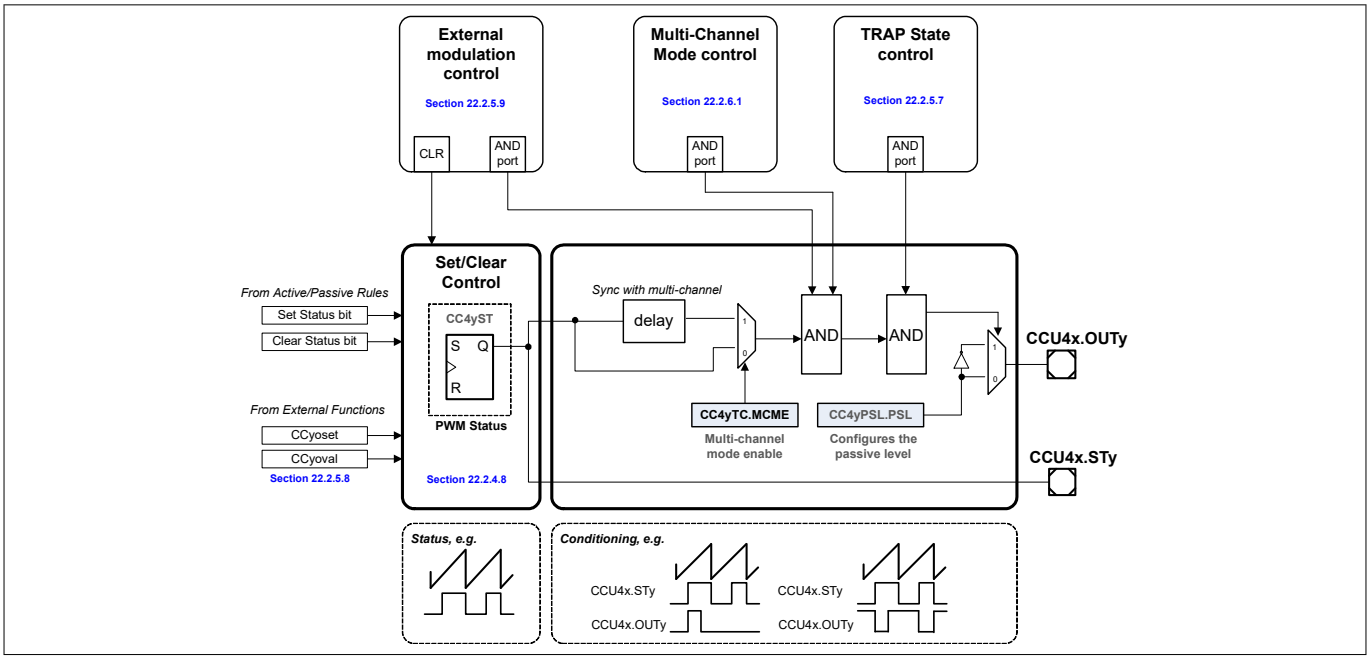


Figure 226 PWM output path

21.2.5 Timer Slice External Functions

Each CCU4 timer slice has the possibility of using up to three different input events, see Timer Slice Input Selector. These three events can then be mapped to Timer Slice functions (the full set of available functions is described at Timer Slice Connection Matrix)

These events can be mapped to any of the CCU4x.INy[BV:AA] 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 (for example, input X triggers event 0 with rising edge and triggers event 1 with the falling edge).

21.2.5.1 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 CC4yINS1.EVxIS field and indicating the active edge of the signal on the CC4yINS2.EVxEM field.

This event should be then mapped to the start or stop functionality by setting the CC4yCMC.STRTS (for the start) or the CC4yCMC.ENDS (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 CC4yINS2.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):

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 227 to Figure 230 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.

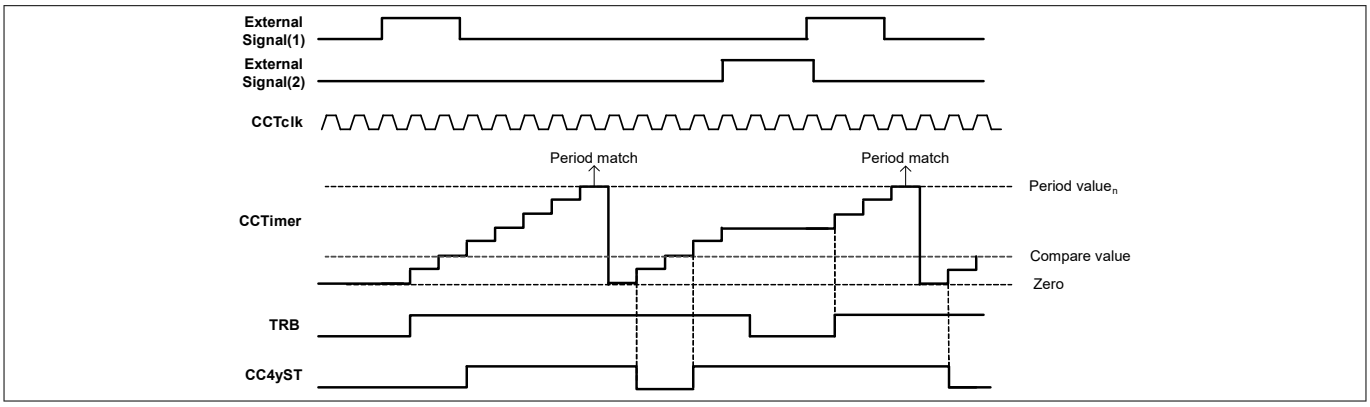


Figure 227 Start (asstart)/ stop (as stop)-CC4yTC.STRM = 0_B, CC4yTC.ENDM = 00_B

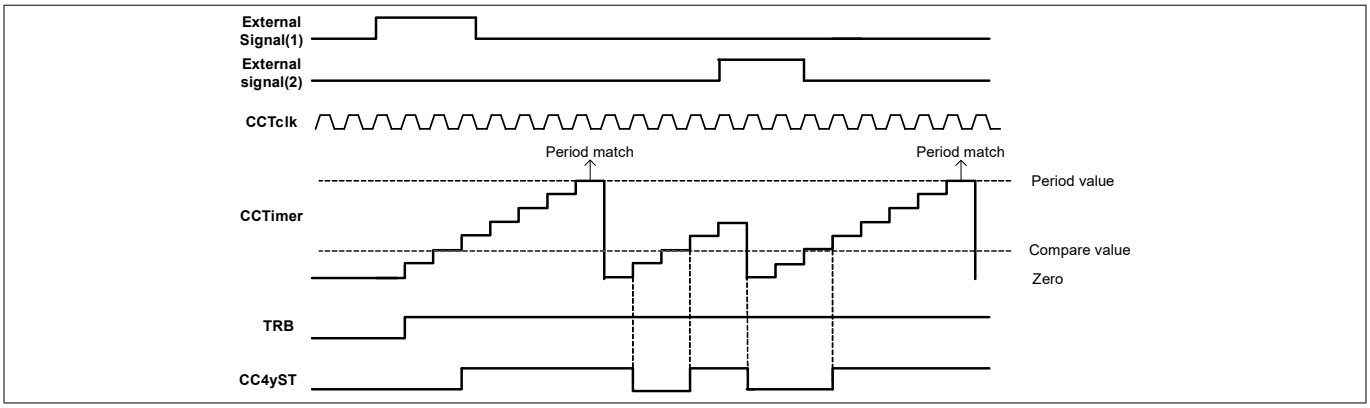


Figure 228 Start (as start)/ stop (as flush) - CC4yTC.STRM = 0_B, CC4yTC.ENDM = 01_B

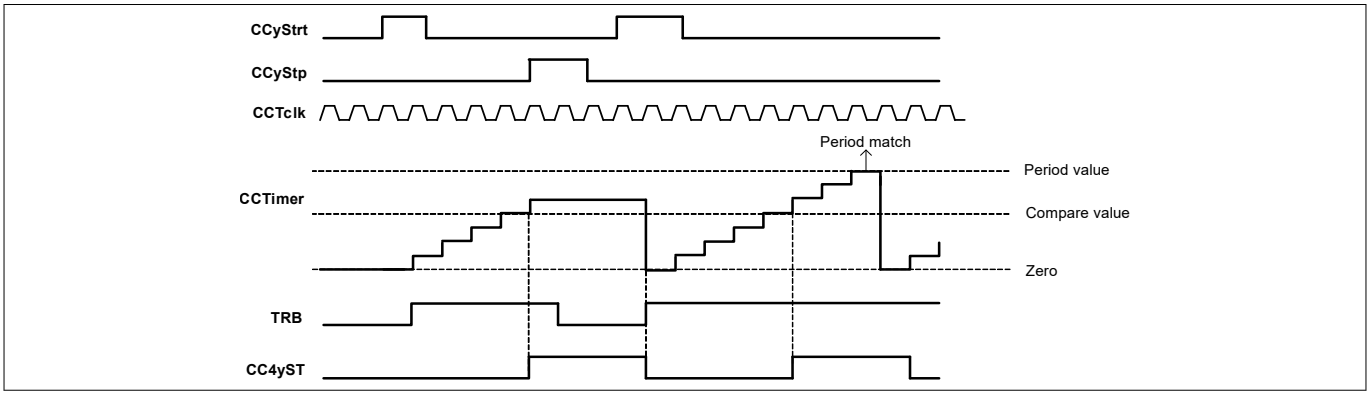


Figure 229 Start (as flush and start)/ stop (as stop) - CC4yTC.STRM = 1_B, CC4yTC.ENDM = 00_B

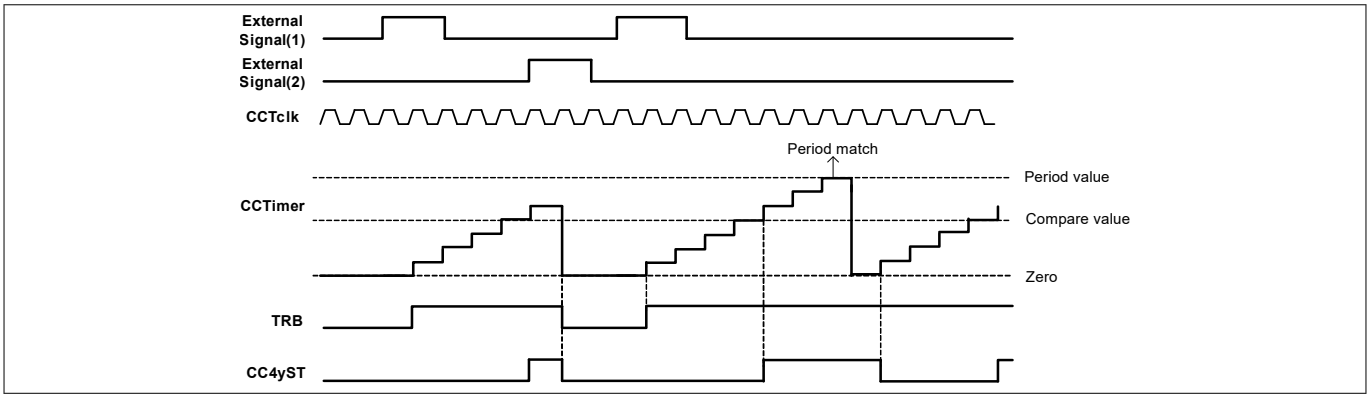


Figure 230 Start (as start)/ stop (as flush and stop) - CC4yTC.STRM = 0_B, CC4yTC.ENDM = 10_B

21.2.5.2 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 CC4yINS1.EVxIS field and indicating the active level of the signal on the CC4yINS2.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 CC4yINS2.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 PWM Active/Passive Rules.

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. Using an external signal to perform the up/down counting function and configuring the event as active LOW means that the timer is counting up when the signal is HIGH and counting down when LOW (this is to match the CDIR value).

Figure 231 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 CC4yINS2.EVxLM = 0_B. When the operation is switched, then the user should set CC4yINS2.EVxLM = 1_B.*

***Note**:* *Using an external counting direction control, sets the slice in edge aligned mode.*

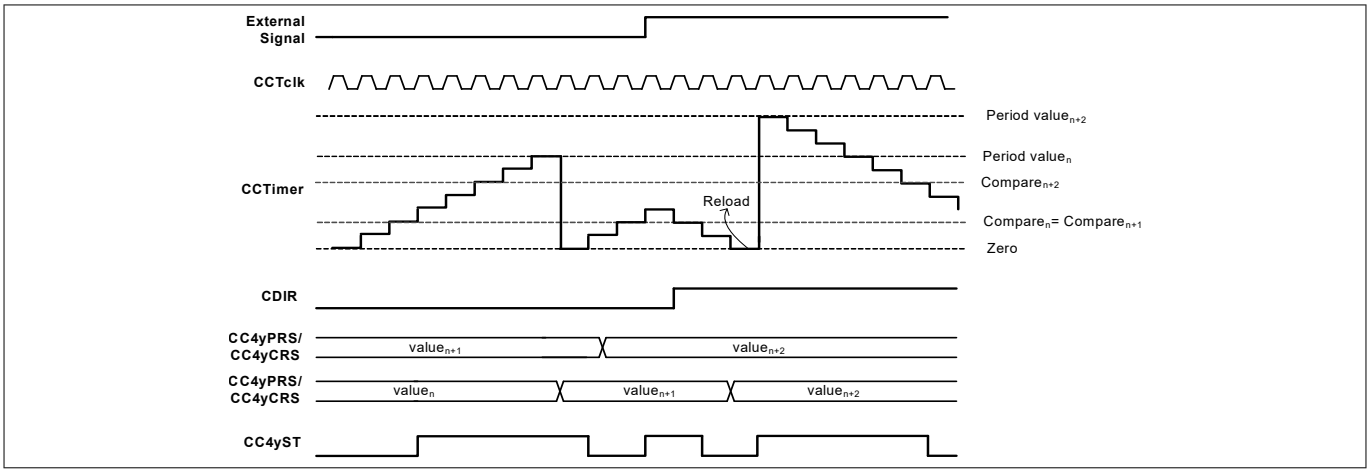


Figure 231 External counting direction

21.2.5.3 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 CC4yINS1.EVxIS register and indicating the active level of the signal on the CC4yINS2.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 CC4yINS2.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 232 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.

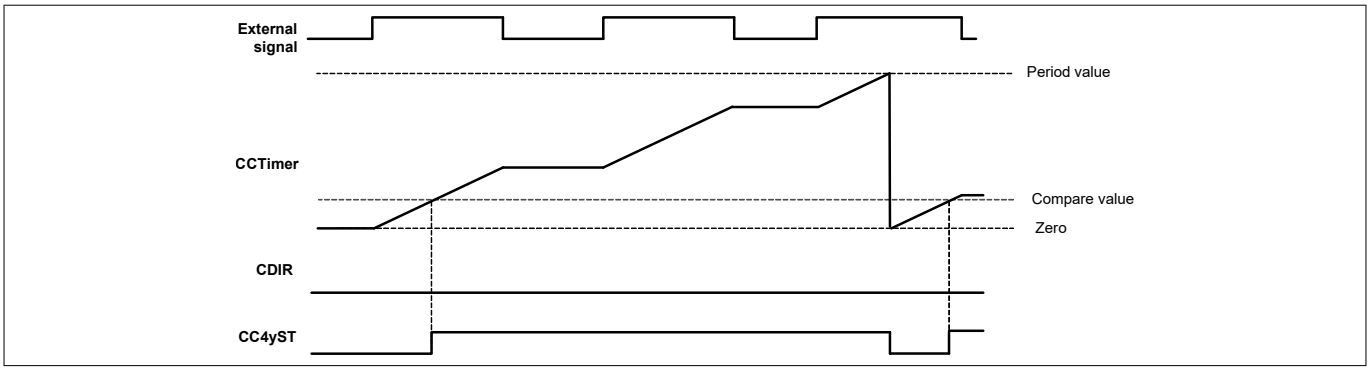


Figure 232 External gating

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.

21.2.5.4 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 CC4yINS1.EVxIS register and indicating the active edge of the signal on the CC4yINS2.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 CC4yINS2.EVxEM.

One can select the rising, falling or both edges to perform a count. On Figure 233, 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.

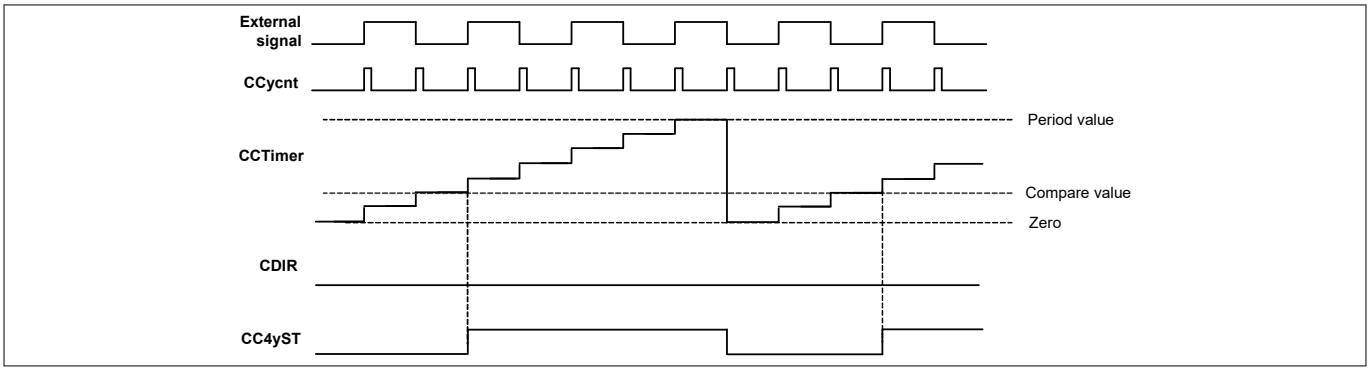


Figure 233 External count

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.

21.2.5.5 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 CC4yINS1.EVxIS register and indicating the active edge of the

signal on the CC4yINS2.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 CC4yINS2.EVxEM.

On figure Figure 234, 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.

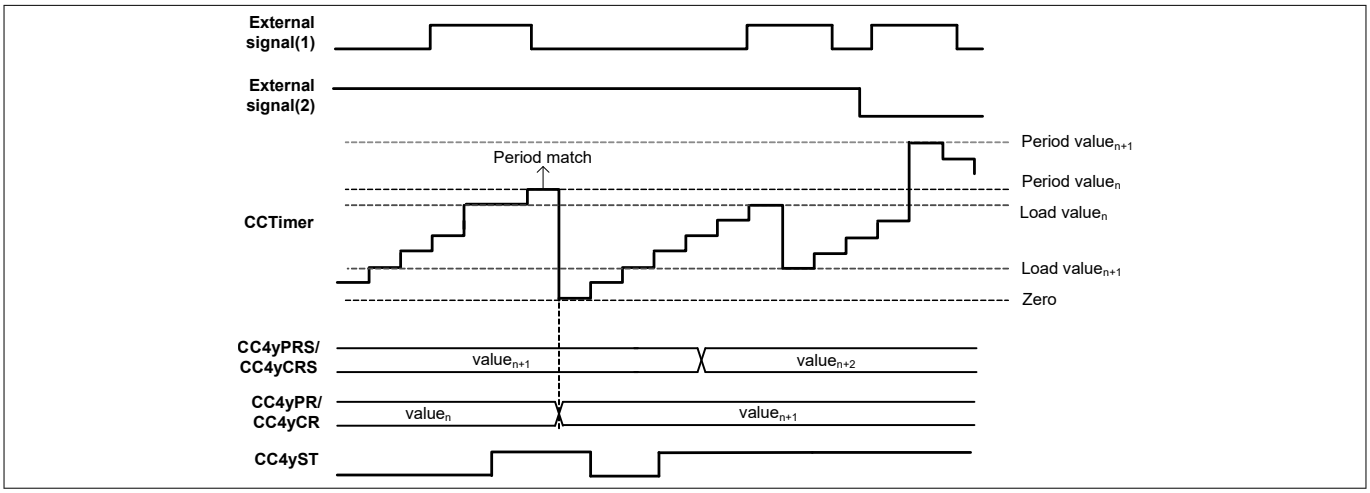


Figure 234 External load

21.2.5.6 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 237: 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 CC4yINS1.EVxIS register and indicating the active edge of the signal on the CC4yINS2.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 CC4yINS2.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 (by reading the specific capture register or by reading the extended capture read value, see Capture Extended Read Back Mode).

The capture of a new value into a specific capture registers is dictated by the status of the full flag as follows:

$$CC4yC1V_{capt} = NOT\left(CC4yC1V_{full_flag} \quad AND \quad CC4yC0V_{full_flag}\right) \tag{39}$$

$$CC4yC0V_{capt} = CC4yC1V_{full_flag} \quad AND \quad NOT\left(CC4yC0V_{full_flag}\right) \tag{40}$$

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 235, 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.

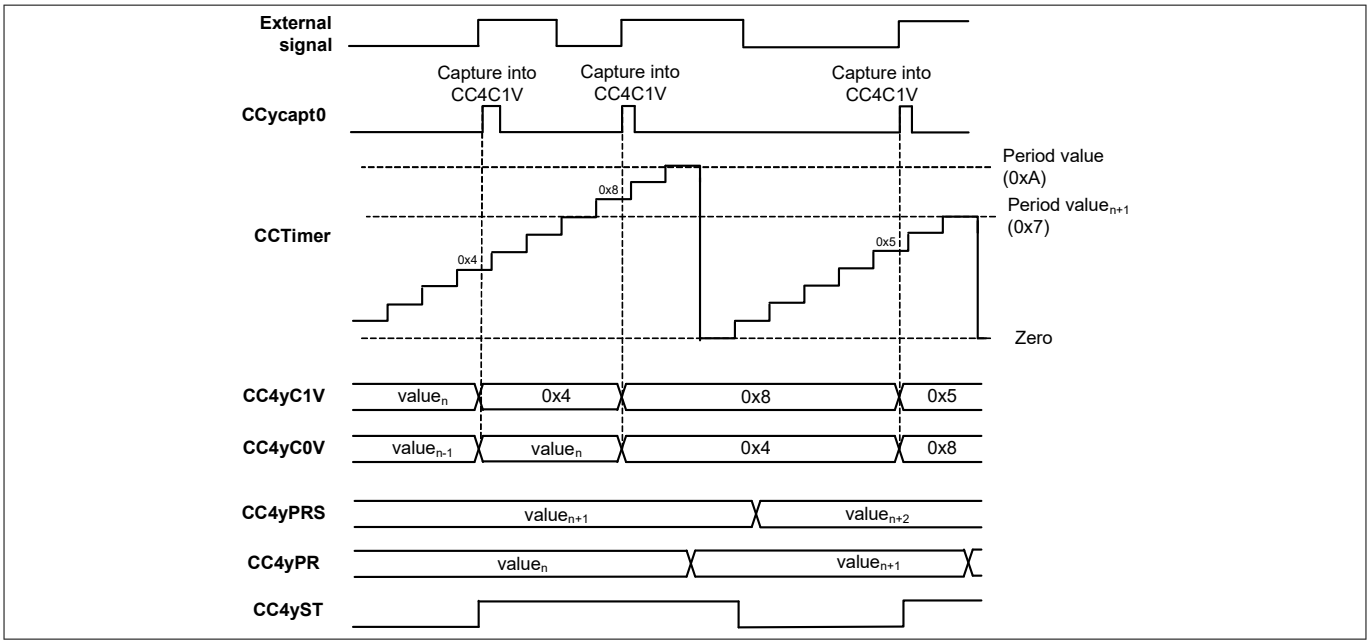


Figure 235 External capture - CC4yCMC.CAP0S != 00_B, CC4yCMC.CAP1S = 00_B

On Figure 236, 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 Capture Mode Usage, for a capture mode usage description.

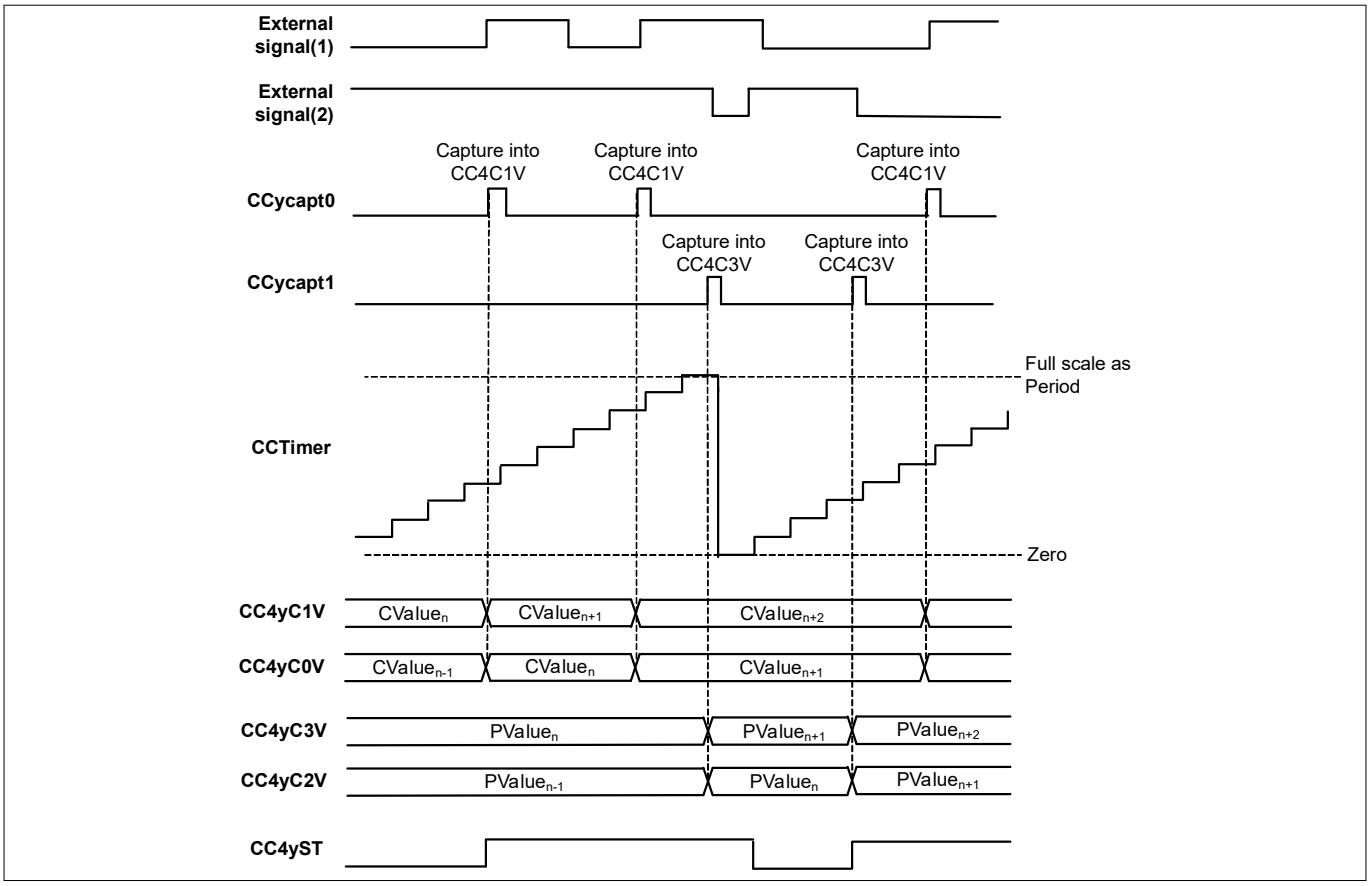


Figure 236 External capture - CC4yCMC.CAP0S != 00_B, CC4yCMC.CAP1S != 00_B

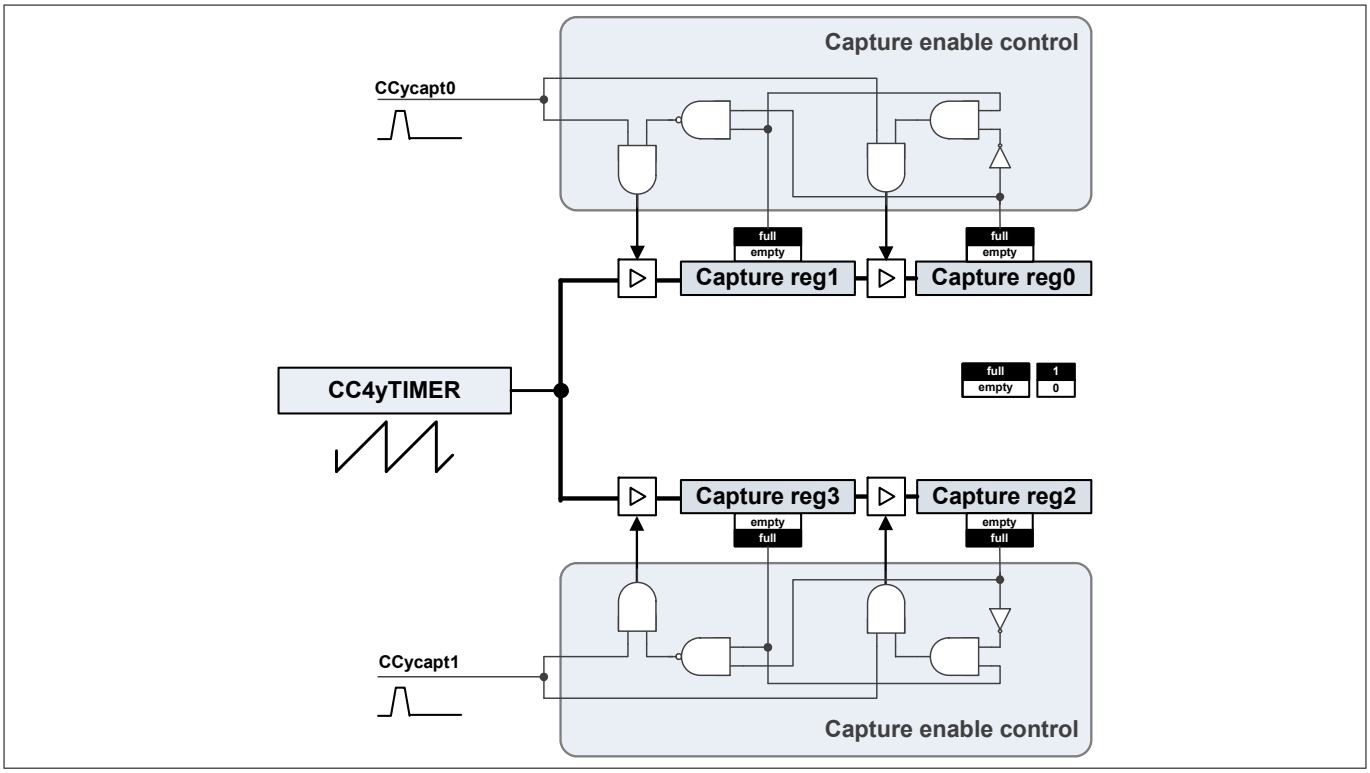


Figure 237 Slice capture logic

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 239. The function that controls the capture is the CCcapt1.

The capture logic follows the same structure shown in Figure 237 but extended to a four register chain, seeFigure 238 . 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):

$$CC4yC3V_{capt} = NOT\left(CC4yC3V_{full_flag} \quad AND \quad CC4yC2V_{full_flag} \quad AND \quad CC4yC2V_{full_flag} \quad AND \quad CC4yC1V_{full_flag}\right) \tag{41}$$

$$CC4yC2V_{capt} = CC4yC3V_{full_flag} \quad AND \quad NOT\left(CC4yC2V_{full_flag} \quad AND \quad CC4yC1V_{full_flag} \quad AND \quad CC4yC0V_{full_flag}\right) \tag{42}$$

$$CC4yC1V_{capt} = CC4yC2V_{full_flag} \quad AND \quad NOT\left(CC4yC1V_{full_flag} \quad AND \quad CC4yC0V_{full_flag}\right) \tag{43}$$

$$CC4yC0V_{capt} = CC4yC1V_{full_flag} \quad AND \quad NOT\left(CC4yC0V_{full_flag}\right) \tag{44}$$

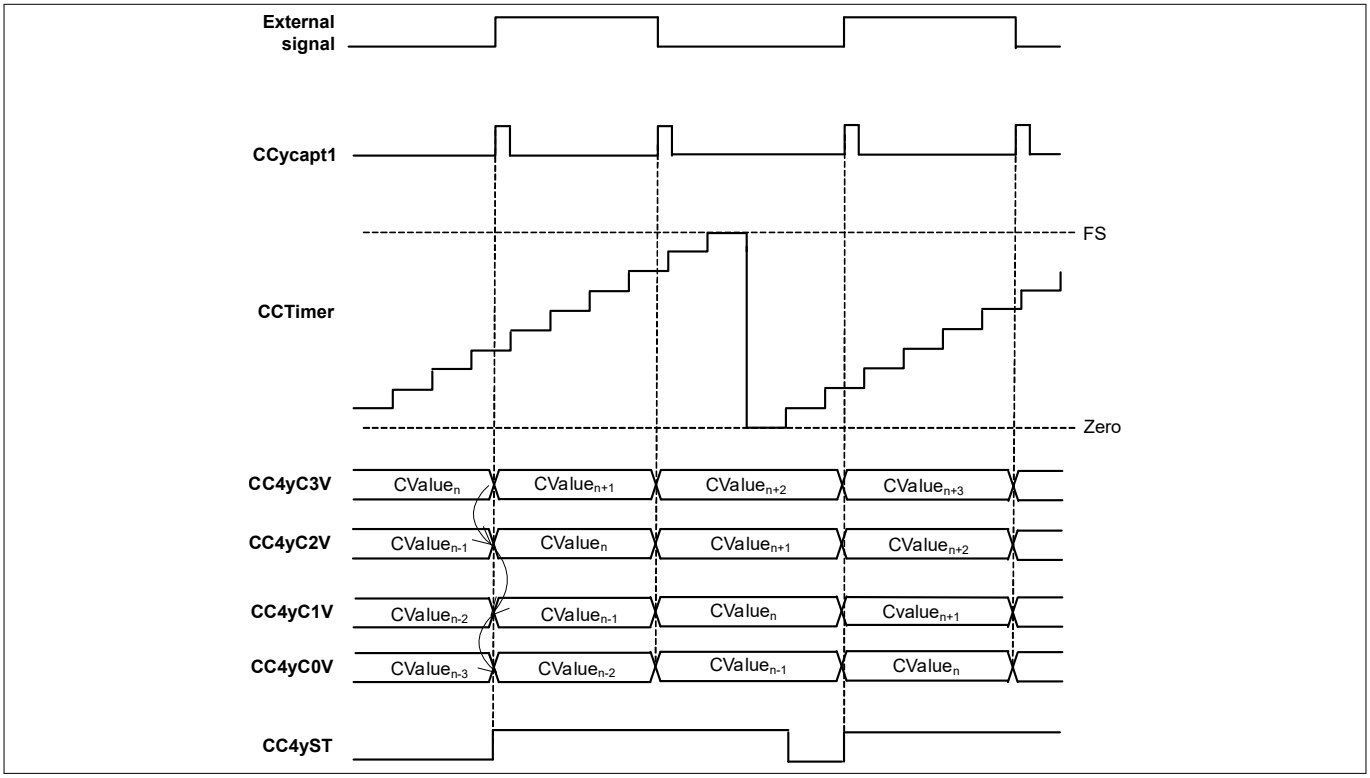


Figure 238 External Capture - CC4yTC.SCE = 1_B

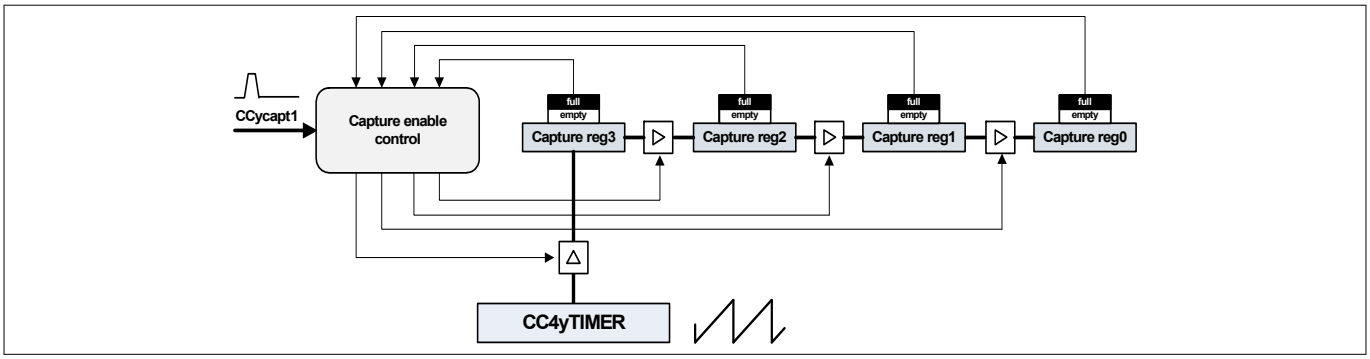


Figure 239 Slice Capture Logic - CC4yTC.SCE = 1_B

21.2.5.7 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 CC4yINS1.EV2IS register and indicating the active level of the signal on the CC4yINS2.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 240:

- 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.

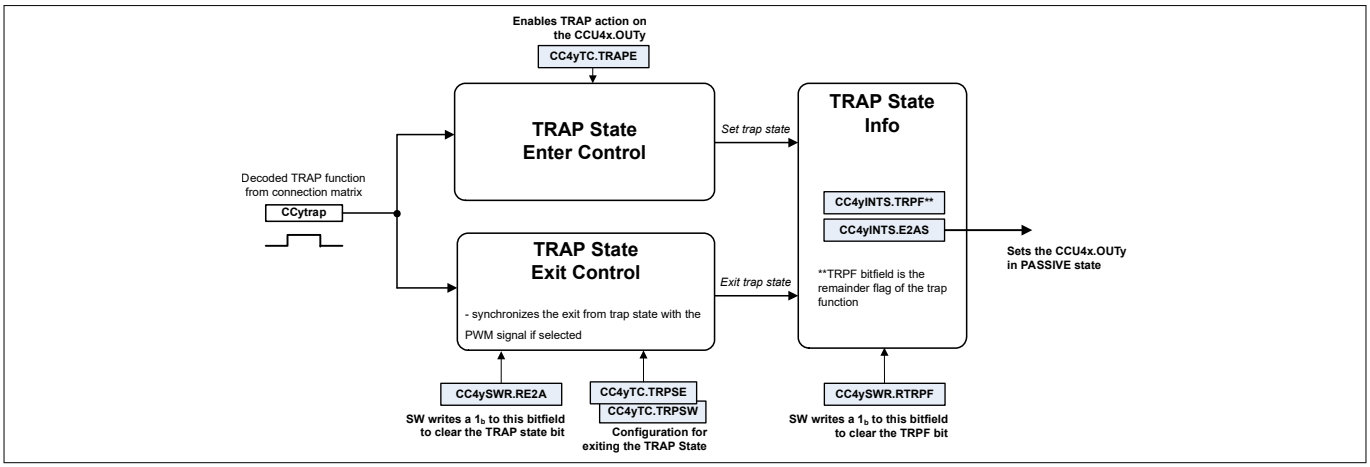


Figure 240 Trap control diagram

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 241.

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

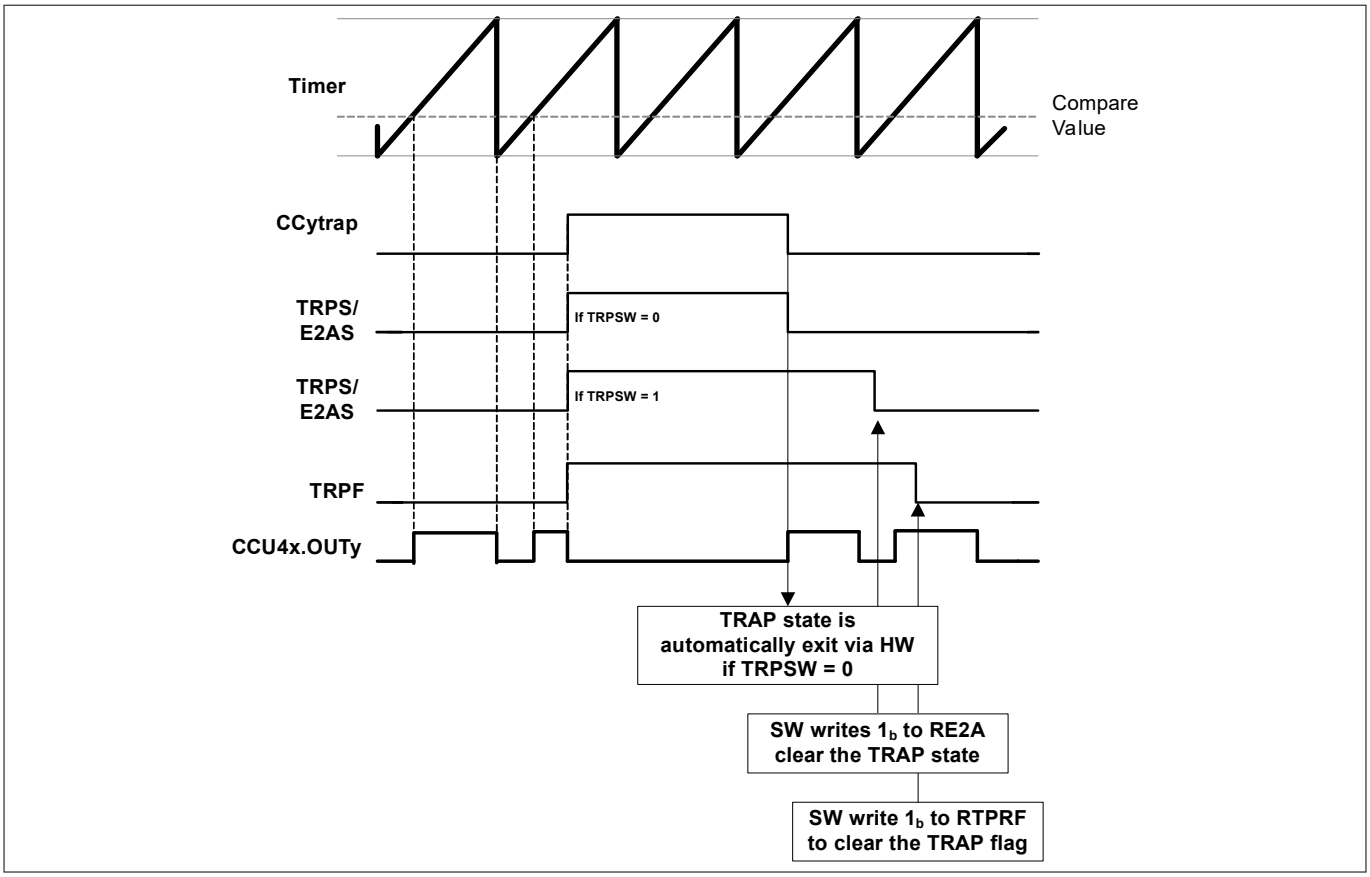


Figure 241 Trap timing diagram, CC4yPSL.PSL = 0_B (output passive level is 0_B)

It is also possible to synchronize the exiting of the TRAP state with the PWM signal, Figure 242. This function is enabled when the bitfield CC4yTC.TRPSE = 1_B.

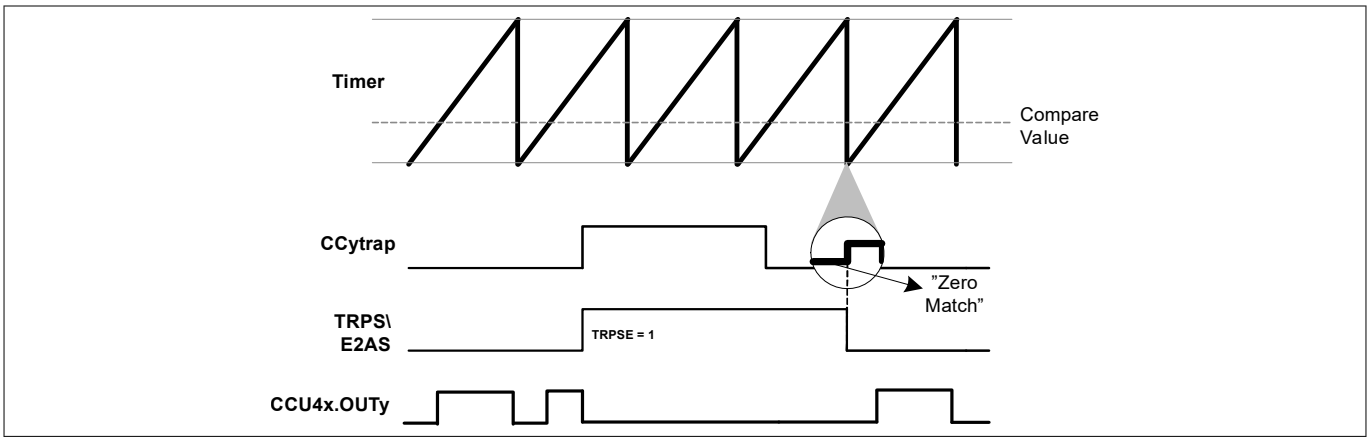


Figure 242 Trap synchronization with the PWM signal, CC4yTC.TRPSE = 1_B

21.2.5.8 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 CC4yINS1.EV1IS register and indicating the active edge of the signal on the CC4yINS2.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 CC4yINS1.EV2IS register. The CC4yINS2.EV2LM register should be set to 0B 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 243 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.

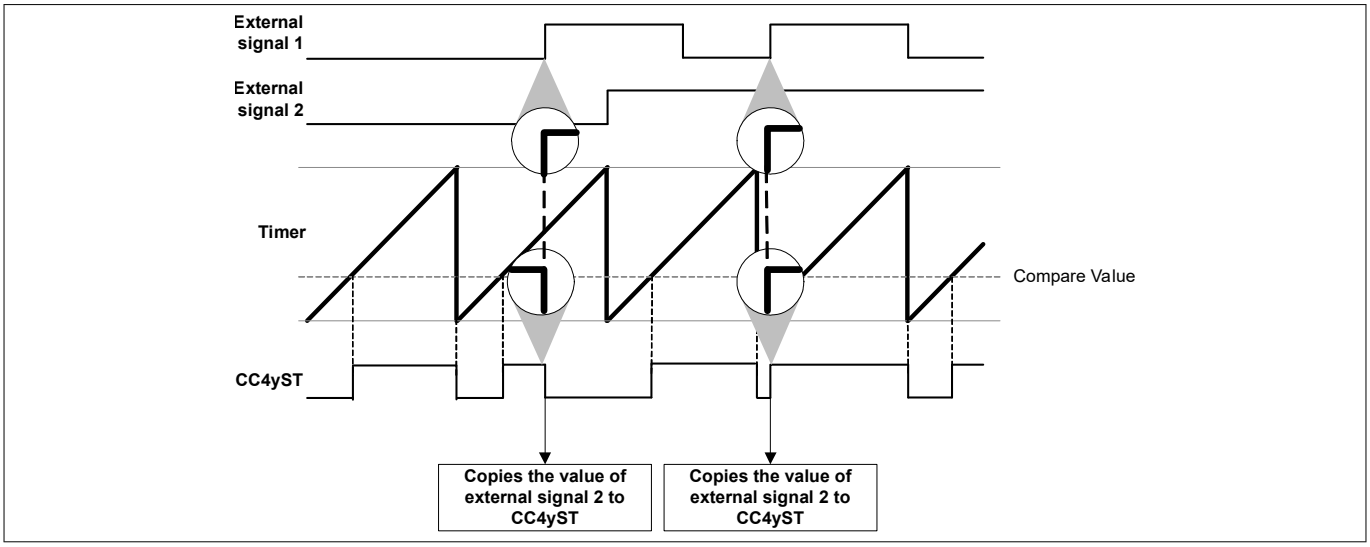


Figure 243 Status bit override

21.2.5.9 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 CC4yINS1.EVxIS register and indicating the active level of the signal on the CC4yINS2.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 CC4yINS2.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 244, 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 245.

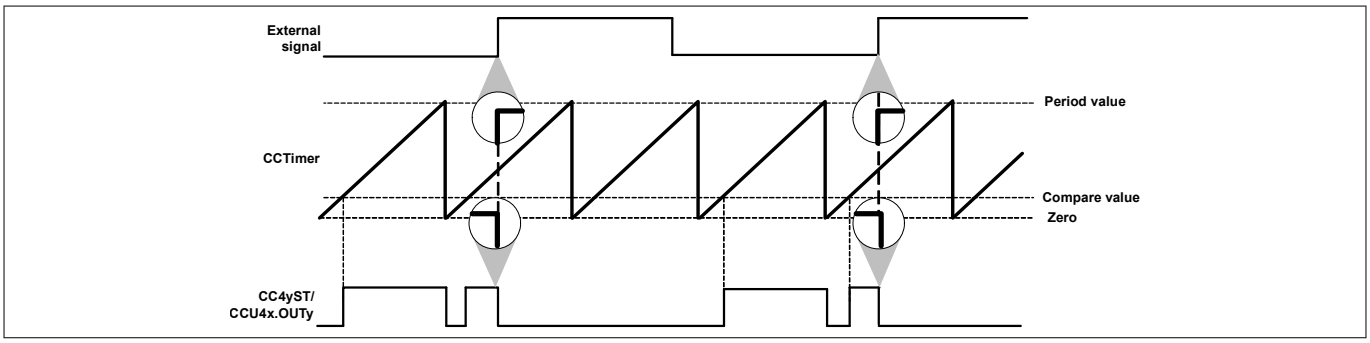


Figure 244 External modulation clearing the ST bit - CC4yTC.EMT = 0_B

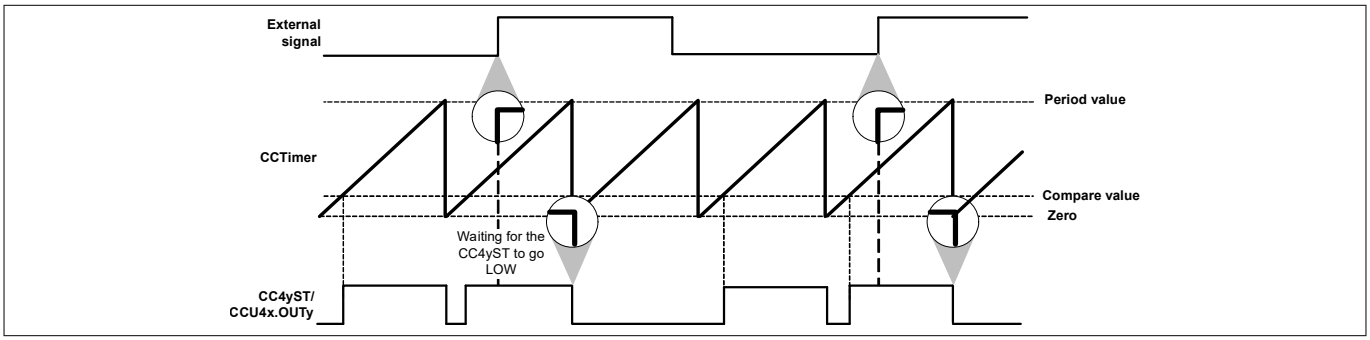


Figure 245 External modulation clearing the ST bit - CC4yTC.EMT = 0_B, CC4yTC.EMS = 1_B

On Figure 246, 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, CC4yINS2.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.

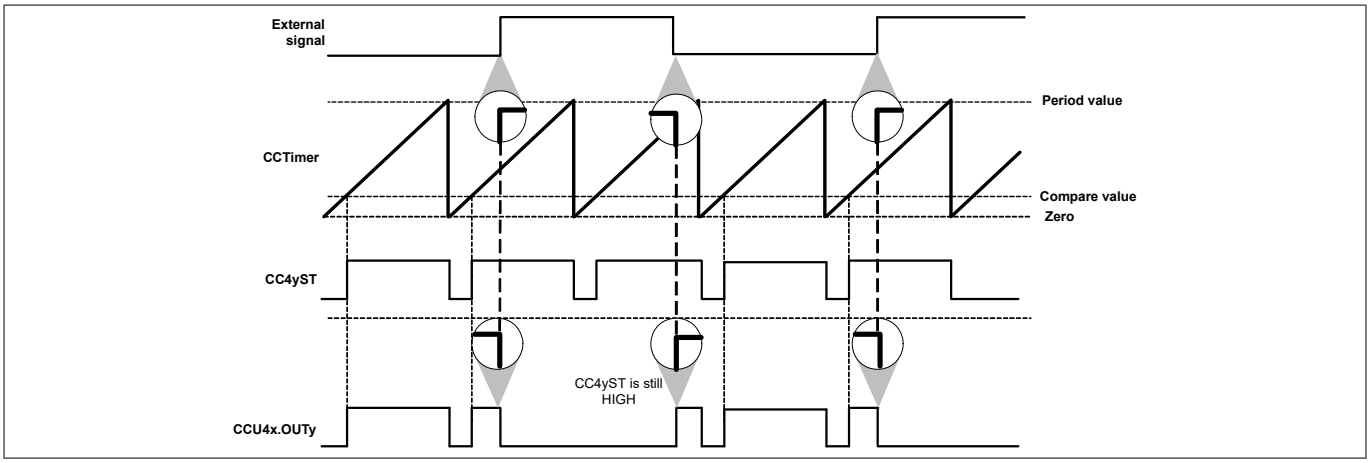


Figure 246 External modulation gating the output - CC4yTC.EMT = 1_B

21.2.6 Timer Slice Advanced Functions

In the following sub sections several advanced functions present in each CC4y slice are described. One should notice that each of the functions is present and works in the same manner for every CCU4 timer slice.

21.2.6.1 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, for example, 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 Interconnects 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 247. This synchronocity feature can be enabled by using some specific modules and therefore one should address Interconnects to check which module is controlling the CCU4x.MCIy inputs.

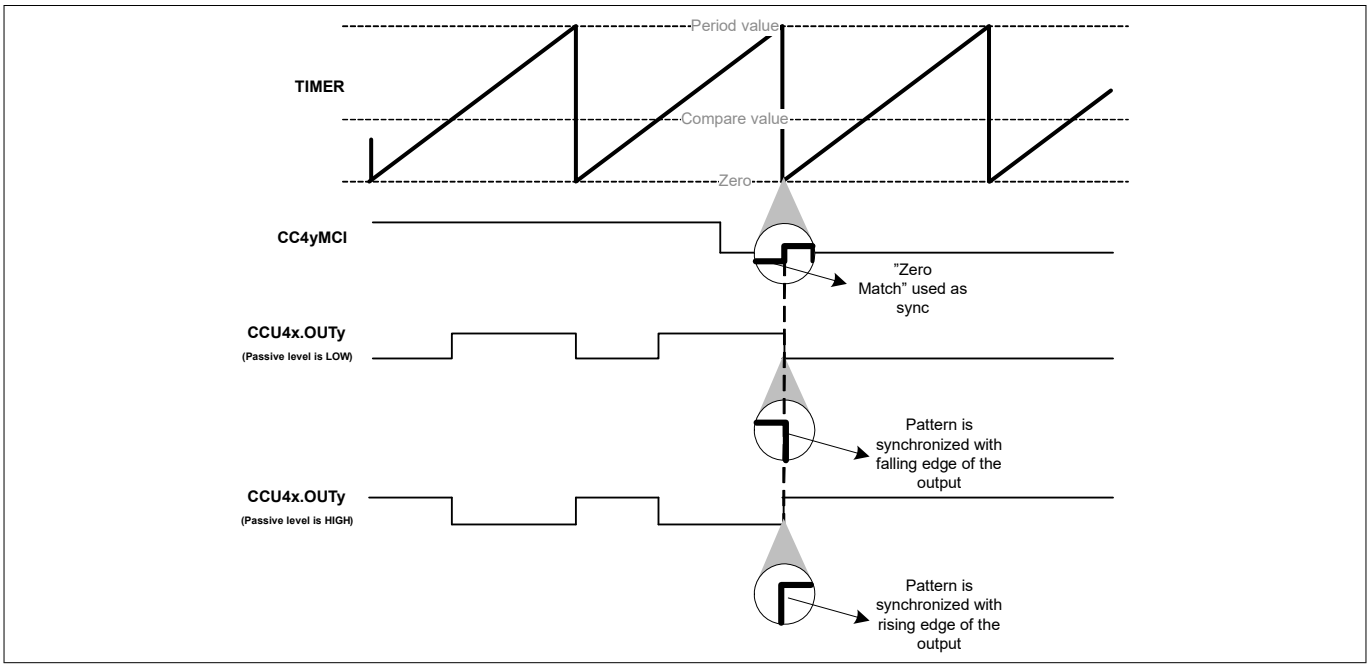


Figure 247 Multi-Channel pattern synchronization

Figure 248 shows the usage of the Multi-Channel mode in conjunction with all four Timer Slices inside the CCU4.

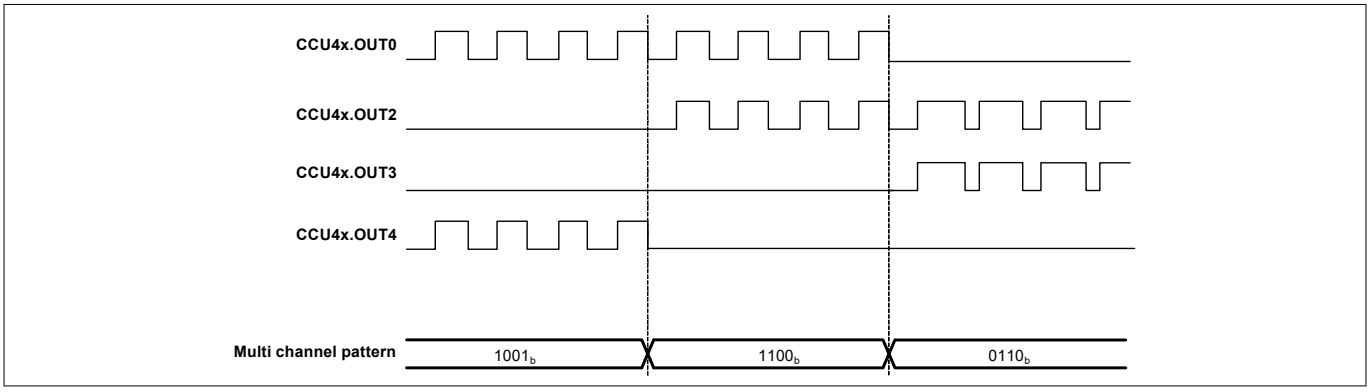


Figure 248 Multi-Channel mode for multiple Timer Slices

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 249.

The multi pattern input synchronization can be seen on Figure 250. 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 250.

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.

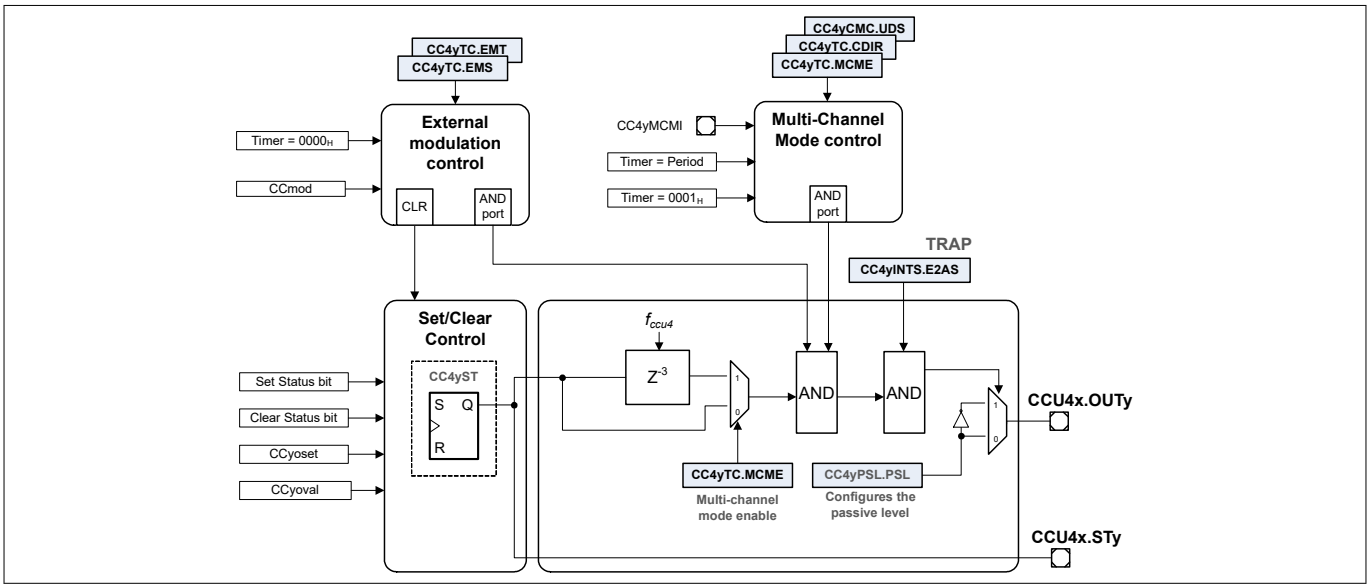


Figure 249 Multi-Channel mode output patrh

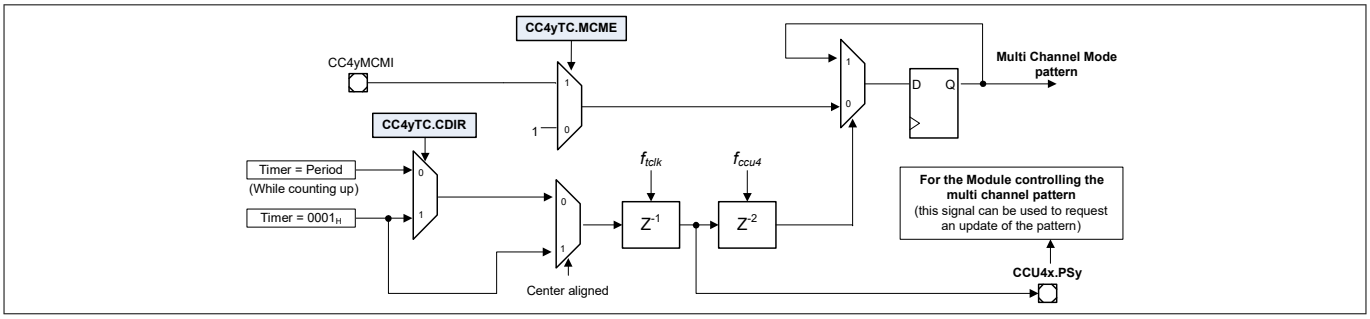


Figure 250 Multi-Channel Mode Control Logic

21.2.6.2 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 251.

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

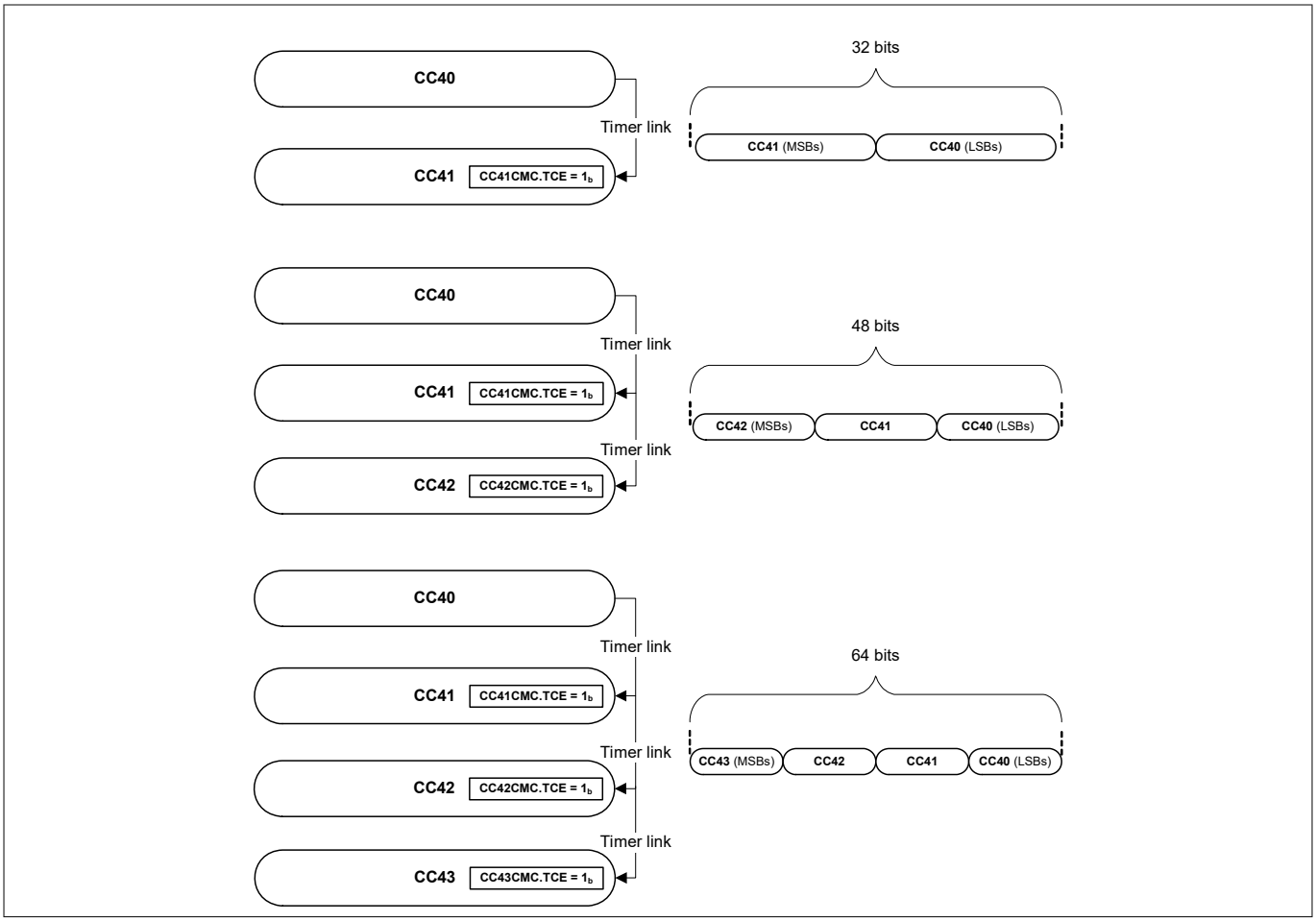


Figure 251 Timer Concatenation Example

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 252.

***Note**: For all cases CC40 and CC43 are not considered adjacent slices*

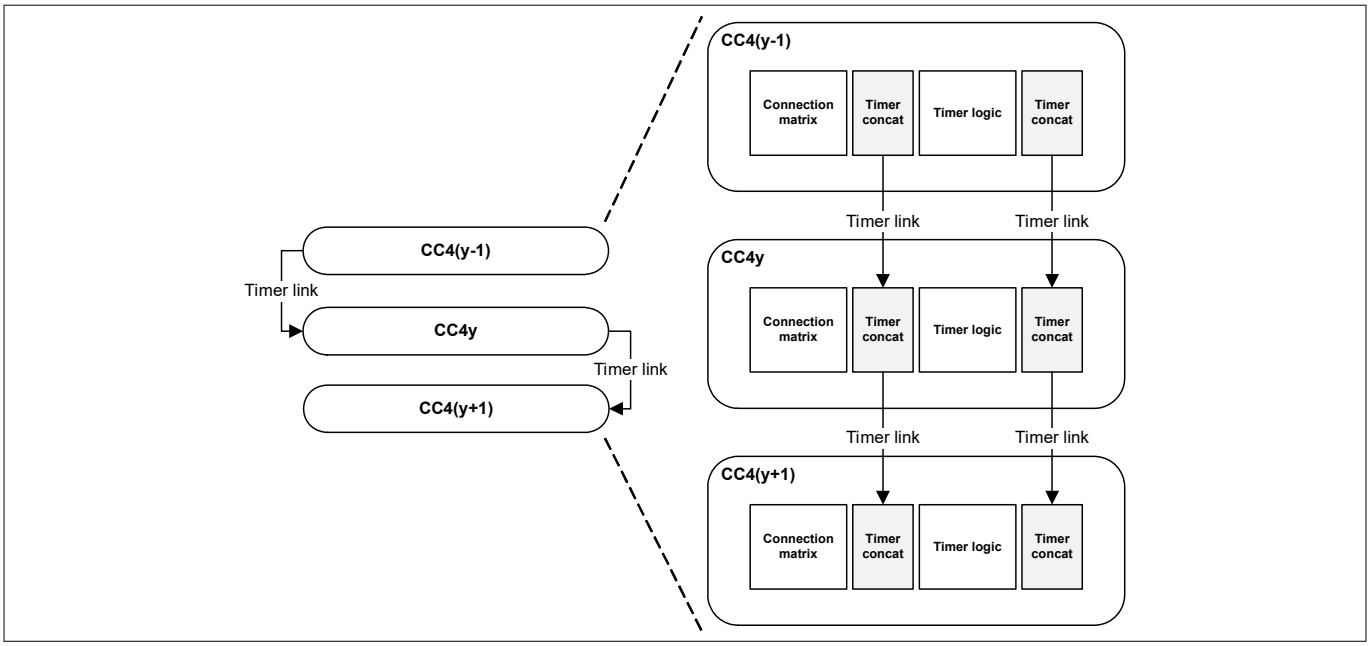


Figure 252 Timer Concatenation Link

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 252. 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 253.

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.

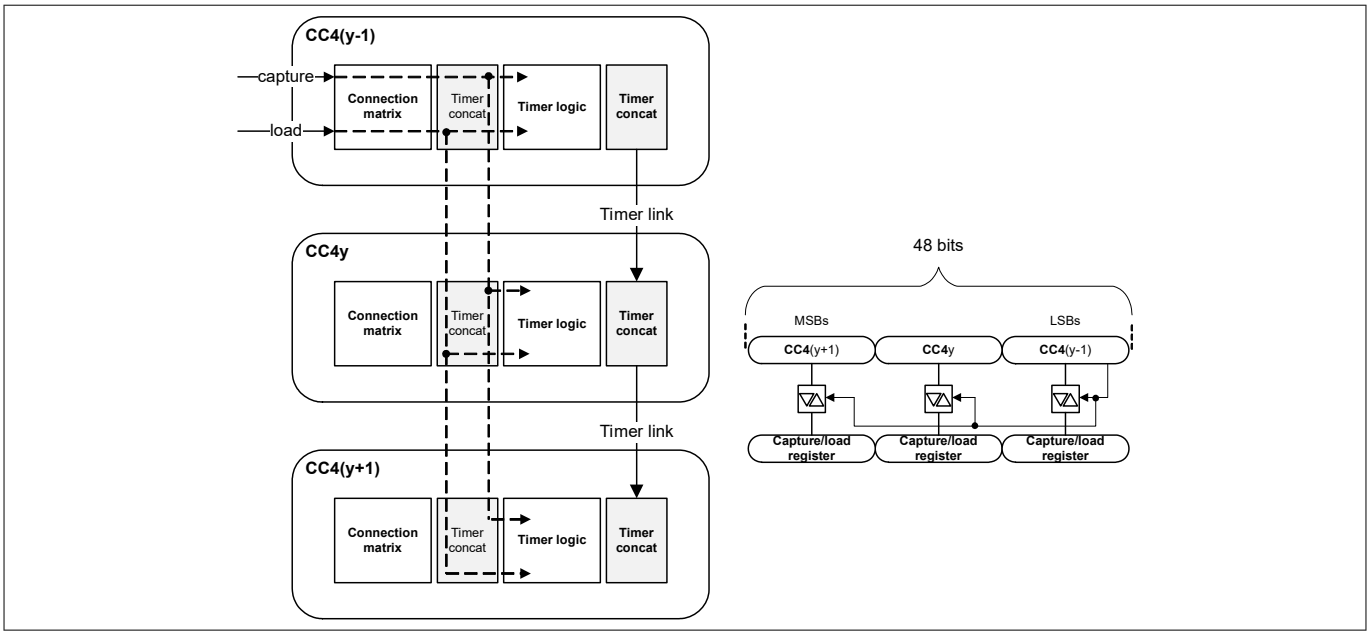


Figure 253 Capture/Load Timer Concatenation

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 254.

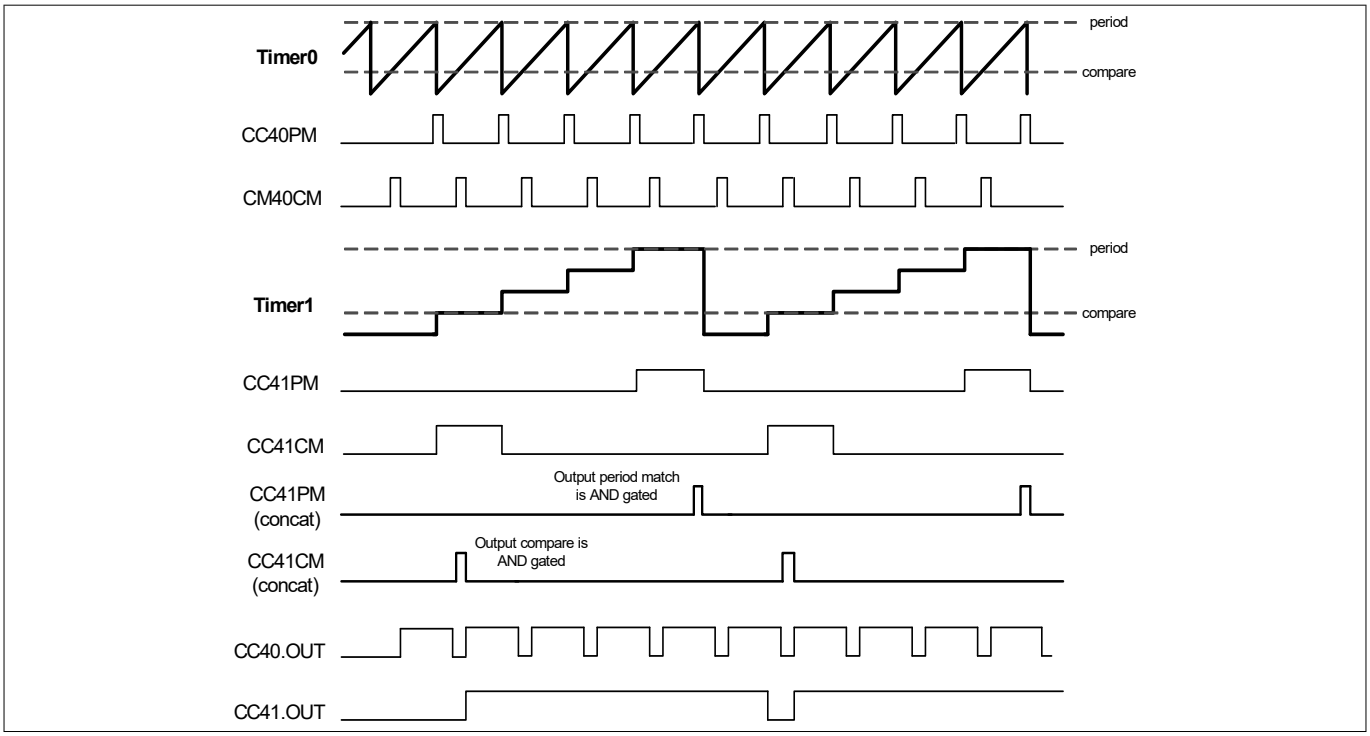


Figure 254 32 bit concatenation timing diagram

***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 255 gives an overview of the timer concatenation logic. Notice that all the mechanism is controlled solely by the CC4yCMC.TCE bitfield.

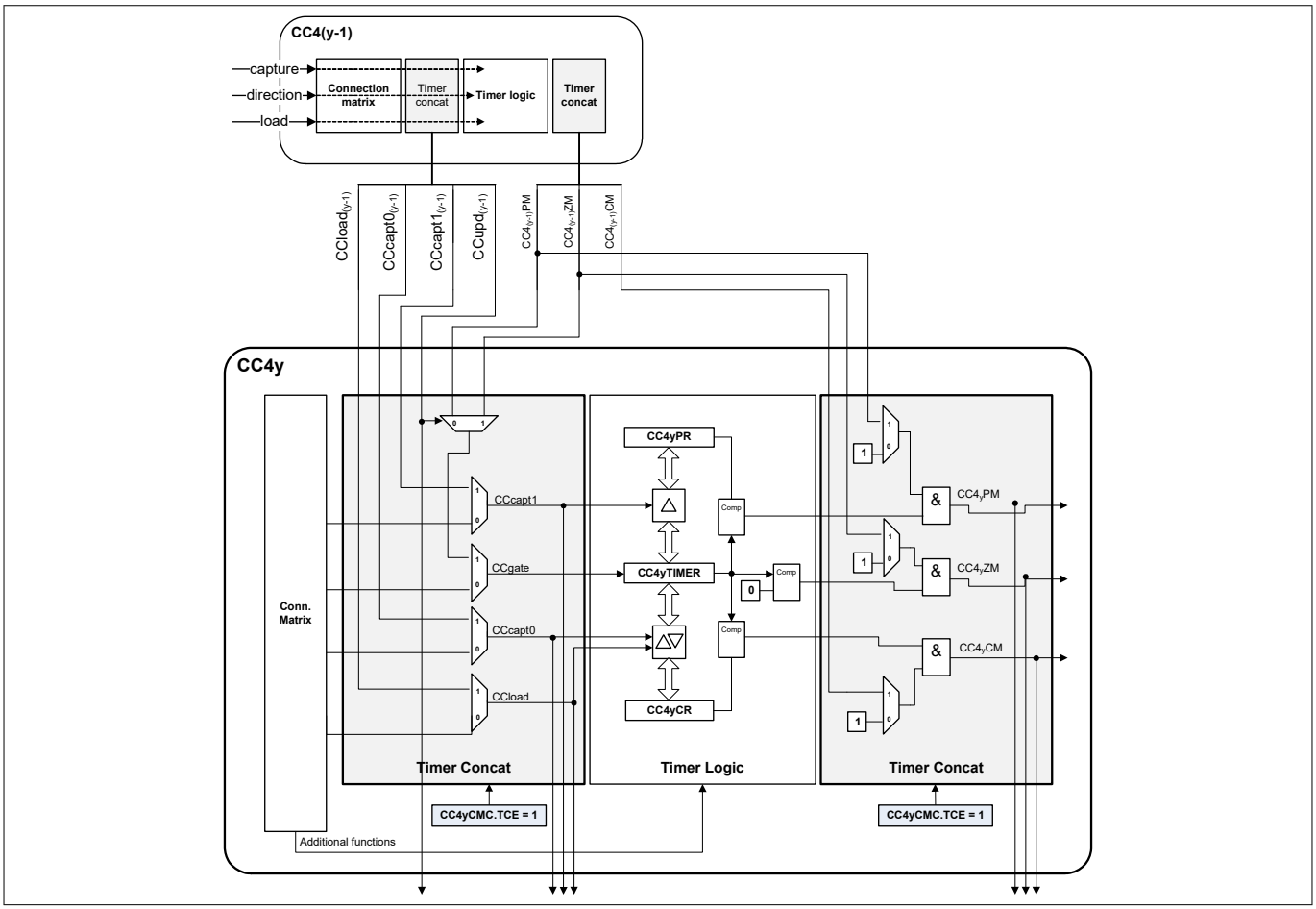


Figure 255 Timer concatenation control logic

21.2.6.3 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 256.

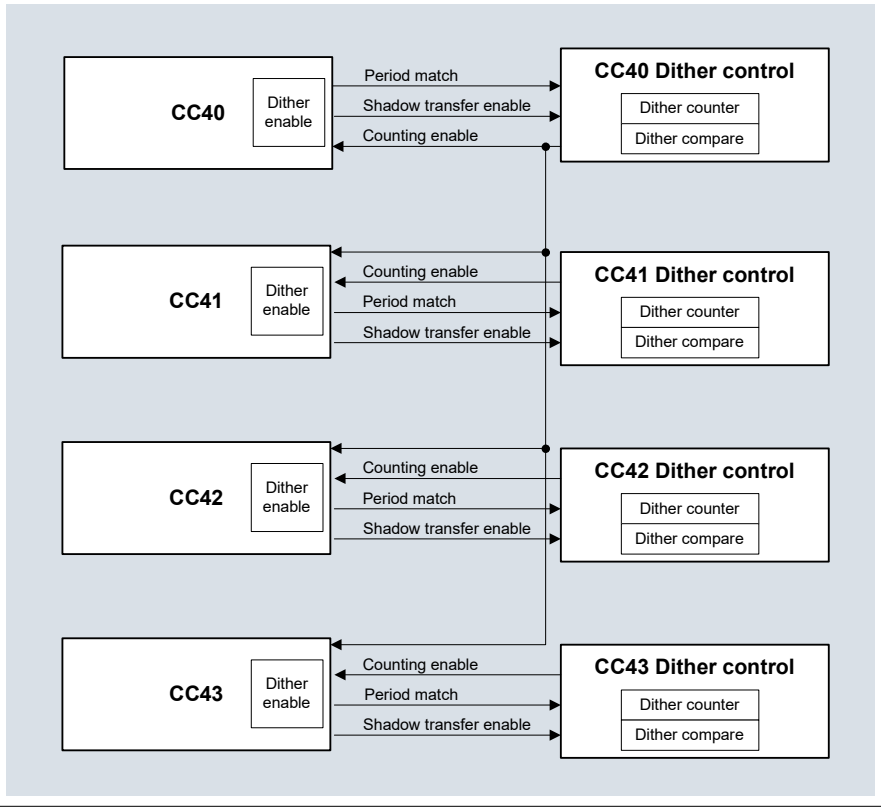


Figure 256 Dither structure overview

The dither control unit contains a 4 bit counter and a compare value. The counter works in a bit reverse mode so the distribution of increments stays uniform over 16 counter periods, see Table 159.

Table 159 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

(table continues...)

Table 159 (continued) Dither bit reverse counter

counter[3]	counter[2]	counter[1]	counter[0]
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 160) for one clock cycle.

Table 160 Dither modes

DITHE[1]	DITHE[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 257.

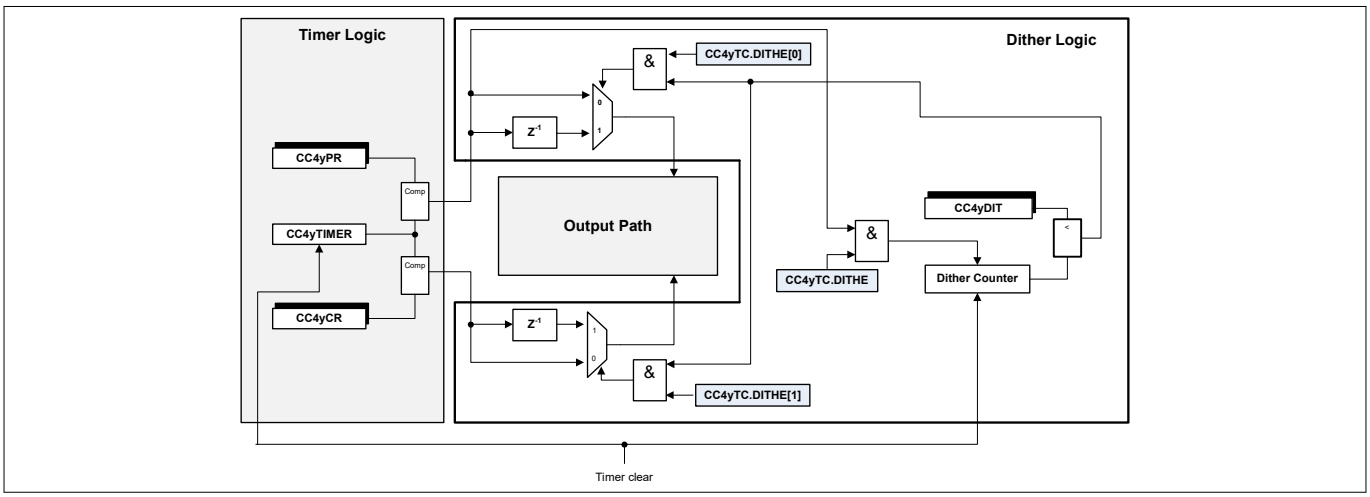


Figure 257 Dither control logic

Figure 258 to Figure 263 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).

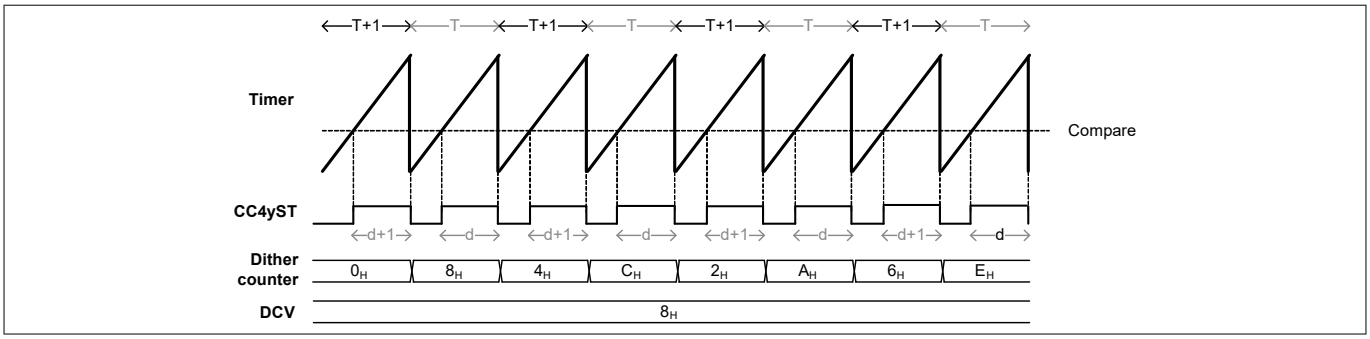


Figure 258 Dither timing diagram in edge aligned - CC4yTC.DITHE = 01_B

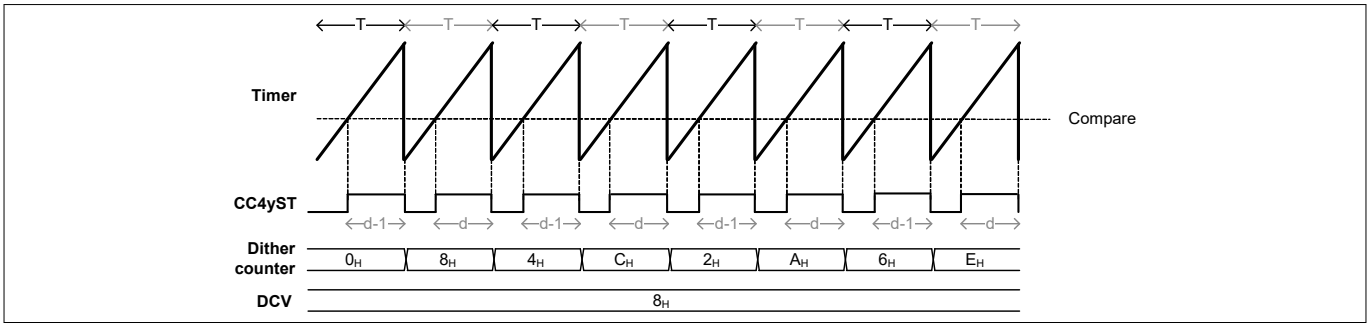


Figure 259 Dither timing diagram in edge aligned - CC4yTC.DITHE = 10_B

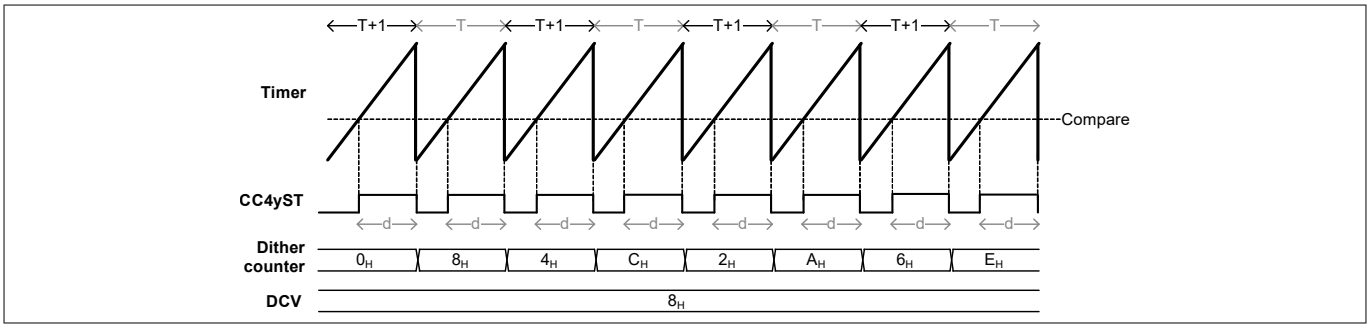


Figure 260 Dither timing diagram in edge aligned - CC4yTC.DITHE = 11_B

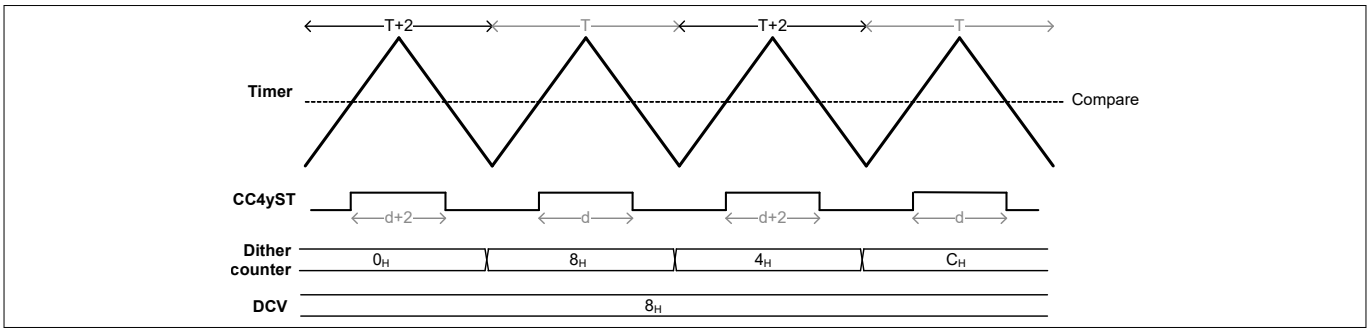


Figure 261 Dither timing diagram in center aligned - CC4yTC.DITHE = 01_B

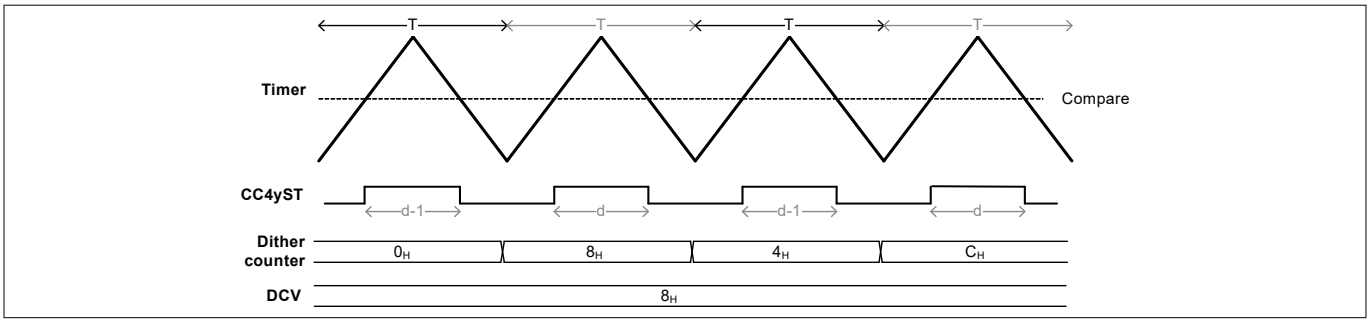


Figure 262 Dither timing diagram in center aligned - CC4yTC.DITHE = 10_B

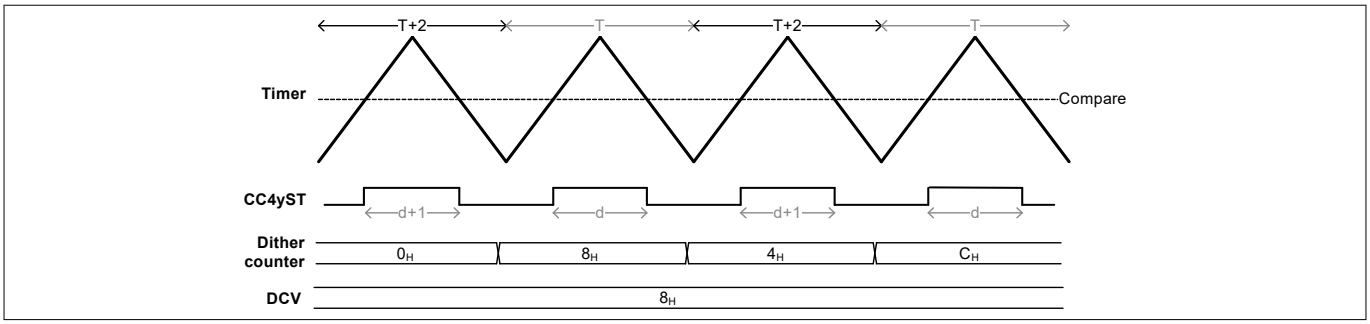


Figure 263 Dither timing diagram in center aligned - CC4yTC.DITHE = 11_B

***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.*

21.2.6.4 Capture Extended Read Back Mode

Each Timer Slice capture logic can operate in a FIFO read back mode. This mode can be enabled by setting the CC4yTC.ECM = 1_B. This Extended Read back mode allows the software to read back the capture data always from the same address (CC4yECRD0 for the structure linked with the capture trigger 0 or CC4yECRD1 for the one linked with capture trigger 1). This read back will always return the oldest captured value, enabling an easy software routine implementation for reconstructing the capture data.

This function allows the usage of a FIFO structure for each capturing trigger. This relaxes the software read back routine when multiple capture triggers are present, and the software is not fast enough to perform a read operation in each capture event.

This FIFO read back function is present for a depth-4 and depth-2 FIFO structure.

The read back data contains also a lost value bitfield, that indicates if a capture trigger was lost due to the fact that the FIFO structure was full. This bitfield is set whenever a

capture event was sensed and the FIFO was full (regardless if the continuos capture mode was enabled or not). This bitfield is cleared automatically by HW whenever the next read of the CC4yECRD0/CC4yECRD1 register occurs. This bitfield does not indicate how many capture events were lost, it just indicates that between two ECRD reads at least a capture event was lost (this can help the SW evaluate which part of the data read, can be used for calculation).

***Note**: When the ECM bitfield is set, reading the individual capture registers is still possible. Nevertheless the full flags can only be cleared by the HW when a read back is done via the CC4yECRD0/CC4yECRD1 address.*

Depth 4 Structure

The FIFO depth-4 structure is present in the hardware when the capture trigger 1 is enabled and the CC4yTC.SCE = 1_B (same capture event), Figure 264.

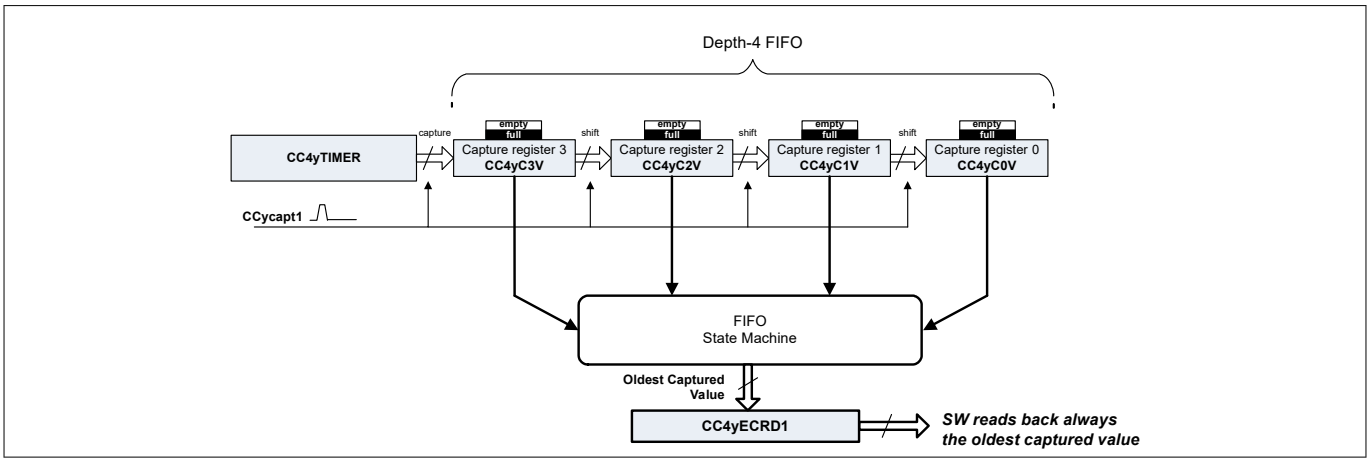


Figure 264 Capture Extended Read Back - Depth 4

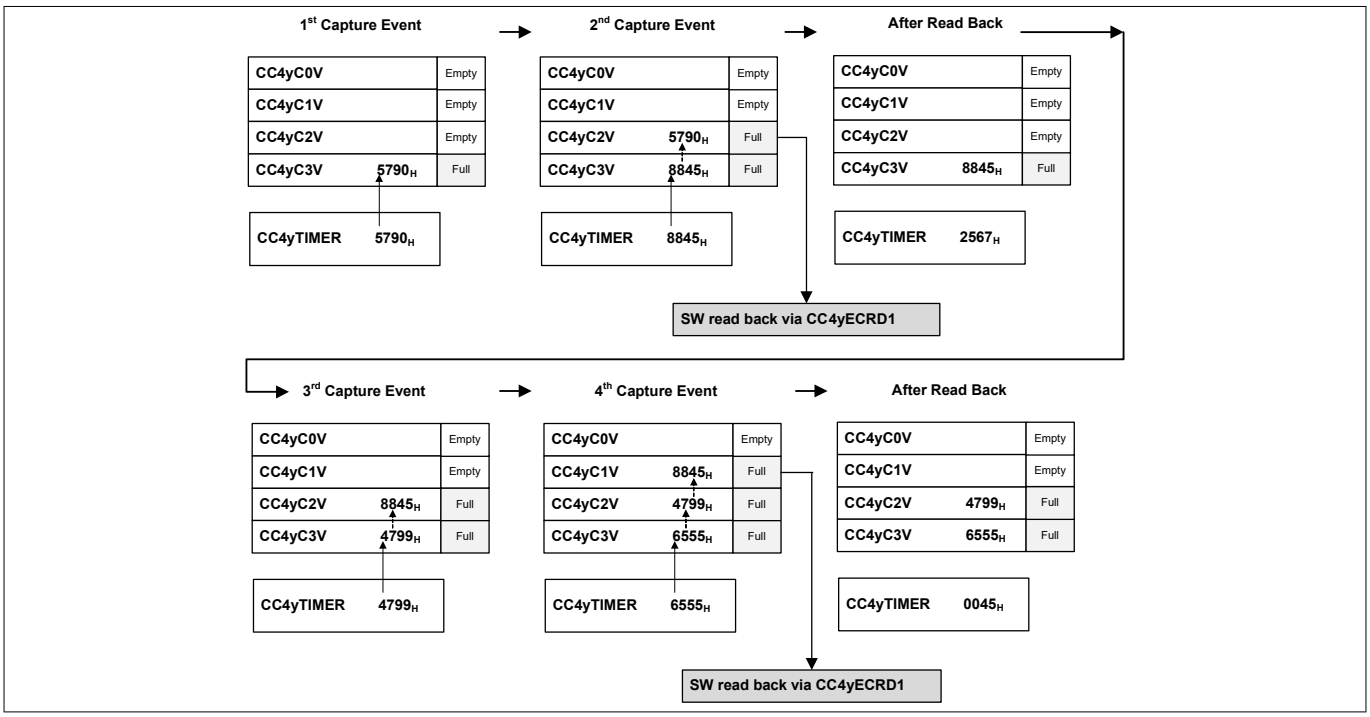


Figure 265 Depth 4 software access example

Depth 2 Structure

Each Timer Slice can have two capture structures of depth-2: one used with capture trigger 0 and another with capture trigger 1.

The one linked with capture trigger 0, is accessed via the CC4yECRD0 while the one linked with the capture trigger 1 is accessed via the CC4yECRD1, Figure 266.

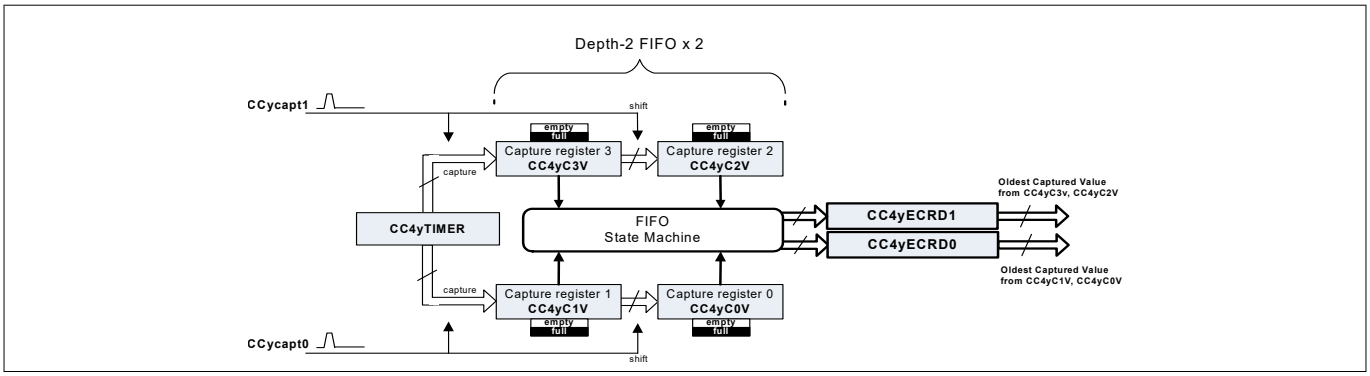


Figure 266 Capture Extended Read Back - Depth 2

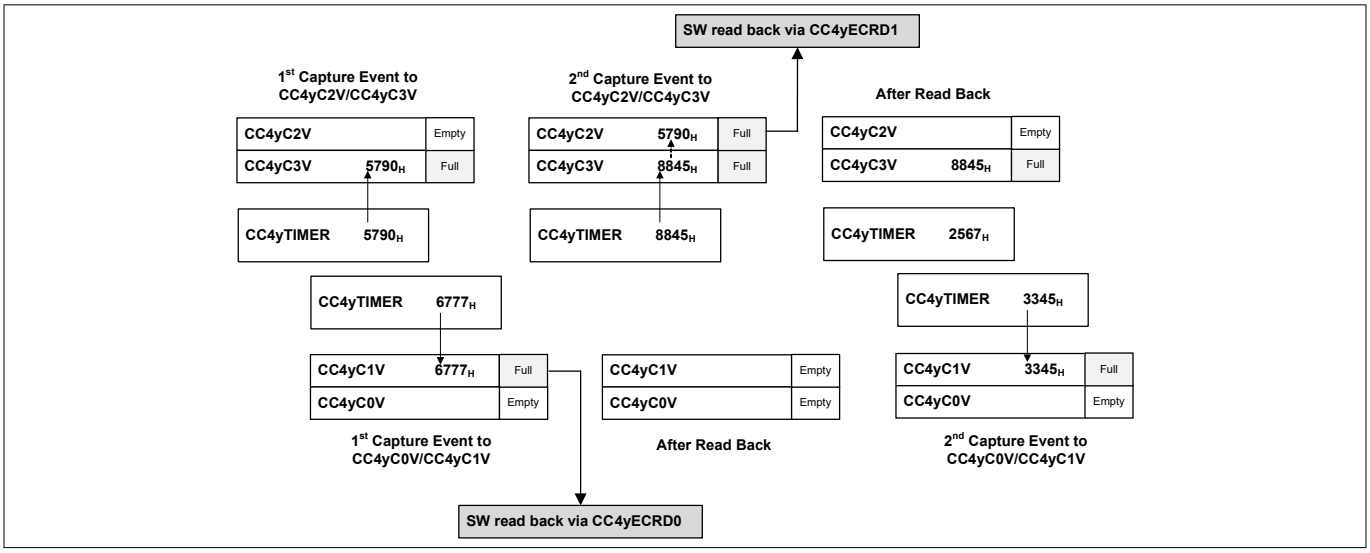


Figure 267 Depth 2 software access example

21.2.7 Clock 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.

21.2.7.1 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 161. 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 161 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	$f_{ccu4}/32768$

21.2.7.2 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 268 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 Updating the Period, Duty Cycle and other PWM conditions.

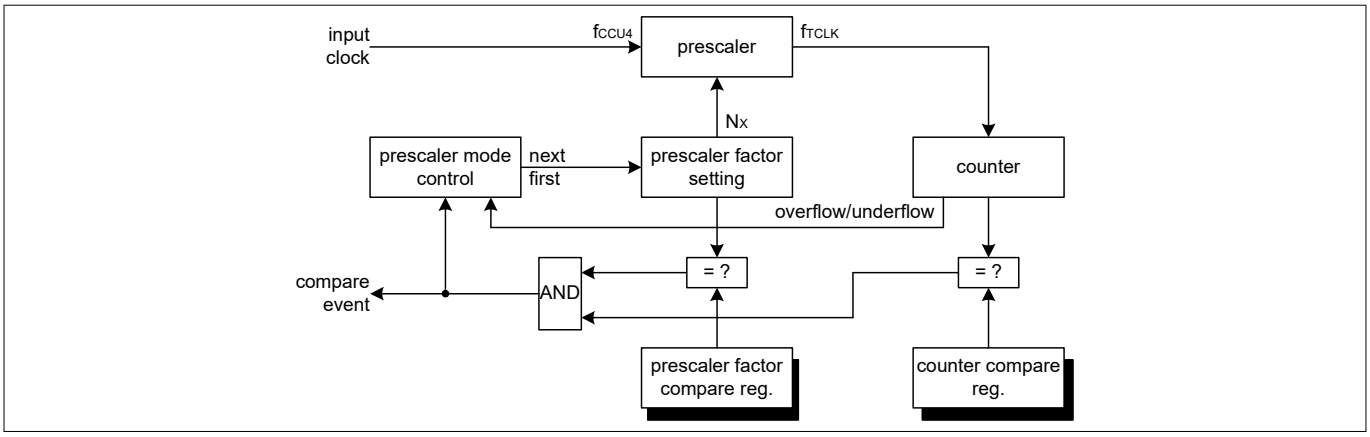


Figure 268 Floating prescaler in compare mode overview

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 Prescaler Usage for a full description of the usage of the floating prescaler mode in conjunction with compare and capture modes.

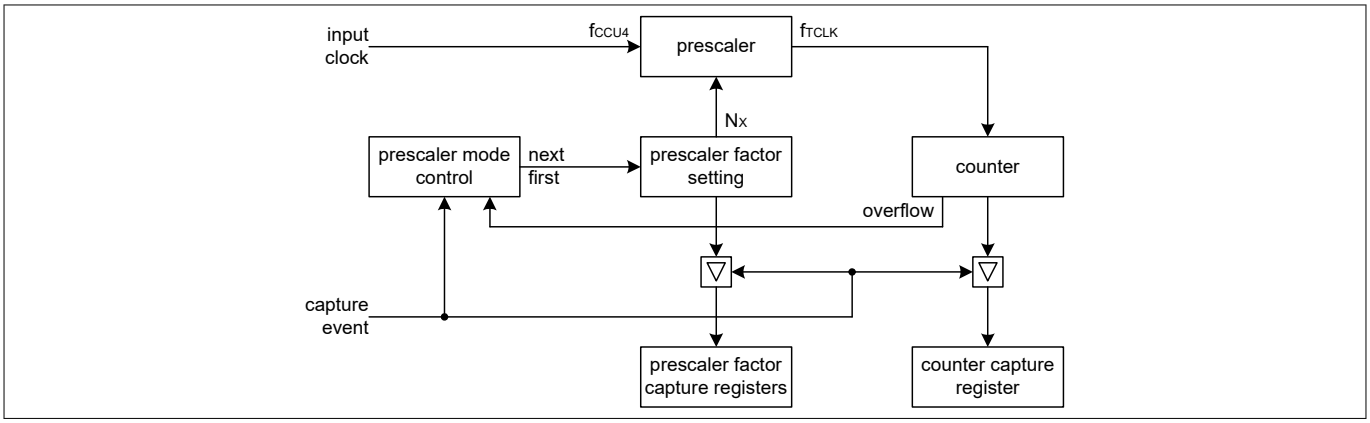


Figure 269 Floating Prescaler in capture mode overview

21.2.8 CCU4 Usage

21.2.8.1 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 270.

In the same manner a 100% duty cycle signal can be generated in Center Aligned Mode, see Figure 271.

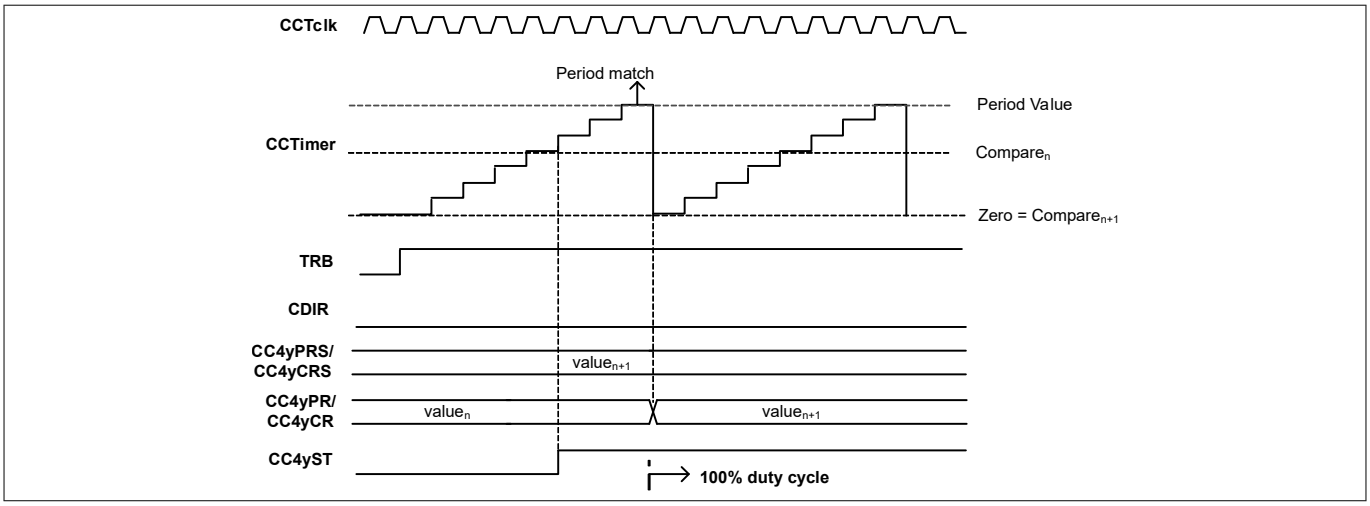


Figure 270 PWM with 100% duty cycle - Edge Aligned Mode

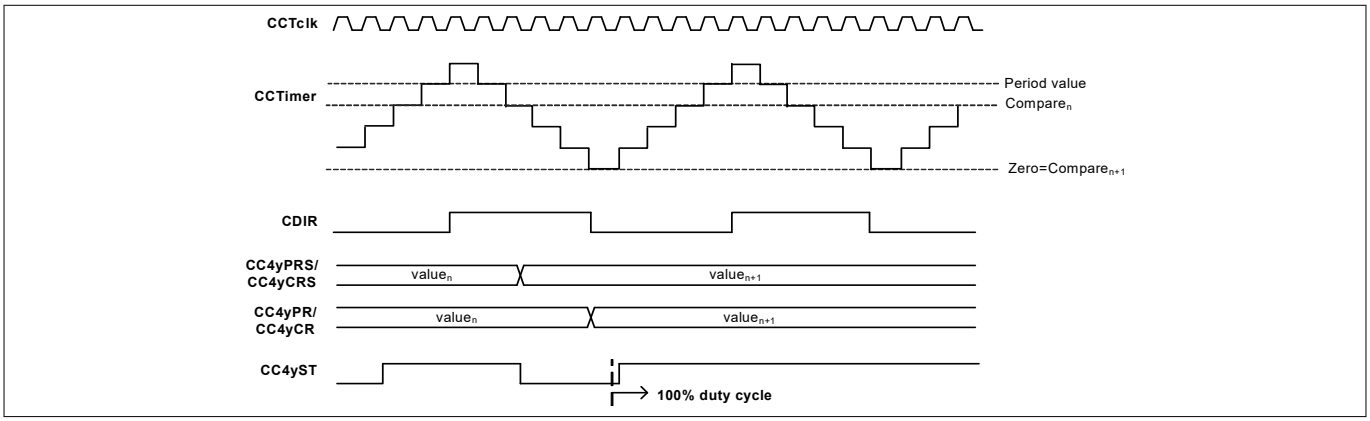


Figure 271 PWM with 100% duty cycle - Center Aligned Mode

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 272.

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 273.

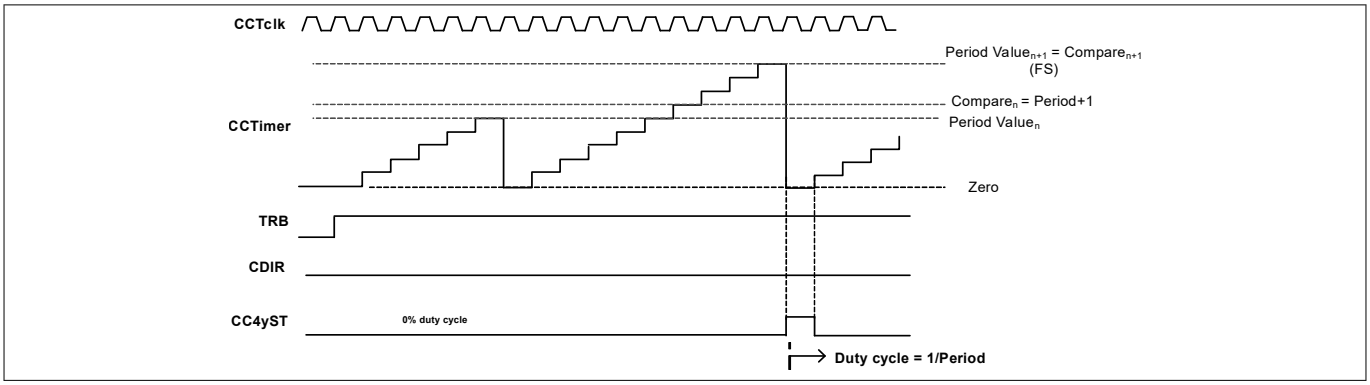


Figure 272 PWM with 0% duty cycle - Edge Aligned Mode

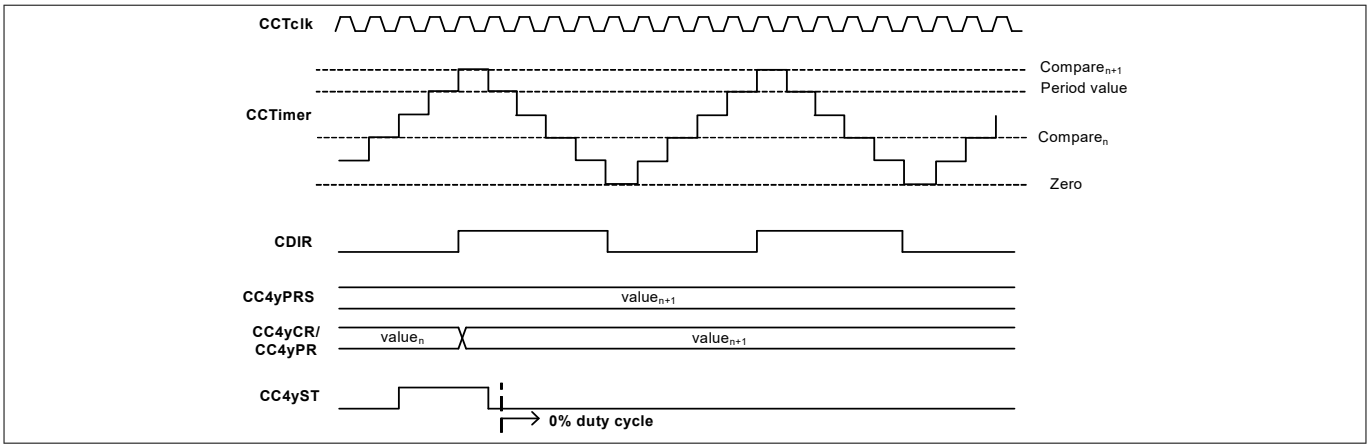


Figure 273 PWM with 0% duty cycle - Center Aligned Mode

21.2.8.2 Prescaler Usage

In Normal Prescaler Mode, the frequency of the f_{tclk} fed to the specific CC4y is chosen from the Table 161, 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 161. 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 274.

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 than 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.

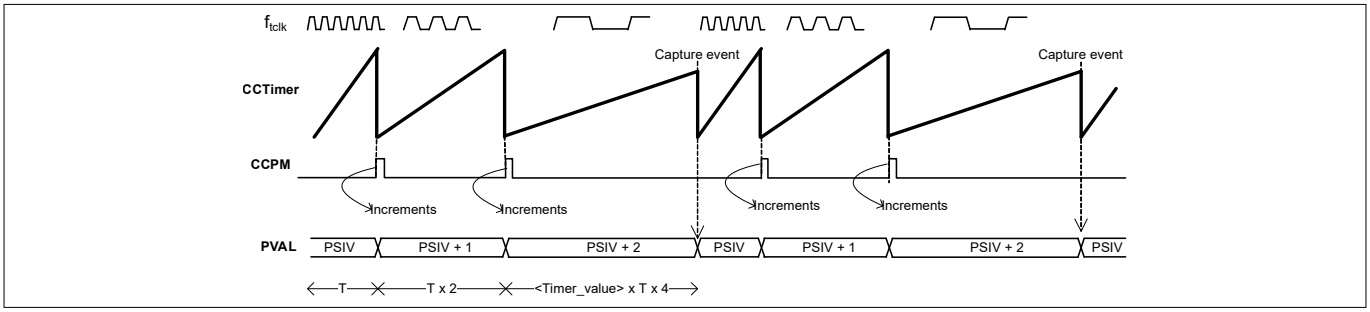


Figure 274 Floating Prescaler capture mode usage

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 275, 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 ftclk 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:

$$Period = (1/f_{tclk} + 2/f_{tclk} + 4/f_{tclk}) / 3 \quad (45)$$

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 276.

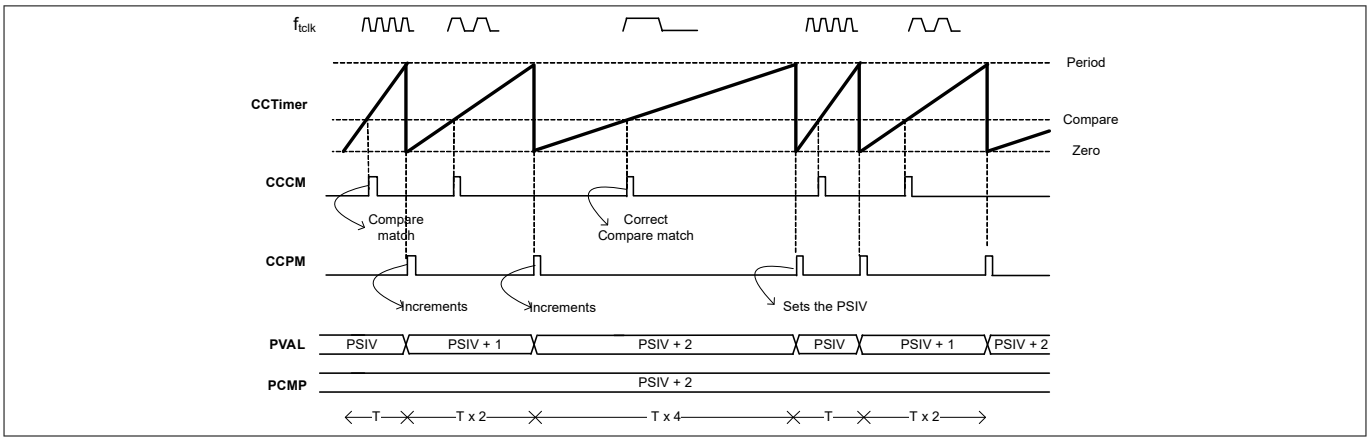


Figure 275 Floating Prescaler compare mode usage - Edge Aligned

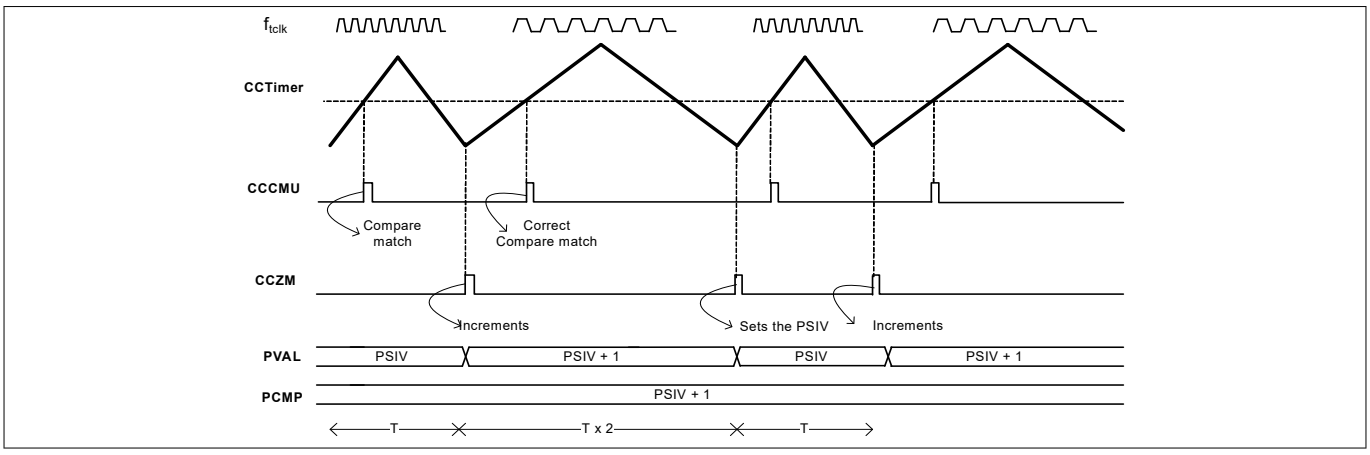


Figure 276 Floating Prescaler compare mode usage - Center Aligned

21.2.8.3 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 162 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 162 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
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

$$Period = [(16 - DCV) \; x \; T + DCV \; x \; (T + 1)]/16; \; in \; Edge \; Aligned \; Mode \quad (46)$$

$$Duty \; cycle = [(16 - DCV) \; x \; d/T + DCV \; x \; (d+1)/(T + 1)]/16; \; in \; Edge \; Aligned \; Mode \quad (47)$$

$$Period = [(16 - DCV) \; x \; T + DCV \; x \; (T + 2)]/16; \; in \; Center \; Aligned \; Mode \tag{48}$$

$$Duty \; cycle = [(16 - DCV) \; x \; d/T + DCV \; x \; (d+2)/(T + 2)]/16; \; in \; Center \; Aligned \; Mode \tag{49}$$

DITHE = 10_B

$$Period = T \; ; \; in \; Edge \; Aligned \; Mode \tag{50}$$

$$Duty \; cycle = [(16 - DCV) \; x \; d/T + DCV \; x \; (d-1)/T \;]/16; \; in \; Edge \; Aligned \; Mode \tag{51}$$

$$Period = T \; ; \; in \; Center \; Aligned \; Mode \tag{52}$$

$$Duty \; cycle = [(16 - DCV) \; x \; d/T + DCV \; x \; (d-1)/T]/16; \; in \; Center \; Aligned \; Mode \tag{53}$$

DITHE = 11_B

$$Period = [(16 - DCV) \; x \; T + DCV \; x \; (T + 1)]/16; \; in \; Edge \; Aligned \; Mode \tag{54}$$

$$Duty \; cycle = [(16 - DCV) \; x \; d/T + DCV \; x \; d/(T + 1)]/16; \; in \; Edge \; Aligned \; Mode \tag{55}$$

$$Period = [(16 - DCV) \; x \; T + DCV \; x \; (T + 2)]/16; \; in \; Center \; Aligned \; Mode \tag{56}$$

$$Duty \; cycle = [(16 - DCV) \; x \; d/T + DCV \; x \; (d+1)/(T + 2)]/16; \; in \; Center \; Aligned \; Mode \tag{57}$$

where:

T - Original period of the signal, see Calculating the PWM Period and Duty Cycle

d - Original duty cycle of the signal, see Calculating the PWM Period and Duty Cycle

21.2.8.4 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

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 277 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).

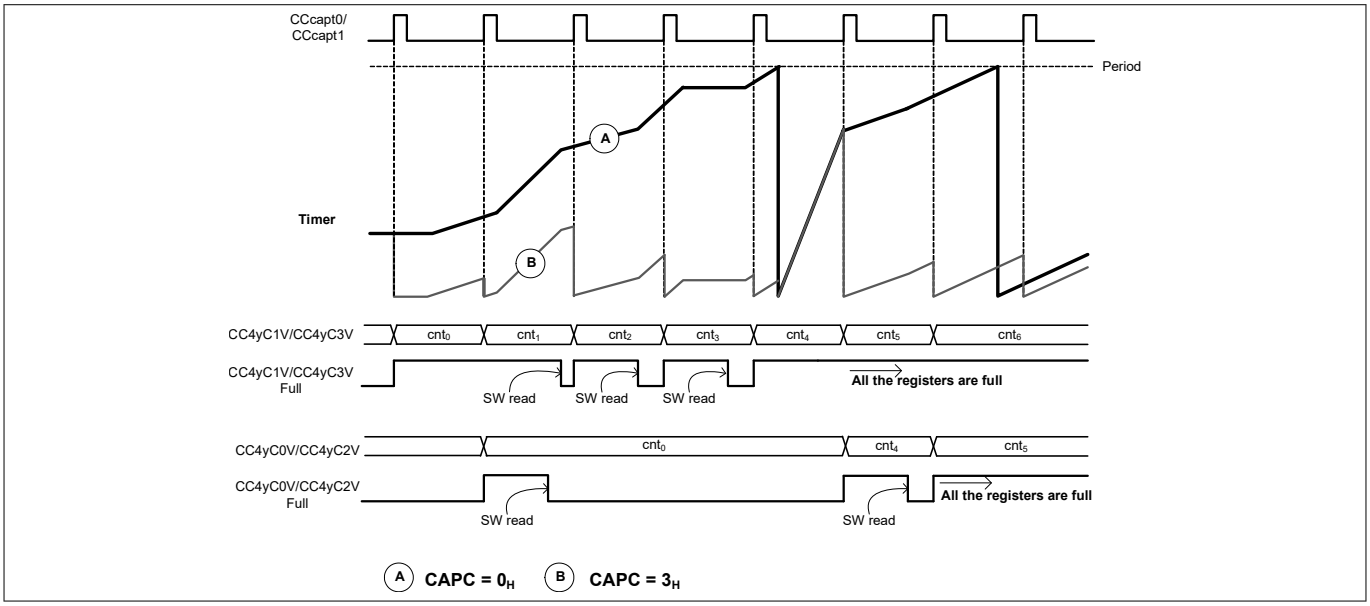


Figure 277 Capture mode usage - single channel

Same Capture Event - SCE = 1

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 278. 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, for example, 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.

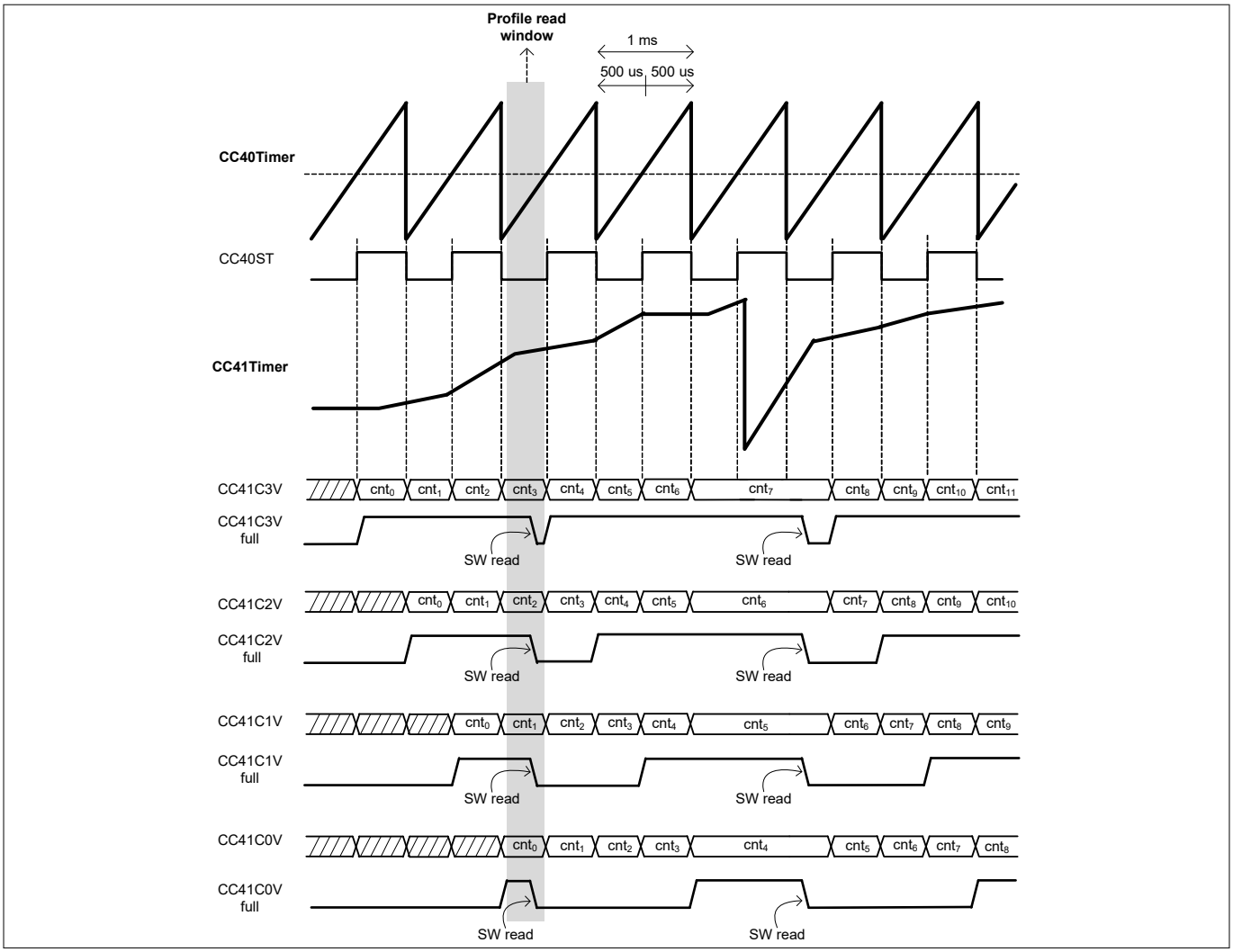


Figure 278 Three Capture profiles - CC4yTC.SCE = 1_B

To calculate the three different profiles in Figure 278, 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.*

High Dynamics Capturing

In some cases the dynamics of the capture trigger(s) may vary greatly over time. This will impose that the software needs to be prepared for the worst case scenario, where the frequency of the capture triggers may be very high. In applications where cycle-by-cycle calculation is needed (calculation in each capture trigger), then this constraints needs to be met by the software. Nevertheless for applications where a cycle-by-cycle calculation is not needed, the software can read back the FIFO data register in a periodic base and fetch all the data that has been captured so far, Figure 279.

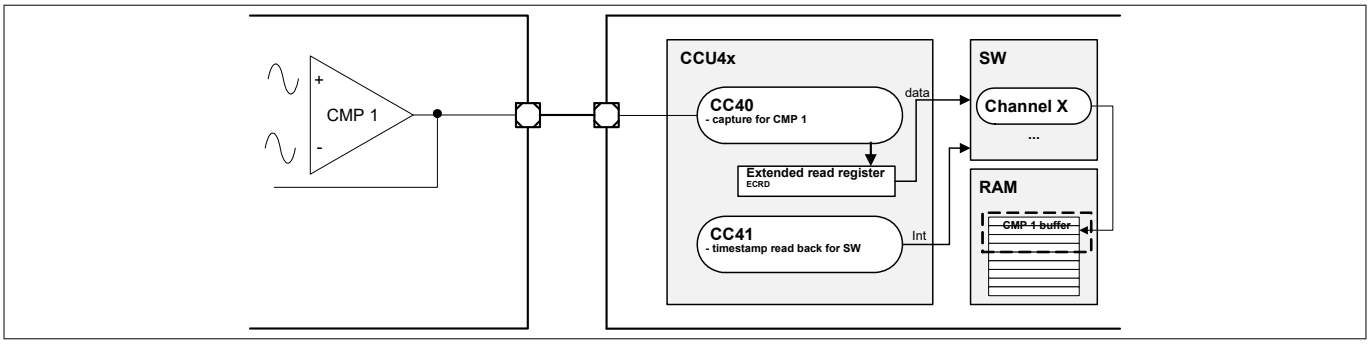


Figure 279 High dynamics capturing with software controlled timestamp

In this scenario, the software/CPU will read back the complete set of capture registers (2 or 4 depending on the chosen configuration), every time that an interrupt is triggered from the timestamp timer (the periodicity of this timer can also be adjusted on-the-fly).

Due to the fact that every capture register offers a full flag status bit, the software/CPU can always read back the complete set of registers. At the time of the data processing, this full flag is then checked, indicating if this value needs to be processed or not.

This FIFO read back functionality can also be used for applications that impose a heavy load on the system, which may not guarantee fixed access times to read back the captured data.

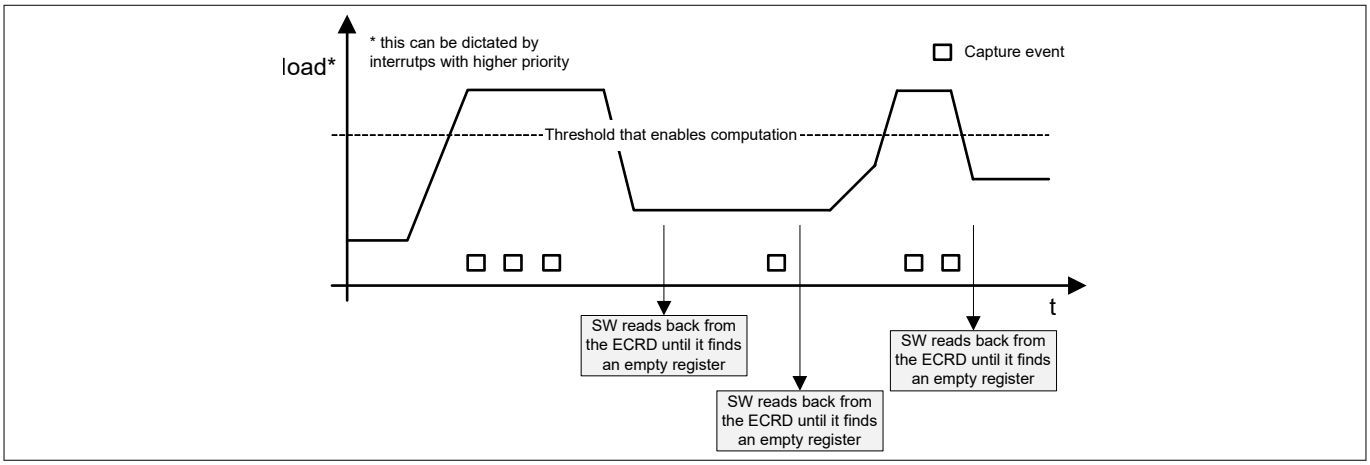


Figure 280 Extended read back during high load

Capture Grouping

In applications where multiple capture Timers are needed and the priority of the capture routines, does not imply that a cycle-by-cycle calculation needs to be done for every event, it may be suitable to group all the timers in the same CCU4x unit, Figure 281.

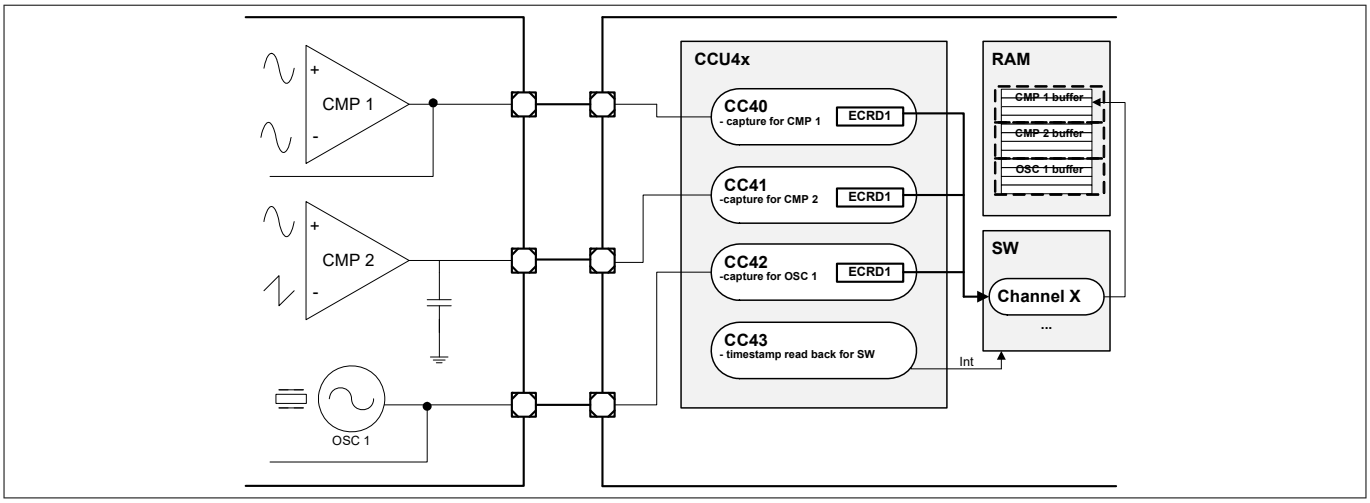


Figure 281 Capture grouping with extended read back

By setting the ECM bitfield for the TImer Slices used for capturing, the extended read back mode enables the reading back of data always in the proper capture order (from oldest to newest data). A timestamp timer is then used to trigger the Software/CPU to read back all the capture data present in the Timer Slices.

Every time that the interrupt is sensed, the Software/CPU (in this example) reads back the complete set of capture registers (via the ECRD address) for all Timer Slices. Due to the fact that each data read has a full flag indicator, the Software/CPU can read back the complete set of capture registers from all timers. This allows a fixed memory allocation that is as big as the number of captured registers, Figure 282 (in this example 4 capture registers for each Timer Slice are being used).

The additional lost value bitfield (LCV) on the header of each ECRD data, will indicate if any capture trigger was lost between read operations (this can happen if the capture triggers are faster than the routine that is reading back the values).

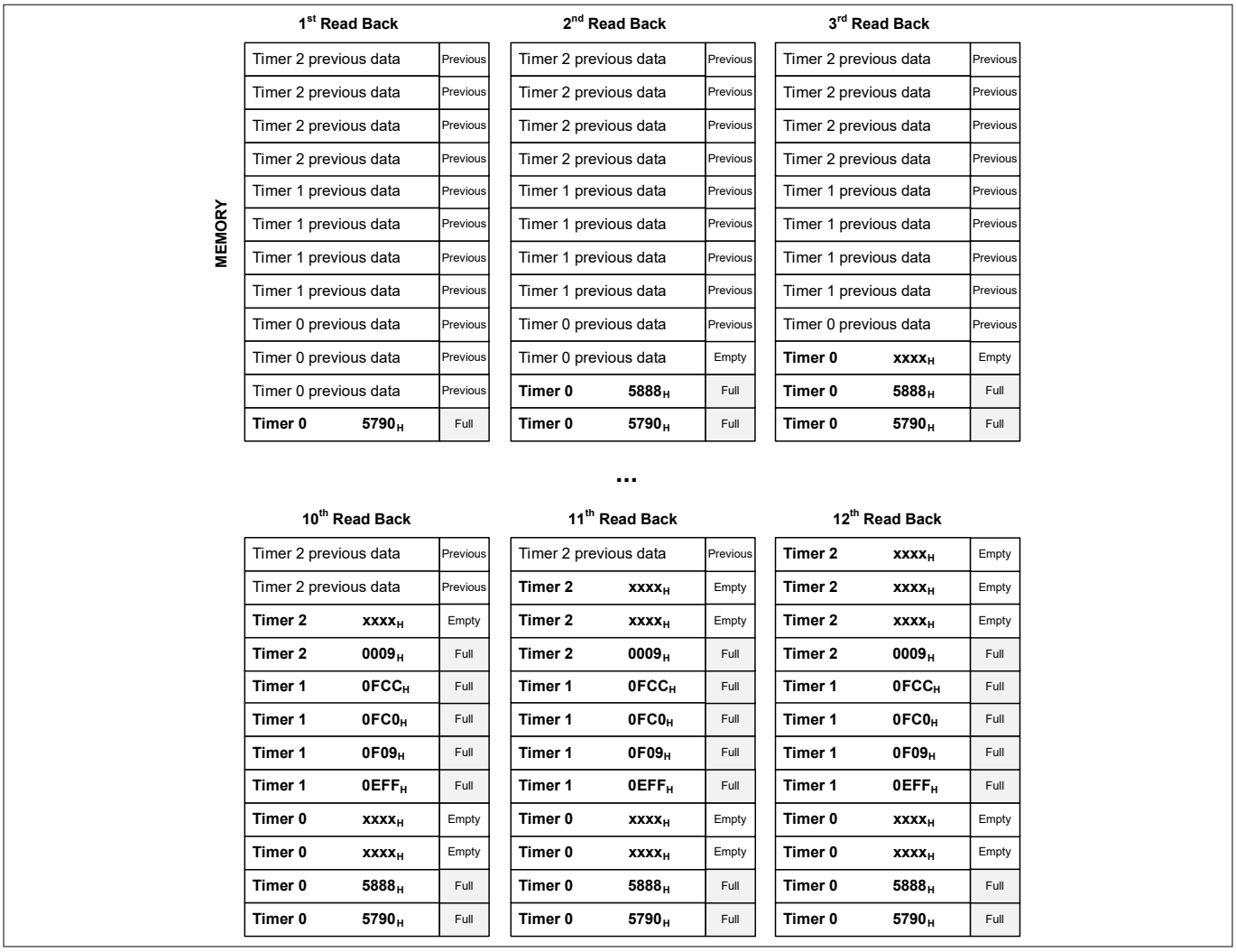


Figure 282 Memory structure for extended read back

21.3 Service Request Generation

Each CCU4 slice has an interrupt structure as the one in Figure 283. 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 163 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 CC4yINS2.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 CC4yINS2.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 163 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

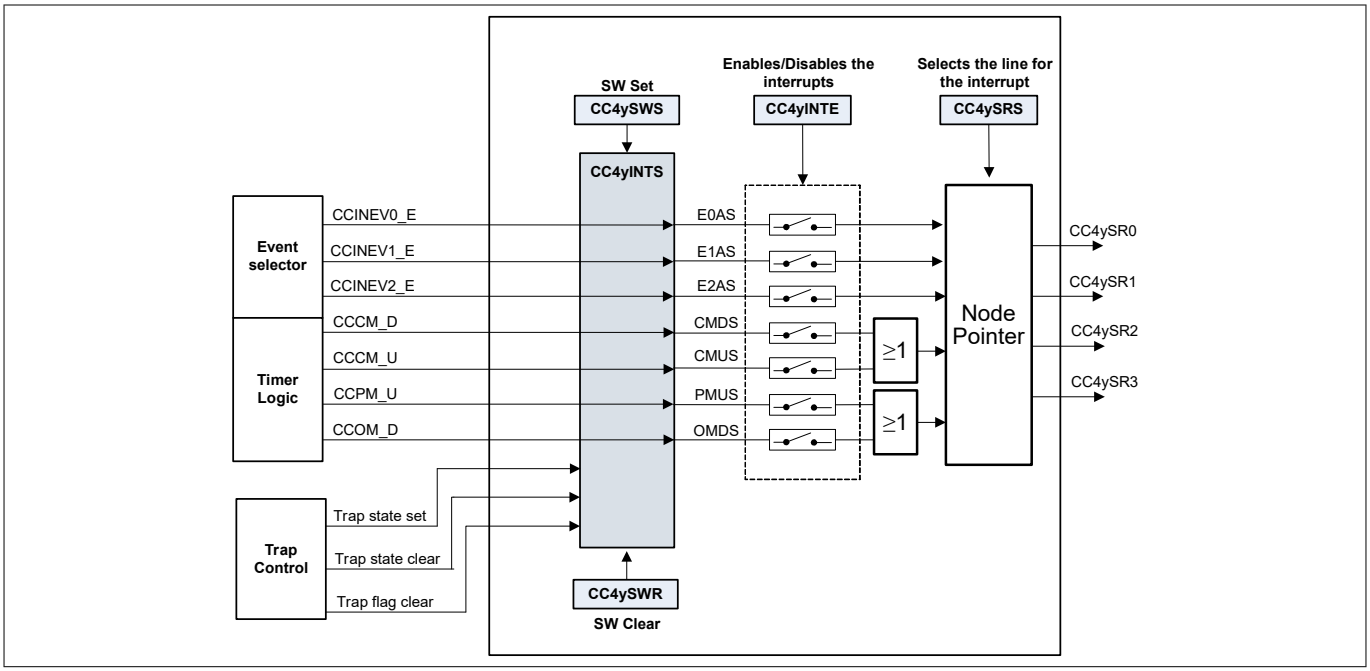


Figure 283 Slice interrupt structure overview

Each of the interrupt events can then be forwarded to one of the slice's four service request lines, Figure 284. The value set on the CC4ySRS controls which interrupt event is mapped into which service request line.

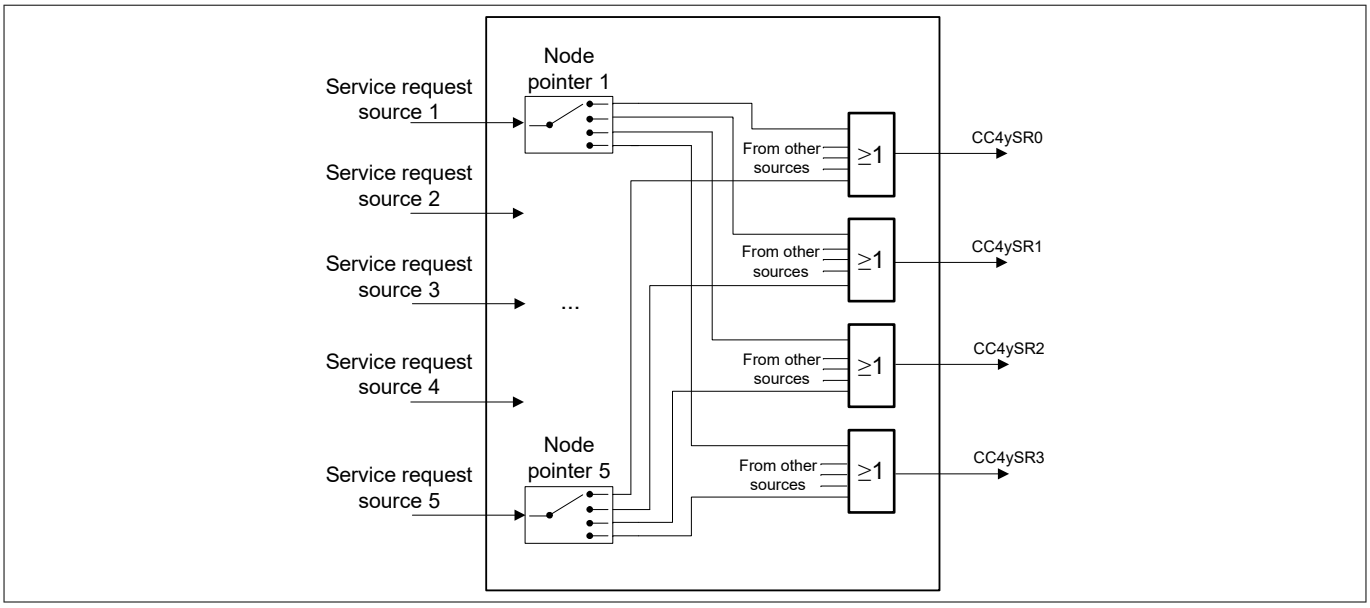


Figure 284 Slice Interrupt Node Pointer overview

The four service request lines of each slice are OR together inside the kernel of the CCU4, see Figure 285. This means that there are only four service request lines per CCU4, that can have in each line interrupt requests coming from different slices.

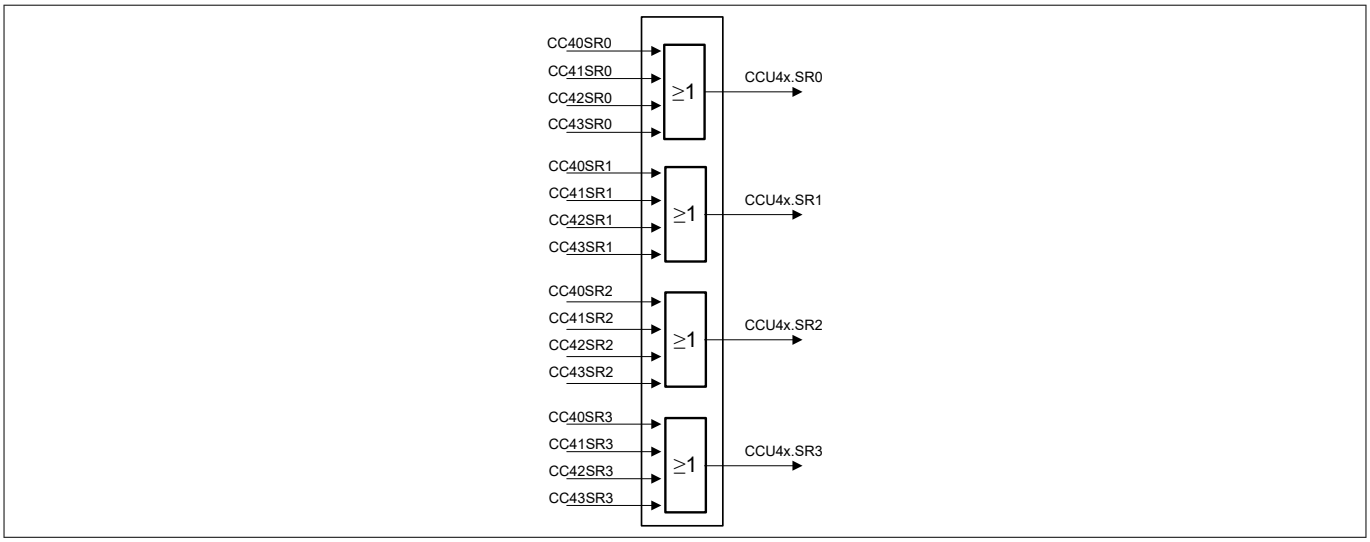


Figure 285 CCU4 service request overview

21.4 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, for example, 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.

21.5 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}.

21.5.1 Clocks

Module Clock

The module clock of the CCU4 module is described in the SCU chapter as f_{PCLK}.

The bus interface clock of the CCU4 module is described in the SCU chapter as f_{MCLK}.

It is possible to disable the module clock for the CCU4 via the GSTAT register, nevertheless, there may be a dependency of the fccu4 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 164 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

21.5.2 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).

21.6 Initialization and System Dependencies

21.6.1 Initialization Sequence

The initialization sequence for an application that is using the CCU4, should be the

following:

1st Step: Enable the CCU4 clock via the specific SCU register, CGATCLR0.

2nd Step: Enable the prescaler block, by writing 1_B to the GIDLC.SPRB field.

3rd Step: Configure the global CCU4 register GCTRL

4th Step: Configure all the registers related to the required Timer Slice(s) functions, including the interrupt/ service request configuration.

5th 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.

6th Step: Enable the specific timer slice(s), CC4y, by writing 1_B to the specific GIDLC.CSyI.

7th 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.GLCSTxx input signal needs to be configured previously as a start function, see External Start/Stop.

21.6.2 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 Clocks
- 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.

21.7 Registers

Registers Overview

The absolute register address is calculated by adding:
Module Base Address + Offset Address

Table 165 Registers Address Space

Module	Base Address	End Address	Note
CCU40	48040000_H	$48043FFF_H$	
CCU41	48044000_H	$48047FFF_H$	

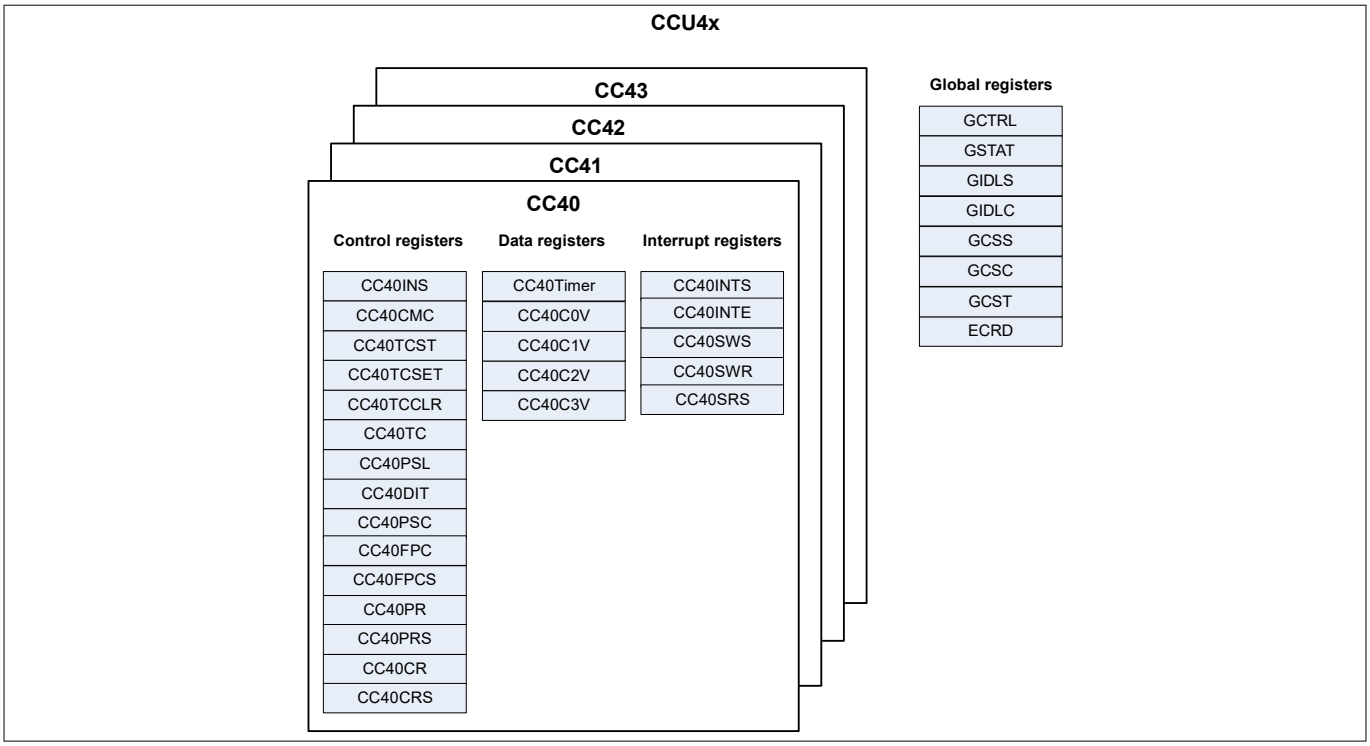


Figure 286 CCU4 registers overview

Table 166 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	
GSTAT	General Slice Status Register	0004_H	U, PV	BE	
GIDLS	General Idle Enable Register	0008_H	U, PV	U, PV	
GIDLC	General Idle Disable Register	$000C_H$	U, PV	U, PV	
GCSS	General Channel Set Register	0010_H	U, PV	U, PV	
GCSC	General Channel Clear Register	0014_H	U, PV	U, PV	
GCST	General Channel Status Register	0018_H	U, PV	BE	
MIDR	Module Identification Register	0080_H	U, PV	BE	
CC4y Registers, y = 0...3, n = y + 1					
CC4yINS1	Input Selector Unit Configuration Register 1	$0ND8_H$	U, PV	U, PV	
CC4yINS2	Input Selector Unit Configuration Register 2	$0N00_H$			
CC4yCMC	Connection Matrix Configuration	$0N04_H$	U, PV	U, PV	
CC4yTST	Timer Run Status	$0N08_H$	U, PV	BE	
CC4yTCSET	Timer Run Set	$0N0C_H$	U, PV	U,PV	
CC4yTCCLR	Timer Run Clear	$0N10_H$	U, PV	U, PV	

(table continues...)

Table 166 (continued) Register Overview of CCU4

Short Name	Description	Offset Addr.[1]	Access Mode		Description See
			Read	Write	
CC4yTC	General Timer Configuration	$0N14_H$	U, PV	U, PV	
CC4yPSL	Output Passive Level Configuration	$0N18_H$	U, PV	U, PV	
CC4yDIT	Dither Configuration	$0N1C_H$	U, PV	BE	
CC4yDITS	Dither Shadow Register	$0N20_H$	U, PV	U, PV	
CC4yPSC	Prescaler Configuration	$0N24_H$	U, PV	U, PV	
CC4yFPC	Prescaler Compare Value	$0N28_H$	U, PV	U, PV	
CC4yFPCS	Prescaler Shadow Compare Value	$0N2C_H$	U, PV	U, PV	
CC4yPR	Timer Period Value	$0N30_H$	U, PV	BE	
CC4yPRS	Timer Period Shadow Value	$0N34_H$	U, PV	U, PV	
CC4yCR	Timer Compare Value	$0N38_H$	U, PV	BE	
CC4yCRS	Timer Compare Shadow Value	$0N3C_H$	U, PV	U, PV	
CC4yTIMER	Timer Current Value	$0N70_H$	U, PV	U, PV	
CC4yC0V	Capture Register 0 Value	$0N74_H$	U, PV	BE	
CC4yC1V	Capture Register 1Value	$0N78_H$	U, PV	BE	
CC4yC2V	Capture Register 2 Value	$0N7C_H$	U, PV	BE	
CC4yC3V	Capture Register 3 Value	$0N80_H$	U, PV	BE	
CC4yINTS	Interrupt Status	$0NA0_H$	U, PV	BE	
CC4yINTE	Interrupt Enable	$0NA4_H$	U, PV	U, PV	
CC4ySRS	Interrupt Configuration	$0NA8_H$	U, PV	U, PV	
CC4ySWS	Interrupt Status Set	$0NAC_H$	U, PV	U, PV	
CC4ySWR	Interrupt Status Clear	$0NB0_H$	U, PV	U, PV	
CC4ySTC	Shadow Transfer Control	$0NB4_H$	U, PV	U, PV	
CC4yECRD0	Extended Read Back 0	$0NB8_H$	U, PV	BE	
CC4yECRD1	Extended Read Back 1	$0NBC_H$	U, PV	BE	

1) The absolute register address is calculated as follows:
Module Base Address + Offset Address (shown in this column)

21.8 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.

21.8.1 CCU40 Pins

Table 167 CCU40 Pin Connections

Global Input/Output	I/O	Connected To	Description
CCU40.MCLK	I	PCLK	
CCU40.CLKA	I	ERU1.IOUT0	
CCU40.CLKB	I	ERU0.IOUT0	
CCU40.CLKC	I	ERU0.IOUT1	
CCU40.MCSS	I	POSIF0.OUT6	
CCU40.SR0	O	NVIC; CCU80.IGBTD; POSIF0.MSETA; ERU0.OGU01; CCU40.IN0BB;	
CCU40.SR1	O	NVIC; ERU0.OGU11; CCU40.IN2AT; CCU40.IN3AT; CCU40.IN1BB;	
CCU40.SR2	O	NVIC; VADC0.BGREQTRA; VADC0.G0REQTRA; VADC0.G1REQTRA; CCU80.IN0AK; CCU80.IN1AK; ERU0.OGU21; CCU40.IN0AT; CCU40.IN1AT; CCU40.IN2BB;	
CCU40.SR3	O	NVIC; VADC0.BGREQTRB; VADC0.G0REQTRB; VADC0.G1REQTRB; CCU80.IGBTB; CCU80.IN2AK; CCU80.IN3AK; ERU0.OGU31; CCU40.IN3BB;	

Table 168 CCU40 - CC40 Pin Connections

Global Input/ Output	I/O	Connected To	Description
CCU40.IN0AA	I	P0.12	General purpose function

(table continues...)

Table 168 (continued) CCU40 - CC40 Pin Connections

Global Input/ Output	I/O	Connected To	Description
CCU40.IN0AB	I	P0.6	General purpose function
CCU40.IN0AC	I	P0.0	General purpose function
CCU40.IN0AD	I	ERU0.PDOUT1	General purpose function
CCU40.IN0AE	I	POSIF0.OUT0	General purpose function
CCU40.IN0AF	I	POSIF0.OUT1	General purpose function
CCU40.IN0AG	I	POSIF0.OUT3	General purpose function
CCU40.IN0AH	I	CCU80.ST3	General purpose function
CCU40.IN0AI	I	SCU.GSC40	General purpose function
CCU40.IN0AJ	I	ERU0.PDOUT0	General purpose function
CCU40.IN0AK	I	ERU0.IOUT0	General purpose function
CCU40.IN0AL	I	USIC0_CH0.DX2INS	General purpose function
CCU40.IN0AM	I	CCU40.GP10	General purpose function
CCU40.IN0AN	I	CCU40.ST1	General purpose function
CCU40.IN0AO	I	CCU40.ST2	General purpose function
CCU40.IN0AP	I	CCU40.ST3	General purpose function
CCU40.IN0AQ	I	BCCU0.OUT0	General purpose function
CCU40.IN0AR	I	ACMP1.OUT	General purpose function
CCU40.IN0AS	I	ACMP0.OUT	General purpose function
CCU40.IN0AT	I	CCU40.SR2	General purpose function
CCU40.IN0AU	I	CCU40.ST0	General purpose function
CCU40.IN0AV	I	P4.8	General purpose function
CCU40.IN0AW	I	ERU1.PDOUT0	General purpose function
CCU40.IN0AX	I	ERU1.IOUT0	General purpose function
CCU40.IN0AY	I	ERU1.PDOUT1	General purpose function
CCU40.IN0AZ	I	BCCU0.OUT6	General purpose function
Reserved	Reserved	Reserved	Reserved
CCU40.IN0BB	I	CCU40.SR0	General purpose function
CCU40.IN0BC	I	0	Reserved
CCU40.IN0BD	I	0	Reserved
CCU40.IN0BE	I	0	Reserved
CCU40.IN0BF	I	0	Reserved
CCU40.IN0BG	I	0	Reserved
CCU40.IN0BH	I	0	Reserved

(table continues...)

Table 168 (continued) CCU40 - CC40 Pin Connections

Global Input/ Output	I/O	Connected To	Description
CCU40.IN0BI	I	0	Reserved
CCU40.IN0BJ	I	0	Reserved
CCU40.IN0BK	I	0	Reserved
CCU40.IN0BL	I	0	Reserved
CCU40.IN0BM	I	0	Reserved
CCU40.IN0BN	I	0	Reserved
CCU40.IN0BO	I	0	Reserved
CCU40.IN0BP	I	0	Reserved
CCU40.IN0BQ	I	0	Reserved
CCU40.IN0BR	I	0	Reserved
CCU40.IN0BS	I	0	Reserved
CCU40.IN0BT	I	0	Reserved
CCU40.IN0BU	I	0	Reserved
CCU40.IN0BV	I	0	Reserved
CCU40.MCI0	I	BCCU0.OUT2	Multi Channel pattern input
CCU40.OUT0	O	P0.0; P0.5; P0.6; P1.0; P2.0; P4.8; P2.0.HW1 direction control;	Slice compare output
CCU40.GP00	O	CCU40.IN3AM	Selected signal for event 0
CCU40.GP01	O	not connected	Selected signal for event 1
CCU40.GP02	O	CCU80.IN0AC	Selected signal for event 2
CCU40.ST0	O	CCU40.IN0AU; CCU40.IN1AN; CCU40.IN2AN; CCU40.IN3AN; CCU41.IN0AL; CCU80.IGBTC; POSIF0.HSDA; VADC0.BGREQGTD; VADC0.G0REQGTD; VADC0.G1REQGTD; ERU1.0B0;	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)
CCU40.IN1AA	I	P0.12	General purpose function
CCU40.IN1AB	I	P0.7	General purpose function

(table continues...)

Table 168 (continued) CCU40 - CC40 Pin Connections

Global Input/ Output	I/O	Connected To	Description
CCU40.IN1AC	I	P0.1	General purpose function
CCU40.IN1AD	I	ERU0.PDOUT0	General purpose function
CCU40.IN1AE	I	POSIF0.OUT0	General purpose function
CCU40.IN1AF	I	POSIF0.OUT1	General purpose function
CCU40.IN1AG	I	POSIF0.OUT3	General purpose function
CCU40.IN1AH	I	POSIF0.OUT4	General purpose function
CCU40.IN1AI	I	SCU.GSC40	General purpose function
CCU40.IN1AJ	I	ERU0.PDOUT1	General purpose function
CCU40.IN1AK	I	ERU0.IOUT1	General purpose function
CCU40.IN1AL	I	USIC0_CH1.DX2INS	General purpose function
CCU40.IN1AM	I	CCU40.GP20	General purpose function
CCU40.IN1AN	I	CCU40.ST0	General purpose function
CCU40.IN1AO	I	CCU40.ST2	General purpose function
CCU40.IN1AP	I	CCU40.ST3	General purpose function
CCU40.IN1AQ	I	BCCU0.OUT8	
CCU40.IN1AR	I	ACMP3.OUT	
CCU40.IN1AS	I	ACMP2.OUT	
CCU40.IN1AT	I	CCU40.SR2	
CCU40.IN1AU	I	CCU40.ST1	
CCU40.IN1AV	I	P4.9	
CCU40.IN1AW	I	ERU1.PDOUT1	
CCU40.IN1AX	I	ERU1.IOUT1	
CCU40.IN1AY	I	ERU1.PDOUT0	
CCU40.IN1AZ	I	BCCU0.OUT3	
CCU40.IN1BA	I	P4.1	
CCU40.IN1BB	I	CCU40.SR1	
CCU40.IN1BC	I	0	Reserved
CCU40.IN1BD	I	0	Reserved
CCU40.IN1BE	I	0	Reserved
CCU40.IN1BF	I	0	Reserved
CCU40.IN1BG	I	0	Reserved
CCU40.IN1BH	I	0	Reserved
CCU40.IN1BI	I	0	Reserved

(table continues...)

Table 168 (continued) CCU40 - CC40 Pin Connections

Global Input/ Output	I/O	Connected To	Description
CCU40.IN1BJ	I	0	Reserved
CCU40.IN1BK	I	0	Reserved
CCU40.IN1BL	I	0	Reserved
CCU40.IN1BM	I	0	Reserved
CCU40.IN1BN	I	0	Reserved
CCU40.IN1BO	I	0	Reserved
CCU40.IN1BP	I	0	Reserved
CCU40.IN1BQ	I	0	Reserved
CCU40.IN1BR	I	0	Reserved
CCU40.IN1BS	I	0	Reserved
CCU40.IN1BT	I	0	Reserved
CCU40.IN1BU	I	0	Reserved
CCU40.IN1BV	I	0	Reserved
CCU40.MCI1	I	BCCU0.OUT3	Multi Channel pattern input
CCU40.OUT1	O	P0.1; P0.4; P0.7; P1.1; P2.1; P4.1; P4.9; P2.1.HW1 direction control;	Slice compare output
CCU40.GP10	O	CCU40.IN0AM	Selected signal for event 0
CCU40.GP11	O	not connected	Selected signal for event 1
CCU40.GP12	O	CCU80.IN1AC	Selected signal for event 2
CCU40.ST1	O	CCU40.IN0AN; CCU40.IN1AU; CCU40.IN2AO; CCU40.IN3AO; CCU41.IN1AL; POSIF0.MSETB; VADC0.BGREQGTC; VADC0.G0REQGTC; VADC0.G1REQGTC; ERU1.1B0;	Slice status bit
CCU40.PS1	O	POSIF0.MSYNCC	Multi channel pattern sync trigger: PM when counting UP (edge aligned) or OM when counting DOWN (center aligned)

Table 169 CCU40 - CC42 Pin Connections

Global Input/Output	I/O	Connected To	Description
CCU40.IN2AA	I	P0.12	General purpose function

(table continues...)

Table 169 (continued) CCU40 - CC42 Pin Connections

Global Input/Output	I/O	Connected To	Description
CCU40.IN2AB	I	P0.8	General purpose function
CCU40.IN2AC	I	P0.2	General purpose function
CCU40.IN2AD	I	ERU0.PDOUT3	General purpose function
CCU40.IN2AE	I	POSIF0.OUT1	General purpose function
CCU40.IN2AF	I	POSIF0.OUT2	General purpose function
CCU40.IN2AG	I	POSIF0.OUT3	General purpose function
CCU40.IN2AH	I	POSIF0.OUT4	General purpose function
CCU40.IN2AI	I	SCU.GSC40	General purpose function
CCU40.IN2AJ	I	ERU0.PDOUT2	General purpose function
CCU40.IN2AK	I	ERU0.IOUT2	General purpose function
CCU40.IN2AL	I	reserved	reserved
CCU40.IN2AM	I	CCU40.GP30	General purpose function
CCU40.IN2AN	I	CCU40.ST0	General purpose function
CCU40.IN2AO	I	CCU40.ST1	General purpose function
CCU40.IN2AP	I	CCU40.ST3	General purpose function
CCU40.IN2AQ	I	BCCU0.OUT4	General purpose function
CCU40.IN2AR	I	ACMP2.OUT	General purpose function
CCU40.IN2AS	I	ACMP1.OUT	General purpose function
CCU40.IN2AT	I	CCU40.SR1	General purpose function
CCU40.IN2AU	I	CCU40.ST2	General purpose function
Reserved	Reserved	Reserved	Reserved
CCU40.IN2AW	I	ERU1.PDOUT2	General purpose function
CCU40.IN2AX	I	ERU1.IOUT2	General purpose function
CCU40.IN2AY	I	ERU1.PDOUT3	General purpose function
CCU40.IN2AZ	I	BCCU0.OUT7	General purpose function
CCU40.IN2BA	I	P4.2	General purpose function
CCU40.IN2BB	I	CCU40.SR2	General purpose function
CCU40.IN2BC	I	POSIF0.OUT0	General purpose function
CCU40.IN2BD	I	0	Reserved
CCU40.IN2BE	I	0	Reserved
CCU40.IN2BF	I	0	Reserved
CCU40.IN2BG	I	0	Reserved
CCU40.IN2BH	I	0	Reserved

(table continues...)

Table 169 (continued) CCU40 - CC42 Pin Connections

Global Input/Output	I/O	Connected To	Description
CCU40.IN2BI	I	0	Reserved
CCU40.IN2BJ	I	0	Reserved
CCU40.IN2BK	I	0	Reserved
CCU40.IN2BL	I	0	Reserved
CCU40.IN2BM	I	0	Reserved
CCU40.IN2BN	I	0	Reserved
CCU40.IN2BO	I	0	Reserved
CCU40.IN2BP	I	0	Reserved
CCU40.IN2BQ	I	0	Reserved
CCU40.IN2BR	I	0	Reserved
CCU40.IN2BS	I	0	Reserved
CCU40.IN2BT	I	0	Reserved
CCU40.IN2BU	I	0	Reserved
CCU40.IN2BV	I	0	Reserved
CCU40.MCI2	I	BCCU0.OUT4	Multi Channel pattern input
CCU40.OUT2	O	P0.2; P0.8; P1.2; P2.10; P4.2; P2.8.HW1 pull control; P2.9.HW1 pull control; P2.10.HW1 direction control;	Slice compare output
CCU40.GP20	O	CCU40.IN1AM	Selected signal for event 0
CCU40.GP21	O	not connected	Selected signal for event 1
CCU40.GP22	O	CCU80.IN2AC	Selected signal for event 2

(table continues...)

Table 169 (continued) CCU40 - CC42 Pin Connections

Global Input/Output	I/O	Connected To	Description
CCU40.ST2	O	CCU40.IN0AO; CCU40.IN1AO; CCU40.IN2AU; CCU40.IN3AP; CCU41.IN2AL; VADC0.BGREQGTB; VADC0.G0REQGTB; VADC0.G1REQGTB; POSIF0.MSETC; ERU1.2B0;	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 170 CCU40 - CC43 Pin Connections

Global Input/Output	I/O	Connected To	Description
CCU40.IN3AA	I	P0.12	General purpose function
CCU40.IN3AB	I	P0.9	General purpose function
CCU40.IN3AC	I	P0.3	General purpose function
CCU40.IN3AD	I	ERU0.PDOUT2	General purpose function
CCU40.IN3AE	I	POSIF0.OUT3	General purpose function
CCU40.IN3AF	I	POSIF0.OUT5	General purpose function
CCU40.IN3AG	I	VADC0.G0ARBCNT	General purpose function
CCU40.IN3AH	I	CCU80.IGBTO	General purpose function
CCU40.IN3AI	I	SCU.GSC40	General purpose function
CCU40.IN3AJ	I	ERU0.PDOUT3	General purpose function
CCU40.IN3AK	I	ERU0.IOUT3	General purpose function
CCU40.IN3AL	I	reserved	reserved
CCU40.IN3AM	I	CCU40.GP00	General purpose function
CCU40.IN3AN	I	CCU40.ST0	General purpose function
CCU40.IN3AO	I	CCU40.ST1	General purpose function
CCU40.IN3AP	I	CCU40.ST2	General purpose function
CCU40.IN3AQ	I	BCCU0.OUT5	General purpose function
CCU40.IN3AR	I	ACMP0.OUT	General purpose function
CCU40.IN3AS	I	ACMP3.OUT	General purpose function
CCU40.IN3AT	I	CCU40.SR1	General purpose function
CCU40.IN3AU	I	CCU40.ST3	General purpose function

(table continues...)

Table 170 (continued) CCU40 - CC43 Pin Connections

Global Input/Output	I/O	Connected To	Description
Reserved	Reserved	Reserved	Reserved
CCU40.IN3AW	I	ERU1.PDOUT3	General purpose function
CCU40.IN3AX	I	ERU1.IOUT3	General purpose function
CCU40.IN3AY	I	ERU1.PDOUT2	General purpose function
CCU40.IN3AZ	I	BCCU0.OUT1	General purpose function
CCU40.IN3BA	I	P4.3	General purpose function
CCU40.IN3BB	I	CCU40.SR3	General purpose function
CCU40.IN3BC	I	POSIF0.OUT0	General purpose function
CCU40.IN3BD	I	POSIF0.OUT1	General purpose function
CCU40.IN3BE	I	0	Reserved
CCU40.IN3BF	I	0	Reserved
CCU40.IN3BG	I	0	Reserved
CCU40.IN3BH	I	0	Reserved
CCU40.IN3BI	I	0	Reserved
CCU40.IN3BJ	I	0	Reserved
CCU40.IN3BK	I	0	Reserved
CCU40.IN3BL	I	0	Reserved
CCU40.IN3BM	I	0	Reserved
CCU40.IN3BN	I	0	Reserved
CCU40.IN3BO	I	0	Reserved
CCU40.IN3BP	I	0	Reserved
CCU40.IN3BQ	I	0	Reserved
CCU40.IN3BR	I	0	Reserved
CCU40.IN3BS	I	0	Reserved
CCU40.IN3BT	I	0	Reserved
CCU40.IN3BU	I	0	Reserved
CCU40.IN3BV	I	0	Reserved
CCU40.MCI3	I	BCCU0.OUT5	Multi Channel pattern input

(table continues...)

Table 170 (continued) CCU40 - CC43 Pin Connections

Global Input/Output	I/O	Connected To	Description
CCU40.OUT3	O	P0.3; P0.9; P1.3; P2.11; P2.13; P4.3; P2.2.HW1 pull control; P2.6.HW1 pull control; P2.7.HW1 pull control; P2.11.HW1 direction control;	Slice compare output
CCU40.GP30	O	CCU40.IN2AM	Selected signal for event 0
CCU40.GP31	O	not connected	Selected signal for event 1
CCU40.GP32	O	CCU80.IN3AC	Selected signal for event 2
CCU40.ST3	O	CCU40.IN0AP; CCU40.IN1AP; CCU40.IN2AP; CCU40.IN3AU; CCU41.IN3AL; VADC0.BGREQGTA; VADC0.G0REQGTA; VADC0.G1REQGTA; CCU80.IGBTA; POSIF0.MSETD; ERU1.3B0;	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)

21.8.2 CCU41 Pins

Table 171 CCU41 Pin Connections

Global Input/Output	I/O	Connected To	Description
CCU41.MCLK	I	PCLK	
CCU41.CLKA	I	ERU0.IOUT0	
CCU41.CLKB	I	ERU1.IOUT0	
CCU41.CLKC	I	ERU1.IOUT1	

(table continues...)

Table 171 (continued) CCU41 Pin Connections

Global Input/Output	I/O	Connected To	Description
CCU41.MCSS	I	POSIF1.OUT6	
CCU41.SR0	O	NVIC; CCU81.IGBTD; POSIF1.MSETA; ERU1.OGU01; CCU41.IN0BB;	
CCU41.SR1	O	NVIC; ERU1.OGU11; CCU41.IN2AT; CCU41.IN3AT; CCU41.IN1BB;	
CCU41.SR2	O	NVIC; VADC0.BGREQTRC; VADC0.G0REQTRC; VADC0.G1REQTRC; CCU81.IN0AK; CCU81.IN1AK; ERU1.OGU21; CCU41.IN0AT; CCU41.IN1AT; CCU41.IN2BB;	
CCU41.SR3	O	NVIC; VADC0.BGREQTRD; VADC0.G0REQTRD; VADC0.G1REQTRD; CCU81.IGBTB; CCU81.IN2AK; CCU81.IN3AK; ERU1.OGU31; CCU41.IN3BB;	

Table 172 CCU41 - CC40 Pin Connections

Global Input/Output	I/O	Connected To	Description
CCU41.IN0AA	I	P3.0	General purpose function
CCU41.IN0AB	I	P0.4	General purpose function
Reserved	Reserved	Reserved	Reserved
CCU41.IN0AD	I	ERU0.PDOUT1	General purpose function
CCU41.IN0AE	I	POSIF1.OUT0	General purpose function
CCU41.IN0AF	I	POSIF1.OUT1	General purpose function
CCU41.IN0AG	I	POSIF1.OUT3	General purpose function
CCU41.IN0AH	I	CCU81.ST3;	General purpose function
CCU41.IN0AI	I	SCU.GSC41	General purpose function
CCU41.IN0AJ	I	ERU0.PDOUT0	General purpose function
CCU41.IN0AK	I	ERU0.IOUT0	General purpose function
CCU41.IN0AL	I	CCU40.ST0	General purpose function
CCU41.IN0AM	I	CCU41.GP10	General purpose function
CCU41.IN0AN	I	CCU41.ST1	General purpose function
CCU41.IN0AO	I	CCU41.ST2	General purpose function
CCU41.IN0AP	I	CCU41.ST3	General purpose function

(table continues...)

Table 172 (continued) CCU41 - CC40 Pin Connections

Global Input/Output	I/O	Connected To	Description
CCU41.IN0AQ	I	BCCU0.OUT0	General purpose function
CCU41.IN0AR	I	ACMP1.OUT	General purpose function
CCU41.IN0AS	I	ACMP0.OUT	General purpose function
CCU41.IN0AT	I	CCU41.SR2	General purpose function
CCU41.IN0AU	I	CCU41.ST0	General purpose function
CCU41.IN0AV	I	P4.4	General purpose function
CCU41.IN0AW	I	ERU1.PDOUT0	General purpose function
CCU41.IN0AX	I	ERU1.IOUT0	General purpose function
CCU41.IN0AY	I	ERU1.PDOUT1	General purpose function
CCU41.IN0AZ	I	BCCU0.OUT6	General purpose function
CCU41.IN0BA	I	P4.8	General purpose function
CCU41.IN0BB	I	CCU41.SR0	General purpose function
CCU41.IN0BC	I	USIC1_CH0.DX2INS	General purpose function
CCU41.IN0BD	I	0	Reserved
CCU41.IN0BE	I	0	Reserved
CCU41.IN0BF	I	0	Reserved
CCU41.IN0BG	I	0	Reserved
CCU41.IN0BH	I	0	Reserved
CCU41.IN0BI	I	0	Reserved
CCU41.IN0BJ	I	0	Reserved
CCU41.IN0BK	I	0	Reserved
CCU41.IN0BL	I	0	Reserved
CCU41.IN0BM	I	0	Reserved
CCU41.IN0BN	I	0	Reserved
CCU41.IN0BO	I	0	Reserved
CCU41.IN0BP	I	0	Reserved
CCU41.IN0BQ	I	0	Reserved
CCU41.IN0BR	I	0	Reserved
CCU41.IN0BS	I	0	Reserved
CCU41.IN0BT	I	0	Reserved
CCU41.IN0BU	I	0	Reserved
CCU41.IN0BV	I	0	Reserved
CCU41.MCI0	I	BCCU0.OUT0	Multi Channel pattern input

(table continues...)

Table 172 (continued) CCU41 - CC40 Pin Connections

Global Input/Output	I/O	Connected To	Description
CCU41.OUT0	O	P0.6; P1.4; P3.0; P4.4; P2.12.HW1 pull control;	Slice compare output
CCU41.GP00	O	CCU41.IN3AM	Selected signal for event 0
CCU41.GP01	O	not connected	Selected signal for event 1
CCU41.GP02	O	CCU81.IN0AC	Selected signal for event 2
CCU41.ST0	O	CCU41.IN0AU; CCU41.IN1AN; CCU41.IN2AN; CCU41.IN3AN; CCU81.IGBTC; POSIF1.HSDA; ERU1.0B1; VADC0.BGREQGTG; VADC0.G0REQGTG; VADC0.G1REQGTG;	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 173 CCU41 - CC41 Pin Connections

Global Input/Output	I/O	Connected To	Description
CCU41.IN1AA	I	P3.0	General purpose function
CCU41.IN1AB	I	P0.5	General purpose function
CCU41.IN1AC	I	P4.1	General purpose function
CCU41.IN1AD	I	ERU0.PDOUT0	General purpose function
CCU41.IN1AE	I	POSIF1.OUT0	General purpose function
CCU41.IN1AF	I	POSIF1.OUT1	General purpose function
CCU41.IN1AG	I	POSIF1.OUT3	General purpose function
CCU41.IN1AH	I	POSIF1.OUT4	General purpose function
CCU41.IN1AI	I	SCU.GSC41	General purpose function
CCU41.IN1AJ	I	ERU0.PDOUT1	General purpose function
CCU41.IN1AK	I	ERU0.IOUT1	General purpose function
CCU41.IN1AL	I	CCU40.ST1	General purpose function
CCU41.IN1AM	I	CCU41.GP20	General purpose function
CCU41.IN1AN	I	CCU41.ST0	General purpose function
CCU41.IN1AO	I	CCU41.ST2	General purpose function
CCU41.IN1AP	I	CCU41.ST3	General purpose function
CCU41.IN1AQ	I	BCCU0.OUT1	General purpose function

(table continues...)

Table 173 (continued) CCU41 - CC41 Pin Connections

Global Input/Output	I/O	Connected To	Description
CCU41.IN1AR	I	ACMP3.OUT	General purpose function
CCU41.IN1AS	I	ACMP2.OUT	General purpose function
CCU41.IN1AT	I	CCU41.SR2	General purpose function
CCU41.IN1AU	I	CCU41.ST1	General purpose function
CCU41.IN1AV	I	P4.5	General purpose function
CCU41.IN1AW	I	ERU1.PDOUT1	General purpose function
CCU41.IN1AX	I	ERU1.IOUT1	General purpose function
CCU41.IN1AY	I	ERU1.PDOUT0	General purpose function
CCU41.IN1AZ	I	BCCU0.OUT3	General purpose function
CCU41.IN1BA	I	P4.9	General purpose function
CCU41.IN1BB	I	CCU41.SR1	General purpose function
CCU41.IN1BC	I	USIC1_CH1.DX2INS	General purpose function
CCU41.IN1BD	I	0	Reserved
CCU41.IN1BE	I	0	Reserved
CCU41.IN1BF	I	0	Reserved
CCU41.IN1BG	I	0	Reserved
CCU41.IN1BH	I	0	Reserved
CCU41.IN1BI	I	0	Reserved
CCU41.IN1BJ	I	0	Reserved
CCU41.IN1BK	I	0	Reserved
CCU41.IN1BL	I	0	Reserved
CCU41.IN1BM	I	0	Reserved
CCU41.IN1BN	I	0	Reserved
CCU41.IN1BO	I	0	Reserved
CCU41.IN1BP	I	0	Reserved
CCU41.IN1BQ	I	0	Reserved
CCU41.IN1BR	I	0	Reserved
CCU41.IN1BS	I	0	Reserved
CCU41.IN1BT	I	0	Reserved
CCU41.IN1BU	I	0	Reserved
CCU41.IN1BV	I	0	Reserved
CCU41.MCI1	I	BCCU0.OUT1	Multi Channel pattern input
CCU41.OUT1	O	P0.7; P1.5; P4.1; P4.5;	Slice compare output

(table continues...)

Table 173 (continued) CCU41 - CC41 Pin Connections

Global Input/Output	I/O	Connected To	Description
CCU41.GP10	O	CCU41.IN0AM	Selected signal for event 0
CCU41.GP11	O	not connected	Selected signal for event 1
CCU41.GP12	O	CCU81.IN1AC	Selected signal for event 2
CCU41.ST1	O	CCU41.IN0AN; CCU41.IN1AU; CCU41.IN2AO; CCU41.IN3AO; POSIF1.MSETB; ERU1.1B1;	Slice status bit
CCU41.PS1	O	POSIF1.MSYNCC	Multi channel pattern sync trigger: PM when counting UP (edge aligned) or OM when counting DOWN (center aligned)

Table 174 CCU41 - CC42 Pin Connections

Global Input/Output	I/O	Connected To	Description
CCU41.IN2AA	I	P3.0	General purpose function
CCU41.IN2AB	I	P0.6	General purpose function
CCU41.IN2AC	I	P4.2	General purpose function
CCU41.IN2AD	I	ERU0.PDOUT3	General purpose function
CCU41.IN2AE	I	POSIF1.OUT1	General purpose function
CCU41.IN2AF	I	POSIF1.OUT2	General purpose function
CCU41.IN2AG	I	POSIF1.OUT3	General purpose function
CCU41.IN2AH	I	POSIF1.OUT4	General purpose function
CCU41.IN2AI	I	SCU.GSC41	General purpose function
CCU41.IN2AJ	I	ERU0.PDOUT2	General purpose function
CCU41.IN2AK	I	ERU0.IOUT2	General purpose function
CCU41.IN2AL	I	CCU40.ST2	General purpose function
CCU41.IN2AM	I	CCU41.GP30	General purpose function
CCU41.IN2AN	I	CCU41.ST0	General purpose function
CCU41.IN2AO	I	CCU41.ST1	General purpose function
CCU41.IN2AP	I	CCU41.ST3	General purpose function
CCU41.IN2AQ	I	BCCU0.OUT2	General purpose function
CCU41.IN2AR	I	ACMP2.OUT	General purpose function
CCU41.IN2AS	I	ACMP1.OUT	General purpose function
CCU41.IN2AT	I	CCU41.SR1	General purpose function
CCU41.IN2AU	I	CCU41.ST2	General purpose function

(table continues...)

Table 174 (continued) CCU41 - CC42 Pin Connections

Global Input/Output	I/O	Connected To	Description
CCU41.IN2AV	I	P4.6	General purpose function
CCU41.IN2AW	I	ERU1.PDOUT2	General purpose function
CCU41.IN2AX	I	ERU1.IOUT2	General purpose function
CCU41.IN2AY	I	ERU1.PDOUT3	General purpose function
CCU41.IN2AZ	I	BCCU0.OUT7	General purpose function
Reserved	Reserved	Reserved	Reserved
CCU41.IN2BB	I	CCU41.SR2	General purpose function
CCU41.IN2BC	I	POSIF1.OUT0	General purpose function
CCU41.IN2BD	I	0	Reserved
CCU41.IN2BE	I	0	Reserved
CCU41.IN2BF	I	0	Reserved
CCU41.IN2BG	I	0	Reserved
CCU41.IN2BH	I	0	Reserved
CCU41.IN2BI	I	0	Reserved
CCU41.IN2BJ	I	0	Reserved
CCU41.IN2BK	I	0	Reserved
CCU41.IN2BL	I	0	Reserved
CCU41.IN2BM	I	0	Reserved
CCU41.IN2BN	I	0	Reserved
CCU41.IN2BO	I	0	Reserved
CCU41.IN2BP	I	0	Reserved
CCU41.IN2BQ	I	0	Reserved
CCU41.IN2BR	I	0	Reserved
CCU41.IN2BS	I	0	Reserved
CCU41.IN2BT	I	0	Reserved
CCU41.IN2BU	I	0	Reserved
CCU41.IN2BV	I	0	Reserved
CCU41.MCI2	I	BCCU0.OUT2	Multi Channel pattern input
CCU41.OUT2	O	P0.8; P1.6; P4.2; P4.6; P2.13.HW1 pull control;	Slice compare output
CCU41.GP20	O	CCU41.IN1AM	Selected signal for event 0
CCU41.GP21	O	not connected	Selected signal for event 1
CCU41.GP22	O	CCU81.IN2AC	Selected signal for event 2

(table continues...)

Table 174 (continued) CCU41 - CC42 Pin Connections

Global Input/Output	I/O	Connected To	Description
CCU41.ST2	O	CCU41.IN0AO; CCU41.IN1AO; CCU41.IN2AU; CCU41.IN3AP; POSIF1.MSETC; ERU1.2B1;	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 175 CCU41 - CC43 Pin Connections

Global Input/Output	I/O	Connected To	Description
CCU41.IN3AA	I	P3.0	General purpose function
CCU41.IN3AB	I	P0.7	General purpose function
CCU41.IN3AC	I	P4.3	General purpose function
CCU41.IN3AD	I	ERU0.PDOUT2	General purpose function
CCU41.IN3AE	I	POSIF1.OUT3	General purpose function
CCU41.IN3AF	I	POSIF1.OUT5	General purpose function
CCU41.IN3AG	I	VADC0.G0ARBCNT	General purpose function
CCU41.IN3AH	I	CCU81.IGBTO	General purpose function
CCU41.IN3AI	I	SCU.GSC41	General purpose function
CCU41.IN3AJ	I	ERU0.PDOUT3	General purpose function
CCU41.IN3AK	I	ERU0.IOUT3	General purpose function
CCU41.IN3AL	I	CCU40.ST3	General purpose function
CCU41.IN3AM	I	CCU41.GP00	General purpose function
CCU41.IN3AN	I	CCU41.ST0	General purpose function
CCU41.IN3AO	I	CCU41.ST1	General purpose function
CCU41.IN3AP	I	CCU41.ST2	General purpose function
CCU41.IN3AQ	I	BCCU0.OUT5	General purpose function
CCU41.IN3AR	I	ACMP0.OUT	General purpose function
CCU41.IN3AS	I	ACMP3.OUT	General purpose function
CCU41.IN3AT	I	CCU41.SR1	General purpose function
CCU41.IN3AU	I	CCU41.ST3	General purpose function
CCU41.IN3AV	I	P4.7	General purpose function
CCU41.IN3AW	I	ERU1.PDOUT3	General purpose function
CCU41.IN3AX	I	ERU1.IOUT3	General purpose function

(table continues...)

Table 175 (continued) CCU41 - CC43 Pin Connections

Global Input/Output	I/O	Connected To	Description
CCU41.IN3AY	I	ERU1.PDOUT2	General purpose function
CCU41.IN3AZ	I	BCCU0.OUT8	General purpose function
Reserved	Reserved	Reserved	Reserved
CCU41.IN3BB	I	CCU41.SR3	General purpose function
CCU41.IN3BC	I	POSIF1.OUT0	General purpose function
CCU41.IN3BD	I	POSIF1.OUT1	General purpose function
CCU41.IN3BE	I	0	Reserved
CCU41.IN3BF	I	0	Reserved
CCU41.IN3BG	I	0	Reserved
CCU41.IN3BH	I	0	Reserved
CCU41.IN3BI	I	0	Reserved
CCU41.IN3BJ	I	0	Reserved
CCU41.IN3BK	I	0	Reserved
CCU41.IN3BL	I	0	Reserved
CCU41.IN3BM	I	0	Reserved
CCU41.IN3BN	I	0	Reserved
CCU41.IN3BO	I	0	Reserved
CCU41.IN3BP	I	0	Reserved
CCU41.IN3BQ	I	0	Reserved
CCU41.IN3BR	I	0	Reserved
CCU41.IN3BS	I	0	Reserved
CCU41.IN3BT	I	0	Reserved
CCU41.IN3BU	I	0	Reserved
CCU41.IN3BV	I	0	Reserved
CCU41.MCI3	I	BCCU0.OUT5	Multi Channel pattern input
CCU41.OUT3	O	P0.9; P2.13; P4.3; P4.7;	Slice compare output
CCU41.GP30	O	CCU41.IN2AM	Selected signal for event 0
CCU41.GP31	O	not connected	Selected signal for event 1
CCU41.GP32	O	CCU81.IN3AC	Selected signal for event 2
CCU41.ST3	O	CCU41.IN0AP; CCU41.IN1AP; CCU41.IN2AP; CCU41.IN3AU; CCU81.IGBTA; POSIF1.MSETD; ERU1.3B1;	Slice status bit

(table continues...)

Table 175 (continued) CCU41 - CC43 Pin Connections

Global Input/Output	I/O	Connected To	Description
CCU41.PS3	O	not connected	Multi channel pattern sync trigger: PM when counting UP (edge aligned) or OM when counting DOWN (center aligned)

22 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 176 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*

22.1 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.

22.1.1 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

- flexible coherent and non coherent update mechanism
- 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 177 a summary of the major features of the CCU8 unit mapped with the most common applications.

Table 177 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 Timer Slice 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
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, for example,, one timer working in capture mode and one working in compare

(table continues...)

Table 177 (continued) Applications summary

Feature	Applications
Flexible coherent and non coherent update schemes	Flexible profiles that can be used for: • fuel injection control • multiple observation points for motor shaft rotation • on-the-fly compensation linked to real-time operation condition sensing • cpu load optimization for fast control loops
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.
4 dedicated service request lines	Specially developed for: • generating interrupts for the microprocessor • flexible connectivity between peripherals, for example,, ADC triggering.
Linking with POSIF	Flexible profiles for: • Rotary Encoder connection • Hall Sensor • Modulating the 4 timer outputs via SW

22.1.2 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, for example, 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 287.

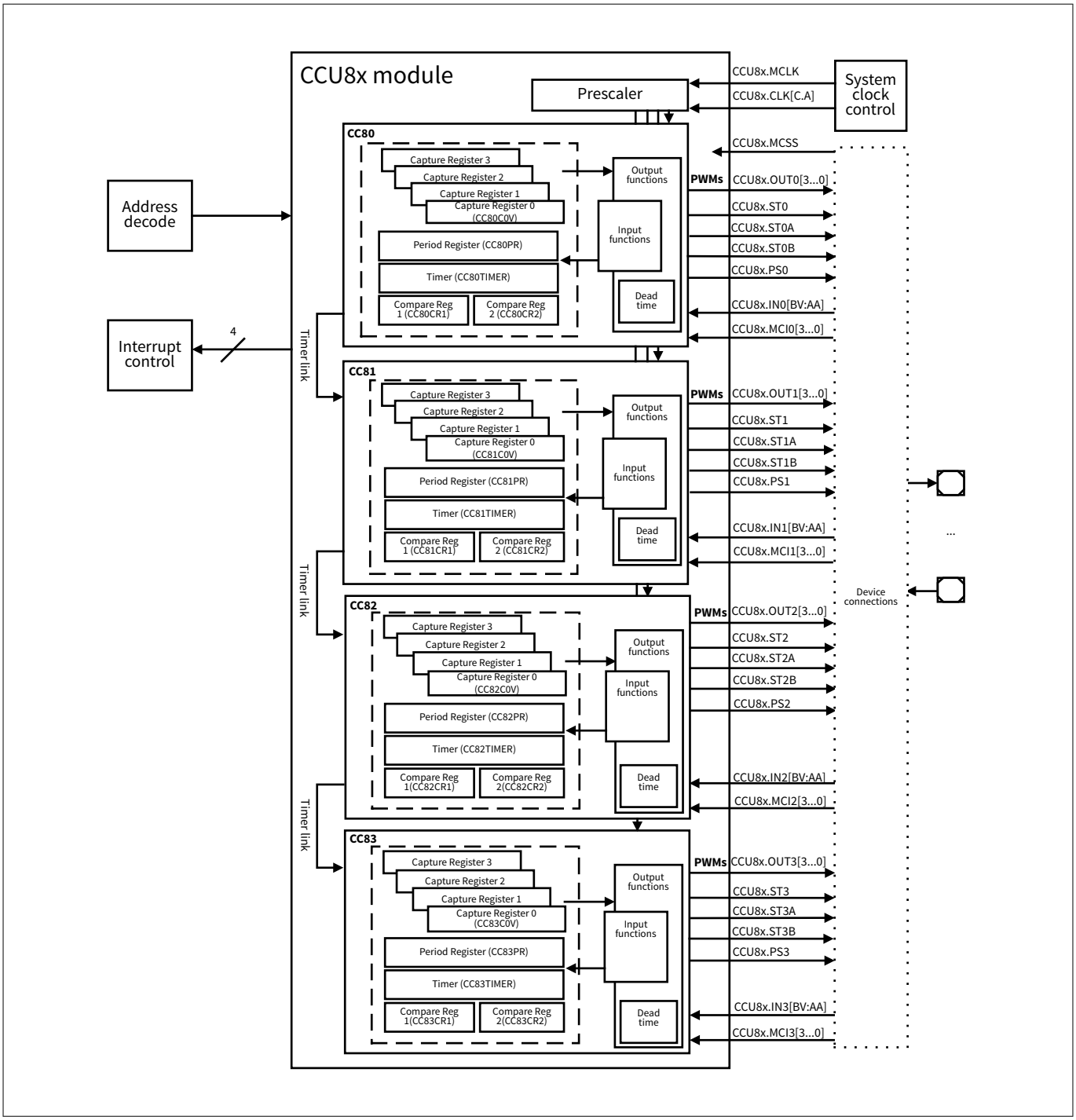


Figure 287 CCU8 block diagram

22.2 Functional Description

The following sections describe the complete set of functions and usability of the CCU8 peripheral.

In each figure several registers may be depicted to indicate controlability or configurability. These registers follow the description given in Figure 288. One should also note that indexing in a register can be done via the non capital y, x or n.

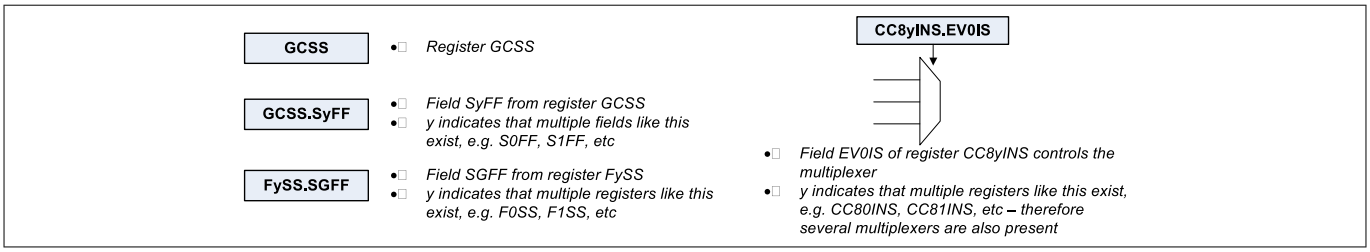


Figure 288 Register description in figures (example)

22.2.1 Timer Slice Overview

The input path of a CCU8 slice is comprised of a selector (Timer Slice Input Selector) and a connection matrix unit (Timer Slice Connection Matrix). 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 289.

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.

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 Timer Slice Connection Matrix.

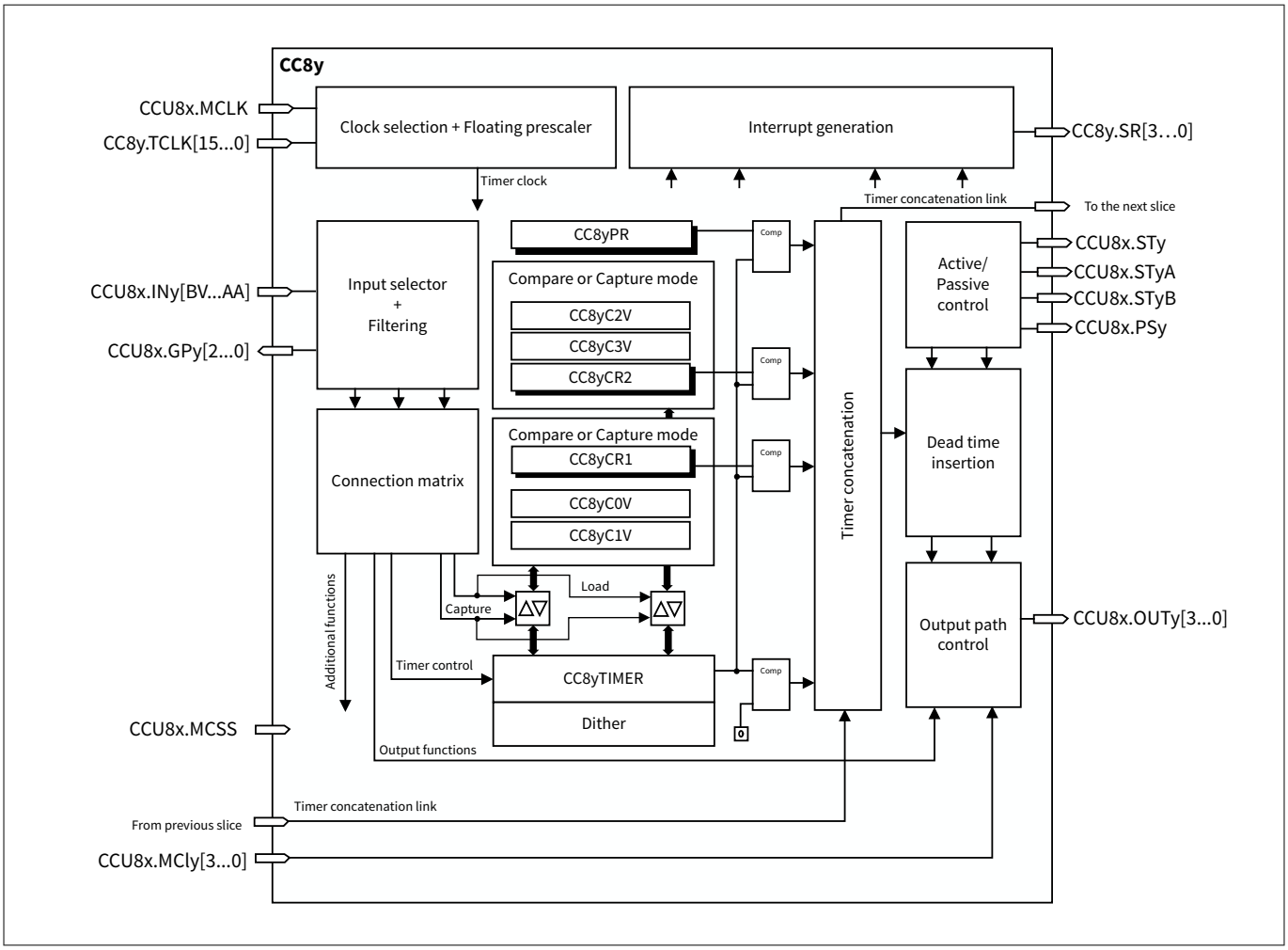


Figure 289 CCU8 slice block diagram

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 178 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 178 CCU8 slice pin description

Pin	I/O	Description
CCU8x.MCLK	I	Module clock
CC8y.TCLK[15:0]		Clocks from the prescaler
CCU8x.INy[BV:AA]	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

(table continues...)

Table 178 (continued) CCU8 slice pin description

Pin	I/O	Description
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

***Notes**:*

1. *The status bit outputs of the Kernel, CCU8x.STy, CCU8x.STyA and CCU8x.STyB are extended for one more kernel clock cycle.*
2. *The Service Request signals at the output of the kernel are extended for one more kernel clock cycle.*
3. *The maximum output signal frequency of the CCU8x.STy, CCU8x.STyA and CCU8x.STyA is module clock divided by 4.*

22.2.2 Timer Slice 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 290.

The user has the possibility of selecting any of the CCU8x.INy[BV:AA] 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.

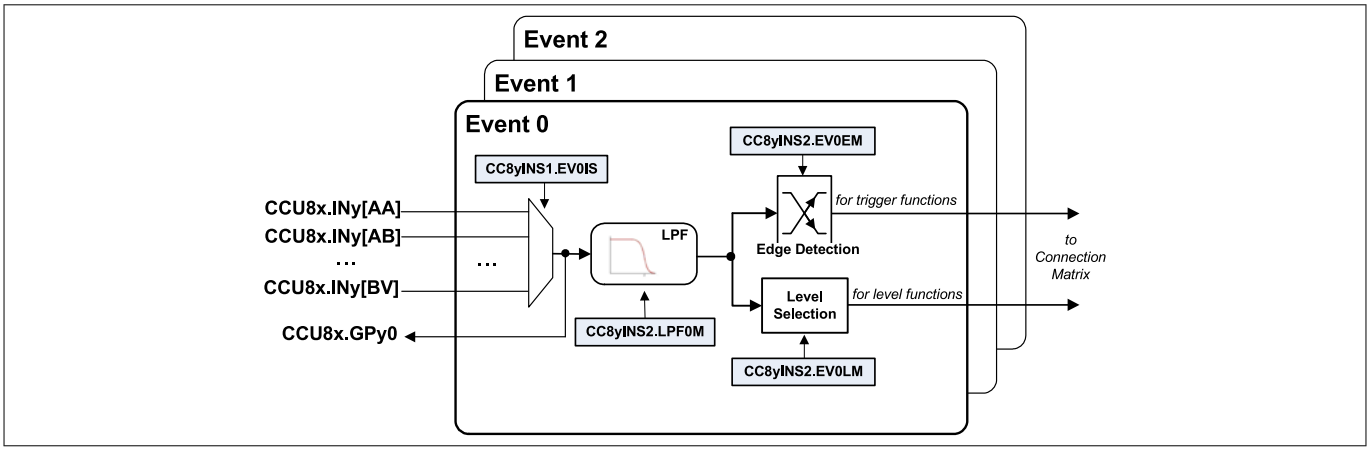


Figure 290 Slice input selector diagram

22.2.3 Timer Slice Connection Matrix

The connection matrix maps the events coming from the input selector to several user configured functions, Figure 291. The following functions can be enabled on the connection matrix:

Table 179 Connection matrix available functions

Function	Brief description	Map to figure Figure 291
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 Timer Concatenation.

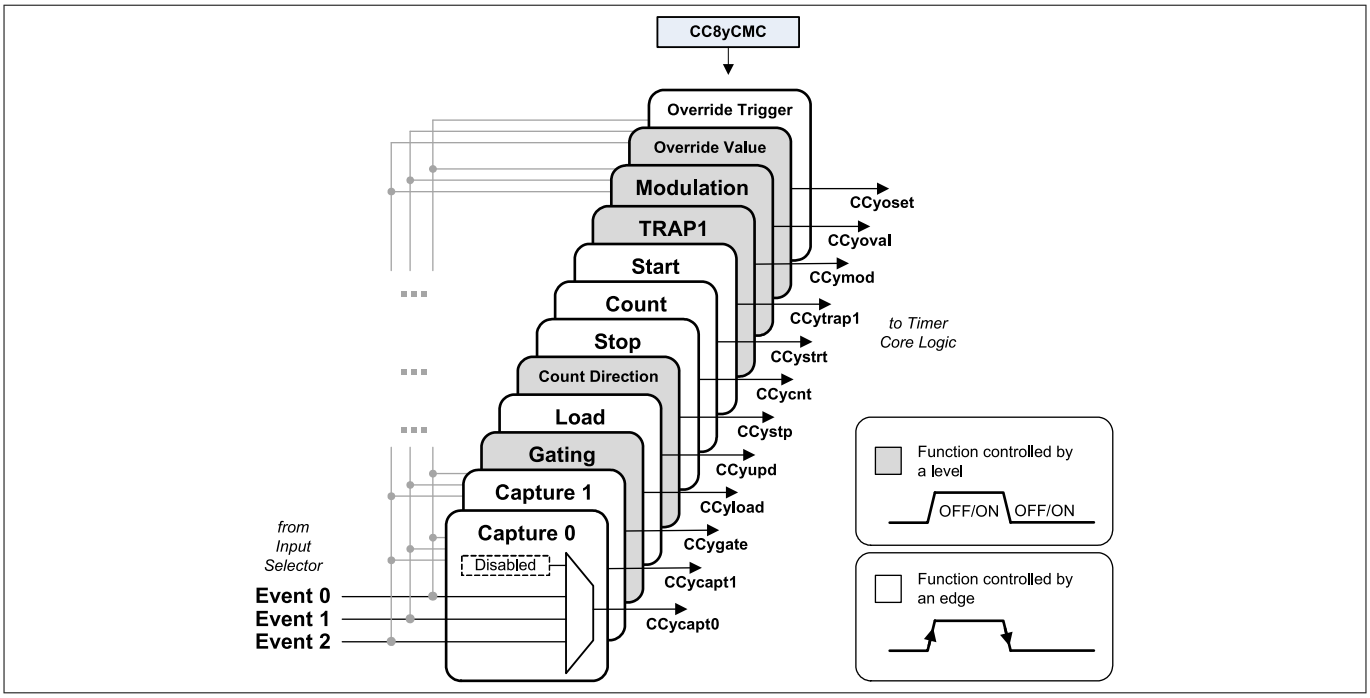


Figure 291 Slice connection matrix diagram

22.2.4 Timer Slice Core Functions

In the following sub sections one can find the description of the main functionalities of each CCU8 Timer Slice core. This functions include the different counting schemes, how to start and stop a timer or how to calculate and update the duty cycle of each Timer Slice.

Besides these functions, there are several others that can be controlled via external signals, for example, port pins or other peripherals - these functions are described in Timer Slice External Functions.

22.2.4.1 Starting/Stopping the Timer

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 292.

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.

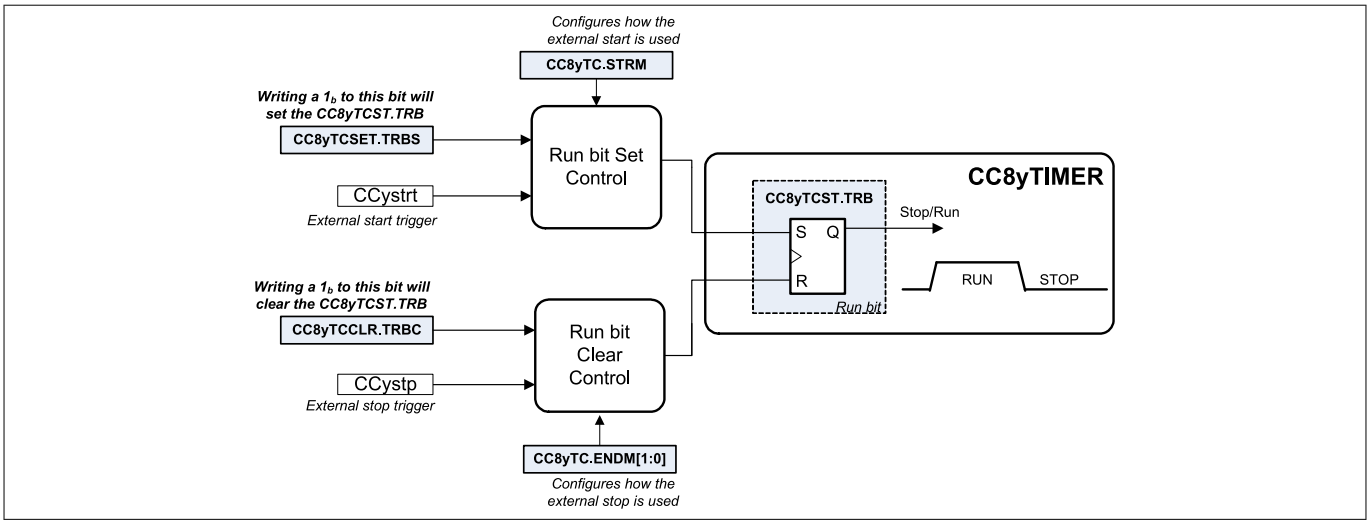


Figure 292 Timer start/stop control diagram

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 External Start/Stop for a description how to configure an input as start function). This input should be connected to all the Timers that need to be started synchronously (see Interconnects for a complete list of module connections), Figure 293.

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 External Start/Stop) and then the software must write $1_B/0_B$ (depending on the external start sunction configuration) to the specific bitfield of the CCUCON register (this register is described on the SCU chapter).

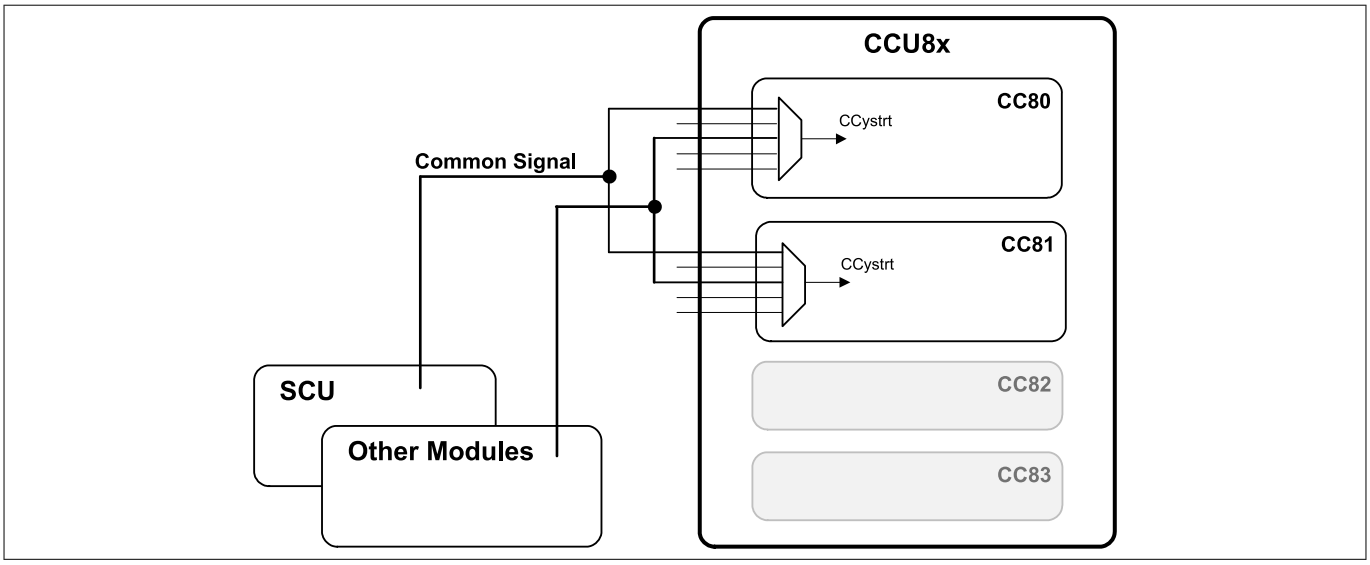


Figure 293 Start multiple timers synchronously

22.2.4.2 Counting Modes Introduction

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 294. 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. Please address the different counting mode descriptions - Edge Aligned Mode to Single Shot Mode - to understand how these bits are set and cleared, and to have a full description of the timer behavior.

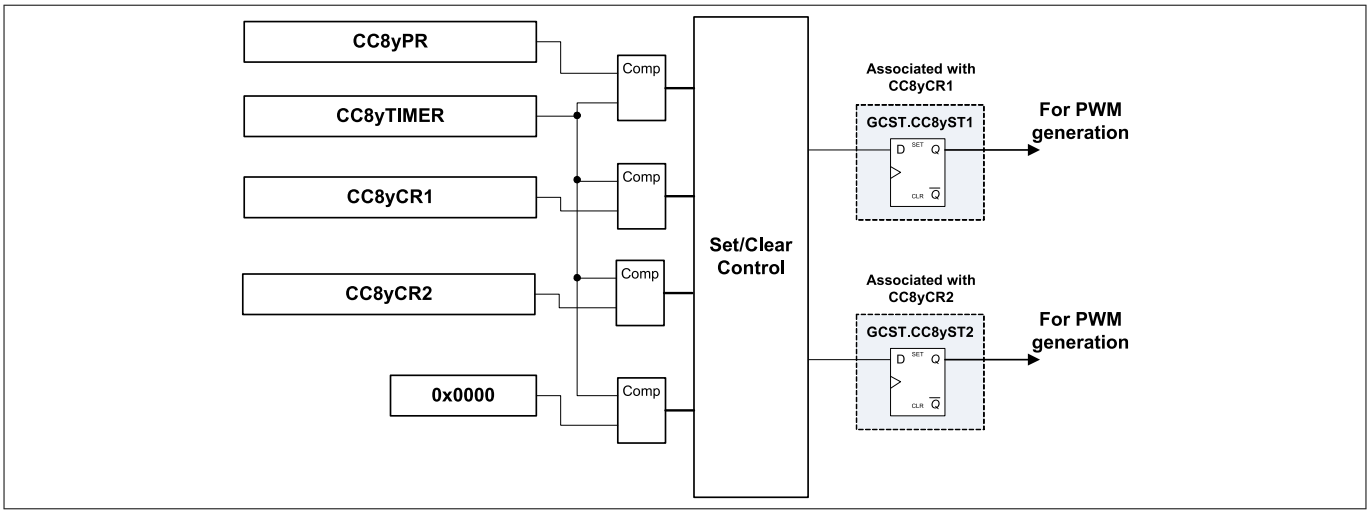


Figure 294 CC8y Status Bits

22.2.4.3 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 - Figure 295.

In edge aligned mode, the status bit of the comparison (CC8ySTn) 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.

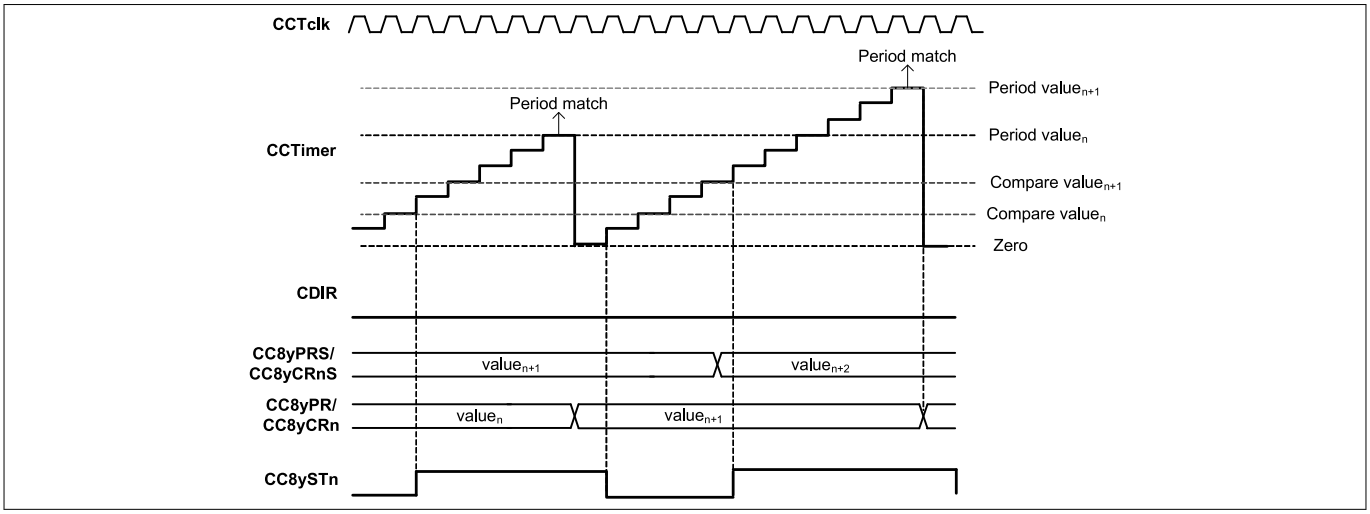


Figure 295 Edge aligned mode, CC8yTC.TCM = 0_B

In edge aligned mode, the value of the period register and compare register are updated with the value written by software into the corresponding shadow register (CC8yPRS and CC8yCRnS), every time that an overflow occurs (period match) - Figure 295.

If an immediate update of the compare value and/or period is needed for the application, then the **CC8ySTC**.IRCCn and **CC8ySTC**.IRPC need to be configure to 1_B - Figure 296 and Figure 297.

A complete description how to update the compare and period values is given in Updating the Period, Duty Cycle and other PWM conditions.

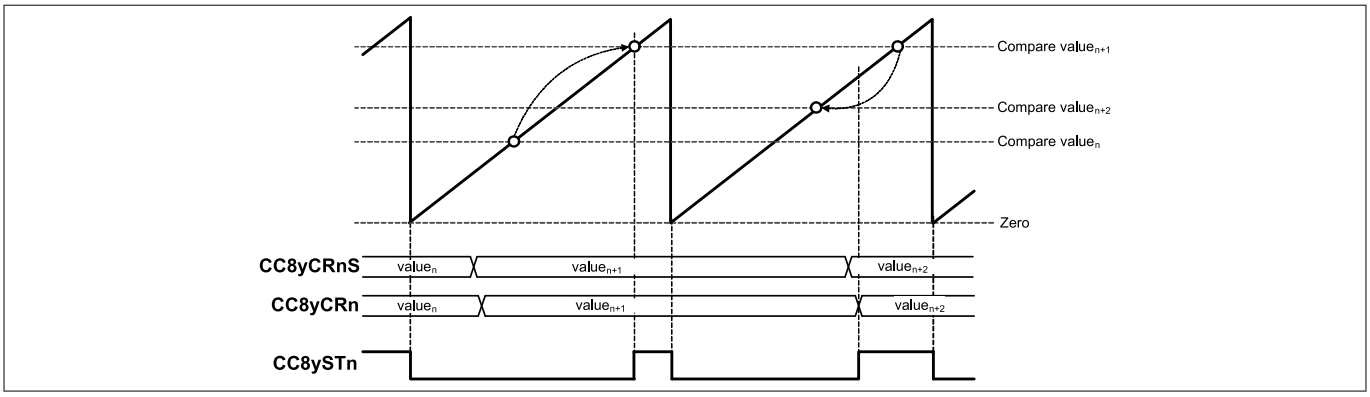


Figure 296 Edge aligned mode, immediate compare value update

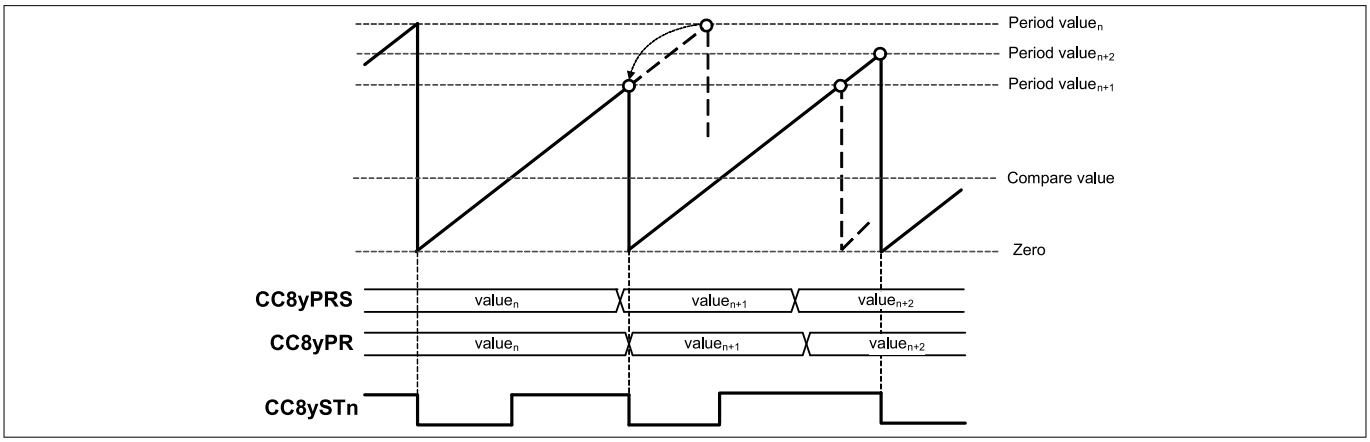


Figure 297 Edge aligned mode, immediate period value update

22.2.4.4 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 (CC8ySTn) is always 1_B when the counter value is equal or greater than the compare value and 0_B otherwise.

***Note**: The 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.*

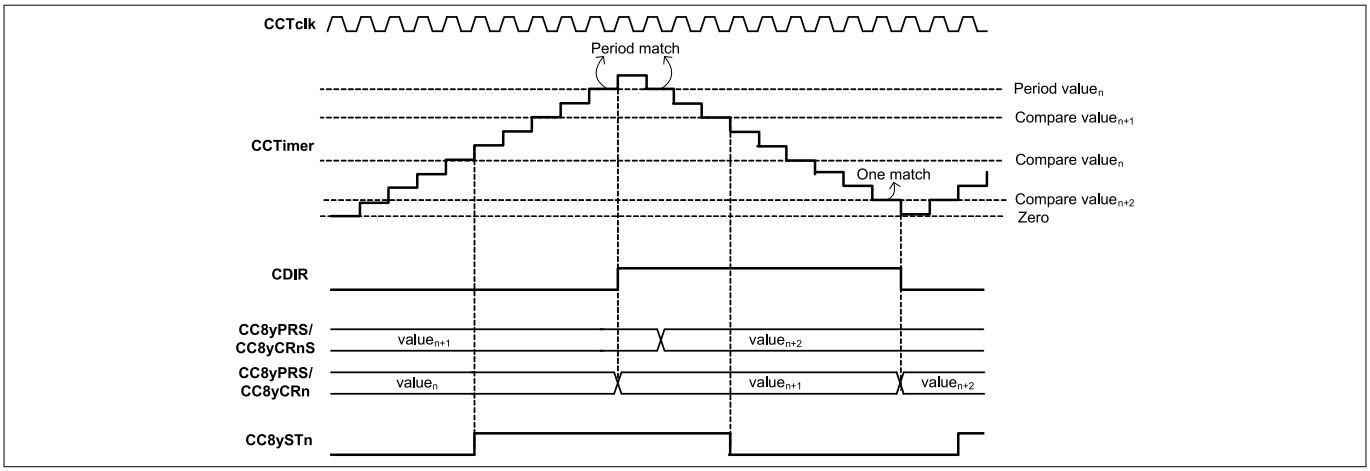


Figure 298 Center aligned mode, CC8yTC.TCM = 1_B

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).

It is also possible to select in which instant the update is done. This is done by configuring the **CC8ySTC**.STM field accordingly.

If an immediate update of the compare value and/or period is needed for the application, then the **CC8ySTC**.IRCCn and **CC8ySTC**.IRPC need to be configure to 1_B - Figure 299 and Figure 300.

A complete description how to update the compare and period values is given in Updating the Period, Duty Cycle and other PWM conditions.

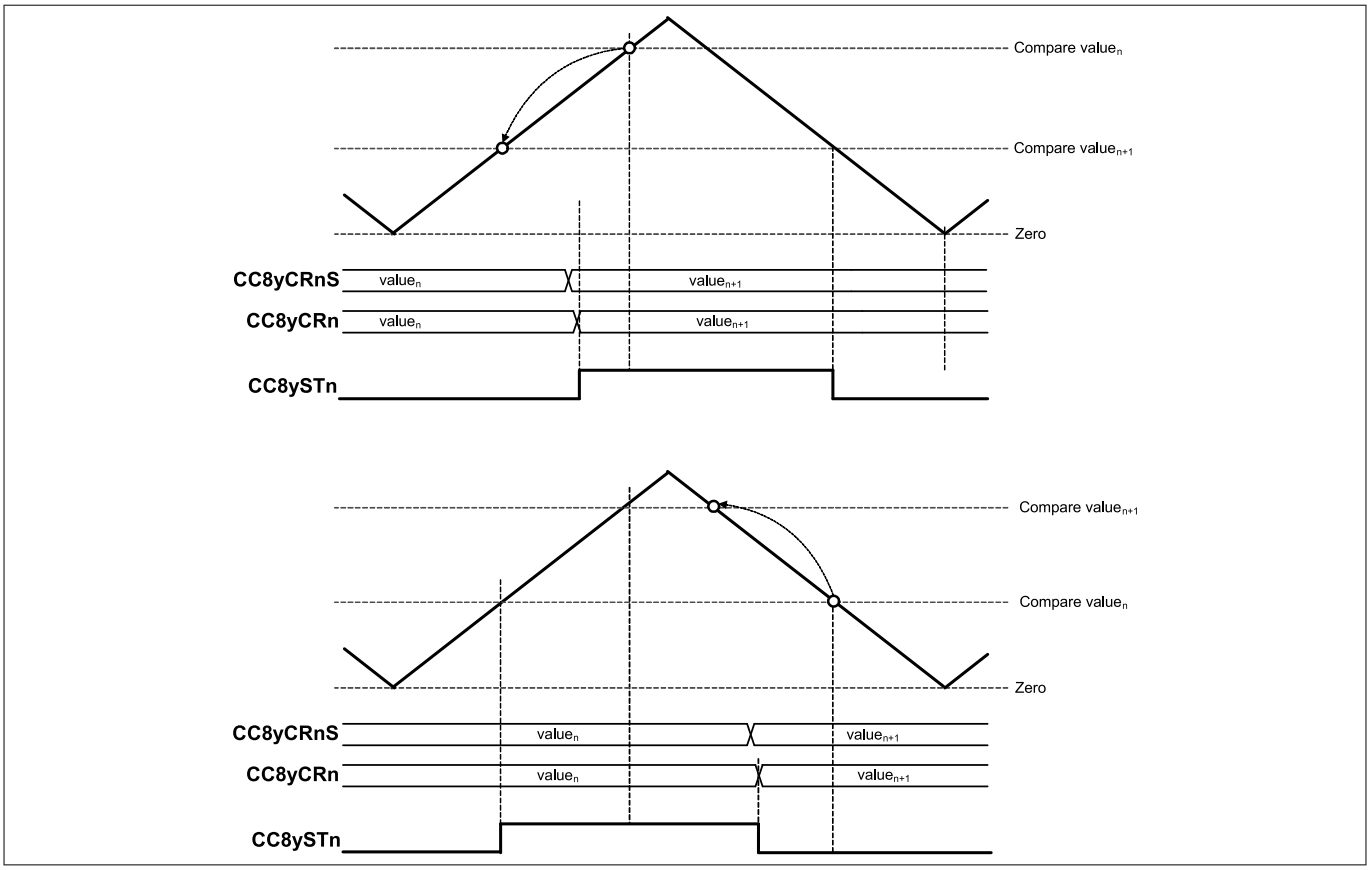


Figure 299 Center aligned mode, immediate compare value update

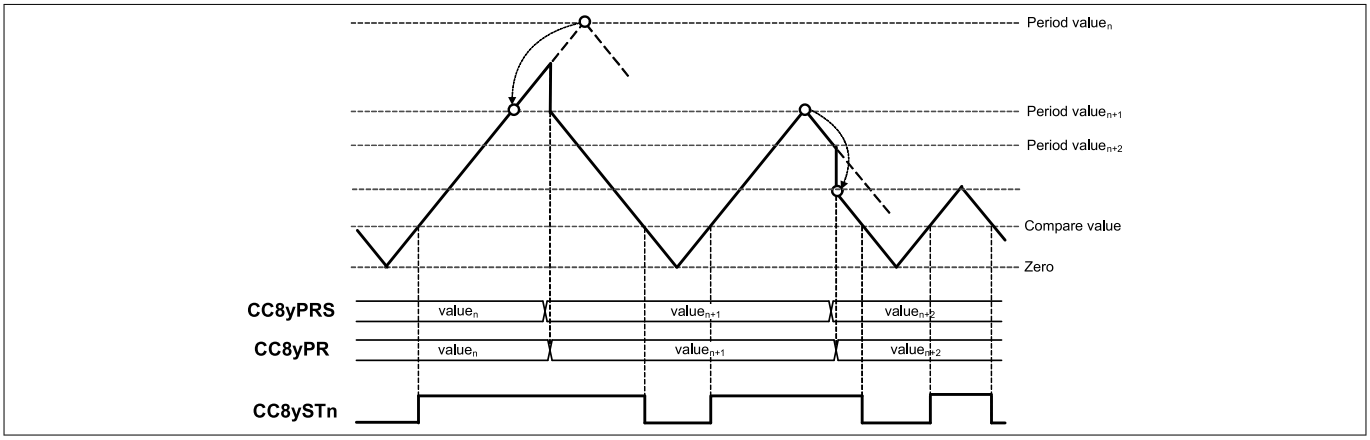


Figure 300 Center aligned mode, immediate period value update

22.2.4.5 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 301, the timer is stopped when it is cleared to 0000_H after having reached the period value. In center aligned mode, Figure 302, the period is finished when the timer has counted down to 0000_H.

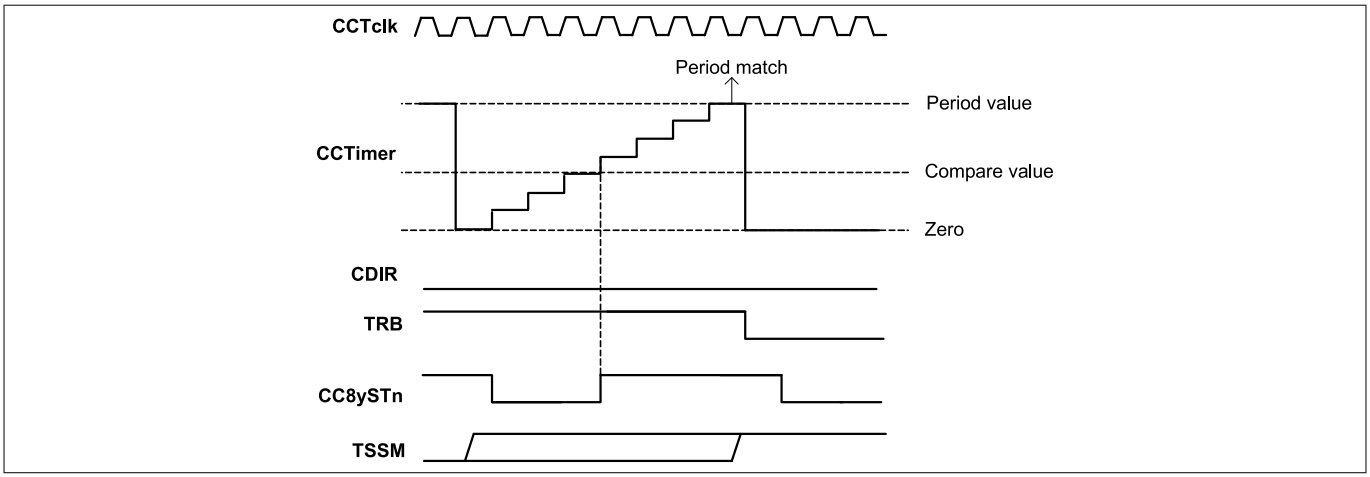


Figure 301 Single shot edge aligned - CC8yTC.TSSM = 1_B, CC8yTC.TCM = 0_B

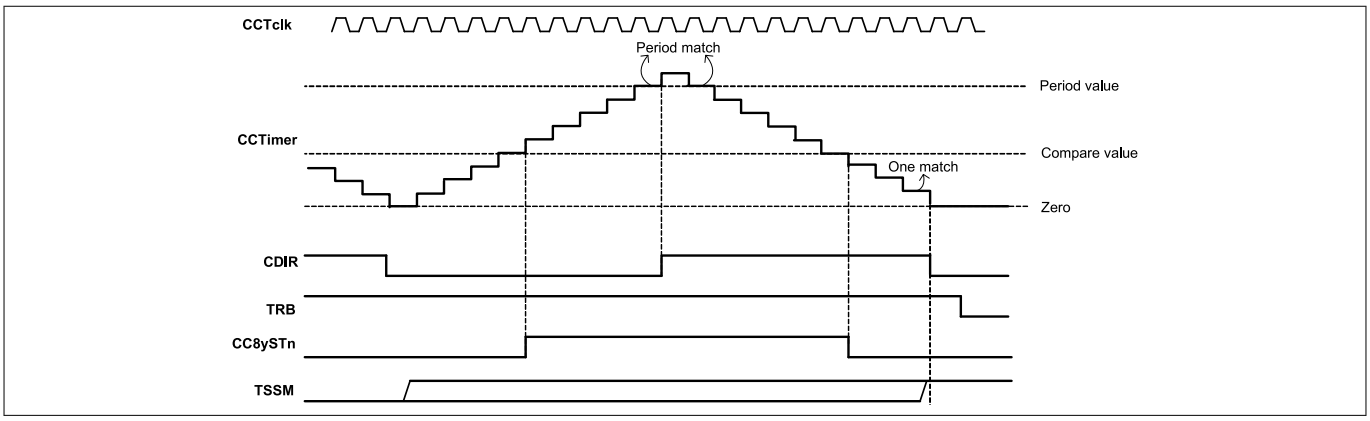


Figure 302 Single shot center aligned - CC8yTC.TSSM = 1_B, CC8yTC.TCM = 1_B

22.2.4.6 Dead Time Generation

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 two dead time generators - one for each compare channel, that are able to delay the switching edges between complementary output signals - Figure 303 and Figure 304.

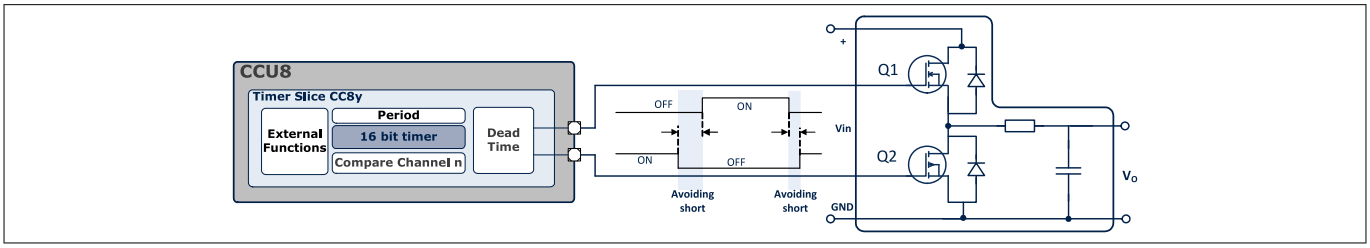


Figure 303 PWM Dead Time insertion

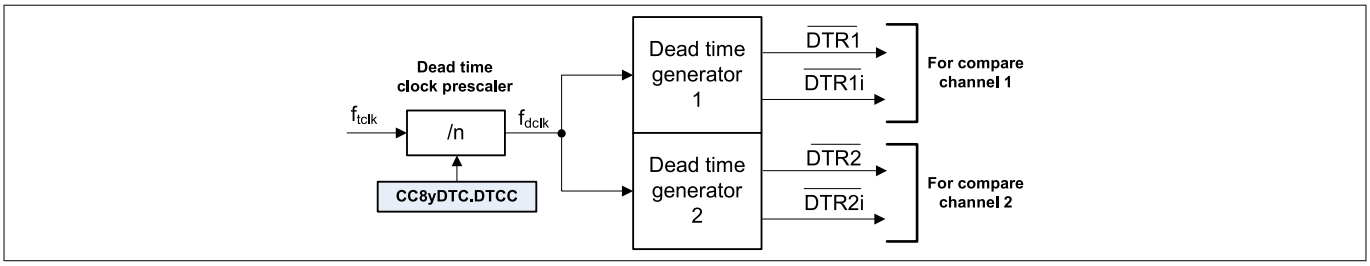


Figure 304 Dead Time scheme

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 180.

Table 180 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, CC8ySTn, will trigger the start of the specific dead time generator, Figure 305.

When a SET (CC8ySTn passes from 0_B to 1_B) action for the CC8ySTn 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 CC8ySTn bit is detected (CC8ySTn 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).

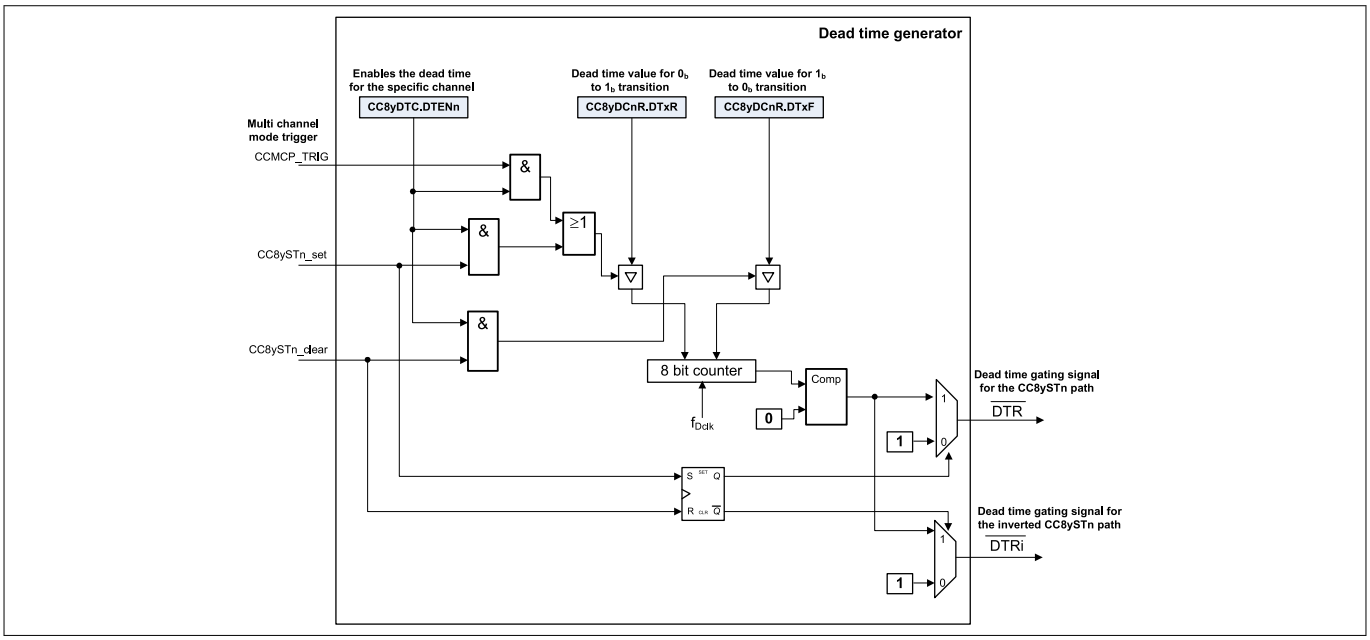


Figure 305 Dead Time generator scheme

Each dead time generator outputs two signals that are used to control the two complementary outputs (the CC8ySTn and the inverted CC8ySTn). The separation of the control signals enable a flexible enable/disable scheme inside of each compare channel, Figure 306. 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.

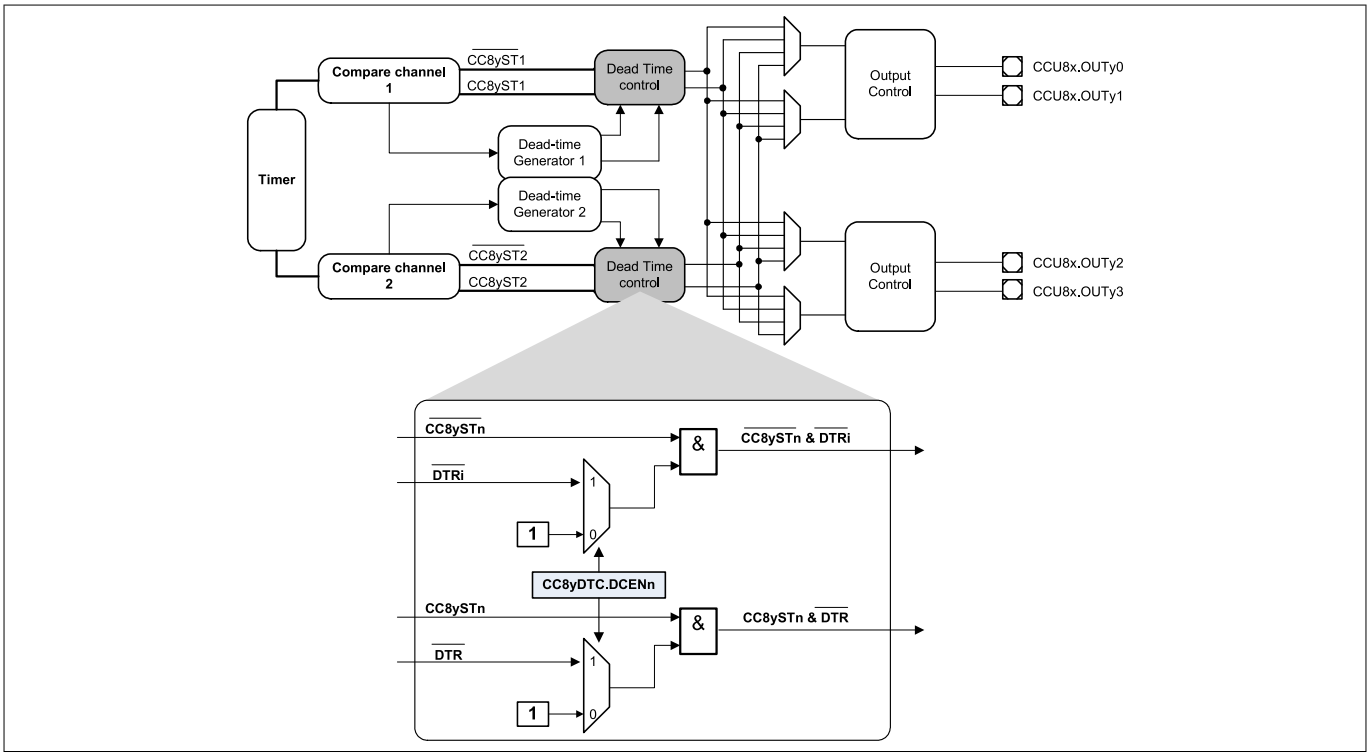


Figure 306 Dead Time control cell

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 305, is fed to the dead time generators. Figure 307 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.

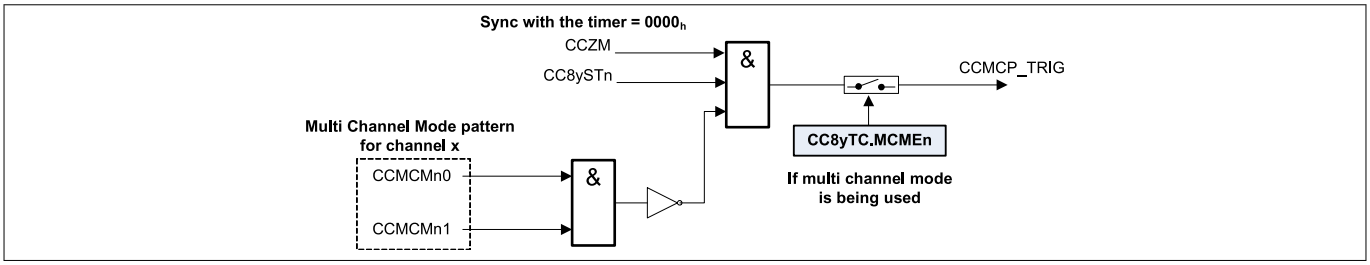


Figure 307 Dead Time trigger with the Multi-Channel pattern

22.2.4.7 Edge and Center Aligned Compare Modes

In this section a description of how the two compare channel scheme can be used to generate PWM signals. There are two major arrangments for the channels:

- symmetric PWM generation - normal mode where each compare channel can generate a pair of complementary signals
- asymmetric PWM generation - a mode where the two channels are groupped together so an asymmetric PWM can be generated by hardware

Each of these modes is applicable in the previously described counting schemes - Edge or Center Aligned Mode.

Compare Channel Scheme Introduction

Each CCU8 slice has two compare channels and two dead time generators, one for each channel, see Figure 308. Each compare uses the information of the status bit, CC8ySTn, 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.

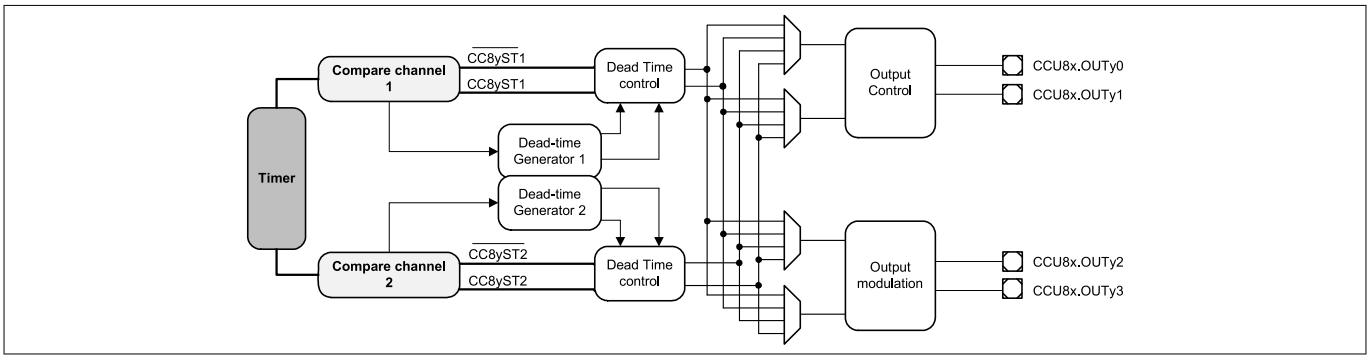


Figure 308 Compare channels diagram

Symmetric 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 309.

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 310 shows the timing diagrams for a specific slice when the compare values of each channel are different.

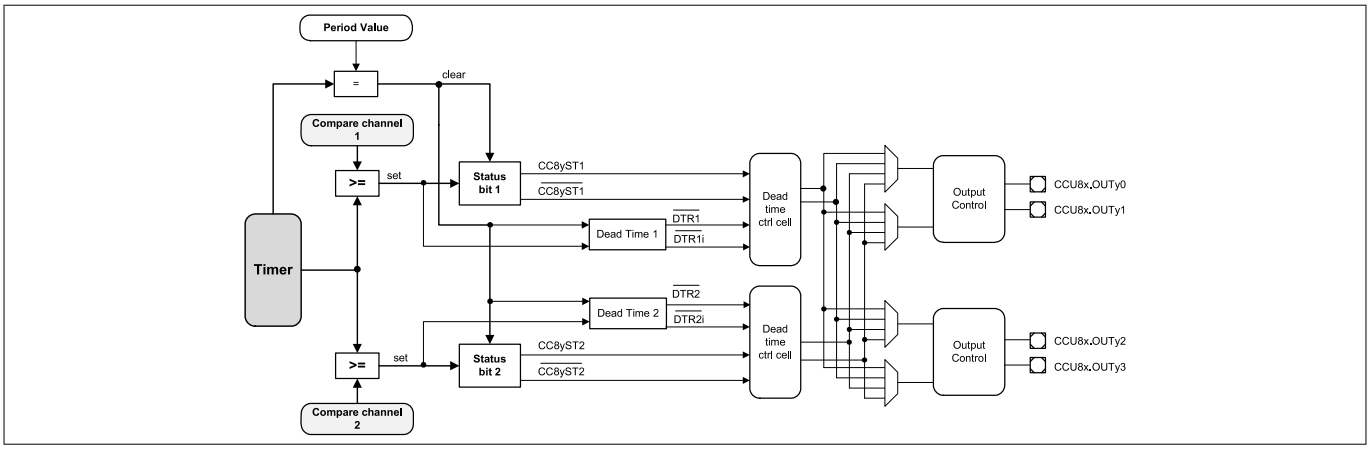


Figure 309 Edge Aligned with two independent channels scheme

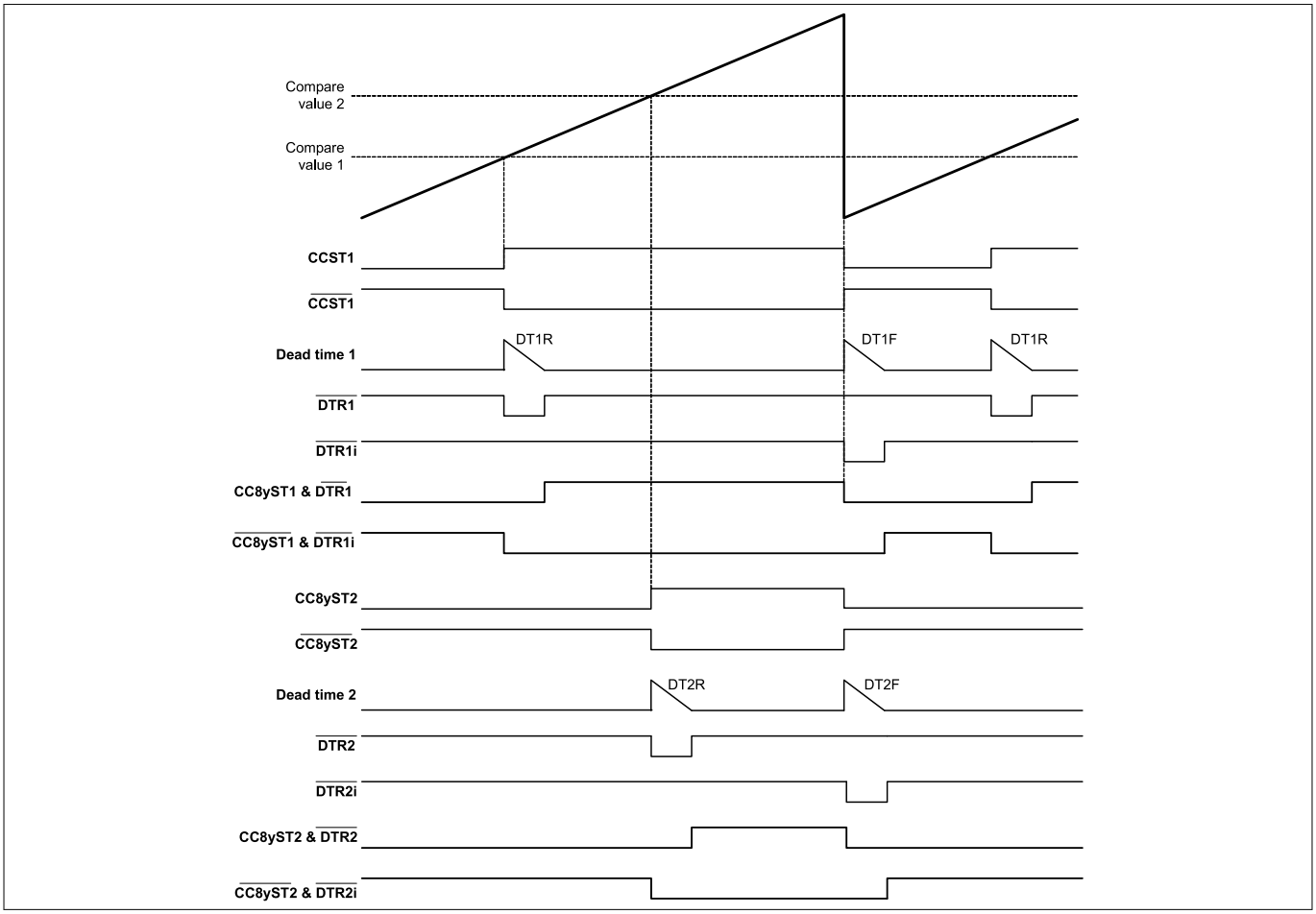


Figure 310 Edge Aligned - four outputs with dead time

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.

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 311.

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 312 and Figure 313 show the timing diagram for the Edge Aligned mode when the asymmetric scheme is active.

In this mode the outputs linked with the compare channel 2 are running and available at the outputs - they behave like a non asymmetrical compare channel.

***Note**: When an external signal is used to control the counting direction, the asymmetric mode cannot be enabled.*

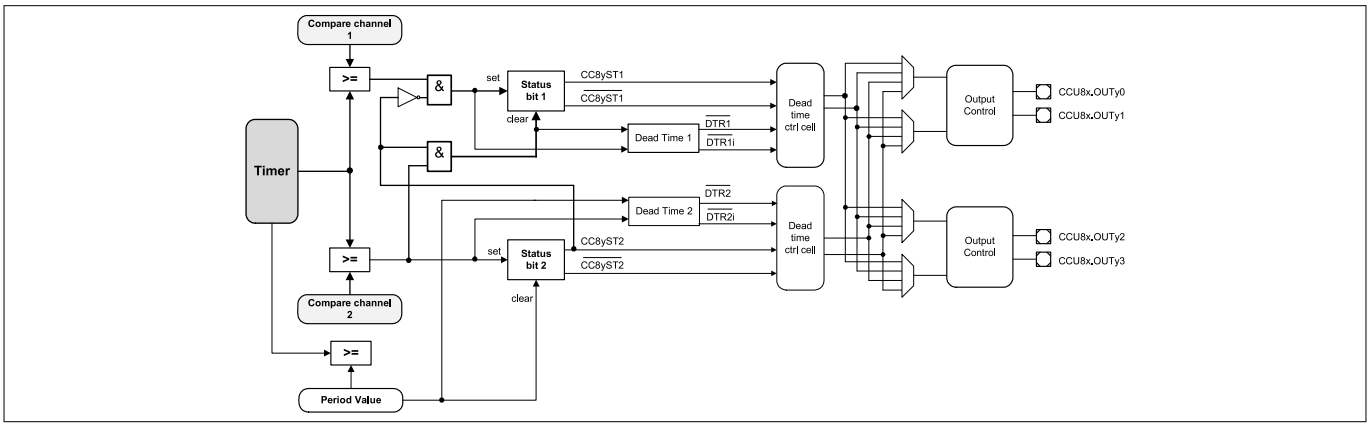


Figure 311 Edge Aligned with combined channels scheme

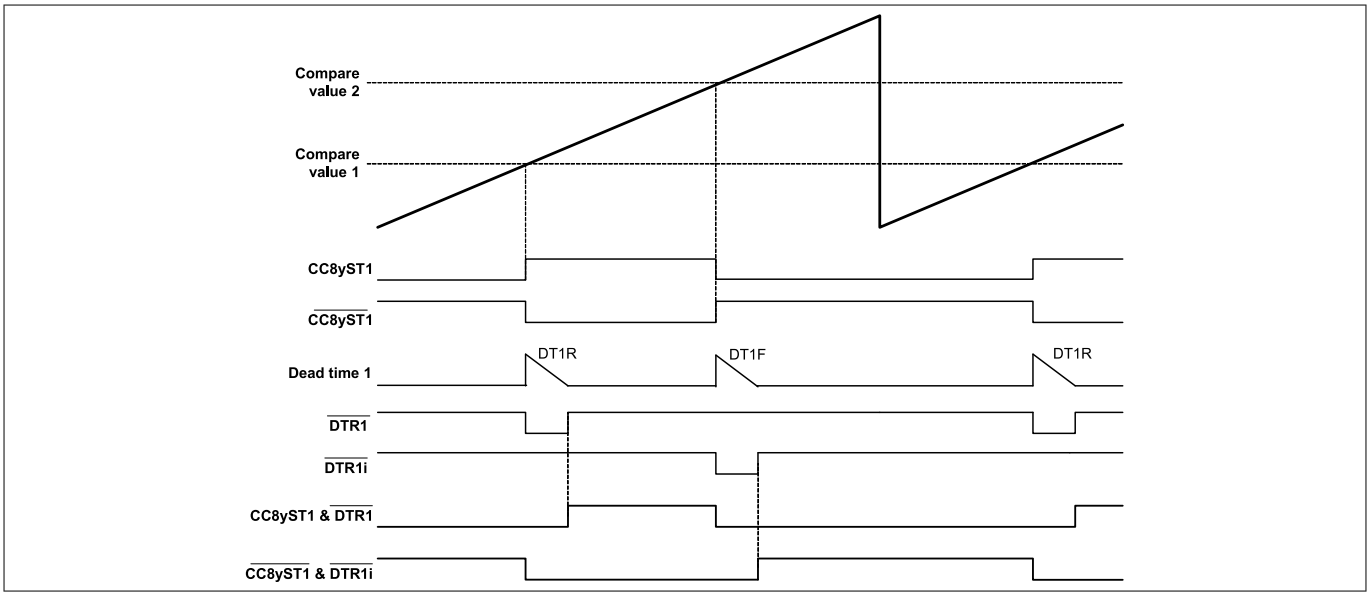


Figure 312 Edge Aligned - Asymmetric PWM timing, CC8yCR1.CR1 < CC8yCR2.CR2

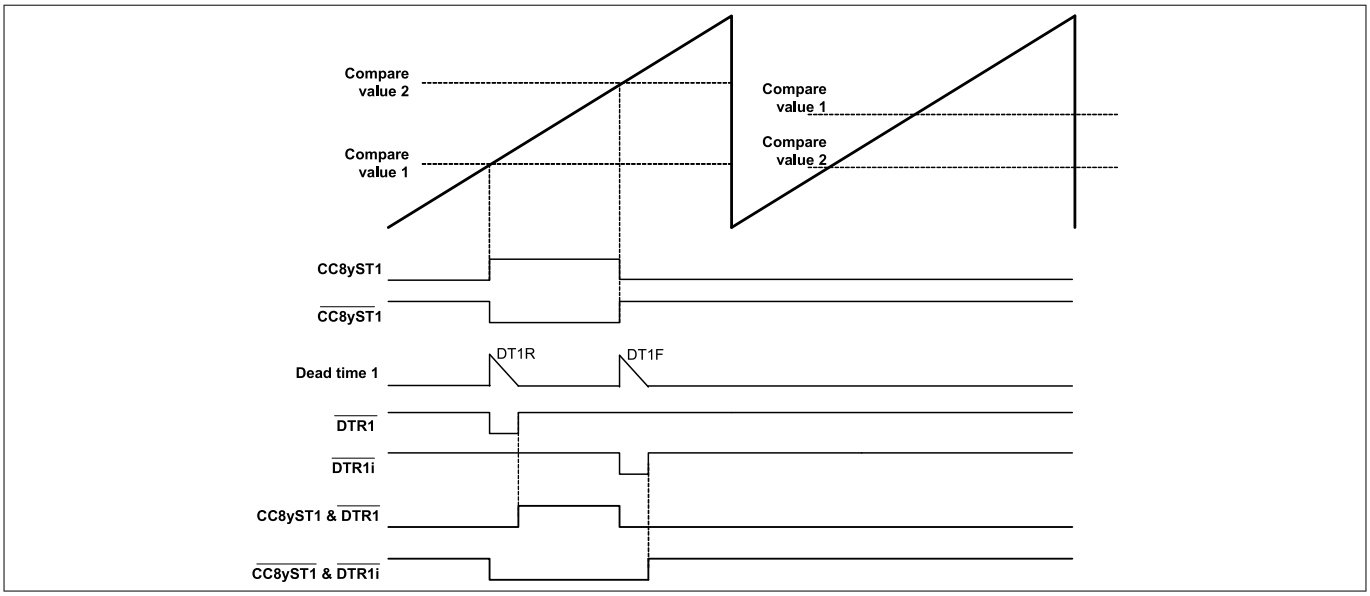


Figure 313 Edge Aligned - Asymmetric PWM timing, CC8yCR1.CR1 > CC8yCR2.CR2

Symmetric 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 has 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 Counting Modes Introduction.

Figure 314 shows the scheme for both channels for this operating mode and Figure 315 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.*

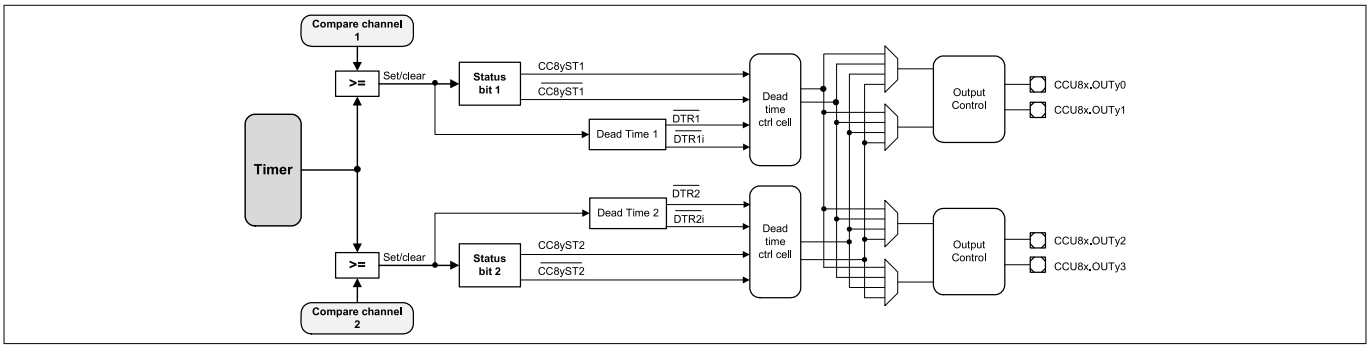


Figure 314 Center Aligned with two independent channels scheme

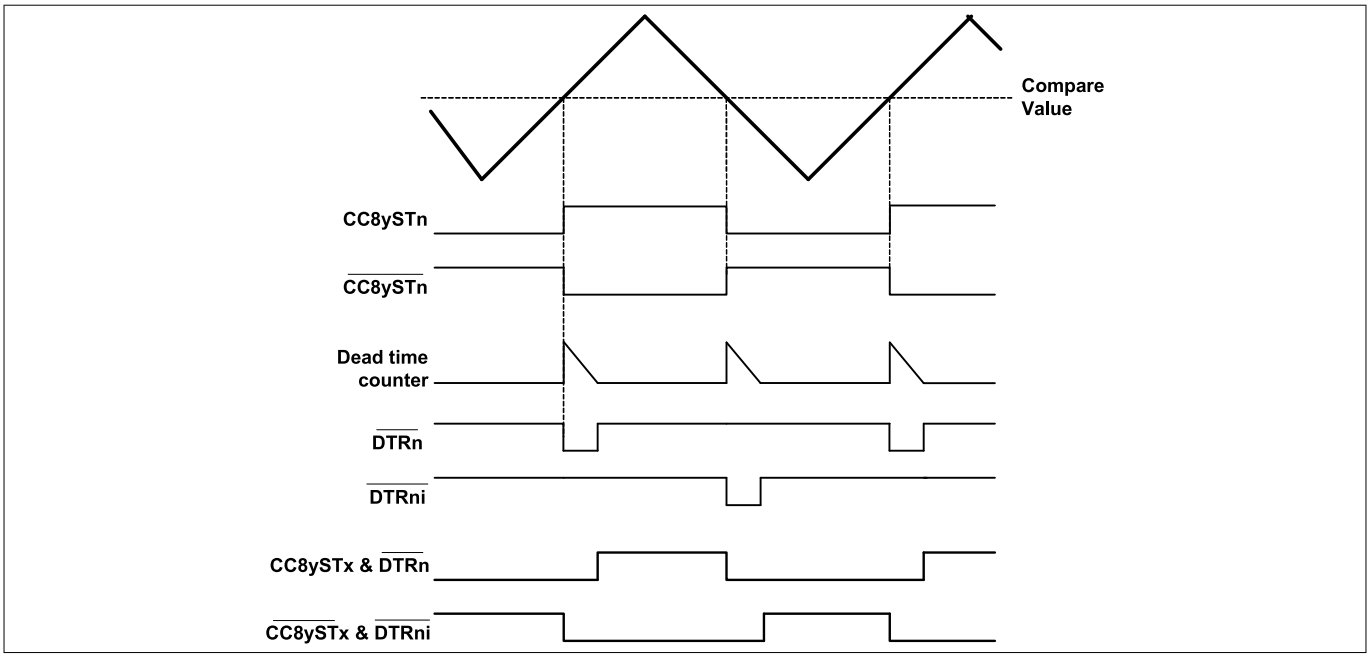


Figure 315 Center aligned - Independent channel with dead time

Asymmetrical Center Aligned Mode

The asymmetric mode is enabled in center aligned by setting the field **CC8yCHC**.ASE to 1_B.

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 316.

The dead time rise and fall times are controlled by the values programmed into the fields, **CC8yDC1R**.DT1R and **CC8yDC1R**.DT1F, respectively.

Figure 317 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.

In this mode the outputs linked with the compare channel 2 are running and available at the outputs - they behave like a non asymmetrical compare channel.

***Note**: When an external signal is used to control the counting direction, the asymmetric mode cannot be enabled.*

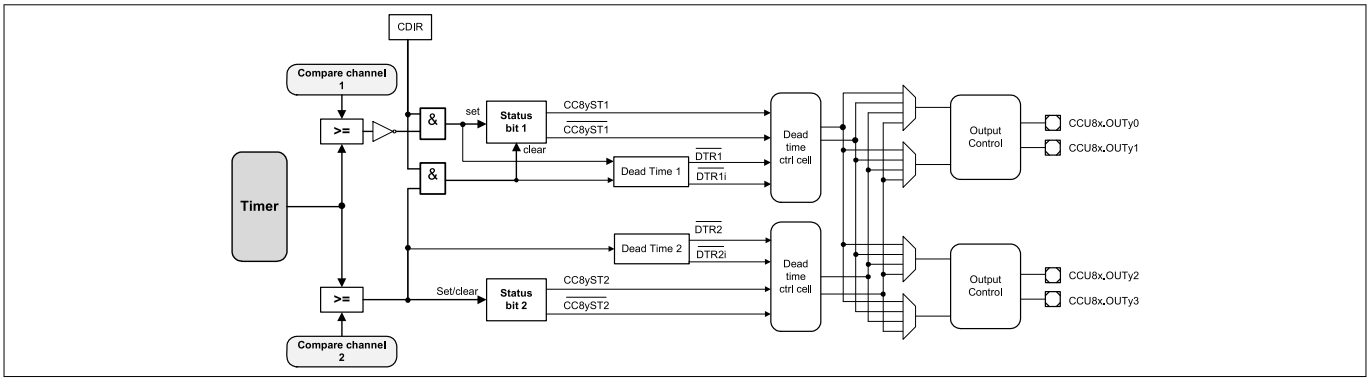


Figure 316 Center Aligned Asymmetric mode scheme

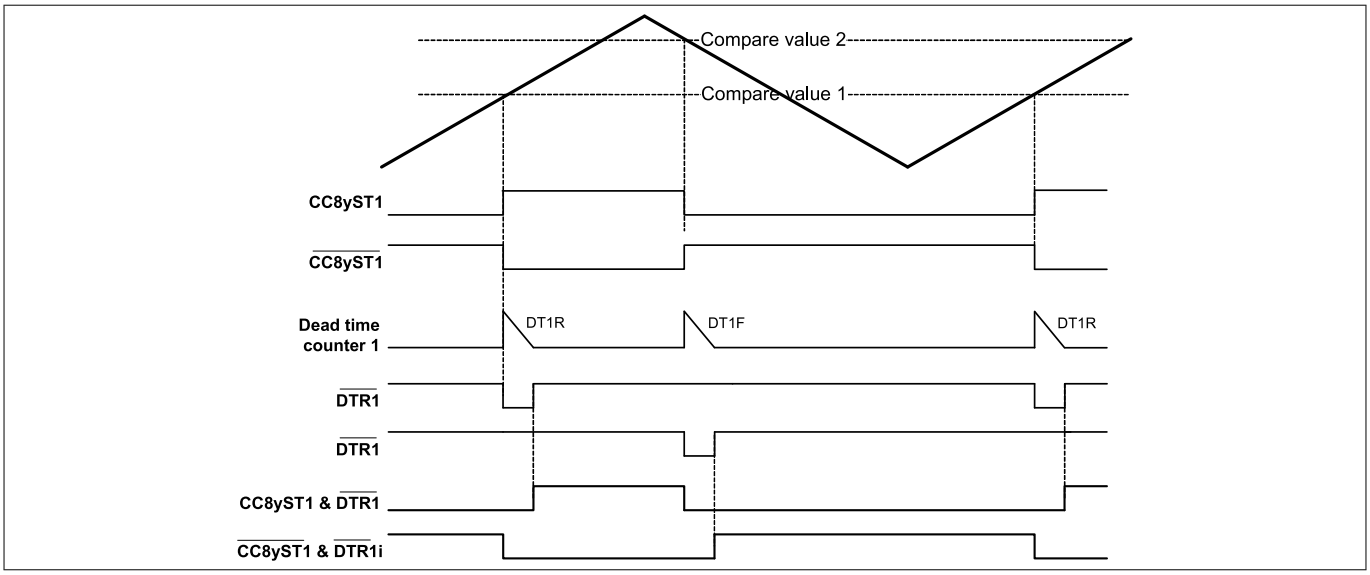


Figure 317 Asymmetric Center aligned mode with dead time

22.2.4.8 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:

$$T_{\text{per}} = \ < \text{Period-Value} > \ + 1; \ \text{in} \ f_{\text{tclk}} \tag{58}$$

In Center Aligned Mode, the timer period is:

$$T_{\text{per}} = \left(< \text{Period-Value} > \ + 1 \right) \times 2; \ \text{in} \ f_{\text{tclk}} \tag{59}$$

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:

$$\text{DC} = 1 - \ < \text{Compare-Value} > /(< \text{Period-Value} > \ + 1) \tag{60}$$

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, Updating the Period, Duty Cycle and other PWM conditions

22.2.4.9 Updating the Period, Duty Cycle and other PWM conditions

Every CCU8 timer slice contains several registers that can be updated on-the-fly, via software, with the intent of modifying certain pwm generation conditions. The most common parameters that need to be updated on-the-fly are normally the period and compare values - that will directly control the duty cycle and switching frequency.

Besides these parameters, each timer slice also permits the user to update on-the-fly the floating prescaler, dither and even the passivel level of the PWM signal. The following text descriptions, will give an overview of the registers/parameters that can be updated on-the-fly, and also the options available to control this update.

Shadow Registers Overview

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 Clock Prescaler and PWM Dithering for a complete description of these functions).

The structure of the shadow registers can be seen in Figure 318.

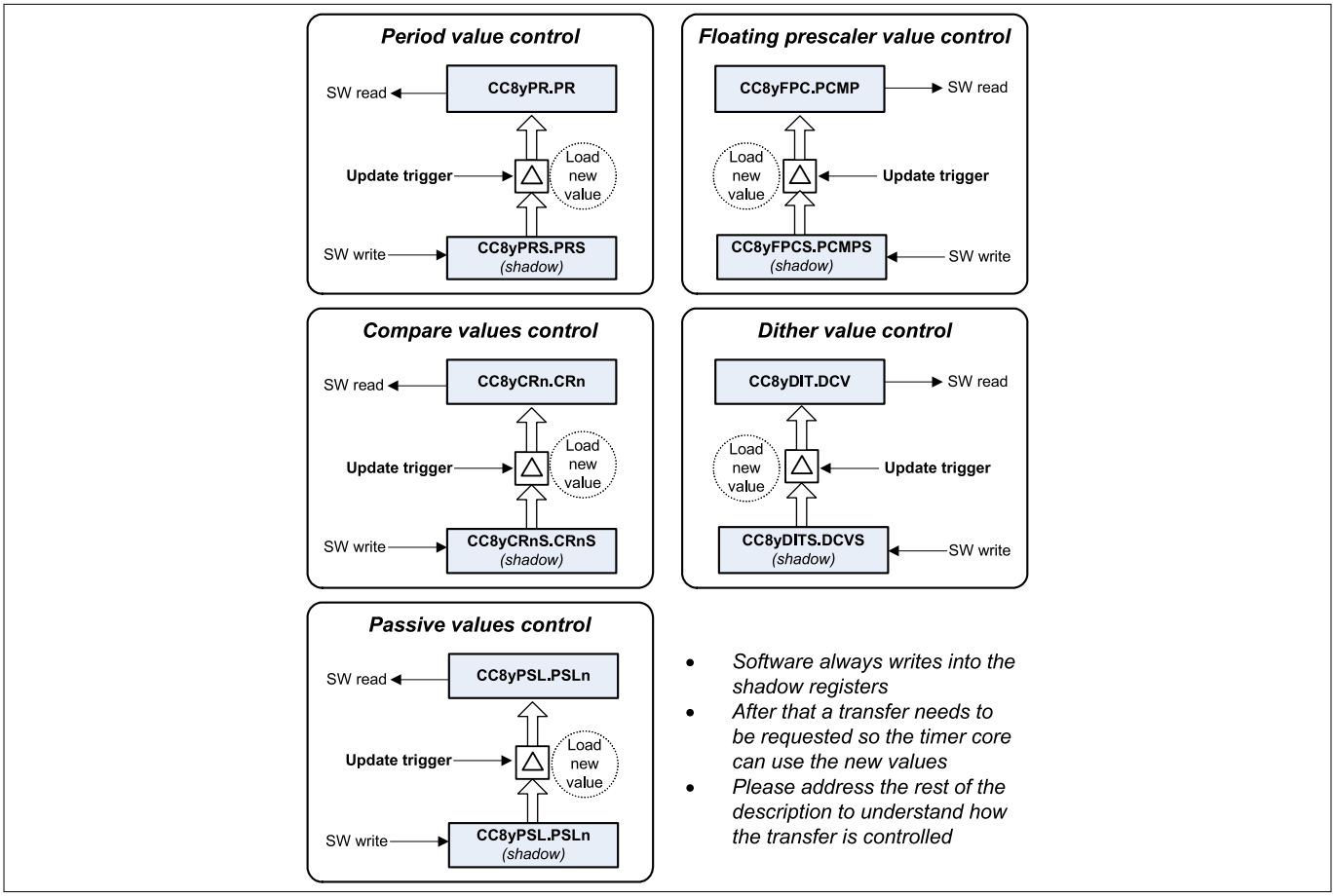


Figure 318 Shadow registers overview

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 - Figure 319.

Each group of shadow registers have an individual shadow transfer enable bit. The software must set this enable bit to 1B, whenever an update of the values is needed. These bits are automatically cleared by the hardware, when the update is done. 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).

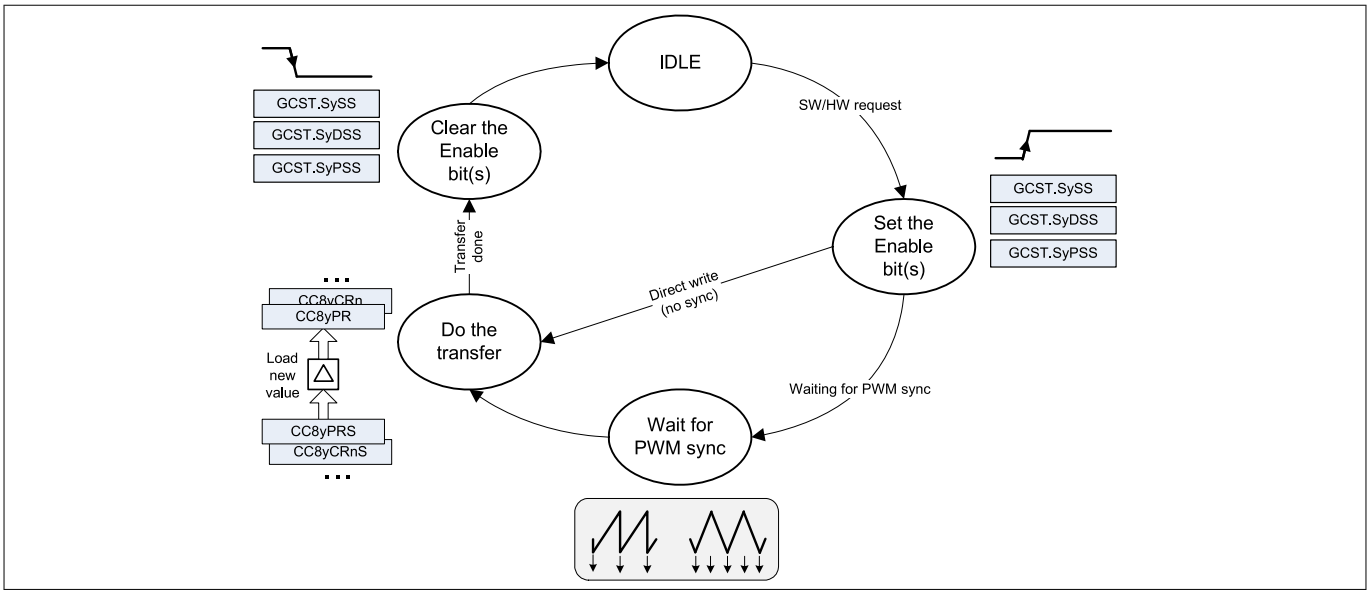


Figure 319 Shadow transfer state machine

The structure of the shadow transfer logic is depicted in Figure 320. In this figure it can be seen the logic structure associated with each of the stages of the state machine - Figure 319:

- In the request sources one can see that the shadow transfer can be requested by:
  - **software** by writing a 1_B into the specific bitfield in the GCST register
  - **an automatic trigger** that is configured in the **CC8ySTC** register, for example, request an update every time that the period shadow register is modified
  - **a hardware pin** linked to the CCU8x.MCSS input (this is configured via the GCTRL.MSEy and GCTRL.MSDE fields)
- In the control logic is visible that after a shadow transfer request is issued, the associated bitfield is set, indicating that an update is pending
- There is also a stage where the user can configure how the update of the values is done (this is configured also via the **CC8ySTC** register):
  - **synchronously with the PWM** switching cycle
  - done **immediately**

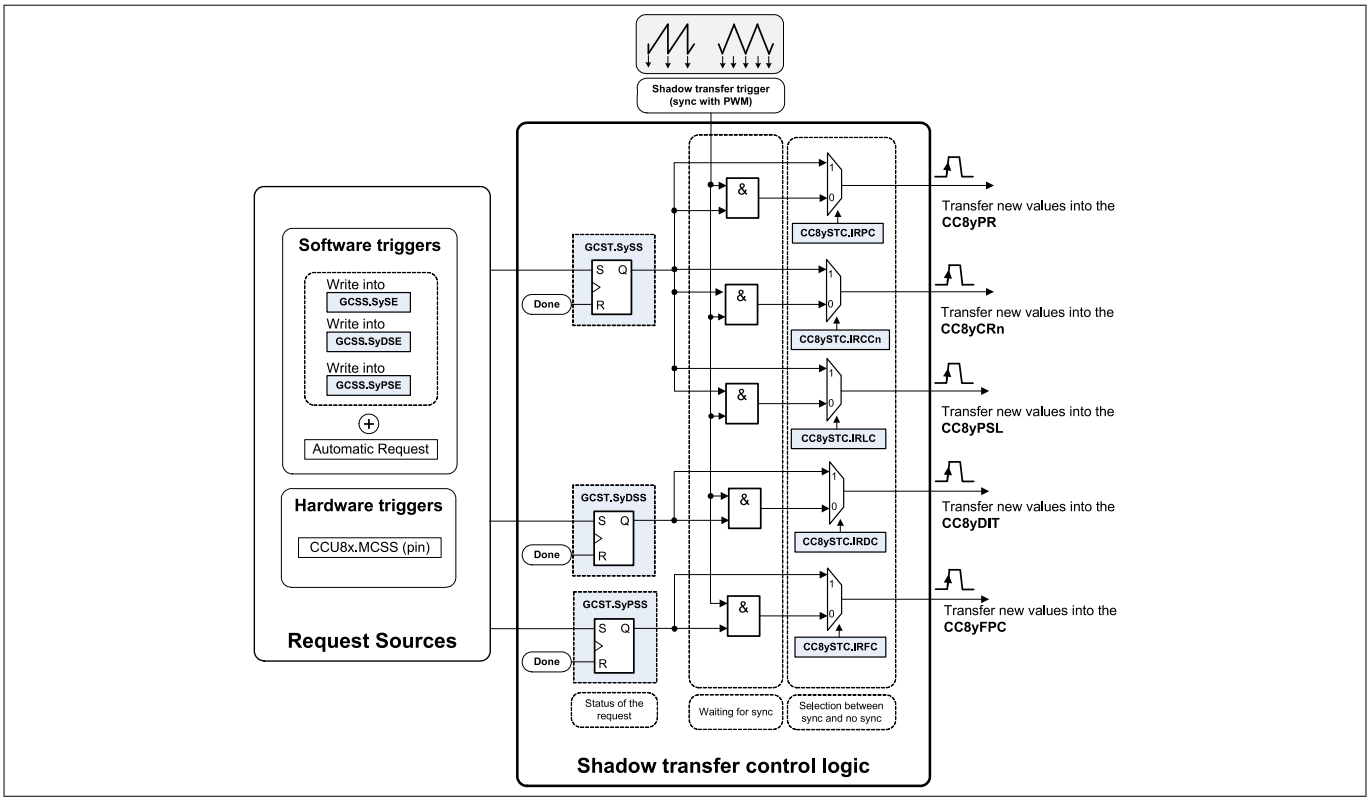


Figure 320 Shadow transfer enable logic

Updating the Shadow Registers

There are two major modes and two sub modes to update the shadow registers:

- Synchronously with the PWM switching cycle - major mode
- Immediate after an update request has been issued - major mode
- Automatic update request - sub mode
- Cascaded shadow transfer - sub mode

The following text will focus and describe each of these modes by the same order mentioned above. It is understood that the a sub mode can be used with any of the major modes.

- Synchronously with the PWM switching cycle

When the update of the registers is done synchronously with the PWM switching cycle, 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 - center aligned
- in the next clock cycle after an One Match while counting down - center aligned and edge aligned
- immediately,if the timer is stopped and the shadow transfer enable bit(s)is set - center aligned and edge aligned.

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 321 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 Edge Aligned Mode and Center Aligned Mode.

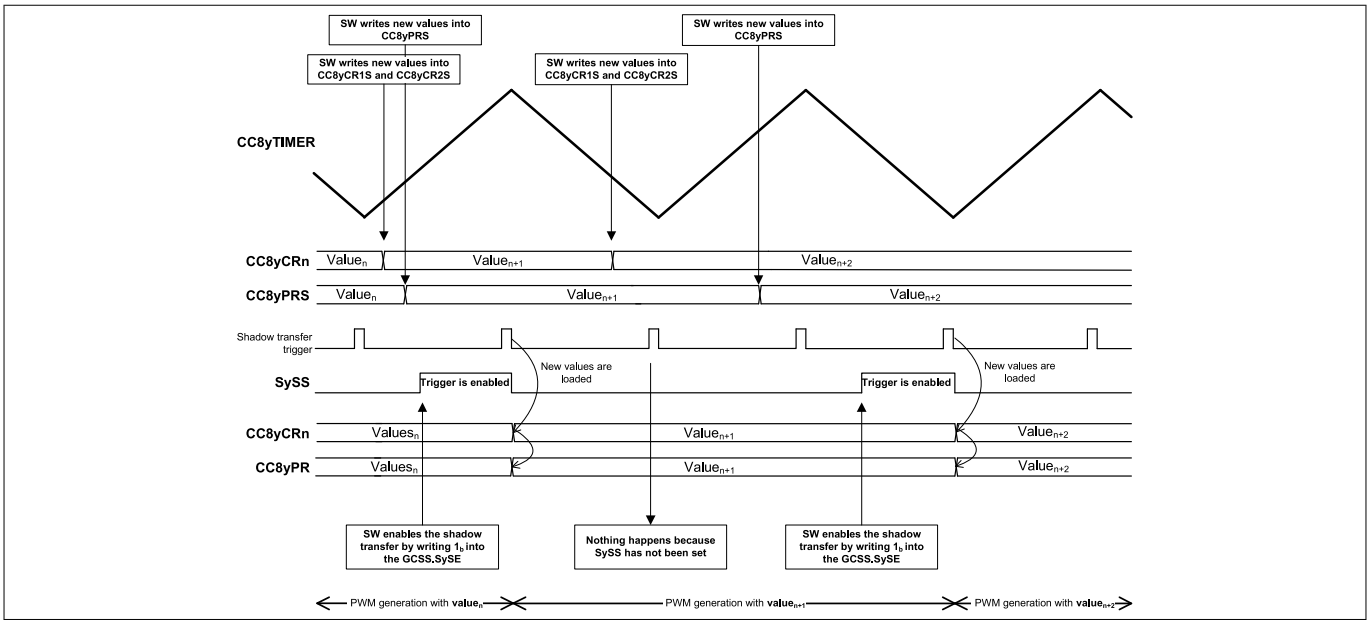


Figure 321 Shadow transfer timing example - center aligned mode with sync

- Immediate after an update request has been issued

In some applications it may be necessary to update the PWM conditions without waiting for a new switching cycle. When this is required, the user can configure via the **CC8ySTC** register, which fields/registers should be updated without waiting for a PWM synchronization (individual configuration fields exist for each parameter, for example, period, compare, etc).

In Figure 322 an example is depicted, of how the update of the period and compare registers can be done, without waiting for the normal switching cycle synchronization - **CC8ySTC**.IRPC and **CC8ySTC**.IRCCn fields are both configured with 1_B.

The software writes new values into the period and compare shadow registers (CC8yPRS and CC8yCRS respectively). After doing that, the software requests an update of the current values being used by the timer kernel - by writing 1_B into the GCSS.SySE bitfield. Due to the fact that the software has configured the CC8ySTC register to perform an immediate update, after the GCSS.SySE is written with 1_B, the hardware will automatically load and use the new values.

The request bit - SySS - is then cleared when the transfer is done. One should notice that all the depicted **CC8ySTC** configuration fields need to be set to 1_B for the request bit (SySS) to be cleared immediately. If one of these **CC8ySTC** configuration bitfields is configured with 0B, then the associated value is updated coherently with the PWM signal - Figure 323.

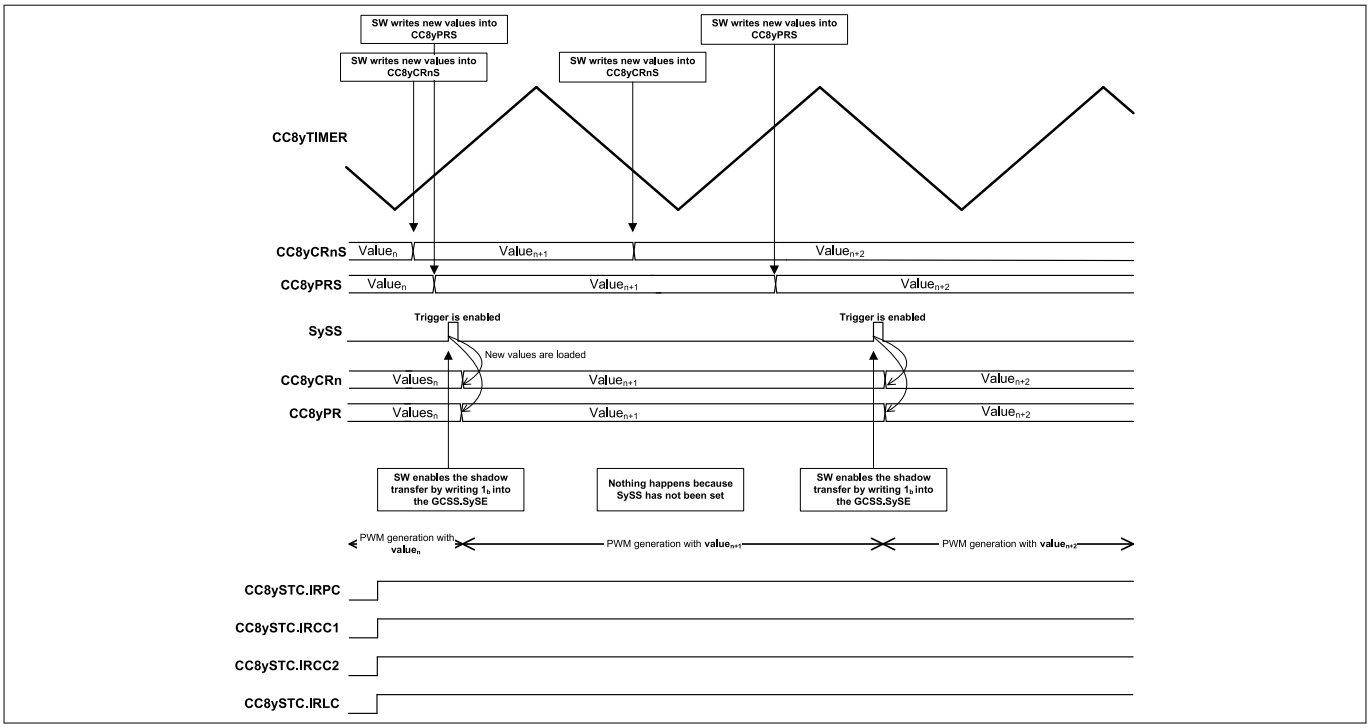


Figure 322 Shadow transfer timing example - center aligned mode without sync

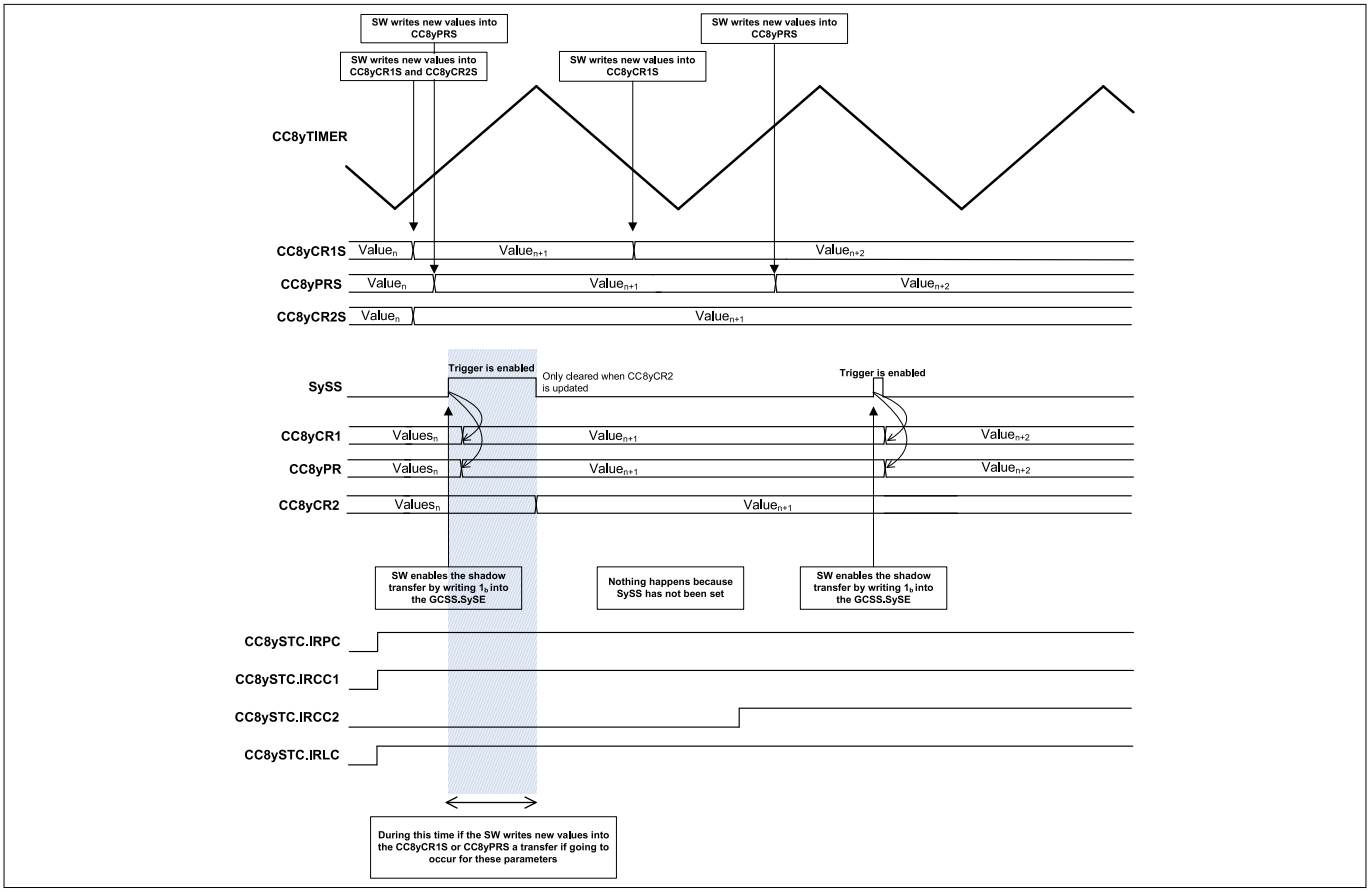


Figure 323 Shadow transfer timing example - dual configuration

- Automatic update request

Each timer slice offer also a possibility of enabling an automatic transfer request. This is configured in the CC8ySTC register (via the ASPC, ASCC1, ASCC2, ASLC, ASDC and ASFC fields). Enabling an automatic transfer request, signifies that after the software writes a value into a specific shadow register, the update of the current value is going to be automatically enabled by hardware. This means for example, that the software does not need to write a 1_B to the GCSS.SySE field after a new period value is loaded into the shadow register.

The operation of this sub mode can be seen in Figure 324 - initially the software had configured the timer slice with an automatic request enable uppon the update of the period shadow register (ASPC is set to 1_B). One can see that after the update of the CC8yPRS (period shadow register) the SySS bitfield is automatically set by hardware, enabling a shadow transfer (in the second time slot the field ASCCn is set to 1_B).

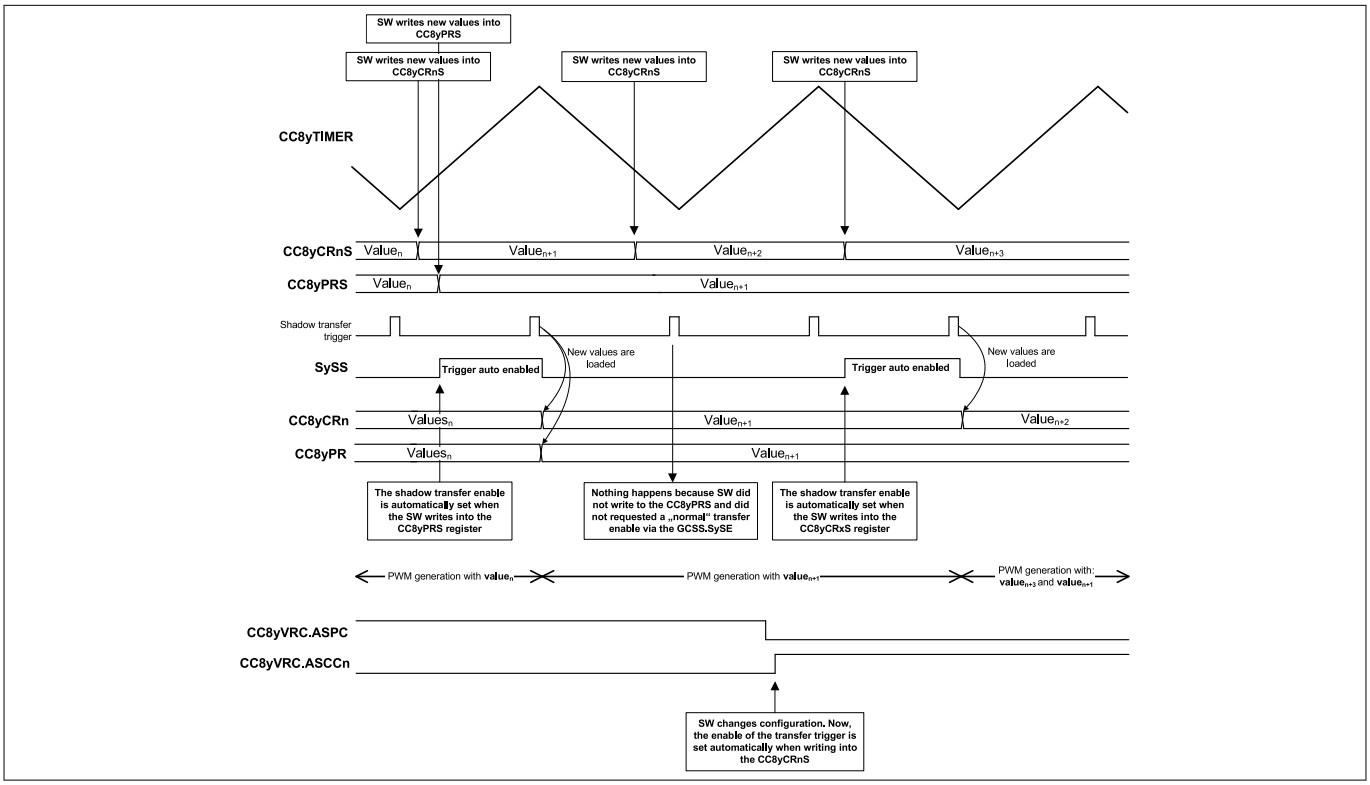


Figure 324 Shadow transfer timing example - center aligned mode with automatic request

- 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 325.

To enable the cascaded shadow transfer function, the bitfield **CC8ySTC**.CSE of the specific timer slice needs to be set to 1_B.

The shadow transfer enable bits, still need to be set via SW for each of the individual slices, Figure 326.

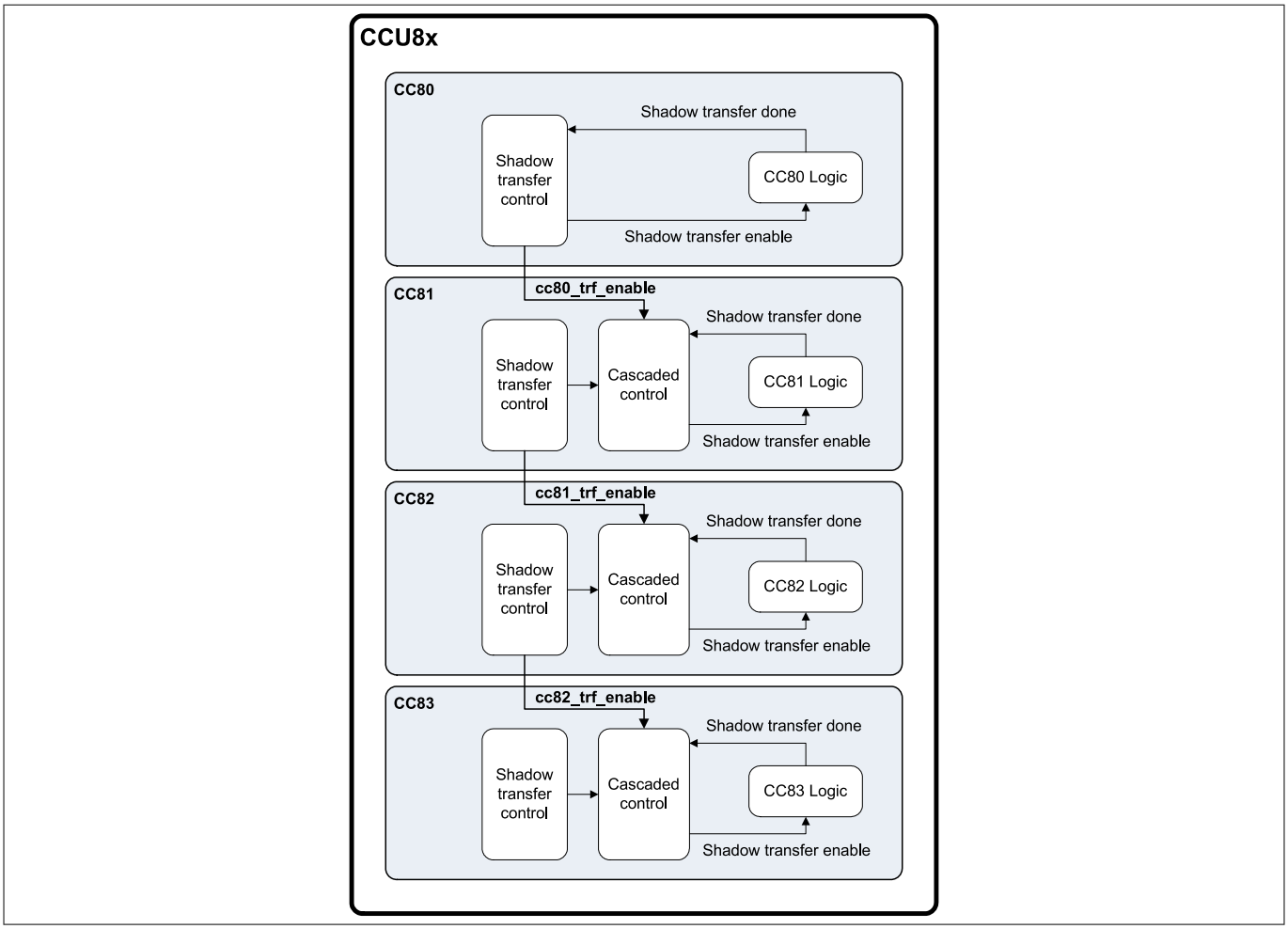


Figure 325 Cascaded shadow transfer linking

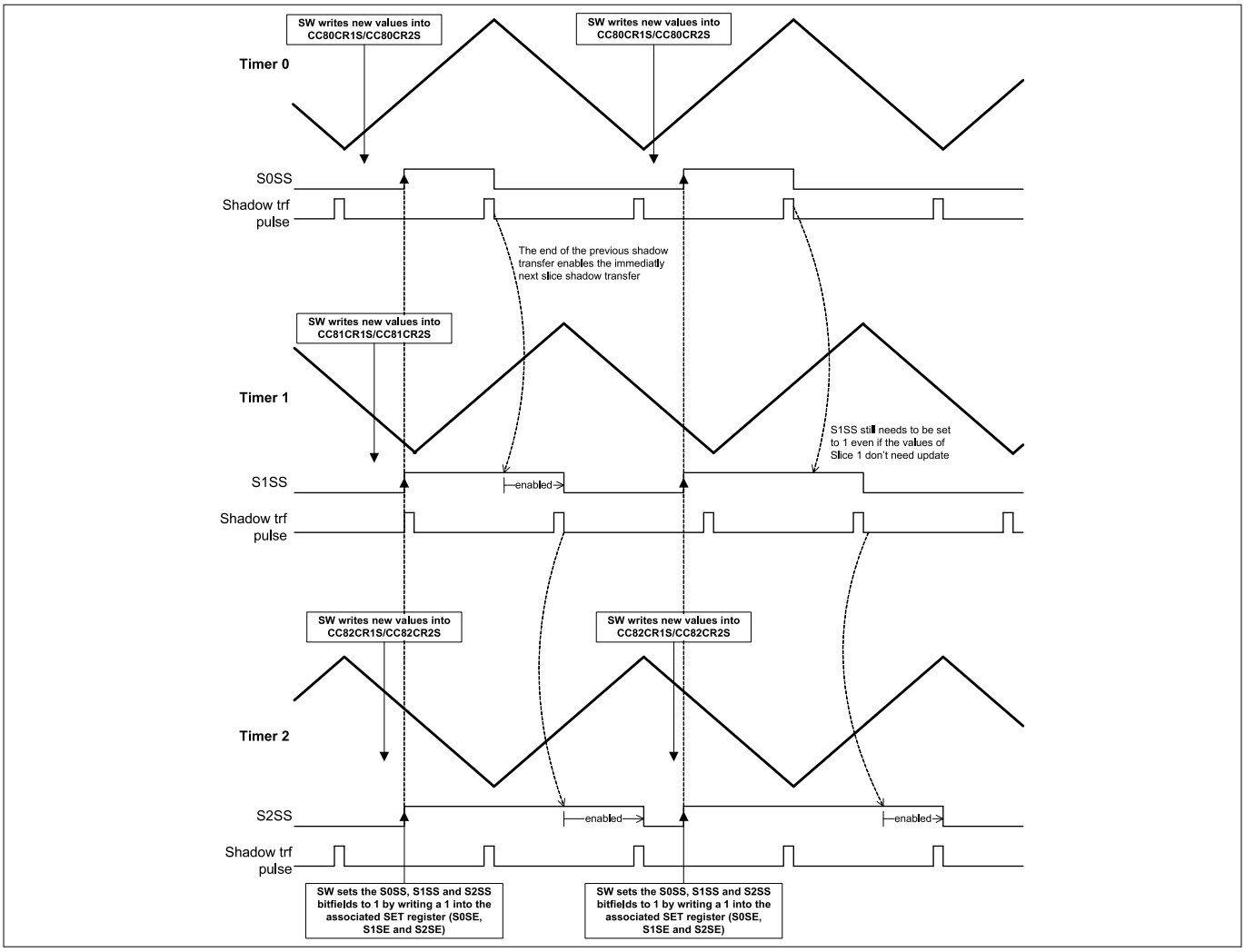


Figure 326 Cascade shadow transfer timing

***Note**: The shadow transfer enable bits - SySS, 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.*

22.2.4.10 PWM Active/Passive Rules

Like previously mentioned on Counting Modes Introduction, each CCU8 timer slice has two status bits (CC8yST1 and CC8yST2) that contain the current state of the comparison logic of the two compare channels, that are then used to generate the output PWM signals.

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, Status Bit Override.

The software at any time can write a 1_B into the **GCSS**.SySTnS bitfield, which will set the status bit **GCST**.CC8ySTn of the specific timer slice. By writing a 1_B into the **GCSC**.SySTnC bitfield, the software imposes a clear of the specific status bit.

22.2.4.11 Output PWM Path

Each Timer Slice contains an output path, where the PWM output signal can be conditioned after the PWM status bit - CC8ySTn - Figure 327. Inside the output path, the polarity of the PWM signal can be inverted in relation to the status bit (including dead time).

The source of each output can also be selected. Each CC8y has 4 outputs that in the default configuration are set to:

- CCU8x.OUTy0 - connected/controlled by Status Bit 1 (CC8yST1)
- CCU8x.OUTy1 - connected/controlled by inverted Status Bit 1 (not(CC8yST1))
- CCU8x.OUTy2 - connected/controlled by Status Bit 2 (CC8yST2)
- CCU8x.OUTy3 - connected/controlled by inverted Status Bit 2 (not(CC8yST2))

To configure a different combination of outputs the **CC8yCHC** register should be addressed.

There are also three different types of PWM outputs:

- CCU8x.OUTy[0...3] - PWM signal from Timer Slice y that can be conditioning/inverted and contains dead time insertion.
- CCU8x.STyA and CCU8x.STyB - PWM signals from Timer Slice y that mirrors the specific PWM status bit (these signals do not contain dead time insertion)
- CCU8x.STy - PWM signal that can be configured to carry the CC8yST1 or CC8yST2 status information or to perform a logical operation between the two status bits - AND or OR gate.

The conditioning of the PWM signals can be done via several sources. This conditioning implies that even if the specific status bit is active, the selected output signal may be inactive. There is also a dedicated configuration bitfield that allows the user to select the polarity of each output pwm signal - **CC8yPSL**.

The conditioning of the output path has three sources:

- External Modulation - conditioning controlled by an external signal
- Multi-Channel Mode - conditioning controlled by a dedicated timer slice input that normally is used in Brushless DC Motor Control patterns
- TRAP State - highest level of conditioning that is controlled by an external function and is normally linked to a system fault, for example, overcurrent protection

Each of these conditioning sources are explained in detailed in their dedicated section.

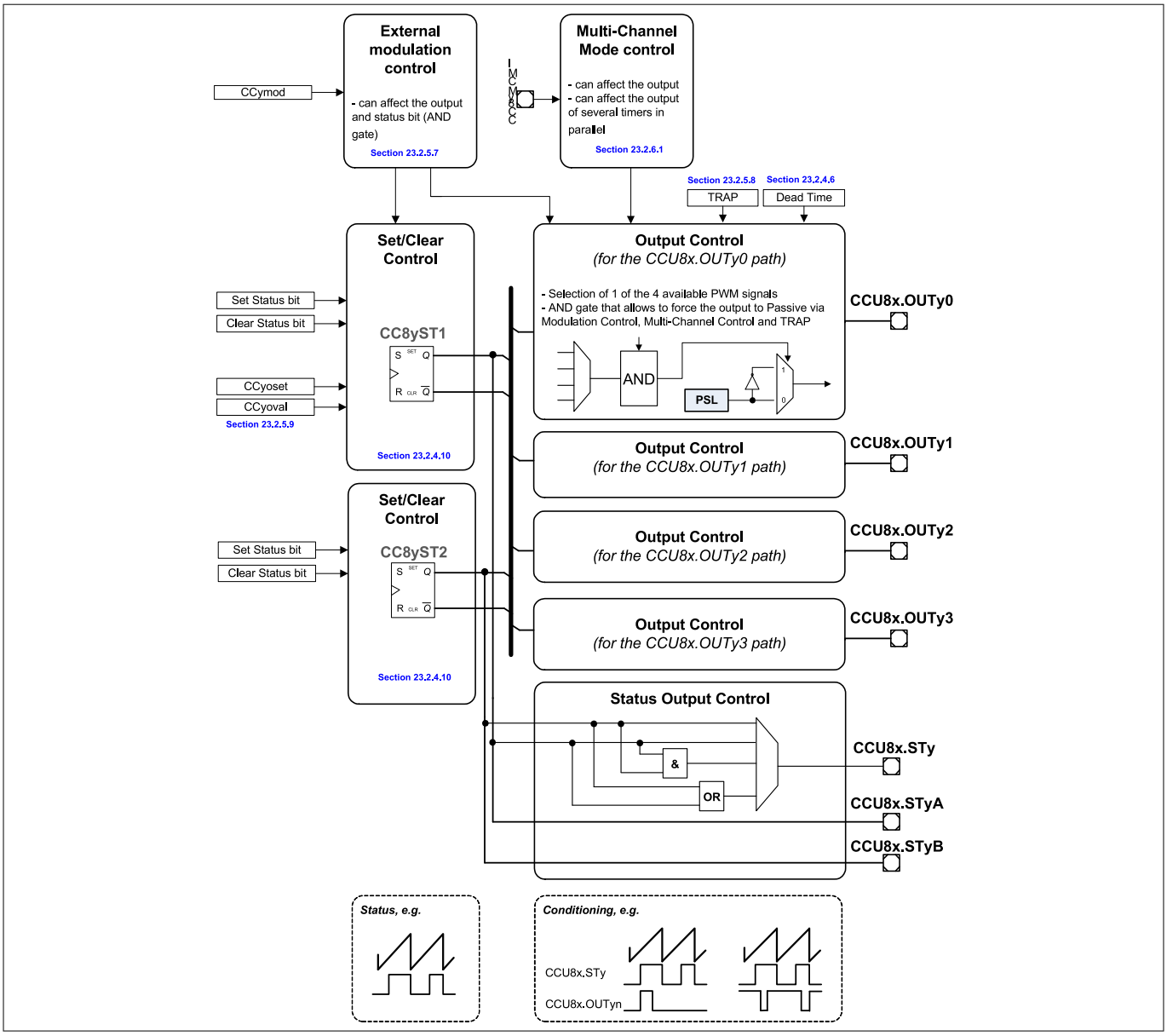


Figure 327 PWM output path

22.2.5 Timer Slice External Functions

Each CCU8 slice has the possibility of using up to three different input events, see Timer Slice Input Selector. These three events can then be mapped to Timer Slice functions (the full set of available functions is described at Timer Slice Connection Matrix).

These events can be mapped to any of the CCU8x.INy[BV:AA] 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 (for example, input X triggers event 0 with rising edge and triggers event 1 with the falling edge).

22.2.5.1 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 **CC8yINS1**.EVxIS register and indicating the active edge of the signal on the **CC8yINS2**.EVxEM register.

This event should be then mapped to the start or stop functionality by setting the **CC8yCMC**.STRTS (for the start) or the **CC8yCMC**.ENDS (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 **CC8yINS2**.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 328 to Figure 331 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.

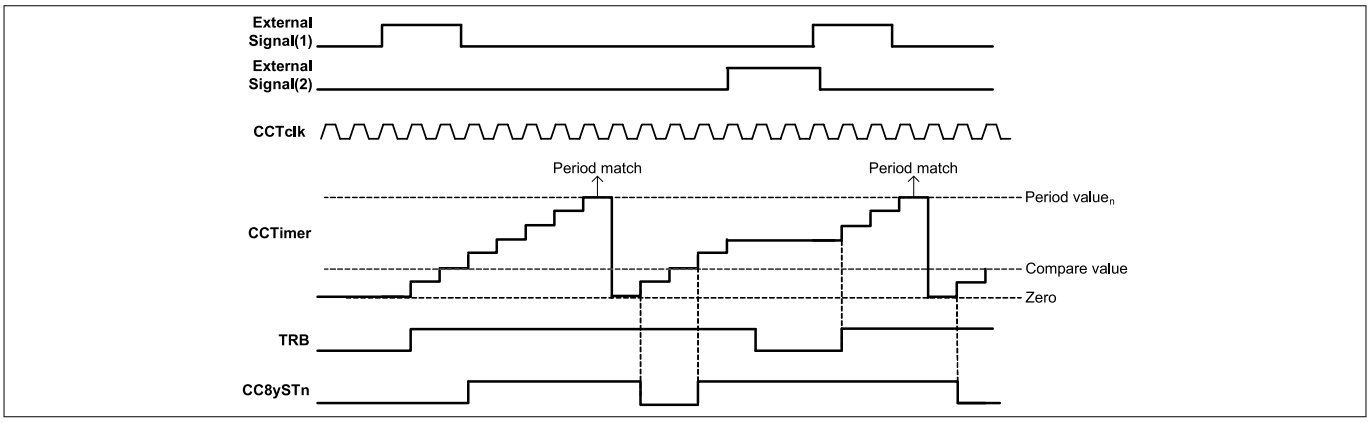


Figure 328 Start (as start)/ stop (as stop) - CC8yTC.STRM = 0_B, CC8yTC.ENDM = 00_B

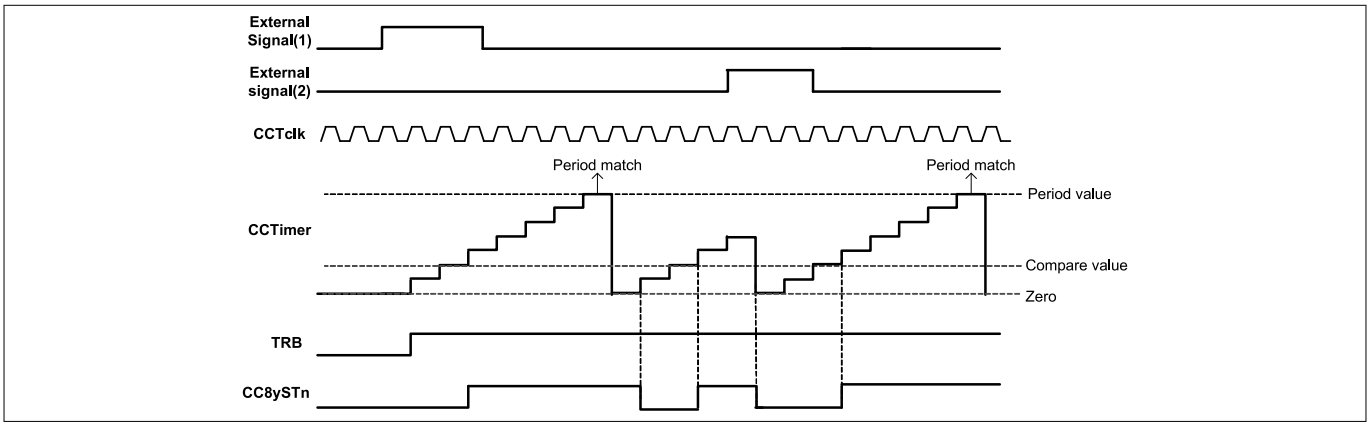


Figure 329 Start (as start)/ stop (as flush) - CC8yTC.STRM = 0_B, CC8yTC.ENDM = 01_B

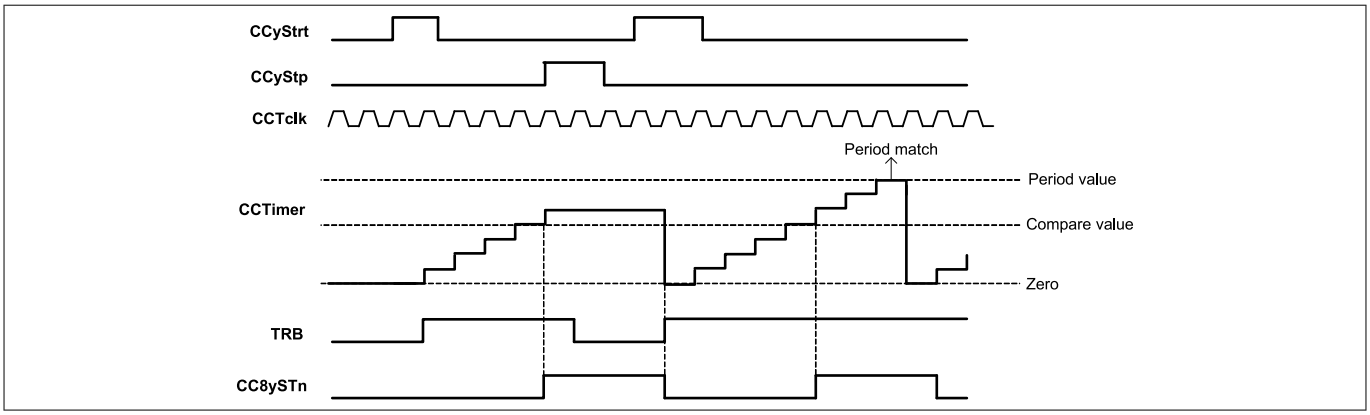


Figure 330 Start (as flush and start)/ stop (as stop) - CC8yTC.STRM = 1_B, CC8yTC.ENDM = 00_B

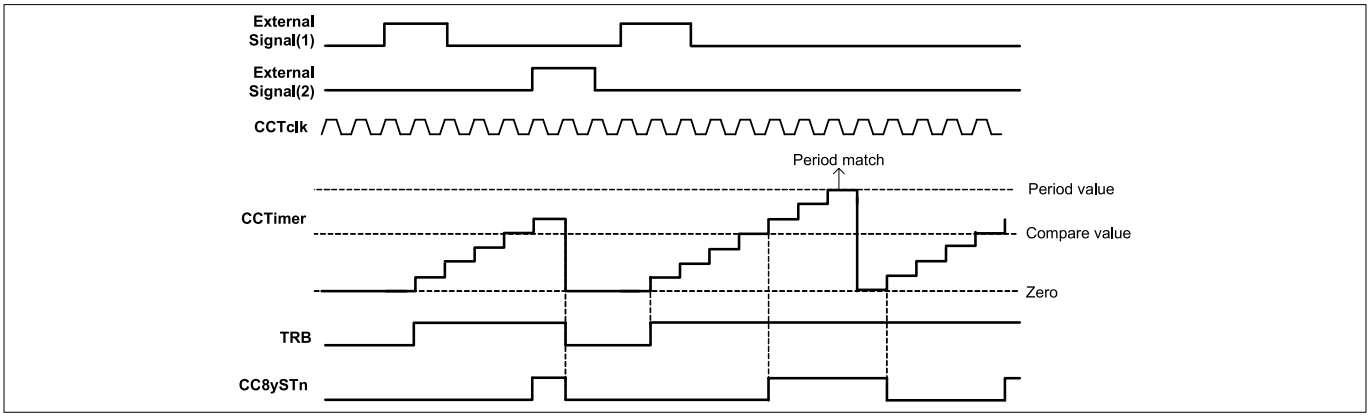


Figure 331 Start (as start)/ stop (as flush and stop) - CC8yTC.STRM = 0_B, CC8yTC.ENDM = 10_B

22.2.5.2 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 **CC8yINS1**.EVxIS register and indicating the active level of the signal on the **CC8yINS2**.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 **CC8yINS2**.EVxLM.

The status bit of the slice (CCSTn) is always set when the timer value is equal or greater than the value stored in the compare register, see PWM Active/Passive Rules.

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. Using an external signal to perform the up/down counting function and configuring the event as active LOW means that the timer is counting up when the signal is HIGH and counting down when LOW (this is to match the CDIR value).

Figure 332 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 **CC8yINS2**.EVxLM = 0_B. When the operation is switched, then the user should set **CC8yINS2**.EVxLM = 1_B.*

***Note**: Using an external counting direction control, sets the slice in edge aligned mode.*

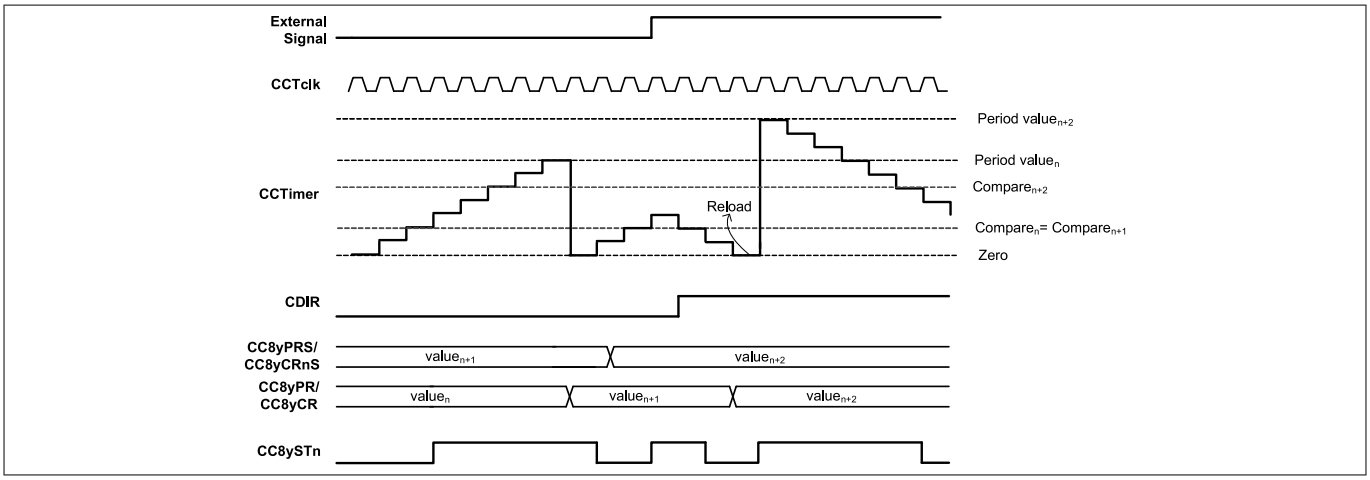


Figure 332 External counting direction

22.2.5.3 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 **CC8yINS1**.EVxIS register and indicating the active level of the signal on the **CC8yINS2**.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 **CC8yINS2**.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 333 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.

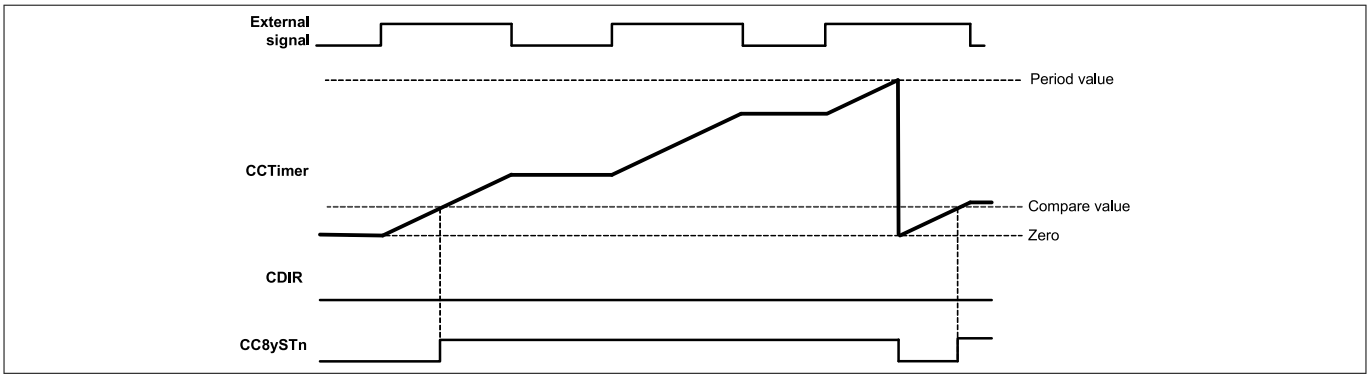


Figure 333 External gating

22.2.5.4 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 **CC8yINS1**.EVxIS register and indicating the active edge of the signal on the **CC8yINS2**.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 **CC8yINS2**.EVxEM.

One can select the rising, falling or both edges to perform a count. On Figure 334, 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.

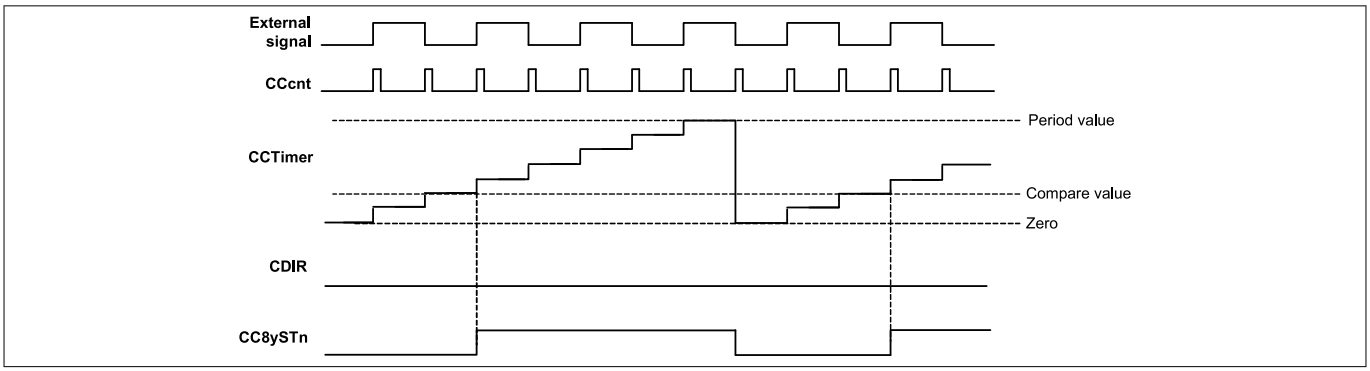


Figure 334 External count

22.2.5.5 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 335.

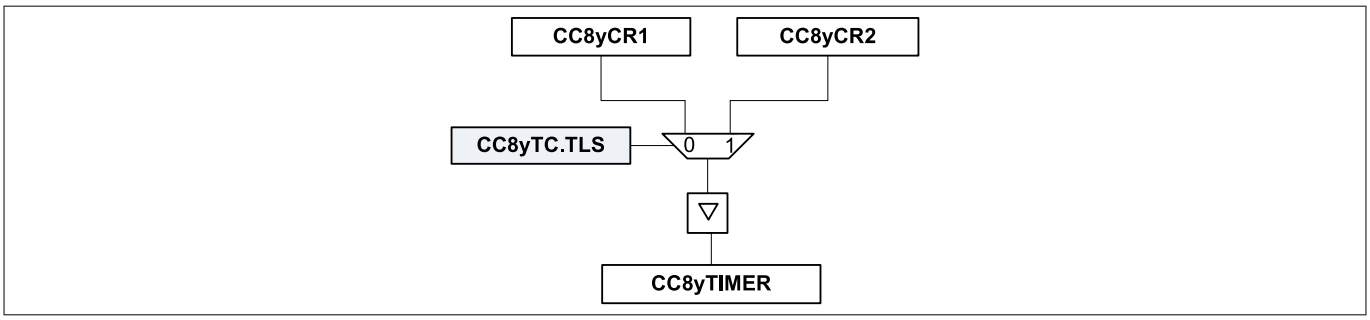


Figure 335 Timer load selection

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 **CC8yINS1**.EVxIS register and indicating the active edge of the signal on the **CC8yINS2**.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 **CC8yINS2**.EVxEM.

On figure Figure 336, 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.

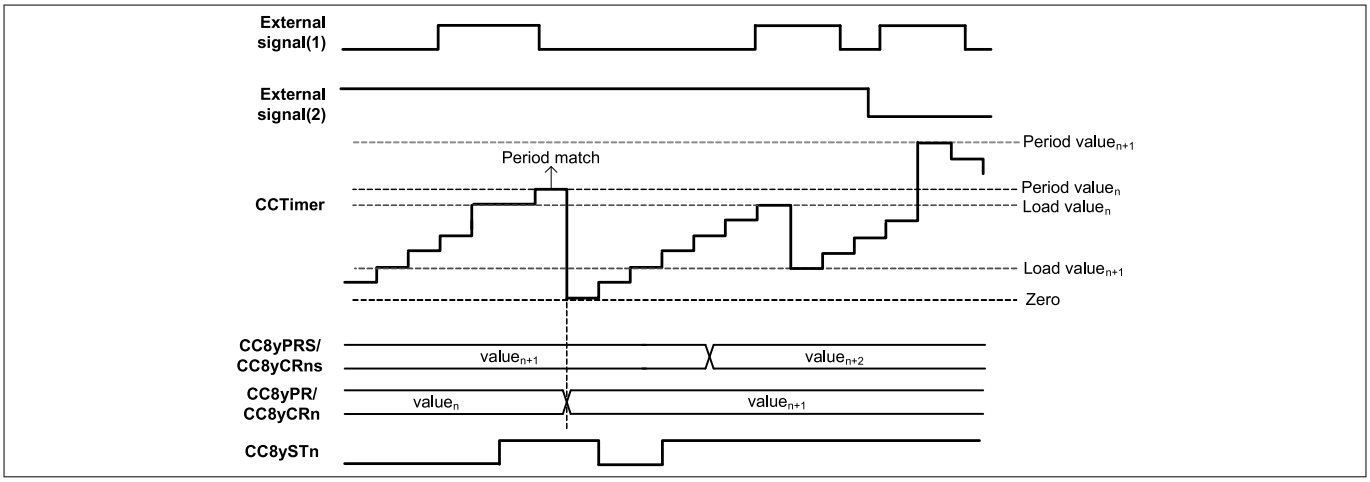


Figure 336 External load

22.2.5.6 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 339: 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 **CC8yINS1**.EVxIS register and indicating the active edge of the signal on the **CC8yINS2**.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 **CC8yINS2**.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 (by reading the specific capture register or by reading the extended capture read value, see Capture Extended Read Back Mode).

The capture of a new value into a specific capture registers is dictated by the status of the full flag as follows:

$$CC8yC1V_{\text{capt}} = \text{NOT}\left(CC8yC1V_{\text{full_flag}} \text{ AND } CC8yC0V_{\text{full_flag}}\right) \quad (61)$$

$$CC8yC0V_{\text{capt}} = CC8yC1V_{\text{full_flag}} \text{ AND NOT}\left(CC8yC0V_{\text{full_flag}}\right) \quad (62)$$

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 337, an external signal was selected as an event for capturing the timer value into the **CC8yC0V**/**CC8yC1V** registers. The status bit, CC8ySTn, during capture mode is asserted whenever a capture trigger is detected and deasserted when the counter matches 0000_H.

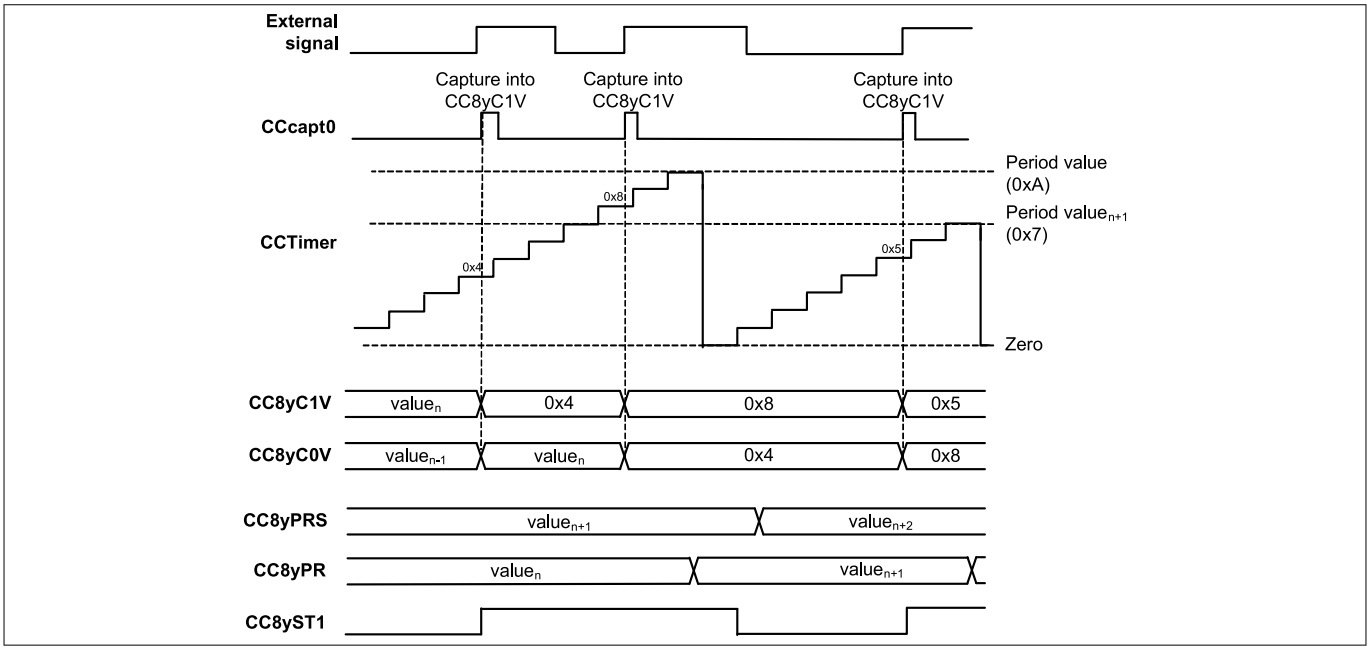


Figure 337 External capture - CC8yCMC.CAP0S != 00_B, CC8yCMC.CAP1S = 00_B

On Figure 338, 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 as the source for the channel 2 capture trigger, but as opposite to the external signal(1), the active edge was set as falling.

See Capture Mode Usage for a capture mode usage description.

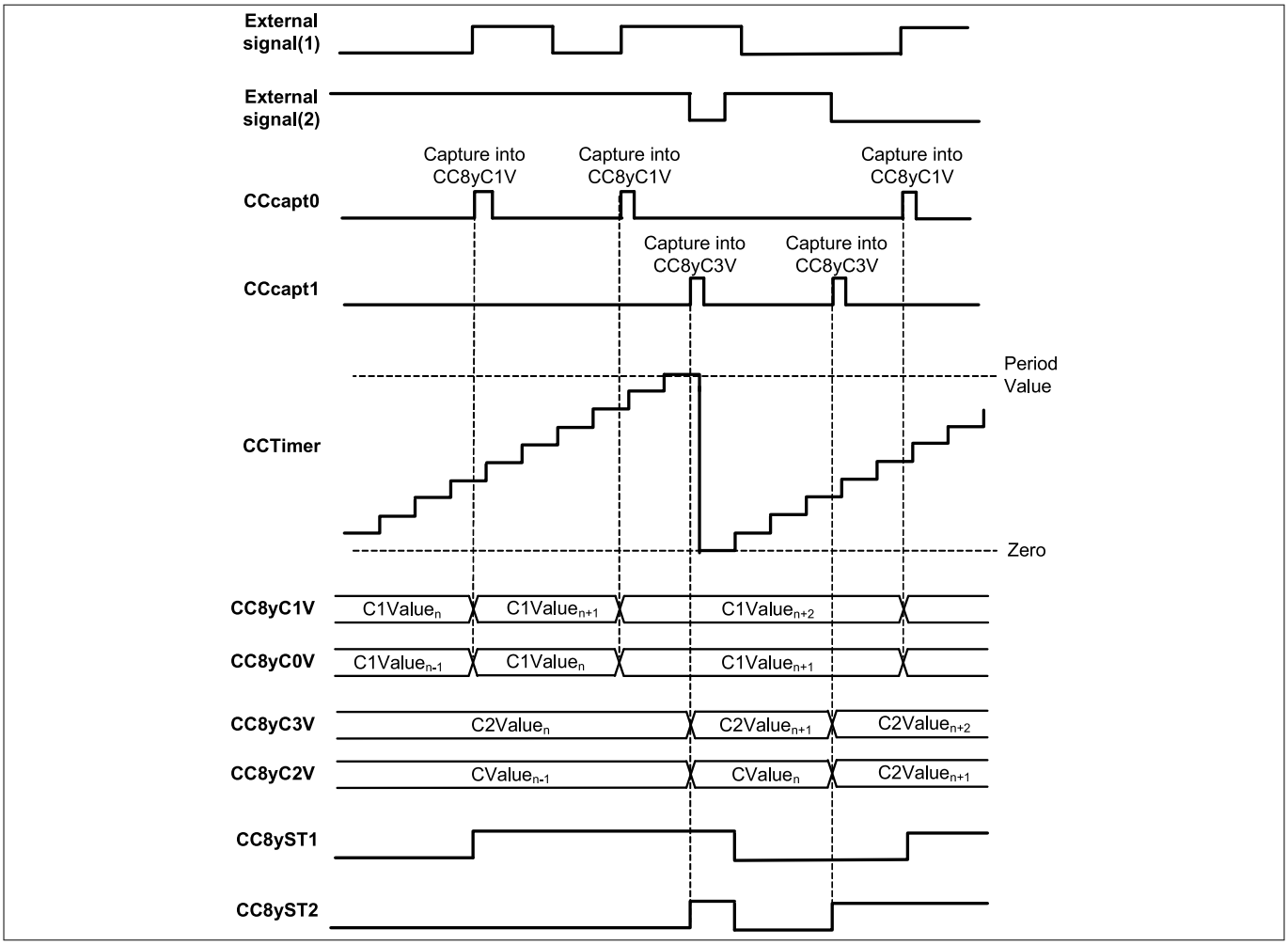


Figure 338 External capture - CC8yCMC.CAP0S != 00_B, CC8yCMC.CAP1S != 00_B

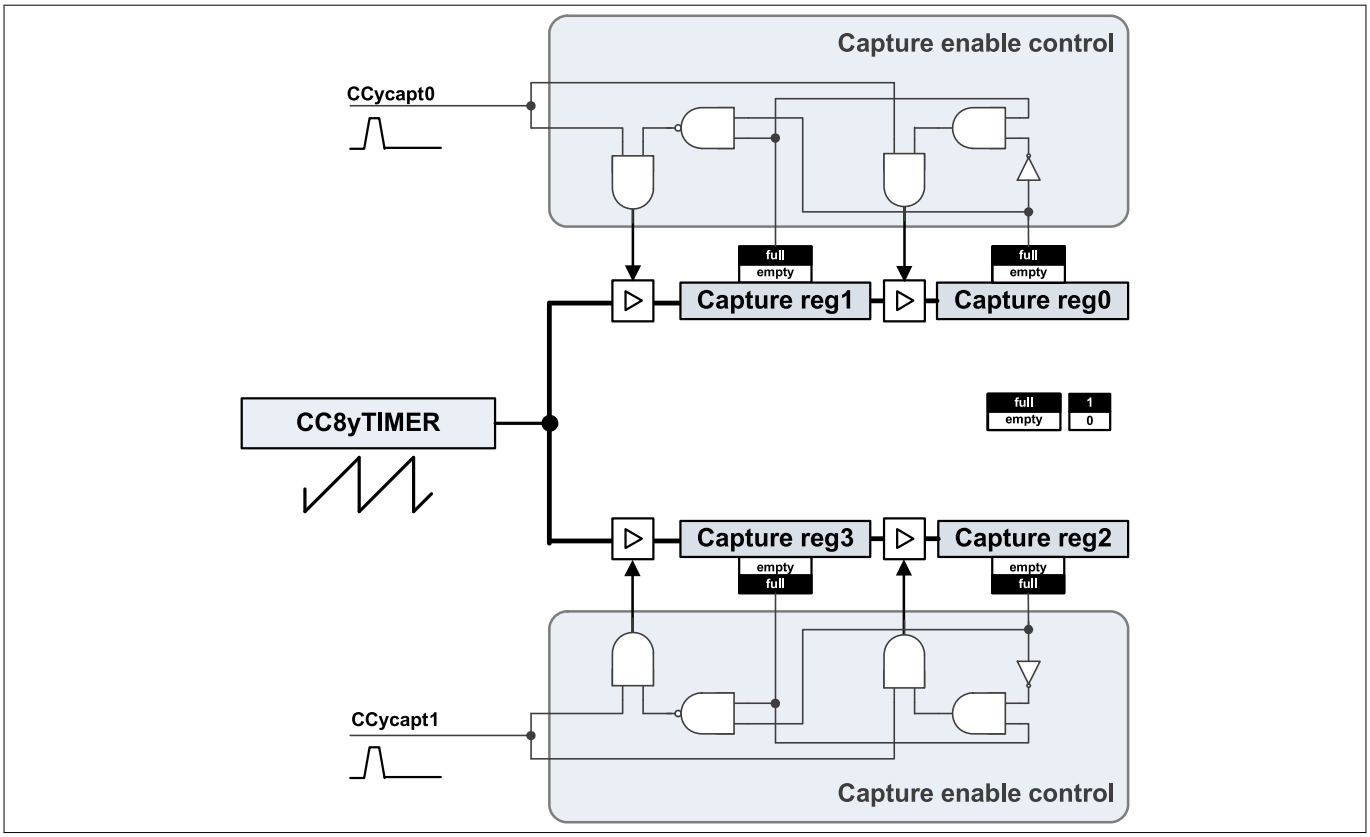


Figure 339 Slice capture logic

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 341.The functionality that controls the capture is the CCcapt1.

The capture logic follows the same structure shown in Figure 339 but extended to a four register chain, see Figure 340.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):

$$CC8yC3V_{\text{capt}} = \text{NOT} \left(CC8yC3V_{\text{full_flag}} \text{ AND } CC8yC2V_{\text{full_flag}} \text{ AND } CC8yC2V_{\text{full_flag}} \text{ AND } CC8yC1V_{\text{full_flag}}\right) \quad (63)$$

$$CC8yC2V_{\text{capt}} = CC8yC3V_{\text{full_flag}} \text{ AND NOT} \left(CC8yC2V_{\text{full_flag}} \text{ AND } CC8yC1V_{\text{full_flag}} \text{ AND } CC8yC0V_{\text{full_flag}}\right) \quad (64)$$

$$CC8yC1V_{\text{capt}} = CC8yC2V_{\text{full_flag}} \text{ AND NOT} \left(CC8yC1V_{\text{full_flag}} \text{ AND } CC8yC0V_{\text{full_flag}}\right) \quad (65)$$

$$CC8yC0V_{\text{capt}} = CC8yC1V_{\text{full_flag}} \text{ AND NOT} \left(CC8yC0V_{\text{full_flag}}\right) \quad (66)$$

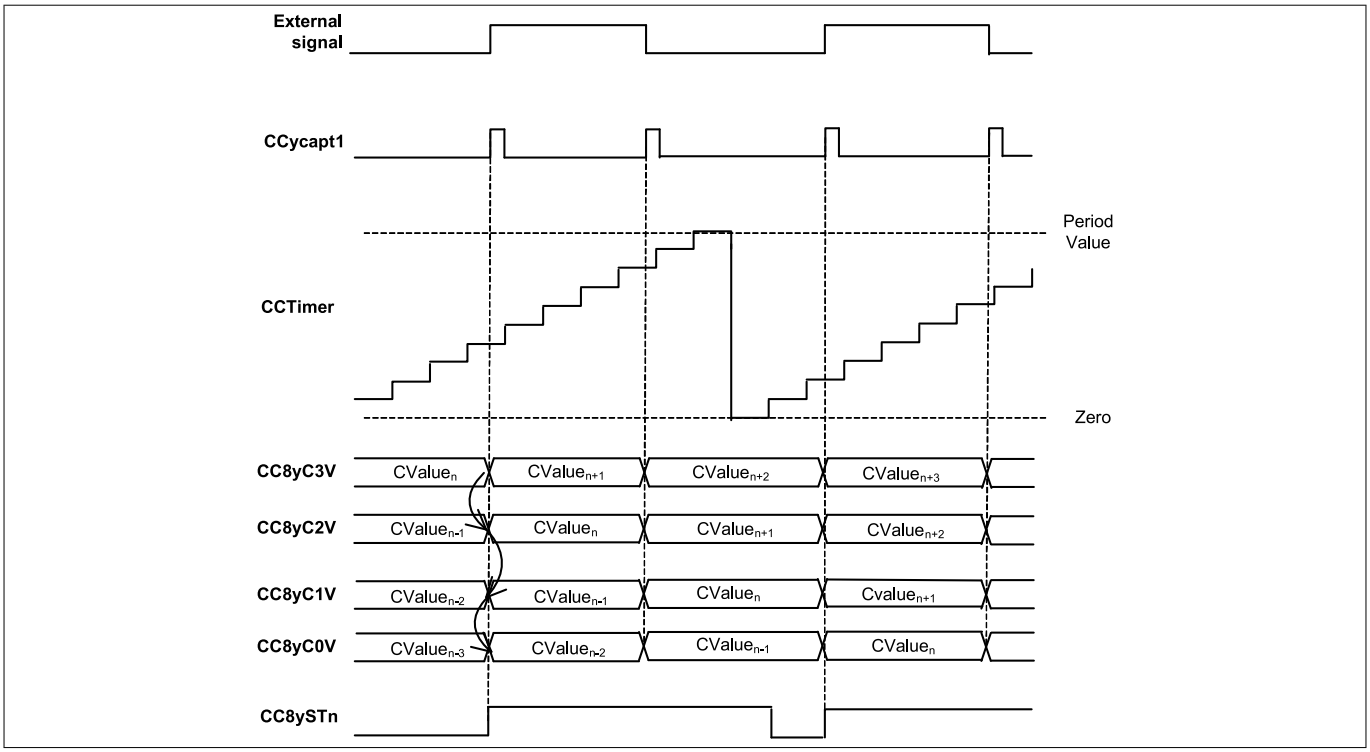


Figure 340 External Capture - CC8yTC.SCE = 1_B

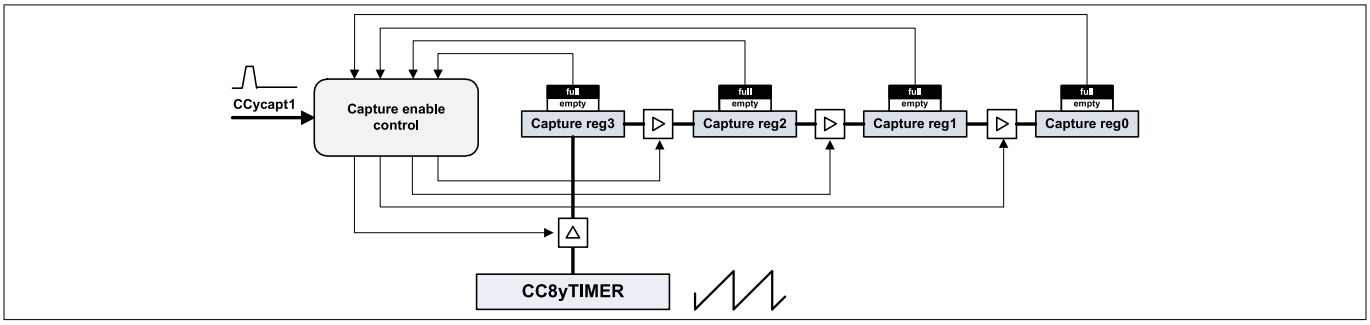


Figure 341 Slice Capture Logic - CC8yTC.SCE = 1_B

22.2.5.7 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 **CC8yINS1**.EVxIS register and indicating the active level of the signal on the **CC8yINS2**.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 **CC8yINS2**.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 CC8ySTn bit - **CC8yTC**.EMT = 0_B
- modulation event is used to gate the outputs - **CC8yTC**.EMT = 1_B

On Figure 342, 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 CC8ySTn bit is cleared and the output is forced into the PASSIVE state. Notice that the values of the status bit, CC8ySTn 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 CC8ySTn 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 343.

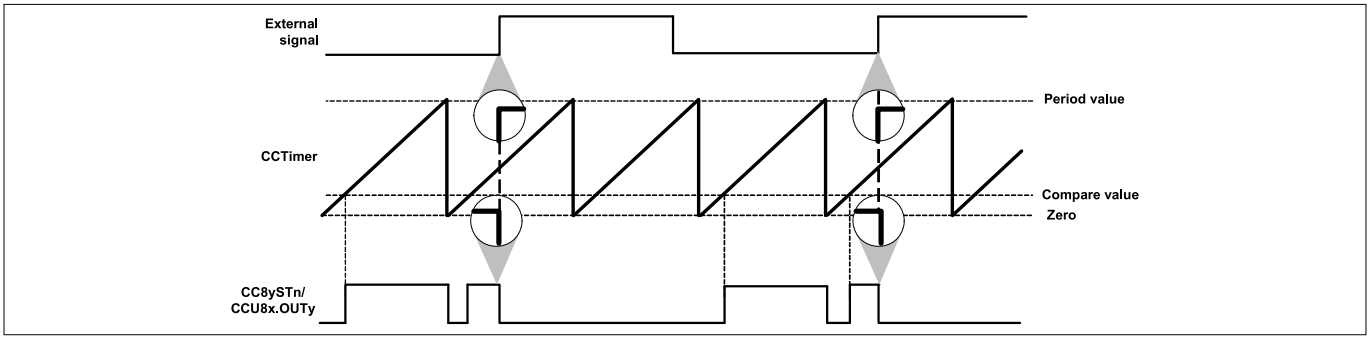


Figure 342 External modulation resets the ST bit - CC8yTC.EMS = 0_B

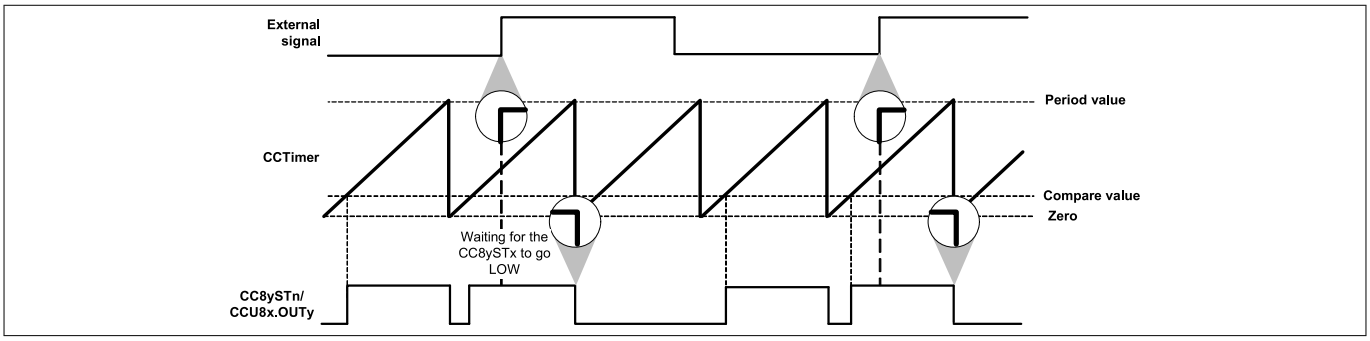


Figure 343 External modulation clearing the ST bit - CC8yTC.EMS = 1_B

On Figure 344, 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, **CC8yINS2**.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.

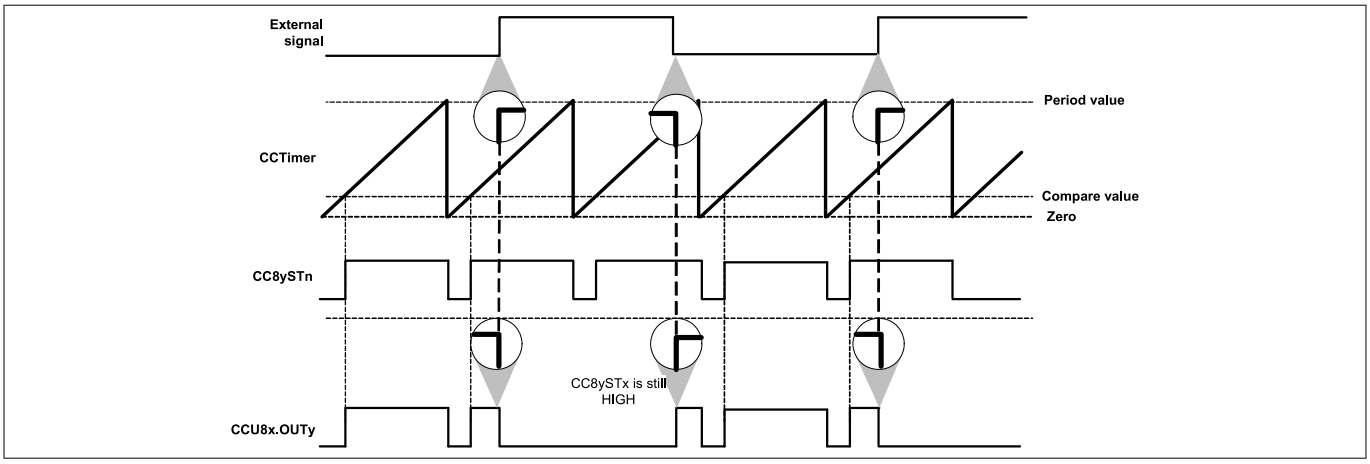


Figure 344 External modulation gating the output - CC8yTC.EMT = 1_B

22.2.5.8 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 **CC8yINS1**.EV2IS register and indicating the active level of the signal on the **CC8yINS2**.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 **CC8yINS2**.EV2LM.

There are two bitfields that can be monitored via software to crosscheck the TRAP function, Figure 345:

- 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.

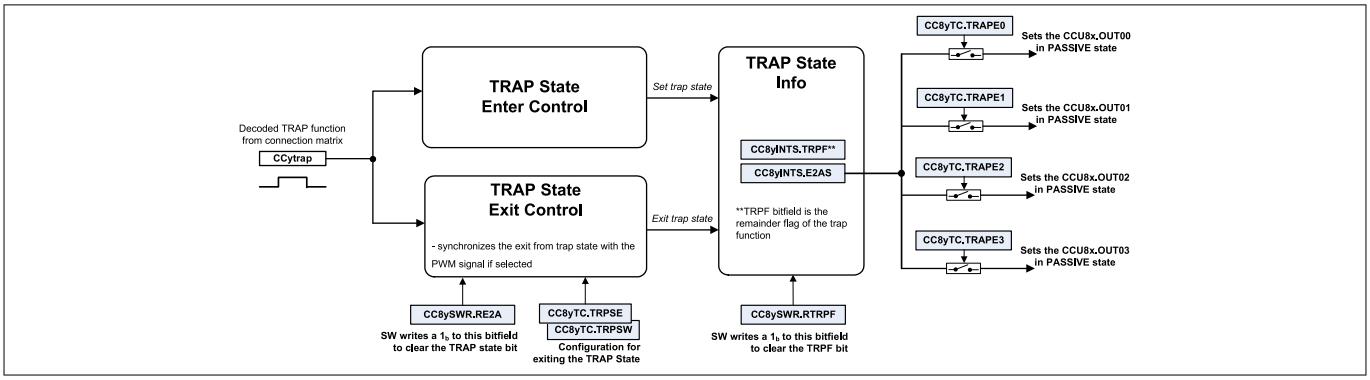


Figure 345 Trap control diagram

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 346.

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

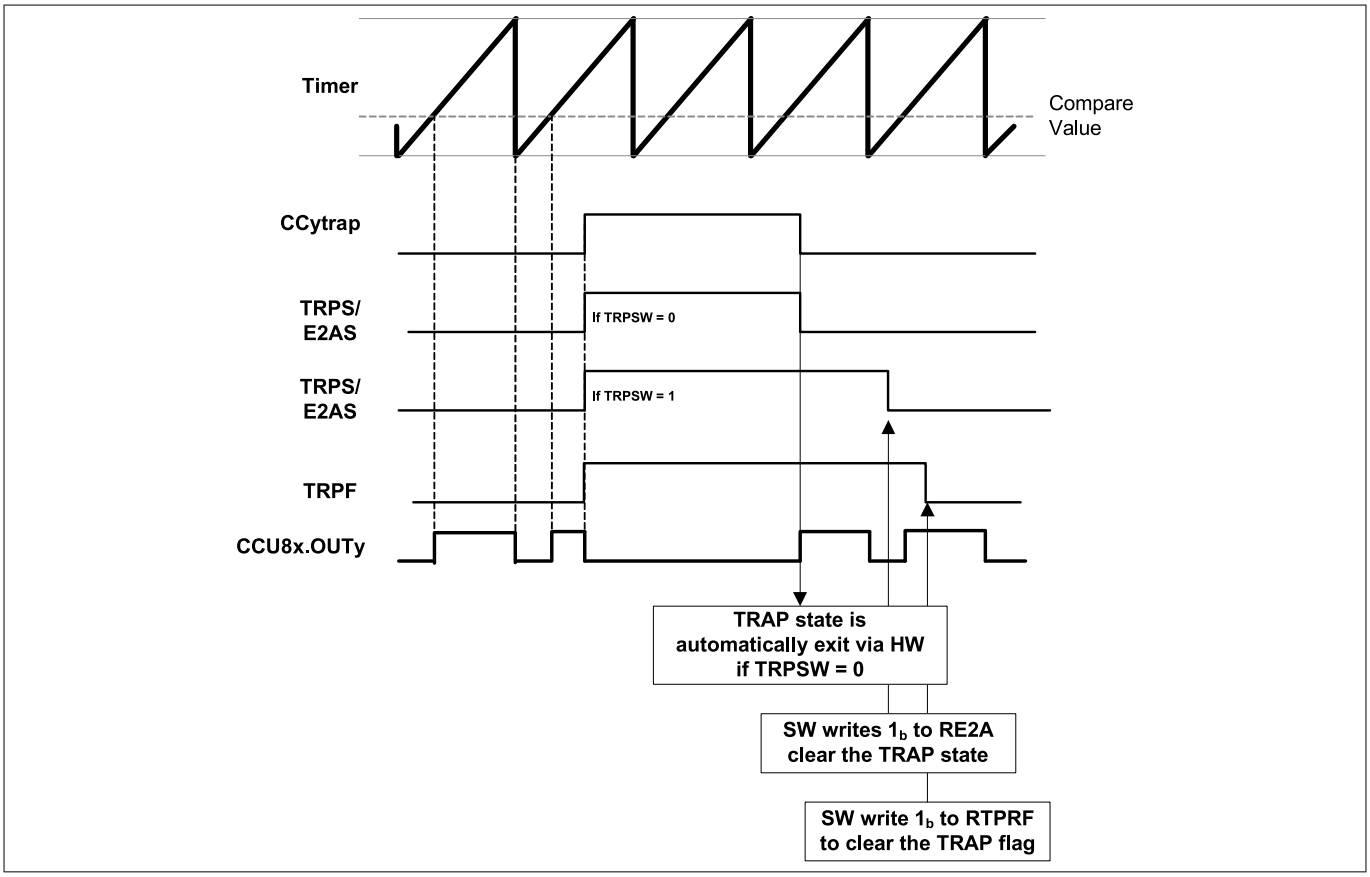


Figure 346 Trap timing diagram, CC8yTCST.CDIR = 0 CC8yPSL.PSL = 0

It is also possible to synchronize the exiting of the TRAP state with the PWM signal, Figure 347. This function is enabled when the bitfield **CC8yTC**.TRPSE = 1_B.

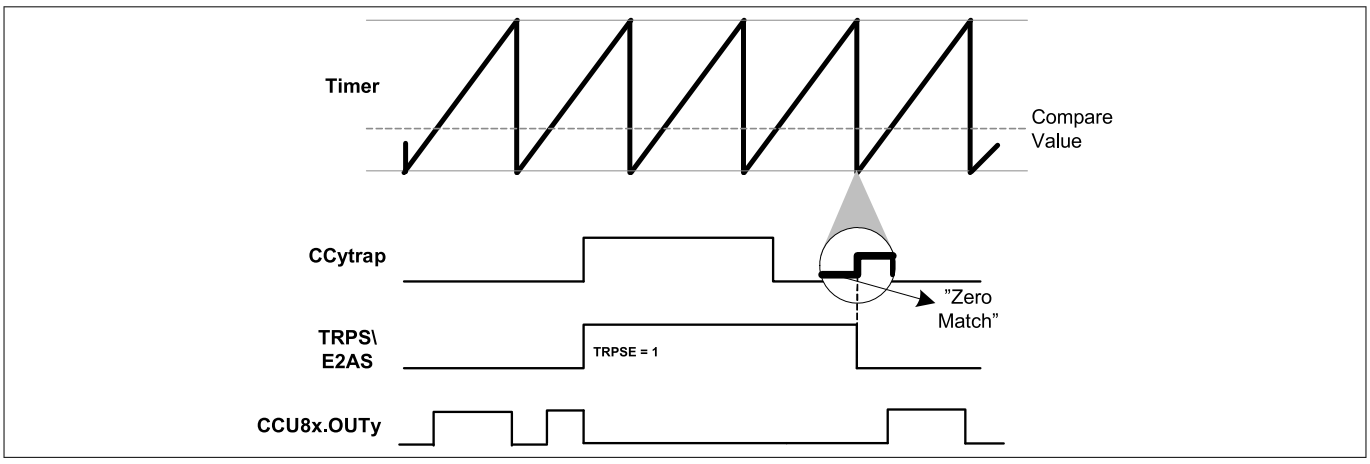


Figure 347 Trap synchronization with the PWM signal

22.2.5.9 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 **CC8yINS1**.EV1IS register and indicating the active edge of the signal on the **CC8yINS2**.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 **CC8yINS1**.EV2IS register. The **CC8yINS2**.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 348 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.

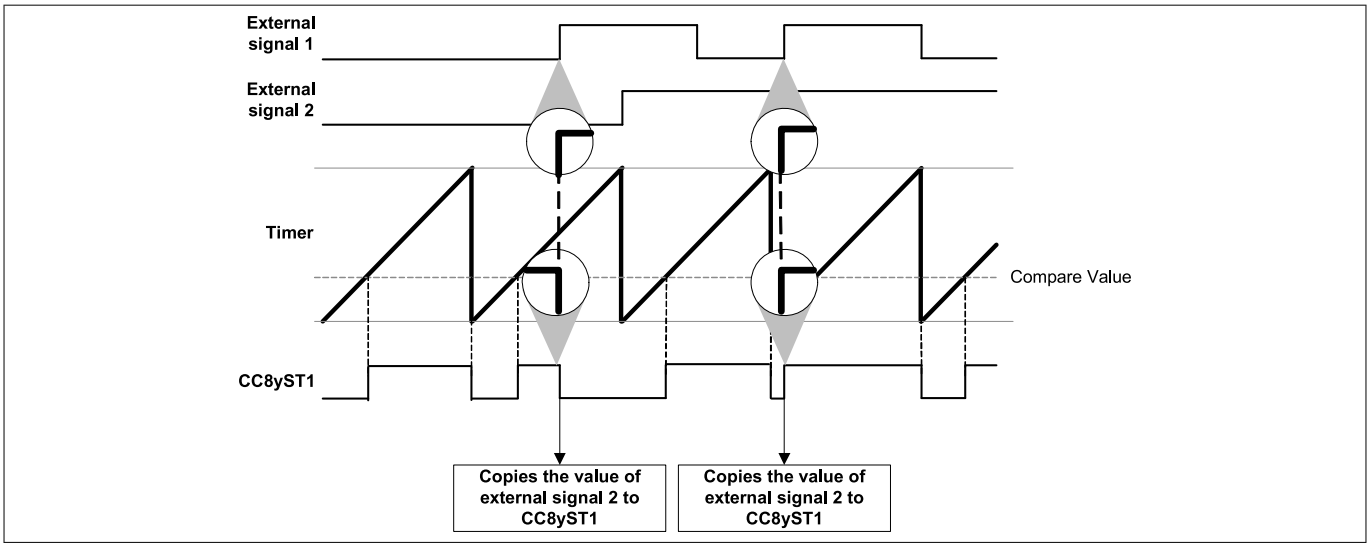


Figure 348 Status bit override

22.2.6 Timer Slice Advanced Functions

In the following sub sections several advanced functions present in each CC8y slice are described. One should notice that each of the functions is present and works in the same manner for every CCU8 timer slice - being the only difference the Output Parity Checker, due to the fact that this functions can use all the CCU8 PWM outputs (in this case the Output Parity Checker is a CCU8 and not a timer slice - CC8y - function).

22.2.6.1 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 Multi-Channel 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: CCU8xMCI0[3:0] 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 349.

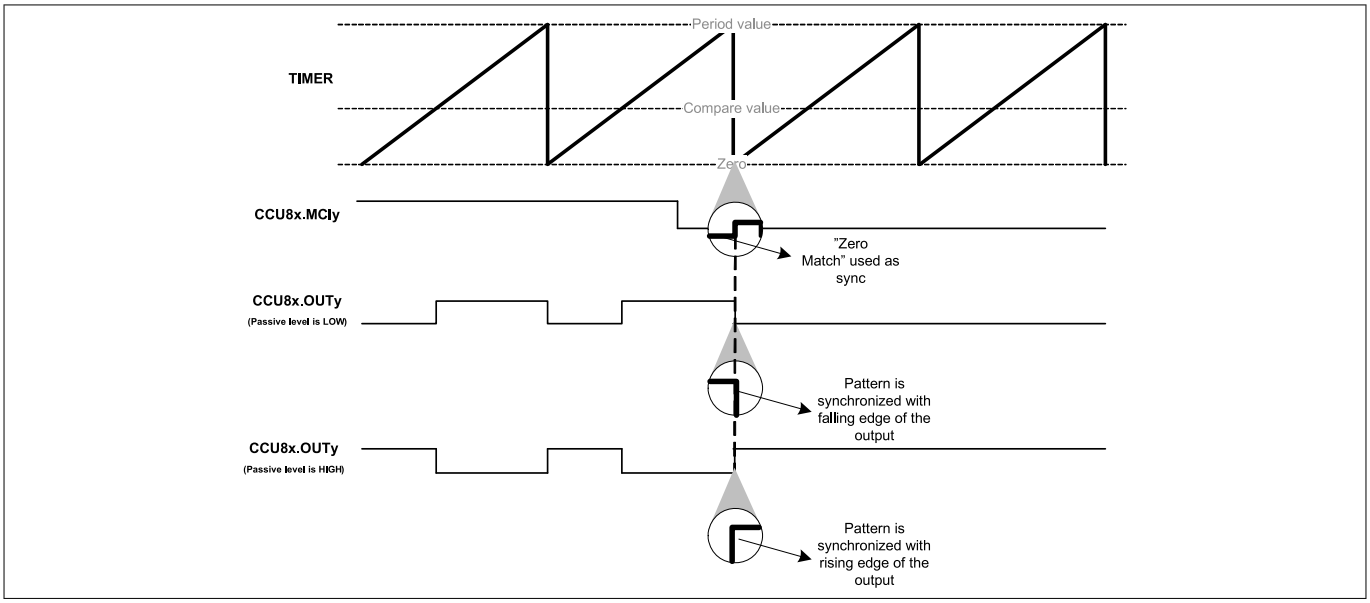


Figure 349 Multi-Channel pattern synchronization

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.MCIy[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 350 shows the simplified scheme for the Multi-Channel control.

Figure 351, 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.

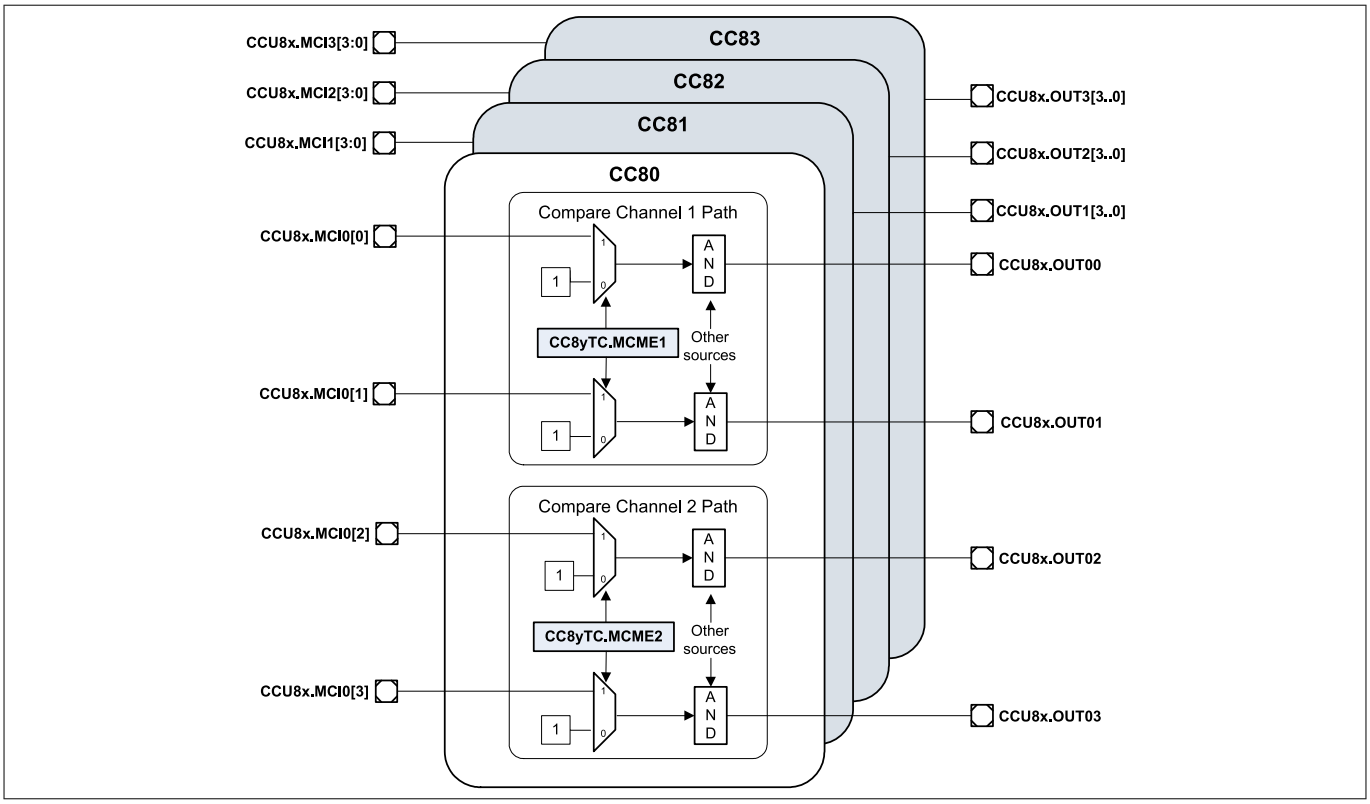


Figure 350 CCU8 Multi-Channel overview

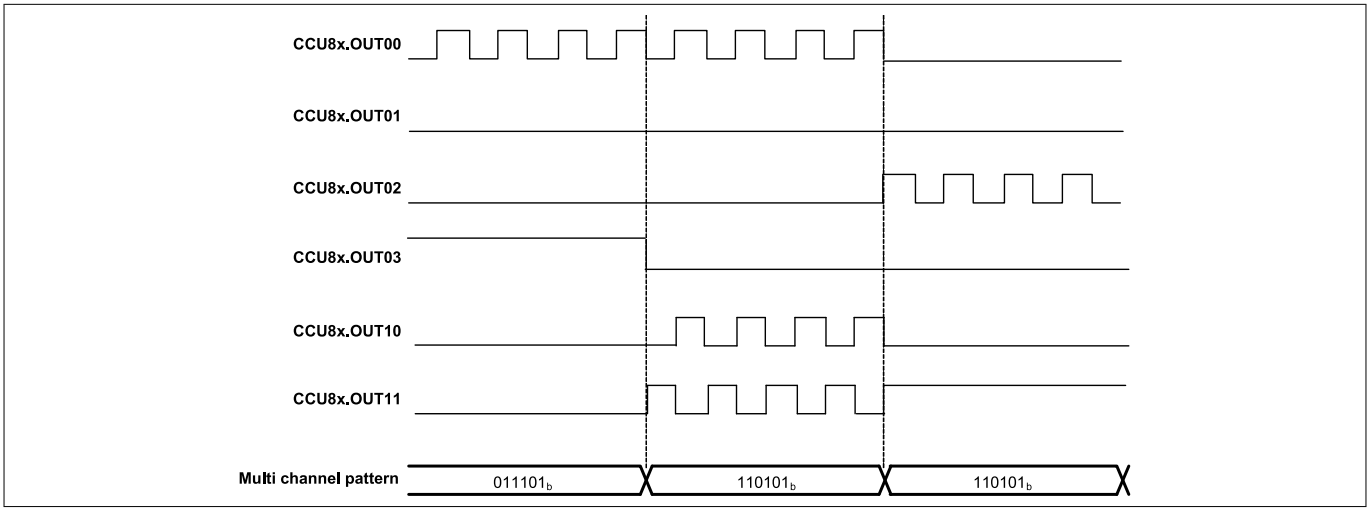


Figure 351 Multi-Channel mode for multiple Timer Slices

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, CC8ySTn and the direct control of the output pin). This path is only selected when **CC8yTC**.MCMEx = 1_B.

On Figure 352 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 353. 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.

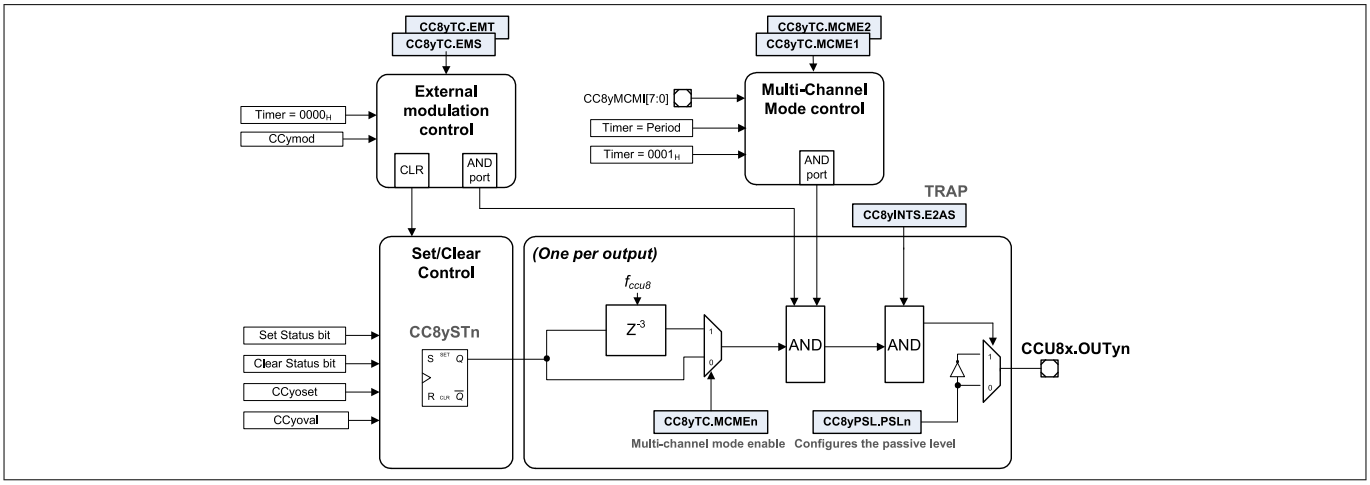


Figure 352 Multi-Channel mode output path

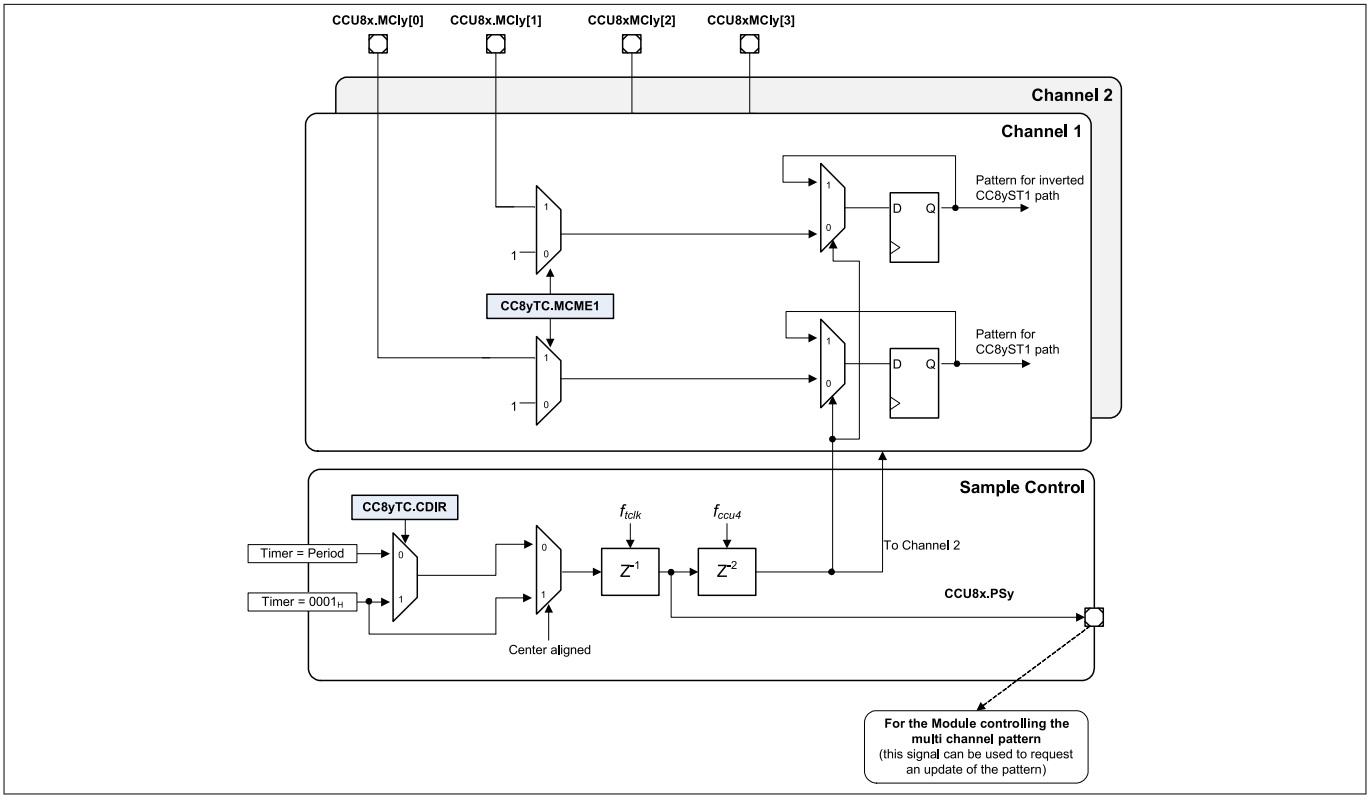


Figure 353 Multi-Channel Pattern Synchronization Control

22.2.6.2 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 354.

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

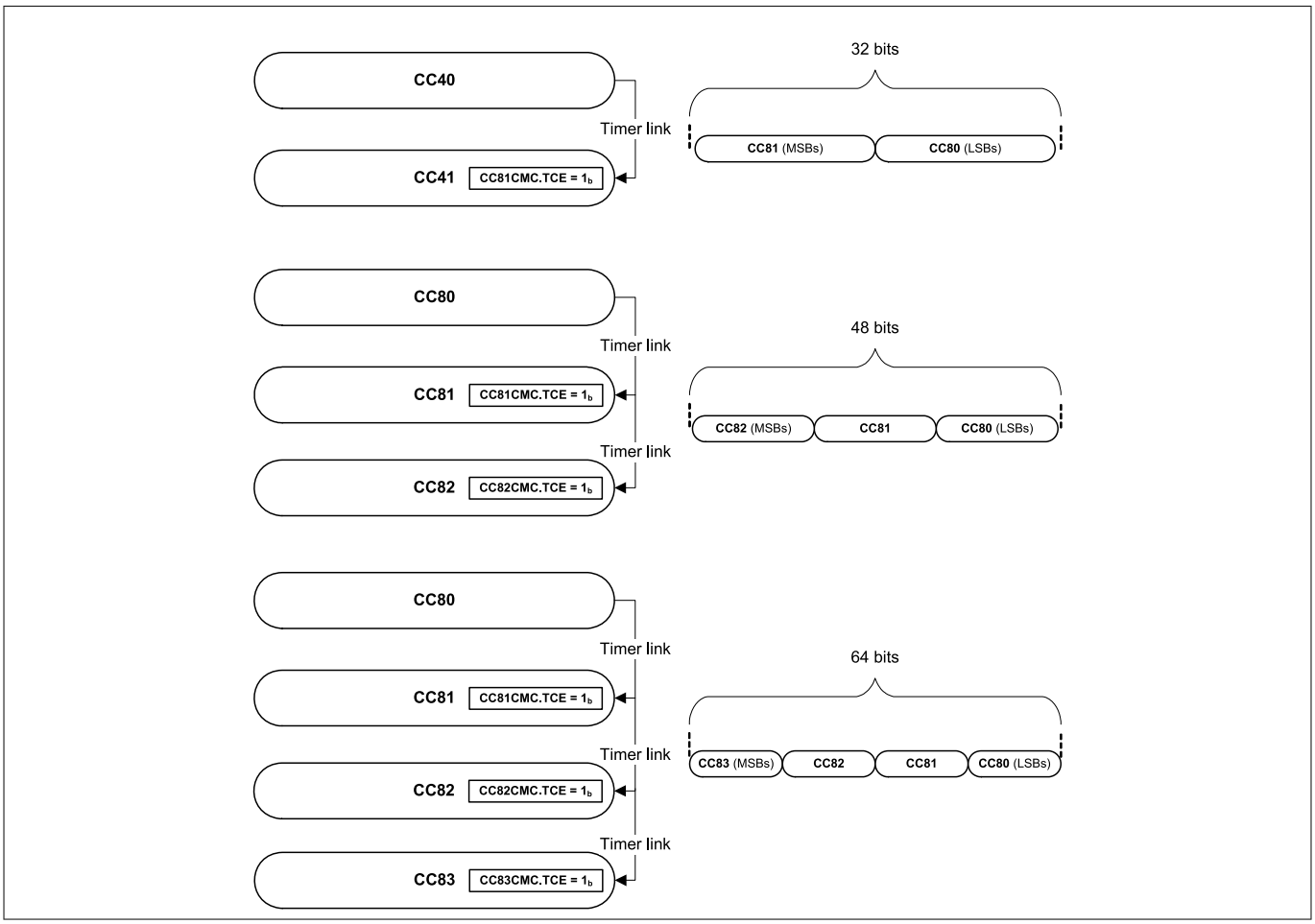


Figure 354 Timer concatenation example

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 355.

***Note**: For all the cases, CC80 and CC83 are not considered adjacent slices*

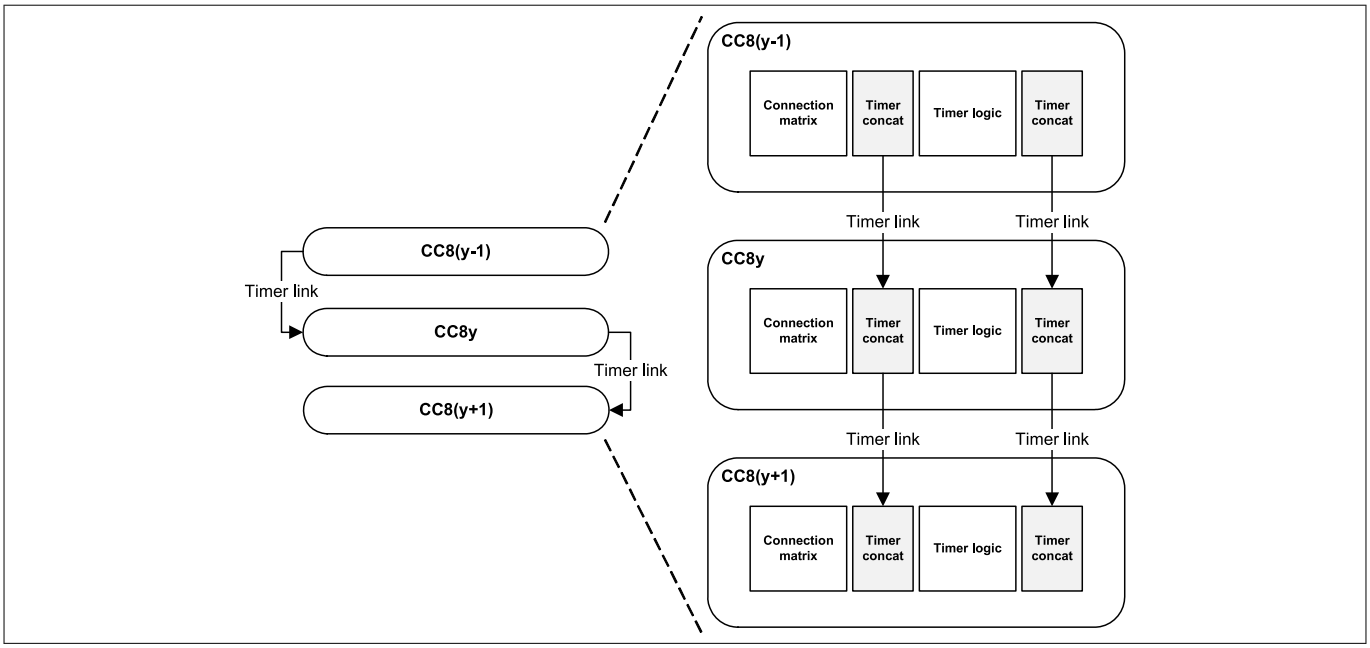


Figure 355 Timer concatenation link

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 355. 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 356.

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.

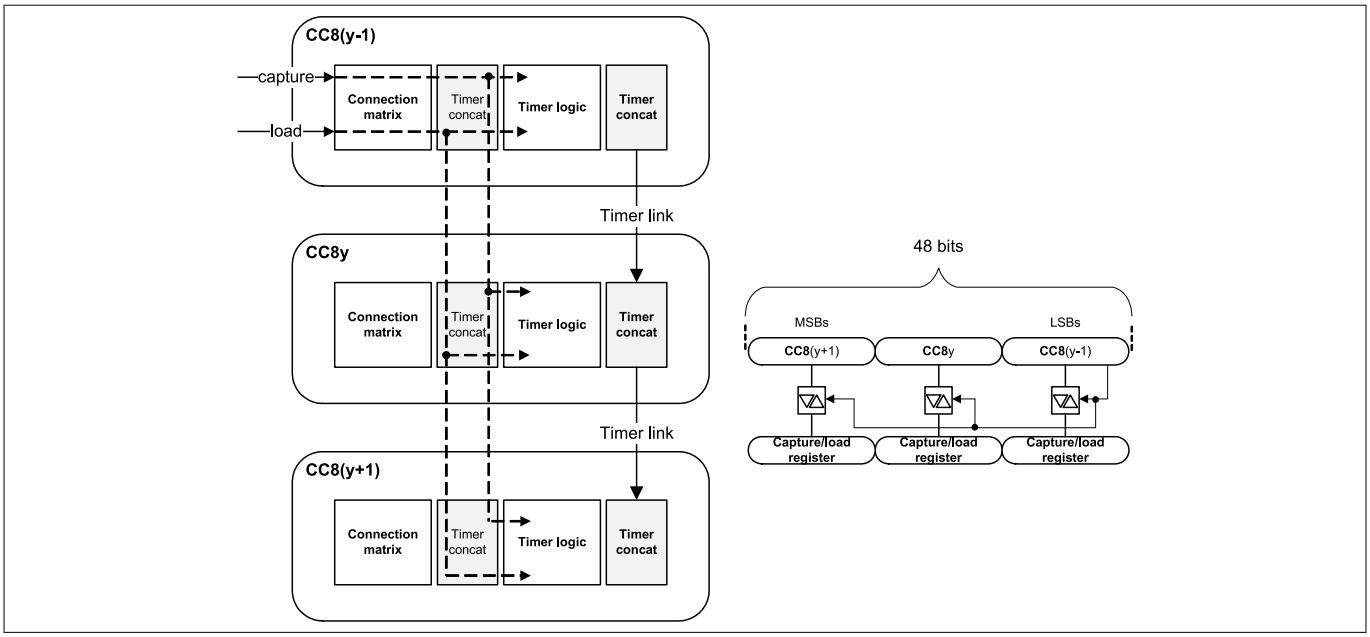


Figure 356 Capture/Load Timer Concatenation

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 357.

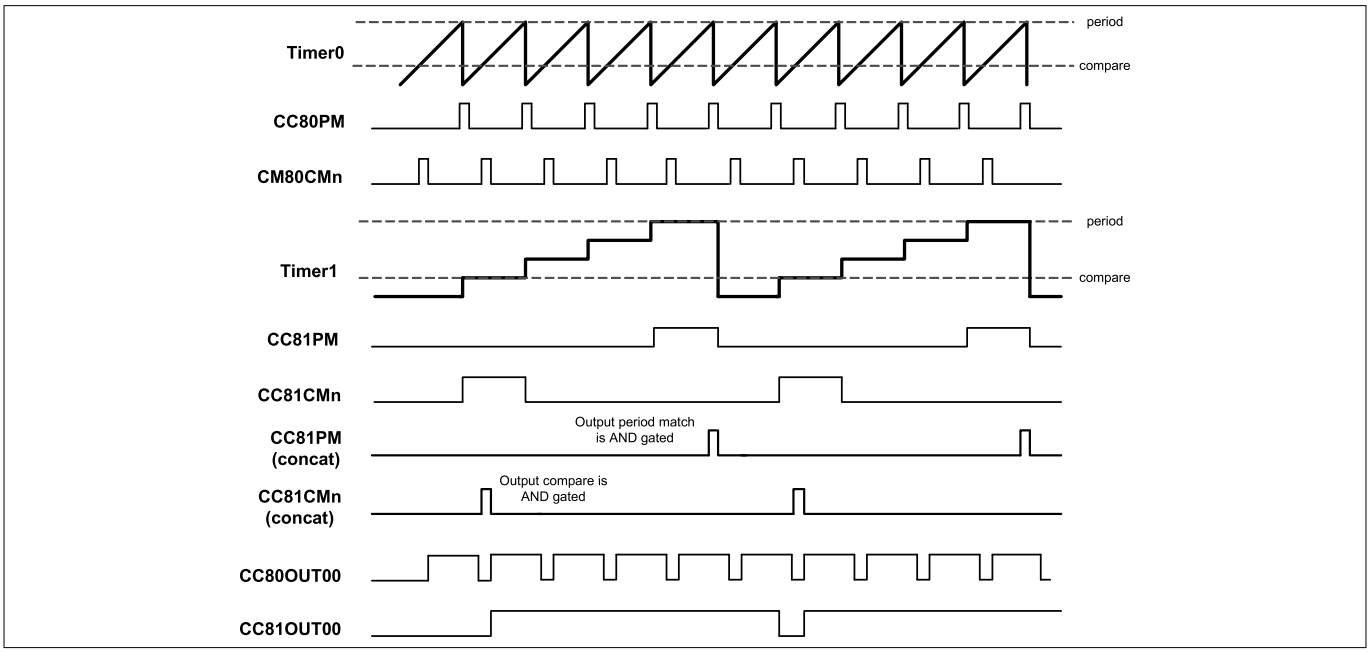


Figure 357 32 bit concatenation timing diagram

***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 358 gives an overview of the timer concatenation logic. Notice that all the mechanism is controlled solely by the **CC8yCMC**.TCE bitfield.

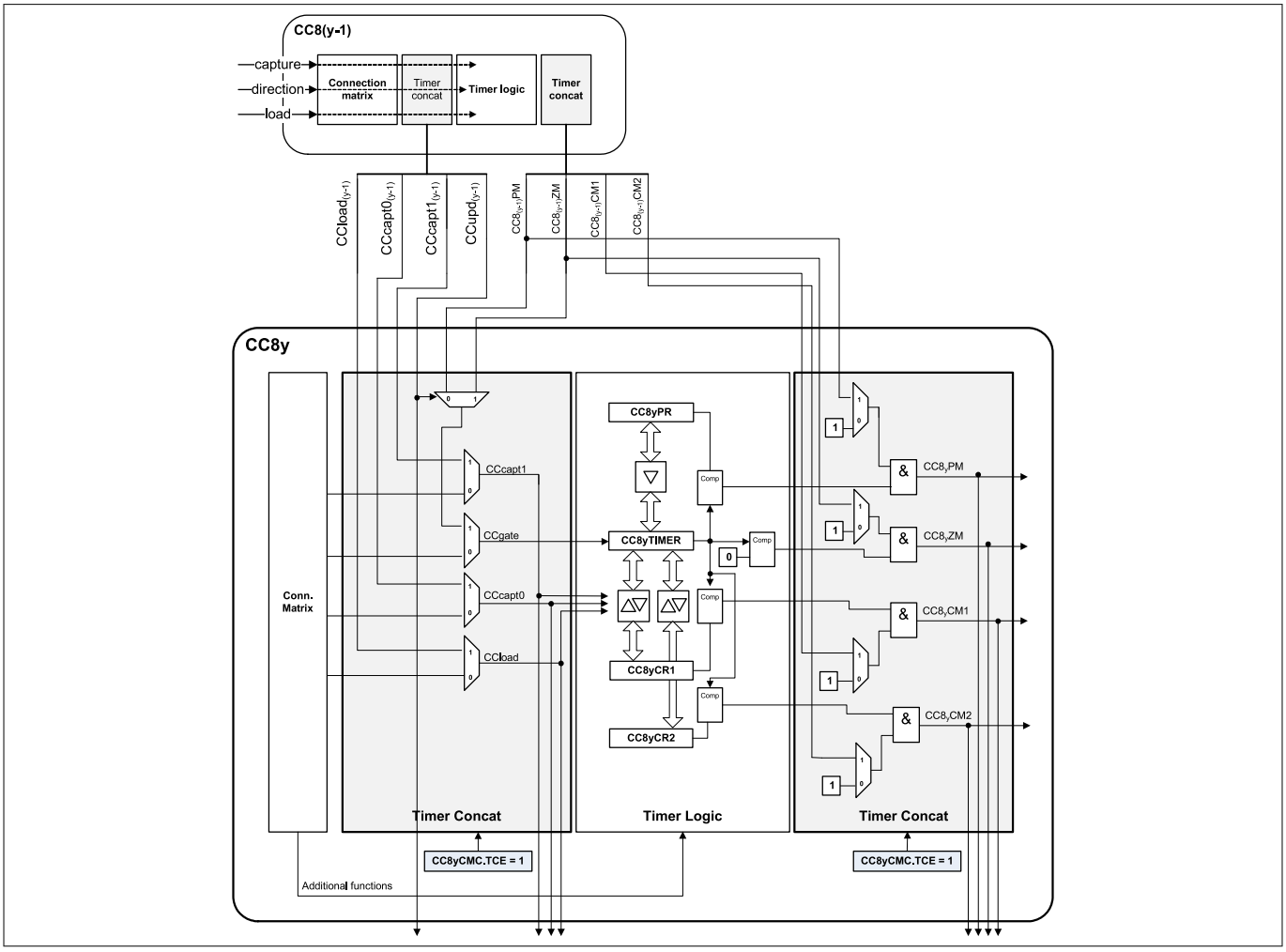


Figure 358 Timer concatenation control logic

22.2.6.3 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 359.

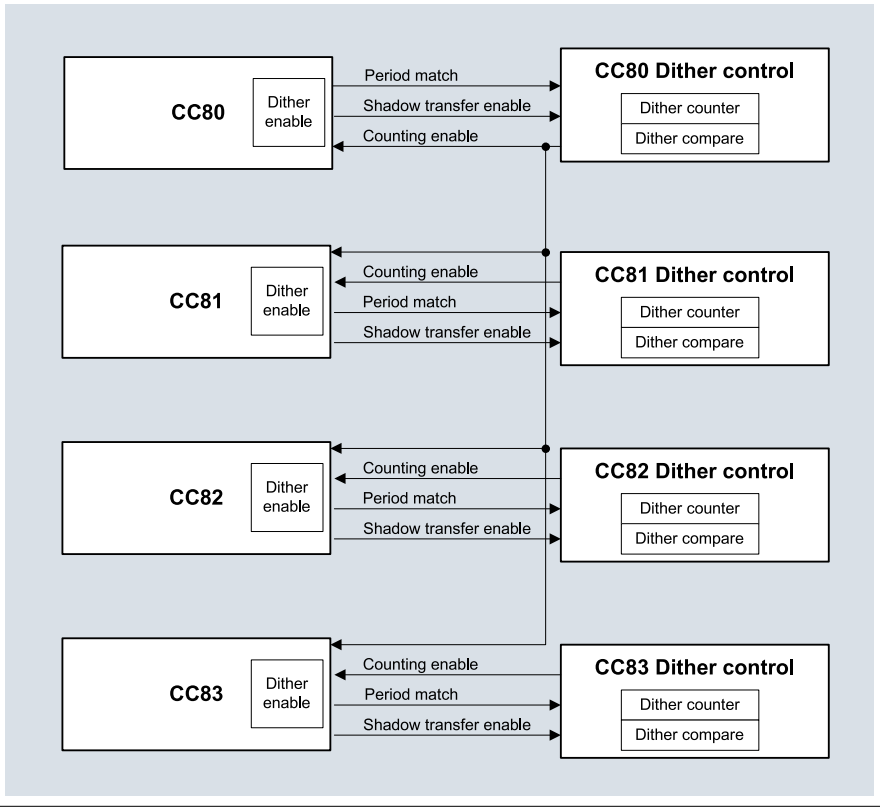


Figure 359 Dither structure overview

The dither control unit contains a 4 bit counter and a compare value. The counter works in a bit reverse mode so the distribution of increments stays uniform over 16 counter periods, see Table 181.

Table 181 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

(table continues...)

Table 181 (continued) Dither bit reverse counter

counter[3]	counter[2]	counter[1]	counter[0]
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 182) for one clock cycle.

Table 182 Dither modes

DITHE[1]	DITHE[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 360.

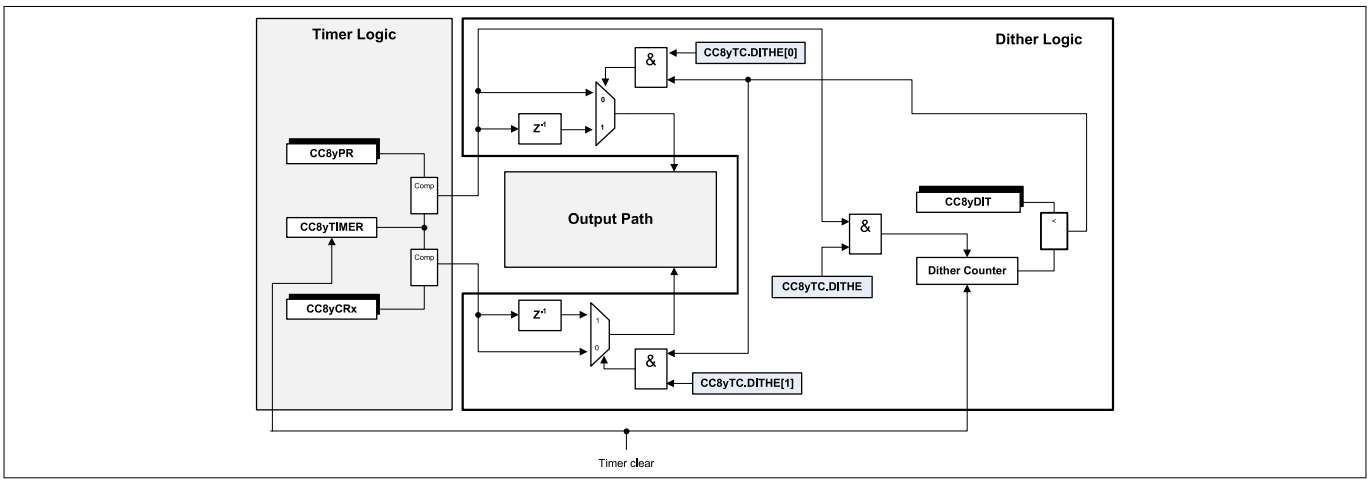


Figure 360 Dither control logic

Figure 361 to Figure 366 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).

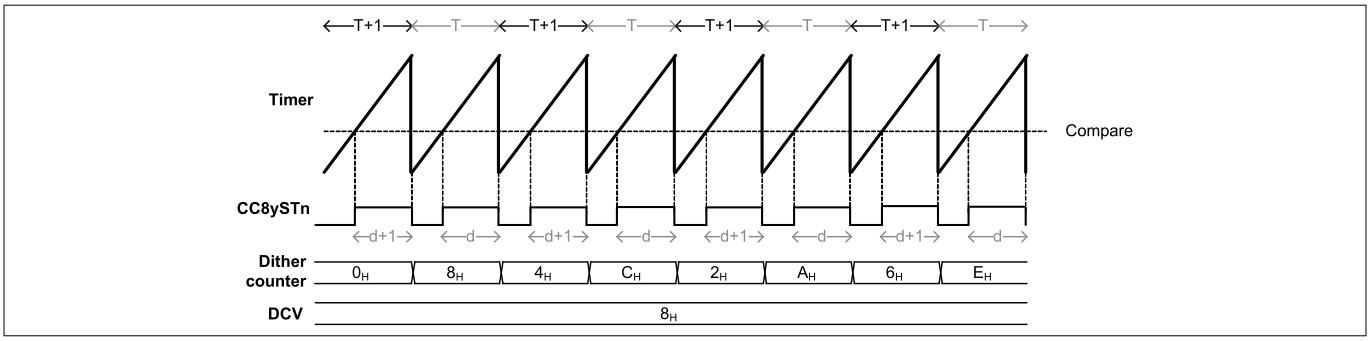


Figure 361 Dither timing diagram in edge aligned - CC8yTC.DITHE = 01_B

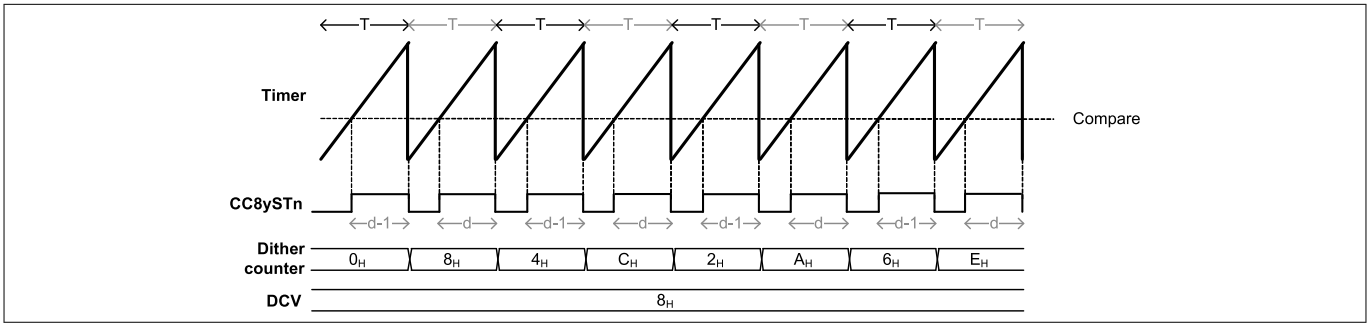


Figure 362 Dither timing diagram in edge aligned - CC8yTC.DITHE = 10_B

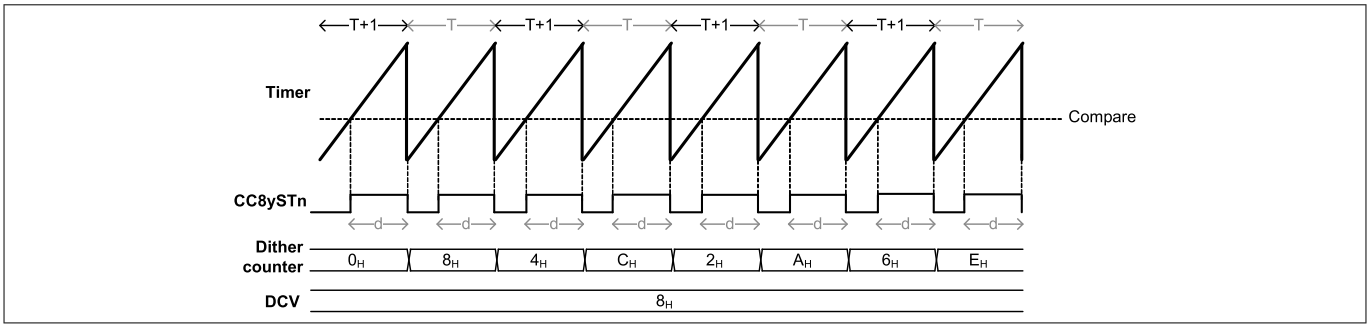


Figure 363 Dither timing diagram in edge aligned - CC8yTC.DITHE = 11_B

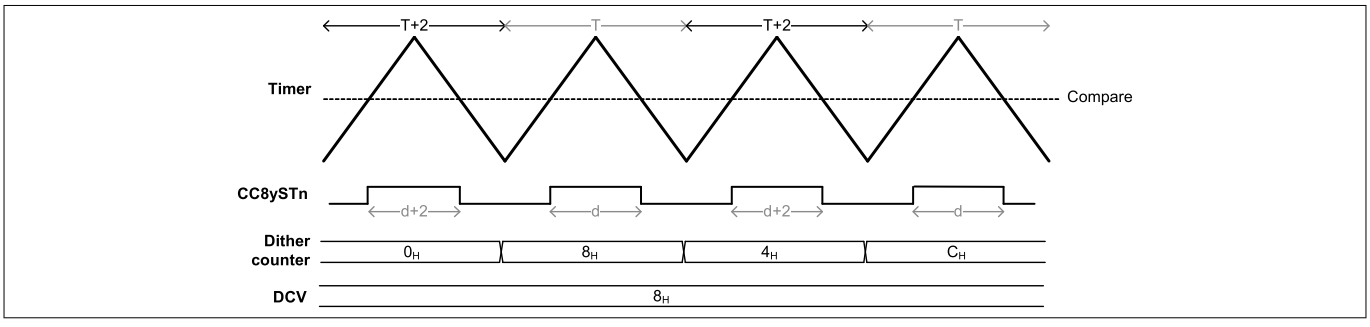


Figure 364 Dither timing diagram in center aligned - CC8yTC.DITHE = 01_B

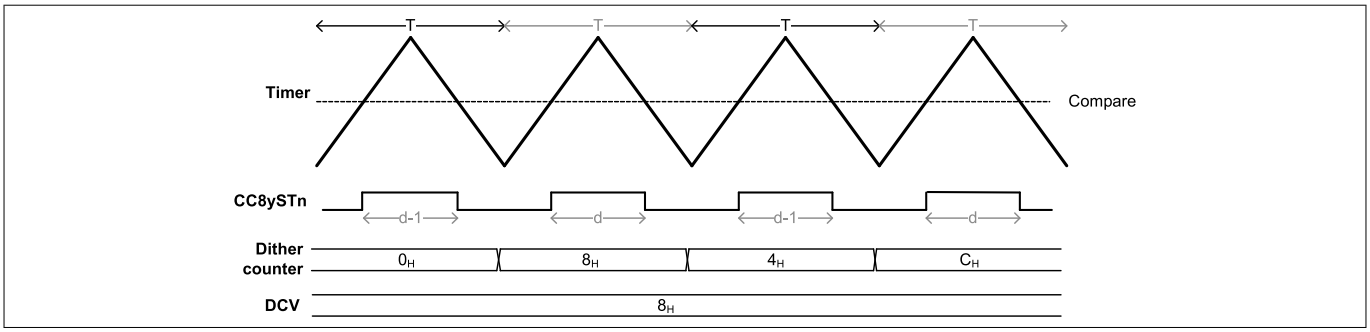


Figure 365 Dither timing diagram in center aligned - CC8yTC.DITHE = 10_B

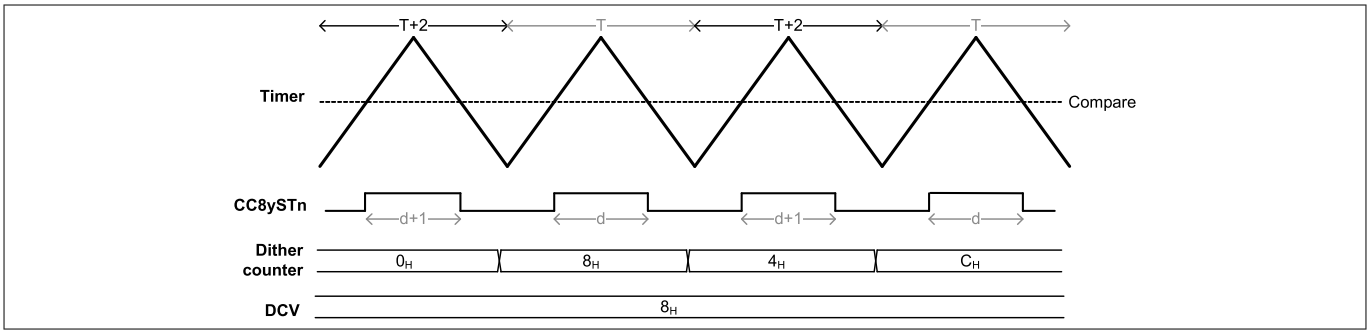


Figure 366 Dither timing diagram in edge aligned - CC8yTC.DITHE = 11_B

***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.*

22.2.6.4 Capture Extended Read Back Mode

Each Timer Slice capture logic can operate in a FIFO read back mode. This mode can be enabled by setting the **CC8yTC**.ECM = 1_B. This Extended Read back mode allows the software to read back the capture data always from the same address (**CC8yECRD0** for the structure linked with the capture trigger 0 or **CC8yECRD1** for the one linked with capture trigger 1). This read back will always return the oldest captured value, enabling an easy software routine implementation for reconstructing the capture data.

This function allows the usage of a FIFO structure for each capturing trigger. This relaxes the software read back routine when multiple capture triggers are present, and the software is not fast enough to perform a read operation in each capture event.

This FIFO read back function is present for a depth-4 and depth-2 FIFO structure.

The read back data contains also a lost value bitfield, that indicates if a capture trigger was lost due to the fact that the FIFO structure was full. This bitfield is set whenever a capture event was sensed and the FIFO was full (regardless if the continuos capture mode was enabled or not). This bitfield is cleared automatically by HW whenever the next read of the **CC8yECRD0**/**CC8yECRD1** register occurs. This bitfield does not indicate how many capture events were lost, it just indicates that between two ECRD reads at least a capture event was lost (this can help the SW evaluate which part of the data read, can be used for calculation).

***Note**: When the ECM bitfield is set, reading the individual capture registers is still possible. Nevertheless the full flags can only be cleared by the HW when a read back is done via the **CC8yECRD0**/**CC8yECRD1** address.*

Depth 4 Structure

The FIFO depth-4 structure is present in the hardware when the capture trigger 1 is enabled and the **CC8yTC**.SCE = 1_B (same capture event), Figure 367.

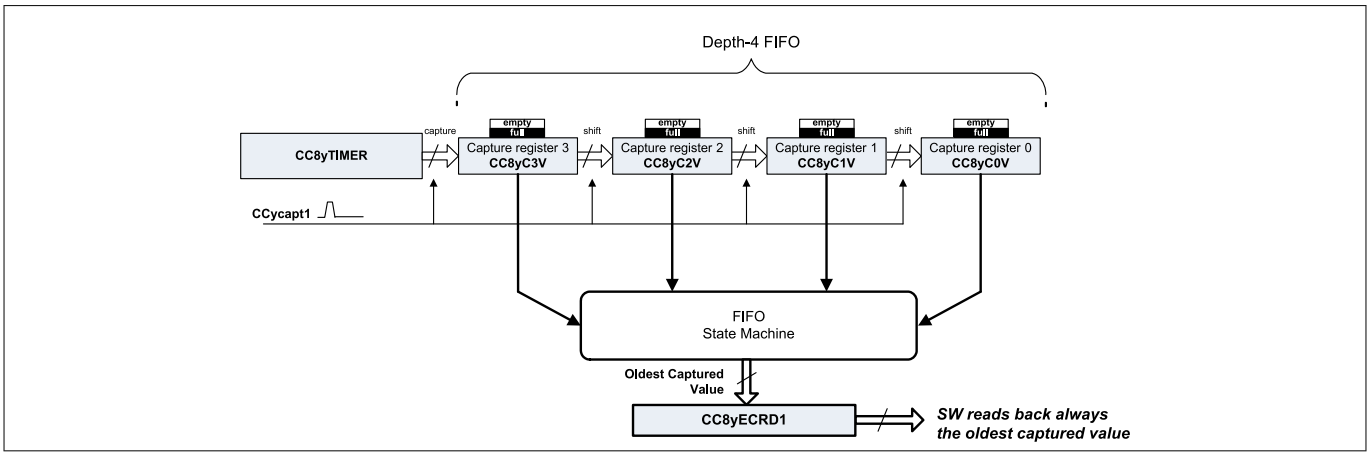


Figure 367 Capture Extended Read Back - Depth 4

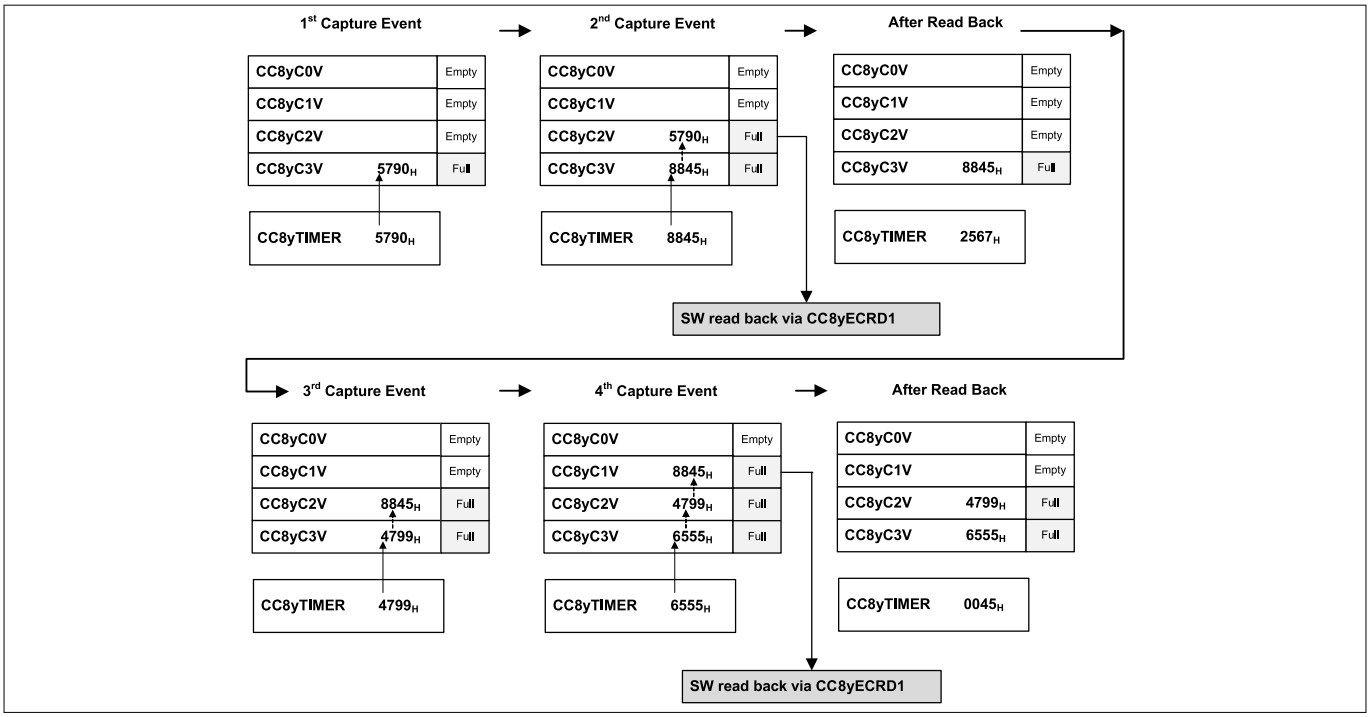


Figure 368 Depth 4 software access example

Depth 2 Structure

Each Timer Slice can have two capture structures of depth-2: one used with capture trigger 0 and another with capture trigger 1.

The one linked with capture trigger 0, is accessed via the **CC8yECRD0** while the one linked with the capture trigger 1 is accessed via the **CC8yECRD1**, Figure 369.

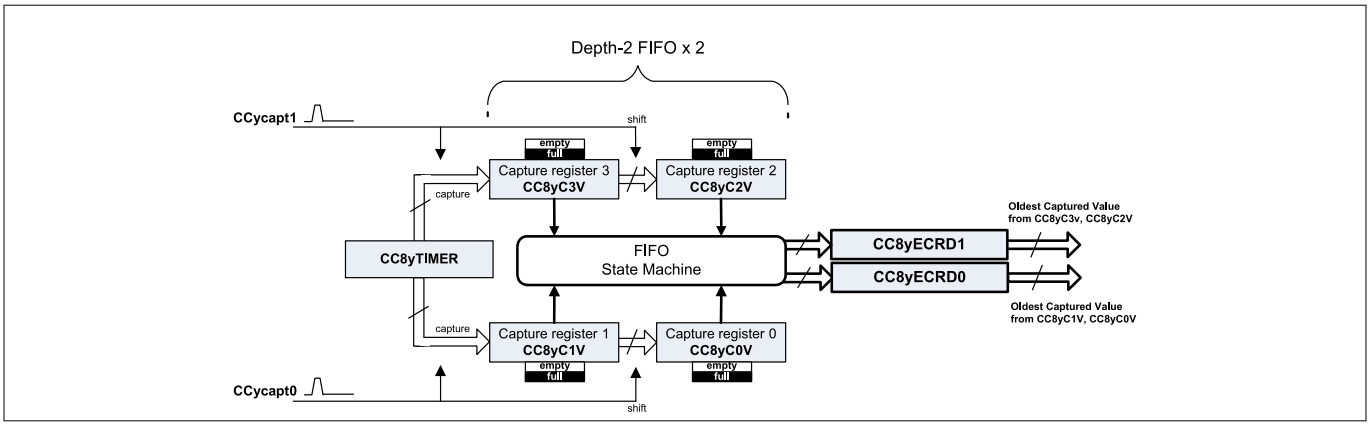


Figure 369 Capture Extended Read Back - Depth 2

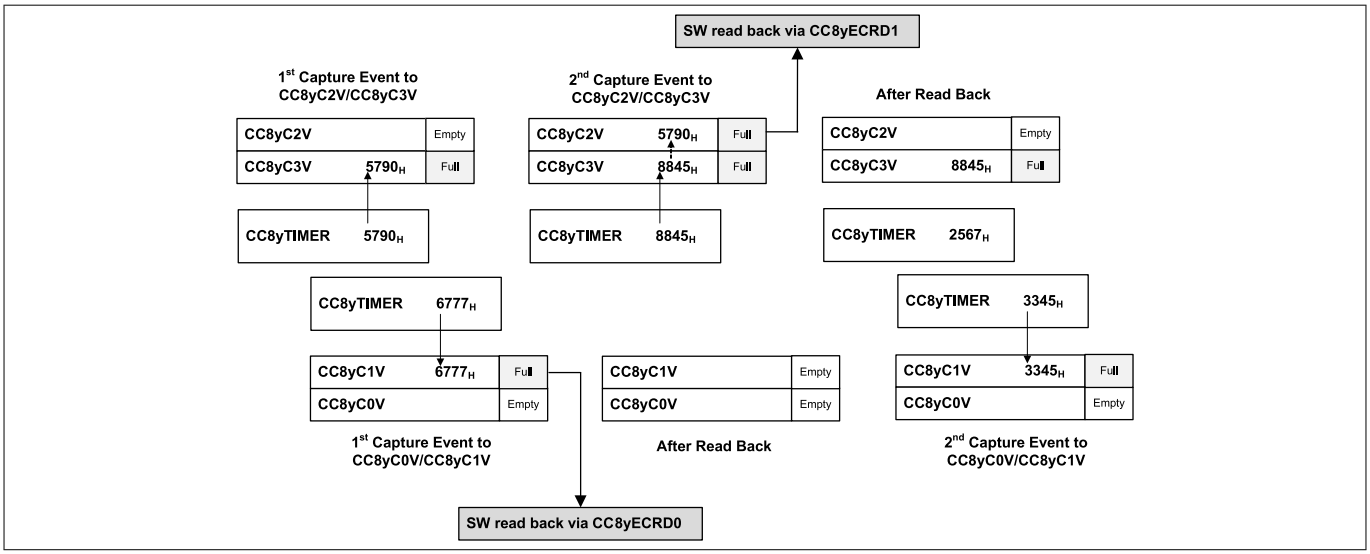


Figure 370 Depth 2 software access example

22.2.6.5 Output Parity Checker

The parity checker function can be enabled by setting the **GIDLC**.SPCH 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 371 shows the structure of the parity checker unit.

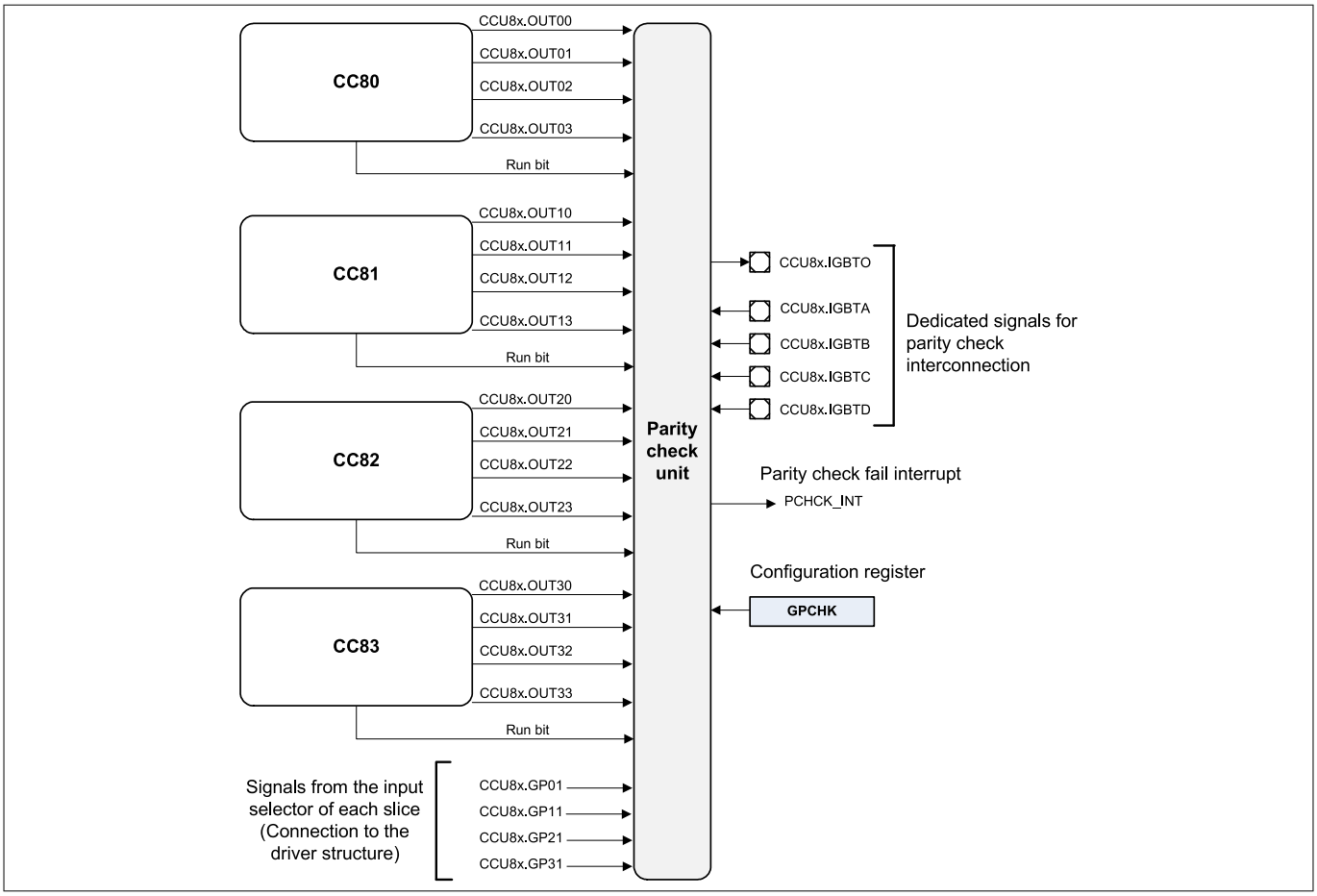


Figure 371 Parity checker structure

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 372. For a more detailed description of resource usage for the parity checker function, please address Parity Checker Usage.

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:

CC8yINS1.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.IGBTC 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.

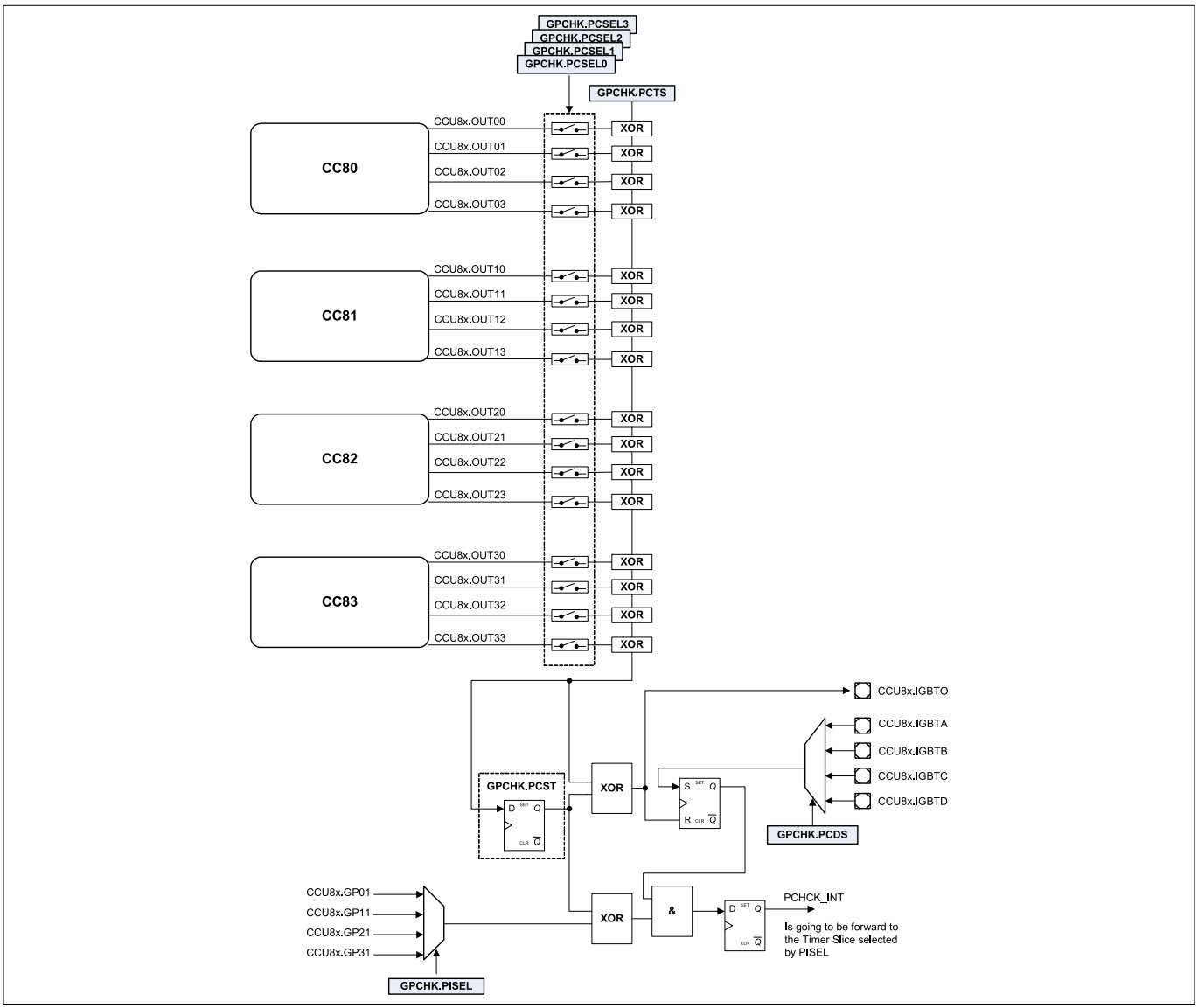


Figure 372 Parity checker logic

22.2.7 Clock 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.

22.2.7.1 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 183. 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 183 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	$f_{ccu8}/32768$

22.2.7.2 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 373 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 do 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 mode, 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 Updating the Period, Duty Cycle and other PWM conditions.

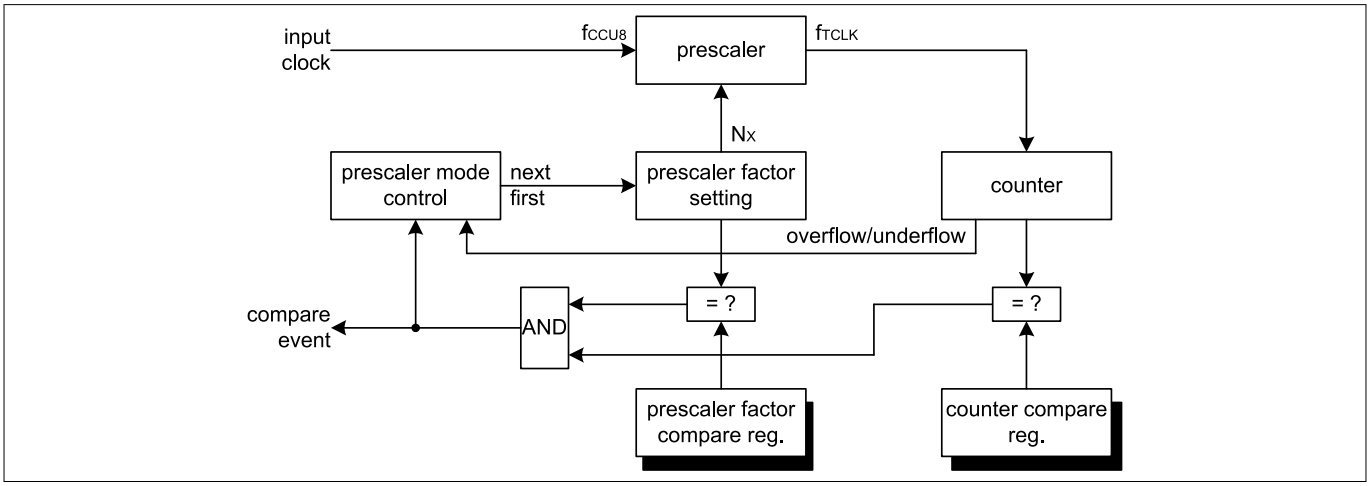


Figure 373 Floating prescaler in compare mode overview

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 CCU8 Usage for a full description of the usage of the floating prescaler mode in conjunction with compare and capture modes.

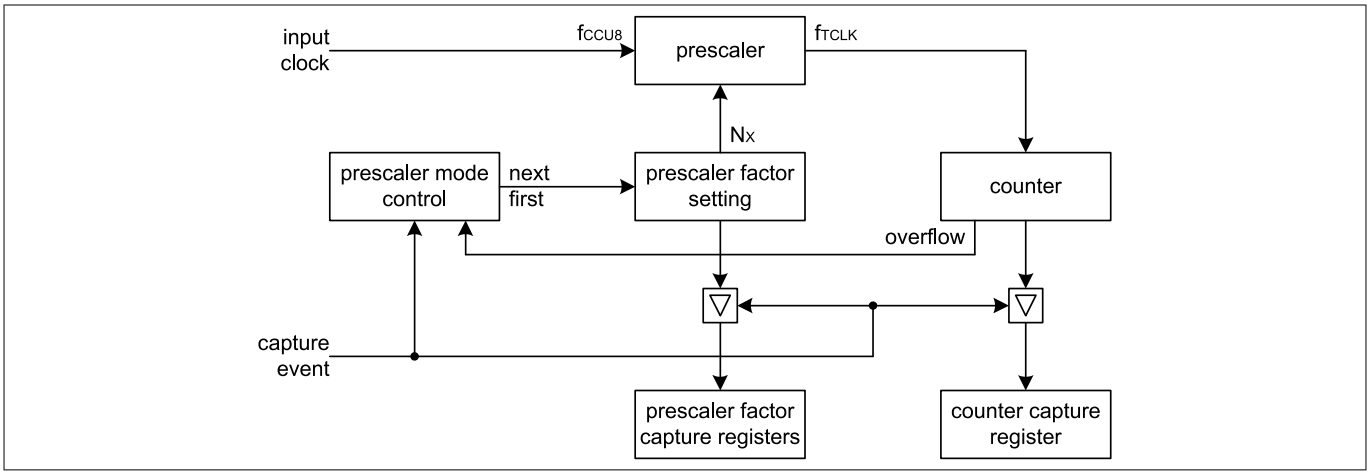


Figure 374 Floating Prescaler in capture mode overview

22.2.8 CCU8 Usage

22.2.8.1 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 375.

In the same manner a 100% duty cycle signal can be generated in Center Aligned Mode, Figure 376.

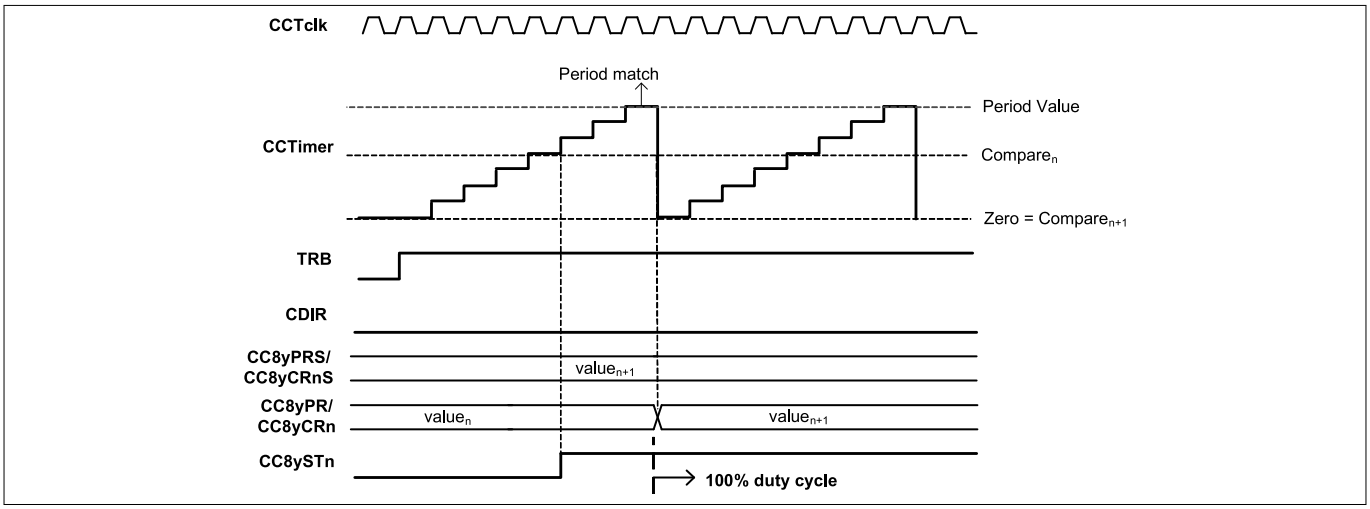


Figure 375 PWM with 100% duty cycle - Edge Aligned Mode

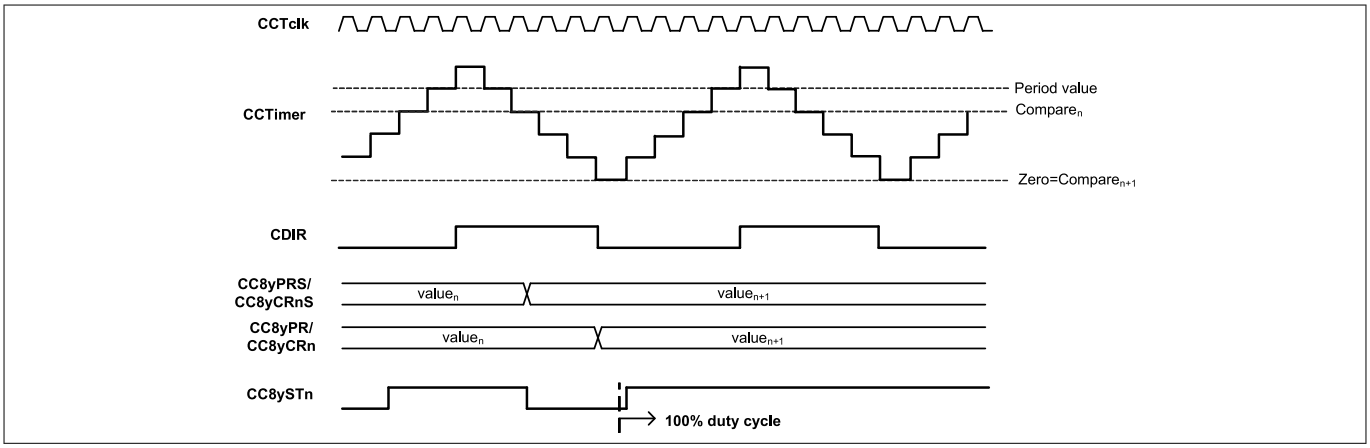


Figure 376 PWM with 100% duty cycle - Center Aligned Mode

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 377.

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 378.

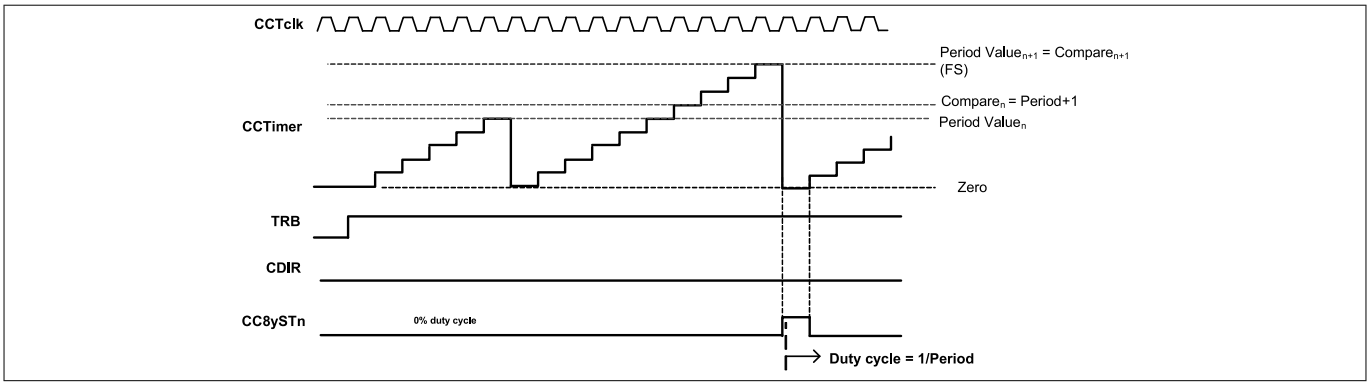


Figure 377 PWM with 0% duty cycle - Edge Aligned Mode

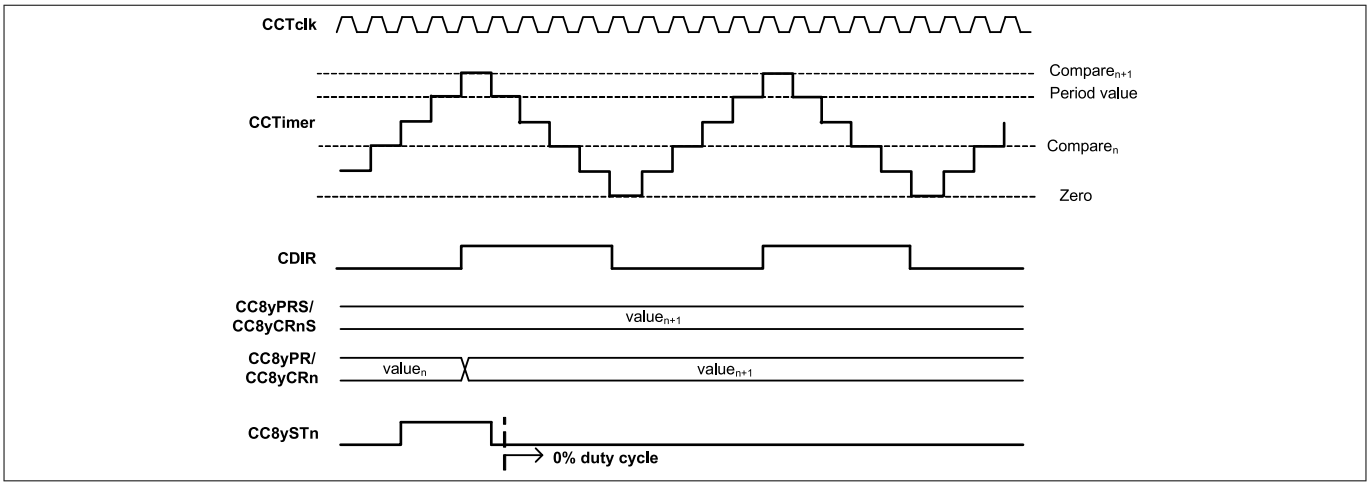


Figure 378 PWM with 0% duty cycle - Center Aligned Mode

22.2.8.2 Prescaler Usage

In Normal Prescaler Mode, the frequency of the f_{tclk} fed to the specific CC8y is chosen from the Table 183, 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 183. 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 379.

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 than 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.

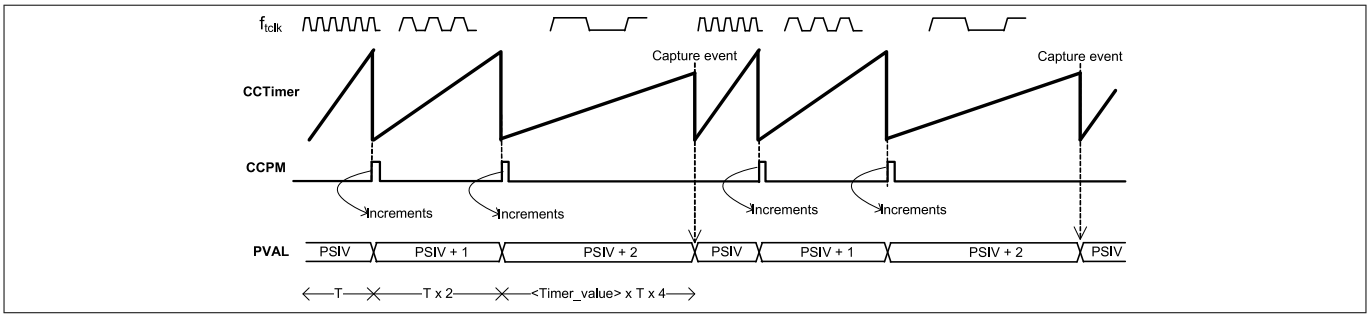


Figure 379 Floating Prescaler capture mode usage

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 do 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 mode, 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 380, 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 381.

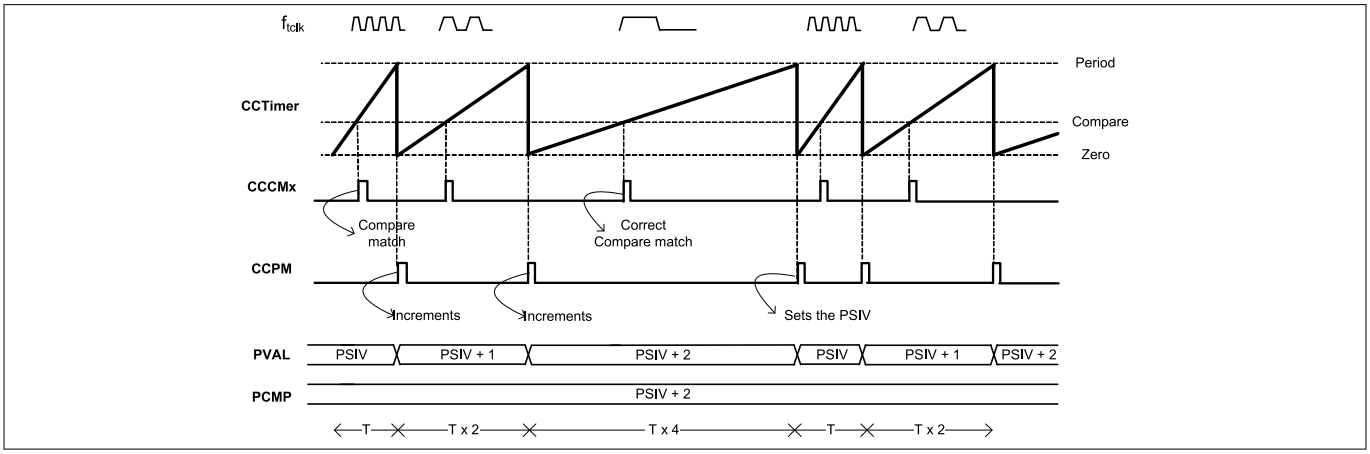


Figure 380 Floating Prescaler compare mode usage - Edge Aligned

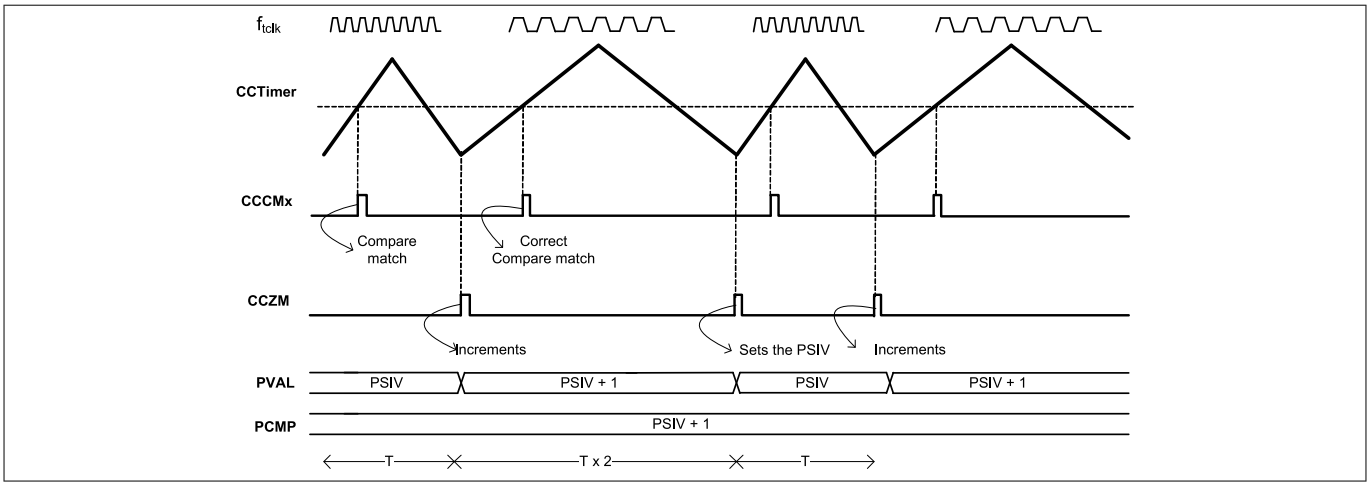


Figure 381 Floating Prescaler compare mode usage - Center Aligned

22.2.8.3 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 184, 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 184 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

(table continues...)

Table 184 (continued) Bit reverse distribution

	DCV															
Dither counter	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
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
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

$$Period = [(16 - \mathrm{DCV}) \times \mathrm{T} + \mathrm{DCV} \times (\mathrm{T} + 1)]/16; \text{ in Edge Aligned Mode} \quad (67)$$

$$Duty\ cycle = [(16 - \mathrm{DCV}) \times \mathrm{d}/\mathrm{T} + \mathrm{DCV} \times (\mathrm{d} + 1)/(\mathrm{T} + 1)]/16; \text{ in Edge Aligned Mode} \quad (68)$$

$$Period = [(16 - \mathrm{DCV}) \times \mathrm{T} + \mathrm{DCV} \times (\mathrm{T} + 2)]/16; \text{ in Center Aligned Mode} \quad (69)$$

$$Duty\ cycle = [(16 - \mathrm{DCV}) \times \mathrm{d}/\mathrm{T} + \mathrm{DCV} \times (\mathrm{d} + 2)/(\mathrm{T} + 2)]/16; \text{ in Center Aligned Mode} \quad (70)$$

DITHE = 10_B

$$Period = \mathrm{T}; \text{ in Edge Aligned Mode} \quad (71)$$

$$Duty\ cycle = [(16 - \mathrm{DCV}) \times \mathrm{d}/\mathrm{T} + \mathrm{DCV} \times (\mathrm{d} - 1)/\mathrm{T}]/16; \text{ in Edge Aligned Mode} \quad (72)$$

$$Period = \mathrm{T}; \text{ in Center Aligned Mode} \quad (73)$$

$$Duty\ cycle = [(16 - \mathrm{DCV}) \times \mathrm{d}/\mathrm{T} + \mathrm{DCV} \times (\mathrm{d} - 1)/\mathrm{T}]/16; \text{ in Center Aligned Mode} \quad (74)$$

DITHE = 11_B

$$Period = [(16 - \mathrm{DCV}) \times \mathrm{T} + \mathrm{DCV} \times (\mathrm{T} + 1)]/16;\ \text{in Edge Aligned Mode} \quad (75)$$

$$Duty\ cycle = [(16 - \mathrm{DCV}) \times \mathrm{d}/\mathrm{T} + \mathrm{DCV} \times \mathrm{d}/(\mathrm{T} + 1)]/16;\ \text{in Edge Aligned Mode} \quad (76)$$

$$Period = [(16 - \mathrm{DCV}) \times \mathrm{T} + \mathrm{DCV} \times (\mathrm{T} + 2)]/16;\ \text{in Center Aligned Mode} \quad (77)$$

$$Duty\ cycle = [(16 - \mathrm{DCV}) \times \mathrm{d}/\mathrm{T} + \mathrm{DCV} \times (\mathrm{d} + 1)/(\mathrm{T} + 2)]/16;\ \text{in Center Aligned Mode} \quad (78)$$

where:

T - Original period of the signal, see Calculating the PWM Period and Duty Cycle

d - Original duty cycle of the signal, see Calculating the PWM Period and Duty Cycle

22.2.8.4 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 382 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).

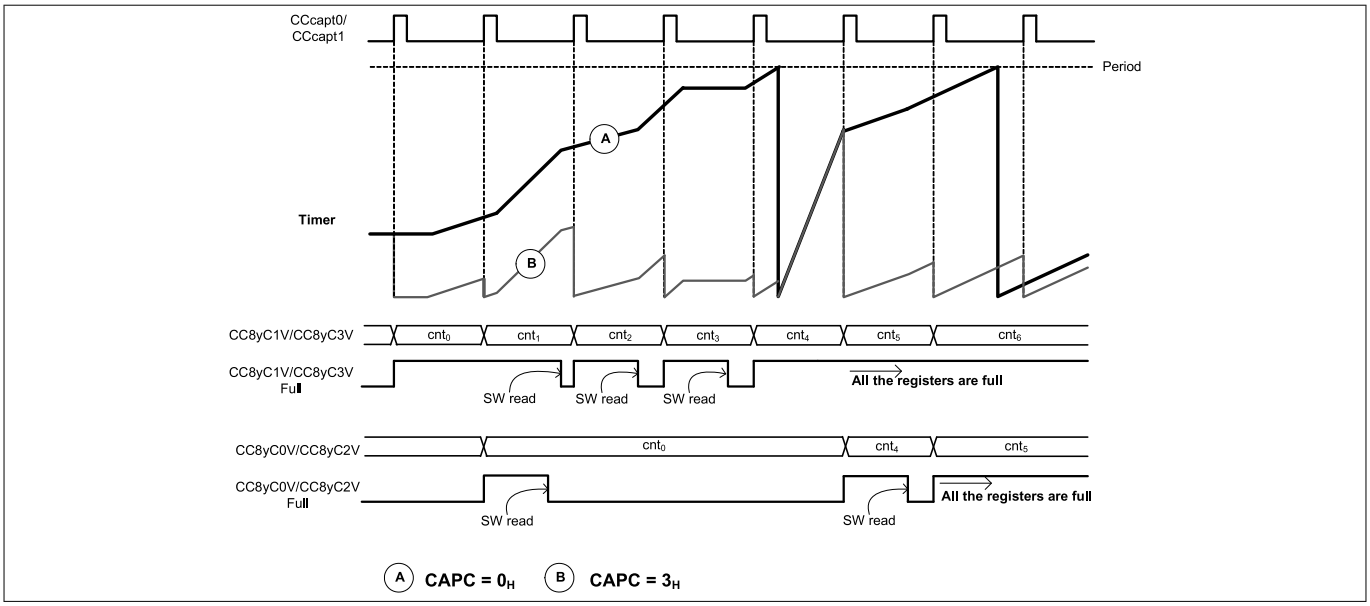


Figure 382 Capture mode usage - single channel

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 383. 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, for example, 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.

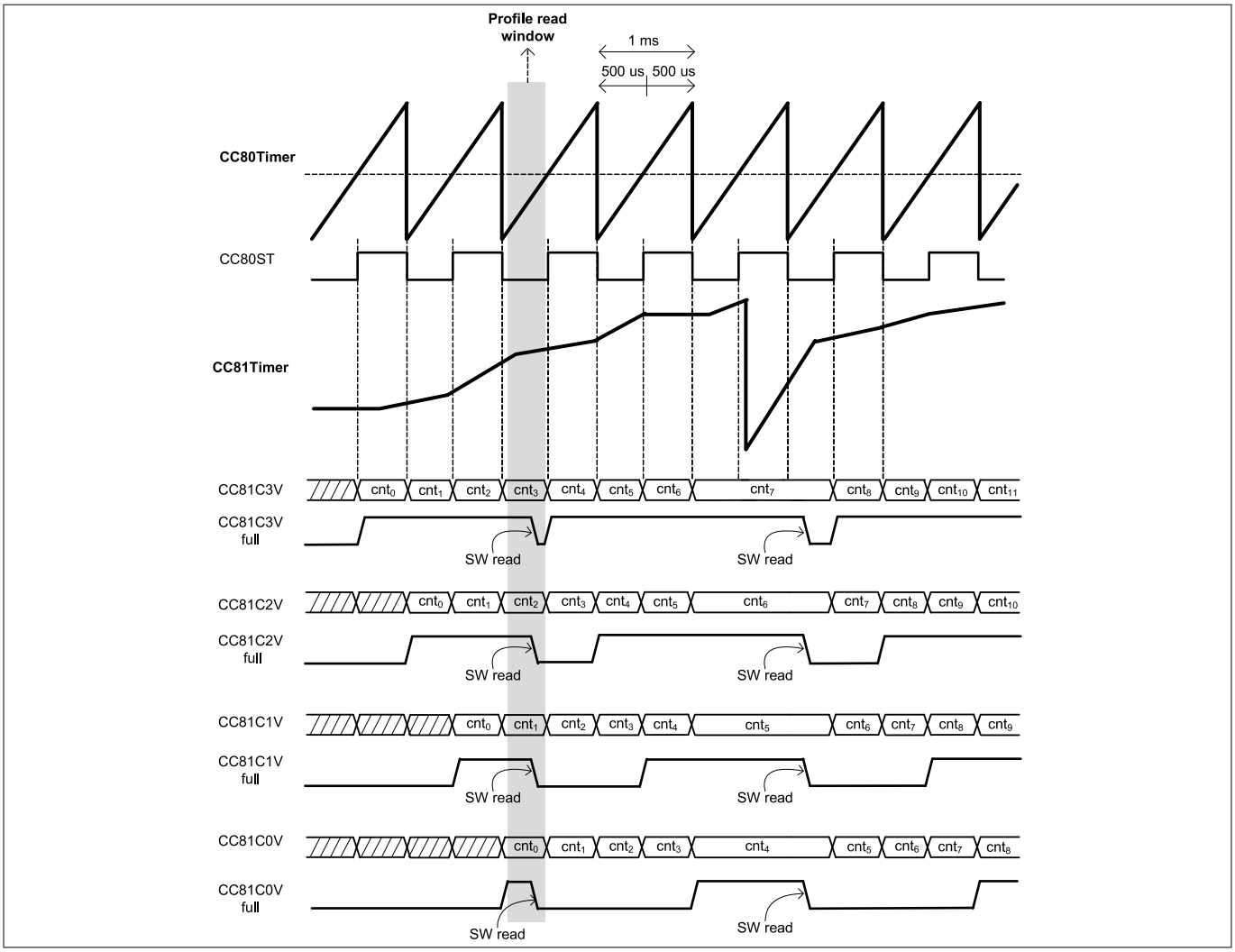


Figure 383 Three Capture profiles - CC8yTC.SCE = 1_B

To calculate the three different profiles in Figure 383, 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.*

High Dynamics Capturing

In some cases the dynamics of the capture trigger(s) may vary greatly over time. This will impose that the software needs to be prepared for the worst case scenario, where the frequency of the capture triggers may be very high. In applications where cycle-by-cycle calculation is needed (calculation in each capture trigger), then this constraints needs to be met by the software. Nevertheless for applications where a cycle-by-cycle calculation is not needed, the software can read back the FIFO data register in a periodic base and fetch all the data that has been captured so far, Figure 384.

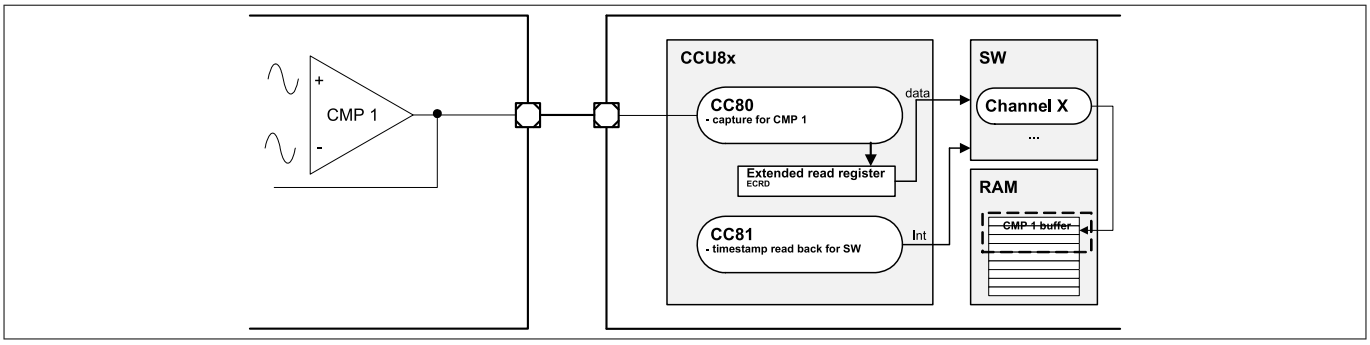


Figure 384 High dynamics capturing with software controlled timestamp

In this scenario, the software will read back the complete set of capture registers (2 or 4 depending on the chosen configuration), every time that an interrupt is triggered from the timestamp timer (the periodicity of this timer can also be adjusted on-the-fly).

Due to the fact that every capture register offers a full flag status bit, the software can always read back the complete set of registers. At the time of the data processing, this full flag is then checked, indicating if this value needs to be processed or not.

This FIFO read back functionality can also be used for applications that impose a heavy load on the system, which may not guarantee fixed access times to read back the captured data.

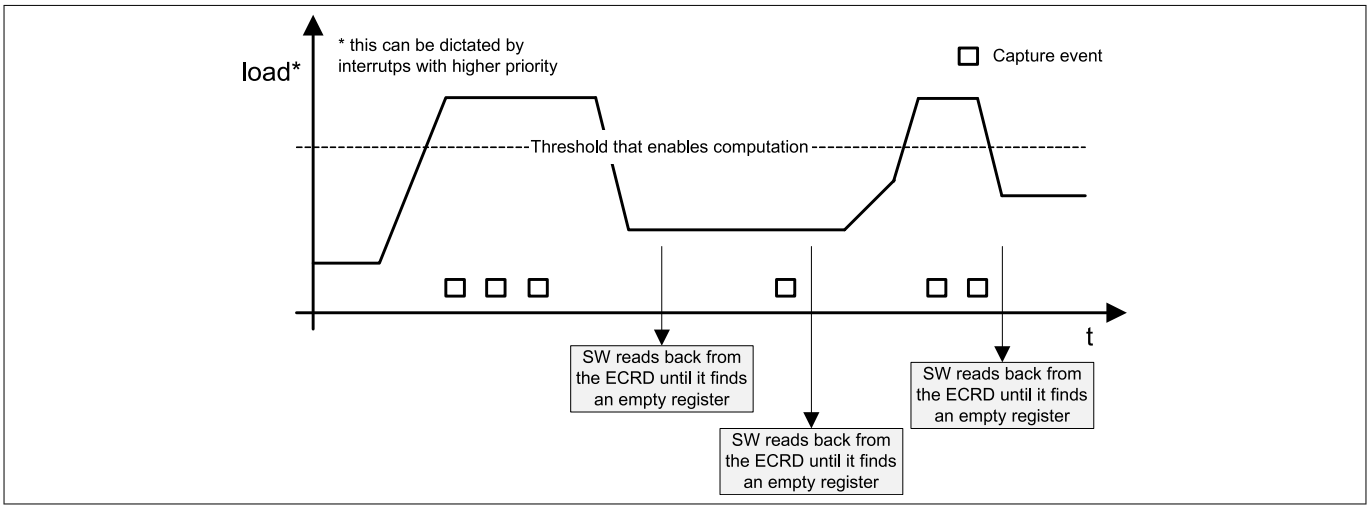


Figure 385 Extended read back during high load

Capture Grouping

In applications where multiple capture Timers are needed and the priority of the capture routines, does not imply that a cycle-by-cycle calculation needs to be done for every event, it may be suitable to group all the timers in the same CCU4x unit, Figure 386.

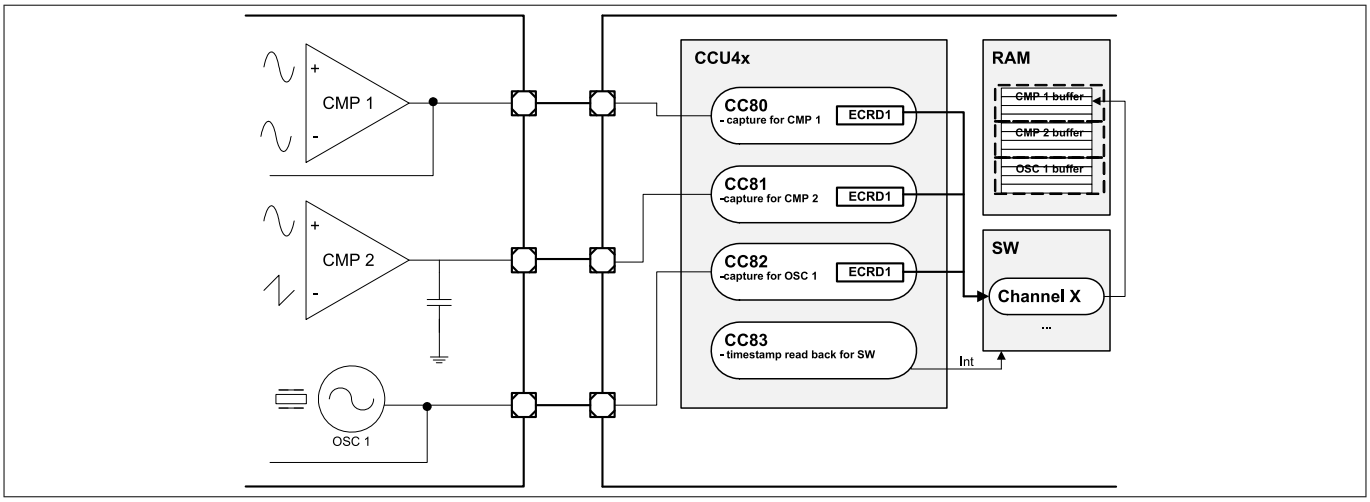


Figure 386 Capture grouping with extended read back

By setting the ECM bitfield for the TImer Slices used for capturing, the extended read back mode enables the reading back of data always in the proper capture order (from oldest to newest data). A timestamp timer is then used to trigger the CPU/Software to read back all the capture data present in the Timer Slices.

Every time that the interrupt is sensed, the CPU/Software (in this example) reads back the complete set of capture registers (via the ECRD address) for all Timer Slices. Due to the fact that each data read has a full flag indicator, the CPU/Software can read back the complete set of capture registers from all timers. This allows a fixed memory allocation that is as big as the number of captured registers, Figure 387 (in this example 4 capture registers for each Timer Slice are being used).

The additional lost value bitfield (LCV) on the header of each ECRD data, will indicate if any capture trigger was lost between read operations (this can happen if the capture triggers are faster than the routine that is reading back the values).

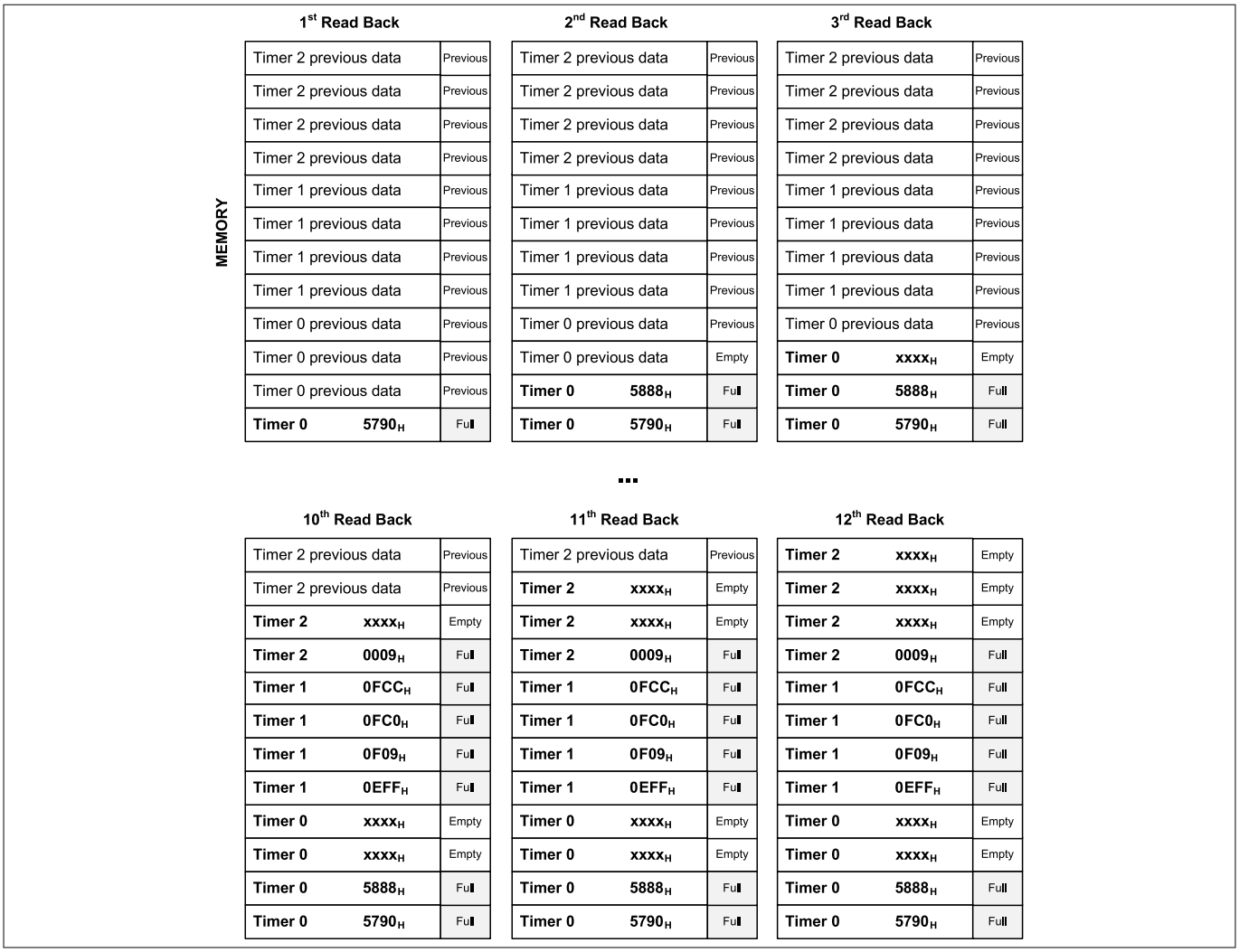


Figure 387 Memory structure for extended read back

22.2.8.5 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 388.

Figure 389 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).

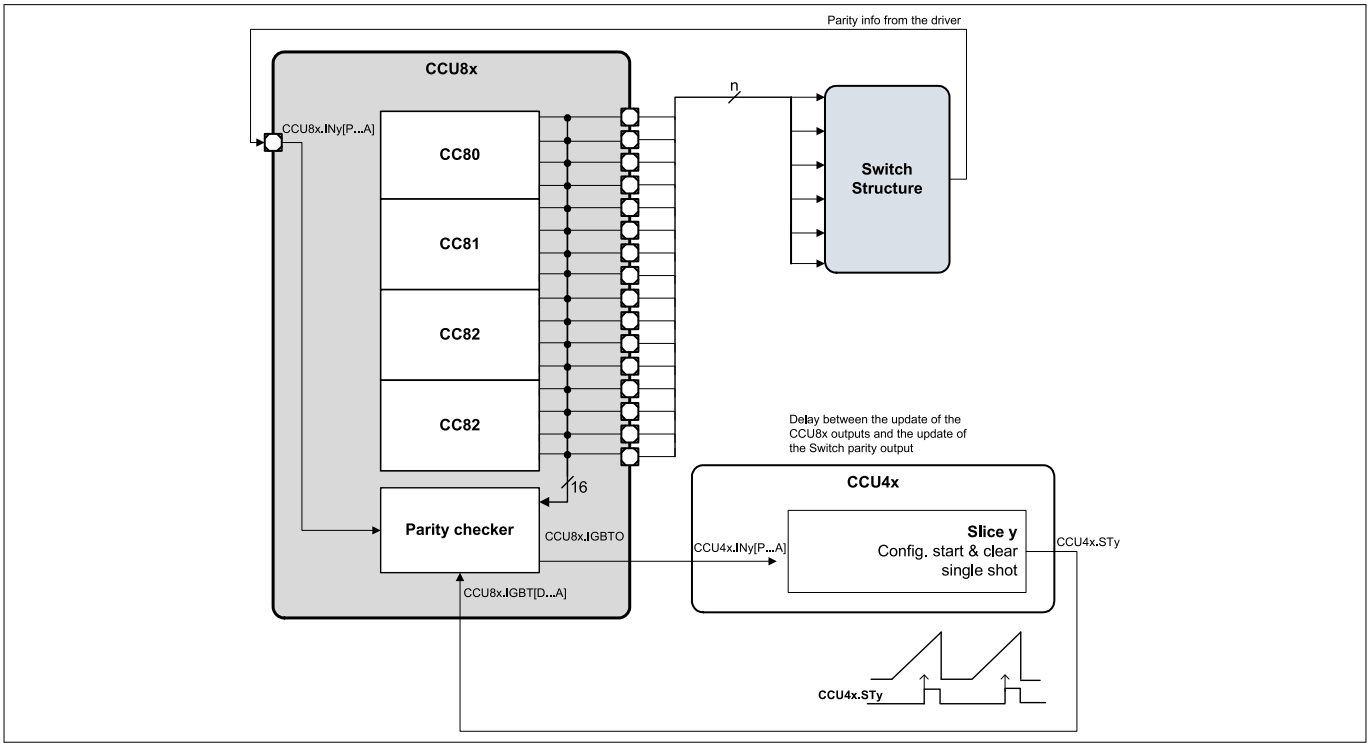


Figure 388 Parity Checker connections

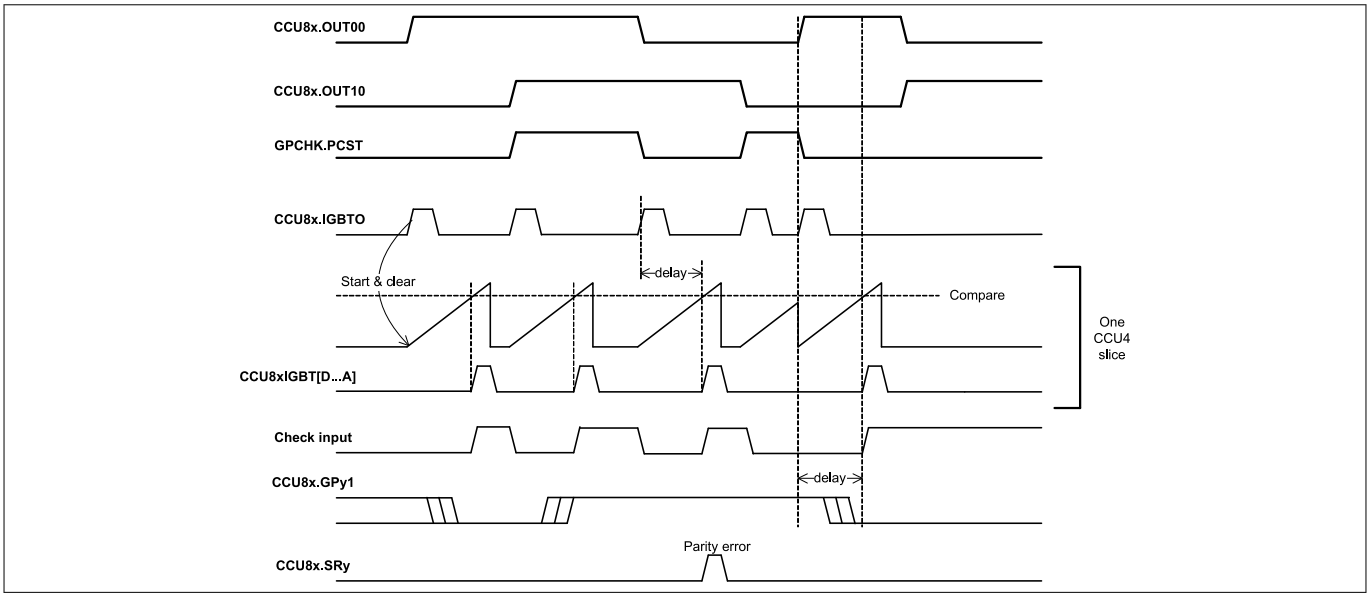


Figure 389 Parity Checker timing example

22.3 Service Request Generation

Each CCU8 slice has a interrupt structure as the one depicted in Figure 390. 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 185 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 **CC8yINS2**.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 **CC8yINS2**.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 185 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 391. The value set on the **CC8ySRS** controls which interrupt event is mapped into which service request line.

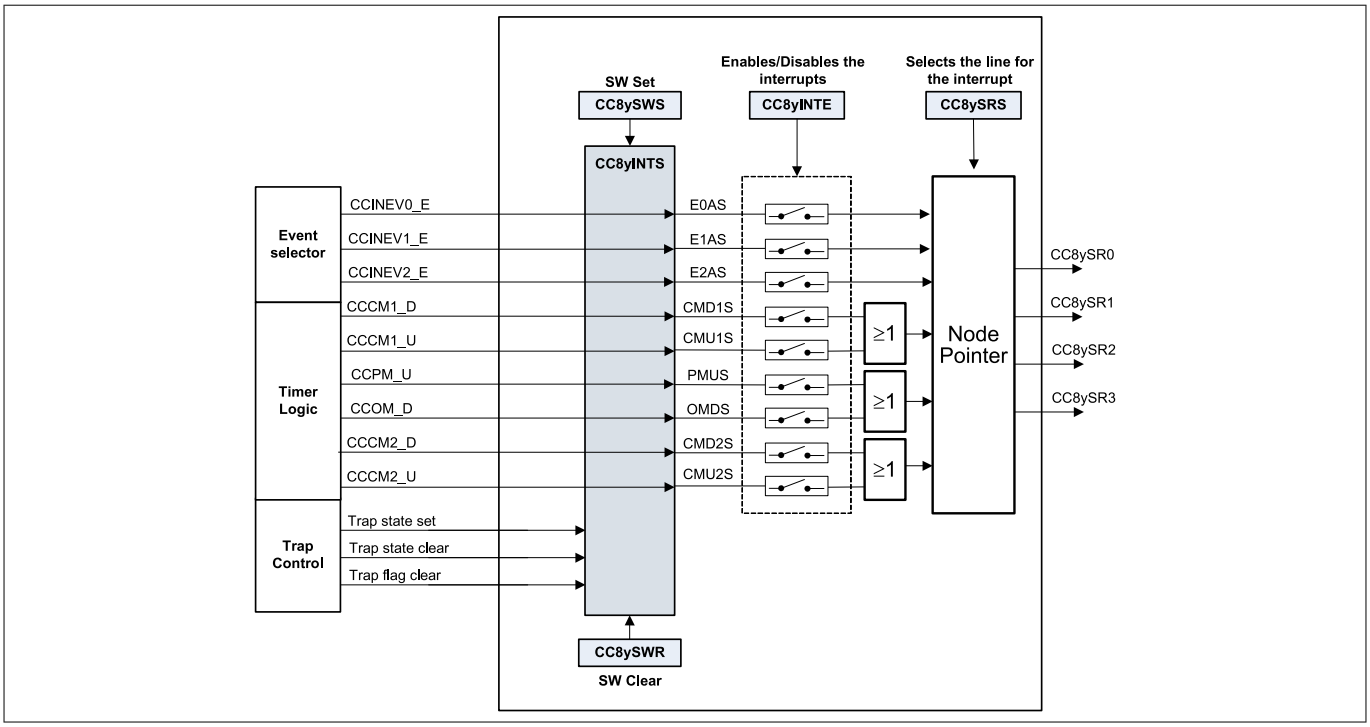


Figure 390 Slice interrupt node pointer overview

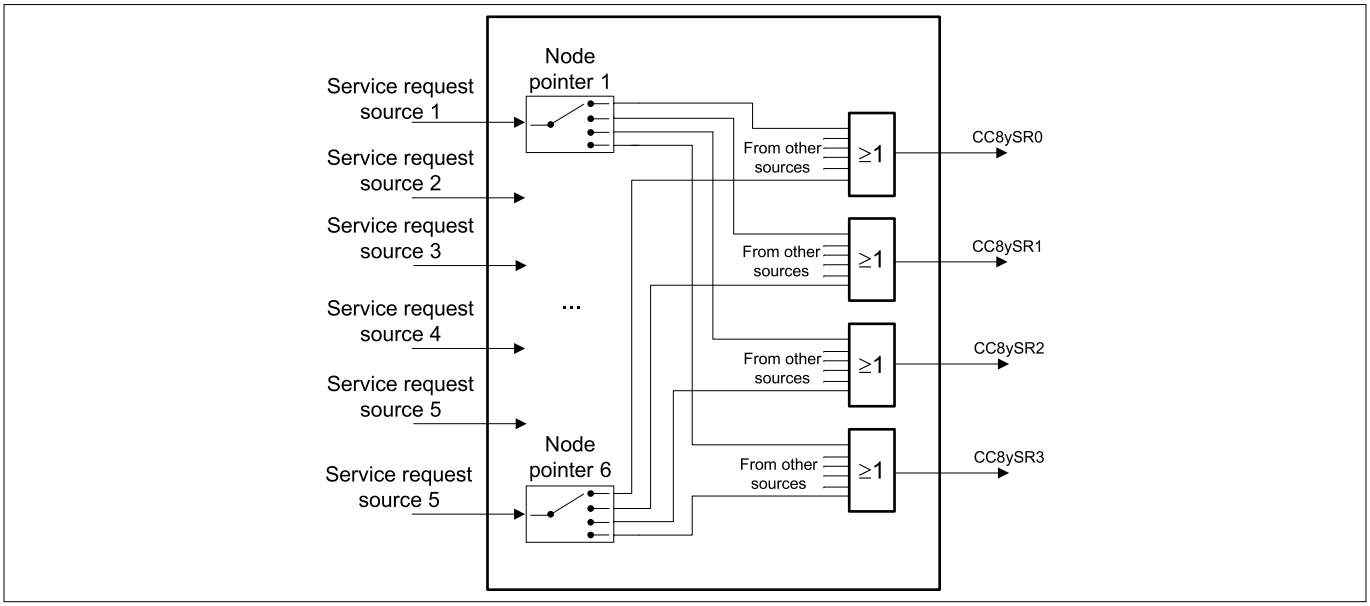


Figure 391 Slice interrupt selector overview

The four service request lines of each slice are OR together inside the kernel of the CCU8, see Figure 392. This means that there are only four service request lines per CCU8, that can have in each line interrupt requests coming from different slices.

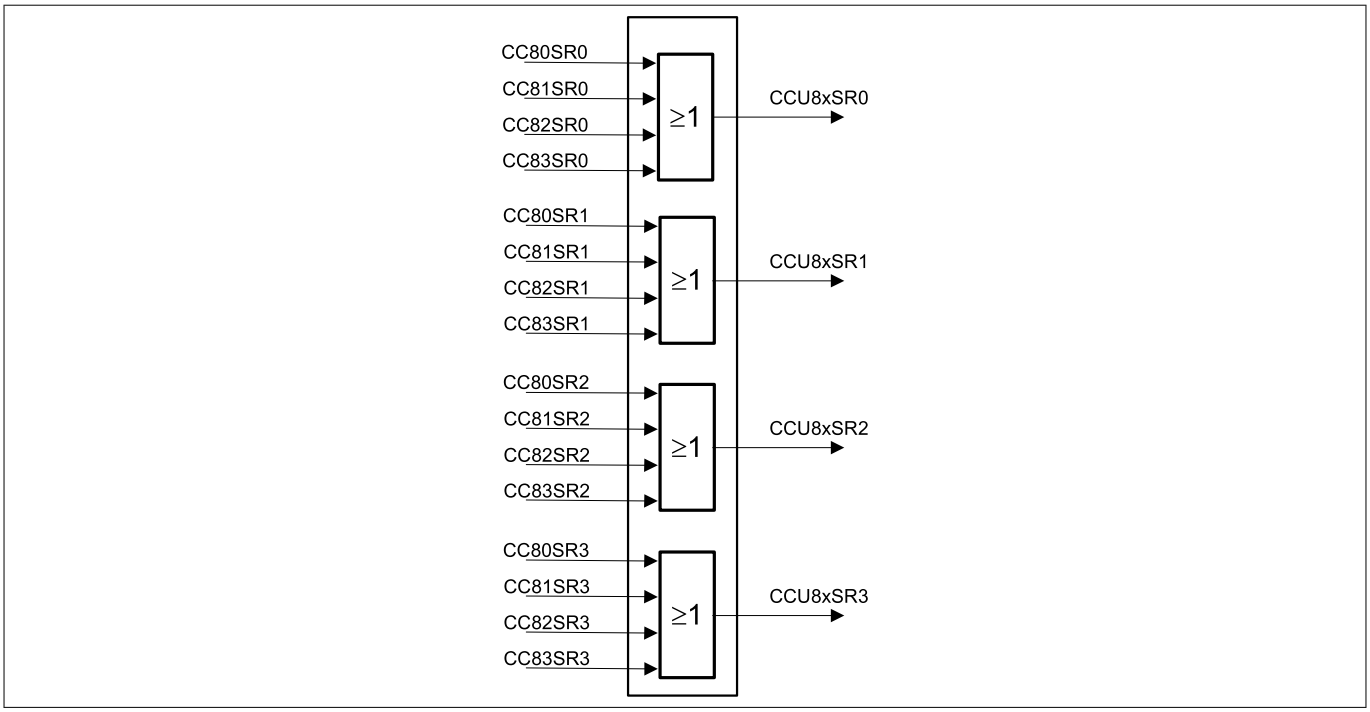


Figure 392 CCU8 service request overview

22.4 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, for example,, 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.

22.5 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}.

22.5.1 Clocks

Module Clock

The module clock of the CCU8 module is described in the SCU chapter as f_{PCLK}.

The bus interface clock of the CCU8 module is described in the SCU chapter as f_{MCLK}.

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 186 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

22.5.2 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**).

22.6 Initialization and System Dependencies

22.6.1 Initialization Sequence

The initialization sequence for an application that is using the CCU8, should be the following:

1st Step: Enable the CCU8 clock via the specific SCU register, CGATCLR0.

2nd Step: Enable the prescaler block, by writing 1_B to the **GIDLC**.SPRB field.

3rd Step: Configure the global CCU8 register **GCTRL**.

4th Step: Configure all the registers related to the required Timer Slice(s) functions, including the interrupt/ service request configuration.

5th 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.

6th Step: Configure the parity checker function if used, by programming the **GPCHK** register

7th Step: Enable the specific timer slice(s), CC8y, by writing 1_B to the specific **GIDLC**.CSyI.

8th 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 External Start/Stop.

22.6.2 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 Clocks
- 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.

22.7 Registers

Registers Overview

The absolute register address is calculated by adding:
Module Base Address + Offset Address

Table 187 Registers Address Space

Module	Base Address	End Address	Note
CCU80	50000000_H	$50003FFF_H$	
CCU81	50004000_H	$50007FFF_H$	

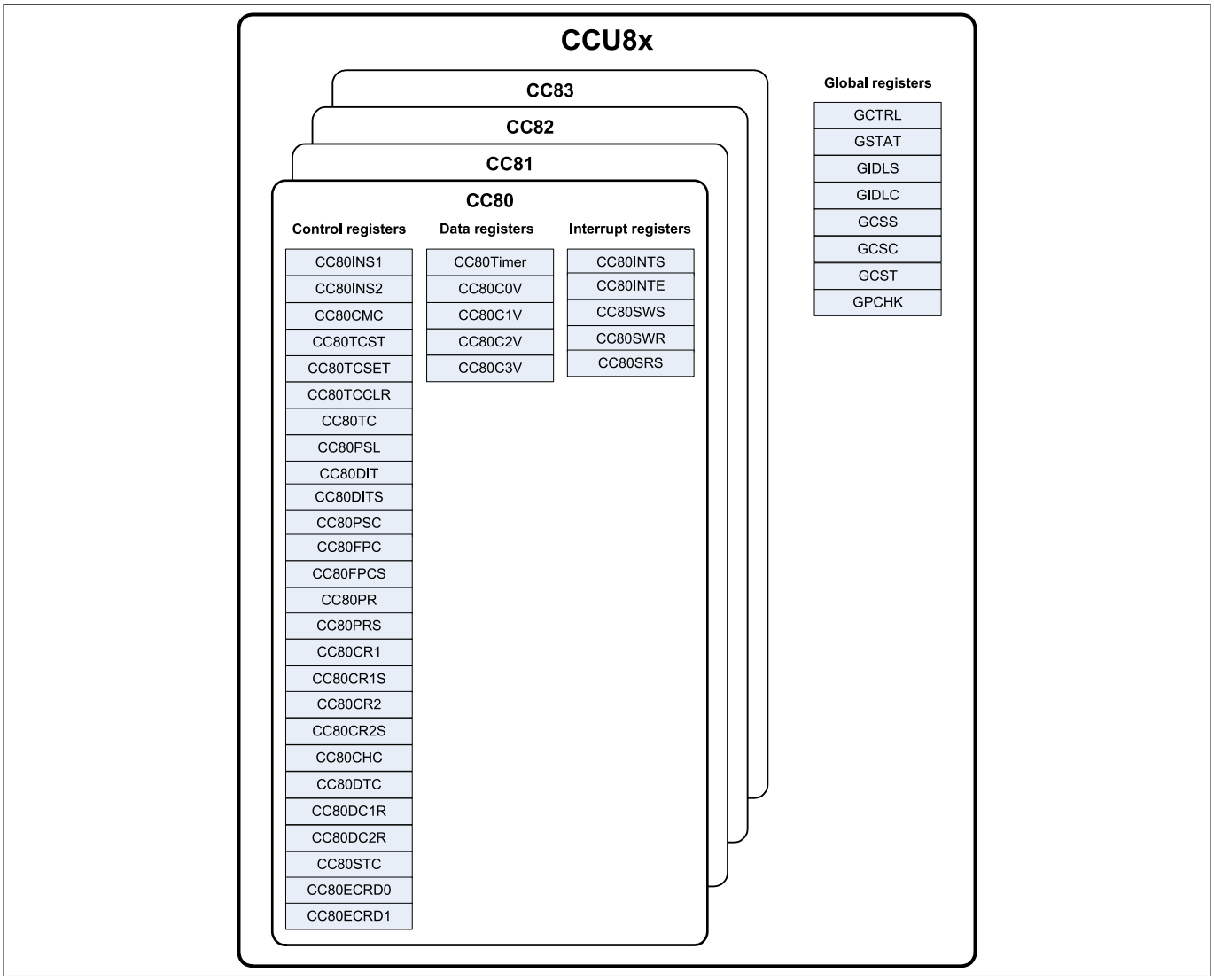


Figure 393 CCU8 registers overview

Table 188 Register Overview of CCU8

Register Name	Description	Offset Addr.[1)]	Access Mode	
			Read	Write
CCU8 Global Registers				
GCTRL	Module General Control Register	0000_H	U, PV	U, PV
GSTAT	General Slice Status Register	0004_H	U, PV	BE
GIDLS	General Idle Enable Register	0008_H	U, PV	U, PV
GIDLC	General Idle Disable Register	$000C_H$	U, PV	U, PV
GCSS	General Channel Set Register	0010_H	U, PV	U, PV
GCSC	General Channel Clear Register	0014_H	U, PV	U, PV
GCST	General Channel Status Register	0018_H	U, PV	BE
GPCHK	Parity Checker Configuration Register	$001C_H$	U, PV	U, PV
MIDR	Module Identification Register	0080_H	U, PV	BE

(table continues...)

Table 188 (continued) Register Overview of CCU8

Register Name	Description	Offset Addr.[1)]	Access Mode	
			Read	Write
CC8y Registers, y = 0...3, n = y + 1				
CC8yINS1	Input Selector Unit Configuration Register 1	$0ND8_H$	U, PV	U, PV
CC8yINS2	Input Selector Unit Configuration Register 2	$0N00_H$	U, PV	U, PV
CC8yCMC	Connection Matrix Configuration	$0N04_H$	U, PV	U, PV
CC8yTCST	Timer Run Status	$0N08_H$	U, PV	BE
CC8yTCSET	Timer Run Set	$0N0C_H$	U, PV	U, PV
CC8yTCCLR	Timer Run Clear	$0N10_H$	U, PV	U, PV
CC8yTC	General Timer Configuration	$0N14_H$	U, PV	U, PV
CC8yPSL	Output Passive Level Configuration	$0N18_H$	U, PV	U, PV
CC8yDIT	Dither Configuration	$0N1C_H$	U, PV	BE
CC8yDITS	Dither Shadow Register	$0N20_H$	U, PV	U, PV
CC8yPSC	Prescaler Configuration	$0N24_H$	U, PV	U, PV
CC8yFPC	Prescaler Compare Value	$0N28_H$	U, PV	U, PV
CC8yFPCS	Prescaler Shadow Compare Value	$0N2C_H$	U, PV	U, PV
CC8yPR	Timer Period Value	$0N30_H$	U, PV	BE
CC8yPRS	Timer Period Shadow Value	$0N34_H$	U, PV	U, PV
CC8yCR1	Timer Compare Value for Channel 1	$0N38_H$	U, PV	BE
CC8yCR1S	Timer Compare Shadow Value for Channel 1	$0N3C_H$	U, PV	U, PV
CC8yCR2	Timer Compare Value for Channel 2	$0N40_H$	U, PV	BE
CC8yCR2S	Timer Compare Shadow Value for Channel 2	$0N44_H$	U, PV	U, PV
CC8yCHC	Channel Control	$0N48_H$	U, PV	U, PV
CC8yDTC	Dead Time Control	$0N4C_H$	U, PV	U, PV
CC8yDC1R	Channel 1 Dead Time Counter Values	$0N50_H$	U, PV	U, PV
CC8yDC2R	Channel 2 Dead Time Counter Values	$0N54_H$	U, PV	U, PV
CC8yTIMER	Timer Current Value	$0N70_H$	U, PV	U, PV
CC8yC0V	Capture Register 0 Value	$0N74_H$	U, PV	BE
CC8yC1V	Capture Register 1 Value	$0N78_H$	U, PV	BE
CC8yC2V	Capture Register 2 Value	$0N7C_H$	U, PV	BE
CC8yC3V	Capture Register 3 Value	$0N80_H$	U, PV	BE
CC8yINTS	Interrupt Status	$0NA0_H$	U, PV	BE
CC8yINTE	Interrupt Enable	$0NA4_H$	U, PV	U, PV
CC8ySRS	Interrupt Configuration	$0NA8_H$	U, PV	U, PV
CC8ySWS	Interrupt Status Set	$0NAC_H$	U, PV	U, PV
CC8ySWR	Interrupt Status Clear	$0NB0_H$	U, PV	U, PV
CC8ySTC	Shadow Transfer Control	$0NB4_H$	U, PV	U, PV

(table continues...)

Table 188 (continued) Register Overview of CCU8

Register Name	Description	Offset Addr.[1]	Access Mode	
			Read	Write
CC8yECRD0	Extended Read Back 0	$0NB8_H$	U, PV	BE
CC8yECRD1	Extended Read Back 1	$0NBC_H$	U, PV	BE

1) The absolute register address is calculated as follows:
Module Base Address + Offset Address (shown in this column)

22.8 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.

22.8.1 CCU80 Pins

Table 189 CCU80 Pin Connections

Global Input/Output	I/O	Connected To	Description
CCU80.MCLK	I	PCLK	Kernel clock
CCU80.CLKA	I	ERU1.IOUT0	another count source for the prescaler
CCU80.CLKB	I	ERU0.IOUT0	another count source for the prescaler
CCU80.CLKC	I	ERU0.IOUT1	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	CCU40.ST0	Parity Checker delay finish trigger
CCU80.IGBTD	I	CCU40.SR0	Parity Checker delay finish trigger
CCU80.IGBTO	O	CCU40.IN3AH;	Parity Checker delay start trigger
CCU80.SR0	O	NVIC;	Service request line
CCU80.SR1	O	NVIC; POSIF0.MSETE;	Service request line
CCU80.SR2	O	VADC0.BGREQTRI; VADC0.G0REQTRI; VADC0.G1REQTRI; ERU0.OGU03; ERU0.OGU13;	Service request line

(table continues...)

Table 189 (continued) CCU80 Pin Connections

Global Input/Output	I/O	Connected To	Description
CCU80.SR3	O	VADC0.BGREQTRJ; VADC0.G0REQTRJ; VADC0.G1REQTRJ; ERU0.OGU23; ERU0.OGU33;	Service request line

Table 190 CCU80 - CC80 Pin Connections

Input/Output	I/O	Connected To	Description
CCU80.IN0AA	I	P0.12	General purpose function
CCU80.IN0AB	I	P0.4	General purpose function
CCU80.IN0AC	I	CCU40.GP02	General purpose function
CCU80.IN0AD	I	POSIF0.OUT2	General purpose function
CCU80.IN0AE	I	POSIF0.OUT5	General purpose function
CCU80.IN0AF	I	ERU0.PDOUT0	General purpose function
CCU80.IN0AG	I	ERU0.IOUT0	General purpose function
CCU80.IN0AH	I	SCU.GSC80	General purpose function
CCU80.IN0AI	I	BCCU0.OUT0	General purpose function
CCU80.IN0AJ	I	BCCU0.OUT1	General purpose function
CCU80.IN0AK	I	CCU40.SR2	General purpose function
CCU80.IN0AL	I	ERU0.PDOUT1	General purpose function
CCU80.IN0AM	I	CCU80.GP10	General purpose function
CCU80.IN0AN	I	CCU80.ST1	General purpose function
CCU80.IN0AO	I	CCU80.ST2	General purpose function
CCU80.IN0AP	I	CCU80.ST3	General purpose function
CCU80.IN0AQ	I	BCCU0.OUT2	General purpose function
CCU80.IN0AR	I	ACMP1.OUT	General purpose function
CCU80.IN0AS	I	ACMP2.OUT	General purpose function
CCU80.IN0AT	I	CCU80.ST0	General purpose function
CCU80.IN0AU	I	–	General purpose function
CCU80.IN0AV	I	ERU1.PDOUT0	General purpose function
CCU80.IN0AW	I	ERU1.IOUT0	General purpose function
CCU80.IN0AX	I	ERU1.PDOUT1	General purpose function
CCU80.IN0AY	I	CCU80.ST3B	Reserved
CCU80.IN0AZ	I	0	Reserved
CCU80.IN0BA	I	0	Reserved

(table continues...)

Table 190 (continued) CCU80 - CC80 Pin Connections

Input/Output	I/O	Connected To	Description
CCU80.IN0BB	I	0	Reserved
CCU80.IN0BC	I	0	Reserved
CCU80.IN0BD	I	0	Reserved
CCU80.IN0BE	I	0	Reserved
CCU80.IN0BF	I	0	Reserved
CCU80.IN0BG	I	0	Reserved
CCU80.IN0BH	I	0	Reserved
CCU80.IN0BI	I	0	Reserved
CCU80.IN0BJ	I	0	Reserved
CCU80.IN0BK	I	0	Reserved
CCU80.IN0BL	I	0	Reserved
CCU80.IN0BM	I	0	Reserved
CCU80.IN0BN	I	0	Reserved
CCU80.IN0BO	I	0	Reserved
CCU80.IN0BP	I	0	Reserved
CCU80.IN0BQ	I	0	Reserved
CCU80.IN0BR	I	0	Reserved
CCU80.IN0BS	I	0	Reserved
CCU80.IN0BT	I	0	Reserved
CCU80.IN0BU	I	0	Reserved
CCU80.IN0BV	I	0	Reserved
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	P0.0; P1.0; P4.4;	Slice compare output from channel 1. Can be the CCST1, NOT(CCST1), CCST2 or NOT(CCST2) path
CCU80.OUT01	O	P0.1; P0.5; P1.1; P4.5;	Slice compare output from channel 1. Can be the CCST1, NOT(CCST1), CCST2 or NOT(CCST2) path

(table continues...)

Table 190 (continued) CCU80 - CC80 Pin Connections

Input/Output	I/O	Connected To	Description
CCU80.OUT02	O	P0.2	Slice compare output from channel 1. Can be the CCST1, NOT(CCST1), CCST2 or NOT(CCST2) path
CCU80.OUT03	O	P0.3	Slice compare output from channel 1. Can be the CCST1, NOT(CCST1), CCST2 or NOT(CCST2) path
CCU80.GP00	O	CCU80.IN3AM	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	CCU80.IN0AT; CCU80.IN1AN; CCU80.IN2AN; CCU80.IN3AN; VADC0.BGREQGTM; VADC0.G0REQGTM; VADC0.G1REQGTM; ERU1.0B2;	Output of the status bit multiplexor. It can be CCST1 or CCST2
CCU80.ST0A	O	not connected	Channel 1 status bit: CCST1
CCU80.ST0B	O	CCU80.IN1AY	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 191 CCU80 - CC81 Pin Connections

Input/Output	I/O	Connected To	Description
CCU80.IN1AA	I	P0.12	General purpose function
CCU80.IN1AB	I	P0.5;	General purpose function
CCU80.IN1AC	I	CCU40.GP12	General purpose function
CCU80.IN1AD	I	POSIF0.OUT2	General purpose function
CCU80.IN1AE	I	POSIF0.OUT5	General purpose function
CCU80.IN1AF	I	ERU0.PDOUT1	General purpose function
CCU80.IN1AG	I	ERU0.IOUT1	General purpose function
CCU80.IN1AH	I	SCU.GSC80	General purpose function
CCU80.IN1AI	I	BCCU0.OUT2	General purpose function
CCU80.IN1AJ	I	BCCU0.OUT3	General purpose function
CCU80.IN1AK	I	CCU40.SR2	General purpose function
CCU80.IN1AL	I	ERU0.PDOUT0	General purpose function

(table continues...)

Table 191 (continued) CCU80 - CC81 Pin Connections

Input/Output	I/O	Connected To	Description
CCU80.IN1AM	I	CCU80.GP20	General purpose function
CCU80.IN1AN	I	CCU80.ST0	General purpose function
CCU80.IN1AO	I	CCU80.ST2	General purpose function
CCU80.IN1AP	I	CCU80.ST3	General purpose function
CCU80.IN1AQ	I	BCCU0.OUT7	General purpose function
CCU80.IN1AR	I	ACMP2.OUT	General purpose function
CCU80.IN1AS	I	ACMP1.OUT	General purpose function
CCU80.IN1AT	I	CCU80.ST1	General purpose function
CCU80.IN1AU	I	P4.1	General purpose function
CCU80.IN1AV	I	ERU1.PDOUT1	General purpose function
CCU80.IN1AW	I	ERU1.IOUT1	General purpose function
CCU80.IN1AX	I	ERU1.PDOUT0	General purpose function
CCU80.IN1AY	I	CCU80.ST0B	Reserved
CCU80.IN1AZ	I	0	Reserved
CCU80.IN1BA	I	0	Reserved
CCU80.IN1BB	I	0	Reserved
CCU80.IN1BC	I	0	Reserved
CCU80.IN1BD	I	0	Reserved
CCU80.IN1BE	I	0	Reserved
CCU80.IN1BF	I	0	Reserved
CCU80.IN1BG	I	0	Reserved
CCU80.IN1BH	I	0	Reserved
CCU80.IN1BI	I	0	Reserved
CCU80.IN1BJ	I	0	Reserved
CCU80.IN1BK	I	0	Reserved
CCU80.IN1BL	I	0	Reserved
CCU80.IN1BM	I	0	Reserved
CCU80.IN1BN	I	0	Reserved
CCU80.IN1BO	I	0	Reserved
CCU80.IN1BP	I	0	Reserved
CCU80.IN1BQ	I	0	Reserved
CCU80.IN1BR	I	0	Reserved
CCU80.IN1BS	I	0	Reserved

(table continues...)

Table 191 (continued) CCU80 - CC81 Pin Connections

Input/Output	I/O	Connected To	Description
CCU80.IN1BT	I	0	Reserved
CCU80.IN1BU	I	0	Reserved
CCU80.IN1BV	I	0	Reserved
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
CCU80.MCI13	I	POSIF0.MOUT[7]	Multi Channel pattern input for NOT(CCST2)
CCU80.OUT10	O	P0.2; P0.7; P1.2; P4.6;	Slice compare output from channel 1. Can be the CCST1, NOT(CCST1), CCST2 or NOT(CCST2) path
CCU80.OUT11	O	P0.3; P0.6; P1.3; P4.7;	Slice compare output from channel 1. Can be the CCST1, NOT(CCST1), CCST2 or NOT(CCST2) path
CCU80.OUT12	O	P0.5;	Slice compare output from channel 1. Can be the CCST1, NOT(CCST1), CCST2 or NOT(CCST2) path
CCU80.OUT13	O	P0.4;	Slice compare output from channel 1. Can be the CCST1, NOT(CCST1), CCST2 or NOT(CCST2) path
CCU80.GP10	O	CCU80.IN0AM	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	CCU80.IN1AT; CCU80.IN0AN; CCU80.IN2AO; CCU80.IN3AO; VADC0.BGREQGTN; VADC0.G0REQGTN; VADC0.G1REQGTN; ERU1.2B2;	Output of the status bit multiplexor. It can be CCST1 or CCST2
CCU80.ST1A	O	not connected	Channel 1 status bit: CCST1
CCU80.ST1B	O	CCU80.IN2AY	Channel 2 status bit: CCST2

(table continues...)

Table 191 (continued) CCU80 - CC81 Pin Connections

Input/Output	I/O	Connected To	Description
CCU80.PS1	O	POSIF0.MSYNCA	Multi channel pattern sync trigger: PM when counting UP (edge aligned) or OM when counting DOWN (center aligned)

Table 192 CCU80 - CC82 Pin Connections

Input/Output	I/O	Connected To	Description
CCU80.IN2AA	I	P0.12	General purpose function
CCU80.IN2AB	I	P0.10;	General purpose function
CCU80.IN2AC	I	CCU40.GP22	General purpose function
CCU80.IN2AD	I	POSIF0.OUT2	General purpose function
CCU80.IN2AE	I	POSIF0.OUT5	General purpose function
CCU80.IN2AF	I	ERU0.PDOUT2	General purpose function
CCU80.IN2AG	I	ERU0.IOUT2	General purpose function
CCU80.IN2AH	I	SCU.GSC80	General purpose function
CCU80.IN2AI	I	BCCU0.OUT4	General purpose function
CCU80.IN2AJ	I	BCCU0.OUT5	General purpose function
CCU80.IN2AK	I	CCU40.SR3	General purpose function
CCU80.IN2AL	I	ERU0.PDOUT0	General purpose function
CCU80.IN2AM	I	CCU80.GP30	General purpose function
CCU80.IN2AN	I	CCU80.ST0	General purpose function
CCU80.IN2AO	I	CCU80.ST1	General purpose function
CCU80.IN2AP	I	CCU80.ST3	General purpose function
CCU80.IN2AQ	I	BCCU0.OUT1	General purpose function
CCU80.IN2AR	I	ACMP0.OUT	General purpose function
CCU80.IN2AS	I	ACMP3.OUT	General purpose function
CCU80.IN2AT	I	CCU80.ST2	General purpose function
CCU80.IN2AU	I	P4.2	General purpose function
CCU80.IN2AV	I	ERU1.PDOUT2	General purpose function
CCU80.IN2AW	I	ERU1.IOUT2	General purpose function
CCU80.IN2AX	I	ERU1.PDOUT0	General purpose function
CCU80.IN2AY	I	CCU80.ST1B	Reserved
CCU80.IN2AZ	I	0	Reserved
CCU80.IN2BA	I	0	Reserved
CCU80.IN2BB	I	0	Reserved
CCU80.IN2BC	I	0	Reserved

(table continues...)

Table 192 (continued) CCU80 - CC82 Pin Connections

Input/Output	I/O	Connected To	Description
CCU80.IN2BD	I	0	Reserved
CCU80.IN2BE	I	0	Reserved
CCU80.IN2BF	I	0	Reserved
CCU80.IN2BG	I	0	Reserved
CCU80.IN2BH	I	0	Reserved
CCU80.IN2BI	I	0	Reserved
CCU80.IN2BJ	I	0	Reserved
CCU80.IN2BK	I	0	Reserved
CCU80.IN2BL	I	0	Reserved
CCU80.IN2BM	I	0	Reserved
CCU80.IN2BN	I	0	Reserved
CCU80.IN2BO	I	0	Reserved
CCU80.IN2BP	I	0	Reserved
CCU80.IN2BQ	I	0	Reserved
CCU80.IN2BR	I	0	Reserved
CCU80.IN2BS	I	0	Reserved
CCU80.IN2BT	I	0	Reserved
CCU80.IN2BU	I	0	Reserved
CCU80.IN2BV	I	0	Reserved
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	P0.8; P0.12; P1.4; P2.0;	Slice compare output from channel 1. Can be the CCST1, NOT(CCST1), CCST2 or NOT(CCST2) path
CCU80.OUT21	O	P0.9; P0.13; P1.5; P2.1; P3.0;	Slice compare output from channel 1. Can be the CCST1, NOT(CCST1), CCST2 or NOT(CCST2) path

(table continues...)

Table 192 (continued) CCU80 - CC82 Pin Connections

Input/Output	I/O	Connected To	Description
CCU80.OUT22	O	P0.10;	Slice compare output from channel 1. Can be the CCST1, NOT(CCST1), CCST2 or NOT(CCST2) path
CCU80.OUT23	O	P0.11;	Slice compare output from channel 1. Can be the CCST1, NOT(CCST1), CCST2 or NOT(CCST2) path
CCU80.GP20	O	CCU80.IN1AM	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	CCU80.IN2AT; CCU80.IN0AO; CCU80.IN1AO; CCU80.IN3AP; ERU1.3B2;	Output of the status bit multiplexor. It can be CCST1 or CCST2
CCU80.ST2A	O	not connected	Channel 1 status bit: CCST1
CCU80.ST2B	O	CCU80.IN3AY	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 193 CCU80 - CC83 Pin Connections

Input/Output	I/O	Connected To	Description
CCU80.IN3AA	I	P0.12	General purpose function
CCU80.IN3AB	I	P0.13;	General purpose function
CCU80.IN3AC	I	CCU40.GP32	General purpose function
CCU80.IN3AD	I	POSIF0.OUT2	General purpose function
CCU80.IN3AE	I	POSIF0.OUT5	General purpose function
CCU80.IN3AF	I	ERU0.PDOUT3	General purpose function
CCU80.IN3AG	I	ERU0.IOUT3	General purpose function
CCU80.IN3AH	I	SCU.GSC80	General purpose function
CCU80.IN3AI	I	BCCU0.OUT6	General purpose function
CCU80.IN3AJ	I	BCCU0.OUT7	General purpose function
CCU80.IN3AK	I	CCU40.SR3	General purpose function
CCU80.IN3AL	I	ERU0.PDOUT0	General purpose function
CCU80.IN3AM	I	CCU80.GP00	General purpose function
CCU80.IN3AN	I	CCU80.ST0	General purpose function

(table continues...)

Table 193 (continued) CCU80 - CC83 Pin Connections

Input/Output	I/O	Connected To	Description
CCU80.IN3AO	I	CCU80.ST1	General purpose function
CCU80.IN3AP	I	CCU80.ST2	General purpose function
CCU80.IN3AQ	I	BCCU0.OUT8	General purpose function
CCU80.IN3AR	I	ACMP3.OUT	General purpose function
CCU80.IN3AS	I	ACMP0.OUT	General purpose function
CCU80.IN3AT	I	CCU80.ST3	General purpose function
CCU80.IN3AU	I	P4.3	General purpose function
CCU80.IN3AV	I	ERU1.PDOUT3	General purpose function
CCU80.IN3AW	I	ERU1.IOUT3	General purpose function
CCU80.IN3AX	I	ERU1.PDOUT0	General purpose function
CCU80.IN3AY	I	CCU80.ST2B	Reserved
CCU80.IN3AZ	I	0	Reserved
CCU80.IN3BA	I	0	Reserved
CCU80.IN3BB	I	0	Reserved
CCU80.IN3BC	I	0	Reserved
CCU80.IN3BD	I	0	Reserved
CCU80.IN3BE	I	0	Reserved
CCU80.IN3BF	I	0	Reserved
CCU80.IN3BG	I	0	Reserved
CCU80.IN3BH	I	0	Reserved
CCU80.IN3BI	I	0	Reserved
CCU80.IN3BJ	I	0	Reserved
CCU80.IN3BK	I	0	Reserved
CCU80.IN3BL	I	0	Reserved
CCU80.IN3BM	I	0	Reserved
CCU80.IN3BN	I	0	Reserved
CCU80.IN3BO	I	0	Reserved
CCU80.IN3BP	I	0	Reserved
CCU80.IN3BQ	I	0	Reserved
CCU80.IN3BR	I	0	Reserved
CCU80.IN3BS	I	0	Reserved
CCU80.IN3BT	I	0	Reserved
CCU80.IN3BU	I	0	Reserved

(table continues...)

Table 193 (continued) CCU80 - CC83 Pin Connections

Input/Output	I/O	Connected To	Description
CCU80.IN3BV	I	0	Reserved
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	P0.15; P2.10; P4.8;	Slice compare output from channel 1. Can be the CCST1, NOT(CCST1), CCST2 or NOT(CCST2) path
CCU80.OUT31	O	P0.14; P2.11; P4.9;	Slice compare output from channel 1. Can be the CCST1, NOT(CCST1), CCST2 or NOT(CCST2) path
CCU80.OUT32	O	P0.13;	Slice compare output from channel 1. Can be the CCST1, NOT(CCST1), CCST2 or NOT(CCST2) path
CCU80.OUT33	O	P0.12;	Slice compare output from channel 1. Can be the CCST1, NOT(CCST1), CCST2 or NOT(CCST2) path
CCU80.GP30	O	CCU80.IN2AM	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	CCU40.IN0AH; CCU80.IN0AP; CCU80.IN1AP; CCU80.IN2AP; CCU80.IN3AT; VADC0.BGREQGTF; VADC0.G0REQGTF; VADC0.G1REQGTF; ERU1.1B2;	Output of the status bit multiplexor. It can be CCST1 or CCST2
CCU80.ST3A	O	VADC0.BGREQGTE; VADC0.G0REQGTE; VADC0.G1REQGTE;	Channel 1 status bit: CCST1
CCU80.ST3B	O	CCU80.IN0AY	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)

22.8.2 CCU81 Pins

Table 194 CCU81 Pin Connections

Global Input/Output	I/O	Connected To	Description
CCU81.MCLK	I	PCLK	Kernel clock
CCU81.CLKA	I	ERU0.IOUT0	another count source for the prescaler
CCU81.CLKB	I	ERU1.IOUT0	another count source for the prescaler
CCU81.CLKC	I	ERU1.IOUT1	another count source for the prescaler
CCU81.MCSS	I	POSIF1.OUT6	Multi pattern sync with shadow transfer trigger
CCU81.IGBTA	I	CCU41.ST3	Parity Checker delay finish trigger
CCU81.IGBTB	I	CCU41.SR3	Parity Checker delay finish trigger
CCU81.IGBTC	I	CCU41.ST0	Parity Checker delay finish trigger
CCU81.IGBTD	I	CCU41.SR0	Parity Checker delay finish trigger
CCU81.IGBTO	O	CCU41.IN3AH;	Parity Checker delay start trigger
CCU81.SR0	O	NVIC; CCU81.IN0AQ;	Service request line
CCU81.SR1	O	NVIC; POSIF1.MSETE; CCU81.IN1AQ;	Service request line
CCU81.SR2	O	VADC0.BGREQTRK; VADC0.G0REQTRK; VADC0.G1REQTRK; ERU1.OGU03; ERU1.OGU13; CCU81.IN2AQ;	Service request line
CCU81.SR3	O	VADC0.BGREQTRL; VADC0.G0REQTRL; VADC0.G1REQTRL; ERU1.OGU23; ERU1.OGU33; CCU81.IN3AQ;	Service request line

Table 195 CCU81 - CC80 Pin Connections

Input/Output	I/O	Connected To	Description
CCU81.IN0AA	I	P3.0	General purpose function
CCU81.IN0AB	I	P4.6	General purpose function
CCU81.IN0AC	I	CCU41.GP02	General purpose function
CCU81.IN0AD	I	POSIF1.OUT2	General purpose function

(table continues...)

Table 195 (continued) CCU81 - CC80 Pin Connections

Input/Output	I/O	Connected To	Description
CCU81.IN0AE	I	POSIF1.OUT5	General purpose function
CCU81.IN0AF	I	ERU0.PDOUT0	General purpose function
CCU81.IN0AG	I	ERU0.IOUT0	General purpose function
CCU81.IN0AH	I	SCU.GSC81	General purpose function
CCU81.IN0AI	I	BCCU0.OUT0	General purpose function
CCU81.IN0AJ	I	BCCU0.OUT6	General purpose function
CCU81.IN0AK	I	CCU41.SR2	General purpose function
CCU81.IN0AL	I	ERU0.PDOUT1	General purpose function
CCU81.IN0AM	I	CCU81.GP10	General purpose function
CCU81.IN0AN	I	CCU81.ST1	General purpose function
CCU81.IN0AO	I	CCU81.ST2	General purpose function
CCU81.IN0AP	I	CCU81.ST3	General purpose function
CCU81.IN0AQ	I	CCU81.SR0	General purpose function
CCU81.IN0AR	I	ACMP1.OUT	General purpose function
CCU81.IN0AS	I	ACMP0.OUT	General purpose function
CCU81.IN0AT	I	CCU81.ST0	General purpose function
CCU81.IN0AU	I	P0.12	General purpose function
CCU81.IN0AV	I	ERU1.PDOUT0	General purpose function
CCU81.IN0AW	I	ERU1.IOUT0	General purpose function
CCU81.IN0AX	I	ERU1.PDOUT1	General purpose function
CCU81.IN0AY	I	CCU81.ST3B	Reserved
CCU81.IN0AZ	I	0	Reserved
CCU81.IN0BA	I	0	Reserved
CCU81.IN0BB	I	0	Reserved
CCU81.IN0BC	I	0	Reserved
CCU81.IN0BD	I	0	Reserved
CCU81.IN0BE	I	0	Reserved
CCU81.IN0BF	I	0	Reserved
CCU81.IN0BG	I	0	Reserved
CCU81.IN0BH	I	0	Reserved
CCU81.IN0BI	I	0	Reserved
CCU81.IN0BJ	I	0	Reserved
CCU81.IN0BK	I	0	Reserved

(table continues...)

Table 195 (continued) CCU81 - CC80 Pin Connections

Input/Output	I/O	Connected To	Description
CCU81.IN0BL	I	0	Reserved
CCU81.IN0BM	I	0	Reserved
CCU81.IN0BN	I	0	Reserved
CCU81.IN0BO	I	0	Reserved
CCU81.IN0BP	I	0	Reserved
CCU81.IN0BQ	I	0	Reserved
CCU81.IN0BR	I	0	Reserved
CCU81.IN0BS	I	0	Reserved
CCU81.IN0BT	I	0	Reserved
CCU81.IN0BU	I	0	Reserved
CCU81.IN0BV	I	0	Reserved
CCU81.MCI00	I	POSIF1.MOUT[0]	Multi Channel pattern input for CCST1
CCU81.MCI01	I	POSIF1.MOUT[1]	Multi Channel pattern input for NOT(CCST1)
CCU81.MCI02	I	POSIF1.MOUT[2]	Multi Channel pattern input for CCST2
CCU81.MCI03	I	POSIF1.MOUT[3]	Multi Channel pattern input for NOT(CCST2)
CCU81.OUT00	O	P0.0; P1.0; P4.4;	Slice compare output from channel 1. Can be the CCST1, NOT(CCST1), CCST2 or NOT(CCST2) path
CCU81.OUT01	O	P1.1; P4.5;	Slice compare output from channel 1. Can be the CCST1, NOT(CCST1), CCST2 or NOT(CCST2) path
CCU81.OUT02	O	P4.6;	Slice compare output from channel 1. Can be the CCST1, NOT(CCST1), CCST2 or NOT(CCST2) path
CCU81.OUT03	O	P4.7;	Slice compare output from channel 1. Can be the CCST1, NOT(CCST1), CCST2 or NOT(CCST2) path
CCU81.GP00	O	CCU81.IN3AM	Selected signal for event 0
CCU81.GP01	O	not connected	Selected signal for event 1
CCU81.GP02	O	not connected	Selected signal for event 2
CCU81.ST0	O	CCU81.IN0AT; CCU81.IN1AN; CCU81.IN2AN; CCU81.IN3AN; ERU1.0B3;	Output of the status bit multiplexor. It can be CCST1 or CCST2

(table continues...)

Table 195 (continued) CCU81 - CC80 Pin Connections

Input/Output	I/O	Connected To	Description
CCU81.ST0A	O	not connected	Channel 1 status bit: CCST1
CCU81.ST0B	O	CCU81.IN1AY	Channel 2 status bit: CCST2
CCU81.PS0	O	not connected	Multi channel pattern sync trigger: PM when counting UP (edge aligned) or OM when counting DOWN (center aligned)

Table 196 CCU81 - CC81 Pin Connections

Input/Output	I/O	Connected To	Description
CCU81.IN1AA	I	P3.0	General purpose function
CCU81.IN1AB	I	P4.2	General purpose function
CCU81.IN1AC	I	CCU41.GP12	General purpose function
CCU81.IN1AD	I	POSIF1.OUT2	General purpose function
CCU81.IN1AE	I	POSIF1.OUT5	General purpose function
CCU81.IN1AF	I	ERU0.PDOUT1	General purpose function
CCU81.IN1AG	I	ERU0.IOUT1	General purpose function
CCU81.IN1AH	I	SCU.GSC81	General purpose function
CCU81.IN1AI	I	BCCU0.OUT4	General purpose function
CCU81.IN1AJ	I	BCCU0.OUT7	General purpose function
CCU81.IN1AK	I	CCU41.SR2	General purpose function
CCU81.IN1AL	I	ERU0.PDOUT0	General purpose function
CCU81.IN1AM	I	CCU81.GP20	General purpose function
CCU81.IN1AN	I	CCU81.ST0	General purpose function
CCU81.IN1AO	I	CCU81.ST2	General purpose function
CCU81.IN1AP	I	CCU81.ST3	General purpose function
CCU81.IN1AQ	I	CCU81.SR1	General purpose function
CCU81.IN1AR	I	ACMP2.OUT	General purpose function
CCU81.IN1AS	I	ACMP1.OUT	General purpose function
CCU81.IN1AT	I	CCU81.ST1	General purpose function
CCU81.IN1AU	I	P0.13	General purpose function
CCU81.IN1AV	I	ERU1.PDOUT1	General purpose function
CCU81.IN1AW	I	ERU1.IOUT1	General purpose function
CCU81.IN1AX	I	ERU1.PDOUT0	General purpose function
CCU81.IN1AY	I	CCU81.ST0B	Reserved
CCU81.IN1AZ	I	0	Reserved
CCU81.IN1BA	I	0	Reserved

(table continues...)

Table 196 (continued) CCU81 - CC81 Pin Connections

Input/Output	I/O	Connected To	Description
CCU81.IN1BB	I	0	Reserved
CCU81.IN1BC	I	0	Reserved
CCU81.IN1BD	I	0	Reserved
CCU81.IN1BE	I	0	Reserved
CCU81.IN1BF	I	0	Reserved
CCU81.IN1BG	I	0	Reserved
CCU81.IN1BH	I	0	Reserved
CCU81.IN1BI	I	0	Reserved
CCU81.IN1BJ	I	0	Reserved
CCU81.IN1BK	I	0	Reserved
CCU81.IN1BL	I	0	Reserved
CCU81.IN1BM	I	0	Reserved
CCU81.IN1BN	I	0	Reserved
CCU81.IN1BO	I	0	Reserved
CCU81.IN1BP	I	0	Reserved
CCU81.IN1BQ	I	0	Reserved
CCU81.IN1BR	I	0	Reserved
CCU81.IN1BS	I	0	Reserved
CCU81.IN1BT	I	0	Reserved
CCU81.IN1BU	I	0	Reserved
CCU81.IN1BV	I	0	Reserved
CCU81.MCI10	I	POSIF1.MOUT[4]	Multi Channel pattern input for CCST1
CCU81.MCI11	I	POSIF1.MOUT[5]	Multi Channel pattern input for NOT(CCST1)
CCU81.MCI12	I	POSIF1.MOUT[6]	Multi Channel pattern input for CCST2
CCU81.MCI13	I	POSIF1.MOUT[7]	Multi Channel pattern input for NOT(CCST2)
CCU81.OUT10	O	P1.2; P4.6;	Slice compare output from channel 1. Can be the CCST1, NOT(CCST1), CCST2 or NOT(CCST2) path
CCU81.OUT11	O	P1.3; P4.1; P4.7;	Slice compare output from channel 1. Can be the CCST1, NOT(CCST1), CCST2 or NOT(CCST2) path

(table continues...)

Table 196 (continued) CCU81 - CC81 Pin Connections

Input/Output	I/O	Connected To	Description
CCU81.OUT12	O	P4.2;	Slice compare output from channel 1. Can be the CCST1, NOT(CCST1), CCST2 or NOT(CCST2) path
CCU81.OUT13	O	P4.3;	Slice compare output from channel 1. Can be the CCST1, NOT(CCST1), CCST2 or NOT(CCST2) path
CCU81.GP10	O	CCU81.IN0AM	Selected signal for event 0
CCU81.GP11	O	not connected	Selected signal for event 1
CCU81.GP12	O	not connected	Selected signal for event 2
CCU81.ST1	O	CCU81.IN0AN; CCU81.IN1AT; CCU81.IN2AO; CCU81.IN3AO; ERU1.2B3;	Output of the status bit multiplexor. It can be CCST1 or CCST2
CCU81.ST1A	O	not connected	Channel 1 status bit: CCST1
CCU81.ST1B	O	CCU81.IN2AY	Channel 2 status bit: CCST2
CCU81.PS1	O	POSIF1.MSYNCA	Multi channel pattern sync trigger: PM when counting UP (edge aligned) or OM when counting DOWN (center aligned)

Table 197 CCU81 - CC82 Pin Connections

Input/Output	I/O	Connected To	Description
CCU81.IN2AA	I	P3.0	General purpose function
CCU81.IN2AB	I	P0.10	General purpose function
CCU81.IN2AC	I	CCU41.GP22	General purpose function
CCU81.IN2AD	I	POSIF1.OUT2	General purpose function
CCU81.IN2AE	I	POSIF1.OUT5	General purpose function
CCU81.IN2AF	I	ERU0.PDOUT2	General purpose function
CCU81.IN2AG	I	ERU0.IOUT2	General purpose function
CCU81.IN2AH	I	SCU.GSC81	General purpose function
CCU81.IN2AI	I	BCCU0.OUT5	General purpose function
CCU81.IN2AJ	I	BCCU0.OUT1	General purpose function
CCU81.IN2AK	I	CCU41.SR3	General purpose function
CCU81.IN2AL	I	ERU0.PDOUT0	General purpose function
CCU81.IN2AM	I	CCU81.GP30	General purpose function
CCU81.IN2AN	I	CCU81.ST0	General purpose function

(table continues...)

Table 197 (continued) CCU81 - CC82 Pin Connections

Input/Output	I/O	Connected To	Description
CCU81.IN2AO	I	CCU81.ST1	General purpose function
CCU81.IN2AP	I	CCU81.ST3	General purpose function
CCU81.IN2AQ	I	CCU81.SR2	General purpose function
CCU81.IN2AR	I	ACMP0.OUT	General purpose function
CCU81.IN2AS	I	ACMP3.OUT	General purpose function
CCU81.IN2AT	I	CCU81.ST2	General purpose function
CCU81.IN2AU	I	P0.14	General purpose function
CCU81.IN2AV	I	ERU1.PDOUT2	General purpose function
CCU81.IN2AW	I	ERU1.IOUT2	General purpose function
CCU81.IN2AX	I	ERU1.PDOUT0	General purpose function
CCU81.IN2AY	I	CCU81.ST1B	Reserved
CCU81.IN2AZ	I	0	Reserved
CCU81.IN2BA	I	0	Reserved
CCU81.IN2BB	I	0	Reserved
CCU81.IN2BC	I	0	Reserved
CCU81.IN2BD	I	0	Reserved
CCU81.IN2BE	I	0	Reserved
CCU81.IN2BF	I	0	Reserved
CCU81.IN2BG	I	0	Reserved
CCU81.IN2BH	I	0	Reserved
CCU81.IN2BI	I	0	Reserved
CCU81.IN2BJ	I	0	Reserved
CCU81.IN2BK	I	0	Reserved
CCU81.IN2BL	I	0	Reserved
CCU81.IN2BM	I	0	Reserved
CCU81.IN2BN	I	0	Reserved
CCU81.IN2BO	I	0	Reserved
CCU81.IN2BP	I	0	Reserved
CCU81.IN2BQ	I	0	Reserved
CCU81.IN2BR	I	0	Reserved
CCU81.IN2BS	I	0	Reserved
CCU81.IN2BT	I	0	Reserved
CCU81.IN2BU	I	0	Reserved

(table continues...)

Table 197 (continued) CCU81 - CC82 Pin Connections

Input/Output	I/O	Connected To	Description
CCU81.IN2BV	I	0	Reserved
CCU81.MCI20	I	POSIF1.MOUT[8]	Multi Channel pattern input for CCST1
CCU81.MCI21	I	POSIF1.MOUT[9]	Multi Channel pattern input for NOT(CCST1)
CCU81.MCI22	I	POSIF1.MOUT[10]	Multi Channel pattern input for CCST2
CCU81.MCI23	I	POSIF1.MOUT[11]	Multi Channel pattern input for NOT(CCST2)
CCU81.OUT20	O	P0.8; P1.4; P2.0; P4.2;	Slice compare output from channel 1. Can be the CCST1, NOT(CCST1), CCST2 or NOT(CCST2) path
CCU81.OUT21	O	P0.9; P1.5; P2.1; P3.0; P4.3;	Slice compare output from channel 1. Can be the CCST1, NOT(CCST1), CCST2 or NOT(CCST2) path
CCU81.OUT22	O	P0.10;	Slice compare output from channel 1. Can be the CCST1, NOT(CCST1), CCST2 or NOT(CCST2) path
CCU81.OUT23	O	P0.11;	Slice compare output from channel 1. Can be the CCST1, NOT(CCST1), CCST2 or NOT(CCST2) path
CCU81.GP20	O	CCU81.IN1AM	Selected signal for event 0
CCU81.GP21	O	not connected	Selected signal for event 1
CCU81.GP22	O	not connected	Selected signal for event 2
CCU81.ST2	O	CCU81.IN0AO; CCU81.IN1AO; CCU81.IN2AT; CCU81.IN3AP; ERU1.3B3;	Output of the status bit multiplexor. It can be CCST1 or CCST2
CCU81.ST2A	O	not connected	Channel 1 status bit: CCST1
CCU81.ST2B	O	CCU83.IN3AY	Channel 2 status bit: CCST2
CCU81.PS2	O	not connected	Multi channel pattern sync trigger: PM when counting UP (edge aligned) or OM when counting DOWN (center aligned)

Table 198 CCU81 - CC83 Pin Connections

Input/Output	I/O	Connected To	Description
CCU81.IN3AA	I	P3.0	General purpose function
CCU81.IN3AB	I	–	General purpose function
CCU81.IN3AC	I	CCU41.GP32	General purpose function
CCU81.IN3AD	I	POSIF1.OUT2	General purpose function
CCU81.IN3AE	I	POSIF1.OUT5	General purpose function
CCU81.IN3AF	I	ERU0.PDOUT3	General purpose function
CCU81.IN3AG	I	ERU0.IOUT3	General purpose function
CCU81.IN3AH	I	SCU.GSC81	General purpose function
CCU81.IN3AI	I	BCCU0.OUT8	General purpose function
CCU81.IN3AJ	I	BCCU0.OUT2	General purpose function
CCU81.IN3AK	I	CCU41.SR3	General purpose function
CCU81.IN3AL	I	ERU0.PDOUT0	General purpose function
CCU81.IN3AM	I	CCU81.GP00	General purpose function
CCU81.IN3AN	I	CCU81.ST0	General purpose function
CCU81.IN3AO	I	CCU81.ST1	General purpose function
CCU81.IN3AP	I	CCU81.ST2	General purpose function
CCU81.IN3AQ	I	CCU81.SR3	General purpose function
CCU81.IN3AR	I	ACMP3.OUT	General purpose function
CCU81.IN3AS	I	ACMP2.OUT	General purpose function
CCU81.IN3AT	I	CCU81.ST3	General purpose function
CCU81.IN3AU	I	P0.15	General purpose function
CCU81.IN3AV	I	ERU1.PDOUT3	General purpose function
CCU81.IN3AW	I	ERU1.IOUT3	General purpose function
CCU81.IN3AX	I	ERU1.PDOUT0	General purpose function
CCU81.IN3AY	I	CCU81.ST2B	Reserved
CCU81.IN3AZ	I	0	Reserved
CCU81.IN3BA	I	0	Reserved
CCU81.IN3BB	I	0	Reserved
CCU81.IN3BC	I	0	Reserved
CCU81.IN3BD	I	0	Reserved
CCU81.IN3BE	I	0	Reserved
CCU81.IN3BF	I	0	Reserved
CCU81.IN3BG	I	0	Reserved

(table continues...)

Table 198 (continued) CCU81 - CC83 Pin Connections

Input/Output	I/O	Connected To	Description
CCU81.IN3BH	I	0	Reserved
CCU81.IN3BI	I	0	Reserved
CCU81.IN3BJ	I	0	Reserved
CCU81.IN3BK	I	0	Reserved
CCU81.IN3BL	I	0	Reserved
CCU81.IN3BM	I	0	Reserved
CCU81.IN3BN	I	0	Reserved
CCU81.IN3BO	I	0	Reserved
CCU81.IN3BP	I	0	Reserved
CCU81.IN3BQ	I	0	Reserved
CCU81.IN3BR	I	0	Reserved
CCU81.IN3BS	I	0	Reserved
CCU81.IN3BT	I	0	Reserved
CCU81.IN3BU	I	0	Reserved
CCU81.IN3BV	I	0	Reserved
CCU81.MCI30	I	POSIF1.MOUT[12]	Multi Channel pattern input for CCST1
CCU81.MCI31	I	POSIF1.MOUT[13]	Multi Channel pattern input for NOT(CCST1)
CCU81.MCI32	I	POSIF1.MOUT[14]	Multi Channel pattern input for CCST2
CCU81.MCI33	I	POSIF1.MOUT[15]	Multi Channel pattern input for NOT(CCST2)
CCU81.OUT30	O	P1.6; P4.8;	Slice compare output from channel 1. Can be the CCST1, NOT(CCST1), CCST2 or NOT(CCST2) path
CCU81.OUT31	O	P2.13; P4.9;	Slice compare output from channel 1. Can be the CCST1, NOT(CCST1), CCST2 or NOT(CCST2) path
CCU81.OUT32	O	–	Slice compare output from channel 1. Can be the CCST1, NOT(CCST1), CCST2 or NOT(CCST2) path
CCU81.OUT33	O	P2.13;	Slice compare output from channel 1. Can be the CCST1, NOT(CCST1), CCST2 or NOT(CCST2) path
CCU81.GP30	O	CCU81.IN2AM	Selected signal for event 0
CCU81.GP31	O	not connected	Selected signal for event 1
CCU81.GP32	O	not connected	Selected signal for event 2

(table continues...)

Table 198 (continued) CCU81 - CC83 Pin Connections

Input/Output	I/O	Connected To	Description
CCU81.ST3	O	CCU41.IN0AH; CCU81.IN0AP; CCU81.IN1AP; CCU81.IN2AP; CCU81.IN3AT; ERU1.1B3; VADC0.BGREQGTH; VADC0.G0REQGTH; VADC0.G1REQGTH;	Output of the status bit multiplexor. It can be CCST1 or CCST2
CCU81.ST3A	O	not connected	Channel 1 status bit: CCST1
CCU81.ST3B	O	CCU81.IN0AY	Channel 2 status bit: CCST2
CCU81.PS3	O	POSIF1.MSYNCB	Multi channel pattern sync trigger: PM when counting UP (edge aligned) or OM when counting DOWN (center aligned)

23 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 199 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*

23.1 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.

23.1.1 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 200 a summary of the major features of the POSIF unit mapped with the most common applications.

Table 200 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

(table continues...)

Table 200 (continued) Applications summary

Feature	Applications
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

23.1.2 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 394, 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 Service Request Generation.

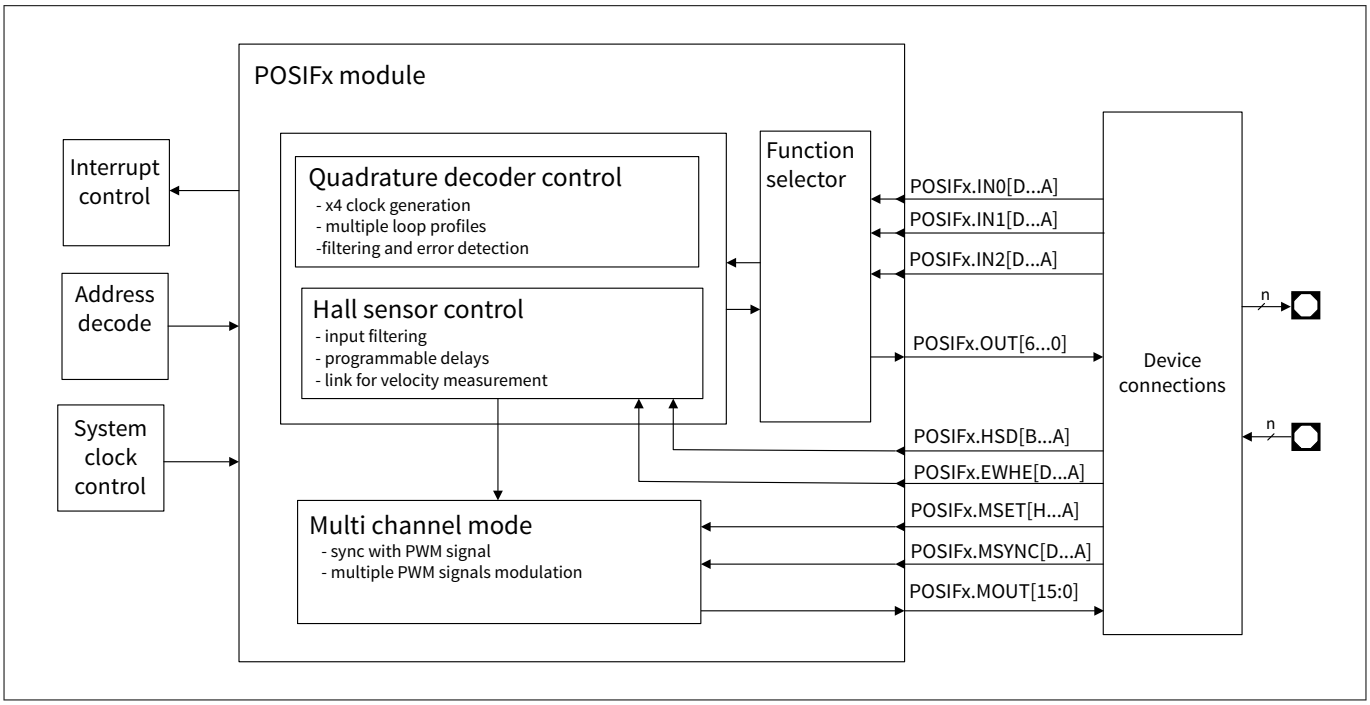


Figure 394 POSIF block diagram

23.2 Functional Description

The following sections describe the complete set of functions and usability of the POSIF peripheral.

In each figure several registers may be depicted to indicate controlability or configurability. These registers follow the description given in Figure 395. One should also note that indexing in a register can be done via the non capital y, x or n.

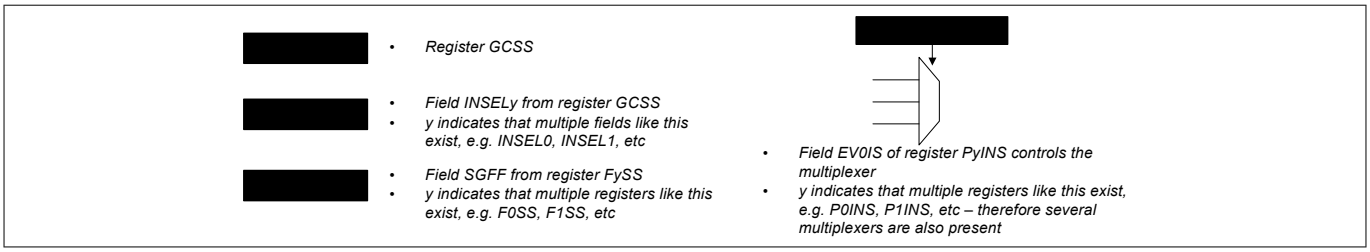


Figure 395 Register description in figures (example)

23.2.1 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 201 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

(table continues...)

Table 201 (continued) POSIF slice pin description

Pin	I/O	Hall Sensor Mode	Quadrature Decoder Mode	Multi-Channel Mode (stand-alone)
POSIFx.OUT0	O	Hall inputs edge detection trigger	Quadrature clock	Not used
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.*

23.2.2 Function Selector

The Function selector maps the input function signals to the selected operating mode, whether Quadrature or Hall Sensor Mode, Figure 396. 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.

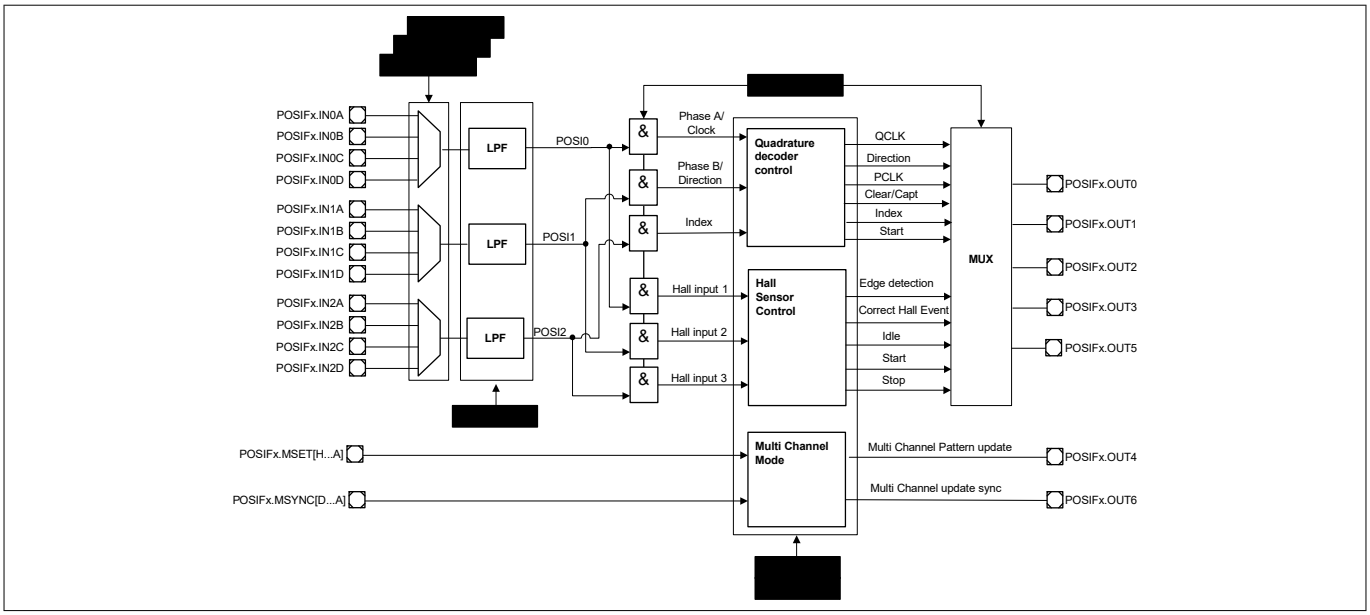


Figure 396 Function selector diagram

23.2.3 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 397. 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).

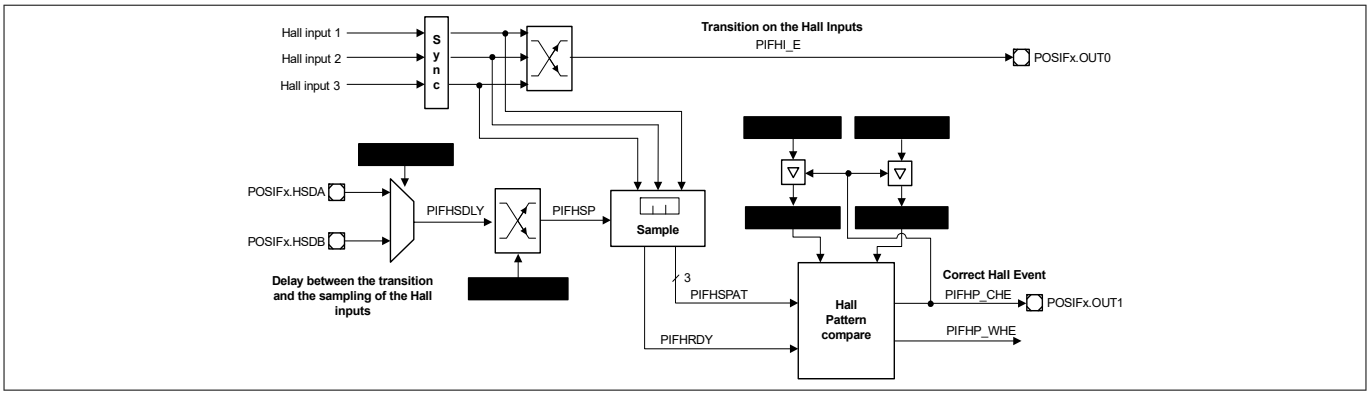


Figure 397 Hall Sensor Control Diagram

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 398.

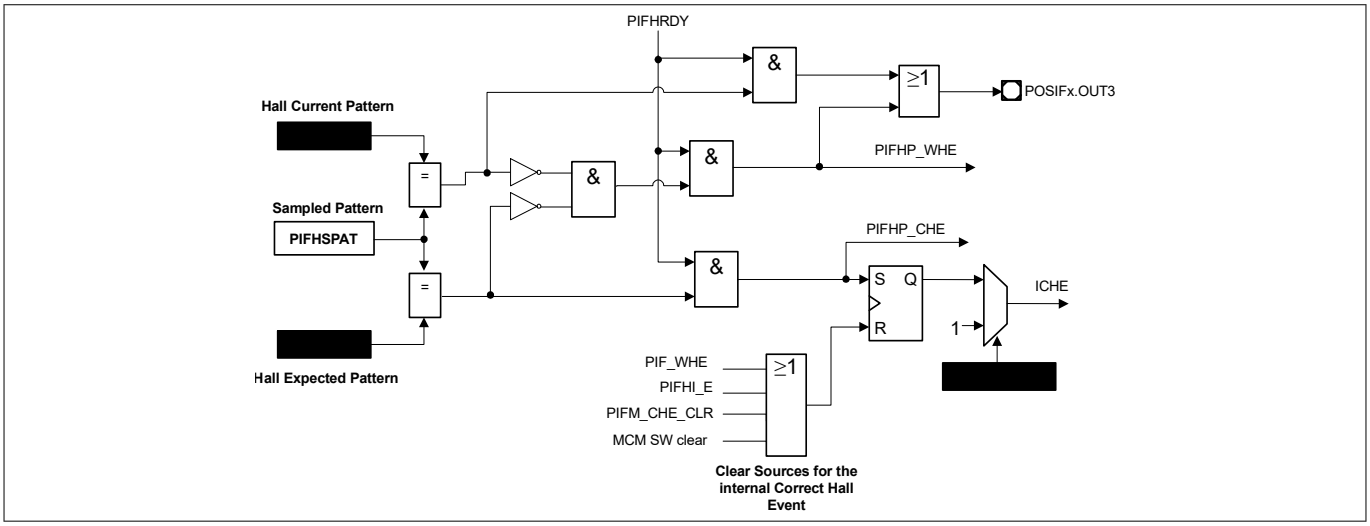


Figure 398 Hall Sensor Compare logic

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 399. The wrong hall event/idle function can also be controlled via a pin.

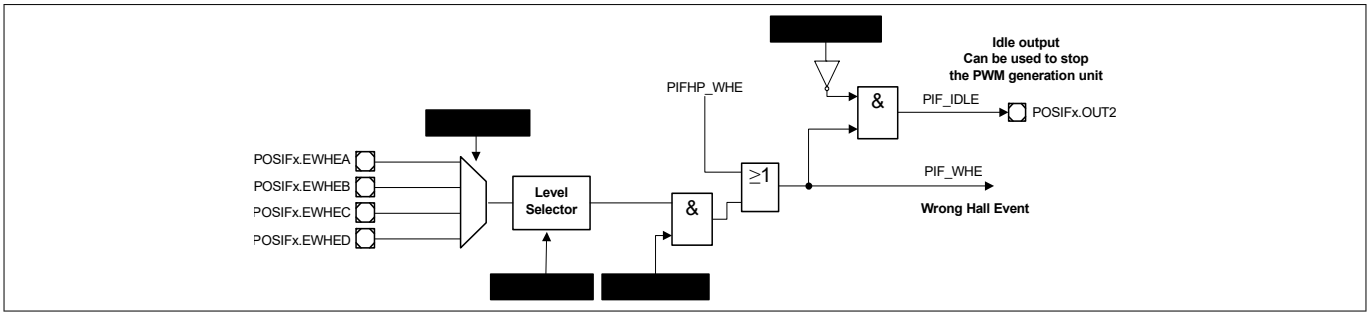


Figure 399 Wrong Hall Event/Idle logic

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 400 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.

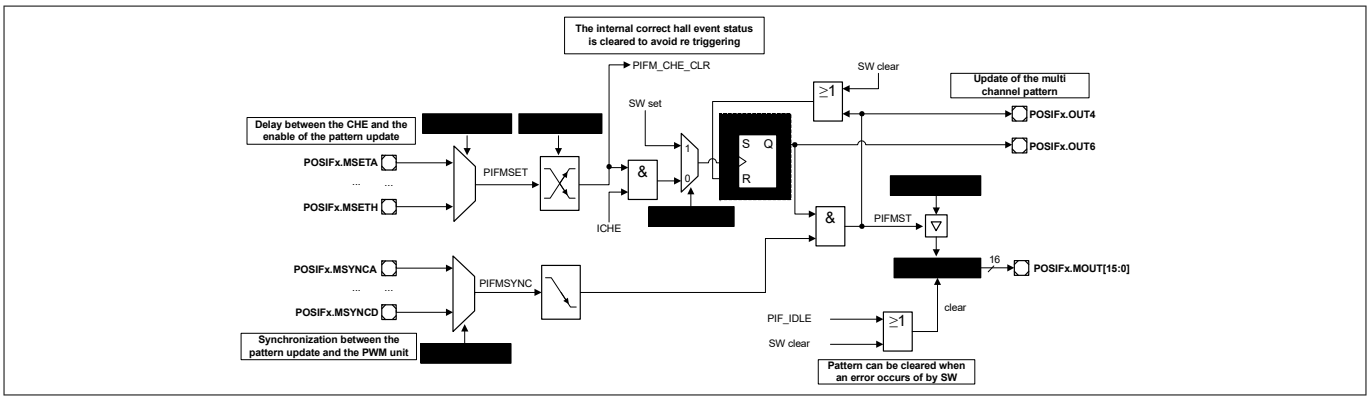


Figure 400 Multi-Channel Mode Diagram

Figure 401 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 401) 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 401 (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).

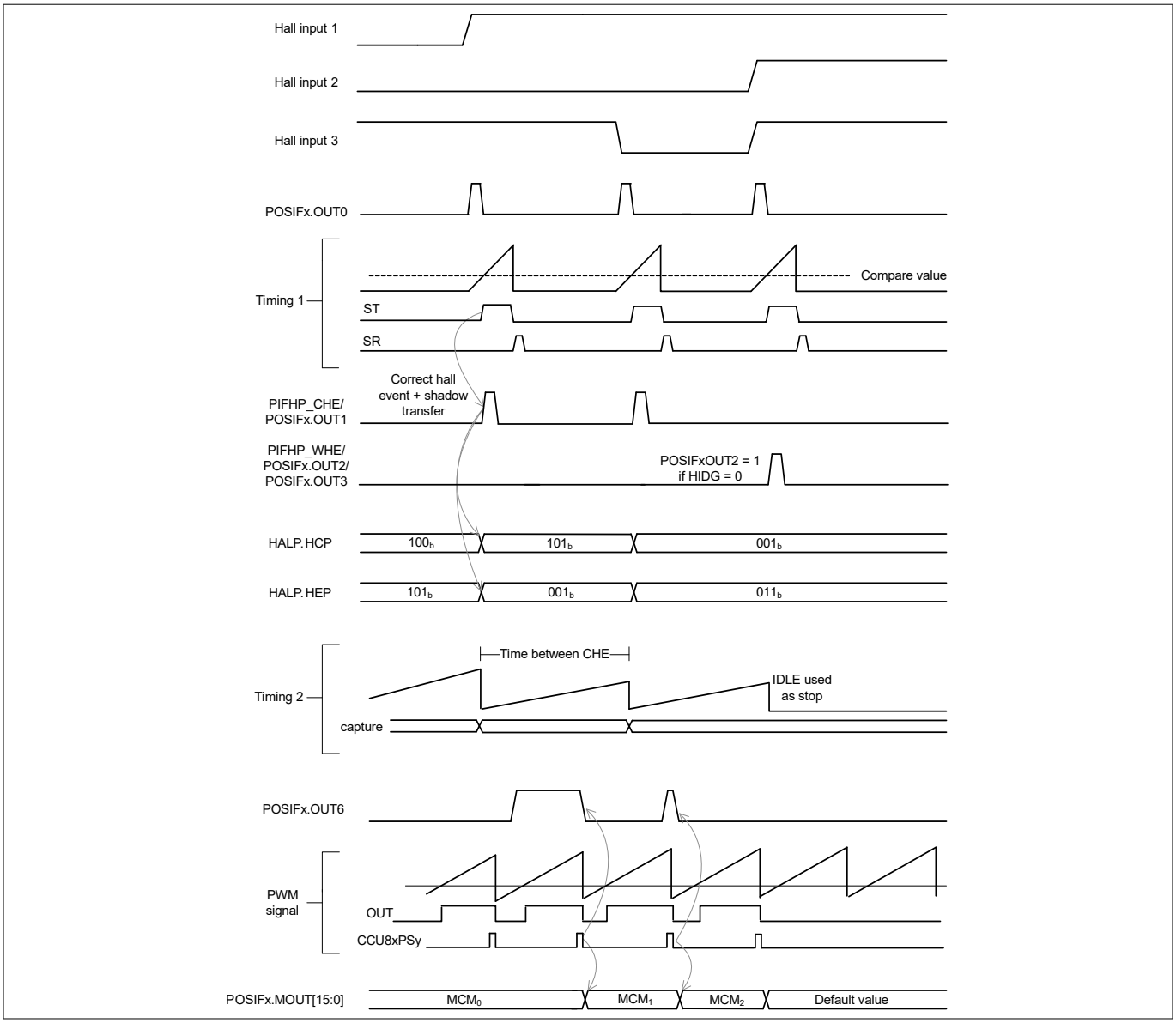


Figure 401 Hall Sensor timing diagram

23.2.4 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 402.

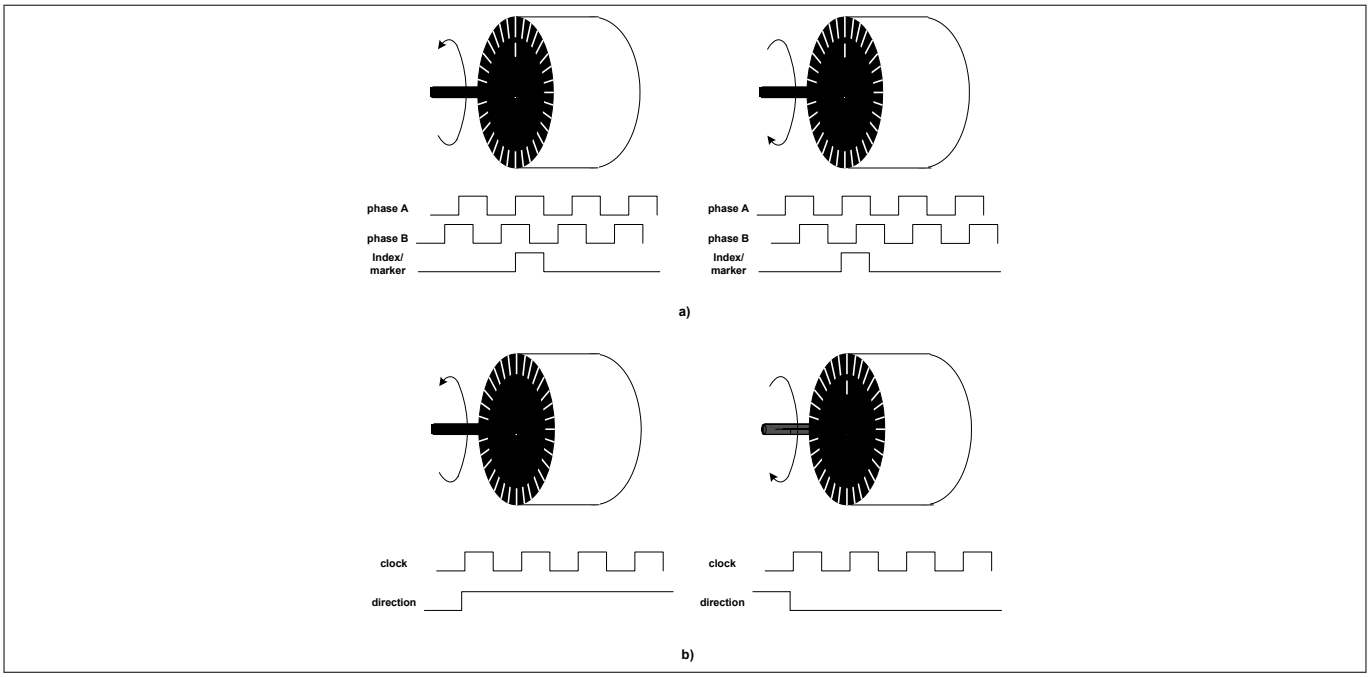


Figure 402 Rotary encoder types - a) standard two phase plus index signal; b) clock plus direction

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 403.

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.

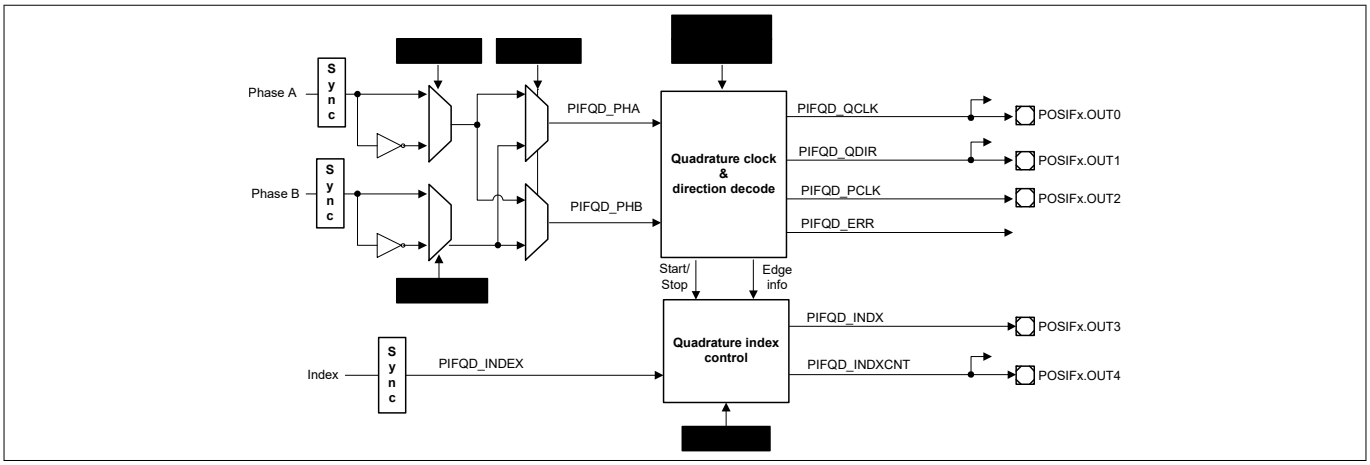


Figure 403 Quadrature Decoder Control Overview

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.

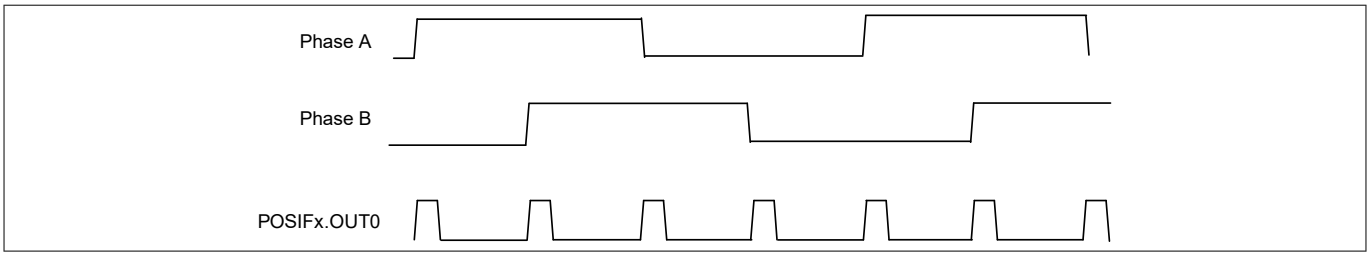


Figure 404 Quadrature clock generation

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 405. 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.

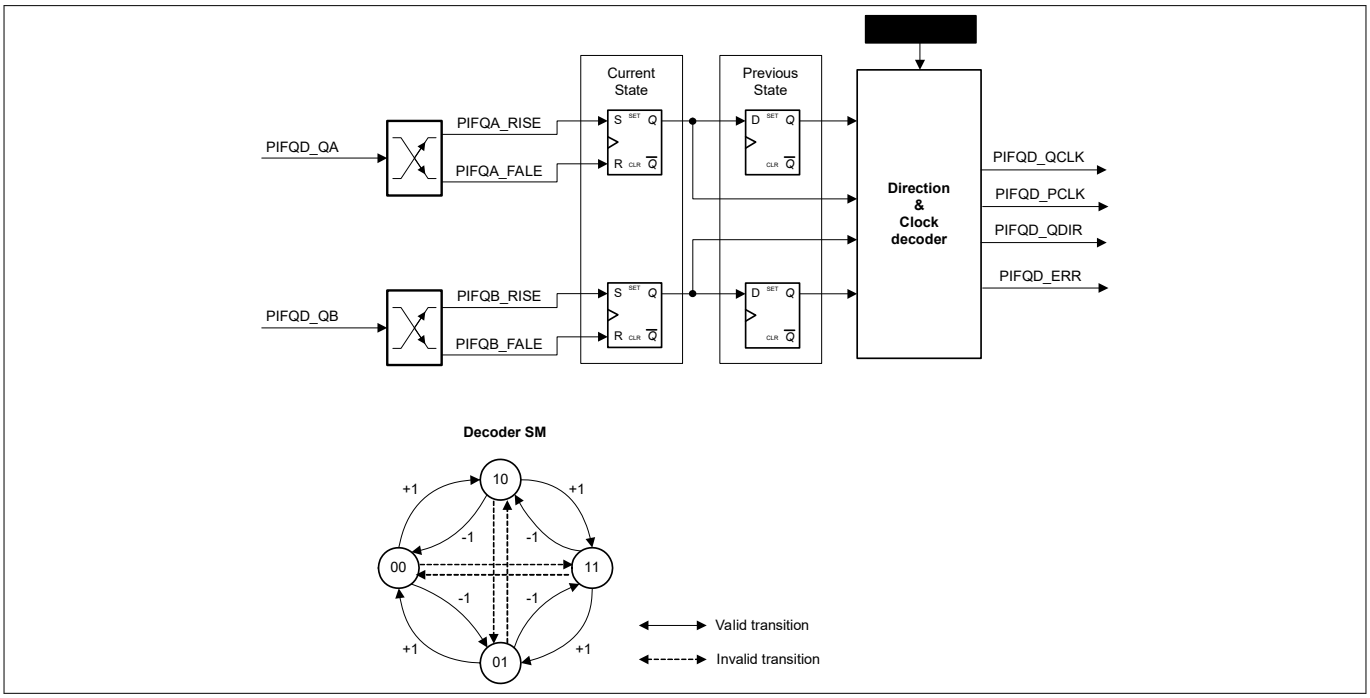


Figure 405 Quadrature Decoder States

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.

23.2.4.1 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 405. Figure 406 shows all the valid transitions of Phase A and Phase B signals.

Figure 407 shows the difference between the two clock signals in the case that some glitches are present in the phase signals.

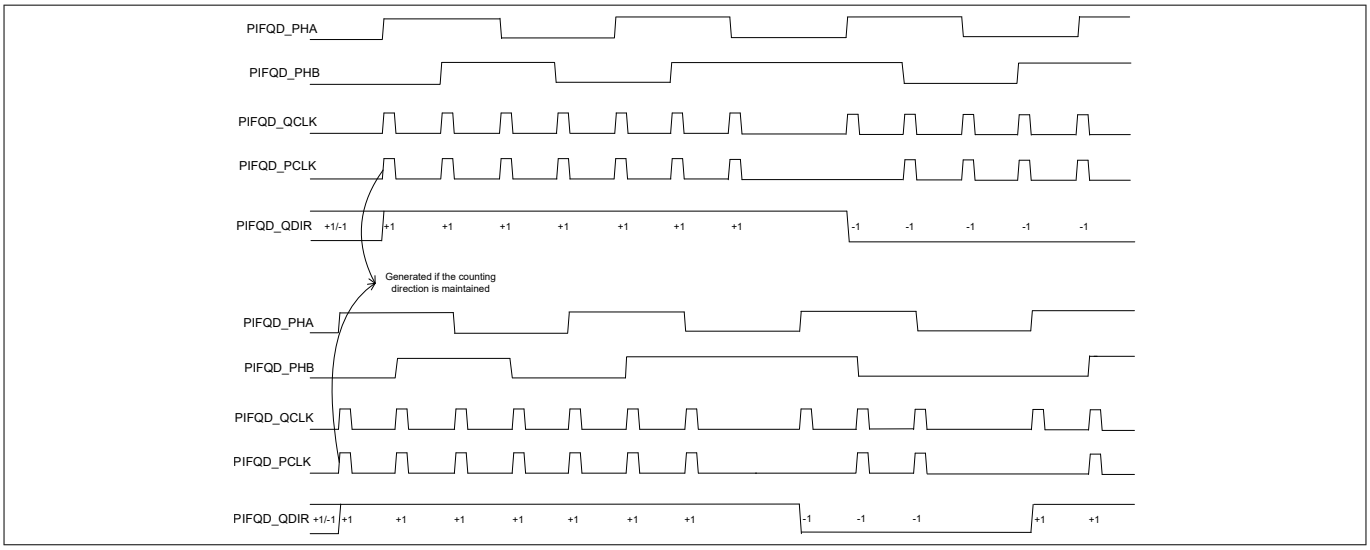


Figure 406 Quadrature clock and direction timings

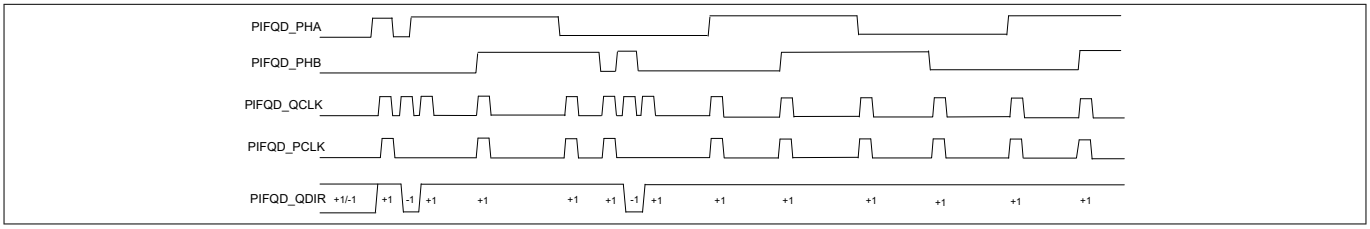


Figure 407 Quadrature clock with jitter

23.2.4.2 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 408 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.

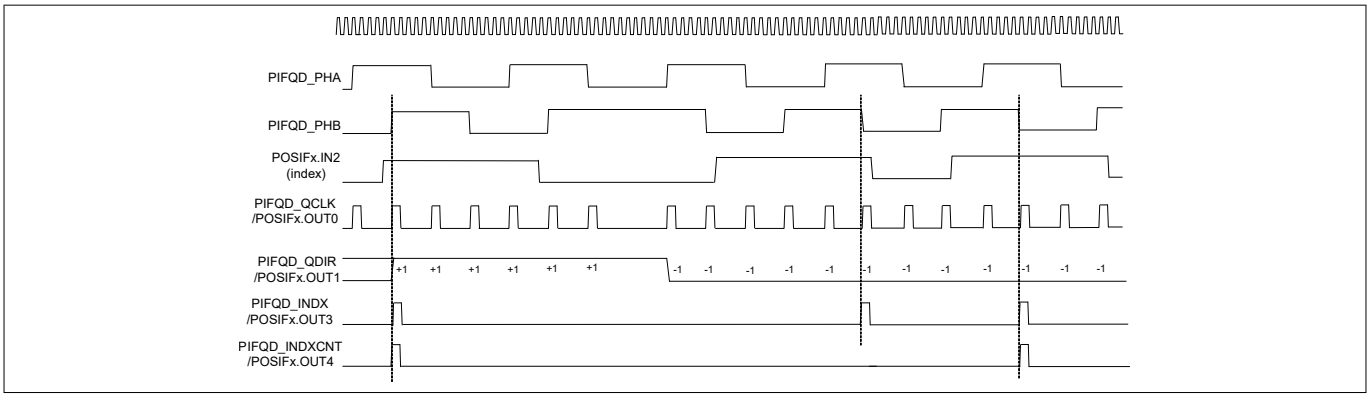


Figure 408 Index signals timing

23.2.5 Stand-Alone Multi-Channel Mode

The Multi-Channel mode (Multi-Channel Mode logic can be see on Figure 400) 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 Hall Sensor Control.

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.

23.2.6 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.

23.2.7 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.

23.2.7.1 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 409, 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.

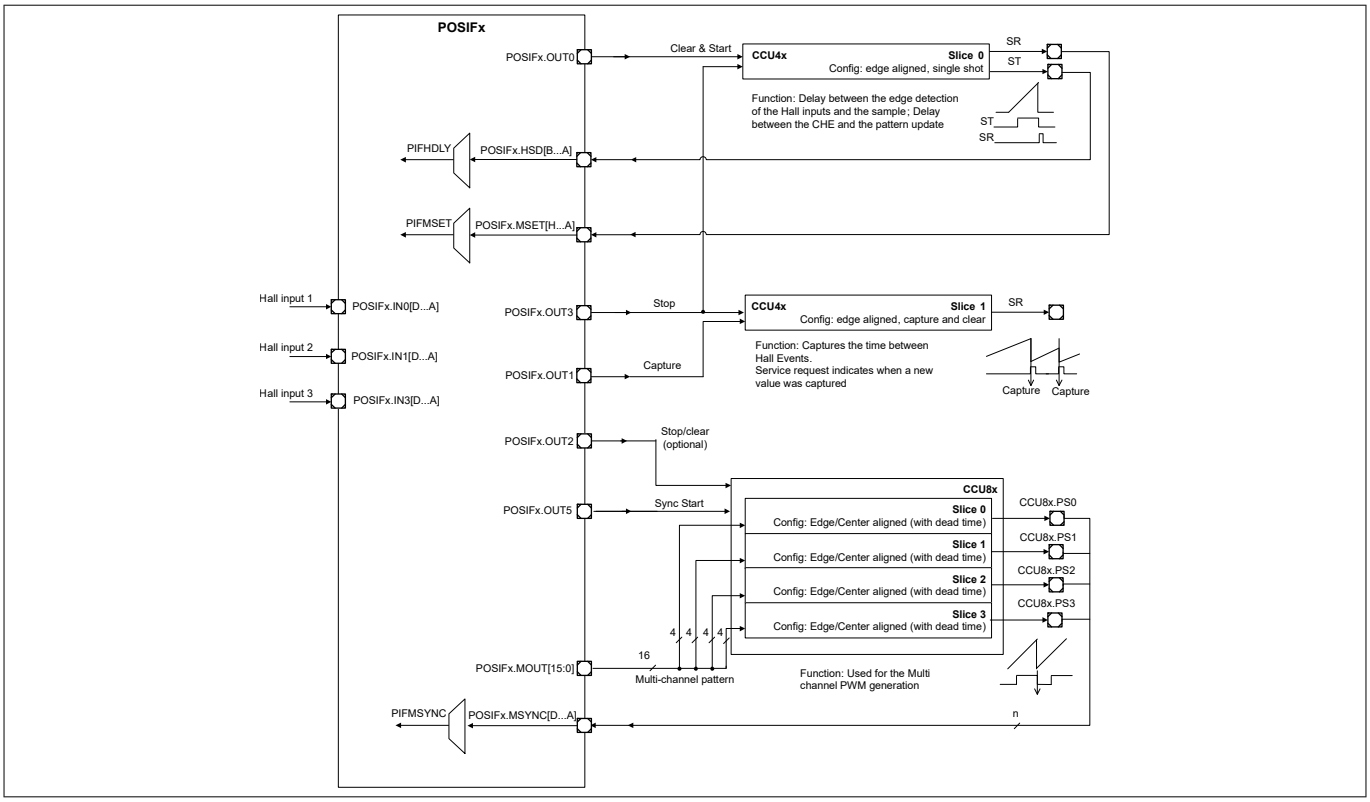


Figure 409 Hall Sensor Mode usage - profile 1

Hall Sensor Mode Usage - Flexible Time Control

On Figure 410, 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.

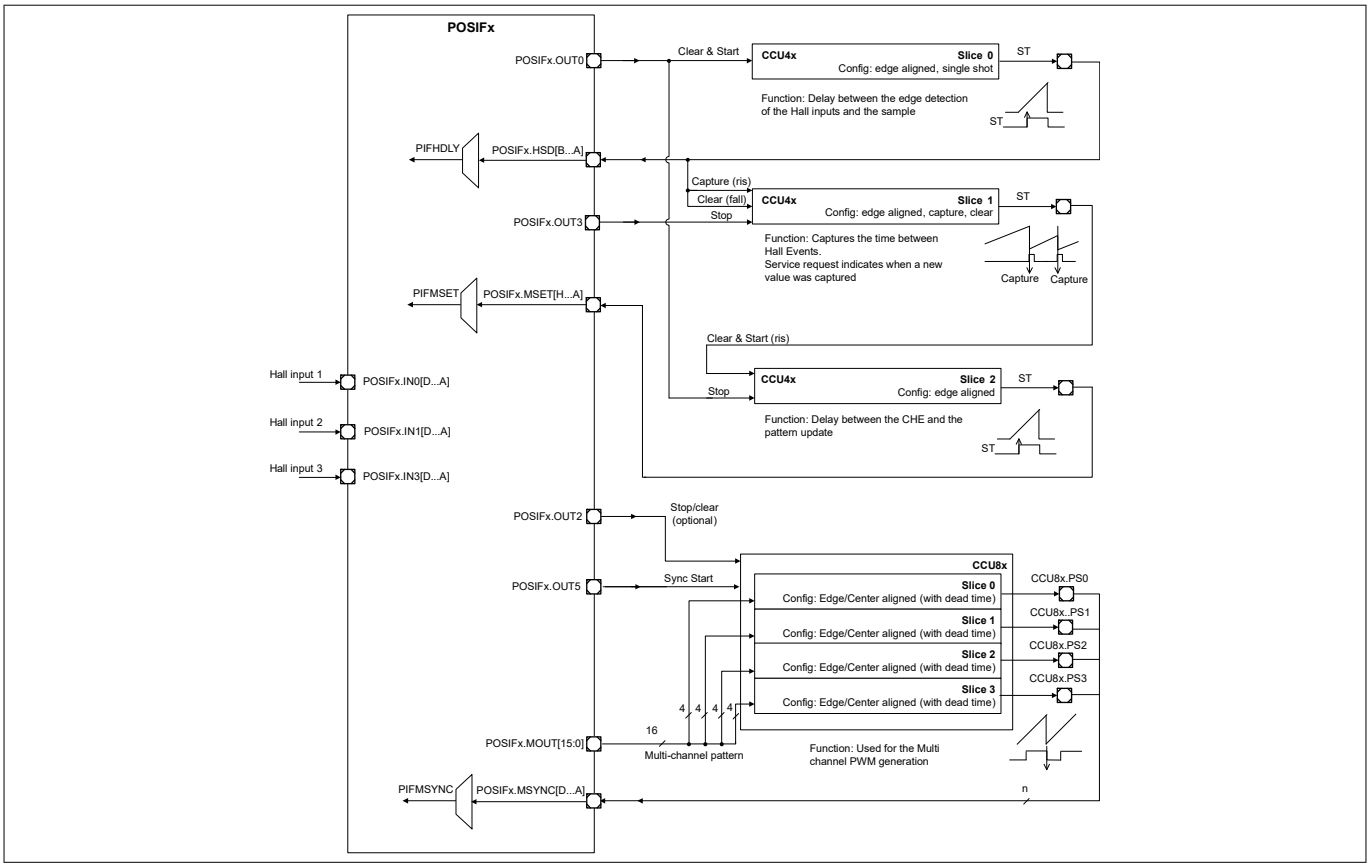


Figure 410 Hall Sensor Mode usage - profile 2

23.2.7.2 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 411, 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.

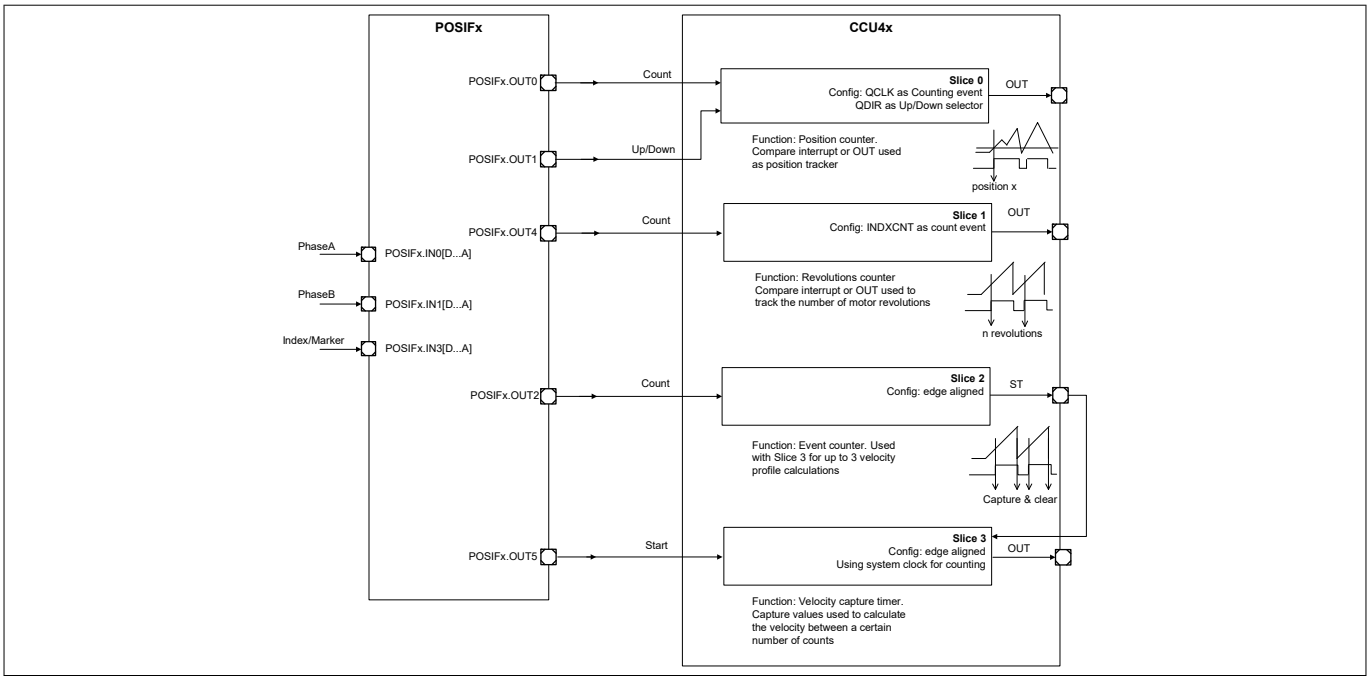


Figure 411 Quadrature Decoder Mode usage - profile 1

Quadrature Decoder Usage - Extended Tick Comparison plus Velocity Between N Ticks

The profile demonstrated in Figure 412 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 412.

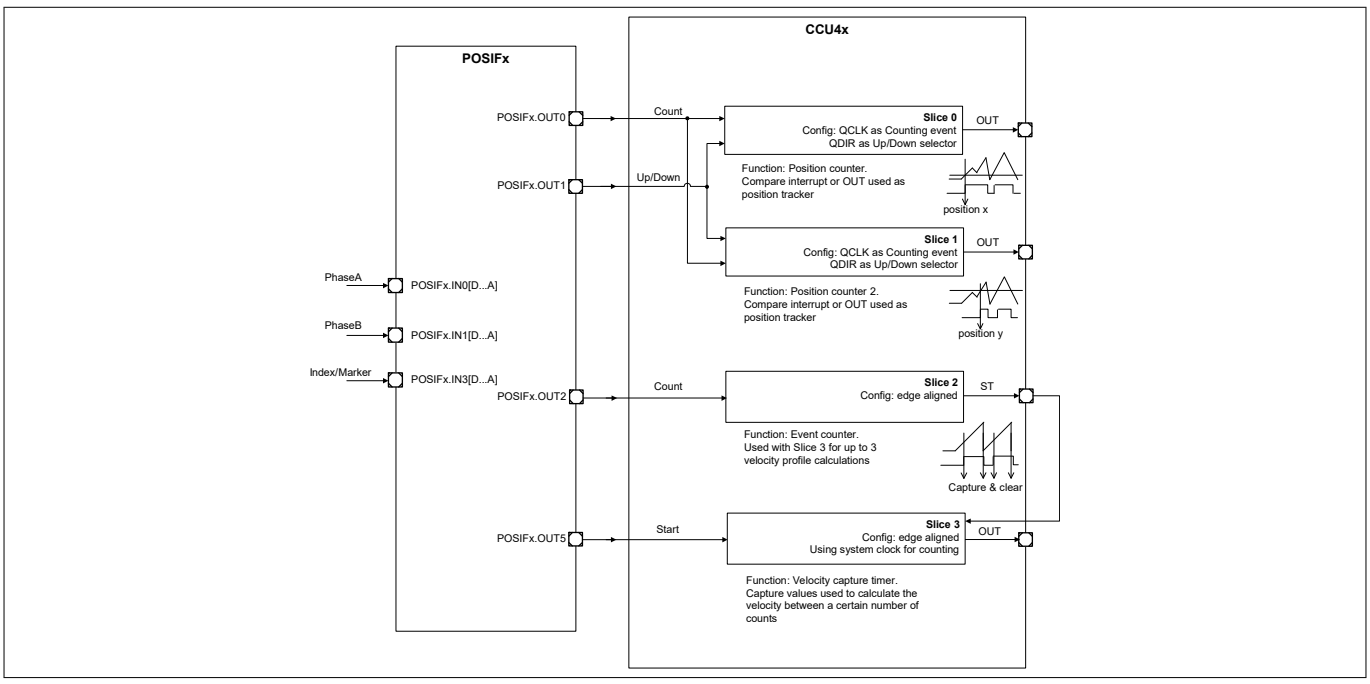


Figure 412 Quadrature Decoder Mode usage - profile 2

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 413 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.

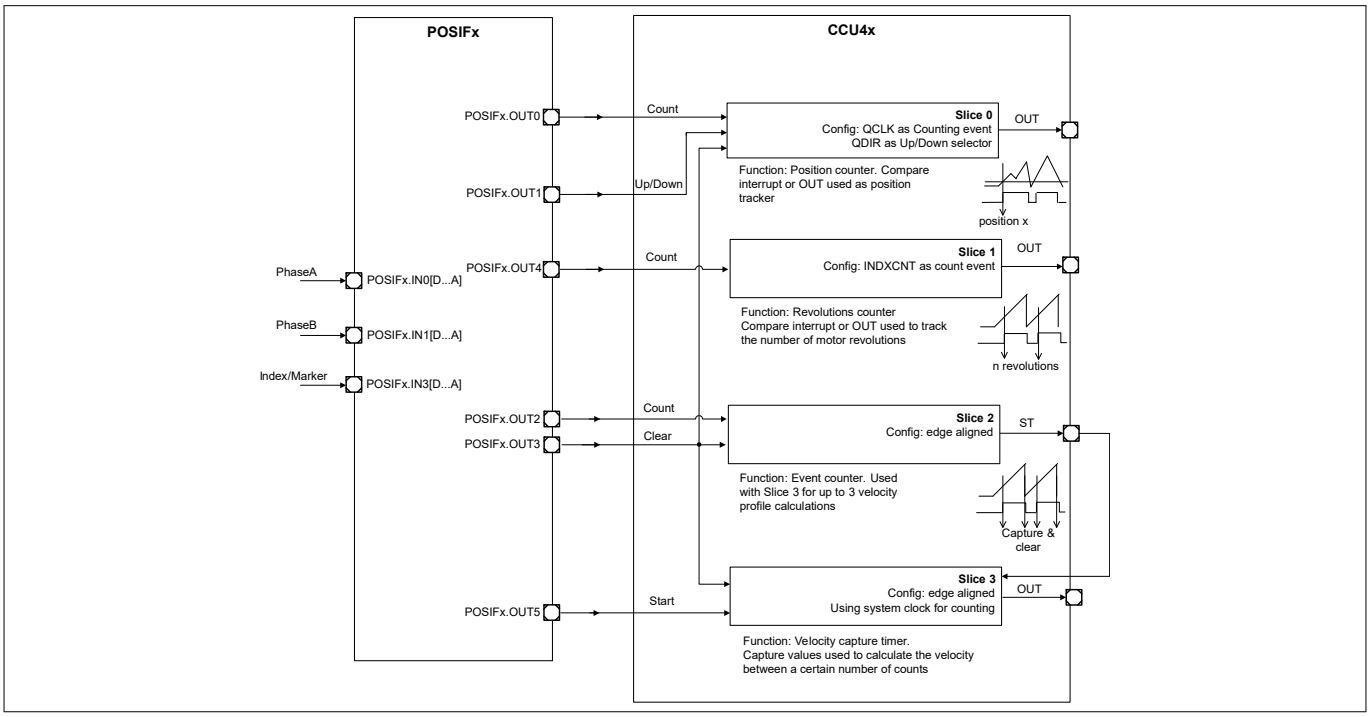


Figure 413 Quadrature Decoder Mode usage - profile 3

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 414 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.

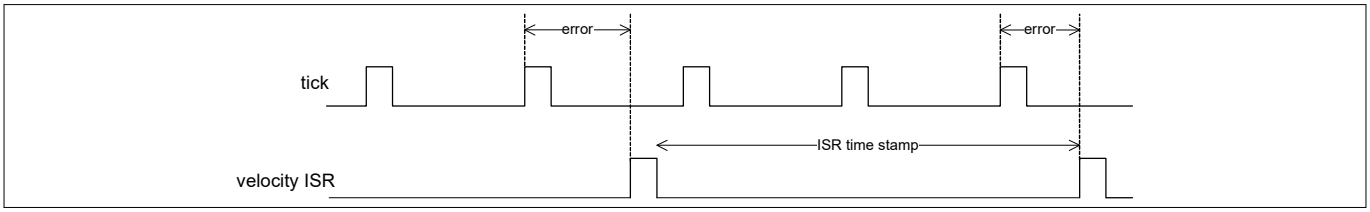


Figure 414 Slow rotating system example

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 415.

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.

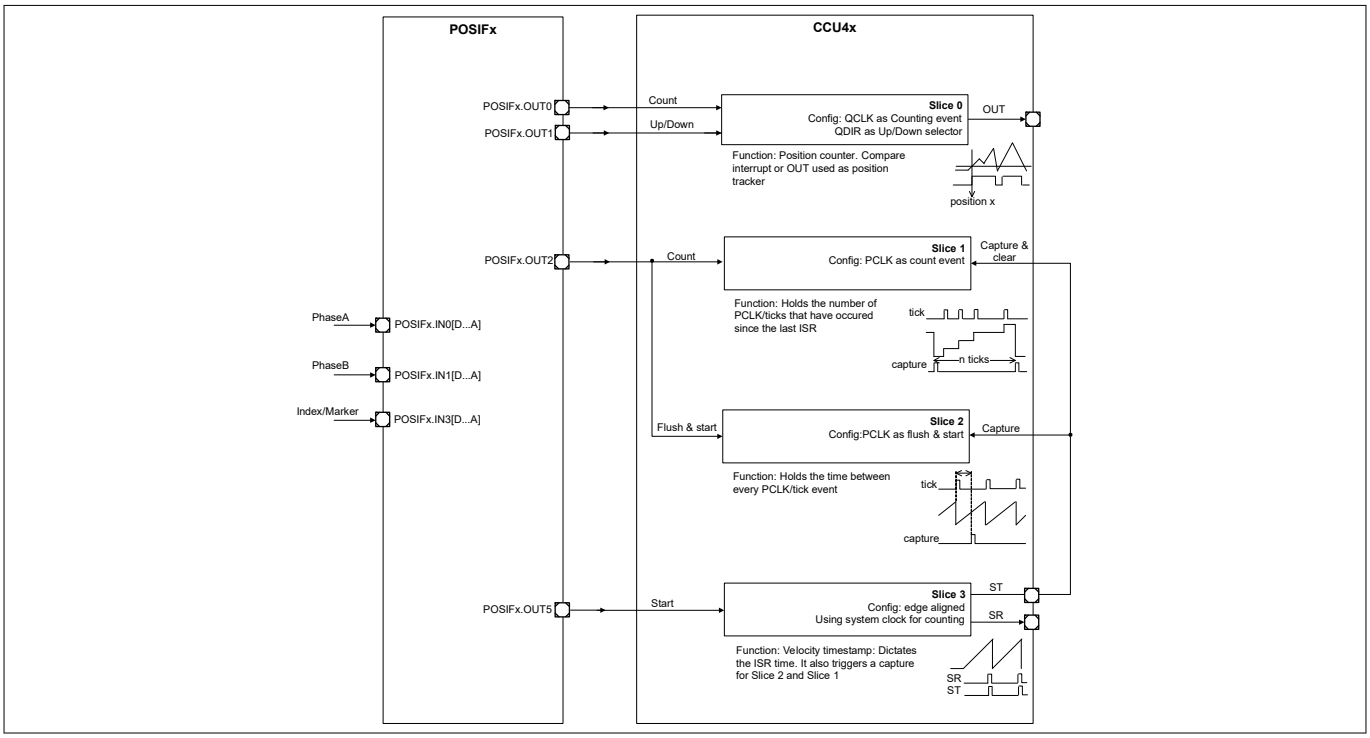


Figure 415 Quadrature Decoder Mode usage - profile 4

23.2.7.3 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 416, a slice from a CCU4 unit is used to control the delay between the update of an external signal (for example, 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).

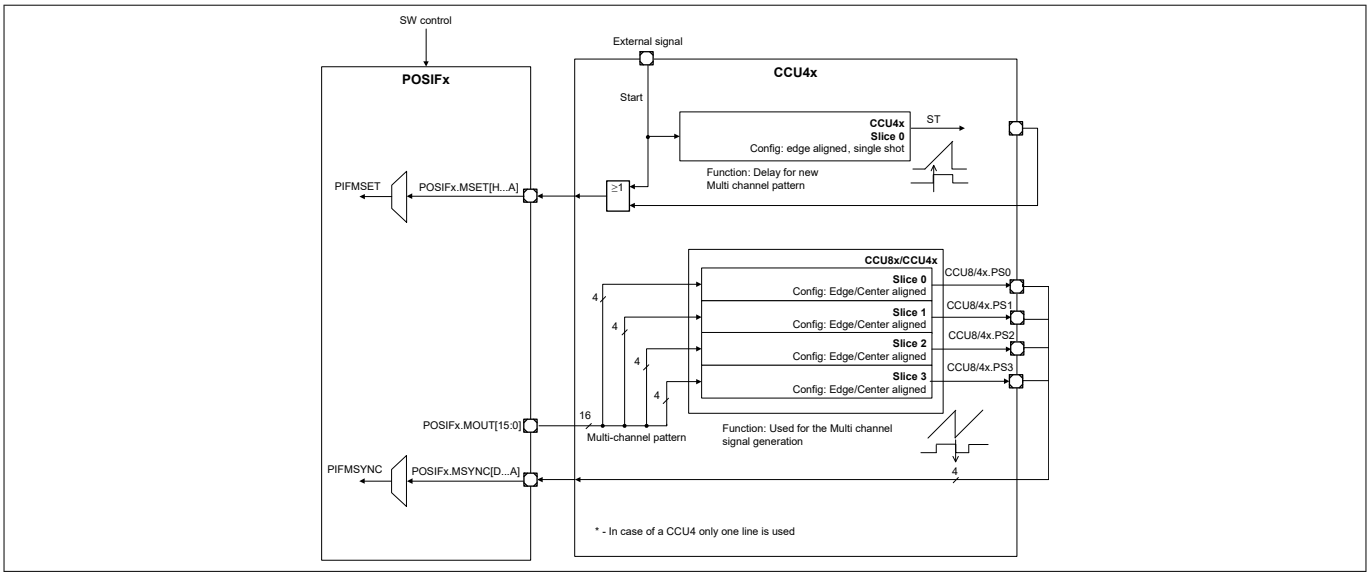


Figure 416 Stand-alone Multi-Channel Mode usage

23.3 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 Hall Sensor Mode flags while the interrupt flags of the Quadrature Decoder Mode are described in Quadrature Decoder Flags.

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.

23.3.1 Hall Sensor Mode flags

The Hall Sensor Control contains four flags that can be configured to generate an interrupt request pulse, see Figure 417.

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 400).

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.

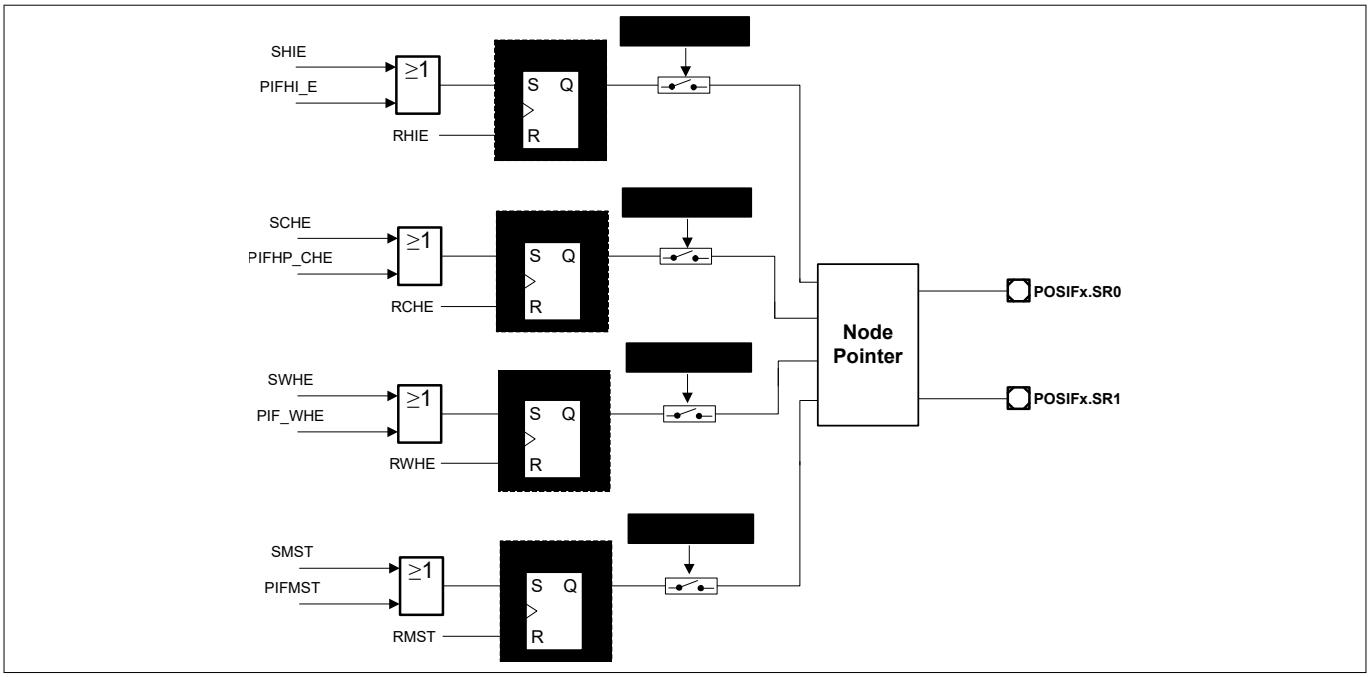


Figure 417 Hall Sensor Mode flags

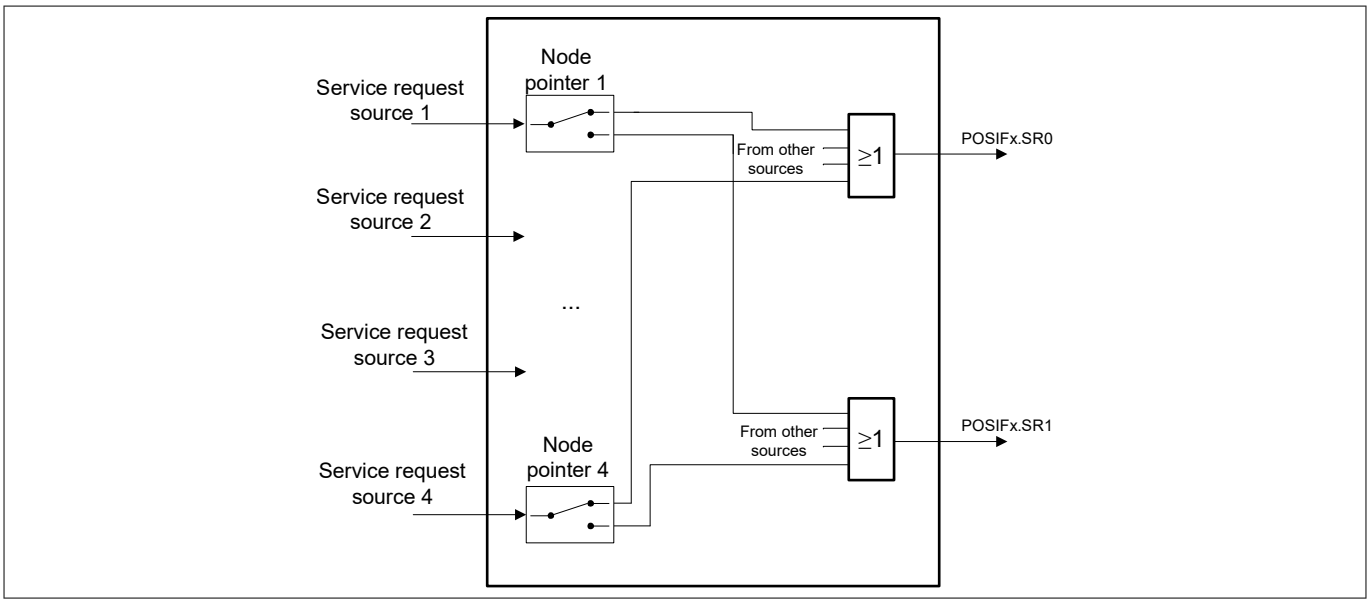


Figure 418 Interrupt node pointer overview - hall sensor mode

23.3.2 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 405.

The quadrature clock and period clock flags are generated accordingly with the timings shown in Figure 406.

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 419.

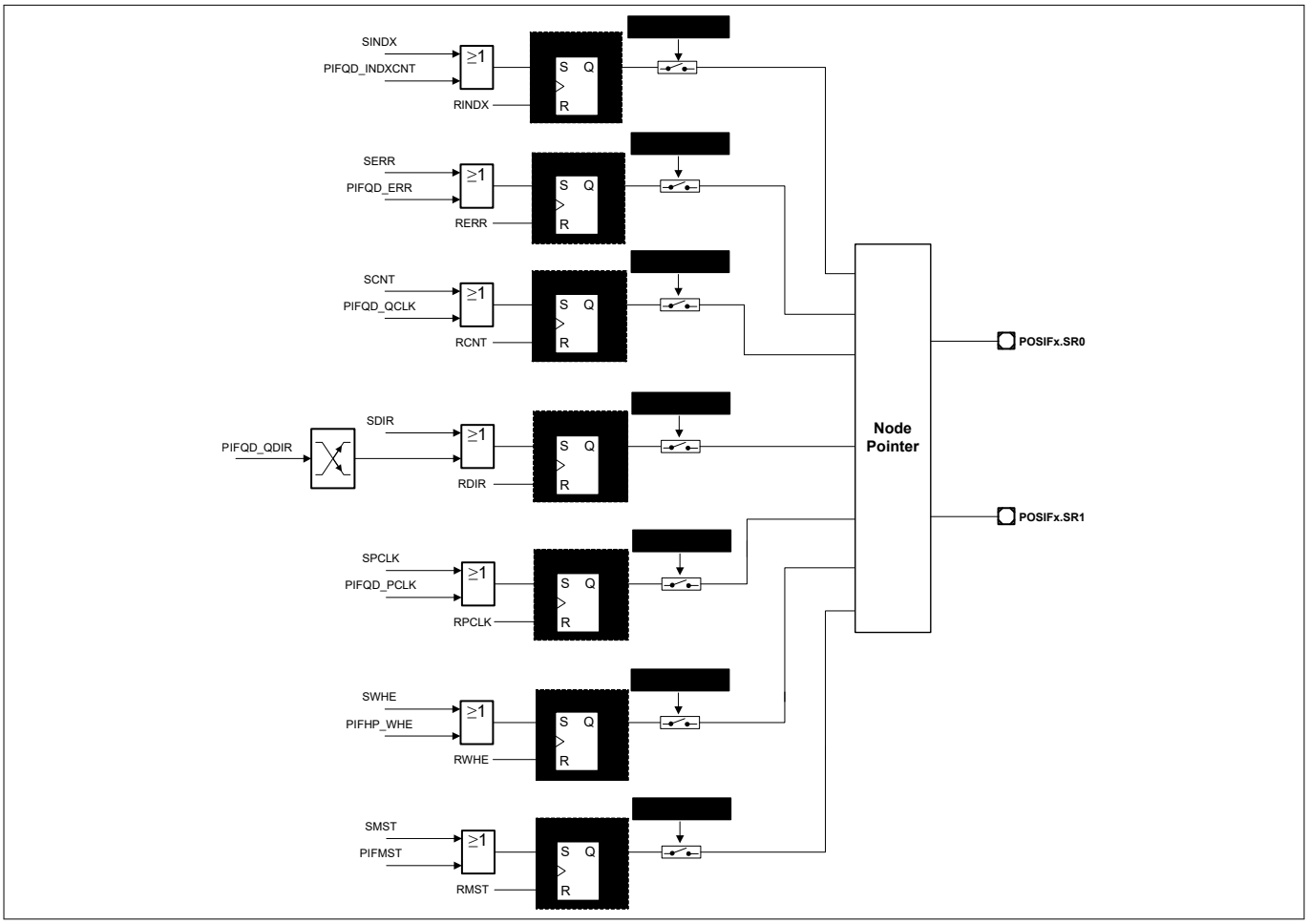


Figure 419 Quadrature Decoder flags

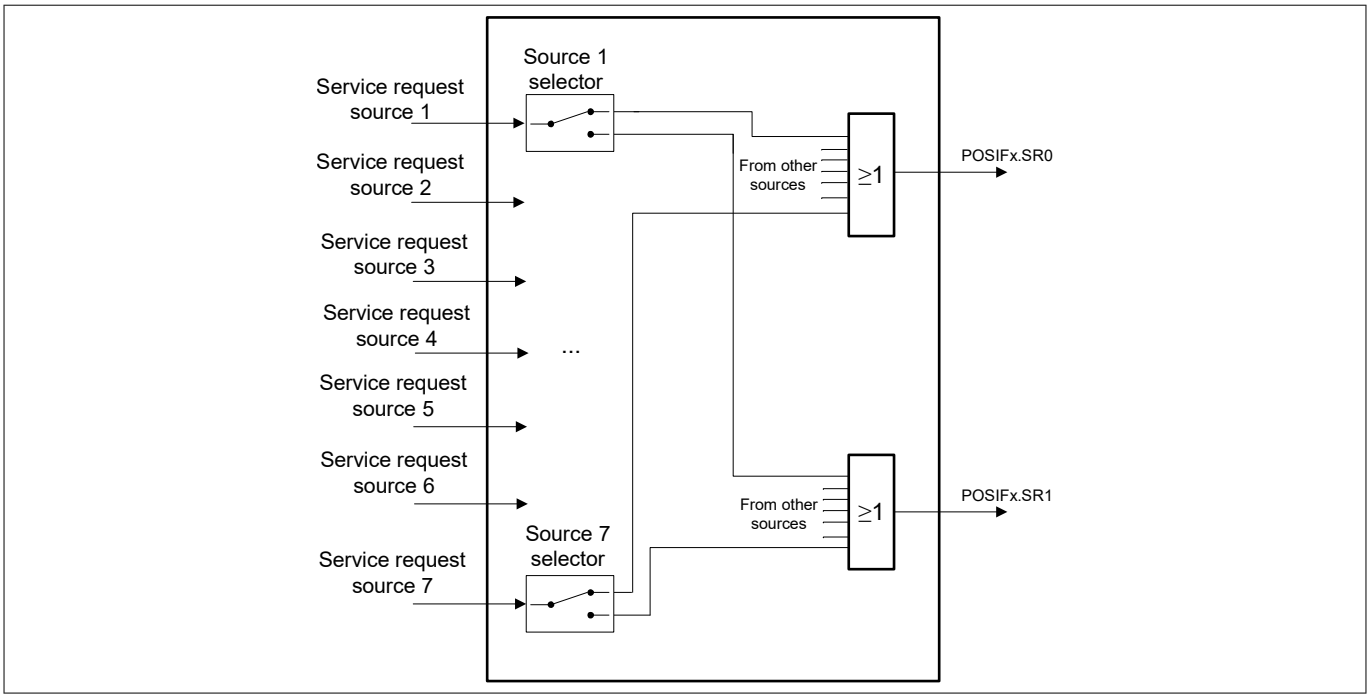


Figure 420 Interrupt node pointer overview - quadrature decoder mode

23.4 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, for example,, 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.

23.5 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}.

23.5.1 Clocks

Module Clock

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 202.

Table 202 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	

23.5.2 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.

23.6 Initialization and System Dependencies

23.6.1 Initialization

The initialization sequence for an application that is using the POSIF, should be the following:

***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.*

23.6.2 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 Clocks
- 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.

23.7 Registers

Registers Overview

The absolute register address is calculated by adding:
Module Base Address + Offset Address

Table 203 Registers Address Space

Module	Base Address	End Address	Note
POSIF0	50010000_H	$50013FFF_H$	
POSIF1	50014000_H	$50017FFF_H$	

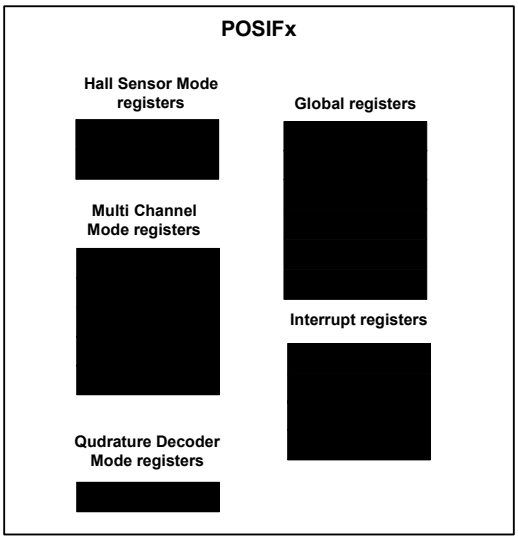


Figure 421 POSIF registers overview

Table 204 Register Overview of POSIF

Register Short Name	Description	Offset Addr.[1)]	Access Mode	
			Read	Write
POSIF Kernel Registers				
PCONF	Global control register	0000_H	U, PV	U, PV
PSUS	Suspend Configuration	0004_H	U, PV	U, PV
PRUNS	POSIF run bit set	0008_H	U, PV	U, PV
PRUNC	POSIF run bit clear	$000C_H$	U, PV	U, PV
PRUN	POSIF run bit status	0010_H	U, PV	BE
PDBG	Debug Design Register	0100_H	U, PV	BE
MIDR	Module Identification register	0020_H	U, PV	BE
Hall Sensor Mode Registers				
HALP	Hall Current and Expected patterns	0030_H	U, PV	BE
HALPS	Hall Current and Expected shadow patterns	0034_H	U, PV	U, PV
Multi-Channel Mode Register				
MCM	Multi-Channel Mode Pattern	0040_H	U, PV	BE
MCSM	Multi-Channel Mode shadow Pattern	0044_H	U, PV	U, PV

(table continues...)

Table 204 (continued) Register Overview of POSIF

Register Short Name	Description	Offset Addr.[1]	Access Mode	
			Read	Write
MCMS	Multi-Channel Mode Control set	0048_H	U, PV	U, PV
MCMC	Multi-Channel Mode Control clear	$004C_H$	U, PV	U, PV
MCMF	Multi-Channel Mode flag status	0050_H	U, PV	BE
Quadrature Decoder Mode Register				
QDC	Quadrature Decoder Configuration	0060_H	U, PV	U, PV
PFLG	POSIF interrupt status	0070_H	U, PV	BE
PFLGE	POSIF interrupt enable	0074_H	U, PV	U, PV
SPFLG	Interrupt set register	0078_H	U, PV	U, PV
RPFLG	Interrupt clear register	$007C_H$	U, PV	U, PV

1) The absolute register address is calculated as follows:
Module Base Address + Offset Address (shown in this column)

23.8 Interconnects

The following tables, describe the connectivity present on the device for each POSIF module.
The GPIO connections are available at the Ports chapter.

23.8.1 POSIF0 Pins

Table 205 POSIF0 Pin Connections

Global Input/Output	I/O	Connected To	Description
POSIF0.CLK	I	PCLK	Module clock is the same one used by the capcoms
POSIF0.IN0A	I	P1.2	Shared connection for rotary encoder and hall sensor
POSIF0.IN0B	I	P0.13	Shared connection for rotary encoder and hall sensor
POSIF0.IN0C	I	VADC0.CBFLOUT0	Shared connection for rotary encoder and hall sensor
POSIF0.IN0D	I	ERU0.PDOUT0	Shared connection for rotary encoder and hall sensor
POSIF0.IN1A	I	P1.1	Shared connection for rotary encoder and hall sensor

(table continues...)

Table 205 (continued) POSIF0 Pin Connections

Global Input/Output	I/O	Connected To	Description
POSIF0.IN1B	I	P0.14	Shared connection for rotary encoder and hall sensor
POSIF0.IN1C	I	VADC0.CBFLOUT1	Shared connection for rotary encoder and hall sensor
POSIF0.IN1D	I	ERU0.PDOUT1	Shared connection for rotary encoder and hall sensor
POSIF0.IN2A	I	P1.0	Shared connection for rotary encoder and hall sensor
POSIF0.IN2B	I	P0.15	Shared connection for rotary encoder and hall sensor
POSIF0.IN2C	I	VADC0.CBFLOUT2	Shared connection for rotary encoder and hall sensor
POSIF0.IN2D	I	ERU0.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
POSIF0.HSDB	I	POSIF0.OUT0	Used for the Hall pattern sample delay. Like the dead time counter in capcom6
POSIF0.EWHEA	I	VADC0.CBFLOUT3	Wrong Hall event emulation, Trap, etc
POSIF0.EWHEB	I	ERU0.IOUT0	Wrong Hall event emulation, Trap, etc
POSIF0.EWHEC	I	ERU0.IOUT3	Wrong Hall event emulation, Trap, etc
POSIF0.EWHED	I	ERU1.IOUT3	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	CCU40.ST2	Multi Pattern updated set. Requests a new shadow transfer
POSIF0.MSETD	I	CCU40.ST3	Multi Pattern updated set. Requests a new shadow transfer
POSIF0.MSETE	I	CCU80.SR1	Multi Pattern updated set. Requests a new shadow transfer

(table continues...)

Table 205 (continued) POSIF0 Pin Connections

Global Input/Output	I/O	Connected To	Description
POSIF0.MSETF	I	ERU0.IOUT2	Multi Pattern updated set. Requests a new shadow transfer
POSIF0.MSETG	I	ERU0.IOUT3	Multi Pattern updated set. Requests a new shadow transfer
POSIF0.MSETH	I	POSIF0.OUT1	Multi Pattern updated set. Requests a new shadow transfer
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	POSIF0.OUT6	Sync for updating the multi channel pattern with the shadow transfer
POSIF0.OUT0	O	CCU40.IN0AE; CCU40.IN1AE; CCU40.IN2BC; CCU40.IN3BC; POSIF0.HSDB;	Quadrature mode: Quadrature clock; Hall sensor mode: Hall input edge detection
POSIF0.OUT1	O	CCU40.IN0AF; CCU40.IN1AF; CCU40.IN2AE; CCU40.IN3BD; POSIF0.MSETH;	Quadrature mode: Shaft direction; Hall sensor mode: Correct Hall Event
POSIF0.OUT2	O	CCU40.IN2AF; CCU80.IN0AD; CCU80.IN1AD; CCU80.IN2AD; CCU80.IN3AD;	Quadrature mode: Pclk for velocity; Hall sensor mode: Idle/wrong hall event
POSIF0.OUT3	O	CCU40.IN0AG; CCU40.IN1AG; CCU40.IN2AG; CCU40.IN3AE;	Quadrature mode: Index event used for Clear/capt; Hall sensor mode: stop in Hall sensor mode
POSIF0.OUT4	O	CCU40.IN1AH; CCU40.IN2AH;	Quadrature mode:Index event; Hall sensor mode: Multi channel pattern update done

(table continues...)

Table 205 (continued) POSIF0 Pin Connections

Global Input/Output	I/O	Connected To	Description
POSIF0.OUT5	O	CCU40.IN3AF; CCU80.IN0AE; CCU80.IN1AE; CCU80.IN2AE; CCU80.IN3AE;	Sync start
POSIF0.OUT6	O	CCU40.MCSS; CCU80.MCSS; POSIF0.MSYNCD;	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; VADC0.BGREQTRO; VADC0.G0REQTRO; VADC0.G1REQTRO;	Service request line 1

23.8.2 POSIF1 Pins

Table 206 POSIF1 Pin Connections

Global Input/Output	I/O	Connected To	Description
POSIF1.CLK	I	PCLK	Module clock is the same one used by the capcoms

(table continues...)

Table 206 (continued) POSIF1 Pin Connections

Global Input/Output	I/O	Connected To	Description
POSIF1.IN2A	I	P1.6	Shared connection for rotary encoder and hall sensor
POSIF1.IN2B	I	P4.3	Shared connection for rotary encoder and hall sensor
POSIF1.IN2C	I	VADC0.CBFLOUT2	Shared connection for rotary encoder and hall sensor
POSIF1.IN2D	I	ERU1.PDOUT2	Shared connection for rotary encoder and hall sensor
POSIF1.HSDA	I	CCU41.ST0	Used for the Hall pattern sample delay. Like the dead time counter in capcom6
POSIF1.HSDB	I	POSIF1.OUT0	Used for the Hall pattern sample delay. Like the dead time counter in capcom6
POSIF1.EWHEA	I	VADC0.CBFLOUT3	Wrong Hall event emulation, Trap, etc
POSIF1.EWHEB	I	ERU1.IOUT0	Wrong Hall event emulation, Trap, etc
POSIF1.EWHEC	I	ERU1.IOUT3	Wrong Hall event emulation, Trap, etc
POSIF1.EWHED	I	ERU0.IOUT3	Wrong Hall event emulation, Trap, etc
POSIF1.MSETA	I	CCU41.SR0	Multi Pattern updated set. Requests a new shadow transfer
POSIF1.MSETB	I	CCU41.ST1	Multi Pattern updated set. Requests a new shadow transfer
POSIF1.MSETC	I	CCU41.ST2	Multi Pattern updated set. Requests a new shadow transfer
POSIF1.MSETD	I	CCU41.ST3	Multi Pattern updated set. Requests a new shadow transfer
POSIF1.MSETE	I	CCU81.SR1	Multi Pattern updated set. Requests a new shadow transfer
POSIF1.MSETF	I	ERU1.IOUT2	Multi Pattern updated set. Requests a new shadow transfer
POSIF1.MSETG	I	ERU1.IOUT3	Multi Pattern updated set. Requests a new shadow transfer
POSIF1.MSETH	I	POSIF1.OUT1	Multi Pattern updated set. Requests a new shadow transfer
POSIF1.MSYNCA	I	CCU81.PS1	Sync for updating the multi channel pattern with the shadow transfer

(table continues...)

Table 206 (continued) POSIF1 Pin Connections

Global Input/Output	I/O	Connected To	Description
POSIF1.MSYNCB	I	CCU81.PS3	Sync for updating the multi channel pattern with the shadow transfer
POSIF1.MSYNCC	I	CCU41.PS1	Sync for updating the multi channel pattern with the shadow transfer
POSIF1.MSYNCD	I	POSIF1.OUT6	Sync for updating the multi channel pattern with the shadow transfer
POSIF1.OUT0	O	CCU41.IN0AE; CCU41.IN1AE; CCU41.IN2BC; CCU41.IN3BC; POSIF1.HSDB;	Quadrature mode: Quadrature clock; Hall sensor mode: Hall input edge detection
POSIF1.OUT1	O	CCU41.IN0AF; CCU41.IN1AF; CCU41.IN2AE; CCU41.IN3BD; POSIF1.MSETH;	Quadrature mode: Shaft direction; Hall sensor mode: Correct Hall Event
POSIF1.OUT2	O	CCU41.IN2AF; CCU81.IN0AD; CCU81.IN1AD; CCU81.IN2AD; CCU81.IN3AD;	Quadrature mode: Pclk for velocity; Hall sensor mode: Idle/wrong hall event
POSIF1.OUT3	O	CCU41.IN0AG; CCU41.IN1AG; CCU41.IN2AG; CCU41.IN3AE;	Quadrature mode: Index event used for Clear/capt; Hall sensor mode: stop in Hall sensor mode
POSIF1.OUT4	O	CCU41.IN1AH; CCU41.IN2AH;	Quadrature mode:Index event; Hall sensor mode: Multi channel pattern update done
POSIF1.OUT5	O	CCU41.IN3AF; CCU81.IN0AE; CCU81.IN1AE; CCU81.IN2AE; CCU81.IN3AE;	Sync start
POSIF1.OUT6	O	CCU41.MCSS; CCU81.MCSS; POSIF1.MSYNCD;	Multi channel pattern update request
POSIF1.MOUT[0]	O	CCU81.MCI00;	Multi channel pattern
POSIF1.MOUT[1]	O	CCU81.MCI01;	Multi channel pattern

(table continues...)

Table 206 (continued) POSIF1 Pin Connections

Global Input/Output	I/O	Connected To	Description
POSIF1.MOUT[2]	O	CCU81.MCI02;	Multi channel pattern
POSIF1.MOUT[3]	O	CCU81.MCI03;	Multi channel pattern
POSIF1.MOUT[4]	O	CCU81.MCI10;	Multi channel pattern
POSIF1.MOUT[5]	O	CCU81.MCI11;	Multi channel pattern
POSIF1.MOUT[6]	O	CCU81.MCI12;	Multi channel pattern
POSIF1.MOUT[7]	O	CCU81.MCI13;	Multi channel pattern
POSIF1.MOUT[8]	O	CCU81.MCI20;	Multi channel pattern
POSIF1.MOUT[9]	O	CCU81.MCI21;	Multi channel pattern
POSIF1.MOUT[10]	O	CCU81.MCI22;	Multi channel pattern
POSIF1.MOUT[11]	O	CCU81.MCI23;	Multi channel pattern
POSIF1.MOUT[12]	O	CCU81.MCI30;	Multi channel pattern
POSIF1.MOUT[13]	O	CCU81.MCI31;	Multi channel pattern
POSIF1.MOUT[14]	O	CCU81.MCI32;	Multi channel pattern
POSIF1.MOUT[15]	O	CCU81.MCI33;	Multi channel pattern
POSIF1.SR0	O	NVIC	Service request line 0
POSIF1.SR1	O	NVIC; ERU1.3A1;	Service request line 1

24 Brightness and Color Control Unit (BCCU)

The BCCU is a dimming control peripheral for LED lighting applications that is capable of controlling multiple LED channels. A one-bit sigma-delta bit stream is provided for every channel that determines the brightness. The brightness can be changed gradually along an exponential curve to appear natural to the human eye by using dedicated dimming engines. The module supports color control by adjusting the relative intensity of selected channels using a linear walk scheme for smooth color changes, and it also supports high-power multi-channel LED lamps by optionally “packing” the bitstream to provide a defined ON-time at the output.

24.1 Overview

The BCCU in PSC1Mx contains 3 identical dimming engines and 9 identical channels. Each channel can generate a one-bit flexible sigma-delta bit stream with a user-adjustable 12-bit average value. This average value can be provided manually or by any of the exponential dimming engines. The bit stream is generated by a 12-bit sigma-delta modulator, and an optional packer which decreases the average rate of output switching by enforcing a defined on-time. Although the primary target application is multi-channel LED dimming with special support for color control, the module can also be used as a multi-channel digital-analog converter with low-pass filters on the outputs.

24.1.1 Features

The BCCU provides the following functionality:

- 3 independent dimming engines
- 12-bit dimming levels
- Dimming along an exponential curve with adjustable dimming time
- 9 independent channels generating a 1-bit on-off signal each
- 12-bit channel intensities to define color
- Channels and dimming engines can be freely interconnected
- One input multiplier and one 12-bit sigma-delta modulator in every channel
- One bit-packer in every channel to decrease the average rate of output switching
- TRAP function
- Two ADC triggering modes

24.2 Functional Description

Each channel can operate independently from the others and can be freely interconnected with any one of the dimming engines. Triggering and TRAP control are globally handled for all channels.

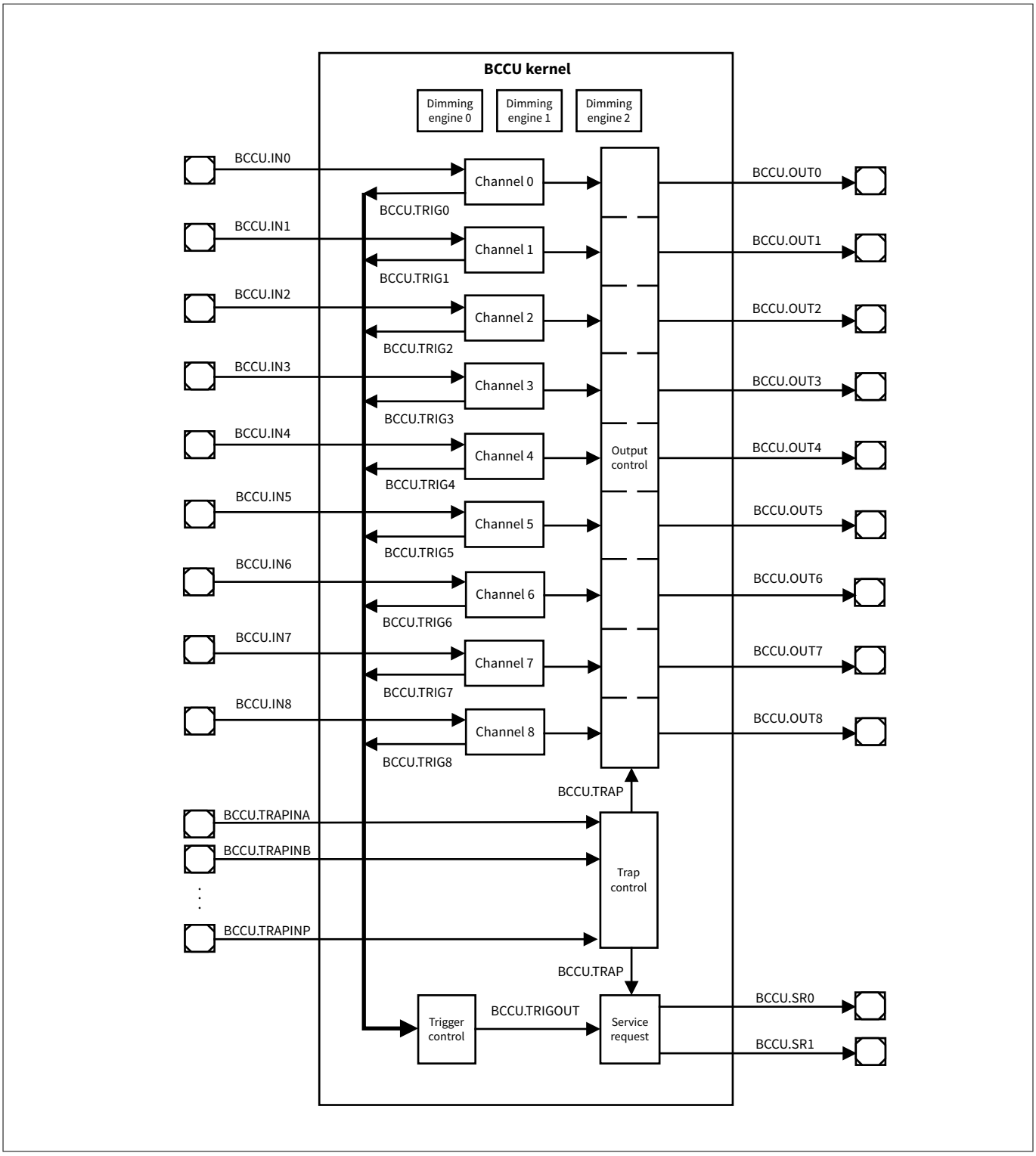


Figure 422 BCCU Kernel Block Diagram

Every dimming engine generates a 12-bit dimming level which can be routed to the input of any channel. Optionally a manually controllable 12-bit "Global Dimming Level" (BCCU_GLOBDIM.GLOBDIM) is also routable to the channels instead.

24.2.1 Channel Structure

Every active channel has a 12-bit intensity level (BCCU_INTy (y=0-8).CHINT). The brightness level (same as the average level of the output bit stream) is the product of the dimming level and the intensity level.

Generally in multi-channel solutions, the intensity level is expected to determine the color and the dimming level is expected to be used for changes in overall brightness. For example, in case of an RGB LED, 3 channels can be used; the individual intensity levels determine the color and a common dimming level (for example, coming from one of the dimming engines) determines the overall brightness.

The 12-bit brightness level is the input to the sigma-delta modulator where it is converted into a fast changing 1-bit signal with the same average value. The average rate of switching of this signal is optionally decreased by a packer which generates a signal with fixed on-time or fixed off-time. This signal still has the same average value.

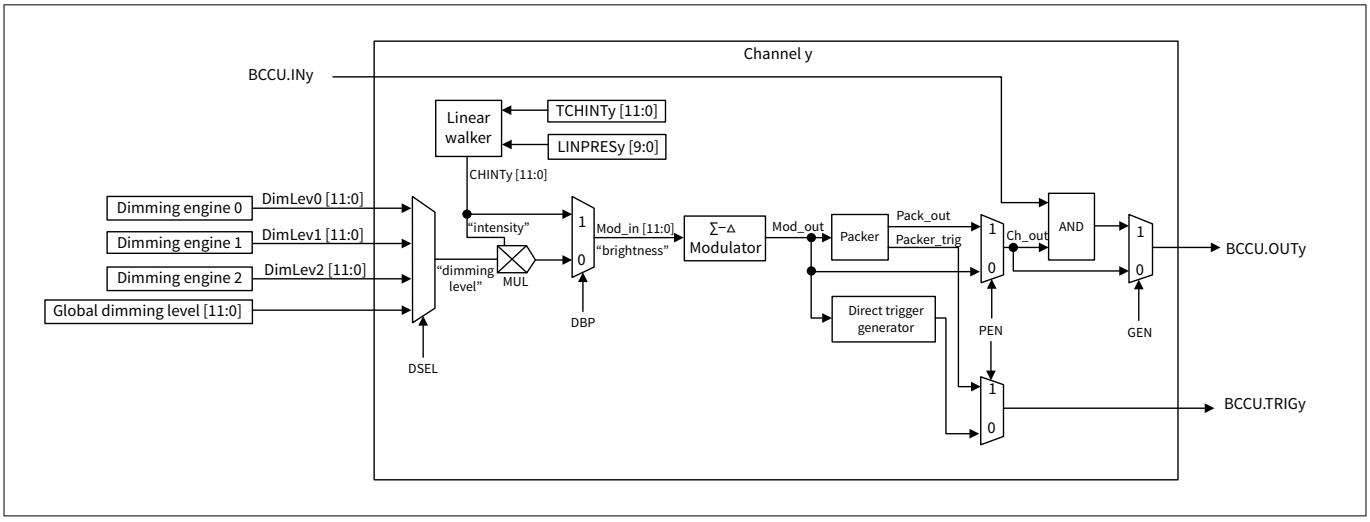


Figure 423 Channel Block Diagram

BCCU.INy is an asynchronous gating signal which can be used if BCCU_CHCONFIGy (y=0-8).GEN is set. The gating connection is made of combinatorial logic only to enable fast control schemes, such as peak-current control.

If the packer is disabled, the trigger signal for the ADC is directly generated from the output of the sigma-delta modulator in the Trigger Generator block. The trigger pulse is generated on rising edge (transition from passive to active or passive to passive or active to active state) if BCCU_CHCONFIGy (y=0-8).TRED is 0 and on falling edge (transition from passive to active or passive to passive or active to active state) if BCCU_CHCONFIGy (y=0-8).TRED is 1. If BCCU_CHCONFIGy (y=0-8).ENFT is 1, forced trigger generation is enabled. A forced trigger is generated if the output signal (Mod_out) hasn't changed state for a long time (256 bit times for rising edge triggering, 257 bit times for falling edge triggering).

24.2.2 Exponential Dimming

The BCCU contains 3 dimming engines that are routable to all 9 channels. The dimming engine is capable of gradually changing the brightness level in order for the human eye to comfortably adapt to changes in light intensity. Since the human eye perceives light with logarithmic sensitivity, the gradual change of brightness is performed along a pseudo-exponential curve. The user has to set the target dimming level (BCCU_DLSz (z=0-2).TDLEV) and initiate the shadow transfer (by setting BCCU_DESTRCON.DEzS). The dimming level (BCCU_DLz (z=0-2).DIMLEV) is a time-varying signal. After shadow transfer, it will gradually reach the target dimming level along the pseudo-exponential curve. When the target is reached after the configured dimming period, BCCU_DESTRCON.DEzS is cleared by hardware. It is recommended to check BCCU_DESTRCON.DEzS before starting a new dimming process.

The dimming process can be aborted anytime by setting BCCU_DESTRCON.DEzA, which is recommended before starting a new dimming process if the actual one hasn't finished yet. This will also clear BCCU_DESTRCON.DEzS.

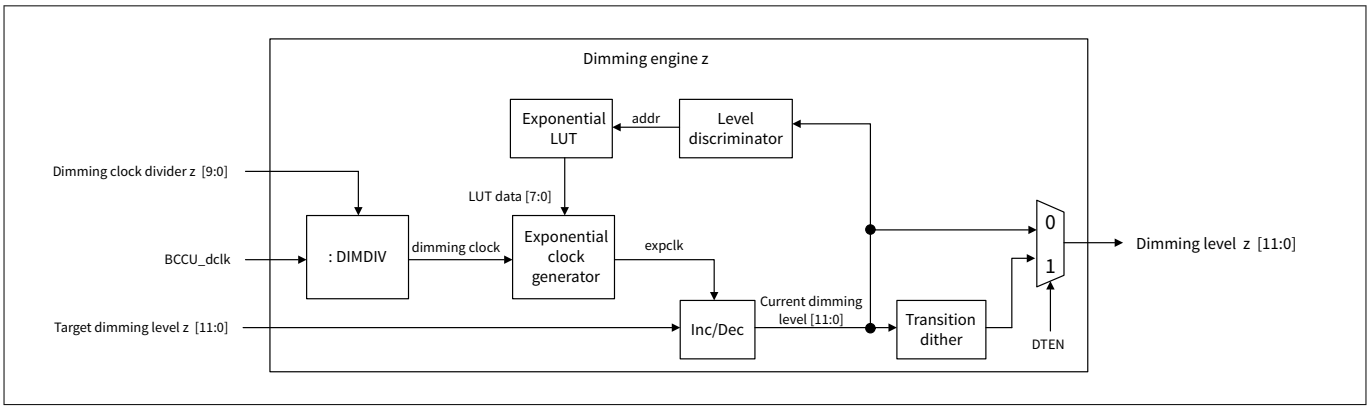


Figure 424 Dimming Engine Block Diagram

Above the level of 16, the ideal exponential dimming curve is described as:

$$\text{DimmingLevel} = 2^{\frac{\text{DimmingClocks} + 4096}{2048}} \tag{79}$$

The actual dimming curve approximates the ideal by combining straight lines forming a piecewise pseudo-exponential curve.

The module provides two curves to choose from, a coarse curve and a fine curve. The desired curve can be selected by BCCU_DTTz (z=0-2).CSEL. The difference is in the number of straight lines used to approximate the ideal exponential curve. Switching between the curves in the application is not supported .

As the dimming level increases along the coarse curve, the slope of the linear pieces doubles every time it passes one of the thresholds of 16, 32, 64, 128, 256, 512, 1024 or 2048.

Table 207 Coarse Piece-Wise Pseudo-Exponential Curve

Upper dimming threshold	Slope of the linear piece	Number of dimming clocks along the line
16	1/256	4096
32	1/128	2048
64	1/64	2048
128	1/32	2048
256	1/16	2048
512	1/8	2048
1024	1/4	2048
2048	1/2	2048
4095	1	2047

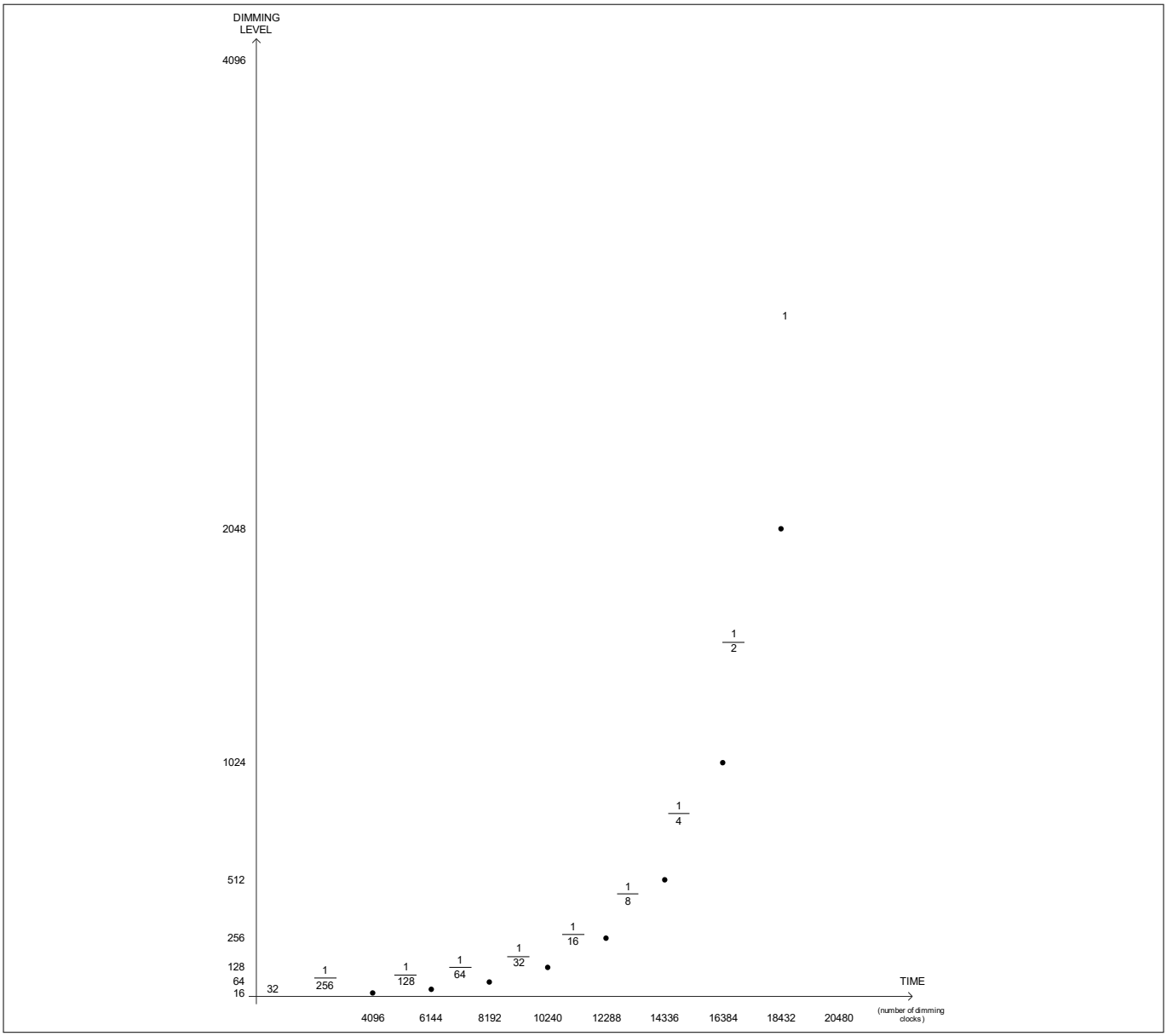


Figure 425 Coarse Piecewise pseudo-exponential curve

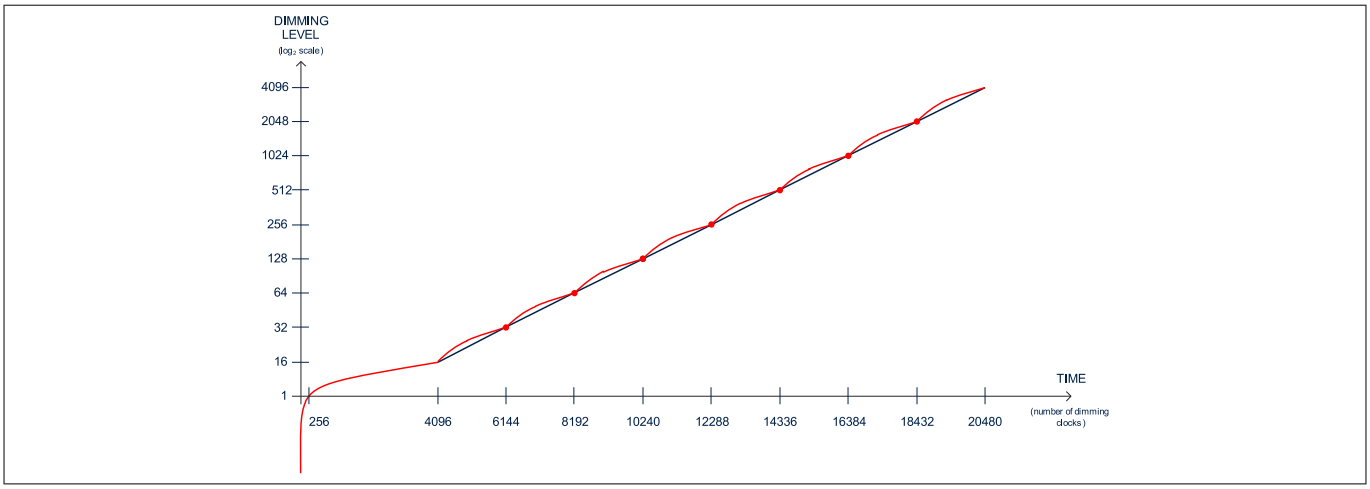


Figure 426 Coarse Piecewise pseudo-exponential curve with the dimming level on a logarithmic scale as the human eye sees it (real and ideal)

This piecewise pseudo-exponential curve is quantized into 4095 steps. When BCCU_DESTRCON.DEzS is set, the first step is immediately taken towards the target and at every dimming level, the dimming engine waits for a certain amount of dimming clocks (“waiting time”) before taking the next step. This waiting time determines the slope of the linear piece. Once the target level has been reached and the appropriate amount of clocks have elapsed, BCCU_DESTRCON.DEzS is cleared by hardware signaling the end of the dimming process. Dimming up from 0 to 4095 takes 20479 dimming clocks (not counting synchronization and bus communication delays).

Table 208 Quantized CoarsePiece-Wise Pseudo-Exponential Curve

Dimming Levels	Clocks before the next step (“waiting time”)	Slope of the linear piece
0..16	256	1/256
17..32	128	1/128
33..64	64	1/64
65..128	32	1/32
129..256	16	1/16
257..512	8	1/8
513..1024	4	1/4
1025..2048	2	1/2
2049..	1	1

For example dimming from 0 to 18 along the coarse curve takes 4352 clocks (16*256 clocks + 2*128 clocks) when measured as the time between setting and clearing BCCU_DESTRCON.DEzS (ignoring delays).

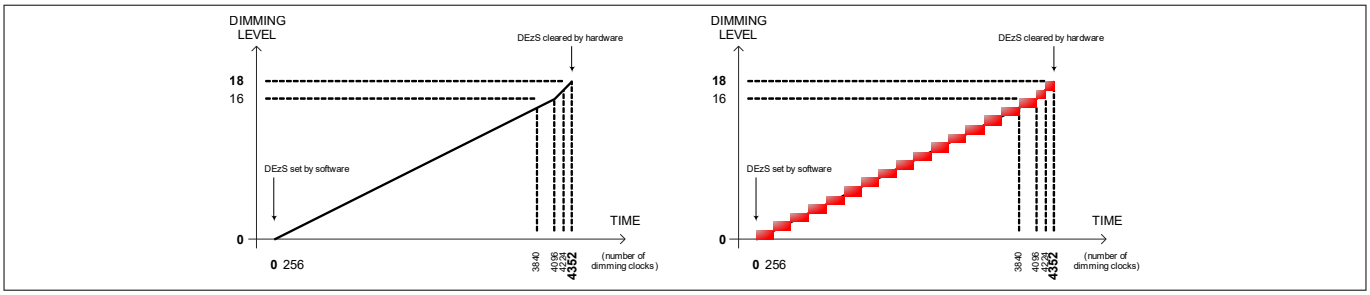


Figure 427 Dimming from 0 to 18 along the coarse curve without and with dither enabled

Exponential clock generation is the same regardless of the direction of the dimming so the waiting time before the next step is the same at a given dimming level whether the next step is an increment or a decrement. Because of this, dimming down takes longer than dimming up with the same settings. The extra time is equivalent to the waiting time of the target level minus the waiting time of the starting level. Dimming down from 4095 to 0 takes 20734 clocks (20479+256-1). Dimming down from 18 to 0 along the coarse curve takes 4480 clocks (4352+256-128).

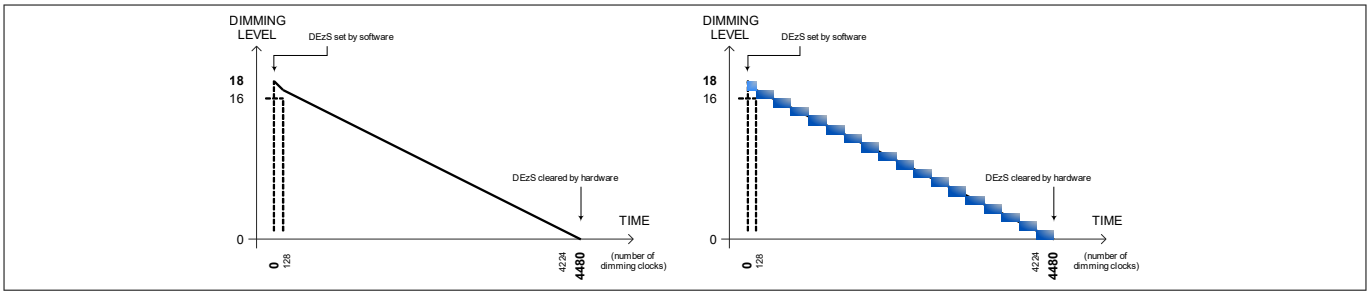


Figure 428 Dimming from 18 to 0 along the coarse curve without and with dither enabled

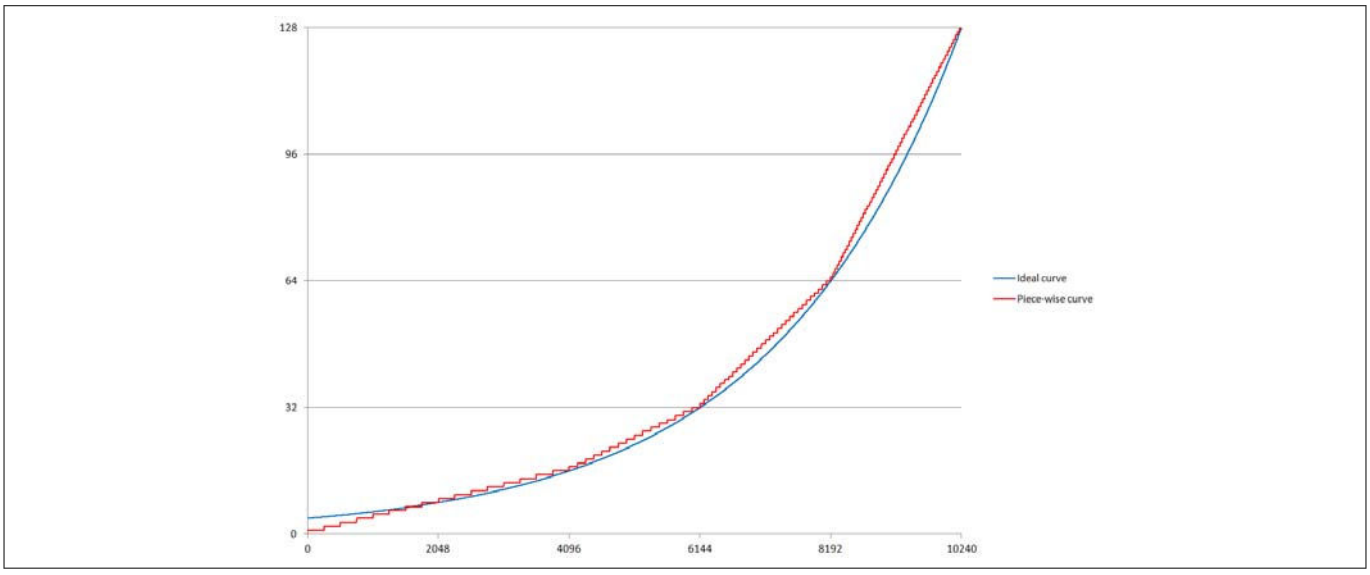


Figure 429 Lower section of the coarse curve (real and ideal)

The fine curve is similar to the coarse one but it has more pieces and different line slopes.

Table 209 Fine Piece-Wise Pseudo-Exponential Curve

Upper dimming threshold	Slope of the linear piece	Number of dimming clocks along the line
16	1/256	4096
18	1/196	392
25	1/132	924
34	1/102	918
41	1/81	567
48	1/64	448
69	1/51	1071
79	1/40	400
107	1/32	896
130	1/25	575
167	1/20	740
203	1/16	576
253	1/13	650
343	1/10	900
512	1/7	1183
1024	1/4	2048
2048	1/2	2048
4095	1	2047

Table 210 Fine Piece-Wise Pseudo-Exponential Curve

Dimming Levels	Clocks before the next step (“waiting time”)	Slope of the linear piece
0..16	256	1/256
17..18	196	1/196
19..25	132	1/132
26..34	102	1/102
35..41	81	1/81
42..48	64	1/64
49..69	51	1/51
70..79	40	1/40
80..107	32	1/32
108..130	25	1/25
131..167	20	1/20
168..203	16	1/16

(table continues...)

Table 210 (continued) Fine Piece-Wise Pseudo-Exponential Curve

Dimming Levels	Clocks before the next step (“waiting time”)	Slope of the linear piece
204..253	13	1/13
254..343	10	1/10
334..512	7	1/7
513..1024	4	1/4
1025..2048	2	1/2
2049..4095	1	1

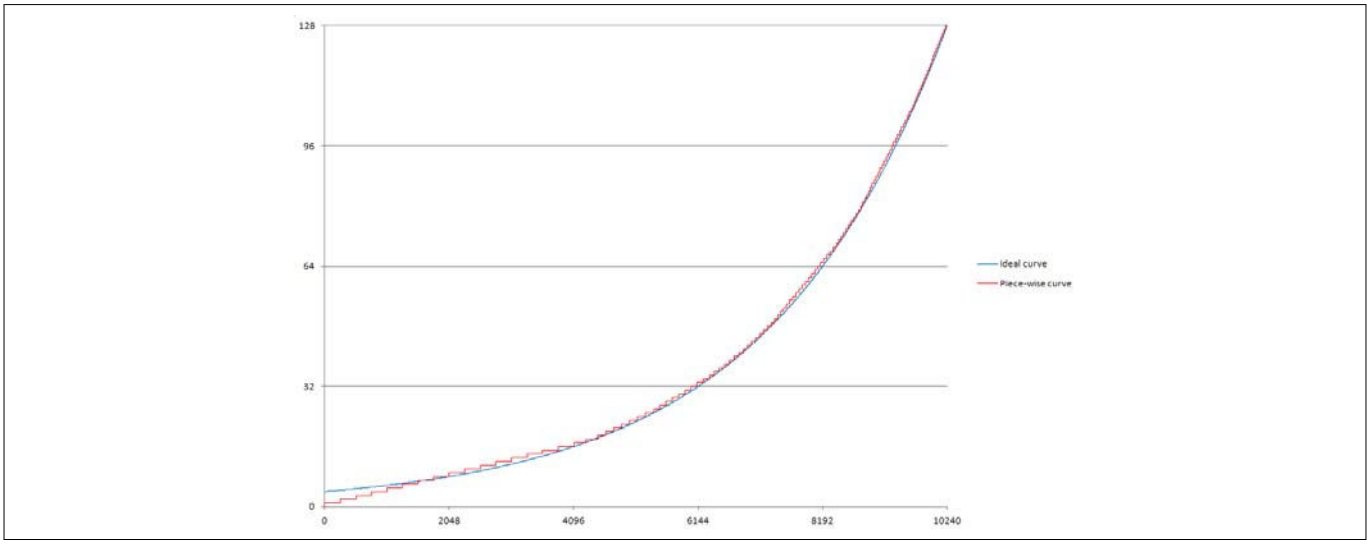


Figure 430 Lower section of the fine curve (real and ideal)

Dimming along the entire dimming curve takes 20479 dimming clocks when dimming up and 20734 clocks when dimming down. The fade time from 0 to 4095 (0% to 100%) is adjustable through the "dimming clock divider" bitfield (BCCU_DTTz (z=0-2).DIMDIV):

$$\mathrm{FadeTime}_{0-100} = \frac{20479}{\left(\frac{f_{BCCUclk}}{DCLKPS \times DIMDIV}\right)} = \frac{20479}{\left(\frac{f_{BCCUdclk}}{DIMDIV}\right)} \quad (80)$$

$$\mathrm{FadeTime}_{100-0} = \frac{20734}{\left(\frac{f_{BCCUclk}}{DCLKPS \times DIMDIV}\right)} = \frac{20734}{\left(\frac{f_{BCCUdclk}}{DIMDIV}\right)} \quad (81)$$

If BCCU_DTTz (z=0-2).DIMDIV is 0, the dimming level will instantly be the same as the target dimming level on shadow transfer, which means that the dimming engine is effectively bypassed.

At dimming levels below 128, where the human eye is potentially very sensitive to changes in brightness, the dimming steps can be dithered to avoid visible discrete steps in brightness. This function can be enabled by setting BCCU_DTTz (z=0-2).DTEN. Only the coarse curve can be used with dithering. Every time the dimming level (BCCU_DLz (z=0-2).DLEV) is supposed to increase or decrease by 1 (“dimming step”), a dither pattern is superimposed on it to make the transition appear smoother to the human eye. The pattern lasts 256 dimming

clocks between dimming levels of 0 and 15, 128 dimming clocks between 16 and 31, 64 dimming clocks between 32 and 63 and 32 dimming clocks between 64 and 127.

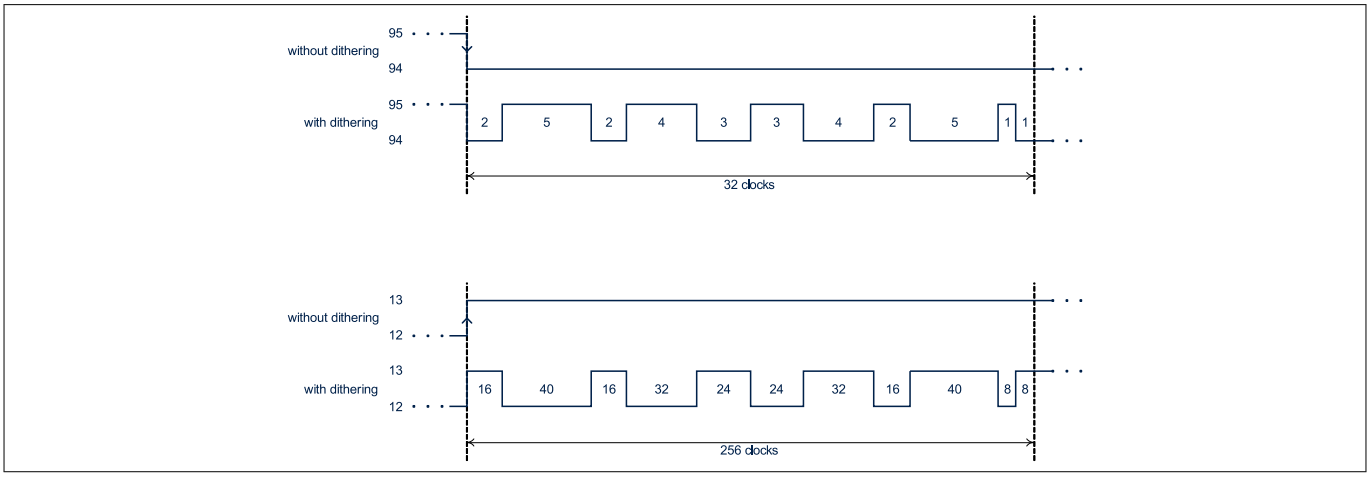


Figure 431 Dithering examples

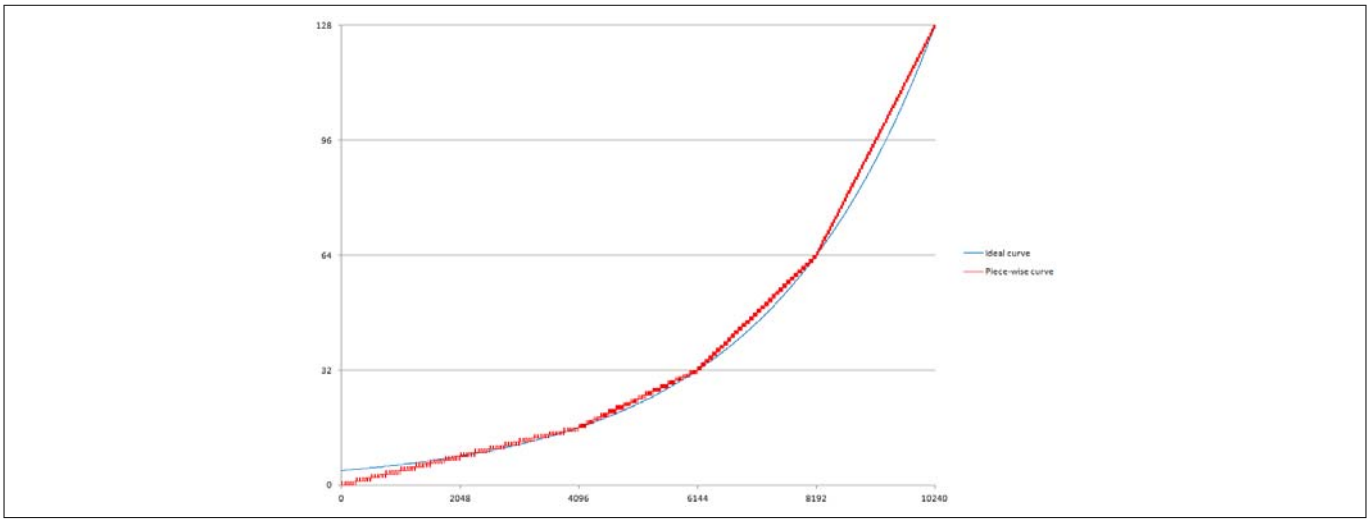


Figure 432 Lower section of the coarse curve with dithering enabled (real and ideal)

Dithering the transition from dimming level of 0 to 1 may cause visible blinking. This can be avoided by enabling the flicker watchdog in the affected channels. The flicker watchdog can be enabled by setting bit CHCONFIGy (y=0-8).WEN.

24.2.3 Linear Color Walk

The intensity level (BCCU_INTy (y=0-8).CHINT) can be changed using a linear walk. The transition time, aka linear walk time, is user adjustable by programming the (BCCU_CHCONFIGy (y=0-8).LINPRES) bitfield. Reaching the target intensity level(BCCU_INTSy y=(0-8).TCHINT) after shadow transfer (setting BCCU_CHSTRCON.CHyS) will take the selected transition time which can be calculated by the following formula:

$$\mathrm{LINEARWALKTIME} = \frac{\mathrm{LINPRES} \times 2^{13}}{f_{\mathrm{BCCUfclk}}} \quad (82)$$

Once the linear walk is done, BCCU_CHSTRCON.CHyS is cleared by hardware. Setting BCCU_CHSTRCON.CHyA will abort the linear walk and will clear BCCU_CHSTRCON.CHyS.

BCCU_INTSy (y=0-8).TCHINT cannot be written while the linear walk is ongoing (BCCU_CHSTRCON.CHyS is set). During this time, write accesses are ignored. The user either has to first wait until the walk is finished or has to manually abort the walk by setting BCCU_CHSTRCON.CHyA.

If a linear walk is previously aborted, the subsequent linear walk starts with a miniscule delay. The maximum delay is one linear clock (LINPRES * 1 fclk cycle).

If BCCU_CHCONFIGy (y=0-8).LINPRES is 0, the intensity level will be the same as the target intensity level on shadow transfer, which means that the linear walker is effectively bypassed.

Smooth color transition can be achieved by setting the same transition time for the respective channels and initiating shadow transfer at the same time. The delta intensity value is calculated as the difference between the channel intensity and the target channel intensity at the time of the channel shadow transfer.

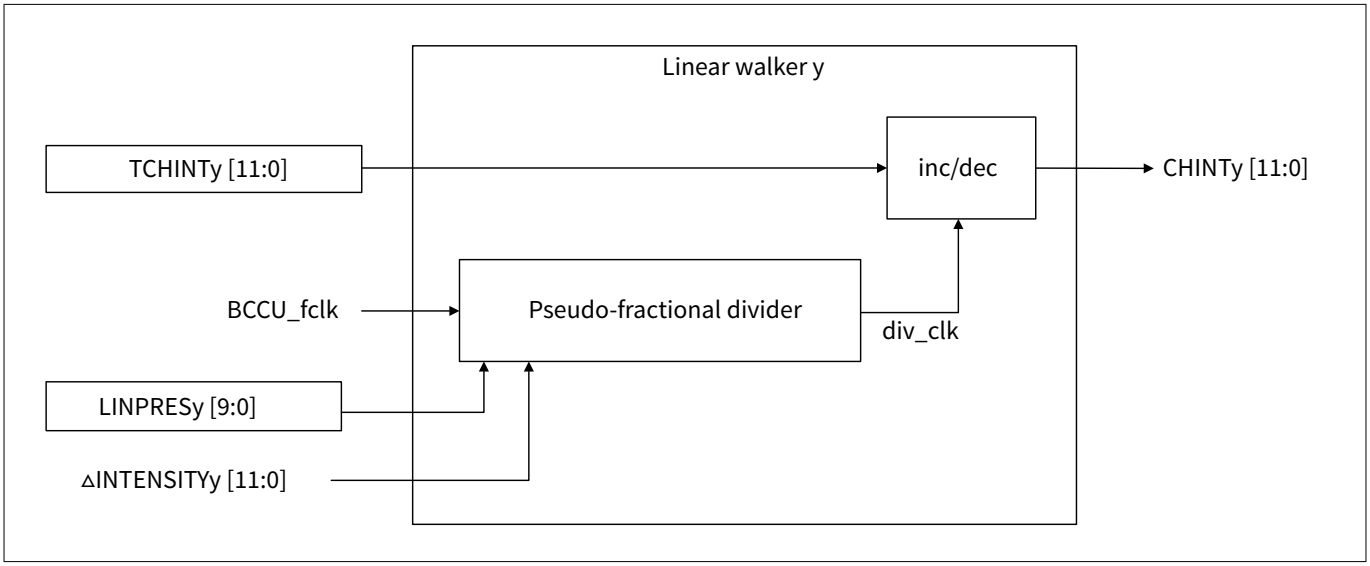


Figure 433 Linear Walker Block Diagram

24.2.4 Sigma-Delta Modulator

The sigma-delta modulator oversamples the slowly changing 12-bit brightness signal and changes it into a high bitrate single-bit signal. The average analog value of the single-bit signal is directly proportional to the 12-bit brightness value. Low-pass filtering is performed by the human eye.

The modulator contains a flicker watchdog which, if enabled, limits the number of consecutive off-bits (0) at the output to avoid visible flickering. The maximum number of off-bits allowed is determined by BCCU_GLOBCON.WDMBN. If the number of consecutive off-bits reaches this threshold, an on-bit (1) is inserted into the bitstream.

The minimum frequency of on bits is defined by the following formula:

$$f_{min} = \frac{1}{WDMBN} \times f_{BCCUbclk} \quad (83)$$

24.2.5 Packer

The main purpose of the packer is to decrease the average rate of switching of the output signal to decrease the load on external switching circuits and improve EMC behavior. The packer contains two 8-bit counters, a 4-stage FIFO and an output generator.

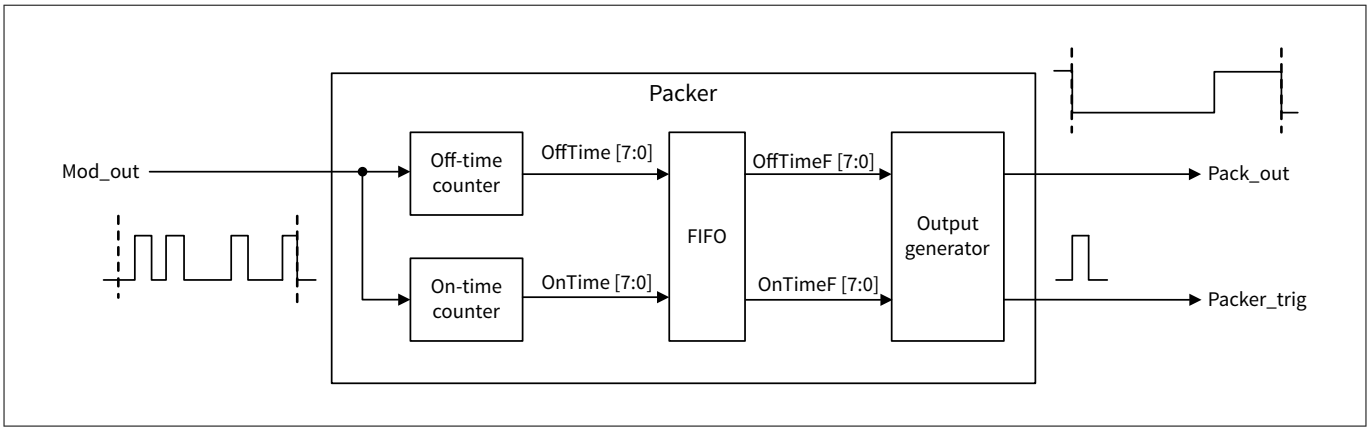


Figure 434 Packer Block Diagram

The on-time counter (BCU_PKCNTRy (y=0-8).ONCNTVAL) counts the on-bits of the input bit stream (Mod_out) and the off-time counter (BCU_PKCNTRy (y=0- 8).OFFCNTVAL) counts the off bits. If the off-time counter or the on-time counter reaches the user adjustable compare level (BCCU_PKCMPy (y=0-8).ONCMP or BCCU_PKCMPy (y=0-8).OFFCMP), both timers reset and start counting again. Before the timers reset, their counts (BCCU_PKCNTRy (y=0-8).ONCNTVAL and BCCU_PKCNTRy (y=0-8).OFFCNTVAL) are loaded into a queue (FIFO). The counts are taken one by one from the queue in an alternating manner by the output generator and a packed off-on signal packet is generated (Pack_out). If the queue is empty, the output generator generates an off-bit (0) for one bit clock before checking the FIFO again. On microcontroller reset or if the channel has been disabled and enabled again, all queue stages are empty. When the packer becomes enabled (BCCU_CHCONFIGy (y=0-8).PEN), the output generator only starts taking the values from the queue once enough stages are filled (determined by BCCU_CHCONFIGy (y=0-8).PKTH) . Until that, only off-bits are generated at the output.

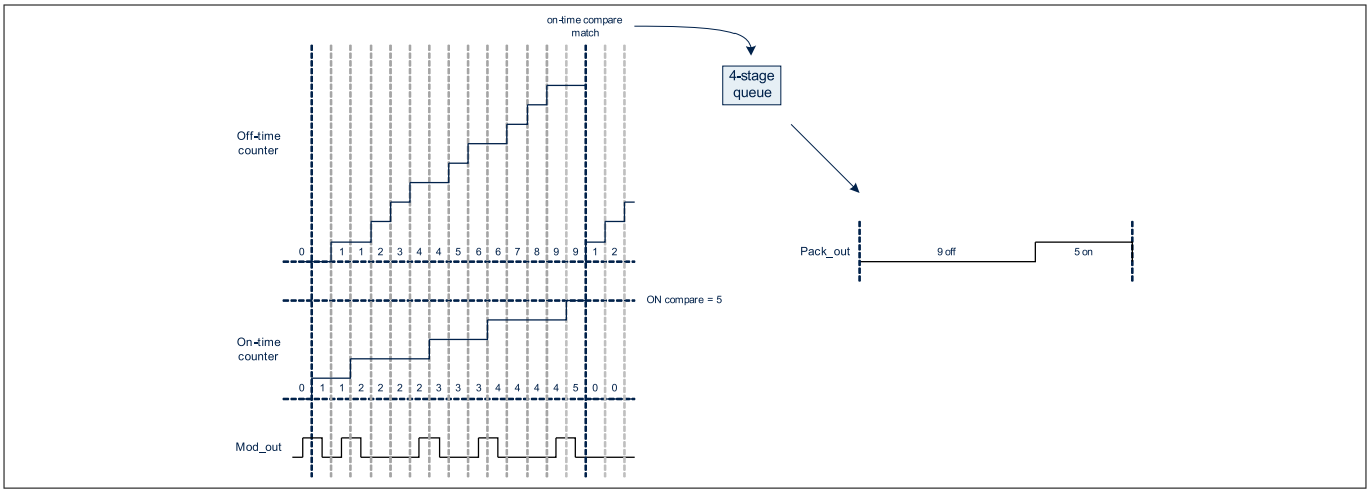


Figure 435 Signal Packing (example)

BCCU_PKCNTRy (y=0-8).ONCNTVAL and BCCU_PKCNTRy (y=0-8).OFFCNTVAL can be manually updated while the channel is not enabled (BCCU_CHEN.ECHy is 0).By putting a starting value other than zero in the counters, the user can introduce a “phase shift” between different channels to avoid identical behavior.

The packed signal has fixed on-time if the brightness level is high enough so that the on- time compare value is reached. Below a certain brightness level, the off-time compare value is reached before the on-time compare value. In this case the packed signal has a fixed off-time. The compare level determines the length of the packed signal packets which can be calculated with the following formula:

$$\mathrm{PacketTime_{min}} = \mathrm{ONCMP} \times \frac{1}{\mathrm{f_{BCCUbclk}}} \quad (84)$$

$$\text{PacketTime}_{\text{max}} = (\text{ONCMP} + \text{OFFCMP}) \times \frac{1}{f_{\text{BCCUbclk}}} \quad (85)$$

24.2.6 Global Trigger Control

The BCCU generates two trigger signals to the ADC (BCCU_TRIGOUT0 and BCCU_TRIGOU1) to start conversions in a synchronized manner. In certain use cases, ADC samples are only meaningful if the respective control signal is active (for example, the respective LED string is on). The output state of the BCCU channels when the last trigger occurred can be observed by checking the "Last Trigger Channel Output Level Register" (BCCU_LTCHOL). The channels can be individually enabled in the "Channel Enable Register" (BCCU_CHEN.ECHy). Triggering can be individually enabled for every channel in the “Channel Trigger Register” (BCCU_CHTRIG.ETy). Only enabled channels with enabled triggering participate in the triggering mechanism. Trigger occurs on a passive-to-active or active-to-passive transition of the channel output which can be individually selected for every channel (BCCU_CHCONFIGy (y=0-8).TRED).

The BCCU offers two trigger modes (BCCU_GLOBCON.TM). In mode 0, any channel trigger causes a trigger to the ADC on both trigger signals. When that happens, the ADC is expected to sample every channel and the user software has to decide which results are valid based on BCCU_LTCHOL.

In mode 1, only one channel trigger generates an ADC trigger at a time. The channels take turns in trigger generation in a round robin manner starting from 0 and counting up. The trigger can be generated either on BCCU_TRIGOUT0 or on BCCU_TRIGOUT1 which is selectable for each channel individually by BCCU_CHTRIG.TOSy. When a trigger has been generated, the channel pointer will point to the next participating channel to generate the next trigger. Every trigger will cause only one ADC measurement. Some additional user software may be necessary to control which ADC channel is triggered at a given time. BCCU_GLOBCON.LTRS shows which BCCU channel generated the last trigger. The channel pointer is reset and will point to channel 0 if all channel trigger enable bits (BCCU_CHTRIG.ETy) are 0.

The trigger can optionally be delayed by a quarter bit time or half a bit time to avoid sampling during switching noise. This is selectable by the BCCU_GLOBCON.TRDEL bitfield. This delay is only functional if BCCU_bclk is generated in normal mode (BCCU_GLOBCLK.BCS is 0).

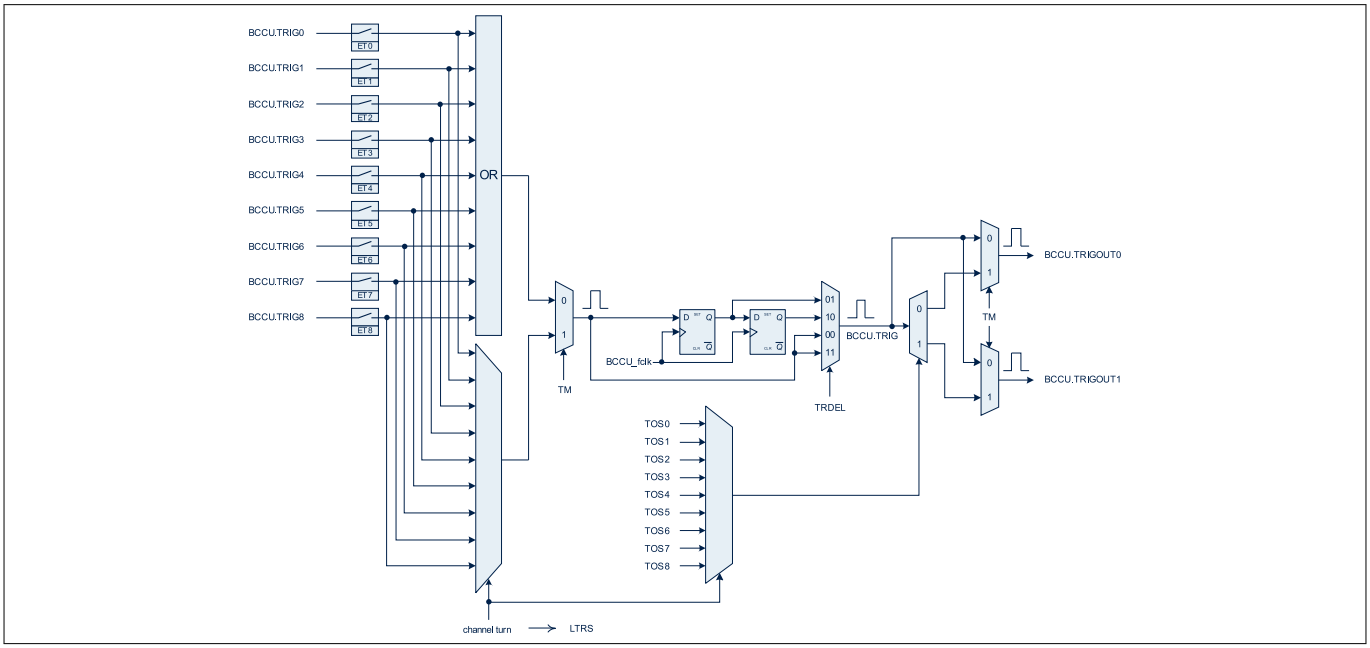


Figure 436 BCCU Trigger Generation

24.2.7 Trap Control

The trap functionality allows the BCCU outputs to react to a state of an input pin. This functionality can be used to switch off the connected power devices if the trap input becomes active (i.e. a trap occurs). A trap is a global event but trap functionality can be enabled for every channel individually by setting the BCCU_CHOCON.CHxTPE bits. If the corresponding bit is not set, the given channel will ignore the trap event.

A trap event occurs if a rising or a falling edge (user selectable via BCCU_GLOBCON.TRAPED) is detected on the BCCU.TRAPL signal. When a trap occurs, the affected channel outputs immediately [24] go to their respective passive levels, determined by BCCU_CHOCON.CHyOP, while the event flag (BCCU_EVFR.TPF) and the trap state flag (BCCU_EVFR.TPSF) bits will be set.

BCCU_EVFR.TPF can be reset by writing 1 to BCCU_EVFCR.TPFC. This operation has no effect on the channel outputs. BCCU_EVFR.TPSF can be reset by writing 1 to BCCU_EVFCR.TPC. This will clear the trap and the channel outputs will go back to their respective output levels based on the channels' conditions. The user can monitor the level of BCCU.TRAPL via bit BCCU_EVFR.TPINL and may only want to exit the trap state once the trap signal is inactive.

BCCU_EVFR.TPSF can also be set by writing 1 to BCCU_EVFSR.TPS to generate a software trap.

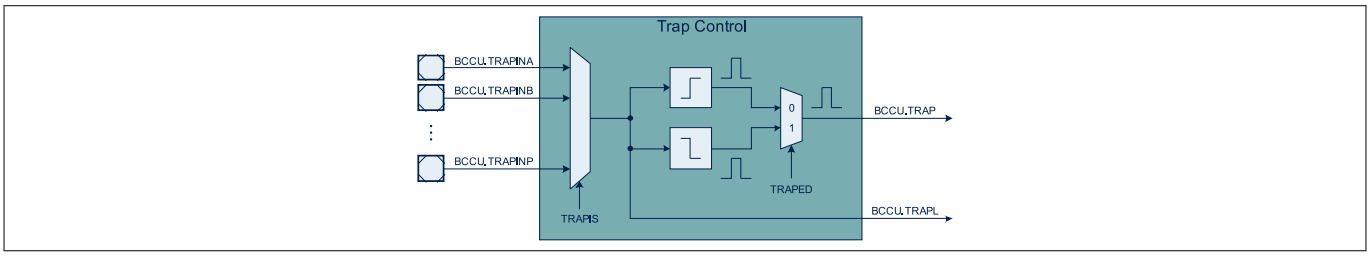


Figure 437 Trap Control

24.3 Power, Reset and Clock

BCCU is located in the core power domain. The module, including all registers other than the bit field BCCU_GLOBCON.SUSCFG, can be reset to its default state by a system reset. The bit field BCCU_GLOBCON.SUSCFG is reset to its default value only by a debug reset.

***Note**: In PSC1Mx, as there is no separate debug reset, the bit field BCCU_GLOBCON.SUSCFG will be reset to its default state by a system reset.*

The BCCU kernel is clocked (BCCU_clk) and accessible on the peripheral bus frequency (PCLK). Three main clocks are generated for the different blocks in the module:

- BCCU_bclk (bit clock) is the clock that determines the sampling rate of the sigma-delta converter and the bit time of the signal it generates. Depending on BCCU_GLOBCLK.BCS, BCCU_bclk is generated from BCCU_fclk via a 4 divider or it is the same as BCCU_fclk.
- BCCU_fclk (fast clock) is used by the Trigger Control block to generate the trigger delay. It is generated from BCCU_clk by a prescaler. The frequency can be set by BCCU_GLOBCLK.FCLK_PS.
- BCCU_dclk (dimming engine clock) is the input clock to the dimming engines. It is generated from BCCU_clk by a prescaler. The frequency can be set by BCCU_GLOBCLK.DCLK_PS.

[24] After a synchronization delay of 4 BCCU_clk cycles

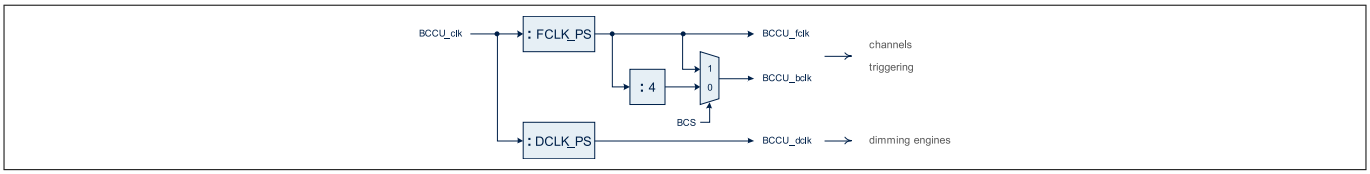


Figure 438 BCCU clocks

The clock frequencies in normal mode can be calculated with the following formula:

$$f_{BCCUfclk} = \frac{1}{FCLKPS} \times f_{BCCUclk} \quad (86)$$

$$f_{BCCUbclk} = \frac{1}{FCLKPS \times 4} \times f_{BCCUclk} \quad (87)$$

$$f_{BCCUdclk} = \frac{1}{DCLKPS} \times f_{BCCUclk} \quad (88)$$

The kernel is only in operation in active mode.

24.4 Service Request Generation

The BCCU has one service request node and five interrupts. The five interrupts are 1) Trigger 0 event, 2) Trigger 1 event, 3) Packer FIFO full event, 4) Packer FIFO empty event and 5) Trap event.

A trigger 0/1 event occurs if there is a trigger pulse on BCCU.TRIGOUT0/1 to the ADC. A packer FIFO full/empty event occurs if any of the packer FIFOs is full/empty . A trap event occurs if a rising or a falling edge is detected on the BCCU.TRAPL signal.

The five interrupts can be generated by a hardware event or by setting the respective bit in BCCU_EVFSR. All five interrupts are assigned to node 0 and have their interrupt enable bit in BCCU_EVIER and their flag bit in BCCU_EVFR. The flag bits are set by the hardware event or by the respective set bits in BCCU_EVFSR and cleared by the respective clear bits in BCCU_EVFCR.

Additionally, the trigger events (BCCU.TRIGOUT0 and BCCU_TRIGOUT1) are connected to the trigger inputs of the ADC.

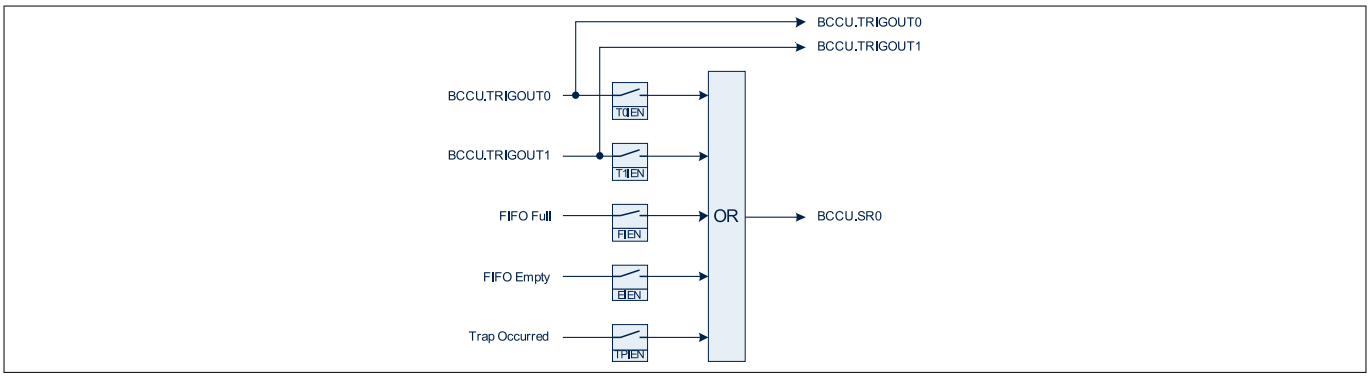


Figure 439 Service Request Nodes

The module clock is disabled by default and can be enabled via the SCU_CGATCLR0 register. Enabling and disabling the module clock could cause a load change and clock blanking could occur as described in the CCU

(Clock Gating Control) section of the SCU chapter. It is strongly recommended to set up the module clock in the user initialization code to avoid clock blanking during runtime.

24.5 Debug Behaviour

The clocks to all channels, dimming engines and other functional blocks are stopped in suspend mode. The registers can still be accessed by the CPU as read-only. This mode is useful for debugging purposes as the current device status is frozen to get a snapshot of the internal values. As the clocks are stopped, all counters and timers are stopped and frozen.

Mode of entry into suspend mode can be configured by changing the BCCU_GLOBCON.SUSCFG bitfield.

When debug suspend is revoked, the kernel resumes operation according to latest SFR settings.

***Note**: In PSC1Mx, bit field BCCU_GLOBCON.SUSCFG is reset to its default value by any reset. If the suspend function is required during debugging, it is recommended that it is enabled in the user initialization code. Before programming the bit field, the module clock has to be enabled and special care needs to be taken while enabling the module clock as described in the CCU (Clock Gating Control) section of the SCU chapter.*

24.6 Initialization

The BCCU registers are recommended to be programmed in a specific sequence.

1. **Program Global Registers**
   - BCCU_CHOCON to set output passive and active levels
   - BCCU_GLOBCLK to set up the clocks
   - BCCU_GLOBCON for general configuration
   - BCCU_EVIER to set up interrupts
   - BCCU_GLOBDIM for a user-defined dimming level when no dimming engine is used
   - BCCU_CHTRIG to set up ADC triggering
2. **Program Channel Registers**
   - BCCU_CHCONFIGy (y=0-8) for general channel configuration
   - BCCU_PKCMPy (y=0-8) if the respective packer is used
   - BCCU_PKCNTRy (y=0-8) to introduce a phase shift between channels. Ensure that the packer is already enabled (BCCU_CHCONFIGy (y=0-8).PEN is 1) before writing to packer counter register.
3. **Program Dimming Engine Registers**
   a. BCCU_DTTz (z=0-2) to adjust the dimming curve
4. **Enable channels and dimming engines**
   a. BCCU_CHEN to enable channels
   b. BCCU_DEEN to enable dimming engines
5. **Set target intensities and start linear walk**
   a. BCCU_CHCONFIGy (y=0-8).LINPRES to set the linear walk time
   b. BCCU_INTSy (y=0-8) to set the target intensities
   c. BCCU_CHSTRCON to start linear walks
   d. When the linear walks are finished, BCCU_CHSTRCON.CHyS are cleared by hardware
6. **Set target dimming levels and start dimming process (if at least one dimming engine is used)**
   a. BCCU_DTTz (z=0-2).DIMDIV to set the fade rate
   b. BCCU_DLSz (z=0-2) to set the target target dimming levels

c. BCCU_DESTRCON to start dimming
d. When the dimming processes are completed, BCCU_DESTRCON.DEzS are cleared by hardware

24.7 Digital-to-Analog Converter

The BCCU can be used as a simple multi-channel Digital-to-Analog Converter (DAC) with 12-bit resolution as well. The BCCU.OUTy signals can control the pull devices of certain port pins. An RC filter can be realized by adding an external capacitor.

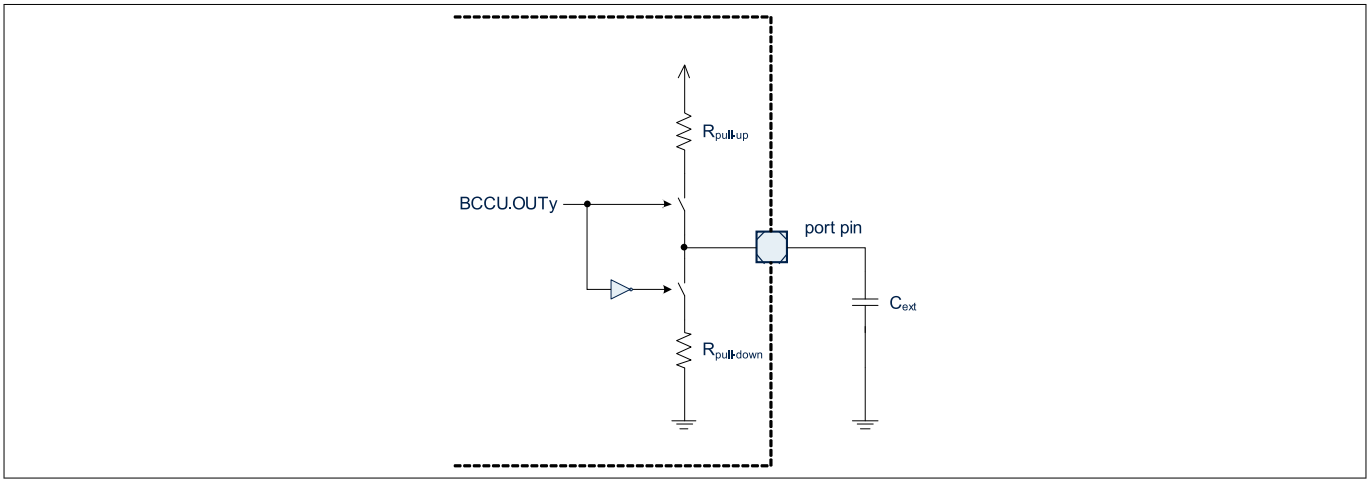


Figure 440 Simple DAC on one pin

To use the BCCU in a DAC configuration, the user must ensure that the HW0 control path is selected for the port pins used. This can be done by setting the selected Pn_HWSEL.HWx bitfields to 01_B. Additionally, the used pads need to be enabled by resetting the respective bits in register P2_PDISC. This will configure the port pins as digital input pins and can still be used by other modules. One example is generating an analog voltage reference for the in-built analog comparator.

It is recommended to select a high bit clock frequency by setting the fast clock prescaler (BCCU_GLOBCLK.FCLK_PS) to a low value and by selecting fast mode (BCCU_GLOBCLK.BCS to 1). The selected BCCU channels should bypass the dimming engines (BCCU_CHCONFIGy (y=0-8).DBP is 1) and the flicker watchdog should be disabled (BCCU_CHCONFIGy (y=0-8).WEN to 0).

The generated voltage level can be controlled with the 12-bit channel intensity value (BCCU_INTy (y=0-8).CHINT) by updating the target channel intensity (BCCU_INTSy (y=0-8).TCHINT) and initiating a shadow transfer (setting BCCU_CHSTRCON.CHyS to 1).

24.8 Registers

All BCCU registers (except bit field BCCU_GLOBCON.SUSCFG) are reset by a system reset. Bit field BCCU_GLOBCON.SUSCFG is reset by a debug reset.

Registers Overview

The absolute register address is calculated by adding:

Module Base Address + Offset Address

Table 211 Registers Address Space

Module	Base Address	End Address	Note
BCCU	50030000_H	$5003FFFF_H$	

Table 212 Register Overview

Register Short Name	Description	Offset Addr.	Access Mode	
			Read	Write
BCCU Global Registers				
GLOBCON	Global Control Register	0000_H	U, PV, 32	U, PV, 32
GLOBCLK	Global Clock Register	0004_H	U, PV, 32	U, PV, 32
ID	Module Identification Register	0008_H	U, PV, 32	BE
CHEN	Channel Enable Register	$000C_H$	U, PV, 32	U, PV, 32
CHOCON	Channel Output Control Register	0010_H	U, PV, 32	U, PV, 32
CHTRIG	Channel Trigger Register	0014_H	U, PV, 32	U, PV, 32
CHSTRCON	Channel Shadow Transfer Control Register	0018_H	U, PV, 32	U, PV, 32
LTCHOL	Last Trigger Channel Output Level Register	$001C_H$	U, PV, 32	BE
DEEN	Dimming Engine Enable Register	0020_H	U, PV, 32	U, PV, 32
DESTRCON	Dimming Engine Shadow Transfer Control Register	0024_H	U, PV, 32	U, PV, 32
GLOBDIM	Global Dimming Level Register	0028_H	U, PV, 32	U, PV, 32
EVIER	Event Interrupt Enable Register	$002C_H$	U, PV, 32	U, PV, 32
EVFR	Event Flag Register	0030_H	U, PV, 32	BE
EVFSR	Event Flag Set Register	0034_H	U, PV, 32	U, PV, 32
EVFCR	Event Flag Clear Register	0038_H	U, PV, 32	U, PV, 32
BCCU Channel Registers				
INTSy	Channel Intensity Shadow Register y	$003C_H + y*0014_H$	U, PV, 32	U, PV, 32
INTy	Channel Intensity Register y	$0040_H + y*0014_H$	U, PV, 32	BE
CHCONFIGy	Channel Configuration Register y	$0044_H + y*0014_H$	U, PV, 32	U, PV, 32
PKCMPy	Packer Compare Register y	$0048_H + y*0014_H$	U, PV, 32	U, PV, 32
PKCNTRy	Packer Counter Register y	$004C_H + y*0014_H$	U, PV, 32	U, PV, 32
Reserved	Reserved	$00F0_H$ - 0178_H	BE	BE
BCCU Dimming Engine Registers				
DLSz	Dimming Level Shadow Register z	$017C_H + z*000C_H$	U, PV, 32	U, PV, 32
DLz	Dimming Level Register z	$0180_H + z*000C_H$	BE	U, PV, 32
DTTz	Dimming Transition Time Register z	$0184_H + z*000C_H$	U, PV, 32	U, PV, 32
Reserved	Reserved	$01A0_H$ - $FFFC_H$	BE	BE

24.9 Interconnects

BCCU0 is connected to the ports, the ERU, the analog comparator, the ADC, the NVIC, CCU4, the suspend signal and the kernel clock.

Table 213 BCCU0 Pin Connections

Global Input/Output	I/O	Connected To	Description
BCCU0.clk	I	PCLK	BCCU kernel clock
BCCU0.SUSPEND	I	CPU Halted	Suspend signal
BCCU0.IN0	I	ACMP1.OUT	Gating input of channel 0
BCCU0.IN1	I	ACMP3.OUT	Gating input of channel 1
BCCU0.IN2	I	ACMP2.OUT	Gating input of channel 2
BCCU0.IN3	I	ACMP2.OUT	Gating input of channel 3
BCCU0.IN4	I	ACMP2.OUT	Gating input of channel 4
BCCU0.IN5	I	ACMP0.OUT	Gating input of channel 5
BCCU0.IN6	I	ACMP0.OUT	Gating input of channel 6
BCCU0.IN7	I	ACMP1.OUT	Gating input of channel 7
BCCU0.IN8	I	ACMP3.OUT	Gating input of channel 8
BCCU0.TRAPINA	I	P0.12	Trap input A
BCCU0.TRAPINB	I	P0.0	Trap input B
BCCU0.TRAPINC	I	P3.0	Trap input C
Reserved	I	Reserved	Reserved
BCCU0.TRAPINE	I	ERU0.IOUT0	Trap input E
BCCU0.TRAPINF	I	ERU0.IOUT1	Trap input F
BCCU0.TRAPING	I	ERU0.IOUT2	Trap input G
BCCU0.TRAPINH	I	ERU0.IOUT3	Trap input H
BCCU0.TRAPINI	I	SCU.SR2	Trap input I
BCCU0.TRAPINJ	I	ERU1.IOUT0	Trap input J
BCCU0.TRAPINK	I	ERU1.IOUT1	Trap input K
BCCU0.TRAPINL	I	ERU1.IOUT2	Trap input L
BCCU0.TRAPINM	I	ERU1.IOUT3	Trap input M
BCCU0.TRAPINN	I	0	Trap input N
BCCU0.TRAPINO	I	0	Trap input O
BCCU0.TRAPINP	I	0	Trap input P

(table continues...)

Table 213 (continued) BCCU0 Pin Connections

Global Input/Output	I/O	Connected To	Description
BCCU0.OUT0	O	P0.4; P1.0; P3.0; P4.4; CCU40.IN0AQ; CCU80.IN0AI; CCU41.MCI0; CCU41.IN0AQ; CCU81.IN0AI; P2.2.HW0 pull control;	Output of channel 0
BCCU0.OUT1	O	P0.5; P1.5; CCU40.IN3AZ; CCU80.IN0AJ; CCU41.MCI1; CCU41.IN1AQ; CCU81.IN2AJ; CCU80.IN2AQ; P2.0.HW0 pull control; P2.8.HW0 pull control;	Output of channel 1
BCCU0.OUT2	O	P0.6; P1.6; P4.6; CCU40.MCI0; CCU80.IN1AI; CCU80.IN0AQ; CCU41.MCI2; CCU41.IN2AQ; CCU81.IN3AJ; P1.0.HW0 pull control; P2.6.HW0 pull control;	Output of channel 2

(table continues...)

Table 213 (continued) BCCU0 Pin Connections

Global Input/Output	I/O	Connected To	Description
BCCU0.OUT3	O	P0.7; P2.12; P4.9; CCU40.MCI1; CCU40.IN1AZ; CCU80.IN1AJ; CCU41.IN1AZ; P1.1.HW0 pull control; P2.12.HW0 pull control;	Output of channel 3
BCCU0.OUT4	O	P0.8; P2.13; P4.2; CCU40.MCI2; CCU40.IN2AQ; CCU80.IN2AI; CCU81.IN1AI; P1.2.HW0 pull control; P2.10.HW0 pull control; P2.13.HW0 pull control;	Output of channel 4
BCCU0.OUT5	O	P0.9; P4.3; P4.7; CCU40.MCI3; CCU40.IN3AQ; CCU80.IN2AJ; CCU41.MCI3; CCU41.IN3AQ; CCU81.IN2AI; P1.3.HW0 pull control; P2.11.HW0 pull control;	Output of channel 5
BCCU0.OUT6	O	P0.10; P0.12; CCU40.IN0AZ; CCU80.IN3AI; CCU41.IN0AZ; CCU81.IN0AJ; P1.4.HW0 pull control; P2.1.HW0 pull control;	Output of channel 6

(table continues...)

Table 213 (continued) BCCU0 Pin Connections

Global Input/Output	I/O	Connected To	Description
BCCU0.OUT7	O	P0.11; P0.14; P4.8; CCU40.IN2AZ; CCU80.IN3AJ; CCU80.IN1AQ; CCU41.IN2AZ; CCU81.IN1AJ; P1.5.HW0 pull control; P2.9.HW0 pull control;	Output of channel 7
BCCU0.OUT8	O	P0.1; P0.15; P4.1; P4.5; CCU40.IN1AQ; CCU80.IN3AQ; CCU41.IN3AZ; CCU81.IN3AI; P1.6.HW0 pull control; P2.4.HW0 pull control; P2.7.HW0 pull control;	Output of channel 8
BCCU0.SR0	O	NVIC	Service Request Node 0: Interrupt
BCCU0.TRIGOUT0	O	VADC0.BGREQTRF; VADC0.G0REQTRF;	ADC trigger signal 0
BCCU0.TRIGOUT1	O	VADC0.G1REQTRF	ADC trigger signal 1

25 General Purpose I/O Ports (Ports)

The PSC1Mx has up to 42 digital General Purpose Input/Output (GPIO) port lines which are connected to the on-chip peripheral units.

25.1 Overview

The Ports provide a generic and very flexible software and hardware interface for all standard digital I/Os. Each Port slice has individual 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 214 gives an overview of the available PORTS and other pins in the different packages of the PSC1Mx:

Table 214 Port/Pin Overview

Port	48	40	38	Note
P0	16	16	16	Standard bi-directional pad
P1	7	7	6	High current and Standard bi-directional pad
P2	14	12	12	Analog/Digital input and bi-directional pad
P3	1	0	0	Standard bi-directional pad
P4	4	0	0	Standard bi-directional pad
Supply VDD, ADC/ORC Reference Voltage	1	1	1	VDD
Supply GND, ADC Reference Ground	1	1	1	VSSP/VSS
I/O Port Supply	3	2	1	VDDP
I/O Port Ground	1	1	1	VSSP
Exposed Die Pad	1	1	0	Must be connected to common V_{SS}

25.1.1 Features

This is a list of the main features of the Ports:

- same generic register interface for each port pin, Registers
- simple and robust software access for general purpose I/O functionality, GPIO and Alternate Functions
- direct input connections to on-chip peripherals, Input Operation
- dedicated hardware interface for BCCU, CCU4, ACMP, and USIC with select option, Hardware Controlled I/Os
- defined power-up/power-fail behavior, Power, Reset and Clock
- up to nine alternate output paths from peripherals selectable
- programmable open-drain or push-pull output driver stage
- programmable weak pull-up and pull-down devices
- programmable input inverter
- programmable pad hysteresis
- disabling of digital input stage on shared analog inputs

- separate set and clear output control to avoid read-modify-write operations
- programmable power-save behavior in Deep-Sleep mode
- Privileged Mode restricted access to configuration registers to avoid accidental modification

25.1.2 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 441.

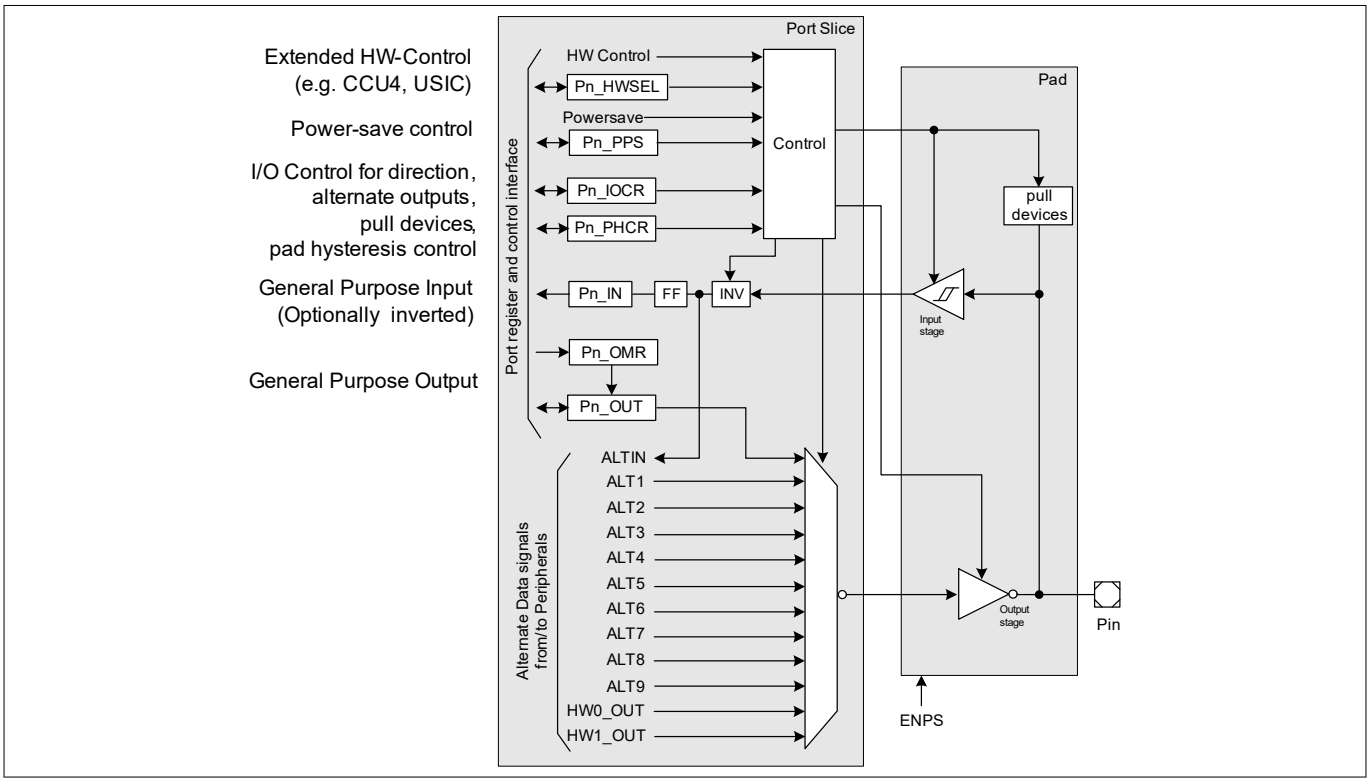


Figure 441 General Structure of a digital Port Pin

25.1.3 Definition of Terms

Some specific terms are used throughout this chapter:

- **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).
- **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.

25.2 GPIO and Alternate Functions

The Ports can be operated as General Purpose Input/Output (GPIO) and with Alternate Functions of the on-chip periphery, configured by the Port Input/Output Control Register (Pn_IOCR). 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 and/or defining the output value by the Port Output Modification Register Pn_OMR. Output modification by Pn_OMR is preferred over the direct change of the output value with the Port Output Register Pn_OUT, 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 Function Description chapter. For specific functions, certain peripherals may also take direct control of “their” port pins, see Hardware Controlled I/Os.

25.2.1 Input Operation

As an input, the actual voltage level at the port pin is translated into a logical 0B or 1B 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). Alternatively, the input can be connected to the multiple on-chip peripherals via the ALTIN signal. Where necessary, these peripherals have internal controls to select the appropriate port pin with an input multiplexer stage, and will take care of synchronization and further processing of the input signals. (See respective peripheral chapters for more details on the input selection and handling). With the Pn_IOCR register, it is also possible to activate an internal weak pull-up or pull-down device in the pad.

The input register Pn_IN and ALTIN signal always represent the state of the input, independent whether the port pin is configured as input or output. This means that 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.

Pad Hysteresis Control

The pad hysteresis can be configured according to the application needs via the Pad Hysteresis Control register (Pn_PHCR). Selecting the appropriate pad hysteresis allows optimized pad oscillation behavior for touch-sensing applications.

25.2.2 Output Operation

In output mode, the output driver is activated and drives the value supplied through the multiplexer to the port pin. Switching between input and output mode is accomplished through the Pn_IOCR register (Port Input/ Output Control Register), 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, Port Output Register)
  - alternate peripheral functions, ALT1...ALT9
- hardware control, Pn_HWSEL
  - HW0_OUT
  - HW1_OUT

***Note**: It is recommended to complete the Port and peripheral configuration with respect to operating mode and initial values before the port pin is switched to output mode.*

The output function is exclusive, meaning that only one peripheral has control of the output path at any one time.

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 (for example, 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, Power Saving Mode Operation.

Because of that, it is preferred to modify Pn_OUT bits by the Output Modification Register Pn_OMR (Port Output Modification Register). 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 ALT9 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 by 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).

There are three pad types in PSC1Mx providing different drive strength or with oscillator function:

- Standard pad (with oscillator function)
- Standard pad (without oscillator function)
- High Current pad

The assignment of each port pin to one of these pad types is listed in the Package Pin Summary table. Further details about pad properties are summarized in the Datasheet. Oscillator functions and controls are described in the Clock Control Unit (CCU) of SCU chapter.

25.3 Hardware Controlled I/Os

Some ports pins are overlaid with peripheral functions for which the connected peripheral needs direct hardware control, for example, 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 (Port Pin Hardware Select Register) allows to select between the hardware “masters”.

Depending on the operating mode, the peripheral can take control of various functions:

- Pin direction, input or output, for example, 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 pad hysteresis by Pn_PHCR 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 when to 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 default hardware input configuration and the pull devices are controlled by Pn_IOCR.

The outputs of BCCU, CCU4 and ACMP modules can be used to control the internal pull devices via direct hardware control. A detailed connectivity list of the hardware I/O and pull control of the peripherals to the port pins is given in the Hardware Controlled I/O Function Description chapter.

***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.*

25.4 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). 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 442). The input signal to the 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.

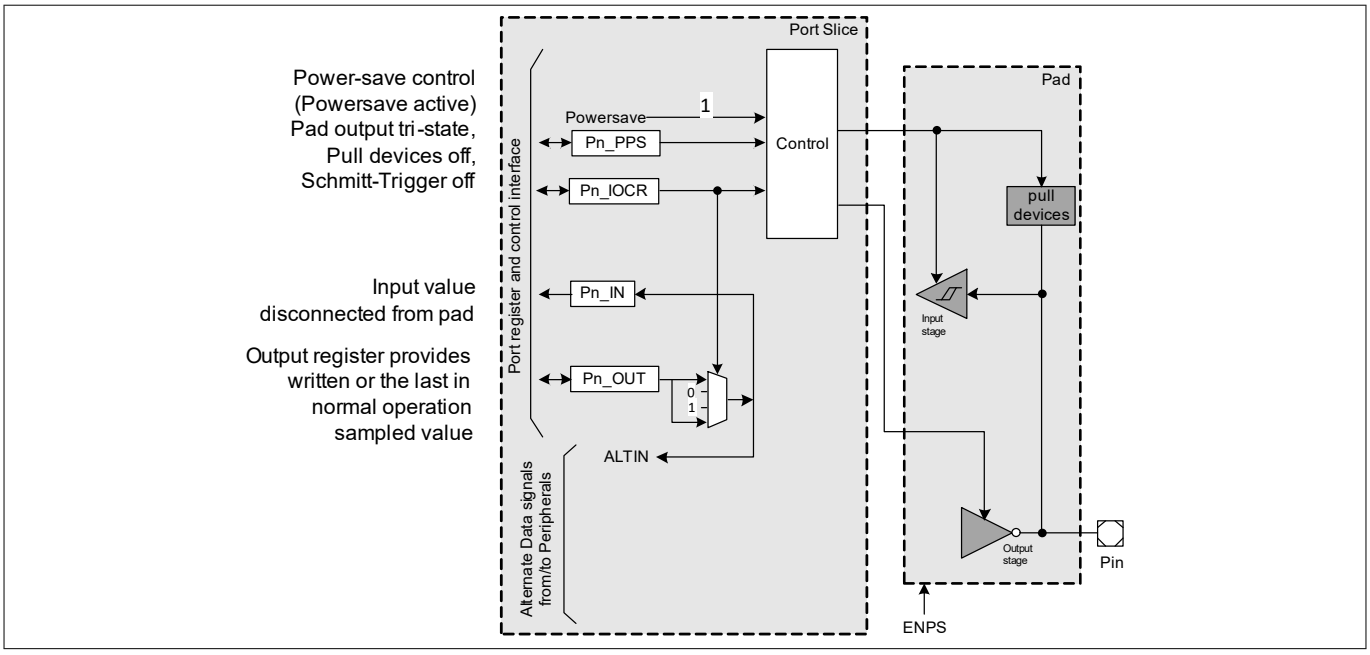


Figure 442 Port Pin in Power Save State

***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.*

25.5 Analog Ports

P2.2 - P2.9 is the analog and digital input port with a simplified port and pad structure, see Figure 443. The analog pads have no output drivers and the digital input Schmitt-Trigger can be controlled by the Pn_PDISC (Pin Function Decision Control Register) 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 Power Saving Mode Operation.

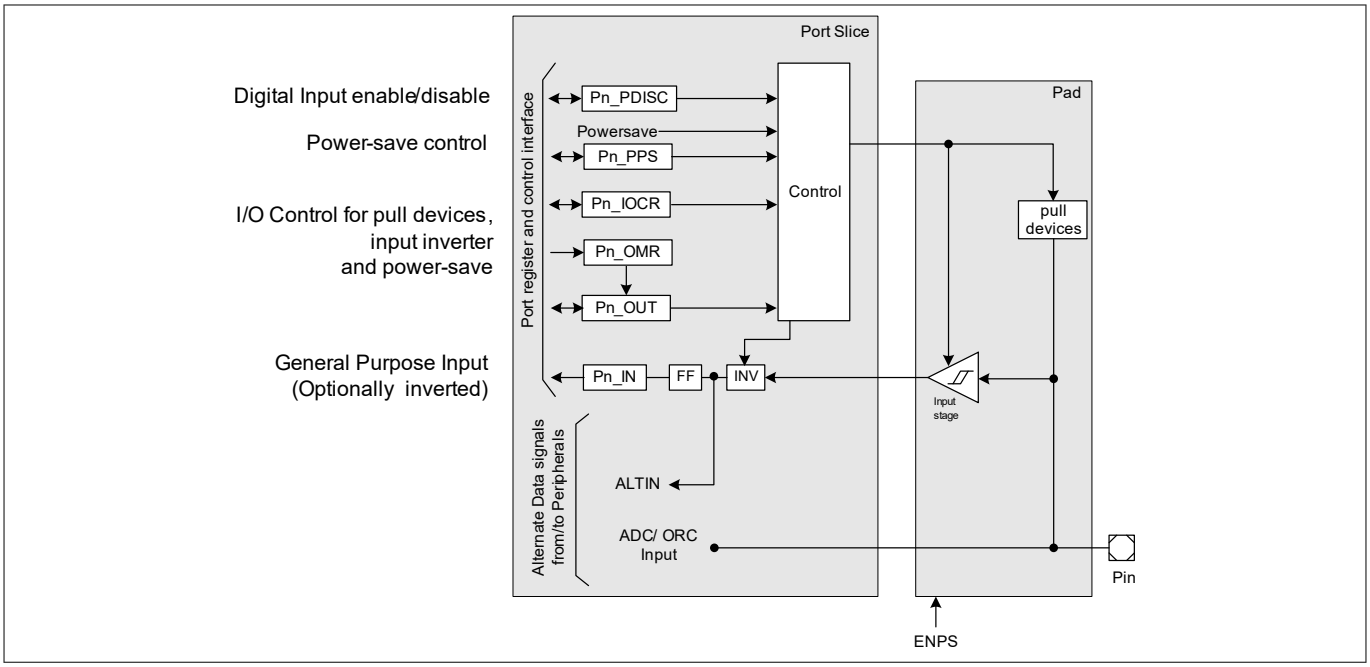


Figure 443 Analog Port Structure

25.6 Power, Reset and Clock

All digital I/O pads are held in a defined state, tristate, with output driver disabled and no pull devices active when one of the following occurs:

- During power-up, until V_{DDC} and V_{DDP} voltage levels are stable and within limits
- During power-fail, one or more voltage levels are outside the limits

Refer to the EVR section in the SCU chapter and Datasheet for details on the power-up, supply monitoring and voltage limits.

All Port registers are reset with the System Reset (see Reset Control Unit 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 or the analog input channels.

All registers of the Ports are clocked with f_{MCLK}.

25.7 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 (for example, 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
- Pn_PHCR
  pad hysteresis configuration (if applicable)
- 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)
- 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 omitted if no power save configuration is used on the port pin
- Pn_IOCR
  tri-state input and no pull device active

25.8 Registers

Registers Overview

The absolute register address is calculated by adding:
Module Base Address + Offset Address

Table 215 Registers Address Space

Module	Base Address	End Address	Note
P0	4004 0000$_H$	4004 00FF$_H$	4 standard pads can be disabled for external oscillator function
P1	4004 0100$_H$	4004 01FF$_H$	High current bi-directional pad
P2	4004 0200$_H$	4004 02FF$_H$	Analog/Digital input and bi-directional pad
P3	4004 0300$_H$	4004 03FF$_H$	
P4	4004 0400$_H$	4004 04FF$_H$	

Table 216 Register Overview

Register Name	Description	Offset Addr.	Access Mode	
			Read	Write
Pn_OUT	Port n Output Register	0000$_H$	U, PV	U, PV
Pn_OMR	Port n Output Modification Register	0004$_H$	U, PV	U, PV
–	Reserved	0008$_H$-000C$_H$	BE	BE
Pn_IOCR0	Port n Input/Output Control Register 0	0010$_H$	U, PV	PV
Pn_IOCR4	Port n Input/Output Control Register 4	0014$_H$	U, PV	PV
Pn_IOCR8	Port n Input/Output Control Register 8	0018$_H$	U, PV	PV
Pn_IOCR12	Port n Input/Output Control Register 12	001C$_H$	U, PV	PV
–	Reserved	0020$_H$	BE	BE
Pn_IN	Port n Input Register	0024$_H$	U, PV	R
–	Reserved	0028$_H$-003C$_H$	BE	BE
Pn_PHCR0	Port n Pad Hysteresis Control Register 0	0040$_H$	U, PV	PV
Pn_PHCR1	Port n Pad Hysteresis Control Register 1	0044$_H$	U, PV	PV
–	Reserved	0048$_H$-005C$_H$	BE	BE
P0_PDISC P1_PDISC	Port n Pin Function Decision Control Register (non-ADC/ACMP ports)	0060$_H$	U, PV	BE
P2_PDISC	Port n Pin Function Decision Control Register (ADC/ACMP ports)	0060$_H$	U, PV	PV
P3_PDISC P4_PDISC	Port n Pin Function Decision Control Register (non-ADC/ACMP ports)	0060$_H$	U, PV	BE
–	Reserved	0064$_H$-006C$_H$	BE	BE
Pn_PPS	Port n Pin Power Save Register	0070$_H$	U, PV	PV

(table continues...)

Table 216 (continued) Register Overview

Register Name	Description	Offset Addr.	Access Mode	
			Read	Write
Pn_HWSEL	Port n Hardware Select Register	0074_H	U, PV	PV
–	Reserved	0078_H-$00FC_H$	BE	BE

Table 217 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/ACMP ports)			
Pn_PH0			
Pn_PH1			
Pn_PPS			
Pn_PDISC (non- ADC/ACMP ports)		BE	

25.9 Package Pin Summary

The following general building block is used to describe each pin:

Table 218 Package Pin Mapping Description

Function	Package A	Package B	...	Pad Type
Px.y	N	N		Pad Class

The table is sorted by the “Function” column, starting with the regular Port pins (Px.y), followed by the 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:

- STD_INOUT (standard bi-directional pads)
- STD_INOUT/AN (standard bi-directional pads with analog input)
- STD_INOUT/clock (standard bi-directional pads with oscillator function)
- High Current (high current bi-directional pads)
- STD_IN/AN (standard input pads with analog input)
- Power (power supply)

Details about the pad properties are defined in the Datasheet.

Table 219 Package Pin Mapping

Function	LQFP 48	VQFN 48	VQFN 40	TSSOP 38	Pad Type	Notes
P0.0	30	29	23	17	STD_INOUT	
P0.1	31	30	24	18	STD_INOUT	
P0.2	32	31	25	19	STD_INOUT	
P0.3	33	32	26	20	STD_INOUT	
P0.4	34	33	27	21	STD_INOUT	
P0.5	35	34	28	22	STD_INOUT	
P0.6	36	35	29	23	STD_INOUT	
P0.7	-	36	30	24	STD_INOUT	
P0.8/ RTC_ XTAL1	39	39	33	27	STD_INOUT/ clock_IN	
P0.9/ RTC_ XTAL2	40	40	34	28	STD_INOUT/ clock_O	
P0.10/XTAL1	41	41	35	29	STD_INOUT/ clock_IN	
P0.11/XTAL2	42	42	36	30	STD_INOUT/ clock_O	
P0.12	43	43	37	31	STD_INOUT	
P0.13	44	44	38	32	STD_INOUT	
P0.14	45	45	39	33	STD_INOUT	
P0.15	46	46	40	34	STD_INOUT	
P1.0	27	26	22	16	High Current	
P1.1	26	25	21	15	High Current	
P1.2	25	24	20	14	High Current	
P1.3	24	23	19	13	High Current	
P1.4	23	22	18	12	High Current	
P1.5	22	21	17	11	High Current	
P1.6	-	20	16	–	High Current	

(table continues...)

Table 219 (continued) Package Pin Mapping

Function	LQFP 48	VQFN 48	VQFN 40	TSSOP 38	Pad Type	Notes
P2.0	6	3	1	35	STD_INOUT/ AN	
P2.1	7	4	2	36	STD_INOUT/ AN	
P2.2	8	5	3	37	STD_IN/AN	
P2.3	9	6	4	38	STD_IN/AN	
P2.4	10	7	5	1	STD_IN/AN	
P2.5	11	8	6	2	STD_IN/AN	
P2.6	12	9	7	3	STD_IN/AN	
P2.7	13	10	8	4	STD_IN/AN	
P2.8	14	11	9	5	STD_IN/AN	
P2.9	15	12	10	6	STD_IN/AN	
P2.10	16	13	11	7	STD_INOUT/ AN	
P2.11	17	14	12	8	STD_INOUT/ AN	
P2.12	18	15	–	–	STD_INOUT/ AN	
P2.13	19	16	–	–	STD_INOUT/ AN	
P3.0	29	28	–	–	STD_INOUT	
P4.1	47	–	–	–	STD_INOUT	
P4.2	48	–	–	–	STD_INOUT	
P4.3	1	–	–	–	STD_INOUT	
P4.4	-	47	–	–	STD_INOUT	
P4.5	-	48	–	–	STD_INOUT	
P4.6	2	1	–	–	STD_INOUT	
P4.7	3	2	–	–	STD_INOUT	
P4.8	4	–	–	–	STD_INOUT	
P4.9	5	–	–	–	STD_INOUT	
VSS	20	17	13	9	Power	Supply GND, ADC reference GND
VDD	21	18	14	10	Power	Supply VDD, ADC reference voltage/ORC reference voltage

(table continues...)

Table 219 (continued) Package Pin Mapping

Function	LQFP 48	VQFN 48	VQFN 40	TSSOP 38	Pad Type	Notes
VDDP	28	19	15	10	Power	When VDD is supplied, VDDP has to be supplied with the same voltage.
VDDP	-	27	–	–	Power	I/O port supply
VDDP	38	38	32	26	Power	I/O port supply
VSSP	37	37	31	25	Power	I/O port ground
VSSP	-	Exp. Pad	Exp. Pad	–	Power	Exposed Die Pad The exposed die pad is connected internally to VSSP. For proper operation, it is mandatory to connect the exposed pad to the board ground. For thermal aspects, please refer to the Package and Reliability chapter

25.10 Port I/O Functions

25.10.1 Port Pin for Boot Modes

Port functions can be overruled by the boot mode selected. The type of boot mode is selected via BMI. Table 220 shows the port pins used for the various boot modes.

Table 220 Port Pin for Boot Modes

Pin	Boot	Boot Description
P0.13	CS(O)	SSC BSL mode
P0.14	SWDIO_0	Debug mode (SWD)
	SPD_0	Debug mode (SPD)
	RX/TX	ASC BSL half-duplex mode
	RX	ASC BSL full-duplex mode
	RX	CAN BSL mode
	SCLK(O)	SSC BSL mode
P0.15	SWDCLK_0	Debug mode (SWD)
	TX	ASC BSL full-duplex mode
	TX	CAN BSL mode
	DATA(I/O)	SSC BSL mode
P1.2	SWDCLK_1	Debug mode (SWD)

(table continues...)

Table 220 (continued) Port Pin for Boot Modes

Pin	Boot	Boot Description
	TX	ASC BSL full-duplex mode
	TX	CAN BSL mode
P1.3	SWDIO_1	Debug mode (SWD)
	SPD_1	Debug mode (SPD)
	RX/TX	ASC BSL half-duplex mode
	RX	ASC BSL full-duplex mode
	RX	CAN BSL mode
P4.6	HWCON0	Boot Pins
P4.7	HWCON1	(Boot from pins mode must be selected)

25.10.2 Port I/O Function Description

The following general building block is used to describe the I/O functions of each PORT pin:

Table 221 Port I/O Function Description

Function	Outputs		Inputs	
	ALT1	ALTn	Input	Input
P0.0		MODA.OUT	MODC.INA	
Pn.y	MODA.OUT		MODA.INA	MODC.INB

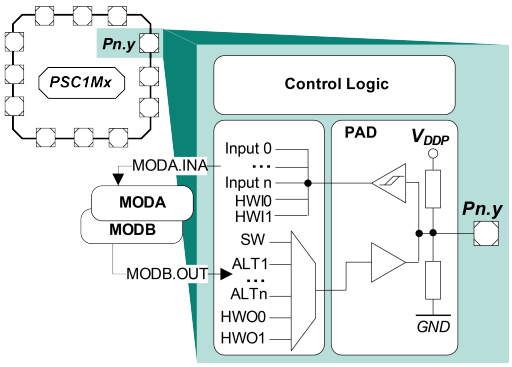


Figure 444 Simplified Port Structure

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 nine alternate output functions (ALT1 to ALT9) 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 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.

The port pin input can be connected to multiple peripherals. Most peripherals have an input multiplexer to select between different possible input sources.

Please refer to the Port I/O Functions table for the complete Port I/O function mapping.

25.10.3 Hardware Controlled I/O Function Description

The following general building block is used to describe the hardware I/O and pull control functions of each PORT pin:

Table 222 Hardware Controlled I/O Function Description

Function	Outputs	Inputs	Pull Control	
	HWO0	HWI0	HW0_PD	HW0_PU
P0.0	MODB.OUT	MODB.INA		
Pn.y			$\overline{\text{MODC.OUT}}$	MODC.OUT

By Pn_HWSEL (Port Pin Hardware Select Register) it is possible to select between different hardware “masters” (HWO0/HWI0, HWO1/HWI1). The selected peripheral can take control of the pin(s). Hardware control overrules settings in the respective port pin registers. Additional hardware signals HW0_PD/HW1_PD and HW0_PU/HW1_PU controlled by the peripherals can be used to control the pull devices of the pin.

Please refer to Table 224 for the complete hardware I/O and pull control function mapping.

Table 223 Port I/O Functions

Function	Outputs									Inputs											
	ALT1	ALT2	ALT3	ALT4	ALT5	ALT6	ALT7	ALT8	ALT9	Input	Input	Input	Input	Input	Input	Input	Input	Input	Input	Input	Input
P0.0	ERU0.PDOUT0		ERU0.GOUT0	CCU40.OUT0	CCU80.OUT00	USIC0_CH0.SELO0	USIC0_CH1.SELO0	CCU81.OUT00	USIC1_CH1.DOUT0	BCCU0.TRAPINB	CCU40.IN0AC					USIC1_CH1.DX0A	USIC0_CH0.DX2A		USIC0_CH1.DX2A		
P0.1	ERU0.PDOUT1		ERU0.GOUT1	CCU40.OUT1	CCU80.OUT01	BCCU0.OUT8	SCU.VDROP	USIC1_CH1.SCLKOUT	USIC1_CH1.DOUT0		CCU40.IN1AC					USIC1_CH1.DX0B	USIC1_CH1.DX1A				
P0.2	ERU0.PDOUT2		ERU0.GOUT2	CCU40.OUT2	CCU80.OUT2	VADC0.EMUX02	CCU80.OUT10	USIC1_CH0.SCLKOUT	USIC1_CH0.DOUT0		CCU40.IN2AC					USIC1_CH0.DX0A	USIC1_CH0.DX1A				
P0.3	ERU0.PDOUT3		ERU0.GOUT3	CCU40.OUT3	CCU80.OUT3	VADC0.EMUX01	CCU80.OUT11	USIC1_CH1.SCLKOUT	USIC1_CH0.DOUT0		CCU40.IN3AC					USIC1_CH0.DX0B					
P0.4	BCCU0.OUT0			CCU40.OUT1	CCU80.OUT13	VADC0.EMUX00	WWDT.SERVICE_OUT	USIC1_CH1.SELO0	CAN.N0_TXD			CCU41.IN0AB	CCU80.IN0AB							CAN.N0_RXDA	

(table continues...)

Table 223 (continued) Port I/O Functions

Function	Outputs									Inputs											
	ALT1	ALT2	ALT3	ALT4	ALT5	ALT6	ALT7	ALT8	ALT9	Input	Input	Input	Input	Input	Input	Input	Input	Input	Input	Input	Input
P0.5	BCCU0.OUT1			CCU40.OUT0	CCU80.OUT12	ACMP2.OUT	CCU80.OUT01	VADC0.EMUX10	CAN.N0_TXD			CCU41.IN1AB	CCU80.IN1AB								CAN.N0_RXDB	
P0.6	BCCU0.OUT2			CCU40.OUT0	CCU80.OUT11	USIC0_CH1.MCLKOUT	USIC0_CH1.DOUT0	VADC0.EMUX11	CCU41.OUT0		CCU40.IN0AB	CCU41.IN2AB						USIC0_CH1.DX0C			
P0.7	BCCU0.OUT3			CCU40.OUT1	CCU80.OUT10	USIC0_CH0.SCLKOUT	USIC0_CH1.DOUT0	VADC0.EMUX12	CCU41.OUT1		CCU40.IN1AB	CCU41.IN3AB					USIC0_CH0.DX1C	USIC0_CH1.DX0D	USIC0_CH1.DX1C		
P0.8/RTC_XTAL1	BCCU0.OUT4			CCU40.OUT2	CCU80.OUT20	USIC0_CH0.SCLKOUT	USIC0_CH1.SCLKOUT	CCU81.OUT20	CCU41.OUT2		CCU40.IN2AB						USIC0_CH0.DX1B		USIC0_CH1.DX1B		
P0.9/RTC_XTAL2	BCCU0.OUT5			CCU40.OUT3	CCU80.OUT21	USIC0_CH0.SELO0	USIC0_CH1.SELO0	CCU81.OUT21	CCU41.OUT3		CCU40.IN3AB						USIC0_CH0.DX2B		USIC0_CH1.DX2B		

(table continues...)

Table 223 (continued) Port I/O Functions

Function	Outputs									Inputs											
	ALT1	ALT2	ALT3	ALT4	ALT5	ALT6	ALT7	ALT8	ALT9	Input	Input	Input	Input	Input	Input	Input	Input	Input	Input	Input	Input
P0.10 / XTAL1	BCCU0.OUT6			ACMP0.OUT	CCU80.OUT22	USIC0_CH0.SELO1	USIC0_CH1.SELO1	CCU81.OUT22					CCU80.IN2AB	CCU81.IN2AB			USIC0_CH0.DX2C		USIC0_CH1.DX2C		
P0.11 / XTAL2	BCCU0.OUT7			USIC0_CH0.MCLKOUT	CCU80.OUT23	USIC0_CH0.SELO2	USIC0_CH1.SELO2	CCU81.OUT23									USIC0_CH0.DX2D		USIC0_CH1.DX2D		
P0.12	BCCU0.OUT6				CCU80.OUT33	USIC0_CH0.SELO3	CCU80.OUT20		CAN.N1_TXD	BCCU0.TRAPINA	CCU40.IN0AA	CCU40.IN1AAA	CCU40.IN2AA	CCU81.IN0AU	CCU40.IN3AA	CCU80.IN0AA	USIC0_CH0.DX2E	CCU80.IN1AA	CCU80.IN2AA	CAN.N1_RXDA	CCU80.IN3AA
P0.13	WWDT.SERVICE_OUT				CCU80.OUT32	USIC0_CH0.SELO4	CCU80.OUT21		CAN.N1_TXD				CCU80.IN3AB	CCU81.IN1AU	POSIF0.IN0B		USIC0_CH0.DX2F			CAN.N1_RXDB	
P0.14	BCCU0.OUT7				CCU80.OUT31	USIC0_CH0.DOUT0	USIC0_CH0.SCLKOUT		CAN.N0_TXD					CCU81.IN2AU	POSIF0.IN1B	USIC0_CH0.DX0A	USIC0_CH0.DX1A	USIC1_CH1.DX5B		CAN.N0_RXDC	

(table continues...)

Table 223 (continued) Port I/O Functions

Function	Outputs									Inputs											
	ALT1	ALT2	ALT3	ALT4	ALT5	ALT6	ALT7	ALT8	ALT9	Input	Input	Input	Input	Input	Input	Input	Input	Input	Input	Input	Input
P0.15	BCCU0.OUT8				CCU80.OUT30	USIC0_CH0.DOUT0	USIC0_CH1.MCLKOUT		CAN.N0_TXD					CCU81.IN3AU	POSIF0.IN2B	USIC0_CH0.DX0B		USIC1_CH1.DX3B	USIC1_CH1.DX4B	CAN.N0_RXDD	
P1.0	BCCU0.OUT0	CCU40.OUT0			CCU80.OUT00	ACMP1.OUT	USIC0_CH0.DOUT0	CCU81.OUT00	CAN.N0_TXD						POSIF0.IN2A	USIC0_CH0.DX0C				CAN.N0_RXDG	
P1.1	ERU1.PDOUT1	CCU40.OUT1			CCU80.OUT01	USIC0_CH0.DOUT0	USIC0_CH1.SELO0	CCU81.OUT01	CAN.N0_TXD						POSIF0.IN1A	USIC0_CH0.DX0D	USIC0_CH0.DX1D		USIC0_CH1.DX2E	CAN.N0_RXDH	
P1.2	ERU1.PDOUT2	CCU40.OUT2			CCU80.OUT10	ACMP2.OUT	USIC0_CH1.DOUT0	CCU81.OUT10	CAN.N1_TXD						POSIF0.IN0A			USIC0_CH1.DX0B		CAN.N1_RXDG	
P1.3	ERU1.PDOUT3	CCU40.OUT3			CCU80.OUT11	USIC0_CH1.SCLKOUT	USIC0_CH1.DOUT0	CCU81.OUT11	CAN.N1_TXD									USIC0_CH1.DX0A	USIC0_CH1.DX1A	CAN.N1_RXDH	

(table continues...)

Table 223 (continued) Port I/O Functions

Function	Outputs									Inputs											
	ALT1	ALT2	ALT3	ALT4	ALT5	ALT6	ALT7	ALT8	ALT9	Input	Input	Input	Input	Input	Input	Input	Input	Input	Input	Input	Input
P1.4	ERU1.PDOUT0	USIC0_CH1.SCLKOUT			CCU80.OUT20	USIC0_CH0.SELO0	USIC0_CH1.SELO1	CCU81.OUT20	CCU41.OUT0							USIC0_CH0.DX5E		USIC0_CH1.DX5E			
P1.5	ERU1.PDOUT1	USIC0_CH0.DOUT0		BCCU0.OUT1	CCU80.OUT21	USIC0_CH0.SELO1	USIC0_CH1.SELO2	CCU81.OUT21	CCU41.OUT1									USIC0_CH1.DX5F			
P1.6	ERU1.PDOUT2	USIC0_CH1.DOUT0		USIC0_CH0.SCLKOUT	BCCU0.OUT2	USIC0_CH0.SELO2	USIC0_CH1.SELO3	CCU81.OUT30	CCU41.OUT2						POSIF1.IN2A	USIC0_CH0.DX5F					
P2.0	ERU0.PDOUT3	CCU40.OUT0	ERU0.GOUT3		CCU80.OUT20	USIC0_CH0.DOUT0	USIC0_CH0.SCLKOUT	CCU81.OUT20	CAN.N0_TXD		VADC0.G0CH5					USIC0_CH0.DX0E	USIC0_CH0.DX1E		USIC0_CH1.DX2F	CAN.N0_RXDE	ERU0.0B0
P2.1	ERU0.PDOUT2	CCU40.OUT1	ERU0.GOUT2		CCU80.OUT21	USIC0_CH0.DOUT0	USIC0_CH1.SCLKOUT	CCU81.OUT21	CAN.N0_TXD	ACMP2.INP	VADC0.G0CH6					USIC0_CH0.DX0F		USIC0_CH1.DX3A	USIC0_CH1.DX4A	CAN.N0_RXDF	ERU0.1B0

(table continues...)

Table 223 (continued) Port I/O Functions

Function	Outputs									Inputs											
	ALT1	ALT2	ALT3	ALT4	ALT5	ALT6	ALT7	ALT8	ALT9	Input	Input	Input	Input	Input	Input	Input	Input	Input	Input	Input	Input
P2.2										ACMP2.INP	VADC0.G0CH7		ORC0.AIN	USIC1_CH0.DX5E		USIC0_CH0.DX3A	USIC0_CH0.DX4A	USIC0_CH1.DX5A			ERU0.0B1
P2.3												VADC0.G1CH5	ORC1.AIN	USIC1_CH0.DX3E	USIC1_CH0.DX4E	USIC1_CH1.DX5C	USIC0_CH0.DX5B	USIC0_CH1.DX3C	USIC0_CH1.DX4C		ERU0.1B1
P2.4												VADC0.G1CH6	ORC2.AIN	USIC1_CH1.DX3C	USIC1_CH1.DX4C	USIC0_CH0.DX3B	USIC0_CH0.DX4B	USIC1_CH0.DX5F	USIC0_CH1.DX5B		ERU0.0A1
P2.5												VADC0.G1CH7	ORC3.AIN	USIC1_CH1.DX5D		USIC0_CH0.DX5D		USIC0_CH1.DX3E	USIC0_CH1.DX4E		ERU0.1A1
P2.6										ACMP1.INN	VADC0.G0CH0		ORC4.AIN	USIC1_CH1.DX3E	USIC1_CH1.DX4E	USIC0_CH0.DX3E	USIC0_CH0.DX4E	USIC0_CH1.DX5D			ERU0.2A1
P2.7										ACMP1.INP		VADC0.G1CH1	ORC5.AIN	USIC1_CH1.DX5E		USIC0_CH0.DX5C		USIC0_CH1.DX3D	USIC0_CH1.DX4D		ERU0.3A1
P2.8										ACMP0.INN	VADC0.G0CH1	VADC0.G1CH0	ORC6.AIN			USIC0_CH0.DX3D	USIC0_CH0.DX4D	USIC0_CH1.DX5C			ERU0.3B1

(table continues...)

Table 223 (continued) Port I/O Functions

Function	Outputs									Inputs										
	ALT1	ALT2	ALT3	ALT4	ALT5	ALT6	ALT7	ALT8	ALT9	Input	Input	Input	Input	Input	Input	Input	Input	Input	Input	Input
P2.9										ACMP0.INP	VADC0.G0CH2	VADC0.G1CH4	ORC7.AIN			USIC0_CH0.DX5A		USIC0_CH1.DX3B	USIC0_CH1.DX4B		ERU0.3B0
P2.10	ERU0.PDOUT1	CCU40.OUT2	ERU0.GOUT1		CCU80.OUT30	ACMP0.OUT	USIC0_CH1.DOUT0		CAN.N1_TXD		VADC0.G0CH3	VADC0.G1CH2				USIC0_CH0.DX3C	USIC0_CH0.DX4C	USIC0_CH1.DX0F		CAN.N1_RXDE	ERU0.2B0
P2.11	ERU0.PDOUT0	CCU40.OUT3	ERU0.GOUT0		CCU80.OUT31	USIC0_CH1.SCLKOUT	USIC0_CH1.DOUT0		CAN.N1_TXD	ACMP.REF	VADC0.G0CH4	VADC0.G1CH3						USIC0_CH1.DX0E	USIC0_CH1.DX1E	CAN.N1_RXDF	ERU0.2B1
P2.12	BCCU0.OUT3	VADC0.EMUX00	USIC1_CH0.SCLKOUT	USIC1_CH1.SCLKOUT		ACMP2.OUT	USIC1_CH1.DOUT0			ACMP3.INN						USIC1_CH0.DX3A	USIC1_CH0.DX4A	USIC1_CH1.DX0C	USIC1_CH1.DX1B		ERU1.3A2
P2.13	BCCU0.OUT4	CCU40.OUT3	USIC1_CH0.MCLKOUT	CCU81.OUT31		VADC0.EMUX01	USIC1_CH1.DOUT0	CCU81.OUT33	CCU41.OUT3	ACMP3.INP						USIC1_CH0.DX5A		USIC1_CH1.DX0D			ERU1.3A3

(table continues...)

Table 223 (continued) Port I/O Functions

Function	Outputs									Inputs										
	ALT1	ALT2	ALT3	ALT4	ALT5	ALT6	ALT7	ALT8	ALT9	Input	Input	Input	Input	Input	Input	Input	Input	Input	Input	Input
P3.0	BCCU0.OUT0	USIC1_CH1.DOUT0	USIC1_CH1.SCLKOUT		CCU80.OUT21	ACMP1.OUT	USIC1_CH0.SELO1	CCU81.OUT21	CCU41.OUT0	BCCU0.TRAPINC	CCU41.IN0AA	CCU41.IN1AA	CCU41.IN2AA	CCU41.IN3AA	CCU81.IN0AA	CCU81.IN1AA	CCU81.IN2AA	USIC1_CH1.DX0E	USIC1_CH1.DX1D	CCU81.IN3AA	ERU1.0A1
P4.1	BCCU0.OUT8	ERU1.PDOUT1		ERU1.GOUT1	CCU40.OUT1	ACMP3.OUT	USIC1_CH1.SELO2	CCU81.OUT11	CCU41.OUT1		CCU40.IN1BA	CCU41.IN1AC	CCU80.IN1AU		POSIF1.IN0B	USIC1_CH0.DX5C					
P4.2	BCCU0.OUT4	ERU1.PDOUT2	CCU81.OUT20	ERU1.GOUT2	CCU40.OUT2	ACMP2.OUT	USIC1_CH1.SELO3	CCU81.OUT12	CCU41.OUT2		CCU40.IN2BA	CCU41.IN2AC	CCU80.IN2AU	CCU81.IN1AB	POSIF1.IN1B	USIC1_CH0.DX5D					
P4.3	BCCU0.OUT5	ERU1.PDOUT3	CCU81.OUT21	ERU1.GOUT3	CCU40.OUT3	ACMP0.OUT	USIC1_CH0.SCLKOUT	CCU81.OUT13	CCU41.OUT3		CCU40.IN3BA	CCU41.IN3AC	CCU80.IN3AU		POSIF1.IN2B		USIC1_CH0.DX1B				
P4.4	BCCU0.OUT0				CCU80.OUT00	USIC1_CH0.DOUT0		CCU81.OUT00	CCU41.OUT0			CCU41.IN0AV				USIC1_CH0.DX0C		USIC1_CH1.DX5F			ERU1.0A2

(table continues...)

Table 223 (continued) Port I/O Functions

Function	Outputs									Inputs											
	ALT1	ALT2	ALT3	ALT4	ALT5	ALT6	ALT7	ALT8	ALT9	Input	Input	Input	Input	Input	Input	Input	Input	Input	Input	Input	Input
P4.5	BCCU0.OUT8				CCU80.OUT01	USIC1_CH0.DOUT0	USIC1_CH0.SCLKOUT	CCU81.OUT01	CCU41.OUT1			CCU41.IN1AV				USIC1_CH0.DX0D	USIC1_CH0.DX1C				ERU1.1A2
P4.6	BCCU0.OUT2		CCU81.OUT10		CCU80.OUT10		USIC1_CH0.SCLKOUT	CCU81.OUT02	CCU41.OUT2			CCU41.IN2AV		CCU81.IN0AB			USIC1_CH0.DX1D				ERU1.2A2
P4.7	BCCU0.OUT5		CCU81.OUT11		CCU80.OUT11		USIC1_CH0.SELO0	CCU81.OUT03	CCU41.OUT3			CCU41.IN3AV					USIC1_CH0.DX2A				ERU1.0A3
P4.8	BCCU0.OUT7				CCU80.OUT30	CCU40.OUT0	USIC1_CH0.SELO1	CCU81.OUT30	CAN.N1_TXD		CCU40.IN0AV	CCU41.IN0BA					USIC1_CH0.DX2B			CAN.N1_RXDC	
P4.9	BCCU0.OUT3				CCU80.OUT31	CCU40.OUT1	USIC1_CH0.SELO2	CCU81.OUT31	CAN.N1_TXD		CCU40.IN1AV	CCU41.IN1BA					USIC1_CH0.DX2C			CAN.N1_RXDD	

Table 224 Hardware I/O Controlled Functions

Function	Outputs	Outputs	Inputs	Inputs	Pull Control	Pull Control	Pull Control	Pull Control
	HWO0	HWO1	HWI0	HWI1	HW0_PD	HW0_PU	HW1_PD	HW1_PU
P1.0		USIC0_CH0.DOUT0		USIC0_CH0.HWIN0	$\overline{\text{BCCU0.OUT2}}$	BCCU0.OUT2		
P1.1		USIC0_CH0.DOUT1		USIC0_CH0.HWIN1	$\overline{\text{BCCU0.OUT3}}$	BCCU0.OUT3		
P1.2		USIC0_CH0.DOUT2		USIC0_CH0.HWIN2	$\overline{\text{BCCU0.OUT4}}$	BCCU0.OUT4		
P1.3		USIC0_CH0.DOUT3		USIC0_CH0.HWIN3	$\overline{\text{BCCU0.OUT5}}$	BCCU0.OUT5		
P1.4					$\overline{\text{BCCU0.OUT6}}$	BCCU0.OUT6		
P1.5					$\overline{\text{BCCU0.OUT7}}$	BCCU0.OUT7		
P1.6					$\overline{\text{BCCU0.OUT8}}$	BCCU0.OUT8		
P2.0					$\overline{\text{BCCU0.OUT1}}$	BCCU0.OUT1		
P2.1					$\overline{\text{BCCU0.OUT6}}$	BCCU0.OUT6		
P2.2					$\overline{\text{BCCU0.OUT0}}$	BCCU0.OUT0	$\overline{\text{CCU40.OUT3}}$	CCU40.OUT3
P2.3					$\overline{\text{ACMP2.OUT}}$	ACMP2.OUT		
P2.4					$\overline{\text{BCCU0.OUT8}}$	BCCU0.OUT8		
P2.5					$\overline{\text{ACMP1.OUT}}$	ACMP1.OUT		
P2.6					$\overline{\text{BCCU0.OUT2}}$	BCCU0.OUT2	$\overline{\text{CCU40.OUT3}}$	CCU40.OUT3
P2.7					$\overline{\text{BCCU0.OUT8}}$	BCCU0.OUT8	$\overline{\text{CCU40.OUT3}}$	CCU40.OUT3
P2.8					$\overline{\text{BCCU0.OUT1}}$	BCCU0.OUT1	$\overline{\text{CCU40.OUT2}}$	CCU40.OUT2
P2.9					$\overline{\text{BCCU0.OUT7}}$	BCCU0.OUT7	$\overline{\text{CCU40.OUT2}}$	CCU40.OUT2
P2.10					$\overline{\text{BCCU0.OUT4}}$	BCCU0.OUT4		
P2.11					$\overline{\text{BCCU0.OUT5}}$	BCCU0.OUT5		

(table continues...)

Table 224 (continued) Hardware I/O Controlled Functions

Function	Outputs	Outputs	Inputs	Inputs	Pull Control	Pull Control	Pull Control	Pull Control
	HWO0	HWO1	HWI0	HWI1	HW0_PD	HW0_PU	HW1_PD	HW1_PU
P2.12					$\overline{\text{BCCU0.OUT3}}$	BCCU0.OUT3	$\overline{\text{CCU41.OUT0}}$	CCU41.OUT0
P2.13					$\overline{\text{BCCU0.OUT4}}$	BCCU0.OUT4	$\overline{\text{CCU41.OUT2}}$	CCU41.OUT2
P3.0								
P4.1								
P4.2								
P4.3								

25.11 Pad Characteristics

This section provides a general overview of the pad characteristics.

25.11.1 Input and Output Voltage

Figure 445 explains the input voltage ranges of V_{IN} and V_{AIN} and its dependency to the supply level of V_{DDP}. The input voltage must not exceed 6.0 V, and it must not be more than 0.5 V above V_{DDP}. For the range up to V_{DDP} + 0.5 V also see the definition of the overload conditions in Pin Reliability in Overload.

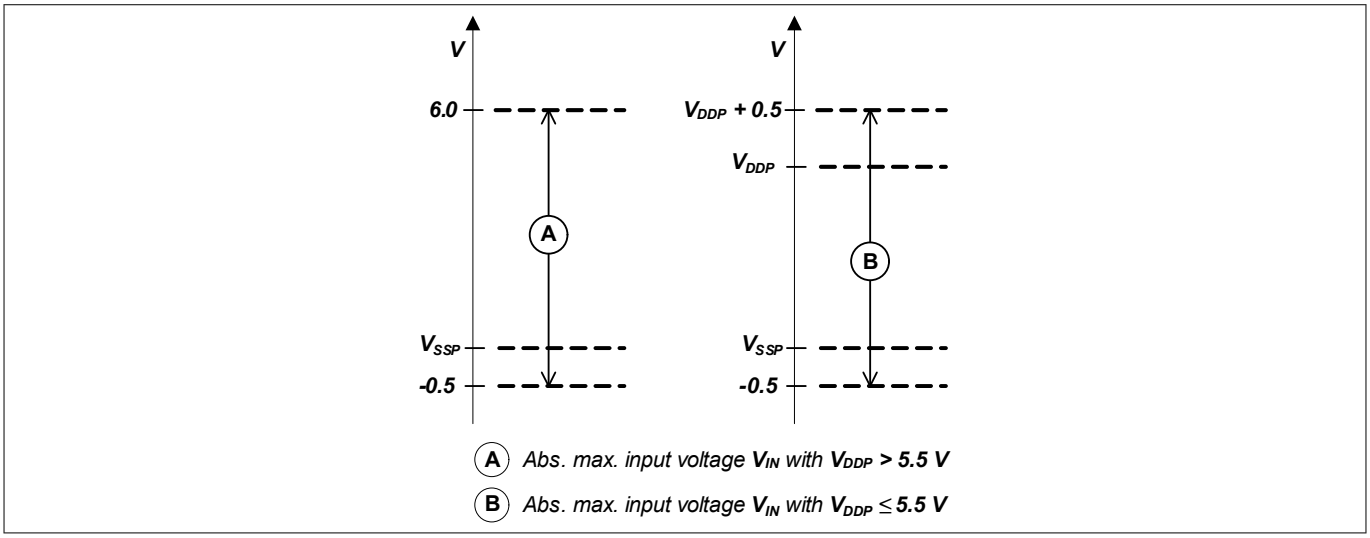


Figure 445 Absolute Maximum Input Voltage Ranges

25.11.2 Input Low and Input High Voltage

Figure 446 explains the Input Low Voltage, V_{ILPS}/V_{ILPL} and Input High Voltage, V_{IHPS}/V_{ILPL} and their ranges in Standard Hysteresis mode (S) and Large Hysteresis mode (L).

V_{ILPx} defines the maximum voltage level that will be interpreted as a '0' by a digital input.V_{IHPx} defines the minimum voltage level that will be interpreted as a '1' by a digital input.

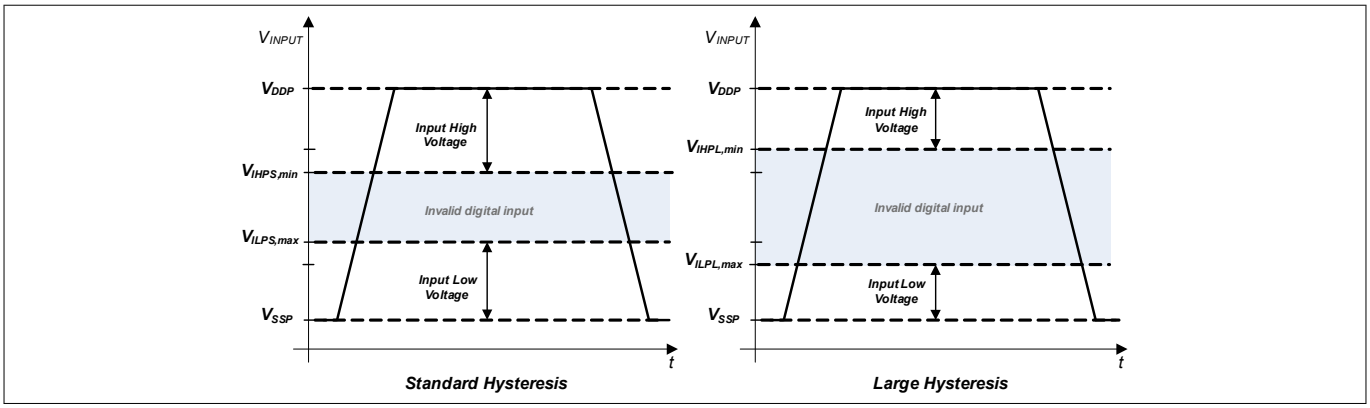


Figure 446 Input Low Voltage and Input High Voltage

A voltage level between V_{ILPx} and V_{IHPx} will result in an undefined logic state. It could be interpreted as either '0' or '1' by the circuit that received these inputs. In addition, most CMOS inputs will draw excessive current if an undefined voltage level is applied.

25.11.3 Output Low and Output High Voltage

V_{OLP} defines the maximum voltage level that will appear on a digital output set to '0'. V_{OHP} defines the minimum voltage level that will appear on a digital output set to '1'.

The output parameters are usually given for a specified I_{OL}, as the output levels are dependant on the load impedance applied to the logic output.

Figure 447 and Figure 448 explain the Output Low Voltage, V_{OLP} and Output High Voltage, V_{OHP} and its dependency to the 2 different output load current, I_{OL_Lx}/ I_{OH_Lx}.

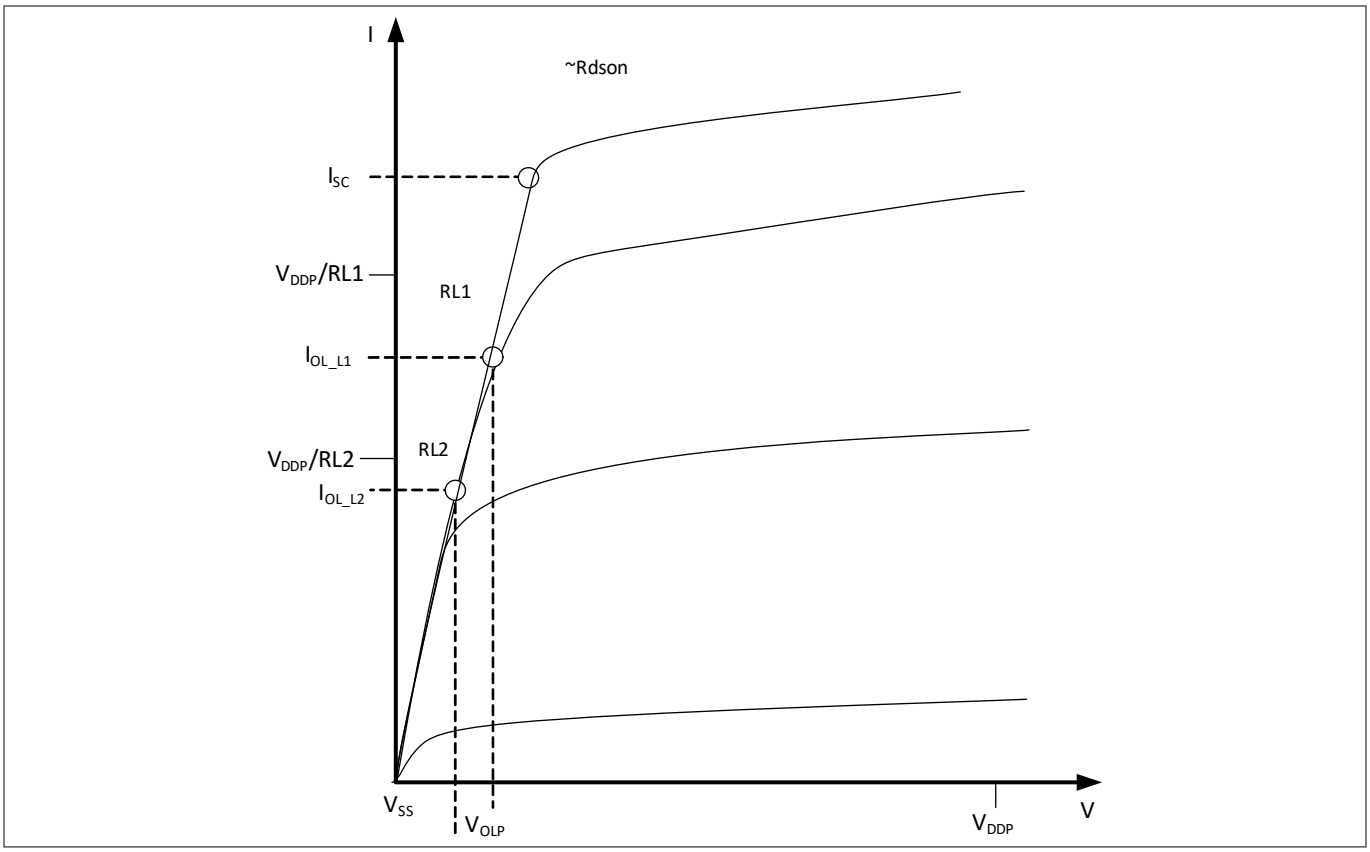


Figure 447 Output Low Voltage

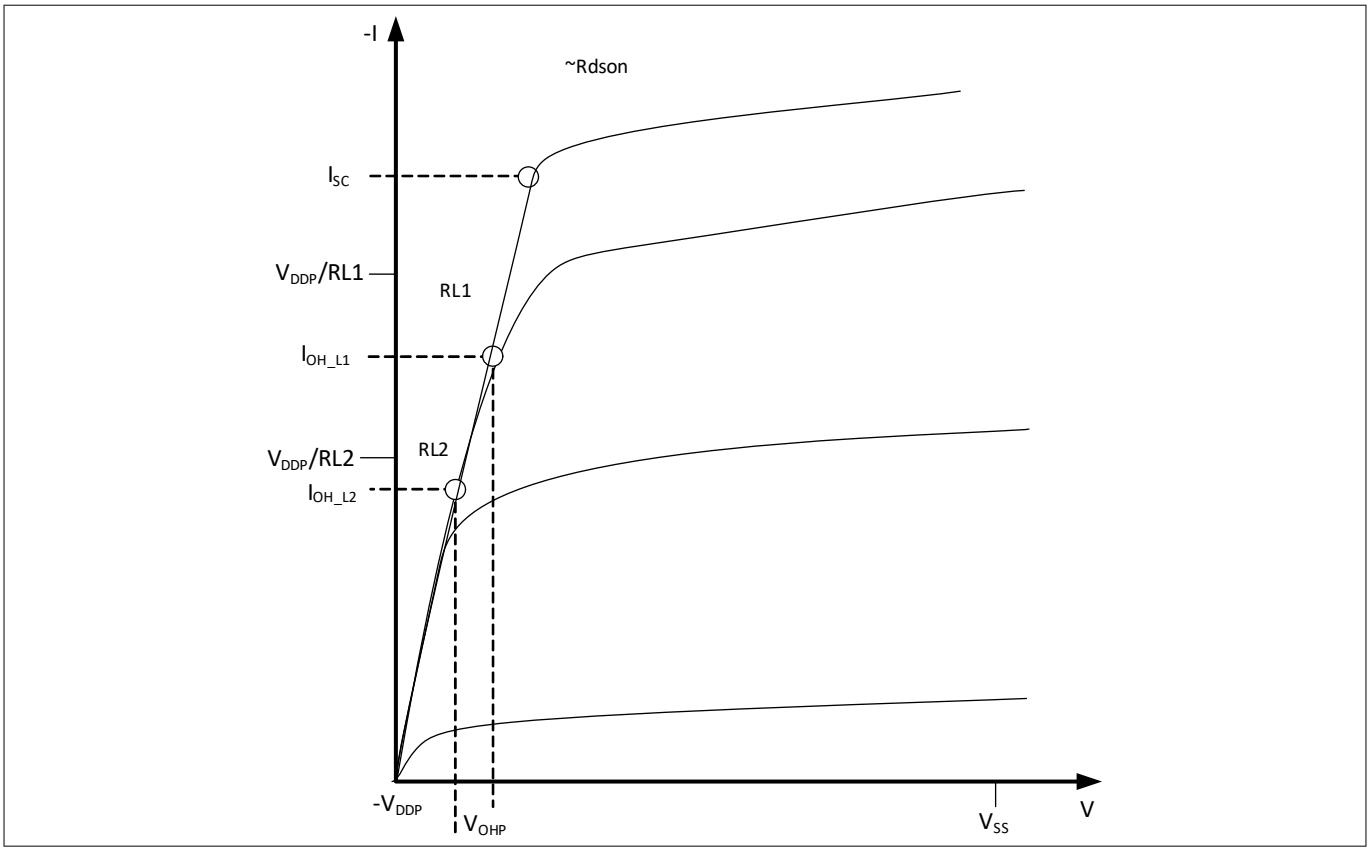


Figure 448 Output High Voltage

25.11.4 Pin Reliability in Overload

When receiving signals from higher voltage devices, low-voltage devices experience overload currents and voltages that go beyond their own IO power supplies specification.

If a pin current is outside of the operating conditions but within the overload conditions, then the parameters of this pin as stated in the operating conditions can no longer be guaranteed. Operation is still possible in most cases but with relaxed parameters.

***Note**: An overload condition on one or more pins does not require a reset.*

***Note**: A series resistor at the pin to limit the current to the maximum permitted overload current is sufficient to handle failure situations like short to battery.*

Figure 449 shows the path of the input currents during overload via the ESD protection structures. The diodes against V_{DDP} and ground are a simplified representation of these ESD protection structures.

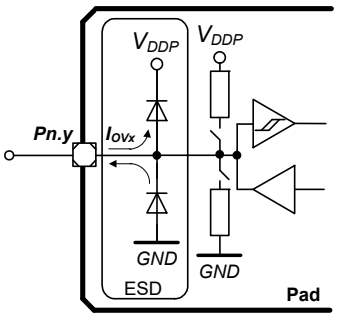


Figure 449 Input Overload Current via ESD structures

Refer to the Datasheet for the overload conditions and limits that will not cause any negative reliability impact .

25.11.5 Internal Pull Device

Figure 450 visualizes the input characteristics with an active internal pull device:

- in the cases “A” the internal pull device is overridden by a strong external driver;
- in the cases “B” the internal pull device defines the input logical state against a weak external load.

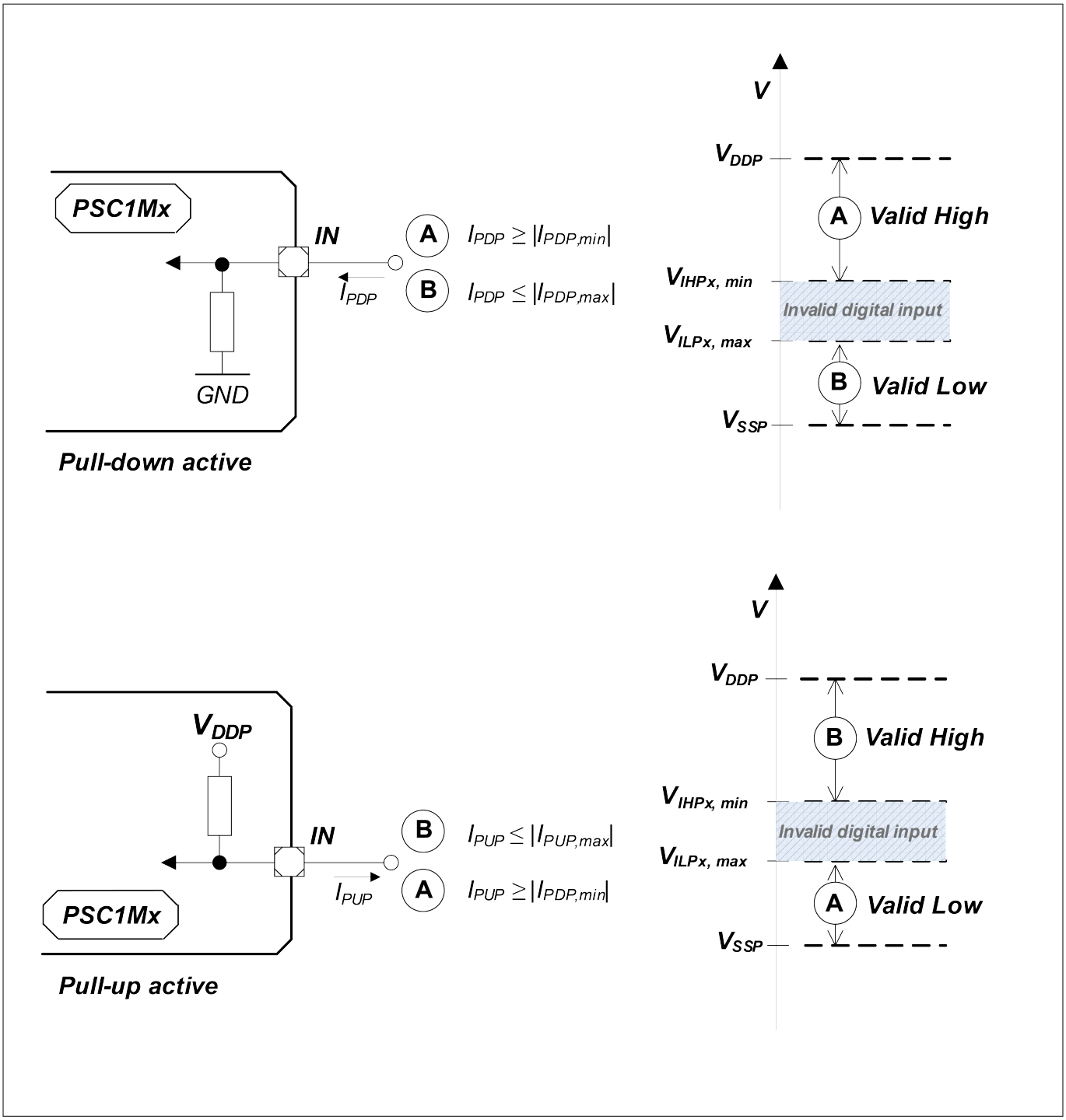


Figure 450 Pull Device Input Characteristics

26 Boot and Startup

The startup sequence of the PSC1Mx is a process taking place before user application software takes control of the system and is comprising of two major phases (see Figure 451), split in several distinctive steps:

Hardware Controlled Startup Phase

The hardware controlled startup 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 Datasheet 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 startup sequence.

For details of the setup requirements, please refer to Datasheet.

Software Controlled Startup Phase

The software controlled startup phase is the part 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.

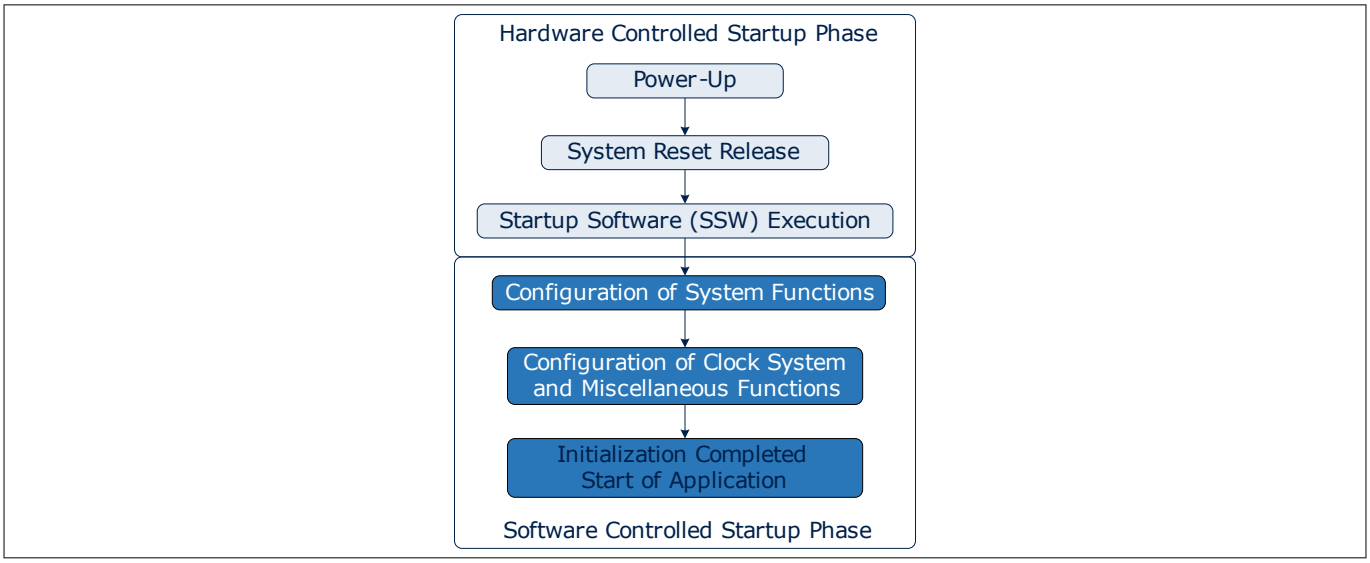


Figure 451 Startup sequence

26.1 Startup Sequence and System Dependencies

A more detailed description of the startup sequence is available in the following sub-chapters.

26.1.1 Power-Up

Power up of the microcontroller gets performed by applying V_{DDP} supply (for details of the supply requirements, please refer to Datasheet). Once V_{DDP} reaches a threshold, the internal core voltage, V_{DDC}, is generated automatically by the EVR.

26.1.2 System Reset Release

The internal power-on reset generation is based on the supply and core voltage validation. When V_{DDP} and V_{DDC} reaches a stable threshold level, the power-on reset is released. Next, after the on-chip oscillators generate a stable clock output, the system reset is released automatically and the SSW code starts to run.

The system reset can be triggered from various sources like software controlled CPU reset register or a watchdog time-out-triggered reset. For more details on reset control details, please refer to the Reset Control Unit (RCU) section in the SCU chapter.

The cause of the last reset gets automatically stored in the SCU_RSTSTAT register and can be checked by user software to determine the state of the system and for debug purpose. The reset status in the SCU_RSTSTAT register shall be reset with SCU_RSTCLR register after each startup in order to ensure consistent source indication after the next reset.

After reset release, MCLK and PCLK are running at 8MHz and most of the peripherals' clocks are disabled except for CPU, memories and PORT. It is recommended to disable the clock of the unused modules in order to reduce power consumption.

26.1.3 Startup Software (SSW) Execution

After the reset is deasserted, CPU starts to execute the SSW code from the ROM memory. To indicate the start of the SSW execution, the pullup device in P0.14 is enabled. Pullup device is disabled during reset.

SSW reads the Boot Mode Index (BMI) stored in Flash and decide the startup mode selected by the user. For more details about the various startup modes and handling of BMI, please refer to Start-up Modes.

To handle the initial hardfault error during the SSW execution, SSW installs "jump to itself" instruction at SRAM location $2000'000C_H$ as temporary HardFault handler - until the user code installs its own. It is installed upon master reset only.

During SSW execution, the SRAM area between $2000'00C0_H$ and $2000'0200_H$ is reserved for usage by PSC1Mx SSW, therefore the user software should not store in this area data which must be preserved throughout (non-power) resets.

The MCLK and PCLK frequency that is used to execute the SSW and to start the user code can be configured by user. In addition, some of the peripheral clock can also be enabled (default disabled) by the SSW. Detailed description can be found in the Clock system handling by SSW.

26.1.3.1 Clock system handling by SSW

PSC1Mx SSW is able to change device clock settings to allow flexibility of device performance, for example, speed vs. power consumption optimization during start-up.

The clock handling by SSW is user configurable by installing values in Flash. For this purpose two word locations - CLK_VAL1 and CLK_VAL2 - are assigned in Flash (refer to Table 227), the values from these locations are processed by SSW as follows:

- if CLK_VAL1[31]=0 - CLK_VAL1[21:10], [7:0] bits are installed into SCU_CLKCR bit fields [19:8], [7:0] register, respectively and CLK_VAL1[9:8] into SCU_CLKCR1.[1:0]-this configures clock dividers and selection
- if CLK_VAL2[31]=0 - CLK_VAL2[21:16],[10:0] bits are installed into SCU_CGATCLR0 bit fields [21:16] and [10:0], respectively - this enables the clock of the selected peripherals

In case some CLK_VALx location is not programmed by the user - like in device delivery state - its bit[31]=1 and no installation is done into the respective register(s).

Limited device frequency change

To avoid big jumps/drops in power consumption, the user-configurable value to be installed during start-up from CLK_VAL1 into CLKCR.IDIV is limited to the range 1..16, resulting in possible range of MCLK after re-configuration 3..48 MHz from initial 8MHz - i.e. MCLK frequency can be increased or decreased no more than 4 times (rounded, ignoring potential FDIV influence).

In case CLK_VAL1 (refer to Table 227) contains IDIV value out of the 1..16 range:

- if IDIV=0 - SSW installs IDIV=1 instead
- if IDIV>16 - SSW install IDIV=16 instead

26.1.4 Configuration of Special System Functions as part of User code initialization

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 as part of the User code initialization before the actual user application starts:

- Start Address and Initial Stack Pointer Value
- Setup the Interrupt handler
- Supply Voltage Brown-out Detection
- Watchdog Timer (WDT)

Start Address and Initial Stack Pointer Value

The start address and the inital stack pointer values in Table 227 need to be defined by the user. It can be done together with the downloading of the user code into the flash memory.

Setup the Interrupt Handler

The vector table is remapped to the SRAM based on the remapped vector table in CPU chapter. The interrupt service routine can be placed in these SRAM address or user can also have a branch instruction to the routine that is placed in other memory location. For details, please refer to CPU chapter.

V_{DDP} Brown Out Detection

V_{DDP} 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. An interrupt request will be flagged if a programmed condition is detected. For details, please refer to the Power Control Unit (PCU) section in the SCU 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. In PSC1Mx, the default WDT clock is the standby clock. 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.

26.1.5 Configuration of Clock System and Miscellaneous Functions

The following functions are available and may require configuration in the user code:

- Clock System Configuration
- System Reset Configuration
- Debug Suspend Configuration

Clock System Configuration

MCLK and PCLK frequency can be configured to be different from the user settings during SSW execution as described in Clock system handling by SSW. The SCU_CRCLK.IDIV/FDIV bits are used to program the target clock frequency. The ratio between MCLK and PCLK is also selectable via bit SCU_CRCLK.PCLKSEL. In addition, some of the peripheral clocks can be enable/disable via the SCU_CGATCLR0/SCU_CGATSET0 register respectively. It is also recommended to enable the oscillator watchdog for loss of clock detection.

System Reset Configuration

Several events such as Flash ECC error or SRAM parity error can be used to trigger a system reset using the SCU_RSTCON register. For details, please refer to the reset control unit (RCU) section in the SCU chapter.

Debug Suspend Configuration

The PSC1Mx device supports a suspend capability for peripherals, if the program execution of the CPU is stopped by the debugger, for example, with a breakpoint,or with the C_HALT. This allows the debugging of critical states of the whole microcontroller. The suspend function has to be enabled or disabled locally at the peripheral based on the application and the debugging approach. Refer to Debug System chapter for details.

26.2 Start-up Modes

Startup Software (short name SSW) is the first software executed by CPU after any device leaves reset state.

SSW is stored in the ROM memory, but nevertheless its flow is influenced by other “flexible” factors as:

- type of the event triggering SSW execution
- start-up configuration as selected by the user in Boot Mode Index (short notation BMI)

26.2.1 Start-up modes in PSC1Mx

Start-up mode selection in PSC1Mx is done by the SSW after reset. For the first start-up, it is based on Boot Mode Index (BMI) stored in flash configuration sector 0 (CS0). Subsequently, there is also an option to boot via pins by programming BMI.PINDIS with 0. Upon master reset, the values at at the boot pins P4.6 (STSTAT.HWCON[0]) and P4.7 (STSTAT.HWCON[1]) are latched in.

Table 225 shows the boot pin encoding and the boot modes.

Table 225 Boot pin modes

HWCON[1]	HWCON[0]	Boot Mode
0	0	User Productive Mode
0	1	ASC BSL
1	0	Alternate Boot Mode
1	1	CAN BSL[1)]

1) A programmed value of BMI.BSLTO results in a time-out duration defined by the bit field. If time-out duration is not intended, BMI.BSLTO shall be configured with 0.

To program a new BMI, user need to call the user function “Request BMI installation” as described in the BSL and User Routines chapter. The selection and handling of BMI by SSW is described in Start-up mode selection. The default start-up mode in device delivery state is the ASC BSL mode.

A summary of the supported start-up modes follows.

26.2.1.1 User productive mode

User code is started from Flash memory, no debugging is possible in this mode.

The address of the first user instruction - to where SSW jumps at its end - is taken from Flash location $1000'1004_H$.

SSW does not check either the target address is inside user Flash. Therefore HardFault will be generated if it’s outside and the execution will jump to exception handler in SRAM (upon master reset an endless loop is installed by SSW).

26.2.1.2 User mode with debug enabled

User code is started from Flash memory, the first address is determined as in User productive mode.

Debug interface is configured according to Boot Mode Index (BMI) value and enabled.

26.2.1.3 User mode with debug enabled and Halt After Reset (HAR)

User code is started from Flash memory as in the two modes above.

Debug interface is configured according to BMI value and enabled, the CPU is stalled upon exit from SSW and before the first user instruction (for the handling refer to Debug system handling).

26.2.1.4 Alternate Boot Mode (ABM)

In this mode, user defined bootloader (application) which resides in the user-defined flash address has to be first downloaded via standard bootstrap loader or SWD. This must be followed by a change in the BMI to boot via pins and the selection of ABM boot mode. For new boot mode to take effect, a master reset must be performed.

SSW branches to the user defined bootloader only with a matching magic key, defined in Table 226.

The ABM header is placed at the last 32 bytes of the user flash area, start address at $1000\ 0000_H$ + (FLSIZE.ADDR<<12-20_H).

Table 226 Alternate Boot Mode Header (ABMHD) structure

Offset Address	Size (Byte)	Field Name	Description
00_H	4	MAGICKEY	Magic Key = $A5C3E10F_H$
04_H	4	STADDABM	Start address of the User Code of ABM
08_H	4	CODELENGTH	Length of the User Code of ABM

26.2.1.5 Standard Bootstrap Loader modes

In this mode (short notation Standard BSL):

- user code is downloaded into SRAM
- at the SSW end the execution jumps to SRAM
- no debugging is possible

The Standard BSL mode supported in PSC1Mx uses USIC0 module for communication with the host:

- ASC - using channel 0 or 1 (auto-detection by firmware) of USIC0 module and ASC (UART) protocol for download

  This is the default start-up mode - selected by BMI value in device delivery state.
- SSC - using USIC0 channel 0 and SSC (SPI) standard protocol for download

  In this mode SPI-compatible serial EEPROM is supposed to be connected as slave device to PSC1Mx.

The functionality of standard BSL routines is described in the BSL chapter.

26.2.1.6 Bootstrap Loader modes with time-out

These modes are using the Standard BSL routines for downloading, but the functionality is different:

- SSW first checks either an external device is connected/active, meaning SSW waits to receive Start and Header Bytes from the host - either via USIC channel 0 or 1 - for time-out which duration is taken from BMI.BLSTO:
  - if yes
    - user code is downloaded into SRAM
    - at the SSW end the execution jumps to SRAM
  - if not - at the SSW end the execution jumps to Flash as in User productive mode
- no debugging is possible in this mode, independently either code execution starts from Flash or SRAM

Three BSL modes with time-out are supported in PSC1Mx, using USIC0 module or CAN for communication with the host. Besides the difference in interface selection, also the check performed as part from the above sequence is different:

- ASC mode
  - SSW configures USIC0 channels 0 and 1 in ASC (UART) mode
  - SSW waits to receive Start and Header Bytes from the host - either via channel 0 or 1 - for time-out which duration is taken from BMI.BLSTO
  - if Start+Header Bytes are received - Standard ASC BSL is further executed for downloading and starting user code from SRAM; otherwise execution from Flash will be taken
- CAN mode
  - SSW configures the clock selection based on BMI.CANCLK as the system clock.
  - SSW configures CAN node 0 in bit timing node.
  - SSW waits to receive the initialisation frame. If the frame is received within the time-out, CAN BSL is further executed for downloading and starting user code from SRAM; otherwise execution from Flash will be taken for BMI.BSLTO > 0. If BMI.BSLTO = 0, endless loop is executed.
- SSC mode
  - SSW configures USIC0 channel 0 as Master in SSC (SPI) mode
  - SSW initiates communication with SPI-compatible serial EEPROM supposed to be connected as slave device
  - if data received upon Read Command sent by SSW corresponds to the expected (serial EEPROM) protocol - Standard SSC BSL is further executed for downloading and starting user code from SRAM; otherwise execution from Flash will be taken

***Note**: If BSLTO=0 is configured in BMI - SSW at all does not perform any BSL check but directly takes User Productive Mode for ASC BSL with time-out mode.*

26.2.2 Boot Mode Index (BMI)

The Boot Mode Index is 2 Byte value stored in Flash (refer to Table 227) and holding information about start-up mode and debug configuration of the device. Additionally to BMI at $1000'0E00_H$ its inverse value is stored at $1000'0E10_H$ for check purposes.

BMI	**Address:**	CS0 offset $0E00_H$
Boot Mode Index	**Reset value:**	$FFC0_H$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BSLTO				CAN CLK	DAPDIS		DAP TYP	PIND IS	HWCFG						
r				r	r		r	r	r						

Field	Bits	Type	Description
HWCFG	6:0	r	**Start-up Mode Selection:** 0000000_B CAN Bootstrap Loader mode (CAN_BSL) 0010000_B CAN BSL mode with time-out (CAN_BSLTO) 0100000_B Secure Bootstrap Loader mode over CANopen (SBSL CANopen) 1000000_B ASC Bootstrap Loader mode (ASC_BSL) 1000001_B User productive Mode (UPM) 1000011_B User mode with debug enabled (UMD) 1000111_B User mode with debug enabled and HAR (UMHAR) 1001000_B SSC Bootstrap Loader mode (SSC_BSL) 1010000_B ASC BSL mode with time-out (ASC_BSLTO) 1011000_B SSC BSL mode with time-out (SSC_BSLTO) 1111010_B Secure Bootstrap Loader mode over ASC (SBSL ASC) else Reserved [1)]
PINDIS	7	r	**Boot Configuration Type Selection** 0_B Boot from pins is selected 1_B Boot from BMI is selected
DAPTYP	8	r	**DAP Type Selection** 0_B Serial wire debug interface is selected 1_B Single pin debug interface is selected
DAPDIS	10:9	r	**SWD/SPD Input/Output Selection** 00_B SWD/SPD_0 pin is selected 01_B SWD/SPD_1 pin is selected 10_B Reserved 11_B Reserved
CANCLK	11	r	**CAN Clock Source for CAN BSL Mode** 0_B Synchronous CAN clock via internal oscillator (DCO1) with enabled trimming via external reference is selected 1_B Synchronous CAN clock via external oscillator (OSC_HP) is selected
BSLTO	15:12	r	**ASC BSL Time-out value** The time-out duration is BSLTO*2664000 MCLK cycles, the supported time-out range is 0.3-5s (333...4995ms)

1) ATTENTION: Installing a reserved BMI value will cause device delivery state to be restored upon the next reset - refer to Start-up mode selection

26.2.3 Start-up mode selection

The start-up modes in PSC1Mx are selected and handled according to Boot Mode Index (BMI) value read from Flash - refer to Start-up modes in PSC1Mx.

The BMI evaluation can lead to:

- taking User mode - SSW will jump at its end into User Flash and start execution from the location which address is stored at $1000'1004_H$ if:
  - a kind of User Mode is selected in BMI
  - ASC BSL with time-out is selected but no valid Start+Header Bytes are received within the selected time frame
- taking Bootstrap Loader mode - executing ASC/SSC or CAN BSL[25] then jumping into SRAM at $2000'0200_H$
- taking Secure Bootstrap Loader (SBSL) mode[25]
- taking Alternate Boot Mode - SSW jumps to User Flash address defined in STADDABM in the Alternate Boot Mode Header
- executing (endless) loop - waiting for valid data to be received via SSC upon SSC BSL mode without time-out, or valid initialization frame via CAN BSL mode without time-out
- erasing the complete user Flash and installing ASC BSL (or SBSL if supported) mode as new BMI value - upon invalid BMI value or if Secure BSL or CAN BSL is selected but not supported for the device. Afterwards master reset is triggered and device is started in a mode according to the new BMI.

26.2.3.1 BMI handling by SSW

This SSW part is intended to be executed after the user function (in ROM) “Request BMI installation” (described in the BSL and User Routines chapter) has been called and has triggered a system reset. Therefore the conditions under which this handling is performed are:

- the last reset was a system reset requested by CPU
- the two half words of SSW0 register are inverse to each other

If all the above conditions are true, the SSW evaluates BMI.PINDIS to determine if boot via pins is the chosen boot option. STSTAT.HWCON determines the actual boot mode via pins. For non pin-booting, SSW considers SSW0[15:0] content as new BMI value to be installed and executes the following:

- check either the current BMI (in CS0) is User_Productive AND the new value is NOT User_Productive:
  - if yes - erase the complete user Flash and install ASC_BSL (or SBSL if supported) mode as new BMI value
  - if not - install the new BMI value from SSW0[15:0]
- instal all zero in SSW0 to indicate no BMI-programming upon the next reset
- request master reset to be triggered

Upon the last action, a master reset is triggered and a next SSW execution will start with the new BMI value installed in CS0.

[25] If available in device and selected in BMI.

26.2.3.2 Debug system handling

If debug system must be enabled - start-up mode with debug support is selected in Boot Mode Index - SSW performs the following:

- configure the debug interface from BMI value
- enable the debug interface
- if start-up mode with Halt After Reset request (UMHAR) is selected in BMI upon master reset - enter an endless NOP (no operation) loop. From this point on, an external tool (debugger) can take over the control of the device, in particular:
  - if/when a debugger connects to device, it can halt the CPU and check BMI and/or the core program counter (PC) register
  - if HAR is selected, this can be identified by BMI value and the PC value will point inside the ROM. The debugger can then manipulate PC and start the user code as desired - the default start address (taken without HAR) is according to the content at location 1000'1004$_H$ in user Flash (refer to Table 227)

26.3 Data in Flash for SSW and User SW

Table 227 shows the data in Flash which is relevant for SSW execution and can be used by user software in PSC1Mx.

Table 227 Flash data for SSW and user SW in PSC1Mx

Address	Length	Function	Target location
Start-up mode selection:			
1000'0E00$_H$	2 B	Boot Mode Index (BMI)	---
1000'0E10$_H$	2 B	Inverse BMI	---
Chip Identification:			
1000'0F00$_H$	4 B	Chip ID	SCU_IDCHIP
1000'0F04$_H$	28 B	Chip Variant Identification Number	---
1000'0FF0$_H$	16 B	Unique Chip ID	---
Temperature-sensor and DCO related data:			
1000'0F20$_H$	16 B	Temperature-sensor constant data 1	---
1000'0F30$_H$	1 B	ANA_TSE_T1	---
1000'0F31$_H$	1 B	ANA_TSE_T2	---
1000'0F32$_H$	1 B	DCO_ADJLO_T1	---
1000'0F33$_H$	1 B	DCO_ADJLO_T2	---
1000'0F34$_H$	161 B	Temperature-sensor constant data 2	---
Application software related data:			
1000'1000$_H$	4 B	Initial Stack Pointer value after SSW	SP_main
1000'1004$_H$	4 B	Start address after SSW in User modes	PC
Clock system related data:			

(table continues...)

Table 227 (continued) Flash data for SSW and user SW in PSC1Mx

Address	Length	Function	Target location
$1000'1010_H$	4 B	Clock configuration value CLK_VAL1	If bit[31]=0: Installation of • bits[21:10] into SCU_CLKCR[19:8] • bits[9:8] into SCU_CLKCR1[1:0] • bits[7:0] into SCU_CLKCR[7:0]
$1000'1014_H$	4 B	Clock gating configuration value CLK_VAL2	If bit[31]=0: Installation of • bits[21:16] into SCU_CGATCLR0[21:16] • bits[10:0] into SCU_CGATCLR0[10:0]
$1000'1018_H$	4 B	Wait time before ASC BSL channel selection configuration value WAIT_ASCBSL_ENTRY	If bit[31]=0: Installation of • bits[30:0] for wait time before ASC BSL channel selection

27 Bootstrap Loaders (BSL) and User Routines

This chapter describes the Bootstrap Loaders (BSL) and the user-accessible routines in the ROM memory.

The ASC (UART) BSL (ASC (UART) Bootstrap Loader) and the SSC BSL (SSC Bootstrap loader) is entered with the BMI settings as described in the Boot and Startup Chapter. The main purpose of BSL Mode is to allow easy and quick programming/erasing of the Flash.

Firmware routines available for the user describes user-accessible routines that can be called by application software. These routines are located in the ROM memory.

27.1 ASC (UART) Bootstrap Loader

ASC (UART) Standard Bootstrap Loader (ASC BSL) in PSC1Mx uses USIC0 module to download code/data into SRAM starting at address 2000 0200_H.

After the last code/data Byte is received and stored, the Startup Software starts user code from address 2000 0200_H.

27.1.1 Pin usage

ASC BSL in PSC1Mx is selected by a single BMI value but it allows to download user code/data via several variants of pins, modes and USIC0 channels:

- Channel 0
  - full duplex mode with RxD at P0.14 and TxD at P0.15
  - half duplex mode with RxD/TxD at P0.14
- Channel 1
  - full duplex mode with RxD at P1.3 and TxD at P1.2
  - half duplex mode with RxD/TxD at P1.3

***Note**: The above set of pins also accommodate all the assignments for SWD/SPD (debug) interfaces.*

27.1.2 ASC BSL execution flow

The complete ASC Bootstrap Loader procedure consists of two parts as described below.

27.1.2.1 ASC BSL entry check sequence

PSC1Mx SSW is able to provide wait time before ASC BSL channel selection. The wait time handling is user-configurable by installing the value in a one word location in Flash (refer to Table 227).

- if WAIT_ASCBSL_ENTRY[31]=0 - SSW executes wait loops before ASC BSL channel selection

Downloading is only initiated after a Start (zero) and a Header (described below) Bytes are received through one of the channels whereas both the channels are active (“listening”) until the selection is done by the routine itself as follows:

- Channel selection - based on the fact at which RxD pin (see the above assignments) first Start+Header Bytes are received
- Full/half duplex selection - based on the Header Byte received

***Note**: For definitions of header/acknowledge byte values exchanged with the host - refer to ASC BSL protocol data definitions.*

The ASC bootloader functionality includes:

1. enable and configure the both USIC0 communication channels 0 and 1 as follows: a) ASC mode, 8 data bits, 1 stop bit, no parity
2. configure ASC clock-generation circuitry for measurement
3. perform continuously the following sequence for both USIC0 channels one after another
   a. check either Start zero Byte has been received

      if yes - then:
   b. reconfigure ASC clock-generating circuitry for normal operation according to the baudrate measured from the Start Byte, then:
   c. check either Header Byte has been received

      if yes - then:
   d. check the Header Byte received
      - if equal to BSL_ASC_F or BSL_ENC_F - full duplex mode is selected - continue with step 4.
      - if equal to BSL_ASC_H or BSL_ENC_H - half duplex mode is selected - continue with step 4.
      - otherwise - go to sequence for the other channel
4. select the current (Cy) channel for further handling; restore default configuration for the channel C(y-1) which is not selected
5. for the selected channel and mode (full/half duplex):
   a. configure TxD pin and ASC transmitter
   b. adapt (in case) RxD pin settings
   c. enable transmission, single shot mode
6. check which Header byte was received:
   a. BSL_ASC_F or BSL_ASC_H - standard BSL entry sequence was executed, the communication will use further the baudrate detected - continue with ASC BSL download sequence
   b. BSL_ENC_F or BSL_ENC_H - the communication during ASC BSL download sequence will potentially use different (for example, higher) baudrate than the initially detected - continue with the Enhanced ASC BSL entry sequence

The above check sequence is executed

- in ASC BSL mode without time-out - until valid Start+Header Bytes are received
- otherwise - only for some time according to BMI.BSLTO

***Note**: If mode with time-out is selected but BMI.BSLTO=0 - User mode with start from Flash is directly taken.*

Enhanced ASC BSL entry sequence

This is an extension of the standard ASC BSL entry check sequence and will be used when higher baudrates are required for the communication channel.

It includes the following steps:

1. Send BSL_ENC_ID to acknowledge the establishment of the initial baud rate and entry into enhanced ASC BSL
2. Send current PDIV divider from USIC0_CHy_BRG register - the 10-bit value is sent in 2 bytes (most significant byte first)
3. Receive from host the 10-bit value to be written into the STEP bit field in USIC0_CHy_FDR register
4. Send BSL_BR_OK to acknowledge the establishment of the new baud rate
5. Configure USIC baud rate generator accordingly
6. Receive from host BSL_BR_OK to indicate that the communication channel has been successfully established
   - If no or wrong acknowledge value is received, a device software reset will be triggered

Figure 452 shows the outline of the sequence to configure the baud rate.

Requirements for Host

For the host to support enhanced ASC BSL, the host additionally has to:

- Receive from PSC1Mx device, the current 10-bit PDIV value in USIC0_CHy_BRG register
- Calculate the MCLK that the device is running on based on the received PDIV through the formula:

$$\text{MCLK} = \text{Initial Baud Rate} \times (\text{PDIV} + 1) \times 8 \tag{89}$$

- Select a new baud rate that is supported by the enhanced ASC BSL given the MCLK (see Table 228)
- Calculate the scaling factor, i.e. ratio of the new baud rate to the initial baud rate. Some examples are given in Table 229. The scaling factor is rounded to the nearest integer in order to meet a frequency deviation of +/- 3%.

Table 228 Supported Baud Rates

MCLK (MHz)	Standard baud rates supported with ASC BSL (kHz)		Max. baud rate supported with Enhanced ASC BSL (MHz)
	Min.	Max.	
8	1.2	28.8	0.999
16	2.4	57.6	1.998
32	4.8	115.2	3.996
48	9.6	153.6	5.994

Table 229 Scaling Factor Examples

Initial standard baud rates (kHz) ---A	Targeted higher baud rates (kHz) ---B	Scaling factor rounded to the nearest integer (B/A)
9.6	1500	156
19.2	1500	78
57.6	1500	26
115.2	1500	13

- Calculate the 10-bit value to be written into the STEP bit field of USIC0_CHy_FDR register through the formula:

$$\text{STEP}\left(\text{in fractional divider mode}\right) = \frac{1024 \times \text{Scaling Factor}}{\text{PDIV} + 1} \tag{90}$$

- Send the 10-bit STEP value to the device
- Wait until the BSL_BR_OK is received from the device
- Echo the BSL_BR_OK back to the device

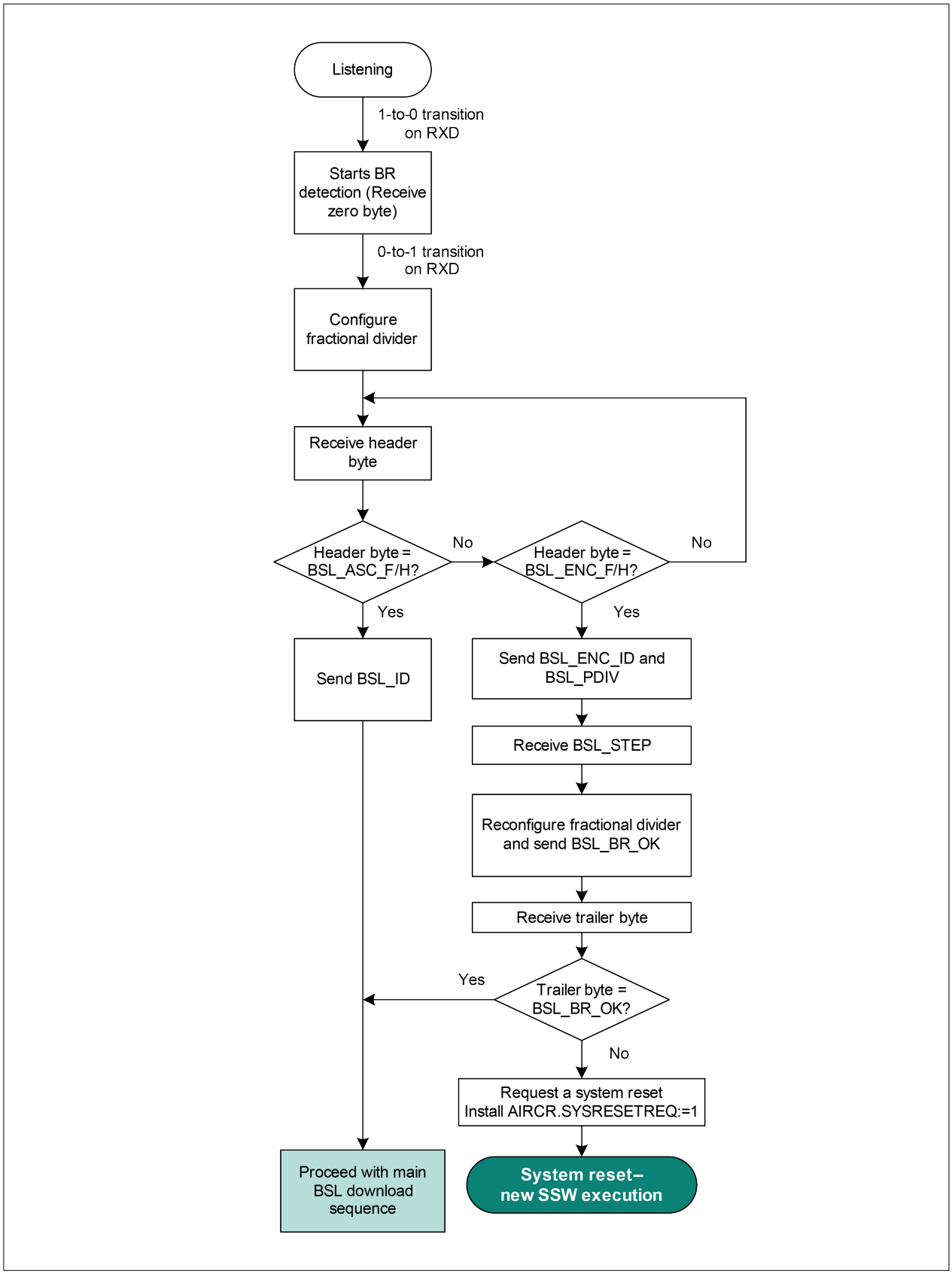


Figure 452 Baud Rate configuration sequence during PSC1Mx ASC BSL entry

27.1.2.2 ASC BSL download sequence

After the baud rate has been detected/configured and channel/mode (full/half duplex) selected, ASC BSL awaits 4 bytes describing the length of the application from the host (refer to Figure 453). The least significant byte is received first.

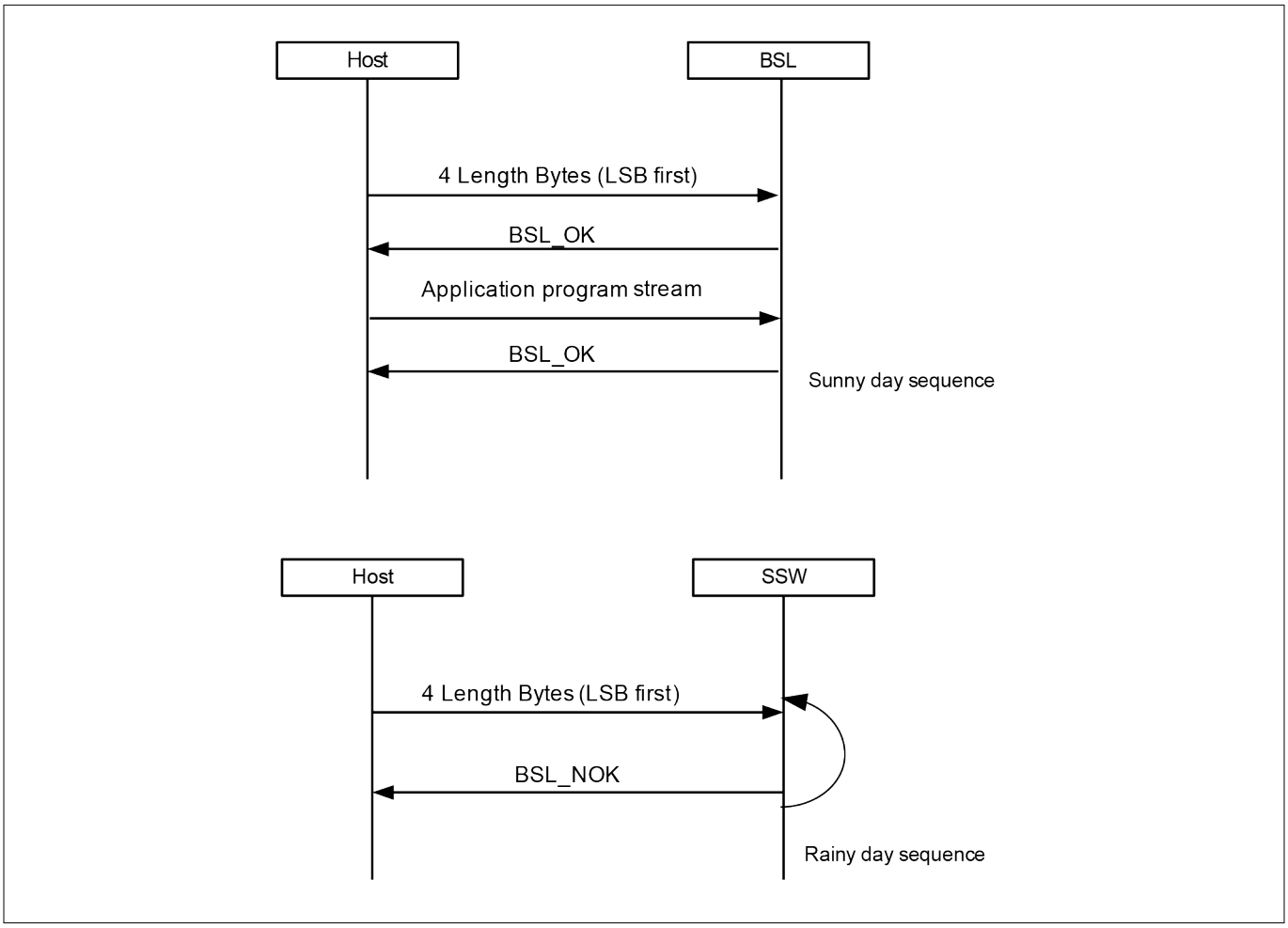


Figure 453 PSC1Mx Standard ASC BSL: Application download protocol

If application length is found alright by SSW, a BSL_OK byte is sent to the host following which the latter sends the byte stream belonging to 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 SRAM size), a BSL_NOK byte is transmitted back to the host and the SSW resumes awaiting the length bytes.

27.1.3 ASC BSL protocol data definitions

The interaction between PSC1Mx ASC Bootstrap Loader routine and the host implements handshake protocol based on Figure 454 and request/acknowledge/data Bytes defined in Table 230.

Table 230 Handshake protocol data definitions in PSC1Mx ASC BSL

Name	Length, Byte	Value	Description
Requests/data/acknowledge sent by the Host:			
BSL_ASC_F	1	$6C_H$	Header requesting full duplex ASC mode with the current baud rate
BSL_ASC_H	1	12_H	Header requesting half duplex ASC mode with the current baud rate
BSL_ENC_F	1	93_H	Header requesting full duplex ASC mode with a request to switch the baud rate
BSL_ENC_H	1	ED_H	Header requesting half duplex ASC mode with a request to switch the baud rate
BSL_STEP	2	$0XXX_H$	10-bit value (LSB aligned) to be programmed into selected USIC channel's FDR.STEP bit field. Most significant 6 bits should contain all 0.
BSL_BR_OK	1	$F0_H$	Final baud rate is established in enhanced ASC BSL
Acknowledges sent by PSC1Mx firmware:			
BSL_ID	1	$5D_H$	Start and header bytes are received, baud rate is established
BSL_ENC_ID	1	$A2_H$	Start and header byte(s) are received in enhanced ASC BSL, initial baud rate is established
BSL_PDIV	2	$0XXX_H$	10-bit value (LSB aligned) containing the selected USIC channel's BRG.PDIV bit field value. Most significant 6 bits should contain all 0.
BSL_BR_OK	1	$F0_H$	Final baud rate is established in enhanced ASC BSL.
BSL_OK	1	01_H	Data received is OK
BSL_NOK	1	02_H	Failure encountered during data reception

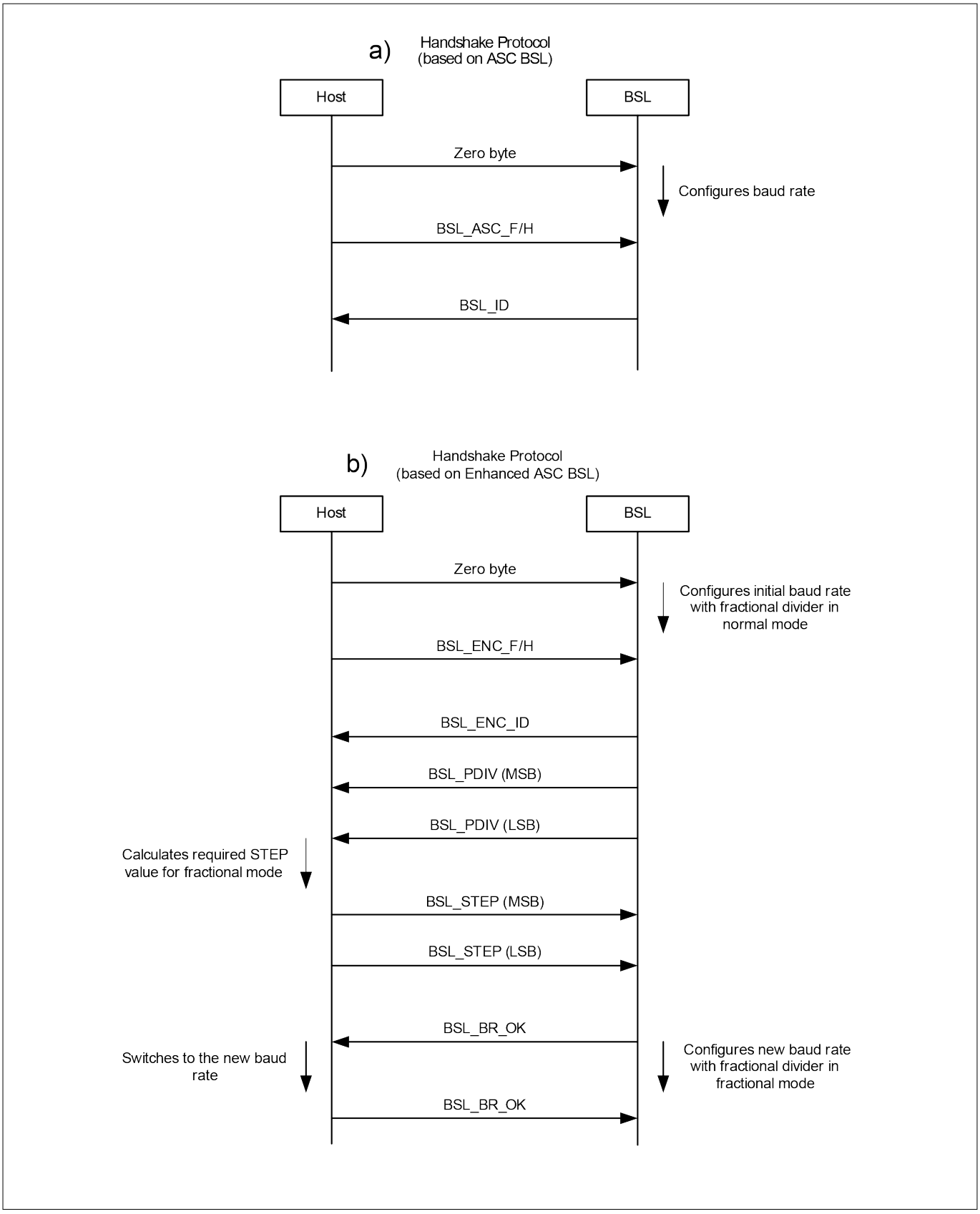


Figure 454 Handshake protocol for PSC1Mx ASC BSL entry

27.2 SSC Bootstrap loader

This procedure downloads code from a SPI-compatible serial EEPROM into SRAM starting at address 2000 0200_H up to the user-available SRAM size, using Channel 0 of USIC0 Module.

PSC1Mx is the master SPI-device, following pins are used by the bootloader:

- MRST (master data input) and MTSR (master data output) - on the same P0.15, half-duplex mode used, open drain
- SCLK (clock signal) - P0.14, open drain
- SLS (chip-select $\overline{CS}$ to the EEPROM) - P0.13, open drain

***Note**: Due to PSC1Mx pin configured as open drain, external pull-up resistors are required to be added on to the bus for each of the above pins.*

***Note**: The EEPROM pins SI (Serial Data Input) and SO (Serial Data Output) have to be connected to each other before connecting to PSC1.*

***Note**: The EEPROM pins $\overline{WP}$ and $\overline{HOLD}$ are not used by the SSC bootloader. An external circuitry is required. (for example, pull-up from V_{DD})*

A SPI-compatible serial EEPROM of type 25xxx must be connected to the pins as above defined. The clock signal is configured at MCLK/10 frequency upon device startup.

The SSC bootloader in PSC1Mx is able to communicate with a very broad range on serial-EEPROM types: from such using 8-bit addressing (size up to 2K bit, quite outdated already) up to devices using 24-bit addressing (1M bit and above). The connected EEPROM type is determined by examining the received header bytes, as indicated in Table 231.

***Note**: Besides the bootloader supports all the device-types, the code/data length which can be downloaded is limited by the size of available SRAM.*

Table 231 SSC BL: Determining the EEPROM Type and data-flow

SSC Frame		EEPROM with 8-bit addressing connected		EEPROM with 16-bit addressing connected		EEPROM with 24-bit addressing connected	
N	data	PSC1Mx-send	PSC1- receive	PSC1Mx-send	PSC1- receive	PSC1Mx-send	PSC1- receive
1	03_H	Read command	XX_H default level	Read command	XX_H default level	Read command	XX_H default level
2	00_H	Address	XX_H	Address_H	XX_H	Address_ H	XX_H
3	00_H/ FF_H[1)]	dummy	Identification	Address_L	XX_H	Address_ M	XX_H
4	00_H/ FF_H[1)]	dummy	Size	dummy	Identification	Address_ L	XX_H
5	FF_H	dummy	Data Byte 1	dummy	Size, High B	dummy	Identification
6	FF_H	dummy	Data Byte 2	dummy	Size, Low B	dummy	Size, High B
7	FF_H	dummy	Data Byte 3	dummy	Data Byte 1	dummy	Size, Mid B
8	FF_H	dummy	Data Byte 4	dummy	Data Byte 2	dummy	Size, Low B

(table continues...)

Table 231 (continued) SSC BL: Determining the EEPROM Type and data-flow

SSC Frame		EEPROM with 8-bit addressing connected		EEPROM with 16-bit addressing connected		EEPROM with 24-bit addressing connected	
9 ...	FF_H	dummy	Data Byte 5 ...n	dummy	Data Byte 3 ...n	dummy	Data Byte 1 ...n

1) Depending on which EEPROM type is checked.

The EEPROM connected for downloading must contain:

- Identification Byte $5D_H$ at the first address (0000_H)
- The length of the code (Code_Length) in Bytes as follows:
  - in case of an EEPROM with 8-bit addressing - in one Byte at address 0001_H
  - in case of an EEPROM with 16-bit addressing - in two Bytes at addresses 0001_H / 0002_H ordered high/low
  - in case of an EEPROM with 24-bit addressing - in three Bytes at addresses 0001_H / 0002_H / 0003_H ordered high/middle/low
- The code of defined length follows sequentially

The SSC bootloader functionality includes:

1. Enable the communication channel by setting USIC0_C0_KSCFG.MODEN:=1
2. Assure clock gating for USIC0 module is de-asserted
3. Configure the USIC0_C0 channel as follows:
  a. SSC mode, 8 data bits, MSB first
  b. SCLK frequency = MCLK/10
  c. no SLS activated (only one serial EEPROM is supported)
  d. pin-configuration as defined above
  e. enable transmission
4. Deselect EEPROM by setting $\overline{CS}$=1
5. Receive data from 8-bit EEPROM
  a. enable EEPROM ($\overline{CS}$=0)
  b. send Read command (03_H)
  c. send address byte (00_H) for data to be received
  d. send read response byte (FF_H)
6. Check the response based on the last received byte:
  a. if equal to the Identification Byte ($5D_H$) - the EEPROM uses 8-bit addressing:
    - SSW reads one more byte and saves it as Code_Length - 1B only effective
    - continue with p.11
  b. if not equal to the Identification Byte ($5D_H$) - the EEPROM does not use 8-bit addressing:
    - deselect EEPROM by setting $\overline{CS}$=1
7. Receive data from 16-bit EEPROM
  a. enable EEPROM ($\overline{CS}$=0)
  b. send Read command (03_H)
  c. send 2 address bytes (00_H, 00_H) for data to be received
  d. send read response byte (FF_H)

8. Check the response based on the last received byte:
   a. if equal to the Identification Byte ($5D_H$) - the EEPROM uses 16-bit addressing:
      - SSW reads two Bytes in order High/Low to determine Code_length (2 Bytes)
      - continue with p.11
   b. if not equal to the Identification Byte ($5D_H$) - the EEPROM does not use 16-bit addressing:
      - deselect EEPROM by setting $\overline{CS}$=1
9. Receive data from 24-bit EEPROM
   a. enable EEPROM ($\overline{CS}$=0)
   b. send Read command (03_H)
   c. send 3 address bytes (00_H, 00_H, 00_H) for data to be received
   d. send read response byte (FF_H)
10. Check the response based on the last received byte:
   a. if equal to the Identification Byte ($5D_H$) - the EEPROM uses 24-bit addressing:
      - SSW reads three more Bytes in order High/Middle/Low to determine Code_length (3 Bytes)
      - continue with p.11
   b. if not - exit the sequence
11. Check the value of Code_length:
   a. if greater than allowed available SRAM in PSC1Mx, exit the sequence
   b. otherwise, continue with p.12
12. Send one FF_H byte for each byte of user code. Read user code bytes [Code_Length] and store them sequentially from the beginning of SRAM (address 2000 0200_H) onwards, disable EEPROM ($\overline{CS}$=1).
13. Disable the communication channel by resetting USIC0_C0_KSCFG.MODEN:=0
14. Set SSW flag BSL_act=1 and exit the sequence

The Startup Software starts the downloaded code from address 2000 0200_H.

27.3 CAN BSL mode

The CAN bootstrap loader mode transfers user application via Node-0 or Node-1 of the CAN module into SRAM.

Users need to adhere to the three phases of the protocol for user application to be downloaded. The size of the user-available SRAM determines the maximum size of the downloadable application code.

Figure 455 shows the CAN BSL mode procedures.

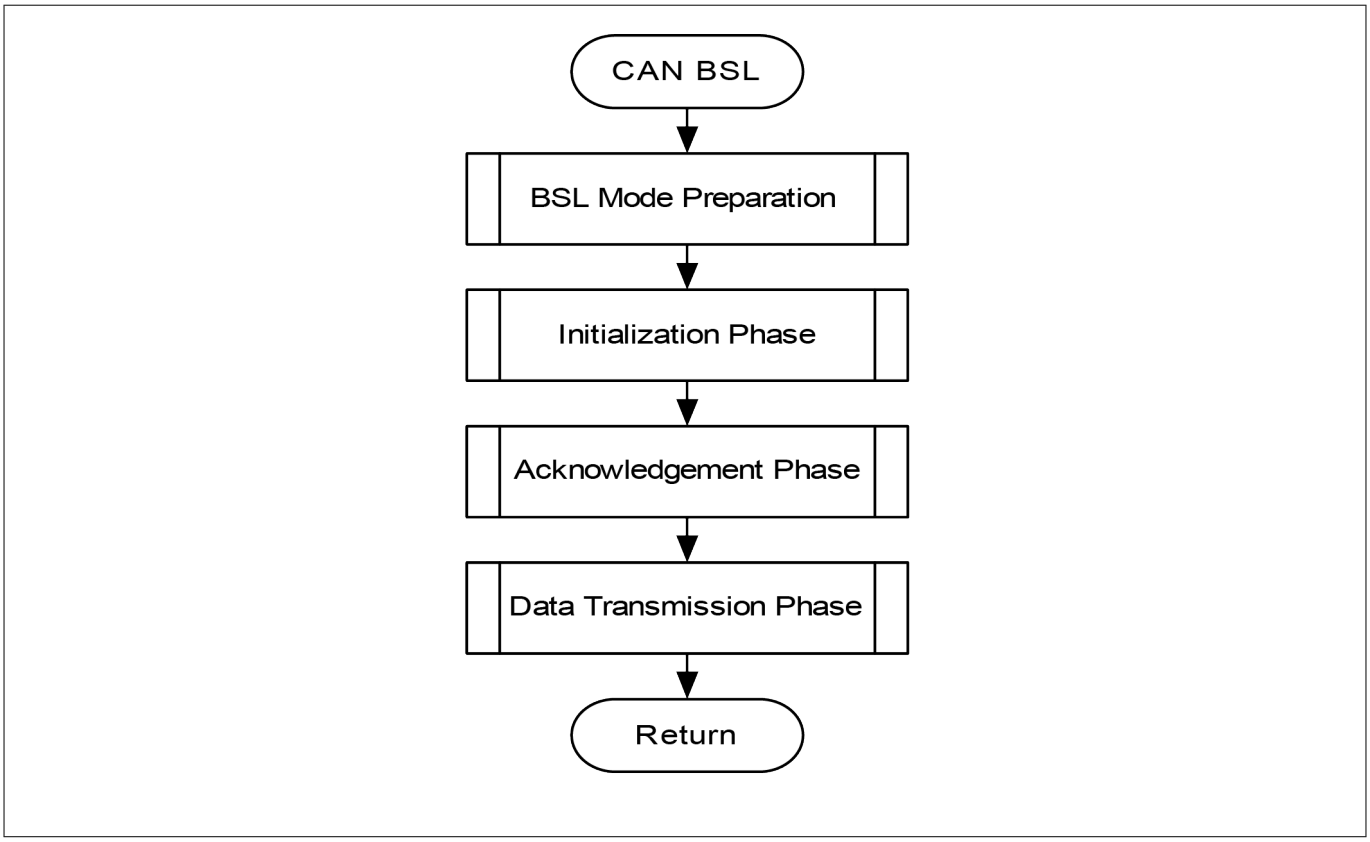


Figure 455 CAN BSL Sequence

27.3.1 Initialization Phase

SSW switches to the synchronous CAN clock source via internal oscillator or external oscillator, selected by BMI.CANCLK. If the internal oscillator is selected, f_{MCLK} is configured for 24MHz. For external oscillator selection, a stable external clock (OSC_HP) is mandatory. For transfer rates of 1 Mbps, the user has to ensure that the external clock is at least 10 MHz. The SSW determines the external clock frequency and configures f_{MCLK} to have the same frequency as f_{OSC_HP}. Independent of the CAN clock source selection, the CAN baud logic clock is configured as f_{MCLK} in MCR.CLKSEL.

In this phase, the CAN baud rate of the host is determined. A standard CAN initialization frame comprising eight data bytes is transmitted continuously by the external host. The LSB-aligned 11 bits message ID (0x555) of the frame is used for baud rate detection. Data bytes 2 and 3 contain the acknowledge identifier (ACKID). The SSW will send the ACKID back to the host in the acknowledge frame to indicate the completion of the initialization phase. The next 2 bytes (byte 4 and 5) contains the message count value, DMSGC. DMSGC indicates the number of eight byte data frames of user application which must be received by SSW and placed into the SRAM. The total size of the data frames should be selected to be smaller or equal than the available SRAM size. The last 2 bytes (byte 6 and 7) is the DMSGID identifier that is to be used in the Data frame to transmit the user application.

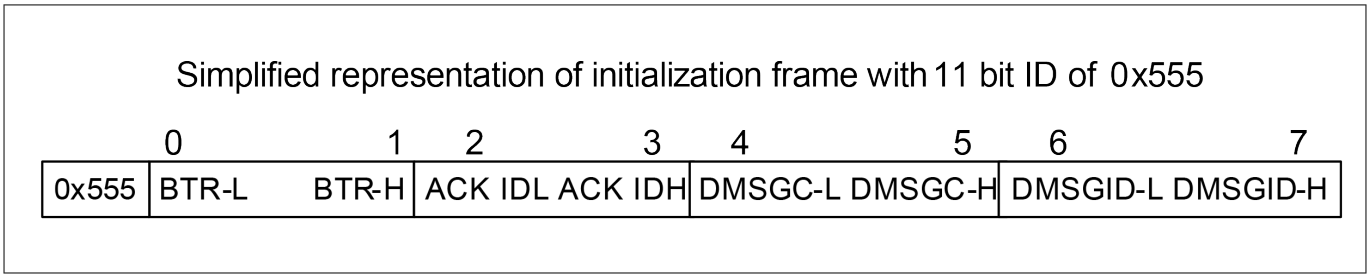


Figure 456 Data field of the Initialization Frame

Port pins used are P0.14 / P0.15 or P1.2 / P1.3. If SSW detects initialization frame on P0.14, it would configure P0.15 for TX functionality on Node 0. In contrary, if SSW detects initialization frame on P1.3, SSW configures P1.2 for TX functionality instead for Node 1.

27.3.2 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. This ACK-ID is used as the LSB-aligned 11-bit message ID and for data bytes 2 and 3. The configured NBTR register value is captured through data bytes 0 and 1. If the size of application intended to be downloaded is greater than the size of SRAM on the device, a negative acknowledgement as depicted below is sent, where data bytes 4-7 contains the error code of 0xDEADBEEF. SSW enters acknowledgement phase again. If the size of the application is smaller or equal to the PSRAM size, then data byte 4 to 7 of the frame is a direct copy of the equivalent data bytes of the initialization frame. Figure 457 depicts data part of acknowledgement frame.

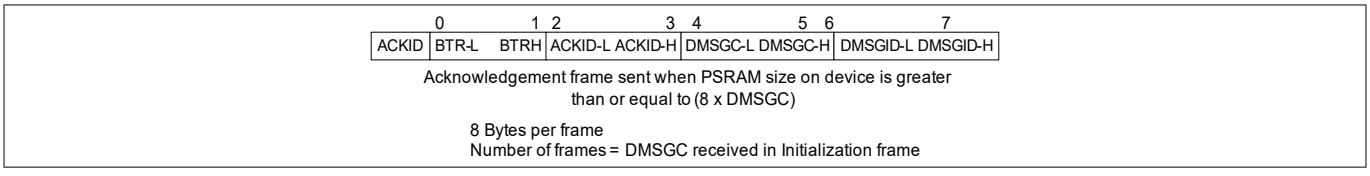


Figure 457 CAN Acknowledgement frame

27.3.3 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. 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.

The first data byte received is stored in the SRAM address 2000 0200H. Subsequent data bytes are stored at incrementing address. After all the frames has been received, SSW continues its startup activities.

27.4 Firmware routines available for the user

Several user routines (refer to Table 232) are available inside ROM so they can be called by application software.

Table 232 User routines' in PSC1Mx ROM

PSC1Mx ROM		Description
Address	**Content**	
$0000\ 0100_H$	_NvmErase	Pointer to Erase Flash Page routine
$0000\ 0104_H$	_NvmProgVerify	Pointer to Erase, Program & Verify Flash Page routine
$0000\ 0108_H$	_BmiInstallationReq	Pointer to Request BMI installation routine
$0000\ 010C_H$	_CalcTemperature	Pointer to Calculate chip temperature routine
$0000\ 0110_H$	_NvmEraseSector	Pointer to Erase Flash Sector routine
$0000\ 0114_H$	_NvmProgVerifyBlock	Pointer to Program & Verify Flash Block routine
$0000\ 0120_H$	_CalcTSEVAR	Pointer to Calculate target level for temperature comparison routine
$0000\ 0124_H$	_DCOCalc	Pointer to DCO Calibration routine

NVM-related functions (see Erase Flash Page and Erase, Program & Verify Flash Page) return status indication as shown in Table 233.

Table 233 Status indicators returned by NVM routines in PSC1Mx ROM

Status Indicator		Description
Symbolic name	**Value**	
NVM_PASS	$0001\ 0000_H$	Function succeeded
NVM_E_FAIL	$8001\ 0001_H$	Generic error
NVM_E_SRC_ AREA_EXCEEDED	$8001\ 0003_H$	Source data is not in RAM
NVM_E_SRC_ ALIGNMENT	$8001\ 0004_H$	Source data is not 4 Byte aligned
NVM_E_NVM_FAIL	$8001\ 0005_H$	NVM module can not be physically accessed
NVM_E_VERIFY	$8001\ 0006_H$	Verification of the written page not successful
NVM_E_DST_ AREA_EXCEEDED	$8001\ 0009_H$	Destination data is not (completely) located in NVM
NVM_E_DST_ ALIGNMENT	$8001\ 0010_H$	Destination data is not properly aligned

The description below provides an overview of the features.

27.4.1 Erase Flash Page

PSC1Mx Flash can be erased with granularity of one page = 16 blocks of 16 Bytes = 256 Bytes using this routine.

- Input parameter
  - logical address of the Flash Page to be erased, must be page aligned and in NVM address range (pageAddr)
- Return status (refer to Table 233)
  - OK (NVM_PASS)
  - invalid address (NVM_E_DST_AREA_EXCEEDED, NVM_E_DST_ALIGNMENT)
  - operation failed (NVM_E_FAIL, NVM_E_NVM_FAIL, NVM_E_VERIFY)
- Prototype

  NVM_STATUS PSC1Mx_NvmErasePage (unsigned long pageAddr)

27.4.2 Erase, Program & Verify Flash Page

This function performs erase (skipped if not necessary), program and verify of selected Flash page.

- Input parameters
  - logical address of the target Flash Page (dstAddr)
  - address in SRAM where the data starts (srcAddr)
- Return status (refer to Table 233)
  - OK (NVM_PASS)
  - invalid addresses (NVM_E_SRC_AREA_EXCEEDED, NVM_E_SRC_ALIGNMENT, NVM_E_DST_AREA_EXCEEDED, NVM_E_DST_ALIGNMENT)
  - operation failed (NVM_E_FAIL, NVM_E_NVM_FAIL, NVM_E_VERIFY)
- Prototype

  NVM_STATUS PSC1Mx_NvmProgVerify (unsigned long srcAddr, unsigned long dstAddr)

27.4.3 Request BMI installation

This procedure initiates installation of a new BMI value. In particular, it can be used as well as to restore the state upon delivery for a device already in User Productive mode.

- Input parameter
  - BMI value to be installed (requestedBmiValue)
- Return status - only upon error, if OK the procedure triggers a reset respectively does not return to calling routine
  - wrong input BMI value (0001_H)
- Prototype

  unsigned long PSC1Mx_BmiInstallationReq (unsigned short requestedBmiValue)

27.4.4 Calculate chip temperature

This procedure calculates the current chip temperature as measured by the PSC1Mx built-in sensor, based on data from Flash including trimming values and pre-calculated constants (refer to Table 234) and data from the actual measurement (read from Temperature Sensor Counter2 Monitor Register ANATSEMON).

- Input parameter
  - none

- Return status
  - chip temperature in degree Kelvin
- Prototype
  unsigned long PSC1Mx_CalcTemperature (void)

27.4.5 DCO Calibration

This procedure calibrates the DCO, based on the measured chip temperature using the PSC1Mx built-in sensor and data from Flash (refer to Table 234). An offset value is calculated and used to start the DCO calibration.

- Input parameter
  - none
- Return status
  - OK
  - operation failed
- Prototype
  unsigned long PSC1Mx_CalibrateDCO (void)

27.4.6 Erase Flash Sector

PSC1Mx Flash can be erased with granularity of one sector = 16 pages of (16 blocks of 16 Bytes) = 4K Bytes using this routine.

- Input parameter
  - logical address of the Flash Sector to be erased, must be in NVM address range (sectorAddr)
- Return status (refer to Table 233)
  - OK (NVM_PASS)
  - invalid address (NVM_E_DST_AREA_EXCEEDED, NVM_E_DST_ALIGNMENT)
  - operation failed (NVM_E_FAIL, NVM_E_NVM_FAIL, NVM_E_VERIFY)
- Prototype
  NVM_STATUS PSC1Mx_NvmEraseSector (unsigned long sectorAddr)

27.4.7 Program & Verify Flash Block

PSC1Mx Flash can be programmed and verified with granularity of one block (4 words of 4 Bytes) = 16 Bytes using this routine.

- Input parameter
  - logical address of the target Flash Block (dstAddr)
  - address in SRAM where the data starts (srcAddr)
- Return status (refer to Table 233)
  - OK (NVM_PASS)
  - invalid addresses (NVM_E_SRC_AREA_EXCEEDED, NVM_E_SRC_ALIGNMENT, NVM_E_DST_AREA_EXCEEDED, NVM_E_DST_ALIGNMENT)
  - operation failed (NVM_E_FAIL, NVM_E_NVM_FAIL, NVM_E_VERIFY)
- Prototype
  NVM_STATUS PSC1Mx_NvmProgVerifyBlock (unsigned long srcAddr, unsigned long dstAddr)

27.4.8 Calculate target level for temperature comparison

This procedure - a kind of reverse of Calculate chip temperature - calculates the value which must be installed in ANATSEIH or ANATSEIL registers to get indication in SRRAW.TSE_HIGH or SRRAW.TSE_LOW bits when the chip temperature is above/below some target/threshold in ANATSEMON.

- Input parameter
  - threshold temperature in degree Kelvin - allowed range 233...388 (temperature)
- Return status
  - equivalent sensor threshold value for the temperature provided as input parameter
- Prototype
  unsigned long PSC1Mx_CalcTSEVAR (unsigned long temperature)

27.5 Data in Flash used by the User Routines

Table 234 shows the data in Flash which is used by the user routines in PSC1Mx.

Table 234 Basic Flash data for SSW and user SW in PSC1Mx

Address	Length	Function	Target location
Start-up mode selection:			
$1000'0E00_H$	2 B	Boot Mode Index (BMI)	---
$1000'0E10_H$	2 B	Inverse BMI	---
Temperature-sensor related data:			
$1000'0F20_H$	16 B	Temperature-sensor constant data 1	---
$1000'0F30_H$	1 B	ANA_TSE_T1	---
$1000'0F31_H$	1 B	ANA_TSE_T2	---
$1000'0F34_H$	161 B	Temperature-sensor constant data 2	---

28 Debug System (DBG)

The debug system is an extension to the regular processor architecture. The PSC1Mx Series Microcontrollers provide a complete hardware debug solution, with hardware breakpoint and watchpoint options. This allows high system visibility of the processor, memory and available peripherals. The debug functions are implemented with a standard Arm® Cortex® M0 configuration. Debug functions are integrated into the Arm® Cortex®-M0 processor architecture. The debug system supports serial wire debug and single pin debug interfacing.

References

[9] Cortex®-M0 Technical Reference Manual
[10] Cortex®-M0 Integration and Implementation Manual
[11] Cortex®-M0 User Guide
[12] Arm®v6-M Architecture Reference Manual
[13] Arm® Debug Interface V5
[14] CoreSight Components Technical Reference Manual
[15] CoreSight DAP-Lite

28.1 Overview

The basic debug functionality of the Cortex®-M0 debug system is limited to invasive debug and includes processor halt, single step, processor core register access, Reset and HardFault Vector Catch. Besides the hardware debug components unlimited software breakpoints are available. The Debugger can connect to the debug system via the Serial Wire Debug (SWD) or Single Pin Debug (SPD) interface port and uses CoreSight infrastructure and the regular access flow. This permits debugger to identify the processor and its debug capabilities. The debug system allows a full system memory access and access to zero-wait state system slaves via the internal debug access port (DAP) connected to the system bus matrix. This access is non-intrusive and a debugger can access the devices, including memory when the processor is halted or running. Core register full access is available, if the processor is halted. The System Control Space (SCS) provides debug through registers available. The Data Watchpoint Unit (DWT) provides two watchpoint register sets, each implement data address and PC based watchpoint functionality including comparator address masking. The processor BreakPoint Unit (BPU) provides four PC based breakpoint register. The system is accessible by the tool through DAP. DAP permits access to debug resources when the processor is running, halted, or held in reset. A processor enters Debug state if it is configured to halt on a debug event, and a debug event occurs. The supported debug infrastructure can be identified by checking the ROM table.

28.1.1 Features

The accurate Debug and Trace System provides the following functionality:

- Serial Wire Debug Port (SWD) provides Serial Wire Debug, which allows to debug via 2 pins.
- Single Pin Debug (SPD) provides a debug capability via 1 pin.
- Processor halt, single-step, processor core register access
- Reset and HardFault Vector Catch.
- Software breakpoints
- Full system memory access
- 4 Hardware breakpoints supported
- 2 watchpoints supported

***Note**: Please refer to Arm® v6-M Architecture Reference Manual for more detailed informations on the debug functionality.*

28.1.2 Block Diagram

The Debug system block diagram is shown in Figure 458.

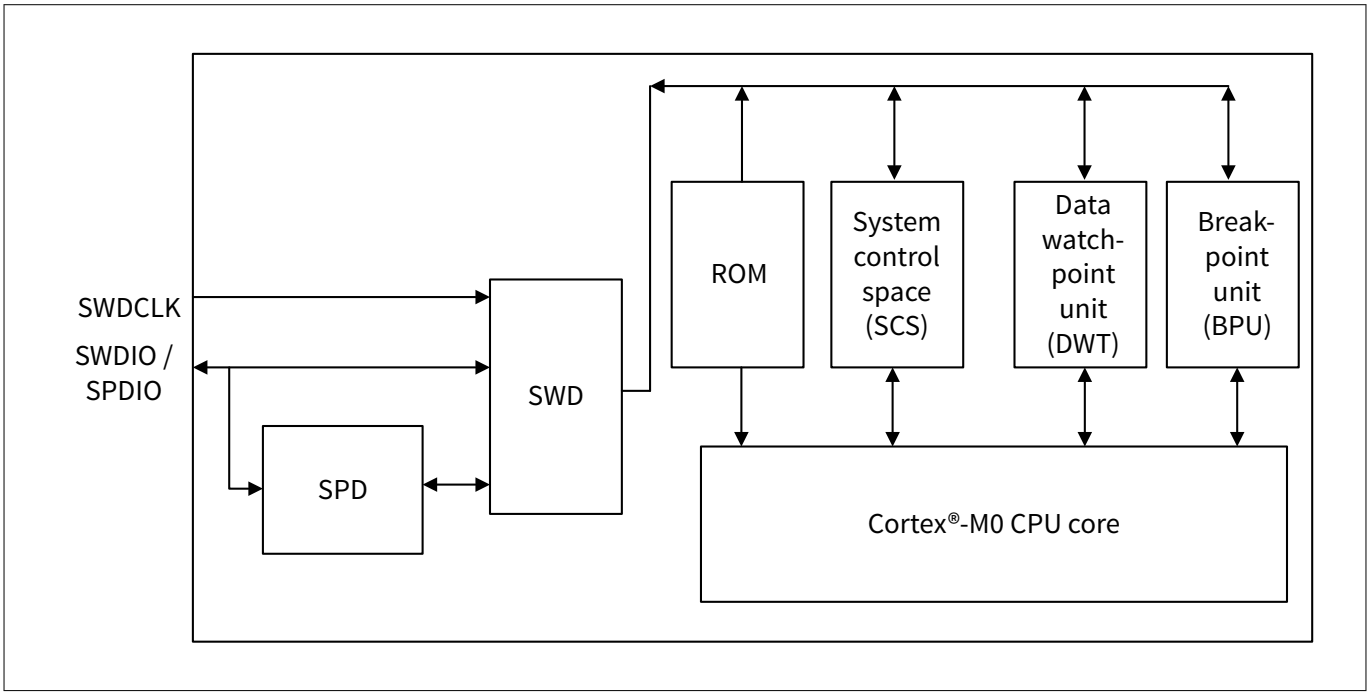


Figure 458 Debug and Trace System block diagram

28.2 Debug System Operation

The Debug System provides general debug options. Debug options are based on break points and CPU halt. Debug resources available are Data Watchpoint and Trace, Breakpoint unit, ROM table and the SCS system control block and debug control block. Debug capabilities can be accessed by a debug tool via Serial Wire Debug interface (SWD) or Single Pin Debug (SPD). The selection of the access protocol (SWD or SPD) is done by BMI mode setting.

28.2.1 System Control Space (SCS)

The SCS is the area where the debugger can have direct access to the memory mapped register debug register. The debug resources together with the debug register in the SCS are accessible through the DAP interface. Accessible resources are for example system control and ID registers including system control block. Additionally the system timer and the interrupt controller (NVIC) can be accessed via the SCS.

28.2.2 Data Watchpoint and Trace (DWT)

The DWT unit provides an external Program Counter (PC) sampling capability based on PC sample register and comparators, that support watchpoints for address matching and PC watchpoints for instruction address matching. The PC sampling feature and watchpoint support operate independently of each other and a watchpoint event is asynchronous to the instruction that caused it. The PSC1Mx supports two watchpoints, each supporting compare, mask and function registers. Watchpoint events result in a processor halt and enters the processor system into debug state. Data address matching results in a creation of a watchpoint event.

Instruction address matching in a creation of a PC watchpoint event. The DWT register are accessible through the DAP interface.

A DWT program counter sample register permits a debugger to periodically sample the PC without halting the processor and allows coarse grained profiling. The PC sampling feature and the watchpoint support operate independently of each other. The register is defined so that a debugger can access it without changing the behavior of any code currently executing on the device.

The recommended mechanism for generating a breakpoint on a single instruction address is to use the BPU. The DWT based mechanism must be used to generate a PC matching event on a range of addresses. The debug return address value for a watchpoint event must be that of an instruction to be executed after the instruction responsible for generating the watchpoint.

28.2.3 Break Point Unit (BPU)

The Breakpoint (BKPT) instruction provides software breakpoints. It can cause a running system to halt depending on the debug configuration. A BKPT is a synchronous debug event, caused by execution of a BKPT instruction or by a match in the BPU. A BKPT causes the processor to enter debug state. The Debug tool can use this situation to investigate the system state when the instruction at a particular address is reached. The BPU provides support for breakpoint functionality on instruction fetches, based on instruction address comparator. The M0 provides four breakpoint comparator registers. Each breakpoint comparator register includes its own enable bit. The comparators match instruction fetches from the Code memory region and operate only on instruction read accesses. The comparators do not match data read or data write access. Address matching is available on the upper half word, lower half word or both half words. Potential reasons for a debug event are a debug halt, a BKPT, a DWT trap and Vector CATCH.

28.2.4 ROM Table

To identify the Cortex®-M0 processor with the respective Debug System components the debugger locates and identifies the ROM table. ROM table Identification values are predefined and contain pointers to the SCS, BPU and DWT. Each of the debug modules requires it's own ROM table and indicate whether the block is present. The ROM table can be accessed through the DAP interface.

28.2.5 Debug tool interface access - SWD

The tool access based on SWD must use a connection sequence, to ensure that hot-plugging the serial connection does not result in unintentional transfer. The connection sequence ensures that the SWD is synchronized correctly to the header. The sequence consists of a sequence of 50 clock cycles data = 1s. This connection sequences is also used as a line reset sequence. The protocol requires that any run of 50 consecutive 1s on the data input is detected as line reset, regardless of the state of the protocol. After the line reset the debugger reads IDCODE register, which gives confirmation that correct packet frame alignment is has been achieved.

28.2.5.1 SWD based transfers

The SWD interface requires to continue the clock for a number of cycles after the data phase of any data transfer. This ensures the transfer is completely clocked through the SWD. The transfer completion can be achieved by a immediate start of a new SWD operation, continuous clocking of SWD interface until the host starts a new SWD operation or after the data parity bit the clock is continued for at least 8 more clock rising edges, before stopping the clock.

Each sequence of operation on the serial wire consists of two or three phases and is defined from debugger point of view. The package request and acknowledge response phases are always there. The data transfer phase is only present when either data read or data write request is followed by a valid acknowledge response or the Overrun Detect flag is set.

The SWD protocol applies a simple parity check to all packet request and data transfer phases. On a packet request the parity check is made over the four bit header. On a data transfer the parity check is made over the complete 32 data bits. The parity is added directly after the packet request and after the data transfers. The parity bits are not included in the parity calculation.

Serial Wire Debug protocol operation

The SWD protocol supports the following operations:

- **Write operation** including OK response (3-Phases: 8-bit write packet request, 3-bit OK ACK, 33-bit data write including1-bit parity at the end)
- **Read operation** including OK response (3-Phases: 8-bit read packet request, 3-bit OK ACK, 33-bit data read including a 1-bit parity at the end).
- **WAIT response** to read or write operation request (2 Phases: 8-bit read or write packet request, 3-bit WAIT ACK response)
- **FAULT response** Read or Write operation (2-Phases: 8-bit read or write packet request, 3bit FAULT ACK response)
- **Protocol error** sequence occurs when target failed to return a ACK response (1 Phase: 8-bit Packet request - line not driven by target for 32 bit cycles)

***Note**: A single-cycle turnaround period between the operation phases is required, for the transfer direction change only. If there is no transfer direction change, no turnaround period is introduced. If the protocol ends with a data transfer towards the debugger a turnaround period is introduced. After the single-cycle turnaround time the protocol must be proceeded.*

28.2.5.2 SWD based errors

On the SWD interface protocol errors can occur. These errors might be detected by the parity checks on the data. A message header error can be detected by a parity error. A debugger not receiving a response or a port does not respond to the message, the debugger must back off. During the back off mechanism the debugger does read the IDCODE register and ensures the Debug Port is responsive and then retries the original access.

Errors can be returned by the DAP itself or might come from a debug resource. When an error occurs the error bit remains sticky until cleared. A debugger must check the error status after performing a series of transaction to be aware of an error.

Error conditions within the SWD interface are recorded using sticky flags. Potential errors are read and write errors, overrun detection, protocol errors. When the sticky bit is set to 1 it remains set until it is explicitly cleared to 0. When an error is flagged the current transaction is completed and any additional access port access transaction is discarded until the sticky flag is cleared to 0. A debugger must check periodically the Control/ Status register after performing a series of transactions to detect the error.

A read and write error might occur within the DAP or come from the resource being accessed. Additionally a read or write error might be generated if the debugger makes an access port transaction request while the debugger power domain is powered down.

An overrun error might be detected on a sequence of transactions. Therefore the debugger must check after each sequence. The Overrun detection mode can be left by clearing the STICKYORUN and ORUNDETECT.

On the Serial Wire Interface protocol errors can occur, for example because of wire-level errors. These errors might be detected by the parity checks of data. If the SWD interface detects a parity error in a message header of the debug message the error is reflected. If the interface does not receive a response to a message, the debugger must back off. It must then requires a read of the IDCODE register, to ensure the debug port is responsive, before retrying the original access, again. If the SWD detects a parity error in the data phase of a write transaction, it sets the sticky write data error flag (WDATAERR) in the control and status register. Subsequent accesses from the debugger, other than IDCODE, CTRL/STAT or ABORT, result in a FAULT response.

28.2.6 Debug tool interface access - Single Pin Debug (SPD)

The SPD protocol based tool interface access allows to debug the system using a single pin only. The bit frequency is 2 MHz and allows an effective SWD telegram of 1.4 Mbits/s. The protocol is very robust against clock deviations between tool and device.

The SPD protocol encodes the SWD protocol bits with the distance between the SPD signal edges. A SWD value of '0' is encoded with a short distance of 0.5 µs, a value of '1' with a distance of 1 µs. This encoding is used in both transfer directions. Initially the SPD signal level always start with a positive SPD signal pulse with 1 µs width, encoded SPD start bit, which is not part of the SWD protocol. The transmission of the rest of the telegram will be independent of the edge direction. As the protocol starts with a logic one start bit detection it is recommended to finish the protocol with a logic 0 signaling level. If the SPD signal level is high after the last bit of the telegram, that tool to add at '0' bit with a negative edge after 0.5 µs. This can be achieved by the tool transferring an odd number of bits.

SWD has a clean request response protocol, where the tool is always the requestor and the device executes and sends a response. So the SWD module will change the direction as defined by the SWD protocol operations (SWD based transfers).

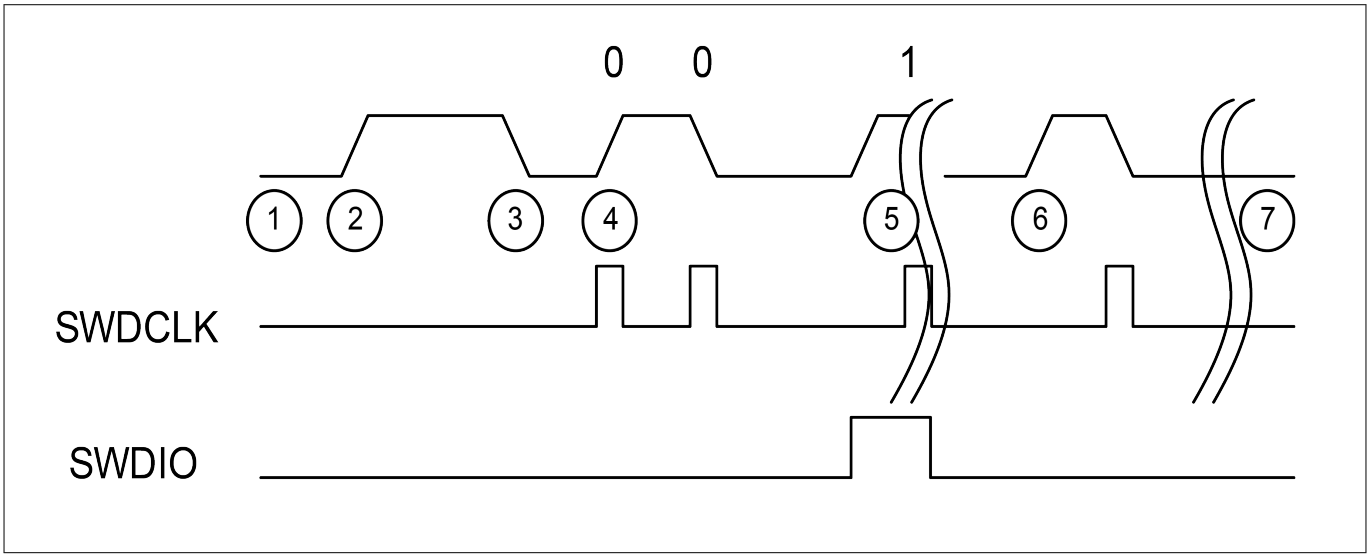


Figure 459 SPD Encoding Example

As a basic example a simple transfer is described in Figure before. Initially (1) the SPD module is in IDLE state. During IDLE state the SPD module does not generate a clock to the SWD module. At point (2) the SPD detects a rising edge and moves to a receive state. The first rising edge of the protocol is a start bit for the SPD module only and is not transferred to the SWD module. The following bits are the SWD header information and transferred to the SWD module. At point (5) a logic 1 shown and at (6) a logic zero. At the time the protocol direction is changed (which means for the implementation to go through IDLE state) the clock to SWD is switched off as shown in state (7). The SPD module requires to end on a Low signal value therefore it is required to have an odd number of bits moved into the interface.

Protocol transfer requirements between SWD and SPD module

The data transfer to and from SWD protocol is single bit based on the SWD clock. The SWD clock is generated by the SPD module and directly derived from the transferred bit. In RECEIVE direction the SPD module transfers the data to the SWD module bit wise, with one clock cycle per bit received.

The SPD module switches off SWD clock when going through module IDLE.

The Debugger adds a SPD start bit in front of the SWD telegram when it writes data to the device (form SPD point of view, RECEIVE). The SPD RECEIVE start bit is not transferred to the SWD module. The RECEIVE start bit is in front of the original SWD protocol on every new RECEIVE start, which is in front of the SWD header and again in front of debugger write data. Additionally the SPD module requires in RECEIVE direction a protocol to

end with a logic zero signal level, which is achieved by an odd number of bits transferred by the tool. Based on the SWD protocol this is already achieved in the header phase by adding the described SPD start bit. During acknowledge phase as this phase does have 3 bits only. During the protocol data phase the Debugger has to add one more zero bit. (SPD Start bit, 32-bit data, parity bit, SPD odd bit as logic Zero).

As the SWD protocol requires in RECEIVE direction at least 8 clocks at the end of the data transferred to finish the protocol operation the Debugger has to add 9 logic zeros to the protocol (8 for this requirement and 1 to have an odd number).

SPD protocol based tool Interaction

All SPD communication is initiated by the tool. The tool will send a SWD header encoded to a SPD telegram (header) and then switches the SPD signal direction to input and wait for the acknowledge from the device.

SPD Receive:

Based on a low level the tool starts the protocol with SPD start bit. The start bit is indicated with a rising edge and a signal duration of 1µs. The following bits are based on the SWD protocol sequence.

The start sequence: SPD start bit + SWD HEADER + 0 Bit + 0 Bit (the two additional 0 bits at the end are required to achieve a low level at the signaling line.).

After receiving the header, the SWD will output his acknowledge response. Thus, SPD will change from RECEIVE to IDLE to SEND and ACKto IDLE again.

Leaving IDLE to RECEIVE state to transfer write data the start bit from tool is also required.

The SPD start bit must be a logic one with the defined time duration of a logic one. Is the signal duration longer it turns into a timeout situation. Is the signal duration shorter the start bit is not recognized.

The debug tool not sending data continuously in SPD RECEIVE direction, generates a Timeout in SPD module, which brings it to IDLE mode again. Starting from IDLE state the tool is required to provide SPD start bit for a new communication. A timeout is recognized after 1.4 µs.

A permanent write (SPD RECEIVE) allows to proceed the protocol without SPD start bit. In this case the SPD interface remains in RECEIVE state and only the SWD receive protocol has to be performed. In this case the SWD header has to start with a logic 1 as defined by the SWD protocol.

SPD SEND:

The tool is required to end a RECEIVE at low level. Based on the low level the SEND start is indicated by a rising edge. SEND has to end with a signal low level, too.

IDLE:

From RECEIVE to IDLE - driven by SPD or by timeout.

From SEND to IDLE - driven by SWD module caused by direction change.

Time to switch from RECEIVE to SEND is between 375 and 500 ns and always goes through IDLE phase. The tool must change from write (SPD RECEIVE) to read (SPD SEND) direction within 375 ns. SEND always starts with a rising edge.

28.2.7 Debug accesses and Flash protection

The PSC1Mx Flash implements read protection for sector 0 only. All other sectors can be read. Additionally a user configuration erase and write protection is available, which allows to protect Flash content from unintended writes or erase. 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. The Boot Mode Index (BMI) determines if the debug mode can be entered and the debug interface enabled at the end of the SSW.

Before the BMI is configured to user productive mode a change of BMI to arbitrary value is supported, afterwards BMI can only be restored to its default value, which does not enable debug access.

Besides the BMI configuration the Debug system supports a protection mechanism which prevents a debug access to the system address area during the time startup software (SSW) is running. Fimware controls the debug access, based on register setting.

Software based Breakpoints are supported, but prevented during SSW.

28.2.8 Halt after reset

There are two possibilities to perform a halt after reset. The first possibility is Infineon specific an allows to perform a CPU halt after “power on”/”master” reset (HAR) at the very first instruction of the Application code. The HAR activation is based on BMI settings and SSW executing endless loop, allowing a debugger to take control over the device. A defined configuration sequence driven by the connected debug tool is required in this case. The second possibility is the regular Arm® flow halt after reset, which is based on breakpoints and a system reset. This halt after reset is also named “Warm Reset”. In both cases the Arm® system can not be accessed for security reasons from the debug port during the time SSW is running, as the physical pins are not available during that time. Based on the BMI setting the debug capability can be enabled at the end of the SSW by firmware. The default BMI configuration does not enable debug access.

28.2.8.1 HAR

The BMI can be configured to allow a HAR (UMHAR) at the end of the SSW, which always requires a power on reset to be executed. A new BMI configuration to BMI.UMHAR (User Mode with debug enabled and HAR) requires to perform a master reset to boot in UMHAR mode.

At a HAR (Cold Reset situation), the system comes from a PORST (or Master Reset). To achieve a HAR, the tool has to register (CDBGPWRUPREQ) and enable (DHSCR.C_DEBUGEN) the debug system. Additionally it has to halt (DHCSR.C_HALT) the CPU and manipulate the PC (Program Counter) to point to the start address of the user code. The debugger registration and debug system enable is possible at the end of the firmware, where firmware is waiting for a tool registration. During SSW execution the debugger as no access capability and cannot set the enabling bits CDBGPWRUPREQ, C_DEBUGEN or halt the CPU. The address of the user code start address can be read from address 0x10001004 in user Flash. It is recommended to check the Boot Mode Index BMI.HWCFG bit for UMHAR before manipulating the PC.

Figure 460 shows the software flow based on the modules participating to the HAR.

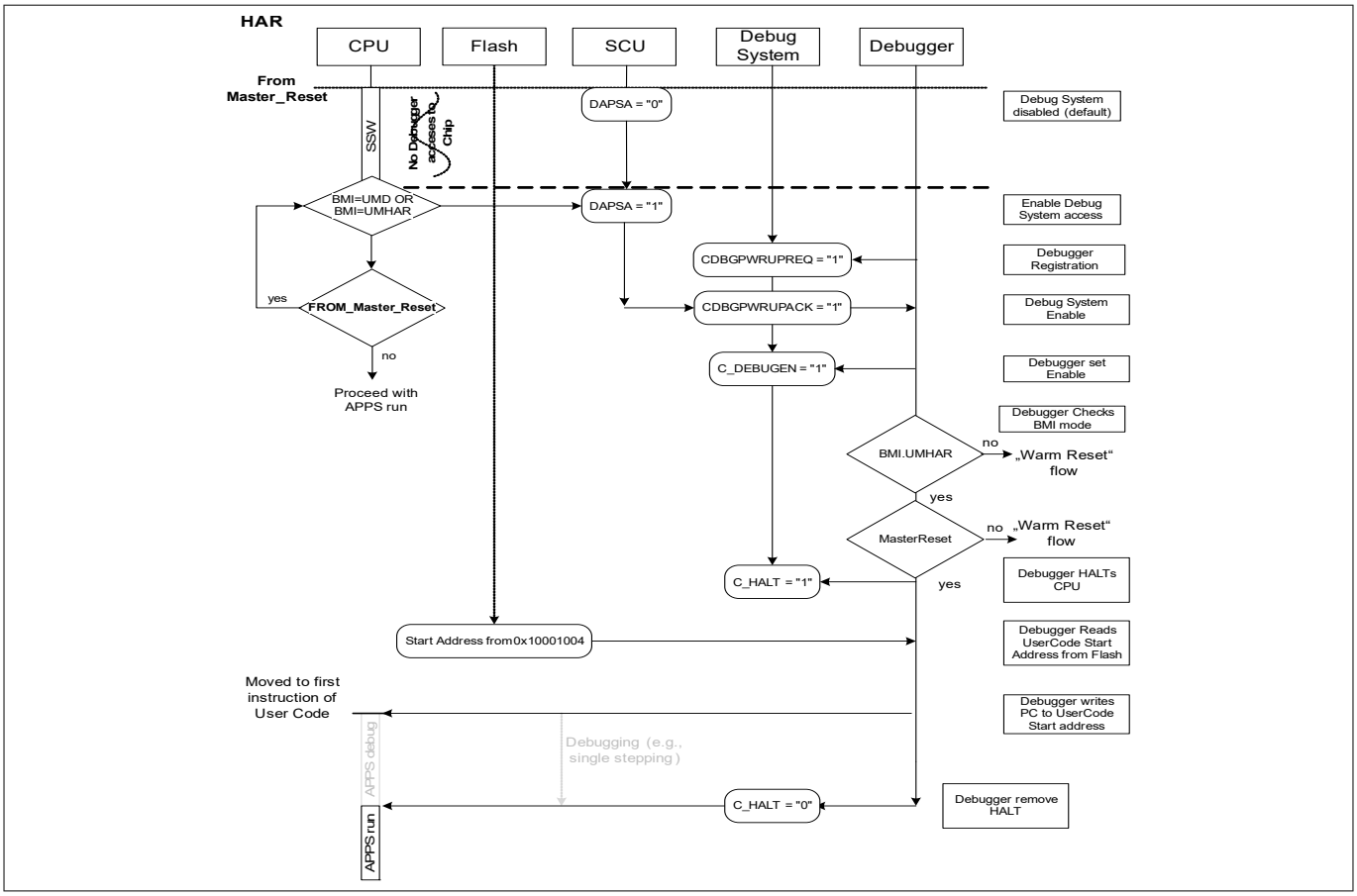


Figure 460 HAR - Halt After Reset Flow

28.2.8.2 Warm Reset

A halt after system reset (Warm Reset) can be achieved by programming a break point at the first instruction of the application code. Additionally the CDBGPWRUPREQ (tool registration) and the C_DEBUGEN has to be set by the debugger. After a system reset, the HAR situation is not considered, as the reset is not coming from PORST (Master_Reset).

***Note**: The CDBGPWRUPREQ and C_DEBUGEN does not have to be set after a system reset, if they have already been set before and the debugger remains registered. The bits are not affected by the system reset.*

A tool hot plug situation allows to debug the system by enabling the CDBGPWRUPREQ and the C_DEBUGEN register. It is recommended to check the Boot Mode Index BMI.HWCFG. The Warm Reset flow can be done in both BMI modes, UMD (User mode with debug enabled) and UMHAR. In both cases the “Warm Reset” flow is based on breakpoint setting and system reset afterwards.

In general after a tool hot plug break points can be set or the CPU can directly be HALTED and stops at the actual program stage. To stop at the very first instruction of the user code the tool has to fire a system reset after setting the breakpoint there.

Figure 461 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.

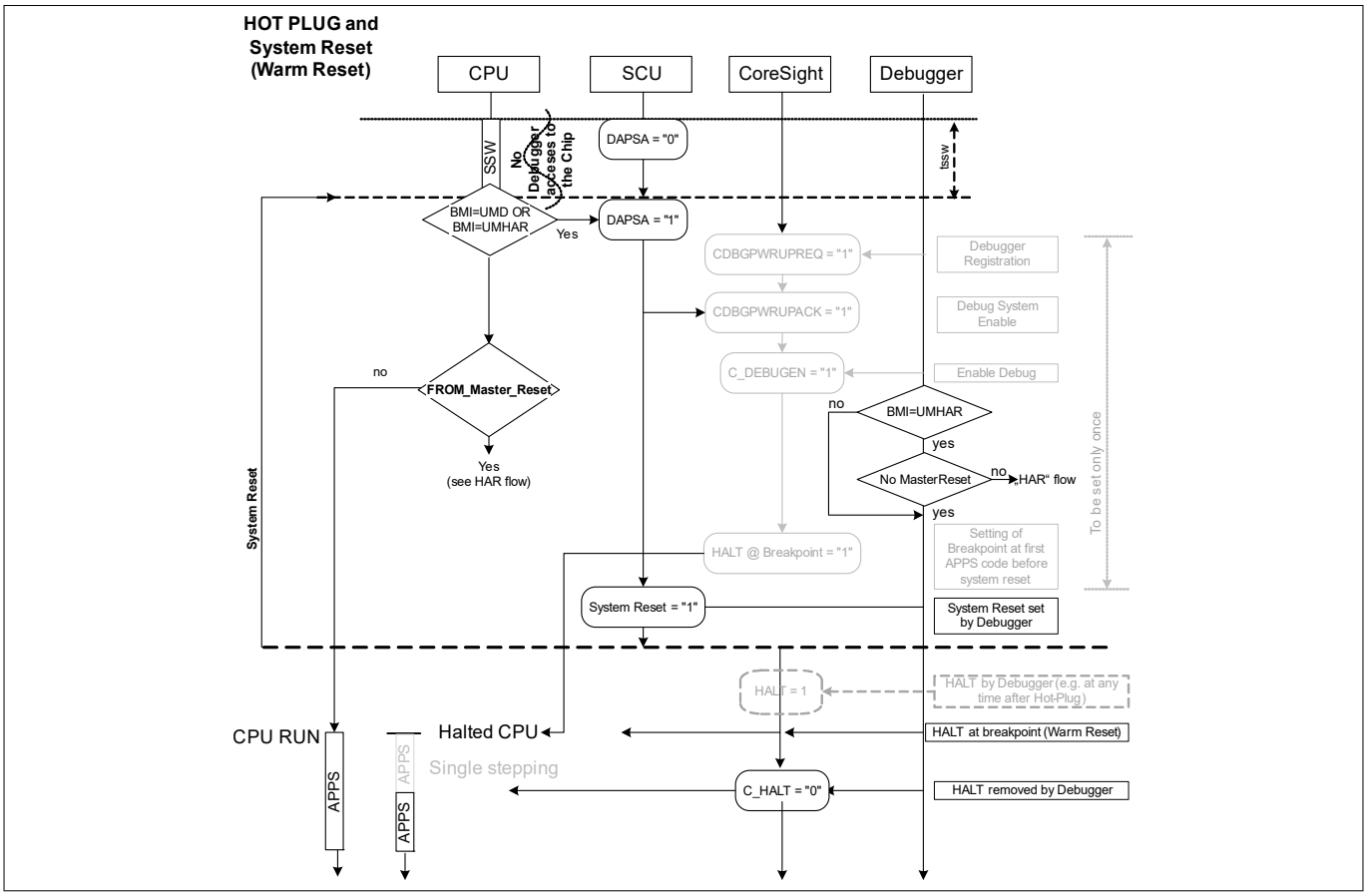


Figure 461 HOT PLUG or Warm Reset Flow

28.2.9 Halting Debug and Peripheral Suspend

The PSC1Mx device supports a suspend capability for peripherals, if the program execution of the CPU is stopped by the debugger, for example, with a breakpoint, or with the C_HALT. This allows to debug critical states of the whole microcontroller. Whether the suspend function is supported or not has to be configured locally at the peripheral.

In some cases it is important to keep certain peripherals running, for example, a PWM or a CAN node, to avoid system errors or even critical damage to the application. Because of this, the peripheral allows to configure how it behaves, when the CPU enters halt debug mode. Per default a peripheral is not sensitive on a suspend request. Sensitivity can be configured based on the system use case.

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. Table 235 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 235 Peripheral Suspend support

Peripheral	supported	default mode	Hard Suspend	Soft Suspend
RTC	yes	not active	yes	no
USIC	yes	not active	no	yes
CCU4	yes	not active	yes	yes
CCU8	yes	not active	yes	yes
POSIF	yes	not active	yes	yes
BCCU	yes	not active	yes	no
WDT	yes	active	yes	no
ADC	yes	not active	yes	yes
VADC	yes	not active	yes	yes
MATH	yes	not active	yes	yes
PRNG	no	- - -	- - -	- - -
MultiCAN+	no	- - -	- - -	- - -

***Note**: Enable debug suspend function in the user initialization code after every reset. In addition the user has to consider debug suspend register at peripherals can only be configured after the clock of the module is enabled via CGATCLR0 register.*

Important tool provider note:

The peripheral suspend logic is connected to the system reset. A system reset activation results in a peripheral suspend configuration loss. Therefore the suspend configuration at the peripherals "outlasting" a system reset, requires the tool to reconfigure the suspend configuration after system reset. This is achieved by shadowing the user peripheral suspend configuration in the tool and set a HW breakpoint at the first instruction of use code, if

the suspend function is activated. After the CPU halts at that breakpoint the tool has to reconfigure the suspend configuration at the peripheral, as it has been configured before the system reset.

***Note**: Suspend activation results in loosing one HW breakpoint, as it is used by the tool to handle the suspend reconfiguration after system reset.*

A debug tool should offer the peripheral suspend reconfiguration after system reset with an option to proceed Usercode without an user interaction requirement after tool reconfiguration or remain stopped at first line of user code and wait for user action.

(The user configures the desired suspend behavior only once and the tool stores the configuration to have it present for a automatic reconfiguration after a system reset.)

28.2.10 Debug System based processor wake-up

The debugger can wake-up a processor in sleep mode (sleep and deep sleep mode). The wake-up is based on a new pending interrupt event from the debug system, which is a HALT. The interrupt event does also wake-up the processor, if the interrupt is disabled or has insufficient priority to cause exception entry. The interrupt wake-up mode is derived by the NVIC (Programmable Multiple Priority Interrupt System) available in the processor system.The debugger is able to register, enable the Debug System and HALT the CPU also in sleep and deep sleep mode.

***Note**: The wake-up from deep-sleep mode via debugger halt is only supported using SWD interface. A wake-up from deep-sleep based on SPD interface is not supported.*

28.2.11 Debug Access Server (DAS)

The DAS API provides an abstraction of the physical device interface for tool access. The key paradigm of DAS is to read or write data in one or several address spaces of the target device.

DAS Features

- Standard interface for all types of tools
- Efficient and robust methods for data transfer
- Several independent tools can share the same physical interface
- Infineon's DAS miniWiggler support SWD and SPD.

***Note**: DAS is not PSC1Mx specific. It can be used for all Infineon 8-, 16, and 32-bit microcontrollers with DAP, SWD, SPD or JTAG interface. For more information refer to www.infineon.com/DAS.*

28.2.12 Debug Signals

PSC1Mx MC product family provides debug capability using Arm® M0 Debug port SWD or the Infineon proprietary SPD. The SWD Port has 2 interface signals (Clock + Bidirectional data). The SPD port has one interface signal (SPDIO - bidirectional data) with an overlay of SWD interface SWDIO pin.

Table 236 SWD toplevel IO signal

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®-M0 debug system.

The HALTED and EDBGRQ signals can be used to halt the CPU based on an external event. This allows to halt an external hardware synchronously with the CPU. The halt can be recognized by the external hardware based on the HALTED output. This can be used for example to synchronize with an logic analyzer and request a breakpoint or in a multi CPU scenario.

28.2.12.1 Internal pull-up and pull-down on SWD/SPD pins

It is a requirement to ensure none floating SWD/SPD 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 SWD/SPD input pins are provided.

- SWDIO/SPD: Internal pull-up
- SWCLK: Internal pull-down

28.2.13 Reset

The debug system register bits are reset by Power-on reset. Other reset in the system do not have an effect on the debug register, if the tool is registered.

28.3 Debug System Power Save Operation

The Debug System is in the “always-on” power domain, which allows to connect the debugger to the device. The Debug System does not support any special power mode or power save operation. The power supply of the Debug System is directly connected to the main chip part including the system components.

28.4 Service Request Generation

The debugger can set DHCSR.C_MASKINTS to 1 to prevent PendSV, SysTick and external configured interrupts from occurring.

28.5 Debug behavior

The Debug System is based on events. Whereas events are an entry to debug state, if halting debug is enabled. Debug events available are:

- Internal halt request
- Breakpoint
- Watchpoint
- Vector catch
- External debug request based on

The following events are synchronous debug events:

- Breakpoint debug events, caused by execution of a BKPT instruction
- Breakpoint debug events, caused by execution of a match in the BPU
- Vector catch debug events
- Step debug events (DHCSR.C_STEP)

The following events are asynchronous debug events:

- Watchpoint debug events, including PC match watchpoints
- Halt request debug event (DHCSR.C_HALT)
- External signal based debug request event (EDBGRQ signal)

Asynchronous debug events do have a lower prioritization then synchronous generated events. An instruction can generate a number of synchronous debug events. It also can generate a number of asynchronous exceptions.

The Debug Fault Status register (DFSR) contains a status bit for each debug event. The bits are write-one-to-clear. These bits are set to 1 when a debug event causes the processor to halt or generate an exception. The bits are also updated if the event is ignored.

Software must write 0xA05F to the DHCSR.DBGKEQ register in order to be able to have debug support available form CPU side.

28.6 Power, Reset and Clock

The requirements for power, reset and clock are derived from Arm®.

28.6.1 Power management

There is no power management implemented for the debug system. Nevertheless the tool does has to follow the regular tool flow by setting the CDBGPWRUPREQ in order to register the tool.

28.6.2 Debug System reset

The SWD register are in the power on reset (PORST) domain.

The Debug logic is reset by system reset, if no tool is connected to chip and therefore not registered, which is indicated by **CDBGPWRUPREQ** not set.

Processor reset

A processor or warm reset (SYSRESETn) initializes the majority of the processor, excluding debug logic, BKPT unit, DWT unit and SCS.

PowerOn reset

PORESETn reset initializes the SWD access port, the AHB-AP logic and all other debug system related functions.

Normal operation

During normal operation the resets PORESETn and SYSRESETn are deasserted.

Processor reset affects Debug System

A System reset also affects the Debug System, if a tool is not registered. The tool registration is reflected by an activation of the register bit CDBGPWRUPACK, which activates by a debugger enabling the CDBGPWRUPREQ.

28.6.3 Debug System Clocks

The Debug system clock is called DCLK and is derived directly from SCLK (free running clock). DCLK must always be driven while a debugger is connected.

28.7 Initialization and System Dependencies

28.7.1 ID Code

Available ID Code is used by a debug tool to identify the available debug components during tool setup.

Table 237 Arm® CoreSight™ Component ID code

ID	Value	Description
SW_DP	0BB1 1477$_H$	The Arm® SW-DP ID

28.7.2 ROM Table

To identify Infineon as manufacturer and PSC1Mx as device, the ROM table has to be read out. The format of the ROM table is illustrated in Table 238.

Table 238 Peripheral ID Values of PSC1Mx ROM Table

Name	Offset	Value[1)]	Bits	Reference	Infineon JTAG ID Code
PID0	FE0$_H$	XXXXXXXX$_B$	[7:0]	Part Number [7:0]	DBGROMID [19:12]
PID1	FE4$_H$	0001XXXX$_B$	[7:4] [3:0]	JEP106 ID code [3:0] Part Number [11:8]	DBGROMID [4:1] DBGROMID [23:20]
PID2	FE8$_H$	XXXX1100$_B$	[7:4] [3] [2:0]	Revision JEDEC assigned ID fields JEP106 ID code [6:4]	DBGROMID [31:28] ‘1’ DBGROMID [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 DBGROMID [11:8]

1) For “X” values please refer to DBGROMID in the SCU chapter.

The ROM table values representing Infineon Part Number, JEP106 and Revision number can be checked in the SCU chapter.

28.8 Debug System Registers

Registers Overview

The absolute register address is calculated by adding:

The DWT, BPU, ROM table, DCB and debug register in the SCS are accessible memory mapped by the debugger and also from the CPU.

Table 239 Registers Address Space

Module	Base Address	End Address	Note
DWT	E000 1000$_H$	E000 1FFF$_H$	Data Watchpoint
BP	E000 2000$_H$	E000 2FFF$_H$	Breakpoint Unit
SCS	E000 E000$_H$	E000 EEFF$_H$	SCS (here DBG CTRL)
ROM	E00F F000$_H$	E00F FFFF$_H$	ROM table area

Table 240 Register Overview

Register Name	Description	Offset Addr.	Access Mode	
			Read	Write
SCS_DFSR	Debug Fault Status Register	D30$_H$		
SCS_DHCSR [Read Mode]	Debug Halting Control and Status Register	DF0$_H$		
SCS_DHCSR [Write Mode]	Debug Halting Control and Status Register	DF0$_H$		
SCS_DCRSR	Debug Core Register Selector Register	DF4$_H$		
SCS_DCRDR	Debug Core Register Data Register	DF8$_H$		
SCS_DEMCR	Debug Exception and Monitor Control Register	DFC$_H$		
DWT_CTRL	DWT Control Register	000$_H$		
DWT_PCSR	DWT program Counter Sample Register	01C$_H$		
DWT_COMP0	DWT Comparator register	020$_H$		
DWT_COMP1	DWT Comparator register	030$_H$		
DWT_MASK0	DWT MASK register	024$_H$		
DWT_MASK1	DWT MASK register	034$_H$		
DWT_FUNCTION0	DWT Comparator Function register	028$_H$		
DWT_FUNCTION1	DWT Comparator Function register	038$_H$		
BP_CTRL	Breakpoint Control register	000$_H$		
BP_COMP0	Breakpoint Comparator register	008$_H$		
BP_COMP1	Breakpoint Comparator register	00C$_H$		
BP_COMP2	Breakpoint Comparator register	010$_H$		
BP_COMP3	Breakpoint Comparator register	014$_H$		

28.8.1 DFSR - Debug Fault Status Register

28.8.1.1 SCS_DFSR

The DFSR registers provides the top level reason why a debug event has occurred. Writing 1 to a register bit clears the bit to 0. A read of the HALTED bit by an instruction executed by stepping returns an UNKNOWN value.

SCS_DFSR **Address:** $D30_H$

Debug Fault Status Register **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											**EXTERNAL**	**VCATCH**	**DWTTRAP**	**BKPT**	**HALTED**
r											rw	rw	rw	rw	rw

Field	Bits	Type	Description
HALTED	0	rw	**HALTED** Indicates a debug event generated by a C_HALT or C_STEP request, triggered by a write to the DHCSR. 0_B no active halt request debug event. 1_B halt request debug event active.
BKPT	1	rw	**BKPT** Indicates a debug event generated by BKPT instruction execution or a breakpoint match in the BPU. 0_B no breakpoint debug event. 1_B at least one breakpoint debug event.
DWTTRAP	2	rw	**DWTTRAP** Indicates a debug event generated by the DWT. 0_B no debug events generated by the DWT. 1_B at least one debug event generated by the DWT.
VCATCH	3	rw	**VCATCH** Indicates whether a vector catch debug event was generated. 0_B no vector catch debug event generated. 1_B vector catch debug event generated. ***Note**: The corresponding FSR shows the primary cause of the exception.*

(table continues...)

(continued)

Field	Bits	Type	Description
EXTERNAL	4	rw	**EXTERNAL** Indicates an asynchronous debug event generated because of EDBGRQ being asserted. 0_B no EDBGRQ debug event. 1_B EDBGRQ debug event.
0	31:5	r	**Reserved**

28.8.2 DHCSR - Debug Halting Control and Status Register

28.8.2.1 SCS_DHCSR [Read Mode]

Controls halting debug. When C_DEBUGEN is set to 1, C_STEP and C_MASKINTS must not be modified when the processor is running (S_HALT is 0 when the processor is running). When C_DEBUGEN is set to 0, the processor ignores the values of all other bits in this register.

A separate register is represented below for read and write as the function changes at some bits access based.

SCS_DHCSR — **Address:** $DF0_H$

Debug Halting Control and Status Register [Read Mode] — **Reset value:** $0000\ 0000_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0						**S_RESET_ST**	**S_RETIRE_ST**	**0**				**S_LOCKKUP**	**S_SLEEP**	**S_HALT**	**S_REGRDY**
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												**C_MASKINTS**	**C_STEP**	**C_HALT**	**C_DEBUGEN**
r												r	r	r	r

Field	Bits	Type	Description
C_DEBUGEN	0	r	**Halting debug enable bit** 0_B Halting debug disabled. 1_B Halting debug enabled. ***Note**: If a debugger writes to DHCSR to change the value of this bit from 0 to 1, it must also write 0 to the C_MASKINTS bit, otherwise behavior is UNPREDICTABLE. This bit can only be written from the DAP Access to the DHCSR from Software running on the processor it cannot be set. This bit is 0 after Power-on reset.*

(table continues...)

(continued)

Field	Bits	Type	Description
C_HALT	1	r	**Processor halt bit.** The effects of writes to this bit are: 0_B Request a halted processor to run. 1_B Request a running processor to halt. ***Note**: This bit is unknown after power-on reset*
C_STEP	2	r	**Processor step bit.** The effects of writes to this bit are: 0_B Single-stepping disabled. 1_B Single-stepping enabled. ***Note**: This bit is unknown after power-on reset*
C_MASKINTS	3	r	**Mask PEDSV, SysTick and external configurable interrupts.** The effects of writes to this bit are: 0_B Do not mask 1_B Mask PendSV, SysTick and external configurable interrupts. The effect of any attempt to change the value of this bit is UNPREDICTABLE unless both: • before the write to DHCSR, the value of the C_HALT bit is 1 • the write to the DHCSR that changes the C_MASKINTS bit also writes 1 to the C_HALT bit. This means that a single write to DHCSR cannot set the C_HALT to 0 and change the value of the C_MASKINTS bit. When DHCSR.C_DEBUGEN is set to 0, the value of this bit is UNKNOWN. This bit is UNKNOWN after Power-on reset.
S_REGRDY	16	r	**S_REGRDY status - a handshake flag for transfers through DCRDR** How to work with: • Writing to DCRSR clears the bit to 0. • Completion of the DCRDR transfer then sets the bit to 1 Check DCRDR for more information. 0_B There has been a wrote to the DCRDR, but the transfer is not complete. 1_B The transfer to or from the DCRDR is complete. ***Note**: This bit is only valid when the processor is in Debug State, otherwise the bit is UNKONWN.*
S_HALT	17	r	**S_HALT indicates whether the processor is in Debug state.** 0_B Not in Debug State. 1_B In Debug State.

(table continues...)

(continued)

Field	Bits	Type	Description
S_SLEEP	18	r	**S_SLEEP indicates whether the processor is sleeping.** 0_B Not sleeping. 1_B Sleeping. ***Note**:* *The debugger must set the DHCSR.C_HALT bit to 1 to gain control, or wait for an interrupt or other wake-up event to wake-up the system.*
S_LOCKKUP	19	r	**S_LOCKUP indicates whether the processor is locked up because of an unrecoverable exception.** 0_B Not locked up. 1_B Locked up. ***Note**:* *This bit is only read as 1 when accessed by a remote debugger using the DAP. The value of 1 indicates that the processor is running, but locked up due to an unrecoverable exception case.*
S_RETIRE_ST	24	r	**S_RETIRE_ST - When not in Debug state, indicates whether the processor has completed the execution of an instruction since the last read of DHCSR:** 0_B No instruction has completed since the last DHCSR read. 1_B At least one instructions has completed since last DHCSR read. This is a sticky bit, that clears to 0 on a read of DHCSR. This bit is UNKNOWN: • after a Local reset, but is set to 1 as soon as the processor completes execution of an instruction • when S_LOCKUP is set to 1 • when S_HALT is set to 1 ***Note**:* *When the processor is not in Debug state, a debugger can check this bit to determine if the processor is stalled on a load store or fetch access.*
S_RESET_ST	25	r	**S_RESET_ST indicates whether the processor has been reset since the last read of DHCSR.** 0_B No reset since last DHCSR read. 1_B At least one rest since last DHCSR read. ***Note**:* *This is a sticky bit, that clears to 0 on a read of DHCSR.*
0	15:4, 23:20, 31:26	r	**Reserved**

28.8.2.2 SCS_DHCSR [Write Mode]

SCS_DHCSR — **Address:** $DF0_H

Debug Halting Control and Status Register [Write Mode] — **Reset value:** $0000\ 0000_H$

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

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0												C_MASKINTS	C_STEP	C_HALT	C_DEBUGEN
r												w	w	w	w

Field	Bits	Type	Description
C_DEBUGEN	0	w	**Halting debug enable bit** 0_B Halting debug disabled. 1_B Halting debug enabled. ***Note**:* *If a debugger writes to DHCSR to change the value of this bit from 0 to 1, it must also write 0 to the C_MASKINTS bit, otherwise behavior is UNPREDICTABLE. This bit can only be written from the DAP Access to the DHCSR from Software running on the processor it cannot be set. This bit is 0 after Power-on reset.*
C_HALT	1	w	**Processor halt bit.** The effects of writes to this bit are: 0_B Request a halted processor to run. 1_B Request a running processor to halt. ***Note**:* *This bit is unknown after power-on reset*
C_STEP	2	w	**Processor step bit.** The effects of writes to this bit are: 0_B Single-stepping disabled. 1_B Single-stepping enabled. ***Note**:* *This bit is unknown after power-on reset*

(table continues...)

(continued)

Field	Bits	Type	Description
C_MASKINTS	3	w	**Mask PEDSV, SysTick and external configurable interrupts.** The effects of writes to this bit are: 0_B Do not mask 1_B Mask PendSV, SysTick and external configurable interrupts. The effect of any attempt to change the value of this bit is UNPREDICTABLE unless both: • before the write to DHCSR, the value of the C_HALT bit is 1 • the write to the DHCSR that changes the C_MASKINTS bit also writes 1 to the C_HALT bit. This means that a single write to DHCSR cannot set the C_HALT to 0 and change the value of the C_MASKINTS bit. When DHCSR.C_DEBUGEN is set to 0, the value of this bit is UNKNOWN. This bit is UNKNOWN after Power-on reset.
DBGKEY	31:16	w	**DEBUG Key Bits [31:16]!!!** Software must write **0xA05F** to this field to enable write access to bits [15:0], otherwise the processor ignores the write access
0	15:4	r	**Reserved**

28.8.3 DCRSR - Debug Core Register Selector Register

28.8.3.1 SCS_DCRSR

The DCRSR together with DCRDR (Debug Core Register Data Register) provides debug access to the Arm® core register and special-purpose registers. A write to DCRSR specifies the register to transfer, whether the transfer is a read or a write, and starts the transfer. This register is only accessible in Debug state.

SCS_DCRSR **Address:** $DF4_H$

Debug Core Register Selector Register **Reset value:** $0000\ 0000_H$

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

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0											**REGSEL**				
r											rw				

Field	Bits	Type	Description
REGSEL	4:0	rw	**REGSEL - Specifies the Arm® core register or special-purpose register to transfer** 00000_B-01100_B Arm® core register R0-R12. For example, 0b00000 specifies R0 and 0b00101 specifies R5 01101_B The current SP. See also values 0b10001 and 0b10010. 01110_B LR. 01111_B Debug Return Address. 10000_B xPSR. 10001_B Main stack pointer, MSP. 10010_B Process stack pointer, PSP. 10100_B Bits[31:24] Control; Bits[23:8] Reserved; Bits[7:0] PRIMASK. In each field, the valid bits are packed with leading zeros. For example DCRDR [31L26] is 0b00000.
REGWnR	16	rw	**REGWnR specifies the type of access for the transfer** 0_B read. 1_B write.
0	31:17, 15:5	r	**Reserved**

28.8.4 DCRDR - Debug Core Register Data Register

28.8.4.1 SCS_DCRDR

The DCRDR works with the DCRSR (Debug Core Register Selector Register). The DCRDR provides debug access to the Arm® core registers and special-purpose registers. The DCRDR is the data register for these accesses.

SCS_DCRDR **Address:** $DF8_H$

Debug Core Register Data Register **Reset value:** $XXXX\ XXXX_H$

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

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DBGTMP															
rw															

Field	Bits	Type	Description
DBGTMP	31:0	rw	**DBGTMP - Data temporary cache, for reading and writing register.** This register is UNKONWN: • on reset • when DHCSR.S_HALT = 0 • when DHCSR.S_REGRDY = 0 during execution of a DCRSR based transaction that updates the register.

28.8.5 DEMCR - Debug Exception and Monitor Control Register

28.8.5.1 SCS_DEMCR

The DEMCR purpose is to manage vector catch behavior and enable the DWT.

A Power-on reset sets all register bits to 0. A local reset sets DWTENA to 0 but does not affect VC-HARDERR or VC_CORERESET.

SCS_DEMCR **Address:** DFC_H

Debug Exception and Monitor Control Register **Reset value:** $0000\ 0000_H$

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

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0					VC_HARDERR	0									VC_CORERESET
r					rw	r									rw

Field	Bits	Type	Description
VC_CORERESET	0	rw	**VC_CORERESET - Enable Reset Vector Catch. This causes a Local reset to handle a running system** 0_B Reset Vector Catch disabled. 1_B Reset Vector Catch enabled. ***Note**: If DHCSR.C_DEBUGEN is set to 0, the processor ignores the value of this bit.*

(table continues...)

(continued)

Field	Bits	Type	Description
VC_HARDERR	10	rw	**VC_CORERESET - Enable Reset Vector Catch. This causes a Local reset to halt a running system.** 0_B halting debug trap disabled. 1_B hating debug trap enabled. ***Note**:* *If DHCSR.C_DEBUGEN is set to 0, the processor ignores the value of this bit.*
DWTENA	24	rw	**DWTENA - Global enable for all features configured by the DWT unit.** 0_B DWT disabled. 1_B DWT enabled. ***Note**:* *When DWTENA is set to 0 DWT registers return UNKNOWN values on reads. In addition the processor ignores writes to the DWT while DWTENA is 0.*
0	31:25, 23:16, 15:11, 9:1	r	**Reserved**

28.8.6 DWT_CTRL - Data Watchpoint Control Register

28.8.6.1 DWT_CTRL

The DWT_CTRL register defines the number of comparators implemented.

DWT_CTRL **Address:** 000_H

Debug Halting Control and Status Register **Reset value:** $0000\ 0000_H$

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

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

Field	Bits	Type	Description
NUMCOMP	31:28	rw	**Number of comparators available** 0000_B No comparator support.
0	27:0	r	**Reserved**

28.8.7 DWT_PCSR - Program Counter Sample Register

28.8.7.1 DWT_PCSR

The DWT_PCSR register samples the current value of the program counter. The register is UNKNOW on reset.

DWT_PCSR **Address:** $01C_H$

Program Counter Sample Register **Reset value:** xxxx $xxxx_H$

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

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EIASAMPLE															
rw															

Field	Bits	Type	Description
EIASAMPLE	31:0	rw	**EIASAMPLE** Executed instruction address sample register

28.8.8 DWT_COMPx - DWT Comparator register

28.8.8.1 DWT_COMPx

The DWT_COMPx register provides a reference value for use by comparator x. The value is UNKNOW on reset. DWT_CTRL.NUMCOMP defines the number of implemented DWT_COMPx register, from 0 to (NUMCOMP-1).

The DWT_COMP [1 .. 0] register are implemented.

DWT_COMP0 **Address:** 020_H

DWT Comparator register 0 **Reset value:** xxxx $xxxx_H$

DWT_COMP1 **Address:** 030_H

DWT Comparator register 1 **Reset value:** xxxx $xxxx_H$

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

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
COMP															
rw															

Field	Bits	Type	Description
COMP	31:0	rw	**COMP** Reference value for comparison

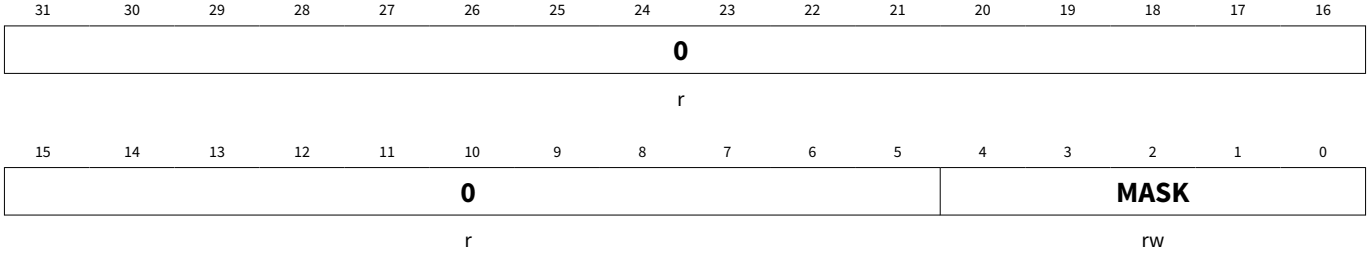
28.8.9 DWT_MASKx - DWT Comparator Mask Register

28.8.9.1 DWT_MASKx

The DWT_MASKx register provides the size of the ignore mask applied to the access address range matching by comparator x. The value is UNKONWN on reset. DWT_CTRL.NUMCOMP defines the number of implemented DWT_COMPx register, from 0 to (NUMCOMP-1).

The DWT_MASK [1 .. 0] register are implemented.

DWT_MASK0	**Address:**	24_H
Debug Halting Control and Status Register	**Reset value:**	$0000\ 0000_H$
DWT_MASK1	**Address:**	34_H
Debug Halting Control and Status Register	**Reset value:**	$0000\ 0000_H$



Field	Bits	Type	Description
MASK	4:0	rw	**MASK** The size of the ignore mask applied to address range matching. Writing all ones to this field and reading it back can be used to determine the maximum mask size supported (not all mask bit must be implemented).
0	31:5	r	**Reserved**

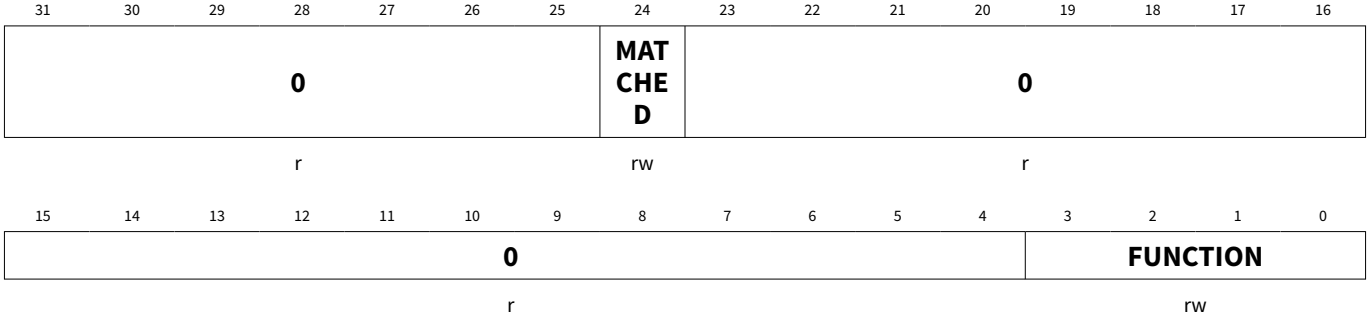
28.8.10 DWT_FUNCTIONx - Comparator Function Register

28.8.10.1 DWT_FUNCTIONx

The DWT_FUNCTIONx register controls the operation of the comparator DWT_COMPx register. DWT_CTRL.NUMCOMP defines the number of implemented DWT_FUNCTIONx registers, from 0 to (NUMCOMP-1).

The DWT_FUNCTION [1 .. 0] register are implemented.

DWT_FUNCTION0	**Address:**	28_H
Comparator Function Register	**Reset value:**	$0000\ 0000_H$
DWT_FUNCTION1	**Address:**	38_H
Comparator Function Register	**Reset value:**	$0000\ 0000_H$



Field	Bits	Type	Description
FUNCTION	3:0	rw	**FUNCTION** Select action on comparator match- 0000_B Disabled. 0001_B Reserved. 0010_B Reserved. 0011_B Reserved. 0100_B PC watchpoint event - input Iaddr. 0101_B Watchpoint event - input Daddr (read only) 0110_B Watchpoint event - input Daddr (write only) 0111_B Watchpoint event - input Daddr (read/write) $1xxx_B$ reserved ***Note**: This field is set to 0 on a Power-on reset.*
MATCHED	24	rw	**MATCHED** Comparator match. It indicates that the operation defined by FUNCTION has occurred since the bit was last read. 0_B the associated comparator has matched. 1_B the associated comparator has not matched. ***Note**: Reading the register clears this bit to 0.*
0	31:25, 23:16, 15:4	r	**Reserved**

28.8.11 BP_CTRL - Breakpoint Control Register

28.8.11.1 BP_CTRL

The Breakpoint Control Register provides BPU implementation information and the global enable for the BPU.

BP_CTRL **Address:** 000_H

Breakpoint Control Register **Reset value:** $0000\ 0040_H$

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								**NUM_CODE**				**0**		**KEY**	**ENABLE**
r								r				r		rw	rw

Field	Bits	Type	Description
ENABLE	0	rw	**ENABLE the BPU** 0_B BPU is disabled. 1_B BPU is enabled. ***Note**: This bit is set to 0 on Power-on reset.*
KEY	1	rw	**KEY reads as 0 on reads, should be 1 for writes.** If written as zero, the write to the register is ignored.
NUM_CODE	7:4	r	**NUM_CODE, the number of breakpoint comparators.**
0	31:8, 3:2	r	**Reserved**

28.8.12 Breakpoint Comparator Registers

28.8.12.1 BP_COMPx

The BP_COMPx register holds a breakpoint address for comparison with instruction addresses in the Code memory region. A comparator can only be enabled when BP_CTRL.ENABLE is set to 1. BP_CTRL.NUM_CODE defines the number of implemented BP_COMPx registers, from 0 to (NUM_CODE-1).

The BP_COMP [3 .. 0] register are implemented.

BP_COMP0 **Address:** 008_H

Breakpoint Comparator X Register **Reset value:** $0000\ 0000_H$

BP_COMP1 **Address:** $00C_H$

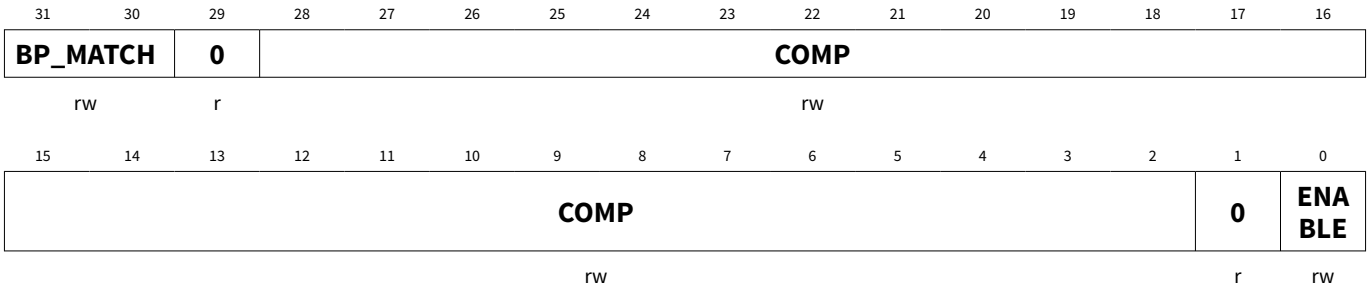
Breakpoint Comparator X Register **Reset value:** $0000\ 0000_H$

BP_COMP2 **Address:** 010_H

Breakpoint Comparator X Register **Reset value:** $0000\ 0000_H$

BP_COMP3 **Address:** 014_H

Breakpoint Comparator X Register **Reset value:** $0000\ 0000_H$



Field	Bits	Type	Description
ENABLE	0	rw	**ENABLE the comparator** 0_B Comparator is disabled. 1_B Comparator is enabled. ***Note**:* *This bit is set to 0 on a Power-on reset. BP_CTRL:ENABLE must also be set to 1 to enable a comparator.*
COMP	28:2	rw	**Stores bits [28:0] of the comparison address.** The comparison address is compared with the address from the Code memory region. Bits [31:29] and [1:0] of the comparison address are zero. ***Note**:* *The field is UNKNOWN on Power-on reset.*
BP_MATCH	31:30	rw	**BP_MATCH defines the behavior when the COMP address is matched.** 00_B no breakpoint matching. 01_B breakpoint on lower halfword, upper is unaffected. 10_B breakpoint on upper halfword, lower is unaffected. 11_B breakpoint on both lower and upper halfwords. ***Note**:* *The field is UNKNOWN on reset.*
0	29, 1	r	**Reserved**

Revision history

Document revision	Date	Description of changes
**	2026-06-26	Initial release

Trademarks

PSOC™, formerly known as PSoC™, is a trademark of Infineon Technologies. Any references to PSoC™ in this document or others shall be deemed to refer to PSOC™.

Trademarks

All referenced product or service names and trademarks are the property of their respective owners.

Edition 2026-06-26

Published by

Infineon Technologies AG

81726 Munich, Germany

© 2026 Infineon Technologies AG

All Rights Reserved.

Do you have a question about any aspect of this document?

Email: erratum@infineon.com

Document reference

IFX-xpu1772622145412

Important notice

Products which may also include samples and may be comprised of hardware or software or both ("Product(s)") are sold or provided and delivered by Infineon Technologies AG and its affiliates ("Infineon") subject to the terms and conditions of the frame supply contract or other written agreement(s) executed by a customer and Infineon or, in the absence of the foregoing, the applicable Sales Conditions of Infineon. General terms and conditions of a customer or deviations from applicable Sales Conditions of Infineon shall only be binding for Infineon if and to the extent Infineon has given its express written consent.

For the avoidance of doubt, Infineon disclaims all warranties of non-infringement of third-party rights and implied warranties such as warranties of fitness for a specific use/purpose or merchantability. Infineon shall not be responsible for any information with respect to samples, the application or customer's specific use of any Product or for any examples or typical values given in this document.

If this document is marked as 'Preliminary', 'Target', 'Draft' or 'Proposal', Infineon reserves the right to change all information given in this document at any time without notice. Subject to the development and release of a Product for series supply by Infineon, the technical specifications of the Product are set forth in the relevant datasheet provided by Infineon without such marking.

The data contained in this document is exclusively intended for technically qualified and skilled customer representatives. It is the responsibility of the customer to evaluate the suitability of the Product for the intended application and the customer's specific use and to verify all relevant technical data contained in this document in the intended application and the customer's specific use. The customer is responsible for properly designing, programming, and testing the functionality and safety of the intended application, as well as complying with any legal requirements related to its use.

Unless otherwise explicitly approved by Infineon, Products may not be used in any application where a failure of the Products or any consequences of the use thereof can reasonably be expected to result in personal injury.

However, the foregoing shall not prevent the customer from using any Product in such fields of use that Infineon has explicitly designed and sold it for, provided that the overall responsibility for the application lies with the customer. Infineon expressly reserves the right to use its content for commercial text and data mining (TDM) according to applicable laws, e.g. Section 44b of the German Copyright Act (UrhG). If the Product includes security features:

Because no computing device can be absolutely secure, and despite security measures implemented in the Product, Infineon does not guarantee that the Product will be free from intrusion, data theft or loss, or other breaches ("Security Breaches"), and Infineon shall have no liability arising out of any Security Breaches.

If this document includes or references software:

The software is owned by Infineon under the intellectual property laws and treaties of the United States, Germany, and other countries worldwide. All rights reserved. Therefore, you may use the software only as provided in the software license agreement accompanying the software.

If no software license agreement applies, Infineon hereby grants you a personal, non-exclusive, non-transferable license (without the right to sublicense) under its intellectual property rights in the software (a) for software provided in source code form, to modify and reproduce the software solely for use with Infineon hardware products, only internally within your organization, and (b) to distribute the software in binary code form externally to end users, solely for use on Infineon hardware products. Any other use, reproduction, modification, translation, or compilation of the software is prohibited.

For further information on the Product, technology, delivery terms and conditions, and prices, please contact your nearest Infineon office or visit www.infineon.com.