# XMC1300 AB-Step

Microcontroller Series
for Industrial Applications

XMC1000 Family

ARM® Cortex®-M0
32-bit processor core

## Reference Manual

V1.3 2016-08

Microcontrollers

**Edition 2016-08**

**Published by**
**Infineon Technologies AG**
**81726 Munich, Germany**

**© 2016 Infineon Technologies AG**
**All Rights Reserved.**

### Legal Disclaimer

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

### Information

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

### Warnings

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

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

# XMC1300 AB-Step

Microcontroller Series
for Industrial Applications

XMC1000 Family

ARM® Cortex®-M0
32-bit processor core

## Reference Manual

V1.3 2016-08

Microcontrollers

**XMC1300 Reference Manual**

**Revision History: V1.3 2016-08**

Previous Version: V1.2

| Page | Subjects |
|---|---|
| | Memory Organization chapter |
| 8-4 | • Corrected user readable to non-readable ROM boundary address |
| | NVM chapter |
| 9-5 | • Add description about using flash with fixed wait states |
| 9-16 | • Bit field WS added to NVMCONF register |
| 9-18 | • Added CONFIG1 register |

**Trademarks**

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

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

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

**We Listen to Your Comments**

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

**mcdocu.comments@infineon.com**




Subject to Agreement on the Use of Product Information

# Table of Contents

**1 Introduction** . . . . . 1-1
1.1 Overview . . . . . 1-1
1.1.1 Block Diagram . . . . . 1-3
1.2 Core Processing Units . . . . . 1-3
1.2.1 Central Processing Unit (CPU) . . . . . 1-4
1.2.2 Programmable Multiple Priority Interrupt System (NVIC) . . . . . 1-4
1.2.3 Math Coprocessor (MATH) . . . . . 1-4
1.3 System Units . . . . . 1-4
1.3.1 Memories . . . . . 1-4
1.3.2 Watchdog Timer (WDT) . . . . . 1-5
1.3.3 Real Timer Clock (RTC) . . . . . 1-5
1.3.4 System Control unit (SCU) . . . . . 1-5
1.3.5 Pseudo Random Bit Generator (PRNG) . . . . . 1-5
1.4 Peripherals Units . . . . . 1-5
1.5 Debug Unit . . . . . 1-7

**2 Central Processing Unit (CPU)** . . . . . 2-1
2.1 Overview . . . . . 2-1
2.1.1 Features . . . . . 2-2
2.1.2 Block Diagram . . . . . 2-2
2.2 Programmers Model . . . . . 2-3
2.2.1 Processor Mode . . . . . 2-3
2.2.2 Stacks . . . . . 2-3
2.2.3 Core Registers . . . . . 2-5
2.2.4 Exceptions and Interrupts . . . . . 2-15
2.2.5 Data Types . . . . . 2-15
2.2.6 The Cortex Microcontroller Software Interface Standard . . . . . 2-15
2.2.7 CMSIS Functions . . . . . 2-16
2.3 Memory Model . . . . . 2-18
2.3.1 Memory Regions, Types and Attributes . . . . . 2-19
2.3.2 Memory System Ordering of Memory Accesses . . . . . 2-19
2.3.3 Behavior of Memory Accesses . . . . . 2-20
2.3.4 Software Ordering of Memory Accesses . . . . . 2-21
2.3.5 Memory Endianness . . . . . 2-22
2.3.5.1 Little-endian format . . . . . 2-22
2.4 Instruction Set . . . . . 2-23
2.4.1 Intrinsic Functions . . . . . 2-25
2.5 Exception Model . . . . . 2-26
2.5.1 Exception States . . . . . 2-26
2.5.2 Exception Types . . . . . 2-27
2.5.3 Exception Handlers . . . . . 2-28


Subject to Agreement on the Use of Product Information

2.5.4 Vector Table . . . 2-28
2.5.4.1 Vector Table Remap . . . 2-29
2.5.5 Exception Priorities . . . 2-30
2.5.6 Exception Entry and Return . . . 2-31
2.5.6.1 Exception entry . . . 2-32
2.5.6.2 Exception return . . . 2-34
2.6 Fault Handling . . . 2-35
2.6.1 Lockup . . . 2-35
2.7 Power Management . . . 2-36
2.7.1 Entering Sleep Mode . . . 2-36
2.7.2 Wakeup from Sleep Mode . . . 2-37
2.7.3 Power Management Programming Hints . . . 2-37
2.8 Private Peripherals . . . 2-38
2.8.1 About the Private Peripherals . . . 2-38
2.8.2 System control block . . . 2-38
2.8.2.1 System control block usage hints and tips . . . 2-38
2.8.3 System timer, SysTick . . . 2-38
2.8.3.1 SysTick usage hints and tips . . . 2-39
2.9 PPB Registers . . . 2-40
2.9.1 SCS Registers . . . 2-41
2.9.2 SysTick Registers . . . 2-52

**3 Bus System . . . 3-1**
3.1 Bus Interfaces . . . 3-1

**4 Service Request Processing . . . 4-1**
4.1 Overview . . . 4-1
4.1.1 Features . . . 4-1
4.1.2 Block Diagram . . . 4-1
4.2 Service Request Distribution . . . 4-3

**5 Interrupt Subsystem . . . 5-1**
5.1 Nested Vectored Interrupt Controller (NVIC) . . . 5-1
5.1.1 Features . . . 5-1
5.1.2 Interrupt Node Assignment . . . 5-1
5.1.3 Interrupt Signal Generation . . . 5-2
5.1.4 NVIC design hints and tips . . . 5-3
5.1.5 Accessing CPU Registers using CMSIS . . . 5-4
5.1.6 Interrupt Priority . . . 5-4
5.1.7 Interrupt Latency . . . 5-5
5.2 General Module Interrupt Structure . . . 5-6
5.3 Registers . . . 5-8
5.3.1 NVIC Registers . . . 5-9
5.4 Interrupt Request Source Overview . . . 5-13


Subject to Agreement on the Use of Product Information

**6 Event Request Unit (ERU) . . . . . 6-1**
6.1 Features . . . . . 6-1
6.2 Overview . . . . . 6-1
6.3 Event Request Select Unit (ERS) . . . . . 6-2
6.4 Event Trigger Logic (ETLx) . . . . . 6-3
6.5 Cross Connect Matrix . . . . . 6-4
6.6 Output Gating Unit (OGUy) . . . . . 6-5
6.7 Power, Reset and Clock . . . . . 6-8
6.8 Initialization and System Dependencies . . . . . 6-9
6.9 Registers . . . . . 6-10
6.9.1 ERU Registers . . . . . 6-11
6.10 Interconnects . . . . . 6-16
6.10.1 ERU0 Connections . . . . . 6-17

**7 MATH Coprocessor (MATH) . . . . . 7-1**
7.1 Overview . . . . . 7-1
7.1.1 Features . . . . . 7-1
7.1.2 Block Diagram . . . . . 7-1
7.2 Divider Unit (DIV) . . . . . 7-2
7.2.1 Features . . . . . 7-2
7.2.2 Division Operation . . . . . 7-2
7.2.2.1 Start Mode Selection . . . . . 7-4
7.2.2.2 Error Handling . . . . . 7-4
7.2.3 Operand/Result Pre-/Post-Processing . . . . . 7-6
7.3 CORDIC Coprocessor . . . . . 7-7
7.3.1 Overview . . . . . 7-7
7.3.1.1 Features . . . . . 7-7
7.3.1.2 Block Diagram . . . . . 7-8
7.3.2 Functional Overview . . . . . 7-10
7.3.2.1 Operation of the CORDIC Coprocessor . . . . . 7-10
7.3.2.2 Normalized Result Data . . . . . 7-11
7.3.3 CORDIC Coprocessor Operating Modes . . . . . 7-12
7.3.3.1 Domains of Convergence . . . . . 7-14
7.3.3.2 Overflow Considerations . . . . . 7-15
7.3.4 CORDIC Coprocessor Data Format . . . . . 7-15
7.3.5 Accuracy of CORDIC Coprocessor . . . . . 7-17
7.3.6 Performance of CORDIC Coprocessor . . . . . 7-19
7.3.7 CORDIC Coprocessor Look-Up Tables . . . . . 7-19
7.3.7.1 Arctangent and Hyperbolic Arctangent Look-Up Tables . . . . . 7-20
7.3.7.2 Linear Function Emulated Look-Up Table . . . . . 7-21
7.4 Global Functions . . . . . 7-22
7.4.1 Result Chaining . . . . . 7-22
7.4.1.1 Result Chaining when Start Mode = 0 . . . . . 7-23


Subject to Agreement on the Use of Product Information

7.4.1.2 Handling Busy Flags when Result Chaining is Enabled . . . . . 7-23
7.5 Service Request Generation . . . . . 7-24
7.6 Debug Behaviour . . . . . 7-25
7.7 Power, Reset and Clock . . . . . 7-26
7.8 Registers . . . . . 7-27
7.8.1 Global Registers Description . . . . . 7-28
7.8.2 Divider Registers Description . . . . . 7-36
7.8.3 CORDIC Registers Description . . . . . 7-41
7.9 Interconnects . . . . . 7-46

**8 Memory Organization** . . . . . 8-1
8.1 Overview . . . . . 8-1
8.1.1 Features . . . . . 8-1
8.1.2 Cortex-M0 Address Space . . . . . 8-1
8.2 Memory Regions . . . . . 8-2
8.3 Memory Map . . . . . 8-3
8.4 Memory Access . . . . . 8-9
8.4.1 Flash Memory Access . . . . . 8-9
8.4.2 SRAM Access . . . . . 8-9
8.4.3 ROM Access . . . . . 8-9
8.5 Memory Protection Strategy . . . . . 8-9
8.5.1 Intellectual Property (IP) Protection . . . . . 8-10
8.5.1.1 Blocking of Unauthorized External Access . . . . . 8-10
8.5.2 Memory Access Protection during Run-time . . . . . 8-10
8.5.2.1 Bit Protection Scheme . . . . . 8-10
8.5.2.2 Peripheral Privilege Access Control . . . . . 8-12
8.6 Service Request Generation . . . . . 8-12
8.7 Debug Behaviour . . . . . 8-13
8.8 Power, Reset and Clock . . . . . 8-13
8.9 Initialization and System Dependencies . . . . . 8-13

**9 Flash Architecture** . . . . . 9-1
9.1 Overview . . . . . 9-1
9.1.1 Features . . . . . 9-1
9.2 Definitions . . . . . 9-1
9.2.1 Logical and Physical States . . . . . 9-2
9.2.2 Data Portions . . . . . 9-2
9.2.3 Address Types . . . . . 9-3
9.2.4 Module Specific Definitions . . . . . 9-3
9.3 Module Components . . . . . 9-3
9.3.1 Memory Cell Array . . . . . 9-4
9.3.1.1 Page . . . . . 9-4
9.3.1.2 Sector . . . . . 9-5
9.4 Functional Description . . . . . 9-5


Subject to Agreement on the Use of Product Information

9.4.1 SFR Accesses . . . 9-5
9.4.2 Memory Read . . . 9-5
9.4.2.1 Flash read access with fixed wait states . . . 9-5
9.4.3 Memory Write . . . 9-6
9.4.4 Memory Erase . . . 9-7
9.4.5 Sector Erase . . . 9-7
9.4.6 Verify . . . 9-8
9.4.7 Erase-Protection and Write-Protection . . . 9-8
9.5 Redundancy . . . 9-8
9.6 Properties and Implementation of Error Correcting Code (ECC) . . . 9-9
9.7 Service Request Generation . . . 9-9
9.8 Power, Reset and Clock . . . 9-9
9.8.1 Power Supply . . . 9-9
9.8.2 Power Saving Modes . . . 9-9
9.8.2.1 NVM Idle Mode . . . 9-10
9.8.2.2 NVM Sleep Mode . . . 9-10
9.8.3 Reset . . . 9-10
9.8.4 Clock . . . 9-10
9.9 Registers . . . 9-11
9.9.1 NVM Registers . . . 9-12
9.10 Example Sequences . . . 9-19
9.10.1 Writing to Memory . . . 9-19
9.10.1.1 Writing a Single Block . . . 9-19
9.10.1.2 Writing Blocks . . . 9-19
9.10.2 Erasing Memory . . . 9-20
9.10.2.1 Erasing a Single Page . . . 9-20
9.10.2.2 Erasing Pages . . . 9-20
9.10.2.3 Erasing a Single Sector . . . 9-20
9.10.2.4 Erasing Sectors . . . 9-21
9.10.3 Verifying Memory . . . 9-21
9.10.3.1 Verifying a Single Block . . . 9-21
9.10.3.2 Verifying Blocks . . . 9-21
9.10.4 Writing to an Already Written Block . . . 9-22
9.10.5 Sleep Mode . . . 9-24
9.10.6 Timing . . . 9-25

**10 Peripheral Access Unit (PAU) . . . 10-1**
10.1 Overview . . . 10-1
10.1.1 Features . . . 10-1
10.2 Peripheral Privilege Access Control . . . 10-1
10.3 Peripheral Availability and Memory Size . . . 10-2
10.4 Service Request Generation . . . 10-2
10.5 Debug Behaviour . . . 10-2


Subject to Agreement on the Use of Product Information

10.6 Power, Reset and Clock . . . . . 10-3
10.7 Initialization and System Dependencies . . . . . 10-3
10.8 PAU Registers . . . . . 10-3
10.8.1 Peripheral Privilege Access Registers (PRIVDISn) . . . . . 10-4
10.8.2 Peripheral Availability Registers (AVAILn) . . . . . 10-8
10.8.3 Memory Size Registers . . . . . 10-12

**11 Window Watchdog Timer (WDT)** . . . . . 11-1
11.1 Overview . . . . . 11-1
11.1.1 Features . . . . . 11-1
11.1.2 Block Diagram . . . . . 11-2
11.2 Time-Out Mode . . . . . 11-3
11.3 Pre-warning Mode . . . . . 11-3
11.4 Bad Service Operation . . . . . 11-4
11.5 Service Request Processing . . . . . 11-6
11.6 Debug Behavior . . . . . 11-6
11.7 Power, Reset and Clock . . . . . 11-6
11.8 Initialization and Control Sequence . . . . . 11-6
11.8.1 Initialization & Start of Operation . . . . . 11-6
11.8.2 Software Stop & Resume Operation . . . . . 11-7
11.8.3 Enter Sleep/Deep-Sleep & Resume Operation . . . . . 11-7
11.8.4 Pre-warning Alarm Handling . . . . . 11-8
11.9 WDT Registers . . . . . 11-9
11.9.1 Registers Description . . . . . 11-9
11.10 Interconnects . . . . . 11-16

**12 Real Time Clock (RTC)** . . . . . 12-1
12.1 Overview . . . . . 12-1
12.1.1 Features . . . . . 12-1
12.1.2 Block Diagram . . . . . 12-1
12.2 RTC Operation . . . . . 12-2
12.3 Register Access Operations . . . . . 12-3
12.4 Service Request Processing . . . . . 12-4
12.4.1 Periodic Service Request . . . . . 12-4
12.4.2 Timer Alarm Service Request . . . . . 12-4
12.5 Debug Behavior . . . . . 12-4
12.6 Power, Reset and Clock . . . . . 12-4
12.7 Initialization and Control Sequence . . . . . 12-5
12.7.1 Initialization & Start of Operation . . . . . 12-5
12.7.2 Configure and Enable Periodic Event . . . . . 12-6
12.7.3 Configure and Enable Timer Event . . . . . 12-6
12.8 RTC Registers . . . . . 12-6
12.8.1 Registers Description . . . . . 12-7
12.9 Interconnects . . . . . 12-18


Subject to Agreement on the Use of Product Information

**13 System Control Unit (SCU)** . . . . . 13-1
13.1 Overview . . . . . 13-1
13.1.1 Features . . . . . 13-1
13.1.2 Block Diagram . . . . . 13-2
13.2 Miscellaneous Control Functions (GCU) . . . . . 13-4
13.2.1 Service Requests Handling . . . . . 13-4
13.2.1.1 Service Request Sources . . . . . 13-4
13.2.2 SRAM Memory Content Protection . . . . . 13-6
13.2.3 Summary of ID . . . . . 13-6
13.3 Power Management (PCU) . . . . . 13-8
13.3.1 Functional Description . . . . . 13-8
13.3.2 System States . . . . . 13-8
13.3.3 Embedded Voltage Regulator (EVR) . . . . . 13-10
13.3.4 Power-on Reset . . . . . 13-10
13.3.5 Power Validation . . . . . 13-10
13.3.6 Supply Voltage Monitoring . . . . . 13-10
13.3.7 $V_{DDC}$ Response During Load Change . . . . . 13-11
13.3.8 Flash Power Control . . . . . 13-12
13.4 Reset Control (RCU) . . . . . 13-13
13.4.1 Functional Description . . . . . 13-13
13.4.2 Reset Status . . . . . 13-14
13.5 Clock Control (CCU) . . . . . 13-15
13.5.1 Features . . . . . 13-15
13.5.2 Clock System and Control . . . . . 13-15
13.5.2.1 Oscillator Watchdog . . . . . 13-18
13.5.2.2 Loss of Clock Detection and Recovery . . . . . 13-18
13.5.2.3 Standby Clock Failure . . . . . 13-19
13.5.2.4 Startup Control for System Clock . . . . . 13-19
13.5.3 Clock Gating Control . . . . . 13-19
13.5.4 Calibrating DCO based on Temperature . . . . . 13-20
13.6 Service Request Generation . . . . . 13-21
13.7 Debug Behavior . . . . . 13-21
13.8 Power, Reset and Clock . . . . . 13-21
13.9 Registers . . . . . 13-23
13.9.1 PCU Registers (ANACTRL) . . . . . 13-25
13.9.2 PCU Registers (SCU) . . . . . 13-26
13.9.3 CCU Registers (SCU) . . . . . 13-27
13.9.4 CCU Registers (ANACTRL) . . . . . 13-35
13.9.5 RCU Registers (SCU) . . . . . 13-36
13.9.6 GCU Registers (SCU) . . . . . 13-41

**14 Pseudo Random Number Generator (PRNG)** . . . . . 14-1
14.1 Overview . . . . . 14-1


Subject to Agreement on the Use of Product Information

14.1.1 Features . . . 14-1
14.2 Description of Operation Modes . . . 14-1
14.2.1 Key Loading Mode . . . 14-1
14.2.2 Streaming Mode . . . 14-2
14.2.3 Refreshing and Restarting a Random Bit Stream . . . 14-2
14.3 Service Request Generation . . . 14-3
14.4 Debug Behavior . . . 14-3
14.5 Power, Reset and Clock . . . 14-3
14.6 Initialization and System Dependencies . . . 14-3
14.7 Registers . . . 14-3
14.7.1 Data Registers . . . 14-4
14.7.2 Control Registers . . . 14-6

**15 Universal Serial Interface Channel (USIC)** . . . 15-1
15.1 Overview . . . 15-1
15.1.1 Features . . . 15-1
15.2 Operating the USIC . . . 15-6
15.2.1 USIC Structure Overview . . . 15-6
15.2.1.1 Channel Structure . . . 15-6
15.2.1.2 Input Stages . . . 15-6
15.2.1.3 Output Signals . . . 15-8
15.2.1.4 Baud Rate Generator . . . 15-9
15.2.1.5 Channel Events and Interrupts . . . 15-10
15.2.1.6 Data Shifting and Handling . . . 15-10
15.2.2 Operating the USIC Communication Channel . . . 15-14
15.2.2.1 Protocol Control and Status . . . 15-15
15.2.2.2 Mode Control . . . 15-16
15.2.2.3 General Channel Events and Interrupts . . . 15-17
15.2.2.4 Data Transfer Events and Interrupts . . . 15-18
15.2.2.5 Baud Rate Generator Event and Interrupt . . . 15-20
15.2.2.6 Protocol-specific Events and Interrupts . . . 15-22
15.2.3 Operating the Input Stages . . . 15-22
15.2.3.1 General Input Structure . . . 15-23
15.2.3.2 Digital Filter . . . 15-25
15.2.3.3 Edge Detection . . . 15-25
15.2.3.4 Selected Input Monitoring . . . 15-26
15.2.3.5 Loop Back Mode . . . 15-26
15.2.4 Operating the Baud Rate Generator . . . 15-26
15.2.4.1 Fractional Divider . . . 15-26
15.2.4.2 External Frequency Input . . . 15-27
15.2.4.3 Divider Mode Counter . . . 15-27
15.2.4.4 Capture Mode Timer . . . 15-28
15.2.4.5 Time Quanta Counter . . . 15-29


Subject to Agreement on the Use of Product Information

15.2.4.6 Master and Shift Clock Output Configuration . . . . . 15-30
15.2.5 Operating the Transmit Data Path . . . . . 15-31
15.2.5.1 Transmit Buffering . . . . . 15-31
15.2.5.2 Transmit Data Shift Mode . . . . . 15-32
15.2.5.3 Transmit Control Information . . . . . 15-33
15.2.5.4 Transmit Data Validation . . . . . 15-34
15.2.6 Operating the Receive Data Path . . . . . 15-36
15.2.6.1 Receive Buffering . . . . . 15-36
15.2.6.2 Receive Data Shift Mode . . . . . 15-37
15.2.6.3 Baud Rate Constraints . . . . . 15-38
15.2.7 Hardware Port Control . . . . . 15-38
15.2.8 Operating the FIFO Data Buffer . . . . . 15-39
15.2.8.1 FIFO Buffer Partitioning . . . . . 15-40
15.2.8.2 Transmit Buffer Events and Interrupts . . . . . 15-41
15.2.8.3 Receive Buffer Events and Interrupts . . . . . 15-45
15.2.8.4 FIFO Buffer Bypass . . . . . 15-50
15.2.8.5 FIFO Access Constraints . . . . . 15-51
15.2.8.6 Handling of FIFO Transmit Control Information . . . . . 15-52
15.3 Asynchronous Serial Channel (ASC = UART) . . . . . 15-54
15.3.1 Signal Description . . . . . 15-54
15.3.2 Frame Format . . . . . 15-55
15.3.2.1 Idle Time . . . . . 15-56
15.3.2.2 Start Bit Detection . . . . . 15-57
15.3.2.3 Data Field . . . . . 15-57
15.3.2.4 Parity Bit . . . . . 15-57
15.3.2.5 Stop Bit(s) . . . . . 15-57
15.3.3 Operating the ASC . . . . . 15-58
15.3.3.1 Bit Timing . . . . . 15-58
15.3.3.2 Baud Rate Generation . . . . . 15-59
15.3.3.3 Noise Detection . . . . . 15-60
15.3.3.4 Collision Detection . . . . . 15-60
15.3.3.5 Pulse Shaping . . . . . 15-60
15.3.3.6 Automatic Shadow Mechanism . . . . . 15-62
15.3.3.7 End of Frame Control . . . . . 15-62
15.3.3.8 Mode Control Behavior . . . . . 15-63
15.3.3.9 Disabling ASC Mode . . . . . 15-63
15.3.3.10 Protocol Interrupt Events . . . . . 15-63
15.3.3.11 Data Transfer Interrupt Handling . . . . . 15-64
15.3.3.12 Baud Rate Generator Interrupt Handling . . . . . 15-64
15.3.3.13 Protocol-Related Argument and Error . . . . . 15-65
15.3.3.14 Receive Buffer Handling . . . . . 15-65
15.3.3.15 Sync-Break Detection . . . . . 15-65
15.3.3.16 Transfer Status Indication . . . . . 15-65


Subject to Agreement on the Use of Product Information

15.3.4 ASC Protocol Registers . . . . . 15-66
15.3.4.1 ASC Protocol Control Register . . . . . 15-66
15.3.4.2 ASC Protocol Status Register . . . . . 15-69
15.3.5 Hardware LIN Support . . . . . 15-72
15.4 Synchronous Serial Channel (SSC) . . . . . 15-74
15.4.1 Signal Description . . . . . 15-74
15.4.1.1 Transmit and Receive Data Signals . . . . . 15-76
15.4.1.2 Shift Clock Signals . . . . . 15-77
15.4.1.3 Slave Select Signals . . . . . 15-80
15.4.2 Operating the SSC . . . . . 15-82
15.4.2.1 Automatic Shadow Mechanism . . . . . 15-82
15.4.2.2 Mode Control Behavior . . . . . 15-82
15.4.2.3 Disabling SSC Mode . . . . . 15-83
15.4.2.4 Data Frame Control . . . . . 15-83
15.4.2.5 Parity Mode . . . . . 15-83
15.4.2.6 Transfer Mode . . . . . 15-85
15.4.2.7 Data Transfer Interrupt Handling . . . . . 15-85
15.4.2.8 Baud Rate Generator Interrupt Handling . . . . . 15-86
15.4.2.9 Protocol-Related Argument and Error . . . . . 15-86
15.4.2.10 Receive Buffer Handling . . . . . 15-86
15.4.2.11 Multi-IO SSC Protocols . . . . . 15-86
15.4.3 Operating the SSC in Master Mode . . . . . 15-88
15.4.3.1 Baud Rate Generation . . . . . 15-89
15.4.3.2 MSLS Generation . . . . . 15-90
15.4.3.3 Automatic Slave Select Update . . . . . 15-91
15.4.3.4 Slave Select Delay Generation . . . . . 15-92
15.4.3.5 Protocol Interrupt Events . . . . . 15-93
15.4.3.6 End-of-Frame Control . . . . . 15-94
15.4.4 Operating the SSC in Slave Mode . . . . . 15-96
15.4.4.1 Protocol Interrupts . . . . . 15-96
15.4.4.2 End-of-Frame Control . . . . . 15-97
15.4.5 SSC Protocol Registers . . . . . 15-98
15.4.5.1 SSC Protocol Control Registers . . . . . 15-98
15.4.5.2 SSC Protocol Status Register . . . . . 15-102
15.4.6 SSC Timing Considerations . . . . . 15-104
15.4.6.1 Closed-loop Delay . . . . . 15-104
15.4.6.2 Delay Compensation in Master Mode . . . . . 15-107
15.4.6.3 Complete Closed-loop Delay Compensation . . . . . 15-108
15.5 Inter-IC Bus Protocol (IIC) . . . . . 15-109
15.5.1 Introduction . . . . . 15-109
15.5.1.1 Signal Description . . . . . 15-109
15.5.1.2 Symbols . . . . . 15-110
15.5.1.3 Frame Format . . . . . 15-111


Subject to Agreement on the Use of Product Information

15.5.2 Operating the IIC . . . . . 15-112
15.5.2.1 Transmission Chain . . . . . 15-113
15.5.2.2 Byte Stretching . . . . . 15-113
15.5.2.3 Master Arbitration . . . . . 15-113
15.5.2.4 Non-Acknowledge and Error Conditions . . . . . 15-114
15.5.2.5 Mode Control Behavior . . . . . 15-114
15.5.2.6 Data Transfer Interrupt Handling . . . . . 15-114
15.5.2.7 IIC Protocol Interrupt Events . . . . . 15-115
15.5.2.8 Baud Rate Generator Interrupt Handling . . . . . 15-116
15.5.2.9 Receiver Address Acknowledge . . . . . 15-116
15.5.2.10 Receiver Handling . . . . . 15-117
15.5.2.11 Receiver Status Information . . . . . 15-117
15.5.3 Symbol Timing . . . . . 15-118
15.5.3.1 Start Symbol . . . . . 15-119
15.5.3.2 Repeated Start Symbol . . . . . 15-119
15.5.3.3 Stop Symbol . . . . . 15-120
15.5.3.4 Data Bit Symbol . . . . . 15-120
15.5.4 Data Flow Handling . . . . . 15-121
15.5.4.1 Transmit Data Formats . . . . . 15-121
15.5.4.2 Valid Master Transmit Data Formats . . . . . 15-123
15.5.4.3 Master Transmit/Receive Modes . . . . . 15-126
15.5.4.4 Slave Transmit/Receive Modes . . . . . 15-128
15.5.5 IIC Protocol Registers . . . . . 15-129
15.5.5.1 IIC Protocol Control Registers . . . . . 15-129
15.5.5.2 IIC Protocol Status Register . . . . . 15-132
15.6 Inter-IC Sound Bus Protocol (IIS) . . . . . 15-135
15.6.1 Introduction . . . . . 15-135
15.6.1.1 Signal Description . . . . . 15-135
15.6.1.2 Protocol Overview . . . . . 15-136
15.6.1.3 Transfer Delay . . . . . 15-137
15.6.1.4 Connection of External Audio Components . . . . . 15-137
15.6.2 Operating the IIS . . . . . 15-138
15.6.2.1 Frame Length and Word Length Configuration . . . . . 15-138
15.6.2.2 Automatic Shadow Mechanism . . . . . 15-139
15.6.2.3 Mode Control Behavior . . . . . 15-139
15.6.2.4 Transfer Delay . . . . . 15-139
15.6.2.5 Parity Mode . . . . . 15-141
15.6.2.6 Transfer Mode . . . . . 15-141
15.6.2.7 Data Transfer Interrupt Handling . . . . . 15-141
15.6.2.8 Baud Rate Generator Interrupt Handling . . . . . 15-142
15.6.2.9 Protocol-Related Argument and Error . . . . . 15-142
15.6.2.10 Transmit Data Handling . . . . . 15-142
15.6.2.11 Receive Buffer Handling . . . . . 15-143


Subject to Agreement on the Use of Product Information

15.6.2.12 Loop-Delay Compensation . . . . . 15-143
15.6.3 Operating the IIS in Master Mode . . . . . 15-143
15.6.3.1 Baud Rate Generation . . . . . 15-144
15.6.3.2 WA Generation . . . . . 15-145
15.6.3.3 Master Clock Output . . . . . 15-145
15.6.3.4 Protocol Interrupt Events . . . . . 15-146
15.6.4 Operating the IIS in Slave Mode . . . . . 15-146
15.6.4.1 Protocol Events and Interrupts . . . . . 15-147
15.6.5 IIS Protocol Registers . . . . . 15-147
15.6.5.1 IIS Protocol Control Registers . . . . . 15-147
15.6.5.2 IIS Protocol Status Register . . . . . 15-150
15.7 Service Request Generation . . . . . 15-153
15.8 Debug Behaviour . . . . . 15-153
15.9 Power, Reset and Clock . . . . . 15-153
15.10 Initialization and System Dependencies . . . . . 15-153
15.11 Registers . . . . . 15-153
15.11.1 Address Map . . . . . 15-157
15.11.2 Module Identification Registers . . . . . 15-157
15.11.3 Channel Control and Configuration Registers . . . . . 15-158
15.11.3.1 Channel Control Register . . . . . 15-158
15.11.3.2 Channel Configuration Register . . . . . 15-163
15.11.3.3 Kernel State Configuration Register . . . . . 15-164
15.11.3.4 Interrupt Node Pointer Register . . . . . 15-167
15.11.4 Protocol Related Registers . . . . . 15-168
15.11.4.1 Protocol Control Registers . . . . . 15-168
15.11.4.2 Protocol Status Register . . . . . 15-169
15.11.4.3 Protocol Status Clear Register . . . . . 15-170
15.11.5 Input Stage Register . . . . . 15-171
15.11.5.1 Input Control Registers . . . . . 15-171
15.11.6 Baud Rate Generator Registers . . . . . 15-177
15.11.6.1 Fractional Divider Register . . . . . 15-177
15.11.6.2 Baud Rate Generator Register . . . . . 15-178
15.11.6.3 Capture Mode Timer Register . . . . . 15-181
15.11.7 Transfer Control and Status Registers . . . . . 15-181
15.11.7.1 Shift Control Register . . . . . 15-181
15.11.7.2 Transmission Control and Status Register . . . . . 15-185
15.11.7.3 Flag Modification Registers . . . . . 15-191
15.11.8 Data Buffer Registers . . . . . 15-193
15.11.8.1 Transmit Buffer Locations . . . . . 15-193
15.11.8.2 Receive Buffer Registers RBUF0, RBUF1 . . . . . 15-194
15.11.8.3 Receive Buffer Registers RBUF, RBUFD, RBUFSR . . . . . 15-200
15.11.9 FIFO Buffer and Bypass Registers . . . . . 15-204
15.11.9.1 Bypass Registers . . . . . 15-204


Subject to Agreement on the Use of Product Information

15.11.9.2 General FIFO Buffer Control Registers . . . . . 15-207
15.11.9.3 Transmit FIFO Buffer Control Registers . . . . . 15-213
15.11.9.4 Receive FIFO Buffer Control Registers . . . . . 15-217
15.11.9.5 FIFO Buffer Data Registers . . . . . 15-222
15.11.9.6 FIFO Buffer Pointer Registers . . . . . 15-225
15.12 Interconnects . . . . . 15-226
15.12.1 USIC Module 0 Interconnects . . . . . 15-227

**16 Versatile Analog-to-Digital Converter (VADC)** . . . . . 16-1
16.1 Overview . . . . . 16-1
16.2 Introduction and Basic Structure . . . . . 16-4
16.3 Electrical Models . . . . . 16-9
16.4 Configuration of General Functions . . . . . 16-12
16.4.1 General Clocking Scheme and Control . . . . . 16-12
16.4.2 Register Access Control . . . . . 16-12
16.4.3 Priority Channel and Result Register Assignment . . . . . 16-13
16.5 Analog Module Activation and Control . . . . . 16-14
16.5.1 Analog Converter Control . . . . . 16-14
16.5.2 Converter Handling in Deep Sleep Mode . . . . . 16-15
16.5.3 Calibration . . . . . 16-16
16.5.4 Reference Voltage Selection . . . . . 16-17
16.5.5 Sigma-Delta-Loop Function . . . . . 16-18
16.6 Conversion Request Generation . . . . . 16-19
16.6.1 Queued Request Source Handling . . . . . 16-21
16.6.2 Channel Scan Request Source Handling . . . . . 16-24
16.7 Request Source Arbitration . . . . . 16-28
16.7.1 Arbiter Operation and Configuration . . . . . 16-29
16.7.2 Conversion Start Mode . . . . . 16-30
16.8 Analog Input Channel Configuration . . . . . 16-32
16.8.1 Channel Parameters . . . . . 16-32
16.8.2 Alias Feature . . . . . 16-33
16.8.3 Conversion Modes . . . . . 16-34
16.8.4 Compare with Standard Conversions (Limit Checking) . . . . . 16-35
16.8.5 Utilizing Fast Compare Mode . . . . . 16-36
16.8.6 Boundary Flag Control . . . . . 16-38
16.9 Conversion Scheduling . . . . . 16-39
16.10 Conversion Timing . . . . . 16-41
16.10.1 Timing Definition . . . . . 16-41
16.10.2 Compatible Timing Mode . . . . . 16-43
16.11 Conversion Result Handling . . . . . 16-44
16.11.1 Storage of Conversion Results . . . . . 16-44
16.11.2 Data Alignment . . . . . 16-46
16.11.3 Wait-for-Read Mode . . . . . 16-47


Subject to Agreement on the Use of Product Information

16.11.4 Result FIFO Buffer . . . . . 16-48
16.11.5 Result Event Generation . . . . . 16-49
16.11.6 Data Modification . . . . . 16-49
16.12 Synchronization of Conversions . . . . . 16-56
16.12.1 Synchronized Conversions for Parallel Sampling . . . . . 16-56
16.12.2 Equidistant Sampling . . . . . 16-59
16.13 Safety Features . . . . . 16-60
16.13.1 Broken Wire Detection . . . . . 16-60
16.13.2 Multiplexer Diagnostics . . . . . 16-61
16.14 External Multiplexer Control . . . . . 16-62
16.15 Service Request Generation . . . . . 16-64
16.16 Registers . . . . . 16-66
16.16.1 Module Identification . . . . . 16-71
16.16.2 System Registers . . . . . 16-72
16.16.3 General Registers . . . . . 16-75
16.16.4 Arbitration and Source Registers . . . . . 16-85
16.16.5 Channel Control Registers . . . . . 16-113
16.16.6 Result Registers . . . . . 16-118
16.16.7 Calbration Registers . . . . . 16-126
16.16.8 Miscellaneous Registers . . . . . 16-130
16.16.9 Service Request Registers . . . . . 16-142
16.17 Interconnects . . . . . 16-153
16.17.1 Product-Specific Configuration . . . . . 16-153
16.17.2 Analog Module Connections in the XMC1300 . . . . . 16-155
16.17.3 Digital Module Connections in the XMC1300 . . . . . 16-156

**17 Analog Comparator (ACMP) and Out of Range Comparator (ORC)** 17-1
17.1 Overview . . . . . 17-1
17.1.1 Features . . . . . 17-1
17.2 Analog Comparator (ACMP) . . . . . 17-1
17.3 Out of Range Comparator (ORC) . . . . . 17-3
17.4 Service Request Generation . . . . . 17-4
17.5 Debug Behavior . . . . . 17-4
17.6 Registers . . . . . 17-4
17.6.1 ORC Register . . . . . 17-5
17.6.2 ACMP Registers . . . . . 17-6
17.7 Interconnects . . . . . 17-12

**18 Temperature Sensor (TSE)** . . . . . 18-1
18.1 General Description . . . . . 18-1
18.2 Service Request Generation . . . . . 18-1
18.3 Registers . . . . . 18-2
18.3.1 Registers . . . . . 18-2


Subject to Agreement on the Use of Product Information

**19 Capture/Compare Unit 4 (CCU4)** . . . . . 19-1
19.1 Overview . . . . . 19-1
19.1.1 Features . . . . . 19-2
19.1.2 Block Diagram . . . . . 19-4
19.2 Functional Description . . . . . 19-6
19.2.1 CC4y Overview . . . . . 19-6
19.2.2 Input Selector . . . . . 19-8
19.2.3 Connection Matrix . . . . . 19-10
19.2.4 Starting/Stopping the Timer . . . . . 19-12
19.2.5 Counting Modes . . . . . 19-13
19.2.5.1 Calculating the PWM Period and Duty Cycle . . . . . 19-14
19.2.5.2 Updating the Period and Duty Cycle . . . . . 19-15
19.2.5.3 Edge Aligned Mode . . . . . 19-19
19.2.5.4 Center Aligned Mode . . . . . 19-20
19.2.5.5 Single Shot Mode . . . . . 19-21
19.2.6 Active/Passive Rules . . . . . 19-22
19.2.7 External Events Control . . . . . 19-22
19.2.7.1 External Start/Stop . . . . . 19-23
19.2.7.2 External Counting Direction . . . . . 19-25
19.2.7.3 External Gating Signal . . . . . 19-27
19.2.7.4 External Count Signal . . . . . 19-27
19.2.7.5 External Load . . . . . 19-28
19.2.7.6 External Capture . . . . . 19-29
19.2.7.7 Capture Extended Read Back Mode . . . . . 19-35
19.2.7.8 External Modulation . . . . . 19-38
19.2.7.9 TRAP Function . . . . . 19-41
19.2.7.10 Status Bit Override . . . . . 19-43
19.2.8 Multi-Channel Control . . . . . 19-44
19.2.9 Timer Concatenation . . . . . 19-47
19.2.10 PWM Dithering . . . . . 19-52
19.2.11 Prescaler . . . . . 19-57
19.2.11.1 Normal Prescaler Mode . . . . . 19-58
19.2.11.2 Floating Prescaler Mode . . . . . 19-58
19.2.12 CCU4 Usage . . . . . 19-60
19.2.12.1 PWM Signal Generation . . . . . 19-60
19.2.12.2 Prescaler Usage . . . . . 19-62
19.2.12.3 PWM Dither . . . . . 19-64
19.2.12.4 Capture Mode Usage . . . . . 19-67
19.3 Service Request Generation . . . . . 19-73
19.4 Debug Behavior . . . . . 19-77
19.5 Power, Reset and Clock . . . . . 19-77
19.5.1 Clocks . . . . . 19-77
19.5.2 Power . . . . . 19-78


Subject to Agreement on the Use of Product Information

19.6 Initialization and System Dependencies . . . . . 19-78
19.6.1 Initialization Sequence . . . . . 19-79
19.6.2 System Dependencies . . . . . 19-79
19.7 Registers . . . . . 19-80
19.7.1 Global Registers . . . . . 19-86
19.7.2 Slice (CC4y) Registers . . . . . 19-101
19.8 Interconnects . . . . . 19-137
19.8.1 CCU40 pins . . . . . 19-137

**20 Capture/Compare Unit 8 (CCU8)** . . . . . 20-1
20.1 Overview . . . . . 20-1
20.1.1 Features . . . . . 20-2
20.1.2 Block Diagram . . . . . 20-5
20.2 Functional Description . . . . . 20-7
20.2.1 Overview . . . . . 20-7
20.2.2 Input Selector . . . . . 20-9
20.2.3 Connection Matrix . . . . . 20-11
20.2.4 Start/Stop Control . . . . . 20-13
20.2.5 Counting Modes . . . . . 20-14
20.2.5.1 Calculating the PWM Period and Duty Cycle . . . . . 20-15
20.2.5.2 Updating the Period and Duty Cycle . . . . . 20-16
20.2.5.3 Edge Aligned Mode . . . . . 20-23
20.2.5.4 Center Aligned Mode . . . . . 20-23
20.2.5.5 Single Shot Mode . . . . . 20-24
20.2.6 Active/Passive Rules . . . . . 20-25
20.2.7 Compare Modes . . . . . 20-26
20.2.7.1 Edge Aligned Compare Modes . . . . . 20-30
20.2.7.2 Center Aligned Compare Modes . . . . . 20-34
20.2.8 External Events Control . . . . . 20-37
20.2.8.1 External Start/Stop . . . . . 20-37
20.2.8.2 External Counting Direction . . . . . 20-40
20.2.8.3 External Gating Signal . . . . . 20-41
20.2.8.4 External Count Signal . . . . . 20-42
20.2.8.5 External Load . . . . . 20-43
20.2.8.6 External Capture . . . . . 20-44
20.2.8.7 Capture Extended Read Back Mode . . . . . 20-50
20.2.8.8 External Modulation . . . . . 20-53
20.2.8.9 Trap Function . . . . . 20-55
20.2.8.10 Status Bit Override . . . . . 20-58
20.2.9 Multi-Channel Support . . . . . 20-59
20.2.10 Timer Concatenation . . . . . 20-64
20.2.11 Output Parity Checker . . . . . 20-69
20.2.12 PWM Dithering . . . . . 20-73


Subject to Agreement on the Use of Product Information

20.2.13 Prescaler . . . . . 20-77
20.2.13.1 Normal Prescaler Mode . . . . . 20-78
20.2.13.2 Floating Prescaler Mode . . . . . 20-78
20.2.14 CCU8 Usage . . . . . 20-80
20.2.14.1 PWM Signal Generation . . . . . 20-80
20.2.14.2 Prescaler Usage . . . . . 20-82
20.2.14.3 PWM Dither . . . . . 20-85
20.2.14.4 Capture Mode Usage . . . . . 20-87
20.2.14.5 Parity Checker Usage . . . . . 20-93
20.3 Service Request Generation . . . . . 20-96
20.4 Debug Behavior . . . . . 20-99
20.5 Power, Reset and Clock . . . . . 20-99
20.5.1 Clocks . . . . . 20-100
20.5.2 Power . . . . . 20-100
20.6 Initialization and System Dependencies . . . . . 20-101
20.6.1 Initialization Sequence . . . . . 20-101
20.6.2 System Dependencies . . . . . 20-101
20.7 Registers . . . . . 20-103
20.7.1 Global Registers . . . . . 20-111
20.7.2 Slice (CC8y) Registers . . . . . 20-131
20.8 Interconnects . . . . . 20-178
20.8.1 CCU80 Pins . . . . . 20-178

**21 Position Interface Unit (POSIF)** . . . . . 21-1
21.1 Overview . . . . . 21-1
21.1.1 Features . . . . . 21-2
21.1.2 Block Diagram . . . . . 21-3
21.2 Functional Description . . . . . 21-4
21.2.1 Overview . . . . . 21-4
21.2.2 Function Selector . . . . . 21-6
21.2.3 Hall Sensor Control . . . . . 21-7
21.2.4 Quadrature Decoder Control . . . . . 21-13
21.2.4.1 Quadrature Clock and Direction decoding . . . . . 21-16
21.2.4.2 Index Control . . . . . 21-17
21.2.5 Stand-Alone Multi-Channel Mode . . . . . 21-18
21.2.6 Synchronous Start . . . . . 21-18
21.2.7 Using the POSIF . . . . . 21-19
21.2.7.1 Hall Sensor Mode Usage . . . . . 21-19
21.2.7.2 Quadrature Decoder Mode usage . . . . . 21-21
21.2.7.3 Stand-alone Multi-Channel Mode . . . . . 21-27
21.3 Service Request Generation . . . . . 21-28
21.3.1 Hall Sensor Mode flags . . . . . 21-28
21.3.2 Quadrature Decoder Flags . . . . . 21-30


Subject to Agreement on the Use of Product Information

21.4 Debug Behavior . . . . . 21-32
21.5 Power, Reset and Clock . . . . . 21-33
21.5.1 Clocks . . . . . 21-33
21.5.2 Power . . . . . 21-33
21.6 Initialization and System Dependencies . . . . . 21-33
21.6.1 Initialization . . . . . 21-33
21.6.2 System Dependencies . . . . . 21-34
21.7 Registers . . . . . 21-36
21.7.1 Global registers . . . . . 21-38
21.7.2 Hall Sensor Mode Registers . . . . . 21-46
21.7.3 Multi-Channel Mode Registers . . . . . 21-48
21.7.4 Quadrature Decoder Registers . . . . . 21-53
21.7.5 Interrupt Registers . . . . . 21-54
21.8 Interconnects . . . . . 21-61
21.8.1 POSIF0 pins . . . . . 21-62

**22 Brightness and Color Control Unit (BCCU)** . . . . . 22-1
22.1 Overview . . . . . 22-1
22.1.1 Features . . . . . 22-1
22.2 Functional Description . . . . . 22-1
22.2.1 Channel Structure . . . . . 22-2
22.2.2 Exponential Dimming . . . . . 22-3
22.2.3 Linear Color Walk . . . . . 22-13
22.2.4 Sigma-Delta Modulator . . . . . 22-14
22.2.5 Packer . . . . . 22-14
22.2.6 Global Trigger Control . . . . . 22-15
22.2.7 Trap Control . . . . . 22-17
22.3 Power, Reset and Clock . . . . . 22-18
22.4 Service Request Generation . . . . . 22-19
22.5 Debug Behaviour . . . . . 22-20
22.6 Initialization . . . . . 22-20
22.7 Digital-to-Analog Converter . . . . . 22-22
22.8 Registers . . . . . 22-23
22.8.1 Global Registers . . . . . 22-25
22.8.2 Channel Registers . . . . . 22-40
22.8.3 Dimming Engine Registers . . . . . 22-46
22.9 Interconnects . . . . . 22-49

**23 General Purpose I/O Ports (Ports)** . . . . . 23-1
23.1 Overview . . . . . 23-1
23.1.1 Features . . . . . 23-1
23.1.2 Block Diagram . . . . . 23-2
23.1.3 Definition of Terms . . . . . 23-3
23.2 GPIO and Alternate Functions . . . . . 23-4


Subject to Agreement on the Use of Product Information

23.2.1 Input Operation . . . 23-4
23.2.2 Output Operation . . . 23-5
23.3 Hardware Controlled I/Os . . . 23-6
23.4 Power Saving Mode Operation . . . 23-7
23.5 Analog Ports . . . 23-8
23.6 Power, Reset and Clock . . . 23-9
23.7 Initialization and System Dependencies . . . 23-10
23.8 Registers . . . 23-12
23.8.1 Port Input/Output Control Registers . . . 23-15
23.8.2 Pad Hysteresis Control Register . . . 23-19
23.8.3 Pin Function Decision Control Register . . . 23-24
23.8.4 Port Output Register . . . 23-27
23.8.5 Port Output Modification Register . . . 23-30
23.8.6 Port Input Register . . . 23-33
23.8.7 Port Pin Power Save Register . . . 23-35
23.8.8 Port Pin Hardware Select Register . . . 23-38
23.9 Package Pin Summary . . . 23-41
23.10 Port I/O Functions . . . 23-44
23.10.1 Port Pin for Boot Modes . . . 23-44
23.10.2 Port I/O Function Description . . . 23-45
23.10.3 Hardware Controlled I/O Function Description . . . 23-46

**24 Boot and Startup** . . . 24-1
24.1 Startup Sequence and System Dependencies . . . 24-2
24.1.1 Power-Up . . . 24-2
24.1.2 System Reset Release . . . 24-2
24.1.3 Startup Software (SSW) Execution . . . 24-2
24.1.3.1 Clock system handling by SSW . . . 24-3
24.1.4 Configuration of Special System Functions as part of User code initialization 24-3
24.1.5 Configuration of Clock System and Miscellaneous Functions . . . 24-4
24.2 Start-up Modes . . . 24-5
24.2.1 Start-up modes in XMC1300 . . . 24-5
24.2.1.1 User productive mode . . . 24-5
24.2.1.2 User mode with debug enabled . . . 24-6
24.2.1.3 User mode with debug enabled and Halt After Reset (HAR) . . . 24-6
24.2.1.4 Standard Bootstrap Loader modes . . . 24-6
24.2.1.5 Bootstrap Loader modes with time-out . . . 24-6
24.2.2 Boot Mode Index (BMI) . . . 24-8
24.2.3 Start-up mode selection . . . 24-9
24.2.3.1 BMI handling by SSW . . . 24-9
24.2.3.2 Debug system handling . . . 24-10
24.3 Data in Flash for SSW and User SW . . . 24-10


Subject to Agreement on the Use of Product Information

**25 Bootstrap Loaders (BSL) and User Routines** . . . . . 25-1
25.1 ASC (UART) Bootstrap Loader . . . . . 25-1
25.1.1 Pin usage . . . . . 25-1
25.1.2 ASC BSL execution flow . . . . . 25-1
25.1.2.1 ASC BSL entry check sequence . . . . . 25-1
25.1.2.2 ASC BSL download sequence . . . . . 25-6
25.1.3 ASC BSL protocol data definitions . . . . . 25-7
25.2 SSC Bootstrap loader . . . . . 25-9
25.3 Firmware routines available for the user . . . . . 25-11
25.3.1 Erase Flash Page . . . . . 25-12
25.3.2 Erase, Program & Verify Flash Page . . . . . 25-13
25.3.3 Request BMI installation . . . . . 25-13
25.3.4 Calculate chip temperature . . . . . 25-13
25.3.5 Erase Flash Sector . . . . . 25-14
25.3.6 Program & Verify Flash Block . . . . . 25-14
25.3.7 Calculate target level for temperature comparison . . . . . 25-14
25.4 Data in Flash used by the User Routines . . . . . 25-15

**26 Debug System (DBG)** . . . . . 26-1
26.1 Overview . . . . . 26-1
26.1.1 Features . . . . . 26-2
26.1.2 Block Diagram . . . . . 26-2
26.2 Debug System Operation . . . . . 26-2
26.2.1 System Control Space (SCS) . . . . . 26-3
26.2.2 Data Watchpoint and Trace (DWT) . . . . . 26-3
26.2.3 Break Point Unit (BPU) . . . . . 26-3
26.2.4 ROM Table . . . . . 26-4
26.2.5 Debug tool interface access - SWD . . . . . 26-4
26.2.5.1 SWD based transfers . . . . . 26-4
26.2.5.2 SWD based errors . . . . . 26-5
26.2.6 Debug tool interface access - Single Pin Debug (SPD) . . . . . 26-6
26.2.7 Debug accesses and Flash protection . . . . . 26-9
26.2.8 Halt after reset . . . . . 26-9
26.2.8.1 HAR . . . . . 26-9
26.2.8.2 Warm Reset . . . . . 26-11
26.2.9 Halting Debug and Peripheral Suspend . . . . . 26-12
26.2.10 Debug System based processor wake-up . . . . . 26-14
26.2.11 Debug Access Server (DAS) . . . . . 26-14
26.2.12 Debug Signals . . . . . 26-15
26.2.12.1 Internal pull-up and pull-down on SWD/SPD pins . . . . . 26-15
26.2.13 Reset . . . . . 26-15
26.3 Debug System Power Save Operation . . . . . 26-15
26.4 Service Request Generation . . . . . 26-16


Subject to Agreement on the Use of Product Information

26.5 Debug behavior . . . . . 26-16
26.6 Power, Reset and Clock . . . . . 26-16
26.6.1 Power management . . . . . 26-17
26.6.2 Debug System reset . . . . . 26-17
26.6.3 Debug System Clocks . . . . . 26-17
26.7 Initialization and System Dependencies . . . . . 26-17
26.7.1 ID Code . . . . . 26-17
26.7.2 ROM Table . . . . . 26-18
26.8 Debug System Registers . . . . . 26-19
26.8.1 DFSR - Debug Fault Status Register . . . . . 26-20
26.8.2 DHCSR - Debug Halting Control and Status Register . . . . . 26-21
26.8.3 DCRSR - Debug Core Register Selector Register . . . . . 26-27
26.8.4 DCRDR - Debug Core Register Data Register . . . . . 26-29
26.8.5 DEMCR - Debug Exception and Monitor Control Register . . . . . 26-29
26.8.6 DWT_CTRL - Data Watchpoint Control Register . . . . . 26-31
26.8.7 DWT_PCSR - Program Counter Sample Register . . . . . 26-31
26.8.8 DWT_COMPx - DWT Comparator register . . . . . 26-32
26.8.9 DWT_MASKx - DWT Comparator Mask Register . . . . . 26-33
26.8.10 DWT_FUNCTIONx - Comparator Function Register . . . . . 26-34
26.8.11 BP_CTRL - Breakpoint Control Register . . . . . 26-35
26.8.12 Breakpoint Comparator Registers . . . . . 26-36


Subject to Agreement on the Use of Product Information

# 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 XMC1300 series functional units and their interaction.

The manual describes the functionality of the superset device of the XMC1300 microcontroller series. For the available functionality (features) of a specific XMC1300 derivative (derivative device), please refer to the respective Data Sheet. For simplicity, the various device types are referenced by the collective term XMC1300 throughout this manual.

### XMC1000 Family User Documentation

The set of user documentation includes:

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

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

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

Please refer to **http://www.infineon.com/xmc1000** 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 XMC1300 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 $1024 \times 1024$ 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.


Subject to Agreement on the Use of Product Information

- 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.<br>***Note: ARM® Cortex M0 processor does not support different privilege levels. Only Privileged (Supervisor) Mode is supported in*** XMC1000 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. |


Subject to Agreement on the Use of Product Information

**Table 2 Register Access Modes** (cont'd)

| Symbol | Description |
|---|---|
| 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 |


Subject to Agreement on the Use of Product Information

| | |
|---|---|
| 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 |
| LEDTS | LED and Touch Sense Control Unit |
| 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 |


Subject to Agreement on the Use of Product Information

| | |
|---|---|
| USIC | Universal Serial Interface Channel |
| VADC | Versatile Analog-to-Digital Converter |
| WDT | Watchdog Timer |


Subject to Agreement on the Use of Product Information

# Introduction


Subject to Agreement on the Use of Product Information

# 1 Introduction

The XMC1300 series belongs to the XMC1000 Family of industrial microcontrollers based on the ARM Cortex-M0 processor core. The XMC1300 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 XMC1300 microcontroller pursue successful hardware and software concepts, which have been established in Infineon microcontroller families.

## 1.1 Overview

The XMC1300 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 XMC1300 series devices:

**CPU Subsystem**

- CPU Core
  - High Performance 32-bit Cortex-M0 CPU
  - Most of 16-bit Thumb instruction set
  - Subset of 32-bit Thumb2 instruction set
  - High code density with 32-bit performance
  - Single cycle 32-bit hardware multiplier
  - System timer (SysTick) for Operating System support
  - Ultra low power consumption
- Nested Vectored Interrupt Controller (NVIC)
- Event Request Unit (ERU) for programmable processing of external and internal service requests
- MATH Co-processor (MATH), consists of a CORDIC unit for trigonometric calculation and a division unit

**On-Chip Memories**

- 8 kbytes on-chip ROM
- 16 kbytes on-chip high-speed SRAM
- up to 200 kbytes on-chip Flash program and data memory

**Communication Peripherals**

- Two Universal Serial Interface Channels (USIC), usable as UART, double-SPI, quad-SPI, IIC, IIS and LIN interfaces


Subject to Agreement on the Use of Product Information

### Analog Frontend Peripherals

- A/D Converters, up to 12 channels, includes 2 sample and hold stages and a fast 12-bit analog to digital converter with adjustable gain
- Up to 8 channels of out of range comparators (ORC)
- Up to 3 fast analog comparators (ACMP)
- Temperature Sensor (TSE)

### Industrial Control Peripherals

- Capture/Compare Units 4 (CCU4) for use as general purpose timers
- Capture/Compare Units 8 (CCU8) for motor control and power conversion
- Position Interfaces (POSIF) for hall and quadrature encoders and motor positioning
- Brightness and Colour Control Unit (BCCU), for LED color and dimming application

### System Control

- Window Watchdog Timer (WDT) for safety sensitive applications
- Real Time Clock module with alarm support (RTC)
- 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 4 hardware breakpoints
- A watchpoint unit (DWT) supporting up to 2 watchpoints

### Packages Information

- PG-VQFN-40
- PG-VQFN-24
- PG-TSSOP-16
- PG-TSSOP-38
- PG-TSSOP-16

*Note: For details about package availability for a particular derivative please check the datasheet.*


Subject to Agreement on the Use of Product Information

### 1.1.1 Block Diagram

The diagram below shows the functional blocks and their basic connectivity within the XMC1300 System.

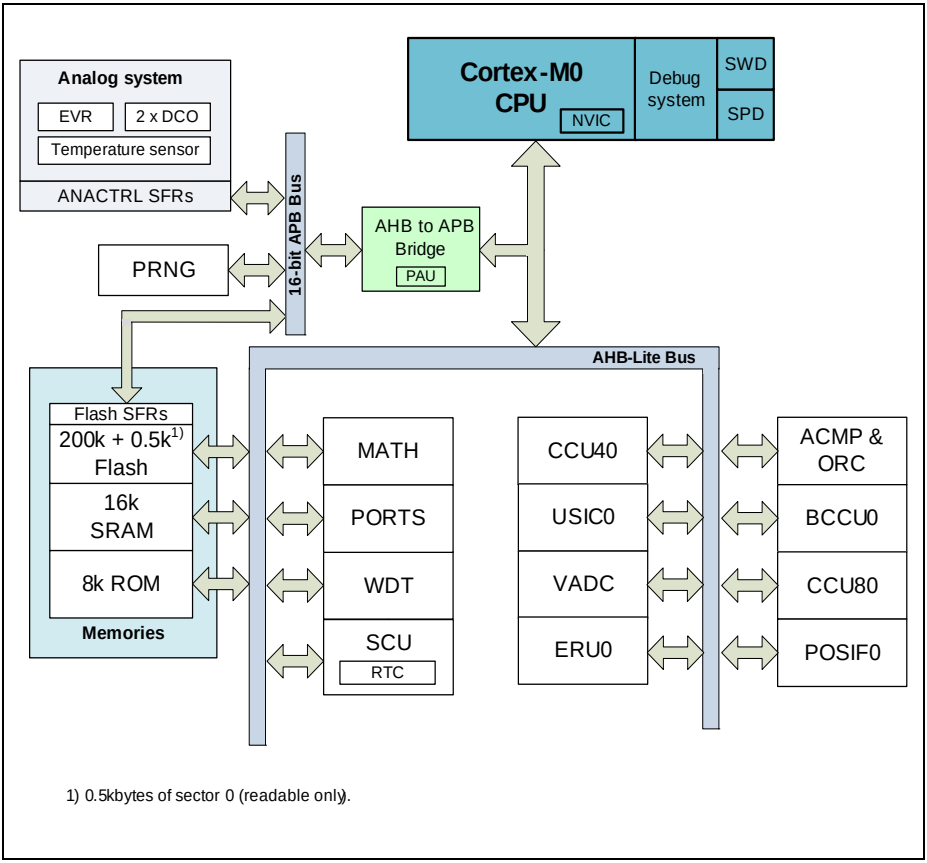



**Figure 1-1 XMC1300 Functional diagram**

## 1.2 Core Processing Units

The XMC1300 system core consists of the CPU and the memory interface blocks for memories.

### 1.2.1 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 Thumb instrcution set and includes Thumb-2 technology. This provides the exceptional perfomance expected of a modern 32-bit architecture, with a higher code density than other 8-bit and 16-bit microcontrollers.

### 1.2.2 Programmable Multiple Priority Interrupt System (NVIC)

The XMC1300 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.

### 1.2.3 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 CORDIC (COrdinate Rotation DIgital Computer) Coprocessor for computation of trigonometric, linear or hyperbolic functions.

## 1.3 System Units

The XMC1300 controllers provide a number of system resources designed around the CPU.

### 1.3.1 Memories

**8 kbytes of ROM (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.

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

**16 kbytes of on-chip code RAM (SRAM)** are provided to store 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.


Subject to Agreement on the Use of Product Information

### 1.3.2 Watchdog Timer (WDT)

The main purpose of the Window Watchdog Timer is to improve the system integrity. WDT triggers the system reset or other corrective action like e.g. an interrupt if the main program, due to some fault condition, neglects to regularly service the watchdog. The intention is to bring the system back from the unresponsive state into normal operation.

### 1.3.3 Real Timer 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..

### 1.3.4 System Control unit (SCU)

The System Control Unit (SCU) handles all system cotnrol tasks besides the debug related tasks. All functions are tightly coupled and thus, they are conveniently handled by one unit, SCU. It consists of the Power Control Unit (PCU), Reset Control Unit, Clock Control Unit (CCU) and the Miscellaneous Control Unit (GCU).

The CCU generates the Main clock (MCLK) and the fast Peripheral clock (PCLK) using the 64MHz DCO1 oscillator. The PCU has a Embedded Voltage Regulator (EVR) that is used to generate the core voltage. It also provides voltage monitoring detectors to secure system performance under critical condition (eg. brownout).

### 1.3.5 Pseudo Random Bit Generator (PRNG)

The pseudo random bit generator (PRNG) provides random data with fast generation times.

## 1.4 Peripherals Units

XMC1300 offers a set of on-chip peripherals to support industrial applications.

#### 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 which 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.

#### Analog to Digital Converter (VADC)

The Versatile Analog-to-Digital Converter module consists of an independent kernels which operate according to the successive approximation principle (SAR). The resolution is programmable from 8 to 12bit.


Subject to Agreement on the Use of Product Information

The kernel provides a versatile state machine allowing complex measurement sequences. The kernels 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 which input voltage is higher. One of the input can either be the internal reference voltage or from external pin. A low power mode is available to help to reduce the total power comsumption.

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 (TSE)

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.

### 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.


Subject to Agreement on the Use of Product Information

### Position Interface Unit (POSIF)

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

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

### 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.

### General Purpose I/O Ports (PORTS)

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.

## 1.5 Debug Unit

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 XMC1300. The user software running on the XMC1300 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, e.g. 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.


Subject to Agreement on the Use of Product Information

# CPU Subsystem


Subject to Agreement on the Use of Product Information

# 2 Central Processing Unit (CPU)

XMC1300 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] ARMv6-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 ARMv6-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.


Subject to Agreement on the Use of Product Information

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 Thumb2 instruction set.

**Nested Vectored Interrupt Controller**

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

**Debug Solution**

The XMC1300 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.


Subject to Agreement on the Use of Product Information

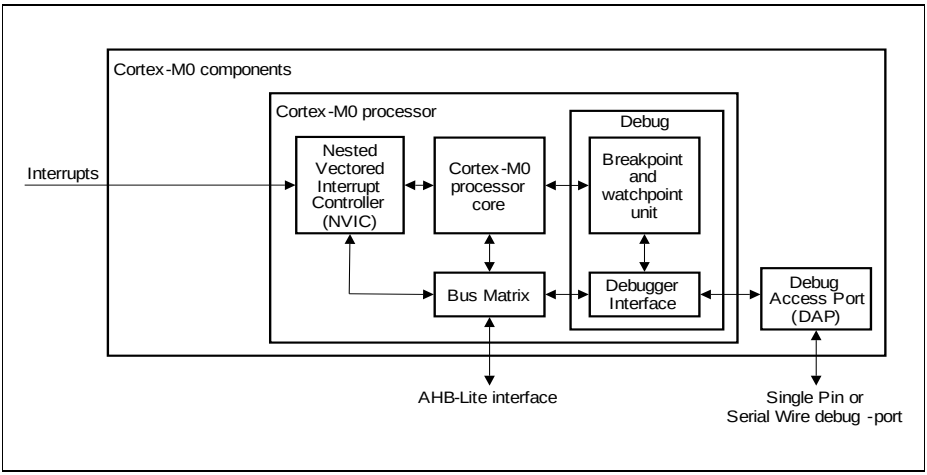



**Figure 2-1 Cortex-M0 Block Diagram**

### 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 2-1 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

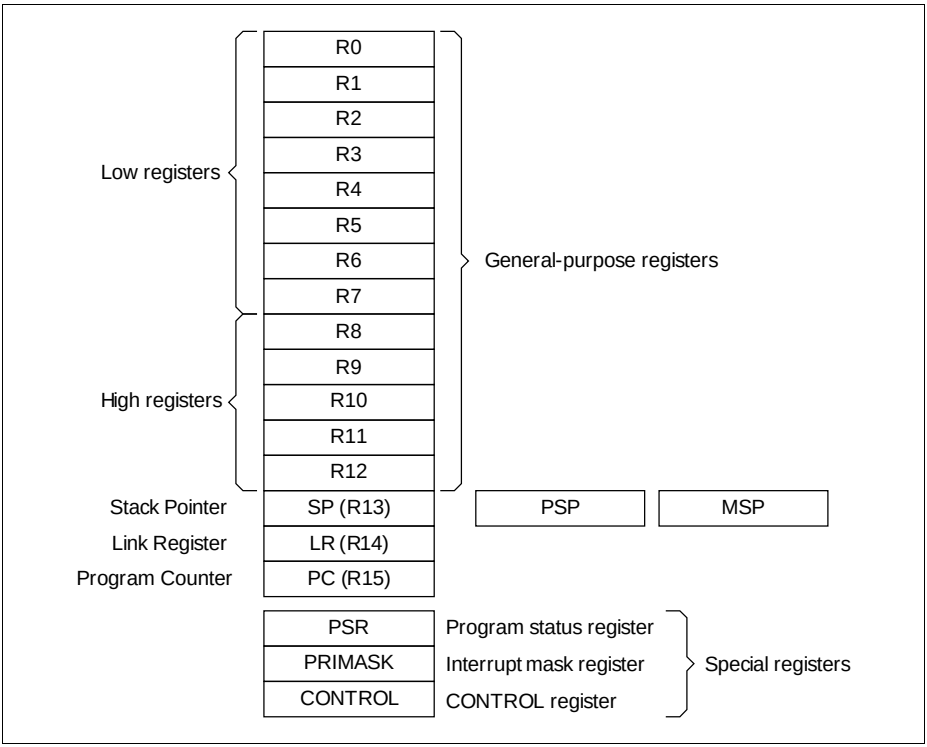



**Figure 2-2 Core registers**

The processor core registers are:

**Table 2-2 Core register set summary**

| Name | Type[1)] | Reset value | Description |
|---|---|---|---|
| R0-R12 | rw | Unknown | General-purpose registers on **Page 2-6** |
| MSP | rw | See description | Stack Pointer on **Page 2-6** |
| PSP | rw | Unknown | Stack Pointer on **Page 2-6** |
| LR | rw | Unknown | Link Register on **Page 2-7** |
| PC | rw | See description | Program Counter on **Page 2-8** |
| PSR | rw | Unknown | Program Status Register on **Page 2-8** |

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

| Name | Type[1)] | Reset value | Description |
|---|---|---|---|
| APSR | rw | Unknown | Application Program Status Register on **Page 2-9** |
| IPSR | r | $00000000_H$ | Interrupt Program Status Register on **Page 2-10** |
| EPSR | r | Unknown | Execution Program Status Register on **Page 2-12** |
| PRIMASK | rw | $00000000_H$ | Priority Mask Register on **Page 2-13** |
| CONTROL | rw | $00000000_H$ | CONTROL Register on **Page 2-14** |

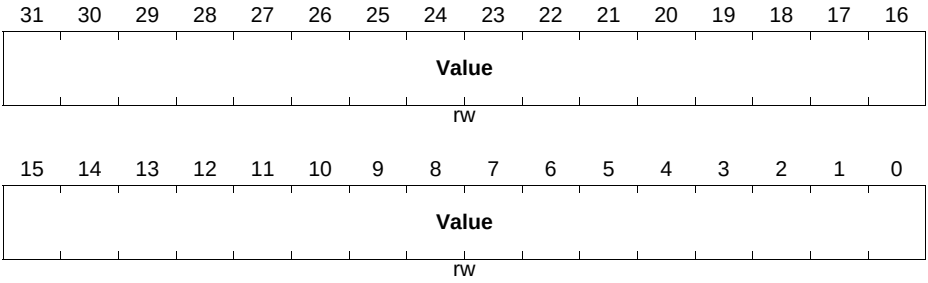
1) Describes access type during program execution in thread mode and handler mode. Debug access can differ.

### 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 ARMv6-M Architecture Reference Manual **[2]**.*

**Rx (x=0-12)**
**General-purpose register Rx** **Reset Value: $XXXXXXXX_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **Value** | [31:0] | rw | **Content of Register** |

### Stack Pointer

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

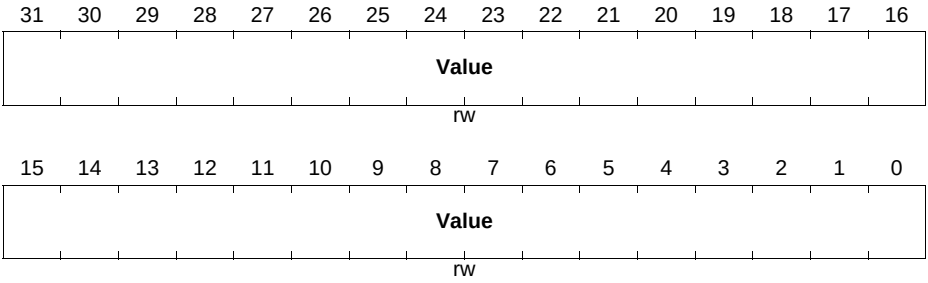
- 0 = Main Stack Pointer (MSP). This is the reset value.

- 1 = Process Stack Pointer (PSP).

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

*Note: For information on how to program the core registers, please refer to the ARMv6-M Architecture Reference Manual [2].*

**SP**
**Stack Pointer** **Reset Value: $00000000_H$**



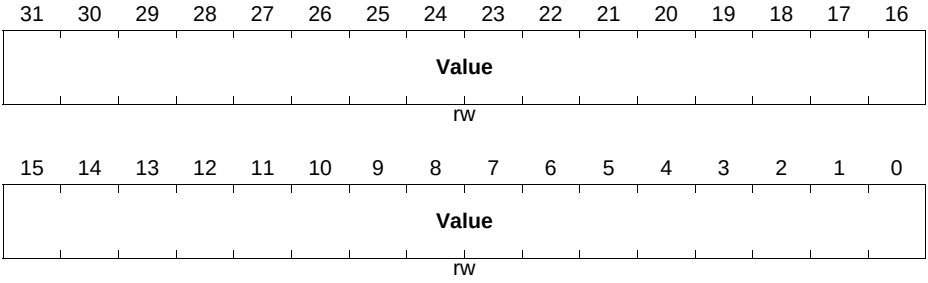

| Field | Bits | Type | Description |
|---|---|---|---|
| **Value** | [31:0] | rw | **Content of Register** |

**Link Register**

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

*Note: For information on how to program the core registers, please refer to the ARMv6-M Architecture Reference Manual [2].*

**LR**
**Link Register** **Reset Value: $XXXXXXXX_H$**



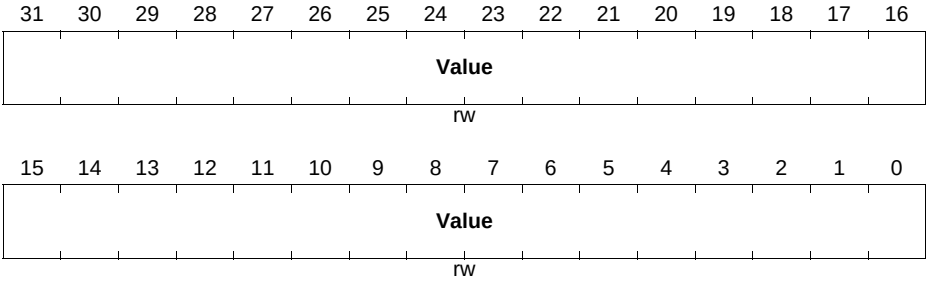
| Field | Bits | Type | Description |
|---|---|---|---|
| **Value** | [31:0] | rw | **Content of Register** |

### Program Counter

The Program Counter (PC) is register R15. It contains the current program address. On reset, the processor loads the PC with the value of the reset vector, which is at address $00000004_H$. 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 ARMv6-M Architecture Reference Manual **[2]**.*

**PC**
**Program Counter** **Reset Value: $00000004_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **Value** | [31:0] | rw | **Content of Register** |

### Program Status Register

The Program Status Register (PSR) combines:

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

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

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

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


Subject to Agreement on the Use of Product Information

The PSR combinations and attributes are:

**Table 2-3 PSR register combinations**

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

1) The processor ignores writes to the IPSR bits.

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

### Application Program Status Register

The APSR contains the current state of the condition flags from previous instruction executions. See the register summary in **Table 2-2** 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 | rw | | | | | | | | | | | |

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

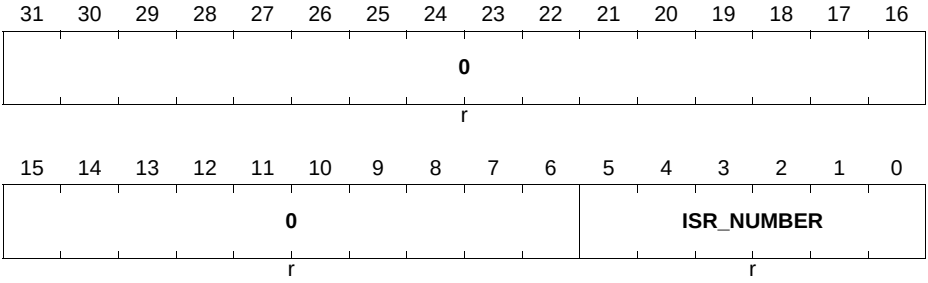
| Field | Bits | Type | Description |
|---|---|---|---|
| **N** | 31 | rw | **Negative flag** |
| **Z** | 30 | rw | **Zero flag** |
| **C** | 29 | rw | **Carry or borrow flag** |
| **V** | 28 | rw | **Overflow flag** |
| **0** | [27:0] | r | **Reserved** |


Subject to Agreement on the Use of Product Information

### Interrupt Program Status Register

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

**IPSR**
**Interrupt Program Status Register** **Reset Value: $00000000_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | [31:6] | r | **Reserved** |

| Field | Bits | Type | Description |
|---|---|---|---|
| **ISR_NUMBER** | [5:0] | r | **Number of the current exception**<br>$0_D$ Thread mode<br>$1_D$ Reserved<br>$2_D$ Reserved<br>$3_D$ HardFault<br>$4_D$ Reserved<br>$5_D$ Reserved<br>$6_D$ Reserved<br>$7_D$ Reserved<br>$8_D$ Reserved<br>$9_D$ Reserved<br>$10_D$ Reserved<br>$11_D$ SVCall<br>$12_D$ Reserved<br>$13_D$ Reserved<br>$14_D$ PendSV<br>$15_D$ SysTick<br>$16_D$ IRQ0<br>...<br>$47_D$ IRQ31<br>$48_D$-$63_D$ Reserved<br>See Exception types in **Section 2.5.2** for more information. |


Subject to Agreement on the Use of Product Information

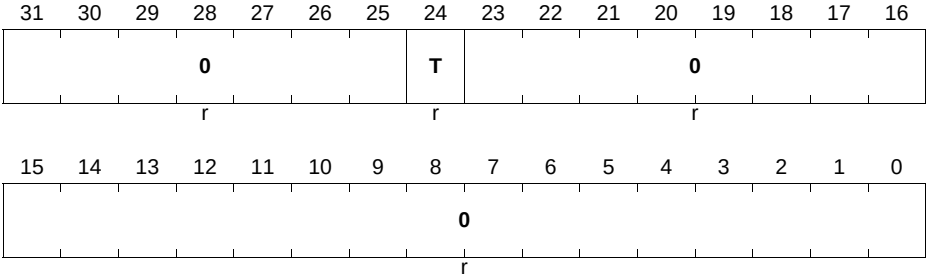
### Execution Program Status Register

The EPSR contains the Thumb state bit.

See the register summary in **Table 2-2** 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 in **Section 2.5.6**.

**EPSR**
**Execution Program Status Register** **Reset Value: XXXXXXXX$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | [31:25] | r | **Reserved** |
| **T** | 24 | r | **Thumb state bit**<br>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 in **Section 2.6.1** for more information.

### Exception mask registers

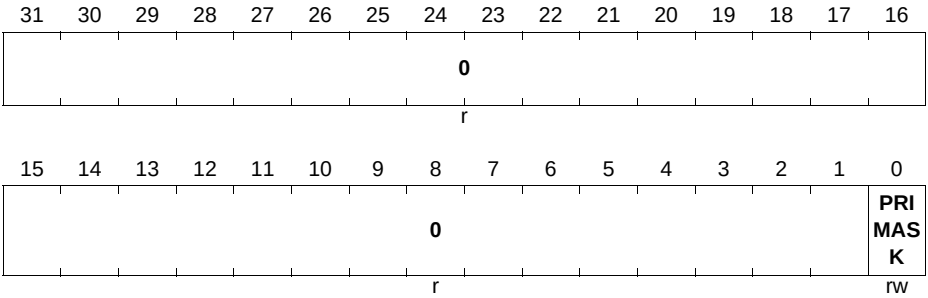
The exception mask registers disable the handling of exceptions by the processor. Disable exceptions where they might impact on timing critical tasks or code sequences requiring atomicity.

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.

### Priority Mask Register

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

**PRIMASK**
**Priority Mask Register** **Reset Value: $00000000_H$**

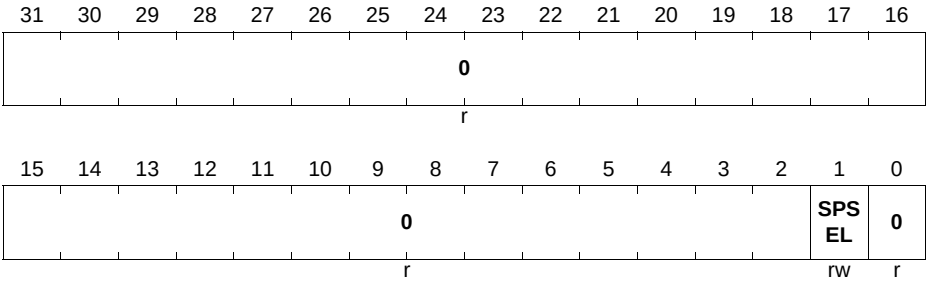



| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | [31:1] | r | **Reserved** |
| **PRIMASK** | 0 | rw | **Priority Mask**<br>$0_B$ No effect.<br>$1_B$ Prevents the activation of all exceptions with configurable priority. |

### CONTROL Register

The CONTROL register controls the stack used when the processor is in Thread mode. See the register summary in **Table 2-2** 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**<br>This bit defines the current stack. In Handler mode, this bit reads as zero and ignores writes.<br>$0_B$ MSP is the current stack pointer<br>$1_B$ 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 on **Section 2.5.6.1** and Exception return on **Section 2.5.6.2** 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 in **Section 2.3.1** 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.


Subject to Agreement on the Use of Product Information

*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 in **Section 2.7.3**
- CMSIS Functions in **Section 2.2.7**
- Accessing CPU Registers using CMSIS in Interrupt System chapter
- NVIC programming hints in Interrupt System chapter

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

### 2.2.7 CMSIS Functions

ISO/IEC C code cannot directly access some Cortex-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 2-4 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:


Subject to Agreement on the Use of Product Information

**Table 2-5 CMSIS functions to access the special registers**

| Special register | Access | CMSIS function |
|---|---|---|
| PRIMASK | Read | uint32_t __get_PRIMASK (void) |
| | Write | void __set_PRIMASK (uint32_t value) |
| 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) |


Subject to Agreement on the Use of Product Information

## 2.3 Memory Model

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

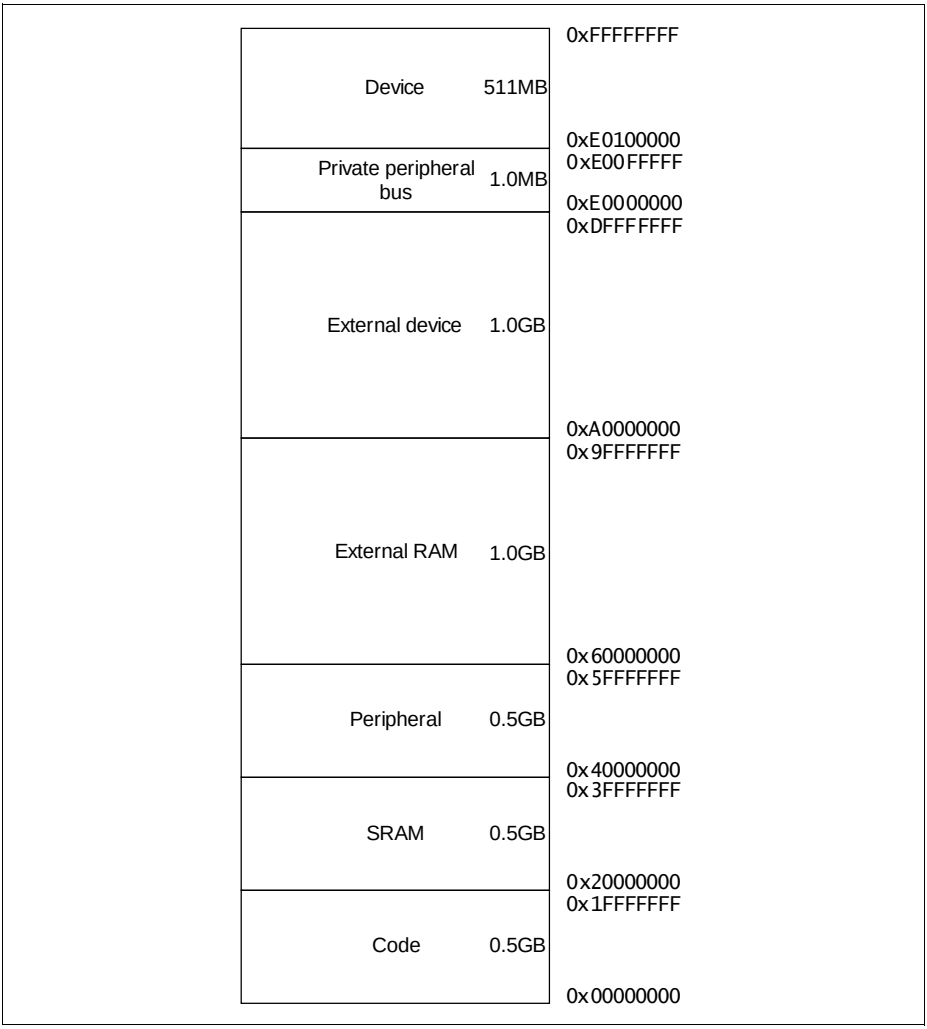



**Figure 2-3 Memory map**


Subject to Agreement on the Use of Product Information

The processor reserves regions of the Private peripheral bus (PPB) address range for core peripheral registers, see About the Private Peripherals in **Section 2.8.1**.

### 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 in **Section 2.3.4**.

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 2-4**.


Subject to Agreement on the Use of Product Information

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

**Figure 2-4 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 2-6 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. |


Subject to Agreement on the Use of Product Information

**Table 2-6 Memory access behavior** (cont'd)

| Address range | Memory region | Memory type[1)] | XN[1)] | Description |
|---|---|---|---|---|
| 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 in **Section 2.3.1** 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 in **Section 2.3.2** 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.


Subject to Agreement on the Use of Product Information

### 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. **Section 2.3.5.1** 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 2-5**.

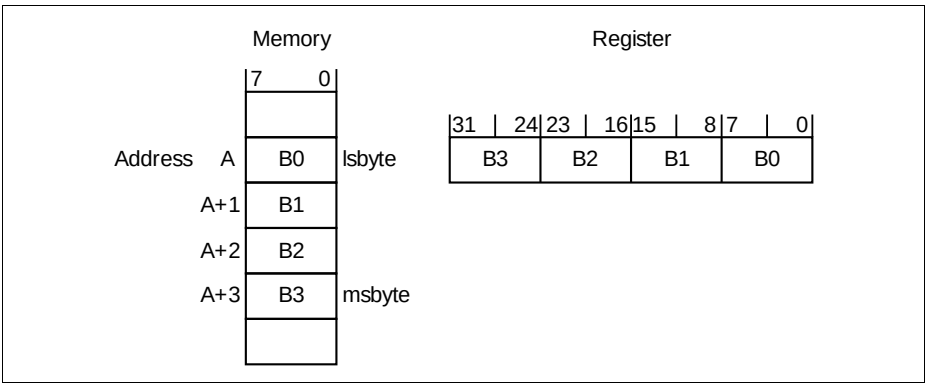



**Figure 2-5 Little-endian format (Example)**

## 2.4 Instruction Set

Table 2-7 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 2-7 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 |
| 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 | - |


Subject to Agreement on the Use of Product Information

**Table 2-7 Cortex-M0 instructions** (cont'd)

| Mnemonic | Operands | Brief description | Flags |
|---|---|---|---|
| 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 |


Subject to Agreement on the Use of Product Information

**Table 2-7 Cortex-M0 instructions** (cont'd)

| 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 **Section 2.2.7**.

## 2.5 Exception Model

This section describes the exception model. It describes:

- Exception states (**Section 2.5.1**)
- Exception types (**Section 2.5.2**)
- Exception handlers (**Section 2.5.3**)
- Vector table (**Section 2.5.4**)
- Exception priorities (**Section 2.5.5**)
- Exception entry and return (**Section 2.5.6**)

### 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.<br>*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. |


Subject to Agreement on the Use of Product Information

### 2.5.2 Exception Types

The exception types are described in **Table 2-8**.

**Table 2-8 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 2-9 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 | - | - | - |
| 3 | -13 | HardFault | -1 | 0x0000000C | Synchronous |


Subject to Agreement on the Use of Product Information

**Table 2-9 Properties of the different exception types** (cont'd)

| Exception number[1] | IRQ number[1] | Exception type | Priority | Vector address or offset[2] | Activation |
|---|---|---|---|---|---|
| 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 in **Section 2.5.4** 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 2-9** which have configurable priority, see Interrupt Clear-enable Register in Interrupt System chapter.

For more information about HardFaults, see Fault handling in **Section 2.6**.

### 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 2-6** shows the order of


Subject to Agreement on the Use of Product Information

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.

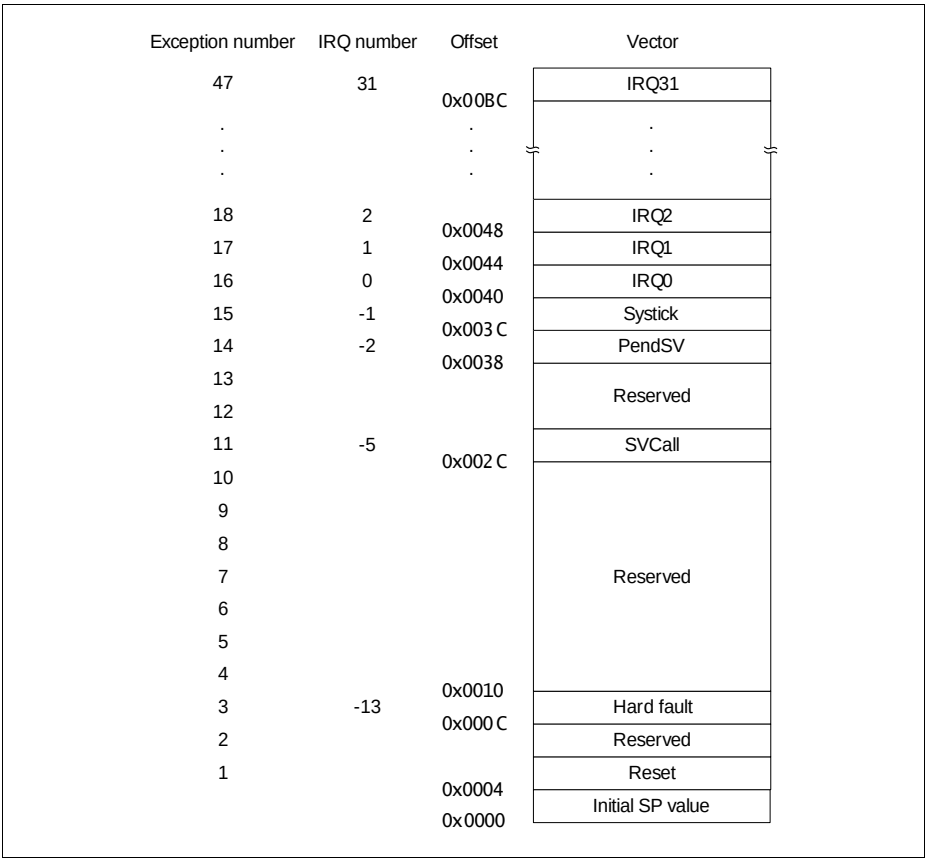



**Figure 2-6 Vector table**

The vector table is fixed at address 0x00000000.

### 2.5.4.1 Vector Table Remap

In XMC1300, the vector table is located inside the ROM. Therefore, the vector table is remapped to the SRAM based on the mapping shown in **Table 2-10**. 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.


Subject to Agreement on the Use of Product Information

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 contraints, 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 2-10 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 | $0000'002C_H$ | $2000'002C_H$ |
| 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 2-9** 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].


Subject to Agreement on the Use of Product Information

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].

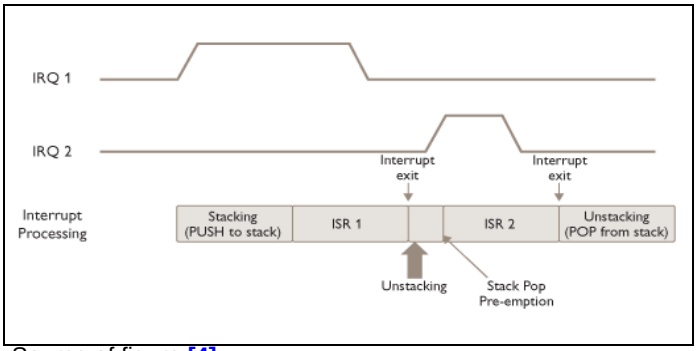
When the processor is executing an exception handler, the exception handler is preempted if a higher priority exception occurs. If an exception occurs with the same priority as the exception being handled, the handler is not preempted, irrespective of the exception number. However, the status of the 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 in **Section 2.5.6.1** for more information.




Source of figure **[4]**.

**Return**

This occurs when the exception handler is completed, and:

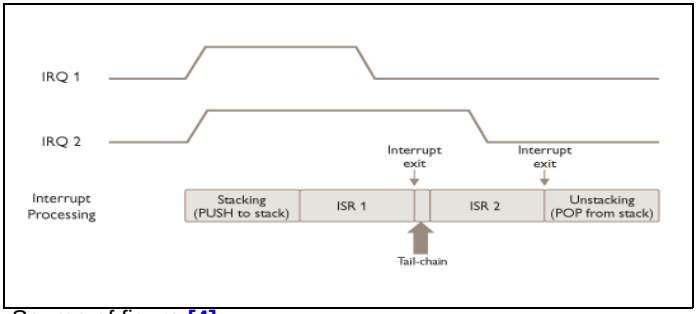
- there is no pending exception with sufficient priority to be serviced
- the completed exception handler was not handling a late-arriving exception.

The processor pops the stack and restores the processor state to the state it had before the interrupt occurred. See Exception return in **Section 2.5.6.2** for more information.


Subject to Agreement on the Use of Product Information

**Tail-chaining**

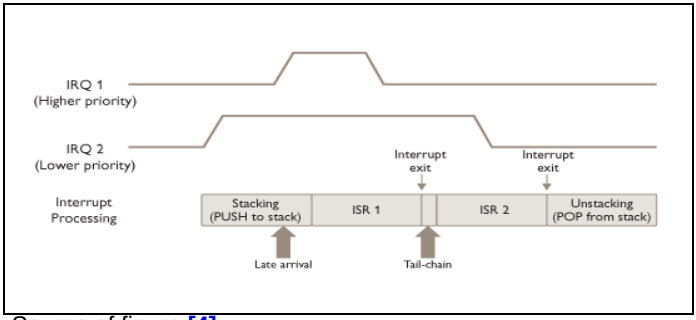
This mechanism speeds up exception servicing. On completion of an exception handler, if there is a pending exception that meets the requirements for exception entry, the stack pop is skipped and control transfers to the new exception handler.




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:


Subject to Agreement on the Use of Product Information

- 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 2-7**.

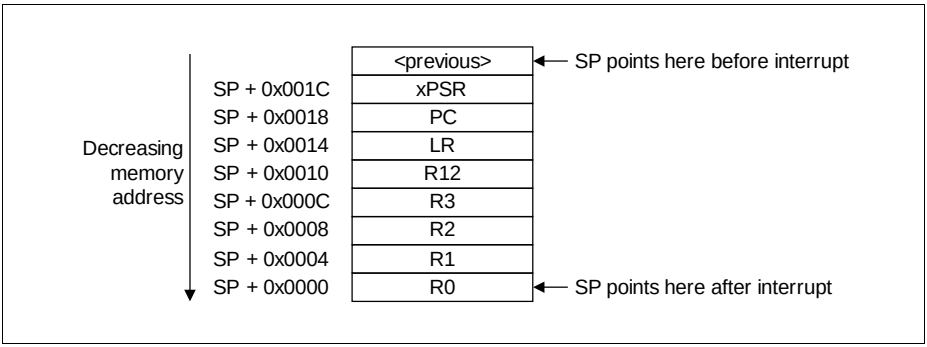



**Figure 2-7 Exception stack frame**

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.


Subject to Agreement on the Use of Product Information

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 2-11** shows the EXC_RETURN values with description of the exception return behavior.

**Table 2-11 Exception return behavior**

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


Subject to Agreement on the Use of Product Information

## 2.6 Fault Handling

Faults are a subset of the exceptions, see Exception model in **Section 2.5**. 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 Resetcan 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


Subject to Agreement on the Use of Product Information

## 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.


Subject to Agreement on the Use of Product Information

### 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 (See **SCR**)
- debug event, if debug is enabled

*Note: Wakeup by Event Register (see The Event Register in ARMv6-M Architecture Reference Manual **[2]**) is not possible in XMC1300. 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
```


Subject to Agreement on the Use of Product Information

## 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 2-12 Core peripheral register regions**

| Address | Core peripheral | Description |
|---|---|---|
| 0xE000E008-0xE000E00F | System Control Block | See **Section 2.8.2** and **Section 2.9.1** |
| 0xE000E010-0xE000E01F | System timer | See **Section 2.8.3** and **Section 2.9.2** |
| 0xE000E100-0xE000E4EF | Nested Vectored Interrupt Controller | See Interrupt System chapter |
| 0xE000ED00-0xE000ED3F | System Control Block | See **Section 2.8.2** and **Section 2.9.1** |
| 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.*


Subject to Agreement on the Use of Product Information

#### 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.


Subject to Agreement on the Use of Product Information

## 2.9 PPB Registers

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

**Table 2-13 Register Overview**

| Short Name | Description | Offset Address | Access Mode | | Description See |
|---|---|---|---|---|---|
| | | | Read | Write | |
| **System Control Space (SCS)** | | | | | |
| CPUID | CPUID Base Register | $D00_H$ | PV, 32 | PV, 32 | Page 2-41 |
| ICSR | Interrupt Control and State Register | $D04_H$ | PV, 32 | PV, 32 | Page 2-42 |
| AIRCR | Application Interrupt and Reset Control Register | $D0C_H$ | PV, 32 | PV, 32 | Page 2-45 |
| SCR | System Control Register | $D10_H$ | PV, 32 | PV, 32 | Page 2-46 |
| CCR | Configuration and Control Register | $D14_H$ | PV, 32 | PV, 32 | Page 2-48 |
| SHPR2 | System Handler Priority Register 2 | $D1C_H$ | PV, 32 | PV, 32 | Page 2-49 |
| SHPR3 | System Handler Priority Register 3 | $D20_H$ | PV, 32 | PV, 32 | Page 2-50 |
| SHCSR | System Handler Control and State Register | $D24_H$ | PV, 32 | PV, 32 | Page 2-51 |
| **System Timer (SysTick)** | | | | | |
| SYST_CSR | SysTick Control and Status Register | $010_H$ | PV, 32 | PV, 32 | Page 2-52 |
| SYST_RVR | SysTick Reload Value Register | $014_H$ | PV, 32 | PV, 32 | Page 2-54 |
| SYST_CVR | SysTick Current Value Register | $018_H$ | PV, 32 | PV, 32 | Page 2-55 |
| SYST_CALIB | SysTick Calibration Value Register | $01C_H$ | PV, 32 | - | Page 2-56 |

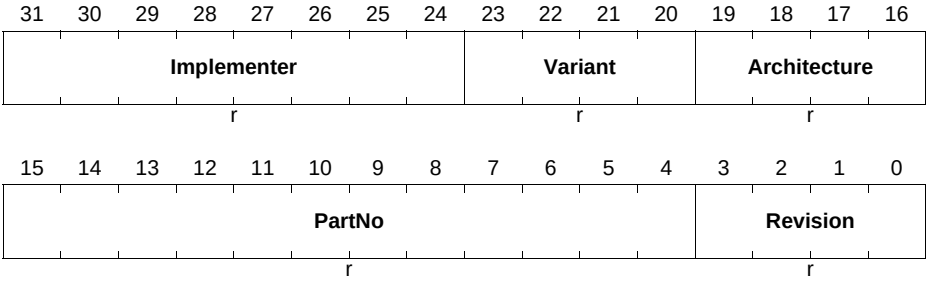

Subject to Agreement on the Use of Product Information

## 2.9.1 SCS Registers

### CPUID

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

**CPUID**
**CPUID Base Register** **(E000ED00$_H$)** **Reset Value: 410CC200$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **Revision** | [3:0] | r | **Revision Number**<br>$0_H$ Patch 0 |
| **PartNo** | [15:4] | r | **Part Number of the Processor**<br>$C20_H$ Cortex-M0 |
| **Architecture** | [19:16] | r | **Architecture**<br>$C_H$ ARMv6-M |
| **Variant** | [23:20] | r | **Variant Number**<br>$0_H$ Revision 0 |
| **Implementer** | [31:24] | r | **Implementer Code**<br>$41_H$ ARM |


Subject to Agreement on the Use of Product Information

### ICSR

The ICSR:

- provides:
  - set-pending and clear-pending bits for the PendSV and SysTick exceptions
- indicates:
  - the exception number of the exception being processed
  - whether there are preempted active exceptions
  - the exception number of the highest priority pending exception
  - whether any interrupts are pending.

**ICSR**
**Interrupt Control and State Register**

**(E000ED04$_H$)** **Reset Value: 00000000$_H$**

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

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

| Field | Bits | Type | Description |
|---|---|---|---|
| **VECTACTIVE**[1)] | [5:0] | r | **Active Exception Number**<br>$00_H$ Thread mode<br>Non-zero value The exception number of the currently active exception.<br>*Note: Subtract 16 from this value to obtain the CMSIS IRQ number required to index into the Interrupt Clear-Enable, Set-Enable, Clear-Pending, Set-Pending, or Priority Registers, see **Interrupt Program Status Register**.* |
| **0** | [11:6] | r | **Reserved**<br>Read as 0; should be written with 0. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **VECTPENDING** | [17:12] | r | **Pending Exception Number**<br>Indicates the exception number of the highest priority pending enabled exception.<br>$0_H$ No pending exceptions<br>Non-zero value: The exception number of the highest priority pending enabled exception. |
| **0** | [21:18] | r | **Reserved**<br>Read as 0; should be written with 0. |
| **ISRPENDING** | 22 | r | **Interrupt Pending Flag**<br>This bit sets the interrupt pending flag, excluding faults.<br>$0_B$ Interrupt not pending<br>$1_B$ Interrupt pending. |
| **0** | [24:23] | r | **Reserved**<br>Read as 0; should be written with 0. |
| **PENDSTCLR** | 25 | w | **SysTick Exception Clear-pending**<br>$0_B$ No effect<br>$1_B$ removes the pending state from the SysTick exception.<br>This bit is write-only. On a register read, this value is unknown. |
| **PENDSTSET** | 26 | rw | **SysTick Exception Set-pending**<br>$0_D$ SysTick exception is not pending<br>$1_D$ SysTick exception is pending.<br>A write of 0 to the bit has no effect. |
| **PENDSVCLR** | 27 | w | **PendSV Clear Pending**<br>This bit clears a pending PendSV exception.<br>$0_B$ Do not clear.<br>$1_B$ Removes pending state from PendSV exception. |
| **PENDSVSET** | 28 | rw | **PendSV Set Pending**<br>This bit sets a pending PendSV exception or reads back the current state.<br>$0_B$ PendSV exception is not pending.<br>$1_B$ PendSV excepton is pending.<br>*Note: Writing 1 to this bit is the only way to set the PendSV exception state to pending.*<br>A software write of 0 to the bit has no effect. |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | [31:29] | r | **Reserved**<br>Read as 0; should be written with 0. |

1) This is the same value as IPSR bits[5:0], see **Interrupt Program Status Register**.

*Note: The result is unpredictable if:*

1. *Both PENDSVSET and PENDSVCLR bits are set to 1.*
2. *Both PENDSTSET and PENDSTCLR bits are set to 1.*

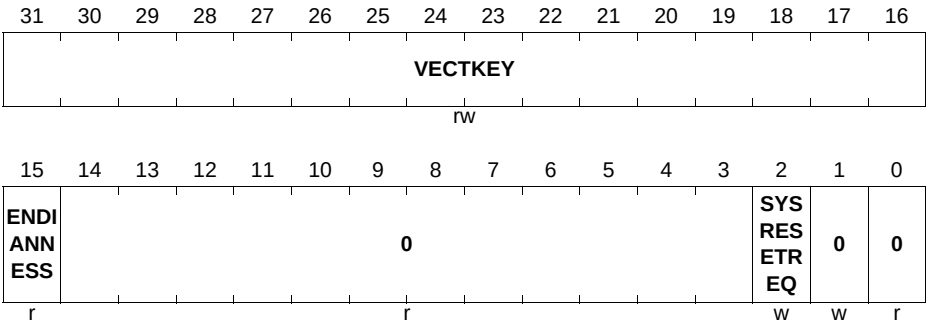

Subject to Agreement on the Use of Product Information

### AIRCR

The AIRCR register provides endian status for data accesses and reset control of the system. To write to this register, you must write 0x5FA to the VECTKEY field, otherwise the processor ignores the write.

### AIRCR
### Application Interrupt and Reset Control Register

**(E000ED0C$_H$)** **Reset Value: FA050000$_H$**



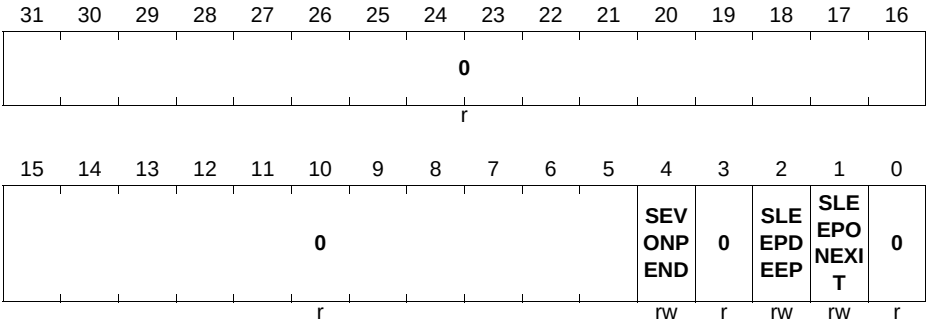

| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | 0 | r | **Reserved**<br>Read as 0; should be written with 0. |
| **0** | 1 | w | **Reserved**<br>Must be written with 0. |
| **SYSRESETREQ** | 2 | w | **System Reset Request**<br>$0_B$ No effect.<br>$1_B$ Requests a system level reset.<br>This bit is read as 0. |
| **0** | [14:3] | r | **Reserved**<br>Read as 0; should be written with 0. |
| **ENDIANNESS** | 15 | r | **Data Endianness**<br>$0_B$ Little-endian |
| **VECTKEY** | [31:16] | rw | **Register Key**<br>Reads as unknown.<br>On writes, write 0x5FA to VECTKEY, otherwise the write is ignored. |


Subject to Agreement on the Use of Product Information

### SCR

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

**SCR**
**System Control Register (E000ED10$_H$) Reset Value: 00000000$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | 0 | r | **Reserved**<br>Read as 0; should be written with 0. |
| **SLEEPONEXIT** | 1 | rw | **Sleep-on-exit**<br>This bit indicates sleep-on-exit when returning from Handler mode to Thread mode.<br>$0_B$ Do not sleep when returning to Thread mode.<br>$1_B$ Enter sleep, or deep sleep, on return from an ISR to Thread mode.<br>Setting this bit to 1 enables an interrupt driven application to avoid returning to an empty main application. |
| **SLEEPDEEP** | 2 | rw | **Low Power Sleep Mode**<br>This bit controls whether the processor uses sleep or deep sleep as its low power mode.<br>$0_B$ Sleep<br>$1_B$ Deep sleep |
| **0** | 3 | r | **Reserved**<br>Read as 0; should be written with 0. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **SEVONPEND** | 4 | rw | **Send Event on Pending bit**<br>$0_B$ Only enabled interrupts or events can wakeup the processor, disabled interrupts are excluded.<br>$1_B$ Enabled events and all interrupts, including disabled interrupts, can wakeup the processor.<br>When an event or interrupt enters pending state, the event signal wakes up the processor from WFE. If the processor is not waiting for an event, the event is registered and affects the next WFE. |
| **0** | [31:5] | r | **Reserved**<br>Read as 0; should be written with 0. |


Subject to Agreement on the Use of Product Information

### CCR

The CCR is a read-only register and it indicates some aspects of the behavior of the Cortex-M0 processor.

**CCR**
**Configuration and Control Register**

**(E000ED14$_H$)** **Reset Value: 00000208$_H$**

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

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

| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | [2:0] | r | **Reserved**<br>Read as 0; should be written with 0. |
| **UNALIGN_TRP** | 3 | r | **Unaligned Access Traps**<br>This bit always reads as 1, indicates that all unaligned accesses generate a HardFault. |
| **0** | [8:4] | r | **Reserved**<br>Read as 0; should be written with 0. |
| **STKALIGN** | 9 | r | **Stack Alignment**<br>This bit always reads as 1, indicates 8-byte stack alignment on exception entry.<br>On exception entry, the processor uses bit [9] of the stacked PSR to indicate the stack alignment. On return from the exception, it uses this stacked bit to restore the correct stack alignment. |
| **0** | [31:10] | r | **Reserved**<br>Read as 0; should be written with 0. |


Subject to Agreement on the Use of Product Information

### System Handler Priority Registers

The SHPR2-SHPR3 registers set the priority level, 0 to 192, of the exception handlers that have configurable priority.

SHPR2-SHPR3 are word accessible. To access to the system exception priority level using CMSIS, the following CMSIS functions are used:

- `uint32_t NVIC_GetPriority(IRQn_Type IRQn)`
- `void NVIC_SetPriority(IRQn_Type IRQn, uint32_t priority)`

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

**Table 2-14 System fault handler priority fields**

| Handler | Field | Register description |
|---|---|---|
| SVCall | PRI_11 | System Handler Priority Register 2 on **Page 2-49** |
| PendSV | PRI_14 | System Handler Priority Register 3 on **Page 2-50** |
| SysTick | PRI_15 | |

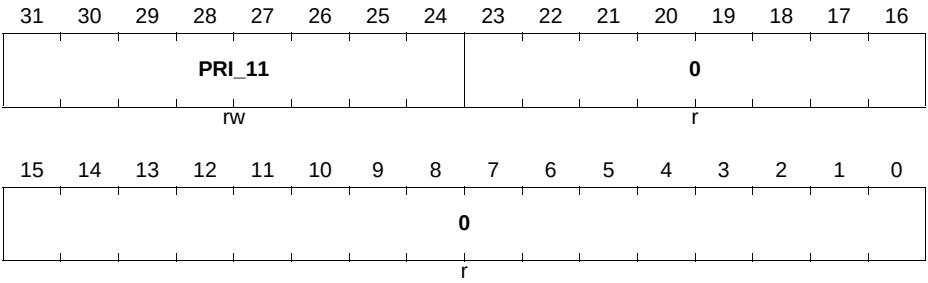
Each PRI_N field is 8 bits wide, but the XMC1300 implements only bits [7:6] of each field, and bits [5:0] read as zero and ignore writes.

### SHPR2

The SHPR2 register sets the priority level for the SVCall handler.

**SHPR2**
**System Handler Priority Register 2**

**(E000ED1C$_H$)** **Reset Value: 00000000$_H$**





Subject to Agreement on the Use of Product Information

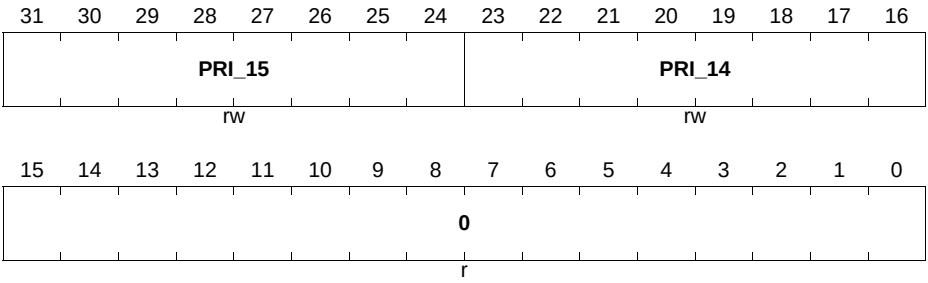
| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | [23:0] | r | **Reserved**<br>Read as 0; should be written with 0. |
| **PRI_11** | [31:24] | rw | **Priority of System Handler 11**<br>SVCall. |

### SHPR3

The SHPR3 register sets the priority level for the PendSV and SysTick handlers.

**SHPR3**
**System Handler Priority Register 3**

**(E000ED20$_H$)** **Reset Value: 00000000$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | [15:0] | r | **Reserved**<br>Read as 0; should be written with 0. |
| **PRI_14** | [23:16] | rw | **Priority of System Handler 14**<br>PendSV. |
| **PRI_15** | [31:24] | rw | **Priority of System Handler 15**<br>SysTick exception. |

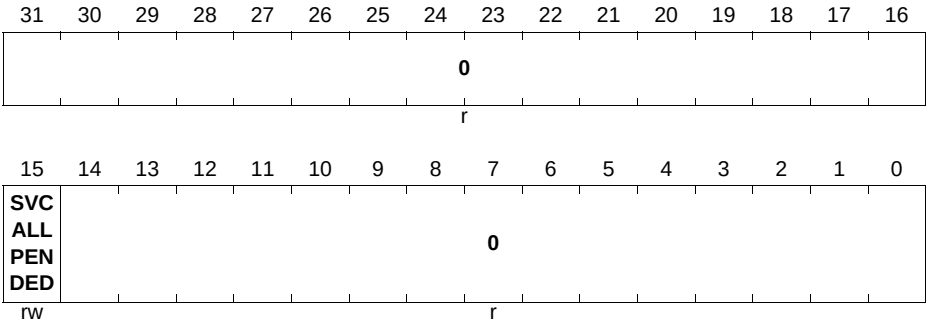

Subject to Agreement on the Use of Product Information

### SHCSR

The SHCSR register controls and provides the status of system handlers.

### SHCSR
**System Handler Control and State Register**

**(E000ED24$_H$)** **Reset Value: 00000000$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | [14:0] | r | **Reserved**<br>Read as 0; should be written with 0. |
| **SVCALLPENDED** | 15 | rw | **SVCall Pending bit**<br>This bit reflects the pending state on a read, and updates the pending state, to the value written, on a write.<br>$0_B$ SVCall is not pending.<br>$1_B$ SVCall is pending[1]. |
| **0** | [31:16] | r | **Reserved**<br>Read as 0; should be written with 0. |

1) Pending state bits are set to 1 when an exception occurs, and are cleared to 0 when an exception becomes active.


Subject to Agreement on the Use of Product Information

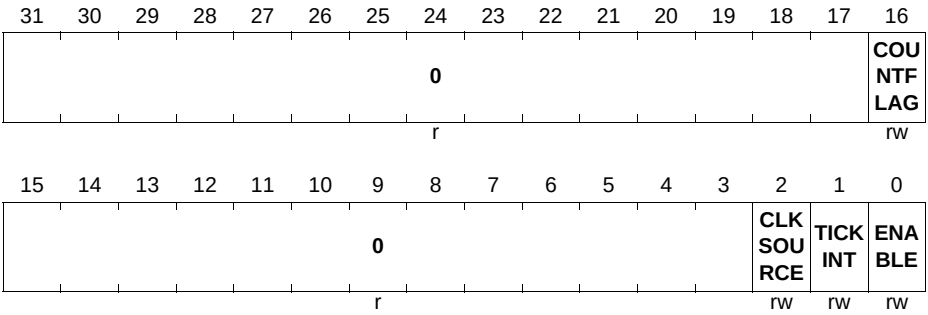
## 2.9.2 SysTick Registers

### SYST_CSR

The SYST_CSR register enables the SysTick features.

Reading SYST_CSR clears the COUNTFLAG bit to 0.

**SYST_CSR**
**SysTick Control and Status Register**
**(E000E010$_H$)** **Reset Value: 00000000$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **ENABLE** | 0 | rw | **Counter Enable**<br>This bit enables the counter.<br>$0_B$ Counter disabled.<br>$1_B$ Counter enabled. |
| **TICKINT** | 1 | rw | **SysTick Exception Request**<br>This bit enables the SysTick exception request.<br>$0_B$ Counting down to zero does not assert the SysTick exception request.<br>$1_B$ Counting down to zero to assert the SysTick exception request.<br>In software, COUNTFLAG bit can be used to determine if SysTick has counted to zero. |
| **CLKSOURCE** | 2 | rw | **Clock Source**<br>This bit selects the SysTick timer clock source.<br>$0_B$ External clock[1)].<br>$1_B$ Processor clock. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | [15:3] | r | **Reserved**<br>Read as 0; should be written with 0. |
| **COUNTFLAG** | 16 | rw | **Counter Flag**<br>This bit returns 1 if timer counted to 0 since the last read of this register. |
| **0** | [31:17] | r | **Reserved**<br>Read as 0; should be written with 0. |

1) In XMC1300, the external clock refers to the on-chip 32 kHz standby clock.

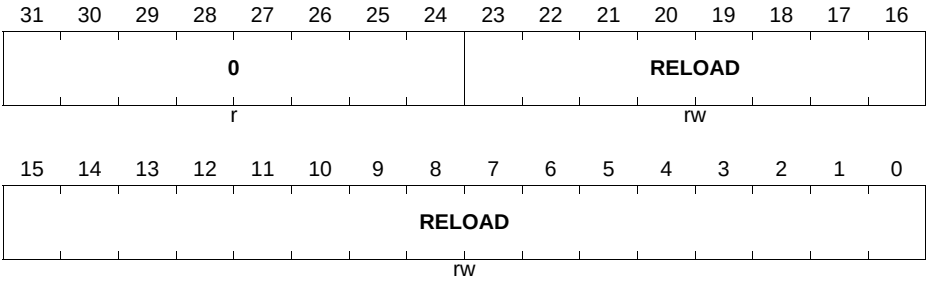
When ENABLE is set to 1, the counter loads the RELOAD value from the SYST_RVR register and then counts down. On reaching 0, it sets the COUNTFLAG to 1 and optionally asserts the SysTick depending on the value of TICKINT. It then loads the RELOAD value again, and begins counting.


Subject to Agreement on the Use of Product Information

### SYST_RVR

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

**SYST_RVR**
**SysTick Reload Value Register (E000E014$_H$) Reset Value: XXXXXXXX$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **RELOAD** | [23:0] | rw | **Reload Value**<br>This field sets the value to load into the SYST_CVR register when the counter is enabled and when it reaches 0. |
| **0** | [31:24] | r | **Reserved**<br>Read as 0; should be written with 0. |

**Notes on calculating the RELOAD value**

1. *The RELOAD value can be any value in the range 0x00000001-0x00FFFFFF. A start value of 0 is possible, but this has no effect because the SysTick exception request and COUNTFLAG are activated when counting from 1 to 0.*
2. *The RELOAD value is calculated according to its use. For example, to generate a multi-shot timer with a period of N processor clock cycles, use a RELOAD value of N-1. If the SysTick interrupt is required every 100 clock pulses, set RELOAD to 99.*

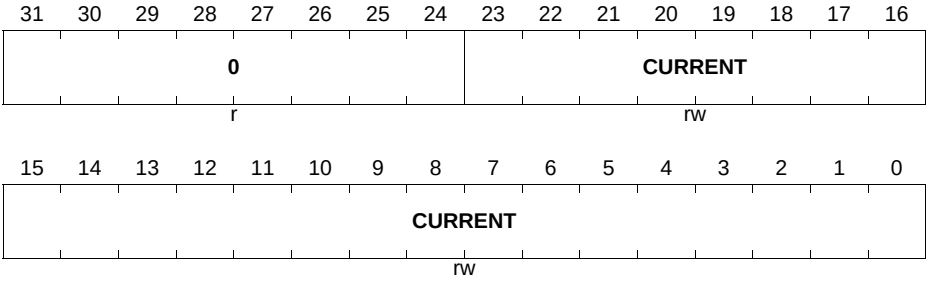

Subject to Agreement on the Use of Product Information

### SYST_CVR

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

Writing to the SYST_CVR clears the register and the COUNTFLAG status bit to 0. The write does not trigger the SysTick exception logic. Reading the register returns its value at the time it is accessed.

**SYST_CVR**
**SysTick Current Value Register (E000E018$_H$) Reset Value: XXXXXXXX$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **CURRENT** | [23:0] | rw | **SysTick Counter Current Value**<br>When read, it returns the current value of the SysTick counter.<br>A write of any value clears the field to 0, and also clears the SYST_CSR.COUNTFLAG bit to 0. |
| **0** | [31:24] | r | **Reserved**<br>Read as 0; should be written with 0. |

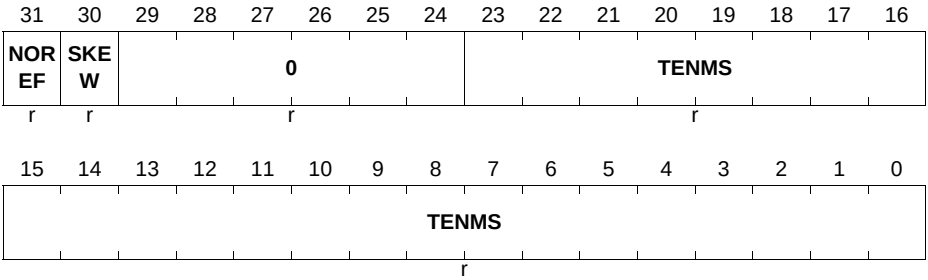

Subject to Agreement on the Use of Product Information

### SYST_CALIB

The SYST_CALIB register indicates the SysTick calibration properties.

### SYST_CALIB

**SysTick Calibration Value Register(E000E01C$_H$)** **Reset Value: 40000147$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **TENMS** | [23:0] | r | **10 Milliseconds**<br>The reload value for 10ms timing is subject to system clock skew errors. The default value of TENMS is 0x000147. |
| **0** | [29:24] | r | **Reserved**<br>Read as 0; should be written with 0. |
| **SKEW** | 30 | r | **Clock Skew**<br>This bit is read as 1. It indicates that 10ms calibration value is inexact, because of the clock frequency. |
| **NOREF** | 31 | r | **Reference Clock**<br>This bit is read as 0. It indicates that external reference clock is provided. |

# 3 Bus System

The single master bus system in XMC1300 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.*


Subject to Agreement on the Use of Product Information

# 4 Service Request Processing

A hardware pulse is called Service Request (SR) in an XMC1300 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 4-1 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. XMC1300 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 4-1** shows a representation of the interaction between the request sources and the request processing units.


Subject to Agreement on the Use of Product Information

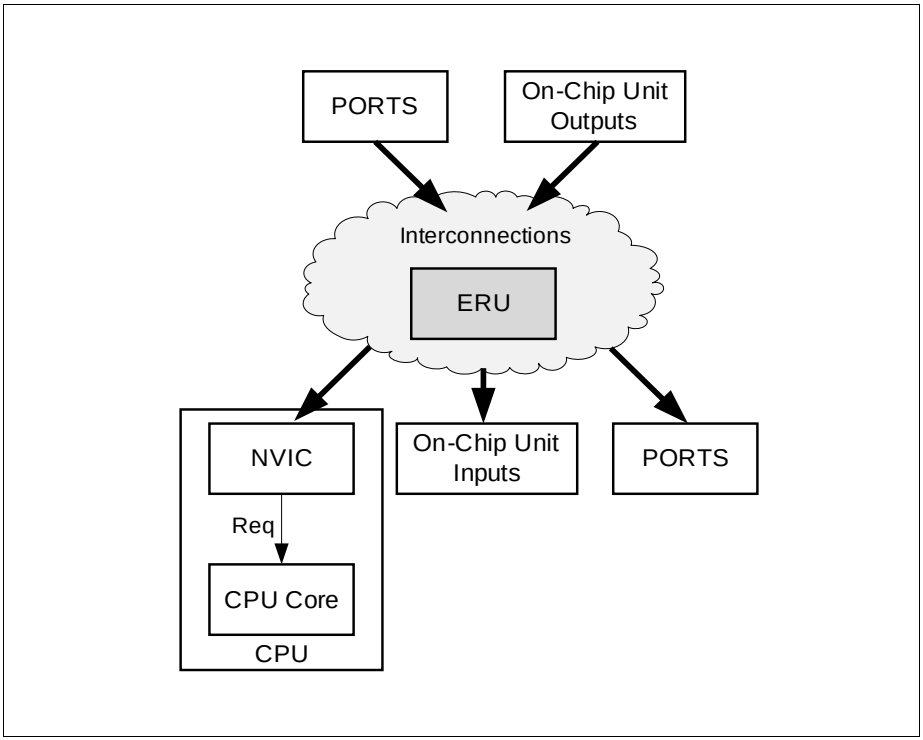



**Figure 4-1 Block Diagram on Service Request Processing**


Subject to Agreement on the Use of Product Information

## 4.2 Service Request Distribution

Figure 4-2 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.

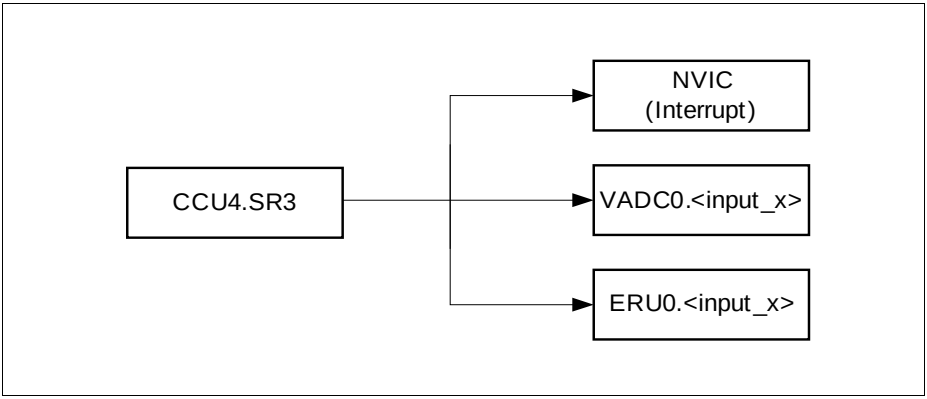



**Figure 4-2 Example for Service Request Distribution**

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 dependant. Related connectivity and availability of pins can be looked up in the

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

### Interrupt services

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

Service Requests are always of of type “Pulse” in XMC1300.


Subject to Agreement on the Use of Product Information

**Table 4-2 Interrupt services per module**

| Modules | Request Sources | NVIC | Type |
|---|---|---|---|
| VADC | 12 | 6 | Pulse |
| CCU40 | 4 | 4 | Pulse |
| CCU80 | 4 | 2 | Pulse |
| POSIF0 | 2 | 2 | Pulse |
| USIC0 | 6 | 6 | Pulse |
| BCCU0 | 1 | 1 | Pulse |
| MATH | 1 | 1 | Pulse |
| SCU | 3 | 3 | Pulse |
| ERU0 | 4 | 4 | Pulse |
| Total | 37 | 29 | - |


Subject to Agreement on the Use of Product Information

# 5 Interrupt Subsystem

The interrupt Subsystem in XMC1300 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 5-1** 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.

**Table 5-1 Interrupt Node assignment**

| Service Request | Node ID | Description |
|---|---|---|
| SCU.SR0 - SCU.SR2 | 0...2 | System Control<br>SR0 is the system critical request<br>SR1 is the common SCU request<br>SR2 is the comparators (ACMPx and ORCx) request |
| ERU0.SR0 - ERU0.SR3 | 3...6 | External Request Unit 0 |
| MATH.SR0 - MATH.SR1 | 7 | MATH |
| NC | 8 | Reserved |
| USIC0.SR0 - USIC0.SR5 | 9...14 | Universal Serial Interface Channel (Module 0) |
| VADC0.C0SR0 - VADC0.C0SR1 | 15...16 | Analog to Digital Converter (Common) |


Subject to Agreement on the Use of Product Information

**Table 5-1 Interrupt Node assignment** (cont'd)

| Service Request | Node ID | Description |
|---|---|---|
| VADC0.G0SR0 - VADC0.G0SR1 | 17...18 | Analog to Digital Converter (Group 0) |
| VADC0.G1SR0 - VADC0.G1SR1 | 19...20 | Analog to Digital Converter (Group 1) |
| CCU40.SR0 - CCU40.SR3 | 21...24 | Capture Compare Unit 4 (Module 0) |
| CCU80.SR0 - CCU80.SR1 | 25...26 | Capture Compare Unit 8 (Module 0) |
| POSIF0.SR0 - POSIF0.SR1 | 27...28 | Position Interface (Module 0) |
| NC | 29...30 | Reserved |
| BCCU0.SR0 | 31 | Brightness Control Unit (Module 0) |

### 5.1.3 Interrupt Signal Generation

In XMC1300, 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:


Subject to Agreement on the Use of Product Information

- 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 5-2 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 |
| 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.


Subject to Agreement on the Use of Product Information

### 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 5-3 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.<br>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.<br>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 5-1**.


Subject to Agreement on the Use of Product Information

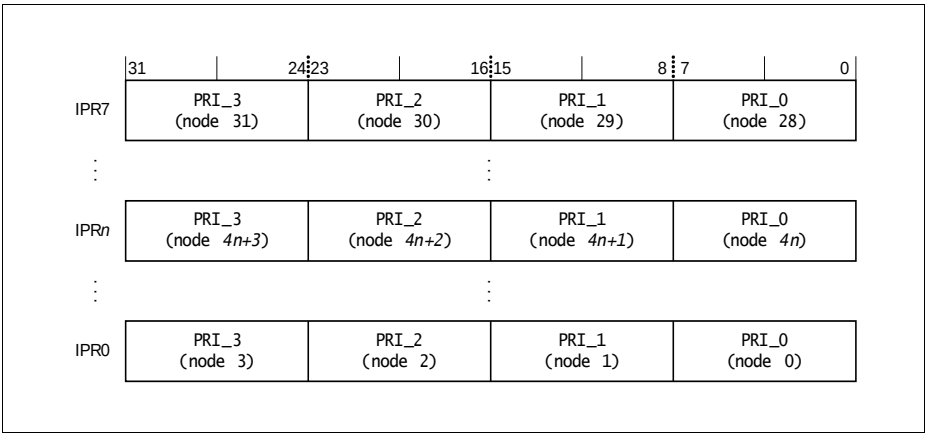



**Figure 5-1 Interrupt Priority Register**

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 5-2** 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 XMC1300 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 5-2**.


Subject to Agreement on the Use of Product Information

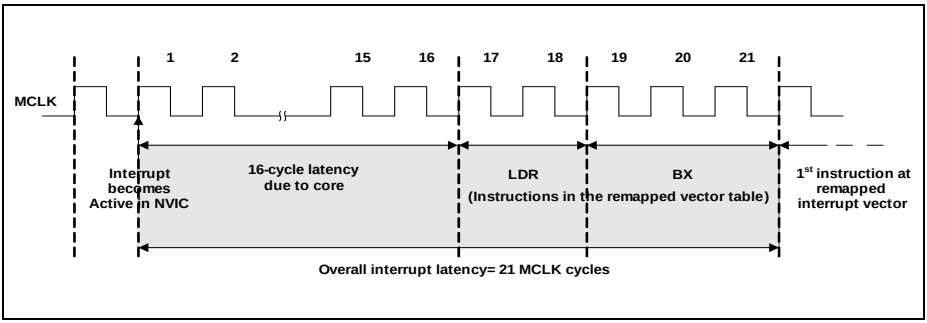



**Figure 5-2 Typical XMC1300 Interrupt Latency**

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 5-3**):

- 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. An overview of all XMC1300 interrupt sources is given at the end of the chapter.

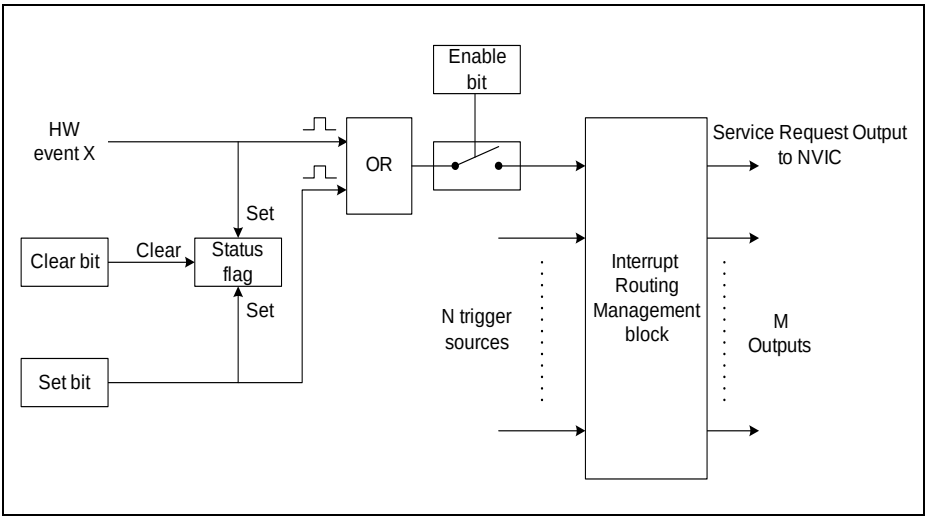



**Figure 5-3 Typical Module Interrupt Structure**

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

- Enable the interrupt node that is allocated to the module in the NVIC, 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.


Subject to Agreement on the Use of Product Information

## 5.3 Registers

**Table 5-4 Registers Address Space**

| Module | Base Address | End Address | Note |
|---|---|---|---|
| CPU PPB: System Control Space (SCS) | $E000E000_H$ | $E000EFFF_H$ | |

### Registers Overview

The absolute register address is calculated by adding:

Module Base Address + Offset Address

**Table 5-5 Register Overview**

| Short Name | Description | Offset Address | Access Mode | | Description See |
|---|---|---|---|---|---|
| | | | Read | Write | |
| **Nested Vectored Interrupt Controller (NVIC)** | | | | | |
| NVIC_ISER | Interrupt Set-enable Registers | $100_H$ | U, PV | U, PV | **Page 5-9** |
| NVIC_ICER | Interrupt Clear-enable Registers | $180_H$ | U, PV | U, PV | **Page 5-10** |
| NVIC_ISPR | Interrupt Set-pending Registers | $200_H$ | U, PV | U, PV | **Page 5-11** |
| NVIC_ICPR | Interrupt Clear-pending Registers | $280_H$ | U, PV | U, PV | **Page 5-12** |
| NVIC_IPR0 - NVIC_IPR7 | Interrupt Priority Registers | $400_H$-$41C_H$ | U, PV | U, PV | **Page 5-13** |

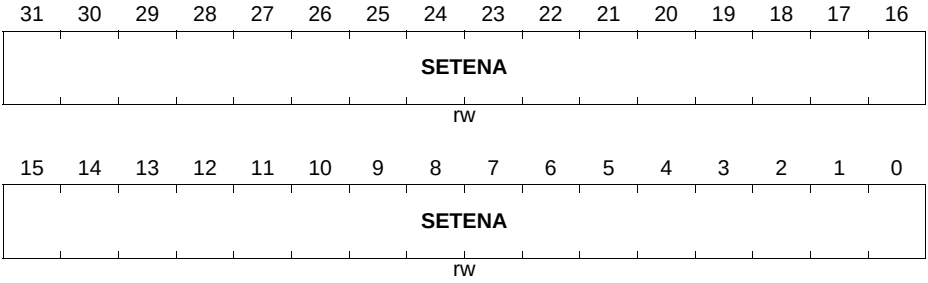

Subject to Agreement on the Use of Product Information

### 5.3.1 NVIC Registers

**NVIC_ISER**

The ISER register enables interrupt nodes, and shows which interrupt nodes are enabled.

**NVIC_ISER**
**Interrupt Set-enable Register (E000E100$_H$) Reset Value: 00000000$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **SETENA** | [31:0] | rw | **Interrupt Node Set-enable**<br>$0_B$ Read: Interrupt node disabled.<br>Write: No effect.<br>$1_B$ Read: Interrupt node enabled.<br>Write: Enable interrupt node |

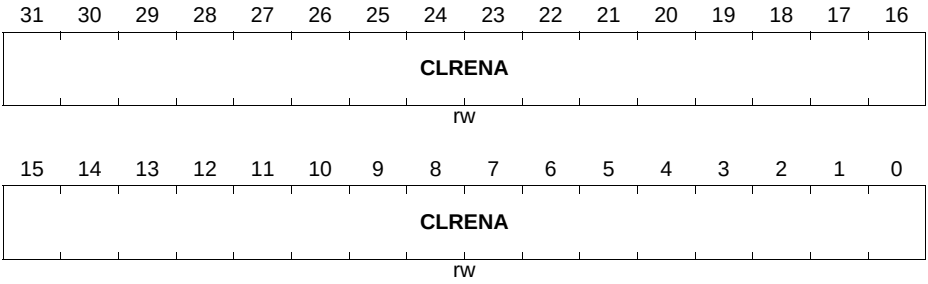
If a pending interrupt is enabled, the NVIC activates the interrupt based on its priority. If an interrupt is not enabled, asserting its interrupt signal changes the interrupt state to pending, but the NVIC never activates the interrupt, regardless of its priority.


Subject to Agreement on the Use of Product Information

### NVIC_ICER

The ICER register disables interrupt nodes, and shows which interrupt nodes are enabled.

**NVIC_ICER**
**IInterrupt Clear-enable Register (E000E180$_H$)** **Reset Value: 00000000$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **CLRENA** | [31:0] | rw | **Interrupt Node Clear-enable**<br>$0_B$ Read: Interrupt node disabled.<br>Write: No effect<br>$1_B$ Read: Interrupt node enabled.<br>Write: Disable interrupt node. |

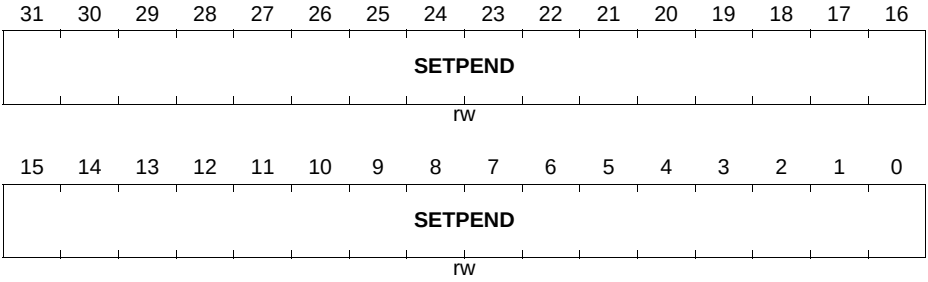

Subject to Agreement on the Use of Product Information

### NVIC_ISPR

The ISPR register forces interrupt nodes into the pending state, and shows which interrupt nodes are pending.

**NVIC_ISPR**
**Interrupt Set-pending Register**

**(E000E200$_H$)** **Reset Value: 00000000$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **SETPEND** | [31:0] | rw | **Interrupt Node Set-pending**<br>$0_B$ Read: Interrupt node is not pending.<br>Write: No effect<br>$1_B$ Read: Interrupt node is pending.<br>Write: Change interrupt state to pending. |

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

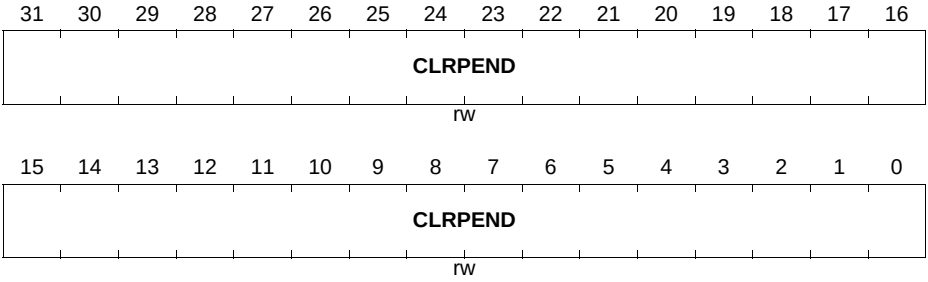

Subject to Agreement on the Use of Product Information

### NVIC_ICPR

The ICPR register removes the pending state from interrupt nodes, and shows which interrupt nodes are pending.

**NVIC_ICPR**
**Interrupt Clear-pending Register**

**(E000E280$_H$)** **Reset Value: 00000000$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **CLRPEND** | [31:0] | rw | **Interrupt Node Clear-pending**<br>$0_B$ Read: Interrupt node is not pending.<br>Write: No effect.<br>$1_B$ Read: Interrupt node is pending.<br>Write: Remove interrupt state from pending. |

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

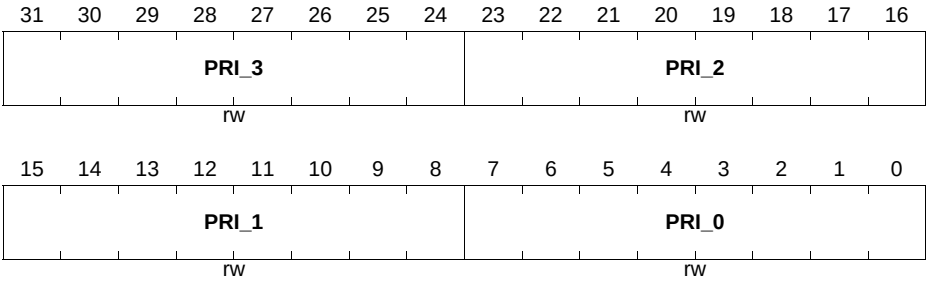
**NVIC_IPRx (x=0-7)**

The IPR0-IPR7 registers provide a 8-bit priority field for each interrupt node. Each register holds four priority fields.

Each priority field holds a priority value, 0-192. The lower the value, the greater the priority of the corresponding interrupt node. The processor implements only bits [7:6] of each field, bits [5:0] reads as 0 and ignores writes. This means writing 255 to a priority register saves value 192 to the register.

**NVIC_IPRx (x=0-7)**
**Interrupt Priority Register x**

**($E000E400_H$ + 4*x)** **Reset Value: $00000000_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **PRI_3** | [31:24] | rw | **Priority, Byte Offset 3** |
| **PRI_2** | [23:16] | rw | **Priority, Byte Offset 2** |
| **PRI_1** | [15:8] | rw | **Priority, Byte Offset 1** |
| **PRI_0** | [7:0] | rw | **Priority, Byte Offset 0** |

## 5.4 Interrupt Request Source Overview

An overview of all XMC1300 interrupt sources and related register bits are shown in the next few pages.


Subject to Agreement on the Use of Product Information

**Table 5-6 Interrupt Source Overview**

| IRQ | Interrupt Node | Interrupt Source | Status Flag | | Interrupt Enable | | Set Flag | | Clear Flag | | Node Pointer | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Register | Bit | Register | Bit | Register | Bit | Register | Bit | Register | Bit |
| 0 | 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 | - | - |

## Table 5-6 Interrupt Source Overview

| IRQ | Interrupt Node | Interrupt Source | Status Flag | | Interrupt Enable | | Set Flag | | Clear Flag | | Node Pointer | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Register | Bit | Register | Bit | Register | Bit | Register | Bit | Register | Bit |
| 1 | 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 | - | - |
| | | 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 | - | - |
| | | RTC TIM1 Mirror Register updated | SCU_SRRAW | RTC_TIM1 | SCU_SRMSK | RTC_TIM1 | SCU_SRSET | RTC_TIM1 | SCU_SRCLR | RTC_TIM1 | - | - |

## Table 5-6 Interrupt Source Overview

| IRQ | Interrupt Node | Interrupt Source | Status Flag | | Interrupt Enable | | Set Flag | | Clear Flag | | Node Pointer | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Register | Bit | Register | Bit | Register | Bit | Register | Bit | Register | Bit |
| 2 | 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 | - | - |
| 3, 4, 5, 6 | ERU0. SR[3:0] | ERU0_IOUTx (x=0-3) | See section on ERU0 for details. | | | | | | | | | |
| 7 | 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 | - | - |
| | MATH. SR1 | 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 | - | - |
| 8 | Reserved | | | | | | | | | | | |
| 9, 10, 11, 12, 13, 14 | USIC0_SR[5:0] | USIC: Standard receive event | USIC0_ PSR | RIF | USIC0_ CCR | RIEN | - | - | USIC0_ PSCR | CRIF | USIC0_ INPR | RINP |
| | | USIC: Receive start event | USIC0_ PSR | RSIF | USIC0_ CCR | RSIEN | - | - | USIC0_ PSCR | CRSIF | USIC0_ INPR | TBINP |
| | | USIC: Alternate receive event | USIC0_ PSR | AIF | USIC0_ CCR | AIEN | - | - | USIC0_ PSCR | CAIF | USIC0_ INPR | AINP |
| | | USIC: Transmit shift event | USIC0_ PSR | TSIF | USIC0_ CCR | TSIEN | - | - | USIC0_ PSCR | CTSIF | USIC0_ INPR | TSINP |
| | | USIC: Transmit buffer event | USIC0_ PSR | TBIF | USIC0_ CCR | TBIEN | - | - | USIC0_ PSCR | CTBIF | USIC0_ INPR | TBINP |
| | | USIC: Data lost event | USIC0_ PSR | DLIF | USIC0_ CCR | DLIEN | - | - | USIC0_ PSCR | CDLIF | USIC0_ INPR | PINP |
| | | USIC: BRG event | USIC0_ PSR | BRGIF | USIC0_ CCR | BRGIEN | - | - | USIC0_ PSCR | CBRGIF | USIC0_ INPR | PINP |

## Table 5-6 Interrupt Source Overview

| IRQ | Interrupt Node | Interrupt Source | Status Flag | | Interrupt Enable | | Set Flag | | Clear Flag | | Node Pointer | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Register | Bit | Register | Bit | Register | Bit | Register | Bit | Register | Bit |
| 9, 10, 11, 12, 13, 14 | USIC0_SR[5:0] | USIC: Standard transmit buffer event | USIC0_TRBSR | STBI | USIC0_TBCTR | STBIEN | - | - | USIC0_TRBSCR | CSTBI | USIC0_TBCTR | STBINP |
| | | USIC: Standard transmit buffer event | USIC0_TRBSR | STBT | USIC0_TBCTR | STBIEN | - | - | - | - | USIC0_TBCTR | STBINP |
| | | USIC: Transmit Buffer error event | USIC0_TRBSR | TBERI | USIC0_TBCTR | TBERIEN | - | - | USIC0_TRBSCR | CTBERI | USIC0_TBCTR | ATBINP |
| | | USIC: Standard receive buffer event | USIC0_TRBSR | SRBI | USIC0_RBCTR | SRBIEN | - | - | USIC0_TRBSCR | CSRBI | USIC0_RBCTR | SRBINP |
| | | USIC: Standard receive buffer event | USIC0_TRBSR | SRBT | USIC0_RBCTR | SRBIEN | - | - | - | - | USIC0_RBCTR | SRBINP |
| | | USIC: Alternate receive buffer event | USIC0_TRBSR | ARBI | USIC0_RBCTR | ARBIEN | - | - | USIC0_TRBSCR | CARBI | USIC0_RBCTR | ARBINP |
| | | USIC: Receive buffer error event | USIC0_TRBSR | RBERI | USIC0_RBCTR | RBERIEN | - | - | USIC0_TRBSCR | CRBERI | USIC0_RBCTR | ARBINP |
| | | ASC: Synchronisation break detected | USIC0_PSR | SBD | USIC0_PCR | SBDIEN | - | - | USIC0_PSCR | CSBD | USIC0_INPR | PINP |
| | | ASC: Collision detected | USIC0_PSR | COL | USIC0_PCR | CDIEN | - | - | USIC0_PSCR | CCOL | USIC0_INPR | PINP |
| | | ASC: Receiver noise detected | USIC0_PSR | RNS | USIC0_PCR | RNIEN | - | - | USIC0_PSCR | CRNS | USIC0_INPR | PINP |
| | | ASC: Format error in stop bit 0 | USIC0_PSR | FER0 | USIC0_PCR | FEIEN | - | - | USIC0_PSCR | CFER0 | USIC0_INPR | PINP |
| | | ASC: Format error in stop bit 1 | USIC0_PSR | FER1 | USIC0_PCR | FEIEN | - | - | USIC0_PSCR | CFER1 | USIC0_INPR | PINP |
| | | ASC: Receive frame finished | USIC0_PSR | RFF | USIC0_PCR | FFIEN | - | - | USIC0_PSCR | CRFF | USIC0_INPR | PINP |
| | | ASC: Transmit frame finished | USIC0_PSR | TFF | USIC0_PCR | FFIEN | - | - | USIC0_PSCR | CTFF | USIC0_INPR | PINP |

## Table 5-6 Interrupt Source Overview

| IRQ | Interrupt Node | Interrupt Source | Status Flag | | Interrupt Enable | | Set Flag | | Clear Flag | | Node Pointer | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Register | Bit | Register | Bit | Register | Bit | Register | Bit | Register | Bit |
| 9, 10, 11, 12, 13, 14 | USIC0_SR[5:0] | SSC: MSLS event detected | USIC0_PSR | MSLSEV | USIC0_PCR | MSLSIEN | - | - | USIC0_PSCR | CMSLSEV | USIC0_INPR | PINP |
| | | SSC: Parity error detected | USIC0_PSR | PAERR | USIC0_PCR | PARIEN | - | - | USIC0_PSCR | CPAERR | USIC0_INPR | PINP |
| | | SSC: DX2T event detected | USIC0_PSR | DX2TEV | USIC0_PCR | DX2TIEN | - | - | USIC0_PSCR | CDX2TEV | USIC0_INPR | PINP |
| | | IIC: Wrong TDF code detected | USIC0_PSR | WTDF | USIC0_PCR | ERRIEN | - | - | USIC0_PSCR | CWTDF | USIC0_INPR | PINP |
| | | IIC: Start condition received | USIC0_PSR | SCR | USIC0_PCR | SCRIEN | - | - | USIC0_PSCR | CSCR | USIC0_INPR | PINP |
| | | IIC: Repeated start condition received | USIC0_PSR | RSCR | USIC0_PCR | RSCRIEN | - | - | USIC0_PSCR | CRSCR | USIC0_INPR | PINP |
| | | IIC: Stop condition received | USIC0_PSR | PCR | USIC0_PCR | PCRIEN | - | - | USIC0_PSCR | CPCR | USIC0_INPR | PINP |
| | | IIC: NACK received | USIC0_PSR | NACK | USIC0_PCR | NACKIEN | - | - | USIC0_PSCR | CNACK | USIC0_INPR | PINP |
| | | IIC: Arbitration lost | USIC0_PSR | ARL | USIC0_PCR | ARLIEN | - | - | USIC0_PSCR | CARL | USIC0_INPR | PINP |
| | | IIC: Slave read request | USIC0_PSR | SRR | USIC0_PCR | SRRIEN | - | - | USIC0_PSCR | CSRR | USIC0_INPR | PINP |
| | | IIC: Error detected | USIC0_PSR | ERR | USIC0_PCR | ERRIEN | - | - | USIC0_PSCR | CERR | USIC0_INPR | PINP |
| | | IIC: ACK received | USIC0_PSR | ACK | USIC0_PCR | ACKIEN | - | - | USIC0_PSCR | CACK | USIC0_INPR | PINP |
| | | IIS: DX2T event detected | USIC0_PSR | DX2TEV | USIC0_PCR | DX2TIEN | - | - | USIC0_PSCR | CDX2TEV | USIC0_INPR | PINP |
| | | IIS: WA falling edge event | USIC0_PSR | WAFE | USIC0_PCR | WAFEIEN | - | - | USIC0_PSCR | CWAFE | USIC0_INPR | PINP |
| | | IIS: WA rising edge event | USIC0_PSR | WARE | USIC0_PCR | WAREIEN | - | - | USIC0_PSCR | CWARE | USIC0_INPR | PINP |
| | | IIS: WA generation end | USIC0_PSR | END | USIC0_PCR | ENDIEN | - | - | USIC0_PSCR | CEND | USIC0_INPR | PINP |

**Table 5-6 Interrupt Source Overview**

| IRQ | Interrupt Node | Interrupt Source | Status Flag | | Interrupt Enable | | Set Flag | | Clear Flag | | Node Pointer | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Register | Bit | Register | Bit | Register | Bit | Register | Bit | Register | Bit |
| 15, 16, 17, 18, 19, 20 | 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 |





Subject to Agreement on the Use of Product Information

## Table 5-6 Interrupt Source Overview

| IRQ | Interrupt Node | Interrupt Source | Status Flag | | Interrupt Enable | | Set Flag | | Clear Flag | | Node Pointer | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Register | Bit | Register | Bit | Register | Bit | Register | Bit | Register | Bit |
| 21, 22, 23, 24 | CCU40_ SR[3:0] | Event 0 edge(s) information from event selector | CCU40_ CC4yINTS | E0AS | CCU40_ CC4yINTE | E0AE | CCU40_ CC4ySWS | SE0A | CCU40_ CC4ySWR | RE0A | CCU40_ CC4ySRS | E0SR |
| | | Event 1 edge(s) information from event selector | CCU40_ CC4yINTS | E1AS | CCU40_ CC4yINTE | E1AE | CCU40_ CC4ySWS | SE1A | CCU40_ CC4ySWR | RE1A | CCU40_ CC4ySRS | E1SR |
| | | Event 2 edge(s) information from event selector | CCU40_ CC4yINTS | E2AS | CCU40_ CC4yINTE | E2AE | CCU40_ CC4ySWS | SE2A | CCU40_ CC4ySWR | RE2A | CCU40_ CC4ySRS | E2SR |
| | | Period Match while counting up | CCU40_ CC4yINTS | PMUS | CCU40_ CC4yINTE | PME | CCU40_ CC4ySWS | SPM | CCU40_ CC4ySWR | RPM | CCU40_ CC4ySRS | POSR |
| | | Compare Match while counting up | CCU40_ CC4yINTS | CMUS | CCU40_ CC4yINTE | CMUE | CCU40_ CC4ySWS | SCMU | CCU40_ CC4ySWR | RCMU | CCU40_ CC4ySRS | CMSR |
| | | Compare Match while counting down | CCU40_ CC4yINTS | CMDS | CCU40_ CC4yINTE | CMDE | CCU40_ CC4ySWS | SCMD | CCU40_ CC4ySWR | RCMD | CCU40_ CC4ySRS | CMSR |
| | | One Match while counting down | CCU40_ CC4yINTS | OMDS | CCU40_ CC4yINTE | OME | CCU40_ CC4ySWS | SOM | CCU40_ CC4ySWR | ROM | CCU40_ CC4ySRS | POSR |
| | | Entering Trap State | CCU40_ CC4yINTS | TRPF | CCU40_ CC4yINTE | E2AE | CCU40_ CC4ySWS | STRPF | CCU40_ CC4ySWR | RTRPF | CCU40_ CC4ySRS | E2SR |

## Table 5-6 Interrupt Source Overview

| IRQ | Interrupt Node | Interrupt Source | Status Flag | | Interrupt Enable | | Set Flag | | Clear Flag | | Node Pointer | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Register | Bit | Register | Bit | Register | Bit | Register | Bit | Register | Bit |
| 25, 26 | CCU80_ SR[1:0] | Event 0 edge(s) information from event selector | CCU80_ CC8yINTS | E0AS | CCU80_ CC8yINTE | E0AE | CCU80_ CC8ySWS | SE0A | CCU80_ CC8ySWR | RE0A | CCU80_ CC8ySRS | E0SR |
| | | Event 1 edge(s) information from event selector | CCU80_ CC8yINTS | E1AS | CCU80_ CC8yINTE | E1AE | CCU80_ CC8ySWS | SE1A | CCU80_ CC8ySWR | RE1A | CCU80_ CC8ySRS | E1SR |
| | | Event 2 edge(s) information from event selector | CCU80_ CC8yINTS | E2AS | CCU80_ CC8yINTE | E2AE | CCU80_ CC8ySWS | SE2A | CCU80_ CC8ySWR | RE2A | CCU80_ CC8ySRS | E2SR |
| | | Period Match while counting up | CCU80_ CC8yINTS | PMUS | CCU80_ CC8yINTE | PME | CCU80_ CC8ySWS | SPM | CCU80_ CC8ySWR | RPM | CCU80_ CC8ySRS | POSR |
| | | Compare Match while counting up from compare channel 1 | CCU80_ CC8yINTS | CMU1S | CCU80_ CC8yINTE | CMU1E | CCU80_ CC8ySWS | SCM1U | CCU80_ CC8ySWR | RCM1U | CCU80_ CC8ySRS | CM1SR |
| | | Compare Match while counting down from compare channel 1 | CCU80_ CC8yINTS | CMD1S | CCU80_ CC8yINTE | CMD1E | CCU80_ CC8ySWS | SCM1D | CCU80_ CC8ySWR | RCM1D | CCU80_ CC8ySRS | CM1SR |
| | | Compare Match while counting up from compare channel 2 | CCU80_ CC8yINTS | CMU2S | CCU80_ CC8yINTE | CMU2E | CCU80_ CC8ySWS | SCM2U | CCU80_ CC8ySWR | RCM2U | CCU80_ CC8ySRS | CM2SR |
| | | Compare Match while counting down from compare channel 2 | CCU80_ CC8yINTS | CMD2S | CCU80_ CC8yINTE | CMD2E | CCU80_ CC8ySWS | SCM2D | CCU80_ CC8ySWR | RCM2D | CCU80_ CC8ySRS | CM2SR |
| | | One Match while counting down | CCU80_ CC8yINTS | OMDS | CCU80_ CC8yINTE | OME | CCU80_ CC8ySWS | SOM | CCU80_ CC8ySWR | ROM | CCU80_ CC8ySRS | POSR |
| | | Entering Trap State | CCU80_ CC8yINTS | TRPF | CCU80_ CC8yINTE | E2AE | CCU80_ CC8ySWS | STRPF | CCU80_ CC8ySWR | RTRPF | CCU80_ CC8ySRS | E2SR |

## Table 5-6 Interrupt Source Overview

| IRQ | Interrupt Node | Interrupt Source | Status Flag | | Interrupt Enable | | Set Flag | | Clear Flag | | Node Pointer | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Register | Bit | Register | Bit | Register | Bit | Register | Bit | Register | Bit |
| 27, 28 | POSIF. SR[1:0] | Transition at Hall inputs | POSIF0_ PFLG | HIES | POSIF0_ PFLGE | EHIE | POSIF0_ SPFLG | SHIE | POSIF0_ RPFLG | RHIE | POSIF0_ PFLGE | HIESEL |
| | | Occurrence of correct Hall event | POSIF0_ PFLG | CHES | POSIF0_ PFLGE | ECHE | POSIF0_ SPFLG | SCHE | POSIF0_ RPFLG | RCHE | POSIF0_ PFLGE | CHESEL |
| | | Occurrence of wrong Hall event | POSIF0_ PFLG | WHES | POSIF0_ PFLGE | EWHE | POSIF0_ SPFLG | SWHE | POSIF0_ RPFLG | RWHE | POSIF0_ PFLGE | WHESEL |
| | | Shadow transfer of MCM pattern | POSIF0_ PFLG | MSTS | POSIF0_ PFLGE | EMST | POSIF0_ SPFLG | SMST | POSIF0_ RPFLG | RMST | POSIF0_ PFLGE | MSTSEL |
| | | Index event detection | POSIF0_ PFLG | INDXS | POSIF0_ PFLGE | EINDX | POSIF0_ SPFLG | SINDX | POSIF0_ RPFLG | RINDX | POSIF0_ PFLGE | INDSEL |
| | | Phase detection error | POSIF0_ PFLG | ERRS | POSIF0_ PFLGE | EERR | POSIF0_ SPFLG | SERR | POSIF0_ RPFLG | RERR | POSIF0_ PFLGE | ERRSEL |
| | | Quadrature clock generation | POSIF0_ PFLG | CNTS | POSIF0_ PFLGE | ECNT | POSIF0_ SPFLG | SCNT | POSIF0_ RPFLG | RCNT | POSIF0_ PFLGE | CNTSEL |
| | | Period clock generation | POSIF0_ PFLG | PCLKS | POSIF0_ PFLGE | EPCLK | POSIF0_ SPFLG | SPCLK | POSIF0_ RPFLG | RPCLK | POSIF0_ PFLGE | PCLSEL |
| | | Direction change | POSIF0_ PFLG | DIRS | POSIF0_ PFLGE | EDIR | POSIF0_ SPFLG | SDIR | POSIF0_ RPFLG | RDIR | POSIF0_ PFLGE | DIRSEL |
| 29, 30 | Reserved | | | | | | | | | | | |
| 31 | 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 | - | - |
| | | 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 | - | - |

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.





Subject to Agreement on the Use of Product Information

# 6 Event Request Unit (ERU)

As described in the Service Request Processing chapter, XMC1300 uses the Event Request Unit (ERU) to support the programmable interconnection for the processing of service requests.

## 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.

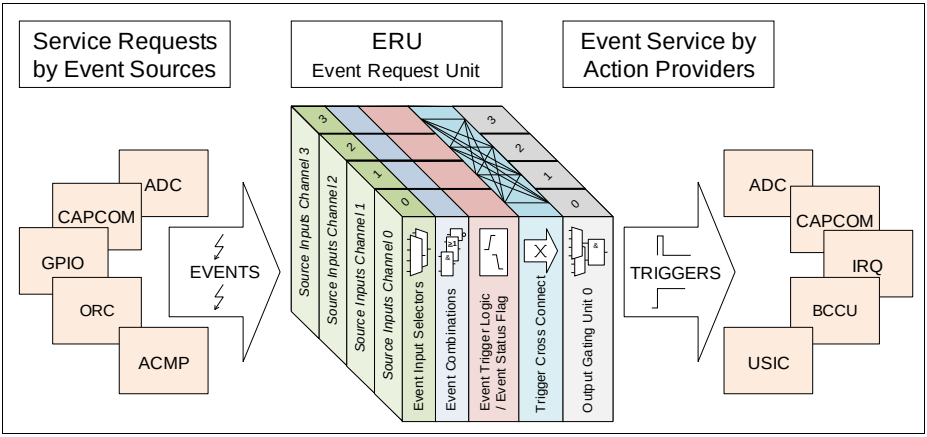



**Figure 6-1 Event Request Unit Overview**

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.

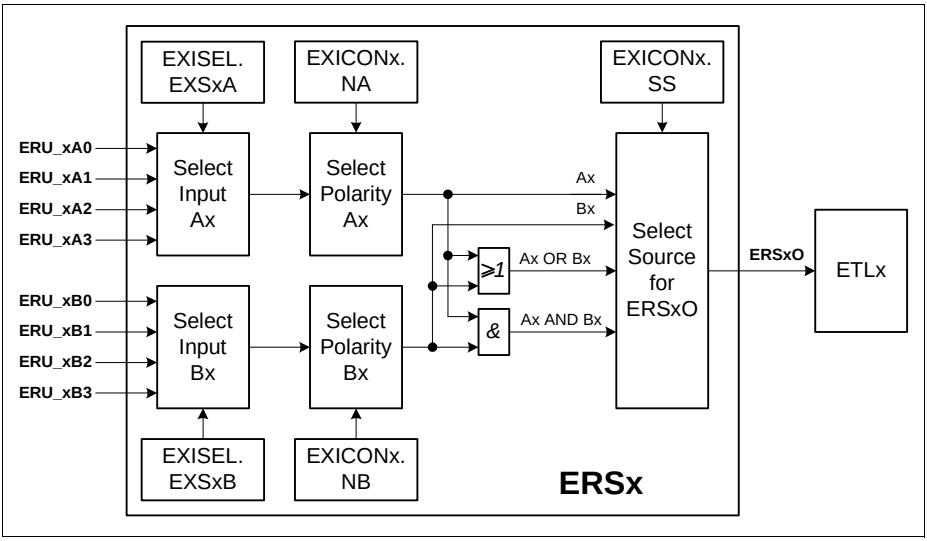



**Figure 6-2 Event Request Select Unit Overview**

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

## 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 (e.g. to handle a toggling input).

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

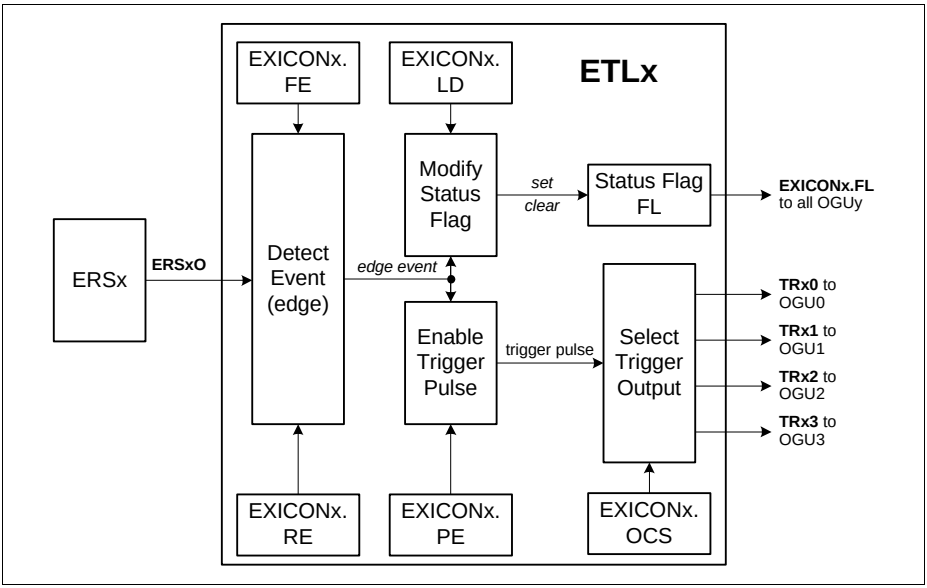



**Figure 6-3 Event Trigger Logic Overview**

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 6-4**) 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 6-4** 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.


Subject to Agreement on the Use of Product Information

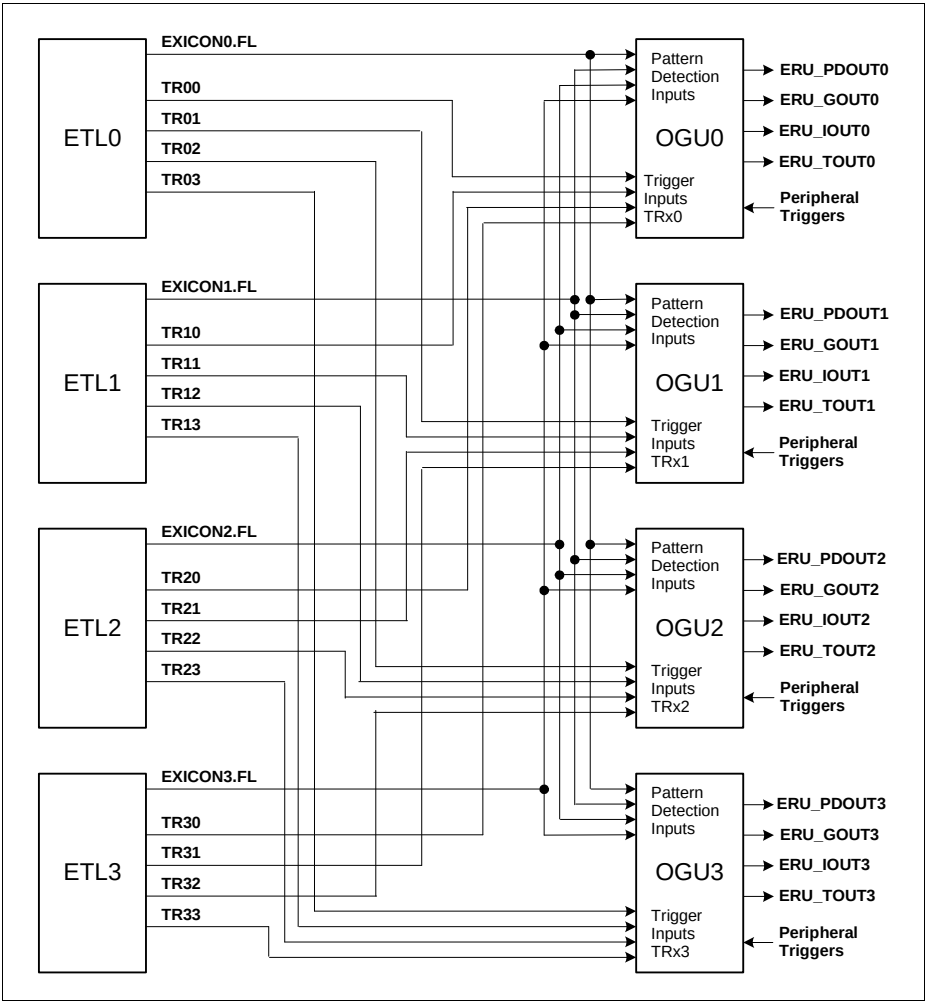



**Figure 6-4 ERU Cross Connect Matrix**

## 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 6-5** illustrates the logic blocks within an OGUy unit. All functions of an OGUy unit are controlled by its associated

EXOCONy register, e.g. **ERU0_EXOCONx (x=0-3)** for OGU0. The function of an OGUy unit can be split into two parts:

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

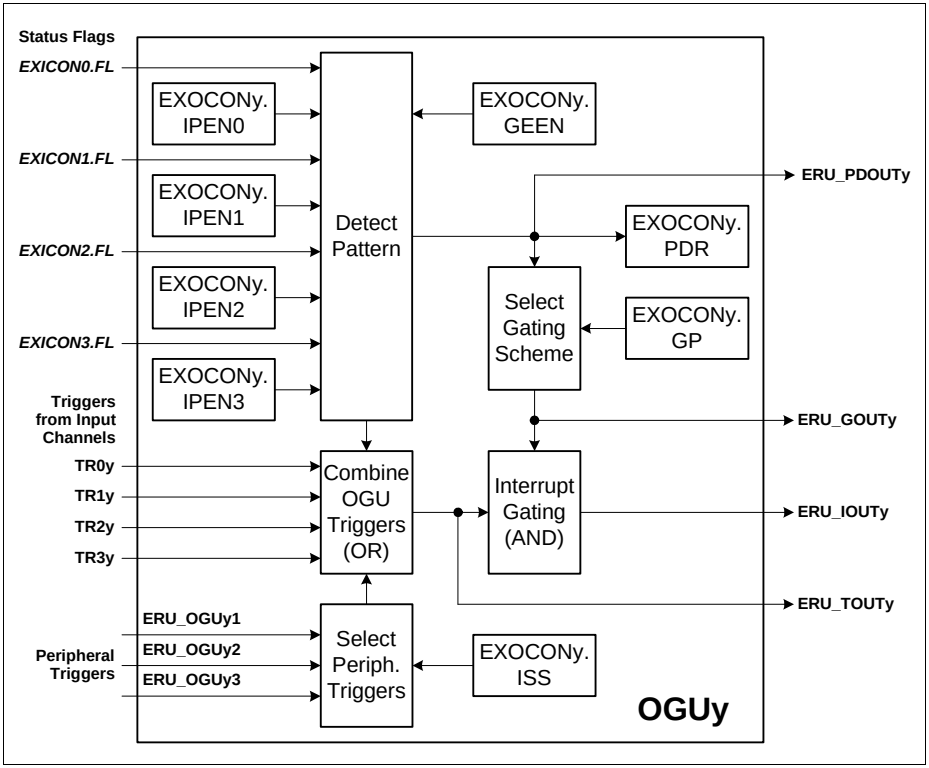



**Figure 6-5 Output Gating Unit for Output Channel y**

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

- **ERU_PDOUTy** to directly output the pattern match information for gating purposes in other modules (pattern match = 1).

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

### Trigger Combination

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

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

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

The selection is defined by the bit fields ISS in registers **ERU0_EXOCONx (x=0-3)**.

### 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.


Subject to Agreement on the Use of Product Information

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

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

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

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

## 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.


Subject to Agreement on the Use of Product Information

## 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 ERU0 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 ERU0 must be programmed to process and route the request

**Enabling External Requests**

- Selected PORTS must be programmed for input
- ERU0 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.*


Subject to Agreement on the Use of Product Information

## 6.9 Registers

**Table 6-1 Registers Address Space**

| Module | Base Address | End Address | Note |
|---|---|---|---|
| ERU0 | $4001\ 0600_H$ | $4001\ 06FF_H$ | |

### Registers Overview

The absolute register address is calculated by adding:

Module Base Address + Offset Address

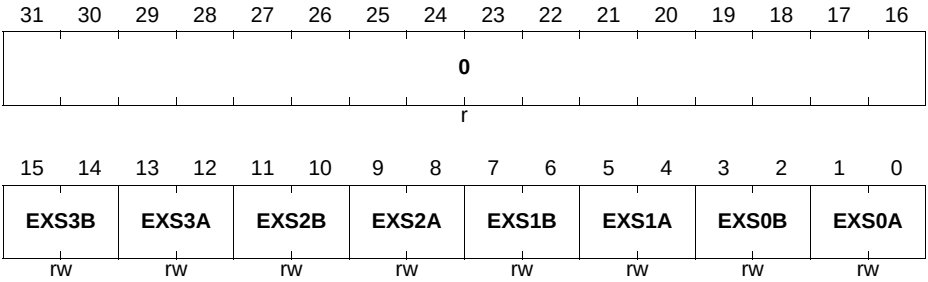
**Table 6-2 Register Overview**

| Short Name | Description | Offset Address | Access Mode | | Description See |
|---|---|---|---|---|---|
| | | | Read | Write | |
| EXISEL | ERU External Input Control Selection | $0000_H$ | U, PV | U, PV | Page 6-11 |
| EXICON0 | ERU External Input Control Selection | $0010_H$ | U, PV | U, PV | Page 6-13 |
| EXICON1 | ERU External Input Control Selection | $0014_H$ | U, PV | U, PV | Page 6-13 |
| EXICON2 | ERU External Input Control Selection | $0018_H$ | U, PV | U, PV | Page 6-13 |
| EXICON3 | ERU External Input Control Selection | $001C_H$ | U, PV | U, PV | Page 6-13 |
| EXOCON0 | ERU Output Control Register | $0020_H$ | U, PV | U, PV | Page 6-15 |
| EXOCON1 | ERU Output Control Register | $0024_H$ | U, PV | U, PV | Page 6-15 |
| EXOCON2 | ERU Output Control Register | $0028_H$ | U, PV | U, PV | Page 6-15 |
| EXOCON3 | ERU Output Control Register | $002C_H$ | U, PV | U, PV | Page 6-15 |


Subject to Agreement on the Use of Product Information

### 6.9.1 ERU Registers

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




| Field | Bits | Type | Description |
|---|---|---|---|
| **EXS0A** | [1:0] | rw | **Event Source Select for A0 (ERS0)**<br>This bit field defines which input is selected for A0.<br>$00_B$ Input ERU_0A0 is selected<br>$01_B$ Input ERU_0A1 is selected<br>$10_B$ Input ERU_0A2 is selected<br>$11_B$ Input ERU_0A3 is selected |
| **EXS0B** | [3:2] | rw | **Event Source Select for B0 (ERS0)**<br>This bit field defines which input is selected for B0.<br>$00_B$ Input ERU_0B0 is selected<br>$01_B$ Input ERU_0B1 is selected<br>$10_B$ Input ERU_0B2 is selected<br>$11_B$ Input ERU_0B3 is selected |
| **EXS1A** | [5:4] | rw | **Event Source Select for A1 (ERS1)**<br>This bit field defines which input is selected for A1.<br>$00_B$ Input ERU_1A0 is selected<br>$01_B$ Input ERU_1A1 is selected<br>$10_B$ Input ERU_1A2 is selected<br>$11_B$ Input ERU_1A3 is selected |
| **EXS1B** | [7:6] | rw | **Event Source Select for B1 (ERS1)**<br>This bit field defines which input is selected for B1.<br>$00_B$ Input ERU_1B0 is selected<br>$01_B$ Input ERU_1B1 is selected<br>$10_B$ Input ERU_1B2 is selected<br>$11_B$ Input ERU_1B3 is selected |

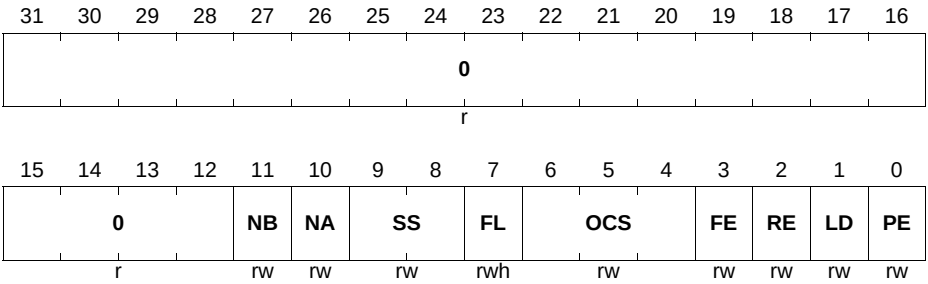

Subject to Agreement on the Use of Product Information

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


Subject to Agreement on the Use of Product Information

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

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




| Field | Bits | Type | Description |
|---|---|---|---|
| **PE** | 0 | rw | **Output Trigger Pulse Enable for ETLx**<br>This bit enables the generation of an output trigger pulse at TRxy when the selected edge is detected (set condition for the status flag FL).<br>$0_B$ The trigger pulse generation is disabled<br>$1_B$ The trigger pulse generation is enabled |
| **LD** | 1 | rw | **Rebuild Level Detection for Status Flag for ETLx**<br>This bit selects if the status flag FL is used as “sticky” bit or if it rebuilds the result of a level detection.<br>$0_B$ The status flag FL is not cleared by hardware and is used as “sticky” bit. Once set, it is not influenced by any edge until it becomes cleared by software.<br>$1_B$ The status flag FL rebuilds a level detection of the desired event. It becomes automatically set with a rising edge if RE = 1 or with a falling edge if FE = 1. It becomes automatically cleared with a rising edge if RE = 0 or with a falling edge if FE = 0. |
| **RE** | 2 | rw | **Rising Edge Detection Enable ETLx**<br>This bit enables/disables the rising edge event as edge event as set condition for the status flag FL or as possible trigger pulse for TRxy.<br>$0_B$ A rising edge is not considered as edge event<br>$1_B$ A rising edge is considered as edge event |

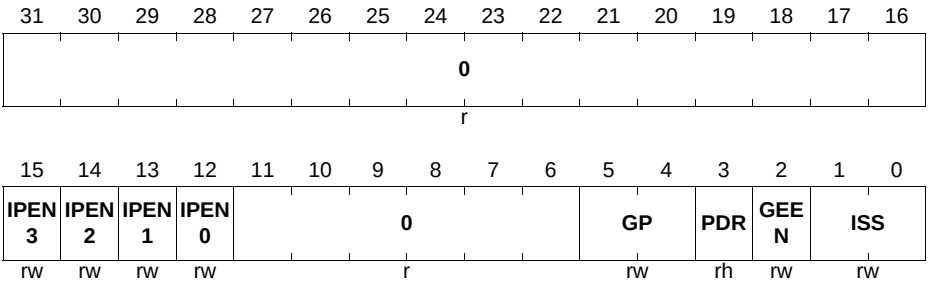

Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **FE** | 3 | rw | **Falling Edge Detection Enable ETLx**<br>This bit enables/disables the falling edge event as edge event as set condition for the status flag FL or as possible trigger pulse for TRxy.<br>$0_B$ A falling edge is not considered as edge event<br>$1_B$ A falling edge is considered as edge event |
| **OCS** | [6:4] | rw | **Output Channel Select for ETLx Output Trigger Pulse**<br>This bit field defines which Output Channel OGUy is targeted by an enabled trigger pulse TRxy.<br>$000_B$ Trigger pulses are sent to OGU0<br>$001_B$ Trigger pulses are sent to OGU1<br>$010_B$ Trigger pulses are sent to OGU2<br>$011_B$ Trigger pulses are sent to OGU3<br>Others: Reserved, do not use this combination |
| **FL** | 7 | rwh | **Status Flag for ETLx**<br>This bit represents the status flag that becomes set or cleared by the edge detection.<br>$0_B$ The enabled edge event has not been detected<br>$1_B$ The enabled edge event has been detected |
| **SS** | [9:8] | rw | **Input Source Select for ERSx**<br>This bit field defines which logical combination is taken into account as ERSxO.<br>$00_B$ Input A without additional combination<br>$01_B$ Input B without additional combination<br>$10_B$ Input A OR input B<br>$11_B$ Input A AND input B |
| **NA** | 10 | rw | **Input A Negation Select for ERSx**<br>This bit selects the polarity for the input A.<br>$0_B$ Input A is used directly<br>$1_B$ Input A is inverted |
| **NB** | 11 | rw | **Input B Negation Select for ERSx**<br>This bit selects the polarity for the input B.<br>$0_B$ Input B is used directly<br>$1_B$ Input B is inverted |
| **0** | [31:12] | r | **Reserved**<br>Read as 0; should be written with 0. |


Subject to Agreement on the Use of Product Information

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

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




| Field | Bits | Type | Description |
|---|---|---|---|
| **ISS** | [1:0] | rw | **Internal Trigger Source Selection**<br>This bit field defines which input is selected as peripheral trigger input for OGUy.<br>$00_B$ The peripheral trigger function is disabled<br>$01_B$ Input ERU_OGUy1 is selected<br>$10_B$ Input ERU_OGUy2 is selected<br>$11_B$ Input ERU_OGUy3 is selected |
| **GEEN** | 2 | rw | **Gating Event Enable**<br>Bit GEEN enables the generation of a trigger event when the result of the pattern detection changes from match to miss or vice-versa.<br>$0_B$ The event detection is disabled<br>$1_B$ The event detection is enabled |
| **PDR** | 3 | rh | **Pattern Detection Result Flag**<br>This bit represents the pattern detection result.<br>$0_B$ A pattern miss is detected<br>$1_B$ A pattern match is detected |


Subject to Agreement on the Use of Product Information

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

## 6.10 Interconnects

This section describes how the ERU0 module is connected within the XMC1300 system.


Subject to Agreement on the Use of Product Information

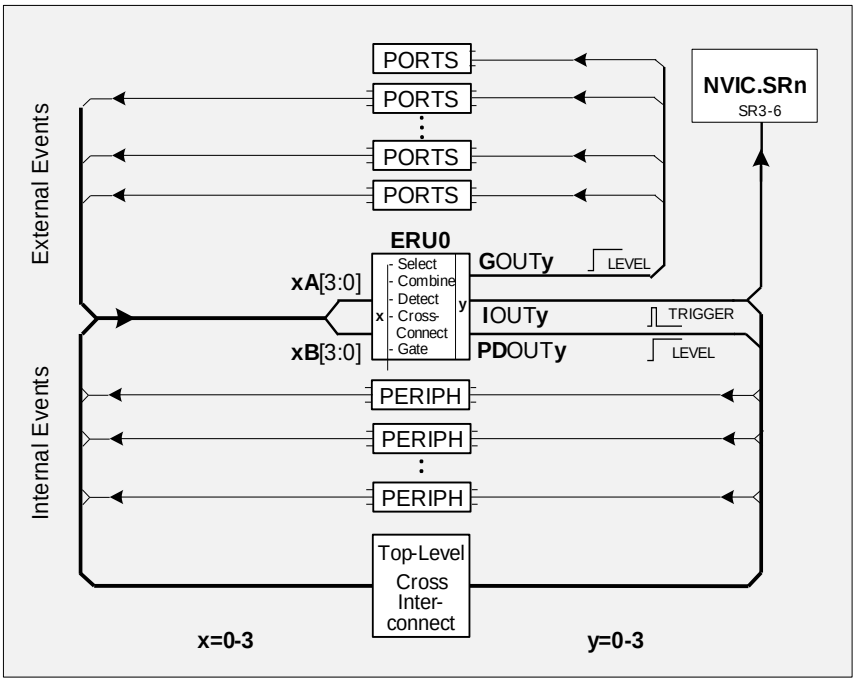



**Figure 6-6 ERU Interconnects Overview**

### 6.10.1 ERU0 Connections

The following table shows the ERU0 connections.

**Table 6-3 ERU0 Pin Connections**

| Global Inputs/Outputs | Connected To | I/O | Description |
|---|---|---|---|
| ERU0.0A0 | ACMP0.OUT | I | |
| ERU0.0A1 | P2.4 | I | |
| ERU0.0A2 | ORC2.OUT | I | |
| ERU0.0A3 | VADC0.G0BFLOUT0 | I | from ADC boundary flag |
| ERU0.0B0 | P2.0 | I | |
| ERU0.0B1 | P2.2 | I | |


Subject to Agreement on the Use of Product Information

**Table 6-3 ERU0 Pin Connections**

| Global Inputs/Outputs | Connected To | I/O | Description |
|---|---|---|---|
| ERU0.0B2 | ORC0.OUT | I | |
| ERU0.0B3 | VADC0.G1BFLOUT0 | I | from ADC boundary flag |
| ERU0.1A0 | ACMP1.OUT | I | |
| ERU0.1A1 | P2.5 | I | |
| ERU0.1A2 | ORC3.OUT | I | |
| ERU0.1A3 | VADC0.G0BFLOUT1 | I | from ADC boundary flag |
| ERU0.1B0 | P2.1 | I | |
| ERU0.1B1 | P2.3 | I | |
| ERU0.1B2 | ORC1.OUT | I | |
| ERU0.1B3 | VADC0.G1BFLOUT1 | I | from ADC boundary flag |
| ERU0.2A0 | ACMP2.OUT | I | |
| ERU0.2A1 | P2.6 | I | |
| ERU0.2A2 | ORC4.OUT | I | |
| ERU0.2A3 | VADC0.G0BFLOUT2 | I | from ADC boundary flag |
| ERU0.2B0 | P2.10 | I | |
| ERU0.2B1 | P2.11 | I | |
| ERU0.2B2 | ORC2.OUT | I | |
| ERU0.2B3 | VADC0.G1BFLOUT2 | I | from ADC boundary flag |
| ERU0.3A0 | ORC7.OUT | I | |
| ERU0.3A1 | P2.7 | I | |
| ERU0.3A2 | ORC5.OUT | I | |
| ERU0.3A3 | VADC0.G0BFLOUT3 | I | from ADC boundary flag |
| ERU0.3B0 | P2.9 | I | |
| ERU0.3B1 | P2.8 | I | |
| ERU0.3B2 | ORC6.OUT | I | |
| ERU0.3B3 | VADC0.G1BFLOUT3 | I | from ADC boundary flag |
| ERU0.OGU01 | CCU40.SR0 | I | |
| ERU0.OGU02 | VADC0.C0SR2 | I | |
| ERU0.OGU03 | CCU80.SR2 | I | |
| ERU0.OGU11 | CCU40.SR1 | I | |


Subject to Agreement on the Use of Product Information

**Table 6-3 ERU0 Pin Connections**

| Global Inputs/Outputs | Connected To | I/O | Description |
|---|---|---|---|
| ERU0.OGU12 | VADC0.C0SR2 | I | |
| ERU0.OGU13 | CCU80.SR2 | I | |
| ERU0.OGU21 | CCU40.SR2 | I | |
| ERU0.OGU22 | VADC0.C0SR3 | I | |
| ERU0.OGU23 | CCU80.SR3 | I | |
| ERU0.OGU31 | CCU40.SR3 | I | |
| ERU0.OGU32 | VADC0.C0SR3 | I | |
| ERU0.OGU33 | CCU80.SR3 | I | |
| ERU0.PDOUT0 | VADC0.BGREQGTO<br>VADC0.G0REQGTO<br>VADC0.G1REQGTO<br>CCU40.IN1D<br>CCU40.IN0J<br>CCU80.IN0F<br>CCU80.IN1L<br>CCU80.IN2L<br>CCU80.IN3L<br>POSIF0.IN0D<br>USIC0_CH1.HWIN0<br>P0.0<br>P2.11 | O | |
| ERU0.GOUT0 | P0.0<br>P2.11 | O | |
| ERU0.TOUT0 | not connected | O | |
| ERU0.IOUT0 | NVIC.ERU0.SR0<br>SCU.RTC_extclk<br>VADC0.BGREQTRM<br>VADC0.G0REQTRM<br>VADC0.G1REQTRM<br>CCU40.CLKB<br>CCU40.IN0K<br>CCU80.CLKB<br>CCU80.IN0G<br>POSIF0.EWHEB<br>BCCU0.TRAPINE | O | |


Subject to Agreement on the Use of Product Information

**Table 6-3 ERU0 Pin Connections**

| Global Inputs/Outputs | Connected To | I/O | Description |
|---|---|---|---|
| ERU0.PDOUT1 | VADC0.BGREQGTP<br>VADC0.G0REQGTP<br>VADC0.G1REQGTP<br>CCU40.IN0D<br>CCU40.IN1J<br>CCU80.IN0L<br>CCU80.IN1F<br>POSIF0.IN1D<br>USIC0_CH1.HWIN2<br>P0.1<br>P2.10 | O | |
| ERU0.GOUT1 | P0.1<br>P2.10 | O | |
| ERU0.TOUT1 | not connected | O | |
| ERU0.IOUT1 | NVIC.ERU0.SR1<br>VADC0.BGREQTRN<br>VADC0.G0REQTRN<br>VADC0.G1REQTRN<br>CCU40.CLKC<br>CCU40.IN1K<br>CCU80.CLKC<br>CCU80.IN1G<br>BCCU0.TRAPINF | O | |
| ERU0.PDOUT2 | VADC0.BGREQGTK<br>VADC0.G0REQGTK<br>VADC0.G1REQGTK<br>CCU40.IN3D<br>CCU40.IN2J<br>CCU80.IN2F<br>POSIF0.IN2D<br>P0.2<br>P2.1 | O | |
| ERU0.GOUT2 | P0.2<br>P2.1 | O | |
| ERU0.TOUT2 | not connected | O | |


Subject to Agreement on the Use of Product Information

**Table 6-3 ERU0 Pin Connections**

| Global Inputs/Outputs | Connected To | I/O | Description |
|---|---|---|---|
| ERU0.IOUT2 | NVIC.ERU0.SR2<br>VADC0.BGREQTRG<br>VADC0.G0REQTRG<br>VADC0.G1REQTRG<br>CCU40.IN2K<br>CCU80.IN2G<br>POSIF0.MSETF<br>BCCU0.TRAPING | O | |
| ERU0.PDOUT3 | VADC0.BGREQGTL<br>VADC0.G0REQGTL<br>VADC0.G1REQGTL<br>CCU40.IN3J<br>CCU40.IN2D<br>CCU80.IN3F<br>P0.3<br>P2.0 | O | |
| ERU0.GOUT3 | P0.3<br>P2.0 | O | |
| ERU0.TOUT3 | not connected | O | |
| ERU0.IOUT3 | NVIC.ERU0.SR3<br>VADC0.BGREQTRH<br>VADC0.G0REQTRH<br>VADC0.G1REQTRH<br>CCU40.IN3K<br>CCU80.IN3G<br>POSIF0.EWHEC<br>POSIF0.MSETG<br>BCCU0.TRAPINH | O | |


Subject to Agreement on the Use of Product Information

# 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 7-1** shows a block diagram of the MATH Coprocessor.

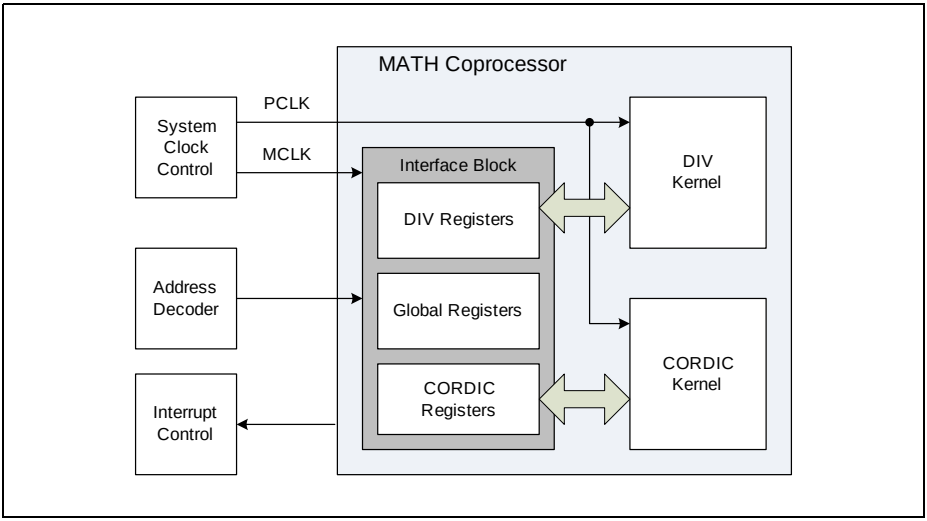



**Figure 7-1 MATH Coprocessor Block Diagram**

## 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


Subject to Agreement on the Use of Product Information

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 7-2** shows the timing diagram for a division operation.

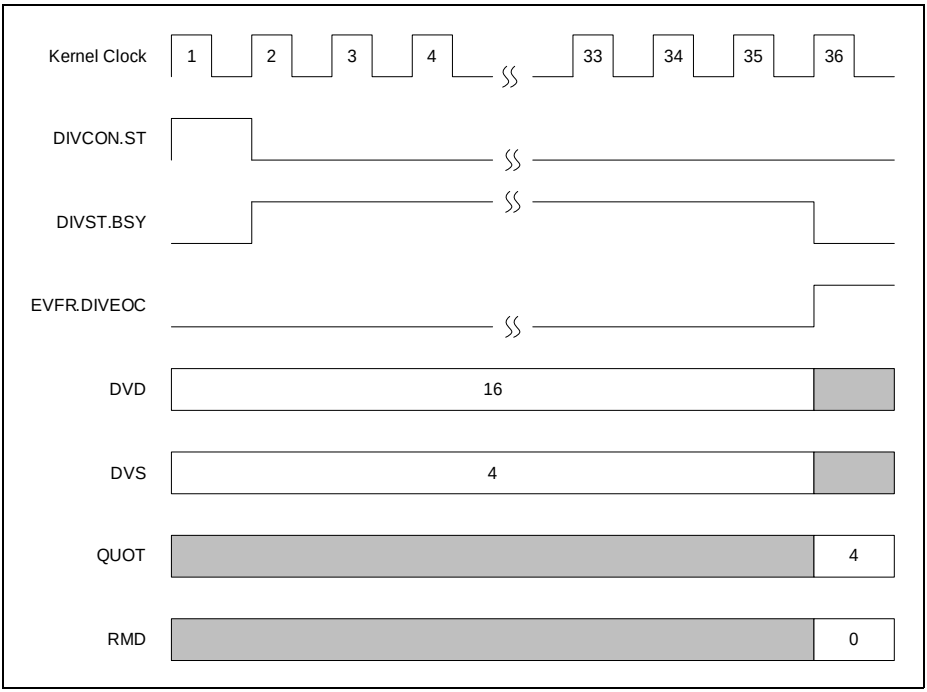



**Figure 7-2 Timing Diagram for a Division Operation**

*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 7-3**.


Subject to Agreement on the Use of Product Information

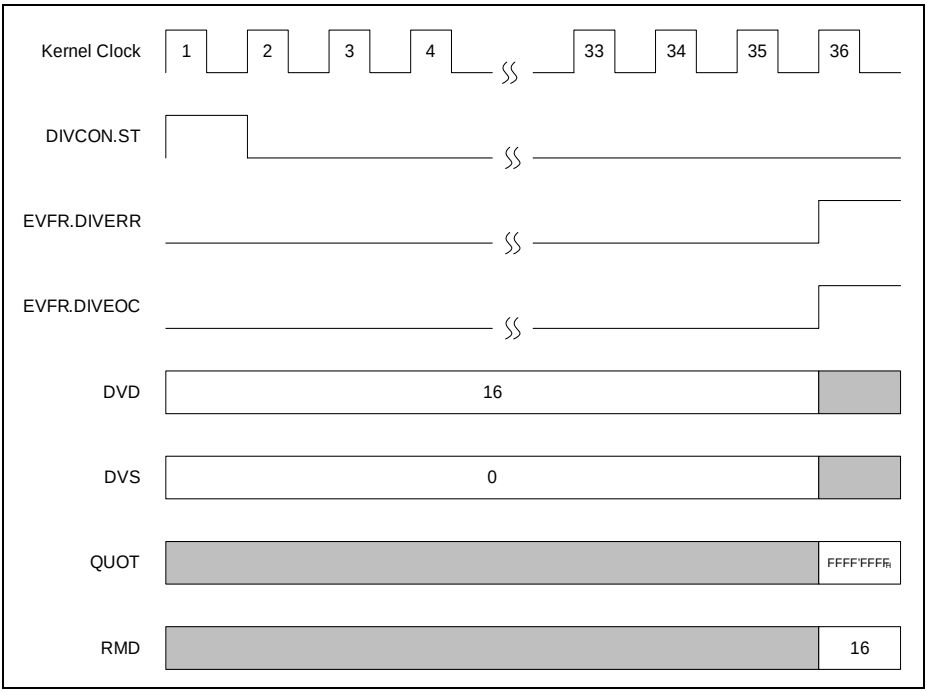



**Figure 7-3 Timing Diagram with Divide by Zero Error**

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 **Section 7.2.3**) 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 7-1** is detected.

**Table 7-1 Overflow Error Conditions**

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

**Table 7-1 Overflow Error Conditions**

| DIVCON.DIVMODE | Dividend (Hex) | Dividend (Dec) | Divisor (Hex) | Divisor (Dec) |
|---|---|---|---|---|
| $01_B$ | $8000'0000_H$ | $-2^{31}$ | $FFFF_H$ | -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.

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.


Subject to Agreement on the Use of Product Information

## 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:

$$x_{i+1} = x_i - m \cdot d_i \cdot y_i \cdot 2^{-i} \quad (7.1)$$

$$y_{i+1} = y_i + d_i \cdot x_i \cdot 2^{-i} \quad (7.2)$$

$$z_{i+1} = z_i - d_i \cdot e_i \quad (7.3)$$

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 7-2 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 7-4** shows a block diagram of the CORDIC Coprocessor kernel.


Subject to Agreement on the Use of Product Information

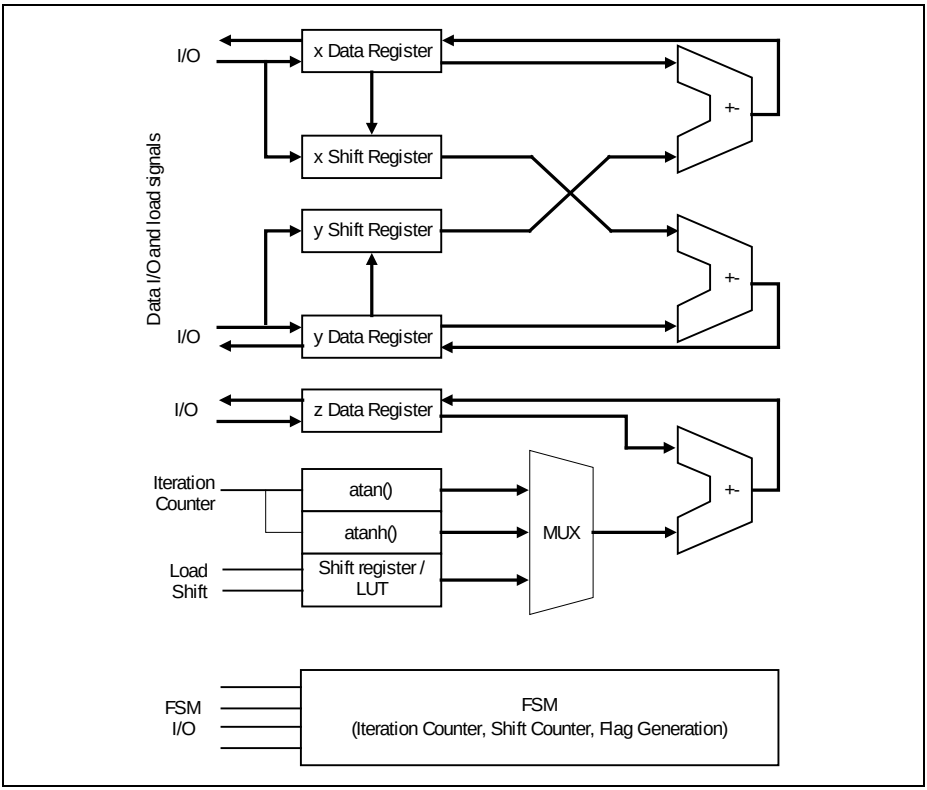



**Figure 7-4 CORDIC Block Diagram**

### 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,


Subject to Agreement on the Use of Product Information

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:

$$\text{X or Y Result Data} = \frac{\text{CORDIC Calculated Data}}{\text{MPS}}$$

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:

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

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

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 7-3**.

**Table 7-3 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 7-4 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:

$$x_{i+1} = x_i - m \cdot d_i \cdot y_i \cdot 2^{-i} \quad (7.1)$$

$$y_{i+1} = y_i + d_i \cdot x_i \cdot 2^{-i} \quad (7.2)$$

$$z_{i+1} = z_i - d_i \cdot e_i \quad (7.3)$$

**Table 7-4 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**<br>$m$ = 1<br>$e_i$ = atan($2^{-i}$) | $X_{final}$ = K[$X$ cos($Z$) - $Y$ sin($Z$)] / MPS<br>$Y_{final}$ = K[$Y$ cos($Z$) + $X$ sin($Z$)] / MPS<br>$Z_{final}$ = 0<br>where K ≈ 1.646760258121 | $X_{final}$ = K sqrt($X^2$+$Y^2$) / MPS<br>$Y_{final}$ = 0<br>$Z_{final}$ = $Z$ + atan($Y$ / $X$)<br>where K ≈ 1.646760258121 |
| | For solving cos($Z$) and sin($Z$), set $X$ = 1 / K, $Y$ = 0.<br>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.<br>Useful domain: Full range of $X$ and $Y$ supported due to pre- and post-processing logic.<br><br>For solving atan($Y$ / $X$), set $Z$ = 0.<br>Useful domain: Full range of $X$ and $Y$, except $X$ = 0. |
| | Relationships:<br>tan(v) = sin(v) / cos(v) | Relationships:<br>acos(w) = atan[sqrt(1-$w^2$) / w]<br>asin(w) = atan[w / sqrt(1-$w^2$)] |
| **Linear**<br>$m$ = 0<br>$e_i$ = $2^{-i}$ | $X_{final}$ = $X$ / MPS<br>$Y_{final}$ = [$Y$ + $X$ $Z$] / MPS<br>$Z_{final}$ = 0 | $X_{final}$ = $X$ / MPS<br>$Y_{final}$ = 0<br>$Z_{final}$ = $Z$ + $Y$ / $X$ |
| | For solving $X \cdot Z$, set $Y$ = 0.<br>Useful domain: $\|Z\| \leq 2$. | For solving ratio $Y$ / $X$, set $Z$ = 0.<br>Useful domain: $\|Y / X\| \leq 2$, $X > 0$. |

**Table 7-4 CORDIC Coprocessor Operating Modes and Corresponding Result Data**

| Function | Rotation Mode | Vectoring Mode |
|---|---|---|
| **Hyperbolic**<br>$m$ = -1<br>$e_i$ = atanh($2^{-i}$) | $X_{final}$ = k[$X$ cosh($Z$) + $Y$ sinh($Z$)] / MPS<br>$Y_{final}$ = k[$Y$ cosh($Z$) + $X$ sinh($Z$)] / MPS<br>$Z_{final}$ = 0<br>where k ≈ 0.828159360960 | $X_{final}$ = k sqrt($X^2$-$Y^2$) / MPS<br>$Y_{final}$ = 0<br>$Z_{final}$ = $Z$ + atanh($Y$ / $X$)<br>where k ≈ 0.828159360960 |
| | For solving cosh($Z$) and sinh($Z$) and $e^Z$, set $X$ = 1 / k, $Y$ = 0.<br>Useful domain: \|$Z$\| ≤ 1.11rad, $Y$ = 0. | For solving sqrt($x^2$-$y^2$), set $X$ = x / k, $Y$ = y / k.<br>Useful domain: \|y\| < \|x\|, $X$ > 0.<br><br>For solving atanh($Y$ / $X$), set $Z$ = 0.<br>Useful domain: \|atanh($Y$ / $X$)\| ≤ 1.11rad, $X$ > 0. |
| | Relationships:<br>tanh(v) = sinh(v) / cosh(v)<br>$e^v$ = sinh(v) + cosh(v)<br>$w^t$ = $e^{t\ ln(w)}$ | Relationships:<br>ln(w) = 2 atanh[(w-1) / (w+1)]<br>sqrt(w) = sqrt((w+0.25)$^2$-(w-0.25)$^2$)<br>acosh(w) = ln[w+sqrt(1-$w^2$)]<br>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 7-4** 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 **Section 7.3.3.1**. For result data accuracy, refer to **Section 7.3.5**.
- 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 $[-\pi,((2^{23}-1)/2^{23})\pi]$ are represented by integer values ranging $[-2^{23},(2^{23}-1)]$. Therefore, Z data is limited (not considering domain of convergence) to represent angles $[-\pi,((2^{23}-1)/2^{23})\pi]$ 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 $\sum d_i \cdot e_i$, where $e_i$ is always decreasing for iteration i. In other words, $|Z| \leq$ Sum of LUT. In circular function, this means $|Z| \leq$ integer value representing 1.74 radians. For linear function, $|Z| \leq 2$. In hyperbolic function, $|Z| \leq$ 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 $|\text{atan}(Y / X)| \leq 1.74$ radians. For linear function, $|Y / X| \leq 2$. For hyperbolic function, $|\text{atanh}(Y / X)| \leq 1.11$ 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.


Subject to Agreement on the Use of Product Information

**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 $|Z| \leq \pi / 2$ to prevent possible Z result data overflow.

*Note: Considerations should also be given to function limitations such as the meaning of the result data, e.g. divide by zero is not meaningful. The 'useful domain' included within **Table 7-4** 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 **Chapter 7.3.5** 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.


Subject to Agreement on the Use of Product Information

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 **Chapter 7.3.2.2** for details on data normalization.

**Table 7-5** summarizes the initial and result data format.

**Table 7-5 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 | | |


Subject to Agreement on the Use of Product Information

### 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 **Chapter 7.3.7.1** and **Chapter 7.3.7.2**.

For input data values within the specified useful domain (see **Table 7-4**), 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 7-6** 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.


Subject to Agreement on the Use of Product Information

**Table 7-6 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%<br>1 : 2.1150%<br>2 : 0.0010%<br>3 : 0.0010%<br>ND for X ≤ 3 | 0 : 69.6738%<br>1 : 29.8307%<br>2 : 0.4919%<br>3 : 0.0036%<br>ND for Z ≤ 3 |
| **Circular Rotation** | Input conditions: Useful Domain (Full range of X, Y and Z) | |
| | 0 : 47.8958%<br>1 : 50.3470%<br>2 : 1.7561%<br>3 : 0.0006%<br>4 : 0.0003%<br>5 : 0.0002%<br>ND for X ≤ 5 | 0 : 47.8773%<br>1 : 50.3626%<br>2 : 1.7438%<br>3 : 0.0103%<br>4 : 0.0052%<br>5 : 0.0008%<br>ND for Y ≤ 5 |
| **Linear Vectoring** | Input conditions: Useful Domain (\|Y / X\| ≤ 2, X > 0) | |
| | 0 : 99.5869%<br>1 : 0.4131%<br>ND for X ≤ 1 | 0 : 47.6909%<br>1 : 52.2723%<br>2 : 0.0239%<br>3 : 0.0129%<br>ND for Z ≤ 3 |
| **Linear Rotation** | Input conditions: Useful Domain (\|Z\| ≤ 2) | |
| | 0 : 100%<br>ND for X ≤ 0 | 0 : 48.2181%<br>1 : 50.7801%<br>2 : 0.9996%<br>3 : 0.0022%<br>ND for Y ≤ 3 |
| **Hyperbolic Vectoring** | Input conditions: Useful Domain (\|Y\| < \|X\|, X > 0, \|atanh(Y / X)\| ≤ 1.11rad) | |
| | 0 : 98.1092%<br>1 : 1.8855%<br>2 : 0.0017%<br>3 : 0.0019%<br>4 : 0.0017%<br>ND for X ≤ 4 | 0 : 47.7010%<br>1 : 51.2954%<br>2 : 0.9938%<br>3 : 0.0098%<br>ND for Z ≤ 3 |


Subject to Agreement on the Use of Product Information

**Table 7-6 Normalized Deviation of a Calculation**

| Mode | X Normalized Deviation | Y or Z Normalized Deviation |
|---|---|---|
| **Hyperbolic Rotation** | Input conditions: Useful Domain (\|Z\| ≤ 1.11rad, Y = 0) | |
| | 0 : 47.6358%<br>1 : 51.1029%<br>2 : 1.2578%<br>3 : 0.0012%<br>4 : 0.0007%<br>5 : 0.0006%<br>6 : 0.0005%<br>7 : 0.0002%<br>8 : 0.0003%<br>ND for X ≤ 8 | 0 : 47.5419%<br>1 : 51.2064%<br>2 : 1.2478%<br>3 : 0.0013%<br>4 : 0.0010%<br>5 : 0.0007%<br>6 : 0.0006%<br>7 : 0.0003%<br>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, e.g. 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.


Subject to Agreement on the Use of Product Information

### 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 7-7**

**Table 7-7 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 |
| 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 7-8**

**Table 7-8 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 |


Subject to Agreement on the Use of Product Information

**Table 7-8 Precomputed Scaled Values for atanh($2^{-i}$)**

| Iteration No. | Scaled atanh($2^{-i}$) in hex | Iteration No. | Scaled atanh($2^{-i}$) in hex |
|---|---|---|---|
| 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.


Subject to Agreement on the Use of Product Information

## 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 7-5** shows an example of chaining the DIV quotient result to the CORDIC X-operand.


Subject to Agreement on the Use of Product Information

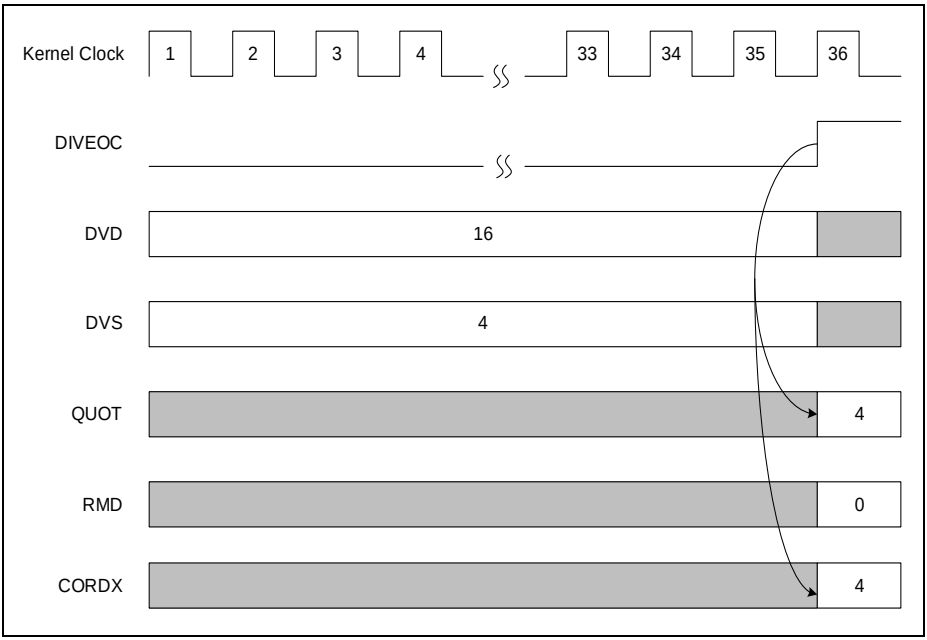



**Figure 7-5 Division Operation with CORDXRC = $01_B$**

### 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 **Section 7.2.2.1** and **Section 7.3.2**). 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 7-5**, 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 7-6** shows the interrupt structure in the MATH Coprocessor.


Subject to Agreement on the Use of Product Information

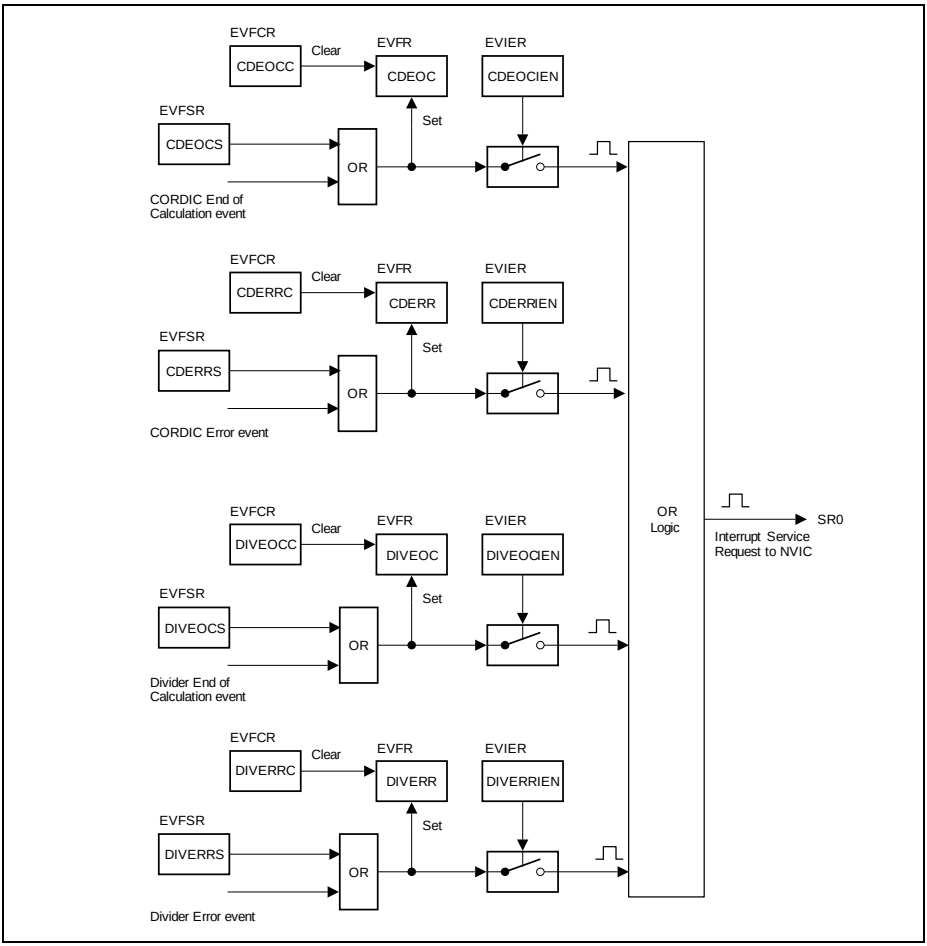



**Figure 7-6 Interrupt Structure**

## 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 XMC1300, 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.


Subject to Agreement on the Use of Product Information

## 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 7-10** supports both 16-bit and 32-bit bus access. All other registers (Global and CORDIC) supports only 32-bit access.*

**Table 7-9 Registers Address Space**

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

**Table 7-10 Register Overview**

| Short Name | Description | Offset Addr. | Access Mode | | Description See |
|---|---|---|---|---|---|
| | | | Read | Write | |
| **Global Registers** | | | | | |
| Reserved | Reserved | $0000_H$ | BE | BE | |
| GLBCON | Global Control Register | $0004_H$ | U, PV | U, PV | Page 7-28 |
| ID | Module Identification Register | $0008_H$ | U, PV | BE | Page 7-31 |
| EVIER | Event Interrupt Enable Register | $000C_H$ | U, PV | U, PV | Page 7-32 |
| EVFR | Event Flag Register | $0010_H$ | U, PV | U, PV | Page 7-33 |
| EVFSR | Event Flag Set Register | $0014_H$ | U, PV | U, PV | Page 7-34 |
| EVFCR | Event Flag Clear Register | $0018_H$ | U, PV | U, PV | Page 7-34 |
| Reserved | Reserved | $001C_H$ | BE | BE | |
| **Divider Registers** | | | | | |
| DVD | Dividend Register | $0020_H$ | U, PV | U, PV | Page 7-36 |
| DVS | Divisor Register | $0024_H$ | U, PV | U, PV | Page 7-36 |
| QUOT | Quotient Register | $0028_H$ | U, PV | BE | Page 7-36 |
| RMD | Dividend Register | $002C_H$ | U, PV | BE | Page 7-37 |
| DIVST | Divider Status Register | $0030_H$ | U, PV | BE | Page 7-37 |
| DIVCON | Divider Control Register | $0034_H$ | U, PV | U, PV | Page 7-38 |
| Reserved | Reserved | $0038_H$ - $003F_H$ | BE | BE | |

**Table 7-10 Register Overview** (cont'd)

| Short Name | Description | Offset Addr. | Access Mode | | Description See |
|---|---|---|---|---|---|
| | | | Read | Write | |
| **CORDIC Registers** | | | | | |
| STATC | Status and Data Control Register | $0040_H$ | U, PV | U, PV | **Page 7-41** |
| CON | Control Register | $0044_H$ | U, PV | U, PV | **Page 7-42** |
| CORDX | X Data Register | $0048_H$ | U, PV | U, PV | **Page 7-44** |
| CORDY | Y Data Register | $004C_H$ | U, PV | U, PV | **Page 7-44** |
| CORDZ | Z Data Register | $0050_H$ | U, PV | U, PV | **Page 7-44** |
| CORRX | X Result Register | $0054_H$ | U, PV | BE | **Page 7-45** |
| CORRY | Y Result Register | $0058_H$ | U, PV | BE | **Page 7-45** |
| CORRZ | Z Result Register | $005C_H$ | U, PV | BE | **Page 7-46** |

## 7.8.1 Global Registers Description

### GLBCON

The GLBCON register contains the global control bits for the MATH.

**GLBCON**
**Global Control Register** **($0004_H$)** **Reset Value: $0000\ 0000_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | CORDZRC | | 0 | CORDYRC | | 0 | CORDXRC | | DVSRC | | | DVDRC | | |
| r | | rw | | r | rw | | r | rw | | rw | | | rw | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **DVDRC** | [2:0] | rw | **Dividend Register Result Chaining**<br>The DVD register in DIV will be updated with the selected result register value when the result chaining trigger event occurs.<br>$000_B$ No result chaining is selected<br>$001_B$ QUOT register is the selected source<br>$010_B$ RMD register is the selected source<br>$011_B$ CORRX is the selected source<br>$100_B$ CORRY is the selected source<br>$101_B$ CORRZ is the selected source<br>$110_B$ Reserved<br>$111_B$ Reserved |
| **DVSRC** | [5:3] | rw | **Divisor Register Result Chaining**<br>The DVS register in DIV will be updated with the selected result register value when the result chaining trigger event occurs.<br>$000_B$ No result chaining is selected<br>$001_B$ QUOT register is the selected source<br>$010_B$ RMD register is the selected source<br>$011_B$ CORRX is the selected source<br>$100_B$ CORRY is the selected source<br>$101_B$ CORRZ is the selected source<br>$110_B$ Reserved<br>$111_B$ Reserved |
| **CORDXRC** | [7:6] | rw | **CORDX Register Result Chaining**<br>The CORDX register in CORDIC will be updated with the selected result register value when the result chaining trigger event occurs.<br>$00_B$ No result chaining is selected<br>$01_B$ QUOT register is the selected source<br>$10_B$ RMD register is the selected source<br>$11_B$ Reserved |

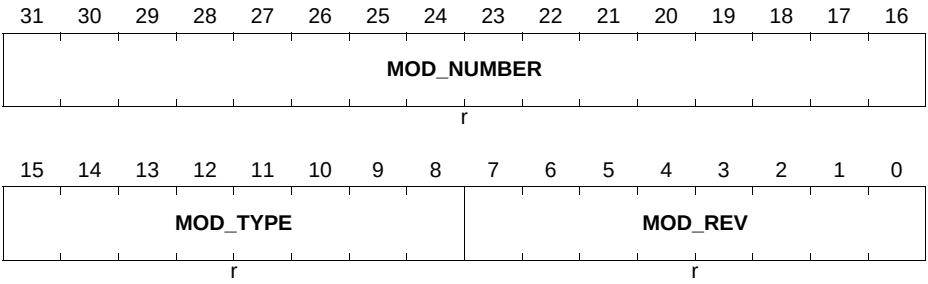
| Field | Bits | Type | Description |
|---|---|---|---|
| **CORDYRC** | [10:9] | rw | **CORDY Register Result Chaining**<br>The CORDY register in CORDIC will be updated with the selected result register value when the result chaining trigger event occurs.<br>$00_B$ No result chaining is selected<br>$01_B$ QUOT register is the selected source<br>$10_B$ RMD register is the selected source<br>$11_B$ Reserved |
| **CORDZRC** | [13:12] | rw | **CORDZ Register Result Chaining**<br>The CORDZ register in CORDIC will be updated with the selected result register value when the result chaining trigger event occurs.<br>$00_B$ No result chaining is selected<br>$01_B$ QUOT register is the selected source<br>$10_B$ RMD register is the selected source<br>$11_B$ Reserved |
| **SUSCFG** | [17:16] | rw | **Suspend Mode Configuration**<br>This bit determines if a suspend mode is entered by the MATH Coprocessor when the CPU is halted.<br>$00_B$ Suspend mode is never entered.<br>$01_B$ Hard suspend mode will be entered when CPU is halted.<br>$10_B$ Soft suspend mode will be entered when CPU is halted.<br>$11_B$ Reserved |
| **0** | [31:18], [15:14], 11, 8 | r | **Reserved**<br>Read as 0; should be written with 0. |

### MATH_ID

The MATH_ID register indicate the function and the design step of the MATH Coprocessor.

### MATH_ID
### Module Identification Register

(0008$_H$) Reset Value: 00F2 C0XX$_H$




| Field | Bits | Type | Description |
|---|---|---|---|
| **MOD_REV** | [7:0] | r | **Module Revision Number**<br>MOD_REV defines the revision number. The value of a module revision starts with 01$_H$ (first revision). |
| **MOD_TYPE** | [15:8] | r | **Module Type**<br>This bit field is C0$_H$. It defines the module as a 32-bit module. |
| **MOD_NUMBER** | [31:16] | r | **Module Number Value**<br>This bit field defines the module identification number. |

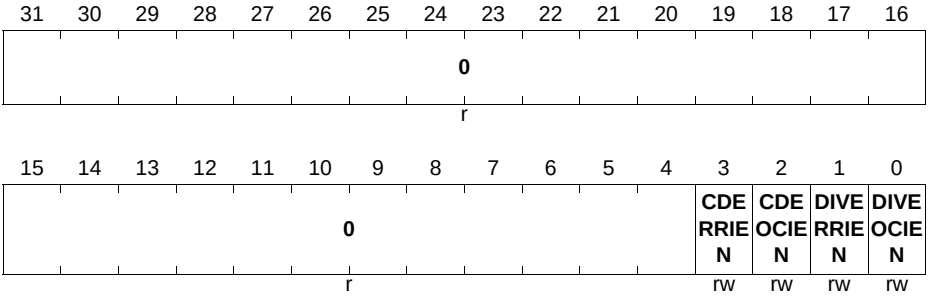
### Event Interrupt Enable Register

This register enables interrupt generation for each event. If enabled, the detection of the event will generate the interrupt pulse through the service request output.


Subject to Agreement on the Use of Product Information

**EVIER**

**Event Interrupt Enable Register (000C$_H$)** **Reset Value: 0000 0000$_H$**



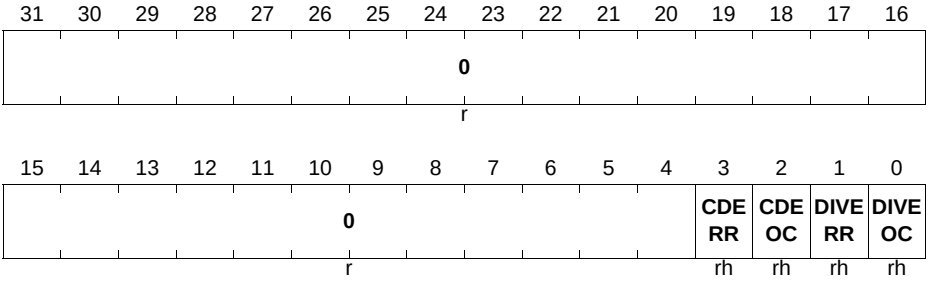

| Field | Bits | Type | Description |
|---|---|---|---|
| **DIVEOCIEN** | 0 | rw | **Divider End of Calculation Interrupt Enable**<br>$0_B$ Divider end of calculation interrupt generation is disabled.<br>$1_B$ Divider end of calculation interrupt generation is enabled. |
| **DIVERRIEN** | 1 | rw | **Divider Error Interrupt Enable**<br>$0_B$ Divider error interrupt generation is disabled<br>$1_B$ Divider error interrupt generation is enabled |
| **CDEOCIEN** | 2 | rw | **CORDIC End of Calculation Interrupt Enable**<br>$0_B$ CORDIC end of calculation interrupt generation is disabled.<br>$1_B$ CORDIC end of calculation interrupt generation is enabled. |
| **CDERRIEN** | 3 | rw | **CORDIC Error Interrupt Enable**<br>$0_B$ CORDIC error interrupt generation is disabled<br>$1_B$ CORDIC error interrupt generation is enabled |
| **0** | [31:4] | r | **Reserved**<br>Read as 0; should be written with 0. |

### Event Flag Register

This register contains the status flags for each event. If set, it indicates that the event has been detected.


Subject to Agreement on the Use of Product Information

**EVFR**
**Event Flag Register** **($0010_H$)** **Reset Value: $0000\ 0000_H$**



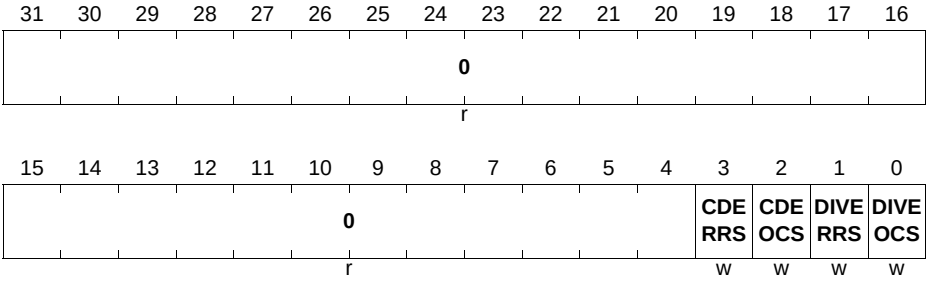

| Field | Bits | Type | Description |
|---|---|---|---|
| **DIVEOC** | 0 | rh | **Divider End of Calculation Event Flag**<br>$0_B$ Divider end of calculation event has not been detected.<br>$1_B$ Divider end of calculation event has been detected. |
| **DIVERR** | 1 | rh | **Divider Error Event Flag**<br>$0_B$ Divider error event has not been detected<br>$1_B$ Divider error event has been detected |
| **CDEOC** | 2 | rh | **CORDIC End of Calculation Event Flag**<br>$0_B$ CORDIC end of calculation event has not been detected.<br>$1_B$ CORDIC end of calculation event has been detected. |
| **CDERR** | 3 | rh | **CORDIC Error Event Flag**<br>$0_B$ CORDIC error event has not been detected<br>$1_B$ CORDIC error event has been detected |
| **0** | [31:4] | r | **Reserved**<br>Read as 0; should be written with 0. |

**Event Flag Set Register**

This register allows the application to set the status flags of each event in EVFR register, as if the event has been detected. If interrupt generation for that event is enabled previously in EVIER register, an interrupt service request will generated to the NVIC.

Writing 0 to these bits has no effect while reading always returns 0.

**EVFSR**
**Event Flag Set Register** **($0014_H$)** **Reset Value: 0000 0000$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **DIVEOCS** | 0 | w | **Divider End of Calculation Event Flag Set**<br>$0_B$ No effect.<br>$1_B$ Sets the Divider end of calculation event flag in EVFR register. Interrupt will be generated if enabled in EVIER register. |
| **DIVERRS** | 1 | w | **Divider Error Event Flag Set**<br>$0_B$ No effect.<br>$1_B$ Sets the Divider error event flag in EVFR register. Interrupt will be generated if enabled in EVIER register. |
| **CDEOCS** | 2 | w | **CORDIC Event Flag Set**<br>$0_B$ No effect.<br>$1_B$ Sets the CORDIC end of calculation event flag in EVFR register. Interrupt will be generated if enabled in EVIER register. |
| **CDERRS** | 3 | w | **CORDIC Error Event Flag Set**<br>$0_B$ No effect.<br>$1_B$ Sets the CORDIC error event flag in EVFR register. Interrupt will be generated if enabled in EVIER register. |
| **0** | [31:4] | r | **Reserved**<br>Read as 0; should be written with 0. |

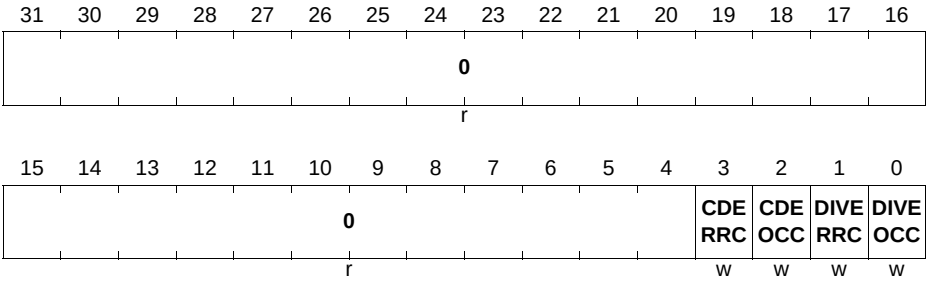

Subject to Agreement on the Use of Product Information

### Event Flag Clear Register

The event flags in EVFR register is cleared by writing a 1 to the corresponding bits in this register.

Writing 0 to these bits has no effect while reading always returns 0.

**EVFCR**
**Event Flag Clear Register** **($0018_H$)** **Reset Value: $0000\ 0000_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **DIVEOCC** | 0 | w | **Divider End of Calculation Event Flag Clear**<br>$0_B$ No effect.<br>$1_B$ Clears the Divider end of calculation event flag in EVFR register. |
| **DIVERRC** | 1 | w | **Divider Error Event Flag Clear**<br>$0_B$ No effect.<br>$1_B$ Clears the Divider error event flag in EVFR register. |
| **CDEOCC** | 2 | w | **CORDIC End of Calculation Event Flag Clear**<br>$0_B$ No effect.<br>$1_B$ Clears the CORDIC end of calculation event flag in EVFR register. |
| **CDERRC** | 3 | w | **CORDIC Error Event Flag Clear**<br>$0_B$ No effect.<br>$1_B$ Clears the CORDIC error event flag in EVFR register. |
| **0** | [31:4] | r | **Reserved**<br>Read as 0; should be written with 0. |

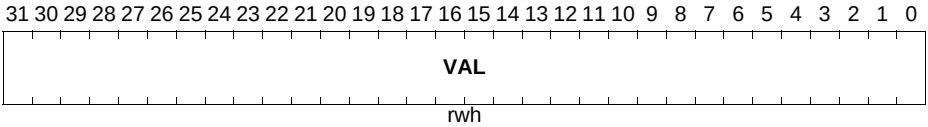

Subject to Agreement on the Use of Product Information

## 7.8.2 Divider Registers Description

### DVD

The DVD register is used to store the dividend operand of the division. It can be written by software and if result chaining is enabled, also by hardware.

**DVD**
**Dividend Register** **($0020_H$)** **Reset Value: $0000\ 0000_H$**



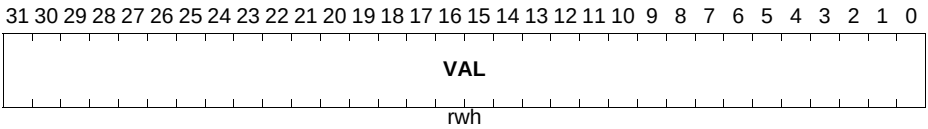

| Field | Bits | Type | Description |
|---|---|---|---|
| **VAL** | [31:0] | rwh | **Dividend Value** |

### DVS

The DVS register is used to store the divisor operand of the division. It can be written by software and if result chaining is enabled, also by hardware.

*Note: A division operation can be started by a write to DVS while DIVCON.STMODE = 0.*

**DVS**
**Divisor Register** **($0024_H$)** **Reset Value: $0000\ 0000_H$**



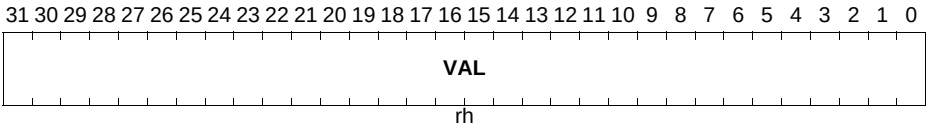

| Field | Bits | Type | Description |
|---|---|---|---|
| **VAL** | [31:0] | rwh | **Divisor Value** |

### QUOT

The QUOT register is used to store the quotient result of the division. It will be automatically updated by hardware upon each completion of a division operation.

*Note: Wait states will be inserted on the bus if the QUOT register is read while BSY = 1. The bus read transaction will be completed only when the new QUOT value becomes available at the end of the active calculation/*

**QUOT**
**Quotient Register** **($0028_H$)** **Reset Value: 0000 0000$_H$**



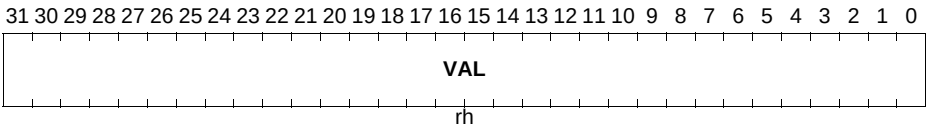

| Field | Bits | Type | Description |
|---|---|---|---|
| **VAL** | [31:0] | rh | **Quotient Value** |

**RMD**

The RMD register is used to store the remainder result of the division. It will be automatically updated by hardware upon each completion of a division operation.

*Note: Wait states will be inserted on the bus if the RMD register is read while BSY = 1. The bus read transaction will be completed only when the new RMD value becomes available at the end of the active calculation/*

**RMD**
**Remainder Register** **($002C_H$)** **Reset Value: 0000 0000$_H$**



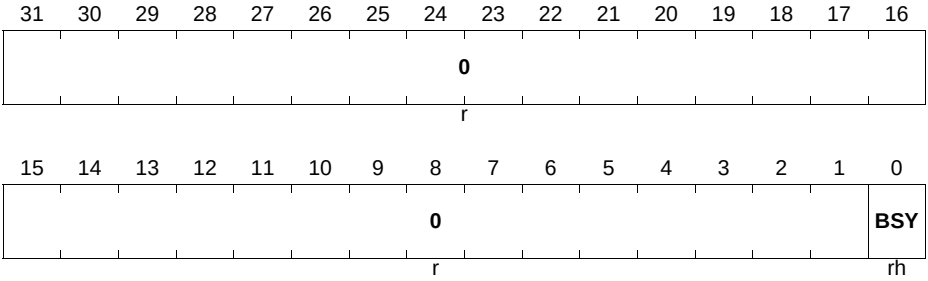

| Field | Bits | Type | Description |
|---|---|---|---|
| **VAL** | [31:0] | rh | **Remainder Value** |

**DIVST**

The DIVST register contains the status flags for the DIV.

**DIVST**

**Divider Status Register** **(0030$_H$)** **Reset Value: 0000 0000$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **BSY** | 0 | rh | **Busy Indication**<br>$0_B$ Divider is not running any division operation.<br>$1_B$ Divider is still running a division operation. |
| **0** | [31:1] | r | **Reserved**<br>Read as 0; should be written with 0. |

**DIVCON**

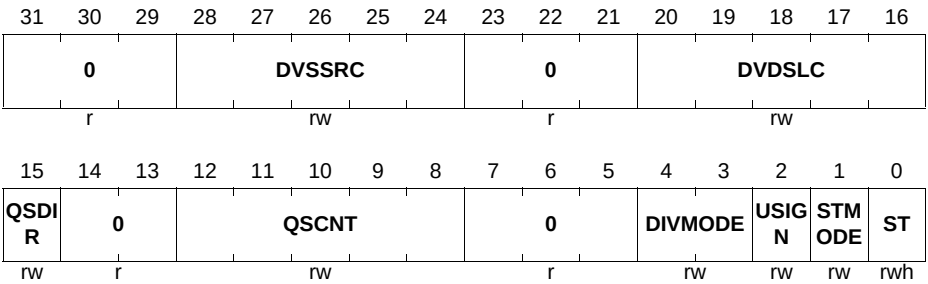
The DIVCON register contains the control bits for the DIV.

Write access to DIVCON register is ignored while BSY=1. No bus error will be generated in this case.

*Note: If result-chaining is enabled, refer also to **Section 7.4.1.2** for description on handling of the busy flag.*

**DIVCON**

**Divider Control Register** **(0034$_H$)** **Reset Value: 0000 0000$_H$**



| Field | Bits | Type | Description |
|---|---|---|---|
| **ST** | 0 | rwh | **Start Bit**<br>$0_B$ No effect<br>$1_B$ Start the division operation when STMODE=$1_B$<br>The bit is automatically cleared by hardware after one kernel clock cycle. |
| **STMODE** | 1 | rw | **Start Mode**<br>Selects the start mode for the division operation:<br>$0_B$ Calculation is automatically started with a write to DVS register<br>$1_B$ Calculation is started by setting the ST bit to 1<br>*Note: The start request for a new division operation will be ignored if BSY = 1.* |
| **USIGN** | 2 | rw | **Unsigned Division Enable**<br>$0_B$ Signed division is selected<br>$1_B$ Unsigned division is selected |
| **DIVMODE** | [4:3] | rw | **Division Mode**<br>$00_B$ 32-bit divide by 32-bit<br>$01_B$ 32-bit divide by 16-bit<br>$10_B$ 16-bit divide by 16-bit<br>$11_B$ Reserved |
| **QSDIR** | 15 | rw | **Quotient Shift Direction**<br>This bit is used to select the shift direction for the quotient after a division:<br>$0_B$ Left shift<br>$1_B$ Right shift |
| **QSCNT** | [12:8] | rw | **Quotient Shift Count**<br>If QSCNT is not equal to 0, it indicates the number of bits the quotient will be shifted by, after the division.<br>If QSCNT=0, no shift operation will take place. |
| **DVDSLC** | [20:16] | rw | **Dividend Shift Left Count**<br>If DVDSLC is not equal to 0, it indicates the number of bits the dividend will be shifted left by, prior to the division.<br>If DVDSLC=0, no shift operation will take place. |


Subject to Agreement on the Use of Product Information

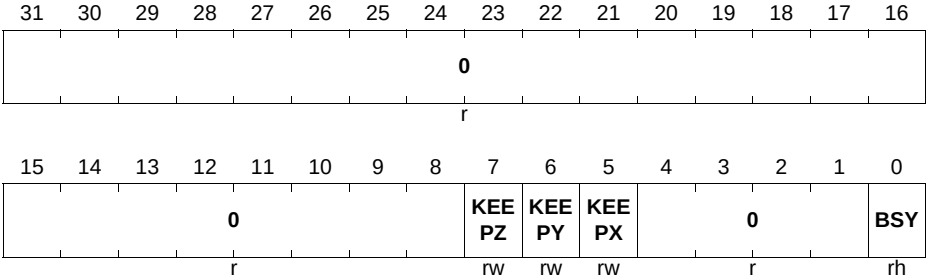
| Field | Bits | Type | Description |
|---|---|---|---|
| **DVSSRC** | [28:24] | rw | **Divisor Shift Right Count**<br>If DVSSRC is not equal to 0, it indicates the number of bits the divisor will be shifted right by, prior to the division.<br>If DVSSRC=0, no shift operation will take place. |
| **0** | [31:29], [23:21], [14:13], [7:5] | r | **Reserved**<br>Read as 0; should be written with 0. |

## 7.8.3 CORDIC Registers Description

### STATC

The STATC register generally reflects the status of the CORDIC Coprocessor. The register also contain bits for data control.

**STATC**
**CORDIC Status and Data Control Register($0040_H$)** **Reset Value: $0000\ 0000_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **BSY** | 0 | rh | **Busy Indication**<br>Indicates a running calculation when set. The flag is asserted one clock cycle after bit ST was set. It is deasserted at the end of a calculation. |
| **0** | [4:1] | r | **Reserved** |
| **KEEPX** | 5 | rw | **Last X Result as Initial Data for New Calculation**<br>If set, a new calculation will use the value of the result from the previous calculation as the initial data. In other words, the respective kernel data register will not be overwritten by the contents of the shadow data register at the beginning of the new calculation. This bit should always be cleared for the very first calculation to load the initial X data.<br>Independent of the KEEPx bit, the shadow data registers will continue to hold the last written initial data value until the next software write.<br>If KEEPx bit is set for a multi-step calculation, the accuracy of the corresponding final x result data may be reduced and is not guaranteed as shown in **Section 7.3.5**. |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **KEEPY** | 6 | rw | **Last Y Result as Initial Data for New Calculation**<br><See description for KEEPX><br>Additionally, it may not be meaningful to set this bit in Vectoring modes as the last Y result converges to 0. |
| **KEEPZ** | 7 | rw | **Last Z Result as Initial Data for New Calculation**<br><See description for KEEPX><br>Additionally, it may not be meaningful to set this bit in Rotation modes as the last Z result converges to 0. |
| **0** | [31:8] | r | **Reserved** |

### CON

The CON register allows for the general control of the CORDIC Coprocessor. Write action to this register while **STATC**.BSY is set has no effect.

*Note: If result-chaining is enabled, refer also to **Section 7.4.1.2** for description on handling of the busy flag.*

**CON**
**CORDIC Control Register** **($0044_H$)** **Reset Value: 0000 $0062_H$**

| 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 | | | | | | | | MPS | | X_USIGN | ST_MODE | ROTVEC | MODE | | ST |
| r | | | | | | | | rw | | rw | rw | rw | rw | | rwh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **ST** | 0 | rwh | **Start Calculation**<br>If ST_MODE = 1, set ST to start a CORDIC calculation. Is effective only while BSY is not set. This bit may be set with the other bits of this register in one write access.<br>Cleared by hardware at the beginning of calculation. |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **MODE** | [2:1] | rw | **Operating Mode**<br>$00_B$ Linear Mode<br>$01_B$ Circular Mode (default)<br>$10_B$ Reserved<br>$11_B$ Hyperbolic Mode |
| **ROTVEC** | 3 | rw | **Rotation Vectoring Selection**<br>$0_B$ Vectoring Mode (default)<br>$1_B$ Rotation Mode |
| **ST_MODE** | 4 | rw | **Start Method**<br>$0_B$ Auto start of calculation after write access to X parameter data register **CORDX**(default).<br>$1_B$ Start calculation only after bit ST is set |
| **X_USIGN** | 5 | rw | **Result Data Format for X in Circular Vectoring Mode**<br>When reading the X result data, X data has a data format of:<br>$0_B$ Signed, twos complement<br>$1_B$ Unsigned (default)<br>With this bit set, the MSB bit of the X result data is processed as a data bit instead of a sign bit.<br>This bit is only effective when operating in circular vectoring mode. In all other modes, X is always processed as twos complement data.<br>X_USIGN = 1 is meaningful in circular vectoring mode because the result data is always positive and always larger than the initial data. |
| **MPS** | [7:6] | rw | **X and Y Magnitude Prescaler**<br>After the last iteration of a calculation, the calculated value of X and Y are each divided by this factor to yield the result.<br>Proper setting of these bits is important to avoid an overflow of the result in the respective kernel data registers.<br>$00_B$ Divide by 1<br>$01_B$ Divide by 2 (default)<br>$10_B$ Divide by 4<br>$11_B$ Reserved, retain the last MPS setting |
| **0** | [31:8] | r | **Reserved** |

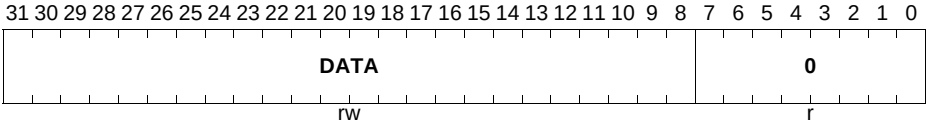

Subject to Agreement on the Use of Product Information

### CORDx

The Data registers are used to initialize the X, Y and Z parameters.

### CORDX

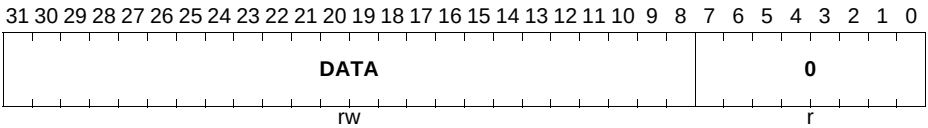
**CORDIC X Data Register** **($0048_H$)** **Reset Value: 0000 $0000_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | [7:0] | r | **Reserved** |
| **DATA** | [31:8] | rw | **Initial X Parameter Data**<br>Writing to this register with **CON**.ST_MODE = 0 starts an operation. |

### CORDY

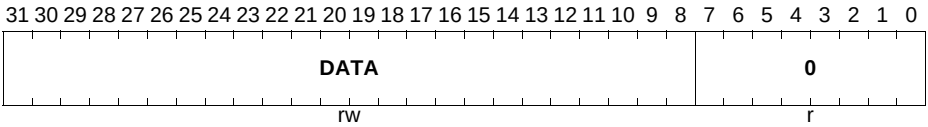
**CORDIC Y Data Register** **($004C_H$)** **Reset Value: 0000 $0000_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | [7:0] | r | **Reserved** |
| **DATA** | [31:8] | rw | **Initial Y Parameter Data** |

### CORDZ

**CORDIC Z Data Register** **($0050_H$)** **Reset Value: 0000 $0000_H$**



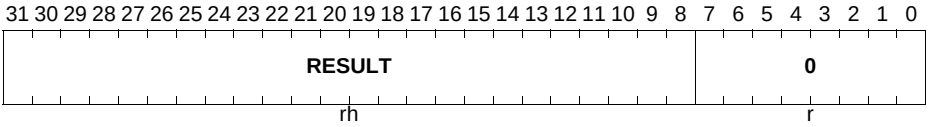
| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | [7:0] | r | **Reserved** |
| **DATA** | [31:8] | rw | **Initial Z Parameter Data** |

### CORRx

The result data from CORDIC calculation will be written to the respective result registers. Any read access on these registers while **STATC**.BSY is set (a calculation is still running) will cause the kernel to issue a bus wait until BSY is reset.

### CORRX

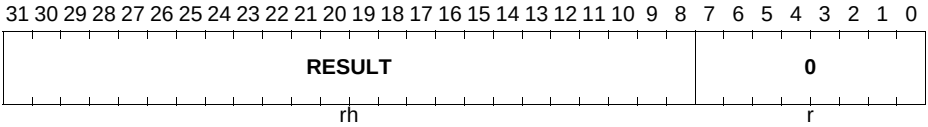
**CORDIC X Result Register** **($0054_H$)** **Reset Value: $0000\ 0000_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | [7:0] | r | **Reserved** |
| **RESULT** | [31:8] | rh | **X Calculation Result** |

### CORRY

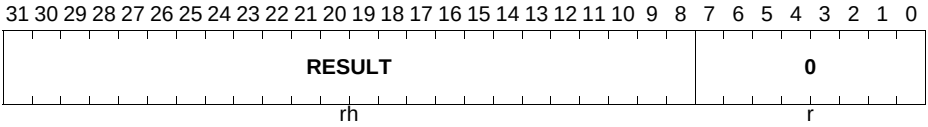
**CORDIC Y Result Register** **($0058_H$)** **Reset Value: $0000\ 0000_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | [7:0] | r | **Reserved** |
| **RESULT** | [31:8] | rh | **Y Calculation Result** |


Subject to Agreement on the Use of Product Information

**CORRZ**
**CORDIC Z Result Register** **(005C$_H$)** **Reset Value: 0000 0000$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | [7:0] | r | **Reserved** |
| **RESULT** | [31:8] | rh | **Z Calculation Result** |

## 7.9 Interconnects

**Table 7-11** shows the module interconnects:

**Table 7-11 Module Interconnects**

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

# On-Chip Memories


Subject to Agreement on the Use of Product Information

# 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 XMC1300 is based on standard ARM Cortex-M0 system memory map.

### 8.1.1 Features

The Memory Map implements the following features:

- Compatibility with standard ARM Cortex-M0 CPU **[5]**
- Full compatibility across entire XMC1000 Family

### 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.


Subject to Agreement on the Use of Product Information

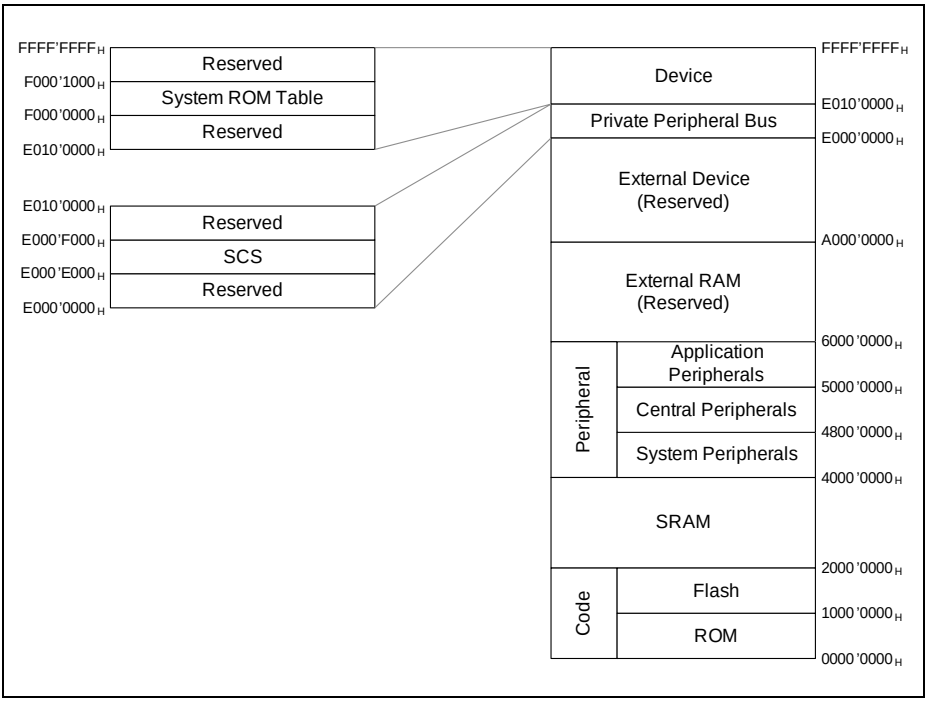



**Figure 8-1 XMC1300 Address Space**

## 8.2 Memory Regions

The XMC1300 device-specific address map contains on-chip memories and peripherals. The memory regions for XMC1300 are described in **Table 8-1**.

**Table 8-1 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 |


Subject to Agreement on the Use of Product Information

**Table 8-1 Memory Regions** (cont'd)

| Start (hex) | End (hex) | Size (hex) | Space name | Usage |
|---|---|---|---|---|
| 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 8-2** defines detailed system memory map of XMC1300 where each individual peripheral or memory instance implement its own address spaces. For detailed register description of the system components and peripherals, please refer to respective chapters of this document.

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

**Table 8-2 Memory Map**

| Address space | Address Range | Description | Access Type[1] | |
|---|---|---|---|---|
| | | | Read | Write |
| Code | $00000000_H$ - $00000BFF_H$ | ROM (user-readable) | U, PV | nBE |
| | $00000C00_H$ - $00001FFF_H$ | 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 |
| | $10001000_H$ - $10032FFF_H$ | Flash (200 Kbytes) | 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 |


Subject to Agreement on the Use of Product Information

**Table 8-2 Memory Map** (cont'd)

| Address space | Address Range | Description | Access Type[1)] | |
|---|---|---|---|---|
| | | | Read | Write |
| 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 |
| | $40011000_H$ - $4001107F_H$ | ANACTRL | U, PV | U, PV |
| | $40011080_H$ - $4001FFFF_H$ | 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 |

**Table 8-2 Memory Map** (cont'd)

| Address space | Address Range | Description | Access Type[1)] | |
|---|---|---|---|---|
| | | | Read | Write |
| System Peripherals (cont'd) | $40040000_H$ - $4004007F_H$ | 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 |
| | $40040280_H$ - $4004FFFF_H$ | 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 |
| | $48000800_H$ - $4801FFFF_H$ | reserved | BE | BE |
| | $48020000_H$ - $4802000F_H$ | PRNG | U, PV | U, PV |
| | $48020010_H$ - $4802FFFF_H$ | reserved | BE | BE |

**Table 8-2 Memory Map** (cont'd)

| Address space | Address Range | Description | Access Type[1)] | |
|---|---|---|---|---|
| | | | Read | Write |
| Central Peripherals (cont'd) | $48030000_H$ - $480303FF_H$ | 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 |
| | $48040000_H$ - $480401FF_H$ | 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 |
| | $48040500_H$ - $4FFFFFFF_H$ | reserved | BE | BE |


Subject to Agreement on the Use of Product Information

**Table 8-2 Memory Map** (cont'd)

| Address space | Address Range | Description | Access Type[1] | |
|---|---|---|---|---|
| | | | Read | Write |
| 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 |
| | $50000500_H$ - $5000FFFF_H$ | reserved | BE | BE |
| | $50010000_H$ - $500101FF_H$ | POSIF0 | U, PV | U, PV |
| | $50010200_H$ - $5002FFFF_H$ | reserved | BE | BE |
| | $50030000_H$ - $500301FF_H$ | BCCU0 | U, PV | U, PV |
| | $50030200_H$ - $57FFFFFF_H$ | reserved | BE | BE |
| Peripheral | $58000000_H$ - $5FFFFFFF_H$ | 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 type of memories in XMC1300.

### 8.4.1 Flash Memory Access

The XMC1300 provides up to 200 Kbytes 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 will insert waitstates during memory read automatically if needed.

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

### 8.4.2 SRAM Access

The XMC1300 provides 16 Kbytes 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 Kbytes of SRAM is logically divided into four blocks of 4 Kbytes each. Accesses to blocks 1, 2 and 3 can be disabled and enabled again during run-time with the peripheral privilege access scheme. See PAU chapter 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/data by the bootstrap loader into the SRAM nor should they store critical data that are still needed by the application after a system reset or SW master reset.*

### 8.4.3 ROM Access

The XMC1300 provides 8 Kbytes 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 XMC1300 are listed in **Table 8-3**.

**Table 8-3 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 XMC1300, 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 (i.e., 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 of all protected bits, and writing $10101_B$ to the bit field PASS closes access to writing of 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 8-4** shows the list of protected bit in XMC1300.

**Table 8-4 List of Protected Register Bit Fields**

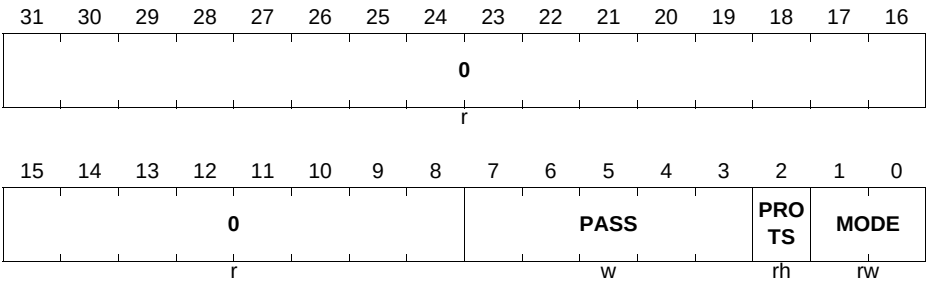
| Register | Bit Field |
|---|---|
| SCU_CLKCR | FDIV, IDIV, PCLKSEL, RTCLKSEL |
| SCU_CGATSET0 | All bits |
| SCU_CGATCLR0 | All bits |
| SCU_ANAOFFSET | ADJL_OFFSET |
| VADC0_ACCPROT0 | All bits |
| VADC0_ACCPROT1 | All bits |

Write 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 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 happen 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 SCU chapter.

**SCU_PASSWD**
**Password Register** **(4001 0024$_H$)** **Reset Value: 0000 0007$_H$**





Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **MODE** | [1:0] | rw | **Bit Protection Scheme Control Bits**<br>$00_B$ Scheme disabled - direct access to the protected bits is allowed.<br>$11_B$ Scheme enabled - the bit field PASS has to be written with the passwords to open and close the access to the protected bits. (Default)<br>Others: Scheme enabled, similar to the setting for MODE = $11_B$.<br>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**<br>This bit shows the status of the protection.<br>$0_B$ Software is able to write to all protected bits.<br>$1_B$ Software is unable to write to any of the protected bits. |
| **PASS** | [7:3] | w | **Password Bits**<br>This bit protection scheme only recognizes the following three passwords:<br>$11000_B$Enables writing of the bit field MODE.<br>$10011_B$Opens access to writing of all protected bits.<br>$10101_B$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 XMC1300, 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.


Subject to Agreement on the Use of Product Information

### 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 (e.g. disable clock gating) before they can be accessed. For details please refer to individual peripheral chapters.


Subject to Agreement on the Use of Product Information

# 9 Flash Architecture

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

## 9.1 Overview

The XMC1300 has an embedded user-programmable NVM for storage of user code and data.

### 9.1.1 Features

The NVM has the following features:

- Reading by word, writing by block and erasing by page (256 bytes) or sector (4 Kbytes).
- Fast personalization support.
- Automatic verify support.
- Up to 50,000 erase cycles per page.
- Minimum data retention of 10 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.

## 9.2 Definitions

Throughout the document the terms NVM (Non Volatile Memory) 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.


Subject to Agreement on the Use of Product Information

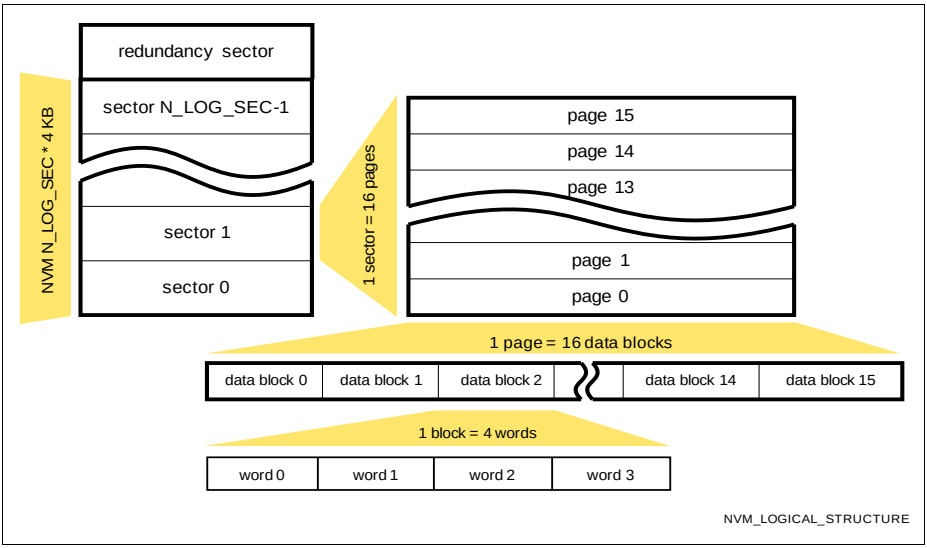



**Figure 9-1 Logical structure of the NVM module**

## 9.2.1 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 (see below).

### 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 (see below).

### Programming

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

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

## 9.2.2 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 4 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.

### 9.2.3 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 **Section 9.3.1**.

**Page Address**

Module specific part of the logical address, for calculation see **Section 9.3.1**.

### 9.2.4 Module Specific Definitions

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

**Table 9-1 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_SECTORS | 52 | Number of sectors including redundancy |
| N_LOG_SEC | 51 | Number of sectors excluding redundancy |

## 9.3 Module Components

### 9.3.1 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:

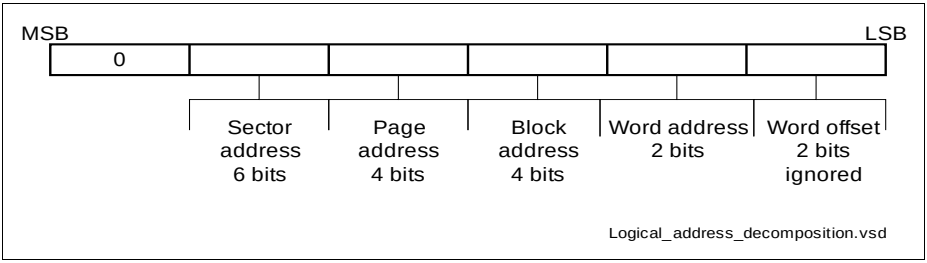



**Figure 9-2 Decomposition of Logical Address**

- 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.

#### 9.3.1.1 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.

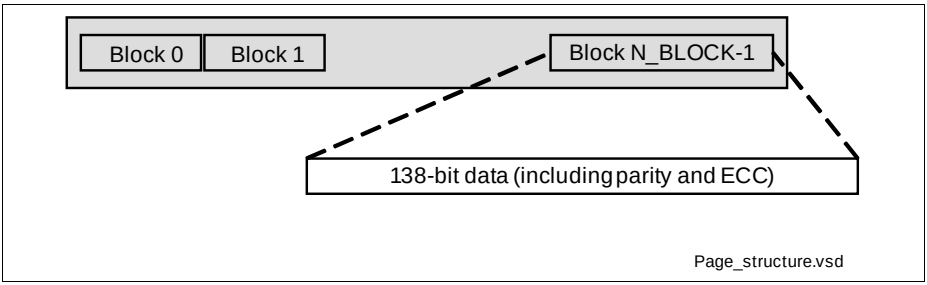



**Figure 9-3 Structure of a Page**

#### 9.3.1.2 Sector

16 pages form a sector.

The whole cell array is build of 52 sectors.

## 9.4 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.

### 9.4.1 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 **Section 9.8.2.2**.

### 9.4.2 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.

Per default the XMC1300 devices use a configuration with adaptive wait states for read accesses to the flash memory, dynamically adapting to the system frequency and flash access timing without user software interaction.

#### 9.4.2.1 Flash read access with fixed wait states

It is possible to configure the XMC1300 devices to apply fixed wait states to each flash read access, improving determinism of program execution from flash. The required number of wait states depends on the system frequency $f_{MCLK}$, as defined in the parameter $N_{FWSFLASH}$. The number of wait states can be configured with the bit **NVMCONF**.WS, the selection of adaptive or fixed wait states is done with the bit **CONFIG1**.WS.

***Attention: Any write operation to the register CONFIG1 to switch between adaptive and fixed wait states configuration must only modify the bit CONFIG1.FIXWS. Changing other bits in CONFIG1 can lead to unpredictable results.***


Subject to Agreement on the Use of Product Information

***Attention: Before and after the fixed wait states configuration or the system frequency $f_{MCLK}$ is changed, the number of selected wait states must always comply to the parameter $N_{FWSFLASH}$.***

Below is a code snippet defining the register addresses, configuring one wait state and then switching to operation with one fixed wait state.

**Example**

```
// Headers and variables to fix number of wait states to "1"
#define ADDR1 0x40050008   //Address of NVM_NVMCONF
uint32_t * NVM_NVMCONF  = (uint32_t *) ADDR1;
#define ADDR2 0x40050048   //Address of NVM_CONFIG1
uint32_t * NVM_CONFIG1  = (uint32_t *) ADDR2;

// init sequence to fix number of wait states to "1"
*NVM_NVMCONF = *NVM_NVMCONF | 0x1000; //Set .WS bit => 1WS
*NVM_CONFIG1 = *NVM_CONFIG1 | 0x0800; //Set .FIXWS bit => fixed
WS scheme
```

### 9.4.3 Memory Write

From the user's viewpoint data is written directly to the memory array. Module internally one block is buffered and the ECC bits are calculated. Then a finite state machine is started that writes the ECC and data bits to the memory field.

Writing does not support wrap-around, i.e. writing of a block has always to 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 has always 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.


Subject to Agreement on the Use of Product Information

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 **Section 9.4.7**).

Depending on the settings of **NVMPROG**, an automatic verify is performed (see **Section 9.4.6**).

In case of a one-shot write, write mode is automatically left. In case of continuous write mode, further write operations to new addresses can follow, until the write mode is explicitly left. Continuous write operations can target blocks within the complete memory without any restriction regarding sector or page borders.

### 9.4.4 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 **Section 9.4.7**).

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.

### 9.4.5 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 **Chapter 9.4.7**).


Subject to Agreement on the Use of Product Information

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.

### 9.4.6 Verify

The data written as described in **Section 9.4.3** 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 **Section 9.4.7**).

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.*

### 9.4.7 Erase-Protection and Write-Protection

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

## 9.5 Redundancy

To increase the production yield, the NVM module contains redundancy. Redundancy is transparent to the user.


Subject to Agreement on the Use of Product Information

## 9.6 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.

## 9.7 Service Request Generation

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

A NVM interrupt can be issued because of 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.

## 9.8 Power, Reset and Clock

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

### 9.8.1 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.

### 9.8.2 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.


Subject to Agreement on the Use of Product Information

#### 9.8.2.1 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.

#### 9.8.2.2 NVM Sleep Mode

Entering and leaving NVM sleep mode requires special state machine sequences, which need 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 activeted, see SCU chapter. 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. No memory access is possible in sleep mode.

### 9.8.3 Reset

The NVM module is reset with the system reset.

### 9.8.4 Clock

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


Subject to Agreement on the Use of Product Information

## 9.9 Registers

Table 9-3 shows a complete list of the NVM special function registers.

**Table 9-2 Registers Address Space**

| Module | Base Address | End Address | Note |
|---|---|---|---|
| NVM | $4005\ 0000_H$ | $4005\ 00FF_H$ | |

**Table 9-3 Registers Overview**

| Register Short Name | Register Long Name | Offset Address | Page Number |
|---|---|---|---|
| NVMSTATUS | NVM Status Register | $0000_H$ | Page 9-12 |
| NVMPROG | NVM Programming Control Register | $0004_H$ | Page 9-14 |
| NVMCONF | NVM Configuration Register | $0008_H$ | Page 9-16 |

The register is addressed wordwise.

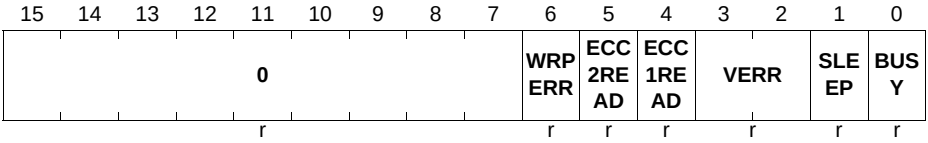
Reading an SFR is not blocked, while the NVM module is busy. If the SFR value depends on the completion of an NVM sequence, NVMSTATUS.BUSY must be polled for $0_B$, before the SFR is read. Otherwise, reading the SFR might yield a value that is not updated yet.


Subject to Agreement on the Use of Product Information

### 9.9.1 NVM Registers

**NVM Status Register**

**NVMSTATUS**
**NVM Status Register** **($0000_H$)** **Reset Value: $0002_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | 15:7 | r | **Reserved**<br>Read as 0. |
| **WRPERR** | 6 | r | **Write Protocol Error**<br>The flag accumulates write protocol violations during the last 4-word write operations (for write or verify). It is also set when a triggered operation was ignored because of write protection. The correct protocol is defined in **Section 9.4.3**. It is reset by hardware when **NVMPROG**.RSTECC is written.<br>$0_B$ **WRPROTOK**, No write protocol failure occurred.<br>$1_B$ **WRPROTFAIL**, At least one write protocol failure was detected. |
| **ECC2READ** | 5 | r | **ECC2 Read**[1)]<br>The flag accumulates ECC two bit failure during memory read operations. It is reset by hardware when **NVMPROG**.RSTECC is written.<br>$0_B$ **ECC2RDOK**, No ECC two bit failure during memory read operations.<br>$1_B$ **ECC2RDFAIL**, At least one ECC two bit failure was detected. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **ECC1READ** | 4 | r | **ECC1 Read**[1)]<br>The flag accumulates ECC single bit failure during the last memory read operations. It is reset by hardware when **NVMPROG**.RSTECC is written.<br>$0_B$ **ECC1RDOK**, No ECC single bit failure occurred.<br>$1_B$ **ECC1RDFAIL**, At least one ECC single bit failure was detected and corrected. |
| **VERR** | 3:2 | r | **Verify Error**<br>The flag is reset by hardware, when **NVMPROG**.RSTVERR is written.<br>The flag is also reset, when write mode or verify-only mode are entered, i.e. **NVMPROG**.ACTION.OPTYPE = $0001_B$ or **NVMPROG**.ACTION.VERIFY = $11_B$.<br>The flag accumulates, i.e. VERR is updated every time a value higher than the current value is required, until write mode or verify-only mode are left (automatically in case of a one-shot write operation).<br>Information on number of fail bits during verify procedure(s):<br>$00_B$ **NOFAIL**, No fail bit.<br>$01_B$ **ONEFAIL**, One fail bit in one data block.<br>$10_B$ **TWOFAIL**, Two fail bits in two different data blocks.<br>$11_B$ **MOREFAIL**, Two or more fail bits in one data block, or three or more fail bits overall. |
| **SLEEP** | 1 | r | **Sleep Mode**<br>$0_B$ **READY**, NVM not in sleep mode, and no sleep or wake up procedure in progress.<br>$1_B$ **SLEEP**, NVM in sleep mode, or busy due to a sleep or wake up procedure. |
| **BUSY** | 0 | r | **Busy**<br>$0_B$ **READY**, The NVM is not busy. Memory reads from the cell array and register write accesses are possible.<br>$1_B$ **BUSY**, The NVM is busy. Memory reads and register write accesses are not possible. |

1) If a block is read from the field that contains two or more parity errors ECC1 and ECC2 errors may be flagged simultaneously.

### NVM Programming Control Register


Subject to Agreement on the Use of Product Information

**NVMPROG**

**NVM Programming Control Register ($0004_H$)** **Reset Value: $0000_H$**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | RSTECC | RSTVERR | 0 | | | | ACTION | | | | | | | |
| r | | rw | rw | r | | | | rw | | | | | | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | 15:14 | r | **Reserved**<br>Read as 0; should by written with 0. |
| **RSTECC** | 13 | rw | **Reset ECC**<br>Can only be set by software, is reset automatically by hardware.<br>$0_B$ **NOP**, No action.<br>$1_B$ **RESET**, Reset of **NVMSTATUS**.ECCxREAD and NVMSTATUS.WRPERR. |
| **RSTVERR** | 12 | rw | **Reset Verify Error**<br>Can only be set by software, is reset automatically by hardware.<br>$0_B$ **NOP**, No action.<br>$1_B$ **RESET**, Reset of **NVMSTATUS**.VERR. |
| **0** | 11:8 | r | **Reserved**<br>Read as 0; should be written with 0. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **ACTION** | 7:0 | rw | **ACTION: [VERIFY, ONE_SHOT, OPTYPE]**<br>This field selects an erase, write, or verify operation. See also **More details on ACTION**. ACTION is a concatenation of three bit fields: ACTION[7:6] = VERIFY, ACTION[5:4] = ONE_SHOT and ACTION[3:0] = OPTYPE.<br>**OPTYPE** defines the following operations:<br>$0000_B$: idle or verify-only, that depends on the setting of VERIFY;<br>$0001_B$: write;<br>$0010_B$: page erase.<br>$0100_B$: sector erase.<br>**ONE_SHOT** is a parameter of OPTYPE with the following values:<br>$01_B$: once or<br>$10_B$: continuously.<br>In case of $01_B$, ACTION is automatically reset to idle mode after operation has been performed.<br>**VERIFY** defines a second parameter of OPTYPE:<br>$01_B$: verification of written data with hardread levels after every write operation,<br>$10_B$: no verification, $11_B$: verification of array content.<br>The following operations are defined, other values are interpreted as $00_H$<br>$00_H$ , Idle state, no action triggered. Writing $00_H$ exits current mode.<br>$51_H$ , Start one-shot write operation with automatic verify.<br>$91_H$ , Start one-shot write operation without verify.<br>$61_H$ , Start continuous write operation with automatic verify of every write.<br>$A1_H$ , Start continuous write operation without verify.<br>$92_H$ , Start one-shot page erase operation.<br>$94_H$ , Start one-shot sector erase operation.<br>$A4_H$ , Start continuous sector erase operation.<br>$A2_H$ , Start continuous page erase operation.<br>$D0_H$ , Start one-shot verify-only: Written data is compared to array content.<br>$E0_H$ , Start continuous verify-only: Written data is compared to array content. |

### More details on ACTION

The operation selected by ACTION is performed, when a write to the NVM address range is performed, which defines the address (and block data) for the operation. Afterwards, in case of a one-shot operation, ACTION is automatically reset.

Verification results can be read at **NVMSTATUS**.VERR.

ACTION can only be changed when the current value of ACTION is $00_H$, otherwise ACTION is set to $00_H$. Once ACTION is not idle, ACTION can only be written again with its current value; any other value leads to a reset of ACTION.

Only write operations can be automatically verified. Page erase operations cannot use automatic verify, they must be started with ACTION.VERIFY = $10_B$.

If the correct erasing of a page or sector is to be verified, a separate sequence using ACTION.VERIFY = $11_B$ has to be started.

A verify operation requires the provision of the data for a complete block and always includes the ECC bits. The ECC bits for every block are generated automatically when the data is written.

A verify operation started with ACTION.VERIFY = $01_B$ is automatically performed with both hardread written and hardread erased levels.
A verify operation started with ACTION.VERIFY = $11_B$ is performed with the single hardread level defined by **NVMCONF**.HRLEV at this starting time.

A sequence started by setting ACTION will set **NVMSTATUS**.BUSY only after the transfer of the address (and block data) is performed. The end of the sequence can be detected either by polling NVMSTATUS.BUSY or by waiting for an NVM interrupt (not for verify-only sequence).

Entering sleep mode resets ACTION.

### NVM Configuration Register

NVMCONF is the only SFR that can be written while the module is in sleep mode. This is necessary to enable the use of NVM_ON = $0_B$ to go to sleep mode and of NVM_ON = $1_B$ to wake-up again.

**NVMCONF**
**NVM Configuration Register** **($0008_H$)** **Reset Value: $9000_H$**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| NVM_ON | INT_ON | 0 | WS | SECPROT | | | | | | | | 0 | HRLEV | | 0 |
| rw | rw | rw | rw | rw | | | | | | | | r | rw | | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **NVM_ON** | 15 | rw | **NVM On**<br>When cleared, no software code can be executed anymore from the NVM, until it is set again. I.e., already the software code that initiates the change in NVM_ON itself may not reside in the NVM, otherwise the software is stalled forever.<br>$0_B$ **SLEEP**, NVM is switched to or stays in sleep mode.<br>$1_B$ **NORM**, NVM is switched to or stays in normal mode. |
| **INT_ON** | 14 | rw | **Interrupt On**<br>When enabled the completion of a sequence started by setting **NVMPROG**.ACTION (write or erase sequence) will be indicated by NVM interrupt. The same is true for the wake-up sequence.<br>$0_B$ **INTOFF**, No NVM ready interrupts are generated.<br>$1_B$ **INTON**, NVM ready interrupts are generated. |
| **0** | 13 | rw | **Reserved for Future Use**<br>Must be written with 0 to allow correct operation. |
| **WS** | 12 | rw | **Number of fixed Wait States**<br>Defines the number of fixed wait states when NVM_CONFIG1.FIXWS = $1_B$.<br>$0_B$ 0 fixed wait states.<br>$1_B$ 1 fixed wait state. |
| **SECPROT** | 11:4 | rw | **Sector Protection**[1)]<br>This field defines the number of write, erase, verify protected sectors, starting with physical sector 0. |
| **0** | 3 | r | **Reserved**<br>Read as 0; should be written with 0. |
| **HRLEV** | 2:1 | rw | **Hardread Level**[2)]<br>Defines single hardread level for verification with **NVMPROG**.ACTION.VERIFY = $11_B$:<br>$00_B$ **NR**, Normal read<br>$01_B$ **HRW**, Hardread written<br>$10_B$ **HRE**, Hardread erased<br>$11_B$ **RFU**, Reserved for Future Use |
| **0** | 0 | rw | **Reserved for Future Use**<br>Must be written with 0 to allow correct operation. |


Subject to Agreement on the Use of Product Information

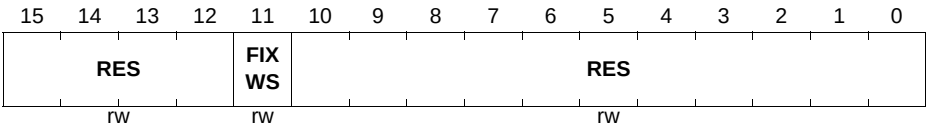
1) For SECPROT > 0, SECPROT defines the number of protected sectors. The sectors 0 to SECPROT-1 cannot be written, erased, or verified. All writes that target the protected sectors are accepted, but are internally ignored.
2) HRLEV defines the hardread level for a stand-alone verification sequence started with NVMPROG.ACTION.VERIFY = $11_B$. This hardread level is used until the end of the verification sequence. HRLEV may not be changed in between.

### Configuration 1 Register

The bit NVM_CONFIG1.FIXWS allows to switch between adaptive and fixed wait state configuration.

***Attention: Any write operation to the register CONFIG1 to switch between adaptive and fixed wait states configuration must only modify the bit CONFIG1.FIXWS. Changing other bits in CONFIG1 can lead to unpredictable results.***

**CONFIG1**
**Configuration 1 Register** **(4005 0048$_H$)** **Reset Value: XXXX$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **RES** | 15:12 | rw | **Reserved**<br>Must not be changed when programming the NVM_CONFIG1 register. |
| **FIXWS** | 11 | rw | **Wait States Scheme**<br>Defines the scheme by which flash wait states are generated. Withfixed wait states NVM_NVMCONF.WS defines the number of wait states.<br>$0_B$ adaptive wait states.<br>$1_B$ fixed wait states. |
| **RES** | 10:0 | rw | **Reserved**<br>Must not be changed when programming the NVM_CONFIG1 register. |


Subject to Agreement on the Use of Product Information

## 9.10 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.

### 9.10.1 Writing to Memory

#### 9.10.1.1 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.

#### 9.10.1.2 Writing Blocks

This sequence requires 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 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.
4. Jump to step 2 unless all data is written.


Subject to Agreement on the Use of Product Information

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.

### 9.10.2 Erasing Memory

#### 9.10.2.1 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 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.

#### 9.10.2.2 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 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.
4. Jump to step 2 unless all pages are erased.
5. Stop continuous page erase operation:
   Write **NVMPROG**.ACTION = $00_H$.

#### 9.10.2.3 Erasing a Single Sector

Assumption: **NVMPROG**.ACTION = $00_H$.

1. Start a one-shot sector erase operation:
   Write **NVMPROG**.ACTION = $94_H$.


Subject to Agreement on the Use of Product Information

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 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.

#### 9.10.2.4 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 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.
4. Jump to step 2 unless all sectors are erased.
5. Stop continuous sector erase operation:
   Write **NVMPROG**.ACTION = $00_H$.

### 9.10.3 Verifying Memory

#### 9.10.3.1 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.

#### 9.10.3.2 Verifying Blocks

Assumption: **NVMPROG**.ACTION = $00_H$.

1. Choose desired hardread level by setting NVMCONF.HRLEV.


Subject to Agreement on the Use of Product Information

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 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 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.

### 9.10.4 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, e.g. 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 9-4**, where a block is updated 32 times without corruption of the ECC, i.e. the data stays ECC protected throughout the procedure.

**Table 9-4 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 |

**Table 9-4 Incremental Update of a Block with Specially Constructed Data**

| Operation | Block Write Data | Resulting Block Content[1)] |
|---|---|---|
| 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 |
| 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 9-4** 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 9-4**. 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.


Subject to Agreement on the Use of Product Information

### 9.10.5 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.


Subject to Agreement on the Use of Product Information

## 9.10.6 Timing

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

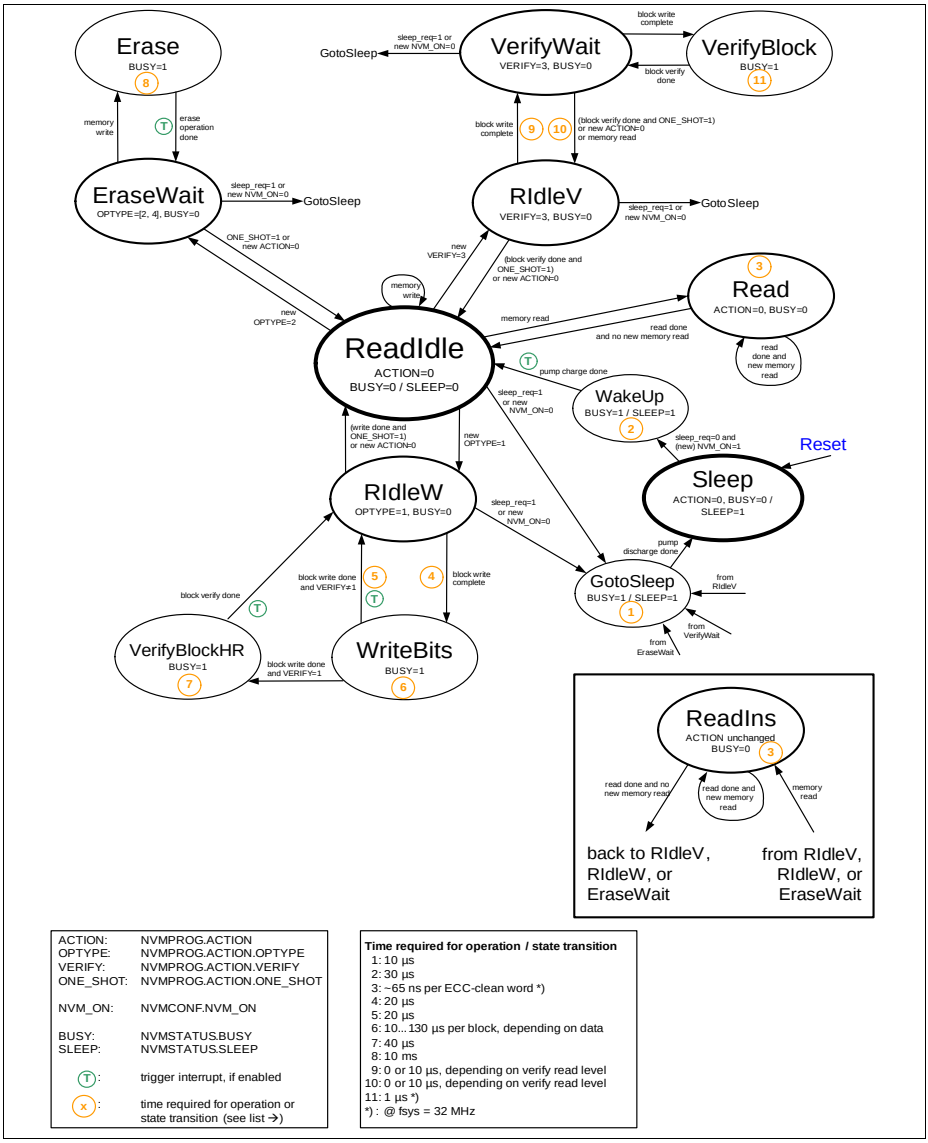



**Figure 9-4 State Diagram of the NVM Module Timings are preliminary**

# 10 Peripheral Access Unit (PAU)

This chapter describes Peripheral Access Unit (PAU) in XMC1300.

## 10.1 Overview

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

### 10.1.1 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

## 10.2 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 10-1** shows the peripherals and their assigned PDISx bits. Peripherals without a PDISx bit are accessible at all times.

**Table 10-1 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 |


Subject to Agreement on the Use of Product Information

**Table 10-1 Peripherals Availability and Privilege Access Control**

| Peripheral | Address Grouping | AVAILn.AVAILx bit | PRIVDIS.PDISx bit |
|---|---|---|---|
| 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 |
| | VADC0 Group 1 SFRs | AVAIL1.7 | PRIVDIS1.7 |
| SHS0 | SHS0 | AVAIL1.8 | PRIVDIS1.8 |
| CCU4 | CC40 and CCU40 Kernel SFRs | AVAIL1.9 | PRIVDIS1.9 |
| | CC41 | AVAIL1.10 | PRIVDIS1.10 |
| | CC42 | AVAIL1.11 | PRIVDIS1.11 |
| | CC43 | AVAIL1.12 | PRIVDIS1.12 |
| CCU80 | CC80 and CCU80 Kernel SFRs | AVAIL2.0 | PRIVDIS2.0 |
| | CC81 | AVAIL2.1 | PRIVDIS2.1 |
| | CC82 | AVAIL2.2 | PRIVDIS2.2 |
| | CC83 | AVAIL2.3 | PRIVDIS2.3 |
| POSIF0 | POSIF0 | AVAIL2.12 | PRIVDIS2.12 |
| BCCU0 | BCCU0 | AVAIL2.15 | PRIVDIS2.15 |

## 10.3 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 10-1** for the bit assignement. Similarly, the Memory Size Registers (e.g. FLSIZE for Flash memory) can be used to check for the size of the available memories.

## 10.4 Service Request Generation

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

## 10.5 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.


Subject to Agreement on the Use of Product Information

## 10.6 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

## 10.7 Initialization and System Dependencies

The PAU is always available after reset.

## 10.8 PAU Registers

### Registers Overview

The absolute register address is calculated by adding:

Module Base Address + Offset Address

**Table 10-2 Registers Address Space**

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

**Table 10-3 Register Overview**

| Short Name | Description | Offset Addr. | Access Mode | | Description See |
|---|---|---|---|---|---|
| | | | Read | Write | |
| Reserved | Reserved | 0000$_H$ - 003C$_H$ | nBE | nBE | |
| AVAIL0 | Peripheral Availability Register 0 | 0040$_H$ | U, PV | BE | Page 10-8 |
| AVAIL1 | Peripheral Availability Register 1 | 0044$_H$ | U, PV | BE | Page 10-10 |
| AVAIL2 | Peripheral Availability Register 2 | 0048$_H$ | U, PV | BE | Page 10-11 |
| Reserved | Reserved | 004C$_H$ - 007C$_H$ | nBE | nBE | |
| PRIVDIS0 | Peripheral Privilege Access Register 0 | 0080$_H$ | U, PV | U, PV | Page 10-4 |
| PRIVDIS1 | Peripheral Privilege Access Register 1 | 0084$_H$ | U, PV | U, PV | Page 10-6 |


Subject to Agreement on the Use of Product Information

**Table 10-3 Register Overview** (cont'd)

| Short Name | Description | Offset Addr. | Access Mode | | Description See |
|---|---|---|---|---|---|
| | | | Read | Write | |
| PRIVDIS2 | Peripheral Privilege Access Register 2 | $0088_H$ | U, PV | U, PV | Page 10-7 |
| Reserved | Reserved | $008C_H$ - $03FC_H$ | nBE | nBE | |
| ROMSIZE | ROM Size Register | $0400_H$ | U, PV | BE | Page 10-12 |
| FLSIZE | Flash Size Register | $0404_H$ | U, PV | BE | Page 10-13 |
| RAM0SIZE | RAM0 Size Register | $0410_H$ | U, PV | BE | Page 10-13 |

## 10.8.1 Peripheral Privilege Access Registers (PRIVDISn)

The PRIVDISn registers provide the bit to enable and disable access to a peripheral during runtime. When disabled, an access to the peripheral throws a BusFault.

**PRIVDIS0**
**Peripheral Privilege Access Register 0 ($0080_H$)** **Reset Value: $0000\ 0000_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | PDIS24 | PDIS23 | PDIS22 | PDIS21 | PDIS20 | PDIS19 | 0 | | |
| r | | | | | | | rw | rw | rw | rw | rw | rw | r | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | PDIS7 | PDIS6 | PDIS5 | 0 | | PDIS2 | 0 | |
| r | | | | | | | | rw | rw | rw | r | | rw | r | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **PDIS2** | 2 | rw | **Flash SFRs Privilege Disable Flag**<br>$0_B$ Flash SFRs are accessible.<br>$1_B$ Flash SFRs are not accessible. |
| **PDIS5** | 5 | rw | **RAM Block 1 Privilege Disable Flag**<br>$0_B$ RAM Block 1 is accessible.<br>$1_B$ RAM Block 1 is not accessible. |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **PDIS6** | 6 | rw | **RAM Block 2 Privilege Disable Flag**<br>$0_B$ RAM Block 2 is accessible.<br>$1_B$ RAM Block 2 is not accessible. |
| **PDIS7** | 7 | rw | **RAM Block 3 Privilege Disable Flag**<br>$0_B$ RAM Block 3 is accessible.<br>$1_B$ RAM Block 3 is not accessible. |
| **PDIS19** | 19 | rw | **WDT Privilege Disable Flag**<br>$0_B$ WDT is accessible.<br>$1_B$ WDT is not accessible. |
| **PDIS20** | 20 | rw | **MATH Global SFRs and Divider Privilege Disable Flag**<br>$0_B$ MATH Global SFRs and Divider are accessible.<br>$1_B$ MATH Global SFRs and Divider are not accessible. |
| **PDIS21** | 21 | rw | **MATH CORDIC Privilege Disable Flag**<br>$0_B$ MATH CORDIC is accessible.<br>$1_B$ MATH CORDIC is not accessible. |
| **PDIS22** | 22 | rw | **Port 0 Privilege Disable Flag**<br>$0_B$ Port 0 is accessible.<br>$1_B$ Port 0 is not accessible. |
| **PDIS23** | 23 | rw | **Port 1 Privilege Disable Flag**<br>$0_B$ Port 1 is accessible.<br>$1_B$ Port 1 is not accessible. |
| **PDIS24** | 24 | rw | **Port 2 Privilege Disable Flag**<br>$0_B$ Port 2 is accessible.<br>$1_B$ Port 2 is not accessible. |
| **0** | [31:25], [18:8], [4:3], [1:0] | r | **Reserved** |


Subject to Agreement on the Use of Product Information

**PRIVDIS1**

**Peripheral Privilege Access Register 1 ($0084_H$)** **Reset Value: $0000\ 0000_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | PDIS12 | PDIS11 | PDIS10 | PDIS9 | PDIS8 | PDIS7 | PDIS6 | PDIS5 | 0 | | | PDIS1 | PDIS0 |
| r | | | rw | rw | rw | rw | rw | rw | rw | rw | r | | | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **PDIS0** | 0 | rw | **USIC0 Channel 0 Privilege Disable Flag**<br>$0_B$ USIC0 Channel 0 is accessible.<br>$1_B$ USIC0 Channel 0 is not accessible. |
| **PDIS1** | 1 | rw | **USIC0 Channel 1 Privilege Disable Flag**<br>$0_B$ USIC0 Channel 1 is accessible.<br>$1_B$ USIC0 Channel 1 is not accessible. |
| **PDIS5** | 5 | rw | **VADC0 Basic SFRs Privilege Disable Flag**<br>$0_B$ VADC0 Basic SFRs are accessible.<br>$1_B$ VADC0 Basic SFRs are not accessible. |
| **PDIS6** | 6 | rw | **VADC0 Group 0 SFRs Privilege Disable Flag**<br>$0_B$ VADC0 Group 0 SFRs are accessible.<br>$1_B$ VADC0 Group 0 SFRs are not accessible. |
| **PDIS7** | 7 | rw | **VADC0 Group 1 SFRs Privilege Disable Flag**<br>$0_B$ VADC0 Group 1 SFRs are accessible.<br>$1_B$ VADC0 Group 1 SFRs are not accessible. |
| **PDIS8** | 8 | rw | **SHS0 Privilege Disable Flag**<br>$0_B$ SHS0 is accessible.<br>$1_B$ SHS0 is not accessible. |
| **PDIS9** | 9 | rw | **CC40 and CCU40 Kernel SFRs Privilege Disable Flag**<br>$0_B$ CC40 and CCU40 Kernel SFRs are accessible.<br>$1_B$ CC40 and CCU40 Kernel SFRs are not accessible. |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **PDIS10** | 10 | rw | **CC41 Privilege Disable Flag**<br>$0_B$ CC41 is accessible.<br>$1_B$ CC41 is not accessible. |
| **PDIS11** | 11 | rw | **CC42 Privilege Disable Flag**<br>$0_B$ CC42 is accessible.<br>$1_B$ CC42 is not accessible. |
| **PDIS12** | 12 | rw | **CC43 Privilege Disable Flag**<br>$0_B$ CC43 is accessible.<br>$1_B$ CC43 is not accessible. |
| **0** | [31:13], [4:2] | r | **Reserved** |

**PRIVDIS2**
**Peripheral Privilege Access Register 2 ($0088_H$)** **Reset Value: $0000\ 0000_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **PDIS 15** | **0** | | **PDIS 12** | **0** | | | | | | | | **PDIS 3** | **PDIS 2** | **PDIS 1** | **PDIS 0** |
| rw | r | | rw | r | | | | | | | | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **PDIS0** | 0 | rw | **CC80 and CCU80 Kernel SFRs Privilege Disable Flag**<br>$0_B$ CC80 and CCU80 Kernel SFRs are accessible.<br>$1_B$ CC80 and CCU80 Kernel SFRs are not accessible. |
| **PDIS1** | 1 | rw | **CC81 Privilege Disable Flag**<br>$0_B$ CC81 is accessible.<br>$1_B$ CC81 is not accessible. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **PDIS2** | 2 | rw | **CC82 Privilege Disable Flag**<br>$0_B$ CC82 is accessible.<br>$1_B$ CC82 is not accessible. |
| **PDIS3** | 3 | rw | **CC83 Privilege Disable Flag**<br>$0_B$ CC83 is accessible.<br>$1_B$ CC83 is not accessible. |
| **PDIS12** | 12 | rw | **POSIF0 Privilege Disable Flag**<br>$0_B$ POSIF0 is accessible.<br>$1_B$ POSIF0 is not accessible. |
| **PDIS15** | 15 | rw | **BCCU0 Privilege Disable Flag**<br>$0_B$ BCCU0 is accessible.<br>$1_B$ BCCU0 is not accessible. |
| **0** | [31:16], [14:13], [11:4] | r | **Reserved** |

## 10.8.2 Peripheral Availability Registers (AVAILn)

The AVAILn registers indicate the available peripherals for the particular device variant.

*Note: The reset values of AVAILn registers show the configuration with all peripherals available. Actual values might differ depending on the product variant.*

**AVAIL0**
**Peripheral Availability Register 0 ($0040_H$)** **Reset Value: 01FF 00FF$_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | AVAIL24 | AVAIL23 | AVAIL22 | AVAIL21 | AVAIL20 | 1 | | | |
| r | | | | | | | r | r | r | r | r | r | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | AVAIL7 | AVAIL6 | AVAIL5 | 1 | | | | |
| r | | | | | | | | r | r | r | r | | | | |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **AVAIL5** | 5 | r | **RAM Block 1 Availability Flag**<br>$0_B$ RAM block 1 is not available.<br>$1_B$ RAM block 1 is available. |
| **AVAIL6** | 6 | r | **RAM Block 2 Availability Flag**<br>$0_B$ RAM block 2 is not available.<br>$1_B$ RAM block 2 is available. |
| **AVAIL7** | 7 | r | **RAM Block 3 Availability Flag**<br>$0_B$ RAM block 3 is not available.<br>$1_B$ RAM block 3 is available. |
| **AVAIL20** | 20 | r | **MATH Global SFRs and Divider Availability Flag**<br>$0_B$ MATH Global SFRs and Divider are not available.<br>$1_B$ MATH Global SFRs and Divider are available. |
| **AVAIL21** | 21 | r | **MATH CORDIC Availability Flag**<br>$0_B$ MATH CORDIC is not available.<br>$1_B$ MATH CORDIC is available. |
| **AVAIL22** | 22 | r | **Port 0 Availability Flag**<br>$0_B$ Port 0 is not available.<br>$1_B$ Port 0 is available. |
| **AVAIL23** | 23 | r | **Port 1 Availability Flag**<br>$0_B$ Port 1 is not available.<br>$1_B$ Port 1 is available. |
| **AVAIL24** | 24 | r | **Port 2 Availability Flag**<br>$0_B$ Port 2 is not available.<br>$1_B$ Port 2 is available. |
| **1** | [19:16], [4:0] | r | **Reserved** |
| **0** | [31:25], [15:8] | r | **Reserved** |


Subject to Agreement on the Use of Product Information

**AVAIL1**

**Peripheral Availability Register 1 ($0044_H$) Reset Value: 0000 1FF7$_H$**

| 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 | | | AVAIL12 | AVAIL11 | AVAIL10 | AVAIL9 | AVAIL8 | AVAIL7 | AVAIL6 | AVAIL5 | AVAIL4 | 0 | 1 | AVAIL1 | AVAIL0 |
| r | | | r | r | r | r | r | r | r | r | r | r | r | r | r |

| Field | Bits | Type | Description |
|---|---|---|---|
| **AVAIL0** | 0 | r | **USIC0 Channel 0 Availability Flag**<br>$0_B$ USIC0 Channel 0 is not available.<br>$1_B$ USIC0 Channel 0 is available. |
| **AVAIL1** | 1 | r | **USIC0 Channel 1 Availability Flag**<br>$0_B$ USIC0 Channel 1 is not available.<br>$1_B$ USIC0 Channel 1 is available. |
| **AVAIL4** | 4 | r | **PRNG Availability Flag**<br>$0_B$ PRNG is not available.<br>$1_B$ PRNG is available. |
| **AVAIL5** | 5 | r | **VADC0 Basic SFRs Availability Flag**<br>$0_B$ VADC0 Basic SFRs are not available.<br>$1_B$ VADC0 Basic SFRs are available. |
| **AVAIL6** | 6 | r | **VADC0 Group 0 SFRs Availability Flag**<br>$0_B$ VADC0 Group 0 SFRs are not available.<br>$1_B$ VADC0 Group 0 SFRs are available. |
| **AVAIL7** | 7 | r | **VADC0 Group 1 SFRs Availability Flag**<br>$0_B$ VADC0 Group 1 SFRs are not available.<br>$1_B$ VADC0 Group 1 SFRs are available. |
| **AVAIL8** | 8 | r | **SHS0 Availability Flag**<br>$0_B$ SHS0 is not available.<br>$1_B$ SHS0 is available. |
| **AVAIL9** | 9 | r | **CC40 Availability Flag**<br>$0_B$ CC40 is not available.<br>$1_B$ CC40 is available. |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **AVAIL10** | 10 | r | **CC41 Availability Flag**<br>$0_B$ CC41 is not available.<br>$1_B$ CC41 is available. |
| **AVAIL11** | 11 | r | **CC42 Availability Flag**<br>$0_B$ CC42 is not available.<br>$1_B$ CC42 is available. |
| **AVAIL12** | 12 | r | **CC43 Availability Flag**<br>$0_B$ CC43 is not available.<br>$1_B$ CC43 is available. |
| **1** | 2 | r | **Reserved** |
| **0** | [31:13], 3 | r | **Reserved** |

**AVAIL2**
**Peripheral Availability Register 2 ($0048_H$)** **Reset Value: 0000 900F$_H$**

| 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AVAIL15 | 0 | | AVAIL12 | 0 | | | | | | | | AVAIL3 | AVAIL2 | AVAIL1 | AVAIL0 |
| r | r | | r | r | | | | | | | | r | r | r | r |

| Field | Bits | Type | Description |
|---|---|---|---|
| **AVAIL0** | 0 | r | **CC80 and CCU80 kernel SFRs Availability Flag**<br>$0_B$ CC80 and CCU80 kernel SFRs are not available.<br>$1_B$ CC80 and CCU80 kernel SFRs are available. |
| **AVAIL1** | 1 | r | **CC81 Availability Flag**<br>$0_B$ CC81 is not available.<br>$1_B$ CC81 is available. |
| **AVAIL2** | 2 | r | **CC82 Availability Flag**<br>$0_B$ CC82 is not available.<br>$1_B$ CC82 is available. |


Subject to Agreement on the Use of Product Information

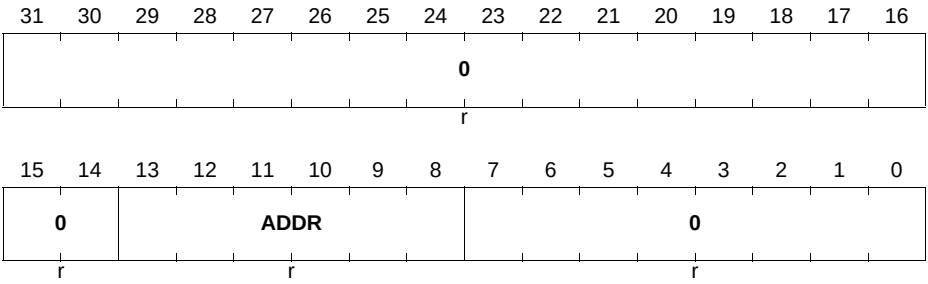
| Field | Bits | Type | Description |
|---|---|---|---|
| **AVAIL3** | 3 | r | **CC83 Availability Flag**<br>$0_B$ CC83 is not available.<br>$1_B$ CC83 is available. |
| **AVAIL12** | 12 | r | **POSIF0 Availability Flag**<br>$0_B$ POSIF0 is not available.<br>$1_B$ POSIF0 is available. |
| **AVAIL15** | 15 | r | **BCCU0 Availability Flag**<br>$0_B$ BCCU0 is not available.<br>$1_B$ BCCU0 is available. |
| **0** | [31:16], [14:13], [11:4] | r | **Reserved** |

### 10.8.3 Memory Size Registers

Memory size registers are available for the ROM, SRAM and Flash memories. They are used to indicate the available size of these memories in the device.

*Note: The reset values of the size registers show the configuration of the superset device. Actual values might differ depending on the product variant.*

**ROMSIZE**
**ROM Size Register** **($0400_H$)** **Reset Value: $0000\ 0B00_H$**

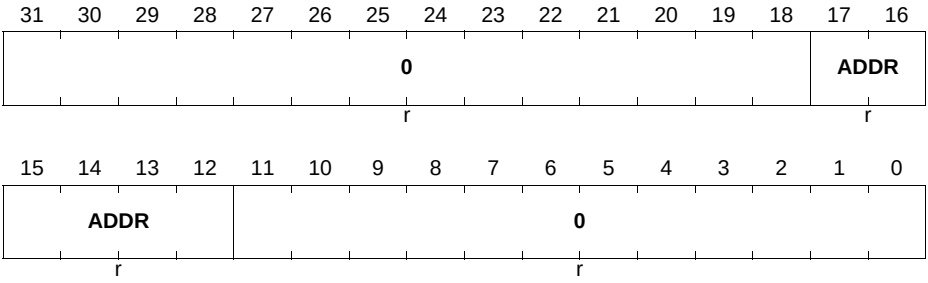



| Field | Bits | Type | Description |
|---|---|---|---|
| **ADDR** | [13:8] | r | **ROM Size**<br>Size of user-readable ROM in bytes = ADDR * 256 |


Subject to Agreement on the Use of Product Information

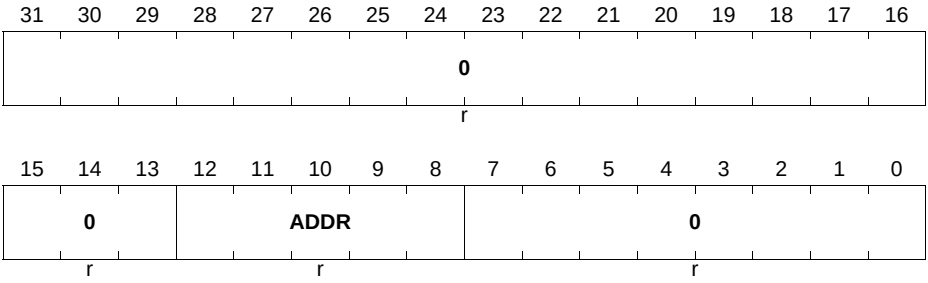
| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | [31:14], [7:0] | r | **Reserved** |

**FLSIZE**
**Flash Size Register** **(0404$_H$)** **Reset Value: 0003 3000$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **ADDR** | [17:12] | r | **Flash Size**<br>Size of the Flash (excluding Flash sector 0) in Kbytes = (ADDR - 1) * 4 |
| **0** | [31:18], [11:0] | r | **Reserved** |

**RAM0SIZE**
**RAM0 Size Register** **(0410$_H$)** **Reset Value: 0000 1000$_H$**





Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **ADDR** | [12:8] | r | **RAM0 Size**<br>Size of RAM block 0 in bytes = ADDR * 256<br>For total RAM size, RAM blocks 1 to 3 have to be also taken into consideration. |
| **0** | [31:13], [7:0] | r | **Reserved** |

# System Control


Subject to Agreement on the Use of Product Information

# 11 Window Watchdog Timer (WDT)

Purpose of the Window Watchdog Timer module is improvement of system integrity. WDT triggers the system reset or other corrective action like e.g. 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)

## 11.1 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 can be used to show to an external watchdog that the system is alive.

The WDT timer is a 32-bit counter, which counts up from $0_H$. It can be serviced while the counter value is within the window boundary, i.e. between the lower and the upper boundary value. Correct servicing results in a reset of the counter to $0_H$. A so called “Bad Service” attempt results in the system reset request.

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

### 11.1.1 Features

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

The features are:

- Triggers system reset when not serviced on time or serviced in a wrong way
- Servicing restricted to be within boundaries of 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


Subject to Agreement on the Use of Product Information

**Table 11-1 Application Features**

| Feature | Purpose/Application |
|---|---|
| System reset upon Bad Servicing | Triggered to restore system stable operation and ensure system integrity |
| Servicing restricted to be within defined boundaries of refresh window | Allows to consider minimum and maximum software timing |
| Independent clocks | To ensure that WDT counts even in case of the system clock failure |
| Service indication on external pin | For dual-channel watchdog solution, additional external control of system integrity |
| Suspending in HALT mode | Enables safe debugging with productive code |
| Pre-warning alarm | Software recovery to allow corrective action via software recovery routine bringing system back from the unresponsive state into normal operation |

### 11.1.2 Block Diagram

The WDT block diagram is shown in **Figure 11-1**.

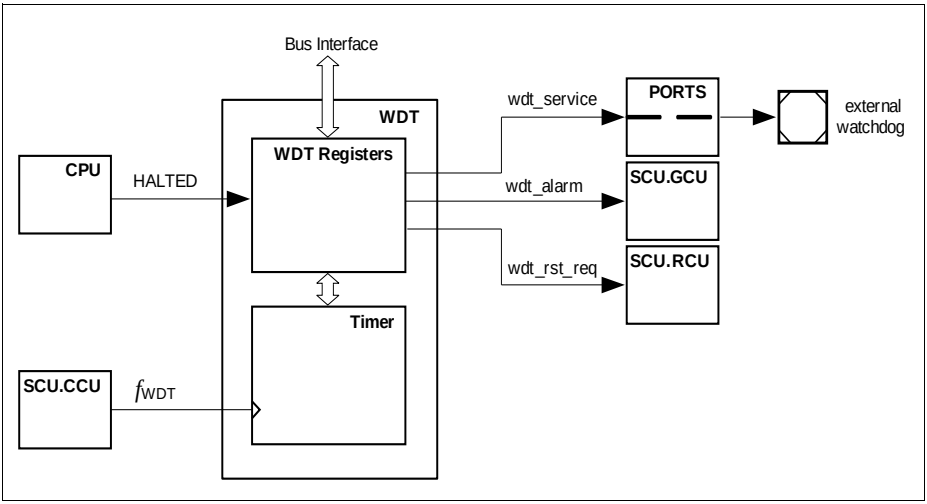



**Figure 11-1 Watchdog Timer Block Diagram**


Subject to Agreement on the Use of Product Information

## 11.2 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.

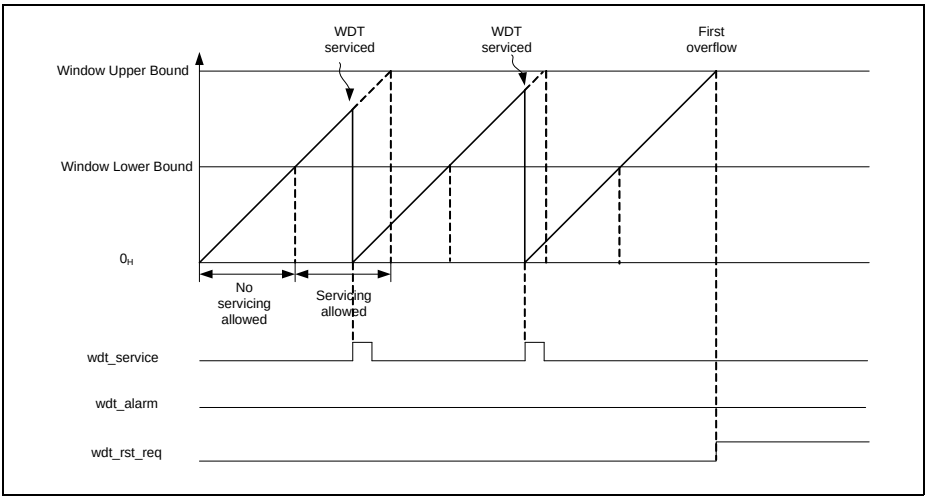



**Figure 11-2 Reset without pre-warning**

The example scenario depicted in **Figure 11-2** shows two consecutive service pulses generated from WDT module as the result of successful servicing within valid time windows. The situation where no service has been performed immediately triggers generation of reset request on the wdt_rst_req output after the counter value has exceeded window upper bound value.

## 11.3 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.


Subject to Agreement on the Use of Product Information

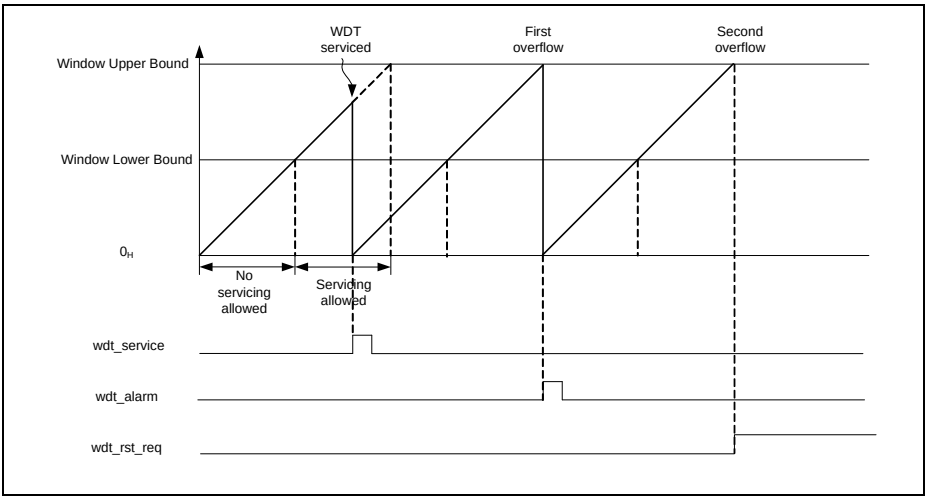



**Figure 11-3 Reset after pre-warning**

The example scenario depicted in **Figure 11-3** shows service pulse generated from WDT module as the result of successful servicing within valid time window. WDT generates alarm pulse on wdt_alarm upon first missing servicing. The alarm signal is routed as interrupt request to the SCU. 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.

## 11.4 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.

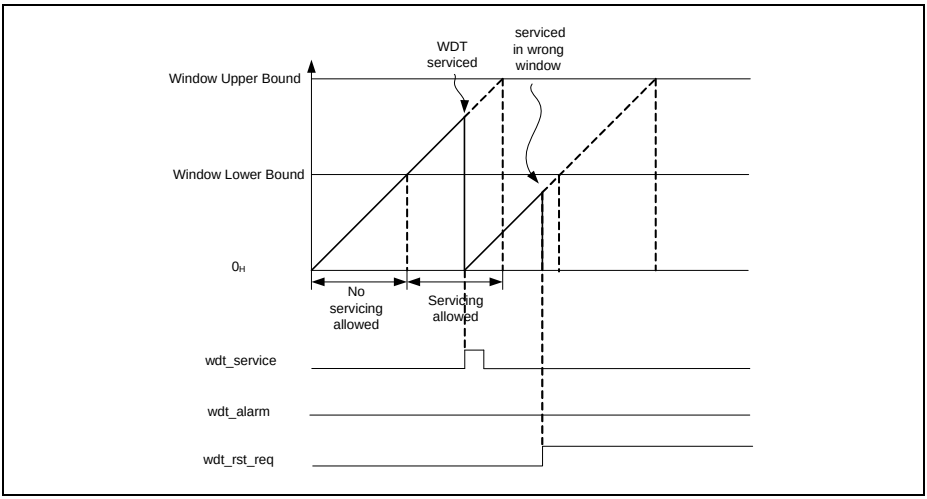



**Figure 11-4 Reset upon servicing in a wrong window**

The example in **Figure 11-4** shows servicing performed outside of valid servicing window. Attempt to service WDT while counter value remains below the window lower bound results in immediate reset request on wdt_rst_req signal.

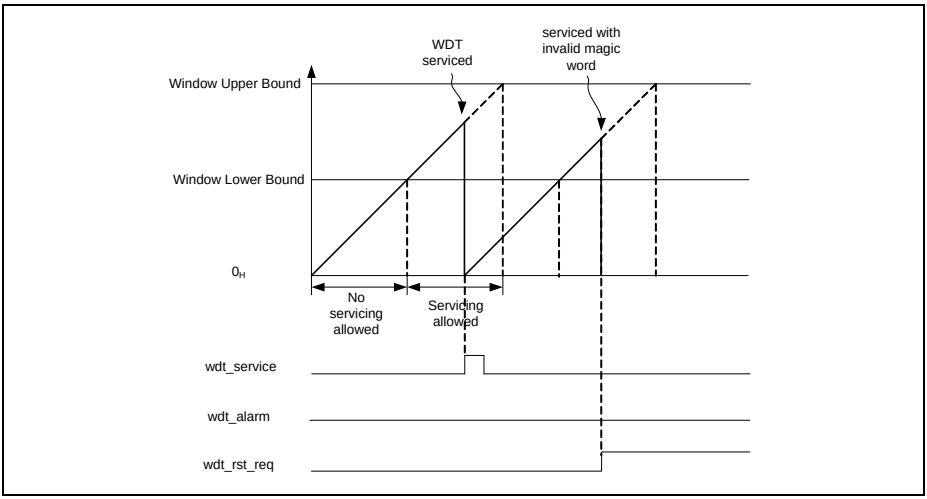



**Figure 11-5 Reset upon servicing with a wrong magic word**

The example in **Figure 11-5** shows servicing performed within a valid servicing window but with an invalid Magic Word. Attempt to write a wrong word to the **SRV** register results in immediate reset request on wdt_rst_req signal.

## 11.5 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.

## 11.6 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.

## 11.7 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 32kHz standby clock from SCU/CCU module, independently from the AHB interface clock.

The WDT module clock is default disabled and can be enabled via the SCU_CGATCLR0 register. Enabling and disabling the module clock could cause load change and clock blanking could happen as explained in the CCU (Clock Gating Control) section of the SCU chapter. It is strongly recommended to setup the module clock in the user initialisation code to avoid clock blanking during runtime.

## 11.8 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 please refer to relevant chapters of this document.

### 11.8.1 Initialization & Start of Operation

Complete WDT module initialization is required upon system reset.


Subject to Agreement on the Use of Product Information

- 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, please refer to PORT chapter)
  - 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

### 11.8.2 Software Stop & Resume Operation

The WDT module can be stopped and re-started at any point of time for e.g. debug purpose using software sequence.

- software stop sequence
  - disable WDT module with WDT_CTR register
- perform any user operations
- software start (resume) sequence
  - enable WDT module with WDT_CTR register with WDT_CTR register
- service the watchdog
  - write magic word to WDT_SRV register within valid time window

### 11.8.3 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


Subject to Agreement on the Use of Product Information

  - 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

### 11.8.4 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


Subject to Agreement on the Use of Product Information

## 11.9 WDT Registers

### Registers Overview

The absolute register address is calculated by adding:

Module Base Address + Offset Address

**Table 11-2 Registers Address Space**

| Module | Base Address | End Address | Note |
|---|---|---|---|
| WDT | 4002 0000$_H$ | 4002 FFFF$_H$ | Watchdog Timer Registers |

**Table 11-3 Register Overview**

| Short Name | Description | Offset Addr. | Access Mode | | Description See |
|---|---|---|---|---|---|
| | | | Read | Write | |
| WDT Kernel Registers | | | | | |
| ID | Module ID Register | 00$_H$ | U, PV | PV | Page 11-9 |
| CTR | Control Register | 04$_H$ | U, PV | PV | Page 11-10 |
| SRV | Service Register | 08$_H$ | BE | PV | Page 11-11 |
| TIM | Timer Register | 0C$_H$ | U, PV | BE | Page 11-13 |
| WLB | Window Lower Bound | 10$_H$ | U, PV | PV | Page 11-13 |
| WUB | Window Upper Bound | 14$_H$ | U, PV | PV | Page 11-14 |
| WDTSTS | Watchdog Status Register | 18$_H$ | U, PV | PV | Page 11-14 |
| WDTCLR | Watchdog Status Clear Register | 1C$_H$ | U, PV | PV | Page 11-15 |

### 11.9.1 Registers Description

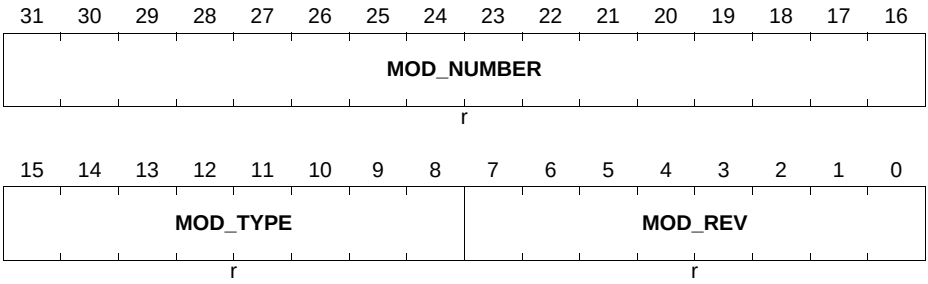
#### ID

The module unique ID register.

### ID

### WDT Module ID Register

**($0000_H$)** **Reset Value: 00AD C0XX$_H$**



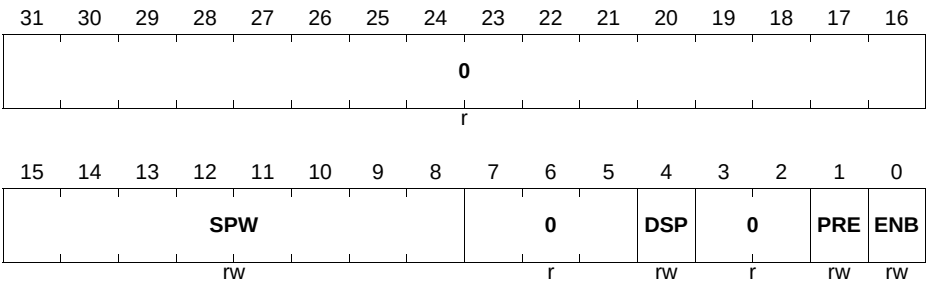

| Field | Bits | Type | Description |
|---|---|---|---|
| **MOD_REV** | [7:0] | r | **Module Revision Number**<br>Indicates the revision number of the implementation. The value of a module revision starts with $01_H$ (first revision). |
| **MOD_TYPE** | [15:8] | r | **Module Type**<br>This bit field is fixed to $C0_H$. |
| **MOD_NUMBER** | [31:16] | r | **Module Number Value**<br>This bit field defines the module identification number. |

### CTR

The operation mode control register.


Subject to Agreement on the Use of Product Information

**CTR**
**WDT Control Register** **($04_H$)** **Reset Value: 0000 0000$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **ENB** | 0 | rw | **Enable**<br>$0_B$ disables watchdog timer,<br>$1_B$ enables watchdog timer |
| **PRE** | 1 | rw | **Pre-warning**<br>$0_B$ disables pre-warning<br>$1_B$ enables pre-warning, |
| **DSP** | 4 | rw | **Debug Suspend**<br>$0_B$ watchdog timer is stopped during debug halting mode<br>$1_B$ watchdog timer is not stopped during debug halting mode |
| **SPW** | [15:8] | rw | **Service Indication Pulse Width**<br>Pulse width (SPW+1) of service indication in $f_{WDT}$ cycles |
| **0** | [3:2], [7:5], [31:16] | r | **reserved** |

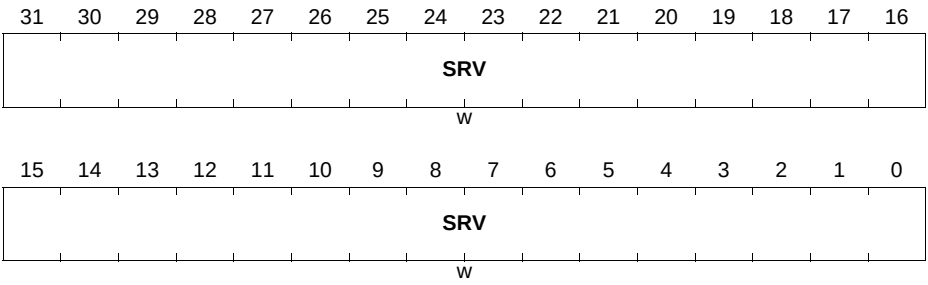
**SRV**

The WDT service register. Software must write a magic word while the timer value is within the valid window boundary. Writing the magic word while the timer value is within the window boundary will service the watchdog and result a reload of the timer with 0H.


Subject to Agreement on the Use of Product Information

Upon writing data different than the magic word within valid time window or writing even correct Magic Word but outside of the valid time window no servicing will be performed. Instead will request an immediate system reset request.

**SRV**
**WDT Service Register** **($08_H$)** **Reset Value: $0000\ 0000_H$**

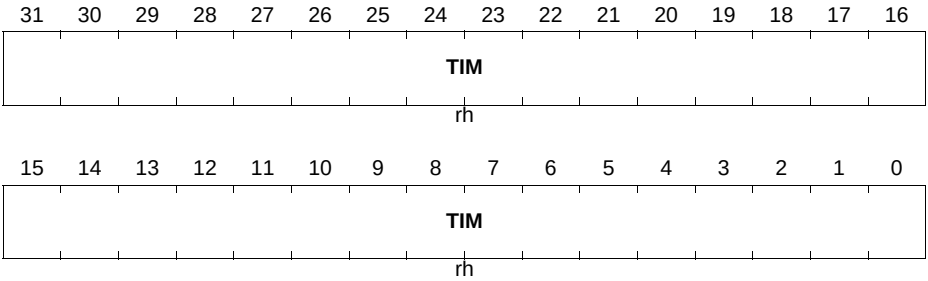



| Field | Bits | Type | Description |
|---|---|---|---|
| **SRV** | [31:0] | w | **Service**<br>Writing the magic word $ABADCAFE_H$ while the timer value is within the window boundary will service the watchdog. |

**TIM**

The actual watchdog timer register count value. This register can be read by software in order to determine current position in the WDT time window.

**TIM**
**WDT Timer Register (0C$_H$) Reset Value: 0000 0000$_H$**

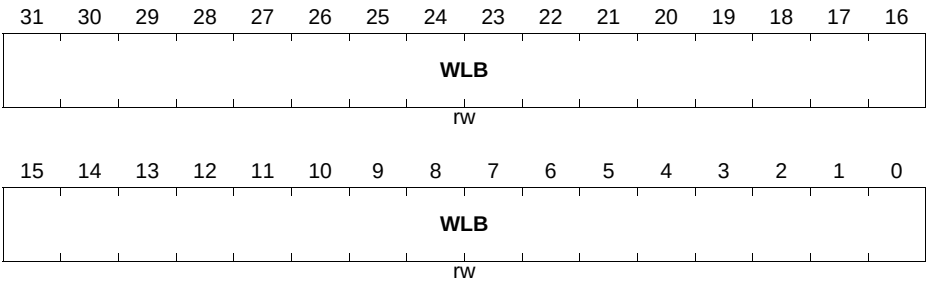



| Field | Bits | Type | Description |
|---|---|---|---|
| **TIM** | [31:0] | rh | **Timer Value**<br>Actual value of watchdog timer value. |

**WLB**

The Window Lower Bound register defines the lower bound for servicing window. Servicing of the watchdog has only effect within the window boundary

**WLB**
**WDT Window Lower Bound Register (10$_H$) Reset Value: 0000 0000$_H$**





Subject to Agreement on the Use of Product Information

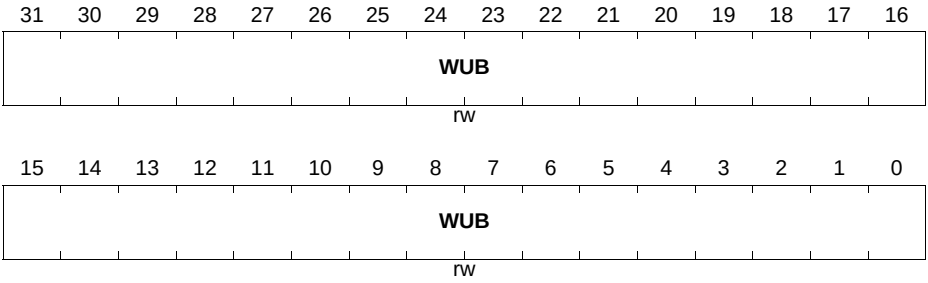
| Field | Bits | Type | Description |
|---|---|---|---|
| **WLB** | [31:0] | rw | **Window Lower Bound**<br>Lower bound for servicing window.<br>Setting the lower bound to $0_H$ disables the window mechanism. |

**WUB**

The Window Upper Bound register defines the upper bound for servicing window. Servicing of the watchdog has only effect within the window boundary.

**WUB**

**WDT Window Upper Bound Register ($14_H$)** **Reset Value: FFFF FFFF$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **WUB** | [31:0] | rw | **Window Upper Bound**<br>Upper Bound for servicing window.<br>The WDT triggers an reset request when the timer is crossing the upper bound value without pre-warning enabled.<br>With pre-warning enabled the first crossing triggers a watchdog alarm and the second crossing triggers a system reset. |

**WDTSTS**

The status register contains sticky bit indicating occurrence of alarm condition.

**WDTSTS**

**WDT Status Register** **($0018_H$)** **Reset Value: $00000000_H$**

| 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 | | | | | | | | | | | | | | | ALMS |
| r | | | | | | | | | | | | | | | rh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **ALMS** | 0 | rh | **Pre-warning Alarm**<br>$1_B$ pre-warning alarm occurred,<br>$0_B$ no pre-warning alarm occurred |
| **0** | [31:1] | r | **reserved** |

**WDTCLR**

The status register contains sticky bitfield indicating occurrence of alarm condition.

**WDTCLR**

**WDT Clear Register** **($001C_H$ )** **Reset Value: $00000000_H$**

| 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 | | | | | | | | | | | | | | | ALMC |
| r | | | | | | | | | | | | | | | w |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **ALMC** | 0 | w | **Pre-warning Alarm**<br>$1_B$ clears pre-warning alarm<br>$0_B$ no-action |
| **0** | [31:1] | r | **reserved** |

## 11.10 Interconnects

**Table 11-4 Pin Table**

| Input/Output | I/O | Connected To | Description |
|---|---|---|---|
| Clock and Reset Signals | | | |
| $f_{WDT}$ | I | SCU.CCU | timer clock |
| Timer Signals | | | |
| wdt_service | O | 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 |


Subject to Agreement on the Use of Product Information

# 12 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.

## 12.1 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 remains operational during sleep or deep sleep mode.

### 12.1.1 Features

The features of the Real Time Clock (RTC) module are:

- 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 12-1 Application Features**

| Feature | Purpose/Application |
|---|---|
| Precise real time keeping | Reduced need for time adjustments |
| Periodic time-based interrupt | Scheduling of operations performed on precisely defined intervals |
| Programmable alarm interrupt on time match | Scheduling of operations performed on precisely defined times |
| Supports wake-up mechanism from sleep or deep sleep mode | Autonomous wakeups from sleep or deep sleep mode for system state control and maintaintenance routine operations |

### 12.1.2 Block Diagram

The RTC block diagram is shown in **Figure 12-1**.

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

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-modulesof SCU.

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

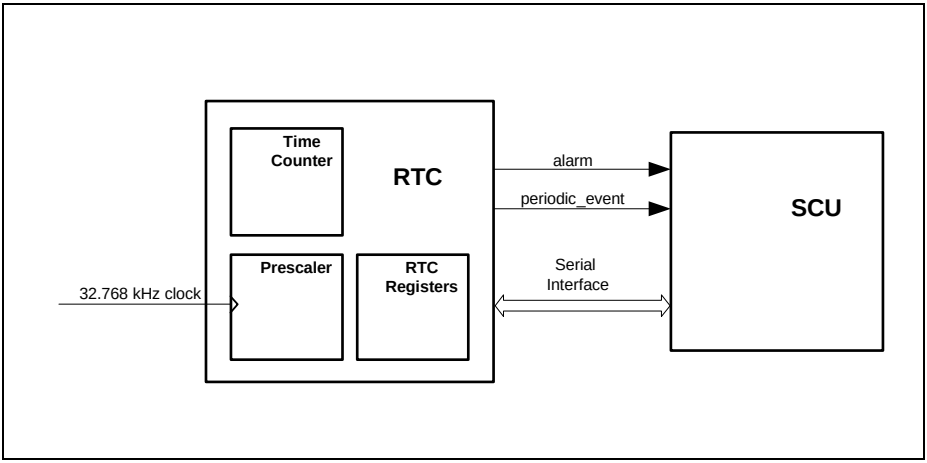



**Figure 12-1 Real Time Clock Block Diagram Structure**

## 12.2 RTC Operation

The RTC timer counts seconds, minutes, hours, days, days of week, months and years the time in separate fields (see **Figure 12-2**). Individual bit fields of the RTC counter can be programmed and read with software over serial interface via mirror registers in SCU module.

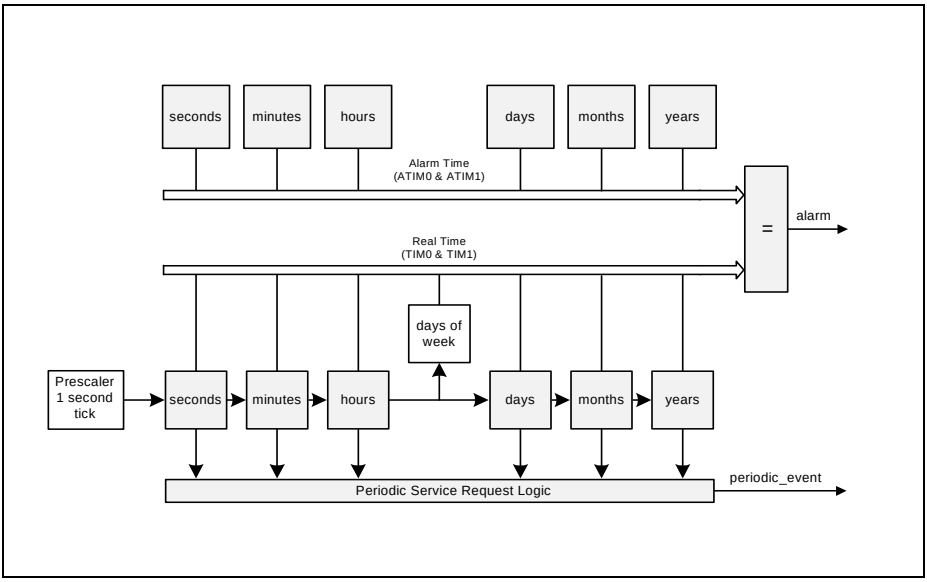



**Figure 12-2 Block Diagram of RTC Time Counter**

Occurrence of an internal timer event is stored in the service request raw status register **RAWSTAT**. The values of the status register **RAWSTAT** drive the outgoing service request lines alarm and periodic_event.

## 12.3 Register Access Operations

The RTC module is a part of SCU from programmming model perepcetive and shares register address space for configuration with other sub-modules of SCU. RTC registers are instantiated in the RTC module and mirrored in SCU. The registers get updated in both clock domains over serial interface running at 32kHz clock rate.

Any update of the registers is performed with some delay required for data to propagate to and from the mirror registers over serial interface. Accesses to the RTC registers in core domain must not block the bus interface of SCU module. For details of the register mirror and serial communication handling please refer to SCU chapter.

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.


Subject to Agreement on the Use of Product Information

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.

## 12.4 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.

### 12.4.1 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**.

### 12.4.2 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 .

## 12.5 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 XMC1300, 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 initialisation 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 CCU (Clock Gating Control) section of the SCU chapter.*

## 12.6 Power, Reset and Clock

RTC can be programmed to remains powered up in sleep and deep sleep mode.

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 12-3**.The prescaler setting of $7FFF_H$ results in an once per second update of the RTC timer.


Subject to Agreement on the Use of Product Information

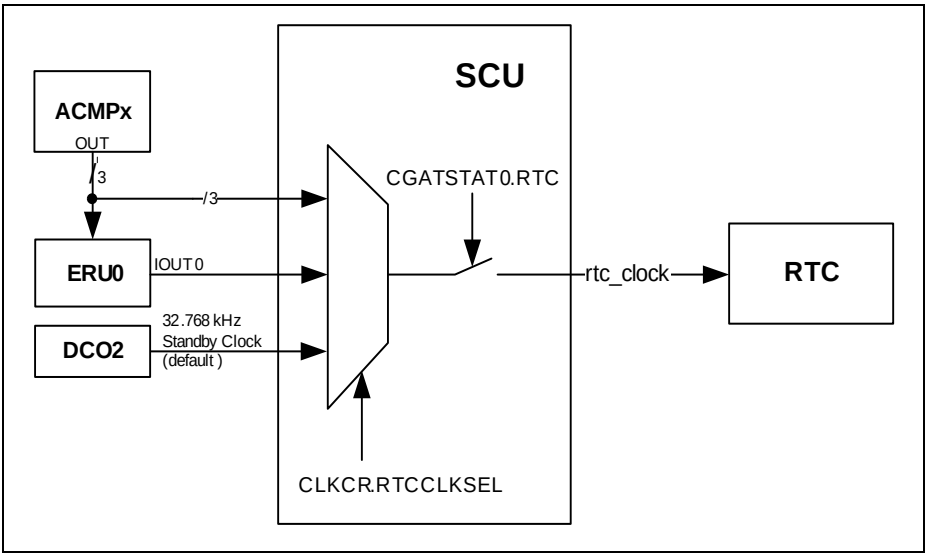



**Figure 12-3 RTC Clock selection**

The RTC module clock is default disabled 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.*

## 12.7 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 please refer to relevant chapters of this document.

### 12.7.1 Initialization & Start of Operation

Complete RTC module initialization is required upon reset. Accesses to RTC registers are performed via dedicated mirror registers (for more details please refer to SCU chapter)

- enable the clock to RTC module
  – write one to SCU_CGATCLR0.RTC
- program RTC_TIM0 and RTC_TIM1 registers with current time


Subject to Agreement on the Use of Product Information

  - 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.*

### 12.7.2 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

### 12.7.3 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.*

## 12.8 RTC Registers

**Registers Overview**

The absolute register address is calculated by adding:

Module Base Address + Offset Address

**Table 12-2 Registers Address Space**

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

**Table 12-3 Register Overview**

| Short Name | Description | Offset Addr. | Access Mode | | Description See |
|---|---|---|---|---|---|
| | | | Read | Write | |
| RTC Kernel Registers | | | | | |
| ID | ID Register | $0000_H$ | U, PV | BE | Page 12-7 |
| CTR | Control Register | $0004_H$ | U, PV | PV | Page 12-8 |
| RAWSTAT | Raw Service Request Register | $0008_H$ | U, PV | BE | Page 12-9 |
| STSSR | Status Service Request Register | $000C_H$ | U, PV | BE | Page 12-10 |
| MSKSR | Mask Service Request Register | $0010_H$ | U, PV | PV | Page 12-11 |
| CLRSR | Clear Service Request Register | $0014_H$ | U, PV | PV | Page 12-13 |
| ATIM0 | Alarm Time Register 0 | $0018_H$ | U, PV | PV | Page 12-14 |
| ATIM1 | Alarm Time Register 1 | $001C_H$ | U, PV | PV | Page 12-15 |
| TIM0 | Time Register 0 | $0020_H$ | U, PV | PV | Page 12-16 |
| TIM1 | Time Register 1 | $0024_H$ | U, PV | PV | Page 12-17 |

## 12.8.1 Registers Description

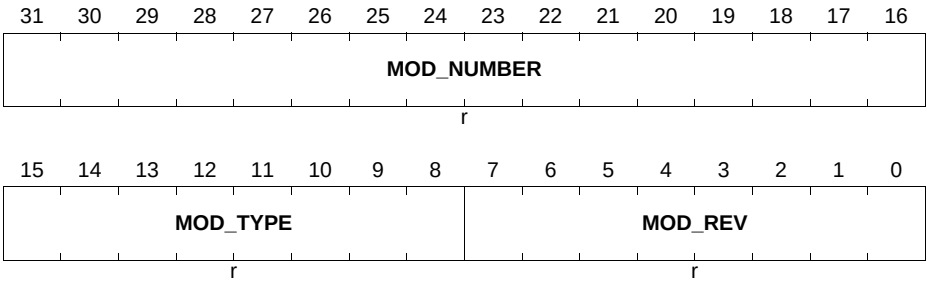
### ID

Read-only ID register of the RTC module containing unique identification code of the RTC module.


Subject to Agreement on the Use of Product Information

### ID
### RTC Module ID Register

($00_H$) Reset Value: $00A3\ C0XX_H$

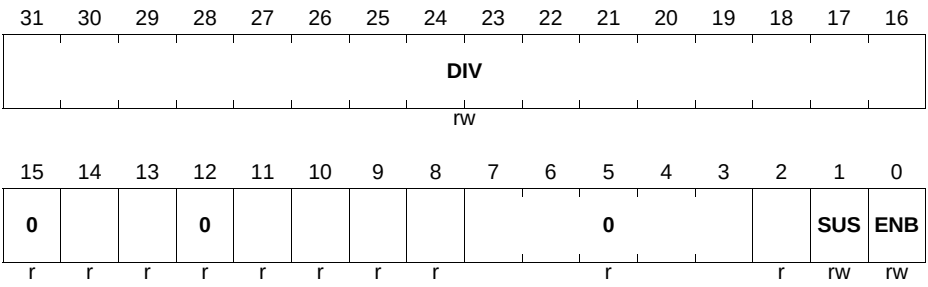



| Field | Bits | Type | Description |
|---|---|---|---|
| **MOD_REV** | [7:0] | r | **Module Revision Number**<br>Indicates the revision number of the implementation. The value of a module revision starts with $01_H$ (first revision). |
| **MOD_TYPE** | [15:8] | r | **Module Type**<br>This bit field is fixed to $C0_H$. |
| **MOD_NUMBER** | [31:16] | r | **Module Number Value**<br>This bit field defines the module identification number. |

### CTR

RTC Control Register providing control means of the operation mode of the module.

**CTR**
**RTC Control Register** **($04_H$)** **Reset Value: $7FFF\ 0000_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **ENB** | 0 | rw | **RTC Module Enable**<br>$0_B$ disables RTC module<br>$1_B$ enables RTC module |
| **SUS** | 1 | rw | **Debug Suspend Control**<br>$0_B$ RTC is not stopped during halting mode debug<br>$1_B$ RTC is stopped during halting mode debug |
| **DIV** | [31:16] | rw | **Divider Value**<br>reload value of RTC prescaler. Clock is divided by DIV+1.<br>$7FFF_H$ is default value for RTC mode with 32.768 kHz crystal or external clock |
| **0** | [15:2] | r | **Reserved**<br>Read as 0; should be written with 0 |

### RAWSTAT

RTC Raw Service Request Register contains raw status info i.e. before status mask takes effect on generation of service requests or iterrupts. This register serves debug purpose but can be aslo used for polling of the status withou generating serice requests.


Subject to Agreement on the Use of Product Information

**RAWSTAT**

**RTC Raw Service Request Register (08$_H$)** **Reset Value: 0000 0000$_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | RAI | 0 | RPYE | RPMO | 0 | RPDA | RPHO | RPMI | RPSE |
| r | | | | | | | rh | r | rh | rh | r | rh | rh | rh | rh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **RPSE** | 0 | rh | **Raw Periodic Seconds Service Request**<br>Set whenever seconds count increments |
| **RPMI** | 1 | rh | **Raw Periodic Minutes Service Request**<br>Set whenever minutes count increments |
| **RPHO** | 2 | rh | **Raw Periodic Hours Service Request**<br>Set whenever hours count increments |
| **RPDA** | 3 | rh | **Raw Periodic Days Service Request**<br>Set whenever days count increments |
| **RPMO** | 5 | rh | **Raw Periodic Months Service Request**<br>Set whenever months count increments |
| **RPYE** | 6 | rh | **Raw Periodic Years Service Request**<br>Set whenever years count increments |
| **RAI** | 8 | rh | **Alarm Service Request**<br>Set whenever count value matches compare value |
| **0** | 4, 7, [31:9] | r | **Reserved** |

**STSSR**

RTC Service Request Status Register contains status info reflecting status mask effect on generation of service requests or iterrupts. This register needs to be accessed by software in order to determine the actual cause of an event.


Subject to Agreement on the Use of Product Information

**STSSR**

**RTC Service Request Status Register ($0C_H$)** **Reset Value: $0000\ 0000_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | SAI | 0 | SPYE | SPMO | 0 | SPDA | SPHO | SPMI | SPSE |
| r | | | | | | | rh | r | rh | rh | r | rh | rh | rh | rh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **SPSE** | 0 | rh | **Periodic Seconds Service Request Status after masking** |
| **SPMI** | 1 | rh | **Periodic Minutes Service Request Status after masking** |
| **SPHO** | 2 | rh | **Periodic Hours Service Request Status after masking** |
| **SPDA** | 3 | rh | **Periodic Days Service Request Status after masking** |
| **SPMO** | 5 | rh | **Periodic Months Service Request Status after masking** |
| **SPYE** | 6 | rh | **Periodic Years Service Request Status after masking** |
| **SAI** | 8 | rh | **Alarm Service Request Status after masking** |
| **0** | 4, 7, [31:9] | r | **reserved** |

**MSKSR**

RTC Service Request Mask Register contains masking value for generation control of service requests or iterrupts.


Subject to Agreement on the Use of Product Information

**MSKSR**

**RTC Service Request Mask Register (10$_H$)** **Reset Value: 0000 0000$_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | MAI | 0 | MPYE | MPMO | 0 | MPDA | MPHO | MPMI | MPSE |
| r | | | | | | | rw | r | rw | rw | r | rw | rw | rw | rw |

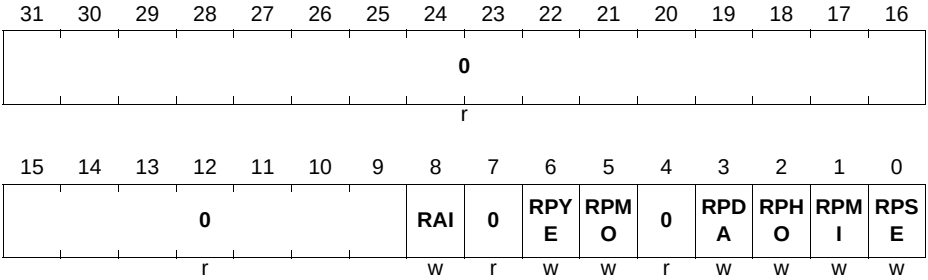
| Field | Bits | Type | Description |
|---|---|---|---|
| **MPSE** | 0 | rw | **Periodic Seconds Interrupt Mask**<br>$0_B$ disable interrupt<br>$1_B$ enable interrupt |
| **MPMI** | 1 | rw | **Periodic Minutes Interrupt Mask**<br>$0_B$ disable interrupt<br>$1_B$ enable interrupt |
| **MPHO** | 2 | rw | **Periodic Hours Interrupt Mask**<br>$0_B$ disable interrupt<br>$1_B$ enable interrupt |
| **MPDA** | 3 | rw | **Periodic Days Interrupt Mask**<br>$0_B$ disable interrupt<br>$1_B$ enable interrupt |
| **MPMO** | 5 | rw | **Periodic Months Interrupt Mask**<br>$0_B$ disable interrupt<br>$1_B$ enable interrupt |
| **MPYE** | 6 | rw | **Periodic Years Interrupt Mask**<br>$0_B$ disable interrupt<br>$1_B$ enable interrupt |
| **MAI** | 8 | rw | **Alarm Interrupt Mask**<br>$0_B$ disable interrupt<br>$1_B$ enable interrupt |
| **0** | 4, 7, [31:9] | r | **Reserved** |


Subject to Agreement on the Use of Product Information

### CLRSR

RTC Clear Service Request Register serves purpose of clearing sticky bits of **RAWSTAT** and **STSSR** registers. Write one to a bit in order to clear the status bit. Writing zero has no effect on the set nor reset bits.

**CLRSR**
**RTC Clear Service Request Register (14$_H$)** **Reset Value: 0000 0000$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **RPSE** | 0 | w | **Raw Periodic Seconds Interrupt Clear**<br>$0_B$ no effect<br>$1_B$ clear status bit |
| **RPMI** | 1 | w | **Raw Periodic Minutes Interrupt Clear**<br>$0_B$ no effect<br>$1_B$ clear status bit |
| **RPHO** | 2 | w | **Raw Periodic Hours Interrupt Clear**<br>$0_B$ no effect<br>$1_B$ clear status bit |
| **RPDA** | 3 | w | **Raw Periodic Days Interrupt Clear**<br>$0_B$ no effect<br>$1_B$ clear status bit |
| **RPMO** | 5 | w | **Raw Periodic Months Interrupt Clear**<br>$0_B$ no effect<br>$1_B$ clear status bit |
| **RPYE** | 6 | w | **Raw Periodic Years Interrupt Clear**<br>$0_B$ no effect<br>$1_B$ clear status bit |

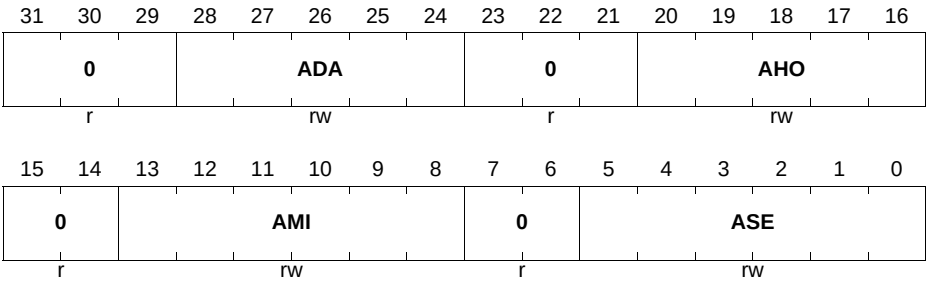

Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **RAI** | 8 | w | **Raw Alarm Interrupt Clear**<br>$0_B$ no effect<br>$1_B$ clear status bit |
| **0** | 4,<br>7,<br>[31:9] | r | **Reserved** |

### ATIM0

RTC Alarm Time Register 0 serves purpose of programming single alarm time at a desired point of time reflecting comparison against **TIM0** register. The ATM0 register contains portion of bit fields for seconds, minutes, hours and days. Upon attempts to write an invalid value to a bit bield e.g. exceeding maximum value a default value gets programmed as described for each individual bit fields.

**ATIM0**
**RTC Alarm Time Register 0** **($18_H$)** **Reset Value: 0000 0000$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **ASE** | [5:0] | rw | **Alarm Seconds Compare Value**<br>Match of seconds timer count to this value triggers alarm seconds interrupt.<br>Setting value equal or above $3C_H$ results in setting the field value to $0_H$ |
| **AMI** | [13:8] | rw | **Alarm Minutes Compare Value**<br>Match of minutes timer count to this value triggers alarm minutes interrupt.<br>Setting value equal or above $3C_H$ results in setting the field value to $0_H$ |

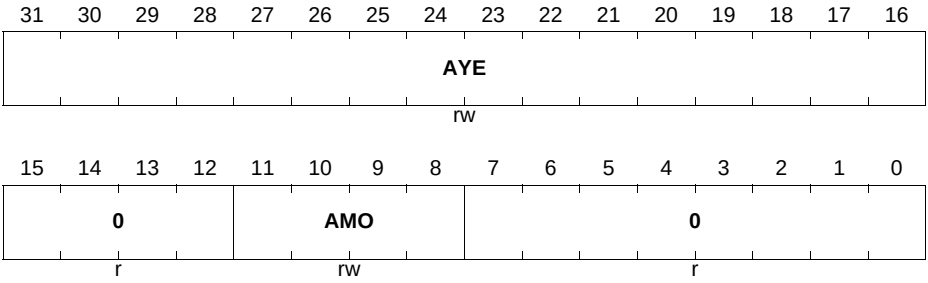

Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **AHO** | [20:16] | rw | **Alarm Hours Compare Value**<br>Match of hours timer count to this value triggers alarm hours interrupt.<br>Setting value equal or above $18_H$ results in setting the field value to $0_H$ |
| **ADA** | [28:24] | rw | **Alarm Days Compare Value**<br>Match of days timer count to this value triggers alarm days interrupt.<br>Setting valueequal above $1F_H$ results in setting the field value to $0_H$ |
| **0** | [7:6], [15:14], [23:21], [31:29] | r | **Reserved** |

### ATIM1

RTC Alarm Time Register 1 serves purpose of programming single alarm time at a desired point of time reflecting comparison against **TIM1** register. The ATM1 register contains portion of bit fields for months and years. Upon attempts to write an invalid value to a bit bield e.g. exceeding maximum value a default value gets programmed as described for each individual bit fields.

**ATIM1**
**RTC Alarm Time Register 1 ($1C_H$) Reset Value: 0000 $0000_H$**





Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **AMO** | [11:8] | rw | **Alarm Month Compare Value**<br>Match of months timer count to this value triggers alarm month interrupt.<br>Setting value equal or above the number of days of the actual month count results in setting the field value to $0_H$ |
| **AYE** | [31:16] | rw | **Alarm Year Compare Value**<br>Match of years timer count to this value triggers alarm years interrupt. |
| **0** | [7:0], [15:12] | r | **Reserved** |

**TIM0**

RTC Time Register 0 contains current time value for seconds, minutes, hours and days. The bit fields get updated in intervals corresponding with their meaning accordingly. The register needs to be programmed to reflect actual time after initial power up and will continue counting time also while in sleep or deep sleep mode if enabled. Upon attempts to write an invalid value to a bit bield e.g. exceeding maximum value a default value gets programmed as described for each individual bit fields.

**TIM0**

**RTC Time Register 0** **($20_H$)** **Reset Value: 0000 0000$_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | | | **DA** | | | | | **0** | | | **HO** | | | | |
| r | | | rwh | | | | | r | | | rwh | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | | | **MI** | | | | | **0** | | | **SE** | | | | |
| r | | | rwh | | | | | r | | | rwh | | | | |


Subject to Agreement on the Use of Product Information

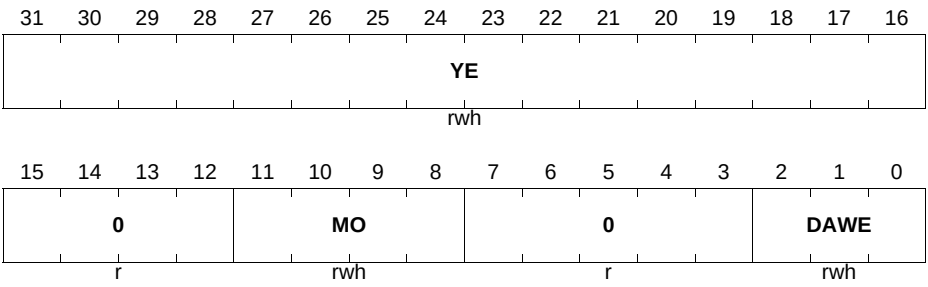
| Field | Bits | Type | Description |
|---|---|---|---|
| **SE** | [5:0] | rwh | **Seconds Time Value**<br>Setting value equal or above $3C_H$ results in setting the field value to $0_H$.<br>Value can only be written, when RTC is disabled via bit CTR.ENB. |
| **MI** | [13:8] | rwh | **Minutes Time Value**<br>Setting value equal or above $3C_H$ results in setting the field value to $0_H$.<br>Value can only be written, when RTC is disabled via bit CTR.ENB. |
| **HO** | [20:16] | rwh | **Hours Time Value**<br>Setting value equal or above $18_H$ results in setting the field value to $0_H$<br>Value can only be written, when RTC is disabled via bit CTR.ENB. |
| **DA** | [28:24] | rwh | **Days Time Value**<br>Setting value equal or above the number of days of the actual month count results in setting the field value to $0_H$<br>Value can only be written, when RTC is disabled via bit CTR.ENB.<br>Days counter starts with value 0 for the first day of month. |
| **0** | [7:6], [15:14], [23:21], [31:29] | r | **Reserved** |

**TIM1**

RTC Time Register 1 contains current time value for days of week, months and years. The bit fields get updated in intervals corresponding with their meaning accordingly. The register needs to be programmed to reflect actual time after initial power up and will continue counting time also while in sleep or deep sleep if enabled. Upon attempts to write an invalid value to a bit bield e.g. exceeding maximum value a default value gets programmed as described for each individual bit fields.


Subject to Agreement on the Use of Product Information

**TIM1**
**RTC Time Register 1** **($24_H$)** **Reset Value: 0000 0000$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **DAWE** | [2:0] | rwh | **Days of Week Time Value**<br>Setting value above $6_H$ results in setting the field value to $0_H$<br>Value can only be written, when RTC is disabled via bit CTR.ENB.<br>Days counter starts with value 0 for the first day of week. |
| **MO** | [11:8] | rwh | **Month Time Value**<br>Setting value equal or above $C_H$ results in setting the field value to $0_H$.<br>Value can only be written, when RTC is disabled via bit CTR.ENB.<br>Months counter starts with value 0 for the first month of year. |
| **YE** | [31:16] | rwh | **Year Time Value**<br>Value can only be written, when RTC is disabled. |
| **0** | [7:3], [15:12] | r | **Reserved** |

## 12.9 Interconnects

**Table 12-4 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 |


Subject to Agreement on the Use of Product Information

# 13 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.

## 13.1 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 **Chapter 13.2**
- Power control, PCU **Chapter 13.3**
- Reset operation, RCU **Chapter 13.4**
- Clock Control, CCU **Chapter 13.5**

### 13.1.1 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


Subject to Agreement on the Use of Product Information

### 13.1.2 Block Diagram

**Figure 13-1** 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 SCU module are accessible via the AHB and the 16-bit APB bus interface as shown in **Figure 13-1**.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 SFR called SCU register are accesible via the AHB bus interface.

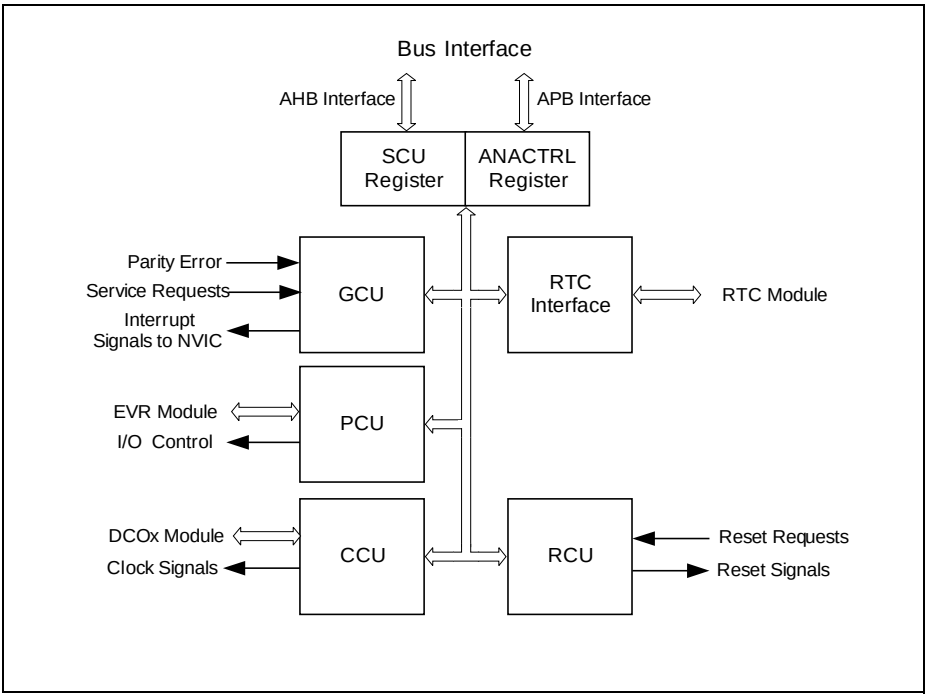



**Figure 13-1 SCU Block Diagram**

### Interface of General Control Unit

The General Control Unit GCU has a memory fault interface to the memory validation logic of each on-chip SRAM and the Flash to receive memory fault events, as parity errors.

### Interface of Power Control Unit

The Power Control Unit 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 **Chapter 13.3**.

### Interface of Reset Control Unit

The Reset Control Unit 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 the clock control unit (CCU). The RCU is providing the reset signals to all other units of the chip in the core power domain. The RCU related signals are described in more detail in **Chapter 13.4**.

### Interface of Clock Control Unit

The Clock Control Unit (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 separate RTC chapter.


Subject to Agreement on the Use of Product Information

## 13.2 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).

### 13.2.1 Service Requests Handling

Service request events listed in **Table 13-1** can result in assertion of an interrupt. Please refer to **SRMSK** a register description.

The interrupt structure is shown in **Figure 13-2**. The interrupt request or the corresponding interrupt set bit (in register **SRSET**) can trigger the interrupt generation at the selected interrupt node x. The service request pulse is generated independently from the interrupt flag in register **SRRAW**. The interrupt flag can be cleared by software by writing to the corresponding bit in register **SRCLR**.

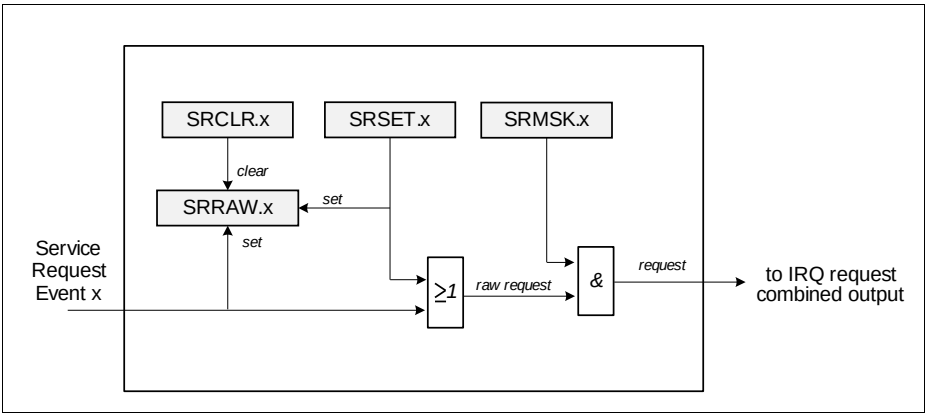



**Figure 13-2 Service Request Handling**

The flag in register **SRRAW** can be cleared by software by writing to the corresponding bit in register **SRCLR**. 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.*

#### 13.2.1.1 Service Request Sources

The SCU supports service request sources listed in **Table 13-1** and reflected in the **SRRAW**, **SRMSK**, **SRCLR** and **SRSET** registers. The events that trigger these service

requests are described in the respective module chapters or in the various sections within SCU.

**Table 13-1 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 | |
| SCU:CCU | Loss of Clock Event | LOCI | |
| | Standby Clock Failure Event | SBYCLKFI | SR1 |
| 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 | |


Subject to Agreement on the Use of Product Information

### 13.2.2 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** status register. It is configurable via **SRMSK** whether a memory error should trigger an interrupt. It can also trigger a system reset when **RSTCON**.SPERSTEN or **RSTCON**.U0PERSTEN 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.*

### 13.2.3 Summary of ID

This section describes the various ID in XMC1300.

#### 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 13-2**. The XMC1300 system ROM table is located at F000 $0000_H$. Cortex-M0 ROM table is described in the debug chapter.

**Table 13-2 PID Values of XMC1300 System ROM Table**

| Name | Offset | Reference | Values |
|---|---|---|---|
| PID0 | $FE0_H$ | XMC1300 Part Number [7:0] | $ED_H$ |
| PID1 | $FE4_H$ | bits [7:4] JEP106 ID code [3:0]<br>bits [3:0] XMC1300 Part Number [11:8] | $11_H$ |
| PID2 | $FE8_H$ | bits [7:4] XMC1300 Revision<br>bit [3] == 1: JEDEC assigned ID fields<br>bits [2:0] JEP106 ID code [6:4] | $1C_H$ |

**Table 13-2 PID Values of XMC1300 System ROM Table**

| Name | Offset | Reference | Values |
|---|---|---|---|
| PID3 | $FEC_H$ | bits [7:4] RevAnd, minor revision field<br>bits [3:0] if non-zero indicate a customer-modified block | $00_H$ |
| PID4 | $FD0_H$ | bits [7:4] 4KB count<br>bits [3:0] JEP106 continuation code | $00_H$ |

### Chip Identification Number

The chip identification number is a 8 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, example, package type, the temperature profile, flash size, RAM size and the peripheral availability.


Subject to Agreement on the Use of Product Information

## 13.3 Power Management (PCU)

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

### 13.3.1 Functional Description

The XMC1300 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.

### 13.3.2 System States

The system has the following general system states:

- Off
- Active
- Sleep
- Deep-Sleep

**Figure 13-3** shows the state diagram and the transitions between the states.

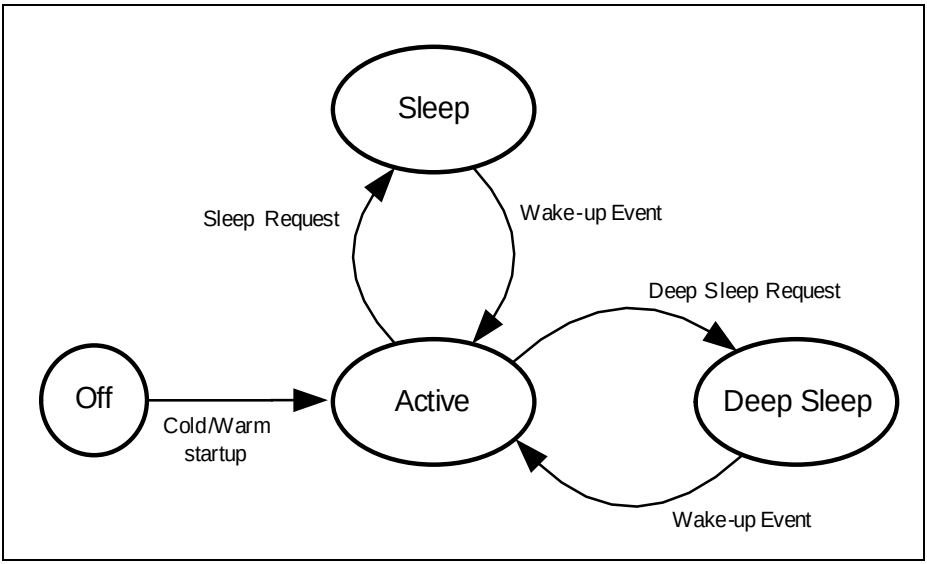



**Figure 13-3 System States Diagram**

### 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 similiar 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.*

### 13.3.3 Embedded Voltage Regulator (EVR)

The EVR generates the core voltage $V_{DDC}$ out of the external supplied voltage $V_{DDP}$. The EVR provides 2 supply monitoring detectors for the input voltage $V_{DDP}$. The generated core voltage $V_{DDC}$ is monitored by a power validation circuit (PV).

### 13.3.4 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.

### 13.3.5 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.

### 13.3.6 Supply Voltage Monitoring

There are 2 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. Similiarly, it is also used to ensure a proper startup during the power-up phase.

The Data Sheet defines the nominal value and applied hysteresis


Subject to Agreement on the Use of Product Information

### 13.3.7 $V_{DDC}$ Response During Load Change

In XMC1300, 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. 2 detectors, VDROP and VCLIP detectors are used to montior 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 monitior via SRRAW register.

In XMC1300, 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 happens (< 4*baseload or > 0.25*baseload), irregardless of an increase or decrease in load, the EVR needs time (15 usec) 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 16mA from the current baseload of 1mA needs at least 2 steps. A step from 1mA to 4mA followed by another step from 4mA to 16mA*

The VDDC2LOW and VDDC2HIGH status bit is generated by a 10-bit counter using DCO1, 64MHz clock as the clock input. It is implemented to count the 16 usec (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 1mA to 4mA, user can poll for VDDC2LOW (CNTADJ=$3FF_H$) to ensure the 16 usec(max) needed to regulate EVR. After VDDC2LOW is set to 0, another load change from 4mA to 16mA 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" on Page 13-17**. CPU clock and peripheral clocks continue to run during VCLIP event.


Subject to Agreement on the Use of Product Information

*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.*

### 13.3.8 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.


Subject to Agreement on the Use of Product Information

## 13.4 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

### 13.4.1 Functional Description

The XMC1300 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 additional, 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 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.


Subject to Agreement on the Use of Product Information

### 13.4.2 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 13-3** shows an overview of the reset signals their source and effects on the various parts of the system.

**Table 13-3 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 section of Debug System chapter 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.


Subject to Agreement on the Use of Product Information

## 13.5 Clock Control (CCU)

### 13.5.1 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

### 13.5.2 Clock System and Control

**Figure 13-4** shows the block diagram of the clock system in XMC1300. It consists of two oscillator (DCO1[1)] and DCO2) and a clock control unit (CCU). DCO1 has a clock output, dco1_dclk running at 64MHz. DCO2 is used to generate the standby clock running at 32.768kHz. The main clock, MCLK, and fast peripherial clock, PCLK, are generated from dco1_dclk. PCLK is running in either the same frequency as MCLK or double the frequency of MCLK. It is selectable via CLKCR.PCLKSEL.

**Figure 13-4** 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. RTC and WDT is running at a frequency of 32.768kHz from the standby clock which is asychronous to the MCLK and PCLK clock.

1) The accuracy of the DCO1 clock output can be improved by calibrating it based on the die temperatue given by the temperature sensor. Refer to **Section 13.5.4** for detailed description.


Subject to Agreement on the Use of Product Information

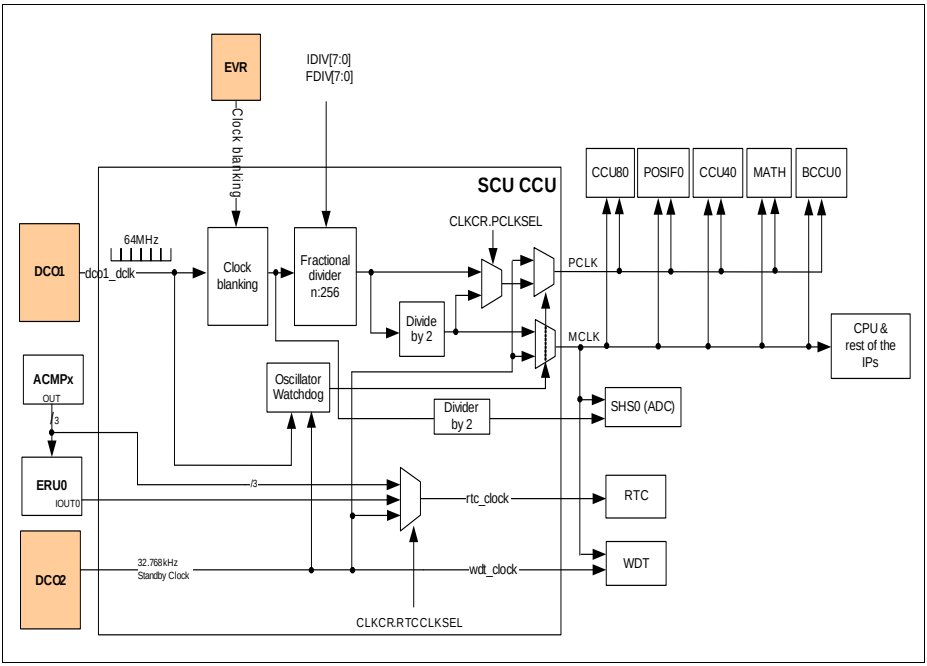



**Figure 13-4 Clock System Block Diagram**

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

*Note: RTC_extclk are further described in RTC chapter .*

### Fractional Divider

The frequency of MCLK and PCLK are programmable through a fractional divider. PCLK has a range of frequency from 125kHz to 64MHz and MCLK has a range from 125kHz to 32MHz.

The following formula calculate the MCLK clock frequency.

(13.1)

$$\left( MCLK = \frac{dco_dclk}{(2) \times \left( IDIV + \frac{FDIV}{256} \right)} \right) \text{for IDIV} > 0$$

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

(13.2)

$$\left( PCLK = \frac{dco_dclk}{\left( IDIV + \frac{FDIV}{256} \right)} \right) \text{for IDIV} > 0$$

IDIV represents an unsigned 8-bit integer from the bit field CLKCR.IDIV and FDIV/256 defines the fractional divider selection in CLKCR.FDIV. Changing of the MCLK and PCLK can be done within 2 clock cycles. While changing the frequency of MCLK clock and PCLK clock, it is recommended to disable all interrupts to prevent any access to flash that may results in an unsuccessful flash operation.

*Note: Changing the MCLK and PCLK frequency may result in a load change that causes clock blanking to happen. Refer to **Clock Blanking** and **VDDC 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 freezed the clock input to the fractional divider to regulate the


Subject to Agreement on the Use of Product Information

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 resume. 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 resume.

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 usec (max). Detail description of the core voltage behaviors during load change are described in **"VDDC Response During Load Change" on Page 13-11**.

As described in **"VDDC Response During Load Change" on Page 13-11**, 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 .*

### 13.5.2.1 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[1)], 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.

### 13.5.2.2 Loss of Clock Detection and Recovery

Loss of clock happens when the oscillator watchdog (OWD) detects a DCO1 frequency that is less than 50 MHz or more than 68.5 MHz during normal operation. In this case,

1) It is recommended to clear the status bit SRRAW.LOCI and SRRAW.SBYCLKFI before restarting the detection.

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 32.768kHz. Emergency routines can be executed to safely shut down the system. Beside triggering an interrupt when the loss of clock happens, a system reset can also be triggered if it is enabled by **RSTCON**.LOCRSTEN.

*Note: Switching to standby clock during loss of clock event happens only if DCO2 is still running (>0MHz). Bit **SRRAW**.SBYCLKFI indicates the fail status of the standby clock.*

The XMC1300 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 50 MHz and lower than 68.5MHz, OSC2L and OSC2H will be set to 0 and the MCLK will switched automatically to the DCO1 clock source.

#### 13.5.2.3 Standby Clock Failure

The standby clock failure event happens when the OWD detects a failure in standby clock where the clock stops running (~0kHz). Bit SRRAW.SBYCLKFI is set to 1 and trigger an interrupt via SCU_SR1 service request if the interrupt is enabled in register SRMSK.

#### 13.5.2.4 Startup Control for System Clock

When the XMC1300 starts up after reset, system frequency is provided by the DCO1 oscillator. After reset, CPU runs in 8MHz, 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 fexibility of device performance e.g. speed vs power consumption optimisation during start-up. For details, please refer to the section “Clock system handling by SSW” in the “Boot and Startup” chapter.

### 13.5.3 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 chapter 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


Subject to Agreement on the Use of Product Information

load would required the system to maintain in the current load and no further load change is allowed for 16 usec (max). See **VDDC 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.

### 13.5.4 Calibrating DCO based on Temperature

In XMC1300, DCO1 clock frequency can be calibrated during runtime to achieve a better accuracy. Based on the measured temperature using the on-chip temperature sensor(TSE), an offset value can be obtained using the formulae as shown below:

(13.3)

$$OFFSET[steps] = b + \frac{(a-b)(c-d)}{(e-d)}$$

where :

OFFSET value is range from 0 to 8

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 4 constants in the formulae are stored in flash configuration page shown in **Table 13-4**.

**Table 13-4 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 |


Subject to Agreement on the Use of Product Information

**Table 13-4 DCO calibration data in Flash sector 0 (CS0)**

| Address | Length | Function |
|---|---|---|
| $1000'0F32_H$ | 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 8. This bit field is bit protected. DCO1 will take about 5 usec(max) for clock to be adjusted to the new frequency.

An example code is provided in the Osciilator Handling Device Guide.

## 13.6 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 TSE 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 **Section 13.2.1** for more details.

## 13.7 Debug Behavior

The SCU module does not get affected with the HALTED signal from SCU upon debug activities performed using external debug probe.

## 13.8 Power, Reset and Clock

The SCU module implements functions that involve varoius 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 synchchronization.

- 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


Subject to Agreement on the Use of Product Information

**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 (please 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


Subject to Agreement on the Use of Product Information

## 13.9 Registers

This section describes the registers of SCU which some of them resides in the ANACTRL module. Most of the registers are reset by 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 13-5 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 13-6 Registers Address Space**

| Module | Base Address | End Address | Note |
|---|---|---|---|
| SCU | $4001\ 0000_H$ | $4001\ FFFF_H$ | System Control Unit Registers |


Subject to Agreement on the Use of Product Information

**Table 13-7 Registers Overview**

| Short Name | Description | Offset Addr.[1)] | Access Mode | | Description See |
|---|---|---|---|---|---|
| | | | Read | Write | |
| **PCU Registers (ANACTRL)** | | | | | |
| ANAVDEL | Voltage Detector Control register | $1050_H$ | U, PV | U, PV | Page 13-25 |
| **PCU Registers (SCU)** | | | | | |
| VDESR | Voltage Detector Status Register | $0000_H$ | U, PV | U, PV | Page 13-26 |
| **CCU Registers (SCU)** | | | | | |
| CLKCR | Clock Control | $0000_H$ | U, PV | U, PV, BP | Page 13-27 |
| PWRSVCR | Power Save Control Register | $0004_H$ | U, PV | U, PV | Page 13-29 |
| CGATSTAT0 | Clock Gating Status for Peripherals 0 | $0008_H$ | U, PV | U, PV | Page 13-30 |
| CGATSET0 | Clock Gating Set for Peripherals 0 | $000C_H$ | U, PV | U, PV, BP | Page 13-31 |
| CGATCLR0 | Clock Gating Clear for Peripherals 0 | $0010_H$ | U, PV | U, PV, BP | Page 13-33 |
| OSCCSR | Oscillator Control and Status Register | $0014_H$ | U, PV | U, PV | Page 13-34 |
| **CCU Registers (ANACTRL)** | | | | | |
| ANAOFFSET | DCO1 Offset Register | $106C_H$ | U, PV | U, PV, BP | Page 13-36 |
| **RCU Registers (SCU)** | | | | | |
| RSTSTAT | Reset Status | $0000_H$ | U, PV | BE | Page 13-37 |
| RSTSET | Reset Set Register | $0004_H$ | U, PV | U, PV | Page 13-38 |
| RSTCLR | Reset Clear Register | $0008_H$ | U, PV | U, PV | Page 13-38 |
| RSTCON | Reset Control Register | $000C_H$ | U, PV | U, PV | Page 13-39 |
| **GCU Registers (SCU)** | | | | | |
| ID | Module Identification Register | $0008_H$ | U, PV | BE | Page 13-41 |
| IDCHIP | Chip ID | $0004_H$ | U, PV | SP | Page 13-42 |

**Table 13-7 Registers Overview** (cont'd)

| Short Name | Description | Offset Addr.[1] | Access Mode | | Description See |
|---|---|---|---|---|---|
| | | | Read | Write | |
| DBGROMID | DBGROMID | $0000_H$ | U, PV | SP | Page 13-43 |
| SSW0 | SSW Support Register | $0014_H$ | U, PV | U, PV | Page 13-43 |
| CCUCON | CCUx Global Start Control Register | $0030_H$ | U, PV | U, PV | Page 13-44 |
| SRRAW | RAW Service Request Status | $0038_H$ | U, PV | BE | Page 13-45 |
| SRMSK | Service Request Mask | $003C_H$ | U, PV | U, PV | Page 13-48 |
| SRCLR | Service Request Clear | $0040_H$ | U, PV | U, PV | Page 13-50 |
| SRSET | Service Request Set | $0044_H$ | U, PV | U, PV | Page 13-53 |
| PASSWD | Bit protection Register | $0024_H$ | U, PV | U, PV | Page 13-56 |
| MIRRSTS | Mirror Update Status Register | $0048_H$ | U, PV | BE | Page 13-57 |
| PMTSR | Parity Memory Test Select Register | $0054_H$ | U, PV | U, PV | Page 13-58 |

1) The absolute register address is calculated as follows:
Module Base Address + Sub-Module Offset Address + Offset Address (shown in this column)

## 13.9.1 PCU Registers (ANACTRL)

### ANAVDEL

Voltage Detector Control register.

**ANAVDEL**
**Voltage Detector Control Register ($1050_H$)** **Reset Value:$001C_H$**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | | | | VDEL_EN | VDEL_TIM_ADJ | | VDEL_SELECT | |
| r | | | | | | | | | | | rw | rw | | rw | |

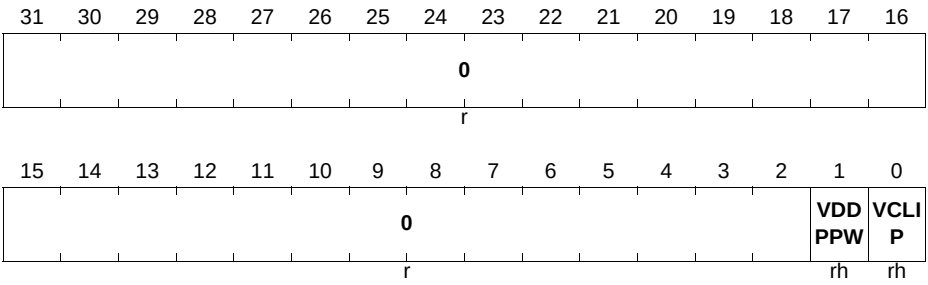
| Field | Bits | Type | Description |
|---|---|---|---|
| **VDEL_SELECT** | 1:0 | rw | **VDEL Range Select**<br>With these bits the VDDP range is set.<br>$00_B$ 2.25V<br>$01_B$ 3.0V<br>$10_B$ 4.4V |
| **VDEL_TIM_ADJ** | 3:2 | rw | **VDEL Timing Setting**<br>These bits control the reaction speed of the VDEL.<br>The value is determined by characterisation.<br>$00_B$ typ 1µs - slowest response time<br>$01_B$ typ 500n<br>$10_B$ typ 250n<br>$11_B$ no delay - fastest response time. |
| **VDEL_EN** | 4 | rw | **VDEL unit Enable**<br>$0_B$ VDEL is disabled<br>$1_B$ VDEL is active |
| **0** | 15:5 | r | **Reserved**<br>Read as 0; should be written with 0. |

## 13.9.2 PCU Registers (SCU)

### VDESR

Voltage Detector status register.

**VDESR**
**Voltage Detector Status Register ($0200_H$) Reset Value: $0000\ 0000_H$**





Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **VCLIP** | 0 | rh | **VCLIP Indication**<br>VCLIP monitoring bit.<br>$0_B$ VCLIP is not active<br>$1_B$ VCLIP is active |
| **VDDPPW** | 1 | rh | **VDDPPW Indication**<br>$0_B$ VDDP is above pre-warning threshold<br>$1_B$ VDDP is below pre-warningthreshold |
| **0** | [31:2] | r | **Reserved**<br>Read as 0; should be written with 0. |

### 13.9.3 CCU Registers (SCU)

#### CLKCR

Clock control register.

**CLKCR**
**Clock Control Register** **($0300_H$)** **Reset Value: 3FF0 $0400_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| VDDC2HIGH | VDDC2LOW | CNTADJ | | | | | | | | | | RTCCLKSEL | | | PCLKSEL |
| rh | rh | rw | | | | | | | | | | rw | | | rw |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| IDIV | | | | | | | | FDIV | | | | | | | |
| rw | | | | | | | | rw | | | | | | | |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Function |
|---|---|---|---|
| **FDIV** | [7:0] | rw | **Fractional Divider Selection**<br>Selects the fractional divider to be n/256, where n is the value of FDIV and is in the range of 0 to 255.<br>For example, writing $0001_B$ to FDIV selects the fractional divider to be 1/256.<br>This bit is protected by the bit protection scheme as described in Memory Organization chapter<br>*Note: Fractional divider has no effect if IDIV = $00_H$ .* |
| **IDIV** | [15:8] | rw | **Divider Selection**<br>$00_H$ Divider is bypassed.<br>$01_H$ 1; MCLK = 32 MHz<br>$02_H$ 2; MCLK = 16 MHz<br>$03_H$ 3; MCLK = 10.67 MHz<br>$04_H$ 4; MCLK = 8 MHz<br>$FE_H$ 254; MCLK = 126 kHz<br>$FF_H$ 255; MCLK = 125.5 kHz<br>This bit is protected by the bit protection scheme as described in Memory Organization chapter |
| **PCLKSEL** | 16 | rw | **PCLK Clock Select**<br>$0_B$ PCLK = MCLK<br>$1_B$ PCLK = 2 x MCLK<br>This bit is protected by the bit protection scheme as described in Memory Organization chapter |
| **RTCCLKSEL** | [19:17] | rw | **RTC Clock Select**<br>$000_B$ 32.768kHz standby clock<br>$001_B$ 32.768kHz external clock from ERU0.IOUT0<br>$010_B$ 32.768kHz external clock from ACMP0.OUT<br>$011_B$ 32.768kHz external clock from ACMP1.OUT<br>$100_B$ 32.768kHz external clock from ACMP2.OUT<br>$101_B$ Reserved<br>$110_B$ Reserved<br>$111_B$ Reserved<br>This bit is protected by the bit protection scheme as described in Memory Organization chapter |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Function |
|---|---|---|---|
| **CNTADJ** | [29:20] | rw | **Counter Adjustment**<br>$000_H$ 1 clock cycles of the DCO1, 64MHz clock<br>$001_H$ 2 clock cycles of the DCO1, 64MHz clock<br>$002_H$ 3 clock cycles of the DCO1, 64MHz clock<br>$003_H$ 4 clock cycles of the DCO1, 64MHz clock<br>$004_H$ 5 clock cycles of the DCO1, 64MHz clock<br>$3FE_H$ 1023 clock cycles of the DCO1, 64MHz clock<br>$3FF_H$ 1024 clock cycles of the DCO1, 64MHz clock |
| **VDDC2LOW** | 30 | rh | **VDDC too low**<br>$0_B$ VDDC is not too low and the fractional divider input clock is running at the targeted frequency<br>$1_B$ VDDC is too low and the fractional divider input clock is not running at the targeted frequency |
| **VDDC2HIGH** | 31 | rh | **VDDC too high**<br>$0_B$ VDDC is not too high<br>$1_B$ VDDC is too high |

### PWRSVCR

Configuration register that defines some system behaviour aspects while in deep-sleep mode or sleep mode. The original system state gets restored upon wakeup from sleep mode or deep-sleep mode.

**PWRSVCR**
**Power Save Control Register ($0304_H$) Reset Value: $0000\ 0000_H$**

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

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


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **FPD** | 0 | rw | **Flash Power Down**<br>$0_B$ no effect<br>$1_B$ Flash power down when entering power save mode. Upon wake-up, CPU is able to fetch code from flash. |
| **0** | [31:1] | r | **Reserved**<br>Read as 0. |

### CGATSTAT0

Clock gating status for XMC1300 peripherals. After reset, all peripherals as listed in the registers are not running. Their module clock are gated.

Every bit in this register is protected by the bit protection scheme as described in Memory Organization chapter.

**CGATSTAT0**
**Peripheral 0 Clock Gating Status ($0308_H$) Reset Value: 0000 07FF$_H$**

| 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 | | | | | RTC | WDT | MATH | POSIF0 | 1 | | BCCU0 | USIC0 | CCU40 | CCU80 | VADC |
| r | | | | | r | r | r | r | r | | r | r | r | r | r |

| Field | Bits | Type | Description |
|---|---|---|---|
| **VADC** | 0 | r | **VADC and SHS Gating Status**<br>$0_B$ gating de-asserted<br>$1_B$ gating asserted |
| **CCU80** | 1 | r | **CCU80 Gating Status**<br>$0_B$ gating de-asserted<br>$1_B$ gating asserted |

| Field | Bits | Type | Description |
|---|---|---|---|
| **CCU40** | 2 | r | **CCU40 Gating Status**<br>$0_B$ gating de-asserted<br>$1_B$ gating asserted |
| **USIC0** | 3 | r | **USIC0 Gating Status**<br>$0_B$ gating de-asserted<br>$1_B$ gating asserted |
| **BCCU0** | 4 | r | **BCCU0 Gating Status**<br>$0_B$ gating de-asserted<br>$1_B$ gating asserted |
| **POSIF0** | 7 | r | **POSIF0 Gating Status**<br>$0_B$ gating de-asserted<br>$1_B$ gating asserted |
| **MATH** | 8 | r | **MATH Gating Status**<br>$0_B$ gating de-asserted<br>$1_B$ gating asserted |
| **WDT** | 9 | r | **WDT Gating Status**<br>$0_B$ gating de-asserted<br>$1_B$ gating asserted |
| **RTC** | 10 | r | **RTC Gating Status**<br>$0_B$ gating de-asserted<br>$1_B$ gating asserted |
| **0** | [31:11] | r | **Reserved** |
| **1** | [6:5] | r | **Reserved** |

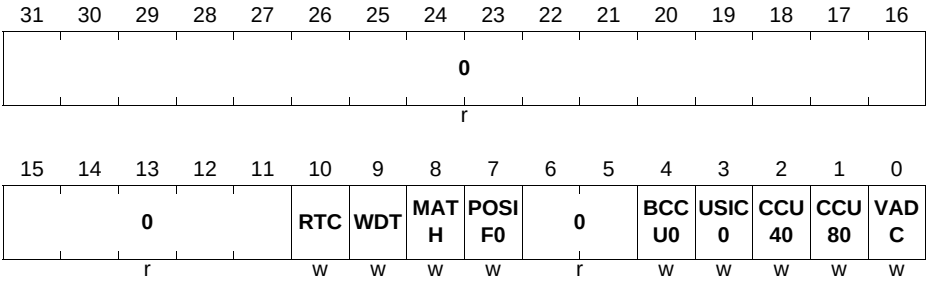
**CGATSET0**

Clock gating enable for XMC1300 peripherals. Write one to selected bit to enable gating of corresponding clock, writing zeros has no effect.

Every bit in this register is protected by the bit protection scheme as described in Memory Organization chapter.


Subject to Agreement on the Use of Product Information

**CGATSET0**
**Peripheral 0 Clock Gating Set** **($030C_H$)** **Reset Value: $0000\ 0000_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **VADC** | 0 | w | **VADC and SHS Gating Set**<br>$0_B$ no effect<br>$1_B$ enable gating |
| **CCU80** | 1 | w | **CCU80 Gating Set**<br>$0_B$ no effect<br>$1_B$ enable gating |
| **CCU40** | 2 | w | **CCU40 Gating Set**<br>$0_B$ no effect<br>$1_B$ enable gating |
| **USIC0** | 3 | w | **USIC0 Gating Set**<br>$0_B$ no effect<br>$1_B$ enable gating |
| **BCCU0** | 4 | w | **BCCU0 Gating Set**<br>$0_B$ no effect<br>$1_B$ enable gating |
| **POSIF0** | 7 | w | **POSIF0 Gating Set**<br>$0_B$ no effect<br>$1_B$ enable gating |
| **MATH** | 8 | w | **MATH Gating Set**<br>$0_B$ no effect<br>$1_B$ enable gating |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **WDT** | 9 | w | **WDT Gating Set**<br>$0_B$ no effect<br>$1_B$ enable gating |
| **RTC** | 10 | w | **RTC Gating Set**<br>$0_B$ no effect<br>$1_B$ enable gating |
| **0** | [6:5], [31:11] | r | **Reserved** |

### CGATCLR0

Clock gating disable for XMC1300 peripherals. Write one to selected bit to disable gating of corresponding clock, writing zeros has no effect.

### CGATCLR0

**Peripheral 0 Clock Gating Clear** **($0310_H$)** **Reset Value: $0000\ 0000_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | RTC | WDT | MATH | POSIF0 | 0 | | BCCU0 | USIC0 | CCU40 | CCU80 | VADC |
| r | | | | | w | w | w | w | r | | w | w | w | w | w |

| Field | Bits | Type | Description |
|---|---|---|---|
| **VADC** | 0 | w | **VADC and SHS Gating Clear**<br>$0_B$ no effect<br>$1_B$ disable gating |
| **CCU80** | 1 | w | **CCU80 Gating Clear**<br>$0_B$ no effect<br>$1_B$ disble gating |
| **CCU40** | 2 | w | **CCU40 Gating Clear**<br>$0_B$ no effect<br>$1_B$ disable gating |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **USIC0** | 3 | w | **USIC0 Gating Clear**<br>$0_B$ no effect<br>$1_B$ disable gating |
| **BCCU0** | 4 | w | **BCCU0 Gating Clear**<br>$0_B$ no effect<br>$1_B$ disable gating |
| **POSIF0** | 7 | w | **POSIF0 Gating Clear**<br>$0_B$ no effect<br>$1_B$ disable gating |
| **MATH** | 8 | w | **MATH Gating Clear**<br>$0_B$ no effect<br>$1_B$ disable gating |
| **WDT** | 9 | w | **WDT Gating Clear**<br>$0_B$ no effect<br>$1_B$ disable gating |
| **RTC** | 10 | w | **RTC Gating Clear**<br>$0_B$ no effect<br>$1_B$ disable gating |
| **0** | [6:5], [31:11] | r | **Reserved** |

### OSCCSR

Oscillator Control and Status Register.

### OSCCSR
### Oscillator Control and Status Register

**($0314_H$)** **Reset Value: 0000 000$X_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | | | | | | | | | | | | | | **OWDEN** | **OWDRES** |
| r | | | | | | | | | | | | | | rw | rwh |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | | | | | | | | | | | | | | **OSC2H** | **OSC2L** |
| r | | | | | | | | | | | | | | rh | rh |

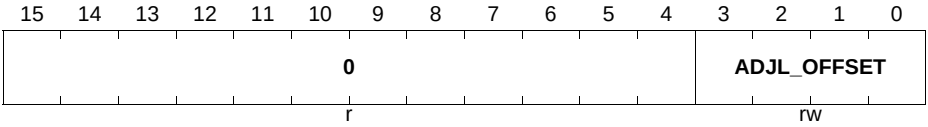
| Field | Bits | Type | Description |
|---|---|---|---|
| **OSC2L** | 0 | rh | **Oscillator Valid Low Status Bit**<br>This bit indicates if the frequency output of OSC is usable. This is checked by the Oscillator Watchdog<br>$0_B$ The OSC frequency is usable<br>$1_B$ The OSC frequency is not usable. Frequency is too low. |
| **OSC2H** | 1 | rh | **Oscillator Valid High Status Bit**<br>This bit indicates if the frequency output of OSC is usable. This is checked by the Oscillator Watchdog.<br>$0_B$ The OSC frequency is usable<br>$1_B$ The OSC frequency is not usable. Frequency is too high. |
| **OWDRES** | 16 | rwh | **Oscillator Watchdog Reset**<br>Setting this bit will restart the oscillator detection. This bit will be automatically reset to 0 after OWD is reset which takes 2 standby clock cycles due to synchronisation.<br>$0_B$ The Oscillator Watchdog is not cleared and remains active<br>$1_B$ The Oscillator Watchdog is cleared and restarted. The OSC2L and OSC2H flag will be held in the last value until it is updated after 3 standby clock cycles. |
| **OWDEN** | 17 | rw | **Oscillator Watchdog Enable**<br>$0_B$ The Oscillator Watchdog is disabled<br>$1_B$ The Oscillator Watchdog is enabled<br>*Note: OSC2H and OSC2L will be cleared to 0 when OWD is disabled.* |
| **0** | [15:2], [31:18] | r | **Reserved**<br>Read as 0. |

### 13.9.4 CCU Registers (ANACTRL)

**ANAOFFSET**

DCO1 Offset register.

**ANAOFFSET**
**DCO1 Offset Register** **(106C$_H$)** **Reset Value: 0004$_H$**



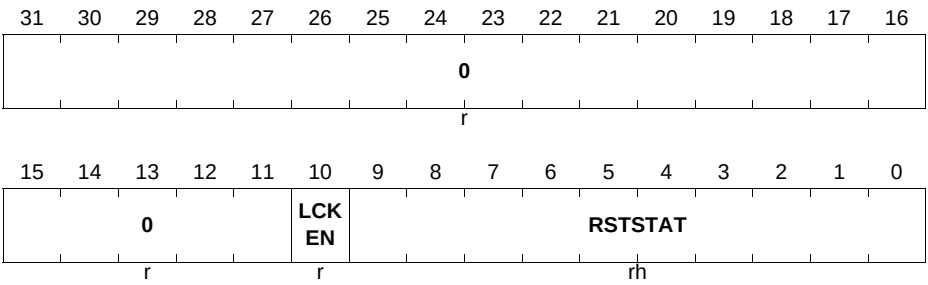

| Field | Bits | Type | Description |
|---|---|---|---|
| **ADJL_OFFSET** | 3:0 | rw | **ADJL Offset register**<br>The adjusted oscillator frequency can be varied according to the steps programmed.<br>This bit is protected by the bit protection scheme as described in Memory Organization chapter<br>$0_H$ - 3.75%, typ.<br>$1_H$ - 2.85%, typ.<br>**...**<br>$4_H$ 0, default<br>$5_H$ + 0.95%, typ.<br>**...**<br>$8_H$ + 3.75%, typ. |
| **0** | 15:4 | r | **Reserved**<br>Read as 0; should be written with 0. |

## 13.9.5 RCU Registers (SCU)

### RSTSTAT

Reset status register. This register needs to be checked after system startup in order to determine last reset reason. User should clear this register after reading it to ensure a clear status when the next reset happen. This register is reset by a power-on reset.

**RSTSTAT**
**RCU Reset Status** **($0400_H$)** **Reset Value: 0000 0XXX$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **RSTSTAT** | [9:0] | rh | **Reset Status Information**<br>Provides reason of last reset<br>$0000000001_B$ Power on reset or Brownout reset<br>$XXXXXXX1X_B$ Master reset via bit RSTCON.MRSTEN<br>$XXXXXX1XX_B$ CPU system reset request<br>$XXXXX1XXX_B$ CPU lockup reset<br>$XXXX1XXXX_B$ Flash ECC reset<br>$XXX1XXXXX_B$ WDT reset<br>$XX1XXXXXX_B$ Loss of clock reset<br>$X1XXXXXXX_B$ Parity Error reset |
| **LCKEN** | 10 | r | **Enable Lockup Status**<br>$0_B$ Reset by Lockup disabled<br>$1_B$ Reset by Lockup enabled |
| **0** | [31:11] | r | **Reserved** |

**RSTSET**

Selective configuration of reset behaviour in the system. Write one to set selected bit, writing zeros has no effect.

**RSTSET**

**RCU Reset Set Register** **($0404_H$)** **Reset Value: 0000 0000$_H$**

| 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 | | | | | LCKEN | 0 | | | | | | | | | |
| r | | | | | w | r | | | | | | | | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **LCKEN** | 10 | w | **Enable Lockup Reset**<br>$0_B$ no effect<br>$1_B$ Enable reset when Lockup gets asserted |
| **0** | [9:0], [31:11] | r | **Reserved** |

**RSTCLR**

Selective configuration of reset behaviour in the system. Write one to clear selected bit, writing zeros has no effect.

**RSTCLR**

**RCU Reset Clear Register** **($0408_H$)** **Reset Value: 0000 0000$_H$**

| 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 | | | | | LCKEN | 0 | | | | | | | | | RSCLR |
| r | | | | | w | r | | | | | | | | | w |

| Field | Bits | Type | Description |
|---|---|---|---|
| **RSCLR** | 0 | w | **Clear Reset Status**<br>$0_B$ no effect<br>$1_B$ Clears field **RSTSTAT.RSTSTAT** |
| **LCKEN** | 10 | w | **Enable Lockup Reset**<br>$0_B$ no effect<br>$1_B$ Disable reset when Lockup gets asserted |
| **0** | [9:1], [31:11] | r | **Reserved** |

### RSTCON

Enabling of reset triggered by critical events. It is reset by any reset type.

**RSTCON**
**RCU Reset Control Register** **($040C_H$)** **Reset Value: 0000 $0000_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | | | | | U0PERSTEN | SPERSTEN | LOCRSTEN | ECCRSTEN |
| r | | | | | | | | | | | | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **ECCRSTEN** | 0 | rw | **Enable ECC Error Reset**<br>$0_B$ No reset when ECC double bit error occur<br>$1_B$ Reset when ECC double bit error occur |
| **LOCRSTEN** | 1 | rw | **Enable Loss of Clock Reset**<br>$0_B$ No reset when loss of clock occur<br>$1_B$ Reset when loss of clock occur |
| **SPERSTEN** | 2 | rw | **Enable 16kbytes SRAM Parity Error Reset**<br>$0_B$ No reset when SRAM parity error occur<br>$1_B$ Reset when SRAM parity error occur |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **U0PERSTEN** | 3 | rw | **Enable USIC0 SRAM Parity Error Reset**<br>$0_B$ No reset when USIC0 memory parity error occur<br>$1_B$ Reset when USIC0 memory parity error occur |
| **MRSTEN** | 16 | w | **Enable Master Reset**<br>$0_B$ No effect<br>$1_B$ Triggered Master reset |
| **0** | [15:4], [31:17] | r | **Reserved** |


Subject to Agreement on the Use of Product Information

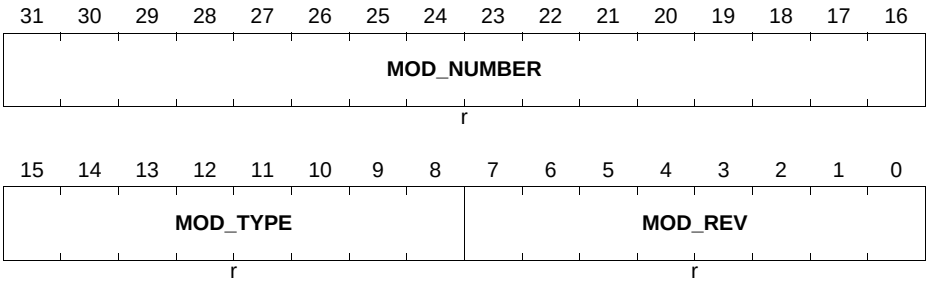
### 13.9.6 GCU Registers (SCU)

**ID**

Register containing uniques ID of the module.

**ID**
**SCU Module ID Register**

($0008_H$) Reset Value: $00F1\ C0XX_H$

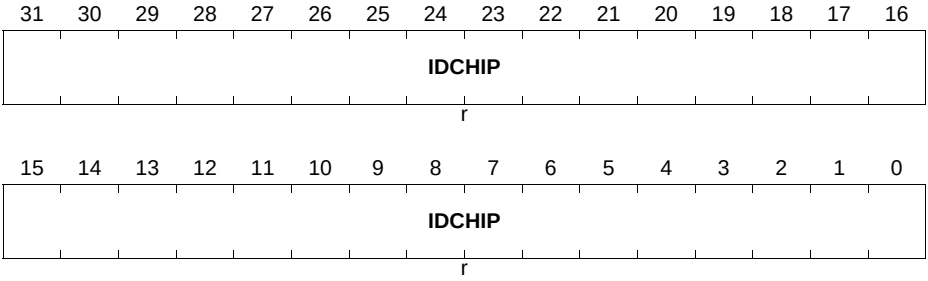



| Field | Bits | Type | Description |
|---|---|---|---|
| **MOD_REV** | [7:0] | r | **Module Revision Number**<br>MOD_REV defines the revision number. The value of a module revision starts with $01_H$ (first revision). |
| **MOD_TYPE** | [15:8] | r | **Module Type**<br>This bit field is $C0_H$. It defines the module as a 32-bit module. |
| **MOD_NUMBER** | [31:16] | r | **Module Number Value**<br>This bit field defines the module identification number. |

**IDCHIP**

Register containing a unique ID of the chip in the XMC family. The value of this register formed part of the chip identification number as described in **Chip Identification Number**.

**IDCHIP**
**Chip ID Register** **($0004_H$)** **Reset Value: 0000 $0000_H$**

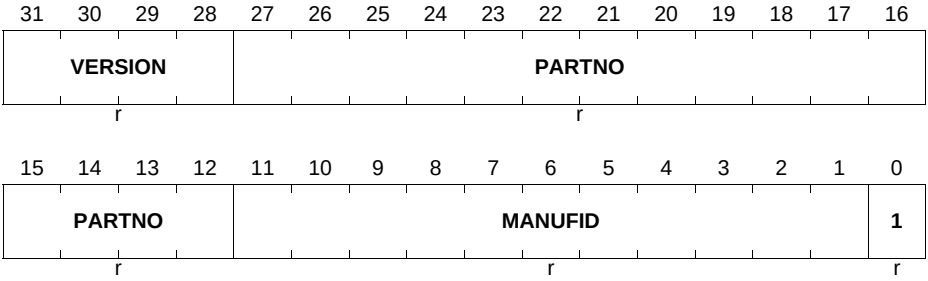



| Field | Bits | Type | Description |
|---|---|---|---|
| **IDCHIP** | [31:0] | r | **CHIP ID**<br>0001 $3XXX_H$ XMC1300<br>0001 $XXX2_H$ temperature : -40 - 85 °C<br>0001 $XXX3_H$ temperature : -40 - 105 °C<br>0001 $XX1X_H$ TSSOP38 pin package<br>0001 $XX2X_H$ TSSOP28 pin package<br>0001 $XX3X_H$ TSSOP16 pin package<br>0001 $XX4X_H$ VQFN40 pin package<br>0001 $XX6X_H$ VQFN24 pin package<br>Others Reserved |

**DBGROMID**

Register containing unique manufactory ID, part number and the design stepping code of the chip.

**DBGROMID**
**Debug System ROM ID Register ($0000_H$) Reset Value: 201E D083$_H$**



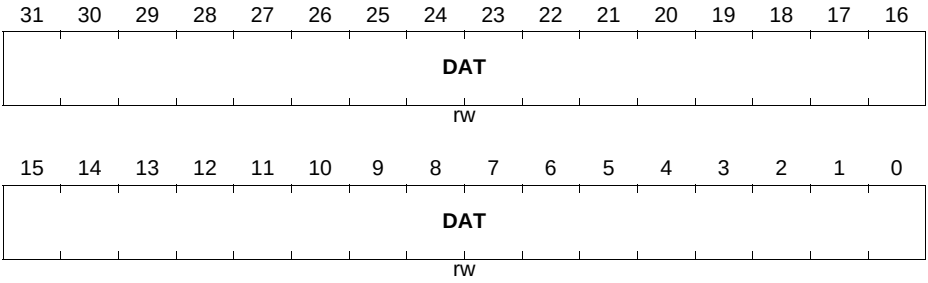

| Field | Bits | Type | Description |
|---|---|---|---|
| **MANUFID** | [11:1] | r | **Manufactory Identity** |
| **PARTNO** | [27:12] | r | **Part Number** |
| **VERSION** | [31:28] | r | **Product version** |
| **1** | 0 | r | **Reserved**<br>Read as 1; should be written with 1. |

**SSW0**

Software support registers.

SSW0 is used to change the BMI value.

**SSW0**
**SSW Register 0 ($0014_H$) Reset Value: 0000 0000$_H$**





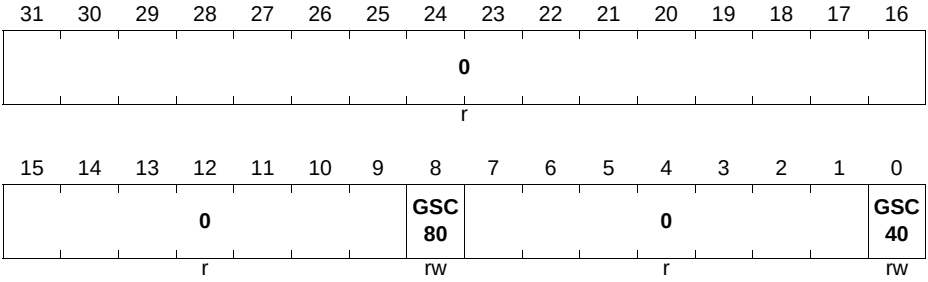
Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **DAT** | [31:0] | rw | **SSW Data**<br>*Note: SSW registers can be reset with master reset only* |

### CCUCON

CAPCOM module control register.

**CCUCON**
**CCU Control Register** **($0030_H$)** **Reset Value: $0000\ 0000_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **GSC40** | 0 | rw | **Global Start Control CCU40**<br>This register can be used to control a synchronous start of multiple timers of CCU40.<br>It also can be used to control additional functions that are available in the timers, e.g. Count or Capture. For a complete description of the available set of functions, please address the specific section of the CCU4 chapters.<br>Writing 1 or 0 into this field will not by itself trigger a synchronous start of multiple timers and one must before configure the specific peripheral function accordingly. |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **GSC80** | 8 | rw | **Global Start Control CCU80**<br>This register can be used to control a synchronous start of multiple timers of CCU80.<br>It also can be used to control additional functions that are available in the timers, e.g. Count or Capture. For a complete description of the available set of functions, please address the specific section of the CCU8 chapters.<br>Writing 1 or 0 into this field will not by itself trigger a synchronous start of multiple timers and one must before configure the specific peripheral function accordingly. |
| **0** | [7:1], [31:9] | r | **Reserved**<br>Read as 0; should be written with 0. |

### SRRAW

Service request status without masking. Write one to a bit in SRCLR register to clear a bit or SRSET to set a bit. Writing zero has no effect.

### SRRAW

**SCU Raw Service Request Status (0038$_H$)** **Reset Value: 0000 0000$_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **TSE_LOW** | **TSE_HIGH** | **TSE_DONE** | **RTC_TIM1** | **RTC_TIM0** | **RTC_ATIM1** | **RTC_ATIM0** | **RTC_CTR** | **0** | **SBYCLKFI** | **VCLIPI** | **FLCMPLTI** | **FLECC2I** | **PEU0I** | **PESRAMI** | **LOCI** |
| rh | rh | rh | rh | rh | rh | rh | rh | r | rh | rh | rh | rh | rh | rh | rh |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **ORC7I** | **ORC6I** | **ORC5I** | **ORC4I** | **ORC3I** | **ORC2I** | **ORC1I** | **ORC0I** | **VDROPI** | **ACMP2I** | **ACMP1I** | **ACMP0I** | **VDDPI** | **AI** | **PI** | **PRWARN** |
| rh | rh | rh | rh | rh | rh | rh | rh | rh | rh | rh | rh | rh | rh | rh | rh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **PRWARN** | 0 | rh | **WDT pre-warning Event Status Before Masking**<br>$0_B$ Event has not occurred<br>$1_B$ Event has occurred |

| Field | Bits | Type | Description |
|---|---|---|---|
| **PI** | 1 | rh | **RTC Raw Periodic Event Status Before Masking**<br>$0_B$ Event has not occurred<br>$1_B$ Event has occurred |
| **AI** | 2 | rh | **RTC Raw Alarm Event Status Before Masking**<br>$0_B$ Event has not occurred<br>$1_B$ Event has occurred |
| **VDDPI** | 3 | rh | **VDDP pre-warning Event Status Before Masking**<br>$0_B$ Event has not occurred<br>$1_B$ Event has occurred |
| **ACMP0I** | 4 | rh | **Analog Comparator 0 Event Status Before Masking**<br>$0_B$ Event has not occurred<br>$1_B$ Event has occurred |
| **ACMP1I** | 5 | rh | **Analog Comparator 1 Event Status Before Masking**<br>$0_B$ Event has not occurred<br>$1_B$ Event has occurred |
| **ACMP2I** | 6 | rh | **Analog Comparator 2 Event Status Before Masking**<br>$0_B$ Event has not occurred<br>$1_B$ Event has occurred |
| **VDROPI** | 7 | rh | **VDROP Event Status Before Masking**<br>$0_B$ Event has not occurred<br>$1_B$ Event has occurred |
| **ORCxI (x=0-7)** | x+8 | rh | **Out of Range Comparator X Event Status Before Masking**<br>$0_B$ Event has not occurred<br>$1_B$ Event has occurred |
| **LOCI** | 16 | rh | **Loss of Clock Event Status Before Masking**<br>$0_B$ Event has not occurred<br>$1_B$ Event has occurred |
| **PESRAMI** | 17 | rh | **16kbytes SRAM Parity Error Event Status Before Masking**<br>$0_B$ Event has not occurred<br>$1_B$ Event has occurred |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **PEU0I** | 18 | rh | **USIC0 SRAM Parity Error Event Status Before Masking**<br>$0_B$ Event has not occurred<br>$1_B$ Event has occurred |
| **FLECC2I** | 19 | rh | **Flash Double Bit ECC Event Status Before Masking**<br>$0_B$ Event has not occurred<br>$1_B$ Event has occurred |
| **FLCMPLTI** | 20 | rh | **Flash Operation Complete Event Status Before Masking**<br>$0_B$ Event has not occurred<br>$1_B$ Event has occurred |
| **VCLIPI** | 21 | rh | **VCLIP Event Status Before Masking**<br>$0_B$ Event has not occurred<br>$1_B$ Event has occurred |
| **SBYCLKFI** | 22 | rh | **Standby Clock Failure Event Status Before Masking**<br>$0_B$ No standby clock failure has occurred<br>$1_B$ Standby clock failure has occurred |
| **RTC_CTR** | 24 | rh | **RTC CTR Mirror Register Update Status Before Masking**<br>$0_B$ not updated<br>$1_B$ update completed |
| **RTC_ATIM0** | 25 | rh | **RTC ATIM0 Mirror Register Update Status Before Masking**<br>$0_B$ not updated<br>$1_B$ update completed |
| **RTC_ATIM1** | 26 | rh | **RTC ATIM1 Mirror Register Update Status Before Masking**<br>$0_B$ not updated<br>$1_B$ update completed |
| **RTC_TIM0** | 27 | rh | **RTC TIM0 Mirror Register Update Before Masking**<br>$0_B$ not updated<br>$1_B$ update completed |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **RTC_TIM1** | 28 | rh | **RTC TIM1 Mirror Register Update Status Before Masking**<br>$0_B$ not updated<br>$1_B$ update completed |
| **TSE_DONE** | 29 | rh | **TSE Measurement Done Event Status Before Masking**<br>$0_B$ Event has not occurred<br>$1_B$ Event has occurred |
| **TSE_HIGH** | 30 | rh | **TSE Compare High Temperature Event Status Before Masking**<br>$0_B$ Event has not occurred<br>$1_B$ Event has occurred |
| **TSE_LOW** | 31 | rh | **TSE Compare Low Temperature Event Status Before Masking**<br>$0_B$ Event has not occurred<br>$1_B$ Event has occurred |
| **0** | 23 | r | **Reserved** |

**SRMSK**

Service request mask used to mask outputs of RAW register. When the bit is set to 1, an interrupt or service request will be triggered when the event happens.

**SRMSK**
**SCU Service Request Mask** **($003C_H$)** **Reset Value: $0000\ 0000_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **TSE_LOW** | **TSE_HIGH** | **TSE_DONE** | **RTC_TIM1** | **RTC_TIM0** | **RTC_ATIM1** | **RTC_ATIM0** | **RTC_CTR** | **0** | **SBYCLKFI** | **VCLIPI** | **0** | **FLECC2I** | **PEU0I** | **PESRAMI** | **LOCI** |
| rw | rw | rw | rw | rw | rw | rw | rw | r | rw | rw | r | rw | rw | rw | rw |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **ORC7I** | **ORC6I** | **ORC5I** | **ORC4I** | **ORC3I** | **ORC2I** | **ORC1I** | **ORC0I** | **VDROPI** | **ACMP2I** | **ACMP1I** | **ACMP0I** | **VDDPI** | **0** | | **PRWARN** |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | r | | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **PRWARN** | 0 | rw | **WDT pre-warning Interrupt Mask**<br>$0_B$ disable interrupt<br>$1_B$ enable interrupt |
| **VDDPI** | 3 | rw | **VDDP pre-warning Interrupt Mask**<br>$0_B$ disable interrupt<br>$1_B$ enable interrupt |
| **ACMP0I** | 4 | rw | **Analog Comparator 0 Interrupt Mask**<br>$0_B$ disable interrupt<br>$1_B$ enable interrupt |
| **ACMP1I** | 5 | rw | **Analog Comparator 1 Interrupt Mask**<br>$0_B$ disable interrupt<br>$1_B$ enable interrupt |
| **ACMP2I** | 6 | rw | **Analog Comparator 2 Interrupt Mask**<br>$0_B$ disable interrupt<br>$1_B$ enable interrupt |
| **VDROPI** | 7 | rw | **VDROP Interrupt Mask**<br>$0_B$ disable interrupt<br>$1_B$ enable interrupt |
| **ORCxI (x=0-7)** | x+8 | rw | **Out of Range Comparator X Interrupt Mask**<br>$0_B$ disable interrupt<br>$1_B$ enable interrupt |
| **LOCI** | 16 | rw | **Loss of Clock Interrupt Mask**<br>$0_B$ disable interrupt<br>$1_B$ enable interrupt |
| **PESRAMI** | 17 | rw | **16kbytes SRAM Parity Error Interrupt Mask**<br>$0_B$ disable interrupt<br>$1_B$ enable interrupt |
| **PEU0I** | 18 | rw | **USIC0 SRAM Parity Error Interrupt Mask**<br>$0_B$ disable interrupt<br>$1_B$ enable interrupt |
| **FLECC2I** | 19 | rw | **Flash Double Bit ECC Interrupt Mask**<br>$0_B$ disable interrupt<br>$1_B$ enable interrupt |
| **VCLIPI** | 21 | rw | **VCLIP Interrupt Mask**<br>$0_B$ disable interrupt<br>$1_B$ enable interrupt |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **SBYCLKFI** | 22 | rw | **Standby Clock Failure Interrupt Mask**<br>$0_B$ disable interrupt<br>$1_B$ enable interrupt |
| **RTC_CTR** | 24 | rw | **RTC CTR Mirror Register Update Mask**<br>$0_B$ disable interrupt<br>$1_B$ enable interrupt |
| **RTC_ATIM0** | 25 | rw | **RTC ATIM0 Mirror Register Update Mask**<br>$0_B$ disable interrupt<br>$1_B$ enable interrupt |
| **RTC_ATIM1** | 26 | rw | **RTC ATIM1 Mirror Register Update Mask**<br>$0_B$ disable interrupt<br>$1_B$ enable interrupt |
| **RTC_TIM0** | 27 | rw | **RTC TIM0 Mirror Register Update Mask**<br>$0_B$ disable interrupt<br>$1_B$ enable interrupt |
| **RTC_TIM1** | 28 | rw | **RTC TIM1 Mirror Register Update Mask**<br>$0_B$ disable interrupt<br>$1_B$ enable interrupt |
| **TSE_DONE** | 29 | rw | **TSE Measurement Done Interrupt Mask**<br>$0_B$ disable interrupt<br>$1_B$ enable interrupt |
| **TSE_HIGH** | 30 | rw | **TSE Compare High Temperature Interrupt Mask**<br>$0_B$ disable interrupt<br>$1_B$ enable interrupt |
| **TSE_LOW** | 31 | rw | **TSE Compare Low Temperature Interrupt Mask**<br>$0_B$ disable interrupt<br>$1_B$ enable interrupt |
| **0** | [2:1], 20, 23 | r | **Reserved** |

**SRCLR**

Clear service request bits of register SRRAW. Write one to clear corresponding bits. Writing zeros has no effect.


Subject to Agreement on the Use of Product Information

## SRCLR

**SCU Service Request Clear** **($0040_H$)** **Reset Value: $0000\ 0000_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TSE_LOW | TSE_HIGH | TSE_DONE | RTC_TIM1 | RTC_TIM0 | RTC_ATIM1 | RTC_ATIM0 | RTC_CTR | 0 | SBYCLKFI | VCLIPI | FLCMPLTI | FLECC2I | PEU0I | PESRAMI | LOCI |
| w | w | w | w | w | w | w | w | r | w | w | w | w | w | w | w |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ORC7I | ORC6I | ORC5I | ORC4I | ORC3I | ORC2I | ORC1I | ORC0I | VDROPI | ACMP2I | ACMP1I | ACMP0I | VDDPI | AI | PI | PRWARN |
| w | w | w | w | w | w | w | w | w | w | w | w | w | w | w | w |

| Field | Bits | Type | Description |
|---|---|---|---|
| **PRWARN** | 0 | w | **WDT pre-warning Interrupt Clear**<br>$0_B$ no effect<br>$1_B$ clear status bit in the raw status register |
| **PI** | 1 | w | **RTC Periodic Interrupt Clear**<br>$0_B$ no effect<br>$1_B$ clear status bit in the raw status register |
| **AI** | 2 | w | **RTC Alarm Interrupt Clear**<br>$0_B$ no effect<br>$1_B$ clear status bit in the raw status register |
| **VDDPI** | 3 | w | **VDDP pre-warning Interrupt Clear**<br>$0_B$ no effect<br>$1_B$ clear status bit in the raw status register |
| **ACMP0I** | 4 | w | **Analog Comparator 0 Interrupt Clear**<br>$0_B$ no effect<br>$1_B$ clear status bit in the raw status register |
| **ACMP1I** | 5 | w | **Analog Comparator 1 Interrupt Clear**<br>$0_B$ no effect<br>$1_B$ clear status bit in the raw status register |
| **ACMP2I** | 6 | w | **Analog Comparator 2 Interrupt Clear**<br>$0_B$ no effect<br>$1_B$ clear status bit in the raw status register |
| **VDROPI** | 7 | w | **VDROP Interrupt Clear**<br>$0_B$ no effect<br>$1_B$ clear status bit in the raw status register |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **ORCxI (x=0-7)** | x+8 | w | **Out of Range Comparator X Interrupt Clear**<br>$0_B$ no effect<br>$1_B$ clear status bit in the raw status register |
| **LOCI** | 16 | w | **Loss of Clock Interrupt Clear**<br>$0_B$ no effect<br>$1_B$ clear status bit in the raw status register |
| **PESRAMI** | 17 | w | **16kbytes SRAM Parity Error Interrupt Clear**<br>$0_B$ no effect<br>$1_B$ clear status bit in the raw status register |
| **PEU0I** | 18 | w | **USIC0 SRAM Parity Error Interrupt Clear**<br>$0_B$ no effect<br>$1_B$ clear status bit in the raw status register |
| **FLECC2I** | 19 | w | **Flash Double Bit ECC Interrupt Clear**<br>$0_B$ no effect<br>$1_B$ clear status bit in the raw status register |
| **FLCMPLTI** | 20 | w | **Flash Operation Complete Interrupt Clear**<br>$0_B$ no effect<br>$1_B$ clear status bit in the raw status register |
| **VCLIPI** | 21 | w | **VCLIP Interrupt Clear**<br>$0_B$ no effect<br>$1_B$ clear status bit in the raw status register |
| **SBYCLKFI** | 22 | w | **Standby Clock Failure Interrupt Clear**<br>$0_B$ no effect<br>$1_B$ clear status bit in the raw status register |
| **RTC_CTR** | 24 | w | **RTC CTR Mirror Register Update Clear**<br>$0_B$ no effect<br>$1_B$ clear status bit in the raw status register |
| **RTC_ATIM0** | 25 | w | **RTC ATIM0 Mirror Register Update Clear**<br>$0_B$ no effect<br>$1_B$ clear status bit in the raw status register |
| **RTC_ATIM1** | 26 | w | **RTC ATIM1 Mirror Register Update Clear**<br>$0_B$ no effect<br>$1_B$ clear status bit in the raw status register |
| **RTC_TIM0** | 27 | w | **RTC TIM0 Mirror Register Update Clear**<br>$0_B$ no effect<br>$1_B$ clear status bit in the raw status register |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **RTC_TIM1** | 28 | w | **RTC TIM1 Mirror Register Update Clear**<br>$0_B$ no effect<br>$1_B$ clear status bit in the raw status register |
| **TSE_DONE** | 29 | w | **TSE Measurement Done Interrupt Clear**<br>$0_B$ no effect<br>$1_B$ clear status bit in the raw status register |
| **TSE_HIGH** | 30 | w | **TSE Compare High Temperature Interrupt Clear**<br>$0_B$ no effect<br>$1_B$ clear status bit in the raw status register |
| **TSE_LOW** | 31 | w | **TSE Compare Low Temperature Interrupt Clear**<br>$0_B$ no effect<br>$1_B$ clear status bit in the raw status register |
| **0** | 23 | r | **Reserved** |

**SRSET**

Set service request fits of register SRRAW. Write one to set corresponding bits. Writing zeros has no effect.

**SRSET**
**SCU Service Request Set** **($0044_H$)** **Reset Value: $0000\ 0000_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TSE_LOW | TSE_HIGH | TSE_DONE | RTC_TIM1 | RTC_TIM0 | RTC_ATIM1 | RTC_ATIM0 | RTC_CTR | 0 | SBYCLKFI | VCLIPI | FLCMPLTI | FLECC2I | PEU0I | PESRAMI | LOCI |
| w | w | w | w | w | w | w | w | r | w | w | w | w | w | w | w |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ORC7I | ORC6I | ORC5I | ORC4I | ORC3I | ORC2I | ORC1I | ORC0I | VDROPI | ACMP2I | ACMP1I | ACMP0I | VDDPI | AI | PI | PRWARN |
| w | w | w | w | w | w | w | w | w | w | w | w | w | w | w | w |

| Field | Bits | Type | Description |
|---|---|---|---|
| **PRWARN** | 0 | w | **WDT pre-warning Interrupt Set**<br>$0_B$ no effect<br>$1_B$ set status bit in the raw status register |

| Field | Bits | Type | Description |
|---|---|---|---|
| **PI** | 1 | w | **RTC Periodic Interrupt Set**<br>$0_B$ no effect<br>$1_B$ set status bit in the raw status register |
| **AI** | 2 | w | **RTC Alarm Interrupt Set**<br>$0_B$ no effect<br>$1_B$ set status bit in the raw status register |
| **VDDPI** | 3 | w | **VDDP pre-warning Interrupt Set**<br>$0_B$ no effect<br>$1_B$ set status bit in the raw status register |
| **ACMP0I** | 4 | w | **Analog Comparator 0 Interrupt Set**<br>$0_B$ no effect<br>$1_B$ set status bit in the raw status register |
| **ACMP1I** | 5 | w | **Analog Comparator 1 Interrupt Set**<br>$0_B$ no effect<br>$1_B$ set status bit in the raw status register |
| **ACMP2I** | 6 | w | **Analog Comparator 2 Interrupt Set**<br>$0_B$ no effect<br>$1_B$ set status bit in the raw status register |
| **VDROPI** | 7 | w | **VDROP Interrupt Set**<br>$0_B$ no effect<br>$1_B$ set status bit in the raw status register |
| **ORCxI (x=0-7)** | x+8 | w | **Out of Range Comparator X Interrupt Set**<br>$0_B$ no effect<br>$1_B$ set status bit in the raw status register |
| **LOCI** | 16 | w | **Loss of Clock Interrupt Set**<br>$0_B$ no effect<br>$1_B$ set status bit in the raw status register |
| **PESRAMI** | 17 | w | **16kbytes SRAM Parity Error Interrupt Set**<br>$0_B$ no effect<br>$1_B$ set status bit in the raw status register |
| **PEU0I** | 18 | w | **USIC0 SRAM Parity Error Interrupt Set**<br>$0_B$ no effect<br>$1_B$ set status bit in the raw status register |
| **FLECC2I** | 19 | w | **Flash Double Bit ECC Interrupt Set**<br>$0_B$ no effect<br>$1_B$ set status bit in the raw status register |


Subject to Agreement on the Use of Product Information

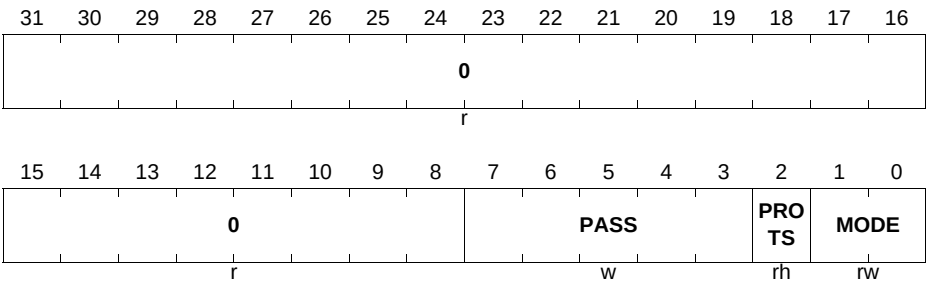
| Field | Bits | Type | Description |
|---|---|---|---|
| **FLCMPLTI** | 20 | w | **Flash Operation Complete Interrupt Set**<br>$0_B$ no effect<br>$1_B$ set status bit in the raw status register |
| **VCLIPI** | 21 | w | **VCLIP Interrupt Set**<br>$0_B$ no effect<br>$1_B$ set status bit in the raw status register |
| **SBYCLKFI** | 22 | w | **Standby Clock Failure Interrupt Set**<br>$0_B$ no effect<br>$1_B$ set status bit in the raw status register |
| **RTC_CTR** | 24 | w | **RTC CTR Mirror Register Update Set**<br>$0_B$ no effect<br>$1_B$ set status bit in the raw status register |
| **RTC_ATIM0** | 25 | w | **RTC ATIM0 Mirror Register Update Set**<br>$0_B$ no effect<br>$1_B$ set status bit in the raw status register |
| **RTC_ATIM1** | 26 | w | **RTC ATIM1 Mirror Register Update Set**<br>$0_B$ no effect<br>$1_B$ set status bit in the raw status register |
| **RTC_TIM0** | 27 | w | **RTC TIM0 Mirror Register Update Set**<br>$0_B$ no effect<br>$1_B$ set status bit in the raw status register |
| **RTC_TIM1** | 28 | w | **RTC TIM1 Mirror Register Update Set**<br>$0_B$ no effect<br>$1_B$ set status bit in the raw status register |
| **TSE_DONE** | 29 | w | **TSE Measurement Done Interrupt Set**<br>$0_B$ no effect<br>$1_B$ set status bit in the raw status register |
| **TSE_HIGH** | 30 | w | **TSE Compare High Temperature Interrupt Set**<br>$0_B$ no effect<br>$1_B$ set status bit in the raw status register |
| **TSE_LOW** | 31 | w | **TSE Compare Low Temperature Interrupt Set**<br>$0_B$ no effect<br>$1_B$ set status bit in the raw status register |
| **0** | 23 | r | **Reserved** |

### PASSWD

The PASSWD register is used to control the bit protection scheme.


Subject to Agreement on the Use of Product Information

**PASSWD**
**Password Register** **($0024_H$)** **Reset Value: 0000 0007$_H$**

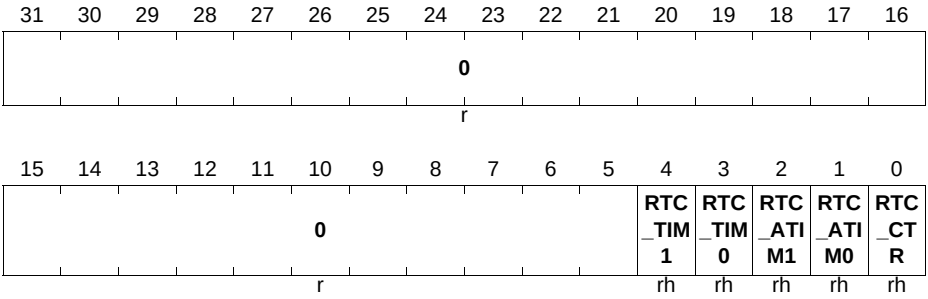



| Field | Bits | Type | Description |
|---|---|---|---|
| **MODE** | [1:0] | rw | **Bit Protection Scheme Control Bits**<br>$00_B$ Scheme disabled - direct access to the protected bits is allowed.<br>$11_B$ Scheme enabled - the bit field PASS has to be written with the passwords to open and close the access to the protected bits. (Default)<br>Others: Scheme enabled, similar to the setting for MODE = $11_B$.<br>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**<br>This bit shows the status of the protection.<br>$0_B$ Software is able to write to all protected bits.<br>$1_B$ Software is unable to write to any of the protected bits. |
| **PASS** | [7:3] | w | **Password Bits**<br>This bit protection scheme only recognizes the following three passwords:<br>$11000_B$Enables writing of the bit field MODE.<br>$10011_B$Opens access to writing of all protected bits.<br>$10101_B$Closes access to writing of all protected bits. |
| **0** | [31:8] | r | **Reserved** |

### MIRRSTS

Mirror status register for control of communication between SCU and RTC.

**MIRRSTS**
**Mirror Update Status Register (0048$_H$) Reset Value: 0000 0000$_H$**



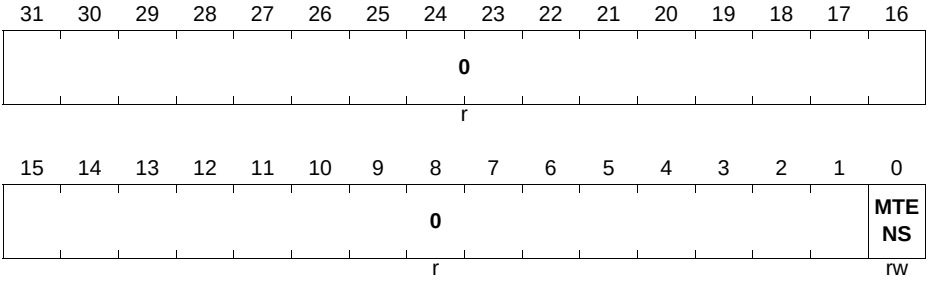

| Field | Bits | Type | Function |
|---|---|---|---|
| **RTC_CTR** | 0 | rh | **RTC CTR Mirror Register Update Status**<br>$0_B$ no update pending<br>$1_B$ update pending |
| **RTC_ATIM0** | 1 | rh | **RTC ATIM0 Mirror Register Update Status**<br>$0_B$ no update pending<br>$1_B$ update pending |
| **RTC_ATIM1** | 2 | rh | **RTC ATIM1 Mirror Register Update Status**<br>$0_B$ no update pending<br>$1_B$ update pending |
| **RTC_TIM0** | 3 | rh | **RTC TIM0 Mirror Register Update Status**<br>$0_B$ no update pending<br>$1_B$ update pending |
| **RTC_TIM1** | 4 | rh | **RTC TIM1 Mirror Register Update Status**<br>$0_B$ no update pending<br>$1_B$ update pending |
| **0** | [31:14] | r | **Reserved**<br>Read as 0; should be written with 0. |

### PMTSR

This register selects parity test output from a memory instance.

**PMTSR**

**Parity Memory Test Select Register (0054$_H$)** **Reset Value: 0000 0000$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **MTENS** | 0 | rw | **Parity Test Enable Control for 16kbytes SRAM**<br>Controls the test multiplexer for the 16kbytes SRAM.<br>$0_B$ standard operation<br>$1_B$ generate an inverted parity bit during a write operation |
| **0** | 31:1] | r | **Reserved**<br>Should be written with 0. |

# 14 Pseudo Random Number Generator (PRNG)

## 14.1 Overview

The pseudo random bit generator (PRNG) provides random data with fast generation times.

### 14.1.1 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

## 14.2 Description of Operation Modes

### 14.2.1 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


Subject to Agreement on the Use of Product Information

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 – i.e., a key must be loaded and a warm-up phase carried out.

### 14.2.2 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 $0_B$ 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.*

### 14.2.3 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 inital 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


Subject to Agreement on the Use of Product Information

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 – i.e., 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.*

## 14.3 Service Request Generation

The PRNG does not generate any service request.

## 14.4 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.

## 14.5 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.

## 14.6 Initialization and System Dependencies

The PRNG is always available after reset.

Refer to **Section 14.2.1** for the initialization steps.

## 14.7 Registers

The interface of the PRNG comprises the registers PRNG_WORD, PRNG_CHK and PRNG_CTRL.

**Table 14-1 Registers Address Space**

| Module | Base Address | End Address | Note |
|---|---|---|---|
| PRNG | $4802\ 0000_H$ | $4802\ 000F_H$ | |

**Table 14-2 Registers Overview**

| Register Short Name | Register Long Name | Offset Address | Page Number |
|---|---|---|---|
| **Data Registers** | | | |
| **PRNG_WORD** | PRNG word register | $00_H$ | **Page 14-4** |
| **PRNG_CHK** | PRNG status check register | $04_H$ | **Page 14-5** |
| **Control Registers** | | | |
| **PRNG_CTRL** | PRNG control register | $0C_H$ | **Page 14-6** |

The register is addressed wordwise.

## 14.7.1 Data Registers

### Pseudo RNG Word Register

**PRNG_WORD**
**PRNG Word Register** **($00_H$)** **Reset Value: $0000_H$**

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

| Field | Bits | Type | Description |
|---|---|---|---|
| **RDATA** | 15:0 | rw | **Random Data**<br>Random bit block or key to load.<br>In the streaming mode the range of valid random bits is defined by the settings given by PRNG_CTRL.RDSB and the value of the flag PRNG_CHK.RDV.<br>In the key loading mode the seed value (key) is written via this register. In this case, the key is written in units of 16 bits. |


Subject to Agreement on the Use of Product Information

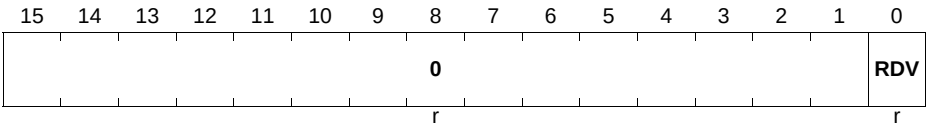
### Additional Information

### Notes

1. *Reading PRNG_WORD while the PRNG is running in key loading mode (configured by setting PRNG_CTRL.KLD) will return the last value written to the register, whereas write accesses to the register while the PRNG is running in streaming mode (PRNG_CTRL.KLD = '0') will be ignored.*
2. *Write access to SFR PRNG_WORD:*
   *It is strongly recommended to wait until the SFR bit PRNG_CHK.RDV indicates that the register PRNG_WORD is ready to receive (new) data (which will be used to seed the PRNG.)*
3. *Read access to SFR PRNG_WORD:*
   *It is strongly recommended to check the SFR bit PRNG_CHK.RDV before reading SFR PRNG_WORD.*

### Pseudo RNG Status Check Register

**PRNG_CHK**
**PRNG Status Check Register** **($04_H$)** **Reset Value: $0000_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **RDV** | 0 | r | **Random Data / Key Valid Flag**<br>$0_B$ **INV**, New random data block is not yet ready to be read. In **"Key Loading Mode" on Page 14-1**) this flag is set to $0_B$ while loading is in progress.<br>$1_B$ **VAL**, Random data block is valid. In key loading mode this value indicates that the next partial key word can be written to **PRNG_WORD**. |
| **0** | 15:1 | r | **Reserved** |

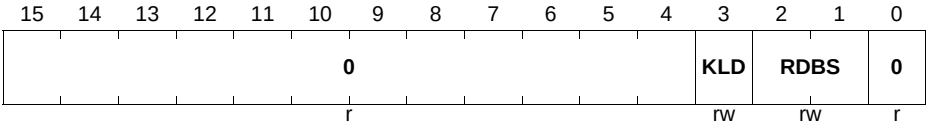
### Additional Information

*Note: If a word or byte is read from **PRNG_WORD** although the data is not ready (**PRNG_CHK**.RDV = $0_B$), the system stalls until the data becomes ready.*

## 14.7.2 Control Registers

### Pseudo RNG Control Register

**PRNG_CTRL**
**PRNG Control Register** **($0C_H$)** **Reset Value:** $0000_H$




| Field | Bits | Type | Description |
|---|---|---|---|
| **KLD** | 3 | rw | **Key Load Operation Mode**<br>$0_B$ **STRM**, Streaming mode (default)<br>$1_B$ **KLD**, Key loading mode |
| **RDBS** | 2:1 | rw | **Random Data Block Size**<br>Set random data block size for read access (16 bits always used for key loading)<br>$00_B$ **RES**, Reset state (no random data block size defined)[1], value of **PRNG_WORD** is undefined.<br>$01_B$ **BYTE**, 8 bits in PRNG_WORD.RDATA[7:0]<br>$10_B$ **WORD**, 16 bits in PRNG_WORD.RDATA[15:0]<br>$11_B$ **RFU**, Reserved for future use, value of **PRNG_WORD** is undefined. |
| **0** | 0 | r | **Reserved** |
| **0** | 15:4 | r | **Reserved** |

1) Once the reset value for RDBS ($00_B$) is set, the PRNG must be fully initialized before the functionality can be used. Initialization requires key loading, followed by a warm-up phase.

### Additional Information

*Note: It is recommended not to change the **PRNG_CTRL**.KLD flag during key load. Otherwise the distribution of the bits of the PRNG will not be equal.*

# Communication Peripherals


Subject to Agreement on the Use of Product Information

# 15 Universal Serial Interface Channel (USIC)

The **U**niversal **S**erial **I**nterface **C**hannel module (USIC) is a flexible interface module covering several serial communication protocols. A USIC module contains two independent communication channels named USICx_CH0 and USICx_CH1, with x being the number of the USIC module (e.g. channel 0 of USIC module 0 is referenced as USIC0_CH0). The user can program during run-time which protocol will be handled by each communication channel and which pins are used.

**References**

The following documents are referenced for further information

[7] IIC Bus Specification (Philips Semiconductors v2.1)

[8] IIS Bus Specification (Philips Semiconductors June 5 1996 revision)

**Table 15-1 Abbreviations**

| | |
|---|---|
| CTQ | Time Quanta Counter |
| DSU | Data Shift Unit |
| $f_{PERIPH}$ | USIC module clock frequency |
| $f_{PIN}$ | Input frequency to baud rate generator |
| MCLK | Master Clock<br>*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 |

## 15.1 Overview

This section gives an overview about the feature set of the USIC.

### 15.1.1 Features

Each USIC channel can be individually configured to match the application needs, e.g. the protocol can be selected or changed during run time without the need for a reset. The following protocols are supported:

- **UART** (ASC, asynchronous serial channel)
  – Module capability: receiver/transmitter with max. baud rate $f_{PERIPH}$ / 4


Subject to Agreement on the Use of Product Information

  - Wide baud rate range down to single-digit baud rates
  - Number of data bits per data frame: 1 to 63
  - MSB or LSB first
- **LIN** Support by hardware (Local Interconnect Network)
  - Data transfers based on ASC protocol
  - Baud rate detection possible by built-in capture event of baud rate generator
  - Checksum generation under software control for higher flexibility
- **SSC/SPI** (synchronous serial channel with or without slave select lines)
  - Standard, Dual and Quad SPI format supported
  - Module capability: maximum baud rate $f_{PERIPH}$ / 2, limited by loop delay
  - Number of data bits per data frame 1 to 63, more with explicit stop condition
  - Parity bit generation supported
  - MSB or LSB first
- **IIC** (Inter-IC Bus)
  - Application baud rate 100 kbit/s to 400 kbit/s
  - 7-bit and 10-bit addressing supported
  - Full master and slave device capability
- **IIS** (infotainment audio bus)
  - Module capability: maximum baud rate $f_{PERIPH}$ / 2

*Note: The real baud rates that can be achieved in a real application depend on the operating frequency of the device, timing parameters as described in the Data Sheet, signal delays on the PCB and timings of the peer device.*

In addition to the flexible choice of the communication protocol, the USIC structure has been designed to reduce the system load (CPU load) allowing efficient data handling. The following aspects have been considered:

- **Data buffer capability**
  The standard buffer capability includes a double word buffer for receive data and a single word buffer for transmit data. This allows longer CPU reaction times (e.g. interrupt latency).
- **Additional FIFO buffer capability**
  In addition to the standard buffer capability, the received data and the data to be transmitted can be buffered in a FIFO buffer structure. The size of the receive and the transmit FIFO buffer can be programmed independently. Depending on the application needs, a total buffer capability of 64 data words can be assigned to the receive and transmit FIFO buffers of a USIC module (the two channels of the USIC module share the 64 data word buffer).
  In addition to the FIFO buffer, a bypass mechanism allows the introduction of high-priority data without flushing the FIFO buffer.
- **Transmit control information**
  For each data word to be transmitted, a 5-bit transmit control information has been added to automatically control some transmission parameters, such as word length, frame length, or the slave select control for the SPI protocol. The transmit control


Subject to Agreement on the Use of Product Information

information is generated automatically by analyzing the address where the user software has written the data word to be transmitted (32 input locations = $2^5$ = 5 bit transmit control information).
This feature allows individual handling of each data word, e.g. the transmit control information associated to the data words stored in a transmit FIFO can automatically modify the slave select outputs to select different communication targets (slave devices) without CPU load. Alternatively, it can be used to control the frame length.

- **Flexible frame length control**
  The number of bits to be transferred within a data frame is independent of the data word length and can be handled in two different ways. The first option allows automatic generation of frames up to 63 bits with a known length. The second option supports longer frames (even unlimited length) or frames with a dynamically controlled length.
- **Interrupt capability**
  The events of each USIC channel can be individually routed to one of 6 service request outputs SR[5:0] available for each USIC module, depending on the application needs. Furthermore, specific start and end of frame indications are supported in addition to protocol-specific events.
- **Flexible interface routing**
  Each USIC channel offers the choice between several possible input and output pins connections for the communications signals. This allows a flexible assignment of USIC signals to pins that can be changed without resetting the device.
- **Input conditioning**
  Each input signal is handled by a programmable input conditioning stage with programmable filtering and synchronization capability.
- **Baud rate generation**
  Each USIC channel contains its own baud rate generator. The baud rate generation can be based either on the internal module clock or on an external frequency input. This structure allows data transfers with a frequency that can not be generated internally, e.g. to synchronize several communication partners.
- **Transfer trigger capability**
  In master mode, data transfers can be triggered by events generated outside the USIC module, e.g. by an input pin or a timer unit (transmit data validation). This feature allows time base related data transmission.
- **Debugger support**
  The USIC offers specific addresses to read out received data without interaction with the FIFO buffer mechanism. This feature allows debugger accesses without the risk of a corrupted receive data sequence.

To reach a desired baud rate, two criteria have to be respected, the module capability and the application environment. The module capability is defined with respect to the module's input clock frequency, being the base for the module operation. Although the module's capability being much higher (depending on the module clock and the number

of module clock cycles needed to represent a data bit), the reachable baud rate is generally limited by the application environment. In most cases, the application environment limits the maximum reachable baud rate due to driver delays, signal propagation times, or due to EMI reasons.

*Note: Depending on the selected additional functions (such as digital filters, input synchronization stages, sample point adjustment, data structure, etc.), the maximum reachable baud rate can be limited. Please also take care about additional delays, such as (internal or external) propagation delays and driver delays (e.g. for collision detection in ASC mode, for IIC, etc.).*


Subject to Agreement on the Use of Product Information

A block diagram of the USIC module/channel structure is shown in **Figure 15-1**.

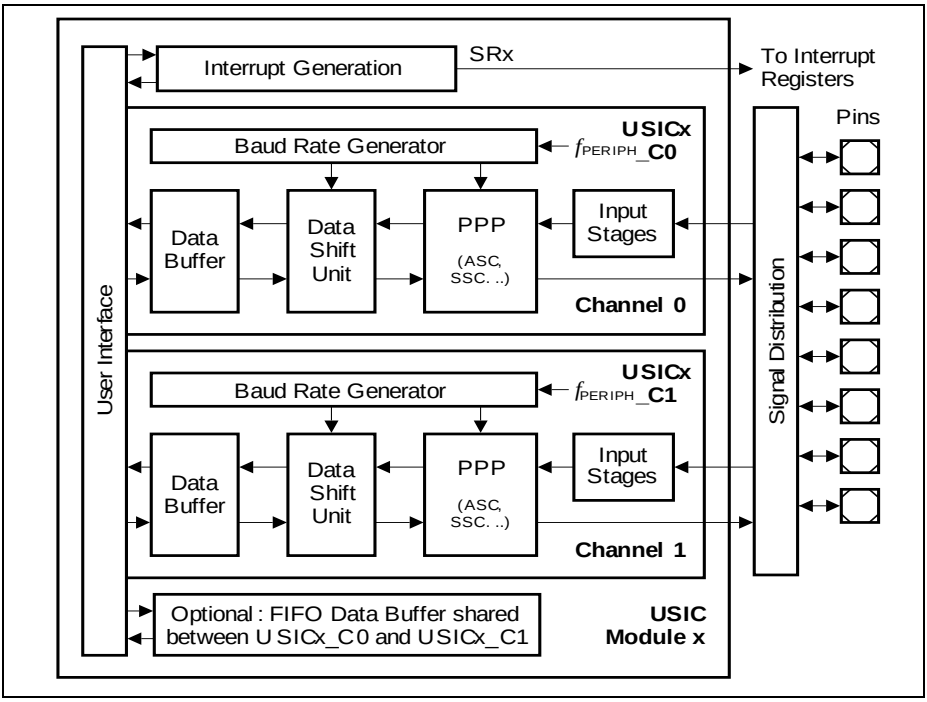



**Figure 15-1 USIC Module/Channel Structure**

## 15.2 Operating the USIC

This section describes how to operate the USIC communication channel.

### 15.2.1 USIC Structure Overview

This section introduces the USIC structure.

#### 15.2.1.1 Channel Structure

The USIC module contains two independent communication channels, with a structure as shown in **Figure 15-1**.

The data shift unit and the data buffering of each channel support full-duplex data transfers. The protocol-specific actions are handled by the protocol pre-processors (PPP). In order to simplify data handling, an additional FIFO data buffer is optionally available for each USIC module to store transmit and receive data for each channel.

Due to the independent channel control and baud rate generation, the communication protocol, baud rate and the data format can be independently programmed for each communication channel.

#### 15.2.1.2 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 15-2**.

The inputs marked as “optional” are not needed for the standard function of a protocol and may be used for enhancements. The descriptions of protocol-specific items are given in the related protocol chapters. For the external frequency input, please refer to the baud rate generator section, and for the transmit data validation, to the data handling section.


Subject to Agreement on the Use of Product Information

**Table 15-2 Input Signals for Different Protocols**

| Selected Protocol | Shift Data Input(s) (handled by DX0, DX3, DX4 and DX5)[1)] | Shift Clock Input (handled by DX1) | Shift Control Input (handled by DX2) |
|---|---|---|---|
| **ASC, LIN** | RXD | optional: external frequency input or TXD collision detection | optional: transmit data validation |
| **Standard SSC, SPI (Master)** | DIN0 (MRST, MISO) | optional: external frequency input or delay compensation | optional: transmit data validation or delay compensation |
| **Standard SSC, SPI (Slave)** | DIN0 (MTSR, MOSI) | SCLKIN | SELIN |
| **Dual-SSC, SPI (Master)** | DIN[1:0] (MRST[1:0], MISO[1:0]) | optional: external frequency input or delay compensation | optional: transmit data validation or delay compensation |
| **Dual-SSC, SPI (Slave)** | DIN[1:0] (MTSR[1:0], MOSI[1:0]) | SCLKIN | SELIN |
| **Quad-SSC, SPI (Master)** | DIN[3:0] (MRST[3:0], MISO[3:0]) | optional: external frequency input or delay compensation | optional: transmit data validation or delay compensation |
| **Quad-SSC, SPI (Slave)** | DIN[3:0] (MTSR[3:0], MOSI[3:0]) | SCLKIN | SELIN |
| **IIC** | SDA | SCL | optional: transmit data validation |
| **IIS (Master)** | DIN0 | optional: external frequency input or delay compensation | optional: transmit data validation or delay compensation |
| **IIS (Slave)** | DIN0 | SCLKIN | WAIN |

1) ASC, IIC, IIS and standard SSC protocols use only DX0 as the shift data input.


Subject to Agreement on the Use of Product Information

*Note: To allow a certain flexibility in assigning required USIC input functions to port pins of the device, each input stage can select the desired input location among several possibilities.*
*The available USIC signals and their port locations are listed in the interconnects section, see* ***Page 15-226****.*

### 15.2.1.3 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 15-3**.

The outputs marked as “optional” are not needed for the standard function of a protocol and may be used for enhancements. The descriptions of protocol-specific items are given in the related protocol chapters. The MCLKOUT output signal has a stable frequency relation to the shift clock output (the frequency of MCLKOUT can be higher than for SCLKOUT) for synchronization purposes of a slave device to a master device. If the baud rate generator is not needed for a specific protocol (e.g. in SSC slave mode), the SCLKOUT and MCLKOUT signals can be used as clock outputs with 50% duty cycle with a frequency that can be independent from the communication baud rate.

**Table 15-3 Output Signals for Different Protocols**

| Selected Protocol | Shift Data Output(s) DOUT[3:0][1)] | Shift Clock Output SCLKOUT | Shift Control Outputs SELO[7:0] | Master Clock Output MCLKOUT |
|---|---|---|---|---|
| **ASC, LIN** | TXD | not used | not used | optional: master time base |
| **Standard SSC, SPI (Master)** | DOUT0 (MTSR, MOSI) | master shift clock | slave select, chip select | optional: master time base |
| **Standard SSC, SPI (Slave)** | DOUT0 (MRST, MISO) | optional: independent clock output | not used | optional: independent clock output |
| **Dual-SSC, SPI (Master)** | DOUT[1:0] (MTSR[1:0], MOSI[1:0]) | master shift clock | slave select, chip select | optional: master time base |
| **Dual-SSC, SPI (Slave)** | DOUT[1:0] (MRST[1:0], MISO[1:0]) | optional: independent clock output | not used | optional: independent clock output |


Subject to Agreement on the Use of Product Information

**Table 15-3 Output Signals for Different Protocols** (cont'd)

| Selected Protocol | Shift Data Output(s) DOUT[3:0][1)] | Shift Clock Output SCLKOUT | Shift Control Outputs SELO[7:0] | Master Clock Output MCLKOUT |
|---|---|---|---|---|
| **Quad-SSC, SPI (Master)** | DOUT[3:0] (MTSR[3:0], MOSI[3:0]) | master shift clock | slave select, chip select | optional: master time base |
| **Quad-SSC, SPI (Slave)** | DOUT[3:0] (MRST[3:0], MISO[3:0]) | optional: independent clock output | not used | optional: independent clock output |
| **IIC** | SDA | SCL | not used | optional: master time base |
| **IIS (master)** | DOUT0 | master shift clock | WA | optional: master time base |
| **IIS (slave)** | DOUT0 | optional: independent clock output | not used | optional: independent clock output |

1) ASC, IIC, IIS and standard SSC protocols use only DOUT0 as the shift data output.

*Note: To allow a certain flexibility in assigning required USIC output functions to port pins of the device, most output signals are made available on several port pins. The port control itself defines pin-by-pin which signal is used as output signal for a port pin (see port chapter).*
*The available USIC signals and their port locations are listed in the interconnects section, see* ***Page 15-226****.*

### 15.2.1.4 Baud Rate Generator

Each USIC Channel contains a baud rate generator structured as shown in **Figure 15-2**. It is based on coupled divider stages, providing the frequencies needed for the different protocols. It contains:

- A fractional divider to generate the input frequency $f_{PIN} = f_{FD}$ for baud rate generation based on the internal system frequency $f_{PERIPH}$.
- The DX1 input to generate the input frequency $f_{PIN} = f_{DX1}$ for baud rate generation based on an external signal.
- Two protocol-related counters: the divider mode counter to provide the master clock signal MCLK, the shift clock signal SCLK, and other protocol-related signals; and the capture mode timer for time interval measurement, e.g. baud rate detection.
- A time quanta counter associated to the protocol pre-processor defining protocol-specific timings, such shift control signals or bit timings, based on the input frequency $f_{CTQIN}$.

- The output signals MCLKOUT and SCLKOUT of the protocol-related divider that can be made available on pins. In order to adapt to different applications, some output characteristics of these signals can be configured.
  For device-specific details about availability of USIC signals on pins please refer to the interconnects section.

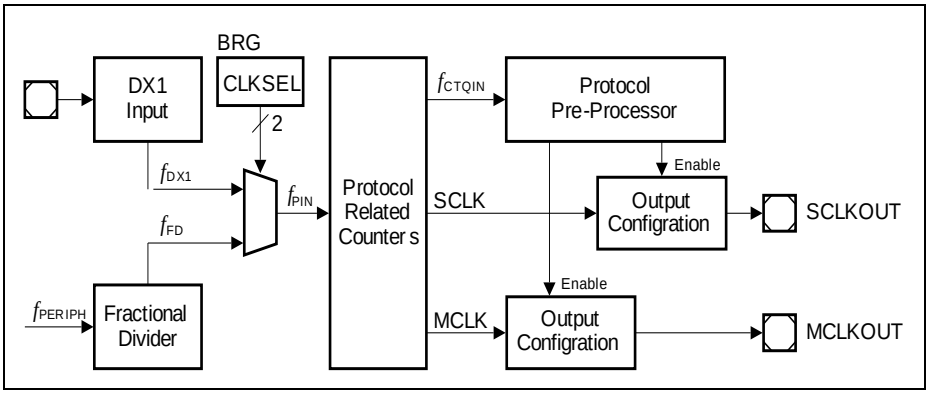



**Figure 15-2 Baud Rate Generator**

### 15.2.1.5 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.

### 15.2.1.6 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.

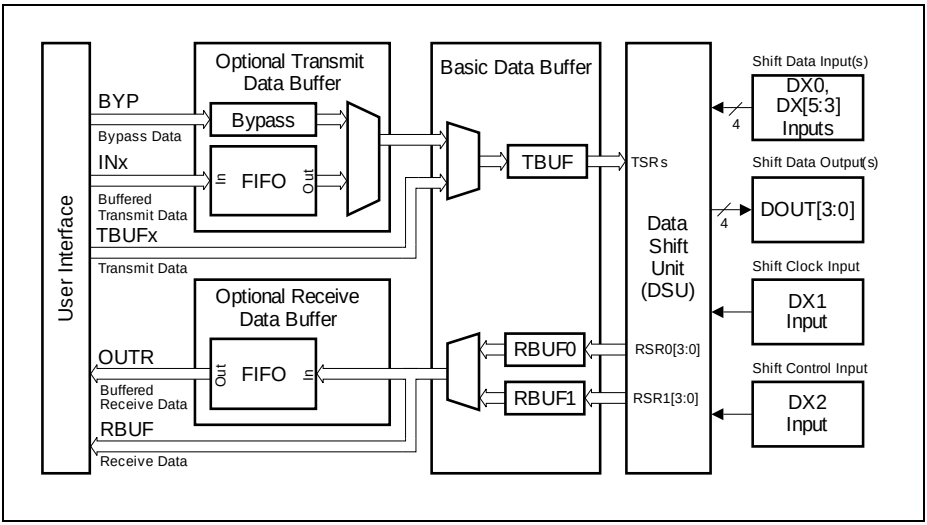



**Figure 15-3 Principle of Data Buffering**

The principle of data handling comprises:

- A transmitter with transmit shift registers (TSR and TSR[3:0]) in the DSU and a transmit data buffer (TBUF). A data validation scheme allows triggering and gating of data transfers by external events under certain conditions.
- A receiver with two alternating sets of receive shift registers (RSR0[3:0] and RSR1[3:0]) in the DSU and a double receive buffer structure (RBUF0, RBUF1). The alternating receive shift registers support the reception of data streams and data frames longer than one data word.
- A user interface to handle data, interrupts, and status and control information.

### Basic Data Buffer Structure

The read access to received data and the write access of data to be transmitted can be handled by a basic data buffer structure.

The received data stored in the receiver buffers RBUF0/RBUF1 can be read directly from these registers. In this case, the user has to take care about the reception sequence to read these registers in the correct order. To simplify the use of the receive buffer structure, register RBUF has been introduced. A read action from this register delivers the data word received first (oldest data) to respect the reception sequence. With a read access from at least the low byte of RBUF, the data is automatically declared to be no longer new and the next received data word becomes visible in RBUF and can be read out next.


Subject to Agreement on the Use of Product Information

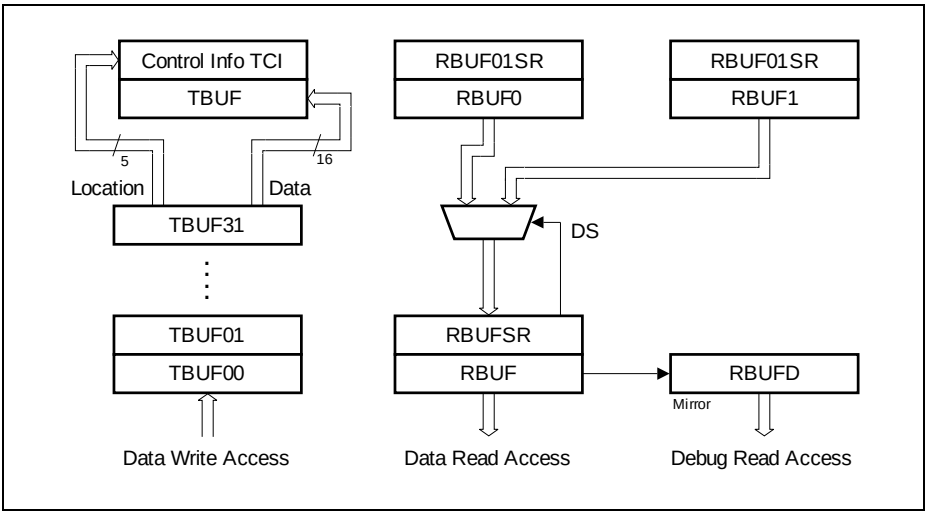



**Figure 15-4 Data Access Structure without additional Data Buffer**

It is recommended to read the received data words by accesses to RBUF and to avoid handling of RBUF0 and RBUF1. The USIC module also supports the use of debug accesses to receive data words. Debugger read accesses should not disturb the receive data sequence and, as a consequence, should not target RBUF. Therefore, register RBUFD has been introduced. It contains the same value as RBUF, but a read access from RBUFD does not change the status of the data (same data can be read several times). In addition to the received data, some additional status information about each received data word is available in the receiver buffer status register RBUF01SR (related to data in RBUF0 and RBUF1) and RBUFSR (related to data in RBUF).

Transmit data can be loaded to TBUF by software by writing to the transmit buffer input locations TBUFx (x = 00-31), consisting of 32 consecutive addresses. The data written to one of these input locations is stored in the transmit buffer TBUF. Additionally, the address of the written location is evaluated and can be used for additional control purposes. This 5-bit wide information (named **T**ransmit **C**ontrol **I**nformation TCI) can be used for different purposes in different protocols.

### FIFO Buffer Structure

To allow easier data setup and handling, an additional data buffering mechanism can be optionally supported. The data buffer is based on the first-in-first-out principle (FIFO) that ensures that the sequence of transferred data words is respected.

If a FIFO buffer structure is used, the data handling scheme (data with associated control information) is similar to the one without FIFO. The additional FIFO buffer can be


Subject to Agreement on the Use of Product Information

independently enabled/disabled for transmission and reception (e.g. if data FIFO buffers are available for a specific USIC channel, it is possible to configure the transmit data path without and the receive data path with FIFO buffering).

The transmit FIFO buffer is addressed by using 32 consecutive address locations for INx instead of TBUFx (x=00-31) regardless of the FIFO depth. The 32 addresses are used to store the 5-bit TCI (together with the written data) associated with each FIFO entry.

The receive FIFO can be read out at two independent addresses, OUTR and OUTDR instead of RBUF and RBUFD. A read from the OUTR location triggers the next data packet to be available for the next read (general FIFO mechanism). In order to allow non-intrusive debugging (without risk of data loss), a second address location (OUTDR) has been introduced. A read at this location delivers the same value as OUTR, but without modifying the FIFO contents.

The transmit FIFO also has the capability to bypass the data stream and to load bypass data to TBUF. This can be used to generate high-priority messages or to send an emergency message if the transmit FIFO runs empty. The transmission control of the FIFO buffer can also use the transfer trigger and transfer gating scheme of the transmission logic for data validation (e.g. to trigger data transfers by events).

*Note: The available size of a FIFO data buffer for a USIC channel depends on the specific device. Please refer to the implementation chapter for details about available FIFO buffer capability.*

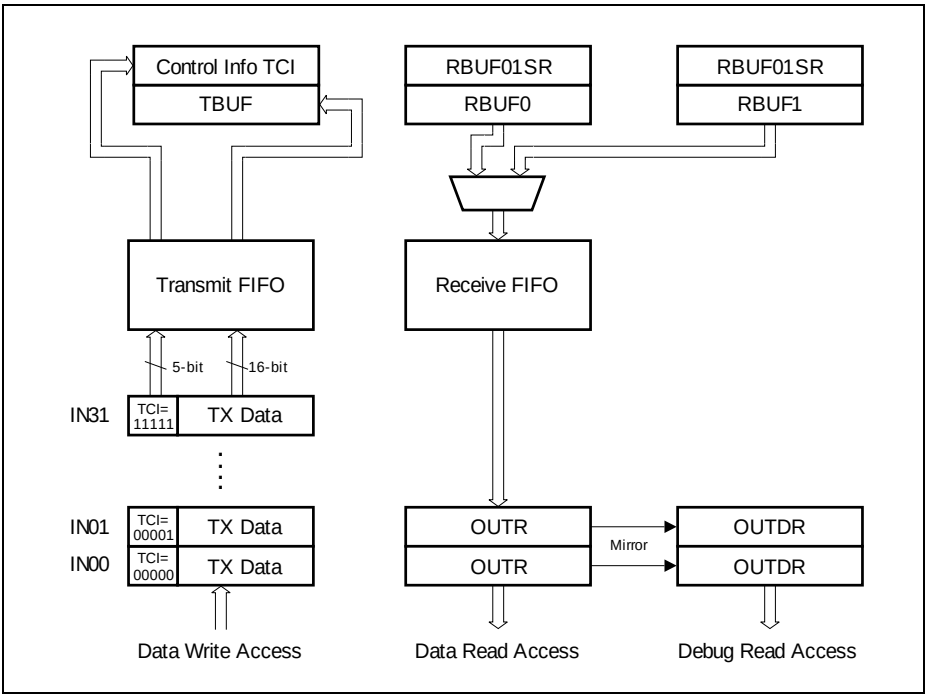



**Figure 15-5 Data Access Structure with FIFO**

## 15.2.2 Operating the USIC Communication Channel

This section describes how to operate a USIC communication channel, including protocol control and status, mode control and interrupt handling. The following aspects have to be taken into account:

- Enable the USIC module for operation and configure the behavior for the different device operation modes (see **Page 15-16**).
- Configure the pinning (refer to description in the corresponding protocol section).
- Configure the data structure (shift direction, word length, frame length, polarity, etc.).
- Configure the data buffer structure of the optional FIFO buffer area. A FIFO buffer can only be enabled if the related bit in register CCFG is set.
- Select a protocol by CCR.MODE. A protocol can only be selected if the related bit in register CCFG is set.

### 15.2.2.1 Protocol Control and Status

The protocol-related control and status information are located in the protocol control register PCR and in the protocol status register PSR. These registers are shared between the available protocols. As a consequence, the meaning of the bit positions in these registers is different within the protocols.

#### Use of PCR Bits

The signification of the bits in register PCR is indicated by the protocol-related alias names for the different protocols.

- PCR for the ASC protocol (see **Page 15-66**)
- PCR for the SSC protocol (see **Page 15-98**)
- PCR for the IIC protocol (see **Page 15-129**)
- PCR for the IIS protocol (see **Page 15-147**)

#### Use of PSR Flags

The signification of the flags in register PSR is indicated by the protocol-related alias names for the different protocols.

- PSR flags for the ASC protocol (see **Page 15-69**)
- PSR flags for the SSC protocol (see **Page 15-102**)
- PSR flags for the IIC protocol (see **Page 15-132**)
- PSR flags for the IIS protocol (see **Page 15-150**)


Subject to Agreement on the Use of Product Information

### 15.2.2.2 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 15-4**.

**Table 15-4 USIC Communication Channel Behavior**

| Kernel Mode | Channel Behavior | KSCFG. NOMCFG |
|---|---|---|
| Run mode 0 | Channel operation as specified, no impact on data transfer | $00_B$ |
| Run mode 1 | | $01_B$ |
| Stop mode 0 | Explicit stop condition as described in the protocol chapters | $10_B$ |
| Stop mode 1 | | $11_B$ |

Generally, bit field KSCFG.NOMCFG should be configured for run mode 0 as default setting for standard operation. If a communication channel should not react to a suspend request (and to continue its operation as in normal mode), bit field KSCFG.SUMCFG has to be configured with the same value as KSCFG.NOMCFG. If the communication channel should show a different behavior and stop operation when a specific stop condition is reached, the code for stop mode 0 or stop mode 1 have to be written to KSCFG.SUMCFG.

The stop conditions are defined for the selected protocol (see mode control description in the protocol section).

*Note: The stop mode selection strongly depends on the application needs and it is very unlikely that different stop modes are required in parallel in the same application. As a result, only one stop mode type (either 0 or 1) should be used in the bit fields in register KSCFG. Do not mix stop mode 0 and stop mode 1 and avoid transitions*


Subject to Agreement on the Use of Product Information

*from stop mode 0 to stop mode 1 (or vice versa) for the same communication channel.*

*Note: In XMC1300, 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.*

### 15.2.2.3 General Channel Events and Interrupts

The general event and interrupt structure is shown in **Figure 15-6**. If a defined condition is met, an event is detected and an event indication flag becomes automatically set. The flag stays set until it is cleared by software. If enabled, an interrupt can be generated if an event is detected. The actual status of the event indication flag has no influence on the interrupt generation. As a consequence, the event indication flag does not need to be cleared to generate further interrupts.

Additionally, the service request output SRx of the USIC channel that becomes activated in case of an event condition can be selected by an interrupt node pointer. This structure allows to assign events to interrupts, e.g. depending on the application, several events can share the same interrupt routine (several events activate the same SRx output) or can be handled individually (only one event activates one SRx output).

The SRx outputs are connected to interrupt control registers to handle the CPU reaction to the service requests. This assignment is described in the implementation section on **Page 15-153**.

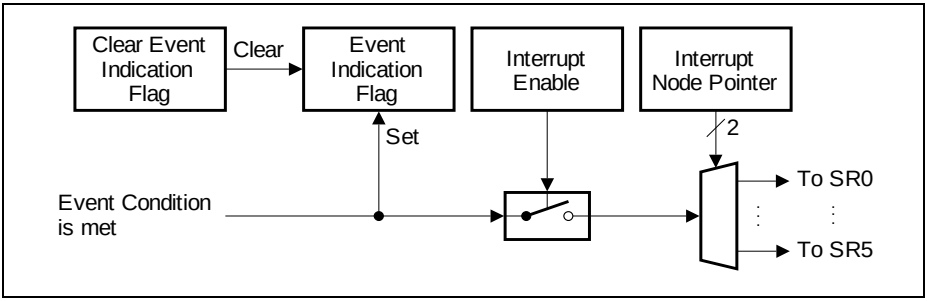



**Figure 15-6 General Event and Interrupt Structure**


Subject to Agreement on the Use of Product Information

### 15.2.2.4 Data Transfer Events and Interrupts

The data transfer events are based on the transmission or reception of a data word. The related indication flags are located in register PSR. All events can be individually enabled for interrupt generation.

- Receive event to indicate that a data word has been received:
  If a new received word becomes available in the receive buffer RBUF0 or RBUF1, either a receive event or an alternative receive event occurs.
  The receive event occurs if bit RBUFSR.PERR = 0. It is indicated by flag PSR.RIF and, if enabled, leads to receive interrupt.
- Receiver start event to indicate that a data word reception has started:
  When the receive clock edge that shifts in the first bit of a new data word is detected and reception is enabled, a receiver start event occurs. It is indicated by flag PSR.RSIF and, if enabled, leads to transmit buffer interrupt.
  In full duplex mode, this event follows half a shift clock cycle after the transmit buffer event and indicates when the shift control settings are internally “frozen” for the current data word reception and a new setting can be programmed.
  In SSC and IIS mode, the transmit data valid flag TCSR.TDV is cleared in single shot mode with the receiver start event.
- Alternative receive event to indicate that a specific data word has been received:
  If a new received word becomes available in the receive buffer RBUF0 or RBUF1, either a receive event or an alternative receive event occurs.
  The alternative receive event occurs if bit RBUFSR.PERR = 1. It is indicated by flag PSR.AIF and, if enabled, leads to alternative receive interrupt.
  Depending on the selected protocol, bit RBUFSR.PERR is set to indicate a parity error in ASC mode, the reception of the first byte of a new frame in IIC mode, and the WA information about right/left channel in IIS mode. In SSC mode, it is used as indication if the received word is the first data word, and is set if first and reset if not.
- Transmit shift event to indicate that a data word has been transmitted:
  A transmit shift event occurs with the last shift clock edge of a data word. It is indicated by flag PSR.TSIF and, if enabled, leads to transmit shift interrupt.
- Transmit buffer event to indicate that a data word transmission has been started:
  When a data word from the transmit buffer TBUF has been loaded to the shift register and a new data word can be written to TBUF, a transmit buffer event occurs. This happens with the transmit clock edge that shifts out the first bit of a new data word and transmission is enabled. It is indicated by flag PSR.TBIF and, if enabled, leads to transmit buffer interrupt.
  This event also indicates when the shift control settings (word length, shift direction, etc.) are internally “frozen” for the current data word transmission.
  In ASC and IIC mode, the transmit data valid flag TCSR.TDV is cleared in single shot mode with the transmit buffer event.
- Data lost event to indicate a loss of the oldest received data word:
  If the data word available in register RBUF (oldest data word from RBUF0 or RBUF1)

has not been read out before it becomes overwritten with new incoming data, this event occurs. It is indicated by flag PSR.DLIF and, if enabled, leads to a protocol interrupt.

**Table 15-5** shows the registers, bits and bit fields indicating the data transfer events and controlling the interrupts of a USIC channel.

**Table 15-5 Data Transfer Events and Interrupt Handling**

| Event | Indication Flag | Indication cleared by | Interrupt enabled by | SRx Output selected by |
|---|---|---|---|---|
| Standard receive event | PSR.RIF | PSCR.CRIF | CCR.RIEN | INPR.RINP |
| Receive start event | PSR.RSIF | PSCR.CRSIF | CCR.RSIEN | INPR.TBINP |
| Alternative receive event | PSR.AIF | PSCR.CAIF | CCR.AIEN | INPR.AINP |
| Transmit shift event | PSR.TSIF | PSCR.CTSIF | CCR.TSIEN | INPR.TSINP |
| Transmit buffer event | PSR.TBIF | PSCR.CTBIF | CCR.TBIEN | INPR.TBINP |
| Data lost event | PSR.DLIF | PSCR.CDLIF | CCR.DLIEN | INPR.PINP |

**Figure 15-7** shows the two transmit events and interrupts.

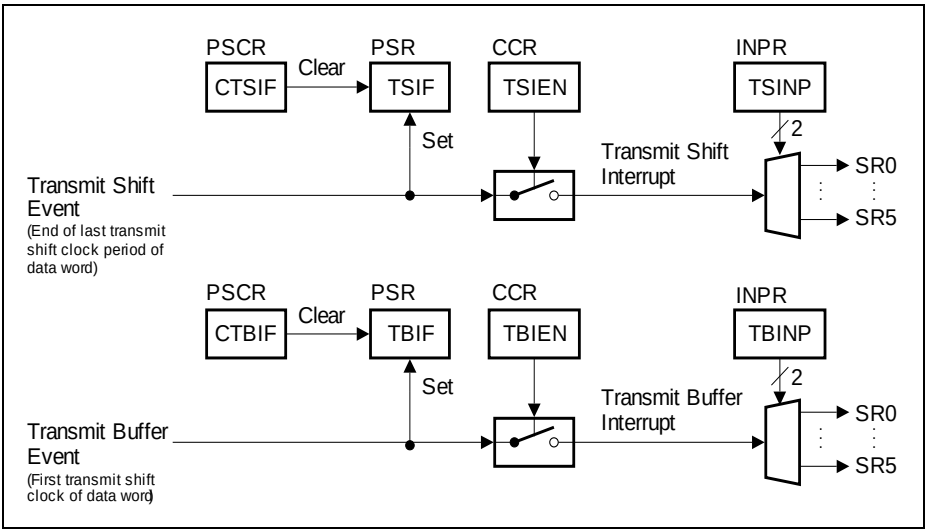



**Figure 15-7 Transmit Events and Interrupts**


Subject to Agreement on the Use of Product Information

**Figure 15-8** shows the receive events and interrupts.

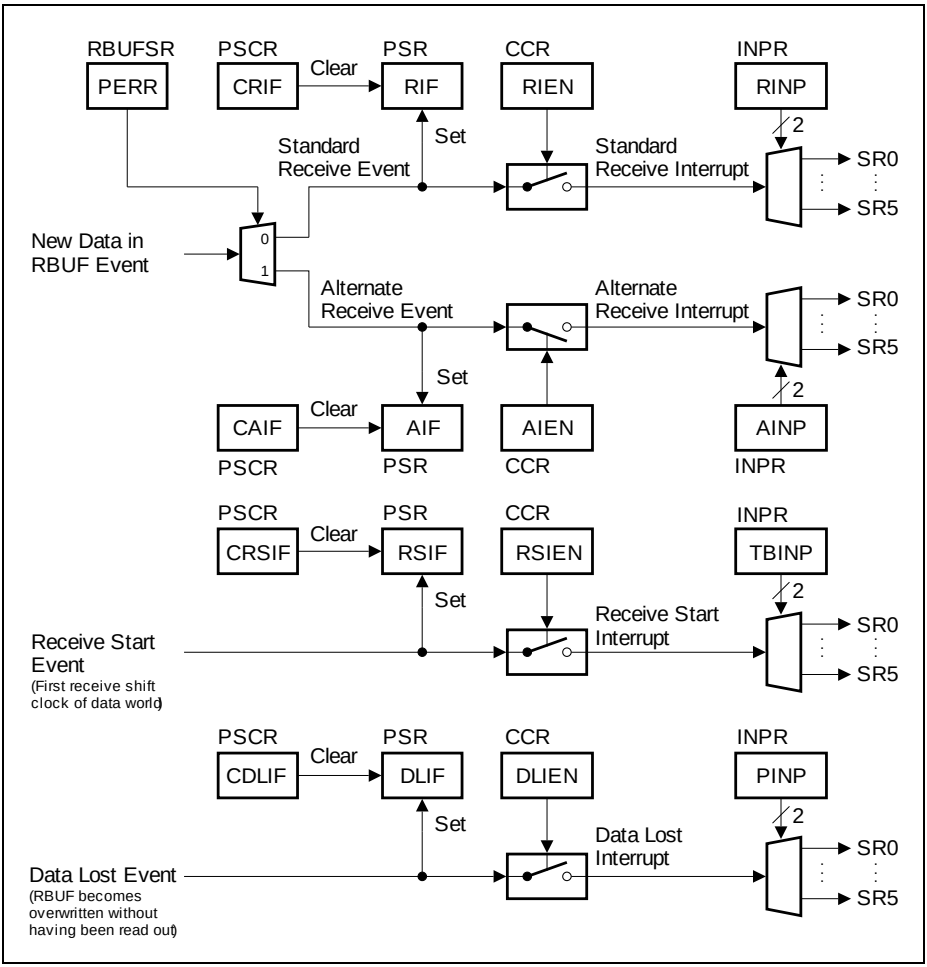



**Figure 15-8 Receive Events and Interrupts**

### 15.2.2.5 Baud Rate Generator Event and Interrupt

The baud rate generator event is based on the capture mode timer reaching its maximum value. It is indicated by flag PSR.BRGIF and, if enabled, leads to a protocol interrupt.

Table 15-6 shows the registers, bits and bit fields indicating the baud rate generator event and controlling the interrupt of a USIC channel.

**Table 15-6 Baud Rate Generator Event and Interrupt Handling**

| Event | Indication Flag | Indication cleared by | Interrupt enabled by | SRx Output selected by |
|---|---|---|---|---|
| Baud rate generator event | PSR. BRGIF | PSCR. CBRGIF | CCR. BRGIEN | INPR.PINP |

Figure 15-9 shows the baud rate generator event and interrupt.

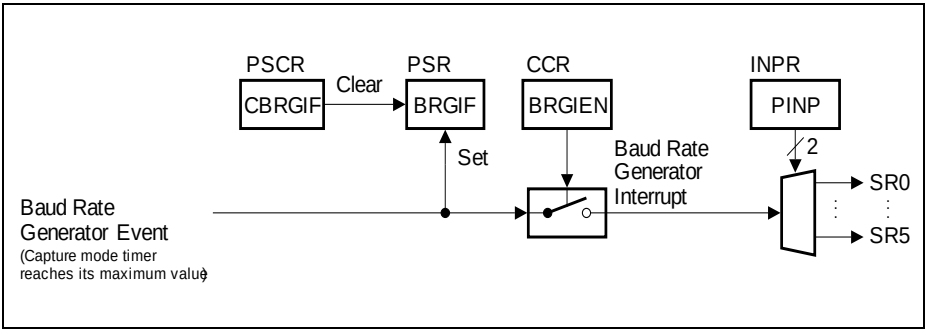



**Figure 15-9 Baud Rate Generator Event and Interrupt**


Subject to Agreement on the Use of Product Information

#### 15.2.2.6 Protocol-specific Events and Interrupts

These events are related to protocol-specific actions that are described in the corresponding protocol chapters. The related indication flags are located in register PSR. All events can be individually enabled for the generation of the common protocol interrupt.

- Protocol-specific events in ASC mode:
  Synchronization break, data collision on the transmit line, receiver noise, format error in stop bits, receiver frame finished, transmitter frame finished
- Protocol-specific events in SSC mode:
  MSLS event (start-end of frame in master mode), DX2T event (start/end of frame in slave mode), both based on slave select signals, parity error
- Protocol-specific events in IIC mode:
  Wrong transmit code (error in frame sequence), start condition received, repeated start condition received, stop condition received, non-acknowledge received, arbitration lost, slave read request, other general errors
- Protocol-specific events in IIS mode:
  DX2T event (change on WA line), WA falling edge or rising edge detected, WA generation finished

**Table 15-7 Protocol-specific Events and Interrupt Handling**

| Event | Indication Flag | Indication cleared by | Interrupt enabled by | SRx Output selected by |
|---|---|---|---|---|
| Protocol-specific events in ASC mode | PSR.ST[8:2] | PSCR.CST[8:2] | PCR.CTR[7:3]] | INPR.PINP |
| Protocol-specific events in SSC mode | PSR.ST[3:2] | PSCR.CST[3:2] | PCR.CTR[15:14] | INPR.PINP |
| Protocol-specific events in IIC mode | PSR.ST[8:1] | PSCR.CST[8:1] | PCR.CTR[24:18] | INPR.PINP |
| Protocol-specific events in IIS mode | PSR.ST[6:3] | PSCR.CST[6:3] | PCR.CTR[6:4], PCR.CTR[15] | INPR.PINP |

### 15.2.3 Operating the Input Stages

All input stages offer the same feature set. They are used for all protocols, because the signal conditioning can be adapted in a very flexible way and the digital filters can be switched on and off separately.


Subject to Agreement on the Use of Product Information

### 15.2.3.1 General Input Structure

There are generally two types of input stages, one for the data input stages DX0, DX[5:3] and the other for non-data input stages DX[2:1], as shown in **Figure 15-10** and **Figure 15-11**. The difference is that for the data input stages, the input signal can be additionally selected from the port signal HWINn if hardware port control is enabled through CCR.HPCEN bit. All other enable/disable functions and selections are controlled independently for each input stage by bits in the registers DXnCR.

The desired input signal can be selected among the input lines DXnA to DXnG and a permanent 1-level by programming bit field DSEL (for the data input stages, hardware port control must be disabled for DSEL to take effect). Please refer to the interconnects section (**Section 15.12**) for the device-specific input signal assignment. Bit DPOL allows a polarity inversion of the selected input signal to adapt the input signal polarity to the internal polarity of the data shift unit and the protocol state machine. For some protocols, the input signals can be directly forwarded to the data shift unit for the data transfers (DSEN = 0, INSW = 1) without any further signal conditioning. In this case, the data path does not contain any delay due to synchronization or filtering.

In the case of noise on the input signals, there is the possibility to synchronize the input signal (signal DXnS is synchronized to $f_{PERIPH}$) and additionally to enable a digital noise filter in the signal path. The synchronized input signal (and optionally filtered if DFEN = 1) is taken into account by DSEN = 1. Please note that the synchronization leads to a delay in the signal path of 2-3 times the period of $f_{PERIPH}$.

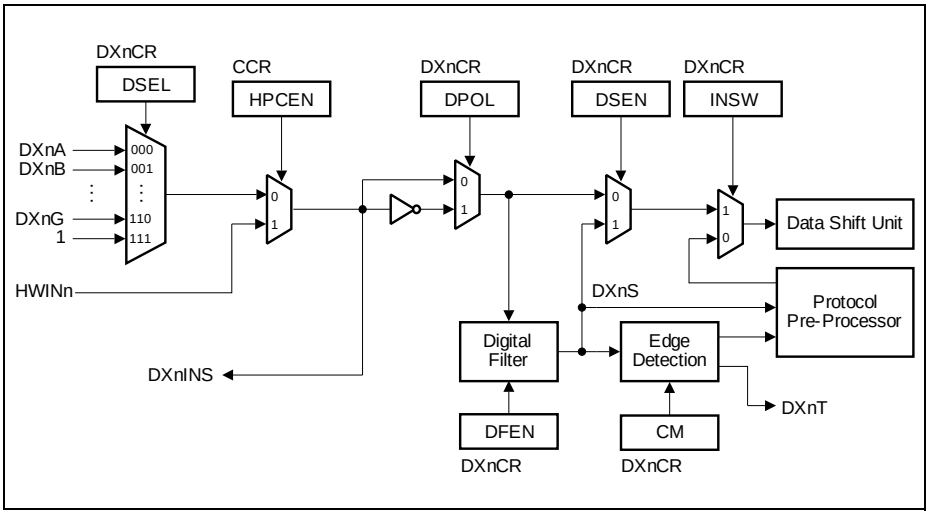



**Figure 15-10 Input Conditioning for DX0 and DX[5:3]**

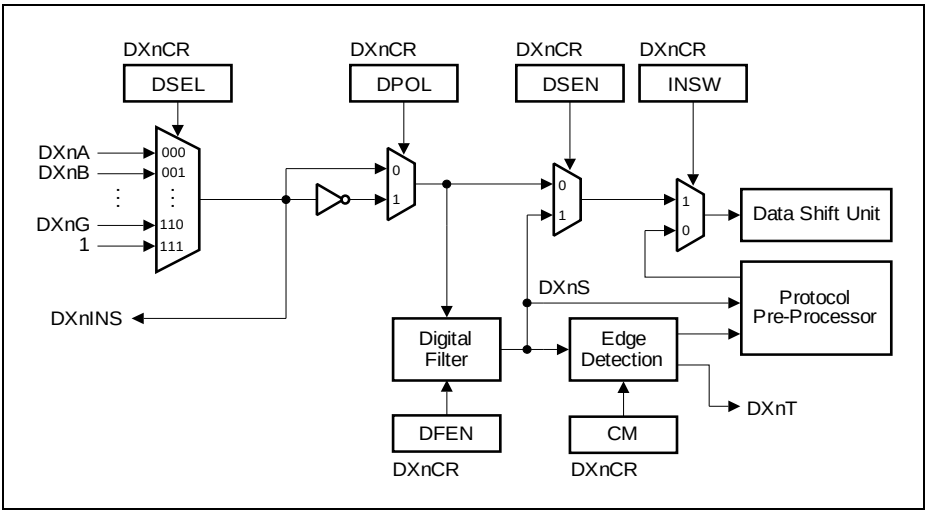



**Figure 15-11 Input Conditioning for DX[2:1]**

If the input signals are handled by a protocol pre-processor, the data shift unit is directly connected to the protocol pre-processor by INSW = 0. The protocol pre-processor is connected to the synchronized input signal DXnS and, depending on the selected protocol, also evaluates the edges.

To support delay compensation in SSC and IIS protocols, the DX1 input stage additionally allows the receive shift clock to be controlled independently from the transmit shift clock through the bit DCEN. When DCEN = 0, the shift clock source is selected by INSW and is the same for both receive and transmit. When DCEN = 1, the receive shift clock is derived from the selected input line as shown in **Figure 15-12**.


Subject to Agreement on the Use of Product Information

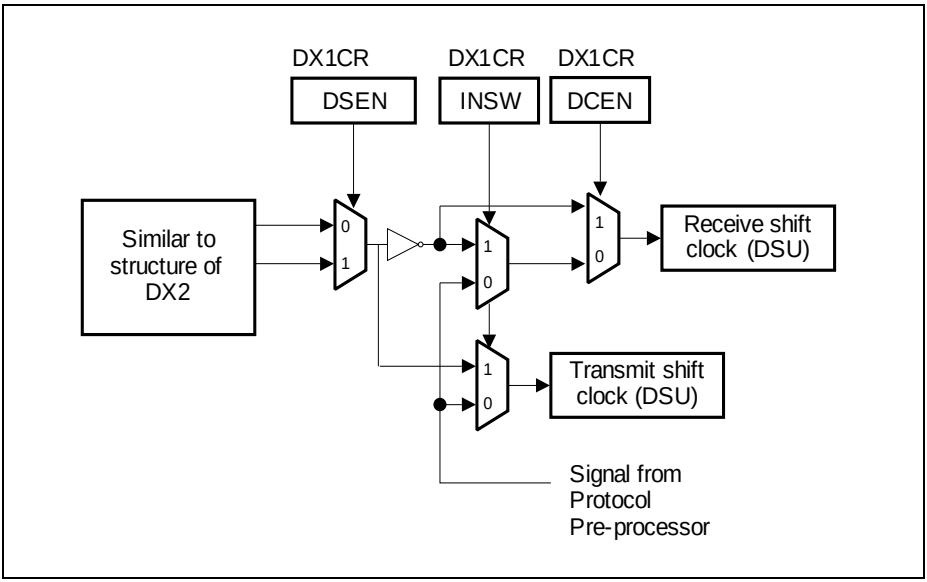



**Figure 15-12 Delay Compensation Enable in DX1**

### 15.2.3.2 Digital Filter

The digital filter can be enabled to reduce noise on the input signals. Before being filtered, the input signal becomes synchronized to $f_{PERIPH}$. If the filter is disabled, signal DXnS corresponds to the synchronized input signal. If the filter is enabled, pulses shorter than one filter sampling period are suppressed in signal DXnS. After an edge of the synchronized input signal, signal DXnS changes to the new value if two consecutive samples of the new value have been detected.

In order to adapt the filter sampling period to different applications, it can be programmed. The first possibility is the system frequency $f_{PERIPH}$. Longer pulses can be suppressed if the fractional divider output frequency $f_{FD}$ is selected. This frequency is programmable in a wide range and can also be used to determine the baud rate of the data transfers.

In addition to the synchronization delay of 2-3 periods of $f_{PERIPH}$, an enabled filter adds a delay of up to two filter sampling periods between the selected input and signal DXnS.

### 15.2.3.3 Edge Detection

The synchronized (and optionally filtered) signal DXnS can be used as input to the data shift unit and is also an input to the selected protocol pre-processor. If the protocol pre-processor does not use the DXnS signal for protocol-specific handling, DXnS can be


Subject to Agreement on the Use of Product Information

used for other tasks, e.g. to control data transmissions in master mode (a data word can be tagged valid for transmission, see chapter about data buffering).

A programmable edge detection indicates that the desired event has occurred by activating the trigger signal DXnT (introducing a delay of one period of $f_{PERIPH}$ before a reaction to this event can take place).

#### 15.2.3.4 Selected Input Monitoring

The selected input signal of each input stage has been made available with the signals DXnINS. These signals can be used in the system to trigger other actions, e.g. to generate interrupts.

#### 15.2.3.5 Loop Back Mode

The USIC transmitter output signals can be connected to the corresponding receiver inputs of the same communication channel in loop back mode. Therefore, the input “G” of the input stages that are needed for the selected protocol have to be selected. In this case, drivers for ASC, SSC, and IIS can be evaluated on-chip without the connections to port pins. Data transferred by the transmitter can be received by the receiver as if it would have been sent by another communication partner.

### 15.2.4 Operating the Baud Rate Generator

The following blocks can be configured to operate the baud rate generator, see also **Figure 15-2**.

#### 15.2.4.1 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:

(15.1)

$$f_{FD} = f_{PERIPH} \times \frac{1}{n} \qquad \text{with } n = 1024 - STEP$$


Subject to Agreement on the Use of Product Information

- Fractional divider mode (FDR.DM = $10_B$):
  In this mode, the output frequency $f_{FD}$ is derived from the input clock $f_{PERIPH}$ by a fractional multiplication of n/1024 for a value of n between 0 and 1023. In general, the fractional divider mode allows to program the average output clock frequency with a finer granularity than in normal divider mode. Please note that in fractional divider mode $f_{FD}$ can have a maximum period jitter of one $f_{PERIPH}$ period. This jitter is not accumulated over several cycles.
  The frequency $f_{FD}$ is generated by an addition of FDR.STEP to FDR.RESULT with $f_{PERIPH}$. The frequency $f_{FD}$ is based on the overflow of the addition result over $3FF_H$.
  The output frequency in fractional divider mode is defined by the equation:

(15.2)

$$f_{FD} = f_{PERIPH} \times \frac{n}{1024} \qquad \text{with n = STEP}$$

The output frequency $f_{FD}$ of the fractional divider is selected for baud rate generation by BRG.CLKSEL = $00_B$.

### 15.2.4.2 External Frequency Input

The baud rate can be generated referring to an external frequency input (instead of to $f_{PERIPH}$) if in the selected protocol the input stage DX1 is not needed (DX1CTR.INSW = 0). In this case, an external frequency input signal at the DX1 input stage can be synchronized and sampled with the system frequency $f_{PERIPH}$. It can be optionally filtered by the digital filter in the input stage. This feature allows data transfers with frequencies that can not be generated by the device itself, e.g. for specific audio frequencies.

If BRG.CLKSEL = $10_B$, the trigger signal DX1T determines $f_{DX1}$. In this mode, either the rising edge, the falling edge, or both edges of the input signal can be used for baud rate generation, depending on the configuration of the DX1T trigger event by bit field DX1CTR.CM. The signal MCLK toggles with each trigger event of DX1T.

If BRG.CLKSEL = $11_B$, the rising edges of the input signal can be used for baud rate generation. The signal MCLK represents the synchronized input signal DX1S.

Both, the high time and the low time of external input signal must each have a length of minimum 2 periods of $f_{PERIPH}$ to be used for baud rate generation.

### 15.2.4.3 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.


Subject to Agreement on the Use of Product Information

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.

$$f_{MCLK} = \frac{f_{PIN}}{2} \tag{15.3}$$

$$f_{SCLK} = \frac{f_{PDIV}}{2} \tag{15.4}$$

In the case that the master clock is used as reference for external devices (e.g. for IIS components) and a fixed phase relation to SCLK and other timing signals is required, it is recommended to use the MCLK signal as input for the PDIV divider. If the MCLK signal is not used or a fixed phase relation is not necessary, the faster frequency $f_{PIN}$ can be selected as input frequency.

$$\begin{aligned} f_{PDIV} &= f_{PIN} \times \frac{1}{PDIV + 1} && \text{if PPPEN = 0} \\ f_{PDIV} &= f_{MCLK} \times \frac{1}{PDIV + 1} && \text{if PPPEN = 1} \end{aligned} \tag{15.5}$$

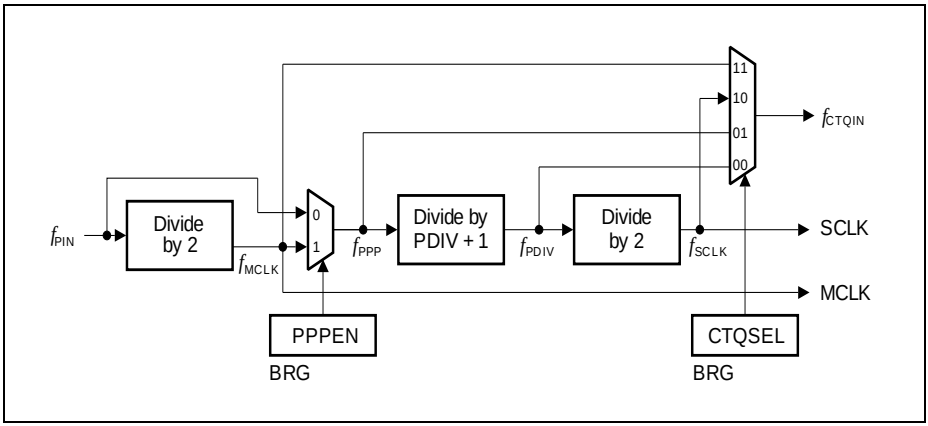



**Figure 15-13 Divider Mode Counter**

### 15.2.4.4 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).

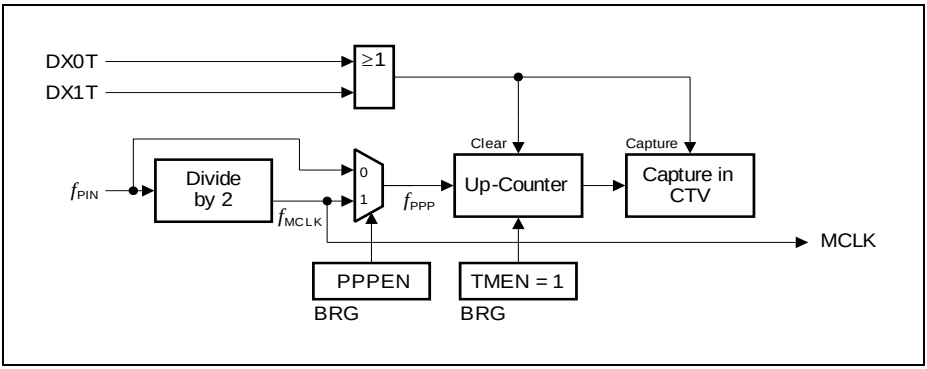



**Figure 15-14 Protocol-Related Counter (Capture Mode)**

The capture mode timer can be used to measure the baud rate in slave mode before starting or during data transfers, e.g. to measure the time between two edges of a data signal (by DX0T) or of a shift clock signal (by DX1T). The conditions to activate the DXnT trigger signals can be configured in each input stage.

### 15.2.4.5 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.

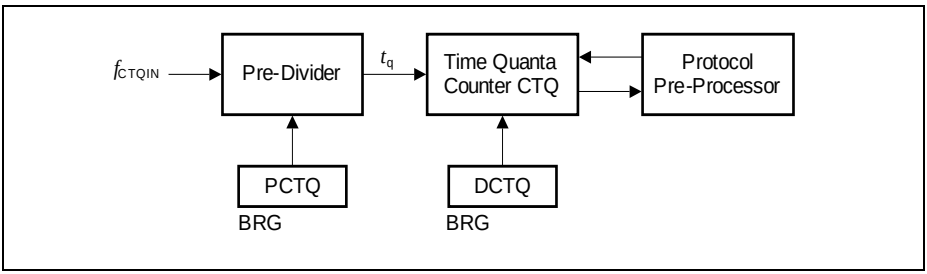



**Figure 15-15 Time Quanta Counter**


Subject to Agreement on the Use of Product Information

### 15.2.4.6 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 15-16**, the value of BRG.MCLKCFG is changed from 0 to 1 during the passive phase of MCLK period 2.

The generation of the MCLKOUT signal is enabled/disabled by the protocol pre-processor, based on bit PCR.MCLK. After this bit has become set, signal MCLKOUT is generated with the next active phase of the MCLK period. If PCR.MCLK = 0 (MCLKOUT generation disabled), the level for the passive phase is also applied for active phase.

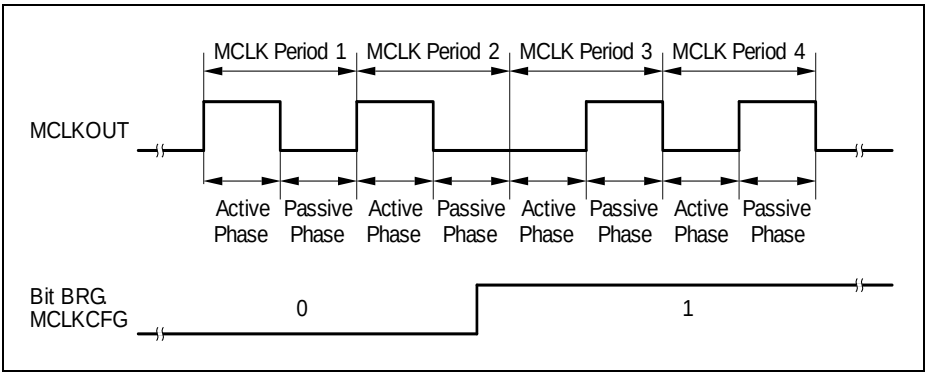



**Figure 15-16 Master Clock Output Configuration**

The shift clock output signal SCLKOUT available at the corresponding output pin can be configured in polarity and additionally, a delay of one period of $f_{PDIV}$ (= half SCLK period) can be introduced. The delay allows to adapt the order of the shift clock edges to the application requirements. If the delay is used, it has to be taken into account for the calculation of the signal propagation times and loop delays.

The mechanism for the polarity control of the SCLKOUT signal is similar to the one for MCLKOUT, but based on bit field BRG.SCLKCFG. The generation of the SCLKOUT

signal is enabled/disabled by the protocol pre-processor. Depending on the selected protocol, the protocol pre-processor can control the generation of the SCLKOUT signal independently of the divider chain, e.g. for protocols without the need of a shift clock available at a pin, the SCLKOUT generation is disabled.

### 15.2.5 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.

#### 15.2.5.1 Transmit Buffering

The transmit shift registers can not be directly accessed by software, because they are automatically updated with the value stored in the transmit buffer TBUF if a currently transmitted data word is finished and new data is valid for transmission. Data words can be loaded directly into TBUF by writing to one of the transmit buffer input locations TBUFx (see **Page 15-33**) or, optionally, by a FIFO buffer stage (see **Page 15-39**).


Subject to Agreement on the Use of Product Information

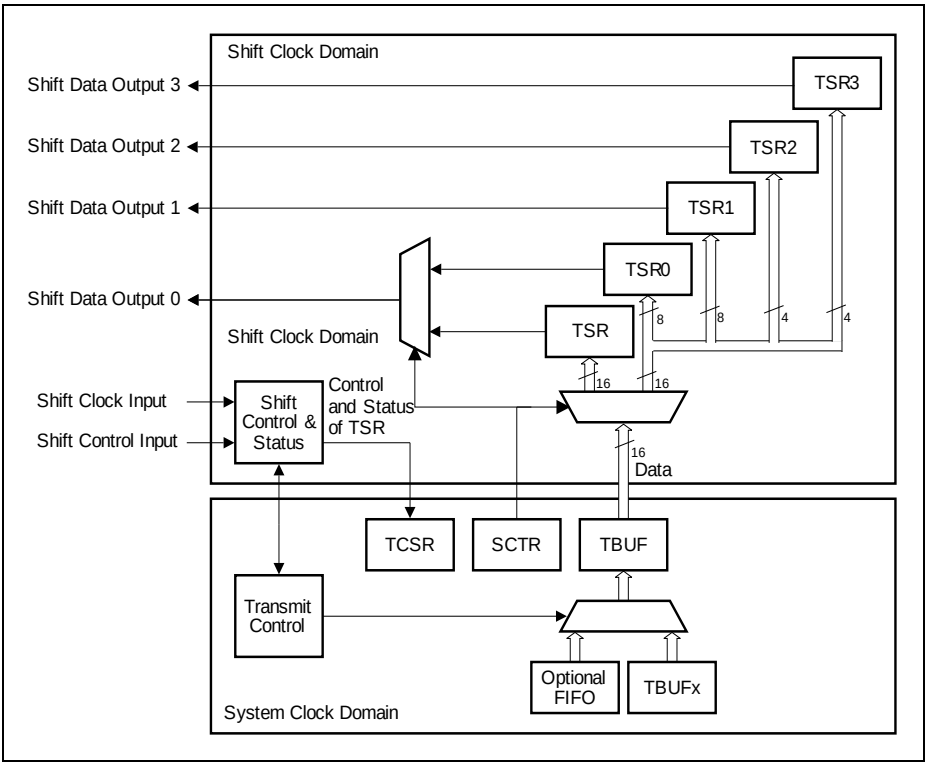



**Figure 15-17 Transmit Data Path**

### 15.2.5.2 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.


Subject to Agreement on the Use of Product Information

Depending on the shift mode, different transmit shift registers with different bit composition are used as shown in **Table 15-8**. Note that the 'n' in the table denotes the shift number less one, i.e. for the first data shift n = 0, the second data shift n = 1 and continues until the total number of shifts less one is reached.

For all transmit shift registers, whether the first bit shifted out is the MSB or LSB depends on the setting of SCTR.SDIR.

**Table 15-8 Transmit Shift Register Composition**

| Transmit Shift Registers | Single Data Output (SCTR.DSM = $00_B$) | Two Data Outputs (SCTR.DSM = $10_B$) | Four Data Outputs (SCTR.DSM = $11_B$) |
|---|---|---|---|
| TSR | All data bits | Not used | Not used |
| TSR0 | Not used | Bit n*2 | Bit n*4 |
| TSR1 | Not used | Bit n*2 + 1 | Bit n*4 + 1 |
| TSR2 | Not used | Not used | Bit n*4 + 2 |
| TSR3 | Not used | Not used | Bit n*4 + 3 |

### 15.2.5.3 Transmit Control Information

The transmit control information TCI is a 5-bit value derived from the address x of the written TBUFx or INx input location. For example, writing to TBUF31 generates a TCI of $11111_B$.

The TCI can be used as an additional control parameter for data transfers to dynamically change the data word length, the data frame length, or other protocol-specific functions (for more details about this topic, please refer to the corresponding protocol chapters). The way how the TCI is used in different applications can be programmed by the bits WLEMD, FLEMD, SELMD, WAMD and HPCMD in register TCSR. Please note that not all possible settings lead to useful system behavior.

- Word length control:
  If TCSR.WLEMD = 1, bit field SCTR.WLE is updated with TCI[3:0] if a transmit buffer input location TBUFx is written. This function can be used in all protocols to dynamically change the data word length between 1 and 16 data bits per data word. Additionally, bit TCSR.EOF is updated with TCI[4]. This function can be used in SSC master mode to control the slave select generation to finish data frames. It is recommended to program TCSR.FLEMD = TCSR.SELMD = TCSR.WAMD = TCSR.HPCMD = 0.
- Frame length control:
  If TCSR.FLEMD = 1, bit field SCTR.FLE[4:0] is updated with TCI[4:0] and SCTR.FLE[5] becomes 0 if a transmit buffer input location TBUFx is written. This function can be used in all protocols to dynamically change the data frame length


Subject to Agreement on the Use of Product Information

between 1 and 32 data bits per data frame. It is recommended to program TCSR.SELMD = TCSR.WLEMD = TCSR.WAMD = TCSR.HPCMD = 0.

- Select output control:
  If TCSR.SELMD = 1, bit field PCR.CTR[20:16] is updated with TCI[4:0] and PCR.CTR[23:21] becomes 0 if a transmit buffer input location TBUFx is written. This function can be used in SSC master mode to define the targeted slave device(s). It is recommended to program TCSR.WLEMD = TCSR.FLEMD = TCSR.WAMD = TCSR.HPCMD = 0.
- Word address control:
  If TCSR.WAMD = 1, bit TCSR.WA is updated with TCI[4] if a transmit buffer input location TBUFx is written. This function can be used in IIS mode to define if the data word is transmitted on the right or the left channel. It is recommended to program TCSR.WLEMD = TCSR.FLEMD = TCSR.SELMD = TCSR.HPCMD = 0.
- Hardware Port control:
  If TCSR.HPCMD = 1, bit field SCTR.DSM is updated with TCI[1:0] if a transmit buffer input location TBUFx is written. This function can be used in SSC protocols to dynamically change the number of data input and output lines to set up for standard, dual and quad SSC formats.
  Additionally, bit TCSR.HPCDIR is updated with TCI[2]. This function can be used in SSC protocols to control the pin(s) direction when the hardware port control function is enabled through CCR.HPCEN = 1. It is recommended to program TCSR.FLEMD = TCSR.WLEMD = TCSR.SELMD = TCSR.WAMD = 0.

#### 15.2.5.4 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.


Subject to Agreement on the Use of Product Information

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 15-18**.

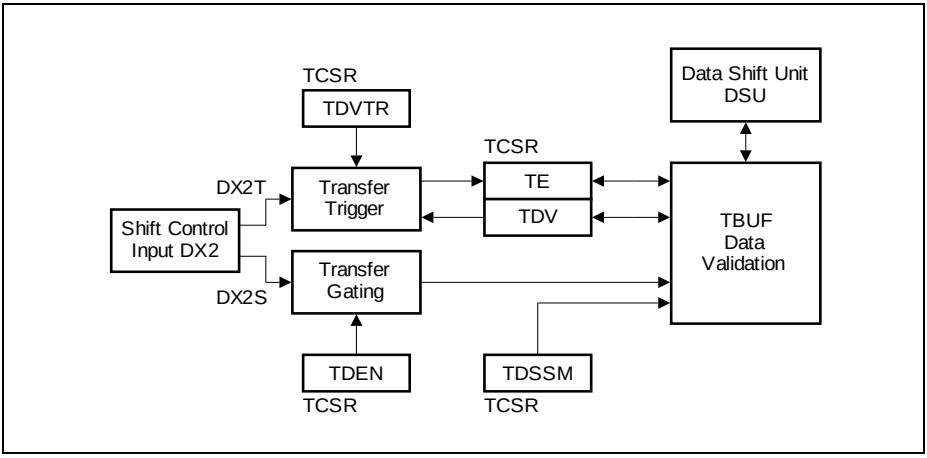



**Figure 15-18 Transmit Data Validation**

- A transfer gating logic enables or disables the data word transfer from TBUF under software or under hardware control. If the input stage DX2 is not needed for data shifting, signal DX2S can be used for gating purposes. The transfer gating logic is controlled by bit field TCSR.TDEN.
- A transfer trigger logic supports data word transfers related to events, e.g. timer based or related to an input pin. If the input stage DX2 is not needed for data shifting, signal DX2T can be used for trigger purposes. The transfer trigger logic is controlled by bit TCSR.TDVTR and the occurrence of a trigger event is indicated by bit TCSR.TE. For example, this can be used for triggering the data transfer upon receiving the Clear to Send (CTS) signal at DX2 in the RS-232 protocol.
- A data validation logic combining the inputs from the gating logic, the triggering logic and DSU signals. A transmission of the data word located in TBUF can only be started if the gating enables the start, bit TCSR.TDV = 1, and bit TCSR.TE = 1.
  The content of the transmit buffer TBUF should not be overwritten with new data while it is valid for transmission and a new transmission can start. If the content of TBUF has to be changed, it is recommended to clear bit TCSR.TDV by writing FMR.MTDV = $10_B$ before updating the data. Bit TCSR.TDV becomes automatically set when TBUF is updated with new data. Another possibility are the interrupts TBI (for ASC and IIC) or RSI (for SSC and IIS) indicating that a transmission has started. While a transmission is in progress, TBUF can be loaded with new data. In this case the user has to take care that an update of the TBUF content takes place before a new transmission starts.

With this structure, the following data transfer functionality can be achieved:


Subject to Agreement on the Use of Product Information

- If bit TCSR.TDSSM = 0, the content of the transmit buffer TBUF is always considered as valid for transmission. The transfer trigger mechanism can be used to start the transfer of the same data word based on the selected event (e.g. on a timer base or an edge at a pin) to realize a kind of life-sign mechanism. Furthermore, in slave mode, it is ensured that always a correct data word is transmitted instead of the passive data level.
- Bit TCSR.TDSSM = 1 has to be programmed to allow word-by-word data transmission with a kind of single-shot mechanism. After each transmission start, a new data word has to be loaded into the transmit buffer TBUF, either by software write actions to one of the transmit buffer input locations TBUFx or by an optional data buffer (e.g. FIFO buffer). To avoid that data words are sent out several times or to allow data handling with an additional data buffer (e.g. FIFO), bit TCSR.TDSSM has to be 1.
- Bit TCSR.TDV becoming automatically set when a new data word is loaded into the transmit buffer TBUF, a transmission start can be requested by a write action of the data to be transmitted to at least the low byte of one of the transmit buffer input locations TBUFx. The additional information TCI can be used to control the data word length or other parameters independently for each data word by a single write access.
- Bit field FMR.MTDV allows software driven modification (set or clear) of bit TCSR.TDV. Together with the gating control bit field TCSR.TDEN, the user can set up the transmit data word without starting the transmission. A possible program sequence could be: clear TCSR.TDEN = $00_B$, write data to TBUFx, clear TCSR.TDV by writing FMR.MTDV = $10_B$, re-enable the gating with TCSR.TDEN = $01_B$ and then set TCSR.TDV under software control by writing FMR.MTDV = $01_B$.

### 15.2.6 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.

#### 15.2.6.1 Receive Buffering

The receive shift registers cannot be directly accessed by software, but their contents are automatically loaded into the receive buffer registers RBUF0 (or RBUF1 respectively) if a complete data word has been received or the frame is finished. The received data words in RBUF0 or RBUF1 can be read out in the correct order directly from register RBUF or, optionally, from a FIFO buffer stage (see **Page 15-39**).

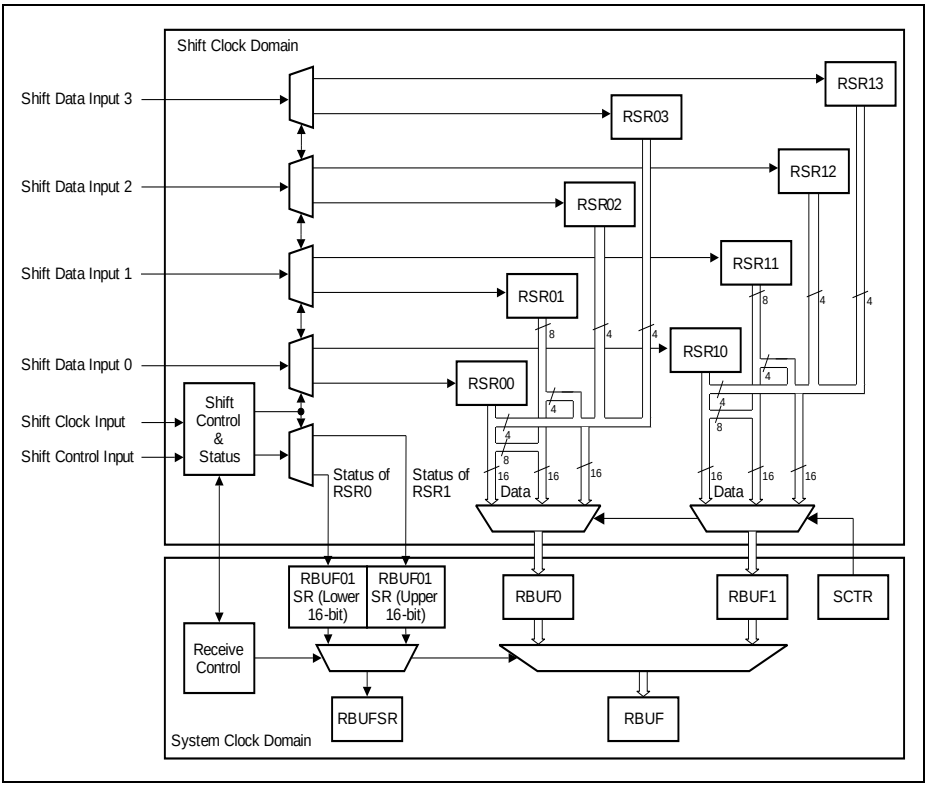



**Figure 15-19 Receive Data Path**

### 15.2.6.2 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 15-8**. Note that the 'n' in the table denotes the shift number less one, i.e. for the first data shift n = 0, the second data shift n = 1 and continues until the total number of shifts less one is reached.

For all receive shift registers, whether the first bit shifted in is the MSB or LSB depends on the setting of SCTR.SDIR.

**Table 15-9 Receive Shift Register Composition**

| Receive Shift Registers | Input stage used | Single Data Input (SCTR.DSM = $00_B$) | Two Data Inputs (SCTR.DSM = $10_B$) | Four Data Inputs (SCTR.DSM = $11_B$) |
|---|---|---|---|---|
| RSRx0 | DX0 | All data bits | Bit n*2 | Bit n*4 |
| RSRx1 | DX3 | Not used | Bit n*2 + 1 | Bit n*4 + 1 |
| RSRx2 | DX4 | Not used | Not used | Bit n*4 + 2 |
| RSRx3 | DX5 | Not used | Not used | Bit n*4 + 3 |

### 15.2.6.3 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.

## 15.2.7 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.


Subject to Agreement on the Use of Product Information

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.

### 15.2.8 Operating the FIFO Data Buffer

The FIFO data buffers of a USIC module are built in a similar way, with transmit buffer and receive buffer capability for each channel. Depending on the device, the amount of available FIFO buffer area can vary. In the XMC1300, totally 64 buffer entries can be distributed among the transmit or receive FIFO buffers of both channels of the USIC module.

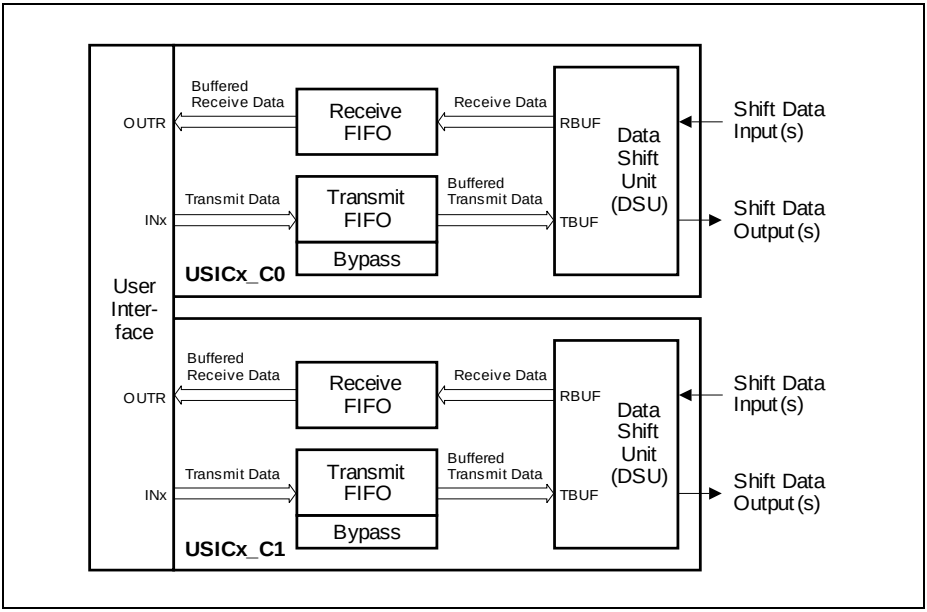



**Figure 15-20 FIFO Buffer Overview**

In order to operate the FIFO data buffers, the following issues have to be considered:


Subject to Agreement on the Use of Product Information

- FIFO buffer available and selected:
  The transmit FIFO buffer and the bypass structure are only available if CCFG.TB = 1, whereas the receive FIFO buffer is only available if CCFG.RB = 1.
  It is recommended to configure all buffer parameters while there is no data traffic for this USIC channel and the FIFO mechanism is disabled by TBCTR.SIZE = 0 (for transmit buffer) or RBCTR.SIZE = 0 (for receive buffer). The allocation of a buffer area by writing TBCTR or RBCTR has to be done while the corresponding FIFO buffer is disabled. The FIFO buffer interrupt control bits can be modified independently of data traffic.
- FIFO buffer setup:
  The total amount of available FIFO buffer entries limits the length of the transmit and receive buffers for each USIC channel.
- Bypass setup:
  In addition to the transmit FIFO buffer, a bypass can be configured as described on **Page 15-50**.

### 15.2.8.1 FIFO Buffer Partitioning

If available, the FIFO buffer area consists of a defined number of FIFO buffer entries, each containing a data part and the associated control information (RCI for receive data, TCI for transmit data). One FIFO buffer entry represents the finest granularity that can be allocated to a receive FIFO buffer or a transmit FIFO buffer. All available FIFO buffer entries of a USIC module are located one after the other in the FIFO buffer area. The overall counting starts with FIFO entry 0, followed by 1, 2, etc.

For each USIC module, a certain number of FIFO entries is available, that can be allocated to the channels of the same USIC module. It is not possible to assign FIFO buffer area to USIC channels that are not located within the same USIC module.
For each USIC channel, the size of the transmit and the receive FIFO buffer can be chosen independently. For example, it is possible to allocate the full amount of available FIFO entries as transmit buffer for one USIC channel. Some possible scenarios of FIFO buffer partitioning are shown in **Figure 15-21**.

Each FIFO buffer consists of a set of consecutive FIFO entries. The size of a FIFO data buffer can only be programmed as a power of 2, starting with 2 entries, then 4 entries, then 8 entries, etc. A FIFO data buffer can only start at a FIFO entry aligned to its size. For example, a FIFO buffer containing n entries can only start with FIFO entry 0, n, 2*n, 3*n, etc. and consists of the FIFO entries [x*n, (x+1)*n-1], with x being an integer number (incl. 0). It is not possible to have “holes” with unused FIFO entries within a FIFO buffer, whereas there can be unused FIFO entries between two FIFO buffers.


Subject to Agreement on the Use of Product Information

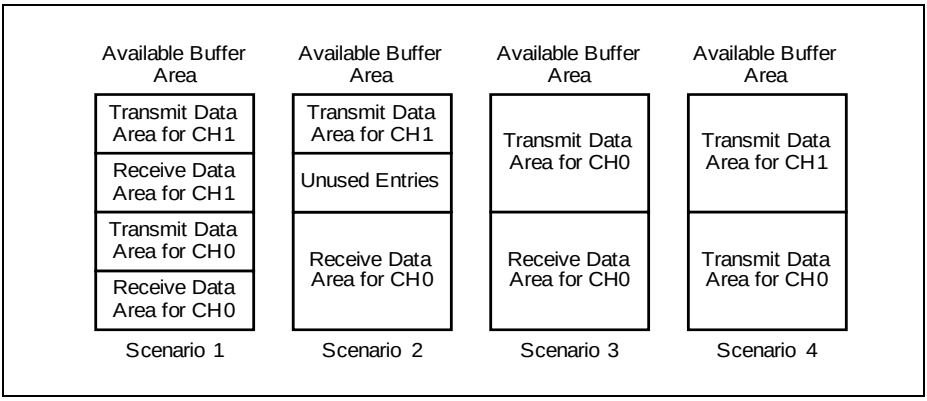



**Figure 15-21 FIFO Buffer Partitioning**

The data storage inside the FIFO buffers is based on pointers, that are internally updated whenever the data contents of the FIFO buffers have been modified. This happens automatically when new data is put into a FIFO buffer or the oldest data is taken from a FIFO buffer. As a consequence, the user program does not need to modify the pointers for data handling. Only during the initialization phase, the start entry of a FIFO buffer has to be defined by writing the number of the first FIFO buffer entry in the FIFO buffer to the corresponding bit field DPTR in register RBCTR (for a receive FIFO buffer) or TBCTR (for a transmit FIFO buffer) while the related bit field RBCTR.SIZE=0 (or TBCTR.SIZE = 0, respectively). The assignment of buffer entries to a FIFO buffer (regarding to size and pointers) must not be changed by software while the related USIC channel is taking part in data traffic.

### 15.2.8.2 Transmit Buffer Events and Interrupts

The transmit FIFO buffer mechanism detects the following events, that can lead to interrupts (if enabled):

- Standard transmit buffer event
- Transmit buffer error event

#### Standard Transmit Buffer Event

The standard transmit buffer event is triggered by the filling level of the transmit buffer (given by TRBSR.TBFLVL) exceeding (TBCTR.LOF = 1) or falling below (TBCTR.LOF = 0)[1)] a programmed limit (TBCTR.LIMIT).

1) If the standard transmit buffer event is used to indicate that new data has to be written to one of the INx locations, TBCTR.LOF = 0 should be programmed.

If the event trigger with TRBSR.STBT feature is disabled (TBCTR.STBTEN = 0), the trigger of the standard transmit buffer event is based on the transition of the fill level from equal to below or above the limit, not the fact of being below or above.

If TBCTR.STBTEN = 1, the transition of the fill level below or above the programmed limit additionally sets TRBSR.STBT. This bit triggers also the standard transmit buffer event whenever there is a transfer data to TBUF event or write data to INx event, depending on TBCTR.LOF setting.

The way TRBSR.STBT is cleared depends on the trigger mode (selected by TBCTR.STBTM). If TBCTR.STBTM = 0, TRBSR.STBT is cleared by hardware when the buffer fill level equals the programmed limit again (TRBSR.TBFLVL = TBCTR.LIMIT). If TBCTR.STBTM = 1, TRBSR.STBT is cleared by hardware when the buffer fill level equals the buffer size (TRBSR.TBFLVL = TBCTR.SIZE).

*Note: The flag TRBSR.STBI is set only when the transmit buffer fill level exceeds or falls below the programmed limit (depending on TBCTR.LOF setting). Standard transmit buffer events triggered by TRBSR.STBT does not set the flag.*

**Figure 15-22** shows examples of the standard transmit buffer event with the different TBCTR.STBTEN and TBCTR.STBTM settings. These examples are meant to illustrate the hardware behaviour and might not always represent real application use cases.


Subject to Agreement on the Use of Product Information

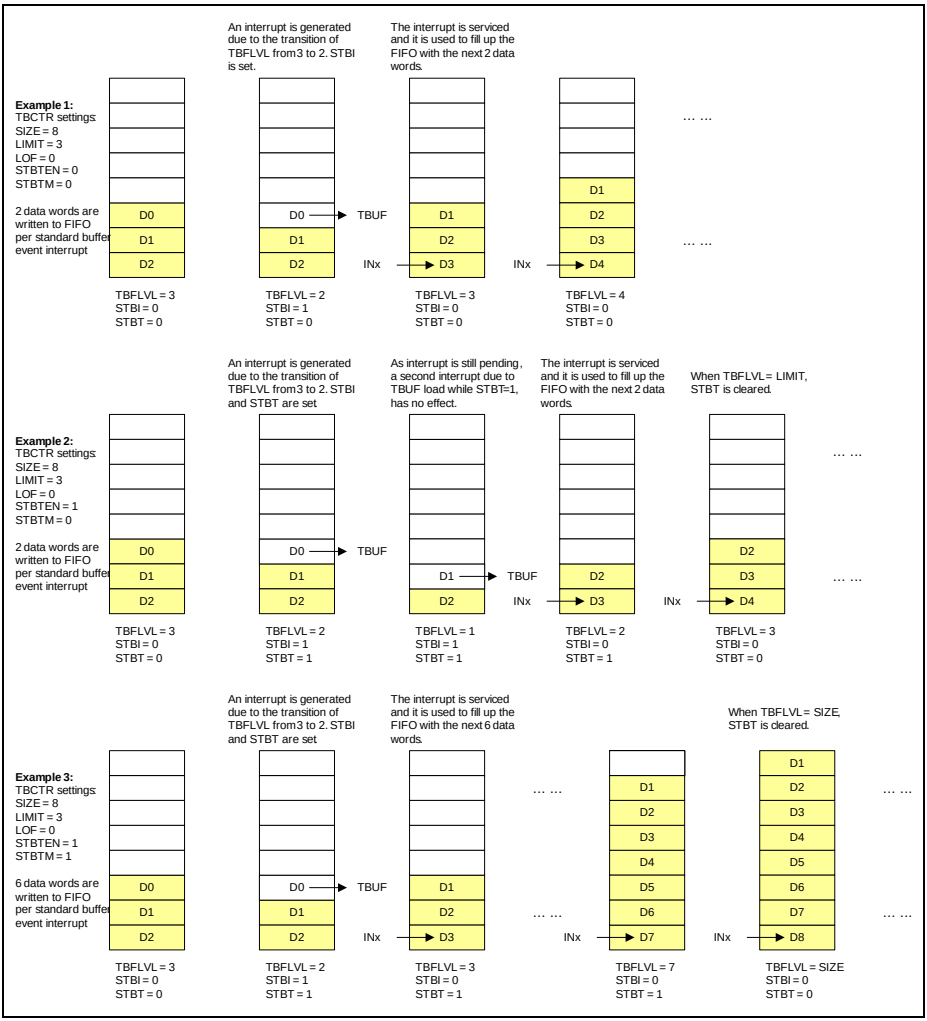



**Figure 15-22 Standard Transmit Buffer Event Examples**

### Transmit Buffer Error Event

The transmit buffer error event is triggered when software has written to a full buffer. The written value is ignored.

### Transmit Buffer Events and Interrupt Handling

**Figure 15-23** shows the transmit buffer events and interrupts.

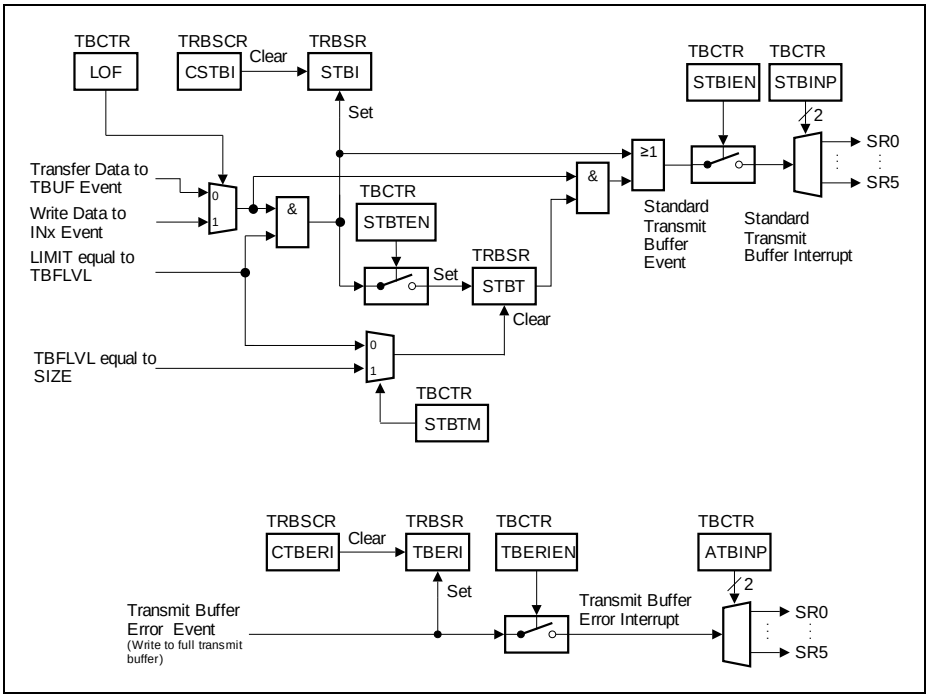



**Figure 15-23 Transmit Buffer Events**

**Table 15-10** shows the registers, bits and bit fields to indicate the transmit buffer events and to control the interrupts related to the transmit FIFO buffers of a USIC channel.

**Table 15-10 Transmit Buffer Events and Interrupt Handling**

| Event | Indication Flag | Indication cleared by | Interrupt enabled by | SRx Output selected by |
|---|---|---|---|---|
| Standard transmit buffer event | TRBSR.STBI | TRBSCR.CSTBI | TBCTR.STBIEN | TBCTR.STBINP |
| | TRBSR.STBT | Cleared by hardware | | |
| Transmit buffer error event | TRBSR.TBERI | TRBSCR.CTBERI | TBCTR.TBERIEN | TBCTR.ATBINP |

### 15.2.8.3 Receive Buffer Events and Interrupts

The receive FIFO buffer mechanism detects the following events, that can lead to an interrupt (if enabled):

- Standard receive buffer event
- Alternative receive buffer event
- Receive buffer error event

The standard receive buffer event and the alternative receive buffer event can be programmed to two different modes, one referring to the filling level of the receive buffer, the other one related to a bit position in the receive control information RCI of the data word that becomes available in OUTR.

If the interrupt generation refers to the filling level of the receive FIFO buffer, only the standard receive buffer event is used, whereas the alternative receive buffer event is not used. This mode can be selected to indicate that a certain amount of data has been received, without regarding the content of the associated RCI.

If the interrupt generation refers to RCI, the filling level is not taken into account. Each time a new data word becomes available in OUTR, an event is detected. If bit RCI[4] = 0, a standard receive buffer event is signaled, otherwise an alternative receive buffer device (RCI[4] = 1). Depending on the selected protocol and the setting of RBCTR.RCIM, the value of RCI[4] can hold different information that can be used for protocol-specific interrupt handling (see protocol sections for more details).

#### Standard Receive Buffer Event in Filling Level Mode

In filling level mode (RBCTR.RNM = 0), the standard receive buffer event is triggered by the filling level of the receive buffer (given by TRBSR.RBFLVL) exceeding (RBCTR.LOF = 1) or falling below (RBCTR.LOF = 0) a programmed limit (RBCTR.LIMIT).[1)]

If the event trigger with bit TRBSR.SRBT feature is disabled (RBCTR.SRBTEN = 0), the trigger of the standard receive buffer event is based on the transition of the fill level from equal to below or above the limit, not the fact of being below or above.

If RBCTR.SRBTEN = 1, the transition of the fill level below or above the programmed limit additionally sets the bit TRBSR.SRBT. This bit also triggers the standard receive buffer event each time there is a data read out event or new data received event, depending on RBCTR.LOF setting.

The way TRBSR.SRBT is cleared depends on the trigger mode (selected by RBCTR.SRBTM). If RBCTR.SRBTM = 0, TRBSR.SRBT is cleared by hardware when the buffer fill level equals the programmed limit again (TRBSR.RBFLVL =

1) If the standard receive buffer event is used to indicate that new data has to be read from OUTR, RBCTR.LOF = 1 should be programmed.


Subject to Agreement on the Use of Product Information

RBCTR.LIMIT). If RBCTR.SRBTM = 1, TRBSR.SRBT is cleared by hardware when the buffer fill level equals 0 (TRBSR.RBFLVL = 0).

*Note: The flag TRBSR.SRBI is set only when the receive buffer fill level exceeds or falls below the programmed limit (depending on RBCTR.LOF setting). Standard receive buffer events triggered by TRBSR.SRBT does not set the flag.*

**Figure 15-24** shows examples of the standard receive buffer event with the different RBCTR.SRBTEN and RBCTR.SRBTM settings. These examples are meant to illustrate the hardware behaviour and might not always represent real application use cases.

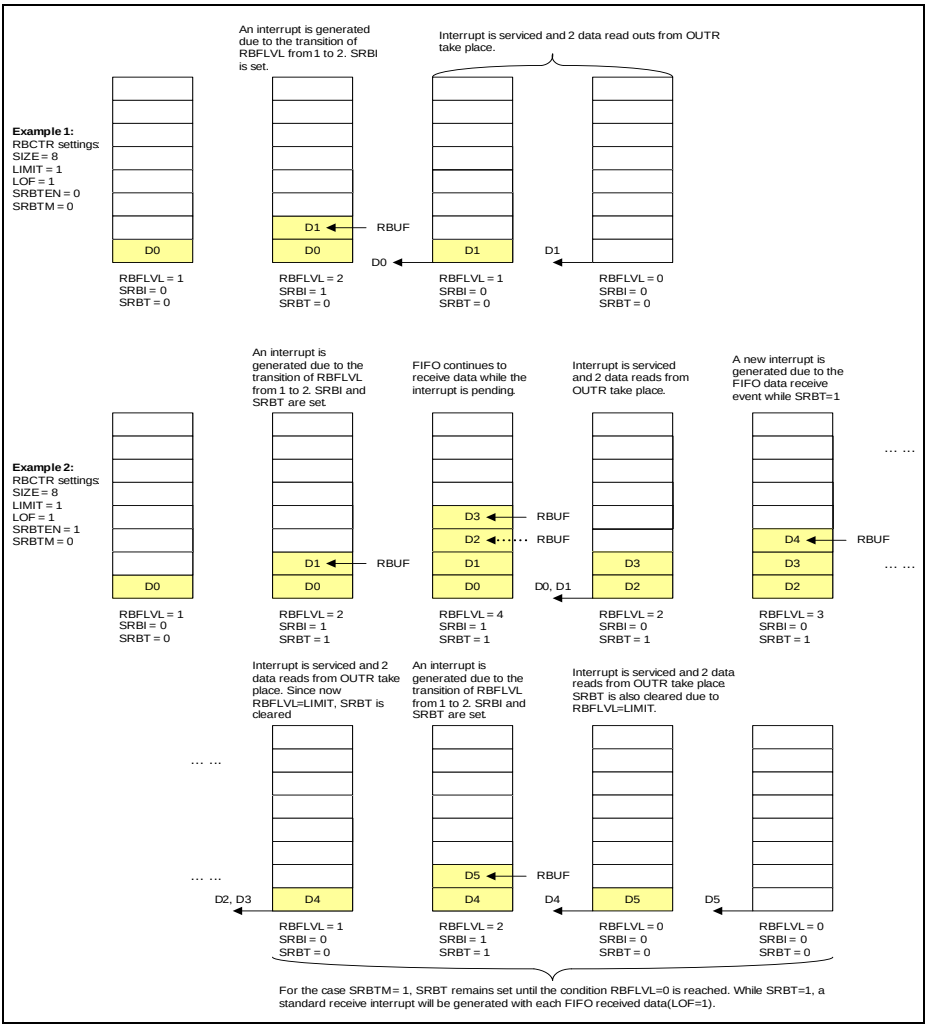



**Figure 15-24 Standard Receive Buffer Event Examples**

### Standard and Alternate Receive Buffer Events in RCI Mode

In RCI mode (RBCTR.RNM = 1), the standard receive buffer event is triggered when the OUTR stage is updated with a new data value with RCI[4] = 0.

If the OUTR stage is updated with a new data value with RCI[4] = 1, an alternate receive buffer event is triggered instead.

### Receive Buffer Error Event

The receive buffer error event is triggered if the software reads from an empty buffer, regardless of RBCTR.RNM value. The read data is invalid.

### Receive Buffer Events and Interrupt Handling

**Figure 15-25** shows the receiver buffer events and interrupts in filling level mode.

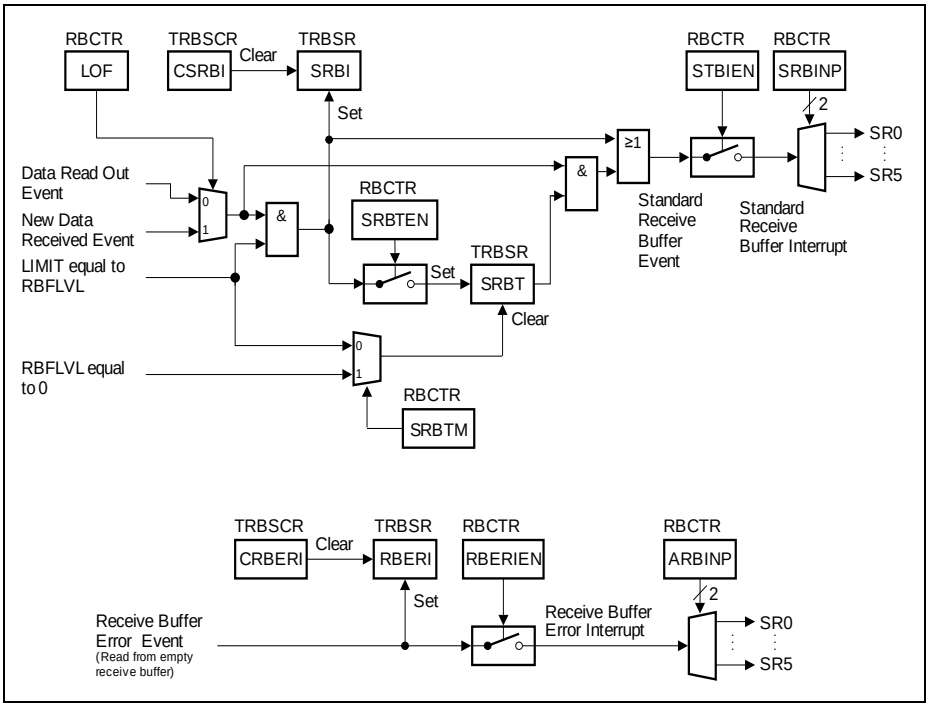



**Figure 15-25 Receiver Buffer Events in Filling Level Mode**

**Figure 15-26** shows the receiver buffer events and interrupts in RCI mode.

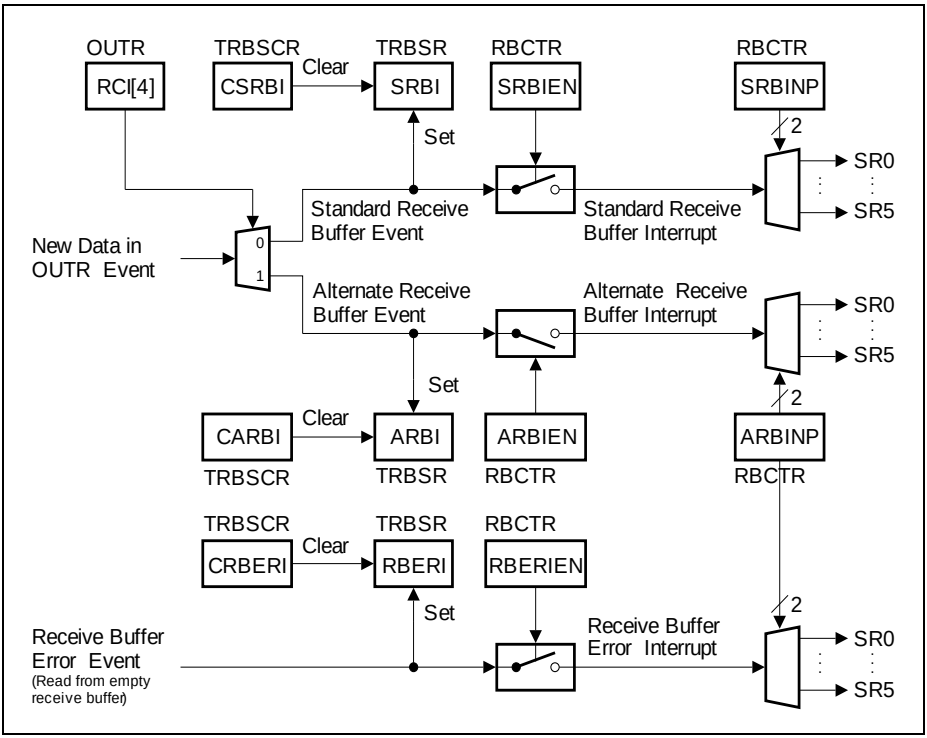



**Figure 15-26 Receiver Buffer Events in RCI Mode**

**Table 15-11** shows the registers, bits and bit fields to indicate the receive buffer events and to control the interrupts related to the receive FIFO buffers of a USIC channel.

**Table 15-11 Receive Buffer Events and Interrupt Handling**

| Event | Indication Flag | Indication cleared by | Interrupt enabled by | SRx Output selected by |
|---|---|---|---|---|
| Standard receive buffer event | TRBSR.SRBI | TRBSCR.CSRBI | RBCTR.SRBIEN | RBCTR.SRBINP |
| | TRBSR.SRBT | Cleared by hardware | | |

**Table 15-11 Receive Buffer Events and Interrupt Handling** (cont'd)

| Event | Indication Flag | Indication cleared by | Interrupt enabled by | SRx Output selected by |
|---|---|---|---|---|
| Alternative receive buffer event | TRBSR. ARBI | TRBSCR. CARBI | RBCTR. ARBIEN | RBCTR. ARBINP |
| Receive buffer error event | TRBSR. RBERI | TRBSCR. CRBERI | RBCTR. RBERIEN | RBCTR. ARBINTXDP |

### 15.2.8.4 FIFO Buffer Bypass

The data bypass mechanism is part of the transmit FIFO control block. It allows to introduce a data word in the data stream without modifying the transmit FIFO buffer contents, e.g. to send a high-priority message. The bypass structure consists of a bypass data word of maximum 16 bits in register BYP and some associated control information in register BYPCR. For example, these bits define the word length of the bypass data word and configure a transfer trigger and gating mechanism similar to the one for the transmit buffer TBUF.

The bypass data word can be tagged valid or invalid for transmission by bit BYRCR.BDV (bypass data valid). A combination of data flow related and event related criteria define whether the bypass data word is considered valid for transmission. A data validation logic checks the start conditions for this data word. Depending on the result of the check, the transmit buffer register TBUF is loaded with different values, according to the following rules:

- Data from the transmit FIFO buffer or the bypass data can only be transferred to TBUF if TCSR.TDV = 0 (TBUF is empty).
- Bypass data can only be transferred to TBUF if the bypass is enabled by BYPCR.BDEN or the selecting gating condition is met.
- If the bypass data is valid for transmission and has either a higher transmit priority than the FIFO data or if the transmit FIFO is empty, the bypass data is transferred to TBUF.
- If the bypass data is valid for transmission and has a lower transmit priority than the FIFO buffer that contains valid data, the oldest transmit FIFO data is transferred to TBUF.
- If the bypass data is not valid for transmission and the FIFO buffer contains valid data, the oldest FIFO data is transferred to TBUF.
- If neither the bypass data is valid for transmission nor the transmit FIFO buffer contains valid data, TBUF is unchanged.

The bypass data validation is based on the logic blocks shown in **Figure 15-27**.

- A transfer gating logic enables or disables the bypass data word transfer to TBUF under software or under hardware control. If the input stage DX2 is not needed for


Subject to Agreement on the Use of Product Information

data shifting, signal DX2S can be used for gating purposes. The transfer gating logic is controlled by bit field BYPCR.BDEN.

- A transfer trigger logic supports data word transfers related to events, e.g. timer based or related to an input pin. If the input stage DX2 is not needed for data shifting, signal DX2T can be used for trigger purposes. The transfer trigger logic is controlled by bit BYPCR.BDVTR.
- A bypass data validation logic combining the inputs from the gating logic, the triggering logic and TCSR.TDV.

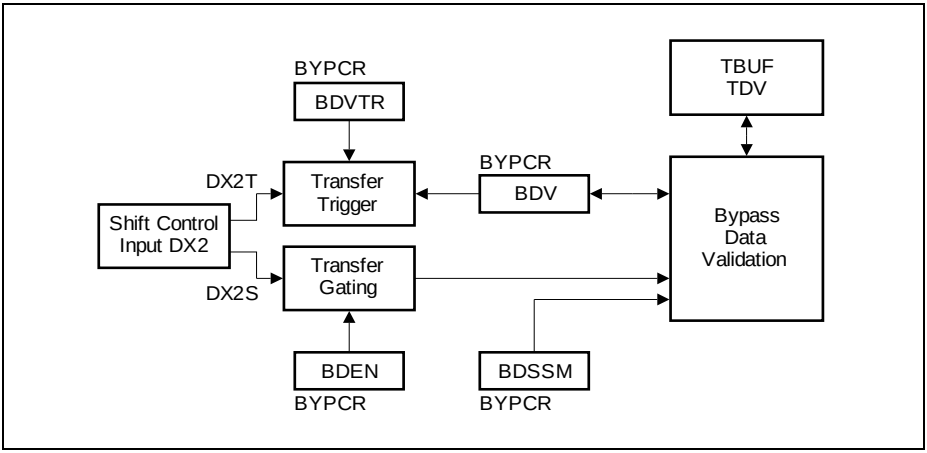



**Figure 15-27 Bypass Data Validation**

With this structure, the following bypass data transfer functionality can be achieved:

- Bit BYPCR.BDSSM = 1 has to be programmed for a single-shot mechanism. After each transfer of the bypass data word to TBUF, the bypass data word has to be tagged valid again. This can be achieved either by writing a new bypass data word to BYP or by DX2T if BDVTR = 1 (e.g. trigger on a timer base or an edge at a pin).
- Bit BYPCR.BDSSM = 0 has to be programmed if the bypass data is permanently valid for transmission (e.g. as alternative data if the data FIFO runs empty).

### 15.2.8.5 FIFO Access Constraints

The data in the shared FIFO buffer area is accessed by the hardware mechanisms for data transfer of each communication channel (for transmission and reception) and by software to read out received data or to write data to be transmitted. As a consequence, the data delivery rate can be limited by the FIFO mechanism. Each access by hardware to the FIFO buffer area has priority over a software access, that is delayed in case of an access collision.

In order to avoid data loss and stalling of the CPU due to delayed software accesses, the

baud rate, the word length and the software access mechanism have to be taken into account. Each access to the FIFO data buffer area by software or by hardware takes one period of $f_{PERIPH}$. Especially a continuous flow of very short, consecutive data words can lead to an access limitation.

### 15.2.8.6 Handling of FIFO Transmit Control Information

In addition to the transmit data, the transmit control information TCI can be transferred from the transmit FIFO or bypass structure to the USIC channel. Depending on the selected protocol and the enabled update mechanism, some settings of the USIC channel parameters can be modified. The modifications are based on the TCI of the FIFO data word loaded to TBUF or by the bypass control information if the bypass data is loaded into TBUF.

- TCSR.SELMD = 1: update of PCR.CTR[20:16] by FIFO TCI or BYPCR.BSELO with additional clear of PCR.CTR[23:21]
- TCSR.WLEMD = 1: update of SCTR.WLE and TCSR.EOF by FIFO TCI or BYPCR.BWLE (if the WLE information is overwritten by TCI or BWLE, the user has to take care that FLE is set accordingly)
- TCSR.FLEMD = 1: update of SCTR.FLE[4:0] by FIFO TCI or BYPCR.BWLE with additional clear of SCTR.FLE[5]
- TCSR.HPCMD = 1: update of SCTR.DSM and SCTR.HPCDIR by FIFO TCI or BYPCR.BHPC
- TCSR.WAMD = 1: update of TCSR.WA by FIFO TCI[4]

See **Section 15.2.5.3** for more details on TCI.

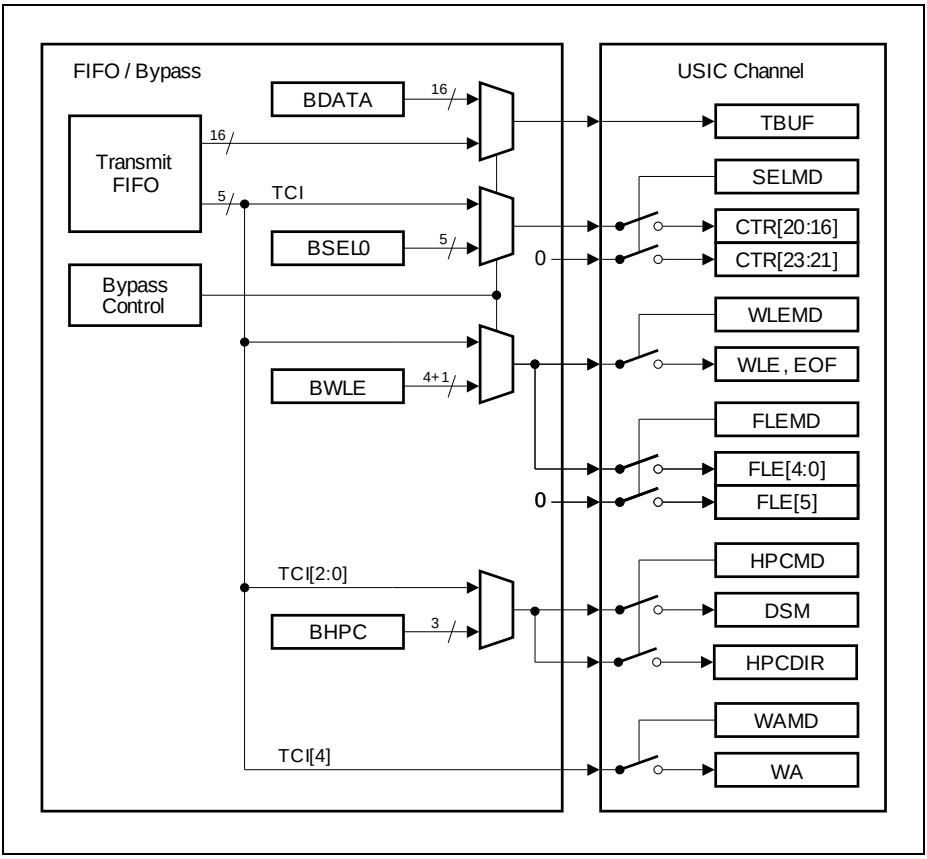



**Figure 15-28 TCI Handling with FIFO / Bypass**


Subject to Agreement on the Use of Product Information

## 15.3 Asynchronous Serial Channel (ASC = UART)

The asynchronous serial channel ASC covers the reception and the transmission of asynchronous data frames and provides a hardware LIN support. The receiver and transmitter being independent, frames can start at different points in time for transmission and reception. The ASC mode is selected by CCR.MODE = $0010_B$ with CCFG.ASC = 1 (ASC mode available).

### 15.3.1 Signal Description

An ASC connection is characterized by the use of a single connection line between a transmitter and a receiver. The receiver input RXD signal is handled by the input stage DX0.

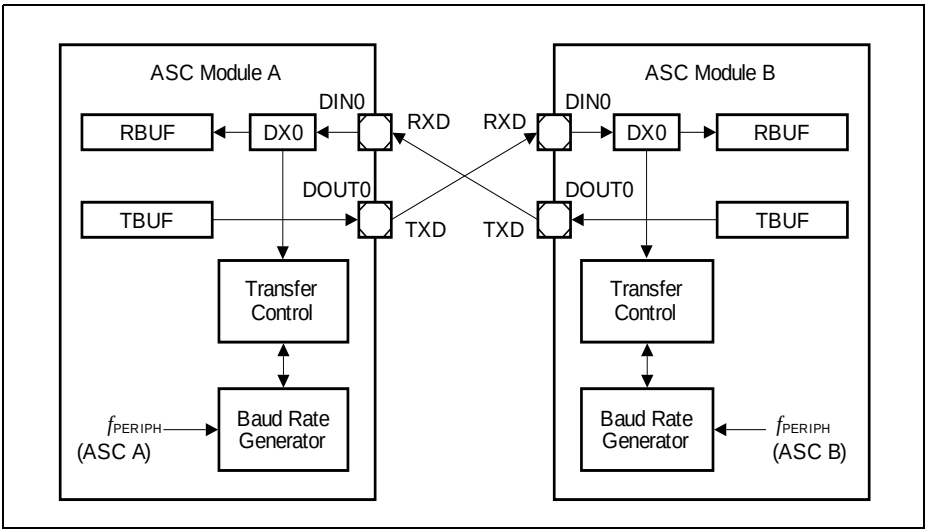



**Figure 15-29 ASC Signal Connections for Full-Duplex Communication**

For full-duplex communication, an independent communication line is needed for each transfer direction. **Figure 15-29** shows an example with a point-to-point full-duplex connection between two communication partners ASC A and ASC B.

For half-duplex or multi-transmitter communication, a single communication line is shared between the communication partners. **Figure 15-30** shows an example with a point-to-point half-duplex connection between ASC A and ASC B. In this case, the user has to take care that only one transmitter is active at a time. In order to support transmitter collision detection, the input stage DX1 can be used to monitor the level of the transmit line and to check if the line is in the idle state or if a collision occurred.
There are two possibilities to connect the receiver input DIN0 to the transmitter output

DOUT0. Communication partner ASC A uses an internal connection with only the transmit pin TXD, that is delivering its input value as RXD to the DX0 input stage for reception and to DX1 to check for transmitter collisions. Communication partner ASC B uses an external connection between the two pins TXD and RXD.

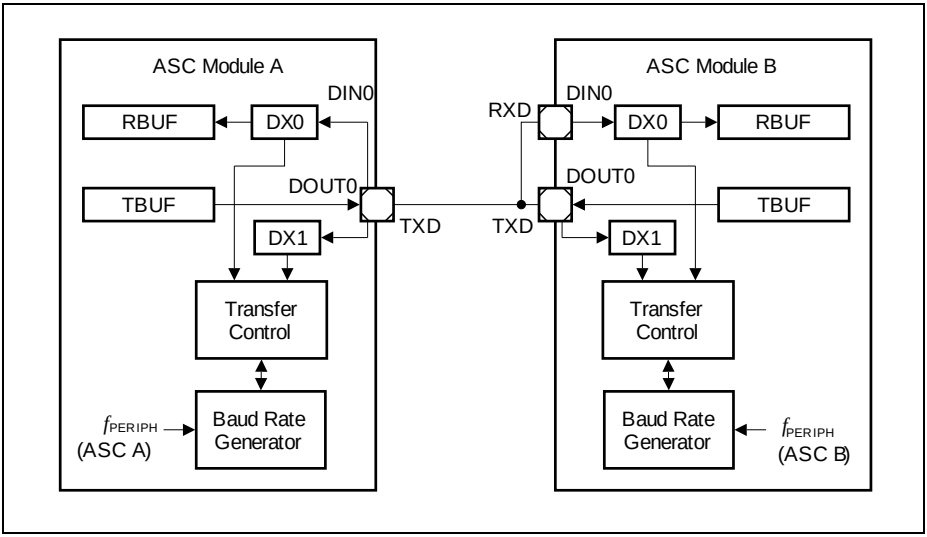



**Figure 15-30 ASC Signal Connections for Half-Duplex Communication**

## 15.3.2 Frame Format

A standard ASC frame is shown in **Figure 15-31**. It consists of:

- An idle time with the signal level 1.
- One start of frame bit (SOF) with the signal level 0.
- A data field containing a programmable number of data bits (1-63).
- A parity bit (P), programmable for either even or odd parity. It is optionally possible to handle frames without parity bit.
- One or two stop bits with the signal level 1.


Subject to Agreement on the Use of Product Information

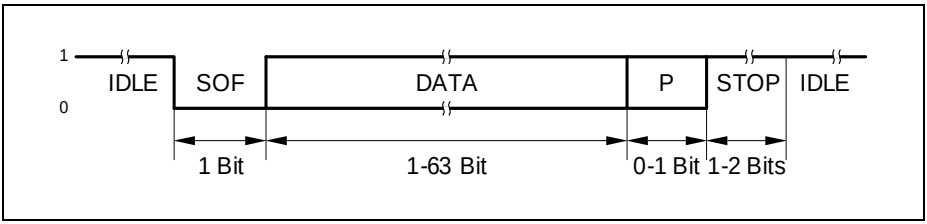



**Figure 15-31 Standard ASC Frame Format**

The protocol specific bits (SOF, P, STOP) are automatically handled by the ASC protocol state machine and do not appear in the data flow via the receive and transmit buffers.

### 15.3.2.1 Idle Time

The receiver and the transmitter independently check the respective data input lines (DX0, DX1) for being idle. The idle detection ensures that an SOF bit of a recently enabled ASC module does not collide with an already running frame of another ASC module.

In order to start the idle detection, the user software has to clear bits PSR.RXIDLE and/or PSR.TXIDLE, e.g. before selecting the ASC mode or during operation. If a bit is cleared by software while a data transfer is in progress, the currently running frame transfer is finished normally before starting the idle detection again. Frame reception is only possible if PSR.RXIDLE = 1 and frame transmission is only possible if PSR.TXIDLE = 1. The duration of the idle detection depends on the setting of bit PCR.IDM. In the case that a collision is not possible, the duration can be shortened and the bus can be declared as being idle by setting PCR.IDM = 0.

In the case that the complete idle detection is enabled by PCR.IDM = 1, the data input of DX0 is considered as idle (PSR.RXIDLE becomes set) if a certain number of consecutive passive bit times has been detected. The same scheme applies for the transmitter's data input of DX1. Here, bit PSR.TXIDLE becomes set if the idle condition of this input signal has been detected.

The duration of the complete idle detection is given by the number of programmed data bits per frame plus 2 (in the case without parity) or plus 3 (in the case with parity). The counting of consecutive bit times with 1 level restarts from the beginning each time an edge is found, after leaving a stop mode or if ASC mode becomes enabled.

If the idle detection bits PSR.RXIDLE and/or TXIDLE are cleared by software, the counting scheme is not stopped (no re-start from the beginning). As a result, the cleared bit(s) can become set immediately again if the respective input line still meets the idle criterion.

Please note that the idle time check is based on bit times, so the maximum time can be up to 1 bit time more than programmed value (but not less).


Subject to Agreement on the Use of Product Information

#### 15.3.2.2 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.

#### 15.3.2.3 Data Field

The length of the data field (number of data bits) can be programmed by bit field SCTR.FLE. It can vary between 1 and 63 data bits, corresponding to values of SCTR.FLE = 0 to 62 (the value of 63 is reserved and must not be programmed in ASC mode).

The data field can consist of several data words, e.g. a transfer of 12 data bits can be composed of two 8-bit words, with the 12 bits being split into 8-bits of the first word and 4 bits of the second word. The user software has to take care that the transmit data is available in-time, once a frame has been started. If the transmit buffer runs empty during a running data frame, the passive data level (SCTR.PDL) is sent out.

The shift direction can be programmed by SCTR.SDIR. The standard setting for ASC frames with LSB first is achieved with the default setting SDIR = 0.

#### 15.3.2.4 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).

#### 15.3.2.5 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.


Subject to Agreement on the Use of Product Information

### 15.3.3 Operating the ASC

In order to operate the ASC protocol, the following issues have to be considered:

- Select ASC mode:
  It is recommended to configure all parameters of the ASC that do not change during run time while CCR.MODE = $0000_B$. Bit field SCTR.TRM = $01_B$ has to be programmed. The configuration of the input stages has to be done while CCR.MODE = $0000_B$ to avoid unintended edges of the input signals and the ASC mode can be enabled by CCR.MODE = $0010_B$ afterwards.
- Pin connections:
  Establish a connection of input stage DX0 with the selected receive data input pin (signal DIN0) with DX0CR.INSW = 0 and configure a transmit data output pin (signal DOUT0). For collision or idle detection of the transmitter, the input stage DX1 has to be connected to the selected transmit output pin, also with DX1CR.INSW = 0. Additionally, program DX2CR.INSW = 0.
  Due to the handling of the input data stream by the synchronous protocol handler, the propagation delay of the synchronization in the input stage has to be considered.
  Note that the step to enable the alternate output port functions should only be done after the ASC mode is enabled, to avoided unintended spikes on the output.
- Bit timing configuration:
  The desired baud rate setting has to be selected, comprising the fractional divider, the baud rate generator and the bit timing. Please note that not all feature combinations can be supported by the application at the same time, e.g. due to propagation delays. For example, the length of a frame is limited by the frequency difference of the transmitter and the receiver device. Furthermore, in order to use the average of samples (SMD = 1), the sampling point has to be chosen to respect the signal settling and data propagation times.
- Data format configuration:
  The word length, the frame length, and the shift direction have to be set up according to the application requirements by programming the register SCTR. If required by the application, the data input and output signals can be inverted.
  Additionally, the parity mode has to be configured (CCR.PM).

#### 15.3.3.1 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 15-32**, one bit time is composed of 16 time quanta (BRG.DCTQ = 15). It is not recommended to program less than 4 time quanta per bit time.


Subject to Agreement on the Use of Product Information

Bit field PCR.SP determines the position of the sampling point for the bit value. The value of PCR.SP must not be set to a value greater than BRG.DCTQ. It is possible to sample the bit value only once per bit time or to take the average of samples. Depending on bit PCR.SMD, either the current input value is directly sampled as bit value, or a majority decision over the input values sampled at the latest three time quanta is taken into account. The standard ASC bit timing consists of 16 time quanta with sampling after 8 or 9 time quanta with majority decision.

The bit timing setup (number of time quanta and the sampling point definition) is common for the transmitter and the receiver. Due to independent bit timing blocks, the receiver and the transmitter can be in different time quanta or bit positions inside their frames. The transmission of a frame is aligned to the time quanta generation.

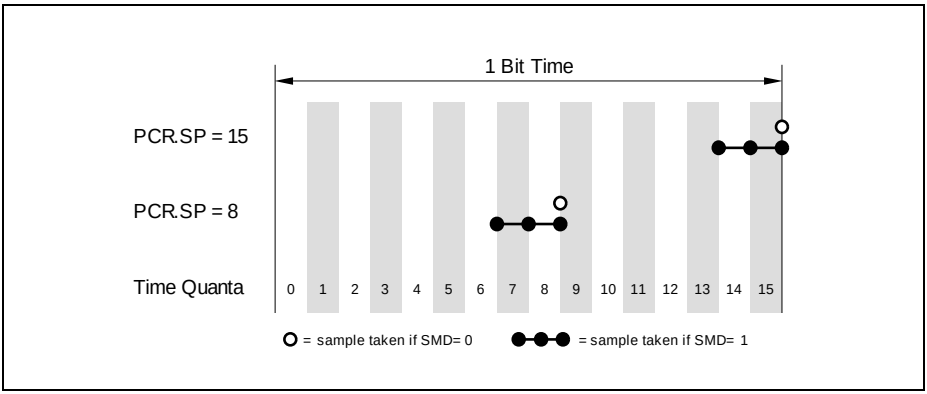



**Figure 15-32 ASC Bit Timing**

The sample point setting has to be adjusted carefully if collision or idle detection is enabled (via DX1 input signal), because the driver delay and some external delays have to be taken into account. The sample point for the transmit line has to be set to a value where the bit level is stable enough to be evaluated.
If the sample point is located late in the bit time, the signal itself has more time to become stable, but the robustness against differences in the clock frequency of transmitter and receiver decreases.

### 15.3.3.2 Baud Rate Generation

The baud rate $f_{ASC}$ in ASC mode depends on the number of time quanta per bit time and their timing. The baud rate setting should only be changed while the transmitter and the receiver are idle. The bits in register BRG define the baud rate setting:

- BRG.CTQSEL
  to define the input frequency $f_{CTQIN}$ for the time quanta generation

- BRG.PCTQ
  to define the length of a time quantum (division of $f_{CTQIN}$ by 1, 2, 3, or 4)
- BRG.DCTQ
  to define the number of time quanta per bit time

The standard setting is given by CTQSEL = $00_B$ ($f_{CTQIN} = f_{PDIV}$) and PPPEN = 0 ($f_{PPP} = f_{PIN}$). Under these conditions, the baud rate is given by:

$$f_{ASC} = f_{PIN} \times \frac{1}{PDIV + 1} \times \frac{1}{PCTQ + 1} \times \frac{1}{DCTQ + 1} \quad (15.6)$$

In order to generate slower frequencies, two additional divide-by-2 stages can be selected by CTQSEL = $10_B$ ($f_{CTQIN} = f_{SCLK}$) and PPPEN = 1 ($f_{PPP} = f_{MCLK}$), leading to:

$$f_{ASC} = \frac{f_{PIN}}{2 \times 2} \times \frac{1}{PDIV + 1} \times \frac{1}{PCTQ + 1} \times \frac{1}{DCTQ + 1} \quad (15.7)$$

### 15.3.3.3 Noise Detection

The ASC receiver permanently checks the data input line of the DX0 stage for noise (the check is independent from the setting of bit PCR.SMD). Bit PSR.RNS (receiver noise) becomes set if the three input samples of the majority decision are not identical at the sample point for the bit value. The information about receiver noise gets accumulated over several bits in bit PSR.RNS (it has to be cleared by software) and can trigger a protocol interrupt each time noise is detected if enabled by PCR.RNIEN.

### 15.3.3.4 Collision Detection

In some applications, such as data transfer over a single data line shared by several sending devices (see **Figure 15-30**), several transmitters have the possibility to send on the same data output line TXD. In order to avoid collisions of transmitters being active at the same time or to allow a kind of arbitration, a collision detection has been implemented.

The data value read at the TXD input at the DX1 stage and the transmitted data bit value are compared after the sampling of each bit value. If enabled by PCR.CDEN = 1 and a bit sent is not equal to the bit read back, a collision is detected and bit PSR.COL is set. If enabled, bit PSR.COL = 1 disables the transmitter (the data output lines become 1) and generates a protocol interrupt. The content of the transmit shift register is considered as invalid, so the transmit buffer has to be programmed again.

### 15.3.3.5 Pulse Shaping

For some applications, the 0 level of transmitted bits with the bit value 0 is not applied at the transmit output during the complete bit time. Instead of driving the original 0 level,

only a 0 pulse is generated and the remaining time quanta of the bit time are driven with 1 level. The length of a bit time is not changed by the pulse shaping, only the signalling is changed.

In the standard ASC signalling scheme, the 0 level is signalled during the complete bit time with bit value 0 (ensured by programming PCR.PL = $000_B$). In the case PCR.PL > $000_B$, the transmit output signal becomes 0 for the number of time quanta defined by PCR.PL. In order to support correct reception with pulse shaping by the transmitter, the sample point has to be adjusted in the receiver according to the applied pulse length.

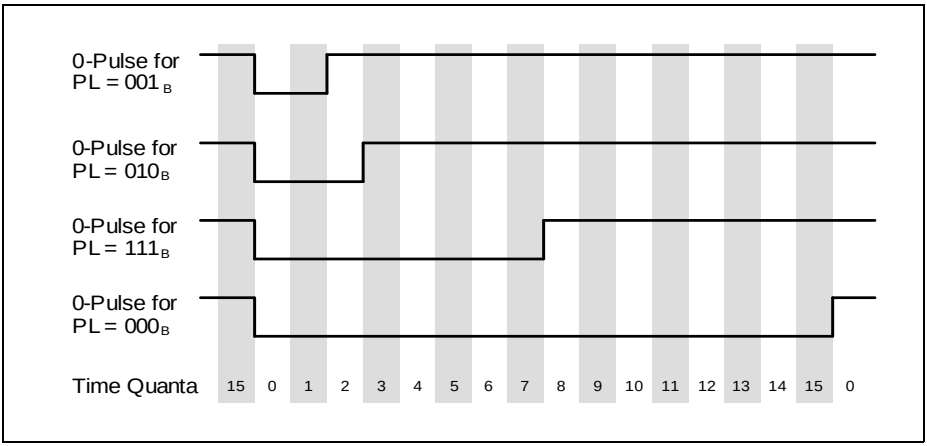



**Figure 15-33 Transmitter Pulse Length Control**

**Figure 15-34** shows an example for the transmission of an 8-bit data word with LSB first and one stop bit (e.g. like for IrDA). The polarity of the transmit output signal has been inverted by SCTR.DOCFG = $01_B$.

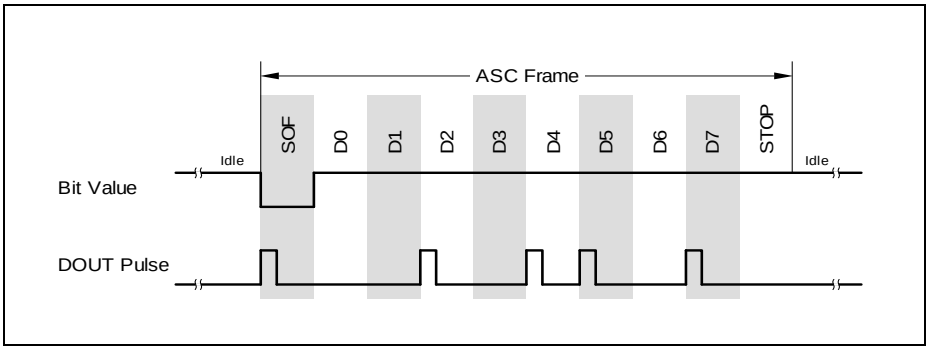



**Figure 15-34 Pulse Output Example**

### 15.3.3.6 Automatic Shadow Mechanism

The contents of the protocol control register PCR, as well as bit field SCTR.FLE are internally kept constant while a data frame is transferred by an automatic shadow mechanism (shadowing takes place with each frame start). The registers can be programmed all the time with new settings that are taken into account for the next data frame. During a data frame, the applied (shadowed) setting is not changed, although new values have been written after the start of the data frame.
Bit fields SCTR.WLE and SCTR.SDIR are shadowed automatically with the start of each data word. As a result, a data frame can consist of data words with a different length. It is recommended to change SCTR.SDIR only when no data frame is running to avoid interference between hardware and software.

Please note that the starting point of a data word can be different for a transmitter and a receiver. In order to ensure correct handling, it is recommended to modify SCTR.WLE only while transmitter and receiver are both idle. If the transmitter and the receiver are referring to the same data signal (e.g. in a LIN bus system), SCTR.WLE can be modified while a data transfer is in progress after the RSI event has been detected.

### 15.3.3.7 End of Frame Control

The number of bits per ASC frame is defined by bit field SCTR.FLE. In order to support different frame length settings for consecutively transmitted frames, this bit field can be modified by hardware. The automatic update mechanism is enabled by TCSR.FLEMD = 1 (in this case, bits TCSR.WLEMD, SELMD, WAMD and HPCMD have to be written with 0).

If enabled, the transmit control information TCI automatically overwrites the bit field TCSR.FLEMD when the ASC frame is started (leading to frames with 1 to 32 data bits). The TCI value represents the written address location of TBUFxx (without additional data

buffer) or INxx (with additional data buffer). With this mechanism, an ASC with 8 data bits is generated by writing a data word to TBUF07 (IN07, respectively).

### 15.3.3.8 Mode Control Behavior

In ASC mode, the following kernel modes are supported:

- Run Mode 0/1:
  Behavior as programmed, no impact on data transfers.
- Stop Mode 0:
  Bit PSR.TXIDLE is cleared. A new transmission is not started. A current transmission is finished normally. Bit PSR.RXIDLE is not modified. Reception is still possible.
  When leaving stop mode 0, bit TXIDLE is set according to PCR.IDM.
- Stop Mode 1:
  Bit PSR.TXIDLE is cleared. A new transmission is not started. A current transmission is finished normally. Bit PSR.RXIDLE is cleared. A new reception is not possible. A current reception is finished normally.
  When leaving stop mode 1, bits TXIDLE and RXIDLE are set according to PCR.IDM.

### 15.3.3.9 Disabling ASC Mode

In order to switch off ASC mode without any data corruption, the receiver and the transmitter have to be both idle. This is ensured by requesting Stop Mode 1 in register KSCFG. After waiting for the end of the frame, the ASC mode can be disabled.

### 15.3.3.10 Protocol Interrupt Events

The following protocol-related events are generated in ASC mode and can lead to a protocol interrupt. The collision detection and the transmitter frame finished events are related to the transmitter, whereas the receiver events are given by the synchronization break detection, the receiver noise detection, the format error checks and the end of the received frame.

Please note that the bits in register PSR are not automatically cleared by hardware and have to be cleared by software in order to monitor new incoming events.

- Collision detection:
  This interrupt indicates that the transmitted value (DOUT0) does not match with the input value of the DX1 input stage at the sample point of a bit. For more details refer to **Page 15-60**.
- Transmitter frame finished:
  This interrupt indicates that the transmitter has completely finished a frame. Bit PSR.TFF becomes set at the end of the last stop bit. The DOUT0 signal assignment to port pins can be changed while no transmission is in progress.
- Receiver frame finished:
  This interrupt indicates that the receiver has completely finished a frame. Bit

PSR.RFF becomes set at the end of the last stop bit. The DIN0 signal assignment to port pins can be changed while no reception is in progress.

- Synchronization break detection:
  This interrupt can be used in LIN networks to indicate the reception of the synchronization break symbol (at the beginning of a LIN frame).
- Receiver noise detection:
  This interrupt indicates that the input value at the sample point of a bit and at the two time quanta before are not identical.
- Format error:
  The bit value of the stop bit(s) is defined as 1 level for the ASC protocol. A format error is signalled if the sampled bit value of a stop bit is 0.

### 15.3.3.11 Data Transfer Interrupt Handling

The data transfer interrupts indicate events related to ASC frame handling.

- Transmit buffer interrupt TBI:
  Bit PSR.TBIF is set after the start of first data bit of a data word. This is the earliest point in time when a new data word can be written to TBUF.
  With this event, bit TCSR.TDV is cleared and new data can be loaded to the transmit buffer.
- Transmit shift interrupt TSI:
  Bit PSR.TSIF is set after the start of the last data bit of a data word.
- Receiver start interrupt RSI:
  Bit PSR.RSIF is set after the sample point of the first data bit of a data word.
- Receiver interrupt RI and alternative interrupt AI:
  Bit PSR.RIF is set after the sampling point of the last data bit of a data word if this data word is not directly followed by a parity bit (parity generation disabled or not the last word of a data frame).
  If the data word is directly followed by a parity bit (last data word of a data frame and parity generation enabled), bit PSR.RIF is set after the sampling point of the parity bit if no parity error has been detected. If a parity error has been detected, bit PSR.AIF is set instead of bit PSR.RIF.
  The first data word of a data frame is indicated by RBUFSR.SOF = 1 for the received word.
  Bit PSR.RIF is set for a receiver interrupt RI with WA = 0. Bit PSR.AIF is set for a alternative interrupt AI with WA = 1.

### 15.3.3.12 Baud Rate Generator Interrupt Handling

The baud rate generator interrupt indicate that the capture mode timer has reached its maximum value. With this event, the bit PSR.BRGIF is set.


Subject to Agreement on the Use of Product Information

### 15.3.3.13 Protocol-Related Argument and Error

The protocol-related argument (RBUFSR.PAR) and the protocol-related error (RBUFSR.PERR) are two flags that are assigned to each received data word in the corresponding receiver buffer status registers.

In ASC mode, the received parity bit is monitored by the protocol-related argument and the result of the parity check by the protocol-related error indication (0 = received parity bit equal to calculated parity value). This information being elaborated only for the last received data word of each data frame, both bit positions are 0 for data words that are not the last data word of a data frame or if the parity generation is disabled.

### 15.3.3.14 Receive Buffer Handling

If a receive FIFO buffer is available (CCFG.RB = 1) and enabled for data handling (RBCTR.SIZE > 0), it is recommended to set RBCTR.RCIM = $11_B$ in ASC mode. This leads to an indication that the data word has been the first data word of a new data frame if bit OUTR.RCI[0] = 1, a parity error is indicated by OUTR.RCI[4] = 1, and the received parity bit value is given by OUTR.RCI[3].

The standard receive buffer event and the alternative receive buffer event can be used for the following operations in RCI mode (RBCTR.RNM = 1):

- A standard receive buffer event indicates that a data word can be read from OUTR that has been received without parity error.
- An alternative receive buffer event indicates that a data word can be read from OUTR that has been received with parity error.

### 15.3.3.15 Sync-Break Detection

The receiver permanently checks the DIN0 signal for a certain number of consecutive bit times with 0 level. The number is given by the number of programmed bits per frame (SCTR.FLE) plus 2 (in the case without parity) or plus 3 (in the case with parity). If a 0 level is detected at a sample point of a bit after this event has been found, bit PSR.SBD is set and additionally, a protocol interrupt can be generated (if enabled by PCR.SBIEN = 1). The counting restarts from 0 each time a falling edge is found at input DIN0. This feature can be used for the detection of a synchronization break for slave devices in a LIN bus system (the master does not check for sync break).
For example, in a configuration for 8 data bits without parity generation, bit PCR.SBD is set after at the next sample point at 0 level after 10 complete bit times have elapsed (representing the sample point of the 11th bit time since the first falling edge).

### 15.3.3.16 Transfer Status Indication

The receiver status can be monitored by flag PSR[9] = BUSY if bit PCR.CTR[16] (receiver status enable RSTEN) is set. In this case, bit BUSY is set during a complete frame reception from the beginning of the start of frame bit to the end of the last stop bit.


Subject to Agreement on the Use of Product Information

The transmitter status can be monitored by flag PSR[9] = BUSY if bit PCR.CTR[17] (transmitter status enable TSTEN) is set. In this case, bit BUSY is set during a complete frame reception from the beginning of the start of frame bit to the end of the last stop bit. If both bits RSTEN and TSTEN are set, flag BUSY indicates the logical OR-combination of the receiver and the transmitter status. If both bits are cleared, flag BUSY is not modified depending on the transfer status (status changes are ignored).

### 15.3.4 ASC Protocol Registers

In ASC mode, the registers PCR and PSR handle ASC related information.

#### 15.3.4.1 ASC Protocol Control Register

In ASC mode, the PCR register bits or bit fields are defined as described in this section.

**PCR**
**Protocol Control Register [ASC Mode]**

**($3C_H$)** **Reset Value: $0000\ 0000_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MCLK | | | | | | | 0 | | | | | | | TSTEN | RSTEN |
| rw | | | | | | | r | | | | | | | rw | rw |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PL | | | SP | | | | | FFIEN | FEIEN | RNIEN | CDEN | SBIEN | IDM | STPB | SMD |
| rw | | | rw | | | | | rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **SMD** | 0 | rw | **Sample Mode**<br>This bit field defines the sample mode of the ASC receiver. The selected data input signal can be sampled only once per bit time or three times (in consecutive time quanta). When sampling three times, the bit value shifted in the receiver shift register is given by a majority decision among the three sampled values.<br>$0_B$ Only one sample is taken per bit time. The current input value is sampled.<br>$1_B$ Three samples are taken per bit time and a majority decision is made. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **STPB** | 1 | rw | **Stop Bits**<br>This bit defines the number of stop bits in an ASC frame.<br>$0_B$ The number of stop bits is 1.<br>$1_B$ The number of stop bits is 2. |
| **IDM** | 2 | rw | **Idle Detection Mode**<br>This bit defines if the idle detection is switched off or based on the frame length.<br>$0_B$ The bus idle detection is switched off and bits PSR.TXIDLE and PSR.RXIDLE are set automatically to enable data transfers without checking the inputs before.<br>$1_B$ The bus is considered as idle after a number of consecutive passive bit times defined by SCTR.FLE plus 2 (in the case without parity bit) or plus 3 (in the case with parity bit). |
| **SBIEN** | 3 | rw | **Synchronization Break Interrupt Enable**<br>This bit enables the generation of a protocol interrupt if a synchronization break is detected. The automatic detection is always active, so bit SBD can be set independently of SBIEN.<br>$0_B$ The interrupt generation is disabled.<br>$1_B$ The interrupt generation is enabled. |
| **CDEN** | 4 | rw | **Collision Detection Enable**<br>This bit enables the reaction of a transmitter to the collision detection.<br>$0_B$ The collision detection is disabled.<br>$1_B$ If a collision is detected, the transmitter stops its data transmission, outputs a 1, sets bit PSR.COL and generates a protocol interrupt.<br>In order to allow data transmission again, PSR.COL has to be cleared by software. |
| **RNIEN** | 5 | rw | **Receiver Noise Detection Interrupt Enable**<br>This bit enables the generation of a protocol interrupt if receiver noise is detected. The automatic detection is always active, so bit PSR.RNS can be set independently of PCR.RNIEN.<br>$0_B$ The interrupt generation is disabled.<br>$1_B$ The interrupt generation is enabled. |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **FEIEN** | 6 | rw | **Format Error Interrupt Enable**<br>This bit enables the generation of a protocol interrupt if a format error is detected. The automatic detection is always active, so bits PSR.FER0/FER1 can be set independently of PCR.FEIEN.<br>$0_B$ The interrupt generation is disabled.<br>$1_B$ The interrupt generation is enabled. |
| **FFIEN** | 7 | rw | **Frame Finished Interrupt Enable**<br>This bit enables the generation of a protocol interrupt if the receiver or the transmitter reach the end of a frame. The automatic detection is always active, so bits PSR.RFF or PSR.TFF can be set independently of PCR.FFIEN.<br>$0_B$ The interrupt generation is disabled.<br>$1_B$ The interrupt generation is enabled. |
| **SP** | [12:8] | rw | **Sample Point**<br>This bit field defines the sample point of the bit value. The sample point must not be located outside the programmed bit timing (PCR.SP ≤ BRG.DCTQ). |
| **PL** | [15:13] | rw | **Pulse Length**<br>This bit field defines the length of a 0 data bit, counted in time quanta, starting with the time quantum 0 of each bit time. Each bit value that is a 0 can lead to a 0 pulse that is shorter than a bit time, e.g. for IrDA applications. The length of a bit time is not changed by PL, only the length of the 0 at the output signal.<br>The pulse length must not be longer than the programmed bit timing (PCR.PL ≤ BRG.DCTQ).<br>This bit field is only taken into account by the transmitter and is ignored by the receiver.<br>$000_B$ The pulse length is equal to the bit length (no shortened 0).<br>$001_B$ The pulse length of a 0 bit is 2 time quanta.<br>$010_B$ The pulse length of a 0 bit is 3 time quanta.<br>...<br>$111_B$ The pulse length of a 0 bit is 8 time quanta. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **RSTEN** | 16 | rw | **Receiver Status Enable**<br>This bit enables the modification of flag PSR[9] = BUSY according to the receiver status.<br>$0_B$ Flag PSR[9] is not modified depending on the receiver status.<br>$1_B$ Flag PSR[9] is set during the complete reception of a frame. |
| **TSTEN** | 17 | rw | **Transmitter Status Enable**<br>This bit enables the modification of flag PSR[9] = BUSY according to the transmitter status.<br>$0_B$ Flag PSR[9] is not modified depending on the transmitter status.<br>$1_B$ Flag PSR[9] is set during the complete transmission of a frame. |
| **MCLK** | 31 | rw | **Master Clock Enable**<br>This bit enables the generation of the master clock MCLK.<br>$0_B$ The MCLK generation is disabled and the MCLK signal is 0.<br>$1_B$ The MCLK generation is enabled. |
| **0** | [30:18] | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

### 15.3.4.2 ASC Protocol Status Register

In ASC mode, the PSR register bits or bit fields are defined as described in this section. The bits and bit fields in register PSR are not cleared by hardware.

The flags in the PSR register can be cleared by writing a 1 to the corresponding bit position in register PSCR. Writing a 1 to a bit position in PSR sets the corresponding flag, but does not lead to further actions (no interrupt generation). Writing a 0 has no effect. The PSR flags should be cleared by software before enabling a new protocol.


Subject to Agreement on the Use of Product Information

**PSR**

**Protocol Status Register [ASC Mode] ($48_H$)** **Reset Value: $0000\ 0000_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AIF | RIF | TBIF | TSIF | DLIF | RSIF | BUSY | TFF | RFF | FER1 | FER0 | RNS | COL | SBD | RXIDLE | TXIDLE |
| rwh | rwh | rwh | rwh | rwh | rwh | r | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **TXIDLE** | 0 | rwh | **Transmission Idle**<br>This bit shows if the transmit line (DX1) has been idle. A frame transmission can only be started if TXIDLE is set.<br>$0_B$ The transmitter line has not yet been idle.<br>$1_B$ The transmitter line has been idle and frame transmission is possible. |
| **RXIDLE** | 1 | rwh | **Reception Idle**<br>This bit shows if the receive line (DX0) has been idle. A frame reception can only be started if RXIDLE is set.<br>$0_B$ The receiver line has not yet been idle.<br>$1_B$ The receiver line has been idle and frame reception is possible. |
| **SBD** | 2 | rwh | **Synchronization Break Detected[1)]**<br>This bit is set if a programmed number of consecutive bit values with level 0 has been detected (called synchronization break, e.g. in a LIN bus system).<br>$0_B$ A synchronization break has not yet been detected.<br>$1_B$ A synchronization break has been detected. |
| **COL** | 3 | rwh | **Collision Detected[1)]**<br>This bit is set if a collision has been detected (with PCR.CDEN = 1).<br>$0_B$ A collision has not yet been detected and frame transmission is possible.<br>$1_B$ A collision has been detected and frame transmission is not possible. |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **RNS** | 4 | rwh | **Receiver Noise Detected[1)]**<br>This bit is set if receiver noise has been detected.<br>$0_B$ Receiver noise has not been detected.<br>$1_B$ Receiver noise has been detected. |
| **FER0** | 5 | rwh | **Format Error in Stop Bit 0[1)]**<br>This bit is set if a 0 has been sampled in the stop bit 0 (called format error 0).<br>$0_B$ A format error 0 has not been detected.<br>$1_B$ A format error 0 has been detected. |
| **FER1** | 6 | rwh | **Format Error in Stop Bit 1[1)]**<br>This bit is set if a 0 has been sampled in the stop bit 1 (called format error 1).<br>$0_B$ A format error 1 has not been detected.<br>$1_B$ A format error 1 has been detected. |
| **RFF** | 7 | rwh | **Receive Frame Finished[1)]**<br>This bit is set if the receiver has finished the last stop bit.<br>$0_B$ The received frame is not yet finished.<br>$1_B$ The received frame is finished. |
| **TFF** | 8 | rwh | **Transmitter Frame Finished[1)]**<br>This bit is set if the transmitter has finished the last stop bit.<br>$0_B$ The transmitter frame is not yet finished.<br>$1_B$ The transmitter frame is finished. |
| **BUSY** | 9 | r | **Transfer Status BUSY**<br>This bit indicates the receiver status (if PCR.RSTEN = 1) or the transmitter status (if PCR.TSTEN = 1) or the logical OR combination of both (if PCR.RSTEN = PCR.TSTEN = 1).<br>$0_B$ A data transfer does not take place.<br>$1_B$ A data transfer currently takes place. |
| **RSIF** | 10 | rwh | **Receiver Start Indication Flag**<br>$0_B$ A receiver start event has not occurred.<br>$1_B$ A receiver start event has occurred. |
| **DLIF** | 11 | rwh | **Data Lost Indication Flag**<br>$0_B$ A data lost event has not occurred.<br>$1_B$ A data lost event has occurred. |
| **TSIF** | 12 | rwh | **Transmit Shift Indication Flag**<br>$0_B$ A transmit shift event has not occurred.<br>$1_B$ A transmit shift event has occurred. |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **TBIF** | 13 | rwh | **Transmit Buffer Indication Flag**<br>$0_B$ A transmit buffer event has not occurred.<br>$1_B$ A transmit buffer event has occurred. |
| **RIF** | 14 | rwh | **Receive Indication Flag**<br>$0_B$ A receive event has not occurred.<br>$1_B$ A receive event has occurred. |
| **AIF** | 15 | rwh | **Alternative Receive Indication Flag**<br>$0_B$ An alternative receive event has not occurred.<br>$1_B$ An alternative receive event has occurred. |
| **BRGIF** | 16 | rwh | **Baud Rate Generator Indication Flag**<br>$0_B$ A baud rate generator event has not occurred.<br>$1_B$ A baud rate generator event has occurred. |
| **0** | [31:17] | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

1) This status bit can generate a protocol interrupt (see **Page 15-22**). The general interrupt status flags are described in the general interrupt chapter.

### 15.3.5 Hardware LIN Support

In order to support the LIN protocol, bit TCSR.FLEMD = 1 should be set for the master. For slave devices, it can be cleared and the fixed number of 8 data bits has to be set (SCTR.FLE = $7_H$). For both, master and slave devices, the parity generation has to be switched off (CCR.PM = $00_B$) and transfers take place with LSB first (SCTR.SDIR = 0) and 1 stop bit (PCR.STPB = 0).

The Local Interconnect Network (LIN) data exchange protocol contains several symbols that can all be handled in ASC mode. Each single LIN symbol represents a complete ASC frame. The LIN bus is a master-slave bus system with a single master and multiple slaves (for the exact definition please refer to the official LIN specification).

A complete LIN frame contains the following symbols:

- Synchronization break:
  The master sends a synchronization break to signal the beginning of a new frame. It contains at least 13 consecutive bit times at 0 level, followed by at least one bit time at 1 level (corresponding to 1 stop bit). Therefore, TBUF11 if the transmit buffer is used, (or IN11 if the FIFO buffer is used) has to be written with 0 (leading to a frame with SOF followed by 12 data bits at 0 level).
  A slave device shall detect 11 consecutive bit times at 0 level, which done by the synchronization break detection. Bit PSR.SBD is set if such an event is detected and a protocol interrupt can be generated. Additionally, the received data value of 0 appears in the receive buffer and a format error is signaled.

If the baud rate of the slave has to be adapted to the master, the baud rate measurement has to be enabled for falling edges by setting BRG.TMEN = 1, DX0CR.CM = $10_H$ and DX1CR.CM = $00_H$ before the next symbol starts.

- Synchronization byte:
  The master sends this symbol after writing the data value $55_H$ to TBUF07 (or IN07). A slave device can either receive this symbol without any further action (and can discard it) or it can use the falling edges for baud rate measurement. Bit PSR.TSIF = 1 (with optionally the corresponding interrupt) indicates the detection of a falling edge and the capturing of the elapsed time since the last falling edge in CMTR.CTV. Valid captured values can be read out after the second, third, fourth and fifth activation of TSIF. After the fifth activation of TSIF within this symbol, the baud rate detection can be disabled (BRG.TMEN = 0) and BRG.PDIV can be programmed with the captured CMTR.CTV value divided by twice the number of time quanta per bit (assuming BRG.PCTQ = $00_B$).
- Other symbols:
  The other symbols of a LIN frame can be handled with ASC data frames without specific actions.

If LIN frames should be sent out on a frame base by the LIN master, the input DX2 can be connected to external timers to trigger the transmit actions (e.g. the synchronization break symbol has been prepared but is started if a trigger occurs). Please note that during the baud rate measurement of the ASC receiver, the ASC transmitter of the same USIC channel can still perform a transmission.


Subject to Agreement on the Use of Product Information

## 15.4 Synchronous Serial Channel (SSC)

The synchronous serial channel SSC covers the data transfer function of an SPI-like module. It can handle reception and transmission of synchronous data frames between a device operating in master mode and at least one device in slave mode. Besides the standard SSC protocol consisting of one input and one output data line, SSC protocols with two (Dual-SSC) or four (Quad-SSC) input/output data lines are also supported. The SSC mode is selected by CCR.MODE = $0001_B$ with CCFG.SSC = 1 (SSC mode is available).

### 15.4.1 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).

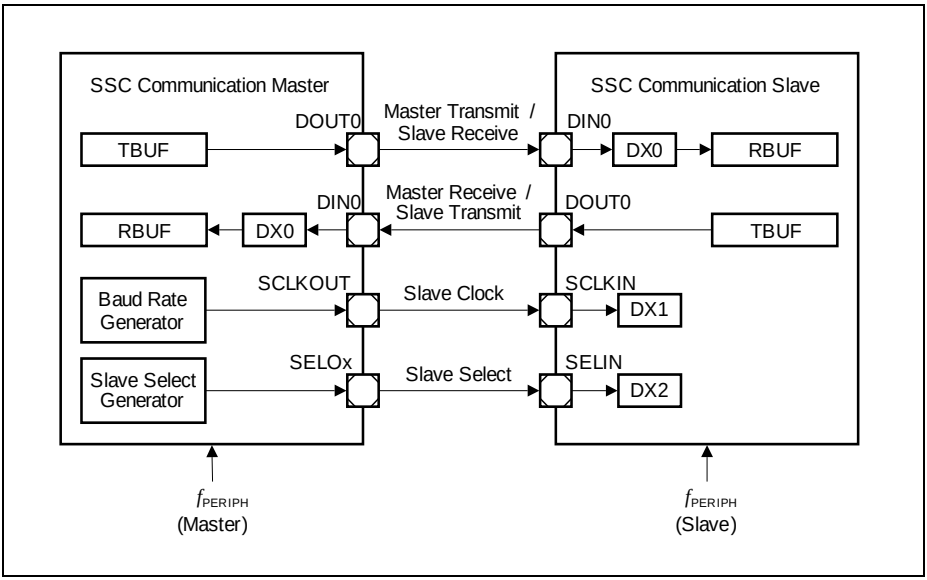



**Figure 15-35 SSC Signals for Standard Full-Duplex Communication**

In order to explicitly indicate the start and the end of a data transfer and to address more than one slave devices individually, the SSC module supports the handling of slave select signals. They are optional and are not necessarily needed for SSC data transfers. The SSC module supports up to 8 different slave select output signals for master mode operation (named SELOx, with x = 0-7) and 1 slave select input SELIN for slave mode. In most applications, the slave select signals are active low.

A device operating in master mode controls the start and end of a data frame, as well as the generation of the shift clock and slave select signals. This comprises the baud rate setting for the shift clock and the delays between the shift clock and the slave select output signals. If several SSC modules are connected together, there can be only one SSC master at a time, but several slaves. Slave devices receive the shift clock and optionally a slave select signal(s). For the programming of the input stages DXn please refer to **Page 15-22**.

**Table 15-12 SSC Communication Signals**

| SSC Mode | Receive Data | Transmit Data | Shift Clock | Slave Select(s) |
|---|---|---|---|---|
| Standard SSC Master | MRST[1], input DIN0, handled by DX0 | MTSR[2], Output DOUT0 | Output SCLKOUT | Output(s) SELOx |
| Standard SSC Slave | MTSR, input DIN0, handled by DX0 | MRST, Output DOUT0 | Input SCLKIN, handled by DX1 | input SELIN, handled by DX2 |
| Dual-SSC Master | MRST[1:0], input DIN[1:0], handled by DX0 and DX3 | MTSR[1:0], Output DOUT[1:0] | Output SCLKOUT | Output(s) SELOx |
| Dual-SSC Slave | MTSR[1:0], input DIN[1:0], handled by DX0 and DX3 | MRST[1:0], Output DOUT[1:0] | Input SCLKIN, handled by DX1 | input SELIN, handled by DX2 |
| Quad-SSC Master | MRST[3:0], input DIN[3:0], handled by DX0, DX3, DX4 and DX5 | MTSR[3:0], Output DOUT[3:0] | Output SCLKOUT | Output(s) SELOx |
| Quad-SSC Slave | MTSR[3:0], input DIN[3:0], handled by DX0, DX3, DX4 and DX5 | MRST[3:0], Output DOUT[3:0] | Input SCLKIN, handled by DX1 | input SELIN, handled by DX2 |

1) MRST = master receive slave transmit, also known as MISO = master in slave out

2) MTSR = master transmit slave receive, also known as MOSI = master out slave in


Subject to Agreement on the Use of Product Information

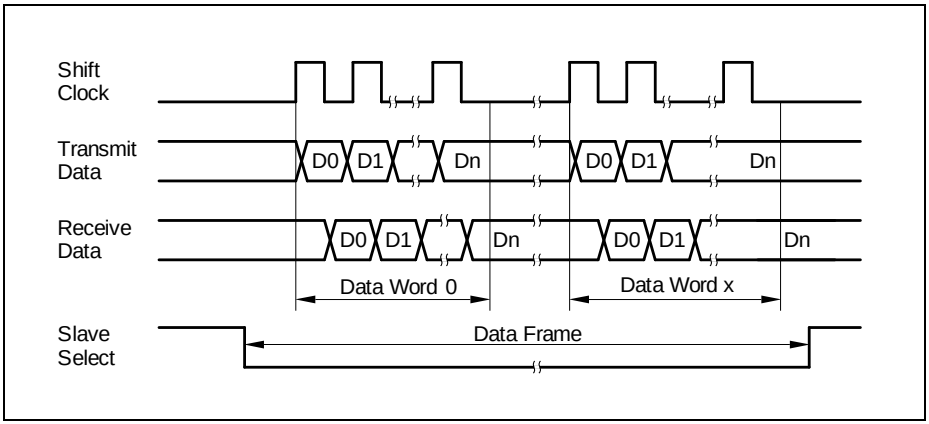



**Figure 15-36 4-Wire SSC Standard Communication Signals**

### 15.4.1.1 Transmit and Receive Data Signals

In standard SSC half-duplex mode, a single data line is used, either for data transfer from the master to a slave or from a slave to the master. In this case, MRST and MTSR are connected together, one signal as input, the other one as output, depending on the data direction. The user software has to take care about the data direction to avoid data collision (e.g. by preparing dummy data of all 1s for transmission in case of a wired AND connection with open-drain drivers, by enabling/disabling push/pull output drivers or by switching pin direction with hardware port control enabled). In full-duplex mode, data transfers take place in parallel between the master device and a slave device via two independent data signals MTSR and MRST, as shown in **Figure 15-35**.

The receive data input signal DIN0 is handled by the input stage DX0. In master mode (referring to MRST) as well as in slave mode (referring to MTSR), the data input signal DIN0 is taken from an input pin. The signal polarity of DOUT0 (data output) with respect to the data bit value can be configured in block DOCFG (data output configuration) by bit field SCTR.DOCFG.

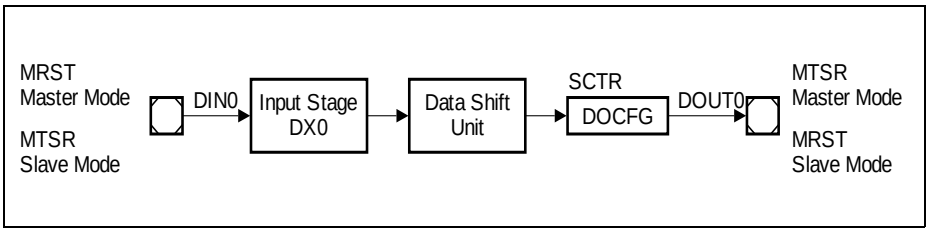



**Figure 15-37 SSC Data Signals**


Subject to Agreement on the Use of Product Information

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.

### 15.4.1.2 Shift Clock Signals

The shift clock signal is handled by the input stage DX1. In slave mode, the signal SCLKIN is received from an external master, so the DX1 stage has to be connected to an input pin. The input stage can invert the received input signal to adapt to the polarity of SCLKIN to the function of the data shift unit (data transmission on rising edges, data reception on falling edges).

In master mode, the shift clock is generated by the internal baud rate generator. The output signal SCLK of the baud rate generator is taken as shift clock input for the data shift unit. The internal signal SCLK is made available for external slave devices by signal SCLKOUT. For complete closed loop delay compensation in a slave mode, SCLKOUT can also take the transmit shift clock from the input stage DX1. The selection is done through the bit BRG.SCLKOSEL. See **Section 15.4.6.3**.

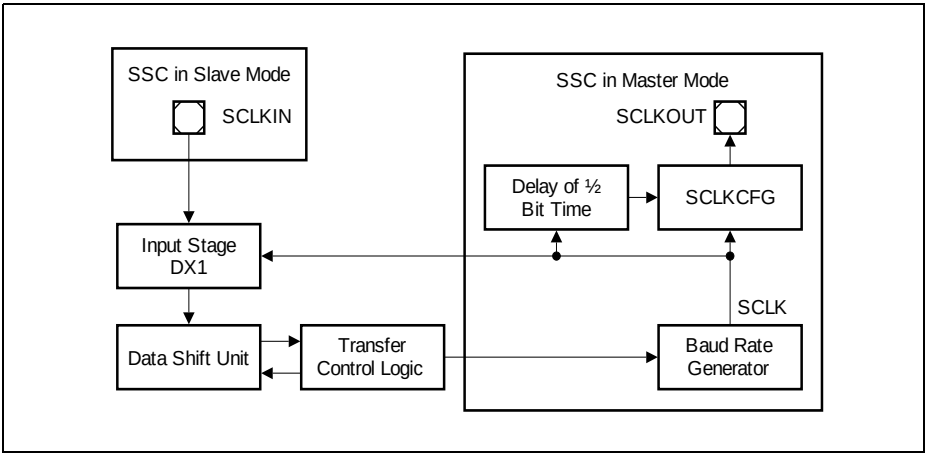



**Figure 15-38 SSC Shift Clock Signals**

Due to the multitude of different SSC applications, in master mode, there are different ways to configure the shift clock output signal SCLKOUT with respect to SCLK. This is done in the block SCLKCFG (shift clock configuration) by bit field BRG.SCLKCFG, allowing 4 possible settings, as shown in **Figure 15-39**.

- No delay, no polarity inversion (SCLKCFG = $00_B$, SCLKOUT equals SCLK):
  The inactive level of SCLKOUT is 0, while no data frame is transferred. The first data bit of a new data frame is transmitted with the first rising edge of SCLKOUT and the first data bit is received in with the first falling edge of SCLKOUT. The last data bit of a data frame is transmitted with the last rising clock edge of SCLKOUT and the last

data bit is received in with the last falling edge of SCLKOUT. This setting can be used in master and in slave mode. It corresponds to the behavior of the internal data shift unit.

- No delay, polarity inversion (SCLKCFG = $01_B$):
  The inactive level of SCLKOUT is 1, while no data frame is transferred. The first data bit of a new data frame is transmitted with the first falling clock edge of SCLKOUT and the first data bit is received with the first rising edge of SCLKOUT. The last data bit of a data frame is transmitted with the last falling edge of SCLKOUT and the last data bit is received with the last rising edge of SCLKOUT.
  This setting can be used in master and in slave mode. For slave mode, bit field DX1CR.DPOL = $1_B$ has to be programmed.
- SCLKOUT is delayed by 1/2 shift clock period, no polarity inversion (SCLKCFG = $10_B$):
  The inactive level of SCLKOUT is 0, while no data frame is transferred.
  The first data bit of a new data frame is transmitted 1/2 shift clock period before the first rising clock edge of SCLKOUT. Due to the delay, the next data bits seem to be transmitted with the falling edges of SCLKOUT. The last data bit of a data frame is transmitted 1/2 period of SCLKOUT before the last rising clock edge of SCLKOUT. The first data bit is received 1/2 shift clock period before the first falling edge of SCLKOUT. Due to the delay, the next data bits seem to be received with the rising edges of SCLKOUT. The last data bit is received 1/2 period of SCLKOUT before the last falling clock edge of SCLKOUT.
  This setting can be used only in master mode and not in slave mode (the connected slave has to provide the first data bit before the first SCLKOUT edge, e.g. as soon as it is addressed by its slave select).
- SCLKOUT is delayed by 1/2 shift clock period, polarity inversion (SCLKCFG = $11_B$):
  The inactive level of SCLKOUT is 1, while no data frame is transferred.
  The first data bit of a new data frame is transmitted 1/2 shift clock period before the first falling clock edge of SCLKOUT. Due to the delay, the next data bits seem to be transmitted with the rising edges of SCLKOUT. The last data bit of a data frame is transmitted 1/2 period of SCLKOUT before the last falling clock edge of SCLKOUT. The first data bit is received 1/2 shift clock period before the first rising edge of SCLKOUT. Due to the delay, the next data bits seem to be received with the falling edges of SCLKOUT. The last data bit is received 1/2 period of SCLKOUT before the last rising clock edge of SCLKOUT.
  This setting can be used only in master mode and not in slave mode (the connected slave has to provide the first data bit before the first SCLKOUT edge, e.g. as soon as it is addressed by its slave select).


Subject to Agreement on the Use of Product Information

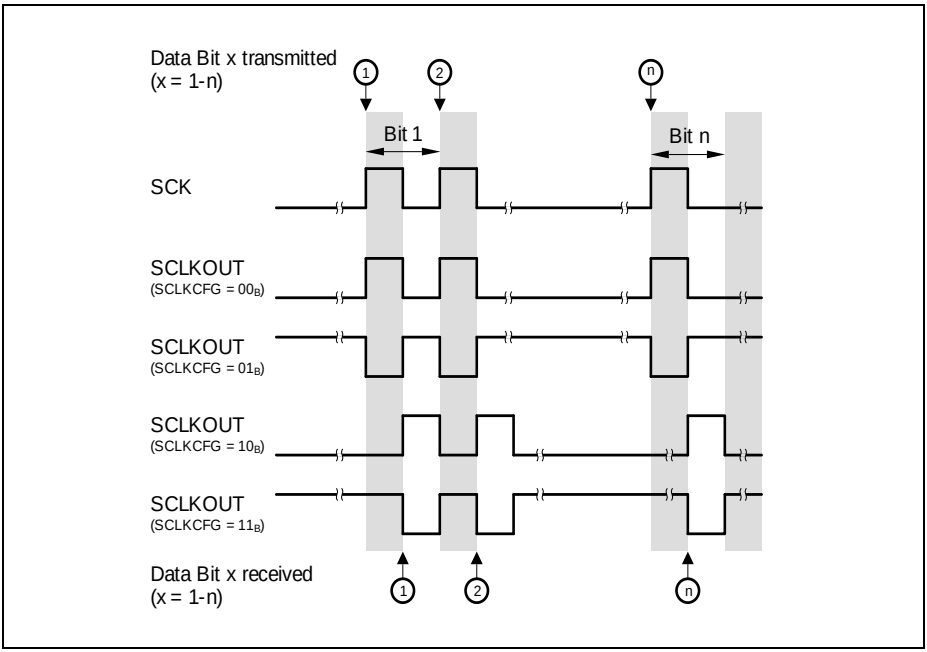



**Figure 15-39 SCLKOUT Configuration in SSC Master Mode**

*Note: If a configuration with delay is selected and a slave select line is used, the slave select delays have to be set up accordingly.*

In SSC slave mode, the bit PCR.SLPHSEL can be used to configure the clock phase of the data shift.

- When SLPHSEL = $0_B$, the slave SSC transmits data bits with each leading edge of the selected shift clock input (SCLKIN) and receives data bits with each trailing edge of SCLKIN
- When SLPHSEL = $1_B$, the slave SSC transmits the first data bit once the selected slave select input (SELIN) becomes active. If SELIN is not used, the DX2 stage has to deliver a 1-level to the data shift unit to shift out the first bit. Subsequent data bits are then transmitted with each trailing edge of SCLKIN. The SSC slave receives all data bits with each leading edge of SCLKIN.

For both settings, the clock polarity is determined by bit 0 of SCLKCFG.

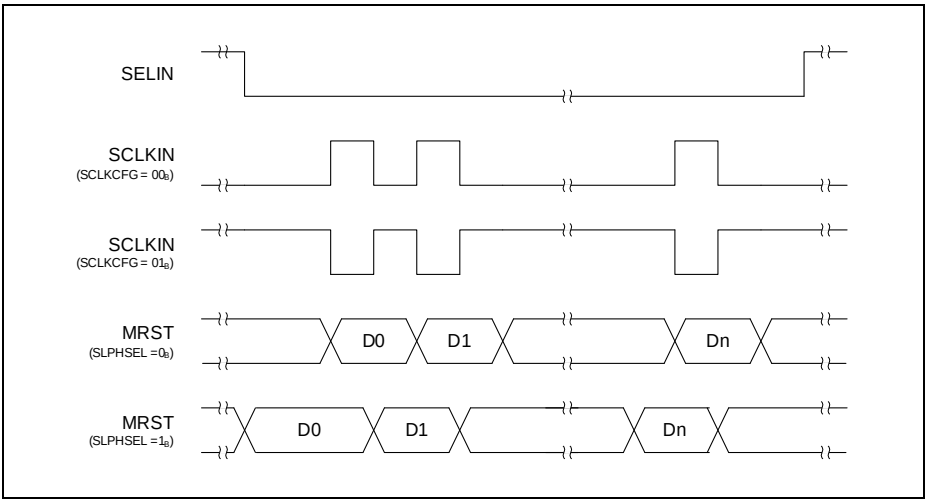



**Figure 15-40 SLPHSEL Configuration in SSC Slave Mode**

### 15.4.1.3 Slave Select Signals

The slave select signal is handled by the input stage DX2. In slave mode, the input signal SELIN is received from an external master via an input pin. The input stage can invert the received input signal to adapt the polarity of signal SELIN to the function of the data shift unit (the module internal signals are considered as high active, so a data transfer is only possible while the slave select input of the data shift unit is at 1-level, otherwise, shift clock pulses are ignored and do not lead to data transfers). If an input signal SELIN is low active, it should be inverted in the DX2 input stage.

In master mode, a master slave select signal MSLS is generated by the internal slave select generator. In order to address different external slave devices independently, the internal MSLS signal is made available externally via up to 8 SELOx output signals that can be configured by the block SELCFG (select configuration).

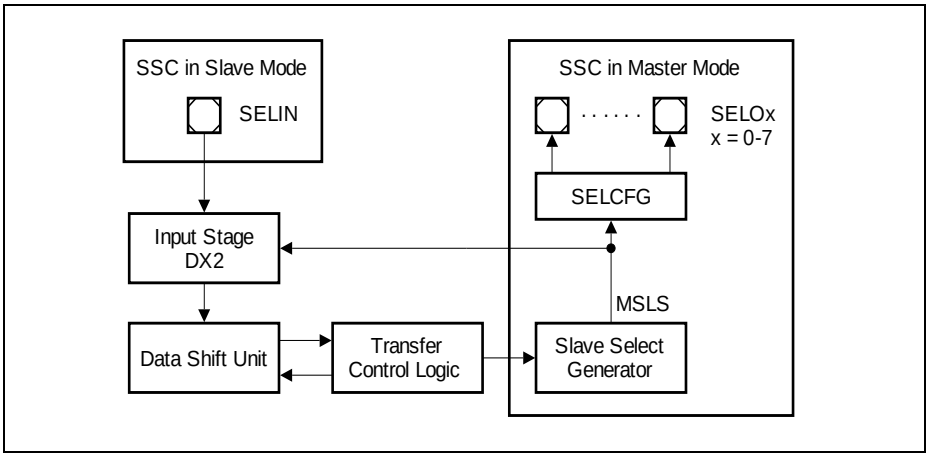



**Figure 15-41 SSC Slave Select Signals**

The control of the SELCFG block is based on protocol specific bits and bit fields in the protocol control register PCR. For the generation of the MSLS signal please refer to **Section 15.4.3.2**.

- PCR.SELCTR to chose between direct and coded select mode
- PCR.SELINV to invert the SELOx outputs
- PCR.SELO[7:0] as individual value for each SELOx line

The SELCFG block supports the following configurations of the SELOx output signals:

- Direct Select Mode (SELCTR = 1):
  Each SELOx line (with x = 0-7) can be directly connected to an external slave device. If bit x in bit field SELO is 0, the SELOx output is permanently inactive. A SELOx output becomes active while the internal signal MSLS is active (see **Section 15.4.3.2**) and bit x in bit field SELO is 1. Several external slave devices can be addressed in parallel if more than one bit in bit field SELO are set during a data frame. The number of external slave devices that can be addressed individually is limited to the number of available SELOx outputs.
- Coded Select Mode (SELCTR = 0):
  The SELOx lines (with x = 1-7) can be used as addresses for an external address decoder to increase the number of external slave devices. These lines only change with the start of a new data frame and have no other relation to MSLS. Signal SELO0 can be used as enable signal for the external address decoder. It is active while MSLS is active (during a data frame) and bit 0 in bit field SELO is 1. Furthermore, in coded select mode, this output line is delayed by one cycle of $f_{PERIPH}$ compared to MSLS to allow the other SELOx lines to stabilize before enabling the address decoder.

### 15.4.2 Operating the SSC

This chapter contains SSC issues, that are of general interest and not directly linked to either master mode or slave mode.

#### 15.4.2.1 Automatic Shadow Mechanism

The contents of the baud rate control register BRG, bit fields SCTR.FLE as well as the protocol control register PCR are internally kept constant while a data frame is transferred (= while MSLS is active) by an automatic shadow mechanism. The registers can be programmed all the time with new settings that are taken into account for the next data frame. During a data frame, the applied (shadowed) setting is not changed, although new values have been written after the start of the data frame.

Bit fields SCTR.WLE, SCTR.DSM, SCTR.HPCDIR and SCTR.SDIR are shadowed automatically with the start of each data word. As a result, a data frame can consist of data words with a different length, or data words that are transmitted or received through different number of data lines. It is recommended to change SCTR.SDIR only when no data frame is running to avoid interference between hardware and software.

Please note that the starting point of a data word are different for a transmitter (first bit transmitted) and a receiver (first bit received). In order to ensure correct handling, it is recommended to refer to the receive start interrupt RSI before modifying SCTR.WLE. If TCSR.WLEMD = 1, it is recommended to update TCSR and TBUFxx after the receiver start interrupt has been generated.

#### 15.4.2.2 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.


Subject to Agreement on the Use of Product Information

### 15.4.2.3 Disabling SSC Mode

In order to disable SSC mode without any data corruption, the receiver and the transmitter have to be both idle. This is ensured by requesting Stop Mode 1 in register KSCFG. After Stop Mode 1 has been acknowledged by KSCFG.2 = 1, the SSC mode can be disabled.

### 15.4.2.4 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.

### 15.4.2.5 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.


Subject to Agreement on the Use of Product Information

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 15-42** shows how a parity bit is added to the transmitted data bits of a frame. The number of the transmitted bits of a complete frame with parity is always one more than that without parity. The parity bit is transmitted as the last bit of a frame, following the data bits, independent of the shift direction (SCTR.SDIR).

*Note: For dual and quad SSC protocols, the parity bit will be transmitted and received only on DOUT0 and DX0 respectively in the extended clock cycle.*




**Figure 15-42 Data Frames without/with Parity**

Similarly, after the receiver receives the last word of a data frame as defined by FLE, it expects an additional one clock cycle, which will contain the parity bit. The receiver interprets this bit as received parity and separates it from the received data. The received parity bit value is instead monitored in the protocol-related argument (PAR) of the receiver buffer status registers as receiver buffer status information. The receiver compares the bit to its internally calculated parity and the result of the parity check is indicated by the flag PSR.PARERR. The parity error event generates a protocol interrupt if PCR.PARIEN = 1.

Parity bit generation and detection is not supported for the following cases:

- When frame length is 64 data bits or greater, i.e. FLE = $63_H$;
- When in slave mode, the end of frame occurs before the number of data bits defined by FLE is reached.

### 15.4.2.6 Transfer Mode

In SSC mode, bit field SCTR.TRM = $01_B$ has to be programmed to allow data transfers. Setting SCTR.TRM = $00_B$ disables and stops the data transfer immediately.

### 15.4.2.7 Data Transfer Interrupt Handling

The data transfer interrupts indicate events related to SSC frame handling.

- Transmit buffer interrupt TBI:
  Bit PSR.TBIF is set after the start of first data bit of a data word.
- Transmit shift interrupt TSI:
  Bit PSR.TSIF is set after the start of the last data bit of a data word.
- Receiver start interrupt RSI:
  Bit PSR.RSIF is set after the reception of the first data bit of a data word.
  With this event, bit TCSR.TDV is cleared and new data can be loaded to the transmit buffer.
- Receiver interrupt RI:
  The reception of the second, third, and all subsequent words in a multi-word frame is always indicated by RBUFSR.SOF = 0. Bit PSR.RIF is set after the reception of the last data bit of a data word if RBUFSR.SOF = 0.
  Bit RBUFSR.SOF indicates whether the received data word has been the first data word of a multi-word frame or some subsequent word. In SSC mode, it decides if alternative interrupt or receive interrupt is generated.
- Alternative interrupt AI:
  The reception of the first word in a frame is always indicated by RBUFSR.SOF = 1. This is true both in case of reception of multi-word frames and single-word frames. In SSC mode, this results in setting PSR.AIF.


Subject to Agreement on the Use of Product Information

#### 15.4.2.8 Baud Rate Generator Interrupt Handling

The baud rate generator interrupt indicate that the capture mode timer has reached its maximum value. With this event, the bit PSR.BRGIF is set.

#### 15.4.2.9 Protocol-Related Argument and Error

The protocol-related argument (RBUFSR.PAR) and the protocol-related error (RBUFSR.PERR) are two flags that are assigned to each received data word in the corresponding receiver buffer status registers.

In SSC mode, the received parity bit is monitored by the protocol-related argument. The received start of frame indication is monitored by the protocol-related error indication (0 = received word is not the first word of a frame, 1 = received word is the first word of a new frame).

*Note: For SSC, the parity error event indication bit is located in the PSR register.*

#### 15.4.2.10 Receive Buffer Handling

If a receive FIFO buffer is available (CCFG.RB = 1) and enabled for data handling (RBCTR.SIZE > 0), it is recommended to set RBCTR.RCIM = $01_B$ in SSC mode. This leads to an indication that the data word has been the first data word of a new data frame if bit OUTR.RCI[4] = 1, and the word length of the received data is given by OUTR.RCI[3:0].

The standard receive buffer event and the alternative receive buffer event can be used for the following operation in RCI mode (RBCTR.RNM = 1):

- A standard receive buffer event indicates that a data word can be read from OUTR that has not been the first word of a data frame.
- An alternative receive buffer event indicates that the first data word of a new data frame can be read from OUTR.

#### 15.4.2.11 Multi-IO SSC Protocols

The SSC implements the following three features to support multiple data input/output SSC protocols, such as the dual- and quad-SSC:

1. Data Shift Mode (**Section 15.2.5.2**)
   Configures the data for transmission and reception using one, two or four data lines in parallel, through the bit field SCTR.DSM.
2. Hardware Port Control (**Section 15.2.7**)
   Sets up a dedicated hardware interface to control the direction of the pins overlaid with both DINx and DOUTx functions, through the bit SCTR.HPCDIR.
3. Transmit Control Information (**Section 15.2.5.3**)
   Allows the dynamic control of both the shift mode and pin direction during data transfers by writing to SCTR.DSM and SCTR.HPCDIR with TCI.


Subject to Agreement on the Use of Product Information

**Figure 15-43** shows an example of a Quad-SSC protocol, which requires the master SSC to first transmit a command byte (to request a quad output read from the slave) and a dummy byte through a single data line. At the end of the dummy byte, both master and slave SSC switches to quad data lines, and with the roles of transmitter and receiver reversed. The master SSC then receives the data four bits per shift clock from the slave through the MRST[3:0] lines.

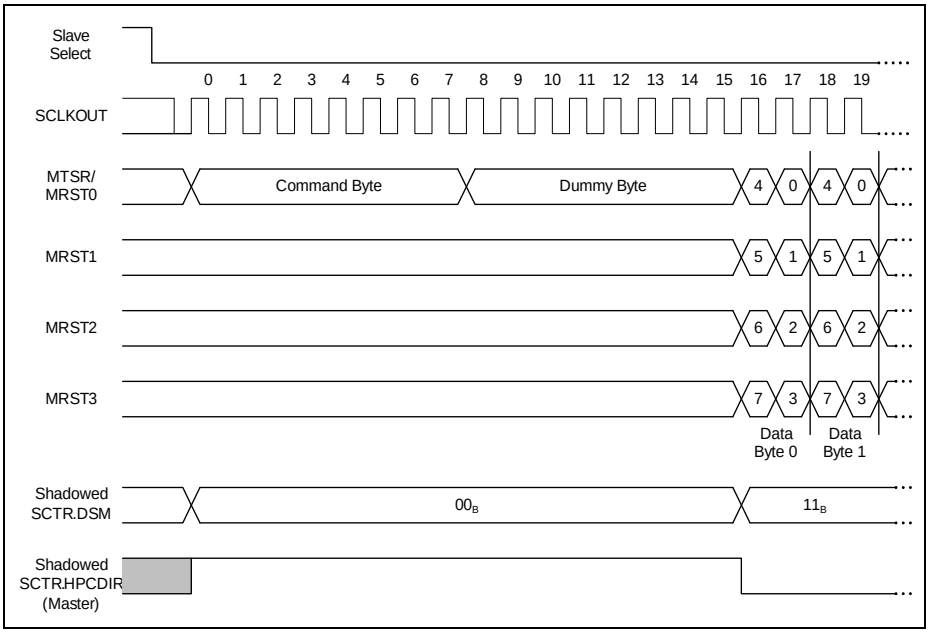



**Figure 15-43 Quad-SSC Example**

To work with the quad-SSC protocol in the given example, the following issues have to be additionally considered on top of those defined in **Section 15.4.3** and **Section 15.4.4**:

- During the initialization phase:
  - Set CCR.HPCEN to $11_B$ to enable the dedicated hardware interface to the DX0/DOUT0, DX3/DOUT1, DX4/DOUT2 and DX5/DOUT3 pins.
  - Set TCSR.[4:0] to $10_H$ to enable hardware port control in TCI
- To start the data transfer:
  - For the master SSC, write the command and dummy bytes into TBUF04 to select a single data line in output mode and initiate the data transfer.
  - For the slave SSC, dummy data can be preloaded into TBUF00 to select a single data line in input mode.
- To switch to quad data lines and pin direction:

– For the master SSC, write subsequent dummy data to TBUF03 to select quad data lines in input mode to read in valid slave data.
– For the slave SSC, write valid data to TBUF07 for transmission through quad data lines in output mode.

*Note: If DOUT[1:0] or DOUT[3:0] pins are already enabled with CCR.HPCEN but SCTR.DSM does not use all pins, the unused DOUTx pins output the passive data level defined by SCTR.PDL.*

**Figure 15-44** shows the connections for the Quad-SSC example.

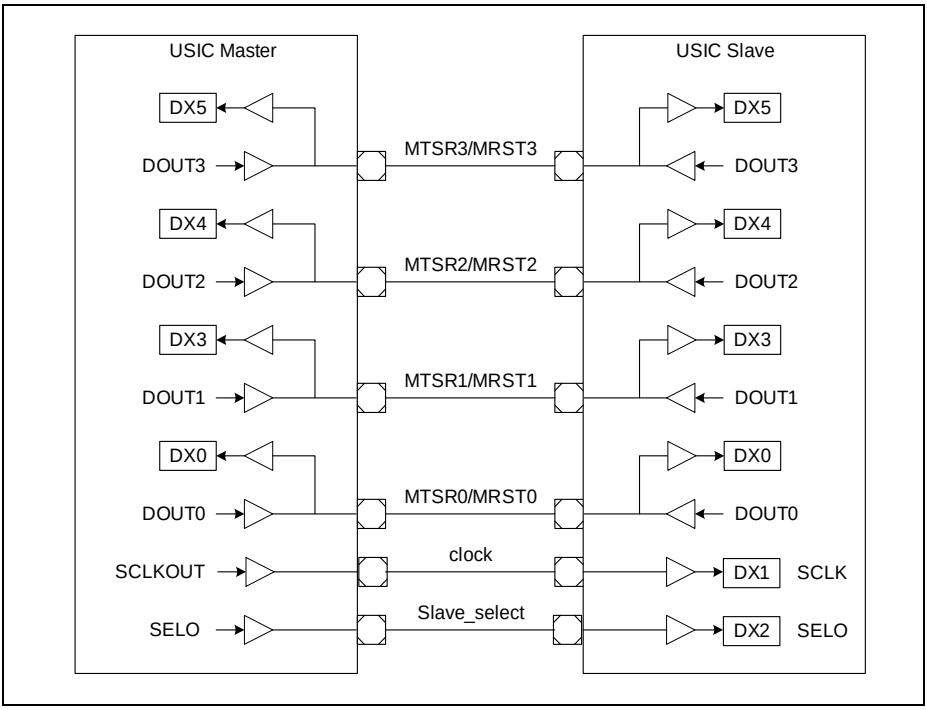



**Figure 15-44 Connections for Quad-SSC Example**

## 15.4.3 Operating the SSC in Master Mode

In order to operate the SSC in master mode, the following issues have to be considered:

- Select SSC mode:
  It is recommended to configure all parameters of the SSC that do not change during run time while CCR.MODE = $0000_B$. Bit field SCTR.TRM = $01_B$ has to be programmed. The configuration of the input stages has to be done while

CCR.MODE = $0000_B$ to avoid unintended edges of the input signals and the SSC mode can be enabled by CCR.MODE = $0001_B$ afterwards.

- Pin connections:
  Establish a connection of the input stage (DX0, DX3, DX4, DX5) with the selected receive data input pin (DIN[3:0]) with DXnCR.INSW = 1 and configure the transmit data output pin (DOUT[3:0]). One, two or four such connections may be needed depending on the protocol. For half-duplex configurations, hardware port control can be also used to establish the required connections.
- Baud rate generation:
  The desired baud rate setting has to be selected, comprising the fractional divider and the baud rate generator. Bit DX1CR.INSW = 0 has to be programmed to use the baud rate generator output SCLK directly as input for the data shift unit. Configure a shift clock output pin (signal SCLKOUT).
- Slave select generation:
  The slave select delay generation has to be enabled by setting PCR.MSLSEN = 1 and the programming of the time quanta counter setting. Bit DX2CR.INSW = 0 has to be programmed to use the slave select generator output MSLS as input for the data shift unit. Configure slave select output pins (signals SELOx) if needed.
- Data format configuration:
  The word length, the frame length, the shift direction and shift mode have to be set up according to the application requirements by programming the register SCTR.

*Note: The USIC can only receive in master mode if it is transmitting, because the master frame handling refers to bit TDV of the transmitter part.*

*Note: The step to enable the alternate output port functions should only be done after the SSC mode is enabled, to avoided unintended spikes on the output.*

### 15.4.3.1 Baud Rate Generation

The baud rate (determining the length of one data bit) of the SSC is defined by the frequency of the SCLK signal (one period of $f_{SCLK}$ represents one data bit). The SSC baud rate generation does not imply any time quanta counter.

In a standard SSC application, the phase relation between the optional MCLK output signal and SCLK is not relevant and can be disabled (BRG.PPPEN = 0). In this case, the SCLK signal directly derives from the protocol input frequency $f_{PIN}$. In the exceptional case that a fixed phase relation between the MCLK signal and SCLK is required (e.g. when using MCLK as clock reference for external devices), the additional divider by 2 stage has to be taken into account (BRG.PPPEN = 1).


Subject to Agreement on the Use of Product Information

The adjustable divider factor is defined by bit field BRG.PDIV.

$$f_{SCLK} = \frac{f_{PIN}}{2} \times \frac{1}{PDIV + 1} \quad \text{if PPPEN = 0}$$
$$f_{SCLK} = \frac{f_{PIN}}{2 \times 2} \times \frac{1}{PDIV + 1} \quad \text{if PPPEN = 1} \qquad (15.8)$$

### 15.4.3.2 MSLS Generation

The slave select signals indicate the start and the end of a data frame and are also used by the communication master to individually select the desired slave device. A slave select output of the communication master becomes active a programmable time before a data part of the frame is started (leading delay $T_{ld}$), necessary to prepare the slave device for the following communication. After the transfer of a data part of the frame, it becomes inactive again a programmable time after the end of the last bit (trailing delay $T_{td}$) to respect the slave hold time requirements. If data frames are transferred back-to-back one after the other, the minimum time between the deactivation of the slave select and the next activation of a slave select is programmable (next-frame delay $T_{nf}$). If a data frame consists of more than one data word, an optional delay between the data words can also be programmed (inter-word delay $T_{iw}$).

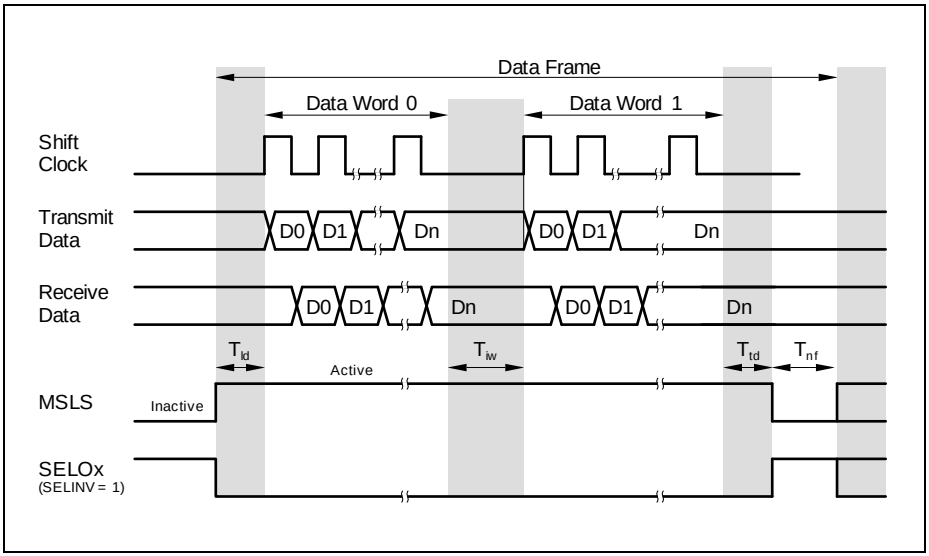



**Figure 15-45 MSLS Generation in SSC Master Mode**

In SSC master mode, the slave select delays are defined as follows:


Subject to Agreement on the Use of Product Information

- 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.

#### 15.4.3.3 Automatic Slave Select Update

If the number of bits per SSC frame and the word length are defined by bit fields SCTR.FLE and SCTR.WLE, the transmit control information TCI can be used to update the slave select setting PCR.CTR[23:16] to control the SELOx select outputs. The automatic update mechanism is enabled by TCSR.SELMD = 1 (bits TCSR.WLEMD, FLEMD, and WAMD have to be cleared). In this case, the TCI of the first data word of a frame defines the slave select setting of the complete frame due to the automatic shadow mechanism (see **Page 15-62**).


Subject to Agreement on the Use of Product Information

### 15.4.3.4 Slave Select Delay Generation

The slave select delay generation is based on time quanta. The length of a time quantum (defined by the period of the $f_{CTQIN}$) and the number of time quanta per delay can be programmed.

In standard SSC applications, the leading delay $T_{ld}$ and the trailing delay $T_{td}$ are mainly used to ensure stability on the input and output lines as well as to respect setup and hold times of the input stages. These two delays have the same length (in most cases shorter than a bit time) and can be programmed with the same set of bit fields.

- BRG.CTQSEL
  to define the input frequency $f_{CTQIN}$ for the time quanta generation for $T_{ld}$ and $T_{td}$
- BRG.PCTQ
  to define the length of a time quantum (division of $f_{CTQIN}$ by 1, 2, 3, or 4) for $T_{ld}$ and $T_{td}$
- BRG.DCTQ
  to define the number of time quanta for the delay generation for $T_{ld}$ and $T_{td}$

The inter-word delay $T_{iw}$ and the next-frame delay $T_{nf}$ are used to handle received data or to prepare data for the next word or frame. These two delays have the same length (in most cases in the bit time range) and can be programmed with a second, independent set of bit fields.

- PCR.CTQSEL1
  to define the input frequency $f_{CTQIN}$ for the time quanta generation for $T_{nf}$ and $T_{iw}$
- PCR.PCTQ1
  to define the length of a time quantum (division of $f_{CTQIN}$ by 1, 2, 3, or 4) for $T_{nf}$ and $T_{iw}$
- PCR.DCTQ1
  to define the number of time quanta for the delay generation for $T_{nf}$ and $T_{iw}$
- PCR.TIWEN
  to enable/disable the inter-word delay $T_{iw}$

Each delay depends on the length of a time quantum and the programmed number of time quanta given by the bit fields CTQSEL/CTQSEL1, PCTQ/DCTQ and PCTQ1/DCTQ1 (the coding of CTQSEL1 is similar to CTQSEL, etc.). To provide a high flexibility in programming the delay length, the input frequencies can be selected between several possibilities (e.g. based on bit times or on the faster inputs of the protocol-related divider). The delay times are defined as follows:

$$T_{ld} = T_{td} = \frac{(PCTQ + 1) \times (DCTQ + 1)}{f_{CTQIN}}$$
$$T_{iw} = T_{nf} = \frac{(PCTQ1 + 1) \times (DCTQ1 + 1)}{f_{CTQIN}} \quad (15.9)$$


Subject to Agreement on the Use of Product Information

#### 15.4.3.5 Protocol Interrupt Events

The following protocol-related events generated in SSC mode and can lead to a protocol interrupt. They are related to the start and the end of a data frame. After the start of a data frame a new setting could be programmed for the next data frame and after the end of a data frame the SSC connections to pins can be changed.

Please note that the bits in register PSR are not all automatically cleared by hardware and have to be cleared by software in order to monitor new incoming events.

- MSLS Interrupt:
  This interrupt indicates in master mode (MSLS generation enabled) that a data frame has started (activation of MSLS) and has been finished (deactivation of MSLS). Any change of the internal MSLS signal sets bit PSR.MSLSEV and additionally, a protocol interrupt can be generated if PCR.MSLSIEN = 1. The actual state of the internal MSLS signal can be read out at PSR.MSLS to take appropriate actions when this interrupt has been detected.
- DX2T Interrupt:
  This interrupt monitors edges of the input signal of the DX2 stage (although this signal is not used as slave select input for data transfers).
  A programmable edge detection for the DX2 input signal sets bit PSR.DX2TEV and additionally, a protocol interrupt can be generated if PCR.DX2TIEN = 1. The actual state of the selected input signal can be read out at PSR.DX2S to take appropriate actions when this interrupt has been detected.
- Parity Error Interrupt:
  This interrupt indicates that there is a mismatch in the received parity bit (in RBUFSR.PAR) with the calculated parity bit of the last received word of a data frame.


Subject to Agreement on the Use of Product Information

### 15.4.3.6 End-of-Frame Control

The information about the frame length is required for the MSLS generator of the master device. In addition to the mechanism based on the number of bits per frame (selected with SCTR.FLE < 63), the following alternative mechanisms for end of frame handling are supported. It is recommended to set SCTRFLE = 63 (if several end of frame mechanisms are activated in parallel, the first end condition being found finishes the frame).

- Software-based start of frame indication TCSR.SOF:
  This mechanism can be used if software handles the TBUF data without data FIFO. If bit SOF is set, a valid content of TBUF is considered as first word of a new frame. Bit SOF has to be set before the content of TBUF is transferred to the transmit shift register, so it is recommended to write it before writing data to TBUF. A current data word transfer is finished completely and the slave select delays $T_{td}$ and $T_{nf}$ are applied before starting a new data frame with $T_{ld}$ and the content of TBUF.
  For software-handling of bit SOF, bit TCSR.WLEMD = 0 has to be programmed. In this case, all TBUF[31:0] address locations show an identical behavior (TCI not taken into account for data handling).
- Software-based end of frame indication TCSR.EOF:
  This mechanism can be used if software handles the TBUF data without data FIFO. If bit EOF is set, a valid content of TBUF is considered as last word of a new frame. Bit EOF has to be set before the content of TBUF is transferred to the transmit shift register, so it is recommended to write it before writing data to TBUF. The data word in TBUF is sent out completely and the slave select delays $T_{td}$ and $T_{nf}$ are applied. A new data frame can start with $T_{ld}$ with the next valid TBUF value.
  For software-handling of bit EOF, bit TCSR.WLEMD = 0 has to be programmed. In this case, all TBUF[31:0] address locations show an identical behavior (TCI not taken into account for data handling).
- Software-based address related end of frame handling:
  This mechanism can be used if software handles the TBUF data without data FIFO. If bit TCSR.WLEMD = 1, the address of the written TBUF[31:0] is used as transmit control information TCI[4:0] to update SCTR.WLE (= TCI[3:0]) and TCSR.EOF (= TCI[4]) for each data word. The written TBUF[31:0] address location defines the word length and the end of a frame (locations TBUF[31:16] lead to a frame end).
  For example, writing transmit data to TBUF[07] results in a data word of 8-bit length without finishing the frame, whereas writing transmit data to TBUF[31] leads to a data word length of 16 bits, followed by $T_{td}$, the deactivation of MSLS and $T_{nf}$.
  If TCSR.WLEMD = 1, bits TCSR.EOF and SOF, as well as SCTR.WLE must not be written by software after writing data to a TBUF location. Furthermore, it is recommended to clear bits TCSR.SELMD, FLEMD and WAMD.
- FIFO-based address related end of frame handling:
  This mechanism can be used if a data FIFO is used to store the transmit data. The general behavior is similar to the software-based address related end of frame


Subject to Agreement on the Use of Product Information

handling, except that transmit data is not written to the locations TBUF[31:0], but to the FIFO input locations IN[31:0] instead. In this case, software must not write to any of the TBUF locations.

- TBUF related end of frame handling:
  If bit PCR.FEM = 0, an end of frame is assumed if the transmit buffer TBUF does not contain valid transmit data at the end of a data word transmission (TCSR.TDV = 0 or in Stop Mode). In this case, the software has to take care that TBUF does not run empty during a data frame in Run Mode. If bit PCR.FEM = 1, signal MSLS stays active while the transmit buffer is waiting for new data (TCSR.TDV = 1 again) or until Stop Mode is left.
- Explicit end of frame by software:
  The software can explicitly stop a frame by clearing bit PSR.MSLS by writing a 1 to the related bit position in register PSCR. This write action immediately clears bit PSR.MSLS, whereas the internal MSLS signal becomes inactive after finishing a currently running word transfer and respecting the slave select delays $T_{td}$ and $T_{nf}$.


Subject to Agreement on the Use of Product Information

### 15.4.4 Operating the SSC in Slave Mode

In order to operate the SSC in slave mode, the following issues have to be considered:

- Select SSC mode:
  It is recommended to configure all parameters of the SSC that do not change during run time while CCR.MODE = $0000_B$. Bit field SCTR.TRM = $01_B$ has to be programmed. The configuration of the input stages has to be done while CCR.MODE = $0000_B$ to avoid unintended edges of the input signals and the SSC mode can be enabled afterwards by CCR.MODE = $0001_B$.
- Pin connections:
  Establish the connection of the input stage (DX0, DX3, DX4, DX5) with the selected receive data input pin (DIN[3:0]) with DXnCR.INSW = 1 and configure the transmit data output pin (DOUT[3:0]). One, two or four such connections may be needed depending on the protocol. For half-duplex configurations, hardware port control can be also used to establish the required connections.
  Establish a connection of input stage DX1 with the selected shift clock input pin (signal SCLKIN) with DX1CR.INSW = 1.
  Establish a connection of input stage DX2 with the selected slave select input pin (signal SELIN) with DX2CR.INSW = 1. If no slave select input signal is used, the DX2 stage has to deliver a 1-level to the data shift unit to allow data reception and transmission. If a slave device is not selected (DX2 stage delivers a 0 to the data shift unit) and a shift clock pulse are received, the incoming data is not received and the DOUTx signal outputs the passive data level defined by SCTR.PDL.
  Note that the step to enable the alternate output port functions should only be done after the SSC mode is enabled, to avoided unintended spikes on the output.
- Baud rate generation:
  The baud rate generator is not needed and can be switched off by the fractional divider.
- Data format configuration:
  If required, the shift mode can be set up for reception and/or transmission of two or four data bits at one time by programming the register SCTR.
- Slave select generation:
  The slave select delay generation is not needed and can be switched off. The bits and bit fields MSLSEN, SELCTR, SELINV, CTQSEL1, PCTQ1, DCTQ1, MSLSIEN, SELO[7:0], and TIWEN in register PCR are not necessary and can be programmed to 0.

#### 15.4.4.1 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.


Subject to Agreement on the Use of Product Information

Please note that the bits in register PSR are not all automatically cleared by hardware and have to be cleared by software in order to monitor new incoming events.

- MSLS event:
  The MSLS generation being switched off, this event is not available.
- DX2T event:
  The slave select input signal SELIN is handled by the DX2 stage and the edges of the selected signal can generate a protocol interrupt. This interrupt allows to indicate that a data frame has started and/or that a data frame has been completely finished. A programmable edge detection for the DX2 input signal activates DX2T, sets bit PSR.DX2TEV and additionally, a protocol interrupt can be generated if PCR.DX2TIEN = 1. The actual state of the selected input signal can be read out at PSR.DX2S to take appropriate actions when this interrupt has been detected.
- Parity Error Interrupt:
  This interrupt indicates that there is a mismatch in the received parity bit (in RBUFSR.PAR) with the calculated parity bit of the last received word of a data frame.

### 15.4.4.2 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.


Subject to Agreement on the Use of Product Information

### 15.4.5 SSC Protocol Registers

In SSC mode, the registers PCR and PSR handle SSC related information.

#### 15.4.5.1 SSC Protocol Control Registers

In SSC mode, the PCR register bits or bit fields are defined as described in this section.

**PCR**
**Protocol Control Register [SSC Mode]**

**($3C_H$)** **Reset Value: $0000\ 0000_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MCLK | 0 | | | | | SLPHSEL | TIWEN | SELO | | | | | | | |
| rw | rw | | | | | rw | rw | rw | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| DX2TIEN | MSLSIEN | PARIEN | DCTQ1 | | | | | PCTQ1 | | CTQSEL1 | | FEM | SELINV | SELCTR | MSLSEN |
| rw | rw | rw | rw | | | | | rw | | rw | | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **MSLSEN** | 0 | rw | **MSLS Enable**<br>This bit enables/disables the generation of the master slave select signal MSLS. If the SSC is a transfer slave, the SLS information is read from a pin and the internal generation is not needed. If the SSC is a transfer master, it has to provide the MSLS signal.<br>$0_B$ The MSLS generation is disabled (MSLS = 0). This is the setting for SSC slave mode.<br>$1_B$ The MSLS generation is enabled. This is the setting for SSC master mode. |
| **SELCTR** | 1 | rw | **Select Control**<br>This bit selects the operating mode for the SELO[7:0] outputs.<br>$0_B$ The coded select mode is enabled.<br>$1_B$ The direct select mode is enabled. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **SELINV** | 2 | rw | **Select Inversion**<br>This bit defines if the polarity of the SELO[7:0] outputs in relation to the master slave select signal MSLS.<br>$0_B$ The SELO outputs have the same polarity as the MSLS signal (active high).<br>$1_B$ The SELO outputs have the inverted polarity to the MSLS signal (active low). |
| **FEM** | 3 | rw | **Frame End Mode**<br>This bit defines if a transmit buffer content that is not valid for transmission is considered as an end of frame condition for the slave select generation.<br>$0_B$ The current data frame is considered as finished when the last bit of a data word has been sent out and the transmit buffer TBUF does not contain new data (TDV = 0).<br>$1_B$ The MSLS signal is kept active also while no new data is available and no other end of frame condition is reached. In this case, the software can accept delays in delivering the data without automatic deactivation of MSLS in multi-word data frames. |
| **CTQSEL1** | [5:4] | rw | **Input Frequency Selection**<br>This bit field defines the input frequency $f_{CTQIN}$ for the generation of the slave select delays $T_{iw}$ and $T_{nf}$.<br>$00_B$ $f_{CTQIN} = f_{PDIV}$<br>$01_B$ $f_{CTQIN} = f_{PPP}$<br>$10_B$ $f_{CTQIN} = f_{SCLK}$<br>$11_B$ $f_{CTQIN} = f_{MCLK}$ |
| **PCTQ1** | [7:6] | rw | **Divider Factor PCTQ1 for $T_{iw}$ and $T_{nf}$**<br>This bit field represents the divider factor PCTQ1 (range = 0 - 3) for the generation of the inter-word delay and the next-frame delay.<br>$T_{iw} = T_{nf} = 1/f_{CTQIN}$ x (PCTQ1 + 1) x (DCTQ1 + 1) |
| **DCTQ1** | [12:8] | rw | **Divider Factor DCTQ1 for $T_{iw}$ and $T_{nf}$**<br>This bit field represents the divider factor DCTQ1 (range = 0 - 31) for the generation of the inter-word delay and the next-frame delay.<br>$T_{iw} = T_{nf} = 1/f_{CTQIN}$ x (PCTQ1 + 1) x (DCTQ1 + 1) |

| Field | Bits | Type | Description |
|---|---|---|---|
| **PARIEN** | 13 | rw | **Parity Error Interrupt Enable**<br>This bit enables/disables the generation of a protocol interrupt with the detection of a parity error.<br>$0_B$ A protocol interrupt is not generated with the detection of a parity error.<br>$1_B$ A protocol interrupt is generated with the detection of a parity error. |
| **MSLSIEN** | 14 | rw | **MSLS Interrupt Enable**<br>This bit enables/disables the generation of a protocol interrupt if the state of the MSLS signal changes (indicated by PSR.MSLSEV = 1).<br>$0_B$ A protocol interrupt is not generated if a change of signal MSLS is detected.<br>$1_B$ A protocol interrupt is generated if a change of signal MSLS is detected. |
| **DX2TIEN** | 15 | rw | **DX2T Interrupt Enable**<br>This bit enables/disables the generation of a protocol interrupt if the DX2T signal becomes activated (indicated by PSR.DX2TEV = 1).<br>$0_B$ A protocol interrupt is not generated if DX2T is activated.<br>$1_B$ A protocol interrupt is generated if DX2T is activated. |
| **SELO** | [23:16] | rw | **Select Output**<br>This bit field defines the setting of the SELO[7:0] output lines.<br>$0_B$ The corresponding SELOx line cannot be activated.<br>$1_B$ The corresponding SELOx line can be activated (according to the mode selected by SELCTR). |
| **TIWEN** | 24 | rw | **Enable Inter-Word Delay $T_{iw}$**<br>This bit enables/disables the inter-word delay $T_{iw}$ after the transmission of a data word.<br>$0_B$ No delay between data words of the same frame.<br>$1_B$ The inter-word delay $T_{iw}$ is enabled and introduced between data words of the same frame. |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **SLPHSEL** | 25 | rw | **Slave Mode Clock Phase Select**<br>This bit selects the clock phase for the data shifting in slave mode.<br>$0_B$ Data bits are shifted out with the leading edge of the shift clock signal and latched in with the trailing edge.<br>$1_B$ The first data bit is shifted out when the data shift unit receives a low to high transition from the DX2 stage. Subsequent bits are shifted out with the trailing edge of the shift clock signal. Data bits are always latched in with the leading edge. |
| **MCLK** | 31 | rw | **Master Clock Enable**<br>This bit enables/disables the generation of the master clock output signal MCLK, independent from master or slave mode.<br>$0_B$ The MCLK generation is disabled and output MCLK = 0.<br>$1_B$ The MCLK generation is enabled. |
| **0** | [30:26] | rw | **Reserved**<br>Returns 0 if read; should be written with 0. |


Subject to Agreement on the Use of Product Information

### 15.4.5.2 SSC Protocol Status Register

In SSC mode, the PSR register bits or bit fields are defined as described in this section. The bits and bit fields in register PSR are not cleared by hardware.

The flags in the PSR register can be cleared by writing a 1 to the corresponding bit position in register PSCR. Writing a 1 to a bit position in PSR sets the corresponding flag, but does not lead to further actions (no interrupt generation). Writing a 0 has no effect. The PSR flags should be cleared by software before enabling a new protocol.

**PSR**
**Protocol Status Register [SSC Mode] ($48_H$)** **Reset Value: $0000\ 0000_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AIF | RIF | TBIF | TSIF | DLIF | RSIF | 0 | | | | | PARERR | DX2TEV | MSLSEV | DX2S | MSLS |
| rwh | rwh | rwh | rwh | rwh | rwh | r | | | | | rwh | rwh | rwh | rwh | rwh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **MSLS** | 0 | rwh | **MSLS Status**<br>This bit indicates the current status of the MSLS signal. It must be cleared by software to stop a running frame.<br>$0_B$ The internal signal MSLS is inactive (0).<br>$1_B$ The internal signal MSLS is active (1). |
| **DX2S** | 1 | rwh | **DX2S Status**<br>This bit indicates the current status of the DX2S signal that can be used as slave select input SELIN.<br>$0_B$ DX2S is 0.<br>$1_B$ DX2S is 1. |
| **MSLSEV** | 2 | rwh | **MSLS Event Detected**[1)]<br>This bit indicates that the MSLS signal has changed its state since MSLSEV has been cleared. Together with the MSLS status bit, the activation/deactivation of the MSLS signal can be monitored.<br>$0_B$ The MSLS signal has not changed its state.<br>$1_B$ The MSLS signal has changed its state. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **DX2TEV** | 3 | rwh | **DX2T Event Detected**[1)]<br>This bit indicates that the DX2T trigger signal has been activated since DX2TEV has been cleared.<br>$0_B$ The DX2T signal has not been activated.<br>$1_B$ The DX2T signal has been activated. |
| **PARERR** | 4 | rwh | **Parity Error Event Detected**[1)]<br>This bit indicates that there is a mismatch in the received parity bit (in RBUFSR.PAR) with the calculated parity bit of the last received word of the data frame.<br>$0_B$ A parity error event has not been activated.<br>$1_B$ A parity error event has been activated. |
| **RSIF** | 10 | rwh | **Receiver Start Indication Flag**<br>$0_B$ A receiver start event has not occurred.<br>$1_B$ A receiver start event has occurred. |
| **DLIF** | 11 | rwh | **Data Lost Indication Flag**<br>$0_B$ A data lost event has not occurred.<br>$1_B$ A data lost event has occurred. |
| **TSIF** | 12 | rwh | **Transmit Shift Indication Flag**<br>$0_B$ A transmit shift event has not occurred.<br>$1_B$ A transmit shift event has occurred. |
| **TBIF** | 13 | rwh | **Transmit Buffer Indication Flag**<br>$0_B$ A transmit buffer event has not occurred.<br>$1_B$ A transmit buffer event has occurred. |
| **RIF** | 14 | rwh | **Receive Indication Flag**<br>$0_B$ A receive event has not occurred.<br>$1_B$ A receive event has occurred. |
| **AIF** | 15 | rwh | **Alternative Receive Indication Flag**<br>$0_B$ An alternative receive event has not occurred.<br>$1_B$ An alternative receive event has occurred. |
| **BRGIF** | 16 | rwh | **Baud Rate Generator Indication Flag**<br>$0_B$ A baud rate generator event has not occurred.<br>$1_B$ A baud rate generator event has occurred. |
| **0** | [9:5], [31:17] | r | **Reserved**<br>Returns 0 if read; not modified in SSC mode. |

1) This status bit can generate a protocol interrupt in SSC mode (see **Page 15-22**). The general interrupt status flags are described in the general interrupt chapter.

### 15.4.6 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.

#### 15.4.6.1 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.

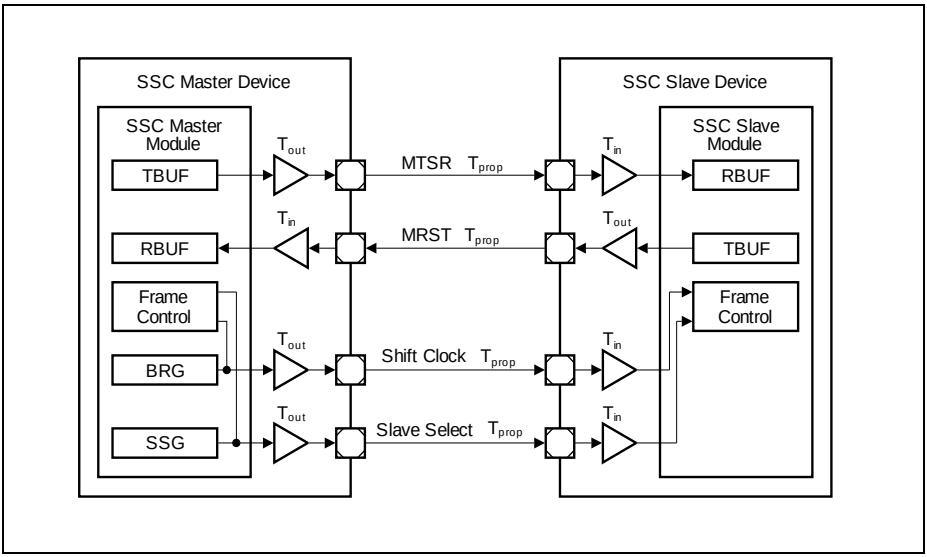



**Figure 15-46 SSC Closed-loop Delay**

The signal path between the SSC modules of the master and the slave device includes the master's output driver, the wiring to the slave device and the slave device's input stage. With the received shift clock edges, the slave device receives the master's transmit data and transmits its own data back to the master device, passing by a similar signal path in the other direction. The master module receives the slave's transmit data related to its internal shift clock edges. In order to ensure correct data reception in the master device, the slave's transmit data has to be stable (respecting setup and hold times) as master receive data with the next shift clock edge of the master (generally 1/2 shift clock period). To avoid data corruption, the accumulated delays of the input and output stages, the signal propagation on the wiring and the reaction times of the transmitter/receiver have to be carefully considered, especially at high baud rates.
In the given example, the time between the generation of the shift clock signal and the evaluation of the receive data by the master SSC module is given by the sum of $T_{out_master}$ + 2 x $T_{prop}$ + $T_{in_slave}$ + $T_{out_slave}$ + $T_{in_master}$ + module reaction times + input setup times.
The input path is characterized by an input delay depending mainly on the input stage characteristics of the pads. The output path delay is determined by the output driver delay and its slew rate, the external load and current capability of the driver. The device specific values for the input/output driver are given in the Data Sheet.


Subject to Agreement on the Use of Product Information

**Figure 15-47** describes graphically the closed-loop delay and the effect of two delay compensation options discussed in **Section 15.4.6.2** and **Section 15.4.6.3**.

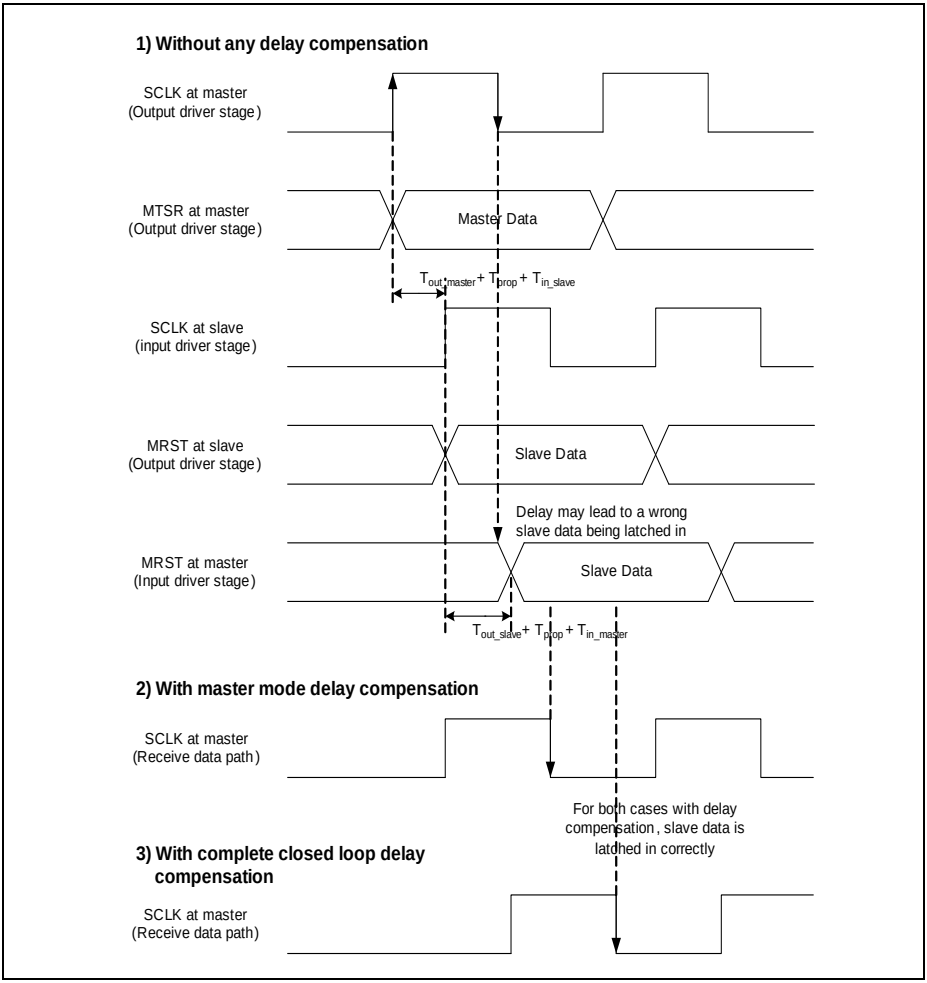



**Figure 15-47 SSC Closed-loop Delay Timing Waveform**

### 15.4.6.2 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.

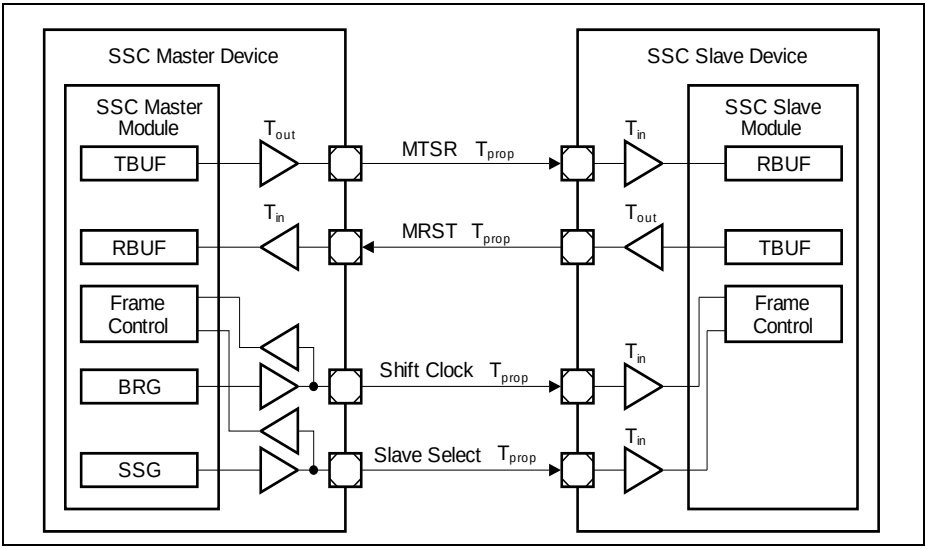



**Figure 15-48 SSC Master Mode with Delay Compensation**

If the receive shift clock signal in master mode is directly taken from the input function in parallel to the output signal, the output delay of the master device's shift clock output is compensated and only the difference between the input delays of the master and the slave devices have to be taken into account instead of the complete master's output delay and the slave's input delay of the shift clock path. The delay compensation is enabled with DX1CR.DCEN = 1 while DX1CR.INSW = 0 (transmit shift clock is taken from the baud-rate generator).

In the given example, the time between the evaluation of the shift clock signal and the receive data by the master SSC module is reduced by $T_{in_master} + T_{out_master}$.

Although being a master mode, the shift clock input and optionally the slave select signal are not directly connected internally to the data shift unit, but are taken as external signals from input pins. The delay compensation does not lead to additional pins for the SSC communication if the shift clock output pin (slave select output pin, respectively) is/are bidirectional. In this case, the input signal is decoupled from other internal signals, because it is related to the signal level at the pin itself.


Subject to Agreement on the Use of Product Information

### 15.4.6.3 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.

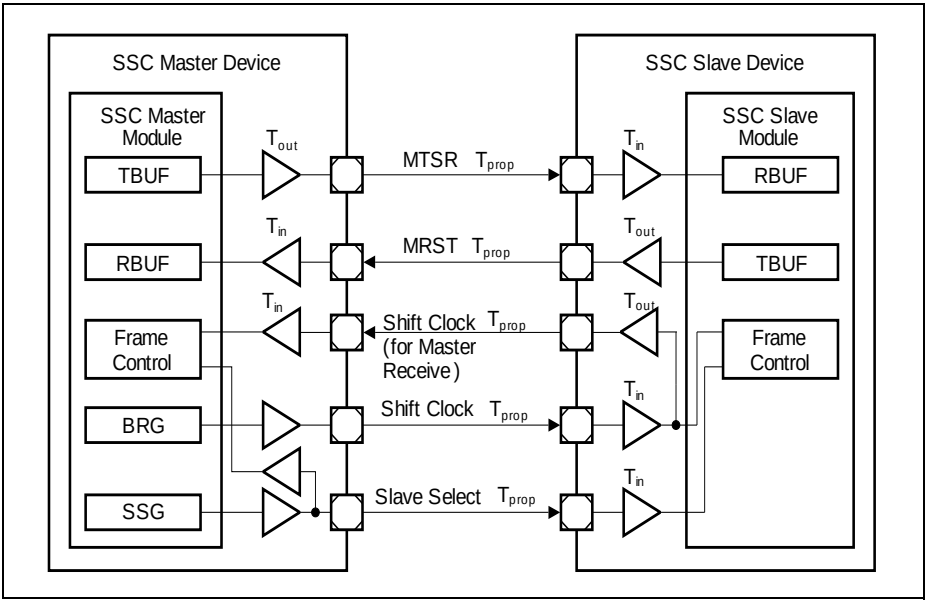



**Figure 15-49 SSC Complete Closed-loop Delay Compensation**

The principle behind this delay compensation method is to have the slave feedback the shift clock back to the master, which uses it as the receive shift clock. By going through a complete closed-loop signal path, the receive shift clock is thus fully compensated.

The slave has to setup the SCLKOUT pin function to output the shift clock by setting the bit BRG.SCLKOSEL to 1, while the master has to setup the DX1 pin function to receive the shift clock from the slave and enable the delay compensation with DX1CR.DCEN = 1 and DX1CR.INSW = 0.

## 15.5 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 = $0100_B$ with CCFG.IIC = 1 (IIC mode available).

### 15.5.1 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

#### 15.5.1.1 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 15-29** shows a connection of two IIC bus participants (modules IIC A and IIC B) using the USIC. In this example, the pin assignment of module IIC A shows separate pins for the input and output signals for SDA and SCL. This assignment can be used if the application does not provide pins having DOUT0 and a DX0 stage input for the same pin


Subject to Agreement on the Use of Product Information

(similar for SCLKOUT and DX1). The pin assignment of module IIC B shows the connection of DOUT0 and a DX0 input at the same pin, also for SCLKOUT and a DX1 input.

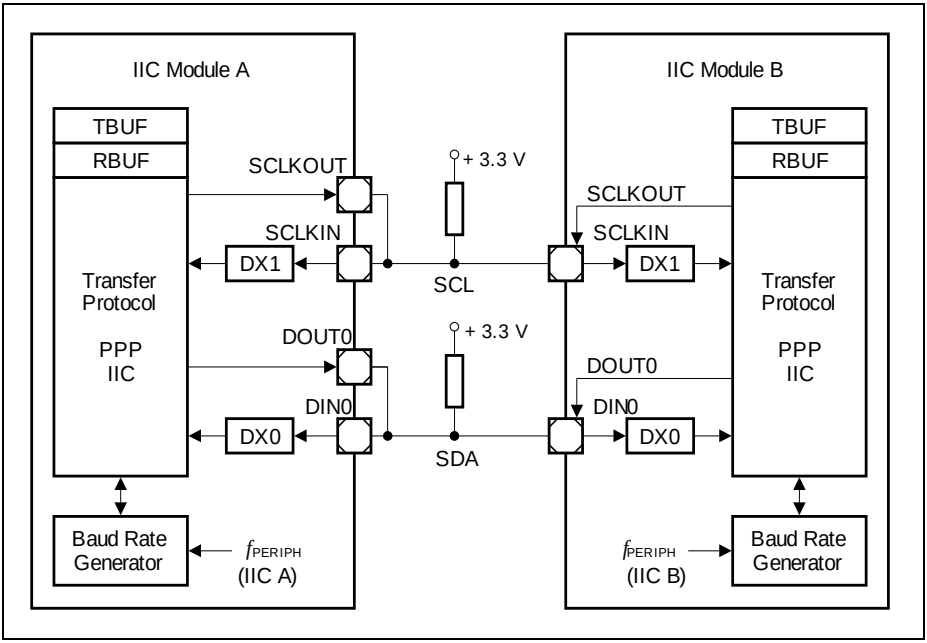



**Figure 15-50 IIC Signal Connections**

### 15.5.1.2 Symbols

A symbol is a sequence of edges on the lines SDA and SCL. Symbols contain 10 or 25 time quanta $t_q$, depending on the selected baud rate. The baud rate generator determines the length of the time quanta $t_q$, the sequence of edges in a symbol is handled by the IIC protocol pre-processor, and the sequence of symbols can be programmed by the user according to the application needs.

The following symbols are defined:

- Bus idle:
  SDA and SCL are high. No data transfer takes place currently.
- Data bit symbol:
  SDA stable during the high phase of SCL. SDA then represents the transferred bit value. There is one clock pulse on SCL for each transferred bit of data. During data transfers SDA may only change while SCL is low.


Subject to Agreement on the Use of Product Information

- Start symbol:
  Signal SDA being high followed by a falling edge of SDA while SCL is high indicates a start condition. This start condition initiates a data transfer over the IIC bus after the bus has been idle.
- Repeated start symbol:
  This start condition initiates a data transfer over the bus after a data symbol when the bus has not been idle. Therefore, SDA is set high and SCL low, followed by a start symbol.
- Stop symbol:
  A rising edge on SDA while SCL is high indicates a stop condition. This stop condition terminates a data transfer to release the bus to idle state. Between a start condition and a stop condition an arbitrary number of bytes may be transferred.

### 15.5.1.3 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.

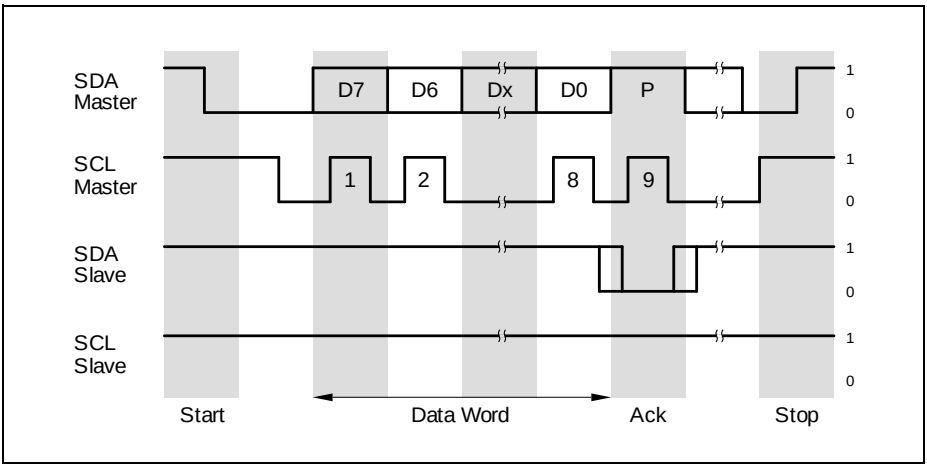



**Figure 15-51 IIC Frame Example (simplified)**


Subject to Agreement on the Use of Product Information

### 15.5.2 Operating the IIC

In order to operate the IIC protocol, the following issues have to be considered:

- Select IIC mode:
  It is recommended to configure all parameters of the IIC that do not change during run time while CCR.MODE = $0000_B$. Bit field SCTR.TRM = $11_B$ should be programmed. The configuration of the input stages has to be done while CCR.MODE = $0000_B$ to avoid unintended edges of the input signals and the IIC mode can be enabled by CCR.MODE = $0100_B$ afterwards.
- Pin connections:
  Establish a connection of input stage DX0 (with DX0CR.DPOL = 0) to the selected shift data pin SDA (signal DIN0) with DX0CR.INSW = 0 and configure the transmit data output signal DOUT0 (with SCTR.DOCFG = $00_B$) to the same pin. If available, this can be the same pin for input and output, or connect the selected input pin and the output pin to form the SDA line.
  The same mechanism applies for the shift clock line SCL. Here, signal SCLKOUT (with BRG.SCLKCFG = $00_B$) and an input of the DX1 stage have to be connected (with DX1CR.DPOL = 0).
  The input stage DX2 is not used for the IIC protocol.
  If the digital input filters are enabled in the DX0/1 stages, their delays have to be taken into account for correct calculation of the signal timings.
  The pins used for SDA and SCL have to be set to open-drain mode to support the wired-AND structure of the IIC bus lines.
  Note that the step to enable the alternate output port functions should only be done after the IIC mode is enabled, to avoided unintended spikes on the output.
- Bit timing configuration:
  In standard mode (100 kBit/s) a minimum module frequency of 2 MHz is necessary, whereas in fast mode (400 kBit/s) a minimum of 10 MHz is required. Additionally, if the digital filter stage should be used to eliminate spikes up to 50 ns, a filter frequency of 20 MHz is necessary.
  There could be an uncertainty in the SCL high phase timing of maximum $1/f_{PPP}$ if another IIC participant lengthens the SCL low phase on the bus.
  More details are given in **Section 15.5.3**.
- Data format configuration:
  The data format has to be configured for 8 data bits (SCTR.WLE = 7), unlimited data flow (SCTR.FLE = $3F_H$), and MSB shifted first (SCTR.SDIR = 1). The parity generation has to be disabled (CCR.PM = $00_B$).
- General hints:
  The IIC slave module becomes active (for reception or transmission) if it is selected by the address sent by the master. In the case that the slave sends data to the master, it uses the transmit path. So a master must not request to read data from the slave address defined for its own channel in order to avoid collisions.
  The built-in error detection mechanisms are only activated while the IIC module is


Subject to Agreement on the Use of Product Information

taking part in IIC bus traffic.
If the slave can not deal with too high frequencies, it can lengthen the low phase of the SCL signal.
For data transfers according to the IIC specification, the shift data line SDA shall only change while SCL = 0 (defined by IIC bus specification).

#### 15.5.2.1 Transmission Chain

The IIC bus protocol requiring a kind of in-bit-response during the arbitration phase and while a slave is transmitting, the resulting loop delay of the transmission chain can limit the reachable maximal baud rate, strongly depending on the bus characteristics (bus load, module frequency, etc.).

**Figure 15-50** shows the general signal path and the delays in the case of a slave transmission. The shift clock SCL is generated by the master device, output on the wire, then it passes through the input stage and the input filter. Now, the edges can be detected and the SDA data signal can be generated accordingly. The SDA signal passes through the output stage and the wire to the master receiver part. There, it passes through the input stage and the input filter before it is sampled.

This complete loop has to be finished (including all settling times to obtain stable signal levels) before the SCL signal changes again. The delays in this path have to be taken into account for the calculation of the baud rate as a function of $f_{PERIPH}$ and $f_{PPP}$.

#### 15.5.2.2 Byte Stretching

If a device is selected as transceiver and should transmit a data byte but the transmit buffer TBUF does not contain valid data to be transmitted, the device ties down SCL = 0 at the end of the previous acknowledge bit. The waiting period is finished if new valid data has been detected in TBUF.

#### 15.5.2.3 Master Arbitration

During the address and data transmission, the master transmitter checks at the rising edge of SCL for each data bit if the value it is sending is equal to the value read on the SDA line. If yes, the next data bit values can be 0. If this is not the case (transmitted value = 1, value read = 0), the master has lost the transmit arbitration. This is indicated by status flag PSR.ARL and can generate a protocol interrupt if enabled by PCR.ARLIEN.
When the transmit arbitration has been lost, the software has to initialize the complete frame again, starting with the first address byte together with the start condition for a new master transmit attempt. Arbitration also takes place for the ACK bit.


Subject to Agreement on the Use of Product Information

### 15.5.2.4 Non-Acknowledge and Error Conditions

In case of a non-acknowledge or an error, the TCSR.TDV flag remains set, but no further transmission will take place. User software must invalidate the transmit buffer and disable transmissions (by writing FMRL.MTDV = $10_B$), before configuring the transmission (by writing TBUF) again with appropriate values to react on the previous event. In the case the FIFO data buffer is used, additionally the FIFO buffer needs to be flushed and filled again.

### 15.5.2.5 Mode Control Behavior

In multi-master mode, only run mode 0 and stop mode 0 are supported, the other modes must not be programmed.

- Run Mode 0:
  Behavior as programmed. If TCSR.TDV = 0 (no new valid TBUF entry found) when a new TBUF entry needs to be processed, the IIC module waits for TDV becoming set to continue operation.
- Run Mode 1:
  Behavior as programmed. If in master mode, TCSR.TDV = 0 (no new valid TBUF entry found) when a new TBUF entry needs to be processed, the IIC module sends a stop condition to finish the frame. In slave mode, no difference to run mode 0.
- Stop Mode 0:
  Bit TCSR.TDV is internally considered as 0 (the bit itself is not modified by the stop mode). A currently running word is finished normally, but no new word is started in case of master mode (wait for TDV active).
  Bit TDV being considered as 0 for master and slave, the slave will force a wait state on the bus if read by an external master, too.
  Additionally, it is not possible to force the generation of a STOP condition out of the wait state. The reason is, that a master read transfer must be finished with a not-acknowledged followed by a STOP condition to allow the slave to release his SDA line. Otherwise the slave may force the SDA line to 0 (first data bit of next byte) making it impossible to generate the STOP condition (rising edge on SDA).
  To continue operation, the mode must be switched to run mode 0
- Stop Mode 1:
  Same as stop mode 0, but additionally, a master sends a STOP condition to finish the frame.
  If stop mode 1 is requested for a master device after the first byte of a 10 bit address, a stop condition will be sent out. In this case, a slave device will issue an error interrupt.

### 15.5.2.6 Data Transfer Interrupt Handling

The data transfer interrupts indicate events related to IIC frame handling. As the data input and output pins are the same in IIC protocol, a IIC transmitter also receives the


Subject to Agreement on the Use of Product Information

output data at its input pin. However, no receive related interrupts will be generated in this case.

- Transmit buffer event:
  The transmit buffer event indication flag PSR.TBIF is set when the content of the transmit buffer TBUF has been loaded to the transmit shift register, indicating that the action requested by the TBUF entry has started.
  With this event, bit TCSR.TDV is cleared. This interrupt can be used to write the next TBUF entry while the last one is in progress (handled by the transmitter part).
- Receive event:
  This receive event indication flag PSR.RIF indicates that a new data byte has been written to the receive buffer RBUF0/1 (except for the first data byte of a new frame, that is indicated by an alternative receive interrupt). The flag becomes set when the data byte is received (after the falling edge of SCL). This interrupt can be used to read out the received data while a new data byte can be in progress (handled by the receiver part).
- Alternate receive event:
  The alternative receive event indication flag AIF is based on bit RBUFSR[9] (same as RBUF[9]), indicating that the received data word has been the first data word of a new data frame.
- Transmit shift event:
  The transmit shift event indication flag TSIF is set after the start of the last data bit of a data byte.
- Receive start event:
  The receive start event indication flag RSIF is set after the sample point of the first data bit of a data byte.

*Note: The transmit shift and receive start events can be ignored if the application does not require them during the IIC data transfer.*

#### 15.5.2.7 IIC Protocol Interrupt Events

The following protocol-related events are generated in IIC mode and can lead to a protocol interrupt.

Please note that the bits in register PSR are not all automatically cleared by hardware and have to be cleared by software in order to monitor new incoming events.

- start condition received at a correct position in a frame (PSR.SCR)
- repeated start condition received at a correct position in a frame (PSR.RSCR)
- stop condition transferred at a correct position in a frame (PSR.PCR)
- master arbitration lost (PSR.ARL)
- slave read requested (PSR.SRR)
- acknowledge received (PSR.ACK)
- non-acknowledge received (PSR.NACK)
- start condition not at the expected position in a frame (PSR.ERR)


Subject to Agreement on the Use of Product Information

- stop condition not at the expected position in a frame (PSR.ERR)
- as slave, 10-bit address interrupted by a stop condition after the first address byte (PSR.ERR)
- TDF slave code in master mode (PSR.WTDF)
- TDF master code in slave mode (PSR.WTDF)
- Reserved TDF code found (PSR.WDTF)
- Start condition code during a running frame in master mode (PSR.WTDF)
- Data byte transmission code after transfer direction has been changed to reception (master read) in master mode (PSR.WTDF)

If a wrong TDF code is found in TBUF, the error event is active until the TDF value is either corrected or invalidated. If the related interrupt is enabled, the interrupt handler should check PSR.WDTF first and correct or invalidate TBUF, before dealing with the other possible interrupt events.

#### 15.5.2.8 Baud Rate Generator Interrupt Handling

The baud rate generator interrupt indicate that the capture mode timer has reached its maximum value. With this event, the bit PSR.BRGIF is set.

#### 15.5.2.9 Receiver Address Acknowledge

After a (repeated) start condition, the master sends a slave address to identify the target device of the communication. The start address can comprise one or two address bytes (for 7 bit or for 10 bit addressing schemes). After an address byte, a slave sensitive to the transmitted address has to acknowledge the reception.

Therefore, the slave's address can be programmed in the device, where it is compared to the received address. In case of a match, the slave answers with an acknowledge (SDA = 0). Slaves that are not targeted answer with an non-acknowledge (SDA = 1).
In addition to the match of the programmed address, another address byte value has to be answered with an acknowledge if the slave is capable to handle the corresponding requests. The address byte $00_H$ indicates a general call address, that can be acknowledged. The value $01_H$ stands for a start byte generation, that is not acknowledged

In order to allow selective acknowledges for the different values of the address byte(s), the following control mechanism is implemented:

- The address byte $00_H$ is acknowledged if bit PCR.ACK00 is set.
- The address byte $01_H$ is not acknowledged.
- The first 7 bits of a received first address byte are compared to the programmed slave address (PCR.SLAD[15:9]). If these bits match, the slave sends an acknowledge. In addition to this, if the slave address is programmed to 1111 $0XX_B$, the slave device waits for a second address byte and compares it also to PCR.SLAD[7:0] and sends an acknowledge accordingly to cover the 10 bit addressing mode. The user has to


Subject to Agreement on the Use of Product Information

take care about reserved addresses (refer to IIC specification for more detailed description). Only the address 1111 $0XX_B$ is supported.

Under each of these conditions, bit PSR.SLSEL will be set when the addressing delivered a match. This bit is cleared automatically by a (repeated) start condition.

### 15.5.2.10 Receiver Handling

A selected slave receiver always acknowledges a received data byte. If the receive buffers RBUF0/1 are already full and can not accept more data, the respective register is overwritten (PSR.DLI becomes set in this case and a protocol interrupt can be generated).

An address reception also uses the registers RBUF0/1 to store the address before checking if the device is selected. The received addresses do not set RDV0/1, so the addresses are not handled like received data.

### 15.5.2.11 Receiver Status Information

In addition to the received data byte, some IIC protocol related information is stored in the 16-bit data word of the receive buffer. The received data byte is available at the bit positions RBUF[7:0], whereas the additional information is monitored at the bit positions RBUF[12:8]. This structure allows to identify the meaning of each received data byte without reading additional registers, also when using a FIFO data buffer.

- RBUF[8]:
  Value of the received acknowledge bit. This information is also available in RBUFSR[8] as protocol argument.
- RBUF[9]:
  A 1 at this bit position indicates that after a (repeated) start condition followed by the address reception the first data byte of a new frame has been received. A 0 at this bit position indicates further data bytes. This information is also available in RBUFSR[9], allowing different interrupt routines for the address and data handling.
- RBUF[10]:
  A 0 at this bit position indicates that the data byte has been received when the device has been in slave mode, whereas a 1 indicates a reception in master mode.
- RBUF[11]:
  A 1 at this bit position indicates an incomplete/erroneous data byte in the receive buffer caused by a wrong position of a START or STOP condition in the frame. The bit is not identical to the frame error status bit in PSR, because the bit in the PSR has to be cleared by software (“sticky” bit), whereas RBUF[11] is evaluated data byte by data byte. If RBUF[11] = 0, the received data byte has been correct, independent of former errors.
- RBUF[12]:
  A 0 at this bit position indicates that the programmed address has been received. A 1 indicates a general call address.


Subject to Agreement on the Use of Product Information

### 15.5.3 Symbol Timing

The symbol timing of the IIC is determined by the master stimulating the shift clock line SCL. It is different in each of the modes.

- 100 kBaud standard mode (PCR.STIM = 0):
  The symbol timing is based on 10 time quanta $t_q$ per symbol. A minimum module clock frequency $f_{PERIPH}$ = 2 MHz is required.
- 400 kBaud fast mode (PCR.STIM = 1):
  The symbol timing is based on 25 time quanta $t_q$ per symbol. A minimum module clock frequency $f_{PERIPH}$ = 10 MHz is required.

The baud rate setting should only be changed while the transmitter and the receiver are idle or CCR.MODE = 0. The bits in register BRG define the length of a time quantum $t_q$ that is given by one period of $f_{PCTQ}$.

- BRG.CTQSEL
  to define the input frequency $f_{CTQIN}$ for the time quanta generation
- BRG.PCTQ
  to define the length of a time quantum (division of $f_{CTQIN}$ by 1, 2, 3, or 4)
- BRG.DCTQ
  to define the number of time quanta per symbol (number of $t_q$ = DCTQ + 1)

The standard setting is given by CTQSEL = $00_B$ ($f_{CTQIN} = f_{PDIV}$) and PPPEN = 0 ($f_{PPP} = f_{IN}$). Under these conditions, the frequency $f_{PCTQ}$ is given by:

$$f_{PCTQ} = f_{PIN} \times \frac{1}{PDIV + 1} \times \frac{1}{PCTQ + 1} \quad (15.10)$$

To respect the specified SDA hold time of 300 ns for standard mode and fast mode after a falling edge of signal SCL, a hold delay $t_{HDEL}$ has been introduced. It also prevents an erroneous detection of a start or a stop condition. The length of this delay can be programmed by bit field PCR.HDEL. Taking into account the input sampling and output update, bit field HDEL can be programmed according to:

(15.11)

$$HDEL \geq 300\ ns \times f_{PPP} - \left(3 \times \frac{f_{PPP}}{f_{PERIPH}}\right) + 1 \quad \text{with digital filter and HDELmin} = 2$$

$$HDEL \geq 300\ ns \times f_{PPP} - \left(3 \times \frac{f_{PPP}}{f_{PERIPH}}\right) + 2 \quad \text{without digital filter and HDELmin} = 1$$

If the digital input filter is used, HDEL compensates the filter delay of 2 filter periods ($f_{PPP}$ should be used) in case of a spike on the input signal. This ensures that a data bit on the SDA line changing just before the rising edge or behind the falling edge of SCL will not be treated as a start or stop condition.


Subject to Agreement on the Use of Product Information

### 15.5.3.1 Start Symbol

**Figure 15-52** shows the general start symbol timing.

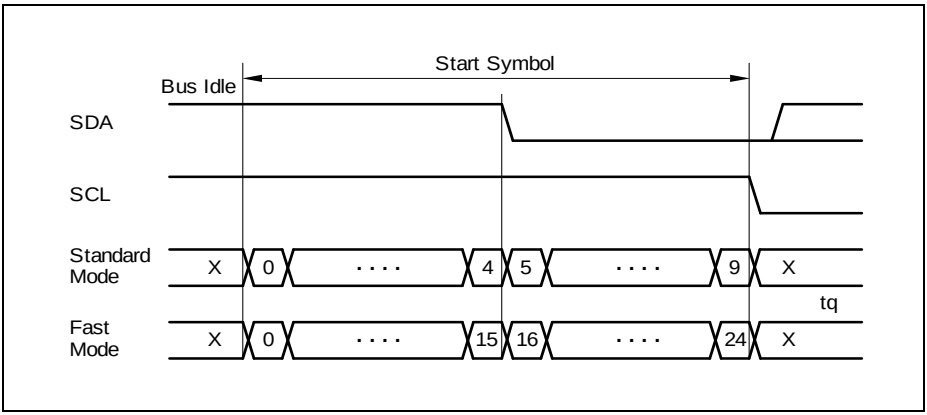



**Figure 15-52 Start Symbol Timing**

### 15.5.3.2 Repeated Start Symbol

During the first part of a repeated start symbol, an SCL low value is driven for the specified number of time quanta. Then a high value is output. After the detection of a rising edge at the SCL input, a normal start symbol is generated, as shown in **Figure 15-53**.

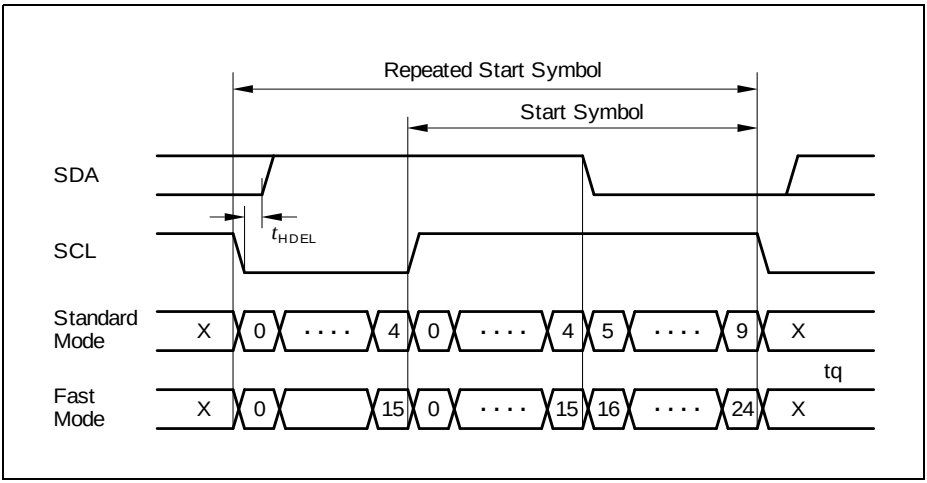



**Figure 15-53 Repeated Start Symbol Timing**


Subject to Agreement on the Use of Product Information

### 15.5.3.3 Stop Symbol

**Figure 15-54** shows the stop symbol timing.

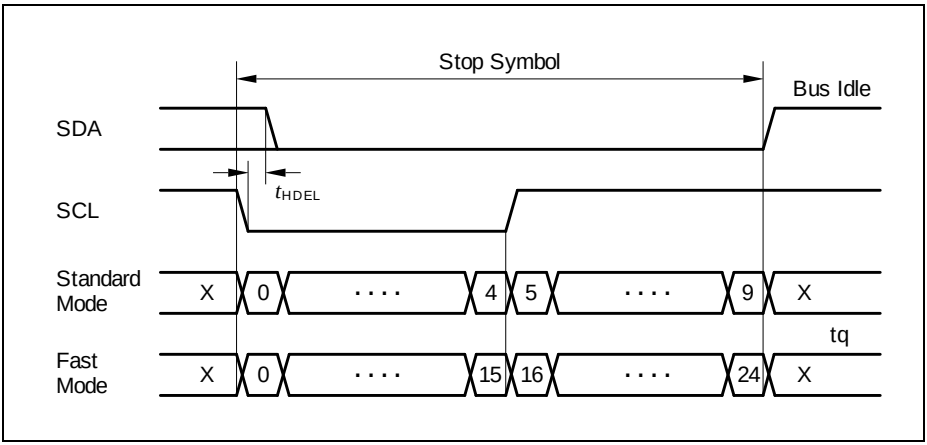



**Figure 15-54 Stop Symbol Timing**

### 15.5.3.4 Data Bit Symbol

**Figure 15-55** shows the general data bit symbol timing.

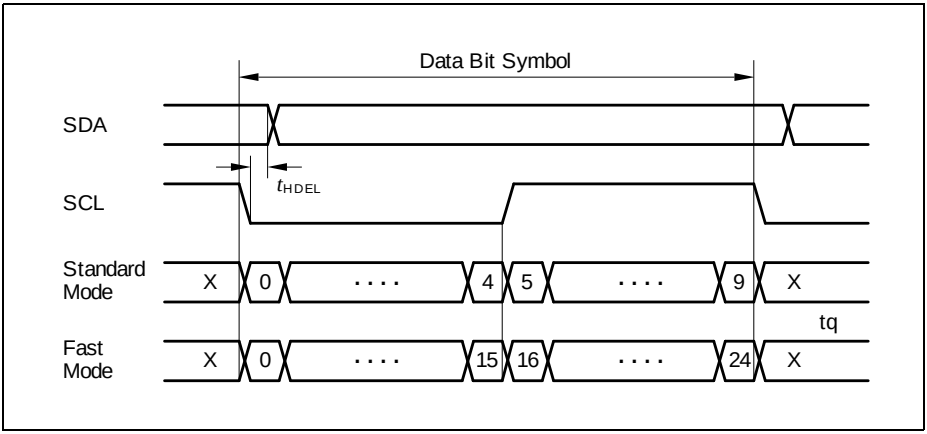



**Figure 15-55 Data Bit Symbol**

Output SDA changes after the time $t_{HDEL}$ defined by PCR.HDEL has elapsed if a falling edge is detected at the SCL input to respect the SDA hold time. The value of PCR.HDEL allows compensation of the delay of the SCL input path (sampling, filtering).

In the case of an acknowledge transmission, the USIC IIC waits for the receiver indicating that a complete byte has been received. This adds an additional delay of 3 periods of $f_{PERIPH}$ to the path. The minimum module input frequency has to be selected properly to ensure the SDA setup time to SCL rising edge.

### 15.5.4 Data Flow Handling

The handling of the data flow and the sequence of the symbols in an IIC frame is controlled by the IIC transmitter part of the USIC communication channel. The IIC bus protocol is byte-oriented, whereas a USIC data buffer word can contain up to 16 data bits. In addition to the data byte to be transmitted (located at TBUF[7:0]), bit field TDF (transmit data format) to control the IIC sequence is located at the bit positions TBUF[10:8]. The TDF code defines for each data byte how it should be transmitted (IIC master or IIC slave), and controls the transmission of (repeated) start and stop symbols. This structure allows the definition of a complete IIC frame for an IIC master device only by writing to TBUFx or by using a FIFO data buffer mechanism, because no other control registers have to be accessed. Alternatively, polling of the ACK and NACK bits in PSR register can be performed, and the next data byte is transmitted only after an ACK is received.

If a wrong or unexpected TDF code is encountered (e.g. due to a software error during setup of the transmit buffer), a stop condition will be sent out by the master. This leads to an abort of the currently running frame. A slave module waits for a valid TDF code and sets SCL = 0. The software then has to invalidate the unexpected TDF code and write a valid one.

Please note that during an arbitration phase in multi-master bus systems an unpredictable bus behavior may occur due to an unexpected stop condition.

#### 15.5.4.1 Transmit Data Formats

The following transmit data formats are available in master mode:

**Table 15-13 Master Transmit Data Formats**

| TDF Code | Description |
|---|---|
| $000_B$ | **Send data byte as master**<br>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**<br>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. |


Subject to Agreement on the Use of Product Information

**Table 15-13 Master Transmit Data Formats** (cont'd)

| TDF Code | Description |
|---|---|
| $011_B$ | **Receive data byte and send not-acknowledge**<br>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**<br>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**<br>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**<br>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**<br>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 15-14 Slave Transmit Data Format**

| TDF Code | Description |
|---|---|
| $001_B$ | **Send data byte as slave**<br>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. |

### 15.5.4.2 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 15-15 Valid TDF Codes Overview**

| Frame Position | Valid TDF Codes |
|---|---|
| First TDF code (master idle) | Start ($100_B$) |
| Read transfer: second TDF code (after start or repeated start) | Receive with acknowledge ($010_B$) or receive with not-acknowledge ($011_B$) |
| Write transfer: second TDF code (after start or repeated start) | Transmit ($000_B$), repeated start ($101_B$), or stop ($110_B$) |
| Read transfer: third and subsequent TDF code after acknowledge | Receive with acknowledge ($010_B$) or receive with not-acknowledge ($011_B$) |
| Read transfer: third and subsequent TDF code after not-acknowledge | Repeated start ($101_B$) or stop ($110_B$) |
| Write transfer: third and subsequent TDF code | Transmit ($000_B$), repeated start ($101_B$), or stop ($110_B$) |

- First TDF code:
  A master transfer starts with the TDF start code ($100_B$). All other codes are ignored, but no WTDF error will be indicated.
- TDF code after a start ($100_B$) or repeated start code ($101_B$) in case of a read access:
  If a master-read transfer is started (determined by the LSB of the address byte = 1), the transfer direction of SDA changes and the slave will actively drive the data line. In this case, only the codes $010_B$ and $011_B$ are valid. To abort the transfer in case of a wrong code, a dummy read must be performed by the master before the STOP condition can be generated.


Subject to Agreement on the Use of Product Information

- 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.


Subject to Agreement on the Use of Product Information

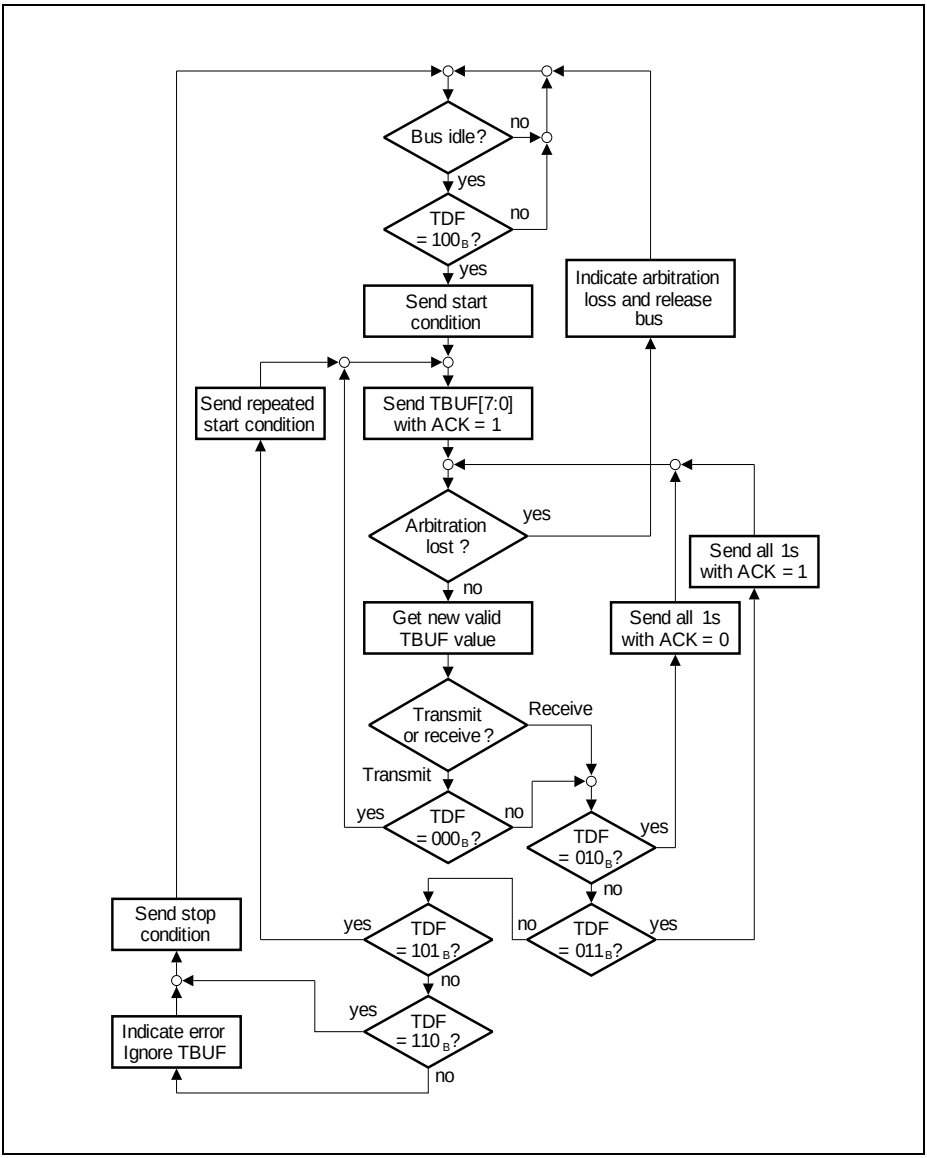



**Figure 15-56 IIC Master Transmission**


Subject to Agreement on the Use of Product Information

### 15.5.4.3 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 15-16**.

**Table 15-16 TDF Code Sequence for Master Transmit**

| TDF Code Sequence | TBUF[10:8] (TDF Code) | TBUF[7:0] | IIC Response | Interrupt Events |
|---|---|---|---|---|
| 1st code | $100_B$ | Slave address + write bit | Send START condition, slave address and write bit | SCR: Indicates a START condition is detected<br>TBIF: Next word can be written to TBUF |
| 2nd code | $000_B$ | Data or 2nd slave address byte | Send data or 2nd slave address byte | TBIF: Next word can be written to TBUF |
| Subsequent codes for data transmit | $000_B$ | Data | Send data | TBIF: Next word can be written to TBUF |
| Last code | $110_B$ | Don't care | Send STOP condition | PCR: Indicates a STOP condition is detected |

In master receive mode, the IIC receives a number of data bytes from a slave transmitter. The TDF code sequence for the master receive 7-bit and 10-bit addressing modes are shown in **Table 15-17** and **Table 15-18**.

**Table 15-17 TDF Code Sequence for Master Receive (7-bit Addressing Mode)**

| TDF Code Sequence | TBUF[10:8] (TDF Code) | TBUF[7:0] | IIC Response | Interrupt Events |
|---|---|---|---|---|
| 1st code | $100_B$ | Slave address + read bit | Send START condition, slave address and read bit | SCR: Indicates a START condition is detected<br>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<br>AIF: First data received can be read |

**Table 15-17 TDF Code Sequence for Master Receive (7-bit Addressing Mode)**

| TDF Code Sequence | TBUF[10:8] (TDF Code) | TBUF[7:0] | IIC Response | Interrupt Events |
|---|---|---|---|---|
| Subsequent codes for data receive | $010_B$ | Don't care | Receive data and send ACK bit | TBIF: Next word can be written to TBUF<br>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<br>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 15-18 TDF Code Sequence for Master Receive (10-bit Addressing Mode)**

| TDF Code Sequence | TBUF[10:8] (TDF Code) | TBUF[7:0] | IIC Response | Interrupt Events |
|---|---|---|---|---|
| 1st code | $100_B$ | Slave address (1st byte) + write bit | Send START condition, slave address (1st byte) and write bit | SCR: Indicates a START condition is detected<br>TBIF: Next word can be written to TBUF |
| 2nd code | $000_B$ | Slave address (2nd byte) | Send address (2nd byte) | TBIF: Next word can be written to TBUF |
| 3rd code | $101_B$ | 1st slave address + read bit | Send repeated START condition, slave address (1st byte) and read bit | RSCR: Indicates a repeated START condition is detected<br>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<br>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<br>RIF: Subsequent data received can be read |


Subject to Agreement on the Use of Product Information

**Table 15-18 TDF Code Sequence for Master Receive (10-bit Addressing Mode)**

| TDF Code Sequence | TBUF[10:8] (TDF Code) | TBUF[7:0] | IIC Response | Interrupt Events |
|---|---|---|---|---|
| Code for last data to be received | $011_B$ | Don't care | Receive data and send NACK bit | TBIF: Next word can be written to TBUF<br>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 15-57** shows the interrupt events during the master transmit-slave receive and master receive/slave transmit sequences.

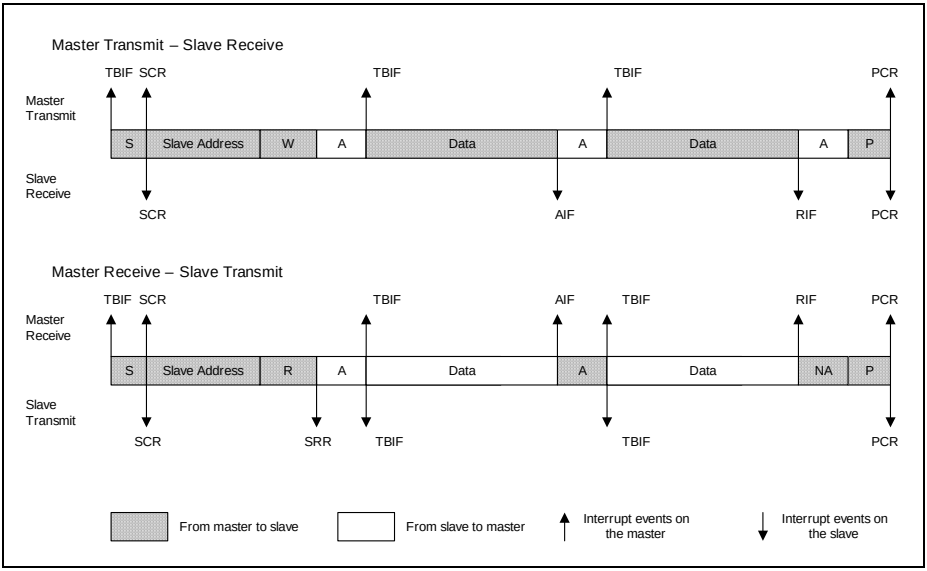



**Figure 15-57 Interrupt Events on Data Transfers**

### 15.5.4.4 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 15-57**.

### 15.5.5 IIC Protocol Registers

In IIC mode, the registers PCR and PSR handle IIC related information.

#### 15.5.5.1 IIC Protocol Control Registers

In IIC mode, the PCR register bits or bit fields are defined as described in this section.

**PCR**
**Protocol Control Register [IIC Mode]**

**($3C_H$)** **Reset Value: $0000\ 0000_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **MCLK** | **ACKIEN** | **HDEL** | | | | **SACKDIS** | **ERRIEN** | **SRRIEN** | **ARLIEN** | **NACKIEN** | **PCRIEN** | **RSCRIEN** | **SCRIEN** | **STIM** | **ACK00** |
| rw | rw | rw | | | | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **SLAD** | | | | | | | | | | | | | | | |
| rw | | | | | | | | | | | | | | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **SLAD** | [15:0] | rw | **Slave Address**<br>This bit field contains the programmed slave address. The corresponding bits in the first received address byte are compared to the bits SLAD[15:9] to check for address match. If SLAD[15:11] = $11110_B$, then the second address byte is also compared to SLAD[7:0]. |
| **ACK00** | 16 | rw | **Acknowledge $00_H$**<br>This bit defines if a slave device should be sensitive to the slave address $00_H$.<br>$0_B$ The slave device is not sensitive to this address.<br>$1_B$ The slave device is sensitive to this address. |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **STIM** | 17 | rw | **Symbol Timing**<br>This bit defines how many time quanta are used in a symbol.<br>$0_B$ A symbol contains 10 time quanta. The timing is adapted for standard mode (100 kBaud).<br>$1_B$ A symbol contains 25 time quanta. The timing is adapted for fast mode (400 kBaud). |
| **SCRIEN** | 18 | rw | **Start Condition Received Interrupt Enable**<br>This bit enables the generation of a protocol interrupt if a start condition is detected.<br>$0_B$ The start condition interrupt is disabled.<br>$1_B$ The start condition interrupt is enabled. |
| **RSCRIEN** | 19 | rw | **Repeated Start Condition Received Interrupt Enable**<br>This bit enables the generation of a protocol interrupt if a repeated start condition is detected.<br>$0_B$ The repeated start condition interrupt is disabled.<br>$1_B$ The repeated start condition interrupt is enabled. |
| **PCRIEN** | 20 | rw | **Stop Condition Received Interrupt Enable**<br>This bit enables the generation of a protocol interrupt if a stop condition is detected.<br>$0_B$ The stop condition interrupt is disabled.<br>$1_B$ The stop condition interrupt is enabled. |
| **NACKIEN** | 21 | rw | **Non-Acknowledge Interrupt Enable**<br>This bit enables the generation of a protocol interrupt if a non-acknowledge is detected by a master.<br>$0_B$ The non-acknowledge interrupt is disabled.<br>$1_B$ The non-acknowledge interrupt is enabled. |
| **ARLIEN** | 22 | rw | **Arbitration Lost Interrupt Enable**<br>This bit enables the generation of a protocol interrupt if an arbitration lost event is detected.<br>$0_B$ The arbitration lost interrupt is disabled.<br>$1_B$ The arbitration lost interrupt is enabled. |
| **SRRIEN** | 23 | rw | **Slave Read Request Interrupt Enable**<br>This bit enables the generation of a protocol interrupt if a slave read request is detected.<br>$0_B$ The slave read request interrupt is disabled.<br>$1_B$ The slave read request interrupt is enabled. |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **ERRIEN** | 24 | rw | **Error Interrupt Enable**<br>This bit enables the generation of a protocol interrupt if an IIC error condition is detected (indicated by PSR.ERR or PSR.WTDF).<br>$0_B$ The error interrupt is disabled.<br>$1_B$ The error interrupt is enabled. |
| **SACKDIS** | 25 | rw | **Slave Acknowledge Disable**<br>This bit disables the generation of an active acknowledge signal for a slave device (active acknowledge = 0 level). Once set by software, it is automatically cleared with each (repeated) start condition. If this bit is set after a byte has been received (indicated by an interrupt) but before the next acknowledge bit has started, the next acknowledge bit will be sent with passive level. This would indicate that the receiver does not accept more bytes. As a result, a minimum of 2 bytes will be received if the first receive interrupt is used to set this bit.<br>$0_B$ The generation of an active slave acknowledge is enabled (slave acknowledge with 0 level = more bytes can be received).<br>$1_B$ The generation of an active slave acknowledge is disabled (slave acknowledge with 1 level = reception stopped). |
| **HDEL** | [29:26] | rw | **Hardware Delay**<br>This bit field defines the delay used to compensate the internal treatment of the SCL signal (see **Page 15-118**) in order to respect the SDA hold time specified for the IIC protocol. |
| **ACKIEN** | 30 | rw | **Acknowledge Interrupt Enable**<br>This bit enables the generation of a protocol interrupt if an acknowledge is detected by a master.<br>$0_B$ The acknowledge interrupt is disabled.<br>$1_B$ The acknowledge interrupt is enabled. |
| **MCLK** | 31 | rw | **Master Clock Enable**<br>This bit enables generation of the master clock MCLK (not directly used for IIC protocol, can be used as general frequency output).<br>$0_B$ The MCLK generation is disabled and MCLK is 0.<br>$1_B$ The MCLK generation is enabled. |


Subject to Agreement on the Use of Product Information

### 15.5.5.2 IIC Protocol Status Register

The following PSR status bits or bit fields are available in IIC mode. Please note that the bits in register PSR are not cleared by hardware.

The flags in the PSR register can be cleared by writing a 1 to the corresponding bit position in register PSCR. Writing a 1 to a bit position in PSR sets the corresponding flag, but does not lead to further actions (no interrupt generation). Writing a 0 has no effect. These flags should be cleared by software before enabling a new protocol.

**PSR**

**Protocol Status Register [IIC Mode] ($48_H$)** **Reset Value: $0000\ 0000_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AIF | RIF | TBIF | TSIF | DLIF | RSIF | ACK | ERR | SRR | ARL | NACK | PCR | RSCR | SCR | WTDF | SLSEL |
| rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **SLSEL** | 0 | rwh | **Slave Select**<br>This bit indicates that this device has been selected as slave.<br>$0_B$ The device is not selected as slave.<br>$1_B$ The device is selected as slave. |
| **WTDF** | 1 | rwh | **Wrong TDF Code Found**[1)]<br>This bit indicates that an unexpected/wrong TDF code has been found. A protocol interrupt can be generated if PCR.ERRIEN = 1.<br>$0_B$ A wrong TDF code has not been found.<br>$1_B$ A wrong TDF code has been found. |
| **SCR** | 2 | rwh | **Start Condition Received**[1)]<br>This bit indicates that a start condition has been detected on the IIC bus lines.A protocol interrupt can be generated if PCR.SCRIEN = 1.<br>$0_B$ A start condition has not yet been detected.<br>$1_B$ A start condition has been detected. |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **RSCR** | 3 | rwh | **Repeated Start Condition Received[1)]**<br>This bit indicates that a repeated start condition has been detected on the IIC bus lines. A protocol interrupt can be generated if PCR.RSCRIEN = 1.<br>$0_B$ A repeated start condition has not yet been detected.<br>$1_B$ A repeated start condition has been detected. |
| **PCR** | 4 | rwh | **Stop Condition Received[1)]**<br>This bit indicates that a stop condition has been detected on the IIC bus lines. A protocol interrupt can be generated if PCR.PCRIEN = 1.<br>$0_B$ A stop condition has not yet been detected.<br>$1_B$ A stop condition has been detected. |
| **NACK** | 5 | rwh | **Non-Acknowledge Received[1)]**<br>This bit indicates that a non-acknowledge has been received in master mode. This bit is not set in slave mode. A protocol interrupt can be generated if PCR.NACKIEN = 1.<br>$0_B$ A non-acknowledge has not been received.<br>$1_B$ A non-acknowledge has been received. |
| **ARL** | 6 | rwh | **Arbitration Lost[1)]**<br>This bit indicates that an arbitration has been lost. A protocol interrupt can be generated if PCR.ARLIEN = 1.<br>$0_B$ An arbitration has not been lost.<br>$1_B$ An arbitration has been lost. |
| **SRR** | 7 | rwh | **Slave Read Request[1)]**<br>This bit indicates that a slave read request has been detected. It becomes active to request the first data byte to be made available in the transmit buffer. For further consecutive data bytes, the transmit buffer issues more interrupts. For the end of the transfer, the master transmitter sends a stop condition. A protocol interrupt can be generated if PCR.SRRIEN = 1.<br>$0_B$ A slave read request has not been detected.<br>$1_B$ A slave read request has been detected. |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **ERR** | 8 | rwh | **Error**[1)]<br>This bit indicates that an IIC error (frame format or TDF code) has been detected. A protocol interrupt can be generated if PCR.ERRIEN = 1.<br>$0_B$ An IIC error has not been detected.<br>$1_B$ An IIC error has been detected. |
| **ACK** | 9 | rwh | **Acknowledge Received**[1)]<br>This bit indicates that an acknowledge has been received in master mode. This bit is not set in slave mode. A protocol interrupt can be generated if PCR.ACKIEN = 1.<br>$0_B$ An acknowledge has not been received.<br>$1_B$ An acknowledge has been received. |
| **RSIF** | 10 | rwh | **Receiver Start Indication Flag**<br>$0_B$ A receiver start event has not occurred.<br>$1_B$ A receiver start event has occurred. |
| **DLIF** | 11 | rwh | **Data Lost Indication Flag**<br>$0_B$ A data lost event has not occurred.<br>$1_B$ A data lost event has occurred. |
| **TSIF** | 12 | rwh | **Transmit Shift Indication Flag**<br>$0_B$ A transmit shift event has not occurred.<br>$1_B$ A transmit shift event has occurred. |
| **TBIF** | 13 | rwh | **Transmit Buffer Indication Flag**<br>$0_B$ A transmit buffer event has not occurred.<br>$1_B$ A transmit buffer event has occurred. |
| **RIF** | 14 | rwh | **Receive Indication Flag**<br>$0_B$ A receive event has not occurred.<br>$1_B$ A receive event has occurred. |
| **AIF** | 15 | rwh | **Alternative Receive Indication Flag**<br>$0_B$ An alternative receive event has not occurred.<br>$1_B$ An alternative receive event has occurred. |
| **BRGIF** | 16 | rwh | **Baud Rate Generator Indication Flag**<br>$0_B$ A baud rate generator event has not occurred.<br>$1_B$ A baud rate generator event has occurred. |
| **0** | [31:17] | r | **Reserved**<br>Returns 0 if read; not modified in IIC mode. |

1) This status bit can generate a protocol interrupt (see **Page 15-22**). The general interrupt status flags are described in the general interrupt chapter.


Subject to Agreement on the Use of Product Information

## 15.6 Inter-IC Sound Bus Protocol (IIS)

This chapter describes how the USIC module handles the IIS protocol. This serial protocol can handle reception and transmission of synchronous data frames between a device operating in master mode and a device in slave mode. An IIS connection based on a USIC communication channel supports half-duplex and full-duplex data transfers. The IIS mode is selected by CCR.MODE = $0011_B$ with CCFG.IIS = 1 (IIS mode is available).

### 15.6.1 Introduction

The IIS protocol is a synchronous serial communication protocol mainly for audio and infotainment applications **[8]**.

#### 15.6.1.1 Signal Description

A connection between an IIS master and an IIS slave is based on the following signals:

- A shift clock signal SCK, generated by the transfer master. It is permanently generated while an IIS connection is established, also while no valid data bits are transferred.
- A word address signal WA (also named WS), generated by the transfer master. It indicates the beginning of a new data word and the targeted audio channel (e.g. left/right). The word address output signal WA is available on all SELOx outputs if the WA generation is enabled (by PCR.WAGEN = 1 for the transfer master). The WA signal changes synchronously to the falling edges of the shift clock.
- If the transmitter is the IIS master device, it generates a master transmit slave receive data signal. The data changes synchronously to the falling edges of the shift clock.
- If the transmitter is the IIS slave device, it generates a master receive slave transmit data signal. The data changes synchronously to the falling edges of the shift clock.

The transmitter part and the receiver part of the USIC communication channel can be used together to establish a full-duplex data connection between an IIS master and a slave device.

**Table 15-19 IIS IO Signals**

| IIS Mode | Receive Data | Transmit Data | Shift Clock | Word Address |
|---|---|---|---|---|
| Master | Input DIN0, handled by DX0 | Output DOUT0 | Output SCLKOUT | Output(s) SELOx |
| Slave | Input DIN0, handled by DX0 | Output DOUT0 | Input SCLKIN, handled by DX1 | Input SELIN, handled by DX2 |


Subject to Agreement on the Use of Product Information

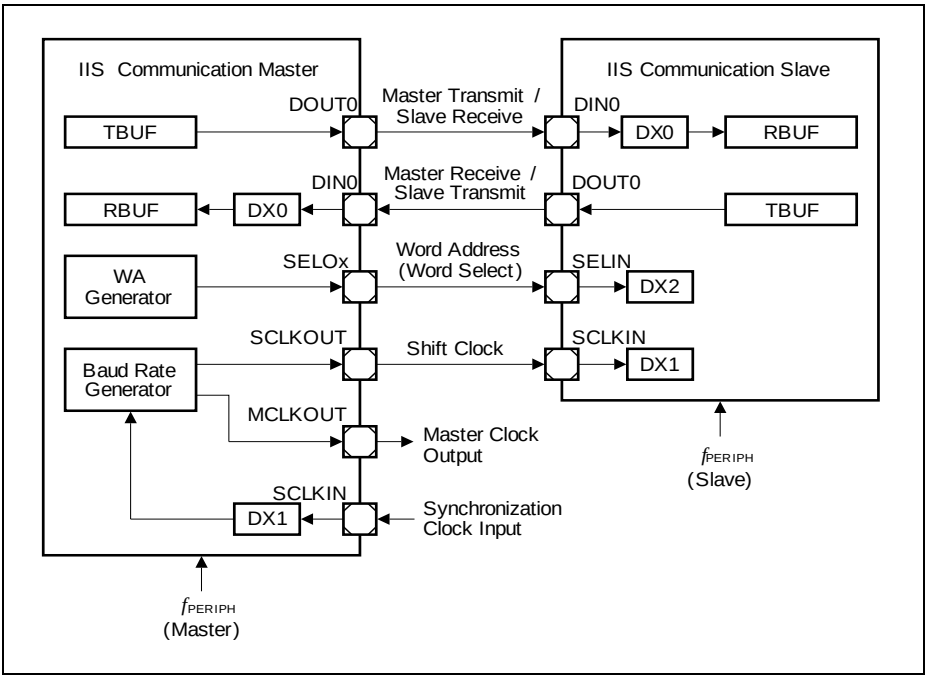



**Figure 15-58 IIS Signals**

Two additional signals are available for the USIC IIS communication master:

- A master clock output signal MCLKOUT with a fixed phase relation to the shift clock to support oversampling for audio components. It can also be used as master clock output of a communication network with synchronized IIS connections.
- A synchronization clock input SCLKIN for synchronization of the shift clock generation to an external frequency to support audio frequencies that can not be directly derived from the system clock $f_{PERIPH}$ of the communication master. It can be used as master clock input of a communication network with synchronized IIS connections.

### 15.6.1.2 Protocol Overview

An IIS connection supports transfers for two different data frames via the same data line, e.g. a data frames for the left audio channel and a data frame for the right audio channel. The word address signal WA is used to distinguish between the different data frames. Each data frame can consist of several data words.

In a USIC communication channel, data words are tagged for being transmitted for the left or for the right channel. Also the received data words contain a tag identifying the WA state when the data has been received.

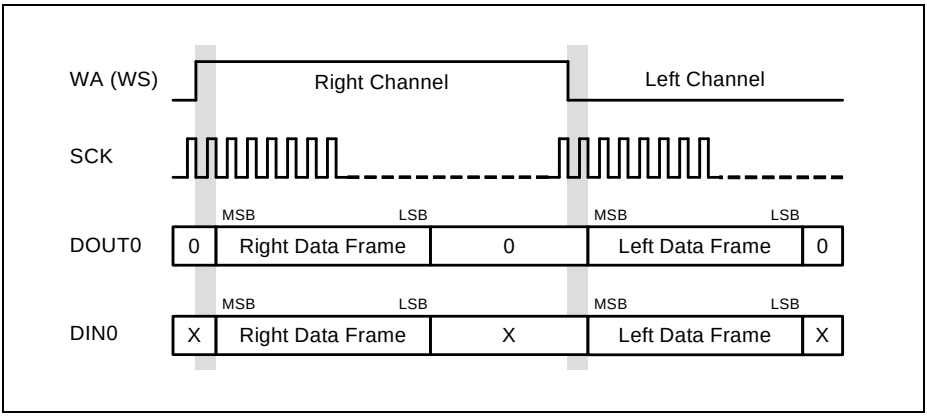



**Figure 15-59 Protocol Overview**

### 15.6.1.3 Transfer Delay

The transfer delay feature allows the transfer of data (transmission and reception) with a programmable delay (counted in shift clock periods).

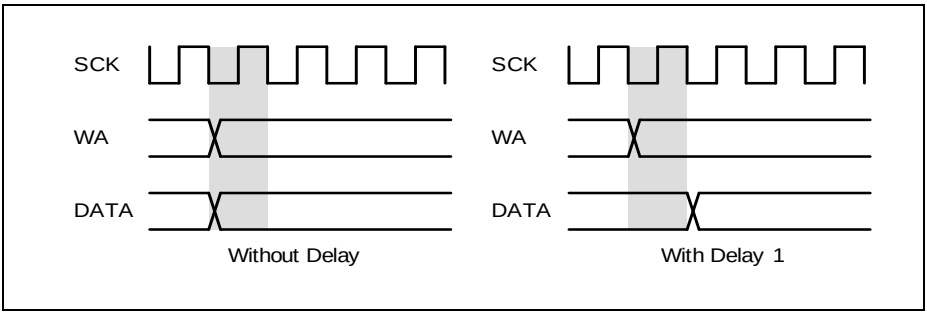



**Figure 15-60 Transfer Delay for IIS**

### 15.6.1.4 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.


Subject to Agreement on the Use of Product Information

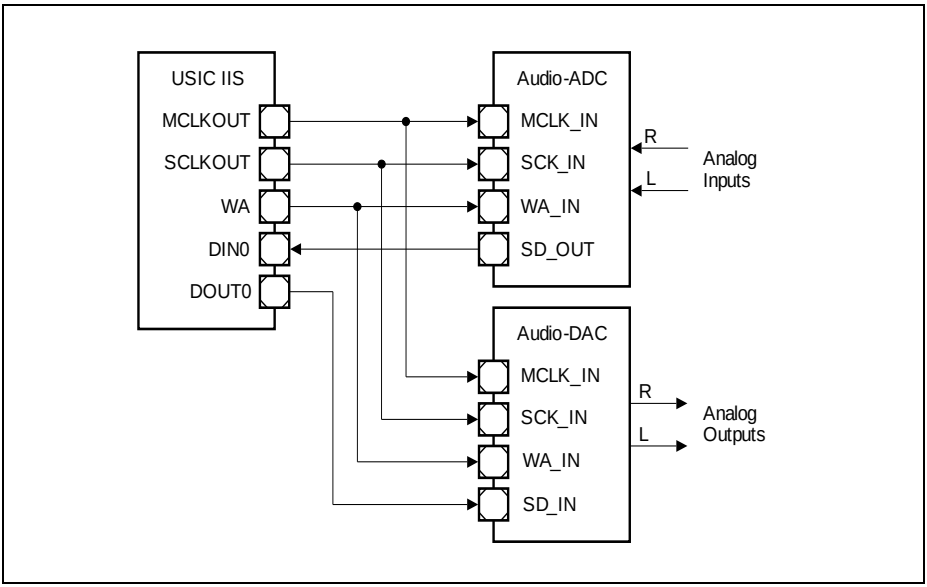



**Figure 15-61 Connection of External Audio Devices**

In some applications, especially for Audio-ADCs or Audio-DACs, a master clock signal is required with a fixed phase relation to the shift clock signal. The frequency of MCLKOUT is a multiple of the shift frequency SCLKOUT. This factor defines the oversampling factor of the external device (commonly used values: 256 or 384).

### 15.6.2 Operating the IIS

This chapter contains IIS issues, that are of general interest and not directly linked to master mode or slave mode.

#### 15.6.2.1 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.

### 15.6.2.2 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.

### 15.6.2.3 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.

### 15.6.2.4 Transfer Delay

The transfer delay can be used to synchronize a data transfer to an event (e.g. a change of the WA signal). This event has to be synchronously generated to the falling edge of the shift clock SCK (like the change of the transmit data), because the input signal for the event is directly sampled in the receiver (as a result, the transmitter can use the detection information with its next edge).

Event signals that are asynchronous to the shift clock while the shift clock is running must not be used. In the example in **Figure 15-60**, the event (change of signal WA) is generated by the transfer master and as a result, is synchronous to the shift clock SCK. With the rising edge of SCK, signal WA is sampled and checked for a change. If a change is detected, a transfer delay counter TDC is automatically loaded with its programmable reload value (PCR.TDEL), otherwise it is decremented with each rising edge of SCK until it reaches 0, where it stops. The transfer itself is started if the value of TDC has become 0. This can happen under two conditions:

- TDC is reloaded with a PCR.TDEL = 0 when the event is detected
- TDC has reached 0 while counting down

The transfer delay counter is internal to the IIS protocol pre-processor and can not be observed by software. The transfer delay in SCK cycles is given by PCR.TDEL+1.

In the example in **Figure 15-62**, the reload value PCR.TDEL for TDC is 0. When the samples taken on receiver side show the change of the WA signal, the counter TDC is reloaded. If the reload value is 0, the data transfer starts with 1 shift clock cycle delay compared to the change of WA.

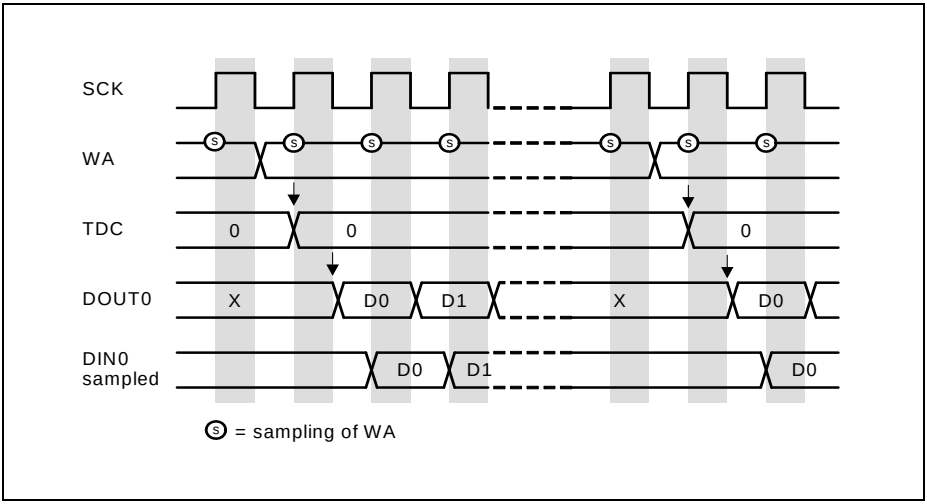



**Figure 15-62 Transfer Delay with Delay 1**

The ideal case without any transfer delay is shown in **Figure 15-63**. The WA signal changes and the data output value become valid at the same time. This implies that the transmitter “knows” in advance that the event signal will change with the next rising edge of TCLK. This is achieved by delaying the data transmission after the previously detected WA change the system word length minus 1.

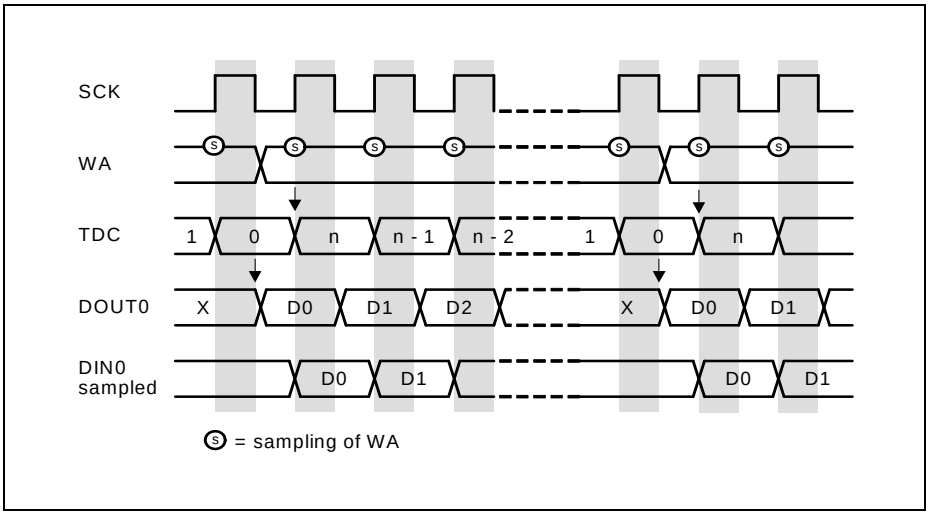



**Figure 15-63 No Transfer Delay**

If the end of the transfer delay is detected simultaneously to change of WA, the transfer is started and the delay counter is reloaded with PCR.TDEL. This allows to run the USIC as IIS device without any delay. In this case, internally the delay from the previous event elapses just at the moment when a new event occurs. If PCR.TDEL is set to a value bigger than the system word length, no transfer takes place.

### 15.6.2.5 Parity Mode

Parity generation is not supported in IIS mode and bit field CCR.PM = $00_B$ has to be programmed.

### 15.6.2.6 Transfer Mode

In IIS mode, bit field SCTR.TRM = $11_B$ has to be programmed to allow data transfers. Setting SCTR.TRM = $00_B$ disables and stops the data transfer immediately.

### 15.6.2.7 Data Transfer Interrupt Handling

The data transfer interrupts indicate events related to IIS frame handling.

- Transmit buffer interrupt TBI:
  Bit PSR.TBIF is set after the start of first data bit of a data word.
- Transmit shift interrupt TSI:
  Bit PSR.TSIF is set after the start of the last data bit of a data word.


Subject to Agreement on the Use of Product Information

- Receiver start interrupt RSI:
  Bit PSR.RSIF is set after the reception of the first data bit of a data word.
  With this event, bit TCSR.TDV is cleared and new data can be loaded to the transmit buffer.
- Receiver interrupt RI and alternative interrupt AI:
  Bit PSR.RIF is set at after the reception of the last data bit of a data word with WA = 0.
  Bit RBUFSR.SOF indicates whether the received data word has been the first data word of a new data frame.
  Bit PSR.AIF is set at after the reception of the last data bit of a data word with WA = 1.
  Bit RBUFSR.SOF indicates whether the received data word has been the first data word of a new data frame.

#### 15.6.2.8 Baud Rate Generator Interrupt Handling

The baud rate generator interrupt indicate that the capture mode timer has reached its maximum value. With this event, the bit PSR.BRGIF is set.

#### 15.6.2.9 Protocol-Related Argument and Error

In order to distinguish between data words received for the left or the right channel, the IIS protocol pre-processor samples the level of the WA input (just after the WA transition) and propagates it as protocol-related error (although it is not an error, but an indication) to the receive buffer status register at the bit position RBUFSR[9]. This bit position defines if either a standard receive interrupt (if RBUFSR[9] = 0) or an alternative receive interrupt (if RBUFSR[9] = 1) becomes activated when a new data word has been received. Incoming data can be handled by different interrupts for the left and the right channel if the corresponding events are directed to different interrupt nodes. Flag PAR is always 0.

#### 15.6.2.10 Transmit Data Handling

The IIS protocol pre-processor allows to distinguish between the left and the right channel for data transmission. Therefore, bit TCSR.WA indicates on which channel the data in the buffer will be transmitted. If TCSR.WA = 0, the data will be transmitted after a falling edge of WA. If TCSR.WA = 1, the data will be transmitted after a rising edge of WA. The WA value sampled after the WA transition is considered to distinguish between both channels (referring to PSR.WA).

Bit TCSR.WA can be automatically updated by the transmit control information TCI[4] for each data word if TCSR.WAMD = 1. In this case, data written to TBUF[15:0] (or IN[15:0] if a FIFO data buffer is used) is considered as left channel data, whereas data written to TBUF[31:16] (or IN[31:16] if a FIFO data buffer is used) is considered as right channel data.

#### 15.6.2.11 Receive Buffer Handling

If a receive FIFO buffer is available (CCFG.RB = 1) and enabled for data handling (RBCTR.SIZE > 0), it is recommended to set RBCTR.RCIM = $11_B$ in IIS mode. This leads to an indication that the data word has been the first data word of a new data frame if bit OUTR.RCI[0] = 1, and the channel indication by the sampled WA value is given by OUTR.RCI[4].

The standard receive buffer event and the alternative receive buffer event can be used for the following operation in RCI mode (RBCTR.RNM = 1):

- A standard receive buffer event indicates that a data word can be read from OUTR that belongs to a data frame started when WA = 0.
- An alternative receive buffer event indicates that a data word can be read from OUTR that belongs to a data frame started when WA = 1.

#### 15.6.2.12 Loop-Delay Compensation

The synchronous signaling mechanism of the IIS protocol being similar to the one of the SSC protocol, the closed-loop delay has to be taken into account for the application setup. In IIS mode, loop-delay compensation in master mode is also possible to achieve higher baud rates.

Please refer to the more detailed description in the SSC chapter.

### 15.6.3 Operating the IIS in Master Mode

In order to operate the IIS in master mode, the following issues have to be considered:

- Select IIS mode:
  It is recommended to configure all parameters of the IIS that do not change during run time while CCR.MODE = $0000_B$. Bit field SCTR.TRM = $11_B$ has to be programmed. The configuration of the input stages has to be done while CCR.MODE = $0000_B$ to avoid unintended edges of the input signals and the IIS mode can be enabled by CCR.MODE = $0011_B$ afterwards.
- Pin connection for data transfer:
  Establish a connection of input stage DX0 with the selected receive data input pin (DIN0) with DX0CR.INSW = 1. Configure a transmit data output pin (DOUT0) for a transmitter.
  The data shift unit allowing full-duplex data transfers based on the same WA signal, the values delivered by the DX0 stage are considered as data bits (receive function can not be disabled independently from the transmitter). To receive IIS data, the transmitter does not necessarily need to be configured (no assignment of DOUT0 signal to a pin).
- Baud rate generation:
  The desired baud rate setting has to be selected, comprising the fractional divider


Subject to Agreement on the Use of Product Information

and the baud rate generator. Bit DX1CR.INSW = 0 has to be programmed to use the baud rate generator output SCLK directly as input for the data shift unit.

- Word address WA generation:
  The WA generation has to be enabled by setting PCR.WAGEN = 1 and the programming of the number of shift clock cycles between the changes of WA. Bit DX2CR.INSW = 0 has to be programmed to use the WA generator as input for the data shift unit. Configure WA output pin for signal SELOx if needed.
- Data format configuration:
  The word length, the frame length, and the shift direction have to be set up according to the application requirements by programming the register SCTR. Generally, the MSB is shifted first (SCTR.SDIR = 1).
  Bit TCSR.WAMD can be set to use the transmit control information TCI[4] to distinguish the data words for transmission while WA = 0 or while WA = 1.

*Note: The step to enable the alternate output port functions should only be done after the IIS mode is enabled, to avoided unintended spikes on the output.*

### 15.6.3.1 Baud Rate Generation

The baud rate is defined by the frequency of the SCLK signal (one period of $f_{SCLK}$ represents one data bit).

If the fractional divider mode is used to generate $f_{PIN}$, there can be an uncertainty of one period of $f_{PERIPH}$ for $f_{PIN}$. This uncertainty does not accumulate over several SCLK cycles. As a consequence, the average frequency is reached, whereas the duty cycle of 50% of the SCLK and MCLK signals can vary by one period of $f_{PERIPH}$.

In IIS applications, where the phase relation between the optional MCLK output signal and SCLK is not relevant, SCLK can be based on the frequency $f_{PIN}$ (BRG.PPPEN = 0). In the case that a fixed phase relation between the MCLK signal and SCLK is required (e.g. when using MCLK as clock reference for external devices), the additional divider by 2 stage has to be taken into account (BRG.PPPEN = 1). This division is due to the fact that signal MCLK toggles with each cycle of $f_{PIN}$. Signal SCLK is then based on signal MCLK, see **Figure 15-64**.

The adjustable integer divider factor is defined by bit field BRG.PDIV.

$$f_{SCLK} = \frac{f_{PIN}}{2} \times \frac{1}{PDIV + 1} \quad \text{if PPPEN = 0}$$
$$f_{SCLK} = \frac{f_{PIN}}{2 \times 2} \times \frac{1}{PDIV + 1} \quad \text{if PPPEN = 1} \qquad (15.12)$$

*Note: In the IIS protocol, the master (unit generating the shift clock and the WA signal) changes the status of its data and WA output line with the falling edge of SCK. The slave transmitter also has to transmit on falling edges. The sampling of the received data is done with the rising edges of SCLK. The input stage DX1 and the*

*SCLKOUT have to be programmed to invert the shift clock signal to fit to the internal signals.*

### 15.6.3.2 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

$$N = (PCTQ + 1) \times (DCTQ + 1) \quad (15.13)$$

### 15.6.3.3 Master Clock Output

The master clock signal MCLK can be generated by the master of the IIS transfer (BRG.PPPEN = 1). It is used especially to connect external Codec devices. It can be configured by bit BRG.MCLKCFG in its polarity to become the output signal MCLKOUT.

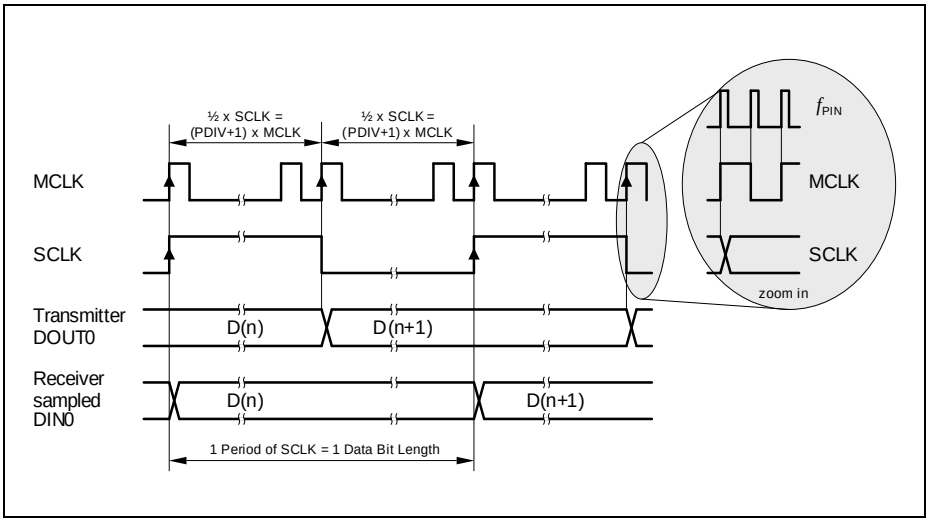



**Figure 15-64 MCLK and SCLK for IIS**

### 15.6.3.4 Protocol Interrupt Events

The following protocol-related events are generated in IIS mode and can lead to a protocol interrupt.

Please note that the bits in register PSR are not all automatically cleared by hardware and have to be cleared by software in order to monitor new incoming events.

- WA rising/falling edge events:
  The WA generation block indicates two events that are monitored in register PSR. Flag PSR.WAFE is set with the falling edge, flag PSR.WARE with the rising edge of the WA signal. A protocol interrupt can be generated if PCR.WAFEIEN = 1 for the falling edge, similar for PCR.WAREIEN = 1 for a rising edge.
- WA end event:
  The WA generation block also indicates when it has stopped the WA generation after it has been disabled by writing PCR.WAGEN = 0. A protocol interrupt can be generated if PCR.ENDIEN = 1.
- DX2T event:
  An activation of the trigger signal DX2T is indicated by PSR.DX2TEV = 1 and can generate a protocol interrupt if PCR.DX2TIEN = 1. This event can be evaluated instead of the WA rising/falling events if a delay compensation like in SSC mode (for details, refer to corresponding SSC section) is used.

## 15.6.4 Operating the IIS in Slave Mode

In order to operate the IIS in slave mode, the following issues have to be considered:

- Select IIS mode:
  It is recommended to configure all parameters of the IIS that do not change during run time while CCR.MODE = $0000_B$. Bit field SCTR.TRM = $11_B$ has to be programmed. The configuration of the input stages has to be done while CCR.MODE = $0000_B$ to avoid unintended edges of the input signals and the IIS mode can be enabled by CCR.MODE = $0011_B$ afterwards.
- Pin connection for data transfer:
  Establish a connection of input stage DX0 with the selected receive data input pin (DIN0) with DX0CR.INSW = 1. Configure a transmit data output pin (DOUT0) for a transmitter.
  The data shift unit allowing full-duplex data transfers based on the same WA signal, the values delivered by the DX0 stage are considered as data bits (receive function can not be disabled independently from the transmitter). To receive IIS data, the transmitter does not necessarily need to be configured (no assignment of DOUT0 signal to a pin).
  Note that the step to enable the alternate output port functions should only be done after the IIS mode is enabled, to avoided unintended spikes on the output.


Subject to Agreement on the Use of Product Information

- Pin connection for shift clock:
  Establish a connection of input stage DX1 with the selected shift clock input pin (SCLKIN) with DX1CR.INSW = 1 and with inverted polarity (DX1CR.DPOL = 1).
- Pin connection for WA input:
  Establish a connection of input stage DX2 with the WA input pin (SELIN) with DX2CR.INSW = 1.
- Baud rate generation:
  The baud rate generator is not needed and can be switched off by the fractional divider.
- WA generation:
  The WA generation is not needed and can be switched off (PCR.WAGEN = 0).

#### 15.6.4.1 Protocol Events and Interrupts

The following protocol-related event is generated in IIS mode and can lead to a protocol interrupt.

Please note that the bits in register PSR are not all automatically cleared by hardware and have to be cleared by software in order to monitor new incoming events.

- WA rising/falling/end events:
  The WA generation being switched off, these events are not available.
- DX2T event:
  An activation of the trigger signal DX2T is indicated by PSR.DX2TEV = 1 and can generate a protocol interrupt if PCR.DX2TIEN = 1.

### 15.6.5 IIS Protocol Registers

In IIS mode, the registers PCR and PSR handle IIS related information.

#### 15.6.5.1 IIS Protocol Control Registers

In IIS mode, the PCR register bits or bit fields are defined as described in this section.


Subject to Agreement on the Use of Product Information

## PCR
## Protocol Control Register [IIS Mode]

($3C_H$) Reset Value: 0000 0000$_H$

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MCLK | 0 | | | | | | | | | TDEL | | | | | |
| rw | rw | | | | | | | | | rw | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| DX2TIEN | 0 | | | | | | | | ENDIEN | WAREIEN | WAFEIEN | 0 | SELINV | DTEN | WAGEN |
| rw | rw | | | | | | | | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **WAGEN** | 0 | rw | **WA Generation Enable**<br>This bit enables/disables the generation of word address control output signal WA.<br>$0_B$ The IIS can be used as slave. The generation of the word address signal is disabled. The output signal WA is 0. The MCLKO signal generation depends on PCR.MCLK.<br>$1_B$ The IIS can be used as master. The generation of the word address signal is enabled. The signal starts with a 0 after being enabled. The generation of MCLK is enabled, independent of PCR.MCLK. After clearing WAGEN, the USIC module stops the generation of the WA signal within the next 4 WA periods. |
| **DTEN** | 1 | rw | **Data Transfers Enable**<br>This bit enables/disables the transfer of IIS frames as a reaction to changes of the input word address control line WA.<br>$0_B$ The changes of the WA input signal are ignored and no transfers take place.<br>$1_B$ Transfers are enabled. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **SELINV** | 2 | rw | **Select Inversion**<br>This bit defines if the polarity of the SELOx outputs in relation to the internally generated word address signal WA.<br>$0_B$ The SELOx outputs have the same polarity as the WA signal.<br>$1_B$ The SELOx outputs have the inverted polarity to the WA signal. |
| **WAFEIEN** | 4 | rw | **WA Falling Edge Interrupt Enable**<br>This bit enables/disables the activation of a protocol interrupt when a falling edge of WA has been generated.<br>$0_B$ A protocol interrupt is not activated if a falling edge of WA is generated.<br>$1_B$ A protocol interrupt is activated if a falling edge of WA is generated. |
| **WAREIEN** | 5 | rw | **WA Rising Edge Interrupt Enable**<br>This bit enables/disables the activation of a protocol interrupt when a rising edge of WA has been generated.<br>$0_B$ A protocol interrupt is not activated if a rising edge of WA is generated.<br>$1_B$ A protocol interrupt is activated if a rising edge of WA is generated. |
| **ENDIEN** | 6 | rw | **END Interrupt Enable**<br>This bit enables/disables the activation of a protocol interrupt when the WA generation stops after clearing PCR.WAGEN (complete system word length is processed before stopping).<br>$0_B$ A protocol interrupt is not activated.<br>$1_B$ A protocol interrupt is activated. |
| **DX2TIEN** | 15 | rw | **DX2T Interrupt Enable**<br>This bit enables/disables the generation of a protocol interrupt if the DX2T signal becomes activated (indicated by PSR.DX2TEV = 1).<br>$0_B$ A protocol interrupt is not generated if DX2T is active.<br>$1_B$ A protocol interrupt is generated if DX2T is active. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **TDEL** | [21:16] | rw | **Transfer Delay**<br>This bit field defines the transfer delay when an event is detected. If bit field TDEL = 0, the additional delay functionality is switched off and a delay of one shift clock cycle is introduced. |
| **MCLK** | 31 | rw | **Master Clock Enable**<br>This bit enables generation of the master clock MCLK (not directly used for IIC protocol, can be used as general frequency output).<br>$0_B$ The MCLK generation is disabled and MCLK is 0.<br>$1_B$ The MCLK generation is enabled. |
| **0** | 3, [14:7], [30:22] | rw | **Reserved**<br>Returns 0 if read; should be written with 0; |

### 15.6.5.2 IIS Protocol Status Register

The following PSR status bits or bit fields are available in IIS mode. Please note that the bits in register PSR are not cleared by hardware.

The flags in the PSR register can be cleared by writing a 1 to the corresponding bit position in register PSCR. Writing a 1 to a bit position in PSR sets the corresponding flag, but does not lead to further actions (no interrupt generation). Writing a 0 has no effect. These flags should be cleared by software before enabling a new protocol.

**PSR**
**Protocol Status Register [IIS Mode] ($48_H$)** **Reset Value: 0000 0000$_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | **0** | | | | | | | | **BRGIF** |
| | | | | | | | r | | | | | | | | rwh |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **AIF** | **RIF** | **TBIF** | **TSIF** | **DLIF** | **RSIF** | | **0** | | **END** | **WARE** | **WAFE** | **DX2TEV** | **0** | **DX2S** | **WA** |
| rwh | rwh | rwh | rwh | rwh | rwh | | r | | rwh | rwh | rwh | rwh | r | rwh | rwh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **WA** | 0 | rwh | **Word Address**<br>This bit indicates the status of the WA input signal, sampled after a transition of WA has been detected. This information is forwarded to the corresponding bit position RBUFSR[9] to distinguish between data received for the right and the left channel.<br>$0_B$ WA has been sampled 0.<br>$1_B$ WA has been sampled 1. |
| **DX2S** | 1 | rwh | **DX2S Status**<br>This bit indicates the current status of the DX2S signal, which is used as word address signal WA.<br>$0_B$ DX2S is 0.<br>$1_B$ DX2S is 1. |
| **DX2TEV** | 3 | rwh | **DX2T Event Detected**[1)]<br>This bit indicates that the DX2T signal has been activated. In IIS slave mode, an activation of DX2T generates a protocol interrupt if PCR.DX2TIEN = 1.<br>$0_B$ The DX2T signal has not been activated.<br>$1_B$ The DX2T signal has been activated. |
| **WAFE** | 4 | rwh | **WA Falling Edge Event**[1)]<br>This bit indicates that a falling edge of the WA output signal has been generated. This event generates a protocol interrupt if PCR.WAFEIEN = 1.<br>$0_B$ A WA falling edge has not been generated.<br>$1_B$ A WA falling edge has been generated. |
| **WARE** | 5 | rwh | **WA Rising Edge Event**[1)]<br>This bit indicates that a rising edge of the WA output signal has been generated. This event generates a protocol interrupt if PCR.WAREIEN = 1.<br>$0_B$ A WA rising edge has not been generated.<br>$1_B$ A WA rising edge has been generated. |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **END** | 6 | rwh | **WA Generation End**[1)]<br>This bit indicates that the WA generation has ended after clearing PCR.WAGEN. This bit should be cleared by software before clearing WAGEN.<br>$0_B$ The WA generation has not yet ended (if it is running and WAGEN has been cleared).<br>$1_B$ The WA generation has ended (if it has been running). |
| **RSIF** | 10 | rwh | **Receiver Start Indication Flag**<br>$0_B$ A receiver start event has not occurred.<br>$1_B$ A receiver start event has occurred. |
| **DLIF** | 11 | rwh | **Data Lost Indication Flag**<br>$0_B$ A data lost event has not occurred.<br>$1_B$ A data lost event has occurred. |
| **TSIF** | 12 | rwh | **Transmit Shift Indication Flag**<br>$0_B$ A transmit shift event has not occurred.<br>$1_B$ A transmit shift event has occurred. |
| **TBIF** | 13 | rwh | **Transmit Buffer Indication Flag**<br>$0_B$ A transmit buffer event has not occurred.<br>$1_B$ A transmit buffer event has occurred. |
| **RIF** | 14 | rwh | **Receive Indication Flag**<br>$0_B$ A receive event has not occurred.<br>$1_B$ A receive event has occurred. |
| **AIF** | 15 | rwh | **Alternative Receive Indication Flag**<br>$0_B$ An alternative receive event has not occurred.<br>$1_B$ An alternative receive event has occurred. |
| **BRGIF** | 16 | rwh | **Baud Rate Generator Indication Flag**<br>$0_B$ A baud rate generator event has not occurred.<br>$1_B$ A baud rate generator event has occurred. |
| **0** | 2, [9:7], [31:17] | r | **Reserved**<br>Returns 0 if read; not modified in IIS mode. |

1) This status bit can generate a protocol interrupt (see **Page 15-22**). The general interrupt status flags are described in the general interrupt chapter.

## 15.7 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, e.g. one for transmission, one for reception, one or two for protocol or error handling, or for the alternative receive events).

## 15.8 Debug Behaviour

Each USIC communication channel can be pre-configured to enter one of four kernel modes, when the program execution of the CPU is halted by the debugger.

Refer to **Section 15.2.2.2** for details.

## 15.9 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 $f_{PERIPH}$.*

## 15.10 Initialization and System Dependencies

The application has to apply the following initialization sequence before operating the USIC module:

- Enable the module by writing 1s to the MODEN and BPMODEN bits in KSCFG register.

## 15.11 Registers

**Table 15-20** shows all registers which are required for programming a USIC channel, as well as the FIFO buffer. It summarizes the USIC communication channel registers and defines the relative addresses and the reset values.

Please note that all registers can be accessed with any access width (8-bit, 16-bit, 32-bit), independent of the described width.


Subject to Agreement on the Use of Product Information

All USIC registers (except bit field KSCFG.SUMCFG) are always reset by a system reset. Bit field KSCFG.SUMCFG is reset by a debug reset.

*Note: The register bits marked "w" always deliver 0 when read. They are used to modify flip-flops in other registers or to trigger internal actions.*

**Figure 15-65** shows the register types of the USIC module registers and channel registers. In a specific microcontroller, module registers of USIC module "x" are marked by the module prefix "USICx_". Channel registers of USIC module "x" are marked by the channel prefix "USICx_CH0_" and "USICx_CH1_".

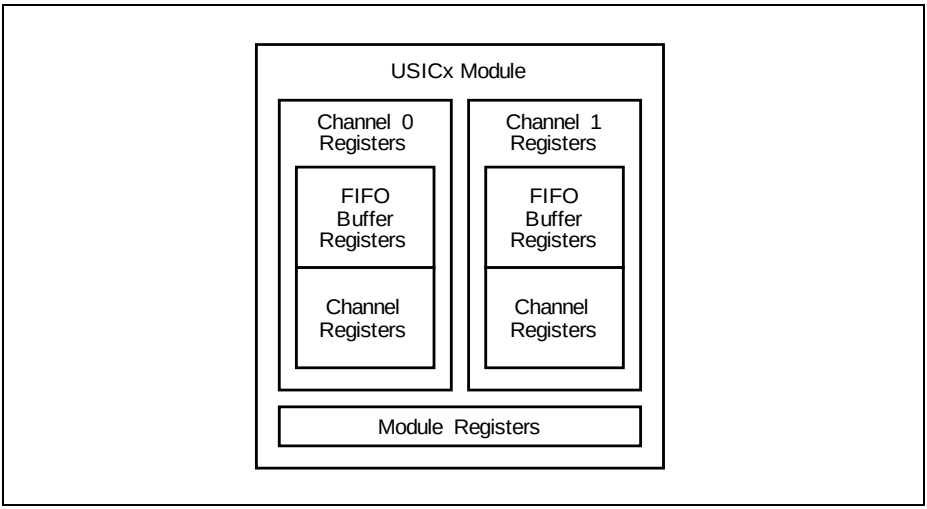



**Figure 15-65 USIC Module and Channel Registers**

**Table 15-20 USIC Kernel-Related and Kernel Registers**

| Register Short Name | Register Long Name | Offset Addr. | Access Mode | | Description see |
|---|---|---|---|---|---|
| | | | Read | Write | |
| **Module Registers**[1)] | | | | | |
| ID | Module Identification Register | $008_H$ | U, PV | U, PV | **Page 15-158** |
| **Channel Registers** | | | | | |
| – | reserved | $000_H$ | BE | BE | – |
| CCFG | Channel Configuration Register | $004_H$ | U, PV | U, PV | **Page 15-163** |
| KSCFG | Kernel State Configuration Register | $00C_H$ | U, PV | U, PV | **Page 15-164** |
| FDR | Fractional Divider Register | $010_H$ | U, PV | PV | **Page 15-177** |

**Table 15-20 USIC Kernel-Related and Kernel Registers** (cont'd)

| Register Short Name | Register Long Name | Offset Addr. | Access Mode | | Description see |
|---|---|---|---|---|---|
| | | | Read | Write | |
| BRG | Baud Rate Generator Register | $014_H$ | U, PV | PV | Page 15-178 |
| INPR | Interrupt Node Pointer Register | $018_H$ | U, PV | U, PV | Page 15-167 |
| DX0CR | Input Control Register 0 | $01C_H$ | U, PV | U, PV | Page 15-172 |
| DX1CR | Input Control Register 1 | $020_H$ | U, PV | U, PV | Page 15-174 |
| DX2CR | Input Control Register 2 | $024_H$ | U, PV | U, PV | Page 15-172 |
| DX3CR | Input Control Register 3 | $028_H$ | U, PV | U, PV | |
| DX4CR | Input Control Register 4 | $02C_H$ | U, PV | U, PV | |
| DX5CR | Input Control Register 5 | $030_H$ | U, PV | U, PV | |
| SCTR | Shift Control Register | $034_H$ | U, PV | U, PV | Page 15-182 |
| TCSR | Transmit Control/Status Register | $038_H$ | U, PV | U, PV | Page 15-185 |
| PCR | Protocol Control Register | $03C_H$ | U, PV | U, PV | Page 15-168 2) |
| | | | U, PV | U, PV | Page 15-66 3) |
| | | | U, PV | U, PV | Page 15-98 4) |
| | | | U, PV | U, PV | Page 15-129 5) |
| | | | U, PV | U, PV | Page 15-148 6) |
| CCR | Channel Control Register | $040_H$ | U, PV | PV | Page 15-159 |
| CMTR | Capture Mode Timer Register | $044_H$ | U, PV | U, PV | Page 15-181 |
| PSR | Protocol Status Register | $048_H$ | U, PV | U, PV | Page 15-169 2) |
| | | | U, PV | U, PV | Page 15-70 3) |
| | | | U, PV | U, PV | Page 15-102 4) |
| | | | U, PV | U, PV | Page 15-132 5) |
| | | | U, PV | U, PV | Page 15-150 6) |
| PSCR | Protocol Status Clear Register | $04C_H$ | U, PV | U, PV | Page 15-170 |

**Table 15-20 USIC Kernel-Related and Kernel Registers** (cont'd)

| Register Short Name | Register Long Name | Offset Addr. | Access Mode | | Description see |
|---|---|---|---|---|---|
| | | | Read | Write | |
| RBUFSR | Receiver Buffer Status Register | $050_H$ | U, PV | U, PV | Page 15-203 |
| RBUF | Receiver Buffer Register | $054_H$ | U, PV | U, PV | Page 15-201 |
| RBUFD | Receiver Buffer Register for Debugger | $058_H$ | U, PV | U, PV | Page 15-202 |
| RBUF0 | Receiver Buffer Register 0 | $05C_H$ | U, PV | U, PV | Page 15-194 |
| RBUF1 | Receiver Buffer Register 1 | $060_H$ | U, PV | U, PV | Page 15-195 |
| RBUF01SR | Receiver Buffer 01 Status Register | $064_H$ | U, PV | U, PV | Page 15-196 |
| FMR | Flag Modification Register | $068_H$ | U, PV | U, PV | Page 15-192 |
| – | reserved; do not access this location | $06C_H$ | U, PV | BE | – |
| – | reserved | $070_H$ - $07C_H$ | BE | BE | – |
| TBUFx | Transmit Buffer Input Location x (x = 00-31) | $080_H$ + x*4 | U, PV | U, PV | Page 15-194 |
| **FIFO Buffer Registers** | | | | | |
| BYP | Bypass Data Register | $100_H$ | U, PV | U, PV | Page 15-204 |
| BYPCR | Bypass Control Register | $104_H$ | U, PV | U, PV | Page 15-205 |
| TBCTR | Transmit Buffer Control Register | $108_H$ | U, PV | U, PV | Page 15-213 |
| RBCTR | Receive Buffer Control Register | $10C_H$ | U, PV | U, PV | Page 15-217 |
| TRBPTR | Transmit/Receive Buffer Pointer Register | $110_H$ | U, PV | U, PV | Page 15-225 |
| TRBSR | Transmit/Receive Buffer Status Register | $114_H$ | U, PV | U, PV | Page 15-208 |
| TRBSCR | Transmit/Receive Buffer Status Clear Register | $118_H$ | U, PV | U, PV | Page 15-212 |
| OUTR | Receive Buffer Output Register | $11C_H$ | U, PV | U, PV | Page 15-223 |
| OUTDR | Receive Buffer Output Register for Debugger | $120_H$ | U, PV | U, PV | Page 15-224 |


Subject to Agreement on the Use of Product Information

**Table 15-20 USIC Kernel-Related and Kernel Registers** (cont'd)

| Register Short Name | Register Long Name | Offset Addr. | Access Mode | | Description see |
|---|---|---|---|---|---|
| | | | Read | Write | |
| – | reserved | $124_H$ - $17C_H$ | BE | BE | – |
| INx | Transmit FIFO Buffer Input Location x (x = 00-31) | $180_H$ + x*4 | U, PV | U, PV | Page 15-222 |

1) Details of the module identification registers are described in the implementation section (see Page 15-158).
2) This page shows the general register layout.
3) This page shows the register layout in ASC mode.
4) This page shows the register layout in SSC mode.
5) This page shows the register layout in IIC mode.
6) This page shows the register layout in IIS mode.

### 15.11.1 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 15-20) plus the channel base address (given in Table 15-21).

**Table 15-21 Registers Address Space**

| Module | Base Address | End Address | Note |
|---|---|---|---|
| USIC0_CH0 | $48000000_H$ | $480001FF_H$ | – |
| USIC0_CH1 | $48000200_H$ | $480003FF_H$ | – |

**Table 15-22 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 | $48000800_H$ | $4800FFFF_H$ | BE | BE | – |

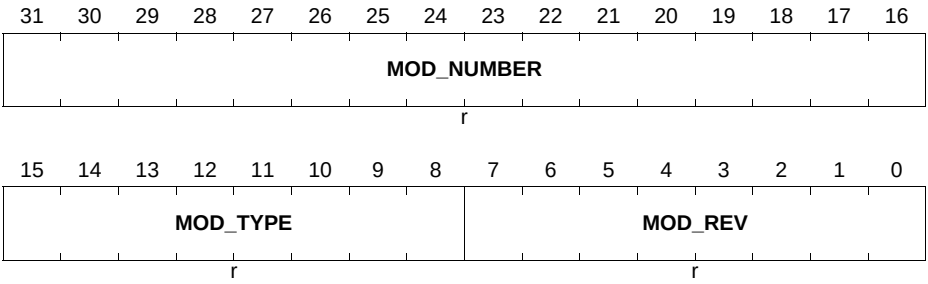
### 15.11.2 Module Identification Registers

The module identification registers indicate the function and the design step of the USIC modules.


Subject to Agreement on the Use of Product Information

**USIC0_ID**
**Module Identification Register**

**(4800 0008$_H$)** **Reset Value: 00AA C0XX$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **MOD_REV** | [7:0] | r | **Module Revision Number**<br>MOD_REV defines the revision number. The value of a module revision starts with $01_H$ (first revision). |
| **MOD_TYPE** | [15:8] | r | **Module Type**<br>This bit field is $C0_H$. It defines the module as a 32-bit module. |
| **MOD_NUMBER** | [31:16] | r | **Module Number Value**<br>This bit field defines the USIC module identification number ($00AA_H$ = USIC). |

## 15.11.3 Channel Control and Configuration Registers

### 15.11.3.1 Channel Control Register

The channel control register contains the enable/disable bits for hardware port control and interrupt generation on channel events, the control of the parity generation and the protocol selection of a USIC channel.

FDR can be written only with a privilege mode access.

**CCR**
**Channel Control Register** **($40_H$)** **Reset Value: 0000 0000$_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AIEN | RIEN | TBIEN | TSIEN | DLIEN | RSIEN | PM | | HPCEN | | 0 | | MODE | | | |
| rw | rw | rw | rw | rw | rw | rw | | rw | | r | | rw | | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **MODE** | [3:0] | rw | **Operating Mode**<br>This bit field selects the protocol for this USIC channel. Selecting a protocol that is not available (see register CCFG) or a reserved combination disables the USIC channel. When switching between two protocols, the USIC channel has to be disabled before selecting a new protocol. In this case, registers PCR and PSR have to be cleared or updated by software.<br>$0_H$ The USIC channel is disabled. All protocol-related state machines are set to an idle state.<br>$1_H$ The SSC (SPI) protocol is selected.<br>$2_H$ The ASC (SCI, UART) protocol is selected.<br>$3_H$ The IIS protocol is selected.<br>$4_H$ The IIC protocol is selected.<br>Other bit combinations are reserved. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **HPCEN** | [7:6] | rw | **Hardware Port Control Enable**<br>This bit enables the hardware port control for the specified set of DX[3:0] and DOUT[3:0] pins.<br>$00_B$ The hardware port control is disabled.<br>$01_B$ The hardware port control is enabled for DX0 and DOUT0.<br>$10_B$ The hardware port control is enabled for DX3, DX0 and DOUT[1:0].<br>$11_B$ The hardware port control is enabled for DX0, DX[5:3] and DOUT[3:0].<br>*Note: The hardware port control feature is useful only for SSC protocols in half-duplex configurations, such as dual- and quad-SSC. For all other protocols HPCEN must always be written with $00_B$.* |
| **PM** | [9:8] | rw | **Parity Mode**<br>This bit field defines the parity generation of the sampled input values.<br>$00_B$ The parity generation is disabled.<br>$01_B$ Reserved<br>$10_B$ Even parity is selected (parity bit = 1 on odd number of 1s in data, parity bit = 0 on even number of 1s in data).<br>$11_B$ Odd parity is selected (parity bit = 0 on odd number of 1s in data, parity bit = 1 on even number of 1s in data). |
| **RSIEN** | 10 | rw | **Receiver Start Interrupt Enable**<br>This bit enables the interrupt generation in case of a receiver start event.<br>$0_B$ The receiver start interrupt is disabled.<br>$1_B$ The receiver start interrupt is enabled.<br>In case of a receiver start event, the service request output SRx indicated by INPR.TBINP is activated. |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **DLIEN** | 11 | rw | **Data Lost Interrupt Enable**<br>This bit enables the interrupt generation in case of a data lost event (data received in RBUFx while RDVx = 1).<br>$0_B$ The data lost interrupt is disabled.<br>$1_B$ The data lost interrupt is enabled. In case of a data lost event, the service request output SRx indicated by INPR.PINP is activated. |
| **TSIEN** | 12 | rw | **Transmit Shift Interrupt Enable**<br>This bit enables the interrupt generation in case of a transmit shift event.<br>$0_B$ The transmit shift interrupt is disabled.<br>$1_B$ The transmit shift interrupt is enabled. In case of a transmit shift interrupt event, the service request output SRx indicated by INPR.TSINP is activated. |
| **TBIEN** | 13 | rw | **Transmit Buffer Interrupt Enable**<br>This bit enables the interrupt generation in case of a transmit buffer event.<br>$0_B$ The transmit buffer interrupt is disabled.<br>$1_B$ The transmit buffer interrupt is enabled. In case of a transmit buffer event, the service request output SRx indicated by INPR.TBINP is activated. |
| **RIEN** | 14 | rw | **Receive Interrupt Enable**<br>This bit enables the interrupt generation in case of a receive event.<br>$0_B$ The receive interrupt is disabled.<br>$1_B$ The receive interrupt is enabled. In case of a receive event, the service request output SRx indicated by INPR.RINP is activated. |
| **AIEN** | 15 | rw | **Alternative Receive Interrupt Enable**<br>This bit enables the interrupt generation in case of a alternative receive event.<br>$0_B$ The alternative receive interrupt is disabled.<br>$1_B$ The alternative receive interrupt is enabled. In case of an alternative receive event, the service request output SRx indicated by INPR.AINP is activated. |

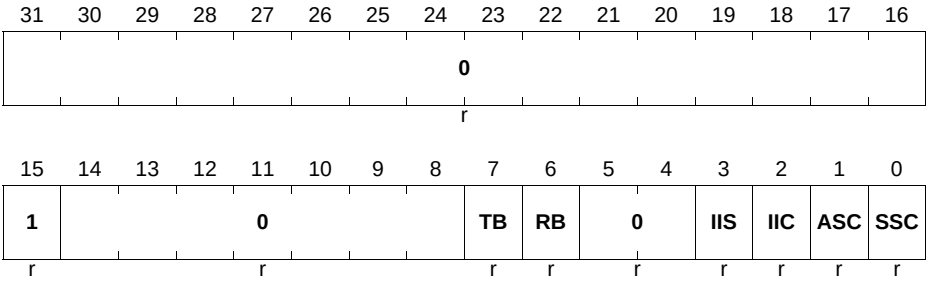

Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **BRGIEN** | 16 | rw | **Baud Rate Generator Interrupt Enable**<br>This bit enables the interrupt generation in case of a baud rate generator event.<br>$0_B$ The baud rate generator interrupt is disabled.<br>$1_B$ The baud rate generator interrupt is enabled. In case of a baud rate generator event, the service request output SRx indicated by INPR.PINP is activated. |
| **0** | [5:4], [31:17] | r | **Reserved**<br>Read as 0; should be written with 0. |

### 15.11.3.2 Channel Configuration Register

The channel configuration register contains indicates the functionality that is available in the USIC channel.

**CCFG**
**Channel Configuration Register ($04_H$) Reset Value: 0000 80CF$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **SSC** | 0 | r | **SSC Protocol Available**<br>This bit indicates if the SSC protocol is available.<br>$0_B$ The SSC protocol is not available.<br>$1_B$ The SSC protocol is available. |
| **ASC** | 1 | r | **ASC Protocol Available**<br>This bit indicates if the ASC protocol is available.<br>$0_B$ The ASC protocol is not available.<br>$1_B$ The ASC protocol is available. |
| **IIC** | 2 | r | **IIC Protocol Available**<br>This bit indicates if the IIC functionality is available.<br>$0_B$ The IIC protocol is not available.<br>$1_B$ The IIC protocol is available. |
| **IIS** | 3 | r | **IIS Protocol Available**<br>This bit indicates if the IIS protocol is available.<br>$0_B$ The IIS protocol is not available.<br>$1_B$ The IIS protocol is available. |
| **RB** | 6 | r | **Receive FIFO Buffer Available**<br>This bit indicates if an additional receive FIFO buffer is available.<br>$0_B$ A receive FIFO buffer is not available.<br>$1_B$ A receive FIFO buffer is available. |

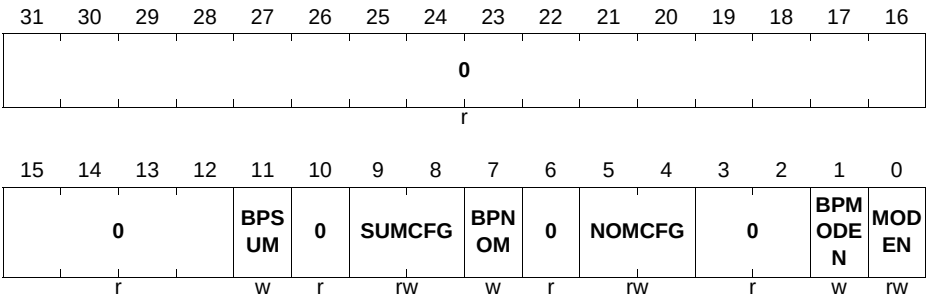

Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **TB** | 7 | r | **Transmit FIFO Buffer Available**<br>This bit indicates if an additional transmit FIFO buffer is available.<br>$0_B$ A transmit FIFO buffer is not available.<br>$1_B$ A transmit FIFO buffer is available. |
| **1** | 15 | r | **Reserved**<br>Read as 1; should be written with 1. |
| **0** | [5:4], [14:8], [31:16] | r | **Reserved**<br>Read as 0; should be written with 0. |

### 15.11.3.3 Kernel State Configuration Register

The kernel state configuration register KSCFG allows the selection of the desired kernel modes for the different device operating modes.

**KSCFG**
**Kernel State Configuration Register ($0C_H$)** **Reset Value: $0000\ 0000_H$**





Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **MODEN** | 0 | rw | **Module Enable**<br>This bit enables the module kernel clock and the module functionality.<br>$0_B$ The module is switched off immediately (without respecting a stop condition). It does not react on mode control actions and the module clock is switched off. The module does not react on read accesses and ignores write accesses (except to KSCFG).<br>$1_B$ The module is switched on and can operate. After writing 1 to MODEN, it is recommended to read register KSCFG to avoid pipeline effects in the control block before accessing other USIC registers. |
| **BPMODEN** | 1 | w | **Bit Protection for MODEN**<br>This bit enables the write access to the bit MODEN. It always reads 0.<br>$0_B$ MODEN is not changed.<br>$1_B$ MODEN is updated with the written value. |
| **NOMCFG** | [5:4] | rw | **Normal Operation Mode Configuration**<br>This bit field defines the kernel mode applied in normal operation mode.<br>$00_B$ Run mode 0 is selected.<br>$01_B$ Run mode 1 is selected.<br>$10_B$ Stop mode 0 is selected.<br>$11_B$ Stop mode 1 is selected. |
| **BPNOM** | 7 | w | **Bit Protection for NOMCFG**<br>This bit enables the write access to the bit field NOMCFG. It always reads 0.<br>$0_B$ NOMCFG is not changed.<br>$1_B$ NOMCFG is updated with the written value. |
| **SUMCFG** | [9:8] | rw | **Suspend Mode Configuration**<br>This bit field defines the kernel mode applied in suspend mode. Coding like NOMCFG. |

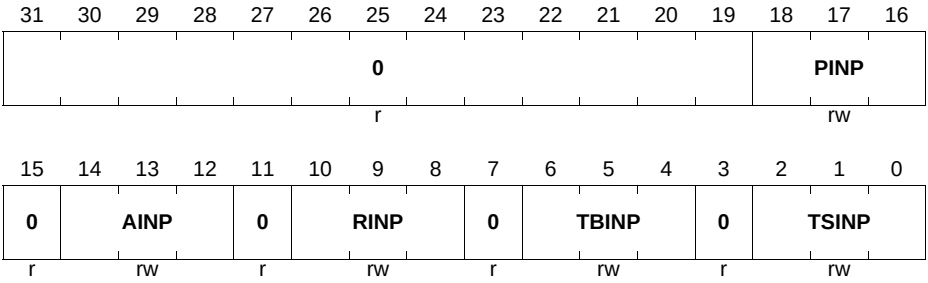

Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **BPSUM** | 11 | w | **Bit Protection for SUMCFG**<br>This bit enables the write access to the bit field SUMCFG. It always reads 0.<br>$0_B$ SUMCFG is not changed.<br>$1_B$ SUMCFG is updated with the written value. |
| **0** | [3:2], 6, 10, [31:12] | r | **Reserved**<br>Read as 0; should be written with 0. Bit 2 can read as 1 after BootROM exit (but can be ignored). |

### 15.11.3.4 Interrupt Node Pointer Register

The interrupt node pointer register defines the service request output SRx that is activated if the corresponding event occurs and interrupt generation is enabled.

**INPR**
**Interrupt Node Pointer Register ($18_H$) Reset Value: 0000 0000$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **TSINP** | [2:0] | rw | **Transmit Shift Interrupt Node Pointer**<br>This bit field defines which service request output SRx becomes activated in case of a transmit shift interrupt.<br>$000_B$ Output SR0 becomes activated.<br>$001_B$ Output SR1 becomes activated.<br>$010_B$ Output SR2 becomes activated.<br>$011_B$ Output SR3 becomes activated.<br>$100_B$ Output SR4 becomes activated.<br>$101_B$ Output SR5 becomes activated.<br>*Note: All other settings of the bit field are reserved.* |
| **TBINP** | [6:4] | rw | **Transmit Buffer Interrupt Node Pointer**<br>This bit field defines which service request output SRx will be activated in case of a transmit buffer interrupt or a receive start interrupt.<br>Coding like TSINP. |
| **RINP** | [10:8] | rw | **Receive Interrupt Node Pointer**<br>This bit field defines which service request output SRx will be activated in case of a receive interrupt.<br>Coding like TSINP. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **AINP** | [14:12] | rw | **Alternative Receive Interrupt Node Pointer**<br>This bit field defines which service request output SRx will be activated in case of a alternative receive interrupt.<br>Coding like TSINP. |
| **PINP** | [18:16] | rw | **Protocol Interrupt Node Pointer**<br>This bit field defines which service request output SRx becomes activated in case of a protocol interrupt.<br>Coding like TSINP. |
| **0** | 3, 7, 11, 15, [31:19] | r | **Reserved**<br>Read as 0; should be written with 0. |

## 15.11.4 Protocol Related Registers

### 15.11.4.1 Protocol Control Registers

The bits in the protocol control register define protocol-specific functions. They have to be configured by software before enabling a new protocol. Only the bits used for the selected protocol are taken into account, whereas the other bit positions always read as 0. The protocol-specific meaning is described in the related protocol section.

**PCR**
**Protocol Control Register** **($3C_H$)** **Reset Value: 0000 0000$_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **CTR 31** | **CTR 30** | **CTR 29** | **CTR 28** | **CTR 27** | **CTR 26** | **CTR 25** | **CTR 24** | **CTR 23** | **CTR 22** | **CTR 21** | **CTR 20** | **CTR 19** | **CTR 18** | **CTR 17** | **CTR 16** |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **CTR 15** | **CTR 14** | **CTR 13** | **CTR 12** | **CTR 11** | **CTR 10** | **CTR 9** | **CTR 8** | **CTR 7** | **CTR 6** | **CTR 5** | **CTR 4** | **CTR 3** | **CTR 2** | **CTR 1** | **CTR 0** |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **CTRx (x = 0-31)** | x | rw | **Protocol Control Bit x**<br>This bit is a protocol control bit. |


Subject to Agreement on the Use of Product Information

### 15.11.4.2 Protocol Status Register

The flags in the protocol status register can be cleared by writing a 1 to the corresponding bit position in register PSCR. Writing a 1 to a bit position in PSR sets the corresponding flag, but does not lead to further actions (no interrupt generation). Writing a 0 has no effect. These flags should be cleared by software before enabling a new protocol. The protocol-specific meaning is described in the related protocol section.

**PSR**
**Protocol Status Register** **($48_H$)** **Reset Value: 0000 $0000_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | **0** | | | | | | | | **BRGIF** |
| | | | | | | | r | | | | | | | | rwh |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **AIF** | **RIF** | **TBIF** | **TSIF** | **DLIF** | **RSIF** | **ST9** | **ST8** | **ST7** | **ST6** | **ST5** | **ST4** | **ST3** | **ST2** | **ST1** | **ST0** |
| rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **STx (x = 0-9)** | x | rwh | **Protocol Status Flag x**<br>See protocol specific description. |
| **RSIF** | 10 | rwh | **Receiver Start Indication Flag**<br>$0_B$ A receiver start event has not occurred.<br>$1_B$ A receiver start event has occurred. |
| **DLIF** | 11 | rwh | **Data Lost Indication Flag**<br>$0_B$ A data lost event has not occurred.<br>$1_B$ A data lost event has occurred. |
| **TSIF** | 12 | rwh | **Transmit Shift Indication Flag**<br>$0_B$ A transmit shift event has not occurred.<br>$1_B$ A transmit shift event has occurred. |
| **TBIF** | 13 | rwh | **Transmit Buffer Indication Flag**<br>$0_B$ A transmit buffer event has not occurred.<br>$1_B$ A transmit buffer event has occurred. |
| **RIF** | 14 | rwh | **Receive Indication Flag**<br>$0_B$ A receive event has not occurred.<br>$1_B$ A receive event has occurred. |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **AIF** | 15 | rwh | **Alternative Receive Indication Flag**<br>$0_B$ An alternative receive event has not occurred.<br>$1_B$ An alternative receive event has occurred. |
| **BRGIF** | 16 | rwh | **Baud Rate Generator Indication Flag**<br>$0_B$ A baud rate generator event has not occurred.<br>$1_B$ A baud rate generator event has occurred. |
| **0** | [31:17] | r | **Reserved;**<br>read as 0; should be written with 0; |

### 15.11.4.3 Protocol Status Clear Register

Read accesses to this register always deliver 0 at all bit positions.

**PSCR**
**Protocol Status Clear Register** **($4C_H$)** **Reset Value: 0000 0000$_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CAIF | CRIF | CTBIF | CTSIF | CDLIF | CRSIF | CST9 | CST8 | CST7 | CST6 | CST5 | CST4 | CST3 | CST2 | CST1 | CST0 |
| w | w | w | w | w | w | w | w | w | w | w | w | w | w | w | w |

| Field | Bits | Type | Description |
|---|---|---|---|
| **CSTx (x = 0-9)** | x | w | **Clear Status Flag x in PSR**<br>$0_B$ No action<br>$1_B$ Flag PSR.STx is cleared. |
| **CRSIF** | 10 | w | **Clear Receiver Start Indication Flag**<br>$0_B$ No action<br>$1_B$ Flag PSR.RSIF is cleared. |
| **CDLIF** | 11 | w | **Clear Data Lost Indication Flag**<br>$0_B$ No action<br>$1_B$ Flag PSR.DLIF is cleared. |

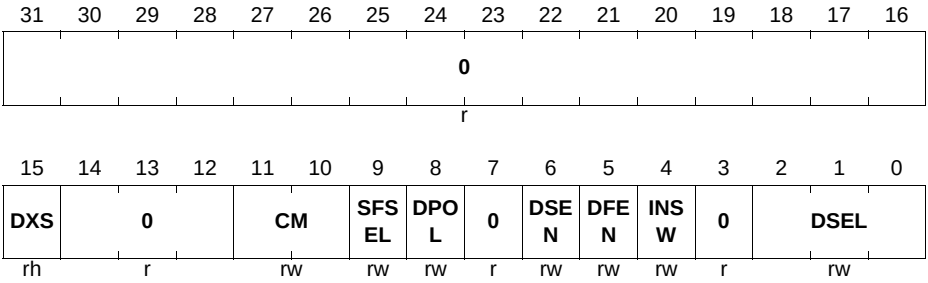
| Field | Bits | Type | Description |
|---|---|---|---|
| **CTSIF** | 12 | w | **Clear Transmit Shift Indication Flag**<br>$0_B$ No action<br>$1_B$ Flag PSR.TSIF is cleared. |
| **CTBIF** | 13 | w | **Clear Transmit Buffer Indication Flag**<br>$0_B$ No action<br>$1_B$ Flag PSR.TBIF is cleared. |
| **CRIF** | 14 | w | **Clear Receive Indication Flag**<br>$0_B$ No action<br>$1_B$ Flag PSR.RIF is cleared. |
| **CAIF** | 15 | w | **Clear Alternative Receive Indication Flag**<br>$0_B$ No action<br>$1_B$ Flag PSR.AIF is cleared. |
| **CBRGIF** | 16 | w | **Clear Baud Rate Generator Indication Flag**<br>$0_B$ No action<br>$1_B$ Flag PSR.BRGIF is cleared. |
| **0** | [31:17] | r | **Reserved;**<br>read as 0; should be written with 0; |

### 15.11.5 Input Stage Register

#### 15.11.5.1 Input Control Registers

The input control registers contain the bits to define the characteristics of the input stages (input stage DX0 is controlled by register DX0CR, etc.).

**DX0CR**
**Input Control Register 0** **($1C_H$)** **Reset Value: 0000 0000$_H$**
**DX2CR**
**Input Control Register 2** **($24_H$)** **Reset Value: 0000 0000$_H$**
**DX3CR**
**Input Control Register 3** **($28_H$)** **Reset Value: 0000 0000$_H$**
**DX4CR**
**Input Control Register 4** **($2C_H$)** **Reset Value: 0000 0000$_H$**
**DX5CR**
**Input Control Register 5** **($30_H$)** **Reset Value: 0000 0000$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **DSEL** | [2:0] | rw | **Data Selection for Input Signal**<br>This bit field defines the input data signal for the corresponding input line for protocol pre-processor. The selection can be made from the input vector DXn[G:A].<br>$000_B$ The data input DXnA is selected.<br>$001_B$ The data input DXnB is selected.<br>$010_B$ The data input DXnC is selected.<br>$011_B$ The data input DXnD is selected.<br>$100_B$ The data input DXnE is selected.<br>$101_B$ The data input DXnF is selected.<br>$110_B$ The data input DXnG is selected.<br>$111_B$ The data input is always 1. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **INSW** | 4 | rw | **Input Switch**<br>This bit defines if the data shift unit input is derived from the input data path DXn or from the selected protocol pre-processors.<br>$0_B$ The input of the data shift unit is controlled by the protocol pre-processor.<br>$1_B$ The input of the data shift unit is connected to the selected data input line. This setting is used if the signals are directly derived from an input pin without treatment by the protocol pre-processor. |
| **DFEN** | 5 | rw | **Digital Filter Enable**<br>This bit enables/disables the digital filter for signal DXnS.<br>$0_B$ The input signal is not digitally filtered.<br>$1_B$ The input signal is digitally filtered. |
| **DSEN** | 6 | rw | **Data Synchronization Enable**<br>This bit selects if the asynchronous input signal or the synchronized (and optionally filtered) signal DXnS can be used as input for the data shift unit.<br>$0_B$ The un-synchronized signal can be taken as input for the data shift unit.<br>$1_B$ The synchronized signal can be taken as input for the data shift unit. |
| **DPOL** | 8 | rw | **Data Polarity for DXn**<br>This bit defines the signal polarity of the input signal.<br>$0_B$ The input signal is not inverted.<br>$1_B$ The input signal is inverted. |
| **SFSEL** | 9 | rw | **Sampling Frequency Selection**<br>This bit defines the sampling frequency of the digital filter for the synchronized signal DXnS.<br>$0_B$ The sampling frequency is $f_{PERIPH}$.<br>$1_B$ The sampling frequency is $f_{FD}$. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **CM** | [11:10] | rw | **Combination Mode**<br>This bit field selects which edge of the synchronized (and optionally filtered) signal DXnS actives the trigger output DXnT of the input stage.<br>$00_B$ The trigger activation is disabled.<br>$01_B$ A rising edge activates DXnT.<br>$10_B$ A falling edge activates DXnT.<br>$11_B$ Both edges activate DXnT. |
| **DXS** | 15 | rh | **Synchronized Data Value**<br>This bit indicates the value of the synchronized (and optionally filtered) input signal.<br>$0_B$ The current value of DXnS is 0.<br>$1_B$ The current value of DXnS is 1. |
| **0** | 3, 7, [14:12], [31:16] | r | **Reserved**<br>Read as 0; should be written with 0. |

**DX1CR**
**Input Control Register 1** **($20_H$)** **Reset Value: 0000 0000$_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **DXS** | **0** | | | **CM** | | **SFSEL** | **DPOL** | **0** | **DSEN** | **DFEN** | **INSW** | **DCEN** | **DSEL** | | |
| rh | r | | | rw | | rw | rw | r | rw | rw | rw | rw | rw | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **DSEL** | [2:0] | rw | **Data Selection for Input Signal**<br>This bit field defines the input data signal for the corresponding input line for protocol pre-processor. The selection can be made from the input vector DX1[G:A].<br>$000_B$ The data input DX1A is selected.<br>$001_B$ The data input DX1B is selected.<br>$010_B$ The data input DX1C is selected.<br>$011_B$ The data input DX1D is selected.<br>$100_B$ The data input DX1E is selected.<br>$101_B$ The data input DX1F is selected.<br>$110_B$ The data input DX1G is selected.<br>$111_B$ The data input is always 1. |
| **DCEN** | 3 | rw | **Delay Compensation Enable**<br>This bit selects if the receive shift clock is controlled by INSW or derived from the input data path DX1.<br>$0_B$ The receive shift clock is dependent on INSW selection.<br>$1_B$ The receive shift clock is connected to the selected data input line. This setting is used if delay compensation is required in SSC and IIS protocols, else DCEN should always be 0. |
| **INSW** | 4 | rw | **Input Switch**<br>This bit defines if the data shift unit input is derived from the input data path DX1 or from the selected protocol pre-processors.<br>$0_B$ The input of the data shift unit is controlled by the protocol pre-processor.<br>$1_B$ The input of the data shift unit is connected to the selected data input line. This setting is used if the signals are directly derived from an input pin without treatment by the protocol pre-processor. |
| **DFEN** | 5 | rw | **Digital Filter Enable**<br>This bit enables/disables the digital filter for signal DX1S.<br>$0_B$ The input signal is not digitally filtered.<br>$1_B$ The input signal is digitally filtered. |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **DSEN** | 6 | rw | **Data Synchronization Enable**<br>This bit selects if the asynchronous input signal or the synchronized (and optionally filtered) signal DX1S can be used as input for the data shift unit.<br>$0_B$ The un-synchronized signal can be taken as input for the data shift unit.<br>$1_B$ The synchronized signal can be taken as input for the data shift unit. |
| **DPOL** | 8 | rw | **Data Polarity for DXn**<br>This bit defines the signal polarity of the input signal.<br>$0_B$ The input signal is not inverted.<br>$1_B$ The input signal is inverted. |
| **SFSEL** | 9 | rw | **Sampling Frequency Selection**<br>This bit defines the sampling frequency of the digital filter for the synchronized signal DX1S.<br>$0_B$ The sampling frequency is $f_{PERIPH}$.<br>$1_B$ The sampling frequency is $f_{FD}$. |
| **CM** | [11:10] | rw | **Combination Mode**<br>This bit field selects which edge of the synchronized (and optionally filtered) signal DX1S actives the trigger output DX1T of the input stage.<br>$00_B$ The trigger activation is disabled.<br>$01_B$ A rising edge activates DX1T.<br>$10_B$ A falling edge activates DX1T.<br>$11_B$ Both edges activate DX1T. |
| **DXS** | 15 | rh | **Synchronized Data Value**<br>This bit indicates the value of the synchronized (and optionally filtered) input signal.<br>$0_B$ The current value of DX1S is 0.<br>$1_B$ The current value of DX1S is 1. |
| **0** | 7, [14:12], [31:16] | r | **Reserved**<br>Read as 0; should be written with 0. |


Subject to Agreement on the Use of Product Information

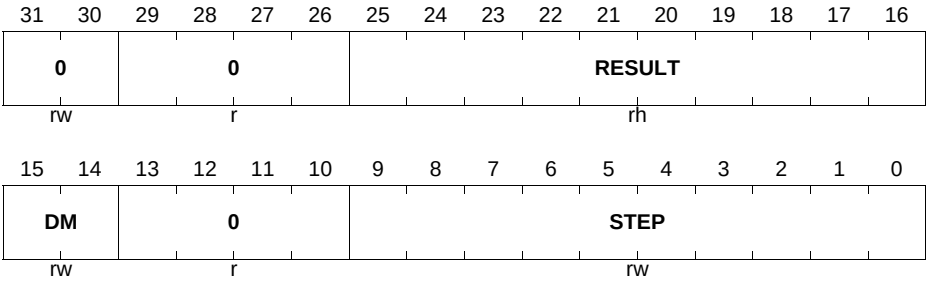
## 15.11.6 Baud Rate Generator Registers

### 15.11.6.1 Fractional Divider Register

The fractional divider register FDR allows the generation of the internal frequency $f_{FD}$, that is derived from the system clock $f_{PERIPH}$.

FDR can be written only with a privilege mode access.

**FDR**
**Fractional Divider Register** **($10_H$)** **Reset Value: 0000 0000$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **STEP** | [9:0] | rw | **Step Value**<br>In normal divider mode STEP contains the reload value for RESULT after RESULT has reached 3FF$_H$.<br>In fractional divider mode STEP defines the value added to RESULT with each input clock cycle. |
| **DM** | [15:14] | rw | **Divider Mode**<br>This bit fields defines the functionality of the fractional divider block.<br>$00_B$ The divider is switched off, $f_{FD}$ = 0.<br>$01_B$ Normal divider mode selected.<br>$10_B$ Fractional divider mode selected.<br>$11_B$ The divider is switched off, $f_{FD}$ = 0. |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **RESULT** | [25:16] | rh | **Result Value**<br>In normal divider mode this bit field is updated with $f_{PERIPH}$ according to: RESULT = RESULT + 1<br>In fractional divider mode this bit field is updated with $f_{PERIPH}$ according to: RESULT = RESULT + STEP<br>If bit field DM is written with $01_B$ or $10_B$, RESULT is loaded with a start value of $3FF_H$. |
| **0** | [31:30] | rw | **Reserved for Future Use**<br>Must be written with 0 to allow correct fractional divider operation. |
| **0** | [13:10], [29:26] | r | **Reserved**<br>Read as 0; should be written with 0. |

### 15.11.6.2 Baud Rate Generator Register

The protocol-related counters for baud rate generation and timing measurement are controlled by the register BRG.

FDR can be written only with a privilege mode access.

**BRG**
**Baud Rate Generator Register ($14_H$) Reset Value: $0000\ 0000_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SCLKCFG | | MCLKCFG | SCLKOSEL | 0 | | PDIV | | | | | | | | | |
| rw | | rw | rw | r | | rw | | | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | DCTQ | | | | | PCTQ | | CTQSEL | | 0 | PPPEN | TMEN | 0 | CLKSEL | |
| r | rw | | | | | rw | | rw | | r | rw | rw | r | rw | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **CLKSEL** | [1:0] | rw | **Clock Selection**<br>This bit field defines the input frequency $f_{PIN}$<br>$00_B$ The fractional divider frequency $f_{FD}$ is selected.<br>$01_B$ Reserved, no action<br>$10_B$ The trigger signal DX1T defines $f_{PIN}$. Signal MCLK toggles with $f_{PIN}$.<br>$11_B$ Signal MCLK corresponds to the DX1S signal and the frequency $f_{PIN}$ is derived from the rising edges of DX1S. |
| **TMEN** | 3 | rw | **Timing Measurement Enable**<br>This bit enables the timing measurement of the capture mode timer.<br>$0_B$ Timing measurement is disabled:<br>The trigger signals DX0T and DX1T are ignored.<br>$1_B$ Timing measurement is enabled:<br>The 10-bit counter is incremented by 1 with $f_{PPP}$ and stops counting when reaching its maximum value. If one of the trigger signals DX0T or DX1T become active, the counter value is captured into bit field CTV, the counter is cleared and a transmit shift event is generated. |
| **PPPEN** | 4 | rw | **Enable 2:1 Divider for $f_{PPP}$**<br>This bit defines the input frequency $f_{PPP}$.<br>$0_B$ The 2:1 divider for $f_{PPP}$ is disabled.<br>$f_{PPP} = f_{PIN}$<br>$1_B$ The 2:1 divider for $f_{PPP}$ is enabled.<br>$f_{PPP} = f_{MCLK} = f_{PIN} / 2$. |
| **CTQSEL** | [7:6] | rw | **Input Selection for CTQ**<br>This bit defines the length of a time quantum for the protocol pre-processor.<br>$00_B$ $f_{CTQIN} = f_{PDIV}$<br>$01_B$ $f_{CTQIN} = f_{PPP}$<br>$10_B$ $f_{CTQIN} = f_{SCLK}$<br>$11_B$ $f_{CTQIN} = f_{MCLK}$ |

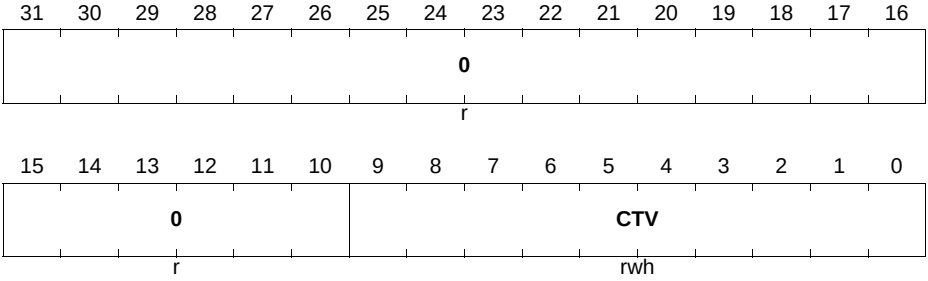

Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **PCTQ** | [9:8] | rw | **Pre-Divider for Time Quanta Counter**<br>This bit field defines length of a time quantum tq for the time quanta counter in the protocol pre-processor.<br>$t_Q$ = (PCTQ + 1) / $f_{CTQIN}$ |
| **DCTQ** | [14:10] | rw | **Denominator for Time Quanta Counter**<br>This bit field defines the number of time quanta $t_q$ taken into account by the time quanta counter in the protocol pre-processor. |
| **PDIV** | [25:16] | rw | **Divider Mode: Divider Factor to Generate $f_{PDIV}$**<br>This bit field defines the ratio between the input frequency $f_{PPP}$ and the divider frequency $f_{PDIV}$. |
| **SCLKOSEL** | 28 | rw | **Shift Clock Output Select**<br>This bit field selects the input source for the SCLKOUT signal.<br>$0_B$ SCLK from the baud rate generator is selected as the SCLKOUT input source.<br>$1_B$ The transmit shift clock from DX1 input stage is selected as the SCLKOUT input source.<br>*Note: The setting SCLKOSEL = 1 is used only when complete closed loop delay compensation is required for a slave SSC/IIS. The default setting of SCLKOSEL = 0 should be always used for all other cases.* |
| **MCLKCFG** | 29 | rw | **Master Clock Configuration**<br>This bit field defines the level of the passive phase of the MCLKOUT signal.<br>$0_B$ The passive level is 0.<br>$1_B$ The passive level is 1. |
| **SCLKCFG** | [31:30] | rw | **Shift Clock Output Configuration**<br>This bit field defines the level of the passive phase of the SCLKOUT signal and enables/disables a delay of half of a SCLK period.<br>$00_B$ The passive level is 0 and the delay is disabled.<br>$01_B$ The passive level is 1 and the delay is disabled.<br>$10_B$ The passive level is 0 and the delay is enabled.<br>$11_B$ The passive level is 1 and the delay is enabled. |
| **0** | 2, 5, 15, [27:26] | r | **Reserved**<br>Read as 0; should be written with 0. |

### 15.11.6.3 Capture Mode Timer Register

The captured timer value is provided by the register CMTR.

**CMTR**
**Capture Mode Timer Register** **($44_H$)** **Reset Value: 0000 0000$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **CTV** | [9:0] | rwh | **Captured Timer Value**<br>The value of the counter is captured into this bit field if one of the trigger signals DX0T or DX1T are activated by the corresponding input stage. |
| **0** | [31:10] | r | **Reserved**<br>Read as 0; should be written with 0. |

## 15.11.7 Transfer Control and Status Registers

### 15.11.7.1 Shift Control Register

The data shift unit is controlled by the register SCTR. The values in this register are applied for data transmission and reception.

Please note that the shift control settings SDIR, WLE, FLE, DSM and HPCDIR are shared between transmitter and receiver. They are internally “frozen” for a each data word transfer in the transmitter with the first transmit shift clock edge and with the first receive shift clock edge in the receiver. The software has to take care that updates of these bit fields by software are done coherently (e.g. refer to the receiver start event indication PSR.RSIF).

**SCTR**
**Shift Control Register** **($34_H$)** **Reset Value: 0000 0000$_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | WLE | | | | 0 | | FLE | | | | | |
| r | | | | rwh | | | | r | | rwh | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | TRM | | DOCFG | | 0 | HPC DIR | DSM | | PDL | SDIR |
| r | | | | | | rw | | rw | | r | rw | rw | | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **SDIR** | 0 | rw | **Shift Direction**<br>This bit defines the shift direction of the data words for transmission and reception.<br>$0_B$ Shift LSB first. The first data bit of a data word is located at bit position 0.<br>$1_B$ Shift MSB first. The first data bit of a data word is located at the bit position given by bit field SCTR.WLE. |
| **PDL** | 1 | rw | **Passive Data Level**<br>This bit defines the output level at the shift data output signal when no data is available for transmission. The PDL level is output with the first relevant transmit shift clock edge of a data word.<br>$0_B$ The passive data level is 0.<br>$1_B$ The passive data level is 1. |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **DSM** | [3:2] | rw | **Data Shift Mode**<br>This bit field describes how the receive and transmit data is shifted in and out.<br>$00_B$ Receive and transmit data is shifted in and out one bit at a time through DX0 and DOUT0.<br>$01_B$ Reserved.<br>$10_B$ Receive and transmit data is shifted in and out two bits at a time through two input stages (DX0 and DX3) and DOUT[1:0] respectively.<br>$11_B$ Receive and transmit data is shifted in and out four bits at a time through four input stages (DX0, DX[5:3]) and DOUT[3:0] respectively.<br>*Note: Dual- and Quad-output modes are used only by the SSC protocol. For all other protocols DSM must always be written with $00_B$.* |
| **HPCDIR** | 4 | rw | **Port Control Direction**<br>This bit defines the direction of the port pin(s) which allows hardware pin control (CCR.PCEN = 1).<br>$0_B$ The pin(s) with hardware pin control enabled are selected to be in input mode.<br>$1_B$ The pin(s) with hardware pin control enabled are selected to be in output mode. |
| **DOCFG** | [7:6] | rw | **Data Output Configuration**<br>This bit defines the relation between the internal shift data value and the data output signal DOUTx.<br>$X0_B$ DOUTx = shift data value<br>$X1_B$ DOUTx = inverted shift data value |

| Field | Bits | Type | Description |
|---|---|---|---|
| **TRM** | [9:8] | rw | **Transmission Mode**<br>This bit field describes how the shift control signal is interpreted by the DSU. Data transfers are only possible while the shift control signal is active.<br>$00_B$ The shift control signal is considered as inactive and data frame transfers are not possible.<br>$01_B$ The shift control signal is considered active if it is at 1-level. This is the setting to be programmed to allow data transfers.<br>$10_B$ The shift control signal is considered active if it is at 0-level. It is recommended to avoid this setting and to use the inversion in the DX2 stage in case of a low-active signal.<br>$11_B$ The shift control signal is considered active without referring to the actual signal level. Data frame transfer is possible after each edge of the signal. |
| **FLE** | [21:16] | rwh | **Frame Length**<br>This bit field defines how many bits are transferred within a data frame. A data frame can consist of several concatenated data words.<br>If TCSR.FLEMD = 1, the value can be updated automatically by the data handler. |
| **WLE** | [27:24] | rwh | **Word Length**<br>This bit field defines the data word length (amount of bits that are transferred in each data word) for reception and transmission. The data word is always right-aligned in the data buffer at the bit positions [WLE down to 0].<br>If TCSR.WLEMD = 1, the value can be updated automatically by the data handler.<br>$0_H$ The data word contains 1 data bit located at bit position 0.<br>$1_H$ The data word contains 2 data bits located at bit positions [1:0].<br>...<br>$E_H$ The data word contains 15 data bits located at bit positions [14:0].<br>$F_H$ The data word contains 16 data bits located at bit positions [15:0]. |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | 5, [15:10], [23:22], [31:28] | r | **Reserved**<br>Read as 0; should be written with 0. |

### 15.11.7.2 Transmission Control and Status Register

The data transmission is controlled and monitored by register TCSR.

**TCSR**
**Transmit Control/Status Register (38$_H$) Reset Value: 0000 0000$_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | | | **TE** | **TVC** | **TV** | **0** | **TSOF** | **0** | | | | | | | |
| r | | | rh | rh | rh | r | rh | r | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | | **WA** | **TDVTR** | **TDEN** | | **0** | **TDSSM** | **TDV** | **EOF** | **SOF** | **HPCMD** | **WAMD** | **FLEMD** | **SELMD** | **WLEMD** |
| r | | rwh | rw | rw | | r | rw | rh | rwh | rwh | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **WLEMD** | 0 | rw | **WLE Mode**<br>This bit enables the data handler to automatically update the bit field SCTR.WLE by the transmit control information TCI[3:0] and bit TCSR.EOF by TCI[4] (see **Page 15-33**). If enabled, an automatic update takes place when new data is loaded to register TBUF, either by writing to one of the transmit buffer input locations TBUFx or by an optional data buffer.<br>$0_B$ The automatic update of SCTR.WLE and TCSR.EOF is disabled.<br>$1_B$ The automatic update of SCTR.WLE and TCSR.EOF is enabled. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **SELMD** | 1 | rw | **Select Mode**<br>This bit can be used mainly for the SSC protocol. It enables the data handler to automatically update bit field PCR.CTR[20:16] by the transmit control information TCI[4:0] and clear bit field PCR.CTR[23:21] (see **Page 15-33**). If enabled, an automatic update takes place when new data is loaded to register TBUF, either by writing to one of the transmit buffer input locations TBUFx or by an optional data buffer.<br>$0_B$ The automatic update of PCR.CTR[23:16] is disabled.<br>$1_B$ The automatic update of PCR.CTR[23:16] is disabled. |
| **FLEMD** | 2 | rw | **FLE Mode**<br>This bit enables the data handler to automatically update bits SCTR.FLE[4:0] by the transmit control information TCI[4:0] and to clear bit SCTR.FLE[5] (see **Page 15-33**). If enabled, an automatic update takes place when new data is loaded to register TBUF, either by writing to one of the transmit buffer input locations TBUFx or by an optional data buffer.<br>$0_B$ The automatic update of FLE is disabled.<br>$1_B$ The automatic update of FLE is enabled. |
| **WAMD** | 3 | rw | **WA Mode**<br>This bit can be used mainly for the IIS protocol. It enables the data handler to automatically update bit TCSR.WA by the transmit control information TCI[4] (see **Page 15-33**). If enabled, an automatic update takes place when new data is loaded to register TBUF, either by writing to one of the transmit buffer input locations TBUFx or by an optional data buffer.<br>$0_B$ The automatic update of bit WA is disabled.<br>$1_B$ The automatic update of bit WA is enabled. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **HPCMD** | 4 | rw | **Hardware Port Control Mode**<br>This bit can be used mainly for the dual and quad SSC protocol. It enables the data handler to automatically update bit SCTR.DSM by the transmit control information TCI[1:0] and bit SCTR.HPCDIR by TCI[2] (see **Page 15-33**). If enabled, an automatic update takes place when new data is loaded to register TBUF, either by writing to one of the transmit buffer input locations TBUFx or by an optional data buffer.<br>$0_B$ The automatic update of bits SCTR.DSM and SCTR.HPCDIR is disabled.<br>$1_B$ The automatic update of bits SCTR.DSM and SCTR.HPCDIR is enabled. |
| **SOF** | 5 | rwh | **Start Of Frame**<br>This bit is only taken into account for the SSC protocol, otherwise it is ignored.<br>It indicates that the data word in TBUF is considered as the first word of a new SSC frame if it is valid for transmission (TCSR.TDV = 1). This bit becomes cleared when the TBUF data word is transferred to the transmit shift register.<br>$0_B$ The data word in TBUF is not considered as first word of a frame.<br>$1_B$ The data word in TBUF is considered as first word of a frame. A currently running frame is finished and MSLS becomes deactivated (respecting the programmed delays). |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **EOF** | 6 | rwh | **End Of Frame**<br>This bit is only taken into account for the SSC protocol, otherwise it is ignored. It can be modified automatically by the data handler if bit WLEMD = 1. It indicates that the data word in TBUF is considered as the last word of an SSC frame. If it is the last word, the MSLS signal becomes inactive after the transfer, respecting the programmed delays. This bit becomes cleared when the TBUF data word is transferred to the transmit shift register.<br>$0_B$ The data word in TBUF is not considered as last word of an SSC frame.<br>$1_B$ The data word in TBUF is considered as last word of an SSC frame. |
| **TDV** | 7 | rh | **Transmit Data Valid**<br>This bit indicates that the data word in the transmit buffer TBUF can be considered as valid for transmission. The TBUF data word can only be sent out if TDV = 1. It is automatically set when data is moved to TBUF (by writing to one of the transmit buffer input locations TBUFx, or optionally, by the bypass or FIFO mechanism).<br>$0_B$ The data word in TBUF is not valid for transmission.<br>$1_B$ The data word in TBUF is valid for transmission and a transmission start is possible. New data should not be written to a TBUFx input location while TDV = 1. |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **TDSSM** | 8 | rw | **TBUF Data Single Shot Mode**<br>This bit defines if the data word TBUF data is considered as permanently valid or if the data should only be transferred once.<br>$0_B$ The data word in TBUF is not considered as invalid after it has been loaded into the transmit shift register. The loading of the TBUF data into the shift register does not clear TDV.<br>$1_B$ The data word in TBUF is considered as invalid after it has been loaded into the shift register. In ASC and IIC mode, TDV is cleared with the TBI event, whereas in SSC and IIS mode, it is cleared with the RSI event.<br>TDSSM = 1 has to be programmed if an optional data buffer is used. |
| **TDEN** | [11:10] | rw | **TBUF Data Enable**<br>This bit field controls the gating of the transmission start of the data word in the transmit buffer TBUF.<br>$00_B$ A transmission start of the data word in TBUF is disabled. If a transmission is started, the passive data level is sent out.<br>$01_B$ A transmission of the data word in TBUF can be started if TDV = 1.<br>$10_B$ A transmission of the data word in TBUF can be started if TDV = 1 while DX2S = 0.<br>$11_B$ A transmission of the data word in TBUF can be started if TDV = 1 while DX2S = 1. |
| **TDVTR** | 12 | rw | **TBUF Data Valid Trigger**<br>This bit enables the transfer trigger unit to set bit TCSR.TE if the trigger signal DX2T becomes active for event driven transfer starts, e.g. timer-based or depending on an event at an input pin. Bit TDVTR has to be 0 for protocols where the input stage DX2 is used for data shifting.<br>$0_B$ Bit TCSR.TE is permanently set.<br>$1_B$ Bit TCSR.TE is set if DX2T becomes active while TDV = 1. |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **WA** | 13 | rwh | **Word Address**<br>This bit is only taken into account for the IIS protocol, otherwise it is ignored. It can be modified automatically by the data handler if bit WAMD = 1. Bit WA defines for which channel the data stored in TBUF will be transmitted.<br>$0_B$ The data word in TBUF will be transmitted after a falling edge of WA has been detected (referring to PSR.WA).<br>$1_B$ The data word in TBUF will be transmitted after a rising edge of WA has been detected (referring to PSR.WA). |
| **TSOF** | 24 | rh | **Transmitted Start Of Frame**<br>This bit indicates if the latest start of a data word transmission has taken place for the first data word of a new data frame. This bit is updated with the transmission start of each data word.<br>$0_B$ The latest data word transmission has not been started for the first word of a data frame.<br>$1_B$ The latest data word transmission has been started for the first word of a data frame. |
| **TV** | 26 | rh | **Transmission Valid**<br>This bit represents the transmit buffer underflow and indicates if the latest start of a data word transmission has taken place with a valid data word from the transmit buffer TBUF. This bit is updated with the transmission start of each data word.<br>$0_B$ The latest start of a data word transmission has taken place while no valid data was available. As a result, the transmission of a data words with passive level (SCTR.PDL) has been started.<br>$1_B$ The latest start of a data word transmission has taken place with valid data from TBUF. |


Subject to Agreement on the Use of Product Information

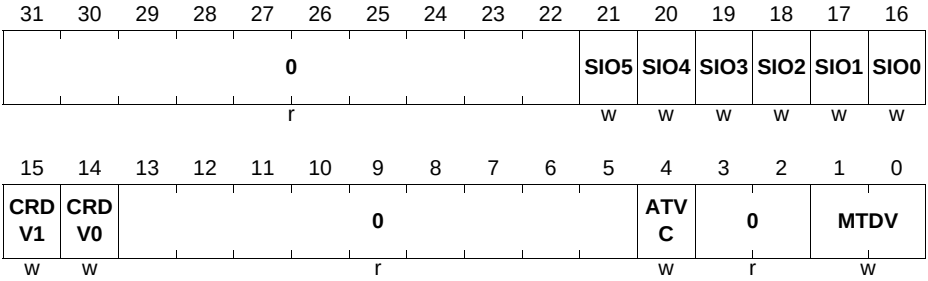
| Field | Bits | Type | Description |
|---|---|---|---|
| **TVC** | 27 | rh | **Transmission Valid Cumulated**<br>This bit cumulates the transmit buffer underflow indication TV. It is cleared automatically together with bit TV and has to be set by writing FMR.ATVC = 1.<br>$0_B$ Since TVC has been set, at least one data buffer underflow condition has occurred.<br>$1_B$ Since TVC has been set, no data buffer underflow condition has occurred. |
| **TE** | 28 | rh | **Trigger Event**<br>If the transfer trigger mechanism is enabled, this bit indicates that a trigger event has been detected (DX2T = 1) while TCSR.TDV = 1. If the event trigger mechanism is disabled, the bit TE is permanently set. It is cleared by writing FMR.MTDV = $10_B$ or when the data word located in TBUF is loaded into the shift register.<br>$0_B$ The trigger event has not yet been detected. A transmission of the data word in TBUF can not be started.<br>$1_B$ The trigger event has been detected (or the trigger mechanism is switched off) and a transmission of the data word in TBUF can be started. |
| **0** | 9, [23:14], 25, [31:29] | r | **Reserved**<br>Read as 0; should be written with 0. |

### 15.11.7.3 Flag Modification Registers

The flag modification register FMR allows the modification of control and status flags related to data handling by using only write accesses. Read accesses to FMR always deliver 0 at all bit positions.

Additionally, the service request outputs of this USIC channel can be activated by software (the activation is triggered by the write access and is deactivated automatically).

**FMR**
**Flag Modification Register** **($68_H$)** **Reset Value: 0000 0000$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **MTDV** | [1:0] | w | **Modify Transmit Data Valid**<br>Writing to this bit field can modify bits TCSR.TDV and TCSR.TE to control the start of a data word transmission by software.<br>$00_B$ No action.<br>$01_B$ Bit TDV is set, TE is unchanged.<br>$10_B$ Bits TDV and TE are cleared.<br>$11_B$ Reserved |
| **ATVC** | 4 | w | **Activate Bit TVC**<br>Writing to this bit can set bit TCSR.TVC to start a new cumulation of the transmit buffer underflow condition.<br>$0_B$ No action.<br>$1_B$ Bit TCSR.TVC is set. |
| **CRDV0** | 14 | w | **Clear Bits RDV for RBUF0**<br>Writing 1 to this bit clears bits RBUF01SR.RDV00 and RBUF01SR.RDV10 to declare the received data in RBUF0 as no longer valid (to emulate a read action).<br>$0_B$ No action.<br>$1_B$ Bits RBUF01SR.RDV00 and RBUF01SR.RDV10 are cleared. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **CRDV1** | 15 | w | **Clear Bit RDV for RBUF1**<br>Writing 1 to this bit clears bits RBUF01SR.RDV01 and RBUF01SR.RDV11 to declare the received data in RBUF1 as no longer valid (to emulate a read action).<br>$0_B$ No action.<br>$1_B$ Bits RBUF01SR.RDV01 and RBUF01SR.RDV11 are cleared. |
| **SIO0,**<br>**SIO1,**<br>**SIO2,**<br>**SIO3,**<br>**SIO4,**<br>**SIO5** | 16,<br>17,<br>18,<br>19,<br>20,<br>21 | w | **Set Interrupt Output SRx**<br>Writing a 1 to this bit field activates the service request output SRx of this USIC channel. It has no impact on service request outputs of other USIC channels.<br>$0_B$ No action.<br>$1_B$ The service request output SRx is activated. |
| **0** | [3:2],<br>[13:5],<br>[31:22] | r | **Reserved**<br>Read as 0; should be written with 0. |

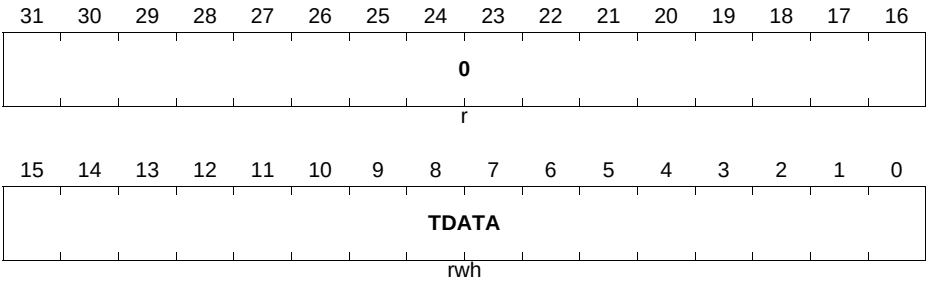
## 15.11.8 Data Buffer Registers

### 15.11.8.1 Transmit Buffer Locations

The 32 independent data input locations TBUF00 to TBUF31 are address locations that can be used as data entry locations for the transmit buffer. Data written to one of these locations will appear in a common register TBUF. Additionally, the 5 bit coding of the number [31:0] of the addressed data input location represents the transmit control information TCI (please refer to the protocol sections for more details).
The internal transmit buffer register TBUF contains the data that will be loaded to the transmit shift register for the next transmission of a data word. It can be read out at all TBUF00 to TBUF31 addresses.


Subject to Agreement on the Use of Product Information

**TBUFx (x = 00-31)**
**Transmit Buffer Input Location x ($80_H$ + x*4) Reset Value: 0000 $0000_H$**

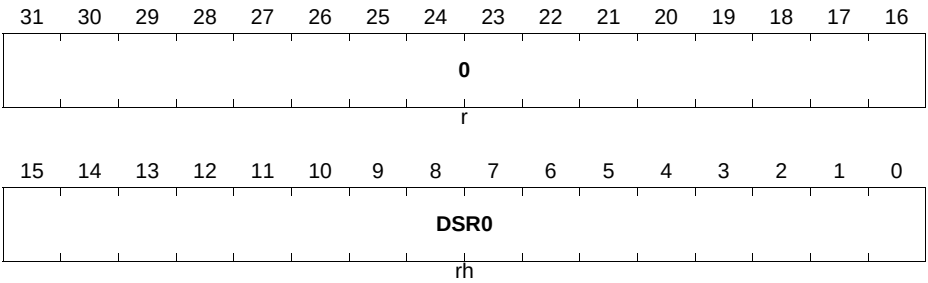



| Field | Bits | Type | Description |
|---|---|---|---|
| **TDATA** | [15:0] | rwh | **Transmit Data**<br>This bit field contains the data to be transmitted (read view).<br>A data write action to at least the low byte of TDATA sets TCSR.TDV. |
| **0** | [31:16] | r | **Reserved**<br>Read as 0; should be written with 0. |

### 15.11.8.2 Receive Buffer Registers RBUF0, RBUF1

The receive buffer register RBUF0 contains the data received from RSR0[3:0]. A read action does not change the status of the receive data from “not yet read = valid” to “already read = not valid”.

**RBUF0**
**Receiver Buffer Register 0 ($5C_H$) Reset Value: 0000 $0000_H$**



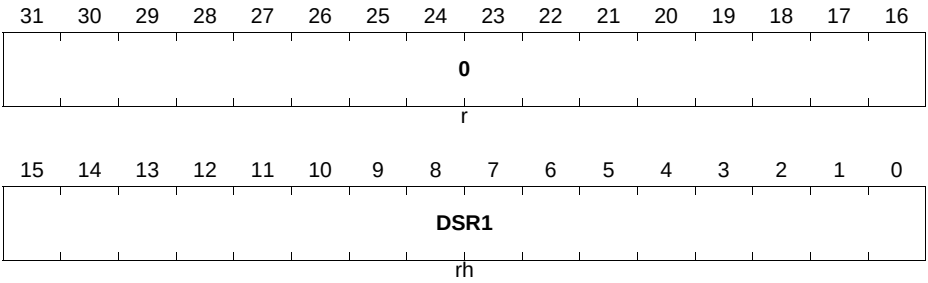


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **DSR0** | [15:0] | rh | **Data of Shift Registers 0[3:0]** |
| **0** | [31:16] | r | **Reserved**<br>Read as 0; should be written with 0. |

The receive buffer register RBUF1 contains the data received from RSR1[3:0]. A read action does not change the status of the receive data from “not yet read = valid” to “already read = not valid”.

**RBUF1**
**Receiver Buffer Register 1** **($60_H$)** **Reset Value: 0000 $0000_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **DSR1** | [15:0] | rh | **Data of Shift Registers 1[3:0]** |
| **0** | [31:16] | r | **Reserved**<br>Read as 0; should be written with 0. |

The receive buffer status register RBUF01SR provides the status of the data in receive buffers RBUF0 and RBUF1.


Subject to Agreement on the Use of Product Information

**RBUF01SR**

**Receiver Buffer 01 Status Register ($64_H$) Reset Value: 0000 0000$_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| DS1 | RDV 11 | RDV 10 | 0 | | | PER R1 | PAR 1 | 0 | SOF 1 | 0 | | WLEN1 | | | |
| rh | rh | rh | r | | | rh | rh | r | rh | r | | rh | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| DS0 | RDV 01 | RDV 00 | 0 | | | PER R0 | PAR 0 | 0 | SOF 0 | 0 | | WLEN0 | | | |
| rh | rh | rh | r | | | rh | rh | r | rh | r | | rh | | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **WLEN0** | [3:0] | rh | **Received Data Word Length in RBUF0**<br>This bit field indicates how many bits have been received within the last data word stored in RBUF0. This number indicates how many data bits have to be considered as receive data, whereas the other bits in RBUF0 have been cleared automatically. The received bits are always right-aligned.<br>For all protocol modes besides dual and quad SSC,<br>Received data word length = WLEN0 + 1<br>For dual SSC mode,<br>Received data word length = WLEN0 + 2<br>For quad SSC mode,<br>Received data word length = WLEN0 + 4 |
| **SOF0** | 6 | rh | **Start of Frame in RBUF0**<br>This bit indicates whether the data word in RBUF0 has been the first data word of a data frame.<br>$0_B$ The data in RBUF0 has not been the first data word of a data frame.<br>$1_B$ The data in RBUF0 has been the first data word of a data frame. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **PAR0** | 8 | rh | **Protocol-Related Argument in RBUF0**<br>This bit indicates the value of the protocol-related argument. This value is elaborated depending on the selected protocol and adds additional information to the data word in RBUF0.<br>The meaning of this bit is described in the corresponding protocol chapter. |
| **PERR0** | 9 | rh | **Protocol-related Error in RBUF0**<br>This bit indicates if the value of the protocol-related argument meets an expected value. This value is elaborated depending on the selected protocol and adds additional information to the data word in RBUF0.<br>The meaning of this bit is described in the corresponding protocol chapter.<br>$0_B$ The received protocol-related argument PAR matches the expected value. The reception of the data word sets bit PSR.RIF and can generate a receive interrupt.<br>$1_B$ The received protocol-related argument PAR does not match the expected value. The reception of the data word sets bit PSR.AIF and can generate an alternative receive interrupt. |
| **RDV00** | 13 | rh | **Receive Data Valid in RBUF0**<br>This bit indicates the status of the data content of register RBUF0. This bit is identical to bit RBUF01SR.RDV10 and allows consisting reading of information for the receive buffer registers. It is set when a new data word is stored in RBUF0 and automatically cleared if it is read out via RBUF.<br>$0_B$ Register RBUF0 does not contain data that has not yet been read out.<br>$1_B$ Register RBUF0 contains data that has not yet been read out. |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **RDV01** | 14 | rh | **Receive Data Valid in RBUF1**<br>This bit indicates the status of the data content of register RBUF1. This bit is identical to bit RBUF01SR.RDV11 and allows consisting reading of information for the receive buffer registers. It is set when a new data word is stored in RBUF1 and automatically cleared if it is read out via RBUF.<br>$0_B$ Register RBUF1 does not contain data that has not yet been read out.<br>$1_B$ Register RBUF1 contains data that has not yet been read out. |
| **DS0** | 15 | rh | **Data Source**<br>This bit indicates which receive buffer register (RBUF0 or RBUF1) is currently visible in registers RBUF(D) and in RBUFSR for the associated status information. It indicates which buffer contains the oldest data (the data that has been received first). This bit is identical to bit RBUF01SR.DS1 and allows consisting reading of information for the receive buffer registers.<br>$0_B$ The register RBUF contains the data of RBUF0 (same for associated status information).<br>$1_B$ The register RBUF contains the data of RBUF1 (same for associated status information). |
| **WLEN1** | [19:16] | rh | **Received Data Word Length in RBUF1**<br>This bit field indicates how many bits have been received within the last data word stored in RBUF1. This number indicates how many data bits have to be considered as receive data, whereas the other bits in RBUF1 have been cleared automatically. The received bits are always right-aligned.<br>For all protocol modes besides dual and quad SSC,<br>Received data word length = WLEN1 + 1<br>For dual SSC mode,<br>Received data word length = WLEN1 + 2<br>For quad SSC mode,<br>Received data word length = WLEN1 + 4 |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **SOF1** | 22 | rh | **Start of Frame in RBUF1**<br>This bit indicates whether the data word in RBUF1 has been the first data word of a data frame.<br>$0_B$ The data in RBUF1 has not been the first data word of a data frame.<br>$1_B$ The data in RBUF1 has been the first data word of a data frame. |
| **PAR1** | 24 | rh | **Protocol-Related Argument in RBUF1**<br>This bit indicates the value of the protocol-related argument. This value is elaborated depending on the selected protocol and adds additional information to the data word in RBUF1.<br>The meaning of this bit is described in the corresponding protocol chapter. |
| **PERR1** | 25 | rh | **Protocol-related Error in RBUF1**<br>This bit indicates if the value of the protocol-related argument meets an expected value. This value is elaborated depending on the selected protocol and adds additional information to the data word in RBUF1.<br>The meaning of this bit is described in the corresponding protocol chapter.<br>$0_B$ The received protocol-related argument PAR matches the expected value. The reception of the data word sets bit PSR.RIF and can generate a receive interrupt.<br>$1_B$ The received protocol-related argument PAR does not match the expected value. The reception of the data word sets bit PSR.AIF and can generate an alternative receive interrupt. |
| **RDV10** | 29 | rh | **Receive Data Valid in RBUF0**<br>This bit indicates the status of the data content of register RBUF0. This bit is identical to bit RBUF01SR.RDV00 and allows consisting reading of information for the receive buffer registers.<br>$0_B$ Register RBUF0 does not contain data that has not yet been read out.<br>$1_B$ Register RBUF0 contains data that has not yet been read out. |

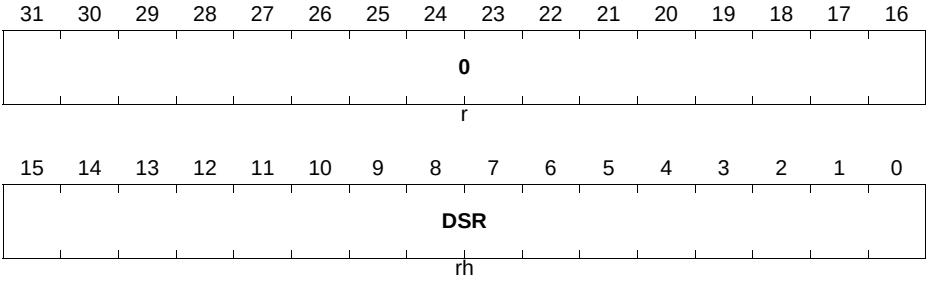

Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **RDV11** | 30 | rh | **Receive Data Valid in RBUF1**<br>This bit indicates the status of the data content of register RBUF1. This bit is identical to bit RBUF01SR.RDV01 and allows consisting reading of information for the receive buffer registers.<br>$0_B$ Register RBUF1 does not contain data that has not yet been read out.<br>$1_B$ Register RBUF1 contains data that has not yet been read out. |
| **DS1** | 31 | rh | **Data Source**<br>This bit indicates which receive buffer register (RBUF0 or RBUF1) is currently visible in registers RBUF(D) and in RBUFSR for the associated status information. It indicates which buffer contains the oldest data (the data that has been received first). This bit is identical to bit RBUF01SR.DS0 and allows consisting reading of information for the receive buffer registers.<br>$0_B$ The register RBUF contains the data of RBUF0 (same for associated status information).<br>$1_B$ The register RBUF contains the data of RBUF1 (same for associated status information). |
| **0** | [5:4], 7, [12:10], [21:20], 23, [28:26] | r | **Reserved**<br>Read as 0; should be written with 0. |

### 15.11.8.3 Receive Buffer Registers RBUF, RBUFD, RBUFSR

The receiver buffer register RBUF shows the content of the either RBUF0 or RBUF1, depending on the order of reception. Always the oldest data (the data word that has been received first) from both receive buffers can be read from RBUF. It is recommended to read out the received data from RBUF instead of RBUF0/1. With a read access of at least the low byte of RBUF, the status of the receive data is automatically changed from “not yet read = valid” to “already read = not valid”, the content of RBUF becomes updated, and the next received data word becomes visible in RBUF.

**RBUF**
**Receiver Buffer Register** **($54_H$)** **Reset Value: 0000 0000$_H$**

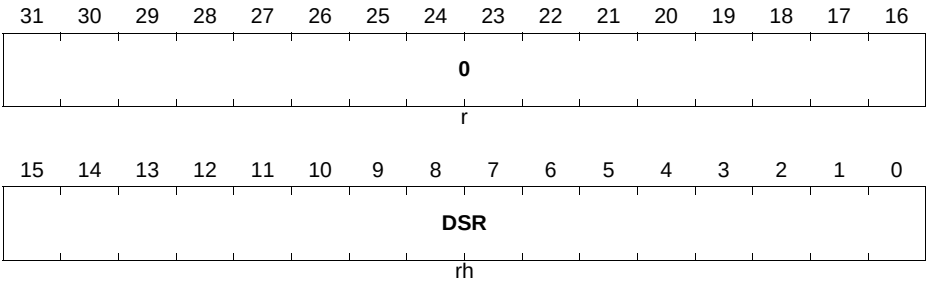



| Field | Bits | Type | Description |
|---|---|---|---|
| **DSR** | [15:0] | rh | **Received Data**<br>This bit field monitors the content of either RBUF0 or RBUF1, depending on the reception sequence. |
| **0** | [31:16] | r | **Reserved**<br>Read as 0; should be written with 0. |


Subject to Agreement on the Use of Product Information

If a debugger should be used to monitor the received data, the automatic update mechanism has to be de-activated to guaranty data consistency. Therefore, the receiver buffer register for debugging RBUFD is available. It is similar to RBUF, but without the automatic update mechanism by a read action. So a debugger (or other monitoring function) can read RBUFD without disturbing the receive sequence.

**RBUFD**
**Receiver Buffer Register for Debugger($58_H$)** **Reset Value: 0000 0000$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **DSR** | [15:0] | rh | **Data from Shift Register**<br>Same as RBUF.DSR, but without releasing the buffer after a read action. |
| **0** | [31:16] | r | **Reserved**<br>Read as 0; should be written with 0. |

The receive buffer status register RBUFSR provides the status of the data in receive buffers RBUF and RBUFD. If bits RBUF01SR.DS0 (or RBUF01SR.DS1) are 0, the lower 16-bit content of RBUF01SR is monitored in RBUFSR, otherwise the upper 16-bit content of RBUF01SR is shown.

**RBUFSR**

**Receiver Buffer Status Register** **($50_H$)** **Reset Value: $0000\ 0000_H$**

| 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| DS | RDV1 | RDV0 | 0 | | | PERR | PAR | 0 | SOF | 0 | | WLEN | | | |
| rh | rh | rh | r | | | rh | rh | r | rh | r | | rh | | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **WLEN** | [3:0] | rh | **Received Data Word Length in RBUF or RBUFD**<br>Description see RBUF01SR.WLEN0 or RBUF01SR.WLEN1. |
| **SOF** | 6 | rh | **Start of Frame in RBUF or RBUFD**<br>Description see RBUF01SR.SOF0 or RBUF01SR.SOF1. |
| **PAR** | 8 | rh | **Protocol-Related Argument in RBUF or RBUFD**<br>Description see RBUF01SR.PAR0 or RBUF01SR.PAR1. |
| **PERR** | 9 | rh | **Protocol-related Error in RBUF or RBUFD**<br>Description see RBUF01SR.PERR0 or RBUF01SR.PERR1. |
| **RDV0** | 13 | rh | **Receive Data Valid in RBUF or RBUFD**<br>Description see RBUF01SR.RDV00 or RBUF01SR.RDV10. |
| **RDV1** | 14 | rh | **Receive Data Valid in RBUF or RBUFD**<br>Description see RBUF01SR.RDV01 or RBUF01SR.RDV11. |


Subject to Agreement on the Use of Product Information

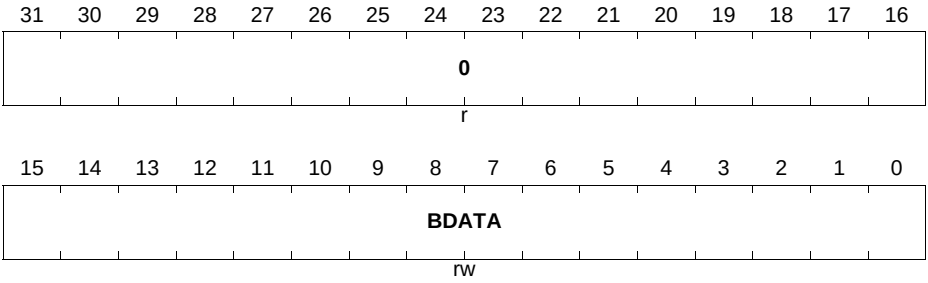
| Field | Bits | Type | Description |
|---|---|---|---|
| **DS** | 15 | rh | **Data Source of RBUF or RBUFD**<br>Description see RBUF01SR.DS0 or RBUF01SR.DS1. |
| **0** | [5:4], 7, [12:10], [31:16] | r | **Reserved**<br>Read as 0; should be written with 0. |

### 15.11.9 FIFO Buffer and Bypass Registers

#### 15.11.9.1 Bypass Registers

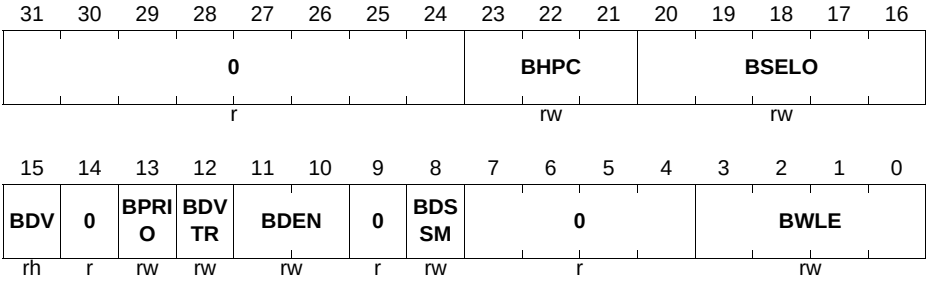
A write action to at least the low byte of the bypass data register sets BYPCR.BDV = 1 (bypass data tagged valid).

**BYP**
**Bypass Data Register** **($100_H$)** **Reset Value: 0000 0000$_H$**




| Bit (Field) | Width | Type | Description |
|---|---|---|---|
| **BDATA** | [15:0] | rw | **Bypass Data**<br>This bit field contains the bypass data. |
| **0** | [31:16] | r | **Reserved**<br>Read as 0; should be written with 0. |

**BYPCR**
**Bypass Control Register** **($104_H$)** **Reset Value: 0000 0000$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **BWLE** | [3:0] | rw | **Bypass Word Length**<br>This bit field defines the word length of the bypass data. The word length is given by BWLE + 1 with the data word being right-aligned in the data buffer at the bit positions [BWLE down to 0].<br>The bypass data word is always considered as an own frame with the length of BWLE.<br>Same coding as SCTR.WLE. |
| **BDSSM** | 8 | rw | **Bypass Data Single Shot Mode**<br>This bit defines if the bypass data is considered as permanently valid or if the bypass data is only transferred once (single shot mode).<br>$0_B$ The bypass data is still considered as valid after it has been loaded into TBUF. The loading of the data into TBUF does not clear BDV.<br>$1_B$ The bypass data is considered as invalid after it has been loaded into TBUF. The loading of the data into TBUF clears BDV. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **BDEN** | [11:10] | rw | **Bypass Data Enable**<br>This bit field defines if and how the transfer of bypass data to TBUF is enabled.<br>$00_B$ The transfer of bypass data is disabled.<br>$01_B$ The transfer of bypass data to TBUF is possible. Bypass data will be transferred to TBUF according to its priority if BDV = 1.<br>$10_B$ Gated bypass data transfer is enabled. Bypass data will be transferred to TBUF according to its priority if BDV = 1 and while DX2S = 0.<br>$11_B$ Gated bypass data transfer is enabled. Bypass data will be transferred to TBUF according to its priority if BDV = 1 and while DX2S = 1. |
| **BDVTR** | 12 | rw | **Bypass Data Valid Trigger**<br>This bit enables the bypass data for being tagged valid when DX2T is active (for time framing or time-out purposes).<br>$0_B$ Bit BDV is not influenced by DX2T.<br>$1_B$ Bit BDV is set if DX2T is active. |
| **BPRIO** | 13 | rw | **Bypass Priority**<br>This bit defines the priority between the bypass data and the transmit FIFO data.<br>$0_B$ The transmit FIFO data has a higher priority than the bypass data.<br>$1_B$ The bypass data has a higher priority than the transmit FIFO data. |
| **BDV** | 15 | rh | **Bypass Data Valid**<br>This bit defines if the bypass data is valid for a transfer to TBUF. This bit is set automatically by a write access to at least the low-byte of register BYP. It can be cleared by software by writing TRBSCR.CBDV.<br>$0_B$ The bypass data is not valid.<br>$1_B$ The bypass data is valid. |
| **BSELO** | [20:16] | rw | **Bypass Select Outputs**<br>This bit field contains the value that is written to PCR.CTR[20:16] if bypass data is transferred to TBUF while TCSR.SELMD = 1.<br>In the SSC protocol, this bit field can be used to define which SELOx output line will be activated when bypass data is transmitted. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **BHPC** | [23:21] | rw | **Bypass Hardware Port Control**<br>This bit field contains the value that is written to SCTR[4:2] if bypass data is transferred to TBUF while TCSR.HPCMD = 1.<br>In the SSC protocol, this bit field can be used to define the data shift mode and if hardware port control is enabled through CCR.HPCEN = 1, the pin direction when bypass data is transmitted. |
| **0** | [7:4], 9, 14, [31:24] | r | **Reserved**<br>Read as 0; should be written with 0. |

### 15.11.9.2 General FIFO Buffer Control Registers

The transmit and receive FIFO status information of USICx_CHy is given in registers USICx_CHy.TRBSR.

The bits related to the transmitter buffer in this register can only by written if the transmit buffer functionality is enabled by CCFG.TB = 1, otherwise write accesses are ignored. A similar behavior applies for the bits related to the receive buffer referring to CCFG.RB = 1.

The interrupt flags (event flags) in the transmit and receive FIFO status register TRBSR can be cleared by writing a 1 to the corresponding bit position in register TRBSCR, whereas writing a 0 has to effect on these bits. Writing a 1 by software to SRBI, RBERI, ARBI, STBI, or TBERI sets the corresponding bit to simulate the detection of a transmit/receive buffer event, but without activating any service request output (therefore, see FMR.SIOx).

Bits TBUS and RBUS have been implemented for testing purposes. They can be ignored by data handling software. Please note that a read action can deliver either a 0 or a 1 for these bits. It is recommended to treat them as “don’t care”.


Subject to Agreement on the Use of Product Information

## TRBSR
## Transmit/Receive Buffer Status Register

**(114$_H$)** **Reset Value: 0000 0808$_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **TBFLVL** | | | | | | | **0** | **RBFLVL** | | | | | | |
| r | rh | | | | | | | r | rh | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **STBT** | **TBUS** | **TFULL** | **TEMPTY** | **0** | **TBERI** | **STBI** | **0** | **SRBT** | **RBUS** | **RFULL** | **REMPTY** | **ARBI** | **RBERI** | **SRBI** |
| r | rh | rh | rh | rh | r | rwh | rwh | r | rh | rh | rh | rh | rwh | rwh | rwh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **SRBI** | 0 | rwh | **Standard Receive Buffer Event**<br>This bit indicates that a standard receive buffer event has been detected. It is cleared by writing TRBSCR.CSRBI = 1.<br>If enabled by RBCTR.SRBIEN, the service request output SRx selected by RBCTR.SRBINP becomes activated if a standard receive buffer event is detected.<br>$0_B$ A standard receive buffer event has not been detected.<br>$1_B$ A standard receive buffer event has been detected. |
| **RBERI** | 1 | rwh | **Receive Buffer Error Event**<br>This bit indicates that a receive buffer error event has been detected. It is cleared by writing TRBSCR.CRBERI = 1.<br>If enabled by RBCTR.RBERIEN, the service request output SRx selected by RBCTR.ARBINP becomes activated if a receive buffer error event is detected.<br>$0_B$ A receive buffer error event has not been detected.<br>$1_B$ A receive buffer error event has been detected. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **ARBI** | 2 | rwh | **Alternative Receive Buffer Event**<br>This bit indicates that an alternative receive buffer event has been detected. It is cleared by writing TRBSCR.CARBI = 1.<br>If enabled by RBCTR.ARBIEN, the service request output SRx selected by RBCTR.ARBINP becomes activated if an alternative receive buffer event is detected.<br>$0_B$ An alternative receive buffer event has not been detected.<br>$1_B$ An alternative receive buffer event has been detected. |
| **REMPTY** | 3 | rh | **Receive Buffer Empty**<br>This bit indicates whether the receive buffer is empty.<br>$0_B$ The receive buffer is not empty.<br>$1_B$ The receive buffer is empty. |
| **RFULL** | 4 | rh | **Receive Buffer Full**<br>This bit indicates whether the receive buffer is full.<br>$0_B$ The receive buffer is not full.<br>$1_B$ The receive buffer is full. |
| **RBUS** | 5 | rh | **Receive Buffer Busy**<br>This bit indicates whether the receive buffer is currently updated by the FIFO handler.<br>$0_B$ The receive buffer information has been completely updated.<br>$1_B$ The OUTR update from the FIFO memory is ongoing. A read from OUTR will be delayed. FIFO pointers from the previous read are not yet updated. |
| **SRBT** | 6 | rh | **Standard Receive Buffer Event Trigger**<br>This bit triggers a standard receive buffer event when set.<br>If enabled by RBCTR.SRBIEN, the service request output SRx selected by RBCTR.SRBINP becomes activated until the bit is cleared.<br>$0_B$ A standard receive buffer event is not triggered using this bit.<br>$1_B$ A standard receive buffer event is triggered using this bit. |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **STBI** | 8 | rwh | **Standard Transmit Buffer Event**<br>This bit indicates that a standard transmit buffer event has been detected. It is cleared by writing TRBSCR.CSTBI = 1.<br>If enabled by TBCTR.STBIEN, the service request output SRx selected by TBCTR.STBINP becomes activated if a standard transmit buffer event is detected.<br>$0_B$ A standard transmit buffer event has not been detected.<br>$1_B$ A standard transmit buffer event has been detected. |
| **TBERI** | 9 | rwh | **Transmit Buffer Error Event**<br>This bit indicates that a transmit buffer error event has been detected. It is cleared by writing TRBSCR.CTBERI = 1.<br>If enabled by TBCTR.TBERIEN, the service request output SRx selected by TBCTR.ATBINP becomes activated if a transmit buffer error event is detected.<br>$0_B$ A transmit buffer error event has not been detected.<br>$1_B$ A transmit buffer error event has been detected. |
| **TEMPTY** | 11 | rh | **Transmit Buffer Empty**<br>This bit indicates whether the transmit buffer is empty.<br>$0_B$ The transmit buffer is not empty.<br>$1_B$ The transmit buffer is empty. |
| **TFULL** | 12 | rh | **Transmit Buffer Full**<br>This bit indicates whether the transmit buffer is full.<br>$0_B$ The transmit buffer is not full.<br>$1_B$ The transmit buffer is full. |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **TBUS** | 13 | rh | **Transmit Buffer Busy**<br>This bit indicates whether the transmit buffer is currently updated by the FIFO handler.<br>$0_B$ The transmit buffer information has been completely updated.<br>$1_B$ The FIFO memory update after write to INx is ongoing. A write to INx will be delayed. FIFO pointers from the previous INx write are not yet updated. |
| **STBT** | 14 | rh | **Standard Transmit Buffer Event Trigger**<br>This bit triggers a standard transmit buffer event when set.<br>If enabled by TBCTR.STBIEN, the service request output SRx selected by TBCTR.STBINP becomes activated until the bit is cleared.<br>$0_B$ A standard transmit buffer event is not triggered using this bit.<br>$1_B$ A standard transmit buffer event is triggered using this bit. |
| **RBFLVL** | [22:16] | rh | **Receive Buffer Filling Level**<br>This bit field indicates the filling level of the receive buffer, starting with 0 for an empty buffer. |
| **TBFLVL** | [30:24] | rh | **Transmit Buffer Filling Level**<br>This bit field indicates the filling level of the transmit buffer, starting with 0 for an empty buffer.<br>*Note: The first data word written to Transmit FIFO will be loaded immediately to TBUF and removed from FIFO.* |
| **0** | 7, 10, 15, 23, 31 | r | **Reserved**<br>Read as 0; should be written with 0. |


Subject to Agreement on the Use of Product Information

The bits in register TRBSCR are used to clear the notification bits in register TRBSR or to clear the FIFO mechanism for the transmit or receive buffer. A read action always delivers 0.

**TRBSCR**
**Transmit/Receive Buffer Status Clear Register**
**($118_H$)** **Reset Value: $0000\ 0000_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FLUSHTB | FLUSHRB | 0 | | | CBDV | CTBERI | CSTBI | 0 | | | | | CARBI | CRBERI | CSRBI |
| w | w | r | | | w | w | w | r | | | | | w | w | w |

| Field | Bits | Type | Description |
|---|---|---|---|
| **CSRBI** | 0 | w | **Clear Standard Receive Buffer Event**<br>$0_B$ No effect.<br>$1_B$ Clear TRBSR.SRBI. |
| **CRBERI** | 1 | w | **Clear Receive Buffer Error Event**<br>$0_B$ No effect.<br>$1_B$ Clear TRBSR.RBERI. |
| **CARBI** | 2 | w | **Clear Alternative Receive Buffer Event**<br>$0_B$ No effect.<br>$1_B$ Clear TRBSR.ARBI. |
| **CSTBI** | 8 | w | **Clear Standard Transmit Buffer Event**<br>$0_B$ No effect.<br>$1_B$ Clear TRBSR.STBI. |
| **CTBERI** | 9 | w | **Clear Transmit Buffer Error Event**<br>$0_B$ No effect.<br>$1_B$ Clear TRBSR.TBERI. |
| **CBDV** | 10 | w | **Clear Bypass Data Valid**<br>$0_B$ No effect.<br>$1_B$ Clear BYPCR.BDV. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **FLUSHRB** | 14 | w | **Flush Receive Buffer**<br>$0_B$ No effect.<br>$1_B$ The receive FIFO buffer is cleared (filling level is cleared and output pointer is set to input pointer value). Should only be used while the FIFO buffer is not taking part in data traffic. |
| **FLUSHTB** | 15 | w | **Flush Transmit Buffer**<br>$0_B$ No effect.<br>$1_B$ The transmit FIFO buffer is cleared (filling level is cleared and output pointer is set to input pointer value). Should only be used while the FIFO buffer is not taking part in data traffic. |
| **0** | [7:3], [13:11], [31:16] | r | **Reserved**<br>Read as 0; should be written with 0. |

### 15.11.9.3 Transmit FIFO Buffer Control Registers

The transmit FIFO buffer is controlled by register TBCTR. TBCTR can only by written if the transmit buffer functionality is enabled by CCFG.TB = 1, otherwise write accesses are ignored.

**TBCTR**
**Transmitter Buffer Control Register ($108_H$)** **Reset Value: $0000\ 0000_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **TBERIEN** | **STBIEN** | **0** | **LOF** | **0** | **SIZE** | | | **0** | | **ATBINP** | | | **STBINP** | | |
| rw | rw | r | rw | r | rw | | | r | | rw | | | rw | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **STBTEN** | **STBTM** | **LIMIT** | | | | | | **0** | | **DPTR** | | | | | |
| rw | rw | rw | | | | | | r | | w | | | | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **DPTR** | [5:0] | w | **Data Pointer**<br>This bit field defines the start value for the transmit buffer pointers when assigning the FIFO entries to the transmit FIFO buffer. A read always delivers 0. When writing DPTR while SIZE = 0, both transmitter pointers TDIPTR and RTDOPTR in register TRBPTR are updated with the written value and the buffer is considered as empty. A write access to DPTR while SIZE > 0 is ignored and does not modify the pointers. |
| **LIMIT** | [13:8] | rw | **Limit For Interrupt Generation**<br>This bit field defines the target filling level of the transmit FIFO buffer that is used for the standard transmit buffer event detection. |
| **STBTM** | 14 | rw | **Standard Transmit Buffer Trigger Mode**<br>This bit selects the standard transmit buffer event trigger mode.<br>$0_B$ Trigger mode 0:<br>While TRBSR.STBT=1, a standard buffer event will be generated whenever there is a data transfer to TBUF or data write to INx (depending on TBCTR.LOF setting). STBT is cleared when TRBSR.TBFLVL=TBCTR.LIMIT.<br>$1_B$ Trigger mode 1:<br>While TRBSR.STBT=1, a standard buffer event will be generated whenever there is a data transfer to TBUF or data write to INx (depending on TBCTR.LOF setting). STBT is cleared when TRBSR.TBFLVL=TBCTR.SIZE. |
| **STBTEN** | 15 | rw | **Standard Transmit Buffer Trigger Enable**<br>This bit enables/disables triggering of the standard transmit buffer event through bit TRBSR.STBT.<br>$0_B$ The standard transmit buffer event trigger through bit TRBSR.STBT is disabled.<br>$1_B$ The standard transmit buffer event trigger through bit TRBSR.STBT is enabled. |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **STBINP** | [18:16] | rw | **Standard Transmit Buffer Interrupt Node Pointer**<br>This bit field defines which service request output SRx becomes activated in case of a standard transmit buffer event.<br>$000_B$ Output SR0 becomes activated.<br>$001_B$ Output SR1 becomes activated.<br>$010_B$ Output SR2 becomes activated.<br>$011_B$ Output SR3 becomes activated.<br>$100_B$ Output SR4 becomes activated.<br>$101_B$ Output SR5 becomes activated.<br>*Note: All other settings of the bit field are reserved.* |
| **ATBINP** | [21:19] | rw | **Alternative Transmit Buffer Interrupt Node Pointer**<br>This bit field define which service request output SRx will be activated in case of a transmit buffer error event.<br>$000_B$ Output SR0 becomes activated.<br>$001_B$ Output SR1 becomes activated.<br>$010_B$ Output SR2 becomes activated.<br>$011_B$ Output SR3 becomes activated.<br>$100_B$ Output SR4 becomes activated.<br>$101_B$ Output SR5 becomes activated.<br>*Note: All other settings of the bit field are reserved.* |
| **SIZE** | [26:24] | rw | **Buffer Size**<br>This bit field defines the number of FIFO entries assigned to the transmit FIFO buffer.<br>$000_B$ The FIFO mechanism is disabled. The buffer does not accept any request for data.<br>$001_B$ The FIFO buffer contains 2 entries.<br>$010_B$ The FIFO buffer contains 4 entries.<br>$011_B$ The FIFO buffer contains 8 entries.<br>$100_B$ The FIFO buffer contains 16 entries.<br>$101_B$ The FIFO buffer contains 32 entries.<br>$110_B$ The FIFO buffer contains 64 entries.<br>$111_B$ Reserved |

| Field | Bits | Type | Description |
|---|---|---|---|
| **LOF** | 28 | rw | **Buffer Event on Limit Overflow**<br>This bit defines which relation between filling level and programmed limit leads to a standard transmit buffer event.<br>$0_B$ A standard transmit buffer event occurs when the filling level equals the limit value and gets lower due to transmission of a data word.<br>$1_B$ A standard transmit buffer interrupt event occurs when the filling level equals the limit value and gets bigger due to a write access to a data input location INx. |
| **STBIEN** | 30 | rw | **Standard Transmit Buffer Interrupt Enable**<br>This bit enables/disables the generation of a standard transmit buffer interrupt in case of a standard transmit buffer event.<br>$0_B$ The standard transmit buffer interrupt generation is disabled.<br>$1_B$ The standard transmit buffer interrupt generation is enabled. |
| **TBERIEN** | 31 | rw | **Transmit Buffer Error Interrupt Enable**<br>This bit enables/disables the generation of a transmit buffer error interrupt in case of a transmit buffer error event (software writes to a full transmit buffer).<br>$0_B$ The transmit buffer error interrupt generation is disabled.<br>$1_B$ The transmit buffer error interrupt generation is enabled. |
| **0** | [7:6], [23:22], 27, 29 | r | **Reserved**<br>Read as 0; should be written with 0. |


Subject to Agreement on the Use of Product Information

### 15.11.9.4 Receive FIFO Buffer Control Registers

The receive FIFO buffer is controlled by register RBCTR. This register can only be written if the receive buffer functionality is enabled by CCFG.RB = 1, otherwise write accesses are ignored.

**RBCTR**
**Receiver Buffer Control Register ($10C_H$)** **Reset Value: $0000\ 0000_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RBERIEN | SRBIEN | ARBIEN | LOF | RNM | SIZE | | | RCIM | | ARBINP | | | SRBINP | | |
| rw | rw | rw | rw | rw | rw | | | rw | | rw | | | rw | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SRBTEN | SRBTM | LIMIT | | | | | | 0 | | DPTR | | | | | |
| rw | rw | rw | | | | | | r | | w | | | | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **DPTR** | [5:0] | w | **Data Pointer**<br>This bit field defines the start value for the receive buffer pointers when assigning the FIFO entries to the receive FIFO buffer. A read always delivers 0. When writing DPTR while SIZE = 0, both receiver pointers RDIPTR and RDOPTR in register TRBPTR are updated with the written value and the buffer is considered as empty. A write access to DPTR while SIZE > 0 is ignored and does not modify the pointers. |
| **LIMIT** | [13:8] | rw | **Limit For Interrupt Generation**<br>This bit field defines the target filling level of the receive FIFO buffer that is used for the standard receive buffer event detection. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **SRBTM** | 14 | rw | **Standard Receive Buffer Trigger Mode**<br>This bit selects the standard receive buffer event trigger mode.<br>$0_B$ Trigger mode 0:<br>While TRBSR.SRBT=1, a standard receive buffer event will be generated whenever there is a new data received or data read out (depending on RBCTR.LOF setting). SRBT is cleared when TRBSR.RBFLVL=RBCTR.LIMIT.<br>$1_B$ Trigger mode 1:<br>While TRBSR.SRBT=1, a standard receive buffer event will be generated whenever there is a new data received or data read out (depending on RBCTR.LOF setting). SRBT is cleared when TRBSR.RBFLVL=0. |
| **SRBTEN** | 15 | rw | **Standard Receive Buffer Trigger Enable**<br>This bit enables/disables triggering of the standard receive buffer event through bit TRBSR.SRBT.<br>$0_B$ The standard receive buffer event trigger through bit TRBSR.SRBT is disabled.<br>$1_B$ The standard receive buffer event trigger through bit TRBSR.SRBT is enabled. |
| **SRBINP** | [18:16] | rw | **Standard Receive Buffer Interrupt Node Pointer**<br>This bit field defines which service request output SRx becomes activated in case of a standard receive buffer event.<br>$000_B$ Output SR0 becomes activated.<br>$001_B$ Output SR1 becomes activated.<br>$010_B$ Output SR2 becomes activated.<br>$011_B$ Output SR3 becomes activated.<br>$100_B$ Output SR4 becomes activated.<br>$101_B$ Output SR5 becomes activated.<br>*Note: All other settings of the bit field are reserved.* |

| Field | Bits | Type | Description |
|---|---|---|---|
| **ARBINP** | [21:19] | rw | **Alternative Receive Buffer Interrupt Node Pointer**<br>This bit field defines which service request output SRx becomes activated in case of an alternative receive buffer event or a receive buffer error event.<br>$000_B$ Output SR0 becomes activated.<br>$001_B$ Output SR1 becomes activated.<br>$010_B$ Output SR2 becomes activated.<br>$011_B$ Output SR3 becomes activated.<br>$100_B$ Output SR4 becomes activated.<br>$101_B$ Output SR5 becomes activated.<br>*Note: All other settings of the bit field are reserved.* |
| **RCIM** | [23:22] | rw | **Receiver Control Information Mode**<br>This bit field defines which information from the receiver status register RBUFSR is propagated as 5 bit receiver control information RCI[4:0] to the receive FIFO buffer and can be read out in registers OUT(D)R.<br>$00_B$ RCI[4] = PERR, RCI[3:0] = WLEN<br>$01_B$ RCI[4] = SOF, RCI[3:0] = WLEN<br>$10_B$ RCI[4] = 0, RCI[3:0] = WLEN<br>$11_B$ RCI[4] = PERR, RCI[3] = PAR, RCI[2:1] = $00_B$, RCI[0] = SOF |
| **SIZE** | [26:24] | rw | **Buffer Size**<br>This bit field defines the number of FIFO entries assigned to the receive FIFO buffer.<br>$000_B$ The FIFO mechanism is disabled. The buffer does not accept any request for data.<br>$001_B$ The FIFO buffer contains 2 entries.<br>$010_B$ The FIFO buffer contains 4 entries.<br>$011_B$ The FIFO buffer contains 8 entries.<br>$100_B$ The FIFO buffer contains 16 entries.<br>$101_B$ The FIFO buffer contains 32 entries.<br>$110_B$ The FIFO buffer contains 64 entries.<br>$111_B$ Reserved |

| Field | Bits | Type | Description |
|---|---|---|---|
| **RNM** | 27 | rw | **Receiver Notification Mode**<br>This bit defines the receive buffer event mode. The receive buffer error event is not affected by RNM.<br>$0_B$ Filling level mode:<br>A standard receive buffer event occurs when the filling level equals the limit value and changes, either due to a read access from OUTR (LOF = 0) or due to a new received data word (LOF = 1).<br>$1_B$ RCI mode:<br>A standard receive buffer event occurs when register OUTR is updated with a new value if the corresponding value in OUTR.RCI[4] = 0. If OUTR.RCI[4] = 1, an alternative receive buffer event occurs instead of the standard receive buffer event. |
| **LOF** | 28 | rw | **Buffer Event on Limit Overflow**<br>This bit defines which relation between filling level and programmed limit leads to a standard receive buffer event in filling level mode (RNM = 0). In RCI mode (RNM = 1), bit fields LIMIT and LOF are ignored.<br>$0_B$ A standard receive buffer event occurs when the filling level equals the limit value and gets lower due to a read access from OUTR.<br>$1_B$ A standard receive buffer event occurs when the filling level equals the limit value and gets bigger due to the reception of a new data word. |
| **ARBIEN** | 29 | rw | **Alternative Receive Buffer Interrupt Enable**<br>This bit enables/disables the generation of an alternative receive buffer interrupt in case of an alternative receive buffer event.<br>$0_B$ The alternative receive buffer interrupt generation is disabled.<br>$1_B$ The alternative receive buffer interrupt generation is enabled. |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **SRBIEN** | 30 | rw | **Standard Receive Buffer Interrupt Enable**<br>This bit enables/disables the generation of a standard receive buffer interrupt in case of a standard receive buffer event.<br>$0_B$ The standard receive buffer interrupt generation is disabled.<br>$1_B$ The standard receive buffer interrupt generation is enabled. |
| **RBERIEN** | 31 | rw | **Receive Buffer Error Interrupt Enable**<br>This bit enables/disables the generation of a receive buffer error interrupt in case of a receive buffer error event (the software reads from an empty receive buffer).<br>$0_B$ The receive buffer error interrupt generation is disabled.<br>$1_B$ The receive buffer error interrupt generation is enabled. |
| **0** | [7:6] | r | **Reserved**<br>Read as 0; should be written with 0. |


Subject to Agreement on the Use of Product Information

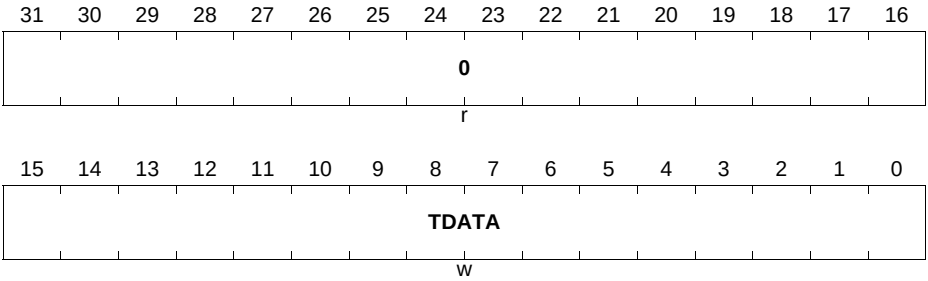
### 15.11.9.5 FIFO Buffer Data Registers

The 32 independent data input locations IN00 to IN31 are addresses that can be used as data entry locations for the transmit FIFO buffer. Data written to one of these locations will be stored in the transmit buffer FIFO. Additionally, the 5-bit coding of the number [31:0] of the addressed data input location represents the transmit control information TCI.

If the FIFO is already full and new data is written to it, the write access is ignored and a transmit buffer error event is signaled.

**INx (x = 00-31)**
**Transmit FIFO Buffer Input Location x**

**($180_H$ + x *4)** **Reset Value: 0000 $0000_H$**

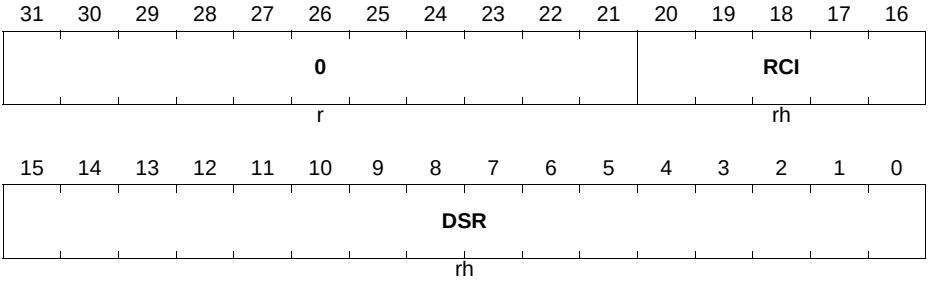



| Field | Bits | Type | Description |
|---|---|---|---|
| **TDATA** | [15:0] | w | **Transmit Data**<br>This bit field contains the data to be transmitted (write view), read actions deliver 0.<br>A write action to at least the low byte of TDATA triggers the data storage in the FIFO. |
| **0** | [31:16] | r | **Reserved**<br>Read as 0; should be written with 0. |


Subject to Agreement on the Use of Product Information

The receiver FIFO buffer output register OUTR shows the oldest received data word in the FIFO buffer and contains the receiver control information RCI containing the information selected by RBCTR.RCIM. A read action from this address location delivers the received data. With a read access of at least the low byte, the data is declared to be read and the next entry becomes visible. Write accesses to OUTR are ignored.

**OUTR**
**Receiver Buffer Output Register (11C$_H$)** **Reset Value: 0000 0000$_H$**

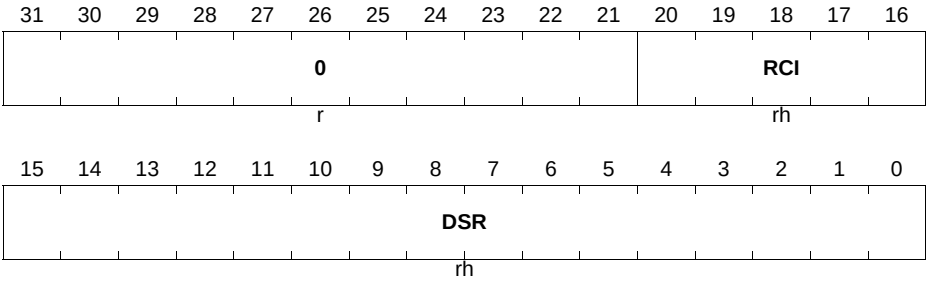



| Field | Bits | Type | Description |
|---|---|---|---|
| **DSR** | [15:0] | rh | **Received Data**<br>This bit field monitors the content of the oldest data word in the receive FIFO.<br>Reading at least the low byte releases the buffer entry currently shown in DSR. |
| **RCI** | [20:16] | rh | **Receiver Control Information**<br>This bit field monitors the receiver control information associated to DSR. The bit structure of RCI depends on bit field RBCTR.RCIM. |
| **0** | [31:21] | r | **Reserved**<br>Read as 0; should be written with 0. |


Subject to Agreement on the Use of Product Information

If a debugger should be used to monitor the received data in the FIFO buffer, the FIFO mechanism must not be activated in order to guaranty data consistency. Therefore, a second address set is available, named OUTDR (D like debugger), having the same bit fields like the original buffer output register OUTR, but without the FIFO mechanism. A debugger can read here (in order to monitor the receive data flow) without the risk of data corruption. Write accesses to OUTDR are ignored.

**OUTDR**
**Receiver Buffer Output Register L for Debugger**
**($120_H$)** **Reset Value: $0000\ 0000_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **DSR** | [15:0] | rh | **Data from Shift Register**<br>Same as OUTR.DSR, but without releasing the buffer after a read action. |
| **RCI** | [20:16] | rh | **Receive Control Information from Shift Register**<br>Same as OUTR.RCI. |
| **0** | [31:21] | r | **Reserved**<br>Read as 0; should be written with 0. |

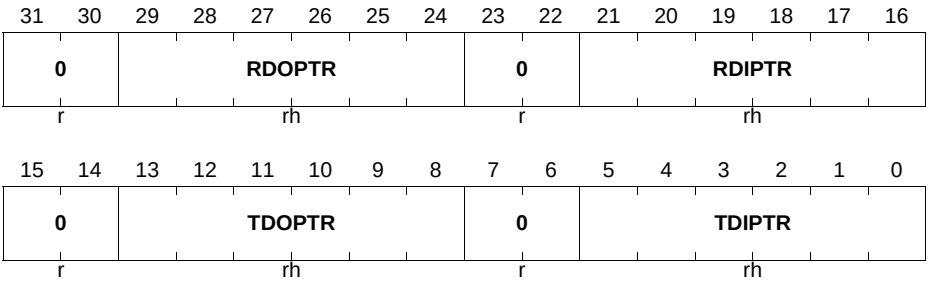

Subject to Agreement on the Use of Product Information

### 15.11.9.6 FIFO Buffer Pointer Registers

The pointers for FIFO handling of the transmit and receive FIFO buffers are located in register TRBPTR. The pointers are automatically handled by the FIFO buffer mechanism and do not need to be modified by software. As a consequence, these registers can only be read by software (e.g. for verification purposes), whereas write accesses are ignored.

**TRBPTR**
**Transmit/Receive Buffer Pointer Register**

**($110_H$)** **Reset Value: $0000\ 0000_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **TDIPTR** | [5:0] | rh | **Transmitter Data Input Pointer**<br>This bit field indicates the buffer entry that will be used for the next transmit data coming from the INx addresses. |
| **TDOPTR** | [13:8] | rh | **Transmitter Data Output Pointer**<br>This bit field indicates the buffer entry that will be used for the next transmit data to be output to TBUF. |
| **RDIPTR** | [21:16] | rh | **Receiver Data Input Pointer**<br>This bit field indicates the buffer entry that will be used for the next receive data coming from RBUF. |
| **RDOPTR** | [29:24] | rh | **Receiver Data Output Pointer**<br>This bit field indicates the buffer entry that will be used for the next receive data to be output at the OUT(D)R addresses. |
| **0** | [7:6], [15:14], [23:22], [31:30] | r | **Reserved**<br>Read as 0; should be written with 0. |

## 15.12 Interconnects

The XMC1300 device contains one USIC module (USIC0) with 2 communication channels.

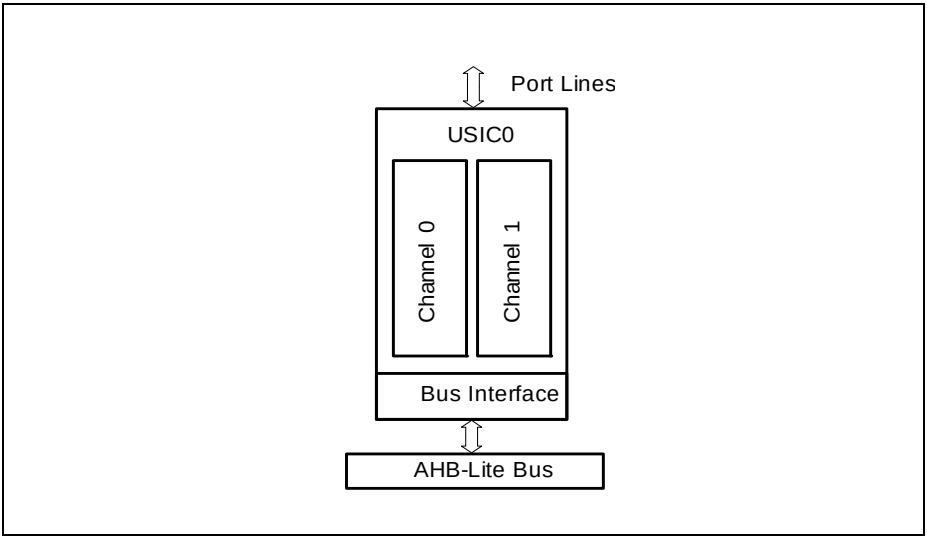



**Figure 15-66 USIC Module Structure in XMC1300**

**Figure 15-67** shows the I/O lines of one USIC channel. The tables in this section define the pin assignments and internal connections of the USIC channels I/O lines in the XMC1300 device. Naming convention: USICx_CHy refers to USIC module x channel y.

The meaning of these pins with respect to each protocol is described in the protocols' respective chapters and summarized in **Table 15-2** and **Table 15-3**.


Subject to Agreement on the Use of Product Information

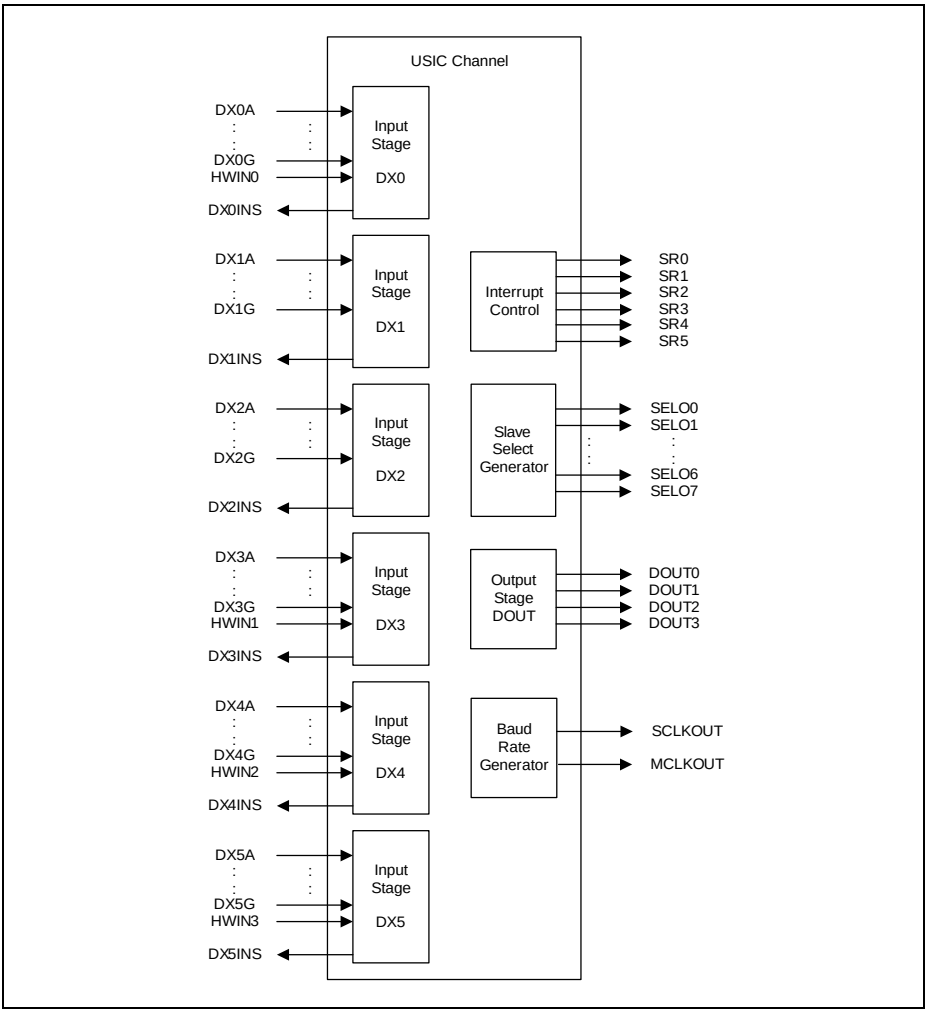



**Figure 15-67 USIC Channel I/O Lines**

The service request outputs SR[5:0] of one USIC channel is combined with those of the other channel with the module. Therefore, only 6 service request outputs are available per module.

### 15.12.1 USIC Module 0 Interconnects

The interconnects of USIC module 0 is grouped into the following categories:

- **USIC Module 0 Channel 0 Interconnects** (**Table 15-23**)
- **USIC Module 0 Channel 1 Interconnects** (**Table 15-24**)
- **USIC Module 0 Module Interconnects** (**Table 15-25**)

**Table 15-23 USIC Module 0 Channel 0 Interconnects**

| Input/Output | I/O | Connected To | Description |
|---|---|---|---|
| **Data Inputs (DX0)** | | | |
| USIC0_CH0.DX0A | I | P0.14 | Shift data input; used for:<br>• ASC RXD<br>• SSC MTSR/MRST<br>• IIC SDA<br>• 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 | |
| USIC0_CH0.HWIN0 | I | P1.0 | HW controlled shift data input |
| **Clock Inputs** | | | |
| USIC0_CH0.DX1A | I | P0.14 | Shift clock input; used for:<br>• SSC Slave SCLKIN<br>• IIC SCL<br>• IIS Slave SCLKIN<br>• 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 | P0.0 | Shift control input; used for:<br>• SSC Slave SELIN<br>• IIS Slave WAIN<br>• 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 | |


Subject to Agreement on the Use of Product Information

**Table 15-23 USIC Module 0 Channel 0 Interconnects** (cont'd)

| Input/Output | I/O | Connected To | Description |
|---|---|---|---|
| **Data Inputs (DX3)** | | | |
| USIC0_CH0.DX3A | I | P2.2 | Shift data input; used for:<br>• Dual and Quad SSC MTSR1/MRST1<br>• Can be ignored for all other protocols<br>It can also be used as a DX0 function if DX0G input is selected. |
| 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:<br>• Quad SSC MTSR2/MRST2<br>• Can be ignored for all other protocols<br>It can also be used as a DX1 function if DX1G input is selected. |
| 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 | |
| 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:<br>• Quad SSC MTSR3/MRST3<br>• Can be ignored for all other protocols<br>It can also be used as a DX2/DX3/DX4 function if DX2G/DX3F/DX4F input is selected. |
| 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 |


Subject to Agreement on the Use of Product Information

**Table 15-23 USIC Module 0 Channel 0 Interconnects** (cont'd)

| Input/Output | I/O | Connected To | Description |
|---|---|---|---|
| **Data Outputs** | | | |
| USIC0_CH0.DOUT0 | O | P0.14<br>P0.15<br>P1.0<br>P1.0 (HW1_OUT)<br>P1.1<br>P1.5<br>P2.0<br>P2.1<br>USIC0_CH0.DX3G | Shift data output; used for:<br>• ASC TXD<br>• SSC MTSR/MRST<br>• IIC SDA<br>• IIS DOUT |
| USIC0_CH0.DOUT1 | O | P1.1 (HW1_OUT) | Shift data output; used for:<br>• Dual and Quad SSC MTSR1/MRST1<br>• Can be ignored for all other protocols |
| USIC0_CH0.DOUT2 | O | P1.2 (HW1_OUT) | Shift data output; used for:<br>• Quad SSC MTSR2/MRST2<br>• Can be ignored for all other protocols |
| USIC0_CH0.DOUT3 | O | P1.3 (HW1_OUT) | Shift data output; used for:<br>• Quad SSC MTSR3/MRST3<br>• Can be ignored for all other protocols |
| **Clock Outputs** | | | |
| USIC0_CH0.MCLKOUT | O | P0.11 | Master clock output<br>(optional for all protocols) |
| USIC0_CH0.SCLKOUT | O | P0.7<br>P0.8<br>P0.14<br>P1.6<br>P2.0<br>USIC0_CH0.DX4G | Shift clock output; used for:<br>• Master SCLKOUT in SSC and IIS<br>• IIC SCL<br>• Can be ignored in ASC |

**Table 15-23 USIC Module 0 Channel 0 Interconnects** (cont'd)

| Input/Output | I/O | Connected To | Description |
|---|---|---|---|
| **Control Outputs** | | | |
| USIC0_CH0.SELO0 | O | P0.0<br>P0.9<br>P1.4<br>USIC0_CH0.DX5G | Shift control output; used for:<br>• SSC Master SELO<br>• IIS WA<br>• Can be ignored for all other protocols |
| USIC0_CH0.SELO1 | O | P0.10<br>P1.5 | |
| USIC0_CH0.SELO2 | O | P0.11<br>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.IN0L | 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<br>USIC0_CH0.DX3F<br>USIC0_CH0.DX4F | Selected DX5 input signal |


Subject to Agreement on the Use of Product Information

**Table 15-24 USIC Module 0 Channel 1 Interconnects**

| Input/Output | I/O | Connected To | Description |
|---|---|---|---|
| **Data Inputs (DX0)** | | | |
| USIC0_CH1.DX0A | I | P1.3 | Shift data input; used for:<br>• ASC RXD<br>• SSC MTSR/MRST<br>• IIC SDA<br>• 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:<br>• SSC Slave SCLKIN<br>• IIC SCL<br>• IIS Slave SCLKIN<br>• 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:<br>• SSC Slave SELIN<br>• IIS Slave WAIN<br>• 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 | |


Subject to Agreement on the Use of Product Information

**Table 15-24 USIC Module 0 Channel 1 Interconnects** (cont'd)

| Input/Output | I/O | Connected To | Description |
|---|---|---|---|
| **Data Inputs (DX3)** | | | |
| USIC0_CH1.DX3A | I | P2.1 | Shift data input; used for:<br>• Dual and Quad SSC MTSR1/MRST1<br>• Can be ignored for all other protocols<br>It can also be used as a DX0 function if DX0G input is selected. |
| USIC0_CH1.DX3B | I | P2.9 | |
| USIC0_CH1.DX3C | I | P2.3 | |
| USIC0_CH1.DX3D | I | P2.7 | |
| USIC0_CH1.DX3E | I | P2.5 | |
| 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:<br>• Quad SSC MTSR2/MRST2<br>• Can be ignored for all other protocols<br>It can also be used as a DX1 function if DX1G input is selected. |
| 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.SCLKOUT | |
| 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:<br>• Quad SSC MTSR3/MRST3<br>• Can be ignored for all other protocols<br>It can also be used as a DX2/DX3/DX4 function if DX2G/DX3F/DX4F input is selected. |
| 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 |


Subject to Agreement on the Use of Product Information

**Table 15-24 USIC Module 0 Channel 1 Interconnects** (cont'd)

| Input/Output | I/O | Connected To | Description |
|---|---|---|---|
| **Data Outputs** | | | |
| USIC0_CH1.DOUT0 | O | P0.6<br>P0.7<br>P1.2<br>P1.3<br>P1.6<br>P2.10<br>P2.11<br>USIC0_CH1.DX3G<br>USIC0_CH1.HWIN3 | Shift data output; used for:<br>• ASC TXD<br>• SSC MTSR/MRST<br>• IIC SDA<br>• IIS DOUT |
| USIC0_CH1.DOUT1 | O | not connected | Shift data output; used for:<br>• Dual and Quad SSC MTSR1/MRST1<br>• Can be ignored for all other protocols |
| USIC0_CH1.DOUT2 | O | not connected | Shift data output; used for:<br>• Quad SSC MTSR2/MRST2<br>• Can be ignored for all other protocols |
| USIC0_CH1.DOUT3 | O | not connected | Shift data output; used for:<br>• Quad SSC MTSR3/MRST3<br>• Can be ignored for all other protocols |
| **Clock Outputs** | | | |
| USIC0_CH1.MCLKOUT | O | P0.6<br>P0.15 | Master clock output<br>(optional for all protocols) |
| USIC0_CH1.SCLKOUT | O | P0.8<br>P1.3<br>P1.4<br>P2.1<br>P2.11<br>USIC0_CH1.DX4G | Shift clock output; used for:<br>• Master SCLKOUT in SSC and IIS<br>• IIC SCL<br>• Can be ignored in ASC |

**Table 15-24 USIC Module 0 Channel 1 Interconnects** (cont'd)

| Input/Output | I/O | Connected To | Description |
|---|---|---|---|
| **Control Outputs** | | | |
| USIC0_CH1.SELO0 | O | P0.0<br>P0.9<br>P1.1 | Shift control output; used for:<br>• SSC Master SELO<br>• IIS WA<br>• Can be ignored for all other protocols |
| USIC0_CH1.SELO1 | O | P0.10<br>P1.4 | |
| USIC0_CH1.SELO2 | O | P0.11<br>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.IN1L | 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 |
| USIC0_CH1.DX5INS | O | USIC0_CH1.DX2G<br>USIC0_CH1.DX3F<br>USIC0_CH1.DX4F | Selected DX5 input signal |

**Table 15-25 USIC Module 0 Module Interconnects**

| Input/Output | I/O | Connected To | Description |
|---|---|---|---|
| USIC0_SR0 | O | NVIC<br>USIC0_CH1.DX5G | interrupt output lines (service requests SRx) |
| USIC0_SR[5:1] | O | NVIC | interrupt output lines (service requests SRx) |


Subject to Agreement on the Use of Product Information

# Analog Peripherals


Subject to Agreement on the Use of Product Information

# 16 Versatile Analog-to-Digital Converter (VADC)

The XMC1300 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 XMC1300 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 16-153**).

**Table 16-1 Abbreviations used in ADC chapter**

| Abbreviation | 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 |

## 16.1 Overview

Each converter of the ADC cluster can operate independent of the others, controlled by a dedicated set of registers and triggered by a dedicated group request source. The results of each channel can be stored in a dedicated channel-specific result register or in a group-specific result register.

A background request source can access all analog input channels that are not assigned to any group request source. These conversions are executed with low priority. The background request source can, therefore, be regarded as an additional background converter.

The Versatile Analog to Digital Converter module (VADC) of the XMC1300 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).*


Subject to Agreement on the Use of Product Information

**Table 16-2 VADC Applications**

| Use Case VADC | Application |
|---|---|
| Automatic scheduling of complex conversion sequences, including priorization of time-critical conversions | Motor control, Power conversion |
| Effective result handling for bursts of high-speed conversions | Highly dynamic input signals |
| Synchronous sampling of 2 input signals | Multi-phase current measurement |

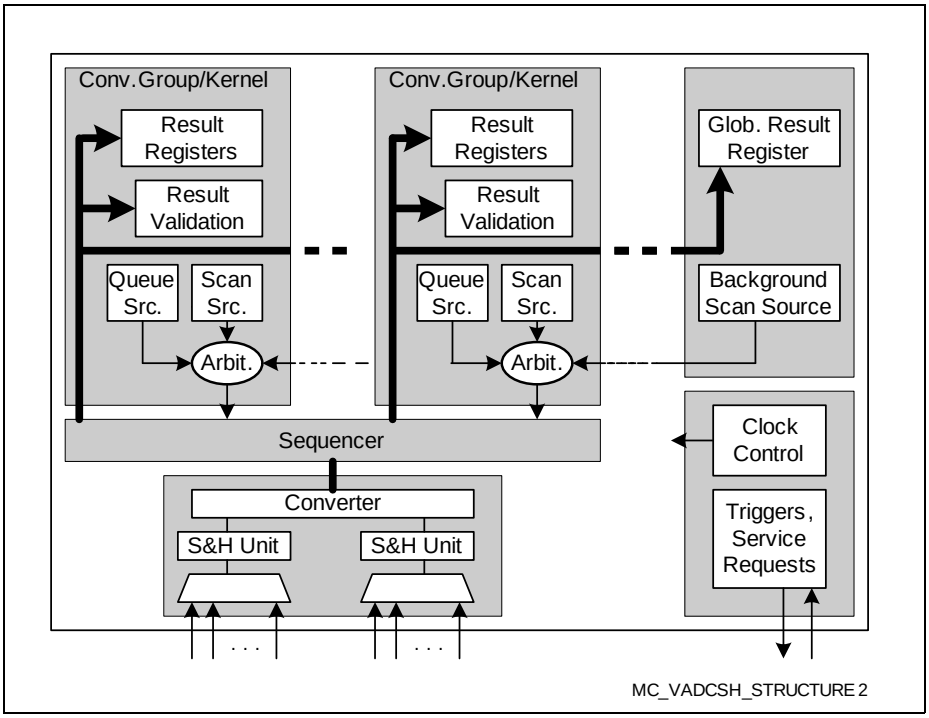



**Figure 16-1 ADC Structure Overview**


Subject to Agreement on the Use of Product Information

## 16.2 Introduction and Basic Structure

The Versatile Analog to Digital Converter module (VADC) of the XMC1300 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.

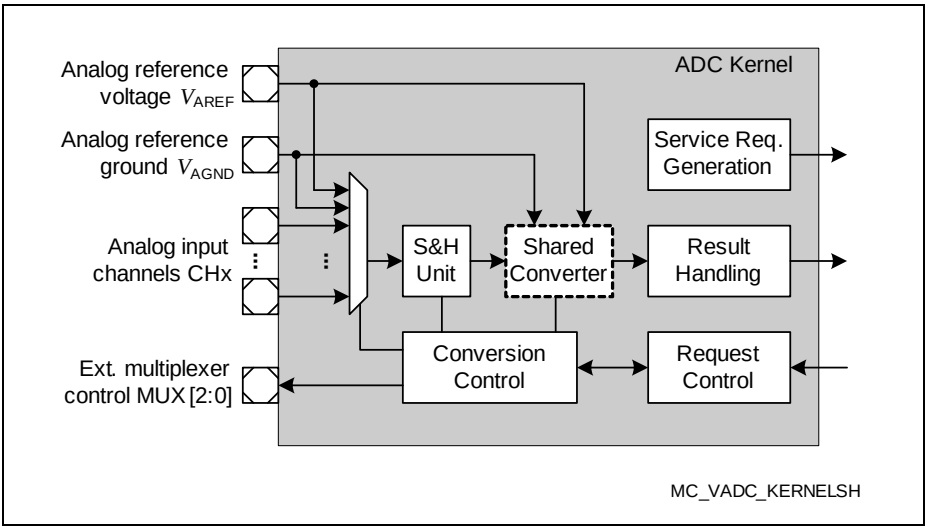



**Figure 16-2 ADC Kernel Block Diagram**

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.


Subject to Agreement on the Use of Product Information

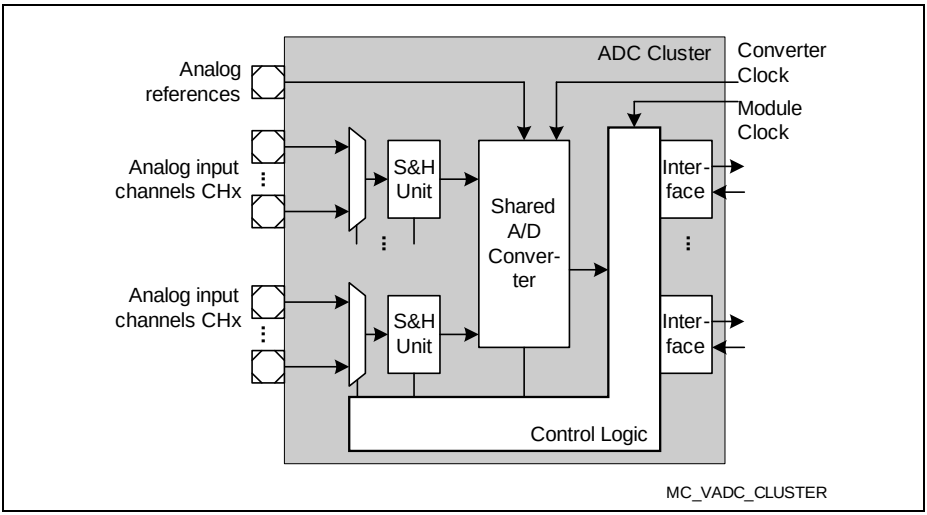



**Figure 16-3 ADC Cluster Structure**

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 clock signal.

### 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.

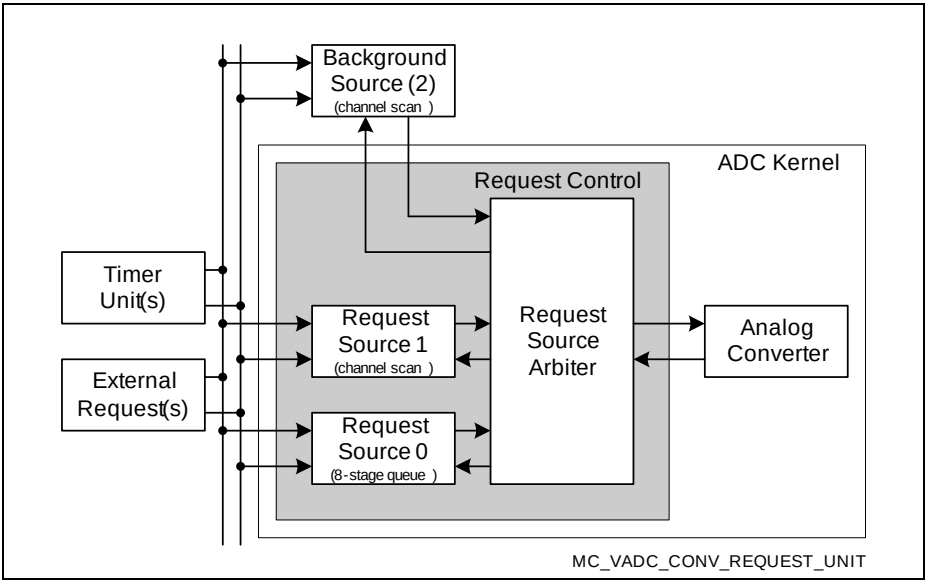



**Figure 16-4 Conversion Request Unit**

### Input Channel Selection

The analog input multiplexer selects one of the available analog inputs (CH0 - CHx[1)]) to be converted. Three sources can select a linear sequence, an arbitrary sequence, or a specific channel. The priorities of these sources can be configured.

1) The availablity of dedicated input channels depends on the package of the used product type. A summary can be found in **Section 16.17.2**.

Additional external analog multiplexers can be controlled automatically, if more separate input channels are required than are built in.

*Note: Not all analog input channels are necessarily available in all packages, due to pin limitations. Please refer to the implementation description in **Section 16.17**.*

### 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.

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 **Section 16.13.1**), 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 (e.g. for digital anti-aliasing filtering) can automatically add up to 4 conversion results before issuing a service request.

Alternatively, an FIR or IIR filter can be enabled that preprocesses the conversion results before sending them to the result register.

Also, result registers can be concatenated to build FIFO structures that store a number of conversion results without overwriting previous data. This increases the allowed CPU latency for retrieving conversion data from the ADC.


Subject to Agreement on the Use of Product Information

### 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 nodes. This allows grouping the requests according to the requirements of the application.

### Safety Features

Safety-aware applications are supported with mechanisms that help to ensure the integrity of a signal path.

**Broken-wire-detection (BWD)** preloads the converter network with a selectable level before sampling the input channel. The result will then reflect the preload value if the input signal is no more connected. If buffer capacitors are used, a certain number of conversions may be required to reach the failure indication level.

**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.


Subject to Agreement on the Use of Product Information

## 16.3 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 XMC1300 uses a switched capacitor field represented by $C_{AINSW}$ (small parasitic capacitances are present at each input pin). During the sample phase, the capacitor field $C_{AINSW}$ is connected to the selected analog input CHx via the input multiplexer (modeled by ideal switches and series resistors $R_{AIN}$).

The switch to CHx is closed during the sample phase and connects the capacitor field to the input voltage $V_{AINx}$.

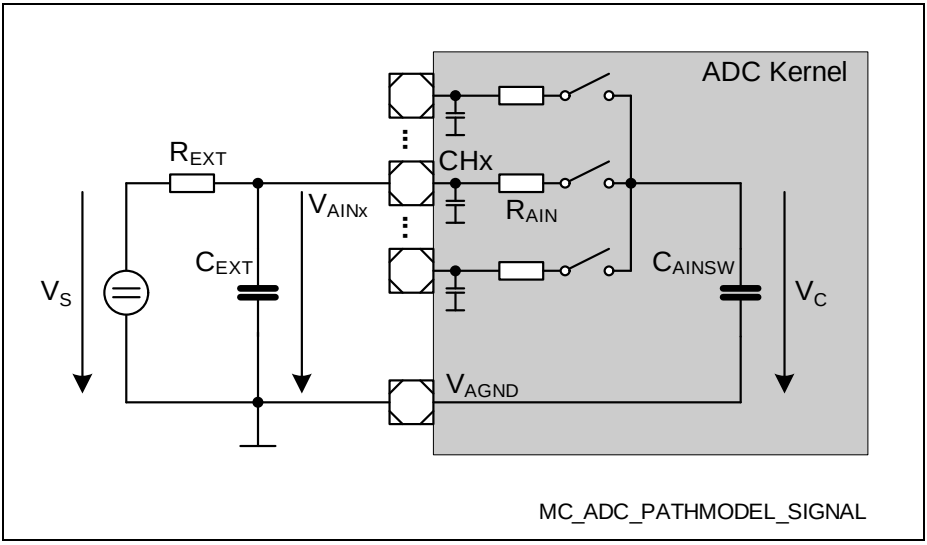



**Figure 16-5 Signal Path Model**

A simplified model for the analog input signal path is given in **Figure 16-5**. 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 $C_{EXT}$ at the analog input pin of the ADC. Like this, mainly $C_{EXT}$ 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 $C_{AINSW}$.

The resulting low-pass filter of $R_{EXT}$ (usually a parameter of the signal source) and $C_{EXT}$ 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 1 $LSB_n$. 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.


Subject to Agreement on the Use of Product Information

### 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).


Subject to Agreement on the Use of Product Information

## 16.4 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

### 16.4.1 General Clocking Scheme and Control

The A/D Converters of the XMC1300 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.

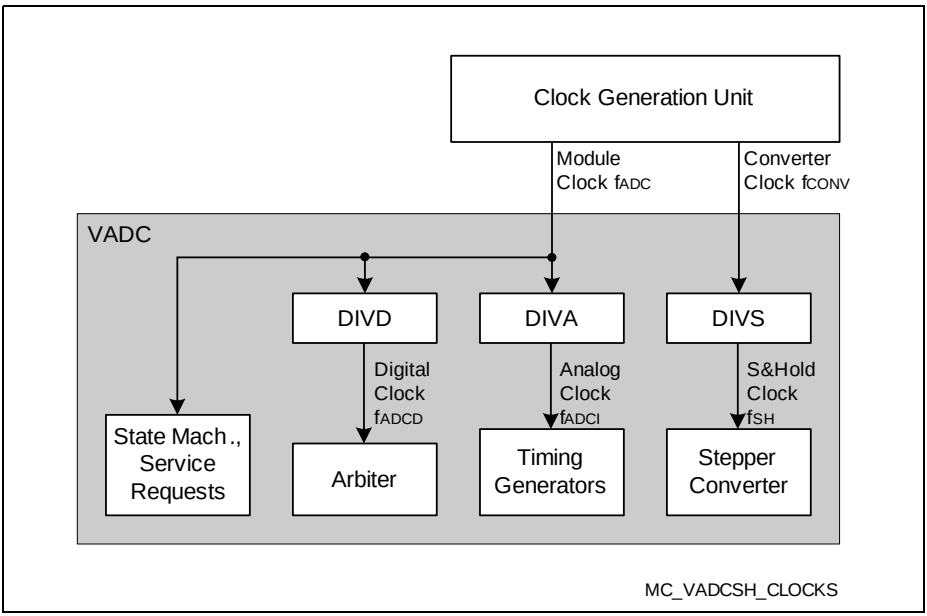



**Figure 16-6 Clock Signal Summary**

### 16.4.2 Register Access Control

Several protection schemes are provided to prevent unintended write access to control bitfields of the VADC.


Subject to Agreement on the Use of Product Information

- 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 16-10** 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).

### 16.4.3 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.


Subject to Agreement on the Use of Product Information

## 16.5 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.

### 16.5.1 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 16-3 Analog Part Power-Down Control Options**

| ANON Settings[1)] | Power-Down Mode |
|---|---|
| All ANON signals are $00_B$ | Analog part off |


Subject to Agreement on the Use of Product Information

**Table 16-3 Analog Part Power-Down Control Options** (cont'd)

| ANON Settings[1)] | Power-Down Mode |
|---|---|
| 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.*

### 16.5.2 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[1)] - - -
- Exit deep sleep mode
- Enable converters by setting ANONC = $01_B$ or $11_B$
- Start new conversions or sequences

1) To reduce power consumption, the VADC module clock can be disabled during the deep sleep phase.


Subject to Agreement on the Use of Product Information

### 16.5.3 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 1 920 cycles (1 920 × 31.25 ns = 60 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.

*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.


Subject to Agreement on the Use of Product Information

### 16.5.4 Reference Voltage Selection

The reference voltage that is required for conversions can be generated on-chip or can be supplied externally ($V_{AREF}$).

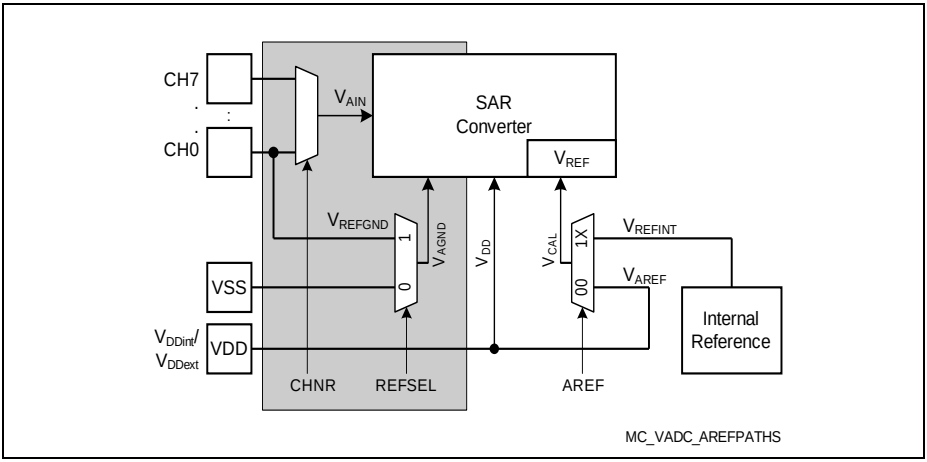



**Figure 16-7 Reference Voltage Sources**

Bitfield AREF in register **SHS0_SHSCFG** selects the intended mode. For internal reference voltage generation the respective operating voltage range (upper / lower) must be selected, external reference voltage mode can only be used in upper voltage range.

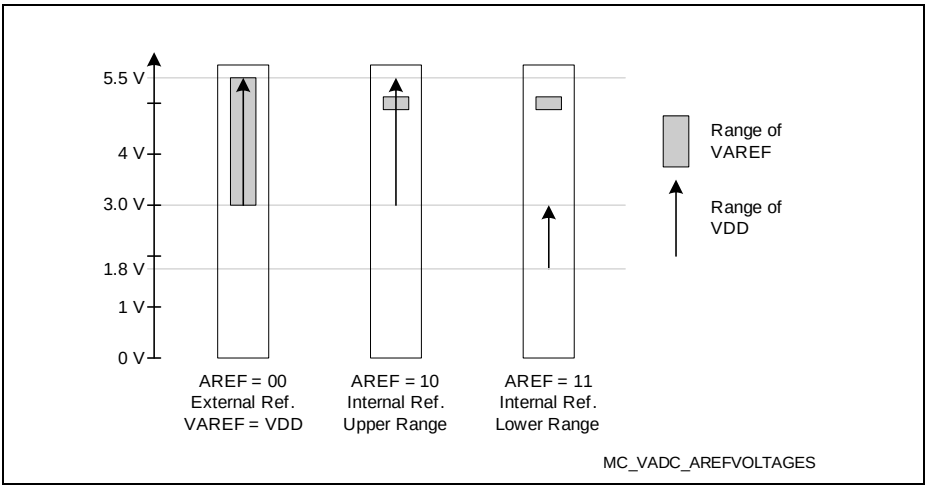



**Figure 16-8 Reference Voltage Ranges**


Subject to Agreement on the Use of Product Information

The nominal internal reference voltage $V_{AREFint}$ is 5 V. The analog input voltage, however, 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.*

### 16.5.5 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**.


Subject to Agreement on the Use of Product Information

## 16.6 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 XMC1300 are listed in **Section 16.17.3**.

*Note: **Figure 16-4 “Conversion Request Unit” on Page 16-6** summarizes the request sources.*


Subject to Agreement on the Use of Product Information

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[1]. This supports application-specific conversion sequences that cannot be covered by a channel scan source. Also, multiple conversions of the same channel within a sequence are supported.
  A queued source converts a series of input channels continuously or on a regular time base. For example, if programmed with medium priority, some input channels can be converted upon a specified event (e.g. synchronized to a PWM). Conversions of lower priority sources are suspended in the meantime.
  Request source 0 is a group-specific 8-stage queued source.
- **A channel scan source** can issue conversion requests for a coherent sequence of input channels. This sequence begins with the highest enabled channel number and continues towards lower channel numbers. All available channels[1] can be enabled for the scan sequence. Each channel is converted once per sequence.
  A scan source converts a series of input channels 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.

1) The availablity of dedicated input channels depends on the package of the used product type. A summary can be found in **Section 16.17.2**.
The background source can only request non-priority channels, i.e. channels that are not selected in registers GxCHASS. Priority channels are reserved for the group-specific request sources 0 and 1.

### 16.6.1 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 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.

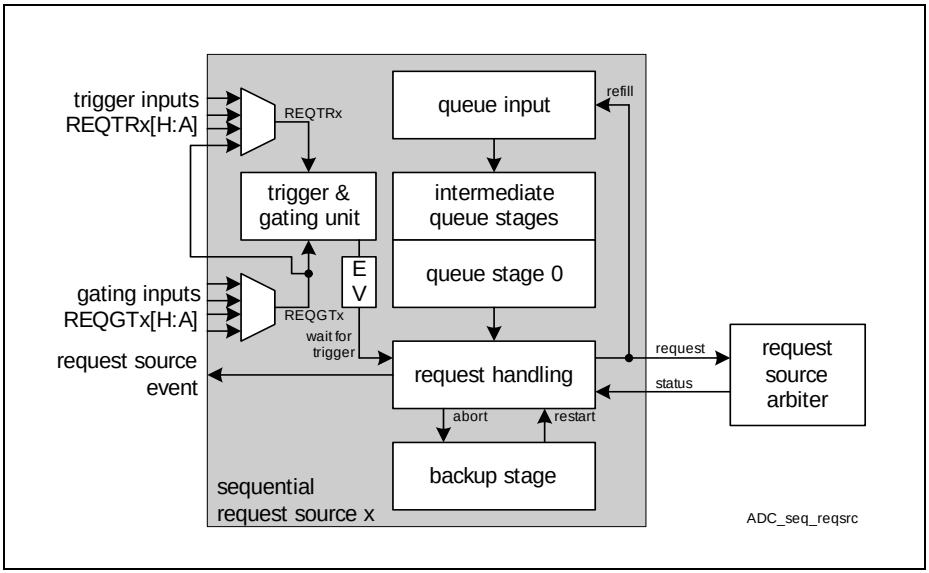



**Figure 16-9 Queued Request Source**

A sequence is defined by entering conversion requests into the queue input register (**GxQINR0 (x = 0 - 1)**). Each entry selects the channel to be converted and can enable an external trigger, generation of an interrupt, and an automatic refill (i.e. copy this entry to the top of the queue after conversion). The entries are stored in the queue buffer stages.

The content of stage 0 (**GxQ0R0 (x = 0 - 1)**) selects the channel to be converted next. When the requested conversion is started, the contents of this queue stage is invalidated and copied to the backup stage. Then the next queue entry can be handled (if available).

*Note: The contents of the queue stages cannot be modified directly, but only by writing to the queue input or by flushing the queue.*
*The current status of the queue is shown in register **GxQSR0 (x = 0 - 1)**.*
*If all queue entries have automatic refill selected, the defined conversion sequence can be repeated without re-programming.*

### Properties of the Queued Request Source

Queued request source 0 provides 8 buffer stages and can handle sequences of up to 8 input channel entries. It supports short application-specific conversion sequences, especially for timing-critical sequences containing also multiple conversions of the same channel.

### Queued Source Operation

**Configure the queued request source** by executing the following actions:

- Define the sequence by writing the entries to the queue input **GxQINR0 (x = 0 - 1)**. Initialize the complete sequence before enabling the request source, because with enabled refill feature, software writes to QINRx are not allowed.
- If hardware trigger or gating is desired, select the appropriate trigger and gating inputs and the proper transitions by programming **GxQCTRL0 (x = 0 - 1)**.
  Enable the trigger and select the gating mode by programming bitfield ENGT in register **GxQMR0 (x = 0 - 1)**.[1)]
- Enable the corresponding arbitration slot (0) to accept conversion requests from the queued source (see register **GxARBPR (x = 0 - 1)**).

**Start a queued sequence** by generating a trigger event:

- If a hardware trigger is selected and enabled, generate the configured transition at the selected input signal, e.g. from a timer or an input pin.
- Generate a software trigger event by setting GxQMR0.TREV = 1.

1) If PDOUT signals from the ERU are used, initialize the ERU accordingly before enabling the gate inputs to avoid un expected signal transitions.


Subject to Agreement on the Use of Product Information

- 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 16-10**. If a request source event is detected, it sets the corresponding indication flag in register **GxSEFLAG (x = 0 - 1)**. These flags can also be set by writing a 1 to the corresponding bit position, whereas writing 0 has no effect. The indication flags can be cleared by SW by writing a 1 to the corresponding bit position in register **GxSEFCLR (x = 0 - 1)**.
The interrupt enable bit is taken from stage 0 for a normal sequential conversion, or from the backup stage for a repeated conversion after an abort.

The service request output line SRx that is selected by the request source event interrupt node pointer bitfields in register **GxSEVNP (x = 0 - 1)** becomes activated each time the related request source event is detected (and enabled by GxQ0R0.ENSI, or GxQBUR0.ENSI respectively) or the related bit position in register **GxSEFLAG (x = 0 - 1)** is written with a 1 (this write action simulates a request source event).


Subject to Agreement on the Use of Product Information

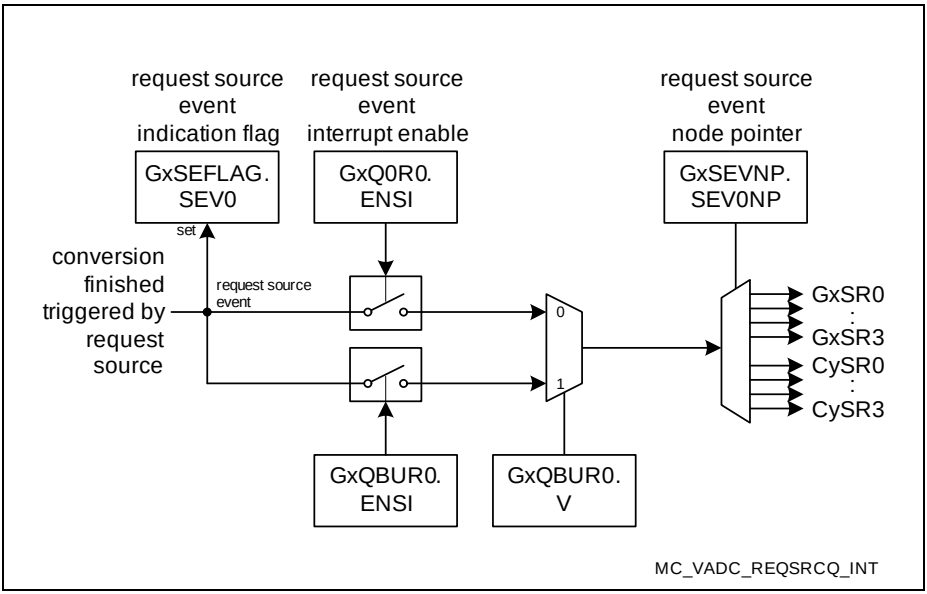



**Figure 16-10 Interrupt Generation of a Queued Request Source**

## 16.6.2 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.

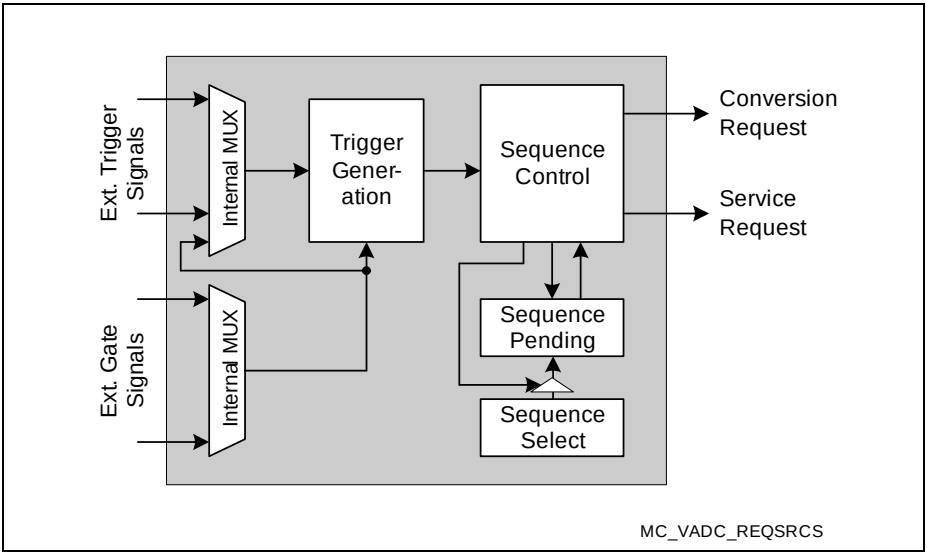



**Figure 16-11 Scan Request Source**


Subject to Agreement on the Use of Product Information

### Scan Source Operation

**Configure the scan request source** by executing the following actions:

- Select the input channels for the sequence by programming **GxASSEL (x = 0 - 1)** or **BRSSELx (x = 0 - 1)**
- If hardware trigger or gating is desired, select the appropriate trigger and gating inputs and the proper signal transitions by programming **GxASCTRL (x = 0 - 1)** or **BRSCTRL**. Enable the trigger and select the gating mode by programming **GxASMR (x = 0 - 1)** or **BRSMR**.[1)]
- Define the load event operation (handling of pending bits, autoscan mode) by programming **GxASMR (x = 0 - 1)** or **BRSMR**.
  A load event with bit LDM = 0 copies the content of **GxASSEL (x = 0 - 1)** or **BRSSELx (x = 0 - 1)** to **GxASPND (x = 0 - 1)** or **BRSPNDx (x = 0 - 1)** (overwrite mode). This starts a new scan sequence and aborts any pending conversions from a previous scan sequence.
  A load event with bit LDM = 1 OR-combines the content of **GxASSEL (x = 0 - 1)** or **BRSSELx (x = 0 - 1)** to **GxASPND (x = 0 - 1)** or **BRSPNDx (x = 0 - 1)** (combine mode). This starts a scan sequence that includes pending conversions from a prvious scan sequence.
- Enable the corresponding arbitration slot (1) to accept conversion requests from the channel scan source (see register **GxARBPR (x = 0 - 1)**).

**Start a channel scan sequence** by generating a load event:

- If a hardware trigger is selected and enabled, generate the configured transition at the selected input signal, e.g. from a timer or an input pin.
- Generate a software load event by setting LDEV = 1 (**GxASMR (x = 0 - 1)** or **BRSMR**).
- Generate a load event by writing the scan pattern directly to the pending bits in **GxASPND (x = 0 - 1)** or **BRSPNDx (x = 0 - 1)**. The pattern is copied to **GxASSEL (x = 0 - 1)** or **BRSSELx (x = 0 - 1)** and a load event is generated automatically.
  In this case, a scan sequence can be defined and started with a single data write action, e.g. under PEC control (provided that the pattern fits into one register).

*Note: If autoscan is enabled, a load event is generated automatically each time a request source event occurs when the scan sequence has finished. This permanently repeats the defined scan sequence (autoscan).*

**Stop or abort an ongoing scan sequence** by executing 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.

1) If PDOUT signals from the ERU are used, initialize the ERU accordingly before enabling the gate inputs to avoid un expected signal transitions.


Subject to Agreement on the Use of Product Information

- 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)**.[1)]

1) Please refer to **"Service Request Generation" on Page 16-64**.

## 16.7 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.

**Figure 16-12** 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 XMC1300, 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).

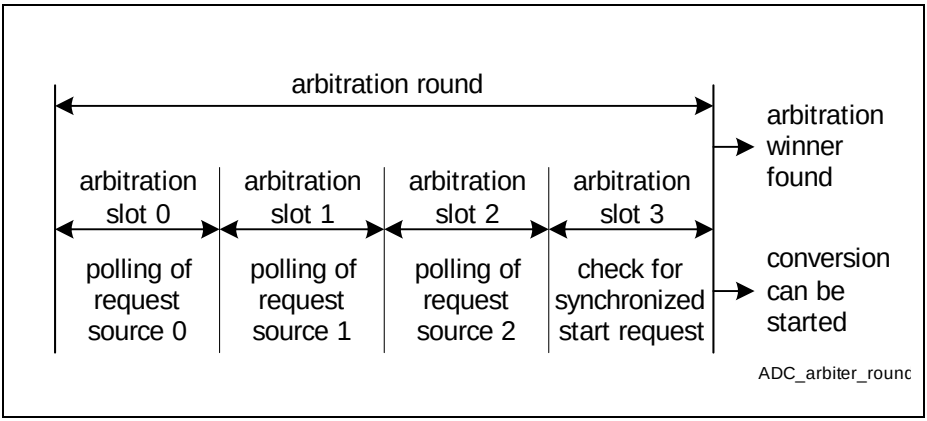



**Figure 16-12 Arbitration Round with 4 Arbitration Slots**


Subject to Agreement on the Use of Product Information

### 16.7.1 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 XMC1300, more can be programmed to obtain the same arbiter timing for different products.

The duration of an arbitration slot is configurable $t_{Slot}$ = (DIVD+1) / $f_{ADC}$.
The duration of an arbitration round, therefore, is $t_{ARB}$ = 4 × $t_{Slot}$.

The period of the arbitration round introduces a timing granularity to detect an incoming conversion request signal and the earliest point to start the related conversion. This granularity can introduce a jitter of maximum one arbitration round. The jitter can be reduced by minimizing the period of an arbitration round.

To achieve a reproducible reaction time (constant delay without jitter) between the trigger event of a conversion request (e.g. by a timer unit or due to an external event) and the start of the related conversion, mainly the following two options exist. For both options, the converter has to be idle and other conversion requests must not be pending for at least one arbiter round before the trigger event occurs:

- If bit **GxARBCFG (x = 0 - 1)**.ARBM = 0, the **arbiter runs permanently**. In this mode, synchronized conversions of more than one ADC kernel are possible.[1)]
  The trigger for a conversion request has to be generated synchronously to the arbiter timing. Incoming triggers should have exactly n-times the granularity of the arbiter (n = 1, 2, 3,...). In order to allow some flexibility, the duration of an arbitration slot can be programmed in cycles of $f_{ADC}$.
- If bit **GxARBCFG (x = 0 - 1)**.ARBM = 1, the **arbiter stops after an arbitration round** when no conversion request have been found pending any more. The arbiter is started again if at least one enabled request source indicates a pending conversion request. The trigger for a conversion request does not need to be synchronous to the arbiter timing.
  In this mode, parallel conversions are not possible for synchronization slave 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.

1) For more information, please refer to **"Synchronization of Conversions" on Page 16-56**.

### 16.7.2 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 16-13**:
    Conversion A is requested (t1) and started (t2). Conversion B is then requested (t3), but started only after completion of conversion A (t4).
  - **Cancel-inject-repeat mode**: the current conversion is aborted, the conversion of the arbitration winner is started after the abortion (3 $f_{ADC}$ cycles).
    The aborted conversion request is restored in the corresponding request source and takes part again in the next arbitration round. This mode provides minimum jitter for the higher priority conversions, but reduces the overall throughput.
    Example in **Figure 16-13**:
    Conversion A is requested (t6) and started (t7). Conversion B is then requested (t8) and started (t9), while conversion A is aborted but requested again. When conversion B is complete (t10), conversion A is restarted.
    Exception: If both requests target the same result register with wait-for-read mode active (see **Section 16.11.3**), the current conversion cannot be aborted.

*Note: A cancelled conversion can be repeated automatically in each case, or it can be discarded if it was cancelled. This is selected for each source by bit RPTDIS in the corresponding source's mode register.*

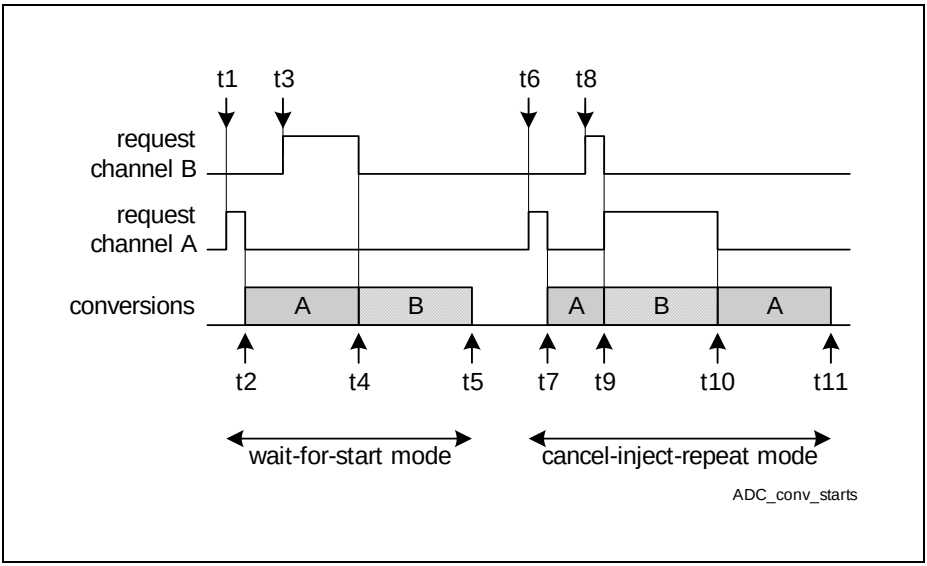



**Figure 16-13 Conversion Start Modes**

The conversion start mode can be individually programmed for each request source by bits in register **GxARBPR (x = 0 - 1)** and is applied to all channels requested by the source. In this example, channel A is issued by a request source with a lower priority than the request source requesting the conversion of channel B.

## 16.8 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 16-20 “Result Storage Options” on Page 16-46**.
- **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.

### 16.8.1 Channel Parameters

Each analog input channel is configured by its associated channel control register.

*Note: For the safety feature “Broken Wire Detection”, refer to **Section 16.13.1**.*

The following features can be defined for each channel:

- The conversion class defines the result width and the sample time
- Generation of channel events and the result value band, if used
- Target of the result defining the target register and the position within the register

The group-specific input class registers define the sample time and data conversion mode for each channel of the respective group that selects them via bitfield ICLSEL in its channel control register GxCHCTRy.


Subject to Agreement on the Use of Product Information

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.

### 16.8.2 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 16-14**), the internal trigger and data handling features for channel CH0 can be used.
- The channel settings for both conversions can be different (boundary values, service requests, etc.).

In typical low-cost AC-drive applications, only one common current sensor is used to determine the phase currents. Depending on the applied PWM pattern, the measured value has different meanings and the sample points have to be precisely located in the PWM period. **Figure 16-14** 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.


Subject to Agreement on the Use of Product Information

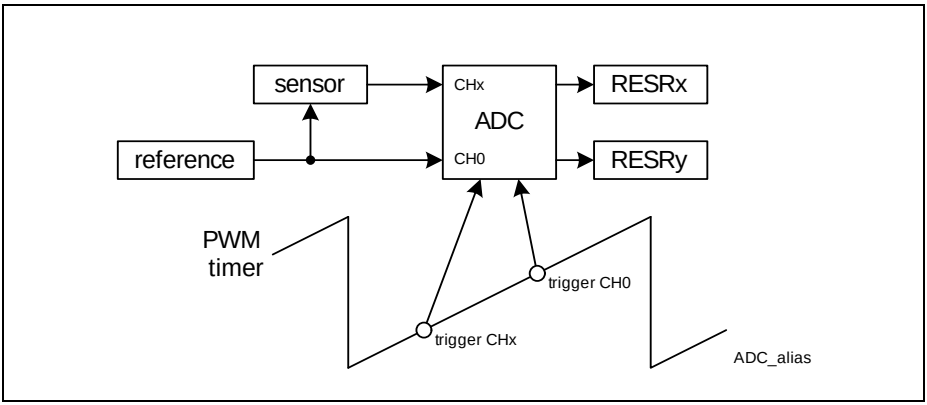



**Figure 16-14 Alias Feature**

### 16.8.3 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 (**Section 16.10**).
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.

### 16.8.4 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 16-15**).

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.

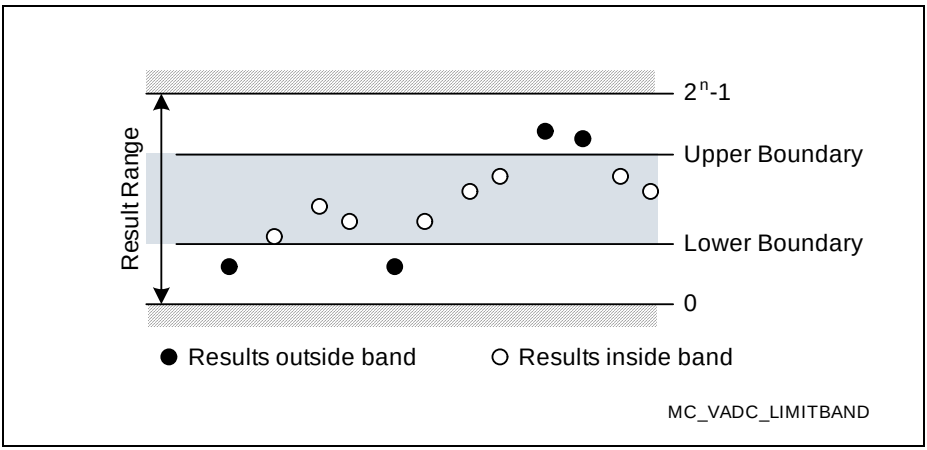



**Figure 16-15 Result Monitoring through Limit Checking**


Subject to Agreement on the Use of Product Information

A result value is considered inside the defined band when both of the following conditions are true:

- the value is less than or equal to the selected upper boundary
- the value is greater than or equal to the selected lower boundary

The result range can also be divided into two areas:
To select the lower part as valid band, set the lower boundary to the minimum value ($000_H$) and set the upper boundary to the highest intended value.
To select the upper part as valid band, set the upper boundary to the maximum value ($FFF_H$) and set the lower boundary to the lowest intended value.

**Finding Extrema (Peak Detection)**

The limit checking mechanism uses standard conversions and, therefore, always can provide the actual conversion result that was used for comparison. Combining this with a special FIFO mode, that only updates the corresponding FIFO stage if the result was above (or below) the current value of the stage, provides the usual conversion results and at the same time stores the highest (or lowest) result of a conversion sequence. For this operation the FIFO stage below the standard result must be selected as the compare value. Mode selection is done via bitfield FEN in register GxRCRy.

Before starting a peak detection sequence, write a reasonable start value to the result bitfield in the peak result register (e.g. $0000_H$ to find the maximum and $FFFF_H$ to find the minimum).

### 16.8.5 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 16-16**).

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)


Subject to Agreement on the Use of Product Information

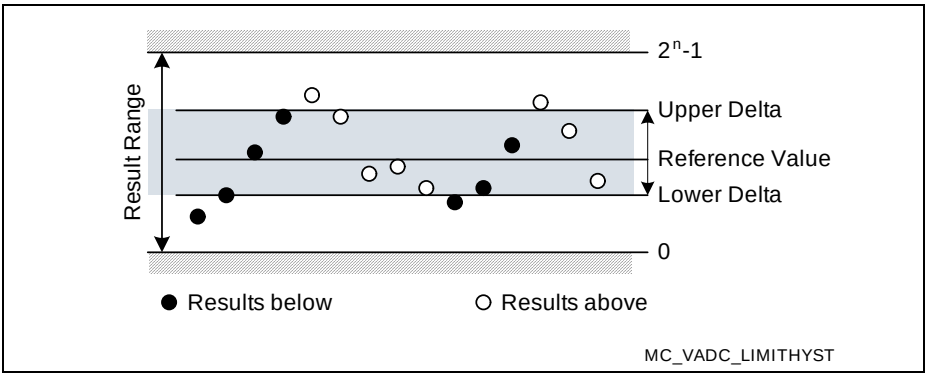



**Figure 16-16 Result Monitoring through Compare with Hysteresis**

### 16.8.6 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.*

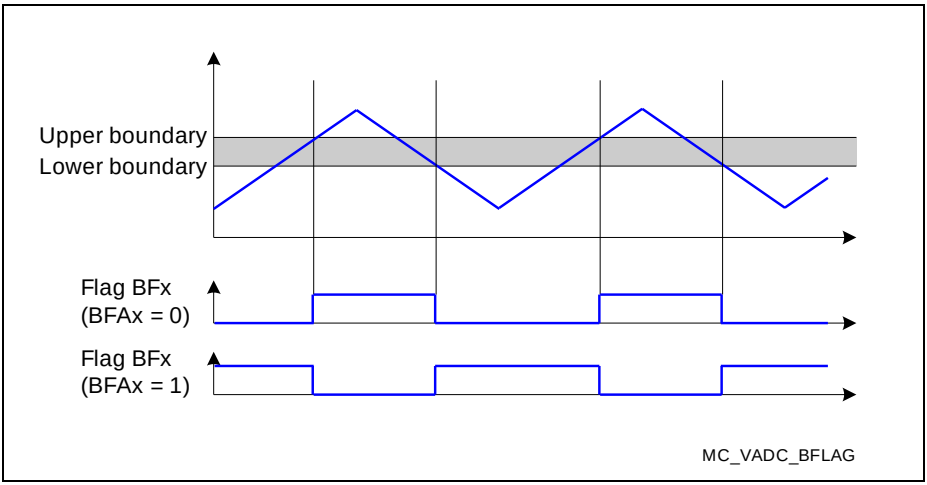



**Figure 16-17 Boundary Flag Switching**


Subject to Agreement on the Use of Product Information

Boundary flags can be switched by each compare operation, or the influence of compare operations can be restricted to the active phases of the corresponding request source gate signal (see **GxBFLC (x = 0 - 1)**).

*Note: If a boundary flag is used together with Fast Compare Mode, it is recommended not to direct results from other channels to the corresponding result register.*

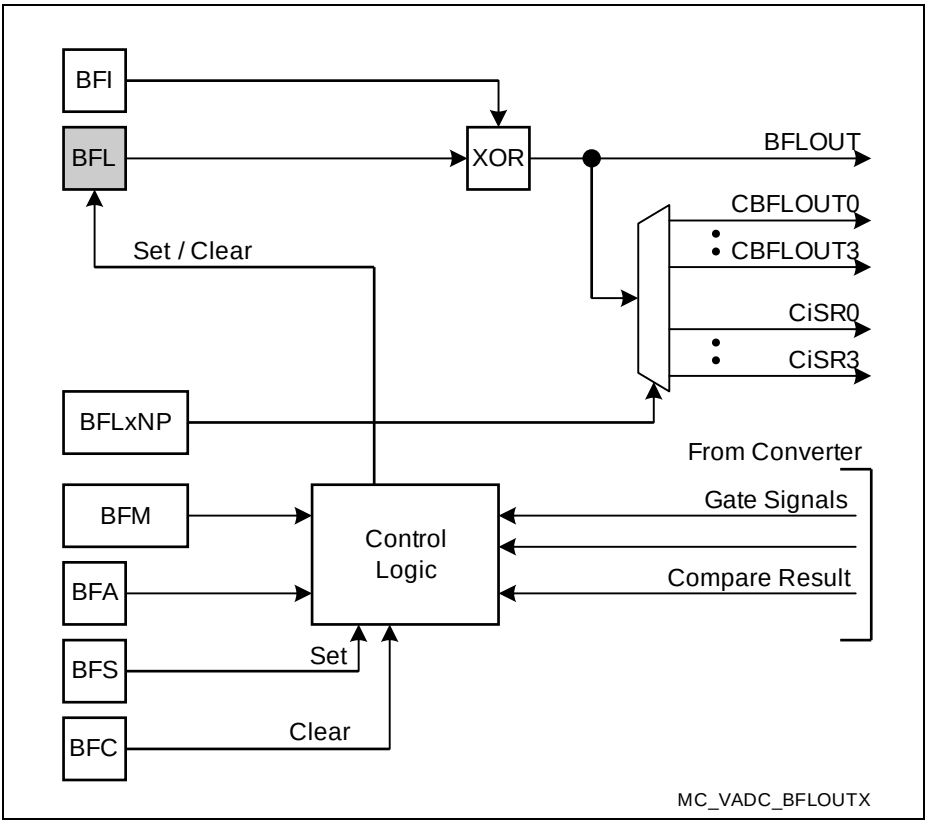



**Figure 16-18 Boundary Flag Control**

A boundary flag BFLy is assigned to result register GxRESy and thus to an arbitrary channel.

## 16.9 Conversion Scheduling

The converter clusters of the VADC combine one high-speed converter with several sample&hold units which can operate in parallel.

The XMC1300 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 **Section** ) is activated.

### General Timing Behavior

Conversion results are returned to the digital logic after the time interval defined by the parameters $f_{ADCI}$, 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 XMC1300.

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.


Subject to Agreement on the Use of Product Information

## 16.10 Conversion Timing

The total time required for a conversion comprises the time from the start of the sample phase[1)] 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.*

### 16.10.1 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)
- 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:

$$t_{CN} = SST \times t_{ADC} + (4 \times t_{SH})^{2)} + (N + 8) \times t_{SH} + (5 \times t_{ADC})^{3)}$$

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 $12 \times t_{SH}$)[4)], so the maximum complete conversion round including all 2 S&H units will take:

$$t_{CR} = SST \times t_{ADC} + 4 \times t_{SH} + (N+8) \times t_{SH} + (N+8) \times t_{SH} + 5 \times t_{ADC} + 12 \times t_{SH}^{4)}$$

1) 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.

2) 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.

3) 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.

4) Offset and gain calibration steps are executed alternating, where gain calibration takes more time than offset calibration (9 cycles).


Subject to Agreement on the Use of Product Information

### Timing Examples

System assumptions:
$f_{CONV}$ = 32 MHz, DIVS = 0, i.e. $t_{SH} = t_{CONV}$ = 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)[1)]

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)[2)]

1) 10.5 calibration clock cycles are the average resulting from alternating offset and gain calibration steps.

2) 12 calibration clock cycles are the maximum resulting from a gain calibration step.


Subject to Agreement on the Use of Product Information

### 16.10.2 Compatible Timing Mode

The conversion timing depends on the following factors (partly user-definable):

- The ADC conversion clock frequency, where $f_{ADCI} = f_{ADC} / (DIVA+1)$[1)]
- The selected sample time, where $t_S = (2 + STC) \times t_{ADCI}$
  (STC = additional sample time, see also **Table 16-11**)
- 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[1)], $f_{ADCI}$ = 8 MHz i.e. $t_{ADCI}$ = 125 ns
$t_{CN12C} = (2 + 12 + 2) \times t_{ADCI} + 2 \times t_{ADC} = 16 \times 125$ ns $+ 2 \times 31.25$ ns = 2 063 ns

System assumptions for 12-bit uncalibrated conversions:
$f_{ADC}$ = 32 MHz i.e. $t_{ADC}$ = 31.25 ns, DIVA = 3[1)], $f_{ADCI}$ = 8 MHz i.e. $t_{ADCI}$ = 125 ns
$t_{CN12U} = (2 + 12) \times t_{ADCI} + 2 \times t_{ADC} = 14 \times 125$ ns $+ 2 \times 31.25$ ns = 1 813 ns

System assumptions for 10-bit conversions:
$f_{ADC}$ = 32 MHz i.e. $t_{ADC}$ = 31.25 ns, DIVA = 3[1)], $f_{ADCI}$ = 8 MHz i.e. $t_{ADCI}$ = 125 ns
$t_{CN10} = (2 + 10) \times t_{ADCI} + 2 \times t_{ADC} = 12 \times 125$ ns $+ 2 \times 31.25$ ns = 1 563 ns

1) $f_{ADCI}$ 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.

## 16.11 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.

Conversion result handling comprises the following functions:

- **Storage of Conversion Results** to user-configurable registers
- **Data Alignment** according to result width and endianess
- **Wait-for-Read Mode** to avoid loss of data
- **Result Event Generation**
- Data reduction or anti-aliasing filtering (see **Section 16.11.6**)

### 16.11.1 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


Subject to Agreement on the Use of Product Information

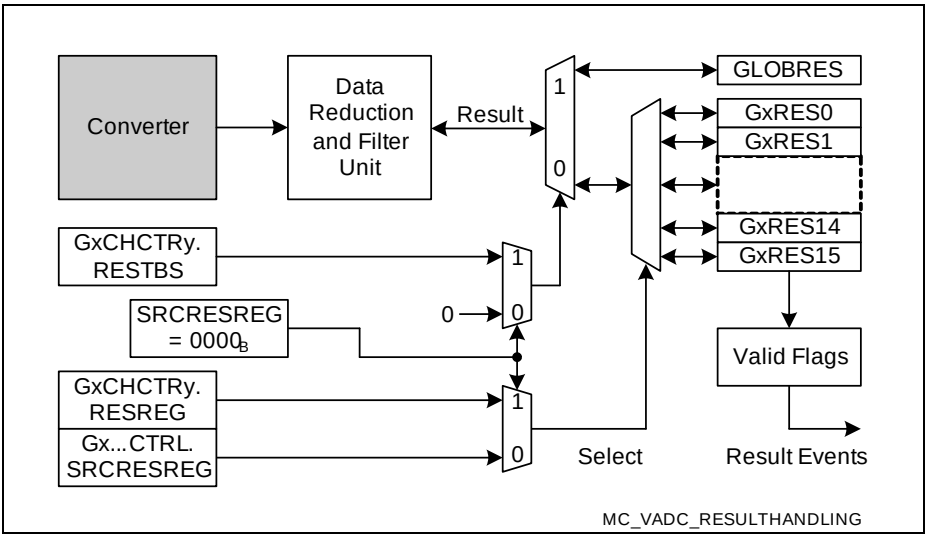



**Figure 16-19 Conversion Result Storage**

### 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.


Subject to Agreement on the Use of Product Information

### 16.11.2 Data Alignment

The position of a conversion result value within the selected result register depends on 3 configurations (summary in **Figure 16-20**):

- The selected result width (12/10/8 bits, selected by the conversion mode)
- The selected result position (Left/Right-aligned
- The selected data accumulation mode (data reduction)

These options provide the conversion results in a way that minimizes data handling for the application software.

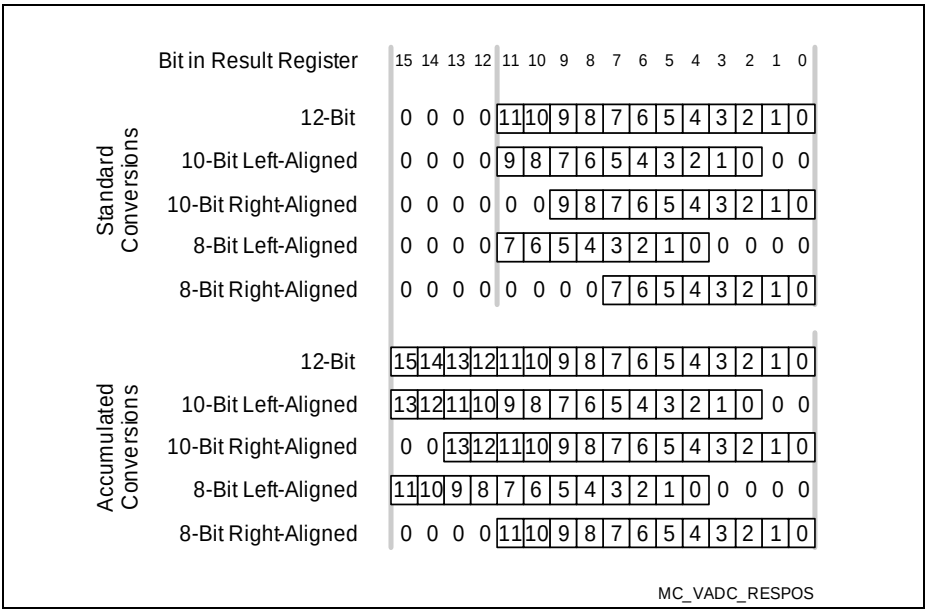



**Figure 16-20 Result Storage Options**

Bitfield RESULT can be written by software to provide the reference value for Fast Compare Mode. In this mode, bits 11-2 are evaluated, the other bits are ignored.

### 16.11.3 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[1)].

Wait-for-read mode automatically suspends the start of a conversion for this channel from this source until the current result has been read. So a conversion or a conversion sequence can be requested by a hardware or software trigger, while each conversion is only started after the result of the previous one has been read. This automatically aligns the conversion sequence with the CPU capability to read the formerly converted result (latency).

If wait-for-read mode is enabled for a result register (bit GxRCRy.WFR = 1), a request source does not generate a conversion request while the targeted result register contains valid data (indicated by the valid flag VF = 1) or if a currently running conversion targets the same result register.

If two request sources target the same result register with wait-for-read mode selected, a higher priority source cannot interrupt a lower priority conversion request started before the higher priority source has requested its conversion. Cancel-inject-repeat mode does not work in this case. In particular, this must be regarded if one of the involved sources is the background source (which usually has lowest priority). If the higher priority request targets a different result register, the lower priority conversion can be cancelled and repeated afterwards.

*Note: Wait-for-read mode is ignored for synchronized conversions of synchronization slaves (see **Section 16.12**).*

1) Repeated conversions of a single channel that use a separate result register will not destroy other results, but rather update their own previous result value. This way, always the actual signal data is available in the result register.


Subject to Agreement on the Use of Product Information

### 16.11.4 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 16-21**). This is enabled by setting bitfield GxRCRy.FEN = $01_B$.
Conversion results are stored to the register with the highest index of a FIFO structure. Software reads the values from the FIFO register with the lowest index.

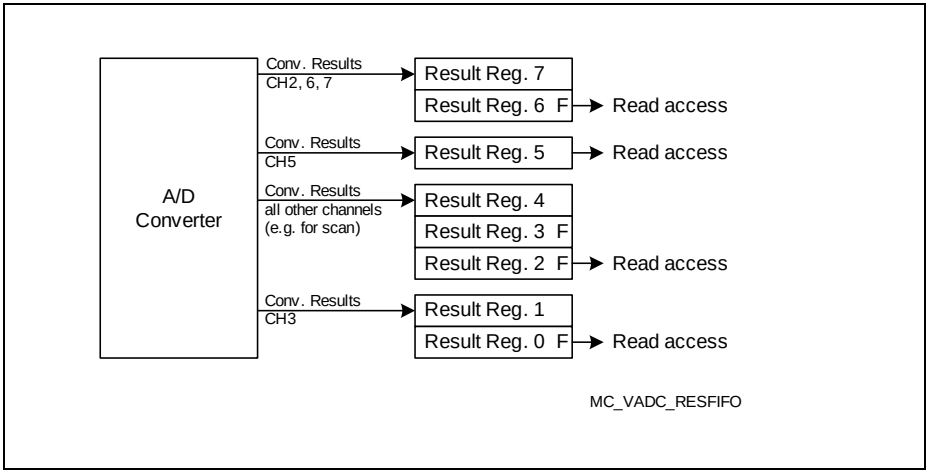



**Figure 16-21 Result FIFO Buffers**

In the example shown the result registers have been configured in the following way:

- 2-stage buffer consisting of result registers 7-6
- dedicated result register 5
- 3-stage buffer consisting of result registers 4-3-2
- 2-stage buffer consisting of result registers 1-0

**Table 16-4** summarizes the required configuration of result registers if they are combined to build result FIFO buffers.

**Table 16-4 Properties of Result FIFO Registers**

| Function | Input Stage | Intermed. Stage | Output Stage |
|---|---|---|---|
| Result target | YES | no | no |
| Application read | no | no | YES |
| Data reduction mode | YES | no | no |
| Wait-for-read mode | YES | no | no |
| Result event interrupt | no | no | YES |
| FIFO enable (FEN) | $00_B$ | $01_B$ | $01_B$ |
| Registers in example | 7, 4, 1 | 3 | 6, 2, 0 |

*Note: If enabled, a result interrupt is generated for each data word in the FIFO.*

### 16.11.5 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.

### 16.11.6 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.


Subject to Agreement on the Use of Product Information

**Table 16-5 Function of Bitfield DRCTR**

| DRCTR | Standard Data Reduction Mode (DMM = $00_B$) | DRCTR | Result Filtering Mode (DMM = $01_B$)[1] |
|---|---|---|---|
| $0000_B$ | Data Reduction disabled | $0000_B$ | FIR filter: a=2, b=1, c=0 |
| $0001_B$ | Accumulate 2 result values | $0001_B$ | FIR filter: a=1, b=2, c=0 |
| $0010_B$ | Accumulate 3 result values | $0010_B$ | FIR filter: a=2, b=0, c=1 |
| $0011_B$ | Accumulate 4 result values | $0011_B$ | FIR filter: a=1, b=1, c=1 |
| $0100_B$ | Reserved | $0100_B$ | FIR filter: a=1, b=0, c=2 |
| $0101_B$ | Reserved | $0101_B$ | FIR filter: a=3, b=1, c=0 |
| $0110_B$ | Reserved | $0110_B$ | FIR filter: a=2, b=2, c=0 |
| $0111_B$ | Reserved | $0111_B$ | FIR filter: a=1, b=3, c=0 |
| $1000_B$ | Reserved | $1000_B$ | FIR filter: a=3, b=0, c=1 |
| $1001_B$ | Reserved | $1001_B$ | FIR filter: a=2, b=1, c=1 |
| $1010_B$ | Reserved | $1010_B$ | FIR filter: a=1, b=2, c=1 |
| $1011_B$ | Reserved | $1011_B$ | FIR filter: a=2, b=0, c=2 |
| $1100_B$ | Reserved | $1100_B$ | FIR filter: a=1, b=1, c=2 |
| $1101_B$ | Reserved | $1101_B$ | FIR filter: a=1, b=0, c=3 |
| $1110_B$ | 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 ≠ $01_B$.


Subject to Agreement on the Use of Product Information

### Standard Data Reduction Mode

The data reduction mode can be used as digital filter for anti-aliasing or decimation purposes. It accumulates a maximum of 4 conversion values to generate a final result.

Each result register can be individually enabled for data reduction, controlled by bitfield DRCTR in registers **G0CHCTRy (y = 0 - 7)**. The data reduction counter DRC indicates the actual status of the accumulation.

*Note: Conversions for other result registers can be inserted between conversions to be accumulated.*

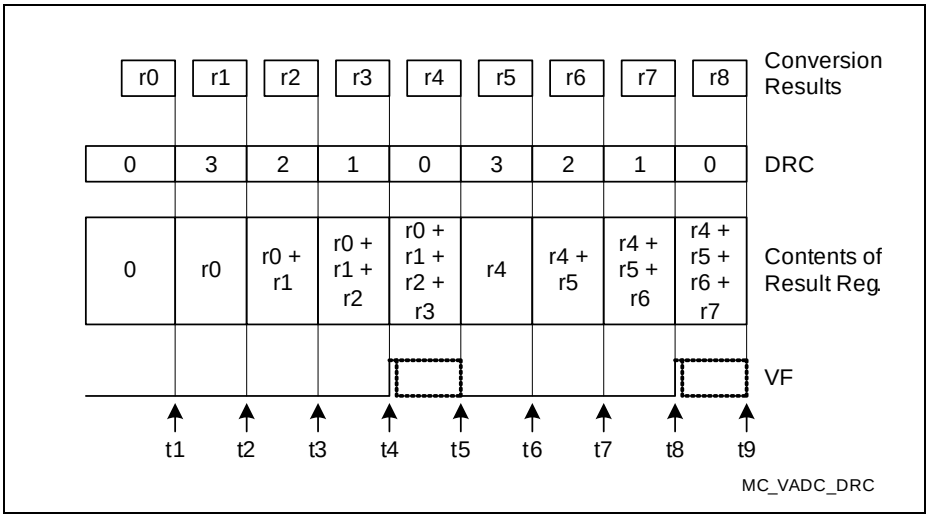



**Figure 16-22 Standard Data Reduction Filter**

This example shows a data reduction sequence of 4 accumulated conversion results. Eight conversion results (r0 … r7) are accumulated and produce 2 final results.

When a conversion is complete and stores data to a result register that has data reduction mode enabled, the data handling is controlled by the data reduction counter DRC:

- If DRC = 0 (t1, t5, t9 in the example), the conversion result is stored to the register. DRC is loaded with the contents of bitfield DRCTR (i.e. the accumulation begins).
- If DRC > 0 (t2, t3, t4 and t6, t7, t8 in the example), the conversion result is added to the value in the result register.
  DRC is decremented by 1.
- If DRC becomes 0, either decremented from 1 (t4 and t8 in the example) or loaded from DRCTR, the valid bit for the respective result register is set and a result register event occurs.

The final result must be read before the next data reduction sequence starts (before t5 or t9 in the example). This automatically clears the valid flag.

*Note: Software can clear the data reduction counter DRC by clearing the corresponding valid Flag (via **GxVFR (x = 0 - 1)**).*

The response time to read the final data reduction results can be increased by associating the adjacent result register to build a result FIFO (see **Figure 16-23**). In this case, the final result of a data reduction sequence is loaded to the adjacent register. The value can be read from this register until the next data reduction sequence is finished (t8 in the 2nd example).

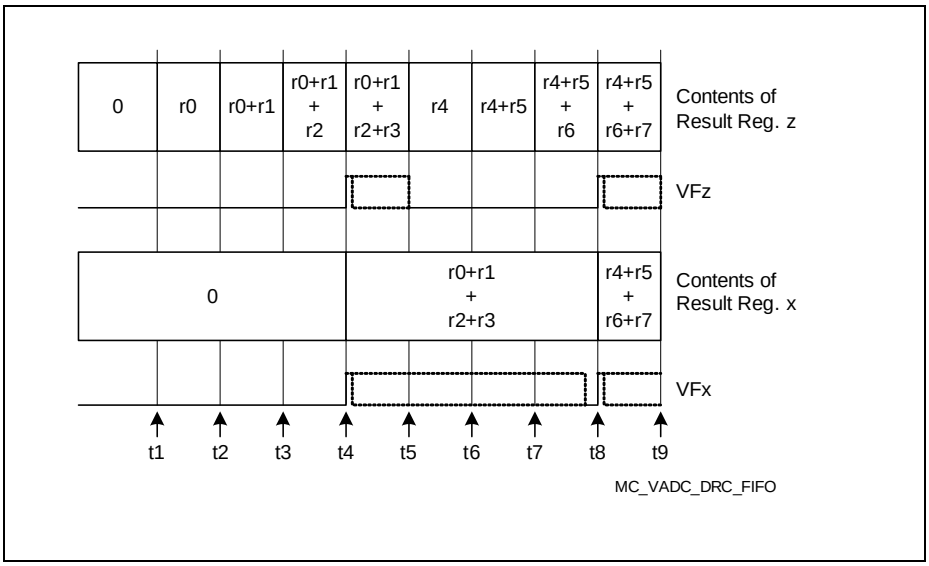



**Figure 16-23 Standard Data Reduction Filter with FIFO Enabled**

### Finite Impulse Response Filter Mode (FIR)

The FIR filter (see **Figure 16-24**) provides 2 result buffers for intermediate results (RB1, RB2) and 3 configurable tap coefficients (a, b, c).

The conversion result and the intermediate result buffer values are added weighted with their respective coefficients to form the final value for the result register. Several predefined sets of coefficients can be selected via bitfield DRCTR (coding listed in **Table 16-5**) 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.*

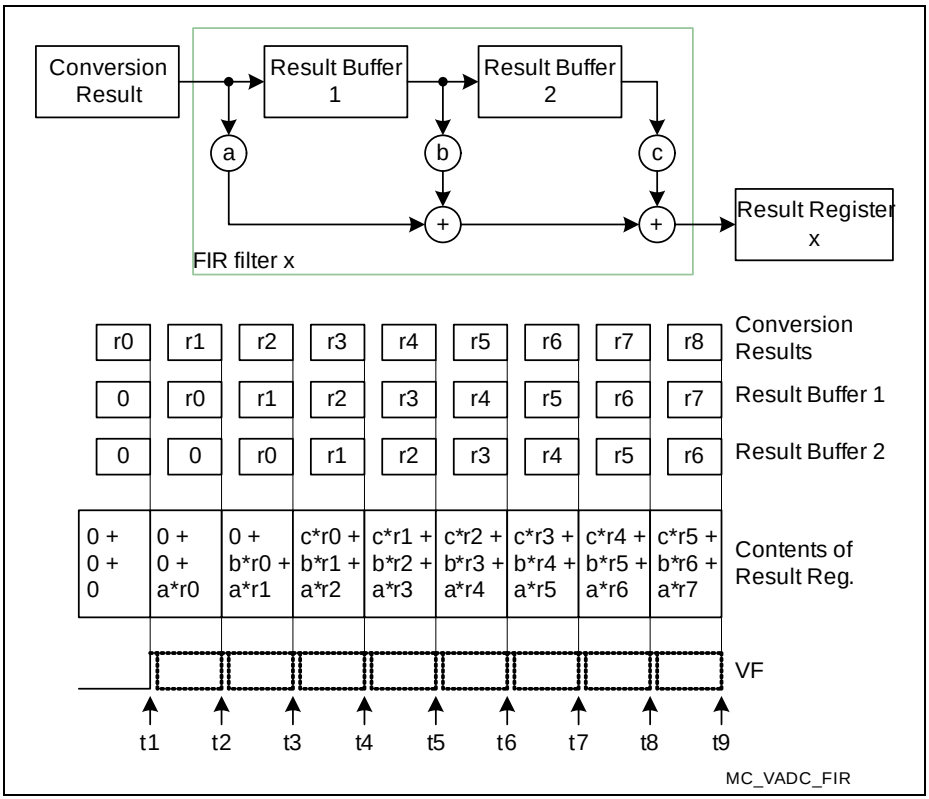



**Figure 16-24 FIR Filter Structure**

*Note: The filter registers are cleared while bitfield DMM ≠ $01_B$.*

### Infinite Impulse Response Filter Mode (IIR)

The IIR filter (see **Figure 16-25**) provides a result buffer (RB) and 2 configurable coefficients (a, b). It represents a first order low-pass filter.

The conversion result, weighted with the respective coefficient, and a fraction of the previous result are added to form the final value for the result register. Several predefined sets of coefficients can be selected via bitfield DRCTR (coding listed in **Table 16-5**) in registers **G0RESy (y = 0 - 15)** and **GLOBRES**. These coefficients lead to a gain of 4 to the ADC result producing a 14-bit value. The valid flag (VF) is activated for each sample after activation, i.e. for each sample generates a valid result.

*Note: Conversions for other result registers can be inserted between conversions to be filtered.*

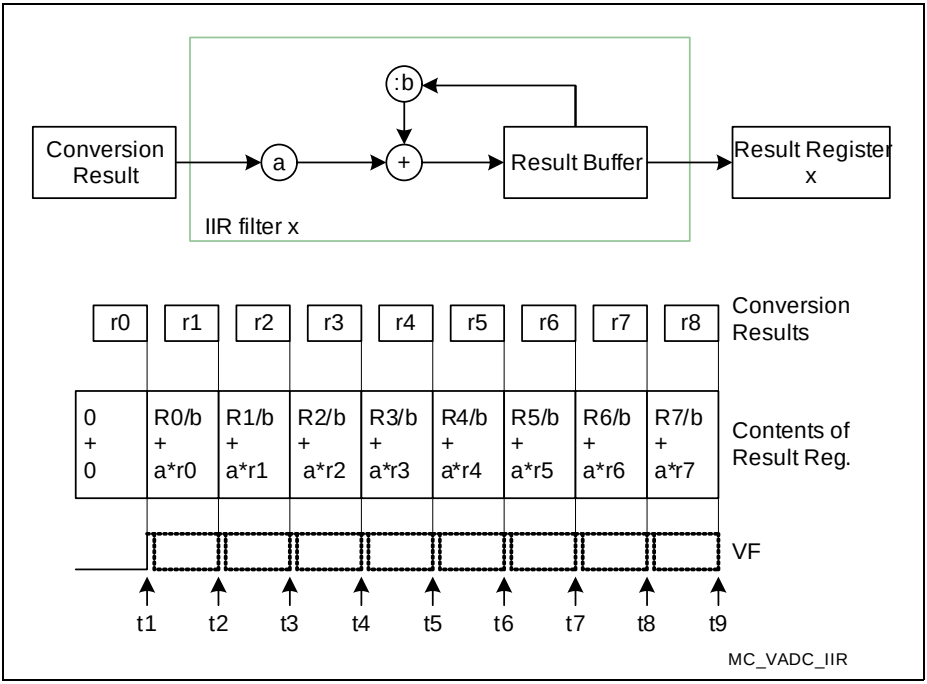



**Figure 16-25 IIR Filter Structure**

*Note: The filter registers are cleared while bitfield DMM ≠ $01_B$.*

### Difference Mode

Subtracting the contents of result register 0 from the actual result puts the results of the respective channel in relation to another signal. No software action is required.

The reference channel must store its result(s) into result register 0. The reference value can be determined once and then be used for a series of conversions, or it can be converted before each related conversion.

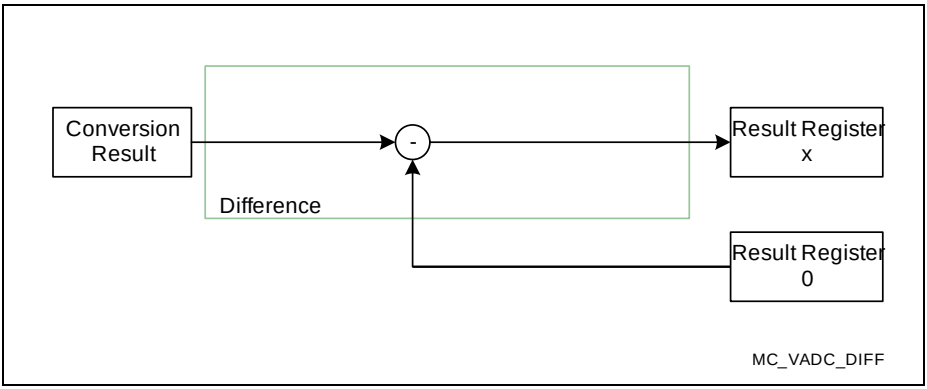



**Figure 16-26 Result Difference**

## 16.12 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[1]. This optimizes e.g. the control of electrical drives.
**Equidistant sampling** supports conversions in a fixed raster with minimum jitter. This optimizes e.g. filter algorithms or audio applications.

### 16.12.1 Synchronized Conversions for Parallel Sampling

Several independent ADC kernels[1] implemented in the XMC1300 can be synchronized for simultaneous measurements of analog input channels. While no parallel conversion is requested, the kernels can work independently.

The synchronization mechanism for parallel conversions ensures that the sample phases of the related channels start simultaneously. Synchronized kernels convert the same channel that is requested by the master. Different values for the resolution and the sample phase length of each kernel for a parallel conversion are supported.

A parallel conversion can be requested individually for each input channel (one or more). In the example shown in **Figure 16-27**, 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.
  – A parallel conversion request is always handled with highest priority and cancel-inject-repeat mode.
  – Wait-for-read mode is ignored in the slave. Previous results may be overwritten, in particular, if the same result register is used by other conversions.

1) For a summary, please refer to **"Synchronization Groups in the XMC1300" on Page 16-154**.

- The arbiter must run permanently (bit **GxARBPR (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.

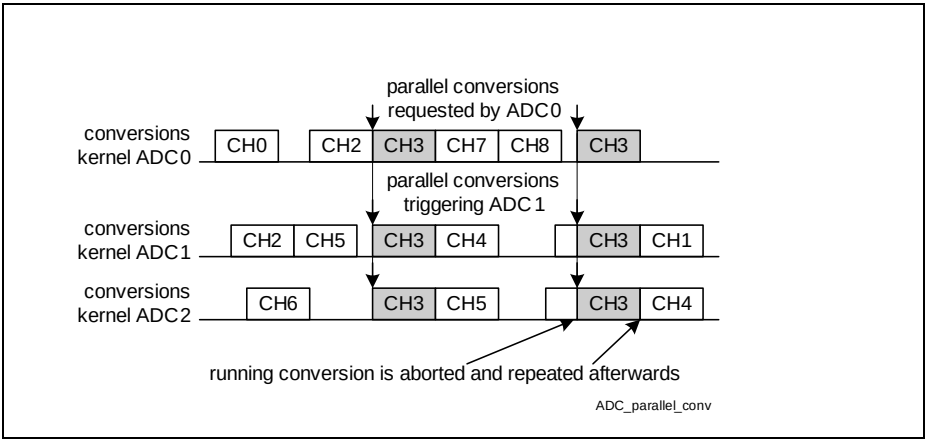



**Figure 16-27 Parallel Conversions**

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 16-28**) 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 16-13 “Synchronization Groups in the XMC1300” on Page 16-154**).

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 **Section 16.8.2**), analog signals from different input channels can be converted. This is advantageous if e.g. CH0 is used as alternate reference.*

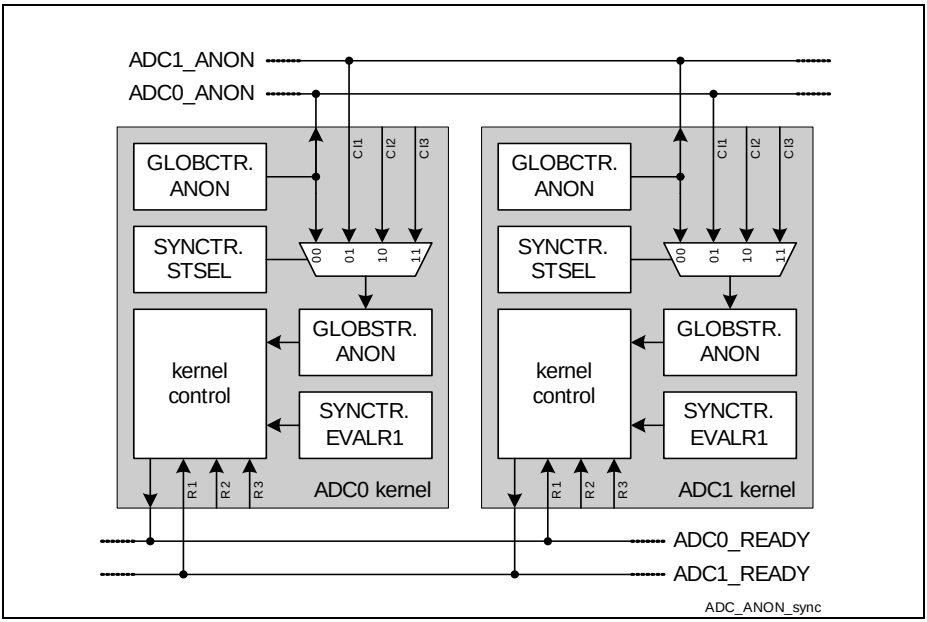



**Figure 16-28 Synchronization via ANON and Ready Signals**

### 16.12.2 Equidistant Sampling

To optimize the input data e.g. for filter or audio applications, conversions can be executed in a fixed timing raster. Conversions for equidistant sampling are triggered by an external signal (e.g. a timer). To generate the trigger signal synchronous to the arbiter, the ADC provides an output signal (ARBCNT) that is activated once per arbitration round and serves as timing base for the trigger timer. In this case, the arbiter must run permanently (**GxARBPR (x = 0 - 1)**.ARBM = 0). If the timer has an independent time base, the arbiter can be stopped while no requests are pending. The preface time (see **Figure 16-29**) must be longer than one arbitration round and the highest possible conversion time.

Select timer mode (TMEN = 1 in register **GxQCTRL0 (x = 0 - 1)** or **GxASCTRL (x = 0 - 1)**) for the intended source of equidistant conversions. In timer mode, a request of this source is triggered and arbitrated, but only started when the trigger signal is removed (see **Figure 16-29**) and the converter is idle.

To ensure that the converter is idle and the start of conversion can be controlled by the trigger signal, the equidistant conversion requests must receive highest priority. The preface time between request trigger and conversion start must be long enough for a currently active conversion to finish.

The frequency of signal REQTRx defines the sampling rate and its high time defines the preface time interval where the corresponding request source takes part in the arbitration.

Depending on the used request source, equidistant sampling is also supported for a sequence of channels. It is also possible to do equidistant sampling for more than one request source in parallel if the preface times and the equidistant conversions do not overlap.

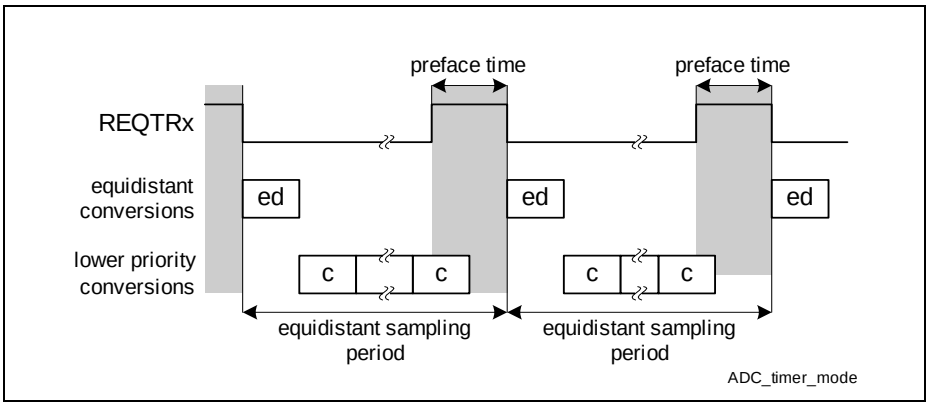



**Figure 16-29 Timer Mode for Equidistant Sampling**


Subject to Agreement on the Use of Product Information

## 16.13 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,

### 16.13.1 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.

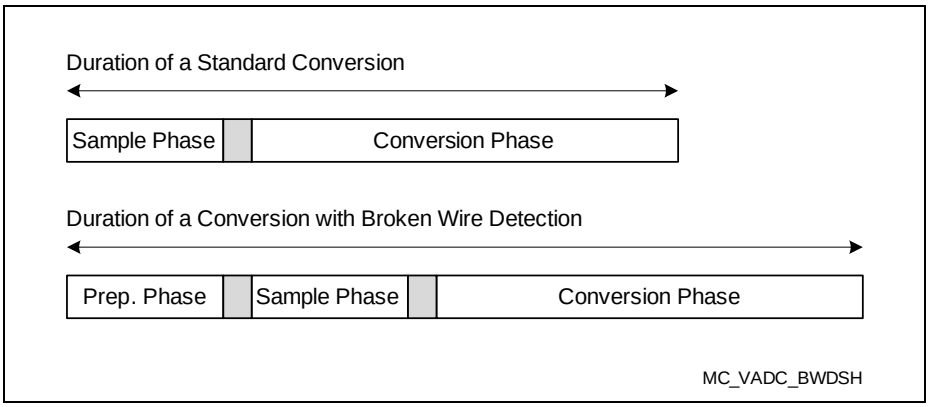



**Figure 16-30 Broken Wire Detection**


Subject to Agreement on the Use of Product Information

### 16.13.2 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.


Subject to Agreement on the Use of Product Information

## 16.14 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 16-31** the following conversions are done: --4-32-31-30-2-1-0--4-32-31-30-2-1-0--…
- **Single-step mode** converts one external channel of the configured sequence when the selected channel is encountered. In the example in **Figure 16-31** the following conversions are done: --4-32-2-1-0--4-31-2-1-0--4-30-2-1-0--4-32-… (Single-step mode works best with one channel)
- **Steady mode** converts the configured external channel when the selected channel is encountered. In the example in **Figure 16-31** the following conversions are done: --4-32-2-1-0--4-32-2-1-0--4-32-2-1-0--…

*Note: The example in **Figure 16-31** has an external multiplexer connected to channel CH3. The start selection value EMUXSET is assumed as 2.*

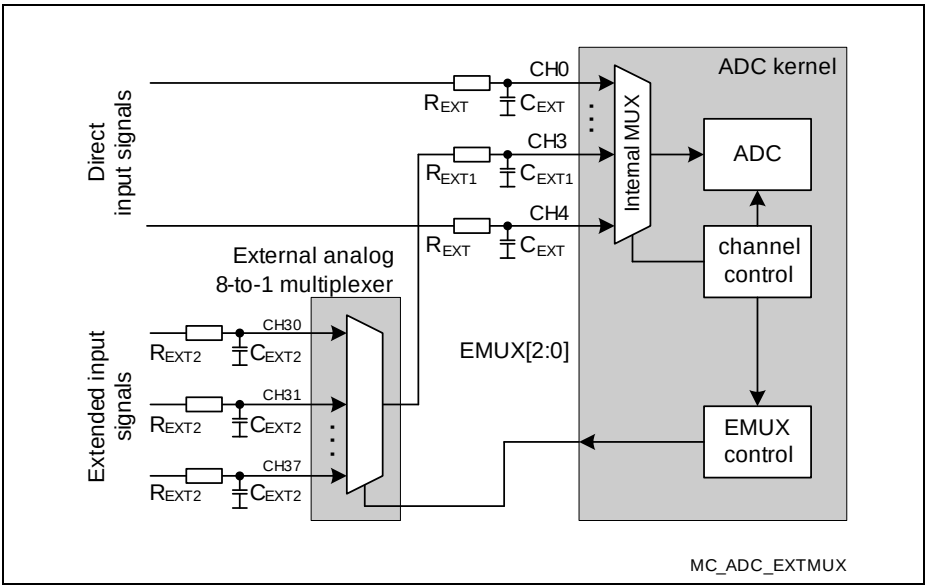



**Figure 16-31 External Analog Multiplexer Example**


Subject to Agreement on the Use of Product Information

Bitfield EMUXACT determines the control information sent to the external multiplexer. In single-step mode, EMUXACT is updated after each conversion of an enabled channel. If EMUXACT = $000_B$ it is reloaded from bitfield EMUXSET, otherwise it is decremented by 1.

Additional external channels may have different properties due to the modified signal path. Local filters may be used at the additional inputs ($R_{EXT2}$-$C_{EXT2}$ on CH3x in **Figure 16-31**). For applications where the external multiplexer is located far from the ADC analog input, it is recommended to add an RC filter directly at the analog input of the ADC ($R_{EXT1}$-$C_{EXT1}$ on CH3 in **Figure 16-31**).

*Note: Each RC filter limits the bandwidth of the analog input signal.*

Conversions for external channels, therefore, use the alternate conversion mode setting CME. This automatically selects a different conversion mode if required.

Switching the external multiplexer usually requires an additional settling time for the input signal. Therefore, the alternate sample time setting STCE is applied each time the external channel is changed. This automatically fulfills the different sampling time requirements in this case.

In each group an arbitrary channel can be assigned to external multiplexer control (register **GxEMUXCTR (x = 0 - 1)**). Each available port interface selects the group whose control lines are output (register **EMUXSEL**).

### Control Signals

The external channel number that controls the external multiplexer can be output in standard binary format or Gray-coded. Gray code avoids intermediate multiplexer switching when selecting a sequence of channels, because only one bit changes at a time. **Table 16-6** indicates the resulting codes.

**Table 16-6 EMUX Control Signal Coding**

| Channel | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Binary | $000_B$ | $001_B$ | $010_B$ | $011_B$ | $100_B$ | $101_B$ | $110_B$ | $111_B$ |
| Gray | 000 | 001 | 011 | 010 | 110 | 111 | 101 | 100 |

### Operation Without External Multiplexer

If no external multiplexers are used in an application, the reset values of the control registers provide the appropriate setup.

EMUXMODE = $00_B$ disables the automatic EMUX control.

Since the control output signals are alternate port output signals, they are only visible at the respective pins if explicitly selected.


Subject to Agreement on the Use of Product Information

## 16.15 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 16-15 “Digital Connections in the XMC1300” on Page 16-156**).

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 16-12**.*

## 16.16 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 16-8**). The exact register location is obtained by adding the respective register offset to the base address (see **Table 16-7**) 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
- ###Y$_H$ means: ###0$_H$ + y × 0004$_H$, for y = 0 - N (depends on register type)
- Z###$_H$ means: 4803 4000$_H$ + 0###$_H$, (SHS0)

**Table 16-7 Registers Address Space**

| Module | Base Address | End Address | Note |
|---|---|---|---|
| VADC0 | 4803 0000$_H$ | 4803 0BFF$_H$ | |

**Table 16-8 Registers Overview**

| Register Short Name | Register Long Name | Offset Addr. | Access Mode | | Page Num. |
|---|---|---|---|---|---|
| | | | Read | Write | |
| ID | Module Identification Register | 0008$_H$ | U, PV | BE | 16-71 |
| CLC | Clock Control Register | 0000$_H$ | U, PV | PV | 16-72 |
| OCS | OCDS Control and Status Register | 0028$_H$ | U, PV | PV | 16-73 |
| GLOBCFG | Global Configuration Register | 0080$_H$ | U, PV | U, PV | 16-75 |
| ACCPROT0 | Access Protection Register 0 | 0088$_H$ | U, PV | U, PV 1) | 16-78 |
| ACCPROT1 | Access Protection Register 1 | 008C$_H$ | U, PV | U, PV 1) | 16-79 |
| GxARBCFG | Arbitration Configuration Register | X480$_H$ | U, PV | U, PV | 16-85 |
| GxARBPR | Arbitration Priority Register | X484$_H$ | U, PV | U, PV | 16-88 |
| GxCHASS | Channel Assignment Register, Group x | X488$_H$ | U, PV | U, PV | 16-81 |
| GxRRASS | Result Assignment Register, Group x | X48C$_H$ | U, SV | U, PV | 16-82 |
| GxQCTRL0 | Queue 0 Source Control Register, Group x | X500$_H$ | U, PV | U, PV | 16-89 |


Subject to Agreement on the Use of Product Information

**Table 16-8 Registers Overview** (cont'd)

| Register Short Name | Register Long Name | Offset Addr. | Access Mode | | Page Num. |
|---|---|---|---|---|---|
| | | | Read | Write | |
| GxQMR0 | Queue 0 Mode Register, Group x | $X504_H$ | U, PV | U, PV | 16-91 |
| GxQSR0 | Queue 0 Status Register, Group x | $X508_H$ | U, PV | U, PV | 16-93 |
| GxQINR0 | Queue 0 Input Register, Group x | $X510_H$ | U, PV | U, PV | 16-95 |
| GxQ0R0 | Queue 0 Register 0, Group x | $X50C_H$ | U, PV | U, PV | 16-97 |
| GxQBUR0 | Queue 0 Backup Register, Group x | $X510_H$ | U, PV | U, PV | 16-99 |
| GxASCTRL | Autoscan Source Control Register, Group x | $X520_H$ | U, PV | U, PV | 16-101 |
| GxASMR | Autoscan Source Mode Register, Group x | $X524_H$ | U, PV | U, PV | 16-103 |
| GxASSEL | Autoscan Source Channel Select Register, Group x | $X528_H$ | U, PV | U, PV | 16-105 |
| GxASPND | Autoscan Source Pending Register, Group x | $X52C_H$ | U, PV | U, PV | 16-106 |
| BRSCTRL | Background Request Source Control Register | $0200_H$ | U, PV | U, PV | 16-107 |
| BRSMR | Background Request Source Mode Register | $0204_H$ | U, PV | U, PV | 16-109 |
| BRSSELx | Background Request Source Channel Select Register, Group x | $018Y_H$ | U, PV | U, PV | 16-111 |
| BRSPNDx | Background Request Source Channel Pending Register, Group x | $01CY_H$ | U, PV | U, PV | 16-112 |
| GxCHCTRy | Channel x Control Register | $X60Y_H$ | U, PV | U, PV | 16-113 |
| GxICLASS0 | Input Class Register 0, Group x | $X4A0_H$ | U, PV | U, PV | 16-115 |
| GxICLASS1 | Input Class Register 1, Group x | $X4A4_H$ | U, PV | U, PV | 16-115 |
| GLOBICLASS0 | Input Class Register 0, Global | $00A0_H$ | U, PV | U, PV | 16-115 |
| GLOBICLASS1 | Input Class Register 1, Global | $00A4_H$ | U, PV | U, PV | 16-115 |
| GxALIAS | Alias Register | $X4B0_H$ | U, PV | U, PV | 16-130 |


Subject to Agreement on the Use of Product Information

**Table 16-8 Registers Overview** (cont'd)

| Register Short Name | Register Long Name | Offset Addr. | Access Mode | | Page Num. |
|---|---|---|---|---|---|
| | | | Read | Write | |
| GxBOUND | Boundary Select Register, Group x | $X4B8_H$ | U, PV | U, PV | 16-132 |
| GLOBBOUND | Global Boundary Select Register | $00B8_H$ | U, PV | U, PV | 16-132 |
| GxBFL | Boundary Flag Register, Group x | $X4C8_H$ | U, PV | U, PV | 16-133 |
| GxBFLS | Boundary Flag Software Register, Group x | $X4CC_H$ | U, PV | U, PV | 16-134 |
| GxBFLC | Boundary Flag Control Register, Group x | $X4D0_H$ | U, PV | U, PV | 16-135 |
| GxBFLNP | Boundary Flag Node Pointer Register, Group x | $X4D4_H$ | U, PV | U, PV | 16-136 |
| GxRCRy | Group x Result Control Register y | $X68Y_H$ | U, PV | U, PV | 16-118 |
| GxRESy | Group x Result Register y | $X70Y_H$ | U, PV | U, PV | 16-120 |
| GxRESDy | Group x Result Register y (debug view) | $X78Y_H$ | U, PV | U, PV | 16-121 |
| GLOBRCR | Global Result Control Register | $0280_H$ | U, PV | U, PV | 16-123 |
| GLOBRES | Global Result Register | $0300_H$ | U, PV | U, PV | 16-124 |
| GLOBRESD | Global Result Register (debug view) | $0380_H$ | U, PV | U, PV | 16-124 |
| GxVFR | Valid Flag Register, Group x | $X5F8_H$ | U, PV | U, PV | 16-125 |
| GxSYNCTR | Synchronization Control Register | $X4C0_H$ | U, PV | U, PV | 16-137 |
| GxEMUXCTR | External Multiplexer Control Register, Group x | $X5F0_H$ | U, PV | U, PV | 16-138 |
| EMUXSEL | External Multiplexer Select Register | $03F0_H$ | U, PV | U, PV | 16-140 |


Subject to Agreement on the Use of Product Information

**Table 16-8 Registers Overview** (cont'd)

| Register Short Name | Register Long Name | Offset Addr. | Access Mode | | Page Num. |
|---|---|---|---|---|---|
| | | | Read | Write | |
| GxSEFLAG | Source Event Flag Register, Group x | $X588_H$ | U, PV | U, PV | 16-142 |
| GxCEFLAG | Channel Event Flag Register, Group x | $X580_H$ | U, PV | U, PV | 16-142 |
| GxREFLAG | Result Event Flag Register, Group x | $X584_H$ | U, PV | U, PV | 16-143 |
| GxSEFCLR | Source Event Flag Clear Register, Group x | $X598_H$ | U, PV | U, PV | 16-144 |
| GxCEFCLR | Channel Event Flag Clear Register, Group x | $X590_H$ | U, PV | U, PV | 16-144 |
| GxREFCLR | Result Event Flag Clear Register, Group x | $X594_H$ | U, PV | U, PV | 16-145 |
| GLOBEFLAG | Global Event Flag Register | $00E0_H$ | U, PV | U, PV | 16-145 |
| GxSEVNP | Source Event Node Pointer Register, Group x | $X5C0_H$ | U, PV | U, PV | 16-146 |
| GxCEVNP0 | Channel Event Node Pointer Register 0, Group x | $X5A0_H$ | U, PV | U, PV | 16-147 |
| GxREVNP0 | Result Event Node Pointer Register 0, Group x | $X5B0_H$ | U, PV | U, PV | 16-148 |
| GxREVNP1 | Result Event Node Pointer Register 1, Group x | $X5B4_H$ | U, PV | U, PV | 16-149 |
| GLOBEVNP | Global Event Node Pointer Register | $0140_H$ | U, PV | U, PV | 16-150 |
| GxSRACT | Service Request Software Activation Trigger, Group x | $X5C8_H$ | U, PV | U, PV | 16-152 |
| ID | SHS Module Identification Register | $Z008_H$ | U, SV | U, SV | 16-71 |
| SHSCFG | SHS Configuration Register | $Z040_H$ | U, SV | U, SV | 16-76 |
| CALCTR | SHS Calibration Control Register | $Z0BC_H$ | U, SV | U, SV | 16-126 |
| CALGC0 | SHS Gain Calibration Control Reg. | $Z0C0_H$ | U, SV | U, SV | 16-128 |


Subject to Agreement on the Use of Product Information

**Table 16-8 Registers Overview** (cont'd)

| Register Short Name | Register Long Name | Offset Addr. | Access Mode | | Page Num. |
|---|---|---|---|---|---|
| | | | Read | Write | |
| CALGC1 | SHS Gain Calibration Control Reg. | $Z0C4_H$ | U, SV | U, SV | 16-128 |
| GNCTR00 | Gain Control Register 0, S&H unit 0 | $Z180_H$ | U, SV | U, SV | 16-130 |
| GNCTR10 | Gain Control Register 0, S&H unit 1 | $Z190_H$ | U, SV | U, SV | 16-130 |
| STEPCFG | SHS Stepper Configuration Register | $Z044_H$ | U, SV | U, SV | 16-83 |
| TIMCFG0 | SHS Timing Configuration Register | $Z080_H$ | U, SV | U, SV | 16-84 |
| TIMCFG1 | SHS Timing Configuration Register | $Z084_H$ | U, SV | U, SV | 16-84 |
| LOOP | SHS SD Loop Control Register | $Z050_H$ | U, SV | U, SV | 16-141 |

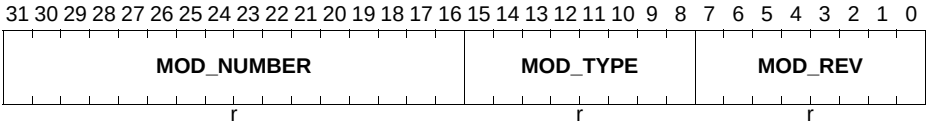
1) Access to these registers is protected by bit protection scheme described in SCU chapter.


Subject to Agreement on the Use of Product Information

### 16.16.1 Module Identification

The module identification register indicates the version of the ADC module that is used in the XMC1300.

**ID**
**Module Identification Register (0008$_H$) Reset Value: 00C5 C0XX$_H$**
**SHS0_ID**
**Module Identification Register (4803 4008$_H$) Reset Value: 0099 C0XX$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **MOD_REV** | [7:0] | r | **Module Revision**<br>Indicates the revision number of the implementation. This information depends on the design step. |
| **MOD_TYPE** | [15:8] | r | **Module Type**<br>This internal marker is fixed to C0$_H$. |
| **MOD_NUMBER** | [31:16] | r | **Module Number**<br>Indicates the module identification number (00C5$_H$ = SARADC, 0099$_H$ = SHS). |


Subject to Agreement on the Use of Product Information

### 16.16.2 System Registers

A set of standardized registers provides general access to the module and controls basic system functions.

The Clock Control Register **CLC** allows the programmer to adapt the functionality and power consumption of the module to the requirements of the application. Register **CLC** controls the module clock signal and the reactivity to the sleep mode signal.

**CLC**
**Clock Control Register** **($0000_H$)** **Reset Value: $0000\ 0003_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **EDIS** | **0** | **DISS** | **DISR** |
| r | r | r | r | r | r | r | r | r | r | r | r | rw | r | r | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **DISR** | 0 | rw | **Module Disable Request Bit**<br>Used for enable/disable control of the module. Also the analog section is disabled by clearing ANONS.<br>$0_B$ On request: enable the module clock<br>$1_B$ Off request: stop the module clock |
| **DISS** | 1 | r | **Module Disable Status Bit**<br>$0_B$ Module clock is enabled<br>$1_B$ Off: module is not clocked |
| **0** | 2 | r | **Reserved, write 0, read as 0** |
| **EDIS** | 3 | rw | **Sleep Mode Enable Control**<br>Used to control module's reaction to sleep mode.<br>$0_B$ Sleep mode request is enabled and functional<br>$1_B$ Module disregards the sleep mode control signal |
| **0** | [31:4] | r | **Reserved, write 0, read as 0** |


Subject to Agreement on the Use of Product Information

The OCDS control and status register OCS controls the module's behavior in suspend mode (used for debugging) and includes the module-related control bits for the OCDS Trigger Bus (OTGB).

The OCDS Control and Status (OCS) register is cleared by Debug Reset.
The OCS register can only be written when the OCDS is enabled.
If OCDS is being disabled, the OCS register value will not change.
When OCDS is disabled the OCS suspend control is ineffective.
Write access is 32 bit wide only and requires Supervisor Mode.

**OCS**

**OCDS Control and Status Register ($0028_H$)** **Reset Value: $0000\ 0000_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **0** | **SUS STA** | **SUS _P** | **SUS** | | | | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| r | r | rh | w | rw | | | | r | r | r | r | r | r | r | r |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **TG _P** | **TGB** | **TGS** | |
| r | r | r | r | r | r | r | r | r | r | r | r | w | rw | rw | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **TGS** | [1:0] | rw | **Trigger Set for OTGB0/1**<br>$00_B$ No Trigger Set output<br>$01_B$ Trigger Set 1: TS16_SSIG, input sample signals<br>$10_B$ Reserved<br>$11_B$ Reserved |
| **TGB** | 2 | rw | **OTGB0/1 Bus Select**<br>$0_B$ Trigger Set is output on OTGB0<br>$1_B$ Trigger Set is output on OTGB1 |
| **TG_P** | 3 | w | **TGS, TGB Write Protection**<br>TGS and TGB are only written when TG_P is 1, otherwise unchanged. Read as 0. |
| **0** | [23:4] | r | **Reserved, write 0, read as 0** |

| Field | Bits | Type | Description |
|---|---|---|---|
| **SUS** | [27:24] | rw | **OCDS Suspend Control**<br>Controls the sensitivity to the suspend signal coming from the OCDS Trigger Switch (OTGS)<br>$0000_B$ Will not suspend<br>$0001_B$ Hard suspend: Clock is switched off immediately.<br>$0010_B$ Soft suspend mode 0:<br>Stop conversions after the currently running one is completed and its result has been stored.<br>No change for the arbiter.<br>$0011_B$ Soft suspend mode 1:<br>Stop conversions after the currently running one is completed and its result has been stored.<br>Stop arbiter after the current arbitration round.<br>others: Reserved |
| **SUS_P** | 28 | w | **SUS Write Protection**<br>SUS is only written when SUS_P is 1, otherwise unchanged. Read as 0. |
| **SUSSTA** | 29 | rh | **Suspend State**<br>$0_B$ Module is not (yet) suspended<br>$1_B$ Module is suspended |
| **0** | [31:30] | r | **Reserved, write 0, read as 0** |

**Table 16-9 TS16_SSIG Trigger Set VADC**

| Bits | Name | Description |
|---|---|---|
| [1:0] | GxSAMPLE | Input signal sample phase of converter group x (x = 1-0) |
| [15:2] | 0 | Reserved |


Subject to Agreement on the Use of Product Information

### 16.16.3 General Registers

The global configuration register provides global control and configuration options that are valid for all converters of the cluster.

**GLOBCFG**
**Global Configuration Register ($0080_H$) Reset Value: $0000\ 000F_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **SUCAL** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **DPCAL1** | **DPCAL0** |
| w | r | r | r | r | r | r | r | r | r | r | r | r | r | rw | rw |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **DIVWC** | **0** | **0** | **0** | **0** | **0** | **DIVD** | | **DCMSB** | **0** | **0** | **DIVA** | | | | |
| w | r | r | r | r | r | rw | | rw | r | r | rw | | | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **DIVA** | [4:0] | rw | **Divider Factor for the Analog Internal Clock**<br>Defines the frequency of the basic converter clock $f_{ADCI}$ (base clock for conversion and sample phase).<br>$00_H$ $f_{ADCI} = f_{ADC} / 2^{1)}$<br>$01_H$ $f_{ADCI} = f_{ADC} / 2$<br>$02_H$ $f_{ADCI} = f_{ADC} / 3$<br>...<br>$1F_H$ $f_{ADCI} = f_{ADC} / 32$ |
| **0** | [6:5] | r | **Reserved, write 0, read as 0** |
| **DCMSB** | 7 | rw | **Double Clock for the MSB Conversion**<br>Selects an additional clock cycle for the conversion step of the MSB.<br>$0_B$ 1 clock cycle for the MSB (standard)<br>$1_B$ Reserved |
| **DIVD** | [9:8] | rw | **Divider Factor for the Arbiter Clock**<br>Defines the frequency of the arbiter clock $f_{ADCD}$.<br>$00_B$ $f_{ADCD} = f_{ADC}$<br>$01_B$ $f_{ADCD} = f_{ADC} / 2$<br>$10_B$ $f_{ADCD} = f_{ADC} / 3$<br>$11_B$ $f_{ADCD} = f_{ADC} / 4$ |
| **0** | [14:10] | r | **Reserved, write 0, read as 0** |

| Field | Bits | Type | Description |
|---|---|---|---|
| **DIVWC** | 15 | w | **Write Control for Divider Parameters**<br>$0_B$ No write access to divider parameters<br>$1_B$ Bitfields DIVA, DCMSB, DIVD can be written |
| **DPCALx (x = 0 - 1)** | x+16 | rw | **Disable Post-Calibration**<br>$0_B$ Automatic post-calibration after each conversion of group x<br>$1_B$ No post-calibration |
| **0** | [30:18] | r | **Reserved, write 0, read as 0** |
| **SUCAL** | 31 | w | **Start-Up Calibration**<br>The 0-1 transition of bit SUCAL initiates the start-up calibration phase of all calibrated analog converters.<br>$0_B$ No action<br>$1_B$ Initiate the start-up calibration phase (indication in bit GxARBCFG.CAL) |

1) For valid seetings of DIVA, please see **"Conversion Timing" on Page 16-41**.

**SHS0_SHSCFG**
**SHS Configuration Register (4803 4040$_H$) Reset Value: 0000 0000$_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **STATE** | | | | **TC** | | | | **0** | **0** | **0** | **0** | **0** | **0** | **SP1** | **SP0** |
| rh | | | | rw | | | | r | r | r | r | r | r | rh | rh |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **SCWC** | **ANRDY** | **0** | **ANOFF** | **AREF** | | **0** | **0** | **0** | **0** | **0** | **0** | **DIVS** | | | |
| w | rh | r | rw | rw | | r | r | r | r | r | r | rw | | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **DIVS** | [3:0] | rw | **Divider Factor for the SHS Clock**<br>Defines the frequency of the clock for the SHS logic and the SAR converter.<br>$0000_B$ $f_{SH} = f_{CONV} / 1$<br>$0001_B$ $f_{SH} = f_{CONV} / 2$<br>...<br>$1111_B$ $f_{SH} = f_{CONV} / 16$ |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | [9:4] | r | **Reserved, write 0, read as 0** |
| **AREF** | [11:10] | rw | **Analog Reference Voltage Selection**<br>$00_B$ External reference, upper supply range<br>$01_B$ Reserved<br>$10_B$ Internal reference, upper supply range<br>$11_B$ Internal reference, lower supply range |
| **ANOFF** | 12 | rw | **Analog Converter Power Down Force**[1)]<br>$0_B$ Converter controlled by bitfields ANONS (digital control block)<br>$1_B$ Converter is permanently off |
| **0** | 13 | r | **Reserved, write 0, read as 0** |
| **ANRDY** | 14 | rh | **Analog Converter Ready**<br>$0_B$ Converter is in power-down mode<br>$1_B$ Converter is operable |
| **SCWC** | 15 | w | **Write Control for SHS Configuration**<br>$0_B$ No write access to SHS configuration<br>$1_B$ Bitfields ANOFF, AREF, DIVS can be written |
| **SP0, SP1** | 16, 17 | rh | **Sample Pending on S&H Unit x**<br>$0_B$ No sample pending<br>$1_B$ S&H unit x has finished the sample phase |
| **0** | [23:18] | r | **Reserved, write 0, read as 0** |
| **TC** | [27:24] | rw | **Test Control**<br>$1011_B$ Internal test functions enabled<br>Others: Normal operation<br>*Note: Make sure to write $0000_B$ to this bitfield for normal operation.* |
| **STATE** | [31:28] | rh | **Current State of Sequencer**<br>$0000_B$ Idle<br>$0001_B$ Offset calibration active<br>$0010_B$ Gain calibration active<br>$0011_B$ Startup calibration active<br>$1000_B$ Stepper process active for S&H unit 0<br>$1001_B$ Stepper process active for S&H unit 1<br>Others: Normal operation |

1) See also **"Analog Converter Control" on Page 16-14**.


Subject to Agreement on the Use of Product Information

### Register Access Control

Several protection schemes are provided to prevent unintended write access to control bitfields of the VADC.

**ACCPROT0**
**Access Protection Register** **($0088_H$)** **Reset Value: $0000\ 0000_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **APGC** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **APPI1** | **APPI0** |
| rw | r | r | r | r | r | r | r | r | r | r | r | r | r | rw | rw |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **APEM** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **APPC1** | **APPC0** |
| rw | r | r | r | r | r | r | r | r | r | r | r | r | r | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **APC0, APC1** | 0, 1 | rw | **Access Protection Channel Control, Group 0 - 1**<br>$0_B$ Full access to registers<br>$1_B$ Write access to channel control registers is blocked |
| **0** | [14:2] | r | **Reserved, write 0, read as 0** |
| **APEM** | 15 | rw | **Access Protection External Multiplexer**<br>$0_B$ Full access to registers<br>$1_B$ Write access to external multiplexer registers is blocked |
| **APPI0, APPI1** | 16, 17 | rw | **Access Protection Initialization, Group 0 - 1**<br>$0_B$ Full access to registers<br>$1_B$ Write access to initialization registers is blocked |
| **0** | [30:18] | r | **Reserved, write 0, read as 0** |
| **APGC** | 31 | rw | **Access Protection Global Configuration**<br>$0_B$ Full access to register<br>$1_B$ Write access to global configuration register is blocked |


Subject to Agreement on the Use of Product Information

**ACCPROT1**
**Access Protection Register (008C$_H$) Reset Value: 0000 0000$_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **APR1** | **APR0** |
| r | r | r | r | r | r | r | r | r | r | r | r | r | r | rw | rw |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **APTF** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **APS1** | **APS0** |
| rw | r | r | r | r | r | r | r | r | r | r | r | r | r | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **APS0, APS1** | 0, 1 | rw | **Access Protection Service Request, Group 0 - 1**<br>$0_B$ Full access to registers<br>$1_B$ Write access to service request registers is blocked |
| **0** | [14:2] | r | **Reserved, write 0, read as 0** |
| **APTF** | 15 | rw | **Access Protection Test Function**<br>$0_B$ Full access to register<br>$1_B$ Write access to test function register is blocked |
| **APR0, APR1** | 16, 17 | rw | **Access Protection Result Registers, Group 0 - 1**<br>$0_B$ Full access to registers<br>$1_B$ Write access to result registers is blocked |
| **0** | [31:18] | r | **Reserved, write 0, read as 0** |

**Table 16-10 Register Protection Groups**

| Control Bits | Registers | Notes |
|---|---|---|
| APCx | GxCHCTR0 … GxCHCTRy | Channel control |
| APEM | EMUXSEL, GxEMUXCTR | External multiplexer control |
| APIx | GxARBCFG, GxARBPR, GxCHASS, GxRRASS, GxICLASS0/1, GxSYNCTR | Initialization |
| APGC | GLOBCFG | Gobal configuration |


Subject to Agreement on the Use of Product Information

**Table 16-10 Register Protection Groups** (cont'd)

| Control Bits | Registers | Notes |
|---|---|---|
| APSx | GxSEFLAG, GxSEVNP, GxCEFLAG, GxCEVNP0/1, GxREFLAG, GxREVNP0/1, GxSRACT | Service request control |
| APTF | - | Test functions |
| APRx | GxRCR0 … GxRCR15, GxBOUND, GxRES0 … GxRES15 | Result control |
| | | |

### Priority Channel Assignment

Each channel of a group can be assigned to this group's request sources and is then regarded as a priority channel. An assigned priority channel can only be converted by its own group's request sources. A not assigned channel can also be converted by the background request source.

**GxCHASS (x = 0 - 1)**
**Channel Assignment Register, Group x**
**($x * 0400_H + 0488_H$)** **Reset Value: $0000\ 0000_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **ASSCH7** | **ASSCH6** | **ASSCH5** | **ASSCH4** | **ASSCH3** | **ASSCH2** | **ASSCH1** | **ASSCH0** |
| r | r | r | r | r | r | r | r | rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **ASSCHy (y = 0 - 7)** | y | rw | **Assignment for Channel y**<br>$0_B$ Channel y can be a background channel converted with lowest priority<br>$1_B$ Channel y is a priority channel within group x |
| **0** | [31:8] | r | **Reserved, write 0, read as 0** |


Subject to Agreement on the Use of Product Information

### Priority Result Register Assignment

Each result register of a group can be assigned to this group's request sources and is then regarded as a reserved resource. 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.

**GxRRASS (x = 0 - 1)**
**Result Assignment Register, Group x**

**(x * $0400_H$ + $048C_H$)** **Reset Value: $0000\ 0000_H$**

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

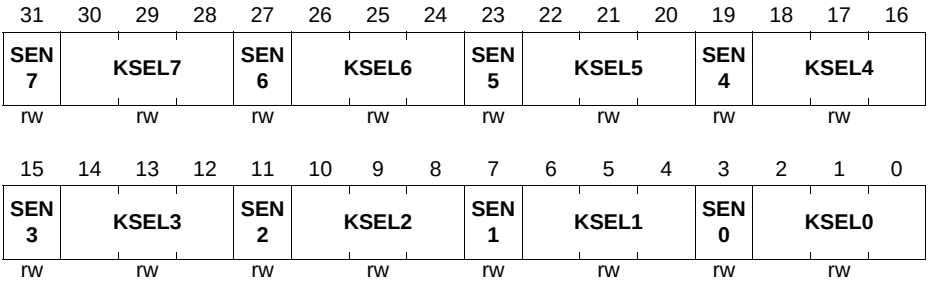
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **ASSRR15** | **ASSRR14** | **ASSRR13** | **ASSRR12** | **ASSRR11** | **ASSRR10** | **ASSRR9** | **ASSRR8** | **ASSRR7** | **ASSRR6** | **ASSRR5** | **ASSRR4** | **ASSRR3** | **ASSRR2** | **ASSRR1** | **ASSRR0** |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **ASSRRy (y = 0 - 15)** | y | rw | **Assignment for Result Register y**<br>$0_B$ Result register y can also be written by the background source<br>$1_B$ Result register y can only be written by sources within group x |
| **0** | [31:16] | r | **Reserved, write 0, read as 0** |


Subject to Agreement on the Use of Product Information

The stepper configuration register selects the S&H unit that is serviced during each step of the stepper state machine.

**SHS0_STEPCFG**
**Stepper Configuration Register (4803 4044$_H$)** **Reset Value: 0000 0098$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **KSEL0,**<br>**KSEL1,**<br>**KSEL2,**<br>**KSEL3,**<br>**KSEL4,**<br>**KSEL5,**<br>**KSEL6,**<br>**KSEL7** | [2:0],<br>[6:4],<br>[10:8],<br>[14:12],<br>[18:16],<br>[22:20],<br>[26:24],<br>[30:28] | rw | **Kernel Select**<br>Defines the group (i.e. the S&H unit) that is served during this step.<br>The valid values depend on the number of supported kernels/interfaces: 0 … 1 |
| **SEN0,**<br>**SEN1,**<br>**SEN2,**<br>**SEN3,**<br>**SEN4,**<br>**SEN5,**<br>**SEN6,**<br>**SEN7** | 3,<br>7,<br>11,<br>15,<br>19,<br>23,<br>27,<br>31 | rw | **Step x Enable**<br>Defines if or not this step is executed.<br>$0_B$ Off:<br>This step is not part of the stepper sequence<br>$1_B$ Active:<br>This step is executed during the sequence |


Subject to Agreement on the Use of Product Information

The timing configuration register TIMCFGx configures the interface timing of the SHS.

**SHS0_TIMCFGx (x = 0 - 1)**
**Timing Configuration Register x**

**(4803 4080$_H$ + x * 4$_H$)** **Reset Value: 0000 0001$_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **0** | **TGEN** | | | | | | | | | | | | | |
| r | r | rh | | | | | | | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **0** | **SST** | | | | | | **FCRT** | | | | **0** | **0** | **0** | **AT** |
| r | r | rw | | | | | | rw | | | | r | r | r | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **AT** | 0 | rw | **Accelerated Timing**<br>Enables the enhanced conversion timing scheme.<br>$0_B$ Compatible timing:<br>Result available after standard conversion time<br>$1_B$ Accelerated timing:<br>Result available as soon as converted |
| **0** | [3:1] | r | **Reserved, write 0, read as 0** |
| **FCRT** | [7:4] | rw | **Fast Compare Mode Response Time**<br>Defines the duration of a fast compare operation. A result is generated after (FCRT+1) × 2 × $t_{ADCI}$.<br>$0_H$ Result after $t_{ADCI}$ × 2<br>...<br>$F_H$ Result after $t_{ADCI}$ × 32 |
| **SST** | [13:8] | rw | **Short Sample Time**<br>Defines the duration of the sample phase.<br>$00_H$ Compatible timing:<br>Sample time is defined by DIVA and STC.<br>$01_H$ Sample time is $t_{ADC}$ × 1<br>$3F_H$ Sample time is $t_{ADC}$ × 63 |
| **0** | [15:14] | r | **Reserved, write 0, read as 0** |

| Field | Bits | Type | Description |
|---|---|---|---|
| **TGEN** | [29:16] | rh | **Timing Generator**<br>Counts the duration of the sample phase, and the conversion in compatible timing mode. |
| **0** | [31:30] | r | **Reserved, write 0, read as 0** |

### 16.16.4 Arbitration and Source Registers

The Arbitration Configuration Register selects the timing and the behavior of the arbiter.

**GxARBCFG (x = 0 - 1)**
**Arbitration Configuration Register, Group x**
**(x * $0400_H$ + $0480_H$)** **Reset Value: $0000\ 0000_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SAMPLE | BUSY | CALS | CAL | 0 | 0 | SYNRUN | CHNR | | | | | CSRC | | ANONS | |
| rh | rh | rh | rh | r | r | rh | rh | | | | | rh | | rh | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ARBM | 0 | ARBRND | | 0 | 0 | ANONC | |
| r | r | r | r | r | r | r | r | rw | r | rw | | r | r | rw | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **ANONC** | [1:0] | rw | **Analog Converter Control**<br>Defines the value of bitfield ANONS in a stand-alone converter or a converter in master mode.<br>Coding see ANONS or **Section 16.5**. |
| **0** | [3:2] | r | **Reserved, write 0, read as 0** |
| **ARBRND** | [5:4] | rw | **Arbitration Round Length**<br>Defines the number of arbitration slots per arb. round (arbitration round length = $t_{ARB}$).[1)]<br>$00_B$ 4 arbitration slots per round ($t_{ARB}$ = 4 / $f_{ADCD}$)<br>$01_B$ 8 arbitration slots per round ($t_{ARB}$ = 8 / $f_{ADCD}$)<br>$10_B$ 16 arbitration slots per round ($t_{ARB}$ = 16 / $f_{ADCD}$)<br>$11_B$ 20 arbitration slots per round ($t_{ARB}$ = 20 / $f_{ADCD}$) |
| **0** | 6 | r | **Reserved, write 0, read as 0** |

| Field | Bits | Type | Description |
|---|---|---|---|
| **ARBM** | 7 | rw | **Arbitration Mode**<br>$0_B$ The arbiter runs permanently. This setting is required for a synchronization slave (see **Section 16.12.1**) and for equidistant sampling using the signal ARBCNT (see **Section 16.12.2**).<br>$1_B$ The arbiter only runs if at least one conversion request of an enabled request source is pending. This setting ensures a reproducible latency from an incoming request to the conversion start, if the converter is idle. Synchronized conversions are not supported. |
| **0** | [15:8] | r | **Reserved, write 0, read as 0** |
| **ANONS** | [17:16] | rh | **Analog Converter Control Status**<br>Defined by bitfield ANONC in a stand-alone kernel or a kernel in master mode.<br>In slave mode, this bitfield is defined by bitfield ANONC of the respective master kernel.<br>See also **Section 16.5**.<br>$00_B$ Analog converter off<br>$01_B$ Reserved<br>$10_B$ Slow standby mode<br>$11_B$ Normal operation (permanently on) |
| **CSRC** | [19:18] | rh | **Currently Converted Request Source**<br>Indicates the arbitration slot number of the current (BUSY = 1) or of the last (BUSY = 0) conversion.<br>This bitfield is updated when a conversion is started.<br>$00_B$ Current/last conversion for request source 0<br>$01_B$ Current/last conversion for request source 1<br>$10_B$ Current/last conversion for background source<br>$11_B$ Current/last conversion for synchronization request (slave converter)<br>Other combinations are reserved. |
| **CHNR** | [24:20] | rh | **Channel Number**<br>Indicates the current or last converted analog input channel.<br>This bitfield is updated when a conversion is started. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **SYNRUN** | 25 | rh | **Synchronous Conversion Running**<br>Indicates that a synchronized (= parallel) conversion is currently running.<br>$0_B$ Normal conversion or no conversion running<br>$1_B$ A synchronized conversion is running (cannot be cancelled by higher priority requests!) |
| **0** | [27:26] | r | **Reserved, write 0, read as 0** |
| **CAL** | 28 | rh | **Start-Up Calibration Active Indication**<br>Indicates the start-up calibration phase of the corresponding analog converter.<br>$0_B$ Completed or not yet started<br>$1_B$ Start-up calibration phase is active (set one clock cycle after setting bit SUCAL)<br>*Note: Start conversions only after the start-up calibration phase is complete.*[2)] |
| **CALS** | 29 | rh | **Start-Up Calibration Started**<br>Indicates that the start-up calibration has begun.<br>$0_B$ Requested but not yet started<br>$1_B$ Start-up calibration has begun<br>*Note: Bit CALS is cleared when setting bit SUCAL and is set together with bit CAL.* |
| **BUSY** | 30 | rh | **Converter Busy Flag**<br>$0_B$ Not busy<br>$1_B$ Converter is busy with a conversion |
| **SAMPLE** | 31 | rh | **Sample Phase Flag**<br>$0_B$ Converting or idle<br>$1_B$ Input signal is currently sampled |

1) The default setting of 4 arbitration slots is sufficient for correct arbitration. The duration of an arbitration round can be increased if required to synchronize requests.

2) The completion of the start-up calibration is indicated by CAL = 0 AND CALS = 1.
CAL = CALS = 0 indicates that the start-up calibration was requested but has not yet begun.
CAL = CALS = 1 indicates an ongoing calibration phase.

The Arbitration Priority Register defines the request source priority and the conversion start mode for each request source.

*Note: Only change priority and conversion start mode settings of a request source while this request source is disabled, and a currently running conversion requested by this source is finished.*


Subject to Agreement on the Use of Product Information

**GxARBPR (x = 0 - 1)**
**Arbitration Priority Register, Group x**

**(x * $0400_H$ + $0484_H$)** **Reset Value: $0000\ 0000_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | ASEN2 | ASEN1 | ASEN0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| r | r | r | r | r | rw | rw | rw | r | r | r | r | r | r | r | r |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | CSM2 | 0 | PRIO2 | | CSM1 | 0 | PRIO1 | | CSM0 | 0 | PRIO0 | |
| r | r | r | r | rw | r | rw | | rw | r | rw | | rw | r | rw | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **PRIO0, PRIO1, PRIO2** | [1:0], [5:4], [9:8] | rw | **Priority of Request Source x**<br>Arbitration priority of request source x (in slot x)<br>$00_B$ Lowest priority is selected.<br>...<br>$11_B$ Highest priority is selected. |
| **CSM0, CSM1, CSM2** | 3, 7, 11 | rw | **Conversion Start Mode of Request Source x**<br>$0_B$ Wait-for-start mode<br>$1_B$ Cancel-inject-repeat mode, i.e. this source can cancel conversion of other sources. |
| **0** | 2, 6, 10, [23:12] | r | **Reserved, write 0, read as 0** |
| **ASENy (y = 0 - 2)** | 24 + y | rw | **Arbitration Slot y Enable**<br>Enables the associated arbitration slot of an arbiter round. The request source bits are not modified by write actions to ASENR.<br>$0_B$ The corresponding arbitration slot is disabled and considered as empty. Pending conversion requests from the associated request source are disregarded.<br>$1_B$ The corresponding arbitration slot is enabled. Pending conversion requests from the associated request source are arbitrated. |
| **0** | [31:27] | r | **Reserved, write 0, read as 0** |

The control register of the queue source selects the external gate and/or trigger signals. Write control bits allow separate control of each function with a simple write access.

**GxQCTRL0 (x = 0 - 1)**
**Queue 0 Source Control Register, Group x**
**($x * 0400_H + 0500_H$)** **Reset Value: $0000\ 0000_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **TM WC** | **0** | **0** | **TM EN** | **0** | **0** | **0** | **0** | **GT WC** | **0** | **0** | **GT LVL** | **GT SEL** | | | |
| w | r | r | rw | r | r | r | r | w | r | r | rh | rw | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **XT WC** | **XT MODE** | | **XT LVL** | **XT SEL** | | | | **0** | **0** | **0** | **0** | **SRCRESREG** | | | |
| w | rw | | rh | rw | | | | r | r | r | r | rwr | | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **SRCRESREG** | [3:0] | rw | **Source-specific Result Register**<br>$0000_B$ Use GxCHCTRy.RESREG to select a group result register<br>$0001_B$ Store result in group result register GxRES1<br>…<br>$1111_B$ Store result in group result register GxRES15 |
| **0** | [7:4] | r | **Reserved, write 0, read as 0** |
| **XTSEL** | [11:8] | rw | **External Trigger Input Selection**<br>The connected trigger input signals are listed in **Table 16-15 “Digital Connections in the XMC1300” on Page 16-156**<br>*Note: XTSEL = $1111_B$ uses the selected gate input as trigger source (ENGT must be $0x_B$).* |
| **XTLVL** | 12 | rh | **External Trigger Level**<br>Current level of the selected trigger input |
| **XTMODE** | [14:13] | rw | **Trigger Operating Mode**<br>$00_B$ No external trigger<br>$01_B$ Trigger event upon a falling edge<br>$10_B$ Trigger event upon a rising edge<br>$11_B$ Trigger event upon any edge |

| Field | Bits | Type | Description |
|---|---|---|---|
| **XTWC** | 15 | w | **Write Control for Trigger Configuration**<br>$0_B$ No write access to trigger configuration<br>$1_B$ Bitfields XTMODE and XTSEL can be written |
| **GTSEL** | [19:16] | rw | **Gate Input Selection**<br>The connected gate input signals are listed in **Table 16-15 “Digital Connections in the XMC1300” on Page 16-156** |
| **GTLVL** | 20 | rh | **Gate Input Level**<br>Current level of the selected gate input |
| **0** | [22:21] | r | **Reserved, write 0, read as 0** |
| **GTWC** | 23 | w | **Write Control for Gate Configuration**<br>$0_B$ No write access to gate configuration<br>$1_B$ Bitfield GTSEL can be written |
| **0** | [27:24] | r | **Reserved, write 0, read as 0** |
| **TMEN** | 28 | rw | **Timer Mode Enable**<br>$0_B$ No timer mode: standard gating mechanism can be used<br>$1_B$ Timer mode for equidistant sampling enabled: standard gating mechanism must be disabled |
| **0** | [30:29] | r | **Reserved, write 0, read as 0** |
| **TMWC** | 31 | w | **Write Control for Timer Mode**<br>$0_B$ No write access to timer mode<br>$1_B$ Bitfield TMEN can be written |


Subject to Agreement on the Use of Product Information

The Queue Mode Register configures the operating mode of a queued request source.

**GxQMR0 (x = 0 - 1)**
**Queue 0 Mode Register, Group x**

**($x * 0400_H + 0504_H$)** **Reset Value: $0000\ 0000_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **RPT DIS** |
| r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | rw |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **0** | **0** | **0** | **CEV** | **FLU SH** | **TR EV** | **CLR V** | **0** | **0** | **0** | **0** | **0** | **EN TR** | **ENGT** | |
| r | r | r | r | w | w | w | w | r | r | r | r | r | rw | rw | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **ENGT** | [1:0] | rw | **Enable Gate**<br>Selects the gating functionality for source 0/2.<br>$00_B$ No conversion requests are issued<br>$01_B$ Conversion requests are issued if a valid conversion request is pending in the queue 0 register or in the backup register<br>$10_B$ Conversion requests are issued if a valid conversion request is pending in the queue 0 register or in the backup register and REQGTx = 1<br>$11_B$ Conversion requests are issued if a valid conversion request is pending in the queue 0 register or in the backup register and REQGTx = 0<br>*Note: REQGTx is the selected gating signal.* |
| **ENTR** | 2 | rw | **Enable External Trigger**<br>$0_B$ External trigger disabled<br>$1_B$ The selected edge at the selected trigger input signal REQTR generates the trigger event |
| **0** | [7:3] | r | **Reserved, write 0, read as 0** |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **CLRV** | 8 | w | **Clear Valid Bit**<br>$0_B$ No action<br>$1_B$ The next pending valid queue entry in the sequence and the event flag EV are cleared. If there is a valid entry in the queue backup register (QBUR.V = 1), this entry is cleared, otherwise the entry in queue register 0 is cleared. |
| **TREV** | 9 | w | **Trigger Event**<br>$0_B$ No action<br>$1_B$ Generate a trigger event by software |
| **FLUSH** | 10 | w | **Flush Queue**<br>$0_B$ No action<br>$1_B$ Clear all queue entries (including backup stage) and the event flag EV. The queue contains no more valid entry. |
| **CEV** | 11 | w | **Clear Event Flag**<br>$0_B$ No action<br>$1_B$ Clear bit EV |
| **0** | [15:12] | r | **Reserved, write 0, read as 0** |
| **RPTDIS** | 16 | rw | **Repeat Disable**<br>$0_B$ A cancelled conversion is repeated<br>$1_B$ A cancelled conversion is discarded |
| **0** | [31:17] | r | **Reserved, write 0, read as 0** |


Subject to Agreement on the Use of Product Information

The Queue Status Register indicates the current status of the queued source. The filling level and the empty information refer to the queue intermediate stages (if available) and to the queue register 0. An aborted conversion stored in the backup stage is not indicated by these bits (therefore, see QBURx.V).

**GxQSR0 (x = 0 - 1)**
**Queue 0 Status Register, Group x**

**(x * $0400_H$ + $0508_H$)** **Reset Value: 0000 $0020_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **0** | **0** | **0** | **0** | **0** | **0** | **EV** | **REQ GT** | **0** | **EMP TY** | **0** | **FILL** | | | |
| r | r | r | r | r | r | r | rh | rh | r | rh | r | rh | | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **FILL** | [3:0] | rh | **Filling Level for Queue 2**<br>Indicates the number of valid queue entries.<br>It is incremented each time a new entry is written to QINRx or by an enabled refill mechanism.<br>It is decremented each time a requested conversion has been started. A new entry is ignored if the filling level has reached its maximum value.<br>$0000_B$ There is 1 ( if EMPTY = 0) or no (if EMPTY = 1) valid entry in the queue<br>$0001_B$ There are 2 valid entries in the queue<br>$0010_B$ There are 3 valid entries in the queue<br>…<br>$0111_B$ There are 8 valid entries in the queue<br>others: Reserved |
| **0** | 4 | r | **Reserved, write 0, read as 0** |
| **EMPTY** | 5 | rh | **Queue Empty**<br>$0_B$ There are valid entries in the queue (see FILL)<br>$1_B$ No valid entries (queue is empty) |
| **0** | 6 | r | **Reserved, write 0, read as 0** |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **REQGT** | 7 | rh | **Request Gate Level**<br>Monitors the level at the selected REQGT input.<br>$0_B$ The gate input is low<br>$1_B$ The gate input is high |
| **EV** | 8 | rh | **Event Detected**<br>Indicates that an event has been detected while at least one valid entry has been in the queue (queue register 0 or backup stage). Once set, this bit is cleared automatically when the requested conversion is started.<br>$0_B$ No trigger event<br>$1_B$ A trigger event has been detected |
| **0** | [31:9] | r | **Reserved, write 0, read as 0** |


Subject to Agreement on the Use of Product Information

The Queue Input Register is the entry point for conversion requests of a queued request source.

**GxQINR0 (x = 0 - 1)**
**Queue 0 Input Register, Group x**

**($x * 0400_H + 0510_H$)** **Reset Value: $0000\ 0000_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **EXTR** | **ENSI** | **RF** | **REQCHNR** | | | | |
| r | r | r | r | r | r | r | r | w | w | w | | | w | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **REQCHNR** | [4:0] | w | **Request Channel Number**<br>Defines the channel number to be converted |
| **RF** | 5 | w | **Refill**<br>$0_B$ No refill: this queue entry is converted once and then invalidated<br>$1_B$ Automatic refill: this queue entry is automatically reloaded into QINRx when the related conversion is started |
| **ENSI** | 6 | w | **Enable Source Interrupt**<br>$0_B$ No request source interrupt<br>$1_B$ A request source event interrupt is generated upon a request source event (related conversion is finished) |
| **EXTR** | 7 | w | **External Trigger**<br>Enables the external trigger functionality.<br>$0_B$ A valid queue entry immediately leads to a conversion request.<br>$1_B$ A valid queue entry waits for a trigger event to occur before issuing a conversion request. |
| **0** | [31:8] | r | **Reserved, write 0, read as 0** |

*Note: Registers QINRx share addresses with registers QBURx.*
*Write operations target the control bits in register QINRx. Read operations return the status bits from register QBURx.*

The queue registers 0 monitor the status of the pending request (queue stage 0).

**GxQ0R0 (x = 0 - 1)**
**Queue 0 Register 0, Group x (x * $0400_H$ + $050C_H$) Reset Value: 0000 $0000_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **0** | **0** | **0** | **0** | **0** | **0** | **V** | **EXTR** | **ENSI** | **RF** | **REQCHNR** | | | | |
| r | r | r | r | r | r | r | rh | rh | rh | rh | rh | | | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **REQCHNR** | [4:0] | rh | **Request Channel Number**<br>Stores the channel number to be converted. |
| **RF** | 5 | rh | **Refill**<br>Selects the handling of handled requests.<br>$0_B$ The request is discarded after the conversion start.<br>$1_B$ The request is automatically refilled into the queue after the conversion start. |
| **ENSI** | 6 | rh | **Enable Source Interrupt**<br>$0_B$ No request source interrupt<br>$1_B$ A request source event interrupt is generated upon a request source event (related conversion is finished) |
| **EXTR** | 7 | rh | **External Trigger**<br>Enables external trigger events.<br>$0_B$ A valid queue entry immediately leads to a conversion request<br>$1_B$ The request handler waits for a trigger event |
| **V** | 8 | rh | **Request Channel Number Valid**<br>Indicates a valid queue entry in queue register 0.<br>$0_B$ No valid queue entry<br>$1_B$ The queue entry is valid and leads to a conversion request |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | [31:9] | r | **Reserved, write 0, read as 0** |

The Queue Backup Registers monitor the status of an aborted queued request.

**GxQBUR0 (x = 0 - 1)**
**Queue 0 Backup Register, Group x**

**($x * 0400_H + 0510_H$)** **Reset Value: $0000\ 0000_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **0** | **0** | **0** | **0** | **0** | **0** | **V** | **EXTR** | **ENSI** | **RF** | **REQCHNR** | | | | |
| r | r | r | r | r | r | r | rh | rh | rh | rh | rh | | | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **REQCHNR** | [4:0] | rh | **Request Channel Number**<br>The channel number of the aborted conversion that has been requested by this request source |
| **RF** | 5 | rh | **Refill**<br>The refill control bit of the aborted conversion |
| **ENSI** | 6 | rh | **Enable Source Interrupt**<br>The enable source interrupt control bit of the aborted conversion |
| **EXTR** | 7 | rh | **External Trigger**<br>The external trigger control bit of the aborted conversion |
| **V** | 8 | rh | **Request Channel Number Valid**<br>Indicates if the entry (REQCHNR, RF, TR, ENSI) in the queue backup register is valid.<br>Bit V is set when a running conversion (that has been requested by this request source) is aborted, it is cleared when the aborted conversion is restarted.<br>$0_B$ Backup register not valid<br>$1_B$ Backup register contains a valid entry. This will be requested before a valid entry in queue register 0 (stage 0) will be requested. |
| **0** | [31:9] | r | **Reserved, write 0, read as 0** |


Subject to Agreement on the Use of Product Information

*Note: Registers QBURx share addresses with registers QINRx.*
*Read operations return the status bits from register QBURx. Write operations target the control bits in register QINRx.*

### Registers of Group Scan Source

There is a separate register set for each group scan source. These sources can be operated independently.

The control register of the autoscan source selects the external gate and/or trigger signals.
Write control bits allow separate control of each function with a simple write access.

**GxASCTRL (x = 0 - 1)**
**Autoscan Source Control Register, Group x**
**(x * $0400_H$ + $0520_H$)** **Reset Value: $0000\ 0000_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **TM WC** | **0** | **0** | **TM EN** | **0** | **0** | **0** | **0** | **GT WC** | **0** | **0** | **GT LVL** | **GT SEL** | | | |
| w | r | r | rw | r | r | r | r | w | r | r | rh | rw | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **XT WC** | **XT MODE** | | **XT LVL** | **XT SEL** | | | | **0** | **0** | **0** | **0** | **SRCRESREG** | | | |
| w | rw | | rh | rw | | | | r | r | r | r | rwr | | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **SRCRESREG** | [3:0] | rw | **Source-specific Result Register**<br>$0000_B$ Use GxCHCTRy.RESREG to select a group result register<br>$0001_B$ Store result in group result register GxRES1<br>…<br>$1111_B$ Store result in group result register GxRES15 |
| **0** | [7:4] | r | **Reserved, write 0, read as 0** |
| **XTSEL** | [11:8] | rw | **External Trigger Input Selection**<br>The connected trigger input signals are listed in **Table 16-15 “Digital Connections in the XMC1300” on Page 16-156**<br>*Note: XTSEL = $1111_B$ uses the selected gate input as trigger source (ENGT must be $0x_B$).* |
| **XTLVL** | 12 | rh | **External Trigger Level**<br>Current level of the selected trigger input |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **XTMODE** | [14:13] | rw | **Trigger Operating Mode**<br>$00_B$ No external trigger<br>$01_B$ Trigger event upon a falling edge<br>$10_B$ Trigger event upon a rising edge<br>$11_B$ Trigger event upon any edge |
| **XTWC** | 15 | w | **Write Control for Trigger Configuration**<br>$0_B$ No write access to trigger configuration<br>$1_B$ Bitfields XTMODE and XTSEL can be written |
| **GTSEL** | [19:16] | rw | **Gate Input Selection**<br>The connected gate input signals are listed in **Table 16-15 “Digital Connections in the XMC1300” on Page 16-156** |
| **GTLVL** | 20 | rh | **Gate Input Level**<br>Current level of the selected gate input |
| **0** | [22:21] | r | **Reserved, write 0, read as 0** |
| **GTWC** | 23 | w | **Write Control for Gate Configuration**<br>$0_B$ No write access to gate configuration<br>$1_B$ Bitfield GTSEL can be written |
| **0** | [27:24] | r | **Reserved, write 0, read as 0** |
| **TMEN** | 28 | rw | **Timer Mode Enable**<br>$0_B$ No timer mode:<br>standard gating mechanism can be used<br>$1_B$ Timer mode for equidistant sampling enabled:<br>standard gating mechanism must be disabled |
| **0** | [30:29] | r | **Reserved, write 0, read as 0** |
| **TMWC** | 31 | w | **Write Control for Timer Mode**<br>$0_B$ No write access to timer mode<br>$1_B$ Bitfield TMEN can be written |

The Conversion Request Mode Register configures the operating mode of the channel scan request source.

**GxASMR (x = 0 - 1)**
**Autoscan Source Mode Register, Group x**
**($x * 0400_H + 0524_H$)** **Reset Value: $0000\ 0000_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | LDEV | CLRPND | REQGT | 0 | LDM | SCAN | ENSI | ENTR | | |
| r | r | r | r | r | r | w | w | rh | r | rw | rw | rw | rw | rw | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **ENGT** | [1:0] | rw | **Enable Gate**<br>Selects the gating functionality for source 1.<br>$00_B$ No conversion requests are issued<br>$01_B$ Conversion requests are issued if at least one pending bit is set<br>$10_B$ Conversion requests are issued if at least one pending bit is set and REQGTx = 1.<br>$11_B$ Conversion requests are issued if at least one pending bit is set and REQGTx = 0.<br>*Note: REQGTx is the selected gating signal.* |
| **ENTR** | 2 | rw | **Enable External Trigger**<br>$0_B$ External trigger disabled<br>$1_B$ The selected edge at the selected trigger input signal REQTR generates the load event |
| **ENSI** | 3 | rw | **Enable Source Interrupt**<br>$0_B$ No request source interrupt<br>$1_B$ A request source interrupt is generated upon a request source event (last pending conversion is finished) |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **SCAN** | 4 | rw | **Autoscan Enable**<br>$0_B$ No autoscan<br>$1_B$ Autoscan functionality enabled: a request source event automatically generates a load event |
| **LDM** | 5 | rw | **Autoscan Source Load Event Mode**<br>$0_B$ Overwrite mode: Copy all bits from the select registers to the pending registers upon a load event<br>$1_B$ Combine mode: Set all pending bits that are set in the select registers upon a load event (logic OR) |
| **0** | 6 | r | **Reserved, write 0, read as 0** |
| **REQGT** | 7 | rh | **Request Gate Level**<br>Monitors the level at the selected REQGT input.<br>$0_B$ The gate input is low<br>$1_B$ The gate input is high |
| **CLRPND** | 8 | w | **Clear Pending Bits**<br>$0_B$ No action<br>$1_B$ The bits in register GxASPNDx are cleared |
| **LDEV** | 9 | w | **Generate Load Event**<br>$0_B$ No action<br>$1_B$ A load event is generated |
| **0** | [15:10] | r | **Reserved, write 0, read as 0** |
| **RPTDIS** | 16 | rw | **Repeat Disable**<br>$0_B$ A cancelled conversion is repeated<br>$1_B$ A cancelled conversion is discarded |
| **0** | [31:17] | r | **Reserved, write 0, read as 0** |


Subject to Agreement on the Use of Product Information

The Channel Select Register selects the channels to be converted by the group scan request source. Its bits are used to update the pending register, when a load event occurs.

The number of valid channel bits depends on the channels available in the respective product type (please refer to **“Product-Specific Configuration” on Page 16-153**).

**GxASSEL (x = 0 - 1)**
**Autoscan Source Channel Select Register, Group x**
**($x * 0400_H + 0528_H$)** **Reset Value: $0000\ 0000_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **CHSEL7** | **CHSEL6** | **CHSEL5** | **CHSEL4** | **CHSEL3** | **CHSEL2** | **CHSEL1** | **CHSEL0** |
| r | r | r | r | r | r | r | r | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **CHSELy (y = 0 - 7)** | y | rwh | **Channel Selection**<br>Each bit (when set) enables the corresponding input channel of the respective group to take part in the scan sequence.<br>$0_B$ Ignore this channel<br>$1_B$ This channel is part of the scan sequence |
| **0** | [31:8] | r | **Reserved, write 0, read as 0** |

*Note: Register GxASSEL is also updated when writing a pattern to register GxASPND.*

The Channel Pending Register indicates the channels to be converted in the current conversion sequence. They are updated from the select register, when a load event occurs.

**GxASPND (x = 0 - 1)**
**Autoscan Source Pending Register, Group x**
**(x * $0400_H$ + $052C_H$)** **Reset Value: $0000\ 0000_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **CHPND7** | **CHPND6** | **CHPND5** | **CHPND4** | **CHPND3** | **CHPND2** | **CHPND1** | **CHPND0** |
| r | r | r | r | r | r | r | r | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **CHPNDy (y = 0 - 7)** | y | rwh | **Channels Pending**<br>Each bit (when set) request the conversion of the corresponding input channel of the respective group.<br>$0_B$ Ignore this channel<br>$1_B$ Request conversion of this channel |
| **0** | [31:8] | r | **Reserved, write 0, read as 0** |

*Note: Writing to register GxASPND automatically updates register GxASSEL.*

### Registers of Background Scan Source

There is a single register set for the background scan source. This source is common for the complete VADC.

The control register of the background request source selects the external gate and/or trigger signals.
Write control bits allow separate control of each function with a simple write access.

**BRSCTRL**
**Background Request Source Control Register**
**($0200_H$)** **Reset Value: $0000\ 0000_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | GTWC | 0 | 0 | GTLVL | GTSEL | | | |
| r | r | r | r | r | r | r | r | w | r | r | rh | rw | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| XTWC | XTMODE | | XTLVL | XTSEL | | | | 0 | 0 | 0 | 0 | SRCRESREG | | | |
| w | rw | | rh | rw | | | | r | r | r | r | rw | | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **SRCRESREG** | [3:0] | rw | **Source-specific Result Register**<br>$0000_B$ Use GxCHCTRy.RESREG to select a group result register<br>$0001_B$ Store result in group result register GxRES1<br>…<br>$1111_B$ Store result in group result register GxRES15 |
| **0** | [7:4] | r | **Reserved, write 0, read as 0** |
| **XTSEL** | [11:8] | rw | **External Trigger Input Selection**<br>The connected trigger input signals are listed in **Table 16-15 “Digital Connections in the XMC1300” on Page 16-156**<br>*Note: XTSEL = $1111_B$ uses the selected gate input as trigger source (ENGT must be $0X_B$).* |
| **XTLVL** | 12 | rh | **External Trigger Level**<br>Current level of the selected trigger input |

| Field | Bits | Type | Description |
|---|---|---|---|
| **XTMODE** | [14:13] | rw | **Trigger Operating Mode**<br>$00_B$ No external trigger<br>$01_B$ Trigger event upon a falling edge<br>$10_B$ Trigger event upon a rising edge<br>$11_B$ Trigger event upon any edge |
| **XTWC** | 15 | w | **Write Control for Trigger Configuration**<br>$0_B$ No write access to trigger configuration<br>$1_B$ Bitfields XTMODE and XTSEL can be written |
| **GTSEL** | [19:16] | rw | **Gate Input Selection**<br>The connected gate input signals are listed in **Table 16-15 “Digital Connections in the XMC1300” on Page 16-156** |
| **GTLVL** | 20 | rh | **Gate Input Level**<br>Current level of the selected gate input |
| **0** | [22:21] | r | **Reserved, write 0, read as 0** |
| **GTWC** | 23 | w | **Write Control for Gate Configuration**<br>$0_B$ No write access to gate configuration<br>$1_B$ Bitfield GTSEL can be written |
| **0** | [31:24] | r | **Reserved, write 0, read as 0** |


Subject to Agreement on the Use of Product Information

The Conversion Request Mode Register configures the operating mode of the background request source.

**BRSMR**
**Background Request Source Mode Register**
**($0204_H$)** **Reset Value: $0000\ 0000_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | RPT DIS |
| r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | rw |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | LD EV | CLR PND | REQ GT | 0 | LDM | SCA N | EN SI | EN TR | ENGT | |
| r | r | r | r | r | r | w | w | rh | r | rw | rw | rw | rw | rw | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **ENGT** | [1:0] | rw | **Enable Gate**<br>Selects the gating functionality for source 1.<br>$00_B$ No conversion requests are issued<br>$01_B$ Conversion requests are issued if at least one pending bit is set<br>$10_B$ Conversion requests are issued if at least one pending bit is set and REQGTx = 1.<br>$11_B$ Conversion requests are issued if at least one pending bit is set and REQGTx = 0.<br>*Note: REQGTx is the selected gating signal.* |
| **ENTR** | 2 | rw | **Enable External Trigger**<br>$0_B$ External trigger disabled<br>$1_B$ The selected edge at the selected trigger input signal REQTR generates the load event |
| **ENSI** | 3 | rw | **Enable Source Interrupt**<br>$0_B$ No request source interrupt<br>$1_B$ A request source interrupt is generated upon a request source event (last pending conversion is finished) |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **SCAN** | 4 | rw | **Autoscan Enable**<br>$0_B$ No autoscan<br>$1_B$ Autoscan functionality enabled: a request source event automatically generates a load event |
| **LDM** | 5 | rw | **Autoscan Source Load Event Mode**<br>$0_B$ Overwrite mode: Copy all bits from the select registers to the pending registers upon a load event<br>$1_B$ Combine mode: Set all pending bits that are set in the select registers upon a load event (logic OR) |
| **0** | 6 | r | **Reserved, write 0, read as 0** |
| **REQGT** | 7 | rh | **Request Gate Level**<br>Monitors the level at the selected REQGT input.<br>$0_B$ The gate input is low<br>$1_B$ The gate input is high |
| **CLRPND** | 8 | w | **Clear Pending Bits**<br>$0_B$ No action<br>$1_B$ The bits in registers BRSPNDx are cleared |
| **LDEV** | 9 | w | **Generate Load Event**<br>$0_B$ No action<br>$1_B$ A load event is generated |
| **0** | [15:10] | r | **Reserved, write 0, read as 0** |
| **RPTDIS** | 16 | rw | **Repeat Disable**<br>$0_B$ A cancelled conversion is repeated<br>$1_B$ A cancelled conversion is discarded |
| **0** | [31:17] | r | **Reserved, write 0, read as 0** |


Subject to Agreement on the Use of Product Information

The Channel Select Registers select the channels to be converted by the background request source (channel scan source). Its bits are used to update the pending registers, when a load event occurs.

The number of valid channel bits depends on the channels available in the respective product type (please refer to **"Product-Specific Configuration" on Page 16-153**).

*Note: Priority channels selected in registers **GxCHASS (x = 0 - 1)** will not be converted.*

**BRSSELx (x = 0 - 1)**
**Background Request Source Channel Select Register, Group x**
**($0180_H$ + x * $0004_H$)** **Reset Value: 0000 $0000_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **CH SEL G7** | **CH SEL G6** | **CH SEL G5** | **CH SEL G4** | **CH SEL G3** | **CH SEL G2** | **CH SEL G1** | **CH SEL G0** |
| r | r | r | r | r | r | r | r | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **CHSELGy (y = 0 - 7)** | y | rwh | **Channel Selection Group x**<br>Each bit (when set) enables the corresponding input channel of the respective group to take part in the background scan sequence.<br>$0_B$ Ignore this channel<br>$1_B$ This channel is part of the scan sequence |
| **0** | [31:8] | r | **Reserved, write 0, read as 0** |

The Channel Pending Registers indicate the channels to be converted in the current conversion sequence. They are updated from the select registers, when a load event occurs.

**BRSPNDx (x = 0 - 1)**
**Background Request Source Pending Register, Group x**

**($01C0_H$ + x * $0004_H$)** **Reset Value: $0000\ 0000_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **CHPNDG7** | **CHPNDG6** | **CHPNDG5** | **CHPNDG4** | **CHPNDG3** | **CHPNDG2** | **CHPNDG1** | **CHPNDG0** |
| r | r | r | r | r | r | r | r | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **CHPNDGy (y = 0 - 7)** | y | rwh | **Channels Pending Group x**<br>Each bit (when set) request the conversion of the corresponding input channel of the respective group.<br>$0_B$ Ignore this channel<br>$1_B$ Request conversion of this channel |
| **0** | [31:8] | r | **Reserved, write 0, read as 0** |

*Note: Writing to any of registers BRSPNDx automatically updates the corresponding register BRSSELx and generates a load event that copies all bits from all registers BRSSELx to BRSPNDx.*
*Use this shortcut only when writing the last word of the request pattern.*


Subject to Agreement on the Use of Product Information

### 16.16.5 Channel Control Registers

**G0CHCTRy (y = 0 - 7)**
**Group 0, Channel y Ctrl. Reg. ($0600_H$ + y * $0004_H$)** **Reset Value: 0000 $0000_H$**
**G1CHCTRy (y = 0 - 7)**
**Group 1, Channel y Ctrl. Reg. ($0A00_H$ + y * $0004_H$)** **Reset Value: 0000 $0000_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | BWDEN | BWDCH | | 0 | 0 | 0 | 0 | 0 | 0 | RESPOS | RESTBS | RESREG | | | |
| r | rw | rw | | r | r | r | r | r | r | rw | rw | rw | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BNDSELX | | | | REFSEL | SYNC | CHEVMODE | | BNDSELU | | BNDSELL | | 0 | 0 | ICLSEL | |
| rw | | | | rw | rw | rw | | rw | | rw | | r | r | rw | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **ICLSEL** | [1:0] | rw | **Input Class Select**<br>$00_B$ Use group-specific class 0<br>$01_B$ Use group-specific class 1<br>$10_B$ Use global class 0<br>$11_B$ Use global class 1 |
| **0** | [3:2] | r | **Reserved, write 0, read as 0** |
| **BNDSELL** | [5:4] | rw | **Lower Boundary Select**[1)]<br>$00_B$ Use group-specific boundary 0<br>$01_B$ Use group-specific boundary 1<br>$10_B$ Use global boundary 0<br>$11_B$ Use global boundary 1 |
| **BNDSELU** | [7:6] | rw | **Upper Boundary Select**[1)]<br>$00_B$ Use group-specific boundary 0<br>$01_B$ Use group-specific boundary 1<br>$10_B$ Use global boundary 0<br>$11_B$ Use global boundary 1 |

| Field | Bits | Type | Description |
|---|---|---|---|
| **CHEVMODE** | [9:8] | rw | **Channel Event Mode**<br>Generate a channel event either<br>in normal compare mode (NCM) with limit checking[2)]<br>or in Fast Compare Mode (FCM)[3)]<br>$00_B$ Never<br>$01_B$ NCM: If result is inside the boundary band<br>FCM: If result becomes high (above cmp. val.)<br>$10_B$ NCM: If result is outside the boundary band<br>FCM: If result becomes low (below cmp. val.)<br>$11_B$ NCM: Always (ignore band)<br>FCM: If result switches to either level |
| **SYNC** | 10 | rw | **Synchronization Request**<br>$0_B$ No synchroniz. request, standalone operation<br>$1_B$ Request a synchronized conversion of this channel (only taken into account for a master) |
| **REFSEL** | 11 | rw | **Reference Input Selection**<br>Defines the reference voltage input to be used for conversions on this channel.<br>$0_B$ Standard reference Ground $V_{SS}$<br>$1_B$ Alternate reference Ground from CH0[4)] |
| **BNDSELX** | [15:12] | rw | **BoundaryExtension[1)]**<br>$0000_B$ Standard mode:<br>select boundaries via BNDSELU/BNDSELL<br>$0001_B$ Use result reg. GxRES1 as upper boundary<br>…<br>$1111_B$ Use result reg. GxRES15 as upper boundary |
| **RESREG** | [19:16] | rw | **Result Register**<br>$0000_B$ Store result in group result register GxRES0<br>…<br>$1111_B$ Store result in group result register GxRES15 |
| **RESTBS** | 20 | rw | **Result Target for Background Source**<br>$0_B$ Store results in the selected group result register<br>$1_B$ Store results in the global result register |
| **RESPOS** | 21 | rw | **Result Position**<br>$0_B$ Store results left-aligned<br>$1_B$ Store results right-aligned |
| **0** | [27:22] | r | **Reserved, write 0, read as 0** |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **BWDCH** | [29:28] | rw | **Broken Wire Detection Channel**<br>$00_B$ Select $V_{AREF}$<br>$01_B$ Select $V_{AGND}$<br>$10_B$ Reserved<br>$11_B$ Reserved |
| **BWDEN** | 30 | rw | **Broken Wire Detection Enable**<br>$0_B$ Normal operation<br>$1_B$ Additional preparation phase is enabled |
| **0** | 31 | r | **Reserved, write 0, read as 0** |

1) While BNDSELX ≠ $0000_B$, bitfields BNDSELU and BNDSELL are concatenated and select the corresponding result register as lower boundary.
2) The boundary band is defined as the area where the result is less than or equal to the selected upper boundary and greater than or equal to the selected lower boundary, see **Section 16.8.4**.
3) The result is bit FCR in the selected result register.
4) Some channels cannot select an alternate reference.

**GxICLASS0 (x = 0 - 1)**
**Input Class Register 0, Group x**

**($x * 0400_H + 04A0_H$)** **Reset Value: $0000\ 0000_H$**

**GxICLASS1 (x = 0 - 1)**
**Input Class Register 1, Group x**

**($x * 0400_H + 04A4_H$)** **Reset Value: $0000\ 0000_H$**

**GLOBICLASSy (y = 0 - 1)**
**Input Class Register y, Global**

**($00A0_H + y * 0004_H$)** **Reset Value: $0000\ 0000_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **0** | **0** | **0** | **0** | **CME** | | | **0** | **0** | **0** | **STCE** | | | | |
| r | r | r | r | r | rw | | | r | r | r | rw | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **0** | **0** | **0** | **0** | **CMS** | | | **0** | **0** | **0** | **STCS** | | | | |
| r | r | r | r | r | rw | | | r | r | r | rw | | | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **STCS** | [4:0] | rw | **Sample Time Control for Standard Conversions**<br>Number of additional clock cycles to be added to the minimum sample phase of 2 analog clock cycles:<br>Coding and resulting sample time see **Table 16-11**.<br>For conversions of external channels, the value from bitfield STCE can be used. |
| **0** | [7:5] | r | **Reserved, write 0, read as 0** |
| **CMS** | [10:8] | rw | **Conversion Mode for Standard Conversions**<br>$000_B$ 12-bit conversion<br>$001_B$ 10-bit conversion<br>$010_B$ 8-bit conversion<br>$011_B$ Reserved<br>$100_B$ Reserved<br>$101_B$ 10-bit fast compare mode<br>$110_B$ Reserved<br>$111_B$ Reserved |
| **0** | [15:11] | r | **Reserved, write 0, read as 0** |
| **STCE** | [20:16] | rw | **Sample Time Control for EMUX Conversions**<br>Number of additional clock cycles to be added to the minimum sample phase of 2 analog clock cycles:<br>Coding and resulting sample time see **Table 16-11**.<br>For conversions of standard channels, the value from bitfield STCS is used. |
| **0** | [23:21] | r | **Reserved, write 0, read as 0** |
| **CME** | [26:24] | rw | **Conversion Mode for EMUX Conversions**<br>$000_B$ 12-bit conversion<br>$001_B$ 10-bit conversion<br>$010_B$ 8-bit conversion<br>$011_B$ Reserved<br>$100_B$ Reserved<br>$101_B$ 10-bit fast compare mode<br>$110_B$ Reserved<br>$111_B$ Reserved |
| **0** | [31:27] | r | **Reserved, write 0, read as 0** |


Subject to Agreement on the Use of Product Information

**Table 16-11 Sample Time Coding**

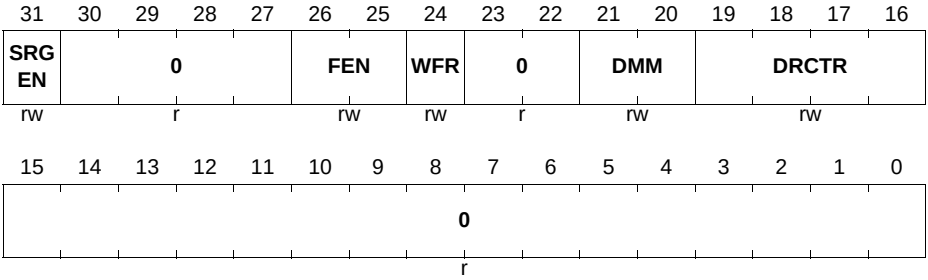
| STCS / STCE | Additional Clock Cycles | Sample Time |
|---|---|---|
| $0\ 0000_B$ | 0 | 2 / $f_{ADCI}$ |
| $0\ 0001_B$ | 1 | 3 / $f_{ADCI}$ |
| … | … | … |
| $0\ 1111_B$ | 15 | 17 / $f_{ADCI}$ |
| $1\ 0000_B$ | 16 | 18 / $f_{ADCI}$ |
| $1\ 0001_B$ | 32 | 34 / $f_{ADCI}$ |
| … | … | … |
| $1\ 1110_B$ | 240 | 242 / $f_{ADCI}$ |
| $1\ 1111_B$ | 256 | 258 / $f_{ADCI}$ |


Subject to Agreement on the Use of Product Information

### 16.16.6 Result Registers

The group result control registers select the behavior of the result registers of a given group.

**G0RCRy (y = 0 - 15)**
**Group 0 Result Control Reg. y (0680$_H$ + y * 0004$_H$)** **Reset Value: 0000 0000$_H$**
**G1RCRy (y = 0 - 15)**
**Group 1 Result Control Reg. y (0A80$_H$ + y * 0004$_H$)** **Reset Value: 0000 0000$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | [15:0] | r | **Reserved, write 0, read as 0** |
| **DRCTR** | [19:16] | rw | **Data Reduction Control**<br>Defines how result values are stored/accumulated in this register for the final result. The data reduction counter DRC can be loaded from this bitfield.<br>The function of bitfield DRCTR is determined by bitfield DMM. |
| **DMM** | [21:20] | rw | **Data Modification Mode**<br>$00_B$ Standard data reduction (accumulation)<br>$01_B$ Result filtering mode[1)]<br>$10_B$ Difference mode<br>$11_B$ Reserved<br>See **“Data Modification” on Page 16-49** |
| **0** | [23:22] | r | **Reserved, write 0, read as 0** |
| **WFR** | 24 | rw | **Wait-for-Read Mode Enable**<br>$0_B$ Overwrite mode<br>$1_B$ Wait-for-read mode enabled for this register |


Subject to Agreement on the Use of Product Information

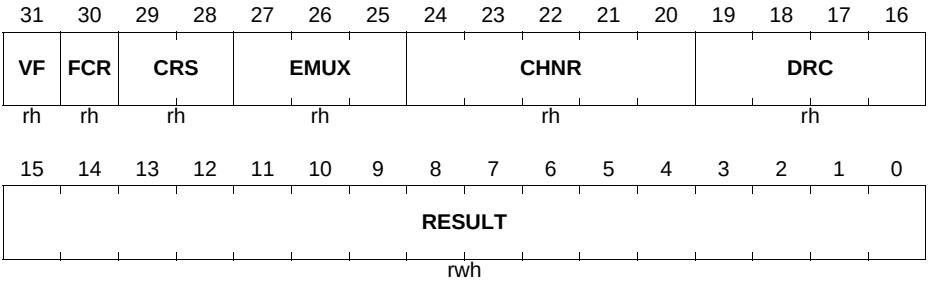
| Field | Bits | Type | Description |
|---|---|---|---|
| **FEN** | [26:25] | rw | **FIFO Mode Enable**<br>$00_B$ Separate result register<br>$01_B$ Part of a FIFO structure: copy each new valid result<br>$10_B$ Maximum mode: copy new result if bigger<br>$11_B$ Minimum mode: copy new result if smaller |
| **0** | [30:27] | r | **Reserved, write 0, read as 0** |
| **SRGEN** | 31 | rw | **Service Request Generation Enable**<br>$0_B$ No service request<br>$1_B$ Service request after a result event |

1) The filter registers are cleared while bitfield DMM ≠ $01_B$.

The group result registers provide a selectable storage location for all channels of a given group.

*Note: The preset value used in fast compare mode is written to the respective result register. The debug result registers are not writable.*

**G0RESy (y = 0 - 15)**
**Group 0 Result Register y** **($0700_H$ + y * $0004_H$)** **Reset Value: 0000 $0000_H$**
**G1RESy (y = 0 - 15)**
**Group 1 Result Register y** **($0B00_H$ + y * $0004_H$)** **Reset Value: 0000 $0000_H$**

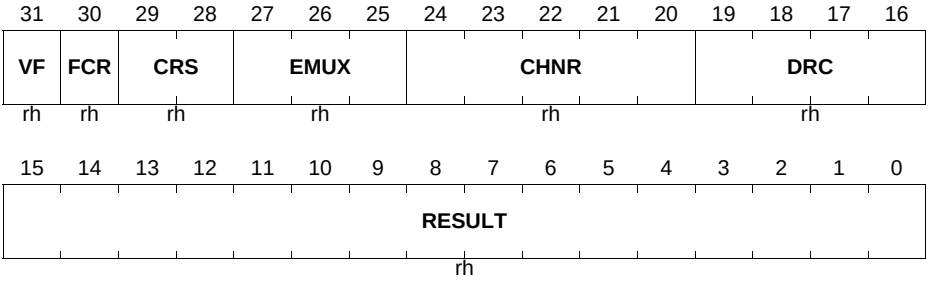



| Field | Bits | Type | Description |
|---|---|---|---|
| **RESULT** | [15:0] | rwh | **Result of Most Recent Conversion**<br>The position of the result bits within this bitfield depends on the configured operating mode.<br>Please, refer to **Section 16.11.2**. |
| **DRC** | [19:16] | rh | **Data Reduction Counter**<br>Indicates the number of values still to be accumulated for the final result. The final result is available and valid flag VF is set when bitfield DRC becomes zero (by decrementing or by reload).<br>See **"Data Modification" on Page 16-49** |
| **CHNR** | [24:20] | rh | **Channel Number**<br>Indicates the channel number corresponding to the value in bitfield RESULT. |
| **EMUX** | [27:25] | rh | **External Multiplexer Setting**<br>Indicates the setting of the external multiplexer, corresponding to the value in bitfield RESULT.<br>*Note: Available in GxRES0 only. Use GxRES0 if EMUX information is required.* |

| Field | Bits | Type | Description |
|---|---|---|---|
| **CRS** | [29:28] | rh | **Converted Request Source**<br>Indicates the request source that as requested the conversion to which the result value in bitfield RESULT belongs.<br>$00_B$ Request source 0<br>$01_B$ Request source 1<br>$10_B$ Request source 2<br>$11_B$ Reserved |
| **FCR** | 30 | rh | **Fast Compare Result**<br>Indicates the result of an operation in Fast Compare Mode.<br>$0_B$ Signal level was below compare value<br>$1_B$ Signal level was above compare value |
| **VF** | 31 | rh | **Valid Flag**<br>Indicates a new result in bitfield RESULT or bit FCR.<br>$0_B$ No new result available<br>$1_B$ Bitfield RESULT has been updated with new result value and has not yet been read, or bit FCR has been updated |

The debug view of the group result registers provides access to all result registers of a given group, however, without clearing the valid flag.

**G0RESDy (y = 0 - 15)**
**Group 0 Result Reg. y, Debug ($0780_H$ + y * $0004_H$)** **Reset Value: 0000 $0000_H$**
**G1RESDy (y = 0 - 15)**
**Group 1 Result Reg. y, Debug ($0B80_H$ + y * $0004_H$)** **Reset Value: 0000 $0000_H$**



| Field | Bits | Type | Description |
|---|---|---|---|
| **RESULT** | [15:0] | rh | **Result of Most Recent Conversion**<br>The position of the result bits within this bitfield depends on the configured operating mode.<br>Please, refer to **Section 16.11.2**. |
| **DRC** | [19:16] | rh | **Data Reduction Counter**<br>Indicates the number of values still to be accumulated for the final result. The final result is available and valid flag VF is set when bitfield DRC becomes zero (by decrementing or by reload).<br>See **"Data Modification" on Page 16-49** |
| **CHNR** | [24:20] | rh | **Channel Number**<br>Indicates the channel number corresponding to the value in bitfield RESULT. |
| **EMUX** | [27:25] | rh | **External Multiplexer Setting**<br>Indicates the setting of the external multiplexer, corresponding to the value in bitfield RESULT.<br>*Note: Available in GxRESD0 only. Use GxRESD0 if EMUX information is required.* |
| **CRS** | [29:28] | rh | **Converted Request Source**<br>Indicates the request source that as requested the conversion to which the result value in bitfield RESULT belongs.<br>$00_B$ Request source 0<br>$01_B$ Request source 1<br>$10_B$ Request source 2<br>$11_B$ Reserved |
| **FCR** | 30 | rh | **Fast Compare Result**<br>Indicates the result of an operation in Fast Compare Mode.<br>$0_B$ Signal level was below compare value<br>$1_B$ Signal level was above compare value |

| Field | Bits | Type | Description |
|---|---|---|---|
| **VF** | 31 | rh | **Valid Flag**<br>Indicates a new result in bitfield RESULT or bit FCR.<br>$0_B$ No new result available<br>$1_B$ Bitfield RESULT has been updated with new result value and has not yet been read, or bit FCR has been updated |

The global result control register selects the behavior of the global result register.

**GLOBRCR**
**Global Result Control Register ($0280_H$) Reset Value: $0000\ 0000_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **SRGEN** | **0** | **0** | **0** | **0** | **0** | **0** | **WFR** | **0** | **0** | **0** | **0** | **DRCTR** | | | |
| rw | r | r | r | r | r | r | rw | r | r | r | r | rw | | | |

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

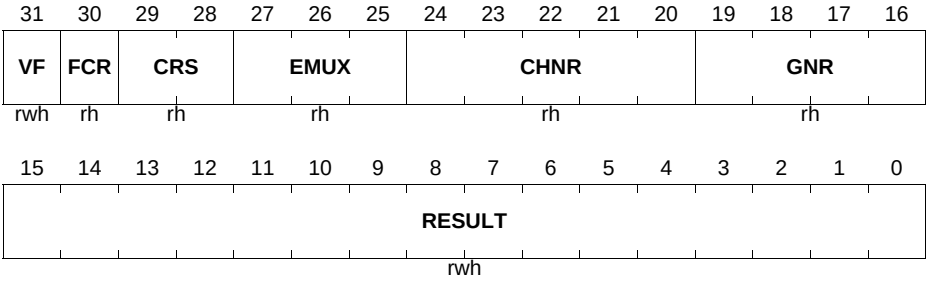
| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | [15:0] | r | **Reserved, write 0, read as 0** |
| **DRCTR** | [19:16] | rw | **Data Reduction Control**<br>Defines how result values are stored/accumulated in this register for the final result. The data reduction counter DRC can be loaded from this bitfield.<br>$0000_B$Data reduction disabled<br>others: see **“Function of Bitfield DRCTR” on Page 16-50**[1)] |
| **0** | [23:20] | r | **Reserved, write 0, read as 0** |
| **WFR** | 24 | rw | **Wait-for-Read Mode Enable**<br>$0_B$ Overwrite mode<br>$1_B$ Wait-for-read mode enabled for this register |
| **0** | [30:25] | r | **Reserved, write 0, read as 0** |

| Field | Bits | Type | Description |
|---|---|---|---|
| **SRGEN** | 31 | rw | **Service Request Generation Enable**<br>$0_B$ No service request<br>$1_B$ Service request after a result event |

1) Only standard data reduction is available for the global result register, i.e. DMM is assumed as $00_B$.

The global result register provides a common storage location for all channels of all groups.

**GLOBRES**
**Global Result Register** **($0300_H$)** **Reset Value: 0000 0000$_H$**
**GLOBRESD**
**Global Result Register, Debug** **($0380_H$)** **Reset Value: 0000 0000$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **RESULT** | [15:0] | rwh | **Result of most recent conversion**<br>The position of the result bits within this bitfield depends on the configured operating mode.[1)]<br>Please, refer to **Section 16.11.2**. |
| **GNR** | [19:16] | rh | **Group Number**<br>Indicates the group to which the channel number in bitfield CHNR refers. |
| **CHNR** | [24:20] | rh | **Channel Number**<br>Indicates the channel number corresponding to the value in bitfield RESULT. |
| **EMUX** | [27:25] | rh | **External Multiplexer Setting**<br>Indicates the setting of the external multiplexer, corresponding to the value in bitfield RESULT. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **CRS** | [29:28] | rh | **Converted Request Source**<br>Indicates the request source that as requested the conversion to which the result value in bitfield RESULT belongs. |
| **FCR** | 30 | rh | **Fast Compare Result**<br>Indicates the result of an operation in Fast Compare Mode.<br>$0_B$ Signal level was below compare value<br>$1_B$ Signal level was above compare value |
| **VF** | 31 | rwh | **Valid Flag**<br>Indicates a new result in bitfield RESULT or bit FCR.<br>$0_B$ Read access: No new valid data available<br>Write access: No effect<br>$1_B$ Read access: Bitfield RESULT contains valid data and has not yet been read, or bit FCR has been updated<br>Write access: Clear this valid flag and the data reduction counter (overrides a hardware set action)[1)] |

1) Only writable in register GLOBRES, not in register GLOBRESD.

The valid flag register summarizes the valid flags of all result registers.

**GxVFR (x = 0 - 1)**
**Valid Flag Register, Group x** **($x * 0400_H + 05F8_H$)** **Reset Value: $0000\ 0000_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **VF15** | **VF14** | **VF13** | **VF12** | **VF11** | **VF10** | **VF9** | **VF8** | **VF7** | **VF6** | **VF5** | **VF4** | **VF3** | **VF2** | **VF1** | **VF0** |
| rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **VFy (y = 0 - 15)** | y | rwh | **Valid Flag of Result Register x**<br>Indicates a new result in bitfield RESULT or in bit FCR.<br>$0_B$ Read access: No new valid data available<br>Write access: No effect<br>$1_B$ Read access: Result register x contains valid data and has not yet been read, or bit FCR has been updated<br>Write access: Clear this valid flag and bitfield DRC in register GxRESy (overrides a hardware set action) |
| **0** | [31:16] | r | **Reserved, write 0, read as 0** |

### 16.16.7 Calbration Registers

The calibration control register CALCTR selects basic calibration features.

**SHS0_CALCTR**
**Calibration Control Register (4803 40BC$_H$) Reset Value: 0310 0400$_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **SUCAL** | **0** | **CALMAX** | | | | | | **0** | **SUCALVAL** | | | | | | |
| w | r | rw | | | | | | r | rw | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **GCDIS** | **OCDIS** | **CALGNSTC** | | | | | | **0** | **0** | **0** | **0** | **0** | **0** | **RANCAL** | **CALORD** |
| rw | rw | rw | | | | | | r | r | r | r | r | r | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **CALORD** | 0 | rw | **Calibration Order**<br>$0_B$ Do conversions then calibration<br>$1_B$ Do calibration then conversions |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **RANCAL** | 1 | rw | **Random Calibration**<br>$0_B$ Calibrate after each conversion round<br>$1_B$ Randomly shift calibration after a conversion round<br>*Note: Random calibration is only supported for postcalibration, i.e. CALORD = 0.* |
| **0** | [7:2] | r | **Reserved, write 0, read as 0** |
| **CALGNSTC** | [13:8] | rw | **Gain Calibration Sample Time Control**<br>Defines the duration of the sample phase for gain calibration cycles.<br>$00_H$ Reserved<br>Others: $t_{SGN}$ = CALGNSTC / $f_{SH}$<br>*Note: To be adjusted for internal reference operation in low voltage range (see Data Sheet).* |
| **OCDIS** | 14 | rw | **Offset Calibration Disable**<br>$0_B$ Standard calibration<br>$1_B$ Exclude offset calibration steps from the sequence |
| **GCDIS** | 15 | rw | **Gain Calibration Disable**<br>$0_B$ Standard calibration<br>$1_B$ Exclude gain calibration steps from the sequence |
| **SUCALVAL** | [22:16] | rw | **Startup Calibration Cycles**<br>Defines the number of calibration sequences for the startup calibration.<br>*Note: No user control required.* |
| **CALMAX** | [29:24] | rw | **Calibration Maximum Timing**<br>Defines the maximum time until the next calibration:<br>$t_{MAX}$ = 128 / $f_{SH}$ × (CALMAX + 1) |
| **0** | 30 | r | **Reserved, write 0, read as 0** |
| **SUCAL** | 31 | w | **Start-Up Calibration**<br>The 0-1 transition of bit SUCAL triggers the start-up calibration phase of the converter.[1)]<br>SUCAL is cleared automatically when the calibration phase begins.<br>$0_B$ No action<br>$1_B$ Initiate the start-up calibration phase (indication in bitfield SHSCFG.STATE) |

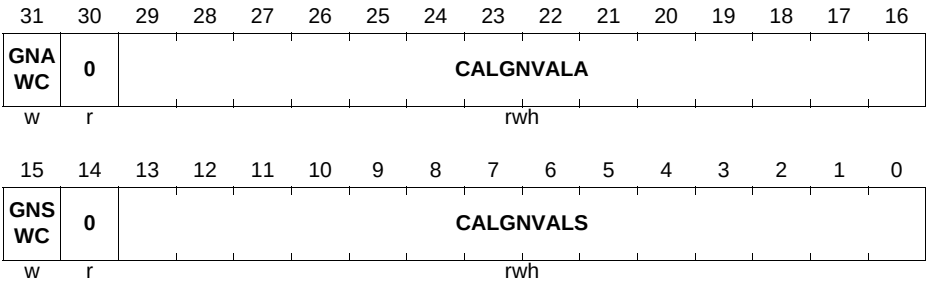

Subject to Agreement on the Use of Product Information

1) Startup calibration can also be triggered by the known bit SUCAL in register GLOBCFG in module VADCDIG.

The gain calibration control register CALGCx configures the gain calibration features of the associated S&H unit. The calibration values are determined automatically by the calibration mechanism. They can be read to compensate the error introduced by using the alternate reference of another group.

**SHS0_CALGCx (x = 0 - 1)**
**Gain Calibration Control Register x**

**(4803 40C0$_H$ + x * 4$_H$)** **Reset Value: 2000 2000$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **CALGNVALS** | [13:0] | rwh | **Gain Calibration Value, Standard Reference**<br>Modified by the calibration steps, controls the gain DAC. |
| **0** | 14 | r | **Reserved, write 0, read as 0** |
| **GNSWC** | 15 | w | **Gain Calibration Write Control, Standard**<br>$0_B$ No write access to gain calibration parameter<br>$1_B$ CALGNVALS can be written |
| **CALGNVALA** | [29:16] | rwh | **Gain Calibration Value, Alternate Reference**<br>Modified by the calibration steps, controls the gain DAC. |
| **0** | 30 | r | **Reserved, write 0, read as 0** |
| **GNAWC** | 31 | w | **Gain Calibration Write Control, Alternate**<br>$0_B$ No write access to gain calibration parameter<br>$1_B$ CALGNVALA can be written |

*Note: After writing to bitfield CALGNVALS or CALGNVALA a settling time of 10* µs must be respected before starting subsequent conversions.


Subject to Agreement on the Use of Product Information

The offset calibration control register CALOCx configures the offset calibration features of the associated S&H unit. The calibration values can be written for testing purposes.

There is a separate calibration value for each gain level.

*Note: If more than 4 gain levels are used, registers CALOCXx can be added at locations ($x * 4_H + 0100_H$).*

**SHS0_CALOCx (x = 0 - 1)**
**Offset Calibration Control Register x**

**(4803 40E0$_H$ + x * 4$_H$)** **Reset Value: 0000 0000$_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **DISCAL** | **CALOFFVAL3** | | | | | | | **0** | **CALOFFVAL2** | | | | | | |
| rw | rwh | | | | | | | r | rwh | | | | | | |

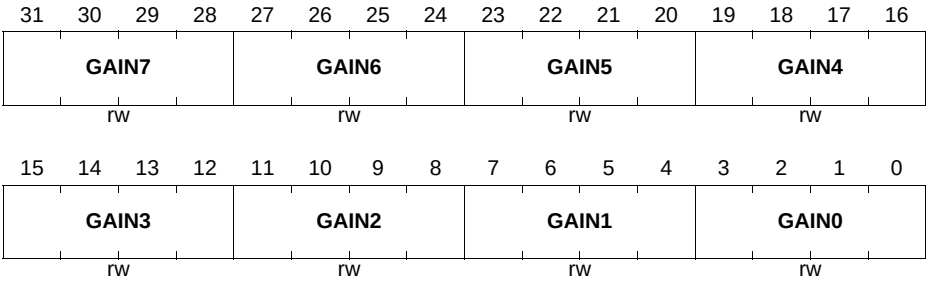
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **OFFWC** | **CALOFFVAL1** | | | | | | | **0** | **CALOFFVAL0** | | | | | | |
| w | rwh | | | | | | | r | rwh | | | | | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **CALOFFVAL0, CALOFFVAL1, CALOFFVAL2, CALOFFVAL3** | [6:0], [14:8], [22:16], [30:24] | rwh | **Offset Calibration Value for Gain Level z**<br>Modified by the calibration steps, controls the offset DAC and the shift DAC<br>CALOFFVALz corresponds to the settings in register GNCTR. |
| **0** | 7 | r | **Reserved, write 0, read as 0** |
| **OFFWC** | 15 | w | **Offset Calibration Write Control**<br>$0_B$ No write access to offset cal. parameters<br>$1_B$ CALOFFVALz can be written |
| **0** | 23 | r | **Reserved, write 0, read as 0** |
| **DISCAL** | 31 | rw | **Disable Calibration**<br>$0_B$ Calibration enabled (offset and gain)<br>$1_B$ No calibration |


Subject to Agreement on the Use of Product Information

The gain control registers GNCTRx0 selects the gain level for each input.

**SHS0_GNCTR00**
**Gain Control Register 00** **(4803 4180$_H$)** **Reset Value: 0000 0000$_H$**
**SHS0_GNCTR10**
**Gain Control Register 10** **(4803 4190$_H$)** **Reset Value: 0000 0000$_H$**



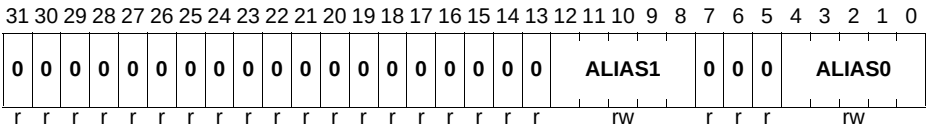

| Field | Bits | Type | Description |
|---|---|---|---|
| **GAINz**<br>**(z = 0 - 7)** | [z*4+3: z*4] | rw | **Gain Control z**<br>$0000_B$Gain factor = 1<br>$0001_B$Gain factor = 3<br>$0010_B$Gain factor = 6<br>$0011_B$Gain factor = 12<br>Others: Reserved |

*Note: The first index (x0) indicates the associated S&H unit.*
*The channel number is z.*

### 16.16.8 Miscellaneous Registers

The alias register can replace the channel numbers of channels CH0 and CH1 with another channel number. The reset value disables this redirection.

**GxALIAS (x = 0 - 1)**
**Alias Register, Group x** **(x * 0400$_H$ + 04B0$_H$)** **Reset Value: 0000 0100$_H$**



| Field | Bits | Type | Description |
|---|---|---|---|
| **ALIAS0** | [4:0] | rw | **Alias Value for CH0 Conversion Requests**<br>Indicates the channel that is converted instead of channel CH0. The conversion is done with the settings defined for channel CH0. |
| **0** | [7:5] | r | **Reserved, write 0, read as 0** |
| **ALIAS1** | [12:8] | rw | **Alias Value for CH1 Conversion Requests**<br>Indicates the channel that is converted instead of channel CH1. The conversion is done with the settings defined for channel CH1. |
| **0** | [31:13] | r | **Reserved, write 0, read as 0** |


Subject to Agreement on the Use of Product Information

The local boundary register GxBOUND defines group-specific boundary values or delta limits for Fast Compare Mode.
The global boundary register GLOBBOUND defines general compare values for all channels.

Depending on the conversion width, the respective left 12/10/8 bits of a bitfield are used. For 10/8-bit results, the lower 2/4 bits must be zero!

**GxBOUND (x = 0 - 1)**
**Boundary Select Register, Group x**
**($x * 0400_H + 04B8_H$)** **Reset Value: $0000\ 0000_H$**

**GLOBBOUND**
**Global Boundary Select Register ($00B8_H$)** **Reset Value: $0000\ 0000_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | 0 | | 0 | | 0 | | BOUNDARY1 | | | | | | | | 0 | | 0 | | 0 | | 0 | | BOUNDARY0 | | | | | | | |
| r | | r | | r | | r | | rw | | | | | | | | r | | r | | r | | r | | rw | | | | | | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **BOUNDARY0** | [11:0] | rw | **Boundary Value 0 for Limit Checking**<br>Standard Mode: This value is compared against the left-aligned conversion result.<br>Fast Compare Mode: This value is added to the reference value (upper delta). |
| **0** | [15:12] | r | **Reserved, write 0, read as 0** |
| **BOUNDARY1** | [27:16] | rw | **Boundary Value 1 for Limit Checking**<br>Standard Mode: This value is compared against the left-aligned conversion result.<br>Fast Compare Mode: This value is subtracted from the reference value (lower delta). |
| **0** | [31:28] | r | **Reserved, write 0, read as 0** |


Subject to Agreement on the Use of Product Information

The Boundary Flag Register holds the boundary flags themselves together with bits to select the activation condition and the output signal polarity for each flag.

**GxBFL (x = 0 - 1)**
**Boundary Flag Register, Group x**

**(x * $0400_H$ + $04C8_H$)** **Reset Value: $0000\ 0000_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **BFI 3** | **BFI 2** | **BFI 1** | **BFI 0** |
| r | r | r | r | r | r | r | r | r | r | r | r | rw | rw | rw | rw |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **0** | **0** | **0** | **BFA 3** | **BFA 2** | **BFA 1** | **BFA 0** | **0** | **0** | **0** | **0** | **BFL 3** | **BFL 2** | **BFL 1** | **BFL 0** |
| r | r | r | r | rw | rw | rw | rw | r | r | r | r | rh | rh | rh | rh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **BFLy (y = 0 - 3)** | y | rh | **Boundary Flag y**<br>$0_B$ Passive state: result has not yet crossed the activation boundary (see bitfield BFAy), or selected gate signal is inactive, or this boundary flag is disabled<br>$1_B$ Active state: result has crossed the activation boundary |
| **0** | [7:4] | r | **Reserved, write 0, read as 0** |
| **BFAy (y = 0 - 3)** | 8 + y | rw | **Boundary Flag y Activation Select**<br>$0_B$ Set boundary flag BFLy if result is above the defined band or compare value, clear if below<br>$1_B$ Set boundary flag BFLy if result is below the defined band or compare value, clear if above |
| **0** | [15:12] | r | **Reserved, write 0, read as 0** |
| **BFIy (y = 0 - 3)** | 16 + y | rw | **Boundary Flag y Inversion Control**<br>$0_B$ Use BFLy directly<br>$1_B$ Invert value and use $\overline{\text{BFLy}}$ |
| **0** | [31:20] | r | **Reserved, write 0, read as 0** |


Subject to Agreement on the Use of Product Information

The Boundary Flag Software Register provides means to set or clear each flag by software.

**GxBFLS (x = 0 - 1)**
**Boundary Flag Software Register, Group x**
**(x * 0400$_H$ + 04CC$_H$)** **Reset Value: 0000 0000$_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **BFS 3** | **BFS 2** | **BFS 1** | **BFS 0** |
| r | r | r | r | r | r | r | r | r | r | r | r | w | w | w | w |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **BFC 3** | **BFC 2** | **BFC 1** | **BFC 0** |
| r | r | r | r | r | r | r | r | r | r | r | r | w | w | w | w |

| Field | Bits | Type | Description |
|---|---|---|---|
| **BFCy (y = 0 - 3)** | y | w | **Boundary Flag y Clear**<br>$0_B$ No action<br>$1_B$ Clear bit BFLy |
| **0** | [15:4] | r | **Reserved, write 0, read as 0** |
| **BFSy (y = 0 - 3)** | 16 + y | w | **Boundary Flag y Set**<br>$0_B$ No action<br>$1_B$ Set bit BFLy |
| **0** | [31:20] | r | **Reserved, write 0, read as 0** |

*Note: If a boundary flag is used together with Fast Compare Mode, it is recommended not to direct results from other channels to the corresponding result register.*


Subject to Agreement on the Use of Product Information

The Boundary Flag Control Register selects the basic operation of the boundary flags.

**GxBFLC (x = 0 - 1)**
**Boundary Flag Control Register, Group x**

**(x * $0400_H$ + $04D0_H$)** **Reset Value: $0000\ 0000_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **BFM 3** | | | | **BFM 2** | | | | **BFM 1** | | | | **BFM 0** | | | |
| rw | | | | rw | | | | rw | | | | rw | | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **BFM0, BFM1, BFM2, BFM3** | [3:0], [7:4], [11:8], [15:12] | rw | **Boundary Flag y Mode Control**<br>$0000_B$ Disable boundary flag, BFLy is not changed<br>$0001_B$ Always enable boundary flag (follow compare results)<br>$0010_B$ Enable boundary flag while gate of source 0 is active, clear BFLy while gate is inactive<br>$0011_B$ Enable boundary flag while gate of source 1 is active, clear BFLy while gate is inactive<br>others: Reserved |
| **0** | [31:16] | r | **Reserved, write 0, read as 0** |


Subject to Agreement on the Use of Product Information

The Boundary Flag Node Pointer Register directs signal GxBFLOUTy to alternate on-chip connections with other modules (in addition to the group-specific outputs). Possible targets are the corresponding common service request lines or the common boundary flag outputs (CBFL0 … CBFL3).

**GxBFLNP (x = 0 - 1)**
**Boundary Flag Node Pointer Register, Group x**
**(x * $0400_H$ + $04D4_H$)** **Reset Value: $0000\ FFFF_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **BFL3NP** | | | | **BFL2NP** | | | | **BFL1NP** | | | | **BFL0NP** | | | |
| rw | | | | rw | | | | rw | | | | rw | | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **BFL0NP, BFL1NP, BFL2NP, BFL3NP** | [3:0], [7:4], [11:8], [15:12] | rw | **Boundary Flag y Node Pointer**<br>$0000_B$Select common bondary flag output 0<br>…<br>$0011_B$Select common bondary flag output 3<br>$0100_B$Select common service request line C0SR0<br>…<br>$0111_B$Select common service request line C0SR3<br>$1111_B$Disabled, no common output signal<br>others: Reserved |
| **0** | [31:16] | r | **Reserved, write 0, read as 0** |

The Synchronization Control Register controls the synchronization of kernels for parallel conversions.

*Note: Program register GxSYNCTR only while bitfield GxARBCFG.ANONS = $00_B$ in all ADC kernels of the conversion group. Set the master's bitfield ANONC to $11_B$ afterwards.*

**GxSYNCTR (x = 0 - 1)**
**Synchronization Control Register, Group x**

**(x * $0400_H$ + $04C0_H$)** **Reset Value: $0000\ 0000_H$**

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

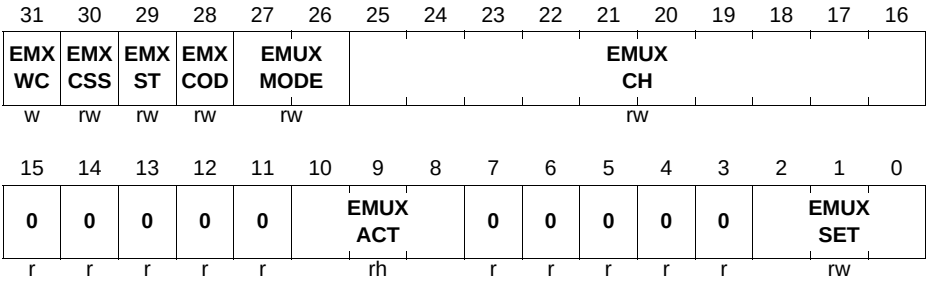
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | EVALR1 | 0 | 0 | STSEL | |
| r | r | r | r | r | r | r | r | r | r | r | rw | r | r | rw | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **STSEL** | [1:0] | rw | **Start Selection**<br>Controls the synchronization mechanism of the ADC kernel.<br>$00_B$ Kernel is synchronization master: Use own bitfield GxARBCFG.ANONC<br>$01_B$ Kernel is synchronization slave: Control information from input CI1<br>$10_B$ Reserved<br>$11_B$ Reserved<br>*Note: Control inputs CIx see **Figure 16-28**, connected kernels see **Table 16-13**.* |
| **0** | [3:2] | r | **Reserved, write 0, read as 0** |
| **EVALR1** | 4 | rw | **Evaluate Ready Input R1**<br>Enables the ready input signal for a kernel of a conversion group.<br>$0_B$ No ready input control<br>$1_B$ Ready input R1 is considered for the start of a parallel conversion of this conversion group |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | [31:5] | r | **Reserved, write 0, read as 0** |

**GxEMUXCTR (x = 0 - 1)**
**External Multiplexer Control Register, Group x**
**(x * 0400$_H$ + 05F0$_H$)** **Reset Value: 0000 0000$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **EMUXSET** | [2:0] | rw | **External Multiplexer Start Selection**[1)]<br>Defines the initial setting for the external multiplexer. |
| **0** | [7:3] | r | **Reserved, write 0, read as 0** |
| **EMUXACT** | [10:8] | rh | **External Multiplexer Actual Selection**<br>Defines the current value for the external multiplexer selection. This bitfield is loaded from bitfield EMUXSET and modified according to the operating mode selected by bitfield EMUXMODE. |
| **0** | [15:11] | r | **Reserved, write 0, read as 0** |
| **EMUXCH** | [25:16] | rw | **External Multiplexer Channel Select**<br>Defines the channel(s) to which the external multiplexer control is applied.<br>**EMXCSS = 0: Channel number**,<br>the lower 5 bits select an arbitrary channel (valid numbers are limited by the number of available channels, unused bits shall be 0)<br>**EMXCSS = 1: Channel enable**,<br>each bit enables the associated channel (multiple channels can be selected/enabled) |
| **0** | [25:21] | r | **Reserved, write 0, read as 0** |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **EMUXMODE** | [27:26] | rw | **External Multiplexer Mode**<br>$00_B$ Software control (no hardware action)<br>$01_B$ Steady mode (use EMUXSET value)<br>$10_B$ Single-step mode[1)2)]<br>$11_B$ Sequence mode[1)] |
| **EMXCOD** | 28 | rw | **External Multiplexer Coding Scheme**<br>$0_B$ Output the channel number in binary code<br>$1_B$ Output the channel number in Gray code |
| **EMXST** | 29 | rw | **External Multiplexer Sample Time Control**<br>$0_B$ Use STCE whenever the setting changes<br>$1_B$ Use STCE for each conversion of an external channel |
| **EMXCSS** | 30 | rw | **External Multiplexer Channel Selection Style**<br>$0_B$ Channel number:<br>Bitfield EMUXCH selects an arbitrary channel<br>$1_B$ Channel enable:<br>Each bit of bitfield EMUXCH selects the associated channel for EMUX control |
| **EMXWC** | 31 | w | **Write Control for EMUX Configuration**<br>$0_B$ No write access to EMUX cfg.<br>$1_B$ Bitfields EMXMODE, EMXCOD, EMXST, EMXCSS can be written |

1) For single-step mode and sequence mode: Select the start value before selecting the respective mode.

2) Single-step mode modifies the EMUX channel number each time an EMUX-enabled channel is converted. Therefore, single-step mode works best with a single channel, because otherwise some external channels may be skipped.


Subject to Agreement on the Use of Product Information

Register EMUXSEL is a global register which assigns an arbitrary group to each of the EMUX interfaces.

**EMUXSEL**
**External Multiplexer Select Register**

**(03F0$_H$)** **Reset Value: 0000 0000$_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **EMUX GRP1** | | | | **EMUX GRP0** | | | |
| r | r | r | r | r | r | r | r | rw | | | | rw | | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **EMUXGRP0, EMUXGRP1** | [3:0], [7:4] | rw | **External Multiplexer Group for Interface x** Defines the group whose external multiplexer control signals are routed to EMUX interface x.[1)] |
| **0** | [31:8] | r | **Reserved, write 0, read as 0** |

1) The pins that are associated with each EMUX interface are listed in **Table 16-15 “Digital Connections in the XMC1300” on Page 16-156**.


Subject to Agreement on the Use of Product Information

The sigma-delta-loop control register LOOP configures the functionality of the sigma-delta-loop(s).

**SHS0_LOOP**
**Loop Control Register** **(4803 4050$_H$)** **Reset Value: 0000 0000$_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **LP EN1** | **0** | **0** | **0** | **0** | **0** | **0** | **LP SH1** | **0** | **0** | **0** | **LPCH1** | | | | |
| rw | r | r | r | r | r | r | rw | r | r | r | rw | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **LP EN0** | **0** | **0** | **0** | **0** | **0** | **0** | **LP SH0** | **0** | **0** | **0** | **LPCH0** | | | | |
| rw | r | r | r | r | r | r | rw | r | r | r | rw | | | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **LPCH0, LPCH1** | [4:0], [20:16] | rw | **Loop y Channel**<br>Selects the input channel, for which the sigma-delta-loop function shall be enabled. |
| **0** | [7:5] | r | **Reserved, write 0, read as 0** |
| **LPSH0, LPSH1** | 8, 24 | rw | **Loop y Sample&Hold Unit**<br>Selects the S&H unit, to which the indicated channel is assigned. |
| **0** | [14:9] | r | **Reserved, write 0, read as 0** |
| **LPEN0, LPEN1** | 15, 31 | rw | **Loop y Enable**<br>$0_H$ Off: standard operation<br>$1_H$ ON: sigma-delta-loop is active |
| **0** | [23:21] | r | **Reserved, write 0, read as 0** |
| **0** | [30:25] | r | **Reserved, write 0, read as 0** |

*Note: Bitfields LPSHy and LPCHy together select one individual input channel that is associated with the corresponding sigma-delta loop.*


Subject to Agreement on the Use of Product Information

## 16.16.9 Service Request Registers

**GxSEFLAG (x = 0 - 1)**
**Source Event Flag Register, Group x**

**($x * 0400_H + 0588_H$)** **Reset Value: 0000 0000$_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **SEV 1** | **SEV 0** |
| r | r | r | r | r | r | r | r | r | r | r | r | r | r | rwh | rwh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **SEV0, SEV1** | 0, 1 | rwh | **Source Event 0/1**<br>$0_B$ No source event<br>$1_B$ A source event has occurred |
| **0** | [31:2] | r | **Reserved, write 0, read as 0** |

*Note: Software can set all flags in register GxSEFLAG and trigger the corresponding event by writing 1 to the respective bit. Writing 0 has no effect.*
*Software can clear all flags in register GxSEFLAG by writing 1 to the respective bit in register GxSEFCLR.*

**GxCEFLAG (x = 0 - 1)**
**Channel Event Flag Register, Group x**

**($x * 0400_H + 0580_H$)** **Reset Value: 0000 0000$_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **CEV 7** | **CEV 6** | **CEV 5** | **CEV 4** | **CEV 3** | **CEV 2** | **CEV 1** | **CEV 0** |
| r | r | r | r | r | r | r | r | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **CEVy (y = 0 - 7)** | y | rwh | **Channel Event for Channel y**<br>$0_B$ No channel event<br>$1_B$ A channel event has occurred |
| **0** | [31:8] | r | **Reserved, write 0, read as 0** |

*Note: Software can set all flags in register GxCEFLAG and trigger the corresponding event by writing 1 to the respective bit. Writing 0 has no effect.*
*Software can clear all flags in register GxCEFLAG by writing 1 to the respective bit in register GxCEFCLR.*

**GxREFLAG (x = 0 - 1)**
**Result Event Flag Register, Group x**

**(x * $0400_H$ + $0584_H$)** **Reset Value: $0000\ 0000_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **REV 15** | **REV 14** | **REV 13** | **REV 12** | **REV 11** | **REV 10** | **REV 9** | **REV 8** | **REV 7** | **REV 6** | **REV 5** | **REV 4** | **REV 3** | **REV 2** | **REV 1** | **REV 0** |
| rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **REVy (y = 0 - 15)** | y | rwh | **Result Event for Result Register y**<br>$0_B$ No result event<br>$1_B$ New result was stored in register GxRESy |
| **0** | [31:16] | r | **Reserved, write 0, read as 0** |

*Note: Software can set all flags in register GxREFLAG and trigger the corresponding event by writing 1 to the respective bit. Writing 0 has no effect.*
*Software can clear all flags in register GxREFLAG by writing 1 to the respective bit in register GxREFCLR.*


Subject to Agreement on the Use of Product Information

**GxSEFCLR (x = 0 - 1)**
**Source Event Flag Clear Register, Group x**

**($x * 0400_H + 0598_H$)** **Reset Value: 0000 0000$_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **SEV 1** | **SEV 0** |
| r | r | r | r | r | r | r | r | r | r | r | r | r | r | w | w |

| Field | Bits | Type | Description |
|---|---|---|---|
| **SEV0, SEV1** | 0, 1 | w | **Clear Source Event 0/1**<br>$0_B$ No action<br>$1_B$ Clear the source event flag in GxSEFLAG |
| **0** | [31:2] | r | **Reserved, write 0, read as 0** |

**GxCEFCLR (x = 0 - 1)**
**Channel Event Flag Clear Register, Group x**

**($x * 0400_H + 0590_H$)** **Reset Value: 0000 0000$_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **CEV 7** | **CEV 6** | **CEV 5** | **CEV 4** | **CEV 3** | **CEV 2** | **CEV 1** | **CEV 0** |
| r | r | r | r | r | r | r | r | w | w | w | w | w | w | w | w |

| Field | Bits | Type | Description |
|---|---|---|---|
| **CEVy (y = 0 - 7)** | y | w | **Clear Channel Event for Channel y**<br>$0_B$ No action<br>$1_B$ Clear the channel event flag in GxCEFLAG |

| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | [31:8] | r | **Reserved, write 0, read as 0** |

**GxREFCLR (x = 0 - 1)**
**Result Event Flag Clear Register, Group x**
**(x * 0400$_H$ + 0594$_H$)** **Reset Value: 0000 0000$_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **REV 15** | **REV 14** | **REV 13** | **REV 12** | **REV 11** | **REV 10** | **REV 9** | **REV 8** | **REV 7** | **REV 6** | **REV 5** | **REV 4** | **REV 3** | **REV 2** | **REV 1** | **REV 0** |
| w | w | w | w | w | w | w | w | w | w | w | w | w | w | w | w |

| Field | Bits | Type | Description |
|---|---|---|---|
| **REVy (y = 0 - 15)** | y | w | **Clear Result Event for Result Register y**<br>$0_B$ No action<br>$1_B$ Clear the result event flag in GxREFLAG |
| **0** | [31:16] | r | **Reserved, write 0, read as 0** |

**GLOBEFLAG**
**Global Event Flag Register** **(00E0$_H$)** **Reset Value: 0000 0000$_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **0** | **0** | **0** | **0** | **0** | **0** | **REV GLB CLR** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **SEV GLB CLR** |
| r | r | r | r | r | r | r | w | r | r | r | r | r | r | r | w |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **0** | **0** | **0** | **0** | **0** | **0** | **REV GLB** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **SEV GLB** |
| r | r | r | r | r | r | r | rwh | r | r | r | r | r | r | r | rwh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **SEVGLB** | 0 | rwh | **Source Event (Background)**<br>$0_B$ No source event<br>$1_B$ A source event has occurred |
| **0** | [7:1] | r | **Reserved, write 0, read as 0** |
| **REVGLB** | 8 | rwh | **Global Result Event**<br>$0_B$ No result event<br>$1_B$ New result was stored in register GLOBRES |
| **0** | [15:9] | r | **Reserved, write 0, read as 0** |
| **SEVGLBCLR** | 16 | w | **Clear Source Event (Background)**<br>$0_B$ No action<br>$1_B$ Clear the source event flag SEVGLB |
| **0** | [23:17] | r | **Reserved, write 0, read as 0** |
| **REVGLBCLR** | 24 | w | **Clear Global Result Event**<br>$0_B$ No action<br>$1_B$ Clear the result event flag REVGLB |
| **0** | [31:25] | r | **Reserved, write 0, read as 0** |

*Note: Software can set flags REVGLB and SEVGLB and trigger the corresponding event by writing 1 to the respective bit. Writing 0 has no effect.*
*Software can clear these flags by writing 1 to bit REVGLBCLR and SECGLBCLR, respectively.*
*Setting both bits (e.g. SEVGLB and SEVGLBCLR) simultaneously clears the corresponding flag (e.g. SEVGLB).*

**GxSEVNP (x = 0 - 1)**
**Source Event Node Pointer Register, Group x**
**(x * $0400_H$ + $05C0_H$)** **Reset Value: $0000\ 0000_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **SEV1NP** | | | | **SEV0NP** | | | |
| r | r | r | r | r | r | r | r | rw | | | | rw | | | |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **SEV0NP, SEV1NP** | [3:0], [7:4] | rw | **Service Request Node Pointer Source Event i[1)]**<br>Routes the corresponding event trigger to one of the service request lines (nodes).<br>$0000_B$ Select service request line 0 of group x<br>…<br>$0011_B$ Select service request line 3 of group x<br>$0100_B$ Select shared service request line 0<br>…<br>$0111_B$ Select shared service request line 3<br>$1xxx_B$ Reserved<br>*Note: For shared service request lines see common groups in **Table 16-12**.* |
| **0** | [31:8] | r | **Reserved, write 0, read as 0** |

1) Source 0 is an 8-stage queued source, source 1 is a channel scan source.

**GxCEVNP0 (x = 0 - 1)**
**Channel Event Node Pointer Register 0, Group x**
**(x * $0400_H$ + $05A0_H$)** **Reset Value: $0000\ 0000_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CEV7NP | | | | CEV6NP | | | | CEV5NP | | | | CEV4NP | | | |
| rw | | | | rw | | | | rw | | | | rw | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CEV3NP | | | | CEV2NP | | | | CEV1NP | | | | CEV0NP | | | |
| rw | | | | rw | | | | rw | | | | rw | | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **CEV0NP, CEV1NP, CEV2NP, CEV3NP, CEV4NP, CEV5NP, CEV6NP, CEV7NP** | [3:0], [7:4], [11:8], [15:12], [19:16], [23:20], [27:24], [31:28] | rw | **Service Request Node Pointer Channel Event i**<br>Routes the corresponding event trigger to one of the service request lines (nodes).<br>$0000_B$ Select service request line 0 of group x<br>…<br>$0011_B$ Select service request line 3 of group x<br>$0100_B$ Select shared service request line 0<br>…<br>$0111_B$ Select shared service request line 3<br>$1xxx_B$ Reserved<br>*Note: For shared service request lines see common groups in **Table 16-12**.* |

**GxREVNP0 (x = 0 - 1)**
**Result Event Node Pointer Register 0, Group x**
**(x * $0400_H$ + $05B0_H$)** **Reset Value: $0000\ 0000_H$**

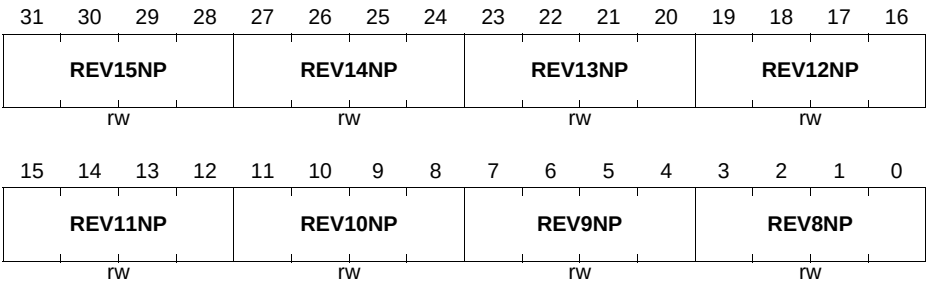
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **REV7NP** | | | | **REV6NP** | | | | **REV5NP** | | | | **REV4NP** | | | |
| rw | | | | rw | | | | rw | | | | rw | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **REV3NP** | | | | **REV2NP** | | | | **REV1NP** | | | | **REV0NP** | | | |
| rw | | | | rw | | | | rw | | | | rw | | | |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **REV0NP,**<br>**REV1NP,**<br>**REV2NP,**<br>**REV3NP,**<br>**REV4NP,**<br>**REV5NP,**<br>**REV6NP,**<br>**REV7NP** | [3:0],<br>[7:4],<br>[11:8],<br>[15:12],<br>[19:16],<br>[23:20],<br>[27:24],<br>[31:28] | rw | **Service Request Node Pointer Result Event i**<br>Routes the corresponding event trigger to one of the service request lines (nodes).<br>$0000_B$ Select service request line 0 of group x<br>…<br>$0011_B$ Select service request line 3 of group x<br>$0100_B$ Select shared service request line 0<br>…<br>$0111_B$ Select shared service request line 3<br>$1xxx_B$ Reserved<br>*Note: For shared service request lines see common groups in **Table 16-12**.* |

**GxREVNP1 (x = 0 - 1)**
**Result Event Node Pointer Register 1, Group x**
**(x * $0400_H$ + $05B4_H$)** **Reset Value: $0000\ 0000_H$**





Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **REV8NP,**<br>**REV9NP,**<br>**REV10NP,**<br>**REV11NP,**<br>**REV12NP,**<br>**REV13NP,**<br>**REV14NP,**<br>**REV15NP** | [3:0],<br>[7:4],<br>[11:8],<br>[15:12],<br>[19:16],<br>[23:20],<br>[27:24],<br>[31:28] | rw | **Service Request Node Pointer Result Event i**<br>Routes the corresponding event trigger to one of the service request lines (nodes).<br>$0000_B$ Select service request line 0 of group x<br>…<br>$0011_B$ Select service request line 3 of group x<br>$0100_B$ Select shared service request line 0<br>…<br>$0111_B$ Select shared service request line 3<br>$1xxx_B$ Reserved<br>*Note: For shared service request lines see common groups in **Table 16-12**.* |

**GLOBEVNP**
**Global Event Node Pointer Register**

**($0140_H$)** **Reset Value: $0000\ 0000_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **REV0NP** | | | |
| r | r | r | r | r | r | r | r | r | r | r | r | rw | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **SEV0NP** | | | |
| r | r | r | r | r | r | r | r | r | r | r | r | rw | | | |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **SEV0NP** | [3:0] | rw | **Service Request Node Pointer Backgr. Source**<br>Routes the corresponding event trigger to one of the service request lines (nodes).<br>$0000_B$ Select shared service request line 0 of common service request group 0<br>…<br>$0011_B$ Select shared service request line 3 of common service request group 0<br>others: Reserved<br>*Note: For shared service request lines see common groups in **Table 16-12**.* |
| **0** | [15:4] | r | **Reserved, write 0, read as 0** |
| **REV0NP** | [19:16] | rw | **Service Request Node Pointer Global Result**<br>Routes the corresponding event trigger to one of the service request lines (nodes).<br>$0000_B$ Select shared service request line 0 of common service request group 0<br>…<br>$0011_B$ Select shared service request line 3 of common service request group 0<br>others: Reserved<br>*Note: For shared service request lines see common groups in **Table 16-12**.* |
| **0** | [31:20] | r | **Reserved, write 0, read as 0** |


Subject to Agreement on the Use of Product Information

**GxSRACT (x = 0 - 1)**
**Service Request Software Activation Trigger, Group x**
**(x * $0400_H$ + $05C8_H$)** **Reset Value: $0000\ 0000_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **0** | **0** | **0** | **ASSR3** | **ASSR2** | **ASSR1** | **ASSR0** | **0** | **0** | **0** | **0** | **0** | **0** | **AGSR1** | **AGSR0** |
| r | r | r | r | w | w | w | w | r | r | r | r | r | r | w | w |

| Field | Bits | Type | Description |
|---|---|---|---|
| **AGSRy (y = 0 - 1)** | y | w | **Activate Group Service Request Node y**<br>$0_B$ No action<br>$1_B$ Activate the associated service request line |
| **0** | [7:2] | r | **Reserved, write 0, read as 0** |
| **ASSRy (y = 0 - 3)** | 8 + y | w | **Activate Shared Service Request Node y**<br>$0_B$ No action<br>$1_B$ Activate the associated service request line |
| **0** | [31:12] | r | **Reserved, write 0, read as 0** |


Subject to Agreement on the Use of Product Information

## 16.17 Interconnects

This section describes the actual implementation of the VADC module into the XMC1300, i.e. the incorporation into the microcontroller system.

### 16.17.1 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 (XMC1300).

Each converter group is equipped with a separate sample and hold stage and a dedicated analog input multiplexer.

**Table 16-12 General Converter Configuration in the XMC1300**

| 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 |


Subject to Agreement on the Use of Product Information

### Synchronization Groups in the XMC1300

The converter kernels in the XMC1300 can be connected to synchronization groups to achieve parallel conversion of several input channels.

**Table 16-13** summarizes which kernels can be synchronized for parallel conversions.

**Table 16-13 Synchronization Groups in the XMC1300**

| 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.

### 16.17.2 Analog Module Connections in the XMC1300

The VADC module accepts a number of analog input signals. The analog input multiplexers select the input channels to be converted from the signals available in this product.

The exact number of analog input channels and the available connection to port pins depend on the employed product type (see also **Table 16-12**). A summary of channels enclosing all versions of the XMC1300 can be found in **Table 16-14**.

**Table 16-14 Analog Connections in the XMC1300**

| Signal | Dir. | Source/Destin. | Description |
|---|---|---|---|
| **Supply Voltage and Reference Voltage** | | | |
| $V_{DDM}$ | 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 |
| **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 |


Subject to Agreement on the Use of Product Information

### 16.17.3 Digital Module Connections in the XMC1300

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 $0000_B$ … $1111_B$ select inputs with suffix -A … -P.*

**Table 16-15 Digital Connections in the** XMC1300

| 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 = $0011_B$) |
| 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 | Gating input I (GTSEL = $1000_B$) |
| GxREQGTJ | I (s) | Reserved | Gating input J (GTSEL = $1001_B$) |
| 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 = $0100_B$) |

**Table 16-15 Digital Connections in the** XMC1300 (cont'd)

| Signal | Dir. | Source/Destin. | Description |
|---|---|---|---|
| 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 | Gating input I (GTSEL = $1000_B$) |
| BGREQGTJ | I (s) | Reserved | Gating input J (GTSEL = $1001_B$) |
| 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 = $0000_B$) |
| 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.TRIGOUT0 | Trigger input F (XTSEL = $0101_B$) |
| G1REQTRF | I | BCCU0.TRIGOUT1 | 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 = $1110_B$) |


Subject to Agreement on the Use of Product Information

**Table 16-15 Digital Connections in the** XMC1300 (cont'd)

| Signal | Dir. | Source/Destin. | Description |
|---|---|---|---|
| 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 = $1110_B$) |
| 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 |


Subject to Agreement on the Use of Product Information

**Table 16-15 Digital Connections in the** XMC1300 (cont'd)

| Signal | Dir. | Source/Destin. | Description |
|---|---|---|---|
| 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<br>ERU0.OGU12 | Service request 2 of common block 0 |
| C0SR3 | O | ERU0.OGU22<br>ERU0.OGU32 | Service request 3 of common block 0 |
| EMUX00 | O | P0.4<br>P1.1 | Control of external analog multiplexer interface 0 |
| EMUX01 | O | P0.3<br>P1.2 | |
| EMUX02 | O | P0.2<br>P1.3 | |
| EMUX10 | O | P1.4 | Control of external analog multiplexer interface 1 |
| EMUX11 | O | P1.5 | |
| EMUX12 | O | P1.6 | |
| 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 |
| 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 |


Subject to Agreement on the Use of Product Information

**Table 16-15 Digital Connections in the** XMC1300 (cont'd)

| Signal | Dir. | Source/Destin. | Description |
|---|---|---|---|
| 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.


Subject to Agreement on the Use of Product Information

# 17 Analog Comparator (ACMP) and Out of Range Comparator (ORC)

This chapter describes the controls for analog comparator (ACMP) and out of range comparator (ORC).

## 17.1 Overview

This section gives an overview about the feature set of the Analog comparator and the Out of Range Comparator. In XMC1300, there are up to 3 Analog Comparators (ACMPx [x=0-2]) and 8 Out of Range Comparator (ORCx [x=0-7]).

### 17.1.1 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 ouput option

## 17.2 Analog Comparator (ACMP)

**Figure 17-2** shows the block diagram of a Analog Comparator unit.


Subject to Agreement on the Use of Product Information

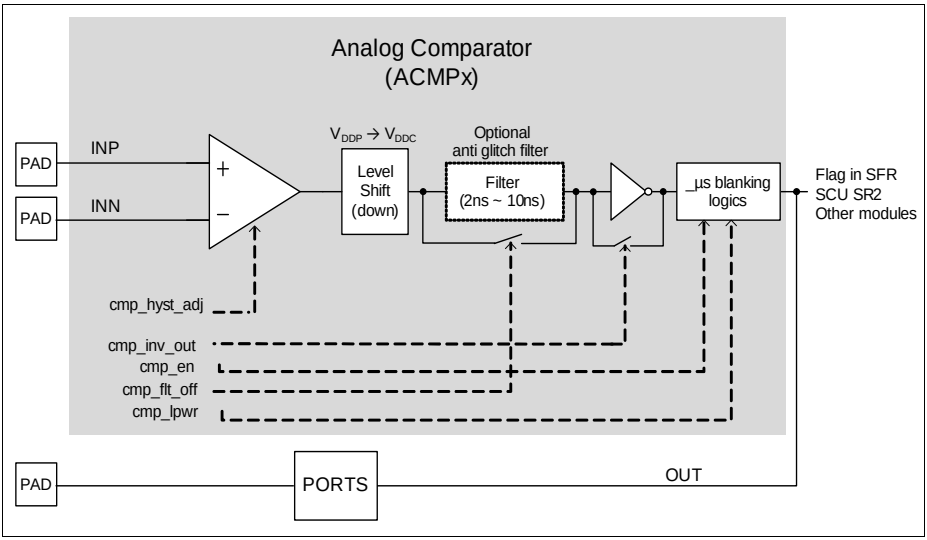



**Figure 17-1 Block diagram of Analog Comparator**

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 500kOhm) divides the reference voltage value from ACMP.REF input if bit ANACMP1.REF_DIV_EN is set. According to **Figure 17-2**, the divided voltage is delivered to pin ACMP1.INP. With bit ANACMP0.ACMP0_SEL and ANACMP2.ACMP2_SEL, all other comparators can be supplied by this divided reference voltage.

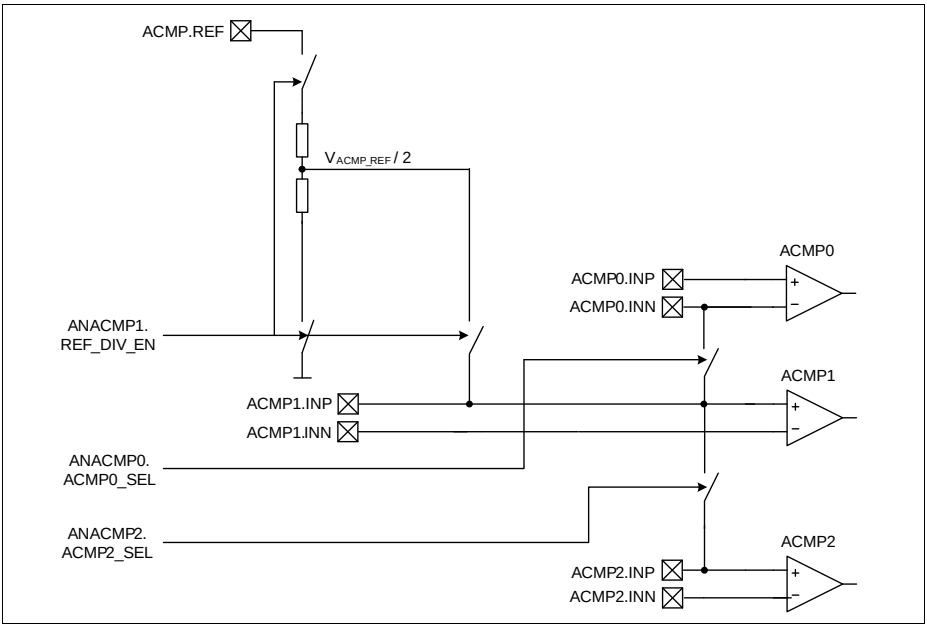



**Figure 17-2 Analog Comparator Reference Divider Function**

### Low Power Mode

A low power state helps to reduce the total power consumption e.g. 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.

## 17.3 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 17-3**).

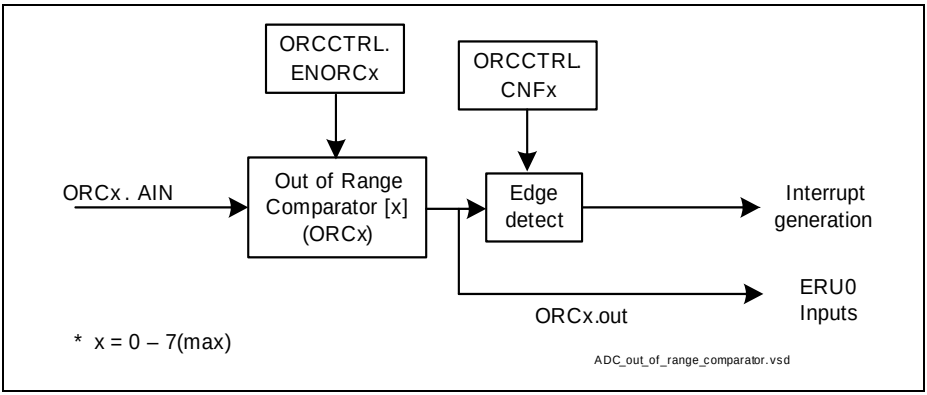



**Figure 17-3 Out of range comparator**

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 XMC1300, ORCx.AIN[x=0-7] are connected to P2.2 - P2.9 respectively.

## 17.4 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.

## 17.5 Debug Behavior

ACMPx and ORCx are not affected by the debug activities performed using external debug probe.

## 17.6 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


Subject to Agreement on the Use of Product Information

**Table 17-1 Registers Address Space**

| Module | Base Address | End Address | Note |
|---|---|---|---|
| COMPARATOR | $4001\ 0000_H$ | $4001\ FFFF_H$ | System Control Unit Registers |

**Table 17-2 Registers Overview**

| Short Name | Description | Offset Addr.[1] | Access Mode | | Description See |
|---|---|---|---|---|---|
| | | | Read | Write | |
| **ORCx Registers** | | | | | |
| ORCCTRL | Out of Range Comparator Control Register | $0500_H$ | U, PV 32 | U, PV 32 | Page 17-6 |
| **ACMPx Registers** | | | | | |
| ANACMP0 | Analog Comparator 0 Control Register | $105C_H$ | U, PV | U, PV | Page 17-7 |
| ANACMP1 | Analog Comparator 1 Control Register | $1060_H$ | U, PV | U, PV | Page 17-7 |
| ANACMP2 | Analog Comparator 2 Control Register | $1064_H$ | U, PV | U, PV | Page 17-7 |

1) The absolute register address is calculated as follows:
Module Base Address + Offset Address (shown in this column)

## 17.6.1 ORC Register

**ORCCTRL**

Out of Range Comparator Control Register

This register enables the out of range comparator and selects the rising edge trigger or falling edge trigger for the flag register.

**ORCCTRL**
**Out Of Range Comparator Control Register**
**($0500_H$)** **Reset Value: 0000 $0000_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | CNF7 | CNF6 | CNF5 | CNF4 | CNF3 | CNF2 | CNF1 | CNF0 |
| r | | | | | | | | rw | rw | rw | rw | rw | rw | rw | rw |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | ENORC7 | ENORC6 | ENORC5 | ENORC4 | ENORC3 | ENORC2 | ENORC1 | ENORC0 |
| r | | | | | | | | rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **ENORCx (x = 0 - 7)** | x | rw | **Enable Out of Range Comparator x**<br>This bit defines if the out of range comparator is enabled in the corresponding analog input channel.<br>$0_B$ Out of range comparator disabled.<br>$1_B$ Out of range comparator enabled. |
| **CNFx (x = 0 - 7)** | x+16 | rw | **Out of Range Comparator Flag x**<br>This bit selects CHx rising edge trigger or falling edge trigger for out of range comparator flag register.<br>$0_B$ Falling edge trigger out of range event register.<br>$1_B$ Rising edge trigger out of range event register. |
| **0** | [15:8], [31:24] | r | **Reserved**<br>Read as 0; should be written with 0. |

## 17.6.2 ACMP Registers

**ANACMP0**

Analog Comparator 0 Control Register

**ANACMP0**

**Analog Comparator 0 Control Register(105C$_H$)** **Reset Value: 0020$_H$**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CMP_OUT | 0 | | | | | | CMP_LPWR | 0 | ACMP0_SEL | CMP_HYST_ADJ | | CMP_INV_OUT | 0 | CMP_FLT_OFF | CMP_EN |
| rh | r | | | | | | rw | r | rw | rw | | rw | r | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **CMP_EN** | 0 | rw | **Comparator enable**<br>0B **CMP_DIS**, Comparator is disabled<br>1B **CMP_EN**, Comparator is enabled |
| **CMP_FLT_OFF** | 1 | rw | **Disables comparator filter**<br>If set, the comparator glitch-filter is switched off<br>0B **FIL_ON**, filter is active<br>1B **FIL_OFF**, filter is switched off (to prevent a filter delay) |
| **CMP_INV_OUT** | 3 | rw | **Inverted Comparator output**<br>The comparator output is simply inverted<br>0B **INV_OFF**, no inversion of comparator signal<br>1B **INV_ON**, comparator signal is inverted |
| **CMP_HYST_ADJ** | 5:4 | rw | **Comparator hysteresis adjust**<br>To reduce noise sensitivity a hysteresis voltage can be chosen. It can be switched off with writing<br>00B **HYS_OFF**, Comparator hysteresis is switched off<br>01B **HYS1**, Hysteresis_typ = 10mV<br>10B **HYS2**, Hysteresis_typ = 15mV<br>11B **HYS3**, Hysteresis_typ = 20mV |

| Field | Bits | Type | Description |
|---|---|---|---|
| **ACMP0_SEL** | 6 | rw | **Connect ACMP0.INN to ACMP1.INP**<br>An internal switch connects comparator pad ACMP0.INN with pad ACMP1.INP together. Only one of both input pad shall be driven by a voltage source. The time delay between both pads is caused by the impedance of the switch. When bit ANACMP1.REF_DIV_EN is set, the divided reference voltage is applied not ony to ACMP1.INP, but also to ACMP0.INN<br>0B **OFF**, ACMP0.INN is not connected<br>1B **ON**, ACMP0.INN is connected to ACMP1.INP |
| **CMP_LPWR** | 8 | rw | **Low Power Mode**<br>If enabled, all three analog comparator units are set into low power mode. If the logic level of this bit is changed, the core has to blank the comparator output information a certain time.<br>0B **HPM**, High Power Mode<br>1B **LPM**, Low Power Mode |
| **CMP_OUT** | 15 | rh | **Comparator output monitor bit**<br>This bit corresponds to the comparator output status<br>0B **OUT0**, state “Vminus > Vplus”<br>1B **OUT1**, state “Vminus < Vplus” |
| **0** | 2, 7, 14:9 | r | **Reserved**<br>Read as 0; should be written with 0. |

### ANACMP1

Analog Comparator 1 Control Register

### ANACMP1

**Analog Comparator 1 Control Register($1060_H$)** **Reset Value: $0020_H$**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **CMP_OUT** | **0** | | | | | | | | **REF_DIV_EN** | **CMP_HYST_ADJ** | | **CMP_INV_OUT** | **0** | **CMP_FLT_OFF** | **CMP_EN** |
| rh | r | | | | | | | | rw | rw | | rw | r | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **CMP_EN** | 0 | rw | **Comparator enable**<br>0B **CMP_DIS**, Comparator is disabled<br>1B **CMP_EN**, Comparator is enabled |
| **CMP_FLT_OFF** | 1 | rw | **Disables comparator filter**<br>If set, the comparator glitch-filter is switched off<br>0B **FIL_ON**, filter is active<br>1B **FIL_OFF**, filter is switched off (to prevent a filter delay) |
| **CMP_INV_OUT** | 3 | rw | **Inverted Comparator output**<br>The comparator output is simply inverted<br>0B **INV_OFF**, no inversion of comparator signal<br>1B **INV_ON**, comparator signal is inverted |
| **CMP_HYST_ADJ** | 5:4 | rw | **Comparator hysteresis adjust**<br>To reduce noise sensitivity a hysteresis voltage can be chosen. It can be switched off with writing<br>00B **HYS_OFF**, Comparator hysteresis is switched off<br>01B **HYS1**, Hysteresis_typ = 10mV<br>10B **HYS2**, Hysteresis_typ = 15mV<br>11B **HYS3**, Hysteresis_typ = 20mV |
| **REF_DIV_EN** | 6 | rw | **Resistor Divider is enabled and Reference Voltage is applied to ACMP1**<br>The divider reference voltage is applied to the positive ACMP1 input<br>0B **OFF**, no resistor is connected<br>1B **ON**, the divider resistor is enabled and the voltage is applied to ACMP1.INP |
| **CMP_OUT** | 15 | rh | **Comparator output monitor bit**<br>This bit corresponds to the comparator output status<br>0B **OUT0**, state “Vminus > Vplus”<br>1B **OUT1**, state “Vminus < Vplus” |
| **0** | 2, 14:7 | r | **Reserved**<br>Read as 0; should be written with 0. |


Subject to Agreement on the Use of Product Information

### ANACMP2

Analog Comparator 2 Control Register

### ANACMP2

**Analog Comparator 2 Control Register(1064$_H$)** **Reset Value: 0020$_H$**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CMP_OUT | 0 | | | | | | | | ACMP2_SEL | CMP_HYST_ADJ | | CMP_INV_OUT | 0 | CMP_FLT_OFF | CMP_EN |
| rh | r | | | | | | | | rw | rw | | rw | r | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **CMP_EN** | 0 | rw | **Comparator enable**<br>0B **CMP_DIS**, Comparator is disabled<br>1B **CMP_EN**, Comparator is enabled |
| **CMP_FLT_OFF** | 1 | rw | **Disables comparator filter**<br>If set, the comparator glitch-filter is switched off<br>0B **FIL_ON**, filter is active<br>1B **FIL_OFF**, filter is switched off (to prevent a filter delay) |
| **CMP_INV_OUT** | 3 | rw | **Inverted Comparator output**<br>The comparator output is simply inverted<br>0B **INV_OFF**, no inversion of comparator signal<br>1B **INV_ON**, comparator signal is inverted |
| **CMP_HYST_ADJ** | 5:4 | rw | **Comparator hysteresis adjust**<br>To reduce noise sensitivity a hysteresis voltage can be chosen. It can be switched off with writing<br>00B **HYS_OFF**, Comparator hysteresis is switched off<br>01B **HYS1**, Hysteresis_typ = 10mV<br>10B **HYS2**, Hysteresis_typ = 15mV<br>11B **HYS3**, Hysteresis_typ = 20mV |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **ACMP2_SEL** | 6 | rw | **Connect ACMP2.INP to ACMP1.INP**<br>An internal switch connects comparator pad ACMP2.INP with pad ACMP1.INP together. Only one of both input pad shall be driven by a voltage source. The time delay between both pads is caused by the impedance of the switch. When bit ANACMP1.REF_DIV_EN is set, the divided reference voltage is applied not ony to ACMP1.INP, but also to ACMP2.INP<br>0B **OFF**, ACMP2.INP is not connected<br>1B **ON**, ACMP2.INP is connected to ACMP1.INP |
| **CMP_OUT** | 15 | rh | **Comparator output monitor bit**<br>This bit corresponds to the comparator output status<br>0B **OUT0**, state “Vminus > Vplus”<br>1B **OUT1**, state “Vminus < Vplus” |
| **0** | 2, 14:7 | r | **Reserved**<br>Read as 0; should be written with 0. |


Subject to Agreement on the Use of Product Information

## 17.7 Interconnects

ACMP and ORC are connected to the ports, the ERU0 and BCCU0.

**Table 17-3 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 |
| ACMP.REF | I | P2.11 | Reference input of ACMP |
| ACMP0.OUT | O | P0.10<br>P2.10<br>ERU0.0A0<br>BCCU0.IN5 | output of ACMP0 |
| ACMP1.OUT | O | P1.0<br>ERU0.1A0<br>BCCU0.IN0<br>P2.5.HW0 pull control | output of ACMP1 |
| ACMP2.OUT | O | P0.5<br>P1.2<br>ERU0.2A0<br>BCCU0.IN3<br>P2.3.HW0 pull control | output of ACMP2 |

**Table 17-4 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 |


Subject to Agreement on the Use of Product Information

**Table 17-4 Out of Range Comparator Pin Connections** (cont'd)

| Input/Output | I/O | Connected To | Description |
|---|---|---|---|
| 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<br>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 |


Subject to Agreement on the Use of Product Information

# 18 Temperature Sensor (TSE)

This chapter describes the controls for Temperature Sensor (TSE).

## 18.1 General Description

The Temperature Sensor (TSE) 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 TSE 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, TSE 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 TSE is capable of generating interrupt requests when TSE 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.

## 18.2 Service Request Generation

The TSE has a service request output for 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.


Subject to Agreement on the Use of Product Information

## 18.3 Registers

TSE registers are accessible via the APB bus. The absolute register address is calculated by adding:

Module Base Address + Offset Address

Following access to TSE SFRs result in an AHB/APB error response:

- Read or write access to undefined address
- Write access to read-only registers

**Table 18-1 Registers Address Space**

| Module | Base Address | End Address | Note |
|---|---|---|---|
| TSE | $4001\ 0000_H$ | $4001\ FFFF_H$ | System Control Unit Registers |

**Table 18-2 Registers Overview**

| Short Name | Description | Offset Addr.[1] | Access Mode | | Description See |
|---|---|---|---|---|---|
| | | | Read | Write | |
| ANATSECTRL | Temperature Sensor Control Register | $0024_H$ | U, PV | U, PV | Page 18-3 |
| ANATSEIH | Temperature Sensor High Interrupt Register | $0030_H$ | U, PV | U, PV | Page 18-3 |
| ANATSEIL | Temperature Sensor Low Interrupt Register | $0034_H$ | U, PV | U, PV | Page 18-4 |
| ANATSEMON | Temperature Sensor Counter2 Monitor Register | $0040_H$ | U, PV | U, PV | Page 18-4 |

1) The absolute register address is calculated as follows:
Module Base Address + Offset Address (shown in this column)

### 18.3.1 Registers

#### ANATSECTRL

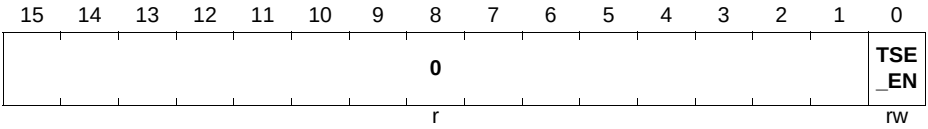
Temperature Sensor Control Register


Subject to Agreement on the Use of Product Information

### ANATSECTRL

**Temperature Sensor Control Register**

**(1024$_H$)** **Reset Value: 0000$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **TSE_EN** | 0 | rw | **Temperature sensor enable**<br>0B Temperature sensor is disabled<br>1B Temperature sensor is switched on |
| **0** | 15:1 | r | **Reserved**<br>Read as 0; should be written with 0. |

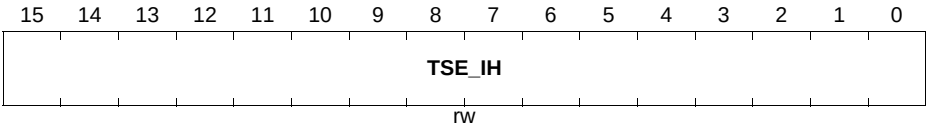
### ANATSEIH

Temperature Sensor High Temperature Interrupt Register

### ANATSEIH

**Temperature Sensor High Temperature Interrupt Register**

**(1030$_H$)** **Reset Value: 0000$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **TSE_IH** | 15:0 | rw | **Counter value for high temperature interrupt**<br>ANATSEIH value is compared with ANATSEMON (with the counter value)<br>An high temperature interrupt is triggered if: ANATSE_MON < ANATSEIH<br><br>The comparison result can be observed from SCU_SRRAW.TSE_HIGH |


Subject to Agreement on the Use of Product Information

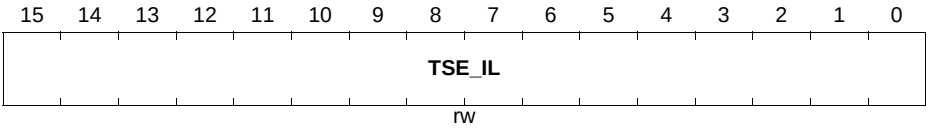
### ANATSEIL

Temperature Sensor Low Temperature Interrupt Register

### ANATSEIL

**Temperature Sensor Low Temperature Interrupt Register**

**($1034_H$)** **Reset Value: $FFFF_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **TSE_IL** | 15:0 | rw | **Counter value for low temperature interrupt**<br>ANATSEIL value is compared with ANATSEMON<br>An low interrupt is triggered if: ANATSEMON > ANATSEIL<br><br>The comparison result can be observed from SCU_SRRAW.TSE_LOW |

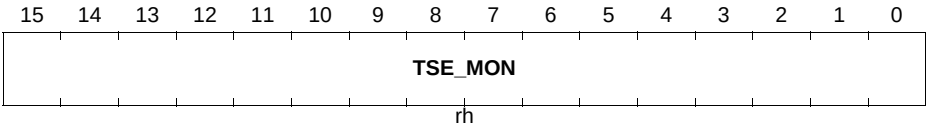
### ANATSEMON

Temperature Sensor Counter2 Monitor Register

### ANATSEMON

**Temperature Sensor Counter2 Monitor Register**

**($1040_H$)** **Reset Value: $0000_H$**





Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **TSE_MON** | 15:0 | rh | **Monitor Counter2 value; loaded by TSE_DONE**<br>After each measurement done event (indicated by a set in SCU_SRRAW.TSE_DONE bit), the result is stored in this register. |


Subject to Agreement on the Use of Product Information

# Industrial Control Peripherals


Subject to Agreement on the Use of Product Information

# 19 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 19-1 Abbreviations table**

| | |
|---|---|
| PWM | Pulse Width Modulation |
| CCU4x | Capture/Compare Unit 4 module instance x |
| CC4y | Capture/Compare Unit 4 Timer Slice instance y |
| ADC | Analog to Digital Converter |
| POSIF | Position Interface peripheral |
| SCU | System Control Unit |
| $f_{ccu4}$ | CCU4 module clock frequency |
| $f_{tclk}$ | CC4y timer clock frequency |

*Note: A small “y” or “x” letter in a register indicates an index*

## 19.1 Overview

Each CCU4 module is comprised of four identical 16 bit Capture/Compare Timer slices, CC4y. Each timer slice can work in compare mode or in capture mode. In compare mode one compare channel is available while in capture mode, up to four capture registers can be used in parallel.

Each CCU4 module has four service request lines and each timer slice contains a dedicated output signal, enabling the generation of up to four independent PWM signals.

Straightforward timer slice concatenation is also possible, enabling up to 64 bit timing operations. This offers a flexible frequency measurement, frequency multiplication and pulse width modulation scheme.

A programmable function input selector for each timer slice, that offers up to nine functions, discards the need of complete resource mapping due to input ports availability.

A built-in link between the CCU4 and several other modules enable flexible digital motor control loops implementation, e.g. with Hall Sensor monitoring or direct coupling with Encoders.


Subject to Agreement on the Use of Product Information

### 19.1.1 Features

**CCU4 module features**

Each CCU4 represents a combination of four timer slices, that can work independently in compare or capture mode. Each timer slice has a dedicated output for PWM signal generation.

All four CCU4 timer slices, CC4y, are identical in terms of available functions and operating modes. Avoiding this way the need of implementing different software routines, depending on which resource of CCU4 is used.

A built-in link between the four timer slices is also available, enabling this way a simplified timer concatenation and sequential operations.

General Features

- 16 bit timer cells
- capture and compare mode for each timer slice
  - four capture registers in capture mode
  - one compare channel in compare mode
- programmable low pass filter for the inputs
- built-in timer concatenation
  - 32, 48 or 64 bit width
- shadow transfer for the period and compare values
- programmable clock prescaler
- normal timer mode
- gated timer mode
- three counting schemes
  - center aligned
  - edge aligned
  - single shot
- PWM generation
- TRAP function
- start/stop can be controlled by external events
- counting external events
- four dedicated service request lines per CCU4

Additional features

- external modulation function
- load controlled by external events
- dithering PWM
- floating point pre scaler
- output state override by an external event
- suitable and flexible connectivity to several modules:
  - motor and power conversion applications
  - high number of signal conditioning possibilities


Subject to Agreement on the Use of Product Information

### CCU4 features vs. applications

On **Table 19-2** a summary of the major features of the CCU4 unit mapped with the most common applications.

**Table 19-2 Applications summary**

| Feature | Applications |
|---|---|
| Four independent timer cells | Independent PWM generation:<br>• Multiple buck/boost converter control (with independent frequencies)<br>• Different modes of operation for each timer, increasing the resource optimization<br>• Up to 2 Half-Bridges control<br>• 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:<br>• High dynamic trigger capturing<br>• High dynamic signal measurement |
| Dithering PWM | Generating a fractional PWM frequency or duty cycle:<br>• To avoid big steps on frequency or duty cycle adjustment in slow control loop applications<br>• Increase the PWM signal resolution over time |
| Floating prescaler | Automated control signal measurement:<br>• decrease SW activity for monitoring signals with high or unknown dynamics<br>• emulating more than a 16 bit timer for system control |
| Up to 9 functions via external signals for each timer | Flexible resource optimization:<br>• The complete set of external functions is always available<br>• Several arrangements can be done inside a CCU4, e.g., one timer working in capture mode and one working in compare |


Subject to Agreement on the Use of Product Information

**Table 19-2 Applications summary** (cont'd)

| Feature | Applications |
|---|---|
| 4 dedicated service request lines | Specially developed for:<br>• generating interrupts for the microprocessor<br>• flexible connectivity between peripherals, e.g. ADC triggering. |
| Linking with other modules | Flexible profiles for:<br>• Hall Sensor feedback/monitoring<br>• Motor Encoders feedback/monitoring<br>• PWM parallel modulation<br>• Flexible signal conditioning |

### 19.1.2 Block Diagram

Each CCU4 timer slice can operate independently from the other slices for all the available modes. Each timer slice contains a dedicated input selector for functions linked with external events and has a dedicated compare output signal, for PWM signal generation.

The built-in timer concatenation is only possible with adjacent slices, e.g. CC40/CC41. Combinations for slice concatenations like, CC40/CC42 or CC40/CC43 are not possible.

The individual service requests for each timer slice (four per slice) are multiplexed into four module service requests lines, **Figure 19-1**.


Subject to Agreement on the Use of Product Information

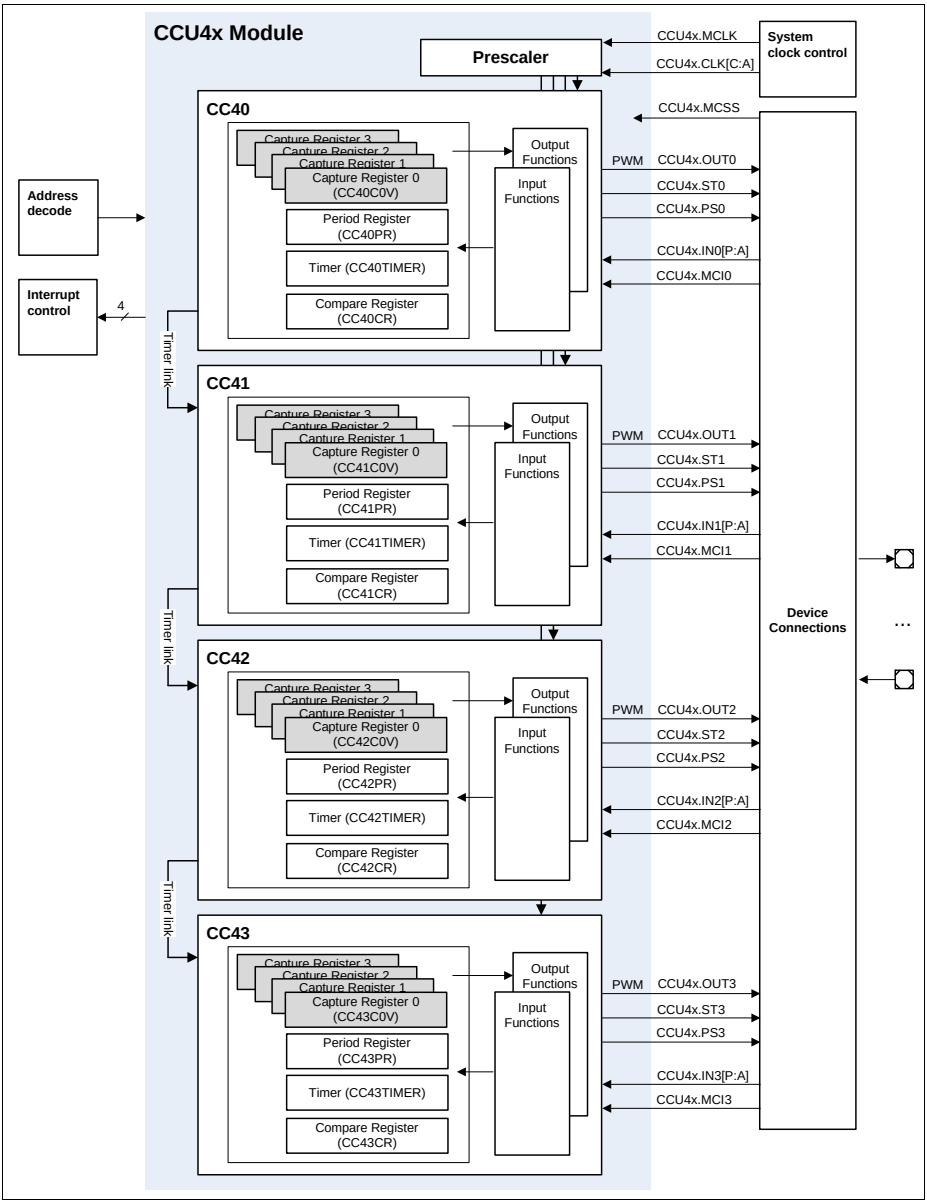



**Figure 19-1 CCU4 block diagram**

## 19.2 Functional Description

### 19.2.1 CC4y Overview

The input path of a CCU4 slice is comprised of a selector (**Section 19.2.2**) and a connection matrix unit (**Section 19.2.3**). The output path contains a service request control unit, a timer concatenation unit and two units that control directly the state of the output signal for each specific slice (for TRAP and modulation handling), see **Figure 19-2**.

The timer core is built of a 16 bit counter one period and one compare register in compare mode, or up to four capture registers in capture mode.

In compare mode the period register sets the maximum counting value while the compare channel is controlling the ACTIVE/PASSIVE state of the dedicated comparison slice output.

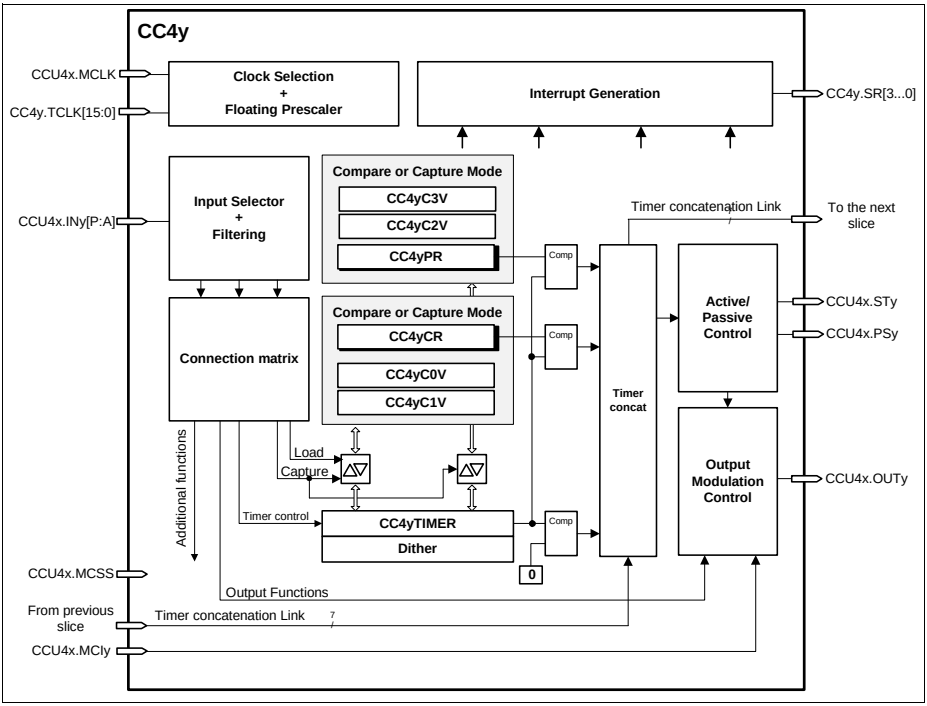



**Figure 19-2 CCU4 slice block diagram**

Each CCU4 slice, with the exception of the first, contains six dedicated inputs outputs that are used for the built-in timer concatenation functionality.

Inputs and outputs that are not seen at the CCU4 boundaries have a nomenclature of CC4y.<name>, whilst CCU4 module inputs and outputs are described as CCU4x.<signal_name>y (indicating the variable y the object slice).

**Table 19-3 CCU4 slice pin description**

| Pin | I/O | Description |
|---|---|---|
| CCU4x.MCLK | I | Module clock |
| CC4y.TCLK[15:0] | I | Clocks from the pre scaler |
| CCU4x.INy[P:A] | I | Slice functional inputs (used to control the functionality throughout slice external events) |
| CCU4x.MCIy | I | Multi Channel mode input |
| CCU4x.MCSS | I | Multi Channel shadow transfer trigger |
| CC4y.SR[3...0] | O | Slice service request lines |
| CC4x.STy | O | Slice comparison status value |
| CCU4x.PSy | O | Multi channel pattern update trigger |
| CCU4x.OUTy | O | Slice dedicated output pin |

*Note:*

4. *The status bit outputs of the Kernel, CCU4x.STy, are extended for one more kernel clock cycle.*
5. *The Service Request signals at the output of the kernel are extended for one more kernel clock cycle.*
6. *The maximum output signal frequency of the CCU4x.STy outputs is module clock divided by 4.*

The slice timer, can count up or down depending on the selected operating mode. A direction flag holds the actual counting direction.

The timer is connected to two stand alone comparators, one for the period match and one for a compare match. The registers used for period match and comparison match can be programmed to serve as capture registers enabling sequential capture capabilities on external events.

In normal edge aligned counting scheme, the counter is cleared to $0000_H$ each time that matches the period value defined in the period register. In center aligned mode, the counter direction changes from ‘up counting’ to ‘down counting’ after reaching the period


Subject to Agreement on the Use of Product Information

value. Both period and compare registers have an aggregated shadow register, which enables the update of the PWM period and duty cycle on the fly.

A single shot mode is also available, where the counter stops after it reaches the value set in the period register.

The start and stop of the counter can be controlled via software access or by a programmable input pin.

Functions like, load, counting direction (up/down), TRAP and output modulation can also be controlled with external events, see **Section 19.2.3**.

### 19.2.2 Input Selector

The first unit of the slice input path, is used to select which inputs are used to control the available external functions.

Inside this block the user also has the possibility to perform a low pass filtering of the signals and selecting the active edge(s) or level of the external event, see **Figure 19-3**.

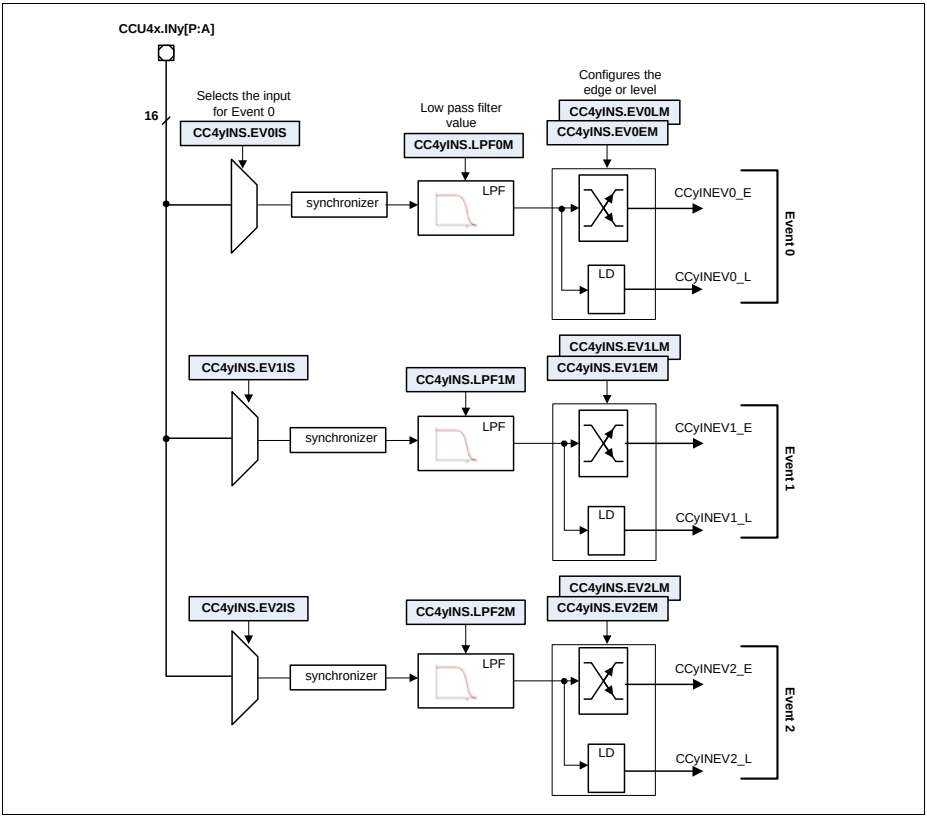



**Figure 19-3 Slice input selector diagram**

The user has the possibility of selecting any of the CCU4x.INy[P:A] inputs as the source of an event.

At the output of this unit we have a user selection of three events, that were configured to be active at rising, falling or both edges, or level active. These selected events can then be mapped to several functions.

Notice that each decoded event contains two outputs, one edge active and one level active, due to the fact that some functions like counting, capture or load are edge sensitive events while, timer gating or up down counting selection are level active.

### 19.2.3 Connection Matrix

The connection matrix maps the events coming from the input selector to several user configured functions, **Figure 19-4**.The following functions can be enabled on the connection matrix:

**Table 19-4 Connection matrix available functions**

| Function | Brief description | Map to figure Figure 19-4 |
|---|---|---|
| Start | Edge signal to start the timer | CCystrt |
| Stop | Edge signal to stop the timer | CCystp |
| Count | Edge signal used for counting events | CCycnt |
| Up/down | Level signal used to select up or down counting direction | CCyupd |
| Capture 0 | Edge signal that triggers a capture into the capture registers 0 and 1 | CCycapt0 |
| Capture 1 | Edge signal that triggers a capture into the capture 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<br>CCyoset for the trigger |

Inside the connection matrix we also have a unit that performs the built-in timer concatenation. This concatenation enables a completely synchronized operation between the concatenated slices for timing operations and also for capture and load actions. The timer slice concatenation is done via the **CC4yCMC**.TCE bitfield. For a complete description of the concatenation function, please address **Section 19.2.9**.


Subject to Agreement on the Use of Product Information

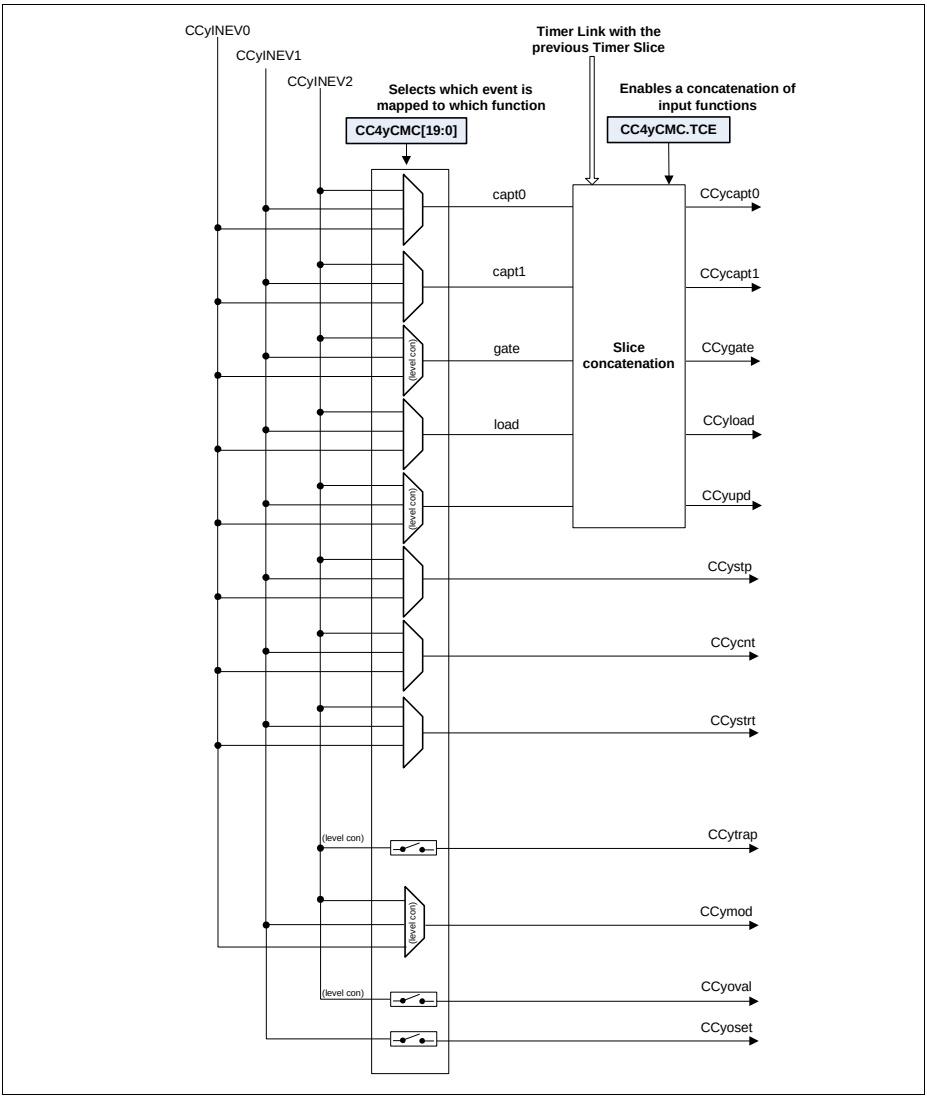



**Figure 19-4 Slice connection matrix diagram**


Subject to Agreement on the Use of Product Information

### 19.2.4 Starting/Stopping the Timer

Each timer slice contains a run bit register that indicates the actual status of the timer, **CC4yTCST**.TRB. The start and stop of the timer can be done via software access or can be controlled directly by external events, see **Figure 19-5**.

Selecting an external signal that acts as a start trigger does not force the user to use an external stop trigger and vice versa.

Selecting the single shot mode, imposes that after the counter reaches the period value the run bit, **CC4yTCST**.TRB, is going to be cleared and therefore the timer is stopped.

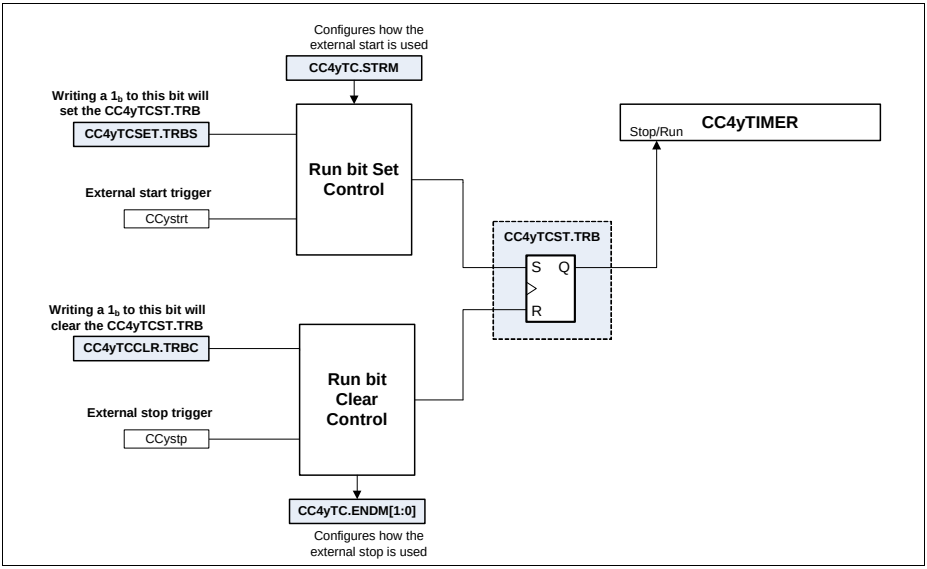



**Figure 19-5 Timer start/stop control diagram**

One can use the external stop signal to perform the following functions (configuration via **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).


Subject to Agreement on the Use of Product Information

To start multiple CCU4 timers at the same time/synchronously one should use a dedicated input as external start (see **Section 19.2.7.1** for a description how to configure an input as start function). This input should be connected to all the Timers that need to started synchronously (see **Section 19.8** for a complete list of module connections), **Figure 19-6**.

For starting the timers synchronously via software there is a dedicated input signal, controlled by the SCU (System Control Unit), that is connected to all the CCU4 timers. This signal should then be configured as an external start signal (see **Section 19.2.7.1**) and then the software must write a $1_B/0_B$ (depending on the external start signal configuration) to the specific bitfield of the CCUCON register (this register is described on the SCU chapter).

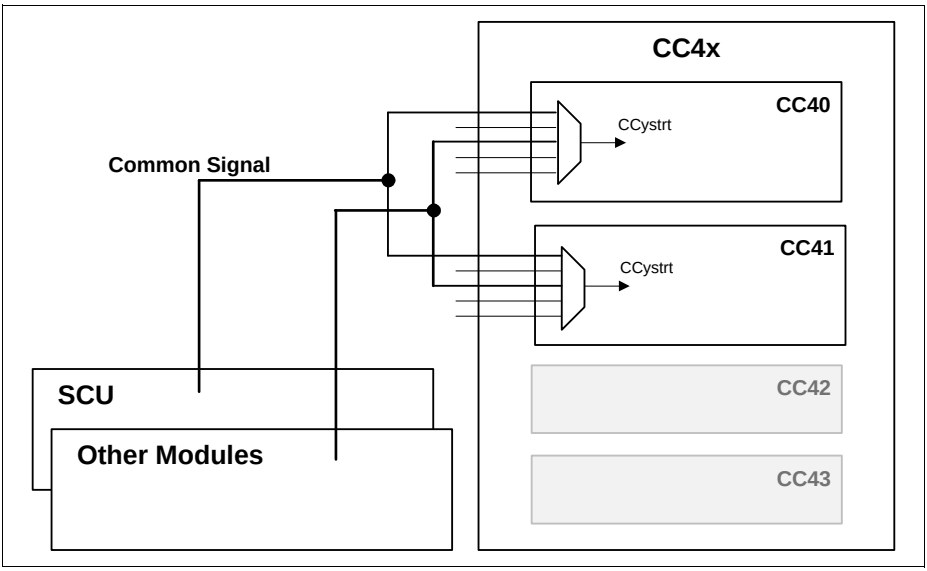



**Figure 19-6 Starting multiple timers synchronously**

## 19.2.5 Counting Modes

Each CC4y timer slice can be programmed into three different counting schemes:

- Edge aligned (default)
- Center aligned
- Single shot (can be edge or center aligned)

These three counting schemes can be used as stand alone without the need of selecting any inputs as external event sources. Nevertheless it is also possible to control the

counting operation via external events like, timer gating, counting trigger, external stop, external start, etc.

For all the counting modes, it is possible to update on the fly the values for the timer period and compare channel. This enables a cycle by cycle update of the PWM frequency and duty cycle.

The compare channel of each CC4y Timer Slice, has an associated Status Bit (**GCST**.CC4yST), that indicates the active or passive state of the channel, **Figure 19-7**. The set and clear of the status bit and the respective PWM signal generation is dictated by the timer period, compare value and the current counting mode. See the different counting mode descriptions, **Section 19.2.5.3** to **Section 19.2.5.5** to understand how this bit is set and cleared.

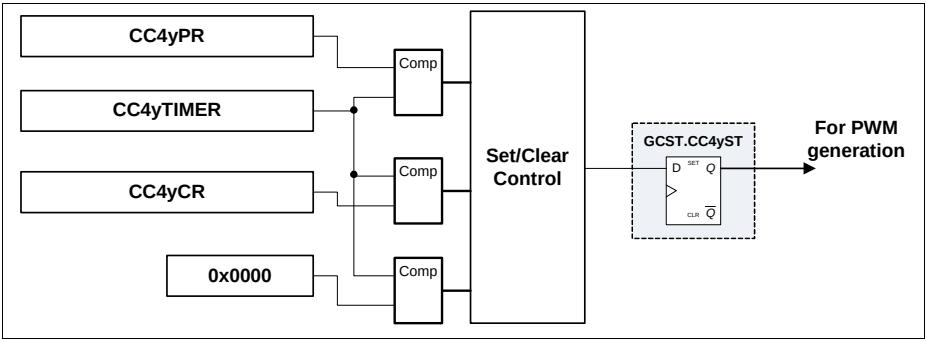



**Figure 19-7 CC4y Status Bit**

### 19.2.5.1 Calculating the PWM Period and Duty Cycle

The period of the timer is determined by the value in the period register, **CC4yPR** and by the timer mode.

The base for the PWM signal frequency and duty cycle, is always related to the clock frequency of the timer itself and not to the frequency of the module clock (due to the fact that the timer clock can be a scaled version of the module clock).

In Edge Aligned Mode, the timer period is:

$$T_{per} = \text{<Period-Value>} + 1; \text{ in } f_{tclk} \quad (19.1)$$

In Center Aligned Mode, the timer period is:

$$T_{per} = (\text{<Period-Value>} + 1) \times 2; \text{ in } f_{tclk} \quad (19.2)$$

For each of these counting schemes, the duty cycle of generated PWM signal is dictated by the value programmed into the **CC4yCR** register.

In Edge Aligned and Center Aligned Mode, the PWM duty cycle is:

$DC$= 1 - <Compare-Value>/(<Period-Value> + 1) (19.3)

Both **CC4yPR** and **CC4yCR** can be updated on the fly via software, enabling a glitch free transition between different period and duty cycle values for the generated PWM signal, **Section 19.2.5.2**

### 19.2.5.2 Updating the Period and Duty Cycle

Each CCU4 timer slice provides an associated shadow register for the period and compare values. This facilitates a concurrent update by software for these two parameters, with the objective of modifying during run time the PWM signal period and duty cycle.

In addition to the shadow registers for the period and compare values, one also has available shadow registers for the floating prescaler and dither functions, **CC4yFPCS** and **CC4yDITS** respectively (please address **Section 19.2.11** and **Section 19.2.10** for a complete description of these functions).

The structure of the shadow registers can be seen in **Figure 19-8**.


Subject to Agreement on the Use of Product Information

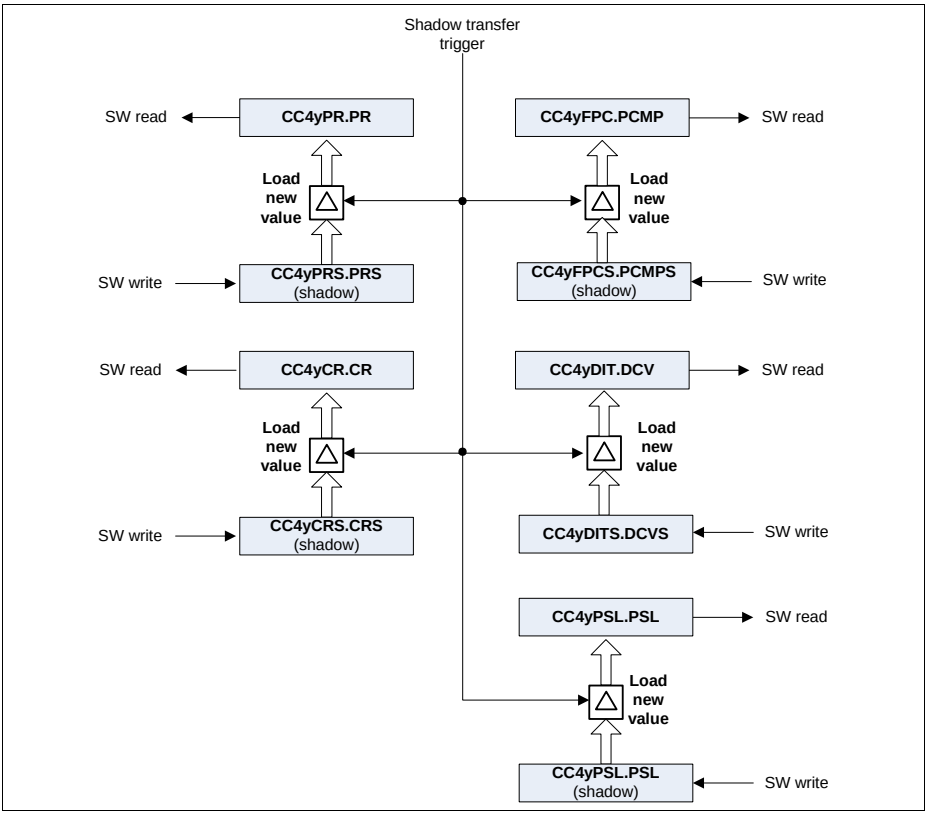



**Figure 19-8 Shadow registers overview**

The update of these registers can only be done by writing a new value into the associated shadow register and wait for a shadow transfer to occur.

Each group of shadow registers have an individual shadow transfer enable bit, **Figure 19-9**. The software must set this enable bit to $1_B$, whenever an update of the values is needed. These bits are automatically cleared by the hardware, whenever an update of the values if finished. Therefore every time that an update of the registers is needed the software must set again the specific bit(s).

Nevertheless it is also possible to clear the enable bit via software. This can be used in the case that an update of the values needs to be cancelled (after the enable bit has already been set).

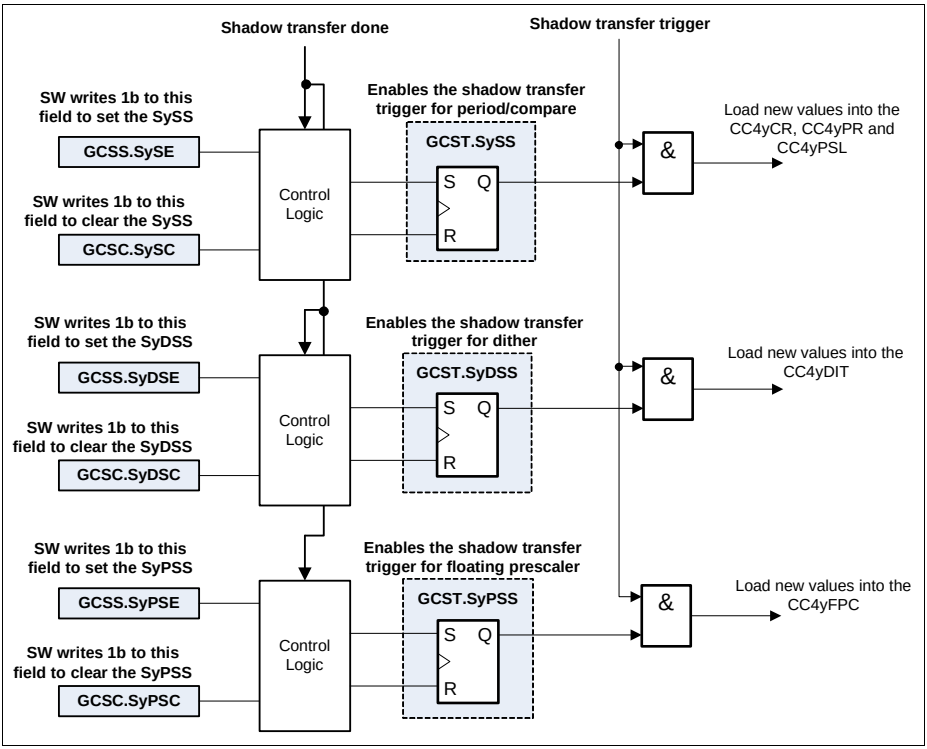



**Figure 19-9 Shadow transfer enable logic**

The shadow transfer operation is going to be done in the immediately next occurrence of a shadow transfer trigger, after the shadow transfer enable is set (**GCST**.SySS, **GCST**.SyDSS, **GCST**.SyPSS set to $1_B$).

The occurrence of the shadow transfer trigger is imposed by the timer counting scheme (edge aligned or center aligned). Therefore the slots when the values are updated can be:

- in the next clock cycle after a Period Match while counting up
- in the next clock cycle after an One Match while counting down
- immediately, if the timer is stopped and the shadow transfer enable bit(s) is set

**Figure 19-10** shows an example of the shadow transfer control when the timer slice has been configured into center aligned mode. For a complete description of all the timer slice counting modes, please address **Section 19.2.5.3**, **Section 19.2.5.4** and **Section 19.2.5.5**.

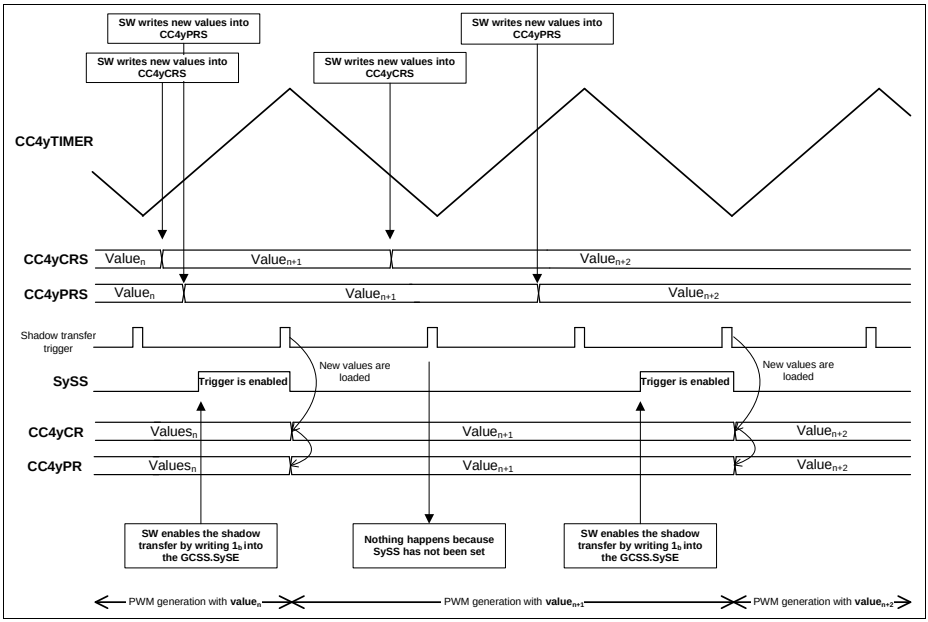



**Figure 19-10 Shadow transfer timing example - center aligned mode**

In some application cases it may be necessary to request shadow transfers not by software but by hardware. To perform this action each CCU4 contains a dedicated input that can be used to request a shadow transfer by hardware, the CCU4x.MCSS.

This input, when enabled, is used to set the shadow transfer enable bitfields (**GCST**.SySS, **GCST**.SyDSS and **GCST**.SyPSS) of the specific slice. It is possible to select which slice is using this input to perform the synchronization via the **GCTRL**.MSEy bit field. It is also possible to enable the usage of this signal for the three different shadow transfer signals: compare and period values, dither compare value and prescaler compare value. This can be configured on the **GCTRL**.MSDE field.

The structure for using the CCU4x.MCSS input signal can be seen in **Figure 19-9**. The usage of this signal is just an add on to the shadow transfer control and therefore all the previous described functions are still available.

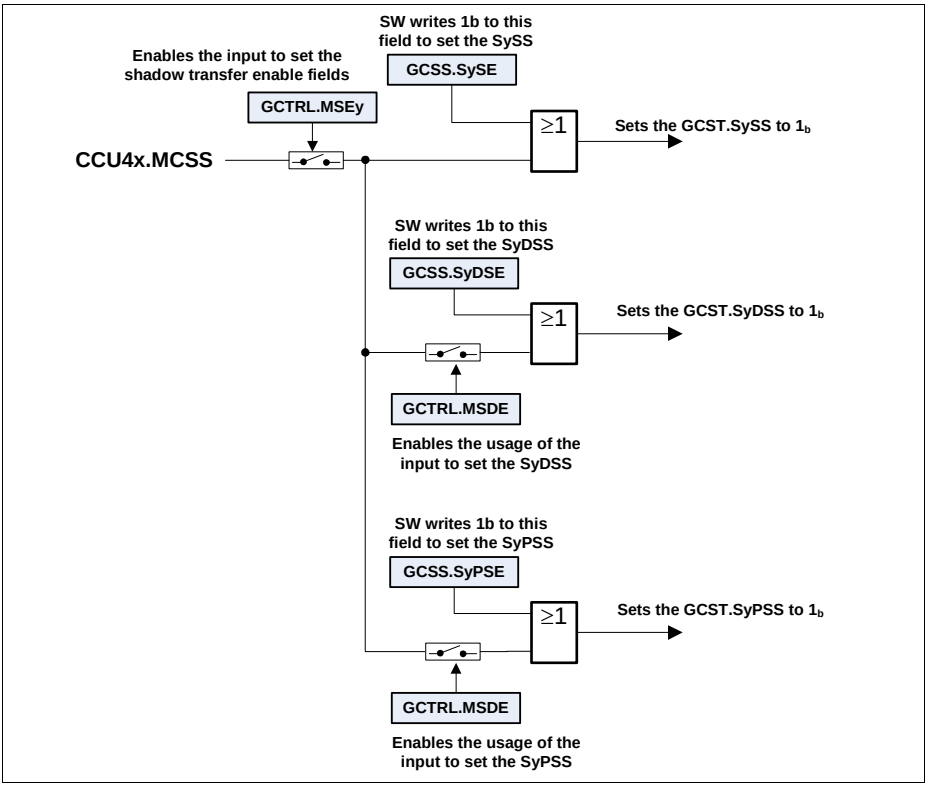



**Figure 19-11 Usage of the CCU4x.MCSS input**

### 19.2.5.3 Edge Aligned Mode

Edge aligned mode is the default counting scheme. In this mode, the timer is incremented until it matches the value programmed in the period register, **CC4yPR**. When period match is detected the timer is cleared to $0000_H$ and continues to be incremented.

In this mode, the value of the period register and compare register are updated with the value written by software into the correspondent shadow register, every time that an overflow occurs (period match), see **Figure 19-12**.

In edge aligned mode, the status bit of the comparison (CC4yST) is set one clock cycle after the timer hits the value programmed into the compare register. The clear of the status bit is done one clock cycle after the timer reaches $0000_H$.

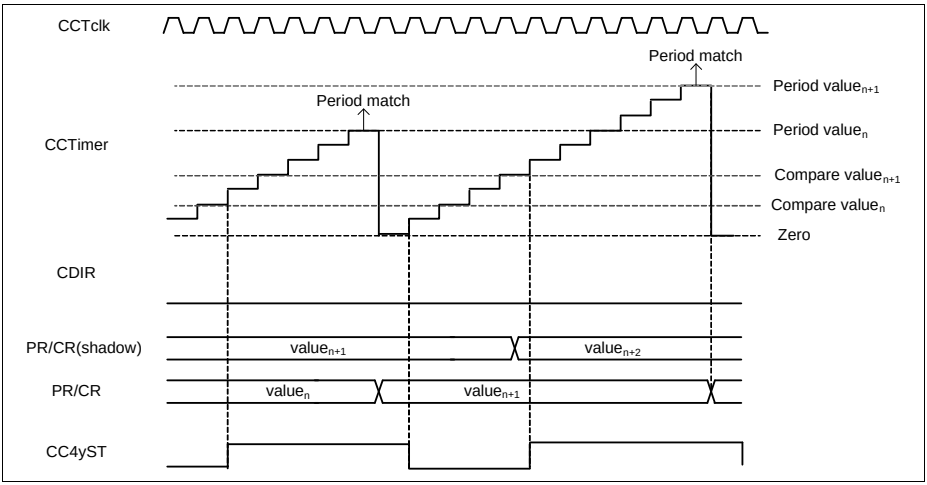



**Figure 19-12 Edge aligned mode, CC4yTC.TCM = $0_B$**

### 19.2.5.4 Center Aligned Mode

In center aligned mode, the timer is counting up or down with respect to the following rules:

- The counter counts up while **CC4yTCST**.CDIR = $0_B$ and it counts down while **CC4yTCST**.CDIR = $1_B$.
- Within the next clock cycle, the count direction is set to counting up (**CC4yTCST**.CDIR = $0_B$) when the counter reaches $0001_H$ while counting down.
- Within the next clock cycle, the count direction is set to counting down (**CC4yTCST**.CDIR = $1_B$), when the period match is detected while counting up.

The status bit (CC4yST) is always $1_B$ when the counter value is equal or greater than the compare value and $0_B$ otherwise.

While in edge aligned mode, the shadow transfer for compare and period registers is executed once per period. It is executed twice in center aligned mode as follows

- Within the next clock cycle after the counter reaches the period value, while counting up (**CC4yTCST**.CDIR = $0_B$).
- Within the next clock cycle after the counter reaches $0001_H$, while counting down (**CC4yTCST**.CDIR = $1_B$).

*Note: Bit **CC4yTCST**.CDIR changes within the next timer clock after the one-match or the period-match, which means that the timer continues counting in the previous direction for one more cycle before changing the direction.*

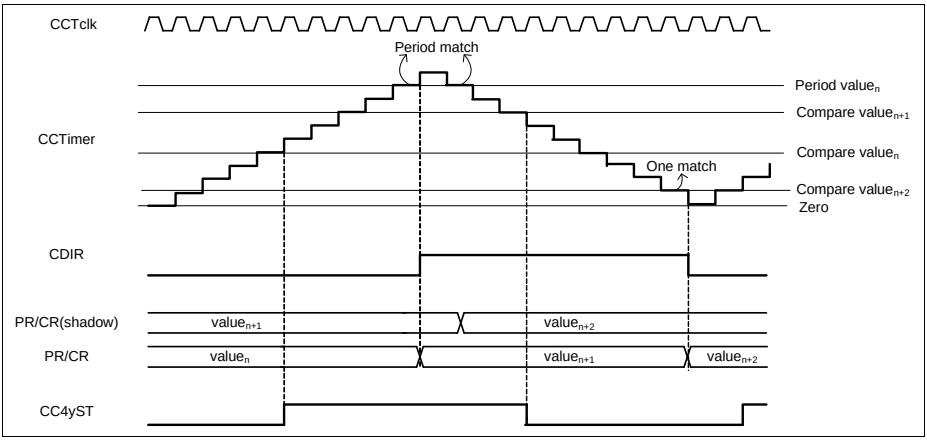



**Figure 19-13 Center aligned mode, CC4yTC.TCM = $1_B$**

### 19.2.5.5 Single Shot Mode

In single shot mode, the timer is stopped after the current timer period is finished. This mode can be used with center or edge aligned scheme.

In edge aligned mode, **Figure 19-14**, the timer is stopped when it is cleared to $0000_H$ after having reached the period value. In center aligned mode, **Figure 19-15**, the period is finished when the timer has counted down to $0000_H$.

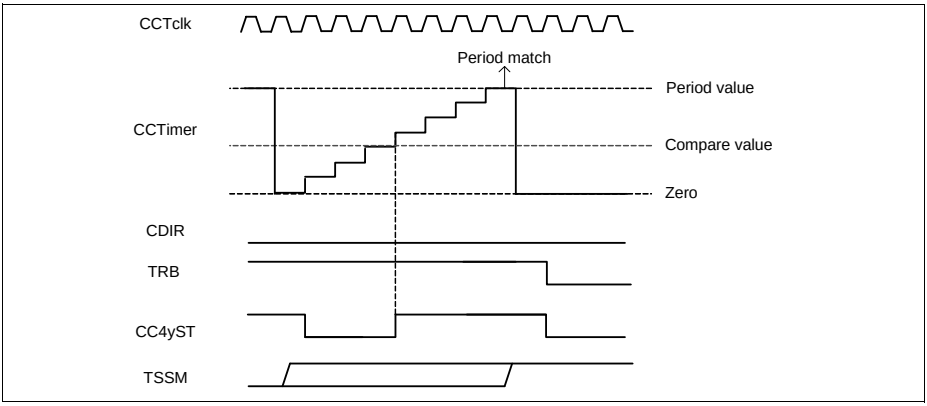



**Figure 19-14 Single shot edge aligned - CC4yTC.TSSM = $1_B$, CC4yTC.TCM = $0_B$**

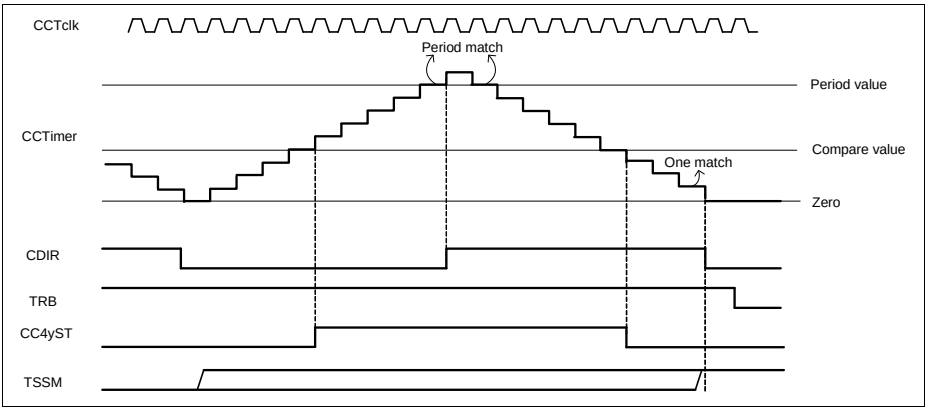



**Figure 19-15 Single shot center aligned - CC4yTC.TSSM = $1_B$, CC4yTC.TCM = $1_B$**

### 19.2.6 Active/Passive Rules

The general rules that set or clear the associated timer slice status bit (CC4yST), can be generalized independently of the timer counting mode.

The following events set the Status bit (CC4yST) to Active:

- in the next $f_{tclk}$ cycle after a compare match while counting up
- in the next $f_{tclk}$ cycle after a zero match while counting down

The following events set the Status bit (CC4yST) to Inactive:

- in the next $f_{tclk}$ cycle after a zero match (and not compare match) while counting up
- in the next $f_{tclk}$ cycle after a compare match while counting down

If external events are being used to control the timer operation, these rules are still applicable.

The status bit state can only be ‘override’ via software or by the external status bit override function, **Section 19.2.7.10**.

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.

### 19.2.7 External Events Control

Each CCU4 timer slice has the possibility of using up to three different input events, see **Section 19.2.2**. These three events can then be mapped to Timer Slice functions (the full set of available functions is described at **Section 19.2.3**)

These events can be mapped to any of the CCU4x.INy[P...A] inputs and there isn’t any imposition that an event cannot be used to perform several functions, or that an input

cannot be mapped to several events (e.g. input X triggers event 0 with rising edge and triggers event 1 with the falling edge).

#### 19.2.7.1 External Start/Stop

To select an external start function, one should map one of the events (output of the input selector) to a specific input signal, by setting the required value in the **CC4yINS**.EVxIS field and indicating the active edge of the signal on the **CC4yINS**.EVxEM field.

This event should be then mapped to the start or stop functionality by setting the **CC4yCMC**.STRTS (for the start) or the **CC4yTC**.ENDM (for the stop) with the proper value.

Notice that both start and stop functions are edge and not level active and therefore the active/passive configuration is set only by the **CC4yINS**.EVxEM.

The external stop by default just clears the run bit (**CC4yTCST**.TRB), while the start functions does the opposite. Nevertheless one can select an extended subset of functions for the external start and stop. This subset is controlled by the registers **CC4yTC**.ENDM (for the stop) and **CC4yTC**.STRM (for the start).

For the start subset (**CC4yTC**.STRM):

- sets the run bit/starts the timer (resume operation)
- clears the timer, sets the run bit/starts the timer (flush and start)

For the stop subset (**CC4yTC**.ENDM):

- clears the run/stops the timer (stop)
- clears the timer (flush)
- clears the timer, clears the run bit/stops the timer (flush and stop)

If in conjunction with an external start/stop (configured also/only as flush) and external up/down signal is used, during the flush operation the timer is going to be set to $0000_H$ if the actual counting direction is up or set with the value of the period register if the counting direction is down.

**Figure 19-16** to **Figure 19-19** shows the usage of two signals to perform the start/stop functions in all the previously mentioned subsets. External Signal(1) acts as an active HIGH start signal, while External Signal(2) is used as an active HIGH stop function.


Subject to Agreement on the Use of Product Information

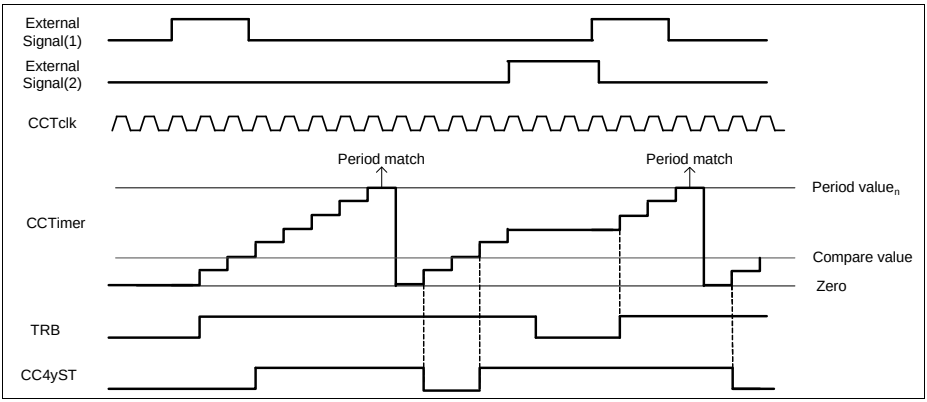



**Figure 19-16 Start (as start)/ stop (as stop) - CC4yTC.STRM = $0_B$, CC4yTC.ENDM = $00_B$**

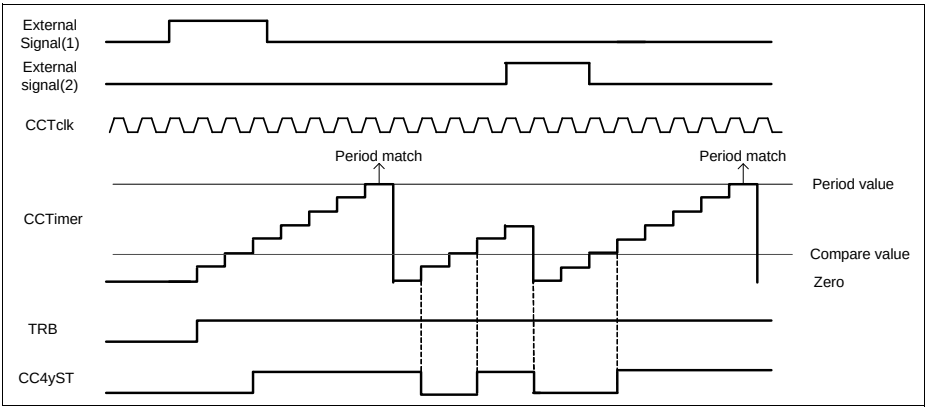



**Figure 19-17 Start (as start)/ stop (as flush) - CC4yTC.STRM = $0_B$, CC4yTC.ENDM = $01_B$**

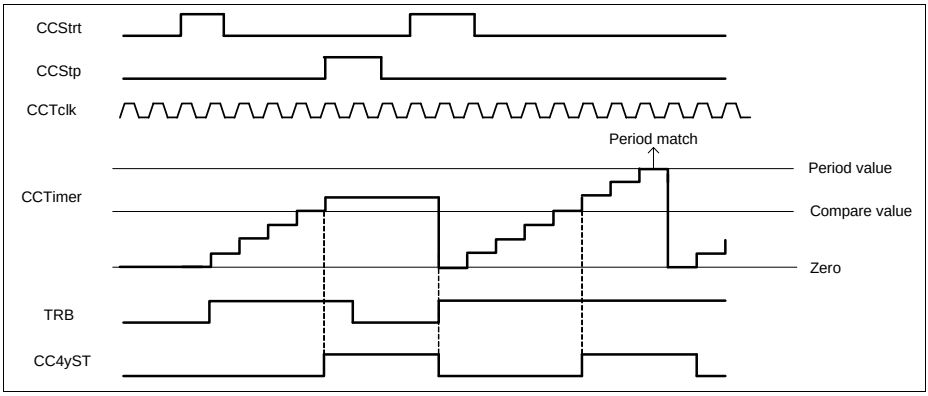



**Figure 19-18 Start (as flush and start)/ stop (as stop) - CC4yTC.STRM = $1_B$, CC4yTC.ENDM = $00_B$**

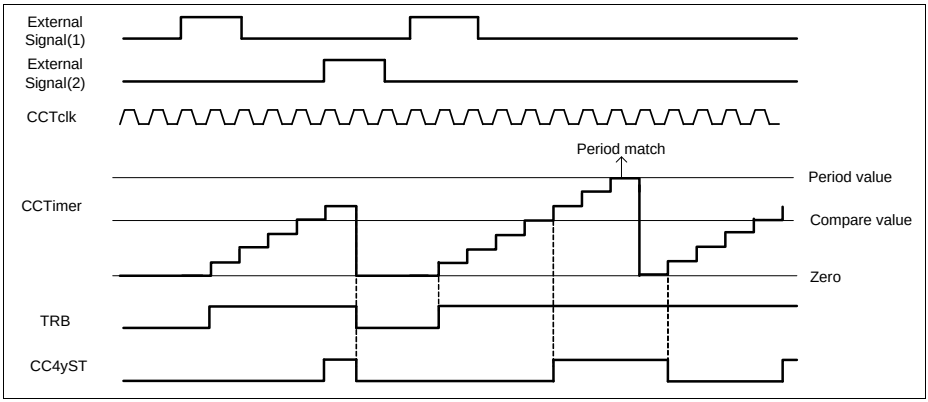



**Figure 19-19 Start (as start)/ stop (as flush and stop) - CC4yTC.STRM = $0_B$, CC4yTC.ENDM = $10_B$**

### 19.2.7.2 External Counting Direction

There is the possibility of selecting an input signal to act as increment/decrement control.

To select an external up/down control, one should map one of the events (output of the input selector) to a specific input signal, by setting the required value in the **CC4yINS**.EVxIS field and indicating the active level of the signal on the **CC4yINS**.EVxLM. This event should be then mapped to the up/down functionality by setting **CC4yCMC**.UDS with the proper value.

Notice that the up/down function is level active and therefore the active/passive configuration is set only by the **CC4yINS**.EVxLM.

The status bit of the slice (CC4yST) is always set when the timer value is equal or greater than the value stored in the compare register, see **Section 19.2.6**.

The update of the period and compare register values is done when:

- with the next clock after a period match, while counting up (**CC4yTCST**.CDIR = $0_B$)
- with the next clock after a one match, while counting down (**CC4yTCST**.CDIR = $1_B$)

The value of the **CC4yTCST**.CDIR register is updated accordingly with the changes on the decoded event. The Up/Down direction is always understood as **CC4yTCST**.CDIR = $1_B$ when counting down and **CC4yTCST**.CDIR = $0_B$ when counting up. Using an external signal to perform the up/down counting function and configuring the event as active HIGH means that the timer is counting up when the signal is HIGH and counting down when LOW.

**Figure 19-20** shows an external signal being used to control the counting direction of the time. This signal was selected as active HIGH, which means that the timer is counting down while the signal is HIGH and counting up when the signal is LOW.

*Note: For a signal that should impose an increment when LOW and a decrement when HIGH, the user needs to set the **CC4yINS**.EVxLM = $0_B$. When the operation is switched, then the user should set **CC4yINS**.EVxLM = $1_B$.*

*Note: Using an external counting direction control, sets the slice in edge aligned mode.*

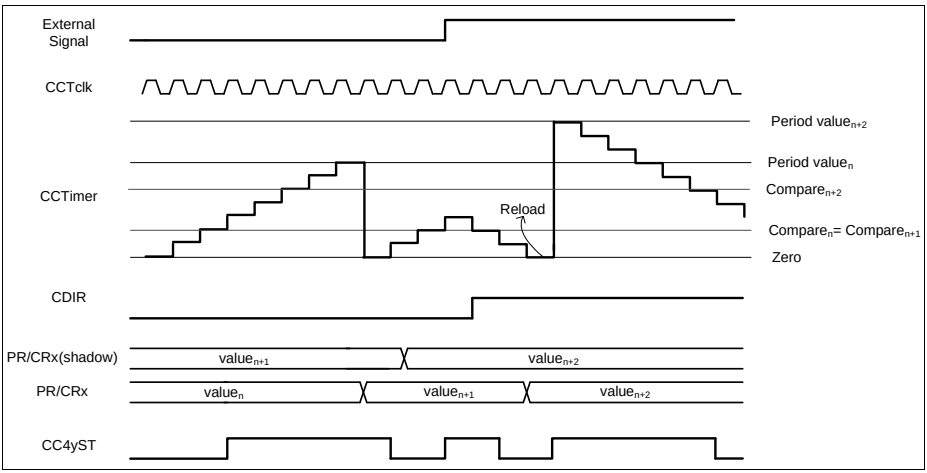



**Figure 19-20 External counting direction**

### 19.2.7.3 External Gating Signal

For pulse measurement, the user has the possibility of selecting an input signal that operates as counting gating.

To select an external gating control, one should map one of the events (output of the input selector) to a specific input signal, by setting the required value in the **CC4yINS**.EVxIS register and indicating the active level of the signal on the **CC4yINS**.EVxLM register. This event should be then mapped to the gating functionality by setting the **CC4yCMC**.GATES with the proper value.

Notice that the gating function is level active and therefore the active/passive configuration is set only by the **CC4yINS**.EVxLM.

The status bit during an external gating signal continues to be asserted when the compare value is reached and deasserted when the counter reaches $0000_H$. One should note that the counter continues to use the period register to identify the wrap around condition. **Figure 19-21** shows the usage of an external signal for gating the slice counter. The signal was set as active LOW, which means the counter gating functionality is active when the external value is zero.

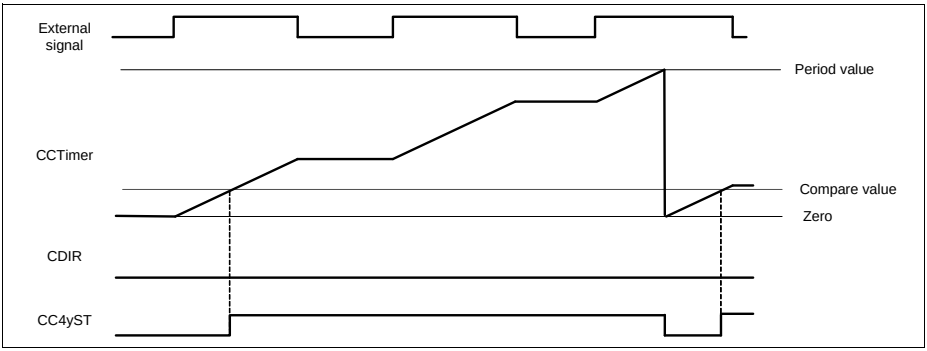



**Figure 19-21 External gating**

For any type of usage of the external gating function, the specific rung bit of the Timer Slice, **CC4yTCST**.TRB, needs to be set. This can be done via an additional external signal or directly via software.

### 19.2.7.4 External Count Signal

There is also the possibility of selecting an external signal to act as the counting event.

To select an external counting, one should map one of the events (output of the input selector) to a specific input signal, by setting the required value in the **CC4yINS**.EVxIS register and indicating the active edge of the signal on the **CC4yINS**.EVxEM register.


Subject to Agreement on the Use of Product Information

This event should be then mapped to the counting functionality by setting the **CC4yCMC**.CNTS with the proper value.

Notice that the counting function is edge active and therefore the active/passive configuration is set only by the **CC4yINS**.EVxEM.

One can select just a the rising, falling or both edges to perform a count. On **Figure 19-22**, the external signal was selected as a counter event for both falling and rising edges. Wrap around condition is still applied with a comparison with the period register.

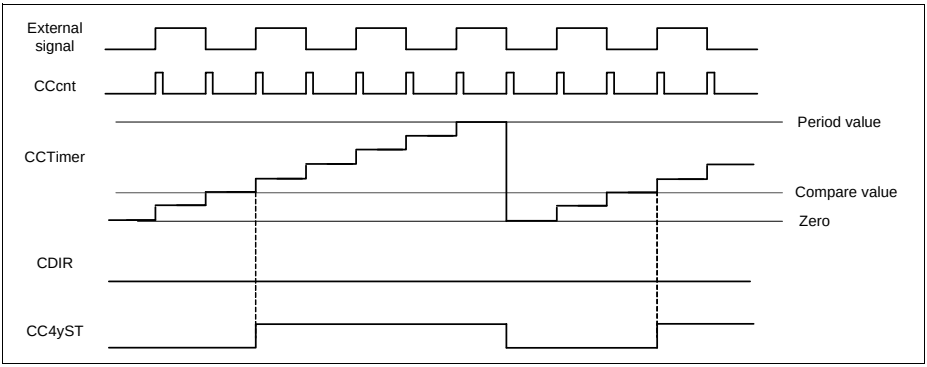



**Figure 19-22 External count**

For any type of usage of the external gating function, the specific rung bit of the Timer Slice, **CC4yTCST**.TRB, needs to be set. This can be done via an additional external signal or directly via software.

### 19.2.7.5 External Load

Each slice of the CCU4 also has a functionality that enables the user to select an external signal as trigger for reloading the value of the timer with the current value of the compare register (if **CC4yTCST**.CDIR = $0_B$) or with the value of the period register (if **CC4yTCST**.CDIR = $1_B$).

To select an external load signal, one should map one of the events (output of the input selector) to a specific input signal, by setting the required value in the **CC4yINS**.EVxIS register and indicating the active edge of the signal on the **CC4yINS**.EVxEM register. This event should be then mapped to the load functionality by setting the **CC4yCMC**.LDS with the proper value.

Notice that the load function is edge active and therefore the active/passive configuration is set only by the **CC4yINS**.EVxEM.

On figure **Figure 19-23**, the external signal (1) was used to act as a load trigger, active on the rising edge. Every time that a rising edge on external signal (1) is detected, the

timer value is loaded with the value present on the compare register. If an external signal is being used to control the counting direction, up or down, the timer value can be loaded also with the value set in the period register. The External signal (2) represents the counting direction control (active HIGH). If at the moment that a load trigger is detected, the signal controlling the counting direction is imposing a decrement, then the value set in the timer is the period value.

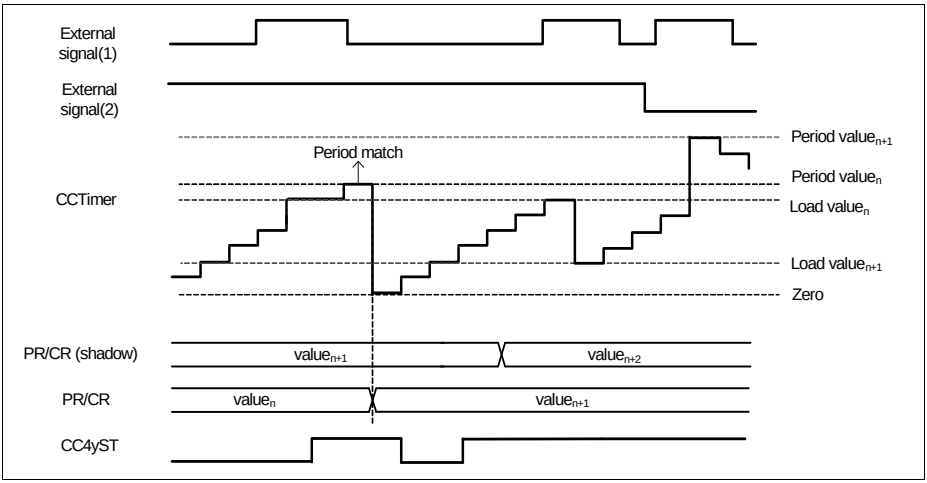



**Figure 19-23 External load**

### 19.2.7.6 External Capture

When selecting an external signal to be used as a capture trigger (if **CC4yCMC**.CAP0S or **CC4yCMC**.CAP1S are different from $0_H$), the user is automatically setting the specific slice into capture mode.

In capture mode the user can have up to four capture registers, see **Figure 19-26**: capture register 0 (**CC4yC0V**), capture register 1 (**CC4yC1V**), capture register 2 (**CC4yC2V**) and capture register 3 (**CC4yC3V**).

These registers are shared between compare and capture modes which imposes:

- if **CC4yC0V** and **CC4yC1V** are used for capturing, the compare registers **CC4yCR** and **CC4yCRS** are not available (no compare channel)
- if **CC4yC2V** and **CC4yC3V** are used for capturing, the period registers **CC4yPR** and **CC4yPRS** are not available (no period control)

To select an external capture signal, one should map one of the events (output of the input selector) to a specific input signal, by setting the required value in the **CC4yINS**.EVxIS register and indicating the active edge of the signal on the


Subject to Agreement on the Use of Product Information

**CC4yINS**.EVxEM register. This event should be then mapped to the capture functionality by setting the **CC4yCMC**.CAP0S/**CC4yCMC**.CAP1S with the proper value.

Notice that the capture function is edge active and therefore the active/passive configuration is set only by the **CC4yINS**.EVxEM.

The user has the possibility of selecting the following capture schemes:

- Different capture events for **CC4yC0V**/**CC4yC1V** and **CC4yC2V**/**CC4yC3V**
- The same capture event for **CC4yC0V**/**CC4yC1V** and **CC4yC2V**/**CC4yC3V** with the same capture edge. For this capture scheme, only the CCcapt1 functionality needs to be programmed. To enable this scheme, the field **CC4yTC**.SCE needs to be set to 1.

### Different Capture Events (SCE = $0_B$)

Every time that a capture trigger 1 occurs, CCcapt1, the actual value of the timer is captured into the capture register 3 and the previous value stored in this register is transferred into capture register 2.

Every time that a capture trigger 0 occurs, CCcapt0, the actual value of the timer is captured into the capture register 1 and the previous value stored in this register is transferred into capture register 0.

Every time that a capture procedure into one of the registers occurs, the respective full flag is set. This flag is cleared automatically by HW when the SW reads back the value of the capture register (by reading the specific capture register or by reading the extended capture read value, see **Section 19.2.7.7**).

The capture of a new value into a specific capture registers is dictated by the status of the full flag as follows:

$$CC4yC1V_{capt} = \text{NOT}(CC4yC1V_{full_flag} \text{ AND } CC4yC0V_{full_flag}) \quad (19.4)$$

$$CC4yC0V_{capt} = CC4yC1V_{full_flag} \text{ AND NOT}(CC4yC0V_{full_flag}) \quad (19.5)$$

It is also possible to disable the effect of the full flags reset by setting the **CC4yTC**.CCS = $1_B$. This enables a continuous capturing independent if the values captured have been read or not.

*Note: When using the period registers for capturing, **CC4yCMC**.CAP1S different from $00_B$, the counter always uses its full 16 bit width as period value.*

On **Figure 19-24**, an external signal was selected as an event for capturing the timer value into the **CC4yC0V**/**CC4yC1V** registers. The status bit, CC4yST, during capture mode is asserted whenever a capture trigger is detected and deasserted when the counter matches $0000_H$.


Subject to Agreement on the Use of Product Information

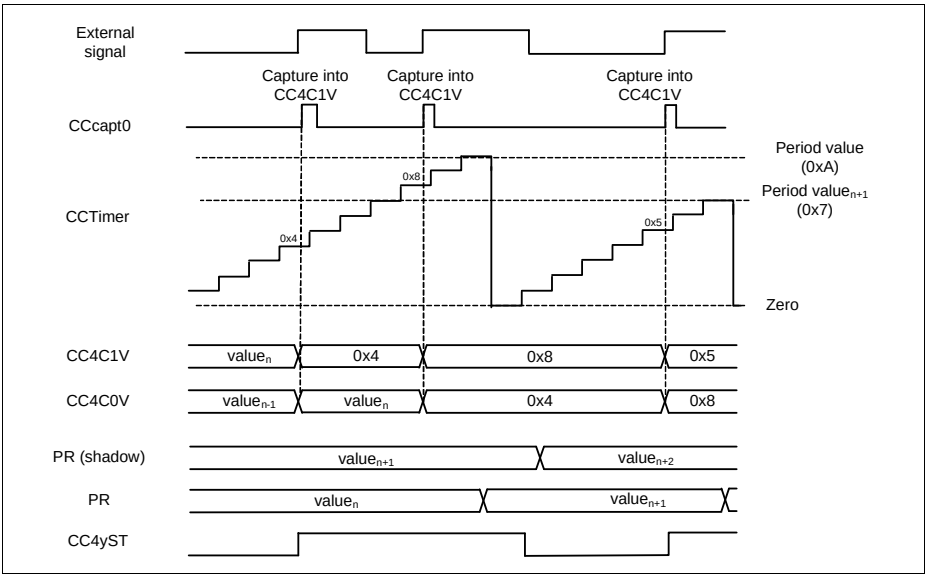



**Figure 19-24 External capture - CC4yCMC.CAP0S != $00_B$, CC4yCMC.CAP1S = $00_B$**

On **Figure 19-25**, two different signals were used as source for capturing the timer value into the **CC4yC0V**/**CC4yC1V** and **CC4yC2V**/**CC4yC3V** registers.

External signal(1) was selected as rising edge active capture source for **CC4yC0V**/**CC4yC1V**. External signal(2) was selected has the capture source for **CC4yC2V**/**CC4yC3V**, but as opposite to the external signal(1), the active edge was selected has falling.

See **Section 19.2.12.4**, for the complete capture mode usage description.

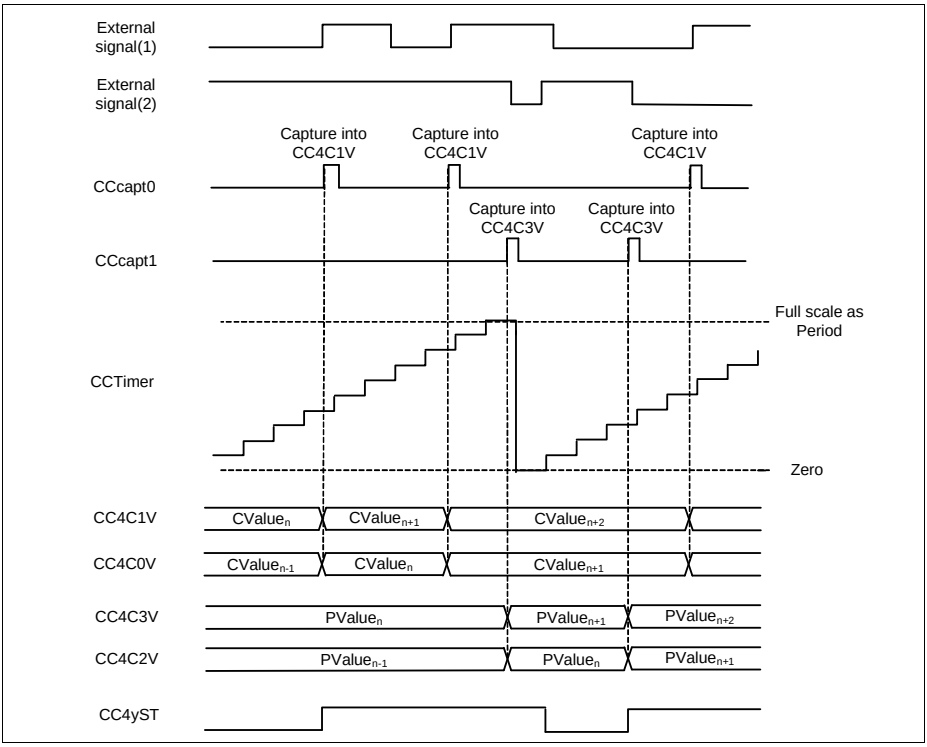



**Figure 19-25 External capture - CC4yCMC.CAP0S != $00_B$, CC4yCMC.CAP1S != $00_B$**

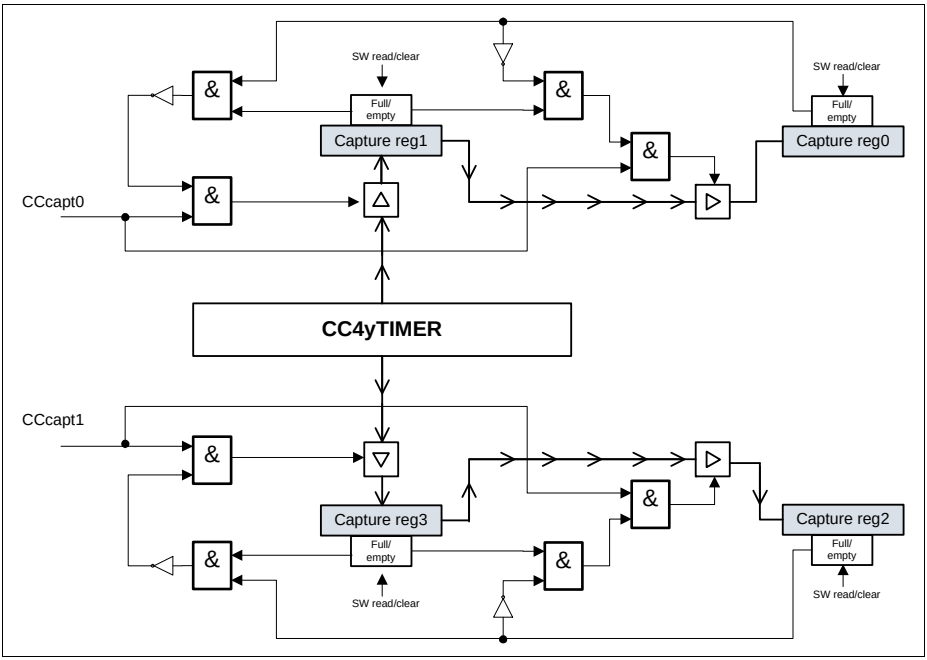



**Figure 19-26 Slice capture logic**

### Same Capture Event (SCE = $1_B$)

Setting the field **CC4yTC**.SCE = $1_B$, enables the possibility of having 4 capture registers linked with the same capture event, **Figure 19-28**. The function that controls the capture is the CCcapt1.

The capture logic follows the same structure shown in **Figure 19-26** but extended to a four register chain, see **Figure 19-27**. The same full flag lock rules are applied to the four register chain (it also can be disabled by setting the **CC4yTC**.CCS = $1_B$):

$$CC4yC3V_{capt} = \text{NOT}(CC4yC3V_{full_flag} \text{ AND } CC4yC2V_{full_flag} \text{ AND } CC4yC2V_{full_flag} \text{ AND } CC4yC1V_{full_flag}) \quad (19.6)$$

$$CC4yC2V_{capt} = CC4yC3V_{full_flag} \text{ AND NOT}(CC4yC2V_{full_flag} \text{ AND } CC4yC1V_{full_flag} \text{ AND } CC4yC0V_{full_flag}) \quad (19.7)$$

$$CC4yC1V_{capt} = CC4yC2V_{full_flag} \text{ AND NOT}(CC4yC1V_{full_flag} \text{ AND } CC4yC0V_{full_flag}) \quad (19.8)$$

$$CC4yC0V_{capt} = CC4yC1V_{full_flag} \text{ AND NOT}(CC4yC0V_{full_flag}) \quad (19.9)$$

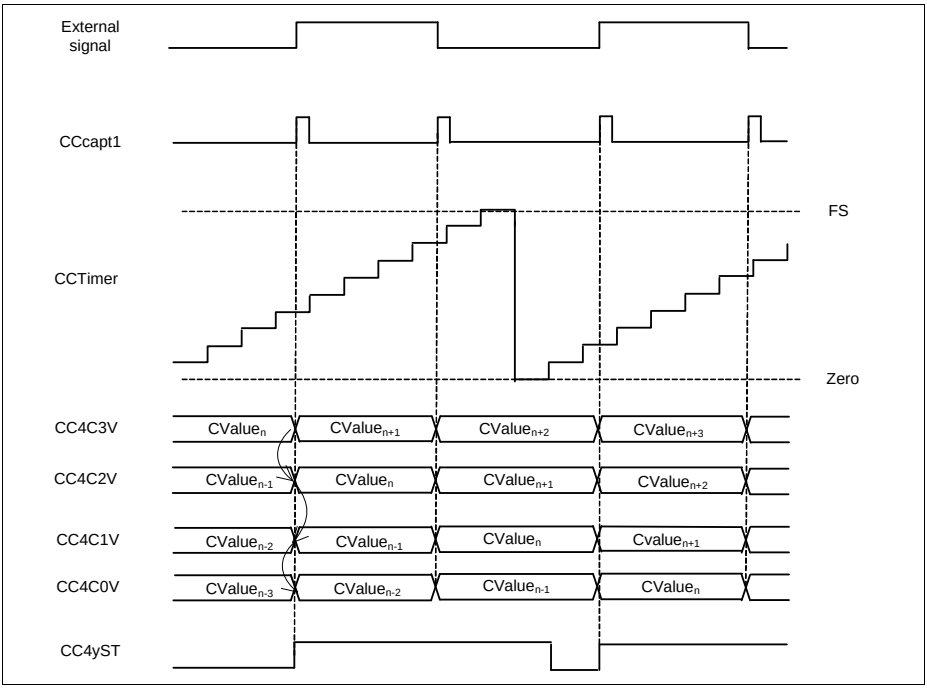



**Figure 19-27 External Capture - CC4yTC.SCE = $1_B$**

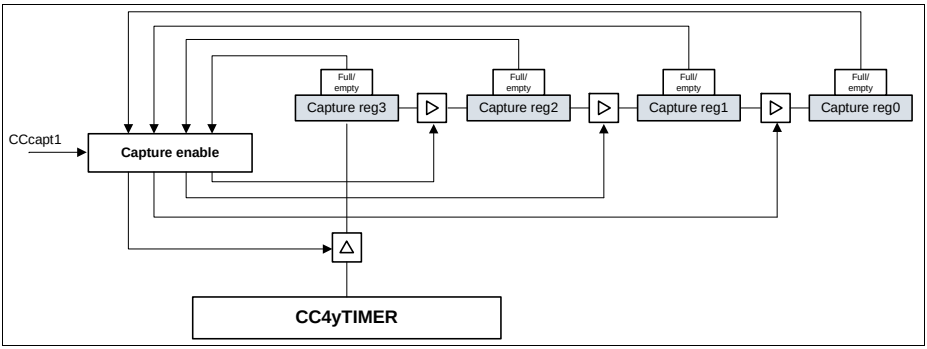



**Figure 19-28 Slice Capture Logic - CC4yTC.SCE = $1_B$**


Subject to Agreement on the Use of Product Information

### 19.2.7.7 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 cpature 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 19-29**.


Subject to Agreement on the Use of Product Information

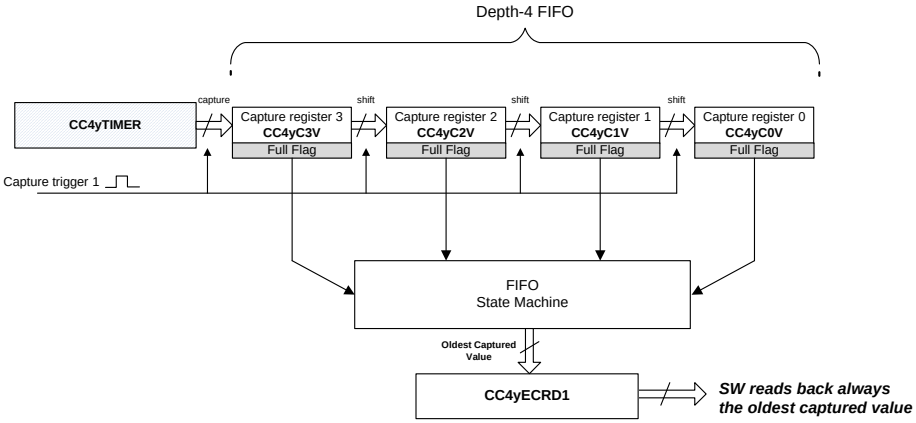



**Figure 19-29 Capture Extended Read Back - Depth 4**

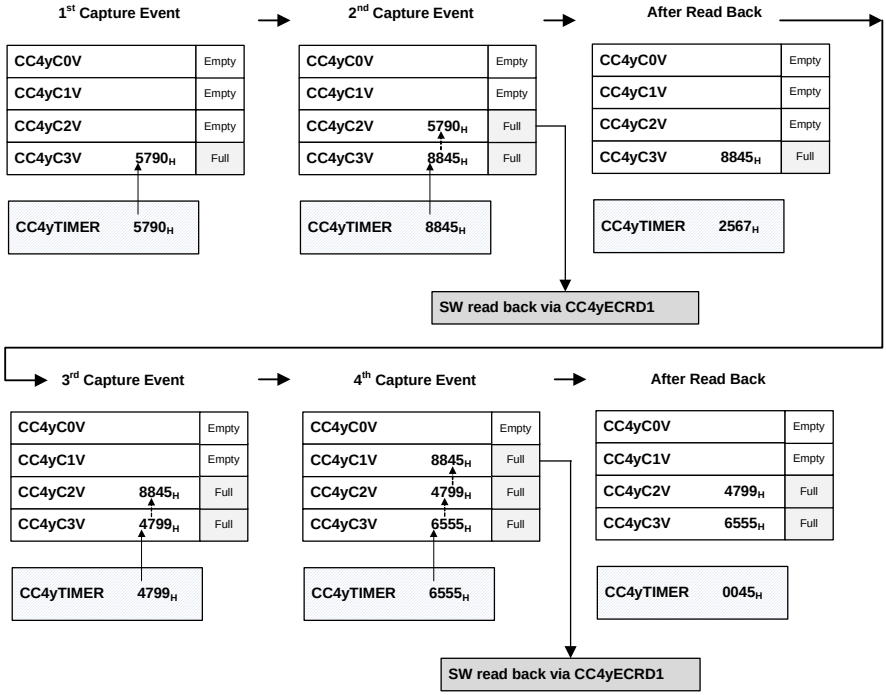



**Figure 19-30 Depth 4 software access example**

### 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 19-31**.


Subject to Agreement on the Use of Product Information

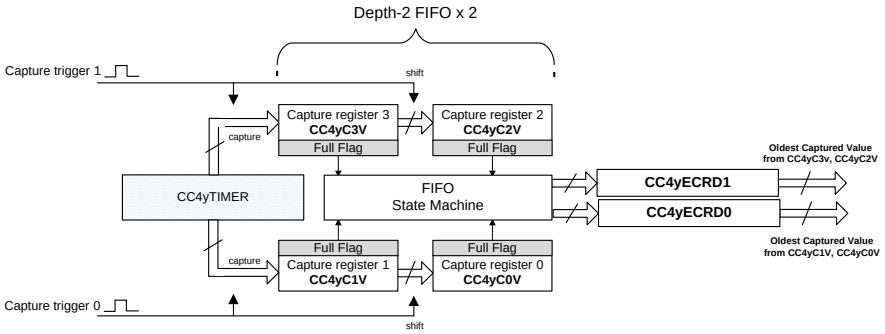



**Figure 19-31 Capture Extended Read Back - Depth 2**

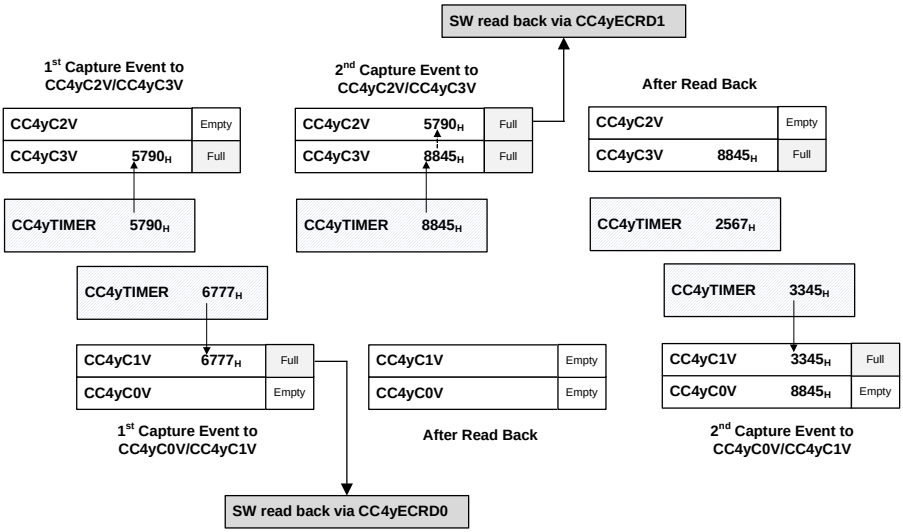



**Figure 19-32 Depth 2 software access example**

### 19.2.7.8 External Modulation

An external signal can be used to perform a modulation at the output of each timer slice.

To select an external modulation signal, one should map one of the input signals to one of the events, by setting the required value in the **CC4yINS**.EVxIS register and indicating the active level of the signal on the **CC4yINS**.EVxLM register. This event should be then

mapped to the modulation functionality by setting the **CC4yCMC**.MOS = $01_B$ if event 0 is being used, **CC4yCMC**.MOS = $10_B$ if event 1 or **CC4yCMC**.MOS = $11_B$ if event 2.

Notice that the modulation function is level active and therefore the active/passive configuration is set only by the **CC4yINS**.EVxLM.

The modulation has two modes of operation:

- modulation event is used to clear the CC4yST bit - **CC4yTC**.EMT = $0_B$
- modulation event is used to gate the outputs - **CC4yTC**.EMT = $1_B$

On **Figure 19-33**, 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 19-34**.

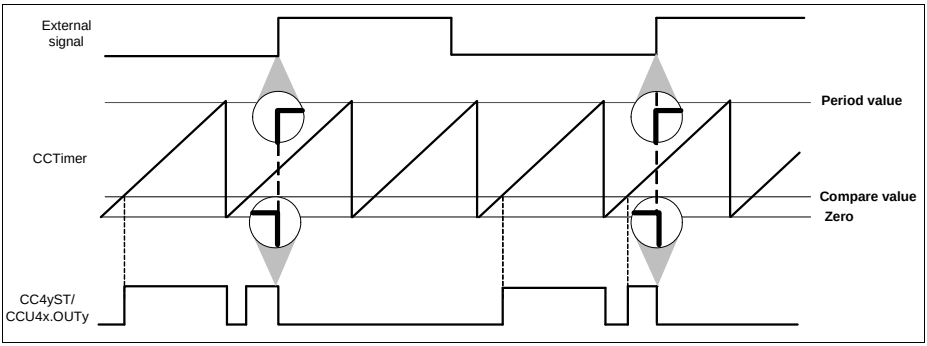



**Figure 19-33 External modulation clearing the ST bit - CC4yTC.EMT = $0_B$**

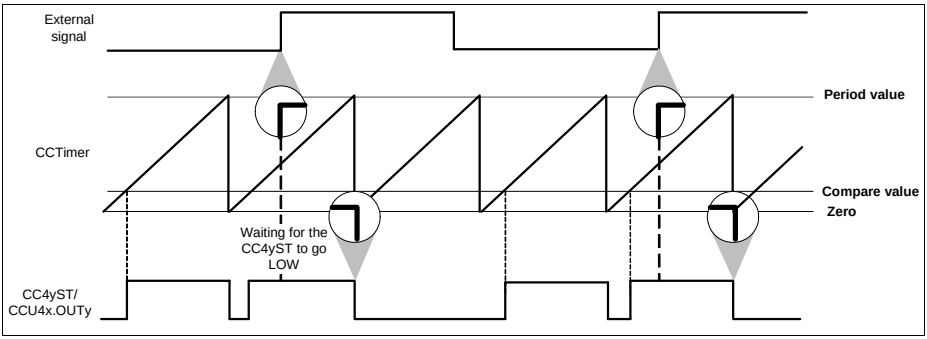



**Figure 19-34 External modulation clearing the ST bit - CC4yTC.EMT = $0_B$, CC4yTC.EMS = $1_B$**

On **Figure 19-35**, the external modulation event was used as gating signal of the outputs, **CC4yTC**.EMT = $1_B$. The external signal was configured to be active HIGH, **CC4yINS**.EVxLM = $0_B$, which means that when the external signal is HIGH the outputs are set to the PASSIVE state.In this mode, the gating event can also be synchronized with the PWM signal by setting the **CC4yTC**.EMS = $1_B$.

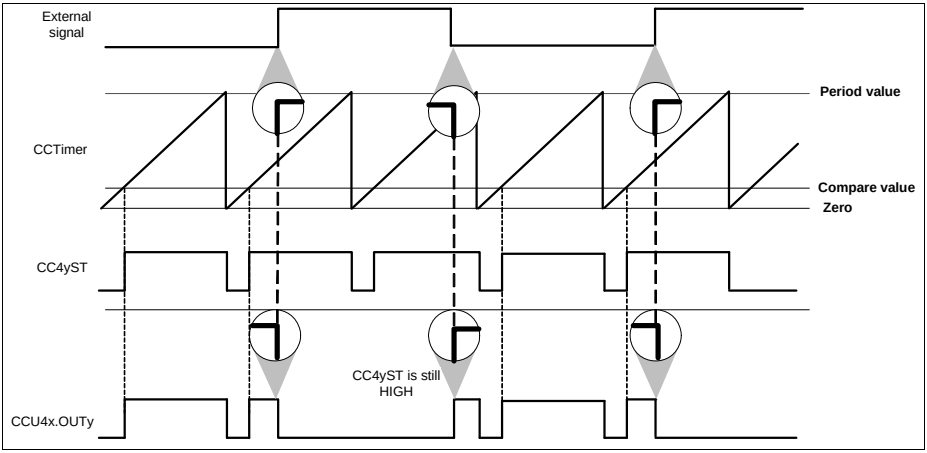



**Figure 19-35 External modulation gating the output - CC4yTC.EMT = $1_B$**

### 19.2.7.9 TRAP Function

The TRAP functionality allows the PWM outputs to react on the state of an input pin. This functionality can be used to switch off the power devices if the TRAP input becomes active.

To select the TRAP functionality, one should map one of the input signals to event number 2, by setting the required value in the **CC4yINS**.EV2IS register and indicating the active level of the signal on the **CC4yINS**.EV2LM register. This event should be then mapped to the trap functionality by setting the **CC4yCMC**.TS = $1_B$.

Notice that the trap function is level active and therefore the active/passive configuration is set only by the **CC4yINTS**.EV2LM.

There are two bitfields that can be monitored via software to crosscheck the TRAP function, **Figure 19-36**:

- 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.

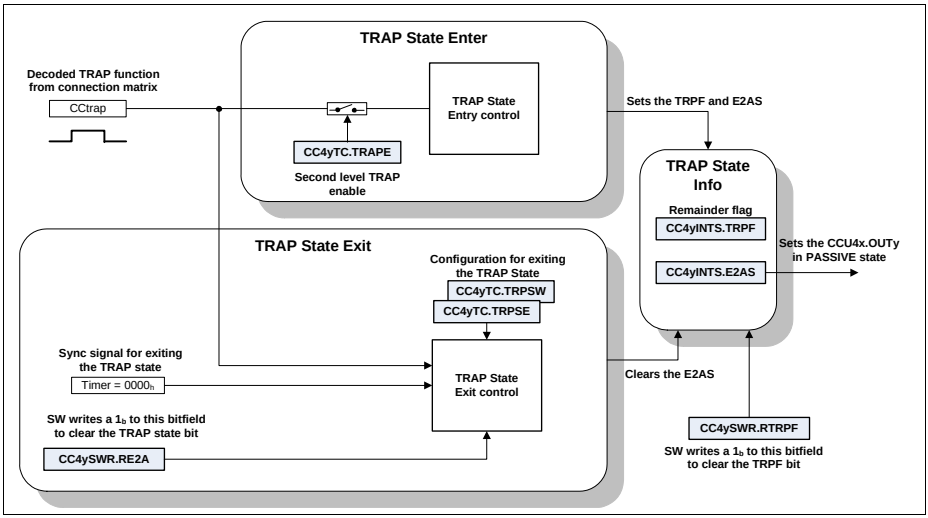



**Figure 19-36 Trap control diagram**

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 19-37**.

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 = $1_B$

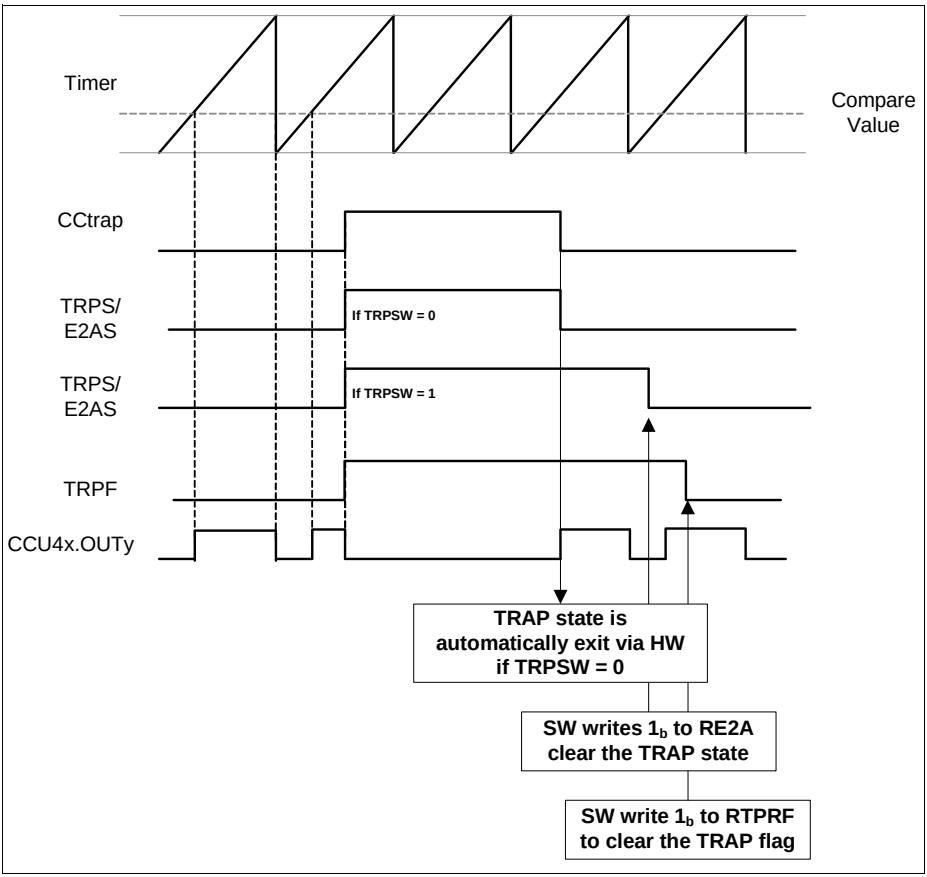



**Figure 19-37 Trap timing diagram, CC4yPSL.PSL = $0_B$ (output passive level is $0_B$)**

It is also possible to synchronize the exiting of the TRAP state with the PWM signal, **Figure 19-38**. This function is enabled when the bitfield **CC4yTC**.TRPSE = $1_B$.

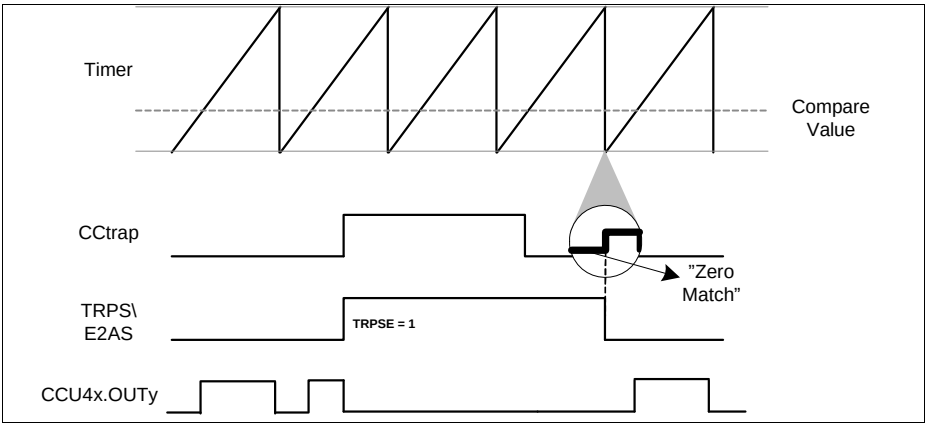



**Figure 19-38 Trap synchronization with the PWM signal, CC4yTC.TRPSE = $1_B$**

### 19.2.7.10 Status Bit Override

For complex timed output control, each Timer Slice has a functionality that enables the override of the status bit (CC4yST) with a value passed trough an external signal.

The override of the status bit, can then lead to a change on the output pin, CCU4xOUTy (from inactive to active or vice versa).

To enable this functionality, two signals are needed:

- One signal that acts as a trigger to override the status bit (edge active)
- One signal that contains the value to be set in the status bit (level active)

To use the status bit override functionality, one should map the signal that acts as trigger to the event number 1, by setting the required value in the **CC4yINS**.EV1IS register and indicating the active edge of the signal on the **CC4yINS**.EV1EM register.

The signal that carries the value to be set on the status bit, needs to be mapped to the event number 2, by setting the required value in the **CC4yINS**.EV2IS register. The **CC4yINS**.EV2LM register should be set to $0_B$ if no inversion on the signal is needed and to $1_B$ otherwise.

The events should be then mapped to the status bit functionality by setting the **CC4yCMC**.OFS = $1_B$.

**Figure 19-39** 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.


Subject to Agreement on the Use of Product Information

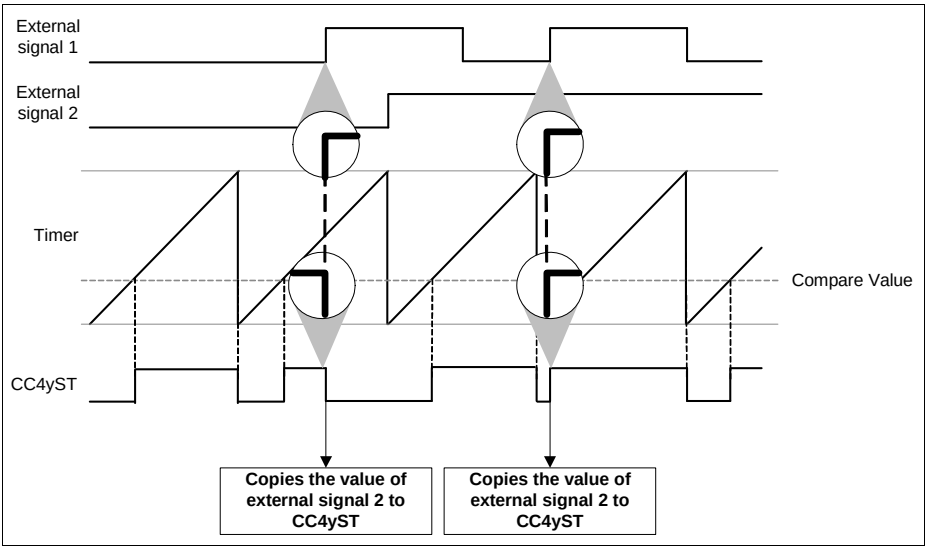



**Figure 19-39 Status bit override**

### 19.2.8 Multi-Channel Control

The multi channel control mode is selected individually in each slice by setting the **CC4yTC**.MCME = $1_B$.

Within this mode, the output state of the Timer Slices (the ones set in multi channel mode) can be controlled in parallel by a single pattern.

The pattern is controlled via the CCU4 inputs, CCU4x.MCI0, CCU4x.MCI1, CCU4x.MCI2 and CCU4x.MCI3. Each of these inputs is connected directly to the associated slice input, e.g. CCU4x.MCI0 to CC40MCI, CCU4x.MCI1 to CC41MCI.

This pattern can be controlled directly by other module. The connectivity of each device may allow different control possibilities therefore one should address **Section 19.8** to check what modules are connected to these inputs.

When using the Multi Channel support of the CCU4, one can achieve a complete synchronicity between the output state update, CCU4x.OUTy, and the update of a new pattern, **Figure 19-40**. This synchronocity feature can be enabled by using some specific modules and therefore one should address **Section 19.8** to check which module is controlling the CCU4x.MCIy inputs.


Subject to Agreement on the Use of Product Information

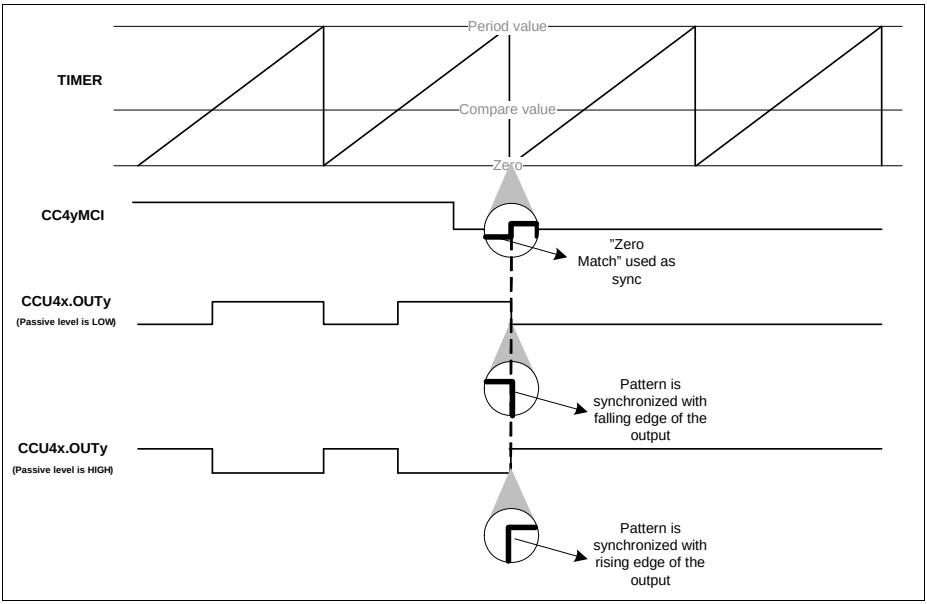



**Figure 19-40 Multi channel pattern synchronization**

**Figure 19-41** shows the usage of the multi channel mode in conjunction with all four Timer Slices inside the CCU4.

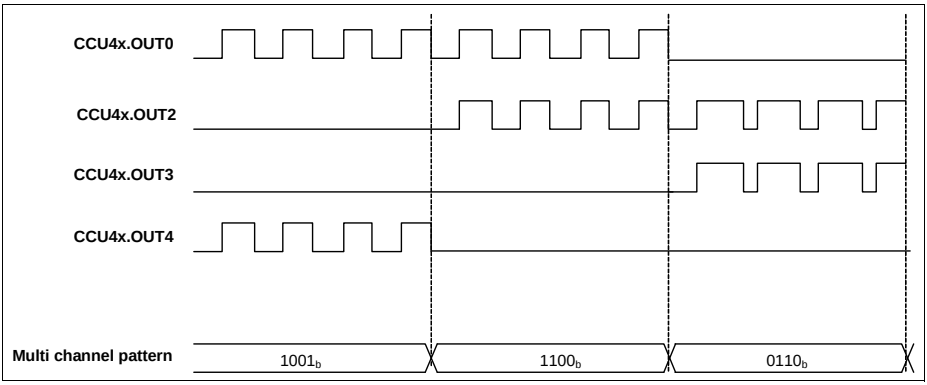



**Figure 19-41 Multi Channel mode for multiple Timer Slices**

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 **CC4yINS**.MCME = $1_B$, see **Figure 19-42**.

The multi pattern input synchronization can be seen on **Figure 19-43**. 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 19-43**.

In a normal operation, where no external signal is used to control the counting direction, the signal used to enable the sampling of the pattern is always the period match when in edge aligned and the one match when in center aligned mode. When an external signal is used to control the counting direction, depending if the counter is counting up or counting down, the period match or the one match signal is used, respectively.

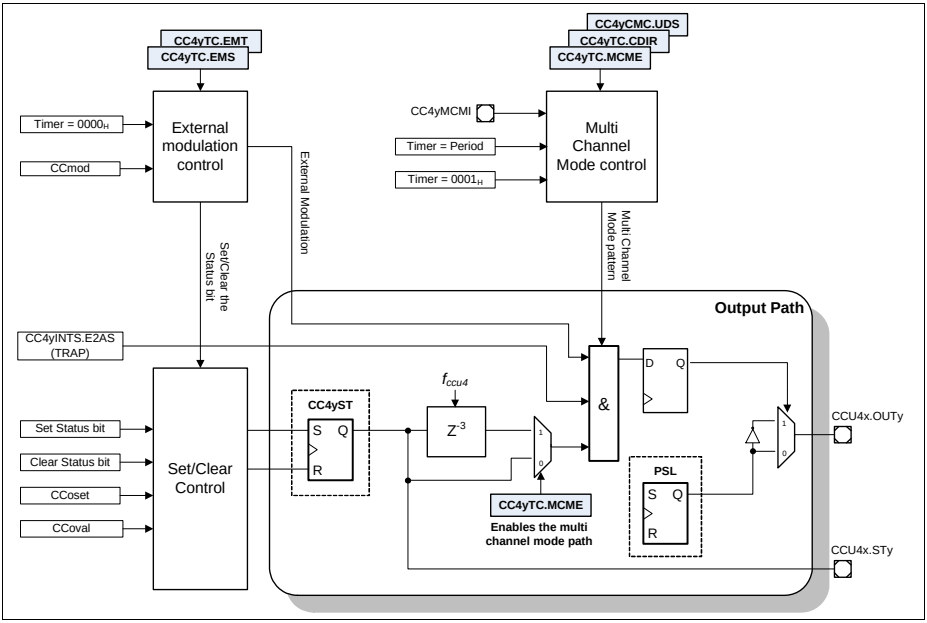



**Figure 19-42 CC4y Status bit and Output Path**

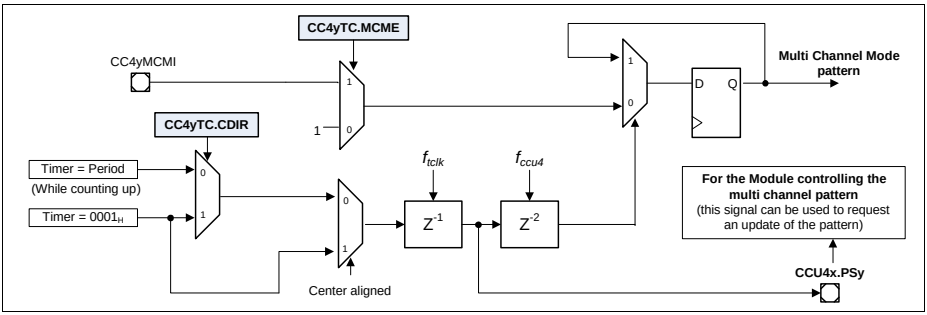



**Figure 19-43 Multi Channel Mode Control Logic**

## 19.2.9 Timer Concatenation

The CCU4 offers a very easy mechanism to perform a synchronous timer concatenation. This functionality can be used by setting the **CC4yTC**.TCE = $1_B$. By doing this the user is doing a concatenation of the actual CCU4 slice with the previous one, see **Figure 19-44**.

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 **CC4yTC**.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 **CC4yTC**.TCE = $1_B$ in two adjacent slices and to enable a 32 bit timer, the **CC4yTC**.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


Subject to Agreement on the Use of Product Information

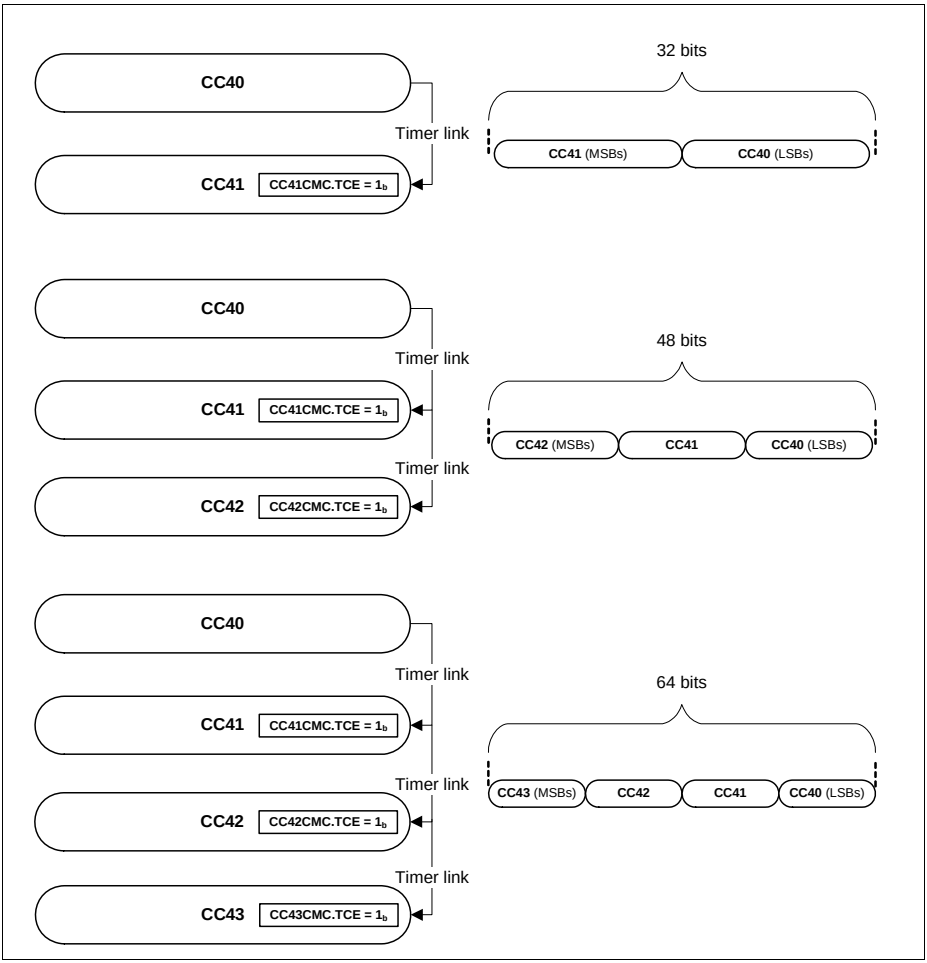



**Figure 19-44 Timer Concatenation Example**

Each Timer Slice is connected to the adjacent Timer Slices via a dedicated concatenation interface. This interface allows the concatenation of not only the Timer counting operation, but also a synchronous input trigger handling for capturing and loading operations, **Figure 19-45**.

*Note: For all cases CC40 and CC43 are not considered adjacent slices*

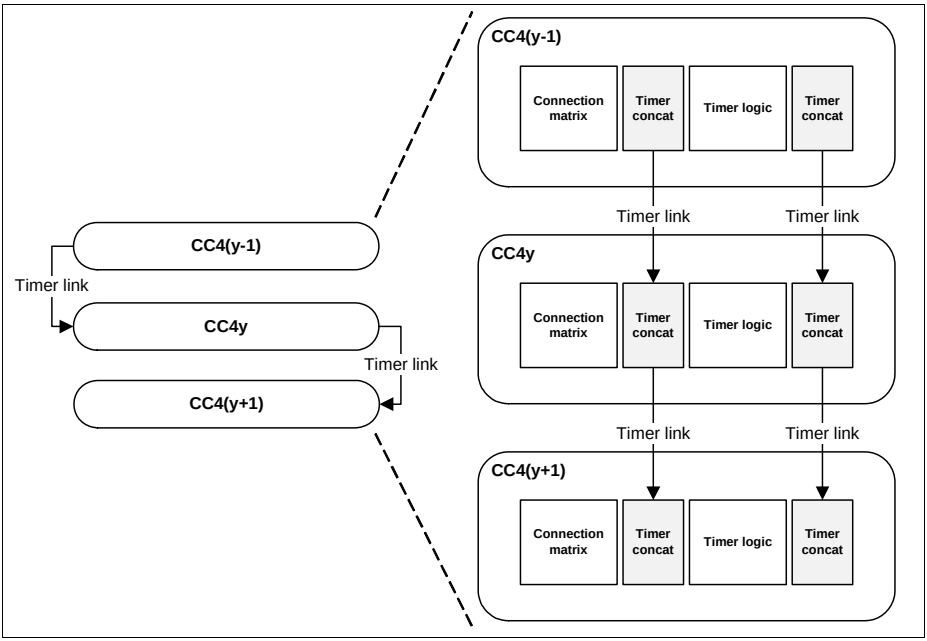



**Figure 19-45 Timer Concatenation Link**

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 19-45**. With this link, the timer length can be easily adjusted to 32, 48 or 64 bits (counting up or counting down).

The last three signals are used to perform a synchronous link between the capture and load functions, for the concatenated timer system. This means that the user can have a capture or load function programmed in the first Timer Slice, and propagate this capture or load trigger synchronously from the LSBs until the MSBs, **Figure 19-46**.

The capture or load function only needs to be configured in the first Timer Slice (the one holding the LSBs). From the moment that **CC4yTC**.TCE is set to $1_B$, in the following Timer Slices, the link between these functions is done automatically by the hardware.

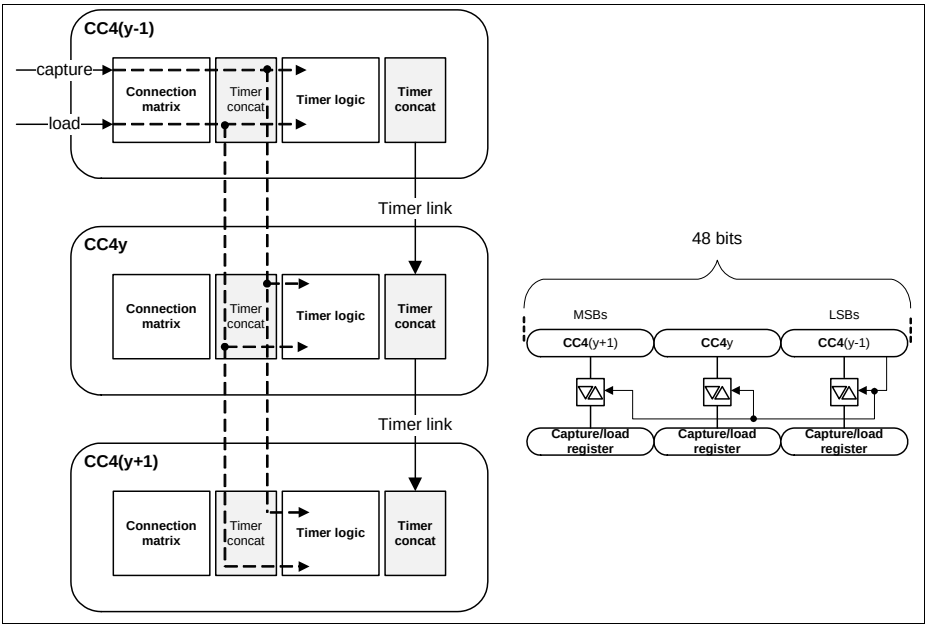



**Figure 19-46 Capture/Load Timer Concatenation**

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 19-47**.

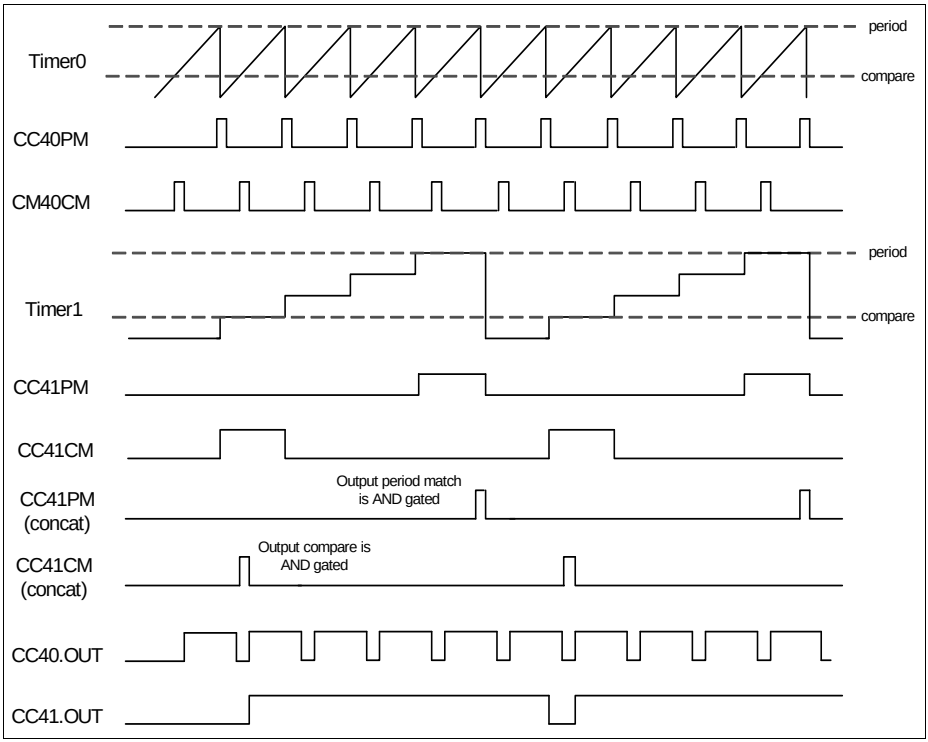



**Figure 19-47 32 bit concatenation timing diagram**

*Note: The counting direction of the concatenated timer needs to be fixed. The timer can count up or count down, but the direction cannot be updated on the fly.*

**Figure 19-48** gives an overview of the timer concatenation logic. Notice that all the mechanism is controlled solely by the **CC4yTC**.TCE bitfield.

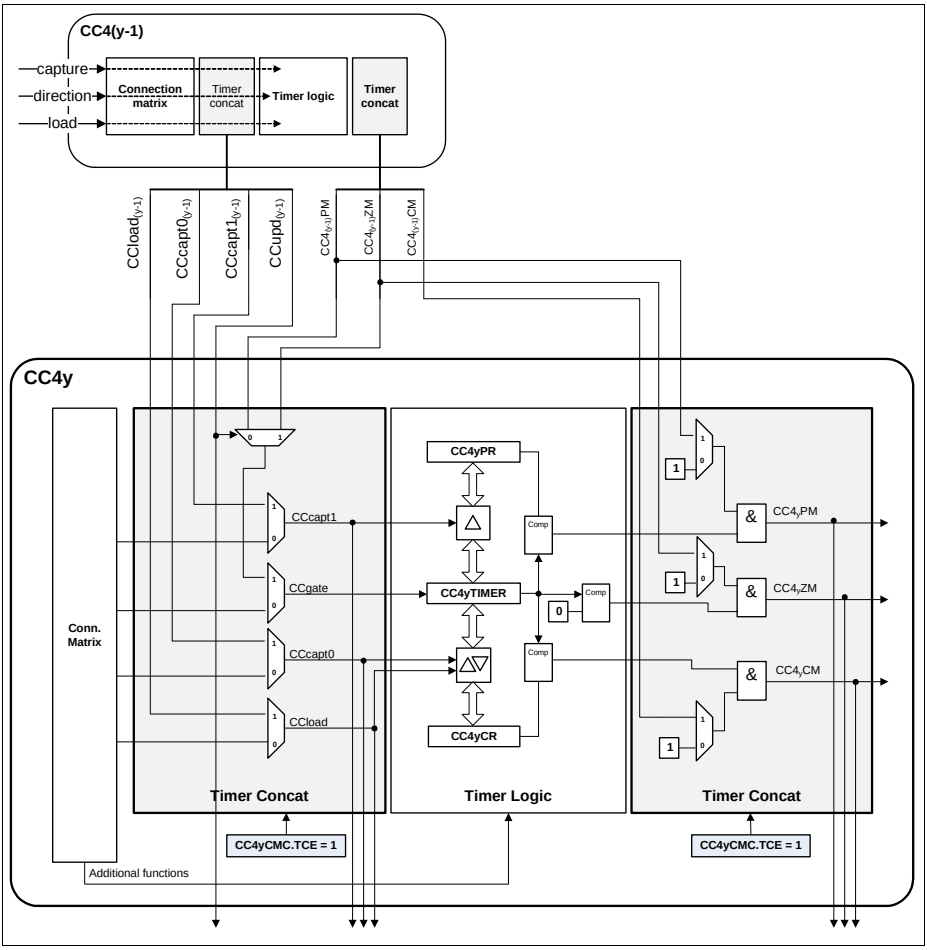



**Figure 19-48 Timer concatenation control logic**

## 19.2.10 PWM Dithering

The CCU4 has an automatic PWM dithering insertion function. This functionality can be used with very slow control loops that cannot update the period/compare values in a fast manner, and by that fact the loop can lose precision on long runs. By introducing dither on the PWM signal, the average frequency/duty cycle is then compensated against that error.

Each slice contains a dither control unit, see **Figure 19-49**.

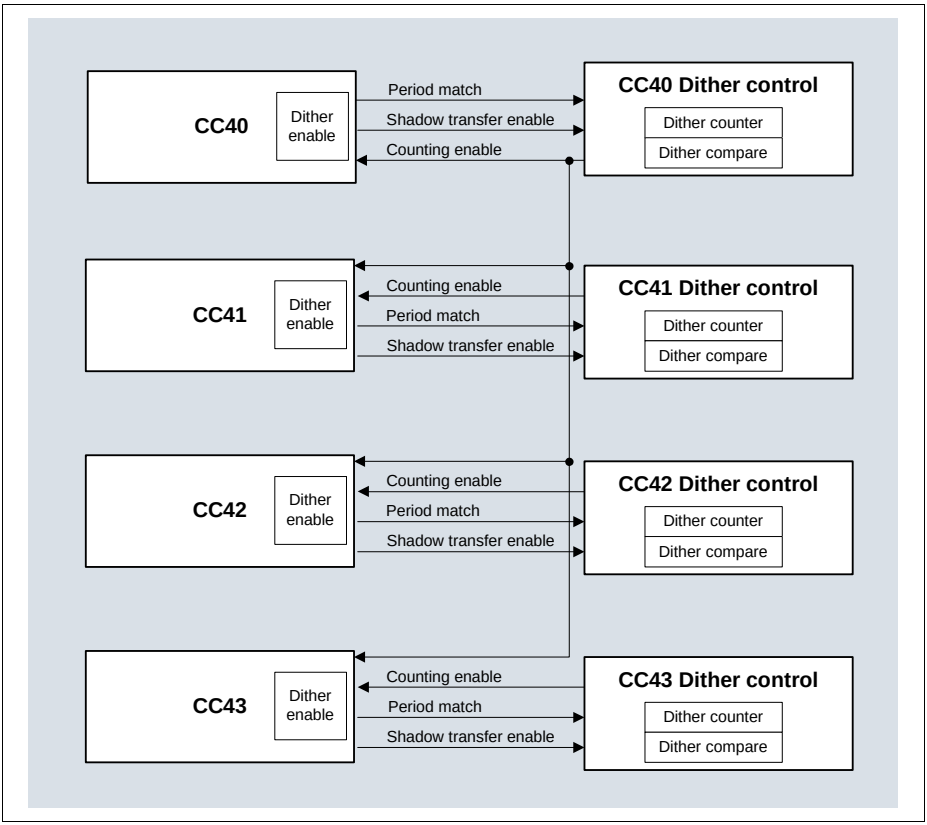



**Figure 19-49 Dither structure overview**

The dither control unit contains a 4 bit counter and a compare value. The four bit counter is incremented every time that a period match occurs. The counter works in a bit reverse mode so the distribution of increments stays uniform over 16 counter periods, see **Table 19-5**.

**Table 19-5 Dither bit reverse counter**

| counter[3] | counter[2] | counter[1] | counter[0] |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 |

The counter is then compared against a programmed value, **CC4yDIT**.DCV. If the counter value is smaller than the programmed value, a gating signal is generated that can be used to extend the period, to delay the compare or both (controlled by the **CC4yTC**.DITHE field, see **Table 19-6**) for one clock cycle.

**Table 19-6 Dither modes**

| DITHE[1] | DITH[0] | Mode |
|---|---|---|
| 0 | 0 | Dither is disabled |
| 0 | 1 | Period is increased by 1 cycle |
| 1 | 0 | Compare match is delayed by 1 cycle |
| 1 | 1 | Period is increased by 1 cycle and compare is delayed by 1 cycle |

The dither compare value also has an associated shadow register that enables concurrent update with the period/compare register of CC4y. The control logic for the dithering unit is represented on **Figure 19-50**.

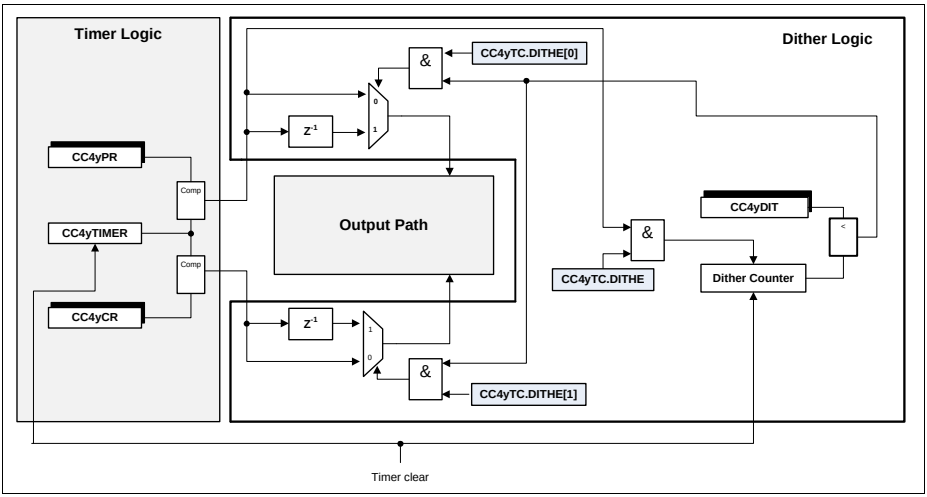



**Figure 19-50 Dither control logic**

**Figure 19-51** to **Figure 19-56** 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).

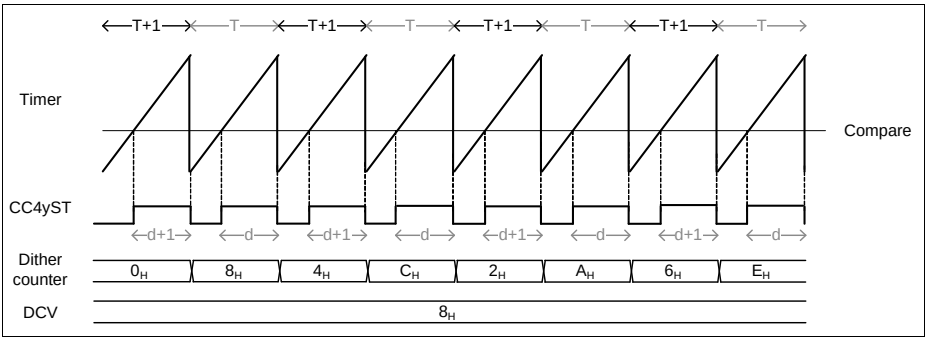



**Figure 19-51 Dither timing diagram in edge aligned - CC4yTC.DITHE = $01_B$**

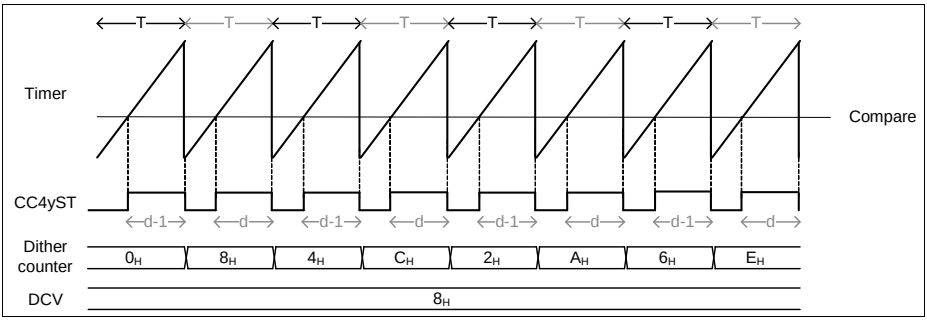



**Figure 19-52 Dither timing diagram in edge aligned - CC4yTC.DITHE = $10_B$**

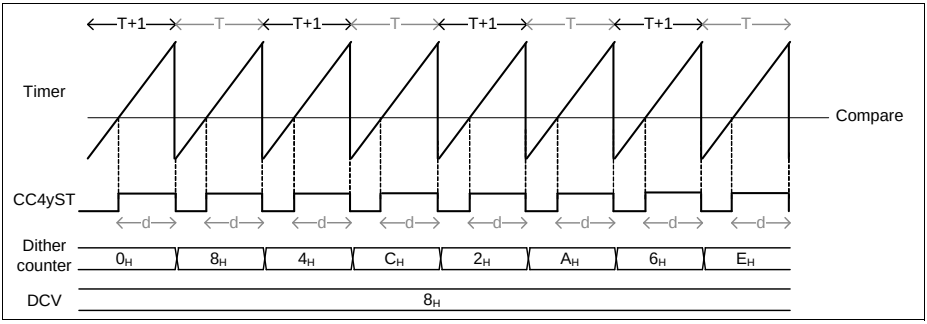



**Figure 19-53 Dither timing diagram in edge aligned - CC4yTC.DITHE = $11_B$**

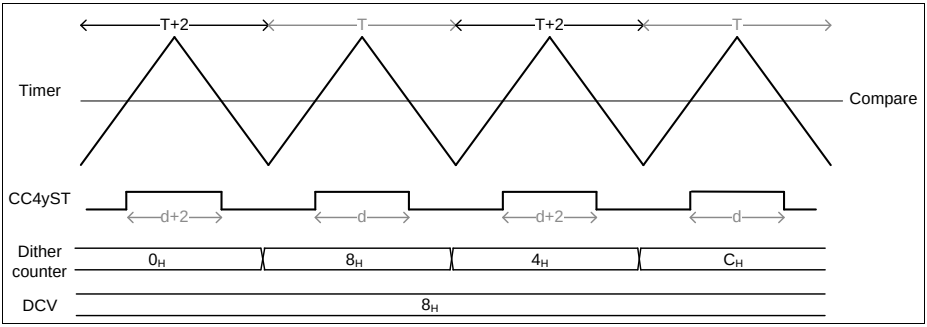



**Figure 19-54 Dither timing diagram in center aligned - CC4yTC.DITHE = $01_B$**

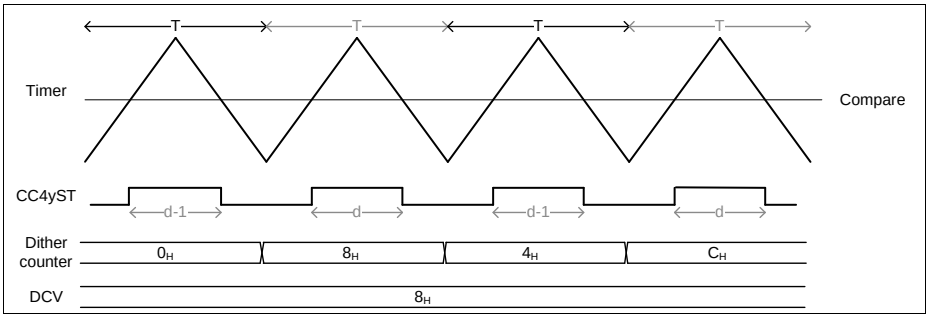



**Figure 19-55 Dither timing diagram in center aligned - CC4yTC.DITHE = $10_B$**

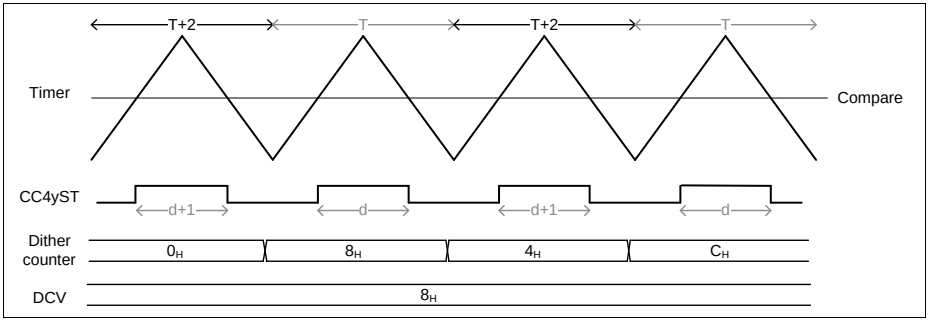



**Figure 19-56 Dither timing diagram in center aligned - CC4yTC.DITHE = $11_B$**

*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.*

### 19.2.11 Prescaler

The CCU4 contains a 4 bit prescaler that can be used in two operating modes for each individual slice:

- normal prescaler mode
- floating prescaler mode

The run bit of the prescaler can be set/cleared by SW by writing into the registers, **GIDLC**.SPRB and **GIDLS**.CPRB respectively, and it can also be cleared by the run bit of a specific slice. With the last mechanism, the run bit of the prescaler is cleared one clock cycle after the clear of the run bit of the selected slice. To select which slice can perform this action, one should program the **GCTRL**.PRBC register.


Subject to Agreement on the Use of Product Information

### 19.2.11.1 Normal Prescaler Mode

In Normal prescaler mode the clock fed to the CC4y counter is a normal fixed division by N, accordingly to the value set in the **CC4yPSC**.PSIV register. The values for the possible division values are listed in **Table 19-7**. The **CC4yPSC**.PSIV value is only modified by a SW access. Notice that each slice has a dedicated prescaler value selector (**CC4yPSC**.PSIV), which means that the user can select different counter clocks for each Timer Slice (CC4y).

**Table 19-7 Timer clock division options**

| **CC4yPSC.PSIV** | **Resulting clock** |
|---|---|
| $0000_B$ | $f_{ccu4}$ |
| $0001_B$ | $f_{ccu4}/2$ |
| $0010_B$ | $f_{ccu4}/4$ |
| $0011_B$ | $f_{ccu4}/8$ |
| $0100_B$ | $f_{ccu4}/16$ |
| $0101_B$ | $f_{ccu4}/32$ |
| $0110_B$ | $f_{ccu4}/64$ |
| $0111_B$ | $f_{ccu4}/128$ |
| $1000_B$ | $f_{ccu4}/256$ |
| $1001_B$ | $f_{ccu4}/512$ |
| $1010_B$ | $f_{ccu4}/1024$ |
| $1011_B$ | $f_{ccu4}/2048$ |
| $1100_B$ | $f_{ccu4}/4096$ |
| $1101_B$ | $f_{ccu4}/8192$ |
| $1110_B$ | $f_{ccu4}/16384$ |
| $1111_B$ | $f_{ccu4}/32768$ |

### 19.2.11.2 Floating Prescaler Mode

The floating prescaler mode can be used individually in each slice by setting the register **CC4yTC**.FPE = $1_B$. With this mode, the user can not only achieve a better precision on the counter clock for compare operations but also reduce the SW read access for the capture mode.

The floating prescaler mode contains additionally to the initial configuration value register, **CC4yPSC**.PSIV, a compare register, **CC4yFPC**.PCMP with an associated shadow register mechanism.

**Figure 19-57** shows the structure of the prescaler in floating mode when the specific slice is in compare mode (no external signal is used for capture). In this mode, the value of the clock division is increment by $1_D$ every time that a timer overflow/underflow (overflow if in Edge Aligned Mode, underflow if in Center Aligned Mode) occurs.

In this mode, the Compare Match from the timer is AND gated with the Compare Match of the prescaler and every time that this event occurs, the value of the clock division is updated with the **CC4yPSC**.PSIV value in the immediately next timer overflow/underflow event.

The shadow transfer of the floating prescaler compare value, **CC4yFPC**.PCMP, is done following the same rules described on **Section 19.2.5.2**.

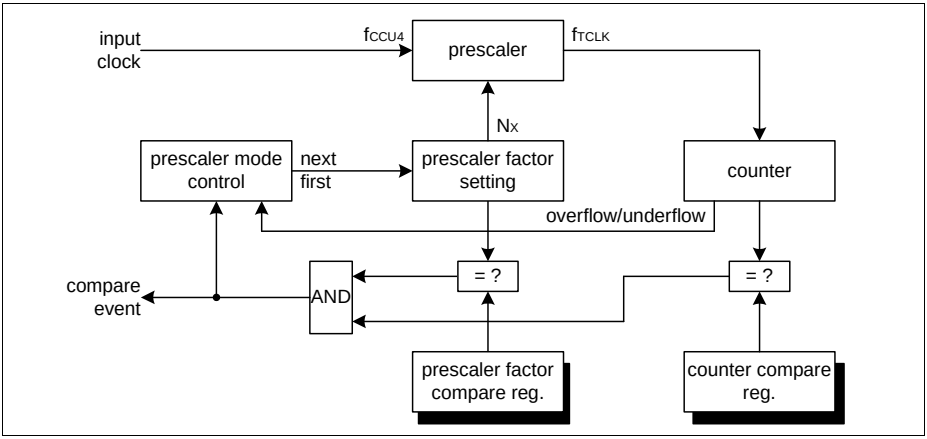



**Figure 19-57 Floating prescaler in compare mode overview**

When the specific CC4y is operating in capture mode (when at least one external signal is decoded as capture functionality), the actual value of the clock division also needs to be stored every time that a capture event occurs. The floating prescaler can have up to 4 capture registers (the maximum number of capture registers is dictated by the number of capture registers used in the specific slice).

The clock division value continues to be increment by $1_D$ every time that a timer overflow (in capture mode, the slice is always operating in Edge Aligned Mode) occurs and it is loaded with the PSIV value every time that a capture triggers is detected.

See the **Section 19.2.12.2** for a full description of the usage of the floating prescaler mode in conjunction with compare and capture modes.

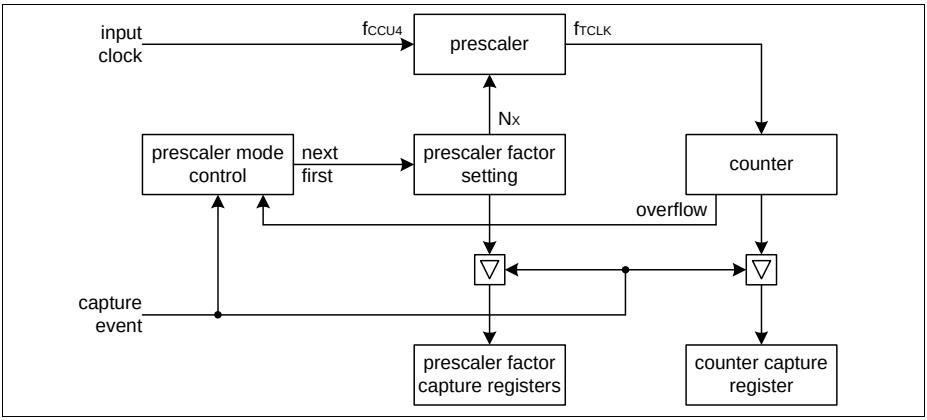



**Figure 19-58 Floating Prescaler in capture mode overview**

## 19.2.12 CCU4 Usage

### 19.2.12.1 PWM Signal Generation

The CCU4 offers a very flexible range in duty cycle configurations. This range is comprised between 0 to 100%.

To generate a PWM signal with a 100% duty cycle in Edge Aligned Mode, one should program the compare value, **CC4yCR**.CR, to $0000_H$, see **Figure 19-59**.

In the same manner a 100% duty cycle signal can be generated in Center Aligned Mode, see **Figure 19-60**.

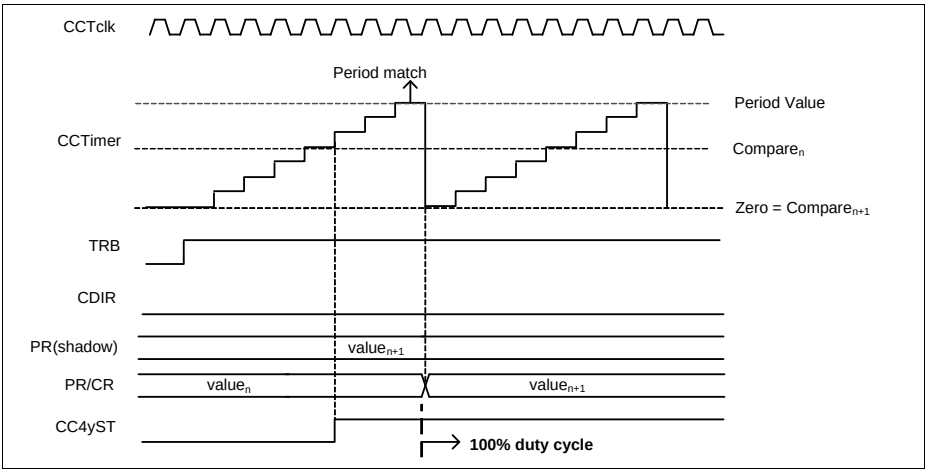



**Figure 19-59 PWM with 100% duty cycle - Edge Aligned Mode**

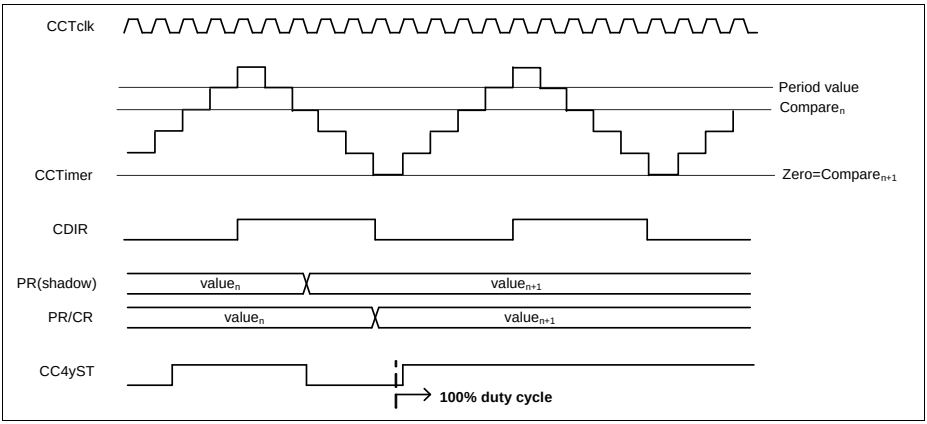



**Figure 19-60 PWM with 100% duty cycle - Center Aligned Mode**

To generate a PWM signal with 0% duty cycle in Edge Aligned Mode, the compare register should be set with the value programmed into the period value plus 1. In the case that the timer is being used with the full 16 bit capability (counting from 0 to 65535), setting a value bigger than the period value into the compare register is not possible and therefore the smallest duty cycle that can be achieved is 1/FS, see **Figure 19-61**.

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 19-62**.

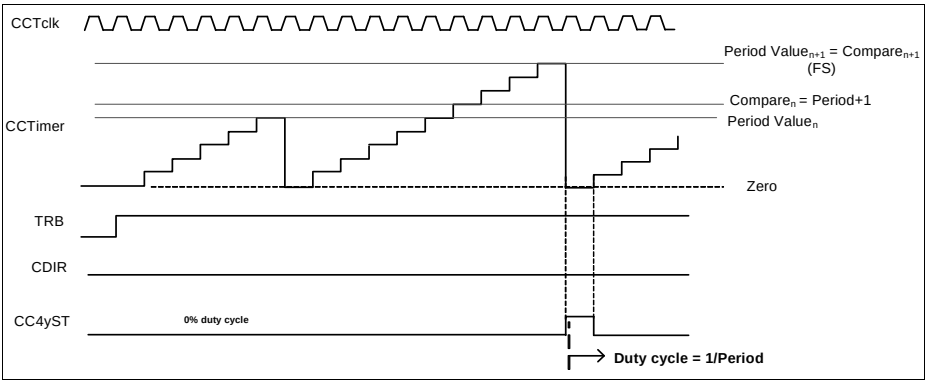



**Figure 19-61 PWM with 0% duty cycle - Edge Aligned Mode**

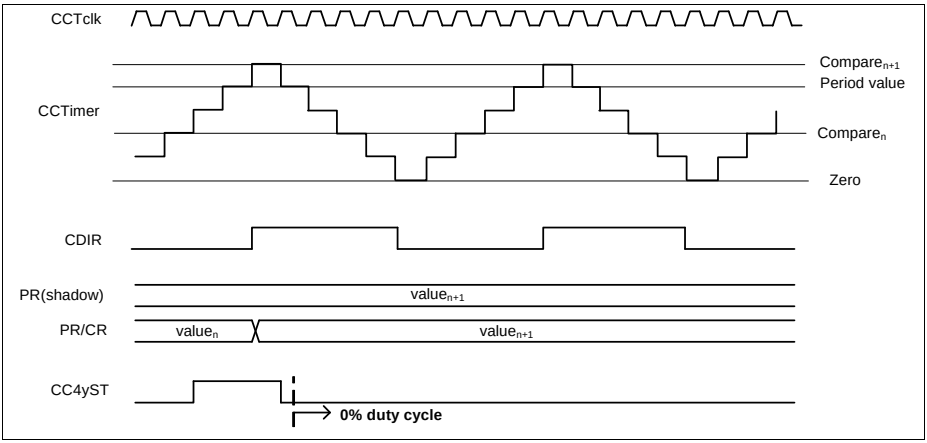



**Figure 19-62 PWM with 0% duty cycle - Center Aligned Mode**

### 19.2.12.2 Prescaler Usage

In Normal Prescaler Mode, the frequency of the $f_{tclk}$ fed to the specific CC4y is chosen from the **Table 19-7**, by setting the **CC4yPSC**.PSIV with the required value.

In Floating Prescaler Mode, the frequency of the $f_{tclk}$ can be modified over a selected timeframe, within the values specified in **Table 19-7**. This mechanism is specially useful if, when in capture mode, the dynamic of the capture triggers is very slow or unknown.

In Capture Mode, the Floating Prescaler value is incremented by 1 every time that a timer overflow happens and it is set with the initial programmed value when a capture event happens, see **Figure 19-63**.

When using the Floating Prescaler Mode in Capture Mode, the timer should be cleared each time that a capture event happens, **CC4yTC**.CAPC = $11_B$. By operating the Capture mode in conjunction with the Floating Prescaler, even for capture signals that have a periodicity bigger that 16 bits, it is possible to use just a single CCU4 Timer Slice without monitoring the interrupt event of the timer overflow, cycle by cycle. For this the user just needs to know what is the timer captured value and the actual prescaler configuration at the time that the capture event occurred. These values are contained in each CC4yCxV register.

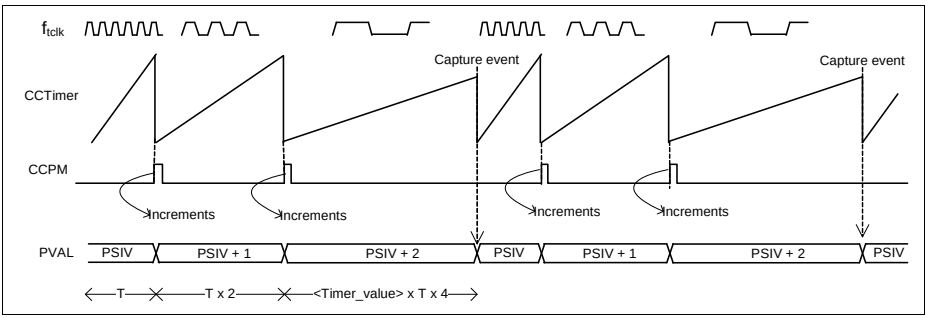



**Figure 19-63 Floating Prescaler capture mode usage**

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 19-64**, the Compare value of the Floating Prescaler was set to PSIV + 2. Every time that a timer overflow occurs, the value of the Prescaler is incremented by 1, which means that if we give $f_{tclk}$ as the reference frequency for the **CC4yPSC**.PSIV value, we have $f_{tclk}/2$ for **CC4yPSC**.PSIV + 1 and $f_{tclk}/4$ for **CC4yPSC**.PSIV + 2.

The period over time of the counter becomes:

$$Period = (1/f_{tclk} + 2/f_{tclk} + 4/f_{tclk}) / 3 \quad (19.10)$$


Subject to Agreement on the Use of Product Information

The same mechanism is used in Center Aligned Mode, but to keep the rising arcade and falling arcade always symmetrical, instead of the overflow of the timer, the underflow is used, see **Figure 19-65**.

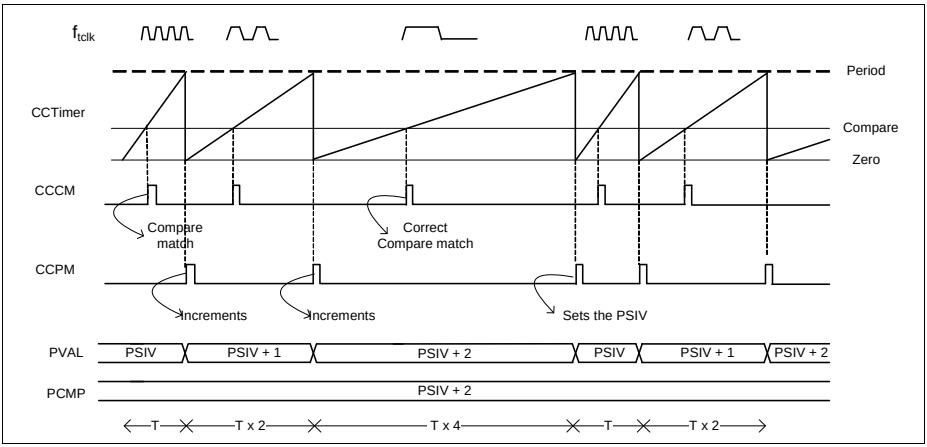



**Figure 19-64 Floating Prescaler compare mode usage - Edge Aligned**

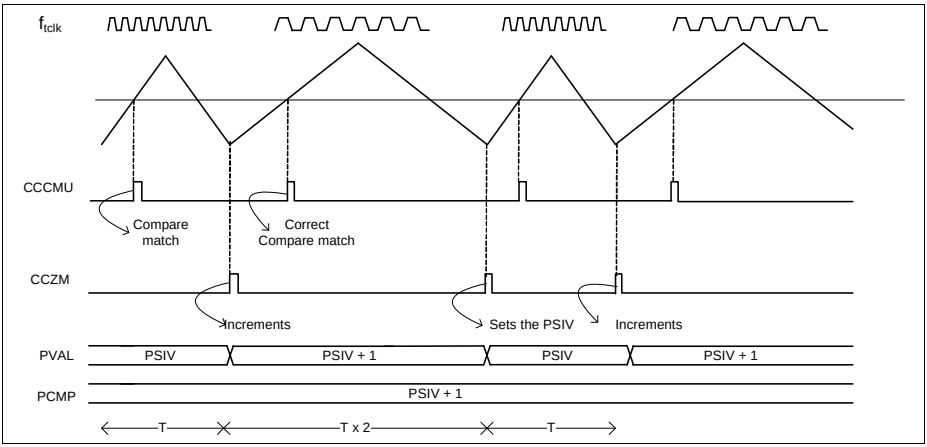



**Figure 19-65 Floating Prescaler compare mode usage - Center Aligned**

### 19.2.12.3 PWM Dither

The Dither functionality can be used to achieve a very fine precision on the periodicity of the output state in compare mode. The value set in the dither compare register, **CC4yDIT**.DCV is crosschecked against the actual value of the dither counter and every

time that the dither counter is smaller than the comparison value one of the follows actions is taken:

- The period is extended for 1 clock cycle - **CC4yTC**.DITHE = $01_B$; in edge aligned mode
- The period is extended for 2 clock cycles - **CC4yTC**.DITHE = $01_B$; in center aligned mode
- The comparison match while counting up (**CC4yTCST**.CDIR = $0_B$) is delayed (this means that the status bit is going to stay in the SET state 1 cycle less) for 1 clock cycle - **CC4yTC**.DITHE = $10_B$;
- The period is extended for 1 clock cycle and the comparison match while counting up is delayed for 1 clock cycle - **CC4yTC**.DITHE = $11_B$; in edge aligned mode
- The period is extended for 2 clock cycles and the comparison match while counting up is delayed for 1 clock cycle; center aligned mode

The bit reverse counter distributes the number programmed in the **CC4yDIT**.DCV throughout a window of 16 timer periods.

**Table 19-8** describes the bit reverse distribution versus the programmed value on the **CC4yDIT**.DCV field. The fields marked as '0' indicate that in that counter period, one of the above described actions, is going to be performed.

**Table 19-8 Bit reverse distribution**

| | **DCV** | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Dither counter** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
| 0 | 1 | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| 8 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| 4 | 1 | 1 | 1 | 1 | 1 | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| C | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | **0** | **0** | **0** |
| 2 | 1 | 1 | 1 | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| A | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | **0** | **0** | **0** | **0** | **0** |
| 6 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| E | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | **0** |
| 1 | 1 | 1 | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| 9 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | **0** | **0** | **0** | **0** | **0** | **0** |
| 5 | 1 | 1 | 1 | 1 | 1 | 1 | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| D | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | **0** | **0** |
| 3 | 1 | 1 | 1 | 1 | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| B | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | **0** | **0** | **0** | **0** |

**Table 19-8 Bit reverse distribution** (cont'd)

| | DCV | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Dither counter** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
| 7 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| F | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

The bit reverse distribution versus the programmed **CC4yDIT**.DCV value results in the following values for the Period and duty cycle:

**DITHE = $01_B$**

$Period = [(16 - DCV) \times T + DCV \times (T + 1)]/16$; in Edge Aligned Mode (19.11)

$Duty\ cycle = [(16 - DCV) \times d/T + DCV \times (d+1)/(T + 1)]/16$; in Edge Aligned Mode (19.12)

$Period = [(16 - DCV) \times T + DCV \times (T + 2)]/16$; in Center Aligned Mode (19.13)

$Duty\ cycle = [(16 - DCV) \times d/T + DCV \times (d+2)/(T + 2)]/16$; in Center Aligned Mode (19.14)

**DITHE = $10_B$**

$Period = T$ ; in Edge Aligned Mode (19.15)

$Duty\ cycle = [(16 - DCV) \times d/T + DCV \times (d-1)/T]/16$; in Edge Aligned Mode (19.16)

$Period = T$ ; in Center Aligned Mode (19.17)

$Duty\ cycle = [(16 - DCV) \times d/T + DCV \times (d-1)/T]/16$; in Center Aligned Mode (19.18)

**DITHE = $11_B$**

$Period = [(16 - DCV) \times T + DCV \times (T + 1)]/16$; in Edge Aligned Mode (19.19)

$Duty\ cycle = [(16 - DCV) \times d/T + DCV \times d/(T + 1)]/16$; in Edge Aligned Mode (19.20)

$Period = [(16 - DCV) \times T + DCV \times (T + 2)]/16$; in Center Aligned Mode (19.21)

$Duty\ cycle = [(16 - DCV) \times d/T + DCV \times (d+1)/(T + 2)]/16$; in Center Aligned Mode (19.22)

where:

$T$ - Original period of the signal, see **Section 19.2.5.1**

$d$ - Original duty cycle of the signal, see **Section 19.2.5.1**

### 19.2.12.4 Capture Mode Usage

Each Timer Slice can make use of 2 or 4 capture registers. Using only 2 capture registers means that only 1 Event was linked to a captured trigger. To use the four capture registers, both capture triggers need to be mapped into an Event (it can be the same signal with different edges selected or two different signals) or the **CC4yTC**.SCE field needs to be set to 1, which enables the linking of the 4 capture registers.

The internal slice mechanism for capturing is the same for the capture trigger 1 or capture trigger 0.

**Different Capture Events - SCE = $0_B$**

Capture trigger 1 (CCcapt1) is appointed to the capture register 2, **CC4yC2V** and capture register 3, **CC4yC3V**, while trigger 0 (CCcapt0) is appointed to capture register 1, **CC4yC1V** and 0, **CC4yC0V**.

In each CCcapt0 event, the timer value is stored into **CC4yC1V** and the value of the **CC4yC1V** is transferred into the **CC4yC0V**.

In each CCcapt1 event, the timer value is stored into capture register **CC4yC3V** and the value of the capture register **CC4yC3V** is transferred into **CC4yC2V**.

The capture/transfer mechanism only happens if the specific register is not full. A capture register becomes full when receives a new value and becomes empty after the SW has read back the value.

The full flag is cleared every time that the SW reads back the **CC4yC0V**, **CC4yC1V**, **CC4yC2V** or **CC4yC3V** register. The SW can be informed of a new capture trigger by enabling the interrupt source linked to the specific Event. This means that every time that a capture is made an interrupt pulse is generated.

In the case that the Floating Prescaler Mode is being used, the actual value of the clock division is also stored in the capture register (CC4yCxV).

**Figure 19-66** 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).


Subject to Agreement on the Use of Product Information

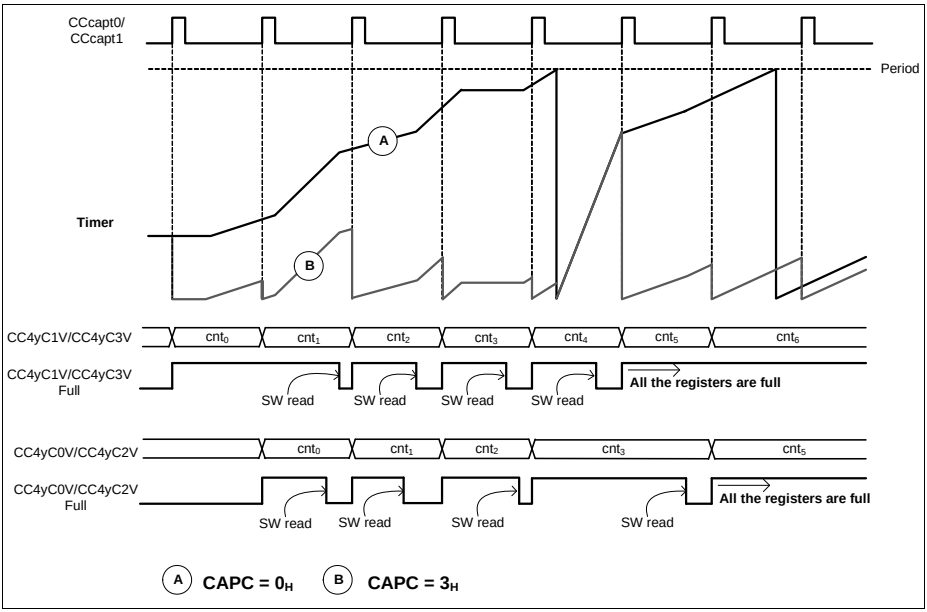



**Figure 19-66 Capture mode usage - single channel**

### Same Capture Event - SCE = $1_B$

If the **CC4yTC**.SCE is set to $1_B$, all the four capture registers are chained together, emulating a fifo with a depth of 4. In this case, only the capture trigger 1, CCcapt1, is used to perform a capture event.

As an example for this mode, one can consider the case where one Timer Slice is being used in capture mode with SCE = $1_B$, with another external signal that controls the counting. This timer slice can be incremented at different speeds, depending on the frequency of the counting signal.

An additional Timer Slice is used to control the capture trigger, dictating the time stamp for the capturing.

A simple scheme for this can be seen in **Figure 19-67**. The CC40ST output of slice 0 was used as capture trigger in the CC41 slice (active on rising and falling edge). The CC40ST output is used as known timebase marker, while the slice timer used for capture is being controlled by external events, e.g. external count.

Due to the fact that we have available 4 capture registers, every time that the SW reads back the complete set of values, 3 speed profiles can be measured.

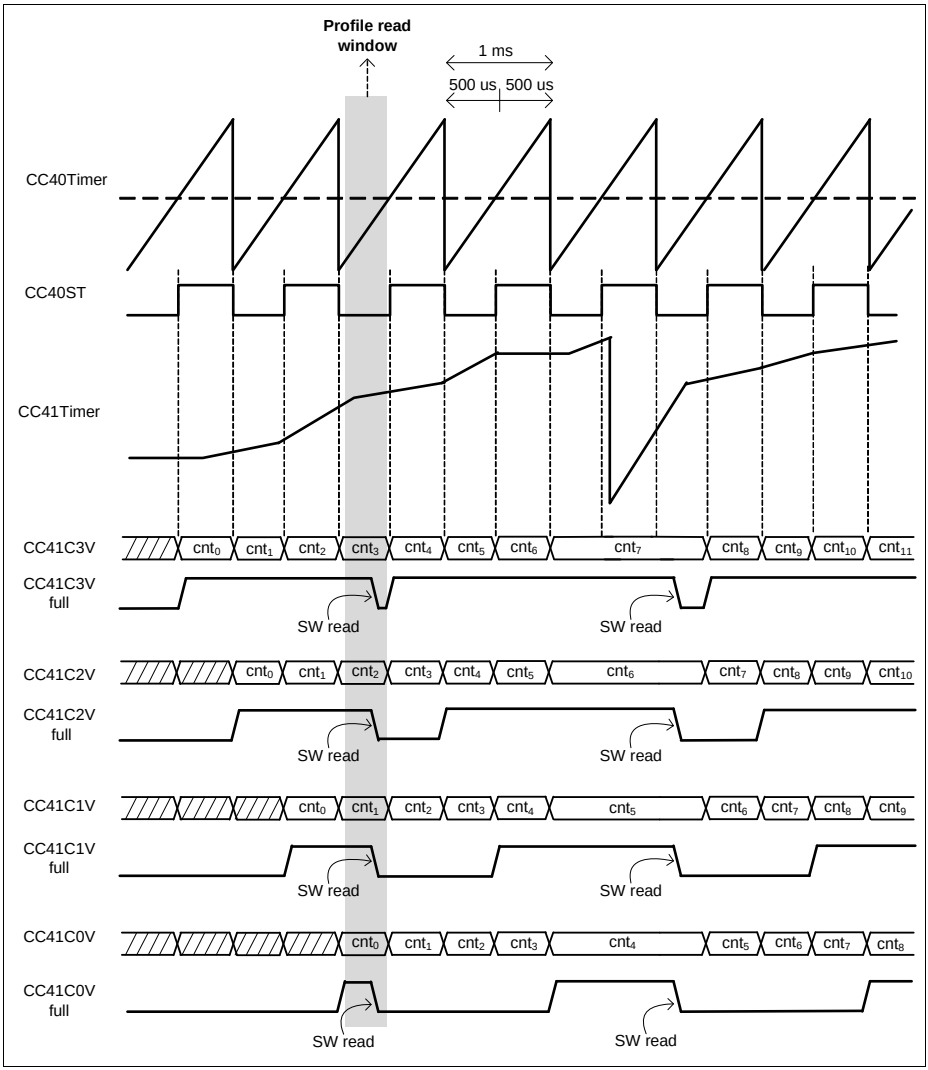



**Figure 19-67 Three Capture profiles - CC4yTC.SCE = $1_B$**

To calculate the three different profiles in **Figure 19-67**, 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 19-68**.

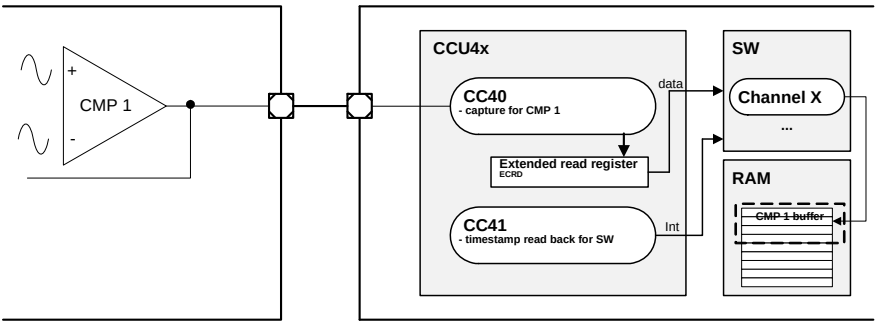



**Figure 19-68 High dynamics capturing with software controlled timestamp**

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.


Subject to Agreement on the Use of Product Information

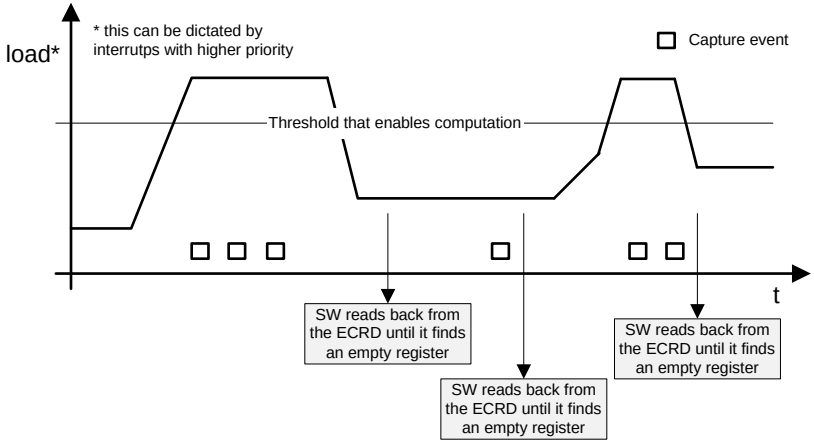



**Figure 19-69 Extended read back during high load**

### 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 19-70**.

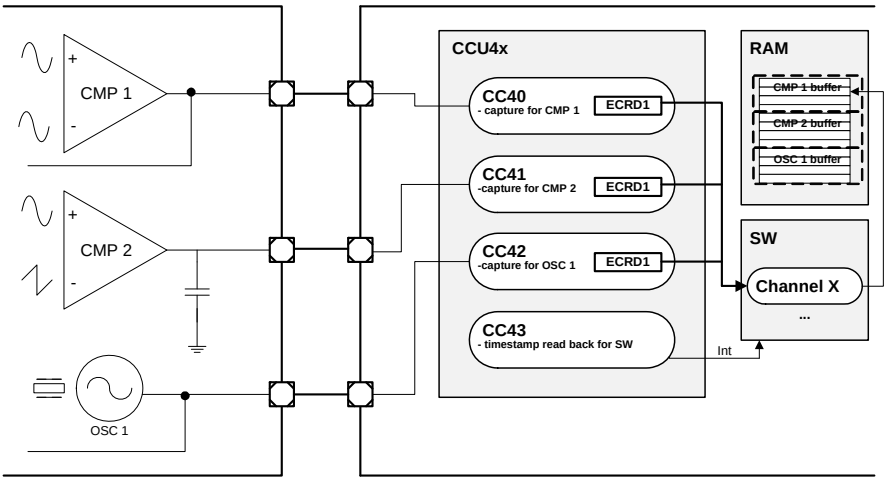



**Figure 19-70 Capture grouping with extended read back**


Subject to Agreement on the Use of Product Information

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 19-71** (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).


Subject to Agreement on the Use of Product Information

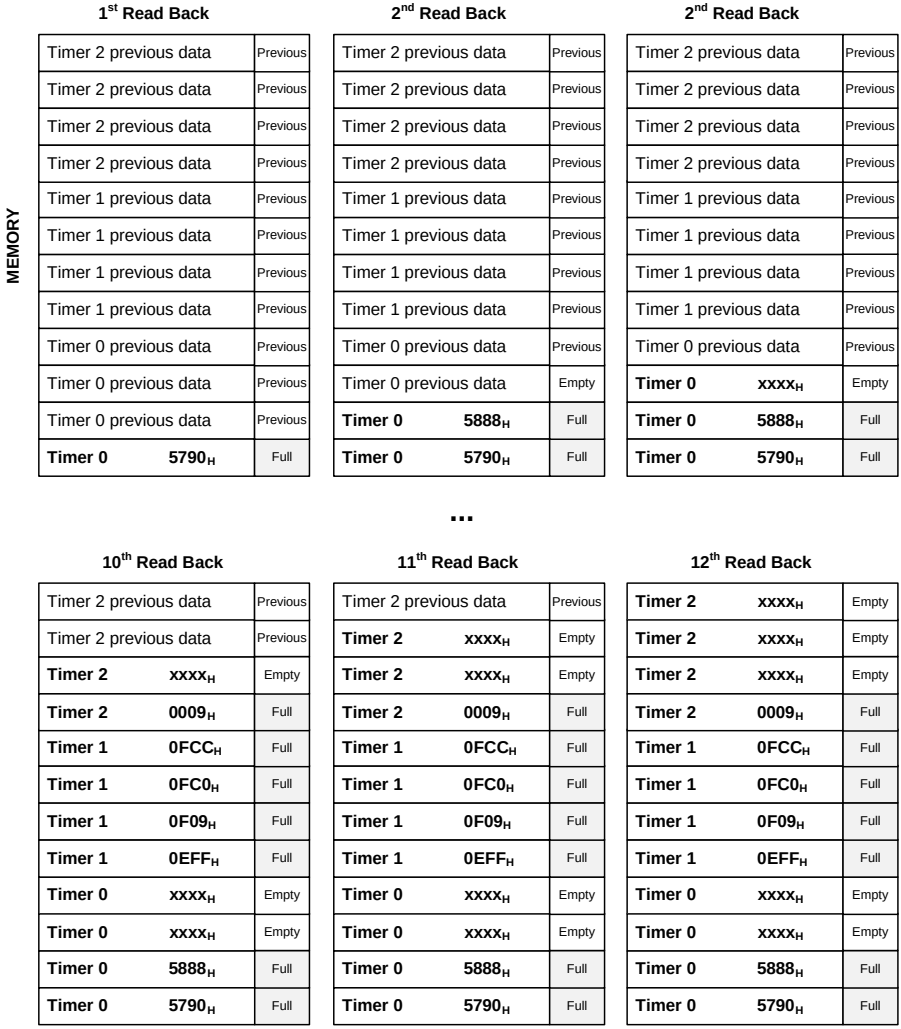



**Figure 19-71 Memory structure for extended read back**

## 19.3 Service Request Generation

Each CCU4 slice has an interrupt structure as the one in **Figure 19-72**. 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 19-9** describes the interrupt sources of each CCU4 slice.

The interrupt sources, Period Match while counting up and one Match while counting down are ORed together. The same mechanism is applied to the Compare Match while counting up and Compare Match while counting down.

The interrupt sources for the external events are directly linked with the configuration set on the **CC4yINS**.EVxEM. If an event is programmed to be active on both edges, that means that service request pulse is going to be generated when any transition on the external signal is detected. If the event is linked with a level function, the **CC4yINS**.EVxEM still can be programmed to enable a service request pulse. The TRAP event doesn't need any of extra configuration for generating the service request pulse when the slice enters the TRAP state.

**Table 19-9 Interrupt sources**

| Signal | Description |
|---|---|
| CCINEV0_E | Event 0 edge(s) information from event selector. Used when an external signal should trigger an interrupt. |
| CCINEV1_E | Event 1 edge(s) information from event selector. Used when an external signal should trigger an interrupt. |
| CCINEV2_E | Event 2 edge(s) information from event selector. Used when an external signal should trigger an interrupt. |
| CCPM_U | Period Match while counting up |
| CCCM_U | Compare Match while counting up |
| CCCM_D | Compare Match while counting down |
| CCOM_D | One Match while counting down |
| Trap state set | Entering Trap State. Will set the E2AS |


Subject to Agreement on the Use of Product Information

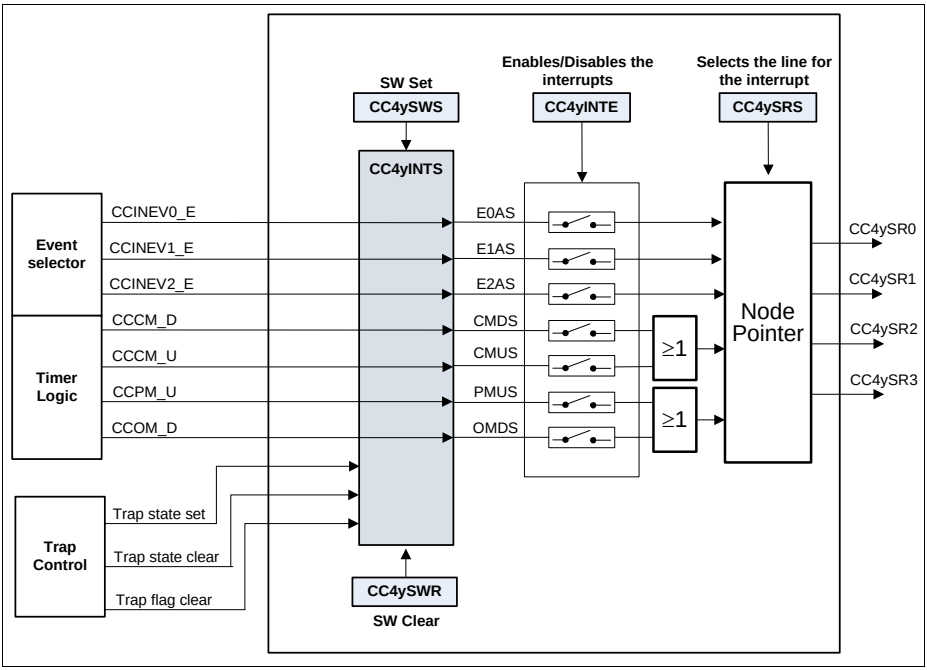



**Figure 19-72 Slice interrupt structure overview**

Each of the interrupt events can then be forwarded to one of the slice's four service request lines, **Figure 19-73**. The value set on the **CC4ySRS** controls which interrupt event is mapped into which service request line.

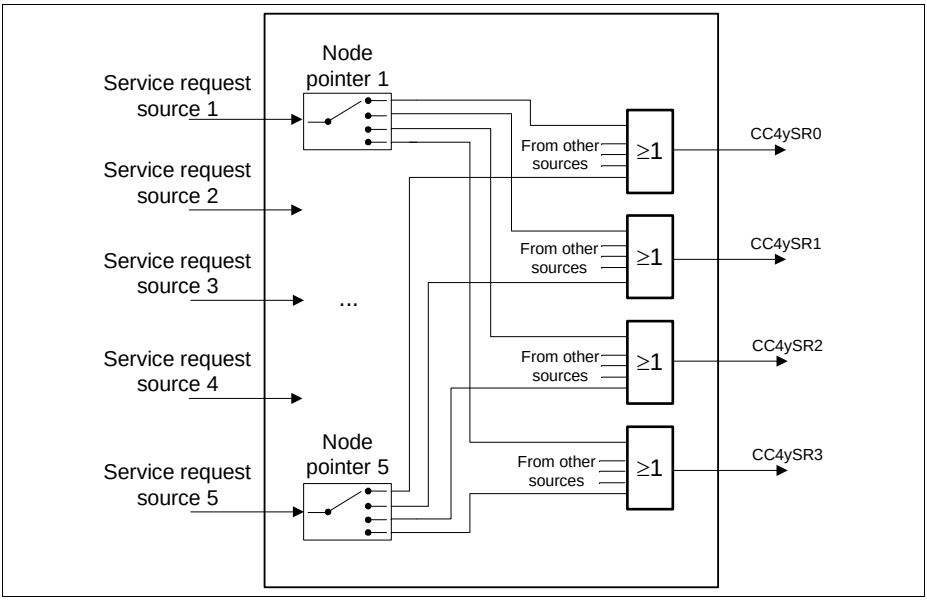



**Figure 19-73 Slice Interrupt Node Pointer overview**

The four service request lines of each slice are OR together inside the kernel of the CCU4, see **Figure 19-74**. This means that there are only four service request lines per CCU4, that can have in each line interrupt requests coming from different slices.

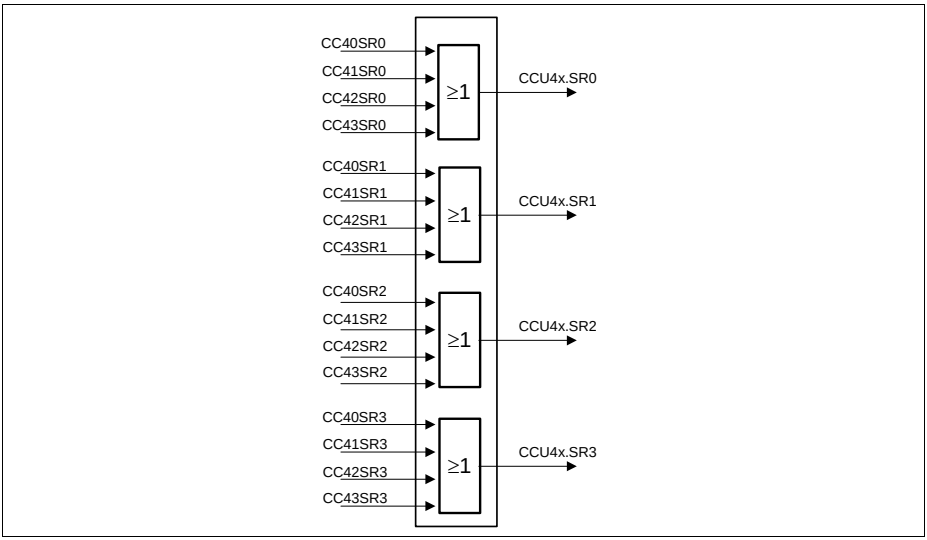



**Figure 19-74 CCU4 service request overview**

## 19.4 Debug Behavior

In suspend mode, the functional clocks for all slices as well the prescaler are stopped. The registers can still be accessed by the CPU (read only). This mode is useful for debugging purposes, e.g. where the current device status should be frozen in order to get a snapshot of the internal values. In suspend mode, all the slice 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.

## 19.5 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, $f_{CCU4}$.

### 19.5.1 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 $f_{MCLK}$.


Subject to Agreement on the Use of Product Information

It is possible to disable the module clock for the CCU4 via the **GSTAT** register, nevertheless, there may be a dependency of the $f_{ccu4}$ through the different CCU4 instances. One should address the SCU Chapter for a complete description of the product clock scheme.

If module clock dependencies exist through different IP instances, then one can disable the module clock internally inside the specific CCU4, by disabling the prescaler (**GSTAT**.PRB = $0_B$).

### External Clock

It is possible to use an external clock as source for the prescaler, and consequently for all the timer Slices, CC4y. This external source can be connected to one of the CCU4x.CLK[C...A] inputs.

This external source is nevertheless synchronized against $f_{ccu4}$.

**Table 19-10 External clock operating conditions**

| Parameter | Symbol | Values | | | Unit | Note / Test Condition |
|---|---|---|---|---|---|---|
| | | Min. | Typ. | Max. | | |
| Frequency | $f_{eclk}$ | – | – | $f_{ccu4}/4$ | MHz | |
| ON time | $ton_{eclk}$ | $2T_{ccu4}$ 1)2) | – | – | ns | |
| OFF time | $toff_{eclk}$ | $2T_{ccu4}$ 1)2) | – | – | ns | Only the rising edge is used |

1) Only valid if the signal was not previously synchronized/generated with the fccu4 clock (or a synchronous clock)

2) 50% duty cycle is not obligatory

## 19.5.2 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**).

## 19.6 Initialization and System Dependencies

### 19.6.1 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 **Section 19.2.7.1**.

### 19.6.2 System Dependencies

Each CCU4 may have different dependencies regarding module and bus clock frequencies. This dependencies should be addressed in the SCU and System Architecture Chapters.

Dependencies between several peripherals, regarding different clock operating frequencies may also exist. This should be addressed before configuring the connectivity between the CCU4 and some other peripheral.

The following topics must be taken into consideration for good CCU4 and system operation:

- CCU4 module clock must be at maximum two times faster than the module bus interface clock
- Module input triggers for the CCU4 must not exceed the module clock frequency (if the triggers are generated internally in the device)
- Module input triggers for the CCU4 must not exceed the frequency dictated in **Section 19.5.1**
- Frequency of the CCU4 outputs used as triggers/functions on other modules, must be crosschecked on the end point
- Applying and removing CCU4 from reset, can cause unwanted operations in other modules. This can occur if the modules are using CCU4 outputs as triggers/functions.


Subject to Agreement on the Use of Product Information

## 19.7 Registers

### Registers Overview

The absolute register address is calculated by adding:

Module Base Address + Offset Address

**Table 19-11 Registers Address Space**

| Module | Base Address | End Address | Note |
|---|---|---|---|
| CCU40 | $48040000_H$ | $4804FFFF_H$ | |

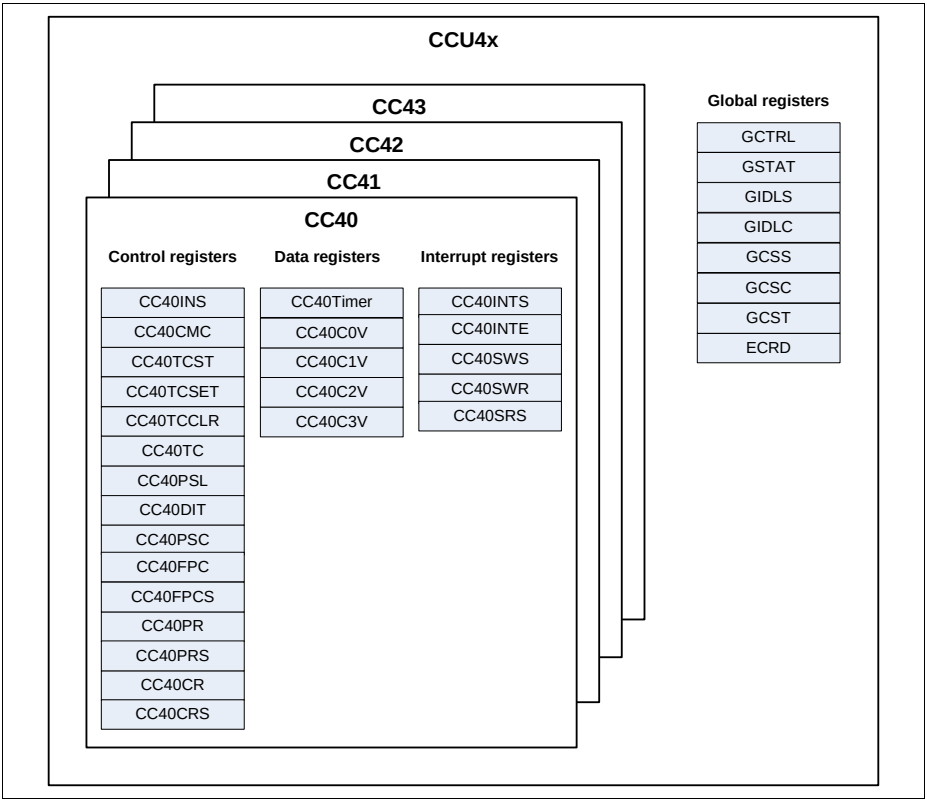



**Figure 19-75 CCU4 registers overview**


Subject to Agreement on the Use of Product Information

**Table 19-12 Register Overview of** CCU4

| Short Name | Description | Offset Addr.[1)] | Access Mode | | Description See |
|---|---|---|---|---|---|
| | | | Read | Write | |
| CCU4 Global Registers | | | | | |
| GCTRL | Module General Control Register | $0000_H$ | U, PV | U, PV | Page 19-86 |
| GSTAT | General Slice Status Register | $0004_H$ | U, PV | BE | Page 19-89 |
| GIDLS | General Idle Enable Register | $0008_H$ | U, PV | U, PV | Page 19-90 |
| GIDLC | General Idle Disable Register | $000C_H$ | U, PV | U, PV | Page 19-92 |
| GCSS | General Channel Set Register | $0010_H$ | U, PV | U, PV | Page 19-93 |
| GCSC | General Channel Clear Register | $0014_H$ | U, PV | U, PV | Page 19-95 |
| GCST | General Channel Status Register | $0018_H$ | U, PV | BE | Page 19-98 |
| MIDR | Module Identification Register | $0080_H$ | U, PV | BE | Page 19-101 |
| CC40 Registers | | | | | |
| CC40INS | Input Selector Unit Configuration | $0100_H$ | U, PV | U, PV | Page 19-101 |
| CC40CMC | Connection Matrix Configuration | $0104_H$ | U, PV | U, PV | Page 19-103 |
| CC40TST | Timer Run Status | $0108_H$ | U, PV | BE | Page 19-106 |
| CC40TCSET | Timer Run Set | $010C_H$ | U, PV | U,PV | Page 19-107 |
| CC40TCCLR | Timer Run Clear | $0110_H$ | U, PV | U, PV | Page 19-108 |
| CC40TC | General Timer Configuration | $0114_H$ | U, PV | U, PV | Page 19-109 |
| CC40PSL | Output Passive Level Configuration | $0118_H$ | U, PV | U, PV | Page 19-114 |
| CC40DIT | Dither Configuration | $011C_H$ | U, PV | BE | Page 19-114 |
| CC40DITS | Dither Shadow Register | $0120_H$ | U, PV | U, PV | Page 19-115 |
| CC40PSC | Prescaler Configuration | $0124_H$ | U, PV | U, PV | Page 19-116 |
| CC40FPC | Prescaler Compare Value | $0128_H$ | U, PV | U, PV | Page 19-117 |


Subject to Agreement on the Use of Product Information

**Table 19-12 Register Overview of** CCU4 (cont'd)

| Short Name | Description | Offset Addr.[1)] | Access Mode | | Description See |
|---|---|---|---|---|---|
| | | | Read | Write | |
| CC40FPCS | Prescaler Shadow Compare Value | $012C_H$ | U, PV | U, PV | Page 19-118 |
| CC40PR | Timer Period Value | $0130_H$ | U, PV | BE | Page 19-118 |
| CC40PRS | Timer Period Shadow Value | $0134_H$ | U, PV | U, PV | Page 19-119 |
| CC40CR | Timer Compare Value | $0138_H$ | U, PV | BE | Page 19-120 |
| CC40CRS | Timer Compare Shadow Value | $013C_H$ | U, PV | U, PV | Page 19-121 |
| CC40TIMER | Timer Current Value | $0170_H$ | U, PV | U, PV | Page 19-122 |
| CC40C0V | Capture Register 0 Value | $0174_H$ | U, PV | BE | Page 19-122 |
| CC40C1V | Capture Register 1Value | $0178_H$ | U, PV | BE | Page 19-123 |
| CC40C2V | Capture Register 2 Value | $017C_H$ | U, PV | BE | Page 19-124 |
| CC40C3V | Capture Register 3 Value | $0180_H$ | U, PV | BE | Page 19-125 |
| CC40INTS | Interrupt Status | $01A0_H$ | U, PV | BE | Page 19-126 |
| CC40INTE | Interrupt Enable | $01A4_H$ | U, PV | U, PV | Page 19-128 |
| CC40SRS | Interrupt Configuration | $01A8_H$ | U, PV | U, PV | Page 19-130 |
| CC40SWS | Interrupt Status Set | $01AC_H$ | U, PV | U, PV | Page 19-131 |
| CC40SWR | Interrupt Status Clear | $01B0_H$ | U, PV | U, PV | Page 19-133 |
| CC40ECRD0 | Extended Read Back 0 | $01B8_H$ | U, PV | BE | Page 19-134 |
| CC40ECRD1 | Extended Read Back 1 | $01BC_H$ | U, PV | BE | Page 19-136 |
| CC41 Registers | | | | | |
| CC41INS | Input Selector Unit Configuration | $0200_H$ | U, PV | U, PV | Page 19-101 |
| CC41CMC | Connection Matrix Configuration | $0204_H$ | U, PV | U, PV | Page 19-103 |
| CC41TST | Timer Run Status | $0208_H$ | U, PV | BE | Page 19-106 |
| CC41TCSET | Timer Run Set | $020C_H$ | U, PV | U,PV | Page 19-107 |
| CC41TCCLR | Timer Run Clear | $0210_H$ | U, PV | U, PV | Page 19-108 |
| CC41TC | General Timer Configuration | $0214_H$ | U, PV | U, PV | Page 19-109 |
| CC41PSL | Output Passive Level Configuration | $0218_H$ | U, PV | U, PV | Page 19-114 |


Subject to Agreement on the Use of Product Information

**Table 19-12 Register Overview of** CCU4 (cont'd)

| Short Name | Description | Offset Addr.[1)] | Access Mode | | Description See |
|---|---|---|---|---|---|
| | | | Read | Write | |
| CC41DIT | Dither Configuration | $021C_H$ | U, PV | BE | Page 19-114 |
| CC41DITS | Dither Shadow Register | $0220_H$ | U, PV | U, PV | Page 19-115 |
| CC41PSC | Prescaler Configuration | $0224_H$ | U, PV | U, PV | Page 19-116 |
| CC41FPC | Prescaler Compare Value | $0228_H$ | U, PV | U, PV | Page 19-117 |
| CC41FPCS | Prescaler Shadow Compare Value | $022C_H$ | U, PV | U, PV | Page 19-118 |
| CC41PR | Timer Period Value | $0230_H$ | U, PV | BE | Page 19-118 |
| CC41PRS | Timer Period Shadow Value | $0234_H$ | U, PV | U, PV | Page 19-119 |
| CC41CR | Timer Compare Value | $0238_H$ | U, PV | BE | Page 19-120 |
| CC41CRS | Timer Compare Shadow Value | $023C_H$ | U, PV | U, PV | Page 19-121 |
| CC41TIMER | Timer Current Value | $0270_H$ | U, PV | U, PV | Page 19-122 |
| CC41C0V | Capture Register 0 Value | $0274_H$ | U, PV | BE | Page 19-122 |
| CC41C1V | Capture Register 1Value | $0278_H$ | U, PV | BE | Page 19-123 |
| CC41C2V | Capture Register 2 Value | $027C_H$ | U, PV | BE | Page 19-124 |
| CC41C3V | Capture Register 3 Value | $0280_H$ | U, PV | BE | Page 19-125 |
| CC41INTS | Interrupt Status | $02A0_H$ | U, PV | BE | Page 19-126 |
| CC41INTE | Interrupt Enable | $02A4_H$ | U, PV | U, PV | Page 19-128 |
| CC41SRS | Interrupt Configuration | $02A8_H$ | U, PV | U, PV | Page 19-130 |
| CC41SWS | Interrupt Status Set | $02AC_H$ | U, PV | U, PV | Page 19-131 |
| CC41SWR | Interrupt Status Clear | $02B0_H$ | U, PV | U, PV | Page 19-133 |
| CC41ECRD0 | Extended Read Back 0 | $02B8_H$ | U, PV | BE | Page 19-134 |
| CC41ECRD1 | Extended Read Back 1 | $02BC_H$ | U, PV | BE | Page 19-136 |
| CC42 Registers | | | | | |
| CC42INS | Input Selector Unit Configuration | $0300_H$ | U, PV | U, PV | Page 19-101 |
| CC42CMC | Connection Matrix Configuration | $0304_H$ | U, PV | U, PV | Page 19-103 |
| CC42TST | Timer Run Status | $0308_H$ | U, PV | BE | Page 19-106 |
| CC42TCSET | Timer Run Set | $030C_H$ | U, PV | U,PV | Page 19-107 |

**Table 19-12 Register Overview of** CCU4 (cont'd)

| Short Name | Description | Offset Addr.[1)] | Access Mode | | Description See |
|---|---|---|---|---|---|
| | | | Read | Write | |
| CC42TCCLR | Timer Run Clear | $0310_H$ | U, PV | U, PV | Page 19-108 |
| CC42TC | General Timer Configuration | $0314_H$ | U, PV | U, PV | Page 19-109 |
| CC42PSL | Output Passive Level Configuration | $0318_H$ | U, PV | U, PV | Page 19-114 |
| CC42DIT | Dither Configuration | $031C_H$ | U, PV | BE | Page 19-114 |
| CC42DITS | Dither Shadow Register | $0320_H$ | U, PV | U, PV | Page 19-115 |
| CC42PSC | Prescaler Configuration | $0324_H$ | U, PV | U, PV | Page 19-116 |
| CC42FPC | Prescaler Compare Value | $0328_H$ | U, PV | U, PV | Page 19-117 |
| CC42FPCS | Prescaler Shadow Compare Value | $032C_H$ | U, PV | U, PV | Page 19-118 |
| CC42PR | Timer Period Value | $0330_H$ | U, PV | BE | Page 19-118 |
| CC42PRS | Timer Period Shadow Value | $0334_H$ | U, PV | U, PV | Page 19-119 |
| CC42CR | Timer Compare Value | $0338_H$ | U, PV | BE | Page 19-120 |
| CC42CRS | Timer Compare Shadow Value | $033C_H$ | U, PV | U, PV | Page 19-121 |
| CC42TIMER | Timer Current Value | $0370_H$ | U, PV | U, PV | Page 19-122 |
| CC42C0V | Capture Register 0 Value | $0374_H$ | U, PV | BE | Page 19-122 |
| CC42C1V | Capture Register 1Value | $0378_H$ | U, PV | BE | Page 19-123 |
| CC42C2V | Capture Register 2 Value | $037C_H$ | U, PV | BE | Page 19-124 |
| CC42C3V | Capture Register 3 Value | $0380_H$ | U, PV | BE | Page 19-125 |
| CC42INTS | Interrupt Status | $03A0_H$ | U, PV | BE | Page 19-126 |
| CC42INTE | Interrupt Enable | $03A4_H$ | U, PV | U, PV | Page 19-128 |
| CC42SRS | Interrupt Configuration | $03A8_H$ | U, PV | U, PV | Page 19-130 |
| CC42SWS | Interrupt Status Set | $03AC_H$ | U, PV | U, PV | Page 19-131 |
| CC42SWR | Interrupt Status Clear | $03B0_H$ | U, PV | U, PV | Page 19-133 |
| CC42ECRD0 | Extended Read Back 0 | $03B8_H$ | U, PV | BE | Page 19-134 |
| CC42ECRD1 | Extended Read Back 1 | $03BC_H$ | U, PV | BE | Page 19-136 |
| CC43 Registers | | | | | |
| CC43INS | Input Selector Unit Configuration | $0400_H$ | U, PV | U, PV | Page 19-101 |


Subject to Agreement on the Use of Product Information

**Table 19-12 Register Overview of** CCU4 (cont'd)

| Short Name | Description | Offset Addr.[1)] | Access Mode | | Description See |
|---|---|---|---|---|---|
| | | | Read | Write | |
| CC43CMC | Connection Matrix Configuration | $0404_H$ | U, PV | U, PV | Page 19-103 |
| CC43TST | Timer Run Status | $0408_H$ | U, PV | BE | Page 19-106 |
| CC43TCSET | Timer Run Set | $040C_H$ | U, PV | U,PV | Page 19-107 |
| CC43TCCLR | Timer Run Clear | $0410_H$ | U, PV | U, PV | Page 19-108 |
| CC43TC | General Timer Configuration | $0414_H$ | U, PV | U, PV | Page 19-109 |
| CC43PSL | Output Passive Level Configuration | $0418_H$ | U, PV | U, PV | Page 19-114 |
| CC43DIT | Dither Configuration | $041C_H$ | U, PV | BE | Page 19-114 |
| CC43DITS | Dither Shadow Register | $0420_H$ | U, PV | U, PV | Page 19-115 |
| CC43PSC | Prescaler Configuration | $0424_H$ | U, PV | U, PV | Page 19-116 |
| CC43FPC | Prescaler Compare Value | $0428_H$ | U, PV | U, PV | Page 19-117 |
| CC43FPCS | Prescaler Shadow Compare Value | $042C_H$ | U, PV | U, PV | Page 19-118 |
| CC43PR | Timer Period Value | $0430_H$ | U, PV | BE | Page 19-118 |
| CC43PRS | Timer Period Shadow Value | $0434_H$ | U, PV | U, PV | Page 19-119 |
| CC43CR | Timer Compare Value | $0438_H$ | U, PV | BE | Page 19-120 |
| CC43CRS | Timer Compare Shadow Value | $043C_H$ | U, PV | U, PV | Page 19-121 |
| CC43TIMER | Timer Current Value | $0470_H$ | U, PV | U, PV | Page 19-122 |
| CC43C0V | Capture Register 0 Value | $0474_H$ | U, PV | BE | Page 19-122 |
| CC43C1V | Capture Register 1Value | $0478_H$ | U, PV | BE | Page 19-123 |
| CC43C2V | Capture Register 2 Value | $047C_H$ | U, PV | BE | Page 19-124 |
| CC43C3V | Capture Register 3 Value | $0480_H$ | U, PV | BE | Page 19-125 |
| CC43INTS | Interrupt Status | $04A0_H$ | U, PV | BE | Page 19-126 |
| CC43INTE | Interrupt Enable | $04A4_H$ | U, PV | U, PV | Page 19-128 |
| CC43SRS | Interrupt Configuration | $04A8_H$ | U, PV | U, PV | Page 19-130 |
| CC43SWS | Interrupt Status Set | $04AC_H$ | U, PV | U, PV | Page 19-131 |
| CC43SWR | Interrupt Status Clear | $04B0_H$ | U, PV | U, PV | Page 19-133 |


Subject to Agreement on the Use of Product Information

**Table 19-12 Register Overview of** CCU4 (cont'd)

| Short Name | Description | Offset Addr.[1] | Access Mode | | Description See |
|---|---|---|---|---|---|
| | | | Read | Write | |
| CC43ECRD0 | Extended Read Back 0 | $04B8_H$ | U, PV | BE | Page 19-134 |
| CC43ECRD1 | Extended Read Back 1 | $04BC_H$ | U, PV | BE | Page 19-136 |

1) The absolute register address is calculated as follows:
Module Base Address + Offset Address (shown in this column)

## 19.7.1 Global Registers

### GCTRL

The register contains the global configuration fields that affect all the timer slices inside CCU4.

**GCTRL**
**Global Control Register** **($0000_H$)** **Reset Value: $00000000_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MSDE | | MSE3 | MSE2 | MSE1 | MSE0 | SUSCFG | | 0 | | PCIS | | 0 | PRBC | | |
| rw | | rw | rw | rw | rw | rw | | r | | rw | | r | rw | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **PRBC** | [2:0] | rw | **Prescaler Clear Configuration**<br>This register controls how the prescaler Run Bit and internal registers are cleared.<br>$000_B$ SW only<br>$001_B$ **GSTAT**.PRB and prescaler registers are cleared when the Run Bit of CC40 is cleared.<br>$010_B$ **GSTAT**.PRB and prescaler registers are cleared when the Run Bit of CC41 is cleared.<br>$011_B$ **GSTAT**.PRB and prescaler registers are cleared when the Run Bit of CC42 is cleared.<br>$100_B$ **GSTAT**.PRB and prescaler registers are cleared when the Run Bit of CC43 is cleared. |
| **PCIS** | [5:4] | rw | **Prescaler Input Clock Selection**<br>$00_B$ Module clock<br>$01_B$ CCU4x.ECLKA<br>$10_B$ CCU4x.ECLKB<br>$11_B$ CCU4x.ECLKC |
| **SUSCFG** | [9:8] | rw | **Suspend Mode Configuration**<br>This field controls the entering in suspend mode for all the CAPCOM4 slices.<br><br>$00_B$ Suspend request ignored. The module never enters in suspend<br>$01_B$ Stops all the running slices immediately. Safe stop is not applied.<br>$10_B$ Stops the block immediately and clamps all the outputs to PASSIVE state. Safe stop is applied.<br>$11_B$ Waits for the roll over of each slice to stop and clamp the slices outputs. Safe stop is applied. |
| **MSE0** | 10 | rw | **Slice 0 Multi Channel shadow transfer enable**<br>When this field is set, a shadow transfer of slice 0 can be requested not only by SW but also via the CCU4x.MCSS input.<br><br>$0_B$ Shadow transfer can only be requested by SW<br>$1_B$ Shadow transfer can be requested via SW and via the CCU4x.MCSS input. |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **MSE1** | 11 | rw | **Slice 1 Multi Channel shadow transfer enable**<br>When this field is set, a shadow transfer of slice 1 can be requested not only by SW but also via the CCU4x.MCSS input.<br><br>$0_B$ Shadow transfer can only be requested by SW<br>$1_B$ Shadow transfer can be requested via SW and via the CCU4x.MCSS input. |
| **MSE2** | 12 | rw | **Slice 2 Multi Channel shadow transfer enable**<br>When this field is set, a shadow transfer of slice 2 can be requested not only by SW but also via the CCU4x.MCSS input.<br><br>$0_B$ Shadow transfer can only be requested by SW<br>$1_B$ Shadow transfer can be requested via SW and via the CCU4x.MCSS input. |
| **MSE3** | 13 | rw | **Slice 3 Multi Channel shadow transfer enable**<br>When this field is set, a shadow transfer of slice 3 can be requested not only by SW but also via the CCU4x.MCSS input.<br><br>$0_B$ Shadow transfer can only be requested by SW<br>$1_B$ Shadow transfer can be requested via SW and via the CCU4x.MCSS input. |
| **MSDE** | [15:14] | rw | **Multi Channel shadow transfer request configuration**<br>This field configures the type of shadow transfer requested via the CCU4x.MCSS input. The field **CC4yTC**.MSEy needs to be set in order for this configuration to have any effect.<br><br>$00_B$ Only the shadow transfer for period and compare values is requested<br>$01_B$ Shadow transfer for the compare, period and prescaler compare values is requested<br>$10_B$ Reserved<br>$11_B$ Shadow transfer for the compare, period, prescaler and dither compare values is requested |

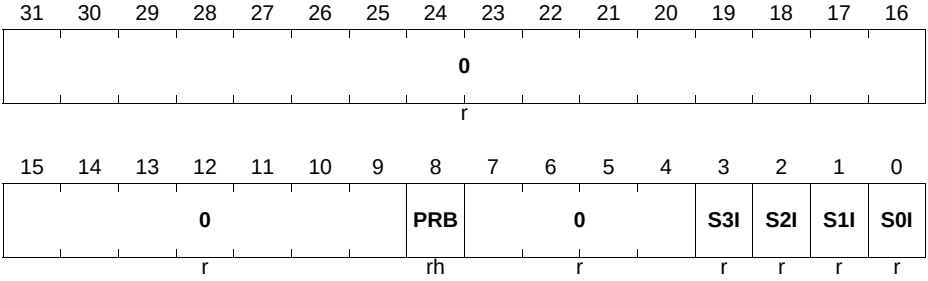

Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| 0 | 3, [7:6], [31:16] | r | **Reserved**<br>A read always returns 0. |

### GSTAT

The register contains the status of the prescaler and each timer slice (idle mode or running).

**GSTAT**
**Global Status Register** **($0004_H$)** **Reset Value: $0000000F_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **S0I** | 0 | r | **CC40 IDLE status**<br>This bit indicates if the CC40 slice is in IDLE mode or not. In IDLE mode the clocks for the CC40 slice are stopped.<br>$0_B$ Running<br>$1_B$ Idle |
| **S1I** | 1 | r | **CC41 IDLE status**<br>This bit indicates if the CC41 slice is in IDLE mode or not. In IDLE mode the clocks for the CC41 slice are stopped.<br>$0_B$ Running<br>$1_B$ Idle |

| Field | Bits | Type | Description |
|---|---|---|---|
| **S2I** | 2 | r | **CC42 IDLE status**<br>This bit indicates if the CC42 slice is in IDLE mode or not. In IDLE mode the clocks for the CC42 slice are stopped.<br>$0_B$ Running<br>$1_B$ Idle |
| **S3I** | 3 | r | **CC43 IDLE status**<br>This bit indicates if the CC43 slice is in IDLE mode or not. In IDLE mode the clocks for the CC43 slice are stopped.<br>$0_B$ Running<br>$1_B$ Idle |
| **PRB** | 8 | rh | **Prescaler Run Bit**<br>$0_B$ Prescaler is stopped<br>$1_B$ Prescaler is running |
| **0** | [7:4], [31:9] | r | **Reserved**<br>Read always returns 0. |

### GIDLS

Through this register one can set the prescaler and the specific timer slices into idle mode.

**GIDLS**
**Global Idle Set** **($0008_H$)** **Reset Value: $00000000_H$**

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

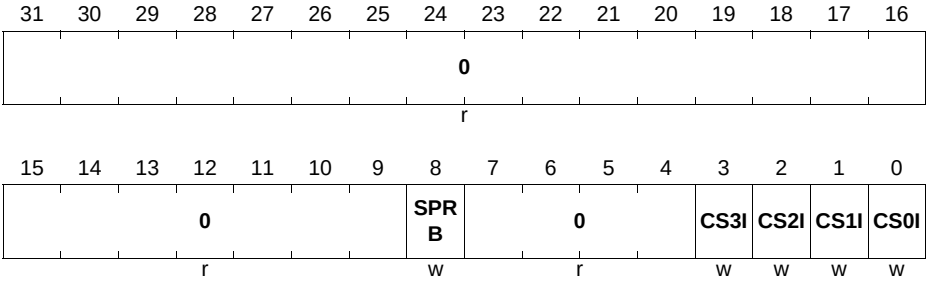
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | PSIC | CPRB | 0 | | | | SS3I | SS2I | SS1I | SS0I |
| r | | | | | | w | w | r | | | | w | w | w | w |

| Field | Bits | Type | Description |
|---|---|---|---|
| **SS0I** | 0 | w | **CC40 IDLE mode set**<br>Writing a $1_B$ to this bit sets the CC40 slice in IDLE mode. The clocks for the slice are stopped when in IDLE mode. When entering IDLE, the internal slice registers are not cleared. A read access always returns 0. |
| **SS1I** | 1 | w | **CC41 IDLE mode set**<br>Writing a $1_B$ to this bit sets the CC41 slice in IDLE mode. The clocks for the slice are stopped when in IDLE mode. When entering IDLE, the internal slice registers are not cleared. A read access always returns 0. |
| **SS2I** | 2 | w | **CC42 IDLE mode set**<br>Writing a $1_B$ to this bit sets the CC42 slice in IDLE mode. The clocks for the slice are stopped when in IDLE mode. When entering IDLE, the internal slice registers are not cleared. A read access always returns 0. |
| **SS3I** | 3 | w | **CC43 IDLE mode set**<br>Writing a $1_B$ to this bit sets the CC43 slice in IDLE mode. The clocks for the slice are stopped when in IDLE mode. When entering IDLE, the internal slice registers are not cleared. A read access always returns 0. |
| **CPRB** | 8 | w | **Prescaler Run Bit Clear**<br>Writing a $1_B$ into this register clears the Run Bit of the prescaler. Prescaler internal registers are not cleared. A read always returns 0. |
| **PSIC** | 9 | w | **Prescaler clear**<br>Writing a $1_B$ to this register clears the prescaler counter. It also loads the PSIV into the PVAL field for all Timer Slices.<br>This performs a re alignment of the timer clock for all Slices.<br>The Run Bit of the prescaler is not cleared.<br>A read always returns 0. |
| **0** | [7:4], [31:10] | r | **Reserved**<br>Read always returns 0. |

### GIDLC

Through this register one can remove the prescaler and the specific timer slices from idle mode.

**GIDLC**
**Global Idle Clear** **($000C_H$)** **Reset Value: $00000000_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **CS0I** | 0 | w | **CC40 IDLE mode clear**<br>Writing a $1_B$ to this bit removes the CC40 from IDLE mode. A read access always returns 0. |
| **CS1I** | 1 | w | **CC41 IDLE mode clear**<br>Writing a $1_B$ to this bit removes the CC41 from IDLE mode. A read access always returns 0. |
| **CS2I** | 2 | w | **CC42 IDLE mode clear**<br>Writing a $1_B$ to this bit removes the CC42 from IDLE mode. A read access always returns 0. |
| **CS3I** | 3 | w | **CC43 IDLE mode clear**<br>Writing a $1_B$ to this bit removes the CC43 from IDLE mode. A read access always returns 0. |
| **SPRB** | 8 | w | **Prescaler Run Bit Set**<br>Writing a $1_B$ into this register sets the Run Bit of the prescaler. A read always returns 0. |
| **0** | [7:4], [31:9] | r | **Reserved**<br>Read always returns 0. |

### GCSS

Through this register one can request a shadow transfer for the specific timer slice(s) and set the status bit for each of the compare channels.

**GCSS**
**Global Channel Set** **($0010_H$)** **Reset Value: $00000000_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | | | | | S3STS | S2STS | S1STS | S0STS |
| r | | | | | | | | | | | | w | w | w | w |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | S3PSE | S3DSE | S3SE | 0 | S2PSE | S2DSE | S2SE | 0 | S1PSE | S1DSE | S1SE | 0 | S0PSE | S0DSE | S0SE |
| r | w | w | w | r | w | w | w | r | w | w | w | r | w | w | w |

| Field | Bits | Type | Description |
|---|---|---|---|
| **S0SE** | 0 | w | **Slice 0 shadow transfer set enable**<br>Writing a $1_B$ to this bit will set the **GCST**.S0SS field, enabling then a shadow transfer for the Period, Compare and Passive level values.<br>A read always returns 0. |
| **S0DSE** | 1 | w | **Slice 0 Dither shadow transfer set enable**<br>Writing a $1_B$ to this bit will set the **GCST**.S0DSS field, enabling then a shadow transfer for the Dither compare value.<br>A read always returns 0. |
| **S0PSE** | 2 | w | **Slice 0 Prescaler shadow transfer set enable**<br>Writing a $1_B$ to this bit will set the **GCST**.S0PSS field, enabling then a shadow transfer for the prescaler compare value.<br>A read always returns 0. |
| **S1SE** | 4 | w | **Slice 1 shadow transfer set enable**<br>Writing a $1_B$ to this bit will set the **GCST**.S1SS field, enabling then a shadow transfer for the Period, Compare and Passive level values.<br>A read always returns 0. |
| **S1DSE** | 5 | w | **Slice 1 Dither shadow transfer set enable**<br>Writing a $1_B$ to this bit will set the **GCST**.S1DSS field, enabling then a shadow transfer for the Dither compare value.<br>A read always returns 0. |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **S1PSE** | 6 | w | **Slice 1 Prescaler shadow transfer set enable**<br>Writing a $1_B$ to this bit will set the **GCST**.S1PSS field, enabling then a shadow transfer for the prescaler compare value.<br>A read always returns 0. |
| **S2SE** | 8 | w | **Slice 2 shadow transfer set enable**<br>Writing a $1_B$ to this bit will set the **GCST**.S2SS field, enabling then a shadow transfer for the Period, Compare and Passive level values.<br>A read always returns 0. |
| **S2DSE** | 9 | w | **Slice 2 Dither shadow transfer set enable**<br>Writing a $1_B$ to this bit will set the **GCST**.S2DSS field, enabling then a shadow transfer for the Dither compare value.<br>A read always returns 0. |
| **S2PSE** | 10 | w | **Slice 2 Prescaler shadow transfer set enable**<br>Writing a $1_B$ to this bit will set the **GCST**.S2PSS field, enabling then a shadow transfer for the prescaler compare value.<br>A read always returns 0. |
| **S3SE** | 12 | w | **Slice 3 shadow transfer set enable**<br>Writing a $1_B$ to this bit will set the **GCST**.S3SS field, enabling then a shadow transfer for the Period, Compare and Passive level values.<br>A read always returns 0. |
| **S3DSE** | 13 | w | **Slice 3 Dither shadow transfer set enable**<br>Writing a $1_B$ to this bit will set the **GCST**.S3DSS field, enabling then a shadow transfer for the Dither compare value.<br>A read always returns 0. |
| **S3PSE** | 14 | w | **Slice 3 Prescaler shadow transfer set enable**<br>Writing a $1_B$ to this bit will set the **GCST**.S3PSS field, enabling then a shadow transfer for the prescaler compare value.<br>A read always returns 0. |
| **S0STS** | 16 | w | **Slice 0 status bit set**<br>Writing a $1_B$ into this field sets the status bit of slice 0 (**GCST**.CC40ST) to $1_B$.<br>A read always returns 0. |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **S1STS** | 17 | w | **Slice 1 status bit set**<br>Writing a $1_B$ into this field sets the status bit of slice 1 (**GCST**.CC41ST) to $1_B$.<br>A read always returns 0. |
| **S2STS** | 18 | w | **Slice 2 status bit set**<br>Writing a $1_B$ into this field sets the status bit of slice 2 (**GCST**.CC42ST) to $1_B$.<br>A read always returns 0. |
| **S3STS** | 19 | w | **Slice 3 status bit set**<br>Writing a $1_B$ into this field sets the status bit of slice 3 (**GCST**.CC43ST) to $1_B$.<br>A read always returns 0. |
| **0** | 3, 7, 11, 15, [31:20] | r | **Reserved**<br>Read always returns 0. |

### GCSC

Through this register one can reset a shadow transfer request for the specific timer slice and clear the status bit for each the compare channels.

**GCSC**
**Global Channel Clear** **($0014_H$)** **Reset Value: $00000000_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | | | | | S3STC | S2STC | S1STC | S0STC |
| r | | | | | | | | | | | | w | w | w | w |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | S3PSC | S3DSC | S3SC | 0 | S2PSC | S2DSC | S2SC | 0 | S1PSC | S1DSC | S1SC | 0 | S0PSC | S0DSC | S0SC |
| r | w | w | w | r | w | w | w | r | w | w | w | r | w | w | w |

| Field | Bits | Type | Description |
|---|---|---|---|
| **S0SC** | 0 | w | **Slice 0 shadow transfer clear**<br>Writing a $1_B$ to this bit will clear the **GCST**.S0SS field, canceling any pending shadow transfer for the Period, Compare and Passive level values.<br>A read always returns 0. |
| **S0DSC** | 1 | w | **Slice 0 Dither shadow transfer clear**<br>Writing a $1_B$ to this bit will clear the **GCST**.S0DSS field, canceling any pending shadow transfer for the Dither compare value.<br>A read always returns 0. |
| **S0PSC** | 2 | w | **Slice 0 Prescaler shadow transfer clear**<br>Writing a $1_B$ to this bit will clear the **GCST**.S0PSS field, canceling any pending shadow transfer for the prescaler compare value.<br>A read always returns 0. |
| **S1SC** | 4 | w | **Slice 1 shadow transfer clear**<br>Writing a $1_B$ to this bit will clear the **GCST**.S1SS field, canceling any pending shadow transfer for the Period, Compare and Passive level values.<br>A read always returns 0. |
| **S1DSC** | 5 | w | **Slice 1 Dither shadow transfer clear**<br>Writing a $1_B$ to this bit will clear the **GCST**.S1DSS field, canceling any pending shadow transfer for the Dither compare value.<br>A read always returns 0. |
| **S1PSC** | 6 | w | **Slice 1 Prescaler shadow transfer clear**<br>Writing a $1_B$ to this bit will clear the **GCST**.S1PSS field, canceling any pending shadow transfer for the prescaler compare value.<br>A read always returns 0. |
| **S2SC** | 8 | w | **Slice 2 shadow transfer clear**<br>Writing a $1_B$ to this bit will clear the **GCST**.S2SS field, canceling any pending shadow transfer for the Period, Compare and Passive level values.<br>A read always returns 0. |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **S2DSC** | 9 | w | **Slice 2 Dither shadow transfer clear**<br>Writing a $1_B$ to this bit will clear the **GCST**.S2DSS field, canceling any pending shadow transfer for the Dither compare value.<br>A read always returns 0. |
| **S2PSC** | 10 | w | **Slice 2 Prescaler shadow transfer clear**<br>Writing a $1_B$ to this bit will clear the **GCST**.S2PSS field, canceling any pending shadow transfer for the prescaler compare value.<br>A read always returns 0. |
| **S3SC** | 12 | w | **Slice 3 shadow transfer clear**<br>Writing a $1_B$ to this bit will clear the **GCST**.S3SS field, canceling any pending shadow transfer for the Period, Compare and Passive level values.<br>A read always returns 0. |
| **S3DSC** | 13 | w | **Slice 3 Dither shadow transfer clear**<br>Writing a $1_B$ to this bit will clear the **GCST**.S3DSS field, canceling any pending shadow transfer for the Dither compare value.<br>A read always returns 0. |
| **S3PSC** | 14 | w | **Slice 3 Prescaler shadow transfer clear**<br>Writing a $1_B$ to this bit will clear the **GCST**.S3PSS field, canceling any pending shadow transfer for the prescaler compare value.<br>A read always returns 0. |
| **S0STC** | 16 | w | **Slice 0 status bit clear**<br>Writing a $1_B$ into this field clears the status bit of slice 0 (**GCST**.CC40ST) to $0_B$.<br>A read always returns 0. |
| **S1STC** | 17 | w | **Slice 1 status bit clear**<br>Writing a $1_B$ into this field clears the status bit of slice 1 (**GCST**.CC41ST) to $0_B$.<br>A read always returns 0. |
| **S2STC** | 18 | w | **Slice 2 status bit clear**<br>Writing a $1_B$ into this field clears the status bit of slice 2 (**GCST**.CC42ST) to $0_B$.<br>A read always returns 0. |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **S3STC** | 19 | w | **Slice 3 status bit clear**<br>Writing a $1_B$ into this field clears the status bit of slice 3 (**GCST**.CC43ST) to $0_B$.<br>A read always returns 0. |
| **0** | 3, 7, 11, 15, [31:20] | r | **Reserved**<br>Read always returns 0. |

### GCST

This register holds the information of the shadow transfer requests and of each timer slice status bit.

**GCST**
**Global Channel Status** **($0018_H$)** **Reset Value: $00000000_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | | | | | | | | | | | | **CC43ST** | **CC42ST** | **CC41ST** | **CC40ST** |
| r | | | | | | | | | | | | rh | rh | rh | rh |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **S3PSS** | **S3DSS** | **S3SS** | **0** | **S2PSS** | **S2DSS** | **S2SS** | **0** | **S1PSS** | **S1DSS** | **S1SS** | **0** | **S0PSS** | **S0DSS** | **S0SS** |
| r | rh | rh | rh | r | rh | rh | rh | r | rh | rh | rh | r | rh | rh | rh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **S0SS** | 0 | rh | **Slice 0 shadow transfer status**<br>$0_B$ Shadow transfer has not been requested<br>$1_B$ Shadow transfer has been requested<br>This field is cleared by HW after the requested shadow transfer has been executed. |
| **S0DSS** | 1 | rh | **Slice 0 Dither shadow transfer status**<br>$0_B$ Dither shadow transfer has not been requested<br>$1_B$ Dither shadow transfer has been requested<br>This field is cleared by HW after the requested shadow transfer has been executed. |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **S0PSS** | 2 | rh | **Slice 0 Prescaler shadow transfer status**<br>$0_B$ Prescaler shadow transfer has not been requested<br>$1_B$ Prescaler shadow transfer has been requested<br>This field is cleared by HW after the requested shadow transfer has been executed. |
| **S1SS** | 4 | rh | **Slice 1 shadow transfer status**<br>$0_B$ Shadow transfer has not been requested<br>$1_B$ Shadow transfer has been requested<br>This field is cleared by HW after the requested shadow transfer has been executed. |
| **S1DSS** | 5 | rh | **Slice 1 Dither shadow transfer status**<br>$0_B$ Dither shadow transfer has not been requested<br>$1_B$ Dither shadow transfer has been requested<br>This field is cleared by HW after the requested shadow transfer has been executed. |
| **S1PSS** | 6 | rh | **Slice 1 Prescaler shadow transfer status**<br>$0_B$ Prescaler shadow transfer has not been requested<br>$1_B$ Prescaler shadow transfer has been requested<br>This field is cleared by HW after the requested shadow transfer has been executed. |
| **S2SS** | 8 | rh | **Slice 2 shadow transfer status**<br>$0_B$ Shadow transfer has not been requested<br>$1_B$ Shadow transfer has been requested<br>This field is cleared by HW after the requested shadow transfer has been executed. |
| **S2DSS** | 9 | rh | **Slice 2 Dither shadow transfer status**<br>$0_B$ Dither shadow transfer has not been requested<br>$1_B$ Dither shadow transfer has been requested<br>This field is cleared by HW after the requested shadow transfer has been executed. |


Subject to Agreement on the Use of Product Information

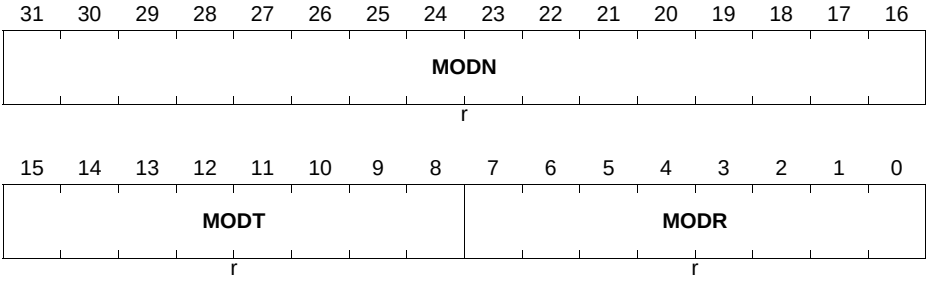
| Field | Bits | Type | Description |
|---|---|---|---|
| **S2PSS** | 10 | rh | **Slice 2 Prescaler shadow transfer status**<br>$0_B$ Prescaler shadow transfer has not been requested<br>$1_B$ Prescaler shadow transfer has been requested<br>This field is cleared by HW after the requested shadow transfer has been executed. |
| **S3SS** | 12 | rh | **Slice 3 shadow transfer status**<br>$0_B$ Shadow transfer has not been requested<br>$1_B$ Shadow transfer has been requested<br>This field is cleared by HW after the requested shadow transfer has been executed. |
| **S3DSS** | 13 | rh | **Slice 3 Dither shadow transfer status**<br>$0_B$ Dither shadow transfer has not been requested<br>$1_B$ Dither shadow transfer has been requested<br>This field is cleared by HW after the requested shadow transfer has been executed. |
| **S3PSS** | 14 | rh | **Slice 3 Prescaler shadow transfer status**<br>$0_B$ Prescaler shadow transfer has not been requested<br>$1_B$ Prescaler shadow transfer has been requested<br>This field is cleared by HW after the requested shadow transfer has been executed. |
| **CC40ST** | 16 | rh | **Slice 0 status bit** |
| **CC41ST** | 17 | rh | **Slice 1 status bit** |
| **CC42ST** | 18 | rh | **Slice 2 status bit** |
| **CC43ST** | 19 | rh | **Slice 3 status bit** |
| **0** | 3, 7, 11, 15, [31:20] | r | **Reserved**<br>Read always returns 0. |

### MIDR

This register contains the module identification number.


Subject to Agreement on the Use of Product Information

**MIDR**
**Module Identification** **($0080_H$)** **Reset Value: $00A6C0XX_H$**



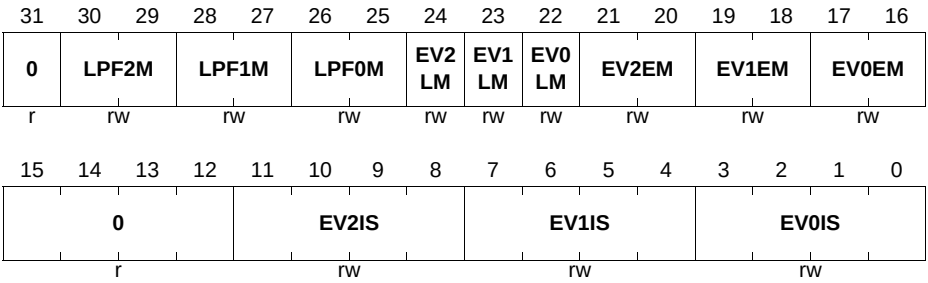

| Field | Bits | Type | Description |
|---|---|---|---|
| **MODR** | [7:0] | r | **Module Revision**<br>This bit field indicates the revision number of the module implementation (depending on the design step). The given value of $00_H$ is a placeholder for the actual number. |
| **MODT** | [15:8] | r | **Module Type** |
| **MODN** | [31:16] | r | **Module Number** |

## 19.7.2 Slice (CC4y) Registers

### CC4yINS

The register contains the configuration for the input selector.

**CC4yINS (y = 0 - 3)**
**Input Selector Configuration** **($0100_H + 0100_H * y$)** **Reset Value: $00000000_H$**





Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **EV0IS** | [3:0] | rw | **Event 0 signal selection**<br>This field selects which pins is used for the event 0.<br>$0000_B$ CCU4x.INyA<br>$0001_B$ CCU4x.INyB<br>$0010_B$ CCU4x.INyC<br>$0011_B$ CCU4x.INyD<br>$0100_B$ CCU4x.INyE<br>$0101_B$ CCU4x.INyF<br>$0110_B$ CCU4x.INyG<br>$0111_B$ CCU4x.INyH<br>$1000_B$ CCU4x.INyI<br>$1001_B$ CCU4x.INyJ<br>$1010_B$ CCU4x.INyK<br>$1011_B$ CCU4x.INyL<br>$1100_B$ CCU4x.INyM<br>$1101_B$ CCU4x.INyN<br>$1110_B$ CCU4x.INyO<br>$1111_B$ CCU4x.INyP |
| **EV1IS** | [7:4] | rw | **Event 1 signal selection**<br>Same as EV0IS description |
| **EV2IS** | [11:8] | rw | **Event 2 signal selection**<br>Same as EV0IS description |
| **EV0EM** | [17:16] | rw | **Event 0 Edge Selection**<br>$00_B$ No action<br>$01_B$ Signal active on rising edge<br>$10_B$ Signal active on falling edge<br>$11_B$ Signal active on both edges |
| **EV1EM** | [19:18] | rw | **Event 1 Edge Selection**<br>Same as EV0EM description |
| **EV2EM** | [21:20] | rw | **Event 2 Edge Selection**<br>Same as EV0EM description |
| **EV0LM** | 22 | rw | **Event 0 Level Selection**<br>$0_B$ Active on HIGH level<br>$1_B$ Active on LOW level |
| **EV1LM** | 23 | rw | **Event 1 Level Selection**<br>Same as EV0LM description |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **EV2LM** | 24 | rw | **Event 2 Level Selection**<br>Same as EV0LM description |
| **LPF0M** | [26:25] | rw | **Event 0 Low Pass Filter Configuration**<br>This field sets the number of consecutive counts for the Low Pass Filter of Event 0. The input signal value needs to remain stable for this number of counts ($f_{CCU4}$), so that a level/transition is accepted.<br>$00_B$ LPF is disabled<br>$01_B$ 3 clock cycles of $f_{CCU4}$<br>$10_B$ 5 clock cycles of $f_{CCU4}$<br>$11_B$ 7 clock cycles of $f_{CCU4}$ |
| **LPF1M** | [28:27] | rw | **Event 1 Low Pass Filter Configuration**<br>Same description as LPF0M |
| **LPF2M** | [30:29] | rw | **Event 2 Low Pass Filter Configuration**<br>Same description as LPF0M |
| **0** | [15:12], 31 | r | **Reserved**<br>Read always returns 0. |

### CC4yCMC

The register contains the configuration for the connection matrix.

**CC4yCMC (y = 0 - 3)**
**Connection Matrix Control** **($0104_H + 0100_H * y$)** **Reset Value: $00000000_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | | | | TCE | MOS | | TS | OFS |
| r | | | | | | | | | | | rw | rw | | rw | rw |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CNTS | | LDS | | UDS | | GATES | | CAP1S | | CAP0S | | ENDS | | STRTS | |
| rw | | rw | | rw | | rw | | rw | | rw | | rw | | rw | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **STRTS** | [1:0] | rw | **External Start Functionality Selector**<br>Selects the Event that is going to be linked with the external start functionality.<br>$00_B$ External Start Function deactivated<br>$01_B$ External Start Function triggered by Event 0<br>$10_B$ External Start Function triggered by Event 1<br>$11_B$ External Start Function triggered by Event 2 |
| **ENDS** | [3:2] | rw | **External Stop Functionality Selector**<br>Selects the Event that is going to be linked with the external stop functionality.<br>$00_B$ External Stop Function deactivated<br>$01_B$ External Stop Function triggered by Event 0<br>$10_B$ External Stop Function triggered by Event 1<br>$11_B$ External Stop Function triggered by Event 2 |
| **CAP0S** | [5:4] | rw | **External Capture 0 Functionality Selector**<br>Selects the Event that is going to be linked with the external capture for capture registers number 1 and 0.<br>$00_B$ External Capture 0 Function deactivated<br>$01_B$ External Capture 0 Function triggered by Event 0<br>$10_B$ External Capture 0 Function triggered by Event 1<br>$11_B$ External Capture 0 Function triggered by Event 2 |
| **CAP1S** | [7:6] | rw | **External Capture 1 Functionality Selector**<br>Selects the Event that is going to be linked with the external capture for capture registers number 3 and 2.<br>$00_B$ External Capture 1 Function deactivated<br>$01_B$ External Capture 1 Function triggered by Event 0<br>$10_B$ External Capture 1 Function triggered by Event 1<br>$11_B$ External Capture 1 Function triggered by Event 2 |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **GATES** | [9:8] | rw | **External Gate Functionality Selector**<br>Selects the Event that is going to be linked with the counter gating function. This function is used to gate the timer increment/decrement procedure.<br>$00_B$ External Gating Function deactivated<br>$01_B$ External Gating Function triggered by Event 0<br>$10_B$ External Gating Function triggered by Event 1<br>$11_B$ External Gating Function triggered by Event 2 |
| **UDS** | [11:10] | rw | **External Up/Down Functionality Selector**<br>Selects the Event that is going to be linked with the Up/Down counting direction control.<br>$00_B$ External Up/Down Function deactivated<br>$01_B$ External Up/Down Function triggered by Event 0<br>$10_B$ External Up/Down Function triggered by Event 1<br>$11_B$ External Up/Down Function triggered by Event 2 |
| **LDS** | [13:12] | rw | **External Timer Load Functionality Selector**<br>Selects the Event that is going to be linked with the timer load function.<br>$00_B$ - External Load Function deactivated<br>$01_B$ - External Load Function triggered by Event 0<br>$10_B$ - External Load Function triggered by Event 1<br>$11_B$ - External Load Function triggered by Event 2 |
| **CNTS** | [15:14] | rw | **External Count Selector**<br>Selects the Event that is going to be linked with the count function. The counter is going to be increment/decremented each time that a specific transition on the event is detected.<br>$00_B$ External Count Function deactivated<br>$01_B$ External Count Function triggered by Event 0<br>$10_B$ External Count Function triggered by Event 1<br>$11_B$ External Count Function triggered by Event 2 |
| **OFS** | 16 | rw | **Override Function Selector**<br>This field enables the ST bit override functionality.<br>$0_B$ Override functionality disabled<br>$1_B$ Status bit trigger override connected to Event 1; Status bit value override connected to Event 2 |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **TS** | 17 | rw | **Trap Function Selector**<br>This field enables the trap functionality.<br>$0_B$ Trap function disabled<br>$1_B$ TRAP function connected to Event 2 |
| **MOS** | [19:18] | rw | **External Modulation Functionality Selector**<br>Selects the Event that is going to be linked with the external modulation function.<br>$00_B$ - Modulation Function deactivated<br>$01_B$ - Modulation Function triggered by Event 0<br>$10_B$ - Modulation Function triggered by Event 1<br>$11_B$ - Modulation Function triggered by Event 2 |
| **TCE** | 20 | rw | **Timer Concatenation Enable**<br>This bit enables the timer concatenation with the previous slice.<br>$0_B$ Timer concatenation is disabled<br>$1_B$ Timer concatenation is enabled<br>*Note: In CC40 this field doesn't exist. This is a read only reserved field. Read access always returns 0.* |
| **0** | [31:21] | r | **Reserved**<br>A read always returns 0 |

### CC4yTCST

The register holds the status of the timer (running/stopped) and the information about the counting direction (up/down).

**CC4yTCST (y = 0 - 3)**
**Slice Timer Status** **($0108_H + 0100_H * y$)** **Reset Value: $00000000_H$**

| 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 | | | | | | | | | | | | | | CDIR | TRB |
| r | | | | | | | | | | | | | | rh | rh |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **TRB** | 0 | rh | **Timer Run Bit**<br>This field indicates if the timer is running.<br>$0_B$ Timer is stopped<br>$1_B$ Timer is running |
| **CDIR** | 1 | rh | **Timer Counting Direction**<br>This filed indicates if the timer is being increment or decremented<br>$0_B$ Timer is counting up<br>$1_B$ Timer is counting down |
| **0** | [31:2] | r | **Reserved**<br>Read always returns 0 |

### CC4yTCSET

Through this register it is possible to start the timer.

**CC4yTCSET (y = 0 - 3)**
**Slice Timer Run Set** **($010C_H + 0100_H * y$)** **Reset Value: $00000000_H$**

| 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** | | | | | | | | | | | | | | | **TRBS** |
| r | | | | | | | | | | | | | | | w |

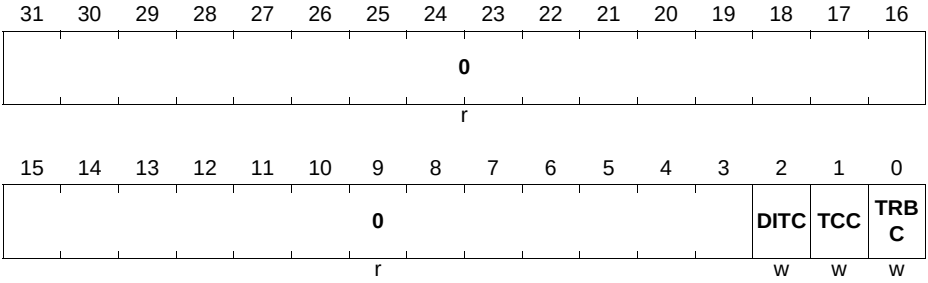
| Field | Bits | Type | Description |
|---|---|---|---|
| **TRBS** | 0 | w | **Timer Run Bit set**<br>Writing a $1_B$ into this field sets the run bit of the timer.<br>Read always returns 0. |
| **0** | [31:1] | r | **Reserved**<br>Read always returns 0 |


Subject to Agreement on the Use of Product Information

### CC4yTCCLR

Through this register it is possible to stop and clear the timer, and clearing also the dither counter

**CC4yTCCLR (y = 0 - 3)**
**Slice Timer Clear** **($0110_H + 0100_H * y$)** **Reset Value: $00000000_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **TRBC** | 0 | w | **Timer Run Bit Clear**<br>Writing a $1_B$ into this field clears the run bit of the timer. The timer is not cleared.<br>Read always returns 0. |
| **TCC** | 1 | w | **Timer Clear**<br>Writing a $1_B$ into this field clears the timer to $0000_H$.<br>Read always returns 0. |
| **DITC** | 2 | w | **Dither Counter Clear**<br>Writing a $1_B$ into this field clears the dither counter to $0_H$.<br>Read always returns 0. |
| **0** | [31:3] | r | **Reserved**<br>Read always returns 0 |

### CC4yTC

This register holds the several possible configurations for the timer operation.

**CC4yTC (y = 0 - 3)**
**Slice Timer Control** **($0114_H + 0100_H * y$)** **Reset Value: $00000000_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | MCME | EMT | EMS | TRPSW | TRPSE | 0 | | | TRAPE | FPE |
| r | | | | | | rw | rw | rw | rw | rw | r | | | rw | rw |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| DIM | DITHE | | CCS | SCE | STRM | ENDM | | 0 | CAPC | | ECM | CMOD | CLST | TSSM | TCM |
| rw | rw | | rw | rw | rw | rw | | r | rw | | rw | rh | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **TCM** | 0 | rw | **Timer Counting Mode**<br>This field controls the actual counting scheme of the timer.<br>$0_B$ Edge aligned mode<br>$1_B$ Center aligned mode<br>*Note: When using an external signal to control the counting direction, the counting scheme is always edge aligned.* |
| **TSSM** | 1 | rw | **Timer Single Shot Mode**<br>This field controls the single shot mode. This is applicable in edge and center aligned modes.<br>$0_B$ Single shot mode is disabled<br>$1_B$ Single shot mode is enabled |
| **CLST** | 2 | rw | **Shadow Transfer on Clear**<br>Setting this bit to $1_B$ enables a shadow transfer when a timer clearing action is performed.<br>Notice that the shadow transfer enable bitfields on the **GCST** register still need to be set to $1_B$ via software. |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **CMOD** | 3 | rh | **Capture Compare Mode**<br>This field indicates in which mode the slice is operating. The default value is compare mode. The capture mode is automatically set by the HW when an external signal is mapped to a capture trigger.<br>$0_B$ Compare Mode<br>$1_B$ Capture Mode |
| **ECM** | 4 | rw | **Extended Capture Mode**<br>This field control the Capture mode of the specific slice. It only has effect if the CMOD bit is $1_B$.<br><br>$0_B$ Normal Capture Mode. Clear of the Full Flag of each capture register is done by accessing the registers individually only.<br>$1_B$ Extended Capture Mode. Clear of the Full Flag of each capture register is done not only by accessing the individual registers but also by accessing the ECRD register. When reading the ECRD register, only the capture register register full flag pointed by the ECRD.VPTR is cleared. |
| **CAPC** | [6:5] | rw | **Clear on Capture Control**<br>$00_B$ Timer is never cleared on a capture event<br>$01_B$ Timer is cleared on a capture event into capture registers 2 and 3. (When SCE = $1_B$, Timer is always cleared in a capture event)<br>$10_B$ Timer is cleared on a capture event into capture registers 0 and 1. (When SCE = $1_B$, Timer is always cleared in a capture event)<br>$11_B$ Timer is always cleared in a capture event. |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **ENDM** | [9:8] | rw | **Extended Stop Function Control**<br>This field controls the extended functions of the external Stop signal.<br>$00_B$ Clears the timer run bit only (default stop)<br>$01_B$ Clears the timer only (flush)<br>$10_B$ Clears the timer and run bit (flush/stop)<br>$11_B$ Reserved<br>*Note: When using an external up/down signal the flush operation sets the timer with zero if the counter is counting up and with the Period value if the counter is being decremented.* |
| **STRM** | 10 | rw | **Extended Start Function Control**<br>This field controls the extended functions of the external Start signal.<br>$0_B$ Sets run bit only (default start)<br>$1_B$ Clears the timer and sets run bit (flush/start)<br>*Note: When using an external up/down signal the flush operation sets the timer with zero if the counter is being incremented and with the Period value if the counter is being decremented.* |
| **SCE** | 11 | rw | **Equal Capture Event enable**<br>$0_B$ Capture into **CC4yC0V**/**CC4yC1V** registers control by CCycapt0 and capture into **CC4yC3V**/**CC4yC2V** control by CCycapt1<br>$1_B$ Capture into **CC4yC0V**/**CC4yC1V** and **CC4yC3V**/**CC4yC2V** control by CCycapt1 |
| **CCS** | 12 | rw | **Continuous Capture Enable**<br>$0_B$ The capture into a specific capture register is done with the rules linked with the full flags, described at **Section 19.2.7.6**.<br>$1_B$ The capture into the capture registers is always done regardless of the full flag status (even if the register has not been read back). |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **DITHE** | [14:13] | rw | **Dither Enable**<br>This field controls the dither mode for the slice. See **Section 19.2.10**.<br>$00_B$ Dither is disabled<br>$01_B$ Dither is applied to the Period<br>$10_B$ Dither is applied to the Compare<br>$11_B$ Dither is applied to the Period and Compare |
| **DIM** | 15 | rw | **Dither input selector**<br>This fields selects if the dither control signal is connected to the dither logic of the specific slice of is connected to the dither logic of slice 0. Notice that even if this field is set to $1_B$, the field DITHE still needs to be programmed.<br>$0_B$ Slice is using its own dither unit<br>$1_B$ Slice is connected to the dither unit of slice 0. |
| **FPE** | 16 | rw | **Floating Prescaler enable**<br>Setting this bit to $1_B$ enables the floating prescaler mode.<br>$0_B$ Floating prescaler mode is disabled<br>$1_B$ Floating prescaler mode is enabled |
| **TRAPE** | 17 | rw | **TRAP enable**<br>Setting this bit to $1_B$ enables the TRAP action at the output pin. After mapping an external signal to the TRAP functionality, the user must set this field to $1_B$ to activate the effect of the TRAP on the output pin. Writing a $0_B$ into this field disables the effect of the TRAP function regardless of the state of the input signal.<br>$0_B$ TRAP functionality has no effect on the output<br>$1_B$ TRAP functionality affects the output |
| **TRPSE** | 21 | rw | **TRAP Synchronization Enable**<br>Writing a $1_B$ into this bit enables a synchronous exiting with the PWM signal of the trap state.<br>$0_B$ Exiting from TRAP state isn't synchronized with the PWM signal<br>$1_B$ Exiting from TRAP state is synchronized with the PWM signal |

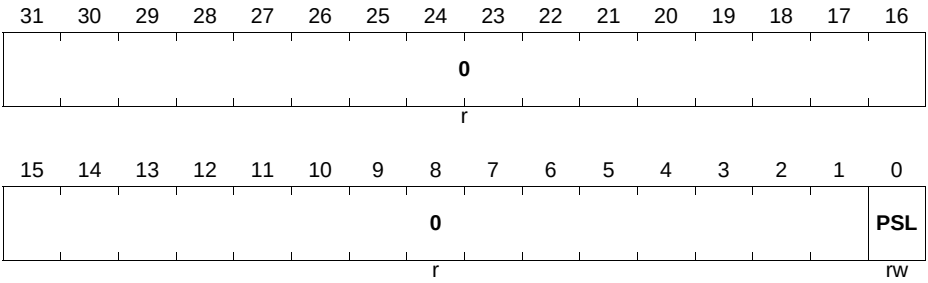

Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **TRPSW** | 22 | rw | **TRAP State Clear Control**<br>$0_B$ The slice exits the TRAP state automatically when the TRAP condition is not present<br>$1_B$ The TRAP state can only be exited by a SW request. |
| **EMS** | 23 | rw | **External Modulation Synchronization**<br>Setting this bit to $1_B$ enables the synchronization of the external modulation functionality with the PWM period.<br>$0_B$ External Modulation functionality is not synchronized with the PWM signal<br>$1_B$ External Modulation functionality is synchronized with the PWM signal |
| **EMT** | 24 | rw | **External Modulation Type**<br>This field selects if the external modulation event is clearing the CC4yST bit or if is gating the outputs.<br>$0_B$ External Modulation functionality is clearing the CC4yST bit.<br>$1_B$ External Modulation functionality is gating the outputs. |
| **MCME** | 25 | rw | **Multi Channel Mode Enable**<br>$0_B$ Multi Channel Mode is disabled<br>$1_B$ Multi Channel Mode is enabled |
| **0** | 7, [20:18], [31:26] | r | **Reserved**<br>Read always returns 0 |

### CC4yPSL

This register holds the configuration for the output passive level control.

**CC4yPSL (y = 0 - 3)**
**Passive Level Config** **($0118_H + 0100_H * y$)** **Reset Value: $00000000_H$**

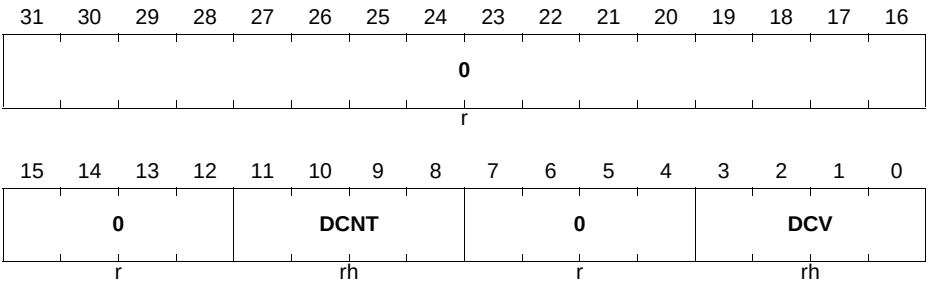



| Field | Bits | Type | Description |
|---|---|---|---|
| **PSL** | 0 | rw | **Output Passive Level**<br>This field controls the passive level of the output pin.<br>$0_B$ Passive Level is LOW<br>$1_B$ Passive Level is HIGH<br>A write always addresses the shadow register, while a read always returns the current used value. |
| **0** | [31:1] | r | **Reserved**<br>A read access always returns 0 |

**CC4yDIT**

This register holds the current dither compare and dither counter values.

**CC4yDIT (y = 0 - 3)**
**Dither Config** **($011C_H + 0100_H * y$)** **Reset Value: $00000000_H$**





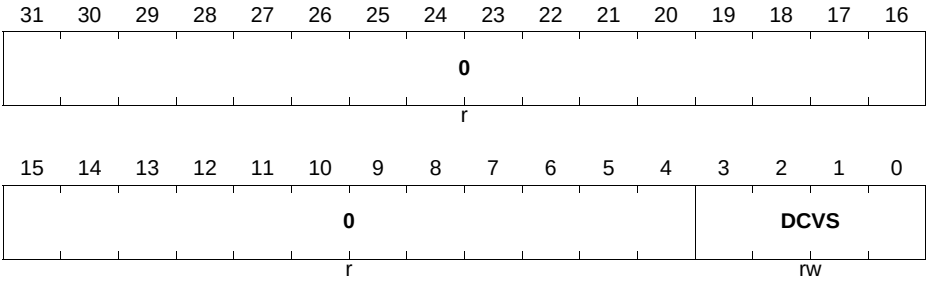
Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **DCV** | [3:0] | rh | **Dither compare Value**<br>This field contains the value used for the dither comparison.<br>This value is updated when a shadow transfer occurs with the **CC4yDITS**.DCVS. |
| **DCNT** | [11:8] | rh | **Dither counter actual value** |
| **0** | [7:4], [31:12] | r | **Reserved**<br>Read always returns 0. |

### CC4yDITS

This register contains the value that is going to be loaded into the **CC4yDIT**.DCV when the next shadow transfer occurs.

**CC4yDITS (y = 0 - 3)**
**Dither Shadow Register** **($0120_H + 0100_H * y$)** **Reset Value: $00000000_H$**

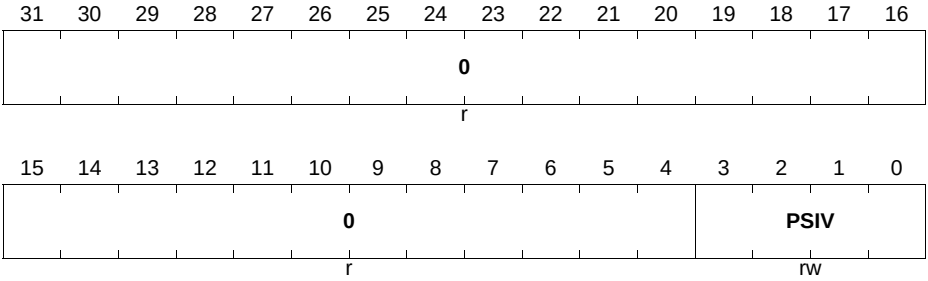



| Field | Bits | Type | Description |
|---|---|---|---|
| **DCVS** | [3:0] | rw | **Dither Shadow Compare Value**<br>This field contains the value that is going to be set on the dither compare value, **CC4yDIT**.DCV, within the next shadow transfer. |
| **0** | [31:4] | r | **Reserved**<br>Read always returns 0. |

### CC4yPSC

This register contains the value that is loaded into the prescaler during restart.

**CC4yPSC (y = 0 - 3)**
**Prescaler Control** **($0124_H + 0100_H * y$)** **Reset Value: $00000000_H$**

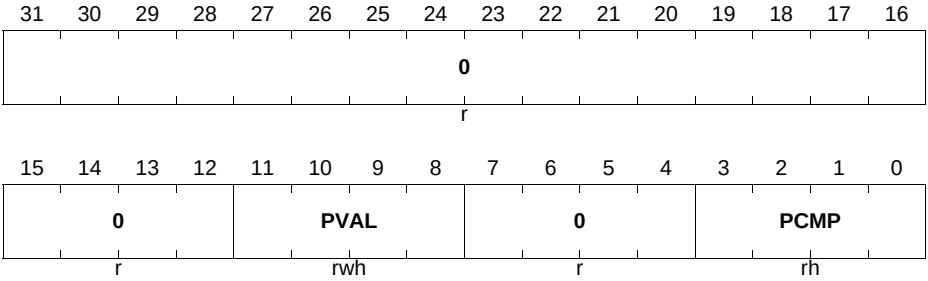



| Field | Bits | Type | Description |
|---|---|---|---|
| **PSIV** | [3:0] | rw | **Prescaler Initial Value**<br>This field contains the value that is applied to the Prescaler at startup.<br>When floating prescaler mode is used, this value is applied when a timer compare match AND prescaler compare match occurs or when a capture event is triggered. |
| **0** | [31:4] | r | **Reserved**<br>Read always returns 0. |

**CC4yFPC**

This register contains the value used for the floating prescaler compare and the actual prescaler division value.

**CC4yFPC (y = 0 - 3)**
**Floating Prescaler Control** **($0128_H$ + $0100_H$ * y)** **Reset Value: $00000000_H$**

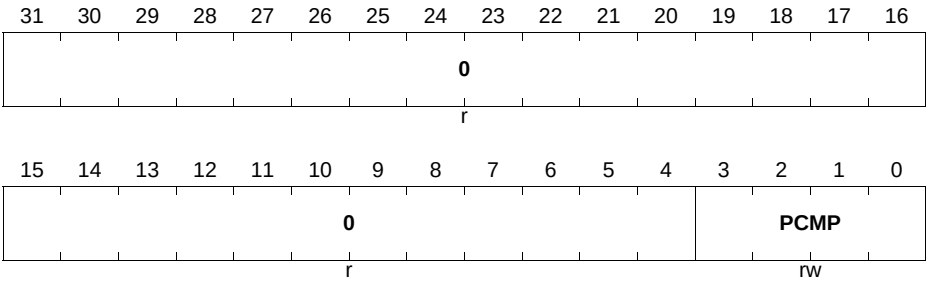



| Field | Bits | Type | Description |
|---|---|---|---|
| **PCMP** | [3:0] | rh | **Floating Prescaler Compare Value**<br>This field contains comparison value used in floating prescaler mode. The comparison is triggered by the Timer Compare match event. See **Section 19.2.11.2**. |
| **PVAL** | [11:8] | rwh | **Actual Prescaler Value**<br>See **Table 19-7**. Writing into this register is only possible when the prescaler is stopped. When the floating prescaler mode is not used, this value is equal to the **CC4yPSC**.PSIV. |
| **0** | [7:4], [15:12], [31:16] | r | **Reserved**<br>Read always returns 0. |

**CC4yFPCS**

This register contains the value that is going to be transferred to the **CC4yFPC**.PCMP field within the next shadow transfer update.

**CC4yFPCS (y = 0 - 3)**
**Floating Prescaler Shadow** **($012C_H + 0100_H * y$)** **Reset Value: $00000000_H$**

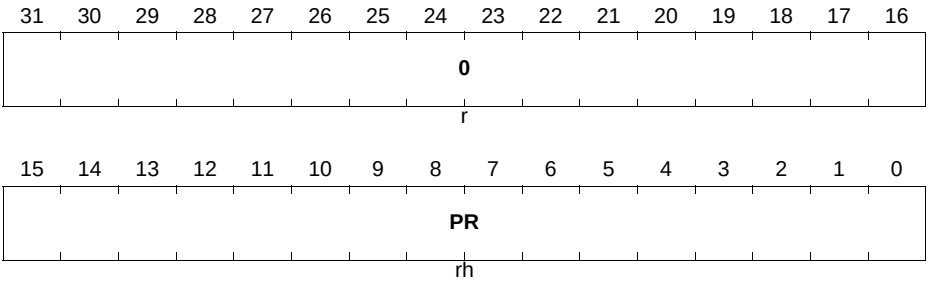



| Field | Bits | Type | Description |
|---|---|---|---|
| **PCMP** | [3:0] | rw | **Floating Prescaler Shadow Compare Value**<br>This field contains the value that is going to be set on the **CC4yFPC**.PCMP within the next shadow transfer. See **Table 19-7**. |
| **0** | [31:4] | r | **Reserved**<br>Read always returns 0. |

**CC4yPR**

This register contains the actual value for the timer period.

**CC4yPR (y = 0 - 3)**
**Timer Period Value** **($0130_H + 0100_H * y$)** **Reset Value: $00000000_H$**





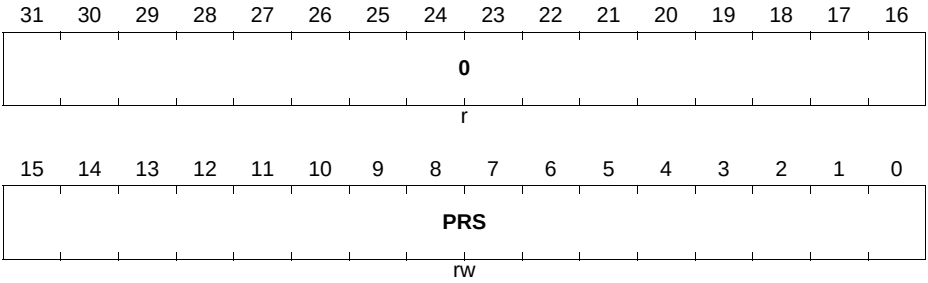
Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **PR** | [15:0] | rh | **Period Register**<br>Contains the value of the timer period.<br>*Note: In Capture Mode when a external signal is selected for capturing the timer value into the capture registers 2 and 3, PR is not accessible for writing. A read always returns 0.* |
| **0** | [31:16] | r | **Reserved**<br>A read always returns 0. |

### CC4yPRS

This register contains the value for the timer period that is going to be transferred into the **CC4yPR**.PR field when the next shadow transfer occurs.

**CC4yPRS (y = 0 - 3)**
**Timer Shadow Period Value** **($0134_H$ + $0100_H$ * y)** **Reset Value: $00000000_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **PRS** | [15:0] | rw | **Period Register**<br>Contains the value of the timer period, that is going to be passed into the **CC4yPR**.PR field when the next shadow transfer occurs.<br>*Note: In Capture Mode when a external signal is selected for capturing the timer value into the capture registers 2 and 3, the PRS is not accessible for writing. A read always returns 0.* |

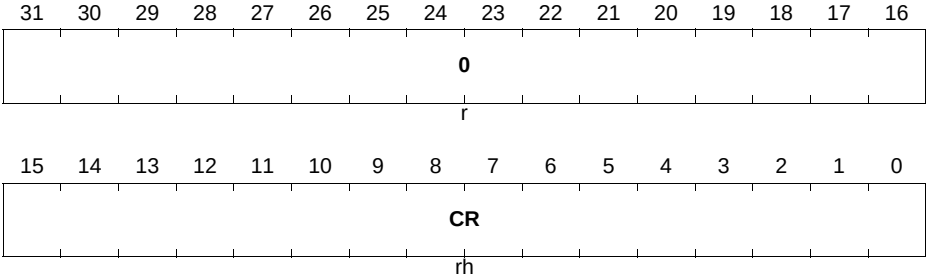

Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | [31:16] | r | **Reserved**<br>A read always returns 0. |

### CC4yCR

This register contains the value for the timer comparison.

**CC4yCR (y = 0 - 3)**
**Timer Compare Value** **($0138_H + 0100_H * y$)** **Reset Value: $00000000_H$**



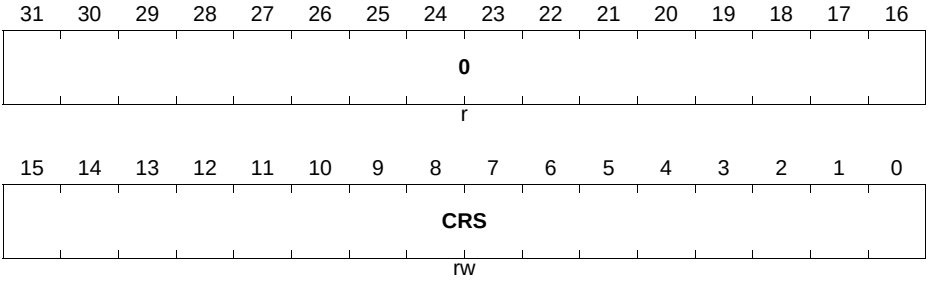

| Field | Bits | Type | Description |
|---|---|---|---|
| **CR** | [15:0] | rh | **Compare Register**<br>Contains the value for the timer comparison.<br>*Note: In Capture Mode when a external signal is selected for capturing the timer value into the capture registers 0 and 1, a read always returns 0.* |
| **0** | [31:16] | r | **Reserved**<br>A read always returns 0. |

### CC4yCRS

This register contains the value that is going to be loaded into the **CC4yCR**.CR field when the next shadow transfer occurs.


Subject to Agreement on the Use of Product Information

**CC4yCRS (y = 0 - 3)**
**Timer Shadow Compare Value ($013C_H$ + $0100_H$ * y)** **Reset Value: $00000000_H$**

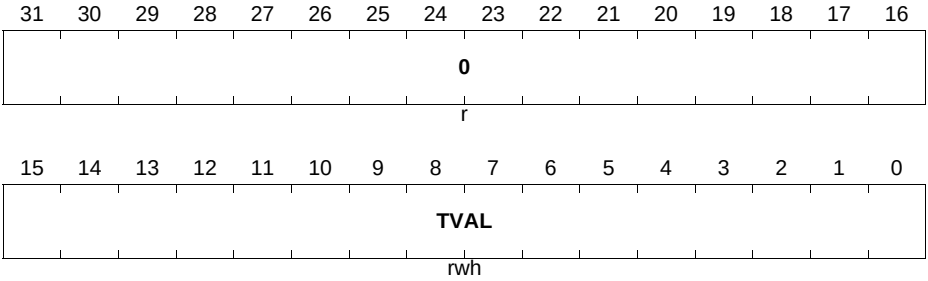



| Field | Bits | Type | Description |
|---|---|---|---|
| **CRS** | [15:0] | rw | **Compare Register**<br>Contains the value for the timer comparison, that is going to be passed into the **CC4yCR**.CR field when the next shadow transfer occurs.<br>*Note: In Capture Mode when a external signal is selected for capturing the timer value into the capture registers 0 and 1, a read always returns 0.* |
| **0** | [31:16] | r | **Reserved**<br>A read always returns 0. |

**CC4yTIMER**

This register contains the current value of the timer.

**CC4yTIMER (y = 0 - 3)**
**Timer Value** **($0170_H + 0100_H * y$)** **Reset Value: $00000000_H$**

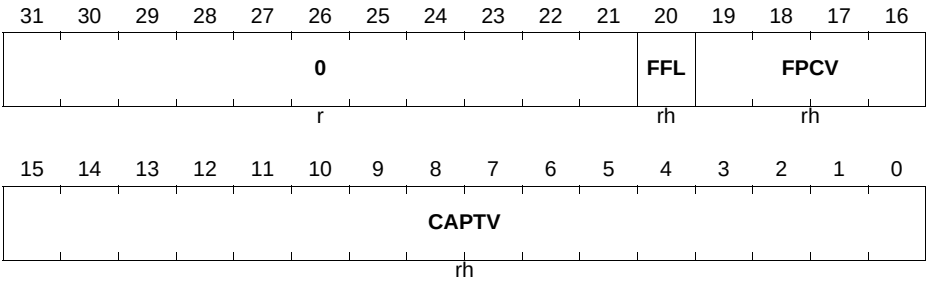



| Field | Bits | Type | Description |
|---|---|---|---|
| **TVAL** | [15:0] | rwh | **Timer Value**<br>This field contains the actual value of the timer.<br>A write access is only possible when the timer is stopped. |
| **0** | [31:16] | r | **Reserved**<br>A read access always returns 0 |

**CC4yC0V**

This register contains the values associated with the Capture 0 field.

**CC4yC0V (y = 0 - 3)**
**Capture Register 0** **($0174_H + 0100_H * y$)** **Reset Value: $00000000_H$**





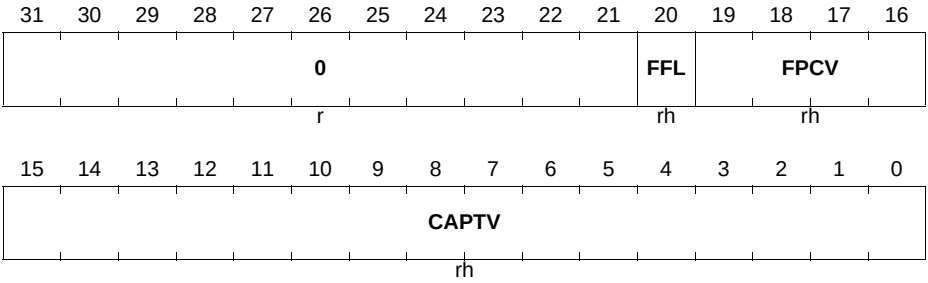
Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **CAPTV** | [15:0] | rh | **Capture Value**<br>This field contains the capture register 0 value. See Figure 19-26.<br>In compare mode a read access always returns 0. |
| **FPCV** | [19:16] | rh | **Prescaler Value**<br>This field contains the prescaler value at the time of the capture event into the capture register 0.<br>In compare mode a read access always returns 0. |
| **FFL** | 20 | rh | **Full Flag**<br>This bit indicates if a new value was capture into the capture register 0 after the last read access. See Figure 19-26. In compare mode a read access always returns 0.<br>$0_B$ No new value was captured into the specific capture register<br>$1_B$ A new value was captured into the specific register |
| **0** | [31:21] | r | **Reserved**<br>A read always returns 0 |

### CC4yC1V

This register contains the values associated with the Capture 1 field.

**CC4yC1V (y = 0 - 3)**
**Capture Register 1** **($0178_H + 0100_H * y$)** **Reset Value: $00000000_H$**





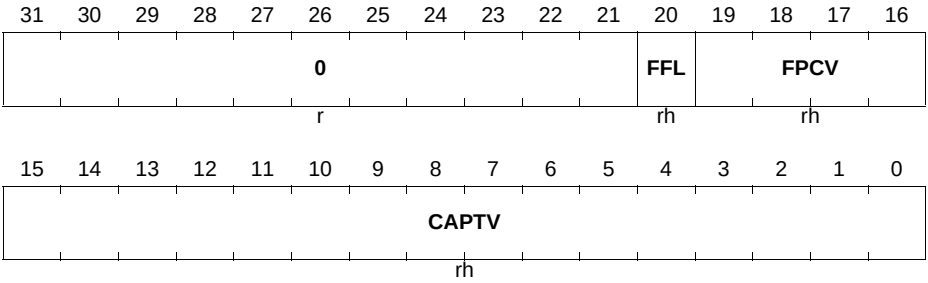
Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **CAPTV** | [15:0] | rh | **Capture Value**<br>This field contains the capture register 1 value. See **Figure 19-26**.<br>In compare mode a read access always returns 0. |
| **FPCV** | [19:16] | rh | **Prescaler Value**<br>This field contains the prescaler value at the time of the capture event into the capture register 1.<br>In compare mode a read access always returns 0. |
| **FFL** | 20 | rh | **Full Flag**<br>This bit indicates if a new value was capture into the capture register 1 after the last read access. See **Figure 19-26**. In compare mode a read access always returns 0.<br>$0_B$ No new value was captured into the specific capture register<br>$1_B$ A new value was captured into the specific register |
| **0** | [31:21] | r | **Reserved**<br>A read always returns 0 |

### CC4yC2V

This register contains the values associated with the Capture 2 field.

**CC4yC2V (y = 0 - 3)**
**Capture Register 2** **($017C_H + 0100_H * y$)** **Reset Value: $00000000_H$**





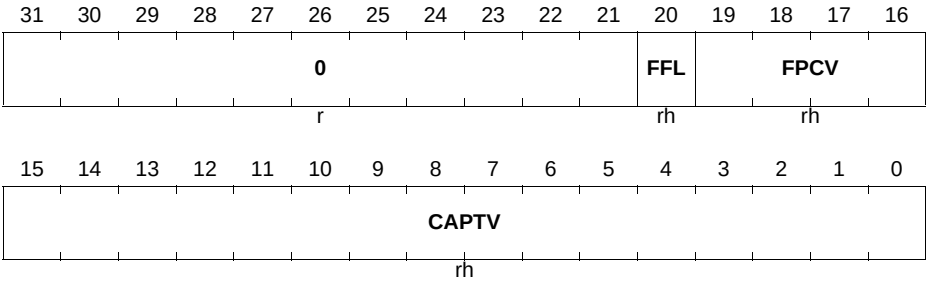
Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **CAPTV** | [15:0] | rh | **Capture Value**<br>This field contains the capture register 2 value. See **Figure 19-26**.<br>In compare mode a read access always returns 0. |
| **FPCV** | [19:16] | rh | **Prescaler Value**<br>This field contains the prescaler value at the time of the capture event into the capture register 2. In compare mode a read access always returns 0. |
| **FFL** | 20 | rh | **Full Flag**<br>This bit indicates if a new value was capture into the capture register 2 after the last read access. See **Figure 19-26**. In compare mode a read access always returns 0.<br>$0_B$ No new value was captured into the specific capture register<br>$1_B$ A new value was captured into the specific register |
| **0** | [31:21] | r | **Reserved**<br>A read always returns 0 |

### CC4yC3V

This register contains the values associated with the Capture 3 field.

**CC4yC3V (y = 0 - 3)**
**Capture Register 3** **($0180_H + 0100_H * y$)** **Reset Value: $00000000_H$**





Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **CAPTV** | [15:0] | rh | **Capture Value**<br>This field contains the capture register 3 value. See **Figure 19-26**. In compare mode a read access always returns 0. |
| **FPCV** | [19:16] | rh | **Prescaler Value**<br>This field contains the prescaler value at the time of the capture event into the capture register 3.<br>In compare mode a read access always returns 0. |
| **FFL** | 20 | rh | **Full Flag**<br>This bit indicates if a new value was capture into the capture register 3 after the last read access. See **Figure 19-26**. In compare mode a read access always returns 0.<br>$0_B$ No new value was captured into the specific capture register<br>$1_B$ A new value was captured into the specific register |
| **0** | [31:21] | r | **Reserved**<br>A read always returns 0 |

### CC4yINTS

This register contains the status of all interrupt sources.

**CC4yINTS (y = 0 - 3)**
**Interrupt Status** **($01A0_H + 0100_H * y$)** **Reset Value: $00000000_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | TRPF | E2AS | E1AS | E0AS | 0 | | | | CMDS | CMUS | OMDS | PMUS |
| r | | | | rh | rh | rh | rh | r | | | | rh | rh | rh | rh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **PMUS** | 0 | rh | **Period Match while Counting Up**<br>$0_B$ Period match while counting up not detected<br>$1_B$ Period match while counting up detected |
| **OMDS** | 1 | rh | **One Match while Counting Down**<br>$0_B$ One match while counting down not detected<br>$1_B$ One match while counting down detected |
| **CMUS** | 2 | rh | **Compare Match while Counting Up**<br>$0_B$ Compare match while counting up not detected<br>$1_B$ Compare match while counting up detected |
| **CMDS** | 3 | rh | **Compare Match while Counting Down**<br>$0_B$ Compare match while counting down not detected<br>$1_B$ Compare match while counting down detected |
| **E0AS** | 8 | rh | **Event 0 Detection Status**<br>Depending on the user selection on the **CC4yINS**.EV0EM, this bit can be set when a rising, falling or both transitions are detected.<br>$0_B$ Event 0 not detected<br>$1_B$ Event 0 detected |
| **E1AS** | 9 | rh | **Event 1 Detection Status**<br>Depending on the user selection on the **CC4yINS**.EV1EM, this bit can be set when a rising, falling or both transitions are detected.<br>$0_B$ Event 1 not detected<br>$1_B$ Event 1 detected |
| **E2AS** | 10 | rh | **Event 2 Detection Status**<br>Depending on the user selection on the **CC4yINS**.EV1EM, this bit can be set when a rising, falling or both transitions are detected.<br>$0_B$ Event 2 not detected<br>$1_B$ Event 2 detected<br>*Note: If this event is linked with the TRAP function, this field is automatically cleared when the slice exits the Trap State.* |
| **TRPF** | 11 | rh | **Trap Flag Status**<br>This field contains the status of the Trap Flag. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | [7:4], [31:12] | r | **Reserved**<br>A read always returns 0. |

### CC4yINTE

Through this register it is possible to enable or disable the specific interrupt source(s).

**CC4yINTE (y = 0 - 3)**
**Interrupt Enable Control** **($01A4_H + 0100_H * y$)** **Reset Value: $00000000_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | E2AE | E1AE | E0AE | 0 | | | | CMDE | CMUE | OME | PME |
| r | | | | | rw | rw | rw | r | | | | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **PME** | 0 | rw | **Period match while counting up enable**<br>Setting this bit to $1_B$ enables the generation of an interrupt pulse every time a period match while counting up occurs.<br>$0_B$ Period Match interrupt is disabled<br>$1_B$ Period Match interrupt is enabled |
| **OME** | 1 | rw | **One match while counting down enable**<br>Setting this bit to $1_B$ enables the generation of an interrupt pulse every time an one match while counting down occurs.<br>$0_B$ One Match interrupt is disabled<br>$1_B$ One Match interrupt is enabled |


Subject to Agreement on the Use of Product Information

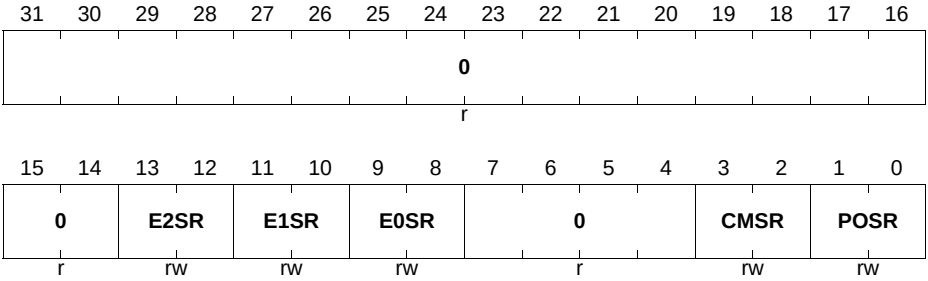
| Field | Bits | Type | Description |
|---|---|---|---|
| **CMUE** | 2 | rw | **Compare match while counting up enable**<br>Setting this bit to $1_B$ enables the generation of an interrupt pulse every time a compare match while counting up occurs.<br>$0_B$ Compare Match while counting up interrupt is disabled<br>$1_B$ Compare Match while counting up interrupt is enabled |
| **CMDE** | 3 | rw | **Compare match while counting down enable**<br>Setting this bit to $1_B$ enables the generation of an interrupt pulse every time a compare match while counting down occurs.<br>$0_B$ Compare Match while counting down interrupt is disabled<br>$1_B$ Compare Match while counting down interrupt is enabled |
| **E0AE** | 8 | rw | **Event 0 interrupt enable**<br>Setting this bit to $1_B$ enables the generation of an interrupt pulse every time that Event 0 is detected.<br>$0_B$ Event 0 detection interrupt is disabled<br>$1_B$ Event 0 detection interrupt is enabled |
| **E1AE** | 9 | rw | **Event 1 interrupt enable**<br>Setting this bit to $1_B$ enables the generation of an interrupt pulse every time that Event 1 is detected.<br>$0_B$ Event 1 detection interrupt is disabled<br>$1_B$ Event 1 detection interrupt is enabled |
| **E2AE** | 10 | rw | **Event 2 interrupt enable**<br>Setting this bit to $1_B$ enables the generation of an interrupt pulse every time that Event 2 is detected.<br>$0_B$ Event 2 detection interrupt is disabled<br>$1_B$ Event 2 detection interrupt is enabled |
| **0** | [7:4], [31:11] | r | **Reserved**<br>A read always returns 0 |

### CC4ySRS

Through this register it is possible to select to which service request line each interrupt source is forwarded.


Subject to Agreement on the Use of Product Information

**CC4ySRS (y = 0 - 3)**
**Service Request Selector** **($01A8_H + 0100_H * y$)** **Reset Value: $00000000_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **POSR** | [1:0] | rw | **Period/One match Service request selector**<br>This field selects to which slice Service request line, the interrupt(s) generated by the Period match while counting up and One match while counting down are going to be forward.<br>$00_B$ Forward to CC4ySR0<br>$01_B$ Forward to CC4ySR1<br>$10_B$ Forward to CC4ySR2<br>$11_B$ Forward to CC4ySR3 |
| **CMSR** | [3:2] | rw | **Compare match Service request selector**<br>This field selects to which slice Service request line, the interrupt(s) generated by the Compare match while counting up and Compare match while counting down are going to be forward.<br>$00_B$ Forward to CC4ySR0<br>$01_B$ Forward to CC4ySR1<br>$10_B$ Forward to CC4ySR2<br>$11_B$ Forward to CC4ySR3 |
| **E0SR** | [9:8] | rw | **Event 0 Service request selector**<br>This field selects to which slice Service request line, the interrupt generated by the Event 0 detection is going to be forward.<br>$00_B$ Forward to CC4ySR0<br>$01_B$ Forward to CC4ySR1<br>$10_B$ Forward to CC4ySR2<br>$11_B$ Forward to CC4ySR3 |

| Field | Bits | Type | Description |
|---|---|---|---|
| **E1SR** | [11:10] | rw | **Event 1 Service request selector**<br>This field selects to which slice Service request line, the interrupt generated by the Event 1detection is going to be forward.<br>$00_B$ Forward to CC4ySR0<br>$01_B$ Forward to CC4ySR1<br>$10_B$ Forward to CC4ySR2<br>$11_B$ Forward to CC4ySR3 |
| **E2SR** | [13:12] | rw | **Event 2 Service request selector**<br>This field selects to which slice Service request line, the interrupt generated by the Event 2 detection is going to be forward.<br>$00_B$ Forward to CC4ySR0<br>$01_B$ Forward to CC4ySR1<br>$10_B$ Forward to CC4ySR2<br>$11_B$ Forward to CC4ySR3 |
| **0** | [7:4], [31:14] | r | **Reserved**<br>Read always returns 0. |

### CC4ySWS

Through this register it is possible for the SW to set a specific interrupt status flag.

**CC4ySWS (y = 0 - 3)**
**Interrupt Status Set** **($01AC_H + 0100_H * y$)** **Reset Value: $00000000_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | STRPF | SE2A | SE1A | SE0A | 0 | | | | SCMD | SCMU | SOM | SPM |
| r | | | | w | w | w | w | r | | | | w | w | w | w |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **SPM** | 0 | w | **Period match while counting up set**<br>Writing a $1_B$ into this field sets the **CC4yINTS**.PMUS bit. An interrupt pulse is generated if the source is enabled. A read always returns 0. |
| **SOM** | 1 | w | **One match while counting down set**<br>Writing a $1_B$ into this bit sets the **CC4yINTS**.OMDS bit. An interrupt pulse is generated if the source is enabled. A read always returns 0. |
| **SCMU** | 2 | w | **Compare match while counting up set**<br>Writing a $1_B$ into this field sets the **CC4yINTS**.CMUS bit. An interrupt pulse is generated if the source is enabled. A read always returns 0. |
| **SCMD** | 3 | w | **Compare match while counting down set**<br>Writing a $1_B$ into this bit sets the **CC4yINTS**.CMDS bit. An interrupt pulse is generated if the source is enabled. A read always returns 0. |
| **SE0A** | 8 | w | **Event 0 detection set**<br>Writing a $1_B$ into this bit sets the **CC4yINTS**.E0AS bit. An interrupt pulse is generated if the source is enabled. A read always returns 0. |
| **SE1A** | 9 | w | **Event 1 detection set**<br>Writing a $1_B$ into this bit sets the **CC4yINTS**.E1AS bit. An interrupt pulse is generated if the source is enabled. A read always returns 0. |
| **SE2A** | 10 | w | **Event 2 detection set**<br>Writing a $1_B$ into this bit sets the **CC4yINTS**.E2AS bit. An interrupt pulse is generated if the source is enabled. A read always returns 0. |
| **STRPF** | 11 | w | **Trap Flag status set**<br>Writing a $1_B$ into this bit sets the **CC4yINTS**.TRPF bit. A read always returns 0. |
| **0** | [7:4], [31:12] | r | **Reserved**<br>Read always returns 0 |

### CC4ySWR

Through this register it is possible for the SW to clear a specific interrupt status flag.


Subject to Agreement on the Use of Product Information

**CC4ySWR (y = 0 - 3)**
**Interrupt Status Clear** **($01B0_H + 0100_H * y$)** **Reset Value: $00000000_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | RTRPF | RE2A | RE1A | RE0A | 0 | | | | RCMD | RCMU | ROM | RPM |
| r | | | | w | w | w | w | r | | | | w | w | w | w |

| Field | Bits | Type | Description |
|---|---|---|---|
| **RPM** | 0 | w | **Period match while counting up clear**<br>Writing a $1_B$ into this field clears the **CC4yINTS**.PMUS bit. A read always returns 0. |
| **ROM** | 1 | w | **One match while counting down clear**<br>Writing a $1_B$ into this bit clears the **CC4yINTS**.OMDS bit. A read always returns 0. |
| **RCMU** | 2 | w | **Compare match while counting up clear**<br>Writing a $1_B$ into this field clears the **CC4yINTS**.CMUS bit. A read always returns 0. |
| **RCMD** | 3 | w | **Compare match while counting down clear**<br>Writing a $1_B$ into this bit clears the **CC4yINTS**.CMDS bit. A read always returns 0. |
| **RE0A** | 8 | w | **Event 0 detection clear**<br>Writing a $1_B$ into this bit clears the **CC4yINTS**.E0AS bit. A read always returns 0. |
| **RE1A** | 9 | w | **Event 1 detection clear**<br>Writing a $1_B$ into this bit clears the **CC4yINTS**.E1AS bit. A read always returns 0. |
| **RE2A** | 10 | w | **Event 2 detection clear**<br>Writing a $1_B$ into this bit clears the **CC4yINTS**.E2AS bit. A read always returns 0. |

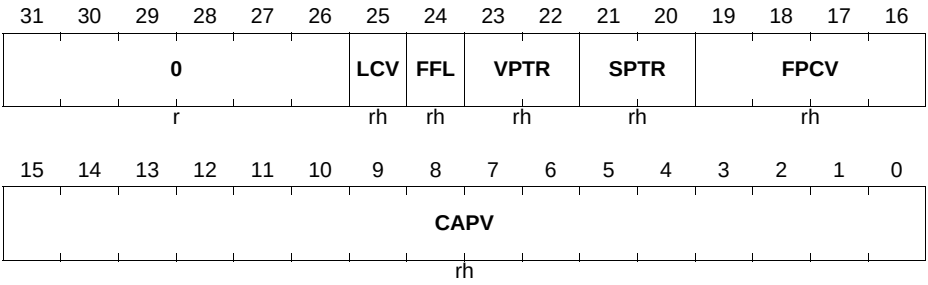

Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **RTRPF** | 11 | w | **Trap Flag status clear**<br>Writing a $1_B$ into this bit clears the **CC4yINTS**.TRPF bit. Not valid if **CC4yTC**.TRPEN = $1_B$ and the Trap State is still active.<br>A read always returns 0. |
| **0** | [7:4], [31:12] | r | **Reserved**<br>Read always returns 0 |

### CC4yECRD0

Through this register it is possible to read back the FIFO structure of the capture function that is linked with the capture trigger 0. The read back is only valid if the **CC4yTC**.ECM = $1_B$.

**CC4yECRD0 (y = 0 - 3)**
**Extended Read Back 0** **($01B8_H + 0100_H * y$)** **Reset Value: $00000000_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **CAPV** | [15:0] | rh | **Timer Capture Value**<br>This field contains the timer captured value |
| **FPCV** | [19:16] | rh | **Prescaler Capture value**<br>This field contains the value of the prescaler clock division associated with the specific CAPV field |


Subject to Agreement on the Use of Product Information

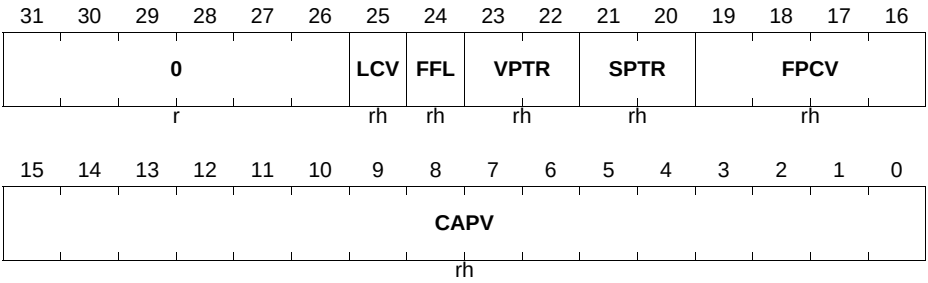
| Field | Bits | Type | Description |
|---|---|---|---|
| **SPTR** | [21:20] | rh | **Slice pointer**<br>This field indicates the slice index in which the value was captured.<br>$00_B$ CC40<br>$01_B$ CC41<br>$10_B$ CC42<br>$11_B$ CC43 |
| **VPTR** | [23:22] | rh | **Capture register pointer**<br>This field indicates the capture register index in which the value was captured.<br>$00_B$ Capture register 0<br>$01_B$ Capture register 1<br>$10_B$ Capture register 2<br>$11_B$ Capture register 3 |
| **FFL** | 24 | rh | **Full Flag**<br>This bit indicates if the associated capture register contains a new value.<br>$0_B$ No new value was captured into this register<br>$1_B$ A new value has been captured into this register |
| **LCV** | 25 | rh | **Lost Capture Value**<br>This field indicates if between two reads of the ECRD0 a capture trigger occured while the FIFO structure was full. If a capture trigger occured between two reads than a capture value was lost. This field is automatically cleared by the HW whenever a read to the ECRD occurs.<br>$0_B$ No capture was lost<br>$1_B$ A capture was lost |
| **0** | [31:26] | r | **Reserved**<br>Read always returns 0 |

### CC4yECRD1

Through this register it is possible to read back the FIFO structure of the capture function that is linked with the capture trigger 1. The read back is only valid if the **CC4yTC**.ECM = $1_B$.


Subject to Agreement on the Use of Product Information

**CC4yECRD1 (y = 0 - 3)**
**Extended Read Back 1** **($01BC_H + 0100_H * y$)** **Reset Value: $00000000_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **CAPV** | [15:0] | rh | **Timer Capture Value**<br>This field contains the timer captured value |
| **FPCV** | [19:16] | rh | **Prescaler Capture value**<br>This field contains the value of the prescaler clock division associated with the specific CAPV field |
| **SPTR** | [21:20] | rh | **Slice pointer**<br>This field indicates the slice index in which the value was captured.<br>$00_B$ CC40<br>$01_B$ CC41<br>$10_B$ CC42<br>$11_B$ CC43 |
| **VPTR** | [23:22] | rh | **Capture register pointer**<br>This field indicates the capture register index in which the value was captured.<br>$00_B$ Capture register 0<br>$01_B$ Capture register 1<br>$10_B$ Capture register 2<br>$11_B$ Capture register 3 |
| **FFL** | 24 | rh | **Full Flag**<br>This bit indicates if the associated capture register contains a new value.<br>$0_B$ No new value was captured into this register<br>$1_B$ A new value has been captured into this register |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **LCV** | 25 | rh | **Lost Capture Value**<br>This field indicates if between two reads of the ECRD0 a capture trigger occured while the FIFO structure was full. If a capture trigger occured between two reads than a capture value was lost. This field is automatically cleared by the HW whenever a read to the ECRD occurs.<br>$0_B$ No capture was lost<br>$1_B$ A capture was lost |
| **0** | [31:26] | r | **Reserved**<br>Read always returns 0 |

## 19.8 Interconnects

The tables that refer to the “global pins” are the ones that contain the inputs/outputs of each module that are common to all slices.

The GPIO connections are available at the Ports chapter.

### 19.8.1 CCU40 pins

**Table 19-13 CCU40 Pin Connections**

| Global Inputs/Outputs | I/O | Connected To | Description |
|---|---|---|---|
| CCU40.MCLK | I | PCLK | Kernel clock |
| CCU40.CLKA | I | not connected | another count source for the prescaler |
| CCU40.CLKB | I | ERU0.IOUT0 | another count source for the prescaler |
| CCU40.CLKC | I | ERU0.IOUT1 | another count source for the prescaler |
| CCU40.MCSS | I | POSIF0.OUT6 | Multi pattern sync with shadow transfer trigger |
| CCU40.SR0 | O | NVIC;<br>CCU80.IGBTD;<br>POSIF0.MSETA;<br>ERU0.OGU01; | Service request line |


Subject to Agreement on the Use of Product Information

**Table 19-13 CCU40 Pin Connections** (cont'd)

| Global Inputs/Outputs | I/O | Connected To | Description |
|---|---|---|---|
| CCU40.SR1 | O | NVIC;<br>ERU0.OGU11; | Service request line |
| CCU40.SR2 | O | NVIC;<br>VADC0.BGREQTRA;<br>VADC0.G0REQTRA;<br>VADC0.G1REQTRA;<br>CCU80.IN0K;<br>CCU80.IN1K;<br>ERU0.OGU21; | Service request line |
| CCU40.SR3 | O | NVIC;<br>VADC0.BGREQTRB;<br>VADC0.G0REQTRB;<br>VADC0.G1REQTRB;<br>CCU80.IGBTB;<br>CCU80.IN2K;<br>CCU80.IN3K;<br>ERU0.OGU31; | Service request line |

**Table 19-14 CCU40 - CC40 Pin Connections**

| Input/Output | I/O | Connected To | Description |
|---|---|---|---|
| CCU40.IN0A | I | P0.12 | General purpose function |
| CCU40.IN0B | I | P0.6 | General purpose function |
| CCU40.IN0C | I | P0.0 | General purpose function |
| CCU40.IN0D | I | ERU0.PDOUT1 | General purpose function |
| CCU40.IN0E | I | POSIF0.OUT0 | General purpose function |
| CCU40.IN0F | I | POSIF0.OUT1 | General purpose function |
| CCU40.IN0G | I | POSIF0.OUT3 | General purpose function |
| CCU40.IN0H | I | CCU80.ST3 | General purpose function |
| CCU40.IN0I | I | SCU.GSC40 | General purpose function |
| CCU40.IN0J | I | ERU0.PDOUT0 | General purpose function |
| CCU40.IN0K | I | ERU0.IOUT0 | General purpose function |
| CCU40.IN0L | I | USIC0_CH0.DX2INS | General purpose function |
| CCU40.IN0M | I | CCU40.GP10 | General purpose function |


Subject to Agreement on the Use of Product Information

**Table 19-14 CCU40 - CC40 Pin Connections** (cont'd)

| Input/Output | I/O | Connected To | Description |
|---|---|---|---|
| CCU40.IN0N | I | CCU40.ST1 | General purpose function |
| CCU40.IN0O | I | CCU40.ST2 | General purpose function |
| CCU40.IN0P | I | CCU40.ST3 | General purpose function |
| CCU40.MCI0 | I | BCCU0.OUT2 | Multi Channel pattern input |
| CCU40.OUT0 | O | P0.0;<br>P0.5;<br>P0.6;<br>P1.0;<br>P2.0;<br>P2.0.HW1 direction control | Slice compare output |
| CCU40.GP00 | O | CCU40.IN3M | Selected signal for event 0 |
| CCU40.GP01 | O | not connected | Selected signal for event 1 |
| CCU40.GP02 | O | CCU80.IN0C | Selected signal for event 2 |
| CCU40.ST0 | O | CCU40.IN1N;<br>CCU40.IN2N;<br>CCU40.IN3N;<br>CCU80.IGBTC;<br>POSIF0.HSDA;<br>VADC0.BGREQGTD;<br>VADC0.G0REQGTD;<br>VADC0.G1REQGTD; | 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 19-15 CCU40 - CC41 Pin Connections**

| Input/Output | I/O | Connected To | Description |
|---|---|---|---|
| CCU40.IN1A | I | P0.12 | General purpose function |
| CCU40.IN1B | I | P0.7 | General purpose function |
| CCU40.IN1C | I | P0.1 | General purpose function |
| CCU40.IN1D | I | ERU0.PDOUT0 | General purpose function |
| CCU40.IN1E | I | POSIF0.OUT0 | General purpose function |
| CCU40.IN1F | I | POSIF0.OUT1 | General purpose function |


Subject to Agreement on the Use of Product Information

**Table 19-15 CCU40 - CC41 Pin Connections** (cont'd)

| Input/Output | I/O | Connected To | Description |
|---|---|---|---|
| CCU40.IN1G | I | POSIF0.OUT3 | General purpose function |
| CCU40.IN1H | I | POSIF0.OUT4 | General purpose function |
| CCU40.IN1I | I | SCU.GSC40 | General purpose function |
| CCU40.IN1J | I | ERU0.PDOUT1 | General purpose function |
| CCU40.IN1K | I | ERU0.IOUT1 | General purpose function |
| CCU40.IN1L | I | USIC0_CH1.DX2INS | General purpose function |
| CCU40.IN1M | I | CCU40.GP20 | General purpose function |
| CCU40.IN1N | I | CCU40.ST0 | General purpose function |
| CCU40.IN1O | I | CCU40.ST2 | General purpose function |
| CCU40.IN1P | I | CCU40.ST3 | General purpose function |
| CCU40.MCI1 | I | BCCU0.OUT3 | Multi Channel pattern input |
| CCU40.OUT1 | O | P0.1:<br>P0.4;<br>P0.7;<br>P1.1;<br>P2.1;<br>P2.1.HW1 direction control | Slice compare output |
| CCU40.GP10 | O | CCU40.IN0M | Selected signal for event 0 |
| CCU40.GP11 | O | not connected | Selected signal for event 1 |
| CCU40.GP12 | O | CCU80.IN1C | Selected signal for event 2 |
| CCU40.ST1 | O | CCU40.IN0N;<br>CCU40.IN2O;<br>CCU40.IN3O;<br>POSIF0.MSETB;<br>VADC0.BGREQGTC;<br>VADC0.G0REQGTC;<br>VADC0.G1REQGTC; | 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) |


Subject to Agreement on the Use of Product Information

**Table 19-16 CCU40 - CC42 Pin Connections**

| Input/Output | I/O | Connected To | Description |
|---|---|---|---|
| CCU40.IN2A | I | P0.12 | General purpose function |
| CCU40.IN2B | I | P0.8 | General purpose function |
| CCU40.IN2C | I | P0.2 | General purpose function |
| CCU40.IN2D | I | ERU0.PDOUT3 | General purpose function |
| CCU40.IN2E | I | POSIF0.OUT1 | General purpose function |
| CCU40.IN2F | I | POSIF0.OUT2 | General purpose function |
| CCU40.IN2G | I | POSIF0.OUT3 | General purpose function |
| CCU40.IN2H | I | POSIF0.OUT4 | General purpose function |
| CCU40.IN2I | I | SCU.GSC40 | General purpose function |
| CCU40.IN2J | I | ERU0.PDOUT2 | General purpose function |
| CCU40.IN2K | I | ERU0.IOUT2 | General purpose function |
| CCU40.IN2L | I | reserved | General purpose function |
| CCU40.IN2M | I | CCU40.GP30 | General purpose function |
| CCU40.IN2N | I | CCU40.ST0 | General purpose function |
| CCU40.IN2O | I | CCU40.ST1 | General purpose function |
| CCU40.IN2P | I | CCU40.ST3 | General purpose function |
| CCU40.MCI2 | I | BCCU0.OUT4 | Multi Channel pattern input |
| CCU40.OUT2 | O | P0.2;<br>P0.8;<br>P1.2;<br>P2.10;<br>P2.8.HW1 pull control<br>P2.9.HW1 pull control<br>P2.10.HW1 direction control | Slice compare output |
| CCU40.GP20 | O | CCU40.IN1M | Selected signal for event 0 |
| CCU40.GP21 | O | not connected | Selected signal for event 1 |
| CCU40.GP22 | O | CCU80.IN2C | Selected signal for event 2 |


Subject to Agreement on the Use of Product Information

**Table 19-16 CCU40 - CC42 Pin Connections** (cont'd)

| Input/Output | I/O | Connected To | Description |
|---|---|---|---|
| CCU40.ST2 | O | CCU40.IN0O;<br>CCU40.IN1O;<br>CCU40.IN3P;<br>VADC0.BGREQGTB;<br>VADC0.G0REQGTB;<br>VADC0.G1REQGTB;<br>POSIF0.MSETC; | 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 19-17 CCU40 - CC43 Pin Connections**

| Input/Output | I/O | Connected To | Description |
|---|---|---|---|
| CCU40.IN3A | I | P0.12 | General purpose function |
| CCU40.IN3B | I | P0.9 | General purpose function |
| CCU40.IN3C | I | P0.3 | General purpose function |
| CCU40.IN3D | I | ERU0.PDOUT2 | General purpose function |
| CCU40.IN3E | I | POSIF0.OUT3 | General purpose function |
| CCU40.IN3F | I | POSIF0.OUT5 | General purpose function |
| CCU40.IN3G | I | VADC0.G0ARBCNT | General purpose function |
| CCU40.IN3H | I | CCU80.IGBTO | General purpose function |
| CCU40.IN3I | I | SCU.GSC40 | General purpose function |
| CCU40.IN3J | I | ERU0.PDOUT3 | General purpose function |
| CCU40.IN3K | I | ERU0.IOUT3 | General purpose function |
| CCU40.IN3L | I | reserved | General purpose function |
| CCU40.IN3M | I | CCU40.GP00 | General purpose function |
| CCU40.IN3N | I | CCU40.ST0 | General purpose function |
| CCU40.IN3O | I | CCU40.ST1 | General purpose function |
| CCU40.IN3P | I | CCU40.ST2 | General purpose function |
| CCU40.MCI3 | I | BCCU0.OUT5 | Multi Channel pattern input |


Subject to Agreement on the Use of Product Information

**Table 19-17 CCU40 - CC43 Pin Connections** (cont'd)

| Input/Output | I/O | Connected To | Description |
|---|---|---|---|
| CCU40.OUT3 | O | P0.3;<br>P0.9;<br>P1.3;<br>P2.11;<br>P2.2.HW1 pull control<br>P2.6.HW1 pull control<br>P2.7.HW1 pull control<br>P2.11.HW1 direction control | Slice compare output |
| CCU40.GP30 | O | CCU40.IN2M | Selected signal for event 0 |
| CCU40.GP31 | O | not connected | Selected signal for event 1 |
| CCU40.GP32 | O | CCU80.IN3C | Selected signal for event 2 |
| CCU40.ST3 | O | CCU40.IN0P;<br>CCU40.IN1P;<br>CCU40.IN2P;<br>VADC0.BGREQGTA;<br>VADC0.G0REQGTA;<br>VADC0.G1REQGTA;<br>CCU80.IGBTA;<br>POSIF0.MSETD; | 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) |


Subject to Agreement on the Use of Product Information

# 20 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 20-1 Abbreviations table**

| | |
|---|---|
| PWM | Pulse Width Modulation |
| CCU8x | Capture/Compare Unit 8 module instance x |
| CC8y | Capture/Compare Unit 8 Timer Slice instance y |
| ADC | Analog to Digital Converter |
| POSIF | Position Interface peripheral |
| SCU | System Control Unit |
| $f_{ccu8}$ | CCU8 module clock frequency |
| $f_{tclk}$ | CC8y timer clock frequency |

*Note: A small “y” or “x” letter in a register indicates an index*

## 20.1 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.


Subject to Agreement on the Use of Product Information

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.

### 20.1.1 Features

**CCU8 Module Features**

Each CCU8 represents a combination of four Timer Slices, that can work independently in compare or capture mode. Each timer slice has 4 dedicated outputs for PWM signal generation.

All four CCU8 timer slices, CC8y, are identical in terms of available functions and operating modes. Avoiding this way the need of implementing different software routines, depending on which resource of CCU8 is used.

A built-in link between the four timer slices is also available, enabling this way a simplified timer concatenation and sequential operations.

General Features

- 16 bit timer cells
- programmable low pass filter for the inputs
- built-in timer concatenation
  - 32, 48 or 64 bit width
- shadow transfer for the period and compare channels
- four capture registers in capture mode
- programmable clock prescaler
- normal timer mode
- gated timer mode
- three counting schemes
  - center aligned
  - edge aligned
  - single shot
- PWM generation
- asymmetric PWM generation
- TRAP function
- dead time generation
- start/stop can be controlled by external events
- counting external events
- four dedicated service request lines per CCU8

Additional features

- external modulation function
- load controlled by external events


Subject to Agreement on the Use of Product Information

- 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 20-2** a summary of the major features of the CCU8 unit mapped with the most common applications.

**Table 20-2 Applications summary**

| Feature | Applications |
|---|---|
| Four independent timer cells | Independent PWM generation:<br>• Multiple buck/boost converter control (with independent frequencies)<br>• Different mode of operation for each timer, increasing the resources optimization<br>• Up to 2 H-Bridge control<br>• multiple Zero Voltage Switch (ZVS) converter control with easy link to the ADC channels.<br>• Multi Level Inverters |
| Two compare channels per Timer Slice | Linking between the two compare channels or linking between two Timer Slices:<br>• Asymmetric PWM signal generation possibility decreases the number of current sensors<br>• Linking between timer slices enable Phase Shift Full Bridge topologies control<br>• Linking between slices enable N-Phase DC/DC converter control |


Subject to Agreement on the Use of Product Information

**Table 20-2 Applications summary** (cont'd)

| Feature | Applications |
|---|---|
| Two Dead Time Generators | Independent dead time values for rising and falling transitions and independent channel dead time counter:<br>• 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<br>• 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:<br>• High dynamic trigger capturing<br>• High dynamic signal measurement |
| Dithering PWM | Generating a fractional PWM frequency or duty cycle:<br>• To avoid big steps on frequency or duty cycle adjustment in slow control loop applications<br>• Increase the PWM signal resolution over time |
| Floating prescaler | Automated control signal measurement:<br>• decrease SW activity for monitoring signals with high or unknown dynamics<br>• generating a more than 16 bit timer for system control |
| Up to 9 functions via external signals for each timer | Flexible resource optimization:<br>• The complete set of external functions is always available<br>• Several arrangements can be done inside a CCU8, e.g., one timer working in capture mode and one working in compare |
| Output Parity Checker | Automated Mosfet signal monitoring:<br>• parity checker can be used to monitor the output of the IGBTs and comparing them against the complete set of PWM outputs of CCU8.<br>• Avoiding short circuits in a multi Mosfet system. |


Subject to Agreement on the Use of Product Information

**Table 20-2 Applications summary** (cont'd)

| Feature | Applications |
|---|---|
| 4 dedicated service request lines | Specially developed for:<br>• generating interrupts for the microprocessor<br>• flexible connectivity between peripherals, e.g., ADC triggering. |
| Linking with POSIF | Flexible profiles for:<br>• Rotary Encoder connection<br>• Hall Sensor<br>• Modulating the 4 timer outputs via SW |

### 20.1.2 Block Diagram

Each CCU8 timer slice can operate independently from the other slices for all the available modes. Each timer slice contains a dedicated input selector for functions linked with external events and has 4 dedicated compare output signals, for PWM signal generation.

The built-in timer concatenation is only possible with adjacent slices, e.g. CC80/CC81. Combinations for slice concatenations like, CC80/CC82 or CC80/CC83 are not possible.

The individual service requests for each timer slice (four per slice) are multiplexed into four module service requests lines, **Figure 20-1**.


Subject to Agreement on the Use of Product Information

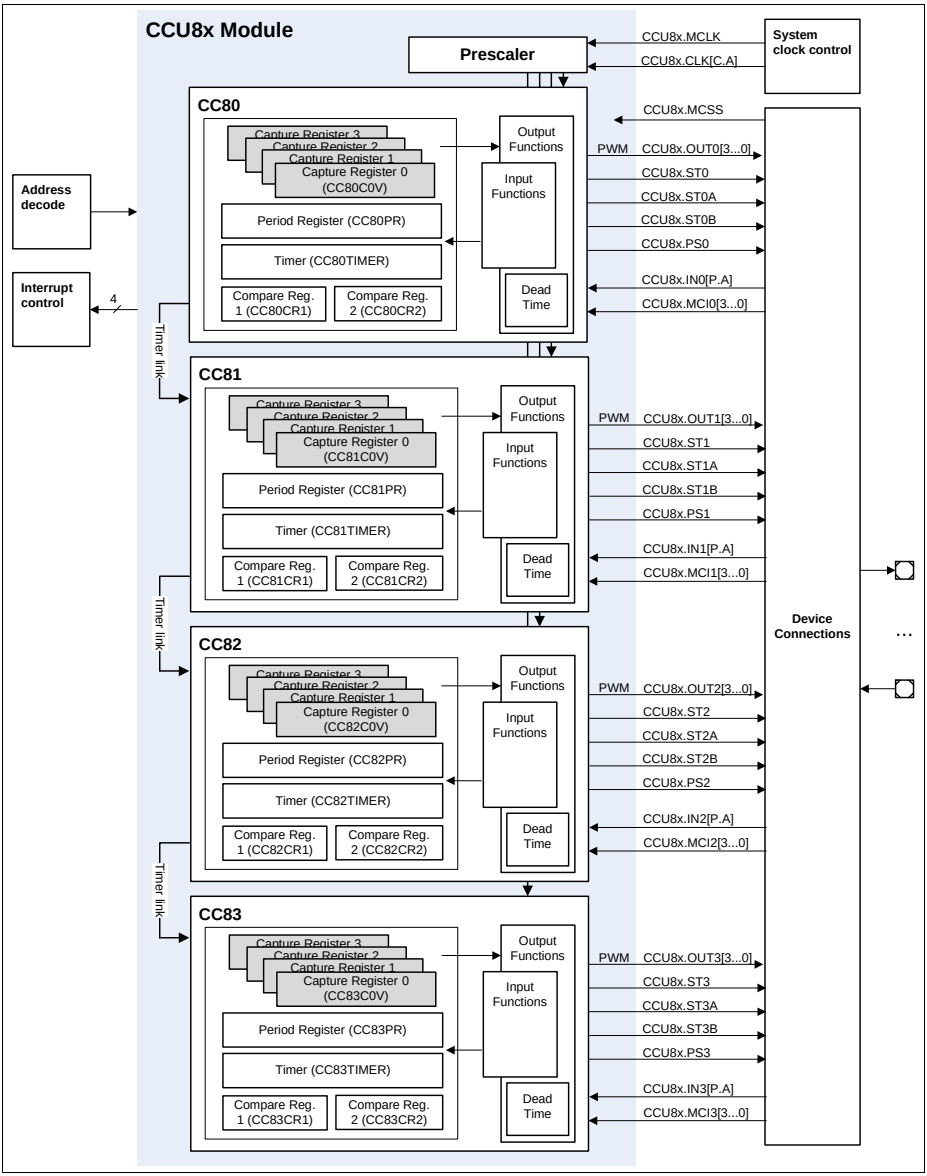



**Figure 20-1 CCU8 block diagram**

## 20.2 Functional Description

### 20.2.1 Overview

The input path of a CCU8 slice is comprised of a selector (**Section 20.2.2**) and a connection matrix unit (**Section 20.2.3**). The output path contains a service request control unit, a timer concatenation unit and two units that control directly the state of the output signal for each specific slice (for TRAP, dead time generation and modulation handling), see **Figure 20-2**.

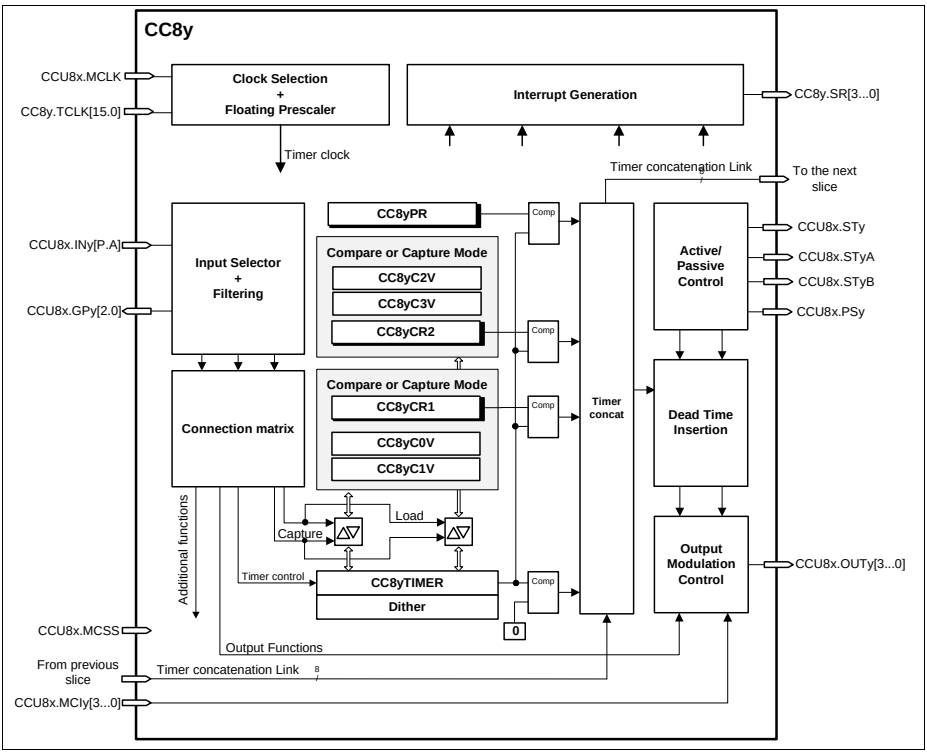



**Figure 20-2 CCU8 slice block diagram**

The timer core is built of 16 bit counter and one period register and two compare channels in compare mode, or four capture channels plus the period register in capture mode.

Individual timer clocks can be selected for each and every Timer Slice, enabling a very flexible resource organization inside each CCU8 module.

In compare mode the period sets the maximum counting value while the two compare registers are used to control the ACTIVE/PASSIVE state of the four dedicated comparison slice outputs.

Each CCU8 slice contains a dedicated timer link interface that is used to perform timer concatenation, up to 64 bits. This timer concatenation is controlled via a single bit field configuration.

**Table 20-3** describes the inputs and outputs for each CCU8 Timer Slice.

Inputs and outputs that are not seen at the CCU8 module boundaries have a nomenclature of CC8y.<name>, whilst CCU8 module inputs and outputs are described as CCU8x.<signal_name>y (indicating the variable y the object slice).

**Table 20-3 CCU8 slice pin description**

| Pin | I/O | Description |
|---|---|---|
| CCU8x.MCLK | I | Module clock |
| CC8y.TCLK | | Clock from the pre scaler |
| CCU8x.INy[P:A] | I | Slice functional inputs (used to control the functionality throughout slice external events) |
| CCU8x.MCIy[3...0] | I | Multi Channel mode inputs |
| CCU8x.MCSS | I | Multi Channel shadow transfer trigger |
| CC8y.SR[3...0] | O | Slice service request lines |
| CCU8x.GPy[2...0] | O | Signals decoded from the input selector (used for the parity checker function) |
| CCU8x.STy | O | This signal can be the slice comparison status value of channel 1, channel 2 or a AND between both |
| CCU8x.STyA | O | Slice comparison status value of channel 1 |
| CCU8x.STyB | O | Slice comparison status value of channel 2 |
| CCU8x.PSy | O | Period match |
| CCU8x.OUTy[3...0] | O | Slice dedicated output pins |

*Note:*

7. *The status bit outputs of the Kernel, CCU8x.STy, CCU8x.STyA and CCU8x.STyB are extended for one more kernel clock cycle.*
8. *The Service Request signals at the output of the kernel are extended for one more kernel clock cycle.*


Subject to Agreement on the Use of Product Information

9. *The maximum output signal frequency of the CCU8x.STy, CCU8x.STyA and CCU8x.STyA is module clock divided by 4.*

The slice timer, can count up or down depending on the selected operating mode. A direction flag contains the actual counting direction.

The timer is connected to three stand alone comparators, one for the period match and two for the compare match of each compare channel. The registers used for comparison match of both compare channels, can be programmed to serve as capture registers, enabling sequential capture capabilities on external events.

In normal edge aligned counting scheme, the counter is cleared to $0000_H$ each time it matches the period value defined in the period register. In center aligned mode, the counter direction changes from ‘up counting’ to ‘down counting’ after reaching the period value. Both period and compare registers have an aggregated shadow register, which enables the update of the PWM period and duty cycle on the fly.

A single shot mode is also available, where the counter stops after it reaches the value set in the period register.

The start and stop of the counter can be set/clear by software access or by a programmable input pin.

The dead time generator can be programmed with different values for the rising and falling edge of the output.

Functions like, load, counting direction (up/down), TRAP, output modulation can also be controlled with external events, see **Section 20.2.3**.

### 20.2.2 Input Selector

The first unit of the slice input path, is used to select from which are used to control the available external functions.

Inside this block the user also has the possibility to perform a low pass filtering of the signals and selecting the active edge(s) or level of the external event, see **Figure 20-3**.

The user has the possibility of selecting any of the CCU8x.INy[P:A] inputs has the source of an event.

At the output of this unit we have a user selection of three events, that were configured to be active at rising, falling or both edges, or level active. These selected events can then be mapped to several functions.

Notice that each decoded event contains two outputs, one edge active and one level active, due to the fact that some functions like counting, capture or load are edge sensitive events while, timer gating or up down counting selection are level active.


Subject to Agreement on the Use of Product Information

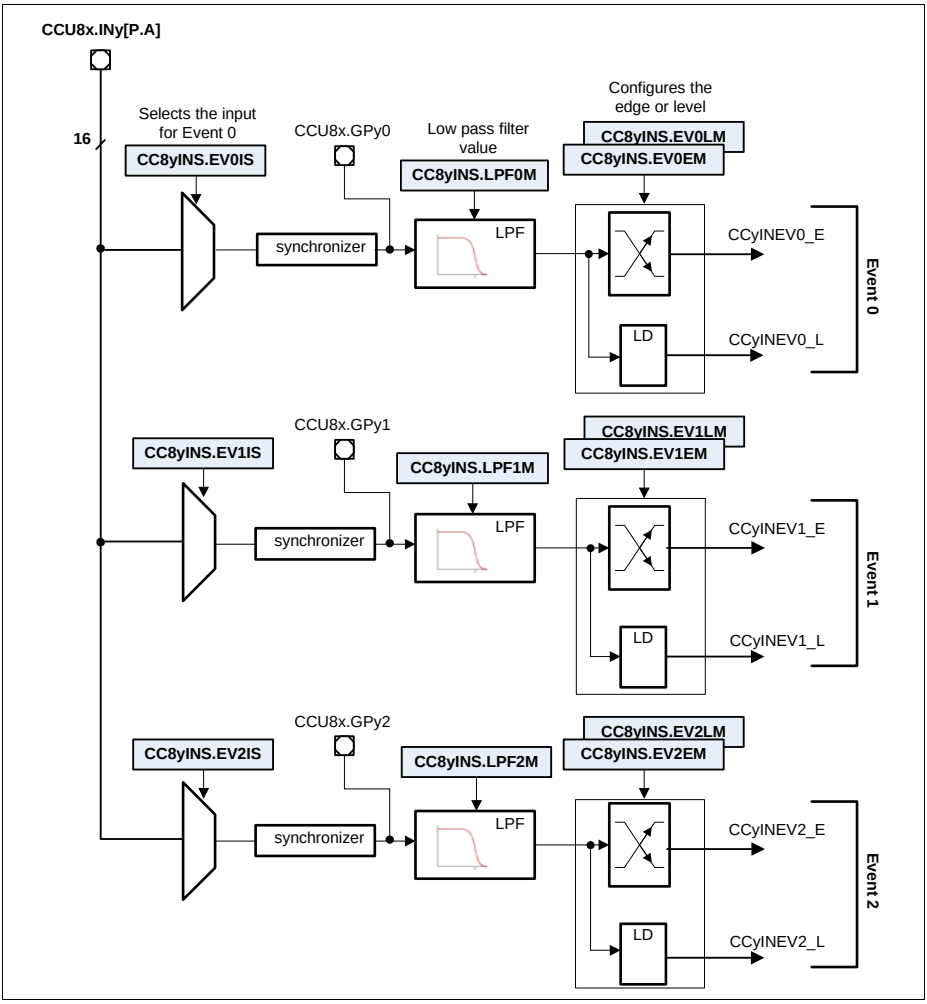



**Figure 20-3 Slice input selector diagram**

### 20.2.3 Connection Matrix

The connection matrix maps the events coming from the input selector to several user configured functions, **Figure 20-4**. The following functions can be enabled on the connection matrix:

**Table 20-4 Connection matrix available functions**

| Function | Brief description | Map to figure Figure 20-4 |
|---|---|---|
| Start | Edge signal to start the timer | CCystrt |
| Stop | Edge signal to stop the timer | CCystp |
| Count | Edge signal used for counting events | CCycnt |
| Up/down | Level signal used to select up or down counting direction | CCyupd |
| Capture 0 | Edge signal that triggers a capture into the capture registers 0 and 1 | CCycapt0 |
| Capture 1 | Edge signal that triggers a capture into the capture registers 2 and 3 | CCycapt1 |
| Gate | Level signal used to gate the timer clock | CCygate |
| Load | Edge signal that loads the timer with the value present at the compare register | CCyload |
| TRAP | Level signal used for fail-safe operation | CCytrap |
| Modulation | Level signal used to modulate/clear the output | CCymod |
| Status bit override | Status bit is going to be overridden with an input value | CCyoval for the value CCyoset for the trigger |

Inside the connection matrix we also have a unit that performs the built-in timer concatenation. This concatenation enables a completely synchronized operation between the concatenated slices for timing operations and also for capture and load actions. The timer slice concatenation is done via the **CC8yCMC**.TCE bitfield. For a complete description of the concatenation function, please address **Section 20.2.10**.


Subject to Agreement on the Use of Product Information

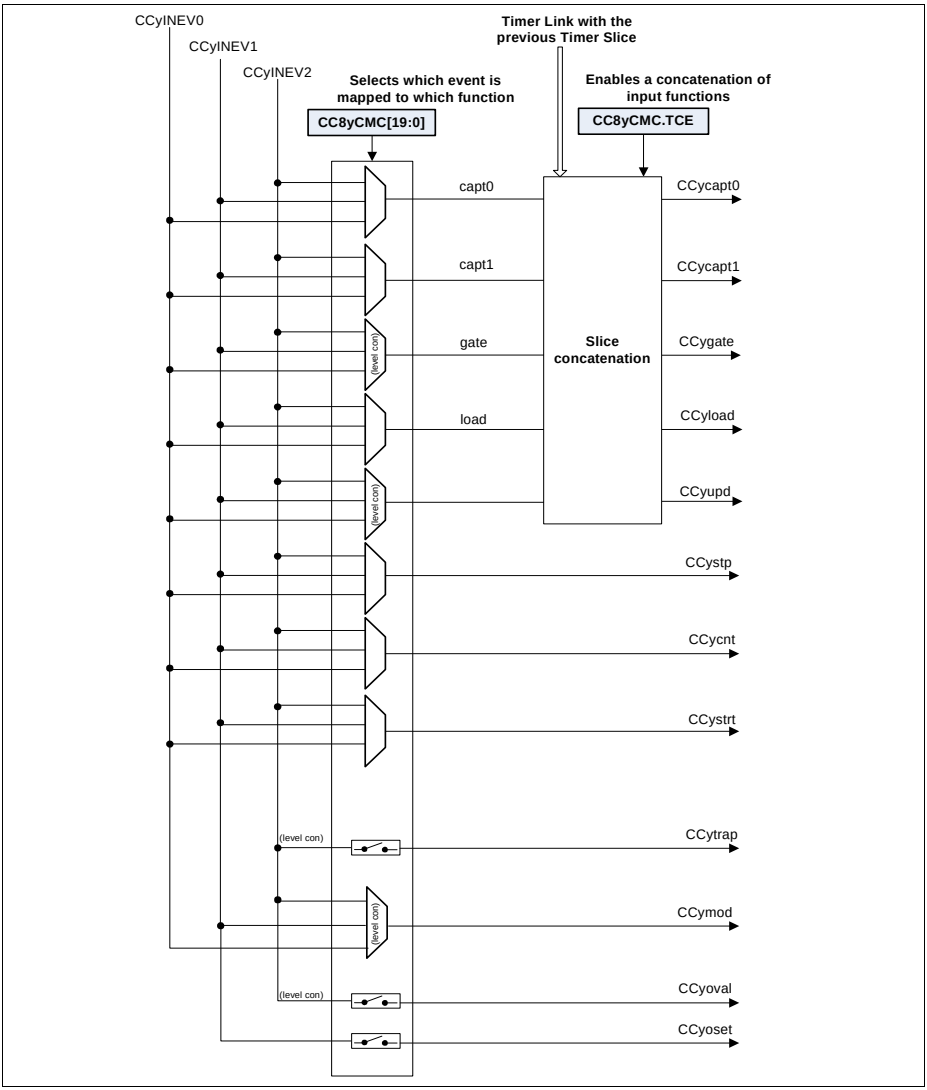



**Figure 20-4 Slice connection matrix diagram**

### 20.2.4 Start/Stop Control

Each slice contains a run bit register, that indicates the actual status of the timer, **CC8yTCST**.TRB. The start and stop of the timer can be done by software access or can be controlled directly by external events, see **Figure 20-5**.

Selecting an external signal that acts as a start trigger does not force the user to use an external stop trigger and vice versa.

Selecting the single shot mode, imposes that after the counter reaches the period value the run bit, **CC8yTCST**.TRB, is going to be cleared and therefore the timer is stopped.

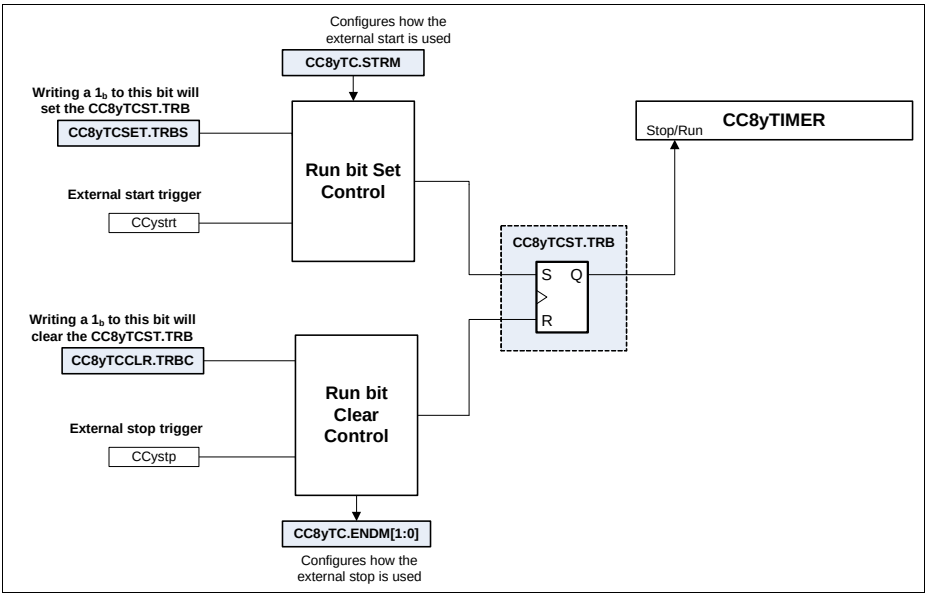



**Figure 20-5 Timer start/stop control diagram**

One can use the external stop signal to perform the following functions (configuration via **CC8yTC**.ENDM):

- Clear the run bit (stops the timer) - default
- Clear the timer (to $0000_H$) but it does not clear the run bit (timer still running)
- Clear the timer and the run bit

One can use the external start to perform the following functions (configuration via **CC8yTC**.STRM):

- Start the timer (resume operation)
- Clear and starts the timer


Subject to Agreement on the Use of Product Information

The set (start the timer) of the timer run bit, always has priority over a clear (stop the timer).

To start multiple CCU8 timers at the same time/synchronously one should use a dedicated input as external start (see **Section 20.2.8.1** for a description how to configure an input as start function). This input should be connected to all the Timers that need to started synchronously (see **Section 20.8** for a complete list of module connections), **Figure 20-6**.

For starting the timers synchronously via software there is a dedicated input signal, controlled by the SCU (System Control Unit), that is connected to all the CCU8 timers. This signal should then be configured as an external start signal (see **Section 20.2.8.1**) and then the software must write a $1_B/0_B$ (depending on the external start signal configuration) to the specific bitfield of the CCUCON register (this register is described on the SCU chapter).

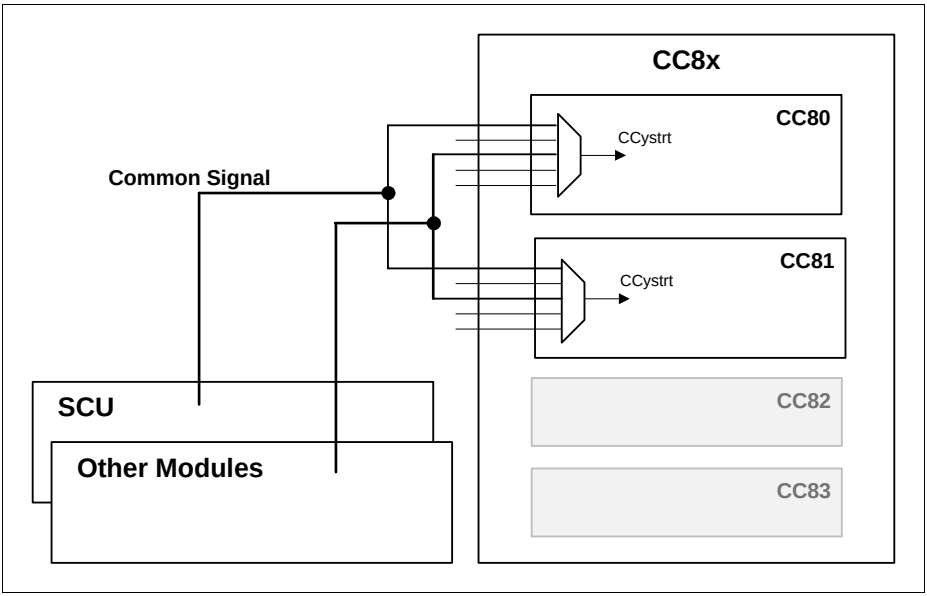



**Figure 20-6 Start multiple timers synchronously**

## 20.2.5 Counting Modes

Each CC8y timer can be programmed into three different counting schemes:

- Edge aligned (default)
- Center aligned
- Single shot (edge or center aligned)


Subject to Agreement on the Use of Product Information

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 20-7**. The set and clear of the status bits and the respective PWM signal generation is dictated by the timer period, compare value and the current counting mode. See the different counting mode descriptions, **Section 20.2.5.3** to **Section 20.2.5.5** to understand how these bits are set and cleared.

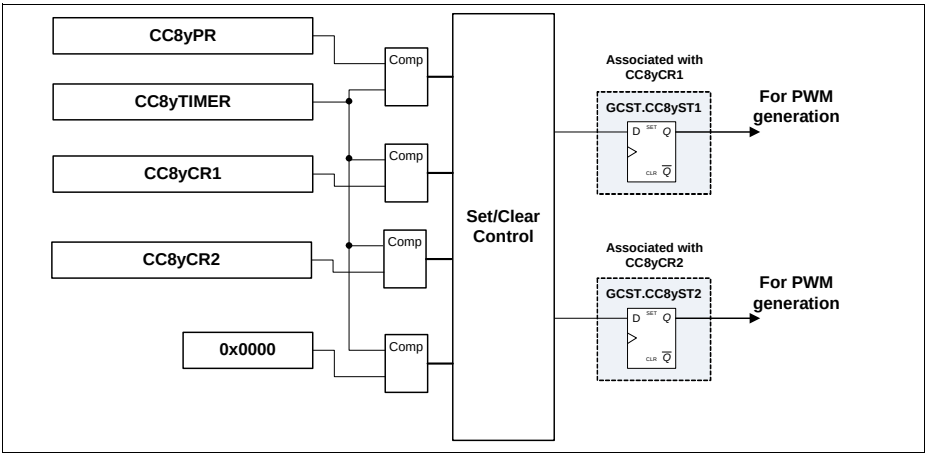



**Figure 20-7 CC8y Status Bits**

### 20.2.5.1 Calculating the PWM Period and Duty Cycle

The period of the timer is determined by the value in the period register, **CC8yPR** and by the timer mode.

The base for the PWM signal frequency and duty cycle, is always related to the clock frequency of the timer itself and not to the frequency of the module clock (due to the fact that the timer clock can be a scaled version of the module clock).

In Edge Aligned Mode, the timer period is:

$T_{per}$= <Period-Value> + 1; in $f_{tclk}$ (20.1)

In Center Aligned Mode, the timer period is:

$T_{per}$= (<Period-Value> + 1) x 2; in $f_{tclk}$ (20.2)

For each of these counting schemes, the duty cycle of generated PWM signal is dictated by the value programmed into the compare channel registers, **CC8yCR1** and **CC8yCR2**. Notice that one can have different duty cycle values for each of the compare channels.

In Edge Aligned and Center Aligned Mode, the PWM duty cycle is:

$DC$= 1 - <Compare-Value>/(<Period-Value> + 1) (20.3)

Both the period and compare registers, **CC8yPR**, **CC8yCR1** and **CC8yCR2** respectively, can be updated on the fly via software enabling a glitch free transition between different period and duty cycle values for the generated PWM signal, **Section 20.2.5.2**

### 20.2.5.2 Updating the Period and Duty Cycle

**Standard Shadow Transfer Control**

Each CCU8 timer slice provides an associated shadow register for the period and the two compare values. This facilitates a concurrent update by software for these three parameters, with the objective of modifying during run time the PWM signal period and duty cycle.

In addition to the shadow registers for the period and compare values, one also has available shadow registers for the floating prescaler, dither and passive level, **CC8yFPCS**, **CC8yDITS** and **CC8yPSL** respectively (please address **Section 20.2.13** and **Section 20.2.12** for a complete description of these functions).

The structure of the shadow registers can be seen in **Figure 20-8**.

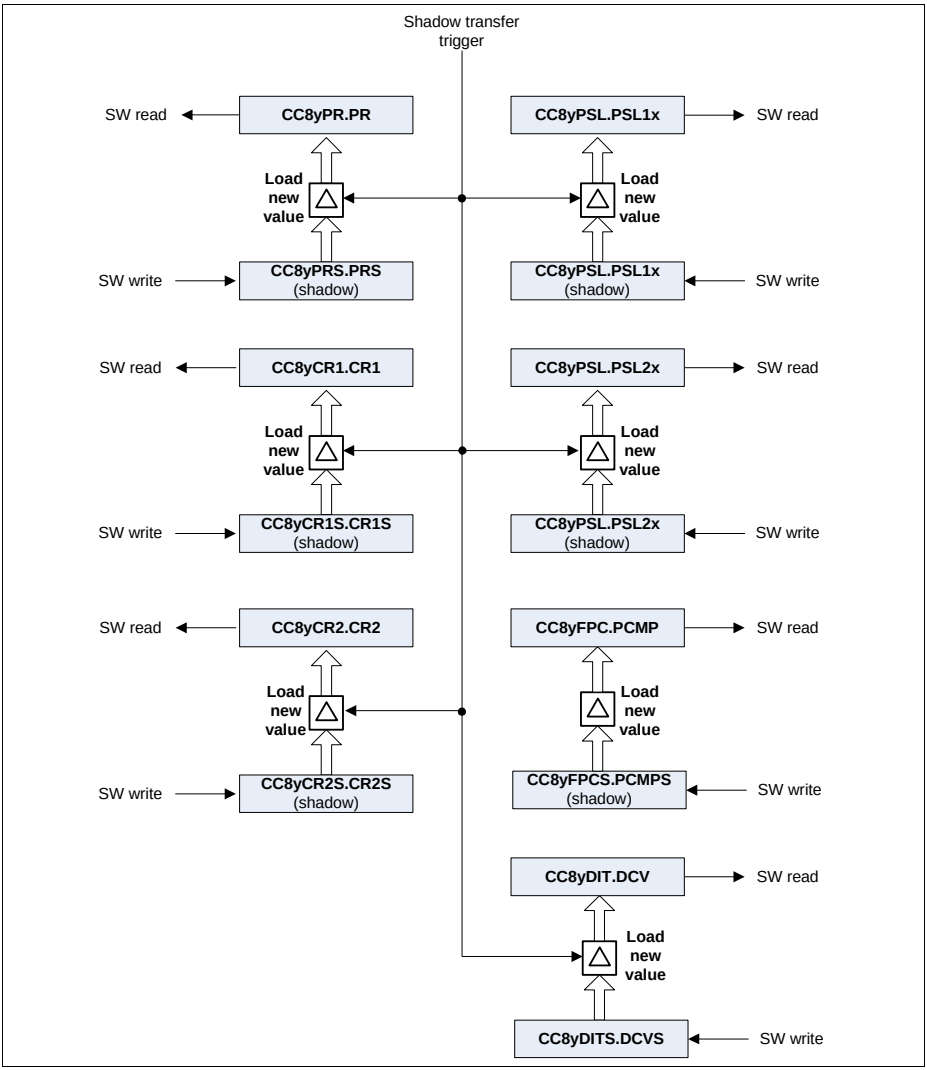



**Figure 20-8 Shadow registers overview**

The update of these registers can only be done by writing a new value into the associated shadow register and wait for a shadow transfer to occur.

Each group of shadow registers have an individual shadow transfer enable bit, **Figure 20-9**. The software must set this enable bit to $1_B$, whenever an update of the

values is needed. These bits are automatically cleared by the hardware, whenever an update of the values if finished. Therefore every time that an update of the registers is needed the software must set again the specific bit(s).

Nevertheless it is also possible to clear the enable bit via software. This can be used in the case that an update of the values needs to be cancelled (after the enable bit has already been set).

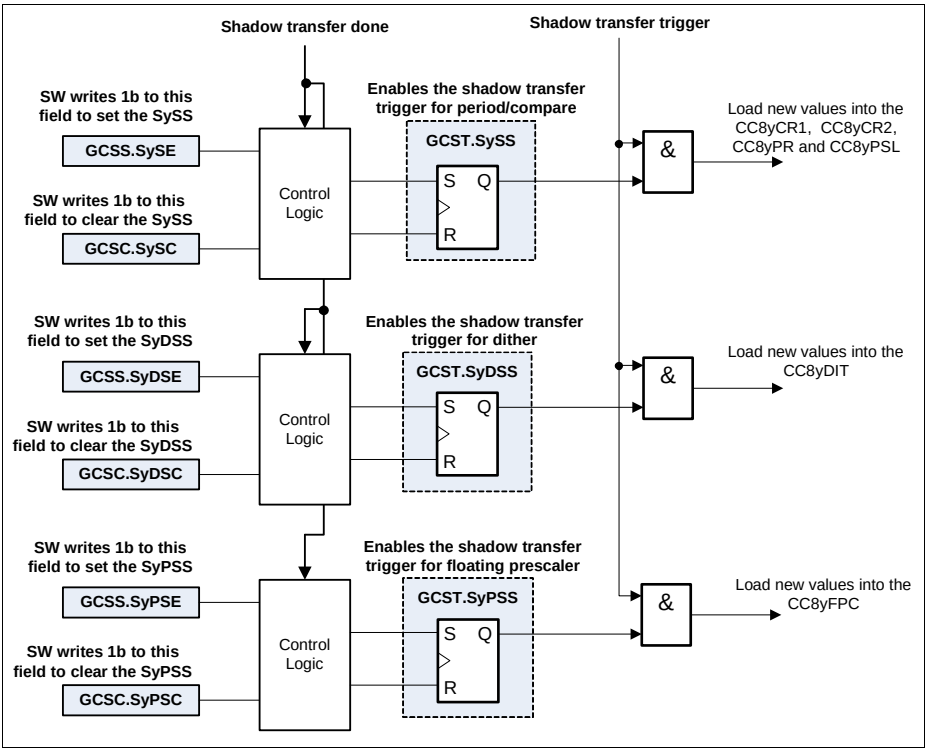



**Figure 20-9 Shadow transfer enable logic**

The shadow transfer operation is going to be done in the immediately next occurrence of a shadow transfer trigger, after the shadow transfer enable is set (**GCST**.SySS, **GCST**.SyDSS, **GCST**.SyPSS set to $1_B$).

The occurrence of the shadow transfer trigger is imposed by the timer counting scheme (edge aligned or center aligned). Therefore the slots when the values are updated can be:

- in the next clock cycle after a Period Match while counting up
- in the next clock cycle after an One Match while counting down

- immediately, if the timer is stopped and the shadow transfer enable bit(s) is set

**Figure 20-10** shows an example of the shadow transfer control when the timer slice has been configured into center aligned mode. For a complete description of all the timer slice counting modes, please address **Section 20.2.5.3**, **Section 20.2.5.4** and **Section 20.2.5.5**.

It is also possible to control in which slot the shadow transfer is done, via the STM field. This is only valid in Center Aligned Mode:

- **CC8ySTC**.STM = $00_B$ (default) - Shadow transfer is done at the Period Match and One match slot
- **CC8ySTC**.STM = $01_B$ - Shadow transfer is done only at the Period Match slot
- **CC8ySTC**.STM = $10_B$ - Shadow transfer is done only at the One Match slot

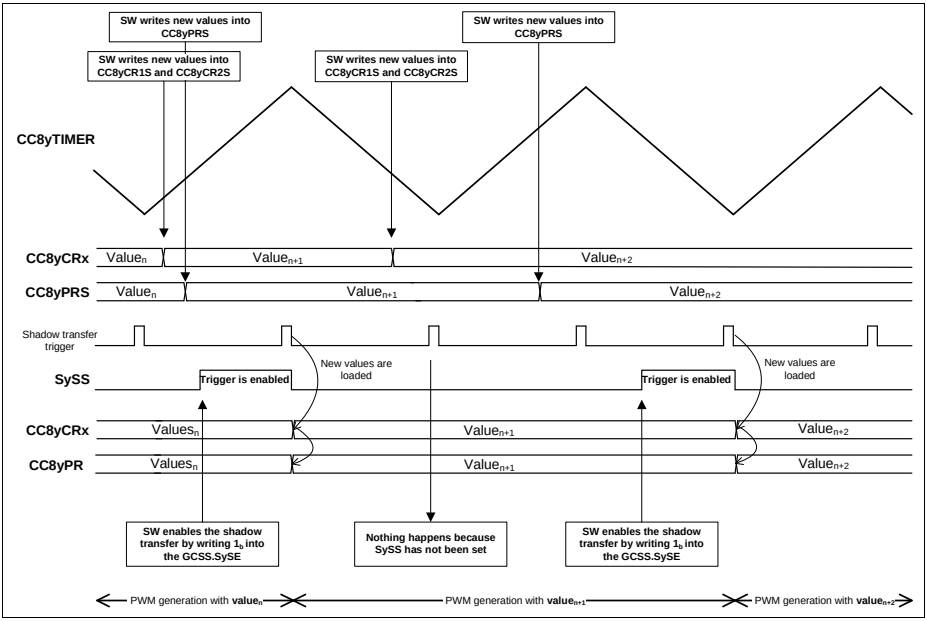



**Figure 20-10 Shadow transfer timing example - center aligned mode**

When using the CCU8 in conjunction with the POSIF to control the multi channel mode, it can be necessary in some cases, to perform the shadow transfers synchronously with the update of the multi channel pattern. To perform this action, each CCU8 contains a dedicated input that can be used to synchronize the two events, the CCU8x.MCSS.

This input, when enabled, is used to set the shadow transfer enable bitfields (**GCST**.SySS, **GCST**.SyDSS and **GCST**.SyPSS) of the specific slice. It is possible to select which slice is using this input to perform the synchronization via the **GCTRL**.MSEy

bit field. It is also possible to enable the usage of this signal for the three different shadow transfer signals: compare and period values, dither compare value and prescaler compare value. This can be configured on the **GCTRL**.MSDE field.

The structure for using the CCU8x.MCSS input signal can be seen in **Figure 20-11**. The usage of this signal is just an add on to the shadow transfer control and therefore all the previous described functions are still available.

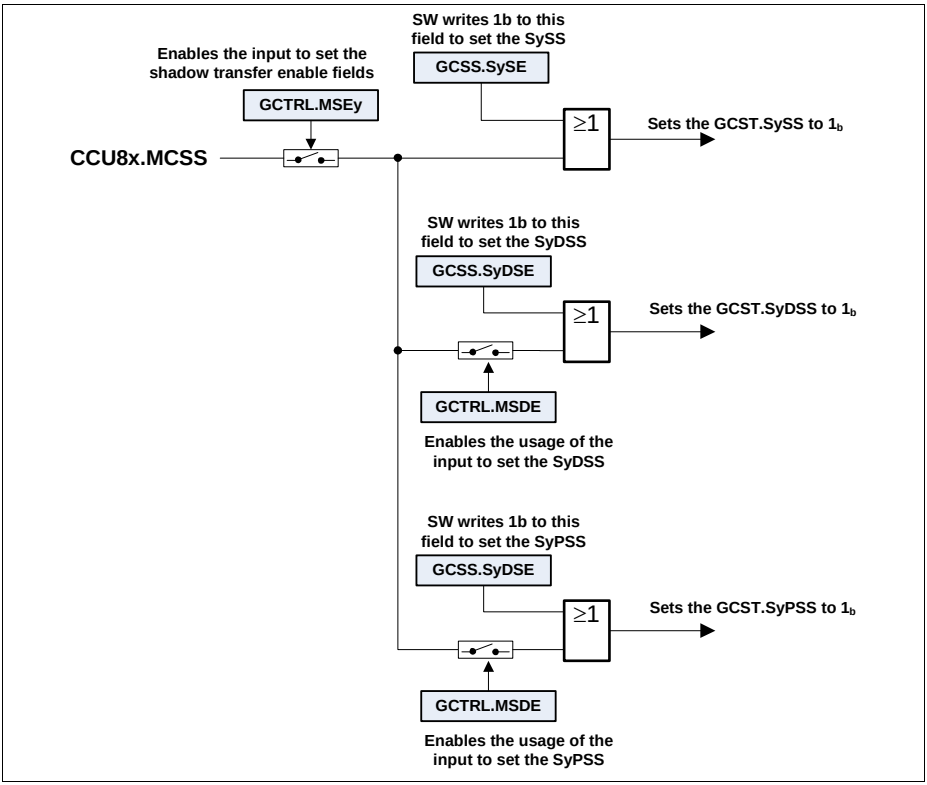



**Figure 20-11 Usage of the CCU8x.MCSS input**

### Cascaded Shadow Transfer

It is possible to cascade the shadow transfer operation throughout the CCU8 timer slices. The specific shadow transfer of a timer slice is cascaded with the adjacent timer slices, **Figure 20-12**.

To enable the cascaded shadow transfer function, the bitfield **CC8ySTC**.CSE of the specific timer slice needs to be set to $1_B$.

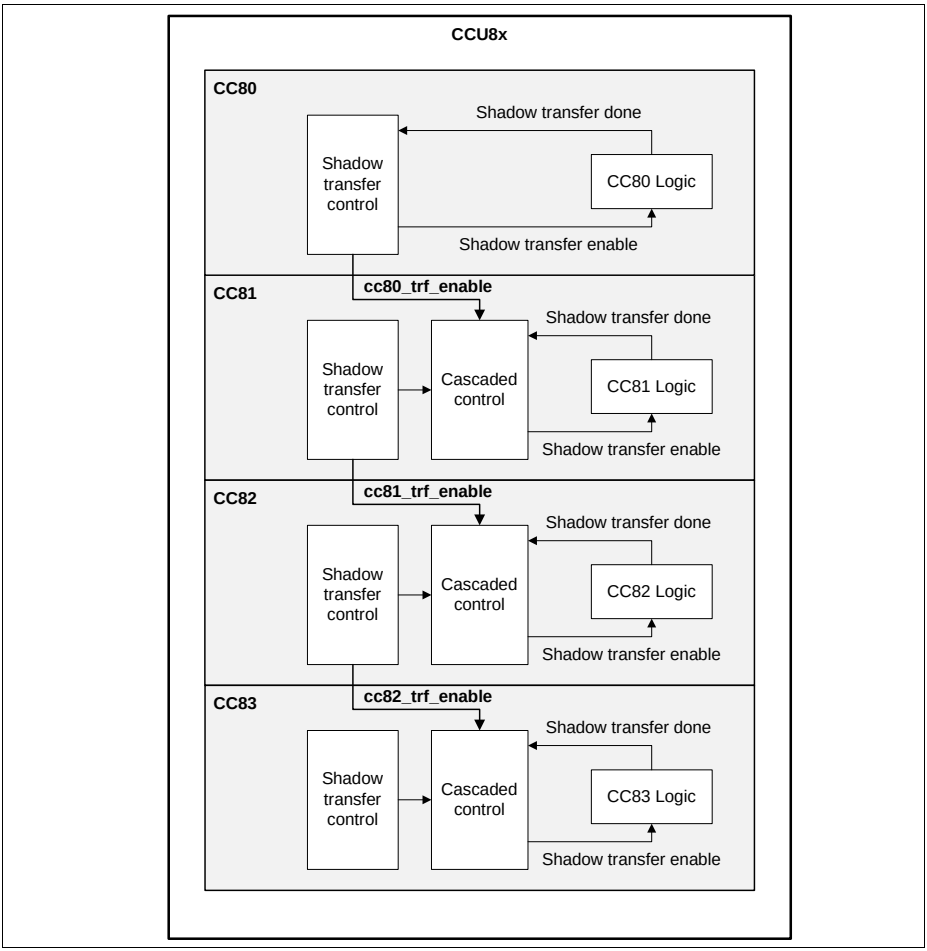



**Figure 20-12 Cascaded shadow transfer linking**

The shadow transfer enable bits, still need to be set via SW for each of the individual slices, **Figure 20-13**.

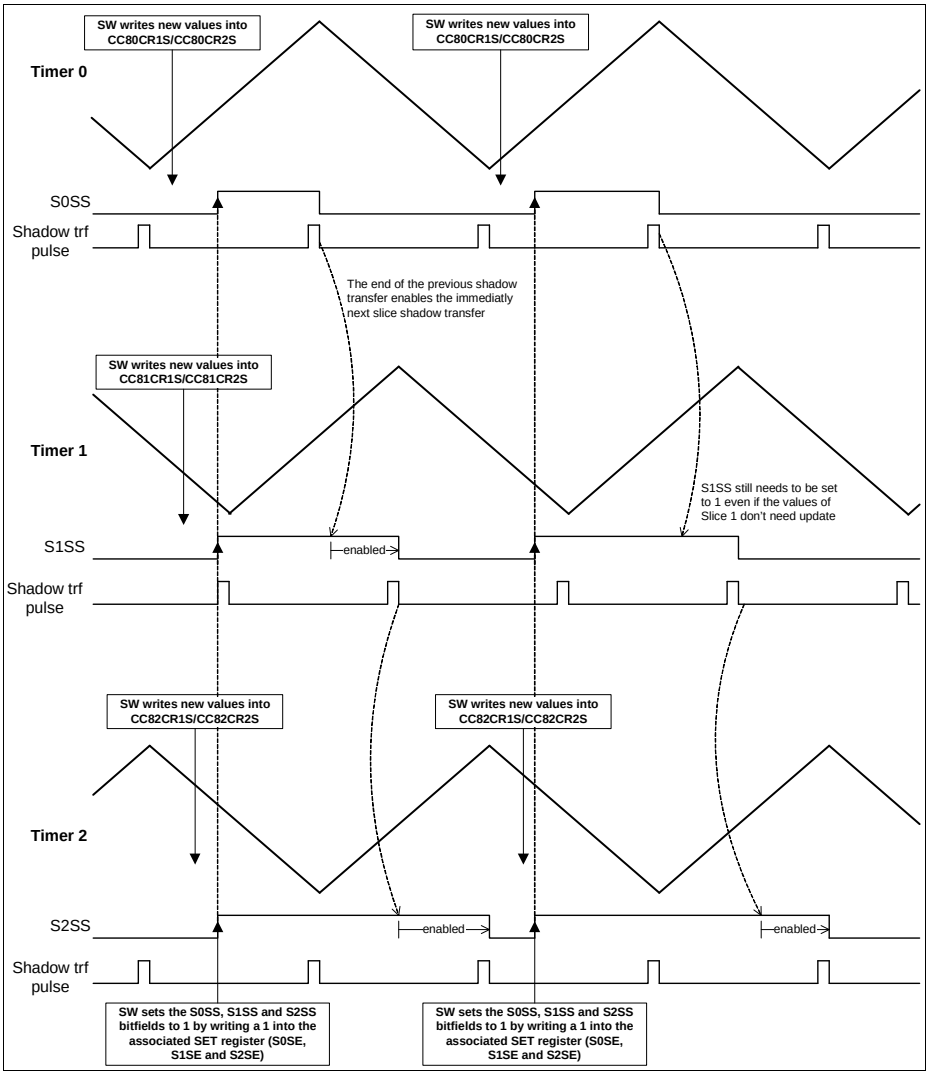



**Figure 20-13 Cascade shadow transfer timing**

*Note: The shadow transfer enable bits, need to be set in all timer slices that are being used in the cascaded architecture, at the same time. The shadow transfer enable bits, also need to be set for all slices even if the shadow values of some slices were not updated.*

### 20.2.5.3 Edge Aligned Mode

Edge aligned mode is the default counting scheme. In this mode, the timer is incremented until it matches the value programmed in the period register, **CC8yPR**. When period match is detected the timer is cleared to $0000_H$ and continues to be incremented.

In this mode, the value of the period register and compare registers are updated with the values written by software into the correspondent shadow register, every time that an overflow occurs (period match), see **Figure 20-14**.

In edge aligned mode, the status bit of the comparison (CC8ySTx) is set one clock cycle after the timer hits the value programmed into the compare register. The clear of the status bit is done one clock cycle after the timer reaches $0000_H$.

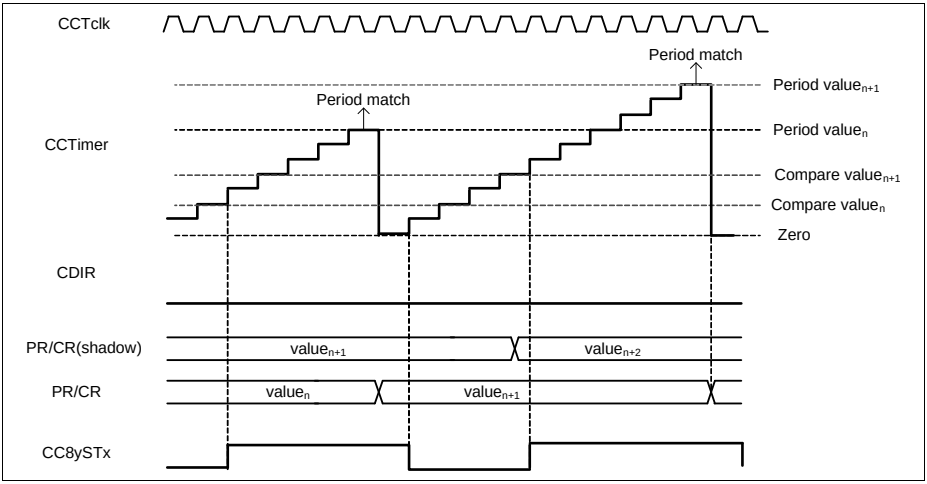



**Figure 20-14 Edge aligned mode, CC8yTC.TCM = $0_B$**

### 20.2.5.4 Center Aligned Mode

In center aligned mode, the timer is counting up or down with respect to the following rules:

- The counter counts up while **CC8yTCST**.CDIR = $0_B$ and it counts down while **CC8yTCST**.CDIR = $1_B$.
- Within the next clock cycle, the count direction is set to counting up (**CC8yTCST**.CDIR = $0_B$) when the counter reaches $0001_H$ while counting down.
- Within the next clock cycle, the count direction is set to counting down (**CC8yTCST**.CDIR = $1_B$), when the period match is detected while counting up.


Subject to Agreement on the Use of Product Information

The status bit (CC8ySTx) is always $1_B$ when the counter value is equal or greater than the compare value and $0_B$ otherwise.

While in edge aligned mode, the shadow transfer for compare and period registers is executed once per period. It is executed twice in center aligned mode as follows

- Within the next clock cycle after the counter reaches the period value, while counting up (**CC8yTCST**.CDIR = $0_B$).
- Within the next clock cycle after the counter reaches $0001_H$, while counting down (**CC8yTCST**.CDIR = $1_B$).

*Note: Bit **CC8yTCST**.CDIR changes within the next timer clock after the one-match or the period-match, which means that the timer continues counting in the previous direction for one more cycle before changing the direction.*

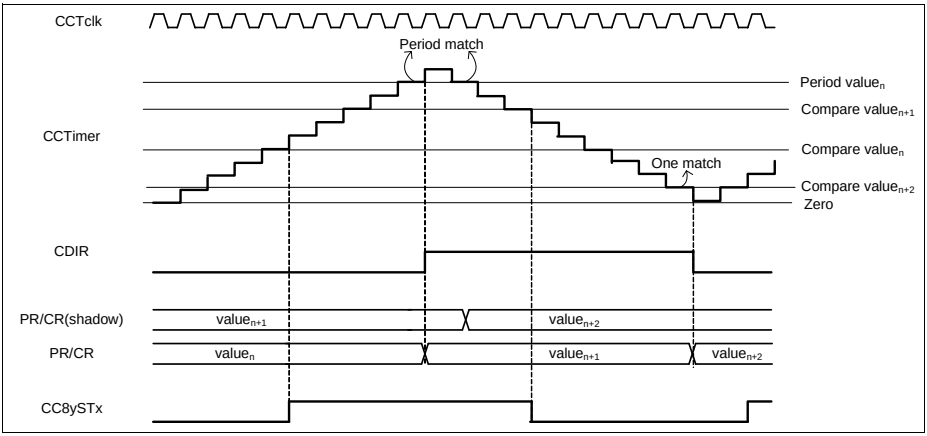



**Figure 20-15 Center aligned mode, CC8yTC.TCM = $1_B$**

### 20.2.5.5 Single Shot Mode

In single shot mode, the timer is stopped after the current timer period is finished. This mode can be used with a center or edge aligned scheme.

In edge aligned mode, **Figure 20-16**, the timer is stopped when it is cleared to $0000_H$ after having reached the period value. In center aligned mode, **Figure 20-17**, the period is finished when the timer has counted down to $0000_H$.

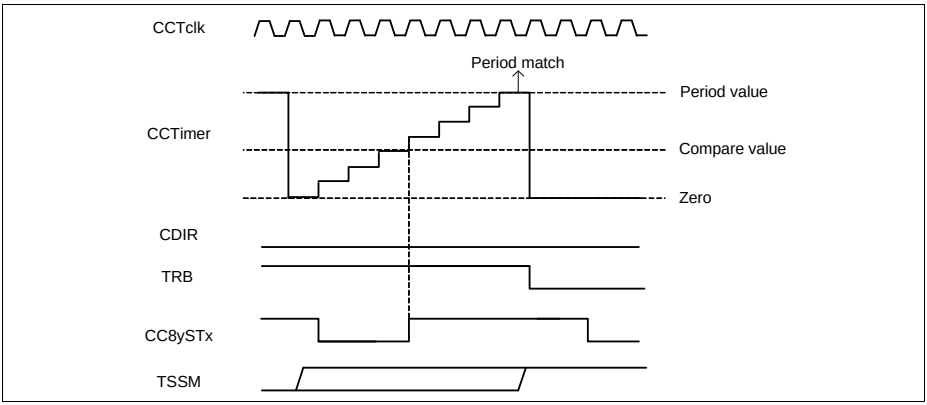



**Figure 20-16 Single shot edge aligned - CC8yTC.TSSM = $1_B$, CC8yTC.TCM = $0_B$**

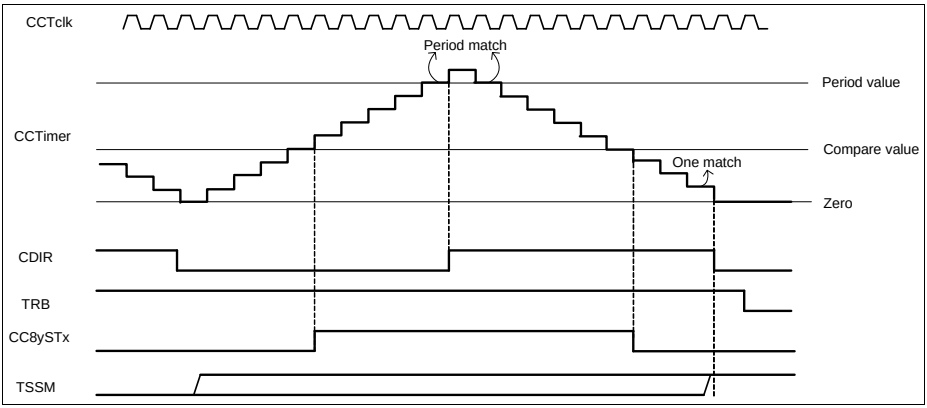



**Figure 20-17 Single shot center aligned - CC8yTC.TSSM = $1_B$, CC8yTC.TCM = $1_B$**

### 20.2.6 Active/Passive Rules

The general rules that set or clear the associated timer slice status bit, can be generalized independently of the timer counting mode.

The following events set the Status bit to Active:

- in the next $f_{tclk}$ cycle after a compare match while counting up
- in the next $f_{tclk}$ cycle after a zero match while counting down

The following events set the Status bit to Inactive:

- in the next $f_{tclk}$ cycle after a zero match (and not compare match) while counting up
- in the next $f_{tclk}$ cycle after a compare match while counting down

If external events are being used to control the timer operation, these rules are still applicable.

The status bit state can only be ‘override’ via software or by the external status bit override function, **Section 20.2.8.10**.

The software at any time can write a $1_B$ into the **GCSS**.SySTS bitfield, which will set the status bit **GCST**.CC8yST of the specific timer slice. By writing a $1_B$ into the **GCSC**.SySTC bitfield, the software imposes a clear of the specific status bit.

## 20.2.7 Compare Modes

### Compare Channel Scheme

Each CCU8 slice has two compare channels and two dead time generators, one for each channel, see **Figure 20-18**. Each compare uses the information of the status bit, CC8ySTx, to generate two complementary outputs. All the outputs, CCU8x.OUTy0, CCU8x.OUTy1, CCU8x.OUTy2 and CCU8x.OUTy3, have a dedicated passive level control bit.

Each compare channel can work in an individual manner for both edge and center aligned modes. This means that two different complementary PWM signals can be generated by using the available compare channels. The PWM frequency is the same for both channels, but the duty cycle can be programmed independently for each channel.

It is also possible to select an asymmetric output scheme, by setting the field **CC8yCHC**.ASE = $1_B$. In the asymmetric mode, the compare channels are grouped together to generate a single complementary PWM signal at the CCU8x.OUTy0 and CCU8x.OUTy1 pins.

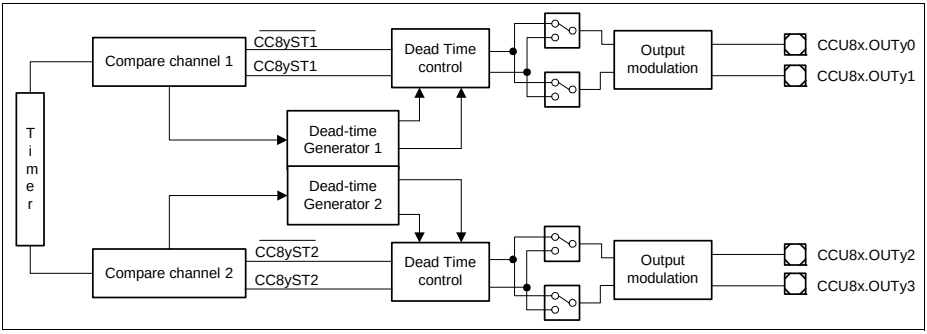



**Figure 20-18 Compare channels diagram**

### Dead Time Generator

In most cases the switching behavior regarding the switch-on and switch-off times is not symmetrical, which can lead to a short circuit if the switch-on time is smaller than the switch-off time. To overcome this problem, each Timer Slice channel contains a dead time generator, which is able to delay the switching edges of the output signals, **Figure 20-19**.

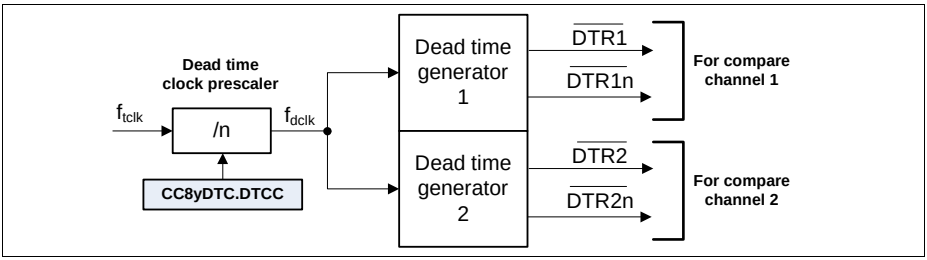



**Figure 20-19 Dead Time scheme**

Each dead time generator contains an eight bit counter with a different programmable reload value for rise and fall times. The dead time generators contain a programmable prescaler for the dead time counter clock, to enable large dead time insertion values, **Table 20-5**.

**Table 20-5 Dead time prescaler values**

| CC8yDTC.DTCC[1:0] | Frequency |
|---|---|
| $00_B$ | $f_{tclk}$ |
| $01_B$ | $f_{tclk}/2$ |
| $10_B$ | $f_{tclk}/4$ |
| $11_B$ | $f_{tclk}/8$ |

Any transition on the associated status bits, CC8ySTx, will trigger the start of the specific dead time generator, **Figure 20-20**.

When a SET (CC8ySTx passes from $0_B$ to $1_B$) action for the CC8ySTx bit is detected, the dead time counter is reloaded with the value present on the **CC8yDC1R**.DT1R or **CC8yDC2R**.DT2R (depending on which channel we are addressing).

When a CLEAR action for the CC8ySTx bit is detected (CC8ySTx passes from $0_B$ to $1_B$), the dead time counter is reloaded with the value present on the **CC8yDC1R**.DT1F or **CC8yDC2R**.DT2F (depending on which channel we are addressing).


Subject to Agreement on the Use of Product Information

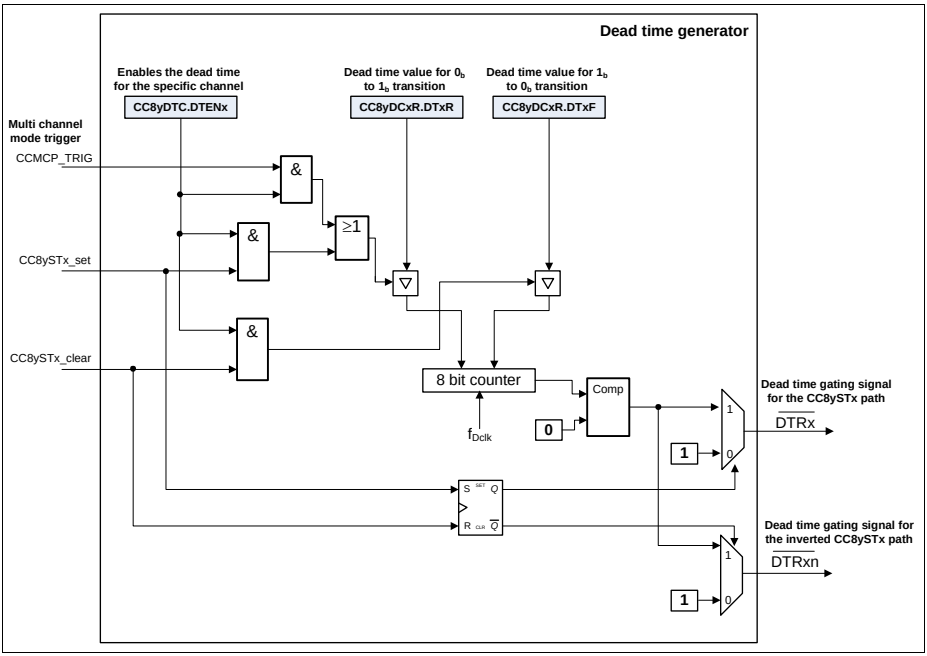



**Figure 20-20 Dead Time generator scheme**

Each dead time generator outputs two signals that are used to control the two complementary outputs (the CC8ySTx and the inverted CC8ySTx). The separation of the control signals enable a flexible enable/disable scheme inside of each compare channel, **Figure 20-21**. This means that the dead time generator can be enabled for one compare channel, but the dead time insertion can be discarded for one of the outputs.

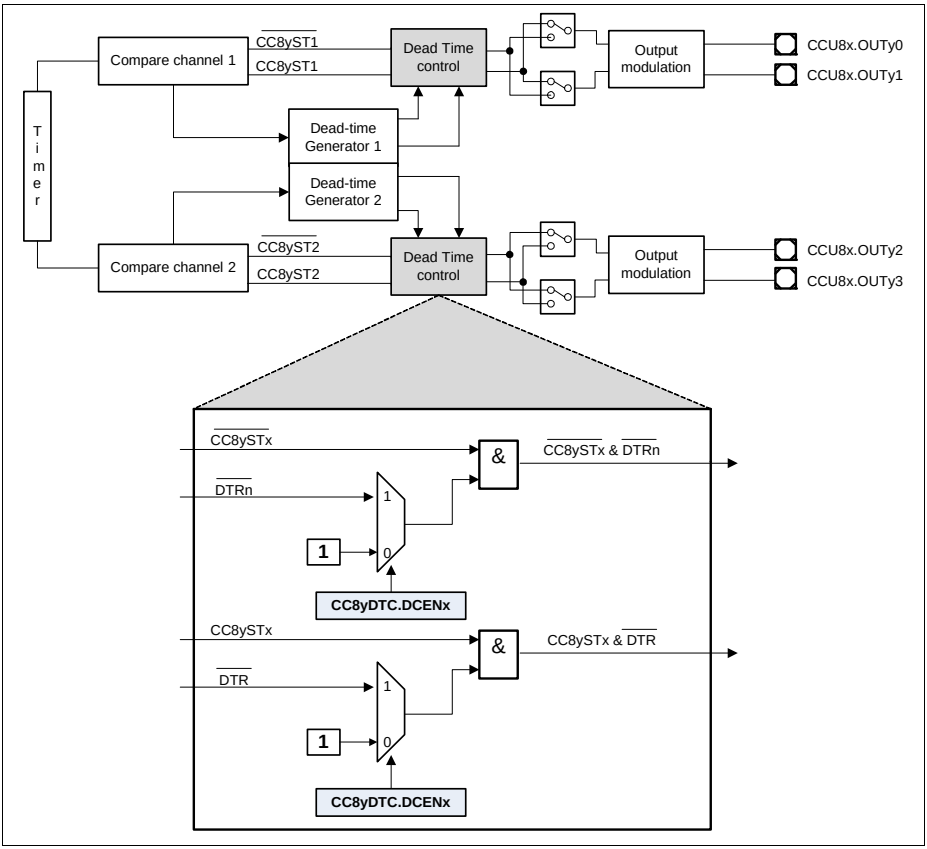



**Figure 20-21 Dead Time control cell**

When using the Multi Channel mode, **CC8yTC**.MCMEy = $1_B$, there can be the scenario where the generated PWM signal has 100% duty cycle. This means that the respective status bit is always set and it is the Multi Channel pattern that is controlling the output modulation. In this case, we can have a transition from Inactive to Active state at the output, without having a transition on the specific status bit, creating a short on the switches due to the non existence of dead time insertion.

To overcome this possible short on the switches, a trigger from the multi channel control, CCMP_TRIG on **Figure 20-20**, is fed to the dead time generators. **Figure 20-22** shows the scheme for the generation of the CCMP_TRIG, where the signals, CCMCMx0 and CCMCMx1 represent the sampled multi channel pattern for a specific channel.


Subject to Agreement on the Use of Product Information

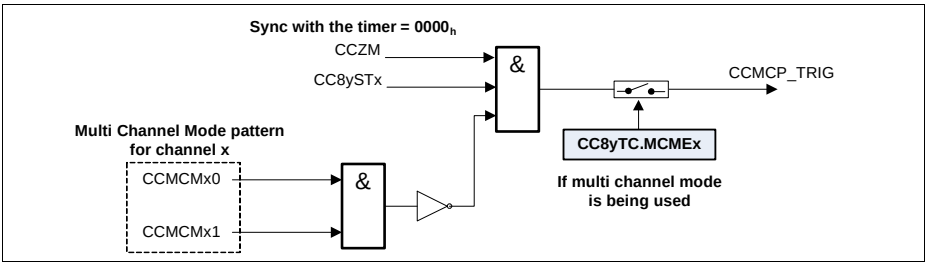



**Figure 20-22 Dead Time trigger with the Multi Channel pattern**

### 20.2.7.1 Edge Aligned Compare Modes

#### Standard Edge Aligned Mode

When the Timer Slice is programmed in edge aligned mode, the two channels can work independently, which means that the compare values can be programmed with different values (originating different duty cycles). In this scenario, each channel can output a pair of PWM signals used to control a high and low side switches, see **Figure 20-23**.

In this mode, for each channel the dead time for rise and fall transitions are controlled by the values programmed in the **CC8yDC1R**.DT1R and **CC8yDC2R**.DT2R, and **CC8yDC1R**.DT1F and **CC8yDC2R**.DT2F fields, respectively.

**Figure 20-24** shows the timing diagrams for a specific slice when the compare values of each channel are different.

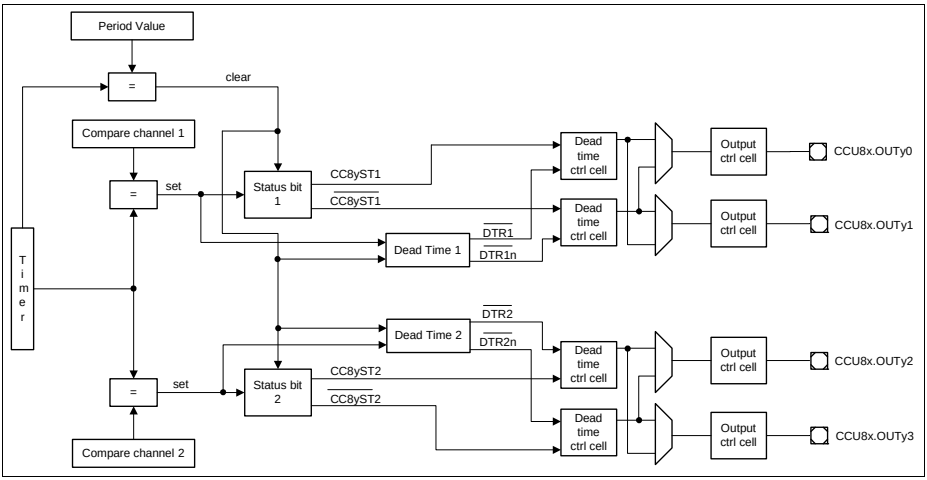



**Figure 20-23 Edge Aligned with two independent channels scheme**

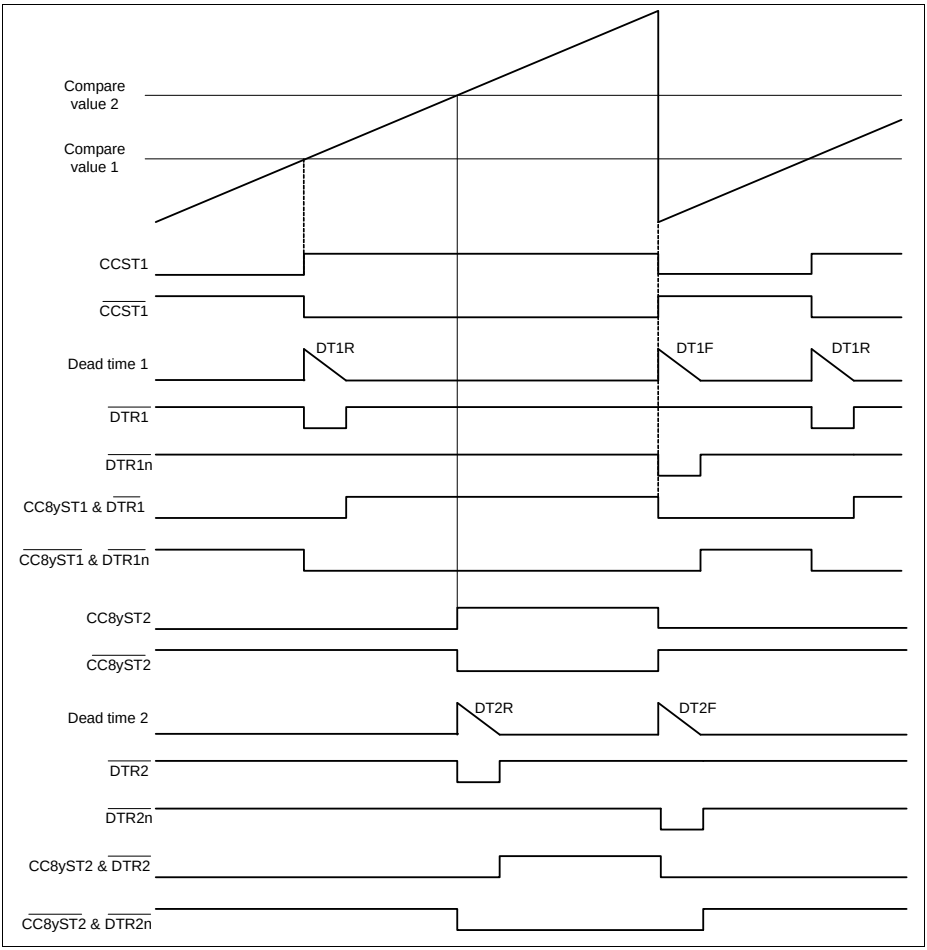



**Figure 20-24 Edge Aligned - four outputs with dead time**

### Asymmetrical Edge Aligned Mode

There is also the possibility of using the two channels combined to generate an asymmetric PWM output. This mode is selected by setting the field **CC8yCHC**.ASE = $1_B$.

In this mode, the compare channel 2 is disabled and therefore the outputs linked with this path are always in the passive state.

The status bit of the compare channel 1 is set when a compare match with the compare value 1 (field **CC8yCR1**.CR1) occurs and is cleared when a compare match with the compare value 2 (field **CC8yCR2**.CR2) occurs, see **Figure 20-25**.

When the **CC8yCR2**.CR2 is programmed with a value smaller than the one present in **CC8yCR1**.CR1, the CCST1 bit is always $0_B$.

The dead time values for the rising and falling transitions are controlled by the fields **CC8yDC1R**.DT1R and **CC8yDC1R**.DT1F, respectively.

**Figure 20-26** and **Figure 20-27** show the timing diagram for the Edge Aligned mode when the asymmetric scheme is active.

*Note: When an external signal is used to control the counting direction, the asymmetric mode cannot be enabled.*

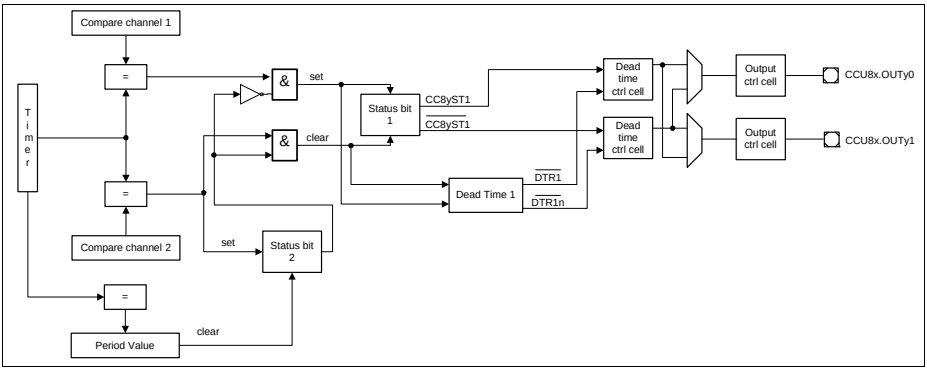



**Figure 20-25 Edge Aligned with combined channels scheme**

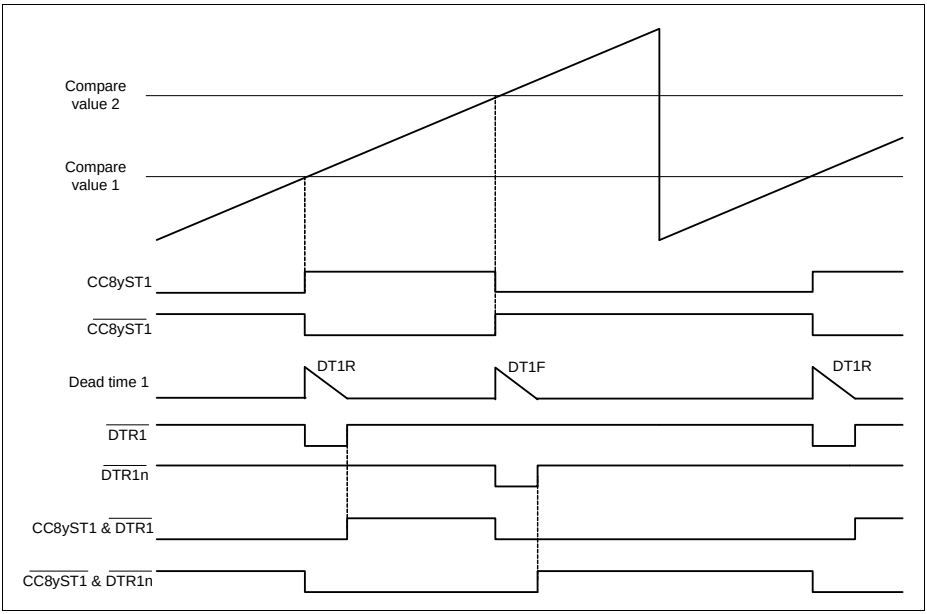



**Figure 20-26 Edge Aligned - Asymmetric PWM timing, CC8yCR1.CR1 < CC8yCR2.CR2**


Subject to Agreement on the Use of Product Information

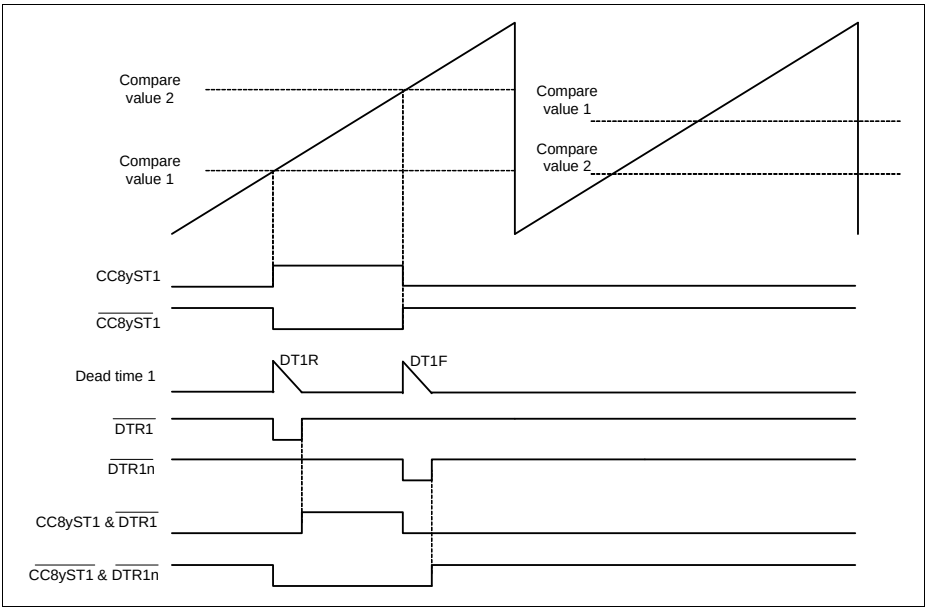



**Figure 20-27 Edge Aligned - Asymmetric PWM timing, CC8yCR1.CR1 > CC8yCR2.CR2**

### 20.2.7.2 Center Aligned Compare Modes

#### Standard Center Aligned Mode

In center aligned mode, like in edge aligned, it is possible to use the two compare channels independently. In this mode, each channel can generate a pair of PWM complementary signals with different duty cycle values, controlled via the **CC8yCR1** for channel 1 and **CC8yCR2** for channel 2.

For the dead time insertion, each channel as a pair of programmable values for the rise and fall transitions: **CC8yDC1R**.DT1R and **CC8yDC1R**.DT1F for channel 1; **CC8yDC2R**.DT2R and **CC8yDC2R**.DT2F for channel 2.

The major difference between the center and the edge aligned mode is directly linked to the set/clear logic of the status bit, see **Section 20.2.5**.

**Figure 20-28** shows the scheme for both channels for this operating mode and **Figure 20-29** shows the timing diagrams for a specific channel.

*Note: When an external signal is used to control the counting direction, the counting scheme is always edge aligned.*


Subject to Agreement on the Use of Product Information

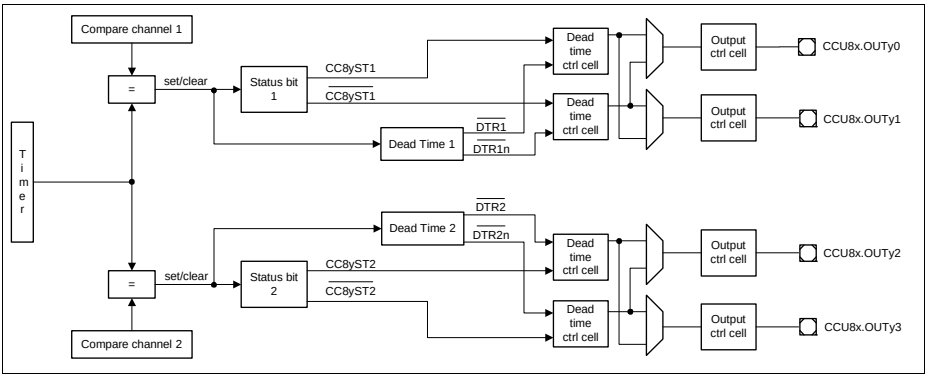



**Figure 20-28 Center Aligned with two independent channels scheme**

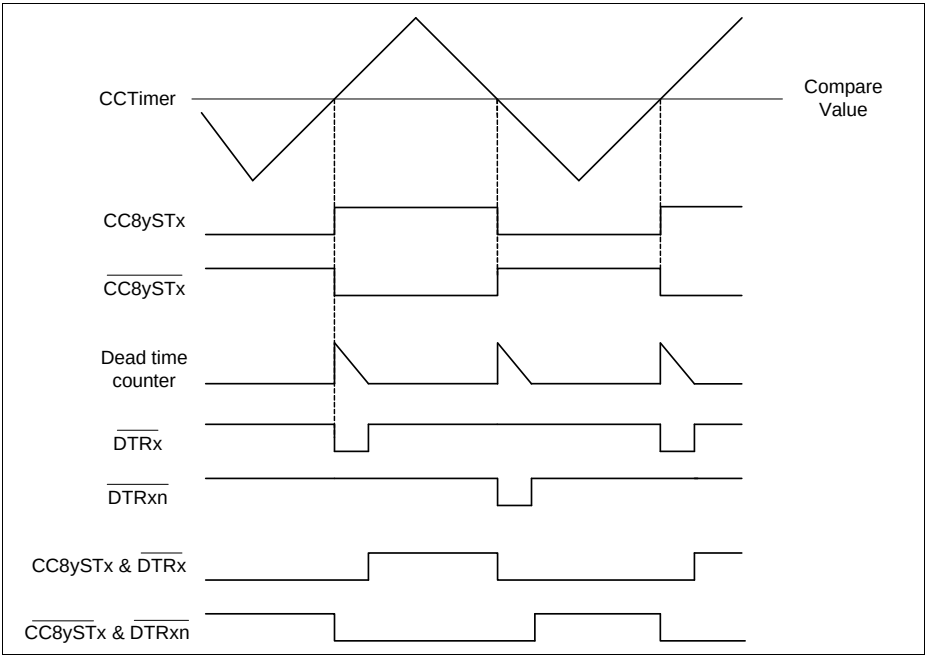



**Figure 20-29 Center aligned - Independent channel with dead time**

### Asymmetrical Center Aligned Mode

The asymmetric mode is enabled in center aligned by setting the field **CC8yCHC**.ASE to $1_B$.

In this mode, like in Edge Aligned, the outputs linked with the compare channel 2 are set to their passive levels.

The status bit, CC8yST1, is set when a compare match of channel 1 occurs while counting up, and is cleared when a compare match of channel 2 occurs while counting down, see **Figure 20-30**.

The dead time rise and fall times are controlled by the values programmed into the fields, **CC8yDC1R**.DT1R and **CC8yDC1R**.DT1F, respectively.

**Figure 20-31** shows the timing diagram for the asymmetric mode. Notice that even in asymmetric mode the dead time can be disabled in each of the outputs independently.

*Note: When an external signal is used to control the counting direction, the asymmetric mode cannot be enabled.*

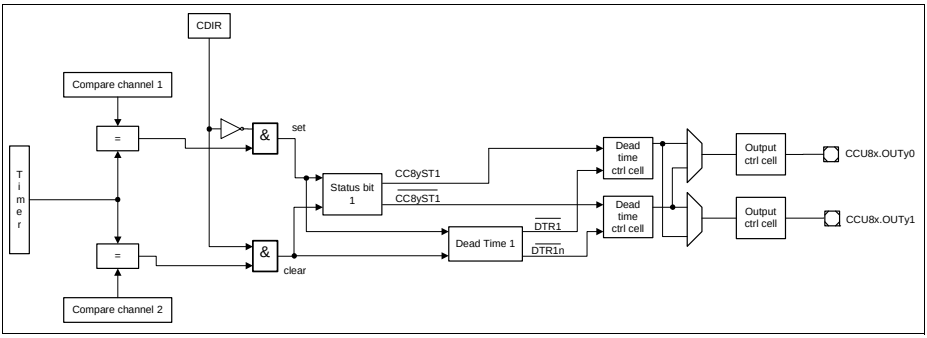



**Figure 20-30 Center Aligned Asymmetric mode scheme**

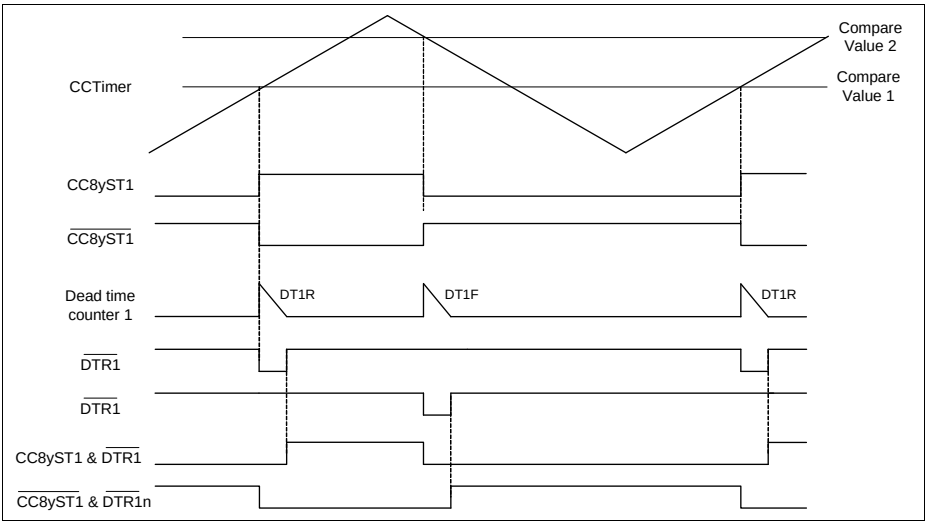



**Figure 20-31 Asymmetric Center aligned mode with dead time**

## 20.2.8 External Events Control

Each CCU8 slice has the possibility of using up to three different input events, see **Section 20.2.2**. These three events can then be mapped to Timer Slice functions (the full set of available functions is described at **Section 20.2.3**).

These events can be mapped to any of the CCU8x.INy[P...A] inputs and there isn't any imposition that an event cannot be used to perform several functions or, that an input cannot be mapped to several events (e.g. input X triggers event 0 with rising edge and triggers event 1 with the falling edge).

### 20.2.8.1 External Start/Stop

To select an external start function, one should map one of the events (output of the input selector) to a specific input signal, by setting the required value in the **CC8yINS**.EVxIS register and indicating the active edge of the signal on the **CC8yINS**.EVxEM register.

This event should be then mapped to the start or stop functionality by setting the **CC8yCMC**.STRTS (for the start) or the **CC8yTC**.ENDM (for the stop) with the proper value.

The same procedure is applicable to the stop functionality.

Notice that both start and stop functions are edge and not level active and therefore the active/passive configuration is set only by the **CC8yINS**.EVxEM.


Subject to Agreement on the Use of Product Information

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 20-32** to **Figure 20-35** shows the usage of two signals to perform the start/stop functions in all the previously mentioned subsets. External Signal(1) acts as an active HIGH start signal, while External Signal(2) is used as an active HIGH stop function.

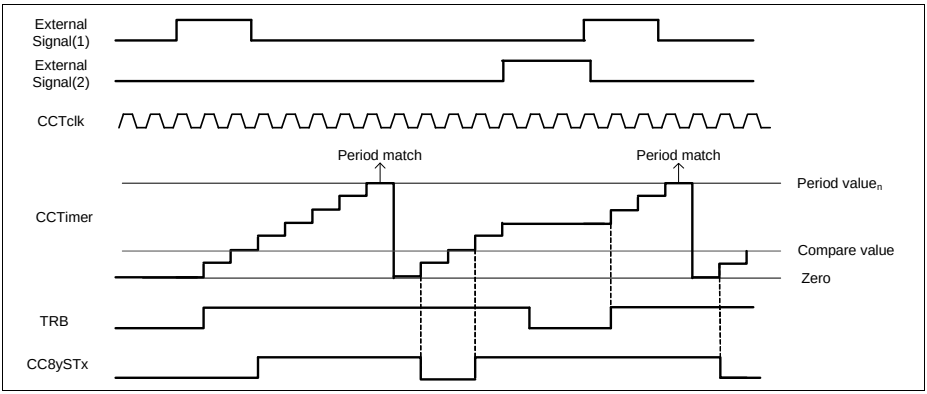



**Figure 20-32 Start (as start)/ stop (as stop) - CC8yTC.STRM = $0_B$, CC8yTC.ENDM = $00_B$**

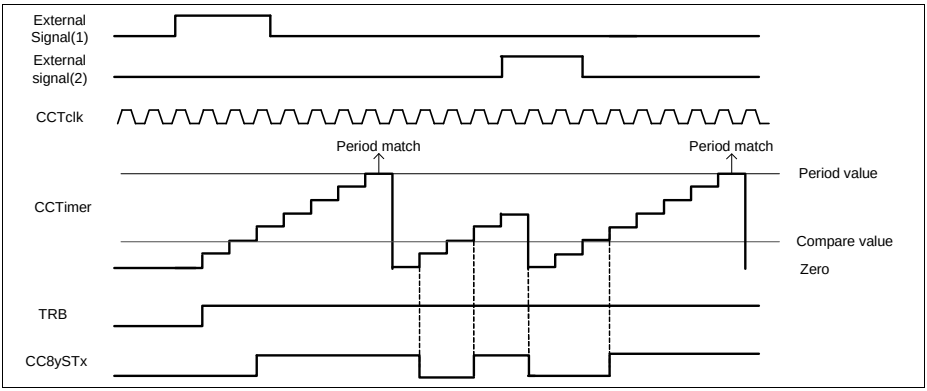



**Figure 20-33 Start (as start)/ stop (as flush) - CC8yTC.STRM = $0_B$, CC8yTC.ENDM = $01_B$**

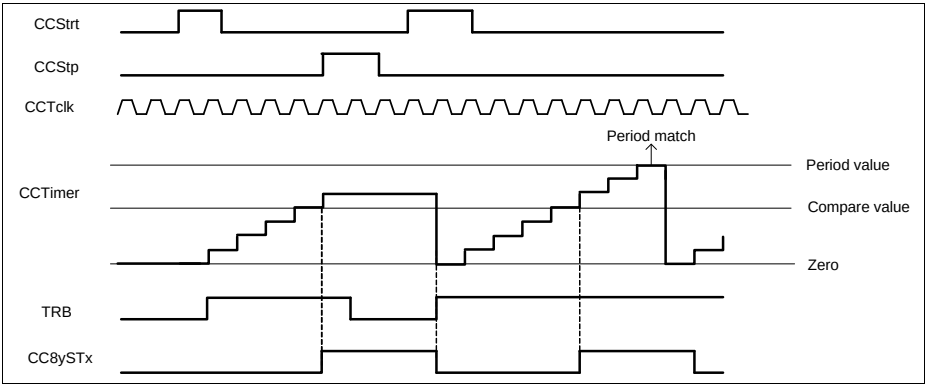



**Figure 20-34 Start (as flush and start)/ stop (as stop) - CC8yTC.STRM = $1_B$, CC8yTC.ENDM = $00_B$**

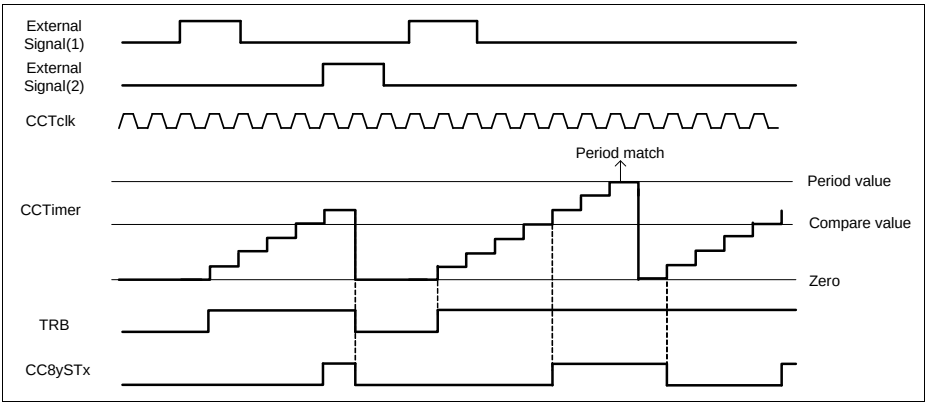



**Figure 20-35 Start (as start)/ stop (as flush and stop) - CC8yTC.STRM = $0_B$, CC8yTC.ENDM = $10_B$**

### 20.2.8.2 External Counting Direction

There is the possibility of selecting an input signal to act as counting up/counting down control.

To select an external up/down control, one should map one of the events (output of the input selector) to a specific input signal, by setting the required value in the **CC8yINS**.EVxIS register and indicating the active level of the signal on the **CC8yINS**.EVxLM register. This event should be then mapped to the up/down functionality by setting the **CC8yCMC**.UDS with the proper value.

Notice that the up/down function is level active and therefore the active/passive configuration is set only by the **CC8yINS**.EVxLM.

The status bit of the slice (CCSTx) is always set when the timer value is equal or greater than the value stored in the compare register, see **Section 20.2.6**.

The update of the period and compare register values is done when:

- with the next clock after a period match, while counting up (**CC8yTCST**.CDIR = $0_B$)
- with the next clock after a one match, while counting down (**CC8yTCST**.CDIR = $1_B$)

The value of the **CC8yTCST**.CDIR register is updated accordingly with the changes on the decoded event. The Up/Down direction is always understood as **CC8yTCST**.CDIR = 1 when counting down and **CC8yTCST**.CDIR = $0_B$ when counting up. Using an external signal to perform the up/down counting function and configuring the event as active HIGH means that the timer is counting up when the signal is HIGH and counting down when LOW.


Subject to Agreement on the Use of Product Information

**Figure 20-36** shows an external signal being used to control the counting direction of the time. This signal was selected as active HIGH, which means that the timer is counting down while the signal is HIGH and counting up when the signal is LOW.

*Note: For a signal that should impose an increment when LOW and a decrement when HIGH, the user needs to set the **CC8yINS**.EVxLM = $0_B$. When the operation is switched, then the user should set **CC8yINS**.EVxLM = $1_B$.*

*Note: Using an external counting direction control, sets the slice in edge aligned mode.*

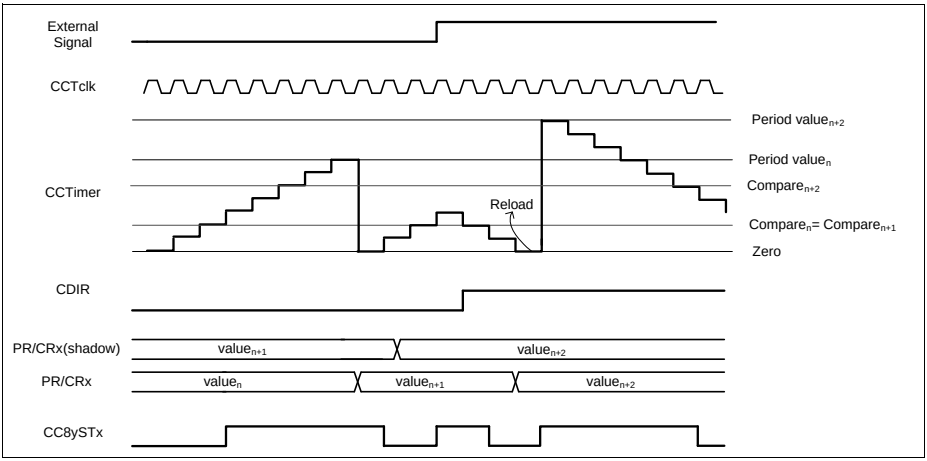



**Figure 20-36 External counting direction**

### 20.2.8.3 External Gating Signal

For pulse measurement, the user has the possibility of selecting an input signal that operates as counting gating.

To select an external gating control, one should map one of the events (output of the input selector) to a specific input signal, by setting the required value in the **CC8yINS**.EVxIS register and indicating the active level of the signal on the **CC8yINS**.EVxLM register. This event should be then mapped to the gating functionality by setting the **CC8yCMC**.GATES with the proper value.

Notice that the gating function is level active and therefore the active/passive configuration is set only by the **CC8yINS**.EVxLM.

The status bit during an external gating signal continues to be asserted when the compare value is reached and deasserted when the counter reaches $0000_H$. One should note that the counter continues to use the period register to identify a wrap around condition. **Figure 20-37** shows the usage of an external signal for gating the slice

counter. The signal was set as active LOW, which means the counter gating functionality is active when the external value is zero.

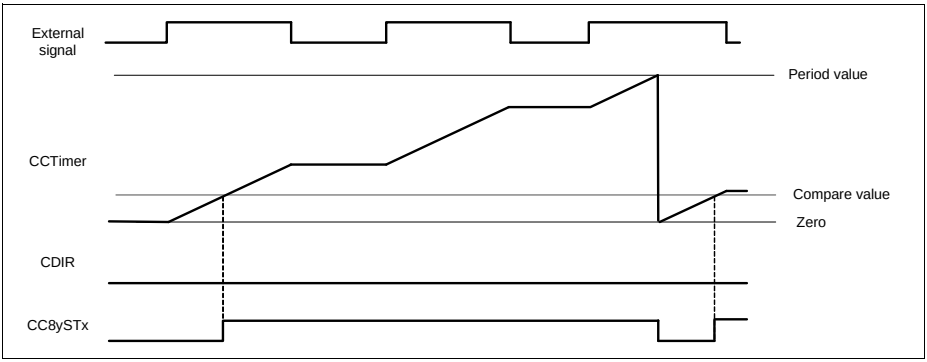



**Figure 20-37 External gating**

### 20.2.8.4 External Count Signal

There is also the possibility of selecting an external signal to act as the counting event.

To select an external counting, one should map one of the events (output of the input selector) to a specific input signal, by setting the required value in the **CC8yINS**.EVxIS register and indicating the active edge of the signal on the **CC8yINS**.EVxEM register. This event should be then mapped to the counting functionality by setting the **CC8yCMC**.CNTS with the proper value.

Notice that the counting function is edge active and therefore the active/passive configuration is set only by the **CC8yINS**.EVxEM.

One can select just a the rising, falling or both edges to perform a count. On **Figure 20-38**, the external signal was selected as a counter event for both falling and rising edges. Wrap around condition is still applied with a comparison with the period register.


Subject to Agreement on the Use of Product Information

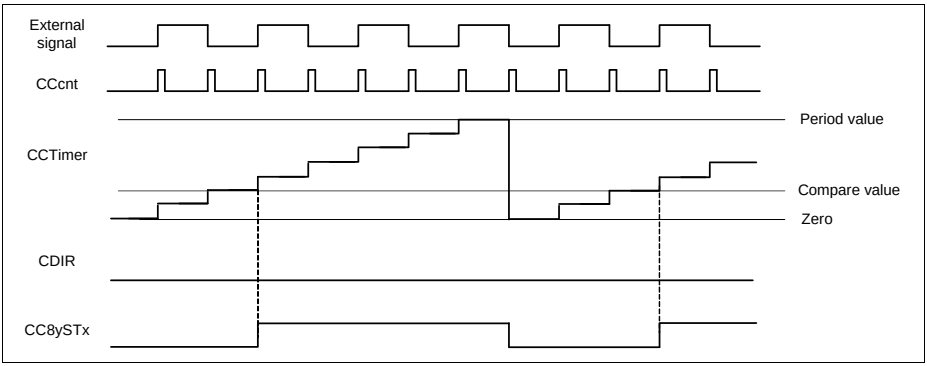



**Figure 20-38 External count**

### 20.2.8.5 External Load

Each slice of the CCU8 also has a functionality that enables the user to select an external signal as trigger for reloading the value of the timer with the current value of one compare register (if **CC8yTCST**.CDIR = $0_B$) or with the value of the period register (if **CC8yTCST**.CDIR = $1_B$).

The timer can be reloaded with the value from the compare channel 1 or compare channel 2 depending on the value set in the **CC8yTC**.TLS field, see **Figure 20-39**.

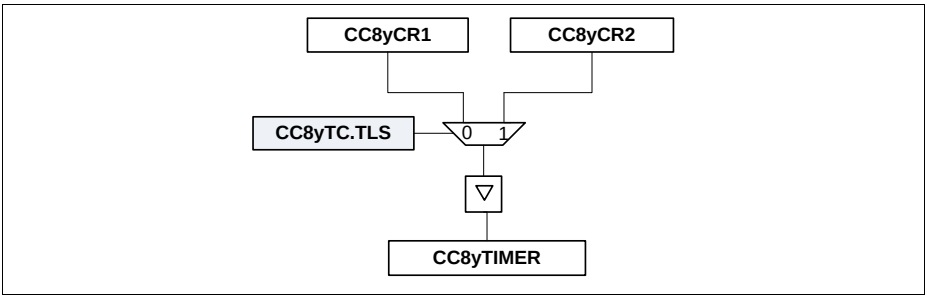



**Figure 20-39 Timer load selection**

To select an external load signal, one should map one of the events (output of the input selector) to a specific input signal, by setting the required value in the **CC8yINS**.EVxIS register and indicating the active edge of the signal on the **CC8yINS**.EVxEM register. This event should be then mapped to the load functionality by setting the **CC8yCMC**.LDS with the proper value.

Notice that the load function is edge active and therefore the active/passive configuration is set only by the **CC8yINS**.EVxEM.


Subject to Agreement on the Use of Product Information

On figure **Figure 20-40**, the external signal (1) was used to act as a load trigger, active on the rising edge. Every time that a rising edge on external signal (1) is detected, the timer value is loaded with the value present on the compare register. If an external signal is being used to control the counting direction, up or down, the timer value can be loaded also with the value set in the period register. The External signal (2) represents the counting direction control (active HIGH). If at the moment that a load trigger is detected, the signal controlling the counting direction is imposing a decrement, then the value set in the timer is the period value.

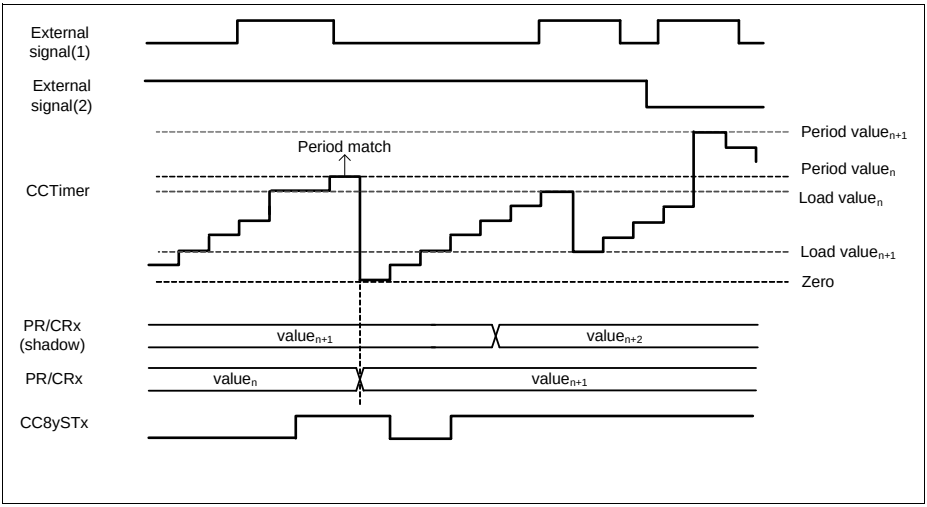



**Figure 20-40 External load**

### 20.2.8.6 External Capture

When selecting an external signal to be used as a capture trigger (if **CC8yCMC**.CAP0S or **CC8yCMC**.CAP1S are different from $0_H$), the user is automatically setting the specific slice into capture mode.

In capture mode the user can have up to four capture registers, see **Figure 20-43**: capture register 0 (**CC8yC0V**), capture register 1 (**CC8yC1V**), capture register 2 (**CC8yC2V**) and capture register 3 (**CC8yC3V**).

These registers are shared between compare and capture modes, which imposes:

- if **CC8yC0V** and **CC8yC1V** are used for capturing, the compare registers **CC8yCR1** and **CC8yCR1S** are not available (compare channel 1 is not available)
- if **CC8yC2V** and **CC8yC3V** are used for capturing, the compare registers **CC8yCR2** and **CC8yCR2S** are not available (compare channel 2 is not available)


Subject to Agreement on the Use of Product Information

To select an external capture signal, one should map one of the events (output of the input selector) to a specific input signal, by setting the required value in the **CC8yINS**.EVxIS register and indicating the active edge of the signal on the **CC8yINS**.EVxEM register.

This event should be then mapped to the capture functionality by setting the **CC8yCMC**.CAP0S/**CC8yCMC**.CAP1S with the proper value.

Notice that the capture function is edge active and therefore the active/passive configuration is set only by the **CC8yINS**.EVxEM.

The user has the possibility of selecting the following capture schemes:

- Different capture events for **CC8yC0V**/**CC8yC1V** and **CC8yC2V**/**CC8yC3V**
- The same capture event for **CC8yC0V**/**CC8yC1V** and **CC8yC2V**/**CC8yC3V** with the same capture edge. For this capture scheme, only the CCcapt1 functionality needs to be programmed. To enable this scheme, the field **CC8yTC**.SCE needs to be set to $1_B$.

**Different Capture Events (SCE = $0_B$)**

Every time that a capture trigger 1 occurs, CCcapt1, the actual value of the timer is captured into the capture register 3 and the previous value stored in this register is transferred into capture register 2.

Every time that a capture trigger 0 occurs, CCcapt0, the actual value of the timer is captured into the capture register 1 and the previous value stored in this register is transferred into capture register 0.

Every time that a capture procedure into one of the registers occurs, the respective full flag is set. This flag is cleared automatically by HW when the SW reads back the value of the capture register (by reading the specific capture register or by reading the extended capture read value, see **Section 20.2.8.7**).

The capture of a new value into a specific capture registers is dictated by the status of the full flag as follows:

$$CC8yC1V_{capt} = \text{NOT}(CC8yC1V_{full_flag} \text{ AND } CC8yC0V_{full_flag}) \quad (20.4)$$

$$CC8yC0V_{capt} = CC8yC1V_{full_flag} \text{ AND NOT}(CC8yC0V_{full_flag}) \quad (20.5)$$

It is also possible to disable the effect of the full flags by setting the **CC8yTC**.CCS = $1_B$. This enables a continuous capturing independent if the values captured have been read or not.

On **Figure 20-41**, an external signal was selected as an event for capturing the timer value into the **CC8yC0V**/**CC8yC1V** registers. The status bit, CC8ySTx, during capture mode is asserted whenever a capture trigger is detected and de asserted when the counter matches $0000_H$.


Subject to Agreement on the Use of Product Information

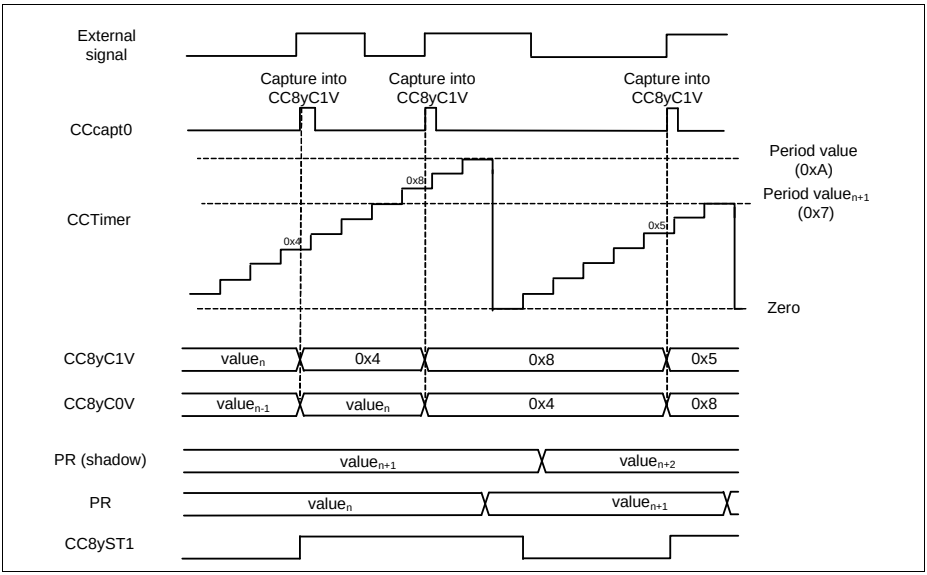



**Figure 20-41 External capture - CC8yCMC.CAP0S != $00_B$, CC8yCMC.CAP1S = $00_B$**

On **Figure 20-42**, two different signals were used as trigger sources for capturing the timer value into the **CC8yC0V**/**CC8yC1V** and **CC8yC2V**/**CC8yC3V** registers. External signal(1) was selected as an rising edge active source for the channel 1 capture trigger. External signal(2) was selected has the source for the channel 2 capture trigger, but as opposite to the external signal(1), the active edge was set as falling.

See **Section 20.2.14.4** for the complete capture mode usage description.


Subject to Agreement on the Use of Product Information

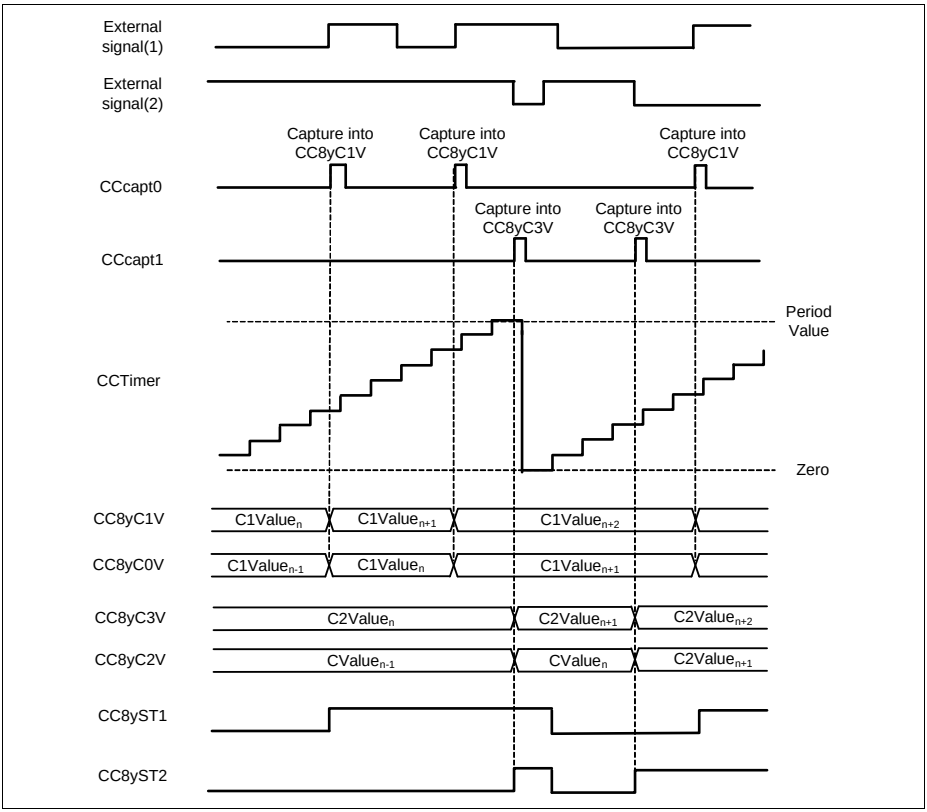



**Figure 20-42 External capture - CC8yCMC.CAP0S != $00_B$, CC8yCMC.CAP1S != $00_B$**


Subject to Agreement on the Use of Product Information

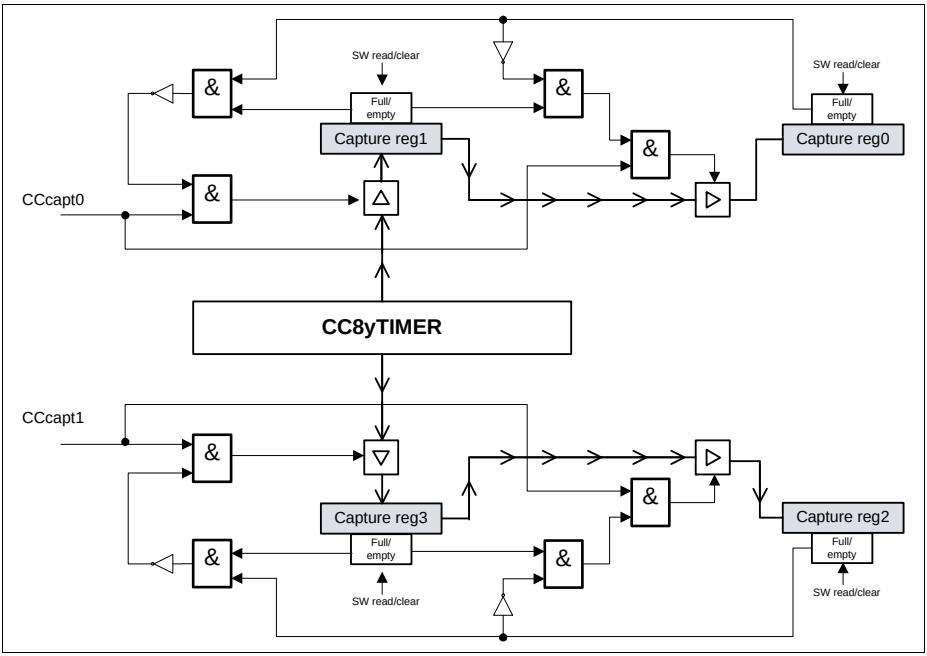



**Figure 20-43 Slice capture logic**

### Same Capture Event (SCE = $1_B$)

Setting the field **CC8yTC**.SCE = $1_B$, enables the possibility of having 4 capture registers linked with the same capture event, **Figure 20-45**.The functionality that controls the capture is the CCcapt1.

The capture logic follows the same structure shown in **Figure 20-43** but extended to a four register chain, see **Figure 20-44**.The same full flag lock rules are applied to the four register chain (it also can be disabled by setting the **CC8yTC**.CCS = $1_B$):

$$CC8yC3V_{capt} = \text{NOT}(CC8yC3V_{full_flag} \text{ AND } CC8yC2V_{full_flag} \text{ AND } CC8yC2V_{full_flag} \text{ AND } CC8yC1V_{full_flag}) \quad (20.6)$$

$$CC8yC2V_{capt} = CC8yC3V_{full_flag} \text{ AND NOT}(CC8yC2V_{full_flag} \text{ AND } CC8yC1V_{full_flag} \text{ AND } CC8yC0V_{full_flag}) \quad (20.7)$$

$$CC8yC1V_{capt} = CC8yC2V_{full_flag} \text{ AND NOT}(CC8yC1V_{full_flag} \text{ AND } CC8yC0V_{full_flag}) \quad (20.8)$$

$$CC8yC0V_{capt} = CC8yC1V_{full_flag} \text{ AND NOT}(CC8yC0V_{full_flag}) \quad (20.9)$$

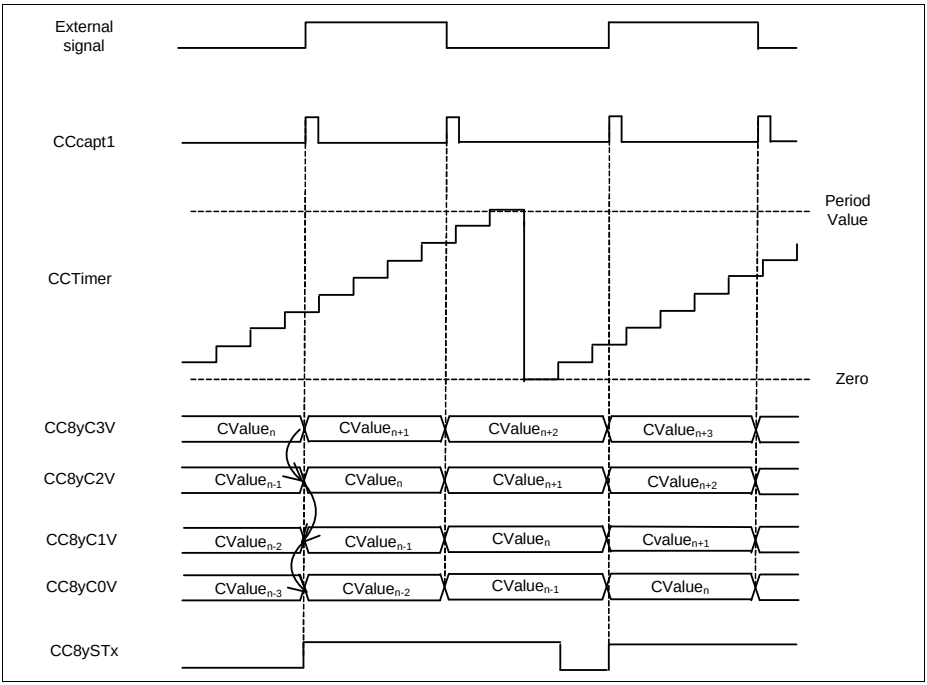



**Figure 20-44 External Capture - CC8yTC.SCE = $1_B$**

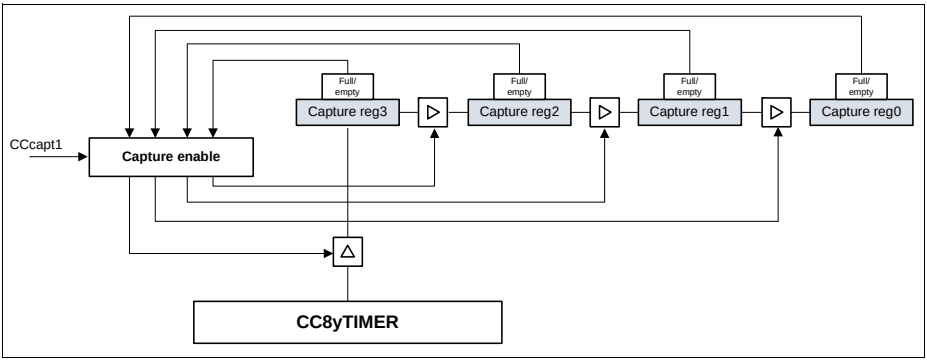



**Figure 20-45 Slice Capture Logic - CC8yTC.SCE = $1_B$**


Subject to Agreement on the Use of Product Information

### 20.2.8.7 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 cpature 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 20-46**.


Subject to Agreement on the Use of Product Information

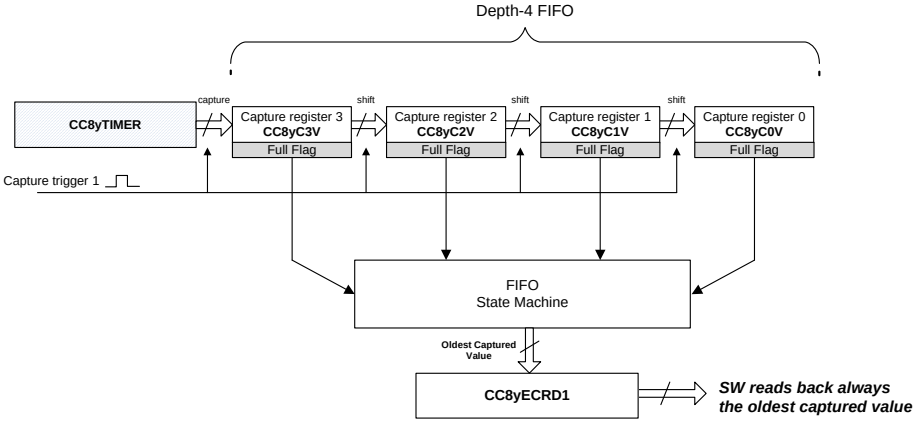



**Figure 20-46 Capture Extended Read Back - Depth 4**

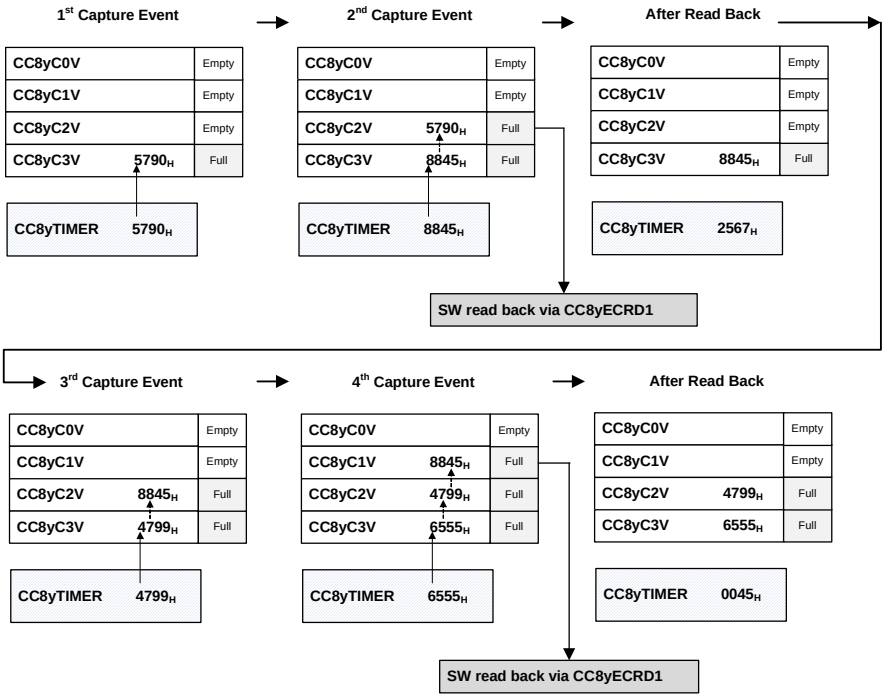



**Figure 20-47 Depth 4 software access example**

### 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 20-48**.


Subject to Agreement on the Use of Product Information

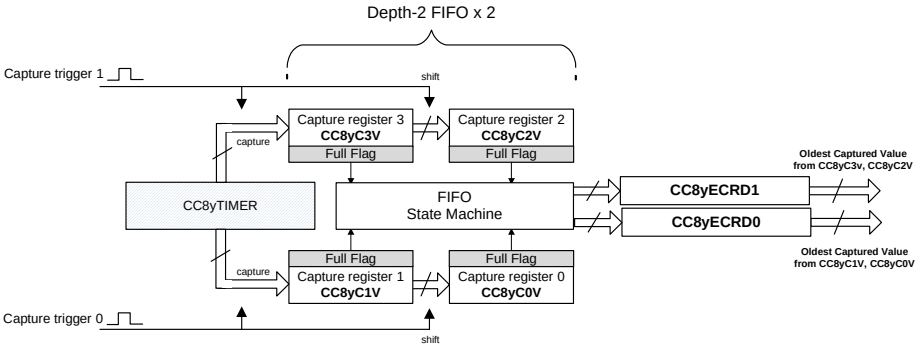



**Figure 20-48 Capture Extended Read Back - Depth 2**

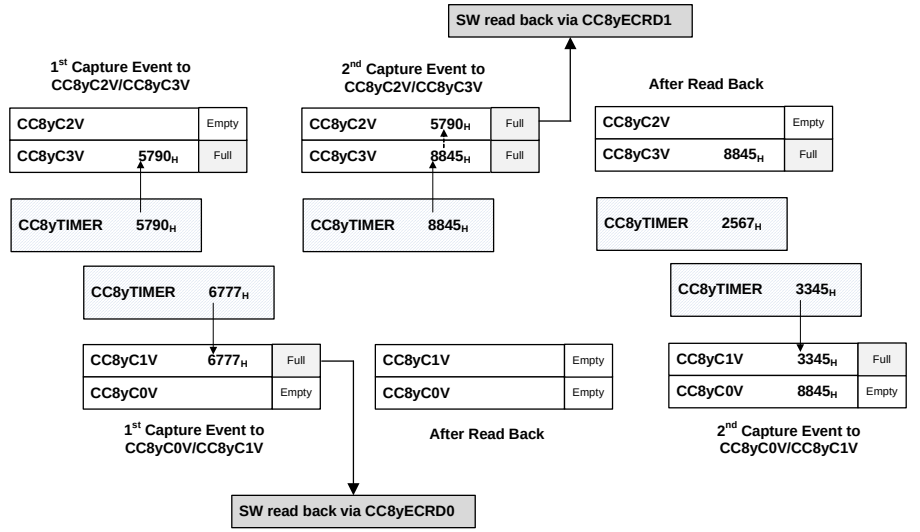



**Figure 20-49 Depth 2 software access example**

### 20.2.8.8 External Modulation

An external signal can be used also to perform a modulation at the output of each slice.

To select an external modulation signal, one should map one of the input signals to one of the events, by setting the required value in the **CC8yINS**.EVxIS register and indicating the active level of the signal on the **CC8yINS**.EVxLM register. This event should be then

mapped to the modulation functionality by setting the **CC8yCMC**.MOS = $01_B$ if event 0 is being used, **CC8yCMC**.MOS = $10_B$ if event 1 or **CC8yCMC**.MOS = $11_B$ if event 2.

Notice that the modulation function is level active and therefore the active/passive configuration is set only by the **CC8yINS**.EVxLM.

The external modulation signal can be applied to each compare channel independently, or it can be applied to both channels, by setting **CC8yTC**.EME = $11_B$.

The modulation has two modes of operation:

- modulation event is used to reset the CC8ySTx bit - **CC8yTC**.EMT = $0_B$
- modulation event is used to gate the outputs - **CC8yTC**.EMT = $1_B$

On **Figure 20-50**, we have a external signal configured to act as modulation source that clears the ST bit, **CC8yTC**.EMT = $0_B$. It was programmed to be an active LOW event and therefore, when this signal is LOW the output value is following the normal ACTIVE/PASSIVE rules.

When the signal is HIGH (inactive state), then the CC8ySTx bit is cleared and the output is forced into the PASSIVE state. Notice that the values of the status bit, CC8ySTx and the specific output CCU8x.OUTy are not linked together. One can choose for the output to be active LOW through the **CC8yPSL**.PSLx bit.

The exit of the external modulation inactive state is synchronized with the PWM period due to the fact that the CC8ySTx bit is cleared and cannot be set while the modulation signal is inactive.

The entering into inactive state also can be synchronized with the PWM period, by setting **CC8yTC**.EMS = $1_B$. With this all possible glitches at the output are avoided, see **Figure 20-51**.

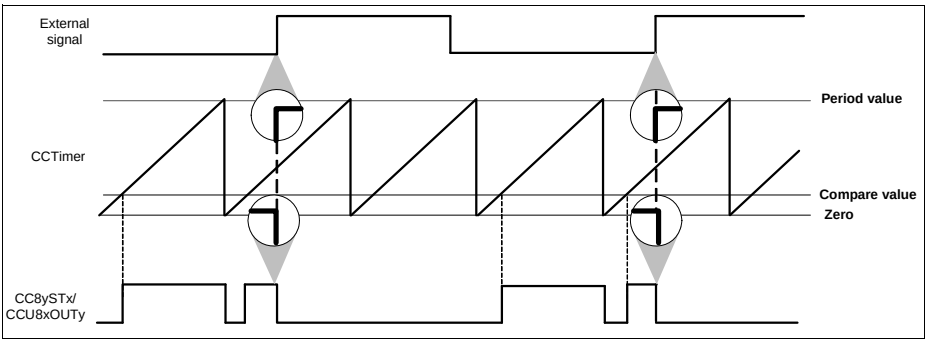



**Figure 20-50 External modulation resets the ST bit - CC8yTC.EMS = $0_B$**

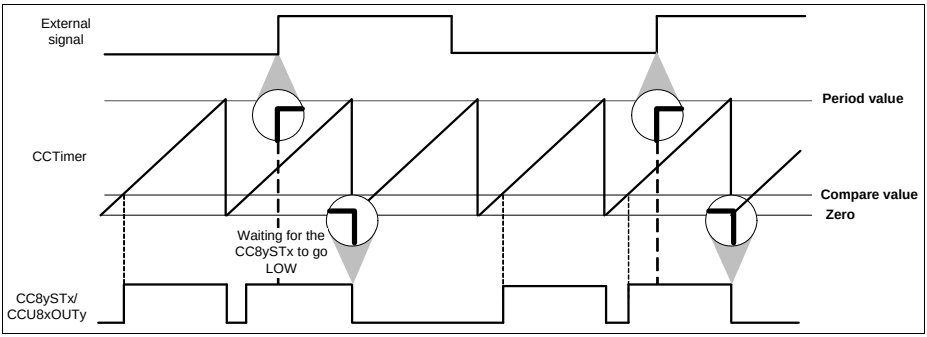



**Figure 20-51 External modulation clearing the ST bit - CC8yTC.EMS = $1_B$**

On **Figure 20-52**, the external modulation event was used as gating signal of the outputs, **CC8yTC**.EMT = $1_B$. The external signal was configured to be active HIGH, **CC8yINS**.EVxLM = $0_B$, which means that when the external signal is HIGH the outputs are set to the PASSIVE state.In this mode, the gating event can also be synchronized with the PWM signal by setting the **CC8yTC**.EMS = $1_B$.

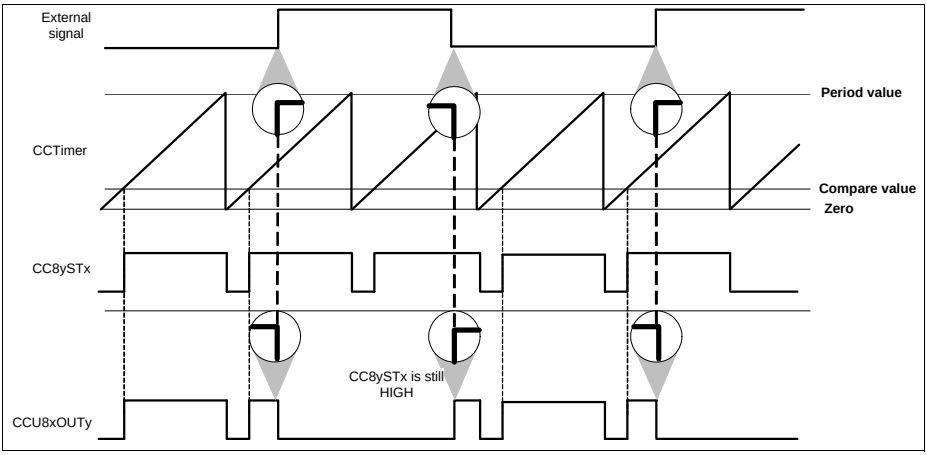



**Figure 20-52 External modulation gating the output - CC8yTC.EMT = $1_B$**

### 20.2.8.9 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.


Subject to Agreement on the Use of Product Information

To select the trap functionality, one should map one of the input signals to event number 2, by setting the required value in the **CC8yINS**.EV2IS register and indicating the active level of the signal on the **CC8yINS**.EV2LM register. This event should be then mapped to the trap functionality by setting the **CC8yCMC**.TS = $1_B$.

Notice that the trap function is level active and therefore the active/passive configuration is set only by the **CC8yINS**.EV2LM.

There are two bitfields that can be monitored via software to crosscheck the TRAP function, **Figure 20-53**:

- 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.

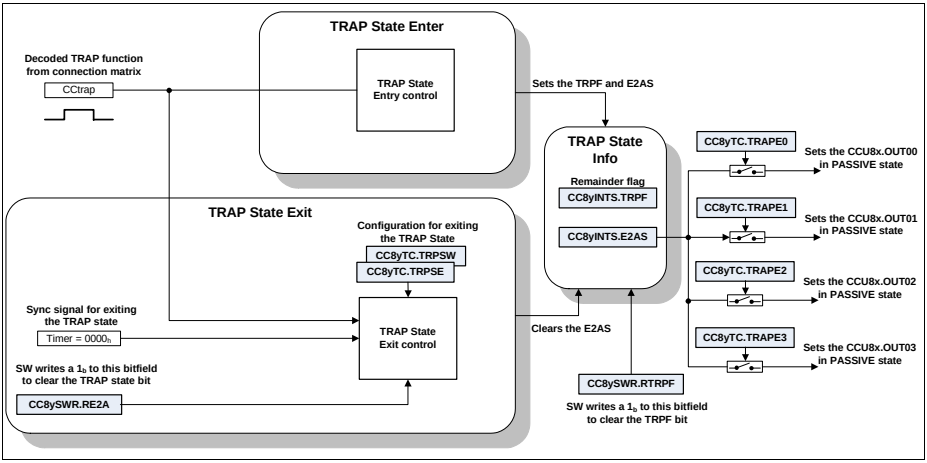



**Figure 20-53 Trap control diagram**

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 20-54**.

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 = $1_B$

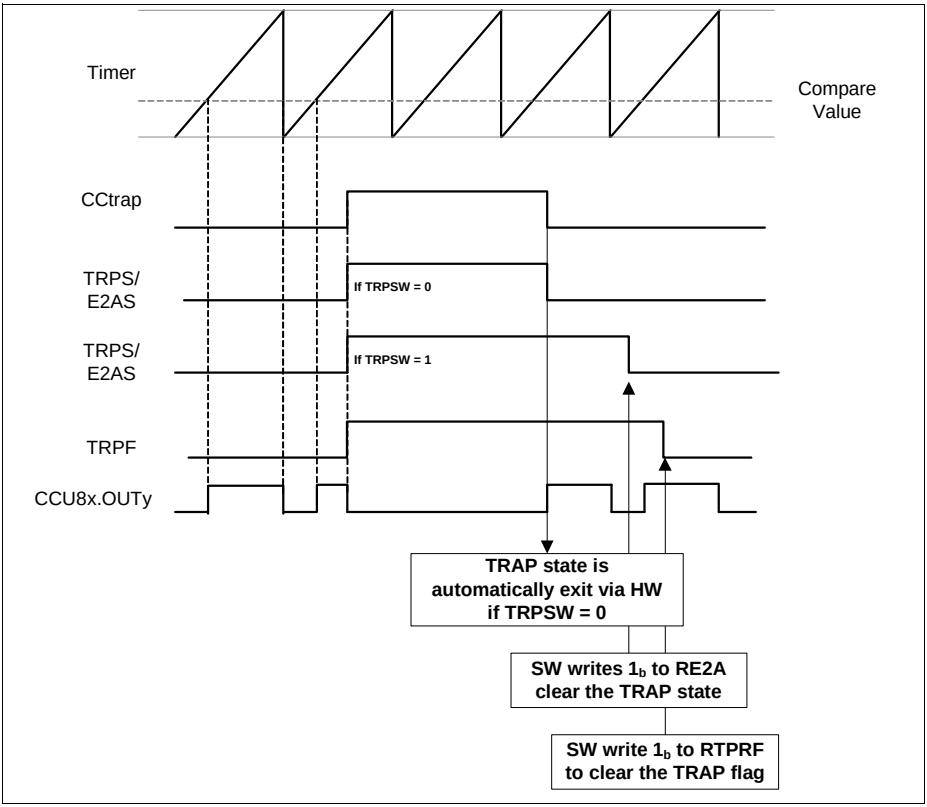



**Figure 20-54 Trap timing diagram, CC8yTCST.CDIR = 0 CC8yPSL.PSL = 0**

It is also possible to synchronize the exiting of the TRAP state with the PWM signal, **Figure 20-55**. This function is enabled when the bitfield **CC8yTC**.TRPSE = $1_B$.


Subject to Agreement on the Use of Product Information

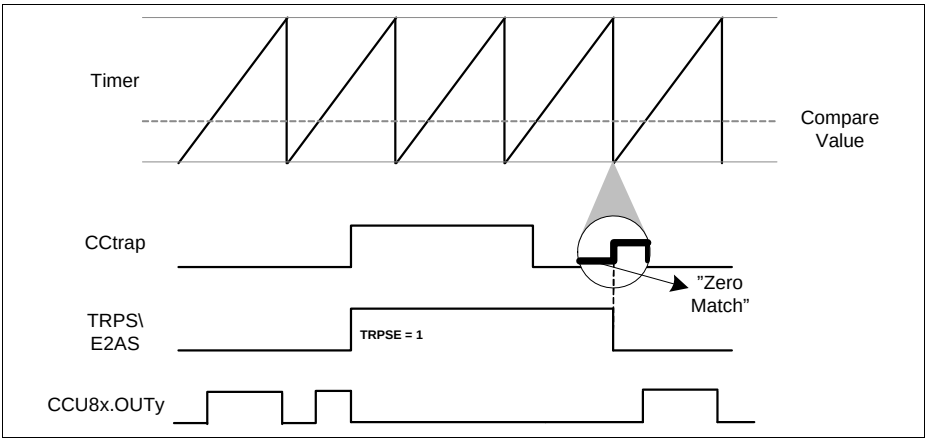



**Figure 20-55 Trap synchronization with the PWM signal**

### 20.2.8.10 Status Bit Override

For complex timed output control, each slice has a functionality that enables the override of the status bit of compare channel 1 (CC8yST1) with a value passed trough an external signal.

The override of the status bit, can then lead to a change on the output pins CCU8x.OUTy0 and CCU8x.OUTy1 (from inactive to active or vice versa).

To enable this functionality, two signals are needed:

- One signal that acts as a trigger to override the status bit (edge active)
- One signal that contains the value to be set in the status bit (level active)

To select the status bit override functionality, one should map the signal that acts as trigger to the event number 1, by setting the required value in the **CC8yINS**.EV1IS register and indicating the active edge of the signal on the **CC8yINS**.EV1EM register.

The signal that carries the value to be set on the status bit, needs to be mapped to the event number 2, by setting the required value in the **CC8yINS**.EV2IS register. The **CC8yINS**.EV2LM register should be set to $0_B$ if no inversion on the signal is needed and to $1_B$ otherwise.

The events should be then mapped to the status bit functionality by setting the **CC8yCMC**.OFS = $1_B$.

**Figure 20-56** 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.


Subject to Agreement on the Use of Product Information

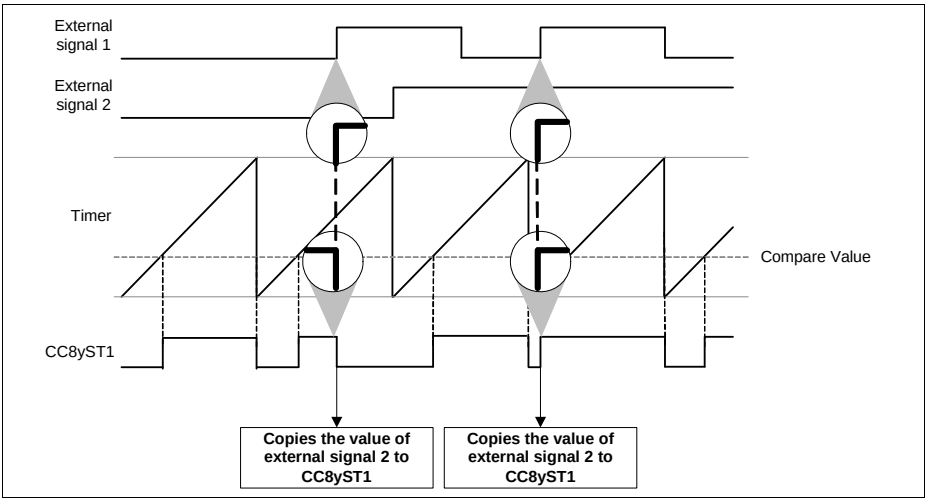



**Figure 20-56 Status bit override**

## 20.2.9 Multi-Channel Support

The multi channel control mode is selected individually in each slice by setting the **CC8yTC**.MCMEx = $1_B$.

With this mode, the output state of the Timer Slices PWM signal(s) (the ones set in multichannel mode) can be controlled in parallel by a single pattern.

The pattern is controlled via the CCU8 inputs, CCU8x.MCIy[3:0]. Each group of these inputs is connected accordingly to the specific Timer Slice: for slice 0, CCU8xMCI1[3:0] for slice 1, CCU8xMCI2[3:0] for slice 2 and CCU8xMCI3[3:0] for slice 3.

This pattern can be controlled directly by one of the POSIF modules and be updated in parallel for all the Timer Slices.

Using the POSIF module in conjunction with the Multi Channel support of the CCU8, one can achieve a complete synchronicity between the output state update, CCU8x.OUTy and the update of a new pattern, **Figure 20-57**.

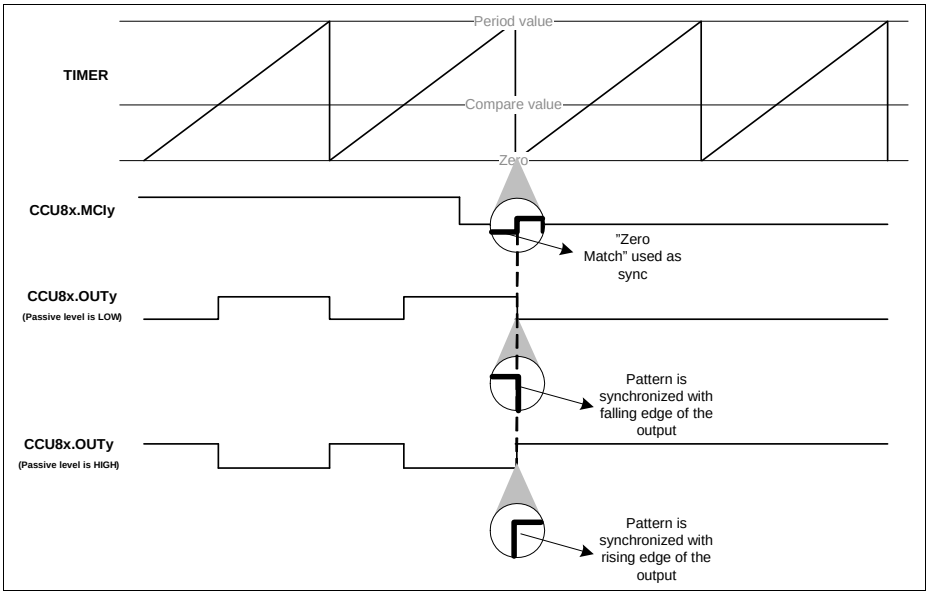



**Figure 20-57 Multi channel pattern synchronization**

These pattern inputs are going to be used in the output modulation control unit to put the specific PWM output into active or passive state: CCU8x.MCIy[0] has effect on the CC8yST1 path and therefore controls the CC8xOUT00 pin, CCU8x.MCy[1] is used in the same manner for the inverted CC8yST1 path, CCU8x.MCIy[2] and CCU8x.MCIy[3] are linked to the CC8yST2 and inverted CC8yST2 path respectively. **Figure 20-58** shows the simplified scheme for the multi channel control.

**Figure 20-59**, 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.

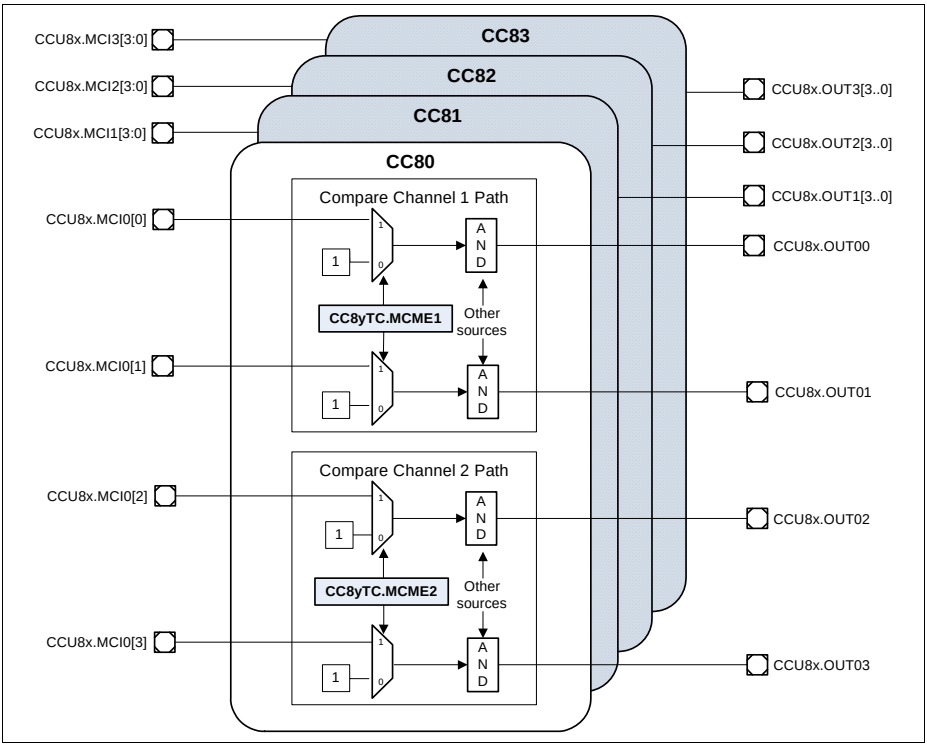



**Figure 20-58 CCU8 Multi Channel overview**


Subject to Agreement on the Use of Product Information

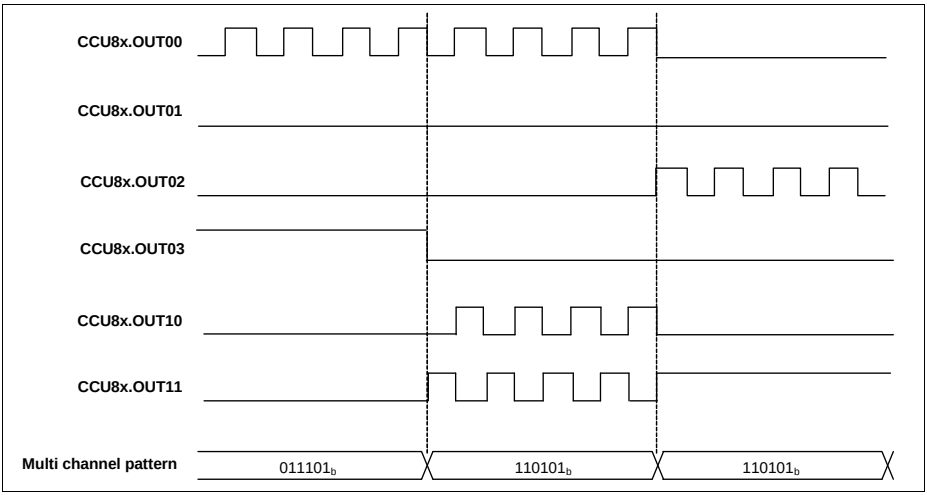



**Figure 20-59 Multi Channel mode for multiple Timer Slices**

The synchronization between the CCU8 and the POSIF is achieved, by adding a 3 cycle delay on the output path of each Timer Slice (between the status bit, CC8ySTx and the direct control of the output pin). This path is only selected when **CC8yTC**.MCMEx = $1_B$.

On **Figure 20-60** 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 20-61**. 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.


Subject to Agreement on the Use of Product Information

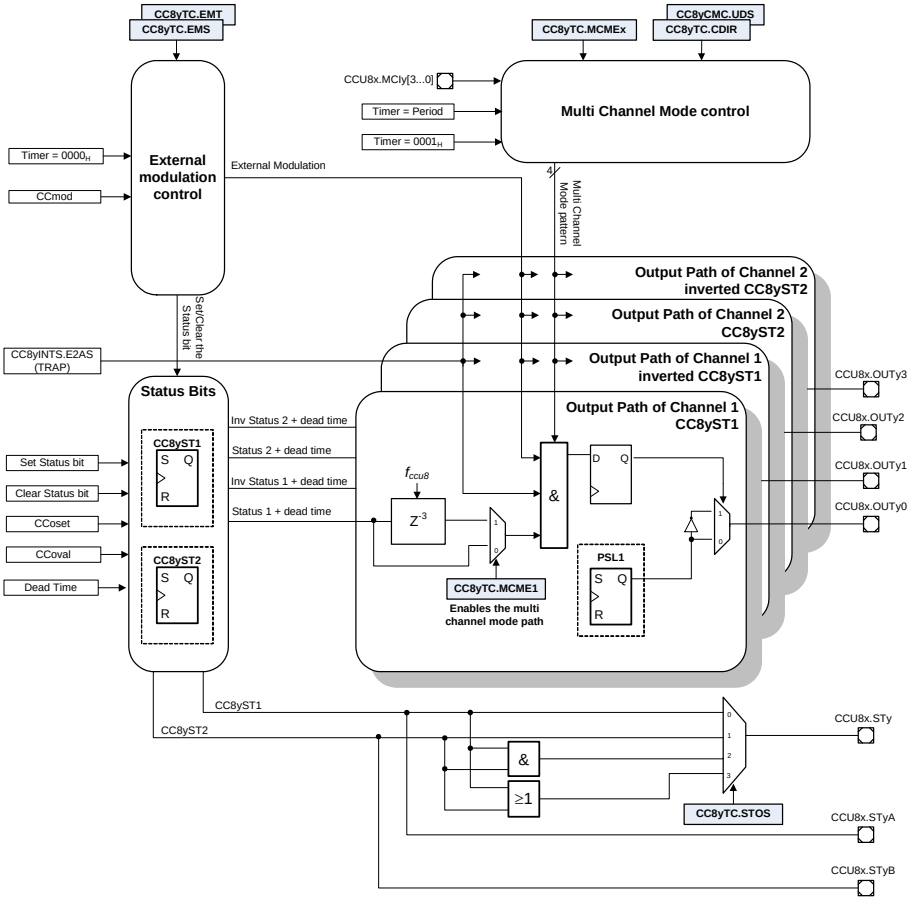



**Figure 20-60 Output Control Diagram**

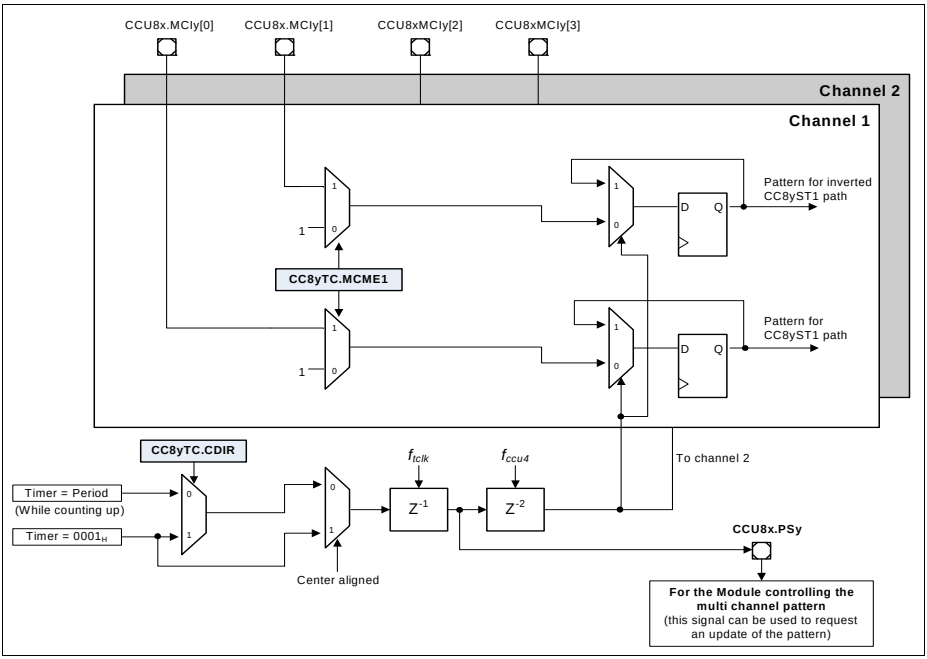



**Figure 20-61 Multi Channel Pattern Synchronization Control**

## 20.2.10 Timer Concatenation

The CCU8 offers a very easy mechanism to perform a synchronous timer concatenation. This functionality can be used by setting the **CC8yCMC**.TCE = $1_B$. By doing this the user is doing a concatenation of the actual CCU8 slice with the previous one, see **Figure 20-62**.

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

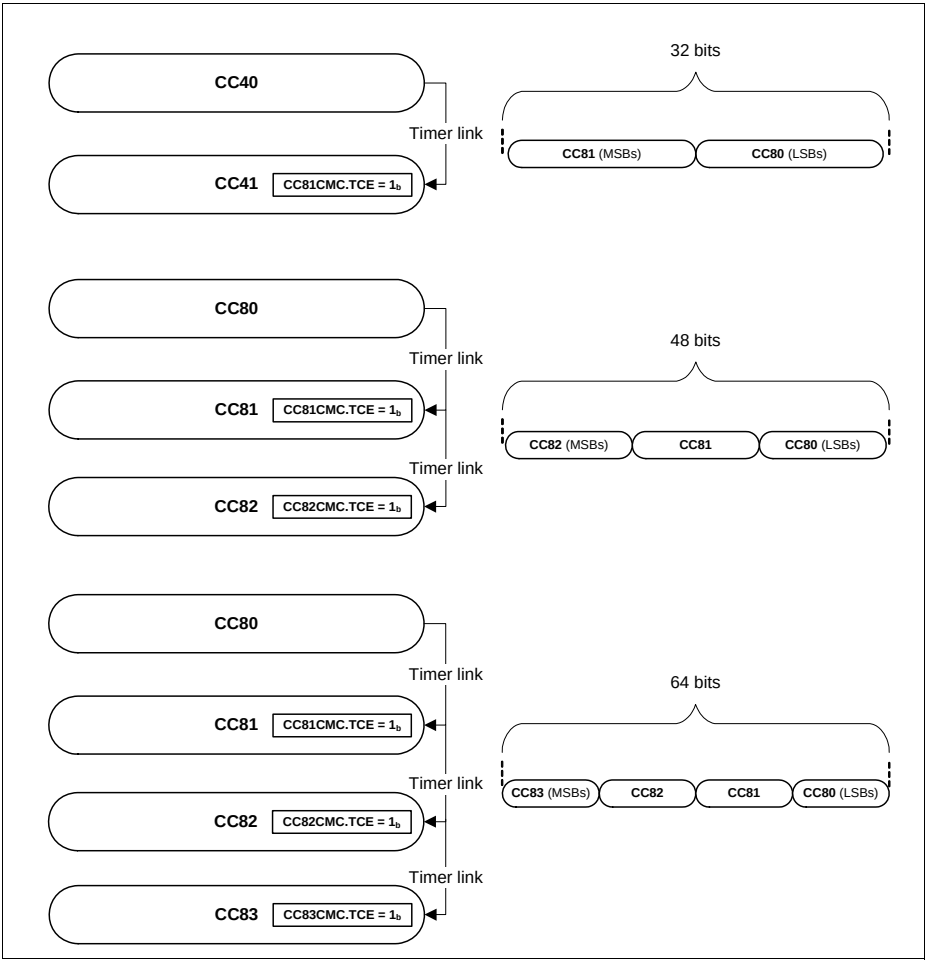



**Figure 20-62 Timer concatenation example**

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 20-63**.

*Note: For all the cases, CC80 and CC83 are not considered adjacent slices*

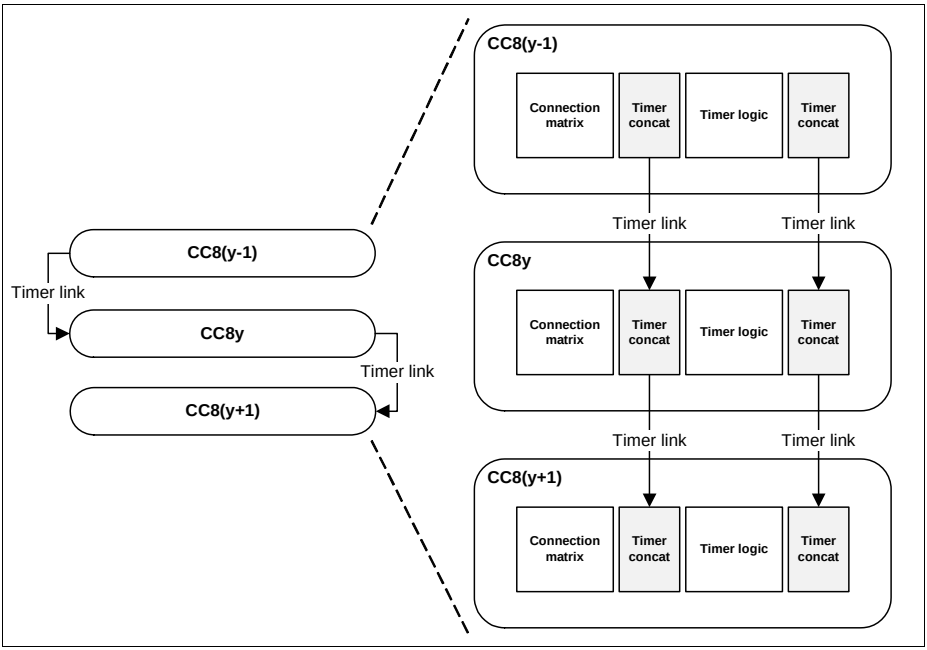



**Figure 20-63 Timer concatenation link**

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 20-63**. 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 20-64**.

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.

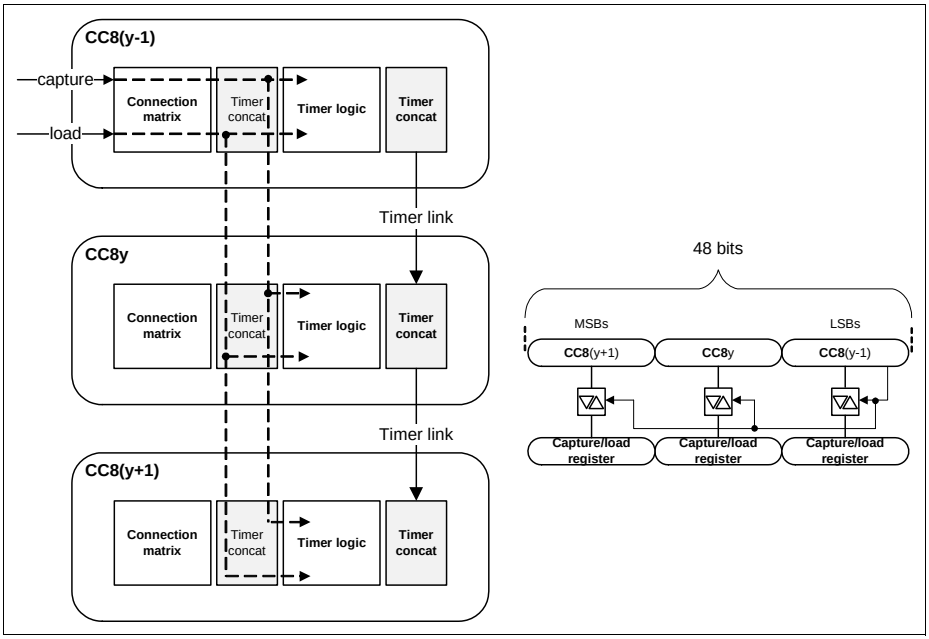



**Figure 20-64 Capture/Load Timer Concatenation**

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 20-65**.

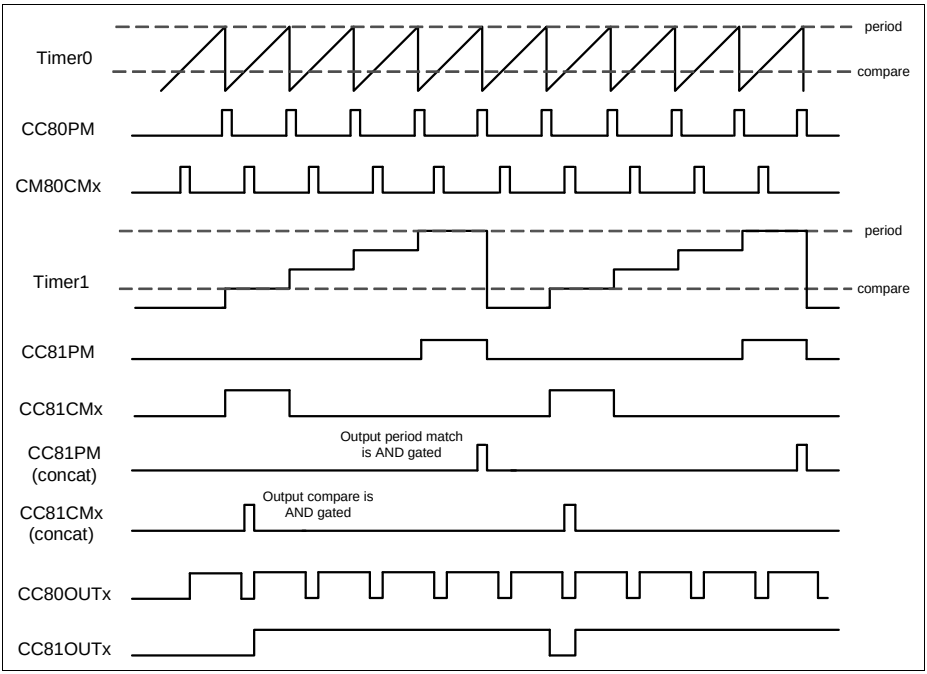



**Figure 20-65 32 bit concatenation timing diagram**

*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 20-66** gives an overview of the timer concatenation logic. Notice that all the mechanism is controlled solely by the **CC8yCMC**.TCE bitfield.

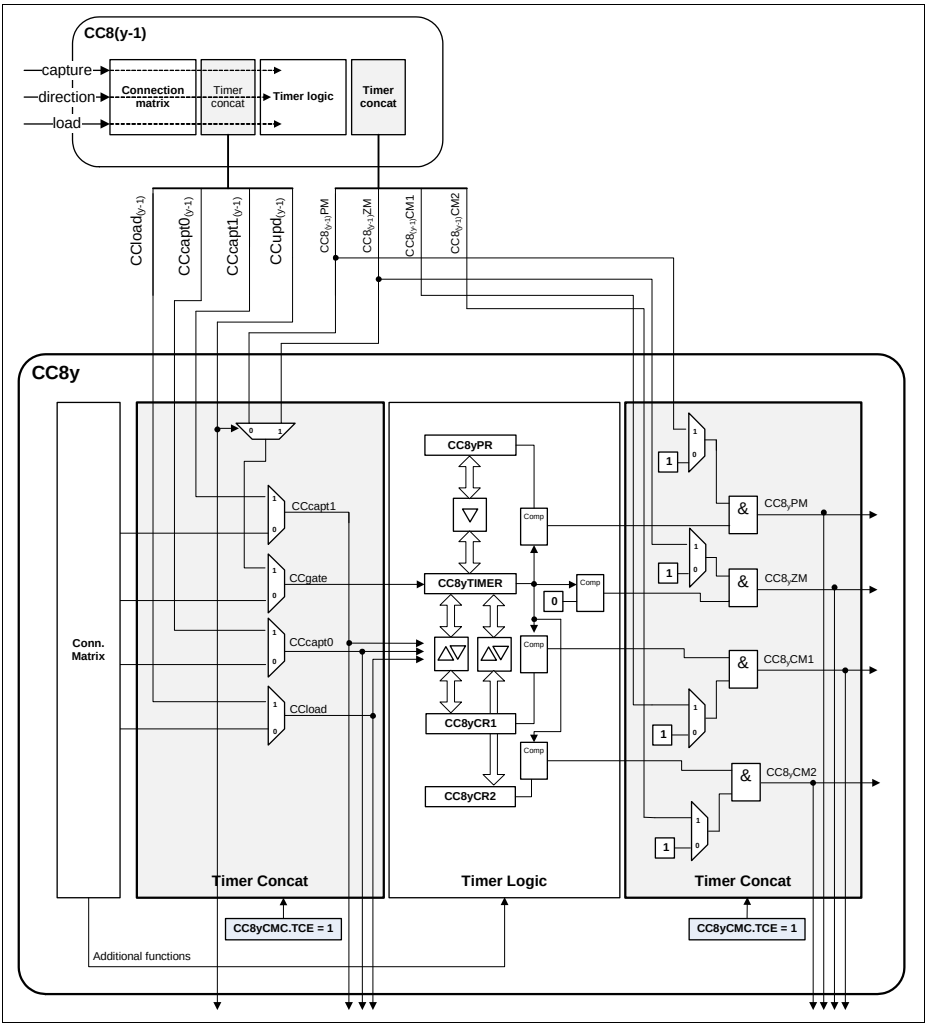



**Figure 20-66 Timer concatenation control logic**

## 20.2.11 Output Parity Checker

The parity checker function can be enabled by setting the **GIDLC**.PCH bit field to $1_B$ (parity checker is disabled while **GSTAT**.PCRB is $0_B$).

This parity checker function, crosschecks the value at the output of the CCU8 module versus an input signal that should be connected to a driver XOR structure.

It is also possible to add a delay between the switching of the outputs and the evaluation of the input signal coming from the driver structure, and select which type of parity, even or odd (via the **GPCHK**.PCTS bit field). **Figure 20-67** shows the structure of the parity checker unit.

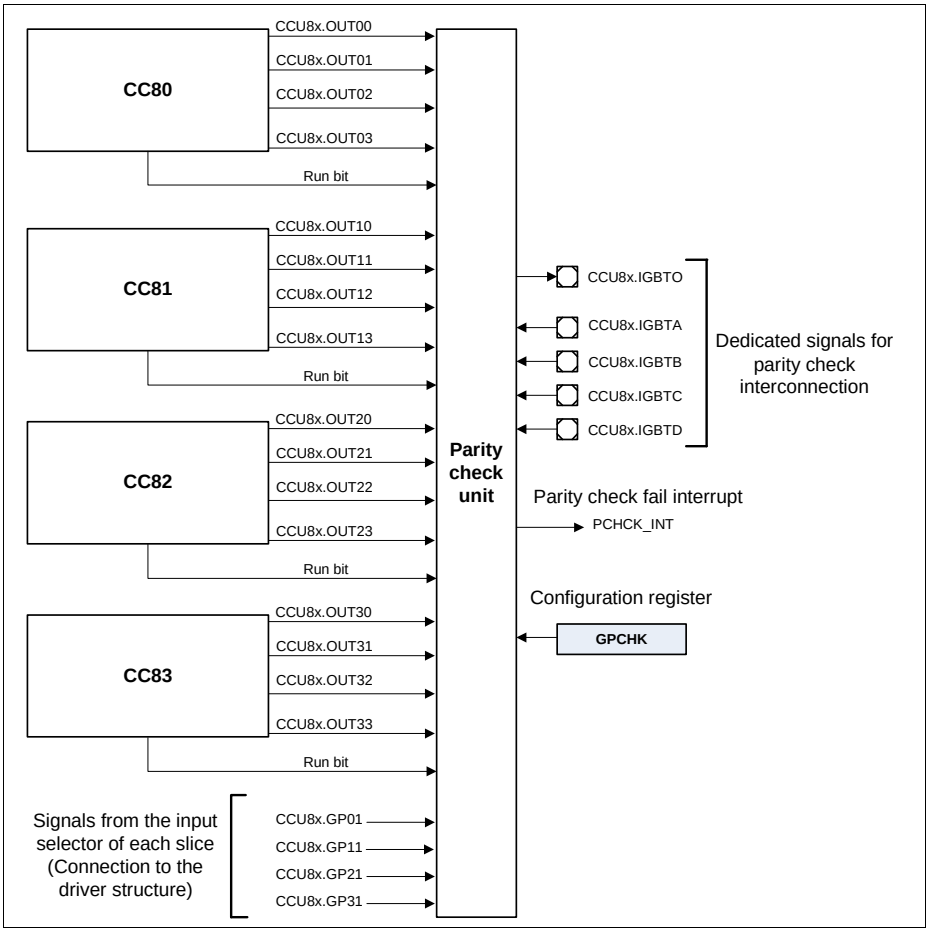



**Figure 20-67 Parity checker structure**

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 20-68**. For a more detailed description of resource usage for the parity checker function, please address **Section 20.2.14.5**.

Configuration Example:

Driver parity output is connected to the input CCU8x.IN1B (where x = CCU8 unit). The input used to control the switching delay is the CCU8x.IGBTCCCU8. The driver is using 12 outputs coming from the first three slices with an even parity (PCTS field is in default).

The following registers should then be programmed:

**CC8yINS**.EV1IS = $0001_B$; selects the input CCU8x.IN1B

**GPCHK**.PISEL = $01_B$; selects the Event 1 coming from slice 1

**GPCHK**.PCDS = $10_B$; selects the CCU8x.IGTBC input for delay control

**GPCHK**.PCSEL = $0FFF_H$; selects only the output signals of the first three slices for parity check

**GIDLC**.SPCH = $1_B$; starts the parity function

When a mismatch between the driver output and the parity checker is detected, an interrupt is generated on Timer Slice 1. The interrupt status bit that stores the information is **CC8yINTS**.E1AS.


Subject to Agreement on the Use of Product Information

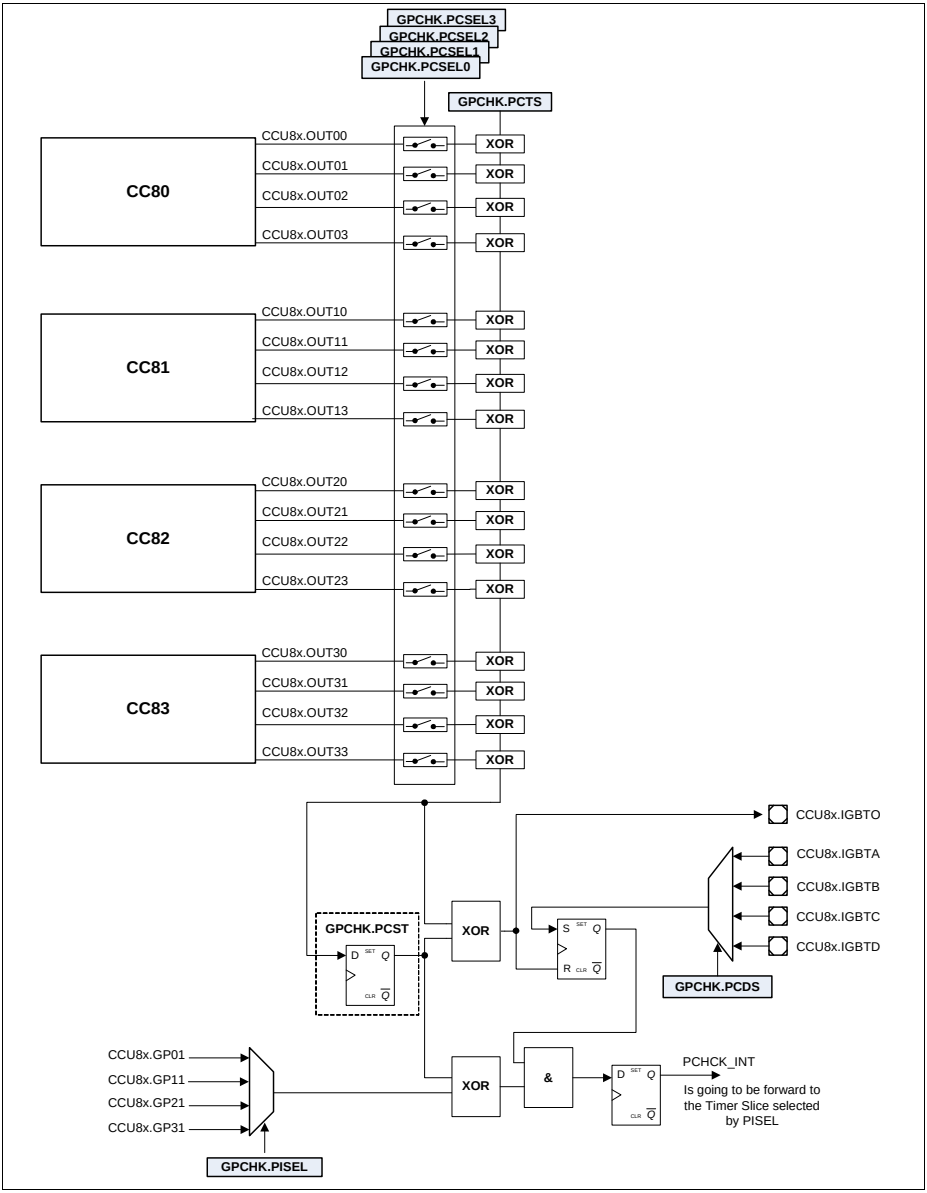



**Figure 20-68 Parity checker logic**

### 20.2.12 PWM Dithering

The CCU8 has an automatic PWM dithering insertion function. This functionality can be used with very slow control loops that cannot update the period/compare values in a fast manner, and by that fact the loop can lose precision on long runs. By introducing dither on the PWM signal, the average frequency/duty cycle is then compensated against that error.

Each slice contains a dither control unit, see **Figure 20-69**.

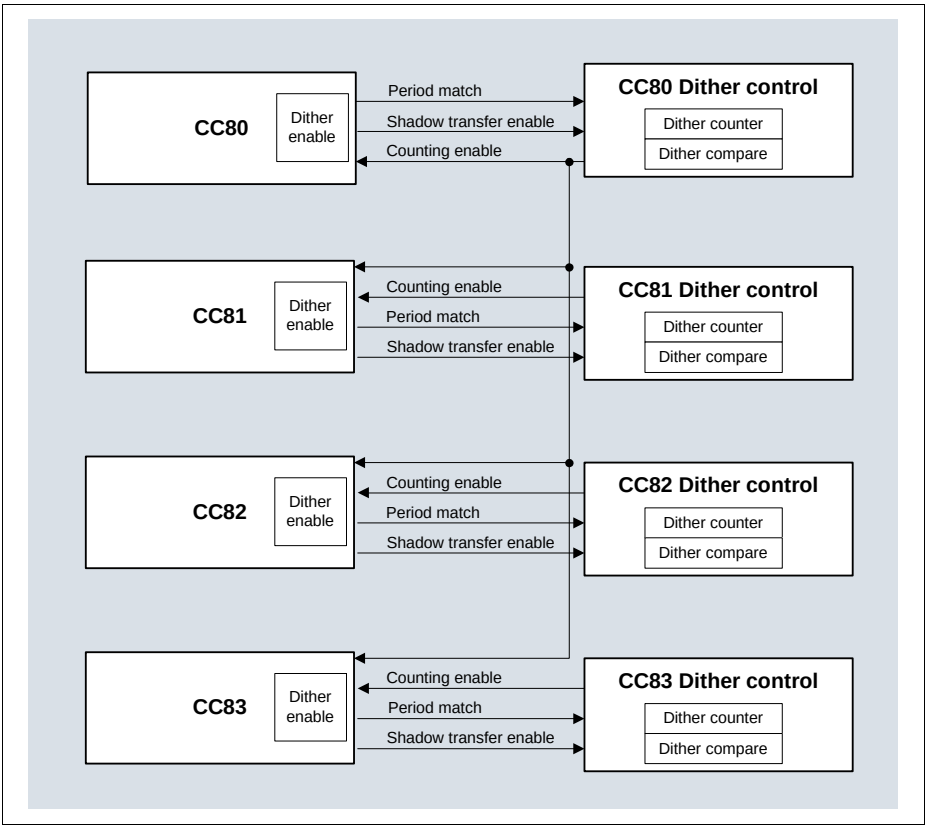



**Figure 20-69 Dither structure overview**

The dither control unit contains a 4 bit counter and a compare value. The four bit counter is incremented every time that a period match occurs. The counter works in a bit reverse mode so the distribution of increments stays uniform over 16 counter periods, see **Table 20-6**.


Subject to Agreement on the Use of Product Information

**Table 20-6 Dither bit reverse counter**

| counter[3] | counter[2] | counter[1] | counter[0] |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 |

The counter is then compared against a programmed value, **CC8yDIT**.DCV. If the counter value is smaller than the programmed value, a gating signal is generated that can be used to extend the period, to delay the compare or both (controlled by the **CC8yTC**.DITHE field, see **Table 20-7**) for one clock cycle.

**Table 20-7 Dither modes**

| DITHE[1] | DITH[0] | Mode |
|---|---|---|
| 0 | 0 | Dither is disabled |
| 0 | 1 | Period is increased by 1 cycle |
| 1 | 0 | Compare match is delayed by 1 cycle |
| 1 | 1 | Period is increased by 1 cycle and compare is delayed by 1 cycle |

The dither compare value also has an associated shadow register that enables concurrent update with the period/compare registers of each CC8y. The control logic for the dithering unit is represented on **Figure 20-70**.


Subject to Agreement on the Use of Product Information

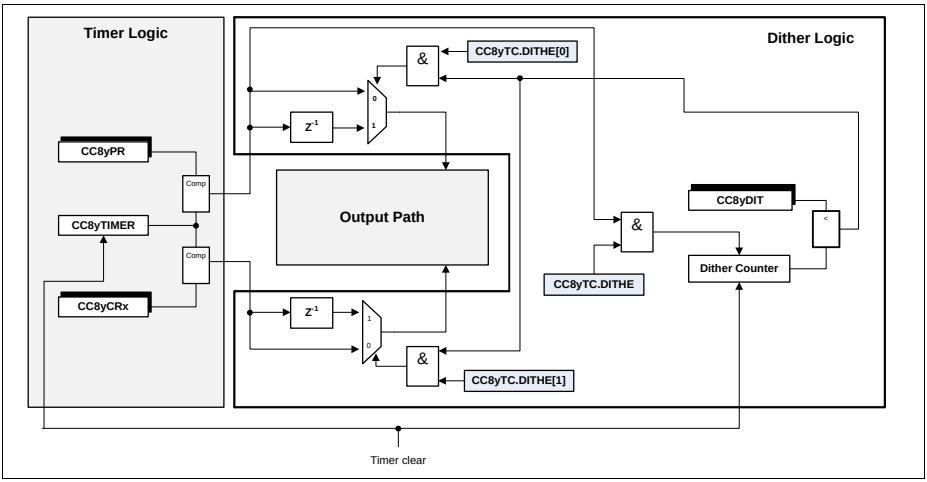



**Figure 20-70 Dither control logic**

**Figure 20-71** to **Figure 20-76** 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).

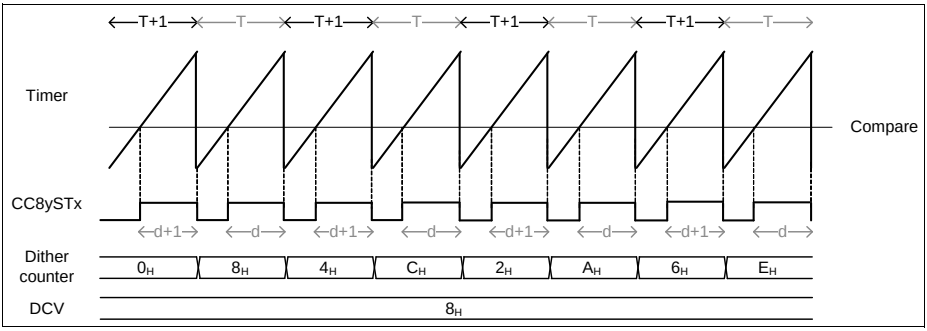



**Figure 20-71 Dither timing diagram in edge aligned - CC8yTC.DITHE = $01_B$**

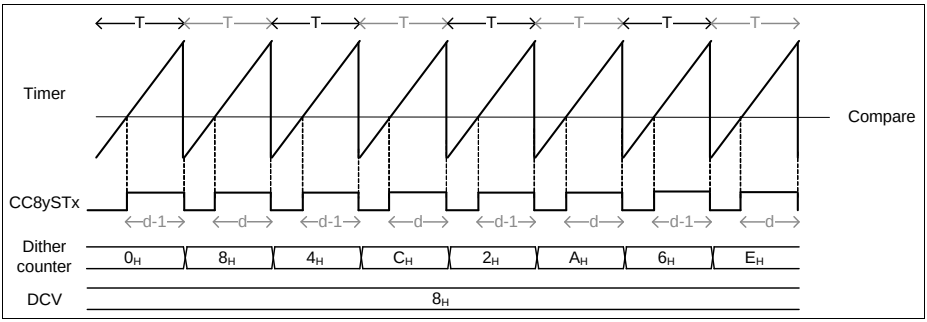



**Figure 20-72 Dither timing diagram in edge aligned - CC8yTC.DITHE = $10_B$**

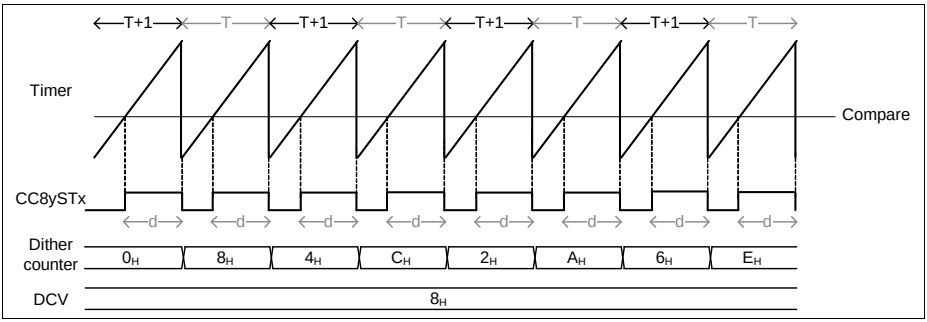



**Figure 20-73 Dither timing diagram in edge aligned - CC8yTC.DITHE = $11_B$**

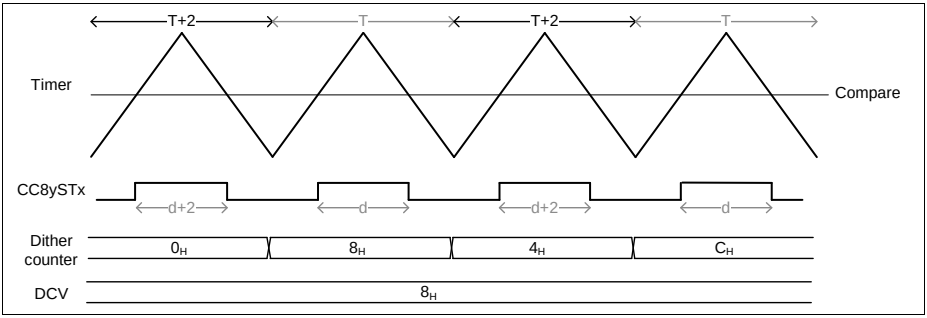



**Figure 20-74 Dither timing diagram in center aligned - CC8yTC.DITHE = $01_B$**

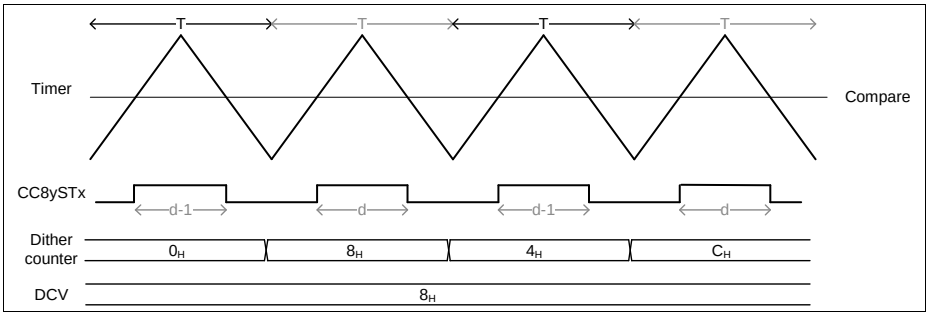



**Figure 20-75 Dither timing diagram in center aligned - CC8yTC.DITHE = $10_B$**

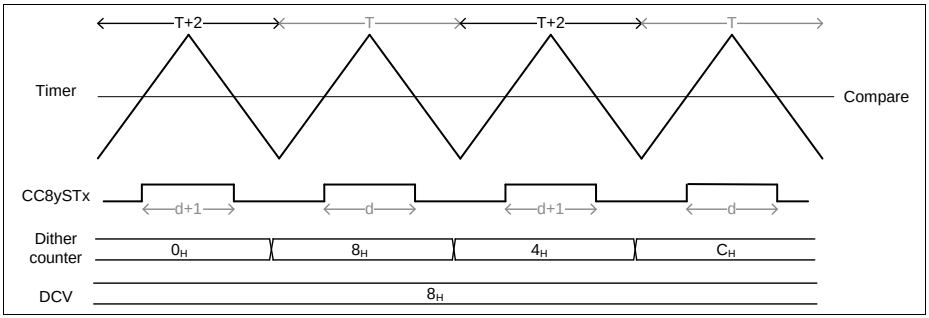



**Figure 20-76 Dither timing diagram in edge aligned - CC8yTC.DITHE = $11_B$**

*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.*

### 20.2.13 Prescaler

The CCU8 contains a 4 bit prescaler that can be used in two operating modes for each individual slice:

- normal prescaler mode
- floating prescaler mode

The run bit of the prescaler can be set/cleared by SW by writing into the registers, **GIDLC**.SPRB and **GIDLS**.CPRB respectively or can also be cleared by the run bit of a specific slice. With the last mechanism, the run bit of the prescaler is cleared one clock cycle after the clear of the run bit of the selected sliced. To select which slice can perform this action, one should program the **GCTRL**.PRBC register.


Subject to Agreement on the Use of Product Information

### 20.2.13.1 Normal Prescaler Mode

In Normal prescaler mode the clock fed to the CC8y counter is a normal fixed division by N, accordingly to the value set in the **CC8yPSC**.PSIV register. The values for the possible division values are listed in **Table 20-8**. The **CC8yPSC**.PSIV value is only modified by a SW access. Notice that each slice has a dedicated prescaler value selector (**CC8yPSC**.PSIV), which means that the user can select different counter clocks for every and each Timer Slice (CC8y).

**Table 20-8 Timer clock division options**

| **CC8yPSC.PSIV** | **Resulting clock** |
|---|---|
| $0000_B$ | $f_{ccu8}$ |
| $0001_B$ | $f_{ccu8}/2$ |
| $0010_B$ | $f_{ccu8}/4$ |
| $0011_B$ | $f_{ccu8}/8$ |
| $0100_B$ | $f_{ccu8}/16$ |
| $0101_B$ | $f_{ccu8}/32$ |
| $0110_B$ | $f_{ccu8}/64$ |
| $0111_B$ | $f_{ccu8}/128$ |
| $1000_B$ | $f_{ccu8}/256$ |
| $1001_B$ | $f_{ccu8}/512$ |
| $1010_B$ | $f_{ccu8}/1024$ |
| $1011_B$ | $f_{ccu8}/2048$ |
| $1100_B$ | $f_{ccu8}/4096$ |
| $1101_B$ | $f_{ccu8}/8192$ |
| $1110_B$ | $f_{ccu8}/16384$ |
| $1111_B$ | $f_{ccu8}/32768$ |

### 20.2.13.2 Floating Prescaler Mode

The floating prescaler mode can be used individually in each slice by setting the register **CC8yTC**.FPE = $1_B$. With this mode, the user can not only achieve a better precision on the counter clock for compare operations but also reduce the SW read access for the capture mode.

The floating prescaler mode contains additionally to the initial value register, **CC8yPSC**.PSIV, a compare register, **CC8yFPC**.PCMP with an associated shadow register.


Subject to Agreement on the Use of Product Information

**Figure 20-77** shows the structure of the prescaler in floating mode when the specific slice is in compare mode (no external signal is used for capture). In this mode, the value of the clock division is incremented by $1_D$ every time that a timer overflow/underflow (overflow if in Edge Aligned Mode, underflow if in Center Aligned Mode) occurs.

In this mode, the Compare Match (both channels) from the timer is AND gated with the Compare Match of the prescaler and every time that this event occurs, the value of the clock division is updated with the **CC8yPSC**.PSIV value in the immediately next timer overflow/underflow event.

To use just one compare channel to control the floating prescaler, the other compare channel must be disabled. To due this, the compare value, **CC8yCR1** or **CC8yCR2** (depending on which channel is used) needs to be set with a value bigger than the period, **CC8yPR**. This means that in edge aligned more, the maximum value for the timer period is $65534_D$, because the compare value of one channel needs to be set to $65535_D$.

The shadow transfer of the floating prescaler compare value, **CC8yFPC**.PCMP, is done following the same rules described on **Section 20.2.5.2**.

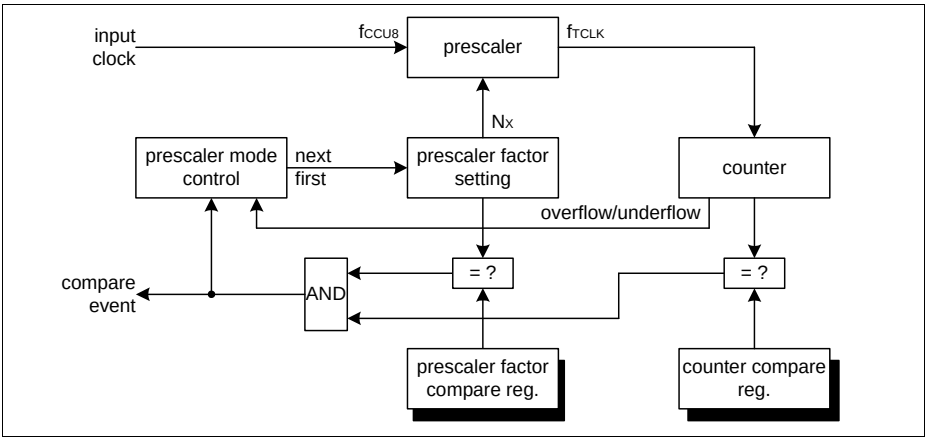



**Figure 20-77 Floating prescaler in compare mode overview**

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.


Subject to Agreement on the Use of Product Information

See the **Section 20.2.14** for a full description of the usage of the floating prescaler mode in conjunction with compare and capture modes.

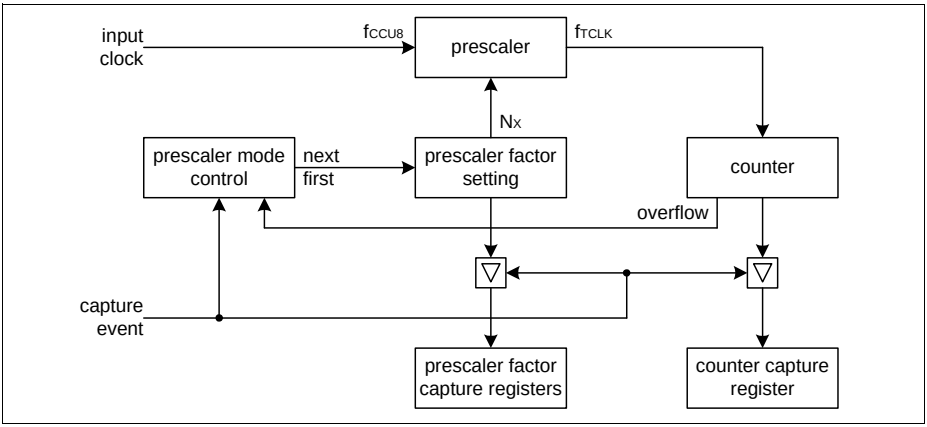



**Figure 20-78 Floating Prescaler in capture mode overview**

## 20.2.14 CCU8 Usage

### 20.2.14.1 PWM Signal Generation

The CCU8 offers a very flexible range in duty cycle configurations. This range is comprised between 0 to 100%.

To generate a PWM signal with a 100% duty cycle in Edge Aligned Mode, one should program the compare value, **CC8yCR1**.CR1/**CC8yCR2**.CR2, to $0000_H$, **Figure 20-79**.

In the same manner a 100% duty cycle signal can be generated in Center Aligned Mode, **Figure 20-80**.


Subject to Agreement on the Use of Product Information

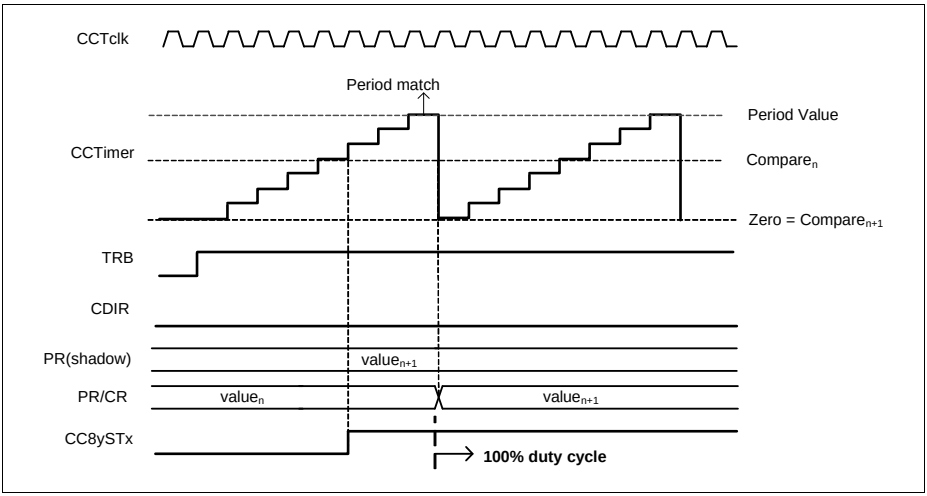



**Figure 20-79 PWM with 100% duty cycle - Edge Aligned Mode**

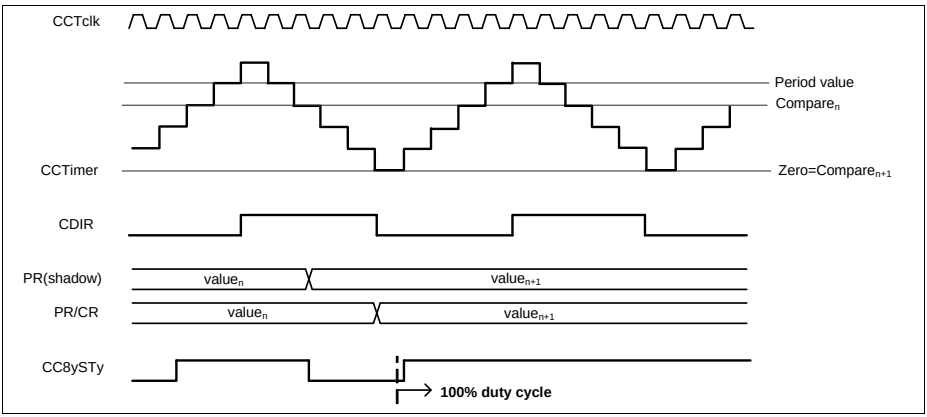



**Figure 20-80 PWM with 100% duty cycle - Center Aligned Mode**

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 20-81**.

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 20-82**.

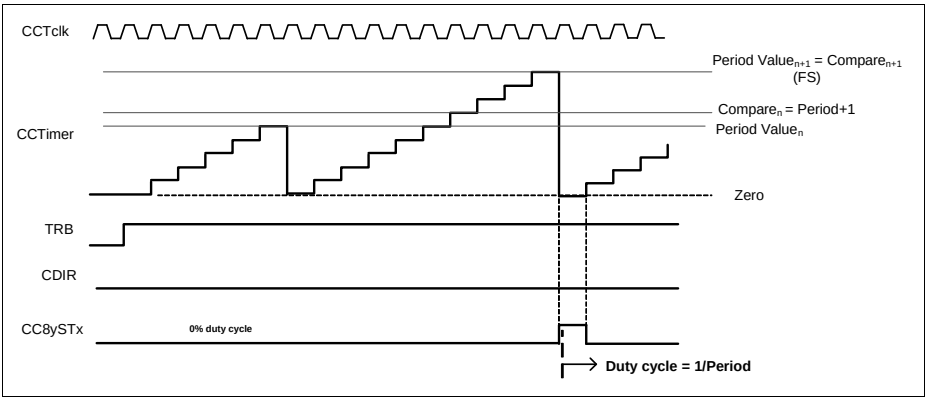



**Figure 20-81 PWM with 0% duty cycle - Edge Aligned Mode**

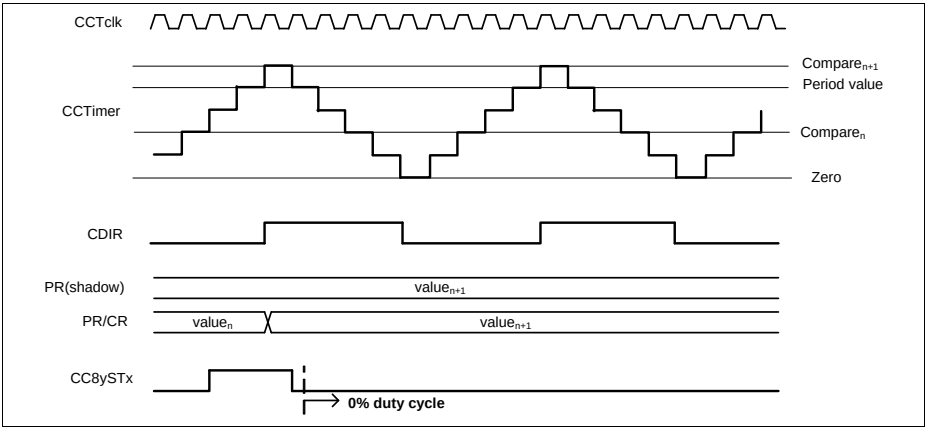



**Figure 20-82 PWM with 0% duty cycle - Center Aligned Mode**

### 20.2.14.2 Prescaler Usage

In Normal Prescaler Mode, the frequency of the $f_{tclk}$ fed to the specific CC8y is chosen from the **Table 20-8**, by setting the **CC8yPSC**.PSIV with the required value.

In Floating Prescaler Mode, the frequency of the $f_{tclk}$ can be modified over a selected timeframe, within the values specified in **Table 20-8**. This mechanism is specially useful if, in capture mode, the dynamic of the capture triggers is very slow or unknown.

In Capture Mode, the Floating Prescaler value is incremented by $1_D$ every time that a timer overflow happens and it is set with the initial programmed value when a capture event happens, see **Figure 20-83**.

When using the Floating Prescaler Mode in Capture Mode, the timer should be cleared each time that a capture event happens, **CC8yTC**.CAPC = $11_B$. By operating the Capture mode in conjunction with the Floating Prescaler, even for capture signals that have a periodicity bigger that 16 bits, it is possible to use just a single CCU8 slice without monitoring the interrupt events triggered by the timer overflow. For this the user just needs to know what is the timer capture value and the actual prescaler configuration at the time that the capture event occurred. These values are contained in each CC8yCxV register.

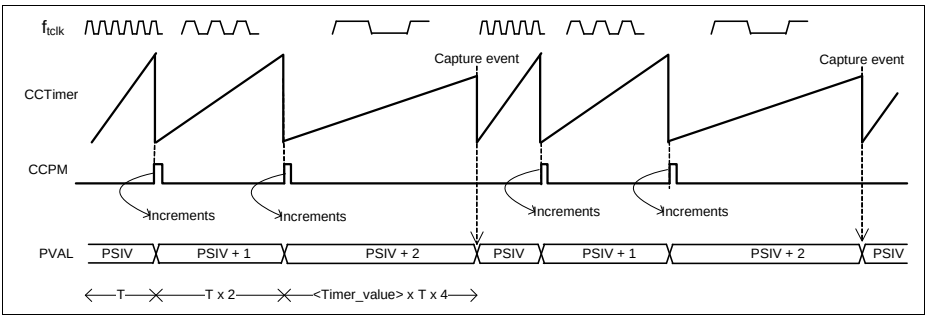



**Figure 20-83 Floating Prescaler capture mode usage**

When used in Compare Mode, the Floating Prescaler function may be used to achieve a fractional PWM frequency or to perform some frequency modulation.

The same incrementing by $1_D$ mechanism is done every time that a overflow/underflow of the Timer occurs (from any of the compare channels) and the actual Prescaler value doesn't match the one programmed into the **CC8yFPC**.PCMP register.

When a Compare Match from the Timer (from any of the compare channels) occurs and the actual Prescaler value is equal to the one programmed on the **CC8yFPC**.PCMP register, then the Prescaler value is set with the initial value, **CC8yPSC**.PSIV, in the immediately next occurrence of a timer overflow/underflow.

To use just one compare channel to control the floating prescaler, the other compare channel must be disabled. To due this, the compare value, **CC8yCR1** or **CC8yCR2** (depending on which channel is used) needs to be set with a value bigger than the period, **CC8yPR**. This means that in edge aligned more, the maximum value for the timer period is $65534_D$, because the compare value of one channel needs to be set to $65535_D$ (so the compare match of the associated channel is disabled).

In **Figure 20-84**, 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


Subject to Agreement on the Use of Product Information

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 20-85**.

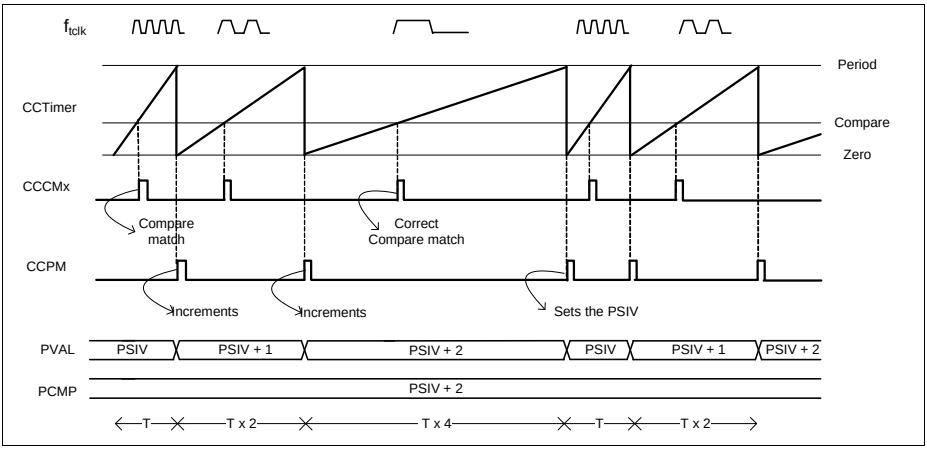



**Figure 20-84 Floating Prescaler compare mode usage - Edge Aligned**

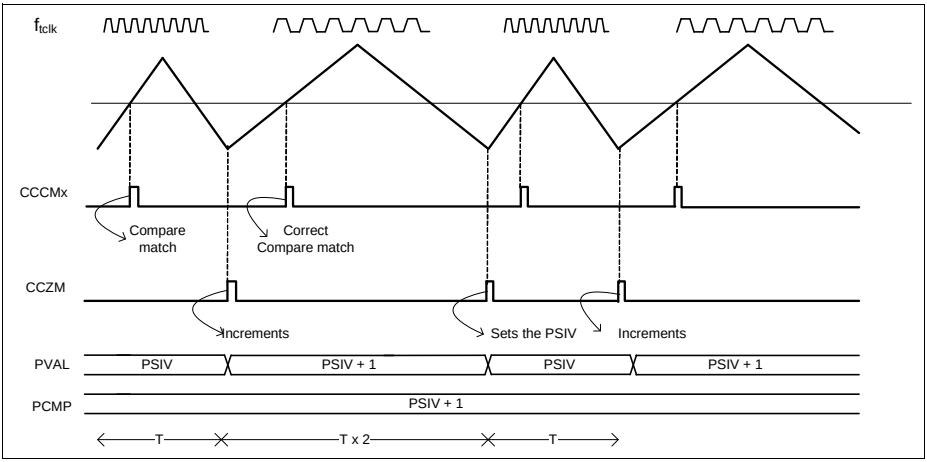



**Figure 20-85 Floating Prescaler compare mode usage - Center Aligned**


Subject to Agreement on the Use of Product Information

### 20.2.14.3 PWM Dither

The Dither function can be used to achieve a very fine precision on the periodicity of the output state in compare mode. The value set in the dither compare register, **CC8yDIT**.DCV is crosschecked against the actual value of the dither counter and every time that the dither counter is smaller than the comparison value one of the follows actions is taken:

- The period is extended for 1 clock cycle - **CC8yTC**.DITHE = $01_B$; in edge aligned mode
- The period is extended for 2 clock cycles - **CC8yTC**.DITHE = $01_B$; in center aligned mode
- The comparison match while counting up (**CC8yTCST**.CDIR = $0_B$) is delayed (this means that the status bit is going to stay in the SET state 1 cycle less) for 1 clock cycle - **CC8yTC**.DITHE = $10_B$;
- The period is extended for 1 clock cycle and the comparison match while counting up is delayed for 1 clock cycle - **CC8yTC**.DITHE = $11_B$; in edge aligned mode
- The period is extended for 2 clock cycles and the comparison match while counting up is delayed for 1 clock cycle; center aligned mode

The bit reverse counter distributes the number programmed in the **CC8yDIT**.DCV throughout 16 timer periods.

**Table 20-9**, describes the bit reverse distribution versus the programmed value on the **CC8yDIT**.DCV field. The fields marked as '0' indicate that in that counter period, one of the above described actions, is going to be performed.

**Table 20-9 Bit reverse distribution**

| | **DCV** | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Dither counter** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| C | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 2 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| A | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 6 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| E | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |


Subject to Agreement on the Use of Product Information

**Table 20-9 Bit reverse distribution** (cont'd)

| | DCV | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Dither counter** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
| 5 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| D | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 3 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| B | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 7 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| F | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

The bit reverse distribution versus the programmed **CC8yDIT**.DCV value results in the following values for the Period and duty cycle:

DITHE = $01_B$

*Period* = [(16 - DCV) x T + DCV x (T + 1)]/16; in Edge Aligned Mode (20.10)

*Duty cycle* = [(16 - DCV) x d/T + DCV x (d+1)/(T + 1)]/16; in Edge Aligned Mode (20.11)

*Period* = [(16 - DCV) x T + DCV x (T + 2)]/16; in Center Aligned Mode (20.12)

*Duty cycle* = [(16 - DCV) x d/T + DCV x (d+2)/(T + 2)]/16; in Center Aligned Mode (20.13)

DITHE = $10_B$

*Period* = T ; in Edge Aligned Mode (20.14)

*Duty cycle* = [(16 - DCV) x d/T + DCV x (d-1)/T ]/16; in Edge Aligned Mode (20.15)

*Period* = T ; in Center Aligned Mode (20.16)

*Duty cycle* = [(16 - DCV) x d/T + DCV x (d-1)/T]/16; in Center Aligned Mode (20.17)

DITHE = $11_B$

*Period* = [(16 - DCV) x T + DCV x (T + 1)]/16; in Edge Aligned Mode (20.18)

*Duty cycle* = [(16 - DCV) x d/T + DCV x d/(T + 1)]/16; in Edge Aligned Mode (20.19)


Subject to Agreement on the Use of Product Information

*Period* = [(16 - DCV) x T + DCV x (T + 2)]/16; in Center Aligned Mode (20.20)

*Duty cycle* = [(16 - DCV) x d/T + DCV x (d+1)/(T + 2)]/16; in Center Aligned Mode(20.21)

where:

T - Original period of the signal, see **Section 20.2.5.1**

d - Original duty cycle of the signal, see **Section 20.2.5.1**

### 20.2.14.4 Capture Mode Usage

Each slice has the possibility of using 2 or 4 capture registers. Using only 2 capture registers means that only 1 Event was linked to a captured trigger. To use the four capture registers, both capture triggers need to be mapped into an Event (it can be the same signal with different edges selected or two different signals) or the **CC8yTC**.SCE field needs to be set to $1_B$, which enables the linking of the 4 capture registers.

The internal slice mechanism for capturing is the same for the capture trigger 1 or capture trigger 0.

Different Capture Events - SCE = $0_B$

Capture trigger 1 (CCcapt1) is appointed to the capture register 2, **CC8yC2V** and capture register 3, **CC8yC3V**, while trigger 0 is appointed to capture register 1, **CC8yC1V** and 0, **CC8yC0V**.

In each CCcapt0 event, the timer value is stored into **CC8yC1V** and the value of the **CC8yC1V** is transferred into the **CC8yC0V**.

In each CCcapt1 event, the timer value is stored into capture register **CC8yC3V** and the value of the capture register **CC8yC3V** is transferred into **CC8yC2V**.

The previous capture/transfer mechanism only happens if the specific register is not full. A capture register becomes full when receives a new value and becomes empty after the SW has read back the value.

The full flag is cleared every time that the SW reads back the **CC8yC0V**, **CC8yC1V**, **CC8yC2V** or **CC8yC3V** register. The SW can be informed of a new capture trigger by enabling the interrupt source linked to the specific Event. This means that every time that a capture is made an interrupt pulse is generated.

In the case that the Floating Prescaler Mode is being used, the actual value of the clock division is also stored in the capture register (CC8yCxV).

**Figure 20-86** 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


Subject to Agreement on the Use of Product Information

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).

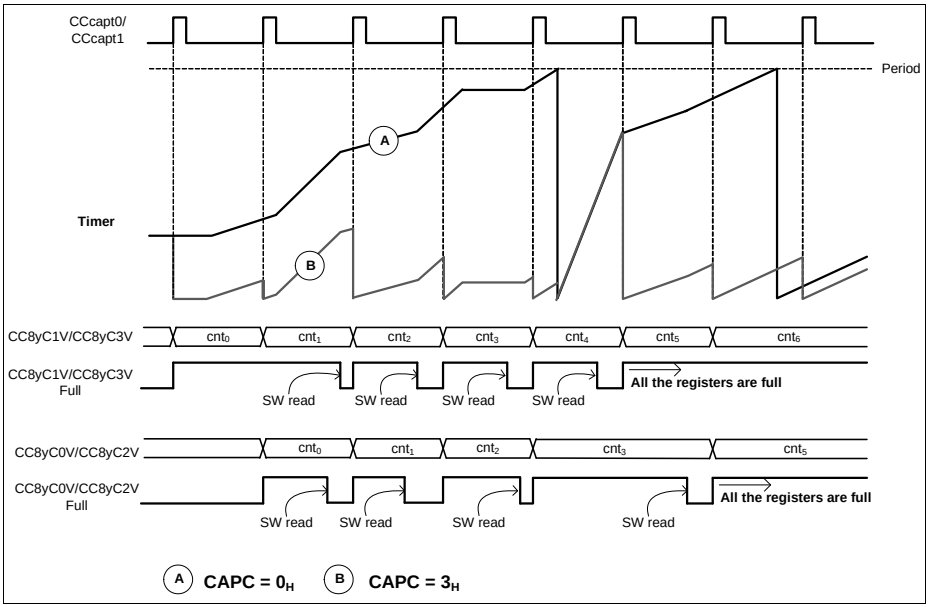



**Figure 20-86 Capture mode usage - single channel**

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 20-87**. The CC80ST output of slice 0 was used as capture trigger in the CC81 slice (active on rising and falling edge). The CC80ST output is used as known timebase marker, while the slice timer used for capture is being controlled by external events, e.g. external count.


Subject to Agreement on the Use of Product Information

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.

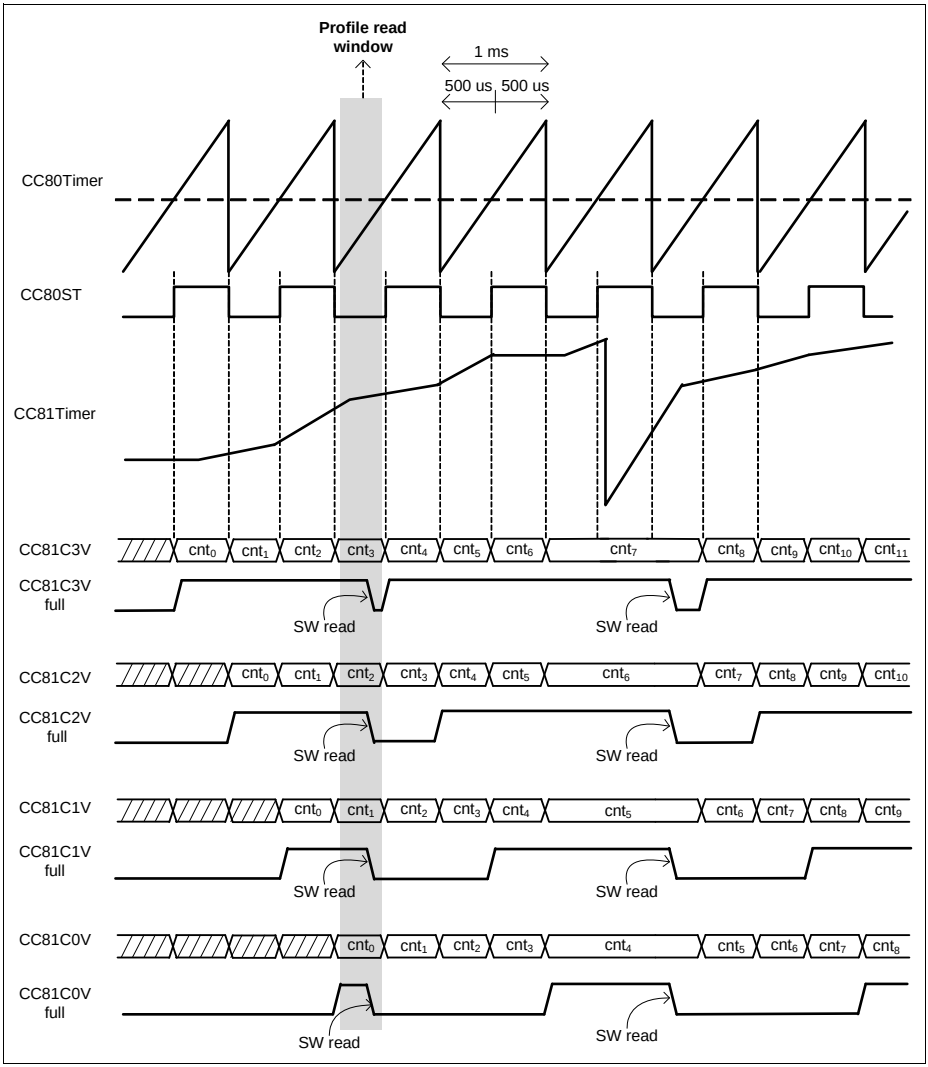



**Figure 20-87 Three Capture profiles - CC8yTC.SCE = $1_B$**

To calculate the three different profiles in **Figure 20-87**, 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 20-88**.

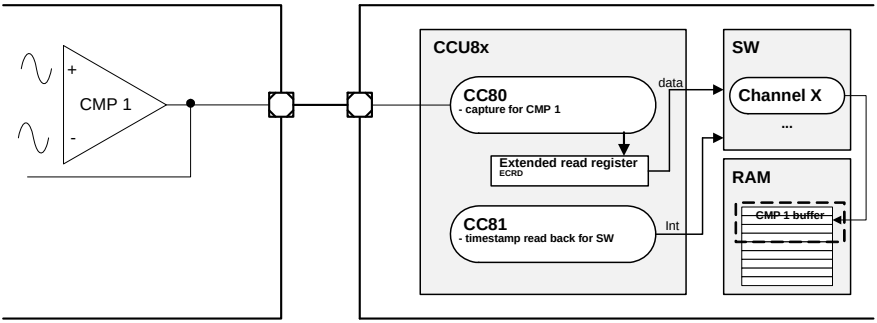



**Figure 20-88 High dynamics capturing with software controlled timestamp**

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.


Subject to Agreement on the Use of Product Information

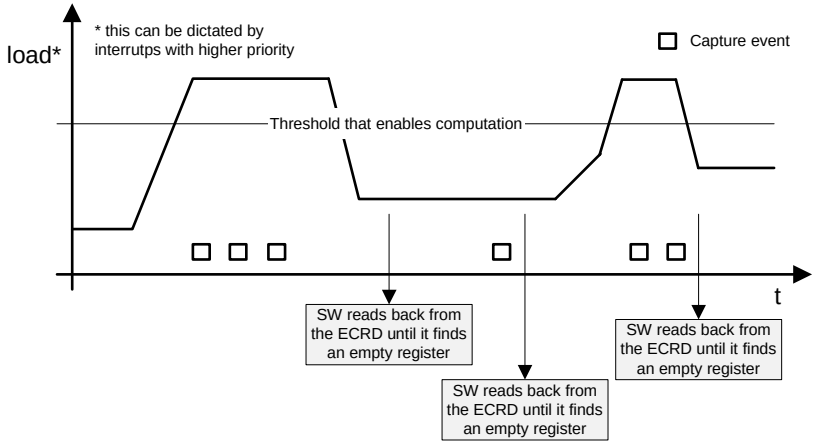



**Figure 20-89 Extended read back during high load**

### 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 20-90**.

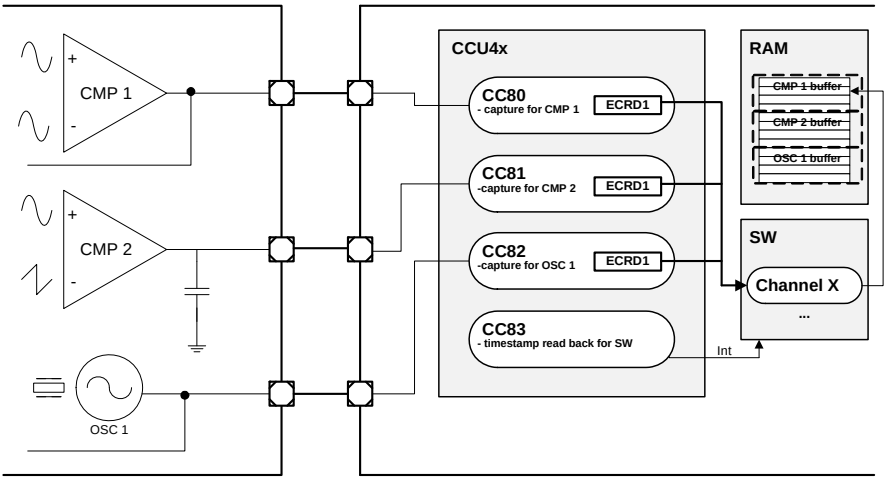



**Figure 20-90 Capture grouping with extended read back**


Subject to Agreement on the Use of Product Information

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 20-91** (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).


Subject to Agreement on the Use of Product Information

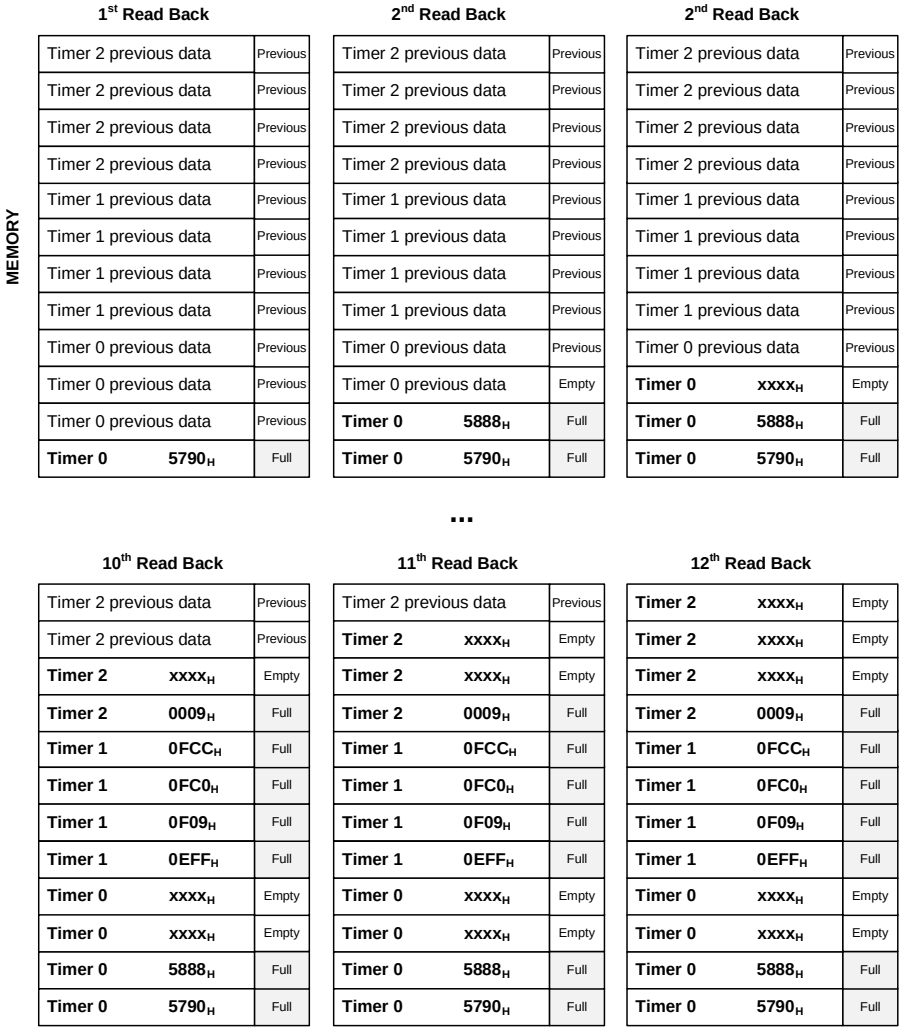



**Figure 20-91 Memory structure for extended read back**

## 20.2.14.5 Parity Checker Usage

The parity checker function available on the CCU8 uses of one CCU4 timer slice, to control the delay between the update of the outputs and the consequent update on the external switch/driver.

This CCU4 timer slice is configured to work in edge aligned mode, with single shot mode active and with a configured external flush & start function. This function is connected to the parity checker update output signal, CCU8x.IGBTO. The timer slice compare and period values need to be programmed accordingly to the delay that is foreseen between the outputs of the CCU8 and the consequent update on the driver/switch parity output. The connections between the CCU8, CCU4 and the external HW can be seen on **Figure 20-92**.

**Figure 20-93** 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).


Subject to Agreement on the Use of Product Information

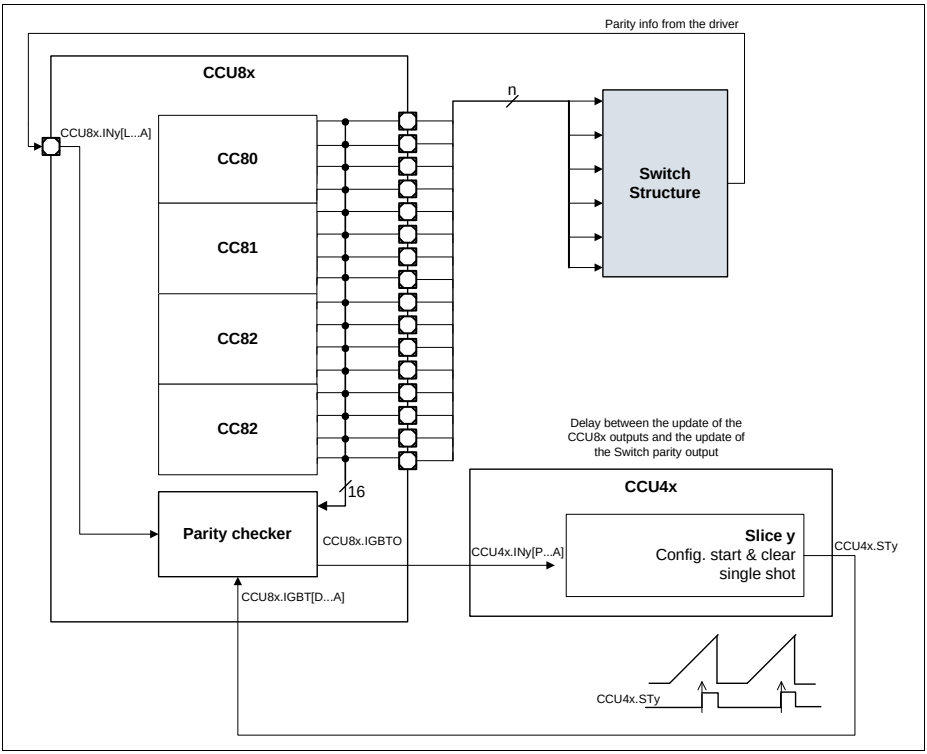



**Figure 20-92 Parity Checker connections**


Subject to Agreement on the Use of Product Information

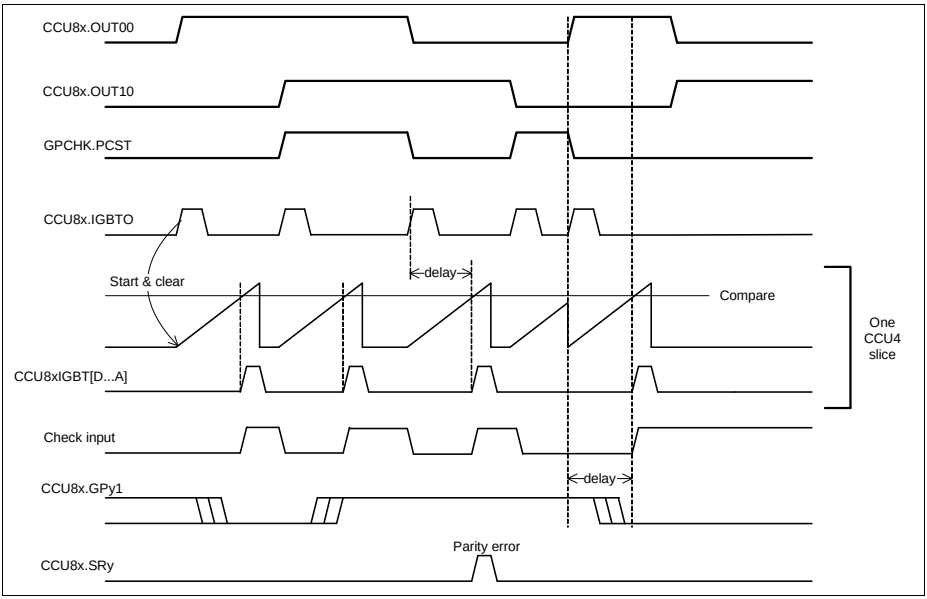



**Figure 20-93 Parity Checker timing example**

## 20.3 Service Request Generation

Each CCU8 slice has a interrupt structure as the one see in **Figure 20-94**. 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 20-10** describes the interrupt sources of each CCU8 slice.

The interrupt sources, Period Match while counting up and one Match while counting down are ORed together. The same mechanism is applied to the Compare Match while counting up and Compare Match while counting down of both compare channels.

The interrupt sources for the external events are directly linked with the configuration set on the **CC8yINS**.EVxEM. If an event is programmed to be active on both edges, that means that service request pulse is going to be generated when any transition on the external signal is detected. If the event is linked with a level function, the **CC8yINS**.EVxEM still can be programmed to enable a service request pulse. The TRAP

event doesn't need any extra configuration for generating the service request pulse when the slice enters the TRAP state.

**Table 20-10 Interrupt sources**

| Signal | Description |
|---|---|
| CCINEV0_E | Event 0 edge(s) information from event selector. Used when an external signal should trigger an interrupt. |
| CCINEV1_E | Event 1 edge(s) information from event selector. Used when an external signal should trigger an interrupt (It also can be the parity checker pattern fail information, if the parity checker was enabled). |
| CCINEV2_E | Event 2 edge(s) information from event selector. Used when an external signal should trigger an interrupt. |
| CCPM_U | Period Match while counting up. |
| CCCM1_U | Compare Match while counting up from compare channel 1. |
| CCCM1_D | Compare Match while counting down from compare channel 1. |
| CCCM2_U | Compare Match while counting up from compare channel 2. |
| CCCM2_D | Compare Match while counting down from compare channel 2. |
| CCOM_D | One Match while counting down. |
| Trap state set | Entering Trap State. Will set the E2AS. |

Each of the interrupt events can then be forwarded to one, of the slice's four service request lines, **Figure 20-95**. The value set on the **CC8ySRS** controls which interrupt event is mapped into which service request line.


Subject to Agreement on the Use of Product Information

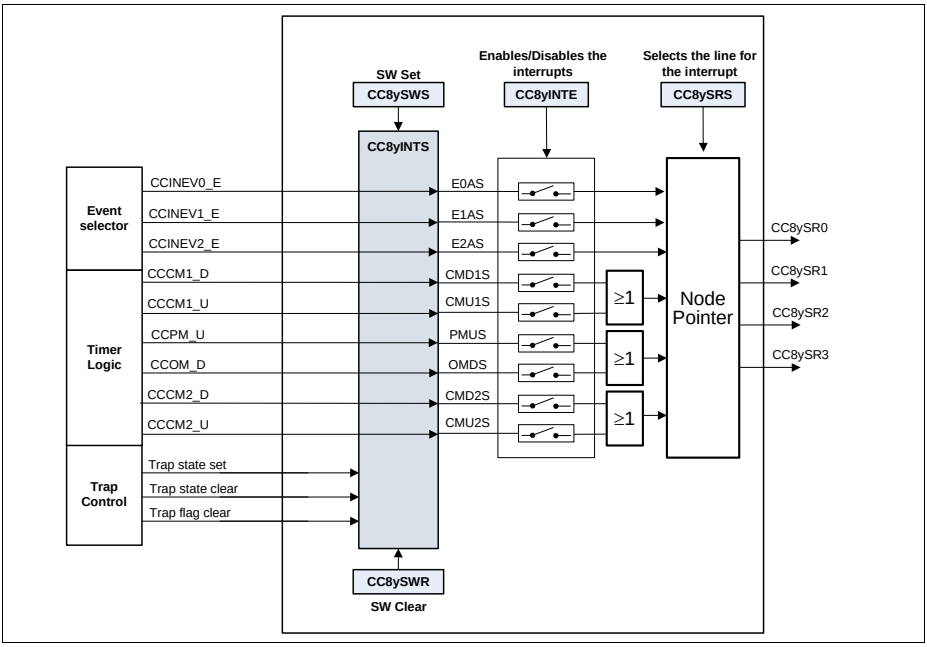



**Figure 20-94 Slice interrupt node pointer overview**

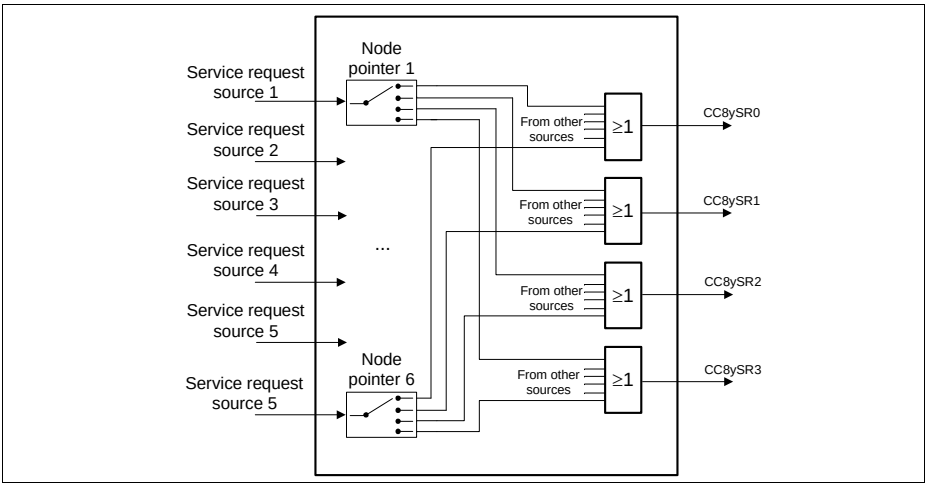



**Figure 20-95 Slice interrupt selector overview**


Subject to Agreement on the Use of Product Information

The four service request lines of each slice are OR together inside the kernel of the CCU8, see **Figure 20-96**. This means that there are only four service request lines per CCU8, that can have in each line interrupt requests coming from different slices.

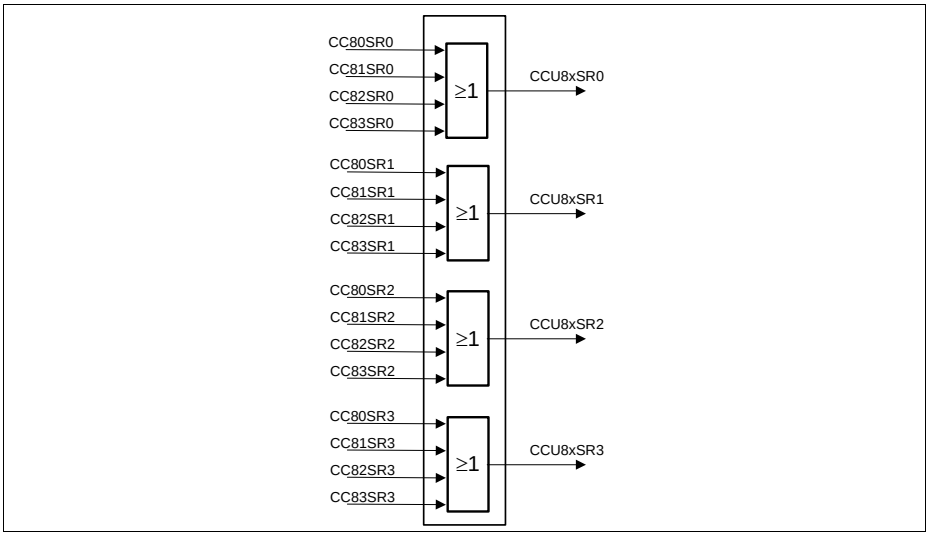



**Figure 20-96 CCU8 service request overview**

## 20.4 Debug Behavior

In suspend mode, the functional clocks for all slices as well the prescaler are stopped. The registers can still be accessed by the CPU (read only). This mode is useful for debugging purposes, e.g., where the current device status should be frozen in order to get a snapshot of the internal values. In suspend mode, all the slice counters are stopped. The suspend mode is non-intrusive concerning the register bits. This means register bits are not modified by hardware when entering or leaving the suspend mode.

Entry into suspend mode can be configured at the kernel level by means of the field **GCTRL**.SUSCFG.

The module is only functional after the suspend signal becomes inactive.

## 20.5 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, $f_{ccu8}$.

### 20.5.1 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 20-11 External clock operating conditions**

| Parameter | Symbol | Values | | | Unit | Note / Test Condition |
|---|---|---|---|---|---|---|
| | | Min. | Typ. | Max. | | |
| Frequency | $f_{eclk}$ | – | – | $f_{ccu8}/4$ | MHz | |
| ON time | $ton_{eclk}$ | $2T_{ccu8}$ 1)2) | – | – | ns | |
| OFF time | $toff_{eclk}$ | $2T_{ccu8}$ 1)2) | – | – | ns | Only the rising edge is used |

1) Only valid if the signal was not previously synchronized/generated with the fccu4 clock (or a synchronous clock)

2) 50% duty cycle is not obligatory

### 20.5.2 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**).

## 20.6 Initialization and System Dependencies

### 20.6.1 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 **Section 20.2.8.1**.

### 20.6.2 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


Subject to Agreement on the Use of Product Information

- Module input triggers for the CCU8 must not exceed the module clock frequency (if the triggers are generated internally in the device)
- Module input triggers for the CCU8 must not exceed the frequency dictated in **Section 20.5.1**
- Frequency of the CCU8 outputs used as triggers/functions on other modules, must be crosschecked on the end point
- Applying and removing CCU8 from reset, can cause unwanted operations in other modules. This can occur if the modules are using CCU8 outputs as triggers/functions.


Subject to Agreement on the Use of Product Information

## 20.7 Registers

### Registers Overview

The absolute register address is calculated by adding:

Module Base Address + Offset Address

**Table 20-12 Registers Address Space**

| Module | Base Address | End Address | Note |
|---|---|---|---|
| CCU80 | $50000000_H$ | $5000FFFF_H$ | |


Subject to Agreement on the Use of Product Information

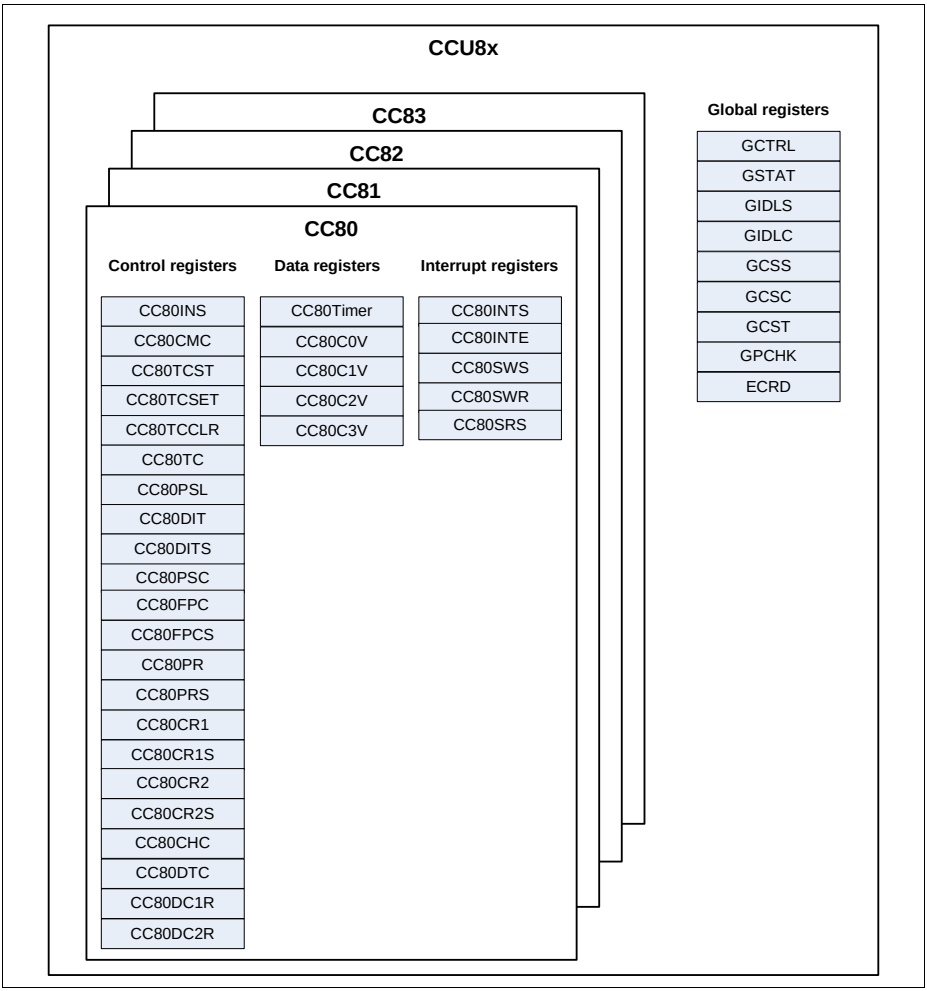



**Figure 20-97 CCU8 registers overview**


Subject to Agreement on the Use of Product Information

**Table 20-13 Register Overview of CCU8**

| Short Name | Description | Offset Addr.[1)] | Access Mode | | Description See |
|---|---|---|---|---|---|
| | | | Read | Write | |
| CCU8 Global Registers | | | | | |
| GCTRL | Module General Control Register | $0000_H$ | U, PV | U, PV | Page 20-111 |
| GSTAT | General Slice Status Register | $0004_H$ | U, PV | BE | Page 20-114 |
| GIDLS | General Idle Enable Register | $0008_H$ | U, PV | U, PV | Page 20-116 |
| GIDLC | General Idle Disable Register | $000C_H$ | U, PV | U, PV | Page 20-118 |
| GCSS | General Channel Set Register | $0010_H$ | U, PV | U, PV | Page 20-119 |
| GCSC | General Channel Clear Register | $0014_H$ | U, PV | U, PV | Page 20-122 |
| GCST | General Channel Status Register | $0018_H$ | U, PV | BE | Page 20-125 |
| GPCHK | Parity Checker Configuration Register | $001C_H$ | U, PV | U, PV | Page 20-128 |
| MIDR | Module Identification Register | $0080_H$ | U, PV | BE | Page 20-130 |
| CC80 Registers | | | | | |
| CC80INS | Input Selector Unit Configuration | $0100_H$ | U, PV | U, PV | Page 20-131 |
| CC80CMC | Connection Matrix Configuration | $0104_H$ | U, PV | U, PV | Page 20-133 |
| CC80TCST | Timer Run Status | $0108_H$ | U, PV | BE | Page 20-137 |
| CC80TCSET | Timer Run Set | $010C_H$ | U, PV | U,PV | Page 20-138 |
| CC80TCCLR | Timer Run Clear | $0110_H$ | U, PV | U, PV | Page 20-139 |
| CC80TC | General Timer Configuration | $0114_H$ | U, PV | U, PV | Page 20-140 |
| CC80PSL | Output Passive Level Configuration | $0118_H$ | U, PV | U, PV | Page 20-145 |
| CC80DIT | Dither Configuration | $011C_H$ | U, PV | BE | Page 20-146 |
| CC80DITS | Dither Shadow Register | $0120_H$ | U, PV | U, PV | Page 20-147 |
| CC80PSC | Prescaler Configuration | $0124_H$ | U, PV | U, PV | Page 20-148 |


Subject to Agreement on the Use of Product Information

**Table 20-13 Register Overview of CCU8** (cont'd)

| Short Name | Description | Offset Addr.[1)] | Access Mode | | Description See |
|---|---|---|---|---|---|
| | | | Read | Write | |
| CC80FPC | Prescaler Compare Value | $0128_H$ | U, PV | U, PV | Page 20-148 |
| CC80FPCS | Prescaler Shadow Compare Value | $012C_H$ | U, PV | U, PV | Page 20-149 |
| CC80PR | Timer Period Value | $0130_H$ | U, PV | BE | Page 20-150 |
| CC80PRS | Timer Period Shadow Value | $0134_H$ | U, PV | U, PV | Page 20-151 |
| CC80CR1 | Timer Compare Value for Channel 1 | $0138_H$ | U, PV | BE | Page 20-151 |
| CC80CR1S | Timer Compare Shadow Value for Channel 1 | $013C_H$ | U, PV | U, PV | Page 20-152 |
| CC80CR2 | Timer Compare Value for Channel 2 | $0140_H$ | U, PV | BE | Page 20-153 |
| CC80CR2S | Timer Compare Shadow Value for Channel 2 | $0144_H$ | U, PV | U, PV | Page 20-154 |
| CC80CHC | Channel Control | $0148_H$ | U, PV | U, PV | Page 20-155 |
| CC80DTC | Dead Time Control | $014C_H$ | U, PV | U, PV | Page 20-157 |
| CC80DC1R | Channel 1 Dead Time Counter Values | $0150_H$ | U, PV | U, PV | Page 20-158 |
| CC80DC2R | Channel 2 Dead Time Counter Values | $0154_H$ | U, PV | U, PV | Page 20-159 |
| CC80TIMER | Timer Current Value | $0170_H$ | U, PV | U, PV | Page 20-159 |
| CC80C0V | Capture Register 0 Value | $0174_H$ | U, PV | BE | Page 20-160 |
| CC80C1V | Capture Register 1 Value | $0178_H$ | U, PV | BE | Page 20-161 |
| CC80C2V | Capture Register 2 Value | $017C_H$ | U, PV | BE | Page 20-162 |
| CC80C3V | Capture Register 3 Value | $0180_H$ | U, PV | BE | Page 20-163 |
| CC80INTS | Interrupt Status | $01A0_H$ | U, PV | BE | Page 20-164 |
| CC80INTE | Interrupt Enable | $01A4_H$ | U, PV | U, PV | Page 20-166 |
| CC80SRS | Interrupt Configuration | $01A8_H$ | U, PV | U, PV | Page 20-168 |
| CC80SWS | Interrupt Status Set | $01AC_H$ | U, PV | U, PV | Page 20-170 |
| CC80SWR | Interrupt Status Clear | $01B0_H$ | U, PV | U, PV | Page 20-172 |
| CC80STC | Shadow Transfer Control | $01B4_H$ | U, PV | U, PV | Page 20-174 |
| CC80ECRD0 | Extended Read Back 0 | $01B8_H$ | U, PV | BE | Page 20-175 |


Subject to Agreement on the Use of Product Information

**Table 20-13 Register Overview of CCU8** (cont'd)

| Short Name | Description | Offset Addr.[1)] | Access Mode | | Description See |
|---|---|---|---|---|---|
| | | | Read | Write | |
| CC80ECRD1 | Extended Read Back 1 | $01BC_H$ | U, PV | BE | Page 20-176 |
| CC81 Registers | | | | | |
| CC81INS | Input Selector Unit Configuration | $0200_H$ | U, PV | U, PV | Page 20-131 |
| CC81CMC | Connection Matrix Configuration | $0204_H$ | U, PV | U, PV | Page 20-133 |
| CC81TCST | Timer Run Status | $0208_H$ | U, PV | BE | Page 20-137 |
| CC81TCSET | Timer Run Set | $020C_H$ | U, PV | U,PV | Page 20-138 |
| CC81TCCLR | Timer Run Clear | $0210_H$ | U, PV | U, PV | Page 20-139 |
| CC81TC | General Timer Configuration | $0214_H$ | U, PV | U, PV | Page 20-140 |
| CC81PSL | Output Passive Level Configuration | $0218_H$ | U, PV | U, PV | Page 20-145 |
| CC81DIT | Dither Configuration | $021C_H$ | U, PV | BE | Page 20-146 |
| CC81DITS | Dither Shadow Register | $0220_H$ | U, PV | U, PV | Page 20-147 |
| CC81PSC | Prescaler Configuration | $0224_H$ | U, PV | U, PV | Page 20-148 |
| CC81FPC | Prescaler Compare Value | $0228_H$ | U, PV | U, PV | Page 20-148 |
| CC81FPCS | Prescaler Shadow Compare Value | $022C_H$ | U, PV | U, PV | Page 20-149 |
| CC81PR | Timer Period Value | $0230_H$ | U, PV | BE | Page 20-150 |
| CC81PRS | Timer Period Shadow Value | $0234_H$ | U, PV | U, PV | Page 20-151 |
| CC81CR1 | Timer Compare Value for Channel 1 | $0238_H$ | U, PV | BE | Page 20-151 |
| CC81CR1S | Timer Compare Shadow Value for Channel 1 | $023C_H$ | U, PV | U, PV | Page 20-152 |
| CC81CR2 | Timer Compare Value for Channel 2 | $0240_H$ | U, PV | BE | Page 20-153 |
| CC81CR2S | Timer Compare Shadow Value for Channel 2 | $0244_H$ | U, PV | U, PV | Page 20-154 |
| CC81CHC | Channel Control | $0248_H$ | U, PV | U, PV | Page 20-155 |
| CC81DTC | Dead Time Control | $024C_H$ | U, PV | U, PV | Page 20-157 |

**Table 20-13 Register Overview of CCU8** (cont'd)

| Short Name | Description | Offset Addr.[1)] | Access Mode | | Description See |
|---|---|---|---|---|---|
| | | | Read | Write | |
| CC81DC1R | Channel 1 Dead Time Counter Values | $0250_H$ | U, PV | U, PV | Page 20-158 |
| CC81DC2R | Channel 2 Dead Time Counter Values | $0254_H$ | U, PV | U, PV | Page 20-159 |
| CC81TIMER | Timer Current Value | $0270_H$ | U, PV | U, PV | Page 20-159 |
| CC81C0V | Capture Register 0 Value | $0274_H$ | U, PV | BE | Page 20-160 |
| CC81C1V | Capture Register 1 Value | $0278_H$ | U, PV | BE | Page 20-161 |
| CC81C2V | Capture Register 2 Value | $027C_H$ | U, PV | BE | Page 20-162 |
| CC81C3V | Capture Register 3 Value | $0280_H$ | U, PV | BE | Page 20-163 |
| CC81INTS | Interrupt Status | $02A0_H$ | U, PV | BE | Page 20-164 |
| CC81INTE | Interrupt Enable | $02A4_H$ | U, PV | U, PV | Page 20-166 |
| CC81SRS | Interrupt Configuration | $02A8_H$ | U, PV | U, PV | Page 20-168 |
| CC81SWS | Interrupt Status Set | $02AC_H$ | U, PV | U, PV | Page 20-170 |
| CC81SWR | Interrupt Status Clear | $02B0_H$ | U, PV | U, PV | Page 20-172 |
| CC81STC | Shadow Transfer Control | $02B4_H$ | U, PV | U, PV | Page 20-174 |
| CC81ECRD0 | Extended Read Back 0 | $02B8_H$ | U, PV | BE | Page 20-175 |
| CC81ECRD1 | Extended Read Back 1 | $02BC_H$ | U, PV | BE | Page 20-176 |
| CC82 Registers | | | | | |
| CC82INS | Input Selector Unit Configuration | $0300_H$ | U, PV | U, PV | Page 20-131 |
| CC82CMC | Connection Matrix Configuration | $0304_H$ | U, PV | U, PV | Page 20-133 |
| CC82TCST | Timer Run Status | $0308_H$ | U, PV | BE | Page 20-137 |
| CC82TCSET | Timer Run Set | $030C_H$ | U, PV | U,PV | Page 20-138 |
| CC82TCCLR | Timer Run Clear | $0310_H$ | U, PV | U, PV | Page 20-139 |
| CC82TC | General Timer Configuration | $0314_H$ | U, PV | U, PV | Page 20-140 |
| CC82PSL | Output Passive Level Configuration | $0318_H$ | U, PV | U, PV | Page 20-145 |
| CC82DIT | Dither Configuration | $031C_H$ | U, PV | BE | Page 20-146 |
| CC82DITS | Dither Shadow Register | $0320_H$ | U, PV | U, PV | Page 20-147 |

**Table 20-13 Register Overview of CCU8** (cont'd)

| Short Name | Description | Offset Addr.[1)] | Access Mode | | Description See |
|---|---|---|---|---|---|
| | | | Read | Write | |
| CC82PSC | Prescaler Configuration | $0324_H$ | U, PV | U, PV | Page 20-148 |
| CC82FPC | Prescaler Compare Value | $0328_H$ | U, PV | U, PV | Page 20-148 |
| CC82FPCS | Prescaler Shadow Compare Value | $032C_H$ | U, PV | U, PV | Page 20-149 |
| CC82PR | Timer Period Value | $0330_H$ | U, PV | BE | Page 20-150 |
| CC82PRS | Timer Period Shadow Value | $0334_H$ | U, PV | U, PV | Page 20-151 |
| CC82CR1 | Timer Compare Value for Channel 1 | $0338_H$ | U, PV | BE | Page 20-151 |
| CC82CR1S | Timer Compare Shadow Value for Channel 1 | $033C_H$ | U, PV | U, PV | Page 20-152 |
| CC82CR2 | Timer Compare Value for Channel 2 | $0340_H$ | U, PV | BE | Page 20-153 |
| CC82CR2S | Timer Compare Shadow Value for Channel 2 | $0344_H$ | U, PV | U, PV | Page 20-154 |
| CC82CHC | Channel Control | $0348_H$ | U, PV | U, PV | Page 20-155 |
| CC82DTC | Dead Time Control | $034C_H$ | U, PV | U, PV | Page 20-157 |
| CC82DC1R | Channel 1 Dead Time Counter Values | $0350_H$ | U, PV | U, PV | Page 20-158 |
| CC82DC2R | Channel 2 Dead Time Counter Values | $0354_H$ | U, PV | U, PV | Page 20-159 |
| CC82TIMER | Timer Current Value | $0370_H$ | U, PV | U, PV | Page 20-159 |
| CC82C0V | Capture Register 0 Value | $0374_H$ | U, PV | BE | Page 20-160 |
| CC82C1V | Capture Register 1 Value | $0378_H$ | U, PV | BE | Page 20-161 |
| CC82C2V | Capture Register 2 Value | $037C_H$ | U, PV | BE | Page 20-162 |
| CC82C3V | Capture Register 3 Value | $0380_H$ | U, PV | BE | Page 20-163 |
| CC82INTS | Interrupt Status | $03A0_H$ | U, PV | BE | Page 20-164 |
| CC82INTE | Interrupt Enable | $03A4_H$ | U, PV | U, PV | Page 20-166 |
| CC82SRS | Interrupt Configuration | $03A8_H$ | U, PV | U, PV | Page 20-168 |
| CC82SWS | Interrupt Status Set | $03AC_H$ | U, PV | U, PV | Page 20-170 |
| CC82SWR | Interrupt Status Clear | $03B0_H$ | U, PV | U, PV | Page 20-172 |
| CC82STC | Shadow Transfer Control | $03B4_H$ | U, PV | U, PV | Page 20-174 |


Subject to Agreement on the Use of Product Information

**Table 20-13 Register Overview of CCU8** (cont'd)

| Short Name | Description | Offset Addr.[1)] | Access Mode | | Description See |
|---|---|---|---|---|---|
| | | | Read | Write | |
| CC82ECRD0 | Extended Read Back 0 | $03B8_H$ | U, PV | BE | Page 20-175 |
| CC82ECRD1 | Extended Read Back 1 | $03BC_H$ | U, PV | BE | Page 20-176 |
| CC83 Registers | | | | | |
| CC83INS | Input Selector Unit Configuration | $0400_H$ | U, PV | U, PV | Page 20-131 |
| CC83CMC | Connection Matrix Configuration | $0404_H$ | U, PV | U, PV | Page 20-133 |
| CC83TCST | Timer Run Status | $0408_H$ | U, PV | BE | Page 20-137 |
| CC83TCSET | Timer Run Set | $040C_H$ | U, PV | U,PV | Page 20-138 |
| CC83TCCLR | Timer Run Clear | $0410_H$ | U, PV | U, PV | Page 20-139 |
| CC83TC | General Timer Configuration | $0414_H$ | U, PV | U, PV | Page 20-140 |
| CC83PSL | Output Passive Level Configuration | $0418_H$ | U, PV | U, PV | Page 20-145 |
| CC83DIT | Dither Configuration | $041C_H$ | U, PV | BE | Page 20-146 |
| CC83DITS | Dither Shadow Register | $0420_H$ | U, PV | U, PV | Page 20-147 |
| CC83PSC | Prescaler Configuration | $0424_H$ | U, PV | U, PV | Page 20-148 |
| CC83FPC | Prescaler Compare Value | $0428_H$ | U, PV | U, PV | Page 20-148 |
| CC83FPCS | Prescaler Shadow Compare Value | $042C_H$ | U, PV | U, PV | Page 20-149 |
| CC83PR | Timer Period Value | $0430_H$ | U, PV | BE | Page 20-150 |
| CC83PRS | Timer Period Shadow Value | $0434_H$ | U, PV | U, PV | Page 20-151 |
| CC83CR1 | Timer Compare Value for Channel 1 | $0438_H$ | U, PV | BE | Page 20-151 |
| CC83CR1S | Timer Compare Shadow Value for Channel 1 | $043C_H$ | U, PV | U, PV | Page 20-152 |
| CC83CR2 | Timer Compare Value for Channel 2 | $0440_H$ | U, PV | BE | Page 20-153 |
| CC83CR2S | Timer Compare Shadow Value for Channel 2 | $0444_H$ | U, PV | U, PV | Page 20-154 |
| CC83CHC | Channel Control | $0448_H$ | U, PV | U, PV | Page 20-155 |
| CC83DTC | Dead Time Control | $044C_H$ | U, PV | U, PV | Page 20-157 |


Subject to Agreement on the Use of Product Information

**Table 20-13 Register Overview of CCU8** (cont'd)

| Short Name | Description | Offset Addr.[1)] | Access Mode | | Description See |
|---|---|---|---|---|---|
| | | | Read | Write | |
| CC83DC1R | Channel 1 Dead Time Counter Values | $0450_H$ | U, PV | U, PV | Page 20-158 |
| CC83DC2R | Channel 2 Dead Time Counter Values | $0454_H$ | U, PV | U, PV | Page 20-159 |
| CC83TIMER | Timer Current Value | $0470_H$ | U, PV | U, PV | Page 20-159 |
| CC83C0V | Capture Register 0 Value | $0474_H$ | U, PV | BE | Page 20-160 |
| CC83C1V | Capture Register 1 Value | $0478_H$ | U, PV | BE | Page 20-161 |
| CC83C2V | Capture Register 2 Value | $047C_H$ | U, PV | BE | Page 20-162 |
| CC83C3V | Capture Register 3 Value | $0480_H$ | U, PV | BE | Page 20-163 |
| CC83INTS | Interrupt Status | $04A0_H$ | U, PV | BE | Page 20-164 |
| CC83INTE | Interrupt Enable | $04A4_H$ | U, PV | U, PV | Page 20-166 |
| CC83SRS | Interrupt Configuration | $04A8_H$ | U, PV | U, PV | Page 20-168 |
| CC83SWS | Interrupt Status Set | $04AC_H$ | U, PV | U, PV | Page 20-170 |
| CC83SWR | Interrupt Status Clear | $04B0_H$ | U, PV | U, PV | Page 20-172 |
| CC83STC | Shadow Transfer Control | $04B4_H$ | U, PV | U, PV | Page 20-174 |
| CC83ECRD0 | Extended Read Back 0 | $04B8_H$ | U, PV | BE | Page 20-175 |
| CC83ECRD1 | Extended Read Back 1 | $04BC_H$ | U, PV | BE | Page 20-176 |

1) The absolute register address is calculated as follows:
Module Base Address + Offset Address (shown in this column)

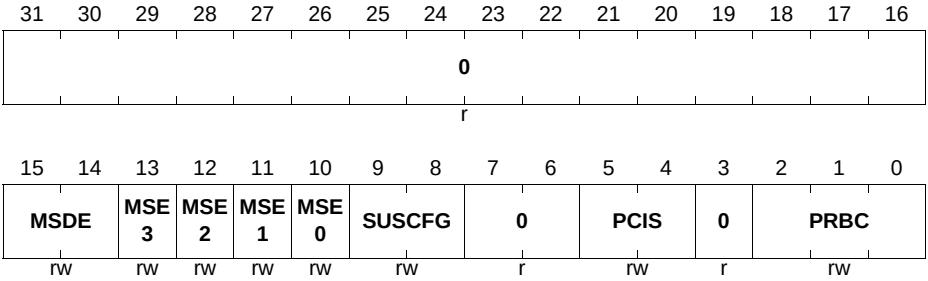
## 20.7.1 Global Registers

### GCTRL

The register contains the global configuration fields that affect all the timer slices inside CCU8.


Subject to Agreement on the Use of Product Information

**GCTRL**
**Global Control Register** **($0000_H$)** **Reset Value: $00000000_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| PRBC | [2:0] | rw | **Prescaler Clear Configuration**<br>This register controls how the prescaler Run Bit and internal registers are cleared.<br>$000_B$ SW only<br>$001_B$ **GSTAT**.PRB and prescaler registers are cleared when the Run Bit of CC80 is cleared.<br>$010_B$ **GSTAT**.PRB and prescaler registers are cleared when the Run Bit of CC81 is cleared.<br>$011_B$ **GSTAT**.PRB and prescaler registers are cleared when the Run Bit of CC82 is cleared.<br>$100_B$ **GSTAT**.PRB and prescaler registers are cleared when the Run Bit of CC83 is cleared. |
| PCIS | [5:4] | rw | **Prescaler Input Clock Selection**<br>$00_B$ Module clock<br>$01_B$ CCU8x.ECLKA<br>$10_B$ CCU8x.ECLKB<br>$11_B$ CCU8x.ECLKC |

| Field | Bits | Type | Description |
|---|---|---|---|
| SUSCFG | [9:8] | rw | **Suspend Mode Configuration**<br>This field controls the entering in suspend mode for all the CCU8 slices.<br><br>$00_B$ Suspend request ignored. The module never enters in suspend<br>$01_B$ Stops all the running slices immediately. Safe stop is not applied.<br>$10_B$ Stops the block immediately and clamps all the outputs to PASSIVE state. Safe stop is applied.<br>$11_B$ Waits for the roll over of each slice to stop and clamp the slices outputs. Safe stop is applied. |
| MSE0 | 10 | rw | **Slice 0 Multi Channel shadow transfer enable**<br>When this field is set, a shadow transfer of slice 0 can be requested not only by SW but also via the CCU8x.MCSS input.<br><br>$0_B$ Shadow transfer can only be requested by SW<br>$1_B$ Shadow transfer can be requested via SW and via the CCU8x.MCSS input. |
| MSE1 | 11 | rw | **Slice 1 Multi Channel shadow transfer enable**<br>When this field is set, a shadow transfer of slice 1 can be requested not only by SW but also via the CCU8x.MCSS input.<br><br>$0_B$ Shadow transfer can only be requested by SW<br>$1_B$ Shadow transfer can be requested via SW and via the CCU8x.MCSS input. |
| MSE2 | 12 | rw | **Slice 2 Multi Channel shadow transfer enable**<br>When this field is set, a shadow transfer of slice 2 can be requested not only by SW but also via the CCU8x.MCSS input.<br><br>$0_B$ Shadow transfer can only be requested by SW<br>$1_B$ Shadow transfer can be requested via SW and via the CCU8xMCSS input. |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| MSE3 | 13 | rw | **Slice 3 Multi Channel shadow transfer enable**<br>When this field is set, a shadow transfer of slice 3 can be requested not only by SW but also via the CCU8x.MCSS input.<br><br>$0_B$ Shadow transfer can only be requested by SW<br>$1_B$ Shadow transfer can be requested via SW and via the CCU8x.MCSS input. |
| MSDE | [15:14] | rw | **Multi Channel shadow transfer request configuration**<br>This field configures the type of shadow transfer requested via the CCU8x.MCSS input. The field **CC8yTC**.MSEx needs to be set in order for this configuration to have any effect.<br><br>$00_B$ Only the shadow transfer for period and compare values is requested<br>$01_B$ Shadow transfer for the compare, period and prescaler compare values is requested<br>$10_B$ Reserved<br>$11_B$ Shadow transfer for the compare, period, prescaler and dither compare values is requested |
| 0 | 3, [7:6], [31:16] | r | **Reserved**<br>A read always returns 0. |

### GSTAT

The register contains the status of the prescaler and each timer slice (idle mode or running).


Subject to Agreement on the Use of Product Information

## GSTAT
**Global Status Register** **($0004_H$)** **Reset Value: $0000000F_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | PCRB | 0 | PRB | 0 | | | | S3I | S2I | S1I | S0I |
| r | | | | | rh | r | rh | r | | | | rh | rh | rh | rh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **S0I** | 0 | rh | **CC80 IDLE status**<br>This bit indicates if the CC80 slice is in IDLE mode or not. In IDLE mode the clocks for the CC80 slice are stopped.<br>$0_B$ Running<br>$1_B$ Idle |
| **S1I** | 1 | rh | **CC81 IDLE status**<br>This bit indicates if the CC81 slice is in IDLE mode or not. In IDLE mode the clocks for the CC81 slice are stopped.<br>$0_B$ Running<br>$1_B$ Idle |
| **S2I** | 2 | rh | **CC82 IDLE status**<br>This bit indicates if the CC82 slice is in IDLE mode or not. In IDLE mode the clocks for the CC82 slice are stopped.<br>$0_B$ Running<br>$1_B$ Idle |
| **S3I** | 3 | rh | **CC83 IDLE status**<br>This bit indicates if the CC83 slice is in IDLE mode or not. In IDLE mode the clocks for the CC83 slice are stopped.<br>$0_B$ Running<br>$1_B$ Idle |

| Field | Bits | Type | Description |
|---|---|---|---|
| PRB | 8 | rh | **Prescaler Run Bit**<br>$0_B$ Prescaler is stopped<br>$1_B$ Prescaler is running |
| PCRB | 10 | rh | **Parity Checker Run Bit**<br>$0_B$ Parity Checker is stopped<br>$1_B$ Parity Checker is running |
| 0 | [7:4], 9, [31:11] | r | **Reserved**<br>Read always returns 0. |

### GIDLS

Through this register one can set the prescaler and the specific timer slices into idle mode.

**GIDLS**
**Global Idle Set** **($0008_H$)** **Reset Value: $00000000_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | CPCH | PSIC | CPRB | 0 | | | | SS3I | SS2I | SS1I | SS0I |
| r | | | | | w | w | w | r | | | | w | w | w | w |

| Field | Bits | Type | Description |
|---|---|---|---|
| **SS0I** | 0 | w | **CC80 IDLE mode set**<br>Writing a $1_B$ to this bit sets the CC80 slice in IDLE mode. The clocks for the slice are stopped when in IDLE mode. When entering IDLE, the internal slice registers are not cleared.<br>A read access always returns 0. |


Subject to Agreement on the Use of Product Information

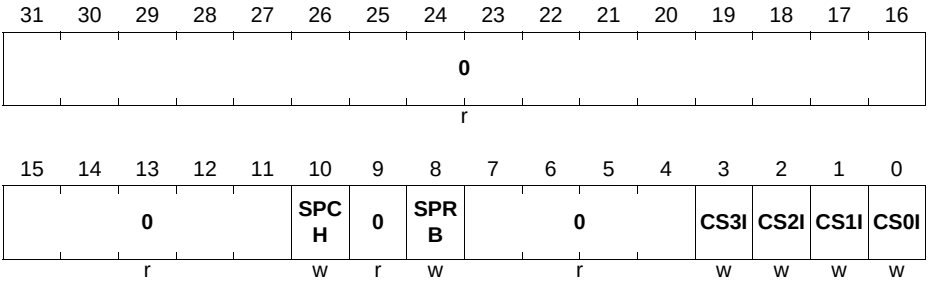
| Field | Bits | Type | Description |
|---|---|---|---|
| **SS1I** | 1 | w | **CC81 IDLE mode set**<br>Writing a $1_B$ to this bit sets the CC81 slice in IDLE mode. The clocks for the slice are stopped when in IDLE mode. When entering IDLE, the internal slice registers are not cleared.<br>A read access always returns 0. |
| **SS2I** | 2 | w | **CC82 IDLE mode set**<br>Writing a $1_B$ to this bit sets the CC82 slice in IDLE mode. The clocks for the slice are stopped when in IDLE mode. When entering IDLE, the internal slice registers are not cleared.<br>A read access always returns 0. |
| **SS3I** | 3 | w | **CC83 IDLE mode set**<br>Writing a $1_B$ to this bit sets the CC83 slice in IDLE mode. The clocks for the slice are stopped when in IDLE mode. When entering IDLE, the internal slice registers are not cleared.<br>A read access always returns 0. |
| CPRB | 8 | w | **Prescaler$_B$ Run Bit Clear**<br>Writing a 1 into this register clears the Run Bit of the prescaler. Prescaler internal registers are not cleared.<br>A read always returns 0. |
| PSIC | 9 | w | **Prescaler clear**<br>Writing a $1_B$ to this register clears the prescaler counter. It also loads the PSIV into the PVAL field for all Timer Slices.<br>This performs a re alignment of the timer clock for all Slices.<br>The Run Bit of the prescaler is not cleared.<br>A read always returns 0. |
| CPCH | 10 | w | **Parity Checker Run bit clear**<br>Writing a $1_B$ to this register clears the run bit of the parity checker. All the internal registers are cleared. The status bit value is kept, **GPCHK**.PCST.<br>A read always returns 0. |
| 0 | [7:4], [31:11] | r | **Reserved**<br>Read always returns 0. |


Subject to Agreement on the Use of Product Information

### GIDLC

Through this register one can remove the prescaler and the specific timer slices from idle mode.

**GIDLC**
**Global Idle Clear** **($000C_H$)** **Reset Value: $00000000_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **CS0I** | 0 | w | **CC80 IDLE mode clear**<br>Writing a $1_B$ to this bit removes the CC80 from IDLE mode. No clear to the internal slice register is done. A read access always returns 0. |
| **CS1I** | 1 | w | **CC81 IDLE mode clear**<br>Writing a $1_B$ to this bit removes the CC81 from IDLE mode. No clear to the internal slice register is done. A read access always returns 0. |
| **CS2I** | 2 | w | **CC82 IDLE mode clear**<br>Writing a $1_B$ to this bit removes the CC82 from IDLE mode. No clear to the internal slice register is done. A read access always returns 0. |
| **CS3I** | 3 | w | **CC83 IDLE mode clear**<br>Writing a $1_B$ to this bit removes the CC83 from IDLE mode. No clear to the internal slice register is done. A read access always returns 0. |
| SPRB | 8 | w | **Prescaler Run Bit Set**<br>Writing a $1_B$ into this register sets the Run Bit of the prescaler. Prescaler internal registers are not cleared. A read always returns 0. |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| SPCH | 10 | w | **Parity Checker run bit set**<br>Writing a $1_B$ into this register sets the Run Bit of the parity checker. A read always returns 0. |
| 0 | [7:4], 9, [31:11] | r | **Reserved**<br>Read always returns 0. |

### GCSS

Through this register one can request a shadow transfer for the specific timer slice(s) and set the status bit for each of the compare channels.

**GCSS**
**Global Channel Set** **($0010_H$)** **Reset Value: $00000000_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | | | | | | | | **S3S T2S** | **S2S T2S** | **S1S T2S** | **S0S T2S** | **S3S T1S** | **S2S T1S** | **S1S T1S** | **S0S T1S** |
| r | | | | | | | | w | w | w | w | w | w | w | w |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **S3P SE** | **S3D SE** | **S3S E** | **0** | **S2P SE** | **S2D SE** | **S2S E** | **0** | **S1P SE** | **S1D SE** | **S1S E** | **0** | **S0P SE** | **S0D SE** | **S0S E** |
| r | w | w | w | r | w | w | w | r | w | w | w | r | w | w | w |

| Field | Bits | Type | Description |
|---|---|---|---|
| **S0SE** | 0 | w | **Slice 0 shadow transfer set enable**<br>Writing a $1_B$ to this bit will set the **GCST**.S0SS field, enabling then a shadow transfer for the Period, Compare and Passive level values.<br>A read always returns 0. |
| **S0DSE** | 1 | w | **Slice 0 Dither shadow transfer set enable**<br>Writing a $1_B$ to this bit will set the **GCST**.S0DSS field, enabling then a shadow transfer for the Dither compare value.<br>A read always returns 0. |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **S0PSE** | 2 | w | **Slice 0 Prescaler shadow transfer set enable**<br>Writing a $1_B$ to this bit will set the **GCST**.S0PSS field, enabling then a shadow transfer for the prescaler compare value.<br>A read always returns 0. |
| **S1SE** | 4 | w | **Slice 1 shadow transfer set enable**<br>Writing a $1_B$ to this bit will set the **GCST**.S1SS field, enabling then a shadow transfer for the Period, Compare and Passive level values.<br>A read always returns 0. |
| **S1DSE** | 5 | w | **Slice 1 Dither shadow transfer set enable**<br>Writing a $1_B$ to this bit will set the **GCST**.S1DSS field, enabling then a shadow transfer for the Dither compare value.<br>A read always returns 0. |
| **S1PSE** | 6 | w | **Slice 1 Prescaler shadow transfer set enable**<br>Writing a $1_B$ to this bit will set the **GCST**.S1PSS field, enabling then a shadow transfer for the prescaler compare value.<br>A read always returns 0. |
| **S2SE** | 8 | w | **Slice 2 shadow transfer set enable**<br>Writing a $1_B$ to this bit will set the **GCST**.S2SS field, enabling then a shadow transfer for the Period, Compare and Passive level values.<br>A read always returns 0. |
| **S2DSE** | 9 | w | **Slice 2 Dither shadow transfer set enable**<br>Writing a $1_B$ to this bit will set the **GCST**.S2DSS field, enabling then a shadow transfer for the Dither compare value.<br>A read always returns 0. |
| **S2PSE** | 10 | w | **Slice 2 Prescaler shadow transfer set enable**<br>Writing a $1_B$ to this bit will set the **GCST**.S2PSS field, enabling then a shadow transfer for the prescaler compare value.<br>A read always returns 0. |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **S3SE** | 12 | w | **Slice 3 shadow transfer set enable**<br>Writing a $1_B$ to this bit will set the **GCST**.S3SS field, enabling then a shadow transfer for the Period, Compare and Passive level values.<br>A read always returns 0. |
| **S3DSE** | 13 | w | **Slice 3 Dither shadow transfer set enable**<br>Writing a $1_B$ to this bit will set the **GCST**.S3DSS field, enabling then a shadow transfer for the Dither compare value.<br>A read always returns 0. |
| **S3PSE** | 14 | w | **Slice 3 Prescaler shadow transfer set enable**<br>Writing a $1_B$ to this bit will set the **GCST**.S3PSS field, enabling then a shadow transfer for the prescaler compare value.<br>A read always returns 0. |
| S0ST1S | 16 | w | **Slice 0 status bit 1 set**<br>Writing a $1_B$ into this register sets the compare channel 1 status bit of slice 0 (**GCST**.CC80ST1).<br>A read always returns 0. |
| S1ST1S | 17 | w | **Slice 1 status bit 1 set**<br>Writing a $1_B$ into this register sets the compare channel 1 status bit of slice 1 (**GCST**.CC81ST1).<br>A read always returns 0. |
| S2ST1S | 18 | w | **Slice 2 status bit 1 set**<br>Writing a $1_B$ into this register sets the compare channel 1 status bit of slice 2 (**GCST**.CC82ST1).<br>A read always returns 0. |
| S3ST1S | 19 | w | **Slice 3 status bit 1 set**<br>Writing a $1_B$ into this register sets the compare channel 2 status bit of slice 3 (**GCST**.CC83ST1).<br>A read always returns 0. |
| S0ST2S | 20 | w | **Slice 0 status bit 2 set**<br>Writing a $1_B$ into this register sets the compare channel 2 status bit of slice 0 (**GCST**.CC80ST2).<br>A read always returns 0. |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| S1ST2S | 21 | w | **Slice 1 status bit 2 set**<br>Writing a $1_B$ into this register sets the compare channel 2 status bit of slice 1 (**GCST**.CC81ST2).<br>A read always returns 0. |
| S2ST2S | 22 | w | **Slice 2 status bit 2 set**<br>Writing a $1_B$ into this register sets the compare channel 2 status bit of slice 2 (**GCST**.CC82ST2).<br>A read always returns 0. |
| S3ST2S | 23 | w | **Slice 3 status bit 2 set**<br>Writing a $1_B$ into this register sets the compare channel 2 status bit of slice 3 (**GCST**.CC83ST2).<br>A read always returns 0. |
| 0 | 3, 7, 11, 15, [31:24] | r | **Reserved**<br>Read always returns 0. |

### GCSC

Through this register one can reset a shadow transfer request for the specific timer slice and clear the status bit for each the compare channels.

**GCSC**
**Global Channel Clear** **($0014_H$)** **Reset Value: $00000000_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | | | | | | | | **S3ST2C** | **S2ST2C** | **S1ST2C** | **S0ST2C** | **S3ST1C** | **S2ST1C** | **S1ST1C** | **S0ST1C** |
| r | | | | | | | | w | w | w | w | w | w | w | w |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **S3PSC** | **S3DSC** | **S3SC** | **0** | **S2PSC** | **S2DSC** | **S2SC** | **0** | **S1PSC** | **S1DSC** | **S1SC** | **0** | **S0PSC** | **S0DSC** | **S0SC** |
| r | w | w | w | r | w | w | w | r | w | w | w | r | w | w | w |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **S0SC** | 0 | w | **Slice 0 shadow transfer request clear**<br>Writing a $1_B$ to this bit will clear the **GCST**.S0SS field, canceling any pending shadow transfer for the Period, Compare and Passive level values.<br>A read always returns 0. |
| **S0DSC** | 1 | w | **Slice 0 Dither shadow transfer clear**<br>Writing a $1_B$ to this bit will clear the **GCST**.S0DSS field, canceling any pending shadow transfer for the Dither compare value.<br>A read always returns 0. |
| **S0PSC** | 2 | w | **Slice 0 Prescaler shadow transfer clear**<br>Writing a $1_B$ to this bit will clear the **GCST**.S0PSS field, canceling any pending shadow transfer for the prescaler compare value.<br>A read always returns 0. |
| **S1SC** | 4 | w | **Slice 1 shadow transfer clear**<br>Writing a $1_B$ to this bit will clear the **GCST**.S1SS field, canceling any pending shadow transfer for the Period, Compare and Passive level values.<br>A read always returns 0. |
| **S1DSC** | 5 | w | **Slice 1 Dither shadow transfer clear**<br>Writing a $1_B$ to this bit will clear the **GCST**.S1DSS field, canceling any pending shadow transfer for the Dither compare value.<br>A read always returns 0. |
| **S1PSC** | 6 | w | **Slice 1 Prescaler shadow transfer clear**<br>Writing a $1_B$ to this bit will clear the **GCST**.S1PSS field, canceling any pending shadow transfer for the prescaler compare value.<br>A read always returns 0. |
| **S2SC** | 8 | w | **Slice 2 shadow transfer clear**<br>Writing a $1_B$ to this bit will clear the **GCST**.S2SS field, canceling any pending shadow transfer for the Period, Compare and Passive level values.<br>A read always returns 0. |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **S2DSC** | 9 | w | **Slice 2 Dither shadow transfer clear**<br>Writing a $1_B$ to this bit will clear the **GCST**.S2DSS field, canceling any pending shadow transfer for the Dither compare value.<br>A read always returns 0. |
| **S2PSC** | 10 | w | **Slice 2 Prescaler shadow transfer clear**<br>Writing a $1_B$ to this bit will clear the **GCST**.S2PSS field, canceling any pending shadow transfer for the prescaler compare value.<br>A read always returns 0. |
| **S3SC** | 12 | w | **Slice 3 shadow transfer clear**<br>Writing a $1_B$ to this bit will clear the **GCST**.S3SS field, canceling any pending shadow transfer for the Period, Compare and Passive level values.<br>A read always returns 0. |
| **S3DSC** | 13 | w | **Slice 3 Dither shadow transfer clear**<br>Writing a $1_B$ to this bit will clear the **GCST**.S3DSS field, canceling any pending shadow transfer for the Dither compare value.<br>A read always returns 0. |
| **S3PSC** | 14 | w | **Slice 3 Prescaler shadow transfer clear**<br>Writing a $1_B$ to this bit will clear the **GCST**.S3PSS field, canceling any pending shadow transfer for the prescaler compare value.<br>A read always returns 0. |
| S0ST1C | 16 | w | **Slice 0 status bit 1 clear**<br>Writing a $1_B$ into this register clears the compare channel 1 status bit of slice 0 (**GCST**.CC80ST1).<br>A read always returns 0. |
| S1ST1C | 17 | w | **Slice 1 status bit 1 clear**<br>Writing a $1_B$ into this register clears the compare channel 1 status bit of slice 1 (**GCST**.CC81ST1).<br>A read always returns 0. |
| S2ST1C | 18 | w | **Slice 2 status bit 1 clear**<br>Writing a $1_B$ into this register clears the compare channel 1 status bit of slice 2 (**GCST**.CC82ST1).<br>A read always returns 0. |

| Field | Bits | Type | Description |
|---|---|---|---|
| S3ST1C | 19 | w | **Slice 3 status bit 1 clear**<br>Writing a $1_B$ into this register clears the compare channel 1 status bit of slice 3 (**GCST**.CC83ST1).<br>A read always returns 0. |
| S0ST2C | 20 | w | **Slice 0 status bit 2 clear**<br>Writing a $1_B$ into this register clears the compare channel 2 status bit of slice 0 (**GCST**.CC80ST2).<br>A read always returns 0. |
| S1ST2C | 21 | w | **Slice 1 status bit 2 clear**<br>Writing a $1_B$ into this register clears the compare channel 2 status bit of slice 1 (**GCST**.CC81ST2).<br>A read always returns 0. |
| S2ST2C | 22 | w | **Slice 2 status bit 2 clear**<br>Writing a $1_B$ into this register clears the compare channel 2 status bit of slice 2 (**GCST**.CC82ST2).<br>A read always returns 0. |
| S3ST2C | 23 | w | **Slice 3 status bit 2 clear**<br>Writing a $1_B$ into this register clears the compare channel 2 status bit of slice 3 (**GCST**.CC83ST2).<br>A read always returns 0. |
| 0 | 3, 7, 11, 15, [31:24] | r | **Reserved**<br>Read always returns 0. |

**GCST**

This register holds the information of the shadow transfer requests and of each timer slice status bit.


Subject to Agreement on the Use of Product Information

**GCST**
**Global Channel status** **($0018_H$)** **Reset Value: $00000000_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | CC83ST2 | CC82ST2 | CC81ST2 | CC80ST2 | CC83ST1 | CC82ST1 | CC81ST1 | CC80ST1 |
| r | | | | | | | | rh | rh | rh | rh | rh | rh | rh | rh |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | S3PSS | S3DSS | S3SS | 0 | S2PSS | S2DSS | S2SS | 0 | S1PSS | S1DSS | S1SS | 0 | S0PSS | S0DSS | S0SS |
| r | rh | rh | rh | r | rh | rh | rh | r | rh | rh | rh | r | rh | rh | rh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **S0SS** | 0 | rh | **Slice 0 shadow transfer status**<br>$0_B$ Shadow transfer has not been requested<br>$1_B$ Shadow transfer has been requested<br>This field is cleared by HW after the requested shadow transfer has been executed. |
| **S0DSS** | 1 | rh | **Slice 0 Dither shadow transfer status**<br>$0_B$ Dither shadow transfer has not been requested<br>$1_B$ Dither shadow transfer has been requested<br>This field is cleared by HW after the requested shadow transfer has been executed. |
| **S0PSS** | 2 | rh | **Slice 0 Prescaler shadow transfer status**<br>$0_B$ Prescaler shadow transfer has not been requested<br>$1_B$ Prescaler shadow transfer has been requested<br>This field is cleared by HW after the requested shadow transfer has been executed. |
| **S1SS** | 4 | rh | **Slice 1 shadow transfer status**<br>$0_B$ Shadow transfer has not been requested<br>$1_B$ Shadow transfer has been requested<br>This field is cleared by HW after the requested shadow transfer has been executed. |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **S1DSS** | 5 | rh | **Slice 1 Dither shadow transfer status**<br>$0_B$ Dither shadow transfer has not been requested<br>$1_B$ Dither shadow transfer has been requested<br>This field is cleared by HW after the requested shadow transfer has been executed. |
| **S1PSS** | 6 | rh | **Slice 1 Prescaler shadow transfer status**<br>$0_B$ Prescaler shadow transfer has not been requested<br>$1_B$ Prescaler shadow transfer has been requested<br>This field is cleared by HW after the requested shadow transfer has been executed. |
| **S2SS** | 8 | rh | **Slice 2 shadow transfer status**<br>$0_B$ Shadow transfer has not been requested<br>$1_B$ Shadow transfer has been requested<br>This field is cleared by HW after the requested shadow transfer has been executed. |
| **S2DSS** | 9 | rh | **Slice 2 Dither shadow transfer status**<br>$0_B$ Dither shadow transfer has not been requested<br>$1_B$ Dither shadow transfer has been requested<br>This field is cleared by HW after the requested shadow transfer has been executed. |
| **S2PSS** | 10 | rh | **Slice 2 Prescaler shadow transfer status**<br>$0_B$ Prescaler shadow transfer has not been requested<br>$1_B$ Prescaler shadow transfer has been requested<br>This field is cleared by HW after the requested shadow transfer has been executed. |
| **S3SS** | 12 | rh | **Slice 3 shadow transfer status**<br>$0_B$ Shadow transfer has not been requested<br>$1_B$ Shadow transfer has been requested<br>This field is cleared by HW after the requested shadow transfer has been executed. |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **S3DSS** | 13 | rh | **Slice 3 Dither shadow transfer status**<br>$0_B$ Dither shadow transfer has not been requested<br>$1_B$ Dither shadow transfer has been requested<br>This field is cleared by HW after the requested shadow transfer has been executed. |
| **S3PSS** | 14 | rh | **Slice 3 Prescaler shadow transfer status**<br>$0_B$ Prescaler shadow transfer has not been requested<br>$1_B$ Prescaler shadow transfer has been requested<br>This field is cleared by HW after the requested shadow transfer has been executed. |
| CC80ST1 | 16 | rh | **Slice 0 compare channel 1 status bit** |
| CC81ST1 | 17 | rh | **Slice 1 compare channel 1 status bit** |
| CC82ST1 | 18 | rh | **Slice 2 compare channel 1 status bit** |
| CC83ST1 | 19 | rh | **Slice 3 compare channel 1 status bit** |
| CC80ST2 | 20 | rh | **Slice 0 compare channel 2 status bit** |
| CC81ST2 | 21 | rh | **Slice 1 compare channel 2 status bit** |
| CC82ST2 | 22 | rh | **Slice 2 compare channel 2 status bit** |
| CC83ST2 | 23 | rh | **Slice 3 compare channel 2 status bit** |
| 0 | 3, 7, 11, 15, [31:24] | r | **Reserved**<br>Read always returns 0. |

### GPCHK

This register contains the configuration for the Parity Check function of CCU8.


Subject to Agreement on the Use of Product Information

**GPCHK**
**Parity Checker Configuration** **($001C_H$)** **Reset Value: $00000000_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PCSEL3 | | | | PCSEL2 | | | | PCSEL1 | | | | PCSEL0 | | | |
| rw | | | | rw | | | | rw | | | | rw | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PCST | 0 | | | | | | | PCTS | PCDS | | PISEL | | PACS | | PASE |
| rh | r | | | | | | | rw | rw | | rw | | rw | | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **PASE** | 0 | rw | **Parity Checker Automatic start/stop**<br>If this field is set, the parity checker run bit is automatically set, when the run bit of the selected slice is set and it is cleared when the run bit of the slice is cleared. The field PACS needs to be programmed accordingly. |
| **PACS** | [2:1] | rw | **Parity Checker Automatic start/stop selector**<br>This fields selects to which slice the automatic start/stop of the parity checker is associated:<br>$00_B$ CC80<br>$01_B$ CC81<br>$10_B$ CC82<br>$11_B$ CC83 |
| **PISEL** | [4:3] | rw | **Driver Input signal selector**<br>This fields selects which signal contains the driver parity information:<br>$00_B$ CC8x.GP01 - driver output is connected to event 1 of slice 0<br>$01_B$ CC8x.GP11 - drive output is connected to event 1 of slice 1<br>$10_B$ CC8x.GP21 - driver output is connected to event 1 of slice 2<br>$11_B$ CC8x.GP31 - driver output is connected to event 1 of slice 3 |

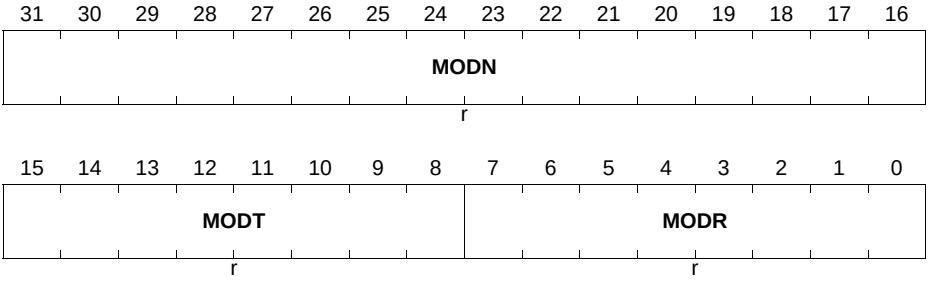
| Field | Bits | Type | Description |
|---|---|---|---|
| **PCDS** | [6:5] | rw | **Parity Checker Delay Input Selector**<br>This fields selects which signal is controlling the delay between the change at the CCU8 outputs and effective change at the driver parity output:<br>$00_B$ CCU8x.IGBTA<br>$01_B$ CCU8x.IGBTB<br>$10_B$ CCU8x.IGBTC<br>$11_B$ CCU8x.IGBTD |
| **PCTS** | 7 | rw | **Parity Checker type selector**<br>This fields selects if we have an odd or even parity:<br>$0_B$ Even parity enabled<br>$1_B$ Odd parity enabled |
| **PCST** | 15 | rh | **Parity Checker XOR status**<br>This field contains the current value of the XOR chain. |
| **PCSEL0** | [19:16] | rw | **Parity Checker Slice 0 output selection**<br>This fields selects which slice 0 outputs are going to be used to perform the parity check. The respective bit field needs to be set to $1_B$ to enable the output in the parity function.<br>PCSEL0[0] - CCU8x.OUT00<br>PCSEL0[1] - CCU8x.OUT01<br>PCSEL0[2] - CCU8x.OUT02<br>PCSEL0[3] - CCU8x.OUT03 |
| **PCSEL1** | [23:20] | rw | **Parity Checker Slice 1 output selection**<br>Same description as PCSEL0. |
| **PCSEL2** | [27:24] | rw | **Parity Checker Slice 2 output selection**<br>Same description as PCSEL0. |
| **PCSEL3** | [31:28] | rw | **Parity Checker Slice 3 output selection**<br>Same description as PCSEL0. |
| 0 | [14:8] | r | **Reserved**<br>Read always returns 0. |

### MIDR

This register contains the module identification number.


Subject to Agreement on the Use of Product Information

**MIDR**
**Module Identification** **($0080_H$)** **Reset Value: $00A7C0XX_H$**



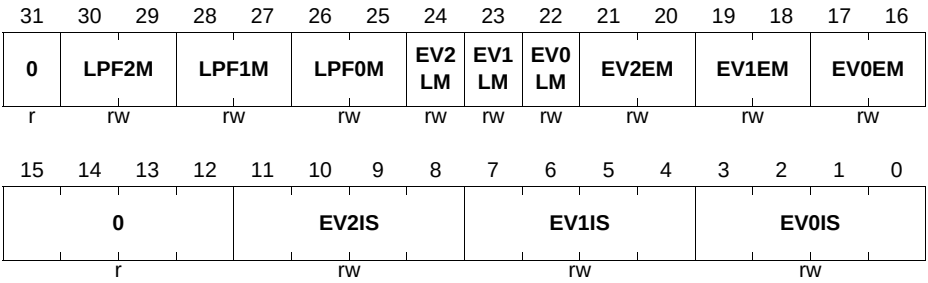

| Field | Bits | Type | Description |
|---|---|---|---|
| **MODR** | [7:0] | r | **Module Revision**<br>This bit field indicates the revision number of the module implementation (depending on the design step). The given value of $00_H$ is a placeholder for the actual number. |
| **MODT** | [15:8] | r | **Module Type** |
| **MODN** | [31:16] | r | **Module Number** |

## 20.7.2 Slice (CC8y) Registers

### CC8yINS

The register contains the configuration for the input selector.

**CC8yINS (y = 0 - 3)**
**Input Selector Configuration** **($0100_H$ + $0100_H$ * y)** **Reset Value: $00000000_H$**





Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **EV0IS** | [3:0] | rw | **Event 0 signal selection**<br>This field selects which pins is used for the event 0.<br>$0000_B$ CCU8x.INyA<br>$0001_B$ CCU8x.INyB<br>$0010_B$ CCU8x.INyC<br>$0011_B$ CCU8x.INyD<br>$0100_B$ CCU8x.INyE<br>$0101_B$ CCU8x.INyF<br>$0110_B$ CCU8x.INyG<br>$0111_B$ CCU8x.INyH<br>$1000_B$ CCU8x.INyI<br>$1001_B$ CCU8x.INyJ<br>$1010_B$ CCU8x.INyK<br>$1011_B$ CCU8x.INyL<br>$1100_B$ CCU8x.INyM<br>$1101_B$ CCU8x.INyN<br>$1110_B$ CCU8x.INyO<br>$1111_B$ CCU8x.INyP |
| **EV1IS** | [7:4] | rw | **Event 1 signal selection**<br>Same as EV0IS description |
| **EV2IS** | [11:8] | rw | **Event 2 signal selection**<br>Same as EV0IS description |
| EV0EM | [17:16] | rw | **Event 0 Edge Selection**<br>$00_B$ No action<br>$01_B$ Signal active on rising edge<br>$10_B$ Signal active on falling edge<br>$11_B$ Signal active on both edges |
| EV1EM | [19:18] | rw | **Event 1 Edge Selection**<br>Same as EV0EM description |
| EV2EM | [21:20] | rw | **Event 2 Edge Selection**<br>Same as EV0EM description |
| EV0LM | 22 | rw | **Event 0 Level Selection**<br>$0_B$ Active on HIGH level<br>$1_B$ Active on LOW level |
| EV1LM | 23 | rw | **Event 1 Level Selection**<br>Same as EV0LM description |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| EV2LM | 24 | rw | **Event 2 Level Selection**<br>Same as EV0LM description |
| LPF0M | [26:25] | rw | **Event 0 Low Pass Filter Configuration**<br>This field sets the number of consecutive counts for the Low Pass Filter of Event 0. The input signal value needs to remain stable for this number of counts ($f_{CCU8}$), so that a level/transition is accepted.<br>$00_B$ LPF is disabled<br>$01_B$ 3 clock cycles of $f_{CCU8}$<br>$10_B$ 5 clock cycles of $f_{CCU8}$<br>$11_B$ 7 clock cycles of $f_{CCU8}$ |
| LPF1M | [28:27] | rw | **Event 1 Low Pass Filter Configuration**<br>Same description as LPF0M |
| LPF2M | [30:29] | rw | **Event 2 Low Pass Filter Configuration**<br>Same description as LPF0M |
| 0 | [15:12], 31 | r | **Reserved**<br>Read always returns 0. |

### CC8yCMC

The register contains the configuration for the connection matrix.

**CC8yCMC (y = 0 - 3)**
**Connection Matrix Control** **($0104_H + 0100_H * y$)** **Reset Value: $00000000_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | | | | TCE | MOS | | TS | OFS |
| r | | | | | | | | | | | rw | rw | | rw | rw |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CNTS | | LDS | | UDS | | GATES | | CAP1S | | CAP0S | | ENDS | | STRTS | |
| rw | | rw | | rw | | rw | | rw | | rw | | rw | | rw | |

| Field | Bits | Type | Description |
|---|---|---|---|
| STRTS | [1:0] | rw | **External Start Functionality Selector**<br>Selects the Event that is going to be linked with the external start functionality.<br>$00_B$ External Start Function deactivated<br>$01_B$ External Start Function triggered by Event 0<br>$10_B$ External Start Function triggered by Event 1<br>$11_B$ External Start Function triggered by Event 2 |
| ENDS | [3:2] | rw | **External Stop Functionality Selector**<br>Selects the Event that is going to be linked with the external stop functionality.<br>$00_B$ External Stop Function deactivated<br>$01_B$ External Stop Function triggered by Event 0<br>$10_B$ External Stop Function triggered by Event 1<br>$11_B$ External Stop Function triggered by Event 2 |
| CAP0S | [5:4] | rw | **External Capture 0 Functionality Selector**<br>Selects the Event that is going to be linked with the external capture for capture registers number 1 and 0. This function is used to capture the value of the timer into the capture registers 1 and 0.<br>$00_B$ External Capture 0 Function deactivated<br>$01_B$ External Capture 0 Function triggered by Event 0<br>$10_B$ External Capture 0 Function triggered by Event 1<br>$11_B$ External Capture 0 Function triggered by Event 2<br>*Note: If the field SCE is set, this functionality is deactivated.* |

| Field | Bits | Type | Description |
|---|---|---|---|
| CAP1S | [7:6] | rw | **External Capture 1 Functionality Selector**<br>Selects the Event that is going to be linked with the external capture for capture registers number 3 and 2. This function is used to capture the value of the timer into the capture registers 3 and 2.<br>$00_B$ External Capture 1 Function deactivated<br>$01_B$ External Capture 1 Function triggered by Event 0<br>$10_B$ External Capture 1 Function triggered by Event 1<br>$11_B$ External Capture 1 Function triggered by Event 2 |
| GATES | [9:8] | rw | **External Gate Functionality Selector**<br>Selects the Event that is going to be linked with the counter gating function. This function is used to gate the timer increment/decrement procedure.<br>$00_B$ External Gating Function deactivated<br>$01_B$ External Gating Function triggered by Event 0<br>$10_B$ External Gating Function triggered by Event 1<br>$11_B$ External Gating Function triggered by Event 2 |
| UDS | [11:10] | rw | **External Up/Down Functionality Selector**<br>Selects the Event that is going to be linked with the Up/Down counting direction control. This function is used to control externally the timer increment/decrement operation.<br>$00_B$ External Up/Down Function deactivated<br>$01_B$ External Up/Down Function triggered by Event 0<br>$10_B$ External Up/Down Function triggered by Event 1<br>$11_B$ External Up/Down Function triggered by Event 2 |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| LDS | [13:12] | rw | **External Timer Load Functionality Selector**<br>Selects the Event that is going to be linked with the timer load function. The value present in the **CC8yCR1**/**CC8yCR2** (depending on the value of **CC8yTC**.TLS) is loaded into the specific slice timer.<br>$00_B$ - External Load Function deactivated<br>$01_B$ - External Load Function triggered by Event 0<br>$10_B$ - External Load Function triggered by Event 1<br>$11_B$ - External Load Function triggered by Event 2 |
| CNTS | [15:14] | rw | **External Count Selector**<br>Selects the Event that is going to be linked with the count function. The counter is going to be increment/decremented each time that a specific transition on the event is detected.<br>$00_B$ External Count Function deactivated<br>$01_B$ External Count Function triggered by Event 0<br>$10_B$ External Count Function triggered by Event 1<br>$11_B$ External Count Function triggered by Event 2<br>*Note: In CC40 this field doesn't exist. This is a read only reserved field. Read access always returns 0.* |
| OFS | 16 | rw | **Override Function Selector**<br>This field enables the ST bit override functionality.<br>$0_B$ Override functionality disabled<br>$1_B$ Status bit trigger override connected to Event 1; Status bit value override connected to Event 2 |
| TS | 17 | rw | **Trap Function Selector**<br>This field enables the trap functionality.<br>$0_B$ Trap function disabled<br>$1_B$ TRAP function connected to Event 2 |
| MOS | [19:18] | rw | **External Modulation Functionality Selector**<br>Selects the Event that is going to be linked with the external modulation function.<br>$00_B$ - Modulation Function deactivated<br>$01_B$ - Modulation Function triggered by Event 0<br>$10_B$ - Modulation Function triggered by Event 1<br>$11_B$ - Modulation Function triggered by Event 2 |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| TCE | 20 | rw | **Timer Concatenation Enable**<br>This bit enables the timer concatenation with the previous slice.<br>$0_B$ Timer concatenation is disabled<br>$1_B$ Timer concatenation is enabled<br>*Note: In CC80 this field doesn't exist. This is a read only reserved field. Read access always returns 0.* |
| 0 | [31:21] | r | **Reserved**<br>A read always returns 0 |

### CC8yTCST

The register holds the status of the timer (running/stopped) and the information about the counting direction (up/down).

**CC8yTCST (y = 0 - 3)**
**Slice Timer Status** **($0108_H + 0100_H * y$)** **Reset Value: $00000000_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | | | | DTR2 | DTR1 | 0 | CDIR | TRB |
| r | | | | | | | | | | | rh | rh | r | rh | rh |

| Field | Bits | Type | Description |
|---|---|---|---|
| TRB | 0 | rh | **Timer Run Bit**<br>This field indicates if the timer is running.<br>$0_B$ Timer is stopped<br>$1_B$ Timer is running |
| CDIR | 1 | rh | **Timer Counting Direction**<br>This filed indicates if the timer is being increment or decremented<br>$0_B$ Timer is counting up<br>$1_B$ Timer is counting down |

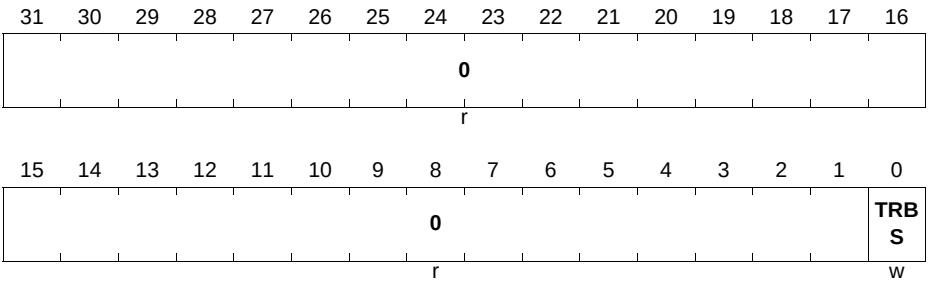

Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **DTR1** | 3 | rh | **Dead Time Counter 1 Run bit**<br>This field indicates if the dead time counter for linked with channel 1 is running.<br>$0_B$ Dead Time counter is idle<br>$1_B$ Dead Time counter is running |
| **DTR2** | 4 | rh | **Dead Time Counter 2 Run bit**<br>This field indicates if the dead time counter for linked with channel 2 is running.<br>$0_B$ Dead Time counter is idle<br>$1_B$ Dead Time counter is running |
| 0 | 2, [31:5] | r | **Reserved**<br>Read always returns 0 |

### CC8yTCSET

Through this register it is possible to start the timer.

**CC8yTCSET (y = 0 - 3)**
**Slice Timer Run Set** **($010C_H + 0100_H * y$)** **Reset Value: $00000000_H$**



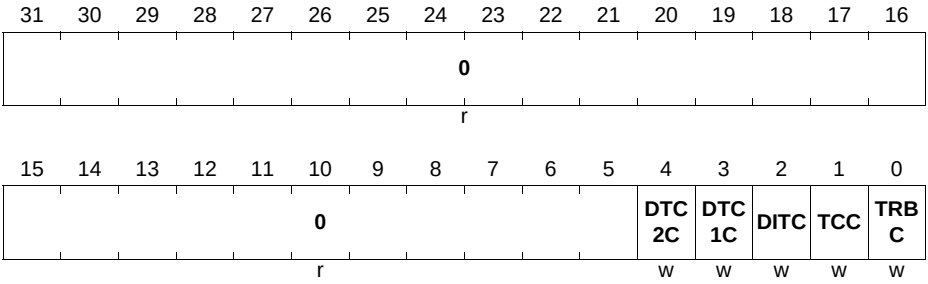

| Field | Bits | Type | Description |
|---|---|---|---|
| TRBS | 0 | w | **Timer Run Bit set**<br>Writing a $1_B$ into this field sets the run bit of the timer. The timer is not cleared.<br>Read always returns 0. |
| 0 | [31:1] | r | **Reserved**<br>Read always returns 0 |


Subject to Agreement on the Use of Product Information

### CC8yTCCLR

Through this register it is possible to stop and clear the timer, and clearing also the dither counter

**CC8yTCCLR (y = 0 - 3)**
**Slice Timer Clear** **($0110_H + 0100_H * y$)** **Reset Value: $00000000_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **TRBC** | 0 | w | **Timer Run Bit Clear**<br>Writing a $1_B$ into this field clears the run bit of the timer. The timer is not cleared.<br>Read always returns 0. |
| **TCC** | 1 | w | **Timer Clear**<br>Writing a $1_B$ into this field clears the timer value.<br>Read always returns 0. |
| **DITC** | 2 | w | **Dither Counter Clear**<br>Writing a $1_B$ into this field clears the dither counter.<br>Read always returns 0. |
| **DTC1C** | 3 | w | **Dead Time Counter 1 Clear**<br>Writing a $1_B$ into this field clears the channel 1 dead time counter. The counter is stopped until a new start trigger is detected.<br>Read always returns 0. |
| **DTC2C** | 4 | w | **Dead Time Counter 2 Clear**<br>Writing a $1_B$ into this field clears the channel 2 dead time counter. The counter is stopped until a new start trigger is detected.<br>Read always returns 0. |

| Field | Bits | Type | Description |
|---|---|---|---|
| 0 | [31:5] | r | **Reserved**<br>Read always returns 0 |

### CC8yTC

This register holds the several possible configurations for the timer operation.

**CC8yTC (y = 0 - 3)**
**Slice Timer Control** **($0114_H + 0100_H * y$)** **Reset Value: $18000000_H$**

| 31 | 30 29 | 28 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **STOS** | **EME** | **MCME2** | **MCME1** | **EMT** | **EMS** | **TRPSW** | **TRPSE** | **TRAPE3** | **TRAPE2** | **TRAPE1** | **TRAPE0** | **FPE** |
| r | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

| 15 | 14 13 | 12 | 11 | 10 | 9 8 | 7 | 6 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **DIM** | **DITHE** | **CCS** | **SCE** | **STRM** | **ENDM** | **TLS** | **CAPC** | **ECM** | **CMOD** | **CLST** | **TSSM** | **TCM** |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rh | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| TCM | 0 | rw | **Timer Counting Mode**<br>This field controls the actual counting scheme of the timer.<br>$0_B$ Edge aligned mode<br>$1_B$ Center aligned mode<br>*Note: When using an external signal to control the counting direction, the counting scheme is always edge aligned.* |
| TSSM | 1 | rw | **Timer Single Shot Mode**<br>This field controls the single shot mode. This is applicable in edge and center aligned modes.<br>$0_B$ Single shot mode is disabled<br>$1_B$ Single shot mode is enabled |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| CLST | 2 | rw | **Shadow Transfer on Clear**<br>Setting this bit to $1_B$ enables a shadow transfer when a timer clearing action is done (by SW or by an external event). Notice that the shadow transfer enable bitfields on the **GCST** register still need to be set to $1_B$ via software. |
| CMOD | 3 | rh | **Capture Compare Mode**<br>This field indicates in which mode the slice is operating. The default value is compare mode. The capture mode is automatically set by the HW when an external signal is mapped to a capture trigger.<br>$0_B$ Compare Mode<br>$1_B$ Capture Mode |
| ECM | 4 | rw | **Extended Capture Mode**<br>This field control the Capture mode of the specific slice. It only has effect if the CMOD bit is $1_B$.<br><br>$0_B$ Normal Capture Mode. Clear of the Full Flag of each capture register is done by accessing the registers individually only.<br>$1_B$ Extended Capture Mode. Clear of the Full Flag of each capture register is done not only by accessing the individual registers but also by accessing the ECRD register. When reading the ECRD register, only the capture register register full flag pointed by the VPTR is cleared |
| CAPC | [6:5] | rw | **Clear on Capture Control**<br>$00_B$ Timer is never cleared on a capture event<br>$01_B$ Timer is cleared on a capture event into capture registers 2 and 3. (When SCE = $1_B$, Timer is always cleared in a capture event)<br>$10_B$ Timer is cleared on a capture event into capture registers 0 and 1. (When SCE = $1_B$, Timer is always cleared in a capture event)<br>$11_B$ Timer is always cleared in a capture event. |
| TLS | 7 | rw | **Timer Load selector**<br>$0_B$ Timer is loaded with the value of CR1<br>$1_B$ Timer is loaded with the value of CR2 |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| ENDM | [9:8] | rw | **Extended Stop Function Control**<br>This field controls the extended functions of the external Stop signal.<br>$00_B$ Clears the timer run bit only (default stop)<br>$01_B$ Clears the timer only (flush)<br>$10_B$ Clears the timer and run bit (flush/stop)<br>$11_B$ Reserved<br>*Note: When using an external up/down signal the flush operation sets the timer with zero if the counter is counting up and with the Period value if the counter is being decremented.* |
| STRM | 10 | rw | **Extended Start Function Control**<br>This field controls the extended functions of the external Start signal.<br>$0_B$ Sets run bit only (default start)<br>$1_B$ Clears the timer and sets run bit, if not set (flush/start)<br>*Note: When using an external up/down signal the flush operation sets the timer with zero if the counter is being incremented and with the Period value if the counter is being decremented.* |
| SCE | 11 | rw | **Equal Capture Event enable**<br>$0_B$ Capture into **CC8yC0V**/**CC8yC1V** registers control by CCycapt0 and capture into **CC8yC3V**/**CC8yC2V** control by CCycapt1<br>$1_B$ Capture into **CC8yC0V**/**CC8yC1V** and **CC8yC3V**/**CC8yC2V** control by CCycapt1 |
| CCS | 12 | rw | **Continuous Capture Enable**<br>$0_B$ The capture into a specific capture register is done with the rules linked with the full flags, described at **Section 20.2.8.6**.<br>$1_B$ The capture into the capture registers is always done regardless of the full flag status (even if the register has not been read back). |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **DITHE** | [14:13] | rw | **Dither Enable**<br>This field controls the dither mode for the slice. See **Section 20.2.12**.<br>$00_B$ Dither is disabled<br>$01_B$ Dither is applied to the Period<br>$10_B$ Dither is applied to the Compare<br>$11_B$ Dither is applied to the Period and Compare |
| DIM | 15 | rw | **Dither input selector**<br>This fields selects if the dither control signal is connected to the dither logic of the specific slice of is connected to the dither logic of slice 0. Notice that even if this field is set to $1_B$, the field DITHE still needs to be programmed.<br>$0_B$ Slice is using it own dither unit<br>$1_B$ Slice is connected to the dither unit of slice 0. |
| **FPE** | 16 | rw | **Floating Prescaler enable**<br>Setting this bit to $1_B$ enables the floating prescaler mode.<br>$0_B$ Floating prescaler mode is disabled<br>$1_B$ Floating prescaler mode is enabled |
| TRAPE0 | 17 | rw | **TRAP enable for CCU8x.OUTy0**<br>Setting this bit to 1 enables the TRAP action at the CCU8x.OUTy0 output pin. After mapping an external signal to the TRAP functionality, the user must set this field to 1 to activate the effect of the TRAP on the specific output pin.<br>Writing a 0 into this field disables the effect of the TRAP function regardless of the state of the input signal.<br>$0_B$ TRAP functionality has no effect on the CCU8x.OUTy0 output<br>$1_B$ TRAP functionality affects the CCU8x.OUTy0 output |
| TRAPE1 | 18 | rw | **TRAP enable for CCU8x.OUTy1**<br>TRAP enable for the CCU8x.OUTy1. Same description as for the TRAPE0 field. |
| TRAPE2 | 19 | rw | **TRAP enable for CCU8x.OUTy2**<br>TRAP enable for the CCU8x.OUTy2. Same description as for the TRAPE0 field. |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| TRAPE3 | 20 | rw | **TRAP enable for CCU8x.OUTy3**<br>TRAP enable for the CCU8x.OUTy3. Same description as for the TRAPE0 field. |
| TRPSE | 21 | rw | **TRAP Synchronization Enable**<br>Writing a 1 into this bit enables a synchronous exiting with the PWM period of the trap state.<br>$0_B$ Exiting from TRAP state isn't synchronized with the PWM signal<br>$1_B$ Exiting from TRAP state is synchronized with the PWM signal |
| TRPSW | 22 | rw | **TRAP State Clear Control**<br>$0_B$ The slice exits the TRAP state automatically when the TRAP condition is not present (Trap state cleared by HW and SW)<br>$1_B$ The TRAP state can only be exited by a SW request. |
| EMS | 23 | rw | **External Modulation Synchronization**<br>Setting this bit to 1 enables the synchronization of the external modulation functionality with the PWM period.<br>$0_B$ External Modulation functionality is not synchronized with the PWM signal<br>$1_B$ External Modulation functionality is synchronized with the PWM signal |
| EMT | 24 | rw | **External Modulation Type**<br>This field selects if the external modulation event is clearing the CC8ySTx bits or if is gating the outputs.<br>$0_B$ External Modulation functionality is clearing the CC8ySTx bits.<br>$1_B$ External Modulation functionality is gating the outputs. |
| MCME1 | 25 | rw | **Multi Channel Mode Enable for Channel 1**<br>$0_B$ Multi Channel Mode in Channel 1 is disabled<br>$1_B$ Multi Channel Mode in Channel 1 is enabled |
| MCME2 | 26 | rw | **Multi Channel Mode Enable for Channel 2**<br>$0_B$ Multi Channel Mode in Channel 2 is disabled<br>$1_B$ Multi Channel Mode in Channel 2 is enabled |

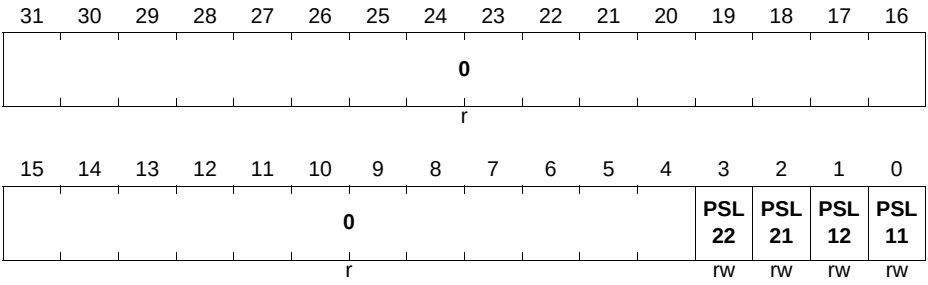

Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| EME | [28:27] | rw | **External Modulation Channel enable**<br>This field controls in which channel, the modulation functionality has effect. The modulations functionality needs to be previously enabled by setting the CC8yCMC.MOS accordingly.<br>$00_B$ External Modulation functionality doesn't affect any channel<br>$01_B$ External Modulation only applied on channel 1<br>$10_B$ External Modulation only applied on channel 2<br>$11_B$ External Modulation applied on both channels |
| STOS | [30:29] | rw | **Status bit output selector**<br>This field selects to which channel the output CC8ySTy is mapped.<br>$00_B$ CC8yST1 forward to CCU8x.STy<br>$01_B$ CC8yST2 forward to CCU8x.STy<br>$10_B$ CC8yST1 AND CC8yST2 forward to CCU8x.STy<br>$11_B$ CC8yST1 OR CC8yST2 forward to CCU8x.STy |
| 0 | 31 | r | **Reserved**<br>Read always returns 0. |

### CC8yPSL

This register holds the configuration for the output passive level control.

**CC8yPSL (y = 0 - 3)**
**Passive Level Config** **($0118_H$ + $0100_H$ * y)** **Reset Value: $00000000_H$**

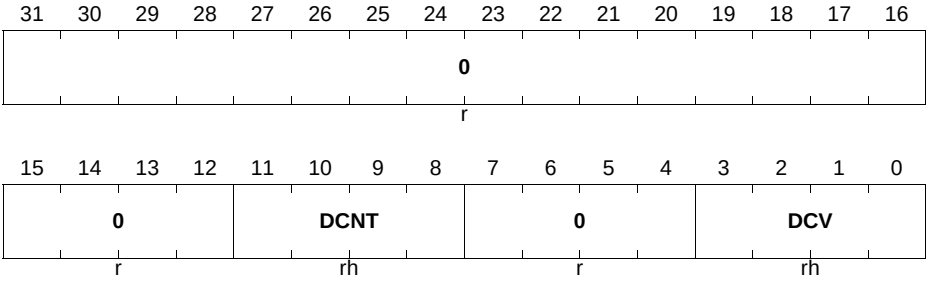


| Field | Bits | Type | Description |
|---|---|---|---|
| **PSL11** | 0 | rw | **Output Passive Level for CCU8x.OUTy0**<br>This field controls the passive level of the CCU8x.OUTy0.<br>$0_B$ Passive Level is LOW<br>$1_B$ Passive Level is HIGH<br>A write always addresses the shadow register, while a read always returns the current used value. |
| **PSL12** | 1 | rw | **Output Passive Level for CCU8x.OUTy1**<br>This field controls the passive level of the CCU8x.OUTy1.<br>$0_B$ Passive Level is LOW<br>$1_B$ Passive Level is HIGH<br>A write always addresses the shadow register, while a read always returns the current used value. |
| **PSL21** | 2 | rw | **Output Passive Level for CCU8x.OUTy2**<br>This field controls the passive level of the CCU8x.OUTy2.<br>$0_B$ Passive Level is LOW<br>$1_B$ Passive Level is HIGH<br>A write always addresses the shadow register, while a read always returns the current used value. |
| **PSL22** | 3 | rw | **Output Passive Level for CCU8x.OUTy3**<br>This field controls the passive level of the CCU8x.OUTy3.<br>$0_B$ Passive Level is LOW<br>$1_B$ Passive Level is HIGH<br>A write always addresses the shadow register, while a read always returns the current used value. |
| **0** | [31:4] | r | **Reserved**<br>A read access always returns 0 |

### CC8yDIT

This register holds the current dither compare and dither counter values.

**CC8yDIT (y = 0 - 3)**
**Dither Config** **($011C_H + 0100_H * y$)** **Reset Value: $00000000_H$**

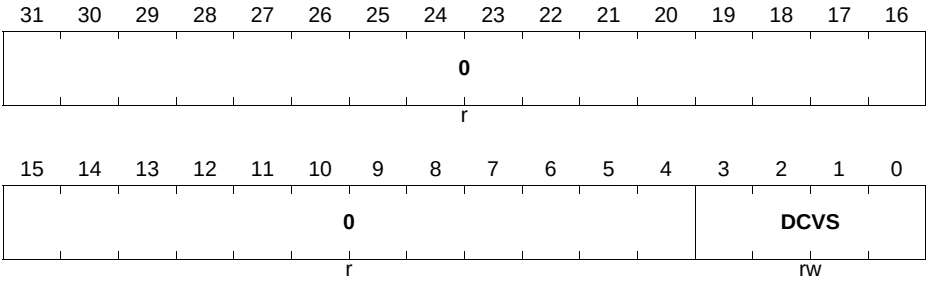



| Field | Bits | Type | Description |
|---|---|---|---|
| **DCV** | [3:0] | rh | **Dither compare Value**<br>This field contains the value used for the dither comparison.<br>This value is updated when a shadow transfer occurs with the **CC8yDITS**.DCVS. |
| DCNT | [11:8] | rh | **Dither counter actual value** |
| 0 | [7:4], [31:12] | r | **Reserved**<br>Read always returns 0. |

**CC8yDITS**

This register contains the value that is going to be loaded into the **CC8yDIT**.DCV when the next shadow transfer occurs.

**CC8yDITS (y = 0 - 3)**
**Dither Shadow Register** **($0120_H + 0100_H * y$)** **Reset Value: $00000000_H$**





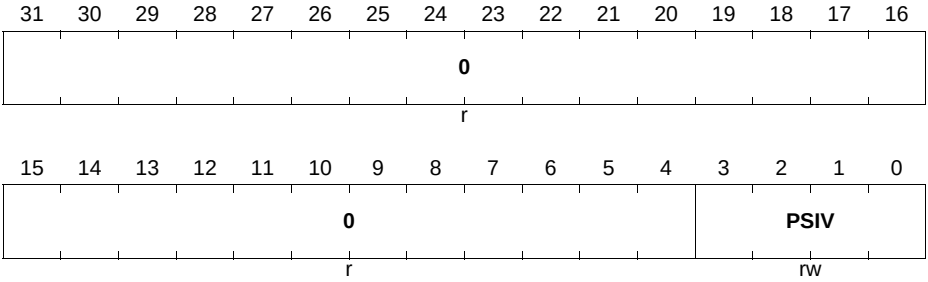
Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **DCVS** | [3:0] | rw | **Dither Shadow Compare Value**<br>This field contains the value that is going to be set on the dither compare value, **CC8yDIT**.DCV, within the next shadow transfer. |
| 0 | [31:4] | r | **Reserved**<br>Read always returns 0. |

### CC8yPSC

This register contains the value that is loaded into the prescaler during restart.

**CC8yPSC (y = 0 - 3)**
**Prescaler Control** **($0124_H$ + $0100_H$ * y)** **Reset Value: $00000000_H$**



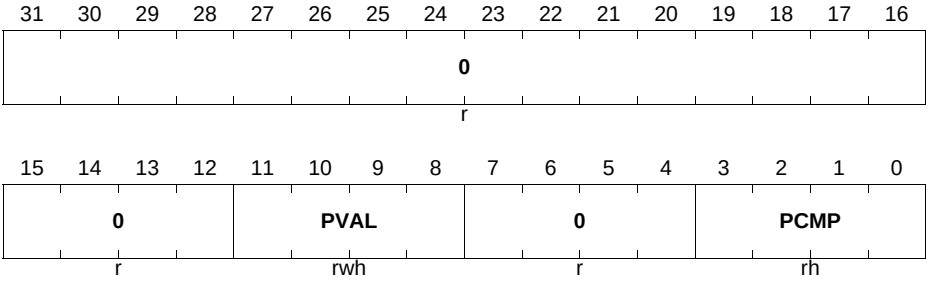

| Field | Bits | Type | Description |
|---|---|---|---|
| **PSIV** | [3:0] | rw | **Prescaler Initial Value**<br>This field contains the value that is applied to the Prescaler at startup.<br>When floating prescaler mode is used, this value is applied when a timer compare match AND prescaler compare match occurs or when a capture event is triggered. |
| 0 | [31:4] | r | **Reserved**<br>Read always returns 0. |

### CC8yFPC

This register contains the value used for the floating prescaler compare and the actual prescaler division value.


Subject to Agreement on the Use of Product Information

**CC8yFPC (y = 0 - 3)**
**Floating Prescaler Control** **($0128_H$ + $0100_H$ * y)** **Reset Value: $00000000_H$**

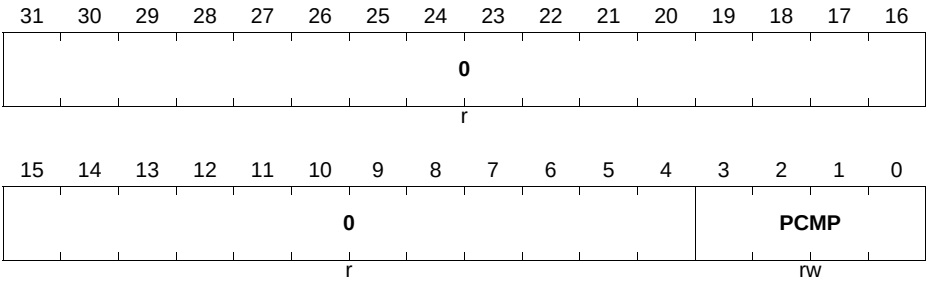



| Field | Bits | Type | Description |
|---|---|---|---|
| PCMP | [3:0] | rh | **Floating Prescaler Compare Value**<br>This field contains the value used to compare the actual prescaler value. The comparison is triggered by the Timer Compare match event. See **Section 20.2.13.2**. |
| PVAL | [11:8] | rwh | **Actual Prescaler Value**<br>See **Table 20-8**. Writing into this register is only possible when the prescaler is stopped. When the floating prescaler mode is not used, this value is equal to the **CC8yPSC**.PSIV. |
| 0 | [7:4], [15:12], [31:16] | r | **Reserved**<br>Read always returns 0. |

### CC8yFPCS

This register contains the value that is going to be transferred to the **CC8yFPC**.PCMP field within the next shadow transfer update.

**CC8yFPCS (y = 0 - 3)**
**Floating Prescaler Shadow** **($012C_H + 0100_H * y$)** **Reset Value: $00000000_H$**

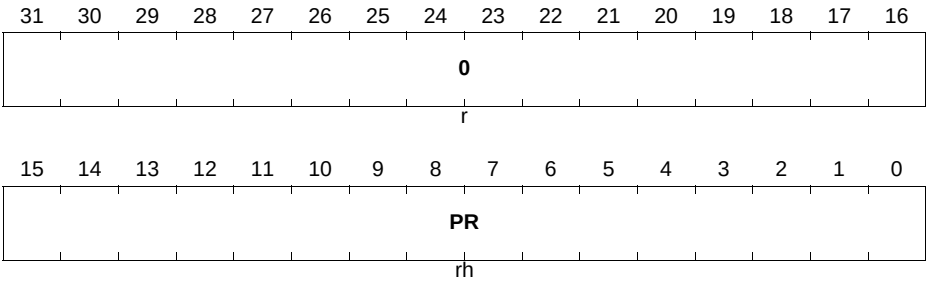



| Field | Bits | Type | Description |
|---|---|---|---|
| PCMP | [3:0] | rw | **Floating Prescaler Shadow Compare Value**<br>This field contains the value that is going to be set on the **CC8yFPC**.PCMP within the next shadow transfer. See **Table 20-8**. |
| 0 | [31:4] | r | **Reserved**<br>Read always returns 0. |

**CC8yPR**

This register contains the actual value for the timer period.

**CC8yPR (y = 0 - 3)**
**Timer Period Value** **($0130_H + 0100_H * y$)** **Reset Value: $00000000_H$**





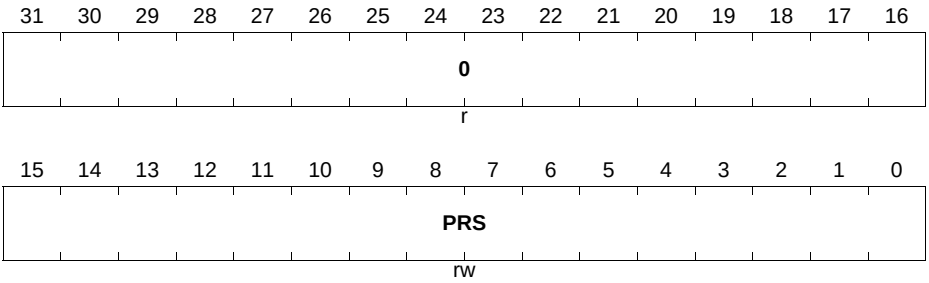
Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **PR** | [15:0] | rh | **Period Register**<br>Contains the value of the timer period. |
| **0** | [31:16] | r | **Reserved**<br>A read always returns 0. |

### CC8yPRS

This register contains the value for the timer period that is going to be transferred into the **CC8yPR**.PR field when the next shadow transfer occurs.

**CC8yPRS (y = 0 - 3)**
**Timer Shadow Period Value** **($0134_H$ + $0100_H$ * y)** **Reset Value: $00000000_H$**

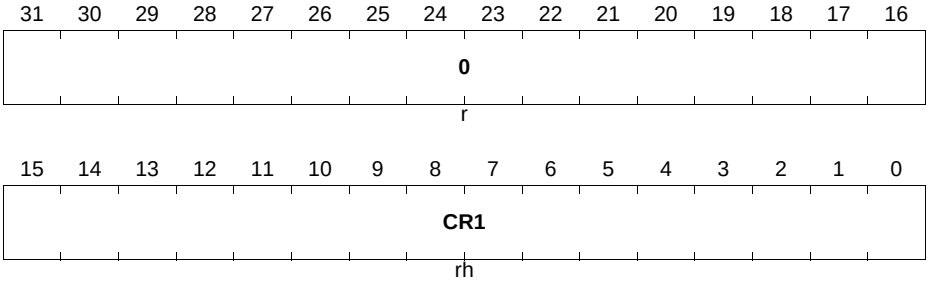



| Field | Bits | Type | Description |
|---|---|---|---|
| **PRS** | [15:0] | rw | **Period Register**<br>Contains the value of the timer period, that is going to be passed into the **CC8yPR**.PR field when the next shadow transfer occurs. |
| **0** | [31:16] | r | **Reserved**<br>A read always returns 0. |

### CC8yCR1

This register contains the value for the timer comparison of channel 1.

**CC8yCR1 (y = 0 - 3)**
**Channel 1 Compare Value** **($0138_H + 0100_H * y$)** **Reset Value: $00000000_H$**

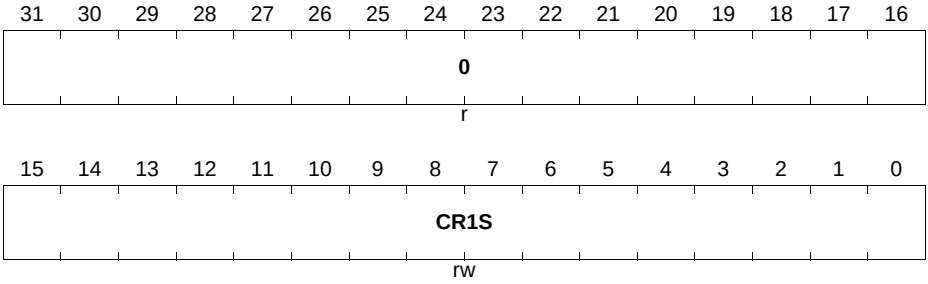



| Field | Bits | Type | Description |
|---|---|---|---|
| **CR1** | [15:0] | rh | **Compare Register for Channel 1**<br>Contains the value for the timer comparison.<br>A write always addresses the shadow register, while a read returns the actual value.<br>*Note: In Capture Mode when a external signal is selected for capturing the timer value into the capture registers 0 and 1, the CR is not accessible for writing. A read always returns 0.* |
| **0** | [31:16] | r | **Reserved**<br>A read always returns 0. |

### CC8yCR1S

This register contains the value that is going to be loaded into the **CC8yCR1**.CR field when the next shadow transfer occurs.

**CC8yCR1S (y = 0 - 3)**
**Channel 1 Compare Shadow Value($013C_H$ + $0100_H$ * y)** **Reset Value: $00000000_H$**

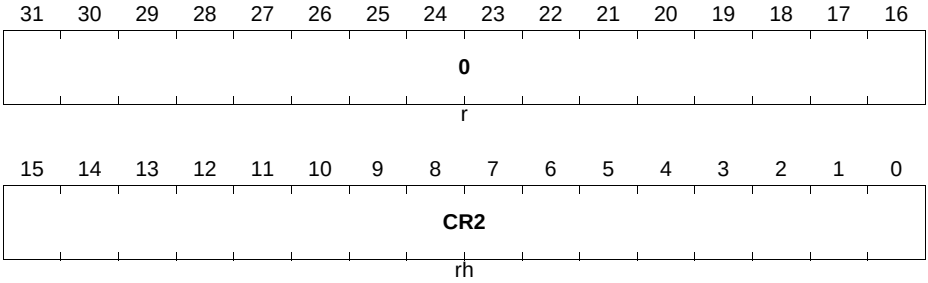



| Field | Bits | Type | Description |
|---|---|---|---|
| **CR1S** | [15:0] | rw | **Shadow Compare Register for Channel 1**<br>Contains the value for the timer comparison, that is going to be passed into the **CC8yCR1**.CR1 field when the next shadow transfer occurs.<br>*Note: In Capture Mode when a external signal is selected for capturing the timer value into the capture registers 0 and 1, the CR is not accessible for writing. A read always returns 0.* |
| **0** | [31:16] | r | **Reserved**<br>A read always returns 0. |

### CC8yCR2

This register contains the value for the timer comparison of channel 2.

**CC8yCR2 (y = 0 - 3)**
**Channel 2 Compare Value** **($0140_H + 0100_H * y$)** **Reset Value: $00000000_H$**

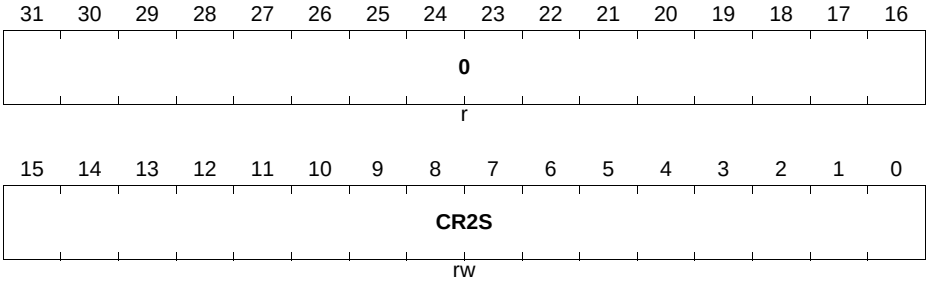



| Field | Bits | Type | Description |
|---|---|---|---|
| **CR2** | [15:0] | rh | **Compare Register for Channel 2**<br>Contains the value for the timer comparison.<br>A write always addresses the shadow register, while a read returns the actual value.<br>*Note: In Capture Mode when a external signal is selected for capturing the timer value into the capture registers 2 and 3, the CR is not accessible for writing. A read always returns 0.* |
| **0** | [31:16] | r | **Reserved**<br>A read always returns 0. |

### CC8yCR2S

This register contains the value that is going to be loaded into the **CC8yCR2**.CR field when the next shadow transfer occurs.

**CC8yCR2S (y = 0 - 3)**
**Channel 2 Compare Shadow Value($0144_H$ + $0100_H$ * y)** **Reset Value: $00000000_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **CR2S** | [15:0] | rw | **Shadow Compare Register for Channel 2**<br>Contains the value for the timer comparison, that is going to be passed into the **CC8yCR2**.CR2 field when the next shadow transfer occurs.<br>*Note: In Capture Mode when a external signal is selected for capturing the timer value into the capture registers 2 and 3, the CR is not accessible for writing. A read always returns 0.* |
| **0** | [31:16] | r | **Reserved**<br>A read always returns 0. |

### CC8yCHC

This register contains the configuration for the output connections from the two compare channels and the enable for the asymmetric mode.

**CC8yCHC (y = 0 - 3)**
**Channel Control** **($0148_H + 0100_H * y$)** **Reset Value: $00000000_H$**

| 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 | | | | | | | | | | | OCS4 | OCS3 | OCS2 | OCS1 | ASE |
| r | | | | | | | | | | | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **ASE** | 0 | rw | **Asymmetric PWM mode Enable**<br>$0_B$ Asymmetric PWM is disabled<br>$1_B$ Asymmetric PWM is enabled |
| **OCS1** | 1 | rw | **Output selector for CCU8x.OUTy0**<br>$0_B$ CC8yST1 signal path is connected to the CCU8x.OUTy0<br>$1_B$ Inverted CC8yST1 signal path is connected to the CCU8x.OUTy0 |
| **OCS2** | 2 | rw | **Output selector for CCU8x.OUTy1**<br>$0_B$ Inverted CC8yST1 signal path is connected to the CCU8x.OUTy1<br>$1_B$ CC8yST1 signal path is connected to the CCU8x.OUTy1 |
| **OCS3** | 3 | rw | **Output selector for CCU8x.OUTy2**<br>$0_B$ CC8yST2 signal path is connected to the CCU8x.OUTy2<br>$1_B$ Inverted CCST2 signal path is connected to the CCU8x.OUTy2 |
| **OCS4** | 4 | rw | **Output selector for CCU8x.OUTy3**<br>$0_B$ Inverted CC8yST2 signal path is connected to the CCU8x.OUTy3<br>$1_B$ CC8yST2 signal path is connected to the CCU8x.OUTy3 |
| **0** | [31:5] | r | **Reserved**<br>A read access always returns 0 |

### CC8yDTC

This register contains the configuration for the dead time generator.

**CC8yDTC (y = 0 - 3)**
**Dead Time Control** **($014C_H + 0100_H * y$)** **Reset Value: $00000000_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | DTCC | | DCEN4 | DCEN3 | DCEN2 | DCEN1 | DTE2 | DTE1 |
| r | | | | | | | | rw | | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **DTE1** | 0 | rw | **Dead Time Enable for Channel 1**<br>This field enables the dead time counter for the compare channel 1.<br>$0_B$ Dead Time for channel 1 is disabled<br>$1_B$ Dead Time for channel 1 is enabled |
| **DTE2** | 1 | rw | **Dead Time Enable for Channel 2**<br>This field enables the dead time counter for the compare channel 2.<br>$0_B$ Dead Time for channel 2 is disabled<br>$1_B$ Dead Time for channel 2 is enabled |
| **DCEN1** | 2 | rw | **Dead Time Enable for CC8yST1**<br>$0_B$ Dead Time for CC8yST1 path is disabled<br>$1_B$ Dead Time for CC8yST1 path is enabled |
| **DCEN2** | 3 | rw | **Dead Time Enable for inverted CC8yST1**<br>$0_B$ Dead Time for inverted CC8yST1 path is disabled<br>$1_B$ Dead Time for inverted CC8yST1 path is enabled |
| **DCEN3** | 4 | rw | **Dead Time Enable for CC8yST2**<br>$0_B$ Dead Time for CC8yST2 path is disabled<br>$1_B$ Dead Time for CC8yST2 path is enabled |

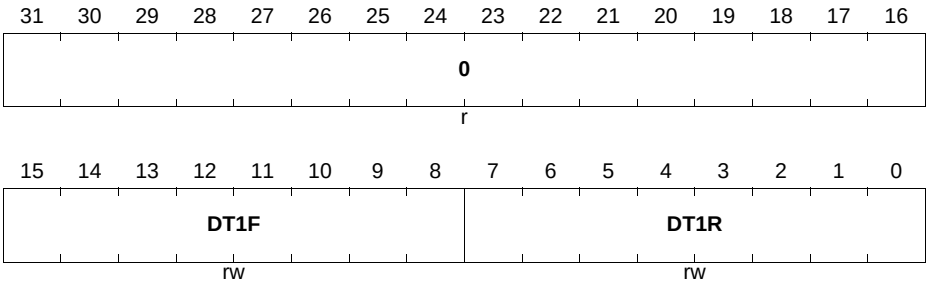

Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **DCEN4** | 5 | rw | **Dead Time Enable for inverted CC8yST2**<br>$0_B$ Dead Time for inverted CC8yST2 path is disabled<br>$1_B$ Dead Time for inverted CC8yST2 path is enabled |
| **DTCC** | [7:6] | rw | **Dead Time clock control**<br>This field controls the prescaler clock configuration for the dead time counters.<br>$00_B$ $f_{tclk}$<br>$01_B$ $f_{tclk}/2$<br>$10_B$ $f_{tclk}/4$<br>$11_B$ $f_{tclk}/8$ |
| **0** | [15:8], [31:16] | r | **Reserved**<br>A read always returns 0. |

### CC8yDC1R

This register contains the dead time value for the compare channel 1.

**CC8yDC1R (y = 0 - 3)**
**Channel 1 Dead Time Values** **($0150_H$ + $0100_H$ * y)** **Reset Value: $00000000_H$**



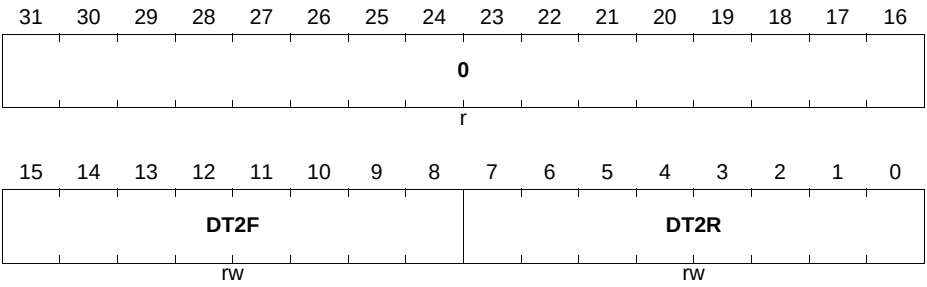

| Field | Bits | Type | Description |
|---|---|---|---|
| **DT1R** | [7:0] | rw | **Rise Value for Dead Time of Channel 1**<br>This field contains the delay value that is applied every time that a 0 to 1 transition occurs in the CC8yST1. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **DT1F** | [15:8] | rw | **Fall Value for Dead Time of Channel 1**<br>This field contains the delay value that is applied every time that a 1 to 0 transition occurs in the CC8yST1. |
| **0** | [31:16] | r | **Reserved**<br>A read access always returns 0 |

### CC8yDC2R

This register contains the dead time value for the compare channel 2.

**CC8yDC2R (y = 0 - 3)**
**Channel 2 Dead Time Values** **($0154_H$ + $0100_H$ * y)** **Reset Value: $00000000_H$**



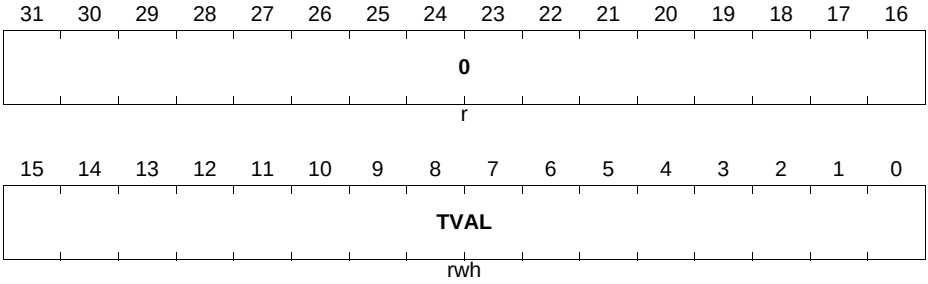

| Field | Bits | Type | Description |
|---|---|---|---|
| **DT2R** | [7:0] | rw | **Rise Value for Dead Time of Channel 2**<br>This field contains the delay value that is applied every time that a 0 to 1 transition occurs in the CC8yST2. |
| **DT2F** | [15:8] | rw | **Fall Value for Dead Time of Channel 2**<br>This field contains the delay value that is applied every time that a 1 to 0 transition occurs in the CC8yST2. |
| **0** | [31:16] | r | **Reserved**<br>A read access always returns 0 |

### CC8yTIMER

This register contains the current value of the timer.

**CC8yTIMER (y = 0 - 3)**

**Timer Value** **($0170_H + 0100_H * y$)** **Reset Value: $00000000_H$**



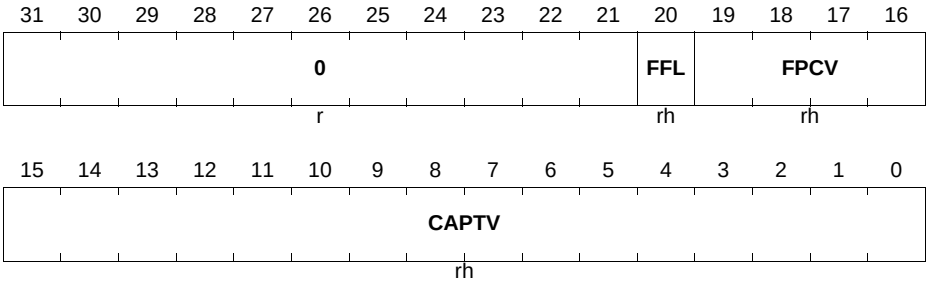

| Field | Bits | Type | Description |
|---|---|---|---|
| **TVAL** | [15:0] | rwh | **Timer Value**<br>This field contains the actual value of the timer.<br>A write access is only possible when the timer is stopped. |
| **0** | [31:16] | r | **Reserved**<br>A read access always returns 0 |

**CC8yC0V**

This register contains the values associated with the Capture 0 field.

**CC8yC0V (y = 0 - 3)**

**Capture Register 0** **($0174_H + 0100_H * y$)** **Reset Value: $00000000_H$**





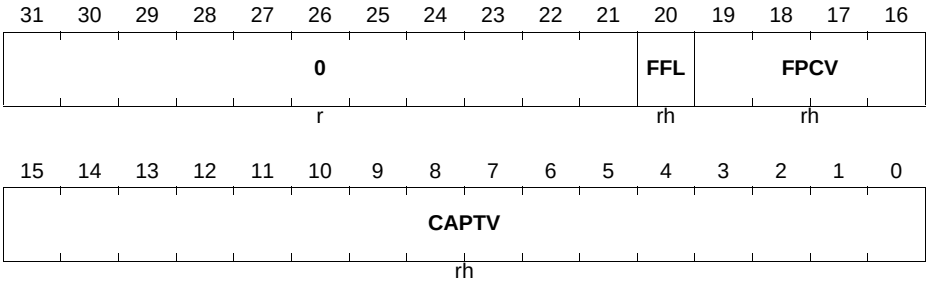
Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **CAPTV** | [15:0] | rh | **Capture Value**<br>This field contains the capture register 0 value. See **Figure 20-43**.<br>In compare mode a read access always returns 0. |
| **FPCV** | [19:16] | rh | **Prescaler Value**<br>This field contains the prescaler value when the time of the capture event into the capture register 0.<br>In compare mode a read access always returns 0. |
| **FFL** | 20 | rh | **Full Flag**<br>This bit indicates if a new value was capture into the capture register 0 after the last read access. See **Figure 20-43**. In compare mode a read access always returns 0.<br>$0_B$ No new value was captured into the specific capture register<br>$1_B$ A new value was captured into the specific register |
| 0 | [31:21] | r | **Reserved**<br>A read always returns 0 |

### CC8yC1V

This register contains the values associated with the Capture 1 field.

**CC8yC1V (y = 0 - 3)**
**Capture Register 1** **($0178_H + 0100_H * y$)** **Reset Value: $00000000_H$**





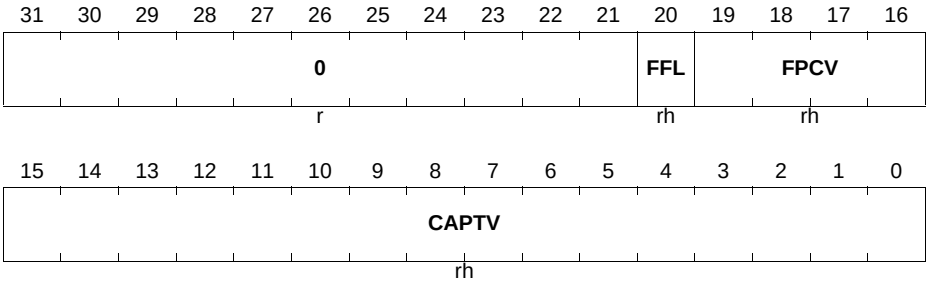
Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **CAPTV** | [15:0] | rh | **Capture Value**<br>This field contains the capture register 1 value. See **Figure 20-43**.<br>In compare mode a read access always returns 0. |
| **FPCV** | [19:16] | rh | **Prescaler Value**<br>This field contains the prescaler value when the time of the capture event into the capture register 1. In compare mode a read access always returns 0. |
| **FFL** | 20 | rh | **Full Flag**<br>This bit indicates if a new value was capture into the capture register 1 after the last read access. See **Figure 20-43**. In compare mode a read access always returns 0.<br>$0_B$ No new value was captured into the specific capture register<br>$1_B$ A new value was captured into the specific register |
| 0 | [31:21] | r | **Reserved**<br>A read always returns 0 |

### CC8yC2V

This register contains the values associated with the Capture 2 field.

**CC8yC2V (y = 0 - 3)**
**Capture Register 2** **($017C_H + 0100_H * y$)** **Reset Value: $00000000_H$**





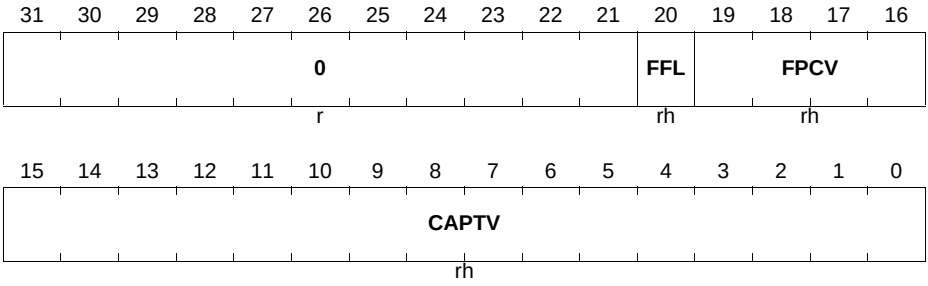
Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **CAPTV** | [15:0] | rh | **Capture Value**<br>This field contains the capture register 2 value. See **Figure 20-43**.<br>In compare mode a read access always returns 0. |
| **FPCV** | [19:16] | rh | **Prescaler Value**<br>This field contains the prescaler value when the time of the capture event into the capture register 2.<br>In compare mode a read access always returns 0. |
| **FFL** | 20 | rh | **Full Flag**<br>This bit indicates if a new value was capture into the capture register 2 after the last read access. See **Figure 20-43**. In compare mode a read access always returns 0.<br>$0_B$ No new value was captured into the specific capture register<br>$1_B$ A new value was captured into the specific register |
| 0 | [31:21] | r | **Reserved**<br>A read always returns 0 |

### CC8yC3V

This register contains the values associated with the Capture 3 field.

**CC8yC3V (y = 0 - 3)**
**Capture Register 3** **($0180_H + 0100_H * y$)** **Reset Value: $00000000_H$**





Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **CAPTV** | [15:0] | rh | **Capture Value**<br>This field contains the capture register 3 value. See Figure 20-43.<br>In compare mode a read access always returns 0. |
| **FPCV** | [19:16] | rh | **Prescaler Value**<br>This field contains the prescaler value when the time of the capture event into the capture register 3.<br>In compare mode a read access always returns 0. |
| **FFL** | 20 | rh | **Full Flag**<br>This bit indicates if a new value was capture into the capture register 3 after the last read access. See Figure 20-43. In compare mode a read access always returns 0.<br>$0_B$ No new value was captured into the specific capture register<br>$1_B$ A new value was captured into the specific register |
| 0 | [31:21] | r | **Reserved**<br>A read always returns 0 |

### CC8yINTS

This register contains the status of all interrupt sources.

**CC8yINTS (y = 0 - 3)**
**Interrupt Status** **($01A0_H + 0100_H * y$)** **Reset Value: $00000000_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | TRPF | E2AS | E1AS | E0AS | 0 | | CMD2S | CMU2S | CMD1S | CMU1S | OMDS | PMUS |
| r | | | | rh | rh | rh | rh | r | | rh | rh | rh | rh | rh | rh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **PMUS** | 0 | rh | **Period Match while Counting Up**<br>$0_B$ Period match while counting up not detected<br>$1_B$ Period match while counting up detected |
| OMDS | 1 | rh | **One Match while Counting Down**<br>$0_B$ One match while counting down not detected<br>$1_B$ One match while counting down detected |
| **CMU1S** | 2 | rh | **Channel 1 Compare Match while Counting Up**<br>$0_B$ Compare match while counting up not detected<br>$1_B$ Compare match while counting up detected |
| **CMD1S** | 3 | rh | **Channel 1 Compare Match while Counting Down**<br>$0_B$ Compare match while counting down not detected<br>$1_B$ Compare match while counting down detected |
| **CMU2S** | 4 | rh | **Channel 2 Compare Match while Counting Up**<br>$0_B$ Compare match while counting up not detected<br>$1_B$ Compare match while counting up detected |
| **CMD2S** | 5 | rh | **Channel 2 Compare Match while Counting Down**<br>$0_B$ Compare match while counting down not detected<br>$1_B$ Compare match while counting down detected |
| E0AS | 8 | rh | **Event 0 Detection Status**<br>Depending on the user selection on the **CC8yINS**.EV0EM, this bit can be set when a rising, falling or both transitions are detected.<br>$0_B$ Event 0 not detected<br>$1_B$ Event 0 detected |
| E1AS | 9 | rh | **Event 1 Detection Status**<br>Depending on the user selection on the **CC8yINS**.EV1EM, this bit can be set when a rising, falling or both transitions are detected.<br>$0_B$ Event 1 not detected<br>$1_B$ Event 1 detected |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| E2AS | 10 | rh | **Event 2 Detection Status**<br>Depending on the user selection on the **CC8yINS**.EV1EM, this bit can be set when a rising, falling or both transitions are detected.<br>$0_B$ Event 2 not detected<br>$1_B$ Event 2 detected<br>*Note: If this event is linked with the TRAP function, this field is automatically cleared when the slice exits the Trap State.* |
| TRPF | 11 | rh | **Trap Flag Status**<br>This field contains the status of the Trap Flag. |
| 0 | [7:6], [31:12] | r | **Reserved**<br>A read always returns 0. |

### CC8yINTE

Through this register it is possible to enable or disable the specific interrupt source(s).

**CC8yINTE (y = 0 - 3)**
**Interrupt Enable Control** **($01A4_H + 0100_H * y$)** **Reset Value: $00000000_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | E2AE | E1AE | E0AE | 0 | | CMD2E | CMU2E | CMD1E | CMU1E | OME | PME |
| r | | | | | rw | rw | rw | r | | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **PME** | 0 | rw | **Period match while counting up enable**<br>Setting this bit to $1_B$ enables the generation of an interrupt pulse every time a period match while counting up occurs.<br>$0_B$ Period Match interrupt is disabled<br>$1_B$ Period Match interrupt is enabled |

| Field | Bits | Type | Description |
|---|---|---|---|
| OME | 1 | rw | **One match while counting down enable**<br>Setting this bit to $1_B$ enables the generation of an interrupt pulse every time an one match while counting down occurs.<br>$0_B$ One Match interrupt is disabled<br>$1_B$ One Match interrupt is enabled |
| **CMU1E** | 2 | rw | **Channel 1 Compare match while counting up enable**<br>Setting this bit to $1_B$ enables the generation of an interrupt pulse every time a compare match while counting up occurs.<br>$0_B$ Compare Match while counting up interrupt is disabled<br>$1_B$ Compare Match while counting up interrupt is enabled |
| **CMD1E** | 3 | rw | **Channel 1 Compare match while counting down enable**<br>Setting this bit to $1_B$ enables the generation of an interrupt pulse every time a compare match while counting down occurs.<br>$0_B$ Compare Match while counting down interrupt is disabled<br>$1_B$ Compare Match while counting down interrupt is enabled |
| **CMU2E** | 4 | rw | **Channel 2 Compare match while counting up enable**<br>Setting this bit to $1_B$ enables the generation of an interrupt pulse every time a compare match while counting up occurs.<br>$0_B$ Compare Match while counting up interrupt is disabled<br>$1_B$ Compare Match while counting up interrupt is enabled |

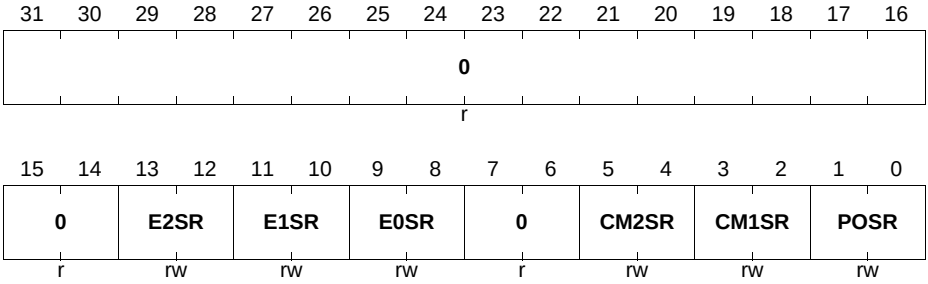
| Field | Bits | Type | Description |
|---|---|---|---|
| **CMD2E** | 5 | rw | **Channel 2 Compare match while counting down enable**<br>Setting this bit to $1_B$ enables the generation of an interrupt pulse every time a compare match while counting down occurs.<br>$0_B$ Compare Match while counting down interrupt is disabled<br>$1_B$ Compare Match while counting down interrupt is enabled |
| E0AE | 8 | rw | **Event 0 interrupt enable**<br>Setting this bit to $1_B$ enables the generation of an interrupt pulse every time that Event 0 is detected.<br>$0_B$ Event 0 detection interrupt is disabled<br>$1_B$ Event 0 detection interrupt is enabled |
| E1AE | 9 | rw | **Event 1 interrupt enable**<br>Setting this bit to $1_B$ enables the generation of an interrupt pulse every time that Event 1 is detected.<br>$0_B$ Event 1 detection interrupt is disabled<br>$1_B$ Event 1 detection interrupt is enabled |
| E2AE | 10 | rw | **Event 2 interrupt enable**<br>Setting this bit to $1_B$ enables the generation of an interrupt pulse every time that Event 2 is detected.<br>$0_B$ Event 2 detection interrupt is disabled<br>$1_B$ Event 2 detection interrupt is enabled |
| 0 | [7:6], [31:11] | r | **Reserved**<br>A read always returns 0 |

### CC8ySRS

Through this register it is possible to select to which service request line each interrupt source is forwarded.


Subject to Agreement on the Use of Product Information

**CC8ySRS (y = 0 - 3)**
**Service Request Selector** **($01A8_H + 0100_H * y$)** **Reset Value: $00000000_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **POSR** | [1:0] | rw | **Period/One match Service request selector**<br>This field selects to which slice Service request line, the interrupt(s) generated by the Period match while counting up and One match while counting down are going to be forward.<br>$00_B$ Forward to CC8ySR0<br>$01_B$ Forward to CC8ySR1<br>$10_B$ Forward to CC8ySR2<br>$11_B$ Forward to CC8ySR3 |
| **CM1SR** | [3:2] | rw | **Channel 1 Compare match Service request selector**<br>This field selects to which slice Service request line, the interrupt(s) generated by the Compare match, of channel 1, while counting up and Compare match while counting down are going to be forward.<br>$00_B$ Forward to CC8ySR0<br>$01_B$ Forward to CC8ySR1<br>$10_B$ Forward to CC8ySR2<br>$11_B$ Forward to CC8ySR3 |

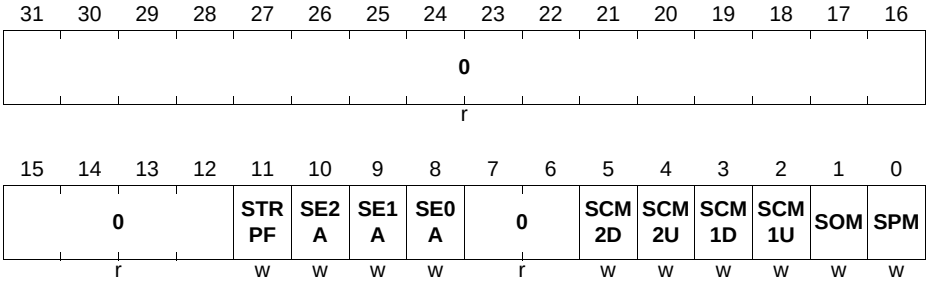

Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **CM2SR** | [5:4] | rw | **Channel 2 Compare match Service request selector**<br>This field selects to which slice Service request line, the interrupt(s) generated by the Compare match, of channel 2, while counting up and Compare match while counting down are going to be forward.<br>$00_B$ Forward to CC8ySR0<br>$01_B$ Forward to CC8ySR1<br>$10_B$ Forward to CC8ySR2<br>$11_B$ Forward to CC8ySR3 |
| **E0SR** | [9:8] | rw | **Event 0 Service request selector**<br>This field selects to which slice Service request line, the interrupt generated by the Event 0 detection are going to be forward.<br>$00_B$ Forward to CCvySR0<br>$01_B$ Forward to CC8ySR1<br>$10_B$ Forward to CC8ySR2<br>$11_B$ Forward to CC8ySR3 |
| E1SR | [11:10] | rw | **Event 1 Service request selector**<br>This field selects to which slice Service request line, the interrupt generated by the Event 1detection are going to be forward.<br>$00_B$ Forward to CC8ySR0<br>$01_B$ Forward to CC8ySR1<br>$10_B$ Forward to CC8ySR2<br>$11_B$ Forward to CC8ySR3 |
| E2SR | [13:12] | rw | **Event 2 Service request selector**<br>This field selects to which slice Service request line, the interrupt generated by the Event 2 detection are going to be forward.<br>$00_B$ Forward to CC8ySR0<br>$01_B$ Forward to CCvySR1<br>$10_B$ Forward to CC8ySR2<br>$11_B$ Forward to CC8ySR3 |
| 0 | [7:6], [31:14] | r | **Reserved**<br>Read always returns 0. |

### CC8ySWS

Through this register it is possible for the SW to set a specific interrupt status flag.

**CC8ySWS (y = 0 - 3)**
**Interrupt Status Set** **($01AC_H + 0100_H * y$)** **Reset Value: $00000000_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **SPM** | 0 | w | **Period match while counting up set**<br>Writing a $1_B$ into this field sets the **CC8yINTS**.PMUS bit. An interrupt pulse is generated if the source is enabled. A read always returns 0. |
| SOM | 1 | w | **One match while counting down set**<br>Writing a $1_B$ into this bit sets the **CC8yINTS**.OMDS bit. An interrupt pulse is generated if the source is enabled. A read always returns 0. |
| **SCM1U** | 2 | w | **Channel 1 Compare match while counting up set**<br>Writing a $1_B$ into this field sets the **CC8yINTS**.CMU1S bit. An interrupt pulse is generated if the source is enabled. A read always returns 0. |
| **SCM1D** | 3 | w | **Channel 1 Compare match while counting down set**<br>Writing a $1_B$ into this bit sets the **CC8yINTS**.CMD1S bit. An interrupt pulse is generated if the source is enabled. A read always returns 0. |
| **SCM2U** | 4 | w | **Compare match while counting up set**<br>Writing a $1_B$ into this field sets the **CC8yINTS**.CMU2S bit. An interrupt pulse is generated if the source is enabled. A read always returns 0. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **SCM2D** | 5 | w | **Compare match while counting down set**<br>Writing a $1_B$ into this bit sets the **CC8yINTS**.CMD2S bit. An interrupt pulse is generated if the source is enabled. A read always returns 0. |
| SE0A | 8 | w | **Event 0 detection set**<br>Writing a $1_B$ into this bit sets the **CC8yINTS**.E0AS bit. An interrupt pulse is generated if the source is enabled. A read always returns 0. |
| SE1A | 9 | w | **Event 1 detection set**<br>Writing a $1_B$ into this bit sets the **CC8yINTS**.E1AS bit. An interrupt pulse is generated if the source is enabled. A read always returns 0. |
| SE2A | 10 | w | **Event 2 detection set**<br>Writing a $1_B$ into this bit sets the **CC8yINTS**.E2AS bit. An interrupt pulse is generated if the source is enabled. A read always returns 0. |
| STRPF | 11 | w | **Trap Flag status set**<br>Writing a $1_B$ into this bit sets the **CC8yINTS**.TRPF bit. A read always returns 0. |
| 0 | [7:6], [31:12] | r | **Reserved**<br>Read always returns 0 |

### CC8ySWR

Through this register it is possible for the SW to clear a specific interrupt status flag.

**CC8ySWR (y = 0 - 3)**
**Interrupt Status Clear** **($01B0_H + 0100_H * y$)** **Reset Value: $00000000_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | RTRPF | RE2A | RE1A | RE0A | 0 | | RCM2D | RCM2U | RCM1D | RCM1U | ROM | RPM |
| r | | | | w | w | w | w | r | | w | w | w | w | w | w |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **RPM** | 0 | w | **Period match while counting up clear**<br>Writing a $1_B$ into this field clears the **CC8yINTS**.PMUS bit. A read always returns 0. |
| ROM | 1 | w | **One match while counting down clear**<br>Writing a 1 into this bit clears the **CC8yINTS**.OMDS bit. A read always returns 0. |
| **RCM1U** | 2 | w | **Channel 1 Compare match while counting up clear**<br>Writing a $1_B$ into this field clears the **CC8yINTS**.CMU1S bit. A read always returns 0. |
| **RCM1D** | 3 | w | **Channel 1 Compare match while counting down clear**<br>Writing a $1_B$ into this bit clears the **CC8yINTS**.CMD1S bit. A read always returns 0. |
| **RCM2U** | 4 | w | **Channel 2 Compare match while counting up clear**<br>Writing a $1_B$ into this field clears the **CC8yINTS**.CMU2S bit. A read always returns 0. |
| **RCM2D** | 5 | w | **Channel 2 Compare match while counting down clear**<br>Writing a $1_B$ into this bit clears the **CC8yINTS**.CMD2S bit. A read always returns 0. |
| RE0A | 8 | w | **Event 0 detection clear**<br>Writing a $1_B$ into this bit clears the **CC8yINTS**.E0AS bit. A read always returns 0. |
| RE1A | 9 | w | **Event 1 detection clear**<br>Writing a $1_B$ into this bit clears the **CC8yINTS**.E1AS bit. A read always returns 0. |
| RE2A | 10 | w | **Event 2 detection clear**<br>Writing a $1_B$ into this bit clears the **CC8yINTS**.E2AS bit. A read always returns 0. |
| RTRPF | 11 | w | **Trap Flag status clear**<br>Writing a $1_B$ into this bit clears the **CC8yINTS**.TRPF bit. Not valid if **CC8yTC**.TRPEN = $1_B$ and the Trap State is still active.<br>A read always returns 0. |

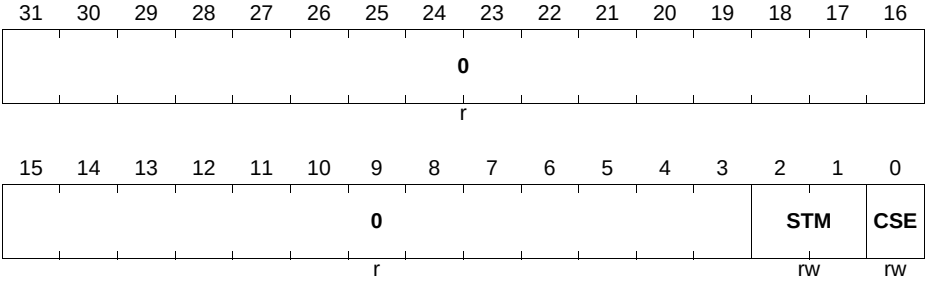

Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| 0 | [7:6], [31:12] | r | **Reserved**<br>Read always returns 0 |

### CC8ySTC

Through this register it is possible to configure the extended options for the shadow transfer mechanism.

**CC8ySTC (y = 0 - 3)**
**Shadow transfer control** **($01B4_H + 0100_H * y$)** **Reset Value: $00000000_H$**

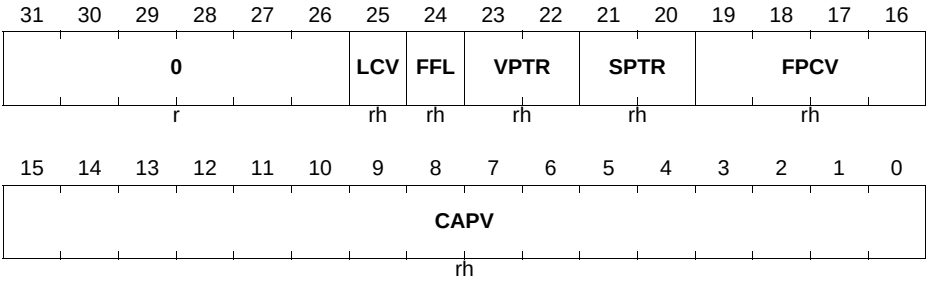



| Field | Bits | Type | Description |
|---|---|---|---|
| **CSE** | 0 | rw | **Cascaded shadow transfer enable**<br>$0_B$ Cascaded shadow transfer disabled<br>$1_B$ Cascaded shadow transfer enabled |
| STM | [2:1] | rw | **Shadow transfer mode**<br>$00_B$ Shadow transfer is done in Period Match and One match.<br>$01_B$ Shadow transfer is done only in Period Match.<br>$10_B$ Shadow transfer is done only in One Match.<br>$11_B$ Reserved<br>*Note: This field only has effect if the timer is in Center Aligned Mode.* |
| 0 | [31:3] | r | **Reserved**<br>Read always returns 0 |

### CC8yECRD0

Through this register it is possible to read back the FIFO structure of the capture function that is linked with the capture trigger 0. The read back is only valid if the **CC8yTC**.ECM = $1_B$.

**CC8yECRD0 (y = 0 - 3)**
**Extended Read Back 0** **($01B8_H + 0100_H * y$)** **Reset Value: $00000000_H$**



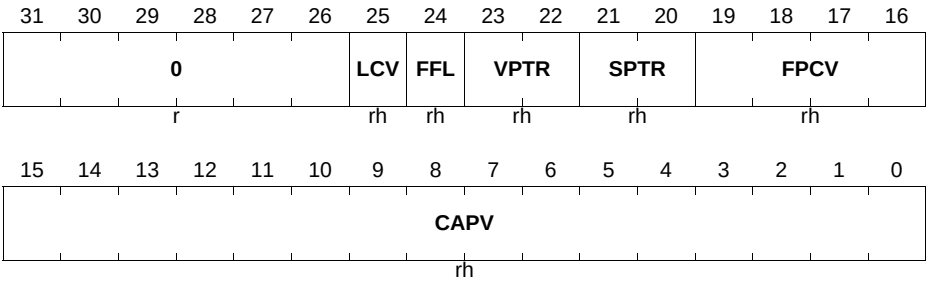

| Field | Bits | Type | Description |
|---|---|---|---|
| **CAPV** | [15:0] | rh | **Timer Capture Value**<br>This field contains the timer captured value |
| **FPCV** | [19:16] | rh | **Prescaler Capture value**<br>This field contains the value of the prescaler clock division associated with the specific CAPV field |
| **SPTR** | [21:20] | rh | **Slice pointer**<br>This field indicates the slice index in which the value was captured.<br>$00_B$ CC80<br>$01_B$ CC81<br>$10_B$ CC82<br>$11_B$ CC83 |
| **VPTR** | [23:22] | rh | **Capture register pointer**<br>This field indicates the capture register index in which the value was captured.<br>$00_B$ Capture register 0<br>$01_B$ Capture register 1<br>$10_B$ Capture register 2<br>$11_B$ Capture register 3 |

| Field | Bits | Type | Description |
|---|---|---|---|
| FFL | 24 | rh | **Full Flag**<br>This bit indicates if the associated capture register contains a new value.<br>$0_B$ No new value was captured into this register<br>$1_B$ A new value has been captured into this register |
| LCV | 25 | rh | **Lost Capture Value**<br>This field indicates if between two reads of the ECRD0 a capture trigger occured while the FIFO structure was full. If a capture trigger occured between two reads than a capture value was lost. This field is automatically cleared by the HW whenever a read to the ECRD occurs.<br>$0_B$ No capture was lost<br>$1_B$ A capture was lost |
| 0 | [31:26] | r | **Reserved**<br>Read always returns 0 |

### CC8yECRD1

Through this register it is possible to read back the FIFO structure of the capture function that is linked with the capture trigger 1. The read back is only valid if the **CC8yTC**.ECM = $1_B$.

**CC8yECRD1 (y = 0 - 3)**
**Extended Read Back 1** **($01BC_H + 0100_H * y$)** **Reset Value: $00000000_H$**



| Field | Bits | Type | Description |
|---|---|---|---|
| **CAPV** | [15:0] | rh | **Timer Capture Value**<br>This field contains the timer captured value |
| **FPCV** | [19:16] | rh | **Prescaler Capture value**<br>This field contains the value of the prescaler clock division associated with the specific CAPV field |
| **SPTR** | [21:20] | rh | **Slice pointer**<br>This field indicates the slice index in which the value was captured.<br>$00_B$ CC80<br>$01_B$ CC81<br>$10_B$ CC82<br>$11_B$ CC83 |
| **VPTR** | [23:22] | rh | **Capture register pointer**<br>This field indicates the capture register index in which the value was captured.<br>$00_B$ Capture register 0<br>$01_B$ Capture register 1<br>$10_B$ Capture register 2<br>$11_B$ Capture register 3 |
| FFL | 24 | rh | **Full Flag**<br>This bit indicates if the associated capture register contains a new value.<br>$0_B$ No new value was captured into this register<br>$1_B$ A new value has been captured into this register |
| LCV | 25 | rh | **Lost Capture Value**<br>This field indicates if between two reads of the ECRD0 a capture trigger occured while the FIFO structure was full. If a capture trigger occured between two reads than a capture value was lost. This field is automatically cleared by the HW whenever a read to the ECRD occurs.<br>$0_B$ No capture was lost<br>$1_B$ A capture was lost |
| 0 | [31:26] | r | **Reserved**<br>Read always returns 0 |


Subject to Agreement on the Use of Product Information

## 20.8 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.

### 20.8.1 CCU80 Pins

**Table 20-14 CCU80 Pin Connections**

| Global Inputs/Outputs | I/O | Connected To | Description |
|---|---|---|---|
| CCU80.MCLK | I | PCLK | Kernel clock |
| CCU80.CLKA | I | not connected | 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.IN3H; | Parity Checker delay start trigger |
| CCU80.SR0 | O | NVIC; | Service request line |
| CCU80.SR1 | O | NVIC;<br>POSIF0.MSETE; | Service request line |


Subject to Agreement on the Use of Product Information

**Table 20-14 CCU80 Pin Connections**

| Global Inputs/Outputs | I/O | Connected To | Description |
|---|---|---|---|
| CCU80.SR2 | O | VADC0.BGREQTRI;<br>VADC0.G0REQTRI;<br>VADC0.G1REQTRI;<br>ERU0.OGU03;<br>ERU0.OGU13; | Service request line |
| CCU80.SR3 | O | VADC0.BGREQTRJ;<br>VADC0.G0REQTRJ;<br>VADC0.G1REQTRJ;<br>ERU0.OGU23;<br>ERU0.OGU33; | Service request line |

**Table 20-15 CCU80 - CC80 Pin Connections**

| Input/Output | I/O | Connected To | Description |
|---|---|---|---|
| CCU80.IN0A | I | P0.12 | General purpose function |
| CCU80.IN0B | I | P0.4 | General purpose function |
| CCU80.IN0C | I | CCU40.GP02 | General purpose function |
| CCU80.IN0D | I | POSIF0.OUT2 | General purpose function |
| CCU80.IN0E | I | POSIF0.OUT5 | General purpose function |
| CCU80.IN0F | I | ERU0.PDOUT0 | General purpose function |
| CCU80.IN0G | I | ERU0.IOUT0 | General purpose function |
| CCU80.IN0H | I | SCU.GSC80 | General purpose function |
| CCU80.IN0I | I | BCCU0.OUT0 | General purpose function |
| CCU80.IN0J | I | BCCU0.OUT1 | General purpose function |
| CCU80.IN0K | I | CCU40.SR2 | General purpose function |
| CCU80.IN0L | I | ERU0.PDOUT1 | General purpose function |
| CCU80.IN0M | I | CCU80.GP10 | General purpose function |
| CCU80.IN0N | I | CCU80.ST1 | General purpose function |
| CCU80.IN0O | I | CCU80.ST2 | General purpose function |
| CCU80.IN0P | I | CCU80.ST3 | General purpose function |
| CCU80.MCI00 | I | POSIF0.MOUT[0] | Multi Channel pattern input for CCST1 |


Subject to Agreement on the Use of Product Information

**Table 20-15 CCU80 - CC80 Pin Connections**

| Input/Output | I/O | Connected To | Description |
|---|---|---|---|
| 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;<br>P1.0; | Slice compare output from channel 1. Can be the CCST1 or NOT(CCST1) path |
| CCU80.OUT01 | O | P0.1;<br>P0.5;<br>P1.1; | Slice compare output from channel 1. Can be the CCST1 or NOT(CCST1) path |
| CCU80.OUT02 | O | P0.2; | Slice compare output from channel 2. Can be the CCST2 or NOT(CCST2) path |
| CCU80.OUT03 | O | P0.3; | Slice compare output from channel 2. Can be the CCST2 or NOT(CCST2) path |
| CCU80.GP00 | O | CCU80.IN3M | 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.IN1N;<br>CCU80.IN2N;<br>CCU80.IN3N;<br>VADC0.BGREQGTM;<br>VADC0.G0REQGTM;<br>VADC0.G1REQGTM; | 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 | not connected | Channel 2 status bit: CCST2 |
| CCU80.PS0 | O | not connected | Multi channel pattern sync trigger: PM when counting UP (edge aligned) or OM when counting DOWN (center aligned) |


Subject to Agreement on the Use of Product Information

**Table 20-16 CCU80 - CC81 Pin Connections**

| Input/Output | I/O | Connected To | Description |
|---|---|---|---|
| CCU80.IN1A | I | P0.12 | General purpose function |
| CCU80.IN1B | I | P0.5; | General purpose function |
| CCU80.IN1C | I | CCU40.GP12 | General purpose function |
| CCU80.IN1D | I | POSIF0.OUT2 | General purpose function |
| CCU80.IN1E | I | POSIF0.OUT5 | General purpose function |
| CCU80.IN1F | I | ERU0.PDOUT1 | General purpose function |
| CCU80.IN1G | I | ERU0.IOUT1 | General purpose function |
| CCU80.IN1H | I | SCU.GSC80 | General purpose function |
| CCU80.IN1I | I | BCCU0.OUT2 | General purpose function |
| CCU80.IN1J | I | BCCU0.OUT3 | General purpose function |
| CCU80.IN1K | I | CCU40.SR2 | General purpose function |
| CCU80.IN1L | I | ERU0.PDOUT0 | General purpose function |
| CCU80.IN1M | I | CCU80.GP20 | General purpose function |
| CCU80.IN1N | I | CCU80.ST0 | General purpose function |
| CCU80.IN1O | I | CCU80.ST2 | General purpose function |
| CCU80.IN1P | I | CCU80.ST3 | General purpose function |
| CCU80.MCI10 | I | POSIF0.MOUT[4] | Multi Channel pattern input for CCST1 |
| CCU80.MCI11 | I | POSIF0.MOUT[5] | Multi Channel pattern input for NOT(CCST1) |
| CCU80.MCI12 | I | POSIF0.MOUT[6] | Multi Channel pattern input for CCST2 |
| CCU80.MCI13 | I | POSIF0.MOUT[7] | Multi Channel pattern input for NOT(CCST2) |
| CCU80.OUT10 | O | P0.2;<br>P0.7;<br>P1.2; | Slice compare output from channel 1. Can be the CCST1 or NOT(CCST1) path |
| CCU80.OUT11 | O | P0.3;<br>P0.6;<br>P1.3; | Slice compare output from channel 1. Can be the CCST1 or NOT(CCST1) path |


Subject to Agreement on the Use of Product Information

**Table 20-16 CCU80 - CC81 Pin Connections**

| Input/Output | I/O | Connected To | Description |
|---|---|---|---|
| CCU80.OUT12 | O | P0.5; | Slice compare output from channel 2. Can be the CCST2 or NOT(CCST2) path |
| CCU80.OUT13 | O | P0.4; | Slice compare output from channel 2. Can be the CCST2 or NOT(CCST2) path |
| CCU80.GP10 | O | CCU80.IN0M | 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.IN0N;<br>CCU80.IN2O;<br>CCU80.IN3O;<br>VADC0.BGREQGTN;<br>VADC0.G0REQGTN;<br>VADC0.G1REQGTN; | 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 | not connected | Channel 2 status bit: CCST2 |
| CCU80.PS1 | O | POSIF0.MSYNCA | Multi channel pattern sync trigger: PM when counting UP (edge aligned) or OM when counting DOWN (center aligned) |

**Table 20-17 CCU80 - CC82 Pin Connections**

| Input/Output | I/O | Connected To | Description |
|---|---|---|---|
| CCU80.IN2A | I | P0.12 | General purpose function |
| CCU80.IN2B | I | P0.10; | General purpose function |
| CCU80.IN2C | I | CCU40.GP22 | General purpose function |
| CCU80.IN2D | I | POSIF0.OUT2 | General purpose function |
| CCU80.IN2E | I | POSIF0.OUT5 | General purpose function |
| CCU80.IN2F | I | ERU0.PDOUT2 | General purpose function |
| CCU80.IN2G | I | ERU0.IOUT2 | General purpose function |
| CCU80.IN2H | I | SCU.GSC80 | General purpose function |
| CCU80.IN2I | I | BCCU0.OUT4 | General purpose function |


Subject to Agreement on the Use of Product Information

**Table 20-17 CCU80 - CC82 Pin Connections**

| Input/Output | I/O | Connected To | Description |
|---|---|---|---|
| CCU80.IN2J | I | BCCU0.OUT5 | General purpose function |
| CCU80.IN2K | I | CCU40.SR3 | General purpose function |
| CCU80.IN2L | I | ERU0.PDOUT0 | General purpose function |
| CCU80.IN2M | I | CCU80.GP30 | General purpose function |
| CCU80.IN2N | I | CCU80.ST0 | General purpose function |
| CCU80.IN2O | I | CCU80.ST1 | General purpose function |
| CCU80.IN2P | I | CCU80.ST3 | General purpose function |
| CCU80.MCI20 | I | POSIF0.MOUT[8] | Multi Channel pattern input for CCST1 |
| CCU80.MCI21 | I | POSIF0.MOUT[9] | Multi Channel pattern input for NOT(CCST1) |
| CCU80.MCI22 | I | POSIF0.MOUT[10] | Multi Channel pattern input for CCST2 |
| CCU80.MCI23 | I | POSIF0.MOUT[11] | Multi Channel pattern input for NOT(CCST2) |
| CCU80.OUT20 | O | P0.8;<br>P0.12;<br>P1.4;<br>P2.0; | Slice compare output from channel 1. Can be the CCST1 or NOT(CCST1) path |
| CCU80.OUT21 | O | P0.9;<br>P0.13;<br>P1.5;<br>P2.1; | Slice compare output from channel 1. Can be the CCST1 or NOT(CCST1) path |
| CCU80.OUT22 | O | P0.10; | Slice compare output from channel 2. Can be the CCST2 or NOT(CCST2) path |
| CCU80.OUT23 | O | P0.11; | Slice compare output from channel 2. Can be the CCST2 or NOT(CCST2) path |
| CCU80.GP20 | O | CCU80.IN1M | 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 |


Subject to Agreement on the Use of Product Information

**Table 20-17 CCU80 - CC82 Pin Connections**

| Input/Output | I/O | Connected To | Description |
|---|---|---|---|
| CCU80.ST2 | O | CCU80.IN0O;<br>CCU80.IN1O;<br>CCU80.IN3P; | 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 | not connected | Channel 2 status bit: CCST2 |
| CCU80.PS2 | O | not connected | Multi channel pattern sync trigger: PM when counting UP (edge aligned) or OM when counting DOWN (center aligned) |

**Table 20-18 CCU80 - CC83 Pin Connections**

| Input/Output | I/O | Connected To | Description |
|---|---|---|---|
| CCU80.IN3A | I | P0.12 | General purpose function |
| CCU80.IN3B | I | P0.13; | General purpose function |
| CCU80.IN3C | I | CCU40.GP32 | General purpose function |
| CCU80.IN3D | I | POSIF0.OUT2 | General purpose function |
| CCU80.IN3E | I | POSIF0.OUT5 | General purpose function |
| CCU80.IN3F | I | ERU0.PDOUT3 | General purpose function |
| CCU80.IN3G | I | ERU0.IOUT3 | General purpose function |
| CCU80.IN3H | I | SCU.GSC80 | General purpose function |
| CCU80.IN3I | I | BCCU0.OUT6 | General purpose function |
| CCU80.IN3J | I | BCCU0.OUT7 | General purpose function |
| CCU80.IN3K | I | CCU40.SR3 | General purpose function |
| CCU80.IN3L | I | ERU0.PDOUT0 | General purpose function |
| CCU80.IN3M | I | CCU80.GP00 | General purpose function |
| CCU80.IN3N | I | CCU80.ST0 | General purpose function |
| CCU80.IN3O | I | CCU80.ST1 | General purpose function |
| CCU80.IN3P | I | CCU80.ST2 | General purpose function |
| CCU80.MCI30 | I | POSIF0.MOUT[12] | Multi Channel pattern input for CCST1 |
| CCU80.MCI31 | I | POSIF0.MOUT[13] | Multi Channel pattern input for NOT(CCST1) |

**Table 20-18 CCU80 - CC83 Pin Connections**

| Input/Output | I/O | Connected To | Description |
|---|---|---|---|
| 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;<br>P2.10; | Slice compare output from channel 1. Can be the CCST1 or NOT(CCST1) path |
| CCU80.OUT31 | O | P0.14;<br>P2.11; | Slice compare output from channel 1. Can be the CCST1 or NOT(CCST1) path |
| CCU80.OUT32 | O | P0.13; | Slice compare output from channel 2. Can be the CCST2 or NOT(CCST2) path |
| CCU80.OUT33 | O | P0.12; | Slice compare output from channel 2. Can be the CCST2 or NOT(CCST2) path |
| CCU80.GP30 | O | CCU80.IN2M | 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.IN0H<br>CCU80.IN0P;<br>CCU80.IN1P;<br>CCU80.IN2P;<br>VADC0.BGREQGTF;<br>VADC0.G0REQGTF;<br>VADC0.G1REQGTF; | Output of the status bit multiplexor. It can be CCST1 or CCST2 |
| CCU80.ST3A | O | VADC0.BGREQGTE;<br>VADC0.G0REQGTE;<br>VADC0.G1REQGTE; | Channel 1 status bit: CCST1 |
| CCU80.ST3B | O | not connected | 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) |


Subject to Agreement on the Use of Product Information

# 21 Position Interface Unit (POSIF)

The POSIF unit is a flexible and powerful component for motor control systems that use Rotary Encoders or Hall Sensors as feedback loop. The several configuration schemes of the module, target a very large universe of motor control application requirements. This enables the build of simple and complex control feedback loops, for industrial and automotive motor applications, targeting high performance motion and position monitoring.

**Table 21-1 Abbreviations table**

| | |
|---|---|
| PWM | Pulse Width Modulation |
| POSIFx | Position Interface module instance x |
| CCU8x | Capture/Compare Unit 8 module instance x |
| CCU4x | Capture/Compare Unit 4 module instance x |
| CC8y | Capture/Compare Unit 8 Timer Slice instance y |
| CC4y | Capture/Compare Unit 4 Timer Slice instance y |
| SCU | System Control Unit |
| $f_{posif}$ | POSIF module clock frequency |

*Note: A small “y” or “x” letter in a register indicates an index*

## 21.1 Overview

The POSIF module is comprised of three major sub units, the Quadrature Decoder unit, the Hall Sensor Control unit and the Multi-Channel Mode unit.

The Quadrature Decoder Unit is used for position control linked with a rotary incremental encoder.

The Hall Sensor Control Unit is used for direct control of brushless DC motors.

The Multi-Channel Mode unit is used in conjunction with the Hall Sensor mode to output the wanted motor control pattern but also can be used in a stand-alone manner to perform a simple multi-channel control of several control units.

The POSIF module is used in conjunction with a CCU4 or CCU8 which enables a very flexible resource arrangement and optimization for any type of application.


Subject to Agreement on the Use of Product Information

### 21.1.1 Features

**POSIF module features**

The POSIF is built of three dedicated control units that can operate in a stand-alone manner.

The Quadrature Decoder Unit contains an output interface that enables position and velocity measurements when linked with a CCU4 module. The Hall Sensor Unit enables the control of a multi-channel pattern, that can be linked with up 8 output control sources. The Multi-Channel unit offers a complete built-in interaction with the Hall Sensor Mode and an easy stand-alone control loop.

General Features

- Quadrature Decoder
  - interface for position measurement
  - interface for motor revolution measurement
  - interface for velocity measurement
  - interrupt sources for phase error, motor revolution, direction change and error on phase decoding
- Hall Sensor Mode
  - Simple build-in mode for brushless DC motor control
  - Shadow register for the multi-channel pattern
  - Complete synchronization with the PWM signals and the multi-channel pattern update
  - interrupt sources for Correct Hall Event detection, Wrong Hall Event Detection
- Multi-Channel Mode
  - Simple usage with Hall Sensor Mode
  - stand-alone Multi-Channel mode
  - Shadow register for the multi-channel pattern

Additional features

- Quadrature Decoder mode can be used in parallel with the stand-alone Multi-Channel mode
- Several profiles (via CCU4) to perform position and velocity measurement for the Quadrature Decoder mode
- Programmable delay times (via CCU4) for input pattern evaluation and new pattern value for the Hall Sensor Mode


Subject to Agreement on the Use of Product Information

**POSIF features vs. applications**

On **Table 21-2** a summary of the major features of the POSIF unit mapped with the most common applications.

**Table 21-2 Applications summary**

| Feature | Applications |
|---|---|
| Quadrature Decoder Mode | Easy plug-in for rotary encoders:<br>• with or without index/top marker signal<br>• gear-slip or shaft winding compensation<br>• separate outputs for position, velocity and revolution control - matching different system requirements<br>• extended profile for position tracking - with revolution measurement and multiple position triggers for each revolution<br>• 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:<br>• 2 or 3 Hall sensor topologies<br>• extended input filtering to avoid unwanted pattern switch due to noisy input signals<br>• synchronization with the PWM signals of the Capture/Compare Unit<br>• Active freewheeling/synchronous rectification with dead time support (link with Capture/Compare Unit 8)<br>• easy velocity measurement function by using a Capture/Compare unit Timer Slice |
| Multi-Channel Mode | Modulating multiple PWM signals:<br>• parallel modulation controlled via SW for N PWM signals - for systems with multiple power converters<br>• generating proprietary PWM modulations<br>• parallel and synchronous shut down of N PWM signals due to system feedback |

## 21.1.2 Block Diagram

Each POSIF module can operate in three different modes, Quadrature Decoder, Hall Sensor and stand-alone Multi-channel Mode. To complete the control/measurement loop of all these three modes, the POSIF needs to be linked with a CCU4/CCU8 module.


Subject to Agreement on the Use of Product Information

In the case of the Quadrature Decoder mode, one CCU4 unit is needed, for the Hall Sensor Mode, one needs one slice of one CCU4 and a CCU8 unit. The connectivity between the stand-alone Multi-Channel mode and CCU4/CCU8 module(s) does not follow any usage constraints.

Each POSIF module contains 30 inputs and 25 outputs (including service requests), **Figure 21-1**, that are going to be mapped to available functions of the module, depending in which configuration it was selected by the user: Quadrature Decoder, Hall Sensor or stand-alone Multi-Channel.

The module also has two dedicated Service request Lines, see **Section 21.3**.

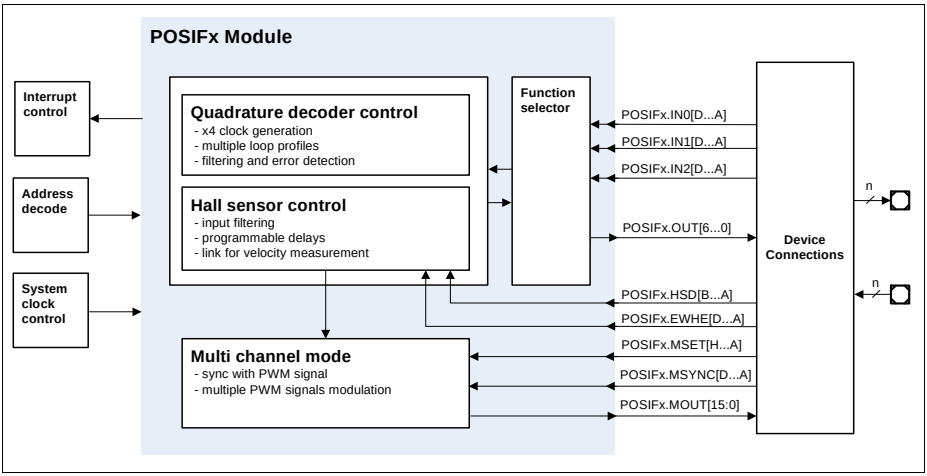



**Figure 21-1 POSIF block diagram**

## 21.2 Functional Description

### 21.2.1 Overview

The POSIF module contains a function selector unit, that is used in parallel by both Quadrature Decoder and Hall Sensor Control units. This block selects which input signals should be decoded for each control unit. The function selector is also decoding the outputs coming from these two modes (Quadrature and Hall Sensor Mode).

The POSIF module also contains a subset of inputs that are only used in Hall Sensor/Multi-Channel Mode. These inputs are connected to a timer structure (CCU4/CCU8) and are used to control the delays between pattern sampling, multi-channel pattern update and synchronization, etc.

The Multi-Channel unit contains 16 dedicated outputs, that contain the current multi-channel pattern and that can be connected to a CCU8/CCU4.

The POSIF control unit contain a dedicated Run bit, that can be set/clear by SW. It can also generate a Sync start signal that can be connected to the timer units (CCU4/CCU8).

For the Quadrature Decoder Mode, there is the possibility to define which of the signals is the leading phase and also the active level for each one.

The Quadrature Decoder Mode can also receive the clock and direction signals directly from an external source.

The Multi-Channel offers a shadow register, for the Multi-Channel pattern, enabling this way an update on the fly of these parameter. A write access always addresses the shadow register while a read, always returns the actual value of the Multi-Channel pattern.

**Table 21-3 POSIF slice pin description**

| Pin | I/O | Hall Sensor Mode | Quadrature Decoder Mode | Multi-Channel Mode (stand-alone) |
|---|---|---|---|---|
| POSIFx.IN0[D...A] | I | Hall Input 1 | Encoder Phase A or Clock | Not used |
| POSIFx.IN1[D...A] | I | Hall Input 2 | Encoder Phase B or Direction | Not used |
| POSIFx.IN2[D...A] | I | Hall Input 3 | Index/Zero marker | Not used |
| POSIFx.HSD[B...A] | I | Hall pattern sample delay | Not used | Not used |
| POSIFx.EWHE[D...A] | I | Wrong hall event emulation | Not used | Wrong hall event emulation |
| POSIFx.MSET[H...A] | I | Multi-Channel next pattern update set | Not used | Multi-Channel next pattern update set |
| POSIFx.MSYNC[D...A] | I | Multi-Channel pattern update synchronization | Not used | Multi-Channel pattern update synchronization |
| POSIFx.OUT0 | O | Hall inputs edge detection trigger | Quadrature clock | Not used |


Subject to Agreement on the Use of Product Information

**Table 21-3 POSIF slice pin description** (cont'd)

| Pin | I/O | Hall Sensor Mode | Quadrature Decoder Mode | Multi-Channel Mode (stand-alone) |
|---|---|---|---|---|
| POSIFx.OUT1 | O | Hall Correct event | Direction | Not used |
| POSIFx.OUT2 | O | Idle/ wrong hall event | Period clock | Not used |
| POSIFx.OUT3 | O | Stop | Clear/capt | Not used |
| POSIFx.OUT4 | O | Multi-Channel pattern update | Index | Not used |
| POSIFx.OUT5 | O | Sync start | Sync start | Not used |
| POSIFx.OUT6 | O | Multi Pattern sync trigger | Not used | Multi Pattern sync trigger |
| POSIFx.MOUT[15:0] | O | Multi-Channel pattern | Not used | Multi-Channel pattern |
| POSIFx.SR0 | O | Service request line 0 | Service request line 0 | Service request line 0 |
| POSIFx.SR1 | O | Service request line 1 | Service request line 1 | Service request line 1 |

*Note: The Service Request signals at the output of the kernel are extended for one more kernel clock cycle.*

### 21.2.2 Function Selector

The Function selector maps the input function signals to the selected operating mode, whether Quadrature or Hall Sensor Mode, **Figure 21-2**. The outputs are also decoded throughout this unit.

For each function input, POSI0, POSI1 and POSI2, the user can select one of the 4 input pins via the fields **PCONF**.INSEL0, **PCONF**.INSEL1 and **PCONF**.INSEL2. It is also possible to perform a low pass filtering on the three inputs, the field **PCONF**.LPC controls the low pass filters cut frequency.

The Hall Sensor Mode is the default function, **PCONF**.FSEL = $00_B$. In this mode, the Function selector maps the POSI0 to the Hall Input 1, the POSI1 to the Hall input 2 and the POSI2 to the Hall input 3.

When the Quadrature Decoder mode is selected, **PCONF**.FSEL = $01_B$, POSI0 is mapped to Phase A or Clock, POSI1 to the Phase B or Direction and POSI2 to the Index signals coming from the rotary motor encoder. Notice that it is also possible to select the


Subject to Agreement on the Use of Product Information

Quadrature Decoder and stand-alone Multi-Channel mode, by setting **PCONF**.FSEL = $11_B$.

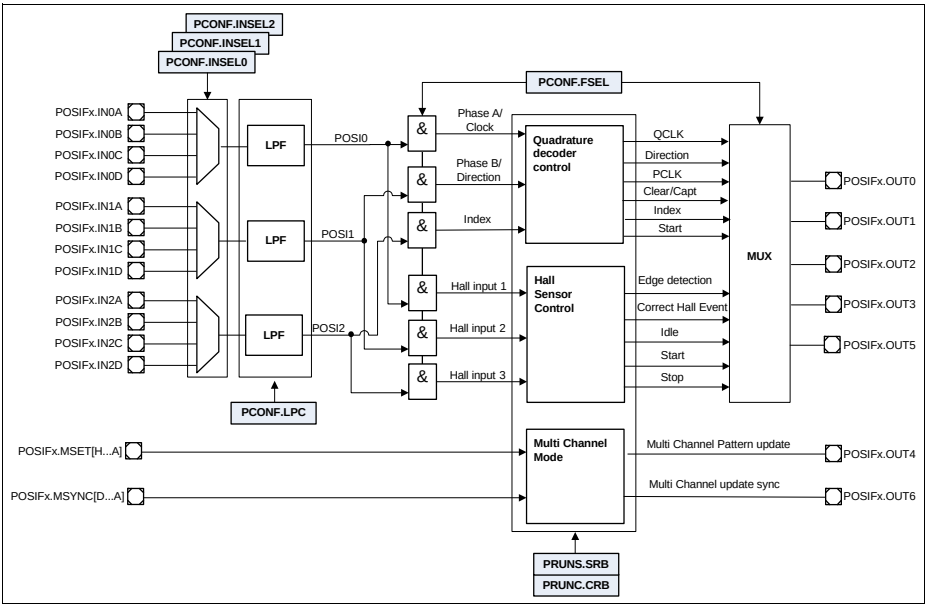



**Figure 21-2 Function selector diagram**

## 21.2.3 Hall Sensor Control

The Hall Sensor mode is divided in three major loops: detection of any update in the Hall inputs, delay between the detection and the sampling of the Hall inputs for comparison against the expected Hall Pattern and the update of the Multi-Channel pattern.

The Hall inputs are directly connected to an edge detection circuitry and with any modification in any of these three inputs, a signal is generated on the POSIFx.OUT0 pin, see **Figure 21-3**. This pin can be connected to one CCU4 slice that is controlling the delay between the edge detection and the next step on the Hall sensor mode, the sampling of the Hall Inputs.

The signal used to trigger the sample of the Hall inputs is selected via the **PCONF**.DSEL field, and this trigger can be active at the rising or falling edge (**PCONF**.SPES).

When the sampling trigger is sensed, the Hall inputs are sampled and compared against the Current Pattern, **HALP**.HCP and the Expected Hall Pattern, **HALP**.HEP (to evaluate if the input pattern match the HEP or HCP values).

The edge detection circuit generates an enable signal for sampling the hall inputs, PIFHSP and the sample logic generates a pulse every time that new values are captured, PIFHRDY, that is used inside the pattern compare logic.

When the sampled value matches the Expected Hall Pattern, a pulse is generated in the POSIFx.OUT1 pin to indicate a correct hall event. At the same time the next values programmed into the shadow registers are loaded. The **HALP**.HCP[LSB] is linked to the Hall input 1, and the **HALP**.HCP[MSB] is linked to the Hall input 3 (the same is applicable for the **HALP**.HEP register).

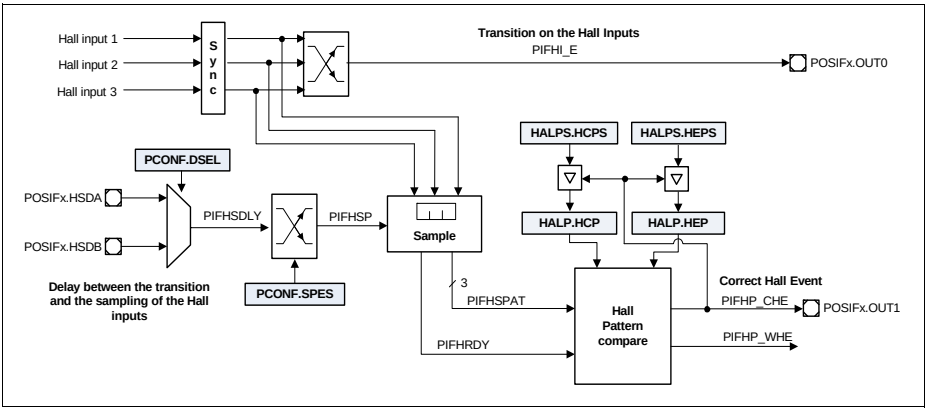



**Figure 21-3 Hall Sensor Control Diagram**

When the sampled value matches the Current Hall Pattern, a line glitch deemed to have occurred and no action is taken. When the sampled value does not match any of the values (the current and the expected pattern), a major error is deemed to have occurred and the Wrong Hall Event signal is generated. Every time that a sampled pattern leads to a wrong hall event or when it matches the current pattern a stop signal is generated throughout the POSIFx.OUT3 pin, **Figure 21-4**.

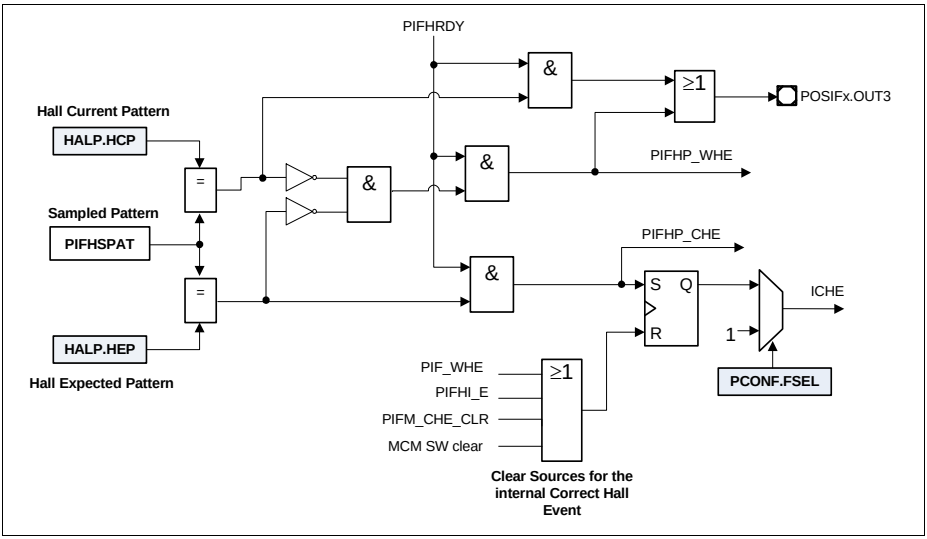



**Figure 21-4 Hall Sensor Compare logic**

A wrong hall event can generate an IDLE signal that is connected to the POSIFx.OUT2 and can be used to clear the run bit of the Hall Sensor Control unit.

The IDLE signal can also be connected to the PWM unit to perform a forced stop operation, **Figure 21-5**. The wrong hall event/idle function can also be controlled via a pin.

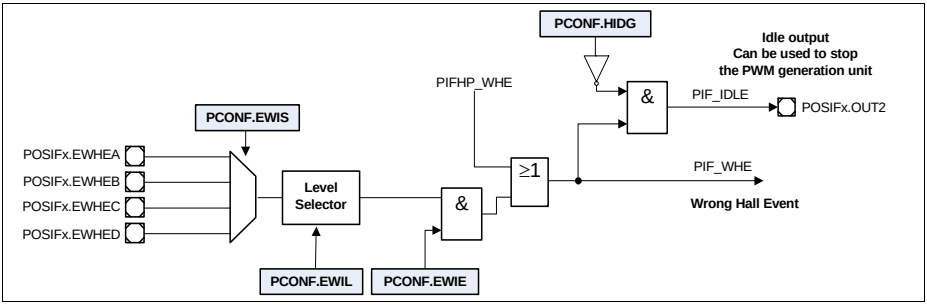



**Figure 21-5 Wrong Hall Event/Idle logic**

After the Correct Hall Event is detected, a delay can be generated between this detection and the update of the Multi-Channel pattern. On **Figure 21-6** it is demonstrated the control logic of the Multi-Channel mode.

The delay for the update of the Multi-Channel pattern can be controlled directly by a CCU4 slice. The trigger that indicates that the delay is finished, can be mapped to one of the input signals POSIFx.MSET[H...A] (**PCONF**.MSETS selects the input signal used for this purpose). One can also select the active edge for the trigger via **PCONF**.MSES.

The **PCONF**.MCUE field selects the source that enables an update of the Multi-Channel pattern. When set to $1_B$, the Multi-Channel pattern can only be updated after the SW has written a $1_B$ into the **MCMS**.MNPS field.

After the update delay, the Multi-Channel pattern still needs to be synchronized with the PWM signal. For this, the user selects a signal from the POSIFx.MSYNC[D...A] inputs via **PCONF**.MSYNS field. When a falling edge is detected in this signal, then the new multi pattern is applied at the POSIFx.MOUT[15:0] outputs, with the **MCM**.MCMP[15] linked to the POSIFx.MOUT[15] and **MCM**.MCMP[0] to POSIFx.MOUT[0].

The POSIFxOUT6 pin is connected to the **MCMF**.MSS register field. This register field is enabling the Multi-Channel pattern update (that is done upon receiving the sync signal, PIFMSYNC) and can be used in conjunction with a CAPCOM module to perform a synchronous update of the Multi-Channel pattern and the compare values used inside of the CAPCOM.

When a wrong hall event is configured to set the Hall Sensor Control into IDLE, the Multi-Channel pattern is also cleared.

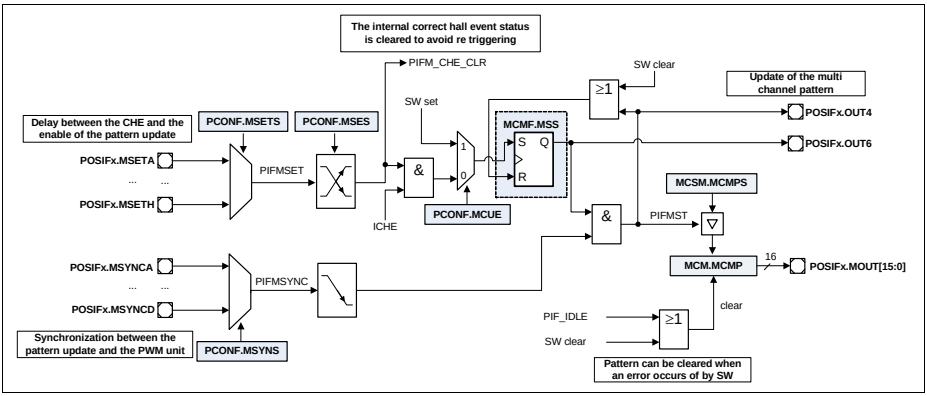



**Figure 21-6 Multi-Channel Mode Diagram**

**Figure 21-7** shows all the previous described steps in the Hall Sensor Mode. Every time that a transition on a Hall input is detected, the pin POSIFx.OUT0 is asserted. This signal is used to start the timing delay between the transition detection and the sampling of the hall inputs.

In this scenario the status output (ST in **Figure 21-7**) of a timer cell was used to control the timing 1 (delay between the transition and the sampling of the hall inputs). With the

rising edge of the ST signal, the hall inputs are sampled and if they match the expected pattern, signal POSIFx.OUT1 is asserted.

A service request line (SR signal) mapped to the same timer cell is used to control the delay between a correct Hall Event and the update of the Multi-Channel pattern. This service request is asserted when a period match is detected.

Another timer cell is used to measure the time between each correct hall event. This timer cell is represented by the Timing 2 on **Figure 21-7** (the CR symbolizes the capture register of the timer cell).

After the delay controlled by the Timing 1 (SR signal) is over, the update of the Multi-Channel pattern needs to be synchronized with the PWM signal. This is done by using one of the pattern synchronization outputs of the Capture/Compare Unit that is used to generate the PWM signals (as an example a CCU8 was used).


Subject to Agreement on the Use of Product Information

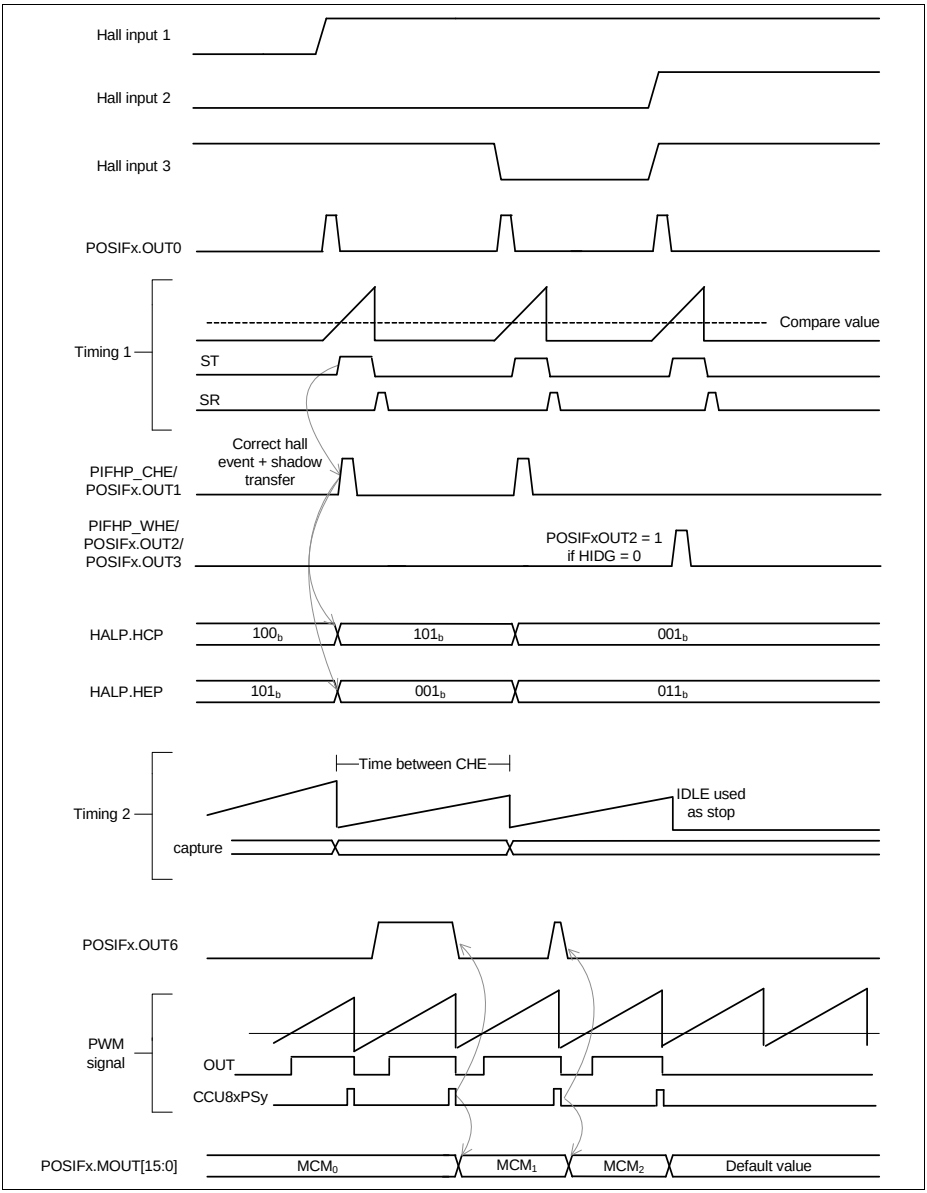



**Figure 21-7 Hall Sensor timing diagram**

### 21.2.4 Quadrature Decoder Control

The Quadrature Decoder Mode is selected by setting **PCONF**.FSEL = $01_B$ or **PCONF**.FSEL = $11_B$ (in this case the Multi-Channel mode is also enabled).

Inside the Quadrature Decoder Mode, two different subsets are available:

- standard Quadrature Mode
- Direction Count Mode

The standard mode is used when the external rotary encoder provides two phase signals and additionally and index/marker signal that is generated once per shaft revolution. The Direction Count Mode is used when the external encoder only provides a clock and a direction signal, **Figure 21-8**.

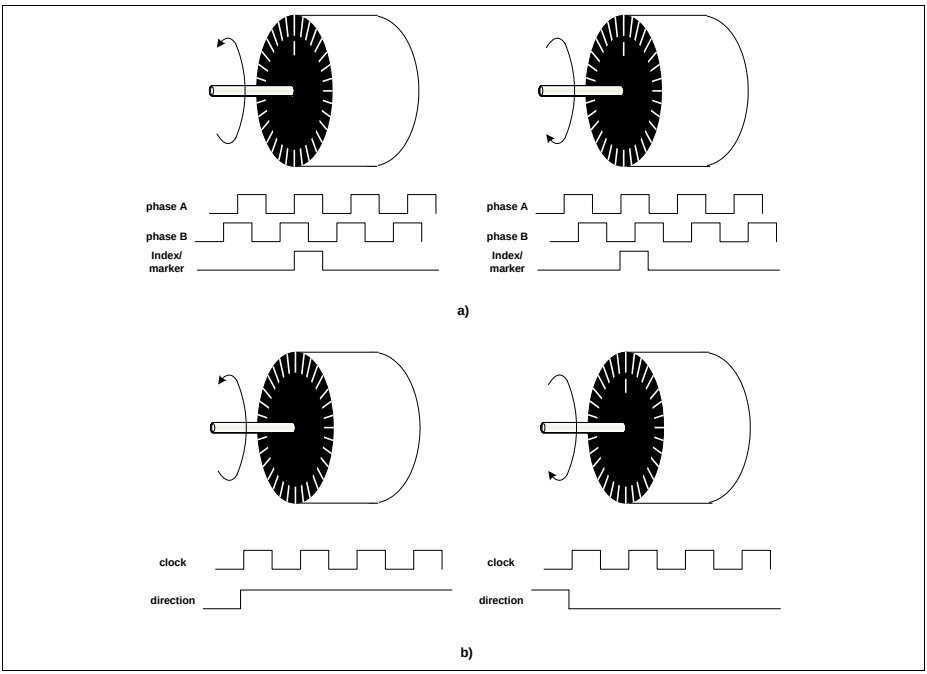



**Figure 21-8 Rotary encoder types - a) standard two phase plus index signal; b) clock plus direction**

#### Standard Quadrature Mode

The Quadrature Decoder unit offers a very flexible PhaseA/PhaseB configuration stage. Normally for a clockwise motor shaft rotation, Phase A should precede Phase B but nevertheless, the user can configure the leading phase as well the specific active state for each signal, **Figure 21-9**.

There are two major blocks that build the Quadrature Decoder Control unit: the block that decodes the quadrature clock and motor shaft direction and the block that handles the index (motor revolution) control.

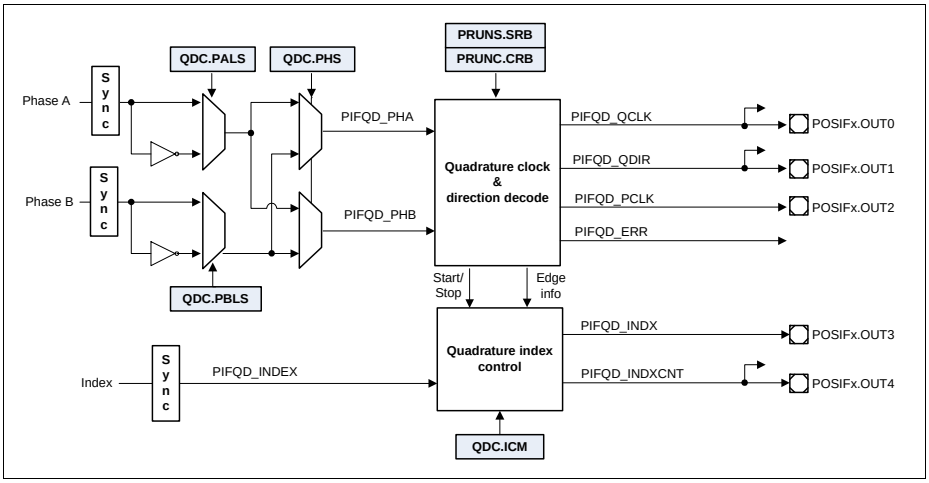



**Figure 21-9 Quadrature Decoder Control Overview**

The quadrature clock is connected to pin POSIFx.OUT0 and is used for position measurement. This clock is decoded from every edge of the phase signals and therefore there are 4 clock pulses per phase period.

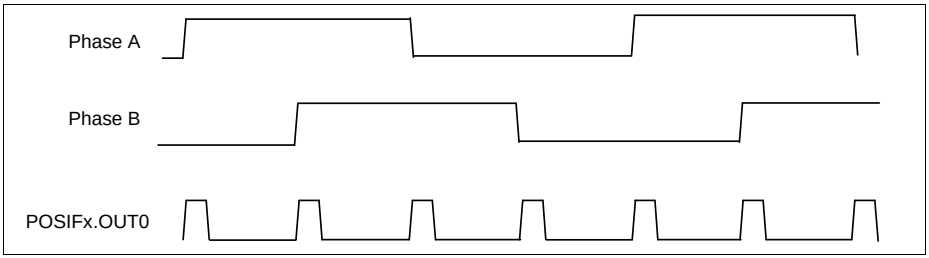



**Figure 21-10 Quadrature clock generation**

A period clock is also generated for velocity measurement operations.

The direction of the motor rotation is connected to the POSIFx.OUT1 pin and is asserted HIGH when the motor is rotating clockwise and LOW when it is turning in the counterclockwise direction.

The index control logic, memorizes which was the first edge after the assertion of the index signal, so the same quadrature transition is used for index event operations. It also

memorizes the direction of the first index, so that it can control when the revolution increment signal should be asserted.

An error signal, connected to a flag (and if enable an interrupt can be generated) also can be generated when a wrong phase pair is detected.

The Quadrature Decoder Control, uses the information of the current and previous phase pair to decode the direction and clocks. Both phase signals pass through a edge detection logic, from which the outputs are going to be used against the valid/invalid transitions stages, see **Figure 21-11**. There is the possibility to reset the decoder state machine (but not the flags and static configuration) by writing a $1_B$ into the **PRUNC**.CSM field.

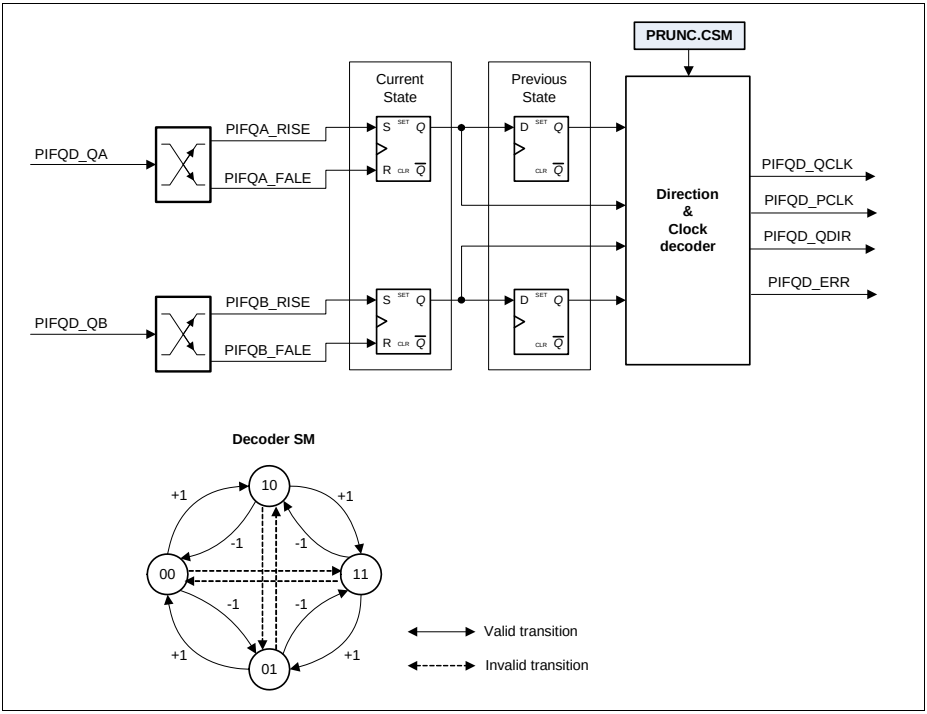



**Figure 21-11 Quadrature Decoder States**

### Direction Count Mode

Some position encoders do not have the phase signals as outputs, instead, they provide two signals that contain the clock and direction information. This is know as the Direct Count Mode.

When using a position encoder of this type, the user should set the **PCONF**.QDCM bit field to $1_B$ (enabling the direction count mode). In this case, the signal selected from the POSIFx.IN0[D...A] is used as clock and the selected signal from the POSIFx.IN1[D...A] contains the direction information.

The input signals are synchronized with the POSIF module clock and sent to the respective outputs. In this case the inputs and outputs linked with the index/zero marker are not used. The POSIFx.OUT2 output is also inactive.

### 21.2.4.1 Quadrature Clock and Direction decoding

The Quadrature Decoder unit outputs two clocks. One clock is used for position control and therefore is generated in each edge transition of the phase signals. The second clock is used for velocity measurements and is immune to possible glitches on the phase signals that can be present at very slow rotation speeds. These glitches are not normal line noise, but are due to the slow movement of the engine.

The decoding of the direction signal is done following the rules on **Figure 21-11**. **Figure 21-12** shows all the valid transitions of Phase A and Phase B signals.

**Figure 21-13** shows the difference between the two clock signals in the case that some glitches are present in the phase signals.

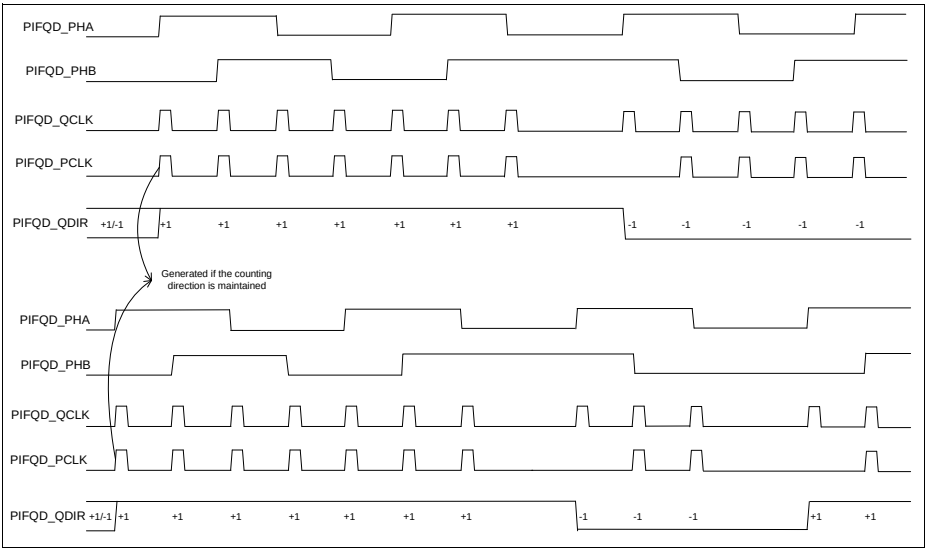



**Figure 21-12 Quadrature clock and direction timings**

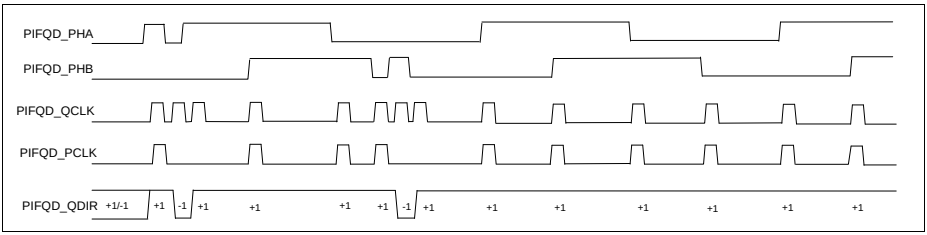



**Figure 21-13 Quadrature clock with jitter**

### 21.2.4.2 Index Control

The Index Control logic has two different outputs. One is asserted every time that an index signal is detected (and the motor shaft rotation is the same), POSIFx.OUT4, and therefore it can be used in a revolution counter. With this, the SW can monitor not only the position of the shaft but also the total number of revolutions that have occurred.

The activity of the other output, POSIFx.OUT3, can be programmed via the **QDC**.ICM field. Depending on the value set in the **QDC**.ICM, this signal can be generated:

- in every index signal occurrence
- only on the first index signal occurrence
- disabled - this outputs is never asserted

This is useful for applications that need to reset the counters on every index event or only at the first one.

The Index Control logic memorizes the immediately next phase edge that follows a index so the generated signals have always the same reference. If the first phase edge after the index is the rising edge of Phase B signal, then the index signals are going to be generated in the next index event with the rising edge of the Phase B signal if the direction is kept or with the falling edge of Phase B signal if the direction has changed.

**Figure 21-14** shows the timing diagram for the generation of the index signals. In this case the **QDC**.ICM = $10_B$, which means that the POSIFx.OUT3 index signal is going to be generated in every occurrence of the input index.


Subject to Agreement on the Use of Product Information

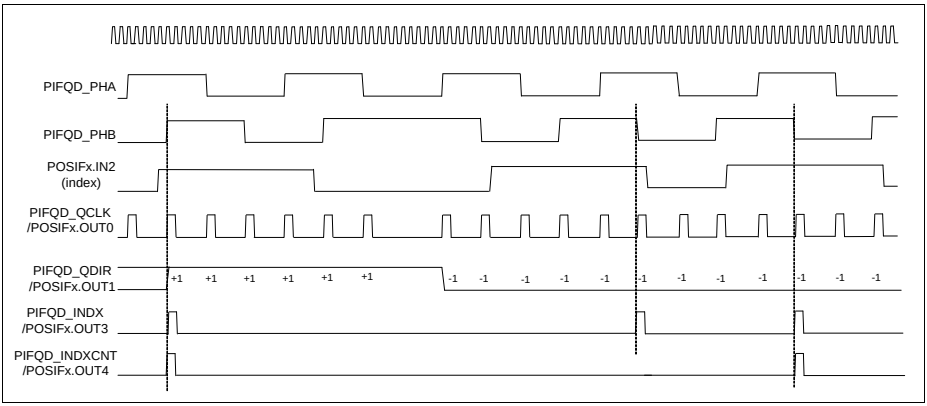



**Figure 21-14 Index signals timing**

### 21.2.5 Stand-Alone Multi-Channel Mode

The Multi-Channel mode (Multi-Channel Mode logic can be see on **Figure 21-6**) can be used without the Hall Sensor Control, by setting the **PCONF**.FSEL = $10_B$ or if the Quadrature Decoder Mode is also needed, by setting **PCONF**.FSEL = $11_B$.

In stand-alone Multi-Channel Mode, the mechanism to update the multi-channel pattern is the same as the one described in **Section 21.2.3**.

The trigger for a pattern update can come from the SW, by writing $1_B$ to **MCMS**.MNPS, or it can come from an external signal like in Hall sensor Mode. This external signal should then be mapped to the PIFMSET function by selecting the appropriate value in the **PCONF**.MSETS field.

The synchronization between the update of the new Multi-Channel pattern and control signal still needs to be done. The user needs to map one input signal of the POSIFx.MSYNC[D...A] range to this function.

The SW has the possibility of clearing the actual Multi-Channel pattern, by writing 1 into the **MCMC**.MPC field.

### 21.2.6 Synchronous Start

The POSIF module has a synchronous start output, POSIFx.OUT5, that can be used together with the CAPCOM for a complete synchronous start of both modules. The synchronous start is linked with the run bit of the POSIF module, which means that every time that the run bit is set, a pulse is generated throughout the POSIFx.OUT5 pin.

By using the synchronous start output, the SW does not perform two independent accesses to start the POSIF and the CCU4/CCU8 and therefore is guaranteed that both modules start their operation at the exact same time.


Subject to Agreement on the Use of Product Information

### 21.2.7 Using the POSIF

The POSIF module needs to be linked with a CCU4/CCU8 module to perform the full set of functions in each of the possible modes (due to the fact that doesn't contain built-in counters/timers).

To operate the POSIF in the Quadrature Decoder Mode, one CCU4 module is needed. The Hall Sensor Mode, needs a CCU8 (at least 3 slices are need to control a brushless DC motor) and also (at least) two CCU4 or CCU8 slices. The stand-alone Multi-Channel Mode linking configuration depends heavily on the use cases and therefore the number of slices of CCU4 or CCU8 used is freely chosen by the user.

#### 21.2.7.1 Hall Sensor Mode Usage

When using the Hall Sensor Mode of the POSIF, the Multi-Channel Mode is also working. Due to that fact, the CCU8 module used to perform the Multi-Channel Modulation, needs to be configured in Multi-Channel Mode.

**Standard Hall Sensor Mode Usage**

On **Figure 21-15**, the Hall Sensor Mode is used in conjunction with two CCU4 slices and one CCU8 module. The first slice of CCU4, slice 0 is being used to control the delays between the edge detection of the Hall Inputs and the actual sampling, and also to control the delay between a Correct Hall Event and the Multi-Channel Pattern update enable.

The rising edge of the CCU4x.ST0 is used as finish trigger for the first delay, while the Service request line is used for triggering the update of a new pattern after a Correct Hall Event. The service request is configured to be active on each period match hit of the specific slice.

Slice 0 is configured in single shot mode, so that the time delay can be re triggered every time that a request from the POSIF occurs.

The second slice of the CCU4 unit, Slice 1, is being used in Capture Mode, to capture the time between Correct Hall Events (storing this way the motor speed between two correct hall events). The POSIFx.OUT1 of the POSIF is used as capture trigger for the slice while the POSIFx.OUT3 is used as stop. The capture and stop triggers are configured in the specific timer slices as active on the rising edge.

The CCU8 is the module that is generating the PWM signals to control the motor and therefore, the Multi-Channel Pattern outputs POSIFx.MOUT[7:0] are linked to this unit.

To close the Multi-Channel loop, an output of the CCU8 needs to be connected to the POSIF module (one of the CCU8x.PSy signals), so that the Multi-Channel Pattern is updated synchronously with the PWM period.


Subject to Agreement on the Use of Product Information

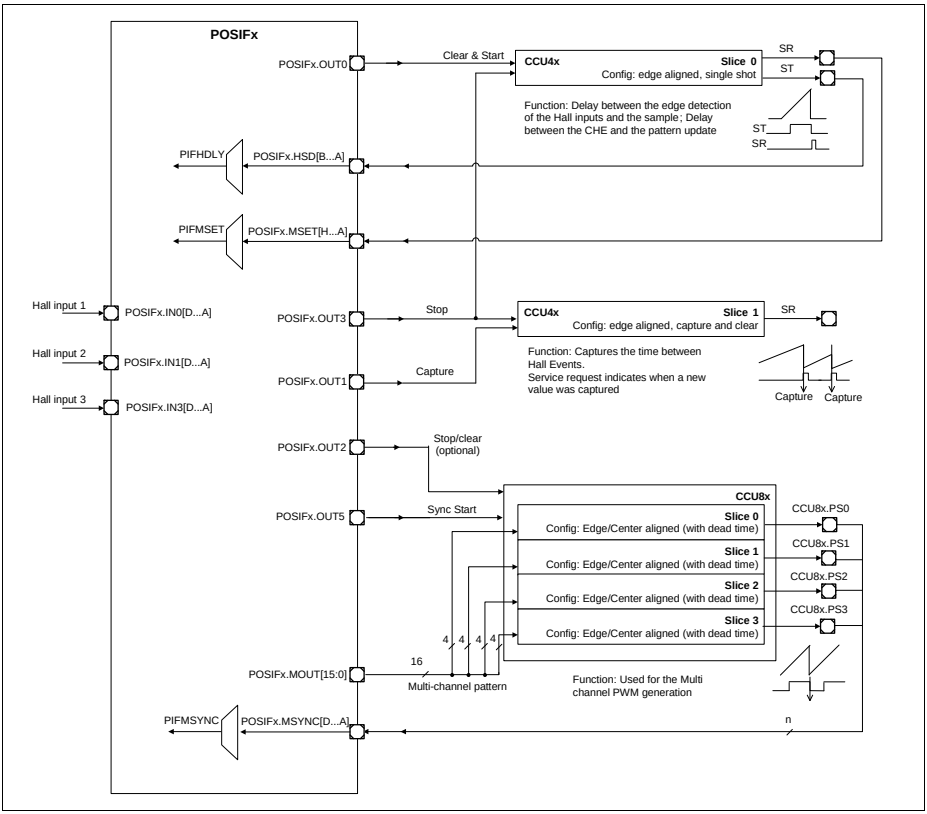



**Figure 21-15 Hall Sensor Mode usage - profile 1**

### Hall Sensor Mode Usage - Flexible Time Control

On **Figure 21-16**, another profile is demonstrated. In this case instead of 2 CCU4 slices, the Hall Sensor Mode is using 3 CCU4 slices. This profile gives more flexibility in terms of delay configuration. It uses two timer slices to control independently the delay between the transition of the hall inputs and sampling, and the delay between a Correct Hall Event and the update of the Multi-Channel pattern. At the same time it also removes the need of using a service request to control the pattern update delay.

Slice 0 is used to control the delay between a transition at the hall inputs and the actual sampling. Slice 2 is used to control the delay between a Correct Hall Event and the update of the Multi-Channel pattern.

Slice 1 is used as keeper of the time stamp between Correct Hall events (the same function as Slice 1 in profile 1).

The synchronism between the PWM signal and the update of the Multi-Channel pattern is again done with one of the CCU8x.PSy outputs.

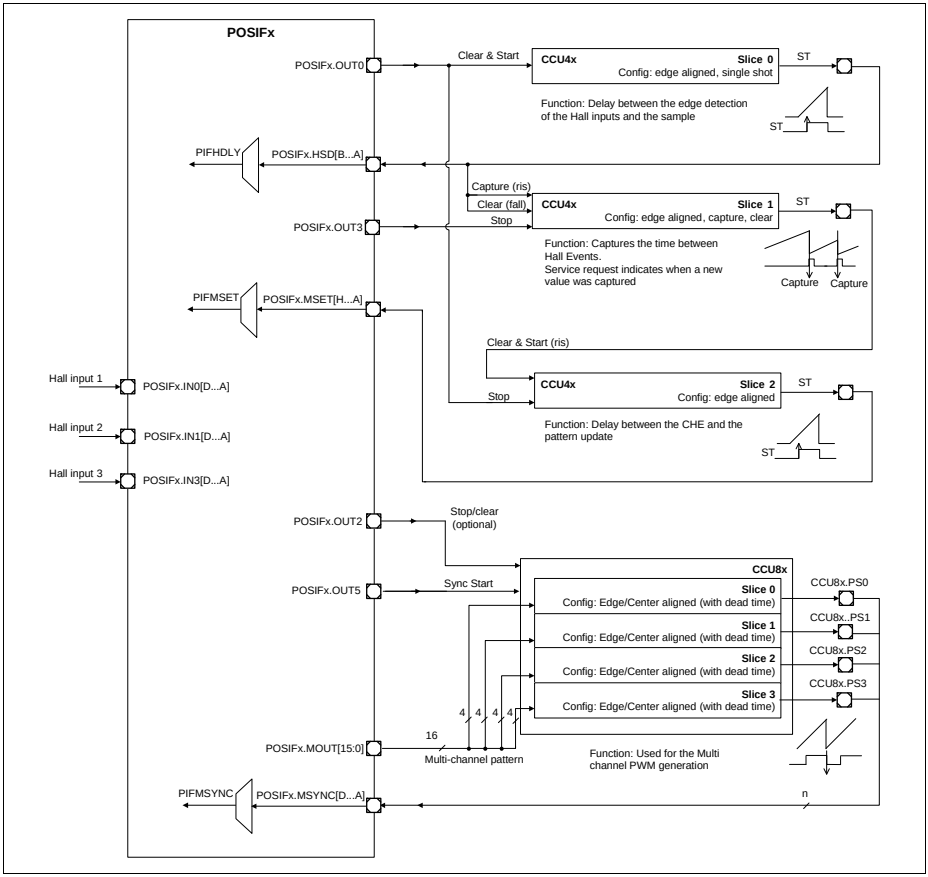



**Figure 21-16 Hall Sensor Mode usage - profile 2**

### 21.2.7.2 Quadrature Decoder Mode usage

The Quadrature Decoder Mode can be used in a very flexible way when connected with a CCU4 unit. The connection profile depends on the amount of functions that the user wants to perform in parallel: position control, revolution control and velocity measurement.

### Quadrature Decoder Usage - Tick and Revolution Compare plus Velocity Between N Ticks

On **Figure 21-17**, the POSIF is linked with a CCU4, using all the module timer slices. In this profile, the user in Quadrature Decoder Mode has a slice being used for position control (comparison), one for revolutions counter and two aggregated slices used for velocity measurement.

Slice 0 is connected to the POSIFx.OUT0 and POSIFx.OUT1 outputs, which means that one has to configure POSIFx.OUT0 as counting functionality and POSIFx.OUT1 as Up/Down counting function. This slice is then used to track the actual position of the system and the compare channel can be configured to trigger the required actions, when the position reaches a certain value.

Slice 1 is connected to POSIFx.OUT4 pin, which means that it is used as a motor revolution counter. The compare channel can be programmed to trigger an interrupt every time that the motor performs N revolutions.

Slice 2 and Slice 3 are used to perform velocity measurements. Slice 2 receives the Quadrature Decoder period clock via POSIFx.OUT2, that is mapped to a counting functionality. The compare channel of this slice is then used to trigger a capture event in Slice3. The last one is using the module clock and therefore in every capture event, the actual system ticks are captured. This way the user knows how much time has elapsed between N phase periods and can calculate the corresponding velocity profiles.


Subject to Agreement on the Use of Product Information

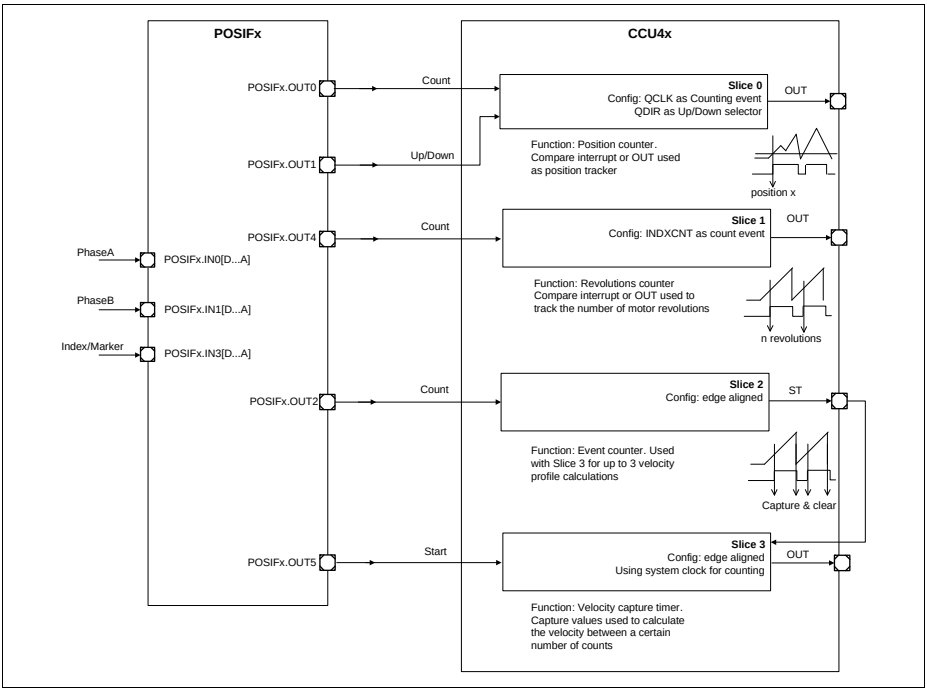



**Figure 21-17 Quadrature Decoder Mode usage - profile 1**

### Quadrature Decoder Usage - Extended Tick Comparison plus Velocity Between N Ticks

The profile demonstrated in **Figure 21-18** enables the usage of 2 compare channels for the position control. This profile is especially useful for multi operations during just one motor revolution.

Slice 0 and Slice 1 are using the POSIFx.OUT0 as counting function and the POSIFxOUT1 as up/down control. The compare channels in each slice are then programmed with different compare values.

Slice 2 and Slice 3 are used in the same manner as profile 1, **Figure 21-18**.

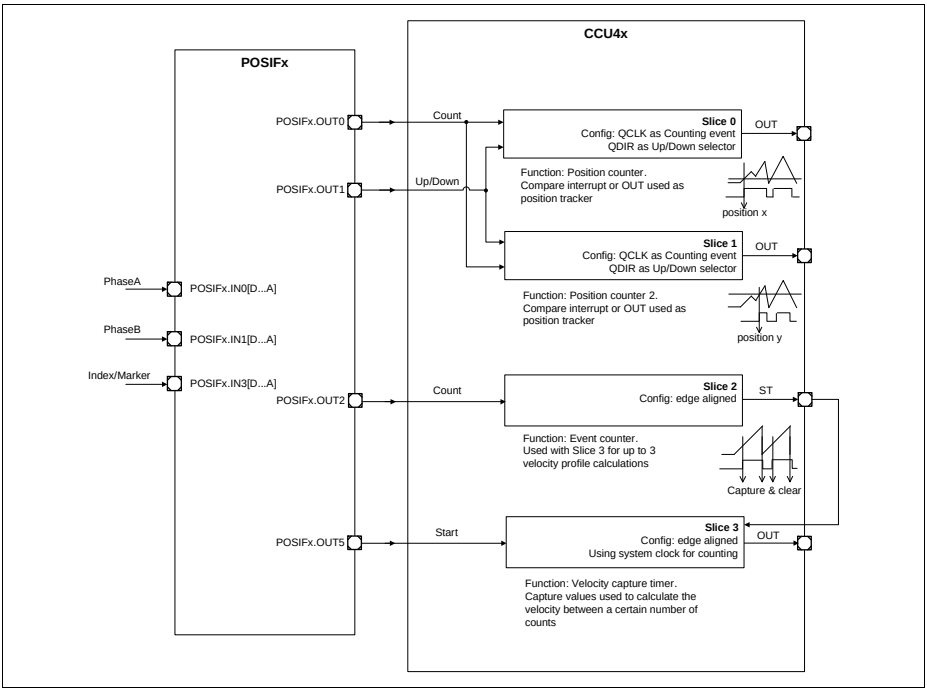



**Figure 21-18 Quadrature Decoder Mode usage - profile 2**

### Quadrature Decoder Usage - Tick and Revolution Comparison with Index Clear plus Velocity Between N Ticks

In some applications it is useful to use the index marker as a clear signal for the position and velocity control. This clear action is linked with the POSIFx.OUT3 pin, that can be programmed to be asserted only at the first index marker or at all maker hits. This pin is then connected to the specific CCU4 slices and used as clear functionality. **Figure 21-19** shows the adaptation of profile 1 with the index marker signal used as clear signal.

The same procedure can be used in profile 2, so that we can have the 2 compare channels plus the velocity measurement and the index used as clear signal.

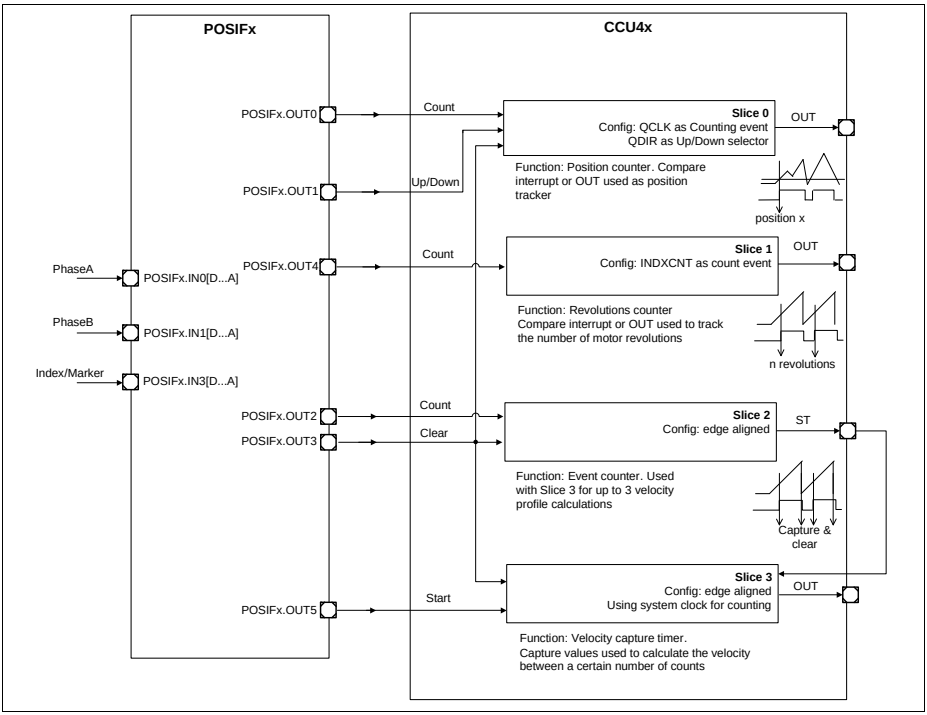



**Figure 21-19 Quadrature Decoder Mode usage - profile 3**

### Quadrature Decoder Usage - Tick Comparison plus Micro Tick Velocity for Slow Rotating Speeds

When the motor rotating speed is slow, it may not be suitable to discard the time between each occurrence of a rotary encoder tick.

In a fast rotating system, the time between ticks is normally discarded and the velocity calculation can be done, by taking into account the number of ticks that have been elapsed since the last ISR. But in a slow rotating system, taking into account the number of ticks may not be enough (because of the associated error), and the software needs also to take into consideration, the time between the last tick and the actual ISR trigger.

**Figure 21-20** shows a slow rotating system, where the ISR for the velocity calculation is triggered in a periodic way. In this case, because of the small amount of ticks between each ISR trigger, the software needs to know not only the amount of elapsed ticks but also the elapsed time between the last tick and the ISR occurrence.

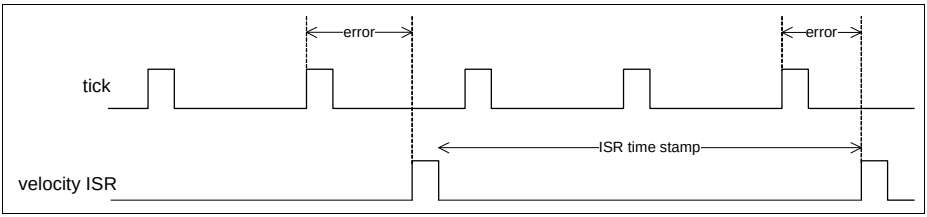



**Figure 21-20 Slow rotating system example**

It is possible to built a profile with the POSIF and one CCU4 module to perform a control loop that is immune to this slow velocity calculation pitfalls. The resource usage is exemplified on **Figure 21-21**.

One timer slice of a CCU4 module, Slice 0, is used to monitor the current position of the motor shaft.

Three additional timer slices are needed to built the slow velocity calculation loop: Slice 1, Slice 2 and Slice 3.

Timer Slice 1, is counting the number of ticks (PCLK) that are elapsed between each velocity ISR occurrence.

Timer Slice 2, is counting the time between each tick (PCLK) occurrence. This timer slice is cleared and started within every tick and therefore it always keeps the timing info between the last and the current tick.

Timer Slice 3, is controlling the periodicity of the velocity ISR. Every time that a velocity ISR is triggered, Slice 3 also triggers a capture in Slice 2 and a capture & clear in Slice 1.

With this mechanism, every time that the software reads back the captured values from Slice 1 and Slice 2, it can calculate the speed based on:

- the amount of ticks elapsed since the last ISR
- plus the amount of time elapsed between the last tick and the ISR

This control loop offers therefore a very accurate way to calculate the motor speed within systems with low velocity.


Subject to Agreement on the Use of Product Information

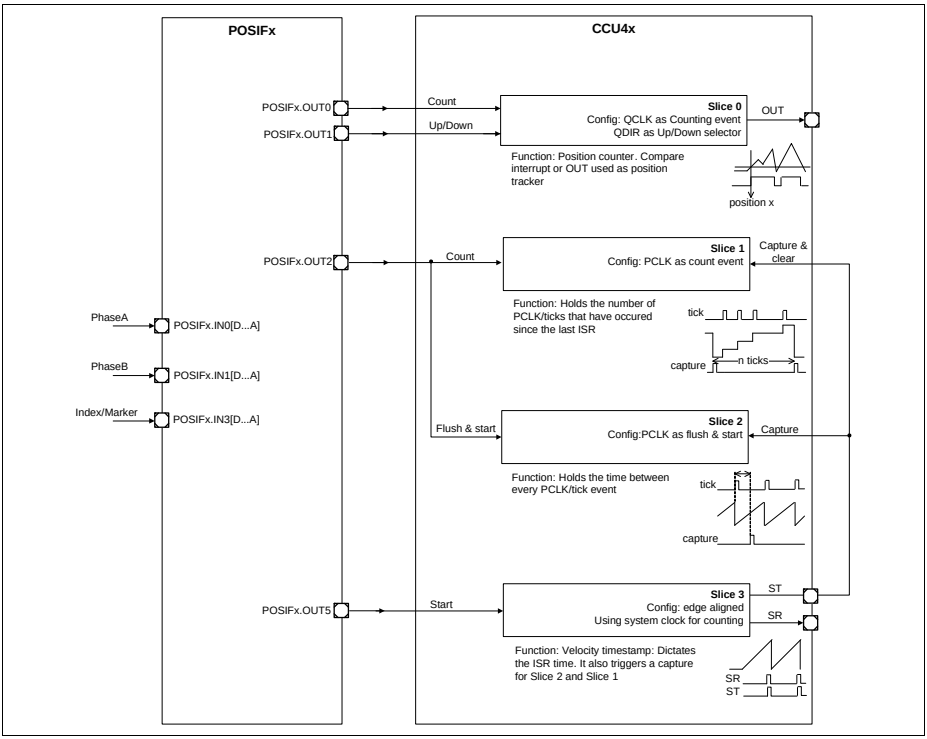



**Figure 21-21 Quadrature Decoder Mode usage - profile 4**

### 21.2.7.3 Stand-alone Multi-Channel Mode

The Multi-Channel Mode can be used in the POSIF module without being linked with the Hall Sensor Mode. This is especially useful for performing generic control loops that need to be perfectly synchronized.

The stand-alone Multi-Channel Mode is very generic and depends very much on the control loop specified by the user. A generic profile for the Multi-Channel mode is to use a CCU4/CCU8 module to perform the signal generation and a slice from a CCU4 module linked with an external pin that is used as trigger for updating the Multi-Channel pattern.

On **Figure 21-22**, a slice from a CCU4 unit is used to control the delay between the update of an external signal (e.g. external sensor) and the update of the Multi-Channel Pattern. Notice that this external signal can also be connected to the POSIF module if no timing delay is needed or it can be just controlled by SW, by writing an one into the **MCMS**.MNPS field.

One output can then be chosen from the CCU4/CCU8 unit that is generating the PWM signals, to act as synchronization trigger between the generated signal and the update of the Multi-Channel pattern (CCU4x.PSy in case of CCU4 and CCU8x.PSy in case of CCU8).

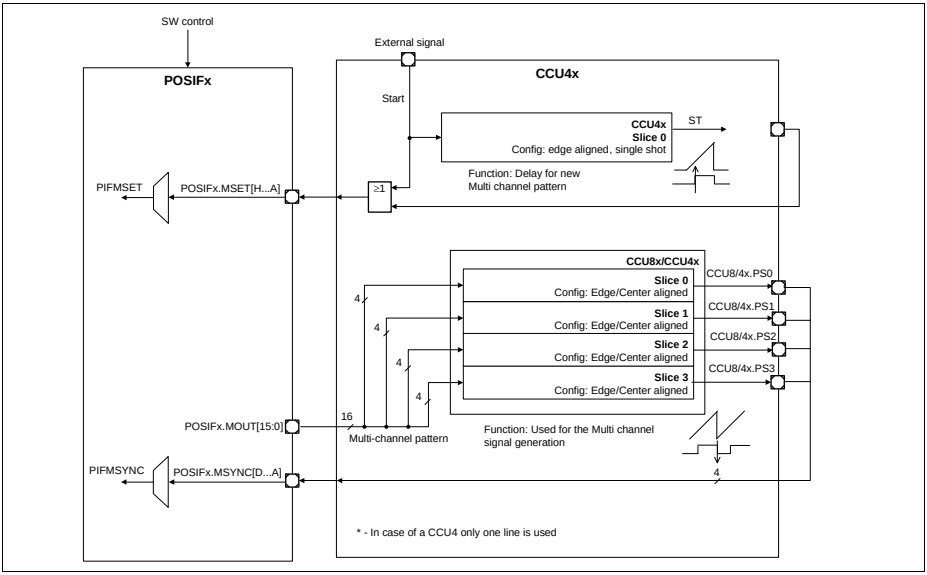



**Figure 21-22 Stand-alone Multi-Channel Mode usage**

## 21.3 Service Request Generation

The POSIF has several interrupt sources that are linked to the different operation modes: Hall Sensor Mode, Quadrature Decoder Mode and stand-alone Multi-Channel Mode.

The interrupt flags for the Hall Sensor Mode are described in **Section 21.3.1** while the interrupt flags of the Quadrature Decoder Mode are described in **Section 21.3.2**.

The flags associated with the Multi-Channel function are available in all the modes. This is due to the fact that the Multi-Channel Mode can operate in parallel with the Quadrature Decoder Mode and is needed whenever the Hall Sensor Mode is activated.

Each of the interrupt sources, can be routed to the POSIFx.SR0 or POSIFx.SR1 output, depending on the value programmed in the **PFLGE** register.

### 21.3.1 Hall Sensor Mode flags

The Hall Sensor Control contains four flags that can be configured to generate an interrupt request pulse, see **Figure 21-23**.

The four interrupt sources are:

- Transition at the Hall Inputs (**PFLG**.HIES)
- occurrence of a correct hall event (**PFLG**.CHES)
- occurrence of a wrong hall event (**PFLG**.WHES)
- shadow transfer of the Multi-Channel pattern (**PFLG**.MSTS)

The last one is triggered every time the Multi-Channel pattern is updated (PIFMST), which means that the POSIFx.MOUT[15:0] output was updated with a new value (see also **Figure 21-6**).

Each of this interrupt sources can be enabled/disabled individually. The SW also has the possibility to set and clear the specific flags by writing a $1_B$ into the specific fields of **SPFLG** and **RPFLG** registers. By enabling an interrupt source, an interrupt pulse is generated every time that a flag set operation occurs, independently if the flag is already set or not.

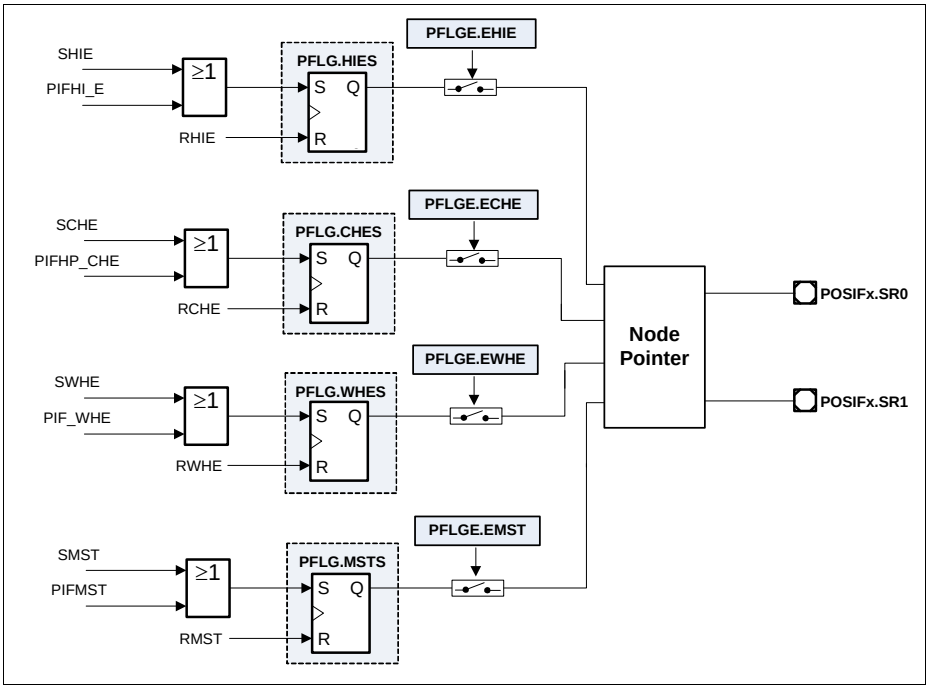



**Figure 21-23 Hall Sensor Mode flags**


Subject to Agreement on the Use of Product Information

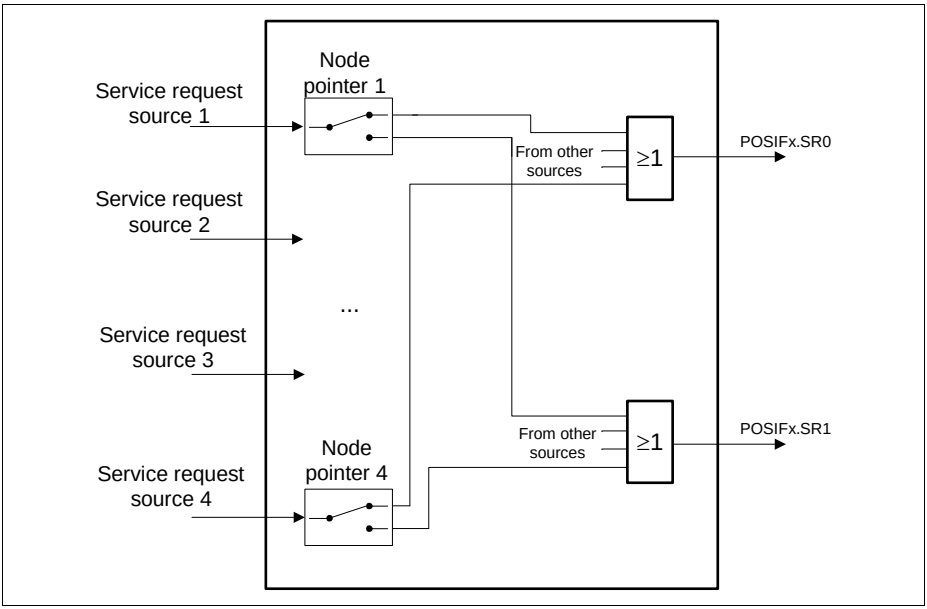



**Figure 21-24 Interrupt node pointer overview - hall sensor mode**

### 21.3.2 Quadrature Decoder Flags

The Quadrature Decoder Mode has five flags that can be enabled individually as interrupt sources (besides these five flags, the ones that are linked to the usage of Multi-Channel mode are also available: Multi-Channel Pattern update (**PFLG**.MSTS) and Wrong Hall event (**PFLG**.WHES). This can be useful if one selects the Quadrature Mode and the Multi-Channel stand-alone functionality.

These five flags are:

- Index event detection (**PFLG**.INDXS)
- phase detection error (**PFLG**.ERRS)
- quadrature clock generation (**PFLG**.CNTS)
- period clock generation (**PFLG**.PCLKS)
- direction change (**PFLG**.DIRS)

By enabling an interrupt source, an interrupt pulse is generated every time that a flag set operation occurs, independently if the flag is already set or not.

The index event detection flag is set every time that a index is detected.

The phase error flag is set when one invalid transition on the phase signals is detected, see **Figure 21-11**.

The quadrature clock and period clock flags are generated accordingly with the timings shown in **Figure 21-12**.

The direction change flag is set, every time that an inversion of the motor rotation happens.

Each flag can be set/cleared individually by SW, by writing into the specific field on the registers **SPFLG** and **RPFLG**, see **Figure 21-25**.

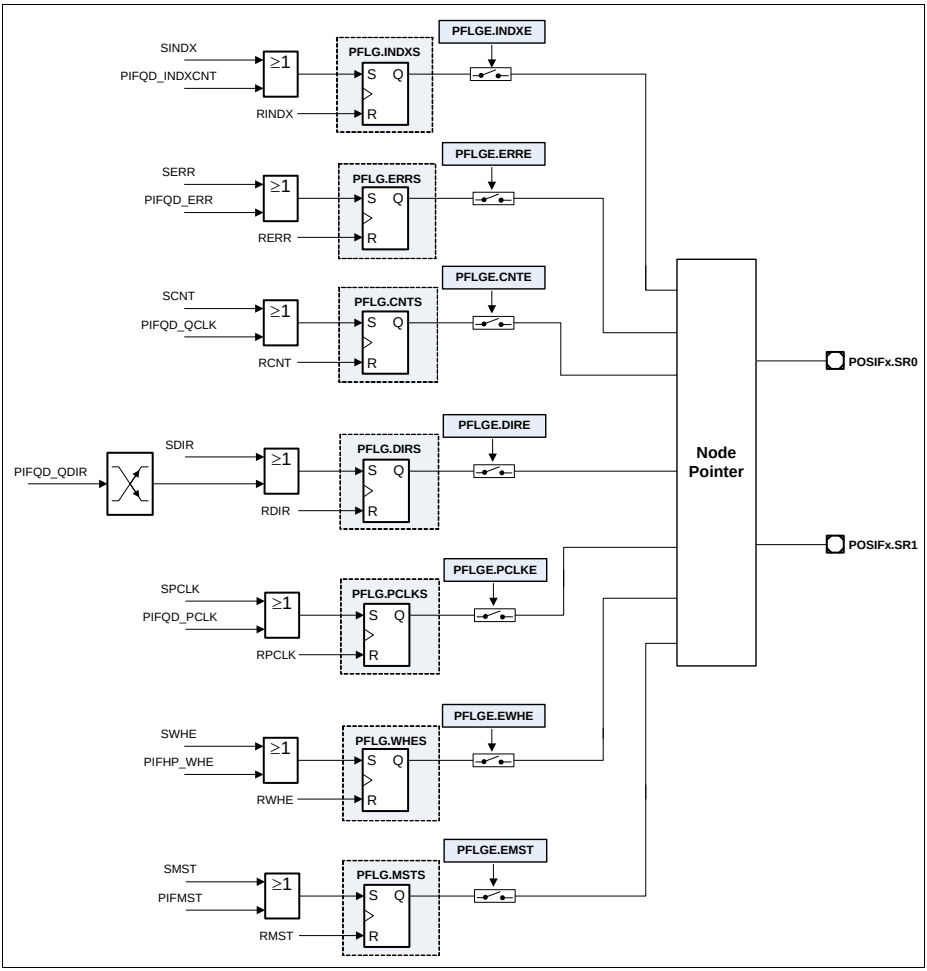



**Figure 21-25 Quadrature Decoder flags**


Subject to Agreement on the Use of Product Information

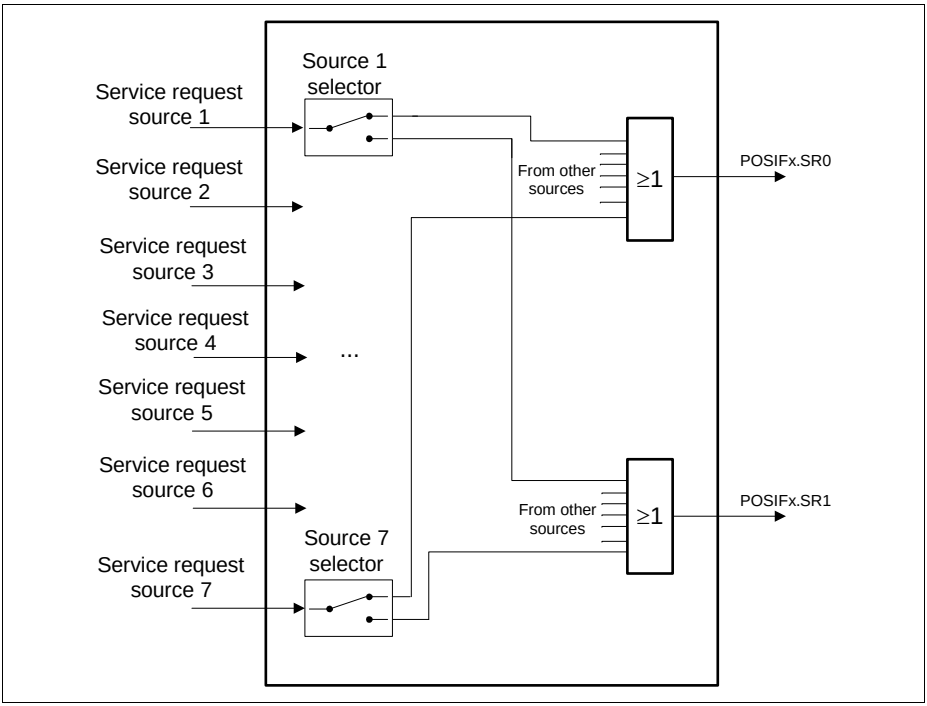



**Figure 21-26 Interrupt node pointer overview - quadrature decoder mode**

## 21.4 Debug Behavior

In suspend mode, the entire block is halted. The registers can still be accessed by the CPU (read only). This mode is useful for debugging purposes, e.g., where the current device status should be frozen in order to get a snapshot of the internal values. In suspend mode, all counters are stopped. The suspend mode is non-intrusive concerning the register bits. This means register bits are not modified by hardware when entering or leaving the suspend mode.

Entry into suspend mode can be configured at the kernel level by means of the register **PSUS**.

The module is only functional after the suspend signal becomes inactive.

## 21.5 Power, Reset and Clock

The following sections describe the operating conditions, characteristics and timing requirements for the POSIF. All the timing information is related to the module clock, $f_{posif}$.

### 21.5.1 Clocks

**Module Clock**

The module clock of the POSIF module is described in the SCU chapter as $f_{PCLK}$.

The bus interface clock of the POSIF module is described in the SCU chapter as $f_{MCLK}$.

It is possible to disable the module clock for the POSIF, via a specific system control bit, nevertheless, there may be a dependency on the $f_{posif}$ through the different POSIF instances or Capture/Compare Units. One should address the SCU Chapter for a complete description of the product clock scheme.

**External Signals**

The maximum frequency for the external signals, linked with the Hall Sensor and Rotary Encoder inputs can be seen in **Table 21-4**.

**Table 21-4 External Hall/Rotary signals operating conditions**

| Parameter | Symbol | Values | | | Unit | Note / Test Condition |
|---|---|---|---|---|---|---|
| | | Min. | Typ. | Max. | | |
| Frequency | $f_{esig}$ | – | – | $f_{posif}/4$ | MHz | |
| ON time | $ton_{esig}$ | $2T_{posif}$ | – | – | ns | |
| OFF time | $toff_{esig}$ | $2T_{posif}$ | – | – | ns | |

### 21.5.2 Power

The POSIF is inside the power core domain, therefore no special considerations about power up or power down sequences need to be taken. For a explanation about the different power domains, please address the SCU (System Control Unit) chapter.

## 21.6 Initialization and System Dependencies

### 21.6.1 Initialization

The initialization sequence for an application that is using the POSIF, should be the following:


Subject to Agreement on the Use of Product Information

**1st Step:** Enable the POSIF clock via the specific SCU register, CGATCLR0.

**2nd Step:** Configure the POSIF operation mode, **PCONF** register.

**3rd Step:** Configure the POSIF registers linked to the wanted operation mode:

- For Hall Sensor Mode: **HALPS**/**HALP**, **MCSM**/**MCM**
- For Quadrature Decoder Mode: **QDC**
- For stand-alone Multichannel Mode: **MCSM**/**MCM**

**4th Step:** Apply the pre programmed patterns into the **HALP** and **MCM** registers, by writing $1_B$ into the **MCMS**.STHR and **MCMS**.STMR fields.

**5th Step:** Configure the POSIF interrupts/service requests via the **PFLGE** register.

**6th Step:** Configure the Capture/Compare Units associated with the POSIF operation mode.

**7th Step:** If the synchronous start function of the POSIF is being used inside the associated Capture/Compare units, then one just needs to set the run bit of the module, by writing a $1_B$ into **PRUNS**.SRB.

**8th Step:** If the synchronous start function of the POSIF is not being used inside the associated Capture/Compare units, then start first the POSIF by writing a $1_B$ into **PRUNS**.SRB. After that, start the associated Capture/Compare Unit timer slices, by addressing the specific bit field inside each timer slice or by using the synchronous start function available on the SCU, CCUCON register.

*Note: Due to different startup conditions of the motor system itself (as well SW control loop implementation) it is possible to receive some erroneous interrupt triggers before the SW and HW loop are completely locked. To overcome this, the SW can ignore the interrupts during start up or the interrupt sources can be enabled only after a proper lock between HW and SW has been sensed.*

### 21.6.2 System Dependencies

Each POSIF may have different dependencies regarding module and bus clock frequencies. This dependencies should be addressed in the SCU and System Architecture Chapters.

Dependencies between several peripherals, regarding different clock operating frequencies may also exist. This should be addressed before configuring the connectivity between the POSIF and some other peripheral.

The following topics must be taken into consideration for good POSIF and system operation:

- POSIF module clock must be at maximum two times faster than the module bus interface clock
- Module input triggers for the POSIF must not exceed the module clock frequency (if the triggers are generated internally in the device)


Subject to Agreement on the Use of Product Information

- Module input triggers for the POSIF must not exceed the frequency dictated in **Section 21.5.1**
- Frequency of the POSIF outputs used as triggers/functions on other modules, must be crosschecked on the end point
- Applying and removing POSIF from reset, can cause unwanted operations in other modules. This can occur if the modules are using POSIF outputs as triggers/functions.

## 21.7 Registers

### Registers Overview

The absolute register address is calculated by adding:

Module Base Address + Offset Address

**Table 21-5 Registers Address Space**

| Module | Base Address | End Address | Note |
|---|---|---|---|
| POSIF0 | $50010000_H$ | $5001FFFF_H$ | |

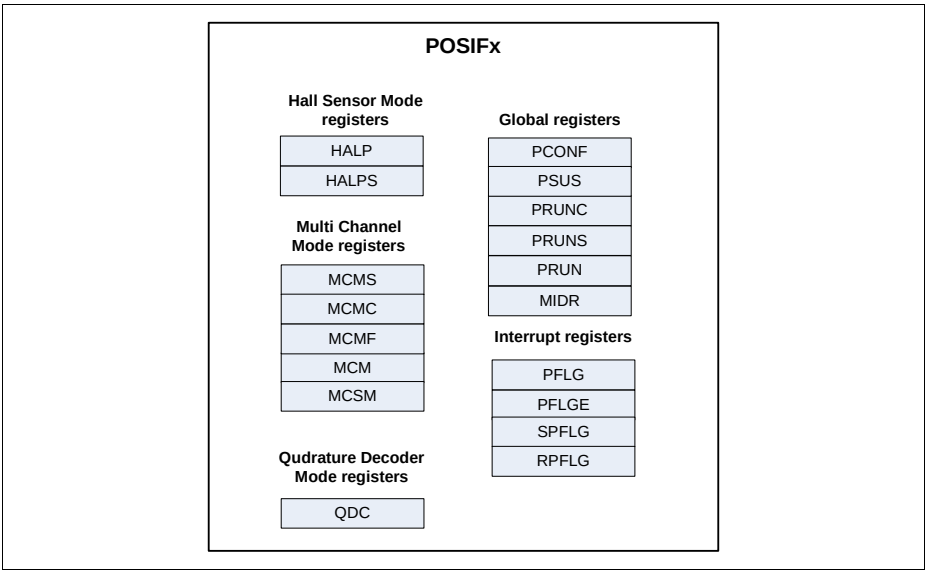



**Figure 21-27 POSIF registers overview**

**Table 21-6 Register Overview of POSIF**

| Short Name | Description | Offset Addr.[1)] | Access Mode | | Description See |
|---|---|---|---|---|---|
| | | | Read | Write | |
| POSIF Kernel Registers | | | | | |
| PCONF | Global control register | $0000_H$ | U, PV | U, PV | Page 21-38 |

**Table 21-6 Register Overview of POSIF** (cont'd)

| Short Name | Description | Offset Addr.[1)] | Access Mode | | Description See |
|---|---|---|---|---|---|
| | | | Read | Write | |
| PSUS | Suspend Configuration | $0004_H$ | U, PV | U, PV | Page 21-42 |
| PRUNS | POSIF run bit set | $0008_H$ | U, PV | U, PV | Page 21-43 |
| PRUNC | POSIF run bit clear | $000C_H$ | U, PV | U, PV | Page 21-43 |
| PRUN | POSIF run bit status | $0010_H$ | U, PV | BE | Page 21-44 |
| PDBG | Debug Design Register | $0100_H$ | U, PV | BE | Page 21-45 |
| MIDR | Module Identification register | $0020_H$ | U, PV | BE | Page 21-46 |
| Hall Sensor Mode Registers | | | | | |
| HALP | Hall Current and Expected patterns | $0030_H$ | U, PV | BE | Page 21-47 |
| HALPS | Hall Current and Expected shadow patterns | $0034_H$ | U, PV | U, PV | Page 21-48 |
| Multi-Channel Mode Register | | | | | |
| MCM | Multi-Channel Mode Pattern | $0040_H$ | U, PV | BE | Page 21-49 |
| MCSM | Multi-Channel Mode shadow Pattern | $0044_H$ | U, PV | U, PV | Page 21-50 |
| MCMS | Multi-Channel Mode Control set | $0048_H$ | U, PV | U, PV | Page 21-50 |
| MCMC | Multi-Channel Mode Control clear | $004C_H$ | U, PV | U, PV | Page 21-51 |
| MCMF | Multi-Channel Mode flag status | $0050_H$ | U, PV | BE | Page 21-52 |
| Quadrature Decoder Mode Register | | | | | |
| QDC | Quadrature Decoder Configuration | $0060_H$ | U, PV | U, PV | Page 21-53 |
| Interrupt Registers | | | | | |
| PFLG | POSIF interrupt status | $0070_H$ | U, PV | BE | Page 21-54 |
| PFLGE | POSIF interrupt enable | $0074_H$ | U, PV | U, PV | Page 21-56 |


Subject to Agreement on the Use of Product Information

**Table 21-6 Register Overview of POSIF** (cont'd)

| Short Name | Description | Offset Addr.[1] | Access Mode | | Description See |
|---|---|---|---|---|---|
| | | | Read | Write | |
| SPFLG | Interrupt set register | $0078_H$ | U, PV | U, PV | Page 21-59 |
| RPFLG | Interrupt clear register | $007C_H$ | U, PV | U, PV | Page 21-60 |

1) The absolute register address is calculated as follows:
Module Base Address + Offset Address (shown in this column)

## 21.7.1 Global registers

### PCONF

The register contains the global configuration for the POSIF operation: operation mode, input selection, filter configuration.

**PCONF**
**POSIF configuration** **($0000_H$)** **Reset Value: $00000000_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | LPC | | EWIL | EWIE | EWIS | | MSYNS | | MSES | MSETS | | | SPES | DSEL |
| r | | rw | | rw | rw | rw | | rw | | rw | rw | | | rw | rw |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | INSEL2 | | INSEL1 | | INSEL0 | | 0 | | MCUE | HIDG | 0 | QDCM | FSEL | |
| r | | rw | | rw | | rw | | r | | rw | rw | r | rw | rw | |

| Field | Bits | Type | Description |
|---|---|---|---|
| FSEL | [1:0] | rw | **Function Selector**<br>$00_B$ Hall Sensor Mode enabled<br>$01_B$ Quadrature Decoder Mode enabled<br>$10_B$ stand-alone Multi-Channel Mode enabled<br>$11_B$ Quadrature Decoder and stand-alone Multi-Channel Mode enabled |

| Field | Bits | Type | Description |
|---|---|---|---|
| **QDCM** | 2 | rw | **Position Decoder Mode selection**<br>This field selects if the Position Decoder block is in Quadrature Mode or Direction Count Mode.<br>In Quadrature mode, the position encoder is providing the phase signals, while in Direction Count Mode is providing a clock and a direction signal.<br>$0_B$ Position encoder is in Quadrature Mode<br>$1_B$ Position encoder is in Direction Count Mode. |
| **HIDG** | 4 | rw | **Idle generation enable**<br>Setting this field to $1_B$ disables the generation of the IDLE signal that forces a clear on the Multi-Channel pattern and run bit. |
| MCUE | 5 | rw | **Multi-Channel Pattern SW update enable**<br>$0_B$ Multi-Channel pattern update is controlled via HW<br>$1_B$ Multi-Channel pattern update is controlled via SW |
| INSEL0 | [9:8] | rw | **PhaseA/Hal input 1 selector**<br>This fields selects which input is used for the Phase A or Hall input 1 function (dependent if the module is set for Quadrature Decoder or Hall Sensor Mode):<br>$00_B$ POSIFx.IN0A<br>$01_B$ POSIFx.IN0B<br>$10_B$ POSIFx.IN0C<br>$11_B$ POSIFx.IN0D |
| INSEL1 | [11:10] | rw | **PhaseB/Hall input 2 selector**<br>This fields selects which input is used for the Phase B or Hall input 2 function (dependent if the module is set for Quadrature Decoder or Hall Sensor Mode):<br>$00_B$ POSIFx.IN1A<br>$01_B$ POSIFx.IN1B<br>$10_B$ POSIFx.IN1C<br>$11_B$ POSIFx.IN1D |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| INSEL2 | [13:12] | rw | **Index/Hall input 3 selector**<br>This fields selects which input is used for the Index or Hall input 3 function (dependent if the module is set for Quadrature Decoder or Hall Sensor Mode):<br>$00_B$ POSIFx.IN2A<br>$01_B$ POSIFx.IN2B<br>$10_B$ POSIFx.IN2C<br>$11_B$ POSIFx.IN2D |
| DSEL | 16 | rw | **Delay Pin selector**<br>This field selects which input is used to trigger the end of the delay between the detection of an edge in the Hall inputs and the actual sample of the Hall inputs.<br>$0_B$ POSIFx.HSDA<br>$1_B$ POSIFx.HSDB |
| SPES | 17 | rw | **Edge selector for the sampling trigger**<br>This field selects which edge is used of the selected POSIFx.HSD[B...A] signal to trigger a sample of the Hall inputs.<br>$0_B$ Rising edge<br>$1_B$ Falling edge |
| MSETS | [20:18] | rw | **Pattern update signal select**<br>Selects the input signal that is used to enable a Multi-Channel pattern update.<br>$000_B$ POSIFx.MSETA<br>$001_B$ POSIFx.MSETB<br>$010_B$ POSIFx.MSETC<br>$011_B$ POSIFx.MSETD<br>$100_B$ POSIFx.MSETE<br>$101_B$ POSIFx.MSETF<br>$110_B$ POSIFx.MSETG<br>$111_B$ POSIFx.MSETH |
| MSES | 21 | rw | **Multi-Channel pattern update trigger edge**<br>$0_B$ The signal used to enable a pattern update is active on the rising edge<br>$1_B$ The signal used to enable a pattern update is active on the falling edge |


Subject to Agreement on the Use of Product Information

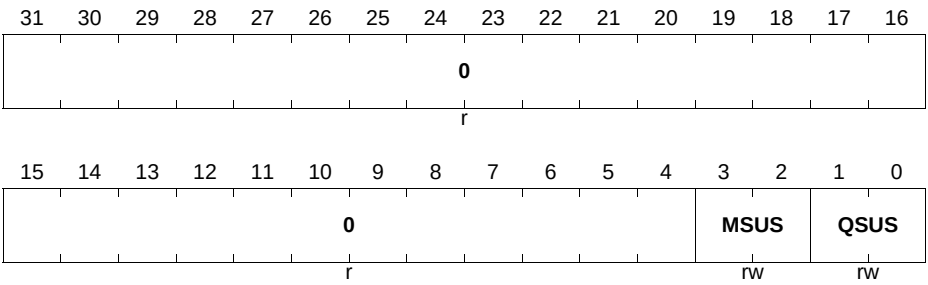
| Field | Bits | Type | Description |
|---|---|---|---|
| MSYNS | [23:22] | rw | **PWM synchronization signal selector**<br>This fields selects which input is used as trigger for Multi-Channel pattern update (synchronization with the PWM signal).<br>$00_B$ POSIFx.MSYNCA<br>$01_B$ POSIFx.MSYNCB<br>$10_B$ POSIFx.MSYNCC<br>$11_B$ POSIFx.MSYNCD |
| EWIS | [25:24] | rw | **Wrong Hall Event selection**<br>$00_B$ POSIFx.EWHEA<br>$01_B$ POSIFx.EWHEB<br>$10_B$ POSIFx.EWHEC<br>$11_B$ POSIFx.EWHED |
| EWIE | 26 | rw | **External Wrong Hall Event enable**<br>$0_B$ External wrong hall event emulation signal, POSIFx.EWHE[D...A], is disabled<br>$1_B$ External wrong hall event emulation signal, POSIFx.EWHE[D...A], is enabled. |
| EWIL | 27 | rw | **External Wrong Hall Event active level**<br>$0_B$ POSIFx.EWHE[D...A] signal is active HIGH<br>$1_B$ POSIFx.EWHE[D...A] signal is active LOW |
| LPC | [30:28] | rw | **Low Pass Filters Configuration**<br>$000_B$ Low pass filter disabled<br>$001_B$ Low pass of 1 clock cycle<br>$010_B$ Low pass of 2 clock cycles<br>$011_B$ Low pass of 4 clock cycles<br>$100_B$ Low pass of 8 clock cycles<br>$101_B$ Low pass of 16 clock cycles<br>$110_B$ Low pass of 32 clock cycles<br>$111_B$ Low pass of 64 clock cycles |
| 0 | 3, [7:6], [15:14], 31 | r | **Reserved**<br>Read always returns 0 |

### PSUS

The register contains the suspend configuration for the POSIF.

**PSUS**

**POSIF Suspend Config** **($0004_H$)** **Reset Value: $00000000_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **QSUS** | [1:0] | rw | **Quadrature Mode Suspend Config**<br>This field controls the entering in suspend for the quadrature decoder mode.<br>$00_B$ Suspend request ignored<br>$01_B$ Stop immediately<br>$10_B$ Suspend in the next index occurrence<br>$11_B$ Suspend in the next phase (PhaseA or PhaseB) occurrence |
| **MSUS** | [3:2] | rw | **Multi-Channel Mode Suspend Config**<br>This field controls the entering in suspend for the Multi-Channel mode. The Hall sensor mode is also covered by this configuration.<br>$00_B$ Suspend request ignored<br>$01_B$ Stop immediately. Multi-Channel pattern is not set to the reset value.<br>$10_B$ Stop immediately. Multi-Channel pattern is set to the reset value.<br>$11_B$ Suspend with the synchronization of the PWM signal. Multi-Channel pattern is set to the reset value at the same time of the synchronization. |
| **0** | [31:4] | r | **Reserved**<br>Read always returns 0 |

**PRUNS**

Via this register it is possible to set the run bit of the module.

**PRUNS**

**POSIF Run Bit Set** **($0008_H$)** **Reset Value: $00000000_H$**

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

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

| Field | Bits | Type | Description |
|---|---|---|---|
| **SRB** | 0 | w | **Set Run bit**<br>Writing an $1_B$ into this bit sets the run bit of the module.<br>Read always returns 0. |
| **0** | [31:1] | r | **Reserved**<br>Read always returns 0 |

**PRUNC**

Via this register it is possible to clear the run bit and the internal state machines of the module.

**PRUNC**

**POSIF Run Bit Clear** **($000C_H$)** **Reset Value: $00000000_H$**

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

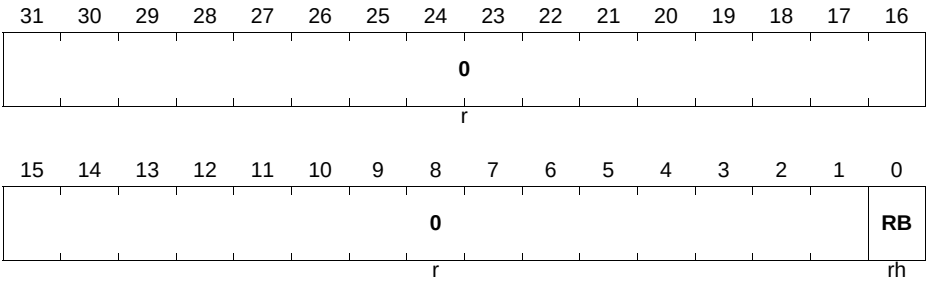
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | | | | | | | CSM | CRB |
| r | | | | | | | | | | | | | | w | w |

| Field | Bits | Type | Description |
|---|---|---|---|
| **CRB** | 0 | w | **Clear Run bit**<br>Writing an $1_B$ into this bit clears the run bit of the module. The module is stopped.<br>Read always returns 0. |
| **CSM** | 1 | w | **Clear Current internal status**<br>Writing an $1_B$ into this bit resets the state machine of the quadrature decoder and the current status of the Hall sensor and Multi-Channel mode registers. The flags and static configuration bit fields are not cleared.<br>Read always returns 0. |
| **0** | [31:2] | r | **Reserved**<br>Read always returns 0 |

### PRUN

The register contains the run bit status of the POSIF.

**PRUN**
**POSIF Run Bit Status** **($0010_H$)** **Reset Value: $00000000_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| RB | 0 | rh | **Run Bit**<br>This field indicates if the module is in running or IDLE state.<br>$0_B$ IDLE<br>$1_B$ Running |

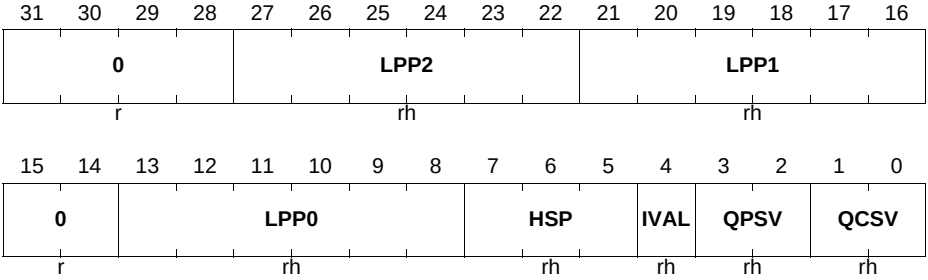

Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | [31:1] | r | **Reserved**<br>Read always returns 0 |

### PDBG

Debug register for current state of the POSIF state machines and Hall Sampled values.

**PDBG**
**POSIF Debug register** **($0100_H$)** **Reset Value: $00000000_H$**

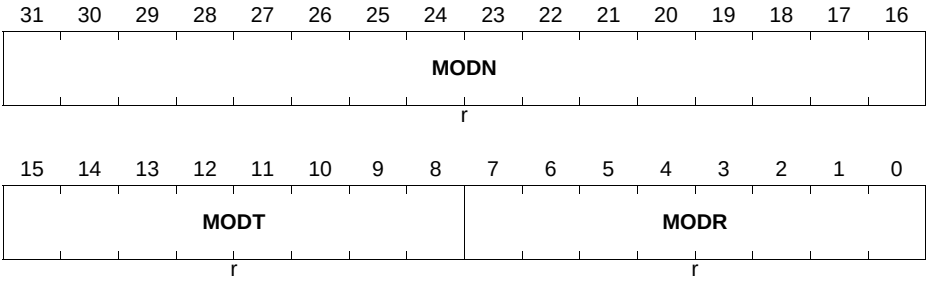



| Field | Bits | Type | Description |
|---|---|---|---|
| QCSV | [1:0] | rh | **Quadrature Decoder Current state**<br>QCSV[0] - Phase A<br>QCSV[1] - Phase B |
| QPSV | [3:2] | rh | **Quadrature Decoder Previous state**<br>QPSV[0] - Phase A<br>QPSV[1] - Phase B |
| IVAL | 4 | rh | **Current Index Value** |
| **HSP** | [7:5] | rh | **Hall Current Sampled Pattern**<br>HSP[0] - Hall Input 1<br>HSP[1] - Hall Input 2<br>HSP[2] - Hall Input 3 |
| LPP0 | [13:8] | rh | **Actual count of the Low Pass Filter for POSI0** |
| LPP1 | [21:16] | rh | **Actual count of the Low Pass Filter for POSI1** |
| LPP2 | [27:22] | rh | **Actual count of the Low Pass Filter for POSI2** |
| 0 | [31:28], [15:14] | r | **Reserved**<br>A read always returns 0. |

**MIDR**

This register contains the module identification number.

**MIDR**
**Module Identification register (0020$_H$) Reset Value: 00A8C0XX$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **MODR** | [7:0] | r | **Module Revision**<br>This bit field indicates the revision number of the module implementation (depending on the design step). |
| **MODT** | [15:8] | r | **Module Type** |
| **MODN** | [31:16] | r | **Module Number** |

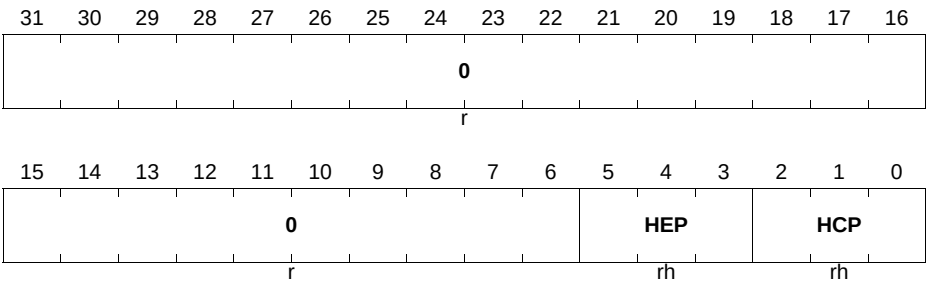
## 21.7.2 Hall Sensor Mode Registers

**HALP**

The register contains the values for the Hall Expected Pattern and Hall Current Pattern.

**HALP**

**Hall Sensor Patterns** **($0030_H$)** **Reset Value: $00000000_H$**

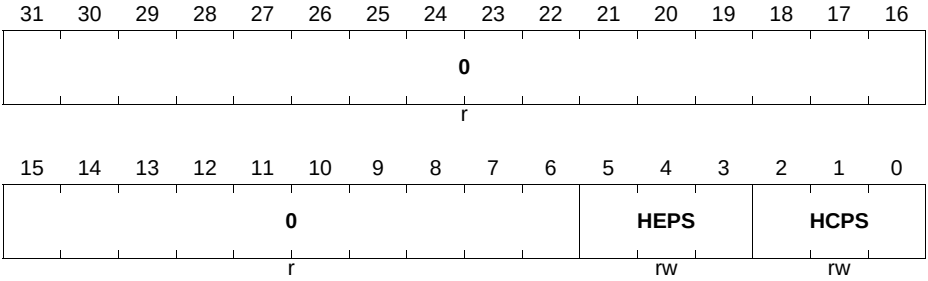



| Field | Bits | Type | Description |
|---|---|---|---|
| **HCP** | [2:0] | rh | **Hall Current Pattern**<br>This field contains the Hall Current pattern. This field is updated with the **HALPS**.HCPS value every time that a correct hall event occurs.<br>HCP[0] - Hall Input 1<br>HCP[1] - Hall Input 2<br>HCP[2] - Hall Input 3 |
| **HEP** | [5:3] | rh | **Hall Expected Pattern**<br>This field contains the Hall Expected pattern. This field is updated with the **HALPS**.HEPS values every time that a correct hall event occurs.<br>HEP[0] - Hall Input 1<br>HEP[1] - Hall Input 2<br>HEP[2] - Hall Input 3 |
| 0 | [31:6] | r | **Reserved**<br>Read always returns 0 |

**HALPS**

The register contains the values that are going to be loaded into the **HALP** register when the next correct hall event occurs.

**HALPS**
**Hall Sensor Shadow Patterns** **($0034_H$)** **Reset Value: $00000000_H$**



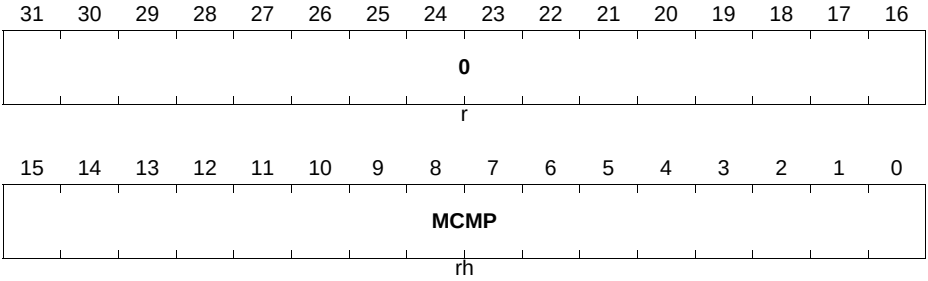

| Field | Bits | Type | Description |
|---|---|---|---|
| **HCPS** | [2:0] | rw | **Shadow Hall Current Pattern**<br>This field contains the next Hall Current pattern value. This field is set on the **HALP**.HCP field every time that a correct hall event occurs.<br>HCPS[0] - Hall Input 1<br>HCPS[1] - Hall Input 2<br>HCPS[2] - Hall Input 3 |
| **HEPS** | [5:3] | rw | **Shadow Hall expected Pattern**<br>This field contains the next Hall Expected pattern. This field is set on the **HALP**.HEP field every time that a correct hall event occurs.<br>HEPS[0] - Hall Input 1<br>HEPS[1] - Hall Input 2<br>HEPS[2] - Hall Input 3 |
| 0 | [31:6] | r | **Reserved**<br>Read always returns 0 |

## 21.7.3 Multi-Channel Mode Registers

**MCM**

The register contains the value of the Multi-Channel pattern that is applied to the outputs POSIFx.OUT[15:0].

**MCM**
**Multi-Channel Pattern** **(0040$_H$)** **Reset Value: 00000000$_H$**



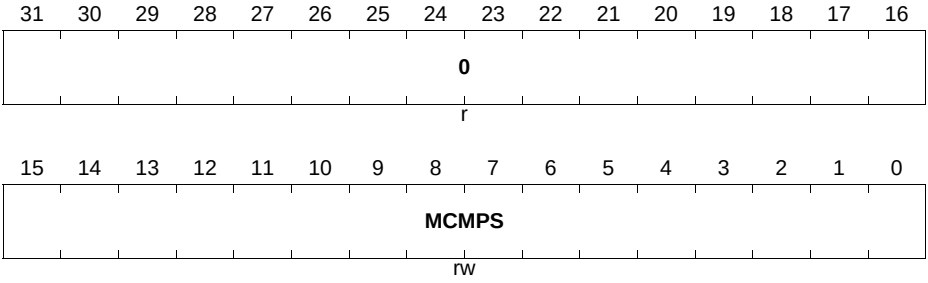

| Field | Bits | Type | Description |
|---|---|---|---|
| **MCMP** | [15:0] | rh | **Multi-Channel Pattern**<br>This field contains the Multi-Channel Pattern that is going to be applied to the Multi-Channel outputs, POSIFx.MOUT[15:0].<br>This field is updated with the value of the **MCSM**.MCMPS every time that a Multi-Channel pattern update is triggered. |
| **0** | [31:16] | r | **Reserved**<br>Read always returns 0 |

**MCSM**

The register contains the value that is going to be loaded into the **MCM** register when the next Multi-Channel update trigger occurs.


Subject to Agreement on the Use of Product Information

**MCSM**
**Multi-Channel Shadow Pattern** **($0044_H$)** **Reset Value: $00000000_H$**

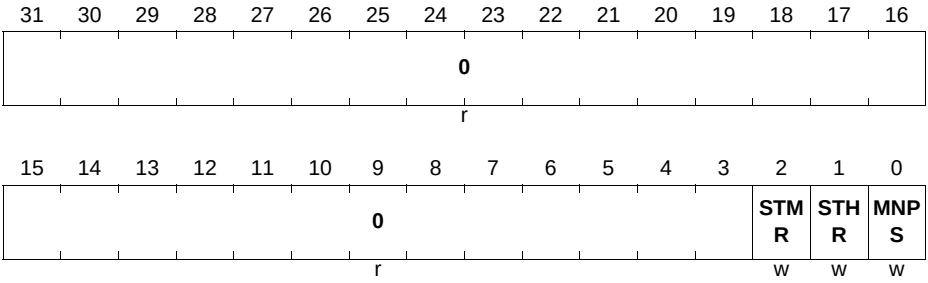



| Field | Bits | Type | Description |
|---|---|---|---|
| **MCMPS** | [15:0] | rw | **Shadow Multi-Channel Pattern**<br>This field contains the next Multi-Channel Pattern. Every time that a Multi-Channel pattern transfer is triggered, this value is passed into the field **MCM**.MCMP. |
| **0** | [31:16] | r | **Reserved**<br>Read always returns 0 |

**MCMS**

Through this register it is possible to request a Multi-Channel pattern update. It is also possible through this register to request an immediate update of the Multi-Channel and Hall Sensor patterns without waiting for the hardware trigger.

**MCMS**
**Multi-Channel Pattern Control set** **($0048_H$)** **Reset Value: $00000000_H$**





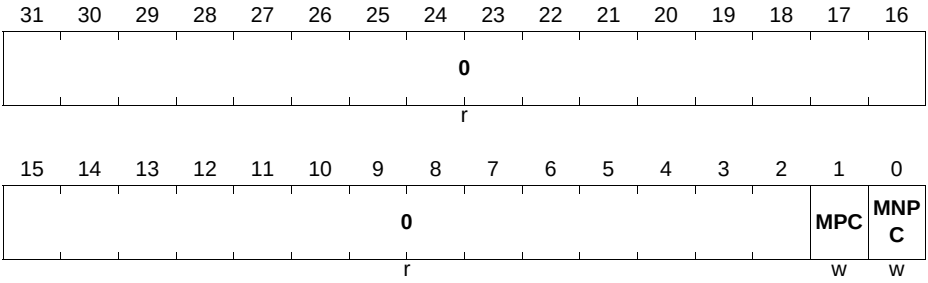
Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **MNPS** | 0 | w | **Multi-Channel Pattern Update Enable Set**<br>Writing a $1_B$ into this field enables the Multi-Channel pattern update (sets the **MCMF**.MSS bit). The update is not done immediately due to the fact that the trigger that synchronizes the update with the PWM is still needed.<br>A read always returns 0. |
| **STHR** | 1 | w | **Hall Pattern Shadow Transfer Request**<br>Writing a $1_B$ into this field leads to an immediate update of the fields **HALP**.HCP and **HALP**.HEP.<br>A read always returns 0. |
| **STMR** | 2 | w | **Multi-Channel Shadow Transfer Request**<br>Writing a $1_B$ into this field leads to an immediate update of the field **MCM**.MCMP.<br>A read always returns 0. |
| 0 | [31:3] | r | **Reserved**<br>Read always returns 0 |

### MCMC

Through this register is possible to cancel a Multi-Channel pattern update and to clear the Multi-Channel pattern to the default value.

**MCMC**
**Multi-Channel Pattern Control clear (004C$_H$)** **Reset Value: 00000000$_H$**

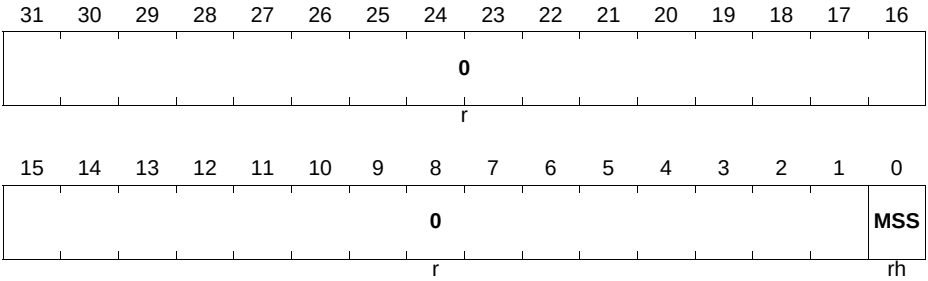


| Field | Bits | Type | Description |
|---|---|---|---|
| **MNPC** | 0 | w | **Multi-Channel Pattern Update Enable Clear**<br>Writing a $1_B$ into this field clears the **MCMF**.MSS bit.<br>A read always returns 0. |
| **MPC** | 1 | w | **Multi-Channel Pattern clear**<br>Writing a $1_B$ into this field clears the Multi-Channel Pattern value to $0000_H$.<br>A read always returns 0. |
| 0 | [31:2] | r | **Reserved**<br>Read always returns 0 |

**MCMF**

The register contains the status of the Multi-Channel update request.

**MCMF**
**Multi-Channel Pattern Control flag ($0050_H$)** **Reset Value: $00000000_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **MSS** | 0 | rh | **Multi-Channel Pattern update status**<br>This field indicates if the Multi-Channel pattern is ready to be updated or not. When this field is set, the Multi-Channel pattern is updated when the triggering signal, selected from the POSIFx.MSYNC[D...A], becomes active.<br>$0_B$ Update of the Multi-Channel pattern is set<br>$1_B$ Update of the Multi-Channel pattern is not set |
| 0 | [31:1] | r | **Reserved**<br>Read always returns 0 |

### 21.7.4 Quadrature Decoder Registers

**QDC**

The register contains the configuration for the operation of the Quadrature Decoder Mode.

**QDC**
**Quadrature Decoder Control** **($0060_H$)** **Reset Value: $00000000_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | DVAL | 0 | | ICM | | 0 | PHS | PBLS | PALS |
| r | | | | | | | rh | r | | rw | | r | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **PALS** | 0 | rw | **Phase A Level selector**<br>$0_B$ Phase A is active HIGH<br>$1_B$ Phase A is active LOW |
| **PBLS** | 1 | rw | **Phase B Level selector**<br>$0_B$ Phase B is active HIGH<br>$1_B$ Phase B is active LOW |
| **PHS** | 2 | rw | **Phase signals swap**<br>$0_B$ Phase A is the leading signal for clockwise rotation<br>$1_B$ Phase B is the leading signal for clockwise rotation |

| Field | Bits | Type | Description |
|---|---|---|---|
| ICM | [5:4] | rw | **Index Marker generations control**<br>This field controls the generation of the index marker that is linked to the output pin POSIFx.OUT3.<br>$00_B$ No index marker generation on POSIFx.OUT3<br>$01_B$ Only first index occurrence generated on POSIFx.OUT3<br>$10_B$ All index occurrences generated on POSIFx.OUT3<br>$11_B$ Reserved |
| DVAL | 8 | rh | **Current rotation direction**<br>$0_B$ Counterclockwise rotation<br>$1_B$ Clockwise rotation |
| 0 | 3, [7:6], [31:9] | r | **Reserved**<br>Read always returns 0 |

## 21.7.5 Interrupt Registers

### PFLG

The register contains the status of all the interrupt flags of the module.

**PFLG**
**POSIF Interrupt Flags** **($0070_H$)** **Reset Value: $00000000_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | PCLKS | DIRS | CNTS | ERRS | INDXS | 0 | | | MSTS | 0 | HIES | WHES | CHES |
| r | | | rh | rh | rh | rh | rh | r | | | rh | r | rh | rh | rh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **CHES** | 0 | rh | **Correct Hall Event Status**<br>$0_B$ Correct Hall Event not detected<br>$1_B$ Correct Hall Event detected |

| Field | Bits | Type | Description |
|---|---|---|---|
| **WHES** | 1 | rh | **Wrong Hall Event Status**<br>$0_B$ Wrong Hall Event not detected<br>$1_B$ Wrong Hall Event detected |
| **HIES** | 2 | rh | **Hall Inputs Update Status**<br>$0_B$ Transition on the Hall Inputs not detected<br>$1_B$ Transition on the Hall Inputs detected |
| MSTS | 4 | rh | **Multi-Channel pattern shadow transfer status**<br>$0_B$ Shadow transfer not done<br>$1_B$ Shadow transfer done |
| INDXS | 8 | rh | **Quadrature Index Status**<br>$0_B$ Index event not detected<br>$1_B$ Index event detected |
| ERRS | 9 | rh | **Quadrature Phase Error Status**<br>$0_B$ Phase Error event not detected<br>$1_B$ Phase Error event detected |
| CNTS | 10 | rh | **Quadrature CLK Status**<br>$0_B$ Quadrature clock not generated<br>$1_B$ Quadrature clock generated |
| DIRS | 11 | rh | **Quadrature Direction Change**<br>$0_B$ Change on direction not detected<br>$1_B$ Change on direction detected |
| PCLKS | 12 | rh | **Quadrature Period Clk Status**<br>$0_B$ Period clock not generated<br>$1_B$ Period clock generated |
| 0 | 3, [7:5], [31:13] | r | **Reserved**<br>Read always returns 0 |

**PFLGE**

Through this register it is possible to enable or disable each of the available interrupt sources. It is also possible to select to which service request line an interrupt is forward.


Subject to Agreement on the Use of Product Information

**PFLGE**
**POSIF Interrupt Enable** **($0074_H$)** **Reset Value: $00000000_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | PCLSEL | DIRSEL | CNTSEL | ERRSEL | INDSEL | 0 | | | MSTSEL | 0 | HIESEL | WHESEL | CHESEL |
| | r | | rw | rw | rw | rw | rw | | r | | rw | r | rw | rw | rw |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | EPCLK | EDIR | ECNT | EERR | EINDX | 0 | | | EMST | 0 | EHIE | EWHE | ECHE |
| | r | | rw | rw | rw | rw | rw | | r | | rw | r | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **ECHE** | 0 | rw | **Correct Hall Event Enable**<br>$0_B$ Correct Hall Event interrupt disabled<br>$1_B$ Correct Hall Event interrupt enabled |
| **EWHE** | 1 | rw | **Wrong Hall Event Enable**<br>$0_B$ Wrong Hall Event interrupt disabled<br>$1_B$ Wrong Hall Event interrupt enabled |
| **EHIE** | 2 | rw | **Hall Input Update Enable**<br>$0_B$ Update of the Hall Inputs interrupt is disabled<br>$1_B$ Update of the Hall Inputs interrupt is enabled |
| EMST | 4 | rw | **Multi-Channel pattern shadow transfer enable**<br>$0_B$ Shadow transfer event interrupt disabled<br>$1_B$ Shadow transfer event interrupt enabled |
| EINDX | 8 | rw | **Quadrature Index Event Enable**<br>$0_B$ Index event interrupt disabled<br>$1_B$ Index event interrupt enabled |
| EERR | 9 | rw | **Quadrature Phase Error Enable**<br>$0_B$ Phase error event interrupt disabled<br>$1_B$ Phase error event interrupt enabled |
| ECNT | 10 | rw | **Quadrature CLK interrupt Enable**<br>$0_B$ Quadrature CLK event interrupt disabled<br>$1_B$ Quadrature CLK event interrupt enabled |
| EDIR | 11 | rw | **Quadrature direction change interrupt Enable**<br>$0_B$ Direction change event interrupt disabled<br>$1_B$ Direction change event interrupt enabled |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| EPCLK | 12 | rw | **Quadrature Period CLK interrupt Enable**<br>$0_B$ Quadrature Period CLK event interrupt disabled<br>$1_B$ Quadrature Period CLK event interrupt enabled |
| CHESEL | 16 | rw | **Correct Hall Event Service Request Selector**<br>$0_B$ Correct Hall Event interrupt forward to POSIFx.SR0<br>$1_B$ Correct Hall Event interrupt forward to POSIFx.SR1 |
| WHESEL | 17 | rw | **Wrong Hall Event Service Request Selector**<br>$0_B$ Wrong Hall Event interrupt forward to POSIFx.SR0<br>$1_B$ Wrong Hall Event interrupt forward to POSIFx.SR1 |
| HIESEL | 18 | rw | **Hall Inputs Update Event Service Request Selector**<br>$0_B$ Hall Inputs Update Event interrupt forward to POSIFx.SR0<br>$1_B$ Hall Inputs Update Event interrupt forward to POSIFx.SR1 |
| MSTSEL | 20 | rw | **Multi-Channel pattern Update Event Service Request Selector**<br>$0_B$ Multi-Channel pattern Update Event interrupt forward to POSIFx.SR0<br>$1_B$ Multi-Channel pattern Update Event interrupt forward to POSIFx.SR1 |
| INDSEL | 24 | rw | **Quadrature Index Event Service Request Selector**<br>$0_B$ Quadrature Index Event interrupt forward to POSIFx.SR0<br>$1_B$ Quadrature Index Event interrupt forward to POSIFx.SR1 |
| ERRSEL | 25 | rw | **Quadrature Phase Error Event Service Request Selector**<br>$0_B$ Quadrature Phase error Event interrupt forward to POSIFx.SR0<br>$1_B$ Quadrature Phase error Event interrupt forward to POSIFx.SR1 |

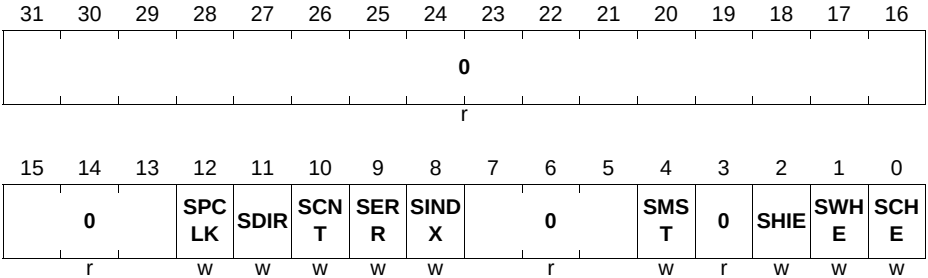

Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| CNTSEL | 26 | rw | **Quadrature Clock Event Service Request Selector**<br>$0_B$ Quadrature Clock Event interrupt forward to POSIFx.SR0<br>$1_B$ Quadrature Clock Event interrupt forward to POSIFx.SR1 |
| DIRSEL | 27 | rw | **Quadrature Direction Update Event Service Request Selector**<br>$0_B$ Quadrature Direction Update Event interrupt forward to POSIFx.SR0<br>$1_B$ Quadrature Direction Update Event interrupt forward to POSIFx.SR1 |
| PCLSEL | 28 | rw | **Quadrature Period clock Event Service Request Selector**<br>$0_B$ Quadrature Period clock Event interrupt forward to POSIFx.SR0<br>$1_B$ Quadrature Period clock Event interrupt forward to POSIFx.SR1 |
| 0 | 3, [7:5], [15:13], 19, [23:21], [31:29] | r | **Reserved**<br>Read always returns 0 |

**SPFLG**

Through this register it is possible for the SW to set a specific interrupt status flag.

**SPFLG**
**POSIF Interrupt Set** **($0078_H$)** **Reset Value: $00000000_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **SCHE** | 0 | w | **Correct Hall Event flag set**<br>Writing a $1_B$ to this field sets the **PFLG**.CHES bit field. An interrupt pulse is generated.<br>A read always returns 0. |
| **SWHE** | 1 | w | **Wrong Hall Event flag set**<br>Writing a $1_B$ to this field sets the **PFLG**.WHES bit field. An interrupt pulse is generated.<br>A read always returns 0. |
| **SHIE** | 2 | w | **Hall Inputs Update Event flag set**<br>Writing a $1_B$ to this field sets the **PFLG**.HIES bit field. An interrupt pulse is generated.<br>A read always returns 0. |
| SMST | 4 | w | **Multi-Channel Pattern shadow transfer flag set**<br>Writing a $1_B$ to this field sets the **PFLG**.MSTS bit field. An interrupt pulse is generated.<br>A read always returns 0. |
| SINDX | 8 | w | **Quadrature Index flag set**<br>Writing a $1_B$ to this field sets the **PFLG**.INDXS bit field. An interrupt pulse is generated.<br>A read always returns 0. |
| SERR | 9 | w | **Quadrature Phase Error flag set**<br>Writing a $1_B$ to this field sets the **PFLG**.ERRS bit field. An interrupt pulse is generated.<br>A read always returns 0. |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| SCNT | 10 | w | **Quadrature CLK flag set**<br>Writing a $1_B$ to this field sets the **PFLG**.CNTS bit field.<br>An interrupt pulse is generated.<br>A read always returns 0. |
| SDIR | 11 | w | **Quadrature Direction flag set**<br>Writing a $1_B$ to this field sets the **PFLG**.DIRS bit field.<br>An interrupt pulse is generated.<br>A read always returns 0. |
| SPCLK | 12 | w | **Quadrature period clock flag set**<br>Writing a $1_B$ to this field sets the **PFLG**.PCLKS bit field. An interrupt pulse is generated.<br>A read always returns 0. |
| 0 | 3, [7:5], [31:13] | r | **Reserved**<br>Read always returns 0 |

### RPFLG

Through this register it is possible for the SW to reset/clear a specific interrupt status flag.

**RPFLG**
**POSIF Interrupt Clear** **($007C_H$)** **Reset Value: $00000000_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | RPCLK | RDIR | RCNT | RERR | RINDX | 0 | | | RMST | 0 | RHIE | RWHE | RCHE |
| r | | | w | w | w | w | w | r | | | w | r | w | w | w |

| Field | Bits | Type | Description |
|---|---|---|---|
| **RCHE** | 0 | w | **Correct Hall Event flag clear**<br>Writing a $1_B$ to this field clears the **PFLG**.CHES bit field.<br>A read always returns 0. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **RWHE** | 1 | w | **Wrong Hall Event flag clear**<br>Writing a $1_B$ to this field clears the **PFLG**.WHES bit field.<br>A read always returns 0. |
| **RHIE** | 2 | w | **Hall Inputs Update Event flag clear**<br>Writing a $1_B$ to this field clears the **PFLG**.HIES bit field.<br>A read always returns 0. |
| RMST | 4 | w | **Multi-Channel Pattern shadow transfer flag clear**<br>Writing a $1_B$ to this field clears the **PFLG**.MSTS bit field.<br>A read always returns 0. |
| RINDX | 8 | w | **Quadrature Index flag clear**<br>Writing a $1_B$ to this field clears the **PFLG**.INDXS bit field.<br>A read always returns 0. |
| RERR | 9 | w | **Quadrature Phase Error flag clear**<br>Writing a $1_B$ to this field clears the **PFLG**.ERRS bit field.<br>A read always returns 0. |
| RCNT | 10 | w | **Quadrature CLK flag clear**<br>Writing a $1_B$ to this field clears the **PFLG**.CNTS bit field.<br>A read always returns 0. |
| RDIR | 11 | w | **Quadrature Direction flag clear**<br>Writing a $1_B$ to this field clears the **PFLG**.DIRS bit field.<br>A read always returns 0. |
| RPCLK | 12 | w | **Quadrature period clock flag clear**<br>Writing a $1_B$ to this field clears the **PFLG**.PCLKS bit field.<br>A read always returns 0. |
| 0 | 3, [7:5], [31:13] | r | **Reserved**<br>Read always returns 0 |

## 21.8 Interconnects

The following tables, describe the connectivity present on the device for each POSIF module.

The GPIO connections are available at the Ports chapter.

### 21.8.1 POSIF0 pins

**Table 21-7 POSIF0 Pin Connections**

| Global Inputs/Outputs | I/O | Connected To | Description |
|---|---|---|---|
| POSIF0.CLK | I | 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 |
| 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 |


Subject to Agreement on the Use of Product Information

**Table 21-7 POSIF0 Pin Connections**

| Global Inputs/Outputs | I/O | Connected To | Description |
|---|---|---|---|
| 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 | not connected | 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 |
| 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 | not connected | 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 |


Subject to Agreement on the Use of Product Information

**Table 21-7 POSIF0 Pin Connections**

| Global Inputs/Outputs | I/O | Connected To | Description |
|---|---|---|---|
| POSIF0.MSYNCD | I | not connected | Sync for updating the multi channel pattern with the shadow transfer |
| POSIF0.OUT0 | O | CCU40.IN0E; CCU40.IN1E; | Quadrature mode: Quadrature clock; Hall sensor mode: Hall input edge detection |
| POSIF0.OUT1 | O | CCU40.IN0F; CCU40.IN1F; CCU40.IN2E; | Quadrature mode: Shaft direction; Hall sensor mode: Correct Hall Event |
| POSIF0.OUT2 | O | CCU40.IN2F; CCU80.IN0D; CCU80.IN1D; CCU80.IN2D; CCU80.IN3D; | Quadrature mode: Pclk for velocity; Hall sensor mode: Idle/wrong hall event |
| POSIF0.OUT3 | O | CCU40.IN0G; CCU40.IN1G; CCU40.IN2G; CCU40.IN3E; | Quadrature mode: Index event used for Clear/capt; Hall sensor mode: stop in Hall sensor mode |
| POSIF0.OUT4 | O | CCU40.IN1H; CCU40.IN2H; | Quadrature mode:Index event; Hall sensor mode: Multi channel pattern update done |
| POSIF0.OUT5 | O | CCU40.IN3F; CCU80.IN0E; CCU80.IN1E; CCU80.IN2E; CCU80.IN3E; | Sync start |
| POSIF0.OUT6 | O | CCU40.MCSS; CCU80.MCSS; | Multi channel pattern update request |
| POSIF0.MOUT[0] | O | CCU80.MCI00; | Multi channel pattern |
| POSIF0.MOUT[1] | O | CCU80.MCI01; | Multi channel pattern |
| POSIF0.MOUT[2] | O | CCU80.MCI02; | Multi channel pattern |
| POSIF0.MOUT[3] | O | CCU80.MCI03; | Multi channel pattern |
| POSIF0.MOUT[4] | O | CCU80.MCI10; | Multi channel pattern |
| POSIF0.MOUT[5] | O | CCU80.MCI11; | Multi channel pattern |
| POSIF0.MOUT[6] | O | CCU80.MCI12; | Multi channel pattern |

**Table 21-7 POSIF0 Pin Connections**

| Global Inputs/Outputs | I/O | Connected To | Description |
|---|---|---|---|
| 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;<br>VADC0.BGREQTRO;<br>VADC0.G0REQTRO;<br>VADC0.G1REQTRO; | Service request line 1 |


Subject to Agreement on the Use of Product Information

# 22 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.

## 22.1 Overview

The BCCU in XMC1300 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.

### 22.1.1 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

## 22.2 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.


Subject to Agreement on the Use of Product Information

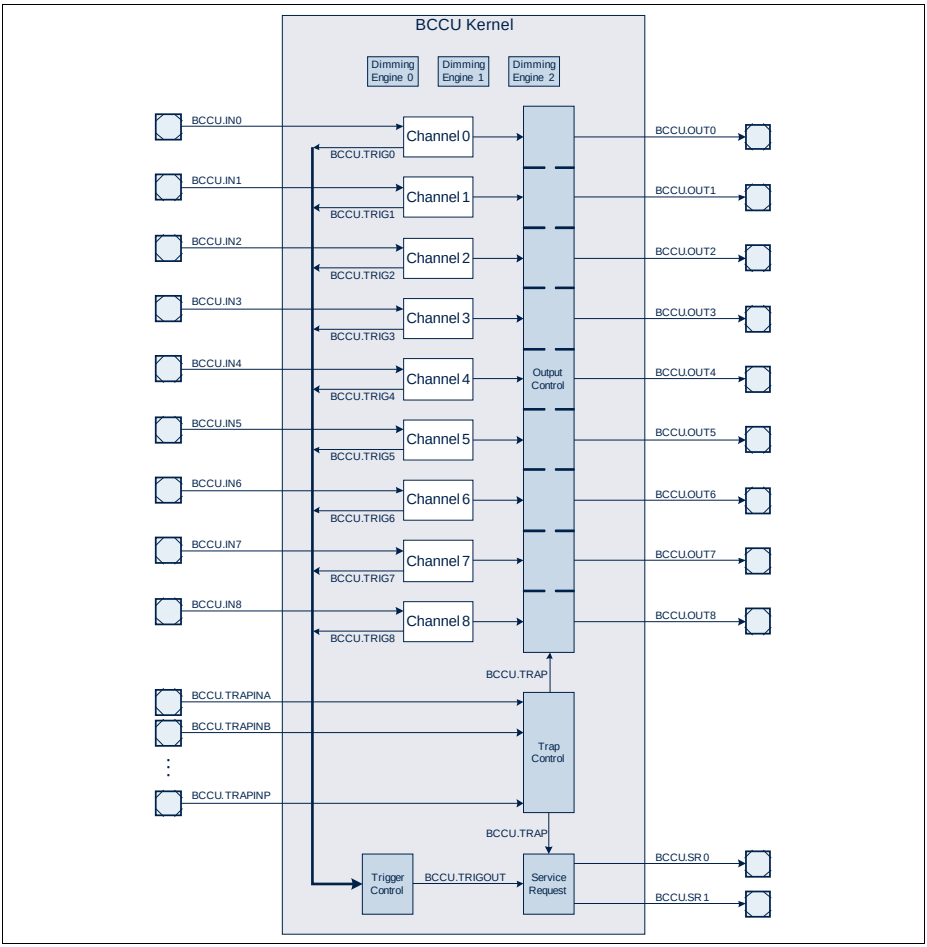



**Figure 22-1 BCCU Kernel Block Diagram**

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.

### 22.2.1 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. E.g. in case of an RGB LED, 3 channels can be used; the individual intensity levels determine the color and a common dimming level (e.g. 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.

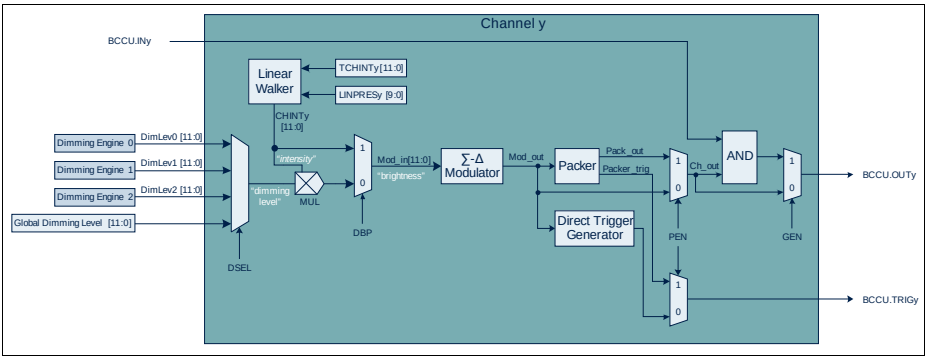



**Figure 22-2 Channel Block Diagram**

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).

## 22.2.2 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


Subject to Agreement on the Use of Product Information

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.

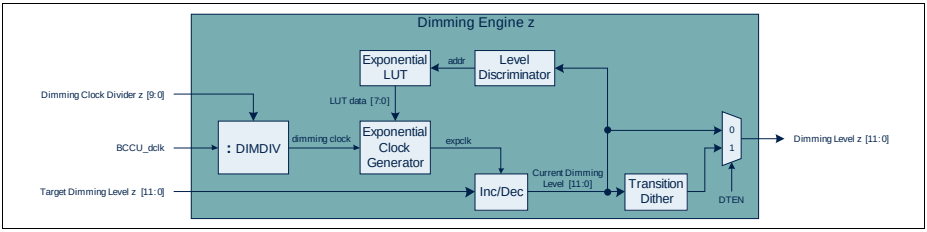



**Figure 22-3 Dimming Engine Block Diagram**

Above the level of 16, the ideal exponential dimming curve is described as:

$$DimmingLevel = 2^{\frac{DimmingClocks + 4096}{2048}} \quad (22.1)$$

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 increasesalong 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 22-1 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 |

**Table 22-1 Coarse Piece-Wise Pseudo-Exponential Curve**

| Upper dimming threshold | Slope of the linear piece | Number of dimming clocks along the line |
|---|---|---|
| 128 | 1/32 | 2048 |
| 256 | 1/16 | 2048 |
| 512 | 1/8 | 2048 |
| 1024 | 1/4 | 2048 |
| 2048 | 1/2 | 2048 |
| 4095 | 1 | 2047 |

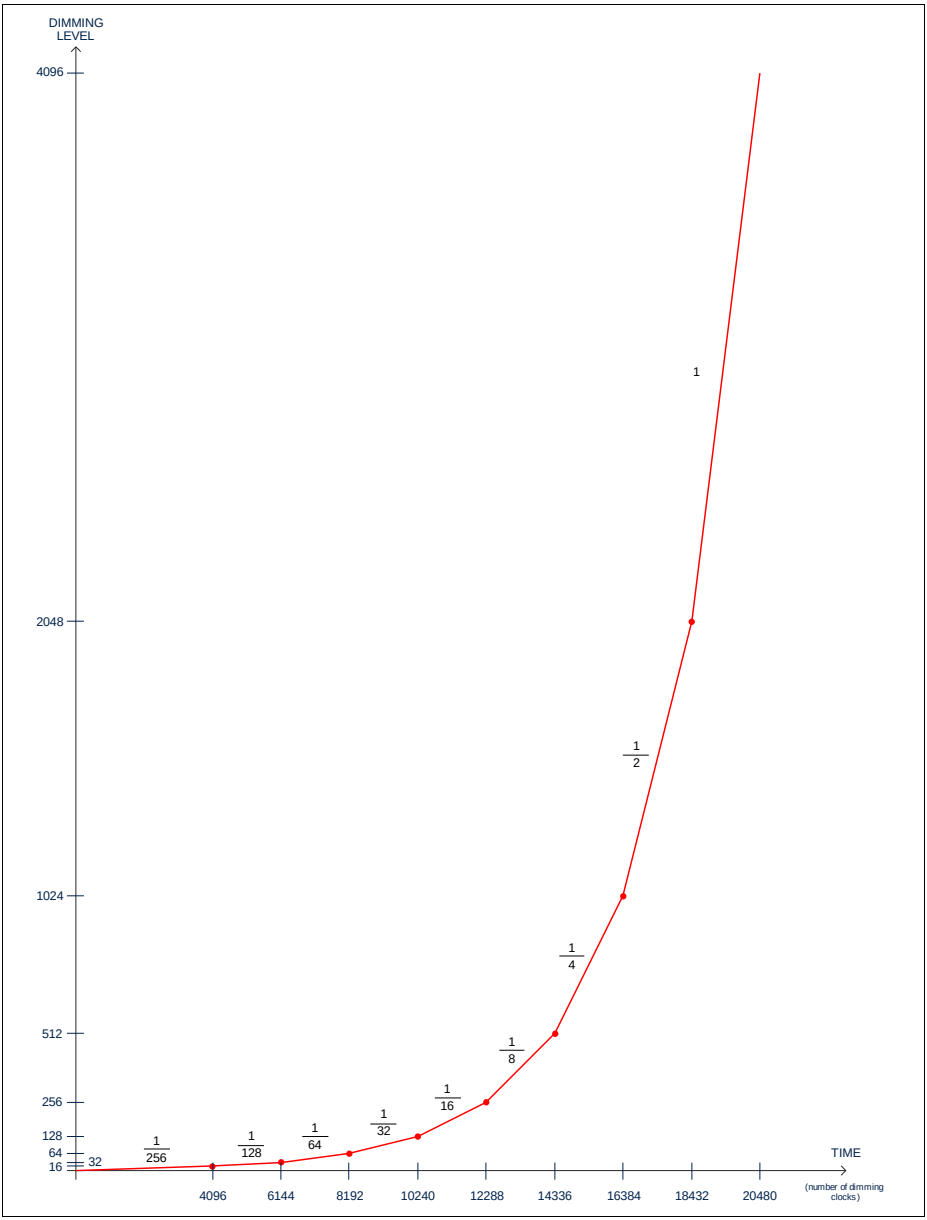



**Figure 22-4 Coarse Piecewise pseudo-exponential curve**

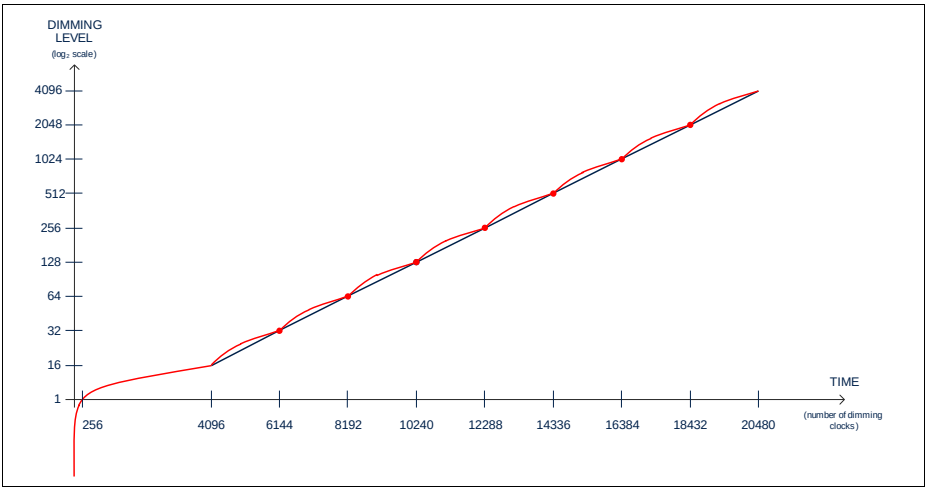



**Figure 22-5 Coarse Piecewise pseudo-exponential curve with the dimming level on a logarithmic scale as the human eye sees it (real and ideal)**

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 signalling the end of the dimming process. Dimming up from 0 to 4095 takes 20479 dimming clocks (not counting syncronization and bus communication delays).

**Table 22-2 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 |

**Table 22-2 Quantized CoarsePiece-Wise Pseudo-Exponential Curve**

| Dimming Levels | Clocks before the next step (“waiting time”) | Slope of the linear piece |
|---|---|---|
| 1025..2048 | 2 | 1/2 |
| 2049.. | 1 | 1 |

For example dimming from 0 to 18 along the coarse curvetakes 4352 clocks (16*256 clocks + 2*128 clocks) when measured as the time between setting and clearing BCCU_**DESTRCON**.DEzS (ignoring delays).

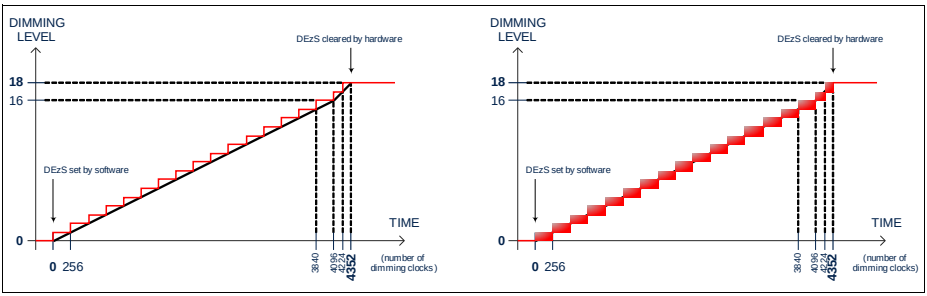



**Figure 22-6 Dimming from 0 to 18 along the coarse curve without and with dither enabled**

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).

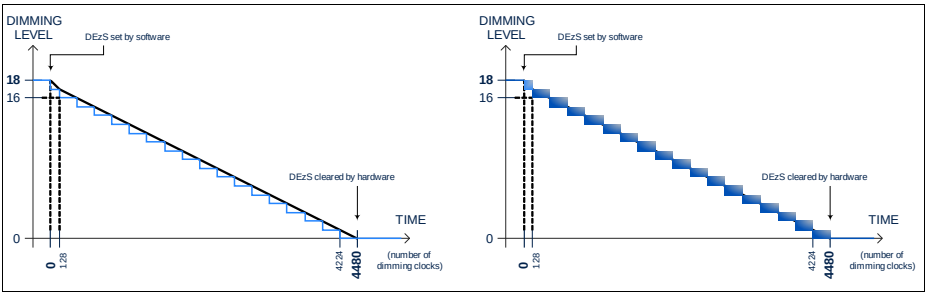



**Figure 22-7 Dimming from 18 to 0 along the coarse curve without and with dither**

**enabled**

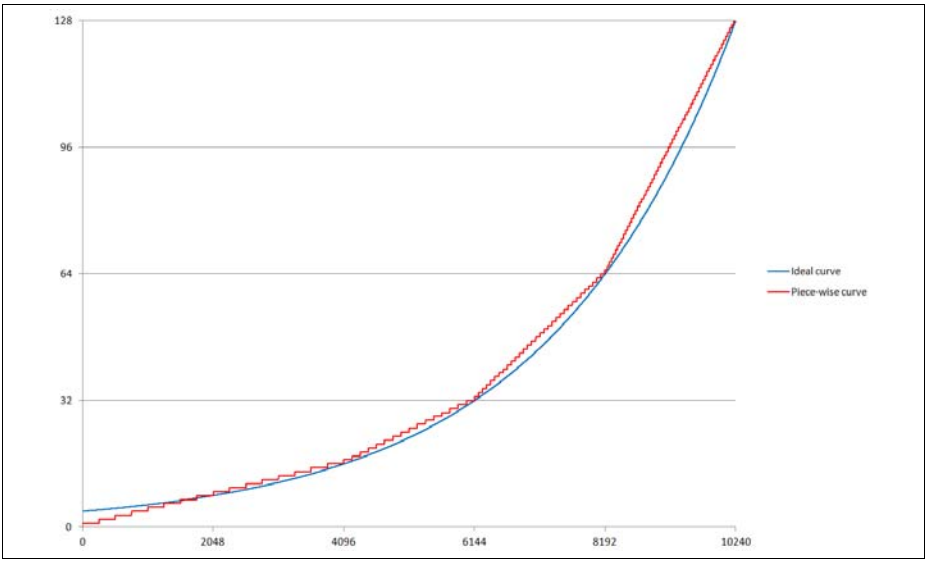



**Figure 22-8 Lower section of the coarse curve (real and ideal)**

The fine curve is similar to the coarse one but it has more pieces and different line slopes.

**Table 22-3 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 |


Subject to Agreement on the Use of Product Information

**Table 22-3 Fine Piece-Wise Pseudo-Exponential Curve**

| Upper dimming threshold | Slope of the linear piece | Number of dimming clocks along the line |
|---|---|---|
| 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 22-4 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 |
| 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 |


Subject to Agreement on the Use of Product Information

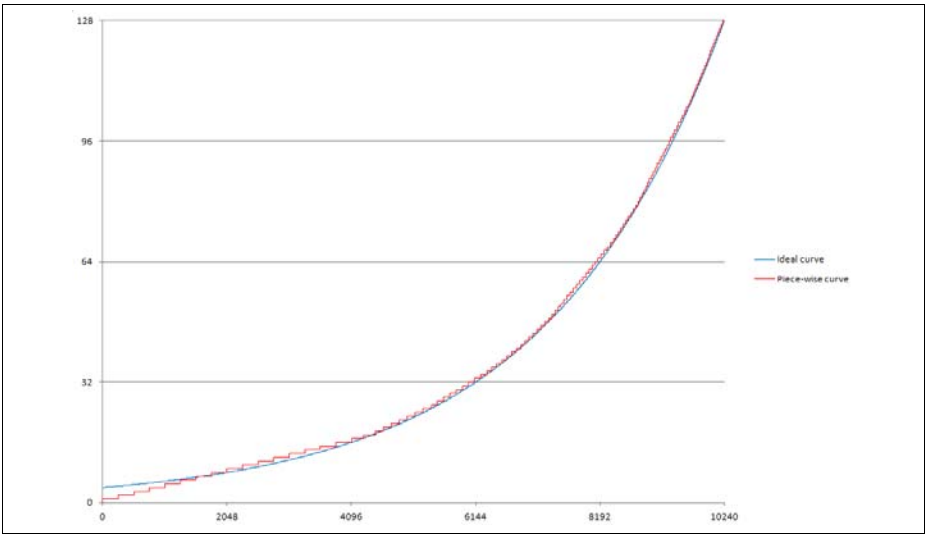



**Figure 22-9 Lower section of the fine curve (real and ideal)**

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):

$$FadeTime_{0-100} = \frac{20479}{\left(\frac{f_{BCCUclk}}{DCLKPS \times DIMDIV}\right)} = \frac{20479}{\left(\frac{f_{BCCUdclk}}{DIMDIV}\right)} \quad (22.2)$$

$$FadeTime_{100-0} = \frac{20734}{\left(\frac{f_{BCCUclk}}{DCLKPS \times DIMDIV}\right)} = \frac{20734}{\left(\frac{f_{BCCUdclk}}{DIMDIV}\right)} \quad (22.3)$$

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.

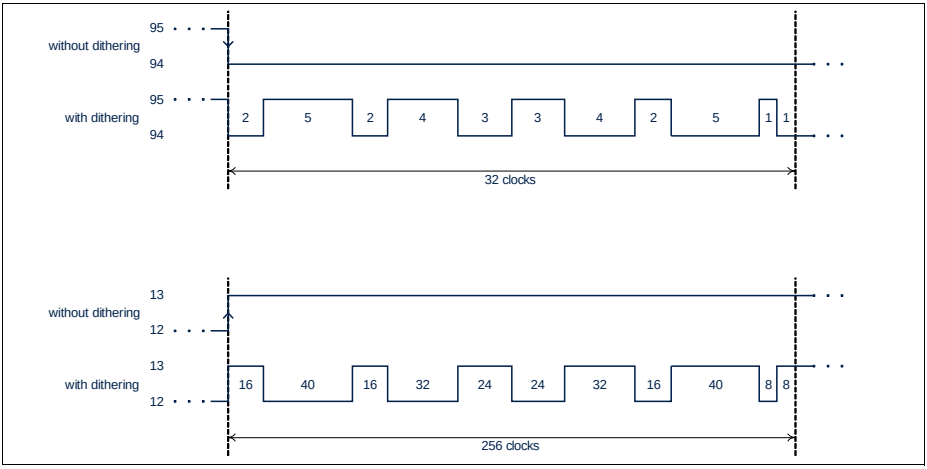



**Figure 22-10 Dithering examples**

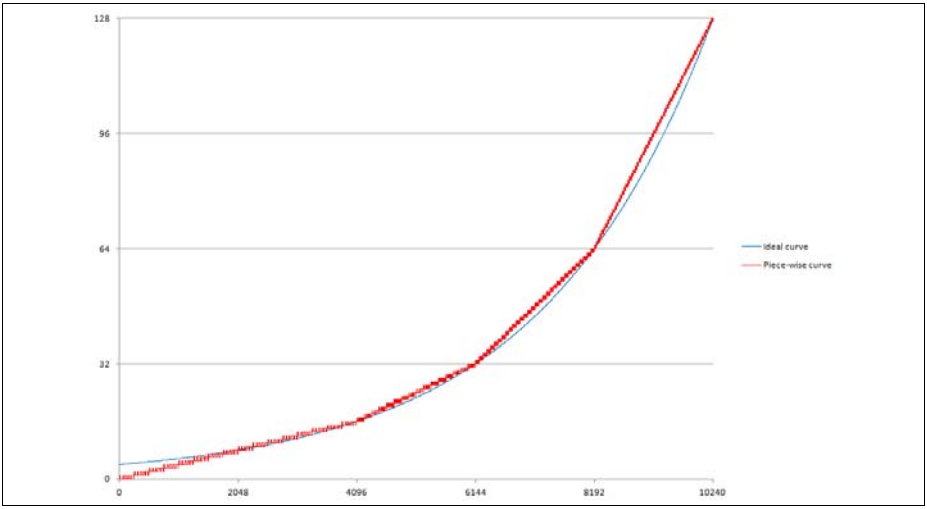



**Figure 22-11 Lower section of the coarse curve with dithering enabled (real and ideal)**

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.

### 22.2.3 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:

$$LINEARWALKTIME = \frac{LINPRES \times 2^{13}}{f_{BCCUfclk}} \quad (22.4)$$

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.

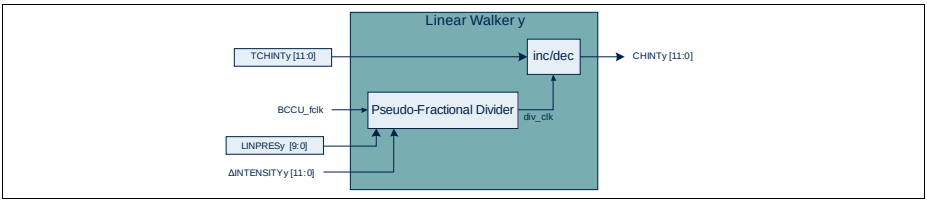



**Figure 22-12 Linear Walker Block Diagram**

### 22.2.4 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:

$$f_{min} = \frac{1}{WDMBN} \times f_{BCCUbclk} \quad (22.5)$$

### 22.2.5 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.

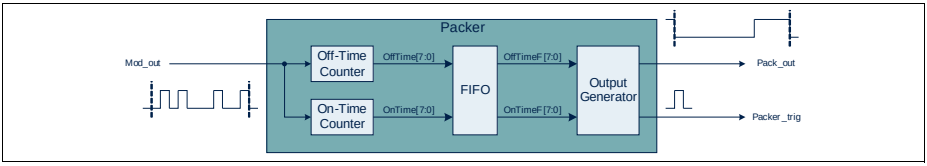



**Figure 22-13 Packer Block Diagram**

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.


Subject to Agreement on the Use of Product Information

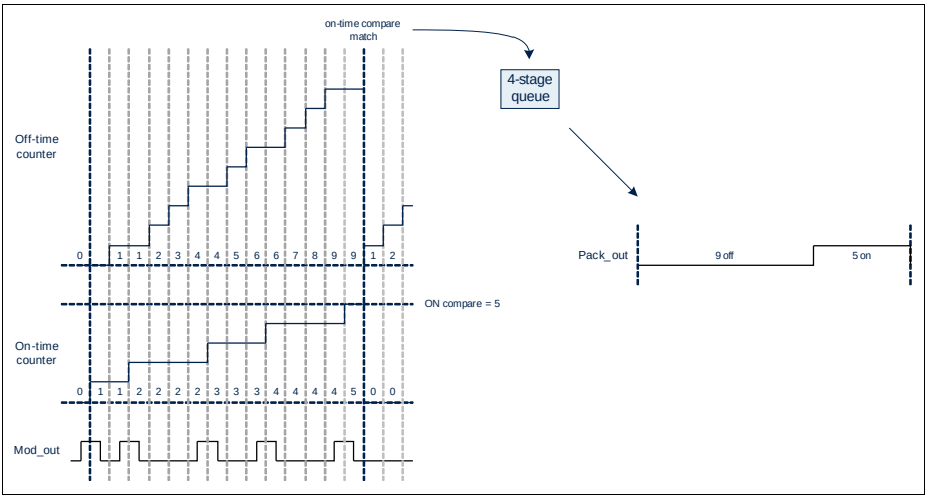



**Figure 22-14 Signal Packing (example)**

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 formulae:

$$PacketTime_{min} = ONCMP \times \frac{1}{f_{BCCUbclk}}$$

$$PacketTime_{max} = (ONCMP + OFFCMP) \times \frac{1}{f_{BCCUbclk}} \quad (22.6)$$

## 22.2.6 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 (e.g. 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).


Subject to Agreement on the Use of Product Information

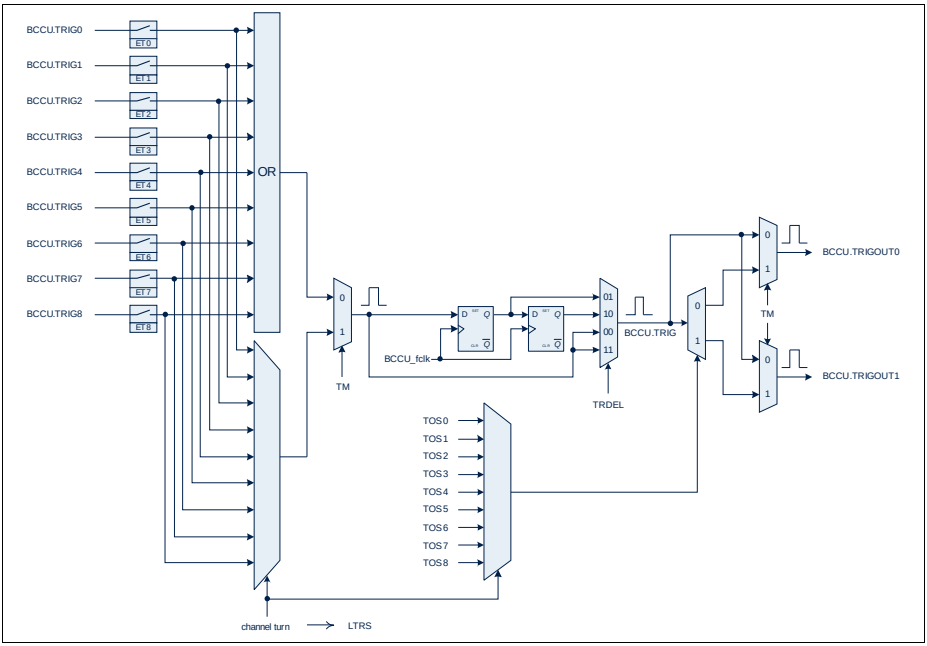



**Figure 22-15 BCCU Trigger Generation**

## 22.2.7 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[1)] 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

1) After a synchronization delay of 4 BCCU_clk cycles

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.

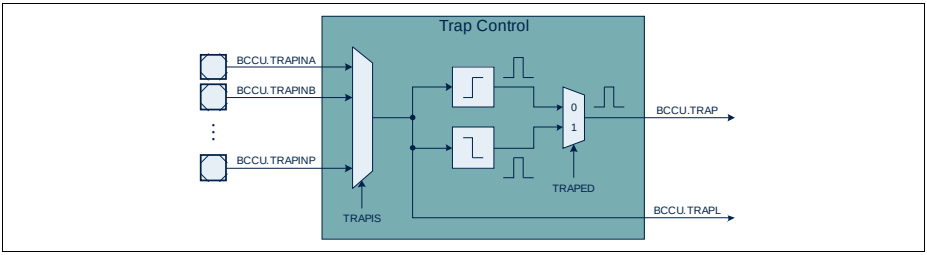



**Figure 22-16 Trap Control**

## 22.3 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 XMC1300, 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.


Subject to Agreement on the Use of Product Information

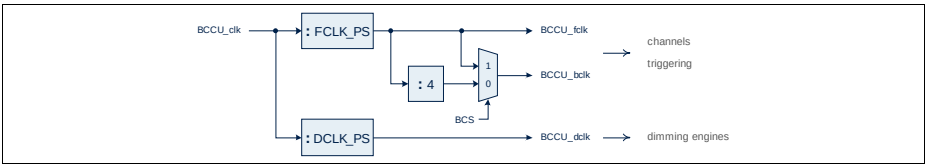



**Figure 22-17 BCCU clocks**

The clock frequencies in normal mode can be calculated with the following formulae:

$$f_{BCCUfclk} = \frac{1}{FCLKPS} \times f_{BCCUclk} \quad (22.7)$$

$$f_{BCCUbclk} = \frac{1}{FCLKPS \times 4} \times f_{BCCUclk} \quad (22.8)$$

$$f_{BCCUdclk} = \frac{1}{DCLKPS} \times f_{BCCUclk} \quad (22.9)$$

The kernel is only in operation in active mode.

## 22.4 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.


Subject to Agreement on the Use of Product Information

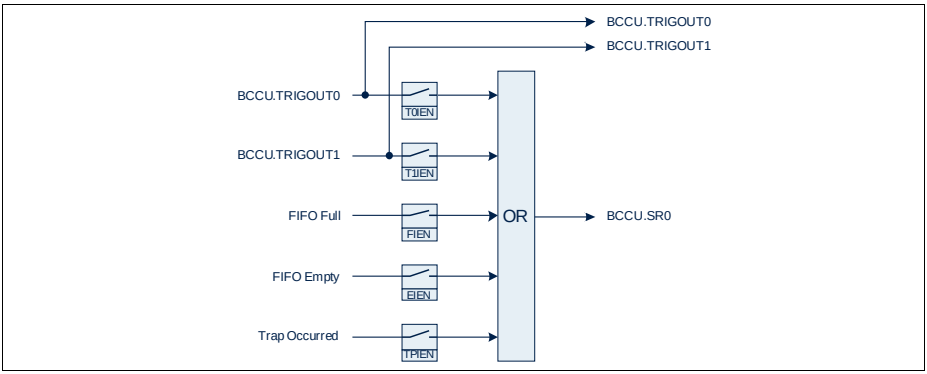



**Figure 22-18 Service Request Nodes**

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.

## 22.5 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 XMC1300, 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.*

## 22.6 Initialization

The BCCU registers are recommended to be programmed in a specific sequence.

### 1. Program Global Registers

1. BCCU_**CHOCON** to set output passive and active levels
2. BCCU_**GLOBCLK** to set up the clocks
3. BCCU_**GLOBCON** for general configuration
4. BCCU_**EVIER** to set up interrupts
5. BCCU_**GLOBDIM** for a user-defined dimming level when no dimming engine is used
6. BCCU_**CHTRIG** to set up ADC triggering

### 2. Program Channel Registers

1. BCCU_**CHCONFIGy (y=0-8)** for general channel configuration
2. BCCU_**PKCMPy (y=0-8)** if the respective packer is used
3. 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

1. BCCU_**DTTz (z=0-2)** to adjust the dimming curve

### 4. Enable channels and dimming engines

1. BCCU_**CHEN** to enable channels
2. BCCU_**DEEN** to enable dimming engines

### 5. Set target intensities and start linear walk

1. BCCU_**CHCONFIGy (y=0-8)**.LINPRES to set the linear walk time
2. BCCU_**INTSy (y=0-8)** to set the target intensities
3. BCCU_**CHSTRCON** to start linear walks
4. 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)

1. BCCU_**DTTz (z=0-2)**.DIMDIV to set the fade rate
2. BCCU_**DLSz (z=0-2)** to set the target target dimming levels
3. BCCU_**DESTRCON** to start dimming
4. When the dimming processes are completed, BCCU_**DESTRCON**.DEzS are cleared by hardware


Subject to Agreement on the Use of Product Information

## 22.7 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.

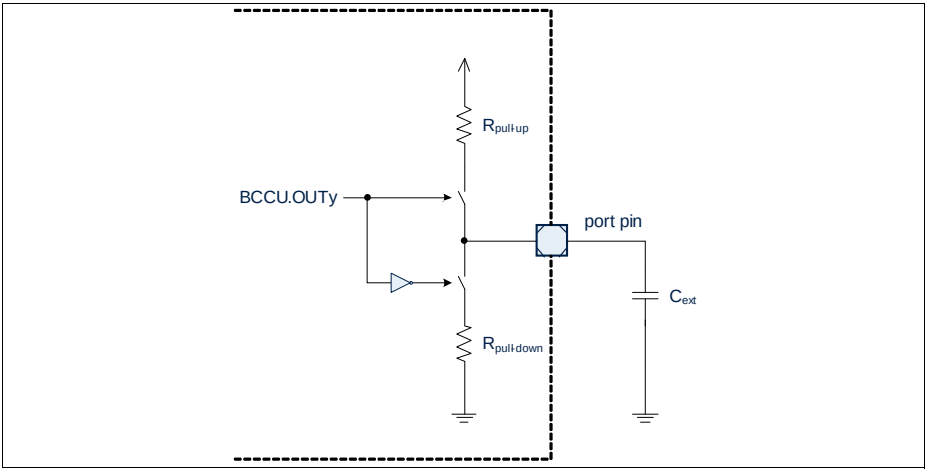



**Figure 22-19 Simple DAC on one pin**

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).


Subject to Agreement on the Use of Product Information

## 22.8 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 22-5 Registers Address Space**

| Module | Base Address | End Address | Note |
|---|---|---|---|
| BCCU | 5003 0000$_H$ | 5003 FFFF$_H$ | |

**Table 22-6 Register Overview**

| Short Name | Description | Offset Addr. | Access Mode | | Description See |
|---|---|---|---|---|---|
| | | | Read | Write | |
| BCCU Global Registers | | | | | |
| GLOBCON | Global Control Register | 0000$_H$ | U, PV, 32 | U, PV, 32 | Page 22-25 |
| GLOBCLK | Global Clock Register | 0004$_H$ | U, PV, 32 | U, PV, 32 | Page 22-27 |
| ID | Module Identification Register | 0008$_H$ | U, PV, 32 | BE | Page 22-29 |
| CHEN | Channel Enable Register | 000C$_H$ | U, PV, 32 | U, PV, 32 | Page 22-29 |
| CHOCON | Channel Output Control Register | 0010$_H$ | U, PV, 32 | U, PV, 32 | Page 22-30 |
| CHTRIG | Channel Trigger Register | 0014$_H$ | U, PV, 32 | U, PV, 32 | Page 22-31 |
| CHSTRCON | Channel Shadow Transfer Control Register | 0018$_H$ | U, PV, 32 | U, PV, 32 | Page 22-32 |
| LTCHOL | Last Trigger Channel Output Level Register | 001C$_H$ | U, PV, 32 | BE | Page 22-33 |
| DEEN | Dimming Engine Enable Register | 0020$_H$ | U, PV, 32 | U, PV, 32 | Page 22-33 |
| DESTRCON | Dimming Engine Shadow Transfer Control Register | 0024$_H$ | U, PV, 32 | U, PV, 32 | Page 22-34 |


Subject to Agreement on the Use of Product Information

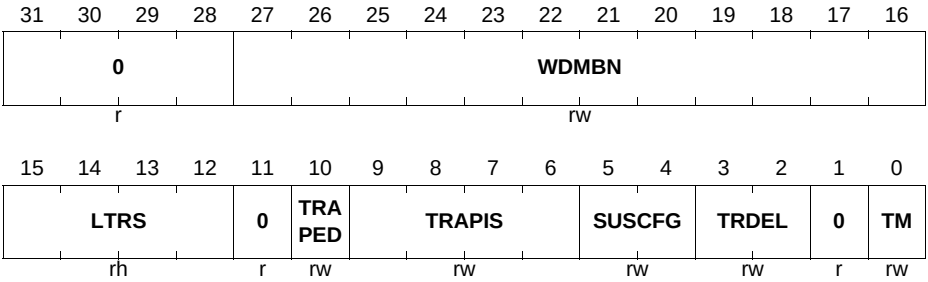
**Table 22-6 Register Overview**

| Short Name | Description | Offset Addr. | Access Mode | | Description See |
|---|---|---|---|---|---|
| | | | Read | Write | |
| GLOBDIM | Global Dimming Level Register | $0028_H$ | U, PV, 32 | U, PV, 32 | Page 22-35 |
| EVIER | Event Interrupt Enable Register | $002C_H$ | U, PV, 32 | U, PV, 32 | Page 22-36 |
| EVFR | Event Flag Register | $0030_H$ | U, PV, 32 | BE | Page 22-37 |
| EVFSR | Event Flag Set Register | $0034_H$ | U, PV, 32 | U, PV, 32 | Page 22-38 |
| EVFCR | Event Flag Clear Register | $0038_H$ | U, PV, 32 | U, PV, 32 | Page 22-39 |
| BCCU Channel Registers | | | | | |
| INTSy | Channel Intensity Shadow Register y | $003C_H$+ y*$0014_H$ | U, PV, 32 | U, PV, 32 | Page 22-41 |
| INTy | Channel Intensity Register y | $0040_H$+ y*$0014_H$ | U, PV, 32 | BE | Page 22-41 |
| CHCONFIGy | Channel Configuration Register y | $0044_H$+ y*$0014_H$ | U, PV, 32 | U, PV, 32 | Page 22-42 |
| PKCMPy | Packer Compare Register y | $0048_H$+ y*$0014_H$ | U, PV, 32 | U, PV, 32 | Page 22-44 |
| PKCNTRy | Packer Counter Register y | $004C_H$+ y*$0014_H$ | U, PV, 32 | U, PV, 32 | Page 22-45 |
| 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 | Page 22-46 |
| DLz | Dimming Level Register z | $0180_H$+ z*$000C_H$ | U, PV, 32 | BE | Page 22-46 |
| DTTz | Dimming Transition Time Register z | $0184_H$+ z*$000C_H$ | U, PV, 32 | U, PV, 32 | Page 22-47 |
| Reserved | Reserved | $01A0_H$ - $FFFC_H$ | BE | BE | |


Subject to Agreement on the Use of Product Information

### 22.8.1 Global Registers

**GLOBCON**
**Global Control** **($0000_H$)** **Reset Value: $0320\ 0000_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **TM** | 0 | rw | **Trigger Mode**<br>$0_B$ Mode 0: BCCU trigger occurs if there is any channel trigger (OR logic)<br>$1_B$ Mode 1: BCCU trigger occurs if there is a channel trigger event on the active channel. When this happens, the next trigger-enabled channel will be active following the round robin. |
| **0** | 1 | r | **Reserved**<br>Read as 0; should be written with 0. |
| **TRDEL** | [3:2] | rw | **Trigger Delay**<br>$00_B$ No delay<br>$01_B$ BCCU trigger occurs a quarter bit time after the channel trigger that caused it; only to be used if BCCU_GLOBCLK.BCS is 0<br>$10_B$ BCCU trigger occurs half a bit time after the channel trigger that caused it; only to be used if BCCU_GLOBCLK.BCS is 0<br>$11_B$ No delay |

| Field | Bits | Type | Description |
|---|---|---|---|
| **SUSCFG** | [5:4] | rw | **Suspend Mode Configuration**<br>This bitfield determines how the BCCU channels enter suspend mode.<br>$00_B$ Suspend request is ignored and the module cannot get suspended<br>$01_B$ All channels stop running immediately and freeze in the last state without any safe stop<br>$10_B$ All channels stop running immediately and freeze in the last state; all outputs go to passive state to achieve safe stop<br>$11_B$ Reserved |
| **TRAPIS** | [9:6] | rw | **Trap Input Pin Selector**<br>Source of the trap input (BCCU.TRAPL)<br>$0000_B$ BCCU.TRAPINA<br>$0001_B$ BCCU.TRAPINB<br>$0010_B$ BCCU.TRAPINC<br>$0011_B$ BCCU.TRAPIND<br>$0100_B$ BCCU.TRAPINE<br>$0101_B$ BCCU.TRAPINF<br>$0110_B$ BCCU.TRAPING<br>$0111_B$ BCCU.TRAPINH<br>$1000_B$ BCCU.TRAPINI<br>$1001_B$ BCCU.TRAPING<br>$1010_B$ BCCU.TRAPINK<br>$1011_B$ BCCU.TRAPINL<br>$1100_B$ BCCU.TRAPINM<br>$1101_B$ BCCU.TRAPINN<br>$1110_B$ BCCU.TRAPINO<br>$1111_B$ BCCU.TRAPINP |
| **TRAPED** | 10 | rw | **Trap Edge**<br>$0_B$ Trap occurs (trap flag is set) on rising edge of the BCCU.TRAPL signal<br>$1_B$ Trap occurs (trap flag is set) on falling edge of the BCCU.TRAPL signal |
| **0** | 11 | r | **Reserved**<br>Read as 0; should be written with 0. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **LTRS** | [15:12] | rh | **Last Trigger Source**<br>Shows which channel generated the last trigger (the value is always 0 in Trigger Mode 0)<br>$0000_B$ The last trigger occurred in channel turn 0<br>$0001_B$ The last trigger occurred in channel turn 1<br>$0010_B$ The last trigger occurred in channel turn 2<br>$0011_B$ The last trigger occurred in channel turn 3<br>$0100_B$ The last trigger occurred in channel turn 4<br>$0101_B$ The last trigger occurred in channel turn 5<br>$0110_B$ The last trigger occurred in channel turn 6<br>$0111_B$ The last trigger occurred in channel turn 7<br>$1000_B$ The last trigger occurred in channel turn 8 |
| **WDMBN** | [27:16] | rw | **Watchdog Maximum Bitnumber**<br>The maximum number of consecutive zeroes allowed at the output of the sigma-delta modulators |
| **0** | [31:28] | r | **Reserved**<br>Read as 0; should be written with 0. |

**GLOBCLK**
**Global Clock** **($0004_H$)** **Reset Value: $00DB\ 0190_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | | | | **DCLK_PS** | | | | | | | | | | | |
| r | | | | rw | | | | | | | | | | | |

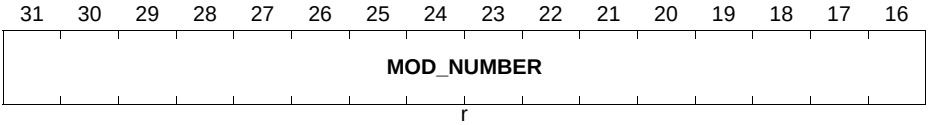
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **BCS** | **0** | | | **FCLK_PS** | | | | | | | | | | | |
| rw | r | | | rw | | | | | | | | | | | |

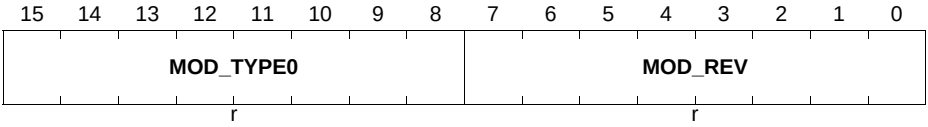

Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **FCLK_PS** | [11:0] | rw | **Fast Clock Prescaler Factor**<br>The constant clock input to the BCCU is prescaled according to this value to generate BCCU_fclk. BCCU_bclk is generated from BCCU_fclk by a division of 4. BCCU_bclk determines the bit-time at the output of the sigma-delta modulators.<br>$0_D$ No clock<br>$1_D$ Divide by 1<br>...<br>$4095_D$ Divide by 4095 |
| **BCS** | 15 | rw | **Bit-Clock Selector**<br>$0_B$ Normal Mode: BCCU_bclk is generated from BCCU_fclk by a division of 4<br>$1_B$ Fast Mode: BCCU_bclk is the same as BCCU_fclk |
| **0** | [14:12] | r | **Reserved**<br>Read as 0; should be written with 0. |
| **DCLK_PS** | [27:16] | rw | **Dimmer Clock Prescaler Factor**<br>The constant clock input to the BCCU is prescaled according to this value to generate BCCU_dclk which is the clock input to the dimming engines.<br>$0_D$ No clock<br>$1_D$ Divide by 1<br>...<br>$4095_D$ Divide by 4095 |
| **0** | [31:28] | r | **Reserved**<br>Read as 0; should be written with 0. |


Subject to Agreement on the Use of Product Information

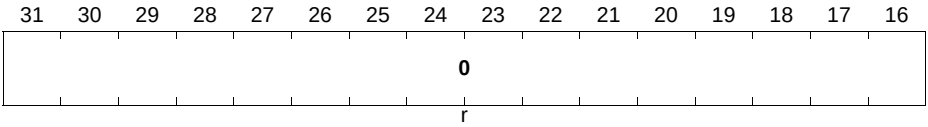
**ID**
**Module Identification** **($0008_H$)** **Reset Value: 00F3 C0XX$_H$**

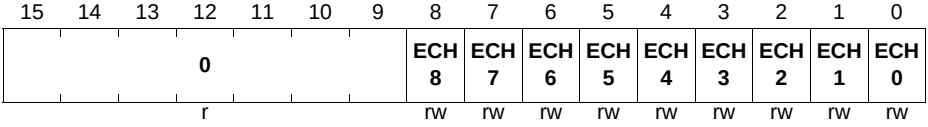



| Field | Bits | Type | Description |
|---|---|---|---|
| **MOD_REV** | [7:0] | r | **Module Revision Number**<br>MOD_REV defines the revision number. The value of a module revision starts with $01_H$ (first revision). |
| **MOD_TYPE0** | [15:8] | r | **Module Type**<br>This bit field is $C0_H$. It defines the module as a 32-bit module. |
| **MOD_NUMBER** | [31:16] | r | **Module Number Value**<br>This bit field defines the module identification number. |

**CHEN**
**Channel Enable** **($000C_H$)** **Reset Value: 0000 0000$_H$**



| Field | Bits | Type | Description |
|---|---|---|---|
| **ECHy (y=0-8)** | y | rw | **Channel y Enable**<br>$0_B$ Channel is disabled, the output level is passive; the Linear Walker and the Sigma-Delta Modulator are reset, the Packer FIFO is flushed; all internal logic and INTy are reset when the channel gets disabled<br>$1_B$ Channel is enabled |
| **0** | [31:9] | r | **Reserved**<br>Read as 0; should be written with 0. |

**CHOCON**
**Channel Output Control** **($0010_H$)** **Reset Value: $0000\ 0000_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | **0** | | | | **CH8 TPE** | **CH7 TPE** | **CH6 TPE** | **CH5 TPE** | **CH4 TPE** | **CH3 TPE** | **CH2 TPE** | **CH1 TPE** | **CH0 TPE** |
| | | | r | | | | rw | rw | rw | rw | rw | rw | rw | rw | rw |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | **0** | | | | **CH8 OP** | **CH7 OP** | **CH6 OP** | **CH5 OP** | **CH4 OP** | **CH3 OP** | **CH2 OP** | **CH1 OP** | **CH0 OP** |
| | | | r | | | | rw | rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **CHyOP (y=0-8)** | y | rw | **Channel y Output Passive Level**<br>$0_B$ Active high<br>$1_B$ Active low |
| **0** | [15:9] | r | **Reserved**<br>Read as 0; should be written with 0. |
| **CHyTPE (y=0-8)** | y+16 | rw | **Channel y Trap Enable**<br>$0_B$ Trap function on channel is disabled<br>$1_B$ Trap function on channel is enabled, the output goes to passive level when trap occurs |
| **0** | [31:25] | r | **Reserved**<br>Read as 0; should be written with 0. |


Subject to Agreement on the Use of Product Information

**CHTRIG**
**Channel Trigger** **($0014_H$)** **Reset Value: $0000\ 0000_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | **0** | | | | **TOS 8** | **TOS 7** | **TOS 6** | **TOS 5** | **TOS 4** | **TOS 3** | **TOS 2** | **TOS 1** | **TOS 0** |
| | | | r | | | | rw | rw | rw | rw | rw | rw | rw | rw | rw |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | **0** | | | | **ET8** | **ET7** | **ET6** | **ET5** | **ET4** | **ET3** | **ET2** | **ET1** | **ET0** |
| | | | r | | | | rw | rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **ETy (y=0-8)** | y | rw | **Channel y Trigger Enable**<br>$0_B$ Channel trigger is disabled<br>$1_B$ Channel trigger is enabled<br>If all Channel y Trigger Enable bits are 0, then the channel pointer is reset and points to channel 0. |
| **0** | [15:9] | r | **Reserved**<br>Read as 0; should be written with 0. |
| **TOSy (y=0-8)** | y+16 | rw | **Channel y Trigger Output Select**<br>Only used in Trigger Mode 1 (GLOBON.TM=1), otherwise ignored<br>$0_B$ The channel trigger pulse will appear on BCCU_TRIGOUT0<br>$1_B$ The channel trigger pulse will appear on BCCU_TRIGOUT1 |
| **0** | [31:25] | r | **Reserved**<br>Read as 0; should be written with 0. |

**CHSTRCON**
**Channel Shadow Transfer** **($0018_H$)** **Reset Value: 0000 $0000_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | CH8A | CH7A | CH6A | CH5A | CH4A | CH3A | CH2A | CH1A | CH0A |
| r | | | | | | | w | w | w | w | w | w | w | w | w |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | CH8S | CH7S | CH6S | CH5S | CH4S | CH3S | CH2S | CH1S | CH0S |
| r | | | | | | | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **CHyS (y=0-8)** | y | rwh | **Channel y Shadow Transfer**<br>$0_B$ No action<br>$1_B$ Initiate channel y target intensity shadow transfer. The linear walk will start and channel y intensity will start to change towards the target. Cleared by hardware when the linear walk is complete and the target has been reached. |
| **0** | [15:9] | r | **Reserved**<br>Read as 0; should be written with 0. |
| **CHyA (y=0-8)** | y+16 | w | **Channel y Linear Walk Abort**<br>$0_B$ No action<br>$1_B$ Abort linear walk; CHyS is cleared, channel y intensity stops changing<br>Always read as 0 |
| **0** | [31:25] | r | **Reserved**<br>Read as 0; should be written with 0. |

**LTCHOL**

**Last Trigger Channel Output Level (001C$_H$)** **Reset Value: 0000 0000$_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | LTOL8 | LTOL7 | LTOL6 | LTOL5 | LTOL4 | LTOL3 | LTOL2 | LTOL1 | LTOL0 |
| r | | | | | | | rh | rh | rh | rh | rh | rh | rh | rh | rh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **LTOLy (y=0-8)** | y | rh | **Last Trigger Channel Output**<br>The output level of the channel when the last trigger occurred. The bit gets updated with every BCCU trigger in Trigger Mode 0. In Trigger Mode 1, the bit gets updated when a trigger occurs in the relevant channel turn.<br>$0_B$ Passive<br>$1_B$ Active |
| **0** | [31:9] | r | **Reserved**<br>Read as 0; should be written with 0. |

**DEEN**

**Dimming Engine Enable (0020$_H$)** **Reset Value: 0000 0000$_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | | | | | | EDE2 | EDE1 | EDE0 |
| r | | | | | | | | | | | | | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **EDEz (z=0-2)** | z | rw | **Dimming Engine z Enable**<br>$0_B$ Dimming Engine is disabled;Channel brightness gets written with the value of channel intensity<br>$1_B$ Dimming Engine is enabled |
| **0** | [31:3] | r | **Reserved**<br>Read as 0; should be written with 0. |

**DESTRCON**
**Dimming Shadow Transfer** **($0024_H$)** **Reset Value: $0000\ 0000_H$**

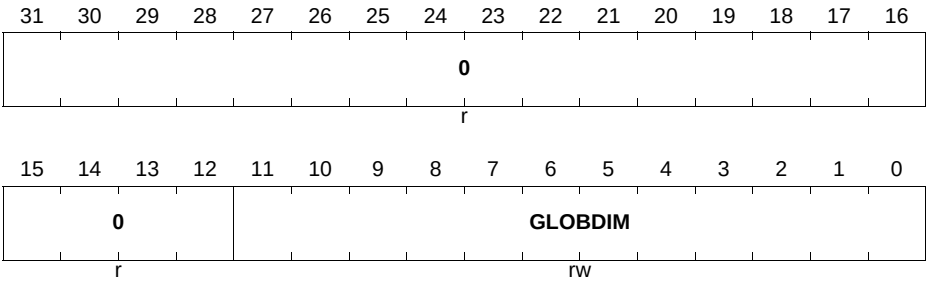
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | **0** | | | | | | | **DE2A** | **DE1A** | **DE0A** |
| | | | | | | r | | | | | | | w | w | w |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | **0** | | | | | | | **DE2S** | **DE1S** | **DE0S** |
| | | | | | | r | | | | | | | rwh | rwh | rwh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **DEzS (z=0-2)** | z | rwh | **Dimming Engine z Shadow Transfer**<br>$0_B$ No action<br>$1_B$ Initiate target dimming level shadow transfer. The dimming process will start and the dimming level will change towards the target. Cleared by hardware when the dimming process is complete and the target has been reached after the configured dimming period. |
| **0** | [15:3] | r | **Reserved**<br>Read as 0; should be written with 0. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **DEzA (z=0-2)** | z+16 | w | **Dimming Engine z Dimming Abort**<br>$0_B$ No action<br>$1_B$ Abort dimming; DEzS is cleared, BCCU_DLz.DLEV stops changing<br>Always read as 0 |
| **0** | [31:19] | r | **Reserved**<br>Read as 0; should be written with 0. |

**GLOBDIM**
**Global Dimming Level** **($0028_H$)** **Reset Value: $0000\ 0000_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **GLOBDIM** | [11:0] | rw | **Global Dimming Level**<br>Can be used for software-controlled dimming |
| **0** | [31:12] | r | **Reserved**<br>Read as 0; should be written with 0. |

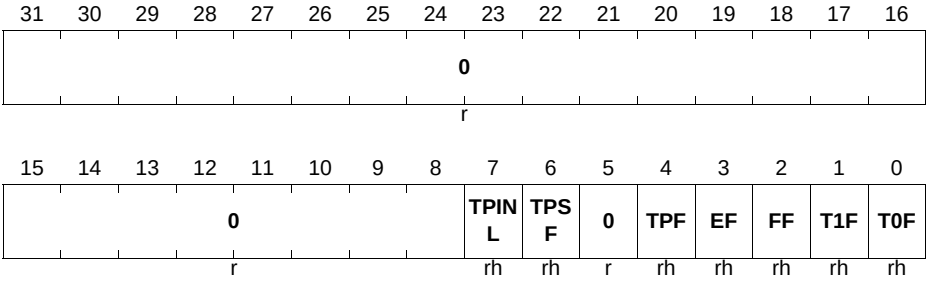
**EVIER**
**Event Interrupt Enable** **(002C$_H$)** **Reset Value: 0000 0000$_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | | | | TPIEN | EIEN | FIEN | T1IEN | T0IEN |
| r | | | | | | | | | | | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **T0IEN** | 0 | rw | **Trigger 0 Interrupt Enable**<br>$0_B$ Trigger 0 interrupt generation is disabled<br>$1_B$ BCCU trigger 0 (BCCU_TRIGOUT0) generates an interrupt on SR0 |
| **T1IEN** | 1 | rw | **Trigger 1 Interrupt Enable**<br>$0_B$ Trigger 1 interrupt generation is disabled<br>$1_B$ BCCU trigger 1 (BCCU_TRIGOUT1) generates an interrupt on SR0 |
| **FIEN** | 2 | rw | **FIFO Full Interrupt Enable**<br>$0_B$ FIFO-full interrupt generation is disabled<br>$1_B$ An interrupt is generated on SR0 if any of the packer FIFOs is full when there is a write attempt by the on-time or off-time counter |
| **EIEN** | 3 | rw | **FIFO Empty Interrupt Enable**<br>$0_B$ FIFO-full interrupt generation is disabled<br>$1_B$ An interrupt is generated on SR0 if any of the packer FIFOs is empty when there is a read attempt by the output generator |
| **TPIEN** | 4 | rw | **Trap Interrupt Enable**<br>$0_B$ Trap interrupt generation is disabled<br>$1_B$ An interrupt is generated on SR0 if a trap occurs |
| **0** | [31:5] | r | **Reserved**<br>Read as 0; should be written with 0. |

**EVFR**
**Event Flag** **($0030_H$)** **Reset Value: $0000\ 0000_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **T0F** | 0 | rh | **Trigger 0 Flag**<br>$0_B$ No trigger event has been detected on BCCU trigger line 0 (BCCU_TRIGOUT0)<br>$1_B$ A trigger event has been detected on BCCU trigger line 0 (BCCU_TRIGOUT0) |
| **T1F** | 1 | rh | **Trigger 1 Flag**<br>$0_B$ No trigger event has been detected on BCCU trigger line 1 (BCCU_TRIGOUT1)<br>$1_B$ A trigger event has been detected on BCCU trigger line 1 (BCCU_TRIGOUT1) |
| **FF** | 2 | rh | **FIFO Full Flag**<br>$0_B$ No FIFO full event has been detected<br>$1_B$ A FIFO full event has been detected because one of the packer FIFOs is full and there has been a write attempt by the on-time or off-time counter |
| **EF** | 3 | rh | **FIFO Empty Flag**<br>$0_B$ No FIFO full event has been detected<br>$1_B$ A FIFO full event has been detected because one of the packer FIFOs is empty and there has been a read attempt by the output generator |
| **TPF** | 4 | rh | **Trap Flag**<br>$0_B$ No trap event has been detected<br>$1_B$ A trap event has been detected |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | 5 | r | **Reserved**<br>Read as 0; should be written with 0. |
| **TPSF** | 6 | rh | **Trap State Flag**<br>$0_B$ BCCU is not in a trap state<br>$1_B$ BCCU is in a trap state, the affected channel outputs are at their passive levels |
| **TPINL** | 7 | rh | **Trap Input Level**<br>$0_B$ The current level of BCCU.TRAPL is low<br>$1_B$ The current level of BCCU.TRAPL is high |
| **0** | [31:8] | r | **Reserved**<br>Read as 0; should be written with 0. |

**EVFSR**
**Event Flag Set** **(0034$_H$)** **Reset Value: 0000 0000$_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | | | | | | | | | **TPS** | **0** | **TPFS** | **EFS** | **FFS** | **T1FS** | **T0FS** |
| r | | | | | | | | | w | r | w | w | w | w | w |

| Field | Bits | Type | Description |
|---|---|---|---|
| **T0FS** | 0 | w | **Trigger 0 Flag Set**<br>$0_B$ No action<br>$1_B$ Sets the Trigger 0 Flag in EVFR and an interrupt will be generated if enabled in EVIER |
| **T1FS** | 1 | w | **Trigger 1 Flag Set**<br>$0_B$ No action<br>$1_B$ Sets the Trigger 1 Flag in EVFR and an interrupt will be generated if enabled in EVIER |

| Field | Bits | Type | Description |
|---|---|---|---|
| **FFS** | 2 | w | **FIFO Full Flag Set**<br>$0_B$ No action<br>$1_B$ Sets the FIFO Full Flag in EVFR and an interrupt will be generated if enabled in EVIER |
| **EFS** | 3 | w | **FIFO Empty Flag Set**<br>$0_B$ No action<br>$1_B$ Sets the FIFO Empty Flag in EVFR and an interrupt will be generated if enabled in EVIER |
| **TPFS** | 4 | w | **Trap Flag Set**<br>$0_B$ No action<br>$1_B$ Sets the Trap Flag in EVFR and an interrupt will be generated if enabled in EVIER, no trap will occur |
| **0** | 5 | r | **Reserved**<br>Read as 0; should be written with 0. |
| **TPS** | 6 | w | **Trap Set**<br>$0_B$ No action<br>$1_B$ Sets the Trap State Flag and Trap Flag in EVFR, a trap will be generated and an interrupt will be generated if enabled in EVIER |
| **0** | [31:7] | r | **Reserved**<br>Read as 0; should be written with 0. |

**EVFCR**
**Event Flag Clear** **($0038_H$)** **Reset Value: $0000\ 0000_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | | | | | | | | | **TPC** | **0** | **TPFC** | **EFC** | **FFC** | **T1FC** | **T0FC** |
| r | | | | | | | | | w | r | w | w | w | w | w |

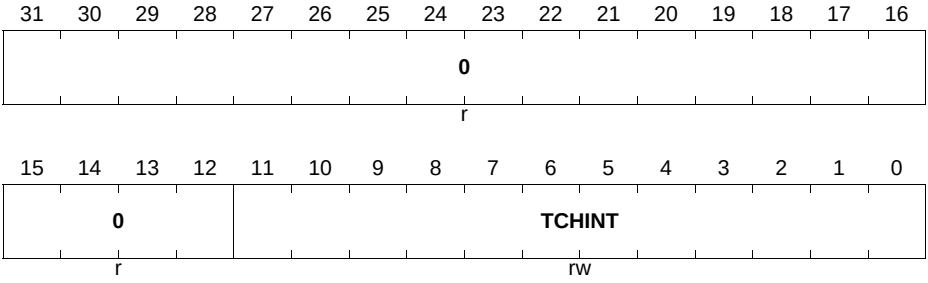
| Field | Bits | Type | Description |
|---|---|---|---|
| **T0FC** | 0 | w | **Trigger 0 Flag Clear**<br>$0_B$ No action<br>$1_B$ Clears the Trigger 0 Flag in EVFR |
| **T1FC** | 1 | w | **Trigger 1 Flag Clear**<br>$0_B$ No action<br>$1_B$ Clears the Trigger 1 Flag in EVFR |
| **FFC** | 2 | w | **FIFO Full Flag Clear**<br>$0_B$ No action<br>$1_B$ Clears the FIFO Full Flag in EVFR |
| **EFC** | 3 | w | **FIFO Empty Flag Clear**<br>$0_B$ No action<br>$1_B$ Clears the FIFO Empty Flag in EVFR |
| **TPFC** | 4 | w | **Trap Flag Clear**<br>$0_B$ No action<br>$1_B$ Clears the Trap Flag in EVFR |
| **0** | 5 | r | **Reserved**<br>Read as 0; should be written with 0. |
| **TPC** | 6 | w | **Trap Clear**<br>$0_B$ No action<br>$1_B$ Clears the Trap State Flag in EVFR; trap state is exited, the affected channels will return to their normal output levels |
| **0** | [31:7] | r | **Reserved**<br>Read as 0; should be written with 0. |

### 22.8.2 Channel Registers

There are 9 channels available on XMC1300. y denotes channel 0 through 8.

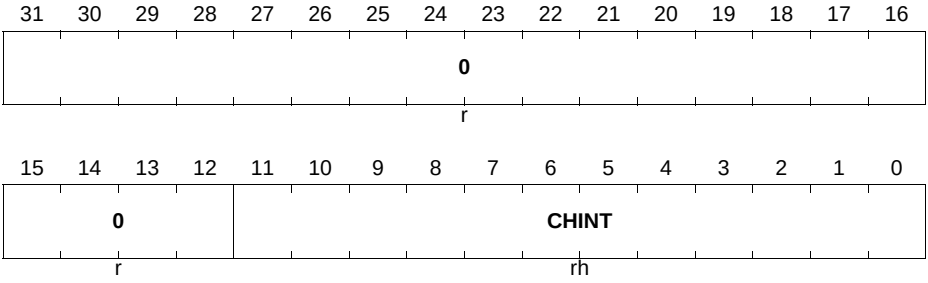

Subject to Agreement on the Use of Product Information

**INTSy (y=0-8)**
**Channel Intensity Shadow** **($003C_H$ + y*$0014_H$)** **Reset Value: 0000 $0000_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **TCHINT** | [11:0] | rw | **Target Channel Intensity**<br>Can only be written if CHSTRCON.CHyS is not set, otherwise the new value will be ignored. |
| **0** | [31:12] | r | **Reserved**<br>Read as 0; should be written with 0. |

**INTy (y=0-8)**
**Channel Intensity** **($0040_H$ + y*$0014_H$)** **Reset Value: 0000 $0000_H$**

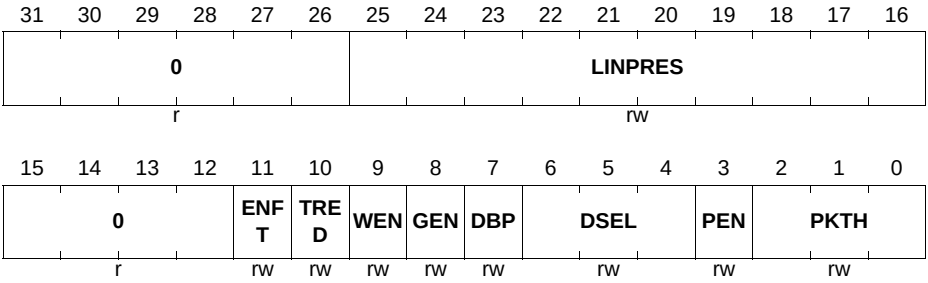



| Field | Bits | Type | Description |
|---|---|---|---|
| **CHINT** | [11:0] | rh | **Channel Intensity**<br>Actual channel intensity. |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | [31:12] | r | **Reserved**<br>Read as 0; should be written with 0. |

**CHCONFIGy (y=0-8)**
**Channel Configuration** **($0044_H$ + y*$0014_H$)** **Reset Value: 0000 $0002_H$**



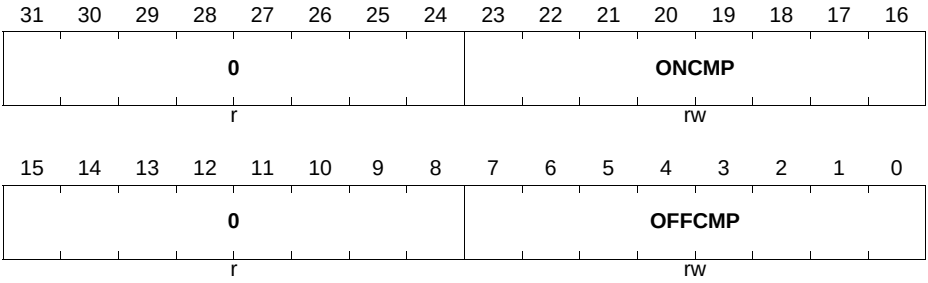

| Field | Bits | Type | Description |
|---|---|---|---|
| **PKTH** | [2:0] | rw | **Packer Threshold**<br>After being enabled, the packer will start generating the output based on the values stored in the FIFO (queue) once the number of full stages reaches this threshold (until that only zeros are generated).<br>$000_B$ Not allowed<br>$001_B$ Threshold value of 1<br>$010_B$ Threshold value of 2<br>$011_B$ Threshold value of 3<br>$100_B$ Threshold value of 4<br>$101_B$ Reserved<br>$110_B$ Reserved<br>$111_B$ Reserved |
| **PEN** | 3 | rw | **Packer Enable**<br>$0_B$ The packer is not used<br>$1_B$ On-time and off-time counters are running and the packed output bitstream with the packer trigger are generated. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **DSEL** | [6:4] | rw | **Dimming Select**<br>Source of the dimming input<br>$000_B$ Dimming Engine 0<br>$001_B$ Dimming Engine 1<br>$010_B$ Dimming Engine 2<br>$011_B$ Reserved<br>$100_B$ Reserved<br>$101_B$ Reserved<br>$110_B$ Reserved<br>$111_B$ Global Dimming Level |
| **DBP** | 7 | rw | **Dimming Input Bypass**<br>$0_B$ Channel brightness is the product of the selected dimming input and the channel intensity<br>$1_B$ No dimming input is used, channel brightness is only determined by the channel intensity level |
| **GEN** | 8 | rw | **Gating Enable**<br>$0_B$ Gating function is disabled, the input signal (BCCU.INy) has no effect<br>$1_B$ Gating function is enabled, the output gating signal is BCCU.INy |
| **WEN** | 9 | rw | **Flicker Watchdog Enable**<br>$0_B$ The flicker watchdog is not used<br>$1_B$ The flicker watchdog is active and limits the number of consecutive zeroes at the sigma-delta modulator output according to GLOBCON.WDMBN |
| **TRED** | 10 | rw | **Trigger Edge**<br>$0_B$ Channel triggers occur on channel output transition from passive to active level<br>$1_B$ Channel triggers occur on channel output transition from active to passive level |
| **ENFT** | 11 | rw | **Forced Trigger Enable**<br>$0_B$ No forced trigger is generated<br>$1_B$ The trigger generator generates a trigger if the output of the sigma-delta modulator hasn't changed state for 256 bit times; only takes effect if the packer is disabled (PEN=0) |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | [15:12] | r | **Reserved**<br>Read as 0; should be written with 0. |
| **LINPRES** | [25:16] | rw | **Linear Walker Clock Prescaler**<br>Determines how long it takes for the Channel Intensity to reach the Target Channel Intensity after shadow transfer (after CHSTRCON.CHyS is set). Necessary for smooth linear color transitions. If this value is 0, then the intensity level will become the same as the target intensity level on shadow transfer and the linear walker is bypassed. |
| **0** | [31:26] | r | **Reserved**<br>Read as 0; should be written with 0. |

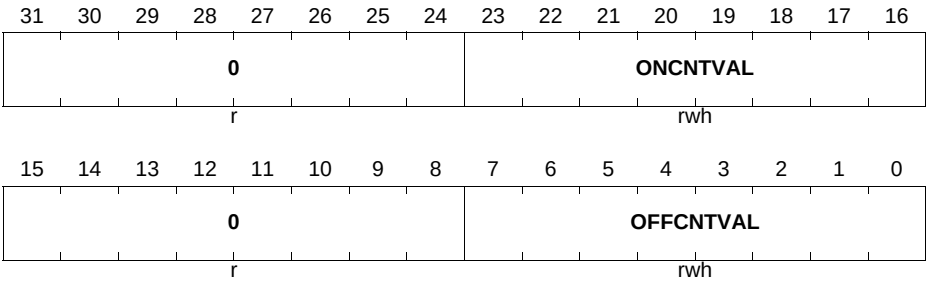
**PKCMPy (y=0-8)**
**Packer Compare** **($0048_H$ + y*$0014_H$)** **Reset Value: $0004\ 0060_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **OFFCMP** | [7:0] | rw | **Packer Off-Time Compare Level**<br>When the off-time counter in the packer reaches this value, the measured on-time and off-time of the output of the sigma-delta modulator are stored; and the counters reset to 0. |
| **0** | [15:8] | r | **Reserved**<br>Read as 0; should be written with 0. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **ONCMP** | [23:16] | rw | **Packer On-Time Compare Level**<br>When the on-time counter in the packer reaches this value, the measured on-time and off-time of the output of the sigma-delta modulator are stored; and the counters reset to 0. |
| **0** | [31:24] | r | **Reserved**<br>Read as 0; should be written with 0. |

**PKCNTRy (y=0-8)**
**Packer Counter** **($004C_H$ + y*$0014_H$)** **Reset Value: $0000\ 0000_H$**



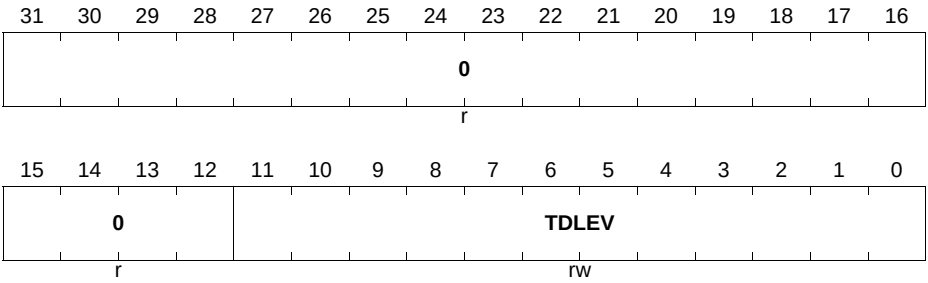

| Field | Bits | Type | Description |
|---|---|---|---|
| **OFFCNTVAL** | [7:0] | rwh | **Off-Time Counter Value**<br>Should be written when the channel is disabled (BCCU_**CHEN**.ECHy is 0).<br>Ensure that the packer is enabled (BCCU_**CHCONFIGy (y=0-8)**.PEN is 1) before writing this bit field. |
| **0** | [15:8] | r | **Reserved**<br>Read as 0; should be written with 0. |
| **ONCNTVAL** | [23:16] | rwh | **On-Time Counter Value**<br>Should be written when the channel is disabled (BCCU_**CHEN**.ECHy is 0).<br>Ensure that the packer is enabled (BCCU_**CHCONFIGy (y=0-8)**.PEN is 1) before writing this bit field. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | [31:24] | r | **Reserved**<br>Read as 0; should be written with 0. |

### 22.8.3 Dimming Engine Registers

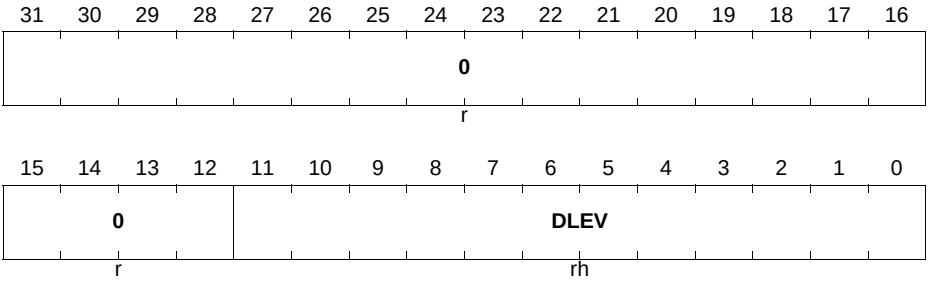
There are 3 dimming engines available on XMC1300. z denotes engine 0 through 2.

**DLSz (z=0-2)**
**Dimming Level Shadow** **($017C_H$ + z*$000C_H$)** **Reset Value: 0000 $0000_H$**



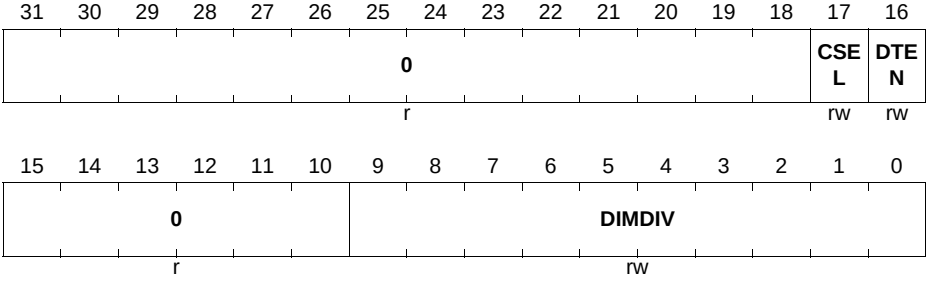

| Field | Bits | Type | Description |
|---|---|---|---|
| **TDLEV** | [11:0] | rw | **Target Dimming Level** |
| **0** | [31:12] | r | **Reserved**<br>Read as 0; should be written with 0. |

**DLz (z=0-2)**
**Dimming Level** **($0180_H$ + z*$000C_H$)** **Reset Value: 0000 $0000_H$**



| Field | Bits | Type | Description |
|---|---|---|---|
| **DLEV** | [11:0] | rh | **Dimming Level** |
| **0** | [31:12] | r | **Reserved**<br>Read as 0; should be written with 0. |

**DTTz (z=0-2)**
**Dimming Transition Time** **($0184_H$ + z*$000C_H$)** **Reset Value: 0000 $0000_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **DIMDIV** | [9:0] | rw | **Dimming Clock Divider**<br>BCCU_dclk is prescaled according to this value to generate the dimming clock; this value can be used to adjust the fade rate of the dimming engine. If this value is 0, then the dimming level will become the same as the target dimming level on shadow transfer and the dimming engine is bypassed. |
| **0** | [15:10] | r | **Reserved**<br>Read as 0; should be written with 0. |
| **DTEN** | 16 | rw | **Dither Enable**<br>$0_B$ No dithering<br>$1_B$ Dithering is added to every dimming step if the dimming level is below 128; the coarse curve is used for the entire dimming range |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **CSEL** | 17 | rw | **Curve Select**<br>$0_B$ Coarse curve<br>$1_B$ Fine curve<br>If dithering is enabled (DTEN=1), this bit is is ignored and the coarse curve is used.<br>Switching between curves in application is not supported. |
| **0** | [31:18] | r | **Reserved**<br>Read as 0; should be written with 0. |

## 22.9 Interconnects

BCCU0 is connected to the ports, the ERU, the analog comparator, the ADC, the NVIC, CCU4, CCU8, the suspend signal and the kernel clock.

**Table 22-7 Pin Connections**

| Input/Output | I/O | Connected To | Description |
|---|---|---|---|
| BCCU0_clk | I | PCLK | BCCU kernel clock |
| SUSPEND | I | CPU Halted | Suspend signal |
| BCCU0.IN0 | I | ACMP1.OUT | Gating Input of channel 0 |
| BCCU0.IN1 | I | not connected (1) | Gating Input of channel 1 |
| BCCU0.IN2 | I | not connected (1) | Gating Input of channel 2 |
| BCCU0.IN3 | I | ACMP2.OUT | Gating Input of channel 3 |
| BCCU0.IN4 | I | not connected (1) | Gating Input of channel 4 |
| BCCU0.IN5 | I | ACMP0.OUT | Gating Input of channel 5 |
| BCCU0.IN6 | I | not connected (1) | Gating Input of channel 6 |
| BCCU0.IN7 | I | not connected (1) | Gating Input of channel 7 |
| BCCU0.IN8 | I | not connected (1) | Gating Input of channel 8 |
| BCCU0.TRAPINA | I | P0.12 | Trap input A |
| BCCU0.TRAPINB | I | P0.0 | Trap input B |
| BCCU0.TRAPINC | I | not connected (0) | Trap input C |
| BCCU0.TRAPIND | I | not connected (0) | Trap input D |
| 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 | not connected (0) | Trap input J |
| BCCU0.TRAPINK | I | not connected (0) | Trap input K |
| BCCU0.TRAPINL | I | not connected (0) | Trap input L |
| BCCU0.TRAPINM | I | not connected (0) | Trap input M |
| BCCU0.TRAPINN | I | not connected (0) | Trap input N |
| BCCU0.TRAPINO | I | not connected (0) | Trap input O |
| BCCU0.TRAPINP | I | not connected (0) | Trap input P |


Subject to Agreement on the Use of Product Information

**Table 22-7 Pin Connections** (cont'd)

| Input/Output | I/O | Connected To | Description |
|---|---|---|---|
| BCCU0.OUT0 | O | P0.4<br>P1.0<br>CCU80.IN0I<br>P2.2.HW0 pull control | Output of channel 0 |
| BCCU0.OUT1 | O | P0.5<br>P1.5<br>CCU80.IN0J<br>P2.0.HW0 pull control<br>P2.8.HW0 pull control | Output of channel 1 |
| BCCU0.OUT2 | O | P0.6<br>P1.6<br>CCU40.MCI0<br>CCU80.IN1I<br>P1.0.HW0 pull control<br>P2.6.HW0 pull control | Output of channel 2 |
| BCCU0.OUT3 | O | P0.7<br>CCU40.MCI1<br>CCU80.IN1J<br>P1.1.HW0 pull control | Output of channel 3 |
| BCCU0.OUT4 | O | P0.8<br>CCU40.MCI2<br>CCU80.IN2I<br>P1.2.HW0 pull control<br>P2.10.HW0 pull control | Output of channel 4 |
| BCCU0.OUT5 | O | P0.9<br>CCU40.MCI3<br>CCU80.IN2J<br>P1.3.HW0 pull control<br>P2.11.HW0 pull control | Output of channel 5 |
| BCCU0.OUT6 | O | P0.10<br>P0.12<br>CCU80.IN3I<br>P1.4.HW0 pull control<br>P2.1.HW0 pull control | Output of channel 6 |


Subject to Agreement on the Use of Product Information

**Table 22-7 Pin Connections** (cont'd)

| Input/Output | I/O | Connected To | Description |
|---|---|---|---|
| BCCU0.OUT7 | O | P0.11<br>P0.14<br>CCU80.IN3J<br>P1.5.HW0 pull control<br>P2.9.HW0 pull control | Output of channel 7 |
| BCCU0.OUT8 | O | P0.1<br>P0.15<br>P1.6.HW0 pull control<br>P2.4.HW0 pull control<br>P2.7.HW0 pull control | Output of channel 8 |
| Service Request Connectivity | | | |
| BCCU0.SR0 | O | NVIC | Service Request Node 0: Interrupt |
| BCCU0.TRIGOUT0 | O | VADC0.BGREQTRF<br>VADC0.G0REQTRF | ADC trigger signal 0 |
| BCCU0.TRIGOUT1 | O | VADC0.G1REQTRF | ADC trigger signal 1 |


Subject to Agreement on the Use of Product Information

# General Purpose I/O Ports


Subject to Agreement on the Use of Product Information

# 23 General Purpose I/O Ports (Ports)

The XMC1300 has 35 digital General Purpose Input/Output (GPIO) port lines which are connected to the on-chip peripheral units.

## 23.1 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 23-1** gives an overview of the available PORTS and other pins in the different packages of the XMC1300:

**Table 23-1 Port/Pin Overview**

| Port | VQFN-40 | TSSOP-38 | VQFN-24 | TSSOP-16 | Note |
|---|---|---|---|---|---|
| P0 | 16 | 16 | 10 | 8 | Standard bi-directional pad |
| P1 | 7 | 6 | 4 | - | High current and Standard bi-directional pad |
| P2 | 12 | 12 | 8 | 6 | Analog/Digital input and bi-directional pad |
| Supply VDD, ADC / ORC Reference Voltage | 1 | 1 | 1 | 1 | VDD |
| Supply GND, ADC Reference Ground | 1 | 1 | 1 | 1 | VSS |
| I/O Port Supply | 1 | 1 | - | - | VDDP |
| I/O Port Ground | 1 | 1 | - | - | VSSP |

### 23.1.1 Features

This is a list of the main features of the Ports:

- same generic register interface for each port pin, **Chapter 23.8**
- simple and robust software access for general purpose I/O functionality, **Chapter 23.2**
- direct input connections to on-chip peripherals, **Chapter 23.2.1**


Subject to Agreement on the Use of Product Information

- dedicated hardware interface for BCCU, CCU and USIC with select option, **Chapter 23.3**
- defined power-up/power-fail behavior, **Chapter 23.6**.
- up to seven alternate output paths from peripherals selectable, **Chapter 23.8.1**
- programmable open-drain or push-pull output driver stage, **Chapter 23.8.1**
- programmable weak pull-up and pull-down devices, **Chapter 23.8.1**
- programmable input inverter, **Chapter 23.8.1**
- programmable pad hysteresis, **Chapter 23.8.2**
- disabling of digital input stage on shared analog inputs, **Chapter 23.8.3**
- separate set and clear output control to avoid read-modify-write operations, **Chapter 23.8.5**
- programmable power-save behavior in Deep-Sleep mode, **Chapter 23.8.7**
- Privileged Mode restricted access to configuration registers to avoid accidential modification

### 23.1.2 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 23-1**.


Subject to Agreement on the Use of Product Information

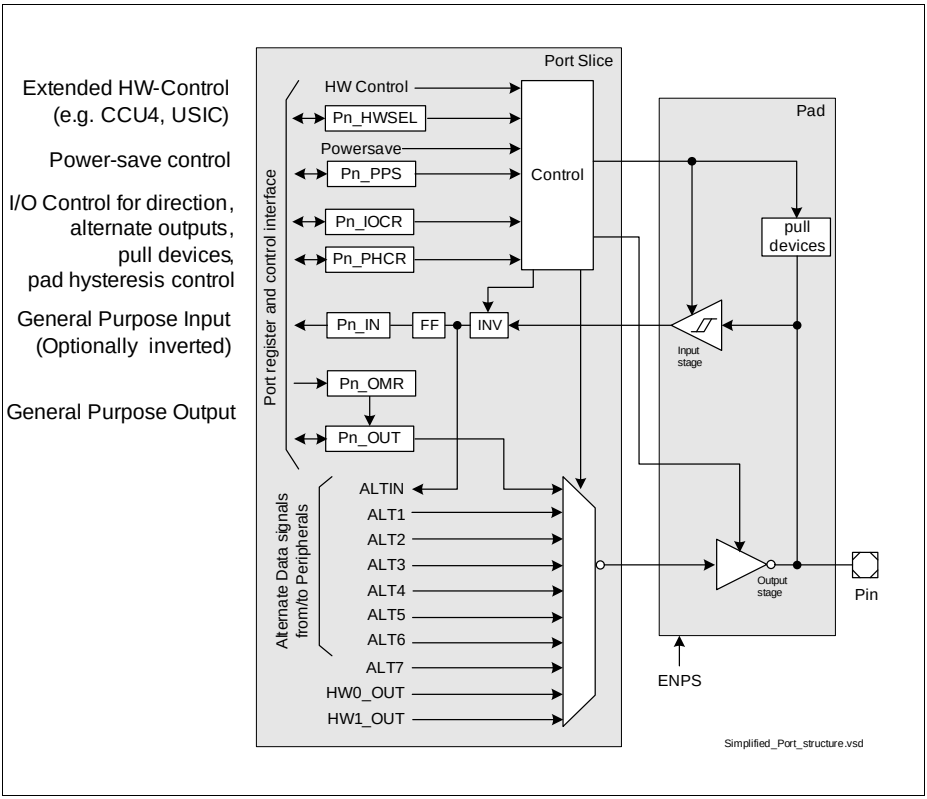



**Figure 23-1 General Structure of a digital Port Pin**

### 23.1.3 Definition of Terms

Some specific terms are used throughout this chaper:

- **Pin/Ball:** External connection of the device to the PCB.
- **Dedicated Pin:** A Pin with a dedicated function that is not under the control of the port logic (i.e. supply pins).
- **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.

## 23.2 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, **Chapter 23.8.1**). It selects between

- Direct or Inverted Input
  - with or without pull device
- Push-pull or Open-Drain Output driven by
  - Pn_OUT (GPIO)
  - selected peripheral output connections.

As GPIO the port pin is controlled by the application software, reading the input value by the Port Input register Pn_IN (**Chapter 23.8.6**) and/or defining the output value by the Output Modification Register Pn_OMR (**Chapter 23.8.5**). Output modification by Pn_OMR is preferred over the direct change of the output value with the Output register Pn_OUT (**Chapter 23.8.4**), as Pn_OMR allows the manipulation of individual port pins in a single access without “disturbing” other pins controlled by the same Pn_OUT register. If an application uses a GPIO as a bi-directional I/O line, register Pn_IOCR has to be written to switch between input and output functionality.

For the operation with Alternate Functions, the port pins are directly connected to input or output functions of the on-chip periphery. This allows the peripheral to directly evaluate the input value or drive the output value of the port pin without further application software interaction after the initial configuration. The connection of alternate functions is used for control and communication interfaces, like a PWM from a CAPCOM unit or a SPI communication of a USIC channel. A detailed connectivity list of the peripherals to the port pins is given in the **Port I/O Function Description** chapter. For specific functions, certain peripherals may also take direct control of “their” port pins, see **Hardware Controlled I/Os**.

### 23.2.1 Input Operation

As an input, the actual voltage level at the port pin is translated into a logical $0_B$ or $1_B$ via a Schmitt-Trigger device within the pad. The resulting input value can be optionally inverted. As general purpose input, the signal is synchronized and can be read with the Input register (Pn_IN, **Chapter 23.8.6**). Alternatively, the input can be connected to the multiple on-chip peripherals via the ALTIN signal. Where neccessary, these peripherals have internal controls to select the appropriate port pin with an input multiplexer stage, and will take care of synchronization and further processing of the input signals. (See respective peripheral chapters for more details on the input selection and handling). With the Pn_IOCR register (**Chapter 23.8.1**), it is also possible to activate an internal weak

pull-up or pull-down device in the pad.
The input register Pn_IN and 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, **Chapter 23.8.2**). Selecting the appropriate pad hysteresis allows optimized pad oscillation behaviour for touch-sensing applications.

### 23.2.2 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 (**Chapter 23.8.1**), which

- enables or disables the output driver,
- selects between open-drain and push-pull mode,
- selects the general purpose or alternate function outputs.

The output multiplexer selects the signal source of the output with

- Pn_IOCR
  - general purpose output (Pn_OUT, **Chapter 23.8.4**)
  - alternate peripheral functions, ALT1...ALT7
- 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 (e.g. P0) that are configured as general purpose output. Updating just one or a selected few general purpose output pins via Pn_OUT requires a masked read-modify-write operation to avoid disturbing pins that shall not be changed. Direct writes to Pn_OUT will also affect Pn_OUT bits configured for use with the Pin Power-save function, **Chapter 23.4**.

Because of that, it is preferred to modify Pn_OUT bits by the Output Modification Register Pn_OMR (**Chapter 23.8.5**). The bits in Pn_OMR allow to individually set, clear or toggle the bits in the Pn_OUT register and only update the “addressed” Pn_OUT bits.


Subject to Agreement on the Use of Product Information

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 ALT7 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 two pad types in XMC1300 providing different drive strength:

- Standard pad
- 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 Data Sheet.

## 23.3 Hardware Controlled I/Os

Some ports pins are overlaid with peripheral functions for which the connected peripheral needs direct hardware control, e.g. for the direction of a bi-directional data bus. There is a dedicated hardware control interface for these functions. As multiple peripherals need access to this interface, the Pn_HWSEL register (**Chapter 23.8.8**) 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, e.g. for bi-directional signals
- Driver type, open-drain or push-pull
- Pull devices under peripheral control or under standard control via Pn_IOCR

Some configurations remain under control by the standard configuration interface, the 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.


Subject to Agreement on the Use of Product Information

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.*

## 23.4 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, **Chapter 23.8.7**). Basically, each pin can be configured to react to the Power Save Mode Request or to ignore it. In case a pin is configured to react to a Power Save Mode Request, the output driver is switched to tri-state, the input Schmitt-Trigger and the pull devices are switched off (see **Figure 23-2**). 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, see **Table 23-8**.


Subject to Agreement on the Use of Product Information

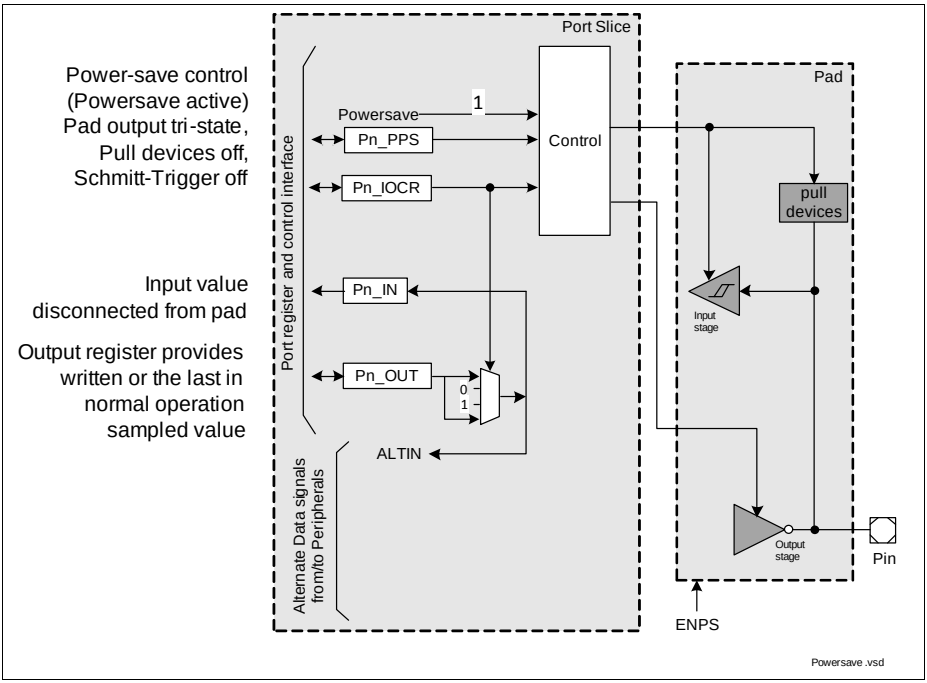



**Figure 23-2 Port Pin in Power Save State**

*Note: Do not enable the Pin Power Save function for pins configured for Hardware Control (Pn_HWSEL.HWx != $00_B$). Doing so may result in an undefined behavior of the pin when the device enters the Deep Sleep state.*

## 23.5 Analog Ports

P2.2 - P2.9 is the analog and digital input port with a simplified port and pad structure, see **Figure 23-3**. The analog pads have no output drivers and the digital input Schmitt-Trigger can be controlled by the Pn_PDISC (**Chapter 23.8.3**) register. Accordingly, the port control interface is reduced in its functionality. The Pn_IOCR register controls the pull devices, the optional input inversion and the input source in power-save mode. The Pn_OUT has only its power-save functionality, as described in **Chapter 23.4**.

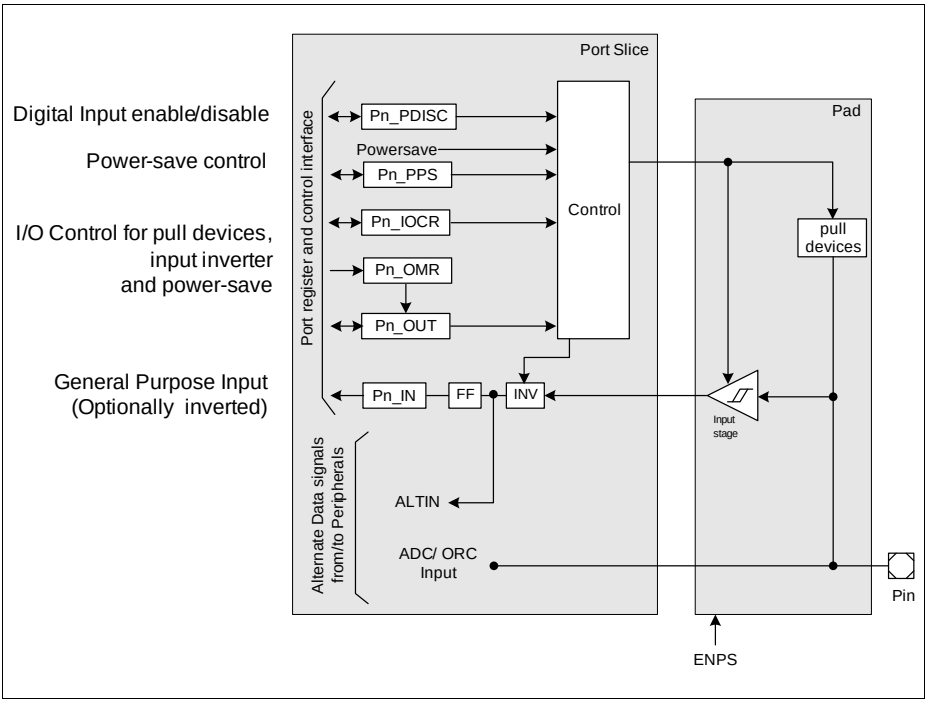



**Figure 23-3 Analog Port Structure**

## 23.6 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 Data Sheet for details on the power-up, supply monitoring and voltage limits.

All Port registers are reset with the System Reset (see Reset Control 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 $f_{MCLK}$.

## 23.7 Initialization and System Dependencies

It is recommended to follow pre-defined routines for the initialization of the port pins.

### Input

When a peripheral shall use a port pin as input, the actual pin levels may immediately trigger an unexpected peripheral event (e.g. clock edge at SPI). This can be avoided by forcing the "passive" level via pull-up/down programming.

The following steps are required to configure a port pin as an input:

- Pn_IOCR
  input configuration with pull device and/or power-save mode configuration
- 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


Subject to Agreement on the Use of Product Information

### Transitions

If a port pin is used for different functions that require a reconfiguration of the port registers, it is recommended to do this transition via an intermediate “neutral” tri-state input configuration.

- Pn_HWSEL
  disable hardware selection; can be omitted if no hardware control is used on the port pin
- Pn_PPS
  disable power save mode control of the pin; can be ommited if no power save configuration is used on the port pin
- Pn_IOCR
  tri-state input and no pull device active


Subject to Agreement on the Use of Product Information

## 23.8 Registers

### Registers Overview

The absolute register address is calculated by adding:

Module Base Address + Offset Address

**Table 23-2 Registers Address Space**

| Module | Base Address | End Address | Note |
|---|---|---|---|
| P0 | $4004\ 0000_H$ | $4004\ 00FF_H$ | |
| 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 |

**Table 23-3 Register Overview**

| Short Name | Description | Offset Addr. | Access Mode | | Description See |
|---|---|---|---|---|---|
| | | | Read | Write | |
| Pn_OUT | Port n Output Register | $0000_H$ | U, PV | U, PV | Page 23-27 |
| Pn_OMR | Port n Output Modification Register | $0004_H$ | U, PV | U, PV | Page 23-30 |
| – | Reserved | $0008_H$-$000C_H$ | BE | BE | – |
| Pn_IOCR0 | Port n Input/Output Control Register 0 | $0010_H$ | U, PV | PV | Page 23-15 |
| Pn_IOCR4 | Port n Input/Output Control Register 4 | $0014_H$ | U, PV | PV | Page 23-16 |
| Pn_IOCR8 | Port n Input/Output Control Register 8 | $0018_H$ | U, PV | PV | Page 23-17 |
| Pn_IOCR12 | Port n Input/Output Control Register 12 | $001C_H$ | U, PV | PV | Page 23-17 |
| – | Reserved | $0020_H$ | BE | BE | – |
| Pn_IN | Port n Input Register | $0024_H$ | U, PV | R | Page 23-33 |
| – | Reserved | $0028_H$-$003C_H$ | BE | BE | – |


Subject to Agreement on the Use of Product Information

**Table 23-3 Register Overview** (cont'd)

| Short Name | Description | Offset Addr. | Access Mode | | Description See |
|---|---|---|---|---|---|
| | | | Read | Write | |
| Pn_PHCR0 | Port n Pad Hysteresis Control Register 0 | $0040_H$ | U, PV | PV | Page 23-20 |
| Pn_PHCR1 | Port n Pad Hysteresis Control Register 1 | $0044_H$ | U, PV | PV | Page 23-22 |
| – | Reserved | $0048_H$-$005C_H$ | BE | BE | – |
| P0_PDISC<br>P1_PDISC | Port n Pin Function Decision Control Register (non-ADC ports) | $0060_H$ | U, PV | BE | Page 23-24 |
| P2_PDISC | Port n Pin Function Decision Control Register (ADC ports) | $0060_H$ | U, PV | PV | Page 23-25 |
| – | Reserved | $0064_H$-$006C_H$ | BE | BE | – |
| Pn_PPS | Port n Pin Power Save Register | $0070_H$ | U, PV | PV | Page 23-35 |
| Pn_HWSEL | Port n Hardware Select Register | $0074_H$ | U, PV | PV | Page 23-38 |
| – | Reserved | $0078_H$-$00FC_H$ | BE | BE | – |


Subject to Agreement on the Use of Product Information

**Table 23-4 Registers Access Rights and Reset Classes**

| Register Short Name | Access Rights | | Reset Class |
|---|---|---|---|
| | Read | Write | |
| Pn_IN | U, PV | R | System Reset |
| Pn_OUT | | U, PV | |
| Pn_OMR | | | |
| Pn_IOCR0 | | PV | |
| Pn_IOCR4 | | | |
| Pn_IOCR8 | | | |
| Pn_IOCR12 | | | |
| Pn_PDISC (ADC ports) | | | |
| Pn_PH0 | | | |
| Pn_PH1 | | | |
| Pn_PPS | | | |
| Pn_PDISC (non-ADC ports) | | BE | |


Subject to Agreement on the Use of Product Information

### 23.8.1 Port Input/Output Control Registers

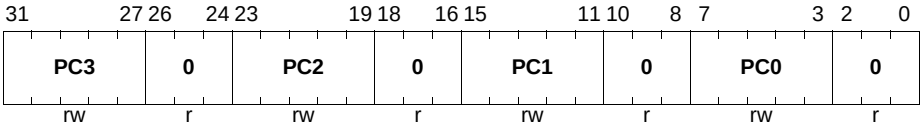
The port input/output control registers select the digital output and input driver functionality and characteristics of a GPIO port pin. Port direction (input or output), pull-up or pull-down devices for inputs, and push-pull or open-drain functionality for outputs can be selected by the corresponding bit fields PCx (x = 0-15). Each 32-bit wide port input/output control register controls four GPIO port lines:

Register Pn_IOCR0 controls the Pn.[3:0] port lines
Register Pn_IOCR4 controls the Pn.[7:4] port lines
Register Pn_IOCR8 controls the Pn.[11:8] port lines
Register Pn_IOCR12 controls the Pn.[15:12] port lines

The diagrams below show the register layouts of the port input/output control registers with the PCx bit fields. One PCx bit field controls exactly one port line Pn.x.

**Pn_IOCR0 (n=0-1)**
**Port n Input/Output Control Register 0**
**(4004 0010$_H$ + n*100$_H$)** **Reset Value: 0000 0000$_H$**

**P2_IOCR0**
**Port 2 Input/Output Control Register 0**
**(0010$_H$)** **Reset Value: 0000 0000$_H$**



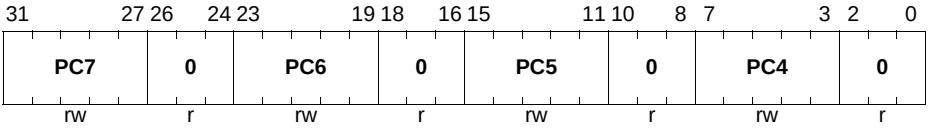

| Field | Bits | Type | Description |
|---|---|---|---|
| **PC0, PC1, PC2, PC3** | [7:3], [15:11], [23:19], [31:27] | rw | **Port Control for Port n Pin 0 to 3**<br>This bit field determines the Port n line x functionality (x = 0-3) according to the coding table (see **Table 23-5**). |
| **0** | [2:0], [10:8], [18:16], [26:24] | r | **Reserved**<br>Read as 0; should be written with 0. |

**P0_IOCR4**
**Port 0 Input/Output Control Register 4**

**($0014_H$)** **Reset Value: 0000 0000$_H$**

**P2_IOCR4**
**Port 2 Input/Output Control Register 4**

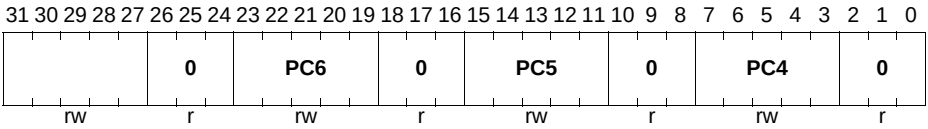
**($0014_H$)** **Reset Value: 0000 0000$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **PC4, PC5, PC6, PC7** | [7:3], [15:11], [23:19], [31:27] | rw | **Port Control for Port n Pin 4 to 7**<br>This bit field determines the Port n line x functionality (x = 4-7) according to the coding table (see **Table 23-5** ). |
| **0** | [2:0], [10:8], [18:16], [26:24] | r | **Reserved**<br>Read as 0; should be written with 0. |

**P1_IOCR4**
**Port 1 Input/Output Control Register 4**

**($0014_H$)** **Reset Value: 0000 0000$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **PC4, PC5, PC6** | [7:3], [15:11], [23:19] | rw | **Port Control for Port n Pin 4 to 6**<br>This bit field determines the Port n line x functionality (x = 4-6) according to the coding table (see **Table 23-5** ). |

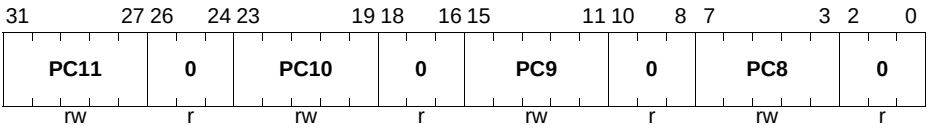
| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | [2:0], [10:8], [18:16], [26:24], [31:27] | r | **Reserved**<br>Read as 0; should be written with 0. |

**P0_IOCR8**
**Port 0 Input/Output Control Register 8**

**($0018_H$)** **Reset Value: $0000\ 0000_H$**

**P2_IOCR8**
**Port 2 Input/Output Control Register 8**

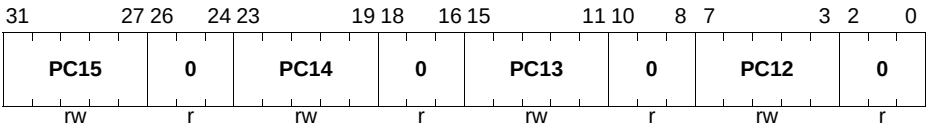
**($0018_H$)** **Reset Value: $0000\ 0000_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **PC8, PC9, PC10, PC11** | [7:3], [15:11], [23:19], [31:27] | rw | **Port Control for Port n Pin 8 to 11**<br>This bit field determines the Port n line x functionality (x = 8-11) according to the coding table (see **Table 23-5** ). |
| **0** | [2:0], [10:8], [18:16], [26:24] | r | **Reserved**<br>Read as 0; should be written with 0. |

**P0_IOCR12**
**Port 0 Input/Output Control Register 12**

**($001C_H$)** **Reset Value: $0000\ 0000_H$ 1)**




1) Upon reset, the value of PC14 (P0.14) is $00000_B$. The Startup Software (SSW) will change the PC14 value to input pull-up device active, $00010_B$. Refer to the Startup chapter for more information.

| Field | Bits | Type | Description |
|---|---|---|---|
| **PC12, PC13, PC14, PC15** | [7:3], [15:11], [23:19], [31:27] | rw | **Port Control for Port n Pin 12 to 15**<br>This bit field determines the Port n line x functionality (x = 12-15) according to the coding table (see **Table 23-5**). |
| **0** | [2:0], [10:8], [18:16], [26:24] | r | **Reserved**<br>Read as 0; should be written with 0. |

Depending on the GPIO port functionality (number of GPIO lines of a port), not all of the port input/output control registers are implemented.

The structure with one control bit field for each port pin located in different register bytes offers the possibility to configure the port pin functionality of a single pin with byte-oriented accesses without accessing the other PCx bit fields.

### Port Control Coding

**Table 23-5** describes the coding of the PCx bit fields that determine the port line functionality.

The Pn_IOCRy PCx bit field is also used to contol the pin behavior in Deep-Sleep mode if the Pin Power Save option is enabled, see **Chapter 23.8.7**.

**Table 23-5 Standard PCx Coding[1)]**

| PCx[4:0] | I/O | Output Characteristics | Selected Pull-up / Pull-down / Selected Output Function |
|---|---|---|---|
| $0X000_B$ | Direct Input | – | No internal pull device active |
| $0X001_B$ | | | Internal pull-down device active |
| $0X010_B$ | | | Internal pull-up device active |
| $0X011_B$ | | | No internal pull device active; Pn_OUTx continuously samples the input value |
| $0X100_B$ | Inverted Input | – | No internal pull device active |
| $0X101_B$ | | | Internal pull-down device active |
| $0X110_B$ | | | Internal pull-up device active |
| $0X111_B$ | | | No internal pull device active; Pn_OUTx continuously samples the input value |


Subject to Agreement on the Use of Product Information

**Table 23-5 Standard PCx Coding[1)]** (cont'd)

| PCx[4:0] | I/O | Output Characteristics | Selected Pull-up / Pull-down / Selected Output Function |
|---|---|---|---|
| $10000_B$ | Output (Direct Input) | Push-pull | General-purpose output |
| $10001_B$ | | | Alternate output function 1 |
| $10010_B$ | | | Alternate output function 2 |
| $10011_B$ | | | Alternate output function 3 |
| $10100_B$ | | | Alternate output function 4 |
| $10101_B$ | | | Alternate output function 5 |
| $10110_B$ | | | Alternate output function 6 |
| $10111_B$ | | | Alternate output function 7 |
| $11000_B$ | | Open-drain | General-purpose output |
| $11001_B$ | | | Alternate output function 1 |
| $11010_B$ | | | Alternate output function 2 |
| $11011_B$ | | | Alternate output function 3 |
| $11100_B$ | | | Alternate output function 4 |
| $11101_B$ | | | Alternate output function 5 |
| $11110_B$ | | | Alternate output function 6 |
| $11111_B$ | | | Alternate output function 7 |

1) For the analog and digital input port P2.2 - P2.9, the combinations with PCx[4]=$1_B$ is reserved.

### 23.8.2 Pad Hysteresis Control Register

The pad structure of the XMC1300 GPIO lines offers the possibility to select the pad hysteresis. These two parameters are controlled by the bit fields in the pad hysteresis control registers Pn_PHCR0/1, independently from input/output and pull-up/pull-down control functionality as programmed in the Pn_IOCR register. Pn_PHCR0 and Pn_PHCR1 registers are assigned to each port.

The 1-bit pad hysteresis selection bit field PHx in the pad hysteresis control registers Pn_PHCR make it possible to select the port line functionality as shown in **Table 23-6**. Note that the pad hysteresis control registers are specific for each port.

**Table 23-6 Pad Hysteresis Selection**

| PHx | Functionality |
|---|---|
| 0 | Standard hysteresis |
| 1 | Large hysteresis |


Subject to Agreement on the Use of Product Information

*Note: Refer to Input/Output Characteristics table in the XMC1300 Data Sheet for hysteresis parameter values.*

### Pad Hysteresis Control Registers

This is the general description of the PHCR registers. Each port contains its own specific PHCR registers, described additionally at each port, that can contain between one and eight PHx fields for PHCR0 and PHCR1 registers, respectively. Each field controls 1 pin. For coding of PHx, see **Page 23-19**.

**P0_PHCR0**
**Port 0 Pad Hysteresis Control Register 0**
**($0040_H$)** **Reset Value: 0000 0000$_H$**

**P2_PHCR0**
**Port 2 Pad Hysteresis Control Register 0**
**($0040_H$)** **Reset Value: 0000 0000$_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **PH7** | **0** | | | **PH6** | **0** | | | **PH5** | **0** | | | **PH4** | **0** | |
| r | rw | r | | | rw | r | | | rw | r | | | rw | r | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **PH3** | **0** | | | **PH2** | **0** | | | **PH1** | **0** | | | **PH0** | **0** | |
| r | rw | r | | | rw | r | | | rw | r | | | rw | r | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **PH0** | 2 | rw | **Pad Hysteresis for Pn.0** |
| **PH1** | 6 | rw | **Pad Hysteresis for Pn.1** |
| **PH2** | 10 | rw | **Pad Hysteresis for Pn.2** |
| **PH3** | 14 | rw | **Pad Hysteresis for Pn.3** |
| **PH4** | 18 | rw | **Pad Hysteresis for Pn.4** |
| **PH5** | 22 | rw | **Pad Hysteresis for Pn.5** |
| **PH6** | 26 | rw | **Pad Hysteresis for Pn.6** |
| **PH7** | 30 | rw | **Pad Hysteresis for Pn.7** |

| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | [1:0], [5:3], [9:7], [13:11], [17:15], [21:19], [25:23], [29:27], 31 | r | **Reserved**<br>Read as 0; should be written with 0. |

**P1_PHCR0**
**Port 1 Pad Hysteresis Control Register 0**

**($0040_H$)** **Reset Value: $0000\ 0000_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | | | | | **PH6** | **0** | | | **PH5** | **0** | | | **PH4** | **0** | |
| r | | | | | rw | r | | | rw | r | | | rw | r | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **PH3** | **0** | | | **PH2** | **0** | | | **PH1** | **0** | | | **PH0** | **0** | |
| r | rw | r | | | rw | r | | | rw | r | | | rw | r | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **PH0** | 2 | rw | **Pad Hysteresis for P1.0** |
| **PH1** | 6 | rw | **Pad Hysteresis for P1.1** |
| **PH2** | 10 | rw | **Pad Hysteresis for P1.2** |
| **PH3** | 14 | rw | **Pad Hysteresis for P1.3** |
| **PH4** | 18 | rw | **Pad Hysteresis for P1.4** |
| **PH5** | 22 | rw | **Pad Hysteresis for P1.5** |
| **PH6** | 26 | rw | **Pad Hysteresis for P1.6** |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | [1:0], [5:3], [9:7], [13:11], [17:15], [21:19], [25:23], [31:27] | r | **Reserved**<br>Read as 0; should be written with 0. |

**P0_PHCR1**
**Port 0 Pad Hysteresis Control Register 1**

**(0044$_H$)** **Reset Value: 0000 0000$_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **PH15** | **0** | | | **PH14** | **0** | | | **PH13** | **0** | | | **PH12** | **0** | |
| r | rw | r | | | rw | r | | | rw | r | | | rw | r | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **PH11** | **0** | | | **PH10** | **0** | | | **PH9** | **0** | | | **PH8** | **0** | |
| r | rw | r | | | rw | r | | | rw | r | | | rw | r | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **PH8** | 2 | rw | **Pad Hysteresis for P0.8** |
| **PH9** | 6 | rw | **Pad Hysteresis for P0.9** |
| **PH10** | 10 | rw | **Pad Hysteresis for P0.10** |
| **PH11** | 14 | rw | **Pad Hysteresis for P0.11** |
| **PH12** | 18 | rw | **Pad Hysteresis for P0.12** |
| **PH13** | 22 | rw | **Pad Hysteresis for P0.13** |
| **PH14** | 26 | rw | **Pad Hysteresis for P0.14** |
| **PH15** | 30 | rw | **Pad Hysteresis for P0.15** |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | [1:0], [5:3], [9:7], [13:11], [17:15], [21:19], [25:23], [29:27], 31 | r | **Reserved**<br>Read as 0; should be written with 0. |

**P2_PHCR1**
**Port 2 Pad Hysteresis Control Register 1**

**(0044$_H$)** **Reset Value: 0000 0000$_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **PH11** | **0** | | | **PH10** | **0** | | | **PH9** | **0** | | | **PH8** | **0** | |
| r | rw | r | | | rw | r | | | rw | r | | | rw | r | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **PH8** | 2 | rw | **Pad Hysteresis for P2.8** |
| **PH9** | 6 | rw | **Pad Hysteresis for P2.9** |
| **PH10** | 10 | rw | **Pad Hysteresis for P2.10** |
| **PH11** | 14 | rw | **Pad Hysteresis for P2.11** |
| **0** | [1:0], [5:3], [9:7], [13:11], [31:15] | r | **Reserved**<br>Read as 0; should be written with 0. |


Subject to Agreement on the Use of Product Information

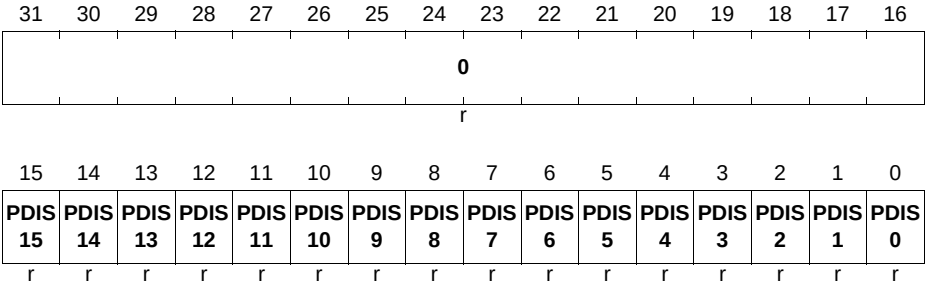
### 23.8.3 Pin Function Decision Control Register

**Pin Function Decision Control Register**

The primary use for this register is to disable/enable the digital pad structure in shared analog and digital ports, see the dedicated description for **P2_PDISC**.

For “normal” digital I/O ports (P0-P1) this register is read-only and the read value corresponds to the available pins in the given package.

**P0_PDISC**
**Port 0 Pin Function Decision Control Register**
**($0060_H$)** **Reset Value: 0000 XXXX$_H$[1)]**




1) The reset value is package dependent.

| Field | Bits | Type | Description |
|---|---|---|---|
| **PDISx (x = 0-15)** | x | r | **Pad Disable for Port 0 Pin x**<br>$0_B$ Pad P0.x is enabled.<br>$1_B$ Pad P0.x is disabled. |
| **0** | [31:16] | r | **Reserved**<br>Read as 0; should be written with 0. |

## P1_PDISC
## Port 1 Pin Function Decision Control Register

**($0060_H$)** **Reset Value: 0000 00XX$_H$[1)]**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | | PDIS6 | PDIS5 | PDIS4 | PDIS3 | PDIS2 | PDIS1 | PDIS0 |
| r | | | | | | | | | r | r | r | r | r | r | r |

1) The reset value is package dependent.

| Field | Bits | Type | Description |
|---|---|---|---|
| **PDISx (x = 0-6)** | x | r | **Pad Disable for Port 1 Pin x**<br>$0_B$ Pad P1.x is enabled.<br>$1_B$ Pad P1.x is disabled. |
| **0** | [31:7] | r | **Reserved**<br>Read as 0; should be written with 0. |

## P2_PDISC
## Port 2 Pin Function Decision Control Register

**($0060_H$)** **Reset Value: 0000 0XXX$_H$[1)]**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | PDIS11 | PDIS10 | PDIS9 | PDIS8 | PDIS7 | PDIS6 | PDIS5 | PDIS4 | PDIS3 | PDIS2 | PDIS1 | PDIS0 |
| r | | | | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

1) The reset value is package dependent.

| Field | Bits | Type | Description |
|---|---|---|---|
| **PDIS0, PDIS1** | 0, 1 | rw | **Pad Disable for Port 2 Pin 0 to 1**<br>This bit disables or enables the digital pad function.<br>$0_B$ Digital Pad input is enabled. Analog and digital input/output path active.<br>$1_B$ Digital Pad input is disabled. Analog input path active. (default) |
| **PDISx (x = 2-9)** | x | rw | **Pad Disable for Port 2 Pin 2 to 9**<br>This bit disables or enables the digital pad function.<br>$0_B$ Digital Pad input is enabled. Analog and digital input path active.<br>$1_B$ Digital Pad is disabled. Analog input path active. (default) |
| **PDIS10, PDIS11** | 10, 11 | rw | **Pad Disable for Port 2 Pin 10 to 11**<br>This bit disables or enables the digital pad function.<br>$0_B$ Digital Pad input is enabled. Analog and digital input/output path active.<br>$1_B$ Digital Pad input is disabled. Analog input path active. (default) |
| **0** | [31:12] | r | **Reserved**<br>Read as 0; should be written with 0. |


Subject to Agreement on the Use of Product Information

### 23.8.4 Port Output Register

The port output register determines the value of a GPIO pin when it is selected by Pn_IOCRx as output. Writing a 0 to a Pn_OUT.Px (x = 0-15) bit position delivers a low level at the corresponding output pin. A high level is output when the corresponding bit is written with a 1. Note that the bits of Pn_OUT.Px can be individually set/reset by writing appropriate values into the port output modification register Pn_OMR, avoiding read-modify-write operations on the Pn_OUT, which might affect other pins of the port.

The Pn_OUT is also used to store/drive a defined value for the input in Deep-Sleep mode. For details on this, see the **Port Pin Power Save Register**. That is also the only use of the Pn_OUT register in the analog and digital input port P2.2 - P2.9.

**P0_OUT**
**Port 0 Output Register** **($0000_H$)** **Reset Value: 0000 $0000_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **Px (x = 0-15)** | x | rwh | **Port 0 Output Bit x**<br>This bit determines the level at the output pin P0.x if the output is selected as GPIO output.<br>$0_B$ The output level of P0.x is 0.<br>$1_B$ The output level of P0.x is 1.<br>P0.x can also be set/reset by control bits of the P0_OMR register. |
| **0** | [31:16] | r | **Reserved**<br>Read as 0; should be written with 0. |

**P1_OUT**
**Port 1 Output Register** **($0000_H$)** **Reset Value: 0000 $0000_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| r | | | | | | | | | rwh | rwh | rwh | rwh | rwh | rwh | rwh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **Px**<br>**(x = 0-6)** | x | rwh | **Port 1 Output Bit x**<br>This bit determines the level at the output pin P1.x if the output is selected as GPIO output.<br>$0_B$ The output level of P1.x is 0.<br>$1_B$ The output level of P1.x is 1.<br>P1.x can also be set/reset by control bits of the P1_OMR register. |
| **0** | [31:7] | r | **Reserved**<br>Read as 0; should be written with 0. |

**P2_OUT**
**Port 2 Output Register** **($0000_H$)** **Reset Value: 0000 $0000_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | P11 | P10 | P9 | P8 | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| r | | | | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **Px**<br>**(x = 0-11)** | x | rwh | **Port 2 Output Bit x**<br>This bit determines the level at the output pin P2.x if the output is selected as GPIO output.<br>$0_B$ The output level of P2.x is 0.<br>$1_B$ The output level of P2.x is 1.<br>P2.x can also be set/reset by control bits of the P2_OMR register. |
| **0** | [31:12] | r | **Reserved**<br>Read as 0; should be written with 0. |


Subject to Agreement on the Use of Product Information

### 23.8.5 Port Output Modification Register

The port output modification register contains control bits that make it possible to individually set, reset, or toggle the logic state of a single port line by manipulating the output register.

**P0_OMR**
**Port 0 Output Modification Register**

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

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **PR15** | **PR14** | **PR13** | **PR12** | **PR11** | **PR10** | **PR9** | **PR8** | **PR7** | **PR6** | **PR5** | **PR4** | **PR3** | **PR2** | **PR1** | **PR0** |
| w | w | w | w | w | w | w | w | w | w | w | w | w | w | w | w |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **PS15** | **PS14** | **PS13** | **PS12** | **PS11** | **PS10** | **PS9** | **PS8** | **PS7** | **PS6** | **PS5** | **PS4** | **PS3** | **PS2** | **PS1** | **PS0** |
| w | w | w | w | w | w | w | w | w | w | w | w | w | w | w | w |

| Field | Bits | Type | Description |
|---|---|---|---|
| **PSx (x = 0-15)** | x | w | **Port 0 Set Bit x**<br>Setting this bit will set or toggle the corresponding bit in the port output register P0_OUT. The function of this bit is shown in **Table 23-7**. |
| **PRx (x = 0-15)** | x + 16 | w | **Port 0 Reset Bit x**<br>Setting this bit will reset or toggle the corresponding bit in the port output register P0_OUT. The function of this bit is shown in **Table 23-7**. |


Subject to Agreement on the Use of Product Information

### P1_OMR
### Port 1 Output Modification Register

(0004$_H$) Reset Value: 0000 0000$_H$

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | | PR6 | PR5 | PR4 | PR3 | PR2 | PR1 | PR0 |
| r | | | | | | | | | w | w | w | w | w | w | w |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | | PS6 | PS5 | PS4 | PS3 | PS2 | PS1 | PS0 |
| r | | | | | | | | | w | w | w | w | w | w | w |

| Field | Bits | Type | Description |
|---|---|---|---|
| **PSx (x = 0-6)** | x | w | **Port 1 Set Bit x**<br>Setting this bit will set or toggle the corresponding bit in the port output register P1_OUT. The function of this bit is shown in **Table 23-7**. |
| **PRx (x = 0-6)** | x + 16 | w | **Port 1 Reset Bit x**<br>Setting this bit will reset or toggle the corresponding bit in the port output register P1_OUT. The function of this bit is shown in **Table 23-7**. |
| **0** | [15:7], [31:23] | r | **Reserved**<br>Read as 0; should be written with 0. |

### P2_OMR
### Port 2 Output Modification Register

(0004$_H$) Reset Value: 0000 0000$_H$

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | PR11 | PR10 | PR9 | PR8 | PR7 | PR6 | PR5 | PR4 | PR3 | PR2 | PR1 | PR0 |
| r | | | | w | w | w | w | w | w | w | w | w | w | w | w |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | PS11 | PS10 | PS9 | PS8 | PS7 | PS6 | PS5 | PS4 | PS3 | PS2 | PS1 | PS0 |
| r | | | | w | w | w | w | w | w | w | w | w | w | w | w |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **PSx (x = 0-11)** | x | w | **Port 2 Set Bit x**<br>Setting this bit will set or toggle the corresponding bit in the port output register P2_OUT. The function of this bit is shown in **Table 23-7**. |
| **PRx (x = 0-11)** | x + 16 | w | **Port 2 Reset Bit x**<br>Setting this bit will reset or toggle the corresponding bit in the port output register P2_OUT. The function of this bit is shown in **Table 23-7**. |
| **0** | [15:12], [31:28] | r | **Reserved**<br>Read as 0; should be written with 0. |

*Note: Register Pn_OMR is virtual and does not contain any flip-flop. A read action delivers the value of 0. A 8 or 16-bits write behaves like a 32-bit write padded with zeros.*

**Table 23-7 Function of the Bits PRx and PSx**

| PRx | PSx | Function |
|---|---|---|
| 0 | 0 | Bit Pn_OUT.Px is not changed. |
| 0 | 1 | Bit Pn_OUT.Px is set. |
| 1 | 0 | Bit Pn_OUT.Px is reset. |
| 1 | 1 | Bit Pn_OUT.Px is toggled. |


Subject to Agreement on the Use of Product Information

### 23.8.6 Port Input Register

The logic level of a GPIO pin can be read via the read-only port input register Pn_IN. Reading the Pn_IN register always returns the current logical value at the GPIO pin, synchronized to avoid meta-stabilities, independently whether the pin is selected as input or output.

**P0_IN**
**Port 0 Input Register** **($0024_H$)** **Reset Value: 0000 XXXX$_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **P15** | **P14** | **P13** | **P12** | **P11** | **P10** | **P9** | **P8** | **P7** | **P6** | **P5** | **P4** | **P3** | **P2** | **P1** | **P0** |
| rh | rh | rh | rh | rh | rh | rh | rh | rh | rh | rh | rh | rh | rh | rh | rh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **Px**<br>**(x = 0-15)** | x | rh | **Port 0 Input Bit x**<br>This bit indicates the level at the input pin P0.x.<br>$0_B$ The input level of P0.x is 0.<br>$1_B$ The input level of P0.x is 1. |
| **0** | [31:16] | r | **Reserved**<br>Read as 0. |

**P1_IN**
**Port 1 Input Register** **($0024_H$)** **Reset Value: 0000 00XX$_H$**

| 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** | | | | | | | | | **P6** | **P5** | **P4** | **P3** | **P2** | **P1** | **P0** |
| r | | | | | | | | | rh | rh | rh | rh | rh | rh | rh |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **Px**<br>**(x = 0-6)** | x | rh | **Port 1 Input Bit x**<br>This bit indicates the level at the input pin P1.x.<br>$0_B$ The input level of P1.x is 0.<br>$1_B$ The input level of P1.x is 1. |
| **0** | [31:7] | r | **Reserved**<br>Read as 0. |

**P2_IN**
**Port 2 Input Register** **($0024_H$)** **Reset Value: 0000 0XXX$_H$**

| 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** | | | | **P11** | **P10** | **P9** | **P8** | **P7** | **P6** | **P5** | **P4** | **P3** | **P2** | **P1** | **P0** |
| r | | | | rh | rh | rh | rh | rh | rh | rh | rh | rh | rh | rh | rh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **Px**<br>**(x = 0-11)** | x | rh | **Port 2 Input Bit x**<br>This bit indicates the level at the input pin P2.x.<br>$0_B$ The input level of P2.x is 0.<br>$1_B$ The input level of P2.x is 1. |
| **0** | [31:12] | r | **Reserved**<br>Read as 0. |


Subject to Agreement on the Use of Product Information

### 23.8.7 Port Pin Power Save Register

When the XMC1300 enters Deep-Sleep mode, pins with enabled Pin Power Save option are set to a defined state and the input Schmitt-Trigger as well as the output driver stage are switched off.

*Note: Do not enable the Pin Power Save function for pins configured for Hardware Control (Pn_HWSEL.HWx != $00_B$). Doing so may result in an undefined behavior of the pin when the device enters the Deep Sleep state.*

**P0_PPS**
**Port 0 Pin Power Save Register**

**($0070_H$)** **Reset Value: $0000\ 0000_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **PPS 15** | **PPS 14** | **PPS 13** | **PPS 12** | **PPS 11** | **PPS 10** | **PPS 9** | **PPS 8** | **PPS 7** | **PPS 6** | **PPS 5** | **PPS 4** | **PPS 3** | **PPS 2** | **PPS 1** | **PPS 0** |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **PPSx (x = 0-15)** | x | rw | **Port 0 Pin Power Save Bit x**<br>$0_B$ Pin Power Save of P0.x is disabled.<br>$1_B$ Pin Power Save of P0.x is enabled. |
| **0** | [31:16] | r | **Reserved**<br>Read as 0. |

### P1_PPS
### Port 1 Pin Power Save Register

(0070$_H$) Reset Value: 0000 0000$_H$

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

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

| Field | Bits | Type | Description |
|---|---|---|---|
| **PPSx (x = 0-6)** | x | rw | **Port 1 Pin Power Save Bit x**<br>$0_B$ Pin Power Save of P1.x is disabled.<br>$1_B$ Pin Power Save of P1.x is enabled. |
| **0** | [31:7] | r | **Reserved**<br>Read as 0. |

### P2_PPS
### Port 2 Pin Power Save Register

(0070$_H$) Reset Value: 0000 0000$_H$

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

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


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **PPSx (x = 0-11)** | x | rw | **Port 2 Pin Power Save Bit x**<br>$0_B$ Pin Power Save of P2.x is disabled.<br>$1_B$ Pin Power Save of P2.x is enabled. |
| **0** | [31:12] | r | **Reserved**<br>Read as 0. |

### Deep-Sleep Pin Power Save behavior

The actual behavior in Deep-Sleep mode with enabled Pin Power Save is controlled by the Pn_IOCRy.PCx bit field (**Page 23-15**) of the respective pin. **Table 23-8** shows the coding.

**Table 23-8 PCx Coding in Deep-Sleep mode**

| PCx[4:0] | I/O | Normal Operation or PPSx=$0_B$ | Deep-Sleep mode and PPSx=$1_B$ |
|---|---|---|---|
| $0X000_B$ | Direct Input | See **Table 23-5** | Input value=Pn_OUTx |
| $0X001_B$ | | | Input value=$0_B$; pull-down deactivated |
| $0X010_B$ | | | Input value=$1_B$; pull-up deactivated |
| $0X011_B$ | | | Input value=Pn_OUTx, storing the last sampled input value |
| $0X100_B$ | Inverted Input | See **Table 23-5** | Input value=$\overline{\text{Pn_OUTx}}$ |
| $0X101_B$ | | | Input value=$1_B$; pull-down deactivated |
| $0X110_B$ | | | Input value=$0_B$; pull-up deactivated |
| $0X111_B$ | | | Input value=$\overline{\text{Pn_OUTx}}$, storing the last sampled input value |
| $1XXXX_B$ | Output | See **Table 23-5** | Output driver off,<br>Input Schmitt-Trigger off,<br>no pull device active,<br>Input value=Pn_OUTx |


Subject to Agreement on the Use of Product Information

### 23.8.8 Port Pin Hardware Select Register

Some peripherals require direct hardware control of their I/Os. As multiple such peripheral I/Os are mapped on some pins, the register Pn_HWSEL is used to select which peripheral has the control over the pin.

*Note: Pn_HWSEL.HWx just pre-assigns the hardware-control of the pin to a certain peripheral, but the peripheral itself decides 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 provisions 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 provisions to disable the hardware control selectively.

**P0_HWSEL**

**Port 0 Pin Hardware Select Register ($0074_H$)** **Reset Value: $0000\ 0000_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| HW15 | | HW14 | | HW13 | | HW12 | | HW11 | | HW10 | | HW9 | | HW8 | |
| rw | | rw | | rw | | rw | | rw | | rw | | rw | | rw | |

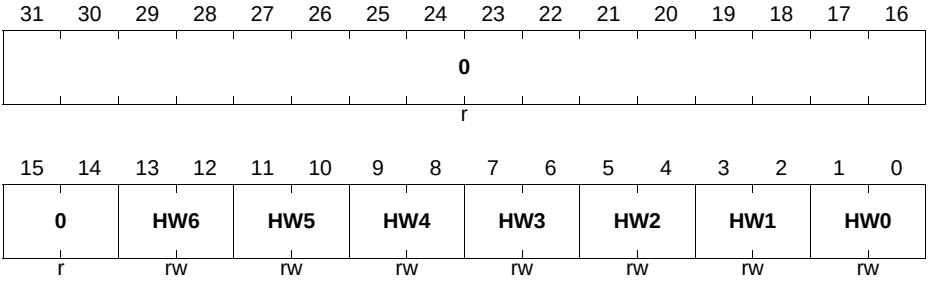
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| HW7 | | HW6 | | HW5 | | HW4 | | HW3 | | HW2 | | HW1 | | HW0 | |
| rw | | rw | | rw | | rw | | rw | | rw | | rw | | rw | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **HWx (x = 0-15)** | [2*x+1: 2*x] | rw | **Port 0 Pin Hardware Select Bit x**<br>$00_B$ Software control only.<br>$01_B$ HW0 control path can override the software configuration.<br>$10_B$ HW1 control path can override the software configuration.<br>$11_B$ Reserved. |


Subject to Agreement on the Use of Product Information

**P1_HWSEL**

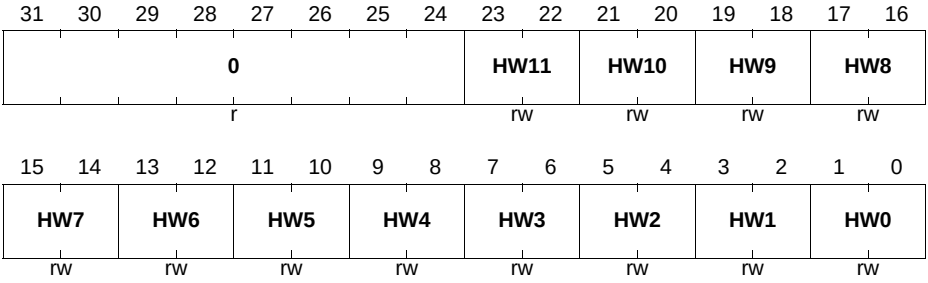
**Port 1 Pin Hardware Select Register (0074$_H$)** **Reset Value: 0000 0000$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **HWx (x = 0-6)** | [2*x+1: 2*x] | rw | **Port 1 Pin Hardware Select Bit x**<br>$00_B$ Software control only.<br>$01_B$ HW0 control path can override the software configuration.<br>$10_B$ HW1 control path can override the software configuration.<br>$11_B$ Reserved. |
| **0** | [31:14] | r | **Reserved**<br>Read as 0; should be written with 0. |

**P2_HWSEL**

**Port 2 Pin Hardware Select Register (0074$_H$)** **Reset Value: 0000 0000$_H$**





Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **HWx (x = 0-11)** | [2*x+1: 2*x] | rw | **Port 2 Pin Hardware Select Bit x**<br>$00_B$ Software control only.<br>$01_B$ HW0 control path can override the software configuration.<br>$10_B$ HW1 control path can override the software configuration.<br>$11_B$ Reserved. |
| **0** | [31:24] | r | **Reserved**<br>Read as 0; should be written with 0. |


Subject to Agreement on the Use of Product Information

## 23.9 Package Pin Summary

The following general building block is used to describe each pin:

**Table 23-9 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)
- 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 23-10 Package Pin Mapping**

| Function | VQFN 40 | TSSOP 38 | VQFN 24 | TSSOP 16 | Pad Type | Notes |
|---|---|---|---|---|---|---|
| P0.0 | 23 | 17 | 15 | 7 | STD_INOUT | |
| P0.1 | 24 | 18 | - | - | STD_INOUT | |
| P0.2 | 25 | 19 | - | - | STD_INOUT | |
| P0.3 | 26 | 20 | - | - | STD_INOUT | |
| P0.4 | 27 | 21 | - | - | STD_INOUT | |
| P0.5 | 28 | 22 | 16 | 8 | STD_INOUT | |
| P0.6 | 29 | 23 | 17 | 9 | STD_INOUT | |
| P0.7 | 30 | 24 | 18 | 10 | STD_INOUT | |
| P0.8 | 33 | 27 | 19 | 11 | STD_INOUT | |
| P0.9 | 34 | 28 | 20 | 12 | STD_INOUT | |
| P0.10 | 35 | 29 | - | - | STD_INOUT | |
| P0.11 | 36 | 30 | - | - | STD_INOUT | |
| P0.12 | 37 | 31 | 21 | - | STD_INOUT | |


Subject to Agreement on the Use of Product Information

**Table 23-10 Package Pin Mapping**

| Function | VQFN 40 | TSSOP 38 | VQFN 24 | TSSOP 16 | Pad Type | Notes |
|---|---|---|---|---|---|---|
| P0.13 | 38 | 32 | 22 | - | STD_INOUT | |
| P0.14 | 39 | 33 | 23 | 13 | STD_INOUT | |
| P0.15 | 40 | 34 | 24 | 14 | STD_INOUT | |
| P1.0 | 22 | 16 | 14 | - | High Current | |
| P1.1 | 21 | 15 | 13 | - | High Current | |
| P1.2 | 20 | 14 | 12 | - | High Current | |
| P1.3 | 19 | 13 | 11 | - | High Current | |
| P1.4 | 18 | 12 | - | - | High Current | |
| P1.5 | 17 | 11 | - | - | High Current | |
| P1.6 | 16 | - | - | - | STD_INOUT | |
| P2.0 | 1 | 35 | 1 | 15 | STD_INOUT/ AN | |
| P2.1 | 2 | 36 | 2 | - | STD_INOUT/ AN | |
| P2.2 | 3 | 37 | 3 | - | STD_IN/AN | |
| P2.3 | 4 | 38 | - | - | STD_IN/AN | |
| P2.4 | 5 | 1 | - | - | STD_IN/AN | |
| P2.5 | 6 | 2 | - | - | STD_IN/AN | |
| P2.6 | 7 | 3 | 4 | 16 | STD_IN/AN | |
| P2.7 | 8 | 4 | 5 | 1 | STD_IN/AN | |
| P2.8 | 9 | 5 | 5 | 1 | STD_IN/AN | |
| P2.9 | 10 | 6 | 6 | 2 | STD_IN/AN | |
| P2.10 | 11 | 7 | 7 | 3 | STD_INOUT/ AN | |
| P2.11 | 12 | 8 | 8 | 4 | STD_INOUT/ AN | |
| VSS | 13 | 9 | 9 | 5 | Power | Supply GND, ADC reference GND |
| VDD | 14 | 10 | 10 | 6 | Power | Supply VDD, ADC reference voltage/ ORC reference voltage |


Subject to Agreement on the Use of Product Information

**Table 23-10 Package Pin Mapping**

| Function | VQFN 40 | TSSOP 38 | VQFN 24 | TSSOP 16 | Pad Type | Notes |
|---|---|---|---|---|---|---|
| VDDP | 15 | 10 | 10 | 6 | Power | When VDD is supplied, VDDP has to be supplied with the same voltage. |
| VSSP | 31 | 25 | - | - | Power | I/O port ground |
| VDDP | 32 | 26 | - | - | Power | I/O port supply |
| 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. |


Subject to Agreement on the Use of Product Information

## 23.10 Port I/O Functions

### 23.10.1 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 23-11** shows the port pins used for the various boot modes.

**Table 23-11 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 |
| | SCLK(O) | SSC BSL mode |
| P0.15 | SWDCLK_0 | Debug mode (SWD) |
| | TX | ASC BSL full-duplex mode |
| | DATA(I/O) | SSC BSL mode |
| P1.2 | SWDCLK_1 | Debug mode (SWD) |
| | TX | ASC BSL full-duplex 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 |


Subject to Agreement on the Use of Product Information

### 23.10.2 Port I/O Function Description

The following general building block is used to describe the I/O functions of each PORT pin:

**Table 23-12 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 |

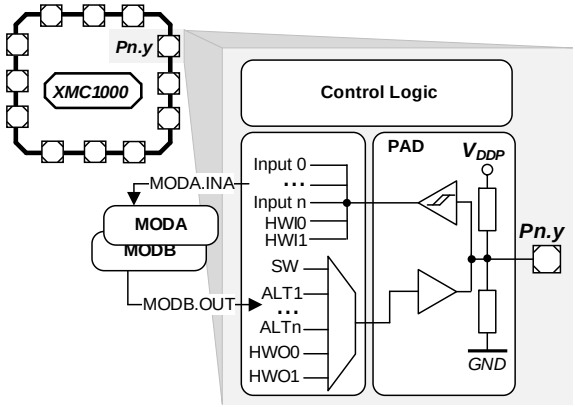



**Figure 23-4 Simplified Port Structure**

Pn.y is the port pin name, defining the control and data bits/registers associated with it. As GPIO, the port is under software control. Its input value is read via Pn_IN.y, Pn_OUT defines the output value.

Up to seven alternate output functions (ALT1/2/3/4/5/6/7) can be mapped to a single port pin, selected by Pn_IOCR.PC. The output value is directly driven by the respective module, with the pin characteristics controlled by the port registers (within the limits of the connected pad).

The port pin input can be connected to multiple peripherals. Most peripherals have an input multiplexer to select between different possible input sources.

The input path is also active while the pin is configured as output. This allows to feedback an output to on-chip resources without wasting an additional external pin.

Please refer to the **Port I/O Functions** table for the complete Port I/O function mapping.


Subject to Agreement on the Use of Product Information

### 23.10.3 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 23-13 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 (**Chapter 23.8.8**) 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 the **Hardware Controlled I/O Functions** table for the complete hardware I/O and pull control function mapping.


Subject to Agreement on the Use of Product Information

## Table 23-14 Port I/O Functions

| Function | Outputs | | | | | | | Inputs | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ALT1 | ALT2 | ALT3 | ALT4 | ALT5 | ALT6 | ALT7 | 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 | BCCU0.TRAPINB | CCU40.IN0C | | | USIC0_CH0.DX2A | USIC0_CH1.DX2A | | | | |
| P0.1 | ERU0.PDOUT1 | | ERU0.GOUT1 | CCU40.OUT1 | CCU80.OUT01 | BCCU0.OUT8 | SCU.VDROP | | CCU40.IN1C | | | | | | | | |
| P0.2 | ERU0.PDOUT2 | | ERU0.GOUT2 | CCU40.OUT2 | CCU80.OUT02 | VADC0.EMUX02 | CCU80.OUT10 | | CCU40.IN2C | | | | | | | | |
| P0.3 | ERU0.PDOUT3 | | ERU0.GOUT3 | CCU40.OUT3 | CCU80.OUT03 | VADC0.EMUX01 | CCU80.OUT11 | | CCU40.IN3C | | | | | | | | |
| P0.4 | BCCU0.OUT0 | | | CCU40.OUT1 | CCU80.OUT13 | VADC0.EMUX00 | WWDT.SERVICE_OUT | | CCU80.IN0B | | | | | | | | |
| P0.5 | BCCU0.OUT1 | | | CCU40.OUT0 | CCU80.OUT12 | ACMP2.OUT | CCU80.OUT01 | | CCU80.IN1B | | | | | | | | |
| P0.6 | BCCU0.OUT2 | | | CCU40.OUT0 | CCU80.OUT11 | USIC0_CH1.MCLKOUT | USIC0_CH1.DOUT0 | | CCU40.IN0B | | | USIC0_CH1.DX0C | | | | | |
| P0.7 | BCCU0.OUT3 | | | CCU40.OUT1 | CCU80.OUT10 | USIC0_CH0.SCLKOUT | USIC0_CH1.DOUT0 | | CCU40.IN1B | | | USIC0_CH0.DX1C | USIC0_CH1.DX0D | USIC0_CH1.DX1C | | | |
| P0.8 | BCCU0.OUT4 | | | CCU40.OUT2 | CCU80.OUT20 | USIC0_CH0.SCLKOUT | USIC0_CH1.SCLKOUT | | CCU40.IN2B | | | USIC0_CH0.DX1B | USIC0_CH1.DX1B | | | | |
| P0.9 | BCCU0.OUT5 | | | CCU40.OUT3 | CCU80.OUT21 | USIC0_CH0.SELO0 | USIC0_CH1.SELO0 | | CCU40.IN3B | | | USIC0_CH0.DX2B | USIC0_CH1.DX2B | | | | |
| P0.10 | BCCU0.OUT6 | | | ACMP0.OUT | CCU80.OUT22 | USIC0_CH0.SELO1 | USIC0_CH1.SELO1 | | CCU80.IN2B | | | USIC0_CH0.DX2C | USIC0_CH1.DX2C | | | | |
| P0.11 | BCCU0.OUT7 | | | USIC0_CH0.MCLKOUT | CCU80.OUT23 | USIC0_CH0.SELO2 | USIC0_CH1.SELO2 | | | | | USIC0_CH0.DX2D | USIC0_CH1.DX2D | | | | |
| P0.12 | BCCU0.OUT6 | | | | CCU80.OUT33 | USIC0_CH0.SELO3 | CCU80.OUT20 | BCCU0.TRAPINA | CCU40.IN0A | CCU40.IN1A | CCU40.IN2A | CCU40.IN3A | CCU80.IN0A | CCU80.IN1A | CCU80.IN2A | CCU80.IN3A | USIC0_CH0.DX2E |
| P0.13 | WWDT.SERVICE_OUT | | | | CCU80.OUT32 | USIC0_CH0.SELO4 | CCU80.OUT21 | | CCU80.IN3B | POSIF0.IN0B | | USIC0_CH0.DX2F | | | | | |

## Table 23-14 Port I/O Functions (cont'd)

| Function | Outputs | | | | | | | Inputs | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ALT1 | ALT2 | ALT3 | ALT4 | ALT5 | ALT6 | ALT7 | Input | Input | Input | Input | Input | Input | Input | Input | Input | Input |
| P0.14 | BCCU0.OUT7 | | | | CCU80.OUT31 | USIC0_CH0.DOUT0 | USIC0_CH0.SCLKOUT | | | POSIF0.IN1B | | USIC0_CH0.DX0A | USIC0_CH0.DX1A | | | | |
| P0.15 | BCCU0.OUT8 | | | | CCU80.OUT30 | USIC0_CH0.DOUT0 | USIC0_CH1.MCLKOUT | | | POSIF0.IN2B | | USIC0_CH0.DX0B | | | | | |
| P1.0 | BCCU0.OUT0 | CCU40.OUT0 | | | CCU80.OUT00 | ACMP1.OUT | USIC0_CH0.DOUT0 | | | POSIF0.IN2A | | USIC0_CH0.DX0C | | | | | |
| P1.1 | VADC0.EMUX00 | CCU40.OUT1 | | | CCU80.OUT01 | USIC0_CH0.DOUT0 | USIC0_CH1.SELO0 | | | POSIF0.IN1A | | USIC0_CH0.DX0D | USIC0_CH0.DX1D | USIC0_CH1.DX2E | | | |
| P1.2 | VADC0.EMUX01 | CCU40.OUT2 | | | CCU80.OUT10 | ACMP2.OUT | USIC0_CH1.DOUT0 | | | POSIF0.IN0A | | USIC0_CH1.DX0B | | | | | |
| P1.3 | VADC0.EMUX02 | CCU40.OUT3 | | | CCU80.OUT11 | USIC0_CH1.SCLKOUT | USIC0_CH1.DOUT0 | | | | | USIC0_CH1.DX0A | USIC0_CH1.DX1A | | | | |
| P1.4 | VADC0.EMUX10 | USIC0_CH1.SCLKOUT | | | CCU80.OUT20 | USIC0_CH0.SELO0 | USIC0_CH1.SELO1 | | | | | USIC0_CH0.DX5E | USIC0_CH1.DX5E | | | | |
| P1.5 | VADC0.EMUX11 | USIC0_CH0.DOUT0 | | BCCU0.OUT1 | CCU80.OUT21 | USIC0_CH0.SELO1 | USIC0_CH1.SELO2 | | | | | USIC0_CH1.DX5F | | | | | |
| P1.6 | VADC0.EMUX12 | USIC0_CH1.DOUT0 | | USIC0_CH0.SCLKOUT | BCCU0.OUT2 | USIC0_CH0.SELO2 | USIC0_CH1.SELO3 | | | USIC0_CH0.DX5F | | | | | | | |
| P2.0 | ERU0.PDOUT3 | CCU40.OUT0 | ERU0.GOUT3 | | CCU80.OUT20 | USIC0_CH0.DOUT0 | USIC0_CH0.SCLKOUT | | VADC0.G0CH5 | | ERU0.0B0 | USIC0_CH0.DX0E | USIC0_CH0.DX1E | USIC0_CH1.DX2F | | | |
| P2.1 | ERU0.PDOUT2 | CCU40.OUT1 | ERU0.GOUT2 | | CCU80.OUT21 | USIC0_CH0.DOUT0 | USIC0_CH1.SCLKOUT | ACMP2.INP | VADC0.G0CH6 | | ERU0.1B0 | USIC0_CH0.DX0F | USIC0_CH1.DX3A | USIC0_CH1.DX4A | | | |
| P2.2 | | | | | | | | ACMP2.INN | VADC0.G0CH7 | | ERU0.0B1 | USIC0_CH0.DX3A | USIC0_CH0.DX4A | USIC0_CH1.DX5A | ORC0.AIN | | |
| P2.3 | | | | | | | | | VADC0.G1CH5 | | ERU0.1B1 | USIC0_CH0.DX5B | USIC0_CH1.DX3C | USIC0_CH1.DX4C | ORC1.AIN | | |
| P2.4 | | | | | | | | | VADC0.G1CH6 | | ERU0.0A1 | USIC0_CH0.DX3B | USIC0_CH0.DX4B | USIC0_CH1.DX5B | ORC2.AIN | | |

**Table 23-14 Port I/O Functions** (cont'd)

| Function | Outputs | | | | | | | Inputs | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ALT1 | ALT2 | ALT3 | ALT4 | ALT5 | ALT6 | ALT7 | Input | Input | Input | Input | Input | Input | Input | Input | Input | Input |
| P2.5 | | | | | | | | | VADC0.G1CH7 | | ERU0.1A1 | USIC0_CH0.DX5D | USIC0_CH1.DX3E | USIC0_CH1.DX4E | ORC3.AIN | | |
| P2.6 | | | | | | | | ACMP1.INN | VADC0.G0CH0 | | ERU0.2A1 | USIC0_CH0.DX3E | USIC0_CH0.DX4E | USIC0_CH1.DX5D | ORC4.AIN | | |
| P2.7 | | | | | | | | ACMP1.INP | VADC0.G1CH1 | | ERU0.3A1 | USIC0_CH0.DX5C | USIC0_CH1.DX3D | USIC0_CH1.DX4D | ORC5.AIN | | |
| P2.8 | | | | | | | | ACMP0.INN | VADC0.G0CH1 | VADC0.G1CH0 | ERU0.3B1 | USIC0_CH0.DX3D | USIC0_CH0.DX4D | USIC0_CH1.DX5C | ORC6.AIN | | |
| P2.9 | | | | | | | | ACMP0.INP | VADC0.G0CH2 | VADC0.G1CH4 | ERU0.3B0 | USIC0_CH0.DX5A | USIC0_CH1.DX3B | USIC0_CH1.DX4B | ORC7.AIN | | |
| P2.10 | ERU0.PDOUT1 | CCU40.OUT2 | ERU0.GOUT1 | | CCU80.OUT30 | ACMP0.OUT | USIC0_CH1.DOUT0 | | VADC0.G0CH3 | VADC0.G1CH2 | ERU0.2B0 | USIC0_CH0.DX3C | USIC0_CH0.DX4C | USIC0_CH1.DX0F | | | |
| P2.11 | ERU0.PDOUT0 | CCU40.OUT3 | ERU0.GOUT0 | | CCU80.OUT31 | USIC0_CH1.SCLKOUT | USIC0_CH1.DOUT0 | ACMP.REF | VADC0.G0CH4 | VADC0.G1CH3 | ERU0.2B1 | USIC0_CH1.DX0E | USIC0_CH1.DX1E | | | | |

## Table 23-15 Hardware Controlled I/O Functions

| Function | Outputs | | Inputs | | Pull Control | | | |
|---|---|---|---|---|---|---|---|---|
| | HWO0 | HWO1 | HWI0 | HWI1 | HW0_PD | HW0_PU | HW1_PD | HW1_PU |
| P0.0 | | | | | | | | |
| P0.1 | | | | | | | | |
| P0.2 | | | | | | | | |
| P0.3 | | | | | | | | |
| P0.4 | | | | | | | | |
| P0.5 | | | | | | | | |
| P0.6 | | | | | | | | |
| P0.7 | | | | | | | | |
| P0.8 | | | | | | | | |
| P0.9 | | | | | | | | |
| P0.10 | | | | | | | | |
| P0.11 | | | | | | | | |
| P0.12 | | | | | | | | |
| P0.13 | | | | | | | | |
| P0.14 | | | | | | | | |
| P0.15 | | | | | | | | |
| 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 | | |

**Table 23-15 Hardware Controlled I/O Functions** (cont'd)

| Function | Outputs | | Inputs | | Pull Control | | | |
|---|---|---|---|---|---|---|---|---|
| | HWO0 | HWO1 | HWI0 | HWI1 | HW0_PD | HW0_PU | HW1_PD | HW1_PU |
| 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 | | |

# Boot and Startup, Bootstrap Loaders, and User Routines


Subject to Agreement on the Use of Product Information

# 24 Boot and Startup

The startup sequence of the XMC1300 is a process taking place before user application software takes control of the system and is comprising of two major phases (see **Figure 24-1**), 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 Data Sheet in order to enable reliable start up of the microcontroller before control is handed over to the user software. The sequence where boot code gets executed is considered a part of the hardware controlled phase of the startup sequence.

For details of the setup requirements, please refer to Data Sheet.

### 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.

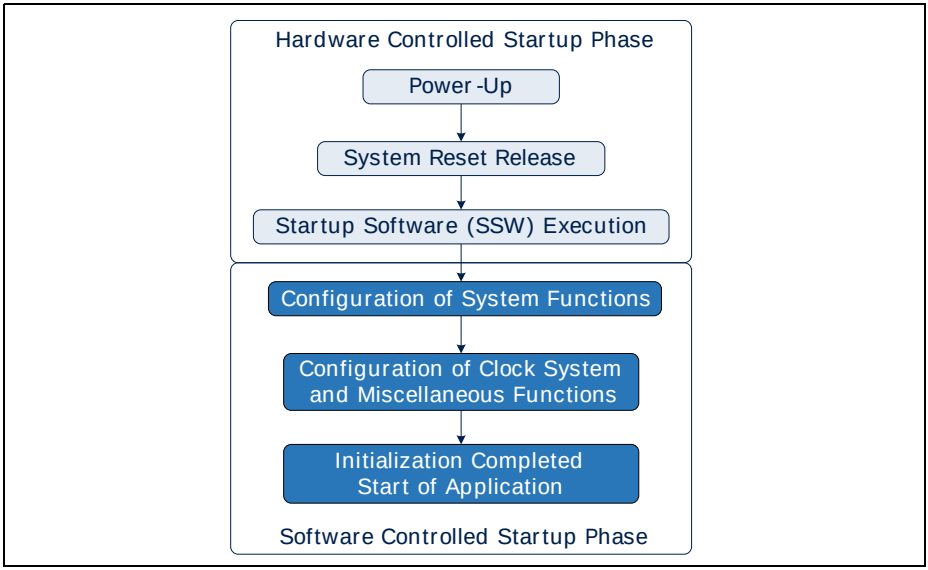



**Figure 24-1 Startup sequence**


Subject to Agreement on the Use of Product Information

## 24.1 Startup Sequence and System Dependencies

A more detailed description of the startup sequence is available in the following sub-chapters.

### 24.1.1 Power-Up

Power up of the microcontroller gets performed by applying $V_{DDP}$ supply (for details of the supply requirements, please refer to Data Sheet). Once $V_{DDP}$ reaches a threshold, the internal core voltage, $V_{DDC}$, is generated automatically by the EVR.

### 24.1.2 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.

### 24.1.3 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.

Another important task that is done by SSW is to read the Boot Mode Index (BMI) stored in Flash and decide the startup mode (such as User Productive mode and Bootstrap Loader) selected by the user. For more details about the various startup modes and handling of BMI, please refer to **Section 24.2**.

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.


Subject to Agreement on the Use of Product Information

During SSW execution, the SRAM area between $2000'00C0_H$ and $2000'0200_H$ is reserved for usage by XMC1300 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 **Section 24.1.3.1**.

### 24.1.3.1 Clock system handling by SSW

XMC1300 SSW is able to change device clock settings to allow flexibility of device performance e.g. 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 24-1**), the values from these locations are processed by SSW as follows:

- if CLK_VAL1[31]=0 - CLK_VAL1[19:0] bits are installed into SCU_CLKCR[19:0] register - this configures clock dividers and selection
- if CLK_VAL2[31]=0 - CLK_VAL2[10:0] bits are installed into SCU_CGATCLR0[10:0] - 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 2..32 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 24-1**) 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

### 24.1.4 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


Subject to Agreement on the Use of Product Information

- 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 24-1** 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 Interrpt 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 XMC1300, 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.

### 24.1.5 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 **Section 24.1.3.1**. 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 XMC1300 device supports a suspend capability for peripherals, if the program execution of the CPU is stopped by the debugger, e.g. 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.

## 24.2 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 so called Boot Mode Index (short notation BMI)

### 24.2.1 Start-up modes in XMC1300

Start-up mode selection in XMC1300 is done by the SSW after reset. It based on **Boot Mode Index (BMI)** stored in flash configuration sector 0 (CS0) meaning no device pin is used for configuration purpose. 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 **Section 24.2.3**. The default start-up mode in device delivery state is the ASC BSL mode.

A summary of the supported start-up modes follows.

#### 24.2.1.1 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 $1000'1004_H$.


Subject to Agreement on the Use of Product Information

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).

#### 24.2.1.2 User mode with debug enabled

User code is started from Flash memory, the first address is determined as in **User productive mode** (see **Section 24.2.1.1**).

Debug interface is configured according to **Boot Mode Index (BMI)** value and enabled.

#### 24.2.1.3 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 **Section 24.2.3.2**).

#### 24.2.1.4 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 XMC1300 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 XMC1300.

The functionality of standard BSL routines is described in the BSL chapter.

#### 24.2.1.5 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 (refer to **Section 24.2.2**):


Subject to Agreement on the Use of Product Information

  - 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

Two BSL modes with time-out are supported in XMC1300, both using USIC0 module 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 (refer to **Section 24.2.2**)
  - 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
- 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

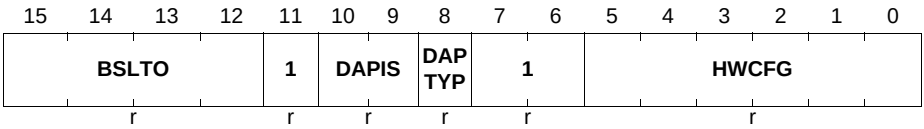
*Note: If BSLTO=0 is configured in BMI - SSW at all does not perform any BSL check but directly takes User Productive Mode.*


Subject to Agreement on the Use of Product Information

### 24.2.2 Boot Mode Index (BMI)

The Boot Mode Index is 2 Byte value stored in Flash (refer to **Table 24-1**) 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**
**Boot Mode Index** **CS0 offset 0E00$_H$** **Delivery state: FFC0$_H$**




| Field | Bits | Typ | Description |
|---|---|---|---|
| **HWCFG** | [5:0] | r | **Start-up mode selection:**<br>$000000_B$ ASC Bootstrap Loader mode (ASC_BSL)<br>$000001_B$ User productive Mode (UPM)<br>$000011_B$ User mode with debug enabled (UMD)<br>$000111_B$ User mode with debug enabled and HAR (UMHAR)<br>$001000_B$ SSC Bootstrap Loader mode (SSC_BSL)<br>$010000_B$ ASC BSL mode with time-out (ASC_BSLTO)<br>$011000_B$ SSC BSL mode with time-out (SSC_BSLTO)<br>$101010_B$ Secure Bootstrap Loader mode (SBSL) 1)<br>else Not defined 2) |
| **DAPTYP** | 8 | r | **DAP Type Selection**<br>Coding as of SCU_DAPCON.DAPTYP |
| **DAPDIS** | [10:9] | r | **SWD/SPD Input/Output Selection**<br>Coding as of SCU_DAPCON.DAPDIS |
| **BSLTO** | [15:12] | r | **ASC BSL Time-out value**<br>The time-out duration is BSLTO*2664000 MCLK cycles, the supported time-out range is 0.3-5s (333...4995ms) |
| **1** | [7:6], 11 | r | **Reserved, must be programmed to 1** |

1) Only in device versions supporting it

2) ATTENTION: Installing one of these values will cause device delivery state to be restored upon the next reset - refer to **Section 24.2.3**


Subject to Agreement on the Use of Product Information

### 24.2.3 Start-up mode selection

The start-up modes in XMC1300 are selected and handled according to Boot Mode Index (BMI) value read from Flash - refer to **Section 24.2.1**.

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 BSL then jumping into SRAM at $2000'0200_H$
- taking Secure Bootstrap Loader (SBSL) mode - if SBSL is selected in BMI and supported for the device
- executing (endless) loop - waiting for valid Start+Header Bytes to be received via ASC upon ASC BSL mode selected 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 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.

#### 24.2.3.1 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 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.


Subject to Agreement on the Use of Product Information

#### 24.2.3.2 Debug system handling

If debug system must be enabled - start-up mode with debug support is selected in Boot Mode Index (refer to **Section 24.2.2**) - 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 (refer to **Section 24.2.2**) 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 24-1**)

## 24.3 Data in Flash for SSW and User SW

**Table 24-1** shows the data in Flash which is relevant for SSW execution and can be used by user software in XMC1300.

**Table 24-1 Flash data for SSW and user SW in XMC1300**

| 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 | --- |


Subject to Agreement on the Use of Product Information

**Table 24-1 Flash data for SSW and user SW in XMC1300**

| Address | Length | Function | Target location |
|---|---|---|---|
| | | 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: | |
| $1000'1010_H$ | 4 B | Clock configuration value CLK_VAL1 | If bit[31]=0: bits[19:0] into SCU_CLKCR[19:0] |
| $1000'1014_H$ | 4 B | Clock gating configuration value CLK_VAL2 | If bit[31]=0: bits[10:0] into SCU_CGATCLR0[10:0] |


Subject to Agreement on the Use of Product Information

# 25 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 (**Section 25.1**) and the SSC BSL (**Section 25.2**) 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.

**Section 25.3** describes 5 user-accessible routines that can be called by application software. These routines are located in the ROM memory.

## 25.1 ASC (UART) Bootstrap Loader

ASC (UART) Standard Bootstrap Loader (ASC BSL) in XMC1300 uses USIC0 module to download code/data into SRAM starting at address 2000'0200H.

After the last code/data Byte is received and stored, the Startup Software starts user code from address 20000'0200H.

### 25.1.1 Pin usage

ASC BSL in XMC1300 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.*

### 25.1.2 ASC BSL execution flow

The complete ASC Bootstrap Loader procedure consists of two parts as described below.

#### 25.1.2.1 ASC BSL entry check sequence

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


Subject to Agreement on the Use of Product Information

*Note: For definitions of header/acknowledge byte values exchanged with the host - refer to **Section 25.1.3**.*

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** (see **Section 25.1.2.2**)
   b) BSL_ENC_F or BSL_ENC_H - the communication during **ASC BSL download sequence** will potentially use different (e.g. higher) baudrate than the initially detected - continue with the **Enhanced ASC BSL entry sequence** (see below))

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.*


Subject to Agreement on the Use of Product Information

### 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 25-1** 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 XMC1300 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:

(25.1)

$$MCLK = \text{Initial Baud Rate} \times (PDIV + 1) \times 8$$

- Select a new baud rate that is supported by the enhanced ASC BSL given the MCLK (see **Table 25-1**)
- Calculate the scaling factor, i.e. ratio of the new baud rate to the initial baud rate. Some examples are given in **Table 25-2**. The scaling factor is rounded to the nearest integer in order to meet a frequency deviation of +/- 3%.


Subject to Agreement on the Use of Product Information

**Table 25-1 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 |

**Table 25-2 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:

(25.2)

$$\text{STEP (in fractional divider mode)} = \frac{1024 \times \text{Scaling Factor}}{\text{PDIV} + 1}$$

- 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


Subject to Agreement on the Use of Product Information

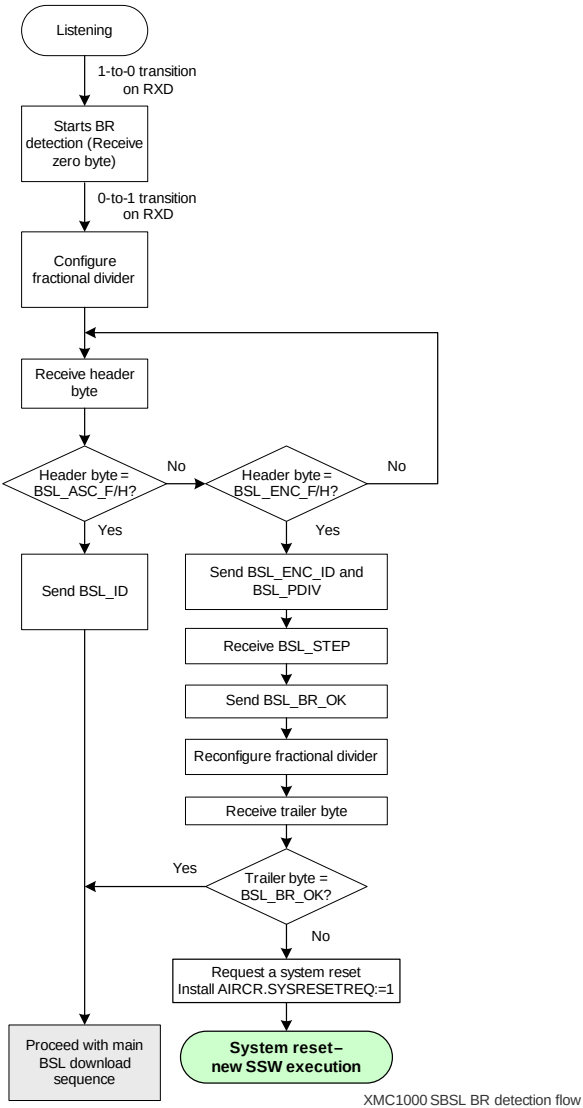



**Figure 25-1 Baud Rate configuration sequence during XMC1300 ASC BSL entry**

### 25.1.2.2 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 25-2**). The least significant byte is received first.

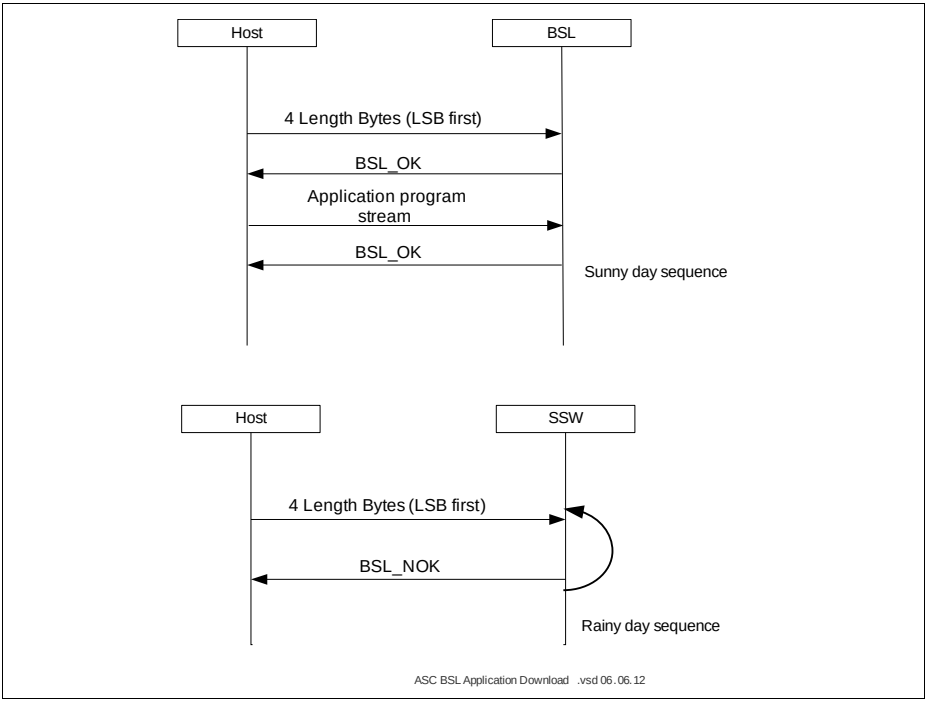



**Figure 25-2 XMC1300 Standard ASC BSL: Application download protocol**

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.

### 25.1.3 ASC BSL protocol data definitions

The interaction between XMC1300 ASC Bootstrap Loader routine and the host implements handshake protocol based on **Figure 25-3** and request/acknowledge/data Bytes defined in **Table 25-3**.

**Table 25-3 Handshake protocol data definitions in XMC1300 ASC BSL**

| Name | Length, Byte | Value | Description |
|---|---|---|---|
| Requests/data/acknowledge sent by the Host: | | | |
| BSL_ASC_F | 1 | 6CH | Header requesting full duplex ASC mode with the current baud rate |
| BSL_ASC_H | 1 | 12H | 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 XMC1300 firmware: | | | |
| BSL_ID | 1 | 5DH | 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 | 01H | Data received is OK |
| BSL_NOK | 1 | 02H | Failure encountered during data reception |


Subject to Agreement on the Use of Product Information

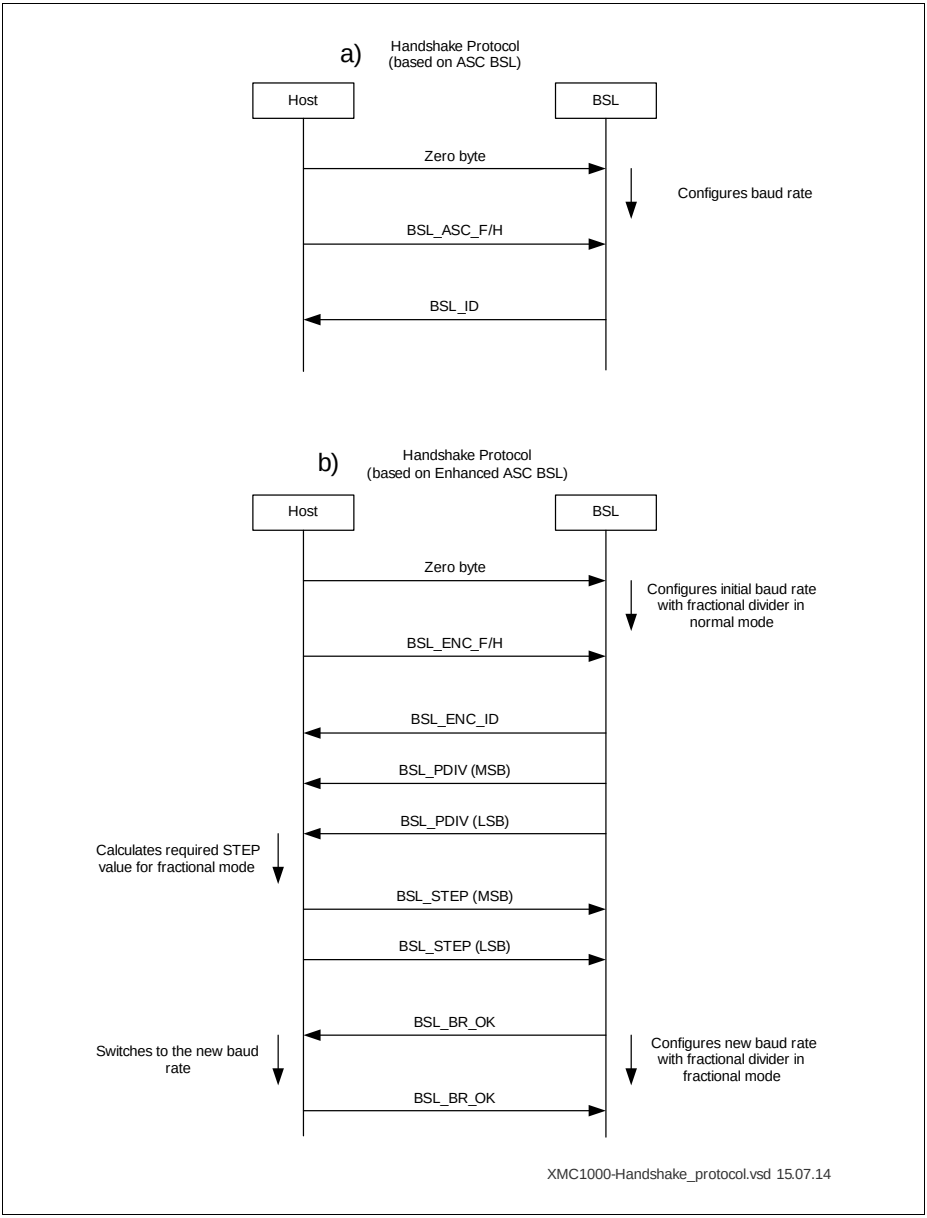



**Figure 25-3 Handshake protocol for XMC1300 ASC BSL entry**

## 25.2 SSC Bootstrap loader

This procedure downloads code from a SPI-compatible serial EEPROM into SRAM starting at address 2000'0200H using Channel 0 of USIC0 Module. The length of the code can be arbitrary up to the user-available SRAM size.

XMC1300 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
- SCLK (clock signal) - P0.14, open drain
- SLS (chip-select CS to the EEPROM) - P0.13, open drain

*Note: Due to the specific functionality of the bootloader - performing one Sequential Read operation only - there is no need of a dedicated SLS-control.*

*Note: Due to XMC1300 pin configured as open drain, external pull-up resistors need to be added on the bus*

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 XMC1300 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 25-4**.

*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 25-4 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 | P11-send | P11-receive | P11-send | P11-receive | P11-send | P11-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 | $00_H$ | dummy | Identification | Address_L | $XX_H$ | Address_M | $XX_H$ |

**Table 25-4 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 | |
|---|---|---|---|---|---|---|---|
| 4 | $00_H$ | dummy | Size | dummy | Identification | Address_L | $XX_H$ |
| 5 | $00_H$ | dummy | Data Byte 1 | dummy | Size, High B | dummy | Identification |
| 6 | $00_H$ | dummy | Data Byte 2 | dummy | Size, Low B | dummy | Size, High B |
| 7 | $00_H$ | dummy | Data Byte 3 | dummy | Data Byte 1 | dummy | Size, Mid B |
| 8 | $00_H$ | dummy | Data Byte 4 | dummy | Data Byte 2 | dummy | Size, Low B |
| 9 … | … | dummy | Data Byte 5 ...n | dummy | Data Byte 3 ...n | dummy | Data Byte 1 ...n |

The EEPROM connected for downloading must contain:

- identification Byte D5H at the first address (0000H)
- 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 0001H
  - in case of an EEPROM with 16-bit addressing - in two Bytes at addresses 0001H/0002H ordered high/low
  - in case of an EEPROM with 24-bit addressing - in three Bytes at addresses 0001H/0002H/0003H 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
   d) pin-configuration as defined above
   e) enable transmission
4. de-select EEPROM by setting CS=1
5. enable EEPROM (CS=0) and send Read command (03h)
6. send two zero Bytes to request data from address 0000h
7. Upon any next Byte transfer as long as the EEPROM is still selected (CS=0) - data from then next following address will be read out.
8. the last Byte received is checked:


Subject to Agreement on the Use of Product Information

a) if equal to the Identification Byte (D5H) - the EEPROM uses 8-bit addressing:
-SSW reads one more Byte and saves it as Code_Length - 1B only effective
-continue with p.10

9. read one more Byte:
   a) if equal to the Identification Byte (D5H) - the EEPROM uses 16-bit addressing:
   -SSW reads two Bytes in order High/Low to determine Code_length (2 Bytes)
   -continue with p.10
10. read one more Byte:
   a) if equal to the Identification Byte (D5H) - 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.10
   b) if Not - directly exit the sequence
11. check the value of Code_length:
   a) if 0 or greater than allowed in XMC1300- directly exit the sequence
12. read [Code_Length] Bytes and store them sequentially from the beginning of SRAM (address 2000'0200H) on
13. disable EEPROM (CS=1)
14. disable the communication channel by resetting USIC0_C0_KSCFG.MODEN:=0
15. set SSW flag BSL_act=1 and exit the sequence

After the last Byte is received and stored, the Startup Software starts the code downloaded from address 20000'0200H.

## 25.3 Firmware routines available for the user

Several user routines (refer to **Table 25-5**) are available inside ROM so they can be called by application software.

**Table 25-5 User routines' in XMC1300 ROM**

| XMC1300 ROM | | Description |
|---|---|---|
| **Address** | **Content** | |
| 0000'0100H | _NvmErase | Pointer to **Erase Flash Page** routine |
| 0000'0104H | _NvmProgVerify | Pointer to **Erase, Program & Verify Flash Page** routine |
| 0000'0108H | _BmiInstallationReq | Pointer to **Request BMI installation** routine |
| 0000'010CH | _CalcTemperature | Pointer to **Calculate chip temperature** routine |
| 0000'0110H | _NvmEraseSector | Pointer to **Erase Flash Sector** routine |


Subject to Agreement on the Use of Product Information

**Table 25-5 User routines' in XMC1300 ROM**

| XMC1300 ROM | | Description |
|---|---|---|
| **Address** | **Content** | |
| 0000'0114H | _NvmProgVerifyBlock | Pointer to **Program & Verify Flash Block** routine |
| 0000'0120H | _CalcTSEVAR | Pointer to **Calculate target level for temperature comparison** routine |

NVM-related functions (see **Section 25.3.1** and **Section 25.3.2**) return status indication as shown in **Table 25-6**.

**Table 25-6 Status indicators returned by NVM routines in XMC1300 ROM**

| Status Indicator | | Description |
|---|---|---|
| **Symbolic name** | **Value** | |
| NVM_PASS | 0001'0000H | Function succeeded |
| NVM_E_FAIL | 8001'0001H | Generic error |
| NVM_E_SRC_ AREA_EXCEEDED | 8001'0003H | Source data is not in RAM |
| NVM_E_SRC_ ALIGNMENT | 8001'0004H | Source data is not 4 Byte aligned |
| NVM_E_NVM_FAIL | 8001'0005H | NVM module can not be physically accessed |
| NVM_E_VERIFY | 8001'0006H | Verification of the written page not successful |
| NVM_E_DST_ AREA_EXCEEDED | 8001'0009H | Destination data is not (completely) located in NVM |
| NVM_E_DST_ ALIGNMENT | 8001'0010H | Destination data is not properly aligned |

The description below provides an overview of the features.

### 25.3.1 Erase Flash Page

XMC1300 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 25-6**)
  - OK (NVM_PASS)


Subject to Agreement on the Use of Product Information

  - 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 XMC1000_NvmErasePage (unsigned long pageAddr)

### 25.3.2 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 25-6**)
  - 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 XMC1000_NvmProgVerify (unsigned long srcAddr, unsigned long dstAddr)

### 25.3.3 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 (0001H)
- Prototype
  unsigned long XMC1000_BmiInstallationReq (unsigned short requestedBmiValue)

### 25.3.4 Calculate chip temperature

This procedure calculates the current chip temperature as measured by the XMC1300 built-in sensor, based on data from Flash including trimming values and pre-calculated constants (refer to **Table 25-7**) and data from the actual measurement (read from Temperature Sensor Counter2 Monitor Register ANATSEMON).

- Input parameter
  - none
- Return status
  - chip temperature in degree Kelvin


Subject to Agreement on the Use of Product Information

- Prototype
  unsigned long XMC1000_CalcTemperature (void)

### 25.3.5 Erase Flash Sector

XMC1300 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 25-6**)
  – 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 XMC1000_NvmEraseSector (unsigned long sectorAddr)

### 25.3.6 Program & Verify Flash Block

XMC1300 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 25-6**)
  – 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 XMC1000_NvmProgVerifyBlock (unsigned long srcAddr, unsigned long dstAddr)

### 25.3.7 Calculate target level for temperature comparison

This procedure - a kind of reverse of **Calculate chip temperature** - calculates the value which must be installed in ANATSEVAR register to get indication in ANATSESTATUS.TSE_CP_VAR_STAT bit when the chip temperature is above/below some target/threshold.

- Input parameter
  – threshold temperature in degree Kelvin - allowed range 223...423 (temperature)
- Return status
  – equivalent sensor threshold value for the temperature provided as input parameter

- Prototype
  unsigned long XMC1000_CalcTSEVAR (unsigned long temperature)

## 25.4 Data in Flash used by the User Routines

Table 25-7 shows the data in Flash which is used by the user routines in XMC1300.

**Table 25-7 Basic Flash data for SSW and user SW in XMC1300**

| 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 | --- |


Subject to Agreement on the Use of Product Information

# Debug System


Subject to Agreement on the Use of Product Information

# 26 Debug System (DBG)

The debug system is an extension to the regular processor architecture. The XMC1300 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] ARMv6-M Architecture Reference Manual

[13] ARM Debug Interface V5

[14] CoreSight Components Technical Reference Manual

[15] CoreSight DAP-Lite

## 26.1 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


Subject to Agreement on the Use of Product Information

configured to halt on a debug event, and a debug event occurs. The supported debug infrastructure can be identified by checking the ROM table.

### 26.1.1 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 **ARMv6-M Architecture Reference Manual** for more detailed informations on the debug functionality.*

### 26.1.2 Block Diagram

The Debug system block diagram is shown in **Figure 26-1**.

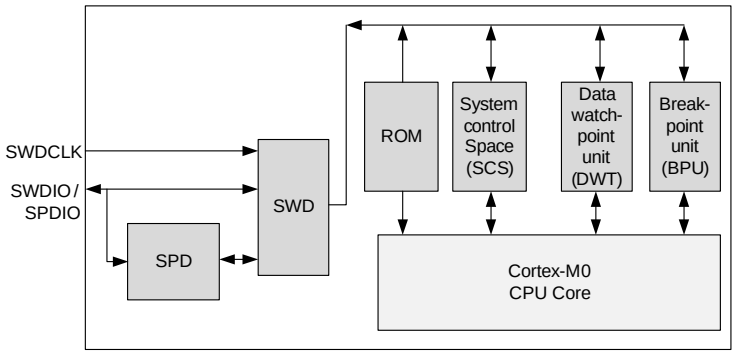



**Figure 26-1 Debug and Trace System block diagram**

## 26.2 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


Subject to Agreement on the Use of Product Information

(SWD) or Single Pin Debug (SPD). The selection of the access protocol (SWD or SPD) is done by BMI mode setting.

### 26.2.1 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.

### 26.2.2 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 XMC1300 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.

### 26.2.3 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


Subject to Agreement on the Use of Product Information

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.

### 26.2.4 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.

### 26.2.5 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.

#### 26.2.5.1 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, continuos 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.


Subject to Agreement on the Use of Product Information

**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.*

### 26.2.5.2 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.


Subject to Agreement on the Use of Product Information

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.

### 26.2.6 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 (**Section 26.2.5.1**).


Subject to Agreement on the Use of Product Information

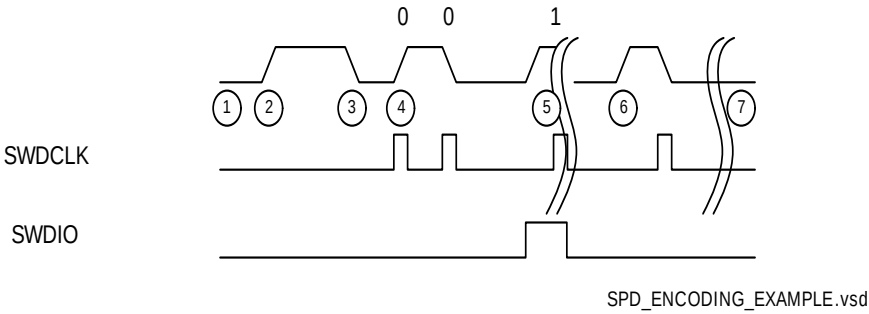



**Figure 26-2 SPD Encoding Example**

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).


Subject to Agreement on the Use of Product Information

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.


Subject to Agreement on the Use of Product Information

### 26.2.7 Debug accesses and Flash protection

The XMC1300 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.

### 26.2.8 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.

#### 26.2.8.1 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


Subject to Agreement on the Use of Product Information

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.

The following Figure (**Figure** HAR - Halt After Reset) shows the software flow based on the modules participating to the HAR.

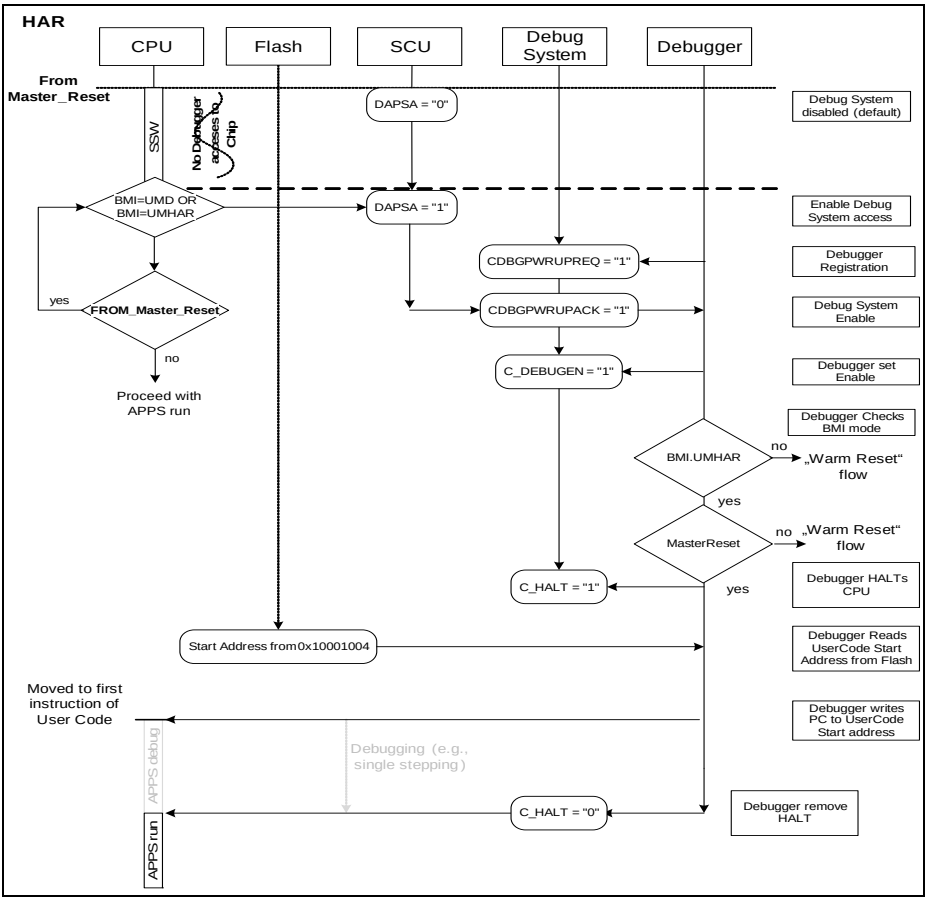



**Figure 26-3 HAR - Halt After Reset Flow**

### 26.2.8.2 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.

The following Figure (**Figure** Hot Plug and Warm Reset) illustrates the debug tool HOT PLUG situation or the Halt after Warm Reset (system reset) and how to proceed to come to a Halt situation.


Subject to Agreement on the Use of Product Information

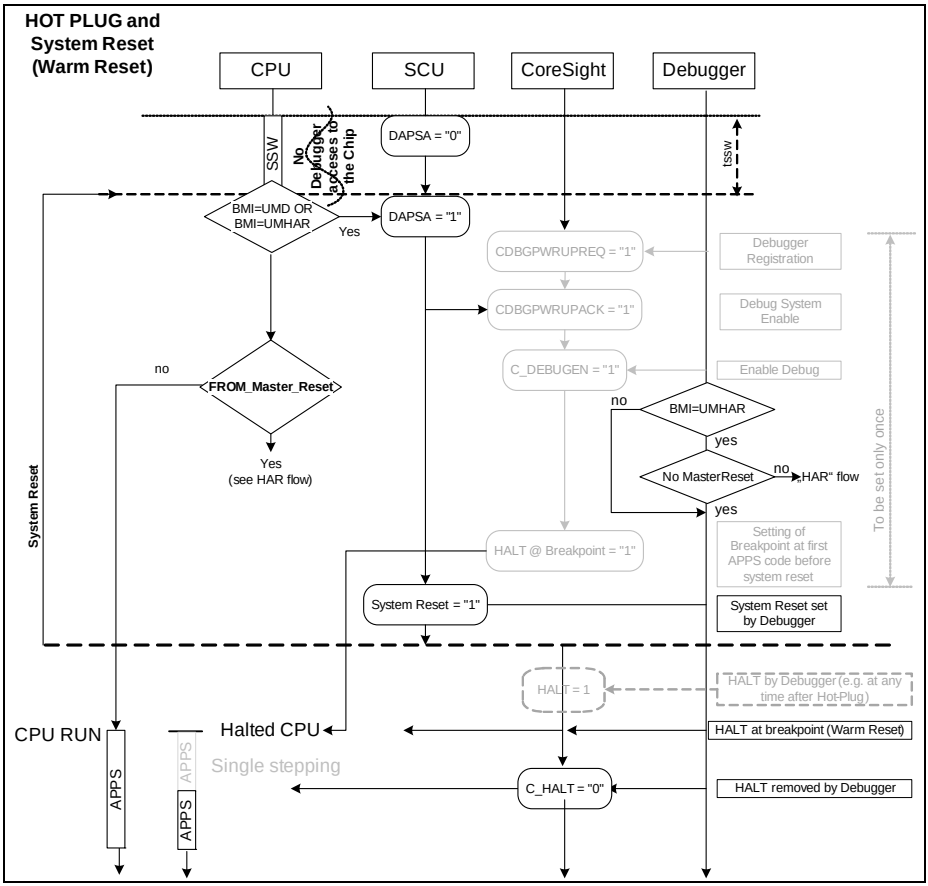



**Figure 26-4 HOT PLUG or Warm Reset Flow**

## 26.2.9 Halting Debug and Peripheral Suspend

The XMC1300 device supports a suspend capability for peripherals, if the program execution of the CPU is stopped by the debugger, e.g. 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, e.g. 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. The following table (**Table 26-1**) 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 26-1 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 |
| VADC | yes | not active | yes | yes |
| MATH | yes | not active | yes | yes |
| PRNG | 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


Subject to Agreement on the Use of Product Information

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.)

### 26.2.10 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.*

### 26.2.11 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 XMC1300 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.*


Subject to Agreement on the Use of Product Information

### 26.2.12 Debug Signals

XMC1300 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 26-2 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.

#### 26.2.12.1 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

### 26.2.13 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.

## 26.3 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.


Subject to Agreement on the Use of Product Information

## 26.4 Service Request Generation

The debugger can set DHCSR.C_MASKINTS to 1 to prevent PendSV, SysTick and external configured interrupts from occurring.

## 26.5 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.

## 26.6 Power, Reset and Clock

The requirements for power, reset and clock are derived from ARM.


Subject to Agreement on the Use of Product Information

### 26.6.1 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.

### 26.6.2 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. .

### 26.6.3 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.

## 26.7 Initialization and System Dependencies

### 26.7.1 ID Code

Available ID Code is used by a debug tool to identify the available debug components during tool setup.


Subject to Agreement on the Use of Product Information

**Table 26-3 ARM CoreSight™ Component ID code**

| ID | Value | Description |
|---|---|---|
| SW_DP | 0BB1 1477$_H$ | The ARM SW-DP ID |

### 26.7.2 ROM Table

To identify Infineon as manufacturer and XMC1300 as device, the ROM table has to be read out. The format of the ROM table is illustrated in **Table 26-4**.

**Table 26-4 Peripheral ID Values of XMC1300 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]<br>[3:0] | JEP106 ID code [3:0]<br>Part Number [11:8] | DBGROMID [4:1]<br>DBGROMID [23:20] |
| PID2 | FE8$_H$ | XXXX1100$_B$ | [7:4]<br>[3]<br>[2:0] | Revision<br>JEDEC assigned ID fields<br>JEP106 ID code [6:4] | DBGROMID [31:28]<br>‘1’<br>DBGROMID [7:5] |
| PID3 | FEC$_H$ | 00000000$_B$ | [7:4]<br>[3:0] | RevAnd, minor revision field<br>Customer-modified block (if non-zero) | |
| PID4 | FD0$_H$ | 00000000$_B$ | [7:4]<br>[3:0] | 4KB count<br>JEP106 continuation code | 4KB count<br>DBGROMID [11:8] |

1) For “X” values please refer to Data Sheet

The ROM table values representing Infineon Part Number, JEP106 and Revision number can be checked in the SCU chapter.


Subject to Agreement on the Use of Product Information

## 26.8 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 26-5 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 26-6 Register Overview**

| Short Name | Description | Offset Addr. | Access Mode | | Description See |
|---|---|---|---|---|---|
| | | | Read | Write | |
| SCS_DFSR | Debug Fault Status Register | D30$_H$ | | | Page 26-20 |
| SCS_DHCSR | Debug Halting Control and Status Register | DF0$_H$ | | | Page 26-22 |
| SCS_DCRSR | Debug Core Register Selector Register | DF4$_H$ | | | Page 26-28 |
| SCS_DCRDR | Debug Core Register Data Register | DF8$_H$ | | | Page 26-29 |
| SCS_DEMCR | Debug Exception and Monitor Control Register | DFC$_H$ | | | Page 26-30 |
| DWT_CTRL | DWT Control Register | 000$_H$ | | | Page 26-31 |
| DWT_PCSR | DWT program Counter Sample Register | 01C$_H$ | | | Page 26-32 |
| DWT_COMP0 | DWT Comparator register | 020$_H$ | | | Page 26-32 |
| DWT_COMP1 | DWT Comparator register | 030$_H$ | | | Page 26-32 |
| DWT_MASK0 | DWT MASK register | 024$_H$ | | | Page 26-33 |
| DWT_MASK1 | DWT MASK register | 034$_H$ | | | Page 26-33 |


Subject to Agreement on the Use of Product Information

**Table 26-6 Register Overview** (cont'd)

| Short Name | Description | Offset Addr. | Access Mode | | Description See |
|---|---|---|---|---|---|
| | | | Read | Write | |
| DWT_FUNCTION0 | DWT Comparator Function register | $028_H$ | | | Page 26-34 |
| DWT_FUNCTION1 | DWT Comparator Function register | $038_H$ | | | Page 26-34 |
| BP_CTRL | Breakpoint Control register | $000_H$ | | | Page 26-35 |
| BP_COMP0 | Breakpoint Comparator register | $008_H$ | | | Page 26-36 |
| BP_COMP1 | Breakpoint Comparator register | 008C | | | Page 26-36 |
| BP_COMP2 | Breakpoint Comparator register | $010_H$ | | | Page 26-36 |
| BP_COMP3 | Breakpoint Comparator register | $014_H$ | | | Page 26-36 |

## 26.8.1 DFSR - Debug Fault Status Register

### 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**
**Debug Fault Status Register ($D30_H$) Reset Value: 0000 0000$_H$**

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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | | | | EXTERNAL | VCATCH | DWTTRAP | BKPT | HALTED |
| r | | | | | | | | | | | rw | rw | rw | rw | rw |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| HALTED | 0 | rw | **HALTED**<br>Indicates a debug event generated by a C_HALT or C_STEP request, triggered by a write to the DHCSR.<br>$0_B$ no active halt request debug event.<br>$1_B$ halt request debug event active. |
| BKPT | 1 | rw | **BKPT**<br>Indicates a debug event generated by BKPT instruction execution or a breakpoint match in the BPU.<br>$0_B$ no breakpoint debug event.<br>$1_B$ at least one breakpoint debug event. |
| DWTTRAP | 2 | rw | **DWTTRAP**<br>Indicates a debug event generated by the DWT.<br>$0_B$ no debug events generated by the DWT.<br>$1_B$ at least one debug event generated by the DWT. |
| VCATCH | 3 | rw | **VCATCH**<br>Indicates whether a vector catch debug event was generated.<br>$0_B$ no vector catch debug event generated.<br>$1_B$ vector catch debug event generated.<br>*Note: The corresponding FSR shows the primary cause of the exception.* |
| EXTERNAL | 4 | rw | **EXTERNAL**<br>Indicates an asynchronous debug event generated because of EDBGRQ being asserted.<br>$0_B$ no EDBGRQ debug event.<br>$1_B$ EDBGRQ debug event. |
| **0** | [31:5] | r | **Reserved** |

## 26.8.2 DHCSR - Debug Halting Control and Status Register

**SCS_DHCSR**

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.


Subject to Agreement on the Use of Product Information

A separate register is represented below for read and write as the function changes at some bits access based.

**SCS_DHCSR**

**Debug Halting Control and Status Register [Read Mode] ($DF0_H$)** **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_LOCKUP | 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**<br>$0_B$ Halting debug disabled.<br>$1_B$ Halting debug enabled.<br>*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 | r | **Processor halt bit.**<br>The effects of writes to this bit are:<br>$0_B$ Request a halted processor to run.<br>$1_B$ Request a running processor to halt.<br>*Note: This bit is unknown after power-on reset* |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **C_STEP** | 2 | r | **Processor step bit.**<br>The effects of writes to this bit are:<br>$0_B$ Single-stepping disabled.<br>$1_B$ Single-stepping enabled.<br>*Note: This bit is unknown after power-on reset* |
| **C_MASKINTS** | 3 | r | **Mask PEDSV, SysTick and external configurable interrupts.**<br>The effects of writes to this bit are:<br>$0_B$ Do not mask<br>$1_B$ Mask PendSV, SysTick and external configurable interrupts.<br>The effect of any attempt to change the value of this bit is UNPREDICTABLE unless both:<br>• before the write to DHCSR, the value of the C_HALT bit is 1<br>• the write to the DHCSR that changes the C_MASKINTS bit also writes 1 to the C_HALT bit.<br>This means that a single write to DHCSR cannot set the C_HALT to 0 and change the value of the C_MASKINTS bit.<br>When DHCSR.C_DEBUGEN is set to 0, the value of this bit is UNKNOWN.<br>This bit is UNKNOWN after Power-on reset. |
| **S_REGRDY** | 16 | r | **S_REGRDY status - a handshake flag for transfers through DCRDR**<br>How to work with:<br>• Writing to DCRSR clears the bit to 0.<br>• Completion of the DCRDR transfer then sets the bit to 1<br>Check DCRDR for more information.<br>$0_B$ There has been a wrote to the DCRDR, but the transfer is not complete.<br>$1_B$ The transfer to or from the DCRDR is complete.<br>*Note: This bit is only valid when the processor is in Debug State, otherwise the bit is UNKONWN.* |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **S_HALT** | 17 | r | **S_HALT indicates whether the processor is in Debug state.**<br>$0_B$ Not in Debug State.<br>$1_B$ In Debug State. |
| **S_SLEEP** | 18 | r | **S_SLEEP indicates whether the processor is sleeping.**<br>$0_B$ Not sleeping.<br>$1_B$ Sleeping.<br>*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.**<br>$0_B$ Not locked up.<br>$1_B$ Locked up.<br>*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.* |

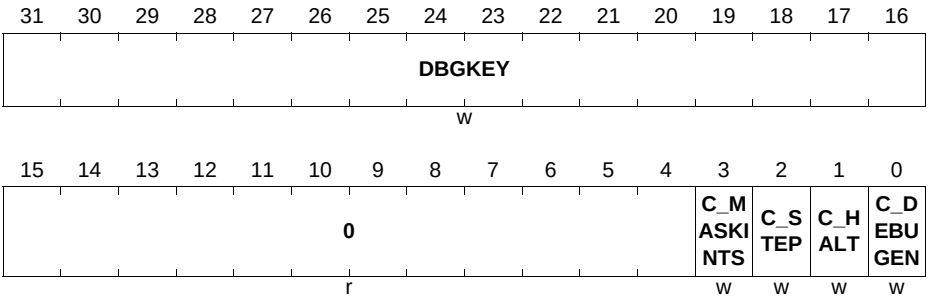

Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **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:**<br>$0_B$ No instruction has completed since the last DHCSR read.<br>$1_B$ At least one instructions has completed since last DHCSR read.<br>This is a sticky bit, that clears to 0 on a read of DHCSR.<br>This bit is UNKNOWN:<br>• after a Local reset, but is set to 1 as soon as the processor completes execution of an instruction<br>• when S_LOCKUP is set to 1<br>• when S_HALT is set to 1<br>*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.**<br>$0_B$ No reset since last DHCSR read.<br>$1_B$ At least one rest since last DHCSR read.<br>*Note: This is a sticky bit, that clears to 0 on a read of DHCSR.* |
| **0** | [15:4], [23:20], [31:26] | r | **Reserved** |


Subject to Agreement on the Use of Product Information

**SCS_DHCSR**
**Debug Halting Control and Status Register [Write Mode]**

**($DF0_H$)** **Reset Value: $0000\ 0000_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **C_DEBUGEN** | 0 | w | **Halting debug enable bit**<br>$0_B$ Halting debug disabled.<br>$1_B$ Halting debug enabled.<br>*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.**<br>The effects of writes to this bit are:<br>$0_B$ Request a halted processor to run.<br>$1_B$ Request a running processor to halt.<br>*Note: This bit is unknown after power-on reset* |
| **C_STEP** | 2 | w | **Processor step bit.**<br>The effects of writes to this bit are:<br>$0_B$ Single-stepping disabled.<br>$1_B$ Single-stepping enabled.<br>*Note: This bit is unknown after power-on reset* |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **C_MASKINTS** | 3 | w | **Mask PEDSV, SysTick and external configurable interrupts.**<br>The effects of writes to this bit are:<br>$0_B$ Do not mask<br>$1_B$ Mask PendSV, SysTick and external configurable interrupts.<br>The effect of any attempt to change the value of this bit is UNPREDICTABLE unless both:<br>• before the write to DHCSR, the value of the C_HALT bit is 1<br>• the write to the DHCSR that changes the C_MASKINTS bit also writes 1 to the C_HALT bit.<br>This means that a single write to DHCSR cannot set the C_HALT to 0 and change the value of the C_MASKINTS bit.<br>When DHCSR.C_DEBUGEN is set to 0, the value of this bit is UNKNOWN.<br>This bit is UNKNOWN after Power-on reset. |
| **DBGKEY** | [31:16] | w | **DEBUG Key Bits [31:16]!!!**<br>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** |

### 26.8.3 DCRSR - Debug Core Register Selector Register

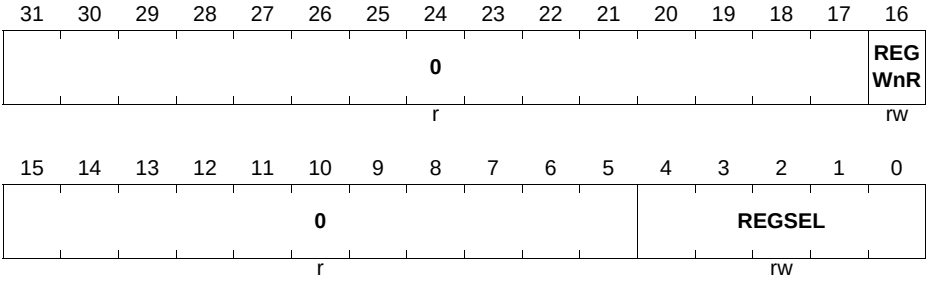
**SCS_DCRSR**

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.


Subject to Agreement on the Use of Product Information

**SCS_DCRSR**

**Debug Core Register Selector Register ($DF4_H$)** **Reset Value: $0000\ 0000_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **REGSEL** | [4:0] | rw | **REGSEL - Specifies the ARM core register or special-purpose register to transfer**<br>$00000_B$- $01100_B$ ARM core register R0-R12. For example, 0b00000 specifies R0 and 0b00101 specifies R5<br>$01101_B$ The current SP. See also values 0b10001 and 0b10010.<br>$01110_B$ LR.<br>$01111_B$ Debug Return Address.<br>$10000_B$ xPSR.<br>$10001_B$ Main stack pointer, MSP.<br>$10010_B$ Process stack pointer, PSP.<br>$10100_B$ Bits[31:24] Control; Bits[23:8] Reserved; Bits[7:0] PRIMASK.<br>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**<br>$0_B$ read.<br>$1_B$ write. |
| **0** | [31:17], [15:5] | r | **Reserved** |

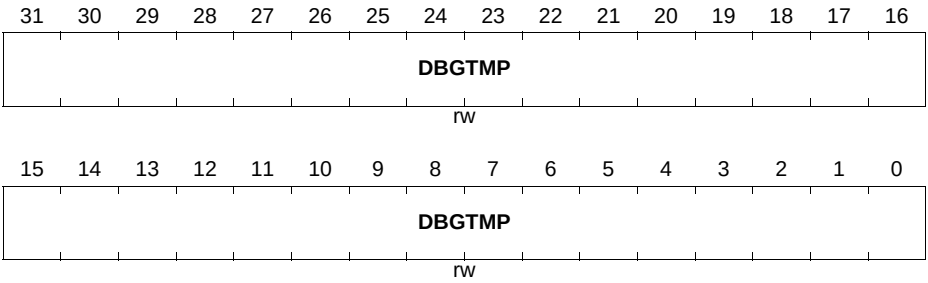

Subject to Agreement on the Use of Product Information

## 26.8.4 DCRDR - Debug Core Register Data Register

### 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**
**Debug Core Register Data Register (DF8$_H$)** **Reset Value: xxxx xxxx$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **DBGTMP** | [31:0] | rw | **DBGTMP - Data temporary cache, for reading and writing register.**<br>This register is UNKONWN:<br>• on reset<br>• when DHCSR.S_HALT = 0<br>• when DHCSR.S_REGRDY = 0 during execution of a DCRSR based transaction that updates the register. |

## 26.8.5 DEMCR - Debug Exception and Monitor Control Register

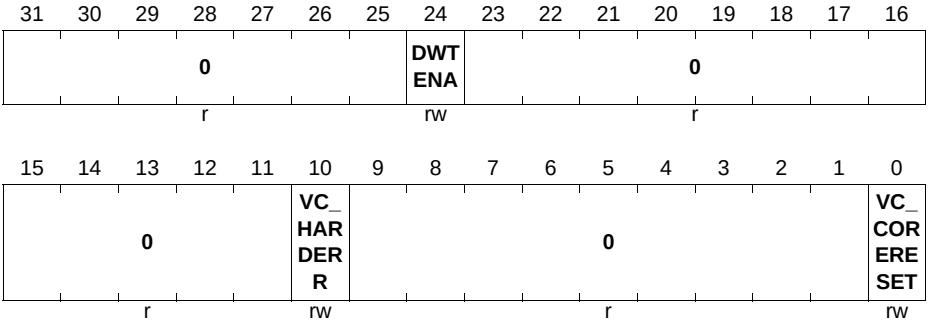
### SCS_DEMCR

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**

**Debug Exception and Monitor Control Register ($DFC_H$) Reset Value: 0000 0000$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **VC_CORERESET** | 0 | rw | **VC_CORERESET - Enable Reset Vector Catch. This causes a Local reset to handle a running system**<br>$0_B$ Reset Vector Catch disabled.<br>$1_B$ Reset Vector Catch enabled.<br>*Note: If DHCSR.C_DEBUGEN is set to 0, the processor ignores the value of this bit.* |
| **VC_HARDERR** | 10 | rw | **VC_CORERESET - Enable Reset Vector Catch. This causes a Local reset to halt a running system.**<br>$0_B$ halting debug trap disabled.<br>$1_B$ hating debug trap enabled.<br>*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.**<br>$0_B$ DWT disabled.<br>$1_B$ DWT enabled.<br>*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.* |

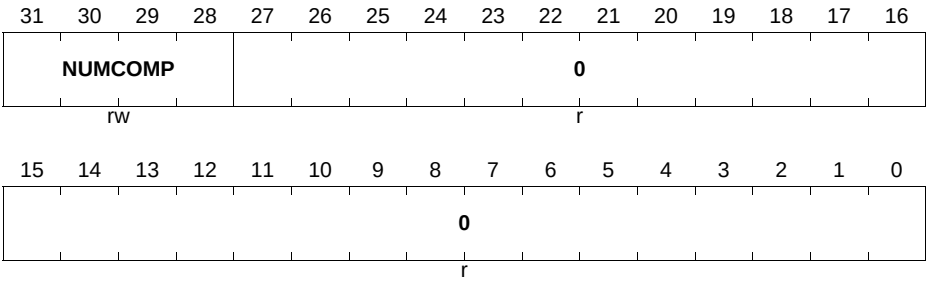
| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | [31:25], [23:16], [15:11], [9:1] | r | **Reserved** |

## 26.8.6 DWT_CTRL - Data Watchpoint Control Register

**DWT_CTRL**

The DWT_CTRL register defines the number of comparators implemented.

**DWT_CTRL**
**Debug Halting Control and Status Register ($000_H$)** **Reset Value: $0000\ 0000_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **NUMCOMP** | [31:28] | rw | **Number of comparators available**<br>$0000_B$No comparator support. |
| **0** | [27:0] | r | **Reserved** |

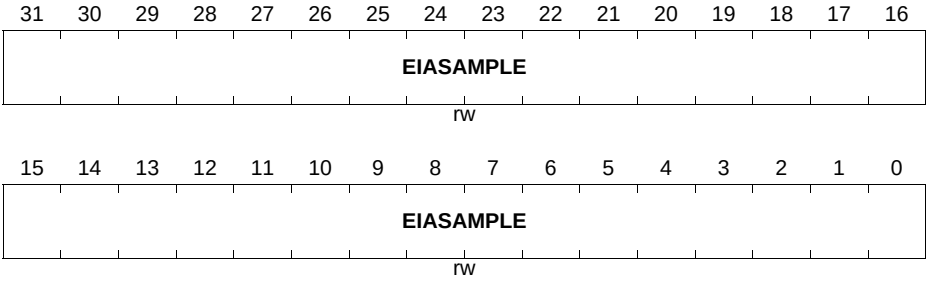
## 26.8.7 DWT_PCSR - Program Counter Sample Register

**DWT_PCSR**

The DWT_PCSR register samples the current value of the program counter. The register is UNKNOW on reset.


Subject to Agreement on the Use of Product Information

**DWT_PCSR**
**Program Counter Sample Register (01C$_H$)** **Reset Value: xxxx xxxx$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **EIASAMPLE** | [31:0] | rw | **EIASAMPLE**<br>Executed instruction address sample register |

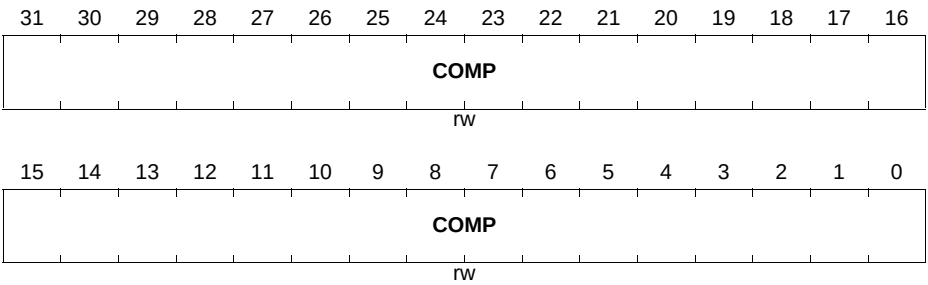
## 26.8.8 DWT_COMPx - DWT Comparator register

**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**
**DWT Comparator register 0 (020$_H$)** **Reset Value: xxxx xxxx$_H$**
**DWT_COMP1**
**DWT Comparator register 1 (030$_H$)** **Reset Value: xxxx xxxx$_H$**





Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **COMP** | [31:0] | rw | **COMP**<br>Reference value for comparison |

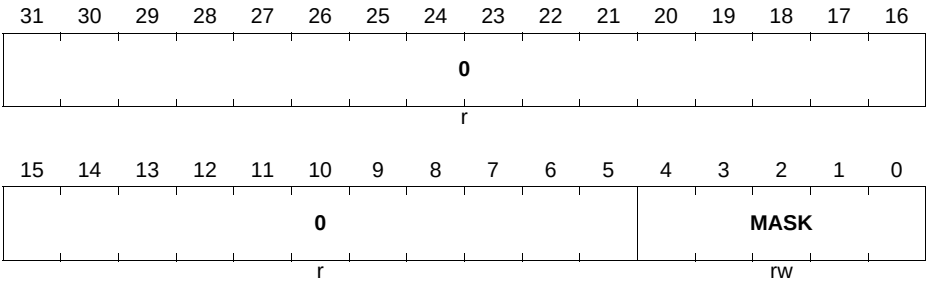
### 26.8.9 DWT_MASKx - DWT Comparator Mask Register

**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**
**Debug Halting Control and Status Register ($24_H$)** **Reset Value: 0000 0000$_H$**
**DWT_MASK1**
**Debug Halting Control and Status Register ($34_H$)** **Reset Value: 0000 0000$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **MASK** | [4:0] | rw | **MASK**<br>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** |


Subject to Agreement on the Use of Product Information

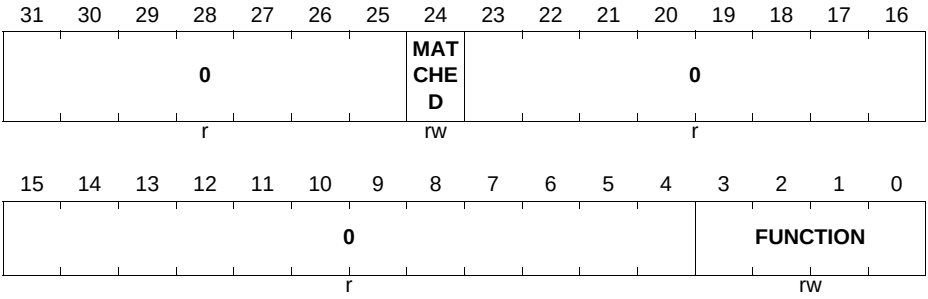
## 26.8.10 DWT_FUNCTIONx - Comparator Function Register

**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**
**Comparator Function Register** **($28_H$)** **Reset Value: 0000 0000$_H$**
**DWT_FUNCTION1**
**Comparator Function Register** **($38_H$)** **Reset Value: 0000 0000$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **FUNCTION** | [3:0] | rw | **FUNCTION**<br>Select action on comparator match-<br>$0000_B$Disabled.<br>$0001_B$Reserved.<br>$0010_B$Reserved.<br>$0011_B$Reserved.<br>$0100_B$PC watchpoint event - input Iaddr.<br>$0101_B$Watchpoint event - input Daddr (read only)<br>$0110_B$Watchpoint event - input Daddr (write only)<br>$0111_B$Watchpoint event - input Daddr (read/write)<br>$1xxx_B$ reserved<br>*Note: This field is set to 0 on a Power-on reset.* |

| Field | Bits | Type | Description |
|---|---|---|---|
| **MATCHED** | 24 | rw | **MATCHED**<br>Comparator match. It indicates that the operation defined by FUNCTION has occurred since the bit was last read.<br>$0_B$ the associated comparator has matched.<br>$1_B$ the associated comparator has not matched.<br>*Note: Reading the register clears this bit to 0.* |
| **0** | [31:25], [23:16], [15:4] | r | **Reserved** |

## 26.8.11 BP_CTRL - Breakpoint Control Register

**BP_CTRL**

The Breakpoint Control Register provides BPU implementation information and the global enable for the BPU.

**BP_CTRL**
**Breakpoint Control Register** **($000_H$)** **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**<br>$0_B$ BPU is disabled.<br>$1_B$ BPU is enabled.<br>*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.**<br>If written as zero, the write to the register is ignored. |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **NUM_CODE** | [7:4] | r | **NUM_CODE, the number of breakpoint comparators.** |
| **0** | [31:8], [3:2] | r | **Reserved** |

## 26.8.12 Breakpoint Comparator Registers

**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**
**Breakpoint Comparator X Register ($008_H$) Reset Value: 0000 0000$_H$**
**BP_COMP1**
**Breakpoint Comparator X Register ($00C_H$) Reset Value: 0000 0000$_H$**
**BP_COMP2**
**Breakpoint Comparator X Register ($010_H$) Reset Value: 0000 0000$_H$**
**BP_COMP3**
**Breakpoint Comparator X Register ($014_H$) Reset Value: 0000 0000$_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BP_MATCH | | 0 | COMP | | | | | | | | | | | | |
| rw | | r | rw | | | | | | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| COMP | | | | | | | | | | | | | | 0 | ENABLE |
| rw | | | | | | | | | | | | | | r | rw |


Subject to Agreement on the Use of Product Information

| Field | Bits | Type | Description |
|---|---|---|---|
| **ENABLE** | 0 | rw | **ENABLE the comparator**<br>$0_B$ Comparator is disabled.<br>$1_B$ Comparator is enabled.<br>*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.**<br>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.<br>*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.**<br>$00_B$ no breakpoint matching.<br>$01_B$ breakpoint on lower halfword, upper is unaffected.<br>$10_B$ breakpoint on upper halfword, lower is unaffected.<br>$11_B$ breakpoint on both lower and upper halfwords.<br>*Note: The field is UNKNOWN on reset.* |
| **0** | 29, 1 | r | **Reserved** |


Subject to Agreement on the Use of Product Information

# Lists of Figures and Tables


Subject to Agreement on the Use of Product Information

# List of Figures

Figure 1-1 XMC1300 Functional diagram . . . 1-3
Figure 2-1 Cortex-M0 Block Diagram . . . 2-3
Figure 2-2 Core registers . . . 2-5
Figure 2-3 Memory map . . . 2-18
Figure 2-4 Memory system ordering . . . 2-20
Figure 2-5 Little-endian format (Example) . . . 2-22
Figure 2-6 Vector table . . . 2-29
Figure 2-7 Exception stack frame . . . 2-33
Figure 4-1 Block Diagram on Service Request Processing . . . 4-2
Figure 4-2 Example for Service Request Distribution . . . 4-3
Figure 5-1 Interrupt Priority Register . . . 5-5
Figure 5-2 Typical XMC1300 Interrupt Latency . . . 5-6
Figure 5-3 Typical Module Interrupt Structure . . . 5-7
Figure 6-1 Event Request Unit Overview . . . 6-1
Figure 6-2 Event Request Select Unit Overview . . . 6-2
Figure 6-3 Event Trigger Logic Overview . . . 6-3
Figure 6-4 ERU Cross Connect Matrix . . . 6-5
Figure 6-5 Output Gating Unit for Output Channel y . . . 6-6
Figure 6-6 ERU Interconnects Overview . . . 6-17
Figure 7-1 MATH Coprocessor Block Diagram . . . 7-1
Figure 7-2 Timing Diagram for a Division Operation . . . 7-3
Figure 7-3 Timing Diagram with Divide by Zero Error . . . 7-5
Figure 7-4 CORDIC Block Diagram . . . 7-9
Figure 7-5 Division Operation with CORDXRC = $01_B$ . . . 7-23
Figure 7-6 Interrupt Structure . . . 7-25
Figure 8-1 XMC1300 Address Space . . . 8-2
Figure 9-1 Logical structure of the NVM module . . . 9-2
Figure 9-2 Decomposition of Logical Address . . . 9-4
Figure 9-3 Structure of a Page . . . 9-4
Figure 9-4 State Diagram of the NVM Module Timings are preliminary . . . 9-25
Figure 11-1 Watchdog Timer Block Diagram . . . 11-2
Figure 11-2 Reset without pre-warning . . . 11-3
Figure 11-3 Reset after pre-warning . . . 11-4
Figure 11-4 Reset upon servicing in a wrong window . . . 11-5
Figure 11-5 Reset upon servicing with a wrong magic word . . . 11-5
Figure 12-1 Real Time Clock Block Diagram Structure . . . 12-2
Figure 12-2 Block Diagram of RTC Time Counter . . . 12-3
Figure 12-3 RTC Clock selection . . . 12-5
Figure 13-1 SCU Block Diagram . . . 13-2
Figure 13-2 Service Request Handling . . . 13-4
Figure 13-3 System States Diagram . . . 13-8


Subject to Agreement on the Use of Product Information

## List of Figures

Figure 13-4 Clock System Block Diagram . . . 13-16
Figure 15-1 USIC Module/Channel Structure . . . 15-5
Figure 15-2 Baud Rate Generator . . . 15-10
Figure 15-3 Principle of Data Buffering . . . 15-11
Figure 15-4 Data Access Structure without additional Data Buffer . . . 15-12
Figure 15-5 Data Access Structure with FIFO . . . 15-14
Figure 15-6 General Event and Interrupt Structure . . . 15-17
Figure 15-7 Transmit Events and Interrupts . . . 15-19
Figure 15-8 Receive Events and Interrupts . . . 15-20
Figure 15-9 Baud Rate Generator Event and Interrupt . . . 15-21
Figure 15-10 Input Conditioning for DX0 and DX[5:3] . . . 15-23
Figure 15-11 Input Conditioning for DX[2:1] . . . 15-24
Figure 15-12 Delay Compensation Enable in DX1 . . . 15-25
Figure 15-13 Divider Mode Counter . . . 15-28
Figure 15-14 Protocol-Related Counter (Capture Mode) . . . 15-29
Figure 15-15 Time Quanta Counter . . . 15-29
Figure 15-16 Master Clock Output Configuration . . . 15-30
Figure 15-17 Transmit Data Path . . . 15-32
Figure 15-18 Transmit Data Validation . . . 15-35
Figure 15-19 Receive Data Path . . . 15-37
Figure 15-20 FIFO Buffer Overview . . . 15-39
Figure 15-21 FIFO Buffer Partitioning . . . 15-41
Figure 15-22 Standard Transmit Buffer Event Examples . . . 15-43
Figure 15-23 Transmit Buffer Events . . . 15-44
Figure 15-24 Standard Receive Buffer Event Examples . . . 15-47
Figure 15-25 Receiver Buffer Events in Filling Level Mode . . . 15-48
Figure 15-26 Receiver Buffer Events in RCI Mode . . . 15-49
Figure 15-27 Bypass Data Validation . . . 15-51
Figure 15-28 TCI Handling with FIFO / Bypass . . . 15-53
Figure 15-29 ASC Signal Connections for Full-Duplex Communication . . . 15-54
Figure 15-30 ASC Signal Connections for Half-Duplex Communication . . . 15-55
Figure 15-31 Standard ASC Frame Format . . . 15-56
Figure 15-32 ASC Bit Timing . . . 15-59
Figure 15-33 Transmitter Pulse Length Control . . . 15-61
Figure 15-34 Pulse Output Example . . . 15-62
Figure 15-35 SSC Signals for Standard Full-Duplex Communication . . . 15-74
Figure 15-36 4-Wire SSC Standard Communication Signals . . . 15-76
Figure 15-37 SSC Data Signals . . . 15-76
Figure 15-38 SSC Shift Clock Signals . . . 15-77
Figure 15-39 SCLKOUT Configuration in SSC Master Mode . . . 15-79
Figure 15-40 SLPHSEL Configuration in SSC Slave Mode . . . 15-80
Figure 15-41 SSC Slave Select Signals . . . 15-81
Figure 15-42 Data Frames without/with Parity . . . 15-84


Subject to Agreement on the Use of Product Information

## List of Figures

Figure 15-43 Quad-SSC Example . . . 15-87
Figure 15-44 Connections for Quad-SSC Example . . . 15-88
Figure 15-45 MSLS Generation in SSC Master Mode . . . 15-90
Figure 15-46 SSC Closed-loop Delay . . . 15-104
Figure 15-47 SSC Closed-loop Delay Timing Waveform . . . 15-106
Figure 15-48 SSC Master Mode with Delay Compensation . . . 15-107
Figure 15-49 SSC Complete Closed-loop Delay Compensation . . . 15-108
Figure 15-50 IIC Signal Connections . . . 15-110
Figure 15-51 IIC Frame Example (simplified) . . . 15-111
Figure 15-52 Start Symbol Timing . . . 15-119
Figure 15-53 Repeated Start Symbol Timing . . . 15-119
Figure 15-54 Stop Symbol Timing . . . 15-120
Figure 15-55 Data Bit Symbol . . . 15-120
Figure 15-56 IIC Master Transmission . . . 15-125
Figure 15-57 Interrupt Events on Data Transfers . . . 15-128
Figure 15-58 IIS Signals . . . 15-136
Figure 15-59 Protocol Overview . . . 15-137
Figure 15-60 Transfer Delay for IIS . . . 15-137
Figure 15-61 Connection of External Audio Devices . . . 15-138
Figure 15-62 Transfer Delay with Delay 1 . . . 15-140
Figure 15-63 No Transfer Delay . . . 15-141
Figure 15-64 MCLK and SCLK for IIS . . . 15-145
Figure 15-65 USIC Module and Channel Registers . . . 15-154
Figure 15-66 USIC Module Structure in XMC1300 . . . 15-226
Figure 15-67 USIC Channel I/O Lines . . . 15-227
Figure 16-1 ADC Structure Overview . . . 16-3
Figure 16-2 ADC Kernel Block Diagram . . . 16-4
Figure 16-3 ADC Cluster Structure . . . 16-5
Figure 16-4 Conversion Request Unit . . . 16-6
Figure 16-5 Signal Path Model . . . 16-9
Figure 16-6 Clock Signal Summary . . . 16-12
Figure 16-7 Reference Voltage Sources . . . 16-17
Figure 16-8 Reference Voltage Ranges . . . 16-17
Figure 16-9 Queued Request Source . . . 16-21
Figure 16-10 Interrupt Generation of a Queued Request Source . . . 16-24
Figure 16-11 Scan Request Source . . . 16-25
Figure 16-12 Arbitration Round with 4 Arbitration Slots . . . 16-28
Figure 16-13 Conversion Start Modes . . . 16-31
Figure 16-14 Alias Feature . . . 16-34
Figure 16-15 Result Monitoring through Limit Checking . . . 16-35
Figure 16-16 Result Monitoring through Compare with Hysteresis . . . 16-37
Figure 16-17 Boundary Flag Switching . . . 16-38
Figure 16-18 Boundary Flag Control . . . 16-39


Subject to Agreement on the Use of Product Information

## List of Figures

Figure 16-19 Conversion Result Storage . . . . . 16-45
Figure 16-20 Result Storage Options . . . . . 16-46
Figure 16-21 Result FIFO Buffers . . . . . 16-48
Figure 16-22 Standard Data Reduction Filter . . . . . 16-51
Figure 16-23 Standard Data Reduction Filter with FIFO Enabled . . . . . 16-52
Figure 16-24 FIR Filter Structure . . . . . 16-53
Figure 16-25 IIR Filter Structure . . . . . 16-54
Figure 16-26 Result Difference . . . . . 16-55
Figure 16-27 Parallel Conversions . . . . . 16-57
Figure 16-28 Synchronization via ANON and Ready Signals . . . . . 16-58
Figure 16-29 Timer Mode for Equidistant Sampling . . . . . 16-59
Figure 16-30 Broken Wire Detection . . . . . 16-60
Figure 16-31 External Analog Multiplexer Example . . . . . 16-62
Figure 17-1 Block diagram of Analog Comparator . . . . . 17-2
Figure 17-2 Analog Comparator Reference Divider Function . . . . . 17-3
Figure 17-3 Out of range comparator . . . . . 17-4
Figure 19-1 CCU4 block diagram . . . . . 19-5
Figure 19-2 CCU4 slice block diagram . . . . . 19-6
Figure 19-3 Slice input selector diagram . . . . . 19-9
Figure 19-4 Slice connection matrix diagram . . . . . 19-11
Figure 19-5 Timer start/stop control diagram . . . . . 19-12
Figure 19-6 Starting multiple timers synchronously . . . . . 19-13
Figure 19-7 CC4y Status Bit . . . . . 19-14
Figure 19-8 Shadow registers overview . . . . . 19-16
Figure 19-9 Shadow transfer enable logic . . . . . 19-17
Figure 19-10 Shadow transfer timing example - center aligned mode . . . . . 19-18
Figure 19-11 Usage of the CCU4x.MCSS input . . . . . 19-19
Figure 19-12 Edge aligned mode, **CC4yTC**.TCM = $0_B$ . . . . . 19-20
Figure 19-13 Center aligned mode, **CC4yTC**.TCM = $1_B$ . . . . . 19-21
Figure 19-14 Single shot edge aligned - **CC4yTC**.TSSM = $1_B$, **CC4yTC**.TCM = $0_B$ . . . . 19-21
Figure 19-15 Single shot center aligned - **CC4yTC**.TSSM = $1_B$, **CC4yTC**.TCM = $1_B$ . . . 19-22
Figure 19-16 Start (as start)/ stop (as stop) - **CC4yTC**.STRM = $0_B$, **CC4yTC**.ENDM = $00_B$ 19-24
Figure 19-17 Start (as start)/ stop (as flush) - **CC4yTC**.STRM = $0_B$, **CC4yTC**.ENDM = $01_B$ 19-24
Figure 19-18 Start (as flush and start)/ stop (as stop) - **CC4yTC**.STRM = $1_B$, **CC4yTC**.ENDM = $00_B$ 19-25
Figure 19-19 Start (as start)/ stop (as flush and stop) - **CC4yTC**.STRM = $0_B$, **CC4yTC**.ENDM = $10_B$ 19-25
Figure 19-20 External counting direction . . . . . 19-26
Figure 19-21 External gating . . . . . 19-27


Subject to Agreement on the Use of Product Information

## List of Figures

Figure 19-22 External count . . . . . 19-28
Figure 19-23 External load . . . . . 19-29
Figure 19-24 External capture - **CC4yCMC**.CAP0S != $00_B$, **CC4yCMC**.CAP1S = $00_B$ . . 19-31
Figure 19-25 External capture - **CC4yCMC**.CAP0S != $00_B$, **CC4yCMC**.CAP1S != $00_B$ . 19-32
Figure 19-26 Slice capture logic . . . . . 19-33
Figure 19-27 External Capture - **CC4yTC**.SCE = $1_B$ . . . . . 19-34
Figure 19-28 Slice Capture Logic - **CC4yTC**.SCE = $1_B$ . . . . . 19-34
Figure 19-29 Capture Extended Read Back - Depth 4 . . . . . 19-36
Figure 19-30 Depth 4 software access example . . . . . 19-37
Figure 19-31 Capture Extended Read Back - Depth 2 . . . . . 19-38
Figure 19-32 Depth 2 software access example . . . . . 19-38
Figure 19-33 External modulation clearing the ST bit - **CC4yTC**.EMT = $0_B$ . . . . 19-39
Figure 19-34 External modulation clearing the ST bit - **CC4yTC**.EMT = $0_B$, **CC4yTC**.EMS = $1_B$ 19-40
Figure 19-35 External modulation gating the output - **CC4yTC**.EMT = $1_B$ . . . . . 19-40
Figure 19-36 Trap control diagram . . . . . 19-41
Figure 19-37 Trap timing diagram, **CC4yPSL**.PSL = $0_B$ (output passive level is $0_B$) . . . 19-42
Figure 19-38 Trap synchronization with the PWM signal, **CC4yTC**.TRPSE = $1_B$ 19-43
Figure 19-39 Status bit override . . . . . 19-44
Figure 19-40 Multi channel pattern synchronization . . . . . 19-45
Figure 19-41 Multi Channel mode for multiple Timer Slices . . . . . 19-45
Figure 19-42 CC4y Status bit and Output Path . . . . . 19-46
Figure 19-43 Multi Channel Mode Control Logic . . . . . 19-47
Figure 19-44 Timer Concatenation Example . . . . . 19-48
Figure 19-45 Timer Concatenation Link . . . . . 19-49
Figure 19-46 Capture/Load Timer Concatenation . . . . . 19-50
Figure 19-47 32 bit concatenation timing diagram . . . . . 19-51
Figure 19-48 Timer concatenation control logic . . . . . 19-52
Figure 19-49 Dither structure overview . . . . . 19-53
Figure 19-50 Dither control logic . . . . . 19-55
Figure 19-51 Dither timing diagram in edge aligned - **CC4yTC**.DITHE = $01_B$ . . . 19-55
Figure 19-52 Dither timing diagram in edge aligned - **CC4yTC**.DITHE = $10_B$ . . . 19-56
Figure 19-53 Dither timing diagram in edge aligned - **CC4yTC**.DITHE = $11_B$ . . . 19-56
Figure 19-54 Dither timing diagram in center aligned - **CC4yTC**.DITHE = $01_B$ . . 19-56
Figure 19-55 Dither timing diagram in center aligned - **CC4yTC**.DITHE = $10_B$ . . 19-57
Figure 19-56 Dither timing diagram in center aligned - **CC4yTC**.DITHE = $11_B$ . . 19-57
Figure 19-57 Floating prescaler in compare mode overview . . . . . 19-59
Figure 19-58 Floating Prescaler in capture mode overview . . . . . 19-60
Figure 19-59 PWM with 100% duty cycle - Edge Aligned Mode . . . . . 19-61
Figure 19-60 PWM with 100% duty cycle - Center Aligned Mode . . . . . 19-61


Subject to Agreement on the Use of Product Information

## List of Figures

Figure 19-61 PWM with 0% duty cycle - Edge Aligned Mode . . . . . 19-62
Figure 19-62 PWM with 0% duty cycle - Center Aligned Mode. . . . . 19-62
Figure 19-63 Floating Prescaler capture mode usage . . . . . 19-63
Figure 19-64 Floating Prescaler compare mode usage - Edge Aligned . . . . . 19-64
Figure 19-65 Floating Prescaler compare mode usage - Center Aligned . . . . . 19-64
Figure 19-66 Capture mode usage - single channel . . . . . 19-68
Figure 19-67 Three Capture profiles - **CC4yTC**.SCE = $1_B$ . . . . . 19-69
Figure 19-68 High dynamics capturing with software controlled timestamp . . . . 19-70
Figure 19-69 Extended read back during high load . . . . . 19-71
Figure 19-70 Capture grouping with extended read back . . . . . 19-71
Figure 19-71 Memory structure for extended read back . . . . . 19-73
Figure 19-72 Slice interrupt structure overview. . . . . 19-75
Figure 19-73 Slice Interrupt Node Pointer overview . . . . . 19-76
Figure 19-74 CCU4 service request overview. . . . . 19-77
Figure 19-75 CCU4 registers overview . . . . . 19-80
Figure 20-1 CCU8 block diagram . . . . . 20-6
Figure 20-2 CCU8 slice block diagram . . . . . 20-7
Figure 20-3 Slice input selector diagram. . . . . 20-10
Figure 20-4 Slice connection matrix diagram . . . . . 20-12
Figure 20-5 Timer start/stop control diagram . . . . . 20-13
Figure 20-6 Start multiple timers synchronously . . . . . 20-14
Figure 20-7 CC8y Status Bits . . . . . 20-15
Figure 20-8 Shadow registers overview . . . . . 20-17
Figure 20-9 Shadow transfer enable logic. . . . . 20-18
Figure 20-10 Shadow transfer timing example - center aligned mode . . . . . 20-19
Figure 20-11 Usage of the CCU8x.MCSS input . . . . . 20-20
Figure 20-12 Cascaded shadow transfer linking . . . . . 20-21
Figure 20-13 Cascade shadow transfer timing . . . . . 20-22
Figure 20-14 Edge aligned mode, **CC8yTC**.TCM = $0_B$ . . . . . 20-23
Figure 20-15 Center aligned mode, **CC8yTC**.TCM = $1_B$ . . . . . 20-24
Figure 20-16 Single shot edge aligned - **CC8yTC**.TSSM = $1_B$, **CC8yTC**.TCM = $0_B$ . . . . 20-25
Figure 20-17 Single shot center aligned - **CC8yTC**.TSSM = $1_B$, **CC8yTC**.TCM = $1_B$ . . . 20-25
Figure 20-18 Compare channels diagram. . . . . 20-26
Figure 20-19 Dead Time scheme . . . . . 20-27
Figure 20-20 Dead Time generator scheme . . . . . 20-28
Figure 20-21 Dead Time control cell . . . . . 20-29
Figure 20-22 Dead Time trigger with the Multi Channel pattern . . . . . 20-30
Figure 20-23 Edge Aligned with two independent channels scheme . . . . . 20-30
Figure 20-24 Edge Aligned - four outputs with dead time . . . . . 20-31
Figure 20-25 Edge Aligned with combined channels scheme. . . . . 20-32
Figure 20-26 Edge Aligned - Asymmetric PWM timing, **CC8yCR1**.CR1 <


Subject to Agreement on the Use of Product Information

**CC8yCR2**.CR2 20-33
Figure 20-27 Edge Aligned - Asymmetric PWM timing, **CC8yCR1**.CR1 > **CC8yCR2**.CR2 20-34
Figure 20-28 Center Aligned with two independent channels scheme . . . . . 20-35
Figure 20-29 Center aligned - Independent channel with dead time. . . . . 20-35
Figure 20-30 Center Aligned Asymmetric mode scheme . . . . . 20-36
Figure 20-31 Asymmetric Center aligned mode with dead time . . . . . 20-37
Figure 20-32 Start (as start)/ stop (as stop) - **CC8yTC**.STRM = $0_B$, **CC8yTC**.ENDM = $00_B$ 20-38
Figure 20-33 Start (as start)/ stop (as flush) - **CC8yTC**.STRM = $0_B$, **CC8yTC**.ENDM = $01_B$ 20-39
Figure 20-34 Start (as flush and start)/ stop (as stop) - **CC8yTC**.STRM = $1_B$, **CC8yTC**.ENDM = $00_B$ 20-39
Figure 20-35 Start (as start)/ stop (as flush and stop) - **CC8yTC**.STRM = $0_B$, **CC8yTC**.ENDM = $10_B$ 20-40
Figure 20-36 External counting direction. . . . . 20-41
Figure 20-37 External gating . . . . . 20-42
Figure 20-38 External count . . . . . 20-43
Figure 20-39 Timer load selection. . . . . 20-43
Figure 20-40 External load . . . . . 20-44
Figure 20-41 External capture - **CC8yCMC**.CAP0S != $00_B$, **CC8yCMC**.CAP1S = $00_B$. . 20-46
Figure 20-42 External capture - **CC8yCMC**.CAP0S != $00_B$, **CC8yCMC**.CAP1S != $00_B$ . 20-47
Figure 20-43 Slice capture logic . . . . . 20-48
Figure 20-44 External Capture - **CC8yTC**.SCE = $1_B$. . . . . 20-49
Figure 20-45 Slice Capture Logic - **CC8yTC**.SCE = $1_B$. . . . . 20-49
Figure 20-46 Capture Extended Read Back - Depth 4 . . . . . 20-51
Figure 20-47 Depth 4 software access example. . . . . 20-52
Figure 20-48 Capture Extended Read Back - Depth 2 . . . . . 20-53
Figure 20-49 Depth 2 software access example. . . . . 20-53
Figure 20-50 External modulation resets the ST bit - **CC8yTC**.EMS = $0_B$ . . . . . 20-54
Figure 20-51 External modulation clearing the ST bit - **CC8yTC**.EMS = $1_B$ . . . . 20-55
Figure 20-52 External modulation gating the output - **CC8yTC**.EMT = $1_B$ . . . . . 20-55
Figure 20-53 Trap control diagram . . . . . 20-56
Figure 20-54 Trap timing diagram, **CC8yTCST**.CDIR = 0 **CC8yPSL**.PSL = 0 . . 20-57
Figure 20-55 Trap synchronization with the PWM signal . . . . . 20-58
Figure 20-56 Status bit override . . . . . 20-59
Figure 20-57 Multi channel pattern synchronization . . . . . 20-60
Figure 20-58 CCU8 Multi Channel overview. . . . . 20-61
Figure 20-59 Multi Channel mode for multiple Timer Slices . . . . . 20-62
Figure 20-60 Output Control Diagram. . . . . 20-63
Figure 20-61 Multi Channel Pattern Synchronization Control . . . . . 20-64


Subject to Agreement on the Use of Product Information

## List of Figures

Figure 20-62 Timer concatenation example . . . . . 20-65
Figure 20-63 Timer concatenation link . . . . . 20-66
Figure 20-64 Capture/Load Timer Concatenation . . . . . 20-67
Figure 20-65 32 bit concatenation timing diagram . . . . . 20-68
Figure 20-66 Timer concatenation control logic . . . . . 20-69
Figure 20-67 Parity checker structure . . . . . 20-70
Figure 20-68 Parity checker logic . . . . . 20-72
Figure 20-69 Dither structure overview . . . . . 20-73
Figure 20-70 Dither control logic . . . . . 20-75
Figure 20-71 Dither timing diagram in edge aligned - **CC8yTC**.DITHE = $01_B$ . . . 20-75
Figure 20-72 Dither timing diagram in edge aligned - **CC8yTC**.DITHE = $10_B$ . . . 20-76
Figure 20-73 Dither timing diagram in edge aligned - **CC8yTC**.DITHE = $11_B$ . . . 20-76
Figure 20-74 Dither timing diagram in center aligned - **CC8yTC**.DITHE = $01_B$ . . 20-76
Figure 20-75 Dither timing diagram in center aligned - **CC8yTC**.DITHE = $10_B$ . . 20-77
Figure 20-76 Dither timing diagram in edge aligned - **CC8yTC**.DITHE = $11_B$ . . . 20-77
Figure 20-77 Floating prescaler in compare mode overview . . . . . 20-79
Figure 20-78 Floating Prescaler in capture mode overview . . . . . 20-80
Figure 20-79 PWM with 100% duty cycle - Edge Aligned Mode . . . . . 20-81
Figure 20-80 PWM with 100% duty cycle - Center Aligned Mode . . . . . 20-81
Figure 20-81 PWM with 0% duty cycle - Edge Aligned Mode . . . . . 20-82
Figure 20-82 PWM with 0% duty cycle - Center Aligned Mode . . . . . 20-82
Figure 20-83 Floating Prescaler capture mode usage . . . . . 20-83
Figure 20-84 Floating Prescaler compare mode usage - Edge Aligned . . . . . 20-84
Figure 20-85 Floating Prescaler compare mode usage - Center Aligned . . . . . 20-84
Figure 20-86 Capture mode usage - single channel . . . . . 20-88
Figure 20-87 Three Capture profiles - **CC8yTC**.SCE = $1_B$ . . . . . 20-89
Figure 20-88 High dynamics capturing with software controlled timestamp . . . . 20-90
Figure 20-89 Extended read back during high load . . . . . 20-91
Figure 20-90 Capture grouping with extended read back . . . . . 20-91
Figure 20-91 Memory structure for extended read back . . . . . 20-93
Figure 20-92 Parity Checker connections . . . . . 20-95
Figure 20-93 Parity Checker timing example . . . . . 20-96
Figure 20-94 Slice interrupt node pointer overview . . . . . 20-98
Figure 20-95 Slice interrupt selector overview . . . . . 20-98
Figure 20-96 CCU8 service request overview . . . . . 20-99
Figure 20-97 CCU8 registers overview . . . . . 20-104
Figure 21-1 POSIF block diagram . . . . . 21-4
Figure 21-2 Function selector diagram . . . . . 21-7
Figure 21-3 Hall Sensor Control Diagram . . . . . 21-8
Figure 21-4 Hall Sensor Compare logic . . . . . 21-9
Figure 21-5 Wrong Hall Event/Idle logic . . . . . 21-9
Figure 21-6 Multi-Channel Mode Diagram . . . . . 21-10
Figure 21-7 Hall Sensor timing diagram . . . . . 21-12


Subject to Agreement on the Use of Product Information

## List of Figures

Figure 21-8 Rotary encoder types - a) standard two phase plus index signal; b) clock plus direction 21-13
Figure 21-9 Quadrature Decoder Control Overview . . . . . 21-14
Figure 21-10 Quadrature clock generation . . . . . 21-14
Figure 21-11 Quadrature Decoder States . . . . . 21-15
Figure 21-12 Quadrature clock and direction timings . . . . . 21-16
Figure 21-13 Quadrature clock with jitter . . . . . 21-17
Figure 21-14 Index signals timing . . . . . 21-18
Figure 21-15 Hall Sensor Mode usage - profile 1 . . . . . 21-20
Figure 21-16 Hall Sensor Mode usage - profile 2 . . . . . 21-21
Figure 21-17 Quadrature Decoder Mode usage - profile 1 . . . . . 21-23
Figure 21-18 Quadrature Decoder Mode usage - profile 2 . . . . . 21-24
Figure 21-19 Quadrature Decoder Mode usage - profile 3 . . . . . 21-25
Figure 21-20 Slow rotating system example . . . . . 21-26
Figure 21-21 Quadrature Decoder Mode usage - profile 4 . . . . . 21-27
Figure 21-22 Stand-alone Multi-Channel Mode usage . . . . . 21-28
Figure 21-23 Hall Sensor Mode flags . . . . . 21-29
Figure 21-24 Interrupt node pointer overview - hall sensor mode . . . . . 21-30
Figure 21-25 Quadrature Decoder flags . . . . . 21-31
Figure 21-26 Interrupt node pointer overview - quadrature decoder mode . . . . . 21-32
Figure 21-27 POSIF registers overview . . . . . 21-36
Figure 22-1 BCCU Kernel Block Diagram . . . . . 22-2
Figure 22-2 Channel Block Diagram . . . . . 22-3
Figure 22-3 Dimming Engine Block Diagram . . . . . 22-4
Figure 22-4 Coarse Piecewise pseudo-exponential curve . . . . . 22-6
Figure 22-5 Coarse Piecewise pseudo-exponential curve with the dimming level on a logarithmic scale as the human eye sees it (real and ideal) 22-7
Figure 22-6 Dimming from 0 to 18 along the coarse curve without and with dither enabled 22-8
Figure 22-7 Dimming from 18 to 0 along the coarse curve without and with dither enabled 22-8
Figure 22-8 Lower section of the coarse curve (real and ideal) . . . . . 22-9
Figure 22-9 Lower section of the fine curve (real and ideal) . . . . . 22-11
Figure 22-10 Dithering examples . . . . . 22-12
Figure 22-11 Lower section of the coarse curve with dithering enabled (real and ideal) 22-12
Figure 22-12 Linear Walker Block Diagram . . . . . 22-13
Figure 22-13 Packer Block Diagram . . . . . 22-14
Figure 22-14 Signal Packing (example) . . . . . 22-15
Figure 22-15 BCCU Trigger Generation . . . . . 22-17
Figure 22-16 Trap Control . . . . . 22-18
Figure 22-17 BCCU clocks . . . . . 22-19
Figure 22-18 Service Request Nodes . . . . . 22-20


Subject to Agreement on the Use of Product Information

## List of Figures

Figure 22-19 Simple DAC on one pin . . . . . 22-22
Figure 23-1 General Structure of a digital Port Pin . . . . . 23-3
Figure 23-2 Port Pin in Power Save State. . . . . 23-8
Figure 23-3 Analog Port Structure. . . . . 23-9
Figure 23-4 Simplified Port Structure . . . . . 23-45
Figure 24-1 Startup sequence. . . . . 24-1
Figure 25-1 Baud Rate configuration sequence during XMC1300 ASC BSL entry. . . . . 25-5
Figure 25-2 XMC1300 Standard ASC BSL: Application download protocol . . . 25-6
Figure 25-3 Handshake protocol for XMC1300 ASC BSL entry . . . . . 25-8
Figure 26-1 Debug and Trace System block diagram. . . . . 26-2
Figure 26-2 SPD Encoding Example. . . . . 26-7
Figure 26-3 HAR - Halt After Reset Flow . . . . . 26-10
Figure 26-4 HOT PLUG or Warm Reset Flow. . . . . 26-12


Subject to Agreement on the Use of Product Information

# List of Tables


| | | |
|---|---|---|
| Table 1 | Bit Function Terminology | P-3 |
| Table 2 | Register Access Modes | P-3 |
| Table 2-1 | Summary of processor mode, execution, and stack use options | 2-4 |
| Table 2-2 | Core register set summary | 2-5 |
| Table 2-3 | PSR register combinations | 2-9 |
| Table 2-4 | CMSIS functions to generate some Cortex-M0 instructions | 2-16 |
| Table 2-5 | CMSIS functions to access the special registers | 2-17 |
| Table 2-6 | Memory access behavior | 2-20 |
| Table 2-7 | Cortex-M0 instructions | 2-23 |
| Table 2-8 | Exception types | 2-27 |
| Table 2-9 | Properties of the different exception types | 2-27 |
| Table 2-10 | Remapped Vector Table | 2-30 |
| Table 2-11 | Exception return behavior | 2-34 |
| Table 2-12 | Core peripheral register regions | 2-38 |
| Table 2-13 | Register Overview | 2-40 |
| Table 2-14 | System fault handler priority fields | 2-49 |
| Table 4-1 | Abbreviations | 4-1 |
| Table 4-2 | Interrupt services per module | 4-4 |
| Table 5-1 | Interrupt Node assignment | 5-1 |
| Table 5-2 | CMSIS functions for NVIC control | 5-3 |
| Table 5-3 | CMSIS access NVIC functions | 5-4 |
| Table 5-4 | Registers Address Space | 5-8 |
| Table 5-5 | Register Overview | 5-8 |
| Table 5-6 | Interrupt Source Overview | 5-14 |
| Table 6-1 | Registers Address Space | 6-10 |
| Table 6-2 | Register Overview | 6-10 |
| Table 6-3 | ERU0 Pin Connections | 6-17 |
| Table 7-1 | Overflow Error Conditions | 7-5 |
| Table 7-2 | CORDIC Applications | 7-8 |
| Table 7-3 | CORDIC Function Inherent Gain Factor for Result Data | 7-11 |
| Table 7-4 | CORDIC Coprocessor Operating Modes and Corresponding Result Data 7-12 | |
| Table 7-5 | Summary of X,Y and Z data format | 7-16 |
| Table 7-6 | Normalized Deviation of a Calculation | 7-18 |
| Table 7-7 | Precomputed Scaled Values for atan($2^{-i}$) | 7-20 |
| Table 7-8 | Precomputed Scaled Values for atanh($2^{-i}$) | 7-20 |
| Table 7-9 | Registers Address Space | 7-27 |
| Table 7-10 | Register Overview | 7-27 |
| Table 7-11 | Module Interconnects | 7-46 |
| Table 8-1 | Memory Regions | 8-2 |
| Table 8-2 | Memory Map | 8-4 |



Subject to Agreement on the Use of Product Information

## List of Tables

Table 8-3 Memory Protection Measures . . . . . 8-10
Table 8-4 List of Protected Register Bit Fields . . . . . 8-11
Table 9-1 Module Specific Definitions . . . . . 9-3
Table 9-2 Registers Address Space . . . . . 9-11
Table 9-3 Registers Overview . . . . . 9-11
Table 9-4 Incremental Update of a Block with Specially Constructed Data . . 9-22
Table 10-1 Peripherals Availability and Privilege Access Control . . . . . 10-1
Table 10-2 Registers Address Space . . . . . 10-3
Table 10-3 Register Overview . . . . . 10-3
Table 11-1 Application Features . . . . . 11-2
Table 11-2 Registers Address Space . . . . . 11-9
Table 11-3 Register Overview . . . . . 11-9
Table 11-4 Pin Table . . . . . 11-16
Table 12-1 Application Features . . . . . 12-1
Table 12-2 Registers Address Space . . . . . 12-7
Table 12-3 Register Overview . . . . . 12-7
Table 12-4 Pin Connections . . . . . 12-19
Table 13-1 Service Requests . . . . . 13-5
Table 13-2 PID Values of XMC1300 System ROM Table . . . . . 13-6
Table 13-3 Reset Overview . . . . . 13-14
Table 13-4 DCO calibration data in Flash sector 0 (CS0) . . . . . 13-20
Table 13-5 Base Addresses of sub-sections of SCU registers . . . . . 13-23
Table 13-6 Registers Address Space . . . . . 13-23
Table 13-7 Registers Overview . . . . . 13-24
Table 14-1 Registers Address Space . . . . . 14-3
Table 14-2 Registers Overview . . . . . 14-4
Table 15-1 Abbreviations . . . . . 15-1
Table 15-2 Input Signals for Different Protocols . . . . . 15-7
Table 15-3 Output Signals for Different Protocols . . . . . 15-8
Table 15-4 USIC Communication Channel Behavior . . . . . 15-16
Table 15-5 Data Transfer Events and Interrupt Handling . . . . . 15-19
Table 15-6 Baud Rate Generator Event and Interrupt Handling . . . . . 15-21
Table 15-7 Protocol-specific Events and Interrupt Handling . . . . . 15-22
Table 15-8 Transmit Shift Register Composition . . . . . 15-33
Table 15-9 Receive Shift Register Composition . . . . . 15-38
Table 15-10 Transmit Buffer Events and Interrupt Handling . . . . . 15-44
Table 15-11 Receive Buffer Events and Interrupt Handling . . . . . 15-49
Table 15-12 SSC Communication Signals . . . . . 15-75
Table 15-13 Master Transmit Data Formats . . . . . 15-121
Table 15-14 Slave Transmit Data Format . . . . . 15-122
Table 15-15 Valid TDF Codes Overview . . . . . 15-123
Table 15-16 TDF Code Sequence for Master Transmit . . . . . 15-126
Table 15-17 TDF Code Sequence for Master Receive (7-bit Addressing Mode) . . . . .


Subject to Agreement on the Use of Product Information

15-126
Table 15-18 TDF Code Sequence for Master Receive (10-bit Addressing Mode) . . . . 15-127
Table 15-19 IIS IO Signals . . . . . 15-135
Table 15-20 USIC Kernel-Related and Kernel Registers . . . . . 15-154
Table 15-21 Registers Address Space . . . . . 15-157
Table 15-22 FIFO and Reserved Address Space . . . . . 15-157
Table 15-23 USIC Module 0 Channel 0 Interconnects . . . . . 15-228
Table 15-24 USIC Module 0 Channel 1 Interconnects . . . . . 15-232
Table 15-25 USIC Module 0 Module Interconnects . . . . . 15-235
Table 16-1 Abbreviations used in ADC chapter . . . . . 16-1
Table 16-2 VADC Applications . . . . . 16-3
Table 16-3 Analog Part Power-Down Control Options . . . . . 16-14
Table 16-4 Properties of Result FIFO Registers . . . . . 16-49
Table 16-5 Function of Bitfield DRCTR . . . . . 16-50
Table 16-6 EMUX Control Signal Coding . . . . . 16-63
Table 16-7 Registers Address Space . . . . . 16-66
Table 16-8 Registers Overview . . . . . 16-66
Table 16-9 TS16_SSIG Trigger Set VADC . . . . . 16-74
Table 16-10 Register Protection Groups . . . . . 16-79
Table 16-11 Sample Time Coding . . . . . 16-117
Table 16-12 General Converter Configuration in the XMC1300 . . . . . 16-153
Table 16-13 Synchronization Groups in the XMC1300 . . . . . 16-154
Table 16-14 Analog Connections in the XMC1300 . . . . . 16-155
Table 16-15 Digital Connections in the XMC1300 . . . . . 16-156
Table 17-1 Registers Address Space . . . . . 17-5
Table 17-2 Registers Overview . . . . . 17-5
Table 17-3 Analog Comparator Pin Connections . . . . . 17-12
Table 17-4 Out of Range Comparator Pin Connections . . . . . 17-12
Table 18-1 Registers Address Space . . . . . 18-2
Table 18-2 Registers Overview . . . . . 18-2
Table 19-1 Abbreviations table . . . . . 19-1
Table 19-2 Applications summary . . . . . 19-3
Table 19-3 CCU4 slice pin description . . . . . 19-7
Table 19-4 Connection matrix available functions . . . . . 19-10
Table 19-5 Dither bit reverse counter . . . . . 19-54
Table 19-6 Dither modes . . . . . 19-54
Table 19-7 Timer clock division options . . . . . 19-58
Table 19-8 Bit reverse distribution . . . . . 19-65
Table 19-9 Interrupt sources . . . . . 19-74
Table 19-10 External clock operating conditions . . . . . 19-78
Table 19-11 Registers Address Space . . . . . 19-80
Table 19-12 Register Overview of CCU4 . . . . . 19-81


Subject to Agreement on the Use of Product Information

## List of Tables

Table 19-13 CCU40 Pin Connections . . . . . 19-137
Table 19-14 CCU40 - CC40 Pin Connections . . . . . 19-138
Table 19-15 CCU40 - CC41 Pin Connections . . . . . 19-139
Table 19-16 CCU40 - CC42 Pin Connections . . . . . 19-141
Table 19-17 CCU40 - CC43 Pin Connections . . . . . 19-142
Table 20-1 Abbreviations table . . . . . 20-1
Table 20-2 Applications summary . . . . . 20-3
Table 20-3 CCU8 slice pin description . . . . . 20-8
Table 20-4 Connection matrix available functions . . . . . 20-11
Table 20-5 Dead time prescaler values . . . . . 20-27
Table 20-6 Dither bit reverse counter . . . . . 20-74
Table 20-7 Dither modes . . . . . 20-74
Table 20-8 Timer clock division options . . . . . 20-78
Table 20-9 Bit reverse distribution . . . . . 20-85
Table 20-10 Interrupt sources . . . . . 20-97
Table 20-11 External clock operating conditions . . . . . 20-100
Table 20-12 Registers Address Space . . . . . 20-103
Table 20-13 Register Overview of CCU8 . . . . . 20-105
Table 20-14 CCU80 Pin Connections . . . . . 20-178
Table 20-15 CCU80 - CC80 Pin Connections . . . . . 20-179
Table 20-16 CCU80 - CC81 Pin Connections . . . . . 20-181
Table 20-17 CCU80 - CC82 Pin Connections . . . . . 20-182
Table 20-18 CCU80 - CC83 Pin Connections . . . . . 20-184
Table 21-1 Abbreviations table . . . . . 21-1
Table 21-2 Applications summary . . . . . 21-3
Table 21-3 POSIF slice pin description . . . . . 21-5
Table 21-4 External Hall/Rotary signals operating conditions . . . . . 21-33
Table 21-5 Registers Address Space . . . . . 21-36
Table 21-6 Register Overview of POSIF . . . . . 21-36
Table 21-7 POSIF0 Pin Connections . . . . . 21-62
Table 22-1 Coarse Piece-Wise Pseudo-Exponential Curve . . . . . 22-4
Table 22-2 Quantized CoarsePiece-Wise Pseudo-Exponential Curve . . . . . 22-7
Table 22-3 Fine Piece-Wise Pseudo-Exponential Curve . . . . . 22-9
Table 22-4 Fine Piece-Wise Pseudo-Exponential Curve . . . . . 22-10
Table 22-5 Registers Address Space . . . . . 22-23
Table 22-6 Register Overview . . . . . 22-23
Table 22-7 Pin Connections . . . . . 22-49
Table 23-1 Port/Pin Overview . . . . . 23-1
Table 23-2 Registers Address Space . . . . . 23-12
Table 23-3 Register Overview . . . . . 23-12
Table 23-4 Registers Access Rights and Reset Classes . . . . . 23-14
Table 23-5 Standard PCx Coding . . . . . 23-18
Table 23-6 Pad Hysteresis Selection . . . . . 23-19


Subject to Agreement on the Use of Product Information

## List of Tables

Table 23-7 Function of the Bits PRx and PSx . . . . . 23-32
Table 23-8 PCx Coding in Deep-Sleep mode . . . . . 23-37
Table 23-9 Package Pin Mapping Description . . . . . 23-41
Table 23-10 Package Pin Mapping . . . . . 23-41
Table 23-11 Port Pin for Boot Modes . . . . . 23-44
Table 23-12 Port I/O Function Description . . . . . 23-45
Table 23-13 Hardware Controlled I/O Function Description . . . . . 23-46
Table 23-14 Port I/O Functions . . . . . 23-47
Table 23-15 Hardware Controlled I/O Functions . . . . . 23-50
Table 24-1 Flash data for SSW and user SW in XMC1300 . . . . . 24-10
Table 25-1 Supported Baud Rates . . . . . 25-4
Table 25-2 Scaling Factor Examples . . . . . 25-4
Table 25-3 Handshake protocol data definitions in XMC1300 ASC BSL . . . . . 25-7
Table 25-4 SSC BL: Determining the EEPROM Type and data-flow . . . . . 25-9
Table 25-5 User routines' in XMC1300 ROM . . . . . 25-11
Table 25-6 Status indicators returned by NVM routines in XMC1300 ROM . . 25-12
Table 25-7 Basic Flash data for SSW and user SW in XMC1300 . . . . . 25-15
Table 26-1 Peripheral Suspend support . . . . . 26-13
Table 26-2 SWD toplevel IO signal . . . . . 26-15
Table 26-3 ARM CoreSight™ Component ID code . . . . . 26-18
Table 26-4 Peripheral ID Values of XMC1300 ROM Table . . . . . 26-18
Table 26-5 Registers Address Space . . . . . 26-19
Table 26-6 Register Overview . . . . . 26-19


Subject to Agreement on the Use of Product Information

w w w . i n f i n e o n . c o m

Published by Infineon Technologies AG