

# 16-Bit

Architecture

## XE167xM/XE164xM/XE 162xM Derivatives

16-Bit Single-Chip

Real Time Signal Controller

XE166 Family / Base Line

User's Manual

V2.0 2009-03

Microcontrollers

**Edition 2009-03**

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

**© 2009 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](http://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.

# 16-Bit

Architecture

## XE167xM/XE164xM/XE 162xM Derivatives

16-Bit Single-Chip

Real Time Signal Controller

XE166 Family / Base Line

User's Manual

V2.0 2009-03

Microcontrollers

## **XC2200M**

### **Revision History: V2.0, 2009-03**

Previous Version(s):

V1.2, 2008-09 (V1.1 skipped)

V1.0, 2008-06

<b>Page</b>	<b>Subjects (major changes since last revision)</b>
all	Two volumes (System Units, Peripheral Units) combined to one document
<a href="#">3-5</a>	Description of IMB registers added
<a href="#">3-11ff</a> , <a href="#">12-3</a>	SBRAM description added
<a href="#">3-16</a>	Description of protected bits added (moved from Architectural Overview)
<a href="#">5-58</a>	Section Bit Protection improved
<a href="#">6-3</a>	Hint added for programming an MPU range of 256 bytes
<a href="#">7-45ff</a>	Name of ERU interrupt control registers corrected
<a href="#">8-3</a>	Figure updated: Clock Generation Unit
<a href="#">8-27</a>	Description of WUOSCCON updated
<a href="#">8-28</a>	Description of bitfield HPOSCCON.MODE changed
<a href="#">8-109ff</a>	Description of some bitfields updated (EVR control registers)
<a href="#">8-123ff</a>	Description of reserved bits changed (GSCEN, GSCPERSTAT, GSCPERSTATEN)
<a href="#">8-153</a>	EXOCON reset value corrected
<a href="#">8-155</a>	Figure updated: SCU Interrupt Structure
<a href="#">8-156</a>	Short names of wake-up interrupts corrected
<a href="#">8-171ff</a>	Bit descriptions updated
<a href="#">8-186</a>	Note added to description of bit WDTCS.OE
<a href="#">8-189</a>	Figure updated: SCU Trap Structure
<a href="#">8-190</a>	Default PLL trap corrected
<a href="#">8-215ff</a>	Description of ECC error handling improved
<a href="#">8-229f</a>	Registers IDDMPM and IDDMP1 added
<a href="#">9-5</a>	Information concerning reset behavior added
<a href="#">9-39</a>	Overlaid analog input channels specified (ADC0/ADC1)
<a href="#">9-40</a>	Signal U3C1_SELO1 added
<a href="#">12-33</a>	Bootstrap loader configuration table corrected



**XC2200M**

**Revision History: V2.0, 2009-03**

<b>16-32<sup>f</sup>, 16-59<sup>f</sup></b>	GPT timer registers reworked, interrupt control registers removed
<b>16-67</b>	Register table added
<b>16-70</b>	Section Port Control added
<b>17-7</b>	Signal T14INT added
<b>19-28<sup>ff</sup></b>	CCU2 registers rearranged into grouped section
<b>20-39<sup>ff</sup></b>	Bitnumber of bit MCC63S corrected to 6, CCPOSx corrected to CCPOS6x
<b>21-2</b>	USIC feature list updated
<b>23-3</b>	Not-readable wake-up counter added to difference list

**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](mailto:mcdocu.comments@infineon.com)



## Summary Of Chapters

This table summarizes all chapters of this document, so you immediately can find the reference to the desired section.

	<b>Summary Of Chapters</b> . . . . .	0-1
	<b>Table Of Contents</b> . . . . .	0-3
<b>1</b>	<b>Introduction</b> . . . . .	1-1
<b>2</b>	<b>Architectural Overview</b> . . . . .	2-1
<b>3</b>	<b>Memory Organization</b> . . . . .	3-1
<b>4</b>	<b>Memory Checker Module (MCHK)</b> . . . . .	4-1
<b>5</b>	<b>Central Processing Unit (CPU)</b> . . . . .	5-1
<b>6</b>	<b>Memory Protection Unit (MPU)</b> . . . . .	6-1
<b>7</b>	<b>Interrupt and Exception Control</b> . . . . .	7-1
<b>8</b>	<b>System Control Unit (SCU)</b> . . . . .	8-1
<b>9</b>	<b>Parallel Ports</b> . . . . .	9-1
<b>10</b>	<b>Dedicated Pins</b> . . . . .	10-1
<b>11</b>	<b>External Bus Controller (EBC)</b> . . . . .	11-1
<b>12</b>	<b>Startup Configuration and Bootstrap Loading</b> . . . . .	12-1
<b>13</b>	<b>Debug System</b> . . . . .	13-1
<b>14</b>	<b>Instruction Set Summary</b> . . . . .	14-1
<b>15</b>	<b>Device Specification</b> . . . . .	15-1
<b>16</b>	<b>General Purpose Timer Units</b> . . . . .	16-1
<b>17</b>	<b>Real Time Clock</b> . . . . .	17-1
<b>18</b>	<b>Analog to Digital Converter</b> . . . . .	18-1
<b>19</b>	<b>Capture/Compare Unit</b> . . . . .	19-1
<b>20</b>	<b>Capture/Compare Unit 6 (CCU6)</b> . . . . .	20-1
<b>21</b>	<b>Universal Serial Interface Channel</b> . . . . .	21-1
<b>22</b>	<b>Controller Area Network (MultiCAN) Controller</b> . . . . .	22-1

**Summary Of Chapters**

<b>23</b>	<b>Appendix: Functional and Operational Updates .....</b>	<b>23-1</b>
	<b>Keyword Index .....</b>	<b>24-1</b>
	<b>Register Index .....</b>	<b>25-8</b>

## Table Of Contents

This table of contents lists all sections of this User's Manual, so you can find the reference to the desired section.

	<b>Summary Of Chapters</b> .....	0-1
	<b>Table Of Contents</b> .....	0-3
<b>1</b>	<b>Introduction</b> .....	1-1
1.1	Members of the 16-bit Microcontroller Families .....	1-3
1.2	Summary of Basic Features .....	1-5
1.3	Abbreviations .....	1-9
1.4	Naming Conventions .....	1-10
<b>2</b>	<b>Architectural Overview</b> .....	2-1
2.1	Basic CPU Concepts and Optimizations .....	2-2
2.1.1	Memory Protection Unit (MPU) .....	2-3
2.1.2	High Instruction Bandwidth/Fast Execution .....	2-4
2.1.3	Powerful Execution Units .....	2-5
2.1.4	High Performance Branch-, Call-, and Loop-Processing .....	2-6
2.1.5	Consistent and Optimized Instruction Formats .....	2-7
2.1.6	Programmable Multiple Priority Interrupt System .....	2-8
2.1.7	Interfaces to System Resources .....	2-9
2.2	On-Chip System Resources .....	2-10
2.3	On-Chip Peripheral Blocks .....	2-15
2.4	Clock Generation .....	2-33
2.5	Power Management .....	2-34
2.6	On-Chip Debug Support (OCDS) .....	2-35
<b>3</b>	<b>Memory Organization</b> .....	3-1
3.1	Address Mapping .....	3-3
3.2	Register Areas .....	3-5
3.3	Data Memory Areas .....	3-10
3.4	Program Memory Areas .....	3-12
3.4.1	Program/Data SRAM (PSRAM) .....	3-13
3.4.2	Non-Volatile Program Memory (Flash) .....	3-14
3.5	System Stack .....	3-15
3.6	Protected Bits .....	3-16
3.7	IO Areas .....	3-17
3.8	External Memory Space .....	3-18
3.9	Crossing Memory Boundaries .....	3-18

**Table Of Contents**

3.10	Embedded Flash Memory .....	3-20
3.10.1	Definitions .....	3-20
3.10.2	Operating Modes .....	3-22
3.10.3	Operations .....	3-24
3.10.4	Details of Command Sequences .....	3-27
3.10.5	Sequence Errors .....	3-37
3.10.6	Instructions for Executing Program and Erase Jobs Concurrently ..	3-38
3.10.7	Data Integrity .....	3-41
3.10.8	Protection Handling Details .....	3-45
3.10.9	Protection Handling Examples .....	3-52
3.10.10	EEPROM Emulation .....	3-54
3.10.11	Interrupt Generation .....	3-56
3.10.12	Recommendations for Optimized Flash Usage .....	3-56
3.11	On-Chip Program Memory Control .....	3-58
3.11.1	Overview .....	3-58
3.11.2	Register Interface .....	3-60
3.11.3	Startup, Shutdown .....	3-79
3.11.4	Error Reporting Summary .....	3-80
3.12	Data Retention Memories .....	3-82
3.12.1	Standby RAM Accesses .....	3-82
3.12.2	Standby RAM Registers .....	3-84
<b>4</b>	<b>Memory Checker Module (MCHK) .....</b>	<b>4-1</b>
4.1	Operational Overview .....	4-1
4.2	Functional Description .....	4-2
4.2.1	Principle of the LFSR .....	4-4
4.2.2	Principle of the MISR .....	4-6
4.2.3	Commonly used Polynomials .....	4-7
4.2.4	Architecture of the Memory Checker Module .....	4-8
4.2.5	Preferable Usage of the Memory Checker Module .....	4-10
4.2.6	Calculation of Seed Values (Magic Word) .....	4-10
4.2.7	Example Application .....	4-12
4.3	Memory Checker Module Registers .....	4-14
4.3.1	Memory Checker Input Register .....	4-15
4.3.2	Memory Checker Result Registers .....	4-16
4.3.3	Memory Checker Count Register .....	4-17
4.3.4	Memory Checker Polynomial Registers .....	4-18
4.4	General Registers .....	4-20
4.4.1	ID Register .....	4-20
4.5	Interfaces of the MCHK Module .....	4-20
<b>5</b>	<b>Central Processing Unit (CPU) .....</b>	<b>5-1</b>
5.1	Components of the CPU .....	5-4
5.2	Instruction Fetch and Program Flow Control .....	5-5

**Table Of Contents**

5.2.1	Branch Detection and Branch Prediction Rules .....	5-7
5.2.2	Zero-Cycle Jumps .....	5-7
5.2.3	Atomic and Extend Instructions .....	5-8
5.3	Instruction Processing Pipeline .....	5-9
5.3.1	Access to the IO Area .....	5-10
5.3.2	Pipeline Conflicts .....	5-10
5.4	CPU Configuration Registers .....	5-21
5.5	Use of General Purpose Registers .....	5-24
5.5.1	GPR Addressing Modes .....	5-26
5.5.2	Context Switching .....	5-28
5.6	Code Addressing .....	5-33
5.7	Data Addressing .....	5-36
5.7.1	Short Addressing Modes .....	5-36
5.7.2	Long Addressing Modes .....	5-38
5.7.3	Indirect Addressing Modes .....	5-41
5.7.4	DSP Addressing Modes .....	5-43
5.7.5	The System Stack .....	5-49
5.8	Standard Data Processing .....	5-53
5.8.1	16-bit Adder/Subtractor, Barrel Shifter, and 16-bit Logic Unit .....	5-57
5.8.2	Bit Manipulation Unit .....	5-58
5.8.3	Multiply and Divide Unit .....	5-60
5.9	DSP Data Processing (MAC Unit) .....	5-62
5.9.1	MAC Unit Control .....	5-63
5.9.2	Representation of Numbers and Rounding .....	5-63
5.9.3	The 16-bit by 16-bit Signed/Unsigned Multiplier and Scaler .....	5-64
5.9.4	Concatenation Unit .....	5-64
5.9.5	One-bit Scaler .....	5-64
5.9.6	The 40-bit Adder/Subtractor .....	5-64
5.9.7	The Data Limiter .....	5-65
5.9.8	The Accumulator Shifter .....	5-65
5.9.9	The 40-bit Signed Accumulator Register .....	5-66
5.9.10	The MAC Unit Status Word MSW .....	5-68
5.9.11	The Repeat Counter MRW .....	5-70
5.10	Constant Registers .....	5-72
<b>6</b>	<b>Memory Protection Unit (MPU) .....</b>	<b>6-1</b>
6.1	Functional Overview .....	6-1
6.2	Memory Protection Registers .....	6-3
6.2.1	Protection Range Registers .....	6-3
6.2.2	Protection Mode Registers .....	6-5
6.2.3	Protection Range Data Register .....	6-8
6.2.4	Protection Range Address Register .....	6-8
6.3	Functional Description .....	6-10

**Table Of Contents**

6.3.1	Enabling Protection .....	6-10
6.3.2	Protection Levels .....	6-10
6.3.3	Intersecting Memory Ranges .....	6-11
6.3.4	Protection of the MPU registers .....	6-11
6.3.5	Accessing SFRs and GPRs .....	6-12
6.3.6	Interrupts and PECs Handling .....	6-12
6.3.7	Special handling of RETI instruction .....	6-13
6.3.8	Context Switch operations .....	6-13
6.3.9	Debugger Access Permissions .....	6-14
6.3.10	Invalid Access Traps .....	6-14
6.4	Initializing and using the MPU .....	6-15
6.4.1	Installing Protection .....	6-15
6.4.2	Changing Protection Level .....	6-17
6.4.3	Executing privileged code from non-privileged one .....	6-17
6.4.4	Fast task switches .....	6-17
6.4.5	Register Bank Selection .....	6-17
6.4.6	Debugger Use Cases .....	6-17
<b>7</b>	<b>Interrupt and Exception Control .....</b>	<b>7-1</b>
7.1	Interrupt System Structure .....	7-3
7.2	Interrupt Arbitration .....	7-5
7.3	Interrupt Control .....	7-8
7.3.1	Interrupt Priority Level and Group Level .....	7-9
7.3.2	General Interrupt Control Functions in Register PSW .....	7-10
7.3.3	Selective Interrupt Disabling .....	7-11
7.3.4	Interrupt Class Management .....	7-12
7.4	Interrupt Vector Table .....	7-14
7.5	Interrupt Jump Table Cache .....	7-16
7.6	CPU Status Saving .....	7-19
7.7	CPU Context Switch .....	7-20
7.8	Fast Bank Switching .....	7-21
7.9	Trap Functions .....	7-23
7.9.1	Software Traps .....	7-23
7.9.2	Hardware Traps .....	7-24
7.10	Peripheral Event Controller .....	7-31
7.10.1	PEC Control Registers .....	7-32
7.10.2	The PEC Source and Destination Pointer .....	7-40
7.10.3	PEC Interrupt Processing Summary .....	7-42
7.10.4	PEC Channel Assignment .....	7-43
7.11	External Interrupts .....	7-45
7.11.1	External Request Unit .....	7-45
7.11.2	Using Peripheral Pins .....	7-45
7.12	OCDS Requests .....	7-47

**Table Of Contents**

7.13	Service Request Latency .....	7-48
7.14	Interrupt Nodes .....	7-50
7.14.1	Physical Interrupt Nodes .....	7-50
7.14.2	Interrupt Node Sharing .....	7-54
7.14.3	Interrupt Source Select Registers .....	7-56
7.15	Interrupt and PEC Configuration Registers .....	7-58
<b>8</b>	<b>System Control Unit (SCU) .....</b>	<b>8-1</b>
8.1	Clock Generation Unit .....	8-3
8.1.1	Overview .....	8-3
8.1.2	Trimmed Current Controlled Wake-Up Clock (OSC_WU) .....	8-5
8.1.3	High Precision Oscillator Circuit (OSC_HP) .....	8-5
8.1.4	Phase-Locked Loop (PLL) Module .....	8-7
8.1.5	Clock Control Unit .....	8-17
8.1.6	External Clock Output .....	8-24
8.1.7	CGU Registers .....	8-27
8.2	System Timer Function (STM) .....	8-48
8.2.1	STM Registers .....	8-49
8.3	Wake-up Timer (WUT) .....	8-51
8.3.1	Wake-up Timer Operation .....	8-51
8.3.2	WUT Registers .....	8-53
8.4	Reset Operation .....	8-57
8.4.1	Reset Architecture .....	8-57
8.4.2	General Reset Operation .....	8-60
8.4.3	Debug Reset Assertion .....	8-62
8.4.4	Coupling of Reset Types .....	8-62
8.4.5	Reset Request Trigger Sources .....	8-63
8.4.6	Module Reset Behavior .....	8-65
8.4.7	Reset Controller Registers .....	8-67
8.5	External Service Request (ESR) Pins .....	8-77
8.5.1	General Operation .....	8-77
8.5.2	ESR Control Registers .....	8-83
8.5.3	ESR Data Register .....	8-89
8.6	Power Supply and Control .....	8-91
8.6.1	Supply Watchdog (SWD) .....	8-93
8.6.2	Monitoring the Voltage Level of a Core Domain .....	8-100
8.6.3	Controlling the Voltage Level of a Core Domain .....	8-107
8.6.4	Handling the Power System .....	8-117
8.7	Global State Controller (GSC) .....	8-118
8.7.1	GSC Control Flow .....	8-118
8.7.2	GSC Registers .....	8-122
8.8	Software Boot Support .....	8-136
8.8.1	Start-up Registers .....	8-136



**Table Of Contents**

8.9	External Request Unit (ERU) .....	8-137
8.9.1	Introduction .....	8-137
8.9.2	ERU Input Connections .....	8-139
8.9.3	External Request Select Unit (ERSx) .....	8-140
8.9.4	Event Trigger Logic (ETLx) .....	8-141
8.9.5	Connecting Matrix .....	8-143
8.9.6	Output Gating Unit (OGUy) .....	8-144
8.9.7	ERU Output Connections .....	8-147
8.9.8	ERU Registers .....	8-148
8.10	SCU Interrupt Generation .....	8-155
8.10.1	Interrupt Support .....	8-156
8.10.2	SCU Interrupt Sources .....	8-156
8.10.3	Interrupt Control Registers .....	8-157
8.11	Temperature Compensation Unit .....	8-176
8.11.1	Temperature Compensation Registers .....	8-178
8.12	Watchdog Timer (WDT) .....	8-180
8.12.1	Introduction .....	8-180
8.12.2	Overview .....	8-180
8.12.3	Functional Description .....	8-181
8.12.4	WDT Kernel Registers .....	8-185
8.13	SCU Trap Generation .....	8-189
8.13.1	Trap Support .....	8-189
8.13.2	SCU Trap Sources .....	8-190
8.13.3	SCU Trap Control Registers .....	8-191
8.14	Memory Content Protection for RAM Areas .....	8-203
8.14.1	Protection Mode Selection .....	8-204
8.14.2	Parity Error Handling .....	8-206
8.14.3	ECC Error Handling .....	8-215
8.15	Register Control .....	8-220
8.15.1	Register Access Control .....	8-220
8.15.2	Register Protection Registers .....	8-223
8.16	Miscellaneous System Registers .....	8-225
8.16.1	System Registers .....	8-225
8.16.2	Identification Block .....	8-226
8.16.3	Marker Memory .....	8-231
8.17	Implementation .....	8-232
8.17.1	Clock Generation Unit .....	8-232
8.17.2	External Service Requests (ESR) .....	8-233
8.17.3	External Request Unit (ERU) .....	8-235
8.18	SCU Register Addresses .....	8-241
<b>9</b>	<b>Parallel Ports .....</b>	<b>9-1</b>
9.1	General Description .....	9-2

**Table Of Contents**

9.1.1	Basic Port Operation .....	9-2
9.1.2	Input Stage Control .....	9-5
9.1.3	Output Driver Control .....	9-5
9.2	Port Register Description .....	9-6
9.2.1	Pad Driver Control .....	9-6
9.2.2	Port Output Register .....	9-9
9.2.3	Port Output Modification Register .....	9-10
9.2.4	Port Input Register .....	9-12
9.2.5	Port Input/Output Control Registers .....	9-13
9.2.6	Port Digital Input Disable Register .....	9-16
9.3	Port Description .....	9-17
9.3.1	Port 0 .....	9-18
9.3.2	Port 1 .....	9-19
9.3.3	Port 2 .....	9-20
9.3.4	Port 3 .....	9-22
9.3.5	Port 4 .....	9-23
9.3.6	Port 5 .....	9-24
9.3.7	Port 6 .....	9-25
9.3.8	Port 7 .....	9-26
9.3.9	Port 8 .....	9-27
9.3.10	Port 9 .....	9-28
9.3.11	Port 10 .....	9-29
9.3.12	Port 11 .....	9-31
9.3.13	Port 15 .....	9-32
9.4	Pin Description .....	9-33
<b>10</b>	<b>Dedicated Pins .....</b>	<b>10-1</b>
<b>11</b>	<b>External Bus Controller (EBC) .....</b>	<b>11-1</b>
11.1	Summary of Features .....	11-1
11.2	Overview .....	11-1
11.3	Naming Conventions .....	11-2
11.4	Timing Description .....	11-2
11.4.1	Bus Phases .....	11-2
11.4.2	Demultiplexed Bus .....	11-5
11.4.3	Multiplexed Bus .....	11-6
11.4.4	Fastest Access Cycles .....	11-7
11.5	Address Windows .....	11-9
11.5.1	Window Allocation .....	11-9
11.5.2	Window Overlap .....	11-10
11.6	Ready Controlled Bus Access .....	11-11
11.6.1	Enabling the Ready Control .....	11-11
11.6.2	Synchronous and Asynchronous READY .....	11-11
11.6.3	Combining the READY function with predefined wait states .....	11-12

**Table Of Contents**

11.7	External Bus Arbitration . . . . .	11-13
11.7.1	Initialization of Arbitration . . . . .	11-13
11.7.2	Arbitration Master Scheme . . . . .	11-13
11.7.3	Arbitration Slave Scheme . . . . .	11-15
11.7.4	Bus Lock Function . . . . .	11-15
11.7.5	Direct Master Slave Connection . . . . .	11-16
11.8	EBC Idle State . . . . .	11-17
11.9	Register Description . . . . .	11-18
11.9.1	EBC Mode Registers . . . . .	11-18
11.9.2	Timing Control Registers . . . . .	11-20
11.9.3	Function Control Registers . . . . .	11-22
11.9.4	Address Window Selection Registers . . . . .	11-23
11.10	EBC Implementation in XE16xyM . . . . .	11-24
11.10.1	Unused Registers . . . . .	11-24
11.10.2	Access Control to LXBUS Modules . . . . .	11-24
11.10.3	Shutdown Control . . . . .	11-24
11.10.4	Dedicated Registers . . . . .	11-25
<b>12</b>	<b>Startup Configuration and Bootstrap Loading . . . . .</b>	<b>12-1</b>
12.1	Startup Mode Selection . . . . .	12-1
12.2	Device Status after Startup . . . . .	12-2
12.2.1	Registers modified by the Startup Procedure . . . . .	12-2
12.2.2	System Frequency after Startup . . . . .	12-3
12.2.3	Watchdog Timer handling . . . . .	12-4
12.2.4	Startup Error state . . . . .	12-5
12.3	Supported Startup Modes and Options . . . . .	12-6
12.3.1	Basic Startup Modes . . . . .	12-7
12.3.2	Startup Modes with Debug Support . . . . .	12-8
12.3.3	Special Startup Features . . . . .	12-11
12.4	Internal Start . . . . .	12-15
12.5	External Start . . . . .	12-15
12.5.1	Specific Settings . . . . .	12-17
12.6	Bootstrap Loading . . . . .	12-18
12.6.1	General Functionality . . . . .	12-18
12.6.2	Bootstrap Loaders using UART Protocol . . . . .	12-20
12.6.3	Synchronous Serial Channel Bootstrap Loader . . . . .	12-27
12.6.4	CAN Bootstrap Loader . . . . .	12-30
12.6.5	Summary of Bootstrap Loader Modes . . . . .	12-33
<b>13</b>	<b>Debug System . . . . .</b>	<b>13-1</b>
13.1	Debug Interface . . . . .	13-2
13.1.1	Routing of Debug Signals . . . . .	13-3
13.2	OCDS Module . . . . .	13-5
13.2.1	Debug Events . . . . .	13-6

**Table Of Contents**

13.2.2	Debug Actions .....	13-7
13.3	Cerberus .....	13-8
13.3.1	Functional Overview .....	13-9
13.4	Emulation Device .....	13-10
13.4.1	MCDS Use Cases .....	13-10
13.4.2	MCDS Features .....	13-11
13.5	Boundary-Scan .....	13-12
<b>14</b>	<b>Instruction Set Summary .....</b>	<b>14-1</b>
<b>15</b>	<b>Device Specification .....</b>	<b>15-1</b>
<b>16</b>	<b>General Purpose Timer Units .....</b>	<b>16-1</b>
16.1	Timer Block GPT1 .....	16-2
16.1.1	GPT1 Core Timer T3 Control .....	16-4
16.1.2	GPT1 Core Timer T3 Operating Modes .....	16-8
16.1.3	GPT1 Auxiliary Timers T2/T4 Control .....	16-15
16.1.4	GPT1 Auxiliary Timers T2/T4 Operating Modes .....	16-20
16.1.5	GPT1 Clock Signal Control .....	16-29
16.1.6	GPT1 Timer Registers .....	16-32
16.1.7	Interrupt Control for GPT1 Timers .....	16-33
16.2	Timer Block GPT2 .....	16-34
16.2.1	GPT2 Core Timer T6 Control .....	16-36
16.2.2	GPT2 Core Timer T6 Operating Modes .....	16-40
16.2.3	GPT2 Auxiliary Timer T5 Control .....	16-43
16.2.4	GPT2 Auxiliary Timer T5 Operating Modes .....	16-46
16.2.5	GPT2 Register CAPREL Operating Modes .....	16-50
16.2.6	GPT2 Clock Signal Control .....	16-56
16.2.7	GPT2 Timer Registers .....	16-59
16.2.8	Interrupt Control for GPT2 Timers and CAPREL .....	16-60
16.3	Miscellaneous Registers .....	16-61
16.4	Register Table .....	16-67
16.5	Interfaces of the GPT Module .....	16-68
<b>17</b>	<b>Real Time Clock .....</b>	<b>17-1</b>
17.1	Defining the RTC Time Base .....	17-2
17.2	RTC Run Control .....	17-5
17.3	RTC Operating Modes .....	17-7
17.4	48-bit Timer Operation .....	17-11
17.5	System Clock Operation .....	17-11
17.6	Cyclic Interrupt Generation .....	17-12
17.7	RTC Interrupt Generation .....	17-13
17.8	Miscellaneous Registers .....	17-15
<b>18</b>	<b>Analog to Digital Converter .....</b>	<b>18-1</b>

**Table Of Contents**

18.1	Introduction . . . . .	18-1
18.1.1	ADC Block Diagram . . . . .	18-2
18.1.2	Feature Set . . . . .	18-3
18.1.3	Abbreviations . . . . .	18-4
18.1.4	ADC Kernel Overview . . . . .	18-5
18.1.5	Conversion Request Unit . . . . .	18-7
18.1.6	Conversion Result Unit . . . . .	18-9
18.1.7	Interrupt Structure . . . . .	18-10
18.1.8	Electrical Models . . . . .	18-12
18.1.9	Transfer Characteristics and Error Definitions . . . . .	18-15
18.2	Operating the ADC . . . . .	18-16
18.2.1	Register Overview . . . . .	18-17
18.2.2	Mode Control . . . . .	18-20
18.2.3	Module Activation and Power Saving Modes . . . . .	18-22
18.2.4	Clocking Scheme . . . . .	18-23
18.2.5	General ADC Registers . . . . .	18-24
18.2.6	Request Source Arbiter . . . . .	18-34
18.2.7	Arbiter Registers . . . . .	18-39
18.2.8	Scan Request Source Handling . . . . .	18-41
18.2.9	Scan Request Source Registers . . . . .	18-45
18.2.10	Sequential Request Source Handling . . . . .	18-49
18.2.11	Sequential Source Registers . . . . .	18-54
18.2.12	Channel-Related Functions . . . . .	18-65
18.2.13	Channel-Related Registers . . . . .	18-70
18.2.14	Conversion Result Handling . . . . .	18-80
18.2.15	Conversion Result-Related Registers . . . . .	18-88
18.2.16	Multiplexer Test Mode for CH7 . . . . .	18-98
18.2.17	External Multiplexer Control . . . . .	18-99
18.2.18	Synchronized Conversions for Parallel Sampling . . . . .	18-102
18.2.19	Equidistant Sampling . . . . .	18-106
18.2.20	Broken Wire Detection . . . . .	18-108
18.2.21	Additional Feature Registers . . . . .	18-110
18.3	Implementation . . . . .	18-117
18.3.1	Address Map . . . . .	18-117
18.3.2	Interrupt Control Registers . . . . .	18-117
18.3.3	Analog Connections . . . . .	18-119
18.3.4	Digital Connections . . . . .	18-122
<b>19</b>	<b>Capture/Compare Unit . . . . .</b>	<b>19-1</b>
19.1	Functional Overview . . . . .	19-2
19.1.1	The CAPCOM Timers . . . . .	19-3
19.1.2	Timer Interrupt . . . . .	19-7
19.1.3	Capture/Compare Channels . . . . .	19-8

**Table Of Contents**

19.1.4	Capture Mode .....	19-9
19.1.5	Compare Modes .....	19-10
19.1.6	Double-Register Compare Mode .....	19-18
19.1.7	CAPCOM Interrupts .....	19-21
19.1.8	Compare Output Signal Generation .....	19-21
19.1.9	Single Event Mode .....	19-22
19.1.10	Staggered and Non-Staggered Operation .....	19-22
19.1.11	External Input Signal Requirements .....	19-27
19.2	CAPCOM2 Registers .....	19-28
19.2.1	Identification Register .....	19-31
19.2.2	Timer 7/8 Registers .....	19-32
19.2.3	Timer 7/8 Control Register .....	19-34
19.2.4	Capture/Compare Registers .....	19-36
19.2.5	Capture/Compare Mode Registers .....	19-37
19.2.6	Compare Output Register .....	19-39
19.2.7	Double-Register Compare Mode Register .....	19-40
19.2.8	IOC Register .....	19-41
19.2.9	Single Event Mode Register .....	19-42
19.2.10	KSCCFG Register .....	19-43
19.3	Module Implementation .....	19-46
19.3.1	Interfaces of the CAPCOM2 Unit .....	19-46
<b>20</b>	<b>Capture/Compare Unit 6 (CCU6)</b> .....	<b>20-1</b>
20.1	Introduction .....	20-1
20.1.1	Feature Set Overview .....	20-2
20.1.2	Block Diagram .....	20-3
20.1.3	Register Overview .....	20-4
20.2	Operating Timer T12 .....	20-7
20.2.1	T12 Overview .....	20-8
20.2.2	T12 Counting Scheme .....	20-10
20.2.3	T12 Compare Mode .....	20-14
20.2.4	Compare Mode Output Path .....	20-21
20.2.5	T12 Capture Modes .....	20-26
20.2.6	T12 Shadow Register Transfer .....	20-30
20.2.7	Timer T12 Operating Mode Selection .....	20-31
20.2.8	T12 related Registers .....	20-32
20.2.9	Capture/Compare Control Registers .....	20-37
20.3	Operating Timer T13 .....	20-49
20.3.1	T13 Overview .....	20-49
20.3.2	T13 Counting Scheme .....	20-52
20.3.3	T13 Compare Mode .....	20-57
20.3.4	Compare Mode Output Path .....	20-59
20.3.5	T13 Shadow Register Transfer .....	20-60

**Table Of Contents**

20.3.6	T13 related Registers .....	20-62
20.4	Trap Handling .....	20-65
20.5	Multi-Channel Mode .....	20-67
20.6	Hall Sensor Mode .....	20-70
20.6.1	Hall Pattern Evaluation .....	20-71
20.6.2	Hall Pattern Compare Logic .....	20-73
20.6.3	Hall Mode Flags .....	20-74
20.6.4	Hall Mode for Brushless DC-Motor Control .....	20-76
20.7	Modulation Control Registers .....	20-78
20.7.1	Modulation Control .....	20-78
20.7.2	Trap Control Register .....	20-80
20.7.3	Passive State Level Register .....	20-83
20.7.4	Multi-Channel Mode Registers .....	20-84
20.8	Interrupt Handling .....	20-89
20.8.1	Interrupt Structure .....	20-89
20.8.2	Interrupt Registers .....	20-91
20.9	General Module Operation .....	20-103
20.9.1	Mode Control .....	20-103
20.9.2	Input Selection .....	20-106
20.9.3	General Registers .....	20-107
20.10	Implementation .....	20-115
20.10.1	Address Map .....	20-115
20.10.2	Interrupt Control Registers .....	20-116
20.10.3	Synchronous Start Feature .....	20-117
20.10.4	Digital Connections .....	20-118
<b>21</b>	<b>Universal Serial Interface Channel .....</b>	<b>21-1</b>
21.1	Introduction .....	21-1
21.1.1	Feature Set Overview .....	21-2
21.1.2	Channel Structure .....	21-5
21.1.3	Input Stages .....	21-6
21.1.4	Output Signals .....	21-7
21.1.5	Baud Rate Generator .....	21-8
21.1.6	Channel Events and Interrupts .....	21-9
21.1.7	Data Shifting and Handling .....	21-9
21.2	Operating the USIC .....	21-13
21.2.1	Register Overview .....	21-13
21.2.2	Operating the USIC Communication Channel .....	21-18
21.2.3	Channel Control and Configuration Registers .....	21-25
21.2.4	Protocol Related Registers .....	21-33
21.2.5	Operating the Input Stages .....	21-36
21.2.6	Input Stage Register .....	21-38
21.2.7	Operating the Baud Rate Generator .....	21-41

**Table Of Contents**

21.2.8	Baud Rate Generator Registers .....	21-46
21.2.9	Operating the Transmit Data Path .....	21-51
21.2.10	Operating the Receive Data Path .....	21-55
21.2.11	Transfer Control and Status Registers .....	21-57
21.2.12	Data Buffer Registers .....	21-69
21.2.13	Operating the FIFO Data Buffer .....	21-79
21.2.14	FIFO Buffer and Bypass Registers .....	21-89
21.3	Asynchronous Serial Channel (ASC = UART) .....	21-110
21.3.1	Signal Description .....	21-110
21.3.2	Frame Format .....	21-111
21.3.3	Operating the ASC .....	21-114
21.3.4	ASC Protocol Registers .....	21-123
21.3.5	Hardware LIN Support .....	21-129
21.4	Synchronous Serial Channel (SSC) .....	21-131
21.4.1	Signal Description .....	21-131
21.4.2	Operating the SSC .....	21-139
21.4.3	Operating the SSC in Master Mode .....	21-143
21.4.4	Operating the SSC in Slave Mode .....	21-150
21.4.5	SSC Protocol Registers .....	21-152
21.4.6	SSC Timing Considerations .....	21-158
21.5	Inter-IC Bus Protocol (IIC) .....	21-161
21.5.1	Introduction .....	21-161
21.5.2	Operating the IIC .....	21-165
21.5.3	Symbol Timing .....	21-171
21.5.4	Data Flow Handling .....	21-174
21.5.5	IIC Protocol Registers .....	21-179
21.6	IIS Protocol .....	21-185
21.6.1	Introduction .....	21-185
21.6.2	Operating the IIS .....	21-189
21.6.3	Operating the IIS in Master Mode .....	21-194
21.6.4	Operating the IIS in Slave Mode .....	21-198
21.6.5	IIS Protocol Registers .....	21-199
21.7	USIC Implementation in XE16xyM .....	21-205
21.7.1	Implementation Overview .....	21-205
21.7.2	Channel Features .....	21-206
21.7.3	Address Map .....	21-207
21.7.4	Module Identification Registers .....	21-208
21.7.5	Interrupt Control Registers .....	21-210
21.7.6	Input/Output Connections .....	21-212
<b>22</b>	<b>Controller Area Network (MultiCAN) Controller .....</b>	<b>22-1</b>
22.1	MultiCAN Short Description .....	22-1
22.1.1	Overview .....	22-1



**Table Of Contents**

22.1.2	CAN Features .....	22-2
22.2	CAN Functional Description .....	22-3
22.2.1	Conventions and Definitions .....	22-3
22.2.2	Introduction .....	22-3
22.2.3	CAN Node Control .....	22-9
22.2.4	Message Object List Structure .....	22-13
22.2.5	CAN Node Analysis Features .....	22-18
22.2.6	Message Acceptance Filtering .....	22-21
22.2.7	Message Postprocessing Interface .....	22-24
22.2.8	Message Object Data Handling .....	22-28
22.2.9	Message Object Functionality .....	22-35
22.3	MultiCAN Kernel Registers .....	22-44
22.3.1	Register Address Map .....	22-44
22.3.2	Global MultiCAN Registers .....	22-49
22.3.3	CAN Node Specific Registers .....	22-62
22.3.4	Message Object Registers .....	22-79
22.4	General Control and Status .....	22-102
22.4.1	Clock Control .....	22-102
22.4.2	Port Input Control .....	22-103
22.4.3	Suspend Mode .....	22-104
22.4.4	Interrupt Structure .....	22-105
22.5	MultiCAN Module Implementation .....	22-106
22.5.1	Interfaces of the CAN Module .....	22-106
22.5.2	Module Clock Generation .....	22-107
22.5.3	Mode Control Behavior .....	22-116
22.5.4	Mode Control .....	22-117
22.5.5	Mode Control Register Description .....	22-119
22.5.6	Connection of External Signals .....	22-122
22.5.7	MultiCAN Module Register Address Map .....	22-126
<b>23</b>	<b>Appendix: Functional and Operational Updates .....</b>	<b>23-1</b>
	<b>Keyword Index .....</b>	<b>24-1</b>
	<b>Register Index .....</b>	<b>25-8</b>

## **1 Introduction**

The rapidly growing area of embedded control applications is representing one of the most time-critical operating environments for today's microcontrollers. Complex control algorithms have to be processed based on a large number of digital as well as analog input signals, and the appropriate output signals must be generated within a defined maximum response time. Embedded control applications also are often sensitive to board space, power consumption, and overall system cost.

Embedded control applications therefore require microcontrollers, which:

- offer a high level of system integration
- eliminate the need for additional peripheral devices and the associated software overhead
- provide system security and fail-safe mechanisms
- provide effective means to control (and reduce) the device's power consumption

The increasing complexity of embedded control applications requires microcontrollers for new high-end embedded control systems to possess a significant CPU performance and peripheral functionality. To achieve this high performance goal, Infineon has decided to develop its families of 16/32-bit CMOS microcontrollers without the constraints of backward compatibility with previous architectures.

Nonetheless the architectures of these microcontroller families pursue successful hardware and software concepts, which have been established in Infineon's popular 8-bit controller families, while delivering 32-bit performance.

This established functionality, which has been the basis for system solutions in a wide range of application areas, is amended with flexible peripheral modules and effective power control features. The sum of this provides the prerequisites for powerful, yet efficient systems-on-chip.

### **Solutions for Industrial Systems**

The XE166 derivatives provide solutions for the requirements of manifold industrial applications by combining versatile controlling power with DSP functionality and sophisticated peripheral modules. Thus, it supports a wide field of applications:

- Motor Control (pumps, fans, compressors, servos, CNC-machines, robots, process control, conveyor belts, ...)
- Renewable Energy (wind energy converters, photovoltaics, fuel cells, battery storage, hydro generators, micro turbines, ...)
- Power Supply (Uninterruptable Power Supplies, general power supplies, battery chargers, lamp ballast, ...)
- Transportation (locomotives, trains, subways, buses, trucks, fork lifts, agricultural vehicles, traffic lights, ...)

Infineon's high quality standards make the XE166 Real-Time Controllers an ideal choice for robust and reliable systems.

## **About this Manual**

This manual describes the functionality of a number of microcontroller types of the Infineon XE166 Family.

These microcontrollers provide identical functionality to a large extent, but each device type has specific unique features as indicated here.

The descriptions in this manual cover a superset of the provided features.

A detailed list of features of the various derivatives is provided in the Data Sheets of these products.

This manual is valid for these derivatives and describes all variations of the different available temperature ranges and packages.

For simplicity, these various device types are referred to by the collective term **XE16xyM** throughout this manual. The complete Pro Electron conforming designations are listed in the respective Data Sheets.

Some sections of this manual do not refer to all of the XE16xyM derivatives which are currently available or planned (such as devices with different types of on-chip memory or peripherals). These sections contain respective notes wherever possible.

## **1.1 Members of the 16-bit Microcontroller Families**

The microcontrollers in the Infineon 16-bit family have been designed to meet the high performance requirements of real-time embedded control applications. The architecture of this family has been optimized for high instruction throughput and minimized response time to external stimuli (interrupts). Intelligent peripheral subsystems have been integrated to reduce the need for CPU intervention to a minimum extent. This also minimizes the need for communication via the external bus interface. The high flexibility of this architecture allows to serve the diverse and varying needs of different application areas such as automotive, industrial control, or data communications.

The core of the 16-bit family has been developed with a modular family concept in mind. All family members execute an efficient control-optimized instruction set (additional instructions for members of the second generation). This allows easy and quick implementation of new family members with different internal memory sizes and technologies, different sets of on-chip peripherals, and/or different numbers of IO pins.

The XBUS/LXBus concept (internal representation of the external bus interface) provides a straightforward path for building application-specific derivatives by integrating application-specific peripheral modules with the standard on-chip peripherals.

As programs for embedded control applications become larger, high level languages are favored by programmers. High level language programs are easier to write, to debug and to maintain. The C166 Family supports this starting with its 2<sup>nd</sup> generation.

The 80C166-type microcontrollers were the **first generation** of the 16-bit controller family. These devices established the C166 architecture.

The C165-type and C167-type devices are members of the **second generation** of this family. This second generation is even more powerful due to additional instructions for HLL support, an increased address space, increased internal RAM, and highly efficient management of various resources on the external bus.

**Enhanced derivatives** of this second generation provide more features such as additional internal high-speed RAM, an integrated CAN-Module, an on-chip PLL, etc.

The design of more efficient systems may require the integration of application-specific peripherals to boost system performance while minimizing the part count. These efforts are supported by the XBUS, defined for the Infineon 16-bit microcontrollers (second generation). The XBUS is an internal representation of the external bus interface which opens and simplifies the integration of peripherals by standardizing the required interface. One representative taking advantage of this technology is the integrated CAN module.

The C165-type devices are reduced functionality versions of the C167 because they do not have the A/D converter, the CAPCOM units, and the PWM module. This results in a smaller package, reduced power consumption, and design savings.

The C164-type devices, the C167CS derivatives, and some of the C161-type devices are further enhanced by a flexible power management and form the **third generation** of

## Introduction

the 16-bit controller family. This power management mechanism provides an effective means to control the power that is consumed in a certain state of the controller and thus minimizes the overall power consumption for a given application.

The XC16x derivatives represent the **fourth generation** of the 16-bit controller family. The XC166 Family dramatically increases the performance of 16-bit microcontrollers by several major improvements and additions. The MAC-unit adds DSP-functionality to handle digital filter algorithms and greatly reduces the execution time of multiplications and divisions. It also adds 32-bit operations and a 40-bit accumulator. The 5-stage pipeline, single-cycle execution of most instructions, and PEC-transfers within the complete addressing range increase system performance. The previous XBUS is replaced by the optimized LXBus. Debugging the target system is supported by integrated functions for On-Chip Debug Support (OCDS).

The present XE166 Family of microcontrollers builds the **fifth generation** of 16-bit microcontrollers which provides 32-bit performance and takes users and applications a considerable step towards industry's target of systems on chip. Integrated memories and peripherals allow compact systems, the integrated core power supply and control reduces system requirements to one single voltage supply, the powerful combination of CPU and MAC-unit is unleashed by optimized compilers. This leaves no performance gap towards 32-bit systems.

Also there are devices with specific functional units.

The devices may be offered in different packages, temperature ranges and speed classes.

Additional standard and application-specific derivatives are planned and are in development.

*Note: Not all derivatives will be offered in all temperature ranges, speed classes, packages, or program memory variations.*

Information about specific versions and derivatives will be made available with the devices themselves. Contact your Infineon representative for up-to-date material or refer to <http://www.infineon.com/microcontrollers>.

*Note: As the architecture and the basic features, such as the CPU core and built-in peripherals, are identical for most of the currently offered versions of the XE16xyM, descriptions within this manual that refer to the "XE16xyM" also apply to the other variations, unless otherwise noted.*

## **1.2 Summary of Basic Features**

The XE16xyM devices are enhanced members of the Infineon XE166 Family of full featured single-chip CMOS microcontrollers.

*Note: The various derivatives are referred to as XE16xyM throughout this manual.*

The XE16xyM combines the extended functionality and performance of the C166SV2 Core with powerful on-chip peripheral subsystems and on-chip memory units and provides several means for power reduction.

The following key features contribute to the high performance of the XE16xyM:

### **Intelligent On-Chip Peripheral Subsystems**

- Two synchronizable A/D Converters with programmable resolution (10-bit or 8-bit) and conversion time (down to below 1  $\mu$ s), up to 24 analog input channels, auto scan modes, channel injection, data reduction features
- One Capture/Compare Unit with 2 independent time bases, very flexible PWM unit/event recording unit with different operating modes, includes two 16-bit timers/counters, maximum resolution  $f_{SYS}$
- Up to Four Capture/Compare Units for flexible PWM Signal Generation (CCU6) (3/6 Capture/Compare Channels and 1 Compare Channel)
- Two Multifunctional General Purpose Timer Units:
  - GPT1: three 16-bit timers/counters, maximum resolution  $f_{SYS}/4$
  - GPT2: two 16-bit timers/counters, maximum resolution  $f_{SYS}/2$
- Up to Eight Serial Channels with baud rate generator, receive/transmit FIFOs, programmable data length and shift direction, usable as UART, SPI-like, IIC, IIS, and LIN interface
- Controller Area Network (MultiCAN) Module, Rev. 2.0B active, up to six nodes operating independently or exchanging data via a gateway function, Full-CAN/Basic-CAN
- Real Time Clock with alarm interrupt
- Watchdog Timer with programmable time intervals
- Bootstrap Loaders for flexible system initialization
- Protection management for system configuration and control registers

### **Integrated On-Chip Memories**

- 8 Kbytes on-chip Stand-By RAM (SBRAM) for data
- 2 Kbytes Dual-Port RAM (DPRAM) for variables, register banks, and stacks
- Up to 16 Kbytes on-chip high-speed Data SRAM (DSRAM) for variables and stacks
- Up to 32 Kbytes on-chip high-speed Program/Data SRAM (PSRAM) for code and data
- Up to 576 Kbytes on-chip Flash Program Memory for instruction code or constant data

*Note: The system stack can be located in any memory area within the complete addressing range.*

### **High Performance 16-bit CPU with Five-Stage Pipeline and MAC Unit**

- Single clock cycle instruction execution
- 1 cycle minimum instruction cycle time (most instructions)
- 1 cycle multiplication (16-bit  $\times$  16-bit)
- 4 + 17 cycles division (32-bit / 16-bit), 4 cycles delay, 17 cycles background execution
- 1 cycle multiply and accumulate instruction (MAC) execution
- 32-bit addition and 32-bit subtraction (MAC unit)
- 40-bit barrel shifter and 40-bit accumulator
- Automatic saturation or rounding included
- Multiple high bandwidth internal data buses
- Register-based design with multiple, variable register banks
- Two additional fast register banks
- Fast context switching support
- 16 Mbytes of linear address space for code and data (von Neumann architecture)
- System stack cache support with automatic stack overflow/underflow detection
- High performance branch, call, and loop processing
- Zero-cycle jump execution

### **Control Oriented Instruction Set with High Efficiency**

- Bit, byte, and word data types
- Flexible and efficient addressing modes for high code density
- Enhanced boolean bit manipulation with direct addressability of 6 Kbits for peripheral control and user-defined flags
- Hardware traps to identify exception conditions during runtime
- HLL support for semaphore operations and efficient data access

### **Power Management Features**

- Two IO power domains fulfill system requirements from 3 V to 5 V
- Gated clock concept for improved power consumption and EMC
- Programmable system slowdown via clock generation unit
- Flexible management of peripherals, can be individually disabled
- Programmable frequency output

### **16-Priority-Level Interrupt System**

- 96 interrupt nodes with separate interrupt vectors on 15 priority levels (8 group levels)
- 7 cycles minimum interrupt latency in case of internal program execution
- Fast external interrupts
- Programmable external interrupt source selection
- Programmable vector table (start location and step-width)



### **8-Channel Peripheral Event Controller (PEC)**

- Interrupt driven single cycle data transfer
- Programmable PEC interrupt request level, (15 down to 8)
- Transfer count option  
(standard CPU interrupt after programmable number of PEC transfers)
- Separate interrupt level for PEC termination interrupts selectable
- Overhead from saving and restoring system state for interrupt requests eliminated
- Full 24-bit addresses for source and destination pointers, supporting transfers within the total address space

### **On-Chip Debug Support**

- Communication through DAP interface (2-wire) or JTAG interface (5-wire)
- On-chip debug controller with optional break interface
- Hardware, software and external pin breakpoints
- Up to 4 instruction pointer breakpoints
- Debug event control, e.g. with monitor call or CPU halt or trigger of data transfer
- Dedicated DEBUG instructions with control via DAP/JTAG interface
- Access to any internal register or memory location via DAP/JTAG interface
- Single step support and watchpoints with MOV-injection

### **Input/Output Lines With Individual Bit Addressability**

- Tri-stated in input mode
- Push/pull or open drain output mode
- Programmable port driver control
- Two I/O power domains with a supply voltage range from 3.0 V to 5.5 V  
(core-logic and oscillator input voltage is 1.5 V)

### **Infineon CMOS Process**

- Low power CMOS technology enables power saving modes with flexible power management.

### **Green Plastic Low-Profile Quad Flat Pack (LQFP) Packages**

- PG-LQFP-144, 20 × 20 mm body, 0.5 mm (19.7 mil) lead spacing, surface mount technology
- PG-LQFP-100, 14 × 14 mm body, 0.5 mm (19.7 mil) lead spacing, surface mount technology
- PG-LQFP-64, 10 × 10 mm body, 0.5 mm (19.7 mil) lead spacing, surface mount technology



**Complete Development Support**

For the development tool support of its microcontrollers, Infineon collaborates with third party tool suppliers. There is a wide range of tool suppliers world-wide, ranging from local niche manufacturers to multinational companies with broad product portfolios, offering powerful development tools for the complete variety of Infineon microcontroller families, providing a remarkable selection of price-performance classes as well as early availability of high quality key tools such as compilers, assemblers, simulators, debuggers or in-circuit emulators.

Infineon incorporates its strategic tool partners very early into the product development process, making sure embedded system developers get reliable, well-tuned tool solutions, which help them unleash the power of Infineon microcontrollers in the most effective way and with the shortest possible learning curve.

The tool environment for the Infineon 16-bit microcontrollers includes the following tools:

- Compilers (C/C++)
- Macro-assemblers, linkers, locators, library managers, format-converters
- Architectural simulators
- HLL debuggers
- Real-time operating systems
- In-circuit emulators
- Logic analyzer disassemblers
- Starter kits and evaluation boards
- Chip configuration code generation tool (DAvE)

### **1.3 Abbreviations**

The following acronyms and terms are used within this document:

ADC	Analog Digital Converter
ALE	Address Latch Enable
ALU	Arithmetic and Logic Unit
ASC	Asynchronous/synchronous Serial Channel
CAN	Controller Area Network (License Bosch)
CAPCOM	CAPture and COMpare unit
CISC	Complex Instruction Set Computing
CMOS	Complementary Metal Oxide Silicon
CPU	Central Processing Unit
DAP	Device Access Port
DMU	Data Management Unit
EBC	External Bus Controller
ECC	Error Correction Code
ESFR	Extended Special Function Register
EVVR	Embedded Validated Voltage Regulator
Flash	Non-volatile memory that may be electrically erased
GPR	General Purpose Register
GPT	General Purpose Timer unit
HLL	High Level Language
IIC	Inter Integrated Circuit (Bus)
IIS	Inter Integrated Circuit Sound (Bus)
IO	Input/Output
JTAG	Joint Test Access Group
LIN	Local Interconnect Network
LPR	Low Power Reference
LQFP	Low Profile Quad Flat Pack
LXBus	Internal representation of the external bus
MAC	Multiply/Accumulate (unit)
MPU	Memory Protection Unit

OCDS	On-Chip Debug Support
OTP	One-Time Programmable memory
PEC	Peripheral Event Controller
PLA	Programmable Logic Array
PLL	Phase Locked Loop
PMU	Program Management Unit
PVC	Power Validation Circuit
PWM	Pulse Width Modulation
RAM	Random Access Memory
RISC	Reduced Instruction Set Computing
ROM	Read Only Memory
RTC	Real Time Clock
SFR	Special Function Register
SSC	Synchronous Serial Channel
SWD	Supply Watchdog
UART	Universal Asynchronous Receiver/Transmitter
USIC	Universal Serial Interface Channel

## **1.4 Naming Conventions**

The diverse bitfields used for control functions and status indication and the registers housing them are equipped with unique names wherever applicable. Thereby these control structures can be referred to by their names rather than by their location. This makes the descriptions by far more comprehensible.

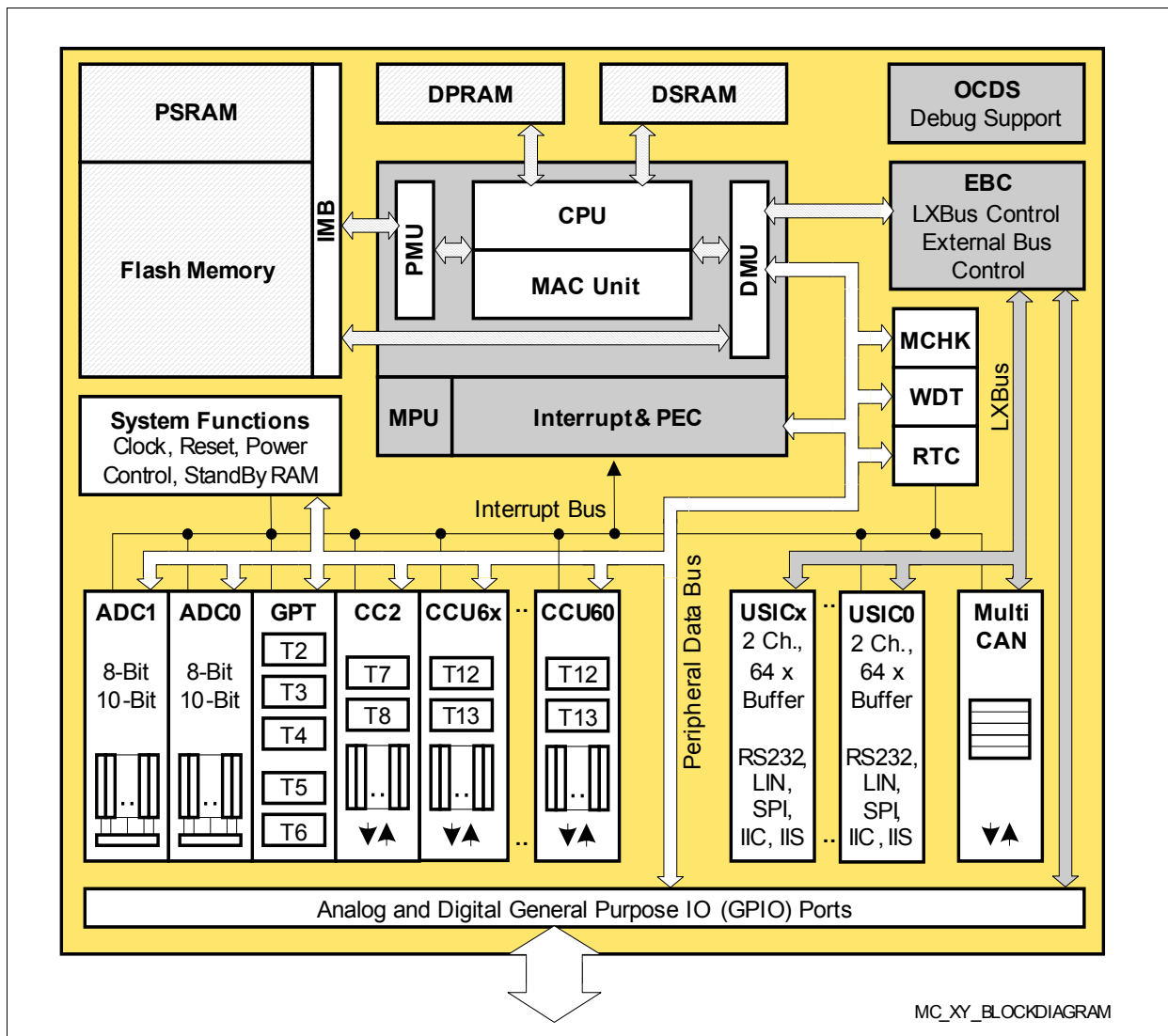
To describe regular structures (such as ports) indices are used instead of a plethora of similar bit names, so bit 3 of port 5 is referred to as P5.3.

Where it helps to clarify the relation between several named structures, the next higher level is added to the respective name to make it unambiguous.

The term ADC0\_GLOBCTR clearly identifies register GLOBCTR as part of module ADC0, the term SYSCON0.CLKSEL clearly identifies bitfield CLKSEL as part of register SYSCON0.

## 2 Architectural Overview

The architecture of the XE16xyM core combines the advantages of both RISC and CISC processors in a very well-balanced way. This computing and controlling power is completed by the DSP-functionality of the MAC-unit. The XE16xyM integrates this powerful CPU core with a set of powerful peripheral units into one chip and connects them very efficiently. On-chip memory blocks with dedicated buses and control units store code and data. This combination of features results in a high performance microcontroller, which is the right choice not only for today's applications, but also for future engineering challenges. One of the buses used concurrently on the XE16xyM is the LXBus, an internal representation of the external bus interface. This bus provides a standardized method for integrating additional application-specific peripherals into derivatives of the standard XE16xyM.

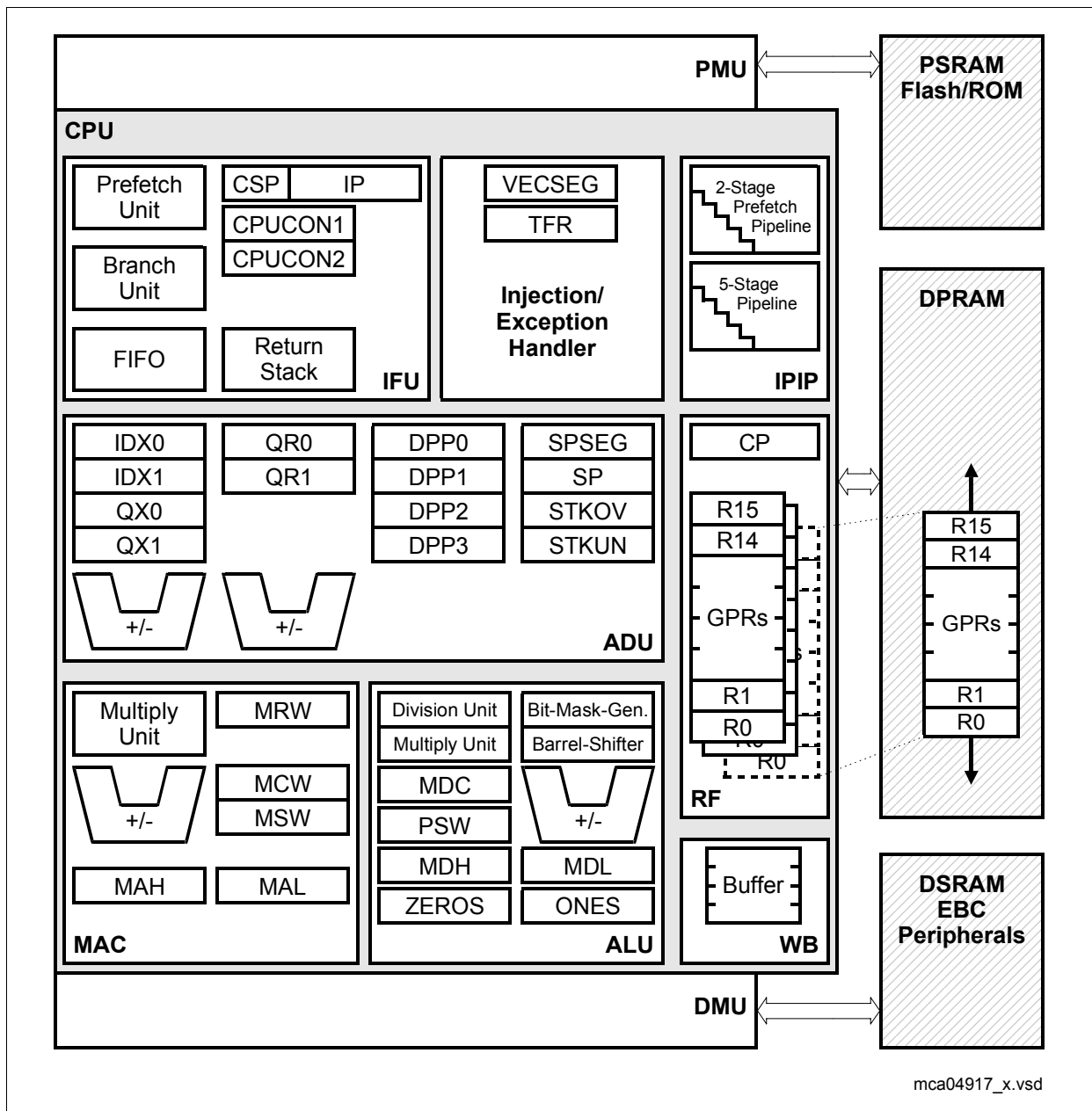


**Figure 2-1 XE16xyM Functional Block Diagram**

## 2.1 Basic CPU Concepts and Optimizations

The main core of the CPU consists of a set of optimized functional units including the instruction fetch/processing pipelines, a 16-bit Arithmetic and Logic Unit (ALU), a 40-bit Multiply and Accumulate Unit (MAC), an Address and Data Unit (ADU), an Instruction Fetch Unit (IFU), a Register File (RF), and dedicated Special Function Registers (SFRs).

Single clock cycle execution of instructions results in superior CPU performance, while maintaining C166 code compatibility. Impressive DSP performance, concurrent access to different kinds of memories and peripherals boost the overall system performance.



**Figure 2-2 CPU Block Diagram**

**Summary of CPU Features**

- Opcode fully upward compatible with C166 Family
- 2-stage instruction fetch pipeline with FIFO for instruction pre-fetching
- 5-stage instruction execution pipeline
- Pipeline forwarding controls data dependencies in hardware
- Multiple high bandwidth buses for data and instructions
- Linear address space for code and data (von Neumann architecture)
- Nearly all instructions executed in one CPU clock cycle
- Fast multiplication (16-bit  $\times$  16-bit) in one CPU clock cycle
- Fast background execution of division (32-bit/16-bit) in 21 CPU clock cycles
- Built-in advanced MAC (Multiply Accumulate) Unit:
  - Single cycle MAC instruction with zero cycle latency including a 16  $\times$  16 multiplier
  - 32-bit addition and 32-bit subtraction
  - 40-bit barrel shifter and 40-bit accumulator to handle overflows
  - Automatic saturation to 32 bits or rounding included with the MAC instruction
  - Fractional numbers supported directly
  - One Finite Impulse Response Filter (FIR) tap per cycle with no circular buffer management
- Enhanced boolean bit manipulation facilities
- High performance branch-, call-, and loop-processing
- Zero cycle jump execution
- Register-based design with multiple variable register banks (byte or word operands)
- Two additional fast register banks
- Variable stack with automatic stack overflow/underflow detection
- “Fast interrupt” and “Fast context switch” features
- Built-in memory protection unit (MPU)

The high performance and flexibility of the CPU is achieved by a number of optimized functional blocks (see [Figure 2-2](#)). Optimizations of the functional blocks are described in detail in the following sections.

**2.1.1 Memory Protection Unit (MPU)**

The Memory Protection Unit (MPU) provides the hardware mechanisms needed for implementing memory protection. The MPU allows detection of unauthorized accesses (read, write or instruction fetch) in user-defined memory ranges. It offers protection for the complete address space, including the peripheral area.

The MPU can be used to support the encapsulation of different applications or software components running on the processor. This encapsulation provides the means to ensure integrity and fault isolation capabilities in today's complex systems relying on multiple-sources software.

## **2.1.2 High Instruction Bandwidth/Fast Execution**

Based on the hardware provisions, most of the XE16xyM's instructions can be executed in just one clock cycle ( $1/f_{SYS}$ ). This includes arithmetic instructions, logic instructions, and move instructions with most addressing modes.

Special instructions such as JMPS take more than one machine cycle. Divide instructions are mainly executed in the background, so other instructions can be executed in parallel. Due to the prediction mechanism (see [Section 5.2](#)), correctly predicted branch instructions require only one cycle or can even be overlaid with another instruction (zero-cycle jump).

The instruction cycle time is dramatically reduced through the use of instruction pipelining. This technique allows the core CPU to process portions of multiple sequential instruction stages in parallel. Up to seven stages can operate in parallel:

**The two-stage instruction fetch pipeline** fetches and preprocesses instructions from the respective program memory:

**PREFETCH:** Instructions are prefetched from the PMU in the predicted order. The instructions are preprocessed in the branch detection unit to detect branches. The prediction logic determines if branches are assumed to be taken or not.

**FETCH:** The instruction pointer for the next instruction to be fetched is calculated according to the branch prediction rules. The branch folding unit preprocesses detected branches and combines them with the preceding instructions to enable zero-cycle branch execution. Prefetched instructions are stored in the instruction FIFO, while stored instructions are moved from the instruction FIFO to the instruction processing pipeline.

**The five-stage instruction processing pipeline** executes the respective instructions:

**DECODE:** The previously fetched instruction is decoded and the GPR used for indirect addressing is read from the register file, if required.

**ADDRESS:** All operand addresses are calculated. For instructions implicitly accessing the stack the stack pointer (SP) is decremented or incremented.

**MEMORY:** All required operands are fetched.

**EXECUTE:** The specified operation (ALU or MAC) is performed on the previously fetched operands. The condition flags are updated. Explicit write operations to CPU-SFRs are executed. GPRs used for indirect addressing are incremented or decremented, if required.

**WRITE BACK:** The result operands are written to the specified locations. Operands located in the DPRAM are stored via the write-back buffer.

### **2.1.3 Powerful Execution Units**

**The 16-bit Arithmetic and Logic Unit (ALU)** performs all standard (word) arithmetic and logical operations. Additionally, for byte operations, signals are provided from bits 6 and 7 of the ALU result to set the condition flags correctly. Multiple precision arithmetic is provided through a 'CARRY-IN' signal to the ALU from previously calculated portions of the desired operation.

Most internal execution blocks have been optimized to perform operations on either 8-bit or 16-bit quantities. Instructions have been provided as well to allow byte packing in memory while providing sign extension of bytes for word wide arithmetic operations. The internal bus structure also allows transfers of bytes or words to or from peripherals based on the peripheral requirements.

A set of consistent flags is updated automatically in the PSW after each arithmetic, logical, shift, or movement operation. These flags allow branching on specific conditions. Support for both signed and unsigned arithmetic is provided through user-specifiable branch tests. These flags are also preserved automatically by the CPU upon entry into an interrupt or trap routine.

A 16-bit barrel shifter provides multiple bit shifts in a single cycle. Rotates and arithmetic shifts are also supported.

**The Multiply and Accumulate Unit (MAC)** performs extended arithmetic operations such as 32-bit addition, 32-bit subtraction, and single-cycle 16-bit  $\times$  16-bit multiplication. The combined MAC operations (multiplication with cumulative addition/subtraction) represent the major part of the DSP performance of the CPU.

**The Address Data Unit (ADU)** contains two independent arithmetic units to generate, calculate, and update addresses for data accesses. The ADU performs the following major tasks:

- The Standard Address Unit supports linear arithmetic for the short, long, and indirect addressing modes. It also supports data paging and stack handling.
- The DSP Address Generation Unit contains an additional set of address pointers and offset registers which are used in conjunction with the CoXXX instructions only.

The CPU provides a lot of powerful addressing modes for word, byte, and bit data accesses (short, long, indirect). The different addressing modes use different formats and have different scopes.

Dedicated bit processing instructions provide efficient control and testing of peripherals while enhancing data manipulation. These instructions provide direct access to two operands in the bit-addressable space without requiring them to be moved into temporary flags. Logical instructions allow the user to compare and modify a control bit for a peripheral in one instruction. Multiple bit shift instructions (single cycle execution) avoid long instruction streams of single bit shift operations. Bitfield instructions allow the modification of multiple bits from one operand in a single instruction.



### **2.1.4 High Performance Branch-, Call-, and Loop-Processing**

Pipelined execution delivers maximum performance with a stream of subsequent instructions. Any disruption requires the pipeline to be refilled and the new instruction to step through the pipeline stages. Due to the high percentage of branching in controller applications, branch instructions have been optimized to require pipeline refilling only in special cases. This is realized by detecting and preprocessing branch instructions in the prefetch stage and by predicting the respective branch target address.

Prefetching then continues from the predicted target address. If the prediction was correct subsequent instructions can be fed to the execution pipeline without a gap, even if a branch is executed, i.e. the code execution is not linear. Branch target prediction (see also [Section 5.2.1](#)) uses the following rules:

- **Unconditional branches:** Branch prediction is trivial in this case, as the branches will always be taken and the target address is defined. This applies to implicitly unconditional branches such as JMPS, CALLR, or RET as well as to branches with condition code “unconditional” such as JMPI cc\_UC.
- **Fixed prediction:** Branch instructions which are often used to realize loops are assumed to be taken if they branch backward to a previous location (the begin of the loop). This applies to conditional branches such as JMPR cc\_XX or JNB.
- **Variable prediction:** In this case the respective prediction (taken or not taken) is coded into the instruction and can, therefore, be selected for each individual branch instruction. Thus, the software designer can optimize the instruction flow to the specific code to be executed<sup>1)</sup>. This applies to the branch instructions JMPA and CALLA.
- **Conditional indirect branches:** These branches are always assumed to be not taken. This applies to branch instructions JMPI cc\_XX, [Rw] and CALLI cc\_XX, [Rw].

The system state information is saved automatically on the internal system stack, thus avoiding the use of instructions to preserve state upon entry and exit of interrupt or trap routines. Call instructions push the value of the IP on the system stack, and require the same execution time as branch instructions. Additionally, instructions have been provided to support indirect branch and call instructions. This feature supports implementation of multiple CASE statement branching in assembler macros and high level languages.

<sup>1)</sup> The programming tools accept either dedicated mnemonics for each prediction leaving the choice up to programmer, or they accept generic mnemonics and apply their own prediction rules.

### **2.1.5 Consistent and Optimized Instruction Formats**

To obtain optimum performance in a pipelined design, an instruction set has been designed which incorporates concepts from Reduced Instruction Set Computing (RISC). These concepts primarily allow fast decoding of the instructions and operands while reducing pipeline holds. These concepts, however, do not preclude the use of complex instructions required by microcontroller users. The instruction set was designed to meet the following goals:

- Provide powerful instructions for frequently-performed operations which traditionally have required sequences of instructions. Avoid transfer into and out of temporary registers such as accumulators and carry bits. Perform tasks in parallel such as saving state upon entry into interrupt routines or subroutines.
- Avoid complex encoding schemes by placing operands in consistent fields for each instruction and avoid complex addressing modes which are not frequently used. Consequently, the instruction decode time decreases and the development of compilers and assemblers is simplified.
- Provide most frequently used instructions with one-word instruction formats. All other instructions use two-word formats. This allows all instructions to be placed on word boundaries: this alleviates the need for complex alignment hardware. It also has the benefit of increasing the range for relative branching instructions.

The high performance of the CPU-hardware can be utilized efficiently by a programmer by means of the highly functional XE16xyM instruction set which includes the following instruction classes:

- Arithmetic Instructions
- DSP Instructions
- Logical Instructions
- Boolean Bit Manipulation Instructions
- Compare and Loop Control Instructions
- Shift and Rotate Instructions
- Prioritize Instruction
- Data Movement Instructions
- System Stack Instructions
- Jump and Call Instructions
- Return Instructions
- System Control Instructions
- Miscellaneous Instructions

Possible operand types are bits, bytes, words, and doublewords. Specific instructions support the conversion (extension) of bytes to words. Various direct, indirect, and immediate addressing modes are provided to specify the required operands.

## **2.1.6 Programmable Multiple Priority Interrupt System**

The XE16xyM provides 96 separate interrupt nodes that may be assigned to 16 priority levels with 8 group priorities on each level. 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.

The following enhancements within the XE16xyM allow processing of a large number of interrupt sources:

- **Peripheral Event Controller (PEC):** This processor is used to off-load many interrupt requests from the CPU. It avoids the overhead of entering and exiting interrupt or trap routines by performing single-cycle interrupt-driven byte or word data transfers between any two locations with an optional increment of the PEC source pointer, the destination pointer, or both. Only one cycle is 'stolen' from the current CPU activity to perform a PEC service.
- **Multiple Priority Interrupt Controller:** This controller allows all interrupts to be assigned any specified priority. Interrupts may also be grouped, which enables the user to prevent similar priority tasks from interrupting each other. For each of the interrupt nodes, there is a separate control register which contains an interrupt request flag, an interrupt enable flag, and an interrupt priority bitfield. After being accepted by the CPU, an interrupt service can be interrupted only by a higher prioritized service request. For standard interrupt processing, each of the interrupt nodes has a dedicated vector location.
- **Multiple Register Banks:** Two local register banks for immediate context switching add to a relocatable global register bank. The user can specify several register banks located anywhere in the internal DPRAM and made of up to sixteen general purpose registers. A single instruction switches from one register bank to another (switching banks flushes the pipeline, changing the global bank requires a validation sequence).

The XE16xyM is capable of reacting very quickly to non-deterministic events because its interrupt response time is within a very narrow range of typically 7 clock cycles (in the case of internal program execution). Its fast external interrupt inputs are sampled every clock cycle and allow even very short external signals to be recognized.

The XE16xyM also provides an excellent mechanism to identify and process exceptions or error conditions that arise during run-time, so called 'Hardware Traps'. A hardware trap causes an immediate non-maskable system reaction which is similar to a standard interrupt service (branching to a dedicated vector table location). The occurrence of a hardware trap is additionally signified by an individual bit in the trap flag register (TFR). Unless another, higher prioritized, trap service is in progress, a hardware trap will interrupt any current program execution. In turn, a hardware trap service can normally not be interrupted by a standard or PEC interrupt.

Software interrupts are supported by means of the 'TRAP' instruction in combination with an individual trap (interrupt) number.

### **2.1.7 Interfaces to System Resources**

The CPU of the XE16xyM interfaces to the system resources via several bus systems which contribute to the overall performance by transferring data concurrently. This avoids stalling the CPU because instructions or operands need to be transferred.

The Dual Port RAM (DPRAM) is directly coupled to the CPU because it houses the global register banks. Transfers from/to these locations affect the performance and are, therefore, carefully optimized.

**The Program Management Unit (PMU)** controls accesses to the on-chip program memory blocks such as the ROM/Flash module and the Program/Data RAM (PSRAM) and also fetches instructions from external memory.

The 64-bit interface between the PMU and the CPU delivers the instruction words, which are requested by the CPU. The PMU decides whether the requested instruction word has to be fetched from on-chip memory or from external memory.

**The Data Management Unit (DMU)** controls accesses to the on-chip Data RAM (DSRAM), to the on-chip peripherals connected to the peripheral bus, and to resources on the external bus. External accesses (including accesses to peripherals connected to the on-chip LxBus) are executed by the External Bus Controller (EBC).

The 16-bit interface between the DMU and the CPU handles all data transfers (operands). Data accesses by the CPU are distributed to the appropriate buses according to the defined address map.

PMU and DMU are directly coupled to perform cross-over transfers with high speed. Crossover transfers are executed in both directions:

- **PMU via DMU:** Code fetches from external locations are redirected via the DMU to EBC. Thus, the XE16xyM can execute code from external resources. No code can be fetched from the Data RAM (DSRAM).
- **DMU via IMB:** Data accesses can also be executed to on-chip resources normally controlled by the PMU. This includes the following types of transfers:
  - Read a constant from the on-chip program ROM/Flash
  - Read data from the on-chip PSRAM
  - Write data to the on-chip PSRAM (required prior to executing out of it)
  - Program/Erase the on-chip Flash memory

## **2.2 On-Chip System Resources**

The XE16xyM controllers provide a number of powerful system resources designed around the CPU. The combination of CPU and these resources results in the high performance of the members of this controller family.

### **Peripheral Event Controller (PEC) and Interrupt Control**

The Peripheral Event Controller enables response to an interrupt request with a single data transfer (word or byte) which consumes only one instruction cycle and does not require saving and restoring the machine status. Each interrupt source is prioritized for every machine cycle in the interrupt control block. If PEC service is selected, a PEC transfer is started. If CPU interrupt service is requested, the current CPU priority level stored in the PSW register is tested to determine whether a higher priority interrupt is currently being serviced. When an interrupt is acknowledged, the current state of the machine is saved on the internal system stack and the CPU branches to the system specific vector for the peripheral.

The PEC contains a set of SFRs which store the count value and control bits for eight data transfer channels. In addition, the PEC uses a dedicated area of RAM which contains the source and destination addresses. The PEC is controlled in a manner similar to any other peripheral: through SFRs containing the desired configuration of each channel.

An individual PEC transfer counter is implicitly decremented for each PEC service except in the continuous transfer mode. When this counter reaches zero, a standard interrupt is performed to the vector location related to the corresponding source. PEC services are very well suited, for example, to moving register contents to/from a memory table. The XE16xyM has eight PEC channels, each of which offers such fast interrupt-driven data transfer capabilities.

### **Memory Areas**

The memory space of the XE16xyM is configured in a Von Neumann architecture. This means that code memory, data memory, registers, and IO ports are organized within the same linear address space which covers up to 16 Mbytes. The entire memory space can be accessed byte-wise or word-wise. Particular portions of the on-chip memory have been made directly bit addressable as well.

*Note: The actual memory sizes depend on the selected device type. This overview describes the maximum block sizes.*

**Architectural Overview**

**Up to 576 Kbytes of on-chip Flash memory** store code or constant data. The on-chip Flash memory consists of 2 or 3 Flash modules, each built up from 4-Kbyte sectors. Each sector can be separately write protected<sup>1)</sup>, erased and programmed (in blocks of 128 bytes). The complete Flash area can be read-protected. A user-defined password sequence temporarily unlocks protected areas. The Flash modules combine 128-bit read accesses with protected and efficient writing algorithms for programming and erasing. Dynamic error correction provides extremely high read data security for all read accesses. Accesses to different Flash modules can be executed in parallel.

*Note: Program execution from on-chip program memory is the fastest of all possible alternatives and results in maximum performance. The size of the on-chip program memory depends on the chosen derivative. On-chip program memory also includes the PSRAM.*

**Up to 32 Kbytes of on-chip Program SRAM (PSRAM)** are provided to store user code or data. The PSRAM is accessed via the PMU and is, therefore, optimized for code fetches. A section of the PSRAM with programmable size can be write-protected.

**Up to 16 Kbytes of on-chip Data SRAM (DSRAM)** are provided as a storage for general user data. The DSRAM is accessed via a separate interface and is, therefore, optimized for data accesses.

**2 Kbytes of on-chip Dual-Port RAM (DPRAM)** are provided as a storage for user defined variables, for the system stack, and in particular for general purpose register banks. A register bank can consist of up to 16 wordwide (R0 to R15) and/or bytewise (RL0, RH0, ..., RL7, RH7) so-called General Purpose Registers (GPRs).

The upper 256 bytes of the DPRAM are directly bitaddressable. When used by a GPR, any location in the DPRAM is bitaddressable.

**8 Kbytes of on-chip Stand-By SRAM (SBRAM)** is provided as a storage for system-relevant user data that must be preserved while the major part of the device is powered down. The SBRAM is accessed via a specific interface and is powered via domain M.

The CPU has an actual register context of up to 16 wordwide and/or bytewise global GPRs at its disposal, which are physically located within the on-chip RAM area. A Context Pointer (CP) register determines the base address of the active global register bank to be accessed by the CPU at a time. The number of register banks is restricted only by the available internal RAM space. For easy parameter passing, a register bank may overlap other register banks.

A system stack of up to 32 Kwords is provided as storage for temporary data. The system stack can be located anywhere within the complete addressing range and it is accessed by the CPU via the Stack Pointer (SP) register and the Stack Pointer Segment (SPSEG) register. Two separate SFRs, STKOV and STKUN, are implicitly compared against the stack pointer value upon each stack access for the detection of a stack overflow or

<sup>1)</sup> To save control bits, sectors are clustered for protection purposes, they remain separate for programming/erasing.



## Architectural Overview

underflow. This mechanism also supports the control of a bigger virtual stack. Maximum performance for stack operations is achieved by allocating the system stack to internal data RAM areas (DPRAM, DSRAM).

Hardware detection of the selected memory space is placed at the internal memory decoders and allows the user to specify any address directly or indirectly and obtain the desired data without using temporary registers or special instructions.

**For Special Function Registers** three areas of the address space are reserved: The standard Special Function Register area (SFR) uses 512 bytes, while the Extended Special Function Register area (ESFR) uses the other 512 bytes. A range of 4 Kbytes is provided for the internal IO area (XSFR). SFRs are wordwide registers which are used for controlling and monitoring functions of the different on-chip units. Unused SFR addresses are reserved for future members of the XE166 Family with enhanced functionality. Therefore, they should either not be accessed, or written with zeros, to ensure upward compatibility.

In order to meet the needs of designs where more memory is required than is provided on chip, up to 12 Mbytes (approximately, see [Table 2-1](#)) of external RAM and/or ROM can be connected to the microcontroller. The External Bus Interface also provides access to external peripherals.

**Table 2-1 XE16xyM Memory Map**

Address Area	Start Loc.	End Loc.	Area Size <sup>1)</sup>	Notes
IMB register space	FF'FF00 <sub>H</sub>	FF'FFFF <sub>H</sub>	256 Bytes	–
Reserved (Access trap)	F0'0000 <sub>H</sub>	FF'FEFF <sub>H</sub>	<1 Mbyte	Minus IMB reg.
Reserved for EPSRAM	E8'8000 <sub>H</sub>	EF'FFFF <sub>H</sub>	480 Kbytes	Mirrors EPSRAM
Emulated PSRAM	E8'0000 <sub>H</sub>	E8'7FFF <sub>H</sub>	32 Kbytes	Flash timing
Reserved for PSRAM	E0'8000 <sub>H</sub>	E7'FFFF <sub>H</sub>	480 Kbytes	Mirrors PSRAM
Program SRAM	E0'0000 <sub>H</sub>	E0'7FFF <sub>H</sub>	32 Kbytes	Maximum speed
Reserved for pr. mem.	CC'0000 <sub>H</sub>	DF'FFFF <sub>H</sub>	<1.25 Mbytes	–
Program Flash 2	C8'0000 <sub>H</sub>	CB'FFFF <sub>H</sub>	256 Kbytes	–
Program Flash 1	C4'0000 <sub>H</sub>	C7'FFFF <sub>H</sub>	256 Kbytes	–
Program Flash 0	C0'0000 <sub>H</sub>	C3'FFFF <sub>H</sub>	256 Kbytes	<sup>2)</sup>
External memory area	40'0000 <sub>H</sub>	BF'FFFF <sub>H</sub>	8 Mbytes	–
Available Ext. IO area <sup>3)</sup>	20'C000 <sub>H</sub>	3F'FFFF <sub>H</sub>	< 2 Mbytes	Minus USIC/CAN
MultiCAN/USIC regs.	20'8000 <sub>H</sub>	20'BFFF <sub>H</sub>	16 Kbytes	Alternate location
Available Ext. IO area <sup>3)</sup>	20'6000 <sub>H</sub>	20'7FFF <sub>H</sub>	8Kbytes	–
USIC registers	20'4000 <sub>H</sub>	20'5FFF <sub>H</sub>	8 Kbytes	Accessed via EBC

**Table 2-1 XE16xyM Memory Map (cont'd)**

Address Area	Start Loc.	End Loc.	Area Size <sup>1)</sup>	Notes
MultiCAN registers	20'0000 <sub>H</sub>	20'3FFF <sub>H</sub>	16 Kbytes	Accessed via EBC
External memory area	01'0000 <sub>H</sub>	1F'FFFF <sub>H</sub>	< 2 Mbytes	Minus segment 0
SFR area	00'FE00 <sub>H</sub>	00'FFFF <sub>H</sub>	0.5 Kbyte	—
Dual-Port RAM	00'F600 <sub>H</sub>	00'FDFF <sub>H</sub>	2 Kbytes	—
Reserved for DPRAM	00'F200 <sub>H</sub>	00'F5FF <sub>H</sub>	1 Kbyte	—
ESFR area	00'F000 <sub>H</sub>	00'F1FF <sub>H</sub>	0.5 Kbyte	—
XSFR area	00'E000 <sub>H</sub>	00'FFFF <sub>H</sub>	4 Kbytes	—
Data SRAM	00'A000 <sub>H</sub>	00'DFFF <sub>H</sub>	16 Kbytes	—
Reserved for DSRAM	00'8000 <sub>H</sub>	00'9FFF <sub>H</sub>	8 Kbytes	—
External memory area	00'0000 <sub>H</sub>	00'7FFF <sub>H</sub>	32 Kbytes	—

<sup>1)</sup> The areas marked with "<" are slightly smaller than indicated, see column "Notes".

<sup>2)</sup> The uppermost 4-Kbyte sector of the first Flash segment is reserved for internal use (C0'F000<sub>H</sub> to C0'FFFF<sub>H</sub>).

<sup>3)</sup> Several pipeline optimizations are not active within the external IO area. This is necessary to control external peripherals properly.

*Note: For an overview of the available memory sections for the different derivatives, please refer to the corresponding Data Sheets.*



## **External Bus Interface**

To meet the needs of designs where more memory is required than is provided on chip, up to 12 Mbytes of external RAM/ROM/Flash or peripherals can be connected to the XE16xyM microcontroller via its external bus interface.

All of the external memory accesses are performed by a particular on-chip External Bus Controller (EBC). It can be programmed either to Single Chip Mode when no external memory is required, or to an external bus mode with the following possible selections<sup>1)</sup>:

- Address Bus Width with a range of 0 ... 24-bit
- Data Bus Width 8-bit or 16-bit
- Bus Operation Multiplexed or Demultiplexed

In the demultiplexed bus modes, addresses are output on Port 0 and Port 1 and data is input/output on Port 10 and Port 2. In the multiplexed bus modes both addresses and data use Port 10 and Port 2 for input/output. The high order address (segment) lines use Port 2. The number of active address lines is selectable, so the external address space can be restricted. This is required when interface lines are assigned to Port 2.

For up to five address areas the bus mode (multiplexed/demultiplexed), the data bus width (8-bit/16-bit) and even the length of a bus cycle (waitstates, signal delays) can be selected independently. This allows access to a variety of memory and peripheral components directly and with maximum efficiency.

Access to very slow memories or modules with varying access times is supported via a particular 'Ready' function. The active level of the control input signal is selectable.

A HOLD/HLDA protocol is available for bus arbitration and allows the sharing of external resources with other bus masters.

The external bus timing is related to the rising edge of the reference clock output CLKOUT. The external bus protocol is compatible with that of the standard C166 Family.

For applications which require less than 64 Kbytes of address space, a non-segmented memory model can be selected, where all locations can be addressed by 16 bits. Thus, the upper Port 2 is not needed as an output for the upper address bits (Axx ... A16), as is the case when using the segmented memory model.

The EBC also controls accesses to resources connected to the **on-chip LXBus**. The LXBus is an internal representation of the external bus and allows accessing integrated peripherals and modules in the same way as external components.

The MultiCAN module and the USIC modules are connected to and accessed via the LXBus.

*Note: The external bus interface is not available in the 64-pin types XE162xM.*

<sup>1)</sup> Bus modes are switched dynamically if several address windows with different mode settings are used.

## **2.3 On-Chip Peripheral Blocks**

The XE166 Family clearly separates peripherals from the core. This structure permits the maximum number of operations to be performed in parallel and allows peripherals to be added or deleted from family members without modifications to the core. Each functional block processes data independently and communicates information over common buses. Peripherals are controlled by data written to the respective Special Function Registers (SFRs). These SFRs are located within either the standard SFR area (00'FE00<sub>H</sub> ... 00'FFFF<sub>H</sub>), the extended ESFR area (00'F000<sub>H</sub> ... 00'F1FF<sub>H</sub>), or within the internal IO area (00'E000<sub>H</sub> ... 00'FFFF<sub>H</sub>).

These built-in peripherals either allow the CPU to interface with the external world or provide functions on-chip that otherwise would need to be added externally in the respective system.

The XE16xyM generic peripherals are:

- **Memory Checker Unit**
- **General Purpose Timer Unit (GPT1, GPT2)**
- **Watchdog Timer**
- **Capture/Compare Unit (CAPCOM2)**
- Up to Four **Capture/Compare Units CCU6** (CCU60, CCU61, CCU62, CCU63)
- Two 10-bit **Analog/Digital Converters (ADC0, ADC1)**
- **Real Time Clock (RTC)**
- Thirteen/Nine **Parallel Ports** with a total of 119/76 I/O lines

Because the LXBus is the internal representation of the external bus, it does not support bit-addressing. Accesses are executed by the EBC as if it were external accesses. The LXBus connects on-chip peripherals to the CPU:

- Optional **MultiCAN Module** with up to 6 CAN nodes and gateway functionality
- Up to Four **Universal Serial Interface Channel Modules (USIC)**

Each peripheral also contains a set of Special Function Registers (SFRs) which control the functionality of the peripheral and temporarily store intermediate data results. Each peripheral has an associated set of status flags. Individually selected clock signals are generated for each peripheral from binary multiples of the master clock.

*Note: For an overview of the available peripherals for the different derivatives, please refer to **"Summary of Basic Features" on Page 1-5.***

## **Peripheral Interfaces**

The on-chip peripherals generally have two different types of interfaces: a bus interface to the CPU and interface signals to other on-chip peripherals or to external hardware. Communication between the CPU and peripherals is performed through Special Function Registers (SFRs) and interrupts. The SFRs serve as control/status and data registers for the peripherals. Interrupt requests are generated by the peripherals based on specific events which occur during their operation, such as operation complete, error, etc.

To interface with external hardware, specific pins of the parallel ports are used, when an input or output function has been selected for a peripheral. During this time, the port pins are controlled either by the peripheral (when used as outputs) or by the external hardware which controls the peripheral (when used as inputs). This is called the 'alternate (input or output) function' of a port pin, in contrast to its function as a general purpose I/O pin.

## **Peripheral Timing**

Internal operation of the CPU and peripherals is based on the system clock ( $f_{SYS}$ ). The clock generation unit uses external (e.g. a crystal) or internal clock sources to generate the system clock signal. Peripherals can be disconnected from the clock signal either temporarily to save energy or permanently if they are not used in a specific application. Peripheral SFRs may be accessed by the CPU once per state. When an SFR is written to by software in the same state where it is also to be modified by the peripheral, the software write operation has priority. Further details on peripheral timing are included in the specific sections describing each peripheral.

## **Programming Hints**

- **Access to SFRs:** The SFRs reside in various data pages of the memory space. The following addressing mechanisms allow access to the SFRs:
  - Indirect or direct addressing with **16-bit (mem) addresses** must guarantee that the used data page pointer (DPP0 ... DPP3) selects the corresponding data page.
  - Accesses via the Peripheral Event Controller (**PEC**) use the SRCPx and DSTPx pointers instead of the data page pointers.
  - **Short 8-bit (reg) addresses** to the standard SFR area do not use the data page pointers but directly access the registers within this 512-byte area.
  - **Short 8-bit (reg) addresses** to the extended **ESFR** area require switching to the 512-byte Extended SFR area. This is done via the EXTension instructions EXTR, EXTP(R), EXTS(R).
- **Byte Write Operations** to wordwide SFRs via indirect or direct 16-bit (mem) addressing or byte transfers via the PEC force zeros in the non-addressed byte. Byte write operations via short 8-bit (reg) addressing can access only the low byte of an SFR and force zeros in the high byte. It is therefore recommended, to use the bitfield

## **Architectural Overview**

instructions (BFLDL and BFLDH) to write to any number of bits in either byte of an SFR without disturbing the non-addressed byte and the unselected bits.

- **Write Operations to Write-Only Bits/Registers** usually modify bits within other registers. In some cases this modification is controlled by state machines. Therefore, the effect of the write operation may not be visible, when the modified register is read immediately after the write access that triggers the modification.
- **Reserved Bits:** Some of the bits which are contained in the XE16xyM's SFRs are marked as 'Reserved'. User software should never write '1's to reserved bits. These bits are currently not implemented and may be used in future products to invoke new functions. In that case, the active state for those new functions will be '1', and the inactive state will be '0'. Therefore writing only '0's to reserved locations allows portability of the current software to future devices. After read accesses, reserved bits should be ignored or masked out.

### **Capture/Compare Unit (CAPCOM2)**

The CAPCOM units support generation and control of timing sequences on up to 16 channels with a maximum resolution of 1 system clock cycle (8 cycles in staggered mode). The CAPCOM unit is typically used to handle high speed I/O tasks such as pulse and waveform generation, pulse width modulation (PMW), Digital to Analog (D/A) conversion, software timing, or time recording relative to external events.

Two 16-bit timers (T7/T8) with reload registers provide two independent time bases for each capture/compare register.

The input clock for the timers is programmable to several prescaled values of the internal system clock, or may be derived from an overflow/underflow of timer T6 in module GPT2. This provides a wide range of variation for the timer period and resolution and allows precise adjustments to the application specific requirements. In addition, external count inputs for CAPCOM timer T7 allow event scheduling for the capture/compare registers relative to external events.

The capture/compare register array contains 16 dual purpose capture/compare registers, each of which may be individually allocated to either CAPCOM timer T7 or T8 and programmed for capture or compare function.

All registers of each module have each one port pin associated with it which serves as an input pin for triggering the capture function, or as an output pin to indicate the occurrence of a compare event.

**Table 2-2      Compare Modes (CAPCOM2)**

<b>Compare Modes</b>	<b>Function</b>
Mode 0	Interrupt-only compare mode; several compare interrupts per timer period are possible
Mode 1	Pin toggles on each compare match; several compare events per timer period are possible
Mode 2	Interrupt-only compare mode; only one compare interrupt per timer period is generated
Mode 3	Pin set '1' on match; pin reset '0' on compare timer overflow; only one compare event per timer period is generated
Double Register Mode	Two registers operate on one pin; pin toggles on each compare match; several compare events per timer period are possible
Single Event Mode	Generates single edges or pulses; can be used with any compare mode

When a capture/compare register has been selected for capture mode, the current contents of the allocated timer will be latched ('captured') into the capture/compare register in response to an external event at the port pin which is associated with this register. In addition, a specific interrupt request for this capture/compare register is generated. Either a positive, a negative, or both a positive and a negative external signal transition at the pin can be selected as the triggering event.

The contents of all registers which have been selected for one of the five compare modes are continuously compared with the contents of the allocated timers.

When a match occurs between the timer value and the value in a capture/compare register, specific actions will be taken based on the selected compare mode.

### **Capture/Compare Units CCU6**

The CCU6 units support generation and control of timing sequences on up to three 16-bit capture/compare channels plus one independent 16-bit compare channel.

In compare mode, the CCU6 units provide two output signals per channel which have inverted polarity and non-overlapping pulse transitions (deadtime control). The compare channel can generate a single PWM output signal and is further used to modulate the capture/compare output signals.

In capture mode the contents of compare timer T12 is stored in the capture registers upon a signal transition at pins CCx.

The output signals can be generated in edge-aligned or center-aligned PWM mode. They are generated continuously or in single-shot mode.

Compare timers T12 and T13 are free running timers which are clocked by the prescaled system clock.

For motor control applications (brushless DC-drives) both subunits may generate versatile multichannel PWM signals which are basically either controlled by compare timer T12 or by a typical hall sensor pattern at the interrupt inputs (block commutation). The latter mode provides noise filtering for the hall inputs and supports automatic rotational speed measurement.

The trap function offers a fast emergency stop without CPU activity. Triggered by an external signal ( $\overline{\text{CTRAP}}$ ) the outputs are switched to selectable logic levels which can be adapted to the connected power stages.

*Note: The number of available CCU6 units and channels depends on the selected device type.*

## **General Purpose Timer Unit (GPT1, GPT2)**

The GPT12E unit represents a very flexible multifunctional timer/counter structure which may be used for many different time related tasks such as event timing and counting, pulse width and duty cycle measurements, pulse generation, or pulse multiplication.

The GPT12E unit incorporates five 16-bit timers which are organized in two separate blocks, GPT1 and GPT2. Each timer in each block may operate independently in a number of different modes, or may be concatenated with another timer of the same block.

Each of the three timers T2, T3, T4 of **block GPT1** can be configured individually for one of four basic modes of operation, which are Timer, Gated Timer, Counter, and Incremental Interface Mode. In Timer Mode, the input clock for a timer is derived from the system clock, divided by a programmable prescaler, while Counter Mode allows a timer to be clocked in reference to external events.

Pulse width or duty cycle measurement is supported in Gated Timer Mode, where the operation of a timer is controlled by the 'gate' level on an external input pin. For these purposes, each timer has one associated port pin (TxIN) which serves as gate or clock input. The maximum resolution of the timers in block GPT1 is 4 system clock cycles.

The count direction (up/down) for each timer is programmable by software or may additionally be altered dynamically by an external signal on a port pin (TxEUD) to facilitate e.g. position tracking.

In Incremental Interface Mode the GPT1 timers (T2, T3, T4) can be directly connected to the incremental position sensor signals A and B via their respective inputs TxIN and TxEUD. Direction and count signals are internally derived from these two input signals, so the contents of the respective timer Tx corresponds to the sensor position. The third position sensor signal TOP0 can be connected to an interrupt input.

Timer T3 has an output toggle latch (T3OTL) which changes its state on each timer overflow/underflow. The state of this latch may be output on pin T3OUT e.g. for time out monitoring of external hardware components. It may also be used internally to clock timers T2 and T4 for measuring long time periods with high resolution.

In addition to their basic operating modes, timers T2 and T4 may be configured as reload or capture registers for timer T3. When used as capture or reload registers, timers T2 and T4 are stopped. The contents of timer T3 is captured into T2 or T4 in response to a signal at their associated input pins (TxIN). Timer T3 is reloaded with the contents of T2 or T4 triggered either by an external signal or by a selectable state transition of its toggle latch T3OTL. When both T2 and T4 are configured to alternately reload T3 on opposite state transitions of T3OTL with the low and high times of a PWM signal, this signal can be constantly generated without software intervention.

With its maximum resolution of 2 system clock cycles, the **GPT2 block** provides precise event control and time measurement. It includes two timers (T5, T6) and a capture/reload register (CAPREL). Both timers can be clocked with an input clock which is



**Architectural Overview**

derived from the CPU clock via a programmable prescaler or with external signals. The count direction (up/down) for each timer is programmable by software or may additionally be altered dynamically by an external signal on a port pin (TxEUD). Concatenation of the timers is supported via the output toggle latch (T6OTL) of timer T6, which changes its state on each timer overflow/underflow.

The state of this latch may be used to clock timer T5, and/or it may be output on pin T6OUT. The overflows/underflows of timer T6 can additionally be used to clock the CAPCOM2 timers, and to cause a reload from the CAPREL register.

The CAPREL register may capture the contents of timer T5 based on an external signal transition on the corresponding port pin (CAPIN), and timer T5 may optionally be cleared after the capture procedure. This allows the XE16xyM to measure absolute time differences or to perform pulse multiplication without software overhead.

The capture trigger (timer T5 to CAPREL) may also be generated upon transitions of GPT1 timer T3's inputs T3IN and/or T3EUD. This is especially advantageous when T3 operates in Incremental Interface Mode.



## **Real Time Clock (RTC)**

The Real Time Clock (RTC) module of the XE16xyM is directly clocked with a separate clock signal. Several internal and external clock sources can be selected via register RTCCLKCON. It is, therefore, independent from the selected clock generation mode of the XE16xyM.

The RTC basically consists of a chain of divider blocks:

- Selectable 32:1 and 8:1 dividers (on - off)
- The reloadable 16-bit timer T14
- The 32-bit RTC timer block (accessible via registers RTCH and RTCL), made of:
  - a reloadable 10-bit timer
  - a reloadable 6-bit timer
  - a reloadable 6-bit timer
  - a reloadable 10-bit timer

All timers count up. Each timer can generate an interrupt request. All requests are combined to a common node request.

*Note: The registers associated with the RTC are not affected by an application reset in order to maintain the contents even when intermediate resets are executed.*

The RTC module can be used for different purposes:

- System clock to determine the current time and date
- Cyclic time based interrupt, to provide a system time tick independent of CPU frequency and other resources
- 48-bit timer for long term measurements

**Analog/Digital Converters (ADC0, ADC1)**

For analog signal measurement, two 10-bit A/D converters (ADC0, ADC1) with 16 (or 8) multiplexed input channels including a sample and hold circuit have been integrated on-chip. Up to 4 inputs can be converted by both A/D converters. Conversions use the method of successive approximation. The sample time (for loading the capacitors) and the conversion time are programmable and can thus be adjusted to the external circuitry. The A/D converters can also operate in 8-bit conversion mode, where the conversion time is further reduced.

Several independent conversion result registers, selectable interrupt requests, and highly flexible conversion sequences provide a high degree of programmability to fulfill the requirements of the respective application. Both modules can be synchronized to allow parallel sampling of two input channels.

For applications that require more analog input channels, external analog multiplexers can be controlled automatically. For applications that require less analog input channels, the remaining channel inputs can be used as digital input port pins.

The A/D converters of the XE16xyM support two types of request sources which can be triggered by several internal and external events.

- Scan requests are activated at the same time and then executed in a predefined sequence.
- Queued requests are executed in a user-defined sequence.

In addition, the conversion of a specific channel can be inserted into a running sequence without disturbing this sequence. All requests are arbitrated according to the priority level that has been assigned to them.

Data reduction features, such as limit checking or result accumulation, reduce the number of required CPU accesses and so allow the precise evaluation of analog inputs (high conversion rate) even at low CPU speed.

The Peripheral Event Controller (PEC) may be used to control the A/D converters or to automatically store conversion results into a table in memory for later evaluation, without requiring the overhead of entering and exiting interrupt routines for each data transfer. Therefore, each A/D converter contains 8 result registers which can be concatenated to build a result FIFO. Wait-for-read mode can be enabled for each result register to prevent loss of conversion data.

In order to decouple analog inputs from digital noise and to avoid input trigger noise those pins used for analog input can be disconnected from the digital input stages under software control. This can be selected for each pin separately via registers P5\_DIDIS and P15\_DIDIS (Port x Digital Input Disable).

The Auto-Power-Down feature of the A/D converters minimizes the power consumption when no conversion is in progress.

*Note: The number of available analog channels depends on the selected device type.*

**Universal Serial Interface Channel Modules (USIC)**

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  $f_{SYS} / 4$
  - number of data bits per data frame 1 to 63
  - MSB or LSB first
- **LIN** Support by HW (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 SW control for higher flexibility
- **SSC/SPI** (synchronous serial channel with or without slave select lines)
  - module capability: maximum baud rate  $f_{SYS} / 2$ , limited by loop delay
  - number of data bits per data frame 1 to 63, more with explicit stop condition
  - 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  $f_{SYS} / 2$

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

**Architectural Overview**

information is generated automatically by analyzing the address where the user SW 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 4 service request outputs, 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 an 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 events generated outside the USIC module, e.g. at an input pin or a timer unit (transmit data validation). This feature allows time base related data transmission.

- **Debugger support**

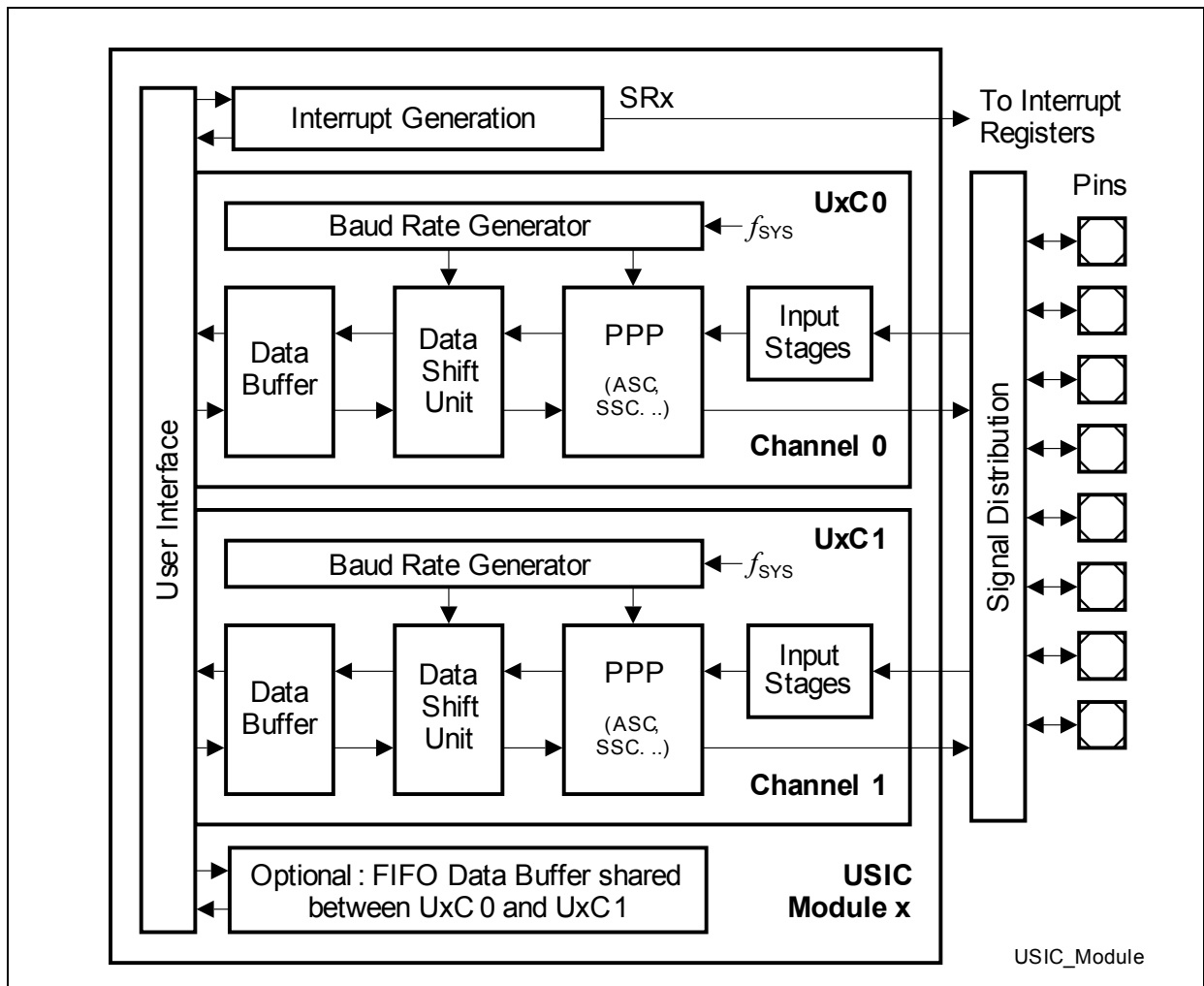
The USIC offers specific addresses to read out received data without interaction with the FIFO buffer mechanism. This feature allows debugger accesses without the risk of a corrupted receive data sequence.

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  $f_{\text{sys}}$ , 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

## Architectural Overview

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.).*



**Figure 2-3 USIC Channel Structure**

The USIC module contains two independent communication channels, with structure shown in [Figure 2-3](#).

The data shift unit and the data buffering of each channel support full-duplex data transfers. The protocol-specific actions are handled by 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. This FIFO

## **Architectural Overview**

data buffer is not necessarily available in all devices (please refer to USIC implementation chapter for details).

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.

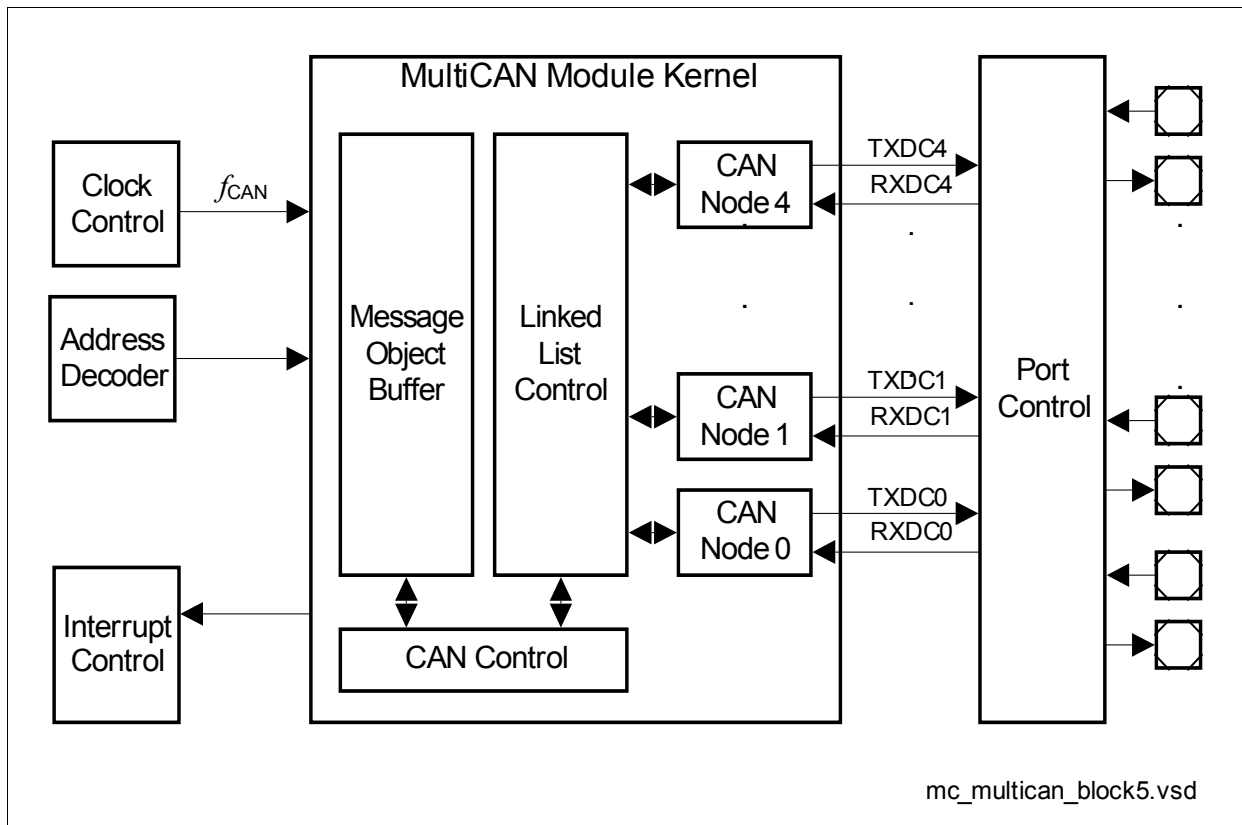
## MultiCAN Module

The MultiCAN module contains up to six independently operating CAN nodes with Full-CAN functionality which are able to exchange Data and Remote Frames via a gateway function. Transmission and reception of CAN frames is handled in accordance with CAN specification V2.0 B (active). Each CAN node can receive and transmit standard frames with 11-bit identifiers as well as extended frames with 29-bit identifiers.

*Note: The number of available CAN nodes depends on the selected device type.*

All CAN nodes share a common set of up to 128 message objects. Each message object can be individually allocated to one of the CAN nodes. Besides serving as a storage container for incoming and outgoing frames, message objects can be combined to build gateways between the CAN nodes or to setup a FIFO buffer.

The message objects are organized in double-chained linked lists, where each CAN node has its own list of message objects. A CAN node stores frames only into message objects that are allocated to its own message object list, and it transmits only messages belonging to this message object list. A powerful, command-driven list controller performs all message object list operations.



**Figure 2-4 Block Diagram of the MultiCAN Module**

### **MultiCAN Features**

- CAN functionality conforms to CAN specification V2.0 B active for each CAN node (compliant to ISO 11898)
- Up to Six independent CAN nodes
- Up to 256 independent message objects (shared by the CAN nodes)
- Dedicated control registers for each CAN node
- Data transfer rate up to 1 Mbit/s, individually programmable for each node
- Flexible and powerful message transfer control and error handling capabilities
- Full-CAN functionality for message objects:
  - Can be assigned to one of the CAN nodes
  - Configurable as transmit or receive objects, or as message buffer FIFO
  - Handle 11-bit or 29-bit identifiers with programmable acceptance mask for filtering
  - Remote Monitoring Mode, and frame counter for monitoring
- Automatic Gateway Mode support
- 16 individually programmable interrupt nodes
- Analyzer mode for CAN bus monitoring



## **Watchdog Timer**

The Watchdog Timer represents one of the fail-safe mechanisms which have been implemented to prevent the controller from malfunctioning for longer periods of time.

The Watchdog Timer is always enabled after a reset of the chip, and can be disabled and enabled at any time by executing instructions DISWDT and ENWDT. Thus, the chip's start-up procedure is always monitored. The software has to be designed to restart the Watchdog Timer before it overflows. If, due to hardware or software related failures, the software fails to do so, the Watchdog Timer overflows and generates a reset request.

The Watchdog Timer is a 16-bit timer, clocked with the system clock divided by 16,384 or 256. The high byte of the Watchdog Timer register can be set to a prespecified reload value (stored in WDTREL) to allow further variation of the monitored time interval. Each time it is serviced by the application software, the high byte of the Watchdog Timer is reloaded and the low byte is cleared.

Thus, time intervals between 3.2  $\mu$ s and 13.4 s can be monitored (@ 80 MHz).

The default Watchdog Timer interval after reset is 6.5 ms (@ 10 MHz).

## **Memory Checker Unit**

The memory checker module (MCHK) of the XE16xyM supports checking the data consistency of memories, registers (e.g. configuration registers), or communication channels. It calculates a checksum on a block of data, often called cyclic redundancy code (CRC). It is implemented as a parallel signature generation based on a multi input linear feedback shift register (MISR). Being based on a linear feedback shift register (LFSR), it also can generate pseudo-random numbers and cyclic codes.

From the programmer's point of view, the MCHK is a set of registers associated with this peripheral. To communicate respective error or operation events a port pin may be used for the signal "MATCH" to generate an external event and an interrupt line may be used for the signal "MISMATCH" to generate an internal event.

## Parallel Ports

The XE16xyM derivatives are available in two different packages:

- In LQFP-144, they provide up to 119 I/O lines which are organized into 11 input/output ports and 2 input ports.
- In LQFP-100, they provide up to 76 I/O lines which are organized into 7 input/output ports and 2 input ports.
- In LQFP-64, they provide up to 40 I/O lines which are organized into 4 input/output ports and 2 input ports.

All port lines are bit-addressable, and all input/output lines can be individually (bit-wise) configured via port control registers. This configuration selects the direction (input/output), push/pull or open-drain operation, and activation of pull devices for each pin. Edge characteristics (shape) and driver characteristics (output current) of the port drivers can be selected for groups of 4 pins. The I/O ports are true bidirectional ports which are switched to high impedance state when configured as inputs. During the internal reset, all port pins are configured as inputs without pull devices active.

All port lines have programmable alternate input or output functions associated with them. These alternate functions can be assigned to various port pins to support the optimal utilization for a given application. For this reason, certain functions appear several times in [Table 2-3](#).

All port lines that are not used for these alternate functions may be used as general purpose IO lines.

**Table 2-3 Summary of the XE16xyM's Parallel Ports**

Port	Width 144 <sup>1)</sup>	Width 100 <sup>1)</sup>	Width 64 <sup>1)</sup>	Alternate Functions
Port 0	8	8	---	Address lines, Serial interface lines of USIC and CAN, Input/Output lines for CCU6
Port 1	8	8	---	Address lines, Serial interface lines of USIC, Input/Output lines for CCU6, OCDS control, interrupts
Port 2	14	14	11	Address and/or data lines, bus control, Serial interface lines of USIC and CAN, Input/Output lines for CCU6 and CAPCOM2, Timer control signals, DAP/JTAG, interrupts, system clock output
Port 3	8	---	---	Bus arbitration signals, Serial interface lines of USIC and CAN

**Table 2-3 Summary of the XE16xyM's Parallel Ports (cont'd)**

Port	Width 144 <sup>1)</sup>	Width 100 <sup>1)</sup>	Width 64 <sup>1)</sup>	Alternate Functions
Port 4	8	4	---	Chip select signals, Serial interface lines of CAN, Input/Output lines for CAPCOM2, Timer control signals
Port 5	16	11	7	Analog input channels to ADC0, Input/Output lines for CCU6, Timer control signals, DAP/JTAG, OCDS control, interrupts
Port 6 <sup>2)</sup>	4	3	2	ADC control lines, Serial interface lines of CAN, Timer control signals, OCDS control
Port 7	5	5	1	ADC control lines, Serial interface lines of USIC and CAN, Input/Output lines for CCU6, Timer control signals, DAP/JTAG, OCDS control, system clock output
Port 8	7	---	---	Input/Output lines for CCU6, Serial interface lines of USIC, DAP/JTAG, OCDS control
Port 9	8	---	---	Serial interface lines of CAN, Input/Output lines for CCU6, DAP/JTAG, OCDS control
Port 10	16	16	16	Address and/or data lines, bus control, Serial interface lines of USIC and CAN, Input/Output lines for CCU6, CAP/JTAG, OCDS control
Port 11	6	---	---	Serial interface lines of USIC and CAN, Input/Output lines for CCU6
Port 15	8	5	2	Analog input channels to ADC1, Timer control signals

<sup>1)</sup> These columns describe the availability of port pins in the different packages.

<sup>2)</sup> The drivers of these pins are supplied by V<sub>DDPA</sub>.

## **2.4 Clock Generation**

The Clock Generation Unit uses a programmable on-chip PLL with multiple prescalers to generate the clock signals for the XE16xyM with high flexibility. The system clock  $f_{SYS}$  is the reference clock signal, which can be output to the external system. The system clock  $f_{SYS}$  can be derived from several internal and external clock sources.

The on-chip high-precision oscillator (OSC\_HP) can drive an external crystal or accepts an external clock signal. The oscillator clock frequency can be multiplied by the on-chip PLL (by a programmable factor) or can be divided by a programmable prescaler factor.

An internal clock source can provide a clock signal without requiring an external crystal.

The Oscillator Watchdog (OWD) supervises the input clock and enables an emergency clock if the input clock appears as not reliable.

## **2.5 Power Management**

The XE16xyM can operate within a wide supply voltage range from 3 V to 5 V. The internal core supply voltage is generated via on-chip Embedded Voltage Regulators and is supervised by on-chip Power Validation Circuits.

Two IO power domains help to reduce heat dissipation by supplying the major part of the device with a low voltage (3 V), while still connecting analog 5 V sensor signals to the ADCs (5 V).

The XE16xyM provides several means to control the power it consumes either at a given time or averaged over a certain timespan. Three mechanisms can be used (partly in parallel):

- **Supply Voltage Management** allows to switch off the supply voltage. This drastically reduces the power consumed because of leakage current, in particular at high temperature. A power-on reset restarts the system.
- **Clock Generation Management** controls the distribution and the frequency of internal and external clock signals. While the clock signals for currently inactive parts of logic are disabled automatically, the user can reduce the XE16xyM's CPU clock frequency which drastically reduces the consumed power.  
External circuitry can be controlled via the programmable frequency output EXTCLK.
- **Peripheral Management** permits temporary disabling of peripheral modules. Each peripheral can separately be disabled/enabled.

*Note: When selecting the supply voltage and the clock source and generation method, the required parameters must be carefully written to the respective bitfields, to avoid unintended intermediate states. Recommended sequences are provided which ensure the intended operation of power supply system and clock system.*

## **2.6 On-Chip Debug Support (OCDS)**

The On-Chip Debug Support system provides a broad range of debug and emulation features built into the XE16xyM. The user software running on the XE16xyM can thus be debugged within the target system environment.

The OCDS is controlled by an external debugging device via the debug interface and an optional break interface. The debugger controls the OCDS via a set of dedicated registers accessible via the debug interface. Additionally, the OCDS system can be controlled by the CPU, e.g. by a monitor program. An injection interface allows the execution of OCDS-generated instructions by the CPU.

Multiple breakpoints can be triggered by on-chip hardware, by software, or by an external trigger input. 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, a data transfer, or/and the activation of an external signal.

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

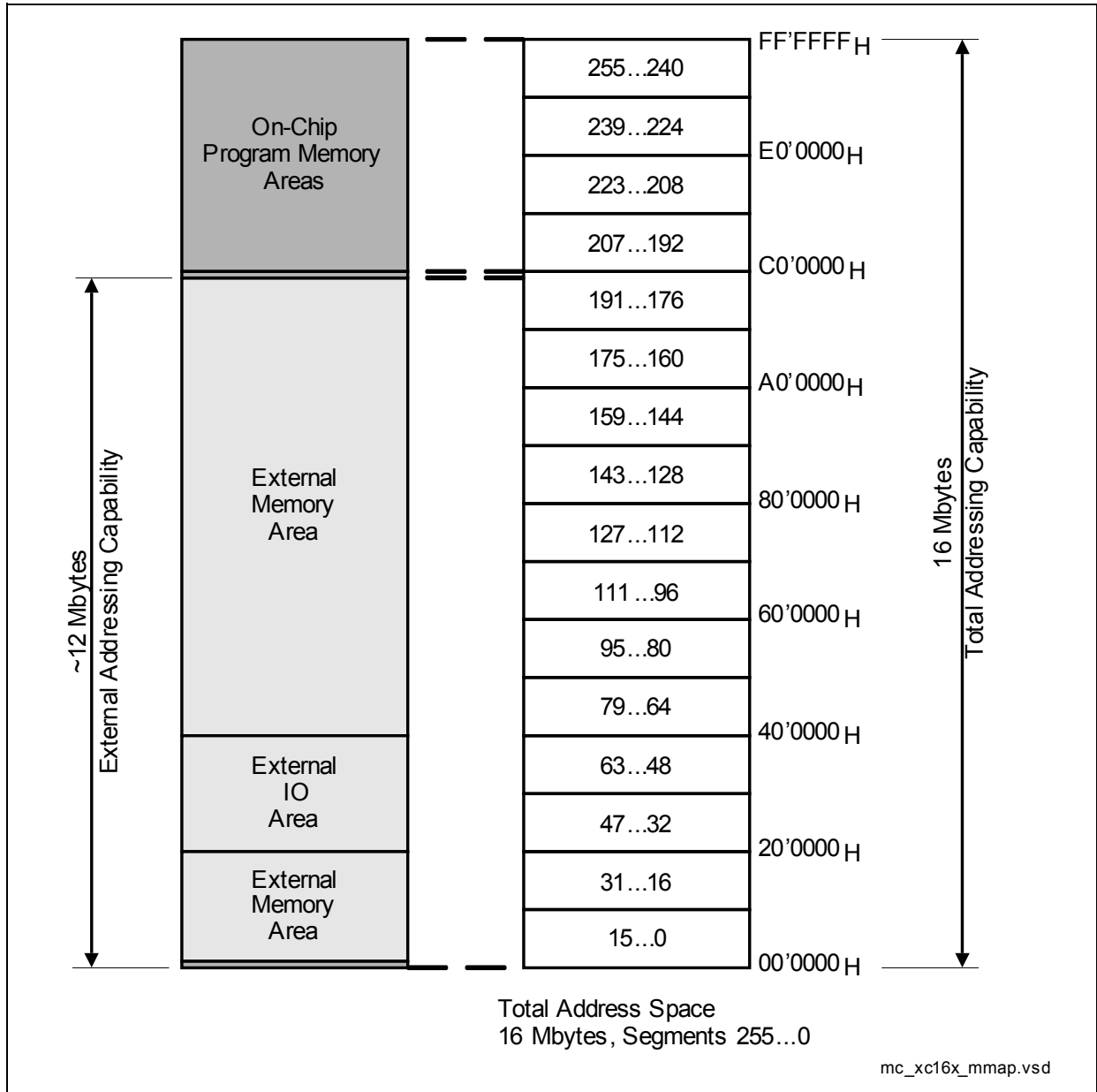
For the debug interface two variants can be used:

- Debug interface through the DAP port. This interface uses 2 DAP lines.
- Debug interface through the IEEE-1149-conforming JTAG port. This interface uses 4 JTAG lines.

The optional break interface uses another 2 lines.

### 3 Memory Organization

The memory space of the XE16xyM is configured in a “Von Neumann” architecture. This means that code and data are accessed within the same linear address space. All of the physically separated memory areas, including internal ROM and Flash, internal RAM, the internal Special Function Register Areas (SFRs and ESFRs), the internal IO area, and external memory are mapped into one common address space.

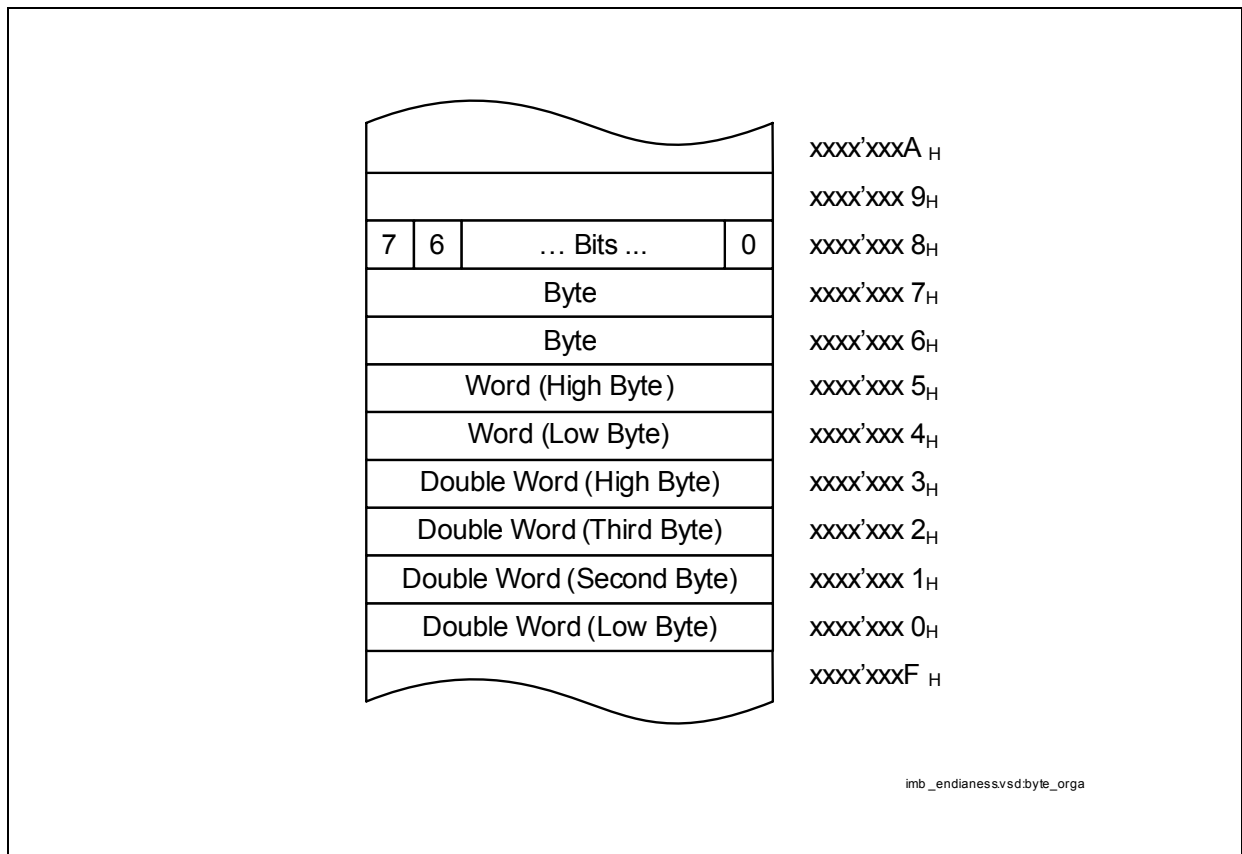


**Figure 3-1 Address Space Overview**

## Memory Organization

The XE16xyM provides a total addressable memory space of 16 Mbytes. This address space is arranged as 256 segments of 64 Kbytes each, and each segment is again subdivided into four data pages of 16 Kbytes each (see [Figure 3-1](#)).

Bytes are stored at even or odd byte addresses. Words are stored in ascending memory locations with the low byte at an even byte address being followed by the high byte at the next odd byte address ("little endian"). Double words (code only) are stored in ascending memory locations as two subsequent words. Single bits are always stored in the specified bit position at a word address. Bit position 0 is the least significant bit of the byte at an even byte address, and bit position 15 is the most significant bit of the byte at the next odd byte address. Bit addressing is supported for a part of the Special Function Registers, a part of the internal RAM and for the General Purpose Registers.



**Figure 3-2 Storage of Words, Bytes and Bits in a Byte Organized Memory**

*Note: Byte units forming a single word or a double word must always be stored within the same physical (internal, external, ROM, RAM) and organizational (page, segment) memory area.*



### 3.1 Address Mapping

All the various memory areas and peripheral registers (see [Table 3-1](#)) are mapped into one contiguous address space. All sections can be accessed in the same way. The memory map of the XE16xyM contains some reserved areas, so future derivatives can be enhanced in an upward-compatible fashion.

*Note: [Table 3-1](#) shows the maximum available memory areas. The actual available memory areas depend on the selected device type.*

**Table 3-1 XE16xyM Memory Map <sup>1)</sup>**

Address Area	Start Loc.	End Loc.	Area Size <sup>2)</sup>	Notes
IMB register space	FF'FF00 <sub>H</sub>	FF'FFFF <sub>H</sub>	256 Bytes	
Reserved	F0'0000 <sub>H</sub>	FF'FEFF <sub>H</sub>	< 1 Mbyte	Minus IMB registers
Reserved for EPSRAM	E8'8000 <sub>H</sub>	EF'FFFF <sub>H</sub>	480 Kbytes	Mirrors EPSRAM
Emulated PSRAM	E8'0000 <sub>H</sub>	E8'7FFF <sub>H</sub>	up to 32 Kbytes	With Flash timing
Reserved for PSRAM	E0'8000 <sub>H</sub>	E7'FFFF <sub>H</sub>	480 Kbytes	Mirrors PSRAM
PSRAM	E0'0000 <sub>H</sub>	E0'7FFF <sub>H</sub>	up to 32 Kbytes	Program SRAM
Reserved for Flash	CD'0000 <sub>H</sub>	DF'FFFF <sub>H</sub>	1,216 Kbytes	
Flash 3	CC'0000 <sub>H</sub>	CC'FFFF <sub>H</sub>	64 Kbytes	
Flash 2	C8'0000 <sub>H</sub>	CB'FFFF <sub>H</sub>	256 Kbytes	
Flash 1	C4'0000 <sub>H</sub>	C7'FFFF <sub>H</sub>	256 Kbytes	
Flash 0	C0'0000 <sub>H</sub>	C3'FFFF <sub>H</sub>	256 Kbytes <sup>3)</sup>	Minus res. seg.
External memory area	40'0000 <sub>H</sub>	BF'FFFF <sub>H</sub>	8 Mbytes	
External IO area <sup>4)</sup>	21'0000 <sub>H</sub>	3F'FFFF <sub>H</sub>	1,984 Kbytes	
Reserved	20'C000 <sub>H</sub>	20'FFFF <sub>H</sub>	16 Kbytes	
USIC0–3 alternate regs.	20'B000 <sub>H</sub>	20'BFFF <sub>H</sub>	4 Kbytes	Accessed via EBC
MultiCAN alternate regs.	20'8000 <sub>H</sub>	20'AFFF <sub>H</sub>	12 Kbytes	Accessed via EBC
Reserved	20'6000 <sub>H</sub>	20'7FFF <sub>H</sub>	8 Kbytes	
USIC0–3 registers	20'4000 <sub>H</sub>	20'5FFF <sub>H</sub>	8 Kbytes	Accessed via EBC
Reserved	20'6800 <sub>H</sub>	20'7FFF <sub>H</sub>	6 Kbytes	
MultiCAN registers	20'0000 <sub>H</sub>	20'3FFF <sub>H</sub>	16 Kbytes	Accessed via EBC
External memory area	01'0000 <sub>H</sub>	1F'FFFF <sub>H</sub>	1984 Kbytes	
SFR area	00'FE00 <sub>H</sub>	00'FFFF <sub>H</sub>	0.5 Kbytes	

**Memory Organization**

**Table 3-1 XE16xyM Memory Map (cont'd)<sup>1)</sup>**

<b>Address Area</b>	<b>Start Loc.</b>	<b>End Loc.</b>	<b>Area Size<sup>2)</sup></b>	<b>Notes</b>
Dualport RAM (DPRAM)	00'F600H	00'FDFFH	2 Kbytes	
Reserved for DPRAM	00'F200H	00'F5FFH	1 Kbytes	
ESFR area	00'F000H	00'F1FFH	0.5 Kbytes	
XSFR area	00'E000H	00'EFFFH	4 Kbytes	
Data SRAM (DSRAM)	00'A000 <sub>H</sub>	00'DFFFH	16 Kbytes	
Reserved for DSRAM	00'8000H	00'9FFFH	8 Kbytes	
External memory area	00'0000H	00'7FFFH	32 Kbytes	

- 1) Accesses to the shaded areas are reserved. In devices with external bus interface these accesses generate external bus accesses.
- 2) The areas marked with "<" are slightly smaller than indicated, see column "Notes".
- 3) The uppermost 4-Kbyte sector of the first Flash segment is reserved for internal use (C0'F000<sub>H</sub> to C0'FFFF<sub>H</sub>).
- 4) Several pipeline optimizations are not active within the external IO area. This is necessary to control external peripherals properly.

### **3.2 Register Areas**

The registers controlling the system and peripheral functions of the XE16xyM can be accessed through five address areas. The address areas differ in their access properties. Please refer to [Chapter 3.7](#) and the CPU chapter for further details.

The first three areas provide Special Function Registers (SFRs) access capabilities for controlling the system and peripheral functions of the XE16xyM:

- 512-byte SFR area (located above the DPRAM: 00'FFFF<sub>H</sub> ... 00'FE00<sub>H</sub>).
- 512-byte ESFR area (located below the DPRAM: 00'F1FF<sub>H</sub> ... 00'F000<sub>H</sub>).
- 4-Kbyte XSFR area (located below the ESFR area: 00'EFFF<sub>H</sub> ... 00'E000<sub>H</sub>).

The USIC and MultiCAN registers are located within the external IO area:

- 64-Kbyte external IO area (located in: 20'0000<sub>H</sub> ... 20'FFFF<sub>H</sub>).

The IMB registers are located within a regular memory area. CPU pipeline effects must be regarded for access in this area:

- 256-byte IMB registers area (located in: FF'FF00<sub>H</sub> ... FF'FFFF<sub>H</sub>).

This arrangement provides upward compatibility with the derivatives of the C166 and XC166 families.

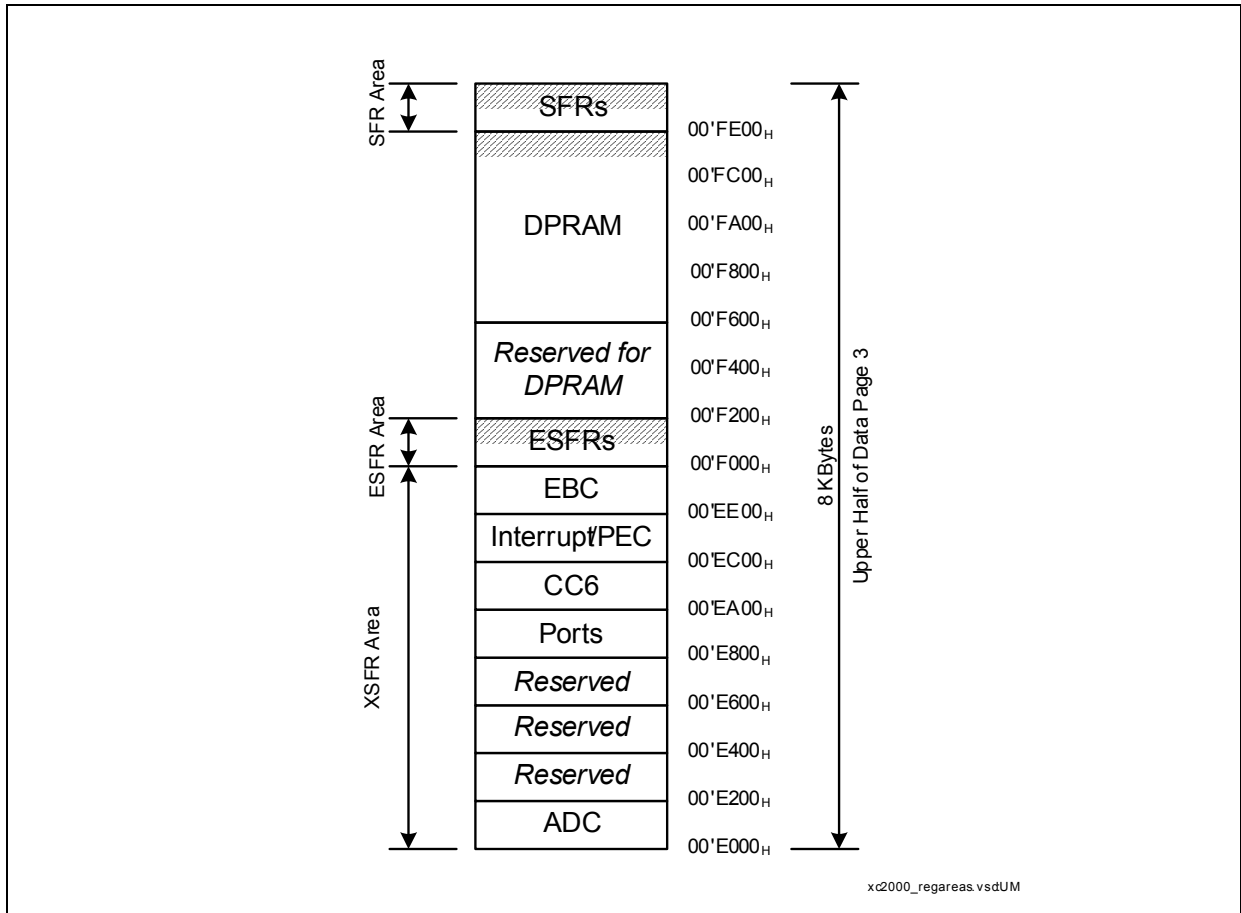
#### **IMB Registers not in IO Area**

Important to note is that IMB registers are not located within the IO area. Only in IO areas the CPU takes care that data accesses are executed exactly in the sequence of their appearance in the instruction stream. Outside of the IO areas the CPU ensures only that accesses to single addresses maintain their sequence. So special care must be taken when accessing the IMB register range. Two examples will help to understand this important issue:

1. Sequence: write to address A; read from address B.
2. Sequence: write to address C; read from address C.

If addresses A, B and C are located in IO areas then the sequence of memory accesses would resemble their sequence in the code.

If addresses A, B and C are located outside of IO areas, then pipeline effects could cause the read from address B in sequence 1 to be performed before the write to address A happens. The CPU will itself ensure that sequence 2 is executed in order. To work around this issue and to enforce sequence order, a read from address A or a write to address B should be performed after the write to address A — both ensure that the read from address B occurs after the write from address A.



**Figure 3-3 Special Function Register Mapping**

*Note: The upper 256 bytes of SFR area, ESFR area, and internal RAM are bit-addressable (see hatched blocks in [Figure 3-3](#)).*

### Special Function Registers

The functions of the CPU, the bus interface, the IO ports, and the on-chip peripherals of the XE16xyM are controlled via a number of Special Function Registers (SFRs).

All Special Function Registers can be addressed via indirect and long 16-bit addressing modes. The (word) SFRs and their respective low bytes in the SFR/ESFR areas can be addressed using an 8-bit offset together with an implicit base address. However, this **does not work** for the respective high bytes!

*Note: Writing to any byte of an SFR causes the not addressed complementary byte to be cleared.*

The upper half of the SFR-area (00'FFFF<sub>H</sub> ... 00'FF00<sub>H</sub>) and the ESFR-area (00'F1FF<sub>H</sub> ... 00'F100<sub>H</sub>) is bit-addressable, so the respective control/status bits can be modified directly or checked using bit addressing.

## Memory Organization

When accessing registers in the ESFR area using 8-bit addresses or direct bit addressing, an Extend Register (EXTR) instruction is required beforehand to switch the short addressing mechanism from the standard SFR area to the Extended SFR area. This is not required for 16-bit and indirect addresses. The GPRs R15 ... R0 are duplicated, i.e. they are accessible within both register blocks via short 2-, 4-, or 8-bit addresses without switching.

ESFR\_SWITCH\_EXAMPLE:

```
EXTR  #4                                ;Switch to ESFR area for next 4 instr.
MOV   STMREL, #data16                   ;STMREL uses 8-bit reg addressing
BFLDL STMCON, #mask, #data8             ;Bit addressing for bitfields
BSET  WUCR.CLRTG                        ;Bit addressing for single bits
MOV   T8REL, R1                         ;T8REL uses 16-bit mem address,
                                         ;R1 is duplicated into the ESFR space
                                         ;(EXTR is not required for this access)
;---- ;-----                        ;The scope of the EXTR #4 instruction ...
                                         ;... ends here!
MOV   T8REL, R1                         ;T8REL uses 16-bit mem address,
                                         ;R1 is accessed via the SFR space
```

In order to minimize the use of the EXTR instructions the ESFR area mostly holds registers which are mainly required for initialization and mode selection. Registers that need to be accessed frequently are allocated to the standard SFR area, wherever possible.

*Note: The tools are equipped to monitor accesses to the ESFR area and will automatically insert EXTR instructions, or issue a warning in case of missing or excessive EXTR instructions.*

Accesses to registers in the XSFR area use 16-bit addresses and require no specific addressing modes or precautions.

### General Purpose Registers

The General Purpose Registers (GPRs) use a block of 16 consecutive words either within the global register bank or within one of the two local register banks. The bit-field BANK in register PSW selects the currently active register bank. The global register bank is mirrored to a section in the DPRAM, the Context Pointer (CP) register determines the base address of the currently active global register bank section. This register bank may consist of up to 16 Word-GPRs (R0, R1, ... R15) and/or of up to 16 byte-GPRs (RL0, RH0, ... RL7, RH7). The sixteen byte-GPRs are mapped onto the first eight Word GPRs (see [Table 3-2](#)).

In contrast to the system stack, a register bank grows from lower towards higher address locations and occupies a maximum space of 32 bytes. The GPRs are accessed via short 2-, 4-, or 8-bit addressing modes using the Context Pointer (CP) register as base

## Memory Organization

address for the global bank (independent of the current DPP register contents). Additionally, each bit in the currently active register bank can be accessed individually.

**Table 3-2 Mapping of General Purpose Registers to DPRAM Addresses**

DPRAM Address	High Byte Registers	Low Byte Registers	Word Registers
<CP> + 1E <sub>H</sub>	–	–	R15
<CP> + 1C <sub>H</sub>	–	–	R14
<CP> + 1A <sub>H</sub>	–	–	R13
<CP> + 18 <sub>H</sub>	–	–	R12
<CP> + 16 <sub>H</sub>	–	–	R11
<CP> + 14 <sub>H</sub>	–	–	R10
<CP> + 12 <sub>H</sub>	–	–	R9
<CP> + 10 <sub>H</sub>	–	–	R8
<CP> + 0E <sub>H</sub>	RH7	RL7	R7
<CP> + 0C <sub>H</sub>	RH6	RL6	R6
<CP> + 0A <sub>H</sub>	RH5	RL5	R5
<CP> + 08 <sub>H</sub>	RH4	RL4	R4
<CP> + 06 <sub>H</sub>	RH3	RL3	R3
<CP> + 04 <sub>H</sub>	RH2	RL2	R2
<CP> + 02 <sub>H</sub>	RH1	RL1	R1
<CP> + 00 <sub>H</sub>	RH0	RL0	R0

The XE16xyM supports fast register bank (context) switching. Multiple global register banks can physically exist within the DPRAM at the same time. Only the global register bank selected by the Context Pointer register (CP) is active at a given time, however. Selecting a new active global register bank is simply done by updating the CP register. A particular Switch Context (SCXT) instruction performs register bank switching by automatically saving the previous context and loading the new context. The number of implemented register banks (arbitrary sizes) is limited only by the size of the available DPRAM.

*Note: The local GPR banks are not memory mapped and the GPRs cannot be accessed using a long or indirect memory address.*

### PEC Source and Destination Pointers

The source and destination address pointers for data transfers on the PEC channels are located in the XSFR area.

## **Memory Organization**

Each channel uses a pair of pointers stored in two subsequent word locations with the source pointer (SRCPx) on the lower and the destination pointer (DSTPx) on the higher word address ( $x = 7 \dots 0$ ). An additional segment register stores the associated source and destination segments, so PEC transfers can move data from/to any location within the complete addressing range.

Whenever a PEC data transfer is performed, the pair of source and destination pointers (selected by the specified PEC channel number) accesses the locations referred to by these pointers independently of the current DPP register contents.

If a PEC channel is not used, the corresponding pointer locations can be used for other purposes.

*Note: Writing to any byte of the PEC pointers causes the not addressed complementary byte to be cleared.*

### **3.3 Data Memory Areas**

The XE16xyM provides two on-chip RAM areas exclusively for data storage:

- The **Dual Port RAM (DPRAM)** can be used for global register banks (GPRs), system stack, storage of variables and other data, in particular for MAC operands.
- The **Data SRAM (DSRAM)** can be used for system stack (recommended), storage of variables and other data.

*Note: Data can also be stored in the PSRAM (see [Section 3.11](#)). However, both data memory areas provide the fastest access.*

Depending on the device additional on-chip memory areas may exist with the special purpose to retain data while the system power domain is switched off. The XE16xyM contains:

- The **Standby RAM (SBRAM)**.
- The **Marker Memory (MKMEM)**.

#### **Dual-Port RAM (DPRAM)**

The XE16xyM provides 2 Kbytes of DPRAM (00'F600<sub>H</sub> ... 00'FDFF<sub>H</sub>). Any word or byte data in the DPRAM can be accessed via indirect or long 16-bit addressing modes, if the selected DPP register points to data page 3. Any word data access is made on an even byte address.

For PEC data transfers, the DPRAM can be accessed independent of the contents of the DPP registers via the PEC source and destination pointers.

The upper 256 bytes of the DPRAM (00'FD00<sub>H</sub> through 00'FDFF<sub>H</sub>) are provided for single bit storage, and thus they are bit addressable.

*Note: Code cannot be executed out of the DPRAM.*

*Note: The locations 00'FBFE<sub>H</sub> ... 00'FC01<sub>H</sub> of the DPRAM may be altered during the initialization phase after a reset. This area, therefore, should not store data to be preserved beyond a reset.*

An area of 3 Kbytes is dedicated to DPRAM (00'F200<sub>H</sub> ... 00'FDFF<sub>H</sub>). The locations without implemented DPRAM are reserved.

#### **Data SRAM (DSRAM)**

The XE16xyM provides 16 Kbytes of DSRAM (00'A000<sub>H</sub> ... 00'DFFF<sub>H</sub>). Any word or byte data in the DSRAM can be accessed via indirect or long 16-bit addressing modes, if the selected DPP register points to data page 3 (for the range 00'C000<sub>H</sub> ... 00'DFFF<sub>H</sub>) or to data page 2 (for the range 00'A000<sub>H</sub> ... 00'BFFF<sub>H</sub>). Any word data access is made on an even byte address.

For PEC data transfers, the DSRAM can be accessed independent of the contents of the DPP registers via the PEC source and destination pointers.



*Note: Code cannot be executed out of the DSRAM.*

An area of 24 Kbytes is dedicated to DSRAM (00'8000<sub>H</sub> ... 00'DFFF<sub>H</sub>). The locations without implemented DSRAM are reserved.

### **Standby RAM (SBRAM)**

The SBRAM provides 8 Kbyte of memory supplied by the wake-up power domain (DMP\_M). Its main purpose is to retain state while the system power domain (DMP\_1) is switched off.

Unlike the other memories the SBRAM is not mapped into the address range of the processor. Reading and writing is done via two address and two data SFRs. Details of the access mechanism are described in [Section 3.12](#).

*Note: Code cannot be executed out of the SBRAM.*

*Note: The upper 32 Bytes of the SBRAM may be altered during the initialization phase after a power reset. This area, therefore, should not store data to be preserved beyond a power reset. If Fast Startup Mode is used, this area must not be altered by the application software.*

### **Marker Memory (MKMEM)**

The MKMEM provides 4 bytes of memory. It can be used to store system state information during power down.

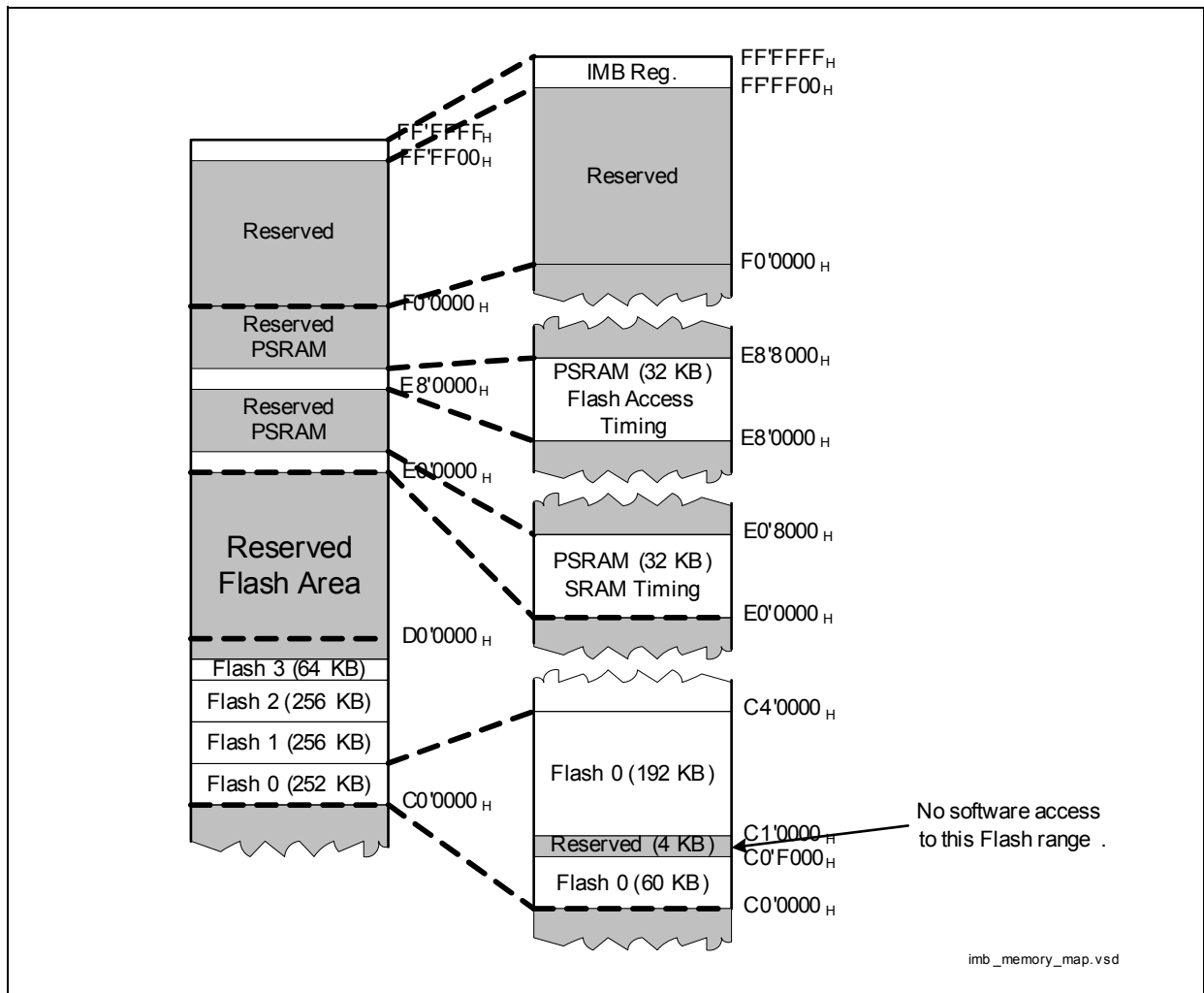
The MKMEM consists of 2 16-bit SFRs that are accessible as all other SFRs. Details are described in the SCU chapter.

*Note: Code cannot be executed out of the MKMEM.*

### 3.4 Program Memory Areas

The XE16xyM provides two on-chip program memory areas for code/data storage:

- The **Program Flash/ROM** stores code and constant data. Flash memory is (re-) programmed by the application software or flash loaders, ROM is mask-programmed in the factory.
- The **Program SRAM (PSRAM)** stores temporary code sequences and other data. For example higher level boot loader software can be written to the PSRAM and then be executed to program the on-chip Flash memory.



**Figure 3-4 On-Chip Program Memory Mapping**

### **3.4.1 Program/Data SRAM (PSRAM)**

The XE16xyM provides up to 32 Kbytes of PSRAM (E0'0000<sub>H</sub> ... E0'7FFF<sub>H</sub>). The PSRAM provides fast code execution without initial delays. Therefore, it supports non-sequential code execution, for example via the interrupt vector table.

Any word or byte data in the PSRAM can be accessed via indirect or long 16-bit addressing modes, if the selected DPP register points to one of its data pages. Any word data access is made on an even byte address.

For PEC data transfers, the PSRAM can be accessed independent of the contents of the DPP registers via the PEC source and destination pointers.

Any data can be stored in the PSRAM. Because the PSRAM is optimized for code fetches, however, data accesses to the data memories provide higher performance.

*Note: The PSRAM is not bit-addressable.*

*Note: The upper 256 Bytes of the PSRAM may be altered during the initialization phase after a reset. This area, therefore, should not store data to be preserved beyond a reset.*

*Also, during bootstrap loader operation, the serially received data is stored in the PSRAM starting at location E0'0000<sub>H</sub>.*

An area of 512 Kbytes is dedicated to PSRAM (E0'0000<sub>H</sub> ... F7'FFFF<sub>H</sub>). The locations without implemented PSRAM are reserved.

### **Flash Emulation**

During code development the PSRAM will often be used for storing code or data that the production chip will later contain in the flash memory. In order to ensure similar execution time the PSRAM supports a second access path in the range E8'0000<sub>H</sub> ... EF'FFFF<sub>H</sub> with timing parameters that correspond to Flash timing. The number of wait-cycles is determined by the flash access timing configuration (see [IMB\\_IMBCTRL.WSFLASH](#)). Writes are always performed without wait-cycles.

This flash access timing imitation is nearly cycle accurate because the same read logic as for reading the flash memory is used<sup>1)</sup>. Discrepancies might occur if the software uses the PSRAM for flash emulation and directly as PSRAM. During emulation access conflicts can cause a slightly different timing as in the product chip where these conflicts do not occur.

Another source of timing differences can be access conflicts at the flash modules in the product chip. Data reads and instruction fetches that target different flash modules can be executed concurrently whereas if they target the same flash module they are

1) The dual use of the flash read logic might cause unexpected behavior: while the IMB Core is busy with updating the protection configuration (after startup or after changing the security pages) read accesses to the flash emulation range of the PSRAM are blocked because Flash data reads would be blocked also.

**Memory Organization**

executed sequentially with the data access as first. In the flash emulation this type of conflict can not occur. The data and the instruction access will both incur the defined number of wait-cycles (as if they would target different flash modules) and if they collide at the PSRAM interface the instruction fetch will see an additional wait-cycle.

**Data Integrity**

The PSRAM contains its own error control which can be switched between ECC and parity. Details are described in the SCU chapter.

**Write Protection**

As the PSRAM is often used to store timing critical code or constant data it is supplied with a write protection. After storing critical data in the PSRAM the register field **IMB\_IMBCTR.H.PSPROT** can be used to split the PSRAM into a read-only and a writable part. Write accesses to the read-only part are blocked and a trap can be activated.

**3.4.2 Non-Volatile Program Memory (Flash)**

The XE16xyM provides up to up to 832 Kbytes of program Flash starting at address C0'0000<sub>H</sub>. Code and data fetches are always 64-bit aligned, using byte select lines for word and byte data.

Any word or byte data in the program memory can be accessed via indirect or long 16-bit addressing modes, if the selected DPP register points to one of the respective data pages. Any word data access is made on an even byte address.

For PEC data transfers, the program memory can be accessed independent of the contents of the DPP registers via the PEC source and destination pointers.

*Note: The program memory is not bit-addressable.*

An area of 2 Mbytes is dedicated to program memory (C0'0000<sub>H</sub> ... DF'FFFF<sub>H</sub>). The locations without implemented program memory are reserved.

A more detailed description can be found in **“Embedded Flash Memory” on Page 3-20**.

### **3.5 System Stack**

The system stack may be defined anywhere within the XE16xyM's memory areas (including external memory).

For all system stack operations the respective stack memory is accessed via a 24-bit stack pointer. The Stack Pointer (SP) register provides the lower 16 bits of the stack pointer (stack pointer offset), the Stack Pointer Segment (SPSEG) register adds the upper 8 bits of the stack pointer (stack segment). The system stack grows downward from higher towards lower locations as it is filled. Only word accesses are supported to the system stack.

Register SP is decremented before data is pushed on the system stack, and incremented after data has been pulled from the system stack. Only word accesses are supported to the system stack.

By using register SP for stack operations, the size of the system stack is limited to 64 KBytes. The stack must be located in the segment defined by register SPSEG.

The stack pointer points to the latest system stack entry, rather than to the next available system stack address.

A stack overflow (STKOV) register and a stack underflow (STKUN) register are provided to control the lower and upper limits of the selected stack area. These two stack boundary registers can be used both for protection against data corruption.

For best performance it is recommended to locate the stack to the DPRAM or to the DSRAM. Using the DPRAM may conflict with register banks or MAC operands.

### **3.6 Protected Bits**

The XE16xyM provides a special mechanism to protect bits which can be modified by the on-chip hardware from being changed unintentionally by software accesses to related bits (refer also to section “Bit Manipulation Unit” in the CPU chapter). The “rwh” and “wh” bits and bifielts of the following registers support bit protection:

**Table 3-3 XE16xyM Protected Bits**

<b>Register</b>	<b>Component(s)</b>	<b>Notes</b>
TFR	CPU	Trap Flag Register
PSW	CPU	Processor Status Word
PECISNC	CPU	PEC channel interrupt request flags
MPU_PRA	MPU	Protection Range Address
SCU_GSCSWREQ	SCU	Global State Control Software Request
RTC_ISNC	RTC	Interrupt node sharing request flags
CC2_OUT	CC2	Compare output bits
GPT12E_T2CON	GPT	GPT1 timer T2 flags
GPT12E_T3CON	GPT	GPT1 timer T3 flags and output toggle latch
GPT12E_T4CON	GPT	GPT1 timer T4 flags
GPT12E_T6CON	GPT	GPT2 timer T6 output toggle latch
xIC	CPU, SCU and Peripheral units	All interrupt control registers. A complete list is given in the interrupt and exception control chapter
Px_OUT	Ports	All port output registers

### **3.7 IO Areas**

The following areas of the XE16xyM's address space are marked as IO area:

- The **external IO area** is provided for external peripherals (or memories) and also comprises the on-chip LXBus-peripherals, such as the MultiCAN or USIC modules. It is located from 20'0000<sub>H</sub> to 3F'FFFF<sub>H</sub> (2 Mbytes).
- The **internal IO area** provides access to the internal peripherals and is split into three blocks:
  - The SFR area, located from 00'FE00<sub>H</sub> to 00'FFFF<sub>H</sub> (512 bytes).
  - The ESFR area, located from 00'F000<sub>H</sub> to 00'F1FF<sub>H</sub> (512 bytes).
  - The XSFR area, located from 00'E000<sub>H</sub> to 00'FFFF<sub>H</sub> (4 Kbytes).

*Note: The external IO area supports real byte accesses. The internal IO area does not support real byte transfers, the complementary byte is cleared when writing to a byte location.*

The IO areas have special properties, because peripheral modules must be controlled in a different way than memories:

- Accesses are not buffered and cached, the write back buffers and caches are not used to store IO read and write accesses.
- Speculative reads are not executed, but delayed until all speculations are solved (e.g. pre-fetching after conditional branches).
- Data forwarding is disabled, an IO read access is delayed until all IO writes pending in the pipeline are executed, because peripherals can change their internal state after a write access.

### **3.8 External Memory Space**

The XE16xyM is capable of using an address space of up to 16 Mbytes. Only parts of this address space are occupied by internal memory areas or are reserved. A total area of approximately 12 Mbytes references external memory locations. This external memory is accessed via the XE16xyM's external bus interface.

**Selectable memory bank sizes** are supported: The maximum size of a bank in the external memory space depends on the number of activated address bits. It can vary from 64 Kbytes (with A15 ... A0 activated) to 12 Mbytes (with A23 ... A0 activated). The logical size of a memory bank and its location in the address space is defined by programming the respective address window. It can vary from 4 Kbytes to 12 Mbytes.

Each bank can be directly addressed via the address bus, while the programmable chip select signals can be used to select various memory banks.

The XE16xyM also supports **four different bus types**:

- Multiplexed 16-bit Bus (default after Reset).
- Multiplexed 8-bit Bus.
- Demultiplexed 16-bit Bus.
- Demultiplexed 8-bit Bus.

For further details about the external bus configuration and control refer to the External Bus Controller chapter.

External word and byte data can only be accessed via indirect or long 16-bit addressing modes using one of the four DPP registers. There is no short addressing mode for external operands. Any word data access is made to an even byte address.

For PEC data transfers the external memory can be accessed independent of the contents of the DPP registers via the PEC source and destination pointers.

*Note: The external memory is not bit addressable.*

### **3.9 Crossing Memory Boundaries**

The address space of the XE16xyM is implicitly divided into equally sized blocks of different granularity and into logical memory areas. Crossing the boundaries between these blocks (code or data) or areas requires special attention to ensure that the controller executes the desired operations.

**Memory Areas** are partitions of the address space assigned to different kinds of memory (if provided at all). These memory areas are the SFR areas, the on-chip program or data RAM areas, the on-chip ROM/Flash (if available), the on-chip LxBus-peripherals (if integrated), and the external memory.

Accessing subsequent data locations which belong to different memory areas is no problem. However, when executing code, the different memory areas must be switched explicitly via branch instructions. Sequential boundary crossing is not supported and leads to erroneous results.



## **Memory Organization**

*Note: Changing from the external memory area to the on-chip RAM area takes place within segment 0.*

**Segments** are contiguous blocks of 64 Kbytes each. They are referenced via the Code Segment Pointer CSP for code fetches and via an explicit segment number for data accesses overriding the standard DPP scheme.

During code fetching, segments are not changed automatically, but rather must be switched explicitly. The instructions JMPS, CALLS and RETS will do this.

**Data Pages** are contiguous blocks of 16 Kbytes each. They are referenced via the data page pointers DPP3 ... DPP0 and via an explicit data page number for data accesses overriding the standard DPP scheme. Each DPP register can select one of the possible 1024 data pages. The DPP register which is used for the current access is selected via the two upper bits of the 16-bit data address. Therefore, subsequent 16-bit data addresses which cross the 16-Kbytes data page boundaries will use different data page pointers, while the physical locations need not be subsequent within memory.

### **3.10 Embedded Flash Memory**

This chapter describes the embedded flash memory of the XE16xyM:

- **Section 3.10.1** defines the flash specific nomenclature and the structure of the flash memory.
- **Section 3.10.2** describes the operating modes.
- **Section 3.10.3** contains all operations.
- **Section 3.10.4** gives the details of operating sequences.
- The three sections **Section 3.10.7**, **Section 3.10.8** and **Section 3.10.9** look more into depth of maintaining data integrity and protection issues.
- **Section 3.10.10** discusses Flash EEPROM emulation.
- **Section 3.10.11** describes interrupt generation by the flash memory.

The **Chapter 3.11** describes how the flash memory is embedded into the memory architecture of the XE16xyM and lists all SFRs that affect its behavior.

#### **3.10.1 Definitions**

This section defines the nomenclature and some abbreviations as a base for the rest of the document. The used flash memory is a non-volatile memory (“**NVM**”) based on a floating gate one-transistor cell. It is called “non-volatile” because the memory content is kept when the memory power supply is shut off.

#### **Logical and Physical States**

Flash memory content can not be changed directly as in SRAMs. Changing data is a complicated process with a typically much longer duration than reading.

- **Erasing:** The erased state of a cell is logical 0. Forcing an flash cell to this state is called “erasing”. Erasing is possible with a minimum granularity of one page (see below). A device is delivered with completely erased flash memory.
- **Programming:** The programmed state of a cell is logical 1. Changing an erased cell to this state is called “programming”. A page must only be programmed once and has to be erased before it can be programmed again.

The above listed processes have certain limitations:

- **Retention:** This is the time during which the data of a flash cell can be read reliably. The retention time is a statistical figure that depends on the operating conditions of the flash array (temperature profile) and the accesses to the flash array. With an increasing number of program/erase cycles (see endurance) the retention is lowered. Drain and gate disturbs decrease data retention as well.
- **Endurance:** As described above the data retention is reduced with an increasing number of program/erase cycles. A flash cell incurs one cycle whenever its page or sector is erased. This number is called “endurance”. As said for the retention it is a statistical figure that depends on operating conditions and the use of the flash cells and not to forget on the required quality level.

## **Memory Organization**

- **Drain Disturb:** Because of using a so called “one-transistor” flash cell each program access disturbs all pages of the same sector slightly. Over long these “drain disturbs” make 0 and 1 values indistinguishable and thus provoke read errors. This effect is again interrelated with the retention. A cell that incurred a high number of drain disturbs will have a lower retention. The physical sectors of the flash array are isolated from each other. So pages of a different sector do not incur a drain disturb. This effect must be therefor considered when the page erase feature is used.

The durations of programming and erasing as well as the limits for endurance, retention and drain disturbs are documented in the data sheet.

***Attention: No means exist in the device that prevent the application from violating these limitation.***

### **Array Structure**

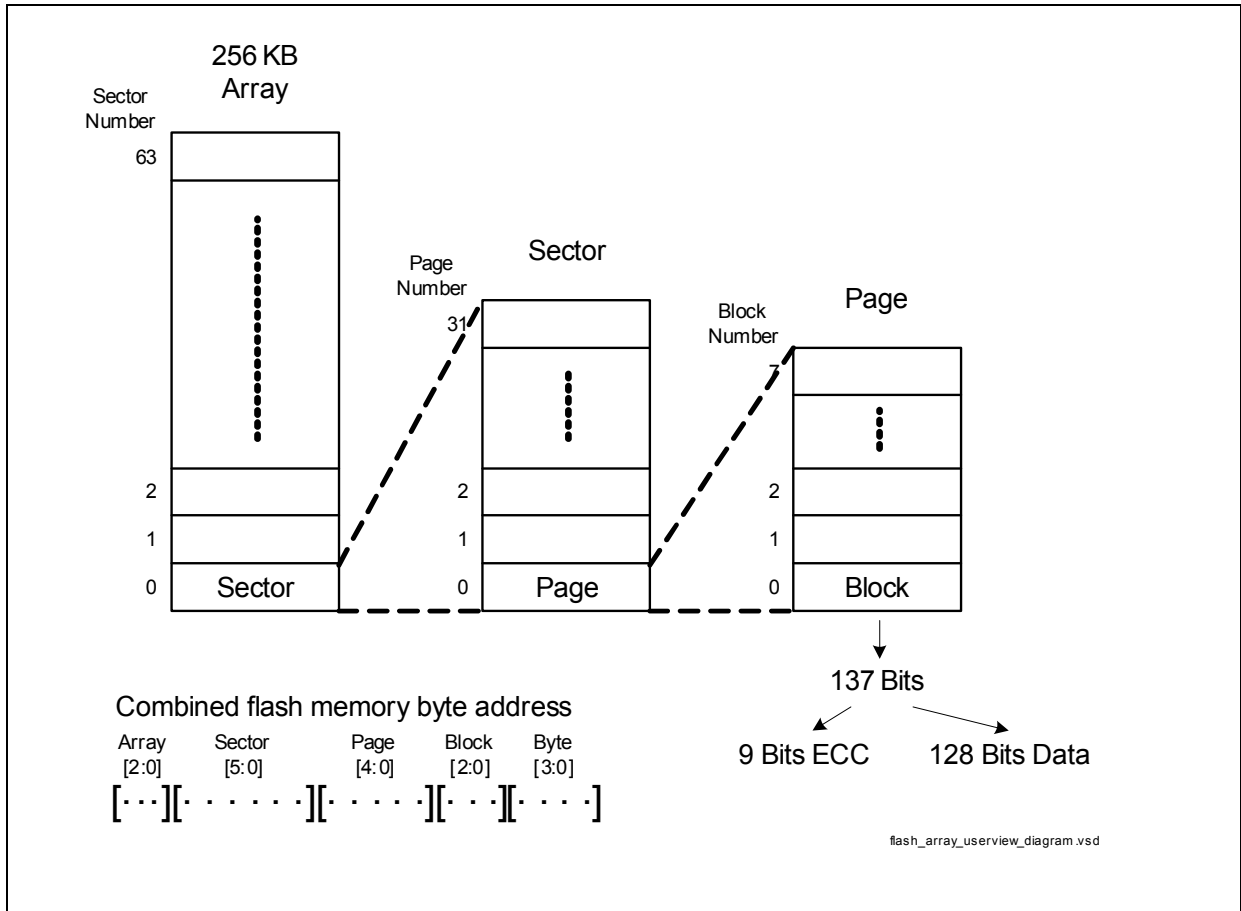
The flash memory is hierarchically structured:

- **Block:** A block consists of 128 user data bits (i.e. 16 bytes) and 9 ECC bits. One read access delivers one block.
- **Page:** A page consists of 8 blocks (i.e. 128 bytes). Programming changes always complete pages.
- **Sector:** A sector consists of 32 pages (i.e. 4096 bytes). The pages of one sector are affected by drain disturb as described above. The pages of different sectors are isolated from each other.
- **Array:** Each 256 KB array has 64 sectors<sup>1)</sup> and the 64 KB array has 16 sectors. Usually when referring to an “array” this contains as well all accompanying logic as assembly buffer, high voltage logic and the digital logic that allows to operate them in parallel.
- **Memory:** The complete flash memory of the XE16xyM consists of 4 flash arrays.

This structure of the 256 KB array is visualized in [Figure 3-5](#). The structure of the 64 KB array is analog.

---

1) In the Flash0 one sector is reserved for device internal purposes. It is not accessible by software.



**Figure 3-5 Flash Structure**

### 3.10.2 Operating Modes

The IMB and the flash memory and each flash module have certain modes of operation. Some modes define clocking and power supply and the operating state of the analog logic as oscillators and voltage pumps. Overall system modes (e.g. startup mode) influence the behavior or the flash memory as well.

Other modes define the functional behavior. These will be discussed here.

#### 3.10.2.1 Standard Read Mode

After reset and after performing a clean startup the flash memory with all its modules is in “standard read mode”. In this mode it behaves as an on-chip ROM. This mode is entered:

- After reset when the complete start-up has been performed.
- After completion of a longer lasting command like “erase” or “program” which is acknowledged by clearing the “busy” flag.
- Immediately after each other command execution.

- In case of detecting an execution error like attempting to write to a write protected range, sending a wrong password, after all sequence errors.

For the long lasting commands the read mode stays active until the last command of the sequence is received and the operation is started.

### **3.10.2.2 Command Mode**

After receiving the last command of a command sequence the addressed flash module (not the whole flash memory!) is placed into command mode. For most commands this will not be noticed by the user as the command executes immediately and afterwards the flash module is placed again into read mode. For the long lasting commands the flash module stays in command mode for several milliseconds. This is reported by setting the corresponding “busy” flag. The data of a busy flash module cannot be read but other not busy flash modules stay readable. New command sequences are generally not accepted and cause a sequence error until the running operation has finished. In certain cases however new command sequences are accepted in order to enable concurrent programming and erase of independent flash modules.

Read accesses to busy flash modules stall the CPU until the read mode is entered again. A stalled CPU responds only to the reset. As no interrupts can be handled this state must be avoided. Nevertheless this feature can be used to execute code from a flash module that erases or programs data in the same flash module.

*Note: Because command sequences to busy flash memory are not always rejected by the hardware with a sequence error it is necessary to handle all commands more careful than in previous device generations that didn't support concurrent processes. A new command sequence shall be only be issued to a flash module after checking that it is not busy anymore. This is especially vital when using the “stall CPU when reading busy flash” feature. Further advice can be found in [Section 3.10.5](#) (sequence errors) and [Section 3.10.6](#) (concurrent processes).*

### **3.10.2.3 Page Mode**

The page mode is entered with the “[Enter Page Mode](#)” command. Please find its description below. A flash module that is in page mode can still be read (so it is concurrently in “read mode”). At a time only one flash module can be in page mode.

When the flash memory is in page mode — i.e. one of the flash modules is in page mode — some command sequences are not allowed. These are all erase sequences and the “change read margin” sequence. These are ignored and a sequence error is reported.

### **3.10.3 Operations**

The flash memory supports the following operations:

- Instruction fetch.
- Data read.
- Command sequences to change data and control the protection.

#### **3.10.3.1 Instruction Fetch from Flash Memory**

Instructions are fetched by the PMU in groups of aligned 64 bits. These code requests are forwarded to the flash memory. It needs a varying number of cycles (depending on the system clock frequency) to perform the read access. The number of cycles must be known to the IMB Core because the flash does not signal data availability. The number of wait states is therefore stored in the **IMB\_IMBCTRL** register.

The complete duration of a flash read access is: **IMB\_IMBCTRL.WSFLASH** + 1 cycles.

Consult the data sheet for correct values of **WSFLASH** dependent on the system clock frequency and device.

One read access to the flash memory delivers 128 data bits and a 9-bit ECC value. The ECC value is used to detect and possibly correct errors. The addressed 64-bit part of the 128-bit chunk is sent to the PMU. The complete 128 data bits and the 9 ECC bits are stored in the IMB Core with their address. If a succeeding fetch request matches this address the data is delivered from the buffer without performing a read access in the flash memory. The delivery from the buffer happens after one cycle. The flash read wait-cycles are not waited.

The stored data are a kind of instruction cache. In order to support self-modifying code (e.g. boot loaders) this cache is invalidated when the corresponding address is written (i.e. erased or programmed).

In addition to this fetch buffer the IMB Core has an additional performance increasing feature — the Linear Code Pre-Fetch. When this feature is enabled with **IMB\_IMBCTRL.DLCPF** = 0 the IMB Core fetches autonomously the following instructions while the CPU executes from its own buffers or the fetch buffer. As this feature is fetching only the linear successors (it does not analyze the code stream) it is most effective for code with longer linear sequences. For code with a high density of jumps and calls it can even cause a reduction of performance and should be switched off.

### **3.10.3.2 Data Reads from Flash Memory**

Data reads are issued by the DMU. Data is always requested in 16-bit words. The flash memory delivers for every read request 128 bits plus ECC as described in **“Instruction Fetch from Flash Memory” on Page 3-24**.

The IMB Core has to get all 128 bits to evaluate the ECC data. The requested 16 bits will be delivered to the DMU. All data and ECC bits are kept in the data register and their address is kept in the address register. For all following data reads the address is compared with the address register and in case of a match the data is delivered after one cycle from the data register. Every data read that is not delivered from this cache invalidates the cache content. When the requested data arrives the cache contains again valid data.

This small data cache is invalidated when a write (i.e. erase or program) access to this address happens.

For data reads the IMB Core does not perform any autonomous pre-fetching.

### **3.10.3.3 Data Writes to Flash Memory**

Flash memory content can not be changed by directly writing data to this memory. Command sequences are used to execute all other operations in the flash except reading. Command sequences consist of data writes with certain data to the flash memory address range. All data moves targeting this range are interpreted as command sequences. If they do not match a defined one or if the IMB Core cannot accept a new one because it is busy a sequence error is reported.

### **3.10.3.4 Command Sequences**

As described before changing data in the flash memory is performed with command sequences.

**Table 3-4 Command Sequence Overview**

<b>Command Sequence</b>	<b>Description</b>	<b>Details on Page</b>
<b>Reset to Read</b>	Reset Flash into read mode and clear error flags.	<b>Page 3-28</b>
<b>Clear Status</b>	Clear error and status flags.	<b>Page 3-28</b>
<b>Change Read Margin</b>	Change read margins.	<b>Page 3-28</b>
<b>Enter Page Mode</b>	Prepare page for programming.	<b>Page 3-29</b>
<b>Enter Security Page Mode</b>	Prepare security page for programming.	<b>Page 3-30</b>
<b>Load Page Word</b>	Load page with data.	<b>Page 3-31</b>
<b>Program Page</b>	Start page programming process.	<b>Page 3-32</b>
<b>Erase Sector</b>	Start sector erase process.	<b>Page 3-33</b>
<b>Erase Page</b>	Start page erase process.	<b>Page 3-34</b>
<b>Erase Security Page</b>	Start security page erase process.	<b>Page 3-34</b>
<b>Disable Read Protection</b>	Disable temporarily read protection with password.	<b>Page 3-35</b>
<b>Disable Write Protection</b>	Disable temporarily write protection with password.	<b>Page 3-36</b>
<b>Re-Enable Read/Write Protection</b>	Re-enable protection.	<b>Page 3-37</b>



### 3.10.4 Details of Command Sequences

The description defines the command sequence with pseudo assembler code. It is “pseudo” because all addresses are direct addresses which is generally not possible in real assembler code.

The commands are called by a sequence of one to six data moves into the flash memory range. The data moves must be of the “word” type, i.e. not byte move instructions. The following sections describe each command. The following abbreviations for addresses and data will be used:

- PA: “Page Address”. This is the base address of the destination page. For example the very first page has the address  $C0'0000_H$ . The page 13 of the second array has the  $PA = C0'0000_H + 1 \cdot 256 \cdot 1024$  (for the array) +  $0 \cdot 4 \cdot 1024$  (for the sector) +  $13 \cdot 128$  (for the page) =  $C4'0680_H$ .
- SECPA: “Security Page Address”. This is the virtual address of a security page. It is “virtual” because SECPA is just used as argument of the command sequence to identify the security page but the physical storage of the security page is hidden.  
 Two security pages are defined:  
 SecP0: address  $C0'0000_H$ .  
 SecP1: address  $C0'0080_H$ .
- WD: “Write Data”. This is a 16-bit data word that is written into the assembly buffer.
- SA: “Sector Address”. This is the physical sector number as defined in [Figure 3-6](#) based on the address of the flash module. Two examples as clarification:
  1. Physical sector number 16 of the first array that is based on  $C0'0000_H$  is addressed with  $SA = C0'0000_H + 16 \cdot 4 \cdot 1024 = C1'0000_H$ .
  2. The second 256 KB array has the base address  $C4'0000_H$  (as shown in [Table 3-1](#)). So its physical sector number 3 has the  $SA = C4'0000_H + 3 \cdot 4 \cdot 1024 = C4'3000_H$ .
- PWD: “Password”. This is a 64-bit password. It is transferred in 4 16-bit data words  $PWD0 = PWD[15:0]$ ,  $PWD1 = PWD[31:16]$ ,  $PWD2 = PWD[47:32]$  and  $PWD3 = PWD[63:48]$ .
- Address XX followed by two hexadecimal digits, for example “XXAA<sub>H</sub>”. If the command targets a certain flash module the XX must be translated to its base address. So “XXAA<sub>H</sub>” means  $C0'00AA_H$  for all commands addressing flash 0,  $C4'00AA_H$  for flash 1 and  $C8'00AA_H$  for flash 2. If a command (e.g. “Clear Status”) addresses the complete flash memory the base address of flash module 0 must be used.
- Data XX followed by two hexadecimal digits, e.g. XXA5<sub>H</sub>. This is a “don’t care” data word where only the low byte must match a certain pattern. So in this example all data words like 12A5<sub>H</sub> or 79A5<sub>H</sub> can be used.
- MR: “Margin”. This 8-bit number defines the read margin. MR can take the values 00<sub>H</sub> (normal read), 01<sub>H</sub> (hard read 0), 02<sub>H</sub> (alternate hard read 0), 05<sub>H</sub> (hard read 1), 06<sub>H</sub> (alternate hard read 1). All other values of MR are reserved.

## **Reset to Read**

*Arguments:* –

*Definition:*

MOV XXAA<sub>H</sub>, XXF0<sub>H</sub>

*Timing:* One cycle command that does not set any “BUSY” flags. But note that an immediately following write access to the IMB Core is stalled for a few clock cycles during which the IMB Core is busy with aborting a previous command.

*Description:* The internal command state machine is reset to initial state and returns to read mode. An already started programming or erase operation is not affected and will be continued (the “**Reset to Read**” command — i.e. all commands — will anyhow not be accepted while the IMB Core is busy).

The “**Reset to Read**” command is a single cycle command. It can be used during a command sequence to reset the command interpreter and return the IMB Core into its initial state. It clears also all error flags in the Flash Status Register IMB\_FSR and an active page mode is aborted. “**Reset to Read**” can not be used to abort an active command mode. When at least one flash module is busy this command is rejected with SQER<sup>1)</sup>.

This command clears: PROER, PAGE, SQER, OPER, ISBER, IDBER, DSBER, DDBER.

## **Clear Status**

*Arguments:* –

*Definition:*

MOV XXAA<sub>H</sub>, XXF5<sub>H</sub>

*Timing:* 1-cycle command that does not set any busy flags.

*Description:* The flags OPER, SQER, PROER, ISBER, IDBER, DSBER, DDBER in Flash status register are cleared. Additionally, the process status bits (PROG, ERASE, POWER, MAR) are cleared.

This command must not be issued when any of the flash modules is in command mode. In this case it is ignored and a sequence error is reported.

## **Change Read Margin**

*Arguments:* MR

---

1) In the XE16xyM there is one exception to this rule: when one flash module is busy with program or erase and the FAPI has received some but not all command cycles of a concurrently executable command sequence (“Erase Sector”, “Erase Page”, “Enter Page Mode”, “Load Page Word”, “Program Page”) then a Reset to Read is performed without issuing a sequence error.

*Definition:*

```
MOV XXAAH, XXB0H
MOV XX54H, XXMRH
```

*Timing:* 2-cycle command that sets “BUSY” of the addressed flash module for around 30 micro seconds.

*Description:* This command sequence changes the read margin of one flash module. The address XX of the second move identifies the targeted flash module. The flash module needs some time to change its read voltage. During this time BUSY is set and this flash module cannot be accessed. The other flash modules stay readable.

The argument “MR” defines the read margin:

- 00<sub>H</sub>: normal read margin.
- 01<sub>H</sub>: hard read 0 margin.
- 02<sub>H</sub>: alternate hard read 0 margin.
- 05<sub>H</sub>: hard read 1 margin.
- 06<sub>H</sub>: alternate hard read 1 margin.
- Other values: reserved.

For understanding the read margins please refer to **“Margin Reads” on Page 3-42**.

This command must not be issued when the flash memory is in page mode or any of the flash modules is in command mode. In this case it is ignored and a sequence error is reported.

*Note:* As noted in **“Margin Control” on Page 3-69** the command sequences **“Program Page”**, **“Erase Sector”**, **“Erase Page”** and **“Erase Security Page”** reset the read margin back to 00<sub>H</sub>, i.e. to the normal read margin. The same happens in case of a flash wake-up.

## **Enter Page Mode**

*Arguments:* PA

*Definition:*

```
MOV XXAAH, XX50H
MOV PA, XXAAH
```

*Timing:* 2-cycle command that sets “BUSY” of the addressed flash module for around 20 clock cycles<sup>1)</sup>.

*Description:* The page mode is entered to prepare a page programming operation on page address PA. (Write data are accepted only with the **“Load Page Word”** command.)

1) When this command is used to abort a page mode of an other flash module the duration increases to around 30 clock cycles.

## Memory Organization

With this command, the IMB Core initializes the write pointer of its block assembly register to zero so that it points to the first word. The page mode is indicated in the status register IMB\_FSR\_BUSY with the PAGE bit, separately for each flash module. The page mode and the read mode are allowed in parallel at the same time and in the same flash module so the flash module stays readable. When the addressed page PA is read the content of the flash memory is delivered. The page mode can be aborted and the related PAGE bit in IMB\_FSR\_BUSY be cleared with the **“Reset to Read”** command. A new **“Enter Page Mode”** command during page mode aborts the actual page mode, which is indicated with the error flag SQER, and restarts a new page operation. So as mentioned above only one of the flash modules can be in page mode at a time. If one of the erase commands or the **“Change Read Margin”** command are received while in page mode it is ignored and a sequence error is reported.

The page mode can be entered in one flash module while others are busy with executing a user data erase or program command, i.e. not while programming or erasing security pages or other blocking sequences.

If write protection is installed for the sector to be programmed, the **“Enter Page Mode”** command is only accepted when write protection has before been disabled using the unlock command sequence **“Disable Write Protection”** with four passwords. If global write protection is installed with read protection, also the command **“Disable Read Protection”** can be used if no sector specific protection is installed. If write protection is not disabled when the **“Enter Page Mode”** command is received, the command is not executed, and the protection error flag PROER is set in the IMB\_FSR\_PROT.

*Note: In previous device families (e.g. XC16x) the “Enter Page Mode” did not set “BUSY”. In these devices the “Load Page Word” could be sent directly after issuing “Enter Page Mode”. In XE16xyM it must be waited until “BUSY” clears before sending the “Load Page Word” command sequence.*

### Enter Security Page Mode

*Arguments:* SECPA

*Definition:*

```
MOV XXAAH, XX55H
MOV SECPA, XXAAH
```

*Timing:* 2-cycle command that sets “BUSY” of flash module 0 for around 100 clock cycles.

*Description:* This command is identical to the **“Enter Page Mode”** command (see above), with the following exceptions: The addressed page (SECPA) belongs to the security pages of the flash memory and not to the user flash range. This command can only be executed when neither flash write protection nor read protection are active (RPA = 0 and WPA = 0), otherwise it fails with PROER.

This command is refused with SQER when any of the flash modules is in command mode.

The use of this command to install passwords and to disable them again is described in **“Protection Handling Details” on Page 3-45**.

## Load Page Word

*Arguments:* WD

*Definition:*

MOV XXF2<sub>H</sub>, WD

*Timing:* 1-cycle command that does not set any “BUSY” flags. But note that an immediately following write access to the IMB Core or read from the flash memory is stalled for a few clock cycles if it arrives while the IMB Core is busy with copying its block assembly register content into the flash module assembly buffer. During this stall time the CPU can not perform any action! So either the user software can accept this stall time (which must be taken into account for the worst-case interrupt latency) or the software must avoid the blocking accesses.

*Description:* Load the IMB Core block assembly register with a 16-bit word and increment the write pointer. The 128 byte assembly buffer (i.e. a complete page) is filled by a sequence of 64 “Load Page Word” commands. The word address is not determined by the command but the **“Enter Page Mode”** command sets a write word pointer to zero which is incremented after each **“Load Page Word”** command.

This (sequential) data write access to the block assembly register belongs to and is only accepted in Page Mode. The command address of this single cycle command is always the same (F2<sub>H</sub>). These low order address bits also identify the **“Load Page Word”** command and the sequential write data to be loaded into the block assembly register. The high order bits XX should address the target page. The IMB Core takes always the page address that was used by the last **“Enter Page Mode”** command.

When the 128-bit block assembly register of the IMB Core is filled completely after 8 **“Load Page Word”** commands the IMB Core calculates the 9 ECC bits and transfers the block into the assembly buffer of the flash module. After that it sets the write pointer of the block assembly register back to zero. The following 8 **“Load Page Word”** commands fill again the block. After all 8 blocks are filled the **“Program Page”** command can be used to trigger the program process that transfers the assembly buffer content into the flash array.

While the IMB Core transfers the completed block assembly register to the flash module it can not accept new data for a few cycles. A **“Load Page Word”** command arriving during this time is stalled by the IMB Core.

If **“Program Page”** is called before all blocks of the assembly buffer have received new data then the remaining bits are cleared.

## **Memory Organization**

If more than 8 times 8 commands are used the additional data is lost. The overflow condition is indicated by the sequence error flag, but the execution of a following “**Program Page**” command is not suppressed (the page mode is not aborted).

When a “**Load Page Word**” command is received and the flash is not in page mode, a sequence error is reported in IMB\_FSR\_OP with SQER flag. In case of a new “**Enter Page Mode**” command or a “**Reset to Read**” command during page mode, or in case of an Application Reset, the write data in the assembly buffer is lost. The current page mode is aborted and in case of a new “**Enter Page Mode**” command entered again for the new address.

### **Program Page**

*Arguments:* –

*Definition:*

```
MOV XXAAH, XXA0H  
MOV XX5AH, XXAAH
```

*Timing:* 2-cycle command that sets “BUSY” of the selected flash module for the whole programming duration. The IMB Core is blocked a few clock cycles after receiving this command and again a few clock cycles before finishing the programming. Write accesses to the flash memory range to execute another command sequence during these times stall the CPU.

*Description:* The assembly buffer of the flash module is programmed into the flash array. If the last block of data was not filled completely this command finalizes its ECC calculation and copies its data into the assembly buffer before it starts the program process. The selection of the flash module and the page to be programmed depends on the page address used by the last “**Enter Page Mode**” command. The user software should always address the targeted page.

The programming process is autonomously performed by the selected flash module. The CPU is not occupied and can continue with its application.

The “**Program Page**” command is only accepted if the addressed flash module is in Page Mode (otherwise, a sequence error is reported instead of execution). With the “**Program Page**” command, the page mode is terminated, indicated by resetting the related PAGE flag and the command mode is entered and the PROG flag in the status register IMB\_FSR\_OP is activated and the related BUSY flag is set in IMB\_FSR\_BUSY.

When the program process has finished BUSY is cleared but PROG stays set. It indicates which operation has finished and will be cleared by a Power-On Reset or by “**Clear Status**”.

Read accesses to the busy flash module are not possible. Reading a busy flash module stalls until the flash module becomes ready again.



## **Memory Organization**

If write protection is active for the sector to be programmed, the “**Program Page**” command is not accepted because the Flash is not in Page Mode (see description of the “**Enter Page Mode**” command).

If the page to be programmed is a security page (accepted only in security page mode), the new protection configuration (including keywords or protection confirmation code) is valid directly after execution of this command. During its execution all commands are rejected with a sequence error.

While the IMB Core reads the new protection configuration all DMU accesses to any flash module are stalled.

### **Erase Sector**

*Arguments:* SA

*Definition:*

```
MOV XXAAH, XX80H
MOV XX54H, XXAAH
MOV SA, XX33H
```

*Timing:* 3-cycle command that sets BUSY of the addressed flash module for the whole erasing duration. The IMB Core is blocked a few clock cycles after receiving this command and again a few clock cycles before finishing the erasing. Write accesses to the flash memory range during these times stall the CPU.

*Description:* The addressed physical sector in the flash array is erased. Following data reads deliver all-zero data with correct ECC.

The erasing process is autonomously performed by the selected flash module. The CPU is not occupied and can continue with its application.

The sector to be erased is addressed by SA (sector address) in the last command cycle.

With the last cycle of the “**Erase Sector**” command, the command mode is entered, indicated by activation of the ERASE flag in IMB\_FSR\_OP and after start of erase operation also by the related busy flag in the status register IMB\_FSR\_BUSY. The BUSY flag is cleared after finishing the operation but ERASE stays set. It can be cleared by a Power-On Reset or the “**Clear Status**” command.

Read accesses to the busy flash module are not possible. Read accesses to the not busy flash module are especially supported. Reading a busy flash module stalls until the flash module becomes ready again.

If write protection is installed for the sector to be erased, the Erase Sector command is only accepted when write protection has before been disabled using the unlock command sequence “**Disable Write Protection**”. If global write protection is installed with read protection, also the command “**Disable Read Protection**” can be used if no sector specific protection is installed. If write protection is not disabled when the “**Erase**

## **Memory Organization**

**Sector** command is received, the command is not executed, and the protection error flag PROER is set in the IMB\_FSR\_PROT.

This command must not be issued when the flash memory is in page mode. In this case it is ignored and a sequence error is reported.

### **Erase Page**

*Arguments:* PA

*Definition:*

```
MOV XXAAH, XX80H
MOV XX54H, XXAAH
MOV PA, XX03H
```

*Timing:* 3-cycle command that sets BUSY of the addressed flash module for the whole erasing duration. The IMB Core is blocked a few clock cycles after receiving this command and again a few clock cycles before finishing the erasing. Write accesses to the flash memory range during these times stall the CPU.

*Description:* The addressed page is erased. Following data reads deliver all-zero data with correct ECC.

With the last cycle of the **Erase Page** command, the command mode is entered, indicated by activation of the ERASE flag in IMB\_FSR\_OP and after start of erase operation also by the related BUSY flag in the status register IMB\_FSR\_BUSY. BUSY is cleared automatically after finishing the operation but ERASE stays set. It is cleared by a Power-On Reset or the **Clear Status** command.

Read accesses to the busy flash array are not possible. Read accesses to the not busy flash modules are especially supported. Reading a busy flash module stalls until the flash module becomes ready again.

If the page to be erased belongs to a sector which is write protected, the command is only executed when write protection has before been disabled (see **Erase Sector** command).

In case of using the page erase care must be taken not to exceed the drain disturb limit of the other pages of the same sector.

This command must not be issued when the flash memory is in page mode. In this case it is ignored and a sequence error is reported.

### **Erase Security Page**

*Arguments:* SECPA

*Definition:*

```
MOV XXAAH, XX80H
MOV XX54H, XXA5H
MOV SECPA, XX53H
```



*Timing:* 3-cycle command that sets BUSY of flash module 0 for the whole erasing duration.

*Description:* The addressed security page is erased.

This command is identical to the “**Erase Page**” command with the following exceptions: The addressed page (SecP0 or SecP1) belongs not to the user visible flash memory range. This command can only be executed after disabling of read protection and of sector write protection.

See “**Protection Handling Examples**” on **Page 3-52** for a detailed description of re-programming security pages.

The structure of the two security pages (SecP0 and SecP1) is described in “**Layout of the Security Pages**” on **Page 3-51**.

After erasing a security page the new protection configuration (including keywords or protection confirmation code) is valid directly after execution of this command.

While the IMB Core reads the protection configuration all DMU accesses to any flash module are stalled.

This command must not be issued when the flash memory is in page mode or any of the flash modules is in command mode. In this case it is ignored and a sequence error is reported.

## **Disable Read Protection**

*Arguments:* PWD

*Definition:*

```
MOV XX3CH, XXXXH
MOV XX54H, PWD0
MOV XXAAH, PWD1
MOV XX54H, PWD2
MOV XXAAH, PWD3
MOV XX5AH, XX55H
```

*Timing:* 6-cycle command that does not set any busy flag.

*Description:* Disable temporarily Flash read protection and — if activated — global write protection of the whole flash memory. The RPA bit in IMB\_IMBCTRH is reset.

This is a protected command sequence, using four user defined passwords to release this command or to check the programmed keywords. For every password one command cycle is required. If the second or fourth password represents the code of the “**Reset to Read**” command, it is interpreted as password and the reset is not executed. The 16-bit passwords are internally compared with the keywords out of the “Security Page 0”. If one or more passwords are not identical to their related keywords, the protected sectors remain in the locked state and a protection error (PROER) is indicated in the Flash status register. In this case, a new “**Disable Read Protection**” command or

## Memory Organization

a “**Disable Write Protection**” command is only accepted after the next Application Reset.

*Note: During execution of the “Disable Read” (or Write) Protection command a password compare error is only indicated after all four passwords have been compared with the related keywords.*

*Note: This command sequence is also used to check the correctness of keywords before the protection is confirmed in the Security Page 1. A wrong keyword is indicated by the IMB\_FSR\_PROT flag PROER.*

After correct execution of this command, the whole flash memory is unlocked and the read protection disable bit RPRODIS is set in the Flash Status Register (IMB\_FSR\_PROT). Erase and program operations on all sectors are then possible, if the flash memory was also globally write protected (WPA=1), and if they are not separately write protected. The read protection (including global write protection, if so selected) remains disabled until the command “**Re-Enable Read/Write Protection**” is executed, or until the next Application Reset (including HW and SW reset).

This command must not be issued when any of the flash modules is in command mode. In this case it is ignored and a sequence error is reported.

### Disable Write Protection

*Arguments:* PWD

*Definition:*

```
MOV XX3CH, XXXXH
MOV XX54H, PWD0
MOV XXAAH, PWD1
MOV XX54H, PWD2
MOV XXAAH, PWD3
MOV XX5AH, XX05H
```

*Timing:* 6-cycle command that does not set any busy flag.

*Description:* Disable temporarily the global flash write protection or/and the sector write protection of all protected sectors. The WPA bit in IMB\_IMBCTRH is reset.

This is a protected command sequence, using four user defined passwords to release this command (as described above for the “**Disable Read Protection**” command).

After correct execution of this command, all write-protected sectors are unlocked, which is indicated in the Flash Status Register (IMB\_FSR\_PROT) with the WPRODIS bit. Erase and program operations on all sectors are now possible, until

- The command “**Re-Enable Read/Write Protection**” is executed, or
- The next Application Reset (including HW and SW reset) is received.

This command must not be issued when any of the flash modules is in command mode. In this case it is ignored and a sequence error is reported.

## Re-Enable Read/Write Protection

*Arguments:* –

*Definition:*

MOV XX5E<sub>H</sub>, XXXX<sub>H</sub>

*Timing:* 1-cycle command that does not set any busy flags.

*Description:* Flash read and write protection is resumed.

This single-cycle command clears RPRODIS and WPRODIS. The IMB Core is triggered to restore the protection states RPA and WPA from the content of the security page 0 as defined in [Table 3-6 “Flash State” Determining RPA and WPA on Page 3-48](#). So in effect this command resumes all kinds of temporarily disabled protection installations.

This command is released immediately after execution.

This command must not be issued when any of the flash modules is in command mode. In this case it is ignored and a sequence error is reported.

### 3.10.5 Sequence Errors

A word (i.e. 16-bit) data move into the flash address range is interpreted by the command interpreter as command sequence. All byte moves are ignored and cause a sequence error which is reported by setting the bit SQER.

As soon the command interpreter detects that the data moves can't be executed as legal sequence it reports the sequence error.

*Note: Data moves addressing not implemented flash areas or powered-down flash modules don't enter the command interpreter and consequently can't cause a sequence error. Usually the next correct command sequence will cause the sequence error because it is interpreted as continuation of the previous one. So instead of checking only for the absence of SQER the other flags (e.g. PAGE, PROG, ERASE) can be further evaluated. For an example see [Section 3.10.6](#).*

Generally each data move received while at least one flash module is BUSY causes a sequence error. But in order to support concurrent execution of command sequences this is under certain conditions not done. A SQER is reported under the following conditions:

- If one of the flash modules is in command mode and the running command does not allow concurrent execution a SQER is reported immediately.
- If at least one of the flash modules is in command mode and the running command allows concurrent execution SQER is only reported when the new command targets a busy flash module.
- If at least one of the flash modules is in command mode SQER is reported as soon as a command cycle is detected that can not belong to a command sequence that allows concurrent execution (i.e. when the received data does not belong to “Enter Page Mode”, “Load Page Word”, “Program Page” or “Erase Page”).

The concurrency issues are summarized in [Table 3-5](#).

**Table 3-5 Concurrency Issues**

<b>New sequence while any module is in mode:</b>	<b>Page Mode</b>	<b>Busy with normal erase or program</b>	<b>Busy with blocking sequence<sup>1)</sup></b>
Reset to Read	Resets page mode	SQER <sup>2)</sup>	SQER <sup>2)</sup>
Enter Page Mode	SQER and Re-enters page mode	OK <sup>3)</sup>	SQER
Enter Sec. Page Mode	SQER and Re-enters page mode	SQER	SQER
Load Page Word	OK	OK in page mode	SQER/– <sup>4)</sup>
Program Page	OK	OK in page mode	SQER/– <sup>4)</sup>
Erase Page/Sector	SQER	OK <sup>3)</sup>	SQER
Erase Sec. Page	SQER	SQER	SQER
*Protection	OK	SQER	SQER
Clear Status	OK	SQER	SQER
Change Read Margin	SQER	SQER	SQER

1) "Blocking sequences" are: "Erase Security Page", "Program Page" for a security page, "Change Read Margin", "Enter Page Mode", "Enter Security Page Mode" only while these set busy.

2) As described in ["Reset to Read"](#) on [Page 3-28](#) there is one exception to this rule.

3) If the new command sequence targets a different flash module that is in read mode else SQER.

4) Situation can not occur because "Program Page" is only allowed in page mode and page mode could not be entered.

Other conditions that cause a sequence error were mentioned above in the command descriptions.

### **3.10.6 Instructions for Executing Program and Erase Jobs**

### **Concurrently**

All flash modules<sup>1)</sup> can be programmed and erased concurrently. This is however an exceptional case for high-speed flash programming. In the normal case at most one flash module shall be busy while the others can be read.

The limitations reported above in the command sequence descriptions enforce certain behavior for concurrent processes:

- A programming task shall be started in one not interrupted sequence: “Enter Page Mode”, then 64 “Load Page Word” and finally the “Program Page”. No other command sequence on any other flash module shall interrupt this sequence.
- All security page handling shall be done while all flash modules are in read mode.
- Clearing of error and status flags is as well only possible when all flash modules are in read mode. An exception is the flash module specific handling via **IMB\_ECC\_STAT**.
- The IMB Core can only finish an ongoing program or erase task successfully when it is not busy with interpreting a command sequence (i.e. the busy of the ongoing tasks is only cleared when the IMB Core is ready to accept a new command sequence and no new command sequence has been started but not completed).

So the required sequence for programming flash modules concurrently is as follows:

1. Send the “Erase Sector” command sequence to each flash module.
2. Wait until all “BUSY” flags are cleared. During this time the data for programming can be read from external.
3. Send “Enter Page Mode”, 64 “Load Page Word” and the “Program Page” to the first flash module. Continue this sequence with the other flash modules.
4. Wait until all “BUSY” flags are cleared. This time can be used to read the data for programming the next pages from external.
5. Verify the programmed data of all flash modules.
6. Continue the steps 3 to 5 until all pages of the erased sectors are programmed.
7. Continue the steps 1 to 6 until all sectors are programmed.

The recommend sequence which detects incorrect sequences as early as possible is as follows:

1. “Clear Status” and check that SQER is 0.
2. Send the “Erase Sector” command sequence to each flash module. Check for SQER after issuing each sequence.
3. Wait until all “BUSY” flags are cleared. During this time the data for programming can be read from external.
4. Check for SQER which would indicate an incorrectly issued command sequence.

---

1) Additional constraints may apply due to power supply and other device specific reasons. The allowed concurrent processes (including read) are described in the data sheet. This section describes only the logic hardware capabilities.

### **Memory Organization**

5. "Clear Status" and check again for SQER. If SQER would be set after "Clear Status" a previous "Erase Sector" hasn't been completed.
6. Send "Enter Page Mode", check if the PAGE flag was set and check if SQER stays 0, send the 64 "Load Page Word" and the "Program Page" to the first flash module. Check if the PAGE flag is cleared and SQER stays cleared after "Program Page" is accepted. Continue this sequence with the other flash modules.
7. Wait until all "BUSY" flags are cleared. This time can be used to read the data for programming the next pages from external.
8. Verify the programmed data of all flash modules.
9. Continue the steps 6 to 8 until all pages of the erased sectors are programmed.
10. Continue the steps 1 to 9 until all sectors are programmed.

### **3.10.7 Data Integrity**

This section describes means for detecting and preventing the inadvertent modification of data in the flash memory.

#### **3.10.7.1 Error Correcting Codes (ECC)**

With very low probability a flash cell can become disturbed or lose its data value faster than specified. In order to reach the defined overall device reliability each 128-bit block of flash data is accompanied with a 9-bit ECC value. This redundancy supplies SEC-DED capability, meaning “single error correction and double error detection”. All single bit errors are corrected (and the incident is detected), all double bit errors are detected and even most triple bit errors are detected but some of these escape as valid data or corrected data.

A detected error is reported in the register **IMB\_FSR\_PROT** and **IMB\_ECC\_STAT**. Software can select which type of error should trigger a trap by the means of register **IMB\_INTCTR**. In the system control further means exist to modify the handling of errors (see “**SCU Trap Control Registers**” on Page 8-240). The enabled trap requests by the flash module are handled there as “Flash Access Trap”. In case of a double-bit error the read data is always replaced with a dummy data word.

#### **3.10.7.2 Aborted Program/Erase Detection**

Where the ECC should protect from intrinsic failures of the flash memory that affect usually only single bits; an interruption of a running program or erase process might cause massive data corruption:

- The erase process programs first all cells to 1 before it erases them. So depending on the time when it is interrupted the data might be in a different state. This can be the old data, all-one, a random value, a weak all-zero or finally all-zero.
- The program process programs all bits concurrently from 0 to 1. If it is interrupted not all set bits might read as 1 or contain a weak 1.

The register **IMB\_FSR\_OP** contains the bits ERASE and PROG. These bits stay set until the next “**Clear Status**” command or Power-On Reset. So if an erase or program process is interrupted by an Application Reset one of these bits is still set which allows to detect the interruption. It lies in the responsibility of the software to send the “**Clear Status**” command after a finalized program/erase process to enable this evaluation.

Another possible measure against aborted program/erase processes is to prevent resets by configuring the SCU appropriately.

If a program or erase process was aborted by a Power-On Reset (e.g. due to a power failure) there do not exist reliable means to detect this by reading the affected flash range. Even with margin reads an early or late aborted process might go unnoticed although it might in the long-term affect reliability.

Therefore the application must ensure that flash processes can perform uninterrupted and under the defined operating conditions, e.g. by early brown-out warning that prevents the software from starting flash processes.

After a flash process aborted the affected address range must be erased and re-programmed.

### **3.10.7.3 Margin Reads**

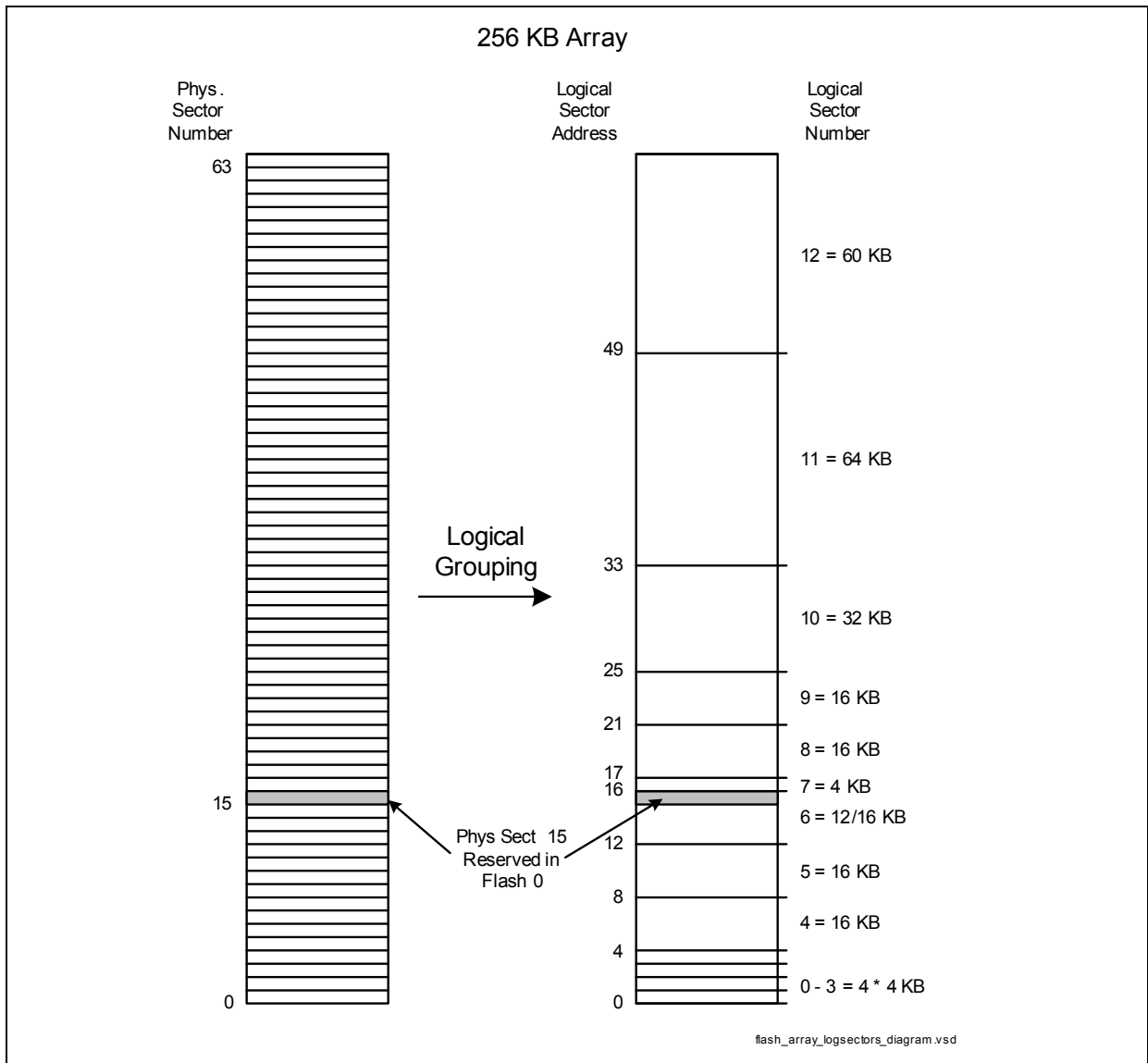
Margin reads can be used to verify that flash data is readable with a certain margin. This is typically used as additional check directly after end-of-line programming. As explained above this is not a reliable method for detecting interrupted program or erase processes but the probability of detecting such cases can be increased.

Reading with “hard read 0 margin” returns weak 0s as 1s and reading with “hard read 1 margin” returns weak 1s as 0s. Changing the read margin is done with the command sequence “**Change Read Margin**” and is reported by the status register “**IMB\_MAR0**”.

### **3.10.7.4 Protection Overview**

The flash memory supports read and write protection for the whole memory and separate write protection for each logical sector. The logical sector structure is depicted in **Figure 3-6** for a 256 KB array. The logical sector structure of a 64 KB array is equivalent, only the logical sector number 7 to 12 do not exist.





**Figure 3-6 Logical Sectors**

If read protection is installed and active, any flash read access is disabled in case of start after reset from external memory or from internal RAM. Debug access is as well disabled and thus the execution of injected OCDS instructions. In case of start after reset in internal flash, all flash access operations are controlled by the flash-internal user code and are therefore allowed, as long as not especially disabled by the user, e.g. before enabling the debug interface.

Per default, the read protection includes a full (global) flash memory write protection covering all flash modules. This is necessary to eliminate the possibility to program a dump routine into the Flash, which reads the whole Flash and writes it out via the external bus or a serial interface. Program and erase accesses to the flash during active read protection are only possible, if write protection is separately disabled. Flash write

## **Memory Organization**

and read protection can be temporarily disabled, if the user authorizes himself with correct passwords.

The device also features a sector specific write protection. Software locking of flash memory sectors is provided to protect code and data. This feature disables both program and erase operations for all protected sectors. With write protection it is supported to protect the flash memory or parts of it from unauthorized programming or erase accesses and to provide virus-proof protection for all sectors.

Read and write protection is installed by specific security configuration words which are programmed by the user directly into two “Security Pages” (SecP0/1). After any reset, the security configuration is checked by the command state machine (IMB Core) and installations are stored (and indicated) in related registers. If any protection is enabled also the security pages are especially protected.

For authorization of short-term disabling of read protection or/and of write protection a password checking feature is provided. Only with correct 64-bit password a temporary unprotected state is taken and the protected command sequences are enabled. If not finished by the command “**Re-Enable Read/Write Protection**”, the unprotected state is terminated with the next reset. Password checking is based on four 16-bit keywords (together 64 bits) which are programmed by the user directly into the “Security Page 0” (SecP0).

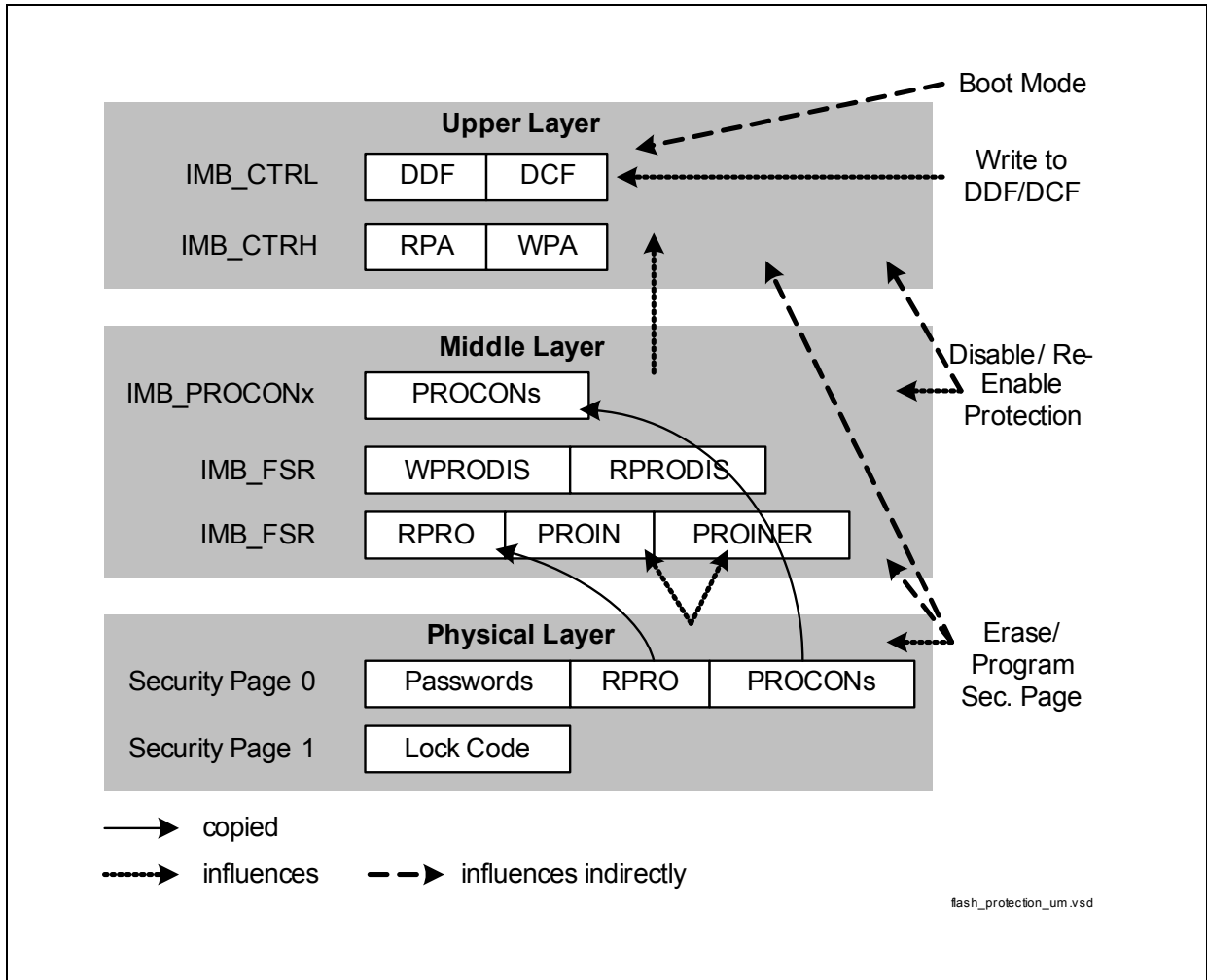
Special support is provided to protect also the protection installation itself against any stressing or beaming aggressors. The codes of configuration bits are selected, so that in case of any violation in the flash array, on the read path or in registers the protected state is taken per default. In registers and security pages, protection control bits are coded always with two bits, having both codes, “00<sub>B</sub>” and “11<sub>B</sub>” as indication of illegal and therefore protected state.

### **3.10.8 Protection Handling Details**

As shortly described in **“Protection Overview” on Page 3-42** the flash memory can be in different protection states. The protection handling can be separated into different layers that interact with each other (see **Figure 3-7**).

- The lowest layer consists of the physical content of the security pages SecP0 and SecP1. This information is used to initialize the protection system during startup.
- The next layer consists of registers that report the state of the physical layer (IMB\_PROCONx) and the protection state (IMB\_FSR\_PROT). The protection state can be temporarily changed with command sequences which is reflected in the IMB\_FSR\_PROT.
- The highest layer is represented by 4 fields of the IMB\_IMBCTR register. These fields define the protection rights of the customer software (are read or write accesses currently allowed or not).

The IMB Core controls the protection state of all connected flash modules centrally. In this position it can supervise all accesses that are issued by the CPU.



**Figure 3-7 Protection Layers**

### 3.10.8.1 The Lower Layer “Physical State”

After reset the protection state of the device is restored from the following information:

- The security page 1 contains a “lock code”. This consists of two words of data (32 bits). If it has the value AA55AA55<sub>H</sub> then security page 0 determines the protection state. Otherwise (i.e. the lock code was not found) the device is in the “non-protected state”. The content of the security page 0 is still copied into the registers as described in **“The Middle Layer “Flash State”” on Page 3-47** but their values are ignored in the non-protected state.
- The security page 0 contains the RPRO double bit, the write protection bits SnU and 4 passwords. If the field RPRO contains a valid 01<sub>B</sub> or 10<sub>B</sub> entry the page is valid and the device is in the “protection installed state”. The page content determines the security settings after startup. If SecP0 contains an invalid RPRO entry the device is in the “errored protection” state.

To summarize: the content of the security pages determines if the device is in the “non-protected state”, “protection installed state” or “errored protection state”. These states are reflected in the register settings of the next layer.

The device is usually delivered in the “non-protected state”.

The exact layout of the security pages is described in **“Layout of the Security Pages” on Page 3-51**.

### **3.10.8.2 The Middle Layer “Flash State”**

The middle layer consists of the registers IMB\_PROCONx and IMB\_FSR\_PROT and commands that manipulate them and the content of the security pages.

During startup the physical state is examined by the IMB Core and it is reflected in the following bit settings of IMB\_FSR\_PROT:

- “non-protected state”: PROIN = 0, PROINER = 0.
- “protection installed state”: PROIN = 1, PROINER = 0.
- “errored protection state”: PROIN = 0, ROINER = 1.

The fourth possible setting PROIN=1 and PROINER=1 is invalid and can not occur.

The IMB\_PROCONx registers are initialized during startup with the content of the security page 0. The bits DSBERR and DDBERR indicate if an ECC error occurred. The customer software has thus the possibility to detect disturbed security pages and it can refresh their content.

### **Commands**

Other bits of the IMB\_FSR\_PROT: RPRODIS, WPRODIS, PROER can be manipulated with command sequences and define together with the other bits the protection effective for the next layer. All three bits are 0 after system startup.

The command **“Disable Read Protection”** sets RPRODIS to 1 if the correct passwords that are stored in SecP0 are supplied. If incorrect passwords are entered the bit PROER is set and RPRODIS stays unchanged. As protection against “brute force attacks” that search the correct password the password detection is locked. So after supplying the first incorrect password all following passwords even the correct ones are rejected with PROER. This state is only left by an Application Reset or by erasing SecP0.

The disabled protection can be enabled again by the Application Reset or by the command **“Re-Enable Read/Write Protection”** which clears RPRODIS again.

The bit PROER can be reset by an Application Reset or by the commands **“Reset to Read”** and **“Clear Status”**.

The command **“Disable Write Protection”** sets WPRODIS to 1 if the correct passwords are supplied. It behaves analog to RPRODIS as described above.

The command **“Re-Enable Read/Write Protection”** clears RPRODIS and WPRODIS.

## Memory Organization

The commands “**Enter Page Mode**”, “**Enter Security Page Mode**”, “**Erase Page**”, “**Erase Security Page**” and “**Erase Sector**” set PROER if the write access to the addressed range is not allowed. If a write access is allowed or not is determined by the next level.

**Table 3-6** summarizes how the “Flash State” of protection determines the RPA and WPA fields of IMB\_IMBCTRH. For the double bits a short notation is used here and in the following sections: 1 means active, 0 means inactive, ‘#’ means invalid and ‘–’ means do not care including invalid states. The symbol ‘|’ means logic or.

**Table 3-6 “Flash State” Determining RPA and WPA**

IMB_FSR. PROI N	IMB_FSR. PROI NER	IMB_FSR. RPR O	IMB_FSR. RPR ODIS	IMB_FSR. WPR ODIS	Resulting Security Level in RPA and WPA
0	0	–	–	–	Non-protected state: RPA = 0, WPA = 0.
1	0				Protection installed state (possibly disabled, see below):
		0	–	0	RPA = 0, WPA = 1.
		0	0	1	RPA = 0, WPA = 0.
		1   #	0	0	RPA = 1, WPA = 1.
		–	1	1	RPA = 0, WPA = 0 (all disabled).
		1   #	0	1	RPA = 1, WPA = 0.
		1   #	1	0	RPA = 0, WPA = 1.
0	1				Errored protection state (see below):
		–	0	0	RPA = 1, WPA = 1.
		–	0	1	RPA = 1, WPA = 0.
		–	1	0	RPA = 0, WPA = 1.
		–	1	1	RPA = 0, WPA = 0.

### 3.10.8.3 The Upper Layer “Protection State”

This layer consists mainly of the 4 fields DCF, DDF, WPA and RPA of the IMB\_IMBCTRH register. These determine the effective protection state together with registers of the lower layers. Some of the above mentioned command sequences directly influence these fields as well. In order to increase the resistance against beaming or power supply manipulation all 4 fields are coded with 2 bits. Generally “01”

## Memory Organization

means active, “10” inactive and the two other states “00” and “11” are invalid and are recognized as “attacked” state.

### Effective Security Level

The effective security level based on these 4 double-bits is summarized in [Table 3-7](#) and [Table 3-8](#). For the double bits the same short notation is used as before: 1 means active, 0 means inactive, ‘#’ means invalid and ‘–’ means do not care including invalid states.

**Table 3-7 Effective Read Security**

RPA	DCF	DDF	Security Level
0	–	–	No read protection.
1   #	0	0	No read protection.
	–	1   #	Data reads prohibited.
	1   #	–	Code fetches prohibited.

**Table 3-8 Effective Write Security**

WPA	RPA	Security Level
0	–	No write protection
1   #	1   #	Global write protection.
1   #	0	Sector specific write protection depending on IMB_PROCONx.

To summarize:

- Read protection is always globally affecting the whole flash memory range. Code fetches and data reads can be separately controlled.
- Write protection can be global when the read protection is effective or it can be specific for each logical sector.

The lower and the middle security layers determine how the 4 effective IMB\_IMBCTR fields are preset, changed and how software can access them. This is discussed in the following paragraphs.

### Initialization of the Effective Security Level

After Application Reset protection is activated so that RPA, WPA, DDF and DCF are set. During startup the IMB Core determines the stored security level as described in [“The Lower Layer “Physical State”” on Page 3-46](#) and sets IMB\_FSR\_PROT.PROIN and IMB\_FSR\_PROT.PROINER and IMB\_PROCONx as described in [“The Middle Layer](#)

**“Flash State”” on Page 3-47.** The IMB Core further initializes the IMB\_IMBCTRH fields RPA and WPA according to the rules of **Table 3-6**.

The bits DDF and DCF of the IMB\_IMBCTRL are not initialized by the IMB Core. During system startup they are initialized depending on the startup condition. If code fetching starts in the flash memory then they are set to the inactive state. In all other cases they are activated to prevent read access to the flash memory without proving password knowledge.

### **Changing the Effective Security Level**

During run-time the effective security level can be changed. This can be done by directly writing to the IMB\_IMBCTRL register or indirectly by changing the bits of the middle layer by commands as **“Disable Write Protection”** or even double indirectly by changing the content of the security pages which changes bits in the middle layer and influences the effective security level.

Writing directly to IMB\_IMBCTRL:

- DCF and DDF can be deactivated only if RPA is inactive. They can always be activated.

Indirectly by using a command sequence:

- A successful **“Disable Read Protection”** sets RPRODIS and clears RPA.
- A successful **“Disable Write Protection”** sets WPRODIS and clears WPA.
- **“Re-Enable Read/Write Protection”** clears RPRODIS and WPRODIS and sets RPA and WPA according to **Table 3-6** depending on PROIN, PROINER and RPRO.

Double indirect by changing security pages. After executing a command sequence that changed the content of a security page the IMB Core immediately reads back the pages and determines all resulting security data as described for system startup in **“Initialization of the Effective Security Level” on Page 3-49**. The examples in **“Protection Handling Examples” on Page 3-52** will show how this can be used for installing and removing protection or changing passwords.

#### **3.10.8.4 Reaction on Protection Violation**

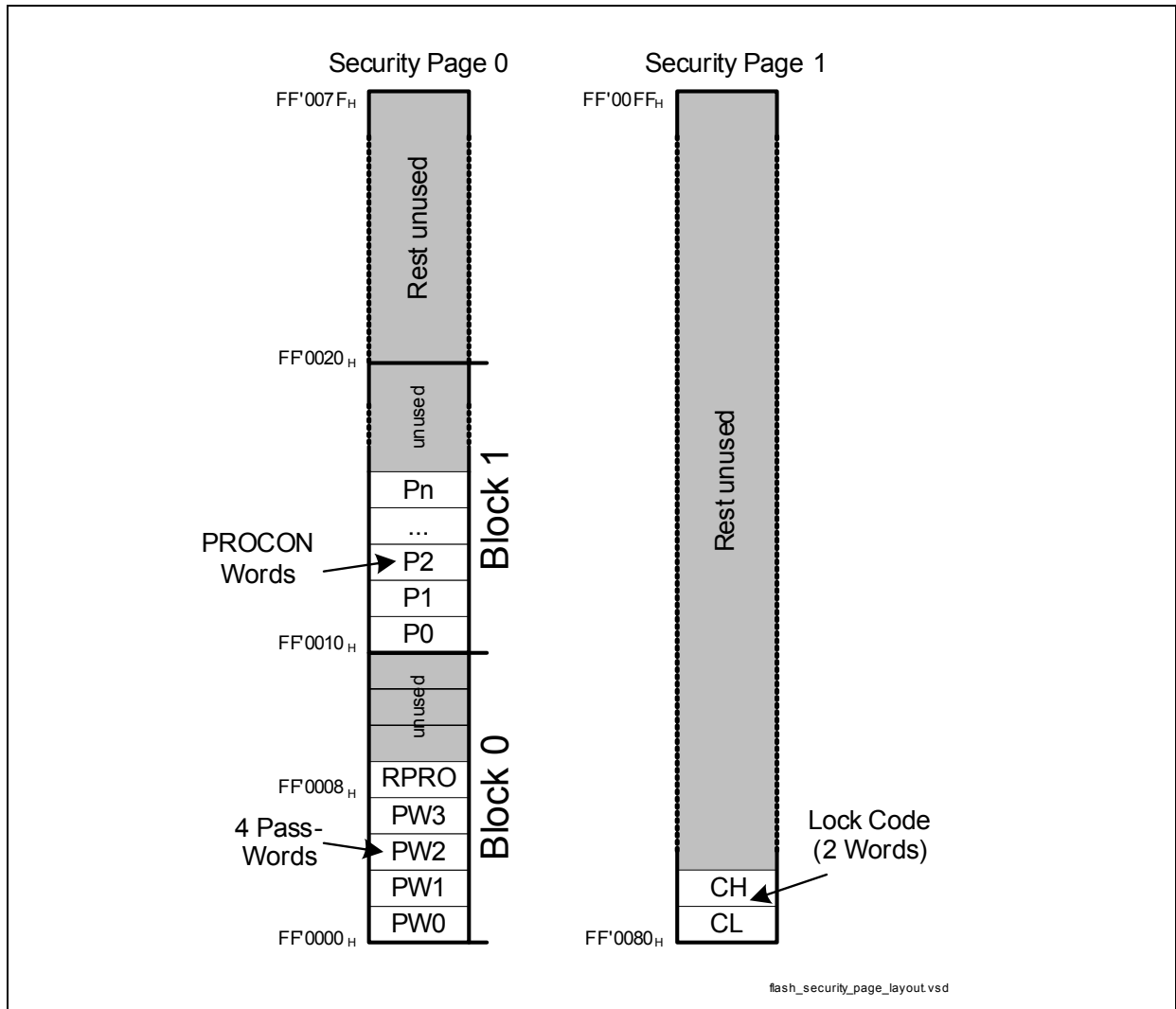
If software tries to violate the protection rules the following happens:

- Reading data when read protection is effective: The bit IMB\_FSR\_PROT.PROER is set and the Flash access trap can be triggered via the SCU if IMB\_INTCTR.DPROTRP is 0. Default data is delivered.
- Fetching code when read protection is effective: the trap code “TRAP 15<sub>D</sub>” is delivered instead.
- Programming or erasing memory ranges when they are write protected: PROER is set.



### 3.10.8.5 Layout of the Security Pages

The previous sections just mentioned the content of the security pages. This section depicts their exact layout. **Figure 3-8** depicts symbolically the layout of the security pages 0 and 1.



**Figure 3-8 Layout of Security Pages**

Generally the 16-bit words are stored as always in the XE16xyM in little endian format.

- The PWx words contain the passwords.
- The double bit RPRO is stored as in the related ISFR **IMB\_FSR\_PROT** in the bits 15 and 14. The other bits of this word are unused and should be kept all-zero.
- The PROCON data is stored as defined in the **IMB\_PROCONx (x=0-3)** ISFR.
- The lock code consists of the two words CL and CH. Both contain "AA55<sub>H</sub>" to form the correct lock code.

All bytes of the used blocks of the security pages (block 0 and 1 of SecP0 and block 0 of SecP1) are to be considered as “reserved” and must be kept erased, i.e. with all-zero content. The unused blocks of the security pages (blocks 2 to 7 of SecP0 and blocks 1 to 7 of SecP1) shall be programmed with all-one data.

### **3.10.9 Protection Handling Examples**

Some examples on how to work with the protection system.

#### **Delivery State**

The device is delivered in the “non-protected state”.

Security page 1 is erased (so it does not contain the “lock code” AA55AA55<sub>H</sub>).

Security page 0 is erased and so “invalid” but because SecP1 is erased this data is anyhow not evaluated. Only its content is copied into corresponding the registers.

During startup the bits DDF and DCF are set depending on the start mode but as RPA and WPA are inactive all accesses to the flash memory are allowed.

The data sectors of the flash memory are delivered in the erased state as well. All sectors can be programmed. After uploading the software the customer can install write and read protection.

#### **First Time Password Installation**

In order to install a password generally the lock code in SecP1 has to be erased. In this case the code is not present.

After that SecP0 must be erased with “**Erase Security Page**” in order to be able to change RPRO. Erasing SecP0 clears RPRO to “00<sub>B</sub>” which is an invalid state. After finishing the erase command the IMB Core restores the IMB\_FSR\_PROT and IMB\_IMBCTRH fields from the flash data.

Because no lock code is present in SecP1 the invalid state of RPRO has no effect on the user visible protection. Still all parts of the flash memory can be written.

The second step is to program the information of SecP0 with the required security information. Again the IMB Core reads immediately back the stored data and initializes the security system. As SecP1 still does not contain the lock code the device stays in the “non-protected” mode.

The security pages cannot be read directly by customer software. The data programmed into SecP0 can therefore only be verified indirectly. The data of the RPRO and SnU fields can be checked by reading the IMB\_PROCON and IMB\_FSR\_PROT registers. The passwords can be verified with the command “**Disable Read Protection**”. If the password does not match the bit PROER is set. But because of the erased SecP1 the flash memory stays writable. So after erasing SecP0 the correct password can be programmed again.

## **Memory Organization**

After the SecP0 was verified successfully SecP1 gets programmed with the lock code AA55AA55<sub>H</sub> which enables the security settings of SecP0.

Because the password validation left RPRODIS set the command “**Re-Enable Read/Write Protection**” must be used to finally activate the new protection.

### **Changing Passwords or Security Settings**

Changing the passwords is a delicate operation. The interrelation of the two security pages must be kept in mind.

Usually in the protected state the SecP1 contains the lock code. First write protection must be disabled with the correct passwords. Then the lock code in SecP1 is erased. If this operation was successful PROIN will be cleared by the IMB Core. Now SecP0 can be safely erased.

From this point on the security pages are in the factory delivery state and the new passwords and security settings can be installed as described above.

***Attention: The number of times a security page may be changed is noted in the data sheet.***

### **3.10.10 EEPROM Emulation**

The flash memory of the XE16xyM is used for three purposes:

1. Storage of program code. Updates happen usually very seldom. The main criteria to be fulfilled is a retention of the life-time of the product.
2. Storage of constant data: this data is stored together with program code. So this data is very seldom updated. Endurance is of no issue here but retention identical to the code memory is required.
3. Data updated during run-time: this might be data with a very high frequency of updates like a mileage counter or access keys for key-less entry. Other data might be changed only in case of failures and other data might only be transferred from RAM to non-volatile memory before the system is powered down.

Especially for the third type of data the non-volatile memory needs EEPROM like characteristics:

- Fine program/erase granularity which is in EEPROMs typically 1 byte.
- Higher endurance than the intrinsic endurance of flash cells.
- Short program and erase duration per byte. Especially for storing data in an emergency (e.g. power failure) short latencies might be required.

A basic requirement for changing data during run-time is that code execution can still resume, especially interrupt requests must still be serviced. This requirement is fulfilled in the XE16xyM because all four flash modules work independently. If one is busy with program or erase then code can still be executed from the other.

The other requirements are more difficult to fulfill because the XE16xyM does not have an EEPROM available but only the flash memory with the already frequently mentioned limitations: big program/erase granularity, moderately long program/erase duration, limited cell endurance with reduced retention at high number of program/erase cycles, pages not isolated but affected by drain disturbs.

In order to alleviate these effects on run-time storage of data software is used to emulate EEPROM. There is quite a number of algorithms for efficiently using flash memory as EEPROM. The following section describes one (the most simple) of these algorithms.

It should be noted that the XE16xyM does not offer the customer any hardware means for EEPROM emulation. All of the following must be realized by software.

#### **3.10.10.1 The Traditional EEPROM Emulation**

The key point is to solve the limited endurance by storing data in N different physical places. In XE16xyM the algorithm could use N sequential pages or groups of pages. If data is currently stored in the page group "x" then the next program happens to the page group " $(x+1) \bmod N$ ".

After boot up the last correct page group must be found. This could be done by either evaluating a counter (from 0 to  $2*N-1$ ) or the old entries are invalidated by erasing the

**Memory Organization**

page after programming the new one. Additionally a CRC check could be performed over the group.

As all involved pages are re-used cyclically the endurance from customer perspective is increased by the factor N. N must be chosen high enough to fulfill endurance and retention requirements. Disturbs in the group of N pages are no issue because they incur at most N-1 disturbs before they get written with new data. Care must be taken however if one sector accommodates different groups of pages with different update behavior. In this case the updates of one group of pages could exceed the disturb limits of the other group. So generally one sector should be used only by one such EEPROM cyclic buffer. The algorithm keeps the old data until the new data is verified so power failure during programming can only destroy the last update but the older data is still available. There are still some issues with power failure that need special treatment:

- Power is cut during programming: the following boot-up might find an apparently correctly programmed page. However the cells might be not fully programmed and thus have a much lower retention or the read data is unstable (e.g. changing operating conditions cause read errors).  
If the power is cut early the page can appear as erased although some cells are partly programmed. When programming different data to this apparently erased page read errors might occur.
- Power is cut during erase: the same as above can happen. Data may appear as erased but the retention is lowered. A power failure during a page-erase can inhibit readability of all data in its physical sector. Therefore an algorithm is advantageous that performs erases only in sectors that don't contain anymore current data.

The algorithm can be improved to be more robust against such cases, e.g. program always two pages, mark the end of an erase process by programming a page. But generally aborting flash processes is a forbidden "operating condition".

The main deficiency of the described algorithm is that the software designer is required to plan the use of the flash memory thoroughly. The user has to choose the correct value of N. Then all data has to be allocated to pages. Data sharing one page should have a similar or better identical update pattern (otherwise unchanged data is unnecessarily written). If one set of data does not fill a complete sector the available pages must be possibly left unused because they might incur too many drain disturbs.

There are other algorithms that try to alleviate these efforts by monitoring the flash usage and adapt automatically the assignment of data to flash cells.

### **3.10.11 Interrupt Generation**

Long lasting processes (these are mainly: program page, erase page, erase sector and margin changes, but also enter page mode) set the `IMB_FSR_BUSY.BUSY` flag of one flash module when accepting the request and reset this flag after finishing the process. Software is required to poll the busy flag in order to determine the end of the operation. In order to release the software from this burden an interrupt can be generated. If the interrupt is enabled by `IMB_INTCTRL.IEN` then all transitions from 1 to 0 of one of the `IMB_FSR_BUSY.BUSY` flags send an interrupt request to the SCU. In the SCU (see **“SCU Interrupt Generation” on Page 8-204** in the SCU chapter) the interrupt request (noted as “PFI” Program Flash Interrupt) is multiplexed with other interrupt sources and is forwarded to one of four interrupt nodes. The selection of the interrupt node is done with the register field `INTNP1.PF`. The SCU contains its own set of interrupt status flags (`INTSTAT`), interrupt disable control (`INTDIS`) and registers for setting this interrupt (`INTSET`) and clearing it (`INTCLR`).

The **“Enter Blue Mode”** command sets `BUSY` only for a few clock cycles. It is usually not advisable to enable the interrupt for this command.

The register `IMB_INTCTR` contains fields for the interrupt status “ISR”, an enable for the interrupt request “IEN” and fields for clearing the status flag “ICLR” or setting it “ISET”. It should be noted that the interrupt request is only sent when `ISR` becomes 1 and `IEN` was already 1. No interrupt is sent when `IEN` becomes 1 when `ISR` was already 1 or both are set to 1 at the same time.

### **3.10.12 Recommendations for Optimized Flash Usage**

This section describes best practices for using the flash in certain application scenarios, e.g. how to use effectively ECC and margin reads. For a description of the hardware features consult **“Data Integrity” on Page 3-41**.

#### **3.10.12.1 Programming Code and Constant Data**

Code and constant data are programmed only few times during life-time of a device, e.g. end-of-line in ECU production or when service updates are performed. As the readability of this data is decisive for the product quality customers might want to implement the elaborate “best practice” advice.

#### **Basic Advice**

Always ensure correct operating conditions and prevent power failures during flash operation.

As basic protection against handling errors all data should be verified after programming. Single-bit ECC errors should be ignored. The appearance of small numbers of single-bit errors is a consequence of known physical effects.

**Best Practice**

This approach offers best possible quality but risks that programming steps need to be repeated even unnecessarily (“false negatives”):

- Use “Erase Sector” to erase complete sectors.
- Program the sector with data. A common protection against software crashes is to fill the unused part of the sector with trap codes.
- Change the read level to hard margin 0.
- Verify the programmed data, note comparison errors and double-bit ECC errors and count single-bit ECC errors. Take care to evaluate the ECC error flags only once per 128-bit data block and clear them afterwards.
- Repeat this check with hard margin 1.
- After programming all sectors:
  - Erase and re-program all sectors with comparison or double-bit ECC errors.
  - If a flash module contained more than a certain number (e.g. 10) of single-bit ECC errors it is recommended to erase and re-program the affected sectors (i.e. those containing at least one single-bit error).
  - Attention: a high number of single-bit errors indicates usually a violation of operating conditions.

The threshold of allowed single-bit errors could be increased for in-service updates in order to reduce the risk of false negatives.

**3.10.12.2 EEPROM Emulation**

For EEPROM emulation the goal is usually not readability over device life-time but highest possible robustness (against violated operating conditions, power failures, even failing flash pages e.g. due to over-cycling). The risk of false negatives should be minimized.

A good robustness is achieved with the following approach:

- Verify data after programming with the normal read level. Single-bit ECC errors should be ignored.
- In case of comparison error or double-bit ECC error the data should be programmed again to the next flash range (e.g. next page or sector).
- The number of re-programming trials should be limited (e.g. to 3) to protect against violated operating conditions.

Obviously this jumping over failed pages can be only used optimally when the algorithm does not expect data on fixed addresses.

Failing pages can prevent “Erase Sector” from erasing any data in the affected sector. The “Erase Page” command however could still erase all other pages. These other pages stay readable and programmable.

### 3.11 On-Chip Program Memory Control

The internal memory block “IMB” contains all memories of the so called “on-chip program memory area” in the address range from C0’0000<sub>H</sub> to FF’FFFF<sub>H</sub>. Included are the program SRAM, the embedded flash memories and central control logic called “IMB Core”.

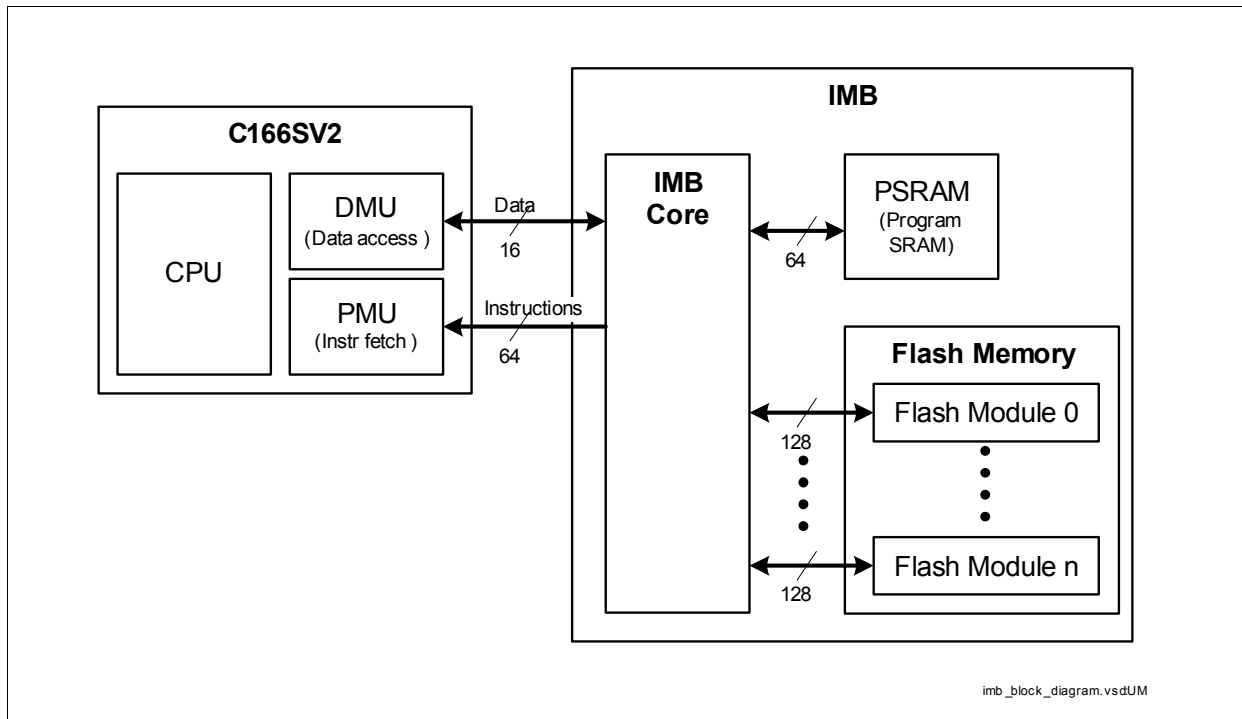
In the XE16xyM device the IMB contains the following memories:

- up to 832 KB flash memory in four independent modules.
- up to 32 KB program SRAM (see [Section 3.4.1](#)).

The IMB connects these memories to the CPU data bus and the instruction fetch bus. Each memory can contain instruction code, data or a mixture of both. The IMB manages accesses to the memories and supports flash programming and erase.

#### 3.11.1 Overview

The [Figure 3-9](#) shows how the IMB and its memories are integrated into the device architecture. Only the main data streams are included. The data buses are usually accompanied by address and control signals and check-sum data like parity or ECC.



**Figure 3-9 IMB Block Diagram**

The CPU has two independent busses. The instruction fetch bus is controlled by the program management unit “PMU” of the CPU. It fetches instructions in aligned groups of 64 bits. The instruction fetch unit of the CPU predicts the outcome of jumps and fetches instructions on the predicted branch in advance. In case of a misprediction this interface



**Memory Organization**

can abort outstanding requests and continues fetching on the correct branch. As the CPU can consume up to one 32-bit instruction per clock cycle the performance of this interface determines the CPU performance.

The data bus is controlled by the data management unit “DMU” of the CPU. It reads data in words of 16 bits. Write accesses address as well 16-bit words but additional byte enables allow changing single bytes.

Because of the CPU’s “von Neumann” architecture data and instructions (and “special function registers” to complete the list) share a common address range. When instructions are used as data (e.g. when copying code from an IO interface to the PSRAM) they are accessed via the data bus. The pipelined behavior of the CPU can cause that code fetches and data accesses are requested simultaneously. The IMB takes care that accesses can perform concurrently if they address different memories or flash modules.

Additional connections of the IMB to central system control units exist. These are not shown in the block diagram.

### 3.11.2 Register Interface

The “**IMB Registers**” on [Page 3-60](#) describes the special function registers of the IMB. In “**System Control Registers**” on [Page 3-75](#) the special function registers that influence the IMB but are not allocated to the IMB address range are described.

#### 3.11.2.1 IMB Registers

The section describes all IMB special function registers.

**Table 3-9 Registers Overview**

Register Short Name	Register Long Name	Offset Address	Page Number
IMB_IMBCTRL	IMB Control Low	FF FF00 <sub>H</sub>	<a href="#">Page 3-61</a>
IMB_IMBCTRH	IMB Control High	FF FF02 <sub>H</sub>	<a href="#">Page 3-62</a>
IMB_INTCTR	Interrupt Control	FF FF04 <sub>H</sub>	<a href="#">Page 3-64</a>
IMB_FSR_BUSY	Flash State Busy	FF FF06 <sub>H</sub>	<a href="#">Page 3-65</a>
IMB_FSR_OP	Flash State Operations	FF FF08 <sub>H</sub>	<a href="#">Page 3-66</a>
IMB_FSR_PROT	Flash State Protection	FF FF0A <sub>H</sub>	<a href="#">Page 3-68</a>
IMB_MAR0	Margin 0	FF FF0C <sub>H</sub>	<a href="#">Page 3-69</a>
IMB_PROCON0	Protection Configuration 0	FF FF10 <sub>H</sub>	<a href="#">Page 3-70</a>
IMB_PROCON1	Protection Configuration 1	FF FF12 <sub>H</sub>	<a href="#">Page 3-70</a>
IMB_PROCON2	Protection Configuration 2	FF FF14 <sub>H</sub>	<a href="#">Page 3-70</a>
IMB_PROCON3	Protection Configuration 3	FF FF16 <sub>H</sub>	<a href="#">Page 3-70</a>
IMB_ECC_TRAP	ECC Trap Control	FF FF20 <sub>H</sub>	<a href="#">Page 3-71</a>
IMB_ECC_STAT	ECC Status	FF FF22 <sub>H</sub>	<a href="#">Page 3-73</a>

#### IMB Control

Global IMB control.

Both IMB\_IMBCTRL and IMB\_IMBCTRH are reset by an Application Reset.

The write access to both registers is controlled by the register security mechanism as defined in the SCU chapter “**Register Control**” on [Page 8-269](#). Please note that the register write-protection is not activated automatically again after an access to IMB\_IMBCTR because this happens only for SCU internal registers.

**Memory Organization**

**IMB\_IMBCTRL**

**IMB Control Low**

**ISFR (FF FF00<sub>H</sub>)**

**Reset value: 55AC<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>DDF</b>		<b>DCF</b>		-	-	-	-	-	-	-	-	<b>DLC PF</b>	<b>WSFLASH</b>		
rw		rw		-	-	-	-	-	-	-	-	rw	rw		

Field	Bits	Typ	Description
WSFLASH	[2:0]	rw	<p><b>Wait States for Flash Access</b></p> <p>Number of wait cycles after which the IMB expects read data from the flash memory is:  <math>N_{WS} = WSFLASH</math>.</p> <p>This field determines as well the read timing of the PSRAM in the flash emulation address range. See <a href="#">“Flash Emulation” on Page 3-13</a>.</p> <p>The correct setting of this field depends on the system clock frequency. The data sheet of the device describes this relation.</p> <p><i>Note: WSFLASH must not be 0. This value is forbidden!</i></p>
DLCPF	3	rw	<p><b>Disable Linear Code Pre-Fetch</b></p> <p>0<sub>B</sub> “High Speed Mode”: When the next read request will be delivered from the buffer and so the flash memory would be idle, the IMB Core autonomously increments the last address and reads the next 128-bit block from the flash memory.</p> <p>1<sub>B</sub> “Low Power Mode”: This feature is disabled. Usually for code with power minimization requirements or for code with short linear code sections this feature should be disabled (DLCPF = 1). Enabling this feature is only advantageous for code section with longer linear sequences. With lower values of WSFLASH the performance gain of DLCPF=0 is reduced. In case of low WSFLASH settings DLCPF=1 might even lead to better performance than with linear code pre-fetch.</p>

**Memory Organization**

Field	Bits	Typ	Description
DCF	[13:12]	rw	<b>Disable Code Fetch from Flash Memory</b> 01 <sub>B</sub> Short notation DCF = 1. If RPA = 1 instructions cannot be fetched from flash memory. If RPA = 0 this field has no effect. 10 <sub>B</sub> Short notation DCF = 0. Instructions can be fetched independent of RPA. 00 <sub>B</sub> Illegal state. 11 <sub>B</sub> Illegal state. Both illegal states have the same effect as "01". This state can only be left by an Application Reset. When RPA = 0 software can change this field to any value. Otherwise code fetch can only be disabled but not enabled anymore until the next Application Reset.
DDF	[15:14]	rw	<b>Disable Data Read from Flash Memory</b> 01 <sub>B</sub> Short notation DDF = 1. If RPA = 1 data cannot be read from flash memory. If RPA = 0 this field has no effect. 10 <sub>B</sub> Short notation DDF = 0. Data can be read independent of RPA. 00 <sub>B</sub> Illegal state. 11 <sub>B</sub> Illegal state. Both illegal states have the same effect as "01". This state can only be left by an Application Reset. When RPA = 0 software can change this field to any value. Otherwise data reads can only be disabled but not enabled anymore until the next Application Reset.

IMB control high word. The WPA and RPA fields are described in **["Protection Handling Details" on Page 3-45.](#)**

**IMB\_IMBCTR\_H**

**IMB Control High**

**ISFR (FF FF02<sub>H</sub>)**

**Reset value: 0005<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>PSPROT</b>								-	-	-	-	<b>RPA</b>		<b>WPA</b>	
rw								-	-	-	-	rh		rh	

**Memory Organization**

Field	Bits	Typ	Description
WPA	[1:0]	rh	<b>Write Protection Activated</b> 01 <sub>B</sub> Short notation WPA = 1. The write protection of the flash memory is activated. 10 <sub>B</sub> Short notation WPA = 0. The write protection is not activated. 00 <sub>B</sub> Illegal state. 11 <sub>B</sub> Illegal state. Both illegal states have the same effect as "01". This state can only be left by an Application Reset. This field is only changed by the IMB Core. Software writes are ignored.
RPA	[3:2]	rh	<b>Read Protection Activated</b> 01 <sub>B</sub> Short notation RPA = 1. The read protection of the flash memory is activated. 10 <sub>B</sub> Short notation RPA = 0. The read protection is not activated. 00 <sub>B</sub> Illegal state. 11 <sub>B</sub> Illegal state. Both illegal states have the same effect as "01". This state can only be left by an Application Reset. This field is only changed by the IMB Core. Software writes are ignored.
PSPROT	[15:8]	rw	<b>PSRAM Write Protection</b> This 8-bit field determines the address up to which the PSRAM is write protected. The start address of the writable range is E0'0000 <sub>H</sub> + 1000 <sub>H</sub> *PSPROT. The end address is determined by the implemented memory. The equivalent range in the PSRAM area with flash access timing is protected as well. Here the writable range starts at E8'0000 <sub>H</sub> + 1000 <sub>H</sub> *PSPROT. So with PSPROT=00 <sub>H</sub> the complete PSRAM is writable.

### Interrupt Control

Interrupt control and status.

Reset by Application Reset.

## Memory Organization

### IMB\_INTCTR

#### Interrupt Control

**ISFR (FF FF04<sub>H</sub>)**

**Reset value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ISR	PSE R	–	–	–	PSE RCL R	ISSET	ICLR	–	–	–	–	DPR OTR P	DDD TRP	DIDT RP	IEN
rh	rh	–	–	–	w	w	w	–	–	–	–	rw	rw	rw	rw

Field	Bits	Typ	Description
IEN	0	rw	<b>Interrupt Enable</b> If set, the interrupt signal of the IMB gets activated when ISR is set.
DIDTRP	1	rw	<b>Disable Instruction Fetch Double Bit Error Trap</b> If set, a double bit ECC error does not cause the replacement of the fetched data by a trap instruction. See also <a href="#">IMB_ECC_TRAP.DITRPx</a> .
DDDTRP	2	rw	<b>Disable Data Read Double Bit Error Trap</b> If set, a double bit ECC error during data read does not send a “Flash Access Error” request to the SCU, i.e. no HW trap is generated and the read data is not replaced with default data. The error flags are still set in IMB_FSR_PROT and IMB_ECC_STAT. See also <a href="#">IMB_ECC_TRAP.DDTRPx</a> .
DPROTRP	3	rw	<b>Disable Protection Trap</b> If set, a read request from read protected flash memory does not generate a “Flash Access Error” request to the SCU, i.e. no HW trap is generated.
ICLR	8	w	<b>Interrupt Clear</b> When written with 1 the ISR is cleared. Reading this bit delivers always 0. Writing a 0 is ignored.
ISSET	9	w	<b>Interrupt Set</b> When written with 1 the ISR is set and if IEN is set the interrupt signal is activated. Reading this bit delivers always 0. Writing a 0 is ignored.
PSERCLR	10	w	<b>Clear PSRAM Error Flag</b> When written with 1 the PSER is cleared. Reading this bit delivers always 0. Writing a 0 is ignored.

## Memory Organization

Field	Bits	Typ	Description
PSER	14	rh	<b>PSRAM Error Flag</b> This flag is set when write requests to the write protected or not implemented PSRAM range are detected. This flag can be cleared by writing 1 to PSERCLR.
ISR	15	rh	<b>Interrupt Service Request</b> If set, it indicates that at least one IMB_FSR_BUSY.BUSY bit changed from 1 to 0. If IEN was set an interrupt request is sent to the interrupt controller. After servicing the interrupt the software handler clears this flag by writing a 1 to ICLR.

### Flash State

Flash state. Split into 3 registers IMB\_FSR\_BUSY, IMB\_FSR\_OP, and IMB\_FSR\_PROT. The protection relevant fields of IMB\_FSR\_PROT are described in [“Protection Handling Details” on Page 3-45](#).

The registers are reset by the Application Reset with the exception of “ERASE”, “PROG”, and “OPER”. These three fields are only reset by a Power-On Reset.

### IMB\_FSR\_BUSY

#### Flash State Busy

ISFR (FF FF06<sub>H</sub>)

Reset value: 0000<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	<b>PAG E3</b>	<b>PAG E2</b>	<b>PAG E1</b>	<b>PAG E0</b>	-	-	-	-	<b>BUS Y3</b>	<b>BUS Y2</b>	<b>BUS Y1</b>	<b>BUS Y0</b>
-	-	-	-	rh	rh	rh	rh	-	-	-	-	rh	rh	rh	rh

Field	Bits	Typ	Description
BUSY0	0	rh	<b>Busy Flash 0</b> Flash module 0 is busy with a task. The task is indicated by the bits MAR, POWER, ERASE or PROG of IMB_FSR_OP. BUSY0 is automatically cleared when the task has finished. The corresponding task indication is not cleared in order to allow an interrupt handler to determine the finished task.

**Memory Organization**

Field	Bits	Typ	Description
BUSY1	1	rh	<b>Busy Flash 1</b> Same as BUSY0 for flash module 1.
BUSY2	2	rh	<b>Busy Flash 2</b> Same as BUSY0 for flash module 2.
BUSY3	3	rh	<b>Busy Flash 3</b> Same as BUSY0 for flash module 3.
PAGE0	8	rh	<b>Page Mode Indication Flash 0</b> Set as long the flash module 0 is in page mode. Page mode is entered by the “ <b>Enter Page Mode</b> ” commands and finished by a “ <b>Program Page</b> ” command. The page mode can be also left by a “ <b>Reset to Read</b> ” command. Also an Application Reset clears this bit.
PAGE1	9	rh	<b>Page Mode Indication Flash 1</b> Same as PAGE0 for flash module 1.
PAGE2	10	rh	<b>Page Mode Indication Flash 2</b> Same as PAGE0 for flash module 2.
PAGE3	11	rh	<b>Page Mode Indication Flash 3</b> Same as PAGE0 for flash module 3.

**IMB\_FSR\_OP**

**Flash State Operations**

**ISFR (FF FF08<sub>H</sub>)**

**Reset value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	OPE R	SQE R	MAR	POW ER	ERA SE	PRO G
-	-	-	-	-	-	-	-	-	-	rh	rh	rh	rh	rh	rh

Field	Bits	Typ	Description
PROG	0	rh	<b>Program Task Indication</b> This bit is set when a program task is started. The affected flash module is indicated by a BUSY bit. The PROG bit is not automatically reset but must be cleared by a “ <b>Clear Status</b> ” command. This bit is not cleared by an Application Reset but only by a Power-On Reset.



**Memory Organization**

<b>Field</b>	<b>Bits</b>	<b>Typ</b>	<b>Description</b>
ERASE	1	rh	<b>Erase Task Indication</b> This bit is set when an erase task is started. The affected flash module is indicated by a BUSY bit. The ERASE bit is not automatically reset but must be cleared by a “ <b>Clear Status</b> ” command. This bit is not cleared by an Application Reset but only by a Power-On Reset.
POWER	2	rh	<b>Power Change Indication</b> This bit indicates that a flash module is in its startup phase or in a shutdown phase. The BUSY bits indicate which flash module is busy. This bit is not automatically reset but must be cleared by a “ <b>Clear Status</b> ” command.
MAR	3	rh	<b>Margin Change Indication</b> If a read margin modification is requested this bit is set together with the corresponding BUSY bit. The BUSY bit is cleared when the margin change is effective and the flash module can be read again. The MAR bit must be cleared by a “ <b>Clear Status</b> ” command.
SQER	4	rh	<b>Sequence Error</b> This bit is set by a errored command sequence or a command that is not accepted. It is cleared by “ <b>Clear Status</b> ” and “ <b>Reset to Read</b> ”.
OPER	5	rh	<b>Operation Error</b> The IMB Core maintains internal bits that are set when starting a program or erase process. They are cleared when this process finishes. These bits are not reset by an Application Reset but only by a Power-On Reset. If one of these bits is set after Application Reset the IMB Core sets OPER. So this signals that a running erase or program process was interrupted by an Application Reset. The OPER is cleared by “ <b>Reset to Read</b> ”, “ <b>Clear Status</b> ” or a Power-On Reset.

## Memory Organization

### IMB\_FSR\_PROT

#### Flash State Protection

**ISFR (FF FF0A<sub>H</sub>)**

**Reset value: x000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RPRO	-	-	DDB ER	DSB ER	IDBE R	ISBE R	-	-	-	PRO ER	WPR ODIS	RPR ODIS	PROI NER	PROI N	
rh	-	-	rh	rh	rh	rh	-	-	-	rh	rh	rh	rh	rh	

Field	Bits	Typ	Description
PROIN	0	rh	<b>Flash Protection Installed</b> Modified by the IMB Core. Cleared by Application Reset.
PROINER	1	rh	<b>Flash Protection Installation Error</b> Modified by the IMB Core. Cleared by Application Reset.
RPRODIS	2	rh	<b>Read Protection Disabled</b> The read protection was temporarily disabled with the “ <b>Disable Read Protection</b> ” command. Modified by the IMB Core. Cleared by Application Reset.
WPRODIS	3	rh	<b>Write Protection Disabled</b> The write protection was temporarily disabled with the “ <b>Disable Write Protection</b> ” command. Modified by the IMB Core. Cleared by Application Reset.
PROER	4	rh	<b>Protection Error</b> Set by a violation of the installed protection. Reset by the “ <b>Clear Status</b> ” and “ <b>Reset to Read</b> ” commands or an Application Reset.
ISBER	8	rh	<b>Instruction Fetch Single Bit Error</b> Set if during instruction fetch a single-bit ECC error was detected (and corrected). Reset by “ <b>Clear Status</b> ” or “ <b>Reset to Read</b> ” commands or an Application Reset.
IDBER	9	rh	<b>Instruction Fetch Double Bit Error</b> Set if during instruction fetch a double-bit ECC error was detected (and not corrected). Reset by “ <b>Clear Status</b> ” or “ <b>Reset to Read</b> ” commands or an Application Reset.

## Memory Organization

Field	Bits	Typ	Description
DSBER	10	rh	<b>Data Read Single Bit Error</b> Same as ISBER for data reads.
DDBER	11	rh	<b>Data Read Double Bit Error</b> Same as IDBER for data reads.
RPRO	[15:14]	rh	<b>Read Protection Configuration</b> This field is copied by the IMB Core from the corresponding field in the security page 0. After Application Reset read protection is activated. See <a href="#">Table 3-6</a> and ff for interpreting this and other protection bit fields. 00 <sub>B</sub> Invalid. 01 <sub>B</sub> Active. 10 <sub>B</sub> Inactive. 11 <sub>B</sub> Invalid.

### Margin Control

Read margin control. Each field corresponds to one flash module. A hard read 0 detects not completely erased cells. These are read as “1”. A hard read 1 detects not completely programmed cells. These are read as “0”. Read margin changes are caused by the command sequence “[Change Read Margin](#)”. The resulting read margin is reflected in this status register.

The command sequences “[Program Page](#)”, “[Erase Sector](#)”, “[Erase Page](#)” and “[Erase Security Page](#)” resets the read margin back to “normal”. The same happens in case of a flash wake-up.

Reset by Application Reset.

### IMB\_MAR0

#### Margin Control 0

ISFR (FF FF0C<sub>H</sub>)

Reset value: 0000<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	HREAD3	HREAD2	HREAD1	HREAD0								
-	-	-	-	rh	rh	rh	rh								

Field	Bits	Typ	Description
HREAD0	[2:0]	rh	<b>Hard Read 0</b> Active read margin of flash module 0. “000”: Normal read. “001”: Hard read 0. “010”: Alternate hard read 0 (usually harder than 001). “101”: Hard read 1. “110”: Alternate hard read 1 (usually harder than 101). other codes: Reserved.
HREAD1	[5:3]	rh	<b>Hard Read 1</b> Same as HREAD0 for flash module 1.
HREAD2	[8:6]	rh	<b>Hard Read 2</b> Same as HREAD0 for flash module 2.
HREAD3	[11:9]	rh	<b>Hard Read 3</b> Same as HREAD0 for flash module 3.

### Protection Configuration

Protection configuration register of each implemented flash module. The logical sector numbering is depicted in [Figure 3-6](#).

Each bit of the PROCONs is related to a logical sector. If it is cleared the write access to the corresponding logical sector (this means to the range of physical sectors) is locked under the conditions that are documented in [“Protection Handling Details” on Page 3-45](#). The PROCON registers are exclusively modified by the IMB Core which copies them from the security page 0.

For flash modules smaller than 256 KB the SsU bits corresponding to the not implemented flash range are reserved and shall be programmed to 0 in the security page.

Reset by Application Reset.

### IMB\_PROCONx (x=0-3)

**Protection Configuration.**

**ISFR (FF FF10<sub>H</sub>+2\*x)**

**Reset value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
–	–	–	S12 U	S11 U	S10 U	S9U	S8U	S7U	S6U	S5U	S4U	S3U	S2U	S1U	S0U
–	–	–	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Typ	Description
SsU (s=0-12)	s	rh	<b>Sector s Unlock</b> 0 <sub>B</sub> : Logical sector s of flash module x is write-protected. 1 <sub>B</sub> : Logical sector s of flash module x is not write-protected.  <i>Note: In previous device families and the TriCore™ based products these are “lock” bits and not “unlock” bits!</i>

### ECC Trap Control

ECC trap control register.

Reset by Application Reset.

The register IMB\_ECC\_TRAP allows to disable the double bit ECC error trap generation for selected flash modules in contrast to IMB\_INTCTR which allows to switch this only globally. This selective control enables to operate part of the flash memory as quasi ROM with enabled error traps. But while a flash module is programmed or erased its trap generation can be switched off without affecting the “ROM” modules. Without this facility the traps would have to be globally disabled and the flash driver had to work from SRAM and all interrupts would have to be blocked.

### IMB\_ECC\_TRAP

#### ECC Trap Control

**ISFR (FF FF20<sub>H</sub>)**

**Reset value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>DDT RP3</b>	<b>DDT RP2</b>	<b>DDT RP1</b>	<b>DDT RP0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>DITR P3</b>	<b>DITR P2</b>	<b>DITR P1</b>	<b>DITR P0</b>
r	r	r	r	rw	rw	rw	rw	r	r	r	r	rw	rw	rw	rw

**Memory Organization**

Field	Bits	Typ	Description
DITRP0	0	rw	<b>Disable Instruction Fetch Double Bit ECC Trap 0</b> $0_B$ Replacing instructions by a trap code for double bit ECC errors when fetching from flash module 0 is handled globally via IMB_INTCTR.DIDTRP. $1_B$ If set, a double bit ECC error does not cause the replacement of the fetched data by a trap instruction for fetches from flash module 0 independent of IMB_INTCTR.DIDTRP. Additionally IMB_FSR_PROT.ISBER/IDBER are not set for ECC errors from flash module 0.
DITRP1	1	rw	<b>Disable Instruction Fetch Double Bit ECC Trap 1</b> Same as DITRP0 for flash module 1.
DITRP2	2	rw	<b>Disable Instruction Fetch Double Bit ECC Trap 2</b> Same as DITRP0 for flash module 2.
DITRP3	3	rw	<b>Disable Instruction Fetch Double Bit ECC Trap 3</b> Same as DITRP0 for flash module 3.
DDTRP0	8	rw	<b>Disable Data Read Double Bit ECC Trap 0</b> $0_B$ Double bit ECC error trap for data reads from flash module 0 are handled globally via IMB_INTCTR.DDDTRP. $1_B$ Double bit ECC error for data reads from flash module 0 does not trigger the "Flash Access Error" trap independent of IMB_INTCTR.DDDTRP. Additionally IMB_FSR_PROT.DSBER/DDBER are not set for ECC errors from flash module 0 and the data from flash memory is delivered not default data. But the bits IMB_ECC_STAT.xBERx are still set for ECC errors.
DDTRP1	9	rw	<b>Disable Data Read Double Bit ECC Trap 1</b> Same as DDTRP0 for flash module 1.
DDTRP2	10	rw	<b>Disable Data Read Double Bit ECC Trap 2</b> Same as DDTRP0 for flash module 2.
DDTRP3	11	rw	<b>Disable Data Read Double Bit ECC Trap 3</b> Same as DDTRP0 for flash module 3.

## ECC Status

ECC status register.

Reset by Application Reset.

This register reports ECC data read single and double bit errors selectively per flash module. Each bit can be cleared independently. This enables to use part of the flash memory quasi as "ROM". In this part all errors trigger traps that are handled by a trap handler and trigger typically a reset of the application. However while flash modules are programmed or erased all ECC errors can be handled by a low-level driver without necessarily affecting the complete system.

## IMB\_ECC\_STAT

### ECC Status

**ISFR (FF FF22<sub>H</sub>)**

**Reset value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	<b>DBE R3</b>	<b>DBE R2</b>	<b>DBE R1</b>	<b>DBE R0</b>	0	0	0	0	<b>SBE R3</b>	<b>SBE R2</b>	<b>SBE R1</b>	<b>SBE R0</b>
r	r	r	r	rwh	rwh	rwh	rwh	r	r	r	r	rwh	rwh	rwh	rwh

Field	Bits	Typ	Description
SBER0	0	rwh	<b>Data Read Single Bit Error 0</b> Set when a single bit ECC errors occurs when reading data from flash module 0. Cleared by Application Reset or by writing 1 to this bit.
SBER1	1	rwh	<b>Data Read Single Bit Error 1</b> Same as SBER0 for flash module 1.
SBER2	2	rwh	<b>Data Read Single Bit Error 2</b> Same as SBER0 for flash module 2.
SBER3	3	rwh	<b>Data Read Single Bit Error 3</b> Same as SBER0 for flash module 3.
DBER0	8	rwh	<b>Data Read Double Bit Error 0</b> Set when a double bit ECC errors occurs when reading data from flash module 0. Cleared by Application Reset or by writing 1 to this bit.
DBER1	9	rwh	<b>Data Read Double Bit Error 1</b> Same as DBER0 for flash module 1.

**Memory Organization**

<b>Field</b>	<b>Bits</b>	<b>Typ</b>	<b>Description</b>
DBER2	10	rwh	<b>Data Read Double Bit Error 2</b> Same as DBER0 for flash module 2.
DBER3	11	rwh	<b>Data Read Double Bit Error 3</b> Same as DBER0 for flash module 3.



### 3.11.2.2 System Control Registers

These registers are used to wakeup and shutdown parts of the memory sub-system.

**Table 3-10 Registers Address Space**

Module	Base Address	End Address	Note
SCU	0000 <sub>H</sub>	0FFF <sub>H</sub>	SCU Module

**Table 3-11 Registers Overview**

Register Short Name	Register Long Name	Offset Address	Page Number
MEM_KSCCFG	Memory Kernel Control	F012 <sub>H</sub>	<a href="#">Page 3-75</a>
FL_KSCCFG	Flash Kernel Control	FE22 <sub>H</sub>	<a href="#">Page 3-76</a>

#### Memory Kernel Configuration

This register controls the shutdown request of the processor sub-system units DMU, PMU, IMB and EBC. The layout of this register is identical to the other KSCCFGs but only the field COMCFG may be used. Two values of this field might be used: 00<sub>B</sub> means that the “Clock-off Mode” does not trigger a shutdown of the processor sub-system. This may be used only if the system clock of DMP\_1 is not disabled in the “Clock-off Mode”.

The second useful value is 10<sub>B</sub>. This value must be used in all cases when the “Clock-off Mode” is accompanied by disabling the system clock of the DMP\_1.

This register is reset by an Application Reset. **Attention:** the reset value of COMCFG is 00<sub>B</sub>.

The access to this register is controlled by the register security mechanism (“Sec” type).

#### MEM\_KSCCFG

**Memory Kernel State Con**      **ESFR (F012<sub>H</sub>/06<sub>H</sub>)**      **Reset Value: 0001<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BP COM	–	COMCFG	–	–	–	–	–	–	–	–	–	–	–	–	1
w	–	rw	–	–	–	–	–	–	–	–	–	–	–	–	rw

Field	Bits	Type	Description
1	0	rw	Has to be written to 1.

## Memory Organization

Field	Bits	Type	Description
<b>COMCFG</b>	[13:12]	rw	<b>Clock Off Mode Configuration</b> This bit field defines if the shutdown request is activated in clock-off mode. If COMCFG[13] is 1 the shutdown request is activated in clock-off mode (i.e. CR = 10). COMCFG[12] has no functionality.
<b>BPCOM</b>	15	w	<b>Bit Protection for COMCFG</b> This bit enables the write access to the bit field COMCFG. It always reads 0. It is only active during the write access cycle. 0 The bit field COMCFG is not changed. 1 The bit field COMCFG is updated with the written value.

### Flash Kernel Configuration

This register controls the power-down request of the flash module. When configuring this register care must be taken not to enable a powered-down flash module when the operating voltage is not sufficient. In this case all CFG fields should contain 10<sub>B</sub>.

This register is reset by an Application Reset.

The access to this register is controlled by the register security mechanism ("Sec" type).

### FL\_KSCCFG

Flash Kernel State Con.

SFR (FE22<sub>H</sub>/11<sub>H</sub>)

Reset Value: 0001<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>BP COM</b>	–	<b>COMCFG</b>	<b>BP SUM</b>	–	<b>SUMCFG</b>	<b>BP NOM</b>	–	<b>NOMCFG</b>	–	<b>BP MOD EN</b>	<b>MOD EN</b>				
w	–	rw	w	–	rw	w	–	rw	–	–	w	rw			

Field	Bits	Type	Description
<b>MODEN</b>	0	rw	<b>Module Enable</b> This bit can directly set the power-down request. 0 The power-down request is activated. 1 This field has no effect.

**Memory Organization**

Field	Bits	Type	Description
<b>BPMODEN</b>	1	w	<b>Bit Protection for MODEN</b> This bit enables the write access to the bit MODEN. It always reads 0. It is only active during the write access cycle. 0 The bit MODEN is not changed. 1 The bit MODEN is updated with the written value.
<b>NOMCFG</b>	[5:4]	rw	<b>Normal Operation Mode Configuration</b> This bit field defines if the power-down request is activated in normal operation mode. If NOMCFG[5] is 1 the power-down request is activated in normal mode (i.e. CR = 00 or 11). NOMCFG[4] has no functionality.
<b>BPNO</b>	7	w	<b>Bit Protection for NOMCFG</b> This bit enables the write access to the bit field NOMCFG. It always reads 0. It is only active during the write access cycle. 0 The bit field NOMCFG is not changed. 1 The bit field NOMCFG is updated with the written value.
<b>SUMCFG</b>	[9:8]	rw	<b>Suspend Mode Configuration</b> This bit field defines if the power-down request is activated in suspend mode (which makes only sense if it is activated in normal mode as well). If SUMCFG[9] is 1 the power-down request is activated in shutdown mode (i.e. CR = 01). SUMCFG[8] has no functionality.
<b>BPSUM</b>	11	w	<b>Bit Protection for SUMCFG</b> This bit enables the write access to the bit field SUMCFG. It always reads 0. It is only active during the write access cycle. 0 The bit field SUMCFG is not changed. 1 The bit field SUMCFG is updated with the written value.

**Memory Organization**

Field	Bits	Type	Description
<b>COMCFG</b>	[13:12]	rw	<b>Clock Off Mode Configuration</b> This bit field defines if the power-down request is activated in clock-off mode. If COMCFG[13] is 1 the power-down request is activated in clock-off mode (i.e. CR = 10). COMCFG[12] has no functionality.
<b>BPCOM</b>	15	w	<b>Bit Protection for COMCFG</b> This bit enables the write access to the bit field COMCFG. It always reads 0. It is only active during the write access cycle. 0     The bit field COMCFG is not changed. 1     The bit field COMCFG is updated with the written value.

### **3.11.3 Startup, Shutdown**

The startup and shutdown of memories and the processor sub-system is described in the Programmer's Guide. Also the use of the Kernel Control registers is described there.

### 3.11.4 Error Reporting Summary

The [Table 3-12](#) summarizes the types of detected errors and the possible reactions.

**Table 3-12 IMB Error Reporting**

Error	Reaction
Data read from PSRAM with parity error.	If PECON.PEENPS: HW trap (see <a href="#">Section 8.14.2</a> ).
Instruction fetch from PSRAM with parity error.	If PECON.PEENPS: HW trap (see <a href="#">Section 8.14.2</a> ).
Data read from flash memory with single bit error.	Silently corrected. Bit IMB_FSR_PROT.DSBER set.
Data read from flash memory with double bit error.	Bit IMB_FSR_PROT.DDBER set. If IMB_INTCTR.DDDTRP = 0: Flash access trap <sup>1)</sup> and default data is delivered.
Instruction fetch from flash memory with single bit error.	Silently corrected. Bit IMB_FSR_PROT.ISBER set.
Instruction fetch from flash memory with double bit error.	Bit IMB_FSR_PROT.IDBER set. If IMB_INTCTR.DIDTRP = 0: “TRAP 15 <sub>D</sub> ” delivered instead of corrupted data.
Data read from protected flash memory.	IMB_FSR_PROT.PROER set. If IMB_INTCTR.DPROTRP = 0: Flash access trap <sup>1)</sup> and default data is delivered.
Instruction fetch from protected flash memory.	“TRAP 15 <sub>D</sub> ” delivered.
Program/erase request of write protected flash range.	Only bit PROER in IMB_FSR_PROT set.
Data read or instruction fetch from busy flash memory.	Read access stalled until end of busy state.
Instruction fetch from ISFR addresses.	Default data (“TRAP 15 <sub>D</sub> ”) delivered.
Data read from not implemented ISFRs.	Default data delivered.
Data writes to not implemented ISFRs.	Silently ignored.
Data read from not implemented address range.	Unpredictable. Mirrored data from other memories might be returned or default values.

**Table 3-12 IMB Error Reporting (cont'd)**

<b>Error</b>	<b>Reaction</b>
Instruction fetch from not implemented address range.	Unpredictable. Mirrored data from other memories might be returned or default values.
Data written to not implemented PSRAM or write protected PSRAM address range (both determined by IMB_IMBCTRH.PSPROT).	Bit IMB_INTCTR.PSER set. Flash access trap <sup>1)</sup> and no data is changed in the PSRAM.
Program or erase command targeting not implemented flash memory.	Unpredictable. Access is ignored <sup>2)</sup> or mirrored into implemented flash memory <sup>3)</sup> .
Data read from powered-down flash modules.	Considered as access to not-implemented memory range. Default data or data from implemented flash modules will be returned.
Instruction fetch from powered-down flash modules.	Considered as access to not-implemented memory range. Default data ("TRAP 15 <sub>D</sub> ") will be returned or data from implemented flash modules.
Program or erase command targeting powered-down flash modules.	Silently ignored <sup>2)</sup> .
Shutdown or power-down request received while the command sequence interpreter is waiting for the last words of a command sequence.	The command interpreter is reset and a <b>"Reset to Read"</b> command sequence is executed.

1) More information about the Flash Access Trap can be found in chapter "SCU".

2) Attention: when an access (i.e. MOV) is ignored, the command sequence interpreter will still wait for this outstanding MOV. So the next command sequence might cause a SQER because it delivers an unexpected MOV.

3) The flash protection can not be by-passed by accessing the reserved memory ranges.

### **3.12 Data Retention Memories**

This section describes the usage of the special purpose data memories Standby RAM (SBRAM) and Marker Memory (MKMEM). Depending on the device not all of them are available. The XE16xyM contains:

- SBRAM.
- MKMEM.

Both are supplied by the wake-up power domain (DMP\_M) and retain their data while the system power domain (DMP\_1) is switched off.

#### **3.12.1 Standby RAM Accesses**

The SBRAM is not mapped into the address range of the processor. All accesses are done via the 4 SFRs SBRAM\_WADD, SBRAM\_RADD, SBRAM\_DATA0 and SBRAM\_DATA1. The following access options exist:

- Write without automatic increment of the write address pointer:  
The software has to write the target address first to WADD and then the data to DATA0. The data written to DATA0 is transferred to the indicated address in the SBRAM if (at least) the lower byte of DATA0 is written. If DATA0 is written again the same address in SBRAM is used for data storage. Bit WADD.MOD is cleared by a write access to DATA0.
- Write with automatic increment of the write address pointer:  
The software has to write the first target address to WADD and thereafter the data block can be written word by word to DATA1. The data written to DATA1 is transferred to the indicated address in the SBRAM if (at least) the lower byte of SRDR1 is written. In parallel to the data storage in the SBRAM, the write address pointer WADD.WPTR is automatically incremented by 1 (one word) for the next data to be stored. The address pointer automatically does a wrap-around after reaching its maximum value and in this case, bit WADD.WA is set. Bit WADD.MOD is set by a write access to DATA1.
- Read without automatic increment of the read address pointer:  
The software has to write the target address first to RADD and then can read the data from DATA0. If DATA0 is read again the same address in SBRAM is read out. Bit RADD.MOD is cleared by a read access to DATA0.
- Read with automatic increment of the read address pointer:  
The SW has to write the first target address to RADD and can then read the data block word by word from DATA1. In parallel to the read action from SBRAM, the read address pointer RADD.RPTR is automatically incremented by 1 (one word) for the next data to be read. The address pointer automatically does a wrap-around after reaching its maximum value and in this case, bit RADD.WA is set. Bit RADD.MOD is set by a read access to DATA1.

The automatic increment accesses allow performing back-to-back data writes and reads.



## **Memory Organization**

*Note: Because read accesses to SBRAM\_DATA0 and SBRAM\_DATA1 return the value that has been pre-read upon the most recent update of register SBRAM\_RADD, any data written to location @SBRAM\_RADD can only be read back after SBRAM\_RADD has been updated with the very same address (either explicitly by writing to it or implicitly via the auto-increment function). Generally when switching from write to read accesses SBRAM\_RADD should be written again before reading SBRAM\_DATAx.*

*Note: Because of this pre-reading feature and the auto-increment behavior it is important to initialize always the address following the last data in order to prevent parity/ECC errors due to this pre-reading.*

### 3.12.2 Standby RAM Registers

This section describes the SBRAM register interface in detail.

**Table 3-13 Registers Overview**

Register Short Name	Register Long Name	Offset Address	Page Number
SBRAM_RADD	SBRAM Read Address	FEDC <sub>H</sub>	<a href="#">Page 3-84</a>
SBRAM_WADD	SBRAM Write Address	FEDE <sub>H</sub>	<a href="#">Page 3-85</a>
SBRAM_DATA0	SBRAM Data Register 0	FEE0 <sub>H</sub>	<a href="#">Page 3-87</a>
SBRAM_DATA1	SBRAM Data Register 1	FEE2 <sub>H</sub>	<a href="#">Page 3-88</a>

#### 3.12.2.1 SBRAM Read Address Register

This register defines the word location to be read.

Reset by Power-On Reset.

##### SBRAM\_RADD

**SBRAM Read Address Register SFR (FEDC<sub>H</sub>/6E<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>MOD</b>	<b>WA</b>	<b>0</b>	<b>RPTR</b>												<b>0</b>
rwh	rwh	r	rwh												r

Field	Bits	Type	Description
<b>RPTR</b>	[12:1]	rwh	<b>Read Pointer</b> Selects the word address to be read from the SBRAM. It is automatically incremented by 1 (i.e. to the next word) when register DATA1 is read.
<b>WA</b>	14	rwh	<b>Wrap Around</b> This bit indicates if a wrap-around of the read pointer RPTR occurred due to the automatic address increment. 0 An address wrap-around has not occurred. 1 An address wrap-around has been detected. It has to be cleared by software.

**Memory Organization**

Field	Bits	Type	Description
<b>MOD</b>	15	rwh	<b>Modification</b> This bit indicates whether the last read access to SBRAM data lead to an automatic increment of RPTR. 0 The last data read access was done to DATA0 and RPTR was not modified automatically. 1 The last data read access was done to DATA1 and RPTR was automatically incremented by 1.
<b>0</b>	0, 13	r	<b>Reserved</b> Read as 0; should be written with 0.

### 3.12.2.2 SBRAM Write Address Register

This register defines the word location to be written.

Reset by Power-On Reset.

#### **SBRAM\_WADD**

**SBRAM Write Address Register SFR (FEDE<sub>H</sub>/6F<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>MOD</b>	<b>WA</b>	<b>0</b>													<b>0</b>
rwh	rwh	r							rwh						r

Field	Bits	Type	Description
<b>WPTR</b>	[12:1]	rwh	<b>Write Pointer</b> Selects the write word address within the SBRAM. It is automatically incremented by 1 when register DATA1 is written.
<b>WA</b>	14	rwh	<b>Wrap-Around</b> This bit indicates if a wrap-around of the write pointer WPTR occurred due to the automatic address increment. 0 An address wrap-around has not occurred. 1 An address wrap-around has been detected. It has to be cleared by software.

**Memory Organization**

Field	Bits	Type	Description
<b>MOD</b>	15	rwh	<b>Modification</b> This bit indicates whether the last write access to SBRAM data lead to an automatic increment of WPTR. 0     The last data write access was done to DATA0 and WPTR was not modified automatically. 1     The last data write access was done to DATA1 and WPTR was automatically incremented by 1.
<b>0</b>	0, 13	r	<b>Reserved</b> Read as 0; should be written with 0.

### 3.12.2.3 SBRAM Data Register 0

This register delivers the read data and is the target for the write data without modification of the respective address pointer.

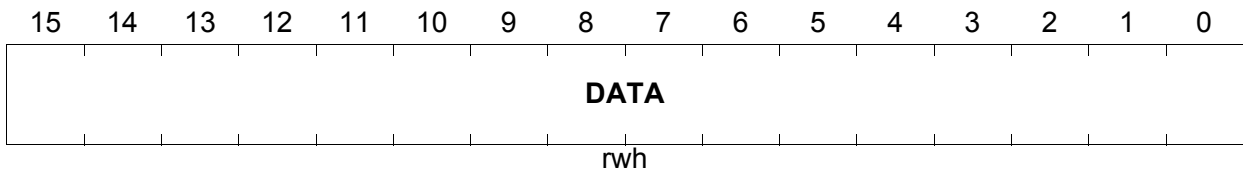
Reset by Power-On Reset.

#### **SBRAM\_DATA0**

**SBRAM Data Register 0**

**SFR (FEE0<sub>H</sub>/70<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
<b>DATA</b>	[15:0]	rwh	<b>SBRAM Data</b> This bit field contains the data read during the latest SBRAM read access and is the target for the data to be written to SBRAM. A read access always delivers the data stored in the SBRAM at the address indicated by the read pointer RADD.RPTR. A write access of (at least) the low byte leads to the storage of the written data at the address indicated by the write pointer WADD.WPTR.

### 3.12.2.4 SBRAM Data Register 1

This register delivers the read data and is the target for the write data with modification of the respective pointer.

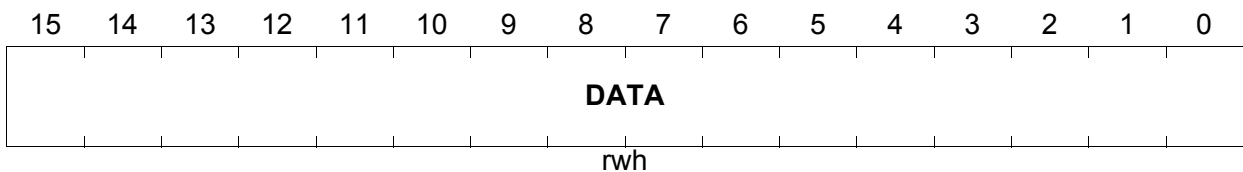
Reset by Power-On Reset.

#### **SBRAM\_DATA1**

##### **SBRAM Data Register 1**

**SFR (FEE2<sub>H</sub>/71<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
<b>DATA</b>	[15:0]	rwh	<b>SBRAM Data</b> This bit field contains the data read during the latest SBRAM read access and is the target for the data to be written to SBRAM. A read access always delivers the data stored in the SBRAM at the address indicated by the read pointer RADD.RPTR. A write access of (at least) the low byte leads to the storage of the written data at the address indicated by the write pointer WADD.WPTR.

## 4 Memory Checker Module (MCHK)

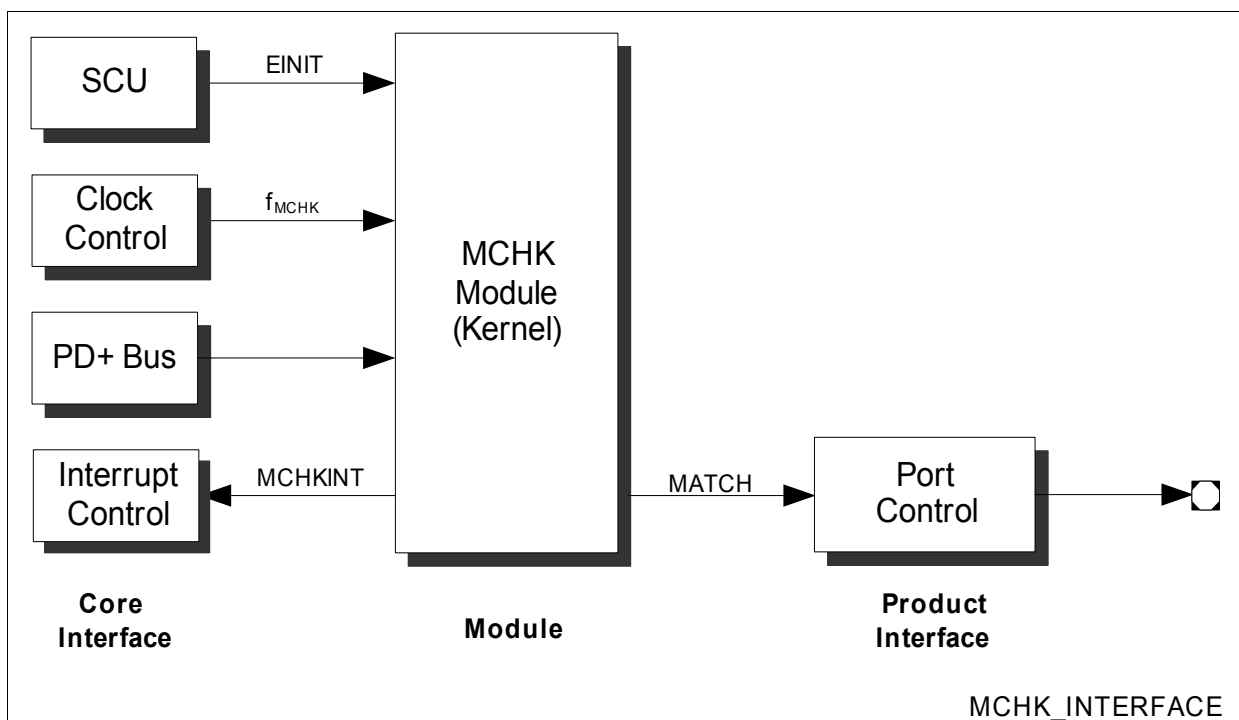
The memory checker module (MCHK) of the XE16xyM supports checking the data consistency of memories, registers (e.g. configuration registers), or communication channels. It calculates a checksum on a block of data, often called cyclic redundancy code (CRC). It is implemented as a parallel signature generation based on a multi input linear feedback shift register (MISR). Being based on a linear feedback shift register (LFSR), it also can generate pseudo-random numbers and cyclic codes.

This chapter is structured as follows:

- An operational overview of the Memory Checker Module (see [Section 4.1](#))
- Functional description of the Memory Checker Module (see [Section 4.2](#))
- Description of the Memory Checker Module registers (see [Section 4.3](#))
- Description of the general registers (see [Section 4.4](#))
- Interfaces of the Memory Checker Module (see [Section 4.5](#))

### 4.1 Operational Overview

From the programmer's point of view, the MCHK is a set of registers associated with this peripheral. To communicate respective error or operation events a port pin may be used for the signal "MATCH" to generate an external event and an interrupt line may be used for the signal "MISMATCH" to generate an internal event. The MCHK is reset together with the CPU so it can be used as a CPU coprocessor. This ensures a deterministic state of the MCHK after the CPU exits the reset state.



**Figure 4-1 Interface Diagram**

*Note: The MATCH output is connected to an external port in packaged devices with 144 or more pins only.*

## **4.2 Functional Description**

Conventional digital processing systems generally are configured around volatile and non-volatile memory elements. These memories provide (store) the data and instructions to the CPU doing the main computing of an embedded system. This includes the administration of the system by coordinating the operation of various system units to perform system tasks.

Faults within these memories are in general critical for the safety and reliability of an embedded system. Therefore these memories have mechanisms to check for data consistency, e.g. parity or ECC (error correction code). These mechanisms can detect faults up to a certain amount (e.g. double bit faults) per word (bit line faults). Concatenated codes (block codes, word codes) can also detect multibit faults per word (word line faults), which increases the fault coverage.

The MCHK is a parallel signature compression circuitry that calculates a concatenated CRC block code to increase test coverage by code concatenation. This enables error detection within a block of data stored in memory, registers, or communicated e.g. via serial communication lines. The MCHK reduces the probability of error masking due to repeated error patterns by compressing parallel test inputs from a block of data to be tested. Furthermore, it can generate pseudo random numbers.

The MCHK uses a multiple input linear feedback shift register to generate a checksum (signature) of a block of data. A multiple input linear feedback shift register (MISR) is a shift register whose internal feedback input bit is driven by the exclusive-or (XOR) of some bits of the overall shift register value (LFSR: linear feedback shift register).

This generator includes an arithmetic circuitry to calculate the block code. This circuitry is implemented as an independent piece of hardware and, therefore, does not rely on the memories to be tested. Only for the configuration it requires initialization data out of the memories. To avoid the need of a multi master system (CPU, DMA, etc.), the CPU (e.g. PEC, subroutine) is used to handle the data read and write transactions. These transactions rely on the memories to be tested because they may contain the respective CPU instruction code. Therefore the MCHK implements additional measures to enable detection of erroneous data move operations by the CPU.

The following error scenarios are detected:

- The CPU configures the MCHK erroneously
- The CPU does not provide the data from the respective address range
- The CPU does not provide the correct amount of data
- The CPU is not able to check correctly the match of the online generated CRC block code and the expected offline (during development) generated CRC block code



### **Memory Checker Module (MCHK)**

The principle of this circuitry is to generate an external coded life signal of the CRC block code check. Furthermore the configuration of the circuitry cannot be changed without an external notification. The life signal is not a static signal, but changes polarity in a predefined manner to avoid static faults, e.g. open and short circuit in the output stages.

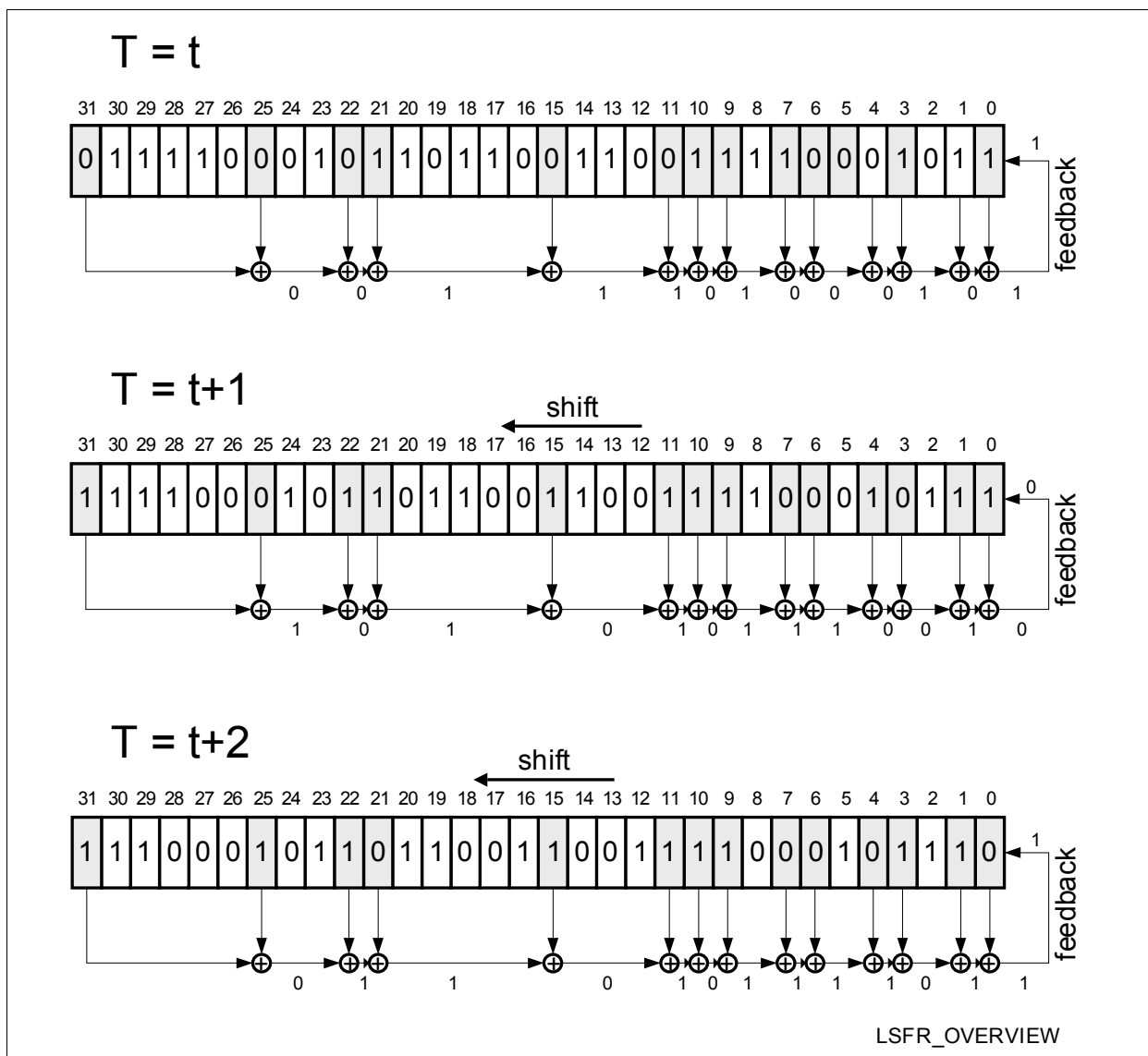
The circuitry consists of the following components:

- An arithmetic circuitry calculating the CRC block code out of the data transferred into an input register of this circuitry.
- A compare unit to check if the value of the calculated CRC block code is correct. The MCHK compares the content of the CRC block code result registers to a fixed value (FADE'EDDA<sub>H</sub>). Before calculating a CRC value, the result register is initialized with a specific value (magic word, seed), which results in a specific value after the CRC calculation. This so called magic word must be selected in a way that the block code ends up with the fixed value (FADE'EDDA<sub>H</sub>). This works fine for linear code, e.g. the CRC block code.
- A method granting the CRC block code calculation over a given amount of data. Therefore functional redundancy is used to grant this. A local count register within the MCHK initiates the compare of the calculated CRC block code after a given amount of input data. Secondly the CPU reloads the magic word (seed) of the CRC block code when initiating a new CRC block code generation (loop variable within data move subroutine or count register within PEC).
- An internal service request generation to enable software recovery in case of a fault. This could be a software routine running out of a different flash block than the one that produced the error, e.g. to support a limb home function. There is a residual risk: The CPU could write dummy data into the memory checker within this error interrupt routine and then rewrite the COUNT register.
- All configuration registers are protected by a time redundant mechanism. So modifications of configuration registers are only possible following a specific sequence of write operations (EINIT protected). Additionally, the COUNT register is protected by a content dependent access scheme.
- An external MATCH signal is generated on every successful CRC block code generation and may be used to trigger an external window watchdog. This window watchdog may generate a reset in case too many or too few MATCH signal toggles fall into a specific time window. To grant a correctly working block code unit, the application must also perform from time to time a block code generation outside the watchdog time window by having an incorrect compare (no MATCH signal toggle). This will check the correct function of the compare circuitry, because in case of an erroneous compare circuitry a MATCH signal toggle is generated outside the watchdog time window.

## 4.2.1 Principle of the LFSR

The list of the bit positions that affect the internal feedback bit is called the tap sequence. **Figure 4-2** shows the principle of an LFSR. It assumes a tap sequence of [32, 26, 23, 22, 16, 12, 11, 10, 8, 7, 5, 4, 2, 1]. On activation, all bits in the LFSR are shifted left one bit position (in direction of the most significant bit). All bits on the tap position are exclusive-ORed and the result is fed into bit 0 as a feedback. On the next activation the same procedure is repeated.

The LFSR outputs that influence the internal feedback input are called taps (marked gray in **Figure 4-2**).



**Figure 4-2 Principle of an LFSR**

### **Memory Checker Module (MCHK)**

The tap sequence of an LFSR can be represented as a polynomial mod 2. This means that the coefficients of the polynomial must be 0 in case a respective feedback is not implemented or 1 in case a feedback tap is implemented. This is called the feedback polynomial or characteristic polynomial. For example, if the taps are at the 32nd, 26th, 23rd, 22nd, 16th, 12th, 11th, 10th, 8th, 7th, 5th, 4th, and the 2nd bits (as below), the resulting LFSR polynomial is

$$G^{32} = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1 \quad (4.1)$$

The '+1' in the polynomial does not correspond to a tap. The powers of the terms represent the tapped bits, counting from the least significant bit. The polynomial may be represented by a binary number (binary representation). Every power of the terms represent a 1 in the binary format counting from the most significant bit. So for the polynomial listed above, the number would be:

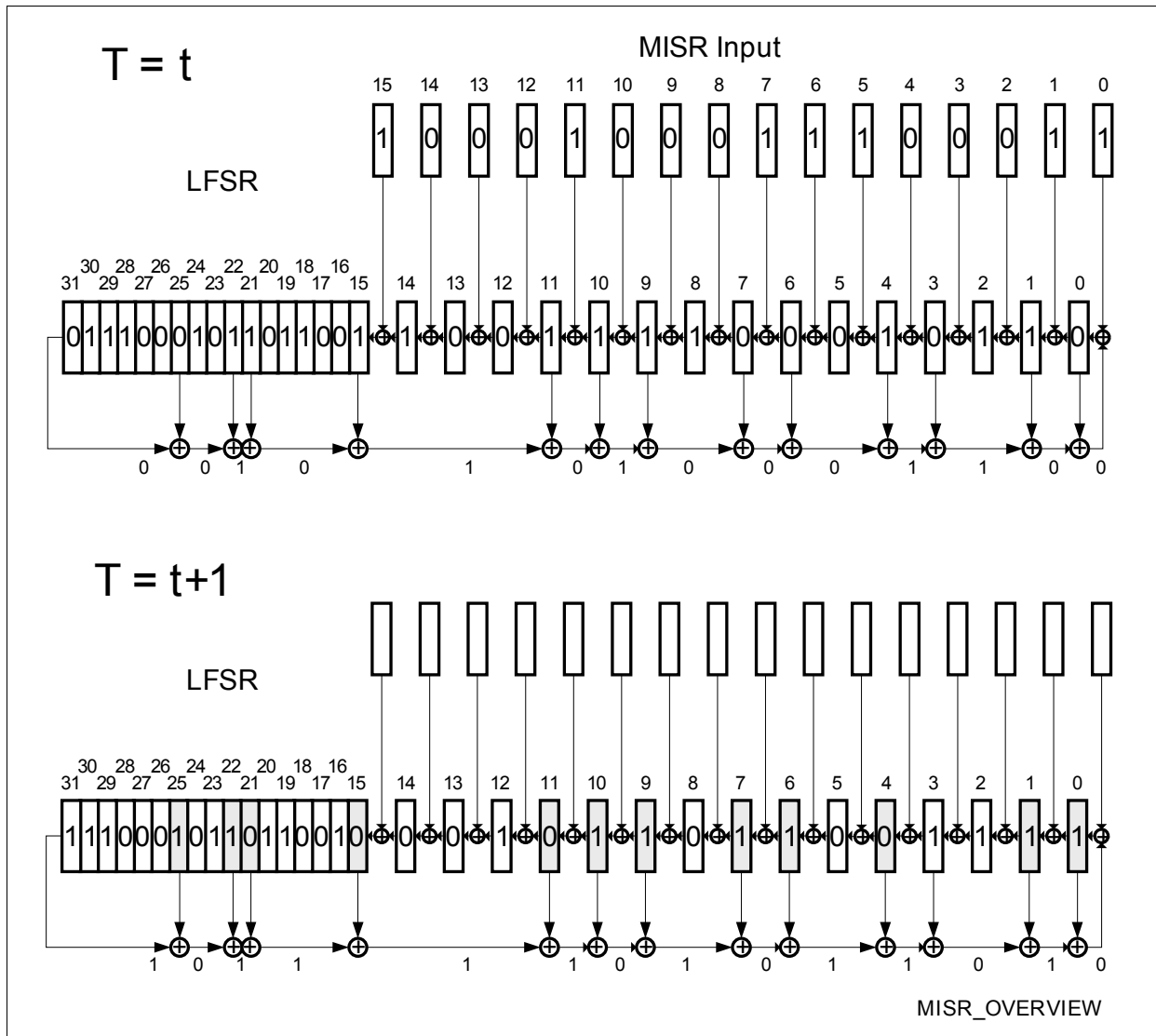
$$G^{32} = 1000\ 0010\ 0110\ 0000\ 1000\ 1110\ 1101\ 1011_B = 8260'8EDB_H$$

The polynomial used by the MCHK is defined in the tap polynomial registers **TPRH** and **TPRL**.

## 4.2.2 Principle of the MISR

In parallel to the internal feedback input bit of the LFSR, 16 external bits may be loaded into the LFSR (multiple input).

These 16 input bits are exclusive-ORed with the bits shifted or fed back in the LFSR.



**Figure 4-3 Principle of a MISR**

The initial value of the LFSR/MISR is called the seed and may be defined by an initial write to the result registers **RRL** and **RRH**. Because the operation of the MISR is deterministic, the sequence of values produced by the MISR is completely determined by its current (or previous) states and inputs.

### 4.2.3 Commonly used Polynomials

Polynomials for cyclic codes as used in globally standardized systems have not been fully standardized themselves. Most cyclic codes in current use have some weakness with respect to strength or construction. Standardization of cyclic codes would allow for better designed cyclic codes to come into common use. The following table provides a list of common polynomials used for sequential CRC signature generation.

**Table 4-1 Some Commonly used Polynomials**

Name	Polynomial	Maximum Data Width	Normal (Reverse) of Reciprocal
CRC-8-ATM	$x^8+x^2+x+1$	8-bit	0000'0083 <sub>H</sub> (0000'00C1 <sub>H</sub> )
CRC-8-CCITT	$x^8+x^7+x^3+x^2+1$	8-bit	0000'00C6 <sub>H</sub>
CRC-8-Dallas	$x^8+x^5+x^4+1$	8-bit	0000'0098 <sub>H</sub>
CRC-8	$x^8+x^7+x^6+x^4+x^2+1$	8-bit	0000'00EA <sub>H</sub>
CRC-8 SAE J1850	$x^8+x^4+x^3+x^2+1$	8-bit	0000'008E <sub>H</sub>
CRC-1 (parity)	$x^8+x^7+x^6+x^5+x^4+x^3+x^2+x+1$	8-bit	0000'00FF <sub>H</sub>
CRC-10	$x^{10}+x^9+x^5+x^4+x+1$	10-bit	0000'0319 <sub>H</sub> (0000'0263 <sub>H</sub> )
CRC-12	$x^{12}+x^{11}+x^3+x^2+x+1$	12-bit	0000'0C07 <sub>H</sub> (0000'0E03 <sub>H</sub> )
CRC-15-CAN	$x^{15}+x^{14}+x^{10}+x^8+x^7+x^4+x^3+1$	15-bit (13-bit)	0000'62CC <sub>H</sub> (0000'19A3 <sub>H</sub> )
CRC-1 (parity)	$x^{16}+x^{15}+x^{14}+x^{13}+x^{12}+x^{11}+x^{10}+x^9+x^8+x^7+x^6+x^5+x^4+x^3+x^2+x+1$	16-bit	0000'FFFF <sub>H</sub>
CRC-16-CCITT	$x^{16}+x^{12}+x^5+1$	16-bit	0000'8810 <sub>H</sub>
CRC-16-IBM	$x^{16}+x^{15}+x^2+1$	16-bit	0000'C002 <sub>H</sub>
CRC-24- Radix-64	$x^{24}+x^{23}+x^{18}+x^{17}+x^{14}+x^{11}+x^{10}+x^7+x^6+x^5+x^4+x^3+x+1$	16-bit (24-bit polynomial)	00C3'267D <sub>H</sub> (BE64'C300 <sub>H</sub> )

**Table 4-1      Some Commonly used Polynomials (cont'd)**

Name	Polynomial	Maximum Data Width	Normal (Reverse) of Reciprocal
CRC-32-IEEE802.3/MPEG2	$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$	16-bit (32-bit polynomial)	8260'8ED6 <sub>H</sub> (DB71'0641 <sub>H</sub> )
CRC-32C	$x^{32}+x^{28}+x^{27}+x^{26}+x^{25}+x^{23}+x^{22}+x^{20}+x^{19}+x^{18}+x^{14}+x^{13}+x^{11}+x^{10}+x^9+x^8+x^6+1$	16-bit (32/27-bit polynomial)	8F6E'37A0 <sub>H</sub> (05EC'76F1 <sub>H</sub> )

*Note: The polynomials above are in general used for sequential signature generation (in general named as CRC), resulting in different signatures than the parallel signature generation algorithm used by the MCHK.*

#### 4.2.4      Architecture of the Memory Checker Module

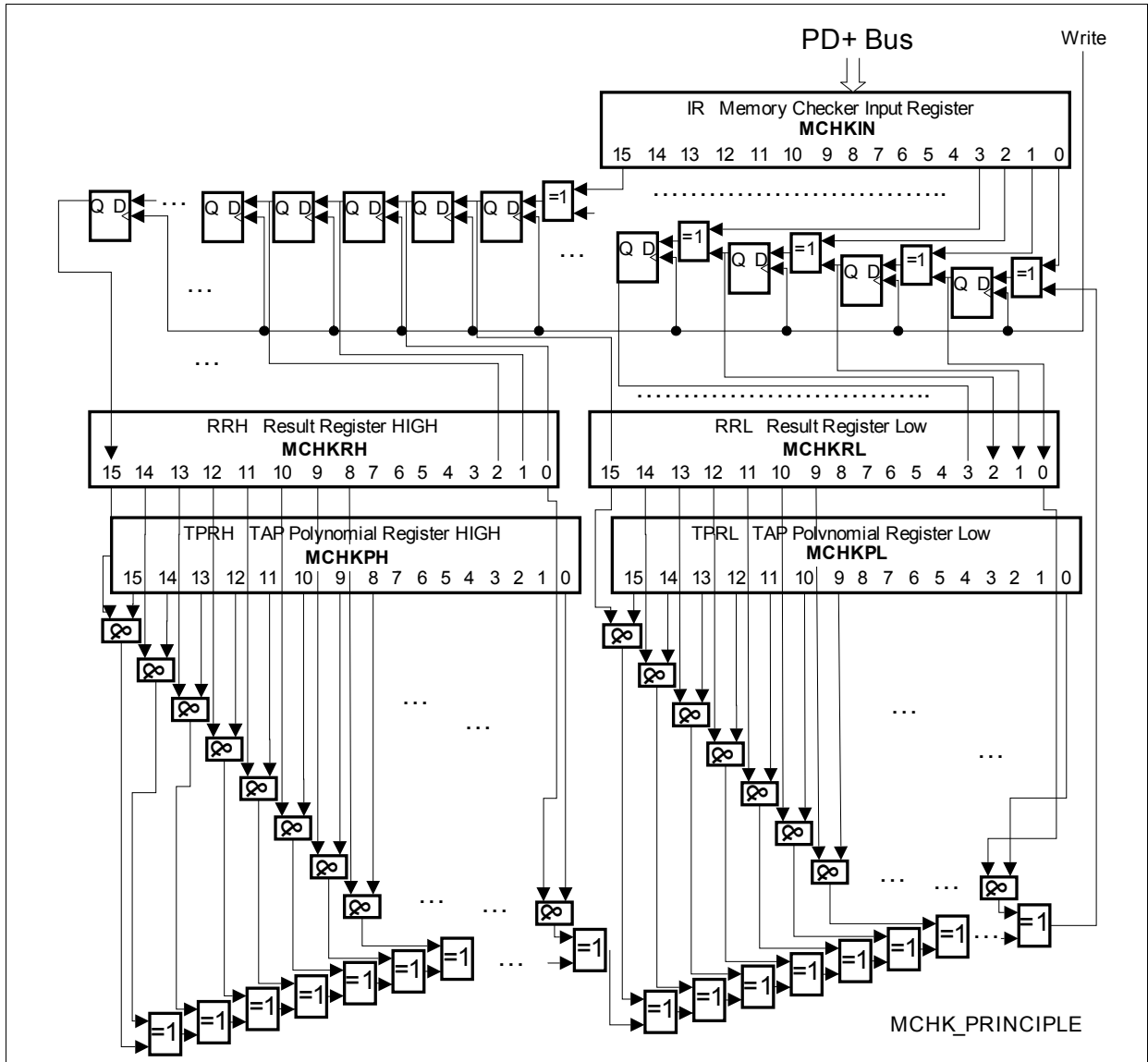
The LFSR is represented by the Result Register High (**RRH**) and the Result Register Low (**RRL**). These may be used to initialize the MCHK with a seed value. When writing to register RRL, the MISR count register **COUNT** is reloaded with the last value written to register **COUNT**. The result registers **RRH** and **RRL** may be used to read the final or intermediate signature of a block of data loaded into the MCHK.

Data may be loaded into the MCHK by writing either 8-bit or 16-bit data into the MISR input register **IR**. Each write access to register IR decrements the content of the MISR count register **COUNT**.

The polynomial is defined by writing the binary normal reciprocal value into the TAP Polynomial Register High (**TPRH**) and TAP Polynomial Register Low (**TPRL**). The TAP polynomial registers and the result registers are combined by a binary AND. If the amount of ones in the result of this AND operation is odd, a 1<sub>B</sub> is fed back, else a 0<sub>B</sub>. The effectiveness of the Memory Checker Module is significantly reduced if a polynomial is used with a the most significant 1<sub>B</sub> bit position in the TAP Polynomial register being smaller than the most significant 1<sub>B</sub> in the data fed into the Memory Checker Module. So in general the content of the TAP polynomial register must be larger than 80<sub>H</sub> for 8-bit data and larger than 8000<sub>H</sub> for 16-bit data.

If the content of the MISR count register **COUNT** is decremented from 0001<sub>H</sub> to 0000<sub>H</sub>, a service request signal is generated in case the content of the LFSR result register high (**RRH**) is not equal FADE<sub>H</sub> or the content of the LFSR result register low (**RRL**) is not equal EDDA<sub>H</sub>. If the content of the LFSR result registers equals FADE'EDDA<sub>H</sub>, the external MATCH signal is toggled. **Figure 4-4** summarizes the architecture of the checksum circuit.

**Memory Checker Module (MCHK)**



**Figure 4-4 Implementation of the MCHK Checksum Circuit**

## 4.2.5 Preferable Usage of the Memory Checker Module

Preferably the MCHK is used together with the CPU. The CPU reads the data block from the selected address area and writes it to the input register **IR**. Alternatively the PEC may be configured to move the data block to input register **IR** using 8-bit or 16-bit moves (PECCx.BWT = 0: 16-bit; PECCx.BWT = 1: 8-bit). Each write operation to register **IR** triggers a intermediate polynomial checksum calculation and the result of the calculation is stored in the result registers **RRL** and **RRH**. Furthermore, every write operation to register **IR** decrements the content of count register **COUNT**.

In order to start a memory check sequence, the result register must be initialized with a seed (e.g. written with the desired start value) and a CPU or PEC transaction must be set up (start address, length, etc.).

When the defined data block is completely written to register **IR**, an interrupt may be generated if the contents of the LFSR result registers **RRH** and **RRL** does not equal FADE'EDDA<sub>H</sub>.

The MCHK may use e.g. the standard Ethernet (IEEE802.3/MPEG2) polynomial, which is given by:

$$G^{32} = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1 \quad (4.2)$$

*Note: Although the polynomial above is used for generation, the result of the parallel signature generation (MISR) differs from the sequential signature generation (LFSR) used by the Ethernet protocol.*

## 4.2.6 Calculation of Seed Values (Magic Word)

To achieve a successful CRC calculation and MATCH or MISMATCH signal on a block of non volatile data, a data specific seed value, the so called magic word, has to be loaded into the LFSR result registers **RRH** and **RRL** prior to the CRC calculation. This magic word should be calculated during development of the respective data. Such a magic word can only be generated, if the order of the TAP polynomial is equal order 32 (most significant bit of TAP polynomial **TPRH** must be equal 1). Otherwise the higher order bits are non equal to the required end result FADE EDDA<sub>H</sub>. The following program sketches the principle of the program. It uses VBA (Microsoft Visual Basic) syntax. The Data\_Array contains all the 16-bit data the CRC is to be calculated. COUNT passes the number of data the CRC is to be calculated, to the subroutine, as defined in the Memory Checker count register **COUNT**. The magical word is passed back to the calling routine through **RRH** and **RRL**, as it has to be written into the Result Register High (**RRH**) and the LFSR Result Register Low (**RRL**) as seed value. Because VBA has no unsigned integer format, a long integer format has been used within this demonstration code.

```
Sub MagicWord(ByRef Data_Array() As Long, _
              ByVal COUNT, TPRH, TPRL as Long, _
              ByRef RRH, RRL As Long)
```



**Memory Checker Module (MCHK)**

```

Dim i, j, order, feedback_bit As Integer
Dim temp As Long
RRH = &HFADE
RRL = &HEDDA
For j = COUNT To 1 Step -1
    If TPRH <> 0 Then                                ' order of polynomial > 16
        order = 31
        Do While TPRH < 2 ^ (order - 16) ' calculate order of polynomial,
            order = order - 1              ' determines bit position
        Loop                               ' "rolled" out of LFSR
    Else                                    ' order of polynomial < 17
        order = 15
        Do While TPRL < 2 ^ order          ' calculate order of polynomial,
            order = order - 1              ' determines bit position
        Loop                               ' "rolled" out of LFSR
    End If
    RRL = RRL Xor Data_Array(j)            ' MISR XOR Input
    feedback_bit = RRL And 1               ' Extract CRC feedback bit
    RRL = RRL \ 2                          ' 32 bit shift right (LFSR)
    RRL = RRL + (RRH And 1) * 2 ^ 15
    RRH = RRH \ 2
    temp = (RRH And TPRH) _
           Xor (RRL And TPRL)             ' generate 32 TAP Bits
    For i = 0 To 15
        feedback_bit = feedback_bit _
                       Xor ((temp \ (2 ^ i)) And 1)
    Next i                                  ' XOR TAP bits to bit
    If feedback_bit <> 0 Then                ' TAP feedback bit is equal 1
        If order > 16 Then
            RRH = RRH Or (2 ^ (order - 16))
        Else
            RRL = RRL Xor (2 ^ order)
        End If
    End If
Next j                                     ' calculate CRC of all data
End Sub

```

## 4.2.7 Example Application

Assuming MCHK is to be used to detect faults within a set of twenty one 16 bit Data. The Memory Checker Module uses the standard Ethernet (IEEE802.3/MPEG2) polynomial, which is given by:

$$G^{32} = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1 \quad (4.3)$$

The Polynomial is therefore 8260 8EDB<sub>H</sub> and is written to **TPRH** and **TPRL**. Next the magical word of this set of data has to be calculated offline using a respective program as described in **Section 4.2.6: “Calculation of Seed Values (Magic Word)” on Page 4-10**. For the given data in **Table 4-2**, a magical word = AA1F ED4E<sub>H</sub> is calculated and written to **RRH** and **RRL**.

**Table 4-2 Example for a CRC Check**

User Action	Data Value	Content of Register				
		COUNT	RRH	RRL	TPRH	TPRL
TAP Polynomial written to <b>TPRH</b>	8260 <sub>H</sub>	xxxx	xxxx	xxxx	8260 <sub>H</sub>	xxxx
TAP Polynomial written to <b>TPRL</b>	8EDB <sub>H</sub>	xxxx	xxxx	xxxx	8260 <sub>H</sub>	8EDB <sub>H</sub>
Magical Word written into <b>RRH</b>	AA1F <sub>H</sub>	0015 <sub>H</sub>	AA1F <sub>H</sub>	xxxx	8260 <sub>H</sub>	8EDB <sub>H</sub>
Magical Word written into <b>RRL</b>	ED4E <sub>H</sub>	0015 <sub>H</sub>	AA1F <sub>H</sub>	ED4E <sub>H</sub>	8260 <sub>H</sub>	8EDB <sub>H</sub>
Data Amount written to <b>COUNT</b>	0015 <sub>H</sub>	0015 <sub>H</sub>	xxxx	xxxx	8260 <sub>H</sub>	8EDB <sub>H</sub>
Data 1 written into <b>IR</b>	8BED <sub>H</sub>	0014 <sub>H</sub>	543F <sub>H</sub>	5171 <sub>H</sub>	8260 <sub>H</sub>	8EDB <sub>H</sub>
Data 2 written into <b>IR</b>	AA61 <sub>H</sub>	0013 <sub>H</sub>	A87E <sub>H</sub>	0883 <sub>H</sub>	8260 <sub>H</sub>	8EDB <sub>H</sub>
Data 3 written into <b>IR</b>	C64E <sub>H</sub>	0012 <sub>H</sub>	50FC <sub>H</sub>	D749 <sub>H</sub>	8260 <sub>H</sub>	8EDB <sub>H</sub>
Data 4 written into <b>IR</b>	17E4 <sub>H</sub>	0011 <sub>H</sub>	A1F9 <sub>H</sub>	B976 <sub>H</sub>	8260 <sub>H</sub>	8EDB <sub>H</sub>
Data 5 written into <b>IR</b>	A329 <sub>H</sub>	0010 <sub>H</sub>	43F3 <sub>H</sub>	D1C5 <sub>H</sub>	8260 <sub>H</sub>	8EDB <sub>H</sub>
Data 6 written into <b>IR</b>	66B5 <sub>H</sub>	000F <sub>H</sub>	87E7 <sub>H</sub>	C53E <sub>H</sub>	8260 <sub>H</sub>	8EDB <sub>H</sub>
Data 7 written into <b>IR</b>	422A <sub>H</sub>	000E <sub>H</sub>	0FCF <sub>H</sub>	C857 <sub>H</sub>	8260 <sub>H</sub>	8EDB <sub>H</sub>
Data 8 written into <b>IR</b>	4FF6 <sub>H</sub>	000D <sub>H</sub>	1F9F <sub>H</sub>	DF58 <sub>H</sub>	8260 <sub>H</sub>	8EDB <sub>H</sub>
Data 9 written into <b>IR</b>	4046 <sub>H</sub>	000C <sub>H</sub>	3F3F <sub>H</sub>	FEF6 <sub>H</sub>	8260 <sub>H</sub>	8EDB <sub>H</sub>
Data 10 written into <b>IR</b>	911C <sub>H</sub>	000B <sub>H</sub>	7E7F <sub>H</sub>	6CF0 <sub>H</sub>	8260 <sub>H</sub>	8EDB <sub>H</sub>
Data 11 written into <b>IR</b>	1FA0 <sub>H</sub>	000A <sub>H</sub>	FCFE <sub>H</sub>	C640 <sub>H</sub>	8260 <sub>H</sub>	8EDB <sub>H</sub>

**Memory Checker Module (MCHK)**

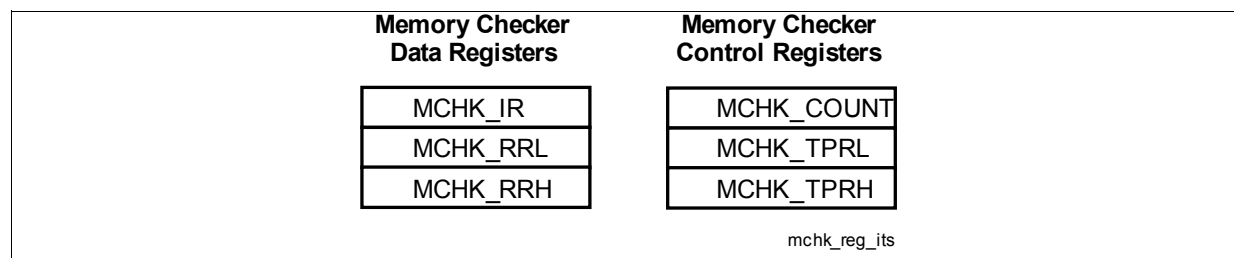
**Table 4-2 Example for a CRC Check (cont'd)**

User Action	Data Value	Content of Register				
		COUNT	RRH	RRL	TPRH	TPRL
Data 12 written into <b>IR</b>	BF38 <sub>H</sub>	0009 <sub>H</sub>	F9FD <sub>H</sub>	33B9 <sub>H</sub>	8260 <sub>H</sub>	8EDB <sub>H</sub>
Data 13 written into <b>IR</b>	9FE3 <sub>H</sub>	0008 <sub>H</sub>	F3FA <sub>H</sub>	F891 <sub>H</sub>	8260 <sub>H</sub>	8EDB <sub>H</sub>
Data 14 written into <b>IR</b>	44DD <sub>H</sub>	0007 <sub>H</sub>	E7F5 <sub>H</sub>	B5FE <sub>H</sub>	8260 <sub>H</sub>	8EDB <sub>H</sub>
Data 15 written into <b>IR</b>	749A <sub>H</sub>	0006 <sub>H</sub>	CFEB <sub>H</sub>	1F67 <sub>H</sub>	8260 <sub>H</sub>	8EDB <sub>H</sub>
Data 16 written into <b>IR</b>	8C09 <sub>H</sub>	0005 <sub>H</sub>	9FD6 <sub>H</sub>	B2C7 <sub>H</sub>	8260 <sub>H</sub>	8EDB <sub>H</sub>
Data 17 written into <b>IR</b>	D0F5 <sub>H</sub>	0004 <sub>H</sub>	3FAD <sub>H</sub>	B57A <sub>H</sub>	8260 <sub>H</sub>	8EDB <sub>H</sub>
Data 18 written into <b>IR</b>	DC5F <sub>H</sub>	0003 <sub>H</sub>	7F5B <sub>H</sub>	B6AB <sub>H</sub>	8260 <sub>H</sub>	8EDB <sub>H</sub>
Data 19 written into <b>IR</b>	DB06 <sub>H</sub>	0002 <sub>H</sub>	FEB7 <sub>H</sub>	B651 <sub>H</sub>	8260 <sub>H</sub>	8EDB <sub>H</sub>
Data 20 written into <b>IR</b>	4604 <sub>H</sub>	0001 <sub>H</sub>	FD6F <sub>H</sub>	2AA7 <sub>H</sub>	8260 <sub>H</sub>	8EDB <sub>H</sub>
Data 21 written into <b>IR</b>	B894 <sub>H</sub>	0000 <sub>H</sub>	FADE <sub>H</sub>	EDDA <sub>H</sub>	8260 <sub>H</sub>	8EDB <sub>H</sub>

## Memory Checker Module (MCHK)

### 4.3 Memory Checker Module Registers

From the programmer's point of view, the MCHK is composed of a set of SFRs as summarized below.



**Figure 4-5 Memory Checker Module Kernel Registers**

The following tables show the MCHK registers and their addresses.

**Table 4-3 Registers Address Space**

Module	Base Address	End Address	Note
MCHK	0000 <sub>H</sub>		

**Table 4-4 Registers Overview**

Register Short Name	Register Long Name	Offset Address	Page Number
<b>ID</b>	Module Identification Register	FFE0 <sub>H</sub>	<a href="#">Page 4-20</a>
<b>IR</b>	Memory Checker Input Register	FE58 <sub>H</sub>	<a href="#">Page 4-15</a>
<b>RRL</b>	Memory Checker Result Register Low	F058 <sub>H</sub>	<a href="#">Page 4-16</a>
<b>RRH</b>	Memory Checker Result Register High	F05A <sub>H</sub>	<a href="#">Page 4-16</a>
<b>COUNT</b>	Memory Checker Count Register	FE5A <sub>H</sub>	<a href="#">Page 4-18</a>
<b>TPRL</b>	Memory Checker Polynomial Register Low	F05C <sub>H</sub>	<a href="#">Page 4-19</a>
<b>TPRH</b>	Memory Checker Polynomial Register High	F05E <sub>H</sub>	<a href="#">Page 4-19</a>

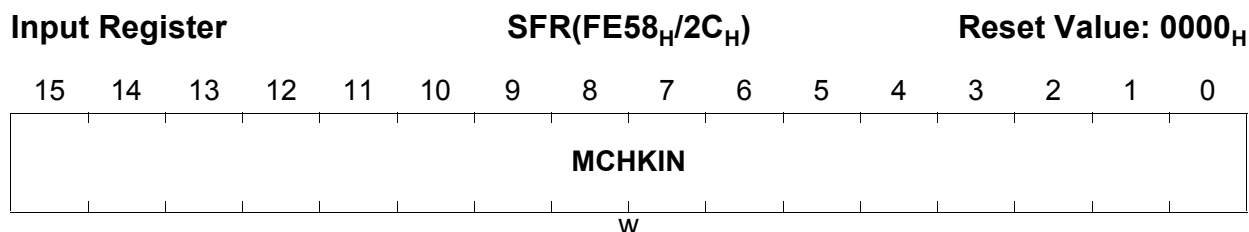
*Note: All registers are reset by the same reset class as the CPU is reset.*

**Memory Checker Module (MCHK)**

### 4.3.1 Memory Checker Input Register

The input register receives the data written to the MCHK for checksum calculation. If the CPU moves to register MCHK\_IR are 8-bit wide, the unused register bits of the 16-bit MCHKIN value are taken as 0s for the current result calculation.

**IR**



Field	Bits	Type	Description
MCHKIN	[15:0]	w	<b>Memory Checker Module Input</b> The value written to MCHKIN is used for the next checksum calculation. Any read action will deliver 0000 <sub>H</sub> .

*Note: MCHK\_IR is a write-only register. Any read action will deliver 0000<sub>H</sub>.*

### 4.3.2 Memory Checker Result Registers

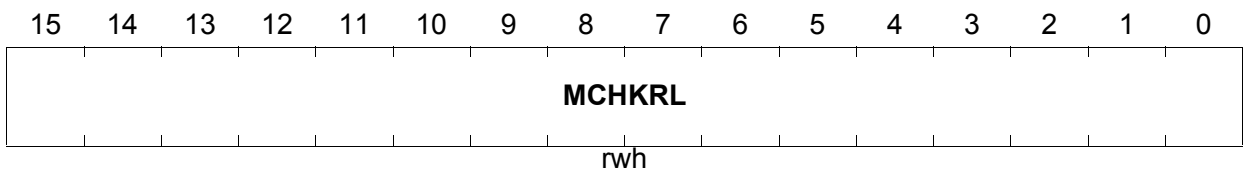
The result registers contain the signature (result) of the memory check operation. Before starting a checksum calculation operation, they should be written with the initial checksum calculation value (seed).

#### RRL

##### Result Register Low

ESFR(F058<sub>H</sub>/2C<sub>H</sub>)

Reset Value: 0000<sub>H</sub>



Field	Bits	Type	Description
MCHKRL	[15:0]	rwh	<b>Memory Checker Result Low</b> This bit field contains the least significant 16 bits of the current result of the 32-bit checksum calculation operation.

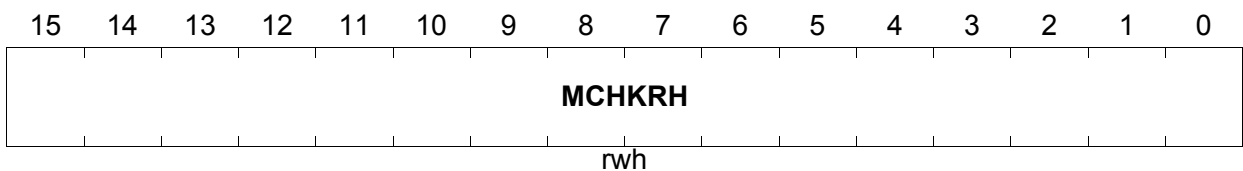
*Note: Writing to the RRL.MCHKRL will reset (reload) the MCHKCNT.COUNT register to the last data written to this register MCHKCNT.COUNT value. Therefore writing to RRL will immediately initialize a new CRC calculation cycle.*

#### RRH

##### Result Register High

ESFR(F05A<sub>H</sub>/2D<sub>H</sub>)

Reset Value: 0000<sub>H</sub>



Field	Bits	Type	Description
MCHKRH	[15:0]	rwh	<b>Memory Checker Result High</b> This bit field contains the most significant 16 bits of the current result of the 32-bit checksum calculation operation.

### **4.3.3 Memory Checker Count Register**

The count register COUNT is decremented on each write access to the input register. If the count register is decremented to  $0000_H$ , a service request (interrupt) is generated if the content of the result registers is non equal  $FADE'EDDA_H$ , or instead the output signal MATCH is toggled if the result registers are equal  $FADE'EDDA_H$ . The count register is reloaded with the last value written to it, when the CPU transfers a new seed value (magic word) to the LFSR result register low (**RRL**).

When the CPU or PEC writes to register COUNT and its content is not equal to the last value written to it, the service request (interrupt) is generated and the output signal MATCH is toggled. This enables detection of software not correctly handling the MCHK, e.g. due to an erroneous program memory. The timely correct toggling of the MATCH signal may be used as a life signal e.g. by an external window watchdog. The reset value of the MATCH signal =  $0_B$ .

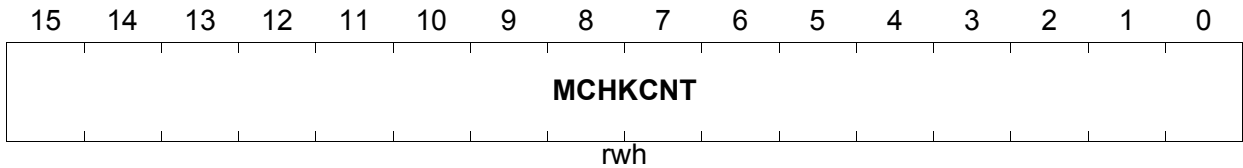
Because register COUNT controls a safety critical system function, it is protected by a special register security mechanism so this vital system function cannot be changed inadvertently after executing the EINIT instruction.

**COUNT**

**Count Register**

**SFR(FE5A<sub>H</sub>/2D<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
MCHKCNT	[15:0]	rwh	<b>Memory Checker Count</b> MCHKCNT indicates the number of remaining data in the current data block to be entered into MCHK. 0001 <sub>H</sub> One remaining data to be written to register <b>IR</b> to trigger the compare for the MATCH signal, interrupt signal NOMATCH. 0002 <sub>H</sub> Two remaining data to be written to register <b>IR</b> . ... <sub>H</sub> ... FFFE <sub>H</sub> 65534 remaining data to be written to register <b>IR</b> . FFFF <sub>H</sub> 65535 remaining data to be written to register <b>IR</b> . 0000 <sub>H</sub> 65536 remaining data to be written to register <b>IR</b> to trigger the compare for the MATCH signal, interrupt signal NOMATCH.

*Note: Register COUNT should only be written if MCHKCNT is equal to the last value written to this register. Otherwise, a service request (interrupt) will be triggered and the MATCH signal will be toggled.*

*Modify register COUNT only after writing to register **RRL**, which reloads COUNT to the previously written value (see also **Table 4-2**).*

*Note: COUNT is write protected after the execution of EINIT by the register security mechanism.*

### 4.3.4 Memory Checker Polynomial Registers

The polynomial registers contain the LFSR polynomial of the checksum calculation operation.

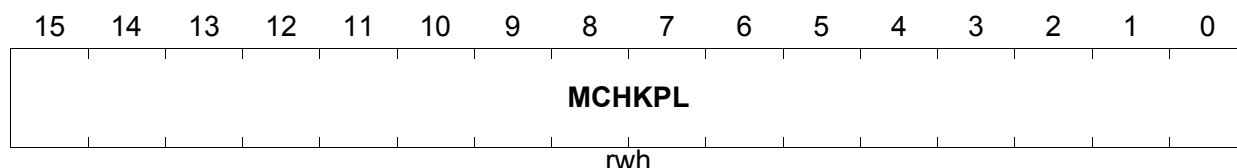


**Memory Checker Module (MCHK)**

Because the polynomial registers control a safety critical system function, they are protected by a special register security mechanism so this vital system function cannot be changed inadvertently after executing the EINIT instruction.

**TPRL**

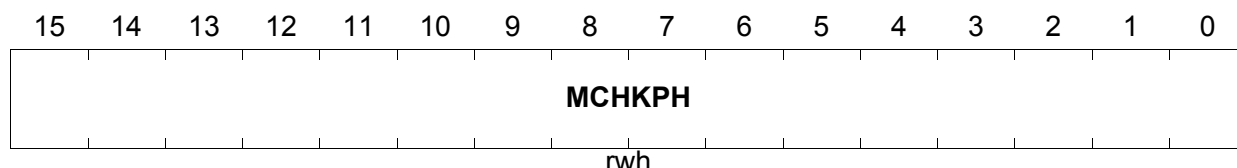
**Tap Polynomial Register Low**    **ESFR(F05C<sub>H</sub>/2E<sub>H</sub>)**                      **Reset Value: FFFF<sub>H</sub>**



Field	Bits	Type	Description
<b>MCHKPL</b>	[15:0]	rwh	<b>Memory Checker Polynomial Low</b> This bit field contains the least significant 16 bits of the binary tap polynomial format.

**TPRH**

**Tap Polynomial Register High**    **ESFR(F05E<sub>H</sub>/2F<sub>H</sub>)**                      **Reset Value: FFFF<sub>H</sub>**



Field	Bits	Type	Description
<b>MCHKPH</b>	[15:0]	rwh	<b>Memory Checker Polynomial High</b> This bit field contains the most significant 16 bits of the binary tap polynomial format.

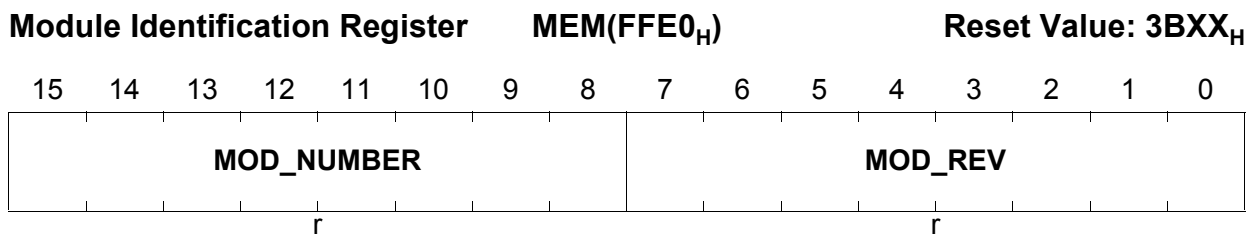
*Note: TPRH and TPRL is write protected after the execution of EINIT by the register security mechanism.*

## 4.4 General Registers

The ID register is a read-only register used for MCHK module identification purposes. It provides 8 bits for module identification and 8 bits for revision numbering.

### 4.4.1 ID Register

#### ID



Field	Bits	Type	Description
MOD_REV	[7:0]	r	<b>Module Revision Number Value</b> Bits 7-0 bits are used for module revision numbering. The value of the module revision number starts with 01 <sub>H</sub> (first revision).
MOD_NUMBER	[15:8]	r	<b>Module Identification Number Value</b> Bits 15-8 are used for module identification. The MCHK has the module number 3B <sub>H</sub> .

## 4.5 Interfaces of the MCHK Module

The MCHK module can generate an interrupt request and an external life signal.

The interrupt request signal is connected to the SCU and can be routed to one of the SCU's interrupt nodes.

The MATCH output is connected to an external pin in packaged devices with 144 or more pins only.

**Table 4-5 MCHK Digital Connections in XE16xyM**

Signal	from/to Module	I/O to MCHK
MATCH	P8.6	O
INT (MISMATCH)	SCU	O
MCHKIN_Write_Access	Write trigger from MCHKIN	I <sup>1)</sup>

**Memory Checker Module (MCHK)**

**Table 4-5 MCHK Digital Connections in XE16xyM (cont'd)**

<b>Signal</b>	<b>from/to Module</b>	<b>I/O to MCHK</b>
MCHKCNT_Write_Access	Write trigger from MCHKCNT	I <sup>1)</sup>
EINIT	SCU	I

1) This signal is generated within the module itself and is not present at the module boundary.

## **5 Central Processing Unit (CPU)**

Basic tasks of the Central Processing Unit (CPU) are to fetch and decode instructions, to supply operands for the Arithmetic and Logic unit (ALU) and the Multiply and Accumulate unit (MAC), to perform operations on these operands in the ALU and MAC, and to store the previously calculated results. As the CPU is the main engine of the XE16xyM microcontroller, it is also affected by certain actions of the peripheral subsystem.

Because a five-stage processing pipeline (plus 2-stage fetch pipeline) is implemented in the XE16xyM, up to five instructions can be processed in parallel. Most instructions of the XE16xyM are executed in one single clock cycle due to this parallelism.

This chapter describes how the pipeline works for sequential and branch instructions in general, and the hardware provisions which have been made to speed up execution of jump instructions in particular. General instruction timing is described, including standard timing, as well as exceptions.

While internal memory accesses are normally performed by the CPU itself, external peripheral or memory accesses are performed by a particular on-chip External Bus Controller (EBC) which is invoked automatically by the CPU whenever a code or data address refers to the external address space.

Whenever possible, the CPU continues operating while an external memory access is in progress. If external data are required but are not yet available, or if a new external memory access is requested by the CPU before a previous access has been completed, the CPU will be held by the EBC until the request can be satisfied. The EBC is described in a separate chapter.

The on-chip peripheral units of the XE16xyM operate independently of the CPU. Data and control information are interchanged between the CPU and these peripherals via Special Function Registers (SFRs) or shared memory areas.

Whenever peripherals need a non-deterministic CPU action, an on-chip Interrupt Controller compares all pending peripheral service requests against each other and prioritizes one of them. If the priority of the current CPU operation is lower than the priority of the selected peripheral request, an interrupt will occur.

There are two basic types of interrupt processing:

- **Standard interrupt processing** forces the CPU to save the current program status and return address on the stack before branching to the interrupt vector jump table.
- **PEC interrupt processing** steals only one machine cycle from the current CPU activity to perform a single data transfer via the on-chip Peripheral Event Controller (PEC).

System errors detected during program execution (hardware traps) and external non-maskable interrupts are also processed as standard interrupts with a very high priority.

## **Central Processing Unit (CPU)**

In contrast to other on-chip peripherals, there is a closer conjunction between the watchdog timer and the CPU. If enabled, the watchdog timer expects to be serviced by the CPU within a programmable period of time, otherwise it will reset the chip. Thus, the watchdog timer is able to prevent the CPU from going astray when executing erroneous code. The CPU provides a set of instructions for enabling (ENWDT), disabling (DISWDT) and servicing (SRVWDT) the watchdog timer.

In addition to its active operation state, the CPU can enter idle mode by executing the IDLE instruction. In idle mode the CPU stops program execution but still reacts to interrupt or PEC requests. Transition to the active state can be forced by an interrupt request or a hardware reset.

The PWRDN instruction is not enabled in the XE16xyM. If executed a NOP will be performed instead. System power state transitions are controlled by the System Control Unit (SCU).

A set of Special Function Registers is dedicated to the CPU core (CSFRs):

- CPU Status Indication and Control: **PSW, CPUCON1, CPUCON2**
- Code Access Control: **IP, CSP**
- Data Paging Control: **DPP0, DPP1, DPP2, DPP3**
- Global GPRs Access Control: **CP**
- System Stack Access Control: **SP, SPSEG, STKUN, STKOV**
- Multiply and Divide Support: **MDL, MDH, MDC**
- Indirect Addressing Offset: **QR0, QR1, QX0, QX1**
- MAC Address Pointers: **IDX0, IDX1**
- MAC Status Indication and Control: **MCW, MSW, MAH, MAL, MRW**
- ALU Constants Support: **ZEROS, ONES**
- CPU identification: **CPUID**

The CPU also uses CSFRs to access the General Purpose Registers (GPRs). Since all CSFRs can be controlled by any instruction capable of addressing the SFR/CSFR memory space, there is no need for special system control instructions.

However, to ensure proper processor operation, certain restrictions on the user access to some CSFRs must be imposed. For example, the instruction pointer (CSP, IP) cannot be accessed directly at all. These registers can only be changed indirectly via branch instructions. Registers PSW, SP, and MDC can be modified not only explicitly by the programmer, but also implicitly by the CPU during normal instruction processing.

*Note: Note that any explicit write request (via software) to an CSFR supersedes a simultaneous modification by hardware of the same register.*

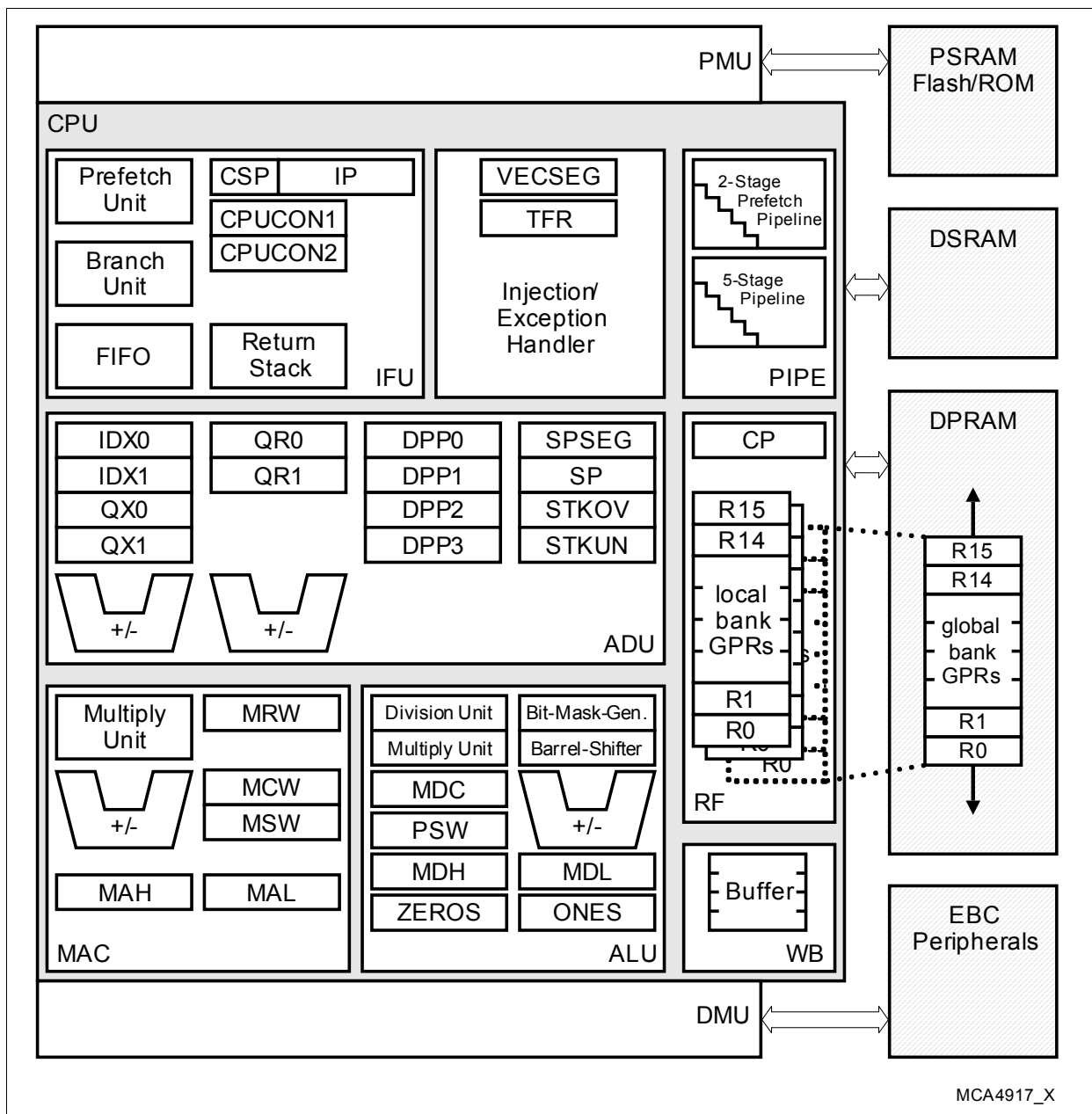
**Central Processing Unit (CPU)**

All CSFRs may be accessed wordwise, or bytewise (some of them even bitwise). Reading bytes from word CSFRs is a non-critical operation. Any write operation to a single byte of a CSFR clears the non-addressed complementary byte within the specified CSFR.

***Attention: Reserved CSFR bits must not be modified explicitly, and will always supply a read value of 0. If a byte/word access is preferred by the programmer or is the only possible access the reserved CSFR bits must be written with 0 to provide compatibility with future versions.***

## 5.1 Components of the CPU

The high performance of the CPU results from the cooperation of several units which are optimized for their respective tasks (see [Figure 5-1](#)). **Prefetch Unit** and **Branch Unit** feed the pipeline minimizing CPU stalls due to instruction reads. The **Address Unit (ADU)** supports sophisticated addressing modes avoiding additional instructions needed otherwise. **Arithmetic and Logic Unit (ALU)** and **Multiply and Accumulate Unit (MAC)** handle differently sized data and execute complex operations. **Three memory interfaces** and **Write Buffer (WB)** minimize CPU stalls due to data transfers.



**Figure 5-1 CPU Block Diagram**

In general the instructions move through 7 pipeline stages ([Section 5.3](#)). The stages can be grouped as follows:

- **2 stages fetch pipeline** - receives instructions from program memory and stores them into an instruction FIFO. Fetch pipeline stages can be bypassed.
- **5 stages processing pipeline** - executes each instruction received from fetch stages.

Because passing through one pipeline stage takes at least one clock cycle and because the fetch pipeline stages can be bypassed, any isolated instruction takes at least five clock cycles to be completed. Pipelining, however, allows parallel (i.e. simultaneous) processing of up to five instructions (with branches up to six instructions). Therefore, most of the instructions appear to be processed during one clock cycle as soon as the pipeline has been filled once after reset.

The pipelining increases the average instruction throughput considered over a certain period of time.

## **5.2 Instruction Fetch and Program Flow Control**

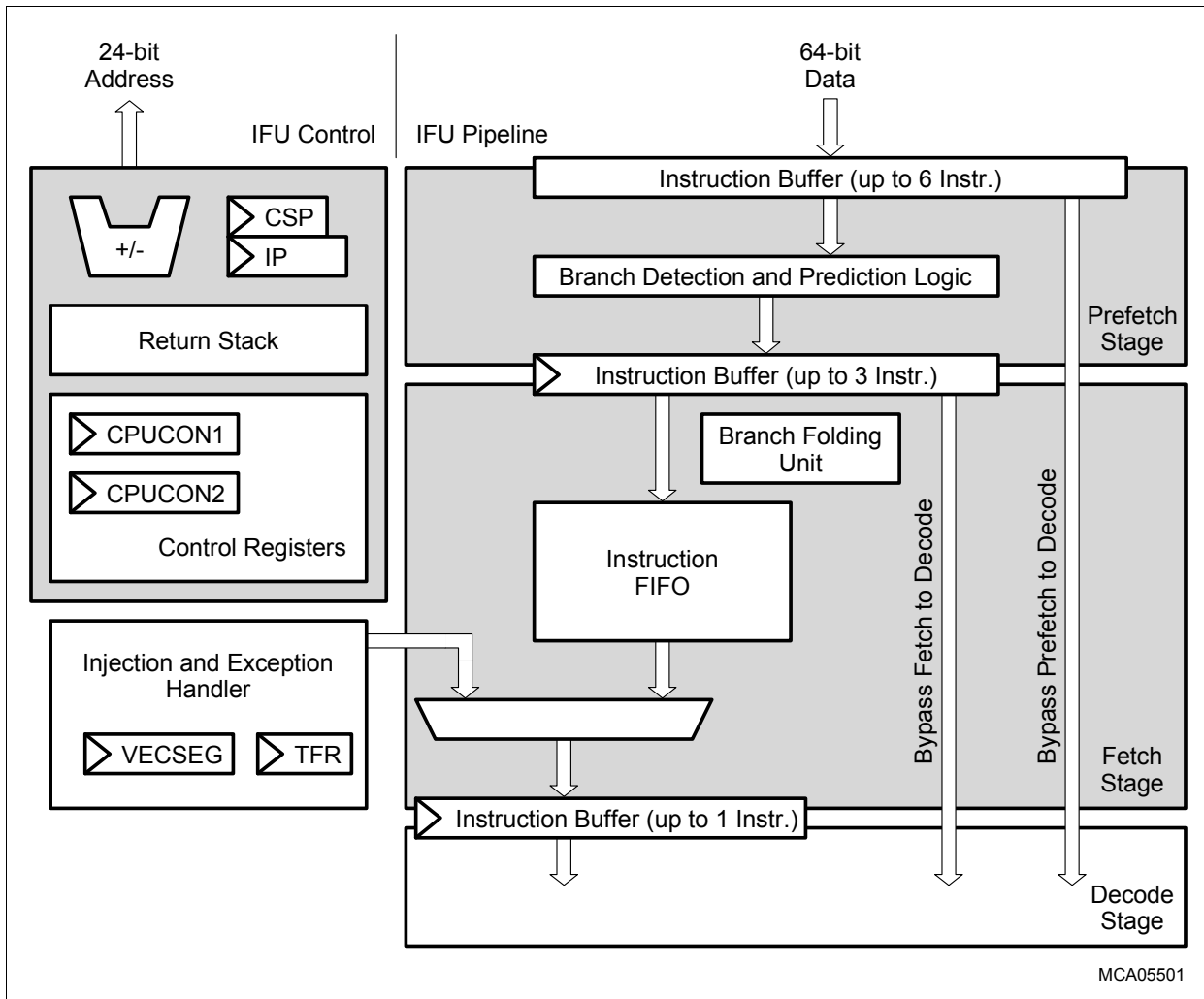
The Instruction Fetch Unit (IFU) prefetches and preprocesses instructions to provide a continuous instruction flow. The IFU can fetch simultaneously at least two instructions via a 64-bit wide bus from the Program Management Unit (PMU). The prefetched instructions are stored in an instruction FIFO.

Preprocessing of branch instructions enables the instruction flow to be predicted. While the CPU is in the process of executing an instruction fetched from the FIFO, the prefetcher of the IFU starts to fetch a new instruction at a predicted target address from the PMU. The latency time of this access is hidden by the execution of the instructions which have already been buffered in the FIFO. Even for a non-sequential instruction execution, the IFU can generally provide a continuous instruction flow. The IFU contains two pipeline stages: the Prefetch Stage and the Fetch Stage.

During the prefetch stage, the Branch Detection and Prediction Logic analyzes up to three prefetched instructions stored in the first Instruction Buffer (can hold up to six instructions). If a branch is detected, then the IFU starts to fetch the next instructions from the PMU according to the prediction rules. After having been analyzed, up to three instructions are stored in the second Instruction Buffer (can hold up to three instructions) which is the input register of the Fetch Stage.

In the case of an incorrectly predicted instruction flow, the instruction fetch pipeline is bypassed to reduce the number of dead cycles.





**Figure 5-2 IFU Block Diagram**

On the Fetch Stage, the prefetched instructions are stored in the instruction FIFO. The Branch Folding Unit (BFU) allows processing of branch instructions in parallel with preceding instructions. To achieve this the BFU preprocesses and reformats the branch instruction. First, the BFU defines (calculates) the absolute target address. This address — after being combined with branch condition and branch attribute bits — is stored in the same FIFO step as the preceding instruction. The target address is also used to prefetch the next instructions.

For the Processing Pipeline, both instructions are fetched from the FIFO again and are executed in parallel. If the instruction flow was predicted incorrectly (or FIFO is empty), the two stages of the IFU can be bypassed.

*Note: Pipeline behavior in case of a incorrectly predicted instruction flow is described in the following sections.*

### 5.2.1 Branch Detection and Branch Prediction Rules

The Branch Detection Unit preprocesses instructions and classifies detected branches. Depending on the branch class, the Branch Prediction Unit predicts the program flow using the following rules:

**Table 5-1 Branch Classes and Prediction Rules**

Branch Instruction Classes	Instructions	Prediction Rule (Assumption)
Inter-segment branch instructions	JMPS seg, caddr CALLS seg, caddr	The branch is always taken
Branch instructions with user programmable branch prediction	JMPA- xcc, caddr JMPA+ xcc, caddr CALLA- xcc, caddr CALLA+ xcc, caddr	User-specified <sup>1)</sup> via bit 8 ('a') of the instruction long word: ...+: branch 'taken' (a = 0) ...-: branch 'not taken' (a = 1)
Indirect branch instructions	JMPI cc, [Rw] CALLI cc, [Rw]	Unconditional: branch 'taken' Conditional: 'not taken'
Relative branch instructions with condition code	JMPR cc, rel	Unconditional or backward: branch 'taken' Conditional forward: 'not taken'
Relative branch instructions without condition code	CALLR rel	The branch is always taken
Branch instructions with bit-condition	JB(C) bitaddr, rel JNB(S) bitaddr, rel	Backward: branch 'taken' Forward: 'not taken'
Return instructions	RET, RETP RETS, RETI	The branch is always taken

1) This bit can be also set/cleared automatically by the Assembler for generic JMPA and CALLA instructions depending on the jump condition

### 5.2.2 Zero-Cycle Jumps

The **"Zero-Cycle Jumps"** are one of the advanced XE16xyM specifics, which becomes possible due to the complex pipelined structure for processing instruction-flow.

This feature allows, under some circumstances, jumps to be executed in "null time". In fact, a jump is "hooked" to the previous instruction and the two instructions pass through the pipeline as one instruction. This can be only possible, if the jump instruction does not need any of the pipeline resources needed by the predecessor. Hence, the following rules are essential:

- a jump can not be hooked onto another jump instruction, as the pipeline resource "target IP" can not be shared between the two;

### Central Processing Unit (CPU)

- a jump can not be executed in zero-cycle if it requires any memory access, as basically any predecessor instruction might access a memory.

The above are only preliminary conditions, needed to make a jump zero-cycle. But would this really happen, it's not reliable enough to predict: it also depends on the exact instruction sequence, speed of the program memory etc.

What can be summarized is:

- only **JMPA**, **JMPR** and **JMPS** Instructions **can be** converted to zero-cycle; **if** the immediately preceding instruction **is not** a branch (any **JMP**, **CALL** or **RET**).
- If a Jump is executed as zero-cycle, in fact the address of this Jump will not be assigned to the Instruction Pointer.

*Note: No IP-Breakpoint must be set over an instruction, which satisfies the two prepositions above for a zero-cycle Jump. Otherwise, if set, it is **very possible** this Breakpoint will be missed by the debug module.*

### 5.2.3 Atomic and Extend Instructions

The atomic and extend instructions (**ATOMIC**, **EXTR**, **EXTP**, **EXTS**, **EXTPR**, **EXTSR**) disable standard and PEC interrupts and class A traps until completion of the immediately following sequence of instructions. The number of instructions in the sequence may vary from 1 to 4. It is coded in the 2-bit constant field **#irang2** and takes values from 0 to 3. The **EXTENDED** instructions additionally change the addressing mechanism during this sequence (see instruction description).

**ATOMIC** and **EXTENDED** instructions become active immediately, so no additional **NOPs** are required. All instructions requiring multi cycles or hold states for execution are considered to be one instruction. The **ATOMIC** and **EXTENDED** instructions can be used with any instruction type.

If a branch instruction following immediately after an atomic sequence is executed as zero-cycle jump, then this branch is part of the atomic sequence as well. If the branch instruction is not a part of the **ATOMIC** sequence, it should not be hooked on to the atomic sequence, a **NOP** could be inserted in between.

*Note: If a class B trap interrupt occurs during an **ATOMIC** or **EXTENDED** sequence, then the sequence is terminated, an interrupt lock is removed, and the standard condition is restored before the trap routine is executed. The remaining instructions of the terminated sequence executed after returning from the trap routine will run under standard conditions.*

*Note: When using nested **ATOMIC** and **EXTENDED** instructions. There is only one counter to control the length of the sequence, i.e. issuing an **ATOMIC** or **EXTENDED** instruction within a sequence will reload the counter with the value of the new instruction.*

### **5.3 Instruction Processing Pipeline**

The XE16xyM uses five pipeline stages to execute an instruction. All instructions pass through each of the five stages of the instruction processing pipeline. The pipeline stages are listed here together with the 2 stages of the fetch pipeline:

**1st -> PREFETCH:** This stage prefetches instructions from the PMU in the predicted order. The instructions are preprocessed in the branch detection unit to detect branches. The prediction logic decides if the branches are assumed to be taken or not.

**2nd -> FETCH:** The instruction pointer of the next instruction to be fetched is calculated according to the branch prediction rules. For zero-cycle branch execution, the Branch Folding Unit preprocesses and combines detected branches with the preceding instructions. Prefetched instructions are stored in the instruction FIFO. At the same time, instructions are transported out of the instruction FIFO to be executed in the instruction processing pipeline.

**3rd -> DECODE:** The instructions are decoded and, if required, the register file is accessed to read the GPR used in indirect addressing modes.

**4th -> ADDRESS:** All the operand addresses are calculated. Register SP is decremented or incremented for all instructions which implicitly access the system stack.

**5th -> MEMORY:** All the required operands are fetched.

**6th -> EXECUTE:** An ALU or MAC-Unit operation is performed on the previously fetched operands. The condition flags are updated. All explicit write operations to CPU-SFRs and all auto-increment/auto-decrement operations of GPRs used as indirect address pointers are performed.

**7th -> WRITE BACK:** All external operands and the remaining operands within the internal DPRAM space are written back. Operands located in the internal SRAM are buffered in the Write Back Buffer.

Specific so-called injected instructions are generated internally to provide the time needed to process instructions requiring more than one CPU cycle for processing. They are automatically injected into the decode stage of the pipeline, then they pass through the remaining stages like every standard instruction. Program interrupt, PEC transfer, and debug operations are also performed by means of injected instructions. Although these internally injected instructions will not be noticed in reality, they help to explain the operation of the pipeline.

The performance of the CPU (pipeline) is decreased by bandwidth limitations (same resource is accessed by different stages) and data dependencies between instructions. The XE16xyM's CPU has dedicated hardware to detect and to resolve different kinds of dependencies. Some of those dependencies are described in the following section.

Because up to five different instructions are processed simultaneously, additional hardware has been dedicated to deal with dependencies which may exist between instructions in different pipeline stages. This extra hardware supports 'forwarding' of the operand read and write values and resolves most of the possible conflicts — such as

multiple usage of buses — in a time optimized way without performance loss. This makes the pipeline unnoticeable for the user in most cases. However, there are some cases in which the pipeline requires attention by the programmer.

### **5.3.1 Access to the IO Area**

Read or write accesses to the IO Areas of the XE16xyM memory space enforce particular pipeline behavior. Thus the requirements of peripheral devices with registers located in these areas are handled appropriately.

The following typical properties of peripheral device registers are considered:

- Upon a write to a peripheral register the contents of any (also multiple) peripheral register(s) may change as a consequence of the write.
- Upon a read from a peripheral register the contents of the same register may change as a consequence of the read (e.g. read buffer of a serial channel)

These cases are handled by following pipeline measures:

#### **Write before read execution enforced**

If the instructions in the pipeline contain a write action followed by a read action both to the IO areas then the read action is delayed (held in memory stage) until the write action has passed through the writeback stage. Thus the write action will always be scheduled before a read action.

***Attention: Due to additional system delay this does not guarantee that a write will become effective before a read at the target registers.***

Additional system delay is accumulated by the bus system or caused by the peripheral itself. In case the additional read delay differs from the write delay the read may overtake the write. However since the on-chip delays are similar the programmer must take care about this in particular when using off-chip peripherals allocating IO area through the EXTBUS.

#### **Prevention of buffered writes**

Write access to the IO area is not buffered in the writeback buffer.

### **5.3.2 Pipeline Conflicts**

The following examples describe the pipeline behavior in special cases and give provide rules to optimize performance by instruction re-ordering.

*Note: The XE16xyM has a fully interlocked pipeline, which means that pipeline conflicts do not cause any malfunction. Instruction re-ordering is only required for performance reasons.*

### 5.3.2.1 Using General Purpose Registers

The GPRs are the working registers of the CPU and there are a lot of possible dependencies between instructions using GPRs. A high-speed five-port register file prevents bandwidth conflicts. Dedicated hardware is implemented to detect and resolve the data dependencies. Special forwarding buses are used to forward GPR values from one pipeline stage to another. In most cases, this allows the execution of instructions without any delay despite of data dependencies.

Conflict\_GPRs\_Resolved:

```
In      ADD R0,R1      ;Compute new value for R0
In+1    ADD R3,R0      ;Use R0 again
In+2    ADD R6,R0      ;Use R0 again
In+3    ADD R6,R1      ;Use R6 again
```

**Table 5-2 Resolved Pipeline Dependencies Using GPRs**

Stage	T <sub>n</sub>	T <sub>n+1</sub>	T <sub>n+2</sub>	T <sub>n+3</sub>	T <sub>n+4</sub> <sup>1)</sup>	T <sub>n+5</sub> <sup>2)</sup>
<b>DECODE</b>	I <sub>n</sub> = ADD R0, R1	I <sub>n+1</sub> = ADD R3, R0	I <sub>n+2</sub> = ADD R6, R0	I <sub>n+3</sub> = ADD R6, R1	I <sub>n+4</sub>	I <sub>n+5</sub>
<b>ADDRESS</b>	I <sub>n-1</sub>	I <sub>n</sub> = ADD R0, R1	I <sub>n+1</sub> = ADD R3, R0	I <sub>n+2</sub> = ADD R6, R0	I <sub>n+3</sub> = ADD R6, R1	I <sub>n+4</sub>
<b>MEMORY</b>	I <sub>n-2</sub>	I <sub>n-1</sub>	I <sub>n</sub> = ADD R0, R1	I <sub>n+1</sub> = ADD R3, <b>R0</b>	I <sub>n+2</sub> = ADD R6, <b>R0</b>	I <sub>n+3</sub> = ADD <b>R6</b> , R1
<b>EXECUTE</b>	I <sub>n-3</sub>	I <sub>n-2</sub>	I <sub>n-1</sub>	I <sub>n</sub> = ADD <b>R0</b> , R1	I <sub>n+1</sub> = ADD R3, R0	I <sub>n+2</sub> = ADD <b>R6</b> , R0
<b>WR.BACK</b>	I <sub>n-4</sub>	I <sub>n-3</sub>	I <sub>n-2</sub>	I <sub>n-1</sub>	I <sub>n</sub> = ADD <b>R0</b> , R1	I <sub>n+1</sub> = ADD R3, R0

1) R0 forwarded from WRITE BACK to MEMORY.

2) R6 forwarded from EXECUTE to MEMORY.

However, if a GPR is used for indirect addressing the address pointer (i.e. the GPR) will be required already in the DECODE stage. In this case the instruction is stalled in the address stage until the operation in the ALU is executed and the result is forwarded to the address stage.

Conflict\_GPRs\_Pointer\_Stall:

```
In      ADD R0,R1      ;Compute new value for R0
In+1    MOV R3,[R0]    ;Use R0 as address pointer
In+2    ADD R6,R0
In+3    ADD R6,R1
```

**Table 5-3 Pipeline Dependencies Using GPRs as Pointers (Stall)**

Stage	$T_n$	$T_{n+1}$	$T_{n+2}^{1)}$	$T_{n+3}^{2)}$	$T_{n+4}$	$T_{n+5}$
<b>DECODE</b>	$I_n = \text{ADD R0, R1}$	$I_{n+1} = \text{MOV R3, [R0]}$	$I_{n+2}$	$I_{n+2}$	$I_{n+2}$	$I_{n+3}$
<b>ADDRESS</b>	$I_{n-1}$	$I_n = \text{ADD R0, R1}$	$I_{n+1} = \text{MOV R3, [R0]}$	$I_{n+1} = \text{MOV R3, [R0]}$	$I_{n+1} = \text{MOV R3, [R0]}$	$I_{n+2}$
<b>MEMORY</b>	$I_{n-2}$	$I_{n-1}$	$I_n = \text{ADD R0, R1}$	—	—	$I_{n+1} = \text{MOV R3, [R0]}$
<b>EXECUTE</b>	$I_{n-3}$	$I_{n-2}$	$I_{n-1}$	$I_n = \text{ADD R0, R1}$	—	—
<b>WR.BACK</b>	$I_{n-4}$	$I_{n-3}$	$I_{n-2}$	$I_{n-1}$	$I_n = \text{ADD R0, R1}$	—

1) New value of R0 not yet available.

2) R0 forwarded from EXECUTE to ADDRESS (next cycle).

To avoid these stalls, one multicycle instruction or two single cycle instructions may be inserted. These instructions must not update the GPR used for indirect addressing.

Conflict\_GPRs\_Pointer\_NoStall:

```

 $I_n$       ADD R0,R1      ;Compute new value for R0
 $I_{n+1}$     ADD R6,R0      ;R0 is not updated, just read
 $I_{n+2}$     ADD R6,R1
 $I_{n+3}$     MOV R3,[R0]    ;Use R0 as address pointer

```

**Table 5-4 Pipeline Dependencies Using GPRs as Pointers (No Stall)**

Stage	$T_n$	$T_{n+1}$	$T_{n+2}$	$T_{n+3}^{1)}$	$T_{n+4}$	$T_{n+5}$
<b>DECODE</b>	$I_n = \text{ADD R0, R1}$	$I_{n+1} = \text{ADD R6, R0}$	$I_{n+2} = \text{ADD R6, R1}$	$I_{n+3} = \text{MOV R3, [R0]}$	$I_{n+4}$	$I_{n+5}$
<b>ADDRESS</b>	$I_{n-1}$	$I_n = \text{ADD R0, R1}$	$I_{n+1} = \text{ADD R6, R0}$	$I_{n+2} = \text{ADD R6, R1}$	$I_{n+3} = \text{MOV R3, [R0]}$	$I_{n+4}$
<b>MEMORY</b>	$I_{n-2}$	$I_{n-1}$	$I_n = \text{ADD R0, R1}$	$I_{n+1} = \text{ADD R6, R0}$	$I_{n+2} = \text{ADD R6, R1}$	$I_{n+3} = \text{MOV R3, [R0]}$
<b>EXECUTE</b>	$I_{n-3}$	$I_{n-2}$	$I_{n-1}$	$I_n = \text{ADD R0, R1}$	$I_{n+1} = \text{ADD R6, R0}$	$I_{n+2} = \text{ADD R6, R1}$
<b>WR.BACK</b>	$I_{n-4}$	$I_{n-3}$	$I_{n-2}$	$I_{n-1}$	$I_n = \text{ADD R0, R1}$	$I_{n+1} = \text{ADD R6, R0}$

1) R0 forwarded from EXECUTE to ADDRESS (next cycle).

### 5.3.2.2 Using Indirect Addressing Modes

In the case of read accesses using indirect addressing modes, the Address Generation Unit uses a speculative addressing mechanism. The read data path to one of the different memory areas (DPRAM, DSRAM, etc.) is selected according to a history table before the address is decoded. This history table has one entry for each of the GPRs. The entries store the information of the last accessed memory area using the corresponding GPR. In the case of an incorrect prediction of the memory area, the read access must be restarted.

It is recommended that the GPRs used for indirect addressing always point to the same memory area. If an updated GPR points to a different memory area, the next read operation will access the wrong memory area. The read access must be repeated, which leads to pipeline stalls.

Conflict\_GPRs\_Pointer\_WrongHistory:

```

In      ADD R3, [R0]          ;R0 points to DPRAM (e.g.)
In+1    MOV R0, R4
...
Ii      MOV DPPX, ...        ;change DPPx
...
Im      ADD R6, [R0]          ;R0 now points to SRAM (e.g.)
Im+1    MOV R6, R1
  
```

**Table 5-5 Pipeline Dependencies with Pointers (Valid Speculation)**

Stage	T <sub>n</sub>	T <sub>n+1</sub>	T <sub>n+2</sub>	T <sub>n+3</sub>	T <sub>n+4</sub>	T <sub>n+5</sub>
<b>DECODE</b>	I <sub>n</sub> = ADD R3, [R0]	I <sub>n+1</sub> = MOV R0, R4	I <sub>n+2</sub>	I <sub>n+3</sub>	I <sub>n+4</sub>	I <sub>n+5</sub>
<b>ADDRESS</b>	I <sub>n-1</sub>	I <sub>n</sub> = ADD R3, [R0]	I <sub>n+1</sub> = MOV R0, R4	I <sub>n+2</sub>	I <sub>n+3</sub>	I <sub>n+4</sub>
<b>MEMORY</b>	I <sub>n-2</sub>	I <sub>n-1</sub>	I <sub>n</sub> = ADD R3, [R0]	I <sub>n+1</sub> = MOV R0, R4	I <sub>n+2</sub>	I <sub>n+3</sub>
<b>EXECUTE</b>	I <sub>n-3</sub>	I <sub>n-2</sub>	I <sub>n-1</sub>	I <sub>n</sub> = ADD R3, [R0]	I <sub>n+1</sub> = MOV R0, R4	I <sub>n+2</sub>
<b>WR.BACK</b>	I <sub>n-4</sub>	I <sub>n-3</sub>	I <sub>n-2</sub>	I <sub>n-1</sub>	I <sub>n</sub> = ADD R3, [R0]	I <sub>n+1</sub> = MOV R0, R4



**Table 5-6 Pipeline Dependencies with Pointers (Invalid Speculation)**

Stage	$T_m$	$T_{m+1}$	$T_{m+2}$ <sup>1)</sup>	$T_{m+3}$	$T_{m+4}$	$T_{m+5}$
<b>DECODE</b>	$I_m = \text{ADD R6, [R0]}$	$I_{m+1} = \text{MOV R6, R1}$	$I_{m+1} = \text{MOV R6, R1}$	$I_{m+2}$	$I_{m+3}$	$I_{m+4}$
<b>ADDRESS</b>	$I_{m-1}$	$I_m = \text{ADD R6, [R0]}$	$I_m = \text{ADD R6, [R0]}$	$I_{m+1} = \text{MOV R6, R1}$	$I_{m+2}$	$I_{m+3}$
<b>MEMORY</b>	$I_{m-2}$	$I_{m-1}$	–	$I_m = \text{ADD R6, [R0]}$	$I_{m+1} = \text{MOV R6, R1}$	$I_{m+2}$
<b>EXECUTE</b>	$I_{m-3}$	$I_{m-2}$	$I_{m-1}$	–	$I_m = \text{ADD R6, [R0]}$	$I_{m+1} = \text{MOV R6, R1}$
<b>WR.BACK</b>	$I_{m-4}$	$I_{m-3}$	$I_{m-2}$	$I_{m-1}$	–	$I_m = \text{ADD R6, [R0]}$

1) Access to location [R0] must be repeated due to wrong history (target area was changed).

### 5.3.2.3 Due to Memory Bandwidth

Memory bandwidth conflicts can occur if instructions in the pipeline access the same memory area at the same time. Special access mechanisms are implemented to minimize conflicts. The DPRAM of the CPU has two independent read/write ports; this allows parallel read and write operation without delays. Write accesses to the DSRAM can be buffered in a Write Back Buffer until read accesses are finished.

All instructions except the CoXXX instructions can read only one memory operand per cycle. A conflict between the read and one write access cannot occur because the DPRAM has two independent read/write ports. Only other pipeline stall conditions can generate a DPRAM bandwidth conflict. The DPRAM is a synchronous pipelined memory. The read access starts with the valid addresses on the address stage. The data are delivered in the Memory stage. If a memory read access is stalled in the Memory stage and the following instruction on the Address stage tries to start a memory read, the new read access must be delayed as well. But, this conflict is hidden by an already existing stall of the pipeline.

The CoXXX instructions are the only instructions able to read two memory operands per cycle. A conflict between the two read and one pending write access can occur if all three operands are located in the DPRAM area. This is especially important for performance in the case of executing a filter routine. One of the operands should be located in the DSRAM to guarantee a single-cycle execution of the CoXXX instructions.

Conflict\_DPRAM\_Bandwidth:

```

In      ADD op1, R1
In+1    ADD R6, R0
In+2    CoMAC [IDX0], [R0]
```

$I_{n+3}$     `MOV R3, [R0]`

**Table 5-7      Pipeline Dependencies in Case of Memory Conflicts (DPRAM)**

Stage	$T_n$	$T_{n+1}$	$T_{n+2}$	$T_{n+3}$	$T_{n+4}$ <sup>1)</sup>	$T_{n+5}$
<b>DECODE</b>	$I_n = \text{ADD op1, R1}$	$I_{n+1} = \text{ADD R6, R0}$	$I_{n+2} = \text{CoMAC ...}$	$I_{n+3} = \text{MOV R3, [R0]}$	$I_{n+4}$	$I_{n+4}$
<b>ADDRESS</b>	$I_{n-1}$	$I_n = \text{ADD op1, R1}$	$I_{n+1} = \text{ADD R6, R0}$	$I_{n+2} = \text{CoMAC ...}$	$I_{n+3} = \text{MOV R3, [R0]}$	$I_{n+3} = \text{MOV R3, [R0]}$
<b>MEMORY</b>	$I_{n-2}$	$I_{n-1}$	$I_n = \text{ADD op1, R1}$	$I_{n+1} = \text{ADD R6, R0}$	$I_{n+2} = \text{CoMAC ...}$	$I_{n+2} = \text{CoMAC ...}$
<b>EXECUTE</b>	$I_{n-3}$	$I_{n-2}$	$I_{n-1}$	$I_n = \text{ADD op1, R1}$	$I_{n+1} = \text{ADD R6, R0}$	–
<b>WR.BACK</b>	$I_{n-4}$	$I_{n-3}$	$I_{n-2}$	$I_{n-1}$	$I_n = \text{ADD op1, R1}$	$I_{n+1} = \text{ADD R6, R0}$

1) COMAC instruction stalls due to memory bandwidth conflict.

The DSRAM is a single-port memory with one read/write port. To reduce the number of bandwidth conflict cases, a Write Back Buffer is implemented. It has three data entries. Only if the buffer is filled and a read access and a write access occur at the same time, must the read access be stalled while one of the buffer entries is written back.

Conflict\_DSRAM\_Bandwidth:

$I_n$       `ADD op1, R1`  
 $I_{n+1}$    `ADD R6, R0`  
 $I_{n+2}$    `ADD R6, op2`  
 $I_{n+3}$    `MOV R3, R2`

**Table 5-8      Pipeline Dependencies in Case of Memory Conflicts (DSRAM)**

Stage	$T_n$	$T_{n+1}$	$T_{n+2}$	$T_{n+3}$	$T_{n+4}$ <sup>1)</sup>	$T_{n+5}$
<b>DECODE</b>	$I_n = \text{ADD op1, R1}$	$I_{n+1} = \text{ADD R6, R0}$	$I_{n+2} = \text{ADD R6, op2}$	$I_{n+3} = \text{MOV R3, R2}$	$I_{n+4}$	$I_{n+4}$
<b>ADDRESS</b>	$I_{n-1}$	$I_n = \text{ADD op1, R1}$	$I_{n+1} = \text{ADD R6, R0}$	$I_{n+2} = \text{ADD R6, op2}$	$I_{n+3} = \text{MOV R3, R2}$	$I_{n+3} = \text{MOV R3, R2}$
<b>MEMORY</b>	$I_{n-2}$	$I_{n-1}$	$I_n = \text{ADD op1, R1}$	$I_{n+1} = \text{ADD R6, R0}$	$I_{n+2} = \text{ADD R6, op2}$	$I_{n+2} = \text{ADD R6, op2}$
<b>EXECUTE</b>	$I_{n-3}$	$I_{n-2}$	$I_{n-1}$	$I_n = \text{ADD op1, R1}$	$I_{n+1} = \text{ADD R6, R0}$	–

**Table 5-8 Pipeline Dependencies in Case of Memory Conflicts (DSRAM)**

Stage	$T_n$	$T_{n+1}$	$T_{n+2}$	$T_{n+3}$	$T_{n+4}$ <sup>1)</sup>	$T_{n+5}$
<b>WR.BACK</b>	$I_{n-4}$	$I_{n-3}$	$I_{n-2}$	$I_{n-1}$	$I_n = \text{ADD op1, R1}$	$I_{n+1} = \text{ADD R6, R0}$
<b>WB.Buffer</b>	full	full	full	full	full	full

1) ADD R6, op2 instruction stalls due to memory bandwidth conflict.

### 5.3.2.4 Caused by CPU-SFR Updates

CPU-SFRs control the CPU functionality and behavior. Changes and updates of CSFRs influence the instruction flow in the pipeline. Therefore, special care is required to ensure that instructions in the pipeline always work with the correct CSFR values. CSFRs are updated late on the EXECUTE stage of the pipeline. Meanwhile, without conflict detection, the instructions in the DECODE, ADDRESS, and MEMORY stages would still work without updated register values. The CPU detects conflict cases and stalls the pipeline to guarantee a correct execution. For performance reasons, the CPU differentiates between different classes of CPU-SFRs. The flow of instructions through the pipeline can be improved by following the given rules used for instruction re-ordering.

There are three classes of CPU-SFRs:

- CSFRs not generating pipeline conflicts (ONES, ZEROS, MCW)
- CSFR result registers updated late in the EXECUTE stage, causing one stall cycle
- CSFRs affecting the whole CPU or the pipeline, causing a pipeline cancellation

### CSFR Result Registers

The CSFR result registers MDH, MDL, MSW, MAH, MAL, and MRW of the ALU and MAC-Unit are updated late in the EXECUTE stage of the pipeline. If an instruction (except CoSTORE) accesses these registers in the MEMORY stage, the value cannot be forwarded. The instruction must be stalled for one cycle on the MEMORY stage.

Conflict\_CSFR\_Update\_Stall:

```

 $I_n$       MUL  R0, R1
 $I_{n+1}$     MOV  R6, MDL
 $I_{n+2}$     ADD  R6, R1
 $I_{n+3}$     MOV  R3, [R0]
```

**Table 5-9 Pipeline Dependencies with Result CSFRs (Stall)**

Stage	$T_n$	$T_{n+1}$	$T_{n+2}$	$T_{n+3}^{1)}$	$T_{n+4}$	$T_{n+5}$
<b>DECODE</b>	$I_n = \text{MUL } R0, R1$	$I_{n+1} = \text{MOV } R6, MDL$	$I_{n+2} = \text{ADD } R6, R1$	$I_{n+3} = \text{MOV } R3, [R0]$	$I_{n+3} = \text{MOV } R3, [R0]$	$I_{n+4}$
<b>ADDRESS</b>	$I_{n-1}$	$I_n = \text{MUL } R0, R1$	$I_{n+1} = \text{MOV } R6, MDL$	$I_{n+2} = \text{ADD } R6, R1$	$I_{n+2} = \text{ADD } R6, R1$	$I_{n+3} = \text{MOV } R3, [R0]$
<b>MEMORY</b>	$I_{n-2}$	$I_{n-1}$	$I_n = \text{MUL } R0, R1$	$I_{n+1} = \text{MOV } R6, MDL$	$I_{n+1} = \text{MOV } R6, MDL$	$I_{n+2} = \text{ADD } R6, R1$
<b>EXECUTE</b>	$I_{n-3}$	$I_{n-2}$	$I_{n-1}$	$I_n = \text{MUL } R0, R1$	–	$I_{n+1} = \text{MOV } R6, MDL$
<b>WR.BACK</b>	$I_{n-4}$	$I_{n-3}$	$I_{n-2}$	$I_{n-1}$	$I_n = \text{MUL } R0, R1$	–

1) Cannot read MDL here.

By reordering instructions, the bubble in the pipeline can be filled with an instruction not using this resource.

Conflict\_CSFR\_Update\_Resolved:

```

 $I_n$       MUL  R0, R1
 $I_{n+1}$     MOV  R3, [R0]
 $I_{n+2}$     MOV  R6, MDL
 $I_{n+3}$     ADD  R6, R1

```

**Table 5-10 Pipeline Dependencies with Result CSFRs (No Stall)**

Stage	$T_n$	$T_{n+1}$	$T_{n+2}$	$T_{n+3}$	$T_{n+4}^{1)}$	$T_{n+5}$
<b>DECODE</b>	$I_n = \text{MUL } R0, R1$	$I_{n+1} = \text{MOV } R3, [R0]$	$I_{n+2} = \text{MOV } R6, MDL$	$I_{n+3} = \text{ADD } R6, R1$	$I_{n+4}$	$I_{n+5}$
<b>ADDRESS</b>	$I_{n-1}$	$I_n = \text{MUL } R0, R1$	$I_{n+1} = \text{MOV } R3, [R0]$	$I_{n+2} = \text{MOV } R6, MDL$	$I_{n+3} = \text{ADD } R6, R1$	$I_{n+4}$
<b>MEMORY</b>	$I_{n-2}$	$I_{n-1}$	$I_n = \text{MUL } R0, R1$	$I_{n+1} = \text{MOV } R3, [R0]$	$I_{n+2} = \text{MOV } R6, MDL$	$I_{n+3} = \text{ADD } R6, R1$
<b>EXECUTE</b>	$I_{n-3}$	$I_{n-2}$	$I_{n-1}$	$I_n = \text{MUL } R0, R1$	$I_{n+1} = \text{MOV } R3, [R0]$	$I_{n+2} = \text{MOV } R6, MDL$
<b>WR.BACK</b>	$I_{n-4}$	$I_{n-3}$	$I_{n-2}$	$I_{n-1}$	$I_n = \text{MUL } R0, R1$	$I_{n+1} = \text{MOV } R3, [R0]$

1) MDL can be read now, no stall cycle necessary.

## CSFRs Affecting the Whole CPU

Some CSFRs affect the whole CPU or the pipeline before the Memory stage. The CPU-SFRs CPUCON1/2, CP, SP, STKUN, STKOV, VECSEG, TFR, and PSW affect the overall CPU function, while the CPU-SFRs IDX0, IDX1, QX1, QX0, DPP0, DPP1, DPP2, and DPP3 only affect the DECODE, ADDRESS, and MEMORY stage when they are modified **explicitly**. In this case the pipeline behavior depends on the instruction and addressing mode used to modify the CSFR.

In the case of modification of these CSFRs by “POP CSFR” or by instructions using the reg,#data16 addressing mode, a special mechanism is implemented to improve performance during the initialization.

For further explanation, the instruction which modifies the CSFR can be called “instruction\_modify\_CSFR”. This special case is detected in the DECODE stage when the instruction\_modify\_CSFR enters the processing pipeline. Further on, instructions described in the following list are held in the DECODE stage (all other instructions are not held):

- Instructions using long addressing mode (mem)
- Instructions using indirect addressing modes ( $[R_w]$ ,  $[R_w+]$ ...), except JMPL and CALLI
- ENWDT, DISWDT, EINIT
- All CoXXX instructions

If the CPUCON1/2, CP, SP, STKUN, STKOV, VECSEG, TFR, or the PSW are modified and the instruction\_modify\_CSFR reaches the EXECUTE stage, the pipeline is canceled. The modification affects the entire pipeline and the instruction prefetch. A clean cancel and restart mechanism is required to guarantee a correct instruction flow. In case of modification of IDX0, IDX1, QX1, QX0, DPP0, DPP1, DPP2, or DPP3 only the DECODE, ADDRESS, and MEMORY stages are affected and the pipeline needs not to be canceled. The modification does not affect the instructions in the ADDRESS, MEMORY stage because they are not using this resource. Other kinds of instructions are held in the DECODE stage until the CSFR is modified.

The following example shows a case in which the pipeline is stalled. The instruction “MOV R6, R1” after the “MOV IDX1, #12” instruction which modifies the CSFR will be held in DECODE Stage until the IDX1 register is updated. The next example shows an optimized initialization routine.

Conflict\_Canceling:

```
In      MOV  IDX1, #12
In+1    MOV  R6, mem
In+2    ADD  R6, R1
In+3    MOV  R3, [R0]
```

**Table 5-11 Pipeline Dependencies with Control CSFRs (Canceling)**

Stage	$T_n$	$T_{n+1}$	$T_{n+2}$	$T_{n+3}$	$T_{n+4}$	$T_{n+5}$
<b>DECODE</b>	$I_n = \text{MOV } \text{IDX1}, \#12$	$I_{n+1} = \text{MOV } \text{R6}, \text{mem}$	$I_{n+1} = \text{MOV } \text{R6}, \text{mem}$	$I_{n+1} = \text{MOV } \text{R6}, \text{mem}$	$I_{n+1} = \text{MOV } \text{R6}, \text{mem}$	$I_{n+2} = \text{ADD } \text{R6}, \text{R1}$
<b>ADDRESS</b>	$I_{n-1}$	$I_n = \text{MOV } \text{IDX1}, \#12$	—	—	—	$I_{n+1} = \text{MOV } \text{R6}, \text{mem}$
<b>MEMORY</b>	$I_{n-2}$	$I_{n-1}$	$I_n = \text{MOV } \text{IDX1}, \#12$	—	—	—
<b>EXECUTE</b>	$I_{n-3}$	$I_{n-2}$	$I_{n-1}$	$I_n = \text{MOV } \text{IDX1}, \#12$	—	—
<b>WR.BACK</b>	$I_{n-4}$	$I_{n-3}$	$I_{n-2}$	$I_{n-1}$	$I_n = \text{MOV } \text{IDX1}, \#12$	—

Conflict\_Canceling\_Optimized:

```

 $I_n$       MOV  IDX1, #12
 $I_{n+1}$     MOV  MAH, #23
 $I_{n+2}$     MOV  MAL, #25
 $I_{n+3}$     MOV  R3, #08

```

**Table 5-12 Pipeline Dependencies with Control CSFRs (Optimized)**

Stage	$T_n$	$T_{n+1}$	$T_{n+2}$	$T_{n+3}$	$T_{n+4}$	$T_{n+5}$
<b>DECODE</b>	$I_n = \text{MOV } \text{IDX1}, \#12$	$I_{n+1} = \text{MOV } \text{MAH}, \#23$	$I_{n+2} = \text{MOV } \text{MAL}, \#25$	$I_{n+3} = \text{MOV } \text{R3}, \#08$	$I_{n+4}$	$I_{n+5}$
<b>ADDRESS</b>	$I_{n-1}$	$I_n = \text{MOV } \text{IDX1}, \#12$	$I_{n+1} = \text{MOV } \text{MAH}, \#23$	$I_{n+2} = \text{MOV } \text{MAL}, \#25$	$I_{n+3} = \text{MOV } \text{R3}, \#08$	$I_{n+4}$
<b>MEMORY</b>	$I_{n-2}$	$I_{n-1}$	$I_n = \text{MOV } \text{IDX1}, \#12$	$I_{n+1} = \text{MOV } \text{MAH}, \#23$	$I_{n+2} = \text{MOV } \text{MAL}, \#25$	$I_{n+3} = \text{MOV } \text{R3}, \#08$
<b>EXECUTE</b>	$I_{n-3}$	$I_{n-2}$	$I_{n-1}$	$I_n = \text{MOV } \text{IDX1}, \#12$	$I_{n+1} = \text{MOV } \text{MAH}, \#23$	$I_{n+2} = \text{MOV } \text{MAL}, \#25$
<b>WR.BACK</b>	$I_{n-4}$	$I_{n-3}$	$I_{n-2}$	$I_{n-1}$	$I_n = \text{MOV } \text{IDX1}, \#12$	$I_{n+1} = \text{MOV } \text{MAH}, \#23$

For all the other instructions that modify this kind of CSFR, a simple stall and cancel mechanism guarantees the correct instruction flow.

A possible explicit write-operation to this kind of CSFRs is detected on the MEMORY stage of the pipeline. The following instructions on the ADDRESS and DECODE Stage are stalled. If the instruction reaches the EXECUTE stage, the entire pipeline and the Instruction FIFO of the IFU are canceled. The instruction flow is completely re-started.

Conflict\_Canceling\_Completely:

```

In      MOV PSW, R4
In+1    MOV R6, R1
In+2    ADD R6, R1
In+3    MOV R3, [R0]

```

**Table 5-13 Pipeline Dependencies with Control CSFRs (Cancel All)**

Stage	T <sub>n+1</sub>	T <sub>n+2</sub>	T <sub>n+3</sub>	T <sub>n+4</sub>	T <sub>n+5</sub>	T <sub>n+6</sub>
<b>DECODE</b>	I <sub>n+1</sub> = MOV R6, R1	I <sub>n+2</sub> = ADD R6, R1	I <sub>n+2</sub> = ADD R6, R1	—	—	I <sub>n+1</sub> = MOV R6, R1
<b>ADDRESS</b>	I <sub>n</sub> = MOV PSW, R4	I <sub>n+1</sub> = MOV R6, R1	I <sub>n+1</sub> = MOV R6, R1	—	—	—
<b>MEMORY</b>	I <sub>n-1</sub>	I <sub>n</sub> = MOV PSW, R4	—	—	—	—
<b>EXECUTE</b>	I <sub>n-2</sub>	I <sub>n-1</sub>	I <sub>n</sub> = MOV PSW, R4	—	—	—
<b>WR.BACK</b>	I <sub>n-3</sub>	I <sub>n-2</sub>	I <sub>n-1</sub>	I <sub>n</sub> = MOV PSW, R4	—	—

## 5.4 CPU Configuration Registers

The CPU configuration registers select a number of general features and behaviors of the XE16xyM's CPU core. In general these registers are only written by the startup software and not altered during application software run time.

*Note: The CPU configuration registers are protected by the register security mechanism after the EINIT instruction has been executed.*

### CPUCON1

#### CPU Control Register 1

**SFR (FE18<sub>H</sub>/0C<sub>H</sub>)**

**Reset Value: 0007<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0										VECSC	WDT CTL	SGT DIS	INTS CXT	BP	ZCJ
r	r	r	r	r	r	r	r	r	r	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
<b>0</b>	[15:7]	r	<b>Reserved</b> Read as 0, should be written 0
<b>VECSC</b>	[6:5]	rw	<b>Scaling Factor of Vector Table</b> 00 <sub>B</sub> Space between two vectors is 2 words <sup>1)</sup> 01 <sub>B</sub> Space between two vectors is 4 words 10 <sub>B</sub> Space between two vectors is 8 words 11 <sub>B</sub> Space between two vectors is 16 words
<b>WDTCTL</b>	4	rw	<b>Configuration of Watchdog Timer</b> 0 <sub>B</sub> DISWDT executable only until End Of Init <sup>2)</sup> 1 <sub>B</sub> DISWDT/ENWDT always executable (enhanced WDT mode)
<b>SGTDIS</b>	3	rw	<b>Segmentation Disable/Enable Control</b> 0 <sub>B</sub> Segmentation enabled 1 <sub>B</sub> Segmentation disabled
<b>INTSCXT</b>	2	rw	<b>Enable Interruptibility of Switch Context</b> 0 <sub>B</sub> Switch context is not interruptible 1 <sub>B</sub> Switch context is interruptible
<b>BP</b>	1	rw	<b>Enable Branch Prediction Unit</b> 0 <sub>B</sub> Branch prediction disabled 1 <sub>B</sub> Branch prediction enabled
<b>ZCJ</b>	0	rw	<b>Enable Zero-Cycle Jump Function</b> 0 <sub>B</sub> Zero-cycle jump function disabled 1 <sub>B</sub> Zero-cycle jump function enabled



**Central Processing Unit (CPU)**

- 1) The default value (2 words) is compatible with the vector distance defined in the C166 Family architecture.
- 2) The DISWDT (executed after EINIT) and ENWDT instructions are internally converted in a NOP instruction.

**CPUCON2**

**CPU Control Register 2**

**SFR (FE1A<sub>H</sub>/0D<sub>H</sub>)**

**Reset Value: 8FBB<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>FIFODEPTH</b>				<b>FIFO FED</b>		<b>BYP PF</b>	<b>BYP F</b>	<b>1</b>	<b>STE N</b>	<b>LFIC</b>	<b>OV RUN</b>	<b>RET ST</b>	<b>FAS TBL</b>	<b>1</b>	<b>SL</b>
rw				rw		rw	rw	r	rw	rw	rw	rw	rw	r	rw

Field	Bits	Type	Description
<b>FIFODEPTH</b>	[15:12]	rw	<b>FIFO Depth Configuration</b> 0 <sub>H</sub> No FIFO (entries) 1 <sub>H</sub> One FIFO entry ... 8 <sub>H</sub> Eight FIFO entries 9 <sub>H</sub> reserved ... F <sub>H</sub> reserved
<b>FIFO FED</b>	[11:10]	rw	<b>FIFO Fed Configuration</b> 00 <sub>B</sub> FIFO disabled 01 <sub>B</sub> FIFO filled with up to one instruction per cycle 10 <sub>B</sub> FIFO filled with up to two instructions per cycle 11 <sub>B</sub> FIFO filled with up to three instruction per cycle
<b>BYP PF</b>	9	rw	<b>Prefetch Bypass Control</b> 0 <sub>B</sub> Bypass path from prefetch to decode disabled 1 <sub>B</sub> Bypass path from prefetch to decode available
<b>BYP F</b>	8	rw	<b>Fetch Bypass Control</b> 0 <sub>B</sub> Bypass path from fetch to decode disabled 1 <sub>B</sub> Bypass path from fetch to decode available
<b>1</b>	7	r	<b>Reserved</b> Read as 1, should be written 1
<b>STEN</b>	6	rw	<b>Stall Instruction Enable</b> (for debug purposes) 0 <sub>B</sub> Stall Instruction disabled 1 <sub>B</sub> Stall Instruction enabled (see example below)
<b>LFIC</b>	5	rw	<b>Linear Follower Instruction Cache</b> 0 <sub>B</sub> Linear Follower Instruction Cache disabled 1 <sub>B</sub> Linear Follower Instruction Cache enabled

**Central Processing Unit (CPU)**

Field	Bits	Type	Description
<b>OVRUN</b>	4	rw	<b>Pipeline Control</b> 0 <sub>B</sub> Overrun of pipeline bubbles not allowed 1 <sub>B</sub> Overrun of pipeline bubbles allowed
<b>RETST</b>	3	rw	<b>Enable Return Stack</b> 0 <sub>B</sub> Return Stack is disabled 1 <sub>B</sub> Return Stack is enabled
<b>FASTBL</b>	2	rw	<b>Enables the fast injection of block transfers</b> 0 <sub>B</sub> Direct injection disabled 1 <sub>B</sub> Direct injection enabled
<b>1</b>	1	r	<b>Reserved</b> Read as 1, should be written 1
<b>SL</b>	0	rw	<b>Enables Short Loop Mode</b> 0 <sub>B</sub> Short loop mode disabled 1 <sub>B</sub> Short loop mode enabled

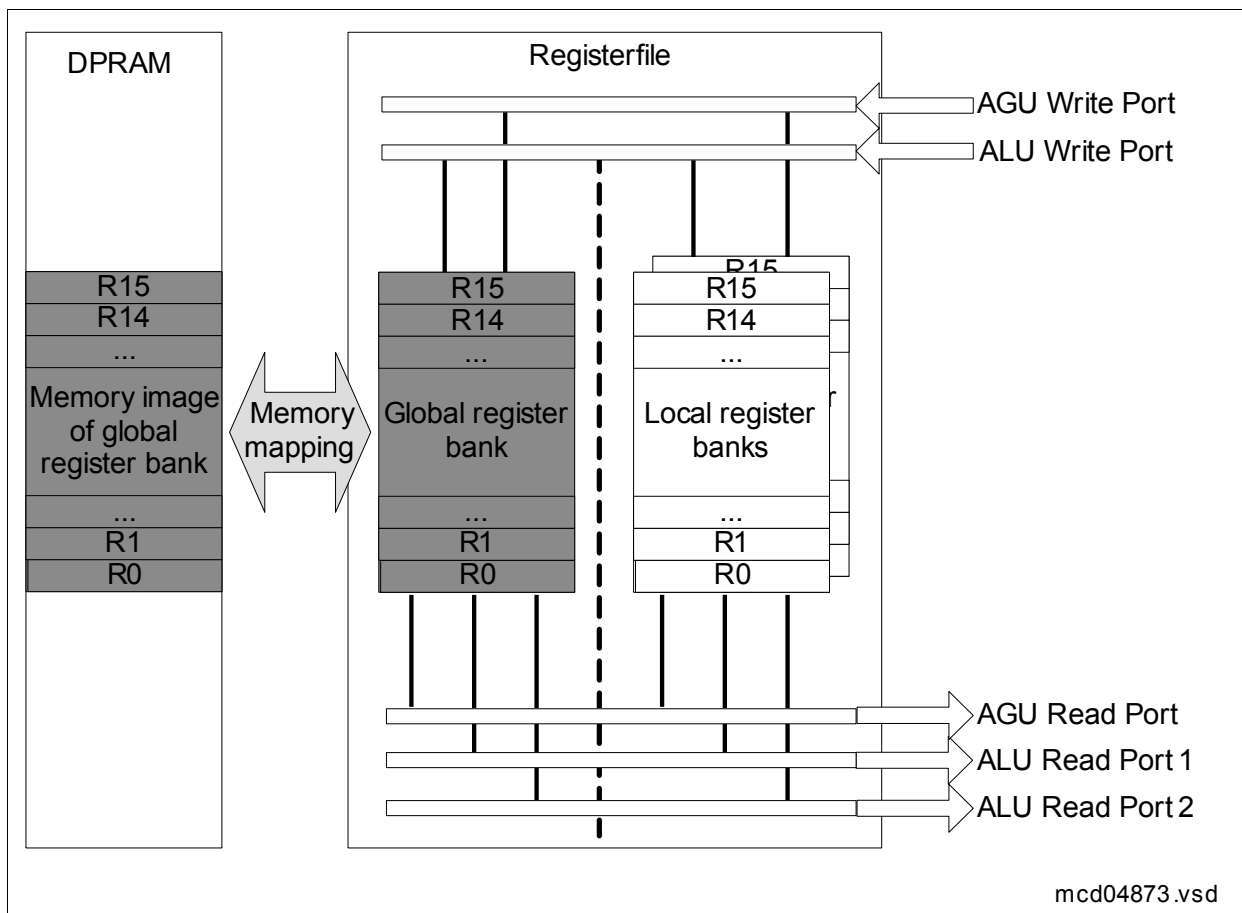
*Note: This register must only be modified when explicitly documented - e.g. in an errata sheet.*

## 5.5 Use of General Purpose Registers

The CPU provides three banks of sixteen dedicated registers R0, R1, R2, ... R15, called General Purpose Registers (GPRs), which can be accessed in one CPU cycle. The GPRs are the working registers of the arithmetic and logic units and many also serve as address pointers for indirect addressing modes.

The register banks are accessed via the 5-port register file providing the high access speed required for the CPU's performance. The register file is split into three independent physical register banks. There are **two types of register banks**:

- **Two local register banks** which are a part of the register file
- **One global register bank** which is memory-mapped and cached in the register file

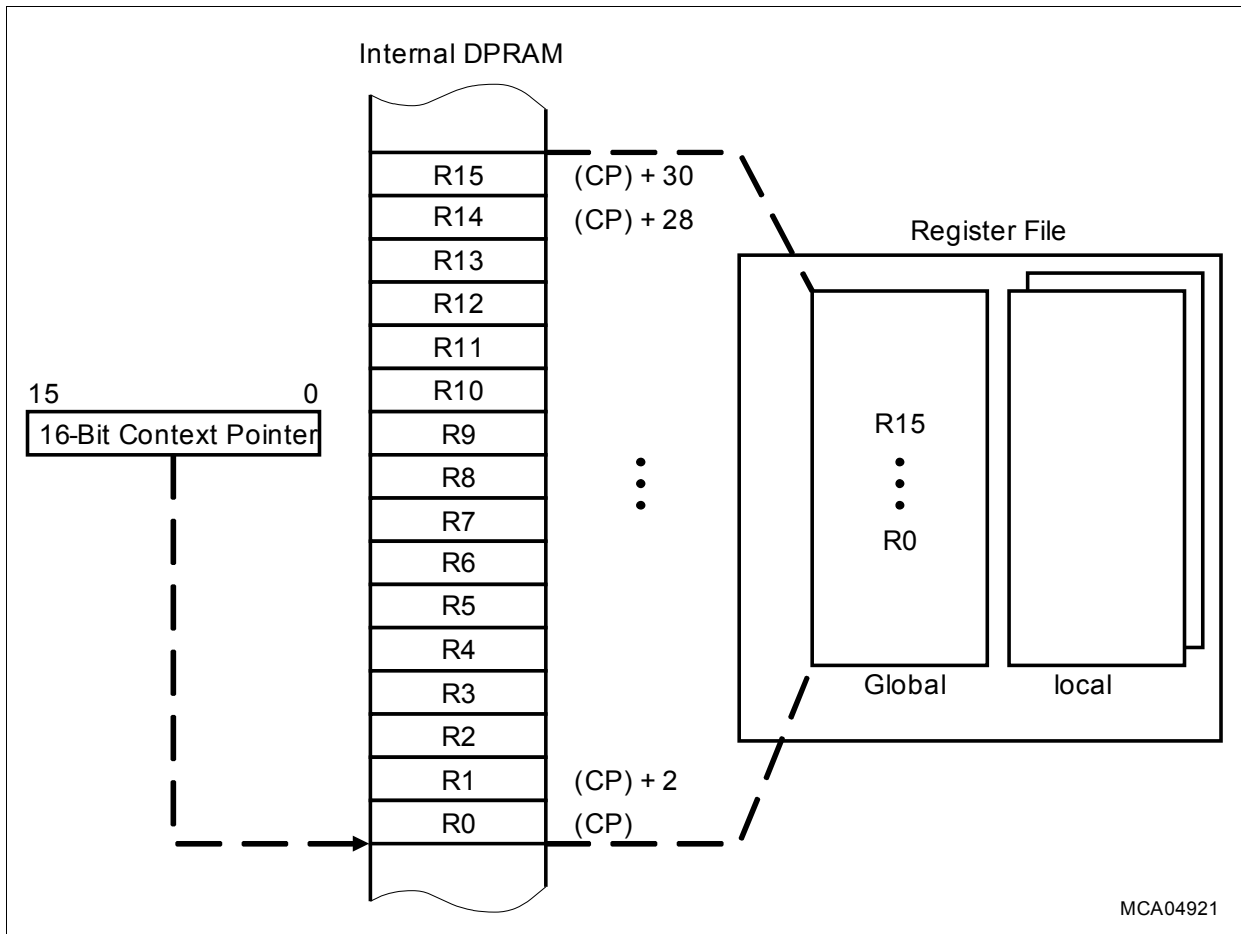


**Figure 5-3 Register File**

Bitfield BANK in register PSW selects which of the three physical register banks is activated. The selected bank can be changed explicitly by any instruction which writes to the PSW, or implicitly by a RETI instruction, an interrupt or hardware trap. In case of an interrupt, the selection of the register bank is configured via registers BNKSELx in the Interrupt Controller ITC. Hardware traps always use the global register bank.

**Central Processing Unit (CPU)**

The local register banks are built of dedicated physical registers, while the global register bank represents a cache. Multiple global banks can be mapped to the internal DPRAM. Each of these banks uses a block of 16 consecutive words. A Context Pointer (CP) register determines the base address of the current selected bank. To provide the required access speed, the GPRs located in the DPRAM are cached in the 5-port register file (only one memory-mapped GPR bank can be cached at the time). If the global register bank is activated, the cache will be validated before further instructions are executed. After validation, all further accesses to the GPRs are redirected to the global register bank.



**Figure 5-4 Register Bank Selection via Register CP**

### 5.5.1 GPR Addressing Modes

Because the GPRs are the working registers and are accessed frequently, there are three possible ways to access a register bank:

- **Short GPR Address** (mnemonic: Rw or Rb)
- **Short Register Address** (mnemonic: reg or bitoff)
- **Long Memory Address** (mnemonic: mem), for the global bank only

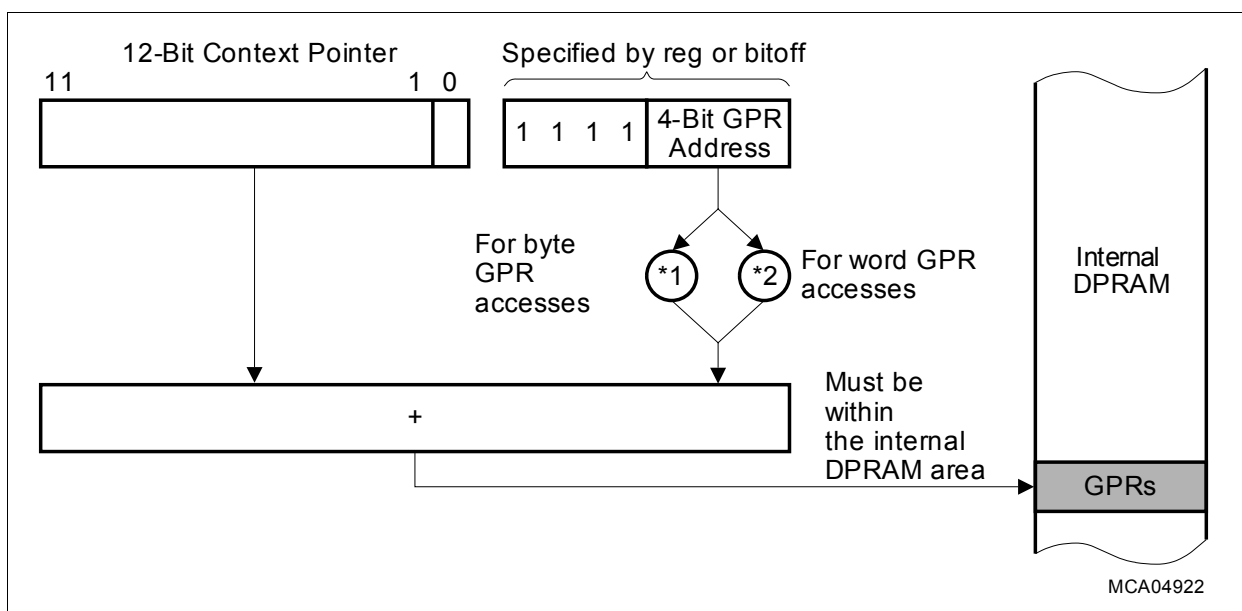
**Short GPR Addresses** specify the register offset within the current register bank (selected via bitfield BANK). Short 4-bit GPR addresses can access all sixteen registers, short 2-bit addresses (used by some instructions) can access the lower four registers.

Depending on whether a register word (Rw) or byte (Rb) address is specified, the short GPR address is either multiplied by two (Rw) or not (Rb) before it is used to physically access the register bank. Thus, both byte and word GPR accesses are possible in this way.

*Note: GPRs used as indirect address pointers are always accessed wordwise.*

For the local register banks the resulting offset is used directly, for the global register bank the resulting offset is logically added to the contents of register CP which points to the memory location of the base of the current global register bank (see [Figure 5-5](#)).

**Short 8-Bit Register Addresses** within a range from F0<sub>H</sub> to FF<sub>H</sub> interpret the four least significant bits as short 4-bit GPR addresses, while the four most significant bits are ignored. The respective physical GPR address is calculated in the same way as for short 4-bit GPR addresses. For single bit GPR accesses, the GPR's word address is calculated in the same way. The accessed bit position within the word is specified by a separate additional 4-bit value.



**Figure 5-5 Implicit CP Use by Logical Short GPR Addressing Modes**

**Central Processing Unit (CPU)**

**24-Bit Memory Addresses** can be directly used to access GPRs located in the DPRAM (not applicable for local register banks). In case of a memory read access, a hit detection logic checks if the accessed memory location is cached in the global register bank. In case of a cache hit the read is redirected to the global register bank. The data that is read from cache will be used and the read from memory will be discarded. This leads to a delay of one CPU cycle (MOV R4, **mem** [ $CP \leq mem \leq CP + 31$ ]). In case of a memory write access, the hit detection logic determines a cache hit in advance. Nevertheless, the address conversion needs one additional CPU cycle. The value is directly written into the global register bank without further delay (MOV **mem**, R4).

*Note: The 24-bit GPR addressing mode requires an extra cycle for the read and write access.*

**Table 5-14 Addressing Modes to Access GPRs**

Word Registers <sup>1)</sup>		Byte Registers		Short Address <sup>2)</sup>		
Name	Mem. Addr. <sup>3)</sup>	Name	Mem. Addr. <sup>3)</sup>	8-Bit	4-Bit	2-Bit
R0	(CP) + 0	RL0	(CP) + 0	F0 <sub>H</sub>	0 <sub>H</sub>	0 <sub>H</sub>
R1	(CP) + 2	RH0	(CP) + 1	F1 <sub>H</sub>	1 <sub>H</sub>	1 <sub>H</sub>
R2	(CP) + 4	RL1	(CP) + 2	F2 <sub>H</sub>	2 <sub>H</sub>	2 <sub>H</sub>
R3	(CP) + 6	RH1	(CP) + 3	F3 <sub>H</sub>	3 <sub>H</sub>	3 <sub>H</sub>
R4	(CP) + 8	RL2	(CP) + 4	F4 <sub>H</sub>	4 <sub>H</sub>	---
R5	(CP) + 10	RH2	(CP) + 5	F5 <sub>H</sub>	5 <sub>H</sub>	---
R6	(CP) + 12	RL3	(CP) + 6	F6 <sub>H</sub>	6 <sub>H</sub>	---
R7	(CP) + 14	RH3	(CP) + 7	F7 <sub>H</sub>	7 <sub>H</sub>	---
R8	(CP) + 16	RL4	(CP) + 8	F8 <sub>H</sub>	8 <sub>H</sub>	---
R9	(CP) + 18	RH4	(CP) + 9	F9 <sub>H</sub>	9 <sub>H</sub>	---
R10	(CP) + 20	RL5	(CP) + 10	FA <sub>H</sub>	A <sub>H</sub>	---
R11	(CP) + 22	RH5	(CP) + 11	FB <sub>H</sub>	B <sub>H</sub>	---
R12	(CP) + 24	RL6	(CP) + 12	FC <sub>H</sub>	C <sub>H</sub>	---
R13	(CP) + 26	RH6	(CP) + 13	FD <sub>H</sub>	D <sub>H</sub>	---
R14	(CP) + 28	RL7	(CP) + 14	FE <sub>H</sub>	E <sub>H</sub>	---
R15	(CP) + 30	RH7	(CP) + 15	FF <sub>H</sub>	F <sub>H</sub>	---

1) The first 8 GPRs (R7 ... R0) may also be accessed byte-wise. Writing to a GPR byte does not affect the other byte of the respective GPR.

2) Short addressing modes are usable for all register banks.

3) Long addressing mode only usable for the memory mapped global bank.

## 5.5.2 Context Switching

When a task scheduler of an operating system activates a new task or an interrupt service routine is called or terminated, the working context (i.e. the registers) of the left task must be saved and the working context of the new task must be restored. The CPU context can be changed in two ways:

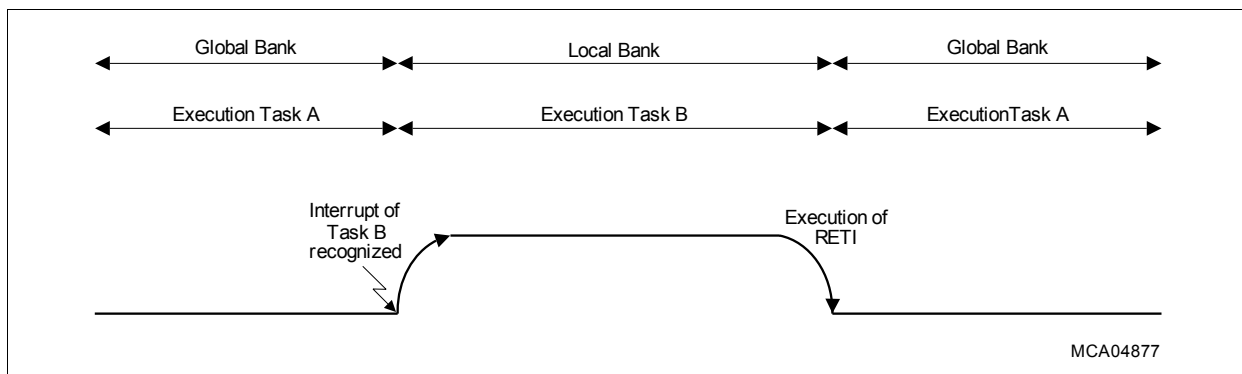
- Switching the selected register bank
- Switching the context of the global register bank

### Switching the Selected Physical Register Bank

By updating bitfield BANK in register PSW the active register bank is switched immediately. It is possible to switch between the current memory-mapped GPR bank cached in the global register bank (BANK = 00<sub>B</sub>), local register bank 1 (BANK = 10<sub>B</sub>), and local register bank 2 (BANK = 11<sub>B</sub>).

In case of an interrupt service, the bank switch can be automatically executed by updating bitfield BANK from registers BNKSELx in the interrupt controller. By executing a RETI instruction, bitfield BANK will automatically be restored and the context will be switched to the original register bank.

The switch between the three physical register banks of the register file can also be executed by writing to bitfield BANK. Because of pipeline dependencies an explicit change of register PSW must cancel the pipeline.



**Figure 5-6 Context Switch by Changing the Physical Register Bank**

After a switch to a local register bank, the new bank is immediately available. After switching to the global register bank, the cached memory-mapped GPRs must be valid before any further instructions can be executed. If the global register bank is not valid at this time (in case if the context switch process has been interrupted), the cache validation process is started automatically.

## **Switching the Context of the Global Register Bank**

The contents of the global register bank are switched by changing the base address of the memory-mapped GPR bank. The base address is given by the contents of the Context Pointer (CP).

After the CP has been updated, a state machine starts to store the old contents of the global register bank and to load the new one. The store and load algorithm is executed in nineteen CPU cycles: the execution of the cache validation process takes sixteen cycles plus three cycles to stall an instruction execution to avoid pipeline conflicts upon the completion of the validation process. The context switch process has two phases:

- **Store phase:** The contents of the global register bank<sup>1)</sup> is stored back into the DPRAM by executing eight injected STORE instructions. After the last STORE instruction the contents of the global register bank are invalidated.
- **Load phase:** The global register bank is loaded with the new context by executing eight injected LOAD instructions. After the last LOAD instruction the contents of the global register bank are validated.

The code execution is stopped until the global register bank is valid again. A hardware interrupt can occur during the validation process. The way the validation process is completed depends on the type of register bank selected for this interrupt:

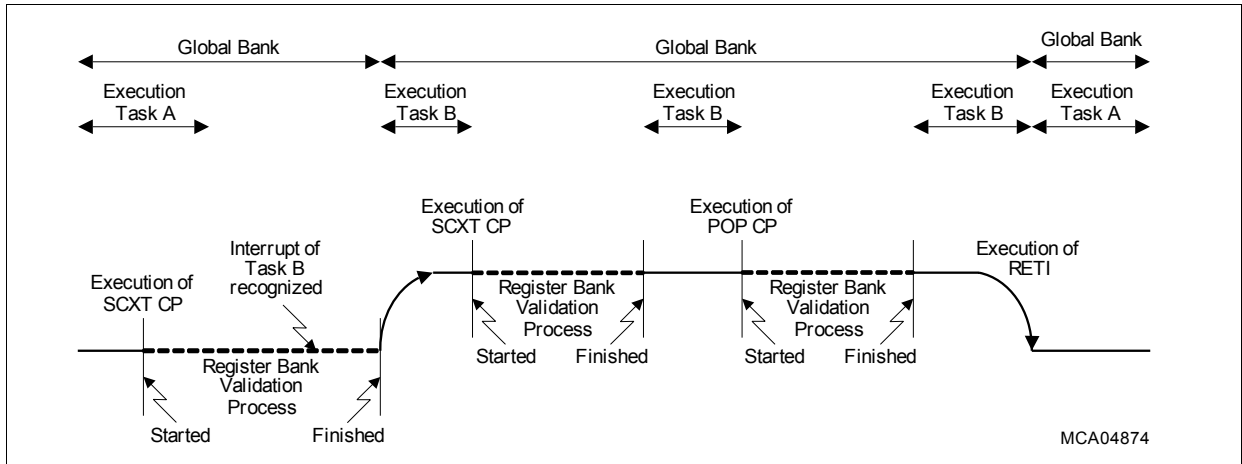
- If the interrupt also uses a global register bank the validation process is finished before executing the service routine (see [Figure 5-7](#)).
- If the interrupt uses a local register bank the validation process is interrupted and the service routine is executed immediately (see [Figure 5-8](#)). After switching back to the global register bank, the validation process is finished:
  - If the interrupt occurred during the store phase, the entire validation process is restarted from the very beginning.
  - If the interrupt occurred during the load phase, only the load phase is repeated.

If a local-bank interrupt routine (Task B in [Figure 5-9](#)) is again interrupted by a global-bank interrupt (Task C), the suspended validation process must be finished before code of Task C can be executed. This means that the validation process of Task A does not affect the interrupt latency of Task B but the latency of Task C.

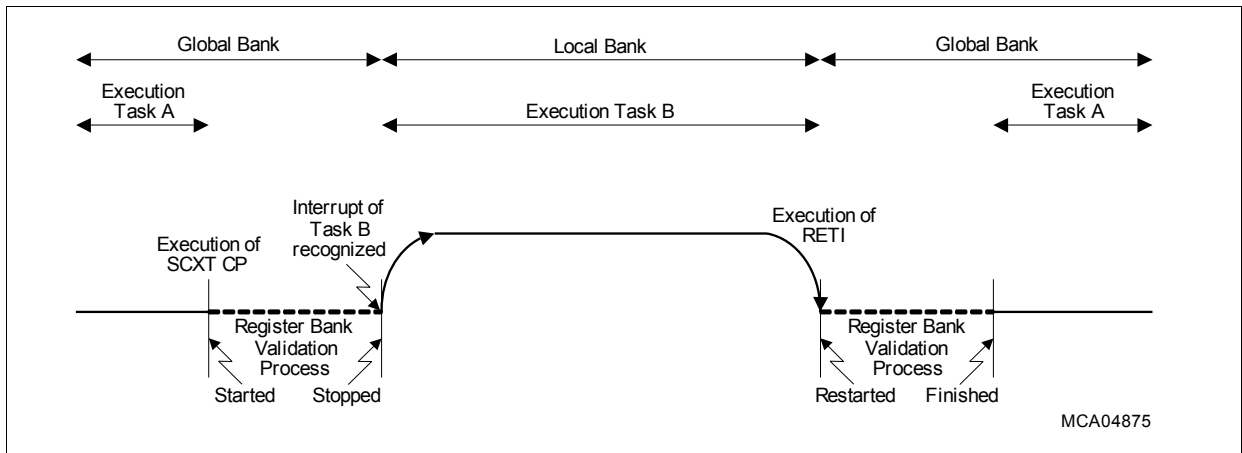
*Note: If Task C would immediately interrupt Task A, the register bank validation process of Task A would be finished first. The worst case interrupt latency is identical in both cases (see [Figure 5-7](#) and [Figure 5-9](#)).*

1) During the store phase of the context switch the complete register bank is written to the DPRAM even if the application only uses a part of this register bank. A register bank must not be located above FDE0<sub>H</sub>, otherwise the store phase will overwrite SFRs (beginning at FE00<sub>H</sub>).

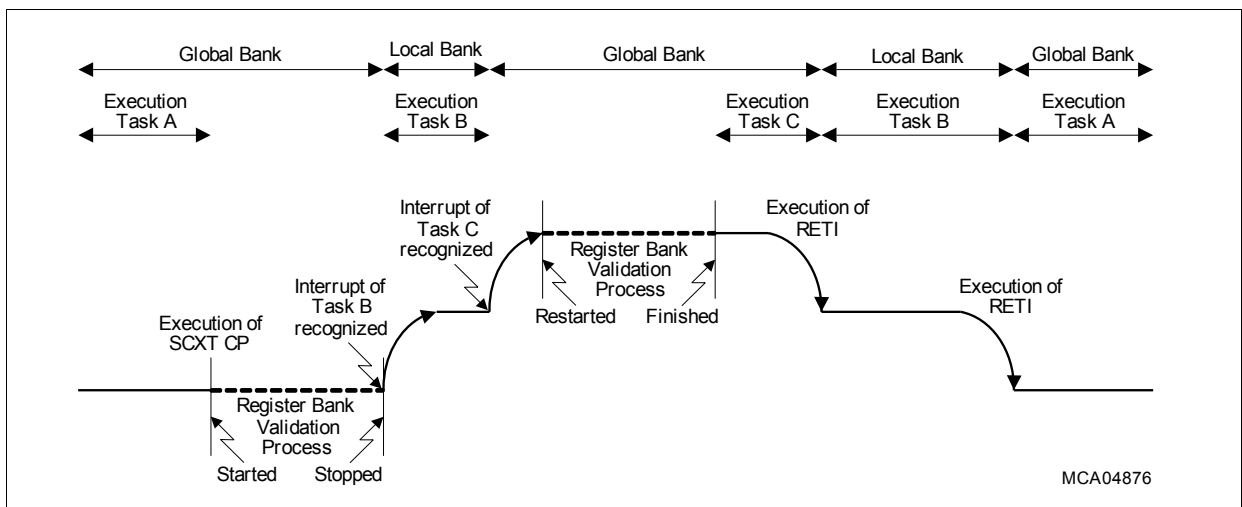




**Figure 5-7 Validation Process Interrupted by Global-Bank Interrupt**



**Figure 5-8 Validation Process Interrupted by Local-Bank Interrupt**



**Figure 5-9 Validation Process Interrupted by Local- and Global-Bank Intr.**

### 5.5.2.1 The Context Pointer (CP)

This non-bit-addressable register selects the current global register bank context. It can be updated via any instruction capable of modifying SFRs.

#### CP

**Context Pointer**

**SFR (FE10<sub>H</sub>/08<sub>H</sub>)**

**Reset Value: FC00<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>CP</b>											<b>0</b>
r	r	r	r						rw						r

Field	Bits	Type	Description
<b>1</b>	15, 14, 13, 12	r	<b>Fixed part of CP</b> Read as 1
<b>CP</b>	[11:1]	rw	<b>Modifiable part of CP</b> Specifies bits [11:1] of the 16-bit base address of the current global (memory-mapped) register bank. When writing a value to register CP with bits CP[11:9] = 000 <sub>B</sub> , bits CP[11:10] are set to 11 <sub>B</sub> by hardware.
<b>0</b>	0	r	<b>Fixed part of CP</b> Read as 0

*Note: It is the user's responsibility to ensure that the physical GPR address specified via CP register plus short GPR address is always an internal DPRAM location. If this condition is not met, unexpected results may occur. Do not set CP below the internal DPRAM start address. Do not set CP above FDE0<sub>H</sub>, otherwise the store phase will overwrite SFRs (beginning at FE00<sub>H</sub>).*

The XE16xyM switches the complete memory-mapped GPR bank with a single instruction. After switching, the service routine executes within its own separate context.

The instruction "SCXT CP, #New\_Bank" pushes the value of the current context pointer (CP) into the system stack and loads CP with the immediate value "New\_Bank", which selects a new register bank. The service routine may now use its "own registers". This memory register bank is preserved when the service routine terminates, i.e. its contents are available on the next call.

Before returning from the service routine (RETI), the previous CP is simply popped from the system stack which returns the registers to the original bank.

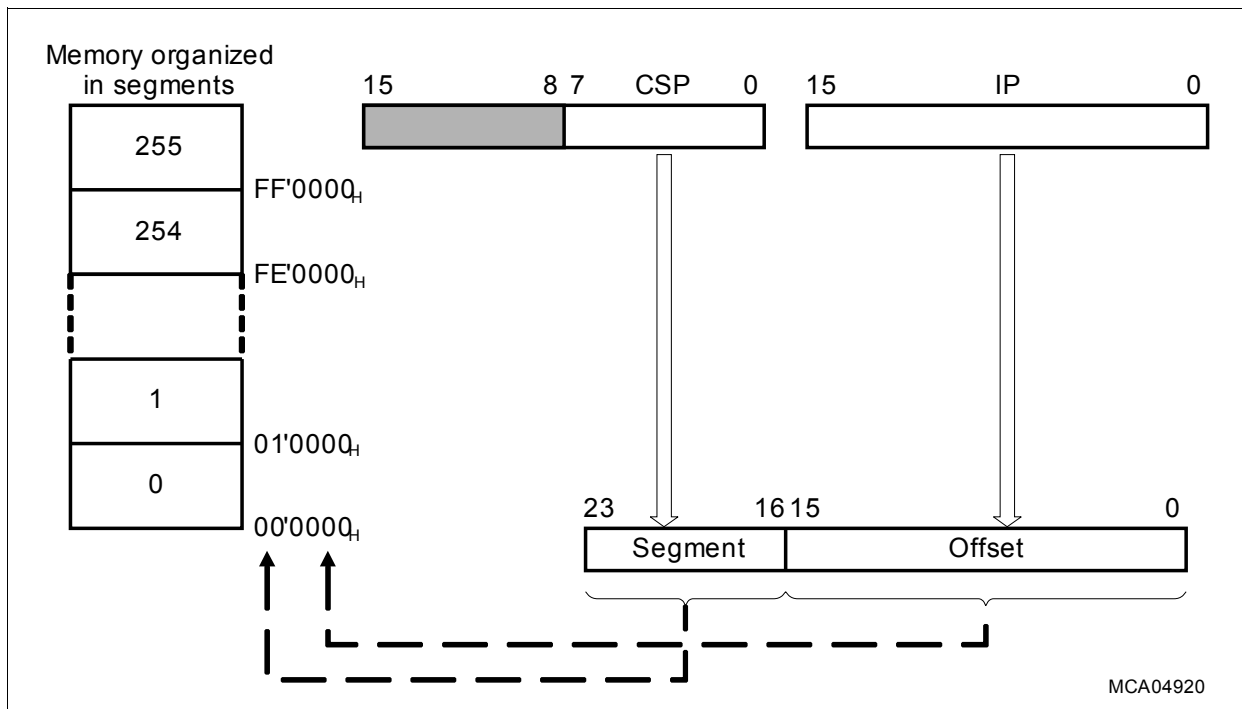
*Note: Due to the internal instruction pipeline, a write operation to the CP register stalls the instruction flow until the register file context switch is really executed. The*

**Central Processing Unit (CPU)**

*instruction immediately following the instruction that updates CP register can use the new value of the changed CP.*

## 5.6 Code Addressing

The XE16xyM provides a total addressable memory space of 16 Mbytes. This address space is arranged as 256 segments of 64 Kbytes each. A dedicated 24-bit code address pointer is used to access the memories for instruction fetches. This pointer has two parts: an 8-bit code segment pointer CSP and a 16-bit offset pointer called Instruction Pointer (IP). The concatenation of the CSP and IP results directly in a correct 24-bit physical memory address.



MCA04920

**Figure 5-10 Addressing via the Code Segment and Instruction Pointer**

**The Code Segment Pointer CSP** selects the code segment being used at run-time to access instructions. The lower 8 bits of register CSP select one of up 256 segments of 64 Kbytes each, while the higher 8 bits are reserved for future use. The hardware reset value is 0000<sub>H</sub>, but immediately after reset it is loaded with the contents of the VECSEG register due to an injected MOVCSIP instruction.

*Note: Register CSP can only be read but cannot be written by data operations.*

**In segmented memory mode** (default after reset), register CSP is modified either directly by JMPS and CALLS instructions, or indirectly via the stack by RETS and RETI instructions.

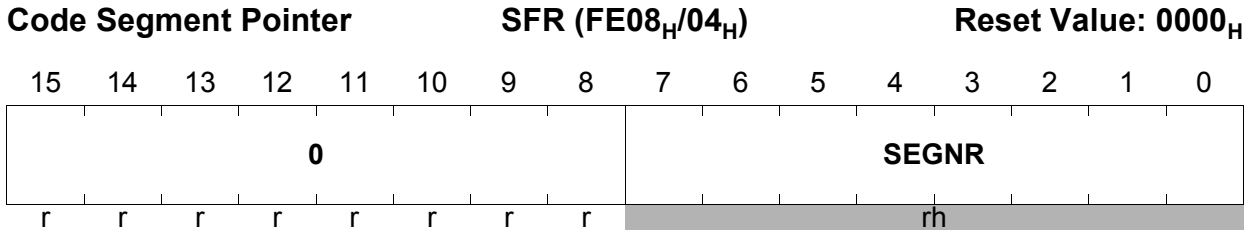
**In non-segmented memory mode** (selected by setting bit SGTDIS in register CPUCON1), CSP is fixed to the segment of the instruction that disabled segmentation. Modification by inter-segment CALLs or RETurns is no longer possible.

**Central Processing Unit (CPU)**

For processing an accepted interrupt or a TRAP, register CSP is automatically loaded with the segment of the vector table (defined in register VECSEG).

*Note: For the correct execution of interrupt tasks in non-segmented memory mode, the contents of VECSEG must select the same segment as the current value of CSP, i.e. the vector table must be located in the segment pointed to by the CSP.*

**CSP**

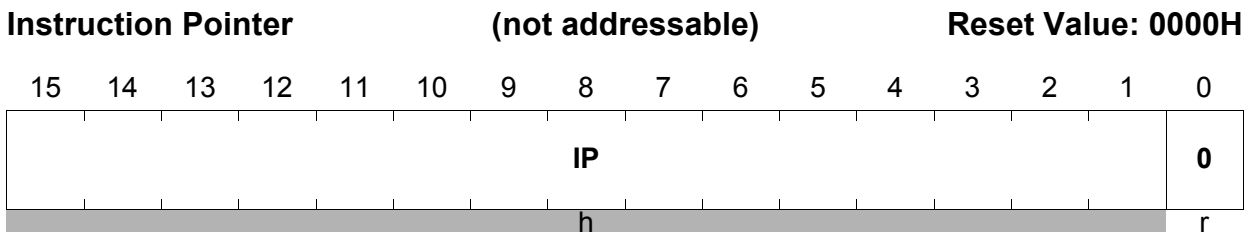


Field	Bits	Type	Description
<b>0</b>	[15:8]	r	<b>Reserved</b> Read as 0, should be written 0
<b>SEGNR</b>	[7:0]	rh	<b>Segment Number</b> Specifies the code segment from which the current instruction is to be fetched.

*Note: After a reset, register CSP is automatically loaded from register VECSEG.*

**The Instruction Pointer IP** determines the 16-bit intra-segment address of the currently fetched instruction within the code segment selected by the CSP register. Register IP is not mapped into the XE16xyM's address space; thus, it is not directly accessible by the programmer. However, the IP can be modified indirectly via the stack by means of a return instruction. IP is implicitly updated by the CPU for branch instructions and after instruction fetch operations.

**IP**



Field	Bits	Type	Description
<b>IP</b>	[15:1]	h	<b>Instruction Pointer</b> Specifies bits [15:1] of the intra segment offset from which the current instruction is to be fetched. IP refers to the current segment <SEGNR>.
<b>0</b>	0	r	<b>Fixed part of IP</b> Read as 0

## 5.7 Data Addressing

The Address Data Unit (ADU) contains two independent arithmetic units to generate, calculate, and update addresses for data accesses, the Standard Address Generation Unit (SAGU) and the DSP Address Generation Unit (DAGU). The ADU performs the following major tasks:

- Standard Address Generation (SAGU)
- DSP Address Generation (DAGU)
- Data Paging (SAGU)
- Stack Handling (SAGU)

The SAGU supports linear arithmetic for the indirect addressing modes and also generates the address in case of all other short and long addressing modes.

The DAGU contains an additional set of address pointers and offset registers which are used in conjunction with the CoXXX instructions only.

The CPU provides a lot of powerful addressing modes (short, long, indirect) for word, byte, and bit data accesses. The different addressing modes use different formats and have different scopes.

### 5.7.1 Short Addressing Modes

Short addressing modes allow access to the GPR, SFR or bit-addressable memory space. All of these addressing modes use an offset (8/4/2 bits) together with an implicit base address to specify a 24-bit physical address:

**Table 5-15 Short Addressing Modes**

<b>Mnemonic</b>	<b>Base Address<sup>1)</sup></b>	<b>Offset</b>	<b>ShortAddress Range</b>	<b>Scope of Access</b>
Rw	(CP)	$2 \times \text{Rw}$	0 ... 15	GPRs (word)
Rb	(CP)	$1 \times \text{Rb}$	0 ... 15	GPRs (byte)
reg	00'FE00 <sub>H</sub>	$2 \times \text{reg}$	00 <sub>H</sub> ... EF <sub>H</sub>	SFRs (word, low byte)
	00'F000 <sub>H</sub>	$2 \times \text{reg}$	00 <sub>H</sub> ... EF <sub>H</sub>	ESFRs (word, low byte)
	(CP)	$2 \times (\text{reg} \wedge 0\text{F}_{\text{H}})$	F0 <sub>H</sub> ... FF <sub>H</sub>	GPRs (word)
	(CP)	$1 \times (\text{reg} \wedge 0\text{F}_{\text{H}})$	F0 <sub>H</sub> ... FF <sub>H</sub>	GPRs (bytes)
bitoff	00'FD00 <sub>H</sub>	$2 \times \text{bitoff}$	00 <sub>H</sub> ... 7F <sub>H</sub>	RAM Bit word offset
	00'FF00 <sub>H</sub>	$2 \times (\text{bitoff} \wedge 7\text{F}_{\text{H}})$	80 <sub>H</sub> ... EF <sub>H</sub>	SFR Bit word offset
	00'F100 <sub>H</sub>	$2 \times (\text{bitoff} \wedge 7\text{F}_{\text{H}})$	80 <sub>H</sub> ... EF <sub>H</sub>	ESFR Bit word offset
	(CP)	$2 \times (\text{bitoff} \wedge 0\text{F}_{\text{H}})$	F0 <sub>H</sub> ... FF <sub>H</sub>	GPR Bit word offset
bitaddr	Bit word see bitoff	Immediate bit position	0 ... 15	Any single bit

1) Accesses to general purpose registers (GPRs) may also access local register banks, instead of using CP.

**Physical Address = Base Address +  $\Delta \times$  Short Address**

*Note:  $\Delta$  is 1 for byte GPRs,  $\Delta$  is 2 for word GPRs.*

**Rw, Rb:** Specifies direct access to any GPR in the currently active context (global register bank or local register bank). Both 'Rw' and 'Rb' require four bits in the instruction format. The base address of the global register bank is determined by the contents of register CP. 'Rw' specifies a 4-bit word GPR address, 'Rb' specifies a 4-bit byte GPR address within a local register bank or relative to (CP).

**reg:** Specifies direct access to any (E)SFR or GPR in the currently active context (global or local register bank). The 'reg' value requires eight bits in the instruction format. Short 'reg' addresses in the range from 00<sub>H</sub> to EF<sub>H</sub> always specify (E)SFRs. In that case, the factor ' $\Delta$ ' equates 2 and the base address is 00'FE00<sub>H</sub> for the standard SFR area or 00'F000<sub>H</sub> for the extended ESFR area. The 'reg' accesses to the ESFR area require a preceding EXT\*R instruction to switch the base address. Depending on the opcode, either the total word (for word operations) or the low byte (for byte operations) of an SFR can be addressed via 'reg'. Note that the high byte of an SFR cannot be accessed via the 'reg' addressing mode. Short 'reg' addresses in the range from F0<sub>H</sub> to FF<sub>H</sub> always specify GPRs. In that case, only the lower four bits of 'reg' are significant for physical address generation and, therefore, it is identical to the address generation described for the 'Rb' and 'Rw' addressing modes.

**bitoff:** Specifies direct access to any word in the bit addressable memory space. The 'bitoff' value requires eight bits in the instruction format. The specified 'bitoff' range selects different base addresses to generate physical addresses (see [Table 5-15](#)). The 'bitoff' accesses to the ESFR area require a preceding EXT\*R instruction to switch the base address.

**bitaddr:** Any bit address is specified by a word address within the bit addressable memory space (see 'bitoff') and a bit position ('bitpos') within that word. Therefore, 'bitaddr' requires twelve bits in the instruction format.

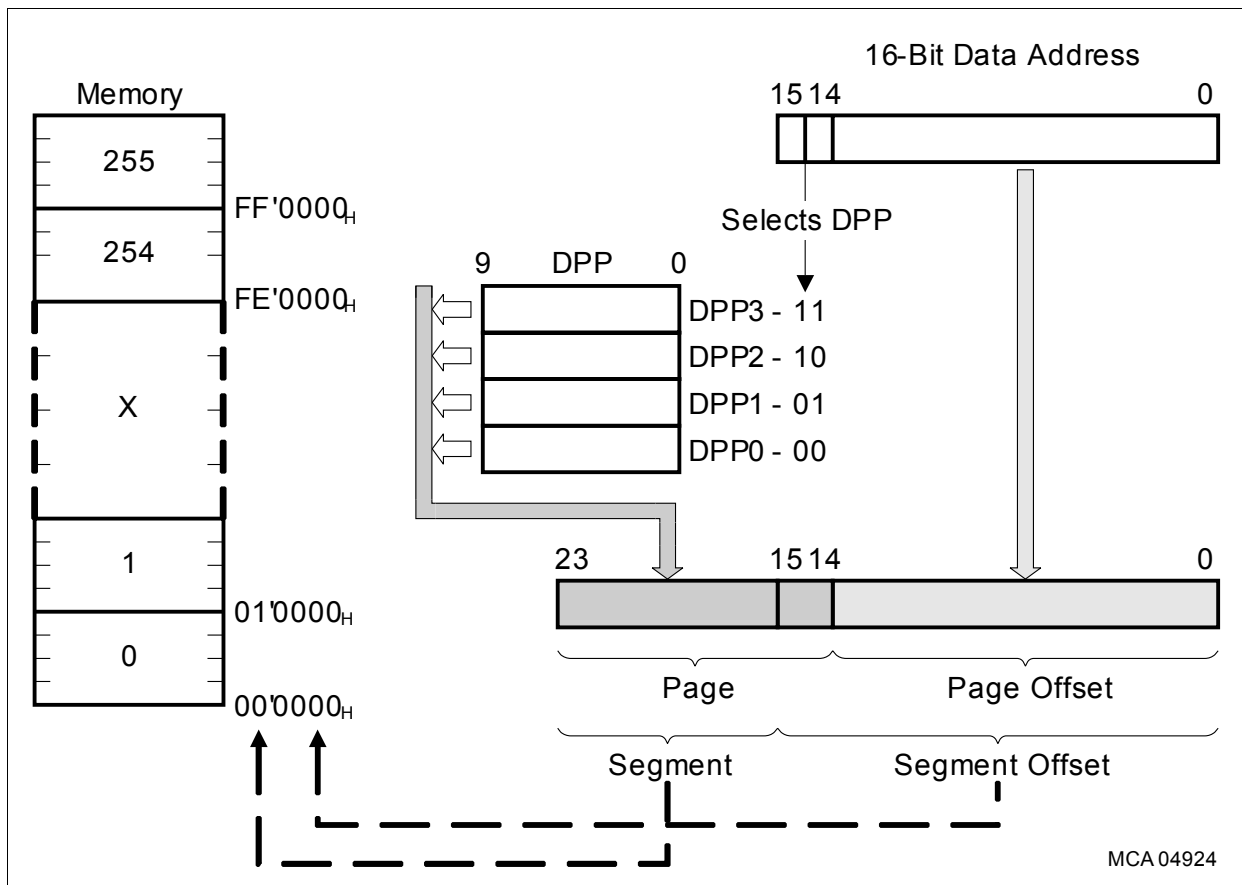


## 5.7.2 Long Addressing Modes

Long addressing modes specify 24-bit addresses and, therefore, can access any word or byte data within the entire address space. Long addresses can be specified in different ways to generate the full 24-bit address:

- **Use one of the four Data Page Pointers (DPP registers):** The used 16-bit pointer selects a DPP with bits 15 ... 14, bits 13 ... 0 specify the 14-bit data page offset (see [Figure 5-11](#)).
- **Select the used data page directly:** The data page is selected by a preceding EXTP(R) instruction, bits 13 ... 0 of the used 16-bit pointer specify the 14-bit data page offset.
- **Select the used segment directly:** The segment is selected by a preceding EXT(S) instruction, the used 16-bit pointer specifies the 16-bit segment offset.

*Note: Word accesses on odd byte addresses are not executed. A hardware trap will be triggered.*



**Figure 5-11 Data Page Pointer Addressing**

### 5.7.2.1 Data Page Pointers DPP0, DPP1, DPP2, DPP3

These four non-bit-addressable registers select up to four different data pages to be active simultaneously at run-time. The lower 10 bits of each DPP register select one of the 1024 possible 16-Kbyte data pages; the upper 6 bits are reserved for future use.

#### DPP0

**Data Page Pointer 0**                      **SFR (FE00<sub>H</sub>/00<sub>H</sub>)**                      **Reset Value: 0000<sub>H</sub>**

#### DPP1

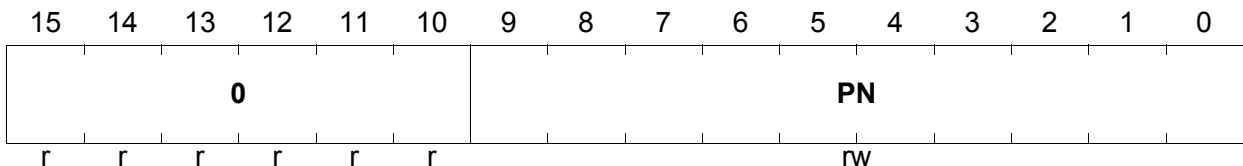
**Data Page Pointer 1**                      **SFR (FE02<sub>H</sub>/01<sub>H</sub>)**                      **Reset Value: 0001<sub>H</sub>**

#### DPP2

**Data Page Pointer 2**                      **SFR (FE04<sub>H</sub>/02<sub>H</sub>)**                      **Reset Value: 0002<sub>H</sub>**

#### DPP3

**Data Page Pointer 3**                      **SFR (FE06<sub>H</sub>/03<sub>H</sub>)**                      **Reset Value: 0003<sub>H</sub>**



Field	Bits	Type	Description
<b>0</b>	[15:8]	r	<b>Reserved</b> Read as 0, should be written 0
<b>PN</b>	[9:0]	rw	<b>Data Page Number of DPPx</b> Specifies the data page selected via DPPx.

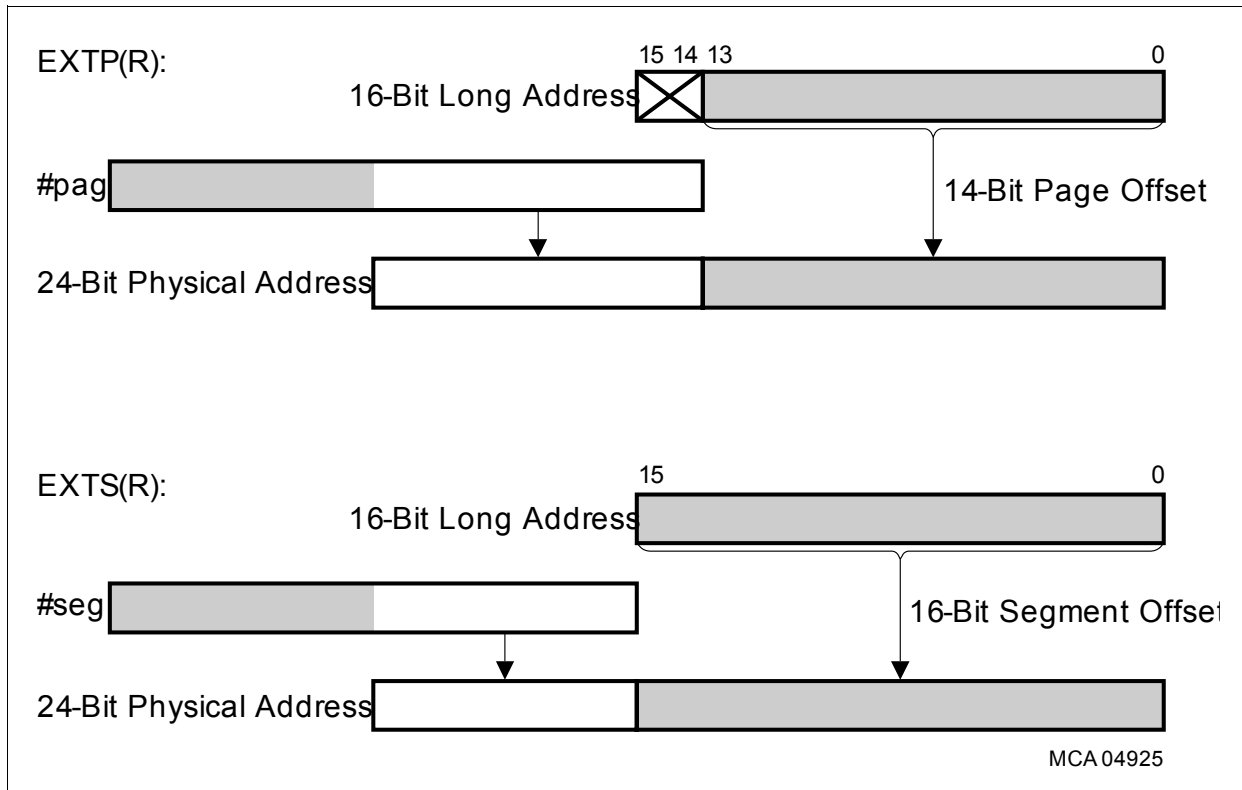
The DPP registers allow access to the entire memory space in pages of 16 Kbytes each. The DPP registers are implicitly used whenever data accesses to any memory location are made via indirect or direct long 16-bit addressing modes (except for override accesses via EXTended instructions and PEC data transfers). After reset, the Data Page Pointers are initialized in such a way that all indirect or direct long 16-bit addresses result in identical 18-bit addresses. This allows access to data pages 3 ... 0 within segment 0 as shown in [Figure 5-11](#). If the user does not want to use data paging, no further action is required.

Data paging is performed by concatenating the lower 14 bits of an indirect or direct long 16-bit address with the contents of the DPP register selected by the upper two bits of the 16-bit address. The contents of the selected DPP register specify one of the 1024 possible data pages. This data page base address together with the 14-bit page offset forms the physical 24-bit address (even if segmentation is disabled).

A DPP register can be updated via any instruction capable of modifying an SFR.

**Central Processing Unit (CPU)**

*Note: Due to the internal instruction pipeline, a write operation to the DPPx registers could stall the instruction flow until the DPP is actually updated. The instruction that immediately follows the instruction which updates the DPP register can use the new value of the changed DPPx.*



**Figure 5-12 Overriding the DPP Mechanism**

*Note: The overriding page or segment may be specified as a constant (#pag, #seg) or via a word GPR (Rw).*

**Table 5-16 Long Addressing Modes**

Mnemonic	Base Address <sup>1)</sup>	Offset	Scope of Access
mem	(DPPx)	mem $\wedge$ 3FFF <sub>H</sub>	Any Word or Byte
mem	pag	mem $\wedge$ 3FFF <sub>H</sub>	Any Word or Byte
mem	seg	mem	Any Word or Byte

1) Represents either a 10-bit data page number to be concatenated with a 14-bit offset, or an 8-bit segment number to be concatenated with a 16-bit offset.

### 5.7.3 Indirect Addressing Modes

Indirect addressing modes can be considered as a combination of short and long addressing modes. This means that the “long” 16-bit pointer is provided indirectly by the contents of a word GPR which itself is specified directly by a short 4-bit address ('Rw' = 0 ... 15).

There are indirect addressing modes, which add a constant value to the GPR contents before the long 16-bit address is calculated. Other indirect addressing modes can decrement or increment the indirect address pointers (GPR contents) by 2 or 1 (referring to words or bytes) or by the contents of the offset registers QR0 or QR1.

**Table 5-17 Generating Physical Addresses from Indirect Pointers**

Step	Executed Action	Calculation	Notes
1	Calculate the address of the indirect pointer (word GPR) from its short address	<b>GPR Address =</b> <b>2 × Short Addr.</b> <b>[+ (CP)]</b>	see <a href="#">Table 5-15</a>
2	Pre-decrement indirect pointer ('-Rw') depending on datatype ( $\Delta = 1$ or 2 for byte or word operations)	<b>(GPR Address) =</b> <b>(GPR Address)</b> <b>- <math>\Delta</math></b>	Optional step, executed only if required by addressing mode
3	Adjust the pointer by a constant value ('Rw + const16')	<b>Pointer =</b> <b>(GPR Address)</b> <b>+ Constant</b>	Optional step, executed only if required by addressing mode
4	Calculate the physical 24-bit address using the resulting pointer	<b>Physical Addr. =</b> <b>Page/Segment +</b> <b>Pointer offset</b>	Uses DPPs or page/segment override mechanisms, see <a href="#">Table 5-16</a>
5	Post-in/decrement indirect pointer ('Rw $\pm$ ') depending on datatype ( $\Delta = 1$ or 2 for byte or word operations), or depending on offset registers ( $\Delta = \text{QRx}$ ) <sup>1)</sup>	<b>(GPR Address) =</b> <b>(GPR Address)</b> <b><math>\pm \Delta</math></b>	Optional step, executed only if required by addressing mode

1) Post-decrement and QRx-based modification is provided only for CoXXX instructions.

*Note: Some instructions only use the lowest four word GPRs (R3 ... R0) as indirect address pointers, which are specified via short 2-bit addresses in that case.*

The following indirect addressing modes are provided:

**Table 5-18 Indirect Addressing Modes**

<b>Mnemonic</b>	<b>Particularities</b>
[Rw]	Most instructions accept any GPR (R15 ... R0) as indirect address pointer. Some instructions accept only the lower four GPRs (R3 ... R0).
[Rw+]	The specified indirect address pointer is automatically post-incremented by 2 or 1 (for word or byte data operations) after the access.
[-Rw]	The specified indirect address pointer is automatically pre-decremented by 2 or 1 (for word or byte data operations) before the access.
[Rw + #data16]	The specified 16-bit constant is added to the indirect address pointer, before the long address is calculated.
[Rw-]	The specified indirect address pointer is automatically post-decremented by 2 (word data operations) after the access.
[Rw + QRx]	The specified indirect address pointer is automatically post-incremented by QRx (word data operations) after the access.
[Rw - QRx]	The specified indirect address pointer is automatically post-decremented by QRx (word data operations) after the access.

### 5.7.3.1 Offset Registers QR0 and QR1

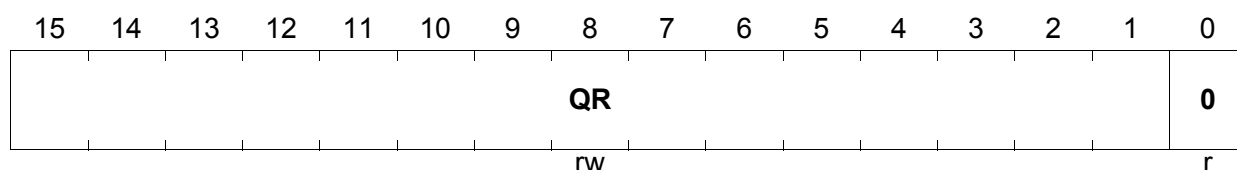
The non-bit-addressable offset registers QR0 and QR1 are used with CoXXX instructions. For possible instruction flow stalls refer to [Section 5.3.2.4](#).

#### QR0

**Offset Register** **ESFR (F004<sub>H</sub>/02<sub>H</sub>)** **Reset Value: 0000<sub>H</sub>**

#### QR1

**Offset Register** **ESFR (F006<sub>H</sub>/03<sub>H</sub>)** **Reset Value: 0000<sub>H</sub>**



<b>Field</b>	<b>Bits</b>	<b>Type</b>	<b>Description</b>
<b>QR</b>	[15:1]	rw	<b>Modifiable part of QRx</b> Specifies the 16-bit offset address for indirect addressing modes (LSB always zero).
<b>0</b>	0	r	<b>Fixed part of QRx</b> Read as 0



**Central Processing Unit (CPU)**

There are indirect addressing modes which allow parallel data move operations before the long 16-bit address is calculated (see [Figure 5-14](#) for an example). Other indirect addressing modes allow decrementing or incrementing the indirect address pointers (IDXx contents) by 2 or by the contents of the offset registers QX0 and QX1 (used in conjunction with the IDX pointers).

**QX0**

**Offset Register**

**ESFR (F000<sub>H</sub>/00<sub>H</sub>)**

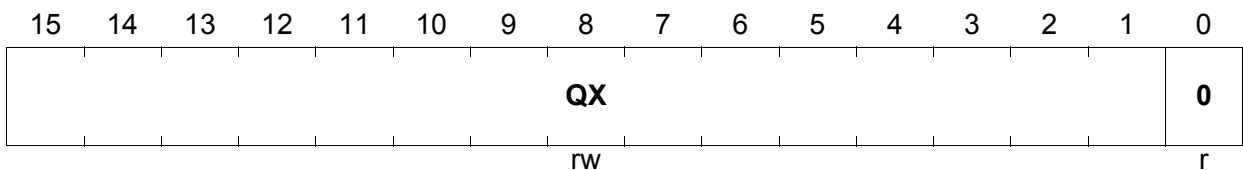
**Reset Value: 0000<sub>H</sub>**

**QX1**

**Offset Register**

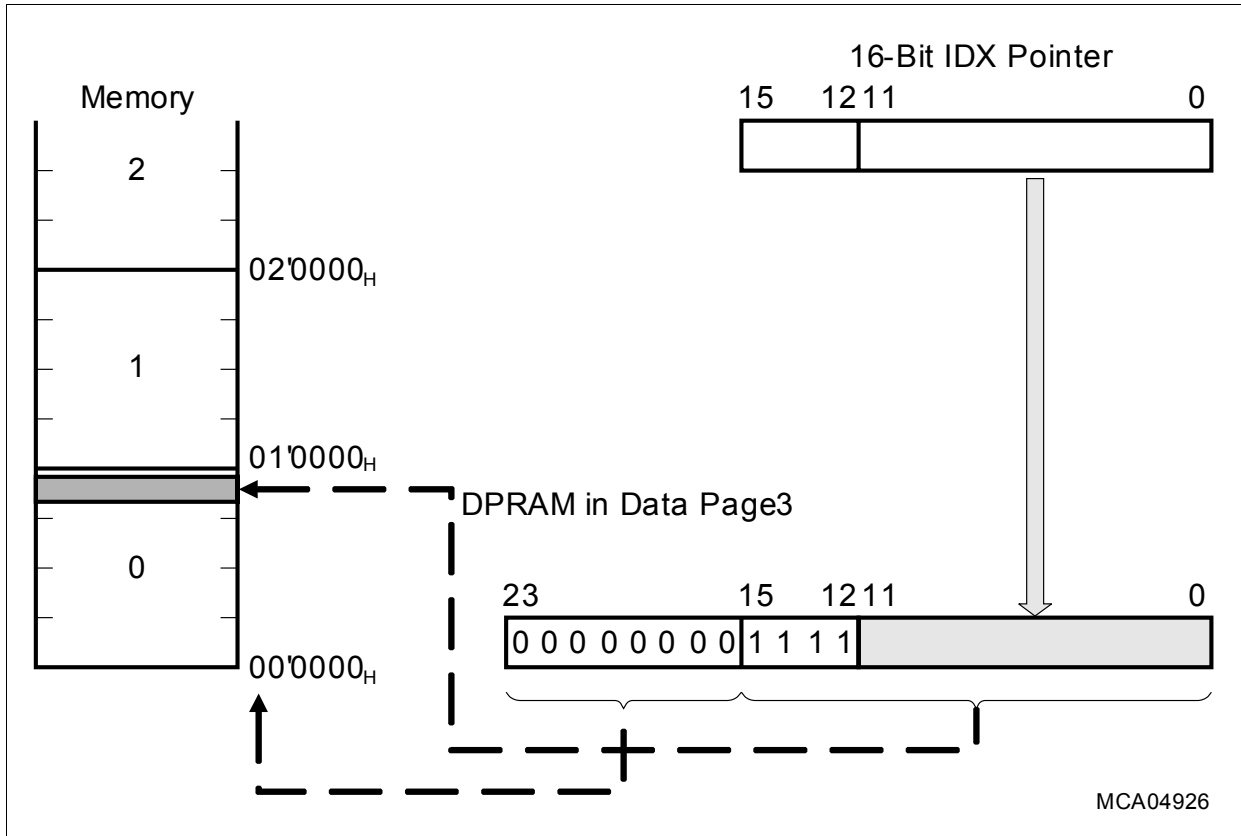
**ESFR (F002<sub>H</sub>/01<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
QX	[15:1]	rw	<b>Modifiable part of QXx</b> Specifies the 16-bit word offset for indirect addressing modes
0	0	r	<b>Fixed part of QXx</b> Read as 0

*Note: During the initialization of the QX registers, instruction flow stalls are possible. For the proper operation, refer to [Section 5.3.2.4](#).*



**Figure 5-13 Arithmetic MAC Operations and Addressing via the IDX Pointers**

**Table 5-19 Generating Physical Addresses from Indirect Pointers (IDXx)**

Step	Executed Action	Calculation	Notes
1	Determine the used IDXx pointer	---	—
2	Calculate an intermediate long address for the parallel data move operation and in/decrement indirect pointer ('IDXx±') by 2 ( $\Delta = 2$ ), or depending on offset registers ( $\Delta = QXx$ )	<b>Interm. Addr. = (IDXx Address) <math>\pm \Delta</math></b>	Optional step, executed only if required by instruction CoXXxM and addressing mode
3	Calculate long 16-bit address	<b>Long Address = (IDXx Pointer)</b>	—



**Table 5-19 Generating Physical Addresses from Indirect Pointers (IDXx) (cont'd)**

Step	Executed Action	Calculation	Notes
4	Calculate the physical 24-bit address using the resulting pointer	<b>Physical Addr. = Page/Segment + Pointer offset</b>	Uses DPPs or page/segment override mechanisms, see <a href="#">Table 5-16</a> and <a href="#">Figure 5-13</a>
5	Post-in/decrement indirect pointer ('IDXx±') by 2 ( $\Delta = 2$ ), or depending on offset registers ( $\Delta = QXx$ )	<b>(IDXx Pointer) = (IDXx Pointer) <math>\pm \Delta</math></b>	Optional step, executed only if required by addressing mode

The following indirect addressing modes are provided:

**Table 5-20 DSP Addressing Modes**

Mnemonic	Particularities
[IDXx]	Most CoXXX instructions accept IDXx (IDX0, IDX1) as an indirect address pointer.
[IDXx+]	The specified indirect address pointer is automatically post-incremented by 2 after the access.
with parallel data move	In case of a CoXXXM instruction, the address stored in the specified indirect address pointer is automatically pre-decremented by 2 for the parallel move operation. The pointer itself is not pre-decremented. Then, the specified indirect address pointer is automatically post-incremented by 2 after the access.
[IDXx-]	The specified indirect address pointer is automatically post-decremented by 2 after the access.
with parallel data move	In case of a CoXXXM instruction, the address stored in the specified indirect address pointer is automatically pre-incremented by 2 for the parallel move operation. The pointer itself is not pre-incremented. Then, the specified indirect address pointer is automatically post-decremented by 2 after the access.
[IDXx + QXx]	The specified indirect address pointer is automatically post-incremented by QXx after the access.
with parallel data move	In case of a CoXXXM instruction, the address stored in the specified indirect address pointer is automatically pre-decremented by QXx for the parallel move operation. The pointer itself is not pre-decremented. Then, the specified indirect address pointer is automatically post-incremented by QXx after the access.

**Table 5-20 DSP Addressing Modes (cont'd)**

<b>Mnemonic</b>	<b>Particularities</b>
[IDXx - QXx]	The specified indirect address pointer is automatically post-decremented by QXx after the access.
with parallel data move	In case of a CoXXXM instruction, the address stored in the specified indirect address pointer is automatically pre-incremented by QXx for the parallel move operation. The pointer itself is not pre-incremented. Then, the specified indirect address pointer is automatically post-decremented by QXx after the access.

*Note: An example for parallel data move operations can be found in [Figure 5-14](#).*

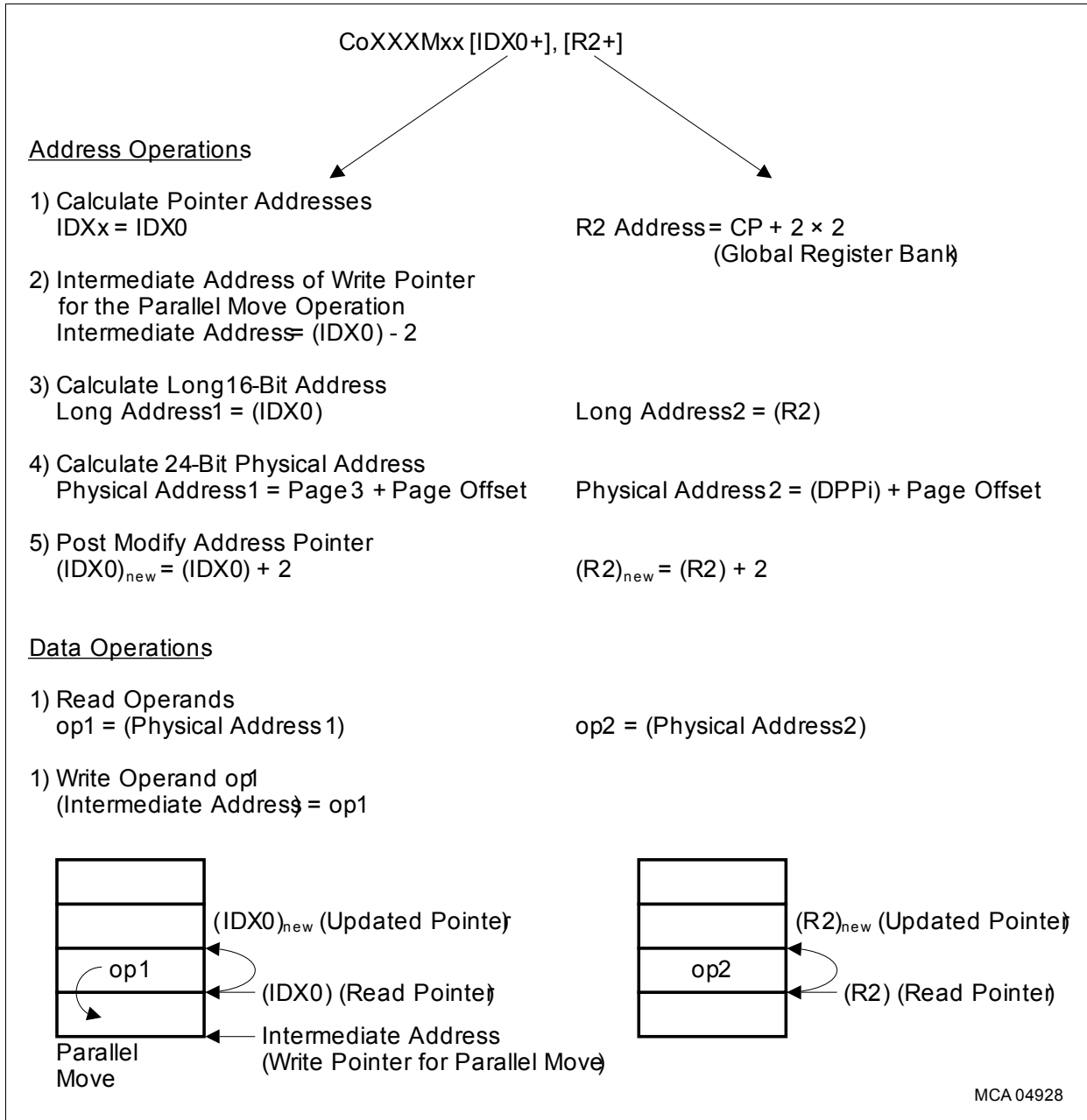
### **The CoREG Addressing Mode**

The CoSTORE instruction utilizes the special CoREG addressing mode for immediate storage of the MAC-Unit register after a MAC operation. The address of the MAC-Unit register is coded in the CoSTORE instruction format as described in [Table 5-21](#):

**Table 5-21 Coding of the CoREG Addressing Mode**

<b>Mnemonic</b>	<b>Register</b>	<b>Coding of www:w bits [31:27]</b>
MSW	MAC-Unit Status Word	00000 <sub>B</sub>
MAH	MAC-Unit Accumulator High Word	00001 <sub>B</sub>
MAS	Limited MAC-Unit Accumulator High Word	00010 <sub>B</sub>
MAL	MAC-Unit Accumulator Low Word	00100 <sub>B</sub>
MCW	MAC-Unit Control Word	00101 <sub>B</sub>
MRW	MAC-Unit Repeat Word	00110 <sub>B</sub>

The example in [Figure 5-14](#) shows the complex operation of CoXXXM instructions with a parallel move operation based on the descriptions about addressing modes given in [Section 5.7.3 \(Indirect Addressing Modes\)](#) and [Section 5.7.4 \(DSP Addressing Modes\)](#).



**Figure 5-14 Arithmetic MAC Operations with Parallel Move**

## 5.7.5 The System Stack

The XE16xyM supports a system stack of up to 64 Kbytes. The stack can be located internally in one of the on-chip memories or externally. The 16-bit Stack Pointer register (SP) addresses the stack within a 64-Kbyte segment selected by the Stack Pointer Segment register (SPSEG). A virtual stack (usually bigger than 64 Kbytes) can be implemented by software. This mechanism is supported by the Stack Overflow register STKOV and the Stack Underflow register STKUN (see descriptions below).

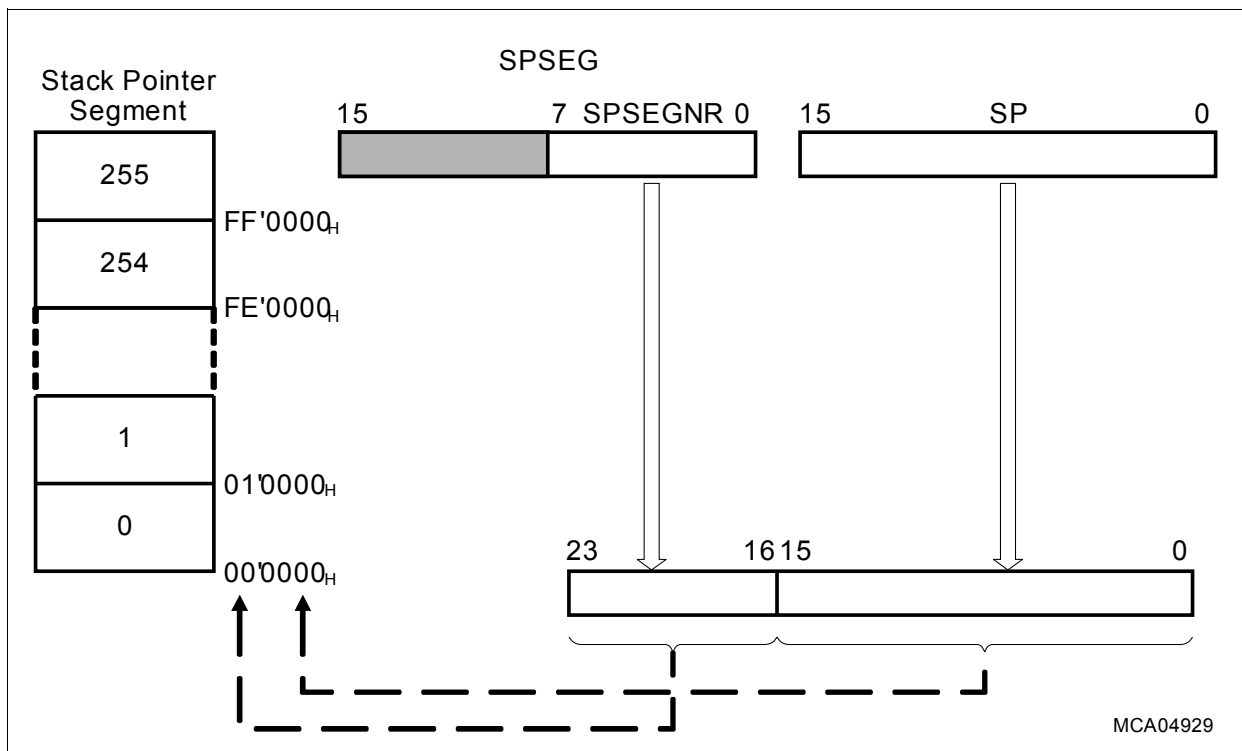
### 5.7.5.1 The Stack Pointer Registers SP and SPSEG

Register SPSEG (not bit addressable) selects the segment being used at run-time to access the system stack. The lower eight bits of register SPSEG select one of up to 256 segments of 64 Kbytes each, while the higher 8 bits are reserved for future use.

The Stack Pointer SP (not bit addressable) points to the top of the system stack (TOS). SP is pre-decremented whenever data is pushed onto the stack, and it is post-incremented whenever data is popped from the stack. Therefore, the system stack grows from higher towards lower memory locations.

System stack addresses are generated by directly extending the 16-bit contents of register SP by the contents of register SPSEG, as shown in [Figure 5-15](#).

The system stack cannot cross a 64-Kbyte segment boundary.



**Figure 5-15 Addressing via the Stack Pointer**

### SP

#### Stack Pointer

**SFR (FE12<sub>H</sub>/09<sub>H</sub>)**

**Reset Value: FC00<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>SP</b>															<b>0</b>
rwh															r

Field	Bits	Type	Description
<b>SP</b>	[15:1]	rwh	<b>Modifiable part of SP</b> Specifies bits [15:1] of the 16-bit system stack pointer intra segment address
<b>0</b>	0	r	<b>Fixed part of SP</b> Read as 0

### SPSEG

#### Stack Pointer Segment

**SFR (FF0C<sub>H</sub>/86<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>0</b>								<b>SPSEGNR</b>							
r	r	r	r	r	r	r	r	rw							

Field	Bits	Type	Description
<b>0</b>	[15:8]	r	<b>Reserved</b> Read as 0, should be written 0
<b>SPSEGNR</b>	[7:0]	rw	<b>Stack Pointer Segment Number</b> Specifies the segment where the stack is located.

*Note: SPSEG and SP can be updated via any instruction capable of modifying a 16-bit SFR. Due to the internal instruction pipeline, a write operation to SPSEG or SP stalls the instruction flow until the register is really updated. The instruction immediately following the instruction updating SPSEG or SP can use the new value.*

### **5.7.5.2 The Stack Overflow/Underflow Pointers STKOV/STKUN**

These limit registers (not bit-addressable) supervise the stack pointer. A trap is generated when the stack pointer reaches its upper or lower limit. The Stack Pointer Segment Register SPSG is not taken into account for the stack pointer comparison. The system stack cannot cross a 64-Kbyte segment.

STKOV is compared with SP before each implicit write operation which decrements the contents of SP (instructions CALLA, CALLI, CALLR, CALLS, PCALL, TRAP, SCXT, or PUSH). If the contents of SP are equal to the contents of STKOV a stack overflow trap is triggered.

STKUN is compared with SP before each implicit read operation which increments the contents of SP (instructions RET, RETS, RETP, RETI, or POP). If the contents of SP are equal to the contents of STKUN a stack underflow trap is triggered.

The Stack Overflow/Underflow Traps may be used in two different ways:

- **Fatal error indication** treats the stack overflow as a system error and executes the associated trap service routine.  
In case of a stack overflow trap, data in the bottom of the stack may have been overwritten by the status information stacked upon servicing the trap itself.
- **Virtual stack control** allows the system stack to be used as a 'Stack Cache' for a bigger external user stack: flush cache in case of an overflow, refill cache in case of an underflow.

### **Scope of Stack Limit Control**

The stack limit control implemented by the register pair STKOV and STKUN detects cases in which the Stack Pointer (SP) crosses the defined stack area as a result of an implicit change.

If the stack pointer was explicitly changed as a result of move or arithmetic instruction, SP is not compared to the contents of STKOV and STKUN. In this case, a stack violation will not be detected if the modified stack pointer is on or outside the defined limits, i.e. below (STKOV) or above (STKUN). Stack overflow/underflow is detected only in case of implicit SP modification.

SP may be operated outside the permitted SP range without triggering a trap. However, if SP reaches the limit of the permitted SP range from outside the range as a result of an implicit change (PUSH or POP, for example), the respective trap will be triggered.

*Note: STKOV and STKUN can be updated via any instruction capable of modifying an SFR. If a stack overflow or underflow event occurs in an ATOMIC/EXT sequence, the stack operations that are part of the sequence are completed. The trap is issued after the completion of the entire ATOMIC/EXT sequence.*

### STKOV

**Stack Overflow Pointer**

**SFR (FE14<sub>H</sub>/0A<sub>H</sub>)**

**Reset Value: FA00<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>STKOV</b>															<b>0</b>
rw															r

Field	Bits	Type	Description
<b>STKOV</b>	[15:1]	rw	<b>Modifiable part of STKOV</b> Specifies the segment offset address of the lower limit of the system stack.
<b>0</b>	0	r	<b>Fixed part of STKOV</b> Read as 0

### STKUN

**Stack Underflow Pointer**

**SFR (FE16<sub>H</sub>/0B<sub>H</sub>)**

**Reset Value: FC00<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>STKUN</b>															<b>0</b>
rw															r

Field	Bits	Type	Description
<b>STKUN</b>	[15:1]	rw	<b>Modifiable part of STKUN</b> Specifies the segment offset address of the upper limit of the system stack.
<b>0</b>	0	r	<b>Fixed part of STKUN</b> Read as 0

## 5.8 Standard Data Processing

All standard arithmetic, shift-, and logical operations are performed in the 16-bit ALU. In addition to the standard functions, the ALU of the XE16xyM includes a bit-manipulation unit and a multiply and divide unit. Most internal execution blocks have been optimized to perform operations on either 8-bit or 16-bit numbers. After the pipeline has been filled, most instructions are completed in one CPU cycle. The status flags are automatically updated in register PSW after each ALU operation and reflect the current state of the microcontroller. These flags allow branching upon specific conditions. Support of both signed and unsigned arithmetic is provided by the user selectable branch test. The status flags are also preserved automatically by the CPU upon entry into an interrupt or trap routine. Another group of bits represents the current CPU interrupt status. Two separate bits (USR0 and USR1) are provided as general purpose flags.

### PSW

**Processor Status Word**                      **SFR (FF10<sub>H</sub>/88<sub>H</sub>)**                      **Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ILVL				IEN	HLD EN_ PL1	BANK		USR 1	USR 0	PL0	E	Z	V	C	N
rwh				rw	rwh	rwh		rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
ILVL	[15:12]	rwh	<b>CPU Priority Level</b> 0 <sub>H</sub> Lowest Priority ... F <sub>H</sub> Highest Priority
IEN	11	rw	<b>Global Interrupt/PEC Enable Bit</b> 0 <sub>B</sub> Interrupt/PEC requests are disabled 1 <sub>B</sub> Interrupt/PEC requests are enabled
HLDEN_PL1	10	rwh	<b>Hold Enable/Protection Level selection 1</b> 0 <sub>B</sub> external bus arbitration disabled or protection level 0/1 (refer to <a href="#">Table 5-23</a> ) 1 <sub>B</sub> external bus arbitration enabled or protection level 2/3 (refer to <a href="#">Table 5-23</a> )
BANK	[9:8]	rwh	<b>Reserved for Register File Bank Selection</b> 00 <sub>B</sub> Global register bank 01 <sub>B</sub> Reserved 10 <sub>B</sub> Local register bank 1 11 <sub>B</sub> Local register bank 2



**Central Processing Unit (CPU)**

Field	Bits	Type	Description
<b>USR1</b>	7	rwh	<b>General Purpose Flag</b> Can be used by application software. Also set when using repeated MAC instructions ( <a href="#">Section 5.9.11</a> )
<b>USR0</b>	6	rwh	<b>General Purpose Flag</b> Can be used by application software. Also set when using repeated MAC instructions ( <a href="#">Section 5.9.11</a> )
<b>PL0</b>	5	rwh	<b>Protection Level selection 0</b> 0 <sub>B</sub> Protection level 0/2 (refer to <a href="#">Table 5-23</a> ) 1 <sub>B</sub> Protection level 1/3 (refer to <a href="#">Table 5-23</a> )
<b>E</b>	4	rwh	<b>End of Table Flag</b> 0 <sub>B</sub> Source operand is neither 8000 <sub>H</sub> nor 80 <sub>H</sub> 1 <sub>B</sub> Source operand is 8000 <sub>H</sub> or 80 <sub>H</sub>
<b>Z</b>	3	rwh	<b>Zero Flag</b> 0 <sub>B</sub> ALU result is not zero 1 <sub>B</sub> ALU result is zero
<b>V</b>	2	rwh	<b>Overflow Flag</b> 0 <sub>B</sub> No Overflow produced 1 <sub>B</sub> Overflow produced
<b>C</b>	1	rwh	<b>Carry Flag</b> 0 <sub>B</sub> No carry/borrow bit produced 1 <sub>B</sub> Carry/borrow bit produced
<b>N</b>	0	rwh	<b>Negative Result</b> 0 <sub>B</sub> ALU result is not negative 1 <sub>B</sub> ALU result is negative

### **ALU/MAC Status (N, C, V, Z, E)**

The condition flags (N, C, V, Z, E) within the PSW indicate the ALU status after the most recently performed ALU operation. They are set by most of the instructions according to specific rules which depend on the ALU or data movement operation performed by an instruction.

After execution of an instruction which explicitly updates the PSW register, the condition flags cannot be interpreted as described below because any explicit write to the PSW register supersedes the condition flag values which are implicitly generated by the CPU. Explicitly reading the PSW register supplies a read value which represents the state of the PSW register after execution of the immediately preceding instruction.

*Note: After reset, all of the ALU status bits are cleared.*

**Central Processing Unit (CPU)**

**N-Flag:** For most of the ALU operations, the N-flag is set to 1, if the most significant bit of the result contains a 1; otherwise, it is cleared. In the case of integer operations, the N-flag can be interpreted as the sign bit of the result (negative: N = 1, positive: N = 0). Negative numbers are always represented as the 2's complement of the corresponding positive number. The range of signed numbers extends from  $-8000_H$  to  $+7FFF_H$  for the word data type, or from  $-80_H$  to  $+7F_H$  for the byte data type. For Boolean bit operations with only one operand, the N-flag represents the previous state of the specified bit. For Boolean bit operations with two operands, the N-flag represents the logical XORing of the two specified bits.

**C-Flag:** After an addition, the C-flag indicates that a carry from the most significant bit of the specified word or byte data type has been generated. After a subtraction or a comparison, the C-flag indicates a borrow which represents the logical negation of a carry for the addition.

This means that the C-flag is set to 1, if **no** carry from the most significant bit of the specified word or byte data type has been generated during a subtraction, which is performed internally by the ALU as a 2's complement addition, and, the C-flag is cleared when this complement addition caused a carry.

The C-flag is always cleared for logical, multiply and divide ALU operations, because these operations cannot cause a carry.

For shift and rotate operations, the C-flag represents the value of the bit shifted out last. If a shift count of zero is specified, the C-flag will be cleared. The C-flag is also cleared for a prioritize ALU operation, because a 1 is never shifted out of the MSB during the normalization of an operand.

For Boolean bit operations with only one operand, the C-flag is always cleared. For Boolean bit operations with two operands, the C-flag represents the logical ANDing of the two specified bits.

**V-Flag:** For addition, subtraction, and 2's complementation, the V-flag is always set to 1 if the result exceeds the range of 16-bit signed numbers for word operations ( $-8000_H$  to  $+7FFF_H$ ), or 8-bit signed numbers for byte operations ( $-80_H$  to  $+7F_H$ ). Otherwise, the V-flag is cleared. Note that the result of an integer addition, integer subtraction, or 2's complement is not valid if the V-flag indicates an arithmetic overflow.

For multiplication and division, the V-flag is set to 1 if the result cannot be represented in a word data type; otherwise, it is cleared. Note that a division by zero will always cause an overflow. In contrast to the result of a division, the result of a multiplication is valid whether or not the V-flag is set to 1.

Because logical ALU operations cannot produce an invalid result, the V-flag is cleared by these operations.

The V-flag is also used as a 'Sticky Bit' for rotate right and shift right operations. With only using the C-flag, a rounding error caused by a shift right operation can be estimated up to a quantity of one half of the LSB of the result. In conjunction with the V-flag, the C-flag allows evaluation of the rounding error with a finer resolution (see [Table 5-22](#)).

**Central Processing Unit (CPU)**

For Boolean bit operations with only one operand, the V-flag is always cleared. For Boolean bit operations with two operands, the V-flag represents the logical ORing of the two specified bits.

**Table 5-22 Shift Right Rounding Error Evaluation**

<b>C-Flag</b>	<b>V-Flag</b>	<b>Rounding Error Quantity</b>
0	0	No rounding error
0	1	$0 < \text{Rounding error} < \frac{1}{2} \text{ LSB}$
1	0	$\text{Rounding error} = \frac{1}{2} \text{ LSB}$
1	1	$\text{Rounding error} > \frac{1}{2} \text{ LSB}$

**Z-Flag:** The Z-flag is normally set to 1 if the result of an ALU operation equals zero, otherwise it is cleared.

For the addition and subtraction with carry, the Z-flag is only set to 1, if the Z-flag already contains a 1 and the result of the current ALU operation also equals zero. This mechanism is provided to support multiple precision calculations.

For Boolean bit operations with only one operand, the Z-flag represents the logical negation of the previous state of the specified bit. For Boolean bit operations with two operands, the Z-flag represents the logical NORing of the two specified bits. For the prioritize ALU operation, the Z-flag indicates whether the second operand was zero.

**E-Flag:** End of table flag. The E-flag can be altered by instructions which perform ALU or data movement operations. The E-flag is cleared by those instructions which cannot be reasonably used for table search operations. In all other cases, the E-flag value depends on the value of the source operand to signify whether the end of a search table is reached or not. If the value of the source operand of an instruction equals the lowest negative number which is representable by the data format of the corresponding instruction ( $8000_H$  for the word data type, or  $80_H$  for the byte data type), the E-flag is set to 1; otherwise, it is cleared.

**General Control Functions (USR0, USR1, BANK, HLDEN)**

A few bits in register PSW are dedicated to general control functions. Thus, they are saved and restored automatically upon task switches and interrupts.

**USR0/USR1-Flags:** These bits can be set automatically during the execution of repeated MAC instructions. These bits can also be used as general flags by an application.

**BANK:** Bitfield BANK selects the currently active register bank (local or global). Bitfield BANK is updated implicitly by hardware upon entering an interrupt service routine, and by a RETI instruction. It can be also modified explicitly via software by any instruction which can write to PSW.

## Central Processing Unit (CPU)

**HLDEN:** Setting this bit for the first time activates the selected bus arbitration mode. Bus arbitration can be disabled by temporarily clearing bit HLDEN. In this case the bus is locked, while the bus arbitration mode remains selected. Please refer to the External Bus Controller (EBC) chapter for functional details. Note that the HLDEN bit can be accessed only when memory protection (MPU) is disabled.

### Protection Level (PL0, PL1)

These flags specify the current protection level of the system. This information is needed for systems implementing memory protection (i.e. MPU). Four different protection levels are defined according to the table below. Refer to the Memory Protection (MPU) chapter for more information on how the protection system works.

**Table 5-23 Decoding of Protection Level**

PL1	PL0	Protection Level
0	0	Protection Level 0
0	1	Protection Level 1
1	0	Protection Level 2
1	1	Protection Level 3

A write into bit PSW.10 will be interpreted as a write into PL1 when the MPU is enabled or as a write into HLDEN when the MPU is disabled. Considering this fact it is possible to use both the EBC arbitration and MPU functionalities. Note that software made to support the external master functionality, i.e. trying to write into this HLDEN bit, may not have write permission when the MPU is enabled unless it runs in privileged mode.

### CPU Interrupt Status (IEN, ILVL)

**IEN:** The Interrupt Enable bit allows interrupts to be globally enabled (IEN = 1) or disabled (IEN = 0).

**ILVL:** The four-bit Interrupt Level field (ILVL) specifies the priority of the current CPU activity. The interrupt level is updated by hardware on entry into an interrupt service routine, but it can also be modified via software to prevent other interrupts from being acknowledged. If an interrupt level 15 has been assigned to the CPU, it has the highest possible priority; thus, the current CPU operation cannot be interrupted except by hardware traps or external non-maskable interrupts.

After reset, all interrupts are globally disabled, and the lowest priority (ILVL = 0) is assigned to the initial CPU activity.

### 5.8.1 16-bit Adder/Subtractor, Barrel Shifter, and 16-bit Logic Unit

All standard arithmetic and logical operations are performed by the 16-bit ALU. In case of byte operations, signals from bits 6 and 7 of the ALU result are used to control the

condition flags. Multiple precision arithmetic is supported by a “CARRY-IN” signal to the ALU from previously calculated portions of the desired operation.

A 16-bit barrel shifter provides multiple bit shifts in a single cycle. Rotations and arithmetic shifts are also supported.

## **5.8.2 Bit Manipulation Unit**

The XE16xyM offers a large number of instructions for bit processing. These instructions are typically used to -

- manipulate software bit flags within CPU registers, GPRs or DPRAM
- control on-chip +Bus peripherals and port logic via control bits of their respective bit addressable (E)SFRs.

The bit manipulation instructions allow short addressing mode with bitoff operands only (see [Chapter 5.7.1](#)).

*Note: All GPRs are bit-addressable independently from the allocation of the register bank via the Context Pointer (CP). Even GPRs which are allocated to non-bit-addressable RAM locations provide this feature.*

Instructions BSET, BCLR, BAND, BOR, BXOR, BMOV, BMOVN explicitly set or clear specific bits. The bitfield instructions BFLDL and BFLDH allow masked manipulation of up to 8 bits of a specific byte at one time. The instructions JBC and JNBS implicitly clear or set the specified bit when the jump is taken. The instructions JB and JNB evaluate the specified bit to determine if the jump is to be taken.

*Note: Bit operations on undefined bit locations will always read a bit value of ‘0’, while the write access will not affect the respective bit location.*

### **Bit protection using mask protected write**

Instructions that manipulate single bits or bit groups either use a read-modify-write sequence or mask protected write to execute the operation.

The read-modify-write sequence accesses the whole word containing the specified bit(s). The read-modify-write approach may be critical with hardware affected bits of type ‘rwh’ or ‘wh’. In these cases, the hardware may change other bits of the register while the read-modify-write operation is in progress. Thus the writeback could overwrite the new bit value generated by the hardware.

To handle this side effect operations on **bit addressable (E)SFR registers** support the bit protection mechanism using a mask protected write.

Example:

```
BCLR          EOPIC.EOPIE      ; disable 'end of PEC' interrupts
```

**Central Processing Unit (CPU)**

The instruction will clear the interrupt enable bit EOPIE while the 'rwh' bit EOPIR will be mask protected. This ensures that an EOP interrupt occurring exactly at the same time will be correctly flagged.

*Note: For the BFLD(LH) instructions the protection mask must be supplied by the programmer.*

*Note: If a direct conflict occurs between a bit manipulation generated by hardware and an intended software access on the **same** bit, the software access has priority and determines the final value of the respective bit.*

### 5.8.3 Multiply and Divide Unit

The XE16xyM's multiply and divide unit has two separated parts. One is the fast  $16 \times 16$ -bit multiplier that executes a multiplication in one CPU cycle. The other one is a division sub-unit which performs the division algorithm in 18 ... 21 CPU cycles (depending on the data and division types). The divide instruction requires four CPU cycles to be executed. For performance reasons, the rest of the division algorithm runs in the background during the following seventeen CPU cycles, while further instructions are executed in parallel. Interrupt tasks can also be started and executed immediately without any delay. If an instruction (from the original instruction stream or from the interrupt task) tries to use the unit while a division is still running, the execution of this new instruction is stalled until the previous division is finished.

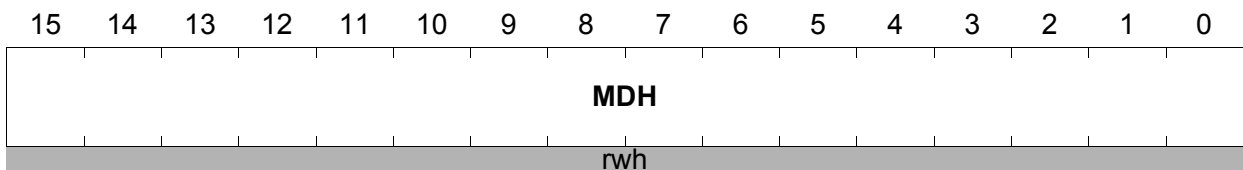
To avoid these stalls, the multiply and division unit should not be used during the first fourteen CPU cycles of the interrupt tasks. For example, this requires up to fourteen one-cycle instructions to be executed between the interrupt entry and the first instruction which uses the multiply and divide unit again (worst case).

Multiplications and divisions implicitly use the 32-bit multiply/divide register MD (represented by the concatenation of the two non-bit-addressable data registers MDH and MDL) and the associated control register MDC. This bit-addressable 16-bit register is implicitly used by the CPU when it performs a division or multiplication in the ALU.

After a multiplication, MD represents the 32-bit result. For long divisions, MD must be loaded with the 32-bit dividend before the division is started. After any division, register MDH represents the 16-bit remainder, register MDL represents the 16-bit quotient.

#### MDH

**Multiply Divide High Word**      **SFR (FE0C<sub>H</sub>/06<sub>H</sub>)**      **Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
MDH	[15:0]	rwh	<b>High Part of MD</b> The high order sixteen bits of the 32-bit multiply and divide register MD.

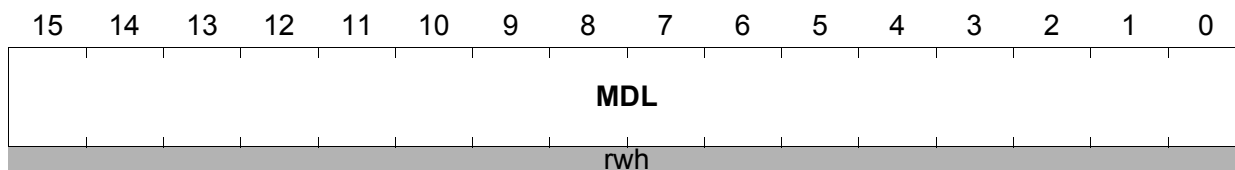
**Central Processing Unit (CPU)**

**MDL**

**Multiply Divide Low Word**

**SFR (FE0E<sub>H</sub>/07<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
MDL	[15:0]	rwh	<b>Low Part of MD</b> The low order sixteen bits of the 32-bit multiply and divide register MD.

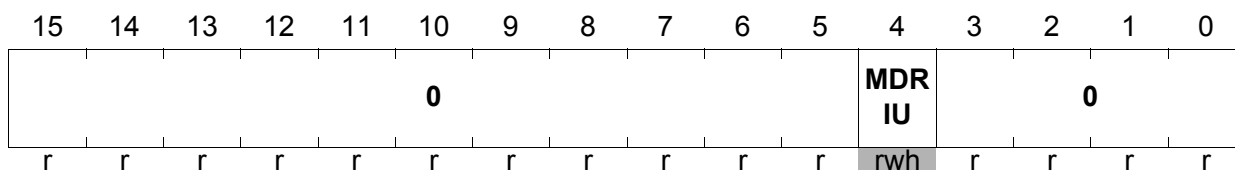
Whenever MDH or MDL is updated via software, the Multiply/Divide Register In Use flag (MDRIU) in the Multiply/Divide Control register (MDC) is set to '1'. The MDRIU flag is cleared, whenever register MDL is read via software.

**MDC**

**Multiply Divide Control**

**SFR (FF0E<sub>H</sub>/87<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
MDRIU	4	rwh	<b>Multiply/Divide Register In Use</b> 0 <sub>B</sub> Cleared when MDL is read via software. 1 <sub>B</sub> Set when MDL or MDH is written via software, or when a multiply or divide instruction is executed.
0	[15:5], [3:0]	r	<b>Reserved</b> Read as 0, should be written 0

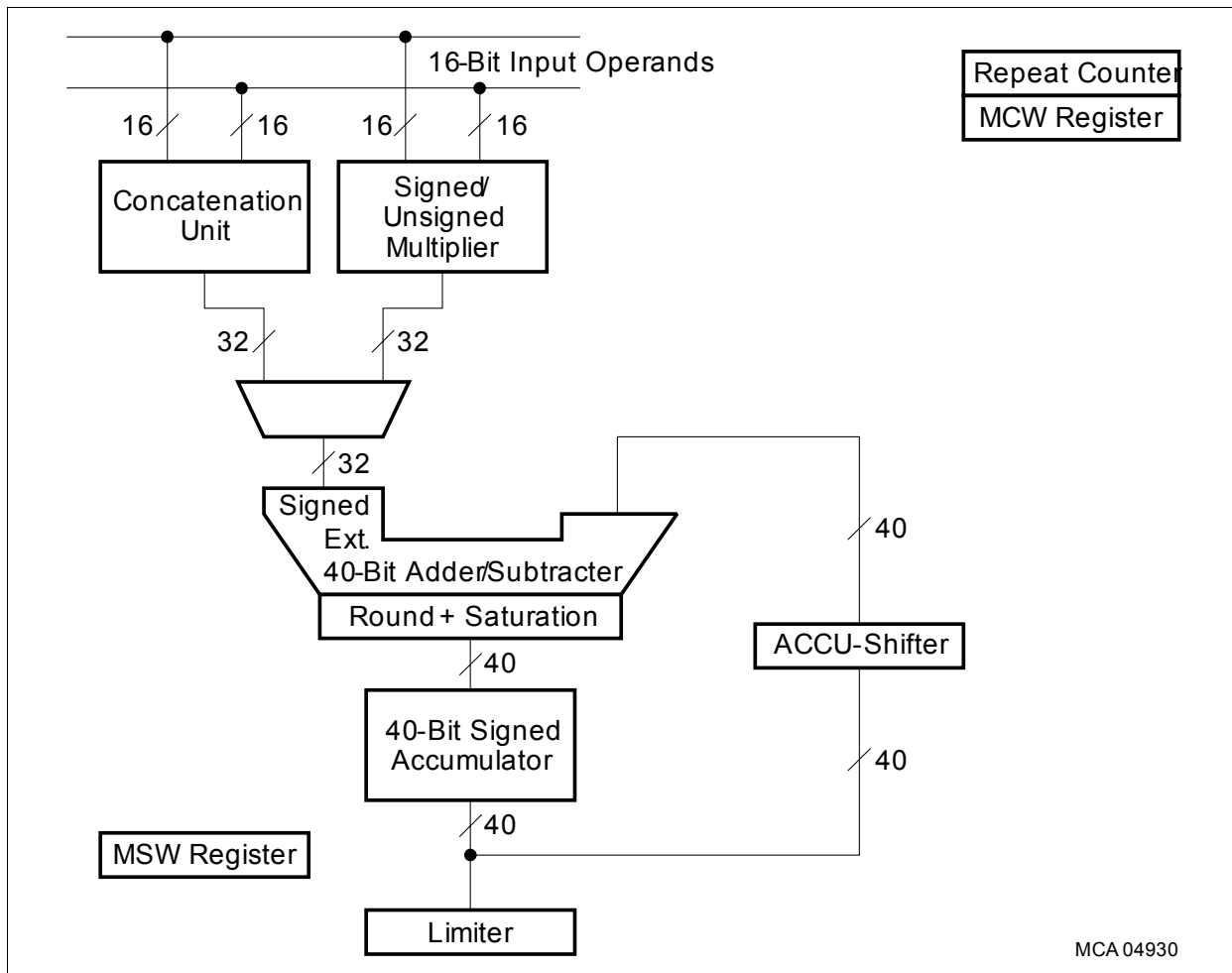
*Note: The MDRIU flag indicates the usage of register MD (MDL and MDH). In this case MD must be saved prior to a new multiplication or division operation.*



## 5.9 DSP Data Processing (MAC Unit)

The CoXXX arithmetic instructions are executed by the MAC unit. It provides single-instruction-cycle, non-pipelined, 32-bit additions; 32-bit subtraction; right and left shifts; 16-bit by 16-bit multiplication; and multiplication with cumulative subtraction/addition. The MAC unit includes the following major components also shown in [Figure 5-16](#):

- 16-bit by 16-bit signed/unsigned multiplier with signed result<sup>1)</sup>
- Concatenation Unit
- Scaler (one-bit left shifter) for fractional computing
- 40-bit Adder/Subtractor
- 40-bit Signed Accumulator
- Data Limiter
- Accumulator Shifter
- Repeat Counter



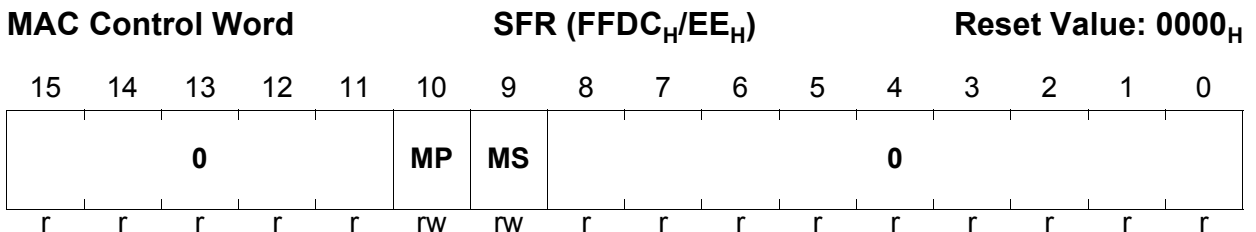
**Figure 5-16 Functional MAC Unit Block Diagram**

1) The same hardware-multiplier is used in the ALU.

### 5.9.1 MAC Unit Control

The working register of the MAC unit is a dedicated 40-bit accumulator register. A set of consistent flags is automatically updated in status register MSW after each MAC operation. These flags allow branching on specific conditions. Unlike the PSW flags, these flags are not preserved automatically by the CPU upon entry into an interrupt or trap routine. All dedicated MAC registers must be saved on the stack if the MAC unit is shared between different tasks and interrupts. General properties of the MAC unit are selected via the MAC control word MCW.

#### MCW



Field	Bits	Type	Description
MP	10	rw	<b>One-Bit Scaler Control</b> 0 <sub>B</sub> Multiplier product shift disabled 1 <sub>B</sub> Multiplier product shift enabled for signed multiplications
MS	9	rw	<b>Saturation Control</b> 0 <sub>B</sub> Saturation disabled 1 <sub>B</sub> Saturation to 32-bit value enabled
0	[15:11] , [8:0]	r	<b>Reserved</b> Read as 0, should be written 0

### 5.9.2 Representation of Numbers and Rounding

The XE16xyM supports the 2's complement representation of binary numbers. In this format, the sign bit is the MSB of the binary word. This is set to zero for positive numbers and set to one for negative numbers. Unsigned numbers are supported only by multiply/multiply-accumulate instructions which specify whether each operand is signed or unsigned.

In 2's complement fractional format, the N-bit operand is represented using the 1.[N-1] format (1 signed bit, N-1 fractional bits). Such a format can represent numbers between -1 and +1 - 2<sup>[-N-1]</sup>. This format is supported when bit MP of register MCW is set.

The XE16xyM implements 2's complement rounding. With this rounding type, one is added to the bit to the right of the rounding point (bit 15 of MAL), before truncation (MAL is cleared).

### **5.9.3 The 16-bit by 16-bit Signed/Unsigned Multiplier and Scaler**

The multiplier executes 16-bit by 16-bit parallel signed/unsigned fractional and integer multiplication in one CPU-cycle. The multiplier allows the multiplication of unsigned and signed operands. The result is always presented in a signed fractional or integer format. The result of the multiplication feeds a one-bit scaler to allow compensation for the extra sign bit gained in multiplying two 16-bit 2's complement numbers.

### **5.9.4 Concatenation Unit**

The concatenation unit enables the MAC unit to perform 32-bit arithmetic operations in one CPU cycle. The concatenation unit concatenates two 16-bit operands to a 32-bit operand before the 32-bit arithmetic operation is executed in the 40-bit adder/subtractor. The second required operand is always the current accumulator contents. The concatenation unit is also used to pre-load the accumulator with a 32-bit value.

### **5.9.5 One-bit Scaler**

The one-bit scaler can shift the result of the concatenation unit or the output of the multiplier one bit to the left. The scaler is controlled by the executed instruction for the concatenation or by control bit MP in register MCW.

If bit MP is set the product is shifted one bit to the left to compensate for the extra sign bit gained in multiplying two 16-bit 2's-complement numbers. The enabled automatic shift is performed only if both input operands are signed.

### **5.9.6 The 40-bit Adder/Subtractor**

The 40-bit Adder/Subtractor allows intermediate overflows in a series of multiply/accumulate operations. The Adder/Subtractor has two input ports. The 40-bit port is the feedback of the accumulator output through the ACCU-Shifter to the Adder/Subtractor. The 32-bit port is the input port for the operand coming from the one-bit Scaler. The 32-bit operands are signed and extended to 40 bits before the addition/subtraction is performed.

The output of the Adder/Subtractor goes to the accumulator. It is also possible to round the result and to saturate it on a 32-bit value automatically after every accumulation. The round operation is performed by adding  $00'0000'8000_H$  to the result. Automatic saturation is enabled by setting the saturation control bit MS in register MCW.

When the accumulator is in the overflow saturation mode and an overflow occurs, the accumulator is loaded with either the most positive or the most negative value

representable in a 32-bit value, depending on the direction of the overflow as well as on the arithmetic used. The value of the accumulator upon saturation is either 00'7FFF'FFFF<sub>H</sub> (positive) or FF'8000'0000<sub>H</sub> (negative).

### 5.9.7 The Data Limiter

Saturation arithmetic is also provided to selectively limit overflow when reading the accumulator by means of a **CoSTORE <destination>., MAS** instruction. Limiting is performed on the MAC-Unit accumulator. If the contents of the accumulator can be represented in the destination operand size without overflow, then the data limiter is disabled and the operand is not modified. If the contents of the accumulator cannot be represented without overflow in the destination operand size, the limiter will substitute a "limited" data as explained in [Table 5-24](#):

**Table 5-24 Limiter Output**

ME-flag	MN-flag	Output of Limiter
0	x	unchanged
1	0	7FFF <sub>H</sub>
1	1	8000 <sub>H</sub>

*Note: In this particular case, both the accumulator and the status register are not affected. MAS is readable by means of a CoSTORE instruction only.*

### 5.9.8 The Accumulator Shifter

The accumulator shifter is a parallel shifter with a 40-bit input and a 40-bit output. The source accumulator shifting operations are:

- No shift (Unmodified)
- Up to 16-bit Arithmetic Left Shift
- Up to 16-bit Arithmetic Right Shift

Notice that bits ME, MSV, and MSL in register MSW are affected by left shifts; therefore, if the saturation mechanism is enabled (MS) the behavior is similar to the one of the Adder/Subtractor.

*Note: Certain precautions are required in case of left shift with saturation enabled. Generally, if MAE contains significant bits, then the 32-bit value in the accumulator is to be saturated. However, it is possible that left shift may move some significant bits out of the accumulator. The 40-bit result will be misinterpreted and will be either not saturated or saturated incorrectly. There is a chance that the result of left shift may produce a result which can saturate an original positive number to the minimum negative value, or vice versa.*

### 5.9.9 The 40-bit Signed Accumulator Register

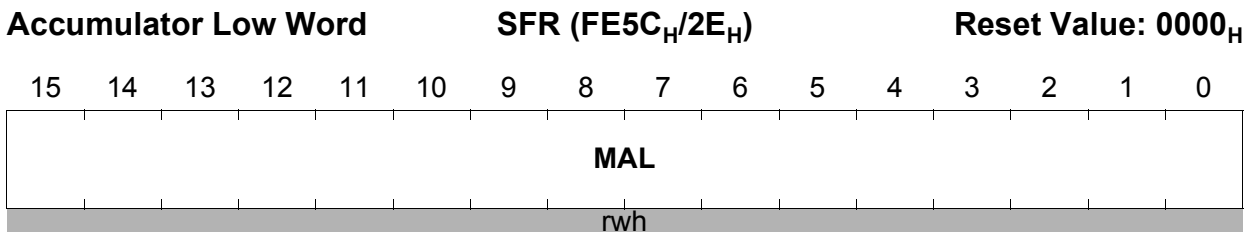
The 40-bit accumulator consists of three concatenated registers MAE, MAH, and MAL. MAE is 8 bits wide, MAH and MAL are 16 bits wide. MAE is the Most Significant Byte of the 40-bit accumulator. This byte performs a guarding function. MAE is accessed as the lower byte of register MSW.

When MAH is written, the value in the accumulator is automatically adjusted to signed extended 40-bit format. That means MAL is cleared and MAE will be automatically loaded with zeros for a positive number (the most significant bit of MAH is 0), and with ones for a negative number (the most significant bit of MAH is 1), representing the extended 40-bit negative number in 2's complement notation. One may see that the extended 40-bit value is equal to the 32-bit value without extension. In other words, after this extension, MAE does not contain significant bits. Generally, this condition is present when the highest 9 bits of the 40-bit signed result are the same.

During the accumulator operations, an overflow may happen and the result may not fit into 32 bits and MAE will change. The extension flag "E" in register MSW is set when the signed result in the accumulator has exceeded the 32-bit boundary. This condition is present when the highest 9 bits of the 40-bit signed result are not the same, i.e. MAE contains significant bits.

Most CoXXX operations specify the 40-bit accumulator register as a source and/or a destination operand.

#### MAL



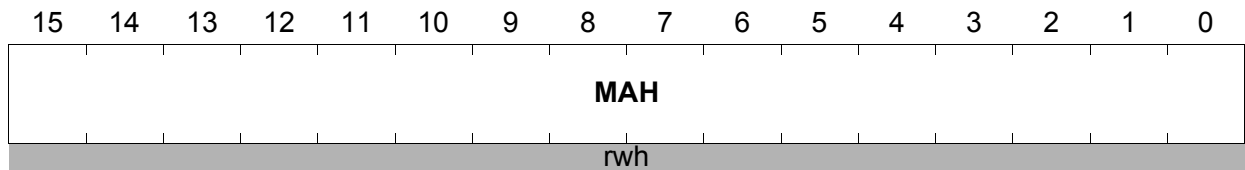
Field	Bits	Type	Description
MAL	[15:0]	rwh	<b>Low Part of Accumulator</b> The 40-bit accumulator is completed by the accumulator high word (MAH) and bitfield MAE

**MAH**

**Accumulator High Word**

**SFR (FE5E<sub>H</sub>/2F<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
<b>MAH</b>	[15:0]	rwh	<b>High Part of Accumulator</b> The 40-bit accumulator is completed by the accumulator low word (MAL) and bitfield MAE

### 5.9.10 The MAC Unit Status Word MSW

The upper byte of register MSW (bit-addressable) shows the current status of the MAC Unit. The lower byte of register MSW represents the 8-bit MAC accumulator extension, building the 40-bit accumulator together with registers MAH and MAL.

#### MSW

#### MAC Status Word

SFR (FFDE<sub>H</sub>/EF<sub>H</sub>)

Reset Value: 0000<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	MV	MSL	ME	MSV	MC	MZ	MN	MAE							
r	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh							

Field	Bits	Type	Description
0	15	r	<b>Reserved</b> Read as 0, should be written 0
MV	14	rwh	<b>Overflow Flag</b> 0 <sub>B</sub> No Overflow produced 1 <sub>B</sub> Overflow produced
MSL	13	rwh	<b>Sticky Limit Flag</b> 0 <sub>B</sub> Result was not saturated 1 <sub>B</sub> Result was saturated
ME	12	rwh	<b>MAC Extension Flag</b> 0 <sub>B</sub> MAE does not contain significant bits 1 <sub>B</sub> MAE contains significant bits
MSV	11	rwh	<b>Sticky Overflow Flag</b> 0 <sub>B</sub> No Overflow occurred 1 <sub>B</sub> Overflow occurred
MC	10	rwh	<b>Carry Flag</b> 0 <sub>B</sub> No carry/borrow produced 1 <sub>B</sub> Carry/borrow produced
MZ	9	rwh	<b>Zero Flag</b> 0 <sub>B</sub> MAC result is not zero 1 <sub>B</sub> MAC result is zero
MN	8	rwh	<b>Negative Result</b> 0 <sub>B</sub> MAC result is positive 1 <sub>B</sub> MAC result is negative

Field	Bits	Type	Description
<b>MAE</b>	[7:0]	rwh	<b>MAC Accumulator Extension</b> The most significant bits of the 40-bit accumulator, completing registers MAH and MAL

### **MAC Unit Status (MV, MN, MZ, MC, MSV, ME, MSL)**

These condition flags indicate the MAC status resulting from the most recently performed MAC operation. These flags are controlled by the majority of MAC instructions according to specific rules. Those rules depend on the instruction managing the MAC or data movement operation.

After execution of an instruction which explicitly updates register MSW, the condition flags may no longer represent an actual MAC status. An explicit write operation to register MSW supersedes the condition flag values implicitly generated by the MAC unit. An explicit read access returns the value of register MSW after execution of the immediately preceding instruction. Register MSW can be accessed via any instruction capable of accessing an SFR.

*Note: After reset, all MAC status bits are cleared.*

**MN-Flag:** For the majority of the MAC operations, the MN-flag is set to 1 if the most significant bit of the result contains a 1; otherwise, it is cleared. In the case of integer operations, the MN-flag can be interpreted as the sign bit of the result (negative: MN = 1, positive: MN = 0). Negative numbers are always represented as the 2's complement of the corresponding positive number. The range of signed numbers extends from 80'0000'0000<sub>H</sub> to 7F'FFFF'FFFF<sub>H</sub>.

**MZ-Flag:** The MZ-flag is normally set to 1 if the result of a MAC operation equals zero; otherwise, it is cleared.

**MC-Flag:** After a MAC addition, the MC-flag indicates that a "Carry" from the most significant bit of the accumulator extension MAE has been generated. After a MAC subtraction or a MAC comparison, the MC-flag indicates a "Borrow" representing the logical negation of a "Carry" for the addition. This means that the MC-flag is set to 1 if **no** "Carry" from the most significant bit of the accumulator has been generated during a subtraction. Subtraction is performed by the MAC Unit as a 2's complement addition and the MC-flag is cleared when this complement addition caused a "Carry".

For left-shift MAC operations, the MC-flag represents the value of the bit shifted out last. Right-shift MAC operations always clear the MC-flag. The arithmetic right-shift MAC operation can set the MC-flag if the enabled round operation generates a "Carry" from the most significant bit of the accumulator extension MAE.

**MSV-Flag:** The addition, subtraction, 2's complement, and round operations always set the MSV-flag to 1 if the MAC result exceeds the maximum range of 40-bit signed numbers. If the MSV-flag indicates an arithmetic overflow, the MAC result of an operation is not valid.



**Central Processing Unit (CPU)**

The MSV-flag is a 'Sticky Bit'. Once set, other MAC operations cannot affect the status of the MSV-flag. Only a direct write operation can clear the MSV-flag.

**ME-Flag:** The ME-flag is set if the accumulator extension MAE contains significant bits, that means if the nine highest accumulator bits are not all equal.

**MSL-Flag:** The MSL-flag is set if an automatic saturation of the accumulator has happened. The automatic saturation is enabled if bit MS in register MCW is set. The MSL-Flag can be also set by instructions which limit the contents of the accumulator. If the accumulator has been limited, the MSL-Flag is set.

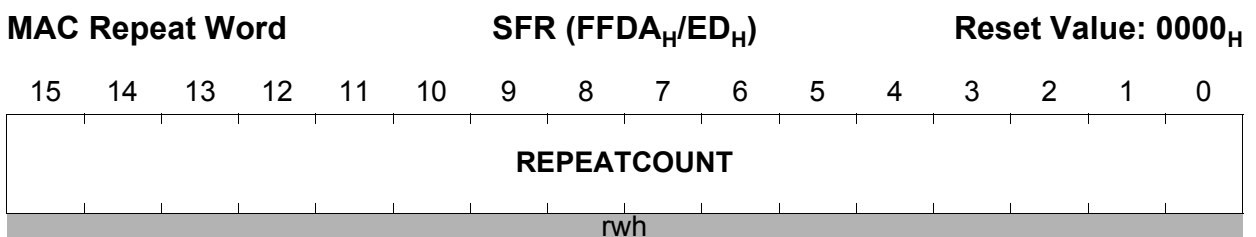
The MSL-Flag is a 'Sticky Bit'. Once set, it cannot be affected by the other MAC operations. Only a direct write operation can clear the MSL-flag.

**MV-Flag:** The addition, subtraction, and accumulation operations set the MV-flag to 1 if the result exceeds the maximum range of signed numbers (80'0000'0000<sub>H</sub> to 7F'FFFF'FFFF<sub>H</sub>); otherwise, the MV-flag is cleared. Note that if the MV-flag indicates an arithmetic overflow, the result of the integer addition, integer subtraction, or accumulation is not valid.

### 5.9.11 The Repeat Counter MRW

The Repeat Counter MRW controls the number of repetitions a loop must be executed. The register must be pre-loaded before it can be used with -USRx CoXXX operations. MAC operations are able to decrement this counter. When a -USRx CoXXX instruction is executed, MRW is checked for zero **before** being decremented. If MRW equals zero, bit USRx is set and MRW is not further decremented. Register **MRW** can be accessed via any instruction capable of accessing a SFR.

#### MRW



Field	Bits	Type	Description
REPEATCOUNT	[15:0]	rwh	MAC repeat counter

All CoXXX instructions have a 3-bit wide repeat control field 'rrr' (bit positions [31:29]) in the operand field to control the MRW repeat counter. [Table 5-25](#) lists the possible encodings.

**Table 5-25    Encoding of MAC Repeat Word Control**

<b>Code in 'rrr'</b>	<b>Effect on Repeat Counter</b>
000 <sub>B</sub>	regular CoXXX instruction
001 <sub>B</sub>	RESERVED
010 <sub>B</sub>	'-USR0 CoXXX' instruction, decrements repeat counter and sets bit USR0 if MRW is zero
011 <sub>B</sub>	'-USR1 CoXXX' instruction, decrements repeat counter and sets bit USR1 if MRW is zero
1XX <sub>B</sub>	RESERVED

*Note: Bit USR0 has been a general purpose flag also in previous architectures. To prevent collisions due to using this flag by programmer or compiler, use '-USR0 C0XXX' instructions very carefully.*

The following example shows a loop which is executed 20 times. Every time the CoMACM instruction is executed, the MRW counter is decremented.

```

                MOV      MRW, #19                ;Pre-load loop counter
loop01:
-USR1          CoMACM [IDX0+], [R0+]            ;Calculate and decrement MSW
                ADD      R2, #0002H
                JMPA     cc_nusr1, loop01 ;Repeat loop until USR1 is set

```

*Note: Because correctly predicted JMPA is executed in 0-cycle, it offers the functionality of a repeat instruction.*

## 5.10 Constant Registers

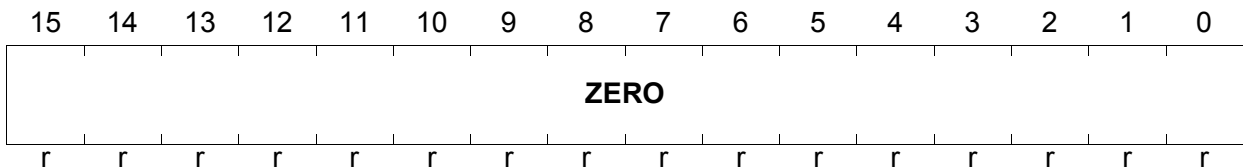
All bits of these bit-addressable registers are fixed to 0 or 1 by hardware. These registers can be read only. Register ZEROS/ONES can be used as a register-addressable constant of all zeros or all ones, for example for bit manipulation or mask generation. The constant registers can be accessed via any instruction capable of addressing an SFR.

### ZEROS

**Zeros Register**

**SFR (FF1C<sub>H</sub>/8E<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



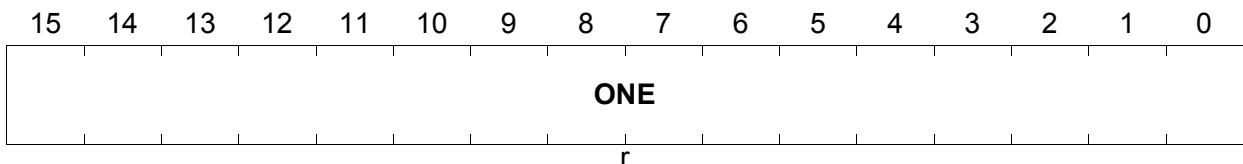
Field	Bits	Type	Description
ZERO	[15:0]	r	Constant Zero Bits

### ONES

**Ones Register**

**SFR (FF1E<sub>H</sub>/8F<sub>H</sub>)**

**Reset Value: FFFF<sub>H</sub>**



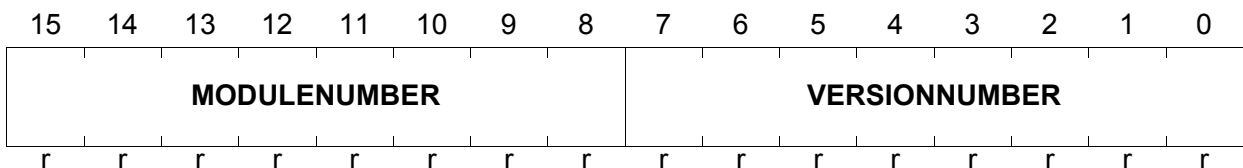
Field	Bits	Type	Description
ONE	[15:0]	r	Constant One Bits

### CPUID

**CPU Identification Register**

**ESFR (F00C<sub>H</sub>/06<sub>H</sub>)**

**Reset Value: 0313<sub>H</sub>**



Field	Bits	Type	Description
MODULENUMBER	[15:8]	r	C166 Family CPU Module Number (C166S-V2)

Field	Bits	Type	Description
VERSIONNUMBER	[7:0]	r	C166S-V2 CPU Version Number

## 6 Memory Protection Unit (MPU)

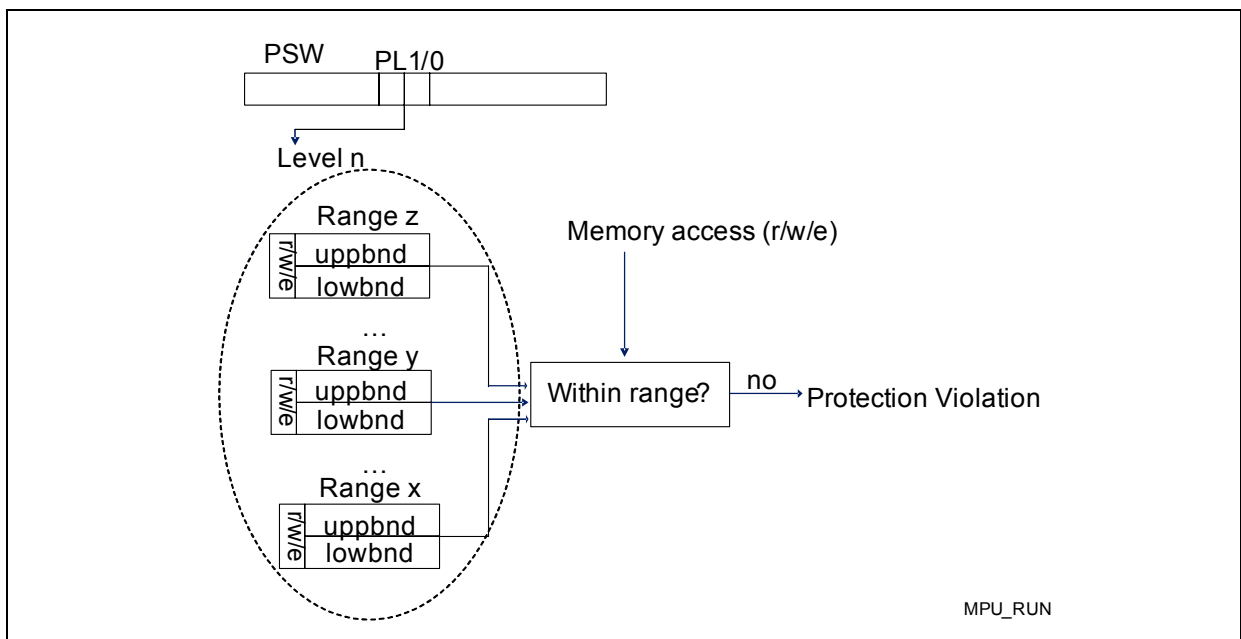
The Memory Protection Unit (MPU) provides the hardware mechanisms needed for implementing memory protection. The MPU allows detection of unauthorized accesses (read, write or instruction fetch) in user-defined memory ranges. It offers protection for the complete address space, including the peripheral area.

The MPU can be used to support the encapsulation of different applications or software components running on the processor. This encapsulation provides the means to ensure integrity and fault isolation capabilities in today's complex systems relying on multiple-sources software.

### 6.1 Functional Overview

Different protection levels are usually needed to support a programming system where for example an operating system or software kernel runs and controls different application and low level drivers parts. One level can be associated to the operating system and for the other tasks that need protection against each other or against the operating system, other levels can be used. For every protection level different address ranges with different access permissions for instructions and/or data can be defined. When a piece of code is executed and the memory protection is enabled, the permissions associated to its protection level are selected and every time a memory access is performed it will be checked if the access is outside of the specified ranges or violates the access permissions. In this case the access may not be performed but marked as invalid and a protection trap routine can be executed.

The basic MPU functionality is shown in **Figure 6-1**.



**Figure 6-1 MPU Operation**

## **Memory Protection Unit (MPU)**

Four Protection Levels can coexist during run time in this architecture. Two bits in the Processor Status Word (PSW) are used to select which protection level is active at a given time. If an application requires more than 4 protection levels, a re-mapping of all the levels to the 4 possible values has to be performed and during run time re-programming of the protection register sets when switching levels is needed.

A protection register set is associated to every protection level, every set contains all the address ranges and the access permissions associated to the corresponding protection level. Every protection register set can contain a programmable number of range registers. All together, a maximum of 12 ranges is supported. Associated to every code or data range, a protection mode register defines the permissions for this range. Refer to the next chapters for a detailed explanation of the MPU registers needed for the protection system and its usage.

## 6.2 Memory Protection Registers

A protection register set consists of a variable number of Protection Range register pairs (PRUx/PRLx) and the corresponding number of Protection Mode registers (PMx). The PMx registers are located in the SFRs area and are accessed through the Peripheral Data Bus -PD-Bus-. The PRUx/PRLx registers are not memory mapped, their access mechanism is supported through the memory mapped registers Protection Range Address register (PRA) and Protection Range Data register (PRD).

**Table 6-1 Registers Address Space**

Module	Base Address	End Address	Note
MPU	0 <sub>H</sub>		

**Table 6-2 Registers Overview**

Register Short Name	Register Long Name	Offset Address	Page Number
PRUx (x =0-11)	Protection Range Register x Upper Bound	none <sub>H</sub>	<a href="#">6-4</a>
PRLx (x =0-11)	Protection Range Register x Lower Bound	none <sub>H</sub>	<a href="#">6-4</a>
PM5	Protection Mode Register 5	FFD2 <sub>H</sub>	<a href="#">6-7</a>
PM4	Protection Mode Register 4	FFD0 <sub>H</sub>	<a href="#">6-7</a>
PM3	Protection Mode Register 3	FFCE <sub>H</sub>	<a href="#">6-7</a>
PM2	Protection Mode Register 2	FFCC <sub>H</sub>	<a href="#">6-7</a>
PM1	Protection Mode Register 1	FFCA <sub>H</sub>	<a href="#">6-7</a>
PM0	Protection Mode Register 0	FFC8 <sub>H</sub>	<a href="#">6-7</a>
PRD	Protection Range Data	FFC6 <sub>H</sub>	<a href="#">6-8</a>
PRA	Protection Range Address	FFC4 <sub>H</sub>	<a href="#">6-9</a>

### 6.2.1 Protection Range Registers

The PRUx/PRLx pairs are 16-bits registers and specify the upper 16 bits of the physical addresses, upper and lower bound, for data and/or code for all the allowed ranges (12 is the maximum supported). Only these upper 16 bits of the physical addresses are considered in the address comparisons, as a consequence, the minimum granularity of the ranges is 256 bytes and all the ranges are aligned to this size.

The PRUx and PRLx registers specify respectively the Upper and Lower addresses of a Range. If due to a programming error PRLx specifies a value bigger than PRUx, the corresponding range will not specify a correct address range, and as a consequence the corresponding range is useless (i.e. ignored). Note that due to the 256 byte range

## Memory Protection Unit (MPU)

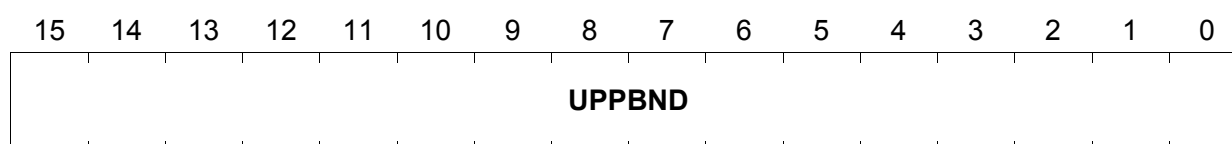
granularity, to program a range of this smallest size, both upper and lower range need to be program with the same value, i.e. the 16-bits base address of the range.

For programming a protection range in the PRUx/PRLx registers, it has to be selected first which range is going to be written by programming the address into PRA, then the data write operation can be performed by writing the data into PRD. In a similar way, a read operation has to be performed by selecting first which range is going to be read (by programming the address into PRA) and then the read operation can be performed by reading PRD. Programming a PRUx/PRLx register requires then two write operations. Similarly, reading a PRUx/PRLx register requires also two operations (one write and one read). For continuous accesses and when using the auto increment feature only one initialization into PRA is needed, afterwards only the PRD register needs to be written/read every time. Registers PRD and PRA are described in [Chapter 6.2.3](#) and [Chapter 6.2.4](#) respectively.

### PRUx (x =0-11)

#### Protection Range Register x Upper Bound

**Reset Value: 0000<sub>H</sub>**

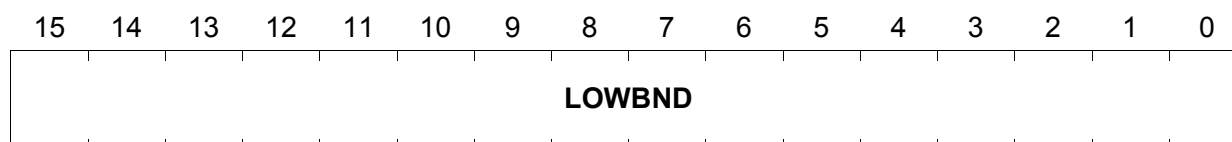


Field	Bits	Type	Description
UPPBND	[15:0]	rw	Upper Boundary Address (upper 16 bits)

### PRLx (x =0-11)

#### Protection Range Register x Lower Bound

**Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
LOWBND	[15:0]	rw	Lower Boundary Address (lower 16 bits)



## 6.2.2 Protection Mode Registers

All the control information associated to every address range is contained in the Protection Mode registers. Access permissions (execute, read and/or write) are defined here and also the range-to-level mapping. Every range can be individually enabled to be used for any protection level, even can be used for more than one level (but with the same access permissions). Also the field used to enable the protection system is implemented in one of the protection mode registers.

Note that no hardware mechanism is implemented to flush the pipeline upon a modification of these registers. This is usually not a problem because a (re-)programming of the MPU configuration registers should be anyhow performed having the protection disabled. Also the configuration affecting a particular protection level will be usually (re-)programmed from another level meaning that even at the point when protection is enabled the software currently running will not be affected by the configuration change (the configuration change is usually seen once the protection level is changed according to the procedure described in [Chapter 6.4.2](#)). For special cases where the change will and needs to be immediately seen, the software has to take care that the write is effective before executing the next affected instruction (by reading for example the latest written register).

The bit fields of the PMx registers in the description below use generic Range names (A, B), their mapping to the physical ranges is given after the PMx register name where they belong to. Given a Protection Mode register x, the range named A is addressing the physical range  $2 \cdot x$  and range named B the range  $2 \cdot x + 1$ .

The PMx registers are EINIT protected.

### PM0

#### Protection Mode Register 0

SFR (FFC8<sub>H</sub>)

Reset Value: 0000<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
L3E B	L2E B	L1E B	L0E B	WEB	REB	XEB	0	L3E A	L2E A	L1E A	L0E A	WEA	REA	XEA	PRO TEN
rw	rw	rw	rw	rw	rw	rw	r	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
PROTEN	0	rw	<b>Protection Enable bit</b> This bit enables the Protection mechanism 0 <sub>B</sub> Protection not enabled 1 <sub>B</sub> Protection enabled

**Memory Protection Unit (MPU)**

Field	Bits	Type	Description
<b>XEA, XEB</b>	1, 9	rw	<b>Execute Enable</b> 0 <sub>B</sub> Instruction fetch accesses to associated address range (A, B) not permitted 1 <sub>B</sub> Instruction fetch accesses to associated address range (A, B) permitted
<b>REA, REB</b>	2, 10	rw	<b>Read Enable</b> 0 <sub>B</sub> Data read accesses to associated address range (A, B) not permitted 1 <sub>B</sub> Data read accesses to associated address range (A, B) permitted
<b>WEA, WEB</b>	3, 11	rw	<b>Write Enable</b> 0 <sub>B</sub> Data write accesses to associated address range (A, B) not permitted 1 <sub>B</sub> Data write accesses to associated address range (A, B) permitted
<b>L0EA, L0EB</b>	4, 12	rw	<b>Level 0 Enable</b> 0 <sub>B</sub> Range (A, B) not enabled for Protection Level 0 1 <sub>B</sub> Range (A, B) enabled for Protection Level 0
<b>L1EA, L1EB</b>	5, 13	rw	<b>Level 1 Enable</b> 0 <sub>B</sub> Range (A, B) not enabled for Protection Level 1 1 <sub>B</sub> Range (A, B) enabled for Protection Level 1
<b>L2EA, L2EB</b>	6, 14	rw	<b>Level 2 Enable</b> 0 <sub>B</sub> Range (A, B) not enabled for Protection Level 2 1 <sub>B</sub> Range (A, B) enabled for Protection Level 2
<b>L3EA, L3EB</b>	7, 15	rw	<b>Level 3 Enable</b> 0 <sub>B</sub> Range (A, B) not enabled for Protection Level 3 1 <sub>B</sub> Range (A, B) enabled for Protection Level 3
<b>0</b>	8	r	<b>Reserved field</b>

The field PROTEN exists only in the Protection Mode Register 0.

**Memory Protection Unit (MPU)**

**PMx (x =1-5)**

**Protection Mode Register x      SFR (FFC8<sub>H</sub>+2\*x)      Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
L3E B	L2E B	L1E B	L0E B	WEB	REB	XEB	0	L3E A	L2E A	L1E A	L0E A	WEA	REA	XEA	0
rw	rw	rw	rw	rw	rw	rw	r	rw	rw	rw	rw	rw	rw	rw	r

Field	Bits	Type	Description
<b>0</b>	0	r	<b>Reserved field</b>
<b>XEA, XEB</b>	1, 9	rw	<b>Execute Enable</b> 0 <sub>B</sub> Instruction fetch accesses to associated address range (A, B) not permitted 1 <sub>B</sub> Instruction fetch accesses to associated address range (A, B) permitted
<b>REA, REB</b>	2, 10	rw	<b>Read Enable</b> 0 <sub>B</sub> Data read accesses to associated address range (A, B) not permitted 1 <sub>B</sub> Data read accesses to associated address range (A, B) permitted
<b>WEA, WEB</b>	3, 11	rw	<b>Write Enable</b> 0 <sub>B</sub> Data write accesses to associated address range (A, B) not permitted 1 <sub>B</sub> Data write accesses to associated address range (A, B) permitted
<b>L0EA, L0EB</b>	4, 12	rw	<b>Level 0 Enable</b> 0 <sub>B</sub> Range (A, B) not enabled for Protection Level 0 1 <sub>B</sub> Range (A, B) enabled for Protection Level 0
<b>L1EA, L1EB</b>	5, 13	rw	<b>Level 1 Enable</b> 0 <sub>B</sub> Range (A, B) not enabled for Protection Level 1 1 <sub>B</sub> Range (A, B) enabled for Protection Level 1
<b>L2EA, L2EB</b>	6, 14	rw	<b>Level 2 Enable</b> 0 <sub>B</sub> Range (A, B) not enabled for Protection Level 2 1 <sub>B</sub> Range (A, B) enabled for Protection Level 2

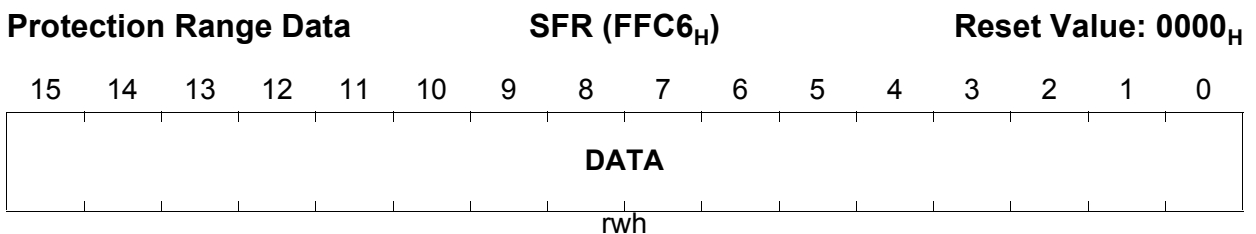
Field	Bits	Type	Description
<b>L3EA, L3EB</b>	7, 15	rw	<b>Level 3 Enable</b> 0 <sub>B</sub> Range (A, B) not enabled for Protection Level 3 1 <sub>B</sub> Range (A, B) enabled for Protection Level 3
<b>0</b>	8	r	<b>Reserved field</b>

### 6.2.3 Protection Range Data Register

The Protection Range Data register contains the 16 bits data value needed to program the content of the Protection Range Registers. It also contains the data read during the last read access on the Protection Range Registers. A write into PRD triggers immediately a write into the corresponding PRUx/PRLx register (the one that is currently selected by the write pointer -in PRA register-). Also a read into PRD delivers the corresponding PRUx/PRLx data immediately (the one that is currently selected by the read pointer -in PRA register-).

The PRD register is EINIT protected.

#### PRD



Field	Bits	Type	Description
<b>DATA</b>	[15:0]	rwh	<b>Data Value for/from PRUx/PRLx</b>

### 6.2.4 Protection Range Address Register

The Protection Range Address register contains two access pointers, one used for write operations and the other for read operations. With every 5-bit pointer it is possible to select a PRUx/PRLx register from a set of 24 register (the 24 PRUx/PRLx registers needed to implement 12 protection ranges).

An auto increment capability can be enabled for the access pointers (controlled by WMOD and RMOD fields), after every write or read into/from PRD the write or read pointers are incremented respectively. This feature enables a faster programming of the protection range registers. When the auto increment mode is active, the access pointers automatically do a wrap around (i.e. initialized to 0) after reaching its maximum value.

## Memory Protection Unit (MPU)

The occurrence of a wrap around is shown in the status bits WWA or RWA. The software can then check if this situation has happened taking the corresponding action and resetting the corresponding flag.

Special care has to be taken when programming the PRA register in order not to modify one of the pointers unintentionally. It is recommended to use bit instructions for that (bit field instructions for example). Also when using the auto increment feature and during debugging it has to be considered that a debugger access can also modify the pointer values, the debugger software should then take care of restoring the original status of this register.

The PRA register is EINIT protected.

### PRA

Protection Range Address					SFR (FFC4 <sub>H</sub> )					Reset Value: 0000 <sub>H</sub>					
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RMO D	RWA	0	RPTR				WM OD	WW A	0	WPTR					
rw	rwh	r	rwh				rw	rwh	r	rwh					

Field	Bits	Type	Description
<b>WPTR</b>	[4:0]	rwh	<b>Write Pointer</b> Selects the Protection Range Register to be written 00000 <sub>B</sub> Selects PRL0 00001 <sub>B</sub> Selects PRU0 00010 <sub>B</sub> Selects PRL1 00011 <sub>B</sub> Selects PRU1 00100 <sub>B</sub> Selects PRL2 00101 <sub>B</sub> Selects PRU2 ... 11110 <sub>B</sub> Selects PRL15 11111 <sub>B</sub> Selects PRU15
<b>0</b>	5	r	<b>Reserved field</b>
<b>WWA</b>	6	rwh	<b>Write Wrap Around Status</b> 0 <sub>B</sub> No WPTR Wrap Around occurred on last Write 1 <sub>B</sub> A WPTR Wrap Around occurred on last Write Bit to be cleared by SW
<b>WMOD</b>	7	rw	<b>Auto increment Write Mode</b> 0 <sub>B</sub> No increment WPTR on every Write 1 <sub>B</sub> Auto increment WPTR on every Write

**Memory Protection Unit (MPU)**

Field	Bits	Type	Description
<b>RPTR</b>	[12:8]	rwh	<b>Read Pointer</b> Selects the Protection Range Register to be read 00000 <sub>B</sub> Selects PRL0 00001 <sub>B</sub> Selects PRU0 00010 <sub>B</sub> Selects PRL1 00011 <sub>B</sub> Selects PRU1 00100 <sub>B</sub> Selects PRL2 00101 <sub>B</sub> Selects PRU2 ... 11110 <sub>B</sub> Selects PRL15 11111 <sub>B</sub> Selects PRU15
<b>0</b>	13	r	<b>Reserved field</b>
<b>RWA</b>	14	rwh	<b>Read Wrap Around Status</b> 0 <sub>B</sub> No RPTR Wrap Around occurred on last Read 1 <sub>B</sub> A RPTR Wrap Around occurred on last Read Bit to be cleared by SW
<b>RMOD</b>	15	rw	<b>Auto increment Read Mode</b> 0 <sub>B</sub> No increment RPTR on every Read 1 <sub>B</sub> Auto increment RPTR on every Read

## 6.3 Functional Description

### 6.3.1 Enabling Protection

Protection has to be globally enabled per software, bit PM0.PROTEN implements this functionality, refer to chapter [Chapter 6.2.2](#).

### 6.3.2 Protection Levels

The bits PSW.PL1/0 select the current protection level, i.e the protection register set currently active. The decoding of PL1/0 is as follows:

**Table 6-3 Decoding of Protection Level**

PL1	PL0	Protection Level
0	0	Protection Level 0
0	1	Protection Level 1
1	0	Protection Level 2
1	1	Protection Level 3

**Memory Protection Unit (MPU)**

PL1 and PL0 bits are mapped into PSW.10 and PSW.5 bits respectively. Note that due to the shared functionality implemented on the bit PSW.10, a write on this bit will be interpreted as a write on the PSW.PL1 only when the MPU is itself enabled (PM0.PROTEN is 1). When the MPU is not enabled a write on this bit will be interpreted as a write on the HLDEN flag. For consistency, the flag PSW.PL0 is handled in a similar way, a write on PSW.PL0 is only effective when the MPU is enabled.

**6.3.2.1 Protection Level 0**

For the protection mechanism to work properly, the MPU has to be operated under a kind of privileged mode, programming and changing the protection information should only be allowed during this mode. Even if the C166 family architecture does not support directly this operation mode (only the one associated to the initialization phase ended by the EINIT execution), the privileged mode can be defined in this context as the mode entered when the processor runs with protection Level 0. This is the level entered after reset and the level automatically entered after an interrupt/trap is taken. Level 0 should be then the level used by the operating system, software kernel or the software components needing access to the whole system resources (specially to system control registers and peripheral area).

But note that defining and programming address ranges and permissions is still needed for Level 0 (even if it is the whole space). Per default (i.e. after reset), no access in any address range is allowed, also not for this level.

*Note: The need to program Level 0 allows in special occasions to give restricted access also to this level. Restriction sometimes needed to probe reliability of the software running under this level.*

**6.3.3 Intersecting Memory Ranges**

The permission to access a memory location is the OR of the memory range permissions. When two or more ranges intersect, the intersecting region has the permission of the most permissive range.

**6.3.4 Protection of the MPU registers**

As mentioned in [Chapter 6.3.2.1](#), the MPU registers need to be protected. A protection mechanism comes automatically with the use of the MPU and the fact that the whole address space, including SFRs, is under control of the MPU, see also [Chapter 6.3.5](#). Once protection is enabled, changing protection information can then only be performed from a protection level which has access to the corresponding SFR area (i.e. to the protection registers).

In addition to this inherited protection mechanism, the protection control registers are also EINIT protected. The EINIT protection creates some overhead during dynamic re-programming, however it adds an additional protection level that may be needed in case

**Memory Protection Unit (MPU)**

different software component need to be executed at the same protection level (the one having access to these control registers -usually level 0-).

**6.3.5 Accessing SFRs and GPRs**

Once the protection system is activated, a task is not free anymore to access per software any special function register (SFR) unless this is explicitly covered by the address ranges and permissions assigned to this task. This applies to the internal IO area (SFR, ESFR, XSFR) and also to the external IO area (on chip LXBus peripherals or external peripherals). Since the minimum granularity of the address ranges is 256 bytes, the IO space is partitioned into blocks. A task will have access either to one of these blocks with its full set of registers or to none. For example the SFR/ESFR area (1 Kbyte), is divided into four blocks (F000h...F0FFh, F100h...F1FFh, FE00h...FEFFh, FF00h...FFFFh). For the XSFRs area, 4 Kbyte, the space is divided in 16 blocks.

CSFRs are also handled by the protection scheme, but exceptions are required for those CSFRs that are user registers. CSFR that are kept under the protection scheme are:

- PSW (partly), CPUCON1/2, CP, CSP, SP, SPSEG, STKUN, STKOV, TFR, VECSEG.

The USR0/1 bits of PSW, that are user bits, are excluded from the protection scheme. Also the PSW condition flags are excluded. Instructions like JBC/JNBS on the PSW conditions flags can then still be used in user mode. Read accesses to all the PSW fields are allowed.

CSFR that are excluded from the protection mechanism are:

- DPP0/1/2/3, MDL, MDH, MSW, MDC, MAH, MAL, MRW, MCW, QR0/1, QX0/1, IDX0/1, ZEROS, ONES, CPUID.

The DPP registers are handled as user registers to support its re-programming during run time (practice needed for code optimization purposes). When used in this way, it will be responsibility of the software to ensure their right handling, for example saving and restoring them in task switches. CPUID is not strictly a user register, however it is not required to define it as protected since it is anyhow not writable.

GPRs are excluded from the memory protection mechanism. Protection on the DPRAM is however guaranteed since the CP itself is protected. Similarly, GPRs mapped into the Local Register Banks are excluded from the protection mechanism.

**6.3.6 Interrupts and PECs Handling**

Any interrupt taken by the CPU will switch automatically the protection level to 0. This is valid for peripheral interrupts, debugger interrupts, hardware and software traps. As a consequence Interrupt Service Routines (ISRs) are always started with protection level 0, having usually access to all the system resources. The ISR itself can afterwards reduce the protection level and execute user code with protection restrictions.



**Memory Protection Unit (MPU)**

Interrupt requests can be also serviced through PEC transfers, that is, fast data transfers between two memory locations. PEC transfers will be executed by the CPU without protection. Protection can still be ensured through the programming of the PEC control registers that should be only performed under the right protection level, usually in privileged mode (i.e. protection level 0). At the configuration time the software should then check for the correctness of the PEC source and destination pointers (according to the permissions allowed) and the PEC control register. Special care has to be taken when using continuous mode, in this case the software can not take care at the configuration time if the PEC will not violate an area in the future. Additional run time checks may be needed to support this mode (executed by the privileged software) or this mode will have to be avoided.

**6.3.7 Special handling of RETI instruction**

The PSW and specially the Protection Level selection flags (PSW.PL0/1) are handled under the protection scheme: explicit writes on the PSW are detected by the hardware and checked if they are triggered under the right protection level. In case the access is not allowed, a trap will be generated and the modification of the PSW will be avoided by the protection logic.

But the PSW can also be modified implicitly by the hardware and this hardware update can hardly be managed by the protection logic. Hardware updates on the PSW.PL0/1 field are triggered by the execution of a RETI instruction. These PSW hardware updates are in principle not critical as long as the PSW (and PSW.PL0/1) value that is taken from the Stack has not been manipulated by any user code. But since there is no possibility to prohibit user code from this possible manipulations (user code may make use of local stacks with write access to it) the only work around is to prohibit un-trusted user code from using the RETI instruction. RETI will be then specially handled as a kind of protected instruction that can only be executed when the protection level 0. This handling is consistent with the fact that interrupts are handled under protection level 0, returning from interrupts should then also be performed under the same level.

**6.3.8 Context Switch operations**

The Context Switch mechanism is executed in the core with the help of internal instructions that are auto-injected in the pipeline. Usually auto-injected instructions should run with the same protection level as the instruction causing the auto injection. However, due to the fact that the context switch is an interruptible operation and its completion may be delayed in certain situations, these context switch auto injected instructions have to be executed without considering protection. That means, while they are executed, the protection checks are not performed and/or are ignored. As a consequence only the CP update operation, that is not performed by the auto-injected instruction but by the context switch instruction itself, is performed under the protection scheme. The saving procedure of registers into DPRAM or the read of GPRs from

## **Memory Protection Unit (MPU)**

DPRAM is not performed under the protection scheme. DPRAM protection in this case will have to be ensured, in case it is needed, by the software, the software can check if the region addressed by the values programmed into CP are allowed.

### **6.3.9 Debugger Access Permissions**

The debugger must be able to access all the memory space even if memory protection is active, this includes also the IO space (i.e. SFRs). The OCDS/Cerberus implements basically 2 mechanisms for accessing the system resources:

- triggering the CPU to execute a Monitor Routine that contains the code to access the resources (Call a Monitor)
- Injecting any instruction that can by itself access any resource

When using the first mechanism, that is started by the injection of an ITRAP instruction, the debugger will automatically run in privileged mode, i.e. with protection level 0. As defined in [Chapter 6.3.2.1](#), this is the level automatically entered after an interrupt (in this case after the injection of the debug TRAP instruction).

When using the second mechanism (also if the CPU is halted) the injected instruction will run without protection. The CPU keeps track of the fact that an instruction was injected by the Debugger and disables the protection check for that instruction.

With respect to accessing OCDS/Cerberus/MCDS SFRs by the debugger it just needs to be ensured that the debug monitor (used to program the debug logic) can access these registers with minimum overhead and without any impact on the user code. Since the debug monitor routine will always be executed with protection level 0, it is expected that all the memory space is then allowed. Also accessing these registers via injecting instructions can be performed without restrictions as explained above.

### **6.3.10 Invalid Access Traps**

If an access is performed in a protected area an invalid access trap will be generated. Three traps are defined for this purpose:

MPR Memory Protection, Read

MPW Memory Protection, Write

MPX Memory Protection, Execute

They are defined as Class B traps. They are mapped to TFR.10,9,8 respectively (MPR is TFR.10, MPW is TFR.9 and MPX is TFR.8). Refer to the Hardware Traps description chapter for the complete description of the TFR register.

Note that no trap must be performed on accesses that are performed speculatively, this is why these traps can just be generated when it is known that the instruction is not cancelled anymore (this is, when the instruction goes into the Execute stage).

**Memory Protection Unit (MPU)**

The already existing trap PRTFLT Protection Fault Trap is also used to indicate the execution of a RETI instruction from a protection level different to 0. Even when RETI causes a protection fault trap, it is normally executed.

**6.3.10.1 Cancelling operations**

Instructions causing a protection violation will be detected by the MPU but its full execution can not be suppressed, only the writes operations causing a protection fault or derived from an instruction causing a protection fault will be cancelled. Read operations can not be cancelled since they are triggered very soon in the pipeline (sometimes speculatively), however, the read data will not be written by the corresponding instruction in any memory mapped address.

There are some exceptions to the above general rule of cancelling writes operations, in particular, for instructions performing 2 write operations sometimes the first write can not be cancelled. These are the concrete cases:

- SCXT instruction. The write into the Stack can not be avoided when a Read protection violation on the mem operand is detected (for SCXT reg, mem) or a Write protection violation on the reg operand.
- CALLS, PCALL instruction. The first write into the Stack (CSP, or reg in case of PCALL) can not be avoided when a Write protection violation on the second Stack address is detected (where the IP should be pushed). This situation assumes that the Stack has grown over the limit of an allowed area right while executing this instruction (first stack push in an allowed area, second stack push in a non-allowed area).

As a consequence of the fact that Read operations can not be cancelled, destructive reads on the IO space can be still performed even if the MPU detects a protection violation.

As a consequence of the fact that Execute operations can not be cancelled, system instructions and their corresponding actions may be still executed even if they trigger an Execute protection violation. For example an IDLE instruction may still put the CPU in idle mode before the corresponding hardware trap routine can be executed (once the idle mode is left).

**6.4 Initializing and using the MPU****6.4.1 Installing Protection**

This chapter describes briefly the SW sequences needed for initializing and using the protection system. It also analyses the overhead created (real time performance). The implementation with 12 ranges is analyzed.

**Memory Protection Unit (MPU)**

The initialization sequence that can be used for installing protection is:

- Disable Protection in case it is not (after reset protection is disabled), 1 write into PM0.
- Program the Range Registers, 1 PRA write, 24 writes into PRD (absolute maximum value, assumes that all ranges are used).
- Program Protection Mode Registers, 6 PMx writes.
- Enable protection, 1 write into PM0. This last write can be performed together with the write into the PM0 above, but in this case care should be taken to write this register at the end.

When the applications or software components using different protection levels can exactly be mapped to the protection sets implemented, this code sequence would set up the system and no additional overhead when using the MPU would exist during run time. After the initialization phase, whenever a change in the protection level is needed, the corresponding protection set has to be selected (changing PSW.PL). This is the only additional operation during run time.

This initialization, and in general any change in the protection registers, should be performed always having the protection disabled and usually will be executed from protection level 0. Note that in cases where the protection configuration and its activation needs to be immediately seen, the software has to take care that the latest write activating the protection is effective before executing the next affected instruction (by reading for example the latest written register). This is because as explained in [Chapter 6.2.2](#), there is no hardware mechanism to flush the pipeline when the protection is activated.

In case the protection needs can not be mapped into the implemented protection sets, some re-programming during run time is needed. The worst case scenario is that the ranges have to be re-programmed, then a sequence similar to the one during the initialization is needed. However it may be that only some already defined Ranges needs to be activated/deactivated, in this case only the Protection Mode Registers will need to be re-programmed. This assumes that at every moment it is known which ranges are used, so the PSW.PL1/0 has to be read before deciding what to change. An additional overhead during reprogramming is coming from the EINIT protection. After EINIT execution, reprogramming of the PMU registers is only allowed by releasing temporarily this protection by going to an unprotected mode (a command sequence of 4 write instructions with the use of a password is needed for that). After the re-programming the EINIT-protection has to be of course restored (again a command sequence of 4 write instructions). Note that reprogramming of protection registers that are currently not active (i.e. selected through PSW.PL) and do not become active through the reprogramming, is still possible without having to disable protection.

### **6.4.2 Changing Protection Level**

Special care has to be taken when changing the protection level by writing explicitly into PSW. This is because any write into PSW takes effect immediately. If the privileged code handling protection would write into the PSW before performing a code/task switch, the level of the privileged code would be itself changed and eventually the code/task switch (function call for example) couldn't be performed. For avoiding this immediate effect when writing into the PSW some tricks have to be used: stack manipulation and calling functions using the RETI/RETP instructions. The value of the new PSW with the new protection level has to be then stored in the stack, instead of writing the PSW explicitly. The RET instruction will then update the PSW associated to the new task with the correct protection level and at the right time.

### **6.4.3 Executing privileged code from non-privileged one**

It is possible for a non-privileged (un-trusted) software to invoke a privileged software component (trusted). The non-privileged part has to give control to a privileged part, this can be performed by executing a software TRAP instruction. Automatically this instruction will change the protection level to 0. This mechanism allows for example invoking low level drivers from an application software and also returning from the user part of an ISR to the ISR itself, i.e. to the part handle by the OS or software kernel.

### **6.4.4 Fast task switches**

Any task switch that is not controlled by the OS or software kernel will be handled as a trusted task with respect to the software that is invoking it. This is because if the switch is not performed under a software running with enough protection level (usually level 0), there is no possibility to change the protection level explicitly for this task.

### **6.4.5 Register Bank Selection**

Since PSW.BANK field is now handled within the protection scheme, the register bank selection (global or local 1/2) will have to be settled by the software running with enough access permissions, i.e. enough protection level (usually level 0). The bank selection should then be done in the part of the ISR running on level 0 (starting level of all ISRs).

### **6.4.6 Debugger Use Cases**

This chapter documents how to debug the system when memory protection is in use.

The following 4 use cases are identified:

- user wants to find the reason for a MPU trap
- user wants to debug the MPU trap routine
- user doesn't want to debug, he just wants to poll a variable with Cerberus
- user wants to debug without any irregular influence from the protection system

## **Memory Protection Unit (MPU)**

For the first use case, the standard debugging resources (OCDS) can be used for setting an IP breakpoint on the ISR/s handling the trap. Once there, it will be known which access type causes the trap, either implicitly because there are different trap routines depending on the exception type or explicitly by reading the TFR flags. Also the protection level causing the violation can be obtained by reading the stacked PSW. With respect to the IP causing the trap, there is no direct access to it but to the linear following one, that is also stored in the stack.

The second use case, i.e. the debugging of the MPU trap routine, can be done similarly to the debugging of any trap routine and will be started probably also by setting an IP breakpoint on the corresponding ISR.

With respect the third use case, the variable polling action can be performed at any time independently of if the MPU is enabled or not. The debugger has always access to all the system resources even if the MPU is enabled.

The fourth use case, i.e. debugging without influence from the PMU, can be covered by disabling explicitly the MPU (PM0.PROTEN) via the debugger. Since the debugger does not know when the application will enable the MPU after the reset of the system, the debugger will have to monitor the status of the MPU (PMU0.PROTEN) and re-disable it once enabled. This use case is however rather strange because debugging is intended to be done with the real system behavior, if an application causes a MPU exception, this should also be seen during debugging.

## **7 Interrupt and Exception Control**

The architecture of the XE16xyM supports several mechanisms for fast and flexible response to service requests from various sources internal or external to the micro controller. Different kinds of exceptions are handled in a similar way:

- Interrupts generated by the Interrupt Controller (ITC)
- DMA transfers issued by the Peripheral Event Controller (PEC)
- Traps caused by the TRAP instruction or issued by faults or specific system states

### **Normal Interrupt Processing**

The CPU temporarily suspends the current program execution and branches to an interrupt service routine in order to service an interrupt requesting device. As a result, the current program status (IP, PSW, and, in segmentation mode, also CSP) is saved on the system stack. A prioritization scheme with sixteen priority levels specifies the execution order of multiple interrupt requests.

### **PEC Interrupt Processing**

A faster alternative to normal interrupt processing is the use of the XE16xyM's integrated **Peripheral Event Controller** (PEC) to service an interrupt requesting device. Triggered by an interrupt request, the PEC performs a single word or byte data transfer between any two memory locations. During a PEC transfer, the normal program execution of the CPU is interrupted only for the data transfer. No internal program status information needs to be saved. The same prioritization scheme is used for PEC service as for normal interrupt processing.

### **Trap Functions**

**Trap Functions** are activated in response to special conditions that occur during the execution of instructions. A trap can also be caused externally by the External Service Request pins ESRx (e.g. used to implement NMI like behavior). Several hardware trap functions are provided for handling erroneous conditions and exceptions that arise during the program execution. Hardware traps always have highest priority and cause immediate system reaction. The software trap function is invoked by the TRAP instruction that generates a software interrupt for a specified interrupt vector. For all types of traps, the current program status is saved on the system stack.

### **External Interrupt Processing**

The XE16xyM does not provide dedicated external interrupt input pins but rather allows to configure a subset of its input pins as interrupt inputs. Interrupt (trap) input pins can be chosen from standard inputs or External Service Request pins ESRx. The available options are detailed in the **External Interrupts** section.



### **Interrupt Sources and Routing**

To activate and correctly route an interrupt source programming of the following on-chip components must be considered:

- Interrupt control of each peripheral
- IMB memory controller **Interrupt Generation**
- SCU **External Request Unit (ERU)**
- **SCU Interrupt Generation**

Additionally the port programming must be considered if external interrupt sources are to be used.



## **7.1 Interrupt System Structure**

The XE16xyM provides 96 separate interrupt nodes assignable to 16 priority levels, with 8 sub-levels (group priority) on each level. In order to support modular and consistent software design techniques, most sources of an interrupt or PEC request are supplied with a separate interrupt control register and an interrupt vector. The control register contains the interrupt request flag, the interrupt enable bit, and the interrupt priority of the associated source. Each source request is then activated by one specific event, determined by the selected operating mode of the respective device. For efficient resource usage, multi-source interrupt nodes are also incorporated. These nodes can be activated by several source requests, such as by different kinds of errors in the serial interfaces. However, specific status flags which identify the type of error are implemented in the respective peripheral control registers. Additional sharing of interrupt nodes is supported via [Interrupt Node Sharing](#).

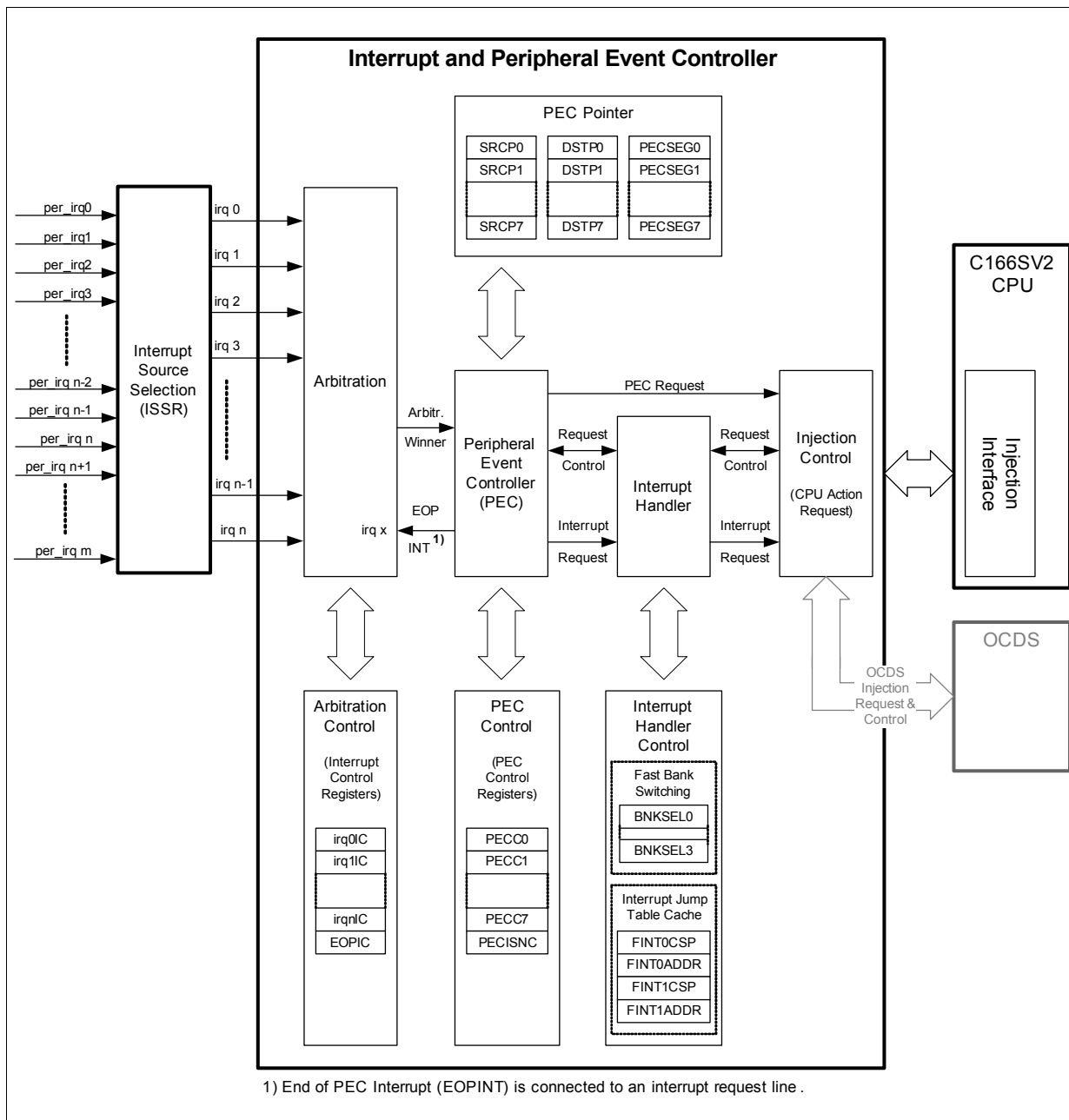
The XE16xyM provides a vectored interrupt system. In this system specific vector locations in the memory space are reserved for the reset, trap, and interrupt service functions. Whenever a request occurs, the CPU branches to the location that is associated with the respective interrupt source. The Class B hardware traps all share the same interrupt vector. The status flags in the Trap Flag Register (TFR) can then be used to determine which exception caused the trap. For the special software TRAP instruction, the vector address is specified by the operand field of the instruction, which is a seven bit trap number.

The reserved vector locations build a jump table in the low end of a segment (selected by register VECSEG) in the XE16xyM's address space. The jump table consists of the appropriate jump instructions which transfer control to the interrupt or trap service routines and which may be located anywhere within the address space. The entries of the jump table are located at the lowest addresses in the selected code segment. Each entry occupies 2, 4, 8, or 16 words (selected by bitfield VECSC in register CPUCON1), providing room for at least one double word instruction. The respective vector location results from multiplying the trap number by the selected step width ( $2^{(VECSC+2)}$ ).

All pending interrupt requests are arbitrated. The arbitration winner is indicated to the CPU together with its priority level and action request. The CPU triggers the corresponding action based on the required functionality (normal interrupt, PEC, jump table cache, etc.) of the arbitration winner.

An action request will be accepted by the CPU if the requesting source has a higher priority than the current CPU priority level and interrupts are globally enabled. If the requesting source has a lower (or equal) interrupt level priority than the current CPU task, it remains pending.

## Interrupt and Exception Control



**Figure 7-1 Block Diagram of the Interrupt and PEC Controller**

## **7.2 Interrupt Arbitration**

The XE16xyM interrupt arbitration system can handle interrupt requests from up to 96 sources. Interrupt requests may be triggered either by the internal peripherals or by external inputs. The “End of PEC” interrupt for supporting enhanced PEC functionality is connected internally to one of the interrupt request lines.

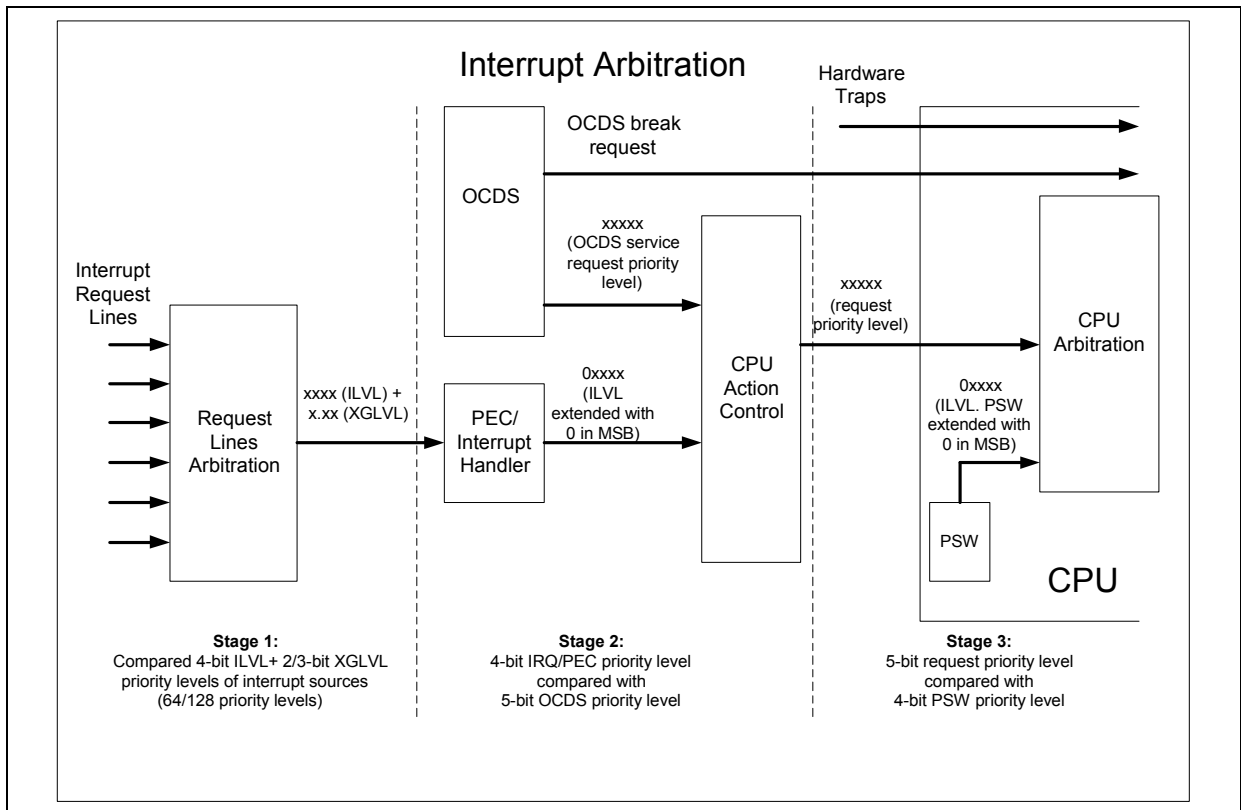
The arbitration process starts with an enabled interrupt request and stays active for as long as an enabled interrupt request is pending.

Each interrupt request line is controlled by its interrupt control register xxIC (here and below xx stands for the mnemonic of the respective interrupt source). An interrupt request event sets the interrupt request flag in the corresponding interrupt control register (bit xxIC.IR). The interrupt request can also be triggered by the software if the program sets the respective interrupt request bit.

If the request bit has been set and this interrupt request is enabled by the interrupt enable bit of the same control register (bit xxIC.IE), an arbitration cycle starts with the next clock cycle. However, if an arbitration cycle is currently in progress, the new interrupt request will be delayed until the next arbitration cycle. If an interrupt request (or PEC request) is accepted by the core, the respective interrupt request flag is cleared automatically.

All interrupt requests pending at the beginning of a new arbitration cycle are considered simultaneous. Within the arbitration cycle, the arbitration is independent of the actual request time.

The XE16xyM uses a three-stage interrupt prioritization scheme for interrupt arbitration as shown in **Figure 7-2**.



**Figure 7-2 Interrupt Arbitration**

The first arbitration stage compares the priority levels of interrupt request lines. The priority level of each requestor consists of interrupt priority level and group priority level. An interrupt priority level is programmed for each interrupt request line by the 4-bit bitfield ILVL of respective xxIC register. The group priority level is programmed for each interrupt request line by the 2-bit bitfield GLVL and the extension bit GPX of the register xxIC. Both together, GPX and GLVL form the 3-bit (extended) group priority level XGLVL, controlling up to eight interrupt sub-priorities within one of the 16 interrupt levels.

*Note: All interrupt request sources that are enabled and programmed to the same interrupt priority level (ILVL) must have different group priority levels. Otherwise, an incorrect interrupt vector may be generated.*

The second arbitration stage compares the priority of the first stage winner with the priority of OCDS service requests. OCDS service requests bypass the first stage of arbitration and go directly to the CPU Action Control Unit. The CPU Action Control Unit disregards the group priority level of interrupt/PEC requests and deals only with interrupt priority levels (ILVL). To compare with OCDS service request priority programmed by 5-bit value, the 4-bit ILVL of the interrupt/PEC request is extended to a 5-bit value with MSB equal to 0. This means that any OCDS request with MSB=1 will always win the second stage arbitration. However, if there is an OCDS request with MSB=0 conflicting with the same priority interrupt/PEC request, the latter is sent to the CPU.

**Interrupt and Exception Control**

On the third arbitration stage, the priority level of the second stage winner is compared with the priority of the current CPU task. An action request will be accepted by the CPU if the requesting source has a higher priority level than the current CPU priority level (bits ILVL of the PSW register) and interrupts are globally enabled by the global interrupt enable flag IEN in PSW. The CPU denies all requests in case of a cleared IEN flag. To compare with the 5-bit priority level of the second stage winner, the 4-bit ILVL.PSW is extended to a 5-bit value with MSB equal to 0. This means that any request with MSB=1 will always win the arbitration against any CPU level. If the requester has a lower or equal priority level than current CPU task, the request stays pending.

*Note: Priority level 0000<sub>B</sub> is the default level of the CPU. Therefore, a request on interrupt priority level 0000<sub>B</sub> will be arbitrated, but the CPU will never accept an action request on this level. However, every enabled interrupt request (including a denied interrupt request and a priority level 0000<sub>B</sub> request) triggers a CPU wake-up from idle state independent of setting the global interrupt enable bit PSW.IEN.*

### 7.3 Interrupt Control

All interrupt control registers are organized identically. The lower nine bits of an interrupt control register contain the complete interrupt control and status information of the associated source required during one round of prioritization (arbitration cycle). The upper seven bits of the respective register are reserved. All interrupt control registers are bit addressable and all control bits can be read or written via software. Therefore, each interrupt source can be programmed or modified with just one instruction. In the case of reading the interrupt control registers with instructions that operate with word data types, the upper 7 bits (15...9) will return zeros. It is recommended to always write zeros to these bit positions.

The IR bit of any IC register is of type “rwh” and is set by hardware upon occurrence of an interrupt. If the software requires to write to the IC register while the interrupt source is enabled the software write may conflict with a hardware access to bit IR. To address this conflict scenario all xxIC registers are located in the bit addressable memory area. The use of C166 bit modification instructions is therefore possible and recommended. These instructions provide a special “protection mask” feature which allows to protect IR bit from unintended software write. Refer to CPU Bit Manipulation Unit chapter for details.

The layout of the interrupt control registers shown below is applicable to all xxIC registers.

#### xxIC

**Interrupt Control Register** (E)SFR (xxxx<sub>H</sub>) **Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	GPX	IR	IE						
r	r	r	r	r	r	r	rw	rwh	rw						
												ILVL			GLVL
												rw			rw

Field	Bits	Type	Description
GPX	8	rw	<b>Group Priority Extension</b> Defines the value of high-order group level bit
IR	7	rwh	<b>Interrupt Request Flag</b> 0 <sub>B</sub> No request pending 1 <sub>B</sub> This source has raised an interrupt request
IE	6	rw	<b>Interrupt Enable Control Bit</b> (individually enables/disables a specific source) 0 <sub>B</sub> Interrupt request is disabled 1 <sub>B</sub> Interrupt request is enabled

## Interrupt and Exception Control

Field	Bits	Type	Description
ILVL	[5:2]	rw	<b>Interrupt Priority Level</b> $F_H$ Highest priority level ..... $0_H$ Lowest priority level
GLVL	[1:0]	rw	<b>Group Priority Level</b> $3_H$ Highest priority level ... .. $0_H$ Lowest priority level
0	[15:9]	r	<b>Reserved</b> read as 0; should be written with 0.

When accessing interrupt control registers through instructions which operate on word data types, their upper 7 bits (15 ... 9) will return zeros when read, and will discard written data. It is recommended to always write zeros to these bit positions.

The **Interrupt Request Flag** is set by hardware whenever a service request from its respective source occurs. It is cleared automatically upon entry into the interrupt service routine or upon a PEC service. In the case of PEC service, the Interrupt Request flag remains set if the COUNT field in register PECCx of the selected PEC channel decrements to zero and bit EOPINT is cleared. This allows a normal CPU interrupt to respond to a completed PEC block transfer on the same priority level.

*Note: Modifying the Interrupt Request flag via software causes the same effects as if it had been set or cleared by hardware.*

The **Interrupt Enable Control Bit** determines whether the respective interrupt node takes part in the arbitration process (enabled) or not (disabled). The associated request flag will be set upon a source request in any case. The occurrence of an interrupt request can so be polled via xxIR even while the node is disabled.

*Note: In this case the interrupt request flag xxIR is not cleared automatically but must be cleared via software.*

### 7.3.1 Interrupt Priority Level and Group Level

The four bits of bitfield ILVL specify the priority level of a service request for the arbitration of simultaneous requests. The priority increases with the numerical value of ILVL: so,  $0000_B$  is the lowest and  $1111_B$  is the highest priority level.

When more than one interrupt request on a specific level becomes active at the same time, the values in the respective bitfields GPX and GLVL are used for second level arbitration to select one request to be serviced. Again, the group priority increases with the numerical value of the concatenation of bitfields GPX and GLVL, so  $000_B$  is the lowest and  $111_B$  is the highest group priority.

## **Interrupt and Exception Control**

*Note: All interrupt request sources enabled and programmed to the same priority level must always be programmed to different group priorities. Otherwise, an incorrect interrupt vector will be generated.*

Upon entry into the interrupt service routine, the priority level of the source that won the arbitration and whose priority level is higher than the current CPU level, is copied into bitfield ILVL of register PSW after pushing the old PSW contents onto the stack.

The interrupt system of the XE16xyM allows nesting of up to 15 interrupt service routines of different priority levels (level 0 cannot be arbitrated).

Interrupt requests programmed to priority levels 15 ... 8 (i.e., ILVL = 1XXX<sub>B</sub>) can be serviced by the PEC if the associated PEC channel is properly assigned and enabled (please refer to [Section 7.10.4](#)). Interrupt requests programmed to priority levels 7 through 1 will always be serviced by normal interrupt processing.

### **7.3.2 General Interrupt Control Functions in Register PSW**

The acceptance of an interrupt request depends on the current CPU priority level (bitfield ILVL in register PSW) and the global interrupt enable control bit IEN in register PSW (see [Section 5.8](#)).

**CPU Priority ILVL** defines the current level for the operation of the CPU. This bitfield reflects the priority level of the routine currently executed. Upon entry into an interrupt service routine, this bitfield is updated with the priority level of the request being serviced. The PSW is saved on the system stack before the request is serviced. The CPU level determines the minimum interrupt priority level which will be serviced. Any request on the same or a lower level will not be acknowledged. The current CPU priority level may be adjusted via software to control which interrupt request sources will be acknowledged. PEC transfers do not really interrupt the CPU, but rather “steal” a single cycle, so PEC services do not influence the ILVL field in the PSW.

Hardware traps switch the CPU level to maximum priority (i.e. 15) so no interrupt or PEC requests will be acknowledged while an exception trap service routine is executed.

*Note: The TRAP instruction does not change the CPU level, so software invoked trap service routines may be interrupted by higher requests.*

**Interrupt Enable bit IEN** globally enables or disables PEC operation and the acceptance of interrupts by the CPU. When IEN is cleared, no new interrupt requests are accepted by the CPU. When IEN is set to 1, all interrupt sources, which have been individually enabled by the interrupt enable bits in their associated control registers, are globally enabled. Traps are non-maskable and are, therefore, not affected by the IEN bit.

*Note: To generate requests, interrupt sources must be also enabled by the interrupt enable bits in their associated control register.*



## Interrupt and Exception Control

**Register Bank Select bitfield BANK** defines the currently used register bank for the CPU operation. When the CPU enters an interrupt service routine, this bitfield is updated to select the register bank associated with the serviced request:

- Requests on priority levels 15 ... 12 use the register bank pre-selected via the respective bitfield GPRSELx in the corresponding BNKSEL register
- Requests on priority levels 11 ... 1 always use the global register bank, i.e. BANK = 00<sub>B</sub>
- Hardware traps always use the global register bank, i.e. BANK = 00<sub>B</sub>
- The TRAP instruction does not change the current register bank

### 7.3.3 Selective Interrupt Disabling

Interrupt requests may be temporarily disabled and enabled during the execution of the software. This may be required to exclude specific interrupt sources based on the current status of the application. In particular, this is necessary to achieve a deterministic execution of time-critical code sequences.

Interrupt requests in the XE16xyM can be disabled and enabled on three different levels:

- Disable all interrupt requests for a certain code sequence
- Disable all interrupt requests globally
- Disable single interrupt requests

**The ATOMIC and EXTend instructions** automatically disable all interrupt requests for the duration of the following 1 ... 4 instructions. This is useful for semaphore handling, for example, and does not require to re-enable the interrupt system after the inseparable instruction sequence.

**Global interrupt control** is achieved with a single instruction:

```
BCLR IEN                ;Clear IEN flag (causes pipeline restart)
```

**Specific interrupt control** is achieved by controlling the enable bits in the associated interrupt control registers.

```
BCLR T2IE              ;Clear enable flag to disable intr.node
```

Due to pipeline effects, however, an interrupt request may be executed after the corresponding node was disabled, if the request coincides with clearing the enable flag.

If the application must avoid this, the following sequence can be used, ensuring that no interrupt requests from this source will be serviced after disabling the interrupt node:

```
BCLR IEN                ;Globally disable interrupts
BCLR T2IE              ;Disable Timer 2 interrupt node
JNB T2IE, Next         ;Any instruction reading T2IC can be used
Next:                  ;(assures that T2IC is written by BCLR
                       ;before being read by JNB or other instr.)
BSET IEN               ;Globally enable interrupts again
```

## **Interrupt and Exception Control**

Please note that the sequence above blindly controls the global enable flag. If the global setting must not be changed, the code sequence can be enhanced, as shown below:

```
JNB  IEN, GlobalIntOff
BCLR IEN          ;Globally disable interrupts
BCLR T2IE         ;Disable Timer 2 interrupt node
JNB  T2IE, Next   ;Any instruction reading T2IC can be used
Next:              ;(assures that T2IC is written by BCLR
                  ;before being read by JNB or other instr.)

BSET IEN          ;Globally enable interrupts again
JMPR cc_uc, Continue
GlobalIntOff:     ;Interrupts are globally disabled anyway
BCLR T2IE         ;Disable Timer 2 interrupt node

JNB  T2IE, Continue ;Reading T2IC can be omitted if the next
Continue:         ;few instructions do not set IEN
...
```

The same function can easily be implemented as a C macro:

```
#define Disable_One_Interrupt(IE_bit) \
{if(IEN) {IEN=0; IE_bit=0; while (IE_bit); IEN=1;} else \
{IE_bit=0; while IE_bit);}}
Usage Example:
Disable_One_Interrupt(T2IE) ; // T2 interrupt enable flag
```

ATOMIC or EXTend sequences preserve the status of the interrupt arbitration when they begin. An accepted request is processed after the ATOMIC/EXTend sequence. Therefore, the following code sequence may not produce the desired result:

```
AvoidThis:
ATOMIC #3
NOP
BCLR T2IE          ;Disable Timer 2 interrupt node
NOP                ;Timer 2 request may be processed
                  ;after this instruction!!!
```

### **7.3.4 Interrupt Class Management**

An interrupt class covers a set of interrupt sources with the same importance, i.e. the same priority from the system's viewpoint. Interrupts of the same class must not interrupt each other. The XE16xyM supports this function with two features:

- **Classes with up to eight members** can be established by using the same interrupt priority (ILVL) and assigning a dedicated group level to each member. This functionality is built-in and handled automatically by the interrupt controller.

## Interrupt and Exception Control

- **Classes with more than eight members** can be established by using a number of adjacent interrupt priorities (ILVL) and the respective group levels (eight per ILVL). Each interrupt service routine within this class sets the CPU level to the highest interrupt priority within the class. All requests from the same or any lower level are blocked now, i.e. no request of this class will be accepted.

The example shown below establishes 3 interrupt classes which cover 2 or 3 interrupt priorities, depending on the number of members in a class. A level 6 interrupt disables all other sources in class 2 by changing the current CPU level to 8, which is the highest priority (ILVL) in class 2. Class 1 requests or PEC requests are still serviced, in this case. In this way, the interrupt sources (excluding PEC requests) are assigned to 3 classes of priority rather than to 7 different levels, as the hardware support would do.

**Table 7-1 Software Controlled Interrupt Classes (Example)**

ILVL (Priority)	Group Level								Interpretation
	7	6	5	4	3	2	1	0	
15									PEC service on up to 8 channels
14									
13									
12	X	X	X	X	X	X	X	X	Interrupt Class 1 9 sources on 2 levels
11	X								
10									
9									
8	X	X	X	X	X	X	X	X	Interrupt Class 2 17 sources on 3 levels
7	X	X	X	X	X	X	X	X	
6	X								
5	X	X	X	X	X	X	X	X	Interrupt Class 3 9 sources on 2 levels
4	X								
3									
2									
1									
0									No service!

## **7.4 Interrupt Vector Table**

The XE16xyM provides a vectored interrupt system. This system reserves a set of specific memory locations, which are accessed automatically upon the respective trigger event. Entries for the following events are provided:

- Reset (hardware, software, watchdog)
- Traps (hardware-generated by fault conditions or via TRAP instruction)
- Interrupt service requests

Whenever a request is accepted, the CPU branches to the location associated with the respective trigger source. This vector position directly identifies the source causing the request, with the following exceptions:

- Class B hardware traps all share the same interrupt vector. The status flags in the Trap Flag Register (TFR) are used to determine which exception caused the trap. For details, see [Section 7.9](#).
- An interrupt node may be shared by several interrupt requests, e.g. within a module. Additional flags identify the requesting source, so the software can handle each request individually. For details, see [Section 7.14.2](#).
- The interrupt jump cache feature is used. For details, see [Section 7.5](#)

The reserved vector locations build a vector table located in the address space of the XE16xyM. The vector table usually contains the appropriate jump instructions that transfer control to the interrupt or trap service routines. These routines may be located anywhere within the address space. The location and organization of the vector table is programmable.

The Vector Segment register VECSEG defines the segment of the Vector Table (can be located in all segments, except for reserved areas).

Bitfield VECSC in register CPUCON1 defines the space between two adjacent vectors (can be 2, 4, 8, or 16 words). For a summary of register CPUCON1, please refer to [Section 5.4](#).

Each vector location has an offset address to the segment base address of the vector table (given by VECSEG). The offset can be easily calculated by multiplying the vector number with the vector space programmed in bitfield VECSC.

**Table 7-9** lists all sources capable of requesting interrupt or PEC service in the XE16xyM, the associated interrupt vector locations, the associated vector numbers, and the associated interrupt control registers.

*Note: Interrupt nodes which are not used by their associated modules or are not connected to a module in the actual derivative may be used to generate software controlled interrupt requests by setting the respective IR flag.*

### VECSEG

**Vector Segment Pointer**

**SFR(FF12<sub>H</sub>)**

**Reset Value: 00XX<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>VECSEG</b>							
r	r	r	r	r	r	r	r	rwh							

Field	Bits	Type	Description
<b>VECSEG</b>	[7:0]	rwh	<b>Segment number of the Vector Table</b>
<b>0</b>	[15:8]	r	<b>Reserved</b> read as 0; should be written with 0.

The initial user value of register VECSEG is configured according to settings made for [Startup Configuration and Bootstrap Loading](#).

## 7.5 Interrupt Jump Table Cache

The mechanism that uses the vector table location as the entry point for the interrupt service routines can be overwritten by the Interrupt Controller (ITC). For a very fast interrupt response time, the XE16xyM offers the Interrupt Jump Table Cache (also called “fast interrupt”). The ITC can transfer to the CPU a 24-bit vector which is directly used as a start address for the service routine. This feature skips the path through the vector table which normally saves the execution of at least one branch. Therefore, avoiding the vector table may significantly improve interrupt response time. However, the number of 24-bit vectors in the ITC is limited.

Fast interrupt is available for two interrupt sources with interrupt priority levels greater than or equal to 12. The Interrupt Jump Table Cache skips the instruction fetches from the interrupt vector table and executes a direct jump to the interrupt service routines entry point. This feature is controlled by a set of two interrupt jump table cache registers (FINTxCSP, FINTxADDR) for each of the two jump table entries.

Every interrupt jump table cache entry contains an enable bit, an associated arbitration priority level (ILVL and GLVL), and the 24-bit address of the interrupt service routine. Note that only the two lower bits of the interrupt priority level are selectable in the respective control registers. The two upper bits of the interrupt priority level are fixed to  $11_B$ , which limits the allowed interrupt priority level to be greater than or equal to 12.

### FINT0CSP

**Fast Interrupt Control 0**

**XSFR(EC00<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

### FINT1CSP

**Fast Interrupt Control 1**

**XSFR(EC04<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>EN</b>	<b>0</b>	<b>0</b>	<b>GPX</b>	<b>ILVL</b>		<b>GLVL</b>						<b>SEG</b>			
rw	r	r	rw	rw		rw						rw			

Field	Bits	Type	Description
<b>EN</b>	15	rw	<b>Fast Interrupt Enable</b> $0_B$ The interrupt jump table cache is disabled. No fast interrupt is used. $1_B$ The interrupt jump table cache is enabled. A fast interrupt (direct jump to the interrupt service routine) is used instead of the normal fetch from the interrupt vector table.

**Interrupt and Exception Control**

Field	Bits	Type	Description
<b>GPX</b>	12	rw	<b>Group Priority Extension</b> This bit together with bitfield GLVL selects the group priority level (XGLVL) of the associated interrupt jump table cache entry.
<b>ILVL</b>	[11:10]	rw	<b>Interrupt Priority Level</b> This bitfield selects the lower two bits of the interrupt priority level associated with this interrupt jump table cache entry. <i>Note: The two upper bits of the interrupt priority level are fixed to 11<sub>B</sub>, which ends in an interrupt priority level greater than or equal to 12.</i>
<b>GLVL</b>	[9:8]	rw	<b>Group Priority Level</b> This bitfield together with GPX-bit selects the group priority level (XGLVL) of the associated interrupt jump table cache entry.
<b>SEG</b>	[7:0]	rw	<b>Segment Number of Interrupt Service Routine</b> Address bits 23:16 of the interrupt service routine's entry point.
<b>0</b>	[14:13]	r	<b>Reserved</b> read as 0; should be written with 0.

**FINT0ADDR**

**Fast Interrupt Address 0**

**XSFR (EC02<sub>H</sub>)**

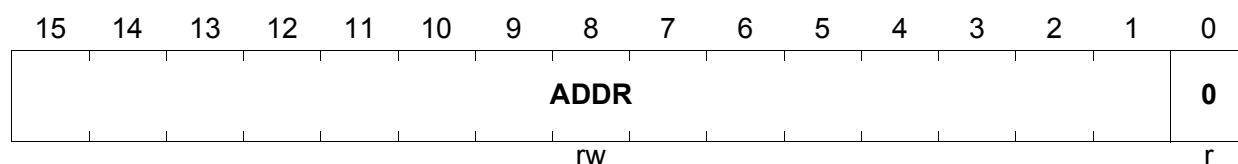
**Reset Value: 0000<sub>H</sub>**

**FINT1ADDR**

**Fast Interrupt Address 1**

**XSFR (EC06<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
<b>ADDR</b>	[15:1]	rw	<b>Address of Interrupt Service Routine</b> Address bits 15:1 of the interrupt service routine's entry point.

**Interrupt and Exception Control**

Field	Bits	Type	Description
<b>0</b>	0	r	<b>Interrupt Service Routine Address Bit 0</b> LSB of the interrupt service routine's entry point address. This address bit is always 0 because of the program code's word alignment.

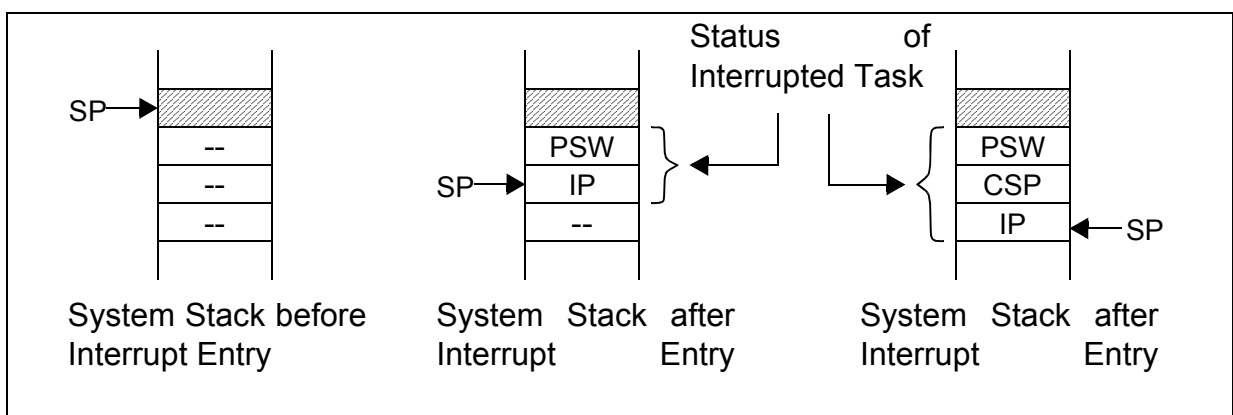


## 7.6 CPU Status Saving

Before an interrupt request that has been arbitrated is actually serviced, the status of the current task is automatically saved on the system stack. The CPU status (PSW) is saved together with the location at which execution of the interrupted task is to be resumed after returning from the service routine. This return location is specified through the Instruction Pointer (IP) and, in the case of a segmented memory model, the Code Segment Pointer (CSP). Bit SGTDIS in register CPUCON1 controls how the return location is stored.

- The system stack receives the PSW first, followed by the IP (unsegmented), or followed by CSP and then IP (segmented mode). This optimizes the usage of the system stack if segmentation is disabled.
- The CPU priority field (ILVL in PSW) is updated with the priority of the interrupt request to be serviced, so the CPU now executes on the new level.
- The register bank select field (BANK in PSW) is changed to select the register bank associated with the interrupt request. The association between interrupt requests and register banks are partly pre-defined and can partly be programmed.
- The interrupt request flag of the source being serviced is cleared. IP and CSP are loaded with the vector associated with the requesting source, and the first instruction of the service routine is fetched from the vector location which is expected to branch to the actual service routine (except when the interrupt jump table cache is used). All other CPU resources, such as data page pointers and the context pointer, are not affected.

When the interrupt service routine is exited (RETI is executed), the status information is popped from the system stack in the reverse order, taking into account the value of bit SGTDIS.



Task Status saved on the System Stack

## **7.7 CPU Context Switch**

An interrupt service routine usually saves all the registers it uses on the stack and restores them before returning. To ease this process the XE16xyM allows switching the complete bank of CPU registers (GPRs) either automatically or with a single instruction, so that the service routine executes within its own separate context (see also [Section 5.5.2](#)).

There are two ways to switch context:

1. **Context switch on interrupt** automatically updates bitfield PSW.BANK to select one of the two local register banks or the current global register bank, so the service routine may now use its “own registers” directly. This local register bank is preserved when the service routine is terminated; thus, its contents are available on the next call. For interrupt priority levels 15 ... 12 the target register bank can be pre-selected. The register bank selection registers BNKSELx provide a 2-bit field for each priority level. The respective bitfield is then copied to bitfield BANK in register PSW to select the register bank, as soon as the respective interrupt request is accepted.

2. **Explicit context switch by software** is initiated by a write to CP or PSW registers.
  - a) A write to PSW.BANK bitfields allows to switch between global and local banks.
  - b) A write to CP allows to relocate the memory mapped global bank to another memory location.

For example the instruction “SCXT CP, #New\_Bank” pushes the contents of the context pointer (CP) on the system stack and loads CP with the immediate value “New\_Bank”. The new CP value sets a new global register bank. The service routine may now use its “own registers”. This global register bank is preserved when the service routine is terminated, i.e. its contents are available for the next call. Before returning (RETI), the previous CP simply be restored from the system stack using “POP CP”.

*Note: Other resources used by an interrupting program (like DPP registers) must be saved and restored separately.*

*Note: There are certain timing restrictions during context switching associated with pipeline behavior. For details, see [Section 5.5.2](#).*

## 7.8 Fast Bank Switching

The interrupt handler supports an additional enhanced feature (compared to the C166 family) for normal interrupts called Fast Bank Switching. To speed up interrupt handling, the core can use fast General Purpose Register (GPR) bank switching for interrupts with an interrupt level greater or equal than 12. For every arbitration priority level with  $[ILVL = '15_D' - '12_D$  and  $XGLVL = '7_D' - '0_D]$ , the register bank can be selected with two bits. The select-bits are located in the four register bank selection registers BNKSELx (x = 0...3).

The following table identifies the arbitration priority level assignment to the respective bit fields within the four register bank selection registers:

**Table 7-2 Register Bank Assignment**

ILVL	XGLVL	Assigned GPRSELx Register	ILVL	XGLVL	Assigned GPRSELx Register
15	7	BNKSEL3.GPRSEL7	13	7	BNKSEL2.GPRSEL7
15	6	BNKSEL3.GPRSEL6	13	6	BNKSEL2.GPRSEL6
15	5	BNKSEL3.GPRSEL5	13	5	BNKSEL2.GPRSEL5
15	4	BNKSEL3.GPRSEL4	13	4	BNKSEL2.GPRSEL4
15	3	BNKSEL1.GPRSEL7	13	3	BNKSEL0.GPRSEL7
15	2	BNKSEL1.GPRSEL6	13	2	BNKSEL0.GPRSEL6
15	1	BNKSEL1.GPRSEL5	13	1	BNKSEL0.GPRSEL5
15	0	BNKSEL1.GPRSEL4	13	0	BNKSEL0.GPRSEL4
14	7	BNKSEL3.GPRSEL3	12	7	BNKSEL2.GPRSEL3
14	6	BNKSEL3.GPRSEL2	12	6	BNKSEL2.GPRSEL2
14	5	BNKSEL3.GPRSEL1	12	5	BNKSEL2.GPRSEL1
14	4	BNKSEL3.GPRSEL0	12	4	BNKSEL2.GPRSEL0
14	3	BNKSEL1.GPRSEL3	12	3	BNKSEL0.GPRSEL3
14	2	BNKSEL1.GPRSEL2	12	2	BNKSEL0.GPRSEL2
14	1	BNKSEL1.GPRSEL1	12	1	BNKSEL0.GPRSEL1
14	0	BNKSEL1.GPRSEL0	12	0	BNKSEL0.GPRSEL0

## Interrupt and Exception Control

### BNKSEL0

**Register Bank Selection 0**      XSFR(EC20<sub>H</sub>)      **Reset Value: 0000<sub>H</sub>**

### BNKSEL1

**Register Bank Selection 1**      XSFR(EC22<sub>H</sub>)      **Reset Value: 0000<sub>H</sub>**

### BNKSEL2

**Register Bank Selection 2**      XSFR(EC24<sub>H</sub>)      **Reset Value: 0000<sub>H</sub>**

### BNKSEL3

**Register Bank Selection 3**      XSFR(EC26<sub>H</sub>)      **Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>GPRSEL7</b>	<b>GPRSEL6</b>	<b>GPRSEL5</b>	<b>GPRSEL4</b>	<b>GPRSEL3</b>	<b>GPRSEL2</b>	<b>GPRSEL1</b>	<b>GPRSEL0</b>	<b>GPRSEL0</b>	<b>GPRSEL0</b>	<b>GPRSEL0</b>	<b>GPRSEL0</b>	<b>GPRSEL0</b>	<b>GPRSEL0</b>	<b>GPRSEL0</b>	<b>GPRSEL0</b>
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
<b>GPRSEL0,</b>	[1:0],	rw	<b>Register Bank Selection</b> 00 <sub>B</sub> Global register bank 01 <sub>B</sub> Reserved 10 <sub>B</sub> Local register bank 1 11 <sub>B</sub> Local register bank 2
<b>GPRSEL1,</b>	[3:2],		
<b>GPRSEL2,</b>	[5:4],		
<b>GPRSEL3,</b>	[7:6],		
<b>GPRSEL4,</b>	[9:8],		
<b>GPRSEL5,</b>	[11:10],		
<b>GPRSEL6,</b>	[13:12],		
<b>GPRSEL7</b>	[15:14]		

*Note: The GPRSELx value of the current triggered interrupt is automatically transferred into the Program Status Word (PSW).*

:

## **7.9 Trap Functions**

The C166SV2 CPU supports software and hardware trap functions.

### **7.9.1 Software Traps**

The TRAP instruction is used to cause a software call to an interrupt service routine. The trap number specified in the operand field of the trap instruction determines which vector location of the vector table will be used.

The TRAP instruction has an effect similar to an interrupt request at the same vector. PSW, CSP (in segmentation mode), and IP are pushed into the system stack and then a jump is taken to the specified vector location. When a software trap is executed, the CSP for the trap service routine is loaded with the value of the VECSEG register. No Interrupt Request flags are affected by the TRAP instruction. The interrupt service routine called by a TRAP instruction must be terminated with a RETI (return from interrupt) instruction to ensure correct operation.

*Note: The CPU priority level and the selected register bank in PSW register are not modified by the TRAP instruction; so, the service routine is executed with the same priority level as the interrupt task. Therefore, the service routine entered by the TRAP instruction can be interrupted by other traps or by higher priority interrupts, unless triggered by a real hardware event. The service routine also works with an unchanged register bank. If the hardware triggers the same service routine, register bank can be selected by the ITC and may be different.*

*Note: Software traps are also generated and issued, when data reads from the internal program memory space are requested which are not allowed, e.g. a user-read access to the protected Flash.*

### **7.9.2 Hardware Traps**

Hardware Traps are issued by faults or specific system states that occur during runtime (not identified at assembly time). The XE16xyM distinguishes twelve different hardware trap functions. When a hardware trap condition has been detected, the CPU branches to the trap vector location for the respective trap condition. The instruction causing the trap event is completed before the trap handling routine is entered.

Hardware traps are not-maskable and always have a priority higher than any other CPU task. If several hardware trap conditions are detected within the same instruction cycle, the highest priority trap is serviced (see [Table 7-3](#)). In case of a hardware trap, the injection unit injects a ITRAP instruction into the pipeline. The ITRAP instruction performs the following actions:

- Push PSW, CSP (in segmented mode) and IP into the System Stack
- Set CPU level in the PSW register to the highest possible priority level, which disables all interrupts and PEC transfers
- Select the global register bank for the trap service routine
- Branch to the trap vector location specified by the trap number of the trap condition

The hardware trap functions of the core are divided in two classes.

**Class A traps** are:

- System Request 0 (SR0)
- Stack Overflow
- Stack Underflow
- Software Break

These traps share the same trap priority, but have an individual vector address.

**Class B traps** are:

- System Request 1 (SR1)
- Memory Protection
- Undefined Opcode
- Memory Access Error
- Protection Fault
- Illegal Word Operand Access

The Class B traps share the same interrupt node and interrupt vector. The bit addressable Trap Flag Register (TFR) allows a trap service routine to identify the trap which caused the exception.

*Note: The trap service routine must clear the respective trap flag; otherwise, a new trap will be requested after exiting the service routine. Setting a trap request flag by software causes the same effects as if it had been set by hardware.*

The reset functions (hardware, software, watchdog) may be also regarded as a type of trap. Reset functions have the highest priority (trap priority III). Class A traps have the

## **Interrupt and Exception Control**

second highest priority (trap priority II), on the 3rd rank are class B traps (trap priority I); thus, a class A trap can interrupt a class B trap (for priority see also [Table 7-3](#)).

### **Class A Traps**

Class A traps are generated by the high priority system request SR0 or by special CPU events such as the software break, a stack overflow, or an underflow event. Class A traps are not used to indicate hardware failures. After a class A event, a dedicated service routine is called to react to the events. Each class A trap has its own vector location in the vector table. After finishing the service routine, the instruction flow must be further correctly executed. This explains why class A traps cannot interrupt atomic/extend sequences and IO accesses in progress. For example, an interrupted extend sequence cannot be restarted.

All class A traps are generated in the pipeline during the execution of instructions, with the exception of SR0, which is an asynchronous external event. It is not possible for two different instructions in the pipeline to generate traps in the same CPU cycle. Class A trap events can be generated only during the memory stage of execution. An execution of instruction which caused a class A trap event is always completed. In the case of a class A trap, the pipeline is directly canceled and the IP of the instruction following the last executed one is pushed into the stack. In the case of an atomic/extend sequence or IO read access in progress, the execution continues till the sequence completion. Upon completion of the sequence, the IP of the instruction following the last one executed is pushed into the stack. Therefore, in the case of a class A trap, the stack always contains the IP of the first not-executed instruction in the instruction flow.

*Note: The Branch Folding Unit allows an execution of branch instructions in parallel with the preceding instruction. The pre-processed branch instruction is combined with the preceding instruction. The branch is executed together with the instruction which caused the Class A trap. The IP of the first following not-executed instruction in the instruction flow is then pushed into the stack.*

If more than one Class A trap occurs at a same time, they are prioritized internally. The SR0 trap has the highest priority and the software break has the lowest.

*Note: In the case of two different class A trap occurring simultaneously, both trap flags are set. The IP of the instruction following the last one executed is pushed into the stack. The trap with the higher priority is executed. After return from the service routine, the IP is popped from the stack and immediately pushed again because of the other pending class A trap (unless the trap related to the second trap flag in TFR has been cleared by the first trap service routine).*

### **Class B Traps**

Class B traps are generated by unrecoverable hardware failures. In the case of a hardware failure, the CPU must immediately start a failure service routine. Class B traps



## **Interrupt and Exception Control**

can interrupt an atomic/extend sequence and an IO read access. After finishing the class B service routine, a restoration of the interrupted instruction flow is not possible.

All Class B traps have the same priority (trap priority I). When several class B traps become active at the same time, the corresponding flags in the TFR register are set and the trap service routine is entered. Because all class B traps have the same vector, the priority of service of simultaneously occurring class B traps is determined by the software in the trap service routine.

All class B traps are synchronous to instruction execution; most of them are generated in the pipeline during the execution of instructions. It is not possible for two different instructions in the pipeline to generate class A and class B traps in the same CPU cycle. Class B trap events can be generated only during memory stage execution. SR1 and ACER are exceptions, because they are generated by the SCU.

Instructions which caused a class B trap event are always executed. In the case of a class B trap, the pipeline is directly canceled and the IP of the instruction following the one which caused the trap is pushed on the stack. Therefore, the stack always contains the IP of the first following not executed instruction in the instruction flow.

*Note: The Branch Folding Unit allows the execution of branch instructions in parallel with the preceding instruction. The pre-processed branch instruction is combined with the preceding instruction. The branch is executed together with the instruction causing the Class B trap. The IP of the first following not executed instruction in the instruction flow is pushed into the stack.*

During execution of a class A trap service routine, any class B trap will not be serviced until the class A trap service routine is exited with a RETI instruction. In this case, the class B trap condition is stored in the TFR register, but the IP value of the instruction which caused this trap will be lost.

*Note: If a class A trap occurs simultaneously with a class B trap, both trap flags are set. The IP of the instruction following the one which caused the trap is pushed into the stack, and the class A trap is executed. If this occurs during execution of an atomic/extend sequence or IO read access in progress, then the presence of the class B trap breaks the protection of atomic/extend operations and the class A trap will be executed immediately without waiting for the sequence completion. After return from the service routine, the IP is popped from the system stack and immediately pushed again because of the other pending class B trap. In this situation, the restoration of the interrupted instruction flow is not possible.*

- **System Request 0 Trap (A):** The control signal is generated by the SCU. See chapter SCU Trap Generation.
- **Stack Overflow Trap (A):** Whenever the stack pointer is implicitly decremented and if the stack pointer was equal to the value in the stack overflow register STKOV, the STKOF flag in register TFR is set and the CPU will enter the stack overflow trap routine.



## **Interrupt and Exception Control**

- **Stack Underflow Trap (A):** Whenever the stack pointer is implicitly incremented and if the stack pointer was equal to the value in the stack underflow register STKUN, the STKUF flag is set in register TFR, and the CPU will enter the stack underflow trap routine.
- **Software Break Trap (A):** When the instruction currently being executed by the CPU is a SBRK instruction, the SOFTBRK flag is set in register TFR and the CPU enters the software break debug routine. The flag generation of the software break instruction can be disabled by an On-chip Emulation Module. In this case, the instruction only breaks the instruction flow and signals this event to the debugger. The flag is not set and the trap will not be executed.
- **System Request 1 Trap (B):** The control signal is generated by the SCU. See chapter SCU Trap Generation.
- **Memory Protection Traps (B):** When an access violation outside the permitted address ranges is detected. Depending on the access type it is differentiated between Read (MPR), Write (MPW) and Execute (MPX) violations.
- **Undefined Opcode Trap (B):** When the instruction currently being decoded by the CPU does not contain a valid C166SV2 opcode, the UNDOPC flag is set in register TFR and the CPU enters the undefined opcode trap routine. The instruction which causes the undefined opcode trap is executed as a NOP.
- **Memory Access Error (B):** The control signal is generated by the SCU. See chapter SCU Trap Generation.
- **Protection Fault Trap (B):** Whenever one of the special protected instructions is executed where the opcode of that instruction is not repeated twice in the second word of the instruction and the byte following the opcode is not the complement of the opcode, the PRTFLT flag in register TFR is set and the CPU enters the protection fault trap routine. The protected instructions include DISWDT, EINIT, IDLE, PWRDN, SRST, ENWDT and SRVWDT. The instruction which causes the protection fault trap is executed as a NOP. For products supporting MPU, RETI is also defined as a protected instruction in the sense that its execution is only allowed for privileged code, i.e. code executed with protection level 0. This flag is then used to indicate that a RETI instruction was tried to be executed from a protection level different to 0. Note that RETI will be still executed even if it causes a protection fault trap (it is not executed as a NOP).
- **Illegal Word Operand Access Trap (B):** Whenever a word operand read or write access (including Flash commands!) is attempted to an odd byte address, the ILLOPA flag in register TFR is set and the CPU enters the illegal word operand access trap routine.

### **Trap Vector Locations**

**Table 7-3** lists the vector locations for hardware traps and the corresponding status flags in register TFR. It also lists the priorities of trap service for those cases in which more than one trap condition might be detected within the same instruction. After any reset

## Interrupt and Exception Control

(hardware reset, software reset instruction SRST, or reset by watchdog timer overflow) program execution starts at the reset vector at location xx'0000<sub>H</sub>. Reset conditions have priority over every other system activity and, therefore, have the highest priority (trap priority III).

Software traps may be initiated to any defined vector location. A service routine entered via a software TRAP instruction is always executed on the current CPU priority level which is indicated in bitfield ILVL in register PSW. This means that routines entered via the software TRAP instruction can be interrupted by all hardware traps or higher level interrupt requests.

**Table 7-3 Hardware Trap Summary**

Exception Condition	Trap Flag	Trap Vector	Vector Location <sup>1)</sup>	Vector Number	Trap Priority
Application Reset	–	RESET	xx'0000 <sub>H</sub>	00 <sub>H</sub>	III
Class A Hardware Traps:					
• System Request 0	SR0	SR0TRAP	xx'0008 <sub>H</sub>	02 <sub>H</sub>	II
• Stack Overflow	STKOF	STOTRAP	xx'0010 <sub>H</sub>	04 <sub>H</sub>	II
• Stack Underflow	STKUF	STUTRAP	xx'0018 <sub>H</sub>	06 <sub>H</sub>	II
• Software Break	SOFTBRK	SBRKTRAP	xx'0020 <sub>H</sub>	08 <sub>H</sub>	II
Class B Hardware Traps:					
• System Request 1	SR1	BTRAP	xx'0028 <sub>H</sub>	0A <sub>H</sub>	I
• Memory Protection	MPR/W/X	BTRAP	xx'0028 <sub>H</sub>	0A <sub>H</sub>	I
• Undefined Opcode	UNDOPC	BTRAP	xx'0028 <sub>H</sub>	0A <sub>H</sub>	I
• Memory Access Error	ACER	BTRAP	xx'0028 <sub>H</sub>	0A <sub>H</sub>	I
• Protected Instruction Fault	PRTFLT	BTRAP	xx'0028 <sub>H</sub>	0A <sub>H</sub>	I
• Illegal Word Operand Access	ILLOPA	BTRAP	xx'0028 <sub>H</sub>	0A <sub>H</sub>	I

<sup>1)</sup> Register VECSEG defines the segment where the vector table is located to.  
 Bitfield VECSC in register CPUCON1 defines the distance between two adjacent vectors. This table represents the default setting, with a distance of 4 (two words) between two vectors.

### 7.9.2.1 The Trap Flag Register TFR

The XE16xyM provides a number of trap vectors (class A and class B) which are indicated in the trap flag register TFR.

#### TFR

#### Trap Flag Register

**SFR(FFAC<sub>H</sub>/D6<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>SR0</b>	<b>STK OF</b>	<b>STK UF</b>	<b>SOFT BRK</b>	<b>SR1</b>	<b>MPR</b>	<b>MPW</b>	<b>MPX</b>	<b>UND OPC</b>	<b>0</b>	<b>0</b>	<b>AC ER</b>	<b>PRT FLT</b>	<b>ILL OPA</b>	<b>0</b>	<b>0</b>
rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	r	r	rwh	rwh	rwh	r	r

Field	Bits	Type	Description
<b>SR0</b>	15	rwh	<b>System request flag 0</b> 0 <sub>B</sub> No trigger detected 1 <sub>B</sub> The selected condition has been detected
<b>STKOF</b>	14	rwh	<b>Stack overflow flag</b> 0 <sub>B</sub> No stack overflow event detected 1 <sub>B</sub> The current stack pointer value falls below the contents of register STKOV
<b>STKUF</b>	13	rwh	<b>Stack underflow flag</b> 0 <sub>B</sub> No stack underflow event detected 1 <sub>B</sub> The current stack pointer value exceeds the contents of register STKUN
<b>SOFTBRK</b>	12	rwh	<b>Software Break</b> 0 <sub>B</sub> No software break event detected 1 <sub>B</sub> Software break event detected
<b>SR1</b>	11	rwh	<b>System request flag 1</b> 0 <sub>B</sub> No trigger detected 1 <sub>B</sub> The selected condition has been detected
<b>MPR</b>	10	rwh	<b>Memory Protection Read</b> 0 <sub>B</sub> No read protection violation detected 1 <sub>B</sub> Read protection violation detected
<b>MPW</b>	9	rwh	<b>Memory Protection Write</b> 0 <sub>B</sub> No write protection violation detected 1 <sub>B</sub> Write protection violation detected

**Interrupt and Exception Control**

Field	Bits	Type	Description
<b>MPX</b>	8	rwh	<b>Memory Protection Execute</b> 0 <sub>B</sub> No execute protection violation detected 1 <sub>B</sub> Execute protection violation detected
<b>UNDOPC</b>	7	rwh	<b>Undefined Opcode</b> 0 <sub>B</sub> No undefined opcode event detected 1 <sub>B</sub> The currently decoded instruction has no valid opcode
<b>ACER</b>	4	rwh	<b>Memory Access Error</b> 0 <sub>B</sub> No access error event detected 1 <sub>B</sub> Illegal or erroneous access detected
<b>PRTFLT</b>	3	rwh	<b>Protection Fault</b> 0 <sub>B</sub> No protection fault event detected 1 <sub>B</sub> A protected instruction with an illegal format has been detected
<b>ILLOPA</b>	2	rwh	<b>Illegal word operand access</b> 0 <sub>B</sub> No illegal word operand access event detected 1 <sub>B</sub> A word operand access (read or write) to an odd address has been attempted
<b>0</b>	[6:5], [1:0]	r	<b>Reserved</b> read as 0; should be written with 0.

*Note: Flags TFR.15, TFR.11 and TFR.4 are generated via SCU. TFR.8, TFR9 and TFR.10 are generated via MPU. Other flags are generated by the CPU.*

## **7.10 Peripheral Event Controller**

The XE16xyM's Peripheral Event Controller (PEC) provides 8 PEC service channels which move a single byte or word between any two locations. A PEC transfer can be triggered by an interrupt service request and is the fastest possible interrupt response. In many cases a PEC transfer is sufficient to service the respective peripheral request (for example, serial channels, etc.).

PEC transfers do not change the current context, but rather “steal” cycles from the CPU, so the current program status and context needs not to be saved and restored as with standard interrupts.

The PEC channels can perform the following actions:

- Byte or word transfer
- Continuous data transfer
- PEC channel-specific interrupt request upon data transfer completion or common for all channels “End of PEC” interrupt for enhanced handling
- Automatic increment of source or/and destination pointers with support of memory to memory transfer

*Note: PEC transfer is executed if its priority level is higher than current CPU priority level.*

### 7.10.1 PEC Control Registers

Each PEC channel is controlled by the respective **PEC** channel **C**ontrol register (PECCx) and a set of source and destination pointers (SRCPx, DSTPx and PECSEGx), where x stands for the PEC channel number. The PECCx registers control the arbitration priority level assignment to the PEC channels and the action to be performed.

#### PECCx (x=0-7)

PEC Channel Control x						SFR(FEC0 <sub>H</sub> +2*x)						Reset Value: 0000 <sub>H</sub>			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>0</b>	<b>EOP INT</b>	<b>PLEV</b>	<b>CL</b>	<b>INC</b>	<b>BWT</b>						<b>COUNT</b>				
r	rw	rw	rw	rw	rw						rwh				

Field	Bits	Type	Description
<b>EOPINT</b>	14	rw	<b>End of PEC Interrupt Selection</b> 0 <sub>B</sub> End of PEC interrupt with the same level as the PEC transfer is triggered 1 <sub>B</sub> End of PEC interrupt is serviced by a separate interrupt node with programmable interrupt level (EOPIC) and interrupt sharing control register (PECISNC)
<b>PLEV</b>	[13:12]	rw	<b>Programmable PEC Interrupt Level</b> 00 <sub>B</sub> Standard (compatible mode): Levels 15 and 14 01 <sub>B</sub> Levels 13 and 12 10 <sub>B</sub> Levels 11 and 10 11 <sub>B</sub> Levels 9 and 8
<b>CL</b>	11	rw	<b>Channel Link Control</b> 0 <sub>B</sub> PEC channels work independent 1 <sub>B</sub> Pairs of channels are linked together

**Interrupt and Exception Control**

Field	Bits	Type	Description
<b>INC</b>	[10:9]	rw	<b>Increment Control</b> (Modification of source and destination pointer after PEC transfer) 00 <sub>B</sub> No modification 01 <sub>B</sub> Increment of destination pointer DSTPx by 1 (BWT = 1) or by 2 (BWT = 0) 10 <sub>B</sub> Increment of source pointer SRCPx by 1 (BWT = 1) or by 2 (BWT = 0) 11 <sub>B</sub> Increment of destination pointer DSTPx and source pointer SRCPx by 1 (BWT = 1) or by 2 (BWT = 0)
<b>BWT</b>	8	rw	<b>Byte/Word Transfer Selection</b> 0 <sub>B</sub> Transfer a word 1 <sub>B</sub> Transfer a byte
<b>COUNT</b>	[7:0]	rwh	<b>PEC Transfer Count</b> Counts PEC transfers and influences the channel's action (see <a href="#">Table 7-4</a> )
<b>0</b>	15	r	<b>Reserved</b> read as 0; should be written with 0.

The **Byte/Word Transfer Bit (BWT)** of the PECCx register selects if a byte or a word is to be moved during a PEC service cycle and defines an increment step size for the pointer(s) to be modified.

The **PEC Transfer Count Field (COUNT)** of the PECCx directly controls the action of the respective PEC channel. The contents of the bitfield COUNT may specify a certain number of PEC transfers, unlimited transfers, or no PEC service at all.

- If the PEC transfer counter COUNT value is set to **00<sub>H</sub>**, the normal interrupt requests are processed instead of PEC data transfers and the corresponding PEC channel remains idle.
- **Continuous data transfers** are selected by setting the bitfield COUNT to **FF<sub>H</sub>** value. In this case, COUNT is not decremented by the transfers and the respective PEC channel can serve unlimited number of PEC requests until it is modified by the program.
- If the bitfield COUNT is set to service a specified number of requests by the respective PEC channel, it is decremented with each PEC transfer and the request flag is cleared to indicate that the request has been serviced. When COUNT reaches **00<sub>H</sub>**, it activates the interrupt service routine which has the same priority level

## Interrupt and Exception Control

(EOPINT = 0) or triggers the “End of PEC” interrupt with a different priority level (EOPINT = 1). When COUNT is **decremented from 01<sub>H</sub> to 00<sub>H</sub>** after a data transfer, the request flag will be cleared if EOPINT is set to 1. If EOPINT is 0, the request flag will not be cleared and another interrupt request will be generated on the same priority level. The respective PEC channel remains idle and the associated interrupt service routine is activated instead of PEC transfer because COUNT contains the 00<sub>H</sub> value. (see [Section 7.10.3](#)).

The EOPIR register is the interrupt control register of the End Of PEC interrupt.

### EOPIC

**End Of PEC Interrupt Control**    **ESFR(F19E<sub>H</sub>/CF<sub>H</sub>)**                      **Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	GPX	EOP IR	EOP IE	ILVL				GLVL	
r	r	r	r	r	r	r	rw	rwh	rw	rw				rw	

Field	Bits	Type	Description
GPX	8	rw	<b>Group Priority Extension</b> Defines the value of high-order group level bit
EOPIR	7	rwh	<b>Interrupt Request Flag</b> 0 <sub>B</sub> No request pending 1 <sub>B</sub> The source has raised an interrupt request
EOPIE	6	rw	<b>Interrupt Enable Control Bit</b> 0 <sub>B</sub> Interrupt request is disabled 1 <sub>B</sub> Interrupt request is enabled
ILVL	[5:2]	rw	<b>Interrupt Priority Level</b> F <sub>H</sub> Highest priority level ... <sub>H</sub> ... 0 <sub>H</sub> Lowest priority level
GLVL	[1:0]	rw	<b>Group Priority Level</b> 3 <sub>H</sub> Highest priority level ... <sub>H</sub> ... 0 <sub>H</sub> Lowest priority level
0	[15:9]	r	<b>Reserved</b> read as 0; should be written with 0.



## Interrupt and Exception Control

*Note: The concatenation of the group priority extension bit GPX and the group priority level bitfield GLVL builds the 3-bit Extended Group Priority Level XGLVL, where  $7_H$  is the highest priority level and  $0_H$  is the lowest priority level.*

The Register **PECISNC** contains flags of the “End of PEC” interrupt node. This node is used when enhanced “End of PEC” interrupt feature was invoked and control bit EOPINT is set to 1 in the corresponding **PECCx**.

### PECISNC

**PEC Interrupt Sub Node ControlSFR(FFD8<sub>H</sub>/EC<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>C7IR</b>	<b>C7IE</b>	<b>C6IR</b>	<b>C6IE</b>	<b>C5IR</b>	<b>C5IE</b>	<b>C4IR</b>	<b>C4IE</b>	<b>C3IR</b>	<b>C3IE</b>	<b>C2IR</b>	<b>C2IE</b>	<b>C1IR</b>	<b>C1IE</b>	<b>C0IR</b>	<b>C0IE</b>
rwh	rw	rwh	rw	rwh	rw	rwh	rw	rwh	rw	rwh	rw	rwh	rw	rwh	rw

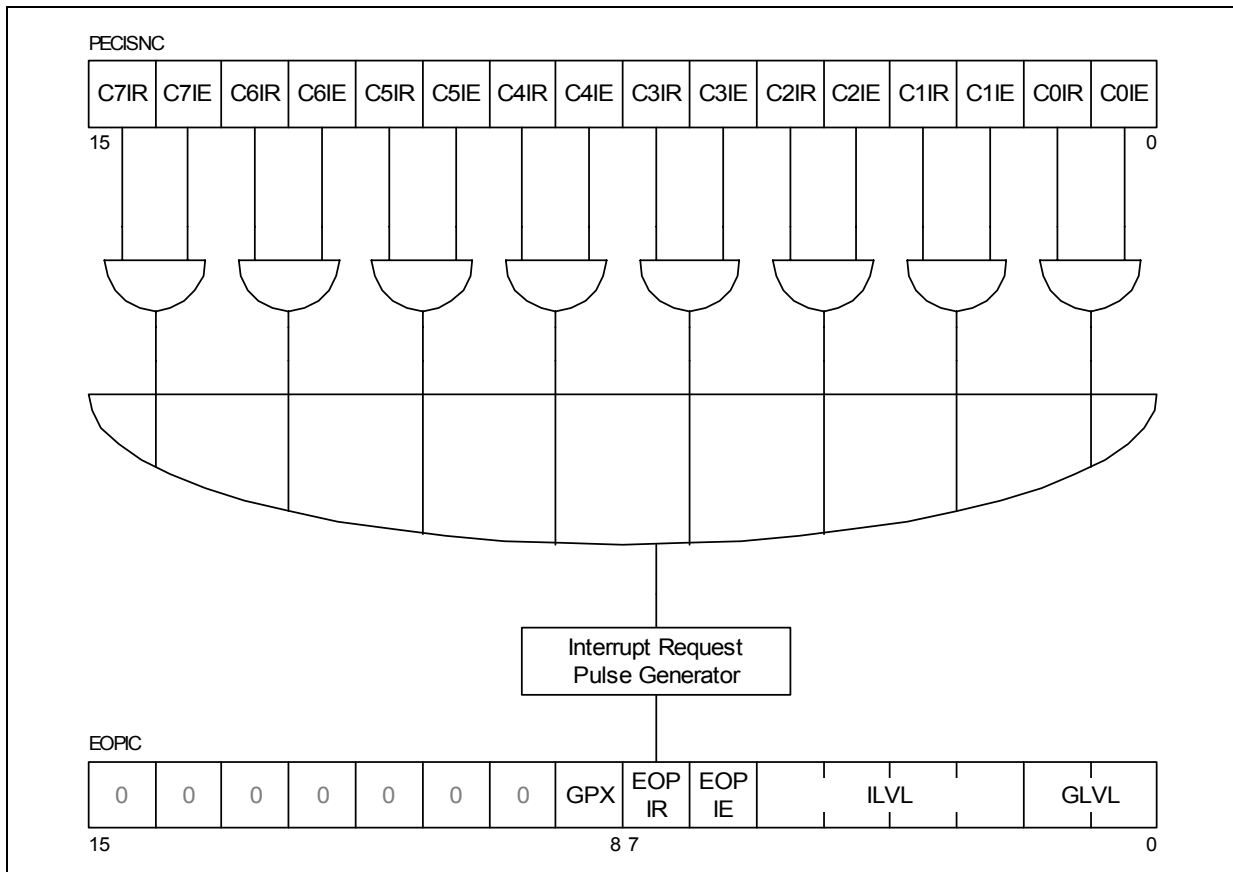
Field	Bits	Type	Description
<b>CxIR</b> (x=0-7)	2*x+1	rwh	<b>Interrupt Sub Node Request Flag of PEC Channel x <sup>1)</sup></b> $0_B$ No special end of PEC interrupt request is pending for PEC channel x $1_B$ PEC channel x has raised an end of PEC interrupt request
<b>CxIE</b> (x=0-7)	2*x	rw	<b>Interrupt Sub Node Enable Control Bit of PEC Channel x <sup>2)</sup></b> (individually enables/disables a specific source) $0_B$ End of PEC interrupt request of PEC channel x is disabled $1_B$ End of PEC interrupt request of PEC channel x is enabled

<sup>1)</sup> x = 7...0, depending on PEC channel number

<sup>2)</sup> It is recommended to clear an interrupt request flag (CxIR) before setting the respective enable flag (CxIE). Otherwise, former requests still pending will immediately trigger an interrupt request after setting the enable bit.

*Note: The “End of PEC” sub-node interrupt request flags are not cleared by hardware when entering the interrupt service routine (interrupt has been accepted by the CPU), unlike the interrupt request flags of the interrupt nodes (request flags xxIC.xxIR). The interrupt service routine must check the request flags and clear them before executing the RETI instruction.*

The following figure shows the usage of the “End of PEC” interrupt subnode:



**Figure 7-3 End of PEC Interrupt Sub Node**

The table below summarizes the values of the bitfield COUNT and the corresponding PEC channel actions

**Table 7-4 PEC Channel Actions**

<b>Previous COUNT field value</b>	<b>Modified COUNT field value</b>	<b>Action of PEC Channel and Comments</b>
FF <sub>H</sub>	FF <sub>H</sub>	<b>Move a Byte/Word</b> Continuous transfer mode, i.e. COUNT is not modified
FE <sub>H</sub> ...02 <sub>H</sub>	FD <sub>H</sub> ...01 <sub>H</sub>	Move a Byte/Word and decrement COUNT
01 <sub>H</sub>	00 <sub>H</sub>	<b>Move a Byte/Word</b> Depending on bit EOPINT, one of two different actions are taken: <b>EOPINT = 0</b> (compatible mode) The service request flag (xxIR) of the respective interrupt remains set (it is cleared for all other COUNT values). Therefore, an additional interrupt request is triggered on the next arbitration cycle with a COUNT field value of 00 <sub>H</sub> (see next row) <b>EOPINT = 1</b> The service request flag (xxIR) of the respective interrupt is cleared. Additionally, the interrupt request flag of the EOP sub node (PECISNC.CxIR) is set. Furthermore, the interrupt request flag of the end of PEC interrupt node (EOPIC.EOPiR) is automatically set if the sub node request is enabled (PECISNC.CxIE = 1'). (see also <a href="#">Section 7.10.3</a> )
00 <sub>H</sub>	00 <sub>H</sub>	<b>No PEC action!</b> A normal interrupt is requested instead of a PEC data transfer (see also <a href="#">Section 7.10.3</a> ).

The **Increment Control Field (INC)** of the PECCx register defines whether one or both of the PEC pointers must be incremented after the PEC transfer. If the pointers are not to be modified (INC='00'), the respective channel will always move data from the same source to the same destination.

### Channel Link Mode for Data Chaining

Channel linking, if enabled, links two channels together to serve the data transfer requests of one peripheral. The whole data transfer (for example a message) is divided into separately controlled and chained block transfers. The two PEC channels which are linked together, handle chained block transfers alternately to each other. At the end of a

**Interrupt and Exception Control**

data block transfer, controlled by one PEC channel, automatically the other (linked) PEC channel is started to continue the transfer with the next data block. Channel linking and thus data (block) chaining is supported within pairs of PEC channels (channels 0&1, 2&3 a.s.o.). Each data block is controlled by one PEC channel of the channel pair. While one of the two channels is active, the CPU can update the pointer and counter values of the other channel to prepare it for continuation of data transfer after next channel linking.

Channel linking is enabled, if in the active PEC channel of the channel pair the Channel Link Control Bit “CL” in its PECCx register is set to 1. The data transfer of linked channels is started always with the even numbered channel of the channel pair. If in Channel Link mode (at least one CL bit of the pair is set) the channel's data block is completely transferred the PEC service request processing is automatically switched to the other PEC channel of the channel-pair.

Channel linking and thus the switching from one channel to the other channel is performed, when the CL bit of first (active) channel is set (in its PEC control register) and its transfer count is changed from one to zero with last transfer. If the channel link flag CL of the first (terminated) PEC control register is found to be zero or the count field of linked channel is zero, the whole data transfer is finished.

*Note: The CL-flags are fully controlled by software and should be cleared by SW when the whole data transfer shall be finished and the termination of transfer shall be executed. Because termination can also be entered with a zero-value of the transfer count field of linked channel, termination of whole data transfer is automatically performed if the channels count field was not updated after the last channel link interrupt for this channel.*

When a data block of a linked channel is completely transferred and PEC servicing switches to the other channel of channel pair, a channel specific channel link interrupt is generated (for old channel) to inform the CPU that the channel is inactive now and may be configured for its next block transfer. The channel link interrupt is requested, indicated and enabled in the respective PEC Interrupt Subnode Control Register (PECISNC), which is also used for the channel's End of PEC interrupt (see chapter above). Thus, all channel link interrupts are also controlled with the one EOP interrupt control register EOPIIC and therefore with the same interrupt priority level as the EOP interrupt. This service request node requests CPU interrupt service in case of one or more pending channel link interrupt requests or End of PEC interrupt requests, if the respective enable control bit(s) is (are) set in the PEC interrupt subnode control register PECISNC and in the interrupt control register EOPIIC.

*Note: The generation of Channel Link/EOP interrupt is automatically enabled, if the CL-bit of the active (terminated) channel is set. If it is not set, either a standard interrupt or an EOP interrupt is initiated according to the EOPINT bit in the channel's PEC Control Register. The channel is not switched in this case, because a missing CL flag defines the last block of data transfer.*

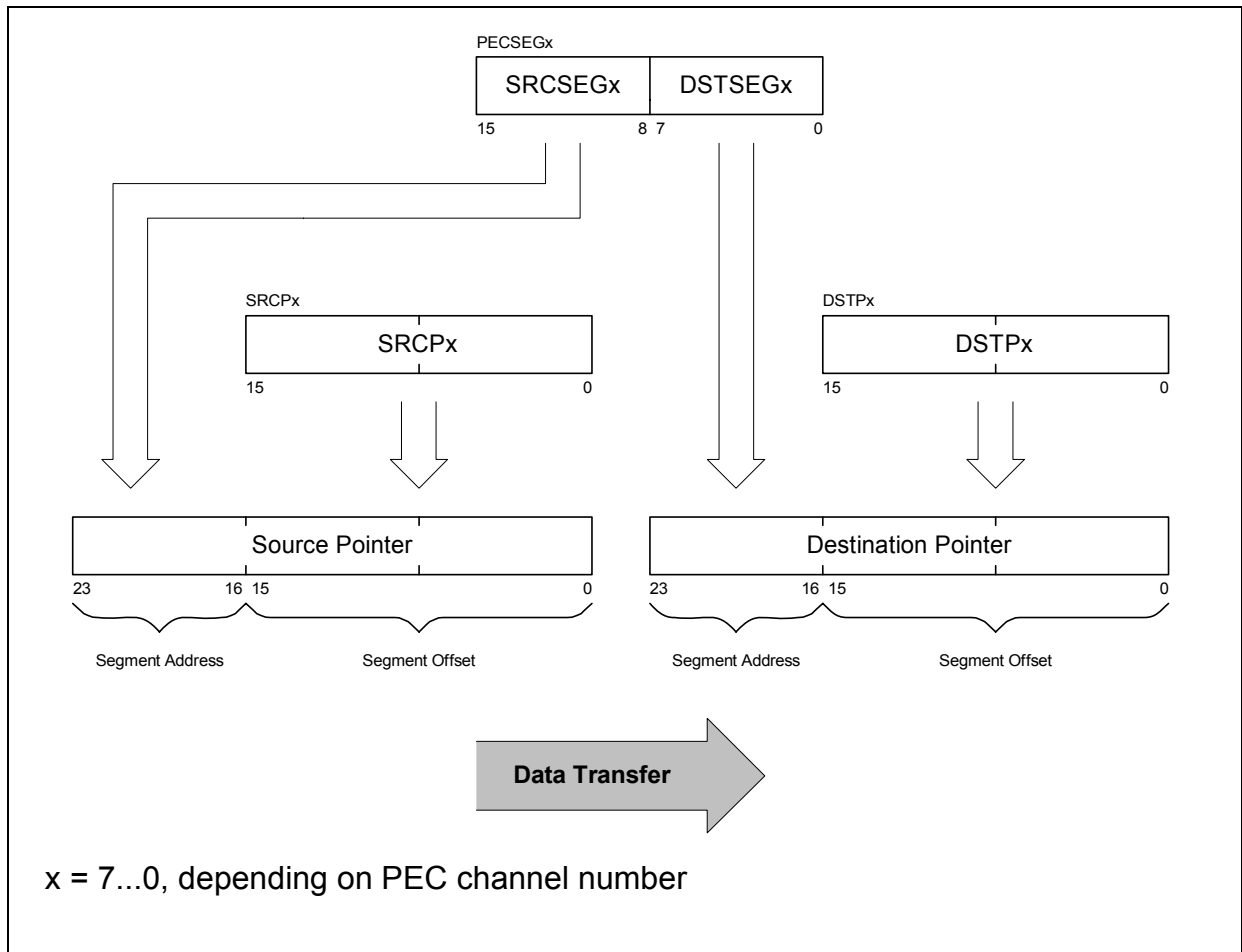
## **Interrupt and Exception Control**

*Note: If Channel Link mode is active (at least one of the pair's CL bits is set), interrupt requests connected to the odd channel (via priority levels) will trigger only a standard interrupt but no PEC transfer.*

*Note: The start of data transfer on linked channels is always performed with the even numbered PEC channel of the channel pair.*

### 7.10.2 The PEC Source and Destination Pointer

The PEC channels source and destination pointers specify the locations between which the data is to be moved. All pointers are 24-bits wide. The 24-bit source address is stored in the register SRCPx (lower 16 bits of address) and in the high byte of register PECSEGx (highest 8 address bits).



**Figure 7-4 PEC Pointer Address Handling**

The 24-bit destination address is stored in the register DSTPx (lower 16 bits of address) and in the low byte of register PECSEGx (highest 8 address bits). Only the lower 16 bits of the PEC address pointers (segment offset) can be modified (incremented) by the PEC transfer mechanism. The highest 8 bits, which represent the segment number, are not modified by hardware. Therefore, the PEC pointers may be incremented within the address space of one segment and may not cross the segment border. If the offset address pointer gets the  $FFFF_H$  value in case of byte transfers ( $BWT = 1$ ) or  $FFFE_H$  in case of word transfers ( $BWT = 0$ ), the next increment will be disregarded. The address register will keep one of these maximum values and no overflow will happen. The described behavior protects the subsequent memory from unintentional overwriting. No

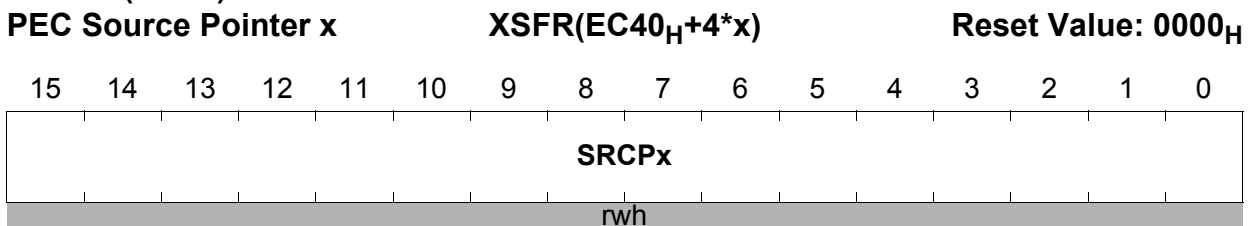
## Interrupt and Exception Control

explicit error event is generated by the system in case of address pointer(s) saturation; therefore, it is the user's responsibility to prevent this condition.

*Note: PEC data transfers do not use the data page pointers DPP3...DPP0.*

*Note: If a word data transfer is selected for a specific PEC channel (i.e. BWT = 0), the respective source and destination pointers must both contain a valid word address which points to an even byte boundary. Otherwise, the Illegal Word Operand Access trap will be invoked when this channel is used.*

### SRCPx (x=0-7)



Field	Bits	Type	Description
SRCPx	[15:0]	rwh	<b>Source Pointer Offset of Channel x</b> Source address bits 15 ... 0

x = 7...0, depending on PEC channel number

### DSTPx (x=0-7)



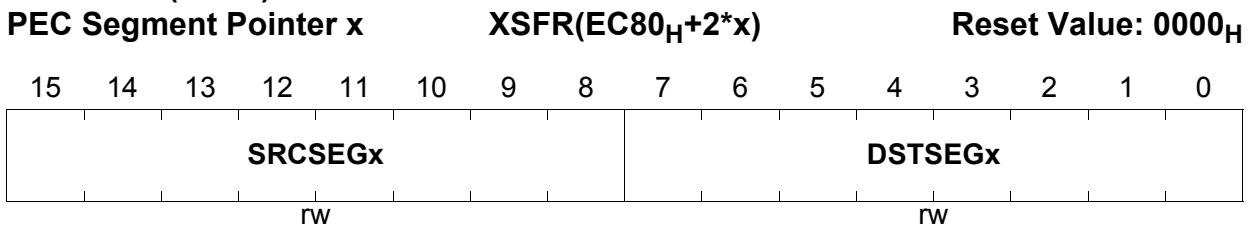
Field	Bits	Type	Description
DSTPx	[15:0]	rwh	<b>Destination Pointer Offset of Channel x</b> Destination address bits 15 ... 0

x = 7...0, depending on PEC channel number

For XSFR addresses see [Table 7-11](#).

## Interrupt and Exception Control

### PECSEGx (x=0-7)



Field	Bits	Type	Description
<b>SRCSEGx</b>	[15:8]	rw	<b>Source Pointer Segment of Channel x</b> Source address bits 23 ... 16
<b>DSTSEGx</b>	[7:0]	rw	<b>Destination Pointer Segment Address of Channel x</b> Destination address bits 23 ... 16

x = 7...0, depending on PEC channel number

### 7.10.3 PEC Interrupt Processing Summary

As described above, two different kinds of interrupts can be triggered by the PEC handler depending on the status of the bitfield COUNT.

- PEC channel is enabled<sup>1)</sup> and the bitfield COUNT has a value higher then 01<sub>H</sub>.
  - Control bit EOPINT = 0 or 1
  - ACTIONS:**
    - PEC request is proceeded
    - No other interrupt activity
- PEC channel is enabled and the bitfield COUNT gets a decrement from 01<sub>H</sub> to 00<sub>H</sub> triggered by a service request.
  - Control bit EOPINT = 0 (compatible with C166)
  - ACTIONS:**
    - PEC request is proceeded
    - Interrupt request flag (xxIR) of the requesting interrupt node (arbitration winner) is not cleared, participates on the next arbitration cycle, and triggers a normal interrupt on the same level as the PEC request is served.
  - Control bit EOPINT = 1 (enhanced end of PEC interrupt feature)
  - ACTIONS:**
    - PEC request is proceeded

<sup>1)</sup> Every PEC channel is automatically enabled, when its COUNT value is greater than 00<sub>H</sub>.



## Interrupt and Exception Control

- Interrupt request flag (xxIR) of requesting interrupt node (arbitration winner) is cleared and will not trigger more actions.
  - Interrupt request flag of the end of PEC interrupt subnode will be set (PECISNC.CxIR = 1)
  - If the respective interrupt enable flag of the end of PEC interrupt subnode was set before by software (PECISNC.CxIE = 1), an end of PEC interrupt is requested (EOPIC.EOPIR = 1). This end of PEC interrupt participates on the next arbitration cycle with its priority (selected via EOPIC.ILVL and EOPIC.GLVL), if this interrupt source was enabled before by software (EOPIC.EOPIR = 1). With this behavior an end of PEC interrupt can be triggered on a lower level than the respective PEC requests have been serviced.
- PEC channel is disabled if the bitfield COUNT is cleared (either by hardware or by software)
    - Control bit EOPINT = 0 or 1
- ACTIONS:**
- A normal interrupt service routine is requested on the PEC channel priority level.

### 7.10.4 PEC Channel Assignment

The PEC channel used for executing the transfer depends on the programming of the interrupt, group and PEC level. The following table lists the channel assignments.

**Table 7-5 PEC Channel Assignment**

ICx. ILVL	ICx. GLVL	PECCy. PLEV	Assigned PEC Channel	ICx. ILVL	ICx. GLVL	PECCy. PLEV	Assigned PEC Channel
1111 <sub>B</sub>	11 <sub>B</sub>	00 <sub>B</sub>	<b>7</b>	1011 <sub>B</sub>	11 <sub>B</sub>	10 <sub>B</sub>	<b>7</b>
1111 <sub>B</sub>	10 <sub>B</sub>		<b>6</b>	1011 <sub>B</sub>	10 <sub>B</sub>		<b>6</b>
1111 <sub>B</sub>	01 <sub>B</sub>		<b>5</b>	1011 <sub>B</sub>	01 <sub>B</sub>		<b>5</b>
1111 <sub>B</sub>	00 <sub>B</sub>		<b>4</b>	1011 <sub>B</sub>	00 <sub>B</sub>		<b>4</b>
1110 <sub>B</sub>	11 <sub>B</sub>		<b>3</b>	1010 <sub>B</sub>	11 <sub>B</sub>		<b>3</b>
1110 <sub>B</sub>	10 <sub>B</sub>		<b>2</b>	1010 <sub>B</sub>	10 <sub>B</sub>		<b>2</b>
1110 <sub>B</sub>	01 <sub>B</sub>		<b>1</b>	1010 <sub>B</sub>	01 <sub>B</sub>		<b>1</b>
1110 <sub>B</sub>	00 <sub>B</sub>		<b>0</b>	1010 <sub>B</sub>	00 <sub>B</sub>		<b>0</b>

**Interrupt and Exception Control**

**Table 7-5      PEC Channel Assignment**

<b>ICx. ILVL</b>	<b>ICx. GLVL</b>	<b>PECCy. PLEV</b>	<b>Assigned PEC Channel</b>	<b>ICx. ILVL</b>	<b>ICx. GLVL</b>	<b>PECCy. PLEV</b>	<b>Assigned PEC Channel</b>
1101 <sub>B</sub>	11 <sub>B</sub>	01 <sub>B</sub>	<b>7</b>	1001 <sub>B</sub>	11 <sub>B</sub>	11 <sub>B</sub>	<b>7</b>
1101 <sub>B</sub>	10 <sub>B</sub>		<b>6</b>	1001 <sub>B</sub>	10 <sub>B</sub>		<b>6</b>
1101 <sub>B</sub>	01 <sub>B</sub>		<b>5</b>	1001 <sub>B</sub>	01 <sub>B</sub>		<b>5</b>
1101 <sub>B</sub>	00 <sub>B</sub>		<b>4</b>	1001 <sub>B</sub>	00 <sub>B</sub>		<b>4</b>
1100 <sub>B</sub>	11 <sub>B</sub>		<b>3</b>	1000 <sub>B</sub>	11 <sub>B</sub>		<b>3</b>
1100 <sub>B</sub>	10 <sub>B</sub>		<b>2</b>	1000 <sub>B</sub>	10 <sub>B</sub>		<b>2</b>
1100 <sub>B</sub>	01 <sub>B</sub>		<b>1</b>	1000 <sub>B</sub>	01 <sub>B</sub>		<b>1</b>
1100 <sub>B</sub>	00 <sub>B</sub>		<b>0</b>	1000 <sub>B</sub>	00 <sub>B</sub>		<b>0</b>

All interrupt requests not assigned to a PEC channel go directly to the interrupt handler.

## **7.11 External Interrupts**

Although the XE16xyM has no dedicated interrupt input pins, it supports many possibilities to react to external asynchronous events by providing a number of IO lines which can be selected as interrupt inputs.

### **7.11.1 External Request Unit**

Please refer to the [External Request Unit \(ERU\)](#) chapter. The ERU provides routing capabilities and allows to define advanced trigger conditions for the interrupt input signals. The resulting ERU interrupt requests are forwarded to the interrupt controller registers ERU\_0IC ... ERU\_3IC.

### **7.11.2 Using Peripheral Pins**

The interrupt function of some peripheral pins may be either combined with the pin's main function or used instead of it if the main pin function is not required.

**Table 7-6 Pins Usable as External Interrupt Inputs**

<b>Port Pin</b>	<b>Original Function</b>	<b>Control Register</b>
P4.7-0/CC31-24IO	CAPCOM Register 31-24 Capture Input	CC31-CC24 <sup>1)</sup>
P2.10-3/CC23-16IO	CAPCOM Register 23-16 Capture Input <sup>2)</sup>	CC23-CC16 <sup>1)</sup>
P4.2/T2IN	Auxiliary timer T2 input pin	T2CON
P4.6/T4IN	Auxiliary timer T4 input pin	T4CON
P2.10/CAPIN	GPT2 capture input pin <sup>2)</sup>	T5CON

<sup>1)</sup> Must be enabled by [Interrupt Node Sharing](#).

<sup>2)</sup> Pin P2.10 overlays two possible input functions.

For each of these pins, either a positive, a negative, or both a positive and a negative external transition can be selected to cause an interrupt or PEC service request. The edge selection is performed in the control register of the peripheral device associated with the respective port pin. The peripheral must be programmed to a specific operating mode to allow generation of an interrupt by the external signal. The priority of the interrupt request is determined by the interrupt control register of the respective peripheral interrupt source, and the interrupt vector of this source will be used to service the external interrupt request.

*Note: In order to use any of the listed pins as an external interrupt input, it must be switched to input mode via its port control register.*

When port pins CCxIO are to be used as external interrupt input pins, bitfield CCMODx in the control register of the corresponding capture/compare register CCx must select capture mode. When CCMODx is programmed to 001<sub>B</sub>, the interrupt request flag CCxIR

**Interrupt and Exception Control**

in register CCxIC will be set on a positive external transition at pin CCxIO. When CCMODx is programmed to 010<sub>B</sub>, a negative external transition will set the interrupt request flag. When CCMODx = 011<sub>B</sub>, both a positive and a negative transition will set the request flag. In all three cases, the contents of the allocated CAPCOM timer will be latched into capture register CCx, independent of whether or not the timer is running. When the interrupt enable bit CCxIE is set, a PEC request or an interrupt request for vector CCxINT will be generated.

Pins T2IN or T4IN can be used as external interrupt input pins when the associated auxiliary timer T2 or T4 in block GPT1 is configured for capture mode. This mode is selected by programming the mode control fields T2M or T4M in control registers T2CON or T4CON to 101<sub>B</sub>. The active edge of the external input signal is determined by bitfields T2I or T4I. When these fields are programmed to X01<sub>B</sub>, interrupt request flags T2IR or T4IR in registers T2IC or T4IC will be set on a positive external transition at pins T2IN or T4IN, respectively. When T2I or T4I is programmed to X10<sub>B</sub>, then a negative external transition will set the corresponding request flag. When T2I or T4I is programmed to X11<sub>B</sub>, both a positive and a negative transition will set the request flag. In all three cases, the contents of the core timer T3 will be captured into the auxiliary timer registers T2 or T4 based on the transition at pins T2IN or T4IN. When the interrupt enable bits T2IE or T4IE are set, a PEC request or an interrupt request for vector T2INT or T4INT will be generated.

Pin CAPIN differs slightly from the timer input pins as it can be used as external interrupt input pin without affecting peripheral functions. When the capture mode enable bit T5SC in register T5CON is cleared to '0', signal transitions on pin CAPIN will only set the interrupt request flag CRIR in register CRIC, and the capture function of register CAPREL is not activated.

So register CAPREL can still be used as reload register for GPT2 timer T5, while pin CAPIN serves as external interrupt input. Bitfield CI in register T5CON selects the effective transition of the external interrupt input signal. When CI is programmed to 01<sub>B</sub>, a positive external transition will set the interrupt request flag. CI = 10<sub>B</sub> selects a negative transition to set the interrupt request flag, and with CI = 11<sub>B</sub>, both a positive and a negative transition will set the request flag. When the interrupt enable bit CRIE is set, an interrupt request for vector CRINT or a PEC request will be generated.

## **7.12 OCDS Requests**

The OCDS module issues high-priority break requests or standard service requests. The break requests are routed directly to the CPU (like the hardware trap requests) and are prioritized there. Therefore, break requests ignore the standard interrupt arbitration and receive highest priority.

The standard OCDS service requests are routed to the CPU Action Control Unit together with the arbitrated interrupt/PEC requests. The service request with the higher priority is sent to the CPU to be serviced. If both the interrupt/PEC request and the OCDS request have the same priority level, the interrupt/PEC request wins.

This approach ensures precise break control, while affecting the system behavior as little as possible.

The CPU Action Control Unit also routes back request acknowledges and denials from the core to the corresponding requestor.

### **7.13 Service Request Latency**

The numerous service requests of the XE16xyM (requests for interrupt or PEC service) are generated asynchronously with respect to the execution of the instruction flow. Therefore, these requests are arbitrated and are inserted into the current instruction stream. This decouples the service request handling from the currently executed instruction stream, but also leads to a certain latency.

The request latency is the time from activating a request signal at the interrupt controller (ITC) until the corresponding instruction reaches the pipeline's execution stage.

**Table 7-7** lists the consecutive steps required for this process.

**Table 7-7 Steps Contributing to Service Request Latency**

<b>Description of Step</b>	<b>Interrupt Response</b>	<b>PEC Response</b>
Request arbitration in 3 stages, leads to acceptance by the CPU (see <a href="#">Section 7.2</a> )	3 cycles	3 cycles
Injection of an internal instruction into the pipeline's instruction stream	4 cycles	4 cycles
The first instruction fetched from the interrupt vector table reaches the pipeline's execution stage	4 cycles / 0 <sup>1)</sup>	- - -
Resulting minimum request latency	11/7 cycles	7 cycles

<sup>1)</sup> Can be saved by using the interrupt jump table cache (see [Section 7.5](#)).

### Sources for Additional Delays

Because the service requests are inserted into the current instruction stream, the properties of this instruction stream can influence the request latency.

**Table 7-8 Additional Delays Caused by System Logic**

Reason for Delay	Interrupt Response	PEC Response
Interrupt controller busy, because the previous interrupt request is still in process	max. 7 cycles	max. 7 cycles
Pipeline is stalled, because instructions preceding the injected instruction in the pipeline need to write/read data to/from a peripheral or memory	$2 \times T_{ACCmax}^{1)}$	$2 \times T_{ACCmax}$
Pipeline cancelled, because instructions preceding the injected instruction in the pipeline update core SFRs	4 cycles	4 cycles
Memory access for stack writes (if not to DPRAM or DSRAM)	$2/3 \times T_{ACC}^{2)}$	- - -
Memory access for vector table read (except for intr. jump table cache)	$2 \times T_{ACC}$	- - -

<sup>1)</sup> This is the longest possible access time within the XE16xyM system.

<sup>2)</sup> Depending on segmentation off/on.

The actual response to an interrupt request may be delayed further depending on programming techniques used by the application. The following factors can contribute:

- Actual interrupt service routine is only reached via a JUMP from the interrupt vector table.  
Time-critical instructions can be placed directly into the interrupt vector table, followed by a branch to the remaining part of the interrupt service routine. The space between two adjacent vectors can be selected via bitfield VECSC in register CPUCON1.
- Context switching is executed before the intended action takes place (see [Section 7.6](#))  
Time-critical instructions can be programmed “non-destructive” and can be executed before switching context for the remaining part of the interrupt service routine.

## **7.14 Interrupt Nodes**

### **7.14.1 Physical Interrupt Nodes**

The full set of enabled and used modules integrated in the XE16xyM would require more than the 96 interrupt nodes provided by the C166SV2 interrupt controller. Therefore some of the physically available interrupt nodes are shared between selected modules.

The following table summarizes the 96 physical interrupt nodes with their related

- trap number
- vector location
- control register name and address
- node sharing information

**Table 7-9 XE16xyM Interrupt Nodes**

<b>Source of Interrupt or PEC Service Request</b>	<b>Trap Number</b>	<b>Vector<sup>1)</sup> Location</b>	<b>Control Register</b>	<b>Register Address</b>
selected by ISSR.ISS0	10 <sub>H</sub>	xx'0040 <sub>H</sub>	CC2_CC16IC	F1C0 <sub>H</sub>
selected by ISSR.ISS1	11 <sub>H</sub>	xx'0044 <sub>H</sub>	CC2_CC17IC	F1C2 <sub>H</sub>
selected by ISSR.ISS2	12 <sub>H</sub>	xx'0048 <sub>H</sub>	CC2_CC18IC	F1C4 <sub>H</sub>
selected by ISSR.ISS3	13 <sub>H</sub>	xx'004C <sub>H</sub>	CC2_CC19IC	F1C6 <sub>H</sub>
selected by ISSR.ISS4	14 <sub>H</sub>	xx'0050 <sub>H</sub>	CC2_CC20IC	F1C8 <sub>H</sub>
selected by ISSR.ISS5	15 <sub>H</sub>	xx'0054 <sub>H</sub>	CC2_CC21IC	F1CA <sub>H</sub>
selected by ISSR.ISS6	16 <sub>H</sub>	xx'0058 <sub>H</sub>	CC2_CC22IC	F1CC <sub>H</sub>
selected by ISSR.ISS7	17 <sub>H</sub>	xx'005C <sub>H</sub>	CC2_CC23IC	F1CE <sub>H</sub>
selected by ISSR.ISS8	18 <sub>H</sub>	xx'0060 <sub>H</sub>	CC2_CC24IC	F1D0 <sub>H</sub>
selected by ISSR.ISS9	19 <sub>H</sub>	xx'0064 <sub>H</sub>	CC2_CC25IC	F1D2 <sub>H</sub>
selected by ISSR.ISS10	1A <sub>H</sub>	xx'0068 <sub>H</sub>	CC2_CC26IC	F1D4 <sub>H</sub>
selected by ISSR.ISS11	1B <sub>H</sub>	xx'006C <sub>H</sub>	CC2_CC27IC	F1D6 <sub>H</sub>
selected by ISSR.ISS12	1C <sub>H</sub>	xx'0070 <sub>H</sub>	CC2_CC28IC	F1D8 <sub>H</sub>
selected by ISSR.ISS13	1D <sub>H</sub>	xx'0074 <sub>H</sub>	CC2_CC29IC	F1DA <sub>H</sub>
selected by ISSR.ISS14	1E <sub>H</sub>	xx'0078 <sub>H</sub>	CC2_CC30IC	F1DC <sub>H</sub>
selected by ISSR.ISS15	1F <sub>H</sub>	xx'007C <sub>H</sub>	CC2_CC31IC	F1DE <sub>H</sub>
GPT1 Timer 2	20 <sub>H</sub>	xx'0080 <sub>H</sub>	GPT12E_T2IC	FF60 <sub>H</sub>
GPT1 Timer 3	21 <sub>H</sub>	xx'0084 <sub>H</sub>	GPT12E_T3IC	FF62 <sub>H</sub>
GPT1 Timer 4	22 <sub>H</sub>	xx'0088 <sub>H</sub>	GPT12E_T4IC	FF64 <sub>H</sub>



**Interrupt and Exception Control**

**Table 7-9 XE16xyM Interrupt Nodes**

<b>Source of Interrupt or PEC Service Request</b>	<b>Trap Number</b>	<b>Vector<sup>1)</sup> Location</b>	<b>Control Register</b>	<b>Register Address</b>
GPT2 Timer 5	23 <sub>H</sub>	xx'008C <sub>H</sub>	GPT12E_T5IC	FF66 <sub>H</sub>
GPT2 Timer 6	24 <sub>H</sub>	xx'0090 <sub>H</sub>	GPT12E_T6IC	FF68 <sub>H</sub>
GPT2 CAPREL	25 <sub>H</sub>	xx'0094 <sub>H</sub>	GPT12E_CRIC	FF6A <sub>H</sub>
CAPCOM2 Timer 7	26 <sub>H</sub>	xx'0098 <sub>H</sub>	CC2_T7IC	FF6C <sub>H</sub>
CAPCOM2 Timer 8	27 <sub>H</sub>	xx'009C <sub>H</sub>	CC2_T8IC	FF6E <sub>H</sub>
A/D Converter Request 0	28 <sub>H</sub>	xx'00A0 <sub>H</sub>	ADC_0IC	FF70 <sub>H</sub>
A/D Converter Request 1	29 <sub>H</sub>	xx'00A4 <sub>H</sub>	ADC_1IC	FF72 <sub>H</sub>
A/D Converter Request 2	2A <sub>H</sub>	xx'00A8 <sub>H</sub>	ADC_2IC	FF74 <sub>H</sub>
A/D Converter Request 3	2B <sub>H</sub>	xx'00AC <sub>H</sub>	ADC_3IC	FF76 <sub>H</sub>
A/D Converter Request 4	2C <sub>H</sub>	xx'00B0 <sub>H</sub>	ADC_4IC	FF78 <sub>H</sub>
A/D Converter Request 5	2D <sub>H</sub>	xx'00B4 <sub>H</sub>	ADC_5IC	FF7A <sub>H</sub>
A/D Converter Request 6	2E <sub>H</sub>	xx'00B8 <sub>H</sub>	ADC_6IC	FF7C <sub>H</sub>
A/D Converter Request 7	2F <sub>H</sub>	xx'00BC <sub>H</sub>	ADC_7IC	FF7E <sub>H</sub>
CCU60 Request 0	30 <sub>H</sub>	xx'00C0 <sub>H</sub>	CCU60_0IC	F160 <sub>H</sub>
CCU60 Request 1	31 <sub>H</sub>	xx'00C4 <sub>H</sub>	CCU60_1IC	F162 <sub>H</sub>
CCU60 Request 2	32 <sub>H</sub>	xx'00C8 <sub>H</sub>	CCU60_2IC	F164 <sub>H</sub>
CCU60 Request 3	33 <sub>H</sub>	xx'00CC <sub>H</sub>	CCU60_3IC	F166 <sub>H</sub>
CCU61 Request 0	34 <sub>H</sub>	xx'00D0 <sub>H</sub>	CCU61_0IC	F168 <sub>H</sub>
CCU61 Request 1	35 <sub>H</sub>	xx'00D4 <sub>H</sub>	CCU61_1IC	F16A <sub>H</sub>
CCU61 Request 2	36 <sub>H</sub>	xx'00D8 <sub>H</sub>	CCU61_2IC	F16C <sub>H</sub>
CCU61 Request 3	37 <sub>H</sub>	xx'00DC <sub>H</sub>	CCU61_3IC	F16E <sub>H</sub>
CCU62 Request 0	38 <sub>H</sub>	xx'00E0 <sub>H</sub>	CCU62_0IC	F170 <sub>H</sub>
CCU62 Request 1	39 <sub>H</sub>	xx'00E4 <sub>H</sub>	CCU62_1IC	F172 <sub>H</sub>
CCU62 Request 2	3A <sub>H</sub>	xx'00E8 <sub>H</sub>	CCU62_2IC	F174 <sub>H</sub>
CCU62 Request 3	3B <sub>H</sub>	xx'00EC <sub>H</sub>	CCU62_3IC	F176 <sub>H</sub>
CCU63 Request 0	3C <sub>H</sub>	xx'00F0 <sub>H</sub>	CCU63_0IC	F178 <sub>H</sub>
CCU63 Request 1	3D <sub>H</sub>	xx'00F4 <sub>H</sub>	CCU63_1IC	F17A <sub>H</sub>
CCU63 Request 2	3E <sub>H</sub>	xx'00F8 <sub>H</sub>	CCU63_2IC	F17C <sub>H</sub>
CCU63 Request 3	3F <sub>H</sub>	xx'00FC <sub>H</sub>	CCU63_3IC	F17E <sub>H</sub>
CAN0	40 <sub>H</sub>	xx'0100 <sub>H</sub>	CAN_0IC	F140 <sub>H</sub>

**Interrupt and Exception Control**

**Table 7-9 XE16xyM Interrupt Nodes**

<b>Source of Interrupt or PEC Service Request</b>	<b>Trap Number</b>	<b>Vector<sup>1)</sup> Location</b>	<b>Control Register</b>	<b>Register Address</b>
CAN1	41 <sub>H</sub>	xx'0104 <sub>H</sub>	CAN_1IC	F142 <sub>H</sub>
CAN2	42 <sub>H</sub>	xx'0108 <sub>H</sub>	CAN_2IC	F144 <sub>H</sub>
CAN3	43 <sub>H</sub>	xx'010C <sub>H</sub>	CAN_3IC	F146 <sub>H</sub>
CAN4	44 <sub>H</sub>	xx'0110 <sub>H</sub>	CAN_4IC	F148 <sub>H</sub>
CAN5	45 <sub>H</sub>	xx'0114 <sub>H</sub>	CAN_5IC	F14A <sub>H</sub>
CAN6	46 <sub>H</sub>	xx'0118 <sub>H</sub>	CAN_6IC	F14C <sub>H</sub>
CAN7	47 <sub>H</sub>	xx'011C <sub>H</sub>	CAN_7IC	F14E <sub>H</sub>
CAN8	48 <sub>H</sub>	xx'0120 <sub>H</sub>	CAN_8IC	F150 <sub>H</sub>
CAN9	49 <sub>H</sub>	xx'0124 <sub>H</sub>	CAN_9IC	F152 <sub>H</sub>
CAN10	4A <sub>H</sub>	xx'0128 <sub>H</sub>	CAN_10IC	F154 <sub>H</sub>
CAN11	4B <sub>H</sub>	xx'012C <sub>H</sub>	CAN_11IC	F156 <sub>H</sub>
CAN12	4C <sub>H</sub>	xx'0130 <sub>H</sub>	CAN_12IC	F158 <sub>H</sub>
CAN13	4D <sub>H</sub>	xx'0134 <sub>H</sub>	CAN_13IC	F15A <sub>H</sub>
CAN14	4E <sub>H</sub>	xx'0138 <sub>H</sub>	CAN_14IC	F15C <sub>H</sub>
CAN15	4F <sub>H</sub>	xx'013C <sub>H</sub>	CAN_15IC	F15E <sub>H</sub>
USIC0 CH0 SR0	50 <sub>H</sub>	xx'0140 <sub>H</sub>	U0C0_0IC	F120 <sub>H</sub>
USIC0 CH0 SR1	51 <sub>H</sub>	xx'0144 <sub>H</sub>	U0C0_1IC	F122 <sub>H</sub>
USIC0 CH0 SR2	52 <sub>H</sub>	xx'0148 <sub>H</sub>	U0C0_2IC	F124 <sub>H</sub>
USIC0 CH1 SR0	53 <sub>H</sub>	xx'014C <sub>H</sub>	U0C1_0IC	F126 <sub>H</sub>
USIC0 CH1 SR1	54 <sub>H</sub>	xx'0150 <sub>H</sub>	U0C1_1IC	F128 <sub>H</sub>
USIC0 CH1 SR2	55 <sub>H</sub>	xx'0154 <sub>H</sub>	U0C1_2IC	F12A <sub>H</sub>
USIC1 CH0 SR0	56 <sub>H</sub>	xx'0158 <sub>H</sub>	U1C0_0IC	F12C <sub>H</sub>
USIC1 CH0 SR1	57 <sub>H</sub>	xx'015C <sub>H</sub>	U1C0_1IC	F12E <sub>H</sub>
USIC1 CH0 SR2	58 <sub>H</sub>	xx'0160 <sub>H</sub>	U1C0_2IC	F130 <sub>H</sub>
USIC1 CH1 SR0	59 <sub>H</sub>	xx'0164 <sub>H</sub>	U1C1_0IC	F132 <sub>H</sub>
USIC1 CH1 SR1	5A <sub>H</sub>	xx'0168 <sub>H</sub>	U1C1_1IC	F134 <sub>H</sub>
USIC1 CH1 SR2	5B <sub>H</sub>	xx'016C <sub>H</sub>	U1C1_2IC	F136 <sub>H</sub>
USIC2 CH0 SR0	5C <sub>H</sub>	xx'0170 <sub>H</sub>	U2C0_0IC	F138 <sub>H</sub>
USIC2 CH0 SR1	5D <sub>H</sub>	xx'0174 <sub>H</sub>	U2C0_1IC	F13A <sub>H</sub>
USIC2 CH0 SR2	5E <sub>H</sub>	xx'0178 <sub>H</sub>	U2C0_2IC	F13C <sub>H</sub>

**Interrupt and Exception Control**

**Table 7-9 XE16xyM Interrupt Nodes**

<b>Source of Interrupt or PEC Service Request</b>	<b>Trap Number</b>	<b>Vector<sup>1)</sup> Location</b>	<b>Control Register</b>	<b>Register Address</b>
USIC2 CH1 SR0	5F <sub>H</sub>	xx'017C <sub>H</sub>	U2C1_0IC	F13E <sub>H</sub>
USIC2 CH1 SR1	60 <sub>H</sub>	xx'0180 <sub>H</sub>	U2C1_1IC	F180 <sub>H</sub>
USIC2 CH1 SR2	61 <sub>H</sub>	xx'0184 <sub>H</sub>	U2C1_2IC	F182 <sub>H</sub>
USIC3 CH0 SR0	62 <sub>H</sub>	xx'0188 <sub>H</sub>	U3C0_0IC	F184 <sub>H</sub>
USIC3 CH0 SR1	63 <sub>H</sub>	xx'018C <sub>H</sub>	U3C0_1IC	F186 <sub>H</sub>
USIC3 CH0 SR2	64 <sub>H</sub>	xx'0190 <sub>H</sub>	U3C0_2IC	F188 <sub>H</sub>
USIC3 CH1 SR0	65 <sub>H</sub>	xx'0194 <sub>H</sub>	U3C1_0IC	F18A <sub>H</sub>
USIC3 CH1 SR1	66 <sub>H</sub>	xx'0198 <sub>H</sub>	U3C1_1IC	F18C <sub>H</sub>
USIC3 CH1 SR2	67 <sub>H</sub>	xx'019C <sub>H</sub>	U3C1_2IC	F18E <sub>H</sub>
SCU External Request 0	68 <sub>H</sub>	xx'01A0 <sub>H</sub>	ERU_0IC	F190 <sub>H</sub>
SCU External Request 1	69 <sub>H</sub>	xx'01A4 <sub>H</sub>	ERU_1IC	F192 <sub>H</sub>
SCU External Request 2	6A <sub>H</sub>	xx'01A8 <sub>H</sub>	ERU_2IC	F194 <sub>H</sub>
SCU Interrupt 1	6B <sub>H</sub>	xx'01AC <sub>H</sub>	SCU_1IC	F196 <sub>H</sub>
SCU Interrupt 0	6C <sub>H</sub>	xx'01B0 <sub>H</sub>	SCU_0IC	F198 <sub>H</sub>
SCU External Request 3	6D <sub>H</sub>	xx'01B4 <sub>H</sub>	ERU_3IC	F19A <sub>H</sub>
RTC	6E <sub>H</sub>	xx'01B8 <sub>H</sub>	RTC_IC	F19C <sub>H</sub>
End of PEC Subchannel	6F <sub>H</sub>	xx'01BC <sub>H</sub>	EOPIC	F19E <sub>H</sub>

<sup>1)</sup> Register VECSEG defines the segment where the vector table is located to.  
 Bitfield VECSC in register CPUCON1 defines the distance between two adjacent vectors. This table represents the default setting, with a distance of 4 (two words) between two vectors.

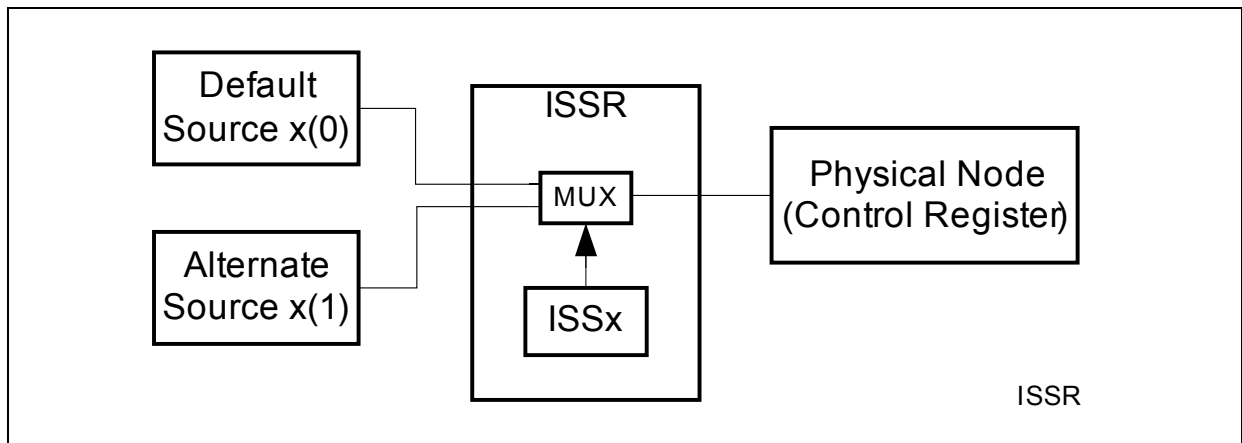
### 7.14.2 Interrupt Node Sharing

Interrupt source selection is adjustable to the application focus. The concept described in this chapter allows to adjust the focus to be more on control (CAPCOM2) or on communication (USIC) side.

Interrupt node sharing is controlled by SCU register ISSR (see [Section 7.14.3](#)).

#### Shared Nodes controlled by ISSR

The following figure visualizes the sharing principle controlled by ISSR register. The default interrupt source is CAPCOM2. The alternate source selections are the SCU interrupts 2 and 3 and the SR3 requests of all USIC channels.



**Figure 7-5 Node Sharing Principle controlled by ISSR**

The table below lists the possible selections of interrupt request sources to the physical interrupt nodes (Control Register). Selection is controlled through ISSx bits of the ISSR register.

**Table 7-10 Nodes Sharing controlled by ISSR**

Control Register	Select Bit	Default Source (ISSx=0)	Alternate Source (ISSx=1)
CC2_CC16IC	ISS0	CAPCOM2 Request 16	(not assigned)
CC2_CC17IC	ISS1	CAPCOM2 Request 17	(not assigned)
CC2_CC18IC	ISS2	CAPCOM2 Request 18	USIC3 CH0 SR3
CC2_CC19IC	ISS3	CAPCOM2 Request 19	USIC3 CH1 SR3
CC2_CC20IC	ISS4	CAPCOM2 Request 20	USIC0 CH0 SR3
CC2_CC21IC	ISS5	CAPCOM2 Request 21	USIC0 CH1 SR3
CC2_CC22IC	ISS6	CAPCOM2 Request 22	USIC1 CH0 SR3
CC2_CC23IC	ISS7	CAPCOM2 Request 23	USIC1 CH1 SR3

**Interrupt and Exception Control**

**Table 7-10 Nodes Sharing controlled by ISSR**

<b>Control Register</b>	<b>Select Bit</b>	<b>Default Source (ISSx=0)</b>	<b>Alternate Source (ISSx=1)</b>
CC2_CC24IC	ISS8	CAPCOM2 Request 24	(not assigned)
CC2_CC25IC	ISS9	CAPCOM2 Request 25	(not assigned)
CC2_CC26IC	ISS10	CAPCOM2 Request 26	(not assigned)
CC2_CC27IC	ISS11	CAPCOM2 Request 27	(not assigned)
CC2_CC28IC	ISS12	CAPCOM2 Request 28	USIC2 CH0 SR3
CC2_CC29IC	ISS13	CAPCOM2 Request 29	USIC2 CH1 SR3
CC2_CC30IC	ISS14	CAPCOM2 Request 30	SCU Interrupt 2
CC2_CC31IC	ISS15	CAPCOM2 Request 31	SCU Interrupt 3

### 7.14.3 Interrupt Source Select Registers

In order to map the interrupt request sources in the complete system to the available interrupt nodes, interrupt nodes are shared between selected modules.

#### SCU\_ISSR

#### Interrupt Source Select Register

**SFR (FF2E<sub>H</sub>/97<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>ISS 15</b>	<b>ISS 14</b>	<b>ISS 13</b>	<b>ISS 12</b>	<b>ISS 11</b>	<b>ISS 10</b>	<b>ISS 9</b>	<b>ISS 8</b>	<b>ISS 7</b>	<b>ISS 6</b>	<b>ISS 5</b>	<b>ISS 4</b>	<b>ISS 3</b>	<b>ISS 2</b>	<b>ISS 1</b>	<b>ISS 0</b>
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
<b>ISS0</b>	0	rw	<b>Interrupt Source Select for CC2_CC16IC</b> 0 <sub>B</sub> CC2 channel 16 interrupt is selected 1 <sub>B</sub> No interrupt source assigned
<b>ISS1</b>	1	rw	<b>Interrupt Source Select for CC2_CC17IC</b> 0 <sub>B</sub> CC2 channel 17 interrupt is selected 1 <sub>B</sub> No interrupt source assigned
<b>ISS2</b>	2	rw	<b>Interrupt Source Select for CC2_CC18IC</b> 0 <sub>B</sub> CC2 channel 18 interrupt is selected 1 <sub>B</sub> USIC3 channel 0 SR3 is selected
<b>ISS3</b>	3	rw	<b>Interrupt Source Select for CC2_CC19IC</b> 0 <sub>B</sub> CC2 channel 19 interrupt is selected 1 <sub>B</sub> USIC3 channel 1 SR3 is selected
<b>ISS4</b>	4	rw	<b>Interrupt Source Select for CC2_CC20IC</b> 0 <sub>B</sub> CC2 channel 20 interrupt is selected 1 <sub>B</sub> USIC0 channel 0 SR3 is selected
<b>ISS5</b>	5	rw	<b>Interrupt Source Select for CC2_CC21IC</b> 0 <sub>B</sub> CC2 channel 21 interrupt is selected 1 <sub>B</sub> USIC0 channel 1 SR3 is selected
<b>ISS6</b>	6	rw	<b>Interrupt Source Select for CC2_CC22IC</b> 0 <sub>B</sub> CC2 channel 22 interrupt is selected 1 <sub>B</sub> USIC1 channel 0 SR3 is selected
<b>ISS7</b>	7	rw	<b>Interrupt Source Select for CC2_CC23IC</b> 0 <sub>B</sub> CC2 channel 23 interrupt is selected 1 <sub>B</sub> USIC1 channel 1 SR3 is selected

**Interrupt and Exception Control**

<b>Field</b>	<b>Bits</b>	<b>Type</b>	<b>Description</b>
<b>ISS8</b>	8	rw	<b>Interrupt Source Select for CC2_CC24IC</b> 0 <sub>B</sub> CC2 channel 24 interrupt is selected 1 <sub>B</sub> No interrupt source assigned
<b>ISS9</b>	9	rw	<b>Interrupt Source Select for CC2_CC25IC</b> 0 <sub>B</sub> CC2 channel 25 interrupt is selected 1 <sub>B</sub> No interrupt source assigned
<b>ISS10</b>	10	rw	<b>Interrupt Source Select for CC2_CC26IC</b> 0 <sub>B</sub> CC2 channel 26 interrupt is selected 1 <sub>B</sub> No interrupt source assigned
<b>ISS11</b>	11	rw	<b>Interrupt Source Select for CC2_CC27IC</b> 0 <sub>B</sub> CC2 channel 27 interrupt is selected 1 <sub>B</sub> No interrupt source assigned
<b>ISS12</b>	12	rw	<b>Interrupt Source Select for CC2_CC28IC</b> 0 <sub>B</sub> CC2 channel 28 interrupt is selected 1 <sub>B</sub> USIC2 channel 0 SR3 is selected
<b>ISS13</b>	13	rw	<b>Interrupt Source Select for CC2_CC29IC</b> 0 <sub>B</sub> CC2 channel 29 interrupt is selected 1 <sub>B</sub> USIC2 channel 1 SR3 is selected
<b>ISS14</b>	14	rw	<b>Interrupt Source Select for CC2_CC30IC</b> 0 <sub>B</sub> CC2 channel 30 interrupt is selected 1 <sub>B</sub> SCU Interrupt 2 is selected
<b>ISS15</b>	15	rw	<b>Interrupt Source Select for CC2_CC31IC</b> 0 <sub>B</sub> CC2 channel 31 interrupt is selected 1 <sub>B</sub> SCU Interrupt 3 is selected

## 7.15 Interrupt and PEC Configuration Registers

The following table lists all registers used to configure the interrupt and PEC behavior of the XE16xyM. Registers are ordered by address. The Interrupt Control registers xxIC, assigned to each interrupt request, are listed separately (see [Section 7.14](#)).

**Bit addressable** SFRs are marked with the letter “b” in column “Name”.

**Table 7-11 Register Overview Interrupt and PEC - ordered by address**

Name	Physical Address	8-bit Address	Description	Reset Value
<b>FINT0CSP</b>	EC00 <sub>H</sub>	--	Fast Interrupt 0 CSP Register	0000 <sub>H</sub>
<b>FINT0ADDR</b>	EC02 <sub>H</sub>	--	Fast Interrupt 0 Address Register	0000 <sub>H</sub>
<b>FINT1CSP</b>	EC04 <sub>H</sub>	--	Fast Interrupt 1 CSP Register	0000 <sub>H</sub>
<b>FINT1ADDR</b>	EC06 <sub>H</sub>	--	Fast Interrupt 1 Address Register	0000 <sub>H</sub>
<b>BNKSEL0</b>	EC20 <sub>H</sub>	--	Bank Selection Register 0	0000 <sub>H</sub>
<b>BNKSEL1</b>	EC22 <sub>H</sub>	--	Bank Selection Register 1	0000 <sub>H</sub>
<b>BNKSEL2</b>	EC24 <sub>H</sub>	--	Bank Selection Register 2	0000 <sub>H</sub>
<b>BNKSEL3</b>	EC26 <sub>H</sub>	--	Bank Selection Register 3	0000 <sub>H</sub>
<b>SRCP0</b>	EC40 <sub>H</sub>	--	PEC Channel 0 Source Pointer	0000 <sub>H</sub>
<b>DSTP0</b>	EC42 <sub>H</sub>	--	PEC Channel 0 Destination Pointer	0000 <sub>H</sub>
<b>SRCP1</b>	EC44 <sub>H</sub>	--	PEC Channel 1 Source Pointer	0000 <sub>H</sub>
<b>DSTP1</b>	EC46 <sub>H</sub>	--	PEC Channel 1 Destination Pointer	0000 <sub>H</sub>
<b>SRCP2</b>	EC48 <sub>H</sub>	--	PEC Channel 2 Source Pointer	0000 <sub>H</sub>
<b>DSTP2</b>	EC4A <sub>H</sub>	--	PEC Channel 2 Destination Pointer	0000 <sub>H</sub>
<b>SRCP3</b>	EC4C <sub>H</sub>	--	PEC Channel 3 Source Pointer	0000 <sub>H</sub>
<b>DSTP3</b>	EC4E <sub>H</sub>	--	PEC Channel 3 Destination Pointer	0000 <sub>H</sub>
<b>SRCP4</b>	EC50 <sub>H</sub>	--	PEC Channel 4 Source Pointer	0000 <sub>H</sub>
<b>DSTP4</b>	EC52 <sub>H</sub>	--	PEC Channel 4 Destination Pointer	0000 <sub>H</sub>
<b>SRCP5</b>	EC54 <sub>H</sub>	--	PEC Channel 5 Source Pointer	0000 <sub>H</sub>
<b>DSTP5</b>	EC56 <sub>H</sub>	--	PEC Channel 5 Destination Pointer	0000 <sub>H</sub>
<b>SRCP6</b>	EC58 <sub>H</sub>	--	PEC Channel 6 Source Pointer	0000 <sub>H</sub>
<b>DSTP6</b>	EC5A <sub>H</sub>	--	PEC Channel 6 Destination Pointer	0000 <sub>H</sub>
<b>SRCP7</b>	EC5C <sub>H</sub>	--	PEC Channel 7 Source Pointer	0000 <sub>H</sub>
<b>DSTP7</b>	EC5E <sub>H</sub>	--	PEC Channel 7 Destination Pointer	0000 <sub>H</sub>



**Interrupt and Exception Control**

**Table 7-11 Register Overview Interrupt and PEC - ordered by address**

<b>Name</b>	<b>Physical Address</b>	<b>8-bit Address</b>	<b>Description</b>	<b>Reset Value</b>
<b>PECSEG0</b>	EC80 <sub>H</sub>	--	PEC Pointer 0 Segment Address	0000 <sub>H</sub>
<b>PECSEG1</b>	EC82 <sub>H</sub>	--	PEC Pointer 1 Segment Address	0000 <sub>H</sub>
<b>PECSEG2</b>	EC84 <sub>H</sub>	--	PEC Pointer 2 Segment Address	0000 <sub>H</sub>
<b>PECSEG3</b>	EC86 <sub>H</sub>	--	PEC Pointer 3 Segment Address	0000 <sub>H</sub>
<b>PECSEG4</b>	EC88 <sub>H</sub>	--	PEC Pointer 4 Segment Address	0000 <sub>H</sub>
<b>PECSEG5</b>	EC8A <sub>H</sub>	--	PEC Pointer 5 Segment Address	0000 <sub>H</sub>
<b>PECSEG6</b>	EC8C <sub>H</sub>	--	PEC Pointer 6 Segment Address	0000 <sub>H</sub>
<b>PECSEG7</b>	EC8E <sub>H</sub>	--	PEC Pointer 7 Segment Address	0000 <sub>H</sub>
<b>PECISNC</b> <b>b</b>	FFD8 <sub>H</sub>	EC <sub>H</sub>	PEC Interrupt Subnode Control	0000 <sub>H</sub>
<b>PECC0</b>	FEC0 <sub>H</sub>	60 <sub>H</sub>	PEC Channel 0 Control Register	0000 <sub>H</sub>
<b>PECC1</b>	FEC2 <sub>H</sub>	61 <sub>H</sub>	PEC Channel 1 Control Register	0000 <sub>H</sub>
<b>PECC2</b>	FEC4 <sub>H</sub>	62 <sub>H</sub>	PEC Channel 2 Control Register	0000 <sub>H</sub>
<b>PECC3</b>	FEC6 <sub>H</sub>	63 <sub>H</sub>	PEC Channel 3 Control Register	0000 <sub>H</sub>
<b>PECC4</b>	FEC8 <sub>H</sub>	64 <sub>H</sub>	PEC Channel 4 Control Register	0000 <sub>H</sub>
<b>PECC5</b>	FECA <sub>H</sub>	65 <sub>H</sub>	PEC Channel 5 Control Register	0000 <sub>H</sub>
<b>PECC6</b>	FECC <sub>H</sub>	66 <sub>H</sub>	PEC Channel 6 Control Register	0000 <sub>H</sub>
<b>PECC7</b>	FECE <sub>H</sub>	67 <sub>H</sub>	PEC Channel 7 Control Register	0000 <sub>H</sub>

## **8 System Control Unit (SCU)**

The System Control Unit (SCU) of the XE16xyM handles all system control tasks besides the debug related tasks which are controlled by the OCDS/Cerberus. All functions described in this chapter are tightly coupled, thus, they are conveniently handled by one unit, the SCU.

The SCU contains the following functional sub-blocks:

- Clock Generation (see [Chapter 8.1](#))
- System Timer (see [Chapter 8.2](#))
- Wake-up Timer (see [Chapter 8.3](#))
- Reset Operation (see [Chapter 8.4](#))
- External Service Requests (see [Chapter 8.5](#))
- Power Supply and Control (see [Chapter 8.6](#))
- Global State Control (see [Chapter 8.7](#))
- Software Boot Support (see [Chapter 8.8](#))
- External Request Unit (see [Chapter 8.9](#))
- Interrupt Generation (see [Chapter 8.10](#))
- Temperature Compensation (see [Chapter 8.11](#))
- Watchdog Timer (see [Chapter 8.12](#))
- Trap Generation (see [Chapter 8.13](#))
- Memory Content Protection (see [Chapter 8.14](#))
- Register Access Control (see [Chapter 8.15](#))
- Miscellaneous System Registers (see [Chapter 8.16](#))
- Implementation (see [Chapter 8.17](#))
- SCU Registers and Address map (see [Chapter 8.18](#))

### **Important Information: Register Programming**

Some of XE16xyM registers are initialized during the startup procedure with values different from their reset-content (defined into respective registerdescriptions). They are listed in section “Registers modified by the Startup Procedure”.

The System Control Unit contains special function registers, which can not be programmed in an arbitrary order in particular due to the usage of an internal voltage regulator. In order to prevent critical system conditions because of an improper setup and to provide means for easy and quick configuration and control of sensitive features such as power supply and clock generation, recommendations and examples for the programming sequence of the registers will be given in the Programmer’s Guide.

In particular the registers listed below have to be updated with care:

- Clock Generation Unit: WUOSCCON, HPOSCCON, PLLOSCCON, PLLCONx
- Power Supply: EVR1CON0, EVR1SET15VHP, EVRMCON0, EVRMSET15VHP,

PVC1CON0, PVCMCON0,  
SWDCON0

- System: SYSCON0

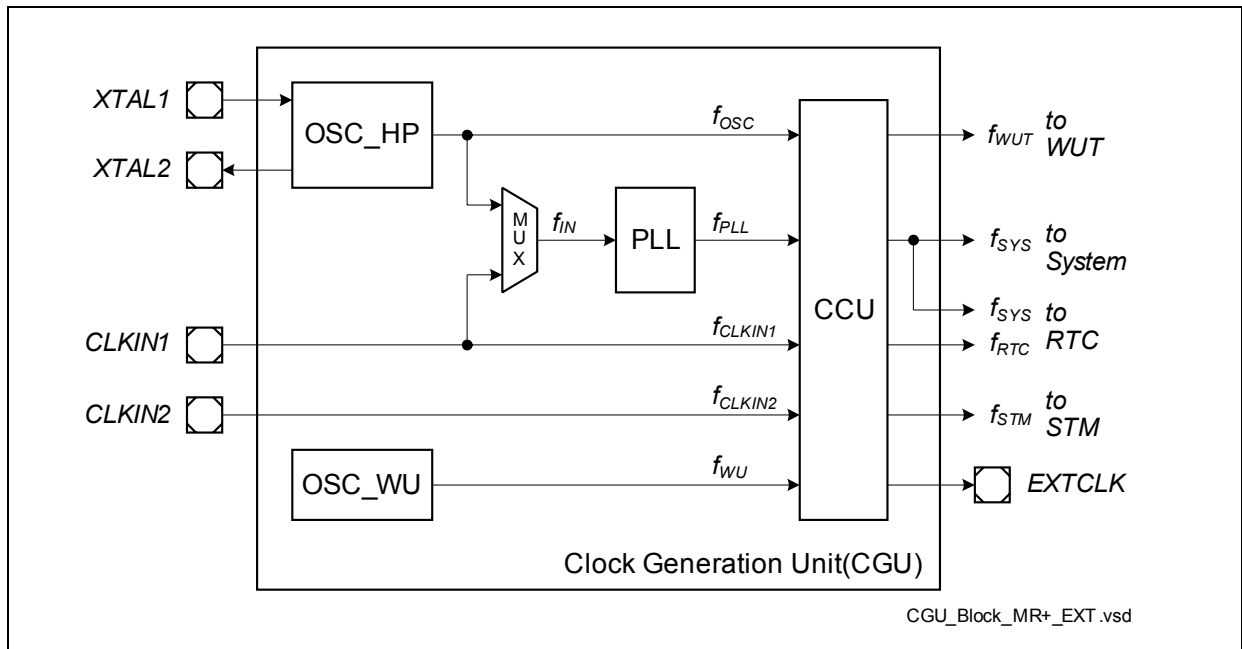
## 8.1 Clock Generation Unit

The Clock Generation Unit (CGU) allows a very flexible clock generation for the XE16xyM. During user program execution the frequency can be programmed for an optimal ratio between performance and power consumption in the actual application state.

### 8.1.1 Overview

The CGU can convert a low-frequency external clock to a high-speed system clock or can create a high-speed system clock without external input.

The CGU consists of a Clock Generator and a Clock Control Unit (CCU).



**Figure 8-1 Clock Generation Unit Block Diagram**

The input connections of the CGU are described in [Chapter 8.17.1](#).

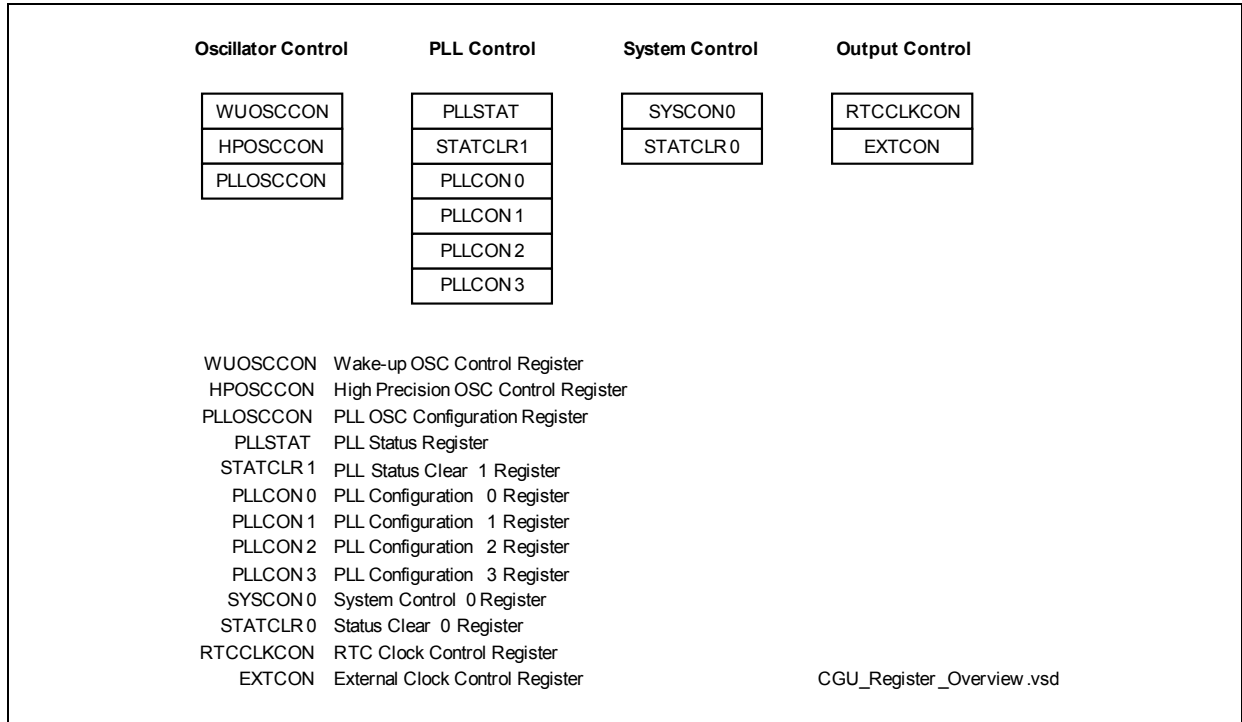
The following clock signals are generated:

- System clock  $f_{SYS}$
- RTC count clock  $f_{RTC}$
- Wake-Up Timer (WUT) clock  $f_{WUT}$
- STM clock  $f_{STM}$
- External clock  $f_{EXT}$

[Chapter 8.1.5](#) and [Chapter 8.1.6](#) describe which clock signals are generated out of which selectable clocks.

## Register Overview

The CGU is controlled by a number of registers shown in the following figure.



**Figure 8-2 Clock Generation Unit Register Overview**

The following sections describe the different parts of the CGU.

### 8.1.2 Trimmed Current Controlled Wake-Up Clock (OSC\_WU)

The trimmed current controlled wake-up clock source provides a clock to control internal operations independent of the standard clock supplies and requires no external components. Its output frequency  $f_{WU}$  is configured via bit field **WUOSCCON.FREQSEL** and has a typical range from 130 kHz to 500 kHz.

### 8.1.3 High Precision Oscillator Circuit (OSC\_HP)

The high precision oscillator circuit can drive an external crystal or accepts an external clock source. It consists of an inverting amplifier with XTAL1 as input, and XTAL2 as output.

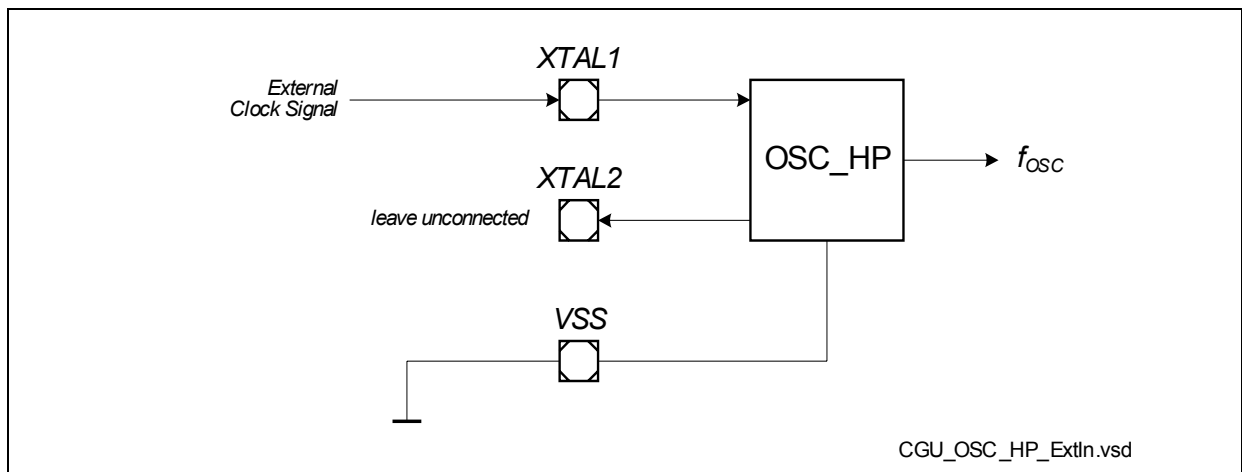
**Figure 8-4** and **Figure 8-3** show the recommended external circuitries for both operating modes, External Crystal Mode and External Input Clock Mode.

#### 8.1.3.1 External Input Clock Mode

An external clock signal is supplied directly not using an external crystal and bypassing the amplifier of the oscillator. The maximum allowed input frequency depends on the characteristics of pin XTAL1.

When using an external clock signal it must be connected to XTAL1. XTAL2 is left open (unconnected).

*Note: Voltages on XTAL1 must comply to the voltage defined in the data sheet.*

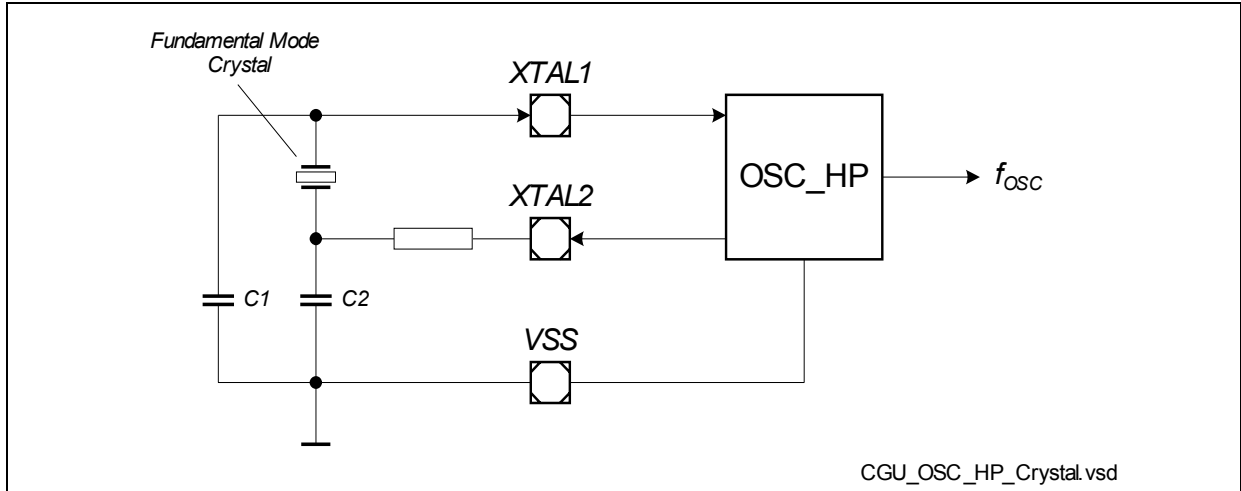


**Figure 8-3 XE16xyM External Clock Input Mode for the High-Precision Oscillator**

#### 8.1.3.2 External Crystal Mode

An external oscillator load circuitry must be used, connected to both pins, XTAL1 and XTAL2. It consists normally of the two load capacitances C1 and C2. For some crystals

a series damping resistor might be necessary. The exact values and related operating range depend on the crystal and have to be determined and optimized together with the crystal vendor using the negative resistance method.



**Figure 8-4 XE16xyM External Crystal Mode Circuitry for the High-Precision Oscillator**

### Support for Start-up Control of an External Crystal

The first time before the system clock is generated based on an external crystal 1000 cycles of the crystal clock should be waited before the clock control system is changed to External Crystal Mode. Bit **PLLSTAT.OSCLOCK** indicates if the oscillator OSC\_HP operates for at least  $2^{11}$  periods. Bit **PLLSTAT.OSCSTAB** indicates if OSC\_HP operates for at least  $2^{15}$  periods.

### Oscillator Gain Control

The oscillator starts with a high drive level (gain) during and after a Power-on Reset to ensure safe start-up behavior in the beginning (force the crystal oscillation). When a stable oscillation has been reached after oscillation start-up (**PLLSTAT.OSCSTAB** = 1), the gain of the oscillator can be reduced. This reduces the power consumption of the oscillator, which is especially important in the power saving modes. This gain reduction is selected by **HPOSCCON.GAINSEL**.

*Note: Choosing the gain setting is only possible with detailed consideration of parasitics, external circuitry, frequency range and quality of the applied crystal and has to be verified by testing together with the crystal manufacturer.*

## **8.1.4 Phase-Locked Loop (PLL) Module**

The PLL can convert a low-frequency external clock signal to a high-speed system clock for maximum performance. The PLL also has fail-safe logic that detects degenerate external clock behavior such as abnormal frequency deviations or a total loss of the external clock. It can execute emergency actions if it loses its lock on the external clock.

This module is a phase locked loop for integer frequency synthesis. It allows the use of input and output frequencies of a wide range by varying the different divider factors.

### **8.1.4.1 Features**

Here is a brief overview of the functions that are offered by the PLL.

- VCO lock detection
- 4-bit input divider **P**: (divide by PDIV+1)
- 6-bit feedback divider **N**: (multiply by NDIV+1)
- 10-bit output divider **K2**: (divide K2DIV+1)
- 10-bit VCO bypass divider **K1**: (divide by either by K1DIV+1)
- Oscillator run detection and Watchdog
- Different operating modes
  - Prescaler Mode
  - Unlocked Mode
  - Normal Mode
- Different power saving modes
  - Power Down
  - Sleep Mode (VCO Power Down)
- Glitchless programming of output divider K2 and VCO bypass divider K1
- Glitchless switching between Normal Mode and Prescaler Mode
- Trimmed current controlled clock source

### **8.1.4.2 PLL Functional Description**

The PLL consists of a Voltage Controlled Oscillator (VCO) with a feedback path. A divider in the feedback path (N-Divider) divides the VCO frequency. The resulting frequency is then compared with the divided external frequency (P-Divider). The phase detection logic determines the difference between the two clocks and accordingly controls the frequency of the VCO ( $f_{VCO}$ ). A PLL lock detection unit monitors and signals this condition. The phase detection logic continues to monitor the two clocks and adjusts the VCO clock if required. The PLL output clock  $f_{PLL}$  is derived from the VCO clock using the K2-Divider or from the oscillator clock using the K1-Divider.

The following figure shows the PLL block structure.





**In Prescaler Mode** the reference frequency  $f_R$  is divided by a factor K1. The output frequency is given by

(8.2)

$$f_{PLL} = \frac{f_R}{K1}$$

**In Unlocked Mode** the base output frequency of the Voltage Controlled Oscillator (VCO)  $f_{VCObase}$  is divided by a factor K2. The output frequency is given by

(8.3)

$$f_{PLL} = \frac{f_{VCObase}}{K2}$$

### **PLL Power Saving Modes**

**PLL Power Down Mode** The PLL offers a Power Down Mode to save power if the PLL is not needed at all. While the PLL is in Power Down Mode no PLL output frequency is generated.

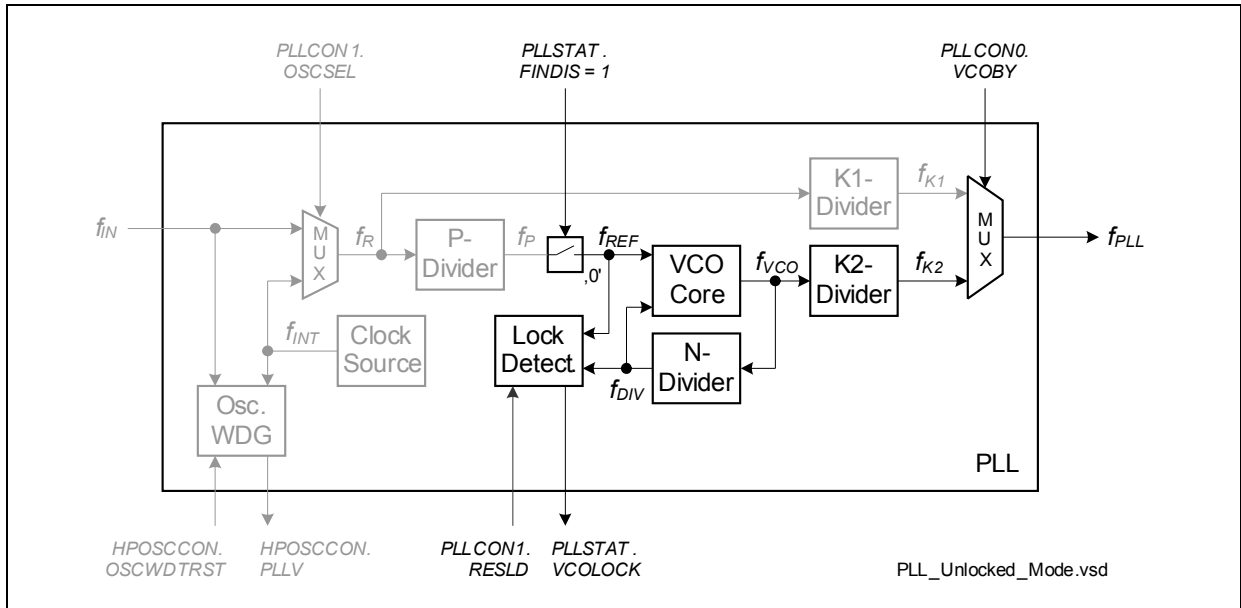
**PLL Sleep Mode** The PLL offers a Sleep Mode (also called VCO Power Down Mode) to save power within the PLL. While the PLL is in Sleep Mode only the Prescaler Mode can be used.

#### **8.1.4.3 Configuration and Operation of the PLL Modes**

The following section describes the configuration and the operation of the different PLL modes.

##### **Configuration and Operation of the Unlocked Mode**

In Unlocked Mode, the PLL is running at its VCO base frequency and  $f_{PLL}$  is derived from  $f_{VCO}$  by the K2-Divider.



**Figure 8-6 PLL Unlocked Mode Diagram**

The Unlocked Mode is selected by the following settings:

- STATCLR1.SETFINDIS = 1
- PLLCON0.VCOBY = 0

The Unlocked Mode is entered when all following conditions are true:

- PLLSTAT.FINDIS = 1
- PLLSTAT.VCOBYST = 1

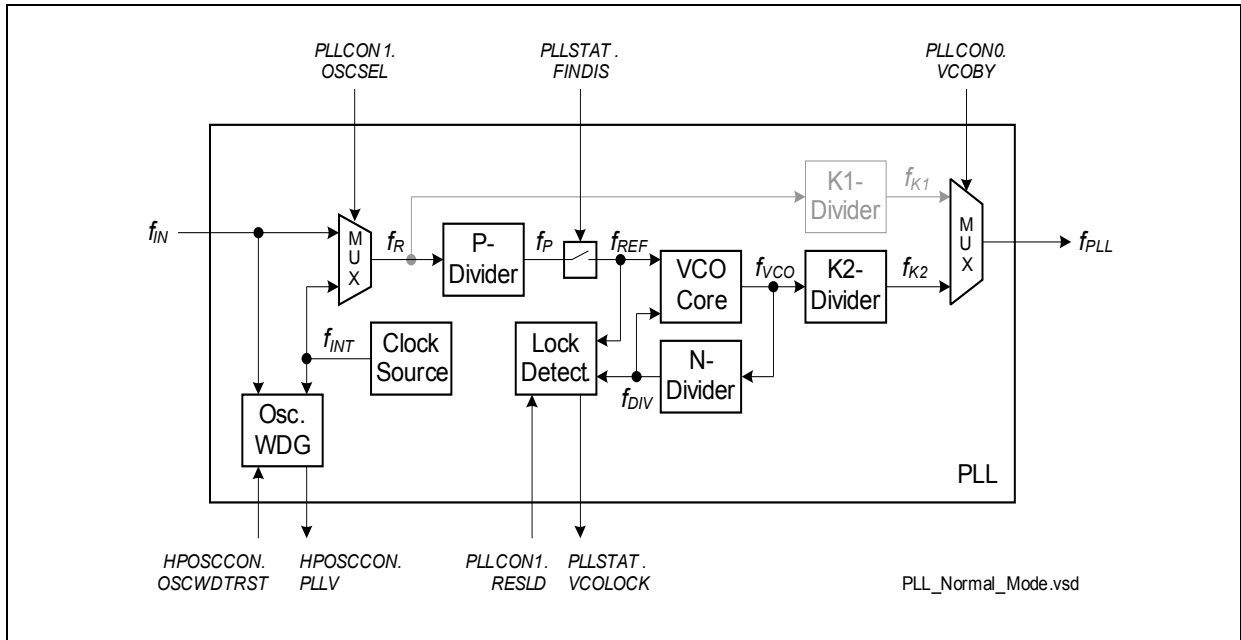
Operation in Unlocked Mode does not require an input clock  $f_{IN}$ . The Unlocked Mode is automatically entered on a PLL VCO Loss-of-Lock event if bit PLLCON1.EMFINDISEN is cleared. This mechanism allows a fail-safe operation of the PLL as in emergency cases still a clock is available.

The frequency of the Unlocked Mode  $f_{VCObase}$  is listed in the Data Sheet.

*Note: Changing the system operation frequency by changing the value of the K2-Divider or the VCO range has a direct influence on the power consumption of the device. Therefore, this has to be done carefully.*

### Configuration and Operation of the Normal Mode

In Normal Mode, the PLL is running at frequency  $f_{PLL}$ , where  $f_R$  is divided by a factor P, multiplied by a factor N and then divided by a factor K2.



**Figure 8-7 PLL Normal Mode Diagram**

The Normal Mode is selected by the following settings:

- PLLCON0.VCOBY = 0
- STATCLR1.CLRFINDIS = 1

The Normal Mode is entered when all following conditions are true:

- PLLSTAT.FINDIS = 0
- PLLSTAT.VCOBYST = 1
- PLLSTAT.VCOLOCK = 1
- HPOSCCON.PLLV = 1

Operation in Normal Mode requires a clock frequency of  $f_R$ . When  $f_{IN}$  is selected as source for  $f_R$  it is recommended to check and monitor if an input frequency  $f_R$  is available at all by checking HPOSCCON.PLLV.

The system operation frequency in Normal Mode is controlled by the values of the three dividers: P, N, and K2. A modification of the two dividers P and N has a direct influence on the VCO frequency and leads to a loss of the VCO Lock status. A modification of the K2-divider has no impact on the VCO Lock status but changes the PLL output frequency.

*Note: Changing the system operation frequency by changing the value of the K2-Divider has a direct influence on the power consumption of the device. Therefore, this has to be done carefully.*

To modify or enter the Normal Mode frequency, follow the sequence described below:

Configure and enter Prescaler Mode. For more details see the Prescaler Mode.

Disable the trap generation for the VCO Lost-of-Lock.

**System Control Unit (SCU)**

While the Prescaler Mode is used the Normal Mode can be configured and checked for a positive VCO Lock status. The first target frequency of the Normal Mode should be selected in a way that it matches or is only slightly higher as the one used in the Prescaler Mode. This avoids big changes in the system operation frequency and, therefore, the power consumption when switching later from Prescaler Mode to Normal Mode. The P and N dividers should be selected in the following way:

- Selecting P and N in a way that  $f_{VCO}$  is in the lower area of its allowed values leads to a slightly reduced power consumption but to a slightly increased jitter
- Selecting P and N in a way that  $f_{VCO}$  is in the upper area of its allowed values leads to a slightly increased power consumption but to a slightly reduced jitter

After the P, and N dividers are updated for the first configuration, the indication of the VCO Lock status (PLLSTAT.VCOLOCK = 1) should be awaited.

*Note: It is recommended to reset the VCO Lock detection (PLLCON1.RESLD = 1) after the new values of the dividers have been configured to get a defined VCO lock check time.*

When this happens the switch from Prescaler Mode to Normal Mode can be done. Normal Mode is requested by clearing PLLCON0.VCOBY. The Normal Mode is entered when the status bit PLLSTAT.VCOBYST is set.

Now the Normal Mode is entered. The trap status flag for the VCO Lock trap should be cleared and then enabled again.

The intended PLL output target frequency can be configured by changing only the K2-Divider. Depending on the selected divider value of the K2-Divider, the duty cycle of the clock is selected. This can have an impact on the operation with an external communication interface. In order to avoid too big frequency changes it might be necessary to change the K2-Divider in multiple steps. When the value of the K2-Divider was changed the next update of this value should not be done before bit PLLSTAT.K2RDY is set.

*Note: The Programmers's Guide describes a smooth frequency stepping to achieve an appropriate load regulation of the internal voltage regulator.*

**PLL VCO Lock Detection**

The PLL has a lock detection that supervises the VCO part of the PLL in order to detect instable VCO circuit behavior. The lock detector marks the VCO circuit and therefore the output  $f_{VCO}$  of the VCO as instable if the two inputs  $f_{REF}$  and  $f_{DIV}$  differ too much. Changes in one or both input frequencies below a level are not marked by a loss of lock because the VCO can handle such small changes without any problem for the system.

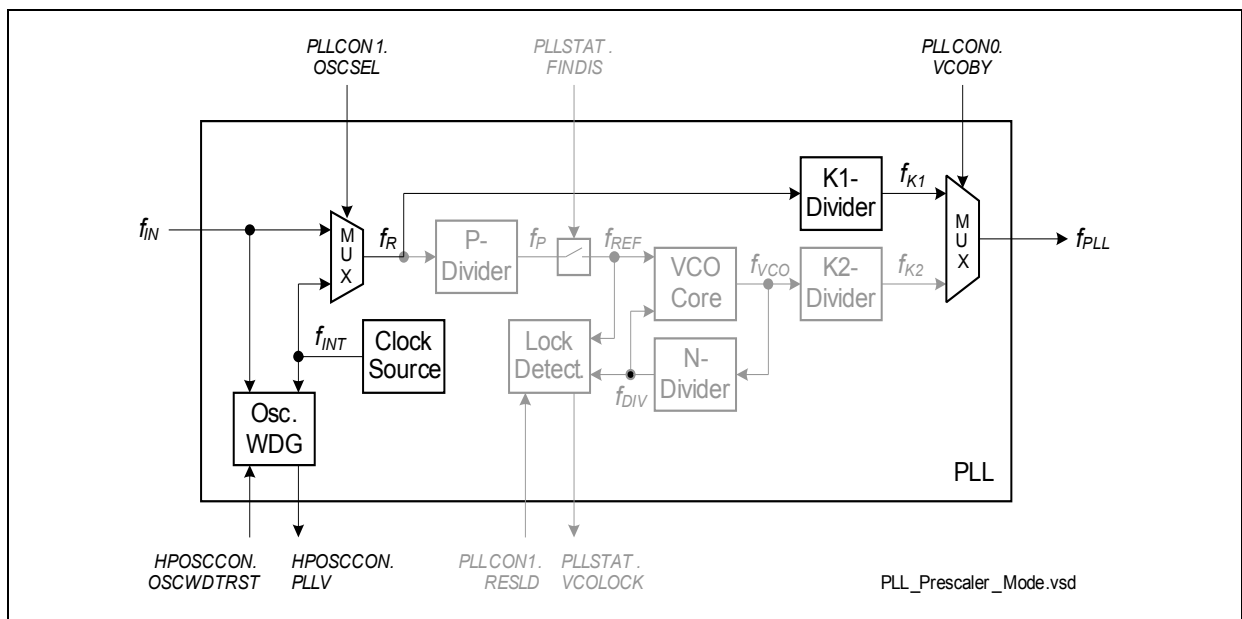
**PLL VCO Loss-of-Lock Event**

The PLL may become unlocked, caused by a break of the crystal or the external clock line. In such a case, a trap is generated if the according trap is enabled. Additionally, the clock  $f_R$  is disconnected from the PLL VCO to avoid unstable operation due to noise or

sporadic clock pulses coming from the oscillator circuit. Without a clock input  $f_R$ , the PLL gradually slows down to its VCO base frequency and remains there. The automatic disconnection of the VCO from its input clock  $f_R$  in case of a VCO Loss-of-Lock event can be enabled by setting bit PLLCON1.EMFINDISEN. If this bit is cleared the clock  $f_R$  remains connected to the VCO.

### Configuration and Operation of the Prescaler Mode

In Prescaler Mode, the PLL is running at frequency  $f_{PLL}$ , where  $f_R$  is divided by the K1-Divider.



**Figure 8-8 PLL Prescaler Mode Diagram**

The Prescaler Mode is selected by the following setting:

- PLLCON0.VCOBY = 1

The Prescaler Mode is entered when all following conditions are true:

- PLLSTAT.VCOBYST = 0
- HPOSCCON.PLLV = 1

Operation in Prescaler Mode requires an input clock frequency  $f_R$ . If  $f_{IN}$  is selected as clock source for  $f_R$  it is recommended to check and monitor if an input frequency  $f_{OSC}$  is available at all by checking HPOSCCON.PLLV. There are no requirements regarding the frequency of  $f_R$ .

The system operation frequency in Prescaler Mode is controlled by the value of the K1-Divider. When the value of PLLCON1.K1DIV was changed the next update of this value should not be done before bit PLLSTAT.K1RDY is set.

*Note: Changing the system operation frequency by changing the value of the K1-Divider has a direct influence on the power consumption of the device. Therefore, this has to be done carefully.*

The duty cycle of the clock signal depends on the selected value of the K1-Divider. This can have an impact for the operation with an external communication interface.

The Prescaler Mode is requested from the Unlocked or Normal Mode by setting bit PLLCON0.VCOBY. The Prescaler Mode is entered when the status bit PLLSTAT.VCOBYST is cleared.

Before the Prescaler Mode is requested the K1-Divider should be configured with a value generating a PLL output frequency  $f_{PLL}$  that matches the one generated by the Unlocked or Normal Mode as much as possible. In this way the frequency change resulting out of the mode change is reduced to a minimum.

The Prescaler Mode is requested to be left by clearing bit PLLCON0.VCOBY. The Prescaler Mode is left when the status bit PLLSTAT.VCOBYST is set.

### **Configuration and Operation of the PLL Power Down Mode**

The Power Down Mode is entered by setting bit PLLCON0.PLLPWD. While the PLL is in Power Down Mode no PLL output frequency is generated.

### **Configuration and Operation of the PLL Sleep Mode**

The Sleep Mode (also called VCO Power Down Mode) is entered by setting bit PLLCON0.VCOPWD. While the PLL is in Sleep Mode only the Prescaler Mode is operable. Selecting the Sleep Mode does not automatically switch to the Prescaler Mode. Therefore, before the Sleep Mode is entered the Prescaler Mode must be active.

#### **8.1.4.4 Power Regulator**

The analog parts of the PLL (VCO, trimmed current controlled clock source) are running on a dedicated supply generated by a dedicated regulator integrated within the PLL unit.

The regulator has to be enabled separately before the analog blocks of the PLL are activated, i.e. trimmed current controlled clock source and VCO must be kept off until the supply is stable. After activation, the PLL regulator will need its ramp-up time to properly ramp-up the analog PLL supply.

When the regulator shall be disabled in conjunction with a power down of the PLL digital part, it has to be taken into account that the digital part needs an active clock at the output of the PLL to ramp down. In case this clock is generated by one of the PLL oscillators, power down of PLL must be entered before the regulator is disabled. VCO and trimmed current controlled clock source may be activated or switched off together.

#### **8.1.4.5 Divider Handshake**

The PLL provides several handshake interfaces for dividers. This section describes how a handshake is to be conducted upon a change of configuration.

The general conduction of the handshake is the same for all interfaces. However, a sample sequence is described here in conjunction with re-programming of a divider.

*Note: The described handshake only works if the new setting (e.g. divider value) changes the current value upon the handshake.*

The handshake should be done in the the following steps:

1. Clear acknowledge bit together with setting the new divider value
2. Poll on ready bit to be 0
3. Set acknowledge bit
4. Poll on ready bit to be 1

This approach will even work in case the handshake has not been properly served before, and ready is already at 0 from the beginning. In any case, a change of the divider value will set ready to 0.

#### **8.1.4.6 Trimmed Current Controlled Clock**

The trimmed current controlled clock source provides a clock  $f_{INT}$  for the PLL.

*Note: The clock  $f_{INT}$  is also required for the operation of the oscillator watchdog.*

#### **8.1.4.7 Input Clock Selection**

The reference clock  $f_R$  can be provided by the PLL input clock source  $f_{IN}$  or by the trimmed current controlled clock source  $f_{INT}$ . This is selected via bit **PLLCON1.OSCSEL**.

The PLL input clock  $f_{IN}$  can be selected to be either taken from the high-precision oscillator clock source  $f_{OSC}$  or from the direct clock input  $f_{CLKIN1}$ . This is configured by **PLLCON1.INSEL**.

#### **8.1.4.8 Oscillator Watchdog**

The oscillator watchdog continuously monitors the input clock  $f_{IN}$ . If the input frequency becomes too low or if the input clock fails, this oscillator fail condition is indicated by **HPOSCCON.PLLV** = 0 and an interrupt request is generated.

By setting bit **HPOSCCON.OSCWDTRST** the detection can be restarted without a reset of the complete PLL, e.g. in case of a VCO loss-of-lock condition.

*Note: The oscillator watchdog requires the trimmed current controlled clock  $f_{INT}$  as a reference. Therefore, it can only be used (**HPOSCCON.PLLV** is valid) while the clock source is active.*



#### **8.1.4.9 Switching PLL Parameters**

The following restriction applies when changing PLL parameters inside the PLLCON0 to PLLCON3 registers:

- The VCO bypass switch may be used at any time, however, it has to be ensured that the maximum operating frequency of the device (see data sheet) will not be exceeded.
- Prescaler Mode should be selected.
- After switching to Prescaler Mode, NDIV and PDIV can be adjusted.
- Before deselecting the Prescaler Mode, the RESLD bit has to be set and then the VCOLOCK flag has to be checked. Only when the VCOLOCK flag is set again, the Prescaler Mode may be deselected.
- Before changing VCOSEL, the Prescaler Mode must be selected.

*Note: PDIV and NDIV can also be switched in Normal Mode. When changing NDIV, it must be regarded that the VCO clock  $f_{VCO}$  may exceed the target frequency until the PLL becomes locked. After changing PDIV or NDIV, it must be waited for the PLL lock condition. This procedure is typically used for increasing the VCO clock step-by-step.*

### **8.1.5 Clock Control Unit**

The Clock Control Unit selects the current clock sources for the clock signals used in the XE16xyM. It generates the following clocks:

- System clock  $f_{\text{SYS}}$
- RTC count clock  $f_{\text{RTC}}$
- WUT clock  $f_{\text{WUT}}$
- System timer clock  $f_{\text{STM}}$
- Output clock  $f_{\text{EXT}}$

The following clock signals can be selected:

- PLL clock  $f_{\text{PLL}}$
- The oscillator clock (OSC\_HP)  $f_{\text{OSC}}$
- Wake-up clock  $f_{\text{WU}}$
- Input CLKIN1 as Direct Clock Input  $f_{\text{CLKIN1}}$
- Input CLKIN2 as Direct Clock Input  $f_{\text{CLKIN2}}$

#### **8.1.5.1 Clock Generation**

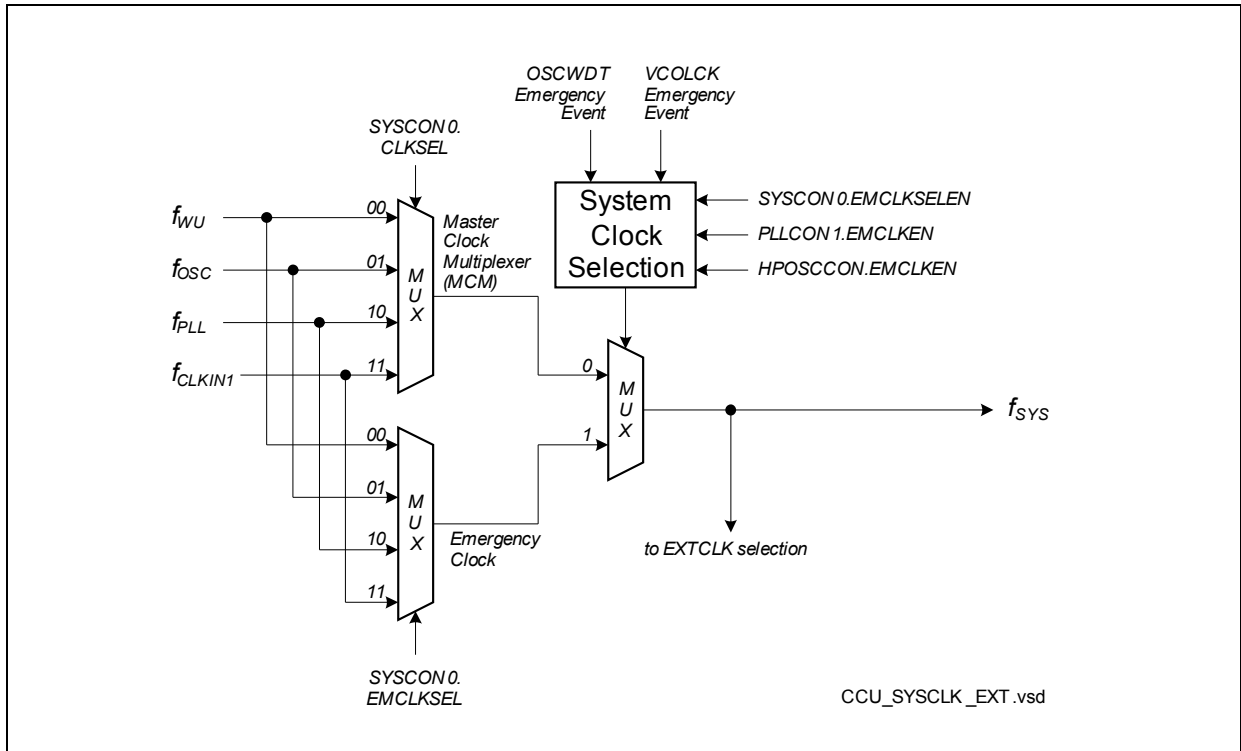
Different clock sources can be selected for the generated clock signals.

*Note: The selected clock sources are affected by the start-up procedure. See chapter Device Status after Start-up for the register values set by the different start-up procedures.*

#### **System Clock Generation**

The system clock  $f_{\text{SYS}}$  can be selected from the following clock sources in the CCU:

- Wake-up clock  $f_{\text{WU}}$
- The oscillator clock (OSC\_HP)  $f_{\text{OSC}}$
- PLL clock  $f_{\text{PLL}}$
- Input CLKIN1 as Direct Clock Input  $f_{\text{CLKIN1}}$

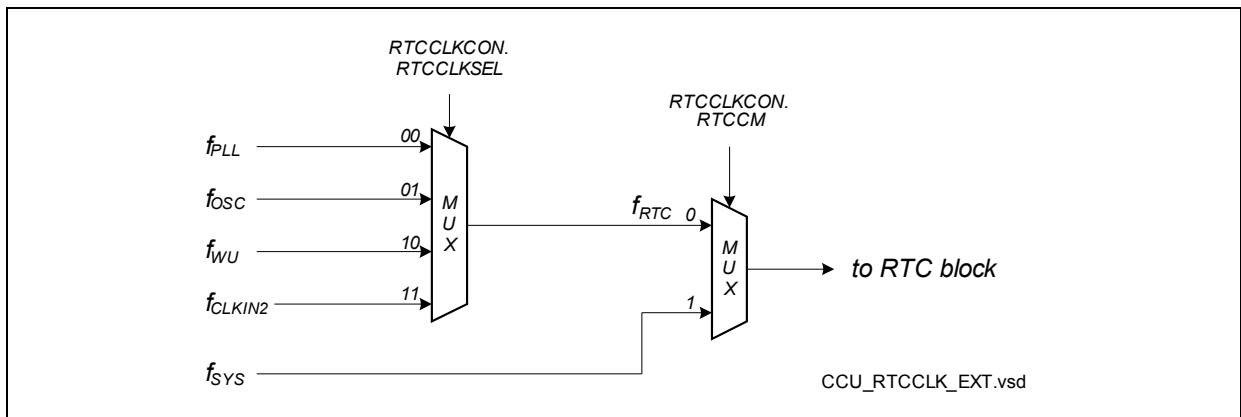


**Figure 8-9 Clock Control Unit, System Clock Generation**

### RTC Clock Generation

For the RTC module it is possible to select the operation in synchronous or asynchronous mode in the module itself. The asynchronous clock for the RTC can be selected out of following clock sources in the CCU:

- PLL clock  $f_{PLL}$
- The oscillator clock (OSC\_HP)  $f_{OSC}$
- Input CLKIN2 as Direct Clock Input  $f_{CLKIN2}$
- Wake-up clock  $f_{WU}$



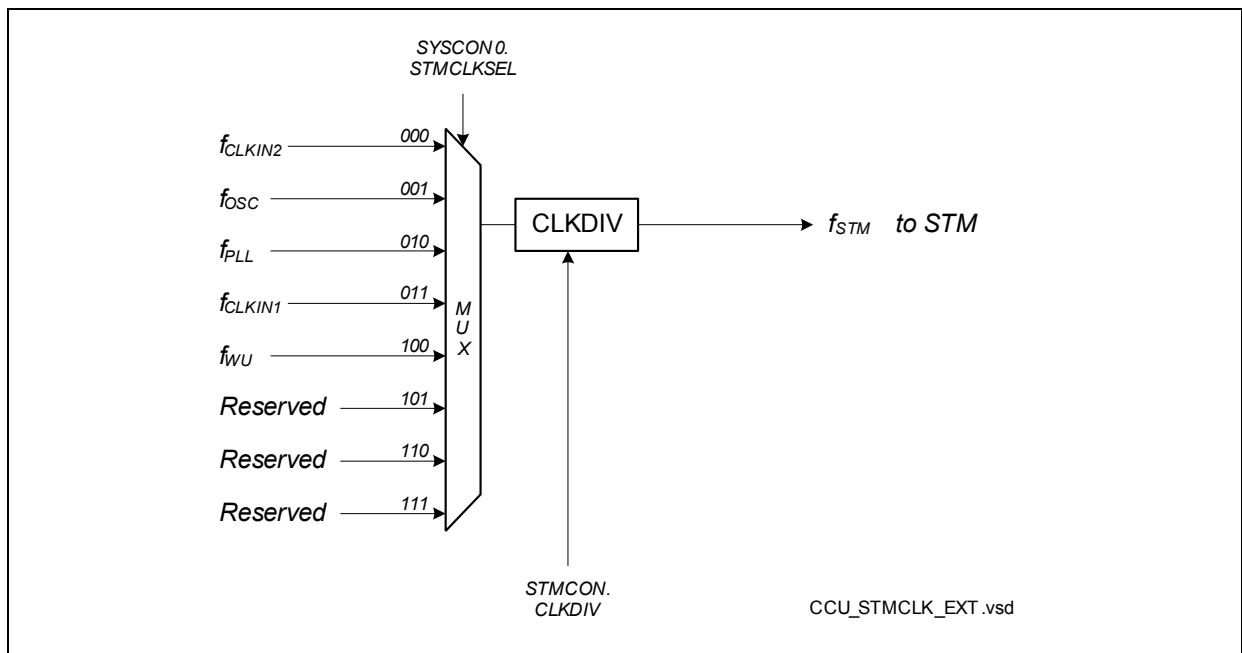
**Figure 8-10 Clock Control Unit, RTC Clock Generation**

### System Timer (STM) Clock Generation

The system timer clock can be selected out of following clock sources:

- The Direct Clock from oscillator OSC\_HP  $f_{OSC}$
- PLL clock  $f_{PLL}$
- Input CLKIN1 as Direct Clock Input  $f_{CLKIN1}$
- Input CLKIN2 as Direct Clock Input  $f_{CLKIN2}$
- Wake-up clock  $f_{WU}$

Then the selected clock can be divided by the factor defined in **STMCON.CLKDIV** (see **Chapter 8.2.1.2**).



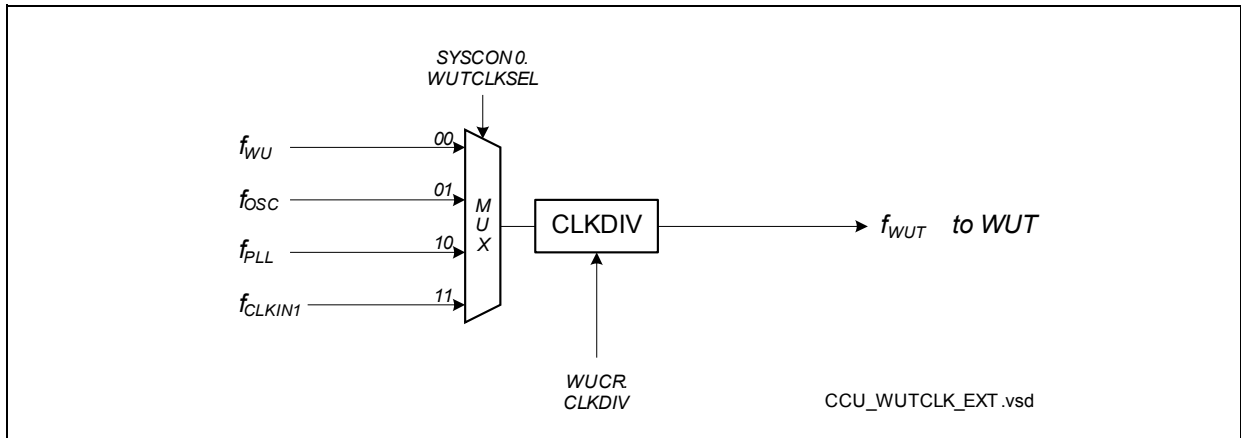
**Figure 8-11 Clock Control Unit, STM Clock Generation**

### Wake-up Timer (WUT) Clock Generation

The wake-up timer clock can be selected out of following clock sources in the CCU:

- Wake-up clock  $f_{WU}$
- The Direct Clock from oscillator OSC\_HP  $f_{OSC}$
- PLL clock  $f_{PLL}$
- Input CLKIN1 as Direct Clock Input  $f_{CLKIN1}$

Then the selected clock can be divided by the factor defined in **WUCR.CLKDIV** (see [Chapter 8.3.2.2](#)).



**Figure 8-12 Clock Control Unit, WUT Clock Generation**

### 8.1.5.2 Selecting and Changing the Operating Frequency

When selecting the clock source and the clock generation method, the required parameters must be carefully written to the respective bit fields, to avoid unintended intermediate states.

Many applications change the frequency of the system clock  $f_{SYS}$  during operation to optimize performance and power consumption of the system. Modifying the operating frequency changes the consumed switching current, which influences the power supply. Therefore, while the core voltage is generated by the on-chip Embedded Voltage Regulators (EVRs), the operating frequency may only be changed according to the rules given in the data sheet.

*Note: To avoid the indicated problems, specific sequences are recommended that ensure the intended operation of the clock system interacting with the power system. Please refer to the document Programmer's Guide.*

### 8.1.5.3 System Clock Emergency Handling

The generation of the system clock  $f_{SYS}$  can be affected, if either the PLL is no more locked to its input signal  $f_{IN}$ , or if the input clock  $f_{IN}$  is no more active. Both events can be detected and are indicated to the application software. The clock system takes appropriate actions where necessary, so the device and the application is never left without an alternate clock signal.

#### Oscillator Watchdog Event

If the clock frequency of the external source drops below a limit value the oscillator watchdog (OSCWDT) (see [Chapter 8.1.4.8](#)) then the clock source for the system clock  $f_{SYS}$  is switched to an alternate clock source, if enabled (HPOSCCON.EMCLKEN = 1). In this case following information is available:

- The oscillator watchdog trap flag (TRAPSTAT.OSCWDTT) is set and a trap request to the CPU is activated, if enabled (TRAPDIS.OSCWDTT = 0).
- Bit HPOSCCON.PLLV = 0, while the clock  $f_{IN}$  is missing
- Bit SYSCON0.EMSOSC is set, if SYSCON0.EMCLKSELEN is set
- The source of the system clock  $f_{SYS}$  is switched to alternate clock source selected by SYSCON0.EMCLKSEL, if enabled (SYSCON0.EMCLKSELEN = 1). This is indicated by bit SYSCON0.SELSTAT = 1.

#### PLL VCO Loss-of-Lock Event

If the PLL output frequency is no longer locked to its input frequency  $f_{IN}$ , the PLL switches from PLL Normal mode to the Unlocked mode, if enabled (PLLCON1.EMFINDISEN = 1). In this case following information is available:

- The PLL VCO loss of lock trap flag (TRAPSTAT.VCOLCKT) is set and a trap request to the CPU is activated, if enabled (TRAPDIS.VCOLCKT = 0).

**System Control Unit (SCU)**

- Bit PLLSTAT.VCOLOCK = 0, while the PLL is not locked
- Bit SYSCON0.EMSVCO is set, if SYSCON0.EMCLKSELEN is set
- The PLL VCO clock input is disconnected (PLLSTAT.FINDIS = 1) and the PLL clock slows down to its VCO base frequency.

**System Behavior**

Emergency routines can be executed with the alternate clock (emergency clock or VCO base frequency). The application can then enter a safe status and stop operation, or it can switch to an emergency operating mode, where a reduced performance and/or feature set is provided.

The Programmer's Guide describes both, how to enable these features, and how to react properly on each of the two events.

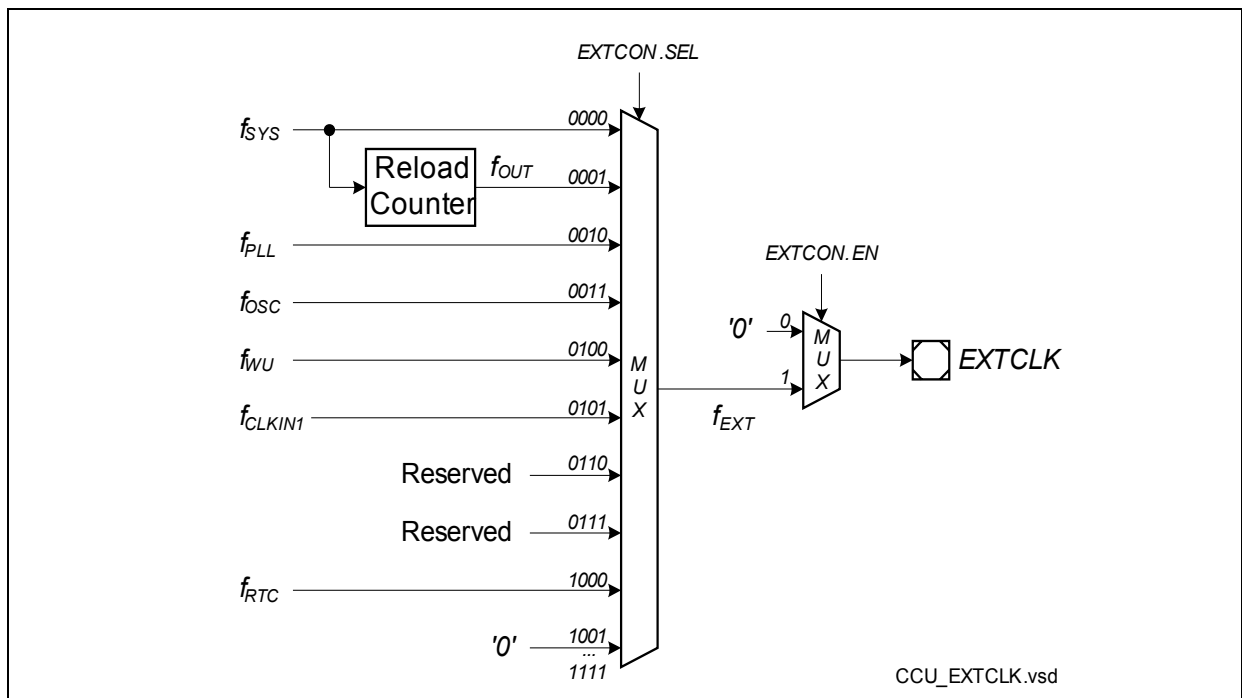


## 8.1.6 External Clock Output

An external clock output can be provided via pin EXTCLK to clock an external system or to observe one of the selectable device clocks. This external clock is enabled by setting bit EXTCON.EN and by selecting the clock signal as alternate output function at pin EXTCLK. Following clocks can be selected by EXTCON.SEL for external clock  $f_{EXT}$ :

- System clock  $f_{SYS}$
- Programmable clock output  $f_{OUT}$
- Direct Clock from oscillator OSC\_HP  $f_{OSC}$
- Direct Clock Input  $f_{CLKIN1}$
- PLL clock  $f_{PLL}$
- Wake-up clock  $f_{WU}$
- RTC clock  $f_{RTC}$

*Note: Changing bit field EXTCON.SEL can lead to spikes at pin EXTCLK.*

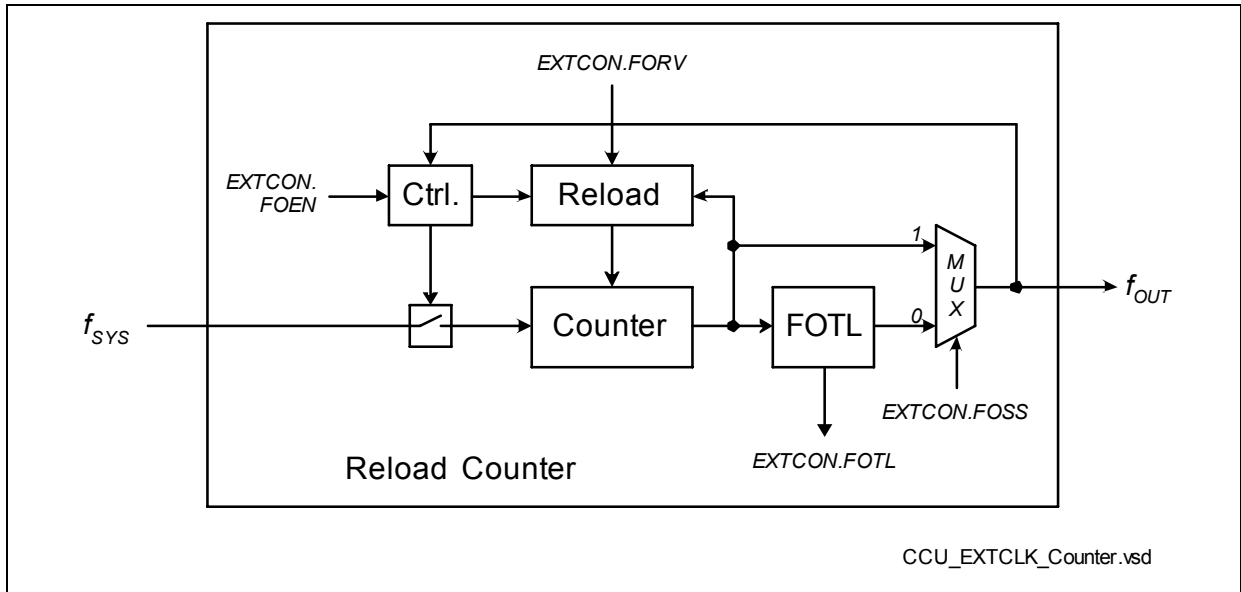


**Figure 8-13 EXTCLK Generation**

### 8.1.6.1 Programmable Frequency Output

The programmable frequency output  $f_{OUT}$  can be selected as clock output (EXTCLK). This clock can be controlled via software, and so can be adapted to the requirements of the connected external circuitry. The programmability also extends the power management to a system level, as also circuitry (peripherals, etc.) outside the XE16xyM can be run at a scalable frequency or can temporarily be left without a clock.

Clock  $f_{OUT}$  is generated via a reload counter, so the output frequency can be selected in small steps.

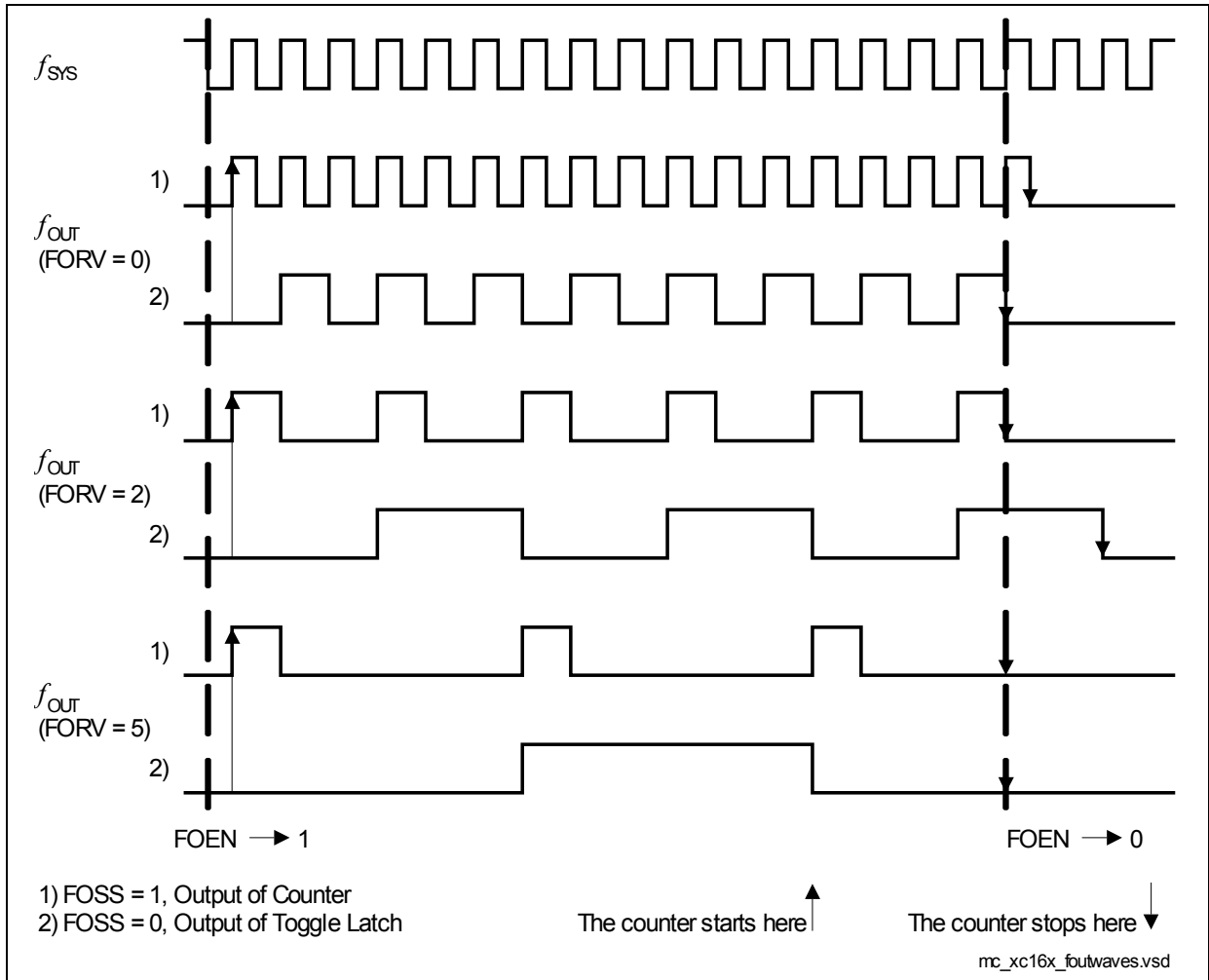


**Figure 8-14 Programmable Frequency Output Generation**

$f_{OUT}$  always provides complete output periods (provided  $f_{SYS}$  is available):

- When  $f_{OUT}$  is started (EXTCON.FOEN is set) counter FOCNT is loaded from EXTCON.FORV
- When OUT is stopped (EXTCON.FOEN is cleared) counter FOCNT is stopped when  $f_{OUT}$  has reached (or is) '0'.

Register EXTCON provides control over the output generation (frequency, waveform, activation) as well as all status information (EXTCON.FOTL).



**Figure 8-15 Output Waveforms Examples**

*Note: The output (for  $EXTCON.FOSS=1$ ) is high for the duration of one  $f_{SYS}$  cycle for all reload values  $EXTCON.FORV > 0$ . For  $EXTCON.FORV = 0$  the output frequency corresponds to  $f_{SYS}$ .  
 When a reference clock is required (e.g. for the bus interface),  $f_{SYS}$  must be selected directly.*

## 8.1.7 CGU Registers

### 8.1.7.1 Wake-up Clock Register

This register controls the settings of OSC\_WU.

#### **WUOSCCON**

**Wake-up OSC Control Register ESFR (F1AE<sub>H</sub>/D7<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

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

Field	Bits	Type	Description
<b>FREQSEL</b>	[1:0]	rw	<b>Frequency Selection</b> The values for the different settings are listed in the data sheet. <i>Note: This value must not be changed while <math>f_{WU}</math> is used as clock source for any logic.</i>
<b>0</b>	[3:2]	rw	<b>Reserved</b> Do not change this value when writing to this register
<b>DIS</b>	4	rw	<b>Clock Disable</b> 0 <sub>B</sub> The oscillator is switched on and the clock is enabled 1 <sub>B</sub> The oscillator is switched off and the clock is disabled
<b>0</b>	[15:5]	r	<b>Reserved</b> Read as 0; should be written with 0.

### 8.1.7.2 High Precision Oscillator Register

This register controls the setting of OSC\_HP.

#### HPOSCCON

#### High Precision OSC Control Register

**ESFR (F1B4<sub>H</sub>/DA<sub>H</sub>)**

**Reset Value: 053C<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0			OSC 2 L0	OSC 2 L1	EM FIN DIS EN	EM CLK EN	SH BY	X1D EN	X1D	GAINSEL		MODE		OSC WDT RST	PLL V
r			rh	rh	rw	rw	rw	rw	rh	rw		rw		w	rh

Field	Bits	Type	Description
<b>PLL V</b>	0	rh	<b>Oscillator for PLL Valid Status Bit</b> This bit indicates whether the frequency output of OSC_HP is usable. This is checked by the Oscillator Watchdog of the PLL. 0 <sub>B</sub> The OSC_HP frequency is not usable. The frequency is below the limit. 1 <sub>B</sub> The OSC_HP frequency is usable. The frequency is not below the limit. For more information see <a href="#">Chapter 8.1.4.8</a> .
<b>OSCWDTRST</b>	1	w	<b>Oscillator Watchdog Reset</b> 0 <sub>B</sub> No action 1 <sub>B</sub> The Oscillator Watchdog of the PLL is reset and restarted <i>Note: This bit is always read as 0.</i>
<b>MODE</b>	[3:2]	rw	<b>Oscillator Mode</b> 00 <sub>B</sub> The oscillator is active (External Crystal Mode) 01 <sub>B</sub> Reserved, do not use 10 <sub>B</sub> External Input Clock Mode; oscillator is in power-saving mode 11 <sub>B</sub> OSC_HP is disabled and in power-saving mode

Field	Bits	Type	Description
<b>GAINSEL</b>	[5:4]	rw	<b>Oscillator Gain Selection</b> 00 <sub>B</sub> Supply current is typically 300 µA (not tested) 01 <sub>B</sub> Supply current is typically 530 µA (not tested) 10 <sub>B</sub> Supply current is typically 450 µA (not tested) 11 <sub>B</sub> Supply current is typically 610 µA (not tested)
<b>X1D</b>	6	rh	<b>XTAL1 Data Value</b> This bit reflects the inverted level of pin XTAL1. This bit is sampled with $f_{SYS}$ while X1DEN is set. <i>Note: Voltages on XTAL1 must comply to the voltage defined in the data sheet.</i>
<b>X1DEN</b>	7	rw	<b>XTAL1 Data Enable</b> 0 <sub>B</sub> Bit X1D is not updated 1 <sub>B</sub> Bit X1D can be updated
<b>SHBY</b>	8	rw	<b>Shaper Bypass</b> The shaper forms a proper signal from the input signal. This bit must be 0 for proper operation. 0 <sub>B</sub> The shaper is not bypassed 1 <sub>B</sub> The shaper is bypassed
<b>EMCLKEN</b>	9	rw	<b>OSCWDT Emergency System Clock Source Select Enable</b> This bit requests the master clock multiplexer (MCM) to switch to an alternate clock (selected by bit field SYSCON0.EMCLKSEL) in an OSCWDT emergency case. 0 <sub>B</sub> MCM remains controlled by SYSCON0.CLKSEL 1 <sub>B</sub> MCM is controlled by SYSCON0.EMCLKSEL
<b>EMFINDISEN</b>	10	rw	<b>Emergency Input Clock Disconnect Enable</b> This bit defines if bit PLLSTAT.FINDIS is set in an OSCWDT emergency case. 0 <sub>B</sub> No action 1 <sub>B</sub> PLLSTAT.FINDIS is set in an emergency case <i>Note: Please refer to the Programmer's Guide for a description of the proper handling.</i>

**System Control Unit (SCU)**

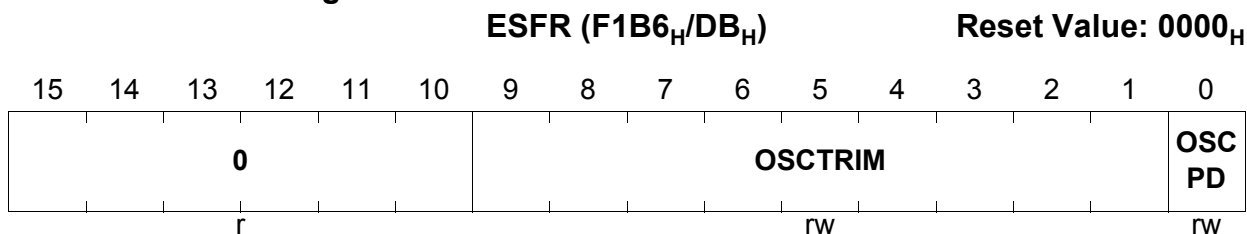
Field	Bits	Type	Description
<b>OSC2L1</b>	11	rh	<b>OSC_HP Not Usable Frequency Event</b> This sticky bit indicates if bit PLLV has been cleared since OSC2L1 has last been cleared (by writing 1 to bit STATCLR1.OSC2L1CLR). 0 <sub>B</sub> No change of PLLV detected 1 <sub>B</sub> Bit PLLV has been cleared at least once
<b>OSC2L0</b>	12	rh	<b>OSC_HP Usable Frequency Event</b> This sticky bit indicates if bit PLLV has been set since OSC2L0 has last been cleared (by writing 1 to bit STATCLR1.OSC2L0CLR). 0 <sub>B</sub> No change of PLLV detected 1 <sub>B</sub> PLLV has been set at least once
<b>0</b>	[15:13]	r	<b>Reserved</b> Read as 0; should be written with 0.

### 8.1.7.3 Trimmed Current Controlled Clock Control Register

This register controls the trimmed current controlled clock source.

#### PLLOSCCON

#### PLL OSC Control Register



Field	Bits	Type	Description
OSCPD	0	rw	<b>Clock Source Power Saving Mode</b> 0 <sub>B</sub> Trimmed current controlled clock source is active 1 <sub>B</sub> Trimmed current controlled clock source is off
OSCTRIM	[9:1]	rw	<b>Clock Source Trim Configuration</b> This value is used to adjust the frequency range of the current controlled clock source. Do not change this value when writing to this register.
0	[15:10]	r	<b>Reserved</b> Read as 0; should be written with 0.



### 8.1.7.4 PLL Registers

#### PLLSTAT

**PLL Status Register**

**ESFR (F0BC<sub>H</sub>/5E<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>OSC LOC K</b>	<b>OSC STA B</b>	<b>0</b>	<b>REG STA T</b>	<b>VCO L1</b>	<b>VCO L0</b>	<b>FIN DIS</b>	<b>K2 RDY</b>	<b>K1 RDY</b>	<b>N RDY</b>	<b>P RDY</b>	<b>0</b>	<b>VCO LOC K</b>	<b>OSC SEL ST</b>	<b>PWD STA T</b>	<b>VCO BY ST</b>
rh	rh	r	rh	rh	rh	rh	rh	rh	rh	rh	r	rh	rh	rh	rh

Field	Bits	Type	Description
<b>VCOBYST</b>	0	rh	<b>VCO Bypass Status</b> 0 <sub>B</sub> The PLL clock is derived from divider K1 (Prescaler Mode) 1 <sub>B</sub> The PLL clock is derived from divider K2 (Normal / Unlocked Mode) <i>Note: Coding of PLLCON0.VCOBY and VCOBYST are different.</i>
<b>PWDSTAT</b>	1	rh	<b>PLL Power-saving Mode Status</b> 0 <sub>B</sub> The PLL is operable 1 <sub>B</sub> The digital part of the PLL is disabled
<b>OSCSELST</b>	2	rh	<b>Oscillator Input Selection Status</b> 0 <sub>B</sub> External input clock source for the PLL ( $f_{IN}$ ) 1 <sub>B</sub> Internal input clock source for the PLL
<b>VCOLOCK</b>	3	rh	<b>PLL VCO Lock Status</b> 0 <sub>B</sub> The frequency difference of $f_{REF}$ and $f_{DIV}$ is greater than allowed. The PLL cannot lock. 1 <sub>B</sub> The PLL clock $f_{PLL}$ is locked to $f_{REF}$ and is stable. <i>Note: In case of a loss of lock, the VCO frequency <math>f_{VCO}</math> approaches to the upper/lower boundary of the selected VCO band if the reference frequency is higher/lower than possible for locking.</i>
<b>PRDY</b>	5	rh	<b>P-Divider Ready Status</b> 0 <sub>B</sub> Bit field PLLCON1.PDIV has been changed, new K1 divider value not yet used. 1 <sub>B</sub> The P-Divider operates with the value defined in bit field PLLCON1.PDIV.

Field	Bits	Type	Description
<b>NRDY</b>	6	rh	<b>N-Divider Ready Status</b> $0_B$ Bit field PLLCON0.NDIV has been changed, new K1 divider value not yet used. $1_B$ The P-Divider operates with the value defined in bit field PLLCON0.NDIV.
<b>K1RDY</b>	7	rh	<b>K1-Divider Ready Status</b> $0_B$ Bit field PLLCON2.K1DIV has been changed, new K1 divider value not yet used. $1_B$ The K1-Divider operates with the value defined in bit field PLLCON2.K1DIV.
<b>K2RDY</b>	8	rh	<b>K2-Divider Ready Status</b> $0_B$ Bit field PLLCON3.K2DIV has been changed, new K2 divider value not yet used. $1_B$ The K2-Divider operates with the value defined in bit field PLLCON3.K2DIV.
<b>FINDIS</b>	9	rh	<b>Input Clock Disconnect Select Status</b> $0_B$ The VCO is connected to the reference clock $1_B$ The VCO is disconnected from the reference clock  <i>Note: Software can control this bit by writing 1 to bits SETFINDIS or CLRFINDIS in register STATCLR1.</i>
<b>VCOL0</b>	10	rh	<b>VCO Lock Detection Lost Status</b> This sticky bit indicates if bit VCOLOCK has been cleared since VCOL0 has last been cleared (by writing 1 to bit STATCLR1.VCOL0CLR). $0_B$ No falling edge detected $1_B$ PLLV has been cleared at least once (VCO lock was lost)
<b>VCOL1</b>	11	rh	<b>VCO Lock Detection Reached Status</b> This sticky bit indicates if bit VCOLOCK has been set since VCOL1 has last been cleared (by writing 1 to bit STATCLR1.VCOL1CLR). $0_B$ No rising edge detected $1_B$ VCO lock was reached

**System Control Unit (SCU)**

Field	Bits	Type	Description
<b>REGSTAT</b>	12	rh	<b>PLL Power Regulator Status</b> The PLL is powered by a separate internal regulator. $0_B$ The PLL is not powered (off) $1_B$ The PLL is powered (operation possible) <i>Note: Software can control this bit by writing 1 to bits <b>REGENSET</b> or <b>REGENCLR</b> in register <b>PLLCON0</b>.</i>
<b>OSCSTAB</b>	14	rh	<b>OSC_HP Stable</b> $0_B$ The oscillator is starting up. None or less than $2^{15}$ clock cycles have been counted $1_B$ At least $2^{15}$ clock cycles have been counted <i>Note: This bit is cleared when <b>HPOSCCON.MODE</b> is <math>1X_B</math>.</i>
<b>OSCCLOCK</b>	15	rh	<b>OSC_HP Lock</b> $0_B$ The oscillator is unlocked. None or less than $2^{11}$ clock cycles have been counted. $1_B$ The oscillator is locked. At least $2^{11}$ clock cycles have been counted. <i>Note: This bit is cleared when <b>HPOSCCON.MODE</b> is <math>1X_B</math>.</i>
<b>0</b>	4, 13	r	<b>Reserved</b> Read as 0; should be written with 0.

## STATCLR1

## PLL Status Clear 1 Register

**ESFR (F0E2<sub>H</sub>/71<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0										CLR	SET	OSC	OSC	VCO	VCO
										FIN	FIN	2L0	2L1	L1	L0
										DIS	DIS	CLR	CLR	CLR	CLR
r										w	w	w	w	w	w

Field	Bits	Type	Description
<b>VCOL0CLR</b>	0	w	<b>VCOL0 Clear Trigger</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit PLLSTAT.VCOL0 is cleared
<b>VCOL1CLR</b>	1	w	<b>VCOL1 Clear Trigger</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit PLLSTAT.VCOL1 is cleared
<b>OSC2L1CLR</b>	2	w	<b>OSC2L1 Clear Trigger</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bits HPOSCCON.OSC2L1 is cleared
<b>OSC2L0CLR</b>	3	w	<b>OSC2L0 Clear Trigger</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit HPOSCCON.OSC2L0 is cleared
<b>SETFINDIS</b>	4	w	<b>Set Status Bit PLLSTAT.FINDIS</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit PLLSTAT.FINDIS is set. The VCO input clock is disconnected.
<b>CLRFINDIS</b>	5	w	<b>Clear Status Bit PLLSTAT.FINDIS</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit PLLSTAT.FINDIS is cleared. The VCO input clock is connected.
<b>0</b>	[15:6]	r	<b>Reserved</b> Read as 0; should be written with 0.

*Note: Bits of type w are always read as 0.*

These registers control the configuration of the PLL.

### PLLCON0

**PLL Configuration 0 Register**    **ESFR (F1B8<sub>H</sub>/DC<sub>H</sub>)**                      **Reset Value: 1302<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>N ACK</b>	<b>0</b>	<b>NDIV</b>					<b>0</b>	<b>IN SEL</b>	<b>REG EN SET</b>	<b>REG EN CLR</b>	<b>VCOSSEL</b>		<b>VCO PWD</b>	<b>VCO BY</b>	
rw	r	rw					r	rw	w	w	rw		rw	rw	

Field	Bits	Type	Description
<b>VCOBY</b>	0	rw	<b>VCO Bypass</b> 0 <sub>B</sub> Select divider K2 for PLL clock (Normal / Unlocked Mode) 1 <sub>B</sub> Select divider K1 for PLL clock (Prescaler Mode, i.e. VCO is bypassed) Bit PLLSTAT.VCOBYST shows the actually selected divider. <i>Note: Coding of VCOBY and PLLSTAT.VCOBYST are different.</i>
<b>VCOPWD</b>	1	rw	<b>VCO Power Saving Mode</b> 0 <sub>B</sub> Normal behavior 1 <sub>B</sub> The VCO is put into a power saving mode and can no longer be used. Only the Prescaler Mode is active if previously selected.
<b>VCOSSEL</b>	[3:2]	rw	<b>VCO Range Select</b> The values for the different settings are listed in the data sheet.
<b>REGENCLR</b>	4	w	<b>Power Regulator Enable Clear</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Switch off the PLL's power regulator. The PLL is not powered (no operation possible). <i>Note: This bit is always read as 0.</i>
<b>REGENSET</b>	5	w	<b>Power Regulator Enable Set</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Switch on the PLL's power regulator. The PLL is powered (operation possible). <i>Note: This bit is always read as 0.</i>

**System Control Unit (SCU)**

Field	Bits	Type	Description
<b>INSEL</b>	6	rw	<b>Input Select</b> $0_B$ $f_{OSC}$ is selected as input for the PLL $1_B$ $f_{CLKIN1}$ is selected as input for the PLL
<b>NDIV</b>	[13:8]	rw	<b>N-Divider Value</b> The value the N-Divider operates is NDIV+1. Only values between N = 8 and N = 28 are allowed for VCOSEL = $00_B$ . Only values between N = 16 and N = 40 are allowed for VCOSEL = $01_B$ . Outside of this range, stable operation cannot be ensured.
<b>NACK</b>	15	rw	<b>N-Divider Ready Acknowledge</b> Setting this bit provides the acknowledge signal to NRDY.
<b>0</b>	7, 14	r	<b>Reserved</b> Read as 0; should be written with 0.

**PLLCON1**

**PLL Configuration 1 Register ESFR (F1BA<sub>H</sub>/DD<sub>H</sub>)** **Reset Value: 000A<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>P ACK</b>		<b>0</b>						<b>0</b>	<b>EM FIN DIS EN</b>	<b>EM CLK EN</b>	<b>0</b>	<b>A OSC SEL</b>	<b>RES LD</b>	<b>OSC SEL</b>	<b>PLL PWD</b>
rw		r				rw		r	rw	rw	r	rw	w	rw	rw

Field	Bits	Type	Description
<b>PLLPWD</b>	0	rw	<b>PLL Power Saving Mode</b> 0 <sub>B</sub> Normal behavior 1 <sub>B</sub> Complete PLL block is put into a power saving mode and no longer operates
<b>OSCSEL</b>	1	rw	<b>Oscillator Input Selection</b> 0 <sub>B</sub> Select external clock as input for PLL 1 <sub>B</sub> Select trimmed current controlled clock as input for PLL
<b>RESLD</b>	2	w	<b>Restart VCO Lock Detection</b> Setting this bit will reset bit PLLSTAT.VCOLOCK and restart the VCO lock detection. <i>Note: This bit is always read as 0.</i>
<b>AOSCSEL</b>	3	rw	<b>Asynchronous Oscillator Input Selection</b> This bit overrides the setting of bit OSCSEL. 0 <sub>B</sub> Configuration is controlled via bit OSCSEL 1 <sub>B</sub> Select asynchronously trimmed current controlled clock as input for PLL
<b>EMCLKEN</b>	5	rw	<b>VCOLCK Emergency System Clock Source Select Enable</b> This bit requests the master clock multiplexer (MCM) to switch to an alternate clock (selected by bit field SYSCON0.EMCLKSEL) in a VCOLCK emergency case. 0 <sub>B</sub> MCM remains controlled by SYSCON0.CLKSEL 1 <sub>B</sub> MCM is controlled by SYSCON0.EMCLKSEL

Field	Bits	Type	Description
<b>EMFINDISEN</b>	6	rw	<b>Emergency Input Clock Disconnect Enable</b> This bit defines if bit PLLSTAT.FINDIS is set in a VCOLCK emergency case. 0 <sub>B</sub> No action 1 <sub>B</sub> PLLSTAT.FINDIS is set in a VCOLCK emergency case  <i>Note: Please refer to the Programmer's Guide for a description of the proper handling.</i>
<b>PDIV</b>	[11:8]	rw	<b>P-Divider Value</b> The value the P-Divider operates is PDIV+1.
<b>PACK</b>	15	rw	<b>P-Divider Ready Acknowledge</b> Setting this bit provides the acknowledge to PRDY.
<b>0</b>	4, 7, [14:12]	r	<b>Reserved</b> Read as 0; should be written with 0.



**PLLCON2**

**PLL Configuration 2 Register**    **ESFR (F1BC<sub>H</sub>/DE<sub>H</sub>)**    **Reset Value: 0001<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>K1 ACK</b>			<b>0</b>												
rw			r												
									<b>K1DIV</b>						
										rw					

Field	Bits	Type	Description
<b>K1DIV</b>	[9:0]	rw	<b>K1-Divider Value</b> The value the K1-Divider operates is K1DIV+1.
<b>K1ACK</b>	15	rw	<b>K1-Divider Ready Acknowledge<sup>1)</sup></b> Setting this bit provides the acknowledge to K1RDY.
<b>0</b>	[14:10]	r	<b>Reserved</b> Read as 0; should be written with 0.

<sup>1)</sup> Please refer to the Programmer's Guide for a description of the proper handling.

**PLLCON3**

**PLL Configuration 3 Register**    **ESFR (F1BE<sub>H</sub>/DF<sub>H</sub>)**    **Reset Value: 00CB<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>K2 ACK</b>			<b>0</b>												
rw			r												
										<b>K2DIV</b>					
										rw					

Field	Bits	Type	Description
<b>K2DIV</b>	[9:0]	rw	<b>K2-Divider Value</b> The value the K2-Divider operates is K2DIV+1.
<b>K2ACK</b>	15	rw	<b>K2-Divider Ready Acknowledge<sup>1)</sup></b> Setting this bit provides the acknowledge to K2RDY.
<b>0</b>	[14:10]	r	<b>Reserved</b> Read as 0; should be written with 0.

<sup>1)</sup> Please refer to the Programmer's Guide for a description of the proper handling.

### 8.1.7.5 System Clock Control Registers

These registers control the system level clock behavior.

#### **SYSCON0**

#### **System Control 0 Register**

**SFR (FF4A<sub>H</sub>/A5<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SEL STA T	0	EMS VCO	EMS OSC	STM CLKSEL		WUT CLKSEL		EM CLK SEL EN	0	EM CLKSEL		0	CLKSEL		
rh	r	rh	rh	rw		rw		rw	r	rw		r	rw		

Field	Bits	Type	Description
<b>CLKSEL</b>	[1:0]	rw	<b>Clock Select</b> This bit field defines the clock source that is used as system clock for normal operation. 00 <sub>B</sub> The Wake-up clock $f_{WU}$ is used 01 <sub>B</sub> The oscillator clock (OSC_HP) $f_{OSC}$ is used 10 <sub>B</sub> The PLL clock $f_{PLL}$ is used 11 <sub>B</sub> CLKIN1 as direct input clock $f_{CLKIN1}$ is used
<b>EMCLKSEL</b>	[4:3]	rw	<b>Emergency Clock Select</b> This bit field defines the clock source that is used as system clock in case of an OSCWDT or VCOLCK emergency event. 00 <sub>B</sub> The Wake-up clock $f_{WU}$ is used 01 <sub>B</sub> The oscillator clock (OSC_HP) $f_{OSC}$ is used 10 <sub>B</sub> The PLL clock $f_{PLL}$ is used 11 <sub>B</sub> CLKIN1 as direct input clock $f_{CLKIN1}$ is used
<b>EMCLKSELEN</b>	6	rw	<b>Emergency Clock Select Enable</b> Controls switching the system clock to an alternate source in case of an OSCWDT or VCOLCK event. 0 <sub>B</sub> The switching is disabled 1 <sub>B</sub> The switching is enabled

Field	Bits	Type	Description
<b>WUTCLKSEL</b>	[8:7]	rw	<b>WUT Clock Select</b> This bit field defines the clock source that is used as wake-up timer clock for operation. 00 <sub>B</sub> The Wake-up clock $f_{WU}$ is used 01 <sub>B</sub> The oscillator clock (OSC_HP) $f_{OSC}$ is used 10 <sub>B</sub> The PLL clock $f_{PLL}$ is used 11 <sub>B</sub> CLKIN1 as direct Input clock $f_{CLKIN1}$ is used
<b>STMCLKSEL</b>	[11:9]	rw	<b>STM Clock Select</b> This bit field defines the clock source that is used as STM clock for operation. 000 <sub>B</sub> CLKIN2 as direct input clock $f_{CLKIN2}$ is used 001 <sub>B</sub> The oscillator clock (OSC_HP) $f_{OSC}$ is used 010 <sub>B</sub> The PLL clock $f_{PLL}$ is used 011 <sub>B</sub> CLKIN1 as direct Input clock $f_{CLKIN1}$ is used 100 <sub>B</sub> The Wake-up clock $f_{WU}$ is used 101 <sub>B</sub> Reserved, do not use this combination 110 <sub>B</sub> Reserved, do not use this combination 111 <sub>B</sub> Reserved, do not use this combination
<b>EMSOSC</b>	12	rh	<b>OSCWDT Emergency Event Source Status</b> 0 <sub>B</sub> No OSCWDT emergency event occurred since EMSOSC has been cleared last 1 <sub>B</sub> An OSCWDT emergency event has occurred <i>Note: This bit is only set if EMCLKSELEN is set.</i>
<b>EMSVCO</b>	13	rh	<b>VCOLCK Emergency Event Source Status</b> 0 <sub>B</sub> No VCOLCK emergency event occurred since EMSVCO has been cleared last 1 <sub>B</sub> A VCOLCK emergency event has occurred <i>Note: This bit is only set if EMCLKSELEN is set.</i>
<b>SELSTAT</b>	15	rh	<b>Clock Select Status</b> 0 <sub>B</sub> The standard configuration from bit field CLKSEL is used currently 1 <sub>B</sub> The configuration from bit field EMCLKSEL is used currently
<b>0</b>	2, 5, 14	r	<b>Reserved</b> Read as 0; should be written with 0.

**STATCLR0**

**Status Clear 0 Register**

**ESFR (F0E0<sub>H</sub>/70<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

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

Field	Bits	Type	Description
<b>EMCOSC</b>	12	w	<b>EMSOSC Clear Trigger</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit SYSCON0.EMSOSC is cleared <i>Note: This bit is always read as 0.</i>
<b>EMCVCO</b>	13	w	<b>EMSVCO Clear Trigger</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit SYSCON0.EMSVCO is cleared <i>Note: This bit is always read as 0.</i>
<b>0</b>	[11:0], [15:14]	r	<b>Reserved</b> Read as 0; should be written with 0.

### 8.1.7.6 RTC Clock Control Register

*Note: Only change register RTCCLKCON while the RTC is off.*

#### RTCCLKCON

**RTC Clock Control Register**

**SFR (FF4E<sub>H</sub>/A7<sub>H</sub>)**

**Reset Value: 0006<sub>H</sub>**

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

Field	Bits	Type	Description
<b>RTCCLKSEL</b>	[1:0]	rw	<b>RTC Clock Select</b> This bit field defines the count clock source for the RTC. 00 <sub>B</sub> The PLL clock $f_{PLL}$ is used 01 <sub>B</sub> The oscillator clock (OSC_HP) $f_{OSC}$ is used 10 <sub>B</sub> The Wake-up clock signal $f_{WU}$ is used 11 <sub>B</sub> CLKIN2 as direct input clock $f_{CLKIN2}$ is used
<b>RTCCM</b>	2	rw	<b>RTC Clocking Mode</b> 0 <sub>B</sub> Asynchronous Mode: The RTC internally operates with $f_{RTC}$ . No register access is possible. 1 <sub>B</sub> Synchronous Mode: The RTC internally operates with $f_{SYS}$ clock. Registers can be read and written.
<b>0</b>	[15:3]	r	<b>Reserved</b> Read as 0; should be written with 0.

### 8.1.7.7 External Clock Control Register

This register control the setting of external clock for pin 2.8 and 7.1.

#### EXTCON

**External Clock Control Register SFR (FF5E<sub>H</sub>/AF<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FO EN	FO SS	FORV					0	FO TL	0	SEL				EN	
rw	rw	rw					r	rh	r	rw				rw	

Field	Bits	Type	Description
EN	0	rw	<b>External Clock Enable</b> 0 <sub>B</sub> No external clock signal is provided. The signal is tied to zero. 1 <sub>B</sub> The configured external clock signal is provided as alternate output signal
SEL	[4:1]	rw	<b>External Clock Select</b> Selects the clock signal to be routed to the EXTCLK pin: 0000 <sub>B</sub> System clock $f_{SYS}$ 0001 <sub>B</sub> Programmable clock signal $f_{OUT}$ 0010 <sub>B</sub> PLL output clock $f_{PLL}$ 0011 <sub>B</sub> Oscillator clock $f_{OSC}$ 0100 <sub>B</sub> Wake-up clock $f_{WU}$ 0101 <sub>B</sub> Direct Input clock $f_{CLKIN1}$ 1000 <sub>B</sub> RTC count clock $f_{RTC}$ All other combination are reserved, do not use.
FOTL	6	rh	<b>Frequency Output Toggle Latch</b> Toggled upon each underflow of FOCNT.
FORV	[13:8]	rw	<b>Frequency Output Reload Value</b> Copied to FOCNT upon each underflow of FOCNT.
FOSS	14	rw	<b>Frequency Output Signal Select</b> 0 <sub>B</sub> Output of the toggle latch 1 <sub>B</sub> Output of the reload counter: duty cycle depends on FORV

**System Control Unit (SCU)**

Field	Bits	Type	Description
<b>FOEN</b>	15	rw	<b>Frequency Output Enable</b> $0_B$ Frequency output generation stops when $f_{OUT}$ is/becomes low. $1_B$ FOCNT is running, $f_{OUT}$ is gated to pin. First reload after 0 - 1 transition.
<b>0</b>	5, 7	r	<b>Reserved</b> Read as 0; should be written with 0.

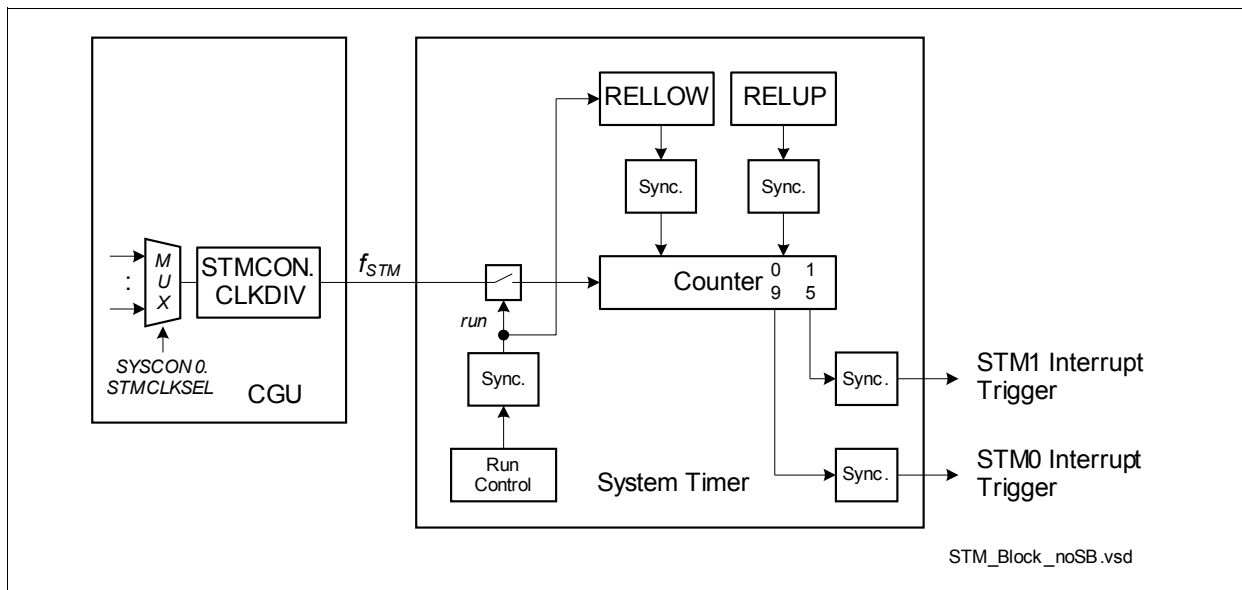


## 8.2 System Timer Function (STM)

The System Timer equips the device with a real time counter function

The STM function can operate on the clock sources described in [System Timer \(STM\) Clock Generation](#).

The STM consists of a 16-bit counter that is able to generate up to two interrupts. Driven by a clock source the counter can be used to count time based events and upon an interrupt trigger based on a time generated out of a clock different than the remaining of the system. A clock function can easily be implemented based on these interrupts in software.



**Figure 8-16 STM Block Diagram**

## 8.2.1 STM Registers

### 8.2.1.1 Register STMREL

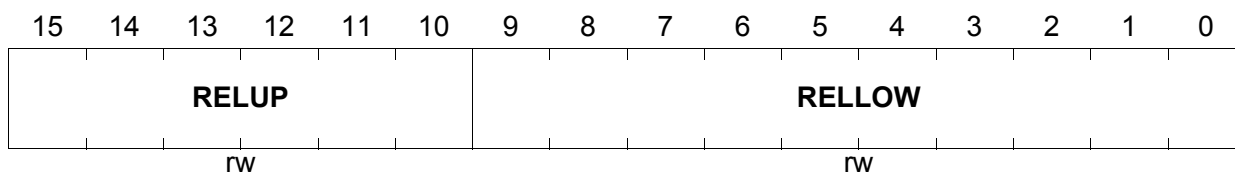
Via this register, the reload value and therefore the period of the STM is defined.

#### STMREL

**STM Reload Register**

**ESFR (F1A8<sub>H</sub>/D4<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
<b>RELOW</b>	[9:0]	rw	<b>Reload Lower Value</b> The counter counts up and issues an interrupt trigger when bit 9 changes from 1 <sub>B</sub> to 0 <sub>B</sub> . Upon this trigger the counter is loaded with the reload value defined by this bit field.
<b>RELUP</b>	[15:10]	rw	<b>Reload Upper Value</b> The counter counts up and issues an interrupt trigger when bit 15 changes from 1 <sub>B</sub> to 0 <sub>B</sub> . Upon this trigger the counter is loaded with the reload value defined by this bit field and by bit field RELOW.

### 8.2.1.2 Register STMCON

This register holds the status and control bits for the STM.

#### STMCON

**STM Control Register**

**ESFR (F1AA<sub>H</sub>/D5<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

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

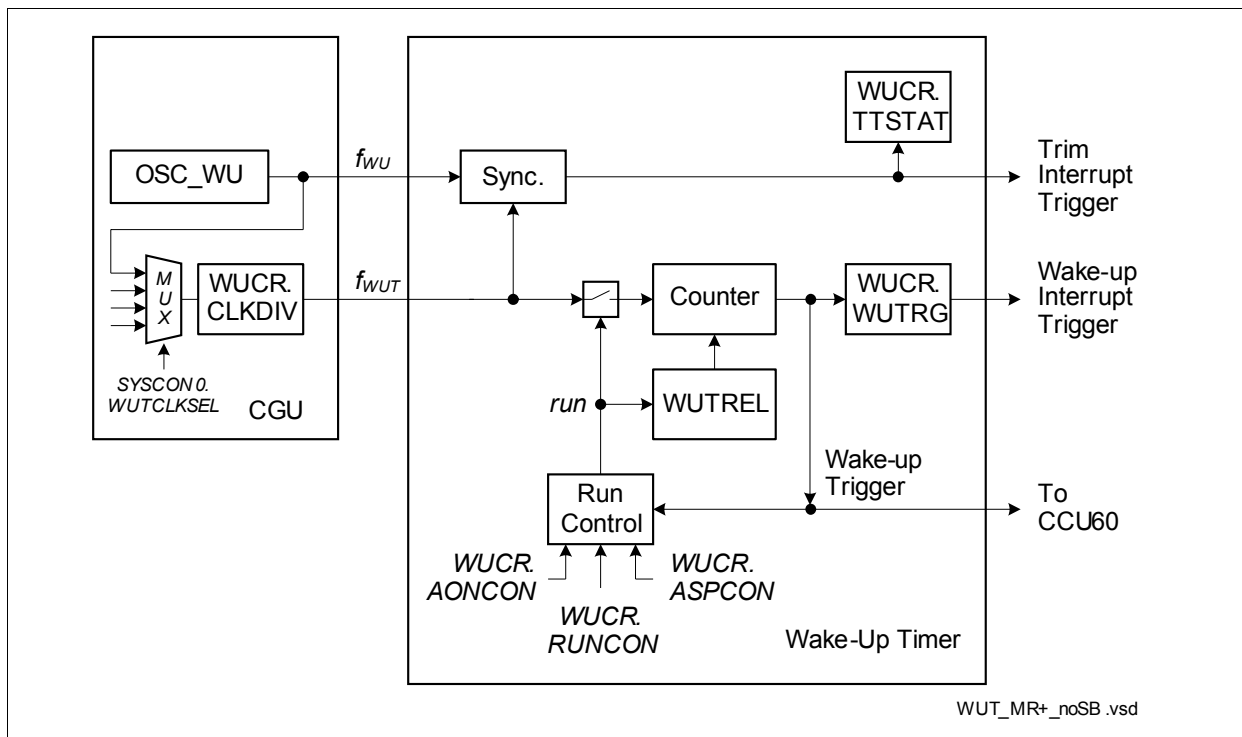
Field	Bits	Type	Description
<b>RUN</b>	0	rw	<b>Run Control</b> 0 <sub>B</sub> STM is stopped 1 <sub>B</sub> STM is operating By setting this bit the STM is started and the reload value STMREL.REL is loaded into the counter.
<b>CLKDIV</b>	[4:1]	rw	<b>Clock Divider for the STM Clock</b> This bit field defines the divider factor of the STM clock input. The selected input clock is divided by 2 <sup>&lt;CLKDIV&gt;</sup> .
<b>0</b>	[15:5]	r	<b>Reserved</b> Read as 0; should be written with 0.

### 8.3 Wake-up Timer (WUT)

The Wake-up Timer provides an additional resource to trigger system functions after a specific period of time.

The Wake-up Timer function can operate on the clock sources described in [Wake-up Timer \(WUT\) Clock Generation](#).

The wake-up timer clock  $f_{WUT}$  drives a simple counter. All functions are controlled by register WUCR.



**Figure 8-17 Wake-up Timer Logic**

#### 8.3.1 Wake-up Timer Operation

The Wake-up Timer start and stop is controlled by the Run Control logic. The timer can be started in the following way:

- bit WUCR.RUN is set

When the timer is started the prescaler is reset and the counter starts to count down.

The wake-up interval counter is clocked with  $f_{WUT}$  and counts down until it reaches zero. It then generates a wake-up trigger and sets bit WUCR.WUTRG.

The timer is stopped in the following ways:

- bit WUCR.RUN is cleared
- bit WUCR.ASP is set AND a wake-up trigger is generated

If the counter is not stopped by its zero trigger it continues counting down from WUTREL.

### **Determination of Wake-up Period**

The actual frequency of the trimmed current controlled wake-up clock (OSC\_WU) can be measured prior to entering power-save mode in order to adjust the number of clock cycles to be counted (value written to the counter) and so to define the time until wake-up. The period of the the OSC\_WU can be measured by evaluating the (synchronized) trigger that can generate interrupt requests or can be monitored with bit WUCR.TTSTAT.

As using an interrupt together with software contain some uncertainty there is a second way to determine the wake-up period. The wake-up triggers generated by the WUT are forwarded to the CCU60 and can there be evaluated compared to the accurate system clock.

## 8.3.2 WUT Registers

### 8.3.2.1 Register WUTREL

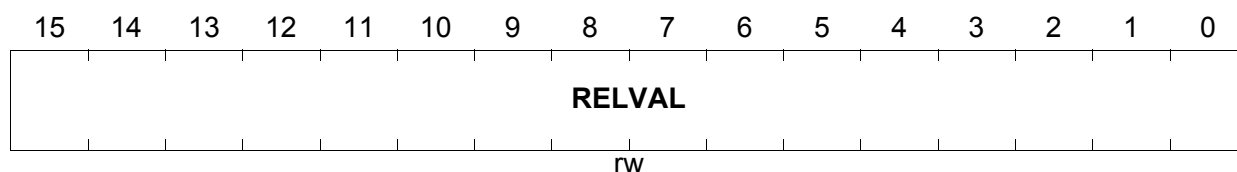
This register configures the reload value of the counter.

#### **WUTREL**

#### **Wake-up Timer Reload Register**

**ESFR (F0B0<sub>H</sub>/58<sub>H</sub>)**

**Reset Value: FFFF<sub>H</sub>**



Field	Bits	Type	Description
RELVAL	[15:0]	rw	<b>Wake-up Timer Reload Value</b> The WUT counter is reloaded with this value and starts to count down when the timer is started.

### 8.3.2.2 Register WUCR

This register the status and control bits for the WUT.

#### WUCR

**Wake-up Control Register**

**ESFR (F1B0<sub>H</sub>/D8<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>WU TRG</b>	<b>TTS TAT</b>	<b>CLKDIV</b>			<b>ASP</b>	<b>AON</b>	<b>RUN</b>	<b>CLR TRG</b>	<b>0</b>	<b>ASP CON</b>		<b>AON CON</b>		<b>RUN CON</b>	
rh	rh	rw			rh	rh	rh	w	r	w		w		w	

Field	Bits	Type	Description
<b>RUNCON</b>	[1:0]	w	<b>Control Field for RUN</b> 00 <sub>B</sub> No action 01 <sub>B</sub> Set bit RUN 10 <sub>B</sub> Clear bit RUN 11 <sub>B</sub> Reserved, do not use this combination <i>Note: This bit is always read as 0.</i>
<b>AONCON</b>	[3:2]	w	<b>Control Field for AON</b> 00 <sub>B</sub> No action 01 <sub>B</sub> Reserved, do not use this combination 10 <sub>B</sub> Clear bit AON 11 <sub>B</sub> Reserved, do not use this combination <i>Note: This bit is always read as 0.</i>
<b>ASPCON</b>	[5:4]	w	<b>Control Field for ASP</b> 00 <sub>B</sub> No action 01 <sub>B</sub> Set bit ASP 10 <sub>B</sub> Clear bit ASP 11 <sub>B</sub> Reserved, do not use this combination <i>Note: This bit is always read as 0.</i>
<b>CLRTRG</b>	7	w	<b>Clear Bit WUTRG</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Clear bit WUTRG <i>Note: This bit is always read as 0.</i>

**System Control Unit (SCU)**

Field	Bits	Type	Description
<b>RUN</b>	8	rh	<b>Run Indicator</b> $0_B$ Wake-up counter is stopped $1_B$ Wake-up counter is counting down <i>Note: Clearing this bit via a write action to bit field RUNCON stops the WUT after four cycles of <math>f_{WUT}</math>.</i>
<b>AON</b>	9	rh	<b>Auto-Start Indicator</b> $0_B$ Wake-up counter is started by software only $1_B$ Reserved, do not use this combination <i>Note: This bit is cleared by writing <math>01_B</math> to bit field AONCON.</i>
<b>ASP</b>	10	rh	<b>Auto-Stop Indicator</b> $0_B$ Wake-up counter runs continuously $1_B$ Wake-up counter stops after generating a trigger when reaching zero
<b>CLKDIV</b>	[13:11]	rw	<b>Clock Divider for the WUT Clock</b> This bit field defines the divider factor of the WUT clock input. The selected input clock is divided by $2^{<CLKDIV>}$ .
<b>TTSTAT</b>	14	rh	<b>Trim Trigger Status</b> $0_B$ No trim trigger event is active. No trim interrupt trigger is generated. $1_B$ A trim trigger event is active. A trim interrupt trigger is generated. <i>Note: This bit is not valid if <math>f_{WUT} = f_{WU}</math> is configured by SYSCON0.WUTCLKSEL</i>
<b>WUTRG</b>	15	rh	<b>WUT Trigger Indicator</b> $0_B$ No trigger event has occurred since WUTRG has been cleared last. No interrupt trigger is generated. $1_B$ A wake-up trigger event has occurred. A wake-up interrupt trigger is generated.
<b>0</b>	6	r	<b>Reserved</b> Read as 0; should be written with 0.

*Note: The bits in the upper byte of register WUCR indicate the current status of the wake-up counter logic. They are not influenced by a write access, but are controlled by their associated control fields (lower byte) or by hardware.*



**System Control Unit (SCU)**

*The control bit(field)s in the lower byte of register WUCR determine the state of the status bits (upper byte) of the wake-up counter logic. Setting bits by software triggers the associated action, writing 0 has no effect.*

## **8.4 Reset Operation**

All resets are generated by the Reset Control Block. It handles the control of the reset triggers as well as the length of a reset and the reset timing. A reset leads the system, or a part of the system depending on the reset, to a initialization into a defined state.

### **8.4.1 Reset Architecture**

The XE16xyM contains a very sophisticated reset architecture to offer the greatest amount of flexibility for the support of different applications. The reset architecture supports the different power domains.

Different reset types for the complete system are supported.

#### **8.4.1.1 Device Reset Hierarchy**

The device reset hierarchy is divided according to the power domains (see [Chapter 8.6](#)) into following linked levels:

Level 1: I/O domain (power domain DMP\_B)

Level 2: System power domain DMP\_M

Level 3: System power domain DMP\_1

If a power domain (level) is deactivated all resets of the deactivated level and all resets of all lower power domains are asserted.

#### **8.4.1.2 Reset Types**

The following summary shows the different reset types.

##### **Power Reset**

- **Power-on Reset**  
This reset leads to a defined state of the complete system. This reset should only be requested on a real power-on event and not by any non power related event.
- **Power Reset for DMP\_M and DMP\_1 power domains**  
This reset regains data consistency upon a power fail in the DMP\_M or DMP\_1 power domains.

##### **Functional / User Reset**

- **Debug Reset**  
This reset leads to a defined state of the complete debug system.
- **Internal Application Reset**  
This reset leads to a defined state of the complete application system with the following parts: all peripherals (except the RTC), the CPU and partially the SCU and the flash memory.

- **Application Reset**

This reset leads to a defined state of the complete application system with the following parts: all peripherals (except the Ports and the RTC), the CPU and partially the SCU and the flash memory.

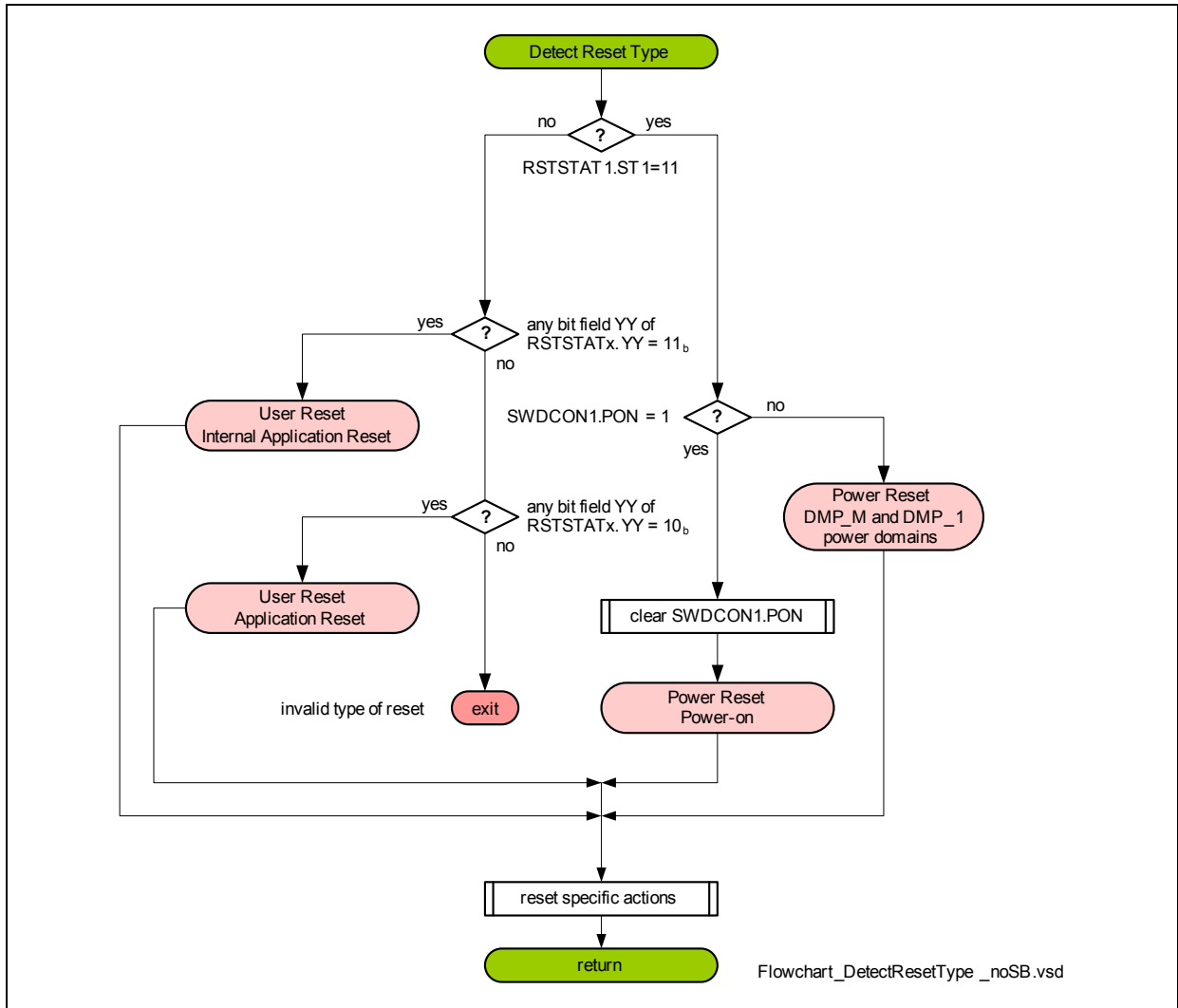
After a reset has been executed, the Reset Status registers RSTSTATx indicate the latest reset that has occurred.

To identify the type and the trigger of the latest reset registers , , and **SWDCON1** may be evaluated according to **Table 8-1**. The latest reset that has occurred is always the reset of the highest type. If two reset triggers of the same type are indicated, this means that the two triggers have been active at the same time. If two or more reset triggers of a different type are reported, always the reset of the highest type is the latest one.

**Table 8-1 Identification of a Reset**

<b>Type of Reset</b> (in hierarchical order, highest on top)	<b>Identification</b>
Power-on Reset	SWDCON1.PON = 1 <sub>B</sub> RSTSTAT1.STM = 11 <sub>B</sub> RSTSTAT1.ST1 = 11 <sub>B</sub> Further action: clear PON bit to be able to identify a Power Reset for DMP_M and DMP_1 power domains.
Power Reset for DMP_M and DMP_1 power domains	SWDCON1.PON = 0 <sub>B</sub> (unchanged after clearing) RSTSTAT1.STM = 11 <sub>B</sub> RSTSTAT1.ST1 = 11 <sub>B</sub>
Internal Application Reset	RSTSTAT1.ST1 = 00 <sub>B</sub> any bit field y in RSTSTATx.y = 10 <sub>B</sub>
Application Reset	RSTSTAT1.ST1 = 00 <sub>B</sub> any bit field y in RSTSTATx.y = 11 <sub>B</sub> (except RSTSAT1.ST1 and RSTSTAT1.STM)

The algorithm depicted in **Figure 8-18** shows a sequence to detect the type of the reset comprising the conditions in **Table 8-1**.



**Figure 8-18 Algorithm for the Detection of the Type of a Reset**

## **8.4.2 General Reset Operation**

A reset is generated if an enabled reset request trigger is asserted. Most reset request triggers can be configured for the reset type it should initiate. No action (disabled) is one possible configuration and can be selected for a reset request trigger by setting the respective bit field in a Reset Configuration Register to 00<sub>B</sub>. The debug reset can only be requested by dedicated reset request triggers and can not be selected via a Reset Configuration Register. For more information see also registers **RSTCON0** and **RSTCON1**.

The duration of a reset is defined by two independent counters. One counter for the System and Application Reset types and one separate counter for the debug reset. A separate counter for the debug reset was implemented to allow a non-intrusive adaptation of the reset length to the debugger needs without modification of the application setting.

### **8.4.2.1 Reset Counters (RSTCNTA and RSTCNTD)**

RSTCNTA is the reset counter that controls the reset length for all application relevant resets (Internal Application Reset, and Application Reset). RSTCNTD is the reset counter that controls the reset length for the debug reset.

The reset counters control the length of the internal resets. This can be used to configure the duration of a reset output via the ESRx pins, so this matches with the reset input requirements of external blocks connected to these signals.

A reset counter RSTCNT is an 8-bit counter counting down from the reload value defined by **RSTCNTCON.RELx** (x = A or D). The counter is started by the reset control block as soon as a reset request trigger condition becomes active (for more information see **Table 8-2** and **Table 8-3**). Whether the counter has to be started or not depends on the reset request trigger and whether the counter is already active or not. In case of that the counter is inactive, not counting down, it is always started. While the counter is already active it depends on the reset type of the new reset request trigger that was asserted anew if the counter is restarted or not. This behavior is summarized in **Table 8-2** and **Table 8-3**.

**Table 8-2 Restart of RSTCNTA**

Reset Active	New Reset Trigger			
	Power-On	Debug Reset	Internal Application Reset	Application Reset
Internal Application Reset	Restart with default delay	No Change	No Change	No Change
Application Reset	Restart with default delay	No Change	Restart with defined delay	No Change

**Table 8-3 Restart of RSTCNTD**

Reset Active	New Reset Trigger			
	Power-On	Debug Reset	Internal Application Reset	Application Reset
Debug Reset	Restart with default delay	No Change	No Change	No Change

The reset counters RSTCNTx ensure a configurable minimum duration of a generated reset. If a reset request trigger remains asserted after the respective counter has counted down, the counter is not started again, instead the reset control block keeps the reset asserted until the reset request trigger is deasserted.

#### **8.4.2.2 De-assertion of a Reset**

The reset of a dedicated type is de-asserted when all of the following conditions are fulfilled:

- The reset counter has been expired (reached zero).
- No reset request trigger that is configured to generate a reset of the dedicated type (or higher) is currently asserted.

#### **Example1**

Reset request trigger A is asserted and leads to an Application Reset. If the reset request trigger is de-asserted before RSTCNTA reached zero the Application Reset is de-asserted when RSTCNTA reaches zero. If the reset request trigger is de-asserted after RSTCNTA reached zero the Application Reset is de-asserted when the reset request trigger is de-asserted.

### **Example2**

Reset request trigger A is asserted and leads to an Application Reset. Reset request trigger A is de-asserted before RSTCNTA reached zero. Reset request trigger B is asserted after reset request trigger A but before RSTCNTA reaches zero. Reset request trigger B is also configured to result in a Application Reset. If the reset request trigger B is de-asserted before RSTCNTA reached zero the Application Reset is de-asserted when RSTCNTA reaches zero. If the reset request trigger B is de-asserted after RSTCNTA reached zero the Application Reset is de-asserted when the reset request trigger B is de-asserted.

### **8.4.3 Debug Reset Assertion**

Unlike the other reset types a Debug Reset can only be asserted if the following two conditions are valid:

- A reset request trigger is asserted that request a debug reset
- An Application Reset is already active in the system

### **8.4.4 Coupling of Reset Types**

The different reset types are coupled for a better usage:

- The assertion of a Power-on Reset automatically asserts also the following reset types:
  - Debug Reset
  - Internal Application Reset
  - Application Reset
- The assertion of an Internal Application Reset automatically asserts also the following reset type:
  - Application Reset

### 8.4.5 Reset Request Trigger Sources

The following overview summarizes the different reset request trigger sources within the system.

#### **Power-On Reset Pin $\overline{\text{PORST}}$**

A Power-on Reset is requests asynchronously, by driving the  $\overline{\text{PORST}}$  pin low.

#### **Supply Watchdog (SWD)**

If the power supply for I/O domain is below the value required for proper functionality, a non-synchronized reset request trigger is generated if the SWD reset generation is enabled. This ensures a reproducible behavior in the case of power-fail. This can also be used to restart the system without the usage of the  $\overline{\text{PORST}}$  pin. As long as the I/O power domain does not get the required voltage level the system is held in the reset.

#### **Core Power Validation (PVC\_M and PVC\_1)**

If the core power supply is below the value required for proper functionality of the main power domain (PVC\_M), a reset request trigger can be forwarded to the system. The generation of a Power-on Reset is configured by bit  $\text{PVC\_MCON0.L1RSTEN} = 1_B$ . If the bit  $\text{PVC\_MCON0.L1RSTEN} = 1_B$  a request trigger is asserted for PVC\_M1 upon a level check match. If the bit  $\text{PVC\_MCON0.L2RSTEN} = 1_B$  a request trigger is asserted for PVC\_M2 upon a level check match.

If the core power supply is below the value required for proper functionality of the application power domain (PVC\_1), a reset request trigger can be forwarded to the system. The generation of a Power-on Reset (Application Power Domain only) is configured by bit  $\text{PVC1CON0.L1RSTEN} = 1_B$ . If bit  $\text{PVC1CON0.L1RSTEN} = 1_B$  a request trigger is asserted for PVC\_11 upon a level check match. If the bit  $\text{PVC1CON0.L2RSTEN} = 1_B$  a request trigger is asserted for PVC\_12 upon a level check match.

For more information about the Power Validation Circuit see [Chapter 8.6.2](#).

#### **$\overline{\text{ESRx}}$**

An  $\overline{\text{ESRx}}$  reset request trigger leads to a configurable reset. The type of reset can be configured via [RSTCON1.ESRx](#).

The pins  $\overline{\text{ESRx}}$  can serve as an external reset input as well as a reset output (open drain) for Internal Application and Application Resets. Furthermore, several GPIO pad triggers, that can be enabled additionally via register  $\text{ESREXCONx}$  ( $x = 1, 2$ ), interfere with the ESR pin function. GPIO and  $\overline{\text{ESRx}}$  pin triggers can be enabled/disabled individually and are combined for the reset trigger generation.



## **System Control Unit (SCU)**

If pin  $\overline{\text{ESRx}}$  is enabled as reset output and the input level is low while the output stage is disabled (indicating that it is still driven low externally), the reset circuitry holds the chip in reset until a high level is detected on  $\overline{\text{ESRx}}$ . Minimum value for RSTCNTCON.RELA must be the reset value.

*Note: The reset output is only driven low for the duration the reset counter RSTCNTA is active. During a possible reset extension the reset output is no longer driven.*

### **Software**

A software reset request trigger leads to a configurable reset. The type of reset can be configured via **RSTCON0**.SW.

### **Watchdog Timer**

A WDT reset request trigger leads to a configurable reset. The type of reset can be configured via **RSTCON1**.WDT. A WDT reset is requested on a WDT overflow event. For more information see **Chapter 8.12**.

### **CPU**

A CPU reset request trigger leads to a configurable reset. The type of reset can be configured via **RSTCON0**.CPU. A CPU reset is requested when instruction SRST is executed.

### **Memory Parity**

A MP reset request trigger leads to a configurable reset. The type of reset can be configured via **RSTCON1**.MP. For more information see **Chapter 8.14.2**.

### **OCDS Block**

The OCDS block has several options to request different reset types:

1. A Debug Reset either via the OCDS reset function or via bit CBS\_OJCONF.RSTCL1 AND CBS\_OJCONF.RSTCL3
2. An Internal Application Reset via bit CBS\_OJCONF.RSTCL2
3. An Application Reset via bit CBS\_OJCONF.RSTCL3

#### **8.4.5.1 Reset Sources Overview**

The connection of the reset sources and the activated reset types are shown in **Table 8-4**.

**Table 8-4 Effects of Reset Types for Reset Activation**

<b>Reset Request Trigger</b>	<b>Application Reset</b>	<b>Internal Application Reset</b>	<b>Debug Reset</b>
<b>PORST</b>	Activated	Activated	Activated
<b>SWD</b>	Activated	Activated	Activated
<b>PVC_M1</b>	Activated	Activated	Activated
<b>PVC_M2</b>	Activated	Activated	Activated
<b>PVC_11</b>	Activated	Activated	Activated
<b>PVC_12</b>	Activated	Activated	Activated
<b>ESR0</b>	Configurable	Configurable	Not Activated
<b>ESR1</b>	Configurable	Configurable	Not Activated
<b>ESR2</b>	Configurable	Configurable	Not Activated
<b>WDT</b>	Configurable	Configurable	Not Activated
<b>SW</b>	Configurable	Configurable	Not Activated
<b>CPU</b>	Configurable	Configurable	Not Activated
<b>MP</b>	Configurable	Configurable	Not Activated
<b>OCDS Reset</b>	Not Activated	Not Activated	Activated <sup>1)</sup>
<b>CBS_OJCONF.RSTCL1</b>	Not Activated	Not Activated	Activated <sup>1)</sup>
<b>CBS_OJCONF.RSTCL2</b>	Activated	Activated	Not Activated
<b>CBS_OJCONF.RSTCL3</b>	Activated	Not Activated	Not Activated

<sup>1)</sup> Only if an Application Reset is active or is requested in parallel.

### 8.4.6 Module Reset Behavior

**Table 8-5** lists how the various functions of the XE16xyM are affected through a reset depending on the reset type. A “X” means that this block has at least some register/bits that are affected by this reset type.

**Table 8-5 Effect of Reset on Device Functions**

Module / Function		Application Reset	Internal Application Reset	Debug Reset
<b>CPU Core</b>		X	X	X
<b>Peripherals (except SCU and RTC)</b>		X	X	X
<b>SCU</b>		X	Not affected	Not affected
<b>RTC</b>		Not affected	Not affected	X
<b>On-chip Static RAMs<sup>1)</sup></b>	<b>DPRAM</b>	Not affected, reliable	Not affected, reliable	Not affected, reliable
	<b>PSRAM</b>	Not affected, reliable	Not affected, reliable	Not affected, reliable
	<b>DSRAM</b>	Not affected, reliable	Not affected, reliable	Not affected, reliable
<b>Flash Memory</b>		X <sup>2)</sup>	X <sup>2)</sup>	Not affected, reliable
<b>JTAG Interface</b>		Not affected	Not affected	Not affected
<b>OCDS</b>		Not affected	Not affected	X
<b>Oscillator, PLL</b>		Not affected	Not affected	Not affected
<b>Port Pins</b>		Not affected <sup>3)</sup>	X	Not affected
<b>Pins ESRx</b>		Not affected	X	Not affected

<sup>1)</sup> Reliable here means that also the redundancy is not affected by the reset.

<sup>2)</sup> Parts of the flash memory block are only reset by a Power-on Reset. For more detail see the flash chapter.

<sup>3)</sup> The reset of the internal peripherals can change the data driven on the outputs, see also description of port behavior in section "Reset Behavior" in chapter "Parallel Ports".

## 8.4.7 Reset Controller Registers

### 8.4.7.1 Status Registers

After a reset has been executed, the Reset Status registers provide information on the type of the last reset. The reset status registers are updated upon each reset.

#### **RSTSTAT0**

**Reset Status 0 Register**

**ESFR (F0B2<sub>H</sub>/59<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>SW</b>		<b>CPU</b>		<b>0</b>											
rh		rh		r											

Field	Bits	Type	Description
<b>CPU</b>	[13:12]	rh	<b>CPU Reset Type Status</b> 00 <sub>B</sub> The CPU reset trigger was not relevant for the last reset 01 <sub>B</sub> Reserved 10 <sub>B</sub> The CPU reset trigger was relevant for the last reset. Internal Application and Application Resets were generated. 11 <sub>B</sub> The CPU reset trigger was relevant for the last reset. Application Reset was generated.
<b>SW</b>	[15:14]	rh	<b>Software Reset Type Status</b> 00 <sub>B</sub> The Software reset trigger was not relevant for the last reset 01 <sub>B</sub> Reserved 10 <sub>B</sub> The Software reset trigger was relevant for the last reset. Internal Application and Application Resets were generated. 11 <sub>B</sub> The Software reset trigger was relevant for the last reset. Application Reset was generated.
<b>0</b>	[11:0]	r	<b>Reserved</b> Read as 0; should be written with 0.

### RSTSTAT1

#### Reset Status 1 Register

**ESFR (F0B4<sub>H</sub>/5A<sub>H</sub>)**

**Reset Value: F000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>ST1</b>		<b>STM</b>		<b>0</b>		<b>MP</b>		<b>WDT</b>		<b>ESR2</b>		<b>ESR1</b>		<b>ESR0</b>	
rh		rh		r		rh		rh		rh		rh		rh	

Field	Bits	Type	Description
<b>ESR0</b>	[1:0]	rh	<b>ESR0 Reset Status</b> 00 <sub>B</sub> The <u>ESR0</u> reset trigger was not relevant for the last reset 01 <sub>B</sub> <u>Reserved</u> 10 <sub>B</sub> The <u>ESR0</u> reset trigger was relevant for the last reset. Internal Application, and Application Resets were generated. 11 <sub>B</sub> The <u>ESR0</u> reset trigger was relevant for the last reset. Application Reset was generated.
<b>ESR1</b>	[3:2]	rh	<b>ESR1 Reset Status</b> 00 <sub>B</sub> The <u>ESR1</u> reset trigger was not relevant for the last reset 01 <sub>B</sub> <u>Reserved</u> 10 <sub>B</sub> The <u>ESR1</u> reset trigger was relevant for the last reset. Internal Application, and Application Resets were generated. 11 <sub>B</sub> The <u>ESR1</u> reset trigger was relevant for the last reset. Application Reset was generated.
<b>ESR2</b>	[5:4]	rh	<b>ESR2 Reset Status</b> 00 <sub>B</sub> The <u>ESR2</u> reset trigger was not relevant for the last reset 01 <sub>B</sub> <u>Reserved</u> 10 <sub>B</sub> The <u>ESR2</u> reset trigger was relevant for the last reset. Internal Application, and Application Resets were generated. 11 <sub>B</sub> The <u>ESR2</u> reset trigger was relevant for the last reset. Application Reset was generated.

Field	Bits	Type	Description
<b>WDT</b>	[7:6]	rh	<b>WDT Reset Status</b> 00 <sub>B</sub> The WDT reset trigger was not relevant for the last reset 01 <sub>B</sub> Reserved 10 <sub>B</sub> The WDT reset trigger was relevant for the last reset. Internal Application, and Application Resets were generated. 11 <sub>B</sub> The WDT reset trigger was relevant for the last reset. Application Reset was generated.
<b>MP</b>	[9:8]	rh	<b>MP Reset Status</b> 00 <sub>B</sub> The MP reset trigger was not relevant for the last reset 01 <sub>B</sub> Reserved 10 <sub>B</sub> The MP reset trigger was relevant for the last reset. Internal Application, and Application Resets were generated. 11 <sub>B</sub> The MP reset trigger was relevant for the last reset. Application Reset was generated.
<b>STM</b>	[13:12]	rh	<b>Power-on for DMP_M Reset Status</b> 00 <sub>B</sub> The power-on reset for DMP_M reset trigger was not relevant for the last reset 01 <sub>B</sub> The power-on reset for DMP_M reset trigger was not relevant for the last reset 10 <sub>B</sub> The power-on reset for DMP_M reset trigger was not relevant for the last reset 11 <sub>B</sub> The power-on reset for DMP_M reset trigger was relevant for the last reset
<b>ST1</b>	[15:14]	rh	<b>Power-on for DMP_1 Reset Status</b> 00 <sub>B</sub> The power-on reset for DMP_1 reset trigger was not relevant for the last reset 01 <sub>B</sub> The power-on reset for DMP_1 reset trigger was not relevant for the last reset 10 <sub>B</sub> The power-on reset for DMP_1 reset trigger was not relevant for the last reset 11 <sub>B</sub> The power-on reset for DMP_1 reset trigger was relevant for the last reset
<b>0</b>	[11:10]	r	<b>Reserved</b> Read as 0; should be written with 0.

**RSTSTAT2**

**Reset Status 2 Register**

**ESFR (F0B6<sub>H</sub>/5B<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0						OJCONF3		OJCONF2		OJCONF1		OJCONF0		DB	
r						rh		rh		rh		rh		rh	

Field	Bits	Type	Description
<b>DB</b>	[1:0]	rh	<b>Debug Reset Status</b> 00 <sub>B</sub> The DB reset trigger was not relevant for the last reset 01 <sub>B</sub> The DB reset trigger was not relevant for the last reset 10 <sub>B</sub> The DB reset trigger was not relevant for the last reset 11 <sub>B</sub> The DB reset trigger was relevant for the last reset
<b>OJCONF0</b>	[3:2]	rh	<b>OJCONF0 Reset Status</b> Read as 0; should be written with 0.
<b>OJCONF1</b>	[5:4]	rh	<b>OJCONF1 Reset Status</b> 00 <sub>B</sub> The OJCONF1 reset trigger was not relevant for the last reset 01 <sub>B</sub> The OJCONF1 reset trigger was not relevant for the last reset 10 <sub>B</sub> The OJCONF1 reset trigger was not relevant for the last reset 11 <sub>B</sub> The OJCONF1 reset trigger was relevant for the last reset. Debug Reset was generated.
<b>OJCONF2</b>	[7:6]	rh	<b>OJCONF2 Reset Status</b> 00 <sub>B</sub> The OJCONF2 reset trigger was not relevant for the last reset 01 <sub>B</sub> The OJCONF2 reset trigger was not relevant for the last reset 10 <sub>B</sub> The OJCONF2 reset trigger was relevant for the last reset. Internal Application, and Application Resets were generated. 11 <sub>B</sub> The OJCONF2 reset trigger was not relevant for the last reset

Field	Bits	Type	Description
<b>OJCONF3</b>	[9:8]	rh	<b>OJCONF3 Reset Status</b> 00 <sub>B</sub> The OJCONF3 reset trigger was not relevant for the last reset 01 <sub>B</sub> The OJCONF3 reset trigger was not relevant for the last reset 10 <sub>B</sub> The OJCONF3 reset trigger was not relevant for the last reset 11 <sub>B</sub> The OJCONF3 reset trigger was relevant for the last reset. Application Reset was generated.
<b>0</b>	[3:2], [15:10]	r	<b>Reserved</b> Read as 0; should be written with 0.



### 8.4.7.2 Configuration Registers

These registers allow the behavioral configuration for the various reset trigger sources.

#### **RSTCON0**

**Reset Configuration 0 Register ESFR (F0B8<sub>H</sub>/5C<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>SW</b>		<b>CPU</b>		<b>0</b>											
rw		rw		rw											

Field	Bits	Type	Description
<b>CPU</b>	[13:12]	rw	<b>CPU Reset Type Selection</b> This bit field defines which reset types are generated by a CPU reset request trigger. 00 <sub>B</sub> No reset is generated 01 <sub>B</sub> Reserved, do not use this combination 10 <sub>B</sub> Internal Application, and Application Resets are generated 11 <sub>B</sub> Application Reset is generated
<b>SW</b>	[15:14]	rw	<b>Software Reset Type Selection</b> This bit field defines which reset types are generated by a software reset request trigger. 00 <sub>B</sub> No reset is generated 01 <sub>B</sub> Reserved, do not use this combination 10 <sub>B</sub> Internal Application, and Application Resets are generated 11 <sub>B</sub> Application Reset is generated
<b>0</b>	[11:0]	rw	<b>Reserved</b> Must be written with reset value 0.

## RSTCON1

**Reset Configuration 1 Register ESR (F0BA<sub>H</sub>/5D<sub>H</sub>)**

**Reset Value: 0002<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>0</b>						<b>MP</b>		<b>WDT</b>		<b>ESR2</b>		<b>ESR1</b>		<b>ESR0</b>	
rw						rw		rw		rw		rw		rw	

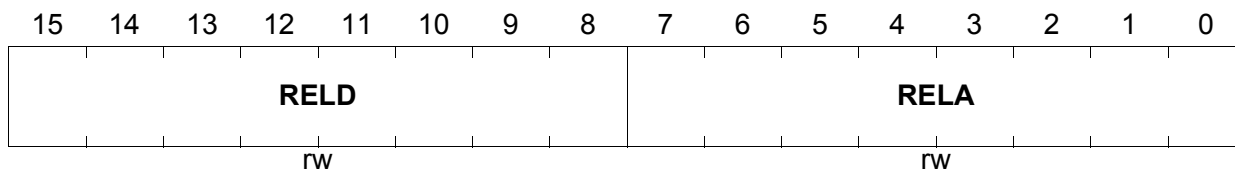
Field	Bits	Type	Description
<b>ESR0</b>	[1:0]	rw	<b>ESR0 Reset Type Selection</b> This bit field defines which reset types are generated by a <u>ESR0</u> reset request trigger. 00 <sub>B</sub> No reset is generated 01 <sub>B</sub> Reserved, do not use this combination 10 <sub>B</sub> Internal Application, and Application Resets are generated 11 <sub>B</sub> Application Reset is generated
<b>ESR1</b>	[3:2]	rw	<b>ESR1 Reset Type Selection</b> This bit field defines which reset types are generated by a <u>ESR1</u> reset request trigger. 00 <sub>B</sub> No reset is generated 01 <sub>B</sub> Reserved, do not use this combination 10 <sub>B</sub> Internal Application, and Application Resets are generated 11 <sub>B</sub> Application Reset is generated
<b>ESR2</b>	[5:4]	rw	<b>ESR2 Reset Type Selection</b> This bit field defines which reset types are generated by a <u>ESR2</u> reset request trigger. 00 <sub>B</sub> No reset is generated 01 <sub>B</sub> Reserved, do not use this combination 10 <sub>B</sub> Internal Application, and Application Resets are generated 11 <sub>B</sub> Application Reset is generated

Field	Bits	Type	Description
<b>WDT</b>	[7:6]	rw	<b>WDT Reset Type Selection</b> This bit field defines which reset types are generated by a WDT reset request trigger. 00 <sub>B</sub> No reset is generated 01 <sub>B</sub> Reserved, do not use this combination 10 <sub>B</sub> Internal Application, and Application Resets are generated 11 <sub>B</sub> Application Reset is generated
<b>MP</b>	[9:8]	rw	<b>MP Reset Type Selection</b> This bit field defines which reset types are generated by a MP reset request trigger. 00 <sub>B</sub> No reset is generated 01 <sub>B</sub> Reserved, do not use this combination 10 <sub>B</sub> Internal Application, and Application Resets are generated 11 <sub>B</sub> Application Reset is generated
<b>0</b>	[15:10]	rw	<b>Reserved</b> Should be written with 0.

**RSTCNTCON**

**Reset Counter Control RegisterESFR (F1B2<sub>H</sub>/D9<sub>H</sub>)**

**Reset Value: 0A0A<sub>H</sub>**



Field	Bits	Type	Description
RELA	[7:0]	rw	<b>Application Reset Counter Reload Value</b> This bit field defines the reload value of RSTCNTA. This value is always used when counter RSTCNTA is started. This counter value is used for Internal Application, and Application Resets. In case of an ESRx reset the counter value must be not less than the reset value.
RELD	[15:8]	rw	<b>Debug Reset Counter Reload Value</b> This bit field defines the reload value of RSTCNTD. This value is always used when counter RSTCNTD is started. This counter value is used for the Debug Reset. In case of an ESRx reset the counter value must be not less than the reset value.

## Software Reset Control Register

This register controls the software reset operation.

### SWRSTCON

**Software Reset Control RegisterESFR (F0AE<sub>H</sub>/57<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>SWCFG</b>								<b>0</b>				<b>SW RST REQ</b>	<b>SW BOO T</b>		
rw								r				w	rw		

Field	Bits	Type	Description
<b>SWBOOT</b>	0	rw	<b>Software Boot Configuration Selection</b> 0 <sub>B</sub> Bit field STSTAT.HWCFG is not updated with the content of SWCFG upon an Application Reset 1 <sub>B</sub> Bit field STSTAT.HWCFG is updated with the content of SWCFG upon an Application Reset
<b>SWRSTREQ</b>	1	w	<b>Software Reset Request</b> 0 <sub>B</sub> No software reset is requested 1 <sub>B</sub> A software reset request trigger is generated <i>Note: This bit is always read as 0.</i>
<b>SWCFG</b>	[15:8]	rw	<b>Software Boot Configuration</b> A valid software boot configuration (also different from the external applied hardware configuration) can be specified with these bits. The configuration encoding is equal to the HWCFG encoding in register STSTAT.
<b>0</b>	[7:2]	r	<b>Reserved</b> Read as 0; should be written with 0.

## **8.5 External Service Request (ESR) Pins**

The  $\overline{\text{ESR}}$  pins serve as multi-functional pins for an amount of different options:

- Act as reset trigger input
- Act as reset output
- Act as trap input
- Act as trigger input for the GSC
- Overlay with other product functions
- Independent pad configuration

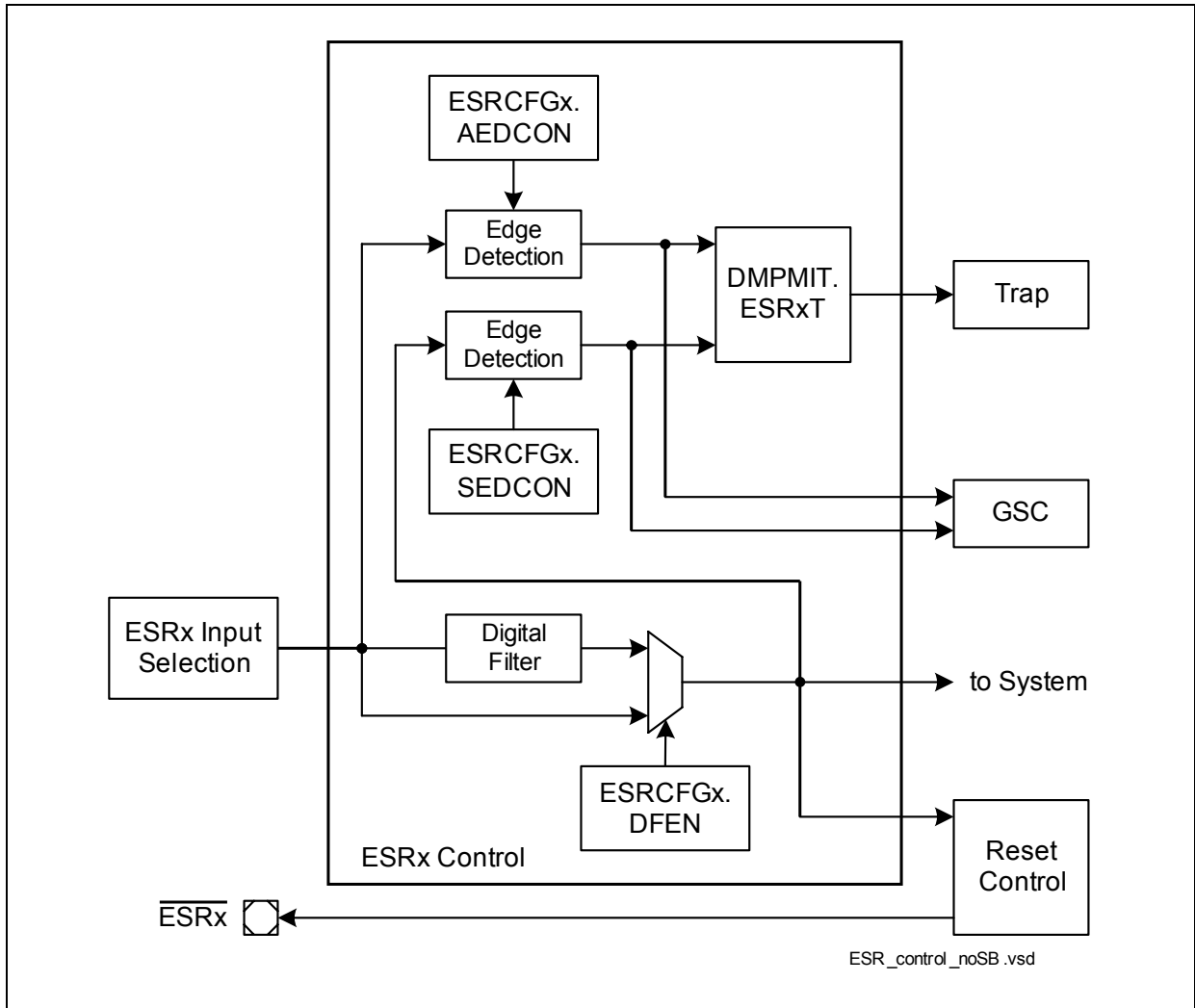
### **8.5.1 General Operation**

Each  $\overline{\text{ESR}}$  pin is equipped with an edge detection that allows the selection of the edges used as triggers. One, both, or no edge can be selected via bit field `ESRCFGx.SEDCON` if a clock is active. Additionally, there is a digital (3-stage median) filter (DF) to suppress spikes. The signal at  $\overline{\text{ESRx}}$  pin has to be held at the active signal level for at least 2 system clock cycles ( $f_{\text{SYS}}$ ) in order to generate a trigger. The digital filter can be disabled by clearing bit `ESRCFGx.DFEN`.

Each  $\overline{\text{ESRx}}$  pin can be individually configured.

If an  $\overline{\text{ESR}}$  trigger is generated please note that triggers for all purposes (reset, trap, GSC, and non SCU module functions) are generated. If some of the actions resulting out of such a trigger should not occur this has to be disabled by each feature for its own.

The pins that should be used as trigger input for an ESR operation have to be configured as input pin.



**Figure 8-19 ESRx Control**

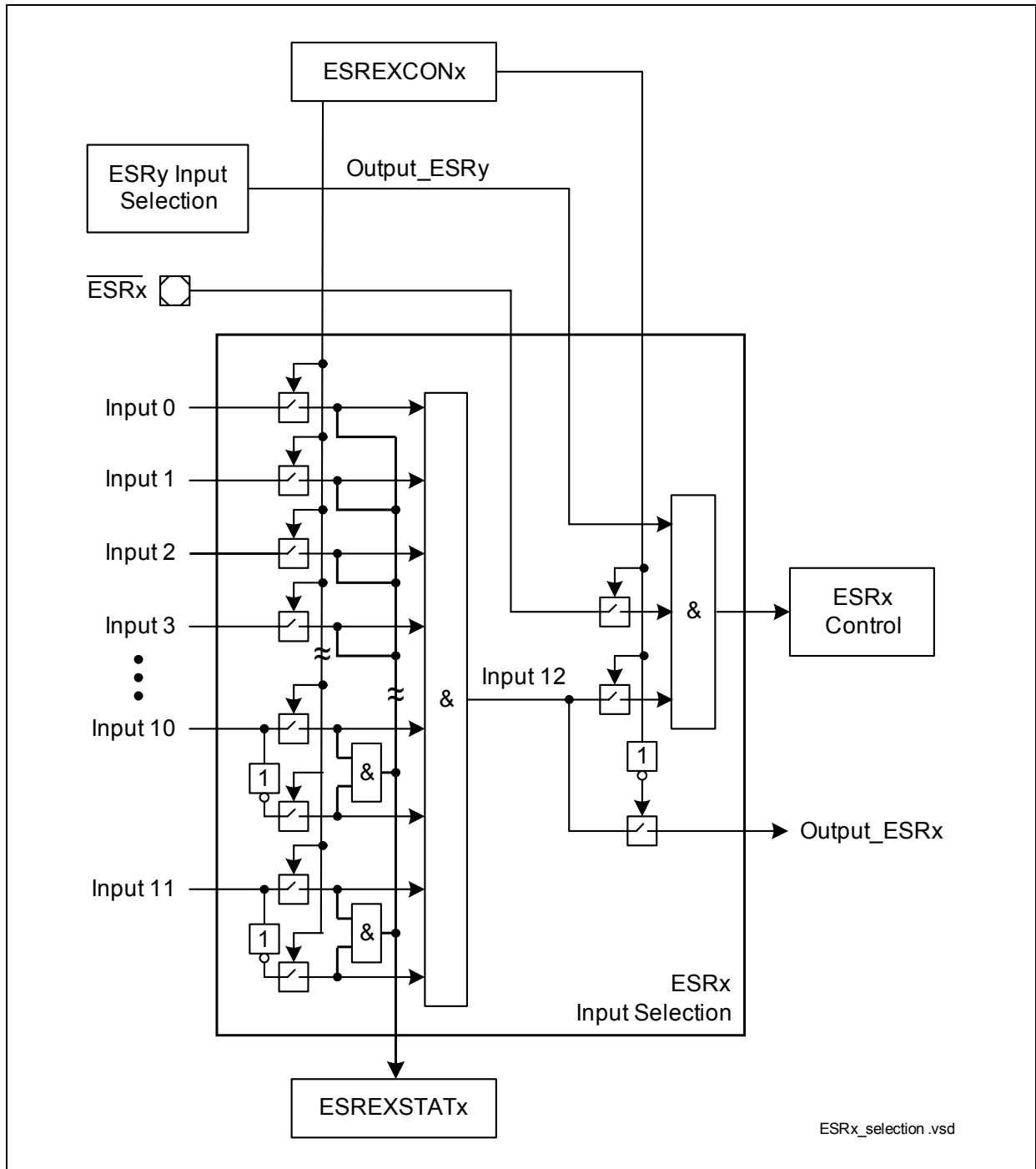
Furthermore, an overlay with other product functions (i.e. inputs of serial interfaces) can be configured via register ESREXCONx to trigger ESR operations. The conjunction of the inputs (logically AND) are used for the trigger generation. Thus, if more than one pin shall be used for ESR trigger generation, then any signal at the respective pin must have an inactive high level. In addition, it is possible to invert some inputs to support active high levels.

To extend the overlay possibilities the conjugated inputs of the ESRx input selection structure are combined to one common event in a second AND gate level with the ESR input stage and the output of the combined inputs of the other input conjugation block, if enabled (ESREXCONx.ESRIN12EN). This allows all possible inputs to trigger an ESR function even if the second ESR logic is used for other purposes.

Pin  $\overline{\text{ESR0}}$  does not offer an overlay with other product functions.

For information which other peripheral input signal is on an ESR overlay pin see [Chapter 8.17.2](#).

The following figure shows the the ESR Input selection function for  $\overline{\text{ESR1}}$  and  $\overline{\text{ESR2}}$ .



**Figure 8-20  $\overline{\text{ESR}}_x$  Input Selection**



**System Control Unit (SCU)**

Up to three  $\overline{\text{ESR}}$  pins ( $\overline{\text{ESR0}}$ / $\overline{\text{ESR1}}$ / $\overline{\text{ESR2}}$ ) are available. The availability of pins  $\overline{\text{ESR1}}$  and  $\overline{\text{ESR2}}$  is device and package dependent and is described in the data sheet.

Even if pin  $\overline{\text{ESR1}}$  or  $\overline{\text{ESR2}}$  are not available in the device an overlay with other product functions (i.e. inputs of serial interfaces) can be configured via register **ESREXCON1** **ESREXCON2** to trigger ESR operations.

### **8.5.1.1 $\overline{\text{ESR}}$ as Reset Input**

The pins  $\overline{\text{ESR}}_x$  can serve as an external reset input as well as a reset output (open drain) for Internal Application and Application Resets. Additionally several GPIO pad triggers that can be enabled additionally via register ESREXCONx interfere with the ESR pin function. GPIO and  $\overline{\text{ESR}}$  pin triggers can be enabled/disabled individually and are combined for the reset trigger generation. For more information about the reset system see [Chapter 8.4](#).

*Note: The reset output is only asserted for the duration the reset counter RSTCNTA is active. During a possible reset extension the reset output is not longer asserted.*

### **8.5.1.2 $\overline{\text{ESR}}$ as Reset Output**

If pin  $\overline{\text{ESR}}_x$  is enabled as reset output and the input level is low while the output stage is disabled (indicating that it is still driven low externally), the reset circuitry holds the chip in reset until a high level is detected on  $\overline{\text{ESR}}_x$ . The internal output stage drives a low level during reset only while RSTCNTA is active. It deactivates the output stage when the time defined by RSTCNTCON.RELA has passed. For more information about the reset system see [Chapter 8.4](#).

### **8.5.1.3 $\overline{\text{ESR}}$ as Trap Trigger**

The  $\overline{\text{ESR}}$  can request traps. The control mechanism if and which trap is requested is located in the trap control logic. For more information see [Chapter 8.13](#).

### **8.5.1.4 $\overline{\text{ESR}}$ as Trigger Input for the GSC**

The  $\overline{\text{ESR}}$  can be used to request a change in the Control Mode. For more information see [Chapter 8.7](#).

### **8.5.1.5 Overlay with other Product Functions**

Additionally other port inputs (e.g. serial communication input) can be used to generate  $\overline{\text{ESR}}$  operations. For more information which other peripheral input is on an ESR overlay pin see [Chapter 8.17](#).

This feature can be used for various applications:

- Wake-up from a Clock-off Mode on an external Interrupt or CCU6x trigger and on a CAN or USIC operation
- Request to enter a Clock-off Mode on an external Interrupt or CCU6x trigger and on a CAN or USIC operation
- Stop input for the CCU6x modules on an external event

For more information about the external interrupt trigger see [Chapter 8.9](#).

### 8.5.1.6 Pad Configuration for ESR Pads

The configuration is selected via bit field ESRCFGx.PC.

The pad functionality control can be configured independently for each pin, comprising:

- A selection of the driver type (open-drain or push-pull)
- An enable function for the output driver (input and/or output capability)
- An enable function for the pull-up/down resistance

The following table defines the coding of the bit fields PC in registers ESRCFG0, ESRCFG1, and ESRCFG2.

*Note: The coding is the same as for the port register bit fields Pn\_IOCRx.PC.*

**Table 8-6 PC Coding**

PCx[3:0]	Selected Pull-up/Pull-down / Selected Output Function	I/O	Output Characteristics
0000 <sub>B</sub>	No pull device activated	Input is not inverted, the input stage is active in power-down mode	
0001 <sub>B</sub>	Pull-down device activated		
0010 <sub>B</sub>	Pull-up device activated		
0011 <sub>B</sub>	No pull device activated		
0100 <sub>B</sub>	No pull device activated	Input is inverted, the input stage is active in power-down mode	
0101 <sub>B</sub>	Pull-down device activated		
0110 <sub>B</sub>	Pull-up device activated		
0111 <sub>B</sub>	No pull device activated		
1000 <sub>B</sub>	Output of ESRCFGx.OUT	Output, the input stage is not inverted and active in power-down mode	Push-pull
1001 <sub>B</sub>	Output of ESRCFGx.OUT		
1010 <sub>B</sub>	Output drives a 0 for an Internal Application Reset, a 1 otherwise.		
1011 <sub>B</sub>	Output drives a 0 for an Application Reset, a 1 otherwise.		
1100 <sub>B</sub>	Output of ESRCFGx.OUT		Open-drain, a pull-up device is activated while the output is not driving a 0
1101 <sub>B</sub>	Output of ESRCFGx.OUT		
1110 <sub>B</sub>	Output drives a 0 for an Internal Application Reset		
1111 <sub>B</sub>	Output drives a 0 for an Application Reset		

## 8.5.2 **ESR Control Registers**

### 8.5.2.1 **Configuration Registers**

#### **ESR External Control Register**

The ESR External Control registers contain enable/disable bits for the different inputs that can lead to an ESR action. For ESR0 this option is not available.

#### **ESREXCON1**

##### **ESR1 External Control Register**

**SFR (FF32<sub>H</sub>/99<sub>H</sub>)**

**Reset Value: 0001<sub>H</sub>**

#### **ESREXCON2**

##### **ESR2 External Control Register**

**SFR (FF34<sub>H</sub>/9A<sub>H</sub>)**

**Reset Value: 0001<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>ESR N IN11 EN</b>	<b>ESR N IN10 EN</b>	<b>ESR IN12 EN</b>	<b>ESR IN11 EN</b>	<b>ESR IN10 EN</b>	<b>ESR IN9 EN</b>	<b>ESR IN8 EN</b>	<b>ESR IN7 EN</b>	<b>ESR IN6 EN</b>	<b>ESR IN5 EN</b>	<b>ESR IN4 EN</b>	<b>ESR IN3 EN</b>	<b>ESR IN2 EN</b>	<b>ESR IN1 EN</b>	<b>ESR IN0 EN</b>	<b>ESR EN</b>
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
<b>ESREN</b>	0	rw	<b>ESRy Pin Enable</b> This bit enables/disables the <u>ESRy</u> pin for the activation of all <u>ESRy</u> related actions. 0 <sub>B</sub> The input from pin <u>ESRy</u> is disabled 1 <sub>B</sub> The input from pin <u>ESRy</u> is enabled
<b>ESRINxEN (x = 0-11)</b>	x+1	rw	<b>ESR Input X Enable</b> This bit enables/disables the input x for the activation of all <u>ESRy</u> related actions. 0 <sub>B</sub> The input is disabled 1 <sub>B</sub> The input is enabled

Field	Bits	Type	Description
<b>ESRIN12EN</b>	13	rw	<b>ESR Input 12 Enable</b> This bit enables/disables the input 12 for the activation of all $\overline{\text{ESRy}}$ related actions. $0_B$ The input 12 is disabled for the activation of all $\overline{\text{ESRy}}$ related actions. It is used in the second conjugation stage of the other ESRz Input Selection. $1_B$ The input 12 is enabled for the activation of all $\overline{\text{ESRy}}$ related actions.
<b>ESRNIN10EN</b>	14	rw	<b>Negated ESR Input 10 Enable</b> This bit enables/disables the negated input 10 for the activation of all $\overline{\text{ESRy}}$ related actions. $0_B$ The input is disabled $1_B$ The input is enabled
<b>ESRNIN11EN</b>	15	rw	<b>Negated ESR Input 11 Enable</b> This bit enables/disables the negated input 11 for the activation of all $\overline{\text{ESRy}}$ related actions. $0_B$ The input is disabled $1_B$ The input is enabled

**ESREXSTAT1**

**ESR1 External Status Register SFR (FF36<sub>H</sub>/9B<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

**ESREXSTAT2**

**ESR2 External Status Register SFR (FF38<sub>H</sub>/9C<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0			IN11	IN10	IN9	IN8	IN7	IN6	IN5	IN4	IN3	IN2	IN1	IN0	ESR
r			rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
<b>ESR</b>	0	rh	<b>Input ESRy Status</b> This bit is set upon a trigger on input x if ESREXCONy.ESREN was set. This bit can be cleared only by software. 0 <sub>B</sub> No trigger for input <u>ESRy</u> occurred 1 <sub>B</sub> A trigger for <u>ESRy</u> occurred since it was cleared last time
<b>INx</b> (x = 0-11)	x+1	rh	<b>Input x Status</b> This bit is set upon a trigger on input x if ESREXCONy.ESRINxEN was set for <u>ESRy</u> . This bit can be cleared only by software. 0 <sub>B</sub> No trigger for input x occurred 1 <sub>B</sub> A trigger for input x occurred since it was cleared last time
<b>0</b>	[15:13]	r	<b>Reserved</b> Read as 0; should be written with 0.

**CLRESREXSTAT1**

**Clear ESR1 External Status RegisterSFR (FF3A<sub>H</sub>/9D<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

**CLRESREXSTAT2**

**Clear ESR2 External Status RegisterSFR (FF3C<sub>H</sub>/9E<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0			IN11	IN10	IN9	IN8	IN7	IN6	IN5	IN4	IN3	IN2	IN1	IN0	ESR
r			w	w	w	w	w	w	w	w	w	w	w	w	w

Field	Bits	Type	Description
<b>ESR</b>	0	w	<b>Clear Input ESRy Status</b> Setting this bit clears the bit ESREXSTATy.ESR. This bit always read as zero. 0 <sub>B</sub> No effect 1 <sub>B</sub> Bit ESREXSTATy.ESR is cleared <i>Note: This bit is always read as 0.</i>
<b>INx</b> (x = 0-11)	x+1	w	<b>Clear Input x Status</b> Setting this bit clears the associated bit ESREXSTATy.INx. This bit always read as zero. 0 <sub>B</sub> No effect 1 <sub>B</sub> Bit ESREXSTATy.INx is cleared <i>Note: This bit is always read as 0.</i>
<b>0</b>	[15:13]	r	<b>Reserved</b> Read as 0; should be written with 0.

## **ESR Configuration Register**

The ESR configuration registers contains bits required for the behavioral control of the ESR pins.

### **ESRCFG0**

**ESR0 Configuration Register**      **ESFR (F100<sub>H</sub>/80<sub>H</sub>)**      **Reset Value: 000E<sub>H</sub>**

### **ESRCFG1**

**ESR1 Configuration Register**      **ESFR (F102<sub>H</sub>/81<sub>H</sub>)**      **Reset Value: 0002<sub>H</sub>**

### **ESRCFG2**

**ESR2 Configuration Register**      **ESFR (F104<sub>H</sub>/82<sub>H</sub>)**      **Reset Value: 0002<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0					AEDCON		SEDCON		IN	OUT	DF EN	PC			
r					rw		rw		rh	rh	rw	rw			

Field	Bits	Type	Description
<b>PC</b>	[3:0]	rw	<b>Pin Control of <u>ESRx</u></b> This bit field controls the behavior of the associated <u>ESRx</u> pin. The coding is described in <a href="#">Table 8-6</a> .
<b>DFEN</b>	4	rw	<b>Digital Filter Enable</b> This bit defines if the 3-stage median filter of the <u>ESRx</u> is used or bypassed. 0 <sub>B</sub> The filter is bypassed 1 <sub>B</sub> The filter is used
<b>OUT</b>	5	rh	<b>Data Output</b> This bit can be used as output value for the associated <u>ESRx</u> pin. 0 <sub>B</sub> If selected, the output level is 0 1 <sub>B</sub> If selected, the output level is 1 This bit is controlled via bit field ESRDAT.MOUTx.
<b>IN</b>	6	rh	<b>Data Input</b> This bit monitors the input value at the associated <u>ESRx</u> pin.



Field	Bits	Type	Description
<b>SEDCON</b>	[8:7]	rw	<b>Synchronous Edge Detection Control</b> This bit field defines the edges that lead to an $\overline{\text{ESRx}}$ trigger of the synchronous path. 00 <sub>B</sub> No trigger is generated 01 <sub>B</sub> A trigger is generated upon a raising edge 10 <sub>B</sub> A trigger is generated upon a falling edge 11 <sub>B</sub> A trigger is generated upon a raising AND falling edge Other combinations than 00 <sub>B</sub> are only allowed if bit field AEDCON is configured to 00 <sub>B</sub> .
<b>AEDCON</b>	[10:9]	rw	<b>Asynchronous Edge Detection Control</b> This bit field defines the edges that lead to an $\overline{\text{ESRx}}$ trigger of the asynchronous path. 00 <sub>B</sub> No trigger is generated 01 <sub>B</sub> A trigger is generated upon a raising edge 10 <sub>B</sub> A trigger is generated upon a falling edge 11 <sub>B</sub> A trigger is generated upon a raising AND falling edge Other combinations than 00 <sub>B</sub> are only allowed if bit field SEDCON is configured to 00 <sub>B</sub> .
<b>0</b>	[15:11]	r	<b>Reserved</b> Read as 0; should be written with 0.

## 8.5.3 ESR Data Register

### 8.5.3.1 ESRDAT

The  $\overline{\text{ESR}}$  data register contains bits required if  $\overline{\text{ESRx}}$  are used as data ports.

#### ESRDAT

##### ESR Data Register

**ESFR (F106<sub>H</sub>/83<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0										MOUT2		MOUT1		MOUT0	
r										w		w		w	

Field	Bits	Type	Description
<b>MOUT0</b>	[1:0]	w	<b>Modification of Data Output at <math>\overline{\text{ESR0}}</math></b> Writing to this bit field can modify the content of bit $\overline{\text{ESRCFG0.OUT}}$ which updates the output at $\overline{\text{ESR0}}$ . It always reads 0. 00 <sub>B</sub> $\overline{\text{ESR0}}$ output (bit $\overline{\text{ESRCFG0.OUT}}$ ) is unchanged 01 <sub>B</sub> $\overline{\text{ESR0}}$ output (bit $\overline{\text{ESRCFG0.OUT}}$ ) is set 10 <sub>B</sub> $\overline{\text{ESR0}}$ output (bit $\overline{\text{ESRCFG0.OUT}}$ ) is cleared 11 <sub>B</sub> Reserved, do not use this combination <i>Note: This bit is always read as 0.</i>
<b>MOUT1</b>	[3:2]	w	<b>Modification of Data Output at <math>\overline{\text{ESR1}}</math></b> Writing to this bit field can modify the content of bit $\overline{\text{ESRCFG1.OUT}}$ which updates the output at $\overline{\text{ESR1}}$ . It always reads 0. 00 <sub>B</sub> $\overline{\text{ESR1}}$ output (bit $\overline{\text{ESRCFG1.OUT}}$ ) is unchanged 01 <sub>B</sub> $\overline{\text{ESR1}}$ output (bit $\overline{\text{ESRCFG1.OUT}}$ ) is set 10 <sub>B</sub> $\overline{\text{ESR1}}$ output (bit $\overline{\text{ESRCFG1.OUT}}$ ) is cleared 11 <sub>B</sub> Reserved, do not use this combination <i>Note: This bit is always read as 0.</i>

Field	Bits	Type	Description
<b>MOUT2</b>	[5:4]	w	<b>Modification of Data Output at ESR2</b> Writing to this bit field can modify the content of bit ESRCFG2.OUT which updates the output at ESR2. It always reads 0. 00 <sub>B</sub> ESR2 output (bit ESRCFG2.OUT) is unchanged 01 <sub>B</sub> ESR2 output (bit ESRCFG2.OUT) is set 10 <sub>B</sub> ESR2 output (bit ESRCFG2.OUT) is cleared 11 <sub>B</sub> Reserved, do not use this combination <i>Note: This bit is always read as 0.</i>
<b>0</b>	[15:6]	w	<b>Reserved</b> Read as 0; should be written with 0.

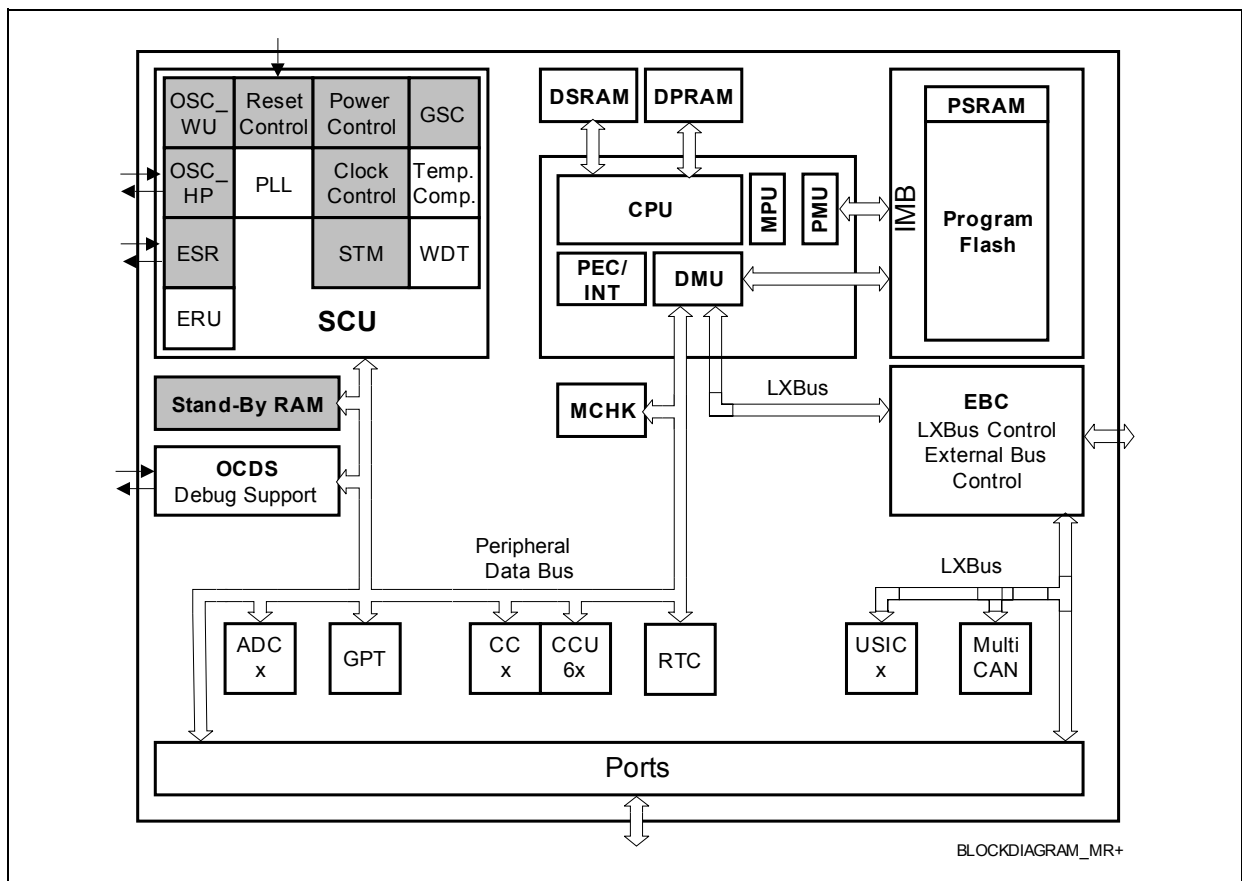
## 8.6 Power Supply and Control

The XE16xyM can run from a single external power supply. The core supply voltages can be generated by on-chip Embedded Voltage Regulators (EVRs).

### Power Domains

The I/O part is divided in two parts DMP\_A and DMP\_B. DMP\_A contains all ADC related I/Os and DMP\_B the remaining system and communication I/Os.

The major part of the on-chip logic is located in an independent core power domain (DMP\_1). A second power domain (DMP\_M), marked grey in the figure below, controls important device infrastructure plus a Standby RAM (SBRAM).



**Figure 8-21 XE16xyM Power Domain Structure**

### Power Supply and Control Functions

The power supply and control is divided into following parts:

- monitoring of the supply voltage
- controlling and adjusting the supply voltage

**System Control Unit (SCU)**

The supply voltage of pad IO domain for system and communication I/Os (power domain DMP\_B) is monitored by a Supply WatchDog (SWD, see [Chapter 8.6.1](#)).

The core voltage for each of the two core supply domains is supervised by a separate Power Validation Circuit (PVC) that provides two monitoring levels. Each monitoring level can request an interrupt (e.g. power-fail warning) or a reset depending on the voltage level. A PVC is used to detect under voltage due to an external short (see [Chapter 8.6.2](#)).

By controlling the regulator, the core power can be switched off to save the leakage current (see [Chapter 8.6.3](#)).

**Table 8-7 XE16xyM Power Domains Supply and Control**

<b>Power Domain</b>	<b>Supply Source</b>	<b>Supply Voltage [V]</b>	<b>Supply Checked by</b>
Pad IO domain (DMP_B)	External supply	$V_{DDPB}$ : 3.0 ... 5.5 typ See data sheet	SWD
ADC IO domain (DMP_A)	External supply	$V_{DDPA}$ : 3.0 ... 5.5 typ See data sheet	-
Core domain (DMP_M and DMP_1)	EVR_M EVR_1	$V_{DDIM}$ , $V_{DDI1}$ : 1.5 typ See data sheet	PVC_1, PVC_M

### **8.6.1 Supply Watchdog (SWD)**

The supply voltage of the pad I/O domain for systems and communication I/Os (DMP\_B) is monitored to validate the overall power supply. The external supply voltage is monitored for following purposes:

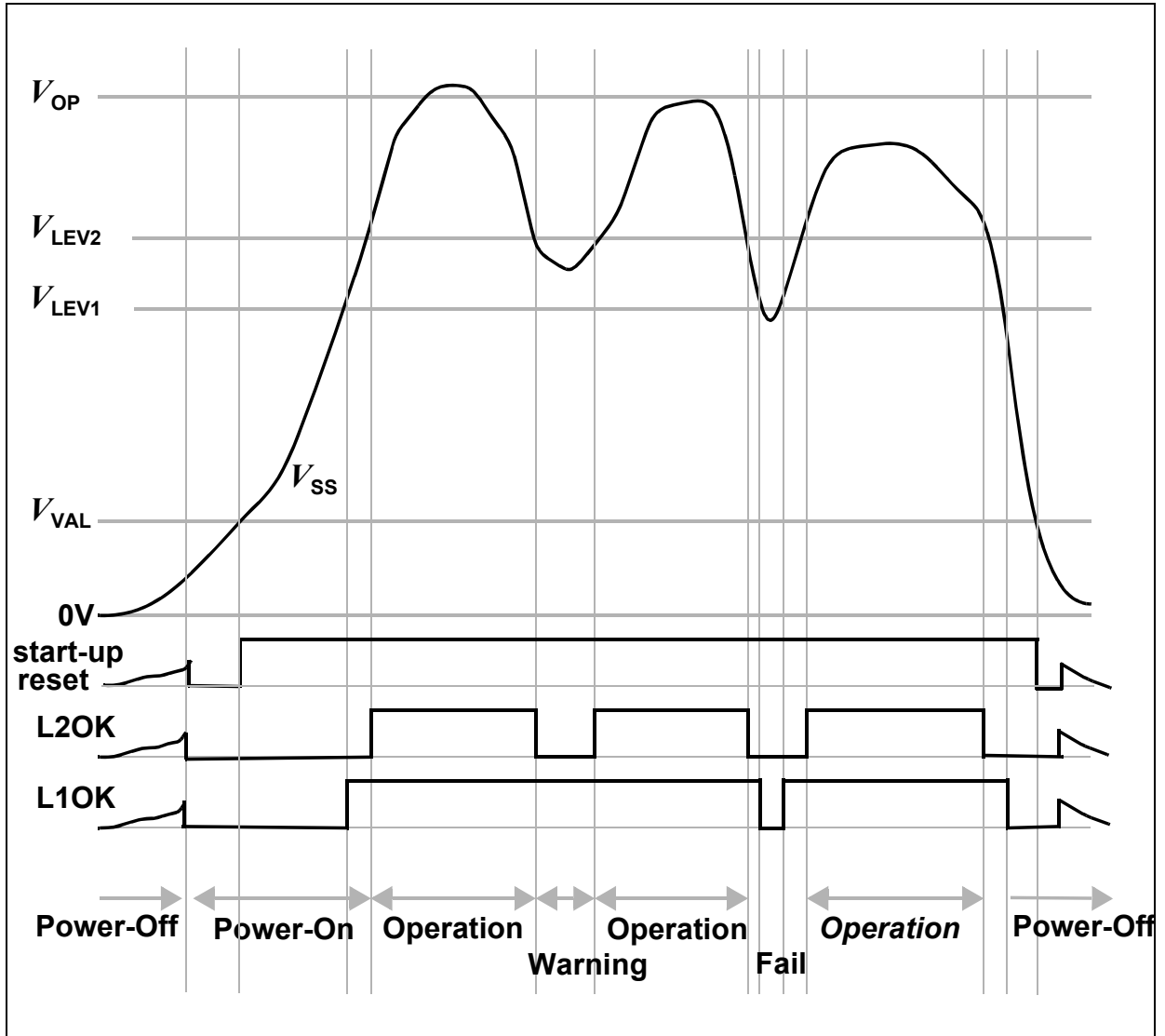
- **POR**  
Detecting the ramp-up of the external supply voltage, so the device can be started without requiring an external power-on reset (PORST).
- **Brown-out**  
Detecting the ramp-down of the external supply voltage, so the device can be brought into a save state without requiring an external power-on reset (PORST).
- Monitoring the external power supply allows the usage of a low-cost regulator without additional status signals (standard 3-pin device).
- Guarantee that the supply voltage for the EVRs is sufficient to generate a valid core voltage under every operating condition

#### **Feature list**

The following list is a summary of the SWD functions.

- Trigger a power-on reset whenever the supply falls and as long as the supply remains below  $V_{VAL}$
- Two completely independent threshold levels and comparators
- 16 selectable threshold levels
- Power Saving Mode (only  $V_{VAL}$  detection active)

## Operating the SWD



**Figure 8-22 SWD Power Validation Example**

The lower fix threshold  $V_{VAL}$  defines the absolute minimum operation voltage for the IO domain. If  $V_{VAL}$  has not been reached the device is held in reset. When  $V_{DDPB}$  raises above  $V_{VAL}$ , bit **SWDCON1.PON** is set.

*Note: The physical value for  $V_{VAL}$  can found in the XE16xyM data sheet.*

The SWD provides two adjustable threshold levels (LEV1 and LEV2) that can be individually programmed, via **SWDCON0.LEV1V** and **SWDCON0.LEV2V**, and deliver a compare value each. The two compare results can be monitored via bits **SWDCON0.L1OK** and **SWDCON0.L2OK**. A reset or interrupt request can be generated while the voltage level is below or equal/above the configured level of a threshold. If an action and which action is triggered by each threshold can be configured via bits

## **System Control Unit (SCU)**

SWDCON0.LxRSTEN and SWDCON0.LxINTEN and bit field SWDCON0.LxALEV (x = 1,2).

The SWD control (programming of the threshold levels) is done by software only.

With these features, an external supply watchdog, e.g. integrated in some external VR, can be replaced. It detects the minimum specified supply voltage level and can be configured to monitor other voltage levels.

*Note: If the  $\overline{PORST}$  pin is used it has the same functionality as the min-power detection of the SWD.*

### **Power-Saving Mode of the SWD**

The two configurable thresholds can be disabled if not needed. This is called the SWD Power Saving Mode. The minimum operating voltage detection (POR/Brown-out detection) can not be disabled and it is always active. The SWD Power Saving Mode is entered by setting bit **SWDCON1.POWENSET** and exit by setting bit SWDCON1.POWENCLR. If the SWD Power Saving Mode is active is indicated by bit SWDCON1.POWEN.

*Note: The reset request and interrupt request action should be switched off before entering power-save mode.*



### 8.6.1.1 SWD Control Registers

The following registers are the software interface for the SWD.

#### SWDCON0

##### SWD Control 0 Register

**ESFR (F080<sub>H</sub>/40<sub>H</sub>)**

**Reset Value: 0941<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>L2 A LEV</b>	<b>L2 RST EN</b>	<b>L2 INT EN</b>	<b>L2 OK</b>	<b>LEV2V</b>				<b>L1 A LEV</b>	<b>L1 RST EN</b>	<b>L1 INT EN</b>	<b>L1 OK</b>	<b>LEV1V</b>			
rw	rw	rw	rh	rw				rw	rw	rw	rh	rw			

Field	Bits	Type	Description
<b>LEV1V</b>	[3:0]	rw	<b>Level Threshold 1 Voltage</b> This bit field defines the voltage level that is used as threshold 1 check level. The values of the level thresholds are listed in the data sheet.
<b>L1OK</b>	4	rh	<b>Level Threshold 1 Check Result</b> 0 <sub>B</sub> The supply voltage is below the Level Threshold 1 voltage LEV1V 1 <sub>B</sub> The supply voltage is equal or above the Level Threshold 1 voltage LEV1V
<b>L1INTEN</b>	5	rw	<b>Level Threshold 1 Interrupt Request Enable</b> This bit field defines if an interrupt is requested if the supply voltage comparison matches the action level L1ALEV. 0 <sub>B</sub> No interrupt is requested 1 <sub>B</sub> An interrupt is requested
<b>L1RSTEN</b>	6	rw	<b>Level Threshold 1 Reset Request Enable</b> This bit field defines if a reset is requested if the supply voltage comparison matches the action level L1ALEV. 0 <sub>B</sub> No reset is requested 1 <sub>B</sub> An reset is requested

Field	Bits	Type	Description
<b>L1ALEV</b>	7	rw	<b>Level Threshold 1 Action Level</b> $0_B$ When the supply voltage is below the Level Threshold 1 voltage LEV1V the actions configured by bits L1INTEN and L1RSTEN are requested $1_B$ When the supply voltage is equal or above the Level Threshold 1 voltage LEV1V the actions configured by bits L1INTEN and L1RSTEN are requested
<b>LEV2V</b>	[11:8]	rw	<b>Level Threshold 2 Voltage</b> This bit field defines the voltage level that is used as check level threshold 2. The values of the level thresholds are listed in the data sheet.
<b>L2OK</b>	12	rh	<b>Level Threshold 2 Check Result</b> $0_B$ The supply voltage is below the Level Threshold 2 voltage LEV2V $1_B$ The supply voltage is equal or above the Level Threshold 2 voltage LEV2V
<b>L2INTEN</b>	13	rw	<b>Level Threshold 2 Interrupt Request Enable</b> This bit field defines if an interrupt is requested if the supply voltage comparison matches the action level L2ALEV. $0_B$ No interrupt is requested $1_B$ An interrupt is requested
<b>L2RSTEN</b>	14	rw	<b>Level Threshold 2 Reset Request Enable</b> This bit field defines if a reset is requested if the supply voltage comparison matches the action level L2ALEV. $0_B$ No reset is requested $1_B$ An reset is requested

Field	Bits	Type	Description
<b>L2ALEV</b>	15	rw	<b>Level Threshold 2 Action Level</b> $0_B$ When the supply voltage is below the Level Threshold 2 voltage LEV2V the actions configured by bits L2INTEN and L2RSTEN are requested $1_B$ When the supply voltage is equal or above the Level Threshold 2 voltage LEV2V the actions configured by bits L2INTEN and L2RSTEN are requested



### **8.6.2 Monitoring the Voltage Level of a Core Domain**

A Power Validation Circuit (PVC) monitors the internal core supply voltage of a core domain. It can be configured to monitor two programmable independent voltage levels.

The voltage of the core domain is monitored by PVC\_1 and PVC\_M.

#### **Feature list**

The following list summarizes the features of a PVC.

- Two independent comparators
- Threshold levels selectable
- Shut-off, which disables the complete module
- Configurable action level

A PVC provides two adjustable threshold levels (LEV1 and LEV2) that can be individually programmed via PVCxCON0.LEV1V and PVCxCON0.LEV2V (x = M or 1) PVC1CON0.LEV1V and PVC1CON0.LEV2V. The current supply level of a domain is compared with the threshold values. The two compare results can be monitored via bits PVCxCON0.LEV1OK and PVCxCON0.LEV2OK (x = M or 1) PVC1CON0.LEV1OK and PVC1CON0.LEV2OK

A reset or interrupt request can be generated in case the core domain voltage level is below or equal / above the configured threshold level. An interrupt is requested if bit PVCxCON0.L1INTEN and / or PVCxCON0.L2INTEN (x = M or 1) PVC1CON0.L1INTEN and / or PVC1CON0.L2INTEN is set. A reset is requested if bit PVCxCON0.L1RSTEN and / or PVCxCON0.L2RSTEN (x = M or 1) PVC1CON0.L1RSTEN and / or PVC1CON0.L2RSTEN is set.

*Note: For a single threshold both interrupt and reset request generation should not be enabled at the same time.*

### 8.6.2.1 PVC Status and Control Registers

These registers are the software interface for PVC\_1 and PVC\_M.

#### PVC1CON0

#### PVC\_1 Control Step 0 Register

**ESFR (F014<sub>H</sub>/0A<sub>H</sub>)**

**Reset Value: 0504<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>L2 AS EN</b>	<b>L2 RST EN</b>	<b>L2 INT EN</b>	<b>L2 A LEV</b>	<b>LEV 2 OK</b>	<b>LEV2V</b>			<b>L1 AS EN</b>	<b>L1 RST EN</b>	<b>L1 INT EN</b>	<b>L1 A LEV</b>	<b>LEV 1 OK</b>	<b>LEV1V</b>		
rw	rw	rw	rw	rh	rw			rw	rw	rrw	rw	rh	rw		

Field	Bits	Type	Description
<b>LEV1V</b>	[2:0]	rw	<b>Level Threshold 1 Voltage</b> This bit field defines the Level Threshold 1 that is compared with the DMP_1 core voltage. The values for the different configurations are listed in the data sheet.
<b>LEV1OK</b>	3	rh	<b>Level Threshold 1 Check Result</b> 0 <sub>B</sub> The core supply voltage of the DMP_1 is below Level Threshold 1 voltage LEV1V 1 <sub>B</sub> The core supply voltage of the DMP_1 is equal or above the Level Threshold 1 voltage LEV1V
<b>L1ALEV</b>	4	rw	<b>Level Threshold 1 Action Level</b> 0 <sub>B</sub> When the core supply voltage is below Level Threshold 1 voltage LEV1V the action configured by bits L1INTEN, L1RSTEN, and L1ASEN are requested 1 <sub>B</sub> When the core supply voltage is equal or above Level Threshold 1 voltage LEV1V the actions configured by bits L1INTEN, L1RSTEN, and L1ASEN are requested
<b>L1INTEN</b>	5	rw	<b>Level Threshold 1 Interrupt Request Enable</b> This bit defines if an interrupt request trigger is requested if the comparison check was successful. When a check is successful is defined via bit L1ALEV. 0 <sub>B</sub> No interrupt is requested 1 <sub>B</sub> An interrupt is requested

Field	Bits	Type	Description
<b>L1RSTEN</b>	6	rw	<b>Level Threshold 1 Reset Request Enable</b> This bit defines if a reset request trigger is requested if the comparison check was successful. When a check is successful is defined via bit L1ALEV. 0 <sub>B</sub> No reset is requested 1 <sub>B</sub> An reset is requested
<b>L1ASEN</b>	7	rw	<b>Level Threshold 1 Asynchronous Action Enable</b> This bit defines if asynchronous action can be performed if the comparison check was successful. When a check is successful is defined via bit L1ALEV. 0 <sub>B</sub> No asynchronous actions are performed 1 <sub>B</sub> Asynchronous actions can be performed
<b>LEV2V</b>	[10:8]	rw	<b>Level Threshold 2 Voltage</b> This bit field defines the level of threshold 2 that is compared with the DMP_1 core voltage.. The values for the different configurations are listed in the data sheet.
<b>LEV2OK</b>	11	rh	<b>Level Threshold 2 Check Result</b> 0 <sub>B</sub> The core supply voltage of the DMP_1 is below the Level Threshold 2 LEV2V 1 <sub>B</sub> The core supply voltage of the DMP_1 is equal or above the Level Threshold 2 LEV2V
<b>L2ALEV</b>	12	rw	<b>Level Threshold 2 Action Level</b> 0 <sub>B</sub> When the core supply voltage is below the Level Threshold 2 voltage LEV2V the action configured by bits L2INTEN, L2RSTEN, and L2ASEN are requested 1 <sub>B</sub> When the core supply voltage is equal or above the Level Threshold 2 voltage LEV2V the action configured by bits L2INTEN, L2RSTEN, and L2ASEN are requested
<b>L2INTEN</b>	13	rw	<b>Level Threshold 2 Interrupt Request Enable</b> This bit defines if an interrupt request trigger is requested if the comparison check was successful. When a check is successful is defined via bit L2ALEV. 0 <sub>B</sub> No interrupt is requested 1 <sub>B</sub> An interrupt is requested

Field	Bits	Type	Description
<b>L2RSTEN</b>	14	rw	<b>Level Threshold 2 Reset Request Enable</b> This bit defines if a reset request trigger is requested if the comparison check was successful. When a check is successful is defined via bit L2ALEV. 0 <sub>B</sub> No reset is requested 1 <sub>B</sub> An reset is requested
<b>L2ASEN</b>	15	rw	<b>Level Threshold 2 Asynchronous Action Enable</b> This bit defines if asynchronous action can be performed if the comparison check was successful. When a check is successful is defined via bit L2ALEV. 0 <sub>B</sub> No asynchronous actions are performed 1 <sub>B</sub> Asynchronous actions can be performed



**PVCMCON0**

**PVC\_M Control Step 0 Register**

**MEM (F1E4<sub>H</sub>/--)**

**Reset Value: 0544<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>L2 AS EN</b>	<b>L2 RST EN</b>	<b>L2 INT EN</b>	<b>L2 A LEV</b>	<b>LEV 2 OK</b>	<b>LEV2V</b>			<b>L1 AS EN</b>	<b>L1 RST EN</b>	<b>L1 INT EN</b>	<b>L1 A LEV</b>	<b>LEV 1 OK</b>	<b>LEV1V</b>		
rw	rw	rw	rw	rh	rw			rw	rw	rw	rw	rh	rw		

Field	Bits	Type	Description
<b>LEV1V</b>	[2:0]	rw	<b>Level Threshold 1 Voltage</b> This bit field defines the Level Threshold 1 that is compared with the DMP_M core supply voltage. The values for the different configurations are listed in the data sheet.
<b>LEV1OK</b>	3	rh	<b>Level Threshold 1 Check Result</b> 0 <sub>B</sub> The core supply voltage of the DMP_M is below Level Threshold 1 voltage LEV1V 1 <sub>B</sub> The core supply voltage of the DMP_M is equal or above the Level Threshold 1 voltage LEV1V
<b>L1ALEV</b>	4	rw	<b>Level Threshold 1 Action Level</b> 0 <sub>B</sub> When the core supply voltage is below Level Threshold 1 voltage LEV1V the action configured by bits L1INTEN, L1RSTEN, and L1ASEN are requested 1 <sub>B</sub> When the core supply voltage is equal or above Level Threshold 1 voltage LEV1V the actions configured by bits L1INTEN, L1RSTEN, and L1ASEN are requested
<b>L1INTEN</b>	5	rw	<b>Level Threshold 1 Interrupt Request Enable</b> This bit defines if an interrupt request trigger is requested if the comparison check was successful. When a check is successful is defined via bit L1ALEV. 0 <sub>B</sub> No interrupt is requested 1 <sub>B</sub> An interrupt is requested

Field	Bits	Type	Description
<b>L1RSTEN</b>	6	rw	<b>Level Threshold 1 Reset Request Enable</b> This bit defines if a reset request trigger is requested if the comparison check was successful. When a check is successful is defined via bit L1ALEV. 0 <sub>B</sub> No reset is requested 1 <sub>B</sub> An reset is requested
<b>L1ASEN</b>	7	rw	<b>Level Threshold 1 Asynchronous Action Enable</b> This bit defines if asynchronous action can be performed if the comparison check was successful. When a check is successful is defined via bit L1ALEV. 0 <sub>B</sub> No asynchronous actions are performed 1 <sub>B</sub> Asynchronous actions can be performed
<b>LEV2V</b>	[10:8]	rw	<b>Level Threshold 2 Voltage</b> This bit field defines the Level Threshold 2 that is compared with the DMP_M core supply voltage. The values for the different configurations are listed in the data sheet.
<b>LEV2OK</b>	11	rh	<b>Level Threshold 2 Check Result</b> 0 <sub>B</sub> The core supply voltage of the DMP_M is below Level Threshold 2 voltage LEV2V 1 <sub>B</sub> The core supply voltage of the DMP_M is equal or above the Level Threshold 2 voltage LEV2V
<b>L2ALEV</b>	12	rw	<b>Level Threshold 2 Action Level</b> 0 <sub>B</sub> When the core supply voltage is below the Level Threshold 2 voltage LEV2V the action configured by bits L2INTEN, L2RSTEN, and L2ASEN are requested 1 <sub>B</sub> When the core supply voltage is equal or above the Level Threshold 2 voltage LEV2V the action configured by bits L2INTEN, L2RSTEN, and L2ASEN are requested
<b>L2INTEN</b>	13	rw	<b>Level Threshold 2 Interrupt Request Enable</b> This bit defines if an interrupt request trigger is requested if the comparison check was successful. When a check is successful is defined via bit L2ALEV. 0 <sub>B</sub> No interrupt is requested 1 <sub>B</sub> An interrupt is requested

**System Control Unit (SCU)**

Field	Bits	Type	Description
<b>L2RSTEN</b>	14	rw	<b>Level Threshold 2 Reset Request Enable</b> This bit defines if a reset request trigger is requested if the comparison check was successful. When a check is successful is defined via bit L2ALEV. 0 <sub>B</sub> No reset is requested 1 <sub>B</sub> An reset is requested
<b>L2ASEN</b>	15	rw	<b>Level Threshold 2 Asynchronous Action Enable</b> This bit defines if asynchronous action can be performed if the comparison check was successful. When a check is successful is defined via bit L2ALEV. 0 <sub>B</sub> No asynchronous actions are performed 1 <sub>B</sub> Asynchronous actions can be performed

### **8.6.3 Controlling the Voltage Level of a Core Domain**

The core power can be controlled within certain limits. The voltage level is controlled by two **Embedded Voltage Regulators** (EVR).

The power domain is controlled by both EVR\_M and EVR\_1.

### **8.6.3.1 Embedded Voltage Regulator**

The main part of the device logic operates at a typical voltage level of 1.5 V. This supply voltage is generated by the Embedded Power Regulators (EVRs) out of the pad voltage. External buffer caps are required for stable regulation.

#### **Feature list:**

- Multiple core voltage levels including zero
- Core voltage generation based on a High Precision Bandgap
- External supply possible via capacitor-pin while EVR is switched-off
- Core current limit

The EVR configurations to select the desired voltage and reference pair are combined within EVR settings EVRxSETyyV (x = M or 1 and yy = 15). Each setting contains a bit field (VRSEL) to select the voltage level and reference and a bit field to fine-tune the voltage level (VLEV). One out of the possible settings is used to control each of the EVRs, but only in the allowed combinations for the two EVRs. The EVRs use a High Precision Bandgap (HP) as reference.

The BG voltage of each setting can be adjusted to compensate application and environmental influences by the bit field EVRxSETyyV.VLEV. VLEV is set by default or trimmed by each device during production test to reach the default setting targets.

#### **High Precision Bandgap (HP)**

The HP bandgap of the system is used for following purposes:

- Provide a very stable reference for the two EVRs
- Provide an accurate reference for the flash memory. For more information see the flash memory description.

Only one HP bandgap is implemented which is used by both EVRs. The HP bandgap can be enabled / disabled via the bit **EVRCON1.HPEN**.

## EVR Status and Control Registers

### EVR1CON0

**EVR\_1 Control 0 Register**

**ESFR (F088<sub>H</sub>/44<sub>H</sub>)**

**Reset Value: DF20<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>EVR DIS</b>	-	<b>0</b>	<b>CC DIS</b>	<b>CCLEV</b>	<b>LPR DIS</b>	<b>1</b>	<b>0</b>	<b>LPRLEV</b>					<b>0</b>		
rh	-	rw	rh	rw	rh	rw	rw				rw			r	

Field	Bits	Type	Description
<b>LPRLEV</b>	[5:3]	rw	<b>Reserved</b> Do not change this value when writing to this register.
<b>0</b>	[7:6]	rw	<b>Reserved</b> Do not change this value when writing to this register
<b>1</b>	8	rw	<b>Reserved</b> Do not change this value when writing to this register
<b>LPRDIS</b>	9	rh	<b>Reserved</b> Value is undefined
<b>CCLEV</b>	[11:10]	rw	<b>Current Control Level</b> This bit field is required for enabling/disabling the current control (CCDIS). Valid values are described in the Programmer's Guide.
<b>CCDIS</b>	12	rh	<b>Current Control Disable</b> 0 <sub>B</sub> The current control is enabled 1 <sub>B</sub> The current control is disabled This bit is updated by bit EVR1SETy.CCDIS.
<b>0</b>	13	rw	<b>Reserved</b> Do not change this value when writing to this register
<b>EVRDIS</b>	15	rh	<b>EVR_1 Disable</b> 0 <sub>B</sub> The EVR_1 is enabled 1 <sub>B</sub> The EVR_1 is disabled This bit is updated by bit EVR1SETy.EVRDIS.
<b>0</b>	[2:0]	r	<b>Reserved</b> Read as 0; should be written with 0.

**EVR1SET15VHP**

**EVR\_1 Setting for 1.5 V HP Register**

**ESFR (F09E<sub>H</sub>/4F<sub>H</sub>)**

**Reset Value: 001B<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>EVR DIS</b>	<b>0</b>	<b>CC DIS</b>	<b>0</b>	<b>LPR DIS</b>	<b>0</b>	<b>VRSEL</b>		<b>VLEV</b>							
rw	rw	rw	rw	rw	rw	rw	rw	rw							

Field	Bits	Type	Description
<b>VLEV</b>	[5:0]	rw	<b>Voltage Level Adjust</b> This bit field adjusts the BG voltage and is trimmed by each device during production test to reach the default setting targets. Do not change this value when writing to this register.
<b>VRSEL</b>	[7:6]	rw	<b>Voltage Reference Selection</b> 00 <sub>B</sub> 15VHP - Full Voltage with high precision bandgap selected 01 <sub>B</sub> Reserved, do not use this combination 10 <sub>B</sub> Reserved, do not use this combination 11 <sub>B</sub> Reserved, do not use this combination <i>Note: The reset value should always be written to this bit field.</i>
<b>LPRDIS</b>	9	rw	<b>Reserved</b> Do not change this value when writing to this register
<b>CCDIS</b>	12	rw	<b>Current Control Disable</b> 0 <sub>B</sub> The current control is enabled 1 <sub>B</sub> The current control is disabled This bit updates bit EVR1CON0.CCDIS. <i>Note: Before switching off the current control the CCLEV setting in EVR1CON0 has to be set to 00<sub>B</sub>.</i>
<b>EVRDIS</b>	15	rw	<b>EVR_1 Disable</b> 0 <sub>B</sub> The EVR_1 is enabled 1 <sub>B</sub> The EVR_1 is disabled This bit updates bit EVR1CON0.EVRDIS.

Field	Bits	Type	Description
<b>0</b>	8, [11:10], [14:13]	rw	<b>Reserved</b> Should be written with 0.



**EVRMCON0**

**EVR\_M Control 0 Register**

**ESFR (F084<sub>H</sub>/42<sub>H</sub>)**

**Reset Value: 0D20<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>EVR DIS</b>	<b>0</b>	<b>CC DIS</b>	<b>CCLEV</b>	<b>LPR DIS</b>	<b>1</b>	<b>0</b>	<b>LPRLEV</b>							<b>0</b>	
rh	r	rh	rw	rh	rw	rw	rw							r	

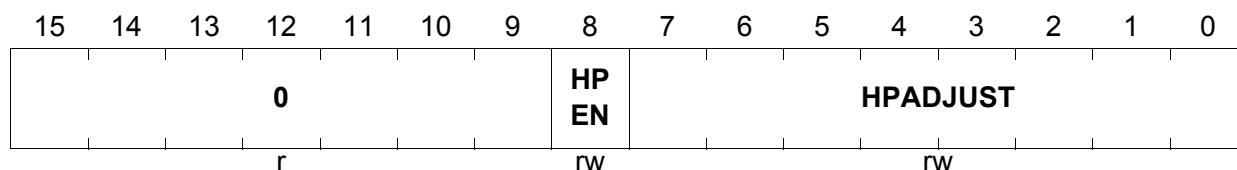
Field	Bits	Type	Description
<b>LPRLEV</b>	[5:3]	rw	<b>Reserved</b> Do not change this value when writing to this register
<b>0</b>	[7:6]	rw	<b>Reserved</b> Do not change this value when writing to this register
<b>1</b>	8	rw	<b>Reserved</b> Do not change this value when writing to this register
<b>LPRDIS</b>	9	rh	<b>Low Power Reference Disable</b> 0 <sub>B</sub> The LPR is enabled 1 <sub>B</sub> The LPR is disabled This bit is updated by bit EVRMSETy.LPRDIS.
<b>CCLEV</b>	[11:10]	rw	<b>Current Control Level</b> This bit field is required for enabling/disabling the current control (CCDIS). Valid values are described in the Programmer's Guide.
<b>CCDIS</b>	12	rh	<b>Current Control Disable</b> 0 <sub>B</sub> The current control is enabled 1 <sub>B</sub> The current control is disabled This bit is updated by bit EVRMSETy.CCDIS.
<b>EVRDIS</b>	15	rh	<b>EVR_M Disable</b> 0 <sub>B</sub> The EVR_M is enabled 1 <sub>B</sub> The EVR_M is disabled This bit is updated by bit EVRMSETy.EVRDIS.
<b>0</b>	[2:0], [14:13]	r	<b>Reserved</b> Read as 0; should be written with 0.

**EVRMCON1**

**EVR\_M Control 1 Register**

**ESFR (F086<sub>H</sub>/43<sub>H</sub>)**

**Reset Value: 0101<sub>H</sub>**



Field	Bits	Type	Description
HPADJUST	[7:0]	rw	<b>HP Bandgap Adjustment</b> This bit field is a device specific trimmvalue for the HP bandgap. Do not change this value when writing to this register.
HPEN	8	rw	<b>HP Bandgap Enable</b> 0 <sub>B</sub> The HP bandgap is disabled 1 <sub>B</sub> The HP bandgap is enabled
0	[15:9]	r	<b>Reserved</b> Read as 0; should be written with 0.

**EVRMSET15VHP**

**EVR\_M Setting for 1.5 V HP Register**

**ESFR (F096<sub>H</sub>/4B<sub>H</sub>)**

**Reset Value: 001B<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>EVR DIS</b>	<b>0</b>	<b>CC DIS</b>	<b>0</b>		<b>LPR DIS</b>	<b>0</b>	<b>VRSEL</b>				<b>VLEV</b>				
rw	rw	rw	rw		rw	rw	rw				rw				

Field	Bits	Type	Description
<b>VLEV</b>	[5:0]	rw	<b>Voltage Level Adjust</b> This bit field adjusts the BG voltage and is trimmed by each device during production test to reach the default setting targets. Do not change this value when writing to this register.
<b>VRSEL</b>	[7:6]	rw	<b>Voltage Reference Selection</b> 00 <sub>B</sub> 15VHP - Full Voltage with high precision bandgap selected 01 <sub>B</sub> Reserved, do not use this combination 10 <sub>B</sub> Reserved, do not use this combination 11 <sub>B</sub> Reserved, do not use this combination <i>Note: The reset value should always be written to this bit field.</i>
<b>LPRDIS</b>	9	rw	<b>Reserved</b> Do not change this value when writing to this register
<b>CCDIS</b>	12	rw	<b>Current Control Disable</b> 0 <sub>B</sub> The current control is enabled 1 <sub>B</sub> The current control is disabled This bit updates bit EVRMCON0.CCDIS. <i>Note: Before switching off the current control the CCLEV setting in EVRMCON0 has to be set to 00<sub>B</sub>.</i>
<b>0</b>	14	rw	<b>Reserved</b> Do not change this value when writing to this register
<b>EVRDIS</b>	15	rw	<b>EVR_M Disable</b> 0 <sub>B</sub> The EVR_M is enabled 1 <sub>B</sub> The EVR_M is disabled This bit updates bit EVR1CON0.EVRDIS.

**System Control Unit (SCU)**

Field	Bits	Type	Description
<b>0</b>	8, [11:10], 13	rw	<b>Reserved</b> Should be written with 0.

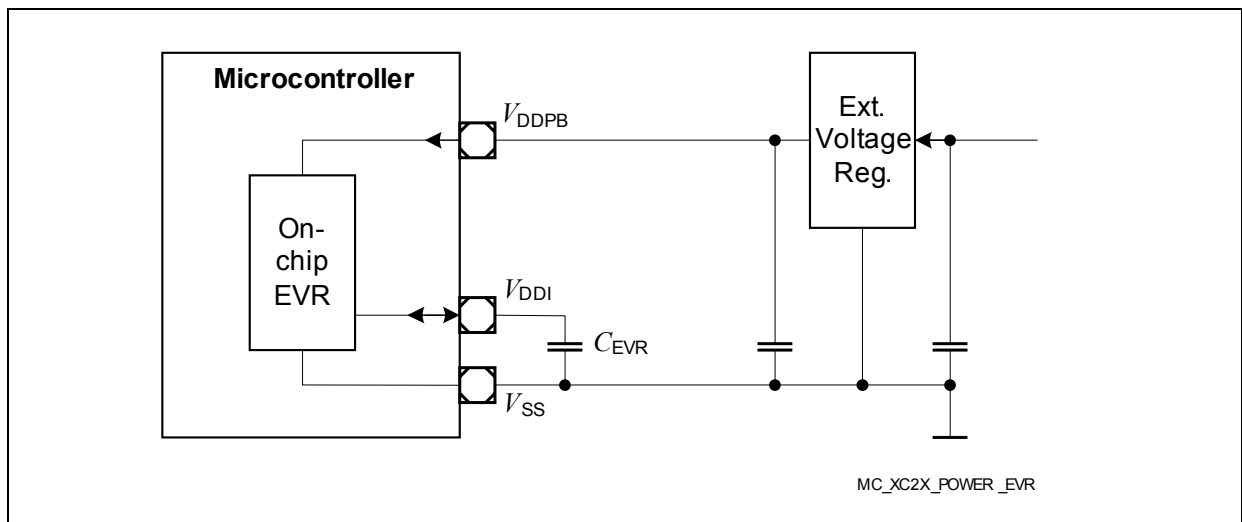
### 8.6.3.2 Sources for Core Supply Voltage

The on-chip EVRs can generate the XE16xyM's core supply voltage from the (externally supplied) IO voltage.

#### Core Supply via On-chip EVRs

Generating the core supply voltage via the integrated EVRs is the preferred operating mode, because it saves an additional external voltage regulator. The integrated EVRs are fed from supply voltage  $V_{DDPB}$ .

Proper operation of the EVRs requires external buffer capacitances. Please refer to the respective Data Sheet for the recommended values. The current is delivered by the integrated pass devices.



**Figure 8-23 Selecting the EVR for Core Supply**

Generating the core supply voltage with on-chip resources provides full control of power reduction modes, so the application can control and minimize the energy consumption of the XE16xyM using built-in mechanisms without requiring additional external circuitry.

#### **8.6.4 Handling the Power System**

Using the power system correctly is the key to power saving. Depending on the application different operating states can be defined in order to save maximal power. The XE16xyM supports following power saving mechanisms:

- Reduction of the system performance
  - the power consumption depends directly on the frequency of the system
  - the system performance is controlled with the clock operation mechanism
- Stopping single unused peripheral
  - a peripheral not needed for an application can be disabled
  - the module operation is controlled via register MOD\_KSCCFG
- Stopping multiple unused peripherals
  - peripherals not needed for an application can be disabled
  - system peripheral operation is controlled via the Global State Controller (GSC)
- Stopping single unused analog parts
  - an analog part not needed for an application can be stopped
  - the operation is controlled via register either located in the SCU (PLL, clocks, PVCs, SWD, Temperature Compensation) or the ADC

## **8.7 Global State Controller (GSC)**

Mode Control for the system peripherals provides besides power saving modes and the clock management an additional opportunity for configuring the system to the application needs.

Mode Control is described in detail in this chapter and is implemented by the Global State Controller (GSC). The GSC enables the user to configure one operating mode in a fast and easy way, reacting fast and explicit to needs of an application.

### **Feature Overview**

The following issues are handled by the GSC:

- Control of peripheral clock operation
- Suspend control for debugging
- Arbitration between the different request sources

According to the requests coming from the OCDS, the SWD pre-warning detection or other blocks, the GSC does an internal prioritization. The result is forwarded as command request broadcast to all peripherals. The GSC internal prioritization scheme for the implemented request sources is shown in [Table 8-8](#).

### **8.7.1 GSC Control Flow**

The sequence begins when at least one request source asserts its trigger in order to request a mode change in the SoC. If several requests are pending there is an arbitration mechanism that treats this issue. Request triggers are not stored by the GSC, therefore a trigger source has to assert its trigger until the trigger is no longer valid or needed.

A request trigger is kept asserted as long as either the request is still pending or the resulting command of the request was entered and acknowledged by the system. The communication of the GSC and the peripherals is based on commands. Three different commands are defined resulting in three modes:

- Wake-up command: requests Normal Mode
- Clock-off command: requests Stop Mode
- Debug command: requests Suspend Mode

The specific behavior in these three modes is defined for each peripheral in its module register `mod_KSCCFG`.

#### **8.7.1.1 Request Source Arbitration**

The highest priority for the arbitration is zero (see [Table 8-8](#)).

Each system clock cycle a new arbitration round is started. The winner of an arbitration round requests the next command towards the SoC. Please note that winning an arbitration does not lead automatically to a new command raised. Only if currently no command is broadcast in the SoC a new command can be generated and broadcast. If

the winner of the arbitration round is the same request trigger as in the previous round or if no winner was detected no new command request is generated.

**Table 8-8 Connection of the Request Sources**

<b>Request Source</b>	<b>Priority</b>
OCDS exit	4
ESR0	5
ESR1	6
ESR2	7
WUT	8
ITC	9
GPT12E	10
SW1	11
SW2	12
OCDS entry	14

### **8.7.1.2 Generation of a New Command**

When a new request trigger was detected and arbitrated a new command request is generated if currently no command request is broadcast that is not received by all slaves.

**Table 8-9 Request Source and Command Request Coupling**

<b>Request Source</b>	<b>Command Description</b>
OCDS exit	Wake-up; Normal Mode
ESR0	Wake-up; Normal Mode
ESR1	Wake-up; Normal Mode
ESR2	Clock-off; Stop Mode
WUT	Wake-up; Normal Mode
ITC	Wake-up; Normal Mode
GPT12E	Wake-up; Normal Mode
SW1	Wake-up; Normal Mode
SW2	Clock-off; Stop Mode
OCDS entry	Debug; Suspend Mode



### **8.7.1.3 Usage of Commands**

The complete control mechanism for the different operation modes of the various slaves is divided into two parts:

- A central control and configuration part; the Global State Controller (GSC)
- One local control part in each slave; the Kernel State Controller (KSC)

Via the GSC either different hardware sources (e.g. the WUT or the OCDS) or the software can request the system to enter a specific mode. The parts that are affected by the mode can be pre-defined locally for each part via the KSC. For each command a specific reaction can be pre-configured in each KSC for each individual part.

*Note: Requesting a peripheral to be permanently shut off by clearing `mod_KSCCFG.MODEN` to 0 does not start a GSC run. However, a GSC run triggered in parallel to the ramp down of this peripheral will have the finite state machine of the GSC waiting for an acknowledge also of this peripheral as long as the peripheral does not deliver its acknowledge or the respective bit in `GSCPERSTATEN` is cleared.*

*The proposal is either to disable automatic GSC runs (by setting `GSCEN` respectively) in case the application needs the information of the shutdown acknowledge of the peripheral or to disable the respective bit in `GSCPERSTATEN`, so that the missing acknowledge is not taken into account.*

*Note: When a GSC mode request has been successfully entered, the GSC arbiter is open for any new request. In case a request occurs to enter the current mode, this request is pipelined and remains pending.*

*It is recommended to request a command by a software trigger. In particular the clock-off command should be triggered by SW2. The usage of commands requested by hardware has to be done carefully. Only hardware resources requesting Normal Mode should be selected. If the software has detected a wake-up then pending mode change requests can be removed by clearing the bits of the selected sources in **GSCEN** and then enabling the bits in `GSCEN` again.*

### **8.7.1.4 Terminating a Request Trigger**

A request trigger is no longer taken into account for the arbitration after the de-assertion of the request trigger, if it is not enabled or when its respective enable bit is cleared.

### **8.7.1.5 Suspend Control Flow**

The suspend feature is controlled by the OCDS block. The GSC operates only as control and communication interface towards the system. The suspend feature is composed out of two requirements:

The mode that has to be entered when the Suspend Mode is requested.

The mode that has to be entered when the Suspend Mode is left.

**System Control Unit (SCU)**

The request to enter Suspend Mode is forwarded from the OCDS. When the Suspend Mode is requested the system is expected to be stopped as soon as possible in an idle state where no internal process is pending and in a way that this system state does not lead to any damage internally or externally and can also be left without any damage. Therefore all peripherals in the system are requested to enter a mode where the clock can be stopped. This is done by sending a debug command.

Leaving the Suspend Mode should serve the goal that debugging is a non-intrusive operation. Therefore leaving the Suspend Mode can not lead to only one dedicated system mode, instead it leads to the system mode the system left when it was requested to exit the Suspend Mode. The system mode is stored when a Suspend Mode request is detected by the GSC and is used as target system mode when a leave Suspend Mode trigger is detected by the GSC.

**8.7.1.6 Error Feedback for a Mode Transition**

In case at least one peripheral reports an error the error flag in register GSCSTAT is set. If no error is currently detected upon a new assertion of a system mode by the GSC the error flag is cleared. To inform the system of this erroneous state an interrupt can be generated.

## 8.7.2 GSC Registers

### 8.7.2.1 GSC Control and Status Registers

The following register control and configure the behavior of the GSC.

#### **GSCSWREQ**

#### **GSC Software Request Register**

**SFR (FF14<sub>H</sub>/8A<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0														<b>SW TRG 2</b>	<b>SW TRG 1</b>
r														rwh	rwh

Field	Bits	Type	Description
<b>SWTRG1</b>	0	rwh	<b>Software Trigger 1 (SW1)</b> 0 <sub>B</sub> No SW1 request trigger is generated 1 <sub>B</sub> A SW1 request trigger is generated This bit is automatically cleared if the SW1 request trigger wins the arbitration and was broadcast to the system.
<b>SWTRG2</b>	1	rwh	<b>Software Trigger 2 (SW2)</b> 0 <sub>B</sub> No SW2 request trigger is generated 1 <sub>B</sub> A SW2 request trigger is generated This bit is automatically cleared if the SW2 request trigger wins the arbitration and was broadcast to the system.
<b>0</b>	[15:2]	r	<b>Reserved</b> Read as 0; should be written with 0.

**GSCEN**

**GSC Enable Register**

**SFR (FF16<sub>H</sub>/8B<sub>H</sub>)**

**Reset Value: 7FFF<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	OCD SEN EN	1	SW2 EN	SW1 EN	GPT EN	ITC EN	WUT EN	ESR 2 EN	ESR 1 EN	ESR 0 EN	OCD SEX EN	PSC AEN EN	PSC AEX EN	PSC BEN EN	PSC BEX EN
r	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
PSCBEXEN	0	rw	<b>Reserved</b> Must be written with 0 <sub>B</sub> .
PSCBENEN	1	rw	<b>Reserved</b> Must be written with 0 <sub>B</sub> .
PSCAEXEN	2	rw	<b>Reserved</b> Must be written with 0 <sub>B</sub> .
PSCAENEN	3	rw	<b>Reserved</b> Must be written with 0 <sub>B</sub> .
OCDSEXEN	4	rw	<b>OCDS Exit Request Trigger Enable</b> 0 <sub>B</sub> OCDS exit request trigger is not taken into account (disabled) 1 <sub>B</sub> OCDS exit request trigger is taken into account (enabled)
ESR0EN	5	rw	<b>ESR0 Request Trigger Enable</b> 0 <sub>B</sub> $\overline{\text{ESR0}}$ request trigger is not taken into account (disabled) 1 <sub>B</sub> $\overline{\text{ESR0}}$ request trigger is taken into account (enabled)
ESR1EN	6	rw	<b>ESR1 Request Trigger Enable</b> 0 <sub>B</sub> $\overline{\text{ESR1}}$ request trigger is not taken into account (disabled) 1 <sub>B</sub> $\overline{\text{ESR1}}$ request trigger is taken into account (enabled)
ESR2EN	7	rw	<b>ESR2 Request Trigger Enable</b> 0 <sub>B</sub> $\overline{\text{ESR2}}$ request trigger is not taken into account (disabled) 1 <sub>B</sub> $\overline{\text{ESR2}}$ request trigger is taken into account (enabled)

Field	Bits	Type	Description
<b>WUTEN</b>	8	rw	<b>WUT Request Trigger Enable</b> $0_B$ WUT request trigger is not taken into account (disabled) $1_B$ WUT request trigger is taken into account (enabled)
<b>ITCEN</b>	9	rw	<b>ITC Request Trigger Enable</b> $0_B$ ITC request trigger is not taken into account (disabled) $1_B$ ITC request trigger is taken into account (enabled)
<b>GPTEN</b>	10	rw	<b>GTP12E Request Trigger Enable</b> $0_B$ GPT12E request trigger is not taken into account (disabled) $1_B$ GPT12E request trigger is taken into account (enabled)
<b>SW1EN</b>	11	rw	<b>Software 1 Request Trigger Enable</b> $0_B$ SW1 request trigger is not taken into account (disabled) $1_B$ SW1 request trigger is taken into account (enabled)
<b>SW2EN</b>	12	rw	<b>Software 2 Request Trigger Enable</b> $0_B$ SW2 request trigger is not taken into account (disabled) $1_B$ SW2 request trigger is taken into account (enabled)
<b>1</b>	13	rw	<b>Reserved</b> Read as 1; should be written with 1.
<b>OCDS ENEN</b>	14	rw	<b>OCDS Entry Request Trigger Enable</b> $0_B$ OCDS entry request trigger is not taken into account (disabled) $1_B$ OCDS entry request trigger is taken into account (enabled) OCDS entry is the request source belonging to the according connector interface.
<b>0</b>	15	r	<b>Reserved</b> Read as 0; should be written with 0.

**GSCSTAT**

**GSC Status Register**

**SFR (FF18<sub>H</sub>/8C<sub>H</sub>)**

**Reset Value: 3C00<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		SOURCE				PEN	ERR	0		NEXT		0		CURRENT	
r		rh				rh	rh	r		rh		r		rh	

Field	Bits	Type	Description
<b>CURRENT</b>	[1:0]	rh	<b>Currently used Command</b> This bit field states the currently used system mode.
<b>NEXT</b>	[5:4]	rh	<b>Next to use Command</b> This bit field states the next to be used system mode.
<b>ERR</b>	8	rh	<b>Error Status Flag</b> This bit flags if with the last command that was broadcast was acknowledge with at least one error. This bit is automatically cleared when a new command is broadcast.
<b>PEN</b>	9	rh	<b>Command Pending Flag</b> This flag states if currently a command is pending or not. A command is pending after the broadcast as long as no all blocks acknowledge that they finished the operation requested by the command.

Field	Bits	Type	Description
<b>SOURCE</b>	[13:10]	rh	<b>Requesting Source Status</b> This bit field monitors the source that triggered the last request. 0000 <sub>B</sub> Reserved 0001 <sub>B</sub> Reserved 0010 <sub>B</sub> Reserved 0011 <sub>B</sub> Reserved 0100 <sub>B</sub> OCDS exit 0101 <sub>B</sub> <u>ESR0</u> 0110 <sub>B</sub> <u>ESR1</u> 0111 <sub>B</sub> <u>ESR2</u> 1000 <sub>B</sub> WUT 1001 <sub>B</sub> ITC 1010 <sub>B</sub> GPT12E 1011 <sub>B</sub> SW1 1100 <sub>B</sub> SW2 1101 <sub>B</sub> Reserved, do not use this combination 1110 <sub>B</sub> OCDS entry 1111 <sub>B</sub> Reserved, do not use this combination
<b>0</b>	[3:2], [7:6], [15:14]	r	<b>Reserved</b> Read as 0; should be written with 0.

## GSCPERSTATEN

### GSC Peripheral Status Enable Register

**SFR (FF04<sub>H</sub>/82<sub>H</sub>)**

**Reset Value: FFFF<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>USIC</b> <b>3</b>	<b>USIC</b> <b>2</b>	<b>USIC</b> <b>1</b>	<b>USIC</b> <b>0</b>	<b>FL</b>	<b>MEM</b>	<b>RTC</b>	<b>CCU</b> <b>63</b>	<b>CCU</b> <b>62</b>	<b>CCU</b> <b>61</b>	<b>CCU</b> <b>60</b>	<b>M</b> <b>CAN</b>	<b>CC2</b>	<b>1</b>	<b>GPT</b> <b>12E</b>	<b>ADC</b>
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
<b>ADC</b>	0	rw	<b>ADC Acknowledge Enable</b> This bit defines if the acknowledge status of ADC modules is taken into account and displayed or ignored. 0 <sub>B</sub> The ADC modules acknowledge is ignored, it is treated as always asserted 1 <sub>B</sub> The ADC modules acknowledge is used
<b>GPT12E</b>	1	rw	<b>GPT12E Acknowledge Enable</b> This bit defines if the acknowledge status of GPT12E module is taken into account and displayed or ignored. 0 <sub>B</sub> The GPT12E module acknowledge is ignored, it is treated as always asserted 1 <sub>B</sub> The GPT12E module acknowledge is used
<b>1</b>	2	rw	<b>Reserved</b> Read as 1; should be written with 1 <sub>B</sub> .
<b>CC2</b>	3	rw	<b>CC2 Acknowledge Enable</b> This bit defines if the acknowledge status of CC2 module is taken into account and displayed or ignored. 0 <sub>B</sub> The CC2 module acknowledge is ignored, it is treated as always asserted 1 <sub>B</sub> The CC2 module acknowledge is used



Field	Bits	Type	Description
<b>MCAN</b>	4	rw	<b>MultiCAN Acknowledge Enable</b> This bit defines if the acknowledge status of MultiCAN module is taken into account and displayed or ignored. 0 <sub>B</sub> The MultiCAN module acknowledge is ignored, it is treated as always asserted 1 <sub>B</sub> The MultiCAN module acknowledge is used
<b>CCU60</b>	5	rw	<b>CCU60 Acknowledge Enable</b> This bit defines if the acknowledge status of CCU60 module is taken into account and displayed or ignored. 0 <sub>B</sub> The CCU60 module acknowledge is ignored, it is treated as always asserted 1 <sub>B</sub> The CCU60 module acknowledge is used
<b>CCU61</b>	6	rw	<b>CCU61 Acknowledge Enable</b> This bit defines if the acknowledge status of CCU61 module is taken into account and displayed or ignored. 0 <sub>B</sub> The CCU61 module acknowledge is ignored, it is treated as always asserted 1 <sub>B</sub> The CCU61 module acknowledge is used
<b>CCU62</b>	7	rw	<b>CCU62 Acknowledge Enable</b> This bit defines if the acknowledge status of CCU62 module is taken into account and displayed or ignored. 0 <sub>B</sub> The CCU62 module acknowledge is ignored, it is treated as always asserted 1 <sub>B</sub> The CCU62 module acknowledge is used
<b>CCU63</b>	8	rw	<b>CCU63 Acknowledge Enable</b> This bit defines if the acknowledge status of CCU63 module is taken into account and displayed or ignored. 0 <sub>B</sub> The CCU63 module acknowledge is ignored, it is treated as always asserted 1 <sub>B</sub> The CCU63 module acknowledge is used

**System Control Unit (SCU)**

Field	Bits	Type	Description
<b>RTC</b>	9	rw	<b>RTC Acknowledge Enable</b> This bit defines if the acknowledge status of RTC module is taken into account and displayed or ignored. 0 <sub>B</sub> The RTC module acknowledge is ignored, it is treated as always asserted 1 <sub>B</sub> The RTC module acknowledge is used
<b>MEM</b>	10	rw	<b>C166SV2 Subsystem Acknowledge Enable</b> This bit defines if the acknowledge status of C166SV2 Subsystem module is taken into account and displayed or ignored. 0 <sub>B</sub> The C166SV2 Subsystem module acknowledge is ignored, it is treated as always asserted 1 <sub>B</sub> The C166SV2 Subsystem module acknowledge is used
<b>FL</b>	11	rw	<b>Flash Acknowledge Enable</b> This bit defines if the acknowledge status of Flash module is taken into account and displayed or ignored. 0 <sub>B</sub> The Flash module acknowledge is ignored, it is treated as always asserted 1 <sub>B</sub> The Flash module acknowledge is used
<b>USIC0</b>	12	rw	<b>USIC0 Acknowledge Enable</b> This bit defines if the acknowledge status of USIC0 module is taken into account and displayed or ignored. 0 <sub>B</sub> The USIC0 module acknowledge is ignored, it is treated as always asserted 1 <sub>B</sub> The USIC0 module acknowledge is used
<b>USIC1</b>	13	rw	<b>USIC1 Acknowledge Enable</b> This bit defines if the acknowledge status of USIC1 module is taken into account and displayed or ignored. 0 <sub>B</sub> The USIC1 module acknowledge is ignored, it is treated as always asserted 1 <sub>B</sub> The USIC1 module acknowledge is used

**System Control Unit (SCU)**

Field	Bits	Type	Description
<b>USIC2</b>	14	rw	<b>USIC2 Acknowledge Enable</b> This bit defines if the acknowledge status of USIC2 module is taken into account and displayed or ignored. 0 <sub>B</sub> The USIC2 module acknowledge is ignored, it is treated as always asserted 1 <sub>B</sub> The USIC2 module acknowledge is used
<b>USIC3</b>	15	rw	<b>USIC3 Acknowledge Enable</b> This bit defines if the acknowledge status of USIC3 module is taken into account and displayed or ignored. 0 <sub>B</sub> The USIC3 module acknowledge is ignored, it is treated as always asserted 1 <sub>B</sub> The USIC3 module acknowledge is used

## GSCPERSTAT

### GSC Peripheral Status Register

**SFR (FF1A<sub>H</sub>/8D<sub>H</sub>)**

**Reset Value: FFFF<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>USIC</b> <b>3</b>	<b>USIC</b> <b>2</b>	<b>USIC</b> <b>1</b>	<b>USIC</b> <b>0</b>	<b>FL</b>	<b>MEM</b>	<b>RTC</b>	<b>CCU</b> <b>63</b>	<b>CCU</b> <b>62</b>	<b>CCU</b> <b>61</b>	<b>CCU</b> <b>60</b>	<b>M</b> <b>CAN</b>	<b>CC2</b>	<b>-</b>	<b>GPT</b> <b>12E</b>	<b>ADC</b>
rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	-	rh	rh

Field	Bits	Type	Description
<b>ADC</b>	0	rh	<b>ADC Acknowledge Status</b> This bit shows the acknowledge status of the modules ADC. 0 <sub>B</sub> The modules ADC change currently their kernel state. Their acknowledge has not been received. 1 <sub>B</sub> Acknowledge of the modules ADC is taken into account and has been received or is not relevant.
<b>GPT12E</b>	1	rh	<b>GPT12E Acknowledge Status</b> This bit shows the acknowledge status of the modules GPT12E. 0 <sub>B</sub> The module GPT12E changes currently its kernel state. Its acknowledge has not been received. 1 <sub>B</sub> Acknowledge of the module GPT12E is taken into account and has been received or is not relevant.
<b>CC2</b>	3	rh	<b>CC2 Acknowledge Status</b> This bit shows the acknowledge status of the module CC2. 0 <sub>B</sub> The module CC2 changes currently its kernel state. Its acknowledge has not been received. 1 <sub>B</sub> Acknowledge of the module CC2 is taken into account and has been received or is not relevant.

Field	Bits	Type	Description
<b>MCAN</b>	4	rh	<b>MultiCAN Acknowledge Status</b> This bit shows the acknowledge status of the module MultiCAN. 0 <sub>B</sub> The module MultiCAN changes currently its kernel state. Its acknowledge has not been received. 1 <sub>B</sub> Acknowledge of the module MultiCAN is taken into account and has been received or is not relevant.
<b>CCU60</b>	5	rh	<b>CCU60 Acknowledge Status</b> This bit shows the acknowledge status of the module CCU60. 0 <sub>B</sub> The module CCU60 changes currently its kernel state. Its acknowledge has not been received. 1 <sub>B</sub> Acknowledge of the module CCU60 is taken into account and has been received or is not relevant.
<b>CCU61</b>	6	rh	<b>CCU61 Acknowledge Status</b> This bit shows the acknowledge status of the module CCU61. 0 <sub>B</sub> The module CCU61 changes currently its kernel state. Its acknowledge has not been received. 1 <sub>B</sub> Acknowledge of the module CCU61 is taken into account and has been received or is not relevant.
<b>CCU62</b>	7	rh	<b>CCU62 Acknowledge Status</b> This bit shows the acknowledge status of the module CCU62. 0 <sub>B</sub> The module CCU62 changes currently its kernel state. Its acknowledge has not been received. 1 <sub>B</sub> Acknowledge of the module CCU62 is taken into account and has been received or is not relevant.

<b>Field</b>	<b>Bits</b>	<b>Type</b>	<b>Description</b>
<b>CCU63</b>	8	rh	<b>CCU63 Acknowledge Status</b> This bit shows the acknowledge status of the module CCU63. 0 <sub>B</sub> The module CCU63 changes currently its kernel state. Its acknowledge has not been received. 1 <sub>B</sub> Acknowledge of the module CCU63 is taken into account and has been received or is not relevant.
<b>RTC</b>	9	rh	<b>RTC Acknowledge Status</b> This bit shows the acknowledge status of the module RTC. 0 <sub>B</sub> The module RTC changes currently its kernel state. Its acknowledge has not been received. 1 <sub>B</sub> Acknowledge of the module RTC is taken into account and has been received or is not relevant.
<b>MEM</b>	10	rh	<b>C166SV2 Subsystem Acknowledge Status</b> This bit shows the acknowledge status of the module C166SV2 Subsystem. 0 <sub>B</sub> The module C166SV2 Subsystem changes currently its kernel state. Its acknowledge has not been received. 1 <sub>B</sub> Acknowledge of the module C166SV2 Subsystem is taken into account and has been received or is not relevant.
<b>FL</b>	11	rh	<b>Flash Acknowledge Status</b> This bit shows the acknowledge status of the module Flash. 0 <sub>B</sub> The module Flash changes currently its kernel state. Its acknowledge has not been received. 1 <sub>B</sub> Acknowledge of the module Flash is taken into account and has been received or is not relevant.

Field	Bits	Type	Description
<b>USIC0</b>	12	rh	<b>USIC0 Acknowledge Status</b> This bit shows the acknowledge status of the module USIC0. 0 <sub>B</sub> The module USIC0 changes currently its kernel state. Its acknowledge has not been received. 1 <sub>B</sub> Acknowledge of the module USIC0 is taken into account and has been received or is not relevant.
<b>USIC1</b>	13	rh	<b>USIC1 Acknowledge Status</b> This bit shows the acknowledge status of the module USIC1. 0 <sub>B</sub> The module USIC1 changes currently its kernel state. Its acknowledge has not been received. 1 <sub>B</sub> Acknowledge of the module USIC1 is taken into account and has been received or is not relevant.
<b>USIC2</b>	14	rh	<b>USIC2 Acknowledge Status</b> This bit shows the acknowledge status of the module USIC2. 0 <sub>B</sub> The module USIC2 changes currently its kernel state. Its acknowledge has not been received. 1 <sub>B</sub> Acknowledge of the module USIC2 is taken into account and has been received or is not relevant.
<b>USIC3</b>	15	rh	<b>USIC3 Acknowledge Status</b> This bit shows the acknowledge status of the module USIC3. 0 <sub>B</sub> The module USIC3 changes currently its kernel state. Its acknowledge has not been received. 1 <sub>B</sub> Acknowledge of the module USIC3 is taken into account and has been received or is not relevant.

The acknowledge status bit is set, if acknowledge of the module x is taken into account and has been received or is not relevant. The acknowledge of the module x is not relevant if the acknowledge of the module x is not taken into account or the module x currently

**System Control Unit (SCU)**

does not undergo a change of kernel state mode. In these cases the acknowledge is assumed to be received.



## 8.8 Software Boot Support

In order to determine the correct starting point of operation for the software a minimum of hardware support is required. As much as possible is done via software. Some decisions have to be done in hardware because they must be known before any software is operational.

### 8.8.1 Start-up Registers

#### 8.8.1.1 Start-up Status Register

Register STSTAT contains the information required by the boot software to identify the different start-up settings that can be selected.

#### STSTAT

##### Start-up Status Register

**MEM (F1E0<sub>H</sub>/--)**

**Reset Value: 8000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>1</b>				<b>0</b>							<b>HWCFG</b>				
r				r							rh				

Field	Bits	Type	Description
<b>HWCFG</b>	[7:0]	rh	<b>Hardware Configuration Setting</b> This bit field contains the value that is used by the boot software. This bit field is updated in case of an Application Reset with the content by register SWRSTCON.SWCFG if bit SWRSTCON.SWBOOT is set.
<b>0</b>	[14:8]	r	<b>Reserved</b> Read as 0; should be written with 0.
<b>1</b>	15	r	<b>Reserved</b> Read as 1; should be written with 1.

## 8.9 External Request Unit (ERU)

The External Request Unit (ERU) is a versatile event and pattern detection unit. Its major task is the **generation of interrupts based on selectable trigger events at different inputs**, e.g. to generate external interrupt requests if an edge occurs at an input pin.

The detected events can also be used by other modules to trigger or to gate module-specific actions, such as conversions of the ADC module.

### 8.9.1 Introduction

The ERU of the XE16xyM can be split in three main functional parts:

- 4 independent **Input Channels x** for input selection and conditioning of trigger or gating functions
- Event distribution: A **Connecting Matrix** defines the events of the Input Channel x that lead to a reaction of an Output Channel y.
- 4 independent **Output Channels y** for combination of events, definition of their effects and distribution to the system (interrupt generation, ADC conversion triggers)

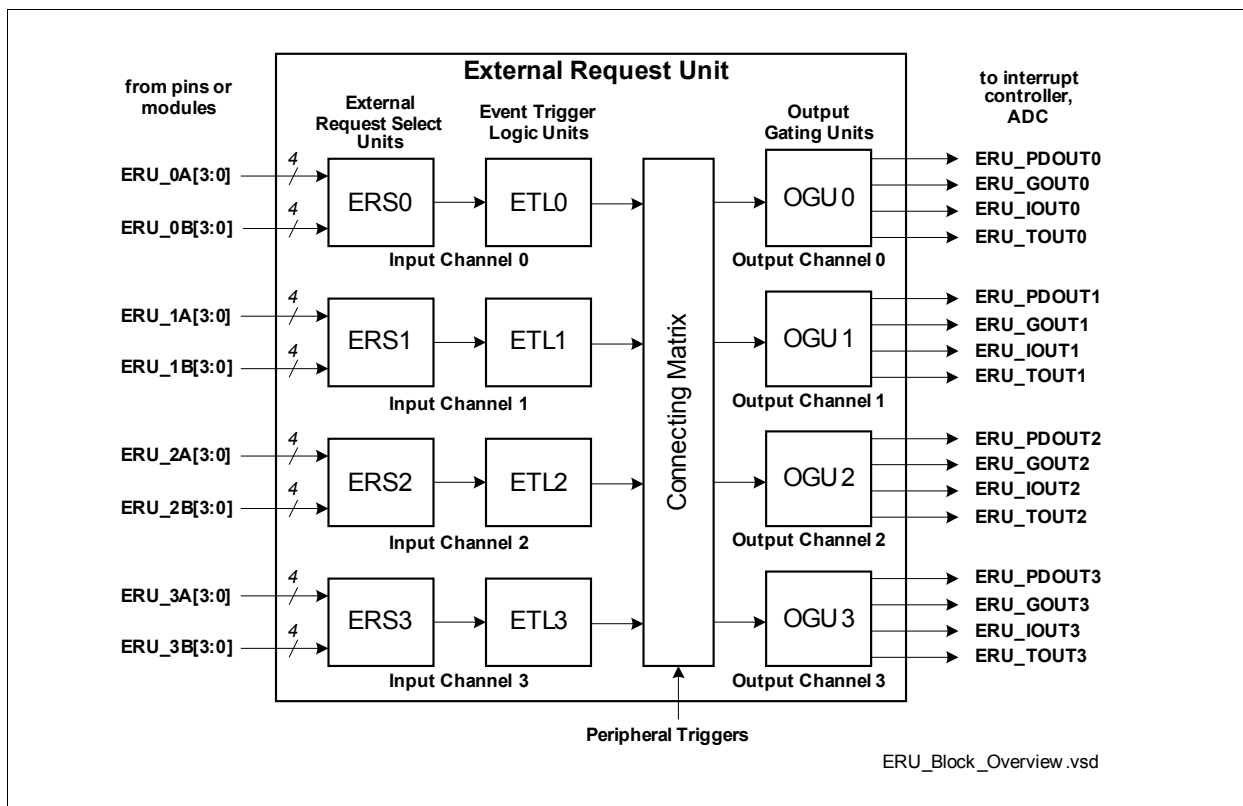


Figure 8-24 External Request Unit Overview

These tasks are handled by the following building blocks:

- An **External Request Select Unit (ERSx)** per Input Channel allows the selection of one out of two or a logical combination of two input signals (ERU\_xA, ERU\_xB) to a

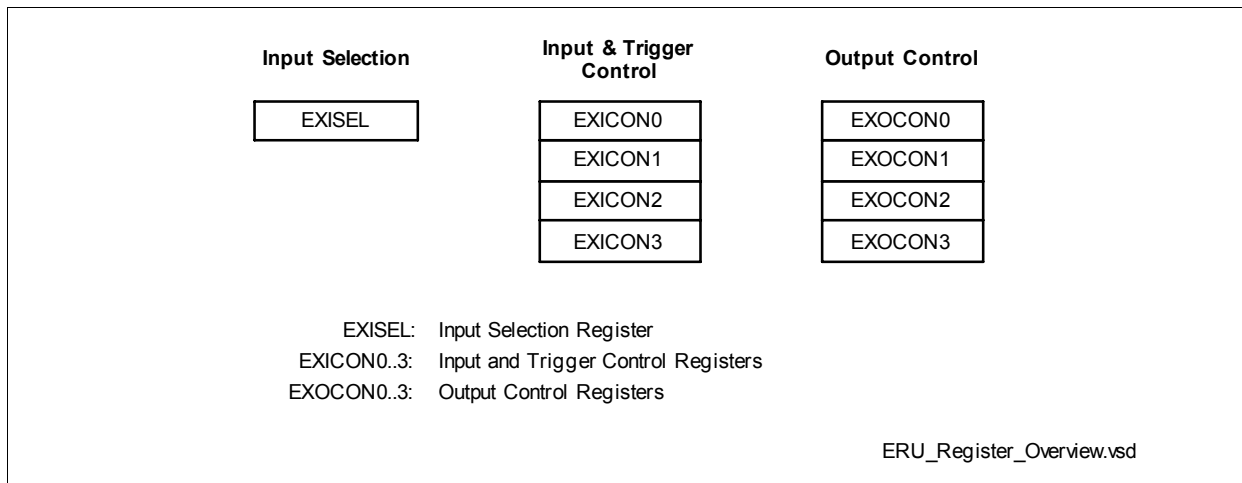
**System Control Unit (SCU)**

common trigger. For each of these two signals, an input vector of 4 possible inputs is available (e.g. the actual input ERU\_xA can be selected from one of the ERU inputs ERU\_xA[3:0], similar for ERU\_xB).

- An **Event Trigger Logic (ETLx)** 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 (event detected = event flag becomes set, independent of the polarity of the original input signals).
- The **Connecting Matrix** distributes the events and status flags generated by the Input Channels to the Output Channels. Additionally, some Peripheral Trigger signals from other modules (e.g. CC2) are made available and can be combined with the triggers generated by the Input Channels of the ERU.
- An **Output Gating Unit (OGUy)** per Output Channel that combines the available trigger events and status information from the Input Channels. An event of one Input Channel can lead to reactions of several Output Channels, or also events of several Input Channels can be combined to a reaction of one Output Channel (pattern detection).

Different types of reactions are possible, e.g. interrupt generation (based on signals ERU\_IOUTy), triggering of ADC conversions (based on signals ERU\_TOUTy), or gating of ADC conversions (based on signals ERU\_GOUTy).

The ERU is controlled by a number of registers, shown in **Figure 8-25**, and described in **Section 8.9.8**.



**Figure 8-25 ERU Registers Overview**

## **8.9.2 ERU Input Connections**

The inputs to the ERU can be selected from a large number of input signals. While some of the inputs come directly from a pin, other inputs use signals from various peripheral modules, such as the USIC (signals named with prefix UxCy to indicate the communication channel) and the MultiCAN modules. These signals are input signals from the pin that has been selected as input for a USIC or MultiCAN function. The selection of the input is made within the respective USIC or MultiCAN module.

In the ERU input connections are described in [Section 8.17.3.1](#).

Usually, such signals would be selected for an ERU function when the input function to the USIC or MultiCAN module is not used otherwise, or the module is not used at all. However, it is also possible to select a input which is actually needed in a USIC or MultiCAN module, and to use it also in the ERU to provide a certain trigger functions, eventually combined with other signals (e.g. to generate an interrupt trigger in case a start of frame is detected at a selected communication).

With this structure, the number of possible input pins is significantly increased, because not only the selection capability of the ERU is used, but also the selection capability of the communication modules.

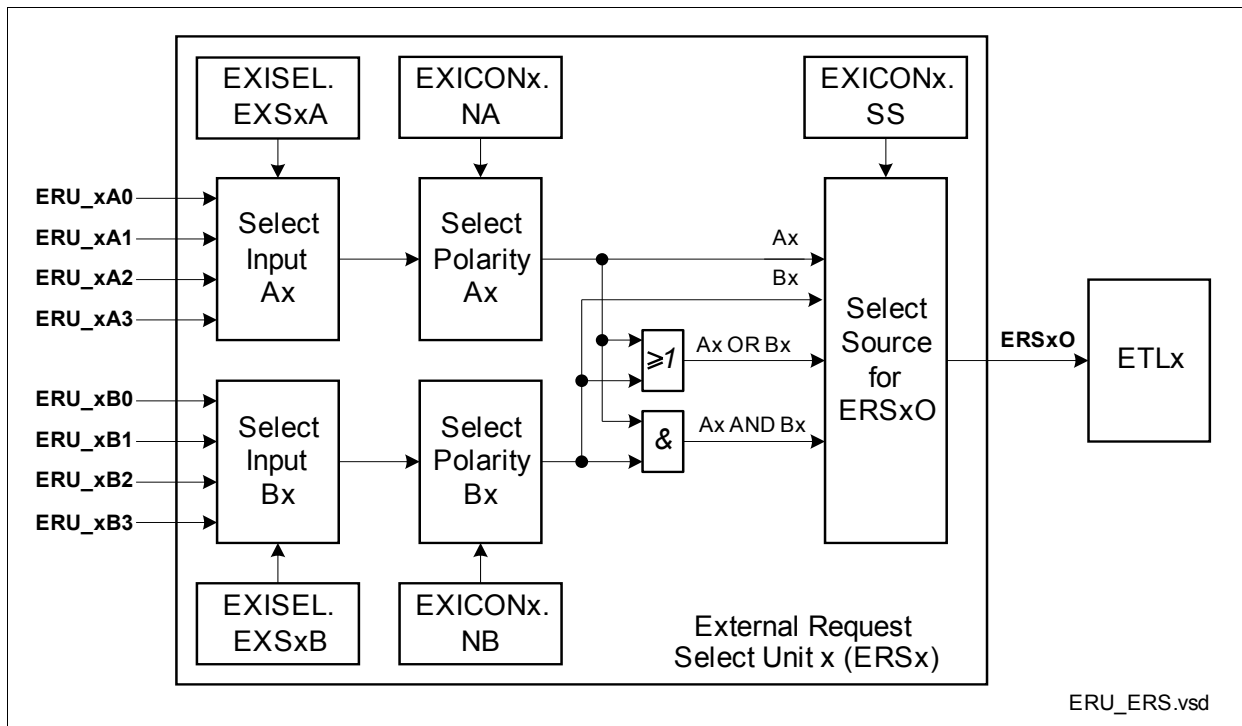
*Note: All functional inputs of the ERU are synchronized to  $f_{SYS}$  before they can affect the internal logic. The resulting delay of  $2/f_{SYS}$  and an uncertainty of  $1/f_{SYS}$  have to be taken into account for precise timing calculation.*

*An edge of an input can only be correctly detected if both, the high phase and the low phase of the input are each longer than  $1/f_{SYS}$ .*

### 8.9.3 External Request Select Unit (ERSx)

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 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.



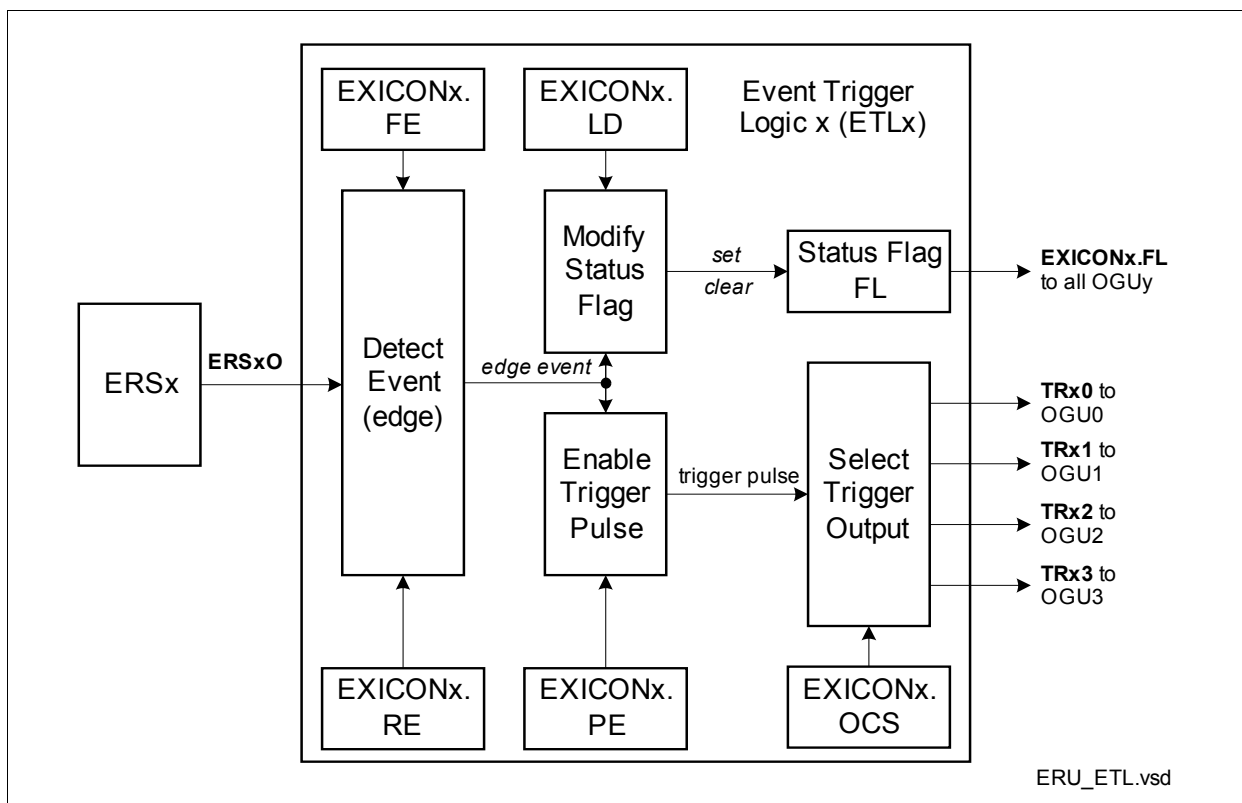
**Figure 8-26 External Request Select Unit Overview**

The ERS units are controlled via register **EXISEL** (one register for all four ERSx units) and registers EXICONx (one register for each ERSx and associated ETLx unit, e.g. **EXICON0** for Input Channel 0).

### 8.9.4 Event Trigger Logic (ETLx)

For each Input Channel  $x$  ( $x = 0-3$ ), an event trigger logic ETL $x$  derives a trigger event and a status from the input ERU $x$ O delivered by the associated ERS $x$  unit. Each ETL $x$  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 ETL $x$  units has an associated EXICON $x$  register, that controls all options of an ETL $x$  (the register also holds control bits for the associated ERS $x$  unit, e.g. **EXICON0** to control ESR0 and ETL0).



**Figure 8-27 Event Trigger Logic Overview**

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

It can be used as “sticky” flag, that 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

**System Control Unit (SCU)**

detection where the actual status of the input is important (enabling both edge detections is not useful in this mode).

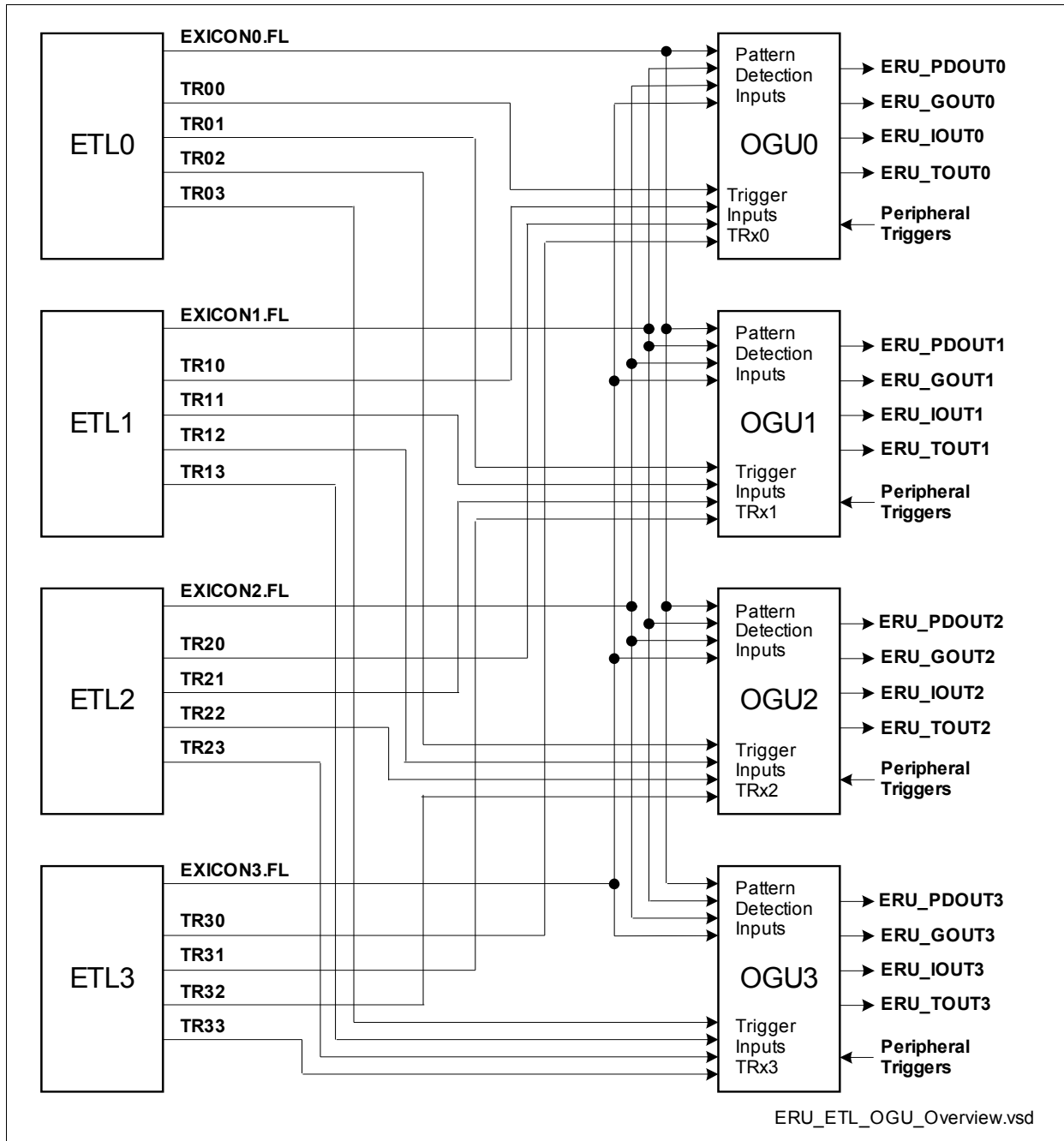
The output of the status flag is connected to all following Output Gating Units (OGUy) in parallel (see [Figure 8-28](#)) to provide **pattern detection capability of all OGUy** units based on different or the same status flags.

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.

### 8.9.5 Connecting Matrix

The connecting matrix distributes the trigger signals (TRxy) and status signals (EXICONx.FL) from the different ETLx units between the OGUy units. In addition, it receives peripheral trigger signals that can be OR-combined with the ETLx trigger signals in the OGUy units. **Figure 8-28** provides a complete overview of the connections between the ETLx and the OGUy units.



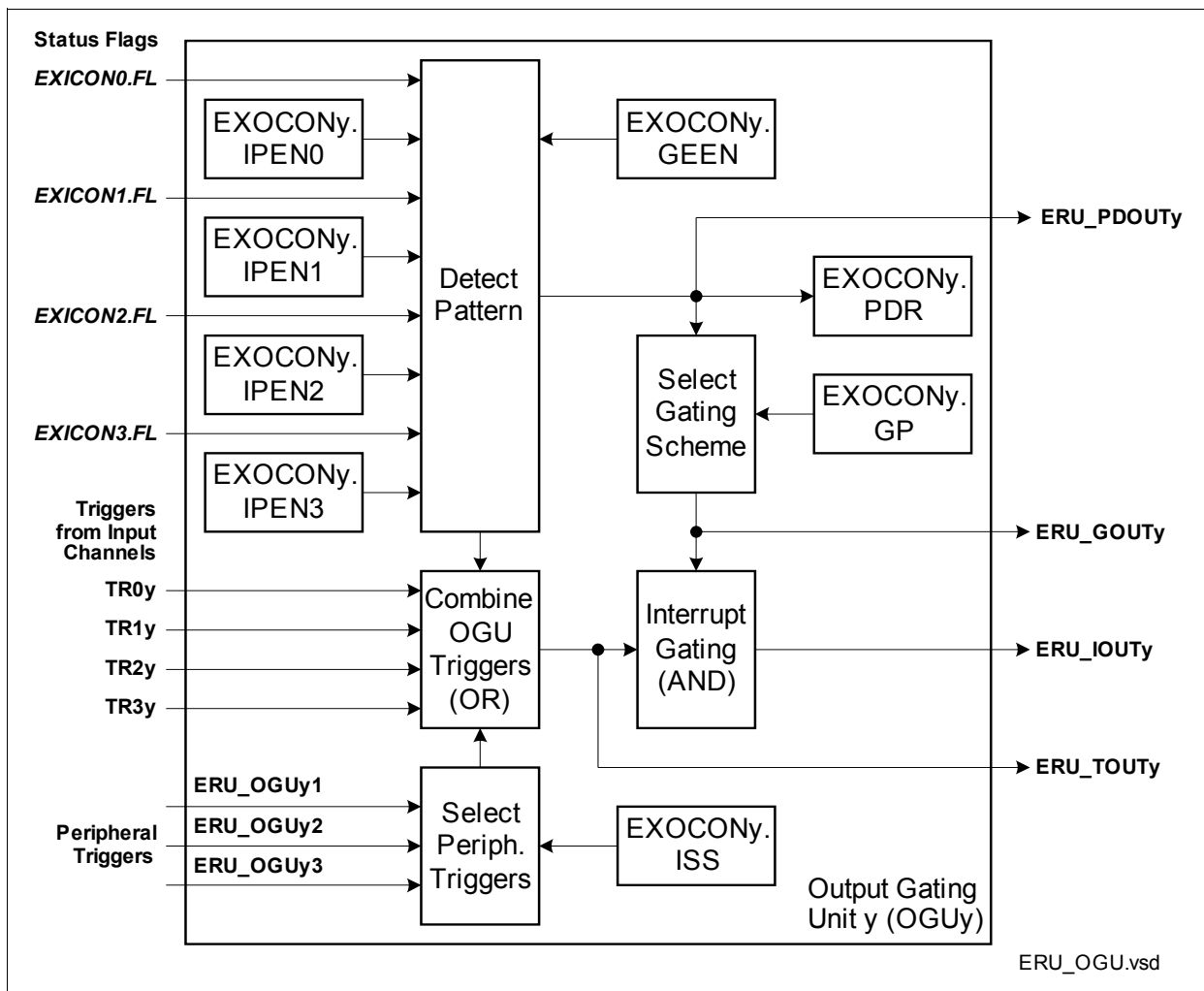
**Figure 8-28 Connecting Matrix between ETLx and OGUy**



### 8.9.6 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 8-29** illustrates the logic blocks within an OGUy unit. All functions of an OGUy unit are controlled by its associated EXOCONy register, e.g. **EXOCON0** for OGU0. The function of an OGUy unit can be split into two parts:

- **Trigger combination** (see **Section 8.9.6.1**):  
 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** (see **Section 8.9.6.2**):  
 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 8-29 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, please refer to [Section 8.9.7](#)):

- **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 interrupts (e.g. the interrupt generation can be gated to allow interrupt activation during a certain time window).

### 8.9.6.1 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 interrupt or to start an ADC conversion. This combination capability allows the generation of an interrupt per OGU that can be triggered by several inputs (multitude of request sources -> one reaction).

[Section 8.17.3.2](#) describes the peripheral trigger connections for the OGUy stages.

The selection is defined by the bit fields ISS in registers **EXOCON0** (for OGU0), **EXOCON1** (for OGU1), **EXOCON2** (for OGU2), or **EXOCON3** (for OGU3).

### 8.9.6.2 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:

**System Control Unit (SCU)**

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

In addition, the pattern detection can deliver a trigger event if the pattern detection result changes from match to miss or vice-versa (if enabled by EXOCONy.GEEN = 1). The pattern result change event is logically OR-combined with the other enabled trigger events to support interrupt 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 interrupt 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 interrupts 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, interrupts 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 interrupt requests ERU\_IOUTy under different conditions:

- **Pattern match** (EXOCONy.GP = 10<sub>B</sub>):  
An interrupt request is issued when a trigger event occurs while the pattern detection shows a pattern match.
- **Pattern miss** (EXOCONy.GP = 11<sub>B</sub>):  
An interrupt request is issued when the trigger event occurs while the pattern detection shows a pattern miss.
- **Independent** of pattern detection (EXOCONy.GP = 01<sub>B</sub>):  
In this mode, each occurring trigger event leads to an interrupt 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 interrupt requests on trigger events).
- **No interrupts** (EXOCONy.GP = 00<sub>B</sub>, default setting)  
In this mode, an occurring trigger event does not lead to an interrupt 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 interrupt requests on trigger events).

## **8.9.7 ERU Output Connections**

**Section 8.17.3.3** describes the connections of the ERU output signals for gating or triggering other module functions, as well as the connections to the interrupt control registers.

## 8.9.8 ERU Registers

### 8.9.8.1 External Input Selection Register EXISEL

This register selects the A and B inputs for all four ERS units. The possible input signals are given in [Table 8-19](#).

#### EXISEL

**External Input Select Register ESFR (F1A0<sub>H</sub>/D0<sub>H</sub>)** **Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>EXS3B</b>		<b>EXS3A</b>		<b>EXS2B</b>		<b>EXS2A</b>		<b>EXS1B</b>		<b>EXS1A</b>		<b>EXS0B</b>		<b>EXS0A</b>	
rw		rw		rw		rw		rw		rw		rw		rw	

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

Field	Bits	Type	Description
<b>EXS2A</b>	[9:8]	rw	<b>External Source Select for A2 (ERS2)</b> This bit field defines which input is selected for A2. 00 <sub>B</sub> Input ERU_2A0 is selected 01 <sub>B</sub> Input ERU_2A1 is selected 10 <sub>B</sub> Input ERU_2A2 is selected 11 <sub>B</sub> Input ERU_2A3 is selected
<b>EXS2B</b>	[11:10]	rw	<b>External Source Select for B2 (ERS2)</b> This bit field defines which input is selected for B2. 00 <sub>B</sub> Input ERU_2B0 is selected 01 <sub>B</sub> Input ERU_2B1 is selected 10 <sub>B</sub> Input ERU_2B2 is selected 11 <sub>B</sub> Input ERU_2B3 is selected
<b>EXS3A</b>	[13:12]	rw	<b>External Source Select for A3 (ERS3)</b> This bit field defines which input is selected for A3. 00 <sub>B</sub> Input ERU_3A0 is selected 01 <sub>B</sub> Input ERU_3A1 is selected 10 <sub>B</sub> Input ERU_3A2 is selected 11 <sub>B</sub> Input ERU_3A3 is selected
<b>EXS3B</b>	[15:14]	rw	<b>External Source Select for B3 (ERS3)</b> This bit field defines which input is selected for B3. 00 <sub>B</sub> Input ERU_3B0 is selected 01 <sub>B</sub> Input ERU_3B1 is selected 10 <sub>B</sub> Input ERU_3B2 is selected 11 <sub>B</sub> Input ERU_3B3 is selected

### 8.9.8.2 External Input Control Registers EXICONx

These registers control the inputs of the ERSx unit and the trigger functions of the ETLx units (x = 0..3).

#### EXICON0

**External Input Control 0 Register**

**ESFR (F030<sub>H</sub>/18<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

#### EXICON1

**External Input Control 1 Register**

**ESFR (F032<sub>H</sub>/19<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

#### EXICON2

**External Input Control 2 Register**

**ESFR (F034<sub>H</sub>/1A<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

#### EXICON3

**External Input Control 3 Register**

**ESFR (F036<sub>H</sub>/1C<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>0</b>				<b>NB</b>	<b>NA</b>	<b>SS</b>		<b>FL</b>	<b>OCS</b>			<b>FE</b>	<b>RE</b>	<b>LD</b>	<b>PE</b>
r				rw	rw	rw		rwh	rw			rw	rw	rw	rw

Field	Bits	Type	Description
<b>PE</b>	0	rw	<b>Output Trigger Pulse Enable for ETLx</b> This bit enables the generation of an output trigger pulse at TRxy when the selected edge is detected (set condition for the status flag FL). 0 <sub>B</sub> The trigger pulse generation is disabled 1 <sub>B</sub> The trigger pulse generation is enabled

Field	Bits	Type	Description
<b>LD</b>	1	rw	<b>Rebuild Level Detection for Status Flag for ETLx</b> This bit selects if the status flag FL is used as “sticky” bit or if it rebuilds the result of a level detection. 0 <sub>B</sub> The status flag FL is not cleared by hardware and is used as “sticky” bit. Once set, it is not influenced by any edge until it becomes cleared by software. 1 <sub>B</sub> 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.
<b>RE</b>	2	rw	<b>Rising Edge Detection Enable ETLx</b> This bit enables/disables the rising edge event as edge event as set condition for the status flag FL or as possible trigger pulse for TRxy. 0 <sub>B</sub> A rising edge is not considered as edge event 1 <sub>B</sub> A rising edge is considered as edge event
<b>FE</b>	3	rw	<b>Falling Edge Detection Enable ETLx</b> This bit enables/disables the falling edge event as edge event as set condition for the status flag FL or as possible trigger pulse for TRxy. 0 <sub>B</sub> A falling edge is not considered as edge event 1 <sub>B</sub> A falling edge is considered as edge event
<b>OCS</b>	[6:4]	rw	<b>Output Channel Select for ETLx Output Trigger Pulse</b> This bit field defines which Output Channel OGUy is targeted by an enabled trigger pulse TRxy. 000 <sub>B</sub> Trigger pulses are sent to OGU0 001 <sub>B</sub> Trigger pulses are sent to OGU1 010 <sub>B</sub> Trigger pulses are sent to OGU2 011 <sub>B</sub> Trigger pulses are sent to OGU3 1XX <sub>B</sub> Reserved, do not use this combination
<b>FL</b>	7	rwh	<b>Status Flag for ETLx</b> This bit represents the status flag that becomes set or cleared by the edge detection. 0 <sub>B</sub> The enabled edge event has not been detected 1 <sub>B</sub> The enabled edge event has been detected



**System Control Unit (SCU)**

Field	Bits	Type	Description
<b>SS</b>	[9:8]	rw	<b>Input Source Select for ERSx</b> This bit field defines which logical combination is taken into account as ESRxO. 00 <sub>B</sub> Input A without additional combination 01 <sub>B</sub> Input B without additional combination 10 <sub>B</sub> Input A OR input B 11 <sub>B</sub> Input A AND input B
<b>NA</b>	10	rw	<b>Input A Negation Select for ERSx</b> This bit selects the polarity for the input A. 0 <sub>B</sub> Input A is used directly 1 <sub>B</sub> Input A is inverted
<b>NB</b>	11	rw	<b>Input B Negation Select for ERSx</b> This bit selects the polarity for the input B. 0 <sub>B</sub> Input B is used directly 1 <sub>B</sub> Input B is inverted
<b>0</b>	[15:12]	r	<b>Reserved</b> Read as 0; should be written with 0.

### 8.9.8.3 Output Control Registers EXOCONy

These registers control the outputs of the Output Gating Unit y (y = 0..3).

#### EXOCON0

**External Output Trigger Control 0 Register**

**SFR (FE30<sub>H</sub>/18<sub>H</sub>)**

**Reset Value: 0008<sub>H</sub>**

#### EXOCON1

**External Output Trigger Control 1 Register**

**SFR (FE32<sub>H</sub>/19<sub>H</sub>)**

**Reset Value: 0008<sub>H</sub>**

#### EXOCON2

**External Output Trigger Control 2 Register**

**SFR (FE34<sub>H</sub>/1A<sub>H</sub>)**

**Reset Value: 0008<sub>H</sub>**

#### EXOCON3

**External Output Trigger Control 3 Register**

**SFR (FE36<sub>H</sub>/1B<sub>H</sub>)**

**Reset Value: 0008<sub>H</sub>**

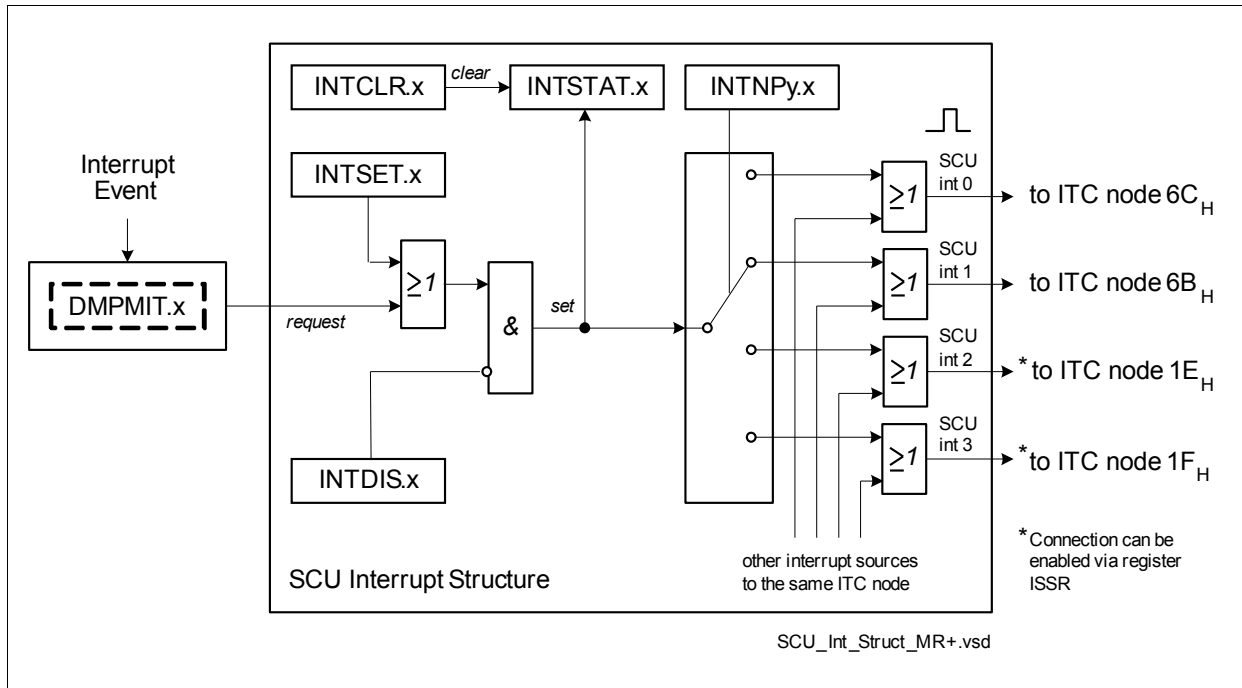
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>IPEN</b>	<b>IPEN</b>	<b>IPEN</b>	<b>IPEN</b>				<b>0</b>				<b>GP</b>	<b>PDR</b>	<b>GE</b>	<b>EN</b>	<b>ISS</b>
<b>3</b>	<b>2</b>	<b>1</b>	<b>0</b>												
rw	rw	rw	rw				r				rw	rh	rw		rw

Field	Bits	Type	Description
<b>ISS</b>	[1:0]	rw	<b>Internal Trigger Source Selection</b> This bit field defines which input is selected as peripheral trigger input for OGUy. The possible input signals are given in <a href="#">Table 8-20</a> . 00 <sub>B</sub> The peripheral trigger function is disabled 01 <sub>B</sub> Input ERU_OGUy1 is selected 10 <sub>B</sub> Input ERU_OGUy2 is selected 11 <sub>B</sub> Input ERU_OGUy3 is selected
<b>GEEN</b>	2	rw	<b>Gating Event Enable</b> Bit GEEN enables the generation of a trigger event when the result of the pattern detection changes from match to miss or vice-versa. 0 <sub>B</sub> The event detection is disabled 1 <sub>B</sub> The event detection is enabled
<b>PDR</b>	3	rh	<b>Pattern Detection Result Flag</b> This bit represents the pattern detection result. 0 <sub>B</sub> A pattern miss is detected 1 <sub>B</sub> A pattern match is detected

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

## 8.10 SCU Interrupt Generation

The interrupt structure of the SCU is shown in **Figure 8-30**.



**Figure 8-30 SCU Interrupt Structure**

If enabled by the corresponding bit in register **INTDIS**, an interrupt is triggered either by the incoming interrupt request line, or by a software set of the respective bit in register **INTSET**. The trigger sets the respective flag in register **INTSTAT** and is gated to one of the interrupt nodes, selected by the node pointer registers **INTNP0** or **INTNP1**.

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

If more than one interrupt source is connected to the same interrupt node pointer (in register **INTNPx**), the requests are combined to one common line.

### Interrupt Node Assignment

The interrupt sources of the SCU module can be mapped to the dedicated interrupt node **6C<sub>H</sub>** or **6B<sub>H</sub>** by programming the interrupt node pointer registers **INTNP0** and **INTNP1**.

Furthermore, If the CAPCOM2 interrupts for channels 30 or 31 are not used the SCU interrupts can be mapped via register **ISSR** to the interrupt nodes **1E<sub>H</sub>** or **1F<sub>H</sub>** which are assigned to the CAPCOM2 interrupts. So for the SCU interrupts can be selected the interrupt node **6C<sub>H</sub>**, **6B<sub>H</sub>**, or in addition via register **ISSR** the node **1E<sub>H</sub>** or **1F<sub>H</sub>**.

The default assignment of the interrupt sources to the nodes and their corresponding control registers are shown in **Table 8-10**.

### 8.10.1 Interrupt Support

Some of the interrupt requests are first fed through a sticky flag register in the DMP\_M domain. These flags are set with a trigger and if set trigger the interrupt generation in the DMP\_1..

Which of the interrupt requests have a sticky flag in register **DMPMIT** is listed in **Table 8-10**.

*Note: When servicing an SCU interrupt request, make sure that all related request flags are cleared after the identified request has been handled. To clear an interrupt request that is stored in register DMPMIT, first clear the request source of the source (e.g. WUTRG), clear the request within DMP\_M via **DMPMITCLR**, and then clear the request within DMP\_1 via INTCLR.*

### 8.10.2 SCU Interrupt Sources

The SCU receives the interrupt request lines listed in **Table 8-10**.

**Table 8-10 SCU Interrupt Overview**

Source of Interrupt	Short Name	Sticky Flag in DMPMIT	Default Interrupt Node Assignment in INTNPx
SWD OK 1 Interrupt	SWDI1	yes	6C <sub>H</sub>
SWD OK 2 Interrupt	SWDI2	yes	6B <sub>H</sub>
PVC_M OK 1 Interrupt	PVCM11	yes	6C <sub>H</sub>
PVC_M OK 2 Interrupt	PVCM12	yes	6B <sub>H</sub>
PVC_1 OK 1 Interrupt	PVC1I1	yes	6C <sub>H</sub>
PVC_1 OK 2 Interrupt	PVC1I2	yes	6B <sub>H</sub>
Wake-up Timer Interrupt	WUI	yes	6B <sub>H</sub>
Wake-up Timer Trim Interrupt	WUTI	yes	6C <sub>H</sub>
Watchdog Timer Interrupt	WDTI	---	6B <sub>H</sub>
GSC Interrupt	GSCI	yes	6C <sub>H</sub>
STM0 Interrupt	STM0I	yes	6B <sub>H</sub>
STM1 Interrupt	STM1I	yes	6C <sub>H</sub>
MCHK Interrupt	MCHKI	---	6B <sub>H</sub>
Program Flash Interrupt	PFI	---	6C <sub>H</sub>

## 8.10.3 Interrupt Control Registers

### 8.10.3.1 Register INTSTAT

This register contains the status flags for all interrupt request trigger sources of the SCU. For setting and clearing of these status bits by software see registers INTSET and INTCLR, respectively.

#### INTSTAT

**Interrupt Status Register**

**SFR (FF00<sub>H</sub>/80<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>0</b>	<b>PF</b> I	<b>M</b> <b>CHK</b> I	<b>STM</b> <b>1</b> I	<b>STM</b> <b>0</b> I	<b>GSC</b> I	<b>WDT</b> I	<b>WU</b> I	<b>WUT</b> I	<b>PVC</b> <b>1</b> <b>I2</b>	<b>PVC</b> <b>1</b> <b>I1</b>	<b>PVC</b> <b>M</b> <b>I2</b>	<b>PVC</b> <b>M</b> <b>I1</b>	<b>SWD</b> <b>I2</b>	<b>SWD</b> <b>I1</b>	
rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
<b>SWDI1</b>	0	rh	<b>SWD Interrupt Request Flag 1</b> This bit is set if bit DMPMIT.SWDI1 is set. 0 <sub>B</sub> No SWDI1 interrupt trigger has occurred since this bit was cleared the last time 1 <sub>B</sub> A SWDI1 interrupt trigger has occurred since this bit was cleared the last time
<b>SWDI2</b>	1	rh	<b>SWD Interrupt Request Flag 2</b> This bit is set if bit DMPMIT.SWDI2 is set. 0 <sub>B</sub> No SWDI2 interrupt trigger has occurred since this bit was cleared the last time 1 <sub>B</sub> A SWDI2 interrupt trigger has occurred since this bit was cleared the last time
<b>PVCMI1</b>	2	rh	<b>PVC_M Interrupt Request Flag 1</b> This bit is set if bit DMPMIT.PVCMI1 is set. 0 <sub>B</sub> No PVCMI1 interrupt trigger has occurred since this bit was cleared the last time 1 <sub>B</sub> A PVCMI1 interrupt trigger has occurred since this bit was cleared the last time

Field	Bits	Type	Description
PVCM12	3	rh	<b>PVC_M Interrupt Request Flag 2</b> This bit is set if bit DMPMIT.PVCM12 is set. 0 <sub>B</sub> No PVC12 interrupt trigger has occurred since this bit was cleared the last time 1 <sub>B</sub> A PVC12 interrupt trigger has occurred since this bit was cleared the last time
PVC111	4	rh	<b>PVC_1 Interrupt Request Flag 1</b> This bit is set if bit DMPMIT.PVC111 is set. 0 <sub>B</sub> No PVC111 interrupt trigger has occurred since this bit was cleared the last time 1 <sub>B</sub> A PVC111 interrupt trigger has occurred since this bit was cleared the last time
PVC112	5	rh	<b>PVC_1 Interrupt Request Flag 2</b> This bit is set if bit DMPMIT.PVC112 is set. 0 <sub>B</sub> No PVC112 interrupt trigger has occurred since this bit was cleared the last time 1 <sub>B</sub> A PVC112 interrupt trigger has occurred since this bit was cleared the last time
WUTI	6	rh	<b>Wake-up Timer Trim Interrupt Request Flag</b> This bit is set if the WUT trim trigger event occur and bit is INTDIS.WUTI = 0. 0 <sub>B</sub> No WUT interrupt trigger has occurred since this bit was cleared the last time 1 <sub>B</sub> A WUT interrupt trigger has occurred since this bit was cleared the last time
WUI	7	rh	<b>Wake-up Timer Interrupt Request Flag</b> This bit is set if the WU trigger event occur and bit is INTDIS.WUI = 0. 0 <sub>B</sub> No WU interrupt trigger has occurred since this bit was cleared the last time 1 <sub>B</sub> A WU interrupt trigger has occurred since this bit was cleared the last time
WDTI	8	rh	<b>Watchdog Timer Interrupt Request Flag</b> This bit is set if the WDT Prewarning Mode is entered and bit is INTDIS.WDTI = 0. 0 <sub>B</sub> No WDT interrupt trigger has occurred since this bit was cleared the last time 1 <sub>B</sub> A WDT interrupt trigger has occurred since this bit was cleared the last time

Field	Bits	Type	Description
<b>GSCI</b>	9	rh	<b>GSC Interrupt Request Flag</b> This bit is set if the GSC error bit is set and bit is INTDIS.GSCI = 0. 0 <sub>B</sub> No GSC interrupt trigger has occurred since this bit was cleared the last time 1 <sub>B</sub> A GSC interrupt trigger has occurred since this bit was cleared the last time
<b>STM0I</b>	10	rh	<b>STM Interrupt 0 Request Flag</b> This bit is set if the STM interrupt trigger 0 is set and bit is INTDIS.STM0I = 0. 0 <sub>B</sub> No STM0 interrupt trigger has occurred since this bit was cleared the last time 1 <sub>B</sub> A STM0 interrupt trigger has occurred since this bit was cleared the last time
<b>STM1I</b>	11	rh	<b>STM Interrupt 1 Request Flag</b> This bit is set if the STM interrupt trigger 1 is set and bit is INTDIS.STM1I = 0. 0 <sub>B</sub> No STM1 interrupt trigger has occurred since this bit was cleared the last time 1 <sub>B</sub> A STM1 interrupt trigger has occurred since this bit was cleared the last time
<b>MCHKI</b>	12	rh	<b>MCHK Interrupt Request Flag</b> This bit is set if the MCHK interrupt trigger is set and bit is INTDIS.MCHKI = 0. 0 <sub>B</sub> No MCHK interrupt trigger has occurred since this bit was cleared the last time 1 <sub>B</sub> A MCHK interrupt trigger has occurred since this bit was cleared the last time
<b>PFI</b>	13	rh	<b>Program Flash Interrupt Request Flag</b> This bit is set if the Program Flash interrupt trigger is set and bit is INTDIS.PFI = 0. 0 <sub>B</sub> No PF interrupt trigger has occurred since this bit was cleared the last time 1 <sub>B</sub> A PF interrupt trigger has occurred since this bit was cleared the last time
<b>0</b>	[15:14]	rh	<b>Reserved</b> Read as 0; should be written with 0.



### 8.10.3.2 Register INTCLR

This register contains the software clear option for all status flags of all interrupt request trigger sources of the SCU.

#### INTCLR

**Interrupt Clear Register**

**SFR (FE82<sub>H</sub>/41<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	PF I	M CHK I	STM 1 I	STM 0 I	GSC I	WDT I	WU I	WUT I	PVC 1 I2	PVC 1 I1	PVC M I2	PVC M I1	SWD I2	SWD I1	
W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W

Field	Bits	Type	Description
<b>SWDI1</b>	0	w	<b>Clear SWD Interrupt Request Flag 1</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit INTSTAT.SWDI1 is cleared
<b>SWDI2</b>	1	w	<b>Clear SWD Interrupt Request Flag 2</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit INTSTAT.SWDI2 is cleared
<b>PVCMI1</b>	2	w	<b>Clear PVC_M Interrupt Request Flag 1</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit INTSTAT.PVCMI1 is cleared
<b>PVCMI2</b>	3	w	<b>Clear PVC_M Interrupt Request Flag 2</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit INTSTAT.PVCMI2 is cleared
<b>PVC1I1</b>	4	w	<b>Clear PVC_1 Interrupt Request Flag 1</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit INTSTAT.PVC1I1 is cleared
<b>PVC1I2</b>	5	w	<b>Clear PVC_1 Interrupt Request Flag 2</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit INTSTAT.PVC1I2 is cleared
<b>WUTI</b>	6	w	<b>Clear Wake-up Timer Trim Interrupt Request Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit INTSTAT.WUTI is cleared
<b>WUI</b>	7	w	<b>Clear Wake-up Timer Interrupt Request Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit INTSTAT.WUI is cleared

Field	Bits	Type	Description
<b>WDTI</b>	8	w	<b>Clear Watchdog Timer Interrupt Request Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit INTSTAT.WDTI is cleared
<b>GSCI</b>	9	w	<b>Clear GSC Interrupt Request Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit INTSTAT.GSCI is cleared
<b>STM0I</b>	10	w	<b>Clear STM0 Interrupt Request Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit INTSTAT.STM0I is cleared
<b>STM1I</b>	11	w	<b>Clear STM1 Interrupt Request Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit INTSTAT.STM1I is cleared
<b>MCHKI</b>	12	w	<b>Clear MCHK Interrupt Request Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit INTSTAT.MCHKI is cleared
<b>PFI</b>	13	w	<b>Clear Program Flash Interrupt Request Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit INTSTAT.PFI is cleared
<b>0</b>	[15:14]	w	<b>Reserved</b> Must be written with 0.

*Note: These bits are always read as 0.*

### 8.10.3.3 Register INTSET

This register contains the software set option for all status flags of all interrupt request trigger sources of the SCU.

#### INTSET

**Interrupt Set Register**

**SFR (FE80<sub>H</sub>/40<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		PF I	M CHK I	STM 1 I	STM 0 I	GSC I	WDT I	WU I	WUT I	PVC 1 I2	PVC 1 I1	PVC M I2	PVC M I1	SWD I2	SWD I1
W		W	W	W	W	W	W	W	W	W	W	W	W	W	W

Field	Bits	Type	Description
<b>SWDI1</b>	0	w	<b>Set SWD Interrupt Request Flag 1</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit INTSTAT.SWDI1 is set
<b>SWDI2</b>	1	w	<b>Set SWD Interrupt Request Flag 2</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit INTSTAT.SWDI2 is set
<b>PVCMI1</b>	2	w	<b>Set PVC_M Interrupt Request Flag 1</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit INTSTAT.PVCMI1 is set
<b>PVCMI2</b>	3	w	<b>Set PVC_M Interrupt Request Flag 2</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit INTSTAT.PVCMI2 is set
<b>PVC1I1</b>	4	w	<b>Set PVC_1 Interrupt Request Flag 1</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit INTSTAT.PVC1I1 is set
<b>PVC1I2</b>	5	w	<b>Set PVC_1 Interrupt Request Flag 2</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit INTSTAT.PVC1I2 is set
<b>WUTI</b>	6	w	<b>Set Wake-up Timer Trim Interrupt Request Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit INTSTAT.WUTI is set
<b>WUI</b>	7	w	<b>Set Wake-up Timer Interrupt Request Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit INTSTAT.WUI is set

<b>Field</b>	<b>Bits</b>	<b>Type</b>	<b>Description</b>
<b>WDTI</b>	8	w	<b>Set Watchdog Timer Interrupt Request Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit INTSTAT.WDTI is set
<b>GSCI</b>	9	w	<b>Set GSC Interrupt Request Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit INTSTAT.GSCI is set
<b>STM0I</b>	10	w	<b>Set STM0 Interrupt Request Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit INTSTAT.STM0I is set
<b>STM1I</b>	11	w	<b>Set STM1 Interrupt Request Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit INTSTAT.STM1I is set
<b>MCHKI</b>	12	w	<b>Set MCHK Interrupt Request Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit INTSTAT.MCHKI is set
<b>PFI</b>	13	w	<b>Set Program Flash Interrupt Request Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit INTSTAT.PFI is set
<b>0</b>	[15:14]	w	<b>Reserved</b> Must be written with 0.

*Note: These bits are always read as 0.*

### 8.10.3.4 Register INTDIS

This register contains the software disable control for all interrupt request trigger sources of the SCU.

#### INTDIS

**Interrupt Disable Register**

**SFR (FE84<sub>H</sub>/42<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	PF I	M CHK I	STM 1 I	STM 0 I	GSC I	WDT I	WU I	WUT I	PVC 1 I2	PVC 1 I1	PVC M I2	PVC M I1	SWD I2	SWD I1	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
<b>SWDI1</b>	0	rw	<b>Disable SWD Interrupt Request 1</b> 0 <sub>B</sub> SWDI1 interrupt request enabled 1 <sub>B</sub> SWDI1 interrupt request disabled
<b>SWDI2</b>	1	rw	<b>Disable SWD Interrupt Request 2</b> 0 <sub>B</sub> SWDI2 interrupt request enabled 1 <sub>B</sub> SWDI2 interrupt request disabled
<b>PVCMI1</b>	2	rw	<b>Disable PVC_M Interrupt Request 1</b> 0 <sub>B</sub> PVCMI1 interrupt request enabled 1 <sub>B</sub> PVCMI1 interrupt request disabled
<b>PVCMI2</b>	3	rw	<b>Disable PVC_M Interrupt Request 2</b> 0 <sub>B</sub> PVCMI2 interrupt request enabled 1 <sub>B</sub> PVCMI2 interrupt request disabled
<b>PVC1I1</b>	4	rw	<b>Disable PVC_1 Interrupt Request 1</b> 0 <sub>B</sub> PVC1I1 interrupt request enabled 1 <sub>B</sub> PVC1I1 interrupt request disabled
<b>PVC1I2</b>	5	rw	<b>Disable PVC_1 Interrupt Request 2</b> 0 <sub>B</sub> PVC1I2 interrupt request enabled 1 <sub>B</sub> PVC1I2 interrupt request disabled
<b>WUTI</b>	6	rw	<b>Disable Wake-up Timer Trim Interrupt Request</b> 0 <sub>B</sub> WUT interrupt request enabled 1 <sub>B</sub> WUT interrupt request disabled
<b>WUI</b>	7	rw	<b>Disable Wake-up Timer Interrupt Request</b> 0 <sub>B</sub> WU interrupt request enabled 1 <sub>B</sub> WU interrupt request disabled

**System Control Unit (SCU)**

<b>Field</b>	<b>Bits</b>	<b>Type</b>	<b>Description</b>
<b>WDTI</b>	8	rw	<b>Disable Watchdog Timer Interrupt Request</b> 0 <sub>B</sub> WDT interrupt request enabled 1 <sub>B</sub> WDT interrupt request disabled
<b>GSCI</b>	9	rw	<b>Disable GSC Interrupt Request</b> 0 <sub>B</sub> GSC interrupt request enabled 1 <sub>B</sub> GSC interrupt request disabled
<b>STM0I</b>	10	rw	<b>Disable STM0 Interrupt Request</b> 0 <sub>B</sub> STM0 interrupt request enabled 1 <sub>B</sub> STM0 interrupt request disabled
<b>STM1I</b>	11	rw	<b>Disable STM1 Interrupt Request</b> 0 <sub>B</sub> STM1 interrupt request enabled 1 <sub>B</sub> STM1 interrupt request disabled
<b>MCHKI</b>	12	rw	<b>Disable MCHK Interrupt Request</b> 0 <sub>B</sub> MCHK interrupt request enabled 1 <sub>B</sub> MCHK interrupt request disabled
<b>PFI</b>	13	rw	<b>Disable Program Flash Interrupt Request</b> 0 <sub>B</sub> PF interrupt request enabled 1 <sub>B</sub> PF interrupt request disabled
<b>0</b>	[15:14]	rw	<b>Reserved</b> Should be written with 0.

### 8.10.3.5 Registers INTNP0 and INPNP1

These registers contain the control for the interrupt node pointers of all interrupt request trigger sources of the SCU.

#### INTNP0

##### Interrupt Node Pointer 0 Register

**SFR (FE86<sub>H</sub>/43<sub>H</sub>)**

**Reset Value: 4444<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>WU</b>		<b>WUT</b>		<b>PVC12</b>		<b>PVC11</b>		<b>PVCM2</b>		<b>PVCM1</b>		<b>SWD2</b>		<b>SWD1</b>	
rw		rw		rw		rw		rw		rw		rw		rw	

Field	Bits	Type	Description
<b>SWD1</b>	[1:0]	rw	<b>Interrupt Node Pointer for SWD 1 Interrupts</b> This bit field defines the interrupt node, which is requested due to the set condition for bit INTSTAT.SWDI1 (if enabled by bit INTDIS.SWDI1). 00 <sub>B</sub> Interrupt node 6C <sub>H</sub> is selected 01 <sub>B</sub> Interrupt node 6B <sub>H</sub> is selected 10 <sub>B</sub> Interrupt node 1E <sub>H</sub> is selected if enabled by bit ISSR.ISS14 (bit is set) 11 <sub>B</sub> Interrupt node 1F <sub>H</sub> is selected if enabled by bit ISSR.ISS15 (bit is set)
<b>SWD2</b>	[3:2]	rw	<b>Interrupt Node Pointer for SWD 2 Interrupts</b> This bit field defines the interrupt node, which is requested due to the set condition for bit INTSTAT.SWDI2 (if enabled by bit INTDIS.SWDI2). 00 <sub>B</sub> Interrupt node 6C <sub>H</sub> is selected 01 <sub>B</sub> Interrupt node 6B <sub>H</sub> is selected 10 <sub>B</sub> Interrupt node 1E <sub>H</sub> is selected if enabled by bit ISSR.ISS14 (bit is set) 11 <sub>B</sub> Interrupt node 1F <sub>H</sub> is selected if enabled by bit ISSR.ISS15 (bit is set)

Field	Bits	Type	Description
<b>PVCM1</b>	[5:4]	rw	<b>Interrupt Node Pointer for PVC_M 1 Interrupts</b> This bit field defines the interrupt node, which is requested due to the set condition for bit INTSTAT.PCVMI1 (if enabled by bit INTDIS.PVCM11). 00 <sub>B</sub> Interrupt node 6C <sub>H</sub> is selected 01 <sub>B</sub> Interrupt node 6B <sub>H</sub> is selected 10 <sub>B</sub> Interrupt node 1E <sub>H</sub> is selected if enabled by bit ISSR.ISS14 (bit is set) 11 <sub>B</sub> Interrupt node 1F <sub>H</sub> is selected if enabled by bit ISSR.ISS15 (bit is set)
<b>PVCM2</b>	[7:6]	rw	<b>Interrupt Node Pointer for PVC_M 2 Interrupts</b> This bit field defines the interrupt node, which is requested due to the set condition for bit INTSTAT.PCVMI2 (if enabled by bit INTDIS.PVCM12). 00 <sub>B</sub> Interrupt node 6C <sub>H</sub> is selected 01 <sub>B</sub> Interrupt node 6B <sub>H</sub> is selected 10 <sub>B</sub> Interrupt node 1E <sub>H</sub> is selected if enabled by bit ISSR.ISS14 (bit is set) 11 <sub>B</sub> Interrupt node 1F <sub>H</sub> is selected if enabled by bit ISSR.ISS15 (bit is set)
<b>PVC11</b>	[9:8]	rw	<b>Interrupt Node Pointer for PVC_1 1 Interrupts</b> This bit field defines the interrupt node, which is requested due to the set condition for bit INTSTAT.PCV111 (if enabled by bit INTDIS.PVC111). 00 <sub>B</sub> Interrupt node 6C <sub>H</sub> is selected 01 <sub>B</sub> Interrupt node 6B <sub>H</sub> is selected 10 <sub>B</sub> Interrupt node 1E <sub>H</sub> is selected if enabled by bit ISSR.ISS14 (bit is set) 11 <sub>B</sub> Interrupt node 1F <sub>H</sub> is selected if enabled by bit ISSR.ISS15 (bit is set)
<b>PVC12</b>	[11:10]	rw	<b>Interrupt Node Pointer for PVC_1 2 Interrupts</b> This bit field defines the interrupt node, which is requested due to the set condition for bit INTSTAT.PCV112 (if enabled by bit INTDIS.PVC112). 00 <sub>B</sub> Interrupt node 6C <sub>H</sub> is selected 01 <sub>B</sub> Interrupt node 6B <sub>H</sub> is selected 10 <sub>B</sub> Interrupt node 1E <sub>H</sub> is selected if enabled by bit ISSR.ISS14 (bit is set) 11 <sub>B</sub> Interrupt node 1F <sub>H</sub> is selected if enabled by bit ISSR.ISS15 (bit is set)



Field	Bits	Type	Description
<b>WUT</b>	[13:12]	rw	<b>Interrupt Node Pointer for Wake-up Timer Trim Interrupts</b> This bit field defines the interrupt node, which is requested due to the set condition for bit INTSTAT.WUTI (if enabled by bit INTDIS.WUTI). 00 <sub>B</sub> Interrupt node 6C <sub>H</sub> is selected 01 <sub>B</sub> Interrupt node 6B <sub>H</sub> is selected 10 <sub>B</sub> Interrupt node 1E <sub>H</sub> is selected if enabled by bit ISSR.ISS14 (bit is set) 11 <sub>B</sub> Interrupt node 1F <sub>H</sub> is selected if enabled by bit ISSR.ISS15 (bit is set)
<b>WU</b>	[15:14]	rw	<b>Interrupt Node Pointer for Wake-up Timer Interrupts</b> This bit field defines the interrupt node, which is requested due to the set condition for bit INTSTAT.WUI (if enabled by bit INTDIS.WUI). 00 <sub>B</sub> Interrupt node 6C <sub>H</sub> is selected 01 <sub>B</sub> Interrupt node 6B <sub>H</sub> is selected 10 <sub>B</sub> Interrupt node 1E <sub>H</sub> is selected if enabled by bit ISSR.ISS14 (bit is set) 11 <sub>B</sub> Interrupt node 1F <sub>H</sub> is selected if enabled by bit ISSR.ISS15 (bit is set)

## INTNP1

### Interrupt Node Pointer 1 Register

**SFR (FE88<sub>H</sub>/44<sub>H</sub>)**

**Reset Value: 1111<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-				<b>PF</b>		<b>MCHK</b>		<b>STM1</b>		<b>STM0</b>		<b>GSC</b>		<b>WDT</b>	
-				rw		rw		rw		rw		rw		rw	

Field	Bits	Type	Description
<b>WDT</b>	[1:0]	rw	<b>Interrupt Node Pointer for WDT Interrupts</b> This bit field defines the interrupt node, which is requested due to the set condition for bit INTSTAT.WDTI (if enabled by bit INTDIS.WDTI). 00 <sub>B</sub> Interrupt node 6C <sub>H</sub> is selected 01 <sub>B</sub> Interrupt node 6B <sub>H</sub> is selected 10 <sub>B</sub> Interrupt node 1E <sub>H</sub> is selected if enabled by bit ISSR.ISS14 (bit is set) 11 <sub>B</sub> Interrupt node 1F <sub>H</sub> is selected if enabled by bit ISSR.ISS15 (bit is set)
<b>GSC</b>	[3:2]	rw	<b>Interrupt Node Pointer for GSC Interrupts</b> This bit field defines the interrupt node, which is requested due to the set condition for bit INTSTAT.GSCI (if enabled by bit INTDIS.GSCI). 00 <sub>B</sub> Interrupt node 6C <sub>H</sub> is selected 01 <sub>B</sub> Interrupt node 6B <sub>H</sub> is selected 10 <sub>B</sub> Interrupt node 1E <sub>H</sub> is selected if enabled by bit ISSR.ISS14 (bit is set) 11 <sub>B</sub> Interrupt node 1F <sub>H</sub> is selected if enabled by bit ISSR.ISS15 (bit is set)
<b>STM0</b>	[5:4]	rw	<b>Interrupt Node Pointer for STM0 Interrupts</b> This bit field defines the interrupt node, which is requested due to the set condition for bit INTSTAT.STM0I (if enabled by bit INTDIS.STM0I). 00 <sub>B</sub> Interrupt node 6C <sub>H</sub> is selected 01 <sub>B</sub> Interrupt node 6B <sub>H</sub> is selected 10 <sub>B</sub> Interrupt node 1E <sub>H</sub> is selected if enabled by bit ISSR.ISS14 (bit is set) 11 <sub>B</sub> Interrupt node 1F <sub>H</sub> is selected if enabled by bit ISSR.ISS15 (bit is set)

<b>Field</b>	<b>Bits</b>	<b>Type</b>	<b>Description</b>
<b>STM1</b>	[7:6]	rw	<b>Interrupt Node Pointer for STM1 Interrupts</b> This bit field defines the interrupt node, which is requested due to the set condition for bit INTSTAT.STM1I (if enabled by bit INTDIS.STM1I). 00 <sub>B</sub> Interrupt node 6C <sub>H</sub> is selected 01 <sub>B</sub> Interrupt node 6B <sub>H</sub> is selected 10 <sub>B</sub> Interrupt node 1E <sub>H</sub> is selected if enabled by bit ISSR.ISS14 (bit is set) 11 <sub>B</sub> Interrupt node 1F <sub>H</sub> is selected if enabled by bit ISSR.ISS15 (bit is set)
<b>MCHK</b>	[9:8]	rw	<b>Interrupt Node Pointer for MCHK Interrupts</b> This bit field defines the interrupt node, which is requested due to the set condition for bit INTSTAT.MCHKI (if enabled by bit INTDIS.MCHKI). 00 <sub>B</sub> Interrupt node 6C <sub>H</sub> is selected 01 <sub>B</sub> Interrupt node 6B <sub>H</sub> is selected 10 <sub>B</sub> Interrupt node 1E <sub>H</sub> is selected if enabled by bit ISSR.ISS14 (bit is set) 11 <sub>B</sub> Interrupt node 1F <sub>H</sub> is selected if enabled by bit ISSR.ISS15 (bit is set)
<b>PF</b>	[11:10]	rw	<b>Interrupt Node Pointer for Program Flash Interrupts</b> This bit field defines the interrupt node, which is requested due to the set condition for bit INTSTAT.PFI (if enabled by bit INTDIS.PFI). 00 <sub>B</sub> Interrupt node 6C <sub>H</sub> is selected 01 <sub>B</sub> Interrupt node 6B <sub>H</sub> is selected 10 <sub>B</sub> Interrupt node 1E <sub>H</sub> is selected if enabled by bit ISSR.ISS14 (bit is set) 11 <sub>B</sub> Interrupt node 1F <sub>H</sub> is selected if enabled by bit ISSR.ISS15 (bit is set)

### 8.10.3.6 Register DMPMIT

This register contains additional sticky interrupt and trap flags within the DMP\_M power domain.

#### DMPMIT

#### DMP\_M Interrupt and Trap Trigger Register

**SFR (FE96<sub>H</sub>/4B<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>RA T</b>	<b>0</b>	<b>ESR 2 T</b>	<b>ESR 1 T</b>	<b>ESR 0 T</b>	<b>STM 1</b>	<b>STM 0</b>	<b>GSC</b>	<b>WU I</b>	<b>WUT I</b>	<b>PVC 1 I2</b>	<b>PVC 1 I1</b>	<b>PVC M I2</b>	<b>PVC M I1</b>	<b>SWD I2</b>	<b>SWD I1</b>
rh	r	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
<b>SWDI1</b>	0	rh	<b>SWD Interrupt Request Flag 1</b> This bit is set if bit SWDCON0.L1OK matches the action level defined by SWDCON0.L1ALEV and SWDCON0.L1INTEN = 1 <sub>B</sub> . 0 <sub>B</sub> No SWDI1 interrupt was requested since this bit was cleared the last time 1 <sub>B</sub> A SWDI1 interrupt was requested since this bit was cleared the last time
<b>SWDI2</b>	1	rh	<b>SWD Interrupt Request Flag 2</b> This bit is set if bit SWDCON0.L2OK matches the action level defined by SWDCON0.L2ALEV and SWDCON0.L2INTEN = 1 <sub>B</sub> . 0 <sub>B</sub> No SWDI2 interrupt was requested since this bit was cleared the last time 1 <sub>B</sub> A SWDI2 interrupt was requested since this bit was cleared the last time
<b>PVCM I1</b>	2	rh	<b>PVC_M Interrupt Request Flag 1</b> This bit is set if bit PVCMDCON0.LEV1OK is cleared and PVCMDCON0.L1INTEN = 1 <sub>B</sub> . 0 <sub>B</sub> No PVCM I1 interrupt was requested since this bit was cleared the last time 1 <sub>B</sub> A PVCM I1 interrupt was requested since this bit was cleared the last time

Field	Bits	Type	Description
<b>PVCM12</b>	3	rh	<b>PVC_M Interrupt Request Flag 2</b> This bit is set if bit PVC1CON0.LEV2OK is cleared and PVC1CON0.L2INTEN = 1 <sub>B</sub> . 0 <sub>B</sub> No PVC12 interrupt was requested since this bit was cleared the last time 1 <sub>B</sub> A PVC12 interrupt was requested since this bit was cleared the last time
<b>PVC111</b>	4	rh	<b>PVC_1 Interrupt Request Flag 1</b> This bit is set if bit PVC1CON0.LEV1OK is cleared and PVC1CON0.L1INTEN = 1 <sub>B</sub> . 0 <sub>B</sub> No PVC111 interrupt was requested since this bit was cleared the last time 1 <sub>B</sub> A PVC111 interrupt was requested since this bit was cleared the last time
<b>PVC112</b>	5	rh	<b>PVC_1 Interrupt Request Flag 2</b> This bit is set if bit PVC1CON0.LEV2OK is cleared and PVC1CON0.L2INTEN = 1 <sub>B</sub> . 0 <sub>B</sub> No PVC112 interrupt was requested since this bit was cleared the last time 1 <sub>B</sub> A PVC112 interrupt was requested since this bit was cleared the last time
<b>WUTI</b>	6	rh	<b>Wake-up Timer Trim Interrupt Request Flag</b> This bit is set if a wake-up timer trim trigger occurs. 0 <sub>B</sub> No WUT interrupt was requested since this bit was cleared the last time 1 <sub>B</sub> A WUT interrupt was requested since this bit was cleared the last time
<b>WUI</b>	7	rh	<b>Wake-up Timer Interrupt Request Flag</b> This bit is set if a wake-up timer trigger occurs. 0 <sub>B</sub> No WU interrupt was requested since this bit was cleared the last time 1 <sub>B</sub> A WU interrupt was requested since this bit was cleared the last time
<b>GSC</b>	8	rh	<b>GSC Interrupt Request Flag</b> This bit is set if a GSC trigger occurs. 0 <sub>B</sub> No GSC interrupt was requested since this bit was cleared the last time 1 <sub>B</sub> A GSC interrupt was requested since this bit was cleared the last time

Field	Bits	Type	Description
<b>STM0</b>	9	rh	<b>STM0 Interrupt Request Flag</b> This bit is set if a STM0 trigger occurs. 0 <sub>B</sub> No STM0 interrupt was requested since this bit was cleared the last time 1 <sub>B</sub> A STM0 interrupt was requested since this bit was cleared the last time
<b>STM1</b>	10	rh	<b>STM1 Interrupt Request Flag</b> This bit is set if a STM1 trigger occurs. 0 <sub>B</sub> No STM1 interrupt was requested since this bit was cleared the last time 1 <sub>B</sub> A STM1 interrupt was requested since this bit was cleared the last time
<b>ESR0T</b>	11	rh	<b>ESR0 Trap Request Flag</b> This bit is set if pin <u>ESR0</u> is asserted. 0 <sub>B</sub> No ESR0 trap was requested since this bit was cleared the last time 1 <sub>B</sub> An ESR0 trap was requested since this bit was cleared the last time
<b>ESR1T</b>	12	rh	<b>ESR1 Trap Request Flag</b> This bit is set if pin <u>ESR1</u> is asserted. 0 <sub>B</sub> No ESR1 trap was requested since this bit was cleared the last time 1 <sub>B</sub> An ESR1 trap was requested since this bit was cleared the last time
<b>ESR2T</b>	13	rh	<b>ESR2 Trap Request Flag</b> This bit is set if pin <u>ESR2</u> is asserted. 0 <sub>B</sub> No ESR2 trap was requested since this bit was cleared the last time 1 <sub>B</sub> An ESR2 trap was requested since this bit was cleared the last time
<b>RAT</b>	15	rh	<b>Register Access Trap Request Flag</b> This bit is set a protected register is written by an non-authorized access. 0 <sub>B</sub> No RA trap was requested since this bit was cleared the last time 1 <sub>B</sub> A RA trap was requested since this bit was cleared the last time
<b>0</b>	14	r	<b>Reserved</b> Read as 0; should be written with 0.

### 8.10.3.7 Register DMPMITCLR

This register contains the software clear option for all sticky status flags of all interrupt and trap request trigger sources of the DMP\_M power domain.

#### DMPMITCLR

#### DMP\_M Interrupt and Trap Clear Register

**SFR (FE98<sub>H</sub>/4C<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>RA</b> <b>T</b>	<b>0</b>	<b>ESR</b> <b>2</b> <b>T</b>	<b>ESR</b> <b>1</b> <b>T</b>	<b>ESR</b> <b>0</b> <b>T</b>	<b>STM</b> <b>1</b>	<b>STM</b> <b>0</b>	<b>GSC</b>	<b>W</b> <b>UI</b>	<b>WUT</b> <b>I</b>	<b>PVC</b> <b>1</b> <b>I2</b>	<b>PVC</b> <b>1</b> <b>I1</b>	<b>PVC</b> <b>M</b> <b>I2</b>	<b>PVC</b> <b>M</b> <b>I1</b>	<b>SWD</b> <b>I2</b>	<b>SWD</b> <b>I1</b>
w	r	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Field	Bits	Type	Description
<b>SWDI1</b>	0	w	<b>Clear SWD1 Interrupt Request Flag 1</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit DMPMIT.SWDI1 is cleared
<b>SWDI2</b>	1	w	<b>Clear SWD Interrupt Request Flag 2</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit DMPMIT.SWDI2 is cleared
<b>PVCMI1</b>	2	w	<b>Clear PVC_M Interrupt Request Flag 1</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit DMPMIT.PVCMI1 is cleared
<b>PVCMI2</b>	3	w	<b>Clear PVC_M Interrupt Request Flag 2</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit DMPMIT.PVCMI2 is cleared
<b>PVC1I1</b>	4	w	<b>Clear PVC_1 Interrupt Request Flag 1</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit DMPMIT.PVC1I1 is cleared
<b>PVC1I2</b>	5	w	<b>Clear PVC_1 Interrupt Request Flag 2</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit DMPMIT.PVC1I2 is cleared
<b>WUTI</b>	6	w	<b>Clear Wake-up Trim Interrupt Request Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit DMPMIT.WUTI is cleared
<b>WUI</b>	7	w	<b>Clear Wake-up Interrupt Request Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit DMPMIT.WUI is cleared

<b>Field</b>	<b>Bits</b>	<b>Type</b>	<b>Description</b>
<b>GSC</b>	8	w	<b>Clear GSC Interrupt Request Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit DMPMIT.GSCI is cleared
<b>STM0</b>	9	w	<b>Clear STM0 Interrupt Request Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit DMPMIT.STM0I is cleared
<b>STM1</b>	10	w	<b>Clear STM1 Interrupt Request Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit DMPMIT.STM1I is cleared
<b>ESR0T</b>	11	w	<b>Clear ESR0 Trap Request Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit DMPMIT.ESR0T is cleared
<b>ESR1T</b>	12	w	<b>Clear ESR1 Trap Request Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit DMPMIT.ESR1T is cleared
<b>ESR2T</b>	13	w	<b>Clear ESR2 Trap Request Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit DMPMIT.ESR2T is cleared
<b>RAT</b>	15	w	<b>Clear Register Access Trap Request Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit DMPMIT.RAT is cleared
<b>0</b>	14	r	<b>Reserved</b> Read as 0; should be written with 0.

*Note: The bits of type w are always read as 0.*



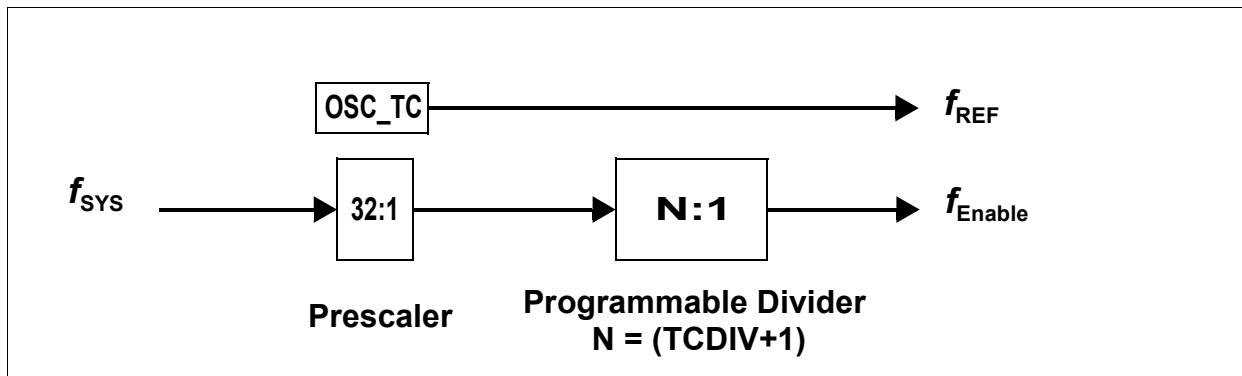
## 8.11 Temperature Compensation Unit

The temperature compensation for the port drivers provides driver output characteristics which are stable (within a certain band of parameter variation) over the specified temperature range.

The temperature compensation oscillator (sensor) provides a reference clock from a free-running temperature-dependent oscillator. An enable trigger is used to define counting cycles where the reference clock pulses are accumulated to build the sensor value TCLR.THCOUNT. The enable trigger is derived from the system clock by a prescaler and a programmable divider (see [Figure 8-31](#)). The value for the programmable divider must be written by the user according to the selected system frequency.

After the count cycle, the resulting count value, i.e. the number of reference clock cycles, is copied to bit field TCLR.THCOUNT. Thus, TCLR.THCOUNT is updated after every count cycle while the temperature compensation is enabled.

Software can compare the temperature-related count value (TCLR.THCOUNT) to several thresholds (temperature levels) in order to determine the control values TCCR.TCC.



**Figure 8-31 Temperature Compensation Clock Generation**

The clock divider is programmed via bit field TCCR.TCDIV. The value that should be used for bit field TCCR.TCDIV can be calculated using the following formula documented in the data sheet.

Generally, temperature compensation is a user-controlled feature. The Temperature Compensation Control Register TCCR provides access to the actual compensation value (generated by the sensor) and allows software control of the pads. During operation the device (i.e. the pads) can be controlled by the value of the on-chip sensor, or by externally provided compensation values. Register TCCR also provides the programmable divider value.

*Note: The relation between the counter value and the temperature can differ between two devices and need to be evaluated for each device individually.*

**System Control Unit (SCU)**

*Note: The temperature compensation circuit does not generate temperature compensation values continuously. The idea is, that the SW frequently updates the pad control with the value currently found in the tempcomp register (e.g. by an interrupt generated by a timer). Since temperature is a continuous function it is not relevant, whether the temperature value read is new or the value of a previous measurement.*

## 8.11.1 Temperature Compensation Registers

### 8.11.1.1 TCCR

This register contains the control options.

#### TCCR

#### Temperature Compensation Control Register

**ESFR (F1AC<sub>H</sub>/D6<sub>H</sub>)**

**Reset Value: 0003<sub>H</sub>**

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

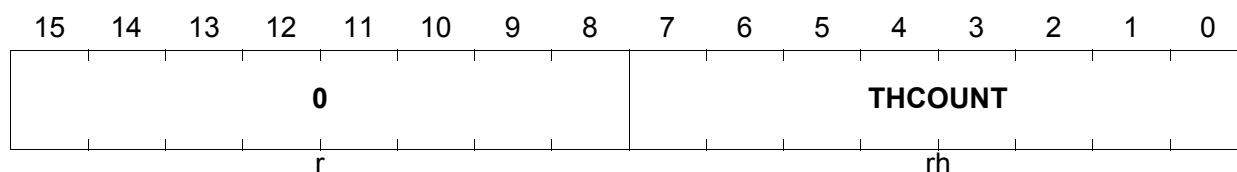
Field	Bits	Type	Description
<b>TCC</b>	[1:0]	rw	<b>Temperature Compensation Control</b> The value which controls the temperature compensation inputs of the pads. 00 <sub>B</sub> Maximum reduction = min. driver strength, i.e. very low temperature 11 <sub>B</sub> No reduction = max. driver strength, i.e. very high temperature
<b>TCDIV</b>	[6:2]	rw	<b>Temperature Compensation Clock Divider</b> This value adjusts the temperature compensation logic to the selected operating frequency.
<b>TCE</b>	7	rw	<b>Temperature Compensation Enable</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Enable counting to generate new temperature values. Clearing this bit also stops the temperature compensation oscillator.
<b>0</b>	[15:8]	r	<b>Reserved</b> Read as 0; should be written with 0.

**TCLR**

**Temperature Comp. Level Register**

**ESFR (F0AC<sub>H</sub>/56<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
<b>THCOUNT</b>	[7:0]	rh	<b>Threshold Counter</b> Returns the result of the most recent count cycle of the temperature sensor, to be compared with the thresholds.
<b>0</b>	[15:8]	r	<b>Reserved</b> Read as 0; should be written with 0.

*Note: The threshold counter will not overflow but rather stop at count 255.*

## 8.12 Watchdog Timer (WDT)

The following part describes the Watchdog Timer (WDT) and its functionality.

### 8.12.1 Introduction

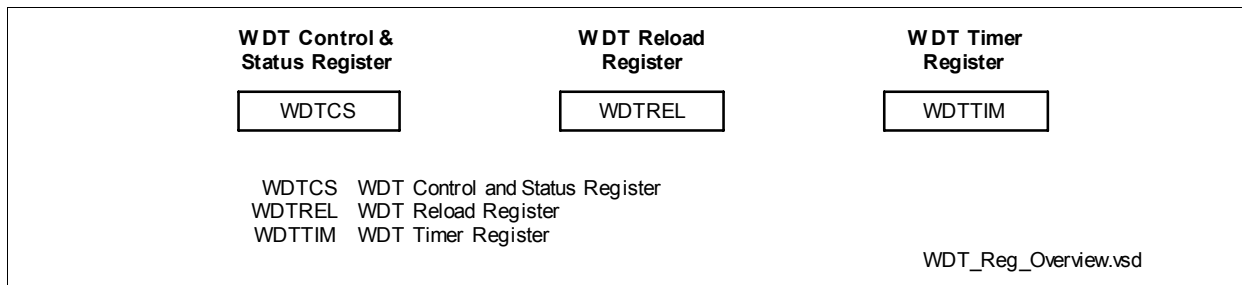
The Watchdog Timer (WDT) is a secure mechanism to overcome life- and dead-locks. An enabled WDT generates a reset for the system if not serviced in a configured time frame.

#### Features

The following list is a summary of the WDT functions:

- 16-bit Watchdog Timer
- Selectable operating frequency:  $f_{IN} / 256$  or  $f_{IN} / 16384$
- Timer overflow error detection
- Individual disable for timer functionality
- Double Reset Detection

**Figure 8-32** provides an overview on the registers of the Watchdog Timer.



**Figure 8-32 Watchdog Timer Register Overview**

### 8.12.2 Overview

The Watchdog Timer (WDT) provides a highly reliable and secure way to detect and recover from software or hardware failure. The WDT helps to abort an accidental malfunction of the XE16xyM in a user-specified time period. When enabled, the WDT will cause the XE16xyM system to be reset if the WDT is not serviced within a user-programmable time period. The CPU must service the WDT within this time interval to prevent the WDT from causing a WDT reset request trigger. Hence, regular service of the WDT confirms that the system is functioning properly.

A further feature of the Watchdog Timer is its reset prewarning operation. Instead of immediately resetting the device on the detection of an error, a prewarning output is given to the system via an interrupt request. This makes it possible to bring the system into a defined and predictable status, before the reset is finally issued.

## 8.12.3 Functional Description

The following part describes all functions of the WDT.

### 8.12.3.1 Timer Operation

The timer is enabled when instruction ENWDT (Enable Watchdog Timer) is executed correctly.

The WDT uses the input clock  $f_{IN}$  which is equal to the system clock  $f_{sys}$ . A clock divider in front of the WDT timer provides two output frequencies,  $f_{IN} / 256$  and  $f_{IN} / 16384$ . The selection of the counting rate is done via bit **WDTCS**.IR.

#### WDT Periods

The general formula to calculate a Watchdog period is:

$$\text{period} = \frac{(2^{16} - \text{startvalue}) \cdot 256 \cdot 2^{(1-IR) \cdot 6}}{f_{IN}} \quad (8.4)$$

The parameter <startvalue> represents either the user-programmable reload value WDTREL.RELV (default value  $FFFC_H$ ) for the calculation of the period in Normal Mode or the fixed value  $FFFF_H$  for the calculation of the period in Prewarning Mode.

#### WDT Timer Reload

The counter is reloaded and the prescaler is cleared when one of the following conditions occurs:

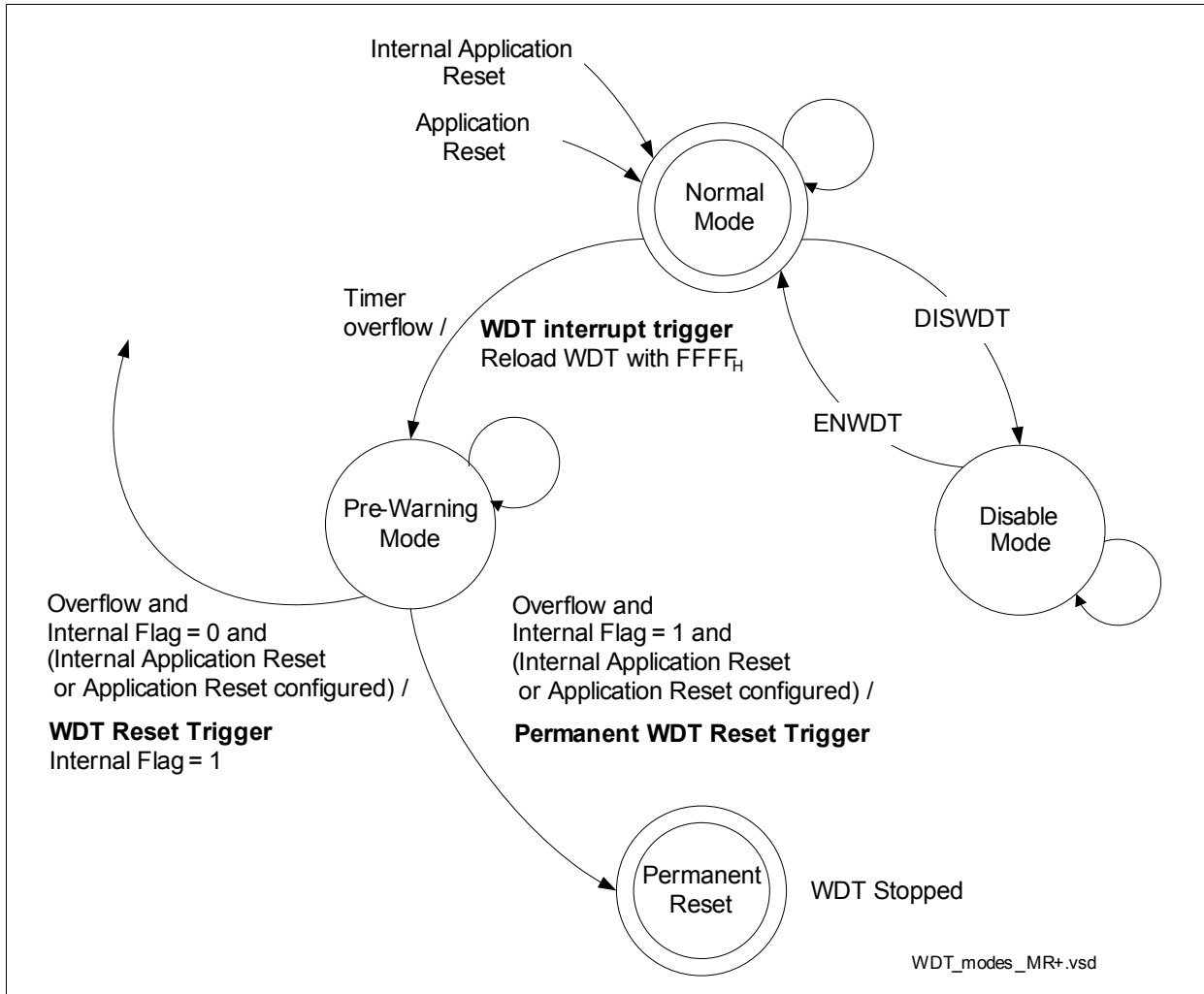
- A successful access to register **WDTREL**
- The WDT is serviced via instruction SRVWDT
- A WDT overflow condition (Prewarning Mode is entered).  
The different reload value for the counter in Prewarning Mode is  $FFFF_H$ .
- The Disable Mode is entered (when instruction DISWDT is executed)
- Upon any reset

### 8.12.3.2 Timer Modes

The Watchdog Timer provides following modes:

- Normal Mode
- Disable Mode
- Prewarning Mode

**Figure 8-33** provides a state diagram of the different Timer Modes and the transition possibilities. Please refer to the description of the conditions for changing from one mode to the other.



**Figure 8-33 State Diagram of the Timer Modes**

### Normal Mode

Normal Mode is the default mode after an Application Reset or an Internal Application Reset. Normal Mode can be entered from Disable Mode only when instruction ENWDT is executed.

The timer is loaded with RELV when the Normal Mode is entered, and it starts counting upwards. After reset the timer is loaded with FFFC<sub>H</sub> (default value of RELV).

It has to be serviced before the counter overflows. Servicing is performed by the CPU via instructions SRVWDT and/or ENWDT.

If the WDT is not serviced before the timer overflows, a system malfunction is assumed, and following operations are done:

- An WDT interrupt trigger request is issued
- Prewarning Mode is entered

- Timer is reloaded with  $FFFF_H$

### **Disable Mode**

Disable Mode is provided for applications that do not require the Watchdog Timer function. Disable Mode is entered when instruction DISWDT is executed, either before End-of-Init, if CPUCON1.WDTCTL = 0, or at any time, if CPUCON1.WDTCTL = 1.

The timer is reloaded with the value of WDTREL.RELV when Disable Mode is entered.

A transition from Disable Mode to Normal Mode is performed when instruction ENWDT is executed while CPUCON1.WDTCTL = 1.

### **Prewarning Mode**

Prewarning Mode is entered always when a Watchdog error is detected. This is an overflow in Normal Mode. Instead of immediately requesting a reset of the device, the WDT enables the system to enter a secure state by issuing the prewarning output before the reset occurs. Receiving the prewarning, the CPU and the system are requested to finish all pending transaction requests and to not generate new ones. The prewarning is signalled via an interrupt. The CPU can recognize the WDT prewarning interrupt via register **INTSTAT**. After finishing all pending transactions, the CPU should execute the IDLE instruction to stop all further processing before the coming reset.

In Prewarning Mode, the WDT starts counting from  $FFFF_H$  upwards, and then requests a WDT reset on the overflow of the WDT from  $FFFF_H$  to  $0000_H$ . A reset request of the type as configured in **RSTCON1.WDT** can not be avoided. No reset will be requested if **RSTCON1.WDT** is cleared. The WDT does not react anymore to accesses to its registers and to the ENWDT or DISWDT instruction, nor will it change its state until it is reset.

A further feature of the WDT detects double errors and sets the whole system into a permanent WDT reset. This feature prevents the XE16xyM from executing random wrong code for longer than the occurrence of the overflow, and prevents the XE16xyM from being repeatedly reset by the WDT.

### **Double WDT Reset**

If the Watchdog induced reset (Application or Internal Application Reset) occurs twice, a severe system malfunction is assumed and the XE16xyM is held in a reset of the type as configured in **RSTCON1.WDT** (or just not) until a Power-on Reset occurs. This prevents the device from being periodically reset if, for instance, connection to the external memory has been lost such that even system initialization could not be performed.

*Note: Triggering a PORST upon a WDT reset will never result in a double WDT overflow.*

If the WDT is configured by **RSTCON1.WDT** to request an Application Reset or an Internal Application Reset the second reset request will be permanently asserted



resulting (without any change in the reset configuration) in a permanent reset of the type configured by RSTCON1.WDT.

The information about the first WDT reset request is stored in an internal flag. The internal flag is set when a WDT overflow has occurred in Prewarning Mode and the reset request is generated. If the internal flag is already set then a double WDT reset event has occurred and a permanent reset request is generated.

This internal flag is cleared by any Power-on Reset or when bit **WDTCS.CLRIRF** is set. A correct service of the WDT does not clear this internal flag nor do any access to the WDT related registers or commands. Therefore, if correct WDT-servicing has been done after the first WDT reset and a next WDT reset must not immediately lead to a double error state, application software has to clear the internal flag.

*Note: Regarding the handling of the internal flag It does not matter whether a reset was generated on a WDT reset request or if the reset configuration was changed between the two reset requests.*

*Note: After the double WDT reset request trigger is generated the counter is stopped after the overflow.*

### **Port Configuration during WDT Reset**

The behavior of the ESRx ports can be defined with respect to the reset type by bit field ESRCFGx.PC. For the coding of PC see **Table 8-6**. This allows to signal the occurrence of a reset.

The configuration of the GPIOs ports depends on the reset type. In case of an Application Reset the pad configuration is unchanged, in case of an Internal Application Reset the ports are configured for input.

### **8.12.3.3 Suspend Mode Support**

In an enabled and active debug session, the Watchdog functionality can lead to unintended resets. Therefore, to avoid these resets, the OCDS can control whether the WDT is enabled or disabled (default after reset). This is done via bit CBS\_IOSR.DB.

**Table 8-11 OCDS Behavior of WDT**

<b>WDTCS.DS</b>	<b>CBS_DBGSR.DBGEN</b>	<b>CBS_IOSR.DB</b>	<b>WDT Action</b>
1	X	X	Stopped
0	0	X	Running
0	1	0	Stopped
0	1	1	Running

## 8.12.4 WDT Kernel Registers

### 8.12.4.1 WDT Reload Register

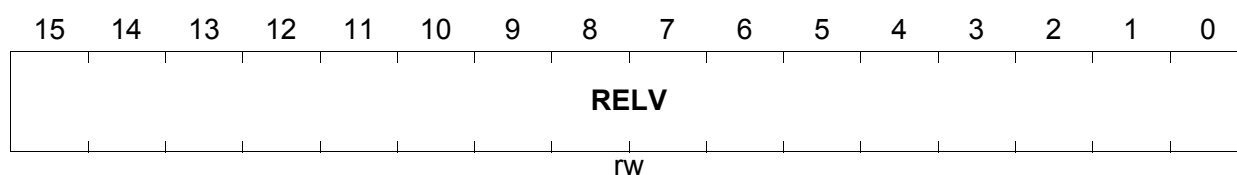
This register defines the WDT reload value.

#### WDTREL

**WDT Reload Register**

**ESFR (F0C8<sub>H</sub>/64<sub>H</sub>)**

**Reset Value: FFFC<sub>H</sub>**



Field	Bits	Type	Description
RELV	[15:0]	rw	<b>Reload Value for the Watchdog Timer</b> This bit field defines the reload value for the WDT.

### 8.12.4.2 WDT Control and Status Register

The Control and Status Register can only be accessed in Secured Mode.

#### WDTCS

#### WDT Control and Status Register

ESFR (F0C6 <sub>H</sub> /63 <sub>H</sub> )												Reset Value: 0000 <sub>H</sub>			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0						IR		0				CLR IRF	PR	DS	OE
r						rw		r				w	rh	rh	rh

Field	Bits	Type	Description
OE	0	rh	<b>Overflow Error Status Flag</b> 0 <sub>B</sub> No WDT overflow error 1 <sub>B</sub> A WDT overflow error has occurred. This bit is set by hardware when the Watchdog Timer overflows from FFFF <sub>H</sub> to 0000 <sub>H</sub> . This bit is only cleared through: <ul style="list-style-type: none"> <li>any Power-on Reset</li> <li>an executed SRVWDT or ENWDT instruction</li> </ul> <i>Note: The WDT is always enabled by ENWDT in the startup procedure (see section "Watchdog Timer handling"). Therefore, the bit is cleared in case of an Application Reset or an Internal Application Reset.</i>  <i>Note: It is not possible to clear this bit in Prewarning Mode with the SRVWDT or ENWDT instruction.</i>
DS	1	rh	<b>Timer Enable/Disable Status Flag</b> 0 <sub>B</sub> Timer is enabled (default after reset) 1 <sub>B</sub> Timer is disabled This bit is cleared when instruction ENWDT was executed and CPUCON1.WDTCTL = 1. This bit is set when instruction DISWDT was executed before EINIT or CPUCON1.WDTCTL = 1. <i>Note: ENWDT and DISWDT instruction will be reflected in this bit but in Prewarning Mode the WDT mode is not changed.</i>

**System Control Unit (SCU)**

Field	Bits	Type	Description
<b>PR</b>	2	rh	<b>Prewarning Mode Flag</b> $0_B$ Normal Mode (default after reset) $1_B$ Prewarning Mode
<b>CLRIRF</b>	3	w	<b>Clear Internal Reset Flag</b> This bit is used to request a clear of the internal flag storing the information about the first WDT reset request. $0_B$ No action $1_B$ Request to clear the internal flag <i>Note: The bit is always read as 0.</i>
<b>IR</b>	8	rw	<b>Input Frequency Request Bit</b> $0_B$ Request to set input frequency to $f_{IN} / 16384$ $1_B$ Request to set input frequency to $f_{IN} / 256$ An update of this bit is taken into account after the next successful execution of instruction SRVWDT or ENWDT, on a write to register WDTREL, and always when the WDT is in Disable Mode.
<b>0</b>	[7:4], [15:9]	r	<b>Reserved</b> Read as 0; should be written with 0.

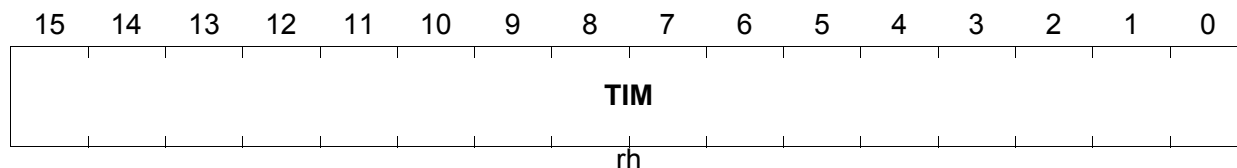
### 8.12.4.3 WDT Timer Register

#### WDTTIM

**WDT Timer Register**

**ESFR (F0CA<sub>H</sub>/65<sub>H</sub>)**

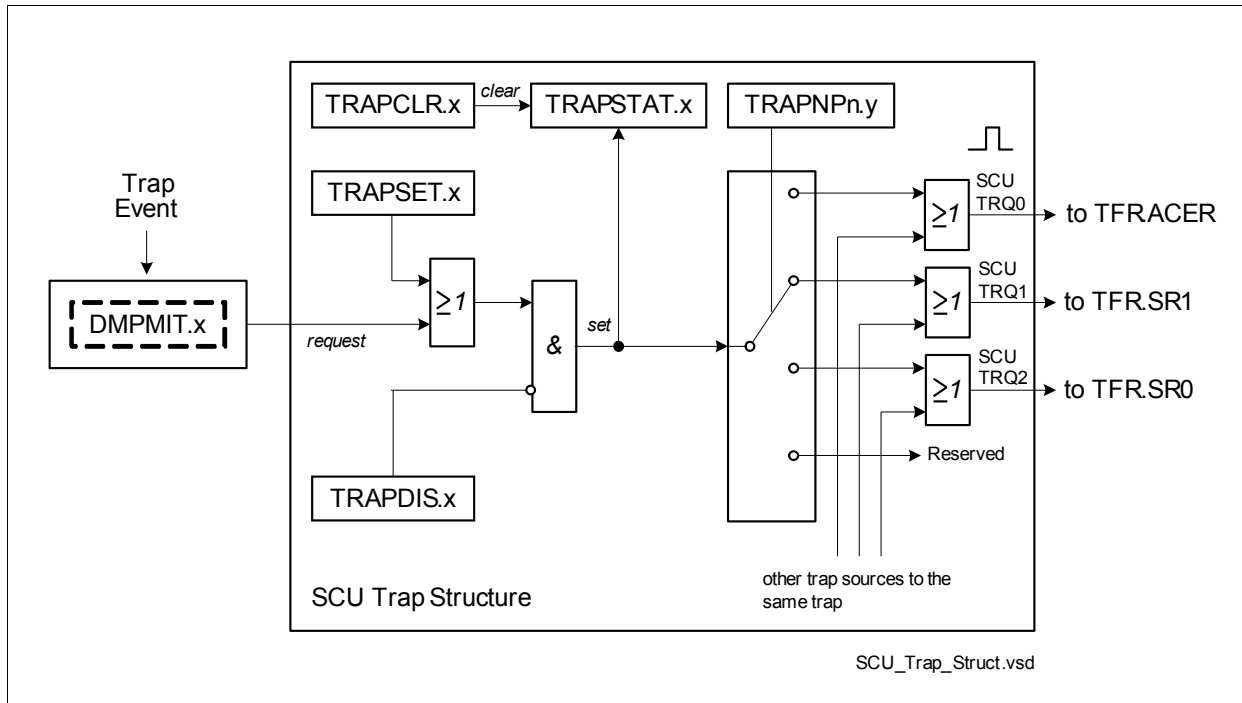
**Reset Value: FFFC<sub>H</sub>**



Field	Bits	Type	Description
TIM	[15:0]	rh	<b>Timer Value</b> Reflects the current contents of the Watchdog Timer.

## 8.13 SCU Trap Generation

The basic trap structure of the SCU is shown in [Figure 8-34](#).



**Figure 8-34 SCU Trap Structure**

If enabled by the corresponding bit in register [TRAPDIS](#), a trap is triggered either by a pulse on the incoming trap line, or by a software set of the respective bit in register [TRAPSET](#). The trigger sets the respective flag in register [TRAPSTAT](#) and is gated to one of the trap nodes, selected by the node pointer registers [TRAPNP](#) and [TRAPNP1](#). The trap flag in register [TRAPSTAT](#) can be cleared by software by writing to the corresponding bit in register [TRAPCLR](#).

If more than one trap source is connected to the same trap (via registers [TRAPNP](#) and [TRAPNP1](#)), the requests are combined to one common line.

### Trap Node Assignment

The trap sources of the system can be mapped to three trap nodes by programming the trap node pointer registers [TRAPNP](#) and [TRAPNP1](#). The default assignment of the trap sources to the nodes and their corresponding control register is listed in [Table 8-12](#).

#### 8.13.1 Trap Support

Some of the trap requests are first fed through a sticky flag register in the [DMP\\_M](#) domain. These flags are set with a trigger and if set trigger the trap generation in the [DMP\\_1](#).

## System Control Unit (SCU)

Which of the trap requests have a sticky flag in register DMPMIT is listed in [Table 8-12](#).

*Note: When servicing an SCU trap request, make sure that all related request flags are cleared after the identified request has been handled. To clear a trap request that is stored in register DMPMIT, first clear the request source of the source, clear the request within DMP\_M via DMPMITCLR, and then clear the request within DMP\_1 via TRAPCLR.*

### 8.13.2 SCU Trap Sources

The SCU receives the trap lines listed in [Table 8-12](#).

**Table 8-12 SCU Trap Request Overview**

Source of Trap	Short Name	Sticky Flag in DMPMIT	Default Trap Flag Assignment in Register TFR
Flash Access Trap	FAT	---	TFR.ACER (SCU_TRQ0)
$\overline{\text{ESR0}}$ Trap	ESR0T	yes	TFR.SR1 (SCU_TRQ1)
$\overline{\text{ESR1}}$ Trap	ESR1T	yes	TFR.SR1 (SCU_TRQ1)
$\overline{\text{ESR2}}$ Trap	ESR2T	yes	TFR.SR1 (SCU_TRQ1)
PLL Trap	OSCWDTT	---	TFR.SR0 (SCU_TRQ1)
Register Access Trap	RAT	yes	TFR.ACER (SCU_TRQ0)
Parity Error Trap	PET	---	TFR.ACER (SCU_TRQ0)
VCO Lock Trap	VCOLCKT	---	TFR.SR0 (SCU_TRQ2)
ECC Error Trap	ECCT	---	TFR.ACER (SCU_TRQ0)

### 8.13.3 SCU Trap Control Registers

#### 8.13.3.1 Register TRAPSTAT

This register contains the status flags for all trap request trigger sources of the SCU.  
 For setting and clearing of these status bits by software see registers TRAPSET and TRAPCLR, respectively.

##### TRAPSTAT

Trap Status Register

SFR (FF02<sub>H</sub>/81<sub>H</sub>)

Reset Value: 0000<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0							ECC T	VCO LCK T	PE T	RA T	OSC WDT T	ESR 2 T	ESR 1 T	ESR 0 T	FA T
r							rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
FAT	0	rh	<b>Flash Access Trap Request Flag</b> TRAPSTAT.FAT is set when a flash access violation occurs and TRAPDIS.FAT = 0. 0 <sub>B</sub> No FA trap trigger has occurred since this bit was cleared the last time 1 <sub>B</sub> A FA trap trigger has occurred since this bit was cleared the last time
ESR0T	1	rh	<b>ESR0 Trap Request Flag</b> TRAPSTAT.ESR0T is set when bit DMPMIT.ESR0T is set and TRAPDIS.ESR0T = 0. 0 <sub>B</sub> No ESR0 trap trigger has occurred since this bit was cleared the last time 1 <sub>B</sub> An ESR0 trap trigger has occurred since this bit was cleared the last time
ESR1T	2	rh	<b>ESR1 Trap Request Flag</b> TRAPSTAT.ESR1T is set when bit DMPMIT.ESR1T is set and TRAPDIS.ESR1T = 0. 0 <sub>B</sub> No ESR1 trap trigger has occurred since this bit was cleared the last time 1 <sub>B</sub> An ESR1 trap trigger has occurred since this bit was cleared the last time



**System Control Unit (SCU)**

<b>Field</b>	<b>Bits</b>	<b>Type</b>	<b>Description</b>
<b>ESR2T</b>	3	rh	<b>ESR2 Trap Request Flag</b> TRAPSTAT.ESR2T is set when bit DMPMIT.ESR0T is set and TRAPDIS.ESR2T = 0. 0 <sub>B</sub> No ESR2 trap trigger has occurred since this bit was cleared the last time 1 <sub>B</sub> An ESR2 trap trigger has occurred since this bit was cleared the last time
<b>OSCWDTT</b>	4	rh	<b>OSCWDT Trap Request Flag</b> TRAPSTAT.OSCWDTT is set when an OSCWDT emergency event occurs and TRAPDIS.OSCWDTT = 0. 0 <sub>B</sub> No OSCWDT trap trigger has occurred since this bit was cleared the last time 1 <sub>B</sub> An OSCWDT trap trigger has occurred since this bit was cleared the last time
<b>RAT</b>	5	rh	<b>Register Access Trap Request Flag</b> TRAPSTAT.RAT is set when bit DMPMIT.RAT is set and TRAPDIS.RAT = 0. 0 <sub>B</sub> No RA trap trigger has occurred since this bit was cleared the last time 1 <sub>B</sub> A RA trap trigger has occurred since this bit was cleared the last time
<b>PET</b>	6	rh	<b>Parity Error Trap Request Flag</b> TRAPSTAT.PET is set when a memory parity error occurs and TRAPDIS.PET = 0. 0 <sub>B</sub> No PE trap trigger has occurred since this bit was cleared the last time 1 <sub>B</sub> A PE trap trigger has occurred since this bit was cleared the last time
<b>VCOLCKT</b>	7	rh	<b>VCOLCK Trap Request Flag</b> TRAPSTAT.VCOLCKT is set when a VCOLCK emergency event occurs and TRAPDIS.VCOLCKT = 0. 0 <sub>B</sub> No VCOLCK trap trigger has occurred since this bit was cleared the last time 1 <sub>B</sub> A VCOLCK trap trigger has occurred since this bit was cleared the last time

Field	Bits	Type	Description
ECCT	8	rh	<b>ECC Error Trap Request Flag</b> TRAPSTAT.ECCT is set when a memory ECC error occurs and TRAPDIS.ECCT = 0. 0 <sub>B</sub> No ECC trap trigger has occurred since this bit was cleared the last time 1 <sub>B</sub> An ECC trap trigger has occurred since this bit was cleared the last time
0	[15:9]	r	<b>Reserved</b> Read as 0; should be written with 0.

### 8.13.3.2 Register TRAPCLR

This register contains the software clear control for the trap status flags in register TRAPSTAT. Clearing a bit in this register has no effect, reading a bit always returns zero.

#### TRAPCLR

**Trap Clear Register**

**SFR (FE8E<sub>H</sub>/47<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0							<b>ECC</b>	<b>VCO</b>	<b>PE</b>	<b>RA</b>	<b>OSC</b>	<b>ESR</b>	<b>ESR</b>	<b>ESR</b>	<b>FA</b>
							<b>T</b>	<b>LCK</b>	<b>T</b>	<b>T</b>	<b>WDT</b>	<b>2</b>	<b>1</b>	<b>0</b>	<b>T</b>
							w	w	w	w	w	w	w	w	w

Field	Bits	Type	Description
<b>FAT</b>	0	w	<b>Clear Flash Access Trap Request Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Flag TRAPSTAT.FAT is cleared
<b>ESR0T</b>	1	w	<b>Clear ESR0 Trap Request Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Flag TRAPSTAT.ESR0T is cleared
<b>ESR1T</b>	2	w	<b>Clear ESR1 Trap Request Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Flag TRAPSTAT.ESR1T is cleared
<b>ESR2T</b>	3	w	<b>Clear ESR2 Trap Request Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Flag TRAPSTAT.ESR2T is cleared
<b>OSCWDTT</b>	4	w	<b>Clear OSCWDT Trap Request Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Flag TRAPSTAT.OSCWDTT is cleared
<b>RAT</b>	5	w	<b>Clear Register Access Trap Request Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Flag TRAPSTAT.RAT is cleared
<b>PET</b>	6	w	<b>Clear Parity Error Access Trap Request Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Flag TRAPSTAT.PET is cleared
<b>VCOLCKT</b>	7	w	<b>Clear VCOLCK Trap Request Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Flag TRAPSTAT.VCOLCKT is cleared

Field	Bits	Type	Description
ECCT	8	w	<b>Clear ECC Error Trap Request Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Flag TRAPSTAT.ECCT is cleared
0	[15:9]	r	<b>Reserved</b> Read as 0; should be written with 0

*Note: These bits are always read as 0.*

### 8.13.3.3 Register TRAPSET

This register contains the software set control for the trap status flags in register TRAPSTAT. Clearing a bit in this register has no effect, reading a bit always returns zero.

#### TRAPSET

Trap Set Register

SFR (FE8C<sub>H</sub>/46<sub>H</sub>)

Reset Value: 0000<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0							ECC T	VCO LCK T	PE T	RA T	OSC WDT T	ESR 2 T	ESR 1 T	ESR 0 T	FA T
r							w	w	w	w	w	w	w	w	w

Field	Bits	Type	Description
FAT	0	w	<b>Set Flash Access Trap Request Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Flag TRAPSTAT.FAT is set
ESR0T	1	w	<b>Set ESR0 Trap Request Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Flag TRAPSTAT.ESR0T is set
ESR1T	2	w	<b>Set ESR1 Trap Request Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Flag TRAPSTAT.ESR1T is set
ESR2T	3	w	<b>Set ESR2 Trap Request Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Flag TRAPSTAT.ESR2T is set
OSCWDTT	4	w	<b>Set OSCWDT Trap Request Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Flag TRAPSTAT.OSCWDTT is set
RAT	5	w	<b>Set Register Access Trap Request Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Flag TRAPSTAT.RAT is set
PET	6	w	<b>Set Parity Error Access Trap Request Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Flag TRAPSTAT.PET is set
VCOLCKT	7	w	<b>Set VCOLCK Trap Request Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Flag TRAPSTAT.VCOLCKT is set

Field	Bits	Type	Description
ECCT	8	w	<b>Set ECC Error Trap Request Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Flag TRAPSTAT.ECCT is set
0	[15:9]	r	<b>Reserved</b> Read as 0; should be written with 0.

*Note: These bits are always read as 0.*

### 8.13.3.4 Register TRAPDIS

This register contains the software disable control for all trap request trigger sources. Note that the bits ESRxT and RAT in this register also disable the setting of the respective flags in register DMPMIT (see [Section 8.10.1](#)).

#### TRAPDIS

**Trap Disable Register**

**SFR (FE90<sub>H</sub>/48<sub>H</sub>)**

**Reset Value: 009E<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
			0				ECC T	VCO LCK T	PE T	RA T	OSC WDT T	ESR 2 T	ESR 1 T	ESR 0 T	FA T
			r				rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
<b>FAT</b>	0	rw	<b>Disable Flash Access Trap Request</b> 0 <sub>B</sub> FA trap request enabled 1 <sub>B</sub> FA trap request disabled
<b>ESR0T</b>	1	rw	<b>Disable ESR0 Trap Request</b> 0 <sub>B</sub> ESR0 trap request enabled 1 <sub>B</sub> ESR0 trap request disabled
<b>ESR1T</b>	2	rw	<b>Disable ESR1 Trap Request</b> 0 <sub>B</sub> ESR1 trap request enabled 1 <sub>B</sub> ESR1 trap request disabled
<b>ESR2T</b>	3	rw	<b>Disable ESR2 Trap Request</b> 0 <sub>B</sub> ESR2 trap request enabled 1 <sub>B</sub> ESR2 trap request disabled
<b>OSCWDTT</b>	4	rw	<b>Disable OSCWDT Trap Request</b> 0 <sub>B</sub> OSCWDT trap request enabled 1 <sub>B</sub> OSCWDT trap request disabled
<b>RAT</b>	5	rw	<b>Disable Register Access Trap Request</b> 0 <sub>B</sub> RA trap request enabled 1 <sub>B</sub> RA trap request disabled
<b>PET</b>	6	rw	<b>Disable Parity Error Trap Request</b> 0 <sub>B</sub> PE trap request enabled 1 <sub>B</sub> PE trap request disabled
<b>VCOLCKT</b>	7	rw	<b>Disable VCOLCK Trap Request</b> 0 <sub>B</sub> VCOLCK trap request enabled 1 <sub>B</sub> VCOLCK trap request disabled

Field	Bits	Type	Description
<b>ECCT</b>	8	rw	<b>Disable ECC Error Trap Request</b> 0 <sub>B</sub> ECC trap request enabled 1 <sub>B</sub> ECC trap request disabled
<b>0</b>	[15:9]	r	<b>Reserved</b> Read as 0; should be written with 0.



### 8.13.3.5 Register TRAPNP and TRAPNP1

These register contain the control for the trap node pointers of all SCU trap request trigger sources.

#### TRAPNP

**Trap Node Pointer Register**

**SFR (FE92<sub>H</sub>/49<sub>H</sub>)**

**Reset Value: 8254<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>VCOLCK</b>		<b>PE</b>		<b>RA</b>		<b>OSCWDT</b>		<b>ESR2</b>		<b>ESR1</b>		<b>ESR0</b>		<b>FA</b>	
rw		rw		rw		rw		rw		rw		rw		rw	

Field	Bits	Type	Description
<b>FA</b>	[1:0]	rw	<b>Trap Node Pointer for Flash Access Traps</b> TRAPNP.FA selects the trap request output for an enabled FAT trap request. 00 <sub>B</sub> Select request output SCU_TRQ0 (TFR.ACER) 01 <sub>B</sub> Select request output SCU_TRQ1 (TFR.SR1) 10 <sub>B</sub> Select request output SCU_TRQ2 (TFR.SR0) 11 <sub>B</sub> Reserved, do not use this combination
<b>ESR0</b>	[3:2]	rw	<b>Trap Node Pointer for ESR0 Traps</b> TRAPNP.ESR0 selects the trap request output for an enabled ESR0 trap request. 00 <sub>B</sub> Select request output SCU_TRQ0 (TFR.ACER) 01 <sub>B</sub> Select request output SCU_TRQ1 (TFR.SR1) 10 <sub>B</sub> Select request output SCU_TRQ2 (TFR.SR0) 11 <sub>B</sub> Reserved, do not use this combination
<b>ESR1</b>	[5:4]	rw	<b>Trap Node Pointer for ESR1 Traps</b> TRAPNP.ESR1 selects the trap request output for an enabled ESR1 trap request. 00 <sub>B</sub> Select request output SCU_TRQ0 (TFR.ACER) 01 <sub>B</sub> Select request output SCU_TRQ1 (TFR.SR1) 10 <sub>B</sub> Select request output SCU_TRQ2 (TFR.SR0) 11 <sub>B</sub> Reserved, do not use this combination

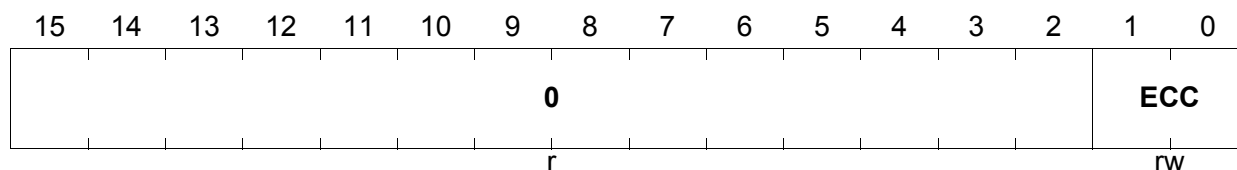
Field	Bits	Type	Description
<b>ESR2</b>	[7:6]	rw	<b>Trap Node Pointer for ESR2 Traps</b> TRAPNP.ESR2 selects the trap request output for an enabled ESR2 trap request. 00 <sub>B</sub> Select request output SCU_TRQ0 (TFR.ACER) 01 <sub>B</sub> Select request output SCU_TRQ1 (TFR.SR1) 10 <sub>B</sub> Select request output SCU_TRQ2 (TFR.SR0) 11 <sub>B</sub> Reserved, do not use this combination
<b>OSCWDT</b>	[9:8]	rw	<b>Trap Node Pointer for OSCWDT Traps</b> TRAPNP.OSCWDT selects the trap request output for an enabled OSCWDT trap request. 00 <sub>B</sub> Select request output SCU_TRQ0 (TFR.ACER) 01 <sub>B</sub> Select request output SCU_TRQ1 (TFR.SR1) 10 <sub>B</sub> Select request output SCU_TRQ2 (TFR.SR0) 11 <sub>B</sub> Reserved, do not use this combination
<b>RA</b>	[11:10]	rw	<b>Trap Node Pointer for Register Access Traps</b> TRAPNP.RA selects the trap request output for an enabled RAT trap request. 00 <sub>B</sub> Select request output SCU_TRQ0 (TFR.ACER) 01 <sub>B</sub> Select request output SCU_TRQ1 (TFR.SR1) 10 <sub>B</sub> Select request output SCU_TRQ2 (TFR.SR0) 11 <sub>B</sub> Reserved, do not use this combination
<b>PE</b>	[13:12]	rw	<b>Trap Node Pointer for Parity Error Traps</b> TRAPNP.PE selects the trap request output for an enabled PET trap request. 00 <sub>B</sub> Select request output SCU_TRQ0 (TFR.ACER) 01 <sub>B</sub> Select request output SCU_TRQ1 (TFR.SR1) 10 <sub>B</sub> Select request output SCU_TRQ2 (TFR.SR0) 11 <sub>B</sub> Reserved, do not use this combination
<b>VCOLCK</b>	[15:14]	rw	<b>Trap Node Pointer for VCOLCK Traps</b> TRAPNP.VCOLCK selects the trap request output for an enabled VCOLCK trap request. 00 <sub>B</sub> Select request output SCU_TRQ0 (TFR.ACER) 01 <sub>B</sub> Select request output SCU_TRQ1 (TFR.SR1) 10 <sub>B</sub> Select request output SCU_TRQ2 (TFR.SR0) 11 <sub>B</sub> Reserved, do not use this combination

**TRAPNP1**

**Trap Node Pointer 1 Register**

**SFR (FE94<sub>H</sub>/4A<sub>H</sub>)**

**Reset Value:0000<sub>H</sub>**



Field	Bits	Type	Description
ECC	[1:0]	rw	<b>Trap Node Pointer for ECC Error Traps</b> TRAPNP.ECC selects the trap request output for an enabled ECCT trap request. 00 <sub>B</sub> Select request output SCU_TRQ0 (TFR.ACER) 01 <sub>B</sub> Select request output SCU_TRQ1 (TFR.SR1) 10 <sub>B</sub> Select request output SCU_TRQ2 (TFR.SR0) 11 <sub>B</sub> Reserved, do not use this combination
0	[15:2]	r	<b>Reserved</b> Read as 0; should be written with 0.

## 8.14 Memory Content Protection for RAM Areas

For supervising the content of the on-chip RAM areas (Flash memory is not considered here) two mechanisms are provided:

- Error Correction Control (ECC)
- Parity Checking

For each piece of data written to a RAM area the corresponding protection bits (parity or ECC bits) are generated and stored along with the user data.

The ECC logic supports single error detection (SED) and single error correction (SEC). The FlexRay module also supports double error detection (DED).

For the standard RAM areas, register MCHKCON selects the intended memory protection mode, while RAM areas embedded into peripheral modules provide a fixed protection mode. **Table 8-13** summarizes the available modes.

**Table 8-13 Available Memory Protection Mechanisms**

Memory	Parity	ECC Error Detection	ECC Error Correction	ECC code
Program SRAM (PSRAM)	yes	SED	SEC	4 bits per byte
Data SRAM (DSRAM)	yes	SED	SEC	4 bits per byte
Dual Port SRAM (DPRAM)	yes	SED	SEC	4 bits per byte
Standby RAM (SBRAM)	yes	SED	SEC	4 bits per byte
USICx SRAM (UxRAM)	yes	No	No	-
MultiCAN SRAM (MCRAM)	no	SED	SEC	7 bits per 32 bits

*Note: Memory Content Protection with Parity or ECC is disabled by default.*

*The intended protection mode must be selected and enabled before the respective RAM area can be used (see [Section 8.14.1](#)).*

The subsequent handling of trap requests is described in section [Chapter 8.13](#).

### **8.14.1 Protection Mode Selection**

After a power-on reset, memory content protection is disabled. The intended protection mode must be selected and enabled before the respective RAM area can be used. Either Parity or ECC protection can be selected for each standard RAM area.

Register **MCHKCON** selects the intended protection mode for the standard RAM areas. By default the ECC protection mode is selected, but not enabled.

#### **ECC Protection Mode**

With ECC protection mode selected, the ECC logic generates additional ECC bits which are stored along with each piece of data which is written to the selected RAM area.

Register **ECCCON** enables the checking of previously stored ECC bits.

If enabled, single-bit errors are detected during read operations (SED). These single-bit errors are automatically corrected before sending data to the CPU (SEC). An error is indicated in register **ECCSTAT**.

If disabled, read data are sent to the CPU unchanged and no error flags are set.

#### **Parity Protection Mode**

With Parity protection mode selected, the parity logic generates additional parity bits which are stored along with each piece of data which is written to the selected RAM area.

Register **PEEN** enables the checking of previously stored parity bits.

If enabled, parity errors are detected and indicated in register **PECON**. Read data are not modified.

#### **Protection Mode Enabling**

To activate RAM content protection, the protection mode must first be selected (register **MCHKCON**).

After a protection mode has been configured for a RAM area, that RAM must be initialized so the correct protection bits are generated and the RAM can be read without error.

CPU memory accesses may produce speculative read operations, in which data is read in advance, in anticipation of its actual use. In some cases, this may result in data accesses outside of the RAM memory regions actually used in the application. For example, the autoincrement mode of the SBRAM interface will access the new location after incrementing the pointer. Therefore, it is strongly recommended to initialize all content protected RAM before use, to avoid unexpected errors.

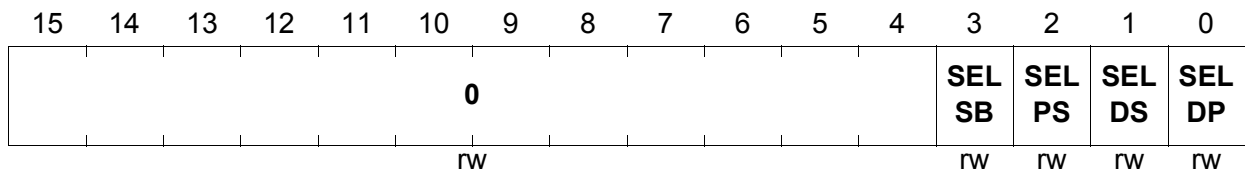
*Note: The sequence to activate one of these mechanisms is described in section "Preparing to activate Memory Content Protection."*

**MCHKCON**

**Memory Checking Control Register**

**ESFR (F0DC<sub>H</sub>/6E<sub>H</sub>)**

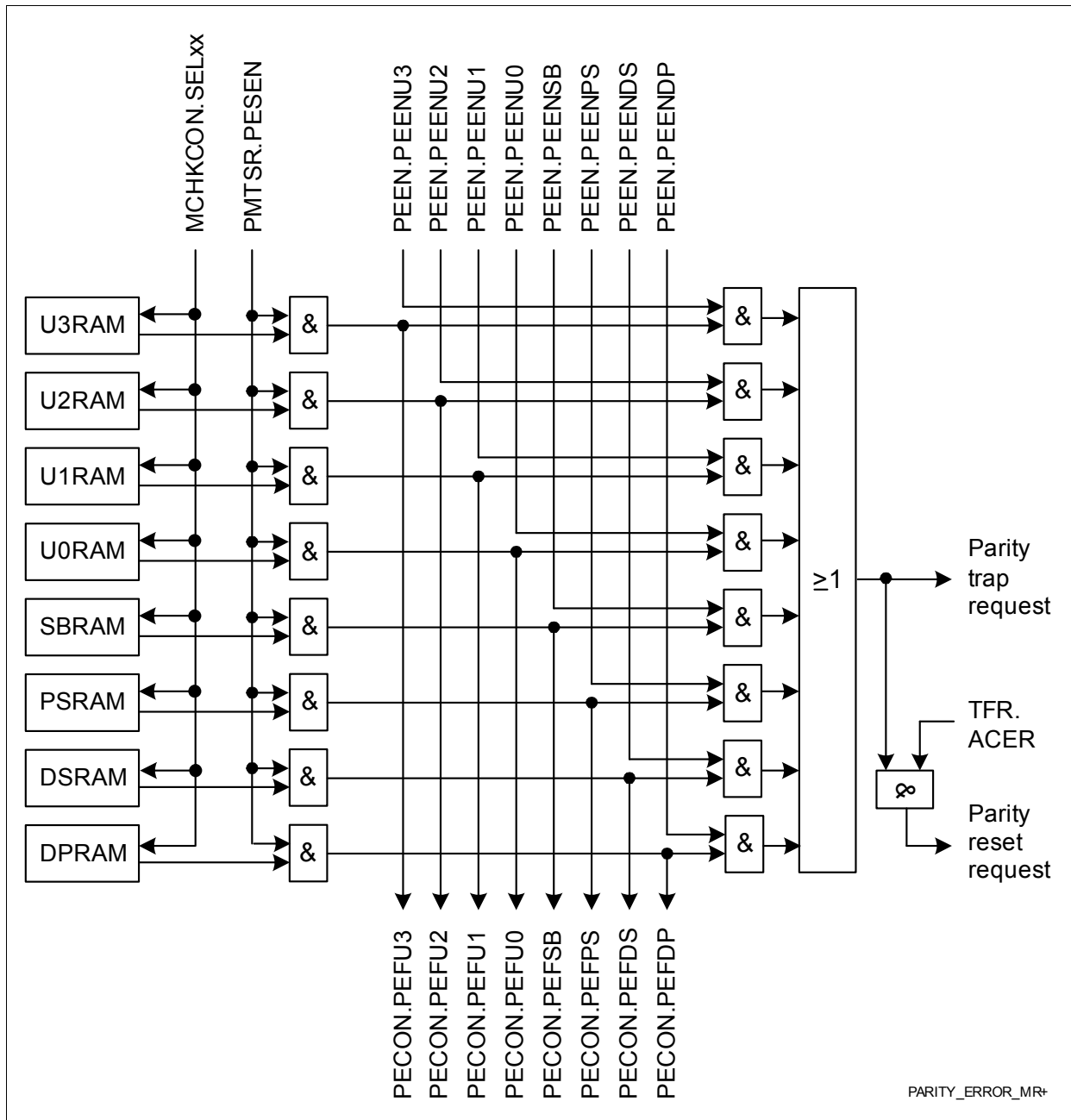
**Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
<b>SELDP</b>	0	rw	<b>Select Protection Mode for Dual Port Memory</b> 0 <sub>B</sub> ECC mode is selected for DPRAM 1 <sub>B</sub> Parity mode is selected for DPRAM
<b>SELDS</b>	1	rw	<b>Select Protection Mode for Data SRAM</b> 0 <sub>B</sub> ECC mode is selected for DSRAM 1 <sub>B</sub> Parity mode is selected for DSRAM
<b>SELPS</b>	2	rw	<b>Select Protection Mode for Program SRAM</b> 0 <sub>B</sub> ECC mode is selected for PSRAM 1 <sub>B</sub> Parity mode is selected for PSRAM
<b>SELSB</b>	3	rw	<b>Select Protection Mode for Standby Memory</b> 0 <sub>B</sub> ECC mode is selected for SBRAM 1 <sub>B</sub> Parity mode is selected for SBRAM
<b>0</b>	[15:4]	rw	<b>Reserved</b> Should be written with 0.

### 8.14.2 Parity Error Handling

During write operations parity information is generated and stored along with the data. During read operations this parity information is checked and in case of an error a trap request is generated (register PECON), if enabled. The requests from all RAMs can be combined (register PEEN) and trigger a trap via bit PET in register **TRAPSTAT**.



**Figure 8-35 Parity Error Control Logic**

If a parity error is detected while the trap flag TFR.ACER is set, i.e. during the execution of the associated trap handler routine, a reset request trigger is generated. This is

because a second error trap would activate the same handler and, therefore, cannot be handled by the CPU.

*Note: The parity trap trigger should activate the Access Error trap (ACER) to support this feature.*

### **8.14.2.1 Parity Software Testing Support**

To support testing algorithms for the parity error trap routines a memory parity test logic is implemented for the standard RAM areas (PSRAM, DSRAM, DPRAM, SBRAM).

This logic is controlled by registers PMTPR and PMTSR. If enabled by the respective bit MTEx in register PMTSR, a parity value can be written to any address of the corresponding RAM area through bitfield PWR in register PMTPR. With each read access from that area the parity from the memory parity control is stored in bitfield PRD of register PMTPR.

**Table 8-14** lists the valid bits in register PMTPR depending on the memory width.

**Table 8-14 Valid Parity Test Bits**

<b>Memory</b>	<b>Number of Parity Bits</b>	<b>Valid Bits in PWR/PRD</b>
Dual Port (DP) Memory	2	PWR[1:0]/PRD[9:8]
Data SRAM (DS) Memory	2	PWR[1:0]/PRD[9:8]
Program SRAM (PS) Memory	8	PWR[7:0]/PRD[15:8]
Standby RAM (SB) Memory	2	PWR[1:0]/PRD[9:8]

Test software should be located in external memory and should be written in a way that no pre-fetching is performed.

Test software should be located in external memory and should be written in a way that no pre-fetching is performed.



### 8.14.2.2 Parity Error Registers

Register PEEN enables the functional parity check mechanism for each RAM area separately.

*Note: Bit PESEN in register PMTSR globally enables the parity mechanism.*

#### PEEN

**Parity Error Enable Register      ESFR (F0C4<sub>H</sub>/41<sub>H</sub>)      Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0							PE EN SB	0	PE EN U3	PE EN U2	PE EN U1	PE EN U0	PE EN PS	PE EN DS	PE EN DP
rw							rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
PEENDP	0	rw	<b>Parity Error Trap Enable for Dual Port Memory</b> 0 <sub>B</sub> Disabled 1 <sub>B</sub> Trap requested for DPRAM parity errors
PEENDS	1	rw	<b>Parity Error Trap Enable for Data SRAM</b> 0 <sub>B</sub> Disabled 1 <sub>B</sub> Trap requested for DSRAM parity errors
PEENPS	2	rw	<b>Parity Error Trap Enable for Program SRAM</b> 0 <sub>B</sub> Disabled 1 <sub>B</sub> Trap requested for PSRAM parity errors
PEENU0	3	rw	<b>Parity Error Trap Enable for USIC0 Memory</b> 0 <sub>B</sub> Disabled 1 <sub>B</sub> Trap requested for USIC0 memory parity errors
PEENU1	4	rw	<b>Parity Error Trap Enable for USIC1 Memory</b> 0 <sub>B</sub> Disabled 1 <sub>B</sub> Trap requested for USIC1 memory parity errors
PEENU2	5	rw	<b>Parity Error Trap Enable for USIC2 Memory</b> 0 <sub>B</sub> Disabled 1 <sub>B</sub> Trap requested for USIC2 memory parity errors

**System Control Unit (SCU)**

Field	Bits	Type	Description
<b>PEENU3</b>	6	rw	<b>Parity Error Trap Enable for USIC3 Memory</b> 0 <sub>B</sub> Disabled 1 <sub>B</sub> Trap requested for USIC3 memory parity errors
<b>PEENSB</b>	8	rw	<b>Parity Error Trap Enable for Standby Memory</b> 0 <sub>B</sub> Disabled 1 <sub>B</sub> Trap requested for SBRAM parity errors
<b>0</b>	7, [15:9]	rw	<b>Reserved</b> Should be written with 0.

**System Control Unit (SCU)**

Register PECON controls the functional parity check mechanism.

If enabled the corresponding error flag is set upon the detection of a parity error in the associated RAM area. Otherwise, there is no indication.

Software can clear an error flag by writing 1 to the flag. Writing 0 has no effect.

**PECON**

**Parity Error Control Register    ESFR (F0DA<sub>H</sub>/6D<sub>H</sub>)                      Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
			<b>0</b>				<b>PEF SB</b>	<b>0</b>	<b>PEF U3</b>	<b>PEF U2</b>	<b>PEF U1</b>	<b>PEF U0</b>	<b>PEF PS</b>	<b>PEF DS</b>	<b>PEF DP</b>
			rwh				rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
<b>PEFDP</b>	0	rwh	<b>Parity Error Flag for Dual Port Memory</b> 0 <sub>B</sub> No DPRAM error 1 <sub>B</sub> A Parity error is indicated and can trigger a trap request trigger, if enabled for DPRAM
<b>PEFDS</b>	1	rwh	<b>Parity Error Flag for Data SRAM</b> 0 <sub>B</sub> No DSRAM error 1 <sub>B</sub> A Parity error is indicated and can trigger a trap request trigger, if enabled for DSRAM
<b>PEFPS</b>	2	rwh	<b>Parity Error Flag for Program SRAM</b> 0 <sub>B</sub> No PSRAM error 1 <sub>B</sub> A Parity error is indicated and can trigger a trap request trigger, if enabled for PSRAM
<b>PEFU0</b>	3	rwh	<b>Parity Error Flag for USIC0 Memory</b> 0 <sub>B</sub> No USIC0 memory error 1 <sub>B</sub> A Parity error is indicated and can trigger a trap request trigger, if enabled for USIC0 memory
<b>PEFU1</b>	4	rwh	<b>Parity Error Flag for USIC1 Memory</b> 0 <sub>B</sub> No USIC1 memory error 1 <sub>B</sub> A Parity error is indicated and can trigger a trap request trigger, if enabled for USIC1 memory
<b>PEFU2</b>	5	rwh	<b>Parity Error Flag for USIC2 Memory</b> 0 <sub>B</sub> No USIC2 memory error 1 <sub>B</sub> A Parity error is indicated and can trigger a trap request trigger, if enabled for USIC2 memory

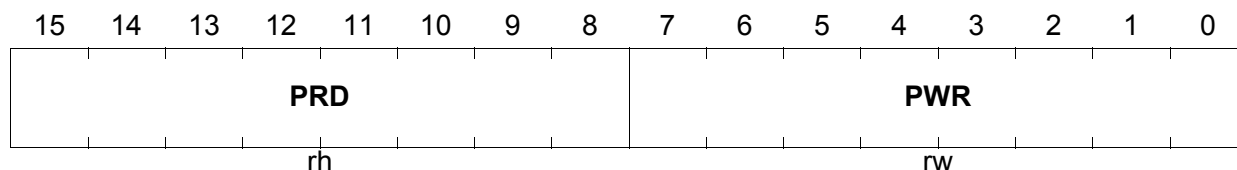
Field	Bits	Type	Description
<b>PEFU3</b>	6	rwh	<b>Parity Error Flag for USIC3 Memory</b> 0 <sub>B</sub> No USIC3 memory error 1 <sub>B</sub> A Parity error is indicated and can trigger a trap request trigger, if enabled for USIC3 memory
<b>PEFSB</b>	8	rwh	<b>Parity Error Flag for Standby Memory</b> 0 <sub>B</sub> No SBRAM error 1 <sub>B</sub> A Parity error is indicated and can trigger a trap request trigger, if enabled for Standby memory
<b>0</b>	7, [15:9]	rwh	<b>Reserved</b> Should be written with 0.

**PMTPR**

**Parity Memory Test Pattern Register**

**ESFR (F0E4<sub>H</sub>/72<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
<b>PRD</b>	[15:8]	rh	<b>Parity Read Values for Memory Test</b> For each byte of a memory module the parity bits generated during the most recent read access are indicated here.
<b>PWR</b>	[7:0]	rw	<b>Parity Write Values for Memory Test</b> For each byte of a memory module the parity bits corresponding to the next write access are stored here.

**PMTSR**

**Parity Memory Test Select Register**

**ESFR (F0E6<sub>H</sub>/73<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>PES EN</b>			<b>0</b>				<b>MT EN SB</b>			<b>0</b>			<b>MT EN PS</b>	<b>MT EN DS</b>	<b>MT EN DP</b>
rw			r				rw			rw			rw	rw	rw

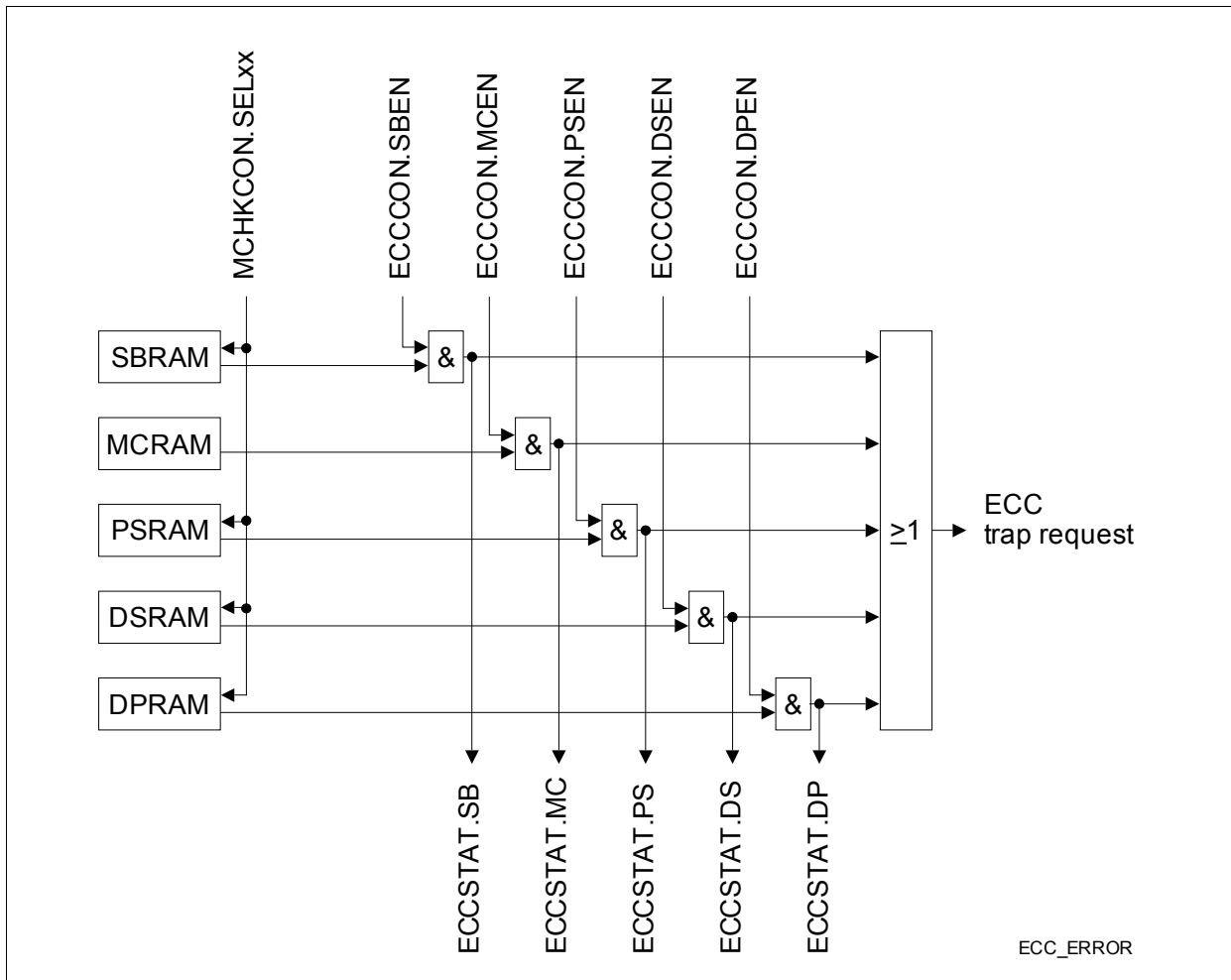
Field	Bits	Type	Description
<b>MTENDP</b>	0	rw	<b>Memory Test Enable Control for Dual Port Memory</b> Controls the test multiplexer for the DPRAM. 0 <sub>B</sub> Standard operation 1 <sub>B</sub> Test parity bits used (from PMTPR)
<b>MTENDS</b>	1	rw	<b>Memory Test Enable Control for Data SRAM</b> Controls the test multiplexer for the DSRAM. 0 <sub>B</sub> Standard operation 1 <sub>B</sub> Test parity bits used (from PMTPR)
<b>MTENPS</b>	2	rw	<b>Memory Test Enable Control for Program SRAM</b> Controls the test multiplexer for the PSRAM. 0 <sub>B</sub> Standard operation 1 <sub>B</sub> Test parity bits used (from PMTPR)
<b>MTENSB</b>	8	rw	<b>Memory Test Enable Control for Standby Memory</b> Controls the test multiplexer for the SBRAM. 0 <sub>B</sub> Standard operation 1 <sub>B</sub> Test parity bits used (from PMTPR)
<b>PESEN</b>	15	rw	<b>Parity Error Sensitivity Enable</b> 0 <sub>B</sub> Parity errors have no effect 1 <sub>B</sub> Parity errors are indicated and can trigger a trap, if enabled
<b>0</b>	[7:3]	rw	<b>Reserved</b> Should be written with 0.
<b>0</b>	[14:9]	r	<b>Reserved</b> Read as 0; should be written with 0.

**System Control Unit (SCU)**

*Note: Only one bit MTENxx should be set at the same time in register PMTSR.  
Otherwise the result of the parity software test is not reliable.*

### 8.14.3 ECC Error Handling

During write operations ECC information is generated and stored along with the data. During read operations this ECC information is checked and in case of an error a trap request is generated (register **ECCSTAT**), if enabled. The requests from all RAMs can be combined (register **ECCCON**) and trigger a trap via bit ECCT in register **TRAPSTAT**.



**Figure 8-36 ECC Error Control Logic**

*Note: The handling of Flash ECC errors cannot be done completely in the SCU, but requires actions also in the IMB.*

*Note: When using the autoincrement mode to read the SBRAM and having ECC enabled, it is necessary not only to initialize the used addresses but also the one following the last used address. This problem does not exist if the whole SBRAM is initialized.*



### **8.14.3.1 ECC Software Testing Support**

The ECC error detection can be triggered on purpose to test the detection itself and the associated trap routine. This test option is available for RAMs that can operate with ECC and with parity protection, i.e. the standard RAM areas (DPRAM, DSRAM, PSRAM, SBRAM).

The software based ECC test uses both ECC and parity checking. It can generate a single-bit ECC error by executing the following sequence:

- Select parity mode for the respective RAM (MCHKCON.SELx = 1)
- Read back MCHKCON to allow the parity logic to activate
- Write 2000<sub>H</sub> to a location within this RAM (while parity is selected)
- Select ECC mode for the respective RAM (MCHKCON.SELx = 0)
- Read back MCHKCON to allow the ECC logic to activate
- Read from the chosen RAM location (while ECC is selected)

This will generate a single bit ECC error for databit 9:

If ECC operation is enabled (ECCCON.xEN = 1) the corrected data value (2000<sub>H</sub>) is read and an ECC error is indicated.

If ECC operation is disabled (ECCCON.xEN = 0) the uncorrected data value (2200<sub>H</sub>) is read and no error is indicated.

*Note: Due to the structure of the PSRAM this test modifies a complete 8-Byte memory line (...000<sub>B</sub> - ...111<sub>B</sub>). A non-destructive test, therefore, must save and restore all 4 words of the respective memory line.*

### 8.14.3.2 ECC Registers

#### ECCCON

**ECC Control Register**

**ESFR (F0A8<sub>H</sub>/54<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0										SB EN	MC EN	0	PS EN	DS EN	DP EN
rw										rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
<b>DPEN</b>	0	rw	<b>Enable for Dual Port Memory</b> 0 <sub>B</sub> Disabled 1 <sub>B</sub> ECC check and error correction for DPRAM enabled
<b>DSEN</b>	1	rw	<b>Enable for Data SRAM</b> 0 <sub>B</sub> Disabled 1 <sub>B</sub> ECC check and error correction for DSRAM enabled
<b>PSEN</b>	2	rw	<b>Enable for Program SRAM</b> 0 <sub>B</sub> Disabled 1 <sub>B</sub> ECC check and error correction for PSRAM enabled
<b>MCEN</b>	4	rw	<b>Enable for MultiCAN Memory</b> 0 <sub>B</sub> Disabled 1 <sub>B</sub> ECC check and error correction for MultiCAN memory enabled
<b>SBEN</b>	5	rw	<b>Enable for Standby Memory</b> 0 <sub>B</sub> Disabled 1 <sub>B</sub> ECC check and error correction for SBRAM enabled
<b>0</b>	3, [15:6]	rw	<b>Reserved</b> Should be written with 0.

**ECCSTAT**

**ECC Status Register**

**ESFR (F0AA<sub>H</sub>/55<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0										SB	MC	0	PS	DS	DP
rh										rh	rh	r	rh	rh	rh

Field	Bits	Type	Description
<b>DP</b>	0	rh	<b>Dual Port Memory ECC Error Status</b> 0 <sub>B</sub> No DPRAM error 1 <sub>B</sub> An ECC error was detected for the DPRAM
<b>DS</b>	1	rh	<b>Data SRAM ECC Error Status</b> 0 <sub>B</sub> No DSRAM error 1 <sub>B</sub> An ECC error was detected for the DSRAM
<b>PS</b>	2	rh	<b>Program SRAM ECC Error Status</b> 0 <sub>B</sub> No PSRAM error 1 <sub>B</sub> An ECC error was detected for the PSRAM
<b>MC</b>	4	rh	<b>MultiCAN Memory ECC Error Status</b> 0 <sub>B</sub> No MultiCAN error 1 <sub>B</sub> An ECC error was detected for the MultiCAN memory
<b>SB</b>	5	rh	<b>Standby Memory ECC Error Status</b> 0 <sub>B</sub> No SBRAM error 1 <sub>B</sub> An ECC error was detected for the SBRAM
<b>0</b>	3, [15:6]	rh	<b>Reserved</b> Read as 0.

**ECCCLRSTAT**

**ECC Clear Status Register**

**ESFR (F0DE<sub>H</sub>/6F<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
										SB	MC	0	PS	DS	DP
										W	W	W	W	W	W

Field	Bits	Type	Description
<b>DP</b>	0	w	<b>Clear Dual Port Memory ECC Error Status</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Setting this bit clears bit ECCSTAT.DP
<b>DS</b>	1	w	<b>Clear Data SRAM ECC Error Status</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Setting this bit clears bit ECCSTAT.DS
<b>PS</b>	2	w	<b>Clear Program SRAM ECC Error Status</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Setting this bit clears bit ECCSTAT.PS
<b>MC</b>	4	w	<b>Clear MultiCAN Memory ECC Error Status</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Setting this bit clears bit ECCSTAT.MC
<b>SB</b>	5	w	<b>Clear Standby Memory ECC Error Status</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Setting this bit clears bit ECCSTAT.SB
<b>0</b>	3, [15:6]	w	<b>Reserved</b>

*Note: These bits are always read as 0.*

## **8.15 Register Control**

This block handles the register accesses of the SCU and the register access control for all system register that use one of the following protection modes:

- Unprotected Mode
- Write Protection Mode
- Secured Mode

### **8.15.1 Register Access Control**

There are some dedicated registers that control critical system functions and modes. These registers are protected by a special register security mechanism so these vital system functions cannot be changed inadvertently after the executing of the EINIT instruction. However, as these registers control central system behavior they need to be accessed during operation. The system control software gets this access via a special security state machine.

If an access violation is detected a trap trigger request is generated.

This security mechanism controls the following security levels which can be configured via register SLC:

- **Unprotected Mode**  
No protection is active. Registers can be written at any time. This mode is entered after the Application Reset.
- **Write Protected Mode**  
Protected registers are locked against any write access. Write accesses have no effect on these registers. This mode is entered automatically after the EINIT instruction is executed.
- **Secured Mode**  
Protected registers can be written using a special command. Registers that are protected by this mode are marked in [Table 8-23](#) as Sec protected.  
Access in Secured Mode can be achieved by preceding the intended write access with writing “command 4” to register SLC. After writing “command 4” to register SLC the register protection mechanism remains disabled until the next write to a register on the PD+Bus (SFR, ESFR, XSFR area), i.e. accesses to registers (e.g. CSFR) outside this area do not enable the protection again automatically. Therefore, the lock mechanism after writing “command 4” works differently depending on the register address. Normally one single write access to a protected register is enabled. After this write access the protected registers are locked again automatically. Thereafter, “command 4” has to be written again in order to enable the next write to a protected register. The lock mechanism is not enabled again after a write access to a CSFR register or to a LXBus peripheral register (XLOC area, e.g. USIC, CAN, IMB).

*Note: In Secured Mode the re-enabling of register protection with respect to the write address after “command 4” can lead to an unexpected, not obvious behaviour of*

*an application:*

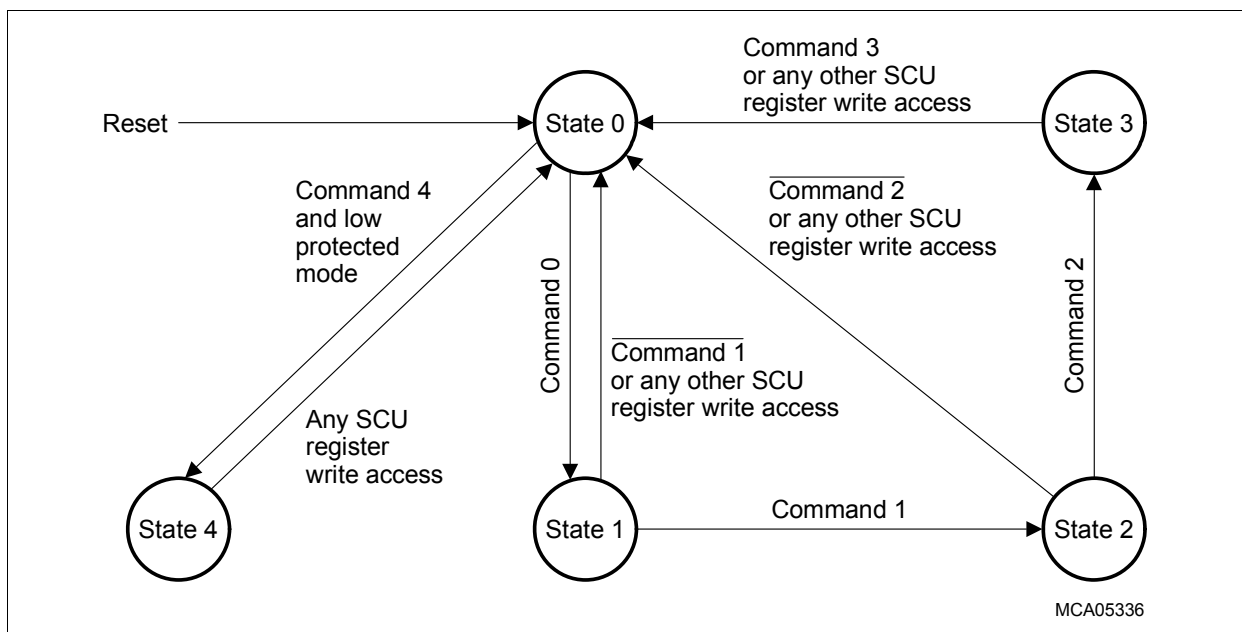
*In case the succeeding write to a protected register is delayed due to an interrupt and the ISR itself uses the “command 4” mechanism. After writing “command 4” inside the ISR the protection is expectedly re-installed instead of released and the following write will lead to an ACER trap within the ISR. An ATOMIC instruction, which couples the unlock with the write to the protected register could be used.*

*In case the succeeding write is to a register which does not re-enable the protection mechanism again then the write itself will succeed, but in a following “command 4” sequence the write to SLC register re-locks the protection again and the write to a protected register fails.*

All registers that are equipped with this protection mechanism have additional to normal access parameters (e.g. read only, bit type r or rh) the access limitations defined by the selected security level. Independently of the security level all protected registers can also be read.

### 8.15.1.1 Controlling the Security Level

The two registers Security Level Command register (SLC) and Security Level Status register (SLS) control the security level. The SLC register accepts the commands to control the state machine modifying the security level, while the SLS register shows the actual password, the actual security level, and the state of the state machine.



**Figure 8-37 State Machine for Security Level Controlling**

The following mechanism is used to control the actual security level:

- **Changing the security level**  
can be done by executing the following command sequence:

“command 0 - command 1 - command 2 - command 3”.

This sequence establishes a new security level and/or a new password.

**Table 8-15    Commands for Security Level Control**

<b>Command</b>	<b>Definition</b>	<b>Note</b>
Command 0	AAAA <sub>H</sub>	
Command 1	5554 <sub>H</sub>	
Command 2	96 <sub>H</sub> + <sup>1)</sup> <inverse password>	
Command 3	000 <sub>B</sub> + <new level> + 000 <sub>B</sub> + <new password>	
Command 4	8E <sub>H</sub> + <inverse password>	Secured Mode only

<sup>1)</sup> '+' denotes a bit field concatenation

*Note: It is recommended to lock all command sequences with an atomic sequence.*

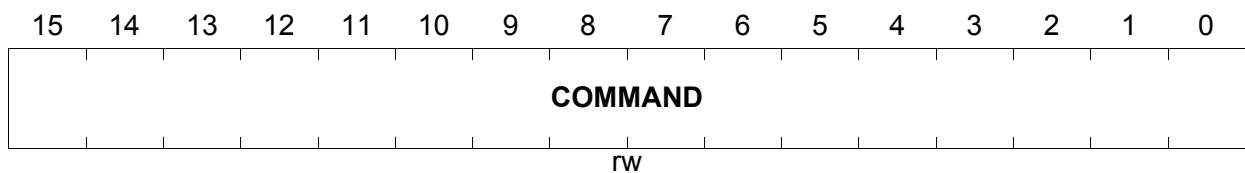
## 8.15.2 Register Protection Registers

### Register SLC

This register is the interface for the protection commands.

#### SLC

**Security Level Command RegisterESFR (F0C0<sub>H</sub>/60<sub>H</sub>)** **Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
COMMAND	[15:0]	rw	<b>Security Level Control Command</b> The commands to control the security level must be written to this register (see table)



## Register SLS

This register monitors the status of the register protection.

### SLS

**Security Level Status Register ESFR (F0C2<sub>H</sub>/61<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

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

Field	Bits	Type	Description
<b>PASSWORD</b>	[7:0]	rh	<b>Current Security Control Password</b> Default after reset = 00 <sub>H</sub>
<b>SL</b>	[12:11]	rh	<b>Security Level <sup>1)</sup></b> 00 <sub>B</sub> Unprotected Mode (default) 01 <sub>B</sub> Secured Mode 10 <sub>B</sub> Reserved, Do not use this combination 11 <sub>B</sub> Write Protected Mode
<b>STATE</b>	[15:13]	rh	<b>Current State of Switching State Machine</b> 000 <sub>B</sub> Awaiting command 0 or command 4 (default) 001 <sub>B</sub> Awaiting command 1 010 <sub>B</sub> Awaiting command 2 011 <sub>B</sub> Awaiting new security level and password 100 <sub>B</sub> Next access granted in Secured Mode 101 <sub>B</sub> Reserved, do not use this combination 11X <sub>B</sub> Reserved, do not use this combination
<b>0</b>	[10:8]	r	<b>Reserved</b> Read as 0; should be written with 0;

<sup>1)</sup> While the security level is “unprotected” after reset, it changes to “write protected” after the execution of instruction EINIT.

## 8.16 Miscellaneous System Registers

This chapter acts as container for various register that are not connected to one specific application topic.

### 8.16.1 System Registers

#### 8.16.1.1 System Control Register

The following register serve several different system tasks.

##### **SYSCON1**

**System Control 1 Register**

**SFR (FF4C<sub>H</sub>/A6<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0												GLC CST	OCD SEN	0	
r												rw	rw	r	

Field	Bits	Type	Description
<b>OCDSEN</b>	2	rw	<b>OCDS/Cerberus Enable</b> 0 <sub>B</sub> OCDS and Cerberus are still in reset state 1 <sub>B</sub> ODCS and Cerberus are operable
<b>GLCCST</b>	3	rw	<b>Global CAPCOM Start</b> This bit starts all CAPCOM units synchronously if enabled. 0 <sub>B</sub> CAPCOM timer start is controlled locally in each unit 1 <sub>B</sub> All CAPCOM timers are started synchronously This bit needs to be cleared via software before setting starts a new CAPCOM start.
<b>0</b>	[1:0], [15:4]	r	<b>Reserved</b> Read as 0; should be written with 0.

## 8.16.2 Identification Block

For identification of the most important silicon parameters a set of identification registers is defined that provide information on the chip manufacturer, the chip type and its properties.

### 8.16.2.1 Identification Registers

#### Register IDMANUF

This register contains information about the manufacturer.

#### IDMANUF

##### Manufacturer Identification Register

**ESFR (F07E<sub>H</sub>/3F<sub>H</sub>)**

**Reset Value: 1820<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>MANUF</b>											<b>DEPT</b>				
r											r				

Field	Bits	Type	Description
<b>DEPT</b>	[4:0]	r	<b>Department</b> Indicates the department within Infineon. 00 <sub>H</sub> AIM MC
<b>MANUF</b>	[15:5]	r	<b>Manufacturer</b> This is the JEDEC normalized manufacturer code. 0C1 <sub>H</sub> Infineon Technologies AG

## Register IDCHIP

This register contains information about the device.

### IDCHIP

**Chip Identification Register**      **ESFR (F07C<sub>H</sub>/3E<sub>H</sub>)**      **Reset Value: XXXX<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>CHIPID</b>								<b>Revision</b>							
r								r							

Field	Bits	Type	Description
<b>Revision</b>	[7:0]	r	<b>Device Revision Code</b> Identifies the device step. Please refer to the data sheet for the device specific value.
<b>CHIPID</b>	[15:8]	r	<b>Device Identification</b> Identifies the device name. Please refer to the data sheet for the device specific value.

## Register IDMEM

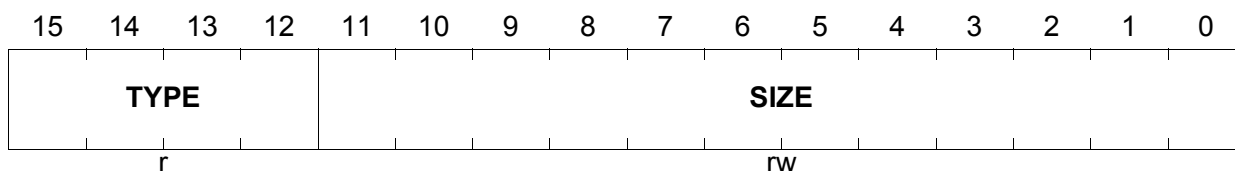
This register contains information about the program memory.

### IDMEM

#### Program Memory Identification Register

**ESFR (F07A<sub>H</sub>/3D<sub>H</sub>)**

**Reset Value: 3XXX<sub>H</sub>**



Field	Bits	Type	Description
<b>SIZE</b>	[11:0]	rw	<b>Size of on-chip Program Memory</b> The size of the implemented program memory in terms of 4 K blocks, i.e. memory size = <SIZE>*4 Kbyte. Please refer to the data sheet for the device specific value.
<b>TYPE</b>	[15:12]	r	<b>Type of on-chip Program Memory</b> Identifies the memory type on this silicon. Please refer to the data sheet for the device specific value.

## Register IDPROG

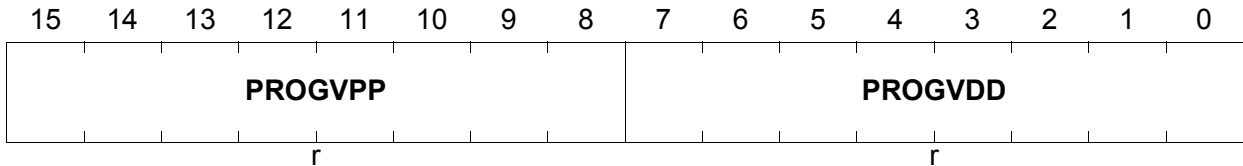
This register contains information about the flash programming voltage.

### IDPROG

#### Programming Voltage Id. Register

**ESFR (F078<sub>H</sub>/3C<sub>H</sub>)**

**Reset Value: 1313<sub>H</sub>**



Field	Bits	Type	Description
<b>PROGVDD</b>	[7:0]	r	<b>Programming VDD Voltage</b> The voltage of the standard power supply required to program or erase (if applicable) the on-chip program memory. Please refer to the data sheet for the device specific value.
<b>PROGVPP</b>	[15:8]	r	<b>Programming VPP Voltage</b> The voltage of the special programming power supply (if existent) required to program or erase (if applicable) the on-chip program memory. Please refer to the data sheet for the device specific value.

## Register IDDMPPM

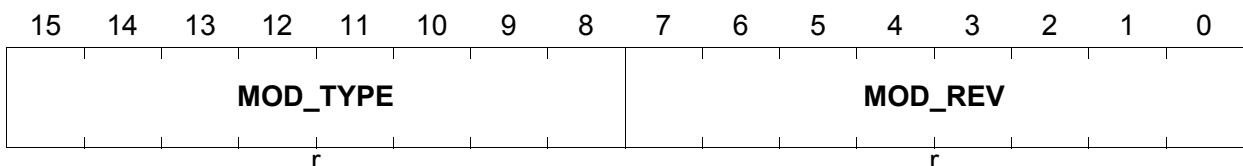
This register contains information about the DMP\_M SCU.

### IDDMPPM

#### DMP\_M Module Identification Register

**SFR (FFE2<sub>H</sub>)**

**Reset Value: 60XX<sub>H</sub>**



Field	Bits	Type	Description
<b>MOD_REV</b>	[7:0]	r	<b>Module Revision Number</b> The value of a module starts with 01 <sub>H</sub> .
<b>MOD_TYPE</b>	[15:8]	r	<b>Module Identification Number</b> SCU_M = 60 <sub>H</sub>

### Register IDDMP1

This register contains information about the DMP\_1 SCU.

#### IDDMP1

#### DMP\_1 Module Identification Register

**SFR (FFE4<sub>H</sub>)**

**Reset Value: 61XX<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>MOD_TYPE</b>								<b>MOD_REV</b>							
r								r							

Field	Bits	Type	Description
<b>MOD_REV</b>	[7:0]	r	<b>Module Revision Number</b> The value of a module starts with 01 <sub>H</sub> .
<b>MOD_TYPE</b>	[15:8]	r	<b>Module Identification Number</b> SCU_1 = 61 <sub>H</sub>

## **8.16.3 Marker Memory**

### **8.16.3.1 Marker Memory Registers**

The marker memory consists of following SFRs located in the DMP\_M for free usage of the user software.

#### **MKMEM0**

**Marker Memory 0 Register**      **SFR (FED0<sub>H</sub>/68<sub>H</sub>)**      **Reset Value: 0000<sub>H</sub>**

#### **MKMEM1**

**Marker Memory 1 Register**      **SFR (FED2<sub>H</sub>/69<sub>H</sub>)**      **Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
<b>MARKER</b>	[15:0]	rw	<b>Marker Content</b>



## **8.17 Implementation**

This section shows the connections of the module to the system.

### **8.17.1 Clock Generation Unit**

The following table shows the input connection of the Clock Generation Unit (see [Chapter 8.1](#)).

**Table 8-16 CGU Input Connection**

<b>Input</b>	<b>Connected to</b>
XTAL 1	XTAL 1
XTAL 2	XTAL 2
CLKIN1	Port 2.9
CLKIN2	Port 4.4

### 8.17.2 External Service Requests (ESR)

The availability of pins  $\overline{\text{ESR1}}$  and  $\overline{\text{ESR2}}$  is device and package dependent. It is described in the data sheet.

Pin  $\overline{\text{ESR0}}$  does not offer an overlay with other product functions.

For pins  $\overline{\text{ESR1}}$  and  $\overline{\text{ESR2}}$  an overlay with the ESRx inputs listed in [Table 8-17](#) and [Table 8-18](#) is possible. Even if an ESRx pin is not available an overlay with the ESRx inputs listed in the tables is possible. The ESRx logic part is fully functional.

Input from not available ESRx pins is tied to fixed value ("1" - value defined by ESRCFGx.PC).

**Table 8-17 ESR1 Input Connection**

Input	Connected to
Input 0	Port 2.4
Input 1	Port 3.0
Input 2	Port 10.0
Input 3	Port 1.0
Input 4	Port 1.2
Input 5	Port 2.1
Input 6	Port 6.1
Input 7	Port 11.0
Input 8	Port 4.1
Input 9	Port 10.4
Input 10	Port 2.5
Input 11	Port 0.0

**Table 8-18 ESR2 Input Connection**

Input	Connected to
Input 0	Port 2.3
Input 1	Port 7.0
Input 2	Port 10.14
Input 3	Port 1.1
Input 4	Port 1.3
Input 5	Port 2.2
Input 6	Port 2.6

**Table 8-18 ESR2 Input Connection**

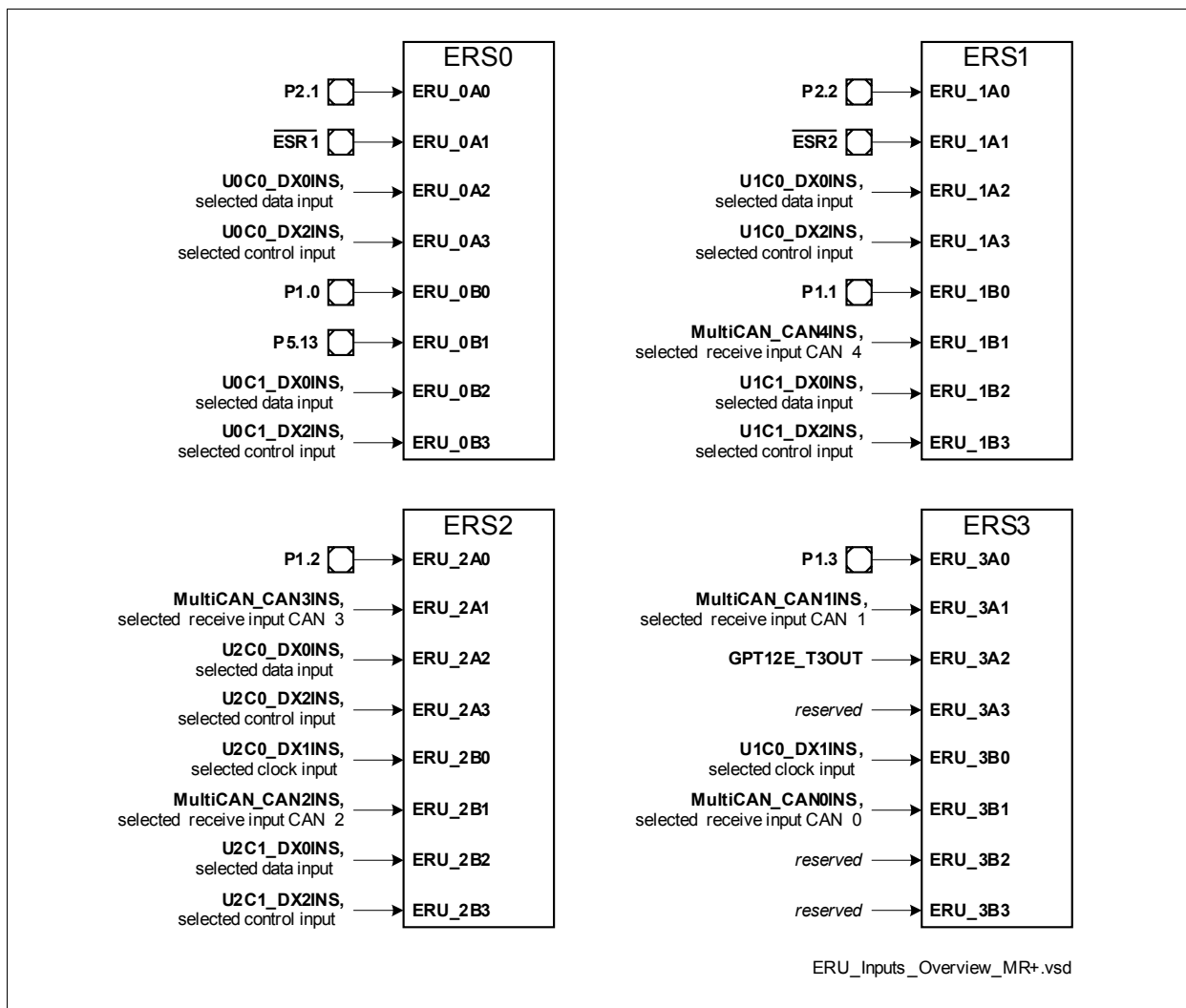
<b>Input</b>	<b>Connected to</b>
Input 7	Port 2.7
Input 8	Port 0.4
Input 9	XTAL 1
Input 10	Port 4.5
Input 11	Reserved, not connected

### 8.17.3 External Request Unit (ERU)

The connections of the ERU (see [Chapter 8.9](#)) are device specific. In the following the connections of the ERU in XE16xyM are described.

#### 8.17.3.1 ERU Input Connections

The following figure shows the ERU input connections, either directly with pins or via communication modules, such as USIC or MultiCAN. These communication modules provide their input signals (e.g. CAN receive input, or USIC data, clock, or control inputs) that have been selected in these modules.



**Figure 8-38 ERU Inputs Overview**

**System Control Unit (SCU)**

The following table describes the ERU input connections for the ERSx stages. The selection is defined by the bit fields in register **EXISEL**.

**Table 8-19 ERSx Connections in XE16xyM**

Input	from/to Module	I/O to ERSx	Can be used to/as
-------	-------------------	----------------	-------------------

**ERS0 Inputs**

ERU_0A0	P2.1	I	ERS0 input A
ERU_0A1	$\overline{\text{ESR1}}$	I	
ERU_0A2	U0C0_DX0INS	I	
ERU_0A3	U0C0_DX2INS	I	
ERU_0B0	P1.0	I	ERS0 input B
ERU_0B1	P5.13	I	
ERU_0B2	U0C1_DX0INS	I	
ERU_0B3	U0C1_DX2INS	I	

**ERS1 Inputs**

ERU_1A0	P2.2	I	ERS1 input A
ERU_1A1	$\overline{\text{ESR2}}$	I	
ERU_1A2	U1C0_DX0INS	I	
ERU_1A3	U1C0_DX2INS	I	
ERU_1B0	P1.1	I	ERS1 input B
ERU_1B1	MultiCAN_CAN4INS	I	
ERU_1B2	U1C1_DX0INS	I	
ERU_1B3	U1C1_DX2INS	I	

**ERS2 Inputs**

ERU_2A0	P1.2	I	ERS2 input A
ERU_2A1	MultiCAN_CAN3INS	I	
ERU_2A2	U2C0_DX0INS	I	
ERU_2A3	U2C0_DX2INS	I	

**Table 8-19 ERSx Connections in XE16xyM (cont'd)**

<b>Input</b>	<b>from/to Module</b>	<b>I/O to ERSx</b>	<b>Can be used to/as</b>
ERU_2B0	U2C0_DX1INS	I	ERS2 input B
ERU_2B1	MultiCAN_CAN2INS	I	
ERU_2B2	U2C1_DX0INS	I	
ERU_2B3	U2C1_DX2INS	I	

**ERS3 Inputs**

ERU_3A0	P1.3	I	ERS3 input A
ERU_3A1	MultiCAN_CAN1INS	I	
ERU_3A2	GPT12E_T3OUT	I	
ERU_3A3	0	I	
ERU_3B0	U1C0_DX1INS	I	ERS3 input B
ERU_3B1	MultiCAN_CAN0INS	I	
ERU_3B2	0	I	
ERU_3B3	0	I	

### 8.17.3.2 Output Gating Unit (OGUy)

The following table describes the peripheral trigger connections for the OGUy stages.

**Table 8-20 OGUy Peripheral Trigger Connections in XE16xyM**

Input	from/to Module	I/O to OGUy	Can be used to/as
-------	-------------------	----------------	-------------------

#### **OGU0 Inputs**

ERU_OGU01	CCU60_MCM_ST	I	Peripheral triggers for OGU0
ERU_OGU02	CCU60_T13_PM	I	
ERU_OGU03	CC2_28	I	

#### **OGU1 Inputs**

ERU_OGU11	CCU61_MCM_ST	I	Peripheral triggers for OGU1
ERU_OGU12	CCU61_T13_PM	I	
ERU_OGU13	CC2_29	I	

#### **OGU2 Inputs**

ERU_OGU21	CCU62_MCM_ST	I	Peripheral triggers for OGU2
ERU_OGU22	CCU62_T13_PM	I	
ERU_OGU23	CC2_30	I	

#### **OGU3 Inputs**

ERU_OGU31	CCU63_MCM_ST	I	Peripheral triggers for OGU3
ERU_OGU32	CCU63_T13_PM	I	
ERU_OGU33	CC2_31	I	

### 8.17.3.3 ERU Output Connections

The following table describes the connections of the ERU output signals for gating or triggering other module functions, as well as the connections to the interrupt control registers.

**Table 8-21 ERU Output Connections in XE16xyM**

Output	from/to Module	I/O to OGUy	Can be used to/as
--------	-------------------	----------------	-------------------

#### OGU0 Outputs

ERU_PDOUT0	ADC0 (REQGT0E) ADC0 (REQGT1E) ADC0 (REQGT2E) ADC1 (REQGT0E) ADC1 (REQGT1E) ADC1 (REQGT2E)	O	Pattern detection output
ERU_GOUT0	not connected	O	Gated pattern detection output
ERU_GOUT0	not connected	O	Gated pattern detection output
ERU_TOUT0	not connected	O	Trigger output
ERU_IOUT0	ITC (SCU_ERU_0IC)	O	Interrupt output

#### OGU1 Outputs

ERU_PDOUT1	ADC0 (REQGT0F) ADC0 (REQGT1F) ADC0 (REQGT2F) ADC1 (REQGT0F) ADC1 (REQGT1F) ADC1 (REQGT2F)	O	Pattern detection output
ERU_GOUT1	not connected	O	Gated pattern detection output
ERU_TOUT1	ADC0 (REQTR0B) ADC0 (REQTR1B) ADC0 (REQTR2B) ADC1 (REQTR0B) ADC1 (REQTR1B) ADC1 (REQTR2B)	O	Trigger output
ERU_IOUT1	ITC (SCU_ERU_1IC)	O	Interrupt output

#### OGU2 Outputs



**System Control Unit (SCU)**

**Table 8-21 ERU Output Connections in XE16xyM (cont'd)**

<b>Output</b>	<b>from/to Module</b>	<b>I/O to OGUy</b>	<b>Can be used to/as</b>
ERU_PDOUT2	CCU60 (CTRAPD)	O	Pattern detection output
ERU_GOUT2	not connected	O	Gated pattern detection output
ERU_TOUT2	not connected	O	Trigger output
ERU_IOUT2	ITC (SCU_ERU_2IC)	O	Interrupt output

**OGU3 Outputs**

ERU_PDOUT3	CCU63 (CTRAPD)	O	Pattern detection output
ERU_GOUT3	not connected	O	Gated pattern detection output
ERU_TOUT3	not connected	O	Trigger output
ERU_IOUT3	ITC (SCU_ERU_3IC)	O	Interrupt output

## 8.18 SCU Register Addresses

The SCU registers are within the (E)SFR space of the XE16xyM. Therefore, their specified addresses equal an offset from zero.

**Table 8-22 Registers Address Space**

Module	Base Address	End Address	Note
SCU	00 0000 <sub>H</sub>	00 FFFE <sub>H</sub>	

### SCU Register Overview

**Table 8-23 Register Overview of SCU**

Short Name	Register Long Name	Offset Addr.	Protection <sup>1)</sup>	Reset <sup>2)</sup>
<b>WUOSCCON</b>	Wake-up OSC Control Register	F1AE <sub>H</sub>	Sec	Power-on Reset
<b>HPOSCCON</b>	High Precision Oscillator Configuration Register	F1B4 <sub>H</sub>	Sec	Power-on Reset
<b>PLLOSCCON</b>	PLL Control Register	F1B6 <sub>H</sub>	Sec	Power-on Reset
<b>PLLSTAT</b>	PLL Status Register	F0BC <sub>H</sub>	-	Power-on Reset
<b>STATCLR1</b>	PLL Status Clear 1 Register	F0E2 <sub>H</sub>	Sec	Power-on Reset
<b>PLLCON0</b>	PLL Configuration 0 Register	F1B8 <sub>H</sub>	Sec	Power-on Reset
<b>PLLCON1</b>	PLL Configuration 1 Register	F1BA <sub>H</sub>	Sec	Power-on Reset
<b>PLLCON2</b>	PLL Configuration 2 Register	F1BC <sub>H</sub>	Sec	Power-on Reset
<b>PLLCON3</b>	PLL Configuration 3 Register	F1BE <sub>H</sub>	Sec	Power-on Reset
<b>SYSCON0</b>	System Configuration 0 Register	FF4A <sub>H</sub>	Sec	Power-on Reset
<b>STATCLR0</b>	Status Clear 0 Register	F0E0 <sub>H</sub>	Sec	Power-on Reset
<b>RTCCLKCON</b>	RTC Clock Control Register	FF4E <sub>H</sub>	Sec	Power-on Reset

**Table 8-23 Register Overview of SCU**

<b>Short Name</b>	<b>Register Long Name</b>	<b>Offset Addr.</b>	<b>Protection<sup>1)</sup></b>	<b>Reset<sup>2)</sup></b>
<b>EXTCON</b>	External Clock Control Register	FF5E <sub>H</sub>	Sec	Power-on Reset
<b>STMREL</b>	STM Reload Register	F1A8 <sub>H</sub>	Sec	Power-on Reset
<b>STMCON</b>	STM Control Register	F1AA <sub>H</sub>	Sec	Power-on Reset
<b>WUTREL</b>	Wake-up Timer Reload Register	F0B0 <sub>H</sub>	Sec	Power-on Reset
<b>WUCR</b>	Wake-up Control Register	F1B0 <sub>H</sub>	Sec	Power-on Reset
<b>RSTSTAT0</b>	Reset Status 0 Register	F0B2 <sub>H</sub>	-	Power-on Reset
<b>RSTSTAT1</b>	Reset Status 1 Register	F0B4 <sub>H</sub>	-	Power-on Reset
<b>RSTSTAT2</b>	Reset Status 2 Register	F0B6 <sub>H</sub>	-	Power-on Reset
<b>RSTCON0</b>	Reset Configuration 0 Register	F0B8 <sub>H</sub>	Sec	Power-on Reset
<b>RSTCON1</b>	Reset Configuration 1 Register	F0BA <sub>H</sub>	Sec	Power-on Reset
<b>RSTCNTCON</b>	Reset Counter Configuration Register	F1B2 <sub>H</sub>	Sec	Power-on Reset
<b>SWRSTCON</b>	SW Reset Control Register	F0AE <sub>H</sub>	Sec	Power-on Reset
<b>ESREXCON1</b>	ESR 1 External Control Register	FF32 <sub>H</sub>	Sec	Power-on Reset
<b>ESREXCON2</b>	ESR 2 External Control Register	FF34 <sub>H</sub>	Sec	Power-on Reset
<b>ESREXSTAT1</b>	ESR 1 External Status Register	FF36 <sub>H</sub>	-	Power-on Reset
<b>ESREXSTAT2</b>	ESR 2 External Status Register	FF38 <sub>H</sub>	-	Power-on Reset
<b>CLRESREXSTAT1</b>	Clear ESR 1 External Status Register	FF3A <sub>H</sub>	Sec	Power-on Reset

**Table 8-23 Register Overview of SCU**

<b>Short Name</b>	<b>Register Long Name</b>	<b>Offset Addr.</b>	<b>Protection<sup>1)</sup></b>	<b>Reset<sup>2)</sup></b>
<b>CLRESREXSTAT2</b>	Clear ESR 2 External Status Register	FF3C <sub>H</sub>	Sec	Power-on Reset
<b>ESRCFG0</b>	ESR 0 Configuration Register	F100 <sub>H</sub>	Sec	Power-on Reset
<b>ESRCFG1</b>	ESR 1 Configuration Register	F102 <sub>H</sub>	Sec	Power-on Reset
<b>ESRCFG2</b>	ESR 2 Configuration Register	F104 <sub>H</sub>	Sec	Power-on Reset
<b>ESRDAT</b>	ESR Data Register	F106 <sub>H</sub>	Sec	Power-on Reset
<b>SWDCON0</b>	SWD Control 0 Register	F080 <sub>H</sub>	Sec	Power-on Reset
<b>SWDCON1</b>	SWD Control 1 Register	F082 <sub>H</sub>	Sec	Power-on Reset
<b>PVC1CON0</b>	PVC_1 Control for Step 0 Register	F014 <sub>H</sub>	Sec	Power-on Reset
<b>PVCMCON0</b>	PVC_M Control for Step 0 Register	F1E4 <sub>H</sub>	Sec	Power-on Reset
<b>EVR1CON0</b>	EVR_1 Control 0 Register	F088 <sub>H</sub>	Sec	Power-on Reset
<b>EVR1SET15VHP</b>	EVR_1 Setting for 1.5V HP Register	F09E <sub>H</sub>	Sec	Power-on Reset
<b>EVRMCON0</b>	EVR_M Control 0 Register	F084 <sub>H</sub>	Sec	Power-on Reset
<b>EVRMCON1</b>	EVR_M Control 1 Register	F086 <sub>H</sub>	Sec	Power-on Reset
<b>EVRMSET15VHP</b>	EVR_M Setting for 1.5V HP Register	F096 <sub>H</sub>	Sec	Power-on Reset
<b>GSCSWREQ</b>	GSC SW Request Register	FF14 <sub>H</sub>	Sec	Application Reset
<b>GSCEN</b>	GSC Enable Register	FF16 <sub>H</sub>	Sec	Application Reset
<b>GSCSTAT</b>	GSC Status Register	FF18 <sub>H</sub>	-	Application Reset

**Table 8-23 Register Overview of SCU**

<b>Short Name</b>	<b>Register Long Name</b>	<b>Offset Addr.</b>	<b>Protection<sup>1)</sup></b>	<b>Reset<sup>2)</sup></b>
<b>GSCPERSTATEN</b>	GSC Peripheral Status Enable Register	FF04 <sub>H</sub>	Sec	Application Reset
<b>GSCPERSTAT</b>	GSC Peripheral Status Register	FF1A <sub>H</sub>	-	Application Reset
<b>STSTAT</b>	Start-up Status Register	F1E0 <sub>H</sub>	-	Application Reset
<b>EXISEL</b>	External Interrupt Input Select Register	F1A0 <sub>H</sub>	Sec	Application Reset
<b>EXICON0</b>	External Interrupt Input Trigger Control 0 Register	F030 <sub>H</sub>	Sec	Application Reset
<b>EXICON1</b>	External Interrupt Input Trigger Control 1 Register	F032 <sub>H</sub>	Sec	Application Reset
<b>EXICON2</b>	External Interrupt Input Trigger Control 2 Register	F034 <sub>H</sub>	Sec	Application Reset
<b>EXICON3</b>	External Interrupt Input Trigger Control 3 Register	F036 <sub>H</sub>	Sec	Application Reset
<b>EXOCON0</b>	External Output Trigger Control 0 Register	FE30 <sub>H</sub>	Sec	Application Reset
<b>EXOCON1</b>	External Output Trigger Control 1 Register	FE32 <sub>H</sub>	Sec	Application Reset
<b>EXOCON2</b>	External Output Trigger Control 2 Register	FE34 <sub>H</sub>	Sec	Application Reset
<b>EXOCON3</b>	External Output Trigger Control 3 Register	FE36 <sub>H</sub>	Sec	Application Reset
<b>INTSTAT</b>	Interrupt Status Register	FF00 <sub>H</sub>	-	Application Reset
<b>INTCLR</b>	Interrupt Clear Register	FE82 <sub>H</sub>	Sec	Application Reset
<b>INTSET</b>	Interrupt Set Register	FE80 <sub>H</sub>	Sec	Application Reset
<b>INTDIS</b>	Interrupt Disable Register	FE84 <sub>H</sub>	Sec	Application Reset
<b>INTNP0</b>	Interrupt Node Pointer 0 Register	FE86 <sub>H</sub>	Sec	Application Reset

**Table 8-23 Register Overview of SCU**

<b>Short Name</b>	<b>Register Long Name</b>	<b>Offset Addr.</b>	<b>Protection<sup>1)</sup></b>	<b>Reset<sup>2)</sup></b>
<b>INTNP1</b>	Interrupt Node Pointer 1 Register	FE88 <sub>H</sub>	Sec	Application Reset
<b>DMPMIT</b>	DMP_M Interrupt and Trap Trigger Register	FE96 <sub>H</sub>	-	Power-on Reset
<b>DMPMITCLR</b>	DMP_M Interrupt and Trap Clear Register	FE98 <sub>H</sub>	Sec	Power-on Reset
<b>TCCR</b>	Temperature Compensation Control Register	F1AC <sub>H</sub>	Sec	Application Reset
<b>TCLR</b>	Temperature Compensation Level Register	F0AC <sub>H</sub>	Sec	Application Reset
<b>WDTREL</b>	WDT Reload Register	F0C8 <sub>H</sub>	Sec	Application Reset
<b>WDTCS</b>	WDT Control and Status Register	F0C6 <sub>H</sub>	Sec	Application Reset
<b>WDTTIM</b>	WDT Timer Register	F0CA <sub>H</sub>	Sec	Application Reset
<b>TRAPSTAT</b>	Trap Status Register	FF02 <sub>H</sub>	-	Power-on Reset
<b>TRAPCLR</b>	Trap Clear Register	FE8E <sub>H</sub>	Sec	Power-on Reset
<b>TRAPSET</b>	Trap Set Register	FE8C <sub>H</sub>	Sec	Power-on Reset
<b>TRAPDIS</b>	Trap Disable Register	FE90 <sub>H</sub>	Sec	Power-on Reset
<b>TRAPNP</b>	Trap Node Pointer Register	FE92 <sub>H</sub>	Sec	Power-on Reset
<b>TRAPNP1</b>	Trap Node Pointer 1 Register	FE94 <sub>H</sub>	Sec	Power-on Reset
<b>MCHKCON</b>	Memory Checking Control Register	F0DC <sub>H</sub>	Sec	Power-on Reset
<b>PEEN</b>	Parity Error Enable Register	F0C4 <sub>H</sub>	Sec	Power-on Reset

**Table 8-23 Register Overview of SCU**

<b>Short Name</b>	<b>Register Long Name</b>	<b>Offset Addr.</b>	<b>Protection<sup>1)</sup></b>	<b>Reset<sup>2)</sup></b>
<b>PECON</b>	Parity Error Control Register	F0DA <sub>H</sub>	Sec	Power-on Reset
<b>PMPTR</b>	Parity Memory Test Pattern Register	F0E4 <sub>H</sub>	Sec	Application Reset
<b>PMTSR</b>	Parity Memory Test Select Register	F0E6 <sub>H</sub>	Sec	Application Reset
<b>ECCCON</b>	ECC Control Register	F0A8 <sub>H</sub>	Sec	Application Reset
<b>ECCSTAT</b>	ECC Status Register	F0AA <sub>H</sub>	-	Application Reset
<b>ECCCLRSTAT</b>	ECC Clear Status Register	F0DE <sub>H</sub>	Sec	Application Reset
<b>SLC</b>	Security Level Command Register	F0C0 <sub>H</sub>	-	Application Reset
<b>SLS</b>	Security Level Status Register	F0C2 <sub>H</sub>	-	Application Reset
<b>SYSCON1</b>	System Control 1 Register	FF4C <sub>H</sub>	Sec	Application Reset
<b>IDMANUF</b>	Manufacturer Identification Register	F07E <sub>H</sub>	-	Power-on Reset
<b>IDCHIP</b>	Chip Identification Register	F07C <sub>H</sub>	-	Power-on Reset
<b>IDMEM</b>	Program Memory Identification Register	F07A <sub>H</sub>	-	Power-on Reset
<b>IDPROG</b>	Programming Voltage Identification Register	F078 <sub>H</sub>	-	Power-on Reset
<b>IDDMPM</b>	DMP_M Identification Register	FFE2 <sub>H</sub>	-	Power-on Reset
<b>IDDMP1</b>	DMP_1 Identification Register	FFE4 <sub>H</sub>	-	Power-on Reset
<b>MKMEM0</b>	Marker Memory 0 Register	FED0 <sub>H</sub>	Sec	Power-on Reset
<b>MKMEM1</b>	Marker Memory 1 Register	FED2 <sub>H</sub>	Sec	Power-on Reset

**System Control Unit (SCU)**

- 1) Register write protection mechanism: "Sec" = register security mechanism, "-" = always accessible (no protection), otherwise no access is possible.
- 2) Reset types are defined in [Chapter 8.4.1.2](#).



## 9 Parallel Ports

The XE16xyM provides a set of General Purpose Input/Output (GPIO) ports that can be controlled by software and by the on-chip peripheral units:

**Table 9-1 Ports of the XE16xyM**

Port	Width	I/O	Connected Modules
P0	8	I/O	EBC (A7...A0), CCU6, USIC, CAN
P1	8	I/O	EBC (A15...A8), CCU6, USIC
P2	14	I/O	EBC (READY, $\overline{\text{BHE}}$ , A23...A16, AD15...AD13, D15...D13), CAN, CC2, GPT12E, USIC, DAP/JTAG
P3	8	I/O	EBC arbitration ( $\overline{\text{BREQ}}$ , $\overline{\text{HLDA}}$ , $\overline{\text{HOLD}}$ ), CAN, USIC
P4	8	I/O	EBC ( $\overline{\text{CS4}}$ ... $\overline{\text{CS0}}$ ), CC2, CAN, GPT12E, USIC
P5	16	I	Analog Inputs, CCU6, DAP/JTAG, GPT12E, CAN
P6	4	I/O	ADC, CAN, GPT12E
P7	5	I/O	CAN, GPT12E, SCU, DAP/JTAG, CCU6, ADC, USIC
P8	7	I/O	CCU6, DAP/JTAG, USIC
P9	8	I/O	CCU6, DAP/JTAG, CAN
P10	16	I/O	EBC(ALE, $\overline{\text{RD}}$ , $\overline{\text{WR}}$ , AD12...AD0, D12...D0), CCU6, USIC, DAP/JTAG, CAN
P11	6	I/O	CCU6, USIC, CAN
P15	8	I	Analog Inputs, GPT12E

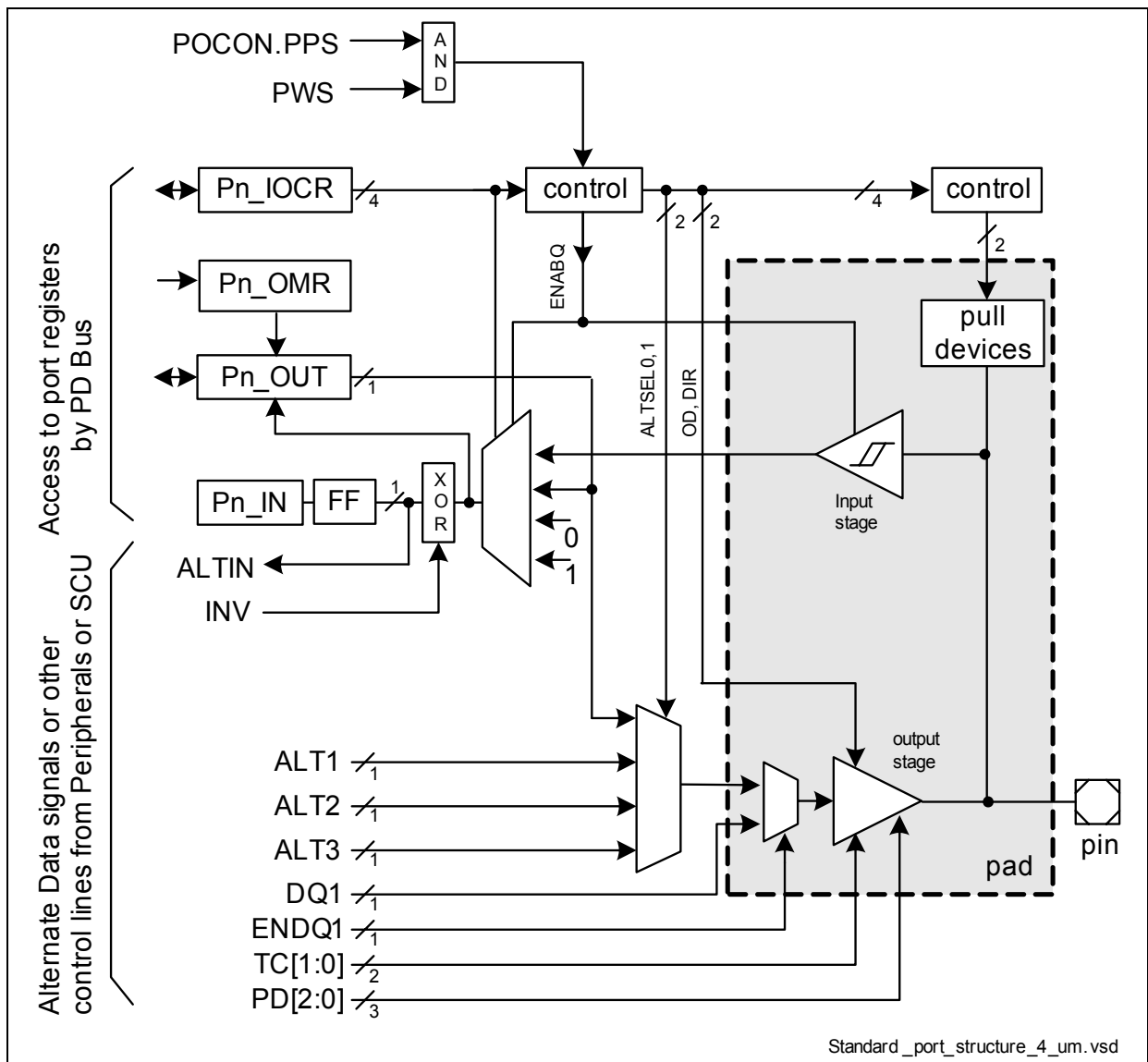
*Note: The availability of ports and port pins depends on the selected device type.  
This chapter describes the maximum set of ports.*

## 9.1 General Description

This chapter describes the architecture of the digital control circuit for a single port pin.

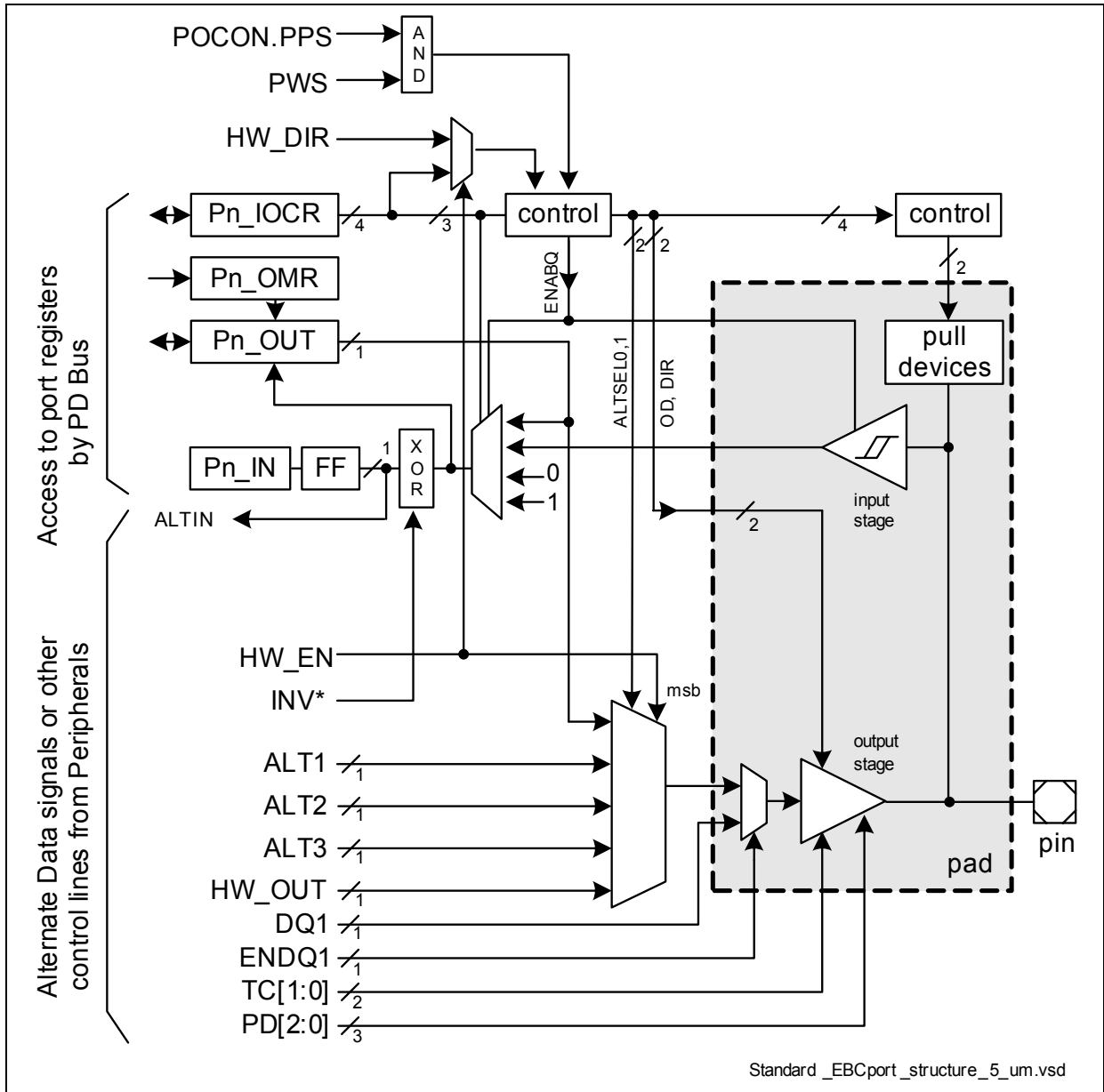
### 9.1.1 Basic Port Operation

There are three types of digital control circuits: with/without hardware override for digital GPIOs, and for one for analog inputs. Each port pin contains one of them.



**Figure 9-1 Structure of the Ports without Hardware Override Functionality**

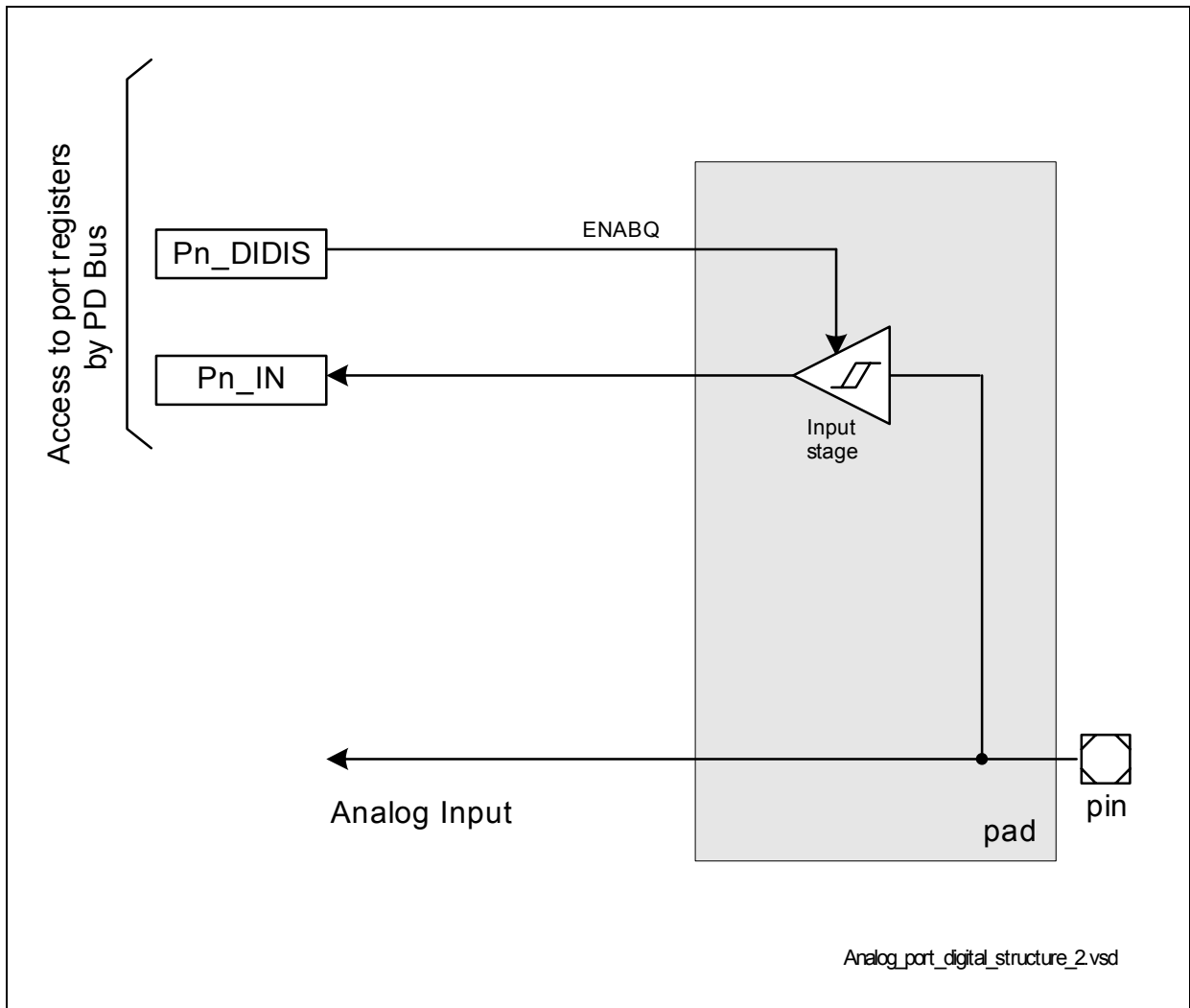
*Note: INV signal is derived from Pn\_IOC.PC[3:2].*



**Figure 9-2 Structure of the Ports with Hardware Override Functionality**

*Note: If HW\_EN is activated, INV\* signal is always zero.*

*Note: When HW\_EN is disabled, the respective ports go to Power Save Mode as all other ports. When HW\_EN is active, then the user should set the POCON.PPSx=0.*



**Figure 9-3 Structure of Port 5 and Port 15**

*Note: There is always a standard digital input connected in parallel to each analog input.*

### **9.1.2 Input Stage Control**

An input stage consists of a Schmitt trigger, which can be enabled or disabled via software, and an input multiplexer that by default selects the output of the input Schmitt trigger.

A disabled input driver drives high logical level. During and after reset, all input stages are enabled by default.

### **9.1.3 Output Driver Control**

An output stage consists of an output driver, output multiplexer, and register bit fields for their control.

#### **9.1.3.1 Active Mode Behavior**

Each output driver can be configured in a push-pull or an open-drain mode, or it can be deactivated (three-stated). An output multiplexer in front of the output driver selects the signal source, choosing either the appropriate bit of the Pn\_OUT register, or one of maximum three lines coming from a peripheral unit, see [Figure 9-1](#). The selection is done via the Pn\_IOCRR register. Software can set or clear the bit Pn\_OUT.Px, which drives the port pin in case it is selected by the output multiplexer.

An output driver with hardware override can select an additional output signal coming from a peripheral. While the hardware override is activated, this signal has higher priority than all other output signals and can not be deselected by the port. In this case, the peripheral controls the direction of the pin.

#### **9.1.3.2 Power Saving Mode Behavior**

In Power Saving Mode (core and IO supply voltages available), the behavior of a pin depends on the setting of the PCONx.PPSx bit. Basically, groups of four pins within a port can be configured to react to Power Save Mode Request or to ignore it. In case a pin group is configured to react to a Power Save Mode Request, each pin within a group reacts according to its own configuration according to the [Table 9-4](#).

#### **9.1.3.3 Reset Behavior**

During an Internal Application Reset, all output stages of GPIO pins go to tri-state mode without any pull-up or pull-down devices.

An Application Reset does not change the GPIO configuration but the reset of the internal peripherals can change the data driven on the outputs. Attention must be paid to ensure that no harm is caused to the connected devices by unexpected transitions and output values.

#### **9.1.3.4 Power-fail Behavior**

When the core supply fails while the pad supply remains stable, the output stages go into tri-state mode.

### **9.2 Port Register Description**

#### **9.2.1 Pad Driver Control**

The pad structure used in this device offers the possibility to select the output driver strength and the slew rate. These selections are independent from the output port functionality, such as open-drain, push/pull or input only.

In order to minimize EMI problems, the driver strength can be adapted to the application requirements by bit fields PDMx. The selection is done in groups of four pins.

The **Port Output Control registers** POCON provide the corresponding control bits. A 4-bit control field configures the driver strength and the edge shape. Word ports consume four control nibbles each, byte ports consume two control nibbles each, where each control nibble controls 4 pins of the respective port.

*Note: P2\_POCON register in the XE16xyM contains an exception regarding the additional strong output driver connected in parallel to the standard output driver of the P2.8 pin. See the respective port section for details.*

**Parallel Ports**

**Px\_POCON (x=0-4)**

**Port x Output Control Register XSFR (E8A0<sub>H</sub>+2\*x)**

**Reset Value: 0000<sub>H</sub>**

**Px\_POCON (x=6-11)**

**Port x Output Control Register XSFR (E8A0<sub>H</sub>+2\*x)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>PPS 3</b>	<b>PDM3</b>			<b>PPS 2</b>	<b>PDM2</b>			<b>PPS 1</b>	<b>PDM1</b>			<b>PPS 0</b>	<b>PDM0</b>		
rw	rwr			rw	rw			rw	rw			rw	rw		

Field	Bit	Type	Description
<b>PDM0, PDM1, PDM2, PDM3</b>	[2:0], [6:4], [10:8], [14:12]	rw	<b>Port Driver Mode x</b> Code Driver strength <sup>1)</sup> Edge Shape <sup>2)</sup> 000 <sub>B</sub> Strong driver                      Sharp edge mode 001 <sub>B</sub> Strong driver                      Medium edge mode 010 <sub>B</sub> Strong driver                      Soft edge mode 011 <sub>B</sub> Weak driver 100 <sub>B</sub> Medium driver 101 <sub>B</sub> Medium driver 110 <sub>B</sub> Medium driver 111 <sub>B</sub> Weak driver
<b>PPS0, PPS1, PPS2, PPS3</b>	3, 7, 11, 15	rw	<b>Pin Power Save</b> 0 <sub>B</sub> Pin behaves like in the Active Mode. Power Save Management is ignored. 1 <sub>B</sub> Behavior in the Power Save Mode described in the <a href="#">Table 9-4</a> .

<sup>1)</sup> Defines the current the respective driver can deliver to the external circuitry.

<sup>2)</sup> Defines the switching characteristics to the respective new output level. This also influences the peak currents through the driver when producing an edge, i.e. when changing the output level.

### Mapping of the POCON Registers to Pins and Ports

The table below lists the defined POCON registers and the allocation of control bit fields and port pins.

**Table 9-2 Port Output Control Register Allocation**

Control Register	Controlled Pins (by Px_POCON.[y:z]) <sup>1)</sup>				Port Width
	[15:12]	[11:8]	[7:4]	[3:0]	
P0_POCON	---	---	P0.[7:4]	P0.[3:0]	8
P1_POCON	---	---	P1.[7:4]	P1.[3:0]	8
P2_POCON	CLOCKOUT driver at P2.8 <sup>2)</sup>	P2.[11:8] + P2.[13:12] <sup>3)</sup>	P2.[7:4]	P2.[3:0]	14
P3_POCON	---	---	P3.[7:4]	P3.[3:0]	8
P4_POCON	---	---	P4.[7:4]	P4.[3:0]	8
P6_POCON	---	---	---	P6.[3:0]	4
P7_POCON	---	---	P7.4	P7.[3:0]	5
P8_POCON	---	---	P8.[6:4]	P8.[3:0]	7
P9_POCON	---	---	P9.[7:4]	P9.[3:0]	8
P10_POCON	P10.[15:12]	P10.[11:8]	P10.[7:4]	P10.[3:0]	16
P11_POCON	---	---	P11.[5:4]	P11.[3:0]	6

<sup>1)</sup> x denotes the port number, while [y:z] represents the bit field range.

<sup>2)</sup> The control of the additional driver is described in [Chapter 9.3.3](#).

<sup>3)</sup> The output control of P2.[13:12] deviates from the standard definition, see [Chapter 9.3.3](#).

*Note: When assigning functional signals to port pins, please consider the fact that the driver strength is selected for pin groups. Assign functions with similar requirements to pins within the same POCON control group.*



## 9.2.2 Port Output Register

The port output register defines the values of the output pins if the pin is used as GPIO output.

**Pn\_OUT (n=0-4)**

**Port n Output Register**

**SFR (FFA2<sub>H</sub>+2\*n)**

**Reset Value: 0000<sub>H</sub>**

**Pn\_OUT (n=6-11)**

**Port n Output Register**

**SFR (FFA2<sub>H</sub>+2\*n)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>P15</b>	<b>P14</b>	<b>P13</b>	<b>P12</b>	<b>P11</b>	<b>P10</b>	<b>P9</b>	<b>P8</b>	<b>P7</b>	<b>P6</b>	<b>P5</b>	<b>P4</b>	<b>P3</b>	<b>P2</b>	<b>P1</b>	<b>P0</b>
rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
<b>Px</b> <b>(x = 0-15)</b>	x	rwh	<b>Port Output Bit x</b> This bit defines the level at the output pin of port Pn, pin x if the output is selected as GPIO output. 0 <sub>B</sub> The output level of Pn.x is 0. 1 <sub>B</sub> The output level of Pn.x is 1.

### 9.2.3 Port Output Modification Register

The port output modification register contains the bits to individually set, clear, or toggle the value of the port n output register.

#### P2\_OMRH

**Port 2 Output Modification Register HighXSFR (E9CA<sub>H</sub>)**      **Reset Value: XXXX<sub>H</sub>**

#### P10\_OMRH

**Port 10 Output Modification Register HighXSFR (E9EA<sub>H</sub>)**      **Reset Value: XXXX<sub>H</sub>**

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

Field	Bits	Type	Description
<b>PSx</b> (x = 8-15)	x-8	w	<b>Port Set Bit x</b> Setting this bit sets or toggles the corresponding bit in the port output register Pn_OUT (see <a href="#">Table 9-3</a> ). On a read access, this bit returns an undefined value.
<b>PCx</b> (x = 8-15)	x	w	<b>Port Clear Bit x</b> Setting this bit clears or toggles the corresponding bit in the port output register Pn_OUT. (see <a href="#">Table 9-3</a> ). On a read access, this bit returns an undefined value.

#### Pn\_OMRL (n=0-4)

**Port n Output Modification Register LowXSFR (E9C0<sub>H</sub>+4\*n)**      **Reset Value: XXXX<sub>H</sub>**

#### Pn\_OMRL (n=6-11)

**Port n Output Modification Register LowXSFR (E9C0<sub>H</sub>+4\*n)**      **Reset Value: XXXX<sub>H</sub>**

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

Field	Bits	Type	Description
<b>PSx</b> (x = 0-7)	x	w	<b>Port Set Bit x</b> Setting this bit sets or toggles the corresponding bit in the port output register Pn_OUT (see <a href="#">Table 9-3</a> ). On a read access, this bit returns an undefined value.
<b>PCx</b> (x = 0-7)	x + 8	w	<b>Port Clear Bit x</b> Setting this bit clears or toggles the corresponding bit in the port output register Pn_OUT. (see <a href="#">Table 9-3</a> ). On a read access, this bit returns an undefined value.

### Function of the PCx and PSx bit fields

**Table 9-3      Function of the Bits PCx and PSx**

PCx	PSx	Function
0 or no write access	0 or no write access	Bit Pn_OUT.Px is not changed.
0 or no write access	1	Bit Pn_OUT.Px is set.
1	0 or no write access	Bit Pn_OUT.Px is cleared.
1	1	Bit Pn_OUT.Px is toggled.

*Note: If a bit position is not written (one out of two bytes not targeted by a byte write), the corresponding value is considered as 0. Toggling a bit requires one 16-bit write.*

## 9.2.4 Port Input Register

The port input register contains the values currently read at the input pins, also if a port line is assigned as output.

**Pn\_IN (n=0-11)**

**Port n Input Register**

**SFR (FF80<sub>H</sub>+2\*n)**

**Reset Value: 0000<sub>H</sub><sup>1)</sup>**

**P15\_IN**

**Port 15 Input Register**

**SFR (FF9E<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub><sup>1)</sup>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>P15</b>	<b>P14</b>	<b>P13</b>	<b>P12</b>	<b>P11</b>	<b>P10</b>	<b>P9</b>	<b>P8</b>	<b>P7</b>	<b>P6</b>	<b>P5</b>	<b>P4</b>	<b>P3</b>	<b>P2</b>	<b>P1</b>	<b>P0</b>
rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

<sup>1)</sup> Px bits for non implemented I/O lines are always read as 0.

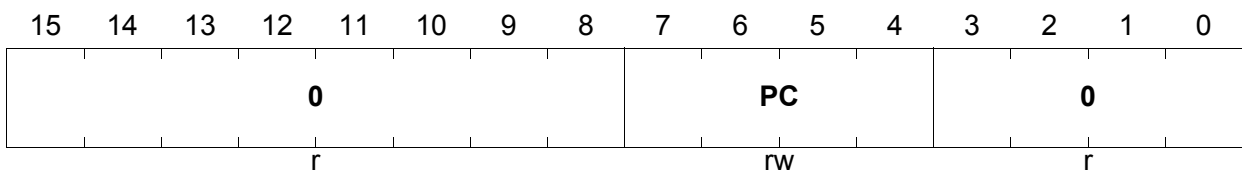
Field	Bits	Type	Description
<b>Px</b> <b>(x = 0-15)</b>	x	rh	<b>Port Input Bit x</b> This bit indicates the level at the input pin of port Pn, pin x. 0 <sub>B</sub> The input level of Pn.x is 0. 1 <sub>B</sub> The input level of Pn.x is 1.

## 9.2.5 Port Input/Output Control Registers

The port input/output control registers contain the bit fields to select the digital output and input driver characteristics, such as pull-up/down devices, port direction (input/output), open-drain and alternate output selections. The coding of the options is shown in [Table 9-4](#).

Depending on the port functionality not all of the input/output control registers may be implemented. The structure with one control bit field for each port pin located in different register offers the possibility to configure port pin functionality of a single pin without accessing some other PCx in the same register by word-oriented writes.

<b>P0_IOCRx (x=00-07)</b>	
<b>Port 0 Input/Output Control Register x XSFR (E800<sub>H</sub>+2*x)</b>	<b>Reset Value: 0000<sub>H</sub></b>
<b>P1_IOCRx (x=00-07)</b>	
<b>Port 1 Input/Output Control Register x XSFR (E820<sub>H</sub>+2*x)</b>	<b>Reset Value: 0000<sub>H</sub></b>
<b>P2_IOCRx (x=00-13)</b>	
<b>Port 2 Input/Output Control Register x XSFR (E840<sub>H</sub>+2*x)</b>	<b>Reset Value: 0000<sub>H</sub></b>
<b>P3_IOCRx (x=00-07)</b>	
<b>Port 3 Input/Output Control Register x XSFR (E860<sub>H</sub>+2*x)</b>	<b>Reset Value: 0000<sub>H</sub></b>
<b>P4_IOCRx (x=00-07)</b>	
<b>Port 4 Input/Output Control Register x XSFR (E880<sub>H</sub>+2*x)</b>	<b>Reset Value: 0000<sub>H</sub></b>
<b>P6_IOCRx (x=00-03)</b>	
<b>Port 6 Input/Output Control Register x XSFR (E8C0<sub>H</sub>+2*x)</b>	<b>Reset Value: 0000<sub>H</sub></b>
<b>P7_IOCRx (x=00-04)</b>	
<b>Port 7 Input/Output Control Register x XSFR (E8E0<sub>H</sub>+2*x)</b>	<b>Reset Value: 0000<sub>H</sub></b>
<b>P8_IOCRx (x=00-06)</b>	
<b>Port 8 Input/Output Control Register x XSFR (E900<sub>H</sub>+2*x)</b>	<b>Reset Value: 0000<sub>H</sub></b>
<b>P9_IOCRx (x=00-07)</b>	
<b>Port 9 Input/Output Control Register x XSFR (E920<sub>H</sub>+2*x)</b>	<b>Reset Value: 0000<sub>H</sub></b>
<b>P10_IOCRx (x=00-15)</b>	
<b>Port 10 Input/Output Control Register x XSFR (E940<sub>H</sub>+2*x)</b>	<b>Reset Value: 0000<sub>H</sub></b>
<b>P11_IOCRx (x=00-05)</b>	
<b>Port 11 Input/Output Control Register x XSFR (E960<sub>H</sub>+2*x)</b>	<b>Reset Value: 0000<sub>H</sub></b>



Field	Bits	Type	Description
<b>PC</b>	[7:4]	rw	<b>Port Input/Output Control Bit</b> see <a href="#">Table 9-4</a>
<b>0</b>	[3:0], [15:8]	r	reserved

### Coding of the PC bit field

The coding of the GPIO port behavior is done by the bit fields in the port control registers Pn\_IOCRx. There's a control bit field PC for each port pin. The bit fields PC are located in separate control registers in order to allow modifying a port pin (without influencing the others) with simple move operations.

*Note: When the pin direction is switched to output and the mode is test mode, the output characteristic must be push-pull only.*

**Table 9-4 PC Coding**

PC[3:0]	I/O	Selected Pull-up/down / Selected Output Function	Behavior in Power Saving Mode <sup>1)</sup>
0000 <sub>B</sub>	Direct Input	No pull device connected	Input value = Pn_OUT; no pull
0001 <sub>B</sub>		Pull-down device connected	Input value = 0; pull-down
0010 <sub>B</sub>		Pull-up device connected	Input value = 1; pull-up
0011 <sub>B</sub>		No pull device connected. In this mode Pn_OUT samples the pad input value continuously.	Input value = Pn_OUT; Pn_OUT always samples input value while not in power save mode = freeze of input value; no pull
0100 <sub>B</sub>	Inverted Input	No pull device connected	Input value = $\overline{\text{Pn\_OUT}}$ ; no pull
0101 <sub>B</sub>		Pull-down device connected	Input value = 1; pull-down
0110 <sub>B</sub>		Pull-up device connected	Input value = 0; pull-up
0111 <sub>B</sub>		No pull device connected In this mode Pn_OUT samples the pad input value continuously.	Input value = $\overline{\text{Pn\_OUT}}$ ; Pn_OUT always samples input value while not in power saving mode = freeze of input value; no pull <sup>2)</sup>

**Table 9-4 PC Coding**

<b>PC[3:0]</b>	<b>I/O</b>	<b>Selected Pull-up/down / Selected Output Function</b>	<b>Behavior in Power Saving Mode<sup>1)</sup></b>
1000 <sub>B</sub>	Output (Direct input) Push- pull	General purpose Output	Output driver off. Input Schmitt trigger off. Pn_OUT delivered to the internal logic; no pull
1001 <sub>B</sub>		Output function ALT1	
1010 <sub>B</sub>		Output function ALT2	
1011 <sub>B</sub>		Output function ALT3	
1100 <sub>B</sub>	Output (Direct input) Open- drain	General purpose Output	
1101 <sub>B</sub>		Output function ALT1	
1110 <sub>B</sub>		Output function ALT2	
1111 <sub>B</sub>		Output function ALT3	

<sup>1)</sup> In power saving mode, the input Schmitt trigger is always switched off. A defined input value is driven to the internal circuitry instead of the level detected at the input pin.

<sup>2)</sup> If the IOCR setting is “inverted input”, then an inverted signal Pn\_OUT is driven internally. The Pn\_OUT register itself always contains the real, non-inverted input value of the pin. See Figure 7-1 and Figure 7-2.

## 9.2.6 Port Digital Input Disable Register

Ports 5 and 15 have, additionally to the analog input functionality, digital input functionality too. In order to save switching of the internal Schmitt triggers of the digital inputs, they can be disabled by means of Px\_DIDIS Register.

### P5\_DIDIS

**Port 5 Digital Input Disable RegisterSFR (FE8A<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

### P15\_DIDIS

**Port 15 Digital Input Disable RegisterSFR (FE9E<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

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
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bit	Type	Description
<b>Py</b> <b>(y = 0-15)</b>	y	rw	Bit y Digital Input Control 0 <sub>B</sub> Digital input stage (schmitt trigger) is enabled. 1 <sub>B</sub> Digital input stage (schmitt trigger) is disabled, necessary if pin is used as analog input.



### **9.3 Port Description**

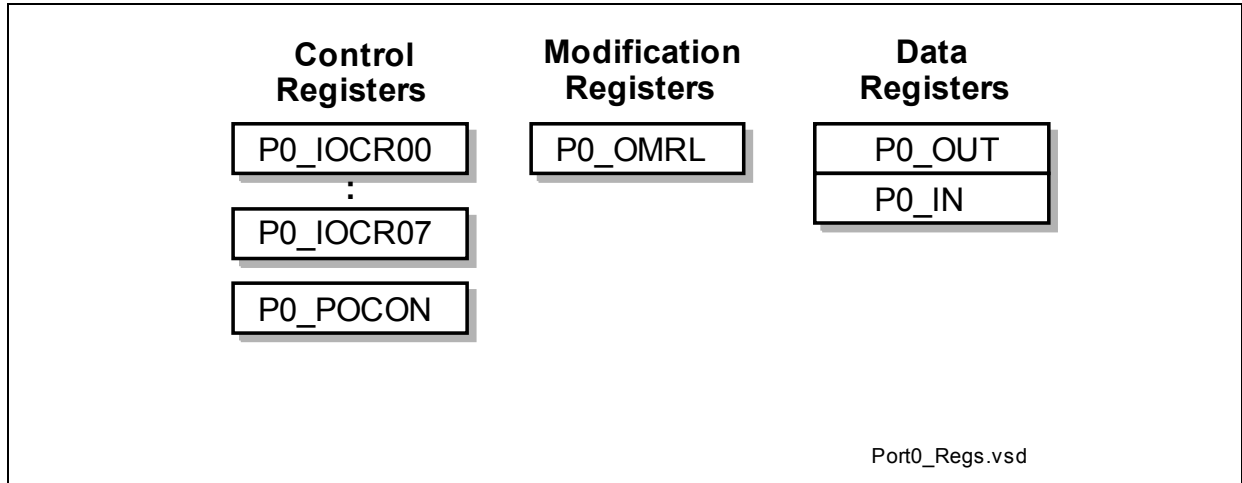
The bit positions in the port registers always start right-aligned. For example, a port comprising only 8 pins only uses the bit positions [7:0] of the corresponding register. The remaining bit positions are filled with 0 (r).

The pad driver mode registers may be different for each port. As a result, they are described independently for each port in the corresponding chapter.

### 9.3.1 Port 0

Port 0 is an 8-bit GPIO port. The registers of Port 0 are shown in [Figure 9-4](#).

For this port, all pins can be read as GPIO, from the Port Input Register.



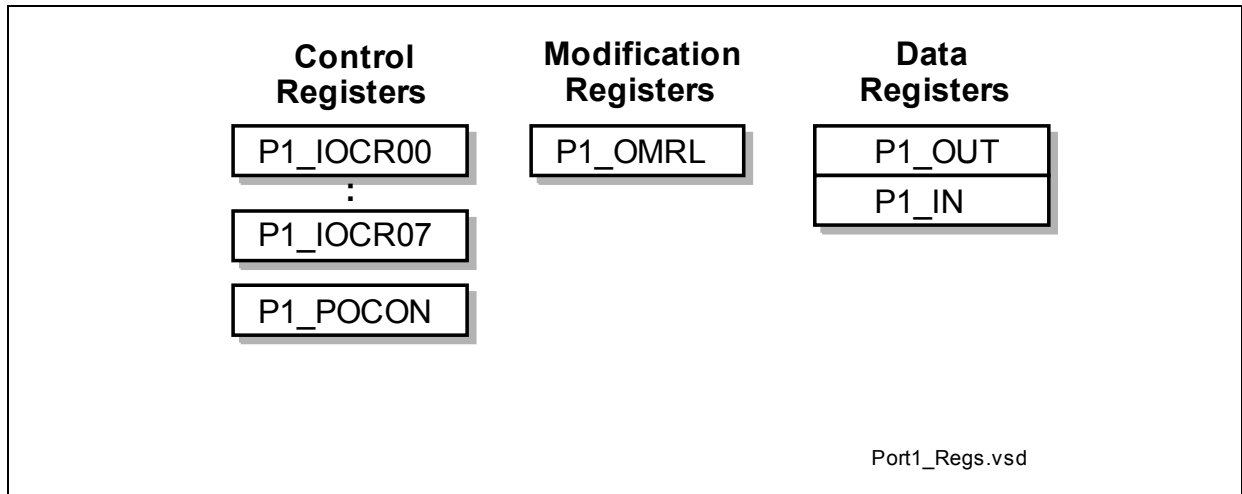
**Figure 9-4 Port 0 Register Overview**

**Table 9-5 Port 0 Registers**

Register Short Name	Register Long Name	Address Offset	Reset Value
P0_OUT	Port 0 Output Register	FFA2 <sub>H</sub>	0000 <sub>H</sub>
P0_IN	Port 0 Input Register	FF80 <sub>H</sub>	0000 <sub>H</sub>
P0_OMRL	Port 0 Output Modification Register Low	E9C0 <sub>H</sub>	XXXX <sub>H</sub>
P0_POCON	Port 0 Output Control Register	E8A0 <sub>H</sub>	0000 <sub>H</sub>
P0_IOCRR0	Port 0 Input/Output Control Register 0	E800 <sub>H</sub>	0000 <sub>H</sub>
P0_IOCRR1	Port 0 Input/Output Control Register 1	E802 <sub>H</sub>	0000 <sub>H</sub>
P0_IOCRR2	Port 0 Input/Output Control Register 2	E804 <sub>H</sub>	0000 <sub>H</sub>
P0_IOCRR3	Port 0 Input/Output Control Register 3	E806 <sub>H</sub>	0000 <sub>H</sub>
P0_IOCRR4	Port 0 Input/Output Control Register 4	E808 <sub>H</sub>	0000 <sub>H</sub>
P0_IOCRR5	Port 0 Input/Output Control Register 5	E80A <sub>H</sub>	0000 <sub>H</sub>
P0_IOCRR6	Port 0 Input/Output Control Register 6	E80C <sub>H</sub>	0000 <sub>H</sub>
P0_IOCRR7	Port 0 Input/Output Control Register 7	E80E <sub>H</sub>	0000 <sub>H</sub>

### 9.3.2 Port 1

Port 1 is an 8-bit GPIO port. The registers of Port 1 are shown in [Figure 9-5](#).



**Figure 9-5 Port 1 Register Overview**

For this port, all pins can be read as GPIO, from the Port Input Register.

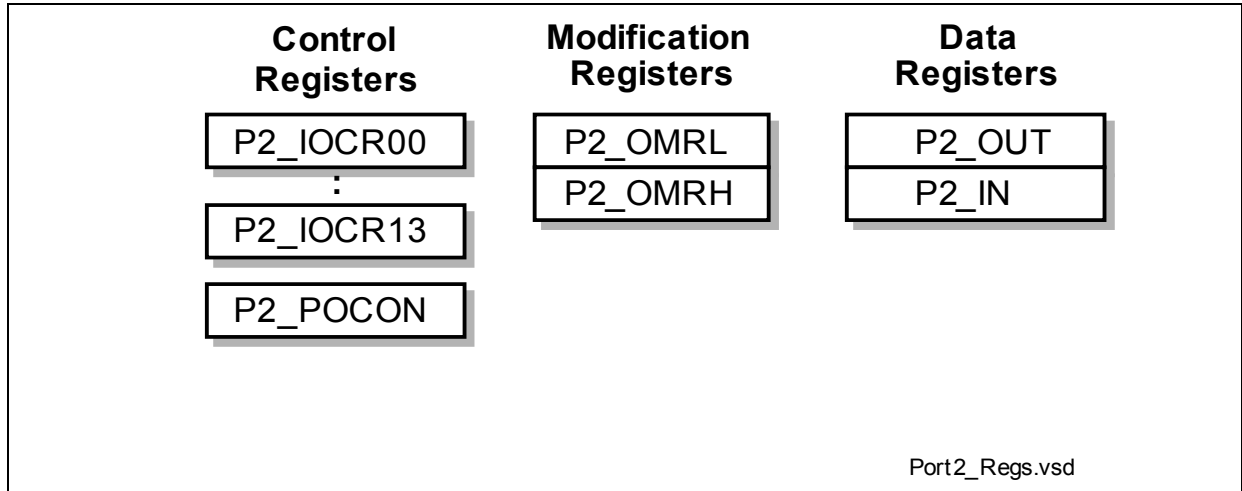
**Table 9-6 Port 1 Registers**

Register Short Name	Register Long Name	Address Offset	Reset Value
P1_OUT	Port 1 Output Register	FFA4 <sub>H</sub>	0000 <sub>H</sub>
P1_IN	Port 1 Input Register	FF82 <sub>H</sub>	0000 <sub>H</sub>
P1_OMRL	Port 1 Output Modification Register Low	E9C4 <sub>H</sub>	XXXX <sub>H</sub>
P1_POCON	Port 1 Output Control Register	E8A2 <sub>H</sub>	0000 <sub>H</sub>
P1_IOCRR00	Port 1 Input/Output Control Register 0	E820 <sub>H</sub>	0000 <sub>H</sub>
P1_IOCRR01	Port 1 Input/Output Control Register 1	E822 <sub>H</sub>	0000 <sub>H</sub>
P1_IOCRR02	Port 1 Input/Output Control Register 2	E824 <sub>H</sub>	0000 <sub>H</sub>
P1_IOCRR03	Port 1 Input/Output Control Register 3	E826 <sub>H</sub>	0000 <sub>H</sub>
P1_IOCRR04	Port 1 Input/Output Control Register 4	E828 <sub>H</sub>	0000 <sub>H</sub>
P1_IOCRR05	Port 1 Input/Output Control Register 5	E82A <sub>H</sub>	0000 <sub>H</sub>
P1_IOCRR06	Port 1 Input/Output Control Register 6	E82C <sub>H</sub>	0000 <sub>H</sub>
P1_IOCRR07	Port 1 Input/Output Control Register 7	E82E <sub>H</sub>	0000 <sub>H</sub>

### 9.3.3 Port 2

Port 2 is an 14-bit GPIO port. The registers of Port 2 are shown in [Figure 9-6](#).

For this port, all pins can be read as GPIO, from the Port Input Register.



**Figure 9-6 Port 2 Register Overview**

**Table 9-7 Port 2 Registers**

Register Short Name	Register Long Name	Address Offset	Reset Value
P2_OUT	Port 2 Output Register	FFA6 <sub>H</sub>	0000 <sub>H</sub>
P2_IN	Port 2 Input Register	FF84 <sub>H</sub>	0000 <sub>H</sub>
P2_OMRL	Port 2 Output Modification Register Low	E9C8 <sub>H</sub>	XXXX <sub>H</sub>
P2_OMRH	Port 2 Output Modification Register High	E9CA <sub>H</sub>	XXXX <sub>H</sub>
P2_POCON	Port 2 Output Control Register	E8A4 <sub>H</sub>	0000 <sub>H</sub>
P2_IOCRR00	Port 2 Input/Output Control Register 0	E840 <sub>H</sub>	0000 <sub>H</sub>
P2_IOCRR01	Port 2 Input/Output Control Register 1	E842 <sub>H</sub>	0000 <sub>H</sub>
P2_IOCRR02	Port 2 Input/Output Control Register 2	E844 <sub>H</sub>	0000 <sub>H</sub>
P2_IOCRR03	Port 2 Input/Output Control Register 3	E846 <sub>H</sub>	0000 <sub>H</sub>
P2_IOCRR04	Port 2 Input/Output Control Register 4	E848 <sub>H</sub>	0000 <sub>H</sub>
P2_IOCRR05	Port 2 Input/Output Control Register 5	E84A <sub>H</sub>	0000 <sub>H</sub>
P2_IOCRR06	Port 2 Input/Output Control Register 6	E84C <sub>H</sub>	0000 <sub>H</sub>
P2_IOCRR07	Port 2 Input/Output Control Register 7	E84E <sub>H</sub>	0000 <sub>H</sub>
P2_IOCRR08	Port 2 Input/Output Control Register 8	E850 <sub>H</sub>	0000 <sub>H</sub>
P2_IOCRR09	Port 2 Input/Output Control Register 9	E852 <sub>H</sub>	0000 <sub>H</sub>

**Table 9-7 Port 2 Registers (cont'd)**

<b>Register Short Name</b>	<b>Register Long Name</b>	<b>Address Offset</b>	<b>Reset Value</b>
P2_IOCRR10	Port 2 Input/Output Control Register 10	E854 <sub>H</sub>	0000 <sub>H</sub>
P2_IOCRR11	Port 2 Input/Output Control Register 11	E856 <sub>H</sub>	0000 <sub>H</sub>
P2_IOCRR12	Port 2 Input/Output Control Register 12	E858 <sub>H</sub>	0000 <sub>H</sub>
P2_IOCRR13	Port 2 Input/Output Control Register 13	E85A <sub>H</sub>	0000 <sub>H</sub>

### **The CLKOUT Pad P2.8**

In order to drive high frequency clock signals, a strong driver is connected in parallel to the normal output driver of P2.8. It is enabled instead of the standard driver while bitfield P2\_POCON.PDM3 = xx1<sub>B</sub>.

The strong clock driver only operates in strong driver sharp edge mode, i.e. it is not controlled by the driver-strength settings (P2\_POCON.PDM2) for the standard driver. It has no pull devices but can be switched to input or output via register P2\_IOCRR08.

### **Output Control for Pins P2.[13:12]**

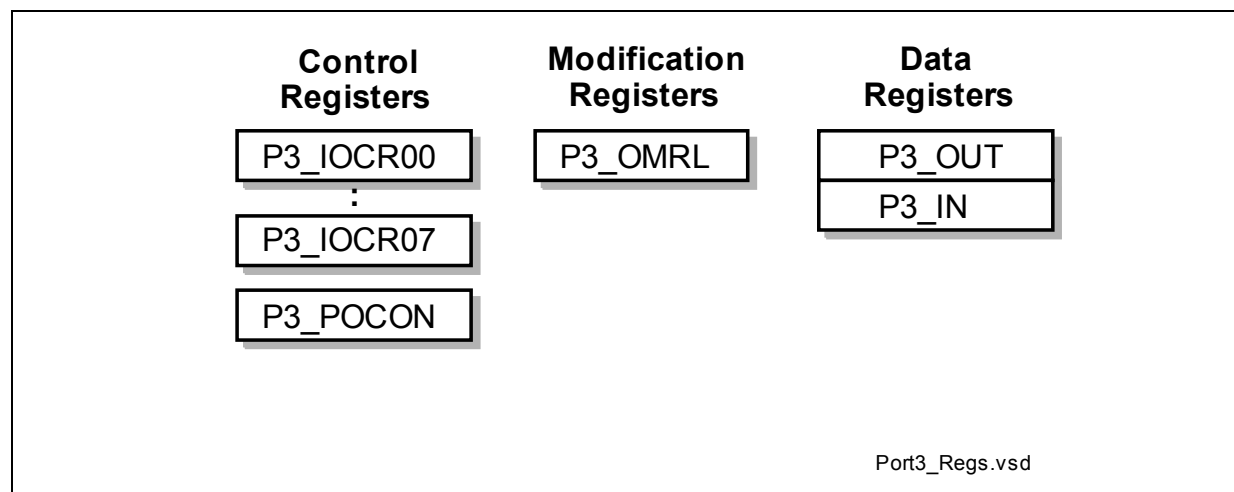
Because bitfield P2\_POCON.PDM3 controls the strong clock driver of P2.8, the driver mode of pins P2.[13:12] is controlled by the bitfield P2\_POCON.PDM2, together with pins P2.[11:8].

The power saving behaviour of pins P2.[13:12] is controlled by bit P2\_POCON.PPS3.

### 9.3.4 Port 3

Port 3 is an 8-bit GPIO port. The registers of Port 3 are shown in [Figure 9-7](#).

For this port, all pins can be read as GPIO, from the Port Input Register.



**Figure 9-7 Port 3 Register Overview**

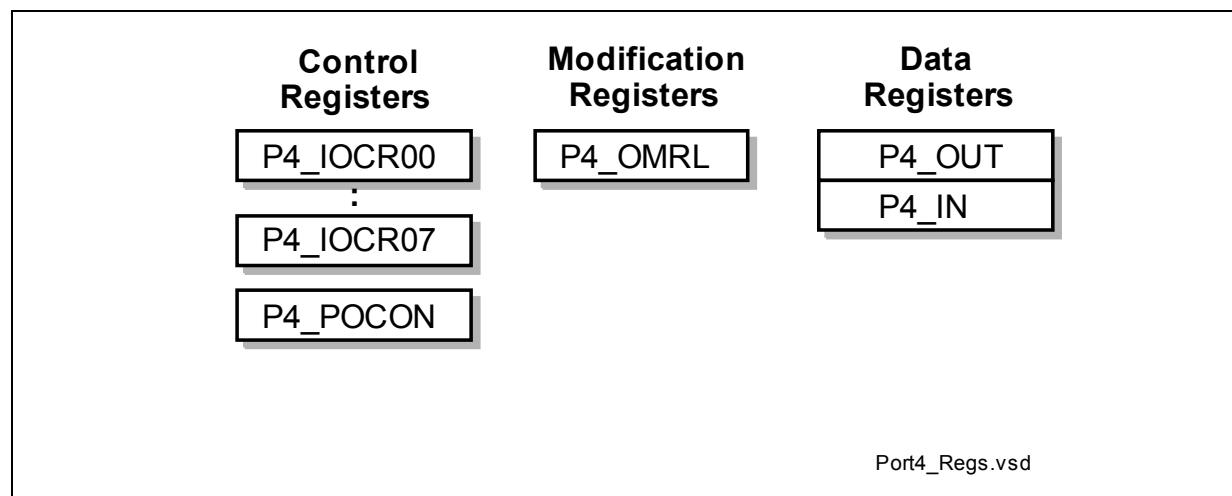
**Table 9-8 Port 3 Registers**

Register Short Name	Register Long Name	Address Offset	Reset Value
P3_OUT	Port 3 Output Register	FFA8 <sub>H</sub>	0000 <sub>H</sub>
P3_IN	Port 3 Input Register	FF86 <sub>H</sub>	0000 <sub>H</sub>
P3_OMRL	Port 3 Output Modification Register Low	E9CC <sub>H</sub>	XXXX <sub>H</sub>
P3_POCON	Port 3 Output Control Register	E8A6 <sub>H</sub>	0000 <sub>H</sub>
P3_IOCRR0	Port 3 Input/Output Control Register 0	E860 <sub>H</sub>	0000 <sub>H</sub>
P3_IOCRR1	Port 3 Input/Output Control Register 1	E862 <sub>H</sub>	0000 <sub>H</sub>
P3_IOCRR2	Port 3 Input/Output Control Register 2	E864 <sub>H</sub>	0000 <sub>H</sub>
P3_IOCRR3	Port 3 Input/Output Control Register 3	E866 <sub>H</sub>	0000 <sub>H</sub>
P3_IOCRR4	Port 3 Input/Output Control Register 4	E868 <sub>H</sub>	0000 <sub>H</sub>
P3_IOCRR5	Port 3 Input/Output Control Register 5	E86A <sub>H</sub>	0000 <sub>H</sub>
P3_IOCRR6	Port 3 Input/Output Control Register 6	E86C <sub>H</sub>	0000 <sub>H</sub>
P3_IOCRR7	Port 3 Input/Output Control Register 7	E86E <sub>H</sub>	0000 <sub>H</sub>

### 9.3.5 Port 4

Port 4 is an 8-bit GPIO port. The registers of Port 4 are shown in [Figure 9-8](#).

For this port, all pins can be read as GPIO, from the Port Input Register.



**Figure 9-8 Port 4 Register Overview**

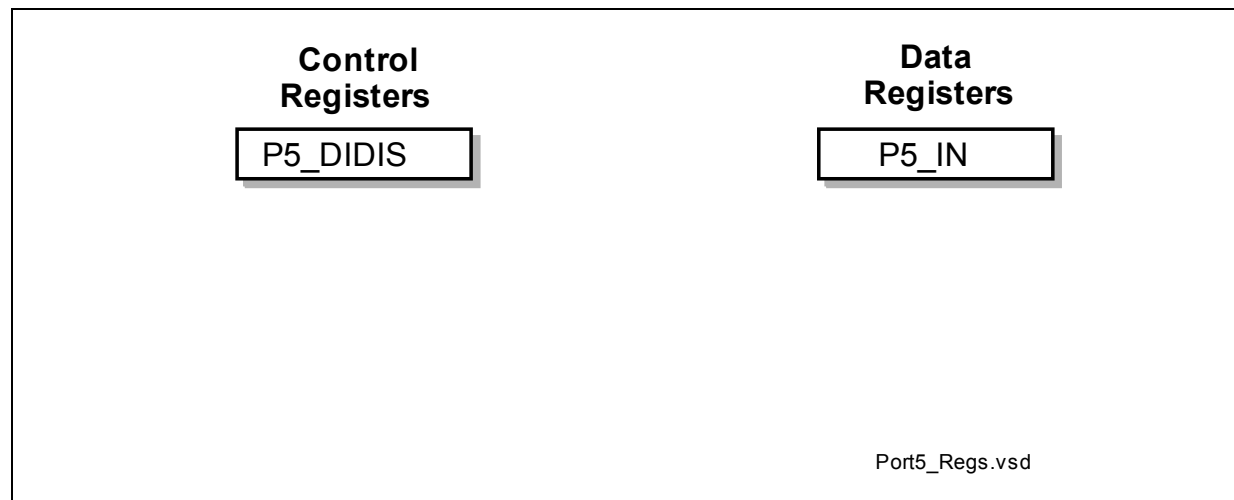
**Table 9-9 Port 4 Registers**

Register Short Name	Register Long Name	Address Offset	Reset Value
P4_OUT	Port 4 Output Register	FFAA <sub>H</sub>	0000 <sub>H</sub>
P4_IN	Port 4 Input Register	FF88 <sub>H</sub>	0000 <sub>H</sub>
P4_OMRL	Port 4 Output Modification Register Low	E9D0 <sub>H</sub>	XXXX <sub>H</sub>
P4_POCON	Port 4 Output Control Register	E8A8 <sub>H</sub>	0000 <sub>H</sub>
P4_IOCRR00	Port 4 Input/Output Control Register 0	E880 <sub>H</sub>	0000 <sub>H</sub>
P4_IOCRR01	Port 4 Input/Output Control Register 1	E882 <sub>H</sub>	0000 <sub>H</sub>
P4_IOCRR02	Port 4 Input/Output Control Register 2	E884 <sub>H</sub>	0000 <sub>H</sub>
P4_IOCRR03	Port 4 Input/Output Control Register 3	E886 <sub>H</sub>	0000 <sub>H</sub>
P4_IOCRR04	Port 4 Input/Output Control Register 4	E888 <sub>H</sub>	0000 <sub>H</sub>
P4_IOCRR05	Port 4 Input/Output Control Register 5	E88A <sub>H</sub>	0000 <sub>H</sub>
P4_IOCRR06	Port 4 Input/Output Control Register 6	E88C <sub>H</sub>	0000 <sub>H</sub>
P4_IOCRR07	Port 4 Input/Output Control Register 7	E88E <sub>H</sub>	0000 <sub>H</sub>

### 9.3.6 Port 5

Port 5 is a 16-bit analog or digital input port.

To use the Port 5 as an analog input, the Schmitt trigger in the input stage must be disabled. This is achieved by setting the corresponding bit in the register P5\_DIDIS.



**Figure 9-9 Port 5 Register Overview**

**Table 9-10 Port 5 Registers**

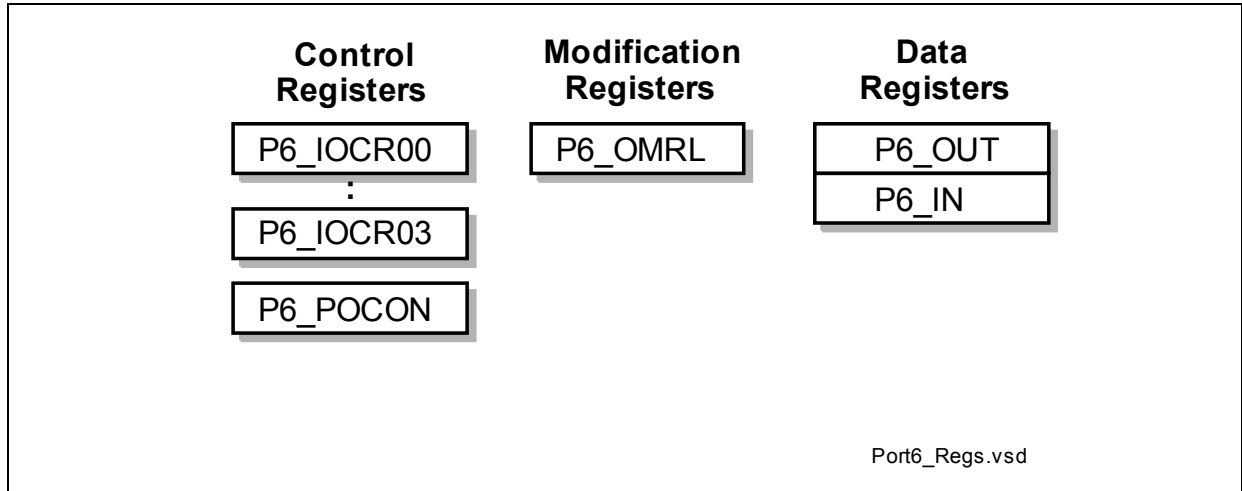
Register Short Name	Register Long Name	Address Offset	Reset Value
P5_IN	Port 5 Input Register	FF8A <sub>H</sub>	0000 <sub>H</sub>
P5_DIDIS	Port 5 Digital Input Disable Register	FE8A <sub>H</sub>	0000 <sub>H</sub>



### 9.3.7 Port 6

Port 6 is an 4-bit GPIO port. The registers of Port 6 are shown in [Figure 9-10](#).

For this port, all pins can be read as GPIO, from the Port Input Register.



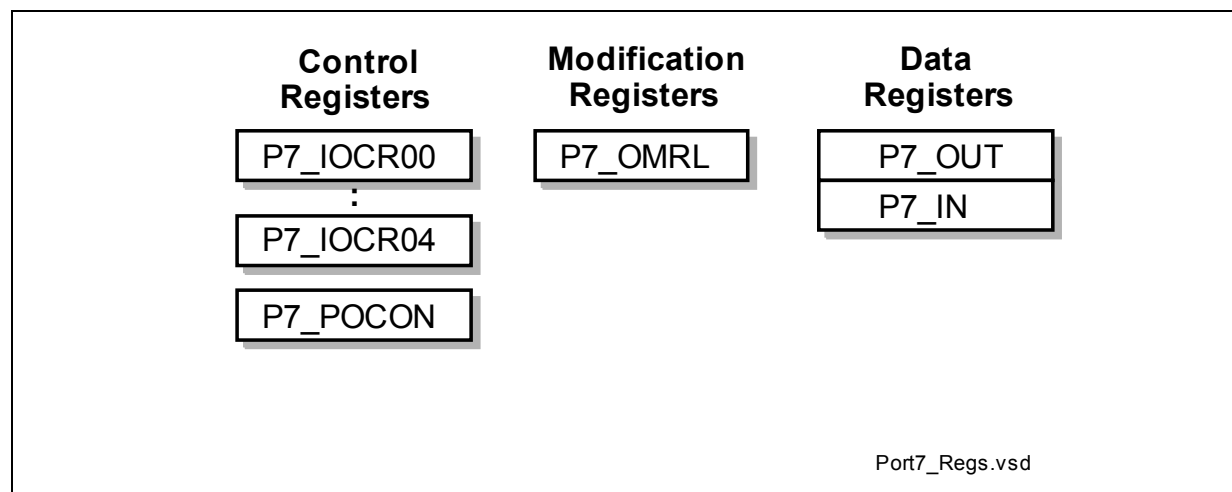
**Figure 9-10 Port 6 Register Overview**

**Table 9-11 Port 6 Registers**

Register Short Name	Register Long Name	Address Offset	Reset Value
P6_OUT	Port 6 Output Register	FFAE <sub>H</sub>	0000 <sub>H</sub>
P6_IN	Port 6 Input Register	FF8C <sub>H</sub>	0000 <sub>H</sub>
P6_OMRL	Port 6 Output Modification Register Low	E9D8 <sub>H</sub>	XXXX <sub>H</sub>
P6_POCON	Port 6 Output Control Register	E8AC <sub>H</sub>	0000 <sub>H</sub>
P6_IOCRR00	Port 6 Input/Output Control Register 0	E8C0 <sub>H</sub>	0000 <sub>H</sub>
P6_IOCRR01	Port 6 Input/Output Control Register 1	E8C2 <sub>H</sub>	0000 <sub>H</sub>
P6_IOCRR02	Port 6 Input/Output Control Register 2	E8C4 <sub>H</sub>	0000 <sub>H</sub>
P6_IOCRR03	Port 6 Input/Output Control Register 3	E8C6 <sub>H</sub>	0000 <sub>H</sub>

### 9.3.8 Port 7

Port 7 is a 5-bit GPIO port. The port registers of Port 7 are shown in [Figure 9-11](#). For this port, all pins can be read as GPIO, from the Port Input Register.



**Figure 9-11 Port 7 Register Overview**

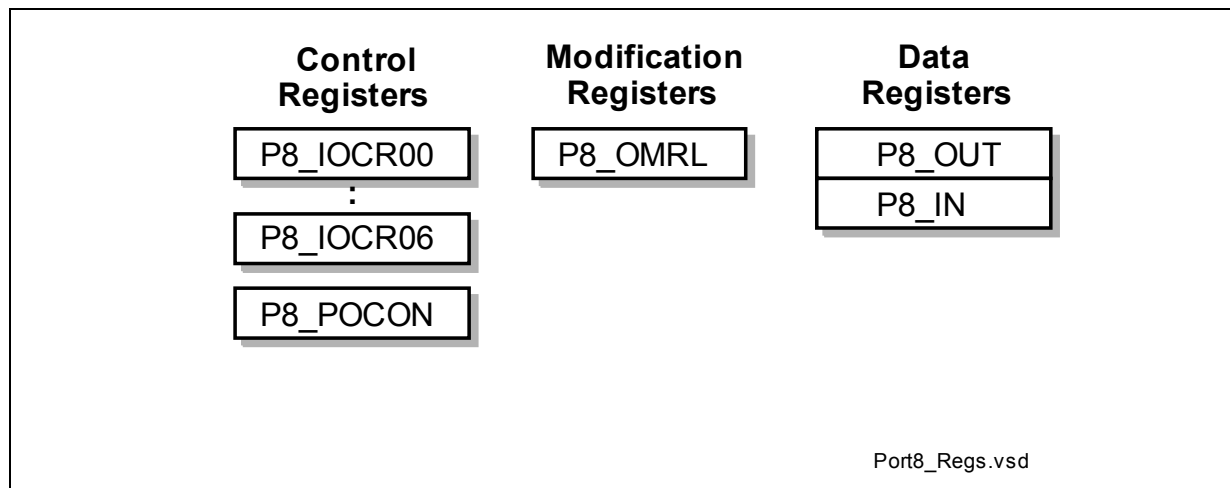
**Table 9-12 Port 7 Registers**

Register Short Name	Register Long Name	Address Offset	Reset Value
P7_OUT	Port 7 Output Register	FFB0 <sub>H</sub>	0000 <sub>H</sub>
P7_IN	Port 7 Input Register	FF8E <sub>H</sub>	0000 <sub>H</sub>
P7_OMRL	Port 7 Output Modification Register Low	E9DC <sub>H</sub>	XXXX <sub>H</sub>
P7_POCON	Port 7 Output Control Register	E8AE <sub>H</sub>	0000 <sub>H</sub>
P7_IOCRR0	Port 7 Input/Output Control Register 0	E8E0 <sub>H</sub>	0000 <sub>H</sub>
P7_IOCRR1	Port 7 Input/Output Control Register 1	E8E2 <sub>H</sub>	0000 <sub>H</sub>
P7_IOCRR2	Port 7 Input/Output Control Register 2	E8E4 <sub>H</sub>	0000 <sub>H</sub>
P7_IOCRR3	Port 7 Input/Output Control Register 3	E8E6 <sub>H</sub>	0000 <sub>H</sub>
P7_IOCRR4	Port 7 Input/Output Control Register 4	E8E8 <sub>H</sub>	0000 <sub>H</sub>

### 9.3.9 Port 8

Port 8 is an 7-bit GPIO port. The registers of Port 8 are shown in [Figure 9-12](#).

For this port, all pins can be read as GPIO, from the Port Input Register.



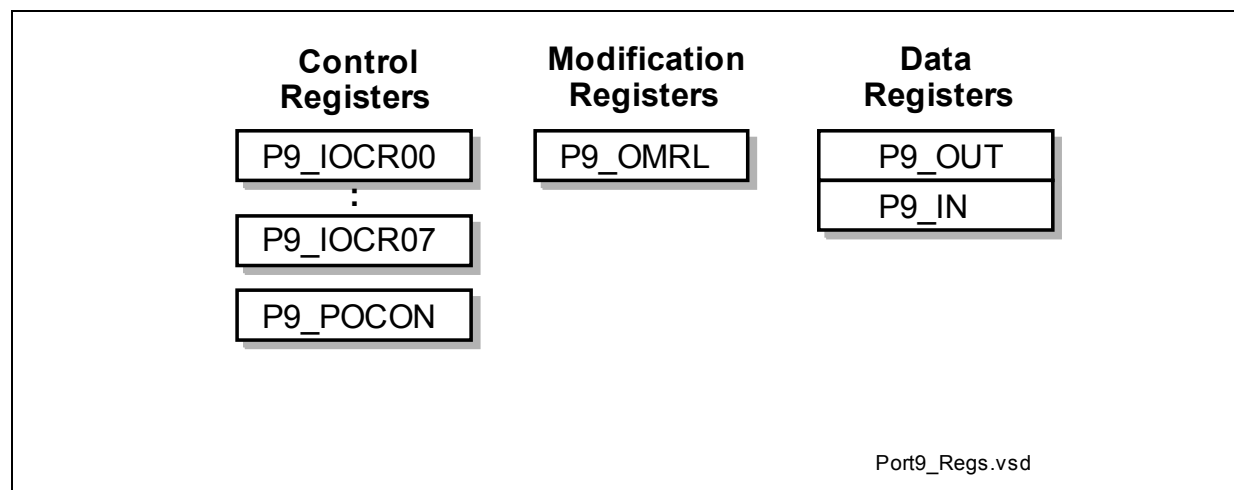
**Figure 9-12 Port 8 Register Overview**

**Table 9-13 Port 8 Registers**

Register Short Name	Register Long Name	Address Offset	Reset Value
P8_OUT	Port 8 Output Register	FFB2 <sub>H</sub>	0000 <sub>H</sub>
P8_IN	Port 8 Input Register	FF90 <sub>H</sub>	0000 <sub>H</sub>
P8_OMRL	Port 8 Output Modification Register Low	E9E0 <sub>H</sub>	XXXX <sub>H</sub>
P8_POCON	Port 8 Output Control Register	E8B0 <sub>H</sub>	0000 <sub>H</sub>
P8_IOCRO0	Port 8 Input/Output Control Register 0	E900 <sub>H</sub>	0000 <sub>H</sub>
P8_IOCRO1	Port 8 Input/Output Control Register 1	E902 <sub>H</sub>	0000 <sub>H</sub>
P8_IOCRO2	Port 8 Input/Output Control Register 2	E904 <sub>H</sub>	0000 <sub>H</sub>
P8_IOCRO3	Port 8 Input/Output Control Register 3	E906 <sub>H</sub>	0000 <sub>H</sub>
P8_IOCRO4	Port 8 Input/Output Control Register 4	E908 <sub>H</sub>	0000 <sub>H</sub>
P8_IOCRO5	Port 8 Input/Output Control Register 5	E90A <sub>H</sub>	0000 <sub>H</sub>
P8_IOCRO6	Port 8 Input/Output Control Register 6	E90C <sub>H</sub>	0000 <sub>H</sub>

### 9.3.10 Port 9

Port 9 is an 8-bit GPIO port. The port registers of Port 9 are shown in [Figure 9-13](#). For this port, all pins can be read as GPIO, from the Port Input Register.



**Figure 9-13 Port 9 Register Overview**

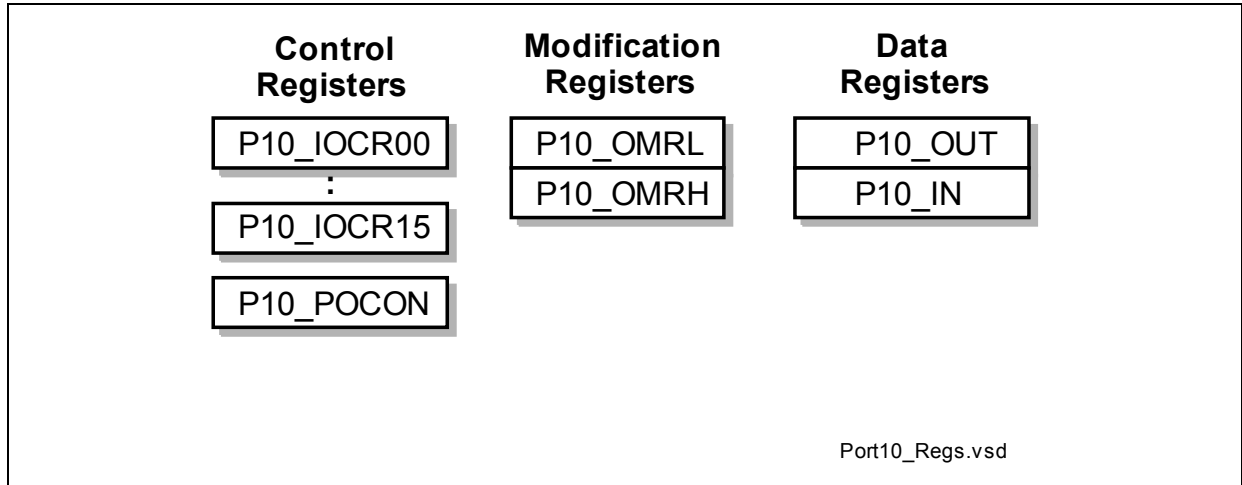
**Table 9-14 Port 9 Registers**

Register Short Name	Register Long Name	Address Offset	Reset Value
P9_OUT	Port 9 Output Register	FFB4 <sub>H</sub>	0000 <sub>H</sub>
P9_IN	Port 9 Input Register	FF92 <sub>H</sub>	0000 <sub>H</sub>
P9_OMRL	Port 9 Output Modification Register Low	E9E4 <sub>H</sub>	XXXX <sub>H</sub>
P9_POCON	Port 9 Output Control Register	E8B2 <sub>H</sub>	0000 <sub>H</sub>
P9_IOCRR00	Port 9 Input/Output Control Register 0	E920 <sub>H</sub>	0000 <sub>H</sub>
P9_IOCRR01	Port 9 Input/Output Control Register 1	E922 <sub>H</sub>	0000 <sub>H</sub>
P9_IOCRR02	Port 9 Input/Output Control Register 2	E924 <sub>H</sub>	0000 <sub>H</sub>
P9_IOCRR03	Port 9 Input/Output Control Register 3	E926 <sub>H</sub>	0000 <sub>H</sub>
P9_IOCRR04	Port 9 Input/Output Control Register 4	E928 <sub>H</sub>	0000 <sub>H</sub>
P9_IOCRR05	Port 9 Input/Output Control Register 5	E92A <sub>H</sub>	0000 <sub>H</sub>
P9_IOCRR06	Port 9 Input/Output Control Register 6	E92C <sub>H</sub>	0000 <sub>H</sub>
P9_IOCRR07	Port 9 Input/Output Control Register 7	E92E <sub>H</sub>	0000 <sub>H</sub>

### 9.3.11 Port 10

Port 10 is a 16-bit GPIO port. The registers of Port 10 are shown in [Figure 9-14](#).

For this port, all pins can be read as GPIO, from the Port Input Register.



**Figure 9-14 Port 10 Register Overview**

**Table 9-15 Port 10 Registers**

Register Short Name	Register Long Name	Address Offset	Reset Value
P10_OUT	Port 10 Output Register	FFB6 <sub>H</sub>	0000 <sub>H</sub>
P10_IN	Port 10 Input Register	FF94 <sub>H</sub>	0000 <sub>H</sub>
P10_OMRL	Port 10 Output Modification Register Low	E9E8 <sub>H</sub>	XXXX <sub>H</sub>
P10_OMRH	Port 10 Output Modification Register High	E9EA <sub>H</sub>	XXXX <sub>H</sub>
P10_POCON	Port 10 Output Control Register	E8B4 <sub>H</sub>	0000 <sub>H</sub>
P10_IOCRR0	Port 10 Input/Output Control Register 0	E940 <sub>H</sub>	0000 <sub>H</sub>
P10_IOCRR1	Port 10 Input/Output Control Register 1	E942 <sub>H</sub>	0000 <sub>H</sub>
P10_IOCRR2	Port 10 Input/Output Control Register 2	E944 <sub>H</sub>	0000 <sub>H</sub>
P10_IOCRR3	Port 10 Input/Output Control Register 3	E946 <sub>H</sub>	0000 <sub>H</sub>
P10_IOCRR4	Port 10 Input/Output Control Register 4	E948 <sub>H</sub>	0000 <sub>H</sub>
P10_IOCRR5	Port 10 Input/Output Control Register 5	E94A <sub>H</sub>	0000 <sub>H</sub>
P10_IOCRR6	Port 10 Input/Output Control Register 6	E94C <sub>H</sub>	0000 <sub>H</sub>
P10_IOCRR7	Port 10 Input/Output Control Register 7	E94E <sub>H</sub>	0000 <sub>H</sub>
P10_IOCRR8	Port 10 Input/Output Control Register 8	E950 <sub>H</sub>	0000 <sub>H</sub>
P10_IOCRR9	Port 10 Input/Output Control Register 9	E952 <sub>H</sub>	0000 <sub>H</sub>

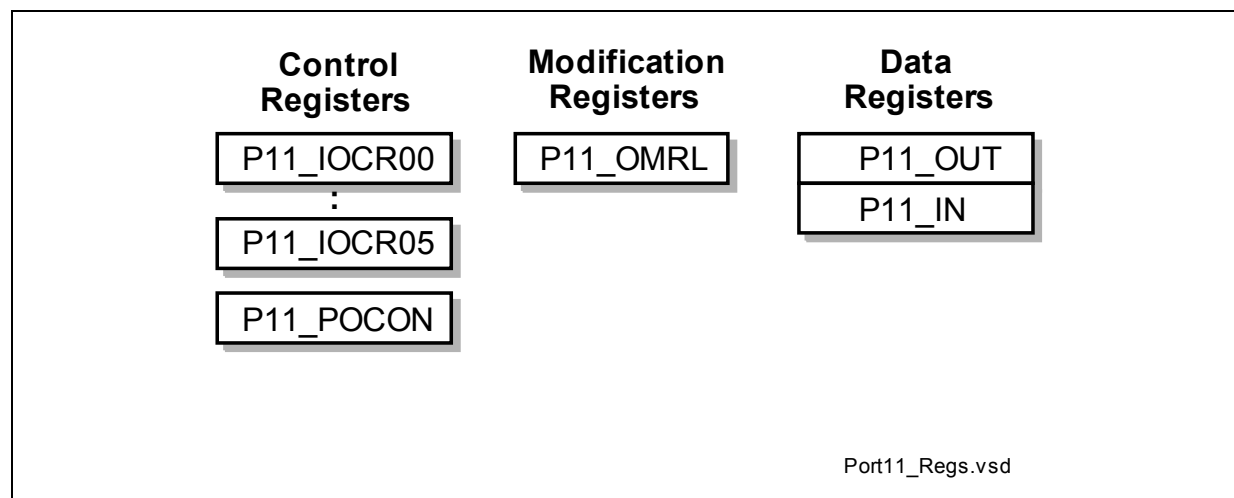
**Table 9-15 Port 10 Registers (cont'd)**

<b>Register Short Name</b>	<b>Register Long Name</b>	<b>Address Offset</b>	<b>Reset Value</b>
P10_IOC10	Port 10 Input/Output Control Register 10	E954 <sub>H</sub>	0000 <sub>H</sub>
P10_IOC11	Port 10 Input/Output Control Register 11	E956 <sub>H</sub>	0000 <sub>H</sub>
P10_IOC12	Port 10 Input/Output Control Register 12	E958 <sub>H</sub>	0000 <sub>H</sub>
P10_IOC13	Port 10 Input/Output Control Register 13	E95A <sub>H</sub>	0000 <sub>H</sub>
P10_IOC14	Port 10 Input/Output Control Register 14	E95C <sub>H</sub>	0000 <sub>H</sub>
P10_IOC15	Port 10 Input/Output Control Register 15	E95E <sub>H</sub>	0000 <sub>H</sub>

### 9.3.12 Port 11

Port 11 is a 6-bit GPIO port. The registers of Port 11 are shown in [Figure 9-15](#).

For this port, all pins can be read as GPIO, from the Port Input Register.



**Figure 9-15 Port 11 Register Overview**

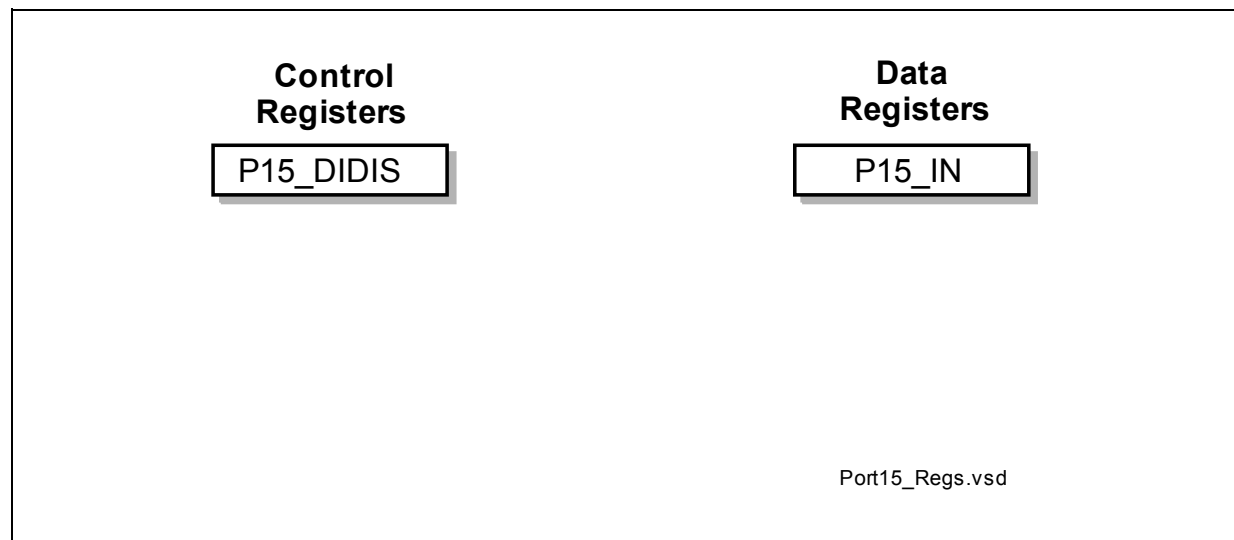
**Table 9-16 Port 11 Registers**

Register Short Name	Register Long Name	Address Offset	Reset Value
P11_OUT	Port 11 Output Register	FFB8 <sub>H</sub>	0000 <sub>H</sub>
P11_IN	Port 11 Input Register	FF96 <sub>H</sub>	0000 <sub>H</sub>
P11_OMRL	Port 11 Output Modification Register Low	E9EC <sub>H</sub>	XXXX <sub>H</sub>
P11_POCON	Port 11 Output Control Register	E8B6 <sub>H</sub>	0000 <sub>H</sub>
P11_IOCRR00	Port 11 Input/Output Control Register 0	E960 <sub>H</sub>	0000 <sub>H</sub>
P11_IOCRR01	Port 11 Input/Output Control Register 1	E962 <sub>H</sub>	0000 <sub>H</sub>
P11_IOCRR02	Port 11 Input/Output Control Register 2	E964 <sub>H</sub>	0000 <sub>H</sub>
P11_IOCRR03	Port 11 Input/Output Control Register 3	E966 <sub>H</sub>	0000 <sub>H</sub>
P11_IOCRR04	Port 11 Input/Output Control Register 4	E968 <sub>H</sub>	0000 <sub>H</sub>
P11_IOCRR05	Port 11 Input/Output Control Register 5	E96A <sub>H</sub>	0000 <sub>H</sub>

### 9.3.13 Port 15

Port 15 is an 8-bit analog or digital input port.

To use the Port 15 as an analog input, the Schmitt trigger in the input stage must be disabled. This is achieved by setting the corresponding bit in the register P15\_DIDIS.



**Figure 9-16 Port 15 Register Overview**

**Table 9-17 Port 15 Registers**

<b>Register Short Name</b>	<b>Register Long Name</b>	<b>Address Offset</b>	<b>Reset Value</b>
P15_IN	Port 15 Input Register	FF9E <sub>H</sub>	0000 <sub>H</sub>
P15_DIDIS	Port 15 Digital Input Disable Register	FE9E <sub>H</sub>	0000 <sub>H</sub>



## 9.4 Pin Description

Each port pin of the XE16xyM can serve several functions of different modules. Also, most functions are available on several port pins. This enables an application so select the optimal connections for its specific circumstances.

A pin can output its own port output signal or one of up to three signals coming from the peripherals. Its input signal is available in its own input register and at several peripherals.

*Note: Output signals are selected at the respective port pin, input signals are selected at the respective peripheral.*

Optionally a pin can be fully controlled by a peripheral, in case the peripheral is enabled (for example, EBC).

**Table 9-18** summarizes the various functions of each port and pin of the XE16xyM. The 'Pin' column references to the PG-LQFP-144 package.

### Notes to Pin Definitions

1. **Ctrl.:** The output signal for a port pin is selected via bitfield PC in the associated register Px\_IOCry. Output O0 is selected by setting the respective bitfield PC to 1x00<sub>B</sub>, output O1 is selected by 1x01<sub>B</sub>, etc.  
Output signal OH is controlled by hardware.
2. **Type:** Indicates the employed pad type (St=standard pad, Sp=special pad, DP=double pad, In=input pad, PS=power supply) and its power supply domain (A, B, M, 1).

**Table 9-18 Pin Definitions and Functions**

Pin	Symbol	Ctrl.	Type	Function
3	TESTM	I	In/B	<b>Testmode Enable</b> Enables factory test modes, must be held HIGH for normal operation (connect to $V_{DDPB}$ ). An internal pullup device will hold this pin high when nothing is driving it.
4	P7.2	O0 / I	St/B	<b>Bit 2 of Port 7, General Purpose Input/Output</b>
	EMUX0	O1	St/B	<b>External Analog MUX Control Output 0 (ADC1)</b>
	TxDC4	O2	St/B	<b>CAN Node 4 Transmit Data Output</b>
	TxDC5	O3	St/B	<b>CAN Node 5 Transmit Data Output</b>
	CCU62_CCP OS0A	I	St/B	<b>CCU62 Position Input 0</b>
	TDI_C	I	St/B	<b>JTAG Test Data Input</b>

**Table 9-18 Pin Definitions and Functions (cont'd)**

Pin	Symbol	Ctrl.	Type	Function
5	P8.4	O0 / I	St/B	<b>Bit 4 of Port 8, General Purpose Input/Output</b>
	CCU60_COU T61	O1	St/B	<b>CCU60 Channel 1 Output</b>
	CCU62_CC6 1	O2	St/B	<b>CCU62 Channel 1 Output</b>
	TMS_D	I	St/B	<b>JTAG Test Mode Selection Input</b>
	CCU62_CC6 1INB	I	St/B	<b>CCU62 Channel 1 Input</b>
6	$\overline{\text{TRST}}$	I	In/B	<b>Test-System Reset Input</b> For normal system operation, pin $\overline{\text{TRST}}$ should be held low. A high level at this pin at the rising edge of $\overline{\text{PORST}}$ activates the XE16xyM's debug system. In this case, pin $\overline{\text{TRST}}$ must be driven low once to reset the debug system. An internal pulldown device will hold this pin low when nothing is driving it.
7	P8.3	O0 / I	St/B	<b>Bit 3 of Port 8, General Purpose Input/Output</b>
	CCU60_COU T60	O1	St/B	<b>CCU60 Channel 0 Output</b>
	CCU62_CC6 0	O2	St/B	<b>CCU62 Channel 0 Output</b>
	TDI_D	I	St/B	<b>JTAG Test Data Input</b>
	CCU62_CC6 0INB	I	St/B	<b>CCU62 Channel 0 Input</b>
8	P7.0	O0 / I	St/B	<b>Bit 0 of Port 7, General Purpose Input/Output</b>
	T3OUT	O1	St/B	<b>GPT12E Timer T3 Toggle Latch Output</b>
	T6OUT	O2	St/B	<b>GPT12E Timer T6 Toggle Latch Output</b>
	TDO_A	OH / I	St/B	<b>JTAG Test Data Output / DAP1 Input/Output</b>
	ESR2_1	I	St/B	<b>ESR2 Trigger Input 1</b>
	RxDC4B	I	St/B	<b>CAN Node 4 Receive Data Input</b>

**Table 9-18 Pin Definitions and Functions (cont'd)**

<b>Pin</b>	<b>Symbol</b>	<b>Ctrl.</b>	<b>Type</b>	<b>Function</b>
9	P7.3	O0 / I	St/B	<b>Bit 3 of Port 7, General Purpose Input/Output</b>
	EMUX1	O1	St/B	<b>External Analog MUX Control Output 1 (ADC1)</b>
	U0C1_DOUT	O2	St/B	<b>USIC0 Channel 1 Shift Data Output</b>
	U0C0_DOUT	O3	St/B	<b>USIC0 Channel 0 Shift Data Output</b>
	CCU62_CCP OS1A	I	St/B	<b>CCU62 Position Input 1</b>
	TMS_C	I	St/B	<b>JTAG Test Mode Selection Input</b>
	U0C1_DX0F	I	St/B	<b>USIC0 Channel 1 Shift Data Input</b>
10	P8.2	O0 / I	St/B	<b>Bit 2 of Port 8, General Purpose Input/Output</b>
	CCU60_CC6 2	O1	St/B	<b>CCU60 Channel 2 Output</b>
	TxDC1	O2	St/B	<b>CAN Node 1 Transmit Data Output</b>
	U1C1_DOUT	O3	St/B	<b>USIC1 Channel 1 Shift Data output</b>
	CCU60_CC6 2INB	I	St/B	<b>CCU60 Channel 2 Input</b>
11	P7.1	O0 / I	St/B	<b>Bit 1 of Port 7, General Purpose Input/Output</b>
	EXTCLK	O1	St/B	<b>Programmable Clock Signal Output</b>
	TXDC4	O2	St/B	<b>CAN Node 4 Transmit Data Output</b>
	CCU62_CTR APA	I	St/B	<b>CCU62 Emergency Trap Input</b>
	<u>BRKIN_C</u>	I	St/B	<b>OCDS Break Signal Input</b>
12	P7.4	O0 / I	St/B	<b>Bit 4 of Port 7, General Purpose Input/Output</b>
	EMUX2	O1	St/B	<b>External Analog MUX Control Output 2 (ADC1)</b>
	U0C1_DOUT	O2	St/B	<b>USIC0 Channel 1 Shift Data Output</b>
	U0C1_SCLK OUT	O3	St/B	<b>USIC0 Channel 1 Shift Clock Output</b>
	CCU62_CCP OS2A	I	St/B	<b>CCU62 Position Input 2</b>
	TCK_C	I	St/B	<b>DAP0/JTAG Clock Input</b>
	U0C0_DX0D	I	St/B	<b>USIC0 Channel 0 Shift Data Input</b>
	U0C1_DX1E	I	St/B	<b>USIC0 Channel 1 Shift Clock Input</b>

**Table 9-18 Pin Definitions and Functions (cont'd)**

<b>Pin</b>	<b>Symbol</b>	<b>Ctrl.</b>	<b>Type</b>	<b>Function</b>
13	P8.1	O0 / I	St/B	<b>Bit 1 of Port 8, General Purpose Input/Output</b>
	CCU60_CC6 1	O1	St/B	<b>CCU60 Channel 1 Output</b>
	CCU60_CC6 1INB	I	St/B	<b>CCU60 Channel 1 Input</b>
	RxDC1F	I	St/B	<b>CAN Node 1 Receive Data Input</b>
14	P8.0	O0 / I	St/B	<b>Bit 0 of Port 8, General Purpose Input/Output</b>
	CCU60_CC6 0	O1	St/B	<b>CCU60 Channel 0 Output</b>
	CCU60_CC6 0INB	I	St/B	<b>CCU60 Channel 0 Input</b>
16	P6.0	O0 / I	St/A	<b>Bit 0 of Port 6, General Purpose Input/Output</b>
	EMUX0	O1	St/A	<b>External Analog MUX Control Output 0 (ADC0)</b>
	TxDC2	O2	St/A	<b>CAN Node 2 Transmit Data Output</b>
	BRKOUT	O3	St/A	<b>OCDS Break Signal Output</b>
	ADCx_REQG TyG	I	St/A	<b>External Request Gate Input for ADC0/1</b>
	U1C1_DX0E	I	St/A	<b>USIC1 Channel 1 Shift Data Input</b>
17	P6.1	O0 / I	St/A	<b>Bit 1 of Port 6, General Purpose Input/Output</b>
	EMUX1	O1	St/A	<b>External Analog MUX Control Output 1 (ADC0)</b>
	T3OUT	O2	St/A	<b>GPT12E Timer T3 Toggle Latch Output</b>
	U1C1_DOUT	O3	St/A	<b>USIC1 Channel 1 Shift Data Output</b>
	ADCx_REQT RyE	I	St/A	<b>External Request Trigger Input for ADC0/1</b>
	RxDC2E	I	St/A	<b>CAN Node 2 Receive Data Input</b>
	ESR1_6	I	St/A	<b>ESR1 Trigger Input 6</b>
18	P6.2	O0 / I	St/A	<b>Bit 2 of Port 6, General Purpose Input/Output</b>
	EMUX2	O1	St/A	<b>External Analog MUX Control Output 2 (ADC0)</b>
	T6OUT	O2	St/A	<b>GPT12E Timer T6 Toggle Latch Output</b>
	U1C1_SCLK OUT	O3	St/A	<b>USIC1 Channel 1 Shift Clock Output</b>
	U1C1_DX1C	I	St/A	<b>USIC1 Channel 1 Shift Clock Input</b>

**Table 9-18 Pin Definitions and Functions (cont'd)**

<b>Pin</b>	<b>Symbol</b>	<b>Ctrl.</b>	<b>Type</b>	<b>Function</b>
19	P6.3	O0 / I	St/A	<b>Bit 3 of Port 6, General Purpose Input/Output</b>
	T3OUT	O2	St/A	<b>GPT12E Timer T3 Toggle Latch Output</b>
	U1C1_SELO 0	O3	St/A	<b>USIC1 Channel 1 Select/Control 0 Output</b>
	U1C1_DX2D	I	St/A	<b>USIC1 Channel 1 Shift Control Input</b>
	ADCx_REQT RyF	I	St/A	<b>External Request Trigger Input for ADC0/1</b>
21	P15.0	I	In/A	<b>Bit 0 of Port 15, General Purpose Input</b>
	ADC1_CH0	I	In/A	<b>Analog Input Channel 0 for ADC1</b>
22	P15.1	I	In/A	<b>Bit 1 of Port 15, General Purpose Input</b>
	ADC1_CH1	I	In/A	<b>Analog Input Channel 1 for ADC1</b>
23	P15.2	I	In/A	<b>Bit 2 of Port 15, General Purpose Input</b>
	ADC1_CH2	I	In/A	<b>Analog Input Channel 2 for ADC1</b>
	T5INA	I	In/A	<b>GPT12E Timer T5 Count/Gate Input</b>
24	P15.3	I	In/A	<b>Bit 3 of Port 15, General Purpose Input</b>
	ADC1_CH3	I	In/A	<b>Analog Input Channel 3 for ADC1</b>
	T5EUDA	I	In/A	<b>GPT12E Timer T5 External Up/Down Control Input</b>
25	P15.4	I	In/A	<b>Bit 4 of Port 15, General Purpose Input</b>
	ADC1_CH4	I	In/A	<b>Analog Input Channel 4 for ADC1</b>
	T6INA	I	In/A	<b>GPT12E Timer T6 Count/Gate Input</b>
26	P15.5	I	In/A	<b>Bit 5 of Port 15, General Purpose Input</b>
	ADC1_CH5	I	In/A	<b>Analog Input Channel 5 for ADC1</b>
	T6EUDA	I	In/A	<b>GPT12E Timer T6 External Up/Down Control Input</b>
27	P15.6	I	In/A	<b>Bit 6 of Port 15, General Purpose Input</b>
	ADC1_CH6	I	In/A	<b>Analog Input Channel 6 for ADC1</b>
28	P15.7	I	In/A	<b>Bit 7 of Port 15, General Purpose Input</b>
	ADC1_CH7	I	In/A	<b>Analog Input Channel 7 for ADC1</b>
29	$V_{AREF1}$	-	PS/A	<b>Reference Voltage for A/D Converter ADC1</b>
30	$V_{AREF0}$	-	PS/A	<b>Reference Voltage for A/D Converter ADC0</b>

**Table 9-18 Pin Definitions and Functions (cont'd)**

<b>Pin</b>	<b>Symbol</b>	<b>Ctrl.</b>	<b>Type</b>	<b>Function</b>
31	$V_{AGND}$	-	PS/A	<b>Reference Ground for A/D Converters ADC0/1</b>
32	P5.0	I	In/A	<b>Bit 0 of Port 5, General Purpose Input</b>
	ADC0_CH0	I	In/A	<b>Analog Input Channel 0 for ADC0</b>
33	P5.1	I	In/A	<b>Bit 1 of Port 5, General Purpose Input</b>
	ADC0_CH1	I	In/A	<b>Analog Input Channel 1 for ADC0</b>
34	P5.2	I	In/A	<b>Bit 2 of Port 5, General Purpose Input</b>
	ADC0_CH2	I	In/A	<b>Analog Input Channel 2 for ADC0</b>
	TDI_A	I	In/A	<b>JTAG Test Data Input</b>
35	P5.3	I	In/A	<b>Bit 3 of Port 5, General Purpose Input</b>
	ADC0_CH3	I	In/A	<b>Analog Input Channel 3 for ADC0</b>
	T3INA	I	In/A	<b>GPT12E Timer T3 Count/Gate Input</b>
39	P5.4	I	In/A	<b>Bit 4 of Port 5, General Purpose Input</b>
	ADC0_CH4	I	In/A	<b>Analog Input Channel 4 for ADC0</b>
	CCU63_T12 HRB	I	In/A	<b>External Run Control Input for T12 of CCU63</b>
	T3EUDA	I	In/A	<b>GPT12E Timer T3 External Up/Down Control Input</b>
	TMS_A	I	In/A	<b>JTAG Test Mode Selection Input</b>
40	P5.5	I	In/A	<b>Bit 5 of Port 5, General Purpose Input</b>
	ADC0_CH5	I	In/A	<b>Analog Input Channel 5 for ADC0</b>
	CCU60_T12 HRB	I	In/A	<b>External Run Control Input for T12 of CCU60</b>
41	P5.6	I	In/A	<b>Bit 6 of Port 5, General Purpose Input</b>
	ADC0_CH6	I	In/A	<b>Analog Input Channel 6 for ADC0</b>
42	P5.7	I	In/A	<b>Bit 7 of Port 5, General Purpose Input</b>
	ADC0_CH7	I	In/A	<b>Analog Input Channel 7 for ADC0</b>

**Table 9-18 Pin Definitions and Functions (cont'd)**

<b>Pin</b>	<b>Symbol</b>	<b>Ctrl.</b>	<b>Type</b>	<b>Function</b>
43	P5.8	I	In/A	<b>Bit 8 of Port 5, General Purpose Input</b>
	ADC0_CH8	I	In/A	<b>Analog Input Channel 8 for ADC0</b>
	ADC1_CH8	I	In/A	<b>Analog Input Channel 8 for ADC1</b>
	CCU6x_T12H RC	I	In/A	<b>External Run Control Input for T12 of CCU60/1/2/3</b>
	CCU6x_T13H RC	I	In/A	<b>External Run Control Input for T13 of CCU60/1/2/3</b>
	U2C0_DX0F	I	In/A	<b>USIC2 Channel 0 Shift Data Input</b>
44	P5.9	I	In/A	<b>Bit 9 of Port 5, General Purpose Input</b>
	ADC0_CH9	I	In/A	<b>Analog Input Channel 9 for ADC0</b>
	ADC1_CH9	I	In/A	<b>Analog Input Channel 9 for ADC1</b>
	CC2_T7IN	I	In/A	<b>CAPCOM2 Timer T7 Count Input</b>
45	P5.10	I	In/A	<b>Bit 10 of Port 5, General Purpose Input</b>
	ADC0_CH10	I	In/A	<b>Analog Input Channel 10 for ADC0</b>
	ADC1_CH10	I	In/A	<b>Analog Input Channel 10 for ADC1</b>
	BRKIN_A	I	In/A	<b>OCDS Break Signal Input</b>
	U2C1_DX0F	I	In/A	<b>USIC2 Channel 1 Shift Data Input</b>
	CCU61_T13 HRA	I	In/A	<b>External Run Control Input for T13 of CCU61</b>
46	P5.11	I	In/A	<b>Bit 11 of Port 5, General Purpose Input</b>
	ADC0_CH11	I	In/A	<b>Analog Input Channel 11 for ADC0</b>
	ADC1_CH11	I	In/A	<b>Analog Input Channel 11 for ADC1</b>
47	P5.12	I	In/A	<b>Bit 12 of Port 5, General Purpose Input</b>
	ADC0_CH12	I	In/A	<b>Analog Input Channel 12 for ADC0</b>
48	P5.13	I	In/A	<b>Bit 13 of Port 5, General Purpose Input</b>
	ADC0_CH13	I	In/A	<b>Analog Input Channel 13 for ADC0</b>
	CCU63_T13 HRF	I	In/A	<b>External Run Control Input for T13 of CCU63</b>
49	P5.14	I	In/A	<b>Bit 14 of Port 5, General Purpose Input</b>
	ADC0_CH14	I	In/A	<b>Analog Input Channel 14 for ADC0</b>

**Table 9-18 Pin Definitions and Functions (cont'd)**

<b>Pin</b>	<b>Symbol</b>	<b>Ctrl.</b>	<b>Type</b>	<b>Function</b>
50	P5.15	I	In/A	<b>Bit 15 of Port 5, General Purpose Input</b>
	ADC0_CH15	I	In/A	<b>Analog Input Channel 15 for ADC0</b>
	RxDC2F	I	In/A	<b>CAN Node 2 Receive Data Input</b>
51	P2.12	O0 / I	St/B	<b>Bit 12 of Port 2, General Purpose Input/Output</b>
	U0C0_SELO 4	O1	St/B	<b>USIC0 Channel 0 Select/Control 4 Output</b>
	U0C1_SELO 3	O2	St/B	<b>USIC0 Channel 1 Select/Control 3 Output</b>
	TXDC2	O3	St/B	<b>CAN Node 2 Transmit Data Output</b>
	READY	I	St/B	<b>External Bus Interface READY Input</b>
52	P2.11	O0 / I	St/B	<b>Bit 11 of Port 2, General Purpose Input/Output</b>
	U0C0_SELO 2	O1	St/B	<b>USIC0 Channel 0 Select/Control 2 Output</b>
	U0C1_SELO 2	O2	St/B	<b>USIC0 Channel 1 Select/Control 2 Output</b>
	U3C1_DOUT	O3	St/B	<b>USIC3 Channel 1 Shift Data Output</b>
	$\overline{\text{BHE}}/\text{WRH}$	OH	St/B	<b>External Bus Interf. High-Byte Control Output</b> Can operate either as Byte High Enable ( $\overline{\text{BHE}}$ ) or as Write strobe for High Byte (WRH).
53	P11.5	O0 / I	St/B	<b>Bit 5 of Port 11, General Purpose Input/Output</b>
	CCU61_CC6 0	O1	St/B	<b>CCU61 Channel 0 Output</b>
	CCU61_COU T63	O2	St/B	<b>CCU61 Channel 3 Output</b>
	U3C1_SELO 1	O3	St/B	<b>USIC3 Channel 1 Select/Control 1 Output</b>
	CCU61_CC6 0INB	I	St/B	<b>CCU61 Channel 0 Input</b>
	U3C1_DX2B	I	St/B	<b>USIC3 Channel 1 Shift Control Input</b>



**Table 9-18 Pin Definitions and Functions (cont'd)**

<b>Pin</b>	<b>Symbol</b>	<b>Ctrl.</b>	<b>Type</b>	<b>Function</b>
55	P2.0	O0 / I	St/B	<b>Bit 0 of Port 2, General Purpose Input/Output</b>
	TxDC5	O1	St/B	<b>CAN Node 5 Transmit Data Output</b>
	CCU63_CC6 0	O2	St/B	<b>CCU63 Channel 0 Output</b>
	AD13	OH / I	St/B	<b>External Bus Interface Address/Data Line 13</b>
	RxDC0C	I	St/B	<b>CAN Node 0 Receive Data Input</b>
	CCU63_CC6 0INB	I	St/B	<b>CCU63 Channel 0 Input</b>
	T5INB	I	St/B	<b>GPT12E Timer T5 Count/Gate Input</b>
56	P2.1	O0 / I	St/B	<b>Bit 1 of Port 2, General Purpose Input/Output</b>
	TxDC0	O1	St/B	<b>CAN Node 0 Transmit Data Output</b>
	CCU63_CC6 1	O2	St/B	<b>CCU63 Channel 1 Output</b>
	AD14	OH / I	St/B	<b>External Bus Interface Address/Data Line 14</b>
	RxDC5C	I	St/B	<b>CAN Node 5 Receive Data Input</b>
	CCU63_CC6 1INB	I	St/B	<b>CCU63 Channel 1 Input</b>
	T5EUIDB	I	St/B	<b>GPT12E Timer T5 External Up/Down Control Input</b>
57	ESR1_5	I	St/B	<b>ESR1 Trigger Input 5</b>
	P11.4	O0 / I	St/B	<b>Bit 4 of Port 11, General Purpose Input/Output</b>
	CCU61_CC6 2	O1	St/B	<b>CCU61 Channel 2 Output</b>
	U3C1_DOUT	O2	St/B	<b>USIC3 Channel 1 Shift Data Output</b>
	RxDC5B	I	St/B	<b>CAN Node 5 Receive Data Input</b>
	CCU61_CC6 2INB	I	St/B	<b>CCU61 Channel 2 Input</b>
	U3C1_DX0B	I	St/B	<b>USIC3 Channel 1 Shift Data Input</b>

**Table 9-18 Pin Definitions and Functions (cont'd)**

<b>Pin</b>	<b>Symbol</b>	<b>Ctrl.</b>	<b>Type</b>	<b>Function</b>
58	P2.2	O0 / I	St/B	<b>Bit 2 of Port 2, General Purpose Input/Output</b>
	TxDC1	O1	St/B	<b>CAN Node 1 Transmit Data Output</b>
	CCU63_CC6 2	O2	St/B	<b>CCU63 Channel 2 Output</b>
	AD15	OH / I	St/B	<b>External Bus Interface Address/Data Line 15</b>
	CCU63_CC6 2INB	I	St/B	<b>CCU63 Channel 2 Input</b>
	ESR2_5	I	St/B	<b>ESR2 Trigger Input 5</b>
59	P11.3	O0 / I	St/B	<b>Bit 3 of Port 11, General Purpose Input/Output</b>
	CCU61_COU T63	O1	St/B	<b>CCU61 Channel 3 Output</b>
	CCU61_COU T62	O2	St/B	<b>CCU61 Channel 2 Output</b>
	TxDC5	O3	St/B	<b>CAN Node 5 Transmit Data Input</b>
	CCU61_T13 HRF	I	St/B	<b>External Run Control Input for T13 of CCU61</b>
60	P4.0	O0 / I	St/B	<b>Bit 0 of Port 4, General Purpose Input/Output</b>
	CC2_CC24	O3 / I	St/B	<b>CAPCOM2 CC24IO Capture Inp./ Compare Out.</b>
	CS0	OH	St/B	<b>External Bus Interface Chip Select 0 Output</b>
61	P2.3	O0 / I	St/B	<b>Bit 3 of Port 2, General Purpose Input/Output</b>
	U0C0_DOUT	O1	St/B	<b>USIC0 Channel 0 Shift Data Output</b>
	CCU63_COU T63	O2	St/B	<b>CCU63 Channel 3 Output</b>
	CC2_CC16	O3 / I	St/B	<b>CAPCOM2 CC16IO Capture Inp./ Compare Out.</b>
	A16	OH	St/B	<b>External Bus Interface Address Line 16</b>
	ESR2_0	I	St/B	<b>ESR2 Trigger Input 0</b>
	U0C0_DX0E	I	St/B	<b>USIC0 Channel 0 Shift Data Input</b>
	U0C1_DX0D	I	St/B	<b>USIC0 Channel 1 Shift Data Input</b>
	RxDC0A	I	St/B	<b>CAN Node 0 Receive Data Input</b>

**Table 9-18 Pin Definitions and Functions (cont'd)**

<b>Pin</b>	<b>Symbol</b>	<b>Ctrl.</b>	<b>Type</b>	<b>Function</b>
62	P11.2	O0 / I	St/B	<b>Bit 2 of Port 11, General Purpose Input/Output</b>
	CCU61_CC61	O1	St/B	<b>CCU61 Channel 1 Output</b>
	U3C1_DOUT	O2	St/B	<b>USIC3 Channel 1 Shift Data Output</b>
	CCU63_CCP OS2A	I	St/B	<b>CCU63 Position Input 2</b>
	CCU61_CC61INB	I	St/B	<b>CCU61 Channel 1 Input</b>
63	P4.1	O0 / I	St/B	<b>Bit 1 of Port 4, General Purpose Input/Output</b>
	U3C0_SELO3	O1	St/B	<b>USIC3 Channel Select/Control 3 Output</b>
	TxDC2	O2	St/B	<b>CAN Node 2 Transmit Data Output</b>
	CC2_CC25	O3 / I	St/B	<b>CAPCOM2 CC25IO Capture Inp./ Compare Out.</b>
	CS1	OH	St/B	<b>External Bus Interface Chip Select 1 Output</b>
	CCU62_CCP OS0B	I	St/B	<b>CCU62 Position Input 0</b>
	T4EADB	I	St/B	<b>GPT12E Timer T4 External Up/Down Control Input</b>
	ESR1_8	I	St/B	<b>ESR1 Trigger Input 8</b>
64	P2.4	O0 / I	St/B	<b>Bit 4 of Port 2, General Purpose Input/Output</b>
	U0C1_DOUT	O1	St/B	<b>USIC0 Channel 1 Shift Data Output</b>
	TxDC0	O2	St/B	<b>CAN Node 0 Transmit Data Output</b>
	CC2_CC17	O3 / I	St/B	<b>CAPCOM2 CC17IO Capture Inp./ Compare Out.</b>
	A17	OH	St/B	<b>External Bus Interface Address Line 17</b>
	ESR1_0	I	St/B	<b>ESR1 Trigger Input 0</b>
	U0C0_DX0F	I	St/B	<b>USIC0 Channel 0 Shift Data Input</b>
	RxDC1A	I	St/B	<b>CAN Node 1 Receive Data Input</b>

**Table 9-18 Pin Definitions and Functions (cont'd)**

<b>Pin</b>	<b>Symbol</b>	<b>Ctrl.</b>	<b>Type</b>	<b>Function</b>
65	P11.1	O0 / I	St/B	<b>Bit 1 of Port 11, General Purpose Input/Output</b>
	CCU61_COUT61	O1	St/B	<b>CCU61 Channel 1 Output</b>
	TxDC0	O2	St/B	<b>CAN Node 0 Transmit Data Output</b>
	U3C1_SELO0	O3	St/B	<b>USIC3 Channel 1 Select/Control 0 Output</b>
	CCU63_CCP0S1A	I	St/B	<b>CCU63 Position Input 1</b>
	CCU61_CTRAPD	I	St/B	<b>CCU61 Emergency Trap Input</b>
	U3C1_DX2A	I	St/B	<b>USIC3 Channel 1 Shift Control Input</b>
66	P11.0	O0 / I	St/B	<b>Bit 0 of Port 11, General Purpose Input/Output</b>
	CCU61_COUT60	O1	St/B	<b>CCU61 Channel 0 Output</b>
	U3C1_SCLKOUT	O2	St/B	<b>USIC3 Channel 1 Shift Clock Output</b>
	CCU63_CCP0S0A	I	St/B	<b>CCU63 Position Input 0</b>
	RxDC0F	I	St/B	<b>CAN Node 0 Receive Data Input</b>
	U3C1_DX1A	I	St/B	<b>USIC3 Channel 1 Shift Clock Input</b>
	ESR1_7	I	St/B	<b>ESR1 Trigger Input 7</b>
67	P2.5	O0 / I	St/B	<b>Bit 5 of Port 2, General Purpose Input/Output</b>
	U0C0_SCLKOUT	O1	St/B	<b>USIC0 Channel 0 Shift Clock Output</b>
	TxDC0	O2	St/B	<b>CAN Node 0 Transmit Data Output</b>
	CC2_CC18	O3 / I	St/B	<b>CAPCOM2 CC18IO Capture Inp./ Compare Out.</b>
	A18	OH	St/B	<b>External Bus Interface Address Line 18</b>
	U0C0_DX1D	I	St/B	<b>USIC0 Channel 0 Shift Clock Input</b>
	ESR1_10	I	St/B	<b>ESR1 Trigger Input 10</b>

**Table 9-18 Pin Definitions and Functions (cont'd)**

<b>Pin</b>	<b>Symbol</b>	<b>Ctrl.</b>	<b>Type</b>	<b>Function</b>
68	P4.2	O0 / I	St/B	<b>Bit 2 of Port 4, General Purpose Input/Output</b>
	U3C0_SCLK OUT	O1	St/B	<b>USIC3 Channel 0 Shift Clock Output</b>
	TxDC2	O2	St/B	<b>CAN Node 2 Transmit Data Output</b>
	CC2_CC26	O3 / I	St/B	<b>CAPCOM2 CC26IO Capture Inp./ Compare Out.</b>
	CS2	OH	St/B	<b>External Bus Interface Chip Select 2 Output</b>
	T2INA	I	St/B	<b>GPT12E Timer T2 Count/Gate Input</b>
	CCU62_CCP OS1B	I	St/B	<b>CCU62 Position Input 1</b>
	U3C0_DX1B	I	St/B	<b>USIC3 Channel 0 Shift Clock Input</b>
69	P2.6	O0 / I	St/B	<b>Bit 6 of Port 2, General Purpose Input/Output</b>
	U0C0_SELO 0	O1	St/B	<b>USIC0 Channel 0 Select/Control 0 Output</b>
	U0C1_SELO 1	O2	St/B	<b>USIC0 Channel 1 Select/Control 1 Output</b>
	CC2_CC19	O3 / I	St/B	<b>CAPCOM2 CC19IO Capture Inp./ Compare Out.</b>
	A19	OH	St/B	<b>External Bus Interface Address Line 19</b>
	U0C0_DX2D	I	St/B	<b>USIC0 Channel 0 Shift Control Input</b>
	RxDC0D	I	St/B	<b>CAN Node 0 Receive Data Input</b>
	ESR2_6	I	St/B	<b>ESR2 Trigger Input 6</b>
70	P4.4	O0 / I	St/B	<b>Bit 4 of Port 4, General Purpose Input/Output</b>
	U3C0_SELO 2	O1	St/B	<b>USIC3 Channel 0 Select/Control 2 Output</b>
	CC2_CC28	O3 / I	St/B	<b>CAPCOM2 CC28IO Capture Inp./ Compare Out.</b>
	CS4	OH	St/B	<b>External Bus Interface Chip Select 4 Output</b>
	CLKIN2	I	St/B	<b>Clock Signal Input 2</b>
	U3C0_DX2C	I	St/B	<b>USIC3 Channel 0 Shift Control Input</b>

**Table 9-18 Pin Definitions and Functions (cont'd)**

<b>Pin</b>	<b>Symbol</b>	<b>Ctrl.</b>	<b>Type</b>	<b>Function</b>
71	P4.3	O0 / I	St/B	<b>Bit 3 of Port 4, General Purpose Input/Output</b>
	U0C1_DOUT	O1	St/B	<b>USIC0 Channel 1 Shift Data Output</b>
	CC2_CC27	O3 / I	St/B	<b>CAPCOM2 CC27IO Capture Inp./ Compare Out.</b>
	$\overline{CS3}$	OH	St/B	<b>External Bus Interface Chip Select 3 Output</b>
	RxDC2A	I	St/B	<b>CAN Node 2 Receive Data Input</b>
	T2EUDA	I	St/B	<b>GPT12E Timer T2 External Up/Down Control Input</b>
	CCU62_CCP OS2B	I	St/B	<b>CCU62 Position Input 2</b>
75	P0.0	O0 / I	St/B	<b>Bit 0 of Port 0, General Purpose Input/Output</b>
	U1C0_DOUT	O1	St/B	<b>USIC1 Channel 0 Shift Data Output</b>
	CCU61_CC6 0	O3	St/B	<b>CCU61 Channel 0 IOutput</b>
	A0	OH	St/B	<b>External Bus Interface Address Line 0</b>
	U1C0_DX0A	I	St/B	<b>USIC1 Channel 0 Shift Data Input</b>
	CCU61_CC6 0INA	I	St/B	<b>CCU61 Channel 0 Input</b>
	ESR1_11	I	St/B	<b>ESR1 Trigger Input 11</b>
76	P4.5	O0 / I	St/B	<b>Bit 5 of Port 4, General Purpose Input/Output</b>
	U3C0_DOUT	O1	St/B	<b>USIC3 Channel 0 Shift Data Output</b>
	CC2_CC29	O3 / I	St/B	<b>CAPCOM2 CC29IO Capture Inp./Compare Out.</b>
	CCU61_CCP OS0A	I	St/B	<b>CCU61 Position Input 0</b>
	U3C0_DX0B	I	St/B	<b>USIC3 Channel 0 Shift Data Input</b>
	ESR2_10	I	St/B	<b>ESR2 Trigger Input 10</b>
77	P4.6	O0 / I	St/B	<b>Bit 6 of Port 4, General Purpose Input/Output</b>
	U3C0_DOUT	O1	St/B	<b>USIC3 Channel 0 Shift Data Output</b>
	CC2_CC30	O3 / I	St/B	<b>CAPCOM2 CC30IO Capture Inp./ Compare Out.</b>
	T4INA	I	St/B	<b>GPT12E Timer T4 Count/Gate Input</b>
	CCU61_CCP OS1A	I	St/B	<b>CCU61 Position Input 1</b>

**Table 9-18 Pin Definitions and Functions (cont'd)**

<b>Pin</b>	<b>Symbol</b>	<b>Ctrl.</b>	<b>Type</b>	<b>Function</b>
78	P2.7	O0 / I	St/B	<b>Bit 7 of Port 2, General Purpose Input/Output</b>
	U0C1_SELO0	O1	St/B	<b>USIC0 Channel 1 Select/Control 0 Output</b>
	U0C0_SELO1	O2	St/B	<b>USIC0 Channel 0 Select/Control 1 Output</b>
	CC2_CC20	O3 / I	St/B	<b>CAPCOM2 CC20IO Capture Inp./ Compare Out.</b>
	A20	OH	St/B	<b>External Bus Interface Address Line 20</b>
	U0C1_DX2C	I	St/B	<b>USIC0 Channel 1 Shift Control Input</b>
	RxDC1C	I	St/B	<b>CAN Node 1 Receive Data Input</b>
	ESR2_7	I	St/B	<b>ESR2 Trigger Input 7</b>
79	P0.1	O0 / I	St/B	<b>Bit 1 of Port 0, General Purpose Input/Output</b>
	U1C0_DOUT	O1	St/B	<b>USIC1 Channel 0 Shift Data Output</b>
	TxDC0	O2	St/B	<b>CAN Node 0 Transmit Data Output</b>
	CCU61_CC61	O3	St/B	<b>CCU61 Channel 1 Output</b>
	A1	OH	St/B	<b>External Bus Interface Address Line 1</b>
	U1C0_DX0B	I	St/B	<b>USIC1 Channel 0 Shift Data Input</b>
	CCU61_CC61INA	I	St/B	<b>CCU61 Channel 1 Input</b>
	U1C0_DX1A	I	St/B	<b>USIC1 Channel 0 Shift Clock Input</b>
80	P2.8	O0 / I	DP/B	<b>Bit 8 of Port 2, General Purpose Input/Output</b>
	U0C1_SCLKOUT	O1	DP/B	<b>USIC0 Channel 1 Shift Clock Output</b>
	EXTCLK	O2	DP/B	<b>Programmable Clock Signal Output<sup>1)</sup></b>
	CC2_CC21	O3 / I	DP/B	<b>CAPCOM2 CC21IO Capture Inp./ Compare Out.</b>
	A21	OH	DP/B	<b>External Bus Interface Address Line 21</b>
	U0C1_DX1D	I	DP/B	<b>USIC0 Channel 1 Shift Clock Input</b>

**Table 9-18 Pin Definitions and Functions (cont'd)**

<b>Pin</b>	<b>Symbol</b>	<b>Ctrl.</b>	<b>Type</b>	<b>Function</b>
81	P4.7	O0 / I	St/B	<b>Bit 7 of Port 4, General Purpose Input/Output</b>
	CC2_CC31	O3 / I	St/B	<b>CAPCOM2 CC31IO Capture Inp./ Compare Out.</b>
	T4EUDA	I	St/B	<b>GPT12E Timer T4 External Up/Down Control Input</b>
	CCU61_CCP OS2A	I	St/B	<b>CCU61 Position Input 2</b>
82	P2.9	O0 / I	St/B	<b>Bit 9 of Port 2, General Purpose Input/Output</b>
	U0C1_DOUT	O1	St/B	<b>USIC0 Channel 1 Shift Data Output</b>
	TxDC1	O2	St/B	<b>CAN Node 1 Transmit Data Output</b>
	CC2_CC22	O3 / I	St/B	<b>CAPCOM2 CC22IO Capture Inp./ Compare Out.</b>
	A22	OH	St/B	<b>External Bus Interface Address Line 22</b>
	CLKIN1	I	St/B	<b>Clock Signal Input 1</b>
	TCK_A	I	St/B	<b>DAP0/JTAG Clock Input</b>
83	P0.2	O0 / I	St/B	<b>Bit 2 of Port 0, General Purpose Input/Output</b>
	U1C0_SCLK OUT	O1	St/B	<b>USIC1 Channel 0 Shift Clock Output</b>
	TxDC0	O2	St/B	<b>CAN Node 0 Transmit Data Output</b>
	CCU61_CC6 2	O3	St/B	<b>CCU61 Channel 2 Output</b>
	A2	OH	St/B	<b>External Bus Interface Address Line 2</b>
	U1C0_DX1B	I	St/B	<b>USIC1 Channel 0 Shift Clock Input</b>
	CCU61_CC6 2INA	I	St/B	<b>CCU61 Channel 2 Input</b>



**Table 9-18 Pin Definitions and Functions (cont'd)**

<b>Pin</b>	<b>Symbol</b>	<b>Ctrl.</b>	<b>Type</b>	<b>Function</b>
84	P10.0	O0 / I	St/B	<b>Bit 0 of Port 10, General Purpose Input/Output</b>
	U0C1_DOUT	O1	St/B	<b>USIC0 Channel 1 Shift Data Output</b>
	CCU60_CC60	O2	St/B	<b>CCU60 Channel 0 Output</b>
	AD0	OH / I	St/B	<b>External Bus Interface Address/Data Line 0</b>
	CCU60_CC60INA	I	St/B	<b>CCU60 Channel 0 Input</b>
	ESR1_2	I	St/B	<b>ESR1 Trigger Input 2</b>
	U0C0_DX0A	I	St/B	<b>USIC0 Channel 0 Shift Data Input</b>
	U0C1_DX0A	I	St/B	<b>USIC0 Channel 1 Shift Data Input</b>
85	P3.0	O0 / I	St/B	<b>Bit 0 of Port 3, General Purpose Input/Output</b>
	U2C0_DOUT	O1	St/B	<b>USIC2 Channel 0 Shift Data Output</b>
	$\overline{\text{BREQ}}$	OH	St/B	<b>External Bus Request Output</b>
	ESR1_1	I	St/B	<b>ESR1 Trigger Input 1</b>
	U2C0_DX0A	I	St/B	<b>USIC2 Channel 0 Shift Data Input</b>
	RxDC3B	I	St/B	<b>CAN Node 3 Receive Data Input</b>
	U2C0_DX1A	I	St/B	<b>USIC2 Channel 0 Shift Clock Input</b>
86	P10.1	O0 / I	St/B	<b>Bit 1 of Port 10, General Purpose Input/Output</b>
	U0C0_DOUT	O1	St/B	<b>USIC0 Channel 0 Shift Data Output</b>
	CCU60_CC61	O2	St/B	<b>CCU60 Channel 1 Output</b>
	AD1	OH / I	St/B	<b>External Bus Interface Address/Data Line 1</b>
	CCU60_CC61INA	I	St/B	<b>CCU60 Channel 1 Input</b>
	U0C0_DX1A	I	St/B	<b>USIC0 Channel 0 Shift Clock Input</b>
	U0C0_DX0B	I	St/B	<b>USIC0 Channel 0 Shift Data Input</b>

**Table 9-18 Pin Definitions and Functions (cont'd)**

<b>Pin</b>	<b>Symbol</b>	<b>Ctrl.</b>	<b>Type</b>	<b>Function</b>
87	P0.3	O0 / I	St/B	<b>Bit 3 of Port 0, General Purpose Input/Output</b>
	U1C0_SELO 0	O1	St/B	<b>USIC1 Channel 0 Select/Control 0 Output</b>
	U1C1_SELO 1	O2	St/B	<b>USIC1 Channel 1 Select/Control 1 Output</b>
	CCU61_COU T60	O3	St/B	<b>CCU61 Channel 0 Output</b>
	A3	OH	St/B	<b>External Bus Interface Address Line 3</b>
	U1C0_DX2A	I	St/B	<b>USIC1 Channel 0 Shift Control Input</b>
	RxDC0B	I	St/B	<b>CAN Node 0 Receive Data Input</b>
88	P3.1	O0 / I	St/B	<b>Bit 1 of Port 3, General Purpose Input/Output</b>
	U2C0_DOUT	O1	St/B	<b>USIC2 Channel 0 Shift Data Output</b>
	TxDC3	O2	St/B	<b>CAN Node 3 Transmit Data Output</b>
	HLDA	OH / I	St/B	<b>External Bus Hold Acknowledge Output/Input</b> Output in master mode, input in slave mode.
	U2C0_DX0B	I	St/B	<b>USIC2 Channel 0 Shift Data Input</b>
89	P10.2	O0 / I	St/B	<b>Bit 2 of Port 10, General Purpose Input/Output</b>
	U0C0_SCLK OUT	O1	St/B	<b>USIC0 Channel 0 Shift Clock Output</b>
	CCU60_CC6 2	O2	St/B	<b>CCU60 Channel 2 Output</b>
	U3C0_SELO 1	O3	St/B	<b>USIC3 Channel 0 Select/Control 1 Output</b>
	AD2	OH / I	St/B	<b>External Bus Interface Address/Data Line 2</b>
	CCU60_CC6 2INA	I	St/B	<b>CCU60 Channel 2 Input</b>
	U0C0_DX1B	I	St/B	<b>USIC0 Channel 0 Shift Clock Input</b>
	U3C0_DX2B	I	St/B	<b>USIC3 Channel 0 Shift Control Input</b>

**Table 9-18 Pin Definitions and Functions (cont'd)**

<b>Pin</b>	<b>Symbol</b>	<b>Ctrl.</b>	<b>Type</b>	<b>Function</b>
90	P0.4	O0 / I	St/B	<b>Bit 4 of Port 0, General Purpose Input/Output</b>
	U1C1_SELO 0	O1	St/B	<b>USIC1 Channel 1 Select/Control 0 Output</b>
	U1C0_SELO 1	O2	St/B	<b>USIC1 Channel 0 Select/Control 1 Output</b>
	CCU61_COU T61	O3	St/B	<b>CCU61 Channel 1 Output</b>
	A4	OH	St/B	<b>External Bus Interface Address Line 4</b>
	U1C1_DX2A	I	St/B	<b>USIC1 Channel 1 Shift Control Input</b>
	RxDC1B	I	St/B	<b>CAN Node 1 Receive Data Input</b>
	ESR2_8	I	St/B	<b>ESR2 Trigger Input 8</b>
92	P2.13	O0 / I	St/B	<b>Bit 13 of Port 2, General Purpose Input/Output</b>
	U2C1_SELO 2	O1	St/B	<b>USIC2 Channel 1 Select/Control 2 Output</b>
	RxDC2D	I	St/B	<b>CAN Node 2 Receive Data Input</b>
93	P3.2	O0 / I	St/B	<b>Bit 2 of Port 3, General Purpose Input/Output</b>
	U2C0_SCLK OUT	O1	St/B	<b>USIC2 Channel 0 Shift Clock Output</b>
	TxDC3	O2	St/B	<b>CAN Node 3 Transmit Data Output</b>
	U2C0_DX1B	I	St/B	<b>USIC2 Channel 0 Shift Clock Input</b>
	HOLD	I	St/B	<b>External Bus Master Hold Request Input</b>
94	P2.10	O0 / I	St/B	<b>Bit 10 of Port 2, General Purpose Input/Output</b>
	U0C1_DOUT	O1	St/B	<b>USIC0 Channel 1 Shift Data Output</b>
	U0C0_SELO 3	O2	St/B	<b>USIC0 Channel 0 Select/Control 3 Output</b>
	CC2_CC23	O3 / I	St/B	<b>CAPCOM2 CC23IO Capture Inp./ Compare Out.</b>
	A23	OH	St/B	<b>External Bus Interface Address Line 23</b>
	U0C1_DX0E	I	St/B	<b>USIC0 Channel 1 Shift Data Input</b>
	CAPINA	I	St/B	<b>GPT12E Register CAPREL Capture Input</b>
	U3C1_DX0A	I	St/B	<b>USIC3 Channel 1 Shift Data Input</b>

**Table 9-18 Pin Definitions and Functions (cont'd)**

<b>Pin</b>	<b>Symbol</b>	<b>Ctrl.</b>	<b>Type</b>	<b>Function</b>
95	P10.3	O0 / I	St/B	<b>Bit 3 of Port 10, General Purpose Input/Output</b>
	CCU60_COUT60	O2	St/B	<b>CCU60 Channel 0 Output</b>
	AD3	OH / I	St/B	<b>External Bus Interface Address/Data Line 3</b>
	U0C0_DX2A	I	St/B	<b>USIC0 Channel 0 Shift Control Input</b>
	U0C1_DX2A	I	St/B	<b>USIC0 Channel 1 Shift Control Input</b>
	U3C0_DX0A	I	St/B	<b>USIC3 Channel 0 Shift Data Input</b>
96	P0.5	O0 / I	St/B	<b>Bit 5 of Port 0, General Purpose Input/Output</b>
	U1C1_SCLKOUT	O1	St/B	<b>USIC1 Channel 1 Shift Clock Output</b>
	U1C0_SELO2	O2	St/B	<b>USIC1 Channel 0 Select/Control 2 Output</b>
	CCU61_COUT62	O3	St/B	<b>CCU61 Channel 2 Output</b>
	A5	OH	St/B	<b>External Bus Interface Address Line 5</b>
	U1C1_DX1A	I	St/B	<b>USIC1 Channel 1 Shift Clock Input</b>
	U1C0_DX1C	I	St/B	<b>USIC1 Channel 0 Shift Clock Input</b>
	RXDC3E	I	St/B	<b>CAN Node 3 Receive Data Input</b>
97	P3.3	O0 / I	St/B	<b>Bit 3 of Port 3, General Purpose Input/Output</b>
	U2C0_SELO0	O1	St/B	<b>USIC2 Channel 0 Select/Control 0 Output</b>
	U2C1_SELO1	O2	St/B	<b>USIC2 Channel 1 Select/Control 1 Output</b>
	U2C0_DX2A	I	St/B	<b>USIC2 Channel 0 Shift Control Input</b>
	RxDC3A	I	St/B	<b>CAN Node 3 Receive Data Input</b>

**Table 9-18 Pin Definitions and Functions (cont'd)**

<b>Pin</b>	<b>Symbol</b>	<b>Ctrl.</b>	<b>Type</b>	<b>Function</b>
98	P10.4	O0 / I	St/B	<b>Bit 4 of Port 10, General Purpose Input/Output</b>
	U0C0_SELO3	O1	St/B	<b>USIC0 Channel 0 Select/Control 3 Output</b>
	CCU60_COUT61	O2	St/B	<b>CCU60 Channel 1 Output</b>
	U3C0_DOUT	O3	St/B	<b>USIC3 Channel 0 Shift Data Output</b>
	AD4	OH / I	St/B	<b>External Bus Interface Address/Data Line 4</b>
	U0C0_DX2B	I	St/B	<b>USIC0 Channel 0 Shift Control Input</b>
	U0C1_DX2B	I	St/B	<b>USIC0 Channel 1 Shift Control Input</b>
	ESR1_9	I	St/B	<b>ESR1 Trigger Input 9</b>
99	P3.4	O0 / I	St/B	<b>Bit 4 of Port 3, General Purpose Input/Output</b>
	U2C1_SELO0	O1	St/B	<b>USIC2 Channel 1 Select/Control 0 Output</b>
	U2C0_SELO1	O2	St/B	<b>USIC2 Channel 0 Select/Control 1 Output</b>
	U0C0_SELO4	O3	St/B	<b>USIC0 Channel 0 Select/Control 4 Output</b>
	U2C1_DX2A	I	St/B	<b>USIC2 Channel 1 Shift Control Input</b>
	RxDC4A	I	St/B	<b>CAN Node 4 Receive Data Input</b>
100	P10.5	O0 / I	St/B	<b>Bit 5 of Port 10, General Purpose Input/Output</b>
	U0C1_SCLKOUT	O1	St/B	<b>USIC0 Channel 1 Shift Clock Output</b>
	CCU60_COUT62	O2	St/B	<b>CCU60 Channel 2 Output</b>
	U2C0_DOUT	O3	St/B	<b>USIC2 Channel 0 Shift Data Output</b>
	AD5	OH / I	St/B	<b>External Bus Interface Address/Data Line 5</b>
	U0C1_DX1B	I	St/B	<b>USIC0 Channel 1 Shift Clock Input</b>

**Table 9-18 Pin Definitions and Functions (cont'd)**

<b>Pin</b>	<b>Symbol</b>	<b>Ctrl.</b>	<b>Type</b>	<b>Function</b>
101	P3.5	O0 / I	St/B	<b>Bit 5 of Port 3, General Purpose Input/Output</b>
	U2C1_SCLK OUT	O1	St/B	<b>USIC2 Channel 1 Shift Clock Output</b>
	U2C0_SELO 2	O2	St/B	<b>USIC2 Channel 0 Select/Control 2 Output</b>
	U0C0_SELO 5	O3	St/B	<b>USIC0 Channel 0 Select/Control 5 Output</b>
	U2C1_DX1A	I	St/B	<b>USIC2 Channel 1 Shift Clock Input</b>
102	P0.6	O0 / I	St/B	<b>Bit 6 of Port 0, General Purpose Input/Output</b>
	U1C1_DOUT	O1	St/B	<b>USIC1 Channel 1 Shift Data Output</b>
	TxDC1	O2	St/B	<b>CAN Node 1 Transmit Data Output</b>
	CCU61_COU T63	O3	St/B	<b>CCU61 Channel 3 Output</b>
	A6	OH	St/B	<b>External Bus Interface Address Line 6</b>
	U1C1_DX0A	I	St/B	<b>USIC1 Channel 1 Shift Data Input</b>
	CCU61_CTR APA	I	St/B	<b>CCU61 Emergency Trap Input</b>
	U1C1_DX1B	I	St/B	<b>USIC1 Channel 1 Shift Clock Input</b>
103	P10.6	O0 / I	St/B	<b>Bit 6 of Port 10, General Purpose Input/Output</b>
	U0C0_DOUT	O1	St/B	<b>USIC0 Channel 0 Shift Data Output</b>
	TxDC4	O2	St/B	<b>CAN Node 4 Transmit Data Output</b>
	U1C0_SELO 0	O3	St/B	<b>USIC1 Channel 0 Select/Control 0 Output</b>
	AD6	OH / I	St/B	<b>External Bus Interface Address/Data Line 6</b>
	U0C0_DX0C	I	St/B	<b>USIC0 Channel 0 Shift Data Input</b>
	U1C0_DX2D	I	St/B	<b>USIC1 Channel 0 Shift Control Input</b>
	CCU60_CTR APA	I	St/B	<b>CCU60 Emergency Trap Input</b>

**Table 9-18 Pin Definitions and Functions (cont'd)**

<b>Pin</b>	<b>Symbol</b>	<b>Ctrl.</b>	<b>Type</b>	<b>Function</b>
104	P3.6	O0 / I	St/B	<b>Bit 6 of Port 3, General Purpose Input/Output</b>
	U2C1_DOUT	O1	St/B	<b>USIC2 Channel 1 Shift Data Output</b>
	TxDC4	O2	St/B	<b>CAN Node 4 Transmit Data Output</b>
	U0C0_SELO 6	O3	St/B	<b>USIC0 Channel 0 Select/Control 6 Output</b>
	U2C1_DX0A	I	St/B	<b>USIC2 Channel 1 Shift Data Input</b>
	U2C1_DX1B	I	St/B	<b>USIC2 Channel 1 Shift Clock Input</b>
105	P10.7	O0 / I	St/B	<b>Bit 7 of Port 10, General Purpose Input/Output</b>
	U0C1_DOUT	O1	St/B	<b>USIC0 Channel 1 Shift Data Output</b>
	CCU60_COUT T63	O2	St/B	<b>CCU60 Channel 3 Output</b>
	AD7	OH / I	St/B	<b>External Bus Interface Address/Data Line 7</b>
	U0C1_DX0B	I	St/B	<b>USIC0 Channel 1 Shift Data Input</b>
	CCU60_CCP OS0A	I	St/B	<b>CCU60 Position Input 0</b>
	RxDC4C	I	St/B	<b>CAN Node 4 Receive Data Input</b>
	T4INB	I	St/B	<b>GPT12E Timer T4 Count/Gate Input</b>
106	P0.7	O0 / I	St/B	<b>Bit 7 of Port 0, General Purpose Input/Output</b>
	U1C1_DOUT	O1	St/B	<b>USIC1 Channel 1 Shift Data Output</b>
	U1C0_SELO 3	O2	St/B	<b>USIC1 Channel 0 Select/Control 3 Output</b>
	TxDC3	O3	St/B	<b>CAN Node 3 Transmit Data Output</b>
	A7	OH	St/B	<b>External Bus Interface Address Line 7</b>
	U1C1_DX0B	I	St/B	<b>USIC1 Channel 1 Shift Data Input</b>
	CCU61_CTR APB	I	St/B	<b>CCU61 Emergency Trap Input</b>

**Table 9-18 Pin Definitions and Functions (cont'd)**

<b>Pin</b>	<b>Symbol</b>	<b>Ctrl.</b>	<b>Type</b>	<b>Function</b>
107	P3.7	O0 / I	St/B	<b>Bit 7 of Port 3, General Purpose Input/Output</b>
	U2C1_DOUT	O1	St/B	<b>USIC2 Channel 1 Shift Data Output</b>
	U2C0_SELO 3	O2	St/B	<b>USIC2 Channel 0 Select/Control 3 Output</b>
	U0C0_SELO 7	O3	St/B	<b>USIC0 Channel 0 Select/Control 7 Output</b>
	U2C1_DX0B	I	St/B	<b>USIC2 Channel 1 Shift Data Input</b>
111	P1.0	O0 / I	St/B	<b>Bit 0 of Port 1, General Purpose Input/Output</b>
	U1C0_MCLK OUT	O1	St/B	<b>USIC1 Channel 0 Master Clock Output</b>
	U1C0_SELO 4	O2	St/B	<b>USIC1 Channel 0 Select/Control 4 Output</b>
	A8	OH	St/B	<b>External Bus Interface Address Line 8</b>
	ESR1_3	I	St/B	<b>ESR1 Trigger Input 3</b>
	CCU62_CTR APB	I	St/B	<b>CCU62 Emergency Trap Input</b>
	T6INB	I	St/B	<b>GPT12E Timer T6 Count/Gate Input</b>
112	P9.0	O0 / I	St/B	<b>Bit 0 of Port 9, General Purpose Input/Output</b>
	CCU63_CC6 0	O1	St/B	<b>CCU63 Channel 0 Output</b>
	CCU63_CC6 0INA	I	St/B	<b>CCU63 Channel 0 Input</b>
	T6EUDB	I	St/B	<b>GPT12E Timer T6 External Up/Down Control Input</b>



**Table 9-18 Pin Definitions and Functions (cont'd)**

<b>Pin</b>	<b>Symbol</b>	<b>Ctrl.</b>	<b>Type</b>	<b>Function</b>
113	P10.8	O0 / I	St/B	<b>Bit 8 of Port 10, General Purpose Input/Output</b>
	U0C0_MCLK OUT	O1	St/B	<b>USIC0 Channel 0 Master Clock Output</b>
	U0C1_SELO 0	O2	St/B	<b>USIC0 Channel 1 Select/Control 0 Output</b>
	U2C1_DOUT	O3	St/B	<b>USIC2 Channel 1 Shift Data Output</b>
	AD8	OH / I	St/B	<b>External Bus Interface Address/Data Line 8</b>
	CCU60_CCP OS1A	I	St/B	<b>CCU60 Position Input 1</b>
	U0C0_DX1C	I	St/B	<b>USIC0 Channel 0 Shift Clock Input</b>
	BRKIN_B	I	St/B	<b>OCDS Break Signal Input</b>
	T3EUDB	I	St/B	<b>GPT12E Timer T3 External Up/Down Control Input</b>
114	P9.1	O0 / I	St/B	<b>Bit 1 of Port 9, General Purpose Input/Output</b>
	CCU63_CC6 1	O1	St/B	<b>CCU63 Channel 1 Output</b>
	CCU63_CC6 1INA	I	St/B	<b>CCU63 Channel 1 Input</b>
115	P10.9	O0 / I	St/B	<b>Bit 9 of Port 10, General Purpose Input/Output</b>
	U0C0_SELO 4	O1	St/B	<b>USIC0 Channel 0 Select/Control 4 Output</b>
	U0C1_MCLK OUT	O2	St/B	<b>USIC0 Channel 1 Master Clock Output</b>
	AD9	OH / I	St/B	<b>External Bus Interface Address/Data Line 9</b>
	CCU60_CCP OS2A	I	St/B	<b>CCU60 Position Input 2</b>
	TCK_B	I	St/B	<b>DAP0/JTAG Clock Input</b>
	T3INB	I	St/B	<b>GPT12E Timer T3 Count/Gate Input</b>

**Table 9-18 Pin Definitions and Functions (cont'd)**

<b>Pin</b>	<b>Symbol</b>	<b>Ctrl.</b>	<b>Type</b>	<b>Function</b>
116	P1.1	O0 / I	St/B	<b>Bit 1 of Port 1, General Purpose Input/Output</b>
	CCU62_COUT62	O1	St/B	<b>CCU62 Channel 2 Output</b>
	U1C0_SELO5	O2	St/B	<b>USIC1 Channel 0 Select/Control 5 Output</b>
	U2C1_DOUT	O3	St/B	<b>USIC2 Channel 1 Shift Data Output</b>
	A9	OH	St/B	<b>External Bus Interface Address Line 9</b>
	ESR2_3	I	St/B	<b>ESR2 Trigger Input 3</b>
	U2C1_DX0C	I	St/B	<b>USIC2 Channel 1 Shift Data Input</b>
117	P10.10	O0 / I	St/B	<b>Bit 10 of Port 10, General Purpose Input/Output</b>
	U0C0_SELO0	O1	St/B	<b>USIC0 Channel 0 Select/Control 0 Output</b>
	CCU60_COUT63	O2	St/B	<b>CCU60 Channel 3 Output</b>
	AD10	OH / I	St/B	<b>External Bus Interface Address/Data Line 10</b>
	U0C0_DX2C	I	St/B	<b>USIC0 Channel 0 Shift Control Input</b>
	U0C1_DX1A	I	St/B	<b>USIC0 Channel 1 Shift Clock Input</b>
	TDI_B	I	St/B	<b>JTAG Test Data Input</b>
118	P10.11	O0 / I	St/B	<b>Bit 11 of Port 10, General Purpose Input/Output</b>
	U1C0_SCLKOUT	O1	St/B	<b>USIC1 Channel 0 Shift Clock Output</b>
	$\overline{\text{BRKOUT}}$	O2	St/B	<b>OCDS Break Signal Output</b>
	U3C0_SELO0	O3	St/B	<b>USIC3 Channel 0 Select/Control 0 Output</b>
	AD11	OH / I	St/B	<b>External Bus Interface Address/Data Line 11</b>
	U1C0_DX1D	I	St/B	<b>USIC1 Channel 0 Shift Clock Input</b>
	RxDC2B	I	St/B	<b>CAN Node 2 Receive Data Input</b>
	TMS_B	I	St/B	<b>JTAG Test Mode Selection Input</b>
	U3C0_DX2A	I	St/B	<b>USIC3 Channel 0 Shift Control Input</b>

**Table 9-18 Pin Definitions and Functions (cont'd)**

<b>Pin</b>	<b>Symbol</b>	<b>Ctrl.</b>	<b>Type</b>	<b>Function</b>
119	P9.2	O0 / I	St/B	<b>Bit 2 of Port 9, General Purpose Input/Output</b>
	CCU63_CC6 2	O1	St/B	<b>CCU63 Channel 2 Output</b>
	CCU63_CC6 2INA	I	St/B	<b>CCU63 Channel 2 Input</b>
	CAPINB	I	St/B	<b>GPT12E Register CAPREL Capture Input</b>
120	P1.2	O0 / I	St/B	<b>Bit 2 of Port 1, General Purpose Input/Output</b>
	CCU62_CC6 2	O1	St/B	<b>CCU62 Channel 2 Output</b>
	U1C0_SELO 6	O2	St/B	<b>USIC1 Channel 0 Select/Control 6 Output</b>
	U2C1_SCLK OUT	O3	St/B	<b>USIC2 Channel 1 Shift Clock Output</b>
	A10	OH	St/B	<b>External Bus Interface Address Line 10</b>
	ESR1_4	I	St/B	<b>ESR1 Trigger Input 4</b>
	CCU61_T12 HRB	I	St/B	<b>External Run Control Input for T12 of CCU61</b>
	CCU62_CC6 2INA	I	St/B	<b>CCU62 Channel 2 Input</b>
	U2C1_DX0D	I	St/B	<b>USIC2 Channel 1 Shift Data Input</b>
	U2C1_DX1C	I	St/B	<b>USIC2 Channel 1 Shift Clock Input</b>
121	P10.12	O0 / I	St/B	<b>Bit 12 of Port 10, General Purpose Input/Output</b>
	U1C0_DOUT	O1	St/B	<b>USIC1 Channel 0 Shift Data Output</b>
	TxDC2	O2	St/B	<b>CAN Node 2 Transmit Data Output</b>
	TDO_B	OH / I	St/B	<b>JTAG Test Data Output / DAP1 Input/Output</b>
	AD12	OH / I	St/B	<b>External Bus Interface Address/Data Line 12</b>
	U1C0_DX0C	I	St/B	<b>USIC1 Channel 0 Shift Data Input</b>
	U1C0_DX1E	I	St/B	<b>USIC1 Channel 0 Shift Clock Input</b>
122	P9.3	O0 / I	St/B	<b>Bit 3 of Port 9, General Purpose Input/Output</b>
	CCU63_COU T60	O1	St/B	<b>CCU63 Channel 0 Output</b>
	BRKOUT	O2	St/B	<b>OCDS Break Signal Output</b>

**Table 9-18 Pin Definitions and Functions (cont'd)**

<b>Pin</b>	<b>Symbol</b>	<b>Ctrl.</b>	<b>Type</b>	<b>Function</b>
123	P10.13	O0 / I	St/B	<b>Bit 13 of Port 10, General Purpose Input/Output</b>
	U1C0_DOUT	O1	St/B	<b>USIC1 Channel 0 Shift Data Output</b>
	TxDC3	O2	St/B	<b>CAN Node 3 Transmit Data Output</b>
	U1C0_SELO 3	O3	St/B	<b>USIC1 Channel 0 Select/Control 3 Output</b>
	$\overline{\text{WR}}/\overline{\text{WRL}}$	OH	St/B	<b>External Bus Interface Write Strobe Output</b> Active for each external write access, when $\overline{\text{WR}}$ , active for ext. writes to the low byte, when $\overline{\text{WRL}}$ .
	U1C0_DX0D	I	St/B	<b>USIC1 Channel 0 Shift Data Input</b>
124	P1.3	O0 / I	St/B	<b>Bit 3 of Port 1, General Purpose Input/Output</b>
	CCU62_COU T63	O1	St/B	<b>CCU62 Channel 3 Output</b>
	U1C0_SELO 7	O2	St/B	<b>USIC1 Channel 0 Select/Control 7 Output</b>
	U2C0_SELO 4	O3	St/B	<b>USIC2 Channel 0 Select/Control 4 Output</b>
	A11	OH	St/B	<b>External Bus Interface Address Line 11</b>
	ESR2_4	I	St/B	<b>ESR2 Trigger Input 4</b>
	CCU62_T12 HRB	I	St/B	<b>External Run Control Input for T12 of CCU62</b>
125	P9.4	O0 / I	St/B	<b>Bit 4 of Port 9, General Purpose Input/Output</b>
	CCU63_COU T61	O1	St/B	<b>CCU63 Channel 1 Output</b>
	U2C0_DOUT	O2	St/B	<b>USIC2 Channel 0 Shift Data Output</b>
	CCU62_COU T63	O3	St/B	<b>CCU62 Channel 3 Output</b>

**Table 9-18 Pin Definitions and Functions (cont'd)**

<b>Pin</b>	<b>Symbol</b>	<b>Ctrl.</b>	<b>Type</b>	<b>Function</b>
126	P9.5	O0 / I	St/B	<b>Bit 5 of Port 9, General Purpose Input/Output</b>
	CCU63_COU T62	O1	St/B	<b>CCU63 Channel 2 Output</b>
	U2C0_DOUT	O2	St/B	<b>USIC2 Channel 0 Shift Data Output</b>
	CCU62_COU T62	O3	St/B	<b>CCU62 Channel 2 Output</b>
	U2C0_DX0E	I	St/B	<b>USIC2 Channel 0 Shift Data Input</b>
	CCU60_CCP OS2B	I	St/B	<b>CCU60 Position Input 2</b>
128	P10.14	O0 / I	St/B	<b>Bit 14 of Port 10, General Purpose Input/Output</b>
	U1C0_SELO 1	O1	St/B	<b>USIC1 Channel 0 Select/Control 1 Output</b>
	U0C1_DOUT	O2	St/B	<b>USIC0 Channel 1 Shift Data Output</b>
	U3C0_SCLK OUT	O3	St/B	<b>USIC3 Channel 0 Shift Clock Output</b>
	$\overline{\text{RD}}$	OH	St/B	<b>External Bus Interface Read Strobe Output</b>
	ESR2_2	I	St/B	<b>ESR2 Trigger Input 2</b>
	U0C1_DX0C	I	St/B	<b>USIC0 Channel 1 Shift Data Input</b>
	RxDC3C	I	St/B	<b>CAN Node 3 Receive Data Input</b>
	U3C0_DX1A	I	St/B	<b>USIC3 Channel 0 Shift Clock Input</b>
129	P1.4	O0 / I	St/B	<b>Bit 4 of Port 1, General Purpose Input/Output</b>
	CCU62_COU T61	O1	St/B	<b>CCU62 Channel 1 Output</b>
	U1C1_SELO 4	O2	St/B	<b>USIC1 Channel 1 Select/Control 4 Output</b>
	U2C0_SELO 5	O3	St/B	<b>USIC2 Channel 0 Select/Control 5 Output</b>
	A12	OH	St/B	<b>External Bus Interface Address Line 12</b>
	U2C0_DX2B	I	St/B	<b>USIC2 Channel 0 Shift Control Input</b>
	RxDC5A	I	St/B	<b>CAN Node 5 Receive Data Input</b>

**Table 9-18 Pin Definitions and Functions (cont'd)**

<b>Pin</b>	<b>Symbol</b>	<b>Ctrl.</b>	<b>Type</b>	<b>Function</b>
130	P10.15	O0 / I	St/B	<b>Bit 15 of Port 10, General Purpose Input/Output</b>
	U1C0_SELO2	O1	St/B	<b>USIC1 Channel 0 Select/Control 2 Output</b>
	U0C1_DOUT	O2	St/B	<b>USIC0 Channel 1 Shift Data Output</b>
	U1C0_DOUT	O3	St/B	<b>USIC1 Channel 0 Shift Data Output</b>
	ALE	OH	St/B	<b>External Bus Interf. Addr. Latch Enable Output</b>
	U0C1_DX1C	I	St/B	<b>USIC0 Channel 1 Shift Clock Input</b>
131	P1.5	O0 / I	St/B	<b>Bit 5 of Port 1, General Purpose Input/Output</b>
	CCU62_COUT60	O1	St/B	<b>CCU62 Channel 0 Output</b>
	U1C1_SELO3	O2	St/B	<b>USIC1 Channel 1 Select/Control 3 Output</b>
	BRKOUT	O3	St/B	<b>OCDS Break Signal Output</b>
	A13	OH	St/B	<b>External Bus Interface Address Line 13</b>
	U2C0_DX0C	I	St/B	<b>USIC2 Channel 0 Shift Data Input</b>
132	P9.6	O0 / I	St/B	<b>Bit 6 of Port 9, General Purpose Input/Output</b>
	CCU63_COUT63	O1	St/B	<b>CCU63 Channel 3 Output</b>
	CCU63_COUT62	O2	St/B	<b>CCU63 Channel 2 Output</b>
	CCU62_COUT61	O3	St/B	<b>CCU62 Channel 1 Output</b>
	CCU63_CTRAPA	I	St/B	<b>CCU63 Emergency Trap Input</b>
	CCU60_CCPOS1B	I	St/B	<b>CCU60 Position Input 1</b>

**Table 9-18 Pin Definitions and Functions (cont'd)**

<b>Pin</b>	<b>Symbol</b>	<b>Ctrl.</b>	<b>Type</b>	<b>Function</b>
133	P1.6	O0 / I	St/B	<b>Bit 6 of Port 1, General Purpose Input/Output</b>
	CCU62_CC6 1	O1 / I	St/B	<b>CCU62 Channel 1 Output</b>
	U1C1_SELO 2	O2	St/B	<b>USIC1 Channel 1 Select/Control 2 Output</b>
	U2C0_DOUT	O3	St/B	<b>USIC2 Channel 0 Shift Data Output</b>
	A14	OH	St/B	<b>External Bus Interface Address Line 14</b>
	U2C0_DX0D	I	St/B	<b>USIC2 Channel 0 Shift Data Input</b>
	CCU62_CC6 1INA	I	St/B	<b>CCU62 Channel 1 Input</b>
134	P9.7	O0 / I	St/B	<b>Bit 7 of Port 9, General Purpose Input/Output</b>
	CCU62_COU T60	O1	St/B	<b>CCU62 Channel 0 Output</b>
	CCU62_COU T63	O2	St/B	<b>CCU62 Channel 3 Output</b>
	CCU63_CTR APB	I	St/B	<b>CCU63 Emergency Trap Input</b>
	U2C0_DX1D	I	St/B	<b>USIC2 Channel 0 Shift Clock Input</b>
	CCU60_CCP OS0B	I	St/B	<b>CCU60 Position Input 0</b>
135	P1.7	O0 / I	St/B	<b>Bit 7 of Port 1, General Purpose Input/Output</b>
	CCU62_CC6 0	O1	St/B	<b>CCU62 Channel 0 Output</b>
	U1C1_MCLK OUT	O2	St/B	<b>USIC1 Channel 1 Master Clock Output</b>
	U2C0_SCLK OUT	O3	St/B	<b>USIC2 Channel 0 Shift Clock Output</b>
	A15	OH	St/B	<b>External Bus Interface Address Line 15</b>
	U2C0_DX1C	I	St/B	<b>USIC2 Channel 0 Shift Clock Input</b>
	CCU62_CC6 0INA	I	St/B	<b>CCU62 Channel 0 Input</b>
	RxDC4E	I	St/B	<b>CAN Node 4 Receive Data Input</b>
136	XTAL2	O	Sp/M	<b>Crystal Oscillator Amplifier Output</b>

**Table 9-18 Pin Definitions and Functions (cont'd)**

Pin	Symbol	Ctrl.	Type	Function
137	XTAL1	I	Sp/M	<b>Crystal Oscillator Amplifier Input</b> To clock the device from an external source, drive XTAL1, while leaving XTAL2 unconnected. Voltages on XTAL1 must comply to the core supply voltage $V_{DDIM}$ .
	ESR2_9	I	St/B	<b>ESR2 Trigger Input 9</b>
138	PORST	I	In/B	<b>Power On Reset Input</b> A low level at this pin resets the XE16xyM completely. A spike filter suppresses input pulses <10 ns. Input pulses >100 ns safely pass the filter. The minimum duration for a safe recognition should be 120 ns. An internal pullup device will hold this pin high when nothing is driving it.
139	ESR1	O0 / I	St/B	<b>External Service Request 1</b>
	RxDC0E	I	St/B	<b>CAN Node 0 Receive Data Input</b>
	U1C0_DX0F	I	St/B	<b>USIC1 Channel 0 Shift Data Input</b>
	U1C0_DX2C	I	St/B	<b>USIC1 Channel 0 Shift Control Input</b>
	U1C1_DX0C	I	St/B	<b>USIC1 Channel 1 Shift Data Input</b>
	U1C1_DX2B	I	St/B	<b>USIC1 Channel 1 Shift Control Input</b>
	U2C1_DX2C	I	St/B	<b>USIC2 Channel 1 Shift Control Input</b>



**Table 9-18 Pin Definitions and Functions (cont'd)**

Pin	Symbol	Ctrl.	Type	Function
140	$\overline{\text{ESR2}}$	O0 / I	St/B	<b>External Service Request 2</b>
	RxDC1E	I	St/B	<b>CAN Node 1 Receive Data Input</b>
	CCU60_CTR APC	I	St/B	<b>CCU60 Emergency Trap Input</b>
	CCU61_CTR APC	I	St/B	<b>CCU61 Emergency Trap Input</b>
	CCU62_CTR APC	I	St/B	<b>CCU62 Emergency Trap Input</b>
	CCU63_CTR APC	I	St/B	<b>CCU63 Emergency Trap Input</b>
	U1C1_DX0D	I	St/B	<b>USIC1 Channel 1 Shift Data Input</b>
	U1C1_DX2C	I	St/B	<b>USIC1 Channel 1 Shift Control Input</b>
	U2C1_DX0E	I	St/B	<b>USIC2 Channel 1 Shift Data Input</b>
	U2C1_DX2B	I	St/B	<b>USIC2 Channel 1 Shift Control Input</b>
141	$\overline{\text{ESR0}}$	O0 / I	St/B	<b>External Service Request 0</b>  <i>Note: After power-up, ESR0 operates as open-drain bidirectional reset with a weak pull-up.</i>
	U1C0_DX0E	I	St/B	<b>USIC1 Channel 0 Shift Data Input</b>
	U1C0_DX2B	I	St/B	<b>USIC1 Channel 0 Shift Control Input</b>
142	P8.6	O0 / I	St/B	<b>Bit 6 of Port 8, General Purpose Input/Output</b>
	CCU60_COU T63	O1	St/B	<b>CCU60 Channel 3 Output</b>
	MCHK_MAT CH	O3	St/B	<b>Memory Checker Match Output</b>
	CCU60_CTR APB	I	St/B	<b>CCU60 Emergency Trap Input</b>
	$\overline{\text{BRKIN\_D}}$	I	St/B	<b>OCDS Break Signal Input</b>
	CCU62_CTR APD	I	St/B	<b>CCU62 Emergency Trap Input</b>

**Table 9-18 Pin Definitions and Functions (cont'd)**

Pin	Symbol	Ctrl.	Type	Function
143	P8.5	O0 / I	St/B	<b>Bit 5 of Port 8, General Purpose Input/Output</b>
	CCU60_COUT62	O1	St/B	<b>CCU60 Channel 2 Output</b>
	CCU62_CC62	O2	St/B	<b>CCU62 Channel 2 Output</b>
	TCK_D	I	St/B	<b>DAP0/JTAG Clock Input</b>
	CCU62_CC62INB	I	St/B	<b>CCU62 Channel 2 Input</b>
15	$V_{DDIM}$	-	PS/M	<b>Digital Core Supply Voltage for Domain M</b> Decouple with a ceramic capacitor, see Data Sheet for details.
54, 91, 127	$V_{DDI1}$	-	PS/1	<b>Digital Core Supply Voltage for Domain 1</b> Decouple with a ceramic capacitor, see Data Sheet for details. All $V_{DDI1}$ pins must be connected to each other.
20	$V_{DDPA}$	-	PS/A	<b>Digital Pad Supply Voltage for Domain A</b> Connect decoupling capacitors to adjacent $V_{DDP}/V_{SS}$ pin pairs as close as possible to the pins. <i>Note: The A/D_Converters and ports P5, P6 and P15 are fed from supply voltage <math>V_{DDPA}</math>.</i>

**Table 9-18 Pin Definitions and Functions (cont'd)**

Pin	Symbol	Ctrl.	Type	Function
2, 36, 38, 72, 74, 108, 110, 144	$V_{DDPB}$	-	PS/B	<b>Digital Pad Supply Voltage for Domain B</b> Connect decoupling capacitors to adjacent $V_{DDP}/V_{SS}$ pin pairs as close as possible to the pins. <i>Note: The on-chip voltage regulators and all ports except P5, P6 and P15 are fed from supply voltage <math>V_{DDPB}</math>.</i>
1, 37, 73, 109	$V_{SS}$	-	PS/--	<b>Digital Ground</b> All $V_{SS}$ pins must be connected to the ground-line or ground-plane. <i>Note: Also the exposed pad is connected internally to <math>V_{SS}</math>. To improve the EMC behavior, it is recommended to connect the exposed pad to the board ground.</i> <i>For thermal aspects, please refer to the Data Sheet. Board layout examples are given in an application note.</i>

<sup>1)</sup> To generate the reference clock output for bus timing measurement,  $f_{SYS}$  must be selected as source for EXTCLK and P2.8 must be selected as output pin. Also the high-speed clock pad must be enabled. This configuration is referred to as reference clock output signal CLKOUT.

## 10 Dedicated Pins

Most of the input/output or control signals of the functional the XE16xyM are realized as alternate functions of pins of the parallel ports. There is, however, a number of signals that use separate pins, including the oscillator, special control signals and, of course, the power supply.

**Table 10-1** summarizes the dedicated pins of the XE16xyM.

**Table 10-1 XE16xyM Dedicated Pins**

Pin(s)	Function
$\overline{\text{PORST}}$	Power-On Reset Input
$\overline{\text{ESR0}}$	External Service Request Input 0
$\overline{\text{ESR1}}$	External Service Request Input 1
$\overline{\text{ESR2}}$	External Service Request Input 2
XTAL1, XTAL2	Oscillator Input/Output (main oscillator)
$\overline{\text{TESTM}}$	Test Mode Enable
$\overline{\text{TRST}}$	Test-System Reset Input
$V_{\text{AREF}_x}, V_{\text{AGND}}$	Reference voltages for the Analog/Digital Converter(s)
$V_{\text{DDIM}}$	Digital Core Supply for Domain M (1 pin)
$V_{\text{DDI1}}$	Digital Core Supply for Domain 1 (3 pins)
$V_{\text{DDPA}}$	Digital Pad Supply for Domain A including ADCs (1 pin)
$V_{\text{DDPB}}$	Digital Pad Supply for Domain B (8 pins)
$V_{\text{SS}}$	Digital Ground (4 pins)

**The Power-On Reset Input  $\overline{\text{PORST}}$**  allows to put the XE16xyM into the well defined reset condition either at power-up or external events like a hardware failure or manual reset.

**The External Service Request Inputs  $\overline{\text{ESR0}}$ ,  $\overline{\text{ESR1}}$ , and  $\overline{\text{ESR2}}$**  can be used for several system-related functions:

- trigger interrupt or trap (Class A or Class B) requests via an external signal (e.g. a power-fail signal)
- generate wake-up request signals
- generate hardware reset requests ( $\overline{\text{ESR0}}$  is bidirectional by default,  $\overline{\text{ESR1}}$  and  $\overline{\text{ESR2}}$  can optionally output a reset signal)
- data/control input for CCU6x, MultiCAN, and USIC ( $\overline{\text{ESR1}}$  or  $\overline{\text{ESR2}}$ )
- software-controlled input/output signal

## Dedicated Pins

**The Oscillator Input XTAL1 and Output XTAL2** connect the internal **Main Oscillator** to the external crystal. The oscillator provides an inverter and a feedback element. The standard external oscillator circuitry comprises the crystal, two low end capacitors and series resistor to limit the current through the crystal. The main oscillator is intended for the generation of a high-precision operating clock signal for the XE16xyM.

An external clock signal may be fed to the input XTAL1, leaving XTAL2 open. The current logic state of input XTAL1 can be read via a status flag, so XTAL1 can be used as digital input if neither the oscillator interface nor the clock input is required.

*Note: Pin XTAL1 belongs to the core power domain DMP\_M. All input signals, therefore, must be within the core voltage range.*

**The Test Mode Input TESTM** puts the XE16xyM into a test mode, which is used during the production tests of the device. In test mode, the XE16xyM behaves different from normal operation. Therefore, pin TESTM must be held HIGH (connect to  $V_{DDPB}$ ) for normal operation in an application system.

**The Test Reset Input TRST** puts the XE16xyM's debug system into reset state. During normal operation this input should be held low. For debugging purposes the on-chip debugging system can be enabled by driving pin TRST high at the rising edge of PORST.

**The Analog Reference Voltage Supply pins  $V_{AREFX}$  and  $V_{AGND}$**  provide separate reference voltage for the on-chip Analog/Digital-Converter(s). This reduces the noise that is coupled to the analog input signals from the digital logic sections and so improves the stability of the conversion results, when  $V_{AREF}$  and  $V_{AGND}$  are properly decoupled from  $V_{DD}$  and  $V_{SS}$ . Also, because conversion results are generated in relation to the reference voltages, ratiometric conversions are easily achieved.

*Note: Channel 0 of each module can be used as an alternate reference voltage input.*

**The Core Supply pins  $V_{DDIM}/V_{DDI1}$**  serve two purposes: While the on-chip EVVRs provide the power for the core logic of the XE16xyM these pins connect the EVVRs to their external buffer capacitors. For external supply, the core voltage is applied to these pins. The respective  $V_{DDI}/V_{SS}$  pairs should be decoupled as close to the pins as possible. Use ceramic capacitors and observe their values recommended in the respective Data Sheet.

**The Power Supply pins  $V_{DDPA}/V_{DDPB}$**  provide the power supply for all the analog and digital logic of the XE16xyM. Each power domain (DMP\_A and DMP\_B) can be supplied with an arbitrary voltage within the specified supply voltage range (please refer to the corresponding Data Sheets). These pins supply the output drivers as well as the on-chip EVVRs ( $V_{DDPB}$ ), except for external core voltage supply. The respective  $V_{DDP}/V_{SS}$  pairs should be decoupled as close to the pins as possible.

## **Dedicated Pins**

**The Ground Reference pins**  $V_{SS}$  provide the ground reference voltage for the power supplies as well as the reference voltage for the input signals.

*Note: All  $V_{DDx}$  pins and all  $V_{SS}$  pins must be connected to the power supplies and ground, respectively.*

## **11 External Bus Controller (EBC)**

The EBC enables the C166SV2 CPU access to chip external and internal peripherals and memories. The access can be of program fetch type or data exchange. If used to interface to the chip external world the external bus is also referred to as EXTBUS if used with internal (local) components it is also referred to as LXBUS.

### **11.1 Summary of Features**

The EBC functional and timing behavior is widely configurable so that it can be tailored to fit into a large range of applications.

- Demultiplexed and multiplexed mode of operation
- Up to 24 address lines
- 8-bit or 16-bit data bus
- Synchronous and asynchronous ready
- External bus arbitration support
- Up to 8 bus channels
- Address window programmable for up to 7 channels
- Bus timing and function programmable for each channel

### **11.2 Overview**

The function and timing characteristics of the EBC are controlled by a set of configuration registers.

The basic and general behavior is programmed via the mode selection registers EBCMOD0 and EBCMOD1.

The EBC supports up to eight ( $x=0\dots7$ ) bus channels linked to a dedicated chip select ( $\overline{CSx}$ ). Each channel is programmable by a dedicated set of registers. The FCONCSx registers specify the functional characteristics while the TCONCSx registers specify the cycle timing. The address area assigned to a channel is definable for seven channels by the ADDRSEL(1...7) registers. The remaining uncovered address areas are assigned to  $\overline{CS0}$ .

External  $\overline{CSx}$  signals can be used in order to save external glue logic. Access to non timing deterministic external devices is supported by a particular READY functionality. A HOLD/HLDA protocol is available for bus arbitration.

The external bus timing is related to the reference CLock OUTput (CLKOUT) signal. All bus signals are generated in relation to the rising edge of this clock. The external bus protocol is compatible with those of the C166 family. However, the external bus timing is improved in terms of wait state granularity and signal flexibility.

## **11.3 Naming Conventions**

For description of EBC timing and functions the following bus signal names will be used.

Control signals:

- ALE - Address Latch Enable (high active). Indicates that the applied address is valid.
- CS - Chip select.
- WR/WRL - Write Strobe, Write Low Byte Strobe (low active). Configured either to a general write request or a write request for the low byte.
- BHE/WRH - Byte High Enable, Write High Byte Strobe (low active). Configured either to an enable for the high byte or a write request for the high byte.
- RD - Read Strobe (low active).
- READY - Ready to indicate end of actions (programmable polarity).
- HOLD - Hold input for foreign bus requests (low active).
- HLDA - Hold Acknowledge (low active), master output to grant bus.
- BREQ - Bus Request (low active).

Bus signals:

- A or ADDR - Address bus.
- D or DATA - Data bus.
- AD - Shared Data/Address[15:0] bus.

## **11.4 Timing Description**

Bus characteristics can be programmed to the following access modes:

- 16-bit data bus with address not multiplexed
- 16-bit data bus with address multiplexed
- 8-bit data bus with address not multiplexed
- 8-bit data bus with address multiplexed

Multiplexed mode means that the data bus is used in a time multiplex mode for address (16 LSB only) and for data. In demultiplexed mode the data bus is used for data only and an additional address bus must be made available.

### **11.4.1 Bus Phases**

The external bus timing is defined by six different timing phases (A-F). These phases influence the control signals needed for any access sequence to a bus device.

At the beginning of a phase the output signals change within a defined output delay time. The particular delay times are specified in the XE16xyM data book. After the output delay the values of the control output signals are stable within this phase. Each phase can occupy a programmable number of CLKOUT cycles.



### **A Phase - $\overline{\text{CS}}$ Change Phase**

The A phase can take 0-3 clocks. It is used for tristating databus drivers from the previous cycle (tristate wait states after chip select switch).

A phase cycles are not inserted at every access cycle, but only when changing the  $\overline{\text{CS}}$ . If an access using one  $\overline{\text{CS}}$  ( $\overline{\text{CSx}}$ ) ends and the next access with a different  $\overline{\text{CS}}$  ( $\overline{\text{CSy}}$ ) is started, then A phase cycles are performed according to the bits set in the **first**  $\overline{\text{CS}}$  ( $\overline{\text{CSx}}$ ). This feature is used to optimize wait states with devices having a long turn-off delay at their databus drivers, such as EPROMs and flash memories.

The A phase cycles are inserted while the addresses and ALE of the next cycle are already applied.

If there are some idle cycles between two accesses, these clocks are taken into account and the A phase is shortened accordingly. For example, if there are three tristate cycles programmed and two idle cycles occur, then the A phase takes only one clock.

### **B Phase - Address Setup / ALE Phase**

The B phase can take 1-2 clocks. It is used for addressing devices before giving a command, and defines the length of time that ALE is active. In multiplexed bus mode, the address is applied for latching.

### **C Phase - Delay Phase**

The C phase is similar to the A and B phases but ALE is already low. It can take 0-3 clocks. In multiplexed bus mode, the address is held in order to be latched safely. Phase C cycles can be used to delay the command signals (RW delay).

### **D Phase - Write Data Setup / Mux Tristate Phase**

The D phase can take 0-1 clocks. It is used to tristate the address on the multiplexed bus when a read cycle is performed. For all write cycles, it is used to ensure that the data are valid on the bus before the command is applied.

### **E Phase - $\overline{\text{RD}}$ / $\overline{\text{WR}}$ Command Phase**

The E phase is the command or access phase, and takes 1-32 clocks. Read data are fetched, write data are put onto the bus, and the command signals are active. Read data are registered with the terminating clock of this phase.

The READY function lengthens this phase, too. READY-controlled access cycles may have an unlimited cycle time.

### **F Phase - Address / Write Data Hold Phase**

The F phase is at the end of an access. It can take 0-3 clocks.

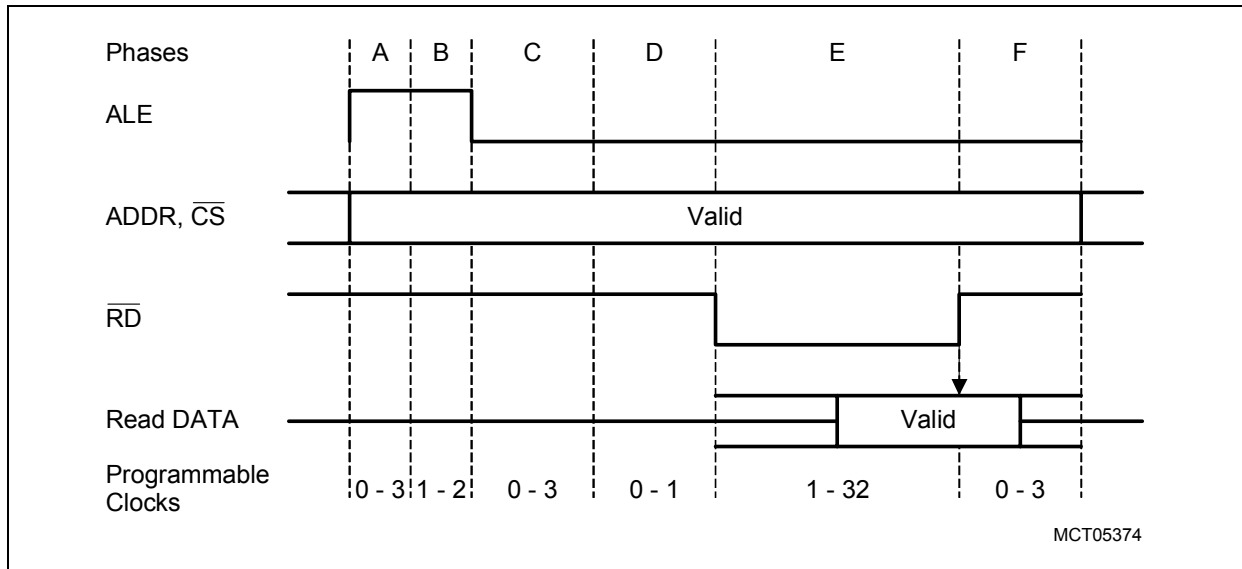
Addresses and write data are held while the command is inactive. The number of wait states inserted during the F phase is independently programmable for read and write

**External Bus Controller (EBC)**

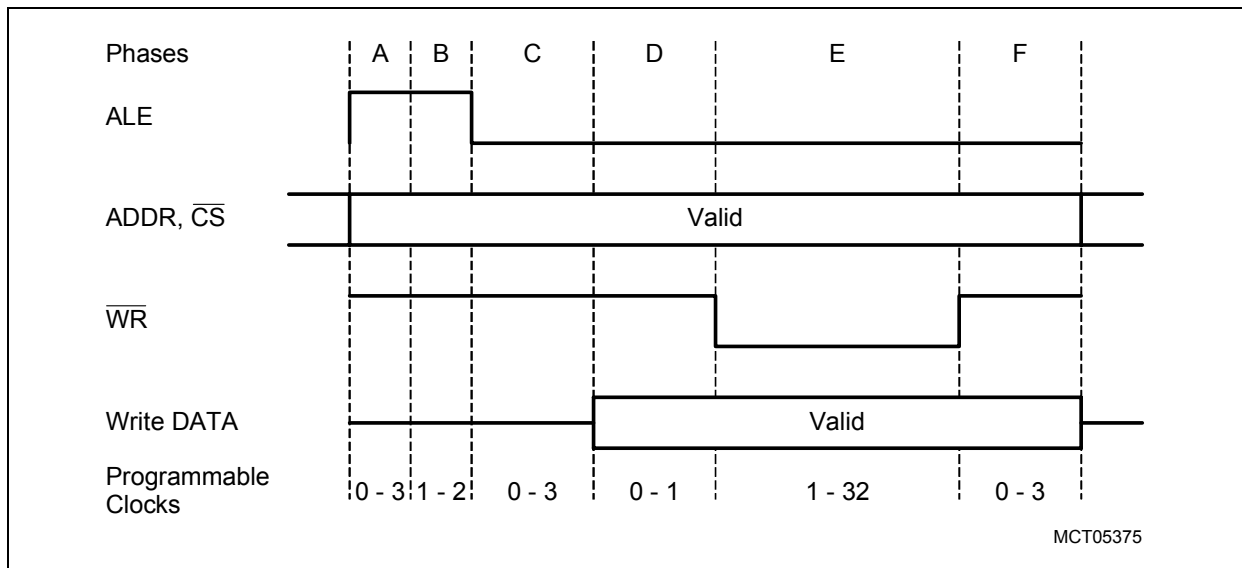
accesses. The F phase is used to program tristate wait states on the bidirectional data bus in order to avoid bus conflicts.

### 11.4.2 Demultiplexed Bus

General timing diagrams of a read and a write demultiplexed access are shown below.



**Figure 11-1 Demultiplexed Bus Read**

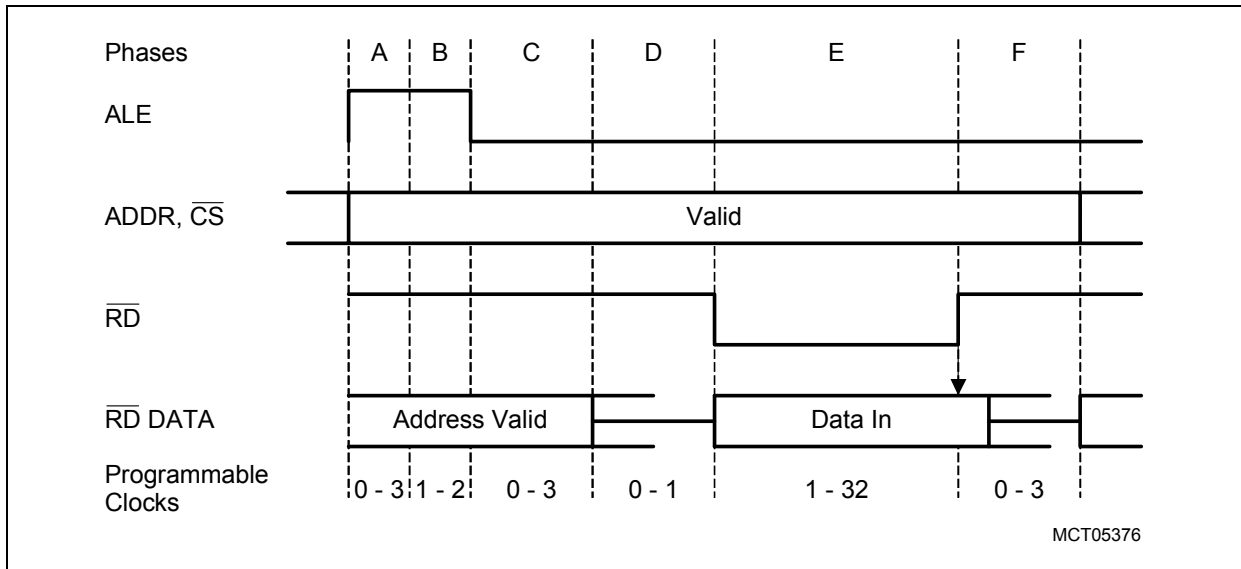


**Figure 11-2 Demultiplexed Bus Write**

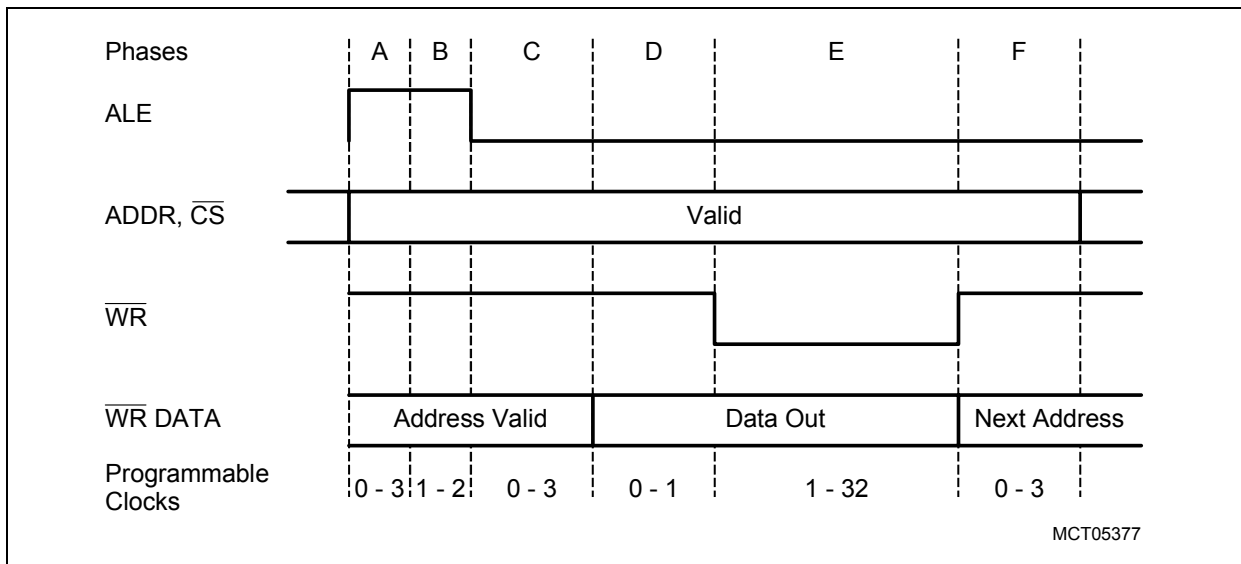
- A phase: Addresses valid, ALE high, no command.  $\overline{CS}$  switch tristate wait states
- B phase: Addresses valid, ALE high, no command. ALE length
- C phase: Addresses valid, ALE low, no command. R/W delay
- D phase: Write data valid, ALE low, no command. Data valid for write cycles
- E phase: Command (read or write) active. Access time
- F phase: Command inactive, address hold. Read data tristate time, write data hold time

### 11.4.3 Multiplexed Bus

General timing diagrams of a read and a write multiplexed access are shown below.



**Figure 11-3 Multiplexed Bus Read**



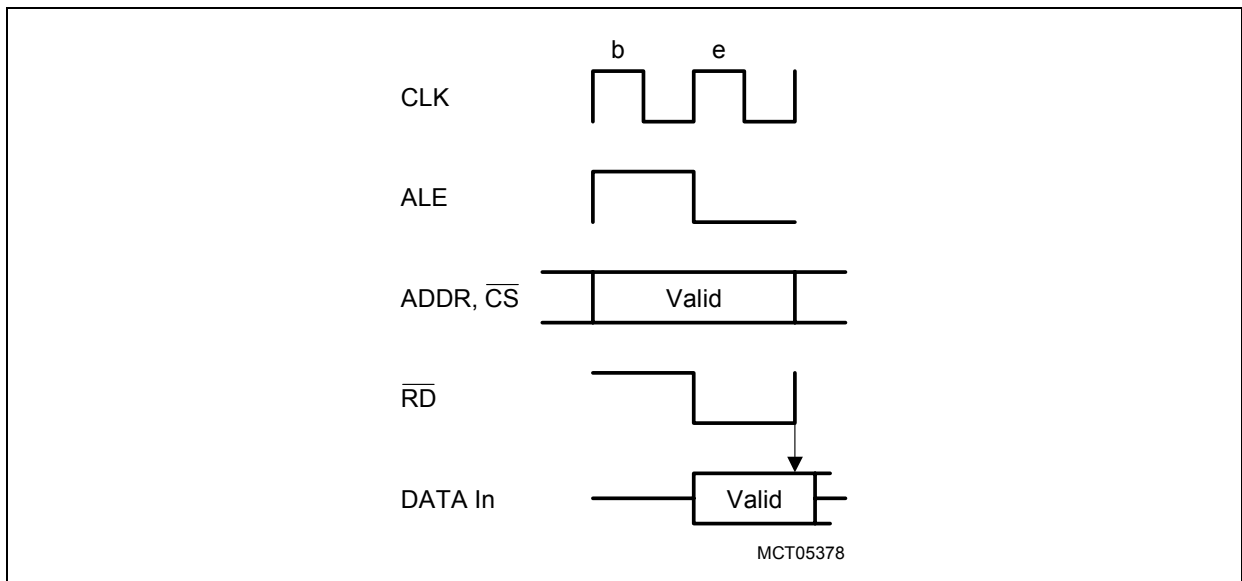
**Figure 11-4 Multiplexed Bus Write**

- A phase: addresses valid, ALE high, no command.  $\overline{CS}$  switch tristate wait states
- B phase: addresses valid, ALE high, no command. ALE length
- C phase: addresses valid, ALE low, no command. Address hold, R/W delay
- D phase: address tristate for read cycles, data valid for write cycles, ALE low, no command
- E phase: command (read or write) active. Access time

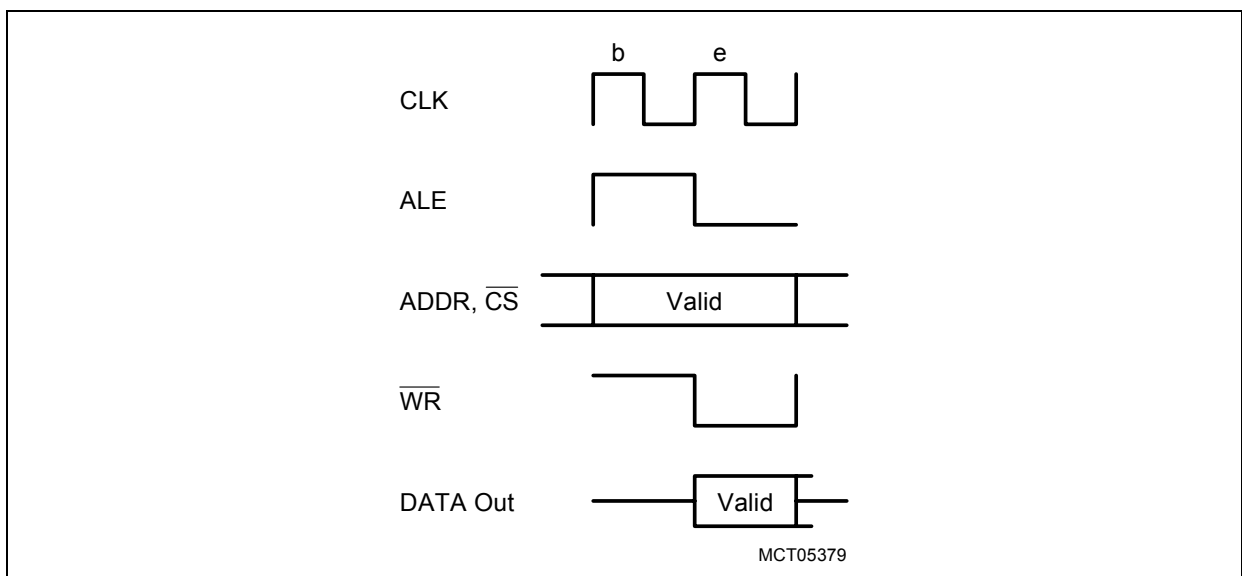
- F phase: command inactive, address hold. Read data tristate time, write data hold time.

#### 11.4.4 Fastest Access Cycles

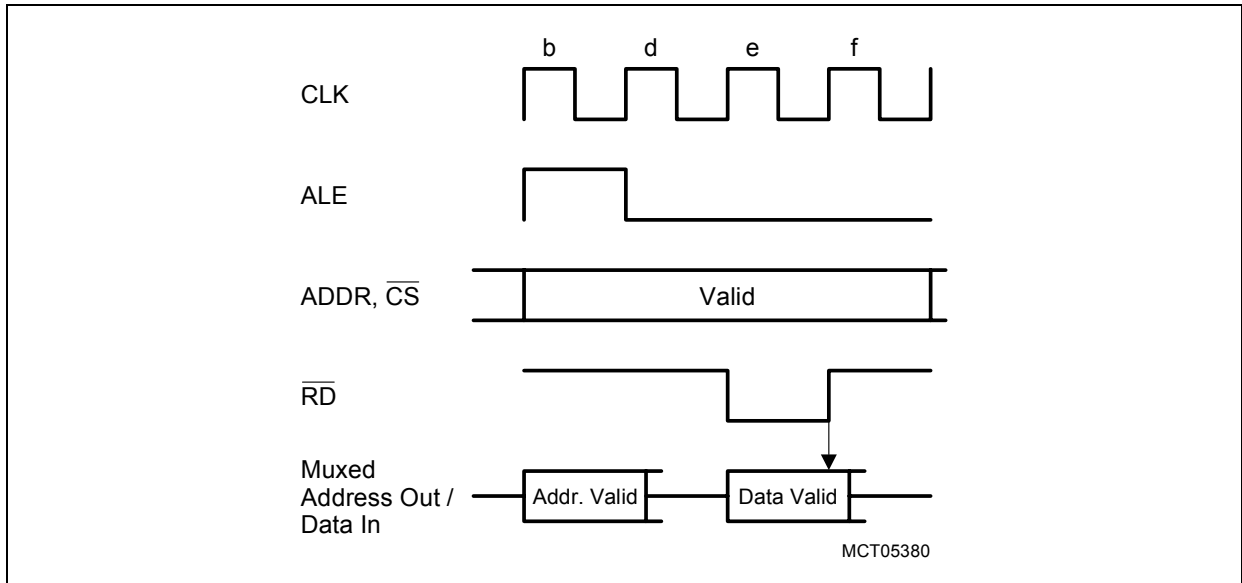
The fastest possible bus cycle in a system depends also on the pad timing. Therefore, the number of required cycles for a bus access depends on the current system frequency. The minimum bus cycles shown below cannot be achieved at very high system frequencies.



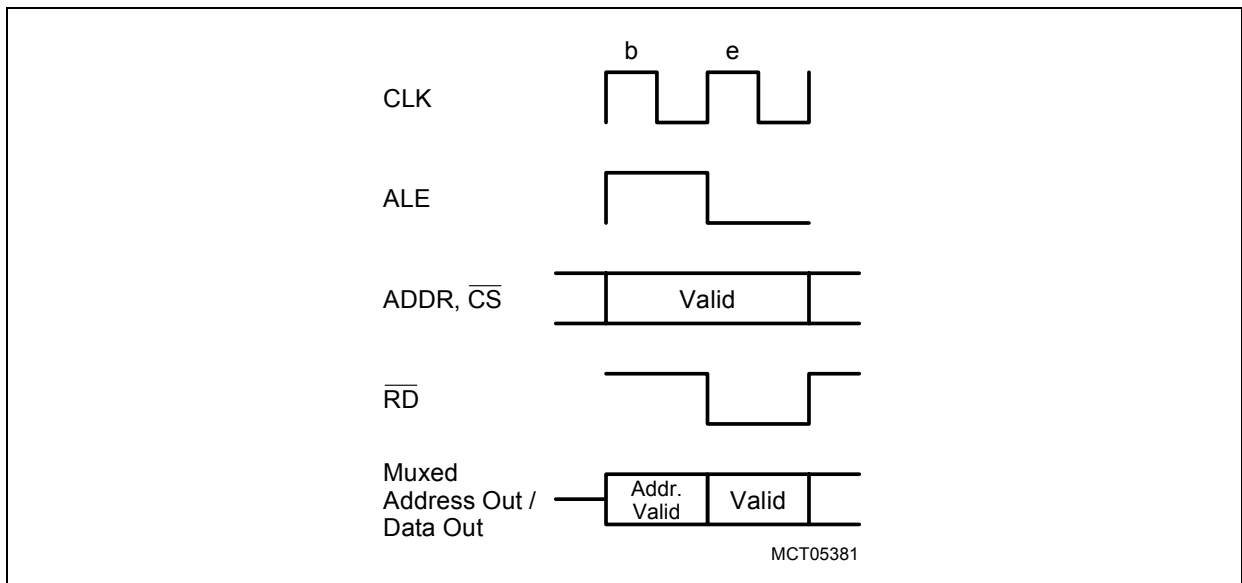
**Figure 11-5 Fastest Read Cycle Demultiplexed Bus**



**Figure 11-6 Fastest Write Cycle Demultiplexed Bus**



**Figure 11-7 Fastest Read Cycle Multiplexed Bus**



**Figure 11-8 Fastest Write Cycle Multiplexed Bus**

## 11.5 Address Windows

The EBC provides up to 8 logical bus channels. For 7 of these (assigned to  $\overline{CS1} \dots \overline{CS7}$ ) the allocated address window can be programmed by the ADDRSEL(1...7) registers. The remaining channel (assigned to  $\overline{CS0}$ ) has no address select register assigned and covers the remaining EBC address space.

Note that not the complete XE16xyM address space can be accessed by the EBC. For an overview on allocation of the complete address space please refer to the "Memory Organization" chapter.

### 11.5.1 Window Allocation

The size and start address of an address window is defined according to [Table 11-1](#). The size of the window is chosen by ADDRSELx.RGSZ. Depending on the size the relevant bits of ADDRSELx.RGSAD (marked with 'R') are used to select the corresponding window start address. The lower bits of ADDRSELx.RGSAD (marked 'x') are disregarded.

A programmed address windows must additionally be enabled by setting the corresponding FCONCSx.ENCS bit.

**Table 11-1 Address Range and Size for ADDRSELx**

ADDRSELx		Address Window	
Range Size RGSZ	Relevant (R) Bits of RGSAD	Selected Address Range	Range Start Address A[23:0] Selected with R-bits of RGSAD
3 ... 0	15 ... 4	Size	A23 ... A0
0000	RRRR RRRR RRRR	4 Kbytes	RRRR RRRR RRRR 0000 0000 0000
0001	RRRR RRRR RRRx	8 Kbytes	RRRR RRRR RRR0 0000 0000 0000
0010	RRRR RRRR RRxx	16 Kbytes	RRRR RRRR RR00 0000 0000 0000
0011	RRRR RRRR Rxxx	32 Kbytes	RRRR RRRR R000 0000 0000 0000
0100	RRRR RRRR xxxx	64 Kbytes	RRRR RRRR 0000 0000 0000 0000
0101	RRRR RRRx xxxx	128 Kbytes	RRRR RRR0 0000 0000 0000 0000
0110	RRRR RRxx xxxx	256 Kbytes	RRRR RR00 0000 0000 0000 0000
0111	RRRR Rxxx xxxx	512 Kbytes	RRRR R000 0000 0000 0000 0000
1000	RRRR xxxx xxxx	1 Mbytes	RRRR 0000 0000 0000 0000 0000
1001	RRRx xxxx xxxx	2 Mbytes	RRR0 0000 0000 0000 0000 0000
1010	RRxx xxxx xxxx	4 Mbytes	RR00 0000 0000 0000 0000 0000
1011	Rxxx xxxx xxxx	8 Mbytes	R000 0000 0000 0000 0000 0000
11xx	xxxx xxxx xxxx	reserved	---- ---- ---- ---- ---- ----

### **11.5.2 Window Overlap**

Since it is possible to program overlapping address areas an arbitration scheme is defined to handle these cases. For each access directed to the EBC it compares the current address with all address select registers of enabled windows. This comparison is done in three levels.

#### **Priority 1:**

Registers ADDRSEL<sub>x</sub> [ $x = 2, 4, 6$ ] of enabled windows are evaluated first. Upon a match the access starts on the respective channel. Overlap of windows within this group will lead to undefined behavior.

#### **Priority 2:**

Registers ADDRSEL<sub>y</sub> [ $y = 1, 3, 5, 7$ ] of enabled windows are evaluated in this step. Upon a match the access starts on the respective channel. Overlap of windows within this group will lead to undefined behavior. Overlaps with priority 1 ADDRSEL<sub>x</sub> are only allowed for the (x,y) pairs (2,1), (4,3) and (6, 5).

#### **Priority 3:**

If channel 0 is enabled the access is directed to it. Otherwise no bus action occurs.



## 11.6 Ready Controlled Bus Access

In cases, where the response time of a peripheral is not constant, or where the programmable wait states are not enough, the XE16xyM EBC provides the so called READY controlled bus access scheme.

In this scheme bus accesses are terminated by the READY input signal. During phase E the EBC first counts a programmable number of clock cycles (1...32) and then starts in the last wait cycle to monitor the internal READY line ("READY Int" in [Figure 11-9](#)) to determine the actual end of the current bus cycle. The external device drives READY active in order to indicate that data has been latched (write cycle) or is available (read cycle).

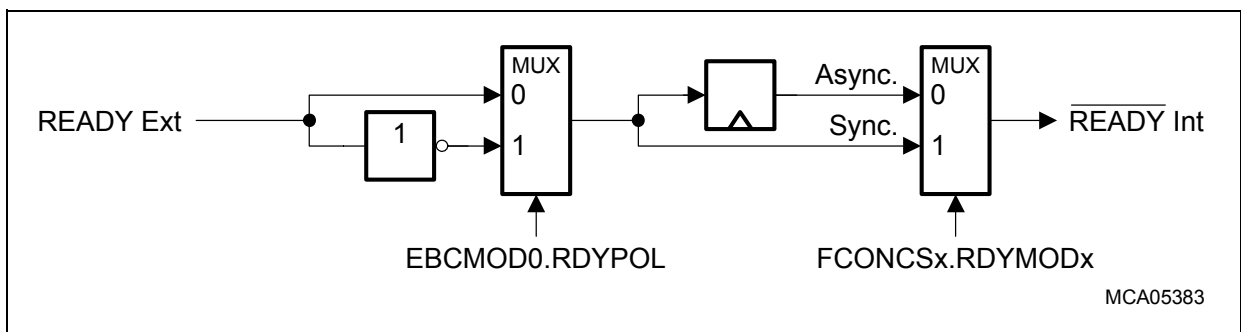
A READY controlled bus cycle requires one synchronization cycle to terminate. Programmed phase F cycles include this synchronization cycle. Therefore setting TCONCSx phase F to 0 clock cycles will have the same effect as setting to 1 clock cycle.

### 11.6.1 Enabling the Ready Control

The READY pin is enabled by setting the bit RDYDIS in EBCMOD0 to '0' in order to activate the corresponding port pin. The polarity of the READY is also defined inside the EBCMOD0 register by the RDYPOL bit.

For a specific address window the READY function is enabled via the RDYEN bit in the FCONCSx register. By programming of FCONCSx.RDYMOD the READY is handled either in synchronous or in asynchronous mode (see also [Figure 11-9](#)).

When the READY function is enabled for a specific address window, each bus cycle within this window must be cleanly terminated with an active READY signal. Otherwise the EBC will be completely blocked by this pending access.



**Figure 11-9 External to internal READY conversion**

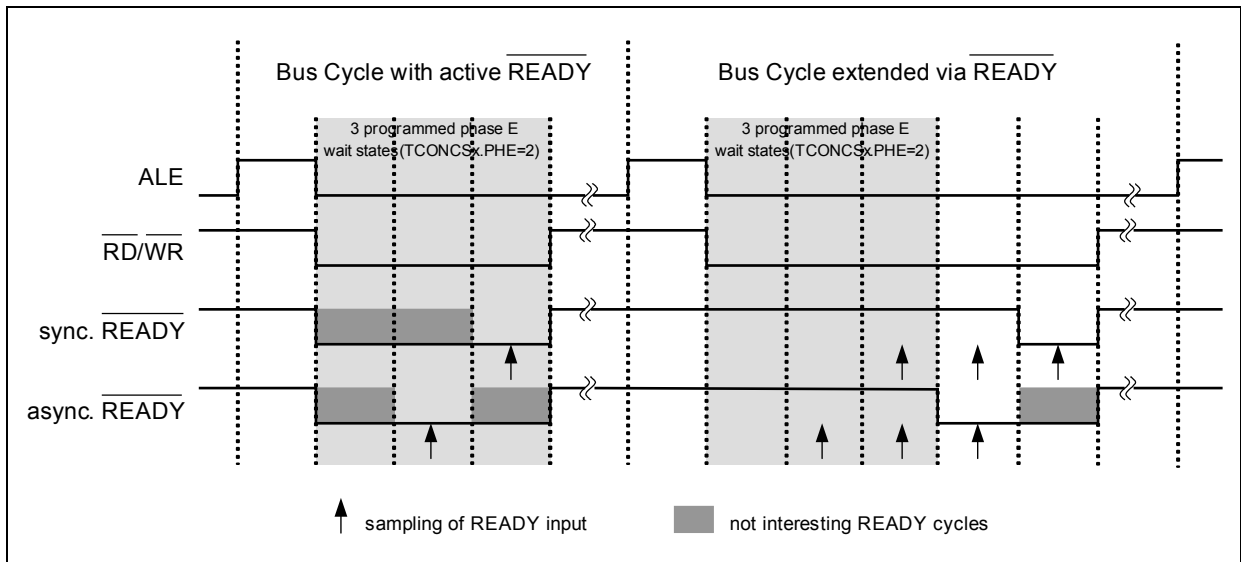
### 11.6.2 Synchronous and Asynchronous READY

The synchronous READY provides the fastest bus cycles, but requires setup and hold times to be met. The CLKOUT signal should be enabled and may be used by the peripheral logic to control the READY timing in this case.

## External Bus Controller (EBC)

The asynchronous READY is less restrictive, but requires one additional wait state caused by the internal synchronization. As the asynchronous READY is sampled earlier programmed wait states may be necessary to provide proper bus cycles

A READY signal (especially asynchronous READY) that has been activated by an external device should be deactivated in response to the trailing (rising) edge of the respective command ( $\overline{\text{RD}}$  or  $\overline{\text{WR}}$ ).



**Figure 11-10 Ready controlled bus cycles**

### 11.6.3 Combining the READY function with predefined wait states

Typically an external wait state or READY control logic takes a while to generate the READY signal when a cycle was started. After a predefined number of clock cycles the XE16xyM will start checking its READY line to determine the end of the bus cycle.

When using the READY function with so-called 'normally-ready' peripherals, it may lead to erroneous bus cycles, if the READY line is sampled too early. These peripherals pull their READY output active, while they are idle. When they are accessed, they drive READY inactive until the bus cycle is complete, then drive it active again. If, however, the peripheral drives READY inactive a little late, after the first sample point of the XE16xyM, the controller samples an active READY and terminates the current bus cycle too early. By inserting predefined wait states the first READY sample point can be shifted to a time, where the peripheral has safely controlled the READY line.

## **11.7 External Bus Arbitration**

The XE16xyM supports multi master systems on the external bus by its external bus arbitration. This bus arbitration allows an external master to request the external bus. The XE16xyM will release the external bus and will float the data and address bus lines and force the control signals via pull ups/downs to their inactive state.

***Attention: This feature is only available if the memory protection unit (MPU) is disabled.***

### **11.7.1 Initialization of Arbitration**

During reset all arbitration pins are tristate, except pin  $\overline{\text{BREQ}}$  which is pulled inactive. After reset the XE16xyM EBC always starts in 'init mode' where the external bus is available but no arbitration is enabled. All arbitration pins are ignored in this state. Other to the external bus connected XE16xyM EBCs assume to have the bus also, so potential bus conflicts are not resolved. For a multi master system the arbitration should be initialized first before starting any bus access. The EBC can either be chosen as arbitration master or as arbitration slave by programming the EBCMOD0 bit SLAVE. The selected mode and the arbitration gets active by the first setting of the HLDEN bit inside the CPUs PSW register. Afterwards a change of the slave/master mode is not possible without resetting the device. Of course for arbitration the dedicated pins have to be activated by setting EBCMOD0.ARBEN.

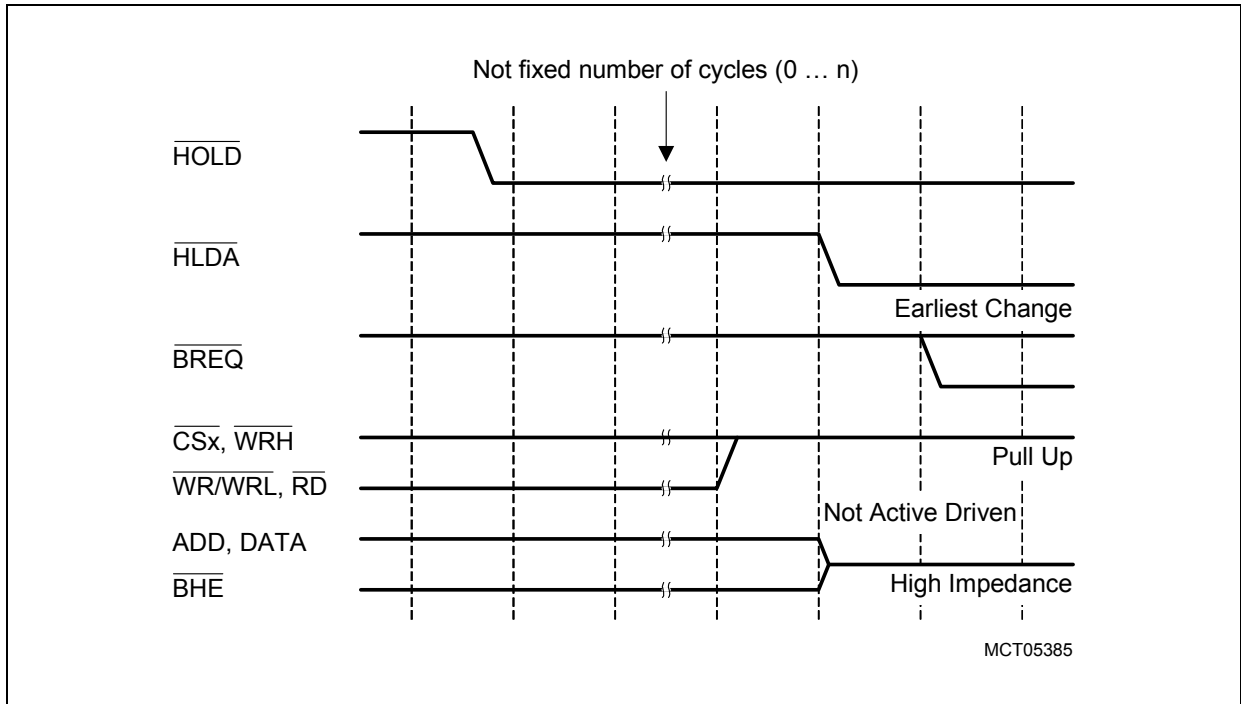
### **11.7.2 Arbitration Master Scheme**

If the XE16xyM EBC is configured as arbitration master, it is default owner of the external bus, controls the arbitration protocol and drives the bus also during idle phases with no bus requests. To perform the arbitration handshake a  $\overline{\text{HOLD}}$  input allows the request of the external bus from the arbitration master. When the arbitration master hands over the bus to the requester this is signaled by driving the hold acknowledge pin HLDA low, which remains at this level until the arbitration slave frees the bus by releasing its request on the  $\overline{\text{HOLD}}$  input. If the arbitration master is not the owner of the bus it treats the external bus interface as follows:

- Address and data bus(es) float to tristate
- Command lines are pulled high by internal pull-up devices ( $\overline{\text{RD}}$ ,  $\overline{\text{WR}}$ / $\overline{\text{WRL}}$ ,  $\overline{\text{BHE}}$ / $\overline{\text{WRH}}$ )
- Address latch control line ALE is pulled low by an internal pull-down device
- CSx outputs are pulled high by internal pull-up devices.

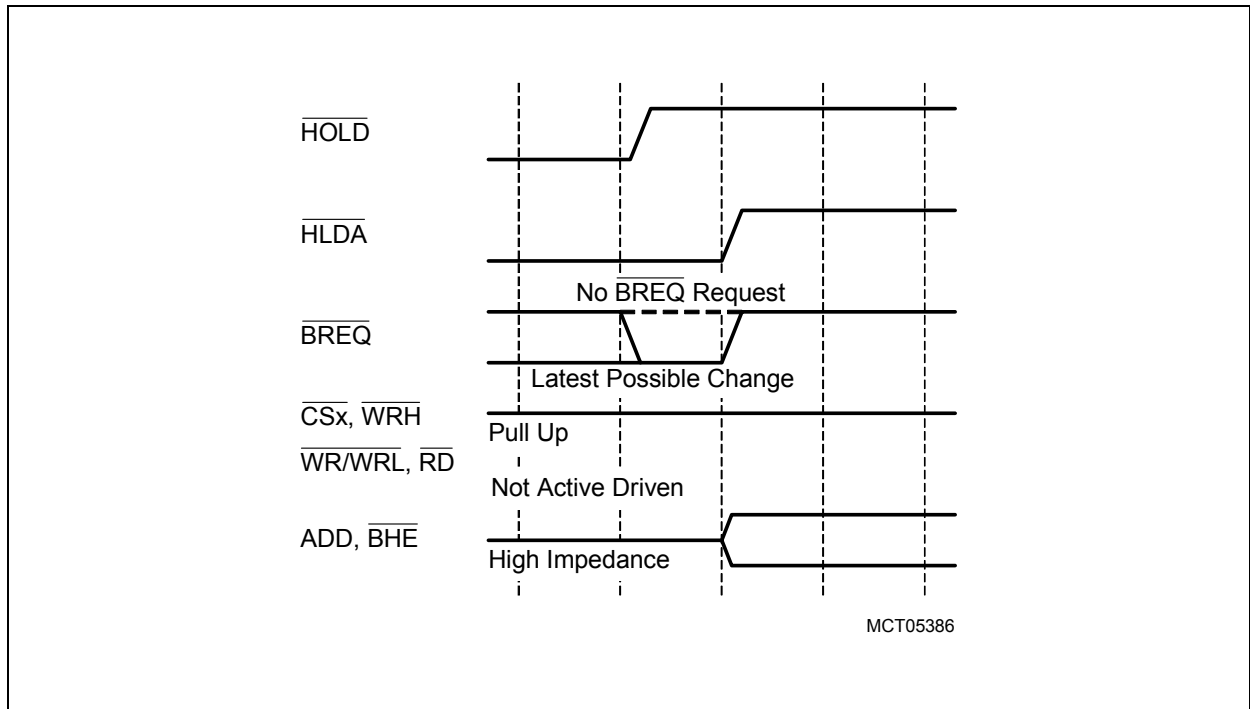
In this state the arbitration slave can take over the bus.

If the arbitration master requires the bus again, it can request the bus via the bus request signal  $\overline{\text{BREQ}}$ . As soon as the arbitration master regains the bus it releases the  $\overline{\text{BREQ}}$  signal and drives HLDA to high.



**Figure 11-11 Releasing the Bus by the Arbitration Master**

*Note: The figure above shows the first possibility for  $\overline{BREQ}$  to get active. The XE16xyM will complete the currently running bus cycle before granting the external bus as indicated by the broken lines.*



**Figure 11-12 Regaining the Bus by the Arbitration Master**

*Note: The falling  $\overline{BREQ}$  edge shows the last chance for  $\overline{BREQ}$  to trigger the indicated regain-sequence. Even if  $\overline{BREQ}$  is activated earlier the regain-sequence is initiated by  $\overline{HOLD}$  going high. Please note that  $\overline{HOLD}$  may also be deactivated without the XE16xyM requesting the bus.*

### 11.7.3 Arbitration Slave Scheme

If the EBC is configured as arbitration slave it is by default not owner of the external bus and has to request the bus first. As long as it has not finished all its queued requests and the arbitration master is not requesting the bus the arbitration slave stays owner of the bus. For the description of the signal handling of the handshake see [Chapter 11.7.2](#). For the arbitration slave the hold acknowledge pin  $\overline{HLDA}$  is configured as input.

### 11.7.4 Bus Lock Function

If an application in a multi master system requires a sequence of undisturbed bus access it has the possibility (independently of being arbitration slave or master) to lock<sup>1)</sup> the bus by setting the PSW bit  $\overline{HLDEN}$  to '0'. In this case the looked EBC will not answer to  $\overline{HOLD}$  requests from other external bus master until  $\overline{HLDEN}$  is set to '1' again. Of course

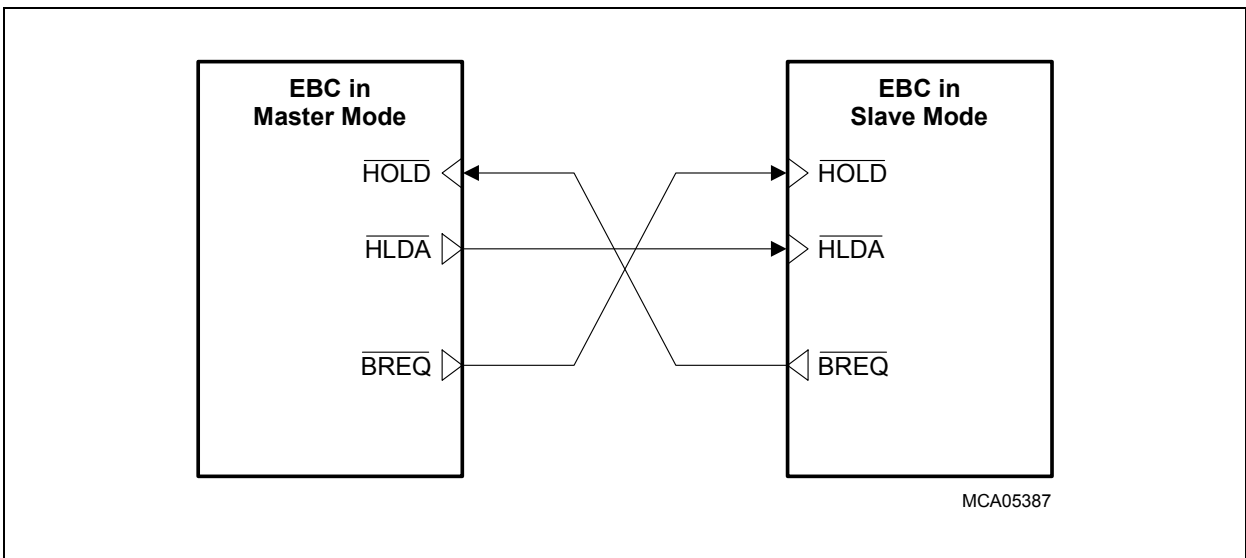
1) It is not allowed to lock the bus by resetting the  $\overline{EBCMOD0.ARBEN}$  bit, as this can lead to bus conflicts.

## External Bus Controller (EBC)

a locked bus master not owning the bus can request the external bus. If a master and a slave are requesting the external bus at the same time for several accesses, they toggle the ownership after each access cycle if the bus is not locked.

### 11.7.5 Direct Master Slave Connection

If one XE16xyM is configured as master and the other as slave and both are working on the same external bus as bus master, they can be connected directly together for bus arbitration as shown in **Figure 11-13**. As both EBCs assume after reset to own the external bus, the 'slave' CPU has to be released from reset and initialized first, before starting the 'master' CPU. The other way is to start both systems at the same time but then both EBC must be configured from internal memory and the PSW.HLDEN bits set before the first external bus request.



**Figure 11-13 Connecting two XE16xyM using Master/Slave Arbitration**

When multiple (more than two) bus masters (XE16xyM or other masters) shall share the same external resources an additional external bus arbiter logic is required that determines the currently active bus master and that controls the necessary signal sequences.

## **11.8 EBC Idle State**

When the external bus interface of EBC is enabled, but no internal or external access is currently executed, the EBC is idle. As long as only on-chip resources such as RAM, peripherals (excluding LXBUS peripherals connected via EBC), registers, etc. are used, the external bus interface does not change (see table 11-2).

The external control signals ( $\overline{RD}$  and  $\overline{WR}$  or  $\overline{WRL}/\overline{WRH}$  if enabled) remain inactive (high) during EBC idle state.

**Table 11-2 Status of the External Bus Outputs During EBC Idle State**

<b>Pins</b>	<b>Status of Pins During EBC Idle</b>
AD15...AD0	tristate (floating)
A15...A0	undefined address (if used for the bus interface)
A23...A16	undefined segment address (on selected pins)
$\overline{CS7}...\overline{CS0}$	inactive (high)
$\overline{BHE}$	level corresponding to last external access
ALE	inactive (low)
$\overline{RD}$ , $\overline{WR}$ , $\overline{WRL}$ , $\overline{WRH}$	inactive (high)

## 11.9 Register Description

The EBC registers are located in the internal IO area. Registers located there use the shorthand XSFR.

**Table 11-3 Registers Address Space**

Module	Base Address	End Address	Note
EBC	00EE00 <sub>H</sub>	00EEFF <sub>H</sub>	

### 11.9.1 EBC Mode Registers

The two mode registers control the XE16xyM EXTBUS pin usage and configuration. Disabled EXTBUS pins may be usable as general purpose IO as described in the "Parallel Ports" chapter.

#### EBCMOD0

##### EBC Mode Register 0

XSFR(00<sub>H</sub>)

Reset value: 5000<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>RDY POL</b>	<b>RDY DIS</b>	<b>ALE DIS</b>	<b>BYT DIS</b>	<b>WR CFG</b>	<b>EBC DIS</b>	<b>SLA VE</b>	<b>ARB EN</b>	<b>CSPEN</b>			<b>SAPEN</b>				
rw	rw	rw	rw	rw	rw	rw	rw	rw			rw				

Field	Bits	Type	Description
<b>RDYPOL</b>	15	rw	<b>READY Pin Polarity</b> 0 <sub>B</sub> READY is active low 1 <sub>B</sub> READY is active high
<b>RDYDIS</b>	14	rw	<b>READY Pin Disable</b> 0 <sub>B</sub> READY enabled 1 <sub>B</sub> READY disabled
<b>ALEDIS</b>	13	rw	<b>ALE Pin Disable</b> 0 <sub>B</sub> ALE enabled 1 <sub>B</sub> ALE disabled
<b>BYTDIS</b>	12	rw	<b>BHE Pin Disable</b> 0 <sub>B</sub> $\overline{\text{BHE}}$ enabled 1 <sub>B</sub> $\overline{\text{BHE}}$ disabled
<b>WRCFG<sup>1)</sup></b>	11	rw	<b>Configuration for Pins <math>\overline{\text{WR}}/\text{WRL}</math>, <math>\overline{\text{BHE}}/\text{WRH}</math></b> 0 <sub>B</sub> $\overline{\text{WR}}$ and $\overline{\text{BHE}}$ 1 <sub>B</sub> $\overline{\text{WRL}}$ and $\overline{\text{WRH}}$



**External Bus Controller (EBC)**

Field	Bits	Type	Description
<b>EBCDIS</b>	10	rw	<b>EBC Pins Disable</b> $0_B$ EBC is using the pins for external bus $1_B$ EBC pins disabled
<b>SLAVE</b>	9	rw	<b>SLAVE Mode Enable</b> $0_B$ Bus arbiter acts in master mode $1_B$ Bus arbiter acts in slave mode
<b>ARBEN</b>	8	rw	<b>BUS Arbitration Pins Enable</b> $0_B$ $\overline{HOLD}$ , $\overline{HLDA}$ and $\overline{BREQ}$ pins are disabled $1_B$ pins act as $\overline{HOLD}$ , $\overline{HLDA}$ and $\overline{BREQ}$
<b>CSPEN</b>	[7:4]	rw	<b>CSx Pins Enable (only external CSx)</b> $0_H$ All external Chip Select pins disabled. $1_H$ $\overline{CS0}$ pin enabled $2_H$ $\overline{CS1}$ and $\overline{CS0}$ pin enabled ... $5_H$ Five $\overline{CSx}$ pins enabled: $\overline{CS4}$ - $\overline{CS0}$ other bit combinations not supported (reserved)
<b>SAPEN</b>	[3:0]	rw	<b>Segment Address Pins Enable</b> $0_H$ All segment address pins disabled $1_H$ One: A[16] enabled ... $8_H$ Eight: A[23:16] enabled other bit combinations not supported (reserved)

1) A change of the bit content is not valid before the next external bus access cycle.

### Byte Write Configurations

For 16-bit data bus configurations the byte write characteristics are programmable by bitfield WRCFG. The following table illustrates the related signals function.

**Table 11-4 Byte Write Configurations**

written byte		WRCFG=0			WRCFG=1		
low	high	$\overline{WR}$	BHE	ADDR[0]	$\overline{WRL}$	$\overline{WRH}$	ADDR[0]
-	-	inactive	don't care	0/1	inactive	inactive	0/1
write	-	active	inactive	0	active	inactive	0/1
-	write	active	active	1	inactive	active	0/1
write	write	active	active	0	active	active	0/1

### EBCMOD1

#### EBC Mode Register 1

**XSFR(02<sub>H</sub>)**

**Reset value: 003F<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0								WRP DIS	DHP DIS	ALP DIS	A0P DIS	APDIS			
r	r	r	r	r	r	r	r	rw	rw	rw	rw	rw			

Field	Bits	Type	Description
<b>0</b>	[15:8]	r	<b>Reserved</b> Read as 0, should be written 0
<b>WRPDIS</b>	7	rw	<b>WR/WRL Pin Disable</b> 0 <sub>B</sub> <u>WR/WRL</u> pin enabled 1 <sub>B</sub> <u>WR/WRL</u> pin disabled
<b>DHPDIS</b>	6	rw	<b>Data High Port Pins Disable</b> 0 <sub>B</sub> Address/Data bus pins 15-8 enabled 1 <sub>B</sub> Address/Data bus pins 15-8 disabled.
<b>ALPDIS</b>	5	rw	<b>Address Low Pins Disable</b> 0 <sub>B</sub> Address bus pins 7-0 generally enabled (depending on APDIS/A0PDIS) 1 <sub>B</sub> Address bus pins 7-0 disabled.
<b>A0PDIS</b>	4	rw	<b>Address Bit 0 Pin Disable</b> 0 <sub>B</sub> Address bus pin 0 enabled 1 <sub>B</sub> Address bus pin 0 disabled.
<b>APDIS</b>	[3:0]	rw	<b>Address Port Pins Disable</b> 0 <sub>H</sub> Address bus pins A15-A1 enabled 1 <sub>H</sub> Pin A15 disabled, A14-A1 enabled 2 <sub>H</sub> Pins A15-A14 disabled, A13-A1 enabled 3 <sub>H</sub> Pins A15-A13 disabled, A12-A1 enabled ... E <sub>H</sub> Pins A15-A2 disabled, A1 enabled F <sub>H</sub> Address bus pins 15-1 disabled.

## 11.9.2 Timing Control Registers

The timing control registers are used to program the cycle timing for the different access phases. The timing control registers may be reprogrammed during code fetches from the affected address window. The new settings become valid for the following access.

**TCONCS0**

Timing Control for CS0

**XSFR(10<sub>H</sub>)**

**Reset value: 7C3D<sub>H</sub>**

**TCONCSx (x=1-4)**

Timing Control for CSx

**XSFR(10<sub>H</sub>+x\*8)**

**Reset value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>0</b>	<b>WRPHF</b>	<b>RDPHF</b>					<b>PHE</b>			<b>PHD</b>	<b>PHC</b>	<b>PHB</b>	<b>PHA</b>		
r	rw	rw					rw			rw	rw	rw	rw		

Field	Bits	Type	Description
<b>0</b>	15	r	<b>Reserved</b> Read as 0, should be written 0
<b>WRPHF</b>	[14:13]	rw	<b>Write Phase F</b> 00 <sub>B</sub> 0 clock cycles ... .. 11 <sub>B</sub> 3 clock cycles
<b>RDPHF</b>	[12:11]	rw	<b>Read Phase F</b> 00 <sub>B</sub> 0 clock cycles ... .. 11 <sub>B</sub> 3 clock cycles
<b>PHE</b>	[10:6]	rw	<b>Phase E</b> 0 <sub>H</sub> 1 clock cycle ... .. 1F <sub>H</sub> 32 clock cycles
<b>PHD</b>	5	rw	<b>Phase D</b> 0 <sub>B</sub> 0 clock cycles 1 <sub>B</sub> 1 clock cycle
<b>PHC</b>	[4:3]	rw	<b>Phase C</b> 00 <sub>B</sub> 0 clock cycles ... .. 11 <sub>B</sub> 3 clock cycles
<b>PHB</b>	2	rw	<b>Phase B</b> 0 <sub>B</sub> 1 clock cycle 1 <sub>B</sub> 2 clock cycles
<b>PHA</b>	[1:0]	rw	<b>Phase A</b> 00 <sub>B</sub> 0 clock cycles ... .. 11 <sub>B</sub> 3 clock cycles

### 11.9.3 Function Control Registers

The Function Control registers are used to control the bus and ready functionality for a selected address window. It can be distinguished between 8 and 16 bit bus and multiplexed and demultiplexed accesses. Furthermore it can be defined whether the address window (and its chip select signal  $\overline{CSx}$ ) is generally enabled or not.

#### **FCONCS0**

**Function Control for CS0**

**XSFR(12<sub>H</sub>)**

**Reset value: 0011<sub>H</sub>**

**FCONCSx (x = 1-4)**

**Function Control for CSx**

**XSFR(12<sub>H</sub>+x\*8)**

**Reset value: 0000<sub>H</sub>**

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	BTYP	0	RDY MOD	RDY EN	EN CS	
r	r	r	r	r	r	r	r	r	r	rw	r	rw	rw	rw	

Field	Bits	Type	Description
0	[15:6]	r	<b>Reserved</b> Read as 0, should be written 0
BTYP	[5:4]	rw	<b>Bus Type Selection</b> 00 <sub>B</sub> 8 bit Demultiplexed 01 <sub>B</sub> 8 bit Multiplexed 10 <sub>B</sub> 16 bit Demultiplexed 11 <sub>B</sub> 16 bit Multiplexed
0	3	r	<b>Reserved</b> Read as 0, should be written 0
RDYMOD	2	rw	<b>Ready Mode</b> 0 <sub>B</sub> asynchronous READY 1 <sub>B</sub> synchronous READY
RDYEN	1	rw	<b>Ready enable</b> 0 <sub>B</sub> access time is controlled by bitfield PHEX 1 <sub>B</sub> access time is controlled by bitfield PHEX and READY signal
ENCS	0	rw	<b>Enable Chip Select</b> 0 <sub>B</sub> disable 1 <sub>B</sub> enable

*Note: The specific ENCSx bits in the FCONCSx registers enable the related address windows and bus functions and the corresponding chip select signal  $\overline{CSx}$ . Additionally it depends on the definition of bitfield CSPEN in register EBCMOD0*

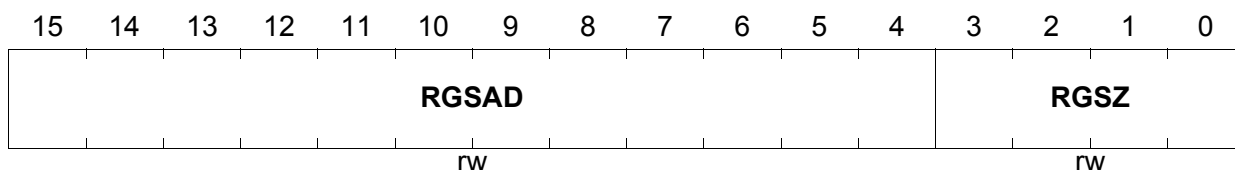
**External Bus Controller (EBC)**

*how many  $\overline{CSx}$  pins are made available for the external system. If an address window is enabled but no external pin is available for the  $\overline{CSx}$ , the bus cycle is executed without assertion of the external chip select signal.*

## 11.9.4 Address Window Selection Registers

**ADDRSELx (x = 1 - 4)**

**Address Range/Size for CSx      XSFR( $16_H + x \cdot 8$ )      Reset value:  $0000_H$**



Field	Bits	Type	Description
<b>RGSAD</b>	[15:4]	rw	<b>Range Start Address</b> Address Range Start Address Selection
<b>RGSZ</b>	[3:0]	rw	<b>Range Size</b> Address Range Size Selection (see <a href="#">Table 11-1</a> )

## **11.10 EBC Implementation in XE16xyM**

The EBC within the XE16xyM automatically affects the behaviour of other device components. In particular with memory mapping and ports allocation the EBC takes a priority role. To understand the particular relations to memory and ports the following chapters are recommended for additional reading:

- Memory Organization
- Parallel Ports

### **11.10.1 Unused Registers**

The XE16xyM external chip select signals are limited due to chip packaging. According to this limitation the corresponding EBC channels are not available. The channels x=(5, 6) with related set of registers ADDRSELx, TCONCSx, FCONCSx are therefore not available for use. For software compatibility reasons the corresponding register address space must not be read or written.

### **11.10.2 Access Control to LXBUS Modules**

In the XE16xyM the EBC channel 7 is reserved for access to chip internal LXBUS peripherals. In general accesses to LXBUS are not visible on the external bus EXTBUS. During LXBUS cycles the EXTBUS remains enabled but is driven to inactive states (control signals) or switched into the read mode (busses).

### **11.10.3 Shutdown Control**

In case of a shutdown request from the SCU the EBC ensures that all the different functions of the EBC are in a non-active state before the whole chip is switched to a power save mode. A running bus cycle is finished, still requested bus cycles are executed. Depending on the master/slave configuration of EBC, the external bus arbiter is controlled for regaining the bus (master) before performing the requested cycles, or the external bus must be released after complete execution of still requested bus cycles. Only when this shutdown sequence is terminated, the shutdown acknowledge is generated from EBC (and from other modules, as described for SCU) and the chip can enter the requested mode.

**Table 11-5** gives an overview of the shutdown control in EBC depending on the EBC configuration.

**Table 11-5 EBC Shutdown Control**

Arbitration Mode	Master Mode		Slave Mode	
	With control of the bus	Without control of the bus	With control of the bus	Without control of the bus
Bus control	Finish all pending cycle requests. Send shutdown acknowledge with the control of the bus.	Ask for the bus. Finish all pending cycle requests. Send shutdown acknowledge with the control of the bus.	Finish all pending requests. Send shutdown acknowledge after leaving the bus	Ask for the bus if needed and finish all requests. Send shutdown acknowledge after leaving the bus

### 11.10.4 Dedicated Registers

The dedicated EBC registers are located in the internal IO area. Registers located there use the shorthand XSFR.

#### 11.10.4.1 Registers dedicated to LXBUS modules

For accesses to MultiCAN and USICs  $\overline{CS7}$  and its control registers ADDRSEL7, TCONCS7 and FCONCS7 are used. The selection of LXBUS is chip internally controlled with  $\overline{CS7}$ .

#### TCONCS7

The LXBUS cycle timing is controlled with register TCONCS7. It uses the shortest possible timing with two clock cycles for one bus cycle. But this minimum timing will be lengthened with waitstate(s) controlled by the modules using the  $\overline{READY}$  function. This timing is reflected by the reset value of TCONCS7.

#### TCONCS7

**Timing Control for CS7**

**XSFR(48<sub>H</sub>)**

**Reset value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	WRPHF	RDPHF	PHE						PHD	PHC	PHB	PHA			
r	r	r					r			r	r	r	r		

Field	Bits	Type	Description
<b>0</b>	15	r	<b>Reserved</b> Read as 0, should be written 0
<b>WRPHF</b>	[14:13]	rw	<b>Write Phase F</b> 00 <sub>B</sub> 0 clock cycles ... .. 11 <sub>B</sub> 3 clock cycles
<b>RDPHF</b>	[12:11]	rw	<b>Read Phase F</b> 00 <sub>B</sub> 0 clock cycles ... .. 11 <sub>B</sub> 3 clock cycles
<b>PHE</b>	[10:6]	rw	<b>Phase E</b> 0 <sub>H</sub> 1 clock cycle ... .. 1F <sub>H</sub> 32 clock cycles
<b>PHD</b>	5	rw	<b>Phase D</b> 0 <sub>B</sub> 0 clock cycles 1 <sub>B</sub> 1 clock cycle
<b>PHC</b>	[4:3]	rw	<b>Phase C</b> 00 <sub>B</sub> 0 clock cycles ... .. 11 <sub>B</sub> 3 clock cycles
<b>PHB</b>	2	rw	<b>Phase B</b> 0 <sub>B</sub> 1 clock cycle 1 <sub>B</sub> 2 clock cycles
<b>PHA</b>	[1:0]	rw	<b>Phase A</b> 00 <sub>B</sub> 0 clock cycles ... .. 11 <sub>B</sub> 3 clock cycles

### **FCONCS7**

The value of this dedicated bus function control register is selected according to the requirements of the internally attached modules: 16-bit demultiplexed bus, access time controlled with synchronous READY.



## FCONCS7

Function Control for CS7

XSFR(4A<sub>H</sub>)

Reset value: 0027<sub>H</sub>

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	BTYP		0	RDY MOD	RDY EN	EN CS
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Field	Bits	Type	Description
<b>0</b>	[15:6]	r	<b>Reserved</b> Read as 0, should be written 0
<b>BTYP</b>	[5:4]	rw	<b>Bus Type Selection</b> 00 <sub>B</sub> 8 bit Demultiplexed 01 <sub>B</sub> 8 bit Multiplexed 10 <sub>B</sub> 16 bit Demultiplexed 11 <sub>B</sub> 16 bit Multiplexed
<b>0</b>	3	r	<b>Reserved</b> Read as 0, should be written 0
<b>RDYMOD</b>	2	rw	<b>Ready Mode</b> 0 <sub>B</sub> asynchronous READY 1 <sub>B</sub> synchronous READY
<b>RDYEN</b>	1	rw	<b>Ready enable</b> 0 <sub>B</sub> access time is controlled by bitfield PHEX 1 <sub>B</sub> access time is controlled by bitfield PHEX and READY signal
<b>ENCS</b>	0	rw	<b>Enable Chip Select</b> 0 <sub>B</sub> disable 1 <sub>B</sub> enable

## Register ADDRSEL7

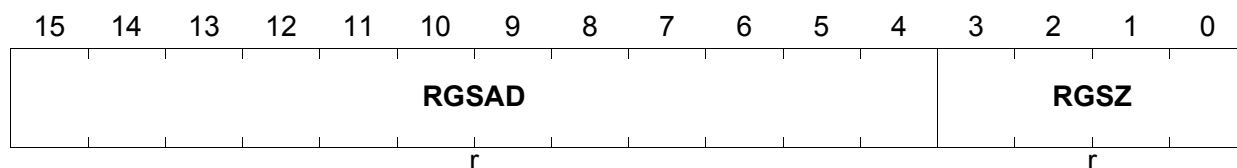
The value of the dedicated address select register allocates the address range 20'0000<sub>H</sub> to 20'FFFF<sub>H</sub> in the external IO area for the attached chip internal modules.

**ADDRSEL7**

**Address Range/Size for CS7**

**XSFR(4E<sub>H</sub>)**

**Reset value: 2004<sub>H</sub>**



Field	Bits	Type	Description
<b>RGSAD</b>	[15:4]	rw	<b>Range Start Address</b> Address Range Start Address Selection
<b>RGSZ</b>	[3:0]	rw	<b>Range Size</b> Address Range Size Selection (see <a href="#">Table 11-1</a> )

## **12 Startup Configuration and Bootstrap Loading**

After startup, the XE16xyM executes code out of an on-chip or off-chip program memory. The initial code source can be selected (refer to the next [Chapter 12.1](#) and [Chapter 12.3](#) to find out how) between the following options:

- **Internal Start** Mode: executes code out of the on-chip program Flash.
- **External Start** Mode: executes code out of an off-chip memory connected to the External Bus Interface.
- **Bootstrap Loading** Modes: execute code out of the on-chip program SRAM (PSRAM). This code is downloaded beforehand via a selectable serial interface.

### **12.1 Startup Mode Selection**

After any startup the currently valid startup configuration is indicated in bitfield HWCFG of register SCU\_STSTAT. The startup configurations and modes supported in XE16xyM are described in [Chapter 12.3](#).

A startup configuration can be selected basically in two ways:

1. Via an externally applied hardware configuration upon a Power-on reset  
 The hardware configuration is applied to the dedicated TRST-pin and to Port 10 pins (P10[6:0]).  
 The hardware that activates a startup configuration during reset may be simple pull resistors for systems that use this feature upon every reset. You may want to use a switchable solution (via jumpers or an external signal) for systems that only temporarily use a hardware configuration.
2. By executing the following software sequence (using SCU\_SWRSTCON and SCU\_RSTCON1 registers):
  - a) Write respective configuration value (refer to [Table 12-2](#) and [Table 12-3](#)) to bitfield SCU\_SWRSTCON.SWCFG;
  - b) Assign desired type of reset to the software request trigger by writing into SCU\_RSTCON1.SW bitfield (by default SCU\_RSTCON1.SW=00<sub>B</sub> meaning no reset generated by software request trigger)
  - c) Set Software Boot Configuration bit: SCU\_SWRSTCON.SWBOOT = 1;
  - d) Trigger a software reset by activating Software Reset Request: SCU\_SWRSTCON.SWRSTREQ = 1.

Additionally, several specific cases of startup configuration handling must be noted:

1. Application reset triggered by hardware request (for example WDT, ESRx) -  
 TRST and P10 pins are not evaluated but the same startup configuration is used as after the previous reset;
2. Application or Internal Application reset triggered by software (with setting SCU\_SWRSTCON.SWRSTREQ=1) -  
 can have different consequences:

## Startup Configuration and Bootstrap Loading

- a) if Software Boot Configuration is selected (SCU\_SWRSTCON.SWBOOT=1) - the startup configuration (SCU\_STSTAT.HWCFG) is updated from SCU\_SWRSTCON.SWCFG bitfield;
- b) otherwise - the same startup configuration is used as after the previous reset.
- 3. Internal Application reset triggered by hardware request - only TRST pin is evaluated (refer to [Page 12-7](#)):
  - a) if TRST=0 - Internal Start from Flash is selected, no debugging is possible;
  - b) otherwise - the same startup configuration is used as after the previous reset.

## 12.2 Device Status after Startup

The main parameters of XE16xyM-status at the point of time when the first user instruction is executed are summarized below.

### 12.2.1 Registers modified by the Startup Procedure

**Table 12-1** shows the XE16xyM registers which are initialized during the startup procedure with values different from their reset-content (defined into respective register-descriptions).

There are two groups of registers regarding the way they are affected by startup procedure:

1. registers initialized after any startup;
2. registers initialized after startup triggered by a power-on in DMP\_1 power domain;

*Note: Power-on in DMP\_M domain means power-on event also in DMP\_1.*

The registers in **Table 12-1** are grouped in accordance to the above differentiation.

Two additional points regarding register-content after startup must be taken into account:

- The register-modifications shown in **Table 12-1** happen independently on the startup mode currently selected, which means also in **Internal Start** mode.  
 Next to these, in other modes - **External Start** and **Bootstrap Loading** (**Chapter 12.5**, **Chapter 12.6**) - more registers are additionally modified during startup, as described into respective Specific Settings chapters for any of the modes.
- The values seen in some bits after startup can be affected not only by the reset procedure itself but also by other events during and even before the last startup - for example an Emergency Event can change the clock-system status.  
 Therefore occasional exceptions are possible from the above values (as well as from the default register content after reset), mainly for some clock control/status flags. For more information on such special cases and their handling - refer to XE16xyM Programmer's Guide.

## Startup Configuration and Bootstrap Loading

**Table 12-1 XE16xyM Registers installed by the Startup Procedure**

Register	Value	Comments
1. After any startup:		
TRAPDIS	019F <sub>H</sub>	All SCU-controlled traps disabled except PET and RAT
RSTCON1	UU: 10uu:U <sub>H</sub>	Internal Application Reset request generated by WDT
IMBCTRL	558C <sub>H</sub>	In External or Bootstrap Loader mode with protected Flash
	A58C <sub>H</sub>	In Internal Start mode or Flash not protected
R8..R15	XXXX	GPRs from Local Bank 1 - used by startup procedure
2. After power-on in DMP_1:		
PLLCON0	0F00 <sub>H</sub>	PLL in Normal Mode, N-divider = 16
PLLCON3	0007 <sub>H</sub>	K2-divider = 8
SYSCON0	0002 <sub>H</sub>	The PLL output (fPLL) used as system clock
WUOSCCON	0000 <sub>H</sub>	Wake up Oscillator enabled with fWU approx. 500kHz
HPOSCCON	U:u0uu: UU <sub>H</sub>	PLLSTAT.FINDIS bit will not be set in an OSCWDT emergency case
PLLOSCCON	XXXX <sub>H</sub>	Device-specific value (chip-to-chip trimming)
EVRMCON0	0100 <sub>H</sub>	EVR_M Control 0 register
EVR1CON0	0D00 <sub>H</sub>	EVR_1 Control 0 register
EVR1SET15VHP	401B <sub>H</sub>	EVR_1 Setting for 1.5V HP register
EVR1SET10V	405B <sub>H</sub>	EVR_1 Setting for 1.0V register
EVR1SET15VLP	40DB <sub>H</sub>	EVR_1 Setting for 1.5V LP register
PVCMCON0	2544 <sub>H</sub>	PVC_M Control for Step 0 register
PVC1CON0	2544 <sub>H</sub>	PVC_1 Control for Step 0 register

### 12.2.2 System Frequency after Startup

The system clock which is active when the first user instruction is executed, depends on the currently selected startup mode and the last startup trigger:

- after power-on in all modes except CAN Bootstrap Loader ([Chapter 12.6.4](#)) - 10MHz (nominal value) from the XE16xyM internal oscillator (doubled frequency);
- after power-on in CAN Bootstrap Loader ([Chapter 12.6.4](#)) - the frequency of an external crystal connected to XTAL-pins, 4MHz minimum;

## **Startup Configuration and Bootstrap Loading**

- after any functional (not power-on) reset - the clock system configuration is not changed by device startup, respectively the system frequency remains as before the reset.

### **12.2.3 Watchdog Timer handling**

The Watchdog Timer (WDT) in XE16xyM is always enabled by the startup procedure and configured to generate Internal Application Reset.

Therefore, the user software must:

- if WDT-usage is foreseen by the code - service it for a first time within approx. 65500 system clock cycles after startup;
- otherwise - disable it within the same time frame as above but before to execute End of Init (EINIT).

The reset requested by WDT serves as response to a device malfunction, due to which malfunction user software can not be anymore executed correctly - respectively the WDT is not regularly served. This reset causes a new device startup followed by user software restart.

The Internal Application Reset - default for WDT - affects not all the modules in XE16xyM - refer to "Module Reset Behavior" in SCU Chapter. The unaffected modules do not change their state, so if this state is "wrong" it will not be recovered to "correct" due to Internal Application Reset, also when triggered by WDT. Therefore, the default WDT configuration and usage can resolve well purely software malfunction but not other failures - for example in clock- or power- system.

One reset-request which puts all the XE16xyM modules into a known - and correctly functioning - initial state is from  $\overline{\text{PORST}}$ -pin. Therefore, if an application requires that correct device restart upon WDT reset is guaranteed, the implementation must be done according to the next [Chapter 12.2.3.1](#).

#### **12.2.3.1 Triggering Power-on Reset by WDT**

This feature requires that either  $\overline{\text{ESR1}}$  or  $\overline{\text{ESR2}}$  pin is dedicated to it, e.g. not available for another functionality.

The following must be done by the user to have power-on reset triggered by WDT:

- in hardware: tie the selected  $\overline{\text{ESRx}}$  ( $x=1,2$ ) pin to  $\overline{\text{PORST}}$  pin of the device;
- in software: at its very beginning, install 1110<sub>B</sub> into bits[3:0] of the respective  $\text{ESRCFGx}$  register ( $x=1,2$ ).

*Note: Keep the WDT reset configuration as installed by startup procedure - Internal Application Reset.*

With the above preparation, any Internal Application Reset - including triggered by WDT - will drive the selected  $\overline{\text{ESRx}}$  pin low so leading to power-on and a next device restart.

## Startup Configuration and Bootstrap Loading

***Attention: When using this solution, Internal Application reset is no more available as separate reset type.***

***Respectively upon this reset all the device resources will be initialized as upon power-on and any previous information will be lost.***

***Therefore, when controlling this feature by WDT do not assign Internal Application Reset to any other trigger if prevention of previous information/status is needed.***

For additional information on this feature - refer to Application Note AP16146 .

### 12.2.4 Startup Error state

To prevent possible negative consequences for the device and/or the system, upon unrecoverable error during startup XE16xyM is put onto a stable, passive and neutral to the external world state - power-save mode with DMP\_1 shut down and DMP\_M powered with 1V.

This state can be exited with power-on reset only.

## **12.3 Supported Startup Modes and Options**

XE16xyM supports variety of startup modes, allowing the user to make selections in three aspects:

- main functionality - where from the user code will be started (on-chip Flash, PSRAM, external memory);
- optionally - a way for initial code-downloading into PSRAM before to start it:
  - from an external host via a communication interface - UART, CAN, SSC;
- debug-related - either debugging will be possible, and if Yes - which debug-interface to use (JTAG, DAP, selectable pin-assignments).

Following from the above differentiation, the startup modes in XE16xyM are divided into several groups, described in [Chapter 12.3.1](#), [Chapter 12.3.2](#) and [Chapter 12.3.3](#).



## Startup Configuration and Bootstrap Loading

### 12.3.1 Basic Startup Modes

These modes (refer to [Table 12-2](#)) have no debug support and no special features.

**Table 12-2 Basic XE16xyM Startup Modes**

Startup Mode	STSTAT.HWCFG Value <sup>1)</sup>	Configuration pins <sup>2)</sup>							
		TRST	P10 [6 : 0]						
Internal Start from Flash	00 <sub>H</sub>	0	x	x	x	x	x	x	x
UART Bootloader 2.x <sup>3)</sup>	02 <sub>H</sub>	1	x	x	x	x	0	1	0
UART Bootloader 7.x <sup>4)</sup>	06 <sub>H</sub>	1	x	x	x	x	1	1	0
SSC Bootloader	09 <sub>H</sub>	1	x	x	x	1	0	0	1
CAN Bootloader	0D <sub>H</sub>	1	x	x	x	1	1	0	1
UART Enhanced Bootloader 2.x <sup>3)</sup>	10 <sub>H</sub>	1	0	0	1	0	0	0	0
External Start <sup>5)</sup>	70 <sub>H</sub>	1	1	1	1	0	0	0	0

1) Bitfield HWCFG can be loaded from Port 10 or from bitfield SWCFG in register SWRSTCON.

2) x means that the level on the corresponding pin is irrelevant.

3) 2.x means: TxD (transmit data) at P2.3 pin, RxD (receive data) at P2.4 pin.

4) 7.x means: TxD (transmit data) at P7.3 pin, RxD (receive data) at P7.4 pin. **Not available in 64-pin package.**

5) Not available in 64-pin package.

The XE16xyM functionality in different modes - Internal start, External start, Bootstrap loading - is described further in [Chapter 12.4](#), [Chapter 12.5](#), [Chapter 12.6](#) respectively.

### 0-pin Configuration

This is a new feature for XE16xyM, meaning usage of no General-purpose Input-output (GPIO) pins - including no assignment of any alternate function to a pin - to select the startup configuration which is supposed to the most used one - Internal Start from Flash.

One dedicated pin -  $\overline{\text{TRST}}$  - is used as a checkpoint, as follows:

- $\overline{\text{TRST}}=0$  during reset - no other pins (also from Port 10) are evaluated, the user code is started from Internal Flash memory - refer to the first row in [Table 12-2](#);
- $\overline{\text{TRST}}=1$  during reset - some of the Port 10 pins are evaluated to determine the further device-functioning. The number of P10 pins used for that purpose varies from 3 up to 7- refer to [Table 12-2](#) and [Table 12-3](#).

### 12.3.2 Startup Modes with Debug Support

These startup selections (refer to [Table 12-3](#)) lead to user-code start either from Internal Flash or from External memory. So from functional point of view they are similar to two from the [Basic Startup Modes](#) in [Table 12-2](#), but additionally these selections allow an external tool (debugger) to be connected and used during the development process.

**Table 12-3 XE16xyM Startup Mode Configurations with debug support**

Startup Mode	Debug Interface	STSTAT.HWCFG Value <sup>1)</sup>	CFG pins P10 [6:0] <sup>2)</sup>							
			TRST=1							
Internal Start from Flash	JTAG pos.B	03 <sub>H</sub>	x	x	x	x	0	1	1	
	DAP pos.1	04 <sub>H</sub>	x	x	x	x	1	0	0	
	from Flash <sup>3)</sup>	07 <sub>H</sub>	x	x	x	x	1	1	1	
	DAP pos.0	01 <sub>H</sub>	x	x	x	0	0	0	1	
	DAP pos.2 <sup>4)</sup>	05 <sub>H</sub>	x	x	x	0	1	0	1	
	JTAG pos.C <sup>4)</sup>	40 <sub>H</sub>	1	0	0	0	0	0	0	
	JTAG pos.D <sup>5)</sup>	50 <sub>H</sub>	1	0	1	0	0	0	0	
External Start <sup>4)</sup>	from Flash <sup>3)</sup>	60 <sub>H</sub>	1	1	0	0	0	0	0	

1) Bitfield HWCFG can be loaded from Port 10 or from bitfield SWCFG in register SWRSTCON.

2) x means that the level on the corresponding pin is irrelevant.

3) A defined location in Flash (C0'01F0<sub>H</sub>) must contain a value (2 Bytes) for DBGPRR register and the next word-location (C0'01F2<sub>H</sub>) must contain the inverse value.

From the 16-bit value in Flash the four most-significant bits are don't care - they are handled by the startup procedure itself.

If the inverse-condition does not match - the value is considered as invalid and JTAG pins at position A are configured by default.

4) Not available in 64-pin package.

5) Not available in 64 and 100-pin package.

The variety of startup configurations ([Table 12-3](#)) in this case serve to select the type (DAP or JTAG) and pin-location of the debug interface which will be enabled. Besides of this selection - done either via P10-pins or by SWRSTCON.HWCFG-bitfield - two basic conditions exist for debug-interface handling:

- debug interface configuration is a part of the complete device startup configuration. Therefore debug interface enabling/disabling/(re)configuration takes place only when the startup configuration (e.g. the startup mode selection) is updated in some of the ways described in [Chapter 12.1](#).  
For example, upon an application reset triggered by hardware source (let say WDT) the debug interface will not be touched.
- debug interface will be always disabled, if Internal Start is selected and the on-chip Flash is protected

## Startup Configuration and Bootstrap Loading

The exact meaning - interface types and pin-assignments - of different debug interface selections is shown in [Table 12-4](#). The last column of this table shows the value, which must be written into Flash location C0 01F0<sub>H</sub>, if Debug Interface Configuration from Flash has been selected as startup option (refer to [Table 12-3](#)).

**Table 12-4 Debug-selections in XE16xyM: interface types and pin assignments**

Debug Interface		Pins used for Debugging:		DBGPRR value in Flash addr. C0 01F0 <sub>H</sub>
Type	Position	main interface (obligatory)	BRKIN (optional)	
DAP	pos.0	P2.9, P7.0	P5.10	1000 <sub>H</sub>
	pos.1	P10.9, P10.12	P10.8	1155 <sub>H</sub>
	pos.2 <sup>1)</sup>	P7.0, P7.4	P7.1	12AA <sub>H</sub>
JTAG	pos.A	P2.9, P5.2, P5.4, P7.0	P5.10	1800 <sub>H</sub>
	pos.B	P10.9, P10.10, P10.11, P10.12	P10.8	1955 <sub>H</sub>
	pos.C <sup>1)</sup>	P7.0, P7.2, P7.3, P7.4	P7.1	1AAA <sub>H</sub>
	pos.D <sup>2)</sup>	P8.3, P8.4, P8.5, P10.12	P8.6	1BFF <sub>H</sub>
not available	---	---	---	0000 <sub>H</sub>

1) Not available in 64-pin package.

2)

There are two types of interface signals/pins related to debugging:

### Main Debug Interface

These are 2 (in case of DAP) or 4 (in case of JTAG) pins listed in the third column of [Table 12-4](#).

If debugging is enabled, these pins are always assigned to the debug-interface, therefore the application software must never use any of them.

### Optional Break Interface

The Break Interface of XE16xyM Debug System includes two signals: BRKIN and BRKOUT.

The usage of this interface is optional, also selectively either only one out of the two signals or both of them can be utilized.

The Break Interface usage requires additional preparation which will be done - when requested - by the external debugger once the main interface is available.

## **Startup Configuration and Bootstrap Loading**

As long as this preparation and the activation of “Break-In”/”Break-Out” feature has not happened, the respective pin(s) selected to host (potentially) BRKIN/BRKOUT-signal(s) can be still used for other functionality by the application software.

The two Break-signals/pins are handled some differently to each other:

- BRKIN - the exact pin which will be used (in case) for this purpose is determined uniquely by the startup selection - refer to the BRKIN-column in [Table 12-4](#).
- BRKOUT - no pin-selection for this signal is done during startup.  
Few pins are potentially available for BRKOUT-selection - done by the external tool before to activate the “Break-Out” feature:
  - P6.0;
  - P1.5 - not available in 64-pin package;
  - P9.3 - not available in 64- and 100-pin packages;
  - P10.11 - can not be used, if JTAG interface at position B is selected.

### **12.3.3 Special Startup Features**

XE16xyM supports some special features, which allow the user software to influence the device startup, providing additional functionality next to the above (in [Chapter 12.3.1](#) and [Chapter 12.3.2](#)) described.

***Attention: The correct usage of these features requires good and detailed understanding of the XE16xyM structure, behavior and programming. The special startup features are dedicated to advanced users, being familiar with device as a whole and especially with the System Control Unit - both as hardware (described in SCU Chapter of this User Manual) and how to control it properly by software (described in the XE16xyM Programmer's Guide).***

#### **12.3.3.1 Supplementary Startup Information from/to the User**

The special startup features require/provide additional information from/to the application software, using a dedicated register inside the System Control Unit - STMEM0.

##### **STMEM0 Register**

The SCU\_STMEM0 register is located in DMP\_M power-supply domain and is Security-protected.

The following startup information can be exchanged with application software using this register:

- the user software can influence the next device startup by writing into STMEM0 bits[15:6]  
 The supported features are described in [Chapter 12.3.3.2](#).
- if STMEM0[4]=0 after startup - this startup has been triggered by a Functional (i.e. Application or Internal Application) Reset;  
 In such a case the emergency-status flags indicated in SCU\_SYSCON0 bits[15:12] upon device startup can be read by user software from SCU\_STMEM0 bits[3:0] as follows:
  - bit[0] - OSCWDT Emergency Event Source status
  - bit[1] - VCOLCK Emergency Event Source status
  - bit[2] - PVC1 Emergency Event Source status
  - bit[3] - Clock Select status
- if STMEM0[4]=1 after startup - this startup has been triggered by a Power-On;  
 In such a case, additional information is provided by SCU\_STMEM0[3] bit:
  - STMEM0[3]=0 - Power-On in DMP\_1 domain only
  - STMEM0[3]=1 - Power-On in both DMP\_1 and DMP\_M domains

## Startup Configuration and Bootstrap Loading

### STMEM0

**Startup Memory 0 Register**

**ESFR (F0A0<sub>H</sub>/50<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
USSET	0	RINDP	RINDS	RINPS	0	0	0	0	0	0	0	0	0	0	0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Typ	Description
<b>STSIND</b>	[4:0]	rw	<b>Startup Status Indication to the user:</b> <i>Note: The values not described here are reserved.</i> 0xxxx <sub>B</sub> Functional reset - bits[3:0] show status flags (refer to the text description) 10000 <sub>B</sub> Power-on in DMP_1 domain only 11000 <sub>B</sub> Power-on in DMP_M and DMP_1 domains
<b>0</b>	5	rw	<b>Reserved</b> , must be written with reset value 0
<b>0</b>	6,[10:7]	rw	<b>Reserved</b> , must be written with reset value 0
<b>RINPS</b>	11	rw	<b>Initialization of the PSRAM:</b> 0 <sub>B</sub> not requested 1 <sub>B</sub> will be performed upon startup
<b>RINDS</b>	12	rw	<b>Initialization of the DSRAM:</b> 0 <sub>B</sub> not requested 1 <sub>B</sub> will be performed upon startup
<b>RINDP</b>	13	rw	<b>Initialization of the DPRAM:</b> 0 <sub>B</sub> not requested 1 <sub>B</sub> will be performed upon startup
<b>0</b>	14	rw	<b>Reserved</b> , must be written with reset value 0
<b>USSET</b>	15	rw	<b>RAM Initialization upon startup:</b> 0 <sub>B</sub> not requested 1 <sub>B</sub> requested in STMEM0 [13:11]

### 12.3.3.2 Preparing to activate Memory Content Protection

XE16xyM supports two mechanisms for Memory Content protection: ECC (Error Correction Code) and Parity, both are disabled by default.

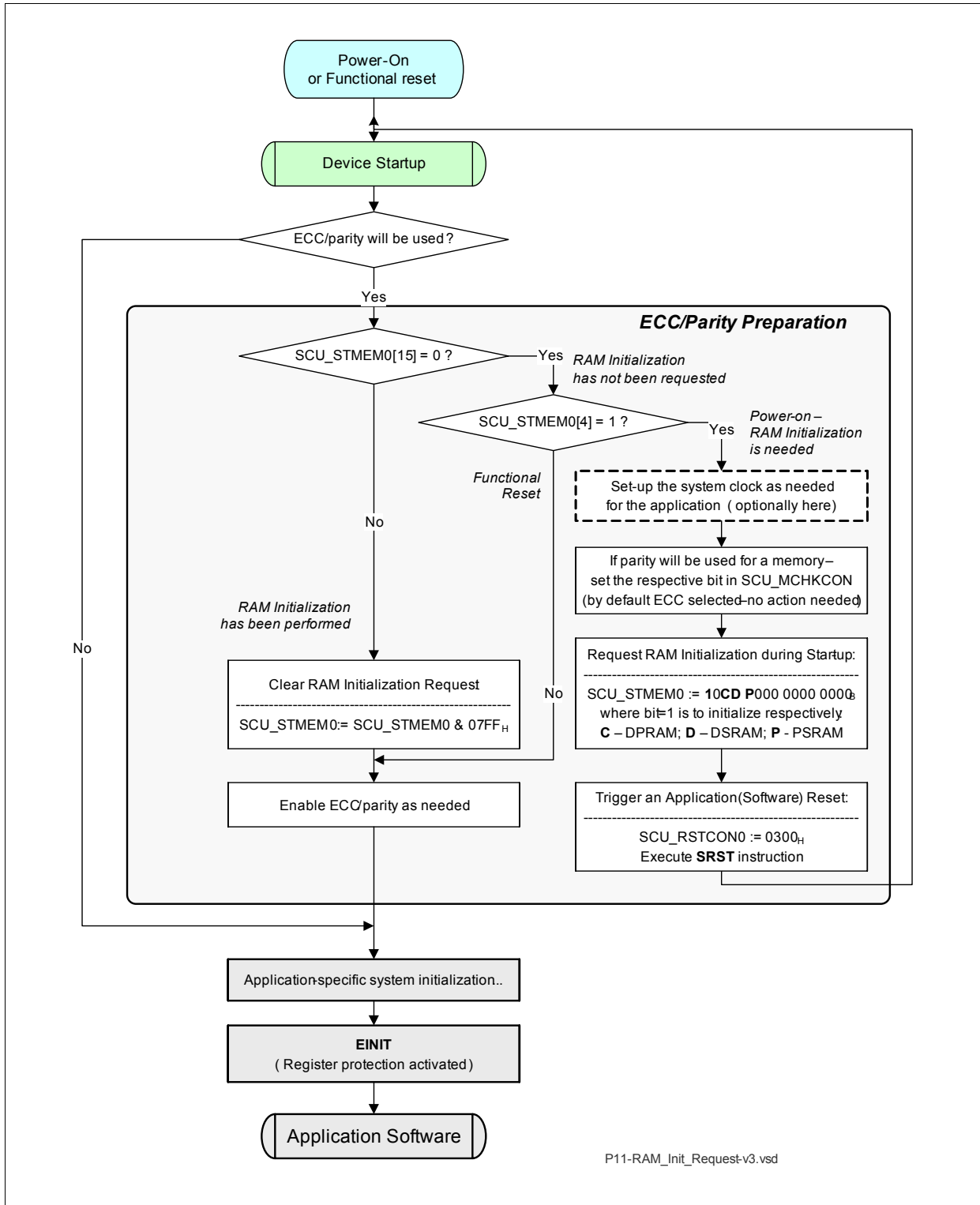
Any of these mechanisms can be only activated by the user, using the sequence shown at [Figure 12-1](#). The processing according to this sequence includes:

## **Startup Configuration and Bootstrap Loading**

- upon power-on of the device (indicated by **STMEM0**[4]=1, **STMEM0**[15] will be 0 in this case ) - RAM initialization is needed:
  - optionally - if the application will run with system clock faster than 10MHz (system frequency after power-on) - the clock reconfiguration can be done still here to use increased speed for a faster RAM initialization;
  - if parity will be used for some memory - write 1 into the respective bit(s) of SCU\_MCHKCON register (by default ECC is the selected protection type);
  - install request for RAMs initialization by setting **STMEM0**[15:11]=10111<sub>B</sub>;  
 It is also possible to set selectively only some of the bits[13:11] corresponding to the memories in which the Content Protection will be activated (refer to **STMEM0**-description and **Figure 12-1**). This will not bring too much - in sense of a faster startup - because all the memories are initialized in parallel and the time-variation if processing one only or all the RAMs will be not so big.
  - trigger an application reset to cause a new device startup  
 During this new device startup the RAMs are initialized as requested in **STMEM0**[13:11].
- if **STMEM0**[15]=1 after startup - meaning RAMs have been just initialized:
  - RAM-initialization request is cleared - **STMEM0**[15:11]=00000<sub>B</sub>;
  - ECC/parity is configured/enabled as required by the application
  - continue with further system initialization (if any) and starting the application
- if **STMEM0**[15]=0 after functional reset (not power-on) - RAM initialization is not needed and the request is not active:
  - ECC/parity is configured/enabled as required by the application - this is needed because some control registers are reset upon any startup;
  - continue with further system initialization (if any) and starting the application.

The read-operations from initialized memories produce no errors, the data delivered is:

- if ECC is active - 0600<sub>H</sub>;
- if parity is active - 3000<sub>H</sub>.



**Figure 12-1 Software sequence to prepare ECC/Parity usage**



## **12.4 Internal Start**

When internal start mode is configured, the XE16xyM immediately begins executing code out of the on-chip Flash memory (first instruction from location C0'0000<sub>H</sub>).

Because internal start mode without debug-support is expected to be the configuration used in most cases, this mode can be selected by pulling low the dedicated (e.g. not available for application-purposes)  $\overline{\text{TRST}}$ -pin only - so-called **0-pin Configuration**.

If debug-support is needed - additional configuration options are available, refer to **Chapter 12.3.2**.

*Note: A read-protected Flash is readable for applications started in Internal mode without disabling the protection.*

## **12.5 External Start**

When external start mode is configured, the XE16xyM begins executing code out of an off-chip memory (first instruction from location 00'0000<sub>H</sub>), connected to the XE16xyM's external bus interface.

**Attention: External Startup mode is not supported in 64-pin package devices.**

The External Bus Controller is adjusted to the employed external memory by evaluating additional configuration pins.

Seven pins of P10 are used to select the EBC mode (P10.[10:8]), the address width (P10.[12:11]), and the number of chip select lines (P10.[14:13]). The following tables summarize the available options.

**Startup Configuration and Bootstrap Loading**

**Table 12-5 EBC Configuration: EBC Mode**

<b>EBC Startup Mode</b>	<b>Cfg. Pins P10[10:8]</b>			<b>Pins Used by the EBC</b>
8-Bit Data, Multiplexed	0	0	0	P2.0 ... P2.2, P10.0 ... P10.15
8-Bit Data, Demultiplexed	0	0	1	P0.0 ... P0.7, P1.0 ... P1.7, P2.0 ... P2.2, P10.0 ... P10.7, P10.13, P10.14
16-Bit Data, MUX, $\overline{\text{BHE}}$ mode	0	1	0	P2.0 ... P2.2, P2.11, P10.0 ... P10.15
16-Bit Data, MUX, $\overline{\text{WRH}}$ mode	0	1	1	P2.0 ... P2.2, P2.11, P10.0 ... P10.15
16-Bit Data, DeMUX, $\overline{\text{BHE}}$ mode, A0	1	0	0	P0.0 ... P0.7, P1.0 ... P1.7, P2.0 ... P2.2, P2.11, P10.0 ... P10.14
16-Bit Data, DeMUX, $\overline{\text{WRH}}$ mode, A0	1	0	1	P0.0 ... P0.7, P1.0 ... P1.7, P2.0 ... P2.2, P2.11, P10.0 ... P10.14
16-Bit Data, DeMUX, $\overline{\text{BHE}}$ mode, A1	1	1	0	P2.0 ... P2.2, P10.0 ... P10.15
16-Bit Data, DeMUX, $\overline{\text{WRH}}$ mode, A1	1	1	1	P0.0 ... P0.7, P1.0 ... P1.7, P2.0 ... P2.2, P10.0 ... P10.7, P10.13, P10.14

**Table 12-6 EBC Configuration: Address Width**

<b>Available Address Lines</b>	<b>Cfg. Pins P10[12:11]</b>		<b>Additional Address Pins</b>
A15 ... A0	0	0	None
A17 ... A0	0	1	P2.3, P2.4
A19 ... A0	1	0	P2.3 ... P2.6
A23 ... A0	1	1	P2.3 ... P2.10

**Table 12-7 EBC Configuration: Chip Select Lines**

<b>Available Chip Select Lines</b>	<b>Cfg. Pins P10[14:13]</b>		<b>Used Pins</b>
$\overline{\text{CS0}} \dots \overline{\text{CS4}}$	0	0	P4.0 ... P4.4
CS0	0	1	P4.0
CS0 ... CS1	1	0	P4.0, P4.1
None	1	1	None

### 12.5.1 Specific Settings

When the XE16xyM has entered External Start mode, the configuration is automatically set according to [Table 12-8](#) and [Table 12-9](#).

Note, that the startup procedure does not configure any address window within ADDRSELx registers. Therefore, even if some CS signal is configured (refer to [Table 12-7](#)), the startup procedure only makes the proper settings to assure the adequate pin-functionality in regard to the selected EBC mode. The user software must take care:

- to configure the address window (in ADDRSELx register) for the  $\overline{\text{CSx}}$  pin(s) which will be used;
- to enable those pins by setting FCONCSx.ENCs.

**Table 12-8 External start mode-Specific State in EBC Registers**

Configuration at P10[10:8]	EBCMOD0 [15:8]	EBCMOD1	FCONCSx <sup>1)</sup>	Comment (EBC Mode)
000 <sub>B</sub>	30 <sub>H</sub>	001F <sub>H</sub>	0011 <sub>H</sub>	8-Bit Multiplexed
001 <sub>B</sub>	70 <sub>H</sub>	0020 <sub>H</sub>	0001 <sub>H</sub>	8-Bit Demultiplexed
010 <sub>B</sub>	40 <sub>H</sub>	0000 <sub>H</sub>	0031 <sub>H</sub>	16-Bit MUX, $\overline{\text{BHE}}$
011 <sub>B</sub>	48 <sub>H</sub>	0000 <sub>H</sub>	0031 <sub>H</sub>	16-Bit MUX, $\overline{\text{WRH}}$
100 <sub>B</sub>	60 <sub>H</sub>	0000 <sub>H</sub>	0021 <sub>H</sub>	16-Bit DeMUX, $\overline{\text{BHE}}$ , A0
101 <sub>B</sub>	61 <sub>H</sub>	0000 <sub>H</sub>	0021 <sub>H</sub>	16-Bit DeMUX, $\overline{\text{WRH}}$ , A0
110 <sub>B</sub>	60 <sub>H</sub>	0010 <sub>H</sub>	0021 <sub>H</sub>	16-Bit DeMUX, $\overline{\text{BHE}}$ , A1
111 <sub>B</sub>	61 <sub>H</sub>	0010 <sub>H</sub>	0021 <sub>H</sub>	16-Bit DeMUX, $\overline{\text{WRH}}$ , A1

1) Which FCONCSx registers are affected is dependant on the configuration at P10[14:13] as follows:

11<sub>B</sub> or 01<sub>B</sub> - FCONCS0 is affected

10<sub>B</sub> - FCONCS0 and FCONCS1 are affected

00<sub>B</sub> - FCONCS0..FCONCS4 are affected

The other (unaffected) FCONCS registers retain their default values - refer to the EBC Chapter of this Manual.

**Table 12-9 External start mode-Specific State in EBCMOD0[7:0]**

Configuration at P10[14:13]	Configuration at P10[12:11]			
	00B (0 Segm.)	01B (2 Segm.)	10 <sub>B</sub> (4 Segm.)	11 <sub>B</sub> (8 Segm.)
00 <sub>B</sub> (5 CS)	50 <sub>H</sub>	52 <sub>H</sub>	54 <sub>H</sub>	58 <sub>H</sub>
01 <sub>B</sub> (1 CS)	10 <sub>H</sub>	12 <sub>H</sub>	14 <sub>H</sub>	18 <sub>H</sub>
10 <sub>B</sub> (2 CS)	20 <sub>H</sub>	22 <sub>H</sub>	24 <sub>H</sub>	28 <sub>H</sub>
11 <sub>B</sub> (0 CS)	00 <sub>H</sub>	02 <sub>H</sub>	04 <sub>H</sub>	08 <sub>H</sub>

## **12.6 Bootstrap Loading**

Bootstrap Loading is the technique of transferring code to the XE16xyM via a certain interface (usually serial) before the regular code execution out of non-volatile program memory commences. Instead, the XE16xyM executes the previously received code.

This boot-code may be complete (e.g. temporary software for testing or calibration), amend existing code in non-volatile program memory (e.g. with product-specific data or routines), or load additional code (e.g. using higher or more secure protocols). A possible application for bootstrap loading is the programming of virgin Flash memory at the end of a production line, with no external memory or internal Flash required for the initialization code.

The BSL mechanism may be used for standard system startup as well as only for special occasions like system maintenance (firmware update) or end-of-line programming or testing.

The XE16xyM supports bootstrap loading using several protocols/modes:

- Standard UART protocol, loading 32 bytes (see [Section 12.6.2.1](#))
- UART protocol, Enhanced bootstrap loader transferring arbitrary number of bytes (see [Section 12.6.2.2](#))
- Synchronous serial protocol (see [Section 12.6.3](#))
- CAN protocol (see [Section 12.6.4](#))

For a summary of these modes, see also [Table 12-16](#).

### **12.6.1 General Functionality**

Even though each bootstrap loader has its particular functionality, the general handling is the same for all of them.

#### **Entering a Bootstrap Loader**

Bootstrap loaders are enabled by selecting a specific startup configuration (see [Section 12.1](#)).

The required configuration patterns are described in [Table 12-16](#) for the bootstrap loaders, and are summarized in [Table 12-2](#).

### **Loading the Startup Code**

After establishing communication, the BSL enters a loop to receive the respective number of bytes. These bytes are stored sequentially into the on-chip PSRAM, starting at location E0'0000<sub>H</sub>. To execute the loaded code the BSL then points register VECSEG to location E0'0000<sub>H</sub>, i.e. the first loaded instruction, and then jumps to this instruction.

The loaded code may be the final application code or another, more sophisticated, loader routine that adds a transmission protocol to enhance the integrity of the loaded code or data. It may also contain a code sequence to change the system configuration and enable the bus interface to store the received data into external memory.

This process may go through several iterations or may directly execute the final application.

### **Exiting Bootstrap Loader Mode**

The watchdog timer and the debug system are disabled as long as the Bootstrap loader is active. Watchdog timer and debug system are released automatically when the BSL terminates after having received the last byte from the host.

If 2<sup>nd</sup> level loaders are used, the loader routine should deactivate the watchdog timer via instruction DISWDT to allow for an extended download period.

The XE16xyM will start executing out of user memory as externally configured after a non-BSL reset .

### **Interface to the Host**

The bootstrap loader communicates with the external host over a predefined set of interface pins. These interface pins are automatically enabled and controlled by the bootstrap loader. The host must connect to these predefined interface pins.

**Table 12-16** indicates the interface pins that are used in each bootstrap loader mode.

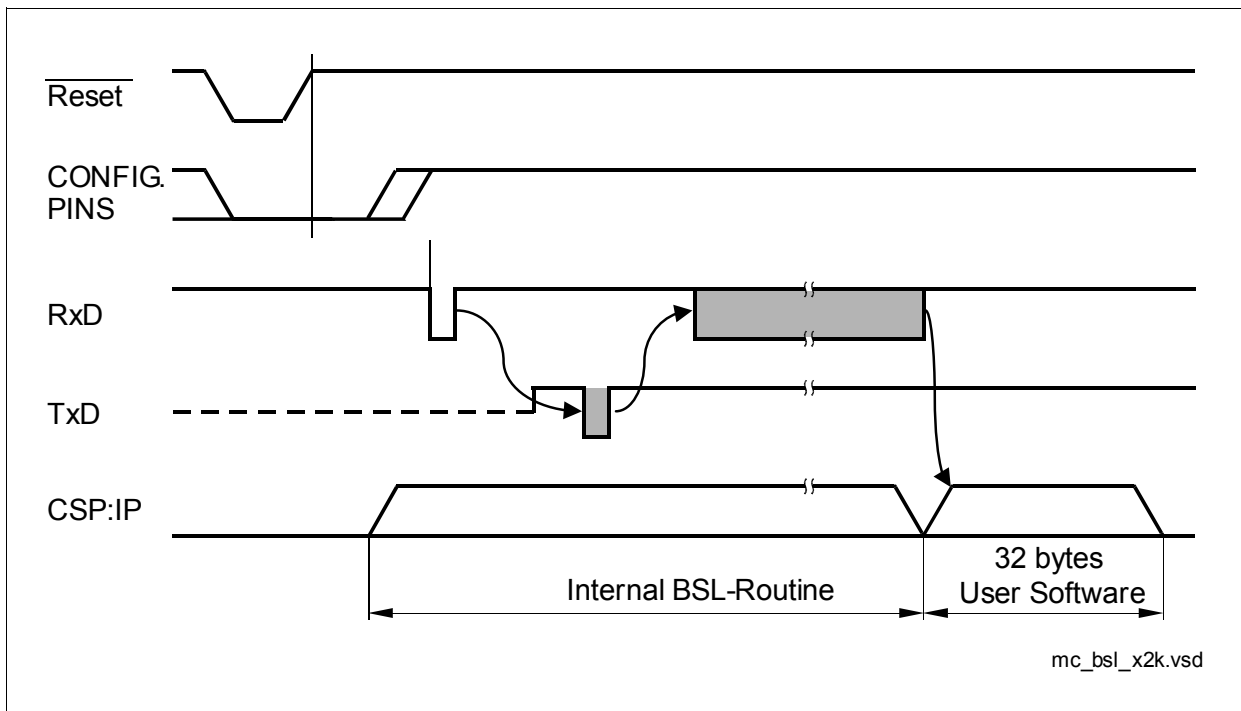
## 12.6.2 Bootstrap Loaders using UART Protocol

XE16xyM users have different possibilities to download code/data in which the communication is based on UART (Universal Asynchronous Receiver and Transmitter) protocol.

### 12.6.2.1 Standard UART Bootstrap Loader

The standard UART bootstrap loader transfers program code/data via channel 0 of USIC0 (U0C0) into the PSRAM. The U0C0 receiver is only enabled after the identification byte has been transmitted. A half duplex connection to the host is, therefore, sufficient to feed the BSL.

Data is transferred from the external host to the XE16xyM using asynchronous eight-bit data frames without parity (1 start bit, 1 stop bit). The number of data bytes to be received in standard UART boot mode is fixed to 32 bytes, which allows up to 16 two-byte instructions.



**Figure 12-2 Bootstrap Loader Sequence**

The XE16xyM scans the RxD line to receive a zero byte after entering UART BSL mode and the respective initialization. The zero byte is considered as containing one start bit, eight 0 data bits and one stop bit. From the duration of this zero byte it calculates the corresponding baudrate factor with respect to the current CPU clock, initializes the serial interface U0C0 accordingly and switches pin TxD to output. Using this baudrate, an identification byte (D5<sub>H</sub>) is returned to the host that provides the loaded data.

## Startup Configuration and Bootstrap Loading

Once the identification byte is transmitted, the BSL enters a loop to receive 32 bytes via U0C0. These bytes are stored sequentially into locations E0'0000<sub>H</sub> through E0'001F<sub>H</sub> of the internal PSRAM and then executed.

*Note: For loading more code, two possibilities exist:*

- via a 2<sup>nd</sup>-level loader - see below
- using the **Enhanced UART Bootstrap Loader** - refer to [Section 12.6.2.2](#)

### Second Level Bootloader

Most probably the initially loaded routine will load additional code or data, as an average application is likely to require substantially more instructions than could fit into 32 bytes. This second receive loop may directly use the pre-initialized interface U0C0 to receive data and store it to arbitrary user-defined locations.

The example code below shows how to fit such a 2<sup>nd</sup>-level loader into the available 32 bytes. This is possible due to the pre-initialized serial channel and the pre-set registers (see [Table 12-10](#)).

```
;Example for Secondary UART Bootstrap Loader Routine
;-----
TargetStart LIT  '0E00020H'      ;Definition of target area:
TargetEnd   LIT  '0E001FFH'      ;480 bytes in this example
StartOfCode LIT  '0E00100H'      ;Continue executing here...
                                   ;...after download

Level2Loader:
    DISWDT                      ;No WDT for further download
    MOV     DPP0,#(PAG TargetStart)
    MOV     R10,#(DPP0:TargetStart);Set pointer to target area
Level2MainLoop:
    MOV     [R1],R3              ;Clear RIF for new byte
Level2RecLoop:
    MOV     R4,[R0]              ;Access PSR
    JNB     R4.14,Level2RecLoop  ;Wait for RIF
    MOVB    [R10],[R2]           ;Copy new byte to target
    CMPI1   R10,#POF (TargetEnd);All bytes received??
    JMPR    cc_NE,Level2MainLoop ;Repeat for complete area
Level2Terminate:
    JMPS    SEG StartOfCode, SOF StartOfCode
```

## Startup Configuration and Bootstrap Loading

### Specific Settings

The following configuration is automatically set when the XE16xyM has entered Standard UART BSL mode:

**Table 12-10 Standard UART BSL-Specific State**

Item	Value	Comments
U0C0_CCR	0002 <sub>H</sub>	ASC mode selected for USIC0 Channel 0
U0C0_PCRL	0401 <sub>H</sub>	1 stop bit, three RxD-samples at point 4
U0C0_SCTRL	0002 <sub>H</sub>	Passive data level = 1
U0C0_SCTRH	0707 <sub>H</sub>	8 data bits
U0C0_FDRL	43FF <sub>H</sub>	Normal divider mode 1:1 selected
U0C0_BRGH	0XXX <sub>H</sub>	Measured PDIV value (zero-byte) in bits[9:0]
U0C0_BRGL	1C00 <sub>H</sub>	Normal mode, FDIV, 8 clocks/bit
U0C0_DX0CR	0003 <sub>H</sub>	Data input selection
DPP1	0081 <sub>H</sub>	Points to USIC0 base address <sup>1)</sup>
R0	4044 <sub>H</sub>	Pointer to U0C0_PSR <sup>1)</sup>
R1	4048 <sub>H</sub>	Pointer to U0C0_PSCR <sup>1)</sup>
R2	405C <sub>H</sub>	Pointer to U0C0_RBUF <sup>1)</sup>
R3	4000 <sub>H</sub>	Mask to clear RIF <sup>1)</sup>
Devices in 144/100-pin package:		
P7_IOCRO3	00B0 <sub>H</sub>	P7.3 is push/pull output (TxD)
P7_IOCRO4	0020 <sub>H</sub>	P7.4 is input with pull-up (RxD)
Devices in 64-pin package:		
P2_IOCRO3	00B0 <sub>H</sub>	P2.3 is push/pull output (TxD)
P2_IOCRO4	0020 <sub>H</sub>	P2.4 is input with pull-up (RxD)

1) This register setting is provided for a 2<sup>nd</sup>-level loader routine (see at [Page 12-21](#)).

The identification byte identifies the device to be booted. The following codes are defined:

55<sub>H</sub>: 8xC166.

A5<sub>H</sub>: Previous versions of the C167 (obsolete).

B5<sub>H</sub>: Previous versions of the C165.

C5<sub>H</sub>: C167 derivatives.

D5<sub>H</sub>: All devices equipped with identification registers (including the XE16xyM).



## **Startup Configuration and Bootstrap Loading**

*Note: The identification byte  $D5_H$  does not directly identify a specific derivative. This information can, in this case, be obtained from the identification registers.*

### **12.6.2.2 Enhanced UART Bootstrap Loader**

The enhanced UART bootstrap loader transfers program code/data via Channel 0 of USIC0 Module (U0C0) into PSRAM.

Data is transferred from the external host to the XE16xyM using asynchronous eight-bit data frames without parity (1 start bit, 1 stop bit). The length of the code/data is not fixed as in the [Standard UART Bootstrap Loader](#) but can be arbitrary up to the PSRAM total size minus 256 bytes. Also the code execution can start from arbitrary PSRAM address, as well as the initial baudrate can be changed - e.g. increased for faster transfer of long code/data blocks.

The initial steps of this bootloader are the same as of the [Standard UART Bootstrap Loader](#). XE16xyM first scans the RxD line to receive a zero byte, i.e. one start bit, eight 0 data bits and one stop bit. From the duration of this zero byte it calculates the corresponding baudrate factor with respect to the current CPU clock, initializes the serial interface U0C0 accordingly and switches pin TxD to output. Using this baudrate, an identification byte ( $DA_H$ ) is returned to the host.

The next steps in this mode are to process the so-called Bootloader Header as follows:

1. XE16xyM sends the current PDIV divider from U0C0\_BRGH register - the 10-bit value is sent in 2 bytes

*Note: In this bootloader, the multi-byte values are sent in high-to-low order.*

2. XE16xyM receives and sends back to the host a Header\_Code (1B)
3. XE16xyM receives and sends back to the host number of bytes to be transferred Code\_Length (3B)
4. XE16xyM receives and sends back to the host the start address STADD for code-execution (3B)
  - the segment address (highest STADD byte) must equal  $E0_H$  for XE16xyM
5. XE16xyM receives and sends back to the host a value for PDIV divider (2B, bits[9:0] effective only)
  - if the new value is different from the current - the new one is written into U0C0\_BRGH register and a zero confirmation byte is sent back to the host with baudrate already changed
6. XE16xyM receives and sends back to the host a Trailer\_Code (1B)
  - a) if both the Header\_Code and Trailer\_Code are equal to the XE16xyM identification byte ( $DA_H$ ) - the Bootloader sends to the Host a zero byte and continues further;
  - b) if the above condition is not true - the Bootloader sends an identification byte ( $DA_H$ ) to the host and restarts Header processing again from point 1.

## Startup Configuration and Bootstrap Loading

Once the Header is successfully processed according to the above steps, the Bootstrap loader receives Code\_Length bytes and stores them sequentially starting from the beginning of PSRAM at address E0'0000<sub>H</sub>.

**Attention: The user must care, that the number of Bytes sent is not bigger than available in the device PSRAM minus 256.**

The Bootstrap loader starts code-execution after the last byte is received and stored. The execution is started from address STADD as received within the header.

### Specific Settings

The following configuration is automatically set when the XE16xyM has entered Enhanced UART BSL mode:

**Table 12-11 Enhanced UART BSL-Specific State**

Item	Value	Comments
U0C0_CCR	0002 <sub>H</sub>	ASC mode selected for USIC0 Channel 0
U0C0_PCRL	0401 <sub>H</sub>	1 stop bit, three RxD-samples at point 4
U0C0_SCTRL	0002 <sub>H</sub>	Passive data level = 1
U0C0_SCTRH	0707 <sub>H</sub>	8 data bits
U0C0_FDRL	43FF <sub>H</sub>	Normal divider mode 1:1 selected
U0C0_BRGH	0XXX <sub>H</sub>	PDIV-value as sent by the host inside header
U0C0_BRGL	1C00 <sub>H</sub>	Normal mode, FDIV, 8 clocks/bit
U0C0_DX0CR	0003 <sub>H</sub>	Data input selection
Devices in 144/100-pin package:		
P7_IOCRO3	00B0 <sub>H</sub>	P7.3 is push/pull output (TxD)
P7_IOCRO4	0020 <sub>H</sub>	P7.4 is input with pull-up (RxD)
Devices in 64-pin package:		
P2_IOCRO3	00B0 <sub>H</sub>	P2.3 is push/pull output (TxD)
P2_IOCRO4	0020 <sub>H</sub>	P2.4 is input with pull-up (RxD)

The identification byte identifies the device to be booted. XE16xyM is the first microcontroller family supporting Enhanced UART BSL mode, the code defined for it is DA<sub>H</sub>.

*Note: The identification byte does not directly identify a specific derivative. This information can, in this case, be obtained from the identification registers.*

### 12.6.2.3 Choosing the Baudrate for the BSL

The calculation of the serial baudrate for U0C0 from the length of the first zero byte that is received, allows the operation of the bootstrap loader of the XE16xyM with a wide range of baudrates. However, the upper and lower limits have to be kept, in order to ensure proper data transfer.

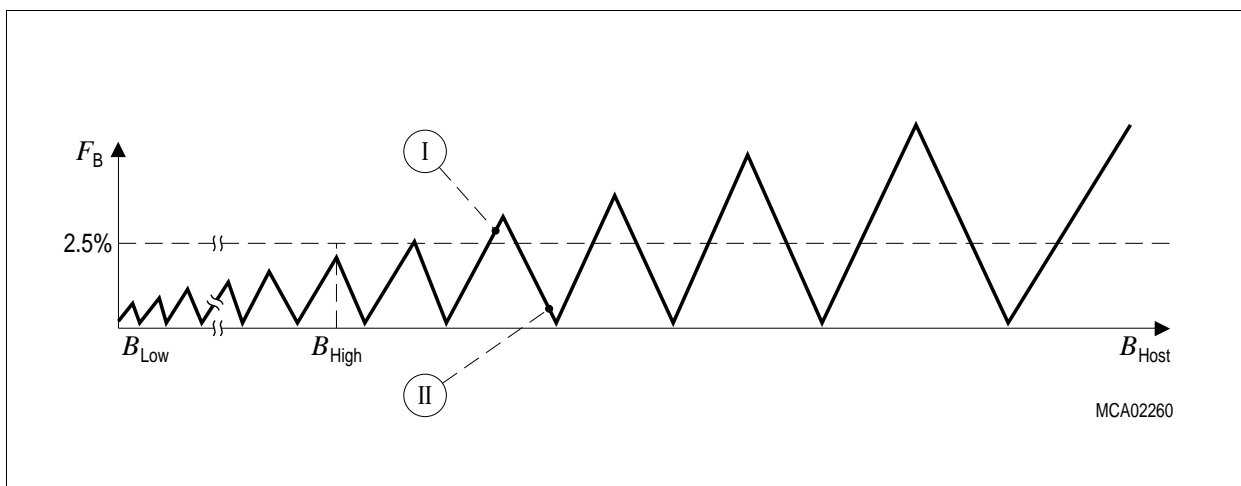
The XE16xyM uses bitfield PDIV to measure the length of the initial zero byte. The quantization uncertainty of this measurement implies the deviation from the real baudrate.

For a correct data transfer from the host to the XE16xyM the maximum deviation between the internal initialized baudrate for U0C0 and the real baudrate of the host should be below 2.5%. The deviation ( $F_B$ , in percent) between host baudrate and XE16xyM baudrate can be calculated via [Equation \(12.1\)](#):

$$F_B = \left| \frac{B_{Contr} - B_{Host}}{B_{Contr}} \right| \times 100\% \quad F_B \leq 2.5\% \quad (12.1)$$

*Note: Function ( $F_B$ ) does not consider the tolerances of oscillators and other devices supporting the serial communication.*

This baudrate deviation is a nonlinear function depending on the system clock and the baudrate of the host. The maxima of the function ( $F_B$ ) increase with the host baudrate due to the smaller baudrate prescaler factors and the implied higher quantization error (see [Figure 12-3](#)).



**Figure 12-3 Baudrate Deviation between Host and XE16xyM**

## **Startup Configuration and Bootstrap Loading**

**The minimum baudrate** ( $B_{Low}$  in [Figure 12-3](#)) is determined by the maximum count capacity of bitfield PDIV, when measuring the zero byte, i.e. it depends on the system clock. The minimum baudrate is obtained by using the maximum PDIV count  $2^{10}$  in the baudrate formula. Baudrates below  $B_{Low}$  would cause PDIV to overflow. In this case U0C0 cannot be initialized properly and the communication with the external host is likely to fail.

**The maximum baudrate** ( $B_{High}$  in [Figure 12-3](#)) is the highest baudrate where the deviation still does not exceed the limit, i.e. all baudrates between  $B_{Low}$  and  $B_{High}$  are below the deviation limit.  $B_{High}$  marks the baudrate up to which communication with the external host will work properly without additional tests or investigations.

**Higher baudrates**, however, may be used as long as the actual deviation does not exceed the indicated limit. A certain baudrate (marked I) in [Figure 12-3](#) may e.g. violate the deviation limit, while an even higher baudrate (marked II) in [Figure 12-3](#) stays very well below it. Any baudrate can be used for the bootstrap loader provided that the following three prerequisites are fulfilled:

- the baudrate is within the specified operating range for U0C0
- the external host is able to use this baudrate
- the computed deviation error is below the limit.

*Note: When the bootstrap loader mode is entered after a power reset, the bootstrap loader will begin to operate with  $f_{SYS} = f_{IOSC} \times 2$  (approximately 10 MHz) which will limit the maximum baudrate for U0C0.*

*Higher levels of the bootstrapping sequence can then switch the clock generation mode in order to achieve higher baudrates for the download.*

### **12.6.3 Synchronous Serial Channel Bootstrap Loader**

The Synchronous Serial Channel (SSC) bootstrap loader transfers program code/data from an external serial EEPROM via channel 0 of USIC0 (U0C0) into the PSRAM. The XE16xyM is the master, so no additional elements (except for the EEPROM) are required.

The SSC bootstrap loading is a convenient way for initial and basic (go/fail) testing during software development - it allows many various code-versions to be easily started on the target system by re-programming a serial EEPROM.

During SSC bootstrap loading data is transferred from the external EEPROM to the XE16xyM using synchronous eight-bit data frames with MSB first. The number of data bytes to be received in SSC boot mode is user-selectable. The serial clock rate is set to  $f_{\text{SYS}}/10$ , which results in 1 MHz after a power reset.

Once SSC BSL mode is entered and the respective initialization done, the XE16xyM first reads the header from the first addresses (00...0) of the target EEPROM.

This header consists of two items:

- The memory identification byte:  $D5_H$
- The data size field: 1, 2 or 3 bytes, depending on the EEPROM's addressing mode (8-bit, 16-bit or 24-bit, see [Section 12.6.3.1](#))

If both items are valid the BSL enters a loop to read the number of bytes defined by the data size field via U0C0.

These bytes are stored sequentially into PSRAM starting at location  $E0'0000_H$  and are then executed. Therefore, the size of the PSRAM in the respective derivative determines the real maximum block size to be downloaded.

**Attention: The user must care, that the data-size is not bigger than available in the device PSRAM minus 256 and does not exceed 32512.**

An invalid header (identification byte  $\neq D5_H$ , data size field = 0 or greater than allowed) is indicated by toggling the  $\overline{\text{CS}}$  line low 3 times. This helps debugging during the system setup phase.

### 12.6.3.1 Supported EEPROM Types

The XE16xyM's SSC bootstrap loader assumes an SPI-compatible EEPROM (25xxx series). It supports devices with 8-bit, 16-bit as well as 24-bit addressing. The connected EEPROM type is determined by examining the received header bytes, as indicated in [Table 12-12](#).

*Note: The data size **n** is in bytes.*

**Table 12-12 Determining the EEPROM Type**

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 <sub>H</sub>	Read command	XX <sub>H</sub> default level	Read command	XX <sub>H</sub> default level	Read command	XX <sub>H</sub> default level
2	00 <sub>H</sub>	Address	XX <sub>H</sub>	Address_H	XX <sub>H</sub>	Address_H	XX <sub>H</sub>
3	00 <sub>H</sub>	dummy	D5 <sub>H</sub> :Ident. B	Address_L	XX <sub>H</sub>	Address_M	XX <sub>H</sub>
4	00 <sub>H</sub>	dummy	Size <b>n</b>	dummy	D5 <sub>H</sub> :Ident. B	Address_L	XX <sub>H</sub>
5	00 <sub>H</sub>	dummy	Data Byte 1	dummy	Size <b>n</b> ,high B	dummy	D5 <sub>H</sub> :Ident. B
6	00 <sub>H</sub>	dummy	Data Byte 2	dummy	Size <b>n</b> ,low B	dummy	Size <b>n</b> ,high B
7	00 <sub>H</sub>	dummy	Data Byte 3	dummy	Data Byte 1	dummy	Size <b>n</b> ,mid B
8	00 <sub>H</sub>	dummy	Data Byte 4	dummy	Data Byte 2	dummy	Size <b>n</b> ,low B
9 ...	...	dummy	Data Byte 5 ... <b>n</b>	dummy	Data Byte 3 ... <b>n</b>	dummy	Data Byte 1 ... <b>n</b>

*Note: The value of the returned default bytes (indicated as XX<sub>H</sub>) depends on the employed EEPROM type.*

### 12.6.3.2 Specific Settings

When the XE16xyM has entered the SSC BSL mode, the following configuration is automatically set:

**Table 12-13 SSC BSL-Specific State**

Item	Value	Comments
U0C0_CCR	0001 <sub>H</sub>	SSC mode selected for USIC0 Channel 0
U0C0_PCRL	0011 <sub>H</sub>	SSC master mode, frequency from fPPP
U0C0_PCRH	8000 <sub>H</sub>	MCLK generation is enabled
U0C0_SCTRL	0103 <sub>H</sub>	MSB first, passive data level=1
U0C0_SCTRH	073F <sub>H</sub>	8 data bits, infinite frame
U0C0_DX0CR	0015 <sub>H</sub>	Data input selection
U0C0_FDRL	43FF <sub>H</sub>	Normal divider mode 1:1 selected
U0C0_BRGL	0000 <sub>H</sub>	Normal mode, FDIV - default value after reset
U0C0_BRGH	8004 <sub>H</sub>	Passive levels MCLK/SCLK=0, PDIV=4
P2_IOCRO3	00D0 <sub>H</sub>	P2.3 is open-drain output (MTSR)
P2_IOCRO4	0020 <sub>H</sub>	P2.4 is input with pull-up (MRST)
P2_IOCRO5	00D0 <sub>H</sub>	P2.5 is open-drain output (SCLK)
P2_IOCRO6	00C0 <sub>H</sub>	P2.6 is open-drain output (SLS)

### **12.6.4 CAN Bootstrap Loader**

The CAN bootstrap loader transfers program code/data via node 0 of the MultiCAN module into the PSRAM. Data is transferred from the external host to the XE16xyM using eight-byte data frames. The number of data frames to be received is programmable and determined by the 16-bit data message count value DMSGC.

The communication between XE16xyM and external host is based on the following three CAN standard frames:

- Initialization frame - sent by the external host to the XE16xyM
- Acknowledge frame - sent by the XE16xyM to the external host
- Data frame(s) - sent by the external host to the XE16xyM

The initialization frame is used in the XE16xyM for baud rate detection. After a successful baud rate detection is reported to the external host by sending the acknowledge frame, data is transmitted using data frames. [Table 12-14](#) shows the parameters and settings for the three utilized CAN standard frames.

*Note: The CAN bootstrap loader requires a point-to-point connection with the host, i.e. the XE16xyM must be the only CAN node connected to the network. A crystal with at least 4 MHz is required for CAN bootstrap loader operation.*

#### **Initialization Phase**

The first BSL task is to determine the CAN baud rate at which the external host is communicating. Therefore the external host must send initialization frames continuously to the XE16xyM. The first two data bytes of the initialization frame must include a 2-byte baud rate detection pattern (5555<sub>H</sub>), an 11-bit (sent in 2 bytes) identifier ACKID<sup>1)</sup> for the acknowledge frame, a 16-bit data message count value DMSGC, and an 11-bit (2-byte) identifier DMSGID<sup>1)</sup> to be used by the data frame(s).

The CAN baud rate is determined by analyzing the received baud rate detection pattern (5555<sub>H</sub>) and the baud rate registers of the MultiCAN module are set accordingly. The XE16xyM is now ready to receive CAN frames with the baud rate of the external host.

#### **Acknowledge Phase**

In the acknowledge phase, the bootstrap loader waits until it receives the next correctly recognized initialization frame from the external host, and acknowledges this frame by generating a dominant bit in its ACK slot. Afterwards, the bootstrap loader transmits an acknowledge frame back to the external host, indicating that it is now ready to receive data frames. The acknowledge frame uses the message identifier ACKID that has been received with the initialization frame.

1) The CAN bootstrap loader copies the two identifier bytes received in the initialization frame directly to register MOAR. Therefore, the respective fields in the initialization frame must contain the intended identifier padded with two dummy bits at the lower end and extended with bitfields IDE (=0<sub>B</sub>) and PRI (=01<sub>B</sub>) at the upper end.



## Startup Configuration and Bootstrap Loading

To summarize: the external host must send initialization frames (the content as above defined) continuously until an acknowledge frame is received back from the XE16xyM having the same message identifier as sent by the host in data bytes 2/3 from the initialization frame, then the **Data Transmission Phase** begins.

### Data Transmission Phase

In the data transmission phase, data frames are sent by the external host and received by the XE16xyM. The data frames use the 11-bit data message identifier DMSGID that has been sent with the initialization frame. Eight data bytes are transmitted with each data frame. The first data byte is stored in PSRAM at E0'0000<sub>H</sub>. Consecutive data bytes are stored at incrementing addresses.

Both communication partners evaluate the data message count DMSGC until the requested number of CAN data frames has been transmitted. After the reception of the last CAN data frame, the bootstrap loader finishes and executes the loaded code.

### Timing Parameters

There are no general restrictions for CAN timings of the external host. During the initialization phase the external host transmits initialization frames. If no acknowledge frame is sent back within a certain time as defined in the external host (e.g. after a dedicated number of initialization frame transmissions), the external host can decide that the XE16xyM is not able to establish the CAN boot communication link.

**Table 12-14 CAN Bootstrap Loader Frames**

Frame Type	Parameter	Description
Initialization Frame	Identifier	11-bit, don't care
	DLC = 8	Data length code, 8 bytes within CAN frame
	Data bytes 0/1	Baud rate detection pattern (5555 <sub>H</sub> )
	Data bytes 2/3	Acknowledge message identifier ACKID (complete register contents)
	Data bytes 4/5	Data message count DMSGC, 16-bit
	Data bytes 6/7	Data message identifier DMSGID (complete register contents)
Acknowledge Frame	Identifier	Acknowledge message identifier ACKID as received by data bytes [3:2] of the initialization frame
	DLC = 4	Data length code, 4 bytes within CAN frame
	Data bytes 0/1	Contents of bit-timing register
	Data bytes 2/3	Copy of acknowledge identifier from initialization frame

**Startup Configuration and Bootstrap Loading**

**Table 12-14 CAN Bootstrap Loader Frames (cont'd)**

Frame Type	Parameter	Description
Data frame	Identifier	Data message identifier DMSGID as sent by data bytes [7:6] of the initialization frame
	DLC = 8	Data length code, 8 bytes within CAN frame
	Data bytes 0 to 7	Data bytes, assigned to increasing destination (PSRAM) addresses

### 12.6.4.1 Specific Settings

When the XE16xyM has entered the CAN BSL mode, the following configuration is automatically set:

**Table 12-15 CAN BSL-Specific State**

Item	Value	Comments
P2_IOCRO5	00A0 <sub>H</sub>	P2.5 is push/pull output (TxD)
P2_IOCRO6	0020 <sub>H</sub>	P2.6 is input with pull-up (RxD)
SCU_HPOSCCON	0030 <sub>H</sub>	OSC_HP enabled, External Crystal/Clock mode
SCU_SYSCON0	0001 <sub>H</sub>	OSC_HP selected as system clock
CAN_MOCTR0L	0008 <sub>H</sub>	Message Object 0 Control, low
CAN_MOCTR0H	00A0 <sub>H</sub>	Message Object 0 Control, high
CAN_MOCTR1L	0000 <sub>H</sub>	Message Object 1 Control, low
CAN_MOCTR1H	0F28 <sub>H</sub>	Message Object 1 Control, high
CAN_MOFCTR1H	0400 <sub>H</sub>	Message Object Function Control, high
CAN_MOAMR0H	1FFF <sub>H</sub>	Message Object 0 - Acceptance Mask bit set
CAN_NPCRO	0003 <sub>H</sub>	Data input selection

## 12.6.5 Summary of Bootstrap Loader Modes

This table summarizes the external hardware provisions that are required to activate a bootstrap loader in a system.

**Table 12-16 Configuration Data for Bootstrap Loader Modes**

<b>Bootstrap Loader Mode</b>	<b>Configuration on P10.[7:0] <sup>1)</sup></b>	<b>Receive Line from Host</b>	<b>Transmit Line to Host</b>	<b>Transferred Data</b>	<b>Supported Host Speed</b>
Standard UART	xxxx x110 <sub>B</sub> <sup>2)</sup>	RxD = P7.4	TxD = P7.3	32 bytes	2.4 - 19.2 kbit/s
	xxxx x010 <sub>B</sub>	RxD = P2.4	TxD = P2.3		
Enhanced UART	x001 0000 <sub>B</sub>	RxD = P2.4	TxD = P2.3	l bytes <sup>3)</sup>	2.4-19.2 kbit/s at start, then changeable by Header
Sync. Serial	xxxx 1001 <sub>B</sub>	MRST = P2.4	MTSR = P2.3 SCLK = P2.5 SLS = P2.6	m bytes <sup>4)</sup>	--- (controlled by XE16xyM)
MultiCAN	xxxx 1101 <sub>B</sub>	RxDC0 = P2.6	TxDC0 = P2.5	8 × n bytes <sup>5)</sup>	125 - 500 kbit/s

1) x means that the level on the corresponding pin is irrelevant.

2) Not available in 64-pin package.

3) l = Code\_Length sent by the host, the values allowed are 1...(PSRAM\_size-256).

4) m = data size read from EEPROM, the values allowed are 1...(PSRAM\_size-256) but not bigger than 32512.

5) n = DMSGC, Data Message Count sent by the host with Initialization frame, the values allowed are 1...(PSRAM\_size-256)/8.

## 13 Debug System

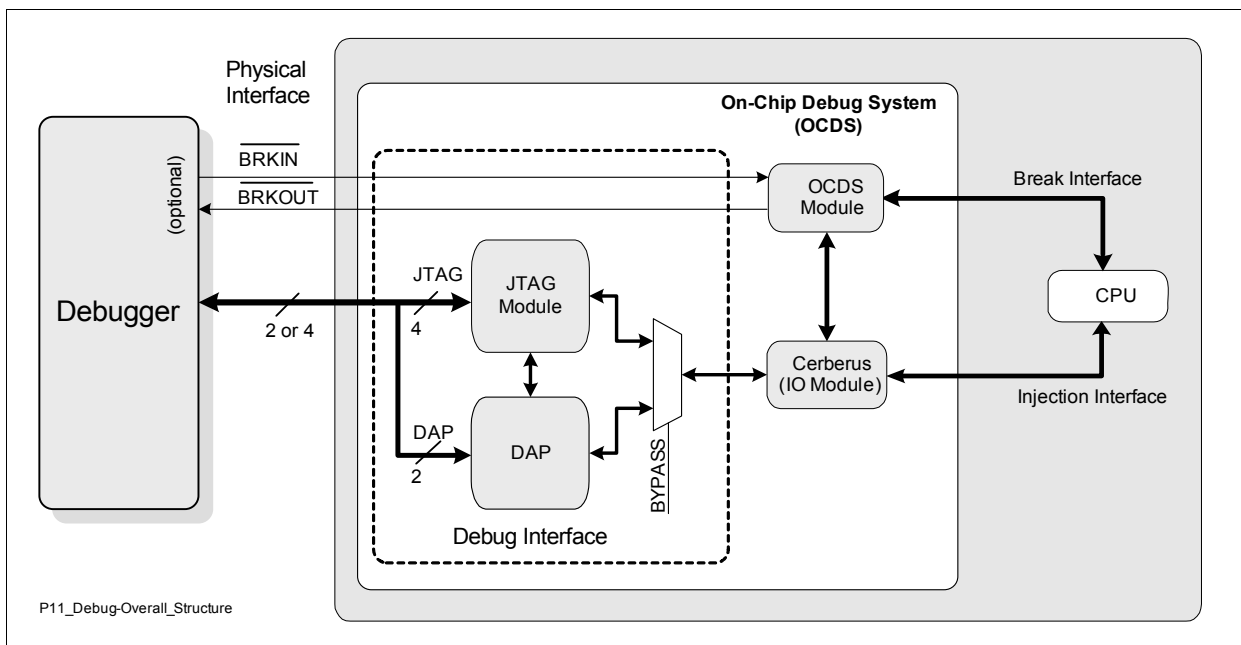
The XE16xyM includes an On-Chip Debug Support (OCDS) system, which provides convenient debugging, controlled directly by an external tool via debug interface pins.

### OCDS Components

- **Debug Interface**
- **Cerberus**
- **OCDS Module**

### On-Chip Debug Support (OCDS)

The OCDS system (**Figure 13-1**) supports a broad range of debug features including breakpoints and the tracing of memory locations. A typical application of the OCDS is to debug user software running on the XE16xyM in a real time system environment.



**Figure 13-1 OCDS Block Diagram**

The OCDS is controlled by an external tool via the **Debug Interface**. The physical interface is either DAP or JTAG plus an optional break interface with one or two pins. The break interface supports very low latency triggers between XE16xyM and tool and/or system environment if needed. The memory mapped OCDS registers are accessible via the DAP/JTAG interface using Cerberus. In addition there is a limited set of special Cerberus debug IO instructions. As an alternative the OCDS can be controlled by a debug monitor program, which communicates with the tool over a user interface like CAN. The OCDS system interacts with the CPU through an injection interface to allow execution of Cerberus-generated instructions, and through a break port.

## **OCDS System Features**

- Hardware, software and external pin breakpoints
- Trigger action can be CPU-halt, monitor call, data transfer and/or  $\overline{\text{BRKOUT}}$  signal
- Read/write access to the whole address space
- Single stepping
- Non intrusive debugging (no debug monitor needed)
- Debug also possible over user interface like CAN (with debug monitor)
- DAP or JTAG interface and optional break interface
- Injection of arbitrary CPU instructions
- Fast memory tracing through transfer to external bus (if available)

## **13.1 Debug Interface**

The Debug Interface allows to access OCDS resources. Data can be transferred to/from all on- and off-chip memories and memory mapped control registers.

### **Features and Functions**

- Independent interface for OCDS
- DAP (Device Access Port) or alternatively JTAG
- Break interface for external trigger input and signaling of internal triggers
- Generic memory access functionality
- Independent data transfer channel for e.g. programming of flash memory

The Debug Interface consists of:

- **DAP Interface**
- Alternatively **JTAG Interface** based on the IEEE 1149.1 JTAG standard
- Two additional XE16xyM specific signals - **OCDS Break-Interface**

**Note: The DAP/JTAG clock frequency must be below the current CPU frequency.**

### **DAP Interface**

The DAP interface is a device access port standardized for the latest Infineon microcontrollers. It reduces the pin count to two pins and offers high noise immunity and robustness.

This interface consists of the signals:

- **DAP0** - clock
- **DAP1** - Serial data input/output

### **JTAG Interface**

The JTAG interface is a standardized and dedicated port, primarily provided for boundary scan board tests.

This interface consists of the JTAG IEEE.1149.1-2001 standard signals:

- **TDI** - Serial data input
- **TDO** - Serial data output
- **TCK** - JTAG clock
- **TMS** - State machine control signal

### OCDS Break-Interface

Two optional additional signals provide a direct trigger interface between the Debugger and XE16xyM **OCDS Module**:

- **BRKIN** (BReaK IN request) allows to trigger directly one of the **Debug Actions**.
- **BRKOUT** (BReaK OUT signal) can be activated by OCDS to notify the external world that some predefined debug event has happened.

### 13.1.1 Routing of Debug Signals

The signals used to connect an external debugger via the JTAG interface and the break interface usually conflict with the requirements to have as many IO pins as possible for the application. In the XE16xyM, these signals are only provided as alternate functions (no dedicated pins). To minimize the impact caused by the debug interface pins, these signals can be mapped to various positions. Thus, each application can select the variant with the least impact. This is controlled via the Debug Pin Routing Register **DBGPRR**. Pin **BRKOUT** can be assigned to pins P6.0, P10.11, P1.5, or P9.3 as a standard alternate output signal via the respective IOC register.

#### 13.1.1.1 Register DBGPRR

This register controls the pin routing of the DAP/JTAG pins. The routing options are controlled with the register **DBGPRR**. The bit field description of **DBGPRR** includes all routing options for all derivatives of the family with DAP/JTAG Interface. For derivatives with lower pin count packages, unavailable positions shall be treated as reserved.

The **DBGPRR** is set during start-up as described in **Section 12.3.2**.

#### DBGPRR

**Debug Pin Routing Register**      **ESFR (F06E<sub>H</sub>/37<sub>H</sub>)**      **Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>TRS TL</b>	<b>TRS TS</b>	<b>TRS TGT</b>	<b>DBG EN</b>	<b>JTAG DAP</b>	<b>DPR E</b>	<b>DPR BRKIN</b>		<b>DPR TCK</b>		<b>DPR TMS</b>		<b>DPR TDI</b>		<b>DPR TDO</b>	
rh	rh	rw	rw	rw	r	rw		rw		rw		rw		rw	

Field	Bits	Type	Description
<b>DPRTDO</b>	[1:0]	rw	<b>Pin Routing for DAP1/TDO</b> 00 <sub>B</sub> P7.0 01 <sub>B</sub> P10.12 10 <sub>B</sub> Reserved 11 <sub>B</sub> Reserved
<b>DPRTDI</b>	[3:2]	rw	<b>Pin Routing for TDI</b> 00 <sub>B</sub> P5.2 01 <sub>B</sub> P10.10 10 <sub>B</sub> P7.2 11 <sub>B</sub> P8.3
<b>DPRTMS</b>	[5:4]	rw	<b>Pin Routing for TMS</b> 00 <sub>B</sub> P5.4 01 <sub>B</sub> P10.11 10 <sub>B</sub> P7.3 11 <sub>B</sub> P8.4
<b>DPRTCK</b>	[7:6]	rw	<b>Pin Routing for DAP0/TCK</b> 00 <sub>B</sub> P2.9 01 <sub>B</sub> P10.9 10 <sub>B</sub> P7.4 11 <sub>B</sub> P8.5
<b>DPRBRKIN</b>	[9:8]	rw	<b>Pin Routing for BRKIN</b> 00 <sub>B</sub> P5.10 01 <sub>B</sub> P10.8 10 <sub>B</sub> P7.1 11 <sub>B</sub> P8.6
<b>DPRE</b>	10	rw	<b>Port 13 Routing for DAP/JTAG</b> DAP0/TCK P13.6, DAP1/TDO P13.8, TMS P13.3, TDI P13.5. Will overrule all other routing settings for these pins. 0 <sub>B</sub> Port 13 is not used. 1 <sub>B</sub> Port 13 is used
<b>JTAG_DAP</b>	11	rw	<b>Selection of Debug Interface</b> 0 <sub>B</sub> DAP is used 1 <sub>B</sub> JTAG is used
<b>DBGEN</b>	12	rw	<b>Enable for selected Debug Interface</b> 0 <sub>B</sub> Interface is disabled 1 <sub>B</sub> Interface is enabled

Field	Bits	Type	Description
TRSTGT	13	rw	<b>Gating of TRST Pin</b> $0_B$ DAP/JTAG reset is internally held active $1_B$ $\overline{\text{TRST}}$ pin is routed to DAP/JTAG reset
TRSTS	14	rh	<b>TRST Pin Value</b> Current value of $\overline{\text{TRST}}$ pin
TRSTL	15	rh	<b>Latched TRST Pin Start-up Value</b> Value of $\overline{\text{TRST}}$ pin latched by PORST release

## 13.2 OCDS Module

The application of the OCDS Module is to debug user software running on the CPU in the customer's system. This is done with an external debugger, which controls the OCDS Module via the independent [Debug Interface](#).

### Features

- Hardware, software and external pin breakpoints
- Hardware trigger generation for breakpoints and external pin output
  - Four single address or two address ranges for instruction or data
  - Combination of instruction (range) and data address (range)
  - Combination of data address (range) and data value (range)
  - Task ID, optional in combination with address (range) for instruction or data
  - Masked comparisons for addresses and data
- The OCDS can also be configured by a debug monitor program
- Single stepping with monitor or CPU halt
- Higher priority interrupts can still be served if CPU is halted
- Instruction pointer visible in Halt Mode

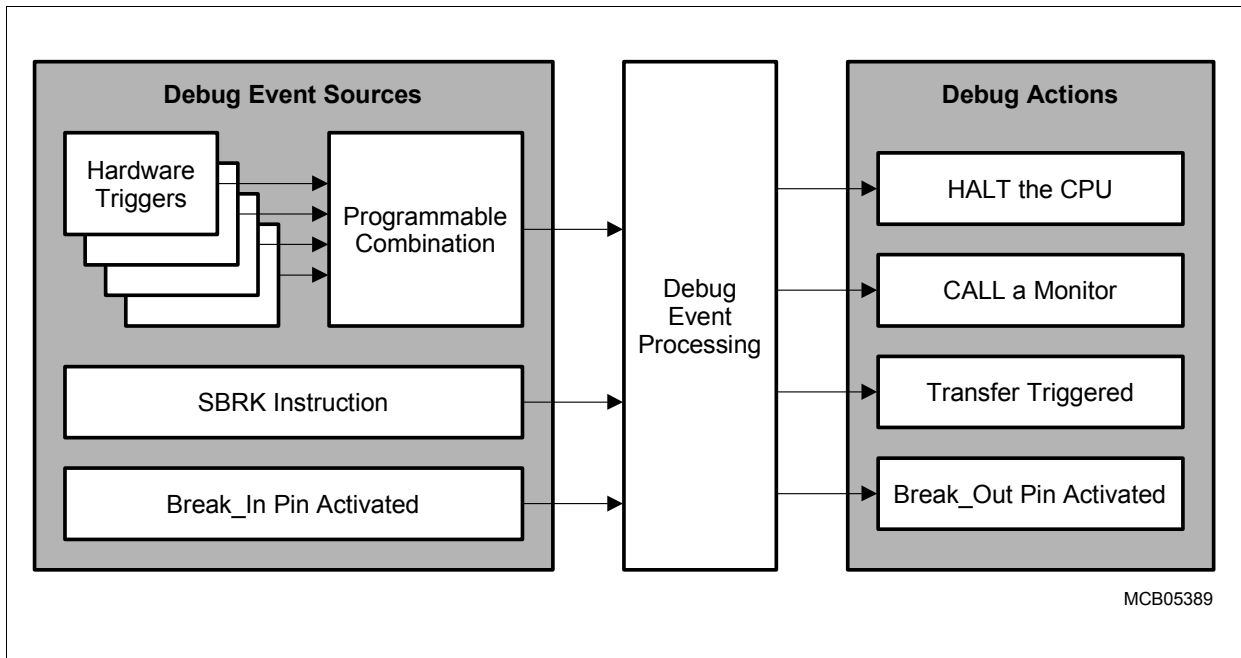
### Basic Concept

The on chip debug concept is split up into two parts. The first part covers the generation of debug events and the second part defines what actions are taken when a debug event is generated.

- Debug events:
  - [Hardware Breakpoints](#)
  - [Software Breakpoints](#)
  - [Break Pin Input](#) activated
- Debug event actions:
  - [Halt Mode](#) of the CPU
  - [Call a Monitor](#)



- **Triggered Transfer**
- **Activate External Pin**



**Figure 13-2 OCDS Concept: Block Diagram**

### 13.2.1 Debug Events

The Debug Events can come from a few different sources.

#### Hardware Breakpoints

The Hardware Breakpoint is a debug-event, raised when a single or a combination of multiple trigger-signals are matching with the programmed conditions. The following hardware trigger sources can be used:

**Table 13-1 Hardware Triggers**

Trigger Source	Size
Task Identifier	16 bits
Instruction Pointer	24 bits
Data address of reads (two buses monitored)	2 × 24 bits
Data address of writes	24 bits
Data value (reads or writes)	16 bits

## **Software Breakpoints**

A special SBRK (Software BReak) instruction is defined with opcode 0x8C00. It can be used for instance by a debugger to temporarily replace code held in RAM in order to implement Software Breakpoints. When the SBRK instruction has been decoded and it reaches the execute stage, the whole pipeline is canceled including the SBRK instruction. This implies that the next instruction will be fetched from the address the SBRK was found at.

The further behavior is dependent on how OCDS has been programmed:

- if the OCDS is enabled and the software breakpoints are also enabled, then the CPU goes into **Halt Mode**
- if the OCDS is disabled or the software breakpoints are disabled, then the Software Break Trap (SBRKTRAP) is executed - Class A Trap, number 08<sub>H</sub>

## **Break Pin Input**

An external debug break pin ( $\overline{\text{BRKIN}}$ ) is provided to allow the debugger to asynchronously interrupt the processor.

### **13.2.2 Debug Actions**

When the OCDS is enabled and a debug event is generated, one of the following actions is taken:

#### **Triggered Transfer**

One of the actions that can be specified to occur on a debug event being raised is to trigger the **Cerberus**:

- to execute a Data Transfer. This can be used in critical routines where the system cannot be interrupted to transfer a memory location
- to inject an instruction to the CPU, using this mechanism, an arbitrary instruction can be injected into the XE16xyM pipeline

#### **Halt Mode**

Upon this Action the OCDS Module sends a Break-Request to the CPU.

The CPU accepts this request, if the OCDS Break Level is higher than current CPU priority level. In case a Break-Request is accepted, the system suspends execution with halting the instruction flow.

The Halt Mode can be still interrupted by higher priority user interrupts. It then relies on the external debugger system to interrogate the target purely through reading and updating via the debug interface.

### **Call a Monitor**

One of the possible actions to be taken when a debug event is raised is to call a Monitor Program. This quick entry to a Monitor allows a flexible debug environment to be defined which is capable of satisfying many of the requirements for efficient debugging of a real time system. In the common case the Monitor has the highest priority and can not be interrupted by any other requesting source.

It is also possible to have an Interruptible Monitor Program. In such a case safety critical code can be still served while the Monitor (Debugger) is active, which gives a maximum flexibility to the user.

### **Activate External Pin**

This action activates the external pin  $\overline{\text{BRKOUT}}$  of the **OCDS Break-Interface**. It can be used in critical routines where the system cannot be interrupted to signal to the external world that a particular event has happened. Note that the code execution timing is not affected.

## **13.3 Cerberus**

Cerberus is the module which provides and controls all the operations necessary to interact between the external debugger (via the **Debug Interface**), the **OCDS Module** and the internal system of XE16xyM.

### **Features**

- DAP/JTAG interface is used as control and data channel
- Generic read/write functionality (RW mode) with access to the whole address space
- Reading and writing of general-purpose registers (GPRs)
- Injection of arbitrary instructions
- External host controls all transactions
- All transactions are available at normal run time and in halt mode
- Priority of transactions can be configured
- Full support for communication between the monitor and debugger
- Optional error protection
- Tracing memory locations through transferring values to the external bus
- Analysis register for internal bus locking situations

The target application of Cerberus is to use the DAP/JTAG interface as an independent port for on-chip debug support. The external debugger can access the OCDS registers and arbitrary memory locations with the injection mechanism.

### **13.3.1 Functional Overview**

Cerberus is operated by an external debugger across the DAP/JTAG interface. The Debugger uses Cerberus IO Instructions to perform bidirectional data-transfers. Cerberus has two main modes of operation:

#### **Read/Write (RW) Mode**

RW Mode is the most common way to operate Cerberus. This mode is used to read and write memory locations or to inject instructions into the CPU pipeline. The injection interface to the CPU is actively used in this mode.

All Cerberus IO Instructions can be used in RW mode. The access to any memory location is performed with injected instructions, as a PEC transfer.

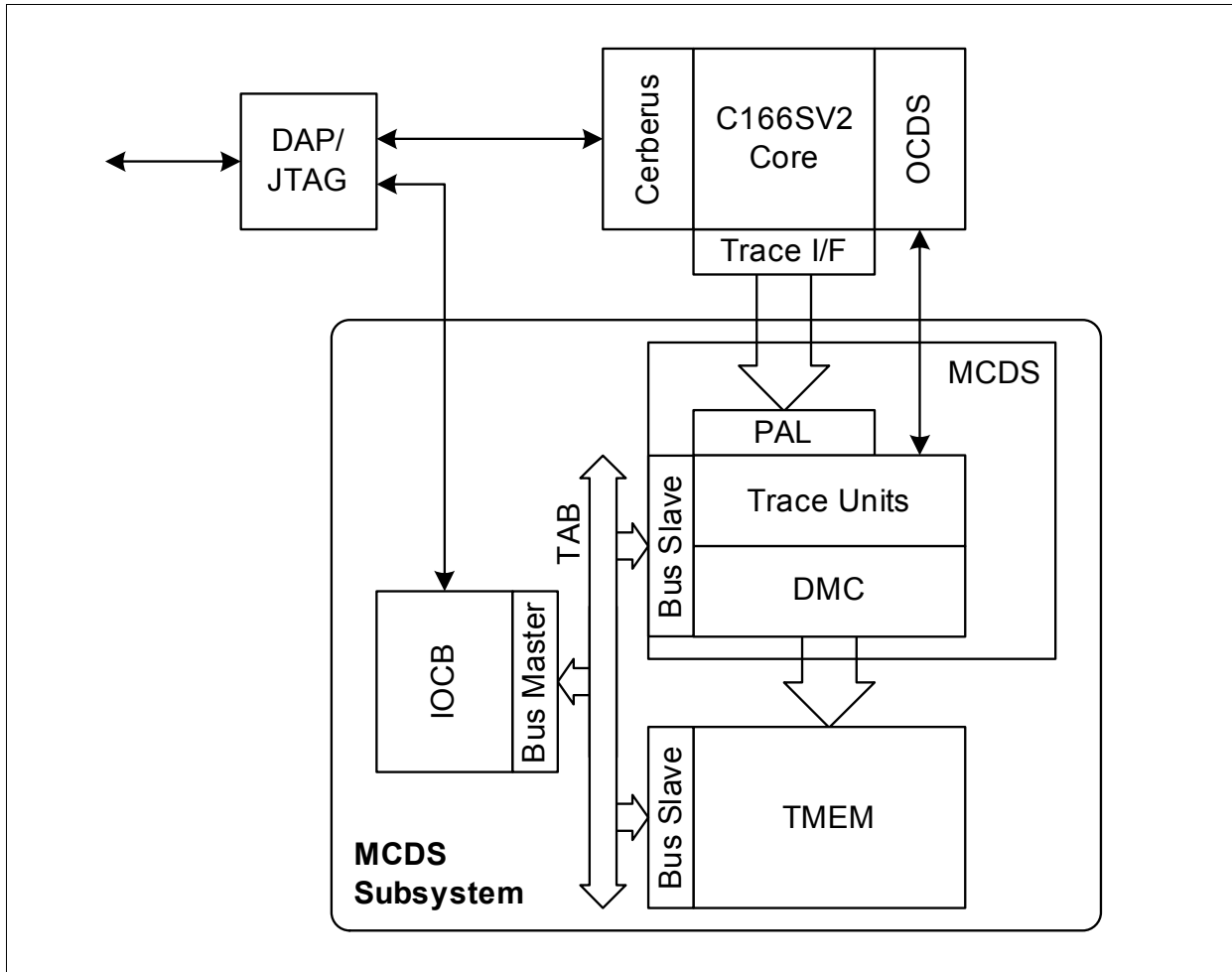
#### **Communication (COM) Mode**

In COM mode the debugger communicates with a monitor program running on the CPU. The difference to **Read/Write (RW) Mode** is that the read or write request is not actively executed. It just sets request bits in a CPU accessible status register to signal the monitor, that the debugger wants to send or receive a value. The monitor has to poll this status register, e.g. triggered by a timer interrupt.

COM Mode is the default mode after reset. It can be used to exchange keys with the application software of a locked (RW Mode disabled) device and to unlock RW Mode only in case of matching keys.

## 13.4 Emulation Device

The XE16xyM can be emulated using an MCDS (Multi-Core Debug Solution) based Emulation Device with an on-chip trace buffer (**Figure 13-3**). For availability of such an emulator please contact your Infineon tool partner.



**Figure 13-3 Emulation Device Block Diagram**

### 13.4.1 MCDS Use Cases

MCDS allows non intrusive tracing and triggering for debugging, performance analysis and data measurement.

#### Debugging

- Halt on very complex trigger conditions
- Record trace around bug
- Halt (suspend) system when trace buffer full. Read out and continue
- Highly compressed or cycle accurate trace

### **Performance Analysis**

- Continuous measurement of performance indicators
- Trigger on performance indicators

### **Data Measurement**

- Continuous trace of data writes or reads
- Qualified by address ranges and e.g. software task

## **13.4.2 MCDS Features**

MCDS provides a rich set of features, which allow a very detailed analysis of the software and system behavior on all levels.

### **CPU Program Trace**

- Complete program trace for the 24 bit instruction pointer
- Four independent range comparators

### **CPU Ownership Trace**

- Complete trace of the pipeline “user” (e.g. PEC channel)
- Two independent masked comparators for data transfer qualification

### **CPU Status Trace**

- Complete trace of the execution mode of the CPU
- Non-intrusive access to the current status.

### **Write Data Trace**

- Complete trace of write-back transactions (24 bit address, 8 or 16 bit data)
- Four independent range comparators on the absolute write address
- Four independent signed data comparators on the data value

### **Read Data Trace**

- Complete trace of non CPU memory read transactions (24 bit addr., 8 or 16 bit data)
- Four independent range comparators on the absolute address
- Four independent signed data comparators on the data value

### **Trace and Trigger Control**

- Dedicated programmable trace enables for each Trace Unit
- Trigger output to OCDS
- Eight universal 16 bit counters
- Programmable combinations of triggers as count and clear signals

- Programmable limit comparator in each counter
- Passing a limit is available as unique trigger for each counter
- The counter values can be traced
- Counters can be cascaded to implement state machines
- Pre-scaled reference clock available as trigger
- Four performance indicator signals directly from the CPU

### **Watch-point Trace**

- Messages for eight different watch-points
- Messages containing the current count value of any event/performance Counter

### **Time Stamping**

- Precise time stamps based on the emulation clock (32 bit)
- Precise time stamps based on a reference clock (32 bit)
- Programmable cyclic trigger based on reference clock

## **13.5 Boundary-Scan**

The XE16xyM eases board-level analysis in the application system by providing Boundary-Scan according to the IEEE standard 1149.1. It supports testing of the interconnections between several devices mounted on a PCB.

Boundary-Scan is accomplished via the JTAG module, using standard JTAG instructions (IEEE1149.1).

*Note: For Boundary-Scan to operate properly, the JTAG interface must use the default pins. The reset value of register DBGPRR ensures this.*

### **Initialization of Boundary-Scan**

The following sequence is defined to activate Boundary-Scan mode:

- Set  $\overline{\text{PORST}} = 1$ ;  $\overline{\text{TRST}} = 1$ ;  $\overline{\text{TESTM}} = 1$
- Negative Pulse on  $\overline{\text{PORST}}$
- Wait for Power Domain to startup.
- Negative pulse on  $\overline{\text{TRST}}$  to reset the JTAG controller.

Now the test access port for Boundary-Scan is enabled. The Boundary-Scan test can be used for board test with instructions like PRELOAD and EXTEST.

## **14 Instruction Set Summary**

This chapter briefly summarizes the XE16xyM's instructions ordered by instruction classes. This provides a basic understanding of the XE16xyM's instruction set, the power and versatility of the instructions and their general usage.

**A detailed description** of each single instruction, including its operand data type, condition flag settings, addressing modes, length (number of bytes) and object code format is provided in the “**Instruction Set Manual**” for the XE166 Family. This manual also provides tables ordering the instructions according to various criteria, to allow quick references.

### **Summary of Instruction Classes**

Grouping the various instruction into classes aids in identifying similar instructions (e.g. SHR, ROR) and variations of certain instructions (e.g. ADD, ADDDB). This provides an easy access to the possibilities and the power of the instructions of the XE16xyM.

*Note: The used mnemonics refer to the detailed description.*

**Table 14-1 Arithmetic Instructions**

Addition of two words or bytes:	ADD	ADDB
Addition with Carry of two words or bytes:	ADDC	ADDCB
Subtraction of two words or bytes:	SUB	SUBB
Subtraction with Carry of two words or bytes:	SUBC	SUBCB
16 × 16 bit signed or unsigned multiplication:	MUL	MULU
16/16 bit signed or unsigned division:	DIV	DIVU
32/16 bit signed or unsigned division:	DIVL	DIVLU
1's complement of a word or byte:	CPL	CPLB
2's complement (negation) of a word or byte:	NEG	NEGB

**Table 14-2 Logical Instructions**

Bitwise ANDing of two words or bytes:	AND	ANDB
Bitwise ORing of two words or bytes:	OR	ORB
Bitwise XORing of two words or bytes:	XOR	XORB



**Table 14-3 Compare and Loop Control Instructions**

Comparison of two words or bytes:	CMP	CMPB
Comparison of two words with post-increment by either 1 or 2:	CMPI1	CMPI2
Comparison of two words with post-decrement by either 1 or 2:	CMPD1	CMPD2

**Table 14-4 Boolean Bit Manipulation Instructions**

Manipulation of a maskable bit field in either the high or the low byte of a word:	BFLDH	BFLDL
Setting a single bit (to '1'):	BSET	–
Clearing a single bit (to '0'):	BCLR	–
Movement of a single bit:	BMOV	–
Movement of a negated bit:	BMOVN	–
ANDing of two bits:	BAND	–
ORing of two bits:	BOR	–
XORing of two bits:	BXOR	–
Comparison of two bits:	BCMP	–

**Table 14-5 Shift and Rotate Instructions**

Shifting right of a word:	SHR	–
Shifting left of a word:	SHL	–
Rotating right of a word:	ROR	–
Rotating left of a word:	ROL	–
Arithmetic shifting right of a word (sign bit shifting):	ASHR	–

**Table 14-6 Prioritize Instruction**

Determination of the number of shift cycles required to normalize a word operand (floating point support):	PRIOR	–
--	-------	---

**Table 14-7 Data Movement Instructions**

Standard data movement of a word or byte:	MOV	MOVB
Data movement of a byte to a word location with either sign or zero byte extension:	MOVBS	MOVBZ

*Note: The data movement instructions can be used with a big number of different addressing modes including indirect addressing and automatic pointer in-/decrementing.*

**Table 14-8 System Stack Instructions**

Pushing of a word onto the system stack:	PUSH	–
Popping of a word from the system stack:	POP	–
Saving of a word on the system stack, and then updating the old word with a new value (provided for register bank switching):	SCXT	–

**Table 14-9 Jump Instructions**

Conditional jumping to an either absolutely, indirectly, or relatively addressed target instruction within the current code segment:	JMPA	JMPI	JMPR
Unconditional jumping to an absolutely addressed target instruction within any code segment:	JMPS	–	–
Conditional jumping to a relatively addressed target instruction within the current code segment depending on the state of a selectable bit:	JB	JNB	–
Conditional jumping to a relatively addressed target instruction within the current code segment depending on the state of a selectable bit with a post-inversion of the tested bit in case of jump taken (semaphore support):	JBC	JNBS	–

**Table 14-10 Call Instructions**

Conditional calling of an either absolutely or indirectly addressed subroutine within the current code segment:	CALLA	CALLI
Unconditional calling of a relatively addressed subroutine within the current code segment:	CALLR	–
Unconditional calling of an absolutely addressed subroutine within any code segment:	CALLS	–
Unconditional calling of an absolutely addressed subroutine within the current code segment plus an additional pushing of a selectable register onto the system stack:	PCALL	–
Unconditional branching to the interrupt or trap vector jump table in code segment <VECSEG>:	TRAP	–

**Table 14-11 Return Instructions**

Returning from a subroutine within the current code segment:	RET	–
Returning from a subroutine within any code segment:	RETS	–
Returning from a subroutine within the current code segment plus an additional popping of a selectable register from the system stack:	RETP	–
Returning from an interrupt service routine:	RETI	–

**Table 14-12 System Control Instructions**

Resetting the XE16xyM via software:	SRST	–
Entering the Idle mode:	IDLE	–
No function, do not use <sup>1)</sup> :	PWRDN	–
Servicing the Watchdog Timer:	SRVWDT	–
Disabling the Watchdog Timer:	DISWDT	–
Enabling the Watchdog Timer (can only be executed in WDT enhanced mode):	ENWDT	–
Signifying the end of the initialization routine (switches the register security mechanism to “protected” and disables the effect of any later execution of a DISWDT instruction in WDT compatibility mode):	EINIT	–

<sup>1)</sup> Instruction PWRDN is used to enter Power Down mode in previous 16-bit architectures. In the XE16xyM devices, however, PWRDN has no effect and is executed like a NOP instruction.

**Table 14-13 Miscellaneous**

Null operation which requires 2 Bytes of storage and the minimum time for execution:	NOP	–
Definition of an unseparable instruction sequence:	ATOMIC	–
Switch ‘reg’, ‘bitoff’ and ‘bitaddr’ addressing modes to the Extended SFR space:	EXTR	–
Override the DPP addressing scheme using a specific data page instead of the DPPs, and optionally switch to ESFR space:	EXTP	EXTPR
Override the DPP addressing scheme using a specific segment instead of the DPPs, and optionally switch to ESFR space:	EXTS	EXTSR

*Note: The ATOMIC and EXT\* instructions provide support for uninterruptable code sequences e.g. for semaphore operations. They also support data addressing beyond the limits of the current DPPs (except ATOMIC), which is advantageous for bigger memory models in high level languages.*

**Table 14-14 MAC-Unit Instructions**

Multiply (and Accumulate):	CoMUL	CoMAC
Add/Subtract:	CoADD	CoSUB
Shift right/Shift left:	CoSHR	CoSHL
Arithmetic Shift right:	CoASHR	–
Load Accumulator:	CoLOAD	–
Store MAC register:	CoSTORE	–
Compare values:	CoCMP	–
Minimum/Maximum:	CoMIN	CoMAX
Absolute value:	CoABS	–
Rounding:	CoRND	–
Move data:	CoMOV	–
Negate accumulator:	CoNEG	–
Null operation:	CoNOP	–

### Protected Instructions

Some instructions of the XE16xyM which are critical for the functionality of the controller are implemented as so-called Protected Instructions. These protected instructions use the maximum instruction format of 32 bits for decoding, while the regular instructions only use a part of it (e.g. the lower 8 bits) with the other bits providing additional information like involved registers. Decoding all 32 bits of a protected doubleword instruction increases the security in cases of data distortion during instruction fetching. Critical operations like a software reset are therefore only executed if the complete instruction is decoded without an error. This enhances the safety and reliability of a microcontroller system.

## **15 Device Specification**

The device specification describes the electrical parameters of the device. It lists DC characteristics like input, output or supply voltages or currents, and AC characteristics like timing characteristics and requirements.

Other than the architecture, the instruction set, or the basic functions of the XE16xyM core and its peripherals, these DC and AC characteristics are subject to changes due to device improvements or specific derivatives of the standard device.

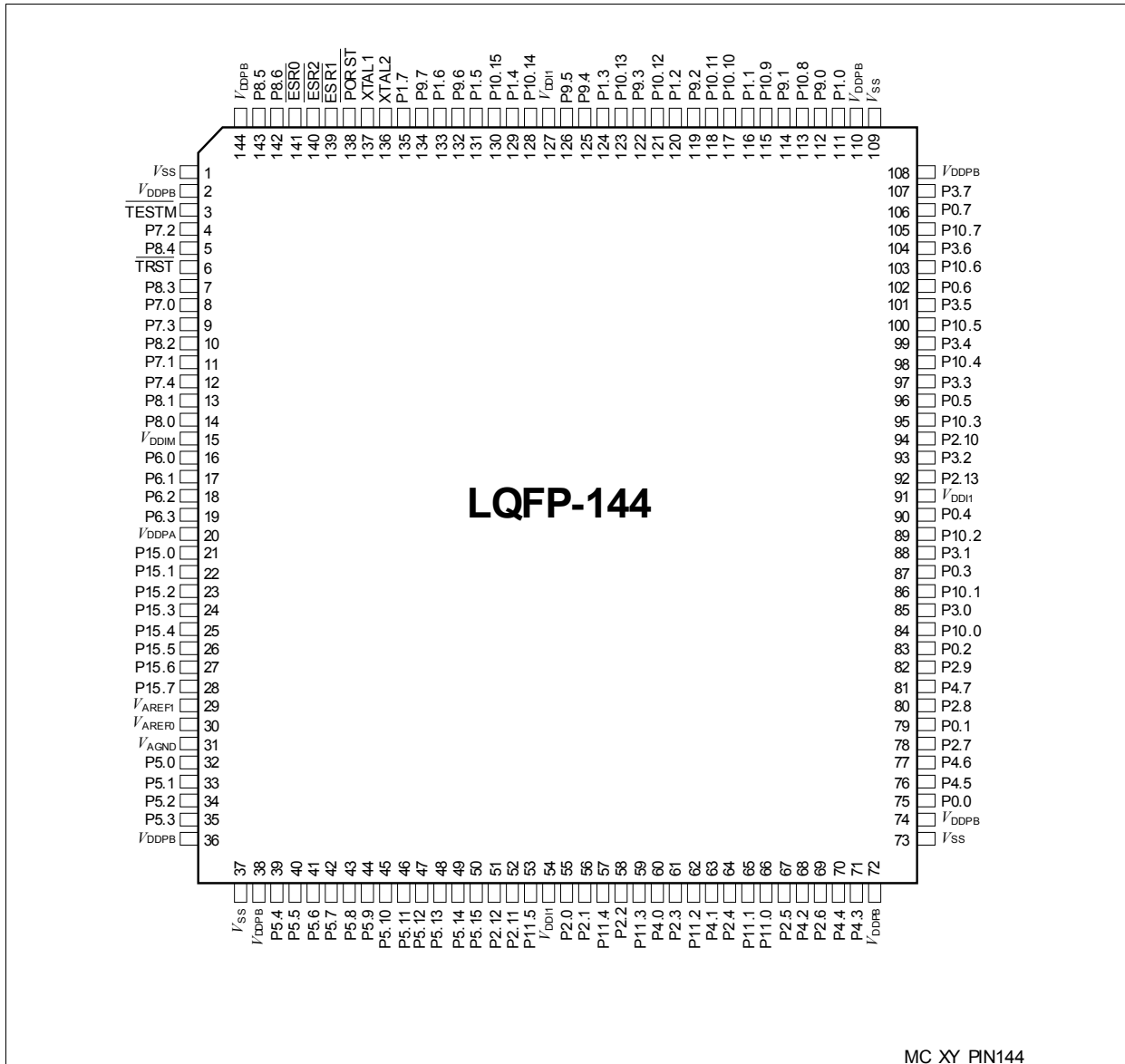
Therefore, these characteristics are not contained in this manual, but rather provided in a separate Data Sheet, which can be updated more frequently.

Please refer to the current version of the Data Sheet of the respective device for all electrical parameters.

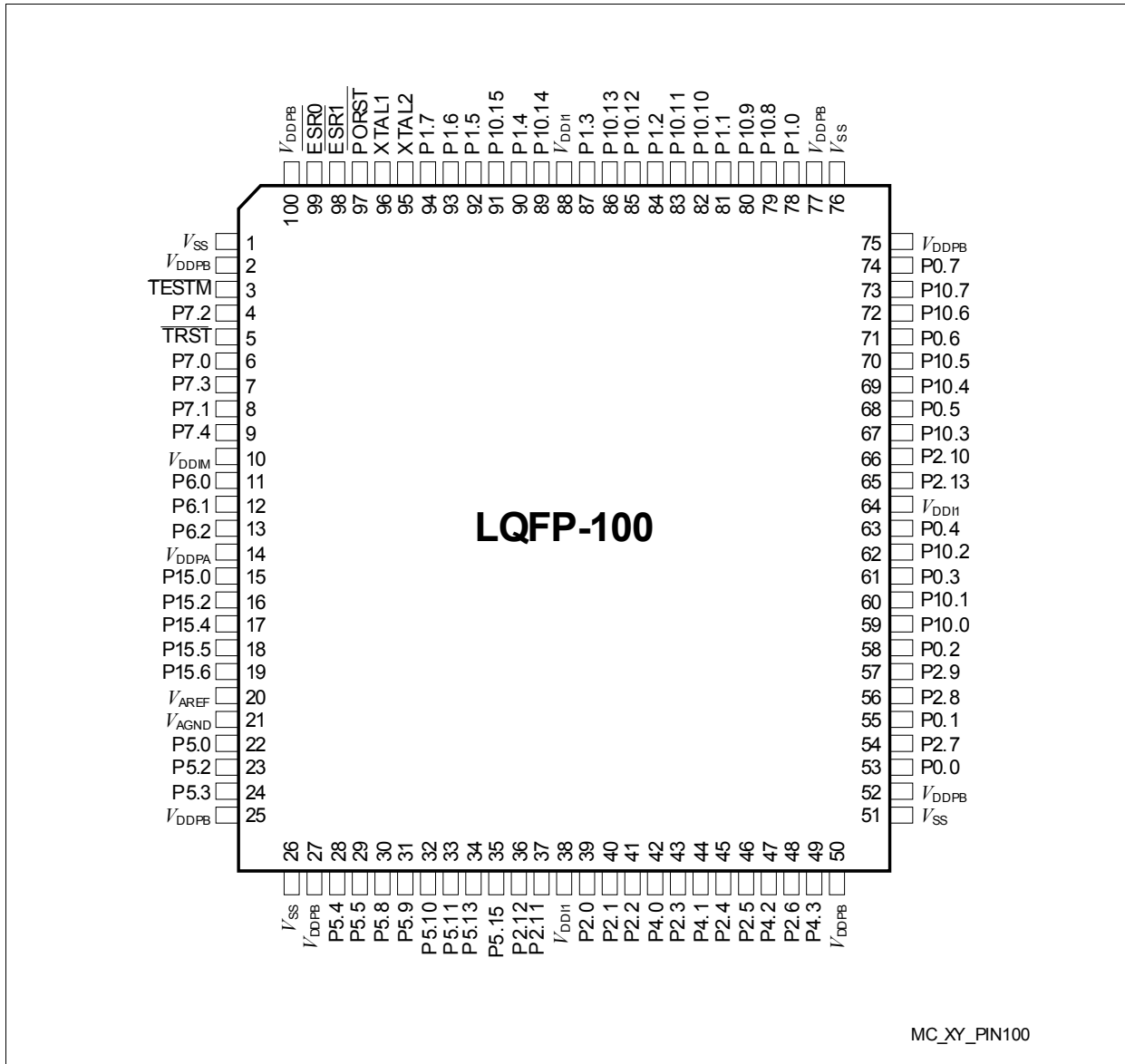
*Note: In any case the specific characteristics of a device should be verified, before a new design is started. This ensures that the used information is up to date.*

The XE16xyM derivatives are shipped in several packages. The following figures show the basic pin diagrams of the XE16xyM. They show the location of the different supply and IO pins. A detailed description of all the pins and their selectable functions can be found in the corresponding Data Sheet.

*Note: Not all packages shown in the figures are supported by all derivatives.  
Please refer to the corresponding descriptions in the data sheets.*

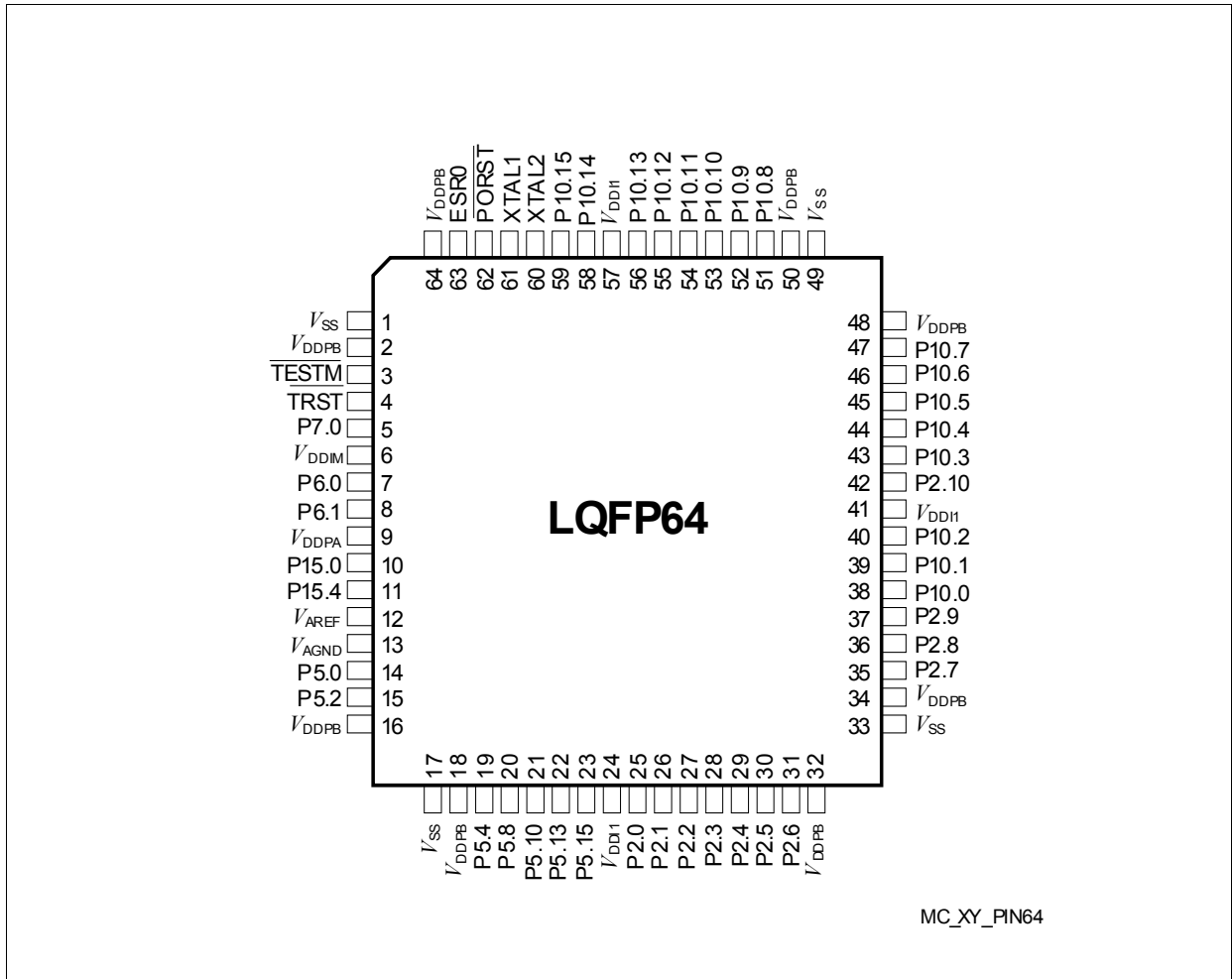


**Figure 15-1 Pin Configuration PG-LQFP-144 Package (top view)**



**Figure 15-2 Pin Configuration PG-LQFP-100 Package (top view)**





**Figure 15-3 Pin Configuration PG-LQFP-64 Package (top view)**

## **16 General Purpose Timer Units**

The General Purpose Timer Unit blocks GPT1 and GPT2 have very flexible multifunctional timer structures which may be used for timing, event counting, pulse width measurement, pulse generation, frequency multiplication, and other purposes.

They incorporate five 16-bit timers that are grouped into the two timer blocks GPT1 and GPT2. Each timer in each block may operate independently in a number of different modes such as gated timer or counter mode, or may be concatenated with another timer of the same block. Each block has alternate input/output functions and specific interrupts associated with it.

*Note: Input signals can be selected from several sources by register PISEL.*

**Block GPT1** contains three timers/counters: The core timer T3 and the two auxiliary timers T2 and T4. The maximum resolution is  $f_{\text{GPT}}/4$ . The auxiliary timers of GPT1 may optionally be configured as reload or capture registers for the core timer. These registers are listed in [Section 16.1.6](#).

- $f_{\text{GPT}}/4$  maximum resolution
- 3 independent timers/counters
- Timers/counters can be concatenated
- 4 operating modes:
  - Timer Mode
  - Gated Timer Mode
  - Counter Mode
  - Incremental Interface Mode
- Reload and Capture functionality
- Separate interrupt lines

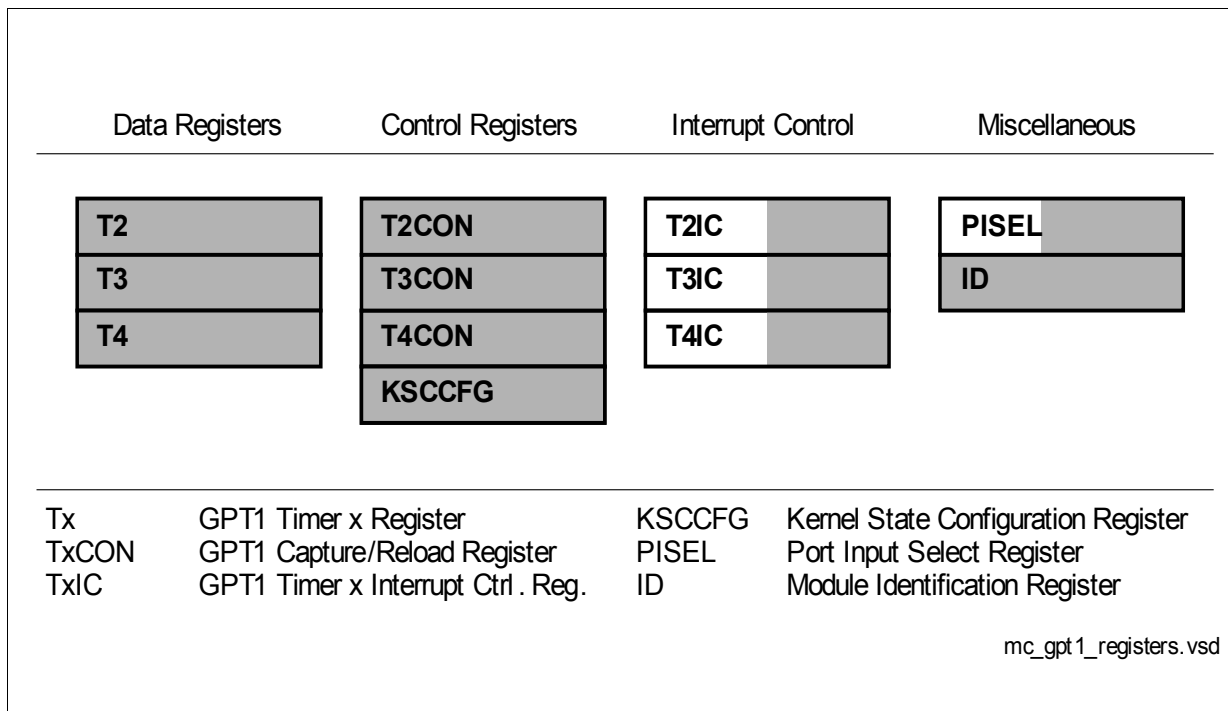
**Block GPT2** contains two timers/counters: The core timer T6 and the auxiliary timer T5. The maximum resolution is  $f_{\text{GPT}}/2$ . An additional Capture/Reload register (CAPREL) supports capture and reload operation with extended functionality. These registers are listed in [Section 16.2.7](#). The core timer T6 may be concatenated with timers of the CAPCOM units (T7 and T8).

The following list summarizes the features which are supported:

- $f_{\text{GPT}}/2$  maximum resolution
- 2 independent timers/counters
- Timers/counters can be concatenated
- 3 operating modes:
  - Timer Mode
  - Gated Timer Mode
  - Counter Mode
- Extended capture/reload functions via 16-bit capture/reload register CAPREL
- Separate interrupt lines

## 16.1 Timer Block GPT1

From a programmer's point of view, the GPT1 block is composed of a set of SFRs as summarized below. Those portions of port and direction registers which are used for alternate functions by the GPT1 block are shaded.



**Figure 16-1 SFRs Associated with Timer Block GPT1**

All three timers of block GPT1 (T2, T3, T4) can run in one of 4 basic modes: Timer Mode, Gated Timer Mode, Counter Mode, or Incremental Interface Mode. All timers can count up or down. Each timer of GPT1 is controlled by a separate control register TxCON.

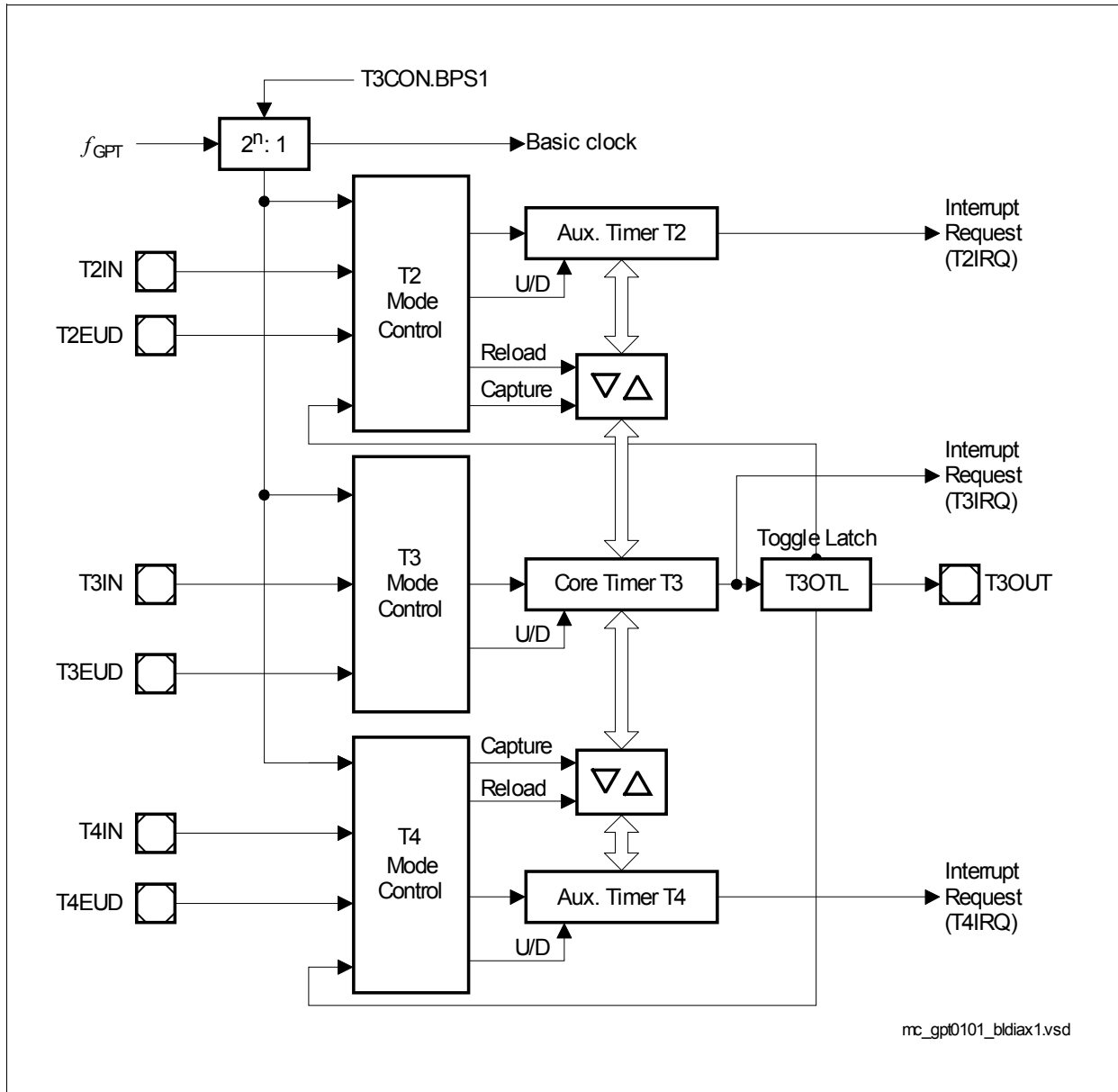
Each timer has an input pin TxIN (alternate pin function) associated with it, which serves as the gate control in gated timer mode, or as the count input in counter mode. The count direction (up/down) may be programmed via software or may be dynamically altered by a signal at the External Up/Down control input TxEUD (alternate pin function). An overflow/underflow of core timer T3 is indicated by the Output Toggle Latch T3OTL, whose state may be output on the associated pin T3OUT (alternate pin function). The auxiliary timers T2 and T4 may additionally be concatenated with the core timer T3 (through T3OTL) or may be used as capture or reload registers for the core timer T3.

The current contents of each timer can be read or modified by the CPU by accessing the corresponding timer count registers T2, T3, or T4, located in the non-bitaddressable SFR space (see [Section 16.1.6](#)). When any of the timer registers is written to by the CPU in the state immediately preceding a timer increment, decrement, reload, or capture operation, the CPU write operation has priority in order to guarantee correct results.

## General Purpose Timer Units

The interrupts of GPT1 are controlled through the Interrupt Control Registers TxIC. These registers are not part of the GPT1 block. The input and output lines of GPT1 are connected to pins of ports P3 and P5. The control registers for the port functions are located in the respective port modules.

*Note: The timing requirements for external input signals can be found in [Section 16.1.5](#), [Section 16.5](#) summarizes the module interface signals, including pins.*



**Figure 16-2 GPT1 Block Diagram (n = 2 ... 5)**

### 16.1.1 GPT1 Core Timer T3 Control

The current contents of the core timer T3 are reflected by its count register T3. This register can also be written to by the CPU, for example, to set the initial start value.

The core timer T3 is configured and controlled via its bitaddressable control register T3CON.

#### GPT12E\_T3CON

**Timer 3 Control Register**

**SFR (FF42<sub>H</sub>/A1<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>T3 R DIR</b>	<b>T3 CH DIR</b>	<b>T3 ED GE</b>	<b>BPS1</b>		<b>T3 OTL</b>	<b>T3 OE</b>	<b>T3 UDE</b>	<b>T3 UD</b>	<b>T3R</b>	<b>T3M</b>			<b>T3I</b>		
rh	rwh	rwh	rw		rwh	rw	rw	rw	rw	rw			rw		

Field	Bits	Type	Description
<b>T3I</b>	[2:0]	rw	<b>Timer T3 Input Parameter Selection</b> Depends on the operating mode, see respective sections for encoding: <a href="#">Table 16-7</a> for Timer Mode and Gated Timer Mode <a href="#">Table 16-2</a> for Counter Mode <a href="#">Table 16-3</a> for Incremental Interface Mode
<b>T3M</b>	[5:3]	rw	<b>Timer T3 Mode Control (Basic Operating Mode)</b> 000 <sub>B</sub> Timer Mode 001 <sub>B</sub> Counter Mode 010 <sub>B</sub> Gated Timer Mode with gate active low 011 <sub>B</sub> Gated Timer Mode with gate active high 100 <sub>B</sub> Reserved. Do not use this combination. 101 <sub>B</sub> Reserved. Do not use this combination. 110 <sub>B</sub> Incremental Interface Mode (Rotation Detection Mode) 111 <sub>B</sub> Incremental Interface Mode (Edge Detection Mode)
<b>T3R</b>	6	rw	<b>Timer T3 Run Bit</b> 0 <sub>B</sub> Timer T3 stops 1 <sub>B</sub> Timer T3 runs
<b>T3UD</b>	7	rw	<b>Timer T3 Up/Down Control<sup>1)</sup></b> 0 <sub>B</sub> Timer T3 counts up 1 <sub>B</sub> Timer T3 counts down

**General Purpose Timer Units**

Field	Bits	Type	Description
<b>T3UDE</b>	8	rw	<b>Timer T3 External Up/Down Enable<sup>1)</sup></b> 0 <sub>B</sub> Input T3EUD is disconnected 1 <sub>B</sub> Direction influenced by input T3EUD
<b>T3OE</b>	9	rw	<b>Overflow/Underflow Output Enable</b> 0 <sub>B</sub> Alternate Output Function Disabled 1 <sub>B</sub> State of T3 toggle latch is output on pin T3OUT
<b>T3OTL</b>	10	rwh	<b>Timer T3 Overflow Toggle Latch</b> Toggles on each overflow/underflow of T3. Can be set or reset by software (see separate description)
<b>BPS1</b>	[12:11]	rw	<b>GPT1 Block Prescaler Control</b> Selects the basic clock for block GPT1 (see also <a href="#">Section 16.1.5</a> ) 00 <sub>B</sub> $f_{GPT}/8$ 01 <sub>B</sub> $f_{GPT}/4$ 10 <sub>B</sub> $f_{GPT}/32$ 11 <sub>B</sub> $f_{GPT}/16$
<b>T3EDGE</b>	13	rwh	<b>Timer T3 Edge Detection Flag</b> The bit is set each time a count edge is detected. T3EDGE must be cleared by SW. 0 <sub>B</sub> No count edge was detected 1 <sub>B</sub> A count edge was detected
<b>T3CHDIR</b>	14	rwh	<b>Timer T3 Count Direction Change Flag</b> This bit is set each time the count direction of timer T3 changes. T3CHDIR must be cleared by SW. 0 <sub>B</sub> No change of count direction was detected 1 <sub>B</sub> A change of count direction was detected
<b>T3RDIR</b>	15	rh	<b>Timer T3 Rotation Direction Flag</b> 0 <sub>B</sub> Timer T3 counts up 1 <sub>B</sub> Timer T3 counts down

1) See [Table 16-1](#) for encoding of bits T3UD and T3UDE.

### Timer T3 Run Control

The core timer T3 can be started or stopped by software through bit T3R (Timer T3 Run Bit). This bit is relevant in all operating modes of T3. Setting bit T3R will start the timer, clearing bit T3R stops the timer.

In gated timer mode, the timer will only run if T3R = 1 and the gate is active (high or low, as programmed).

*Note: When bit T2RC or T4RC in timer control register T2CON or T4CON is set, bit T3R will also control (start and stop) the auxiliary timer(s) T2 and/or T4.*

### Count Direction Control

The count direction of the GPT1 timers (core timer and auxiliary timers) can be controlled either by software or by the external input pin TxEUD (Timer Tx External Up/Down Control Input). These options are selected by bits TxUD and TxUDE in the respective control register TxCON. When the up/down control is provided by software (bit TxUDE = 0), the count direction can be altered by setting or clearing bit TxUD. When bit TxUDE = 1, pin TxEUD is selected to be the controlling source of the count direction. However, bit TxUD can still be used to reverse the actual count direction, as shown in [Table 16-1](#). The count direction can be changed regardless of whether or not the timer is running.

*Note: When pin TxEUD is used as external count direction control input, it must be configured as input.*

**Table 16-1 GPT1 Timer Count Direction Control**

Pin TxEUD	Bit TxUDE	Bit TxUD	Count Direction	Bit TxRDIR
X	0	0	Count Up	0
X	0	1	Count Down	1
0	1	0	Count Up	0
1	1	0	Count Down	1
0	1	1	Count Down	1
1	1	1	Count Up	0

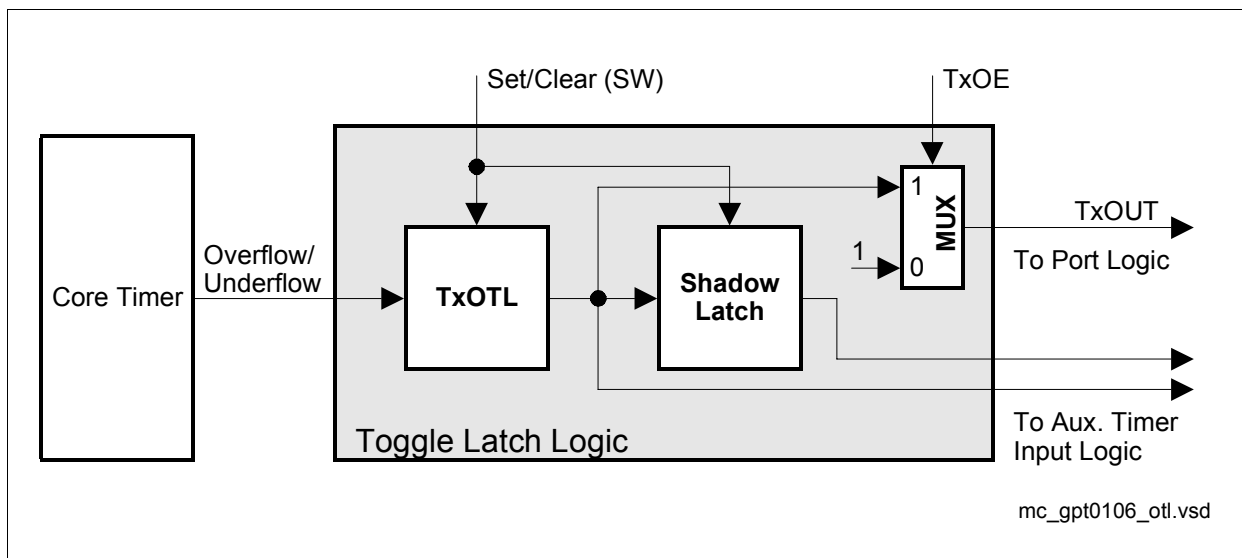
### Timer 3 Output Toggle Latch

The overflow/underflow signal of timer T3 is connected to a block named 'Toggle Latch', shown in the timer mode diagrams. **Figure 16-3** illustrates the details of this block. An overflow or underflow of T3 will clock two latches: The first latch represents bit T3OTL in control register T3CON. The second latch is an internal latch toggled by T3OTL's output. Both latch outputs are connected to the input control blocks of the auxiliary timers T2 and T4. The output level of the shadow latch will match the output level of T3OTL, but is delayed by one clock cycle. When the T3OTL value changes, this will result in a temporarily different output level from T3OTL and the shadow latch, which can trigger the selected count event in T2 and/or T4.

When software writes to T3OTL, both latches are set or cleared simultaneously. In this case, both signals to the auxiliary timers carry the same level and no edge will be detected. Bit T3OE (overflow/underflow output enable) in register T3CON enables the state of T3OTL to be monitored via an external pin T3OUT. When T3OTL is linked to an external port pin (must be configured as output), T3OUT can be used to control external HW. If T3OE = 1, pin T3OUT outputs the state of T3OTL. If T3OE = 0, pin T3OUT outputs a high level (as long as the T3OUT alternate function is selected for the port pin).

The trigger signals can serve as an input for the counter function or as a trigger source for the reload function of the auxiliary timers T2 and T4.

As can be seen from **Figure 16-3**, when latch T3OTL is modified by software to determine the state of the output line, also the internal shadow latch is set or cleared accordingly. Therefore, no trigger condition is detected by T2/T4 in this case.



**Figure 16-3 Block Diagram of the Toggle Latch Logic of Core Timer T3**

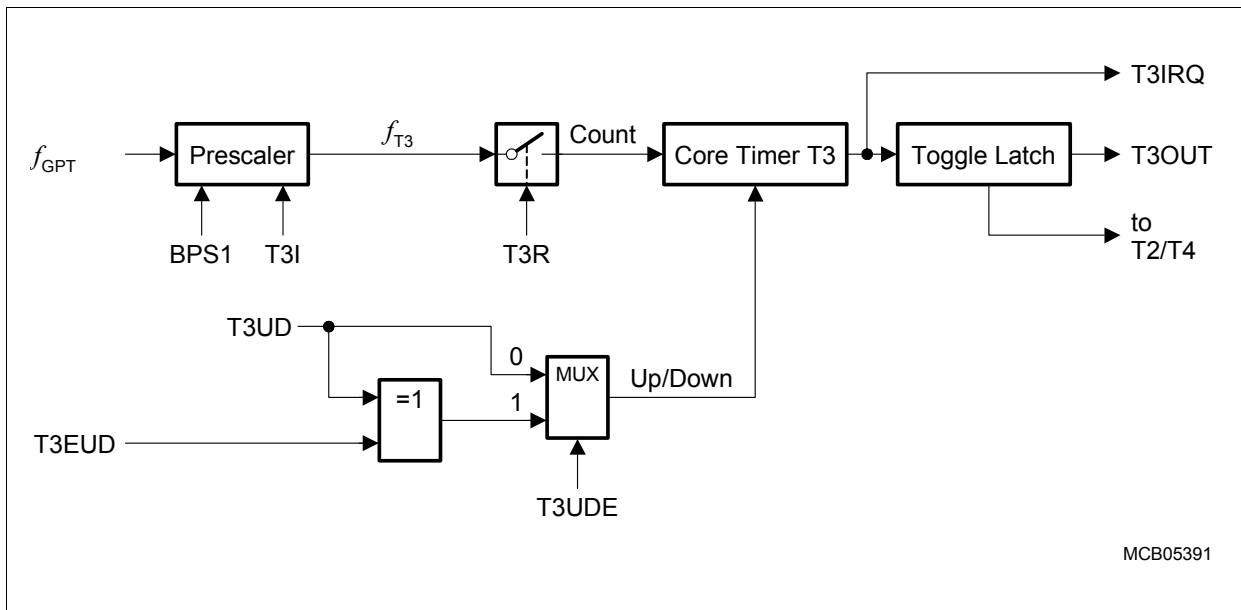


### 16.1.2 GPT1 Core Timer T3 Operating Modes

Timer T3 can operate in one of several modes.

#### Timer 3 in Timer Mode

Timer mode for the core timer T3 is selected by setting bitfield T3M in register T3CON to 000<sub>B</sub>. In timer mode, T3 is clocked with the module's input clock  $f_{GPT}$  divided by two programmable prescalars controlled by bitfields BPS1 and T3I in register T3CON. Please see [Section 16.1.5](#) for details on the input clock options.

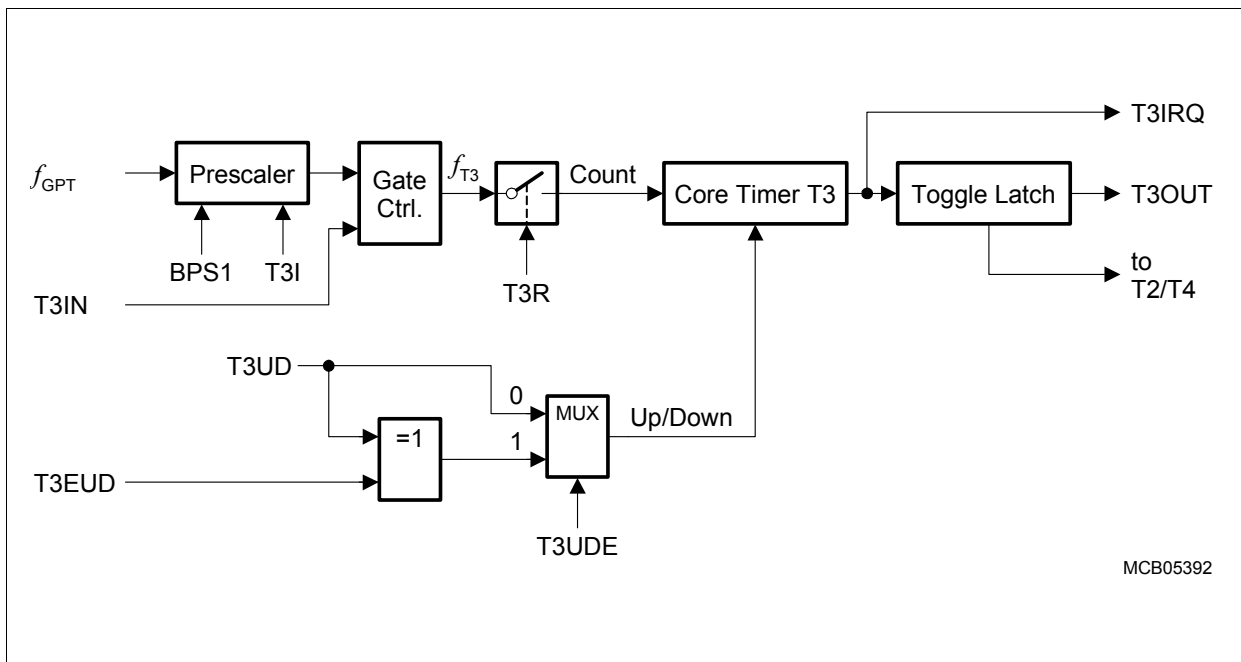


**Figure 16-4 Block Diagram of Core Timer T3 in Timer Mode**

### Gated Timer Mode

Gated timer mode for the core timer T3 is selected by setting bitfield T3M in register T3CON to 010<sub>B</sub> or 011<sub>B</sub>. Bit T3M.0 (T3CON.3) selects the active level of the gate input. The same options for the input frequency are available in gated timer mode as in timer mode (see [Section 16.1.5](#)). However, the input clock to the timer in this mode is gated by the external input pin T3IN (Timer T3 External Input).

To enable this operation, the associated pin T3IN must be configured as input.



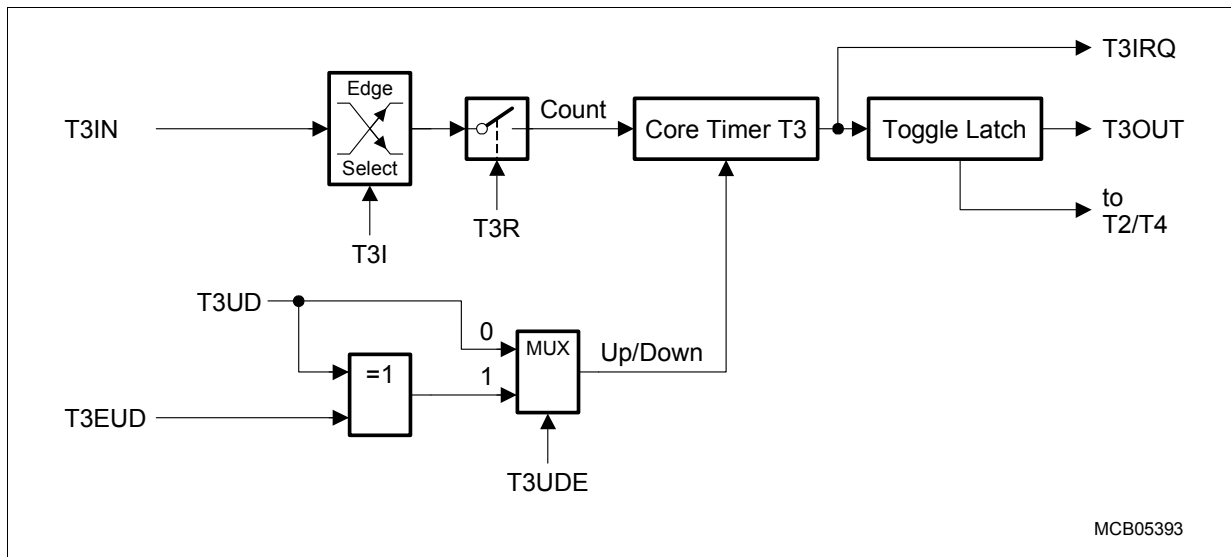
**Figure 16-5 Block Diagram of Core Timer T3 in Gated Timer Mode**

If T3M = 010<sub>B</sub>, the timer is enabled when T3IN shows a low level. A high level at this line stops the timer. If T3M = 011<sub>B</sub>, line T3IN must have a high level in order to enable the timer. Additionally, the timer can be turned on or off by software using bit T3R. The timer will only run if T3R is 1 and the gate is active. It will stop if either T3R is 0 or the gate is inactive.

*Note: A transition of the gate signal at pin T3IN does not cause an interrupt request.*

## Counter Mode

Counter Mode for the core timer T3 is selected by setting bitfield T3M in register T3CON to 001<sub>B</sub>. In counter mode, timer T3 is clocked by a transition at the external input pin T3IN. The event causing an increment or decrement of the timer can be a positive, a negative, or both a positive and a negative transition at this line. Bitfield T3I in control register T3CON selects the triggering transition (see [Table 16-2](#)).



**Figure 16-6 Block Diagram of Core Timer T3 in Counter Mode**

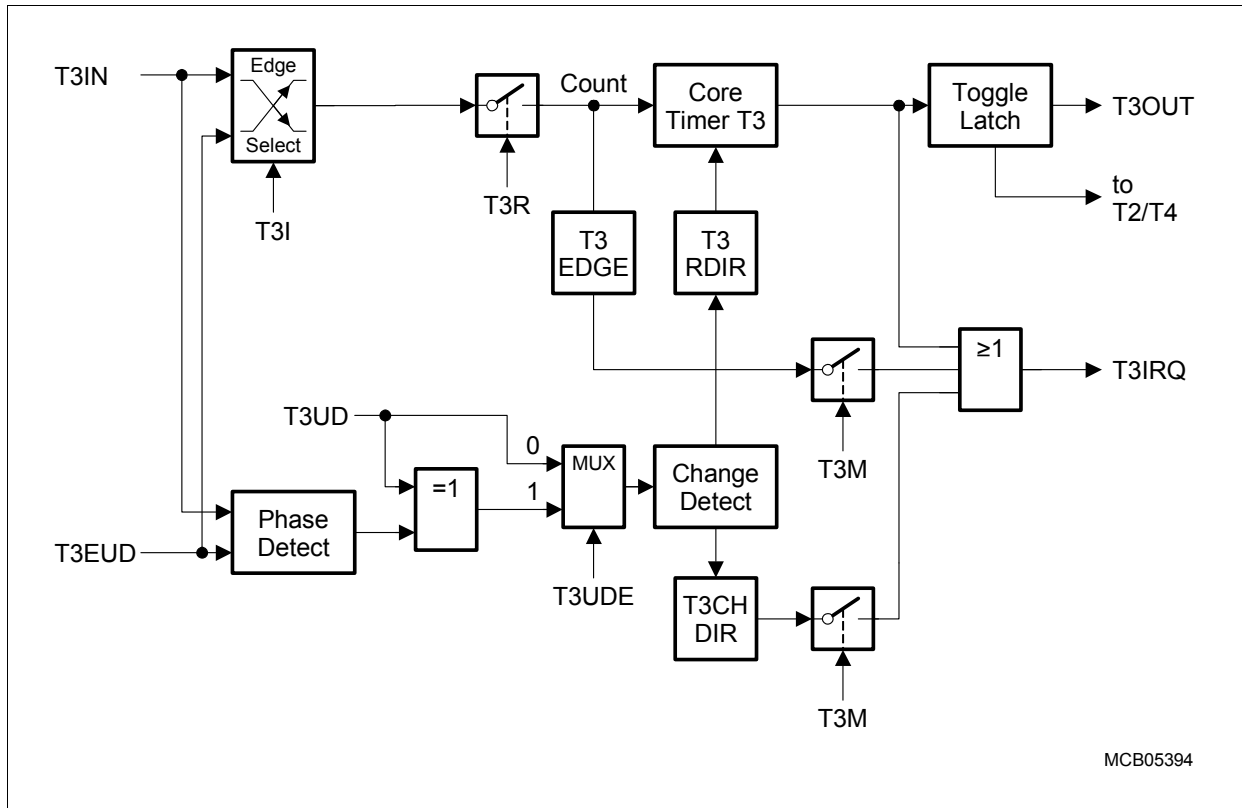
**Table 16-2 GPT1 Core Timer T3 (Counter Mode) Input Edge Selection**

T3I	Triggering Edge for Counter Increment/Decrement
000 <sub>B</sub>	None. Counter T3 is disabled
001 <sub>B</sub>	Positive transition (rising edge) on T3IN
010 <sub>B</sub>	Negative transition (falling edge) on T3IN
011 <sub>B</sub>	Any transition (rising or falling edge) on T3IN
1XX <sub>B</sub>	Reserved. Do not use this combination

For counter mode operation, pin T3IN must be configured as input. The maximum input frequency allowed in counter mode depends on the selected prescaler value. To ensure that a transition of the count input signal applied to T3IN is recognized correctly, its level must be held high or low for a minimum number of module clock cycles before it changes. This information can be found in [Section 16.1.5](#).

### Incremental Interface Mode

Incremental interface mode for the core timer T3 is selected by setting bitfield T3M in register T3CON to 110<sub>B</sub> or 111<sub>B</sub>. In incremental interface mode, the two inputs associated with core timer T3 (T3IN, T3EUD) are used to interface to an incremental encoder. T3 is clocked by each transition on one or both of the external input pins to provide 2-fold or 4-fold resolution of the encoder input.



**Figure 16-7 Block Diagram of Core Timer T3 in Incremental Interface Mode**

Bitfield T3I in control register T3CON selects the triggering transitions (see [Table 16-3](#)). The sequence of the transitions of the two input signals is evaluated and generates count pulses as well as the direction signal. So T3 is modified automatically according to the speed and the direction of the incremental encoder and, therefore, its contents always represent the encoder's current position.

The interrupt request (T3IRQ) generation mode can be selected: In Rotation Detection Mode (T3M = 110<sub>B</sub>), an interrupt request is generated each time the count direction of T3 changes. In Edge Detection Mode (T3M = 111<sub>B</sub>), an interrupt request is generated each time a count edge for T3 is detected. Count direction, changes in the count direction, and count requests are monitored by status bits T3RDIR, T3CHDIR, and T3EDGE in register T3CON.

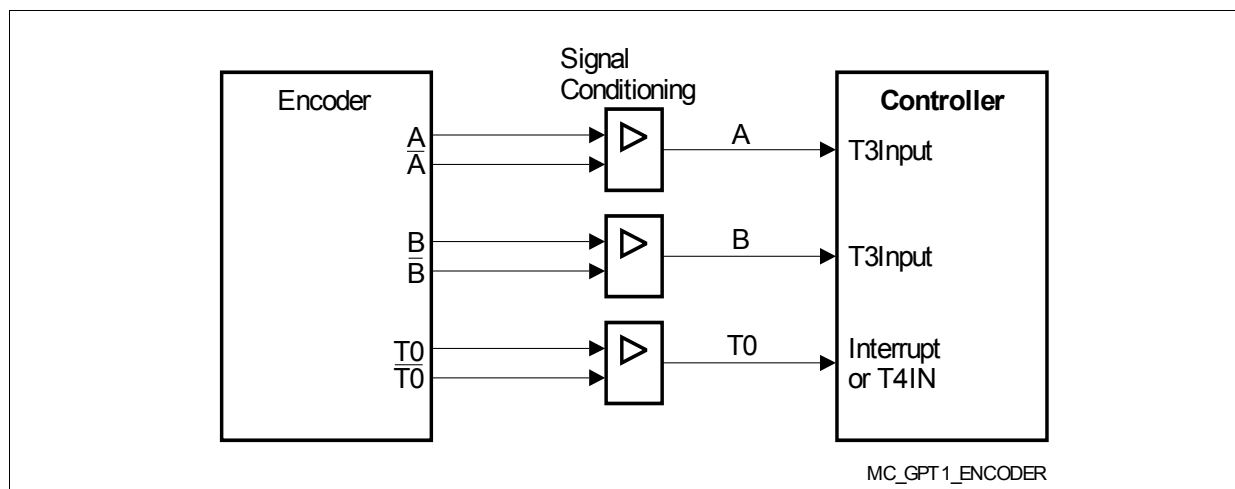
**Table 16-3 Core Timer T3 (Incremental Interface Mode) Input Edge Selection**

<b>T3I</b>	<b>Triggering Edge for Counter Increment/Decrement</b>
000 <sub>B</sub>	None. Counter T3 stops.
001 <sub>B</sub>	Any transition (rising or falling edge) on T3IN.
010 <sub>B</sub>	Any transition (rising or falling edge) on T3EUD.
011 <sub>B</sub>	Any transition (rising or falling edge) on any T3 input (T3IN or T3EUD).
1XX <sub>B</sub>	Reserved. Do not use this combination.

The incremental encoder can be connected directly to the XE16xyM without external interface logic. In a standard system, however, comparators will be employed to convert the encoder's differential outputs (such as A,  $\overline{A}$ ) to digital signals (such as A). This greatly increases noise immunity.

*Note: The third encoder output T0, which indicates the mechanical zero position, may be connected to an external interrupt input and trigger a reset of timer T3 (for example via PEC transfer from ZEROS).*

*If input T4IN is available, T0 can be connected there and clear T3 automatically without requiring an interrupt.*



**Figure 16-8 Connection of the Encoder to the XE16xyM**

For incremental interface operation, the following conditions must be met:

- Bitfield T3M must be 110<sub>B</sub> or 111<sub>B</sub>.
- Both pins T3IN and T3EUD must be configured as input.
- Pin T4IN must be configured as input, if used for T0.
- Bit T3UDE must be 1 to enable automatic external direction control.

The maximum count frequency allowed in incremental interface mode depends on the selected prescaler value. To ensure that a transition of any input signal is recognized

### General Purpose Timer Units

correctly, its level must be held high or low for a minimum number of module clock cycles before it changes. This information can be found in [Section 16.1.5](#).

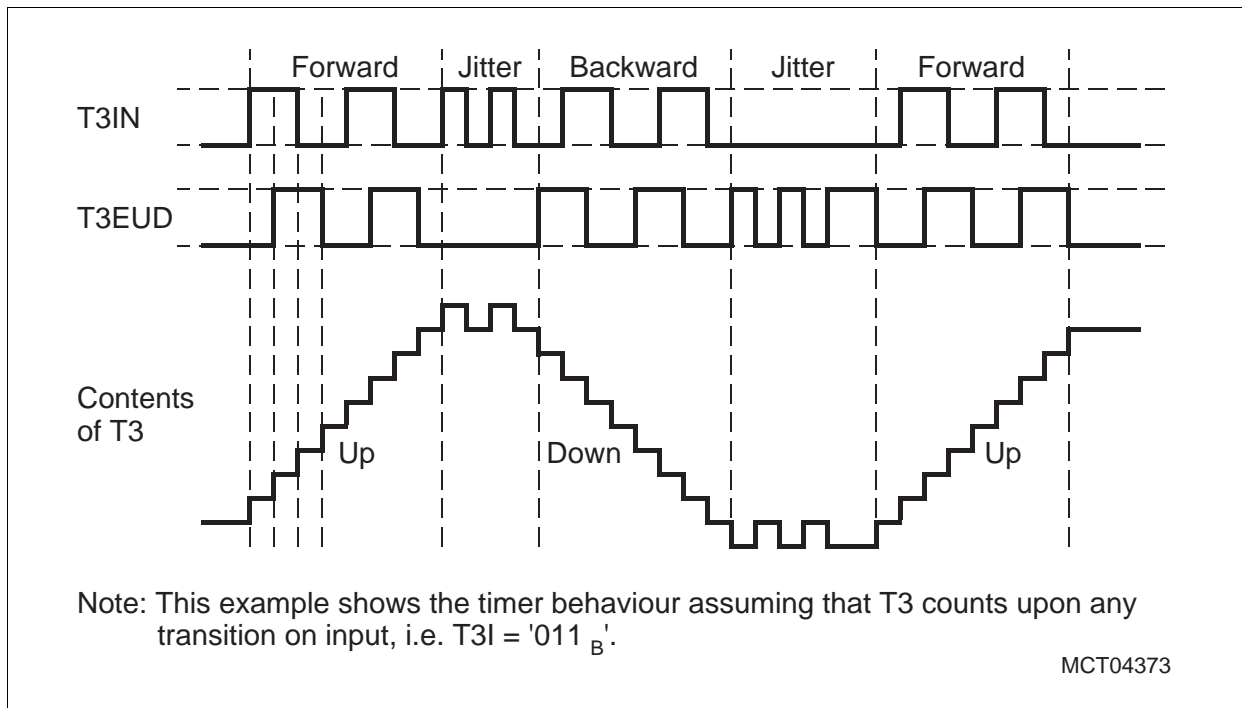
As in incremental interface mode two input signals with a 90° phase shift are evaluated, their maximum input frequency can be half the maximum count frequency.

In incremental interface mode, the count direction is automatically derived from the sequence in which the input signals change, which corresponds to the rotation direction of the connected sensor. [Table 16-4](#) summarizes the possible combinations.

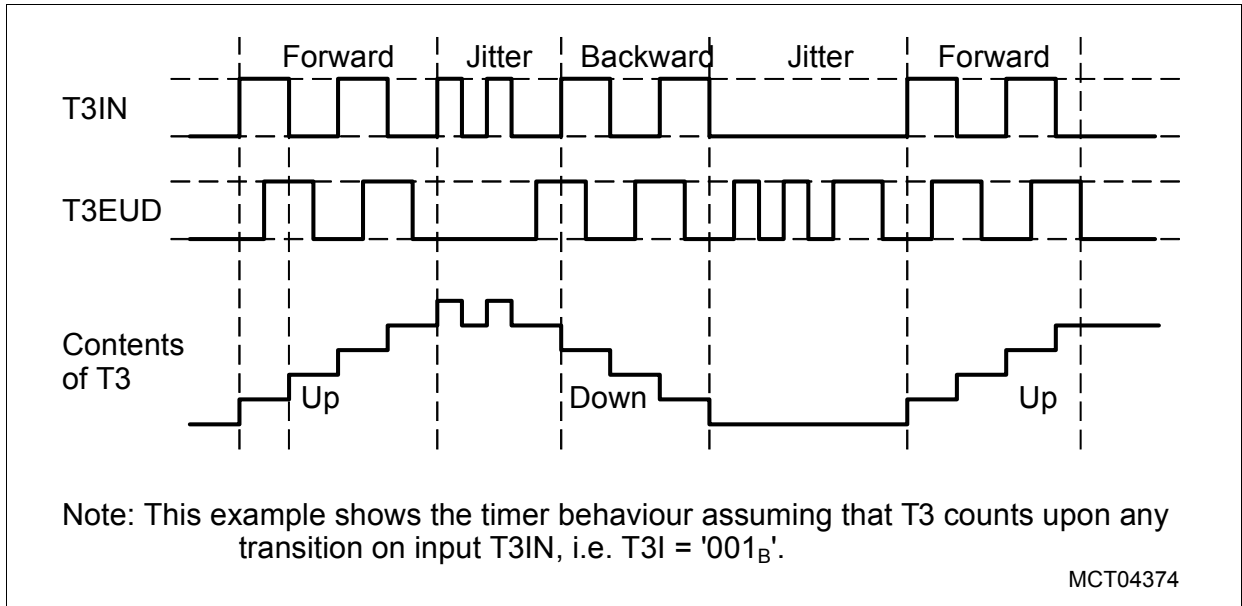
**Table 16-4 GPT1 Core Timer T3 (Incremental Interface Mode) Count Direction**

Level on Respective other Input	T3IN Input		T3EUD Input	
	Rising ↑	Falling ↓	Rising ↑	Falling ↓
High	Down	Up	Up	Down
Low	Up	Down	Down	Up

[Figure 16-9](#) and [Figure 16-10](#) give examples of T3's operation, visualizing count signal generation and direction control. They also show how input jitter is compensated, which might occur if the sensor rests near to one of its switching points.



**Figure 16-9 Evaluation of Incremental Encoder Signals, 2 Count Inputs**



**Figure 16-10 Evaluation of Incremental Encoder Signals, 1 Count Input**

*Note: Timer T3 operating in incremental interface mode automatically provides information on the sensor's current position. Dynamic information (speed, acceleration, deceleration) may be obtained by measuring the incoming signal periods (see ["Combined Capture Modes" on Page 16-55](#)).*

### 16.1.3 GPT1 Auxiliary Timers T2/T4 Control

Auxiliary timers T2 and T4 have exactly the same functionality. They can be configured for timer mode, gated timer mode, counter mode, or incremental interface mode with the same options for the timer frequencies and the count signal as the core timer T3. In addition to these 4 counting modes, the auxiliary timers can be concatenated with the core timer, or they may be used as reload or capture registers in conjunction with the core timer. The start/stop function of the auxiliary timers can be remotely controlled by the T3 run control bit. Several timers may thus be controlled synchronously.

The current contents of an auxiliary timer are reflected by its count register T2 or T4, respectively. These registers can also be written to by the CPU, for example, to set the initial start value.

The individual configurations for timers T2 and T4 are determined by their bitaddressable control registers T2CON and T4CON, which are organized identically. Note that functions which are present in all 3 timers of block GPT1 are controlled in the same bit positions and in the same manner in each of the specific control registers.

*Note: The auxiliary timers have no output toggle latch and no alternate output function.*

#### GPT12E\_T2CON

**Timer 2 Control Register**

**SFR (FF40<sub>H</sub>/A0<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>T2 R DIR</b>	<b>T2 CH DIR</b>	<b>T2 ED GE</b>	<b>T2 IR DIS</b>	-	-	<b>T2 RC</b>	<b>T2 UDE</b>	<b>T2 UD</b>	<b>T2R</b>	<b>T2M</b>			<b>T2I</b>		
rh	rwh	rwh	rw	-	-	rw	rw	rw	rw	rw			rw		

Field	Bits	Type	Description
<b>T2I</b>	[2:0]	rw	<b>Timer T2 Input Parameter Selection</b> Depends on the operating mode, see respective sections for encoding: <a href="#">Table 16-7</a> for Timer Mode and Gated Timer Mode <a href="#">Table 16-2</a> for Counter Mode <a href="#">Table 16-3</a> for Incremental Interface Mode



**General Purpose Timer Units**

Field	Bits	Type	Description
<b>T2M</b>	[5:3]	rw	<b>Timer T2 Mode Control</b> (Basic Operating Mode) 000 <sub>B</sub> Timer Mode 001 <sub>B</sub> Counter Mode 010 <sub>B</sub> Gated Timer Mode with gate active low 011 <sub>B</sub> Gated Timer Mode with gate active high 100 <sub>B</sub> Reload Mode 101 <sub>B</sub> Capture Mode 110 <sub>B</sub> Incremental Interface Mode (Rotation Detect) 111 <sub>B</sub> Incremental Interface Mode (Edge Detection)
<b>T2R</b>	6	rw	<b>Timer T2 Run Bit</b> 0 <sub>B</sub> Timer T2 stops 1 <sub>B</sub> Timer T2 runs <i>Note: This bit only controls timer T2 if bit T2RC = 0.</i>
<b>T2UD</b>	7	rw	<b>Timer T2 Up/Down Control<sup>1)</sup></b> 0 <sub>B</sub> Timer T2 counts up 1 <sub>B</sub> Timer T2 counts down
<b>T2UDE</b>	8	rw	<b>Timer T2 External Up/Down Enable<sup>1)</sup></b> 0 <sub>B</sub> Input T2EUD is disconnected 1 <sub>B</sub> Direction influenced by input T2EUD
<b>T2RC</b>	9	rw	<b>Timer T2 Remote Control</b> 0 <sub>B</sub> Timer T2 is controlled by its own run bit T2R 1 <sub>B</sub> Timer T2 is controlled by the run bit T3R of core timer T3, not by bit T2R
<b>T2IRDIS</b>	12	rw	<b>Timer T2 Interrupt Request Disable</b> 0 <sub>B</sub> Interrupt generation for T2CHDIR and T2EDGE interrupts in Incremental Interface Mode is enabled 1 <sub>B</sub> Interrupt generation for T2CHDIR and T2EDGE interrupts in Incremental Interface Mode is disabled
<b>T2EDGE</b>	13	rwh	<b>Timer T2 Edge Detection</b> The bit is set each time a count edge is detected. T2EDGE must be cleared by SW. 0 <sub>B</sub> No count edge was detected 1 <sub>B</sub> A count edge was detected

**General Purpose Timer Units**

Field	Bits	Type	Description
<b>T2CHDIR</b>	14	rwh	<b>Timer T2 Count Direction Change</b> This bit is set each time the count direction of timer T4 changes. T4CHDIR must be cleared by SW. 0 <sub>B</sub> No change in count direction was detected 1 <sub>B</sub> A change in count direction was detected
<b>T2RDIR</b>	15	rh	<b>Timer T2 Rotation Direction</b> 0 <sub>B</sub> Timer T2 counts up 1 <sub>B</sub> Timer T2 counts down

1) See [Table 16-1](#) for encoding of bits T2UD and T2UDE.

**GPT12E\_T4CON**

**Timer 4 Control Register**

**SFR (FF44<sub>H</sub>/A2<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>T4 R DIR</b>	<b>T4 CH DIR</b>	<b>T4 ED GE</b>	<b>T4 IR DIS</b>	<b>CLR T3 EN</b>	<b>CLR T2 EN</b>	<b>T4 RC</b>	<b>T4 UDE</b>	<b>T4 UD</b>	<b>T4R</b>	<b>T4M</b>			<b>T4I</b>		
rh	rwh	rwh	rw	-	-	rw	rw	rw	rw	rw			rw		

Field	Bits	Type	Description
<b>T4I</b>	[2:0]	rw	<b>Timer T4 Input Parameter Selection</b> Depends on the operating mode, see respective sections for encoding: <a href="#">Table 16-7</a> for Timer Mode and Gated Timer Mode <a href="#">Table 16-2</a> for Counter Mode <a href="#">Table 16-3</a> for Incremental Interface Mode
<b>T4M</b>	[5:3]	rw	<b>Timer T4 Mode Control (Basic Operating Mode)</b> 000 <sub>B</sub> Timer Mode 001 <sub>B</sub> Counter Mode 010 <sub>B</sub> Gated Timer Mode with gate active low 011 <sub>B</sub> Gated Timer Mode with gate active high 100 <sub>B</sub> Reload Mode 101 <sub>B</sub> Capture Mode 110 <sub>B</sub> Incremental Interface Mode (Rotation Detect.) 111 <sub>B</sub> Incremental Interface Mode (Edge Detection)

**General Purpose Timer Units**

Field	Bits	Type	Description
<b>T4R</b>	6	rw	<b>Timer T4 Run Bit</b> 0 <sub>B</sub> Timer T4 stops 1 <sub>B</sub> Timer T4 runs <i>Note: This bit only controls timer T4 if bit T4RC = 0.</i>
<b>T4UD</b>	7	rw	<b>Timer T4 Up/Down Control<sup>1)</sup></b> 0 <sub>B</sub> Timer T4 counts up 1 <sub>B</sub> Timer T4 counts down
<b>T4UDE</b>	8	rw	<b>Timer T4 External Up/Down Enable<sup>1)</sup></b> 0 <sub>B</sub> Input T4EUD is disconnected 1 <sub>B</sub> Direction influenced by input T4EUD
<b>T4RC</b>	9	rw	<b>Timer T4 Remote Control</b> 0 <sub>B</sub> Timer T4 is controlled by its own run bit T4R 1 <sub>B</sub> Timer T4 is controlled by the run bit T3R of core timer T3, not by bit T4R
<b>CLRT2EN</b>	10	rw	<b>Clear Timer 2 Enable</b> Enables the automatic clearing of T2 upon a falling edge of the selected T4EUD input. 0 <sub>B</sub> No effect of T4EUD on T2 1 <sub>B</sub> A falling edge on T4EUD clears timer T2
<b>CLRT3EN</b>	11	rw	<b>Clear Timer 3 Enable</b> Enables the automatic clearing of T3 upon a falling edge of the selected T4IN input. 0 <sub>B</sub> No effect of T4IN on T3 1 <sub>B</sub> A falling edge on T4IN clears timer T3
<b>T4IRDIS</b>	12	rw	<b>Timer T4 Interrupt Request Disable</b> 0 <sub>B</sub> Interrupt generation for T4CHDIR and T4EDGE interrupts in Incremental Interface Mode is enabled 1 <sub>B</sub> Interrupt generation for T4CHDIR and T4EDGE interrupts in Incremental Interface Mode is disabled
<b>T4EDGE</b>	13	rwh	<b>Timer T4 Edge Detection</b> The bit is set each time a count edge is detected. T4EDGE must be cleared by SW. 0 <sub>B</sub> No count edge was detected 1 <sub>B</sub> A count edge was detected

**General Purpose Timer Units**

Field	Bits	Type	Description
<b>T4CHDIR</b>	14	rwh	<b>Timer T4 Count Direction Change</b> This bit is set each time the count direction of timer T4 changes. T4CHDIR must be cleared by SW. 0 <sub>B</sub> No change in count direction was detected 1 <sub>B</sub> A change in count direction was detected
<b>T4RDIR</b>	15	rh	<b>Timer T4 Rotation Direction</b> 0 <sub>B</sub> Timer T4 counts up 1 <sub>B</sub> Timer T4 counts down

1) See [Table 16-1](#) for encoding of bits T4UD and T4UDE.

### Timer T2/T4 Run Control

Each of the auxiliary timers T2 and T4 can be started or stopped by software in two different ways:

- Through the associated timer run bit (T2R or T4R). In this case it is required that the respective control bit TxRC = 0.
- Through the core timer's run bit (T3R). In this case the respective remote control bit must be set (TxRC = 1).

The selected run bit is relevant in all operating modes of T2/T4. Setting the bit will start the timer, clearing the bit stops the timer.

In gated timer mode, the timer will only run if the selected run bit is set and the gate is active (high or low, as programmed).

*Note: If remote control is selected T3R will start/stop timer T3 and the selected auxiliary timer(s) synchronously.*

### Count Direction Control

The count direction of the GPT1 timers (core timer and auxiliary timers) is controlled in the same way, either by software or by the external input pin TxEUD. Please refer to the description in [Table 16-1](#).

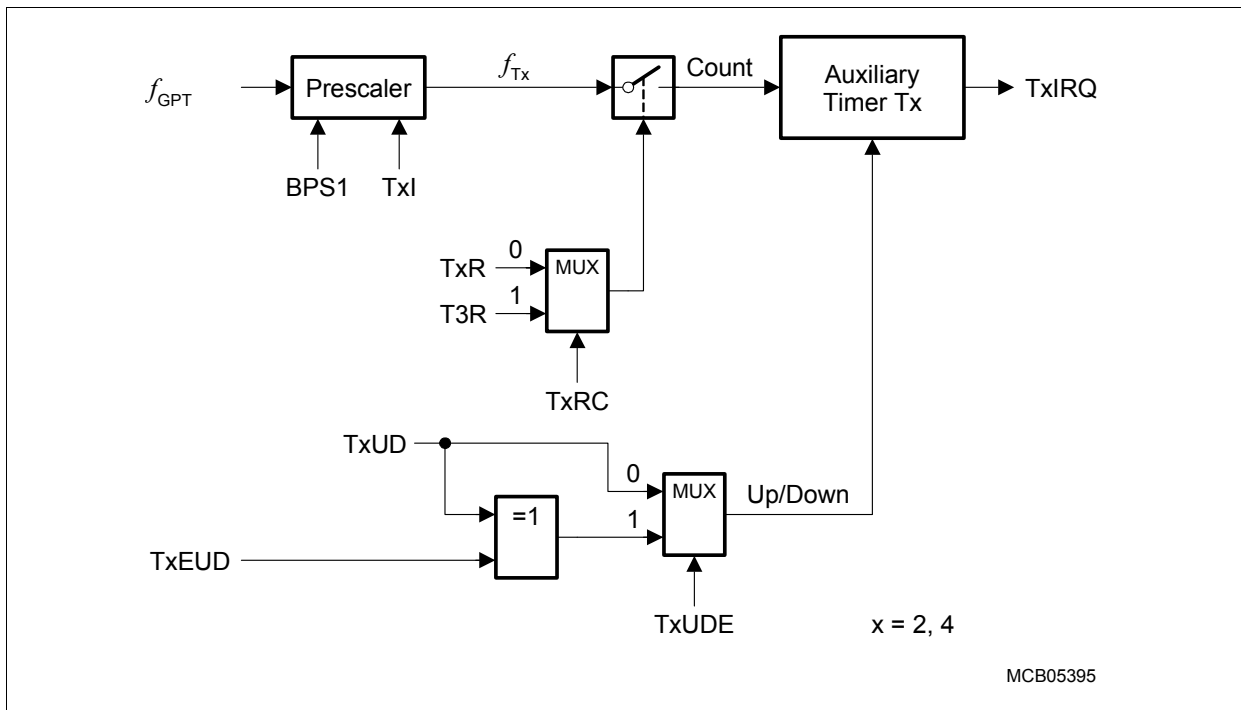
*Note: When pin TxEUD is used as external count direction control input, it must be configured as input.*

### 16.1.4 GPT1 Auxiliary Timers T2/T4 Operating Modes

The operation of the auxiliary timers in the basic operating modes is almost identical with the core timer's operation, with very few exceptions. Additionally, some combined operating modes can be selected.

#### Timers T2 and T4 in Timer Mode

Timer mode for an auxiliary timer Tx is selected by setting its bitfield TxM in register TxCON to 000<sub>B</sub>.

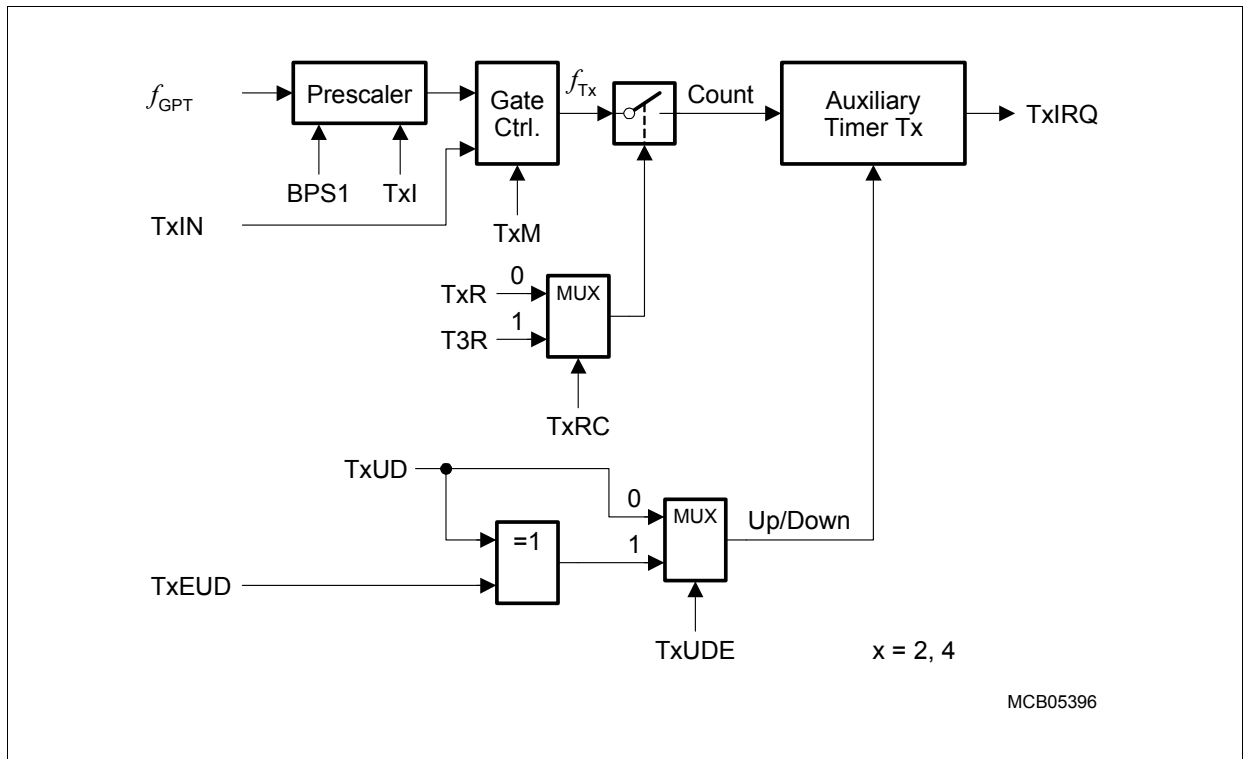


**Figure 16-11 Block Diagram of an Auxiliary Timer in Timer Mode**

### Timers T2 and T4 in Gated Timer Mode

Gated timer mode for an auxiliary timer Tx is selected by setting bitfield TxM in register TxCON to 010<sub>B</sub> or 011<sub>B</sub>. Bit TxM.0 (TxCON.3) selects the active level of the gate input.

*Note: A transition of the gate signal at line TxIN does not cause an interrupt request.*



**Figure 16-12 Block Diagram of an Auxiliary Timer in Gated Timer Mode**

*Note: There is no output toggle latch for T2 and T4.*

*Start/stop of an auxiliary timer can be controlled locally or remotely.*

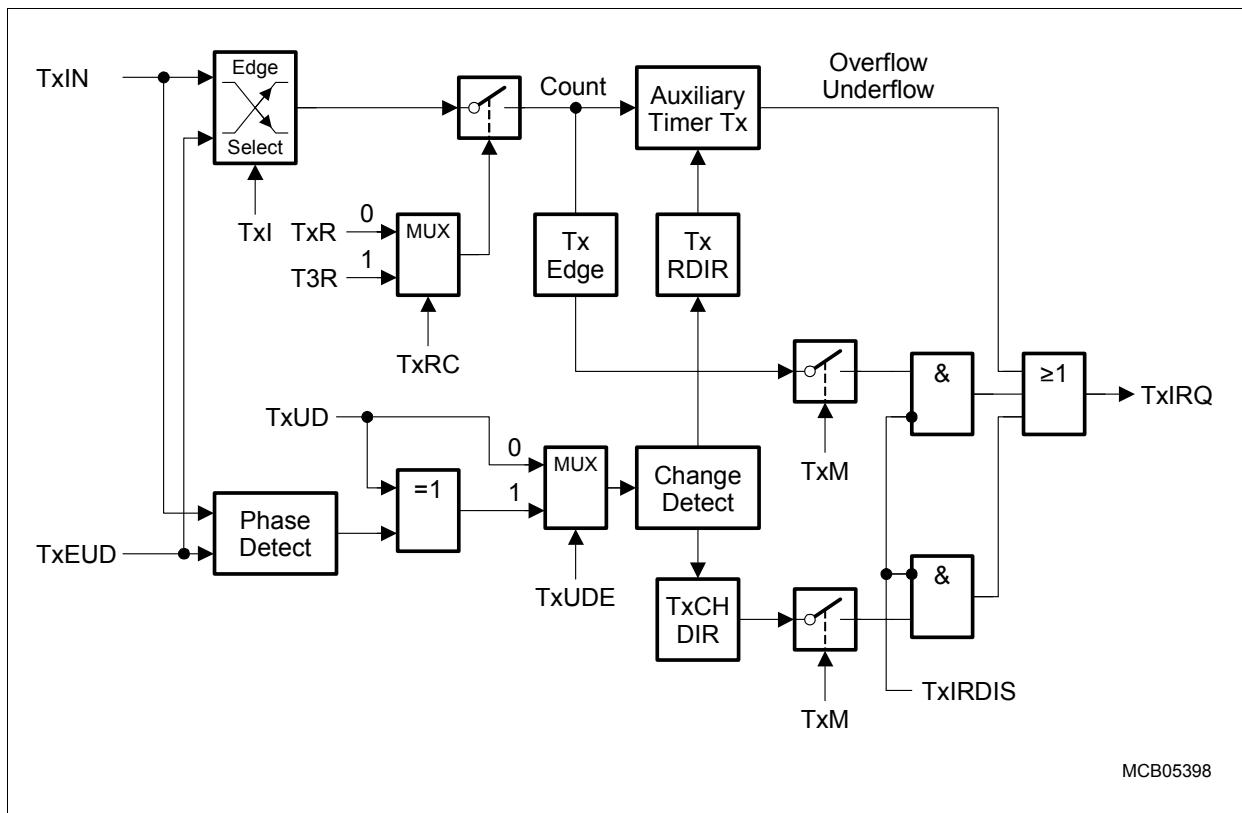


## General Purpose Timer Units

For counter operation, pin TxIN must be configured as input. The maximum input frequency allowed in counter mode depends on the selected prescaler value. To ensure that a transition of the count input signal applied to TxIN is recognized correctly, its level must be held high or low for a minimum number of module clock cycles before it changes. This information can be found in [Section 16.1.5](#).

### Timers T2 and T4 in Incremental Interface Mode

Incremental interface mode for an auxiliary timer Tx is selected by setting bitfield TxM in the respective register TxCON to 110<sub>B</sub> or 111<sub>B</sub>. In incremental interface mode, the two inputs associated with an auxiliary timer Tx (TxIN, TxEUD) are used to interface to an incremental encoder. Tx is clocked by each transition on one or both of the external input pins to provide 2-fold or 4-fold resolution of the encoder input.



**Figure 16-14 Block Diagram of an Auxiliary Timer in Incremental Interface Mode**

The operation of the auxiliary timers T2 and T4 in incremental interface mode and the interrupt generation are the same as described for the core timer T3. The descriptions, figures and tables apply accordingly.

*Note: Timers T2 and T4 operating in incremental interface mode automatically provide information on the sensor's current position. For dynamic information (speed, acceleration, deceleration) see ["Combined Capture Modes" on Page 16-55](#).*



### Timer Concatenation

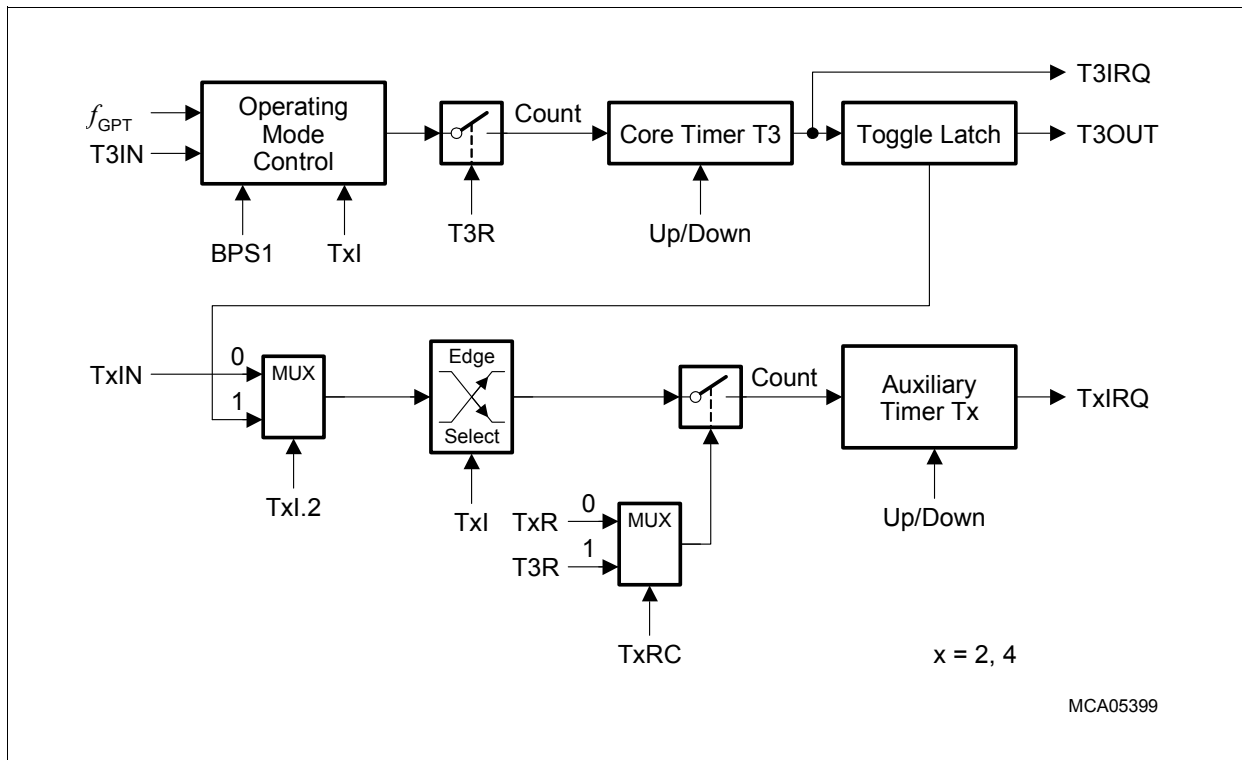
Using the toggle bit T3OTL as a clock source for an auxiliary timer in counter mode concatenates the core timer T3 with the respective auxiliary timer. This concatenation forms either a 32-bit or a 33-bit timer/counter, depending on which transition of T3OTL is selected to clock the auxiliary timer.

- **32-bit Timer/Counter:** If both a positive and a negative transition of T3OTL are used to clock the auxiliary timer, this timer is clocked on every overflow/underflow of the core timer T3. Thus, the two timers form a 32-bit timer.
- **33-bit Timer/Counter:** If either a positive or a negative transition of T3OTL is selected to clock the auxiliary timer, this timer is clocked on every second overflow/underflow of the core timer T3. This configuration forms a 33-bit timer (16-bit core timer + T3OTL + 16-bit auxiliary timer).

As long as bit T3OTL is not modified by software, it represents the state of the internal toggle latch, and can be regarded as part of the 33-bit timer.

The count directions of the two concatenated timers are not required to be the same. This offers a wide variety of different configurations.

T3, which represents the low-order part of the concatenated timer, can operate in timer mode, gated timer mode or counter mode in this case.



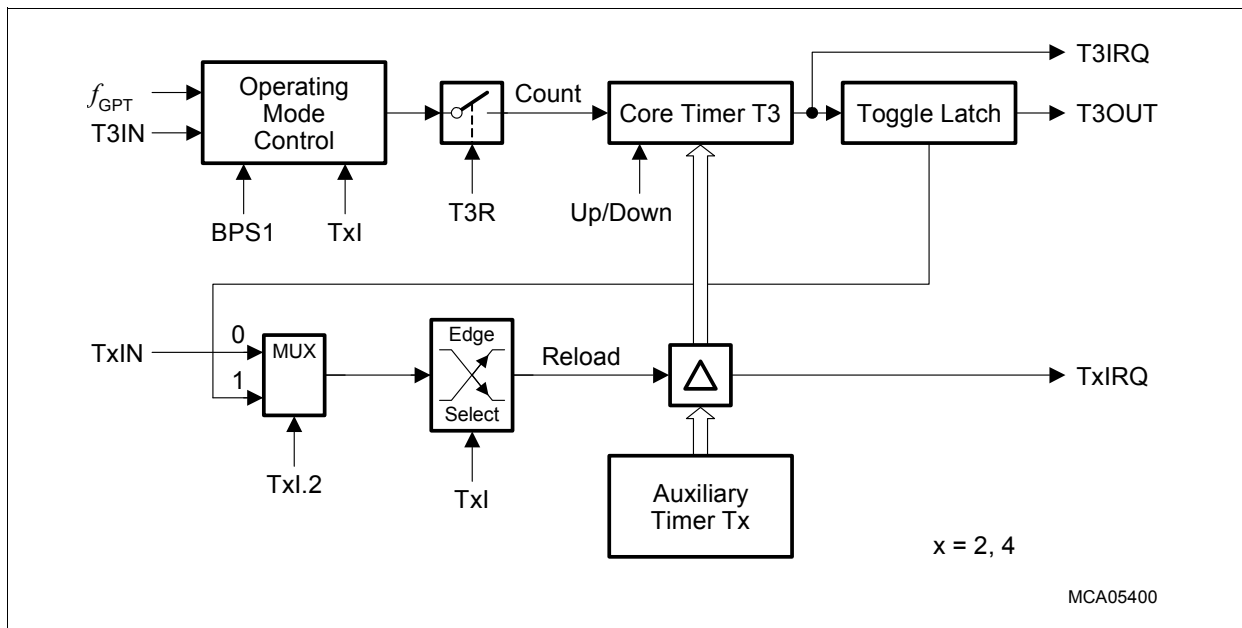
**Figure 16-15 Concatenation of Core Timer T3 and an Auxiliary Timer**

### Auxiliary Timer in Reload Mode

Reload Mode for an auxiliary timer Tx is selected by setting bitfield TxM in the respective register TxCON to 100<sub>B</sub>. In reload mode, the core timer T3 is reloaded with the contents of an auxiliary timer register, triggered by one of two different signals. The trigger signal is selected the same way as the clock source for counter mode (see [Table 16-5](#)), i.e. a transition of the auxiliary timer's input TxIN or the toggle latch T3OTL may trigger the reload.

*Note: When programmed for reload mode, the respective auxiliary timer (T2 or T4) stops independently of its run flag T2R or T4R.*

*The timer input pin TxIN must be configured as input if it shall trigger a reload operation.*



**Figure 16-16 GPT1 Auxiliary Timer in Reload Mode**

Upon a trigger signal, T3 is loaded with the contents of the respective timer register (T2 or T4) and the respective interrupt request flag (T2IR or T4IR) is set.

*Note: When a T3OTL transition is selected for the trigger signal, the interrupt request flag T3IR will also be set upon a trigger, indicating T3's overflow or underflow. Modifications of T3OTL via software will NOT trigger the counter function of T2/T4.*

To ensure that a transition of the reload input signal applied to TxIN is recognized correctly, its level must be held high or low for a minimum number of module clock cycles, detailed in [Section 16.1.5](#).

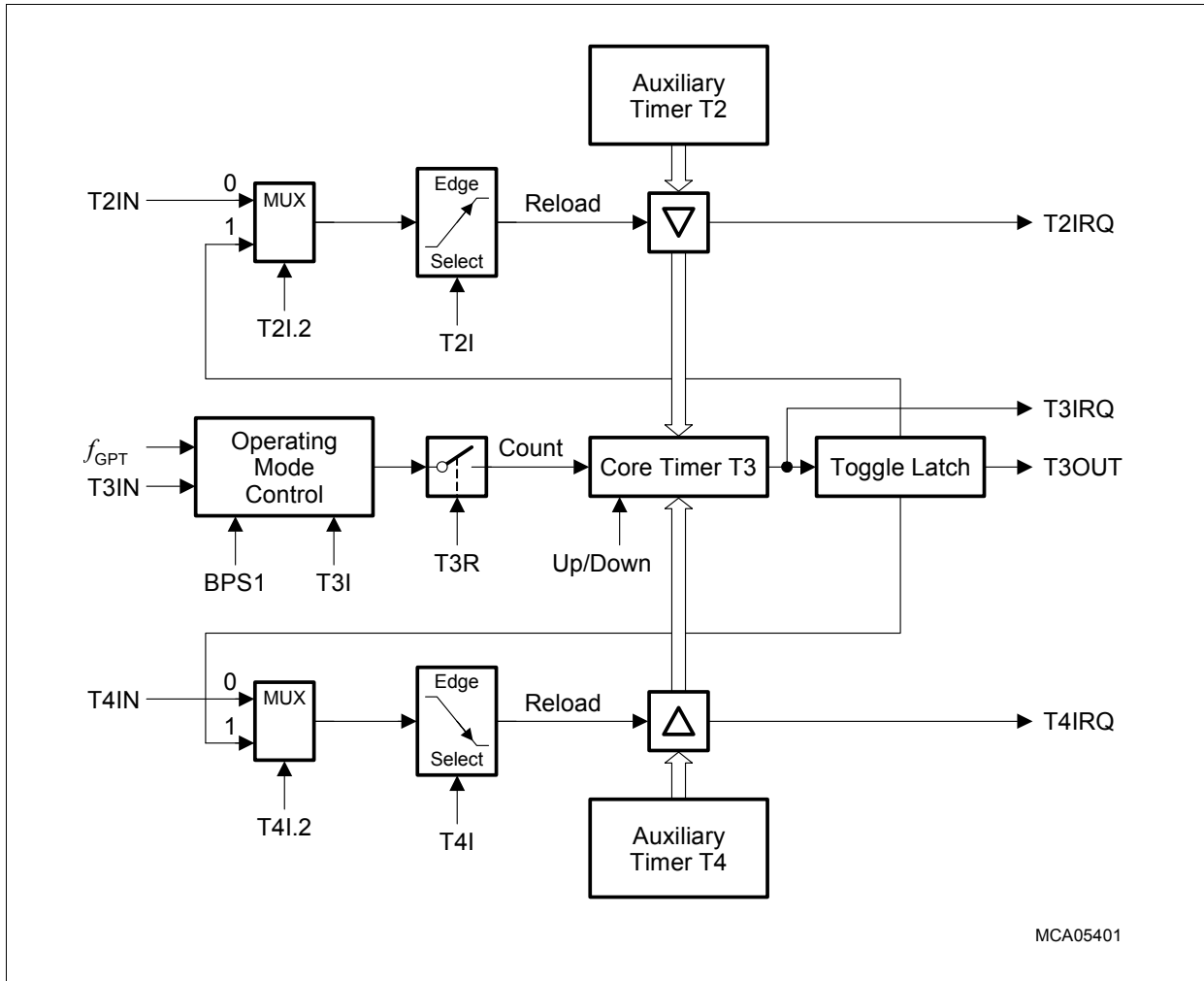
The reload mode triggered by the T3 toggle latch can be used in a number of different configurations. The following functions can be performed, depending on the selected active transition:

### **General Purpose Timer Units**

- If both a positive and a negative transition of T3OTL are selected to trigger a reload, the core timer will be reloaded with the contents of the auxiliary timer each time it overflows or underflows. This is the standard reload mode (reload on overflow/underflow).
- If either a positive or a negative transition of T3OTL is selected to trigger a reload, the core timer will be reloaded with the contents of the auxiliary timer on every second overflow or underflow.
- Using this “single-transition” mode for both auxiliary timers allows to perform very flexible Pulse Width Modulation (PWM). One of the auxiliary timers is programmed to reload the core timer on a positive transition of T3OTL, the other is programmed for a reload on a negative transition of T3OTL. With this combination the core timer is alternately reloaded from the two auxiliary timers.

**Figure 16-17** shows an example for the generation of a PWM signal using the “single-transition” reload mechanism. T2 defines the high time of the PWM signal (reloaded on positive transitions) and T4 defines the low time of the PWM signal (reloaded on negative transitions). The PWM signal can be output on pin T3OUT if T3OE = 1. With this method, the high and low time of the PWM signal can be varied in a wide range.

*Note: The output toggle latch T3OTL is accessible via software and may be changed, if required, to modify the PWM signal.  
However, this will NOT trigger the reloading of T3.*



**Figure 16-17 GPT1 Timer Reload Configuration for PWM Generation**

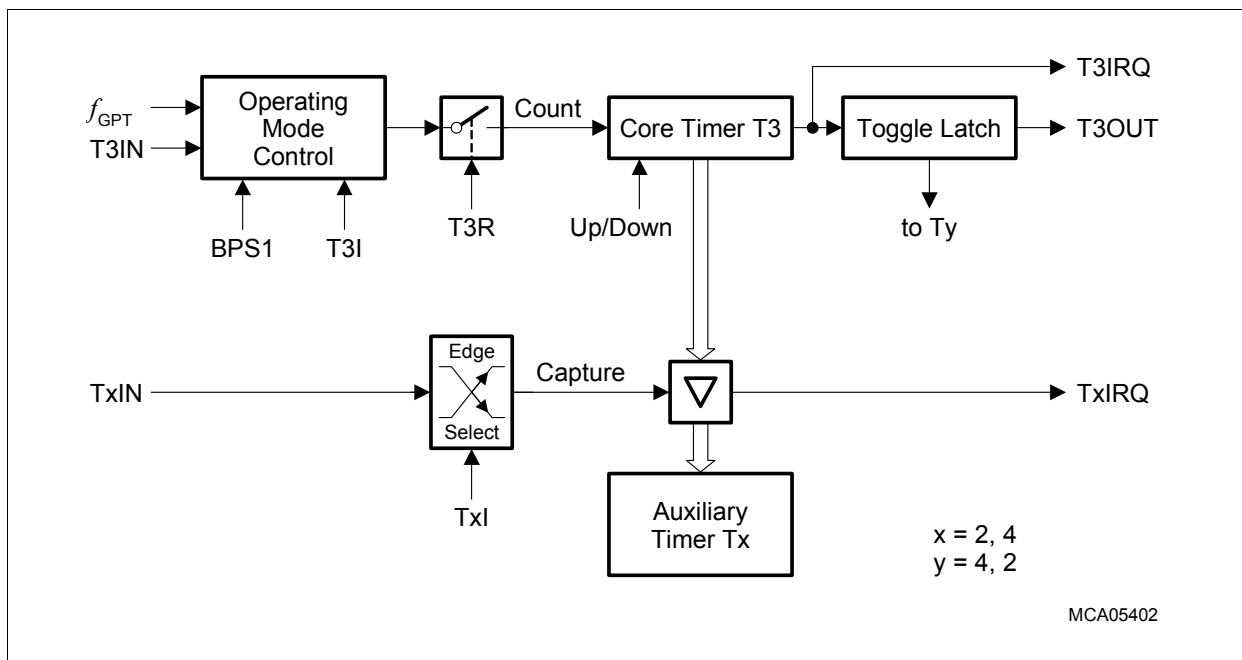
*Note: Although possible, selecting the same reload trigger event for both auxiliary timers should be avoided. In such a case, both reload registers would try to load the core timer at the same time. If this combination is selected, T2 is disregarded and the contents of T4 is reloaded.*

### Auxiliary Timer in Capture Mode

Capture mode for an auxiliary timer Tx is selected by setting bitfield TxM in the respective register TxCON to 101<sub>B</sub>. In capture mode, the contents of the core timer T3 are latched into an auxiliary timer register in response to a signal transition at the respective auxiliary timer's external input pin TxIN. The capture trigger signal can be a positive, a negative, or both a positive and a negative transition.

The two least significant bits of bitfield TxI select the active transition (see [Table 16-5](#)). Bit 2 of TxI is irrelevant for capture mode and must be cleared (TxI.2 = 0).

*Note: When programmed for capture mode, the respective auxiliary timer (T2 or T4) stops independently of its run flag T2R or T4R.*



**Figure 16-18 GPT1 Auxiliary Timer in Capture Mode**

Upon a trigger (selected transition) at the corresponding input pin TxIN the contents of the core timer are loaded into the auxiliary timer register and the associated interrupt request flag TxIR will be set.

For capture mode operation, the respective timer input pin TxIN must be configured as input. To ensure that a transition of the capture input signal applied to TxIN is recognized correctly, its level must be held high or low for a minimum number of module clock cycles, detailed in [Section 16.1.5](#).

### 16.1.5 GPT1 Clock Signal Control

All actions within the timer block GPT1 are triggered by transitions of its basic clock. This basic clock is derived from the system clock by a basic block prescaler, controlled by bitfield BPS1 in register T3CON (see [Figure 16-2](#)). The count clock can be generated in two different ways:

- **Internal count clock**, derived from GPT1's basic clock via a programmable prescaler, is used for (gated) timer mode.
- **External count clock**, derived from the timer's input pin(s), is used for counter mode.

For both ways, the basic clock determines the maximum count frequency and the timer's resolution:

**Table 16-6 Basic Clock Selection for Block GPT1**

Block Prescaler <sup>1)</sup>	BPS1 = 01 <sub>B</sub>	BPS1 = 00 <sub>B</sub> <sup>2)</sup>	BPS1 = 11 <sub>B</sub>	BPS1 = 10 <sub>B</sub>
<b>Prescaling Factor for GPT1: F(BPS1)</b>	F(BPS1) = 4	F(BPS1) = 8	F(BPS1) = 16	F(BPS1) = 32
<b>Maximum External Count Frequency</b>	$f_{\text{GPT}}/8$	$f_{\text{GPT}}/16$	$f_{\text{GPT}}/32$	$f_{\text{GPT}}/64$
<b>Input Signal Stable Time</b>	$4 \times t_{\text{GPT}}$	$8 \times t_{\text{GPT}}$	$16 \times t_{\text{GPT}}$	$32 \times t_{\text{GPT}}$

1) Please note the non-linear encoding of bitfield BPS1.

2) Default after reset.

#### Internal Count Clock Generation

In timer mode and gated timer mode, the count clock for each GPT1 timer is derived from the GPT1 basic clock by a programmable prescaler, controlled by bitfield TxI in the respective timer's control register TxCON.

The count frequency  $f_{\text{Tx}}$  for a timer Tx and its resolution  $r_{\text{Tx}}$  are scaled linearly with lower clock frequencies, as can be seen from the following formula:

$$f_{\text{Tx}} = \frac{f_{\text{GPT}}}{F(\text{BPS1}) \times 2^{<\text{TxI}>}} \quad r_{\text{Tx}}[\mu\text{s}] = \frac{F(\text{BPS1}) \times 2^{<\text{TxI}>}}{f_{\text{GPT}}[\text{MHz}]} \quad [16.1]$$

The effective count frequency depends on the common module clock prescaler factor F(BPS1) as well as on the individual input prescaler factor  $2^{<\text{TxI}>}$ . [Table 16-7](#) summarizes the resulting overall divider factors for a GPT1 timer that result from these cascaded prescalers.

**General Purpose Timer Units**

**Table 16-8** lists a timer's parameters (such as count frequency, resolution, and period) resulting from the selected overall prescaler factor and the applied system frequency. Note that some numbers may be rounded.

**Table 16-7 GPT1 Overall Prescaler Factors for Internal Count Clock**

Individual Prescaler for Tx	Common Prescaler for Module Clock <sup>1)</sup>			
	BPS1 = 01 <sub>B</sub>	BPS1 = 00 <sub>B</sub>	BPS1 = 11 <sub>B</sub>	BPS1 = 10 <sub>B</sub>
Txl = 000 <sub>B</sub>	4	8	16	32
Txl = 001 <sub>B</sub>	8	16	32	64
Txl = 010 <sub>B</sub>	16	32	64	128
Txl = 011 <sub>B</sub>	32	64	128	256
Txl = 100 <sub>B</sub>	64	128	256	512
Txl = 101 <sub>B</sub>	128	256	512	1024
Txl = 110 <sub>B</sub>	256	512	1024	2048
Txl = 111 <sub>B</sub>	512	1024	2048	4096

1) Please note the non-linear encoding of bitfield BPS1.

**Table 16-8 GPT1 Timer Parameters**

System Clock = 10 MHz			Overall Divider Factor	System Clock = 40 MHz		
Frequency	Resolution	Period		Frequency	Resolution	Period
2.5 MHz	400 ns	26.21 ms	4	10.0 MHz	100 ns	6.55 ms
1.25 MHz	800 ns	52.43 ms	8	5.0 MHz	200 ns	13.11 ms
625.0 kHz	1.6 µs	104.9 ms	16	2.5 MHz	400 ns	26.21 ms
312.5 kHz	3.2 µs	209.7 ms	32	1.25 MHz	800 ns	52.43 ms
156.25 kHz	6.4 µs	419.4 ms	64	625.0 kHz	1.6 µs	104.9 ms
78.125 kHz	12.8 µs	838.9 ms	128	312.5 kHz	3.2 µs	209.7 ms
39.06 kHz	25.6 µs	1.678 s	256	156.25 kHz	6.4 µs	419.4 ms
19.53 kHz	51.2 µs	3.355 s	512	78.125 kHz	12.8 µs	838.9 ms
9.77 kHz	102.4 µs	6.711 s	1024	39.06 kHz	25.6 µs	1.678 s
4.88 kHz	204.8 µs	13.42 s	2048	19.53 kHz	51.2 µs	3.355 s
2.44 kHz	409.6 µs	26.84 s	4096	9.77 kHz	102.4 µs	6.711 s

### External Count Clock Input

The external input signals of the GPT1 block are sampled with the GPT1 basic clock (see [Figure 16-2](#)). To ensure that a signal is recognized correctly, its current level (high or low) must be held active for at least one complete sampling period, before changing. A signal transition is recognized if two subsequent samples of the input signal represent different levels. Therefore, a minimum of two basic clock periods are required for the sampling of an external input signal. Thus, the maximum frequency of an input signal must not be higher than half the basic clock.

**Table 16-9** summarizes the resulting requirements for external GPT1 input signals.

**Table 16-9 GPT1 External Input Signal Limits**

System Clock = 10 MHz		Input Freq. Factor	GPT1 Divider BPS1	Input Phase Duration	System Clock = 40 MHz	
Max. Input Frequency	Min. Level Hold Time				Max. Input Frequency	Min. Level Hold Time
1.25 MHz	400 ns	$f_{GPT}/8$	01 <sub>B</sub>	$4 \times t_{GPT}$	5.0 MHz	100 ns
625.0 kHz	800 ns	$f_{GPT}/16$	00 <sub>B</sub>	$8 \times t_{GPT}$	2.5 MHz	200 ns
312.5 kHz	1.6 $\mu$ s	$f_{GPT}/32$	11 <sub>B</sub>	$16 \times t_{GPT}$	1.25 MHz	400 ns
156.25 kHz	3.2 $\mu$ s	$f_{GPT}/64$	10 <sub>B</sub>	$32 \times t_{GPT}$	625.0 kHz	800 ns

These limitations are valid for all external input signals to GPT1, including the external count signals in counter mode and incremental interface mode, the gate input signals in gated timer mode, and the external direction signals.



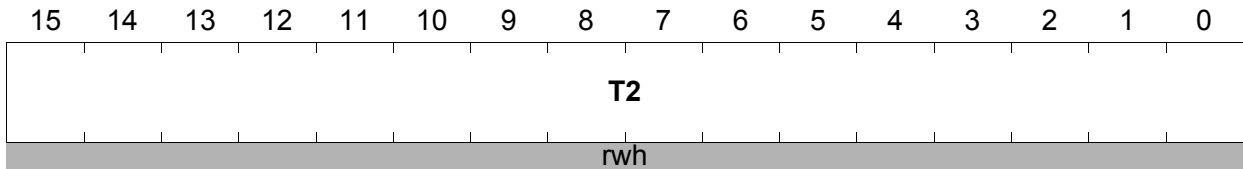
### 16.1.6 GPT1 Timer Registers

#### GPT12E\_T2

**Timer 2 Count Register**

**SFR (FE40<sub>H</sub>/20<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



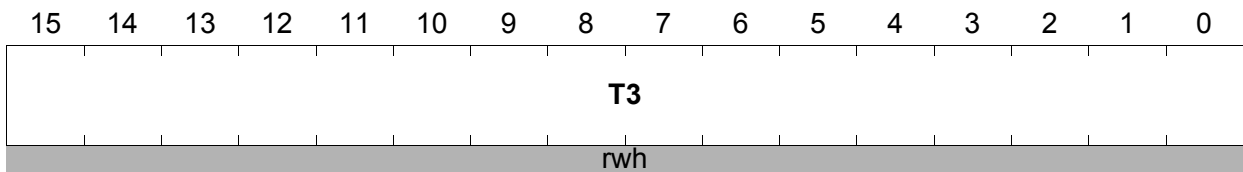
Field	Bits	Typ	Description
T2	[15:0]	rwh	<b>Timer T2 Current Value</b> Contains the current value of the timer T2

#### GPT12E\_T3

**Timer 3 Count Register**

**SFR (FE42<sub>H</sub>/21<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



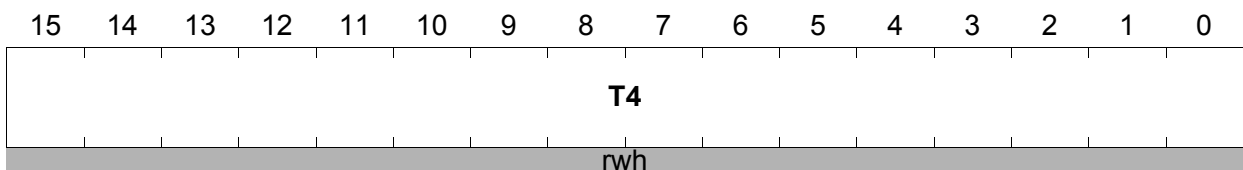
Field	Bits	Typ	Description
T3	[15:0]	rwh	<b>Timer T3 Current Value</b> Contains the current value of the timer T3

#### GPT12E\_T4

**Timer 4 Count Register**

**SFR (FE44<sub>H</sub>/22<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



Field	Bits	Typ	Description
T4	[15:0]	rwh	<b>Timer T4 Current Value</b> Contains the current value of the timer T4

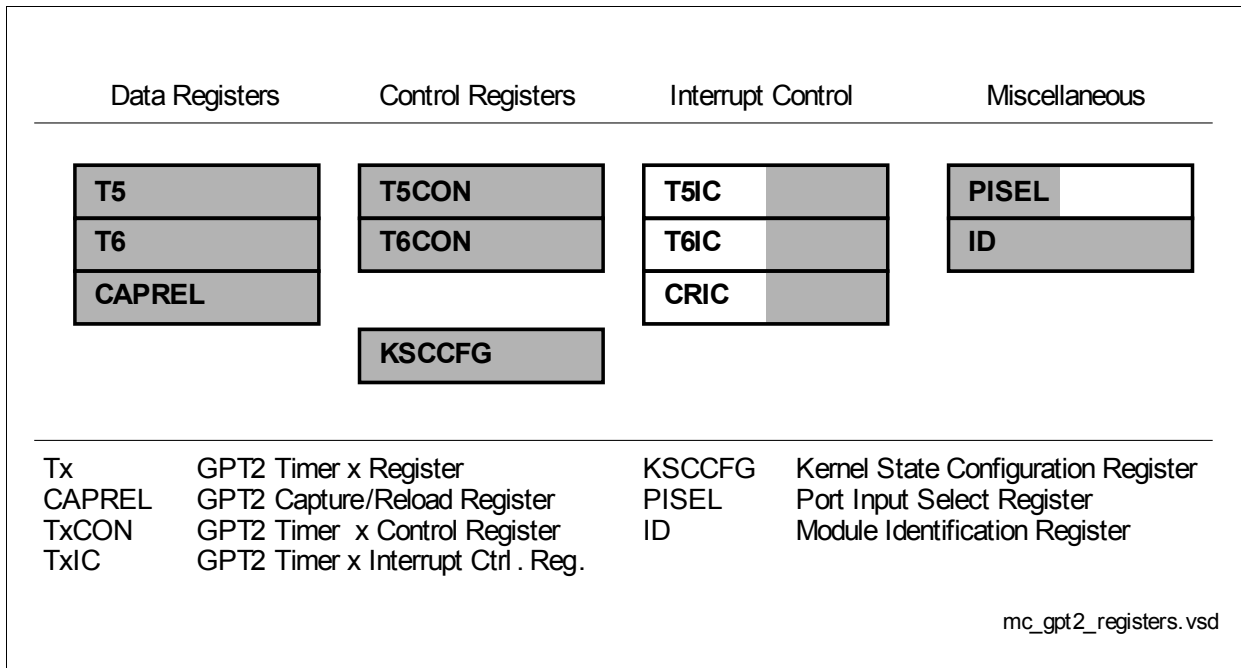
### **16.1.7 Interrupt Control for GPT1 Timers**

When a timer overflows from  $FFFF_H$  to  $0000_H$  (when counting up), or when it underflows from  $0000_H$  to  $FFFF_H$  (when counting down), its interrupt request flag in register GPT12E\_TxIC ( $x = 2, 3, 4$ ) will be set. This will cause an interrupt to the respective timer interrupt vector or trigger a PEC service, if the respective interrupt enable bit is set.

There is an interrupt control register for each of the three timers (T2, T3, T4). All interrupt control registers have the same structure described in section Interrupt Control.

## 16.2 Timer Block GPT2

From a programmer's point of view, the GPT2 block is represented by a set of SFRs as summarized below. Those portions of port and direction registers which are used for alternate functions by the GPT2 block are shaded.



**Figure 16-19 SFRs Associated with Timer Block GPT2**

Both timers of block GPT2 (T5, T6) can run in one of 3 basic modes: Timer Mode, Gated Timer Mode, or Counter Mode. All timers can count up or down. Each timer of GPT2 is controlled by a separate control register TxCON.

Each timer has an input pin TxIN (alternate pin function) associated with it, which serves as the gate control in gated timer mode, or as the count input in counter mode. The count direction (up/down) may be programmed via software or may be dynamically altered by a signal at the External Up/Down control input TxEUD (alternate pin function). An overflow/underflow of core timer T6 is indicated by the Output Toggle Latch T6OTL, whose state may be output on the associated pin T6OUT (alternate pin function). The auxiliary timer T5 may additionally be concatenated with core timer T6 (through T6OTL).

The Capture/Reload register CAPREL can be used to capture the contents of timer T5, or to reload timer T6. A special mode facilitates the use of register CAPREL for both functions at the same time. This mode allows frequency multiplication. The capture function is triggered by the input pin CAPIN, or by GPT1 timer's T3 input lines T3IN and T3EUD. The reload function is triggered by an overflow or underflow of timer T6. Overflows/underflows of timer T6 may also clock the timers of the CAPCOM units.

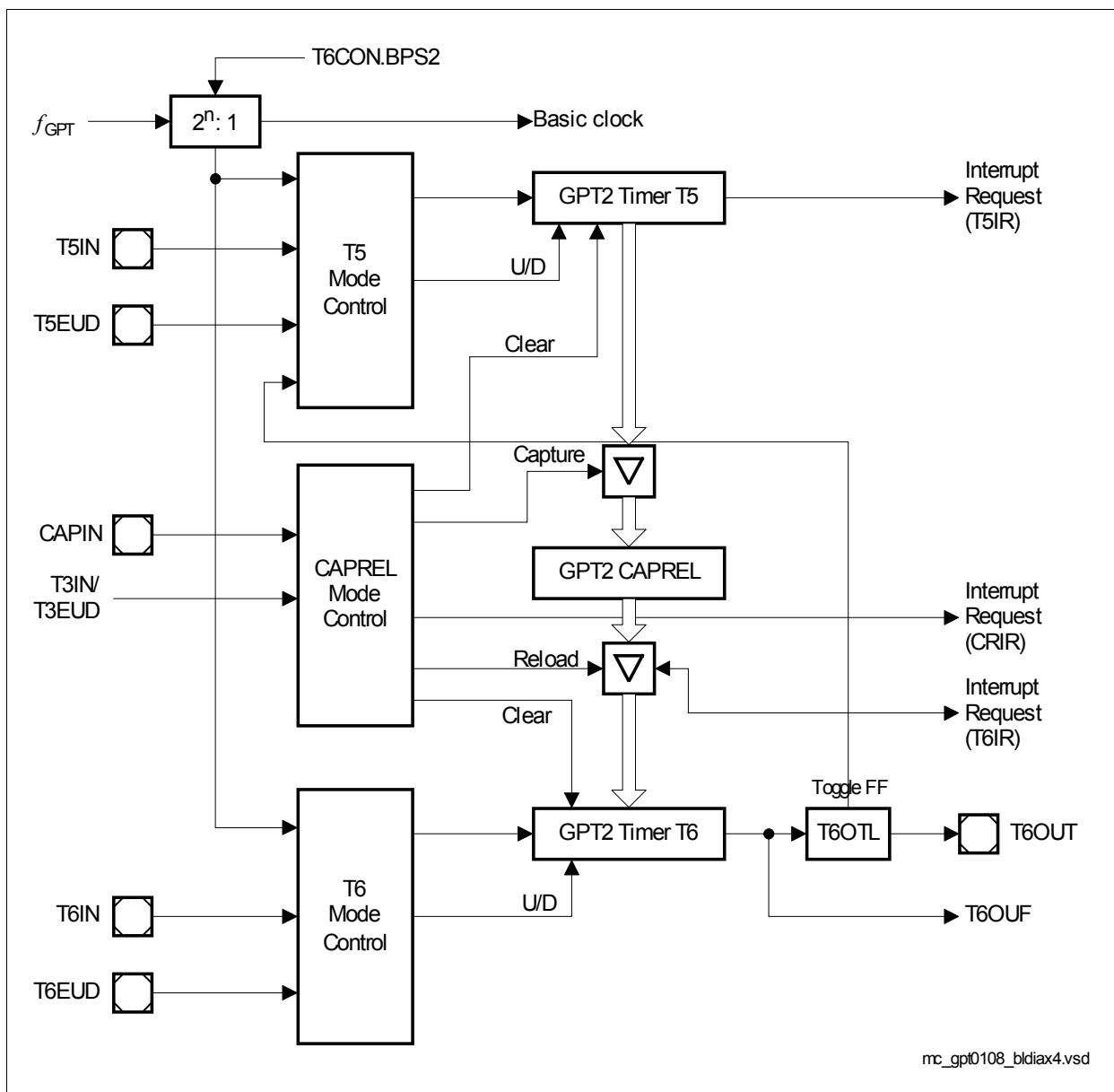
The current contents of each timer can be read or modified by the CPU by accessing the corresponding timer count registers T5 or T6, located in the non-bitaddressable SFR

## General Purpose Timer Units

space (see [Section 16.2.7](#)). When any of the timer registers is written to by the CPU in the state immediately preceding a timer increment, decrement, reload, or capture operation, the CPU write operation has priority in order to guarantee correct results.

The interrupts of GPT2 are controlled through the Interrupt Control Registers TxIC. These registers are not part of the GPT2 block. The input and output lines of GPT2 are connected to pins of Ports P3 and P5. The control registers for the port functions are located in the respective port modules.

*Note: The timing requirements for external input signals can be found in [Section 16.2.6](#), [Section 16.5](#) summarizes the module interface signals, including pins.*



**Figure 16-20 GPT2 Block Diagram**

## 16.2.1 GPT2 Core Timer T6 Control

The current contents of the core timer T6 are reflected by its count register T6. This register can also be written to by the CPU, for example, to set the initial start value.

The core timer T6 is configured and controlled via its bitaddressable control register T6CON.

### GPT12E\_T6CON

**Timer 6 Control Register**

**SFR (FF48<sub>H</sub>/A4<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>T6 SR</b>	<b>T6 CLR</b>	-	<b>BPS2</b>		<b>T6 OTL</b>	<b>T6 OE</b>	<b>T6 UDE</b>	<b>T6 UD</b>	<b>T6R</b>		<b>T6M</b>			<b>T6I</b>	
rw	rw	-	rw		rwh	rw	rw	rw	rw		rw			rw	

Field	Bits	Type	Description
<b>T6I</b>	[2:0]	rw	<b>Timer T6 Input Parameter Selection</b> Depends on the operating mode, see respective sections for encoding: <a href="#">Table 16-15</a> for Timer Mode and Gated Timer Mode <a href="#">Table 16-11</a> for Counter Mode
<b>T6M</b>	[5:3]	rw	<b>Timer T6 Mode Control (Basic Operating Mode)</b> 000 <sub>B</sub> Timer Mode 001 <sub>B</sub> Counter Mode 010 <sub>B</sub> Gated Timer Mode with gate active low 011 <sub>B</sub> Gated Timer Mode with gate active high 100 <sub>B</sub> Reserved. Do not use this combination. 101 <sub>B</sub> Reserved. Do not use this combination. 110 <sub>B</sub> Reserved. Do not use this combination. 111 <sub>B</sub> Reserved. Do not use this combination.
<b>T6R</b>	6	rw	<b>Timer T6 Run Bit</b> 0 <sub>B</sub> Timer T6 stops 1 <sub>B</sub> Timer T6 runs
<b>T6UD</b>	7	rw	<b>Timer T6 Up/Down Control<sup>1)</sup></b> 0 <sub>B</sub> Timer T6 counts up 1 <sub>B</sub> Timer T6 counts down
<b>T6UDE</b>	8	rw	<b>Timer T6 External Up/Down Enable<sup>1)</sup></b> 0 <sub>B</sub> Input T6EUD is disconnected 1 <sub>B</sub> Direction influenced by input T6EUD

**General Purpose Timer Units**

Field	Bits	Type	Description
<b>T6OE</b>	9	rw	<b>Overflow/Underflow Output Enable</b> $0_B$ Alternate Output Function Disabled $1_B$ State of T6 toggle latch is output on pin T6OUT
<b>T6OTL</b>	10	rwh	<b>Timer T6 Overflow Toggle Latch</b> Toggles on each overflow/underflow of T6. Can be set or reset by software (see separate description)
<b>BPS2</b>	[12:11]	rw	<b>GPT2 Block Prescaler Control</b> Selects the basic clock for block GPT2 (see also <a href="#">Section 16.2.6</a> ) $00_B$ $f_{GPT}/4$ $01_B$ $f_{GPT}/2$ $10_B$ $f_{GPT}/16$ $11_B$ $f_{GPT}/8$
<b>T6CLR</b>	14	rw	<b>Timer T6 Clear Enable Bit</b> $0_B$ Timer T6 is not cleared on a capture event $1_B$ Timer T6 is cleared on a capture event
<b>T6SR</b>	15	rw	<b>Timer 6 Reload Mode Enable</b> $0_B$ Reload from register CAPREL disabled $1_B$ Reload from register CAPREL enabled

1) See [Table 16-10](#) for encoding of bits T6UD and T6UDE.

### Timer T6 Run Control

The core timer T6 can be started or stopped by software through bit T6R (timer T6 run bit). This bit is relevant in all operating modes of T6. Setting bit T6R will start the timer, clearing bit T6R stops the timer.

In gated timer mode, the timer will only run if T6R = 1 and the gate is active (high or low, as programmed).

*Note: When bit T5RC in timer control register T5CON is set, bit T6R will also control (start and stop) the Auxiliary Timer T5.*

### Count Direction Control

The count direction of the GPT2 timers (core timer and auxiliary timer) can be controlled either by software or by the external input pin TxEUD (Timer Tx External Up/Down Control Input). These options are selected by bits TxUD and TxUDE in the respective control register TxCON. When the up/down control is provided by software (bit TxUDE = 0), the count direction can be altered by setting or clearing bit TxUD. When bit TxUDE = 1, pin TxEUD is selected to be the controlling source of the count direction. However, bit TxUD can still be used to reverse the actual count direction, as shown in [Table 16-10](#). The count direction can be changed regardless of whether or not the timer is running.

**Table 16-10 GPT2 Timer Count Direction Control**

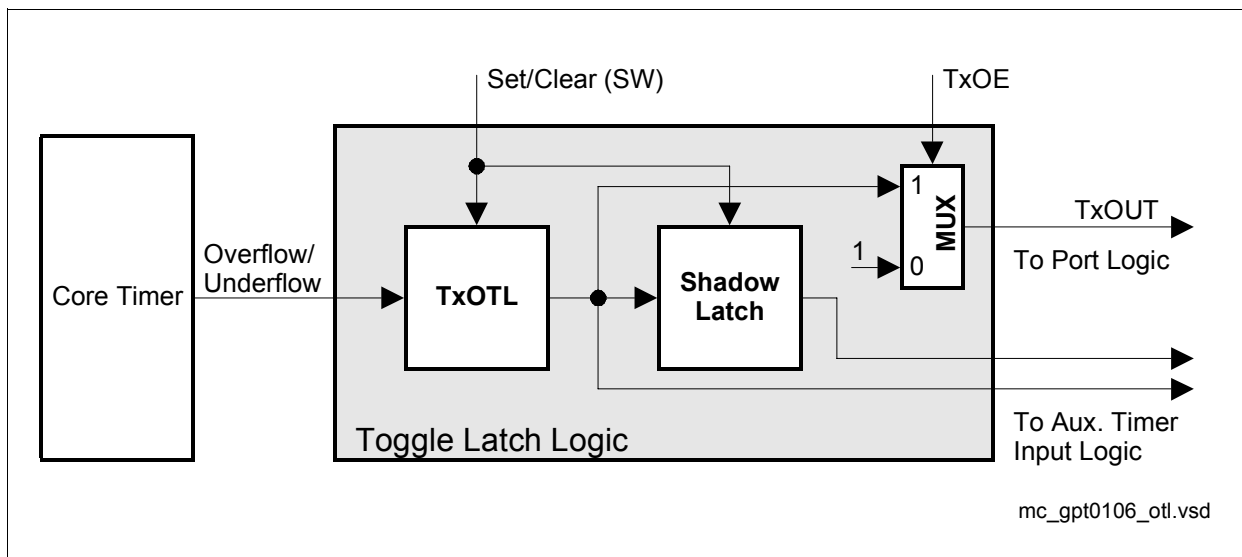
Pin TxEUD	Bit TxUDE	Bit TxUD	Count Direction
X	0	0	Count Up
X	0	1	Count Down
0	1	0	Count Up
1	1	0	Count Down
0	1	1	Count Down
1	1	1	Count Up

### Timer 6 Output Toggle Latch

The overflow/underflow signal of timer T6 is connected to a block named 'Toggle Latch', shown in the timer mode diagrams. **Figure 16-21** illustrates the details of this block. An overflow or underflow of T6 will clock two latches: The first latch represents bit T6OTL in control register T6CON. The second latch is an internal latch toggled by T6OTL's output. Both latch outputs are connected to the input control block of the auxiliary timer T5. The output level of the shadow latch will match the output level of T6OTL, but is delayed by one clock cycle. When the T6OTL value changes, this will result in a temporarily different output level from T6OTL and the shadow latch, which can trigger the selected count event in T5.

When software writes to T6OTL, both latches are set or cleared simultaneously. In this case, both signals to the auxiliary timers carry the same level and no edge will be detected. Bit T6OE (overflow/underflow output enable) in register T6CON enables the state of T6OTL to be monitored via an external pin T6OUT. When T6OTL is linked to an external port pin (must be configured as output), T6OUT can be used to control external HW. If T6OE = 1, pin T6OUT outputs the state of T6OTL. If T6OE = 0, pin T6OUT outputs a high level (while it selects the timer output signal).

As can be seen from **Figure 16-21**, when latch T6OTL is modified by software to determine the state of the output line, also the internal shadow latch is set or cleared accordingly. Therefore, no trigger condition is detected by T5 in this case.



**Figure 16-21 Block Diagram of the Toggle Latch Logic of Core Timer T6**

*Note: T6 is also used to clock the timers in the CAPCOM units. For this purpose, there is a direct internal connection between the T6 overflow/underflow line and the CAPCOM timers (signal T6OUF).*

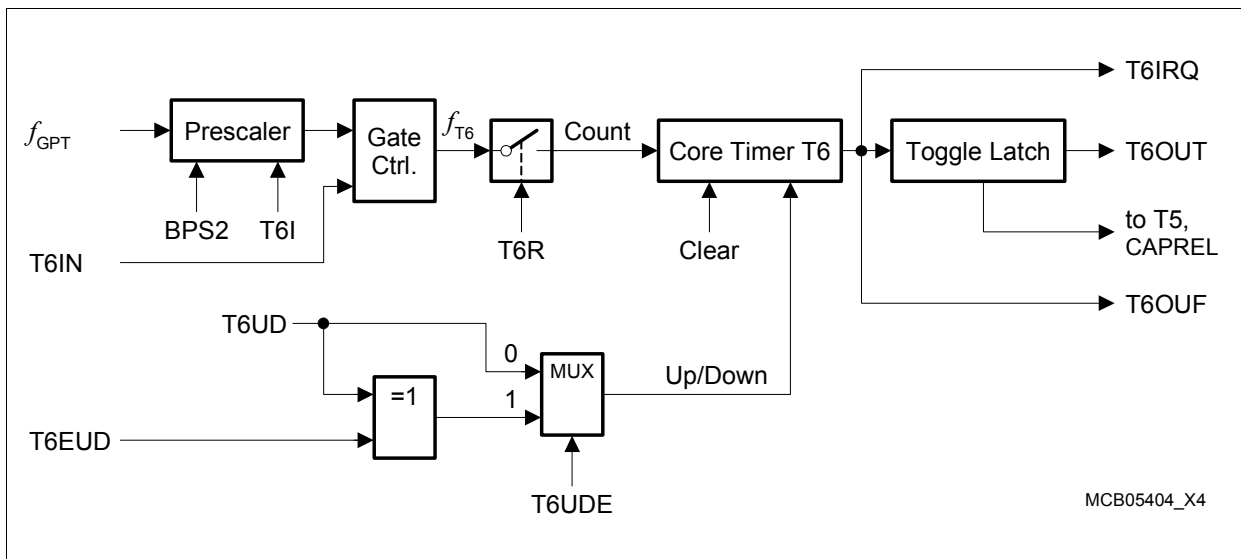




### Gated Timer Mode

Gated timer mode for the core timer T6 is selected by setting bitfield T6M in register T6CON to 010<sub>B</sub> or 011<sub>B</sub>. Bit T6M.0 (T6CON.3) selects the active level of the gate input. The same options for the input frequency are available in gated timer mode as in timer mode (see [Section 16.2.6](#)). However, the input clock to the timer in this mode is gated by the external input pin T6IN (Timer T6 External Input).

To enable this operation, the associated pin T6IN must be configured as input.



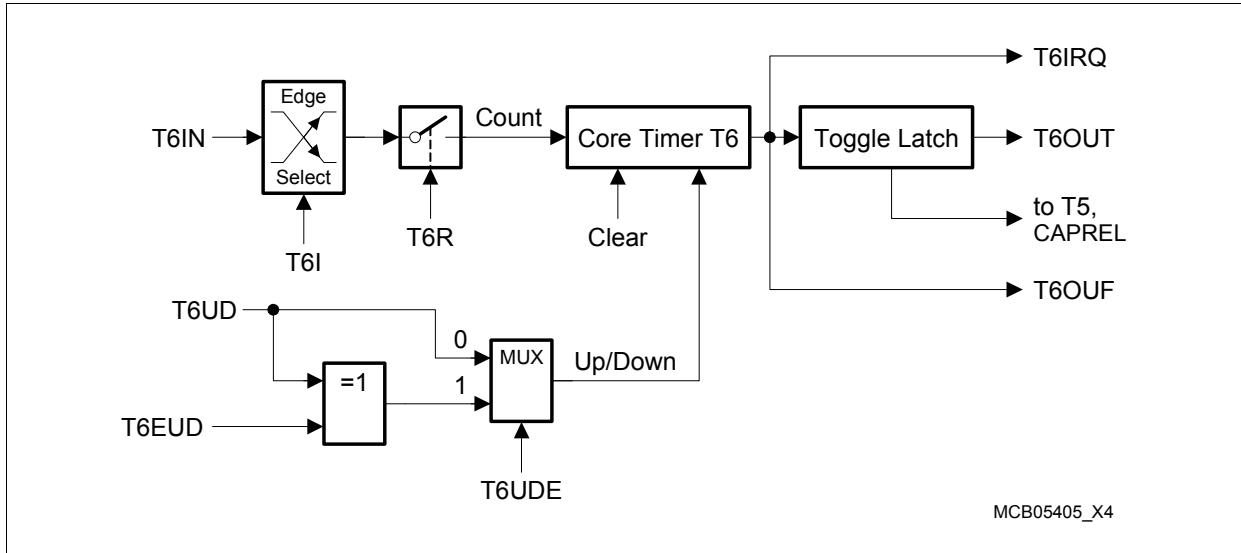
**Figure 16-23 Block Diagram of Core Timer T6 in Gated Timer Mode**

If T6M = 010<sub>B</sub>, the timer is enabled when T6IN shows a low level. A high level at this line stops the timer. If T6M = 011<sub>B</sub>, line T6IN must have a high level in order to enable the timer. Additionally, the timer can be turned on or off by software using bit T6R. The timer will only run if T6R is 1 and the gate is active. It will stop if either T6R is 0 or the gate is inactive.

*Note: A transition of the gate signal at pin T6IN does not cause an interrupt request.*

### Counter Mode

Counter mode for the core timer T6 is selected by setting bitfield T6M in register T6CON to 001<sub>B</sub>. In counter mode, timer T6 is clocked by a transition at the external input pin T6IN. The event causing an increment or decrement of the timer can be a positive, a negative, or both a positive and a negative transition at this line. Bitfield T6I in control register T6CON selects the triggering transition (see [Table 16-11](#)).



**Figure 16-24 Block Diagram of Core Timer T6 in Counter Mode**

**Table 16-11 GPT2 Core Timer T6 (Counter Mode) Input Edge Selection**

T6I	Triggering Edge for Counter Increment/Decrement
000 <sub>B</sub>	None. Counter T6 is disabled
001 <sub>B</sub>	Positive transition (rising edge) on T6IN
010 <sub>B</sub>	Negative transition (falling edge) on T6IN
011 <sub>B</sub>	Any transition (rising or falling edge) on T6IN
1XX <sub>B</sub>	Reserved. Do not use this combination

For counter mode operation, pin T6IN must be configured as input. The maximum input frequency allowed in counter mode depends on the selected prescaler value. To ensure that a transition of the count input signal applied to T6IN is recognized correctly, its level must be held high or low for a minimum number of module clock cycles before it changes. This information can be found in [Section 16.2.6](#).

### 16.2.3 GPT2 Auxiliary Timer T5 Control

Auxiliary timer T5 can be configured for timer mode, gated timer mode, or counter mode with the same options for the timer frequencies and the count signal as the core timer T6. In addition to these 3 counting modes, the auxiliary timer can be concatenated with the core timer. The contents of T5 may be captured to register CAPREL upon an external or an internal trigger. The start/stop function of the auxiliary timers can be remotely controlled by the T6 run control bit. Several timers may thus be controlled synchronously.

The current contents of the auxiliary timer are reflected by its count register T5. This register can also be written to by the CPU, for example, to set the initial start value.

The individual configurations for timer T5 are determined by its bitaddressable control register T5CON. Some bits in this register also control the function of the CAPREL register. Note that functions which are present in all timers of block GPT2 are controlled in the same bit positions and in the same manner in each of the specific control registers.

*Note: The auxiliary timer has no output toggle latch and no alternate output function.*

#### GPT12E\_T5CON

**Timer 5 Control Register**

**SFR (FF46<sub>H</sub>/A3<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T5 SC	T5 CLR	CI		-	CT3	T5 RC	T5 UDE	T5 UD	T5R	T5M			T5I		
rw	rw	rw		-	rw	rw	rw	rw	rw		rw			rw	

Field	Bits	Type	Description
<b>T5I</b>	[2:0]	rw	<b>Timer T5 Input Parameter Selection</b> Depends on the operating mode, see respective sections for encoding: <a href="#">Table 16-15</a> for Timer Mode and Gated Timer Mode <a href="#">Table 16-11</a> for Counter Mode
<b>T5M</b>	[5:3]	rw	<b>Timer T5 Mode Control (Basic Operating Mode)</b> 000 <sub>B</sub> Timer Mode 001 <sub>B</sub> Counter Mode 010 <sub>B</sub> Gated Timer Mode with gate active low 011 <sub>B</sub> Gated Timer Mode with gate active high 100 <sub>B</sub> Reserved. Do not use this combination 101 <sub>B</sub> Reserved. Do not use this combination 110 <sub>B</sub> Reserved. Do not use this combination 111 <sub>B</sub> Reserved. Do not use this combination

**General Purpose Timer Units**

Field	Bits	Type	Description
<b>T5R</b>	6	rw	<b>Timer T5 Run Bit</b> 0 <sub>B</sub> Timer T5 stops 1 <sub>B</sub> Timer T5 runs <i>Note: This bit only controls timer T5 if bit T5RC = 0.</i>
<b>T5UD</b>	7	rw	<b>Timer T5 Up/Down Control<sup>1)</sup></b> 0 <sub>B</sub> Timer T5 counts up 1 <sub>B</sub> Timer T5 counts down
<b>T5UDE</b>	8	rw	<b>Timer T5 External Up/Down Enable<sup>1)</sup></b> 0 <sub>B</sub> Input T5EUD is disconnected 1 <sub>B</sub> Direction influenced by input T5EUD
<b>T5RC</b>	9	rw	<b>Timer T5 Remote Control</b> 0 <sub>B</sub> Timer T5 is controlled by its own run bit T5R 1 <sub>B</sub> Timer T5 is controlled by the run bit T6R of core timer 6, not by bit T5R
<b>CT3</b>	10	rw	<b>Timer T3 Capture Trigger Enable</b> 0 <sub>B</sub> Capture trigger from input line CAPIN 1 <sub>B</sub> Capture trigger from T3 input lines T3IN and/or T3EUD
<b>CI</b>	[13:12]	rw	<b>Register CAPREL Capture Trigger Selection<sup>2)</sup></b> 00 <sub>B</sub> Capture disabled 01 <sub>B</sub> Positive transition (rising edge) on CAPIN <sup>3)</sup> or any transition on T3IN 10 <sub>B</sub> Negative transition (falling edge) on CAPIN or any transition on T3EUD 11 <sub>B</sub> Any transition (rising or falling edge) on CAPIN or any transition on T3IN or T3EUD
<b>T5CLR</b>	14	rw	<b>Timer T5 Clear Enable Bit</b> 0 <sub>B</sub> Timer T5 is not cleared on a capture event 1 <sub>B</sub> Timer T5 is cleared on a capture event
<b>T5SC</b>	15	rw	<b>Timer T5 Capture Mode Enable</b> 0 <sub>B</sub> Capture into register CAPREL disabled 1 <sub>B</sub> Capture into register CAPREL enabled

1) See [Table 16-10](#) for encoding of bits T5UD and T5UDE.

2) To define the respective trigger source signal, also bit CT3 must be regarded (see [Table 16-13](#)).

3) Rising edge must be selected if capturing is triggered by the internal GPT1 read signals (see register PISEL and [“Combined Capture Modes” on Page 16-55](#)).

### **Timer T5 Run Control**

The auxiliary timer T5 can be started or stopped by software in two different ways:

- Through the associated timer run bit (T5R). In this case it is required that the respective control bit T5RC = 0.
- Through the core timer's run bit (T6R). In this case the respective remote control bit must be set (T5RC = 1).

The selected run bit is relevant in all operating modes of T5. Setting the bit will start the timer, clearing the bit stops the timer.

In gated timer mode, the timer will only run if the selected run bit is set and the gate is active (high or low, as programmed).

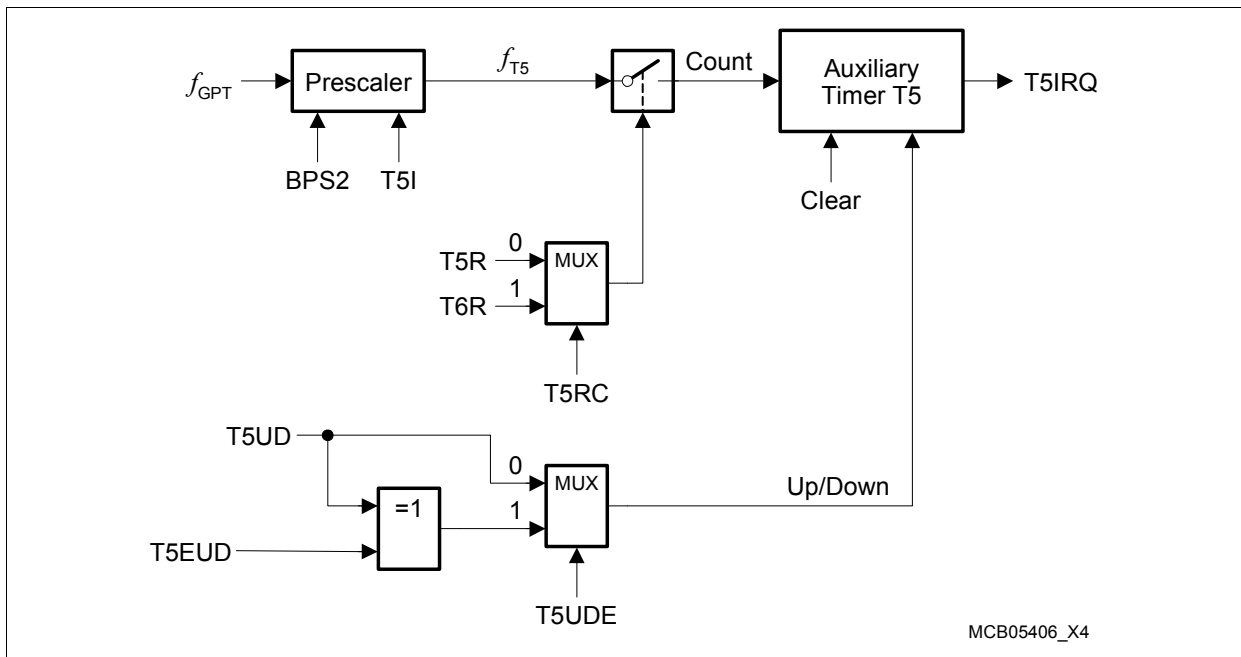
*Note: If remote control is selected T6R will start/stop timer T6 and the auxiliary timer T5 synchronously.*

### 16.2.4 GPT2 Auxiliary Timer T5 Operating Modes

The operation of the auxiliary timer in the basic operating modes is almost identical with the core timer's operation, with very few exceptions. Additionally, some combined operating modes can be selected.

#### Timer T5 in Timer Mode

Timer Mode for the auxiliary timer T5 is selected by setting its bitfield T5M in register T5CON to 000<sub>B</sub>.

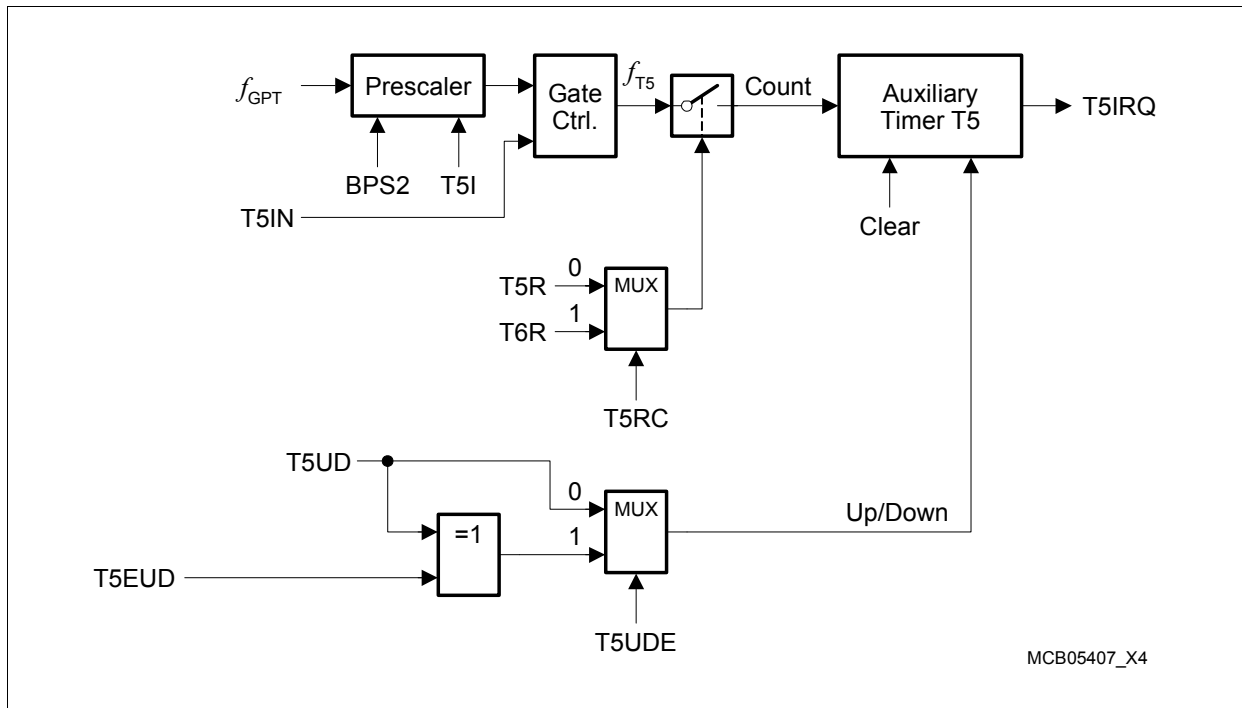


**Figure 16-25 Block Diagram of Auxiliary Timer T5 in Timer Mode**

### Timer T5 in Gated Timer Mode

Gated timer mode for the auxiliary timer T5 is selected by setting bitfield T5M in register T5CON to 010<sub>B</sub> or 011<sub>B</sub>. Bit T5M.0 (T5CON.3) selects the active level of the gate input.

*Note: A transition of the gate signal at line T5IN does not cause an interrupt request.*



**Figure 16-26 Block Diagram of Auxiliary Timer T5 in Gated Timer Mode**

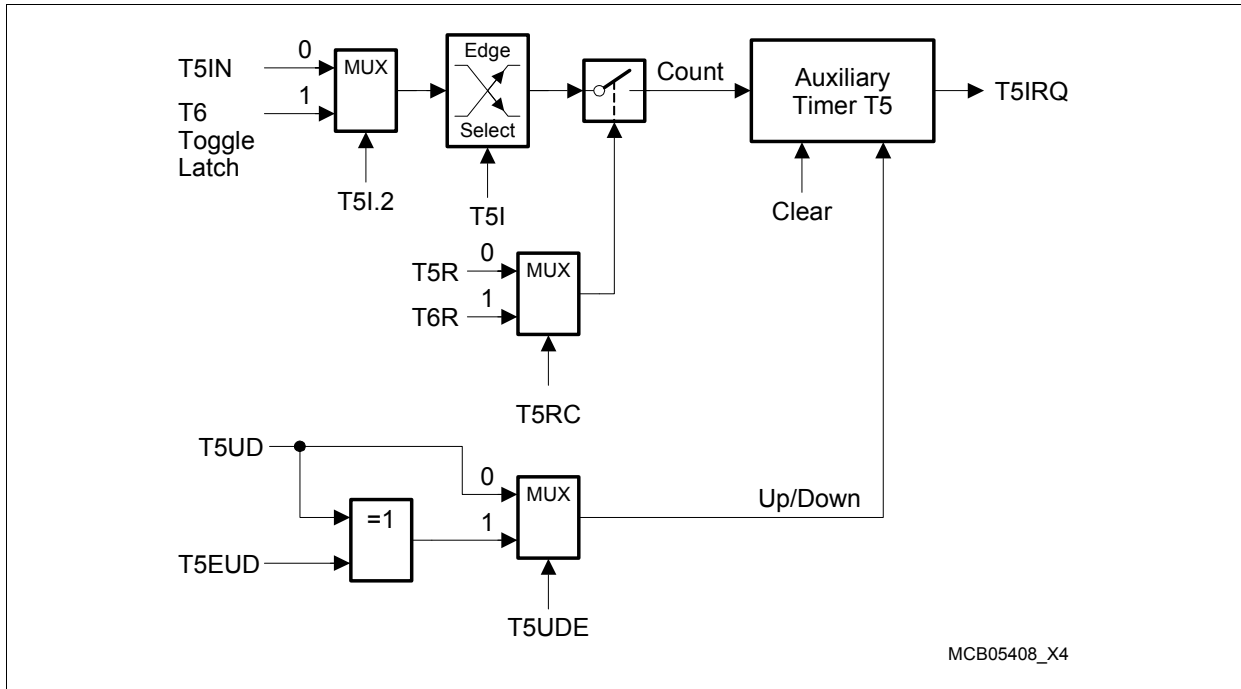
*Note: There is no output toggle latch for T5.*

*Start/stop of the auxiliary timer can be controlled locally or remotely.*

### Timer T5 in Counter Mode

Counter mode for auxiliary timer T5 is selected by setting bitfield T5M in register T5CON to 001<sub>B</sub>. In counter mode, the auxiliary timer can be clocked either by a transition at its external input line T5IN, or by a transition of timer T6's toggle latch T6OTL. The event causing an increment or decrement of a timer can be a positive, a negative, or both a positive and a negative transition at either the respective input pin or at the toggle latch. Bitfield T5I in control register T5CON selects the triggering transition (see [Table 16-12](#)).





**Figure 16-27 Block Diagram of Auxiliary Timer T5 in Counter Mode**

**Table 16-12 GPT2 Auxiliary Timer (Counter Mode) Input Edge Selection**

<b>T5I</b>	<b>Triggering Edge for Counter Increment/Decrement</b>
X00 <sub>B</sub>	None. Counter T5 is disabled
001 <sub>B</sub>	Positive transition (rising edge) on T5IN
010 <sub>B</sub>	Negative transition (falling edge) on T5IN
011 <sub>B</sub>	Any transition (rising or falling edge) on T5IN
101 <sub>B</sub>	Positive transition (rising edge) of T6 toggle latch T6OTL
110 <sub>B</sub>	Negative transition (falling edge) of T6 toggle latch T6OTL
111 <sub>B</sub>	Any transition (rising or falling edge) of T6 toggle latch T6OTL

*Note: Only state transitions of T6OTL which are caused by the overflows/underflows of T6 will trigger the counter function of T5. Modifications of T6OTL via software will NOT trigger the counter function of T5.*

For counter operation, pin T5IN must be configured as input. The maximum input frequency allowed in counter mode depends on the selected prescaler value. To ensure that a transition of the count input signal applied to T5IN is recognized correctly, its level must be held high or low for a minimum number of module clock cycles before it changes. This information can be found in [Section 16.2.6](#).

### Timer Concatenation

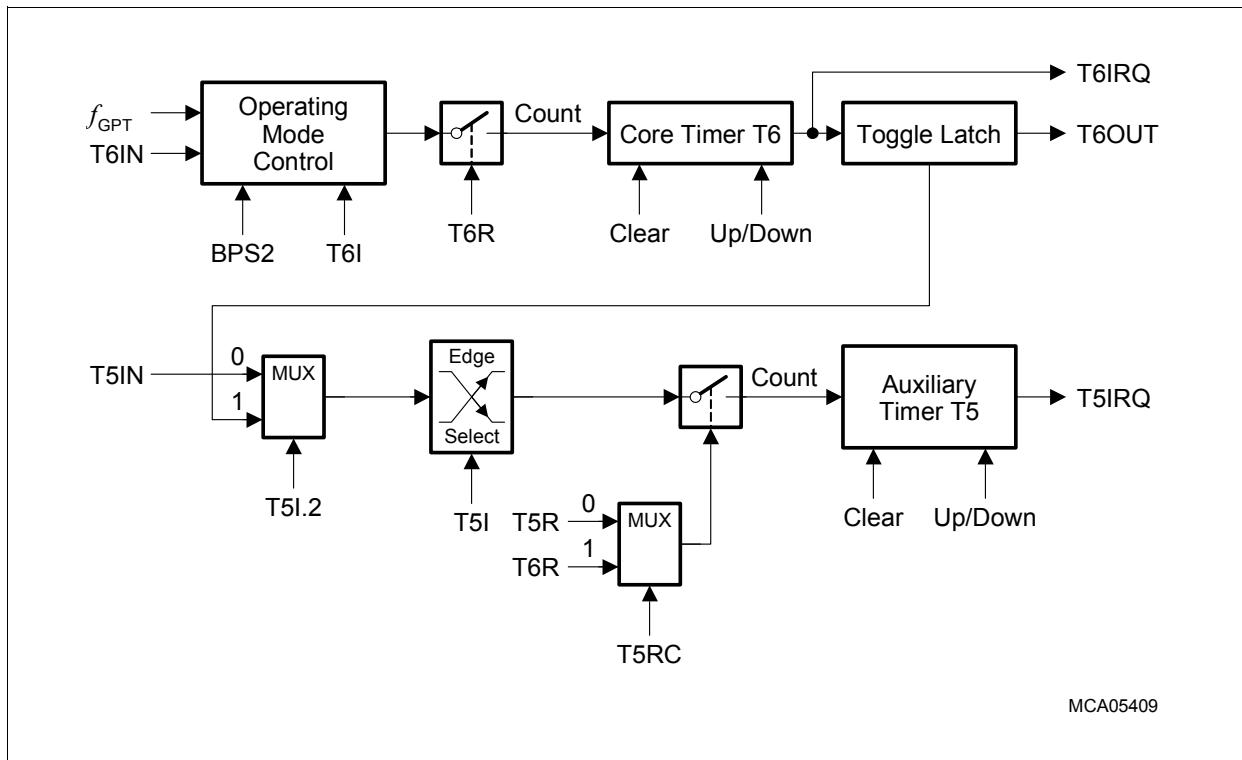
Using the toggle bit T6OTL as a clock source for the auxiliary timer in counter mode concatenates the core timer T6 with the auxiliary timer T5. This concatenation forms either a 32-bit or a 33-bit timer/counter, depending on which transition of T6OTL is selected to clock the auxiliary timer.

- **32-bit Timer/Counter:** If both a positive and a negative transition of T6OTL are used to clock the auxiliary timer, this timer is clocked on every overflow/underflow of the core timer T6. Thus, the two timers form a 32-bit timer.
- **33-bit Timer/Counter:** If either a positive or a negative transition of T6OTL is selected to clock the auxiliary timer, this timer is clocked on every second overflow/underflow of the core timer T6. This configuration forms a 33-bit timer (16-bit core timer + T6OTL + 16-bit auxiliary timer).

As long as bit T6OTL is not modified by software, it represents the state of the internal toggle latch, and can be regarded as part of the 33-bit timer.

The count directions of the two concatenated timers are not required to be the same. This offers a wide variety of different configurations.

T6, which represents the low-order part of the concatenated timer, can operate in timer mode, gated timer mode or counter mode in this case.



**Figure 16-28 Concatenation of Core Timer T6 and Auxiliary Timer T5**

### **16.2.5 GPT2 Register CAPREL Operating Modes**

The Capture/Reload register CAPREL can be used to capture the contents of timer T5, or to reload timer T6. A special mode facilitates the use of register CAPREL for both functions at the same time. This mode allows frequency multiplication. The capture function is triggered by the input pin CAPIN, by GPT1 timer's T3 input lines T3IN and T3EUD, or by read accesses to GPT1 timers. The reload function is triggered by an overflow or underflow of timer T6.

In addition to the capture function, the capture trigger signal can also be used to clear the contents of timers T5 and T6 individually.

The functions of register CAPREL are controlled via several bit(field)s in the timer control registers T5CON and T6CON.

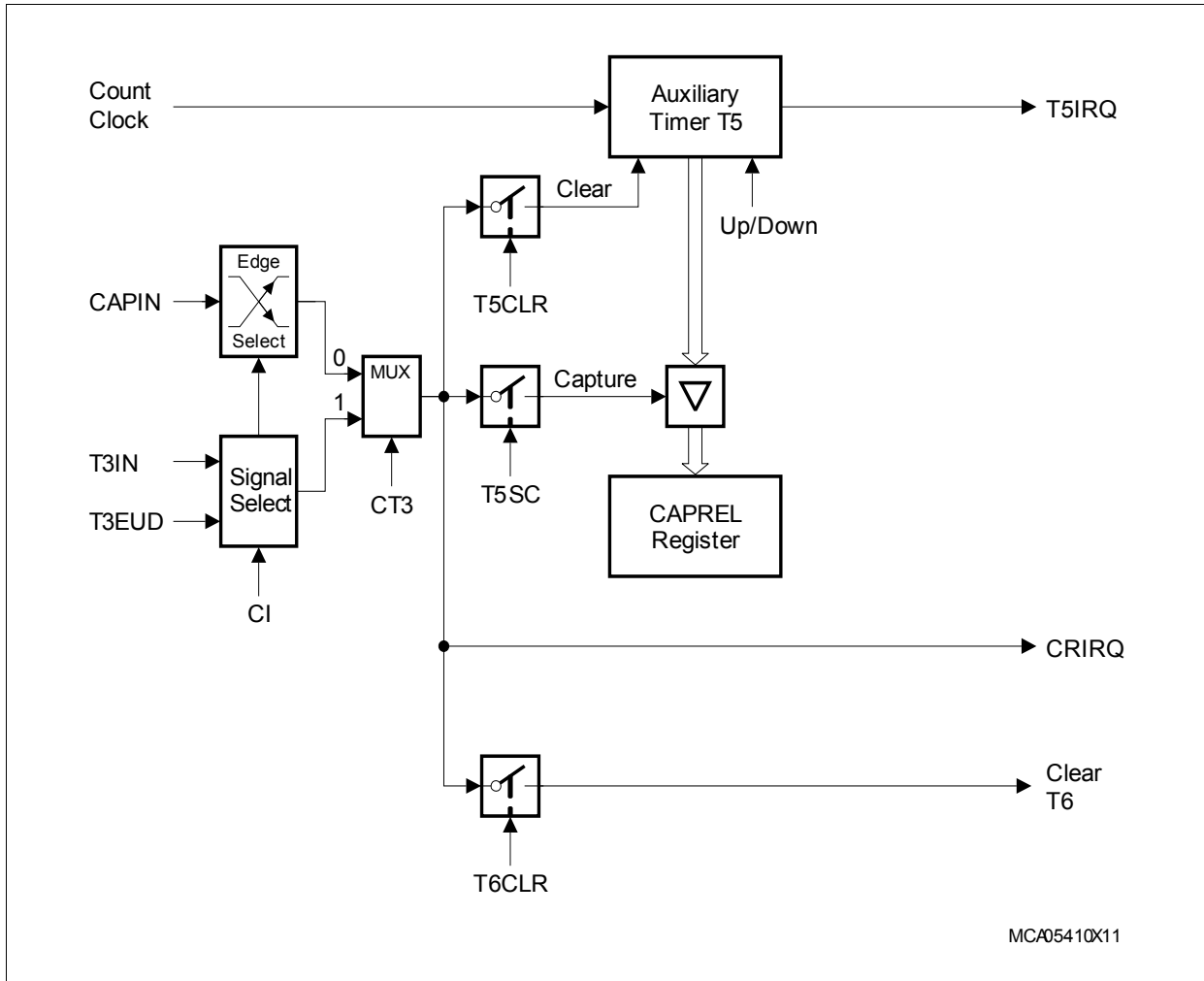
#### **GPT2 Capture/Reload Register CAPREL in Capture Mode**

Capture mode for register CAPREL is selected by setting bit T5SC in control register T5CON (set bitfield CI in register T5CON to a non-zero value to select a trigger signal). In capture mode, the contents of the auxiliary timer T5 are latched into register CAPREL in response to a signal transition at the selected external input pin(s). Bit CT3 selects the external input line CAPIN or the input lines T3IN and/or T3EUD of GPT1 timer T3 as the source for a capture trigger. Either a positive, a negative, or both a positive and a negative transition at line CAPIN can be selected to trigger the capture function, or transitions on input T3IN or input T3EUD or both inputs, T3IN and T3EUD. The active edge is controlled by bitfield CI in register T5CON. **Table 16-13** summarizes these options.

**Table 16-13 CAPREL Register Input Edge Selection**

<b>CT3</b>	<b>CI</b>	<b>Triggering Signal/Edge for Capture Mode</b>
X	00 <sub>B</sub>	None. Capture Mode is disabled.
0	01 <sub>B</sub>	Positive transition (rising edge) on CAPIN. <sup>1)</sup>
0	10 <sub>B</sub>	Negative transition (falling edge) on CAPIN.
0	11 <sub>B</sub>	Any transition (rising or falling edge) on CAPIN.
1	01 <sub>B</sub>	Any transition (rising or falling edge) on T3IN.
1	10 <sub>B</sub>	Any transition (rising or falling edge) on T3EUD.
1	11 <sub>B</sub>	Any transition (rising or falling edge) on T3IN or T3EUD.

1) Rising edge must be selected if capturing is triggered by the internal GPT1 read signals (see register PISEL and **"Combined Capture Modes"** on Page 16-55).



**Figure 16-29 GPT2 Register CAPREL in Capture Mode**

When a selected trigger is detected, the contents of the auxiliary timer T5 are latched into register CAPREL and the interrupt request line CRIRQ is activated. The same event can optionally clear timer T5 and/or timer T6. This option is enabled by bit T5CLR in register T5CON and bit T6CLR in register T6CON, respectively. If TxCLR = 0 the contents of timer Tx is not affected by a capture. If TxCLR = 1 timer Tx is cleared after the current timer T5 value has been latched into register CAPREL.

*Note: Bit T5SC only controls whether or not a capture is performed. If T5SC is cleared the external input pin(s) can still be used to clear timer T5 and/or T6, or as external interrupt input(s). This interrupt is controlled by the CAPREL interrupt control register CRIC.*

When capture triggers T3IN or T3EUD are enabled (CT3 = 1), register CAPREL captures the contents of T5 upon transitions of the selected input(s). These values can be used to measure T3's input signals. This is useful, for example, when T3 operates in

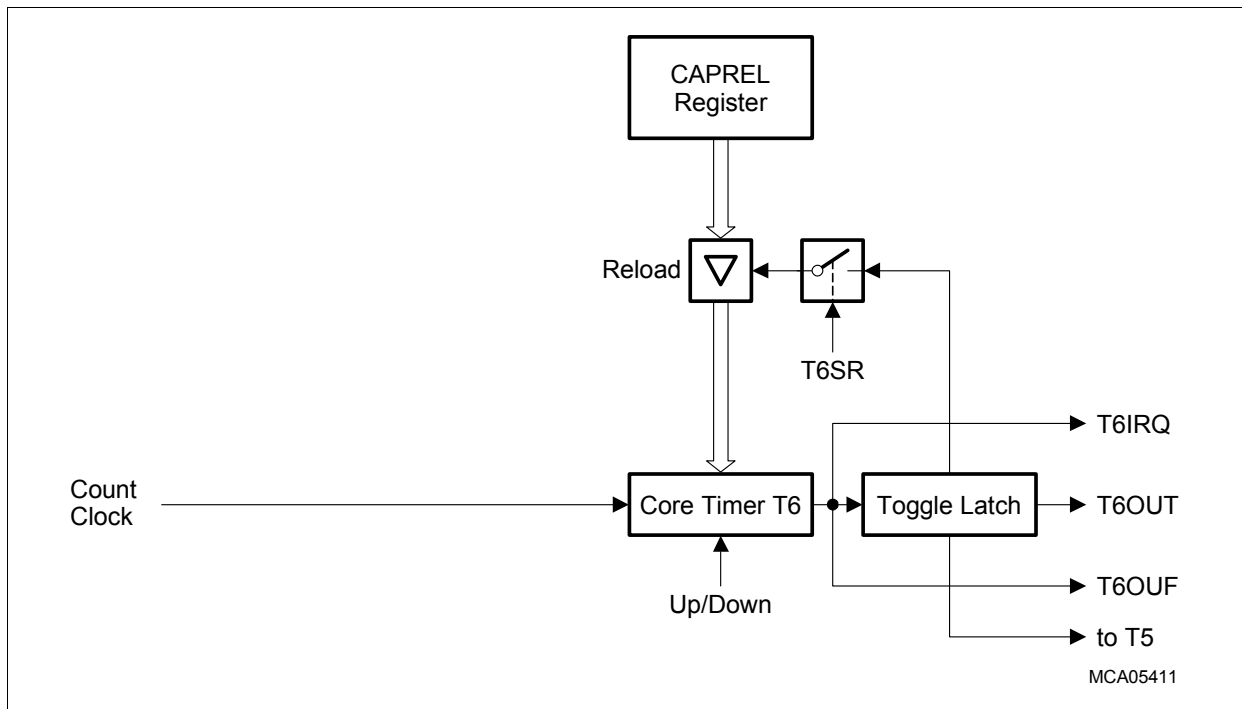
## General Purpose Timer Units

incremental interface mode, in order to derive dynamic information (speed, acceleration) from the input signals.

For capture mode operation, the selected pins CAPIN, T3IN, or T3EUD must be configured as input. To ensure that a transition of a trigger input signal applied to one of these inputs is recognized correctly, its level must be held high or low for a minimum number of module clock cycles, detailed in [Section 16.2.6](#).

### GPT2 Capture/Reload Register CAPREL in Reload Mode

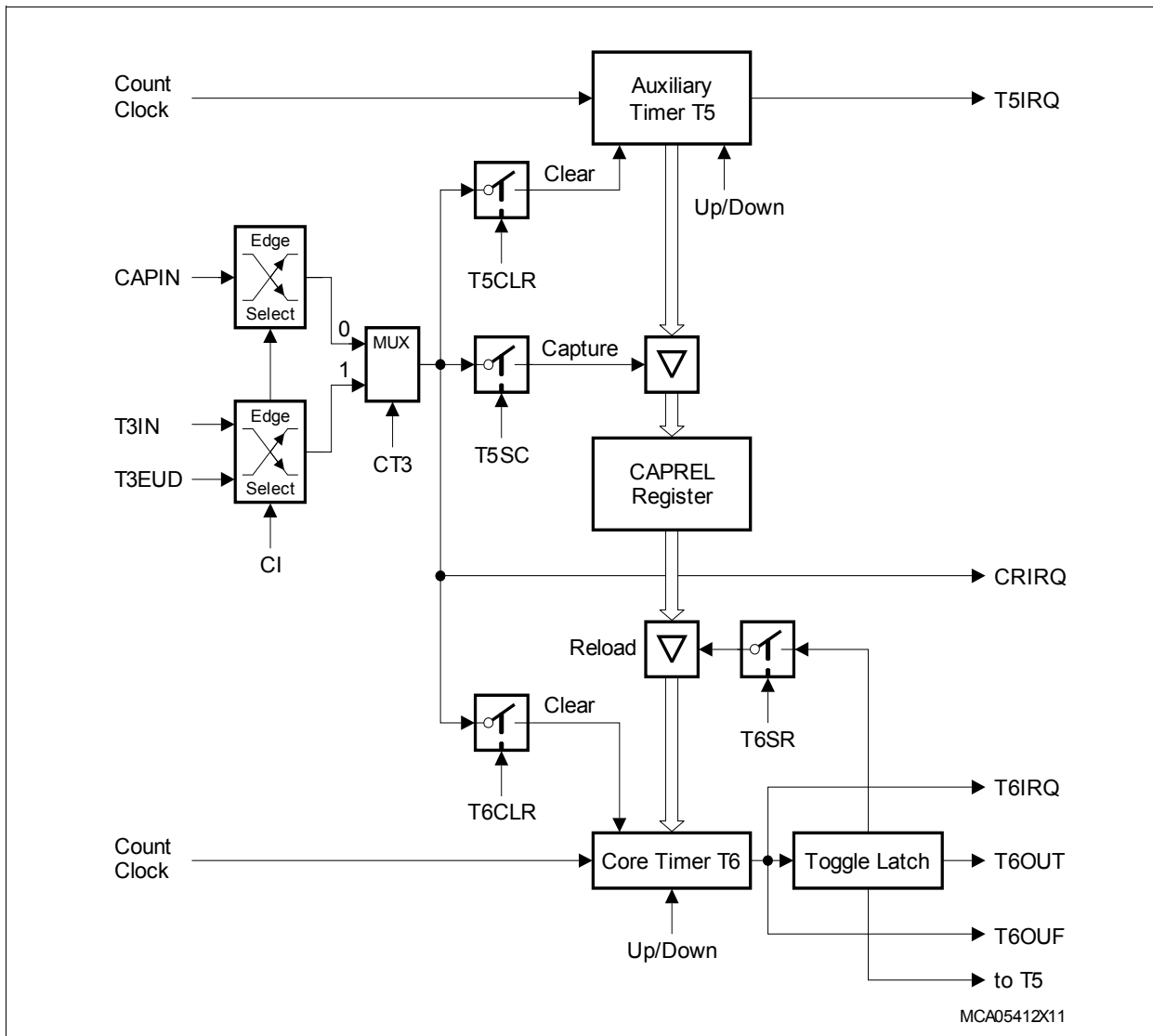
Reload mode for register CAPREL is selected by setting bit T6SR in control register T6CON. In reload mode, the core timer T6 is reloaded with the contents of register CAPREL, triggered by an overflow or underflow of T6. This will not activate the interrupt request line CRIRQ associated with the CAPREL register. However, interrupt request line T6IRQ will be activated, indicating the overflow/underflow of T6.



**Figure 16-30 GPT2 Register CAPREL in Reload Mode**

### GPT2 Capture/Reload Register CAPREL in Capture-And-Reload Mode

Since the reload function and the capture function of register CAPREL can be enabled individually by bits T5SC and T6SR, the two functions can be enabled simultaneously by setting both bits. This feature can be used to generate an output frequency that is a multiple of the input frequency.



**Figure 16-31 GPT2 Register CAPREL in Capture-And-Reload Mode**

This combined mode can be used to detect consecutive external events which may occur aperiodically, but where a finer resolution, that means, more 'ticks' within the time between two external events is required.

For this purpose, the time between the external events is measured using timer T5 and the CAPREL register. Timer T5 runs in timer mode counting up with a frequency of e.g.  $f_{GPT}/32$ . The external events are applied to pin CAPIN. When an external event occurs,

## **General Purpose Timer Units**

the contents of timer T5 are latched into register CAPREL and timer T5 is cleared ( $T5CLR = 1$ ). Thus, register CAPREL always contains the correct time between two events, measured in timer T5 increments. Timer T6, which runs in timer mode counting down with a frequency of e.g.  $f_{GPT}/4$ , uses the value in register CAPREL to perform a reload on underflow. This means, the value in register CAPREL represents the time between two underflows of timer T6, now measured in timer T6 increments. Since (in this example) timer T6 runs 8 times faster than timer T5, it will underflow 8 times within the time between two external events. Thus, the underflow signal of timer T6 generates 8 'ticks'. Upon each underflow, the interrupt request line T6IRQ will be activated and bit T6OTL will be toggled. The state of T6OTL may be output on pin T6OUT. This signal has 8 times more transitions than the signal which is applied to pin CAPIN.

*Note: The underflow signal of Timer T6 can furthermore be used to clock one or more of the timers of the CAPCOM units, which gives the user the possibility to set compare events based on a finer resolution than that of the external events. This connection is accomplished via signal T6OUF.*

### **Capture Correction**

A certain deviation of the output frequency is generated by the fact that timer T5 will count actual time units (e.g. T5 running at 1 MHz will count up to the value  $64_H/100_D$  for a 10 kHz input signal), while T6OTL will only toggle upon an underflow of T6 (i.e. the transition from  $0000_H$  to  $FFFF_H$ ). In the above mentioned example, T6 would count down from  $64_H$ , so the underflow would occur after 101 timing ticks of T6. The actual output frequency then is 79.2 kHz, instead of the expected 80 kHz.

Another possibility is to use T6 overflows. In this case, T5 counts down and T6 counts up. Upon a signal transition on pin CAPIN, the count value in T5 is captured into CAPREL and T5 is cleared to  $0000_H$ . In its next clock cycle, T5 underflows to  $FFFF_H$ , and continues to count down with the following clocks. T6 is reloaded from CAPREL upon an overflow, and continues to count up with its following clock cycles (8 times faster in the above example). In this case, T5 and T6 count the same number of steps with their respective internal count frequency.

In the above example, T5 running at 1 MHz will count down to the value  $FF9C_H/-100_D$  for a 10 kHz input signal applied at CAPIN, while T6 counts up from  $FF9C_H$  through  $FFFF_H$  to  $0000_H$ . So the overflow occurs after 100 timing ticks of T6, and the actual output frequency at T6OUT then is the expected 80 kHz.

However, in this case CAPREL does not directly contain the time between two CAPIN events, but rather its 2's complement. Software will have to convert this value, if it is required for the operation.

### **Combined Capture Modes**

For incremental interface applications in particular, several timer features can be combined to obtain dynamic information such as speed, acceleration, or deceleration. The current position itself can be obtained directly from the timer register (T2, T3, T4).

The time information to determine the dynamic parameters is generated by capturing the contents of the free-running timer T5 into register CAPREL. Two trigger sources for this event can be selected:

- Capture trigger on sensor signal transitions
- Capture trigger on position read operations

Capturing on sensor signal transitions is available for timer T3 inputs. This mode is selected by setting bit CT3 and selecting the intended signal(s) via bitfield CI in register T5CON. CAPREL then indicates the time between two selected transitions (measured in T5 counts).

Capturing on position read operations is available for timers T2, T3, and T4. This mode is selected by clearing bit CT3 and selecting the rising edge via bitfield CI in register T5CON. Bitfield ISCAPIN in register PISEL then selects either a read access from T3 or a read access from any of T2 or T3 or T4. CAPREL then indicates the time between two read accesses.

These operating modes directly support the measurement of position and rotational speed. Acceleration and deceleration can then be determined by evaluating subsequent speed measurements.



## 16.2.6 GPT2 Clock Signal Control

All actions within the timer block GPT2 are triggered by transitions of its basic clock. This basic clock is derived from the system clock by a basic block prescaler, controlled by bitfield BPS2 in register T6CON (see [Figure 16-20](#)). The count clock can be generated in two different ways:

- **Internal count clock**, derived from GPT2's basic clock via a programmable prescaler, is used for (gated) timer mode.
- **External count clock**, derived from the timer's input pin(s), is used for counter mode.

For both ways, the basic clock determines the maximum count frequency and the timer's resolution:

**Table 16-14 Basic Clock Selection for Block GPT2**

Block Prescaler <sup>1)</sup>	BPS2 = 01 <sub>B</sub>	BPS2 = 00 <sub>B</sub> <sup>2)</sup>	BPS2 = 11 <sub>B</sub>	BPS2 = 10 <sub>B</sub>
<b>Prescaling Factor for GPT2: F(BPS2)</b>	F(BPS2) = 2	F(BPS2) = 4	F(BPS2) = 8	F(BPS2) = 16
<b>Maximum External Count Frequency</b>	$f_{\text{GPT}}/4$	$f_{\text{GPT}}/8$	$f_{\text{GPT}}/16$	$f_{\text{GPT}}/32$
<b>Input Signal Stable Time</b>	$2 \times t_{\text{GPT}}$	$4 \times t_{\text{GPT}}$	$8 \times t_{\text{GPT}}$	$16 \times t_{\text{GPT}}$

1) Please note the non-linear encoding of bitfield BPS2.

2) Default after reset.

### Internal Count Clock Generation

In timer mode and gated timer mode, the count clock for each GPT2 timer is derived from the GPT2 basic clock by a programmable prescaler, controlled by bitfield TxI in the respective timer's control register TxCON.

The count frequency  $f_{\text{Tx}}$  for a timer Tx and its resolution  $r_{\text{Tx}}$  are scaled linearly with lower clock frequencies, as can be seen from the following formula:

$$f_{\text{Tx}} = \frac{f_{\text{GPT}}}{F(\text{BPS2}) \times 2^{<\text{TxI}>}} \quad r_{\text{Tx}}[\mu\text{s}] = \frac{F(\text{BPS2}) \times 2^{<\text{TxI}>}}{f_{\text{GPT}}[\text{MHz}]} \quad [16.2]$$

The effective count frequency depends on the common module clock prescaler factor F(BPS2) as well as on the individual input prescaler factor  $2^{<\text{TxI}>}$ . [Table 16-15](#) summarizes the resulting overall divider factors for a GPT2 timer that result from these cascaded prescalers.

**Table 16-15 GPT2 Overall Prescaler Factors for Internal Count Clock**

Individual Prescaler for Tx	Common Prescaler for Module Clock <sup>1)</sup>			
	BPS2 = 01 <sub>B</sub>	BPS2 = 00 <sub>B</sub>	BPS2 = 11 <sub>B</sub>	BPS2 = 10 <sub>B</sub>
<b>Txl = 000<sub>B</sub></b>	2	4	8	16
<b>Txl = 001<sub>B</sub></b>	4	8	16	32
<b>Txl = 010<sub>B</sub></b>	8	16	32	64
<b>Txl = 011<sub>B</sub></b>	16	32	64	128
<b>Txl = 100<sub>B</sub></b>	32	64	128	256
<b>Txl = 101<sub>B</sub></b>	64	128	256	512
<b>Txl = 110<sub>B</sub></b>	128	256	512	1024
<b>Txl = 111<sub>B</sub></b>	256	512	1024	2048

1) Please note the non-linear encoding of bitfield BPS2.

**Table 16-16** lists a timer's parameters (such as count frequency, resolution, and period) resulting from the selected overall prescaler factor and the applied system frequency. Note that some numbers may be rounded.

**Table 16-16 GPT2 Timer Parameters**

System Clock = 10 MHz			Overall Divider Factor	System Clock = 40 MHz		
Frequency	Resolution	Period		Frequency	Resolution	Period
5.0 MHz	200 ns	13.11 ms	2	20.0 MHz	50 ns	3.28 ms
2.5 MHz	400 ns	26.21 ms	4	10.0 MHz	100 ns	6.55 ms
1.25 MHz	800 ns	52.43 ms	8	5.0 MHz	200 ns	13.11 ms
625.0 kHz	1.6 µs	104.9 ms	16	2.5 MHz	400 ns	26.21 ms
312.5 kHz	3.2 µs	209.7 ms	32	1.25 MHz	800 ns	52.43 ms
156.25 kHz	6.4 µs	419.4 ms	64	625.0 kHz	1.6 µs	104.9 ms
78.125 kHz	12.8 µs	838.9 ms	128	312.5 kHz	3.2 µs	209.7 ms
39.06 kHz	25.6 µs	1.678 s	256	156.25 kHz	6.4 µs	419.4 ms
19.53 kHz	51.2 µs	3.355 s	512	78.125 kHz	12.8 µs	838.9 ms
9.77 kHz	102.4 µs	6.711 s	1024	39.06 kHz	25.6 µs	1.678 s
4.88 kHz	204.8 µs	13.42 s	2048	19.53 kHz	51.2 µs	3.355 s

### External Count Clock Input

The external input signals of the GPT2 block are sampled with the GPT2 basic clock (see [Figure 16-20](#)). To ensure that a signal is recognized correctly, its current level (high or low) must be held active for at least one complete sampling period, before changing. A signal transition is recognized if two subsequent samples of the input signal represent different levels. Therefore, a minimum of two basic clock periods are required for the sampling of an external input signal. Thus, the maximum frequency of an input signal must not be higher than half the basic clock.

**Table 16-17** summarizes the resulting requirements for external GPT2 input signals.

**Table 16-17 GPT2 External Input Signal Limits**

System Clock = 10 MHz		Input Freq. Factor	GPT2 Divider BPS2	Input Phase Duration	System Clock = 40 MHz	
Max. Input Frequency	Min. Level Hold Time				Max. Input Frequency	Min. Level Hold Time
2.5 MHz	200 ns	$f_{\text{GPT}}/4$	01 <sub>B</sub>	$2 \times t_{\text{GPT}}$	10.0 MHz	50 ns
1.25 MHz	400 ns	$f_{\text{GPT}}/8$	00 <sub>B</sub>	$4 \times t_{\text{GPT}}$	5.0 MHz	100 ns
625.0 kHz	800 ns	$f_{\text{GPT}}/16$	11 <sub>B</sub>	$8 \times t_{\text{GPT}}$	2.5 MHz	200 ns
312.5 kHz	1.6 $\mu$ s	$f_{\text{GPT}}/32$	10 <sub>B</sub>	$16 \times t_{\text{GPT}}$	1.25 MHz	400 ns

These limitations are valid for all external input signals to GPT2, including the external count signals in counter mode and the gate input signals in gated timer mode.

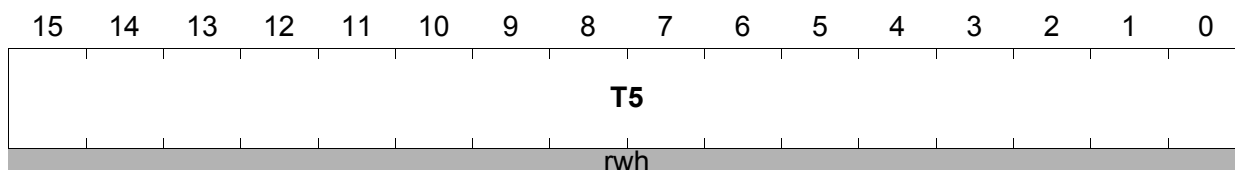
## 16.2.7 GPT2 Timer Registers

### GPT12E\_T5

**Timer 5 Count Register**

**SFR (FE46<sub>H</sub>/23<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



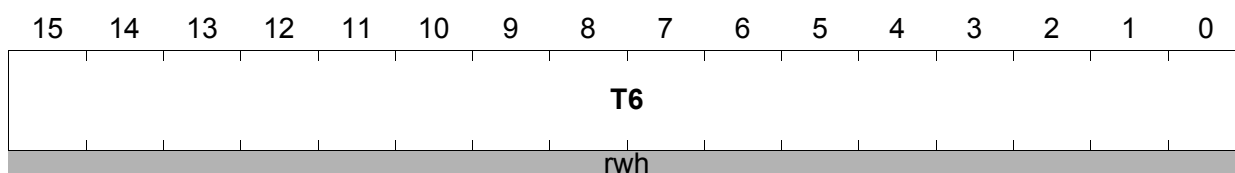
Field	Bits	Typ	Description
<b>T5</b>	[15:0]	rwh	<b>Timer T5 Current Value</b> Contains the current value of the timer T5

### GPT12E\_T6

**Timer 6 Count Register**

**SFR (FE48<sub>H</sub>/24<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



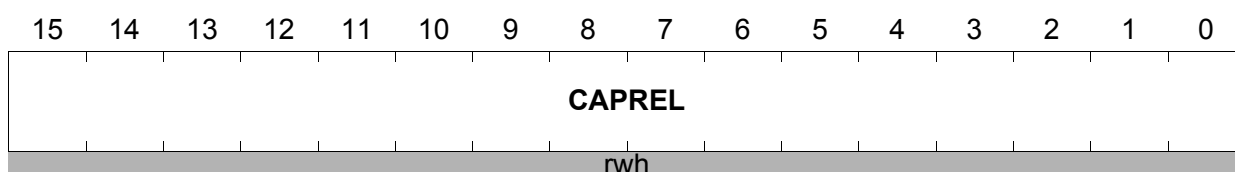
Field	Bits	Typ	Description
<b>T6</b>	[15:0]	rwh	<b>Timer T6 Current Value</b> Contains the current value of the timer T6

### GPT12E\_CAPREL

**Capture/Reload Register**

**SFR (FE4A<sub>H</sub>/25<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



Field	Bits	Typ	Description
<b>CAPREL</b>	[15:0]	rwh	<b>Current reload value or Captured value</b> Contains the current value of the CAPREL register

### **16.2.8 Interrupt Control for GPT2 Timers and CAPREL**

When a timer overflows from  $FFFF_H$  to  $0000_H$  (when counting up), or when it underflows from  $0000_H$  to  $FFFF_H$  (when counting down), its interrupt request flag in register GPT12E\_TxIC ( $x = 5, 6$ ) will be set. Whenever a transition according to the selection in bit field CI is detected at pin CAPIN, interrupt request flag in register GPT12E\_CRIC is set. Setting any request flag will cause an interrupt to the respective timer or CAPREL interrupt vector or trigger a PEC service, if the respective interrupt enable bit is set.

There is an interrupt control register for each of the two timers (T5, T6) and for the CAPREL register. All interrupt control registers have the same structure described in section Interrupt Control.

### 16.3 Miscellaneous Registers

The following registers are not assigned to a specific timer block. They control general functions and/or give general information.

Register GPT12E\_PISEL selects timer input signal from several sources under software control.

#### GPT12E\_PISEL

**Port Input Select Register**

**SFR (FE4C<sub>H</sub>/26<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ISCAPIN	IST6 EUD	IST6 IN	IST5 EUD	IST5 IN	IST4EUD	IST4IN	IST3EUD	IST3IN	IST2 EUD	IST2 IN					
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Typ	Description
IST2IN	0	rw	<b>Input Select for T2IN</b> 0 <sub>B</sub> Signal T2INA is selected 1 <sub>B</sub> Signal T2INB is selected
IST2EUD	1	rw	<b>Input Select for T2EUD</b> 0 <sub>B</sub> Signal T2EUDA is selected 1 <sub>B</sub> Signal T2EUDB is selected
IST3IN	[3:2]	rw	<b>Input Select for T3IN</b> 00 <sub>B</sub> Signal T3INA is selected 01 <sub>B</sub> Signal T3INB is selected 10 <sub>B</sub> Signal T3INC is selected 11 <sub>B</sub> Signal T3IND is selected
IST3EUD	[5:4]	rw	<b>Input Select for T3EUD</b> 00 <sub>B</sub> Signal T3EUDA is selected 01 <sub>B</sub> Signal T3EUDB is selected 10 <sub>B</sub> Signal T3EUDC is selected 11 <sub>B</sub> Signal T3EUIDD is selected
IST4IN	[7:6]	rw	<b>Input Select for T4IN</b> 00 <sub>B</sub> Signal T4INA is selected 01 <sub>B</sub> Signal T4INB is selected 10 <sub>B</sub> Signal T4INC is selected 11 <sub>B</sub> Signal T4IND is selected

**General Purpose Timer Units**

<b>Field</b>	<b>Bits</b>	<b>Typ</b>	<b>Description</b>
<b>IST4EUD</b>	[9:8]	rw	<b>Input Select for T4EUD</b> 00 <sub>B</sub> Signal T4EUDA is selected 01 <sub>B</sub> Signal T4EUDB is selected 10 <sub>B</sub> Signal T4EUDC is selected 11 <sub>B</sub> Signal T4EUDD is selected
<b>IST5IN</b>	10	rw	<b>Input Select for T5IN</b> 0 <sub>B</sub> Signal T5INA is selected 1 <sub>B</sub> Signal T5INB is selected
<b>IST5EUD</b>	11	rw	<b>Input Select for T5EUD</b> 0 <sub>B</sub> Signal T5EUDA is selected 1 <sub>B</sub> Signal T5EUDB is selected
<b>IST6IN</b>	12	rw	<b>Input Select for T6IN</b> 0 <sub>B</sub> Signal T6INA is selected 1 <sub>B</sub> Signal T6INB is selected
<b>IST6EUD</b>	13	rw	<b>Input Select for T6EUD</b> 0 <sub>B</sub> Signal T6EUDA is selected 1 <sub>B</sub> Signal T6EUDB is selected
<b>ISCAPIN</b>	[15:14]	rw	<b>Input Select for CAPIN</b> 00 <sub>B</sub> Signal CAPINA is selected 01 <sub>B</sub> Signal CAPINB is selected 10 <sub>B</sub> Signal CAPINC (Read trigger from T3) is selected 11 <sub>B</sub> Signal CAPIND (Read trigger from T2 or T3 or T4) is selected

*Note: PISEL's reset value represents the connections available in previous versions.*

**General Purpose Timer Units**

Register GPT12E\_KSCCFG controls the overall operation of the timer module.

**GPT12E\_KSCCFG**

**Kernel State Configuration Register**

**SFR(FE1C<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>BP COM</b>	<b>0</b>	<b>COMCFG</b>	<b>BP SUM</b>	<b>0</b>	<b>SUMCFG</b>	<b>BP NOM</b>	<b>0</b>	<b>NOMCFG</b>	<b>0</b>	<b>BP MOD EN</b>	<b>MOD EN</b>				
w	r	rw	w	r	rw	w	r	rw	r	w	rw				

Field	Bits	Type	Description
<b>MODEN</b>	0	rw	<p><b>Module Enable</b></p> <p>This bit enables the module kernel clock and the module functionality.</p> <p>0<sub>B</sub> The module is switched off. It does not react on mode control actions and the module clock is switched off immediately (without stop condition). The module does not react on read accesses and ignores write accesses.</p> <p>1<sub>B</sub> The module is switched on and can operate. After writing 1 to MODEN, it is recommended to read register KSCCFG to avoid pipeline effects in the control block before accessing other GPT registers.</p> <p><i>Note: This bit is reset by an application reset.</i></p>
<b>BPMODEN</b>	1	w	<p><b>Bit Protection for MODEN</b></p> <p>This bit enables the write access to the bit MODEN. It always reads 0. It is only active during the write access cycle.</p> <p>0<sub>B</sub> MODEN is not changed.</p> <p>1<sub>B</sub> MODEN is updated with the written value.</p> <p><i>Note: This bit is reset by an application reset.</i></p>



**General Purpose Timer Units**

Field	Bits	Type	Description
<b>NOMCFG</b>	[5:4]	rw	<b>Normal Operation Mode Configuration</b> This bit field defines the kernel mode applied in normal operation mode. 0X <sub>B</sub> The module is switched on. 1X <sub>B</sub> The module is switched off. This field is taken into account for CR = 00 or 11. <i>Note: This bit is reset by an application reset.</i>
<b>BPNOM</b>	7	w	<b>Bit Protection for NOMCFG</b> This bit enables the write access to the bit field NOMCFG. It always reads 0. It is only active during the write access cycle. 0 <sub>B</sub> NOMCFG is not changed. 1 <sub>B</sub> NOMCFG is updated with the written value. <i>Note: This bit is reset by an application reset.</i>
<b>SUMCFG</b>	[9:8]	rw	<b>Suspend Mode Configuration</b> This bit field defines the kernel mode applied in suspend mode. 0X <sub>B</sub> The module is switched on. 1X <sub>B</sub> The module is switched off. This field is taken into account for CR = 01. <i>Note: This bit is reset by a debug reset.</i>
<b>BPSUM</b>	11	w	<b>Bit Protection for SUMCFG</b> This bit enables the write access to the bit field SUMCFG. It always reads 0. It is only active during the write access cycle. 0 <sub>B</sub> SUMCFG is not changed. 1 <sub>B</sub> SUMCFG is updated with the written value. <i>Note: This bit is reset by a debug reset.</i>
<b>COMCFG</b>	[13:12]	rw	<b>Clock Off Mode Configuration</b> This bit field defines the kernel mode applied in clock off mode. 0X <sub>B</sub> The module is switched on. 1X <sub>B</sub> The module is switched off. This field is taken into account for CR = 10. <i>Note: This bit is reset by an application reset.</i>

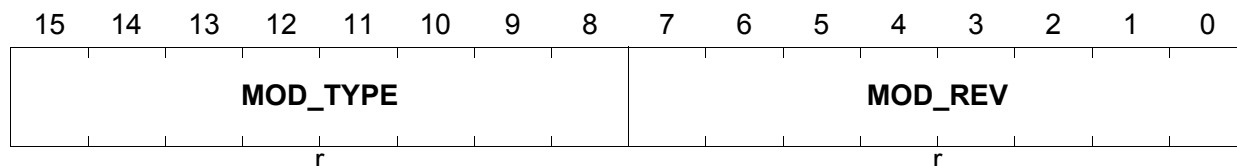
**General Purpose Timer Units**

Field	Bits	Type	Description
<b>BPCOM</b>	15	w	<b>Bit Protection for COMCFG</b> This bit enables the write access to the bit field COMCFG. It always reads 0. It is only active during the write access cycle. 0 <sub>B</sub> COMCFG is not changed. 1 <sub>B</sub> COMCFG is updated with the written value. <i>Note: This bit is reset by an application reset.</i>
<b>0</b>	[3:2], 6, 10, 14	r	<b>Reserved;</b> returns 0 if read; should be written with 0;

Register GPT12E\_ID indicates the module version.

### GPT12E\_ID

**Module Identification Register**      **MEM (FFE6<sub>H</sub>)**      **Reset Value: 58XX<sub>H</sub>**



Field	Bits	Type	Description
<b>MOD_REV</b>	[7:0]	r	<b>Module Revision Number</b> MOD_REV defines the revision number. The value of a module revision starts with 01 <sub>H</sub> (first revision).
<b>MOD_TYPE</b>	[15:8]	r	<b>Module Identification Number</b> This bitfield defines the module identification number (58 <sub>H</sub> = GPT12E).

## 16.4 Register Table

**Table 16-18** shows all registers which are required for programming of the GPT12E module. It summarizes the GPT12E kernel registers and the module external registers and defines their addresses and reset values.

**Table 16-18 GPT12E Module Register Summary**

Name	Description	Address		Reset Value
		16-Bit	8-Bit	

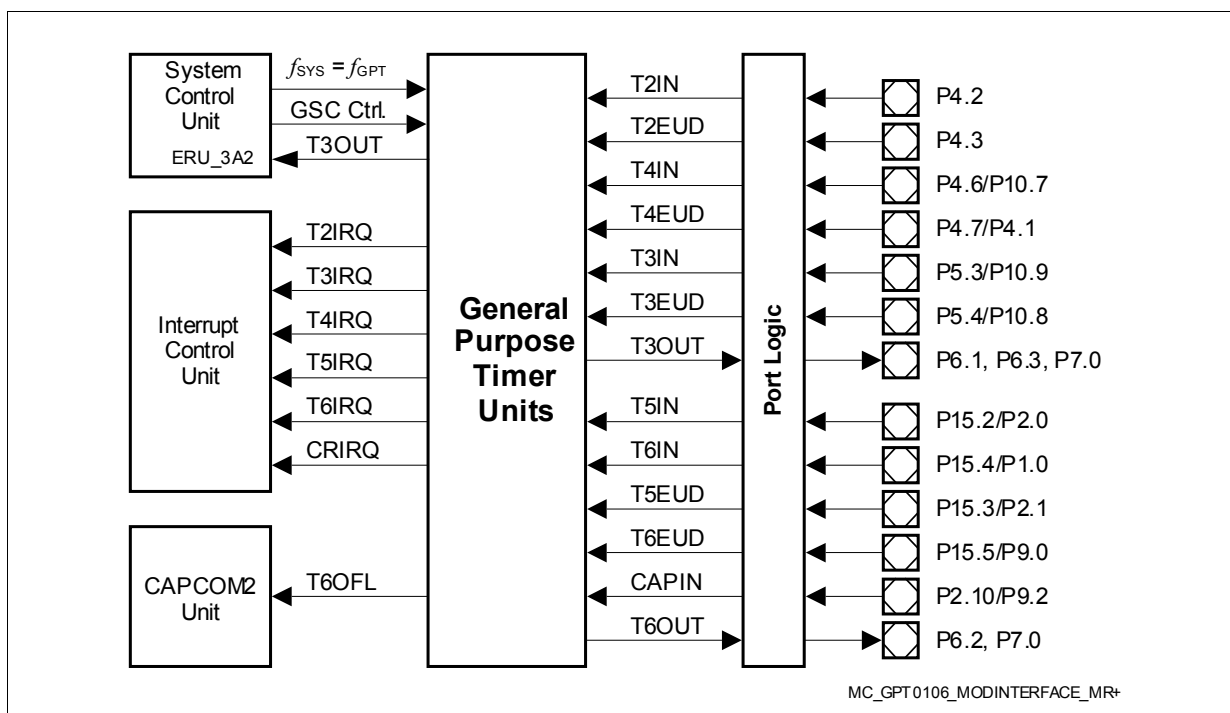
### General Purpose Timer Unit (GPT12E)

<b>GPT12E_ID</b>	GPT12E Module ID Register	FFE6 <sub>H</sub>	F3 <sub>H</sub>	58XX <sub>H</sub>
<b>GPT12E_PISEL</b>	Input Signal Selection	FE4C <sub>H</sub>	26 <sub>H</sub>	0000 <sub>H</sub>
<b>GPT12E_T2CON</b>	GPT12E Timer 2 Control Register	FF40 <sub>H</sub>	A0 <sub>H</sub>	0000 <sub>H</sub>
<b>GPT12E_T3CON</b>	GPT12E Timer 3 Control Register	FF42 <sub>H</sub>	A1 <sub>H</sub>	0000 <sub>H</sub>
<b>GPT12E_T4CON</b>	GPT12E Timer 4 Control Register	FF44 <sub>H</sub>	A2 <sub>H</sub>	0000 <sub>H</sub>
<b>GPT12E_T5CON</b>	GPT12E Timer 5 Control Register	FF46 <sub>H</sub>	A3 <sub>H</sub>	0000 <sub>H</sub>
<b>GPT12E_T6CON</b>	GPT12E Timer 6 Control Register	FF48 <sub>H</sub>	A4 <sub>H</sub>	0000 <sub>H</sub>
<b>GPT12E_CAPREL</b>	GPT12E Capture/Reload Register	FE4A <sub>H</sub>	25 <sub>H</sub>	0000 <sub>H</sub>
<b>GPT12E_T2</b>	GPT12E Timer 2 Register	FE40 <sub>H</sub>	20 <sub>H</sub>	0000 <sub>H</sub>
<b>GPT12E_T3</b>	GPT12E Timer 3 Register	FE42 <sub>H</sub>	21 <sub>H</sub>	0000 <sub>H</sub>
<b>GPT12E_T4</b>	GPT12E Timer 4 Register	FE44 <sub>H</sub>	22 <sub>H</sub>	0000 <sub>H</sub>
<b>GPT12E_T5</b>	GPT12E Timer 5 Register	FE46 <sub>H</sub>	23 <sub>H</sub>	0000 <sub>H</sub>
<b>GPT12E_T6</b>	GPT12E Timer 6 Register	FE48 <sub>H</sub>	24 <sub>H</sub>	0000 <sub>H</sub>
<b>GPT12E_T2IC</b>	GPT12E Timer 2 Interrupt Control Register	FF60 <sub>H</sub>	B0 <sub>H</sub>	0000 <sub>H</sub>
<b>GPT12E_T3IC</b>	GPT12E Timer 3 Interrupt Control Register	FF62 <sub>H</sub>	B1 <sub>H</sub>	0000 <sub>H</sub>
<b>GPT12E_T4IC</b>	GPT12E Timer 4 Interrupt Control Register	FF64 <sub>H</sub>	B2 <sub>H</sub>	0000 <sub>H</sub>
<b>GPT12E_T5IC</b>	GPT12E Timer 5 Interrupt Control Register	FF66 <sub>H</sub>	B3 <sub>H</sub>	0000 <sub>H</sub>
<b>GPT12E_T6IC</b>	GPT12E Timer 6 Interrupt Control Register	FF68 <sub>H</sub>	B4 <sub>H</sub>	0000 <sub>H</sub>
<b>GPT12E_CRIC</b>	GPT12E CAPREL Interrupt Control Register	FF6A <sub>H</sub>	B5 <sub>H</sub>	0000 <sub>H</sub>

## 16.5 Interfaces of the GPT Module

Besides the described intra-module connections, the timer unit blocks GPT1 and GPT2 are connected to their environment in two basic ways:

- **Internal connections** interface the timers with on-chip resources such as clock generation unit, interrupt controller, or other timers.  
 The GPT module is clocked with the XE16xyM system clock, so  $f_{GPT} = f_{SYS}$ .
- **External connections** interface the timers with external resources via port pins.



**Figure 16-32 GPT Module Interfaces**

*Note: The GPT12E output signal 'T6OFL' is connected to the CAPCOM2 input 'TOUF' and to the GSC.*

**Table 16-19 GPT Digital Connections in XE16xyM**

Signal	from/to Module	I/O to GPT	Can be used to/as
T2INA	P4.2	I	count input signals for timer T2
T2INB	0	I	
T2EUDA	P4.3	I	direction input signals for timer T2
T2EUDB	0	I	
T2IRQ	ICU	O	interrupt request from timer T2

**General Purpose Timer Units**

**Table 16-19 GPT Digital Connections in XE16xyM (cont'd)**

<b>Signal</b>	<b>from/to Module</b>	<b>I/O to GPT</b>	<b>Can be used to/as</b>
T3INA	P5.3	I	count input signals for timer T3
T3INB	P10.9	I	
T3INC	0	I	
T3IND	0	I	
T3EUDA	P5.4	I	direction input signals for timer T3
T3EADB	P10.8	I	
T3EUDC	0	I	
T3EUDD	0	I	
T3OUT	P7.0	O	count output signal for timer T3
	P6.1	O	
	P6.3	O	
	ERU_3A2 (SCU)	O	
T3IRQ	ICU	O	interrupt request from timer T3
T4INA	P4.6	I	count input signals for timer T4
T4INB	P10.7	I	
T4INC	0	I	
T4IND	0	I	
T4EUDA	P4.7	I	direction input signals for timer T4
T4EADB	P4.1	I	
T4EUDC	0	I	
T4EUDD	0	I	
T4IRQ	ICU	O	interrupt request from timer T4
T5INA	P15.2	I	count input signals for timer T5
T5INB	P2.0	I	
T5EUDA	P15.3	I	direction input signals for timer T5
T5EADB	P2.1	I	
T5IRQ	ICU	O	interrupt request from timer T5
T6INA	P15.4	I	count input signals for timer T6
T6INB	P1.0	I	

**Table 16-19 GPT Digital Connections in XE16xyM (cont'd)**

Signal	from/to Module	I/O to GPT	Can be used to/as
T6EUDA	P15.5	I	direction input signals for timer T6
T6EUDB	P9.0	I	
T6OUT	P7.0	O	count output signal for timer T6
	P6.2	O	
T6IRQ	ICU	O	interrupt request from timer T6
T6OFL	CC2_TOUF, SCU (GSC)	O	over/under-flow signal from timer T6
CAPINA	P2.10	I	input capture signals
CAPINB	P9.2	I	
CAPINC	Read trigger from T3	I	
CAPIND	Read trigger from T2 or T3 or T4	I	
CRIRQ	ICU	O	interrupt request from capture control

### Port Control

Port pins to be used for timer input signals must be switched to input (bitfield PC in the respective port control register must be 0xxx<sub>B</sub>) and must be selected via register PISEL. Port pins to be used for timer output signals must be switched to output and the alternate timer output signal must be selected (bitfield PC in the respective port control register must be 1xxx<sub>B</sub>).

For the inputs assigned to Ports 5 and 15 (uni-directional input ports), the digital input must be enabled by the digital input control registers (PxDIDIS).

*Note: The P5/P15 inputs are directly enabled after reset by the default values of registers P5DIDIS and P15DIDIS.*

*Note: For a description of the port control registers, please refer to chapter "Parallel Ports".*

### Interrupts

The GPT12 has six interrupt request lines.

Interrupt nodes to be used for timer interrupt requests must be enabled and programmed to a specific interrupt level.

### **Debug Details**

While the module GPT is disabled, its registers can still be read. While disabled the following registers can be written: PISEL, T5CON.

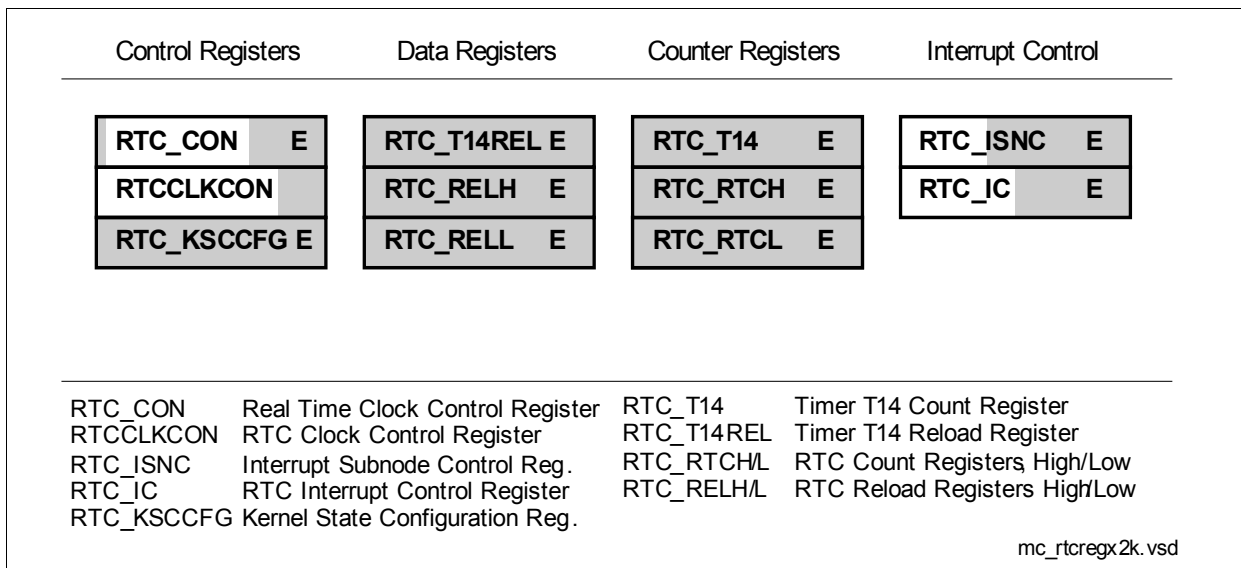


## 17 Real Time Clock

The Real Time Clock (RTC) module of the XE16xyM basically consists of a chain of prescalers and timers. Its count clock is derived from the auxiliary oscillator or from the prescaled main oscillator. The RTC serves various purposes:

- 48-bit timer for long term measurements
- System clock to determine the current time and date  
(the RTC's structure supports the direct representation of time and date)
- Cyclic time based interrupt (can be generated by any timer of the chain)

A number of programming options as well as interrupt request signals adjust the operation of the RTC to the application's requirements. The RTC can continue its operation while the XE16xyM is in certain power-saving modes, such that real time date and time information is provided.



**Figure 17-1 SFRs Associated with the RTC Module**

The RTC module consists of a chain of 3 divider blocks:

- a selectable 8:1 divider (on - off)
- the reloadable 16-bit timer T14
- the 32-bit RTC timer block (accessible via RTC\_RTCH and RTC\_RTCL), made of:
  - the reloadable 10-bit timer CNT0
  - the reloadable 6-bit timer CNT1
  - the reloadable 6-bit timer CNT2
  - the reloadable 10-bit timer CNT3

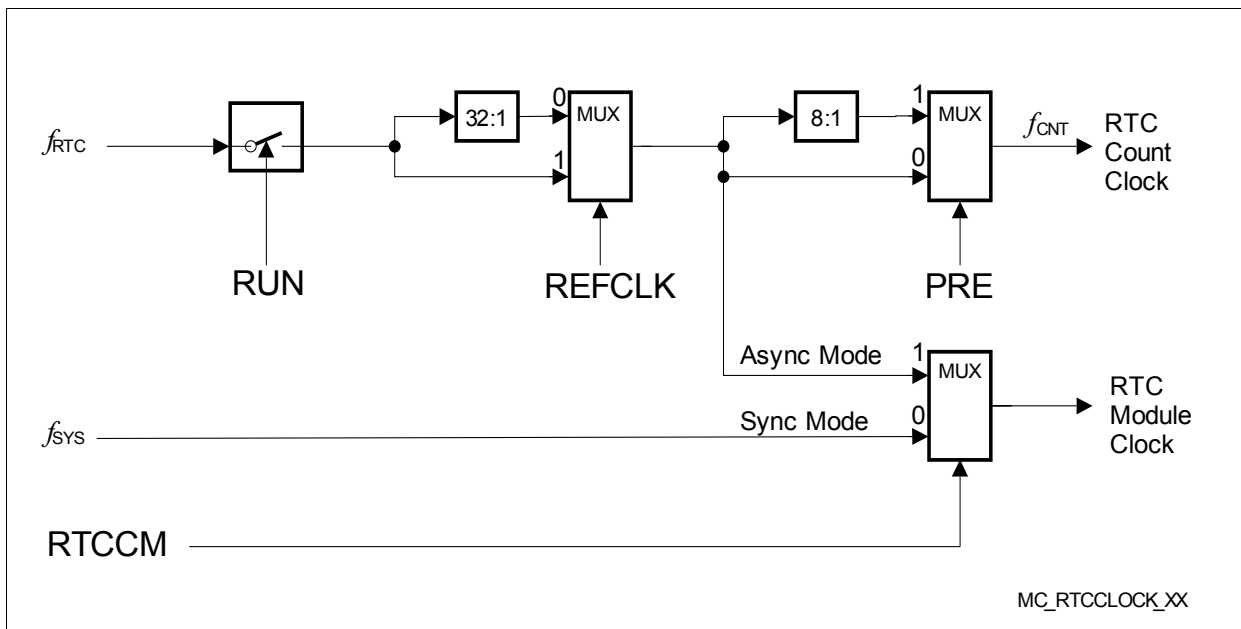
All timers count upwards. Each of the five timers can generate an interrupt request. All requests are combined to a common node request.

*Note: The RTC registers are only affected by a power reset in order to maintain the correct system time even when system or application resets are executed.*

## 17.1 Defining the RTC Time Base

The timer chain of the RTC is clocked with the count clock signal  $f_{\text{RTC}}$  which is derived from internal sources (oscillators or PLL) or external sources (pins). The currently active clock source is selected by bitfield RTCCLKSEL in register RTCCLKCON. Optionally prescaled by a factor of 32 and/or 8, this is the basic RTC clock. Depending on the operating mode, timer T14 may provide the count increments used by the application and thus determine the input frequency of the RTC timer, that is, the RTC time base (see also [Table 17-2](#)).

The RTC is also supplied with the system clock  $f_{\text{SYS}}$  of the XE16xyM. This clock signal is used to control the RTC's logic blocks and its bus interface. To synchronize properly to the count clock, the system clock must run at least four times faster than the count clock, this means  $f_{\text{SYS}} \geq 4 \times f_{\text{CNT}}$ .



**Figure 17-2 RTC Clock Supply Block Diagram**

For an example, [Table 17-1](#) lists the interrupt period range and the T14 reload values (for a time base of 1 s and 1 ms):

**Table 17-1 RTC Time Base Examples**

Oscillator Frequency	T14 Intr. Period		Reload Value A		Reload Value B	
	Min.	Max.	T14REL	Base	T14REL	Base
32.768 kHz	30.52 $\mu$ s	16.0 s	8000 <sub>H</sub> /F000 <sub>H</sub>	1.000 s	FFDF <sub>H</sub> /FFFC <sub>H</sub>	1.007 ms/ 0.977 ms

*Note: Select one value from the reload value pairs, depending if the 8:1 prescaler is disabled/enabled.*

### **Asynchronous Operation**

When the system clock frequency becomes lower than  $4 \times f_{CNT}$  proper synchronization is not possible and count events may be missed. This can be the case, when the XE16xyM reduces the system frequency to save power consumption.

In these cases the RTC can be switched to Asynchronous Mode (by clearing bit RTCCM in register RTCCLKCON). In this mode the count registers are directly controlled by the count clock independent of the system clock (hence the name). Asynchronous operation ensures correct time-keeping even during power-save modes.

However, as no synchronization between the count registers and the bus interface can be maintained in asynchronous mode, the RTC registers cannot be written. Read accesses may interfere with count events and, therefore, must be verified (e.g. by reading the same value with three consecutive read accesses).

*Note: The access restrictions in asynchronous mode are only meaningful if the system clock is not switched off, of course.*

### **Switching Clocking Modes**

The clocking mode of the RTC (synchronous or asynchronous) is selected via bit RTCCM in register RTCCLKCON. After reset, the RTC operates in Synchronous Mode (RTCCM = 1).

The selected clocking mode also affects the access to RTC registers. Bit ACCPOS in register RTC\_CON indicates if full register access is possible (ACCPOS = 1, default after reset) or not (ACCPOS = 0). This also indicates the current clocking mode.

**Attention: Software should poll bit ACCPOS to determine the proper transition to the intended clocking mode.**

After switching to Asynchronous Mode (RTCCM = 0), bit ACCPOS = 0 indicates proper operation in Asynchronous Mode. In this case the system clock can be stopped or reduced.

*Note: The clock source for asynchr. mode operation must be selected before switching to asynchr. mode and must only be changed in synchronous mode.*

After switching to Synchronous Mode, (RTCCM = 1), bit ACCPOS = 1 indicates proper operation in Synchronous Mode. In this case the RTC registers can again be accessed properly (read and write).

*Note: The RTC might lose a counting event (edge of  $f_{CNT}$ ) when switching from synchronous mode to asynchronous mode while the 8:1 prescaler is disabled. For these applications it is, therefore, recommended to set up the RTC with the 8:1 prescaler enabled.*

### **Increased RTC Accuracy through Software Correction**

The accuracy of the XE16xyM's RTC is determined by the oscillator frequency and by the respective prescaling factor (excluding or including T14 and the 8:1 prescaler). The accuracy limit generated by the prescaler is due to the quantization of a binary counter (where the average is zero), while the accuracy limit generated by the oscillator frequency is due to the difference between the ideal and real frequencies (and therefore accumulates over time). This effect is predictable and can be compensated. The total accuracy of the RTC can be further increased via software for specific applications that demand a high time accuracy.

The key to the improved accuracy is knowledge of the exact oscillator frequency. The relation of this frequency to the expected ideal frequency is a measure of the RTC's deviation. The number of cycles, N, after which this deviation causes an error of  $\pm 1$  cycle can be easily computed. So, the only action is to correct the count by  $\pm 1$  after each series of N cycles. The correction may be made cyclically, for instance, within an interrupt service routine, or by evaluating a formula when the RTC registers are read (for this the respective "last" RTC value must be available somewhere).

*Note: For the majority of applications, however, the standard accuracy provided by the RTC's structure will be more than sufficient.*

Adjusting the current RTC value would require reading and then writing the complete 48-bit value. This can only be accomplished by three successive accesses each. To avoid the hassle of reading/writing multi-word values, the RTC incorporates a correction option to simply add or suppress one count pulse.

This is done by setting bit T14INC or T14DEC, respectively, in register RTC\_CON. This will add an extra count pulse (T14INC) upon the next count event, or suppress the next count event (T14DEC). The respective bit remains set until its associated action has been performed and is automatically cleared by hardware after this event.

*Note: Setting both bits, T14INC and T14DEC, at the same time will have no effect on the count values.*

## **17.2 RTC Run Control**

If the RTC shall operate bit RUN in register RTC\_CON must be set (default after reset). Bit RUN can be cleared, for example, to exclude certain operation phases from time keeping.

*Note: A valid count clock  $f_{RTC}$  is required for proper RTC operation, of course.*

The RTC is reset by a power reset, a system/application reset does not affect the RTC registers and its operation (RTC\_IC will be reset, however). The initialization software must ensure the proper RTC operating mode.

### **Initialization and Disabling of the RTC**

Upon a Power-on Reset, register RTC\_CON adopts its reset value of 8003<sub>H</sub>, which enables the RTC and both prescalers (factor =  $8 \times 32 = 256$ ).

The RTC's clocking mode (synchronous/asynchronous) is selected bit RTCCM in register RTCCLKCON. Upon a Power-on Reset, register RTCCLKCON adopts its reset value of 0006<sub>H</sub>, which selects synchronous operation mode and the WUT as the clock source.

For an application reset that is followed by an initialization of the RTC module, the following steps are recommended:

- select synchronous RTC clocking mode, i.e. set bit RTCCLKCON.RTCCM
- select the intended (running) clock source
- make sure that bit ACCPOS is set, before writing to RTC registers
- initialize the RTC

When the RTC module is not used and shall be disabled after a Power-on Reset, the following steps are recommended:

- stop the RTC by clearing its run bit RTC\_CON.RUN
- disable the RTC module by clearing its enable bit RTC\_KSCCFG.MODEN.

When the RTC module operates in asynchronous mode and shall stop in a power saving mode, software must make sure that no active clock signal is selected by the RTC clock multiplexer.

## Real Time Clock

The RTC control register RTC\_CON selects the basic operation of the RTC module.

### RTC\_CON

Control Register

ESFR (F110<sub>H</sub>/88<sub>H</sub>)

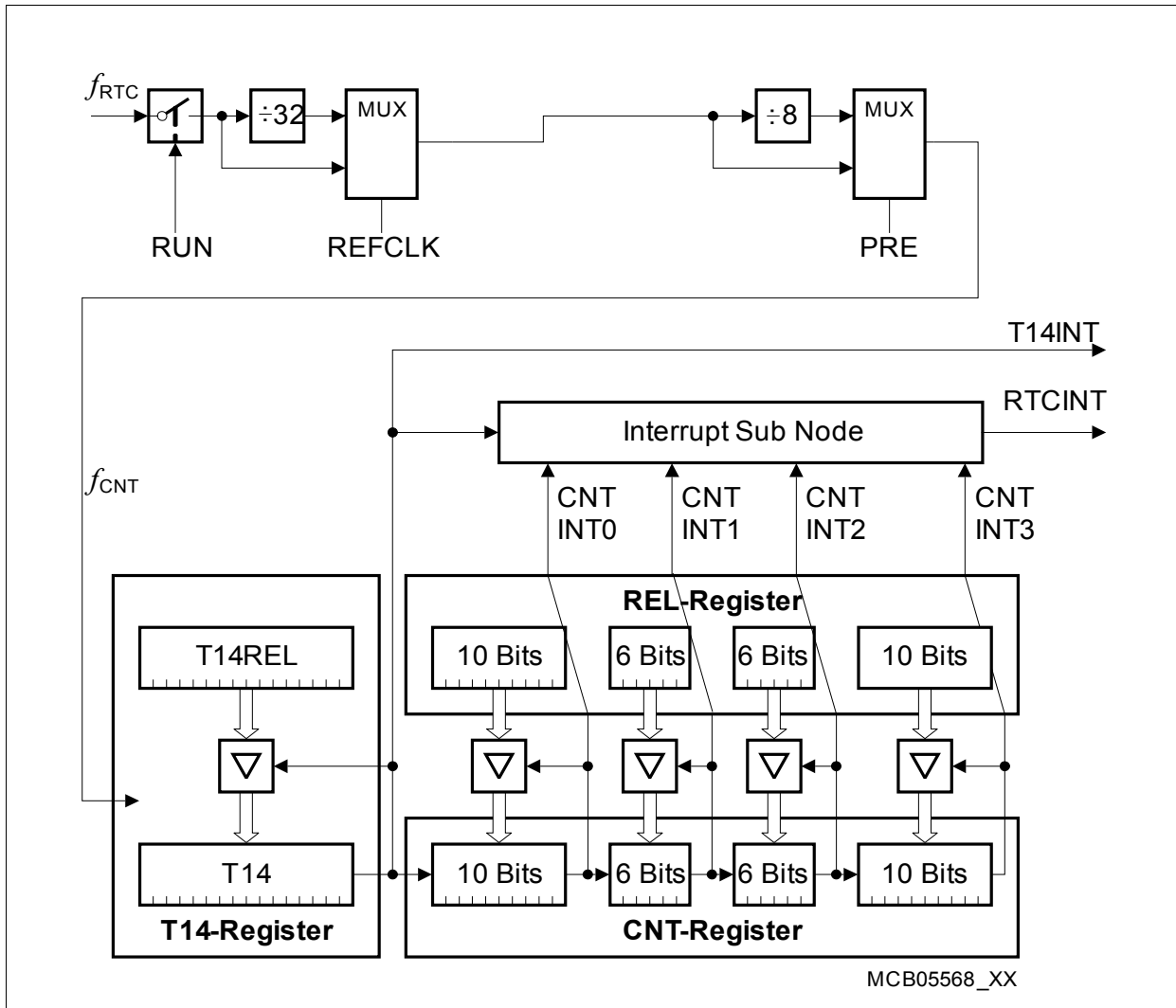
Reset Value: 8003<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>ACC POS</b>	-	-	-	-	-	-	-	-	-	-	<b>REF CLK</b>	<b>T14 INC</b>	<b>T14 DEC</b>	<b>PRE</b>	<b>RUN</b>
rh	-	-	-	-	-	-	-	-	-	-	rw	rwh	rwh	rw	rw

Field	Bits	Type	Description
<b>ACCPOS</b>	15	rh	<b>RTC Register Access Possible</b> 0 No write access is possible, only asynchronous reads 1 Registers can be read and written
<b>REFCLK</b>	4	rw	<b>RTC Input Source Prescaler (32:1) Disable</b> 0 Input Prescaler enabled 1 Input Prescaler disabled
<b>T14INC</b>	3	rwh	<b>Increment Timer T14 Value</b> Setting this bit to 1 adds one count pulse upon the next count event, thus incrementing T14. This bit is cleared by hardware after incrementation.
<b>T14DEC</b>	2	rwh	<b>Decrement Timer T14 Value</b> Setting this bit to 1 suppresses the next count event, thus decrementing T14. This bit is cleared by hardware after decrementation.
<b>PRE</b>	1	rw	<b>RTC Input Source Prescaler (8:1) Enable</b> 0 Prescaler disabled 1 Prescaler enabled
<b>RUN</b>	0	rw	<b>RTC Run Bit</b> 0 RTC stopped 1 RTC runs

### 17.3 RTC Operating Modes

The RTC can be configured for several operating modes according to the purpose it is meant to serve. These operating modes are configured by selecting appropriate reload values and interrupt signals.



**Figure 17-3 RTC Block Diagram**

*Note: Signal T14INT can trigger transfers in serial communication channels.*

#### RTC Register Access

The actual value of the RTC is indicated by the three registers T14, RTCL, and RTCH. As these registers are concatenated to build the RTC counter chain, internal overflows occur while the RTC is running. When reading or writing the RTC value, such internal overflows must be taken into account to avoid reading/writing corrupted values.

## Real Time Clock

Care must be taken, when reading the timer(s), as this requires up to three read accesses to the different registers with an inherent time delay between the accesses. An overflow from T14 to RTCL and/or from RTCL to RTCH might occur between the accesses, which needs to be taken into account appropriately.

For example, reading/writing 0000<sub>H</sub> from/to RTCH and then accessing RTCL could produce a corrupted value as RTCL may overflow before it can be accessed. In this case, RTCH would be 0001<sub>H</sub>. The same precautions must be taken for T14 and T14REL.

Timer T14 and its reload register are accessed via dedicated locations. The four RTC counters CNT3 ... CNT0 are accessed via the two 16-bit RTC timer registers, RTCH and RTCL. The associated four reload values REL3 ... REL0 are accessed via the two 16-bit RTC reload registers, RELH and RELL.

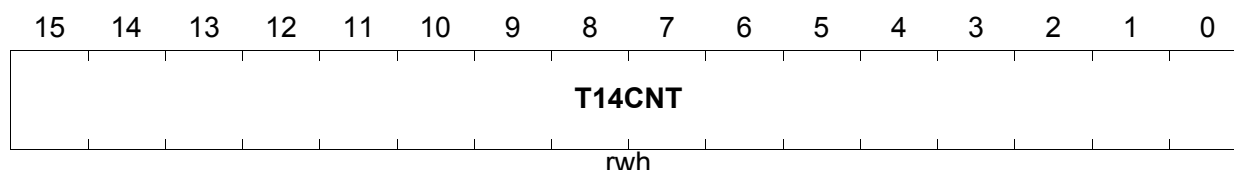
### Prescaler T14 and T14 Reload Registers

#### RTC\_T14

##### T14 Count Register

ESFR(F0D2<sub>H</sub>/69<sub>H</sub>)

Reset Value: 0000<sub>H</sub>



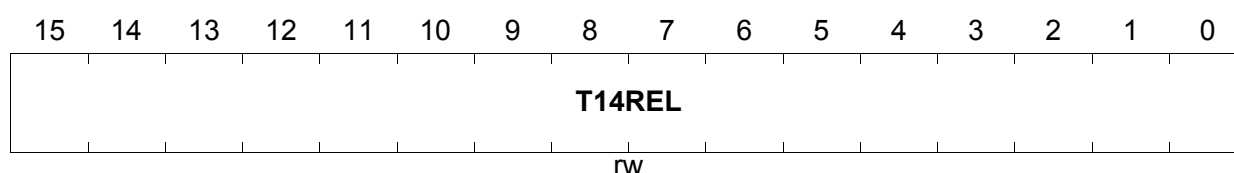
Field	Bits	Typ	Description
T14CNT	[15:0]	rwh	T14 counter

#### RTC\_T14REL

##### T14 Reload Register

ESFR(F0D0<sub>H</sub>/68<sub>H</sub>)

Reset Value: 0000<sub>H</sub>



Field	Bits	Typ	Description
T14REL	[15:0]	rw	T14 reload value

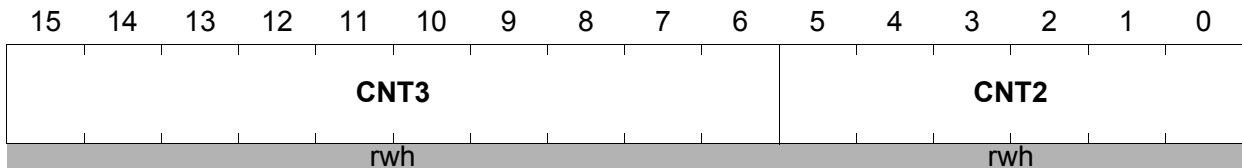


### RTC\_RTCH

**RTC Timer High Register**

**ESFR (F0D6<sub>H</sub>/6B<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



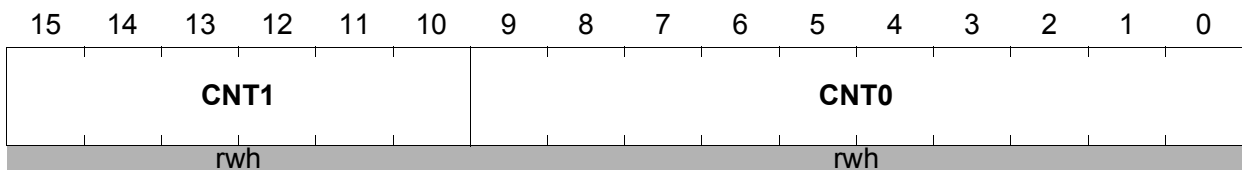
Field	Bits	Type	Description
<b>CNTx</b> (x = 3 ... 2)	[15:6], [5:0]	rwh	<b>RTC Timer Count Section CNTx</b> An overflow of bitfield CNT2 triggers a count pulse to count section CNT3 followed by a reload of CNT2 from bitfield REL2. In addition, an interrupt request is triggered.

### RTC\_RTCL

**RTC Timer Low Register**

**ESFR (F0D4<sub>H</sub>/6A<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



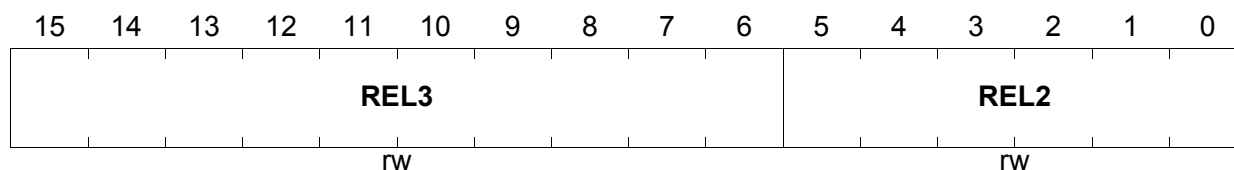
Field	Bits	Type	Description
<b>CNTx</b> (x = 1 ... 0)	[15:10], [9:0]	rwh	<b>RTC Timer Count Section CNTx</b> An overflow of bitfield CNTx triggers a count pulse to the next count section CNTx+1 followed by a reload of CNTx from bitfield RELx. In addition, an interrupt request is triggered.

### RTC\_RELH

**RTC Reload High Register**

**ESFR (F0CE<sub>H</sub>/67<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



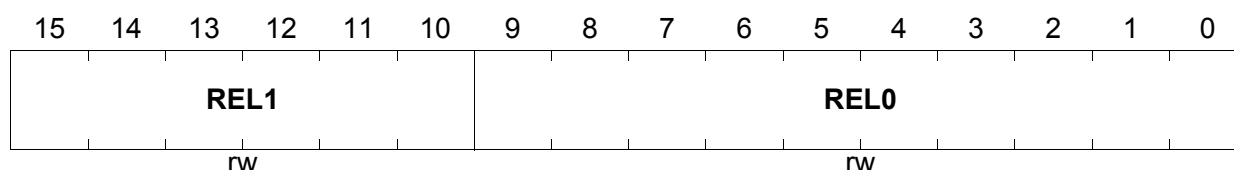
Field	Bits	Type	Description
<b>RELx</b> (x = 3 ... 2)	[15:6], [5:0]	rw	<b>RTC Reload Value RELx</b> This bitfield is copied to bitfield CNTx upon an overflow of count section CNTx.

### RTC\_RELL

**RTC Reload Low Register**

**ESFR (F0CC<sub>H</sub>/66<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
<b>RELx</b> (x = 1 ... 0)	[15:10], [9:0]	rw	<b>RTC Reload Value RELx</b> This bitfield is copied to bitfield CNTx upon an overflow of count section CNTx.

*Note: The registers of the RTC receive their reset values only upon a power reset.*

## 17.4 48-bit Timer Operation

The concatenation of timers T14 and COUNT0 ... COUNT3 can be regarded as a 48-bit timer which is clocked with the RTC input frequency, optionally divided by the prescaler. The reload registers T14REL, REL1, and RELH must be cleared to produce a true binary 48-bit timer. However, any other reload value may be used. Reload values other than zero must be used carefully, due to the individual sections of the RTC timer with their own individual overflows and reload values.

The maximum usable timespan is  $2^{48}$  ( $\approx 10^{14}$ ) T14 input clocks. Assuming no prescaler, this would equal more than 200 years at a count frequency of 32 kHz.

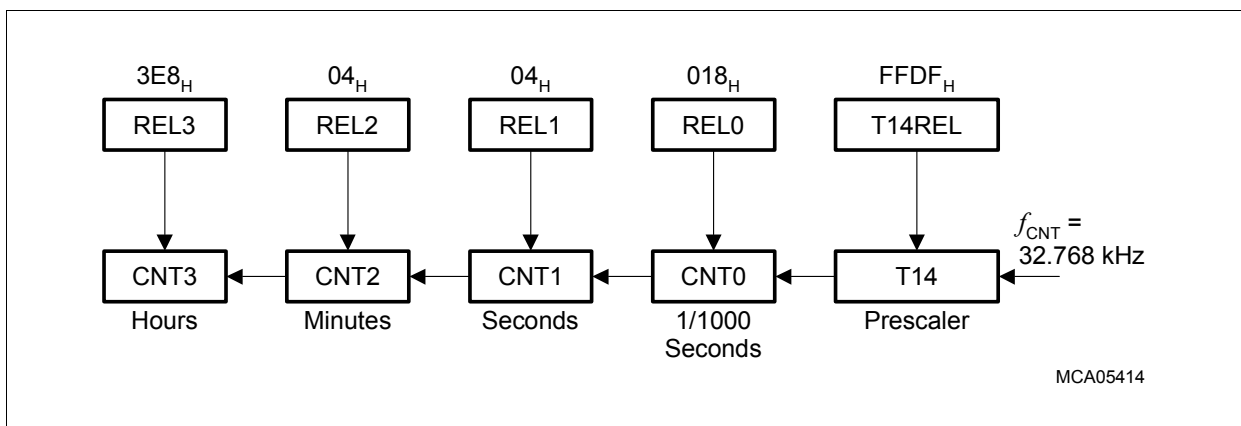
## 17.5 System Clock Operation

A real time system clock can be maintained that keeps on running also during power saving modes (optionally) that maintain a suitable clock signal and supply voltage and indicates the current time and date. This is possible because the RTC module is only reset and cleared by a power reset.

The resolution for this clock information is determined by the input clock of timer T14. By selecting appropriate reload values each cascaded timer can represent directly a part of the current time and/or date. Due to its width, T14 can adjust the RTC to the intended range of operation (time or date). The maximum usable timespan is achieved when T14REL is loaded with 0000<sub>H</sub> and so T14 divides by  $2^{16}$ .

### System Clock Example

The RTC count clock is 32.768 kHz. By selecting appropriate reload values the RTC timers directly indicate the current time (see [Figure 17-4](#) and [Table 17-2](#)).



**Figure 17-4 RTC Configuration Example**

*Note: This setup can generate an interrupt request every millisecond, every second, every minute, every hour, or every day.*

## Real Time Clock

Each timer in the chain divides the clock by  $(2^{\text{timer\_width}} - \text{reload\_value}) : 1$ , as the timers count up. **Table 17-2** shows the reload values which must be chosen for a specific scenario (i.e. operating mode of the RTC).

**Table 17-2 Reload Value Scenarios**

		REL3	REL2	REL1	REL0	T14REL
Time of Day (Figure 17-4)	Formula	$2^{10} - 24$	$2^6 - 60$	$2^6 - 60$	$2^{10} - 1000$	$2^{16} - 33$
	Rel. Value	3E8 <sub>H</sub>	04 <sub>H</sub>	04 <sub>H</sub>	018 <sub>H</sub>	FFDF <sub>H</sub>
	Function	h	m	s	1/1000 s	Prescaler
	Intr. Period	day	hour	minute	second	millisec. <sup>1)</sup>
Day of the Week	Formula	$2^{10} - 7$	$2^6 - 24$	$2^6 - 60$	$2^{10} - 60$	$2^{16} - 32768$
	Rel. Value	3F9 <sub>H</sub>	28 <sub>H</sub>	04 <sub>H</sub>	3C4 <sub>H</sub>	8000 <sub>H</sub>
	Function	day	h	m	s	Prescaler
	Intr. Period	week	day	hour	minute	second

1) T14 errors in the first example (ms) can be compensated either by choosing an adapted value for REL0, or by using software correction.

## 17.6 Cyclic Interrupt Generation

The RTC module can generate an interrupt request whenever one of the timers overflows and is reloaded. This interrupt request may be used, for example, to provide a system time tick independent of the CPU frequency without loading the general purpose timers. The interrupt cycle time can be adjusted by choosing appropriate reload values and by enabling the appropriate interrupt request.

In this mode, the other operating modes can be combined. For example, a reload value of T14REL = F9C0<sub>H</sub> ( $2^{16} - 1600$ ) generates a T14 interrupt request every 50 ms. Still the subsequent timers can be configured to represent the time or build a binary counter, however with a different time base.

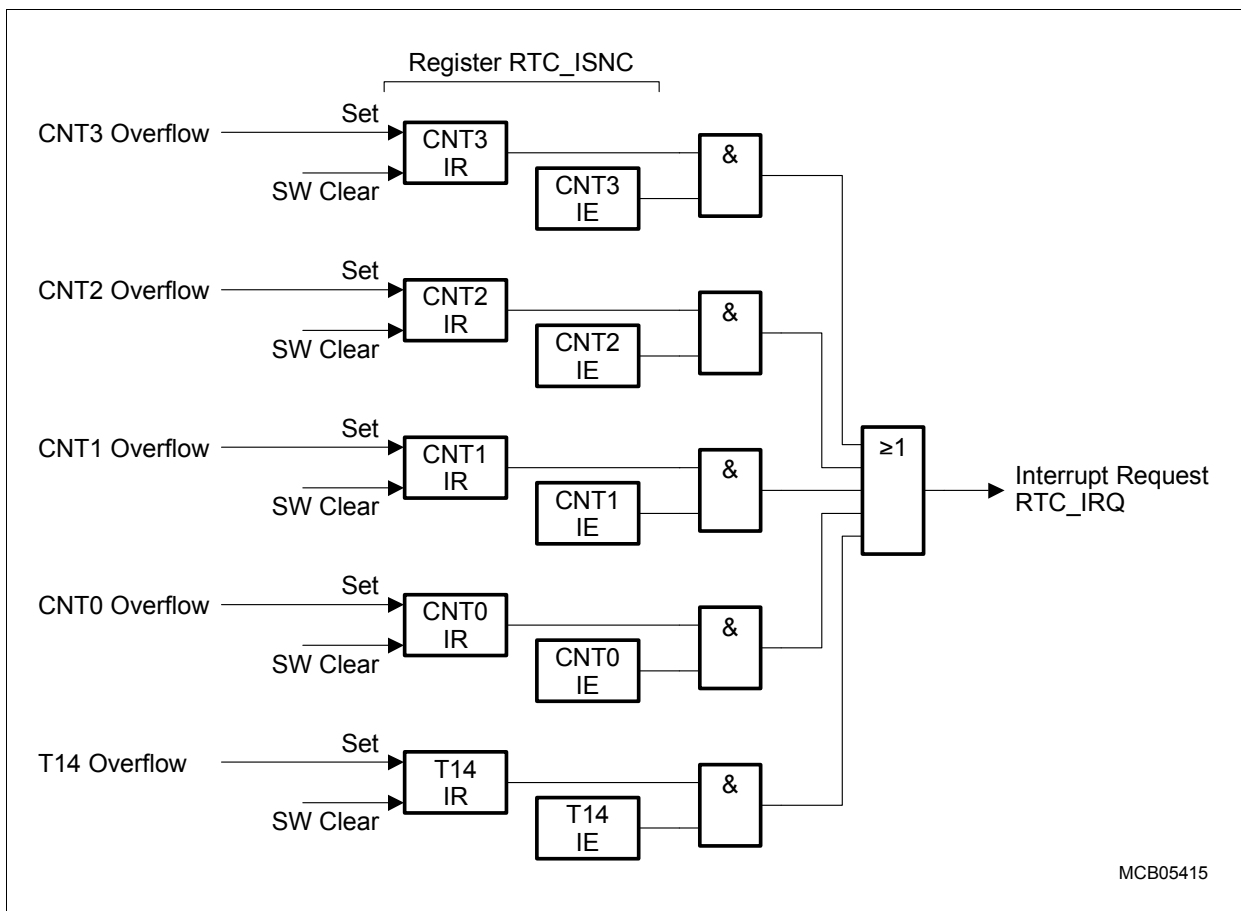
## 17.7 RTC Interrupt Generation

The overflow signals of each timer of the RTC timer chain can generate an interrupt request. The RTC's interrupt subnode control register ISNC combines these requests to activate the common RTC interrupt request line RTC\_IRQ.

Each timer overflow sets its associated request flag in register ISNC. Individual enable bits for each request flag determine whether this request also activates the common interrupt line. The enabled requests are ORed together on this line (see [Figure 17-5](#)). The common interrupt request signal is delayed by 2 system clock cycles due to synchronization.

The interrupt handler can determine the source of an interrupt request via the specific request flags and must clear them after appropriate processing (not cleared by hardware). The common node request bit is automatically cleared when the interrupt handler is vectored to.

*Note: If only one source is enabled, no additional software check is required, of course. Both the individual request and the common interrupt node must be enabled.*



**Figure 17-5 Interrupt Block Diagram**

**Real Time Clock**

**RTC\_ISNC**

**Interrupt Subnode Ctrl. Reg.**      **ESFR (F10C<sub>H</sub>/86<sub>H</sub>)**      **Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	CNT 3IR	CNT 3IE	CNT 2IR	CNT 2IE	CNT 1IR	CNT 1IE	CNT 0IR	CNT 0IE	T14 IR	T14 IE
-	-	-	-	-	-	rwh	rw	rwh	rw	rwh	rw	rwh	rw	rwh	rw

Field	Bits	Type	Description
<b>CNTxIR</b> (x = 3 ... 0)	9, 7, 5, 3	rwh	<b>Section CNTx Interrupt Request Flag</b> 0 No request pending 1 This source has raised an interrupt request
<b>CNTxIE</b> (x = 3 ... 0)	8, 6, 4, 2	rw	<b>Section CNTx Interrupt Enable Control Bit</b> 0 Interrupt request is disabled 1 Interrupt request is enabled
<b>T14IR</b>	1	rwh	<b>T14 Overflow Interrupt Request Flag</b> 0 No request pending 1 This source has raised an interrupt request
<b>T14IE</b>	0	rw	<b>T14 Overflow Interrupt Enable Control Bit</b> 0 Interrupt request is disabled 1 Interrupt request is enabled

*Note: The interrupt request flags in register ISNC must be cleared by software. They are not cleared automatically when the service routine is entered.*

**RTC\_IC**

**RTC Interrupt Ctrl. Reg.**      **ESFR (F19C<sub>H</sub>/CE<sub>H</sub>)**      **Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	GPX	RTC IR	RTC IE	ILVL				GLVL	
-	-	-	-	-	-	-	rw	rwh	rw	rw				rw	

*Note: Please refer to the general Interrupt Control Register description for an explanation of the control fields.*  
*Register RTC\_IC is not part of the RTC module and is reset with any application reset.*

## 17.8 Miscellaneous Registers

### RTC\_KSCCFG

#### Kernel State Configuration Register

**ESFR(F010<sub>H</sub>)**

**Reset Value: 0001<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>BP COM</b>	<b>0</b>	<b>COMCFG</b>	<b>BP SUM</b>	<b>0</b>	<b>SUMCFG</b>	<b>BP NOM</b>	<b>0</b>	<b>NOMCFG</b>	<b>0</b>	<b>BP MOD EN</b>	<b>MOD EN</b>				
w	r	rw	w	r	rw	w	r	rw	r	w	rw				

Field	Bits	Type	Description
<b>MODEN</b>	0	rw	<b>Module Enable</b> This bit enables the module kernel clock and the module functionality. 0 <sub>B</sub> The module clock is switched off immediately (without stop condition). The module does not react to mode control actions or read access and ignores write access (except KSCCFG). 1 <sub>B</sub> The module is switched on and can operate. To avoid pipeline effects, it is recommended to read register KSCCFG after setting MODEN before accessing other RTC registers.
<b>BPMODEN</b>	1	w	<b>Bit Protection for MODEN</b> 0 <sub>B</sub> Bit MODEN is not changed. 1 <sub>B</sub> MODEN is updated with the written value.
<b>NOMCFG</b>	[5:4]	rw	<b>Kernel Configuration in Normal Operation Mode</b> 0X <sub>B</sub> The module is switched on. 1X <sub>B</sub> The module is switched off.
<b>BPNO</b>	7	w	<b>Bit Protection for NOMCFG</b> 0 <sub>B</sub> Bitfield NOMCFG is not changed. 1 <sub>B</sub> NOMCFG is updated with the written value.
<b>SUMCFG</b>	[9:8]	rw	<b>Kernel Configuration in Suspend Mode</b> Same coding as <b>NOMCFG</b>
<b>BPSUM</b>	11	w	<b>Bit Protection for SUMCFG</b> 0 <sub>B</sub> Bitfield SUMCFG is not changed. 1 <sub>B</sub> SUMCFG is updated with the written value.
<b>COMCFG</b>	[13:12]	rw	<b>Kernel Configuration in Clock Off Mode</b> Same coding as <b>NOMCFG</b>

Field	Bits	Type	Description
<b>BPCOM</b>	15	w	<b>Bit Protection for COMCFG</b> 0 <sub>B</sub> Bitfield COMCFG is not changed. 1 <sub>B</sub> COMCFG is updated with the written value.
<b>0</b>	[3:2], 6, 10, 14	r	<b>Reserved;</b> returns 0 if read; should be written with 0;

*Note: The protection bits BPxxx enable the write access to their associated bitfields when set. Selected bitfields can be modified by a simple write access without requiring a read-modify-write sequence. They are only active during a write access and are read as 0.*

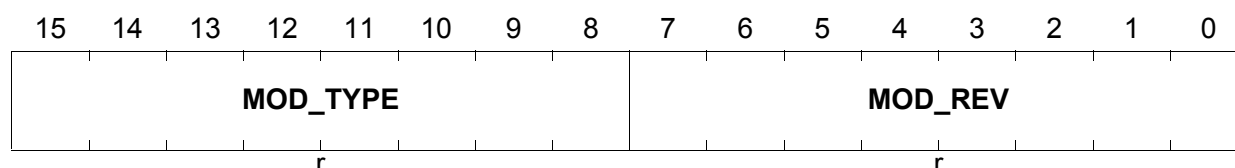
*Bitfield SUMCFG is reset by a debug reset, all other bitfields are reset by an application reset.*

## ID

### Identification Register

**MEM (FFF8<sub>H</sub>/FC<sub>H</sub>)**

**Reset Value: 5AXX<sub>H</sub>**



Field	Bits	Typ	Description
<b>MOD_REV</b>	[7:0]	r	<b>Module Revision Number</b> MOD_REV defines the revision number. The value of a module revision starts with 01 <sub>H</sub> (first revision).
<b>MOD_TYPE</b>	[15:8]	r	<b>Module Type</b> This bit field defines the module identification number (5A <sub>H</sub> = RTC).



## **18 Analog to Digital Converter**

The **Analog to Digital Converter** module (ADC) of the XE16xyM allows the conversion of analog input values into discrete digital values based on the successive approximation method. With this method, the conversion result is elaborated bit by bit, starting with the most significant bit. As a consequence, an analog to digital conversion requires a certain number of clock cycles (see [Section 18.1.4](#) and the respective Data Sheet).

This chapter is structured as follows:

- Introduction (see [Section 18.1](#))
- Operating the ADC (see [Section 18.2](#))
- Module implementation in XE16xyM (see [Section 18.3](#))

### **18.1 Introduction**

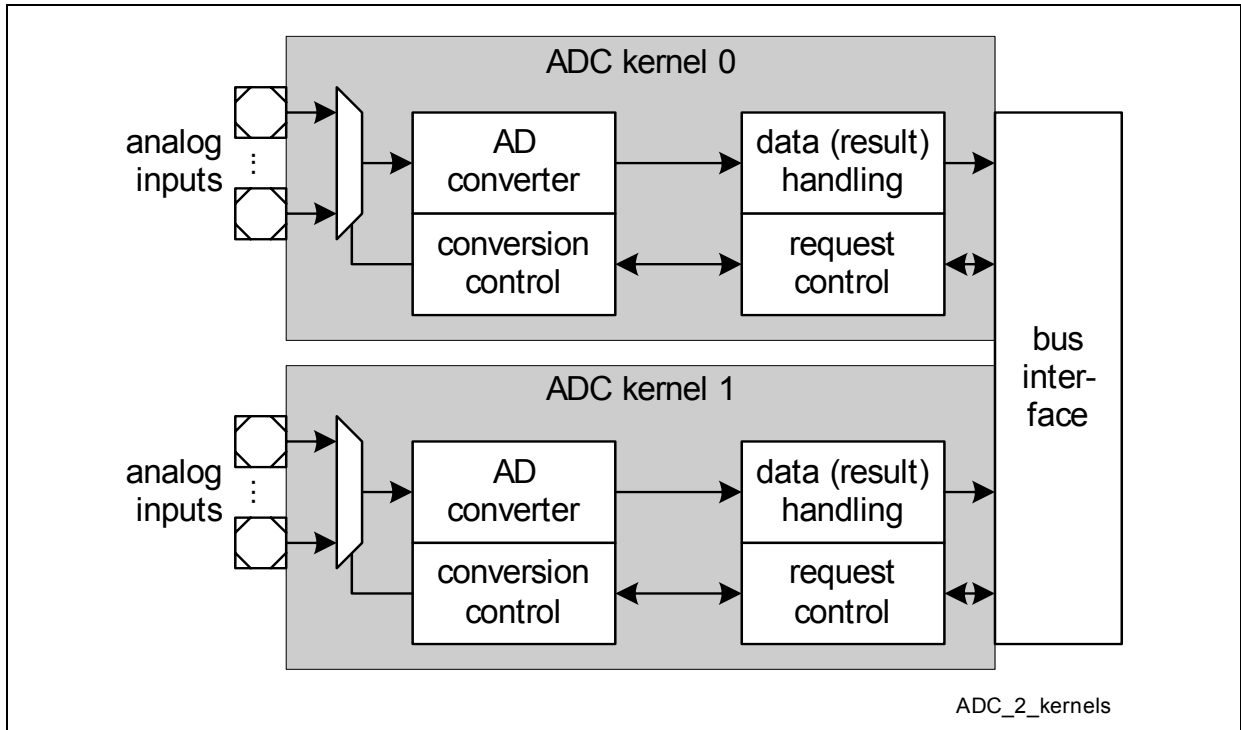
This section gives an overview about the feature set of the ADC module and introduces the general structure. It describes the:

- ADC block diagram with two kernels (see [Section 18.1.1](#))
- Feature set description (see [Section 18.1.2](#))
- Abbreviations (see [Section 18.1.3](#))
- Kernel overview (see [Section 18.1.4](#))
- Conversion request handling (see [Section 18.1.5](#))
- Conversion result handling (see [Section 18.1.6](#))
- Interrupt structure (see [Section 18.1.7](#))
- Electrical models (see [Section 18.1.8](#))
- Transfer characteristics and error definitions (see [Section 18.1.9](#))

### 18.1.1 ADC Block Diagram

The ADC module contains 2 independent kernels (ADC0, ADC1) that can operate autonomously or can be synchronized to each other. An ADC kernel is a unit used to convert an analog input signal into a digital value and provides means for triggering conversions, data handling and storage.

With this structure, parallel conversion of up to two analog input channels is supported.



**Figure 18-1 ADC Module Block Diagram**

### **18.1.2 Feature Set**

Features of each ADC kernel:

- Analog supply voltage range from 3.3 V (minimum) to 5 V (nominal) for  $V_{DDPA}$
- Input voltage range from 0 V to analog supply voltage  $V_{DDPA}$
- Input multiplexer for a maximum of 16 possible analog input channels (CH0 to CH15)
- 16 analog input channels (CH0 to CH15) with fully configurable conversion setup
- For safety purposes, the input multiplexer also provides access to  $V_{AGND}$  (connected to CH16) and  $V_{AREF}$  (connected to CH17) that can be accessed indirectly (alias feature)
- 10-bit conversion time less than 1  $\mu$ s
- One standard reference input ( $V_{AREF}$ ) and one alternative reference input (CH0) available
- Multiplexer test mode for analog input CH7
- Broken wire detection can be enabled for each input channel
- 3 conversion request sources for external or timer-driven events, auto-scan, programmable sequences, SW-driven conversions, etc.
- Synchronization of the ADC kernels for concurrent conversion starts and parallel sampling and measuring of analog input signals, e.g. for phase current measurements in AC drives
- Control capability for an external analog multiplexer, respecting the additional set up time and scan support
- Adjustable sampling times to accommodate output impedance of different analog signal sources (sensors, etc.)
- Possibility to cancel running conversions on demand with automatic restart
- Flexible interrupt generation (possibility of PEC support)
- Limit checking to reduce interrupt load (e.g. for temperature measurements or overload detection, only values outside programmable boundary values lead to an interrupt)
- Programmable data reduction filter, e.g. for digital anti-aliasing filtering, by adding a programmable number of conversion results
- Independent result registers (8 independent registers)
- Support of conversion result FIFO mechanism to allow a longer interrupt latency
- Support of suspend and power saving modes
- Individually programmable reference selection for each channel, e.g. to allow measurements of 3.3 V and 5 V signals in the full measurement range with the same ADC kernel

### 18.1.3 Abbreviations

The following acronyms and terms are used in the ADC chapter:

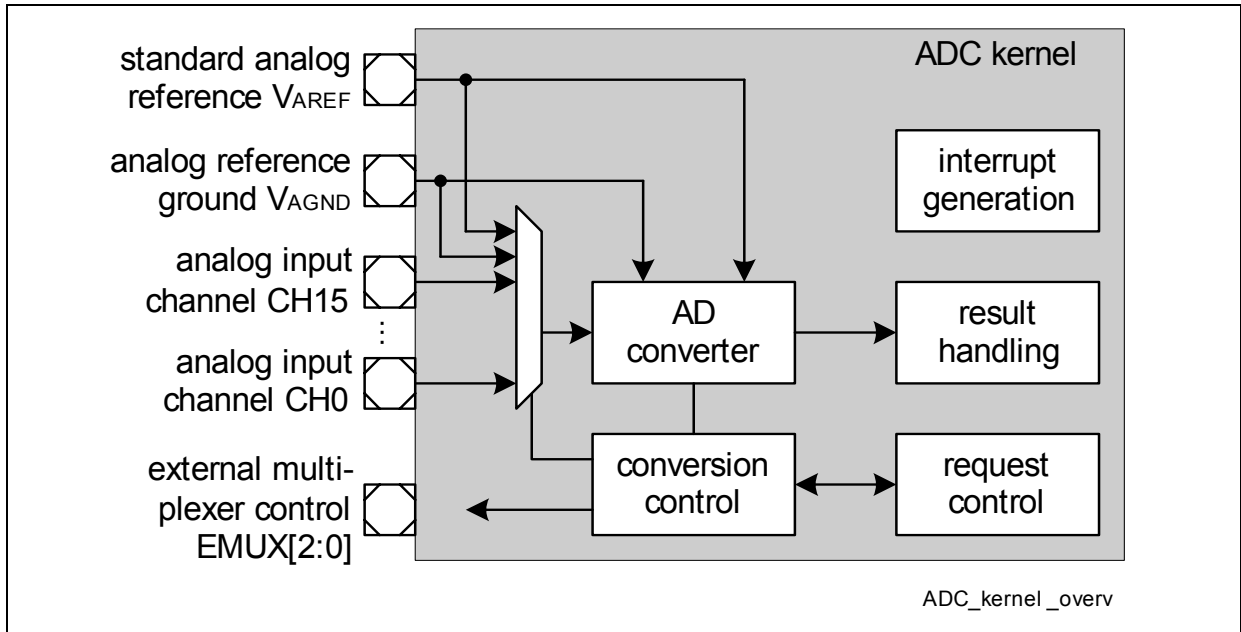
**Table 18-1 Abbreviations in ADC chapter**

<b>Abbreviation</b>	<b>Meaning</b>
ADC	Analog to digital converter
DNL	Differential non-linearity error
FIFO	First-in-first-out data buffer mechanism
INL	Integral non-linearity error
LSB <sub>n</sub>	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 <sup>n</sup> equally distributed steps)
PEC	Peripheral Event Controller
SCU	System control unit of the device
TUE	Total unadjusted error

### **18.1.4 ADC Kernel Overview**

Each ADC kernel comprises:

- An **analog to digital converter** with a maximum of 16 analog inputs (CH0 - CH15). This block selects an input signal CHx and translates the analog voltage into a digital value.  
Not all analog input channels are necessarily available in all packages, please refer to the implementation description in [Section 18.3](#).
- A **conversion control** unit defining the conversion parameters like the length of the sample phase, the resolution and the reference for each conversion. The length of the sample phase and the resolution depend on the type of sensor (or other analog sources) connected to the ADC. These values are similar for several channels and, therefore, are grouped together to form the so-called input classes. Each channel can be individually assigned to an input class to define these parameters.  
The conversion control also handles the start conditions for the conversions, such as the immediate start (cancel-inject-repeat), overwrite of former results (wait-for-read), or synchronization of the ADC kernels (parallel conversions).  
Additionally, an external analog multiplexer can be controlled by the output signals EMUX[2:0] of each ADC kernel.
- A **request control** unit defining which analog input channel has to be converted next. It contains 3 request sources that can trigger conversions depending on different events, such as edges of PWM or timer signals or events at port pins. Each request source can trigger either 1, up to 4, or up to 16 conversions in a sequence.
- A **result handling** unit providing 8 result registers for the conversion results. The conversion result of each analog input channel can be directed to one of the result registers to be stored there. The result handling block also supports data reduction (e.g. for digital anti-aliasing filtering) by automatically adding up to 4 conversion results before informing the CPU that new data is available.  
Additionally, the results registers can be concatenated to FIFO structures to provide storage capability for more than one conversion result without overwriting previous data. This feature also helps to handle CPU latency effects.
- An **interrupt generation** unit issuing interrupt requests to the CPU depending on ADC events. The interrupt generation in the ADC kernels support different mechanisms, e.g. some interrupts can be coupled to a value range of the conversion result (limit checking), some interrupts can be used to transport conversion data to locations in memory for further treatment, and other interrupts are generated after a complete sequence of conversions.



**Figure 18-2 ADC Kernel Block Diagram**

The time required for a conversion depends on the result width and on the selected sample time. Both values depend on the configured ADC clock frequency, of course.

Conversion time:

$$t_{CN} = ((N + 1) \times t_{ADCI}) + ((2 + STC) \times t_{ADCI}) + 2 \times t_{ADC} \quad (18.1)$$

where N = number of result bits (10/8), STC = additional sample time (see also [Section 18.2.4](#)).

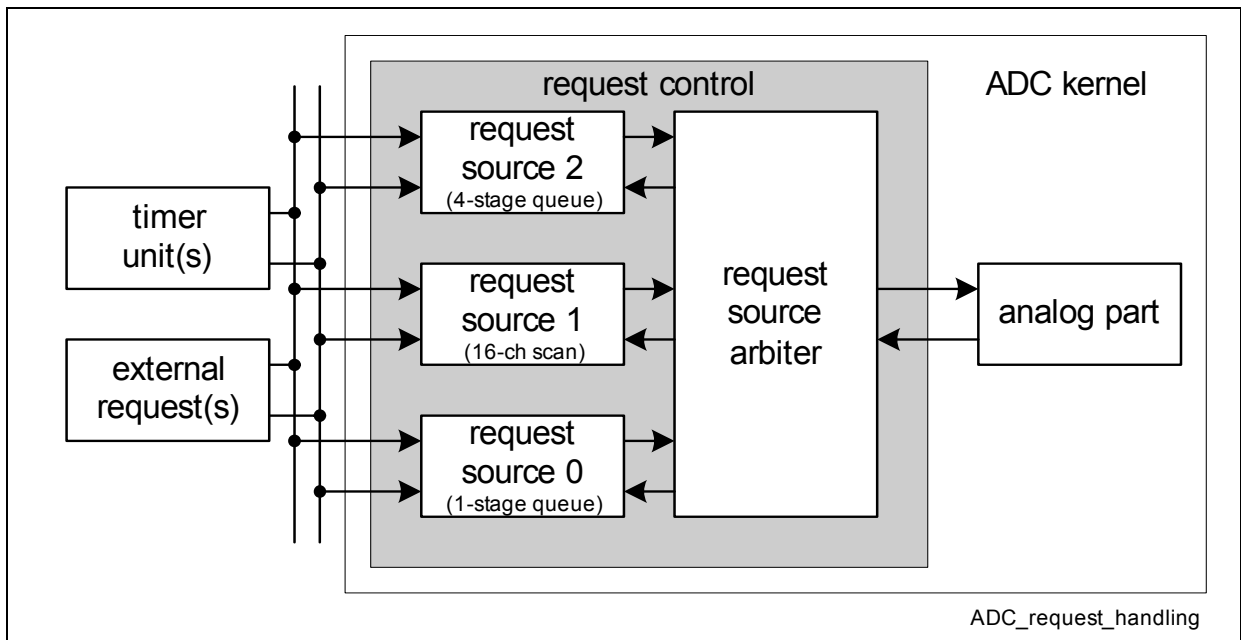
### 18.1.5 Conversion Request Unit

The conversion request unit of each ADC kernel autonomously handles the generation of conversion requests.

It contains three independent request sources that are connected to several modules to trigger the start of a conversion. A request source defines the analog input channel to be converted if a defined event occurs. For example, a trigger pulse from a timer unit generating a PWM signal can start the conversion of a single input channel or a programmed sequence of input channels.

Depending on the application, the request sources can be triggered by different events, either issued by other modules or under SW control. As a consequence, there can be two or more conversion requests pending at the same time. To allow the user to adapt the request source mechanism to the application needs, the trigger capability, the channel number(s) to be converted, and the priority can be individually programmed for each request source.

An arbiter block regularly scans the request sources for pending conversion requests and acts upon the conversion request with the highest priority. This conversion request is forwarded to the converter to start the conversion of the requested channel.



**Figure 18-3 Conversion Request Unit**

The functional characteristics of the request sources are adapted to the most common application requirements. In all request sources, a continuous operation or a single-shot operation can be selected. For continuous operation, the programmed sequence of conversions requests are continuously issued (once started), whereas in single-shot mode, each sequence of conversion requests has to be explicitly started. The trigger for a conversion request or a sequence can be handled under SW control or can be

synchronized to ADC-external events, such as timer signals or port pins. For each request source, the user can select an input signal (from 8 possible signals REQTRx[H:A]) as trigger input REQTRx and an input signal (from 8 possible signals REQGTx[H:A]) as gating input REQGTx.

- **Request source 0** (1-stage sequential source) can issue a conversion request for a single input channel. The channel number can directly be programmed.  
This mechanism could be used for SW-controlled conversion requests or HW-triggered conversions of a single input channel. If programmed with a high priority, it can interrupt the sequences of the other request sources to inject a single conversion.
- **Request source 1** (16-channel scan source) can issue conversion requests for a sequence of up to 16 input channels. It can be programmed which channel takes part in this sequence. The sequence always starts with the highest enabled channel number and continues towards lower channel numbers (order defined by the channel number, each channel can be converted only once per sequence).  
This mechanism could be used to scan input channels permanently or on a regular time base. For example, if programmed with a low priority, some input channels can be scanned in a background task to update information that is not time-critical.
- **Request source 2** (4-stage sequential source) can issue a conversion request for a sequence of up to 4 input channels. The channel numbers can be freely programmed, especially multiple conversions of the same channel within the sequence are supported.  
This mechanism could be used to support application-specific conversion sequences that can not be covered by the scanning mechanism of request source 1. Especially for timing-critical sequences containing multiple conversions of the same channel, request source 2 should be used. For example, if programmed with a medium priority, some input channels can be converted when a specified event occurs (e.g. synchronized to a PWM) while the scan of other input channels of the background task (handled by request source 1) is interrupted.



### **18.1.6 Conversion Result Unit**

The conversion result unit comprises:

- A set of **8 result registers** for storing the conversion results. A pointer mechanism for each analog input channel distributes the conversion results to the result registers. Especially for auto-scan applications, this feature simplifies PEC use (only one PEC channel needed to transfer a complete auto-scan sequence into the device memory).
- The result registers are accompanied by **valid flags** to indicate if new data has been stored since it has been read out (new data indication).
- A **result FIFO mechanism** for conversion results handling with a “relaxed” CPU timing. Result registers not directly used as target for a conversion result can be concatenated to form a result FIFO. This structure allows to store a sequence of conversion results before the CPU has to interact.
- A **digital anti-aliasing or data reduction filter**, accumulating a programmable number of conversion results before generating a result event interrupt.  
This feature can be used to avoid CPU intervention on each conversion result if a certain number of conversion results are added before further treatment, especially for fast conversions sequences and averaging of results.
- A **wait-for-read mechanism** can be enabled independently for each result register to delay conversions targeting a result register that has not yet been read out.
- A **flexible interrupt generation** based on result register events. A result register event occurs if a new valid data word becomes available in a result register and can be read out. Especially when using data reduction or digital anti-aliasing filtering, the result register event indicates that the final result is available.
- Result register read view compatible to the ADC result register on XC16x devices available at one address and a new read view for data reduction capability at another address.
- **Debugger support** for ADC result registers supporting read out of ADC conversion results without changing the result status (new data indication).

### **18.1.7 Interrupt Structure**

Each ADC kernel provides 4 independent service request output signals (ADCx\_SR[3:0]) used for interrupt handling (SRx signals connected to interrupt control registers). The interrupt generation inside the ADC kernel is based on three different types of events.

- **Channel events:**

A channel event is detected if a conversion is finished and the conversion result is within a programmable value range.

This type of event can be used to check if analog input values are inside or out of a nominal operating range, especially to reduce CPU load for background tasks. This allows the user to interrupt the CPU only if the specified conversion result range is met (or not met) instead of comparing each result by SW.

- **Result events:**

A result event is detected if a new result is available in a result register and can be read out, e.g. to store the data in memory for further treatment by SW.

This type of event can be used to trigger a read action by the CPU (or PEC). Especially when using data reduction or digital anti-aliasing filtering, not all finished conversion leads to a new result. Furthermore, when using a result FIFO, a result event decouples the CPU (PEC) read out from the channel events and tolerates a higher interrupt latency. The result register structure allows to use a single PEC channel for a complete auto-scan sequence by triggering the read out by a result event (if the conversion results of all channels taking part in the auto-scan sequence target the same result register, e.g. with FIFO mechanism or with a wait-for-read condition to avoid data loss).

- **Request source events:**

A request source event is detected if a scan source has completely finished the requested conversion sequence. For a sequential source, the user can define where inside a conversion sequence a request source event is generated.

This type of event can be used to inform the CPU that a conversion sequence has reached a defined state and SW can start the treatment of the related results in a block.

Each ADC event is indicated by a dedicated flag that can be cleared by SW. An interrupt can be generated (if enabled) for each event, independently from the status of the corresponding event indication flag. This structure ensures efficient PEC handling of ADC events (the ADC event can generate an interrupt without the need to clear the indication flag). A node pointer mechanism allows the user to group interrupt events by selecting which service request output signals SRx becomes activated by which event. Each ADC event can be individually directed to one of the service request output signals to adapt easily to application needs.

**Analog to Digital Converter**

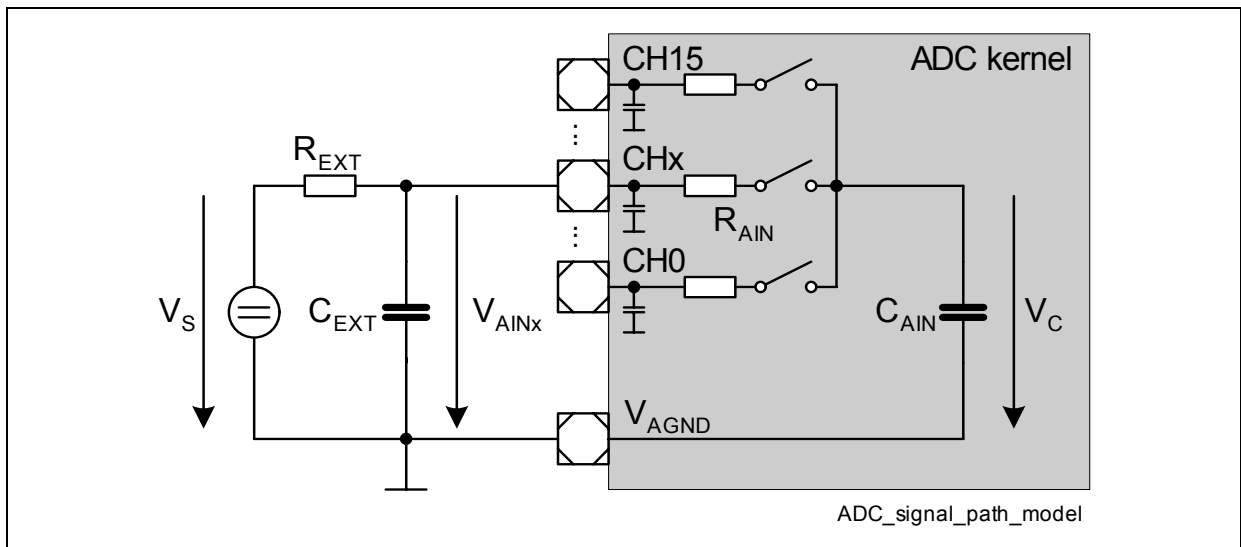
*Note: A conversion can lead to three interrupts, one of each type. In this case, the ADC module first triggers the request source event interrupt, then the channel event interrupt, followed by the result event interrupt (all within a few  $f_{ADC}$  clock cycles).*

## 18.1.8 Electrical Models

Each conversion of an analog input voltage into a digital value consists of two consecutive phases. During the sample phase, the input voltage is sampled and prepared for the following conversion phase. A simplified model for the sample phase describes the input signal path, whereas a second simplified model for the conversion phase is related to the reference voltage handling.

### 18.1.8.1 Input Signal Path

The ADC kernel in the XE16xyM is based on one switched capacitor field for measurement with a total capacity represented by  $C_{AIN}$  and a small static capacitor at each input pin. During the sample phase, the capacitor field  $C_{AIN}$  is connected to one of the analog input CHx via an input multiplexer. The multiplexer is modeled by ideal switches and series resistors  $R_{AIN}$ . Only the switch to the selected analog input is closed during the sample phase. During the conversion phase or while no conversion is running (ADC is idle), all switches are open. The voltage at the analog input channel CHx is represented by  $V_{AINx}$ .



**Figure 18-4 Signal Path Model**

A simplified model for the analog input signal path is given in [Figure 18-4](#). 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  $C_{AIN}$  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  $C_{AIN}$ .

Additionally, the capacitor  $C_{AIN}$  is automatically precharged to a voltage of approximately the half of the standard reference voltage  $V_{AREF}$  to minimize the average difference between  $V_{AINx}$  and  $V_C$  at the beginning of a sample phase. Due to varying parameters and parasitic effects, the precharge voltage of  $C_{AIN}$  is typically smaller than  $V_{AREF} / 2$ .

On the other hand, the charge redistribution between  $C_{EXT}$  and  $C_{AIN}$  leads to a voltage change of  $V_{AINx}$  during the sample phase. In order to keep this voltage change lower than  $1 \text{ LSB}_n$ , it is recommended to use an external blocking capacitor  $C_{EXT}$  in the range of at least  $2^n \times C_{AIN}$ .

The resulting low-pass filter of  $R_{EXT}$  and  $C_{EXT}$  should be dimensioned in a way to allow  $V_{AINx}$  to follow  $V_S$  between two sample phases of the same analog input channel.

Please note that, especially at high temperatures, the analog input structure of an ADC can lead to a leakage current and introduces an error due to a voltage drop over  $R_{EXT}$ . The ADC input leakage current increases if the input voltage level is close to the analog supply ground  $V_{SS}$  or to the analog power supply  $V_{DDPA}$ . It is recommended to use an operating range for the input voltage between approximately 3% and 97% of  $V_{DDPA}$  to reduce the input leakage current of the respective ADC channel.

Furthermore, the leakage is influenced by an overload condition at adjacent analog inputs. During an overload condition, an input voltage exceeding the supply range is applied at an input and the built-in protection circuit limits the resulting input voltage. This leads to an overload current through the protection circuit that is translated (by a coupling factor) into an additional leakage at adjacent inputs.

### **18.1.8.2 Reference Path**

During the conversion phase, parts of the capacitor field  $C_{AIN}$  are switched to a reference input or to  $V_{AGND}$ . The ADC kernel supports two possible reference inputs,  $V_{AREF}$  as standard reference and CH0 as alternative reference. The reference selection between both possibilities is handled individually for each analog input channel. For example, this structure allows conversions of 5 V and 3.3 V based analog input signals with the same ADC kernel.

A high accuracy of the conversion results requires a stable and noise-free reference voltage and analog supply voltages during the conversion phase. Instable voltages or noise on the supply or reference inputs lead to a reduced conversion accuracy. Please note that noise can also be introduced into the ADC module by other modules, e.g. by switching of neighboring pins. It is strongly recommended to carefully decouple analog from digital signal domains.

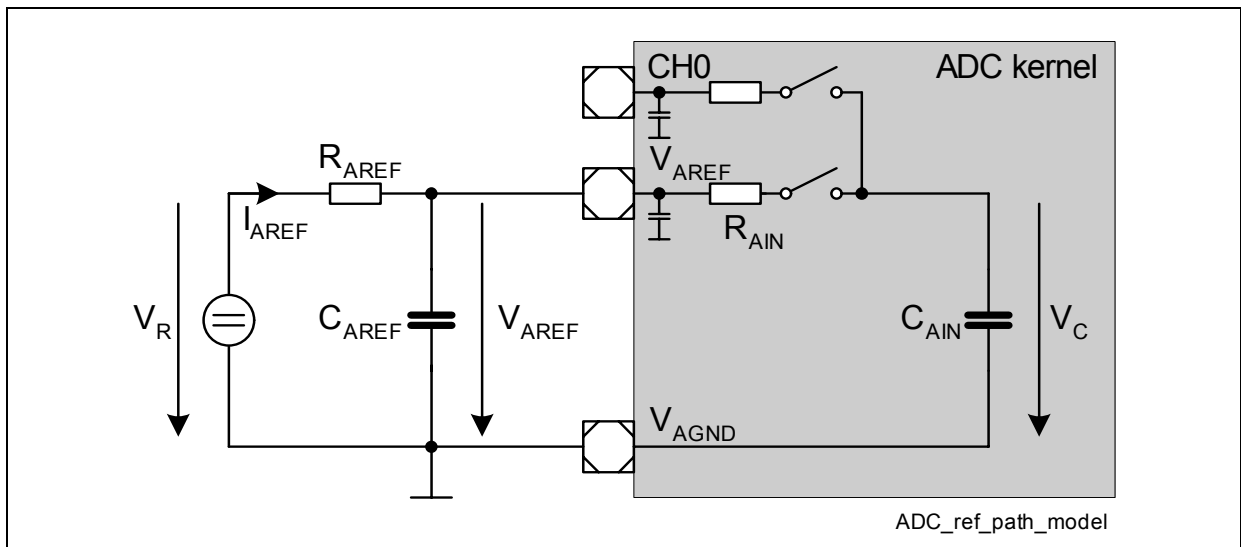
Due to the switching of parts of  $C_{AIN}$ , the ADC requires a dynamic current at the selected reference input. Thus, the impedance  $R_{AREF}$  of the reference voltage source  $V_R$  has to

be low enough to supply the reference current during the conversion phase. An external blocking capacitor  $C_{AREF}$  should be used to supply the peak currents and to minimize the current delivered by the reference source.

Due to the charge redistribution between  $C_{AREF}$  and parts of  $C_{AIN}$ , the voltage  $V_{AREF}$  decreases during the conversion phase. In order to limit the error introduced by this effect to  $1/2 \text{ LSB}_n$ , the external blocking capacitor  $C_{AREF}$  for the reference input should be at least  $2^n \times C_{AIN}$ .

The reference current  $I_{AREF}$  introduces a voltage drop at  $R_{AREF}$  that should not be neglected for the calculation of the overall accuracy. The average reference current during a conversion depends on the reference voltage level and the time  $t_{CONV}$  between two conversion starts.

$$I_{AREF} = C_{AIN} \times V_{AREF} / t_{CONV}$$



**Figure 18-5 Reference Path Model**

### 18.1.9 Transfer Characteristics and Error Definitions

The ideal transfer characteristic of the ADC translates a continuous analog input voltage into a discrete digital value out of a result range of  $2^n$  steps for  $n$  bit resolution over a measurement range between 0 and a reference voltage. Each digital value in the available result range (from 0 to  $2^n-1$ ) represents an input voltage range that is defined by the reference voltage divided by  $2^n$ . This range (called quantization step) represents the smallest granularity (called  $\text{LSB}_n$ ) that can be handled by the ADC. Due to the discrete character of the digital result, each ADC conversion result has a system-inherent quantization uncertainty of  $\pm 0.5 \text{ LSB}_n$ . According to the ideal transfer characteristics, the first digital transition (between the digital values 0 and 1) takes place when the analog input reaches  $0.5 \text{ LSB}_n$ .

An analog input voltage above the reference voltage leads to a saturation of the digital result at  $2^n-1$ .

Deviations of the conversion result from the ideal transfer characteristics can appear:

- An **offset error** is the deviation from the ideal transfer characteristics for an input voltage close to 0. It describes the difference between  $0.5 \text{ LSB}_n$  and the input voltage where the first digital transition (between the values of 0 and 1) occurs.
- A **gain error** is the deviation from the ideal transfer characteristics for an input voltage close to the reference voltage. It describes the difference between the reference voltage and the input voltage where the last digital transition (between the values of  $2^n-2$  and  $2^n-1$ ) occurs.
- A **differential non-linearity error** (DNL) describes the variations in the analog input voltage between two adjacent digital conversion results, over the full measurement range. If each step between the digital conversion results  $x$  and  $x+1$  is exactly  $1 \text{ LSB}_n$ , the DNL value is zero. If the DNL value is lower than  $1 \text{ LSB}_n$ , the possibility of missing codes is excluded. A missing code occurs if not all values of the possible conversion result range can be reached.
- An **integral non-linearity error** (INL) describes the maximum difference between the transfer characteristics between the first and the last point of the measurement range and the real transfer characteristics (without quantization uncertainty, offset and gain errors).
- The **total unadjusted error** (TUE) describes the maximum deviation between a real conversion result and the ideal transfer characteristics over a given measurement range. Since some of these errors noted above can compensate each other, the TUE value generally is much less than the sum of the individual errors.

The TUE also covers production process variations and internal noise effects (if switching noise is generated by the system, this generally leads to an increased TUE value).

## **18.2 Operating the ADC**

This section describes the kernel functions and how to operate the kernel. It provides the functional description and the associated register descriptions.

- Register overview (see [Section 18.2.1](#))

### **General module, kernel and arbiter operation:**

- Enabling the ADC module for configuration of the behavior for the different device operating modes (see mode control description in [Section 18.2.2](#)).
- Enabling the converter for operation or selecting the desired power saving mode (see [Section 18.2.3](#))
- Selecting the appropriate frequency for the converter and for the request source arbiter (see [Section 18.2.4](#)).
- General ADC registers (see [Section 18.2.5](#))
- Configuring the request source arbiter (see [Section 18.2.6](#))
- Arbiter registers (see [Section 18.2.7](#))

### **Request source operation:**

- Scan request source handling (see [Section 18.2.8](#))
- Scan request source registers (see [Section 18.2.9](#))
- Sequential request source handling (see [Section 18.2.10](#))
- Sequential request source registers (see [Section 18.2.11](#))

### **Channel and result register operation:**

- Configuring the channel-related functions (see [Section 18.2.12](#))
- Channel-related registers (see [Section 18.2.13](#))
- Conversion result handling (see [Section 18.2.14](#))
- Conversion request handling (see [Section 18.2.15](#))

### **Additional features:**

- Multiplexer test mode for CH7 (see [Section 18.2.16](#))
- External multiplexer control (see [Section 18.2.17](#))
- Synchronization for parallel conversions (see [Section 18.2.18](#))
- Equidistant sampling (see [Section 18.2.19](#))
- Broken wire detection (see [Section 18.2.20](#))
- Additional feature registers (see [Section 18.2.21](#))



## 18.2.1 Register Overview

**Table 18-2** shows all registers required for programming the ADC module. It summarizes the ADC kernel registers and defines their offsets and the reset values. The offset has to be added to the base address of the respective ADC kernels (see **Section 18.3.1**) to obtain the absolute address for each register.

The prefix “**ADCx\_**” has to be added to the register names in this table for each ADC kernel to distinguish registers of the different kernels. In this naming convention, x indicates the kernel number.

All ADC registers (including KSCFG.NOMCFG and KSCFG.COMCFG) are reset by an application reset, whereas bit field KSCFG.SUMCFG is reset by a debug reset.

*Note: Register bits marked “w” always deliver 0 when read.*

**Table 18-2 ADC Module Register Summary**

Short Name	Description	Offset <sup>1)</sup>	See Page
------------	-------------	----------------------	----------

### General Registers

<b>ID</b>	Module Identification Register	08 <sub>H</sub>	<a href="#">Page 18-26</a>
<b>KSCFG<sup>2)</sup></b>	Kernel State Configuration Register	0C <sub>H</sub>	<a href="#">Page 18-24</a>
<b>GLOBCTR</b>	Global Control Register	10 <sub>H</sub>	<a href="#">Page 18-27</a>
<b>GLOBSTR</b>	Global Status Register	12 <sub>H</sub>	<a href="#">Page 18-29</a>
<b>RSIR0</b>	Request Source 0 Input Select Register	00 <sub>H</sub>	<a href="#">Page 18-32</a>
<b>RSIR1</b>	Request Source 1 Input Select Register	02 <sub>H</sub>	<a href="#">Page 18-32</a>
<b>RSIR2</b>	Request Source 2 Input Select Register	04 <sub>H</sub>	<a href="#">Page 18-32</a>

### Arbiter Registers

<b>ASENR</b>	Arbitration Slot Enable Register	18 <sub>H</sub>	<a href="#">Page 18-39</a>
<b>RSPR0</b>	Request Source Priority Register 0	14 <sub>H</sub>	<a href="#">Page 18-40</a>

### Channel-Related Registers

<b>CHCTR0-15</b>	Channel Control Register 0-15	20 <sub>H</sub> -3E <sub>H</sub>	<a href="#">Page 18-70</a>
<b>INPCR0</b>	Input Class Register 0	C0 <sub>H</sub>	<a href="#">Page 18-72</a>
<b>INPCR1</b>	Input Class Register 1	C2 <sub>H</sub>	<a href="#">Page 18-72</a>
<b>LCBR0</b>	Limit Checking Boundary Register 0	84 <sub>H</sub>	<a href="#">Page 18-73</a>
<b>LCBR1</b>	Limit Checking Boundary Register 1	86 <sub>H</sub>	<a href="#">Page 18-73</a>

**Table 18-2 ADC Module Register Summary (cont'd)**

<b>Short Name</b>	<b>Description</b>	<b>Offset<sup>1)</sup></b>	<b>See Page</b>
<b>LCBR2</b>	Limit Checking Boundary Register 2	88 <sub>H</sub>	<a href="#">Page 18-73</a>
<b>LCBR3</b>	Limit Checking Boundary Register 3	8A <sub>H</sub>	<a href="#">Page 18-73</a>
<b>CHINFR</b>	Channel Event Indication Flag Register	90 <sub>H</sub>	<a href="#">Page 18-74</a>
<b>CHINCR</b>	Channel Event Indication Clear Register	92 <sub>H</sub>	<a href="#">Page 18-75</a>
<b>CHINPR0</b>	Channel Interrupt Node Pointer Register 0	98 <sub>H</sub>	<a href="#">Page 18-76</a>
<b>CHINPR4</b>	Channel Interrupt Node Pointer Register 4	9A <sub>H</sub>	<a href="#">Page 18-76</a>
<b>CHINPR8</b>	Channel Interrupt Node Pointer Register 8	9C <sub>H</sub>	<a href="#">Page 18-77</a>
<b>CHINPR12</b>	Channel Interrupt Node Pointer Register 12	9E <sub>H</sub>	<a href="#">Page 18-78</a>
<b>ALR0</b>	Alias Register 0	1C <sub>H</sub>	<a href="#">Page 18-79</a>

### **Result Registers**

<b>RESR0-7</b>	Result Register 0-7, normal view	40 <sub>H</sub> -4E <sub>H</sub>	<a href="#">Page 18-88</a>
<b>RESRA0-7</b>	Result Register 0-7, view A	50 <sub>H</sub> -5E <sub>H</sub>	<a href="#">Page 18-89</a>
<b>RESRV0-7</b>	Result Register 0-7, view V	60 <sub>H</sub> -6E <sub>H</sub>	<a href="#">Page 18-88</a>
<b>RESRAV0-7</b>	Result Register 0-7, view AV	70 <sub>H</sub> -7E <sub>H</sub>	<a href="#">Page 18-89</a>
<b>VFR</b>	Valid Flag Register	80 <sub>H</sub>	<a href="#">Page 18-91</a>
<b>RSSR</b>	Result Status Shadow Register	82 <sub>H</sub>	<a href="#">Page 18-90</a>
<b>RCR0-7</b>	Result Control Register 0-7	B0 <sub>H</sub> -BE <sub>H</sub>	<a href="#">Page 18-92</a>
<b>EVINFR</b>	Event Indication Flag Register	A0 <sub>H</sub>	<a href="#">Page 18-94</a>
<b>EVINCR</b>	Event Indication Clear Register	A2 <sub>H</sub>	<a href="#">Page 18-95</a>
<b>EVINPR0</b>	Event Interrupt Node Pointer Register 0	A8 <sub>H</sub>	<a href="#">Page 18-96</a>
<b>EVINPR8</b>	Event Interrupt Node Pointer Register 8	AC <sub>H</sub>	<a href="#">Page 18-96</a>
<b>EVINPR12</b>	Event Interrupt Node Pointer Reg. 12	AE <sub>H</sub>	<a href="#">Page 18-97</a>

### **Request Source 0 Registers**

<b>QMR0</b>	Queue 0 Mode Register	E0 <sub>H</sub>	<a href="#">Page 18-54</a>
<b>QSR0</b>	Queue 0 Status Register	E2 <sub>H</sub>	<a href="#">Page 18-57</a>
<b>Q0R0</b>	Queue 0 Register 0	E4 <sub>H</sub>	<a href="#">Page 18-59</a>
<b>QBUR0</b>	Queue 0 Backup Register	E6 <sub>H</sub>	<a href="#">Page 18-61</a>
<b>QINR0</b>	Queue 0 Input Register	shared	<a href="#">Page 18-63</a>

**Table 18-2 ADC Module Register Summary (cont'd)**

Short Name	Description	Offset <sup>1)</sup>	See Page
------------	-------------	----------------------	----------

**Request Source 1 Registers**

<b>CRCR1</b>	Conversion Request 1 Control Register	E8 <sub>H</sub>	<a href="#">Page 18-45</a>
<b>CRPR1</b>	Conversion Request 1 Pending Register	EA <sub>H</sub>	<a href="#">Page 18-46</a>
<b>CRMR1</b>	Conversion Request 1 Mode Register	EC <sub>H</sub>	<a href="#">Page 18-47</a>

**Request Source 2 Registers**

<b>QMR2</b>	Queue 2 Mode Register	F0 <sub>H</sub>	<a href="#">Page 18-54</a>
<b>QSR2</b>	Queue 2 Status Register	F2 <sub>H</sub>	<a href="#">Page 18-57</a>
<b>Q0R2</b>	Queue 2 Register 0	F4 <sub>H</sub>	<a href="#">Page 18-59</a>
<b>QBUR2</b>	Queue 2 Backup Register	F6 <sub>H</sub>	<a href="#">Page 18-61</a>
<b>QINR2</b>	Queue 2 Input Register	shared	<a href="#">Page 18-63</a>

**Additional Feature Registers**

<b>SYNCTR</b>	Synchronization Control Register	1A <sub>H</sub>	<a href="#">Page 18-114</a>
<b>EMENR</b>	External Multiplexer Enable Register	D6 <sub>H</sub>	<a href="#">Page 18-110</a>
<b>EMCTR</b>	External Multiplexer Control Register	D0 <sub>H</sub>	<a href="#">Page 18-112</a>
<b>BWDENR</b>	Broken Wire Detection Enable Register	C8 <sub>H</sub>	<a href="#">Page 18-115</a>
<b>BWDCFGR</b>	Broken Wire Detection Configuration Register	CA <sub>H</sub>	<a href="#">Page 18-116</a>

<sup>1)</sup> Short 8-bit addresses are not available for kernel registers of this module.

<sup>2)</sup> Register KSCFG is available only in the address range of ADC0, named ADC0\_KSCFG.

*Note: The offsets 06<sub>H</sub>, 16<sub>H</sub>, C4<sub>H</sub>, C6<sub>H</sub>, 8C<sub>H</sub>, 8E<sub>H</sub>, A4<sub>H</sub>, A6<sub>H</sub>, and AA<sub>H</sub> are reserved for future use and must not be accessed.*

### **18.2.2 Mode Control**

The mode control concept for system control tasks, such as power saving, or suspend request for debugging, allows to program the module behavior under different device operating conditions. The behavior of the ADC kernels can be programmed for each of the device operating modes, that are requested by the global state control part of the SCU. It is advantageous that the ADC kernels of an ADC module show an identical behavior regarding the device operating modes (e.g. to avoid that a non-suspended kernel waits for a suspended kernel to start a synchronized conversion). Therefore, the ADC module has a common associated register **ADC0\_KSCFG** defining the behavior of all kernels of the module in the following device operating modes:

- **Normal operation:**  
This operating mode is the default operating mode when neither a suspend request nor a clock-off request are pending. The module clock is not switched off and the ADC registers can be read or written. The kernel behavior is defined by KSCFG.NOMCFG.
- **Suspend mode:**  
This operating mode is requested when a suspend request (issued by a debugger) is pending in the device. The module clock is not switched off and the ADC registers can be read or written. The kernel behavior is defined by KSCFG.SUMCFG.
- **Clock-off mode:**  
This operating mode is requested for power saving purposes. The module clock is switched off automatically when all kernels of the ADC module reached their specified state in a stop mode. In this case, ADC registers can not be accessed. The kernel behavior is defined by KSCFG.COMCFG.

For the ADC module, the following internal actions can be influenced by mode control:

- A current conversion of an analog value:  
If the request control unit has found a pending conversion request, the conversion can be started. This start has to be enabled by the mode control. If the current kernel mode allows the conversion start (run modes 0 and 1), it will be executed. If the kernel mode does not allow a start (stop modes 0 and 1), the conversion is not started. The start request is not cancelled, but frozen. A “frozen” conversion is started as programmed if the kernel mode is changed to a run mode again.
- An arbiter round:  
The start of a new arbiter round has to be enabled by the kernel modes. In stop mode 1, a new arbiter round will not start.

The behavior of the ADC kernels can be programmed for each of the device operating modes (normal operation, suspend mode, clock-off mode). Therefore, the ADC kernels support four kernel modes, as shown in **Table 18-3**.

**Table 18-3    ADC Kernel Behavior**

<b>Kernel Mode</b>	<b>Kernel Behavior</b>	<b>Code</b>
run mode 0	kernel operation as specified, no impact on data transfer (same behavior for run mode 0 and run mode 1)	00 <sub>B</sub>
run mode 1		01 <sub>B</sub>
stop mode 0	A currently running AD conversion is completely finished and the result is treated. Pending conversion request to start a new conversion are not taken into account (but not deleted). They start conversions after entering a run mode as programmed. The arbiter continues as programmed.	10 <sub>B</sub>
stop mode 1	Like stop mode 0, but the arbiter is stopped after it has finished its arbitration round. Additionally, bit field GLOBSTR.ANON is considered being 00 <sub>B</sub> when the kernel has reached the defined stop condition (the bit field itself is not changed).	11 <sub>B</sub>

Generally, bit field KSCFG.NOMCFG should be configured for run mode 0 as default setting for standard operation. If the ADC kernels should not react to a suspend request (and to continue operation as in normal mode), bit field KSCFG.SUMCFG has to be configured with the same value as KSCFG.NOMCFG. If the ADC kernels 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 has to be written to KSCFG.SUMCFG.

A similar mechanism applies for the clock-off mode with the possibility to program the desired behavior by bit field KSCFG.COMCFG.

*Note: The stop mode selection strongly depends on the application needs and it is very unlikely that different stop modes are required in parallel in the same application. As a result, only one stop mode type (either 0 or 1) should be used in the bit fields in register KSCFG. Do not mix stop mode 0 and stop mode 1 and avoid transitions from stop mode 0 to stop mode 1 (or vice versa) for the ADC module.*

If the module clock is disabled by KSCFG.MODEN = 0 or in clock-off mode when the stop condition is reached (in stop mode 0 or 1), the module can not be accessed by read or write operations (except register KSCFG that can always be accessed). As a consequence, it can not be configured.

Please note that bit KSCFG.MODEN should only be set by SW while all configuration fields are configured for run mode 0.

### 18.2.3 Module Activation and Power Saving Modes

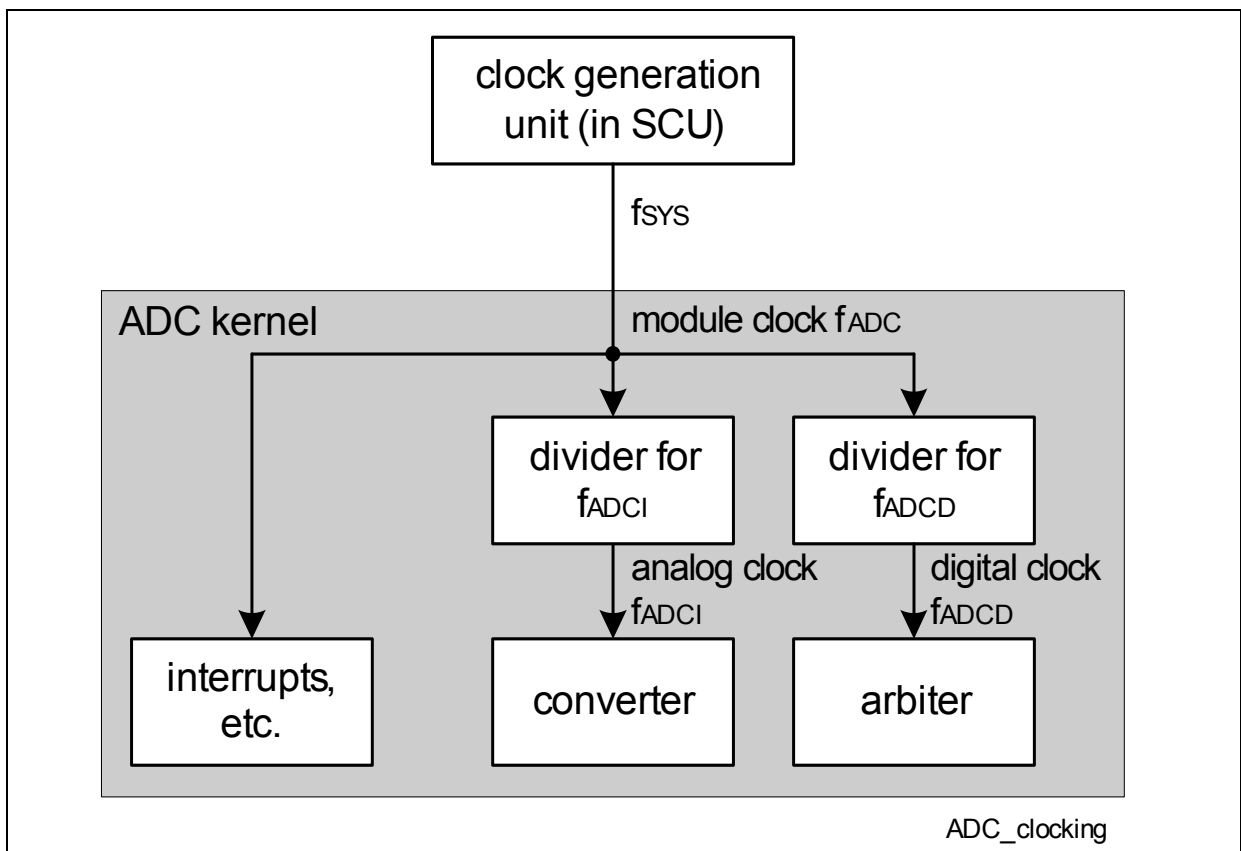
The converter of the ADC supports specific power down modes allowing an automatic reduction of the power consumption between two conversions. The following modes are determined by bit field **GLOBSTR.ANON**:

- **ANON = 00<sub>B</sub>: Converter switched off** (default after reset)  
The complete converter is switched off and held in its reset state, conversions are not possible. To start a conversion, ANON has to be programmed to the desired mode. A maximum wake-up time of about 10 µs has to be respected before starting a conversion. Furthermore, digital logic blocks are set to their initial state.
- **ANON = 01<sub>B</sub>: Slow stand-by mode**  
The converter enters a power reduction mode after each conversion. It switches automatically to normal operation if a conversion is requested. A maximum wake-up time of about 10 µs has to be added to the sample time. This leads to the lowest power consumption for the ADC supply with wake-up capability.
- **ANON = 10<sub>B</sub>: Fast stand-by mode**  
The converter enters a power reduction mode with less reduction than in slow stand-by mode after each conversion. It switches automatically to normal operation if a conversion is requested. A maximum wake-up time of about 3 µs has to be added to the sample time. This leads to a reduced power consumption for the ADC supply compared to normal operation.
- **ANON = 11<sub>B</sub>: Normal operation**  
Conversions are always possible with the desired sample time. The converter stays active permanently.

### 18.2.4 Clocking Scheme

The different parts of an ADC kernel are driven by clock signals that are based on the clock  $f_{ADC}$  of the bus that is used to access the ADC module. The ADCs in the XE16xyM device are connected to the system clock, so  $f_{ADC} = f_{SYS}$ .

- The analog clock  $f_{ADCI}$  is used as internal clock for the converter and defines the conversion length and the sample time. It can be adjusted by programming bit field **GLOBCTR.DIVA**.
- The digital clock  $f_{ADCD}$  is used for the arbiter and defines the duration of an arbiter round. It can be adjusted by programming bit field **GLOBCTR.DIVD**.
- All other digital structures (such as interrupts, etc.) are directly driven by the module clock  $f_{ADC}$ .



**Figure 18-6 Clocking Scheme**

*Note: If the clock generation for the converter of the ADC falls below a minimum value or is stopped during a running conversion, the conversion result can be corrupted. For correct ADC results, the frequency of  $f_{ADCI}$  must not exceed the defined range. Please, refer to the range indicated in the respective Data Sheet.*

## 18.2.5 General ADC Registers

### 18.2.5.1 Kernel State Configuration Register

The kernel state configuration register KSCFG allows the selection of the desired kernel modes for the different device operating modes.

Bit fields KSCFG.NOMCFG and KSCFG.COMCFG are reset by an application reset. Bit field KSCFG.SUMCFG is reset by a debug reset.

This register is a common register for all ADC kernels and can be accessed in the address range of ADC0.

*Note: The coding of the bit fields NOMCFG, SUMCFG and COMCFG is described in [Table 18-3](#).*

#### ADC0\_KSCFG

##### Kernel State Configuration Register

XSFR(0C <sub>H</sub> )										Reset Value: 0000 <sub>H</sub>					
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BP COM	0	COMCFG		BP SUM	0	SUMCFG		BP NOM	0	NOMCFG		0		BP MOD EN	MOD EN
w	r	rw		w	r	rw		w	r	rw		r		w	rw

Field	Bits	Type	Description
<b>MODEN</b>	0	rw	<b>Module Enable</b> This bit enables the module kernel clock and the module functionality. 0 <sub>B</sub> The module is switched off immediately (without respecting a stop condition). It does not react on mode control actions and the module clock is switched off. The module does not react on read accesses and ignores write accesses (except to KSCFG). 1 <sub>B</sub> 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 ADC registers.



Field	Bits	Type	Description
<b>BPMODEN</b>	1	w	<b>Bit Protection for MODEN</b> This bit enables the write access to the bit MODEN. It always reads 0. 0 <sub>B</sub> MODEN is not changed. 1 <sub>B</sub> MODEN is updated with the written value.
<b>NOMCFG</b>	[5:4]	rw	<b>Normal Operation Mode Configuration</b> This bit field defines the kernel mode applied in normal operation mode. 00 <sub>B</sub> Run mode 0 is selected. 01 <sub>B</sub> Run mode 1 is selected. 10 <sub>B</sub> Stop mode 0 is selected. 11 <sub>B</sub> Stop mode 1 is selected.
<b>BPNOM</b>	7	w	<b>Bit Protection for NOMCFG</b> This bit enables the write access to the bit field NOMCFG. It always reads 0. 0 <sub>B</sub> NOMCFG is not changed. 1 <sub>B</sub> NOMCFG is updated with the written value.
<b>SUMCFG</b>	[9:8]	rw	<b>Suspend Mode Configuration</b> This bit field defines the kernel mode applied in suspend mode. Coding like NOMCFG.
<b>BPSUM</b>	11	w	<b>Bit Protection for SUMCFG</b> This bit enables the write access to the bit field SUMCFG. It always reads 0. 0 <sub>B</sub> SUMCFG is not changed. 1 <sub>B</sub> SUMCFG is updated with the written value.
<b>COMCFG</b>	[13:12]	rw	<b>Clock Off Mode Configuration</b> This bit field defines the kernel mode applied in clock-off mode. Coding like NOMCFG.
<b>BPCOM</b>	15	w	<b>Bit Protection for COMCFG</b> This bit enables the write access to the bit field COMCFG. It always reads 0. 0 <sub>B</sub> COMCFG is not changed. 1 <sub>B</sub> COMCFG is updated with the written value.
<b>0</b>	[3:2], 6, 10, 14	r	<b>Reserved</b> returns 0 if read; should be written with 0;

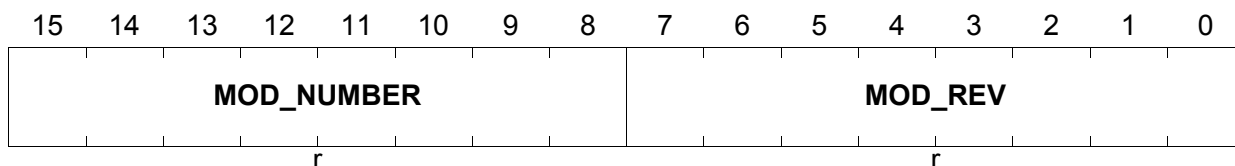
*Note: To change a configuration bit field, the associated protection bit BPxxx must be set during the write operation. This allows modifying selected configuration bit fields with a single write operation.*

### 18.2.5.2 ID Register

The ID register is a read-only register which is used for ADC module identification purposes. It provides 8 bits for module identification and 8 bits for revision numbering.

#### ID

**Module Identification Register**      **XSFR(08<sub>H</sub>)**      **Reset Value: 33XX<sub>H</sub>**



Field	Bits	Type	Description
<b>MOD_REV</b>	[7:0]	r	<b>Module Revision Number Value</b> Bits 7-0 bits are used for module revision numbering. The value of the module revision number starts with 01 <sub>H</sub> (first revision), 02 <sub>H</sub> , 03 <sub>H</sub> , up to FF <sub>H</sub> .
<b>MOD_NUMBER</b>	[15:8]	r	<b>Module Identification Number Value</b> Bits 15-8 are used for module identification. The ADC has the module number 33 <sub>H</sub> .

### 18.2.5.3 Global Control Register

The global control register contains bits to control the timing of the arbiter and the general enable function for the converter.

#### GLOBCTR

#### Global Control Register

**XSFR(10<sub>H</sub>)**

**Reset Value: 00FF<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>ARB M</b>	<b>0</b>			<b>ARBRND</b>		<b>ANON</b>		<b>DIVD</b>		<b>DIVA</b>					
rw	rw			rw		rw		rw		rw					

Field	Bits	Type	Description
<b>DIVA</b>	[5:0]	rw	<b>Divider Factor for Analog Internal Clock</b> This bit field defines the number of $f_{ADC}$ clock cycles to generate the $f_{ADCI}$ clock for the converter (used as internal base for the conversions and the sample time calculation). $00_H \quad f_{ADCI} = f_{ADC}$ $01_H \quad f_{ADCI} = f_{ADC} / 2$ $02_H \quad f_{ADCI} = f_{ADC} / 3$ ... $3F_H \quad f_{ADCI} = f_{ADC} / 64$
<b>DIVD</b>	[7:6]	rw	<b>Divider Factor for Digital Arbiter Clock</b> This bit field defines the number of $f_{ADC}$ clock cycles within each arbitration slot (each arbitration slot lasts one period of $f_{ADCD}$ ). It is recommended to use the default setting $00_B$ to obtain the minimum arbiter reaction time. $00_B \quad f_{ADCD} = f_{ADC}$ $01_B \quad f_{ADCD} = f_{ADC} / 2$ $10_B \quad f_{ADCD} = f_{ADC} / 3$ $11_B \quad f_{ADCD} = f_{ADC} / 4$
<b>ANON</b>	[9:8]	rw	<b>Analog Part Switched On</b> This bit field defines the setting of bit field <b>GLOBSTR.ANON</b> (refer to description of this bit) if this kernel is the synchronization master or without synchronization feature (see register SYNCTR). For a synchronization slave, this bit field is not taken into account.

Field	Bits	Type	Description
<b>ARBRND</b>	[11:10]	rw	<b>Arbitration Round Length</b> This bit field defines the number of arbitration slots per arbitration round (arbitration round length = $t_{ARB}$ ). <sup>1)</sup> 00 <sub>B</sub> An arbitration round contains 4 arbitration slots ( $t_{ARB} = 4 / f_{ADCD}$ ). 01 <sub>B</sub> An arbitration round contains 8 arbitration slots ( $t_{ARB} = 8 / f_{ADCD}$ ). 10 <sub>B</sub> An arbitration round contains 16 arbitration slots ( $t_{ARB} = 16 / f_{ADCD}$ ). 11 <sub>B</sub> An arbitration round contains 20 arbitration slots ( $t_{ARB} = 20 / f_{ADCD}$ ).
<b>0</b>	[14:12]	rw	<b>Reserved for Future Use</b> This bit field is reserved for future use and has to be written with 000 <sub>B</sub> .
<b>ARBM</b>	15	rw	<b>Arbitration Mode</b> This bit field defines whether the arbiter runs permanently or only while at least one conversion request is pending. 0 <sub>B</sub> The arbiter runs permanently. This setting has to be chosen in a synchronization slave (see <a href="#">Section 18.2.18</a> ) and for equidistant sampling using the signal ARBCNT (see <a href="#">Section 18.2.19</a> ). 1 <sub>B</sub> The arbiter only runs if at least one conversion request of an enabled request source is pending. This setting leads to a reproducible latency from an incoming request to the conversion start if the converter is idle. Synchronized conversions are not supported.

<sup>1)</sup> 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.

### 18.2.5.4 Global Status Register

The status control register contains bits indicating the current status of a conversion.

#### GLOBSTR

#### Global Status Register

**XSFR(12<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		CSRC			SYN RUN	ANON		CHNR				0	SAM PLE	BU SY	
r		rh			rh	rh		rh				r	rh	rh	

Field	Bits	Type	Description
<b>BUSY</b>	0	rh	<b>Analog Part Busy</b> This bit indicates that a conversion is currently running. 0 <sub>B</sub> The converter is idle. 1 <sub>B</sub> A conversion is currently running.
<b>SAMPLE</b>	1	rh	<b>Sample Phase</b> This bit indicates that an analog input signal is currently sampled. 0 <sub>B</sub> The converter is not in the sampling phase. 1 <sub>B</sub> The converter is in the sampling phase.
<b>CHNR</b>	[7:3]	rh	<b>Channel Number</b> This bit field indicates which analog input channel is currently converted. This information is updated when a new conversion is started.

Field	Bits	Type	Description
<b>ANON</b>	[9:8]	rh	<p><b>Analog Part Switched On</b></p> <p>This bit field defines the operation mode of the converter. It monitors either bit field GLOBCTR.ANON of the same ADC kernel (in master mode or without synchronization feature) or bit field GLOBCTR.ANON of the ADC kernel selected as synchronization master for this kernel (in slave mode). This ensures that all kernels of a synchronization group can be controlled with a single write operation to bit field GLOBCTR of the synchronization master.</p> <p>00<sub>B</sub> The converter is switched off and conversions are not possible. The arbiter finishes its current arbitration round (if running) and then remains in the idle state.</p> <p>01<sub>B</sub> The converter of the ADC module is switched on and conversions are possible. The automatic power-down capability of the converter is enabled, leading to the lowest power consumption of the ADC supply with wake-up capability, but a longer wake-up time before each conversion.</p> <p>10<sub>B</sub> The converter of the ADC module is switched on and conversions are possible. The automatic power-down capability of the converter is enabled, leading to a reduced power consumption of the ADC supply, but a shorter wake-up time before each conversion.</p> <p>11<sub>B</sub> The converter of the ADC module is switched on and conversions are possible. The automatic power-down capability of the converter is disabled leading to the nominal power consumption of the ADC supply.</p>

Field	Bits	Type	Description
<b>SYNRUN</b>	10	rh	<b>Synchronous Conversion Running</b> This bit indicates that a synchronized (= parallel) conversion is currently running. 0 <sub>B</sub> There is no synchronized conversion running (either there is no conversion currently running or a synchronized conversion has not been requested). A running conversion can be cancelled and repeated in case of a new incoming conversion request with higher priority. 1 <sub>B</sub> A synchronized conversion is running. This conversion can not be cancelled while running. Higher priority requests can trigger conversions only after the end of a synchronized conversion.
<b>CSRC</b>	[13:11]	rh	<b>Currently Converted Request Source</b> This bit field indicates the arbitration slot number of the current conversion (if BUSY = 1, a conversion is still running) or of the last conversion (if BUSY = 0, no conversion is running). This bit field is updated with each conversion start. 000 <sub>B</sub> The channel requested by the request source of arbitration slot 0 is (has been) converted. 001 <sub>B</sub> The channel requested by the request source of arbitration slot 1 is (has been) converted. 010 <sub>B</sub> The channel requested by the request source of arbitration slot 2 is (has been) converted. other combinations are reserved
<b>0</b>	2, [15:14]	r	<b>Reserved</b> returns 0 if read; should be written with 0;

### 18.2.5.5 Input Select Registers

Registers RSIRx contain bit fields selecting the input signal for the trigger and gating inputs of the request sources. The connections depend on the device implementation, please refer to the implementation chapter for details.

#### RSIR0

**Request Source 0 Input Register**    **XSFR(00<sub>H</sub>)**                      **Reset Value: 0000<sub>H</sub>**

#### RSIR1

**Request Source 1 Input Register**    **XSFR(02<sub>H</sub>)**                      **Reset Value: 0000<sub>H</sub>**

#### RSIR2

**Request Source 2 Input Register**    **XSFR(04<sub>H</sub>)**                      **Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TRI	0	R EN	F EN	0	TRSEL			GTI		0		TM EN	0	GTSEL	
rh	r	rw	rw	r	rw			rh		r		rw	r	rw	

Field	Bits	Type	Description
<b>GTSEL</b>	[2:0]	rw	<b>Input Selection for REQGTx of Source x</b> This bit field defines the input signal used for request gating in request source x. 000 <sub>B</sub> The input signal REQGTxA is selected. 001 <sub>B</sub> The input signal REQGTxB is selected. 010 <sub>B</sub> The input signal REQGTxC is selected. 011 <sub>B</sub> The input signal REQGTxD is selected. 100 <sub>B</sub> The input signal REQGTxE is selected. 101 <sub>B</sub> The input signal REQGTxF is selected. 110 <sub>B</sub> The input signal REQGTxG is selected. 111 <sub>B</sub> The input signal REQGTxH is selected.
<b>TMEN</b>	4	rw	<b>Timer Mode Enable of Source x</b> This bit enables the timer mode for equidistant sampling for request source x. 0 <sub>B</sub> The timer mode is disabled. The standard gating mechanism can be used. 1 <sub>B</sub> The timer mode for equidistant sampling is enabled. The standard gating mechanism has to be enabled permanently (no influence of gating signal).



Field	Bits	Type	Description
<b>GTI</b>	7	rh	<b>Gating Input of Source x</b> This flag monitors the status of the selected gating signal REQGTx for request source x. 0 <sub>B</sub> The selected gating signal is 0. 1 <sub>B</sub> The selected gating signal is 1.
<b>TRSEL</b>	[10:8]	rw	<b>Input Selection for REQTRx of Source x</b> This bit field defines the input signal used for request triggering in request source x. 000 <sub>B</sub> The input signal REQTRxA is selected. 001 <sub>B</sub> The input signal REQTRxB is selected. 010 <sub>B</sub> The input signal REQTRxC is selected. 011 <sub>B</sub> The input signal REQTRxD is selected. 100 <sub>B</sub> The input signal REQTRxE is selected. 101 <sub>B</sub> The input signal REQTRxF is selected. 110 <sub>B</sub> The input signal REQTRxG is selected. 111 <sub>B</sub> The input signal REQTRxH is selected.
<b>FEN</b>	12	rw	<b>Falling Edge Enable of Source x</b> This bit enables the request trigger for falling edges of the selected REQTRx signal for request source x. 0 <sub>B</sub> The request trigger with a falling edge is disabled. 1 <sub>B</sub> The request trigger with a falling edge is enabled.
<b>REN</b>	13	rw	<b>Rising Edge Enable of Source x</b> This bit enables the request trigger for rising edges of the selected REQTRx signal for request source x. 0 <sub>B</sub> The request trigger with a rising edge is disabled. 1 <sub>B</sub> The request trigger with a rising edge is enabled.
<b>TRI</b>	15	rh	<b>Trigger Input of Source x</b> This flag monitors the status of the selected trigger signal REQTRx for request source x. 0 <sub>B</sub> The selected trigger signal is 0. 1 <sub>B</sub> The selected trigger signal is 1.
<b>0</b>	3, [6:5], 11, 14	r	<b>Reserved</b> returns 0 if read; should be written with 0.

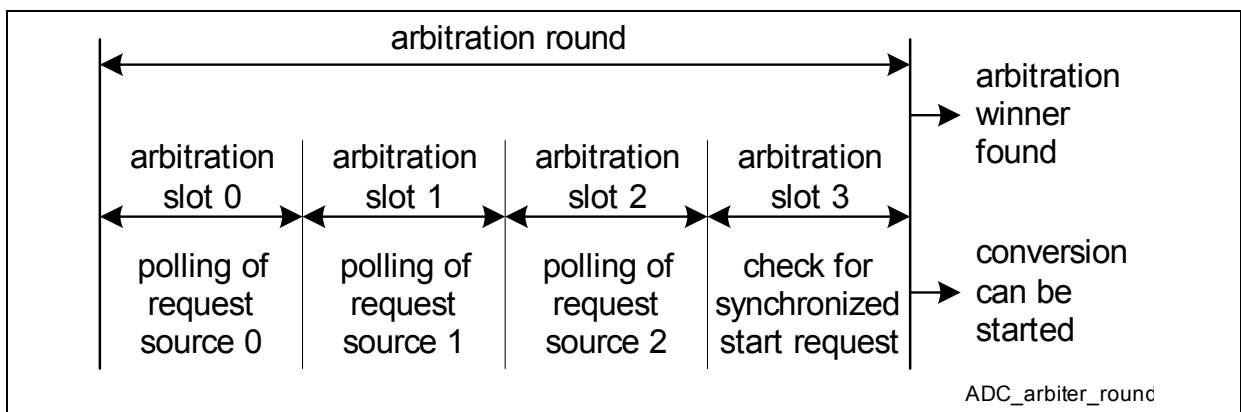
## 18.2.6 Request Source Arbiter

The request source arbiter evaluates which analog input channel has to be converted. Therefore, it regularly polls the request sources one after the other for pending conversion requests. The polling sequence is based on time slots with programmable length, called arbitration slots. If a request source is disabled or if no request source is available for an arbitration slot, the slot is considered as being empty and has no influence on the evaluation of the arbitration winner. After reset, all request sources are disabled and have to be enabled by bits in register **ASENR** to take part in the arbitration process.

An arbitration round consists of one arbitration slot for each available request source plus one final synchronization slot (see **Figure 18-7**). At the end of each arbitration round, the arbiter has determined the request source with the highest priority and a pending conversion request. This arbitration result is stored as arbitration winner for further actions. If a conversion is started in an arbitration round, this arbitration round does not deliver an arbitration winner.

In the XE16xyM, the following request sources are available:

- Request source 0 in arbitration slot 0: **1-stage sequential source**  
This request source can issue a conversion request for a single input channel.
- Request source 1 in arbitration slot 1: **16-channel scan source**  
This request source can issue a conversion request sequence of up to 16 input channels in a defined order.
- Request source 2 in arbitration slot 2: **4-stage sequential source**  
This request source can issue a conversion request sequence of up to 4 input channels in a freely programmable order.
- Last arbitration slot of the arbitration round: **Synchronization source**  
In this slot, the arbiter checks for a synchronized request from another ADC kernel and does not evaluate any internal request source. A request for a synchronized conversion is always handled with the highest priority in a synchronization slave kernel (pending requests from other sources are not considered).



**Figure 18-7 Arbitration Round**

The period  $t_{ARB}$  of an arbitration round is given by:

$$t_{ARB} = N \times (\text{GLOBCTR.DIVD} + 1) / f_{ADC}$$

with N being 4, 8, 16, or 20 as defined by **GLOBCTR.ARBND**

The number of arbitration slots forming an arbitration round can be programmed to obtain a similar arbiter timing for different devices, even if the number of available request sources differs from one device to another.

Because the XE16xyM's ADC has 3 request sources, 4 slots per arbitration round are sufficient.

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 (numbers of arbitration slots and their length).

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 **GLOBCTR.ARBMD** = 0, the **arbiter runs permanently**. In this mode, synchronized conversions of more than one ADC kernel are possible.  
 The trigger for the conversion triggers has to be generated synchronously to the arbiter timing. Incoming triggers should have exactly n-times the granularity of the arbiter ( $n = 1, 2, 3, \dots$ ). In order to allow some flexibility, the duration of an arbitration slot can be programmed in cycles of  $f_{ADC}$ .
- If bit **GLOBCTR.ARBMD** = 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 of a conversion request does need not to be synchronous to the arbiter timing. In this mode, parallel conversions are not possible for synchronization slave kernels.

### **18.2.6.1 Request Source Priority**

Each request source has an individually programmable priority to be able to adapt to different applications (see register **RSPR0**). The priorities define the order the request sources are handled by the arbiter if two or more request sources indicate pending conversion requests at the same time.

Starting with request source 0, the arbiter checks if an enabled request source has a pending request for a conversion. The arbitration winner is the request source with a pending conversion request and the highest priority that has been found first in an arbitration round.

### **18.2.6.2 Conversion Start Modes**

To start the requested conversion of the arbitration winner, the following aspects are automatically taken into consideration by the arbiter:

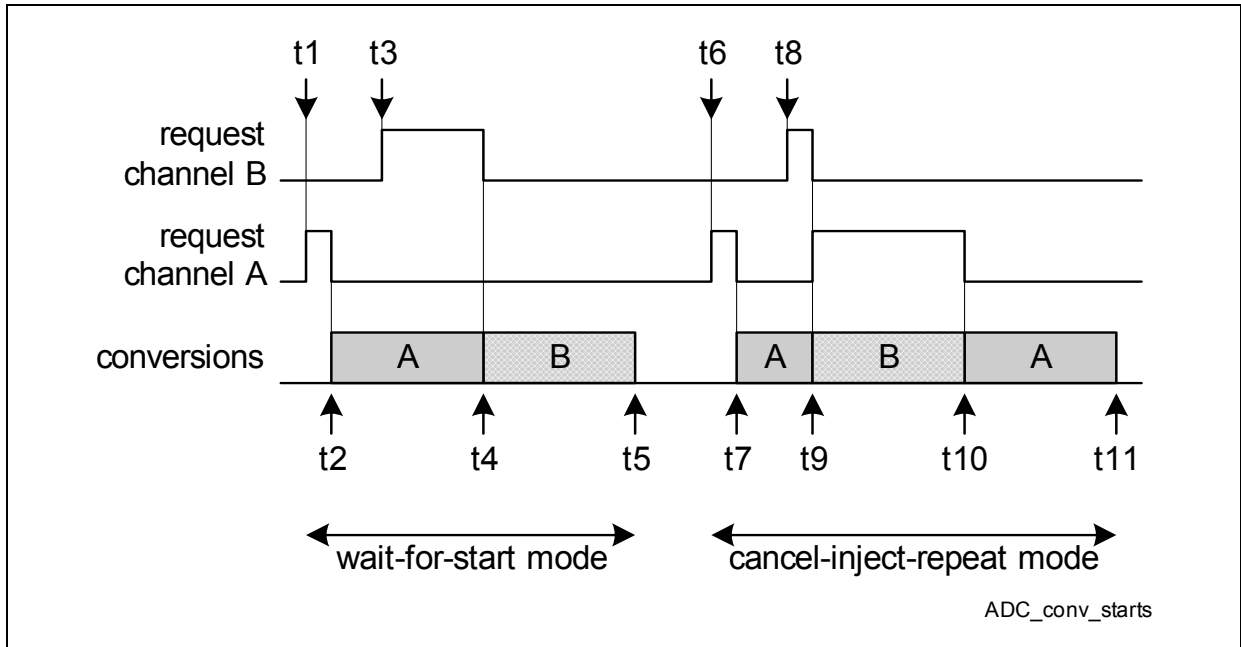
- If the converter is currently idle (no conversion running), the conversion of the arbitration winner is started immediately. If a conversion is currently running, the arbitration winner is compared to the priority of the currently running conversion. The arbiter handles the requested conversion of the arbitration winner in one of the following ways:
- In the case that the current conversion has the same or a higher priority, it is completed. Then, the conversion of the arbitration winner is started.
- In the case that the current conversion has the lower priority and the arbiter winner has been programmed for **wait-for-start mode**, the currently running conversion is completed. Then, the conversion of the arbitration winner is started.

This mode can be used if the timing requirement for the higher priority conversions allow a jitter (between  $t_3$  and  $t_4$  in [Figure 18-8](#)) in the range of a running conversion.

- In the case that the current conversion has the lower priority and the arbiter winner has been programmed for **cancel-inject-repeat mode**, the current conversion is aborted immediately if a new request with a higher priority has been found, unless both requests target the same result register with wait-for-read active (see [Section 18.2.14.2](#)). The conversion of the arbitration winner is started after the abort action. The aborted conversion request is restored in the request source that has requested the aborted conversion. As a consequence, it takes part again in the next arbitration round.

Please note that the abort mechanism can take between 1 and  $3 f_{\text{ADCI}}$  cycles, depending on the state of the current conversion.

This mode can be used if higher priority conversions only tolerate a small jitter (between  $t_8$  and  $t_9$  in [Figure 18-8](#)).



**Figure 18-8 Conversion Start Modes**

The conversion start mode can be individually programmed for each request source by bits in register **RSPR0** and is applied to all channels requested by the source. **Figure 18-8** shows the influence of both conversion start modes on the conversion sequence if two request sources generate conversion requests. 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.

- t1: The trigger event for channel A occurs and a conversion request is activated.
- t2: At the end of the arbitration round, channel A is determined as arbitration winner, the conversion of channel A is started. With the start of the conversion, conversion request A is cleared.
- t3: The trigger event for channel B occurs and a conversion request is activated. In wait-for-read mode, the currently running conversion of channel A is finished normally.
- t4: After the conversion of channel A is finished, the conversion of channel B is started. With the start of the conversion, conversion request B is cleared.
- t5: The conversion of channel B is finished.
- t6: The trigger event for channel A occurs and a conversion request is activated.
- t7: At the end of the arbitration round, channel A is determined as arbitration winner, the conversion of channel A is started. With the start of the conversion, conversion request A is cleared.
- t8: The trigger event for channel B occurs and a conversion request is activated.
- t9: At the end of the arbitration round, channel B is determined as arbitration winner. In cancel-inject-repeat mode, the currently running conversion of channel A is aborted and the conversion of channel B is started. With the abort of conversion A,

**Analog to Digital Converter**

conversion request A is set again. With the start of conversion B, conversion request B is cleared.

- t10: The conversion of channel B is finished. In the meantime, the pending request for channel A has been identified as arbitration winner and the conversion of channel A is started. With the start of the conversion, conversion request A is cleared.
- t11: The conversion of channel A is finished.



### 18.2.7.2 Request Source Priority Register

The request source priority register contains bits to define the request source priority and the conversion start mode. The priority and conversion start mode settings of an enabled request source must not be changed by SW. If a request source is disabled, the setting can be changed by SW if a currently running conversion requested by this source is finished.

#### **RSPR0**

#### **Request Source Priority Register 0**

**XSFR(14<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

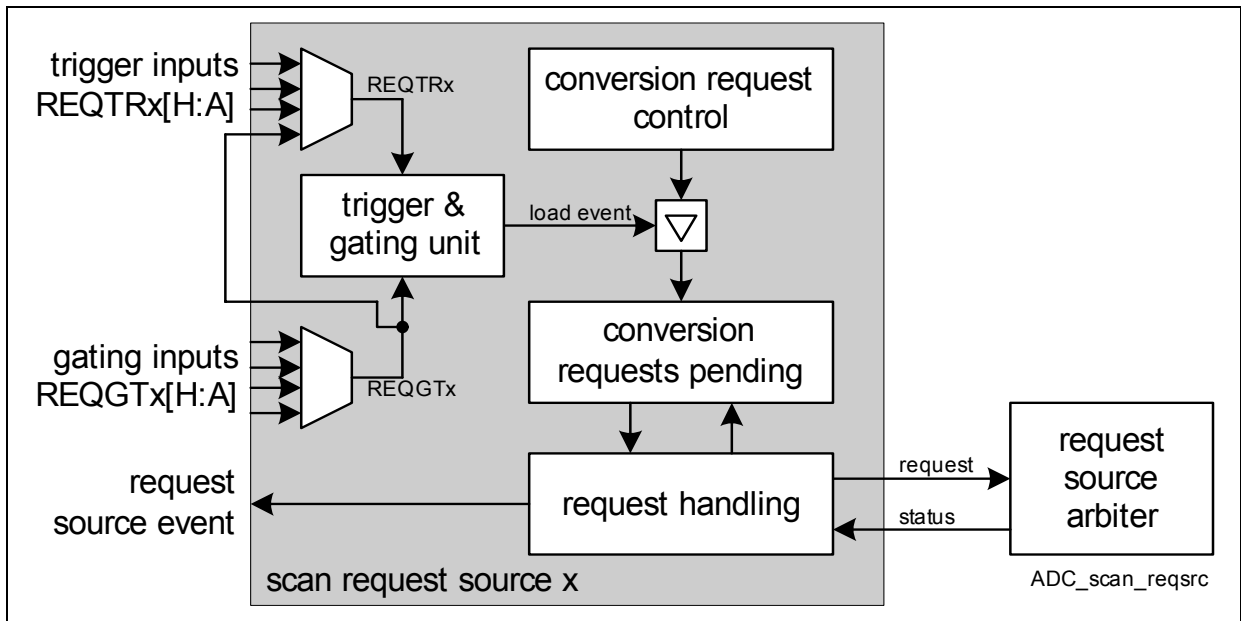
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0			<b>CSM</b> 2	0	<b>PRIO</b> 2	<b>CSM</b> 1	0	<b>PRIO</b> 1	<b>CSM</b> 0	0	<b>PRIO</b> 0				
r			rw	r	rw	rw	r	rw	rw	r	rw				

Field	Bits	Type	Description
<b>PRI00, PRI01, PRI02</b>	[1:0], [5:4], [9:8]	rw	<b>Priority of Request Source x</b> This bit field defines the priority of the conversion request source x, located in arbitration slot x. 00 <sub>B</sub> Lowest priority is selected. ... 11 <sub>B</sub> Highest priority is selected.
<b>CSM0, CSM1, CSM2</b>	3, 7, 11	rw	<b>Conversion Start Mode of Request Source x</b> This bit defines the conversion start mode of the conversion request source x, located in arbitration slot x. 0 <sub>B</sub> Wait-for-start mode is selected. 1 <sub>B</sub> Cancel-inject-repeat mode is selected.
<b>0</b>	2, 6, 10, [15:12]	r	<b>Reserved</b> returns 0 if read; should be written with 0;



## 18.2.8 Scan Request Source Handling

A scan request source can issue conversion requests for a sequence of up to 16 input channels. It can be programmed individually for each input channel if it takes part in the scan sequence. The scan sequence always starts with the highest enabled channel number and continues towards lower channel numbers (order defined by the channel number, each channel can be converted only once per sequence).



**Figure 18-9 Scan Request Source**

### 18.2.8.1 Overview

A scan request source performs the:

- **Conversion request control:**  
The conversion request control defines if an analog input channel takes part in the scan sequence (see bits in register **CRCR1**). The programmed register value is kept unchanged by an ongoing scan sequence.
- **Conversion request pending:**  
The pending conversion requests indicate if an input channel has to be converted in an ongoing scan sequence (see bits in register **CRPR1**). A conversion request can only be issued to the request source arbiter if at least one pending bit is set. With each conversion start that has been triggered by the scan request source, the corresponding pending bit is automatically cleared. The scan sequence is considered finished and a request source event is generated if the last conversion triggered by the scan source is finished and all pending bits have been cleared.
- **Request handling:**  
The request handling block interfaces with the request source arbiter. It requests conversion due to pending bits in the scan sequence and handles the conversion

status information. If a conversion triggered by the scan request source is aborted due to a conversion request from another request source with a higher priority, the corresponding pending bit is automatically set. This mechanism ensures that an aborted conversion takes part in the next arbitration round and does not get lost. The control of the scan sequence is done based on bits in register **CRMR1**.

- **Trigger and gating signal handling:**  
 The trigger and gating unit interfaces with signals and modules outside the ADC module that can request conversions. For example, a timer unit can issue a request signal to synchronize conversions to PWM events. A load event starts a scan sequence by modifying the request pending bits according to the request control bits.

### 18.2.8.2 Scan Sequence Operation

To **operate a scan request source**, the following aspects should be taken into account:

- The bits in register **CRCR1** have to be programmed to define the channels participating in the scan sequence.
- If a trigger or gating function by external signals is desired, the gating and trigger inputs have to be defined by bit fields TRSEL and GTSEL in register **RSIR1**. Also the edge selection for the trigger event is done in this register.
- The gating mechanism has to be defined by **CRMR1**.ENGT.
- The corresponding arbitration slot has to be enabled to accept conversion requests from the scan source (see register **ASENR**).
- The load event has to be defined by bits in **CRMR1** to start a scan sequence.
- If a load event occurs while **CRMR1**.LDM = 0, the content of **CRCR1** is copied to **CRPR1** (overwrite). This setting allows starting a new scan sequence and to “forget” remaining pending bits if a load event occurs while a scan sequence is running.
- If a load event occurs while **CRMR1**.LDM = 1, the content of **CRCR1** is bit-wisely logical OR-combined to **CRPR1** (no overwrite). This setting allows starting a new scan sequence without “forgetting” remaining pending bits if a load event occurs while a scan sequence is running.

To **start a scan sequence**, the following mechanisms are supported to generate a load event:

- An external trigger signal can be selected to start a scan sequence controlled by HW by an external module or signal, e.g. a timer unit or an input pin. The trigger feature is enabled by **CRMR1**.ENTR = 1. The load event is generated if the selected edge is detected at the selected trigger input. The edge selection is done in register **RSIRx**.
- A load event is generated under SW control by writing **CRMR1**.LDEV = 1. This mechanism starts a scan sequence without modifying the bits in register **CRCR1**. A data write action to **CRCR1** does not lead to a load event (first prepare the channel control, then start the sequence).
- If SW writes data to register **CRPR1**, the written data is stored in register **CRCR1** and a load event is generated automatically. This mechanism starts a scan sequence with

the channels defined by the written data (the sequence is defined and started with a single data write action, e.g. under PEC control).

- A load event is generated each time a scan sequence has finished and the request source event occurs if bit **CRMR1**.SCAN = 1. This setting leads to a permanent repetition of the scan sequence.

To **stop or abort an ongoing scan sequence**, the following mechanisms are supported:

- An external gating signal can be selected to stop and to continue a scan sequence at any point in time controlled by an external module or signal, e.g. a timer unit or an input pin. The gating feature can be enabled and the polarity of the gating signal REQGTx can be selected by **CRMR1**.ENGT. The gating mechanism does not modify the contents of the conversion pending bits, but only prevents the request handling block from issuing conversion requests to the arbiter.
- The arbiter can be disabled by SW for this arbiter slot by clearing the corresponding bit **ASENR**.ASENx. This mechanism does not modify the contents of the conversion pending bits, but only prevents the arbiter from accepting requests from the request handling block.
- The pending request bits can be cleared by writing bit **CRMR1**.CLRPND = 1. It is recommended to stop the scan sequence before clearing the pending bits.

### 18.2.8.3 Request Source Event and Interrupt

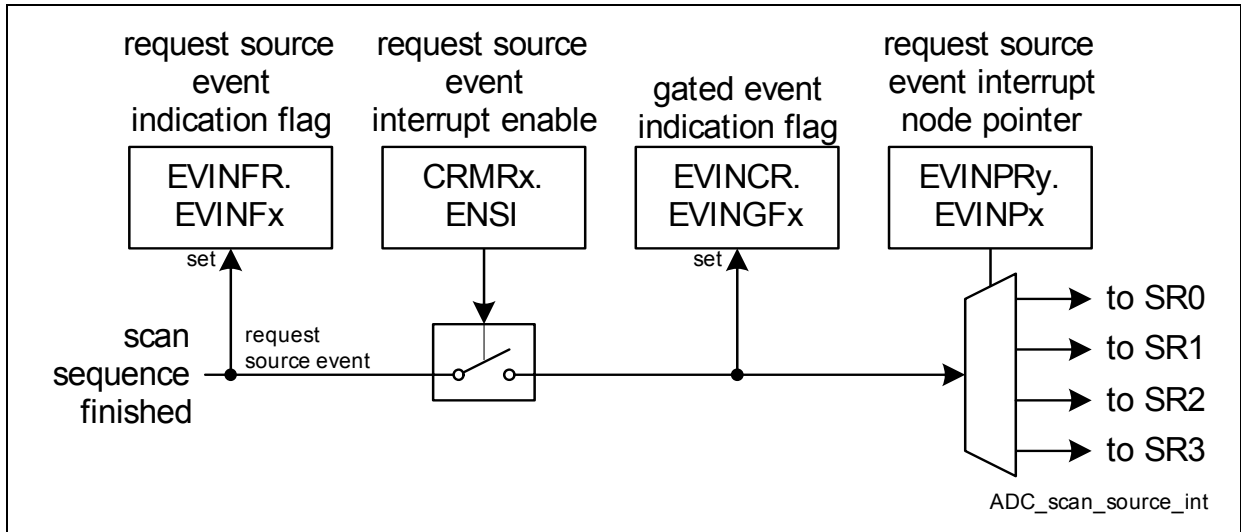
A request source event of a scan source occurs if the last conversion of a scan sequence is finished (all pending bits = 0). A request source event interrupt can be generated based on a request source event according to the structure shown in **Figure 18-10**. If a request source event is detected, it sets the corresponding indication flag in register **EVINFR**. These flags can also be set by writing a 1 to the corresponding bit position, whereas writing 0 has no effect. Additionally, a gated event flag **EVINCR**.EVINGFx indicates that a request source interrupt has been activated. The indication flags can be cleared by SW by writing a 1 to the corresponding bit position in register **EVINCR**.

The service request output SRx that is selected by the request source event interrupt node pointer bit fields in register **EVINPRO** becomes activated each time the related request source event is detected (and enabled by CRMRx.ENS1) or the related bit position in register **EVINFR** is written with a 1 (this write action simulates a request source event).

Additionally, a gated event indication flag **EVINCR**.EVINGFx (after the gating with the enable bit) becomes set if a service request output becomes activated due to a request source event.

The request source events and the result events share the same registers. The request source event is located at the bit position in register **EVINFR**:

- Event 1 (indicated by bit EVINF0):  
Request source event of scan source in arbitration slot 1.



**Figure 18-10 Interrupt Generation of a Scan Request Source**

## **18.2.9 Scan Request Source Registers**

### **18.2.9.1 Conversion Request Control Register**

The register contains the control and status bits of the scan request source(s). The index 1 describes the number of the arbitration slot where the request source is taking part in the arbitration.

The conversion request control register contains the bits that are copied to the pending register when the load event occurs. Register **CRCR1** is updated by write accesses to register **CRPR1**, and can be accessed (read and written) directly.

#### **CRCR1**

##### **Conversion Request 1 Control Register**

**XSFR(E8<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>CH 15</b>	<b>CH 14</b>	<b>CH 13</b>	<b>CH 12</b>	<b>CH 11</b>	<b>CH 10</b>	<b>CH 9</b>	<b>CH 8</b>	<b>CH 7</b>	<b>CH 6</b>	<b>CH 5</b>	<b>CH 4</b>	<b>CH 3</b>	<b>CH 2</b>	<b>CH 1</b>	<b>CH 0</b>
rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

<b>Field</b>	<b>Bits</b>	<b>Type</b>	<b>Description</b>
<b>CHx</b> <b>(x = 0-15)</b>	x	rwh	<b>Channel Bit x</b> Each bit corresponds to one analog input channel, the channel number CHx is defined by the bit position x in this register. 0 <sub>B</sub> The analog channel CHx will not be requested for conversion by this scan request source. 1 <sub>B</sub> The analog channel CHx will be requested for conversion by this scan request source.

### 18.2.9.2 Conversion Request Pending Register

The conversion request pending register contains the bits that are requesting a conversion of the corresponding analog channel.

A write operation to **CRPR1** leads to a data write to the bits in **CRCR1** with an automatic load event generation.

A read operation to **CRPR1** delivers the pending bits.

#### **CRPR1**

#### **Conversion Request 1 Pending Register**

**XSFR(EA<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>CHP</b> <b>15</b>	<b>CHP</b> <b>14</b>	<b>CHP</b> <b>13</b>	<b>CHP</b> <b>12</b>	<b>CHP</b> <b>11</b>	<b>CHP</b> <b>10</b>	<b>CHP</b> <b>9</b>	<b>CHP</b> <b>8</b>	<b>CHP</b> <b>7</b>	<b>CHP</b> <b>6</b>	<b>CHP</b> <b>5</b>	<b>CHP</b> <b>4</b>	<b>CHP</b> <b>3</b>	<b>CHP</b> <b>2</b>	<b>CHP</b> <b>1</b>	<b>CHP</b> <b>0</b>
rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
<b>CHPx</b> <b>(x = 0-15)</b>	x	rwh	<b>Channel Pending Bit x</b> <u>Write view:</u> A write to this address targets the bits in register CRCR1. <u>Read view:</u> Each bit corresponds to one analog channel, the channel number CHx is defined by the bit position in the register. 0 <sub>B</sub> The analog channel CHx is not requested for conversion by this request source. 1 <sub>B</sub> The analog channel CHx is requested for conversion by this request source.

### 18.2.9.3 Conversion Request Mode Register

The conversion request mode register contains bits to configure the desired operating mode of the scan request source.

#### CRMR1

#### Conversion Request 1 Mode Register

XSFR(EC <sub>H</sub> )										Reset Value: 0000 <sub>H</sub>					
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0						LD EV	CLR PND	REQ GT	0	LD M	SC AN	EN SI	EN TR	ENGT	
r						w	w	rh	r	rw	rw	rw	rw	rw	

Field	Bits	Type	Description
ENGT	[1:0]	rw	<b>Enable Gate</b> This bit field enables the gating functionality for the request source. 00 <sub>B</sub> The request source does not issue conversion requests. 01 <sub>B</sub> The request source issues conversion requests if at least one pending bit is set. 10 <sub>B</sub> The request source issues conversion requests if at least one pending bit is set and the selected gating signal REQGTx = 1. 11 <sub>B</sub> The request source issues conversion requests if at least one pending bit is set and the selected gating signal REQGTx = 0.
ENTR	2	rw	<b>Enable External Trigger</b> This bit enables the external trigger possibility. If enabled, the load event takes place if the selected edge is detected at the selected trigger input signal REQTR. 0 <sub>B</sub> The external trigger is disabled. 1 <sub>B</sub> The external trigger is enabled.
ENSI	3	rw	<b>Enable Source Interrupt</b> This bit enables the request source interrupt generation if a request source event occurs (last pending conversion is finished). 0 <sub>B</sub> The request source interrupt is disabled. 1 <sub>B</sub> The request source interrupt is enabled.

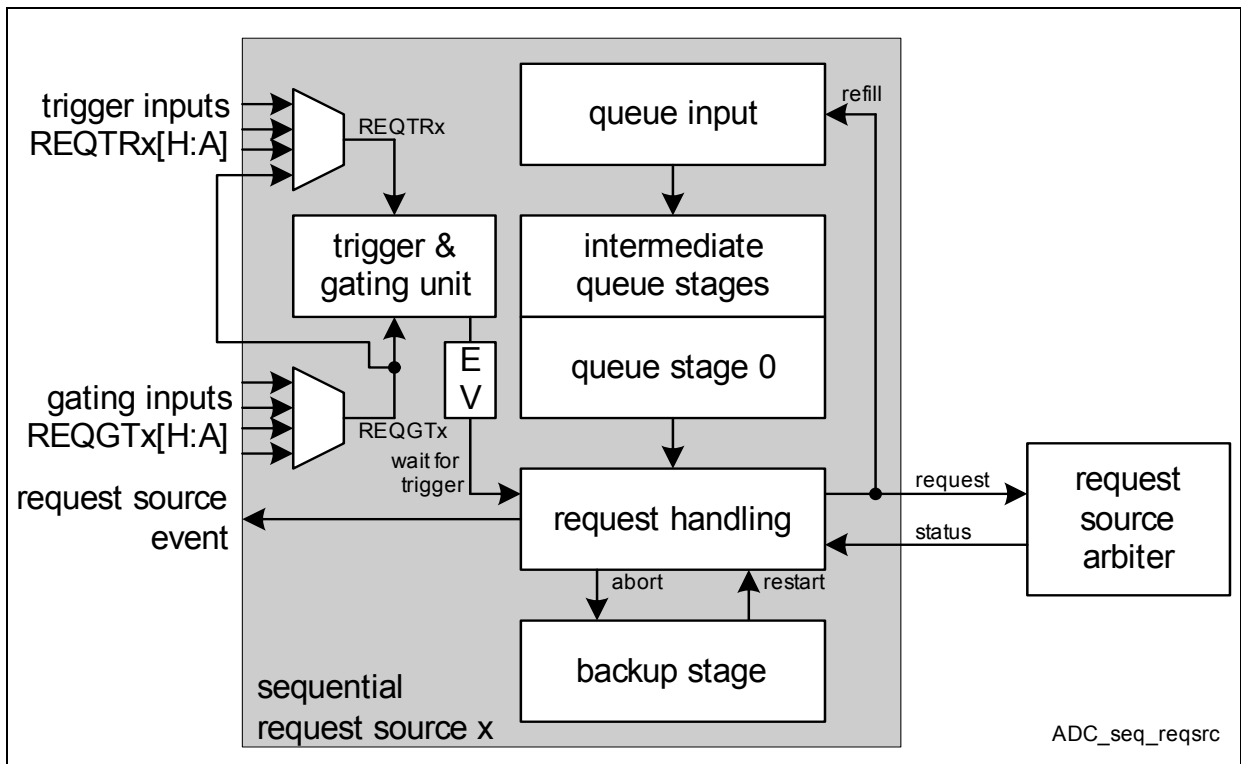
Field	Bits	Type	Description
<b>SCAN</b>	4	rw	<b>Autoscan Enable</b> This bit enables a permanent scan functionality. If enabled, the load event is automatically generated if a request source event occurs. 0 <sub>B</sub> The permanent scan functionality is disabled. 1 <sub>B</sub> The permanent scan functionality is enabled.
<b>LDM</b>	5	rw	<b>Load Event Mode</b> This bit defines the transfer mechanism triggered by the load event. 0 <sub>B</sub> With the load event, the value of register CRCRx is copied to the pending register CRPRx (overwrite). 1 <sub>B</sub> With the load event, the value of register CRCRx is bit-wisely logical OR combined to the pending register CRPRx.
<b>REQGT</b>	7	rh	<b>Request Gate Level</b> This bit monitors the level at the REQGT input. 0 <sub>B</sub> The level is 0. 1 <sub>B</sub> The level is 1.
<b>CLRPND</b>	8	w	<b>Clear Pending Bits</b> 0 <sub>B</sub> No action. 1 <sub>B</sub> The bits in register CRPRx are cleared.
<b>LDEV</b>	9	w	<b>Generate Load Event</b> 0 <sub>B</sub> No action. 1 <sub>B</sub> A load event is generated.
<b>0</b>	6, [15:10]	r	<b>Reserved</b> returns 0 if read; should be written with 0;



### 18.2.10 Sequential Request Source Handling

Sequential request sources have been introduced to allow short conversion sequences with freely programmable channel numbers (contrary to a scan request source with a fixed conversion order for the enabled channels). Two versions of the sequential sources are available in each ADC kernel:

- Request source in arbitration slot 2:  
 This request source can handle a sequence of up to 4 input channels (4-stage queue for 4 entries). This mechanism could be used to support application-specific conversion sequences, especially for timing-critical sequences containing multiple conversions of the same channel.
- Request source in arbitration slot 0:  
 This request source can handle a single input channel (1-stage queue for 1 entry). This mechanism could be used for SW-controlled conversion requests or HW-triggered conversions of a single input channel (to “inject” a single conversion into a running sequence).



**Figure 18-11 Sequential Request Source**

The internal structure and the handling of the sequential sources is similar for both versions. The programmed sequence is stored in a queue buffer (based on a FIFO mechanism) with at least one queue stage (stage 0) and a backup stage for aborted conversions. The only difference between both versions is given by the number of intermediate queue stages for storing the sequence. The request source in arbitration

slot 0 does not provide intermediate queue stages (1-stage queue with only queue stage 0), whereas the one in arbitration slot 2 provides 3 intermediate queue stages in addition to queue stage 0 (leading to a 4-stage queue).

### 18.2.10.1 Overview

A sequential request source performs the:

- Queue input:  
The queue input represents the programming interface where the sequence is defined (see [QINR0](#), [QINR2](#)). It does not provide any buffer capability, but handles the filling of the queue buffer (queue stage 0 plus optional intermediate queue stages) by writing data to it. The contents of the queue stages can not be directly modified by program, except by the command for flushing the complete queue.  
The queue input also handles the refill mechanism, an automatic re-insertion of a started conversion from queue stage 0 (including the control parameters) as new queue input. This feature allows a single setup (by SW) of a conversion sequence and multiple repetitions of the same sequence without the need to re-program it each time. A conversion sequence is repeated automatically if all queue entries of the sequence are setup for refill mode.
- Queue stage 0:  
The content of this queue stage defines which channel will be requested next for a conversion (see [QOR0](#), [QOR2](#)). It also defines if the request should be triggered by an external event or if the requested conversion should follow the previous one as soon as possible. It also enables the request source interrupt generation after the conversion.  
The contents of this queue stage is cleared when the requested conversion is started and the next queue entry can be handled (if available).
- Queue backup stage:  
The queue backup stage is used to store the request control parameters when a conversion requested by this request source is aborted. A validation bit indicates that the aborted conversion has to be requested next (before the current contents of queue stage 0) to maintain the original sequence (see [QBUR0](#), [QBUR2](#)).
- Request handling:  
The request handling block interfaces with the request source arbiter. It requests a conversion due to valid information in queue stage 0 and handles the conversion status information. The control of the queue sequence is done based on bits in registers [QMR0](#) (for the request source in arbitration slot 0) and [QMR2](#) (for the arbitration slot 2).
- Trigger and gating signal handling:  
The trigger and gating unit interfaces with signals and modules outside the ADC module that can request conversions. For example, a timer unit can issue a request signal to synchronize conversions to PWM events. A trigger event can start a conversion request for the entry in queue stage 0 (see [QMR0](#), [QMR2](#)). An event flag

QSRx.EV indicates that a trigger event has been detected (selected edge of selected trigger input signal REQTRx if enabled by QMRx.ENTR or write action with QMRx.TREV = 1). This bit is cleared with each conversion start requested by this source or by writing bits CEV = 1, FLUSH = 1, or CLRV = 1.

### **18.2.10.2 Sequential Source Operation**

To **operate a sequential request source**, the following aspects should be taken into account:

- The sequence has to be initialized by writing to the queue input **QINR0** (for arbitration slot 0) or **QINR2** (for arbitration slot 2) when using the refill mechanism. Each write access corresponds to one conversion request.  
The desired sequence should be completely initialized before enabling the request source, because with enabled refill feature, write accesses by SW to QINRx are not allowed.
- If a trigger or gating function by external signals is desired, the gating and trigger inputs have to be defined by bit fields TRSEL and GTSEL in register **RSIR0** (for arbitration slot 0) or **RSIR2** (for arbitration slot 2). Also the edge selection for the trigger event is done in these registers.
- The gating mechanism has to be defined by QMRx.ENGTL.
- If an external trigger mechanism is desired, it has to be enabled by QMRx.ENTR = 1.
- The corresponding arbitration slot has to be enabled to accept conversion requests from the sequential source (see register **ASENR**).

To **start a sequence** of a sequential request source, the following mechanisms are supported:

- An external trigger signal can be selected to start a queue sequence controlled by HW by an external module or signal, e.g. a timer unit or an input pin. The trigger feature is enabled by QMRx.ENTR = 1. The trigger event is generated if the selected edge is detected at the selected trigger input.
- A trigger event is generated under SW control by writing QMRx.TREV = 1. This mechanism starts a request if queue stage 0 contains valid data (or the queue backup stage respectively).
- A write operation to a queue input leads to a (new) valid queue entry. If the queue is empty (no valid entry), the written data arrives in queue stage 0 and starts a conversion request (if enabled by QMRx.ENGTL and without waiting for an external trigger). If the refill mechanism is used, the queue inputs must not be written while the queue is running. Write operations to a completely filled queue are ignored.

To **stop or abort an ongoing sequence** of a sequential request source, the following mechanisms are supported:

- An external gating signal can be selected to stop and to continue a sequence at any point in time controlled by an external module or signal, e.g. a timer unit or an input pin. The gating feature can be enabled and the polarity of the gating signal can be

selected by QMRx.ENGTL. The gating mechanism does not modify the queue entries, but only prevents the request handling block from issuing conversion requests to the arbiter.

- The arbiter can be disabled by SW for this arbiter slot by clearing the corresponding bit **ASEN**.ASENx. This mechanism does not modify the queue entries, but only prevents the arbiter from accepting requests from the request handling block.
- The next pending queue entry is cleared by writing bit QMRx.CLRV = 1. It is recommended to stop the sequence before clearing a queue entry (ENGL = 00<sub>B</sub>). If the queue backup stage contains a valid entry, this one is cleared, otherwise a valid entry in queue register 0 is cleared.
- All queue entries are cleared by writing bit QMRx.FLUSH = 1. It is recommended to stop the sequence before clearing queue entries.

### 18.2.10.3 Request Source Event and Interrupt

A request source event occurs when a conversion that has been requested by this source is completely finished. The interrupt enable bits are located in the queue 0 register (if this has not been a repeated start after an abort) or in the queue backup register (if this has been a repeated start after an abort).

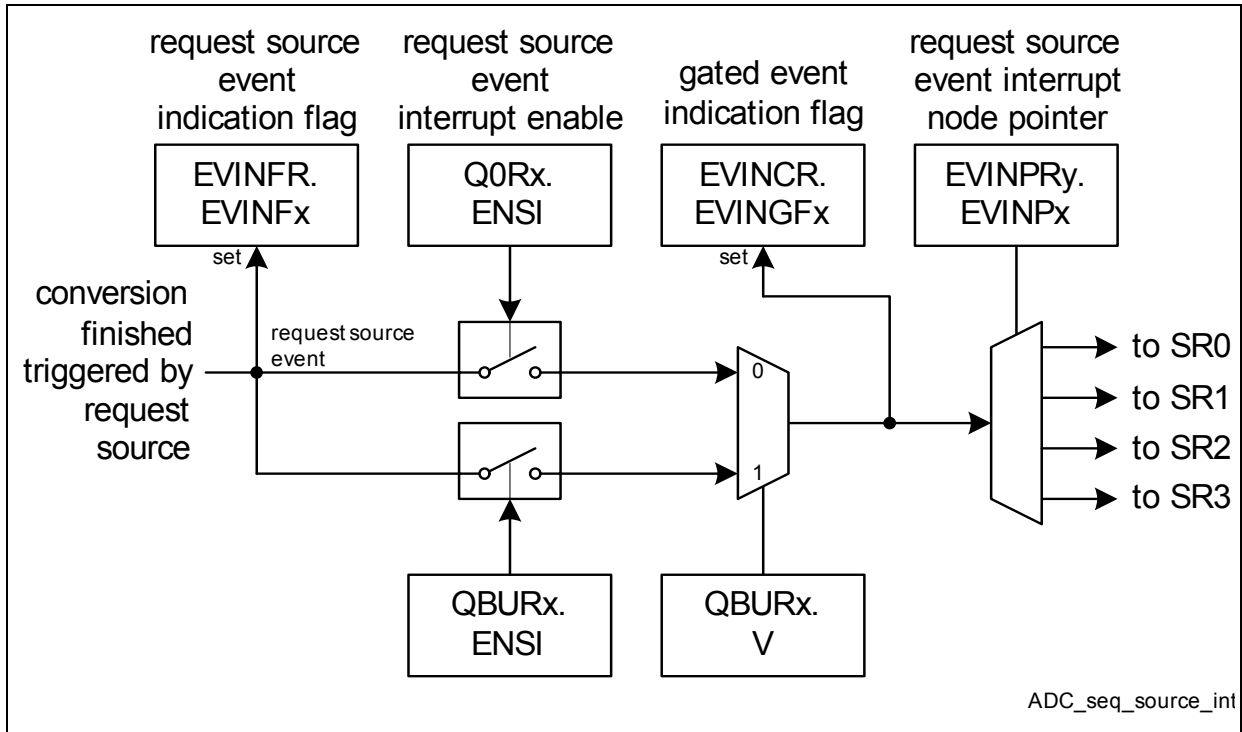
A request source event interrupt can be generated based on a request source event according to the structure shown in **Figure 18-12**. If a request source event is detected, it sets the corresponding indication flag in register **EVINFR**. These flags can also be set by writing a 1 to the corresponding 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 **EVINCR**.

The service request output line SRx that is selected by the request source event interrupt node pointer bit fields in register **EVINPRO** issues an interrupt each time the related request source event is detected (and enabled by Q0Rx.ENSEI, or QBURx.ENSEI respectively) or the related bit position in register **EVINFR** is written with a 1 (this write action simulates a request source event).

Additionally, a gated event indication flag **EVINCR.EVINGFx** (after the gating with the enable bit) becomes set if a service request output becomes activated due to a request source event.

The request source events and the result events share the same registers. The request source event is located at the bit position in register **EVINFR**:

- Event 0: Request source event of sequential source in arbitration slot 0.
- Event 2: Request source event of sequential source in arbitration slot 2.



**Figure 18-12 Interrupt Generation of a Sequential Request Source**

## 18.2.11 Sequential Source Registers

### 18.2.11.1 Queue Mode Register

These registers contain the control bits of a sequential source.

The index 0/2 describes the number of the arbitration slot where the request source is taking part in the arbitration.

*Note: Before SW modifies the queue content by QMR.CLRV or QMR.FLUSH, all HW actions related to this queue have to be finished. Therefore, the arbitration slot has to be disabled and SW has to wait for at least two arbitration rounds (to be sure that this request source can no longer be an arbitration winner). Then, it has to check **GLOBSTR**.CRSC and **GLOBSTR**.BUSY to be sure that a conversion triggered by this request source is no longer running. Then SW can read QBURx and QORx and can start modification of the queue content.*

#### **QMR0**

**Queue 0 Mode Register**

**XSFR(E0<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

#### **QMR2**

**Queue 2 Mode Register**

**XSFR(F0<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				CEV	FLU SH	TR EV	CLR V	0				EN TR	ENGT		
r				w	w	w	w	r				rw	rw		

Field	Bits	Type	Description
<b>ENGT</b>	[1:0]	rw	<p><b>Enable Gate</b>  This bit field enables the gating functionality for the request source.</p> <p>00<sub>B</sub> The request source does not issue conversion requests.</p> <p>01<sub>B</sub> The request source issues conversion requests if a valid conversion request is pending in the queue 0 register or in the backup register.</p> <p>10<sub>B</sub> The request source issues conversion requests if a valid conversion request is pending in the queue 0 register or in the backup register and the selected gating signal REQGTx = 1.</p> <p>11<sub>B</sub> The request source issues conversion requests if a valid conversion request is pending in the queue 0 register or in the backup register and the selected gating signal REQGTx = 0.</p>
<b>ENTR</b>	2	rw	<p><b>Enable External Trigger</b>  This bit enables the external trigger possibility.</p> <p>0<sub>B</sub> The external trigger is disabled and the trigger event is not generated.</p> <p>1<sub>B</sub> The external trigger is enabled and a trigger event is generated if the selected edge is detected at the selected trigger input signal for REQTRx.</p>
<b>CLRV</b>	8	w	<p><b>Clear V Bit</b></p> <p>0<sub>B</sub> No action.</p> <p>1<sub>B</sub> 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.</p>

Field	Bits	Type	Description
<b>TREV</b>	9	w	<b>Trigger Event</b> 0 <sub>B</sub> No action. 1 <sub>B</sub> A trigger event is generated by SW. If the a valid entry in the request source waits for a trigger event, a conversion request is started.
<b>FLUSH</b>	10	w	<b>Flush Queue</b> 0 <sub>B</sub> No action. 1 <sub>B</sub> All entries in the queue (including the backup stage) and the event flag EV are cleared. The queue contains no more valid entry.
<b>CEV</b>	11	w	<b>Clear Event Flag</b> 0 <sub>B</sub> No action. 1 <sub>B</sub> Bit EV is cleared.
<b>0</b>	[7:3], [15:12]	r	<b>Reserved</b> returns 0 if read; should be written with 0;



### 18.2.11.2 Queue Status Register

The queue status register contains bits indicating the status of the sequential 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).

#### QSR0

**Queue 0 Status Register**

**XSFR(E2<sub>H</sub>)**

**Reset Value: 0020<sub>H</sub>**

#### QSR2

**Queue 2 Status Register**

**XSFR(F2<sub>H</sub>)**

**Reset Value: 0020<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0							EV	REQ GT	0	EMP TY	0			FILL	
r							rh	rh	r	rh	r			rh	

Field	Bits	Type	Description
<b>FILL</b>	[1:0]	rh	<b>Filling Level<sup>1)</sup></b> This bit field indicates how many queue entries are valid in the sequential source. It is incremented each time a new entry is written to QINRx or by an enabled refill mechanism. It is decremented each time a requested conversion has been started. A new entry is ignored if the filling level has reached its maximum value. 00 <sub>B</sub> EMPTY = 1: There is no valid entry in the queue. EMPTY = 0: There is 1 valid entries in the queue. 01 <sub>B</sub> There are 2 valid entries in the queue. 10 <sub>B</sub> There are 3 valid entries in the queue. 11 <sub>B</sub> There are 4 valid entries in the queue.
<b>EMPTY</b>	5	rh	<b>Queue Empty</b> This bit indicates if the sequential source contains valid entries. 0 <sub>B</sub> There are FILL+1 valid entries in the queue. 1 <sub>B</sub> There are no valid entries (queue is empty).

Field	Bits	Type	Description
<b>REQGT</b>	7	rh	<b>Request Gate Level</b> This bit monitors the level at the selected REQGT input. 0 <sub>B</sub> The level is 0. 1 <sub>B</sub> The level is 1.
<b>EV</b>	8	rh	<b>Event Detected</b> This bit indicates that an event has been detected while at least one valid entry has been in the queue (queue register 0 or backup stage). Once set, this bit is cleared automatically when the requested conversion is started. 0 <sub>B</sub> A trigger event has not been detected. 1 <sub>B</sub> A trigger event has been detected.
<b>0</b>	[4:2], 6, [15:9]	r	<b>Reserved</b> returns 0 if read; should be written with 0;

<sup>1)</sup> This bit field is always 00<sub>B</sub> for the 1-stage queue in arbitration slot 0.

### 18.2.11.3 Queue 0 Register

The queue registers 0 monitor the status of the pending request (queue stage 0).

**Q0R0**

**Queue 0 Register 0**

**XSFR(E4<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

**Q0R2**

**Queue 2 Register 0**

**XSFR(F4<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>0</b>							<b>V</b>	<b>EX TR</b>	<b>EN SI</b>	<b>RF</b>	<b>REQCHNR</b>				
r							rh	rh	rh	rh	rh				

Field	Bits	Type	Description
<b>REQCHNR</b>	[4:0]	rh	<b>Request Channel Number</b> This bit field indicates the requested channel number.
<b>RF</b>	5	rh	<b>Refill</b> This bit indicates if the pending request is discarded after the conversion start or if it is automatically refilled into the queue input of the request queue. 0 <sub>B</sub> The request is discarded after the conversion start. 1 <sub>B</sub> The request is refilled into the queue after the conversion start.
<b>ENSI</b>	6	rh	<b>Enable Source Interrupt</b> This bit indicates if a request source event interrupt is generated when the conversion is finished. 0 <sub>B</sub> The request source event interrupt generation is disabled. 1 <sub>B</sub> The request source event interrupt generation is enabled.
<b>EXTR</b>	7	rh	<b>External Trigger</b> This bit indicates if a valid queue entry immediately leads to a conversion request or if the request handler waits for a trigger event. 0 <sub>B</sub> The request handler does not wait for a trigger event. 1 <sub>B</sub> The request handler waits for a trigger event.

Field	Bits	Type	Description
<b>V</b>	8	rh	<b>Request Channel Number Valid</b> This bit indicates if the queue register 0 contains a valid queue entry. 0 <sub>B</sub> The queue entry is not valid and does not lead to a conversion request. 1 <sub>B</sub> The queue entry is valid and leads to a conversion request.
<b>0</b>	[15:9]	r	<b>Reserved</b> returns 0 if read; should be written with 0;

#### 18.2.11.4 Queue Backup Register

The queue backup registers monitor the status of an aborted sequential request.

The registers QBURx and QINRx share the same register address. A read operation at this register address will deliver the “rh” bits of register QBURx. A write operation to this address will target the “w” bits in register QINRx.

##### QBUR0

**Queue 0 Backup Register**

**XSFR(E6<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

##### QBUR2

**Queue 2 Backup Register**

**XSFR(F6<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0							V	EXT R	EN SI	RF	REQCHNR				
r							rh	rh	rh	rh	rh				

Field	Bits	Type	Description
REQCHNR	[4:0]	rh	<b>Request Channel Number</b> This bit field contains the channel number of an aborted conversion that has been requested by this request source.
RF	5	rh	<b>Refill</b> This bit contains the refill bit of an aborted conversion that has been requested by this request source.
ENSI	6	rh	<b>Enable Source Interrupt</b> This bit contains the request source event interrupt enable bit of an aborted conversion that has been requested by this request source.
EXTR	7	rh	<b>External Trigger</b> This bit contains the external trigger bit of an aborted conversion that has been requested by this request source.

Field	Bits	Type	Description
<b>V</b>	8	rh	<b>Request Channel Number Valid</b> This bit indicates if the entry in the queue backup register is valid (REQCHNR, RF, TR and ENSI are valid). Bit V is set if a running conversion that has been requested by this request source is aborted. It is cleared when the repeated conversion is started. 0 <sub>B</sub> The backup register does not contain a valid entry. 1 <sub>B</sub> The backup register contains a valid entry. It will be requested before a valid entry in queue register 0 will be requested.
<b>0</b>	[15:9]	r	<b>Reserved</b> returns 0 if read; should be written with 0;



Field	Bits	Type	Description
<b>EXTR</b>	7	w	<b>External Trigger</b> This bit defines the external trigger functionality. $0_B$ A valid queue entry immediately leads to a conversion request. $1_B$ A valid queue entry waits for a trigger event to occur before issuing a conversion request.
<b>0</b>	[15:8]	r	<b>Reserved</b> returns 0 if read; should be written with 0;



### **18.2.12 Channel-Related Functions**

The channel control unit defines the conversion settings, that can be programmed individually for each analog input channel. Therefore, a channel control register CHCTR<sub>x</sub> (see [Section 18.2.13.1](#)) is associated to each analog input channel CH<sub>x</sub>. After the arbiter has determined the channel to be converted, the defined settings are applied to the AD converter, comprising information about:

- **Conversion parameters:**  
Bit field ICLSEL defines which input class is taken into account for the conversion (see [Section 18.2.12.1](#)).
- **Reference selection:**  
Bit field REFSEL defines which reference input is used for the conversion (see [Section 18.2.12.2](#))
- **Channel event handling:**  
Bit fields LCC, BNDASEL, and BNDBSEL define which boundaries are used for limit checking (see [Section 18.2.12.4](#)) and which channel event leads to a channel event interrupt (see [Section 18.2.12.5](#)).
- **Synchronous conversion request:**  
Bit SYNC defines if the channel triggers a synchronized conversion (see [Section 18.2.18](#)).

In addition to the general channel control, the ADC kernel supports a mechanism (named alias feature, see [Section 18.2.12.3](#)) to redirect a conversion request to another channel number.

#### **18.2.12.1 Input Classes**

An input class defines the length of the sample phase and the resolution of the conversion. In most applications, the characteristics of the input circuitries (RC input low-pass filter and impedance of the signal source) are quite similar for several analog input signals, leading to similar timings for the sample phase of these channels. As a consequence, input channels with similar parameters can be grouped together to form an input class.

All channels with the same ICLSEL setting belong to the same input class and have the same sample phase length and resolution. In the XE16xyM, 2 input classes are supported. Registers **INPCR<sub>x</sub> (x = 0 - 1)** can be programmed to adjust the sample time and the resolution to the application requirements independently for each input class.

The default setting of these registers lead to the minimum sample phase length of 2  $f_{\text{ADCI}}$  cycles and conversions with 10 bits resolution. If this default setting fits to the application requirements, bit fields CHCTR<sub>x</sub>.ICLSEL and registers **INPCR<sub>x</sub> (x = 0 - 1)** need not to be changed.

### 18.2.12.2 Reference Selection

The conversion result of the ADC is always referring to a reference voltage. The maximum digital result value (full scale) is obtained if the analog input voltage equals the reference voltage. In order to support more than one measurement range with full scale digital representation, the user can select between the standard reference input  $V_{AREF}$  and an alternative reference input at the analog input channel CH0 for each ADC kernel. The reference selection can be individually programmed for each input channel.

This feature can be used to connect 5 V based sensors and 3.3 V based sensors to the same ADC kernel. In this case, one set of input channels refers to the standard reference input, whereas the other one refers to the voltage level at input CH0.

Please note that the smallest granularity 1 LSB<sub>n</sub> for n bit resolution refers to the selected reference voltage. The granularity becomes very small if a low reference voltage is applied, and as a consequence, the resulting TUE increases due to noise effects. Therefore, it is recommended to avoid small reference voltages.

### 18.2.12.3 Alias Feature

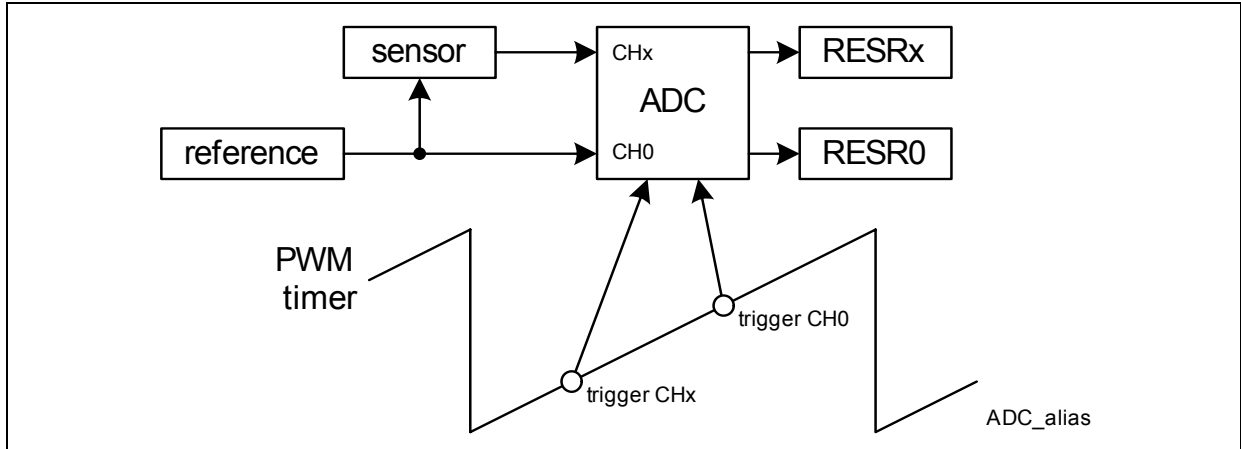
The ADC kernel provides an alias feature, allowing a re-direction of conversion requests for channels CH0 or CH1 to other channel numbers. This feature can be used to measure the same input channel and to store the conversion results in two different result registers.

- 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 interrupt latency.
- The sensor signal is connected to only one input channel (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 18-13](#)), the internal trigger and data handling features for channel CH0 can be used.
- The channel settings for both conversions can be different (boundary values, interrupts, etc.).
- If a sequential conversion request source has been set up, a conversion request for channels CH0 or CH1 can be easily directed to other input channels without flushing the queue.

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 18-13](#) 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

## Analog to Digital Converter

results can be stored and read out from the result registers RESRx (conversion triggered for CHx) and RESR0 (conversion triggered for CH0). Additionally, different interrupts or limit boundaries can be selected, enabled or disabled.



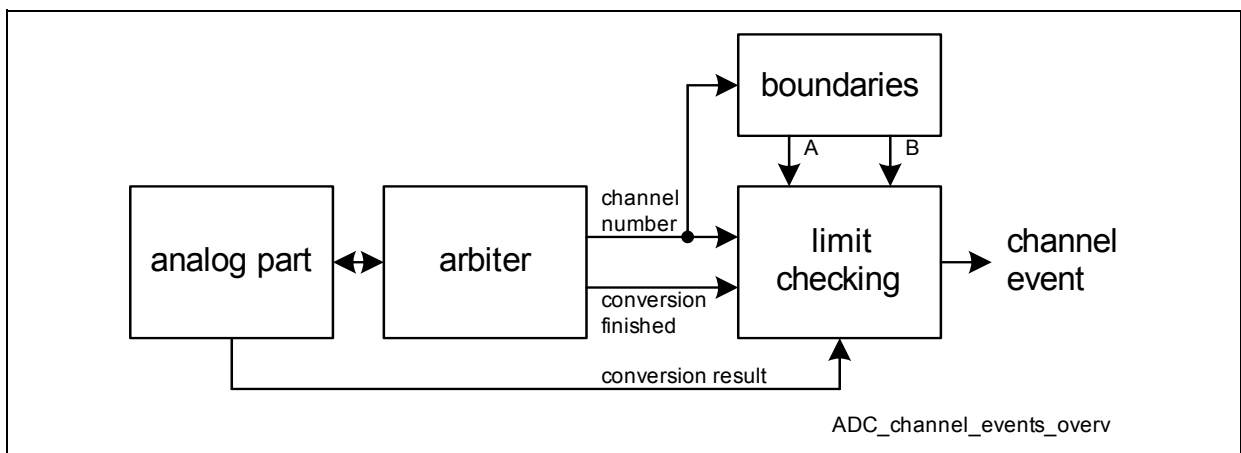
**Figure 18-13 Alias Feature**

### 18.2.12.4 Limit Checking

The limit checking mechanism automatically compares each conversion result to two boundary values (boundaries A and B). For each channel, the user can select these boundaries from a set of 4 programmable values (**LCBR0** to **LCBR3**).

With this structure, the conversion result range is split into three areas:

- Area I: The conversion result is below or equal to both boundaries.
- Area II: The conversion result is above one boundary and below or equal to the other boundary.
- Area III: The conversion result is above both boundaries.



**Figure 18-14 Channel Event Generation**

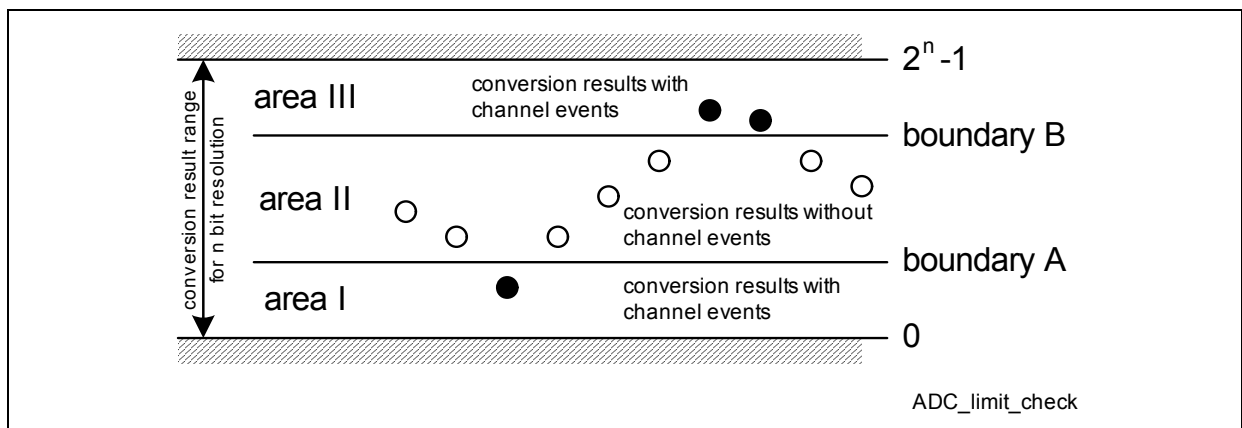
Bit field LCC in the channel control register defines the condition to generate a channel event, leading to a channel event interrupt:

- LCC = 000<sub>B</sub>: No trigger, the channel event generation is disabled.
- LCC = 001<sub>B</sub>: A channel event is generated if the conversion result is not in area I.
- LCC = 010<sub>B</sub>: A channel event is generated if the conversion result is not in area II.
- LCC = 011<sub>B</sub>: A channel event is generated if the conversion result is not in area III.
- LCC = 100<sub>B</sub>: A channel event is always generated (regardless of the boundaries).
- LCC = 101<sub>B</sub>: A channel event is generated if the conversion result is in area I.
- LCC = 110<sub>B</sub>: A channel event is generated if the conversion result is in area II.
- LCC = 111<sub>B</sub>: A channel event is generated if the conversion result is in area III.

**Figure 18-15** shows an example for limit checking where channel events are generated only if the conversion results are not in the normal operating range defined by area II (LCC = 010<sub>B</sub>).

Typical applications for limit checking are temperature monitoring or overcurrent sensing. As long as the measured temperature value is below a boundary value, the CPU does not need to be informed. In this case, a channel event should be generated only if the conversion result is in area III (LCC = 111<sub>B</sub>) to indicate an over-temperature condition. If the conversion of the analog temperature input signal is part of an auto-scan sequence autonomously triggered on a regular time base, the CPU load for the temperature monitoring is zero until the over-temperature condition is detected.

In the case of an over-current protection, the channel event can be used to disable PWM generation to reduce the current (in the XE16xyM, an interrupt output line of the ADC module is connected to a corresponding input of the CCU6x units to allow fast reactions without CPU intervention).

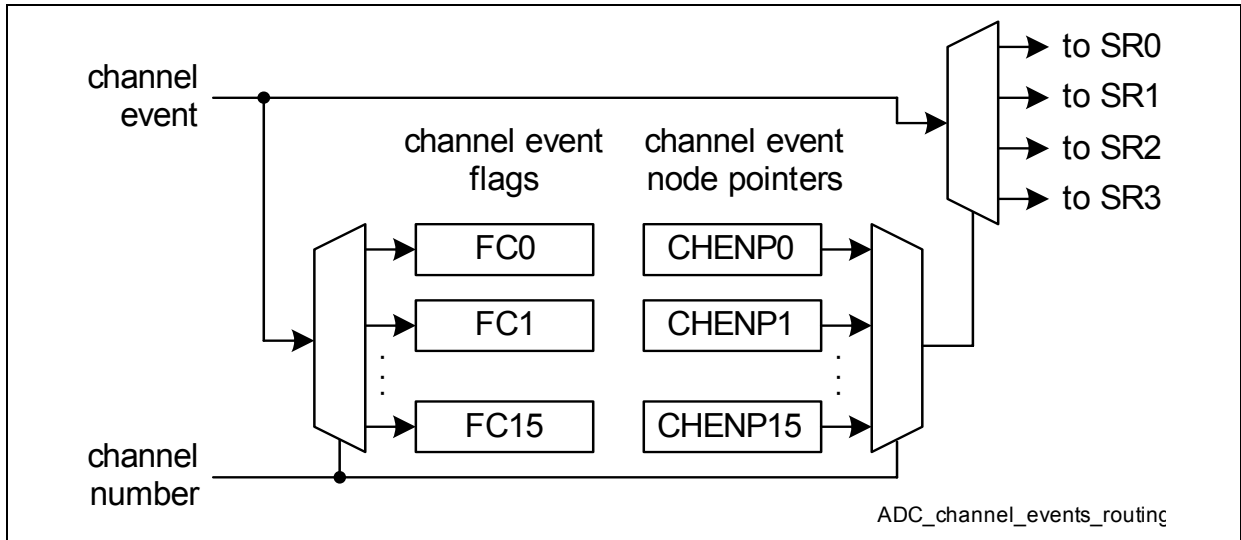


**Figure 18-15 Limit Checking**

*Note: It is also possible to select the same boundary register for boundaries A and B. In this case, the conversion result range is split into two ranges (area II is empty).*

### 18.2.12.5 Channel Event Interrupts

A channel event interrupt can be generated based on a channel event according to the structure shown in **Figure 18-16**. If a channel event is detected, it sets the corresponding indication flag in register **CHINFR**. 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 **CHINCR**.



**Figure 18-16 Channel Event Interrupt Generation**

The service request output line SRx that is selected by the channel node pointer bit fields in registers **CHINPR0**, **CHINPR4**, **CHINPR8**, or **CHINPR12** is activated each time the related channel event is detected or the related bit position in register **CHINFR** is written with a 1.

## 18.2.13 Channel-Related Registers

### 18.2.13.1 Channel Control Registers

The channel control registers contain bits to select the targeted result register, to control the limit check mechanism and to select an input class.

The channel control register 0 defines the settings for the input channel CH0, etc.

**CHCTR<sub>x</sub> (x = 0 - 15)**

**Channel x Control Register      XSFR(20<sub>H</sub> + x \* 2)      Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>0</b>	<b>RESR SEL</b>		<b>ICL SEL</b>		<b>REF SEL</b>		<b>SYN C</b>	<b>LCC</b>		<b>BNDB SEL</b>		<b>BNDA SEL</b>			
r	rw		rw		rw		rw	rw		rw		rw			

Field	Bits	Type	Description
<b>BNDASEL</b>	[1:0]	rw	<b>Boundary A Selection</b> This bit field defines which boundary will be taken as boundary A for the limit checking. 00 <sub>B</sub> The value given by register LCBR0 is selected. 01 <sub>B</sub> The value given by register LCBR1 is selected. 10 <sub>B</sub> The value given by register LCBR2 is selected. 11 <sub>B</sub> The value given by register LCBR3 is selected.
<b>BNDBSEL</b>	[3:2]	rw	<b>Boundary B Selection</b> This bit field defines which boundary will be taken as boundary B for the limit checking. 00 <sub>B</sub> The value given by register LCBR0 is selected. 01 <sub>B</sub> The value given by register LCBR1 is selected. 10 <sub>B</sub> The value given by register LCBR2 is selected. 11 <sub>B</sub> The value given by register LCBR3 is selected.
<b>LCC</b>	[6:4]	rw	<b>Limit Check Control</b> This bit field defines the behavior of the limit checking mechanism. Please refer to the coding in <a href="#">Section 18.2.12.4</a> on <a href="#">Page 18-67</a> .

Field	Bits	Type	Description
<b>SYNC</b>	7	rw	<b>Synchronization Request</b> This bit defines if a conversion request for this channel leads to a synchronized (parallel) conversion with other ADC kernels. This bit is only taken into account if the ADC kernel is a potential conversion master ( <b>SYNCTR</b> .STSEL = 00), otherwise it is considered to be 0. 0 <sub>B</sub> This channel does not request a synchronized conversion. 1 <sub>B</sub> This channel requests a synchronized conversion if the ADC kernel is a potential synchronization master.
<b>REFSEL</b>	[9:8]	rw	<b>Reference Input Selection</b> This bit field defines the reference source for this channel. 00 <sub>B</sub> The standard reference input V <sub>AREF</sub> is selected. 01 <sub>B</sub> The alternative reference input CH0 is selected. 10 <sub>B</sub> reserved, do not use 11 <sub>B</sub> reserved, do not use
<b>ICLSEL</b>	[11:10]	rw	<b>Input Class Selection</b> These bits are used to select the input class. 00 <sub>B</sub> The input class 0 is selected. 01 <sub>B</sub> The input class 1 is selected. 10 <sub>B</sub> reserved, do not use 11 <sub>B</sub> reserved, do not use
<b>RESRSEL</b>	[14:12]	rw	<b>Result Register Selection</b> This bit field defines which result register will be the target of the conversion result of this channel. 000 <sub>B</sub> The result register 0 is selected. 001 <sub>B</sub> The result register 1 is selected. ... 111 <sub>B</sub> The result register 7 is selected.
<b>0</b>	15	r	<b>Reserved</b> returns 0 if read; should be written with 0;

### 18.2.13.2 Input Class Registers

The input class registers contain bits to control the sample time and the resolution for each input class.

The input class register 0 defines the settings for the input class 0, etc.

**INPCR<sub>x</sub> (x = 0 - 1)**

Input Class Register x						XSFR(C0 <sub>H</sub> + x * 2)						Reset Value: 0000 <sub>H</sub>			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>0</b>						<b>DW</b>		<b>STC</b>							
r						rw		rw							

Field	Bits	Type	Description
<b>STC</b>	[7:0]	rw	<b>Sample Time Control</b> This bit field defines the additional length of the sample phase, given in analog clock cycles $f_{\text{ADCI}}$ . A minimum sample phase of 2 analog clock cycles is extended by the programmed value. sample phase length = $(2 + \text{STC}) / f_{\text{ADCI}}$
<b>DW</b>	[9:8]	rw	<b>Data Width</b> This bit field defines how many bits are converted for the result <sup>1)</sup> . The MSBs of conversion results with different DW settings are left aligned in the result bit fields. Bit positions that are not converted are 0. 00 <sub>B</sub> The result is 10 bits wide. 01 <sub>B</sub> reserved, do not use 10 <sub>B</sub> The result is 8 bits wide. 11 <sub>B</sub> reserved, do not use
<b>0</b>	[15:10]	r	<b>Reserved</b> returns 0 if read; should be written with 0;

<sup>1)</sup> The setting 00<sub>B</sub> is default. In case a 10-bit AD converter is used, the combinations 01<sub>B</sub> and 11<sub>B</sub> are ignored by the converter and treated like 00<sub>B</sub>.



### 18.2.13.3 Limit Check Boundary Registers

The bit fields in these registers define compare value (boundary) for the limit checking unit. The reset values of the boundaries are defined as 10%, 90%, 33% and 66% of the complete result range (in 12-bit representation).

#### LCBR0

**Limit Check Boundary Register 0** XSFR(84<sub>H</sub>) **Reset Value: 0198<sub>H</sub>**

#### LCBR1

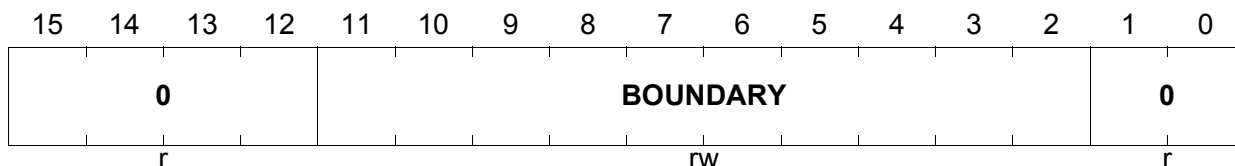
**Limit Check Boundary Register 1** XSFR(86<sub>H</sub>) **Reset Value: 0E64<sub>H</sub>**

#### LCBR2

**Limit Check Boundary Register 2** XSFR(88<sub>H</sub>) **Reset Value: 0554<sub>H</sub>**

#### LCBR3

**Limit Check Boundary Register 3** XSFR(8A<sub>H</sub>) **Reset Value: 0AA8<sub>H</sub>**



Field	Bits	Type	Description
<b>BOUNDARY</b>	[11:2]	rw	<b>Boundary for Limit Checking</b> This bit field contains the value for the limit checking unit that is compared to the actual conversion result. The result of the limit check is used for the generation of the channel event, see <a href="#">Section 18.2.12.4</a> .
<b>0</b>	[1:0], [15:12]	r	<b>Reserved</b> returns 0 if read; should be written with 0;

### 18.2.13.4 Channel Event Indication Flag Register

The channel event indication flag register CHINFR monitors the detected channel events.

#### CHINFR

**Channel Event Indication Flag RegisterXSFR(90<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>CHIN F15</b>	<b>CHIN F14</b>	<b>CHIN F13</b>	<b>CHIN F12</b>	<b>CHIN F11</b>	<b>CHIN F10</b>	<b>CHIN F9</b>	<b>CHIN F8</b>	<b>CHIN F7</b>	<b>CHIN F6</b>	<b>CHIN F5</b>	<b>CHIN F4</b>	<b>CHIN F3</b>	<b>CHIN F2</b>	<b>CHIN F1</b>	<b>CHIN F0</b>
rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
<b>CHINFx</b> <b>(x = 0 - 15)</b>	x	rwh	<b>Channel x Event Indication Flag</b> Flag CHINFx indicates that a channel event for channel x has been detected. Writing a 0 has no effect, whereas writing a 1 sets the written bit position and generates the corresponding interrupt request. 0 <sub>B</sub> A channel x event has not yet been detected. 1 <sub>B</sub> A channel x event has been detected.

### 18.2.13.5 Clear Channel Event Indication Register

Writing a 1 to a bit position in the channel indication clear register CHINCR clears the corresponding channel event indication flag CHINFR in register **CHINFR**. If a channel event is detected when the corresponding bit position is written with a 1, flag CHINFR is cleared.

#### **CHINCR**

#### **Channel Event Indication Clear Register**

**XSFR(92<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>CHIN C15</b>	<b>CHIN C14</b>	<b>CHIN C13</b>	<b>CHIN C12</b>	<b>CHIN C11</b>	<b>CHIN C10</b>	<b>CHIN C9</b>	<b>CHIN C8</b>	<b>CHIN C7</b>	<b>CHIN C6</b>	<b>CHIN C5</b>	<b>CHIN C4</b>	<b>CHIN C3</b>	<b>CHIN C2</b>	<b>CHIN C1</b>	<b>CHIN C0</b>
W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W

Field	Bits	Type	Description
<b>CHINCx</b> (x = 0 - 15)	x	w	<b>Clear Channel Indication Flag</b> 0 <sub>B</sub> No action. 1 <sub>B</sub> Flag CHINFR.x is cleared.

### 18.2.13.6 Channel Interrupt Node Pointer Registers

The bit fields in these registers define the service request output ADCx\_SR[3:0] that is used to signal a channel event interrupt.

#### CHINPR0

##### Channel Interrupt Node Pointer Register 0

**XSFR(98<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>0</b>	<b>CHINP3</b>	<b>0</b>	<b>CHINP2</b>	<b>0</b>	<b>CHINP1</b>	<b>0</b>	<b>CHINP0</b>	<b>0</b>	<b>CHINP3</b>	<b>0</b>	<b>CHINP2</b>	<b>0</b>	<b>CHINP1</b>	<b>0</b>	<b>CHINP0</b>
r	rw	r	rw	r	rw	r	rw	r	rw	r	rw	r	rw	r	rw

Field	Bits	Type	Description
<b>CHINP0, CHINP1, CHINP2, CHINP3</b>	[1:0], [5:4], [9:8], [13:12]	rw	<b>Interrupt Node Pointer for Channel x</b> This bit field selects which service request output indicates a channel event interrupt of channel x. 00 <sub>B</sub> Service request output SR0 is selected. 01 <sub>B</sub> Service request output SR1 is selected. 10 <sub>B</sub> Service request output SR2 is selected. 11 <sub>B</sub> Service request output SR3 is selected.
<b>0</b>	[3:2], [7:6], [11:10], [15:14]	r	<b>Reserved</b> returns 0 if read; should be written with 0;

#### CHINPR4

##### Channel Interrupt Node Pointer Register 4

**XSFR(9A<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>0</b>	<b>CHINP7</b>	<b>0</b>	<b>CHINP6</b>	<b>0</b>	<b>CHINP5</b>	<b>0</b>	<b>CHINP4</b>	<b>0</b>	<b>CHINP3</b>	<b>0</b>	<b>CHINP2</b>	<b>0</b>	<b>CHINP1</b>	<b>0</b>	<b>CHINP0</b>
r	rw	r	rw	r	rw	r	rw	r	rw	r	rw	r	rw	r	rw

Field	Bits	Type	Description
<b>CHINP4, CHINP5, CHINP6, CHINP7</b>	[1:0], [5:4], [9:8], [13:12]	rw	<b>Interrupt Node Pointer for Channel x</b> This bit field selects which service request output indicates a channel event interrupt of channel x. 00 <sub>B</sub> Service request output SR0 is selected. 01 <sub>B</sub> Service request output SR1 is selected. 10 <sub>B</sub> Service request output SR2 is selected. 11 <sub>B</sub> Service request output SR3 is selected.
<b>0</b>	[3:2], [7:6], [11:10], [15:14]	r	<b>Reserved</b> returns 0 if read; should be written with 0;

### CHINPR8

#### Channel Interrupt Node Pointer Register 8

**XSFR(9C<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>0</b>	<b>CHINP11</b>	<b>0</b>	<b>CHINP10</b>	<b>0</b>	<b>CHINP9</b>	<b>0</b>	<b>CHINP8</b>								
r	rw	r	rw	r	rw	r	rw					r			rw

Field	Bits	Type	Description
<b>CHINP8, CHINP9, CHINP10, CHINP11</b>	[1:0], [5:4], [9:8], [13:12]	rw	<b>Interrupt Node Pointer for Channel x</b> This bit field selects which service request output indicates a channel event interrupt of channel x. 00 <sub>B</sub> Service request output SR0 is selected. 01 <sub>B</sub> Service request output SR1 is selected. 10 <sub>B</sub> Service request output SR2 is selected. 11 <sub>B</sub> Service request output SR3 is selected.
<b>0</b>	[3:2], [7:6], [11:10], [15:14]	r	<b>Reserved</b> returns 0 if read; should be written with 0;

## CHINPR12

### Channel Interrupt Node Pointer Register 12

**XSFR(9E<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>0</b>	<b>CHINP15</b>	<b>0</b>	<b>CHINP14</b>	<b>0</b>	<b>CHINP13</b>	<b>0</b>	<b>CHINP12</b>	<b>0</b>	<b>CHINP11</b>	<b>0</b>	<b>CHINP10</b>	<b>0</b>	<b>CHINP9</b>	<b>0</b>	<b>CHINP8</b>
r	rw	r	rw	r	rw	r	rw	r	rw	r	rw	r	rw	r	rw

Field	Bits	Type	Description
<b>CHINP12, CHINP13, CHINP14, CHINP15</b>	[1:0], [5:4], [9:8], [13:12]	rw	<b>Interrupt Node Pointer for Channel x</b> This bit field selects which service request output indicates a channel event interrupt of channel x. 00 <sub>B</sub> Service request output SR0 is selected. 01 <sub>B</sub> Service request output SR1 is selected. 10 <sub>B</sub> Service request output SR2 is selected. 11 <sub>B</sub> Service request output SR3 is selected.
<b>0</b>	[3:2], [7:6], [11:10], [15:14]	r	<b>Reserved</b> returns 0 if read; should be written with 0;

### 18.2.13.7 Alias Register

The alias register contains bits to change a requested channel number from CH0 and CH1 to another channel number, see also [Section 18.2.12.3](#). The programmed alias channel number is replacing the internally requested number for analog input multiplexer (of the converter). The internally requested channel number is taken into account for all other internal actions and the synchronization request.

#### ALR0

##### Alias Register 0

**XSFR(1C<sub>H</sub>)**

**Reset Value: 0100<sub>H</sub>**

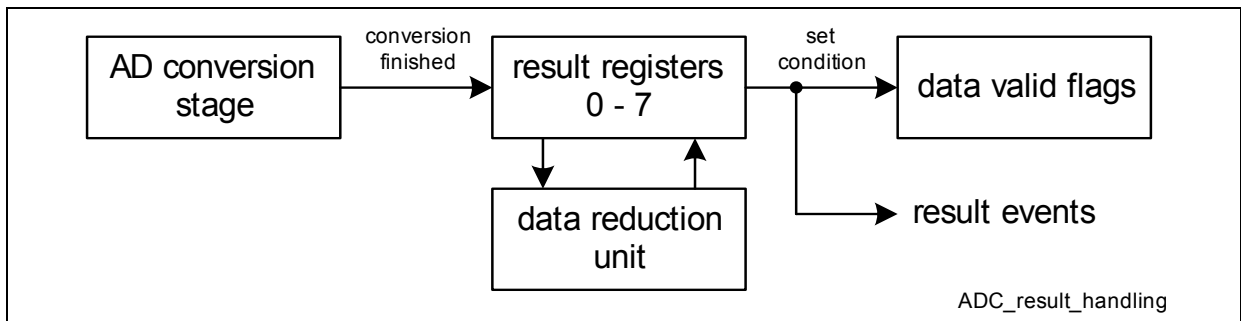
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0			ALIAS1					0			ALIAS0				
r			rw					r			rw				

Field	Bits	Type	Description
ALIAS0	[4:0]	rw	<b>Alias Value for CH0 Conversion Requests</b> The channel indicated in this bit field is converted instead of channel CH0. The conversion is done with the settings defined for channel CH0.
ALIAS1	[12:8]	rw	<b>Alias Value for CH1 Conversion Requests</b> The channel indicated in this bit field is converted instead of channel CH1. The conversion is done with the settings defined for channel CH1.
0	[7:5], [15:13]	r	<b>Reserved</b> returns 0 if read; should be written with 0;

### 18.2.14 Conversion Result Handling

The result generation part handles the:

- Storage of the conversion results (see [Section 18.2.14.1](#))
- Wait-for-read mode (see [Section 18.2.14.2](#))
- Result event interrupts (see [Section 18.2.14.3](#))
- Result FIFO buffer (see [Section 18.2.14.4](#))
- Data reduction or anti-aliasing filtering (see [Section 18.2.14.5](#))



**Figure 18-17 Conversion Result Handling**

#### 18.2.14.1 Storage of Conversion Results

For each analog input channel, the associated channel control register **CHCTR<sub>x</sub>** ( $x = 0 - 15$ ) [Section 18.2.13.1](#) contains a pointer bit field (RESRSEL) defining the result register to store the conversion result of this channel. This structure allows the user to direct conversion results of different channels to one or more result registers. Depending on the application needs (data reduction, auto-scan, alias feature, result FIFO, etc.), the user can distribute the conversion results to minimize CPU load or to be more tolerant against interrupt latency.

An individual data valid flag **VFR.VF<sub>x</sub>** for each result register indicates that “new” valid data has been stored in the corresponding result register and can be read out.

Due to different result handling mechanisms, the conversion result can be represented in different ways:

- **Data reduction filter disabled:**  
 The conversion result is maximum 10 bits wide with the MSB of the conversion result being always at bit position 11 and the remaining LSBs filled with 0.  
 The data valid flag is set and a result event occurs each time a new conversion result is stored in the result register.  
 It is possible to share a result register among several analog input channels.
- **Data reduction filter enabled:**  
 The conversion result is maximum 10 bits wide with the MSB of the conversion result being always at bit position 11 and the remaining LSBs filled with 0. The additional bits [13:12] show the MSBs of the data accumulation.  
 The data valid flag is set and a result event occurs each time a data reduction



sequence is finished and the final result is available in the result register.

The channel number is not included in the result register read view.

In order to support a wait-for-read and FIFO buffer features, the valid flag has to be cleared automatically when SW does a read access or the result is transferred into another FIFO element (if result FIFO buffering is enabled).

This behavior is contradictory to debugging requirements. For debugging, it has to be possible to introduce read or write commands into the normal program flow, e.g. to monitor conversion results. If a debugger reads out a result register, it would change the status of the conversion result from valid = “new” (not yet read out) to “old” (already read out). This would have an undesired impact on the application.

Therefore, the read views with “V” deliver the same value as the read views without “V”, but without clearing the valid bit. As a result, a debugger using read views with “V” can monitor the conversion results without influencing their status for the application.

The application requirements for results with enabled or disabled data reduction filter being different and debugger accesses can occur, four different scenarios with different result register read views are supported. The four read views refer to the same result register contents, but show a different behavior according to the address that has been read:

- Standard read view **RESRx (x = 0-7)**:  
The data reduction filter has to be disabled, the channel number is included to identify which channel has been converted, and a read action clears the corresponding valid bit. This representation is compatible to the ADC result register in XC16x devices.
- Read view **RESRAx (x = 0-7)**:  
The data reduction filter can be enabled, the channel number is not included, and a read action clears the corresponding valid bit.
- Read view **RESRVx (x = 0-7)** for debugger:  
The data reduction filter has to be disabled, the channel number is included, but a read action does not clear the corresponding valid bit.
- Read view **RESRAVx (x = 0-7)** for debugger:  
The data reduction filter can be enabled, the channel number is not included, but a read action does not clear the corresponding valid bit.

### **18.2.14.2 Wait-for-Read Mode**

The wait-for-read mode is a feature of a result register allowing the CPU (or PEC) to treat each conversion result independently without the risk of data loss. Data loss could occur if the CPU does not read a conversion result from a result register before a new result overwrites the previous one.

Especially for auto-scan conversion sequences (or other sequences with “relaxed” timing requirements), the wait-for-read offers the possibility to request a conversion sequence according to an event (HW or SW), but to start a new conversion according to the CPU capability to read the formerly converted result.

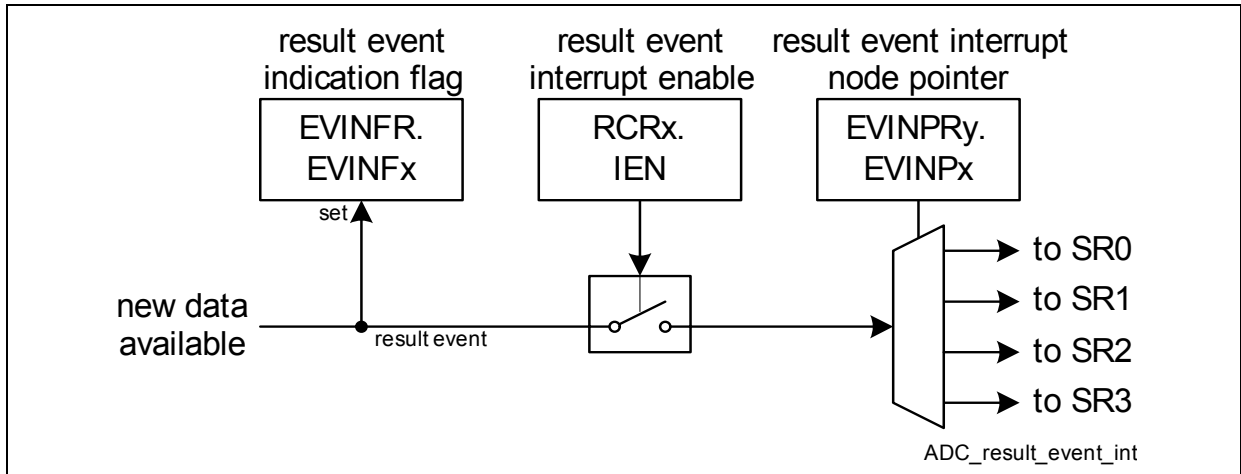
If wait-for-read mode is enabled for a result register by setting bit WFR in register **RCRx (x = 0 - 7)**, a request source does not generate a conversion request while the targeted result register contains valid data (indicated by the valid flag VFx = 1) or if a currently running conversion targets the same result register.

A new conversion request is generated only after the targeted result register has been read out.

If two request sources target the same result register with wait-for-read selected, a lower priority request started before the higher priority source has requested its conversion can not be interrupted by the higher priority request. If a higher priority request targets a different result register, the lower priority conversion can be cancelled and repeated afterwards.

### 18.2.14.3 Result Event Interrupts

A result event interrupt can be generated based on a result event according to the structure shown in **Figure 18-18**. If a result event is detected, it sets the corresponding indication flag in register **EVINFR**. 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 **EVINCR**.



**Figure 18-18 Result Event Interrupt Generation**

The service request output line SRx that is selected by the result event interrupt node pointer bit fields in registers **EVINPR8** or **EVINPR12** issues an interrupt each time the related result event is detected or the related bit position in register **EVINFR** is written with a 1.

The result events and the request source events share the same registers. The result events are located at the following bit positions in register **EVINFR**:

- Event 8: Result event of result register 0.
- Event 9: Result event of result register 1.
- ...
- Event 15: Result event of result register 7.

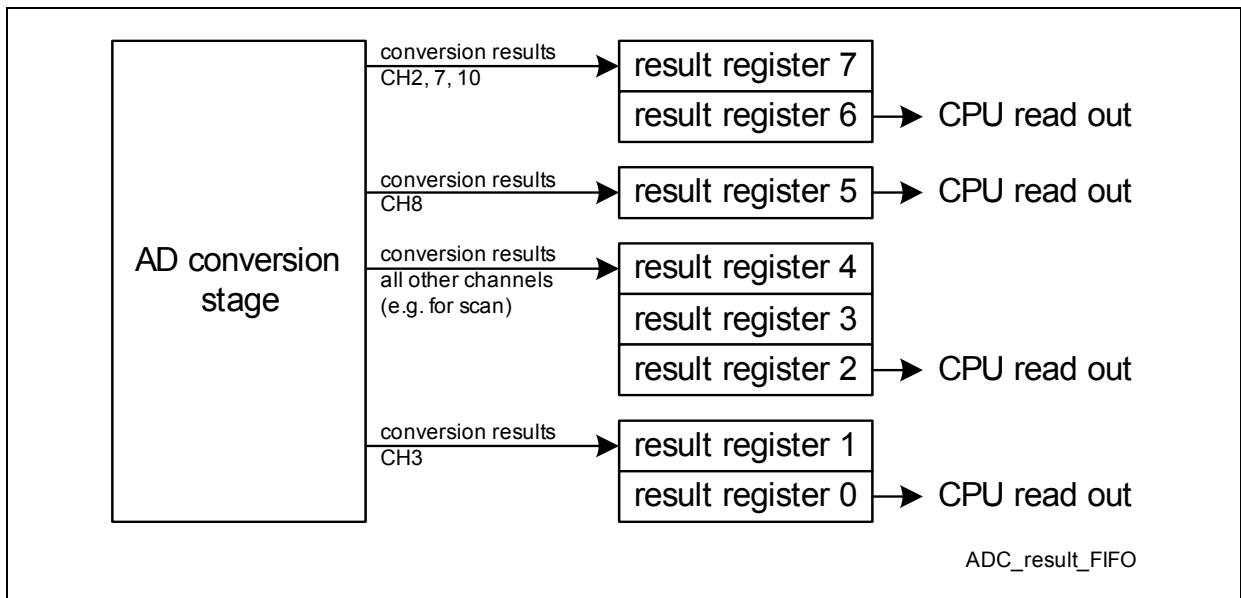
#### 18.2.14.4 Result FIFO Buffer

If a result register is not used as direct target for a conversion result, it can be concatenated with other result registers of the same ADC kernel to form a result FIFO buffer (first-in-first-out buffer mechanism). This allows to store measurement results and to read them out later with a “relaxed” CPU access timing. It is possible to set up more than one FIFO buffer structure with the available result registers.

A FIFO structure can be built by at least two “neighbor” result registers with the indices  $x$  and  $z = x + 1$ , where result register  $z$  represents the input and result register  $x$  represents the output of the FIFO buffer. The conversion result has to be delivered by the converter stage to the FIFO input, whereas the buffered data has to be read out from the FIFO output.

The FIFO buffer function is enabled by setting bit FEN in registers **RCRx** ( $x = 0 - 7$ ), except for RCRz.

In the example shown in **Figure 18-19**, the result registers have been configured to form two FIFO buffers with two buffer stages (result registers 0/1 and 6/7, respectively), one FIFO buffer with three buffer stages (result registers 2/3/4), whereas result register 5 is used as “normal” result register without additional FIFO buffer functionality.



**Figure 18-19 Result FIFO Buffers**

If more than two result neighbor registers are concatenated to a FIFO buffer (from result register  $z$  to result register  $x$ , with  $z > x$ ), the one with the highest index ( $z$ ) is always the input and the one with the lowest index ( $x$ ) is always the output. All intermediate result registers  $y$  ( $x < y < z$ ) are used as intermediate FIFO stages without data input or data output functionality.

Result register features for each FIFO buffer:

**Analog to Digital Converter**

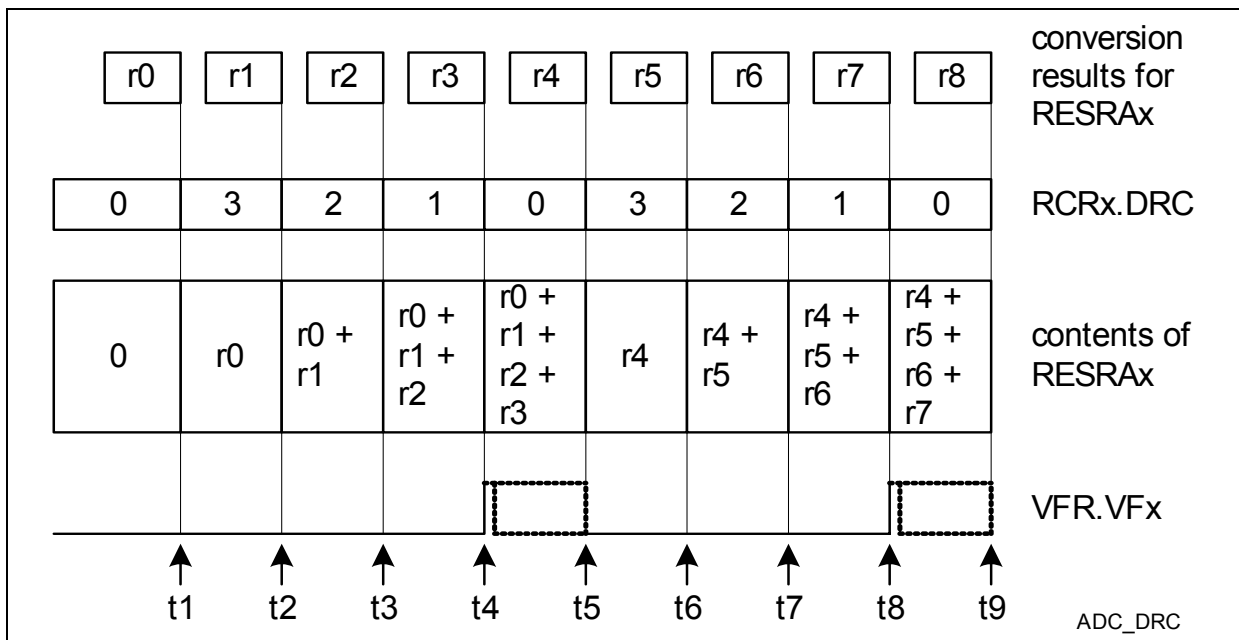
- **Result register z (FIFO buffer input, FEN = 0):**  
This result register can be enabled for data reduction. The wait-for-read mode is supported to avoid data loss if the FIFO is full. Result event interrupt generation is not supported. Must not be read at a read view modifying the valid bit.
- **Result register y (intermediate buffer stage, FEN = 1):**  
This/these result register(s) must not be enabled neither for wait-for-read mode, nor for data reduction. Result event interrupt generation is not supported. Must not be read at a read view modifying the valid bit, nor be the target of a conversion result.
- **Result register x (FIFO buffer output, FEN = 1):**  
This result register can be enabled for result event interrupt generation to inform the CPU that new data can be read out from this register location. Data reduction and wait-for-read are not supported and have to be disabled. Must not be the target of a conversion result.  
If enabled, a result interrupt is generated for each data word in the FIFO.

### 18.2.14.5 Data Reduction Filter

The data reduction filter can be used as digital filter for anti-aliasing or decimation purposes. It can accumulate a maximum of 4 conversion results to generate a final result.

Each result register can be individually enabled for data reduction. The feature is controlled by bit field DRCTR in registers **RCRx** ( $x = 0 - 7$ ). The actual status is given by bit field DRC (data reduction counter) in the same register.

Conversions delivering results to other result registers do not influence the data reduction filter of result register x. As a consequence, other channels can be converted between two conversions targeting result register x.



**Figure 18-20 Data Reduction Filter**

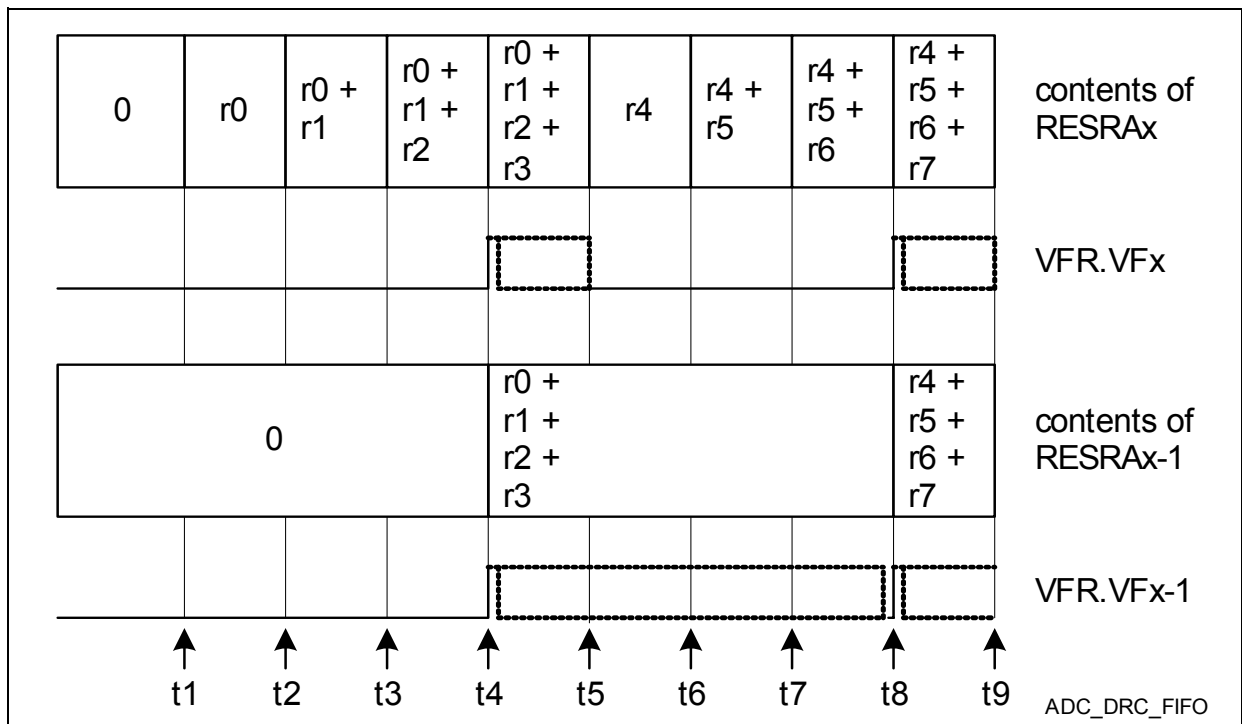
In the example given in **Figure 18-20**, a data reduction sequence of 4 accumulated conversion results is shown. The data reduction is based on three rules:

- Each time bit field DRC is 0 and a conversion targeting result register x is completed ( $t_1$ ,  $t_5$ ,  $t_9$ ), the contents of bit field RCR $x$ .DRCTR is loaded into bit field DRC and the conversion result is stored in result register x (i.e. the result accumulation begins).
- Each time bit field DRC is not 0 and a conversion targeting result register x is completed ( $t_2$ ,  $t_3$ ,  $t_4$  for the first final result and  $t_6$ ,  $t_7$ ,  $t_8$  for the next one), bit field DRC is decremented by 1 and the conversion result is added to the value already stored in result register x.
- Each time bit field DRC is 0 after decrementing or after loading it with RCR $x$ .DRCTR = 0 ( $t_4$  for the first final result and  $t_8$  for the next one), the valid bit for the result register x becomes set and a result register event occurs.

**Analog to Digital Converter**

The final result of a data reduction sequence has to be read out from result register x before the next data reduction sequence starts (interval between t4 and t5, or t8 and t9 respectively). With the read out of the final result from this register, the valid flag is automatically cleared.

If this interval is too short, it is recommended to associate a second result register z to result register x by enabling the result FIFO mechanism for result register x, see **Figure 18-21** ( $z = x + 1$ ). In this case, result register x is loaded with the final result elaborated by result register z when a data reduction sequence is finished. The final result has to be read out from result register x before the next data reduction sequence is finished (interval between t4 and t8).



**Figure 18-21 Data Reduction Filter with Result FIFO**

The data reduction counter **RCRx** ( $x = 0 - 7$ ).DRC can be cleared by SW by writing a 1 to bit position x in register **VFR**.

## 18.2.15 Conversion Result-Related Registers

### 18.2.15.1 Standard Views RESRx and RESRVx

These result registers deliver the conversion result and the related channel number.

The corresponding valid flag is cleared when register RESRx is read, whereas it is left unchanged when reading RESRVx.

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

**Result Register x**

**XSFR(40<sub>H</sub> + 2 \* x)**

**Reset Value: 0000<sub>H</sub>**

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

**Result Register x, View V**

**XSFR(60<sub>H</sub> + 2 \* x)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>CHNR</b>				<b>RESULT</b>											<b>0</b>
rh				rh											r

Field	Bits	Type	Description
<b>RESULT</b>	[11:2]	rh	<b>Conversion Result</b> This bit field contains the conversion result.
<b>CHNR</b>	[15:12]	rh	<b>Channel Number</b> This bit field contains the channel number of the latest register update. In case that the external multiplexer control is enabled, bits CHNR[3:1] are replaced by the multiplexer setting EMUX[2:0].
<b>0</b>	[1:0]	r	<b>Reserved</b> returns 0 if read; should be written with 0;



### 18.2.15.2 Data Reduction Read Views RESRAX and RESRAVx

These result registers deliver the accumulated conversion result and no channel number.

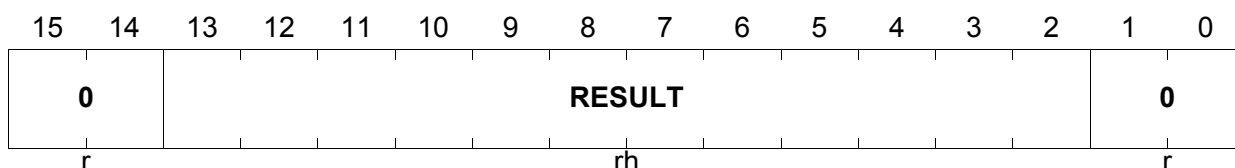
The corresponding valid flag is cleared when register RESRAX is read, whereas it is left unchanged when reading RESRAVx.

#### **RESRAX (x = 0-7)**

**Result Register x, View A**      XSFR(50<sub>H</sub> + 2 \* x)      **Reset Value: 0000<sub>H</sub>**

#### **RESRAVx (x = 0-7)**

**Result Register x, View AV**      XSFR(70<sub>H</sub> + 2 \* x)      **Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
<b>RESULT</b>	[13:2]	rh	<b>Conversion Result</b> This bit field contains the result of the data reduction filter.
<b>0</b>	[1:0], [15:14]	rh	<b>Reserved</b> returns 0 if read; should be written with 0;

### 18.2.15.3 Result Status Shadow Register

The result status shadow register contains the status information related to the latest result register (view without extension “V”) that has been read out. The update of the bit fields is done when a result register is read out.

*Note: The standard view of the result register RESRx shows only the 4-bit channel number of the last conversion. If the application requires the full 5-bit channel number, then it can be read out from the bit field RSSR.CHNR after a read-access to RESRx or RESRAx.*

#### **RSSR**

#### **Result Status Shadow Register**

**XSFR(82<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

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

Field	Bits	Type	Description
<b>CHNR</b>	[3:0]	rh	<b>Channel Number</b> This bit field indicates the channel number related to the latest result that has been read out.
<b>RRNR</b>	[14:12]	rh	<b>Result Register Number</b> This bit field indicates to which result register the information stored in CHNR belongs.
<b>0</b>	[11:4], 15	r	<b>Reserved</b> returns 0 if read; should be written with 0;

### 18.2.15.4 Valid Flag Register

The valid flag register contains the flags indicating that the corresponding result register contents are valid (valid = “new” = not read out).

#### VFR

#### Valid Flag Register

**XSFR(80<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

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

Field	Bits	Type	Description
<b>VFx</b> (x = 0 - 7)	x	rwh	<b>Valid Flag for Result Register x</b> This bit indicates that the contents of the result register x is valid. Writing a 0 has no effect, whereas writing a 1 clears the written bit position and bit field DRC in register <b>RCRx (x = 0 - 7)</b> . If a hardware event triggers the setting of a bit VFx and SW writes a 1 to the same bit position, the bit VFx is cleared (software overrules hardware). 0 <sub>B</sub> The result register x does not contain valid data. Either this register has been read out or no data has been moved to it. 1 <sub>B</sub> The result register x contains valid data that has not yet been read out.
<b>0</b>	[15:8]	r	<b>Reserved</b> returns 0 if read; should be written with 0;

### 18.2.15.5 Result Control Registers

The result control registers contain bits to control the behavior of the result registers and to monitor their status.

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

**Result Control Register x**                      **XSFR(B0<sub>H</sub> + x \* 2)**                      **Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		VF		0		DRC		0		WFR	FEN	IEN	0		DRCTR
r		rh		r		rh		r	rw	rw	rw	r		rw	

Field	Bits	Type	Description
<b>DRCTR</b>	[1:0]	rw	<b>Data Reduction Control</b> This bit field defines how many conversion results are accumulated for data reduction (see <a href="#">Section 18.2.14.5</a> ). It defines the reload value for bit field DRC. 00 <sub>B</sub> The data reduction filter is disabled. The reload value for DRC is 0, so no accumulation is done. 01 <sub>B</sub> The data reduction filter is enabled. The reload value for DRC is 1, so the accumulation is done over 2 conversions. 10 <sub>B</sub> The data reduction filter is enabled. The reload value for DRC is 2, so the accumulation is done over 3 conversions. 11 <sub>B</sub> The data reduction filter is enabled. The reload value for DRC is 3, so the accumulation is done over 4 conversions.
<b>IEN</b>	4	rw	<b>Interrupt Enable</b> This bit enables the result event interrupt if a result event is detected for result register x. 0 <sub>B</sub> The result event interrupt is disabled. 1 <sub>B</sub> The result event interrupt is enabled.

Field	Bits	Type	Description
<b>FEN</b>	5	rw	<b>FIFO Enable</b> This bit enables the FIFO functionality for result register x, see <a href="#">Section 18.2.14.4</a> . 0 <sub>B</sub> The FIFO functionality is disabled. Use this for the FIFO input register. 1 <sub>B</sub> The FIFO functionality is enabled. Use this for the other FIFO registers
<b>WFR</b>	6	rw	<b>Wait-for-Read Mode</b> This bit enables the wait-for-read mode for result register x. 0 <sub>B</sub> The wait-for-read mode is disabled. 1 <sub>B</sub> The wait-for-read mode is enabled.
<b>DRC</b>	[9:8]	rh	<b>Data Reduction Counter</b> This bit field indicates how many conversion results have still to be accumulated to generate the final result for data reduction. The valid flag is automatically set and a result event is generated when this bit field becomes 0 (by decrementing or by reload). Bit field RCRx.DRC can be cleared by SW by writing a 1 to bit position x in register <a href="#">VFR</a> . 00 <sub>B</sub> The final result is available in the result register. 01 <sub>B</sub> 1 more conversion result has to be added to obtain the final result in the result register. 10 <sub>B</sub> 2 more conversion results have to be added to obtain the final result in the result register. 11 <sub>B</sub> 3 more conversion results have to be added to obtain the final result in the result register.
<b>VF</b>	12	rh	<b>Valid Flag</b> This flag indicates that the contents of the result register x is valid. It is another view of the corresponding bit in register VFR. 0 <sub>B</sub> The result register x does not contain valid data. 1 <sub>B</sub> The result register x contains valid data.
<b>0</b>	[3:2], 7, [11:10], [15:13]	r	<b>Reserved</b> returns 0 if read; should be written with 0;

### 18.2.15.6 Event Indication Flag Register

The event indication flag register EVINFR monitors the detected request source events (flags EVINF0 - EVINF2) and result events (flags EVINF8 - EVINF15).

#### EVINFR

**Event Indication Flag Register**      **XSFR(A0<sub>H</sub>)**      **Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>EVIN F15</b>	<b>EVIN F14</b>	<b>EVIN F13</b>	<b>EVIN F12</b>	<b>EVIN F11</b>	<b>EVIN F10</b>	<b>EVIN F9</b>	<b>EVIN F8</b>			<b>0</b>			<b>EVIN F2</b>	<b>EVIN F1</b>	<b>EVIN F0</b>
rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh			r			rwh	rwh	rwh

Field	Bits	Type	Description
<b>EVINF<sub>x</sub></b> ( <b>x = 0 - 2</b> )	x	rwh	<b>Event Indication Flag for Request Source x</b> Flag EVINF <sub>x</sub> indicates that a request source event of request source x has been detected. Writing a 0 has no effect, whereas writing a 1 sets the written bit position and generates the corresponding interrupt request. 0 <sub>B</sub> An event of request source x has not yet been detected. 1 <sub>B</sub> An event of request source x has been detected.
<b>EVINF<sub>x</sub></b> ( <b>x = 8 - 15</b> )	x	rwh	<b>Event Indication Flag for Result Register x - 8</b> Flag EVINF <sub>x</sub> indicates that a result event of result register x-8 has been detected. Writing a 0 has no effect, whereas writing a 1 sets the written bit position and generates the corresponding interrupt request. 0 <sub>B</sub> An event of result register x-8 has not yet been detected. 1 <sub>B</sub> An event of result register x-8 has been detected.
<b>0</b>	<b>[7:3]</b>	r	<b>Reserved</b> returns 0 if read; should be written with 0;

### 18.2.15.7 Clear Event Indication Register

Writing a 1 to a bit position in the event indication clear register EVINCR clears the corresponding event indication flag EVINFx in register **EVINFR**. If a request source or result event is detected when the corresponding bit position is written with a 1, flag EVINFx is cleared.

#### EVINCR

**Event Indication Clear Register**

**XSFR(A2<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EVIN C15	EVIN C14	EVIN C13	EVIN C12	EVIN C11	EVIN C10	EVIN C9	EVIN C8			0			EVIN GF2	EVIN GF1	EVIN GF0
w	w	w	w	w	w	w	w			r			rwh	rwh	rwh

Field	Bits	Type	Description
<b>EVINGFx</b> (x = 0 - 2)	x	rwh	<b>Event Indication GF for Request Source x</b> 0 <sub>B</sub> Read: A service request output has not yet been activated due to an event of request source x. Write: No action. 1 <sub>B</sub> Read: A service request output has been activated due to an event of request source x. Write: Bits EVINFR.x and EVINGFx are cleared.
<b>EVINCx</b> (x = 8 - 15)	x	w	<b>Clear Event Indication Flag for Result Reg. x-8</b> 0 <sub>B</sub> No action. 1 <sub>B</sub> Bit EVINFR.x is cleared.
<b>0</b>	[7:3]	r	<b>Reserved</b> returns 0 if read; should be written with 0;

### 18.2.15.8 Event Interrupt Node Pointer Registers

The bit fields in these registers define the service request output SR[3:0] that is used to signal a request source or result event interrupt.

#### EVINPR0

##### Event Interrupt Node Pointer Register 0

**XSFR(A8<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0						EVINP2		0		EVINP1		0		EVINP0	
r						rw		r		rw		r		rw	

Field	Bits	Type	Description
<b>EVINP0, EVINP1, EVINP2</b>	[1:0], [5:4], [9:8]	rw	<b>Interrupt Node Pointer for Request Source x</b> This bit field selects which service request output indicates an event interrupt of request source x. 00 <sub>B</sub> Service request output SR0 is selected. 01 <sub>B</sub> Service request output SR1 is selected. 10 <sub>B</sub> Service request output SR2 is selected. 11 <sub>B</sub> Service request output SR3 is selected.
<b>0</b>	[3:2], [7:6], [15:10]	r	<b>Reserved</b> returns 0 if read; should be written with 0;

#### EVINPR8

##### Event Interrupt Node Pointer Register 8

**XSFR(AC<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		EVINP11		0		EVINP10		0		EVINP9		0		EVINP8	
r		rw		r		rw		r		rw		r		rw	



Field	Bits	Type	Description
<b>EVINP8, EVINP9, EVINP10, EVINP11</b>	[1:0], [5:4], [9:8], [13:12]	rw	<b>Interrupt Node Pointer for Result Event x-8</b> This bit field selects which service request output indicates an event interrupt of result register x-8. 00 <sub>B</sub> Service request output SR0 is selected. 01 <sub>B</sub> Service request output SR1 is selected. 10 <sub>B</sub> Service request output SR2 is selected. 11 <sub>B</sub> Service request output SR3 is selected.
<b>0</b>	[3:2], [7:6], [11:10], [15:14]	r	<b>Reserved</b> returns 0 if read; should be written with 0;

#### **EVINPR12**

##### **Event Interrupt Node Pointer Register 12**

**XSFR(AE<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>0</b>	<b>EVINP15</b>	<b>0</b>	<b>EVINP14</b>	<b>0</b>	<b>EVINP13</b>	<b>0</b>	<b>EVINP12</b>	<b>0</b>	<b>EVINP11</b>	<b>0</b>	<b>EVINP10</b>	<b>0</b>	<b>EVINP9</b>	<b>0</b>	<b>EVINP8</b>
r	rw	r	rw	r	rw	r	rw	r	rw	r	rw	r	rw	r	rw

Field	Bits	Type	Description
<b>EVINP12, EVINP13, EVINP14, EVINP15</b>	[1:0], [5:4], [9:8], [13:12]	rw	<b>Interrupt Node Pointer for Result Event x-8</b> This bit field selects which service request output indicates an event interrupt of result register x-8. 00 <sub>B</sub> Service request output SR0 is selected. 01 <sub>B</sub> Service request output SR1 is selected. 10 <sub>B</sub> Service request output SR2 is selected. 11 <sub>B</sub> Service request output SR3 is selected.
<b>0</b>	[3:2], [7:6], [11:10], [15:14]	r	<b>Reserved</b> returns 0 if read; should be written with 0;



### 18.2.17 External Multiplexer Control

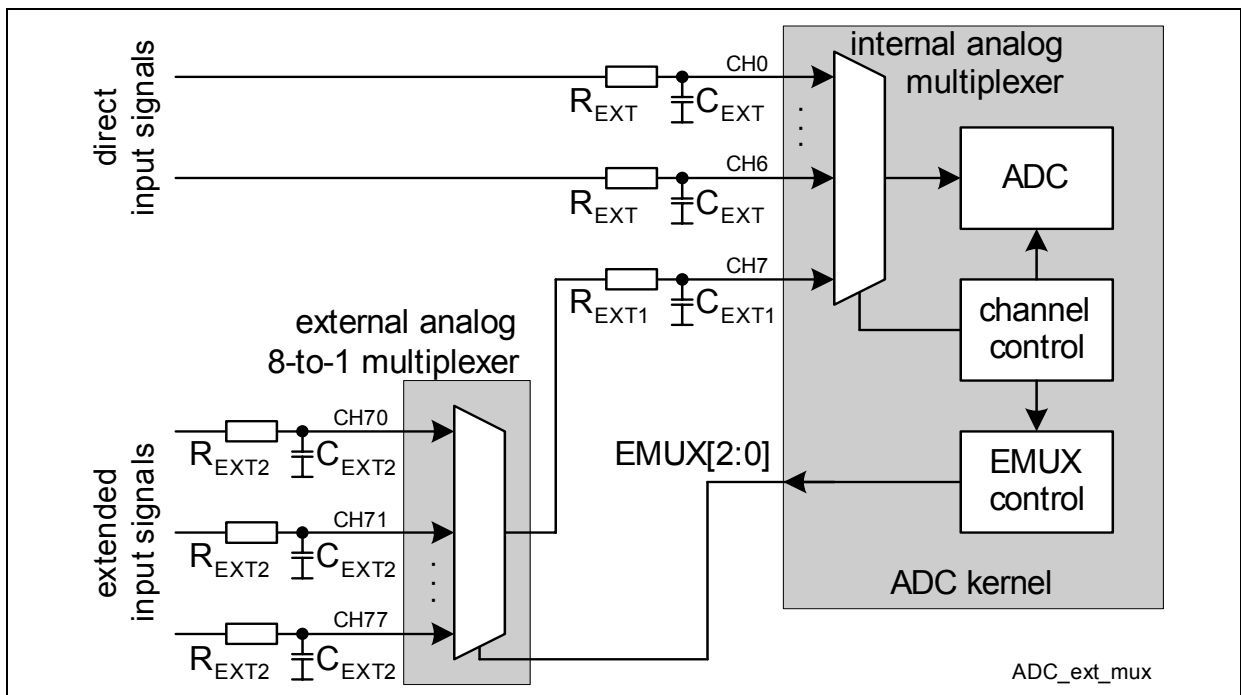
If an application requires more analog inputs channels than available on the XE16xyM, the ADC kernel supports an extension of analog channels by adding an external analog multiplexer. Three output signals EMUX[2:0] are delivered by each ADC kernel to control the settings of an external analog multiplexer. They can be used to extend the number of analog input channels by adding an external 1-out-of-8 multiplexer.

The external multiplexer control behavior is defined by the bits in registers **EMCTR** and **EMENR**.

The current setting of EMUX[2:0] is given by bit field EMUX. If another extended input channel should be converted, bit field SETEMUX has to be programmed to the desired value or the scan function has to be enabled. The SETEMUX value is automatically applied with the start of the next conversion of the related analog ADC input channel.

The external multiplexer support can be enabled for each of the input channels CH0 to CH15.

In the example shown in **Figure 18-23** and in the description below, the analog input CH7 has been extended, leading to additional analog inputs named CH70 to CH77.



**Figure 18-23 External Analog Multiplexer**

If the external multiplexer is located far from the ADC analog input, it is recommended to introduce an RC filter  $R_{EXT1}-C_{EXT1}$  directly at the analog input CH7 of the ADC. If needed for signal filtering, local RC filters  $R_{EXT2}-C_{EXT2}$  can be optionally added at the inputs of the external analog multiplexer.

If the external multiplexer is located close to the analog ADC input, the components  $R_{EXT1}$  and  $C_{EXT1}$  are not necessarily needed. In this case it is strongly recommended to

introduce RC filters ( $R_{EXT2}$ ,  $C_{EXT2}$ ) at the multiplexer inputs.

Please note that each RC filter limits the bandwidth of the analog input signal.

The RC filters used with an external multiplexer may lead to another impedance “seen” by the ADC analog input CH7 than for the other (direct) analog inputs. The adaptation of the sample phase length can be done by using a different input class with a different value for the sample phase extension. This value can be adapted to execute conversions with an EMUX[2:0] setting that has changed a sufficiently long time before the conversion of CH7 starts. “A sufficiently long time before” signifies that signal transitions at the analog ADC input due to changing multiplexer setting are finished and the input signal is stable enough.

After changing the EMUX[2:0] setting of the external multiplexer, an additional settling time has to elapse before the switched analog signal is stable and can be measured. To compensate for this settling time, an alternative sample phase length (instead of the one given by the input class) is automatically applied for the first conversion of CH7 after EMUX[2:0] has changed. The alternative sample phase length can be programmed by bit field **EMCTR.EMSAMPLE**. If the first conversion of CH7 after the EMUX[2:0] setting has changed is aborted due to a higher priority request, the repeated conversion of CH7 also uses the value of EMSAMPLE. The settling time is considered to be finished after the complete conversion of CH7.

The external multiplexer control block supports different modes, programmable by the bits in register **EMENR**:

- **SW control** without any HW control (EMUXEN = 0):  
 The automatic control of the external multiplexer setting and of the sampling time is disabled. Bit field EMUX is permanently updated with the value of SETEMUX. The changes of EMUX are related to write actions to SETEMUX and not to conversion timing. The setting of EMSAMPLE is not taken into account. It is recommended to write the start value of the first scan sequence to SETEMUX while EMUXEN = 0.
- **HW control without scan** (EMUXEN = 1, SCANEN = 0):  
 The update of EMUX with the value of SETEMUX happens with each conversion start of the channel selected by EMUXCHNR. For the first conversion with a new EMUX value, the setting of EMSAMPLE is applied.
- **HW control with single-input scan** (EMUXEN = 1, SCANEN = 1, TROEN = 0):  
 The update of EMUX with a new value happens after each conversion of the channel selected by EMUXCHNR. For each update, EMUX is automatically decremented by 1. If EMUX = 0, it is reloaded with the value of SETEMUX for the next update. For each conversion of the selected channel, the setting of EMSAMPLE is applied.  
 With this setting, an autoscan sequence requesting the conversion of the channel defined by EMUXCHNR leads to one conversion of the channel connected to the external multiplexer (trigger option disabled). As a result, for each completed auto scan sequence, another EMUX setting is applied.  
 Assuming inputs 1, 2, 70, 71, and 72 being selected for scan, the following sequence will be executed: 1, 2, 72, 1, 2, 71, 1, 2, 70, 1, 2, 72, 1, 2, 71, 1, 2, 70, 1, 2, 72,...

**Analog to Digital Converter**

- **HW control with multi-input scan** (EMUXEN = 1, SCANEN = 1, TROEN = 1):  
The update of EMUX with a new value happens after each conversion of the channel selected by EMUXCHNR. For each update, EMUX is automatically decremented by 1. If EMUX = 0, it is reloaded with the value of SETEMUX for the next update. For each conversion of the selected channel, the setting of EMSAMPLE is applied. With enabled trigger option, the external multiplexer control block triggers a new conversion request each time a conversion is started of the channel defined by EMUXCHNR while EMUX > 0.  
In a scan request source, the corresponding pending bit becomes set, whereas in a sequential request source, the content of the backup stage becomes valid (V bit of backup stage becomes set).  
With this setting, all external multiplexer inputs are scanned during a single autoscan sequence, starting with the channel indicated by SETEMUX (same update rate of all channels of this sequence).  
Assuming inputs 1, 2, 70, 71, and 72 being selected for scan, the following sequence will be executed: 1, 2, 72, 71, 70, 1, 2, 72, 71, 70, 1, 2, 72, 71, 70, 1, 2, 72,...

### 18.2.18 Synchronized Conversions for Parallel Sampling

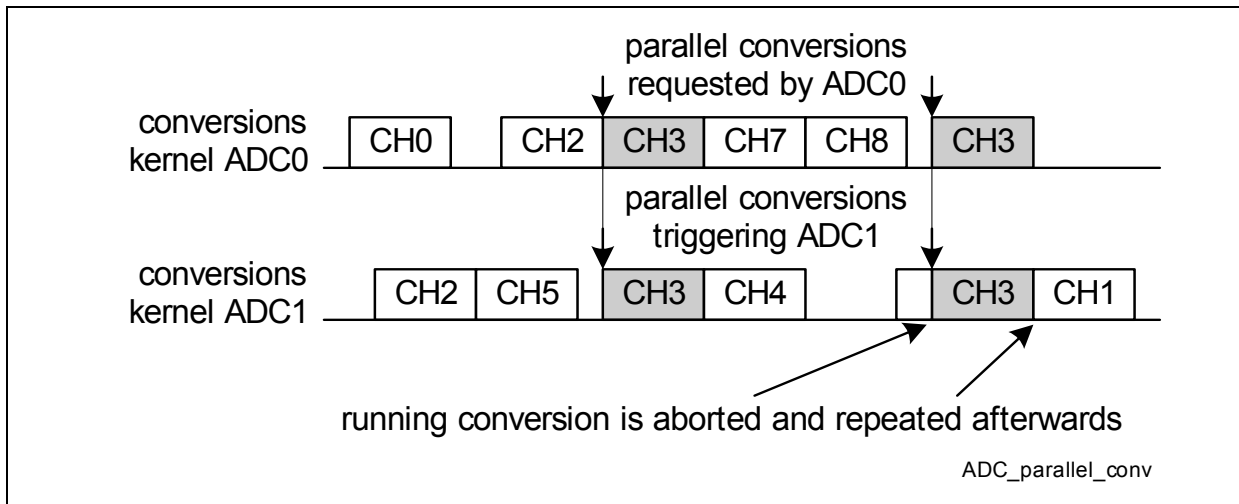
The independent ADC kernels implemented in the XE16xyM can be synchronized for simultaneous measurements of analog input channels. While no parallel conversion is requested, the kernels can work independently.

The synchronization mechanism for parallel conversions ensures that the sample phases of the related channel(s) start simultaneously. 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 (also several channels can be enabled for parallel conversions). In the example shown in **Figure 18-24**, input channels CH3 of the ADC kernels ADC0 and ADC1 are converted synchronously, whereas other input channels do not lead to parallel conversions.

This leads to the following structure:

- The **synchronization master** ADC kernel can request a conversion of an analog channel. If this channel is selected for a synchronized conversion, it is also requested in the connected slave ADC kernel(s).
- The **synchronization slave** ADC kernel reacts to incoming synchronized conversion requests from its master. While no incoming master requests are active, the slave kernel can convert its own requests.
- All ADC kernels in an ADC module being similar, each kernel can be set up to be a synchronization master or a synchronization slave (depending on the application needs, such as trigger capability of request sources).



**Figure 18-24 Parallel Conversions**

The master kernel and the slave kernel form a “conversion group” that allows parallel sampling:

- Kernels in the same conversion group can execute parallel conversions.
- A conversion group contains at least 1 ADC kernel.
- The conversion group contains exactly one synchronization master kernel that issues a parallel conversion request and defines the internal frequencies  $f_{\text{ADCI}}$  and  $f_{\text{ADCD}}$  and the channel number for a parallel conversion of the conversion group.
- The other kernels in the conversion group are synchronization slaves and have to be programmed with the same values of **GLOBCTR.DIVA**, **DIVD** and **ARBRND** as the synchronization master.
- If there is no need for parallel conversions, each kernel can be considered to form an own conversion group with only an ADC kernel as synchronization master, but without any synchronization slave.
- The channel number and the synchronization request are issued by the synchronization master to the kernels in the same synchronization group if a conversion is requested with **CHCTR<sub>x</sub> (x = 0 - 15).SYNC = 1** in the synchronization master kernel. The synchronization slave(s) can not issue synchronization requests.
- Once started, a parallel conversion can not be aborted.
- A parallel conversion request is always handled with highest priority and cancel-inject-repeat mode in the synchronization slave (see **Section 18.2.6.2**).
- Bit **GLOBCTR.ARB** has to be 0 for the synchronization slave(s).
- The wait-for-read mode is supported for the master kernel, whereas the setting is ignored in the slave kernel(s) (previous results may be overwritten).

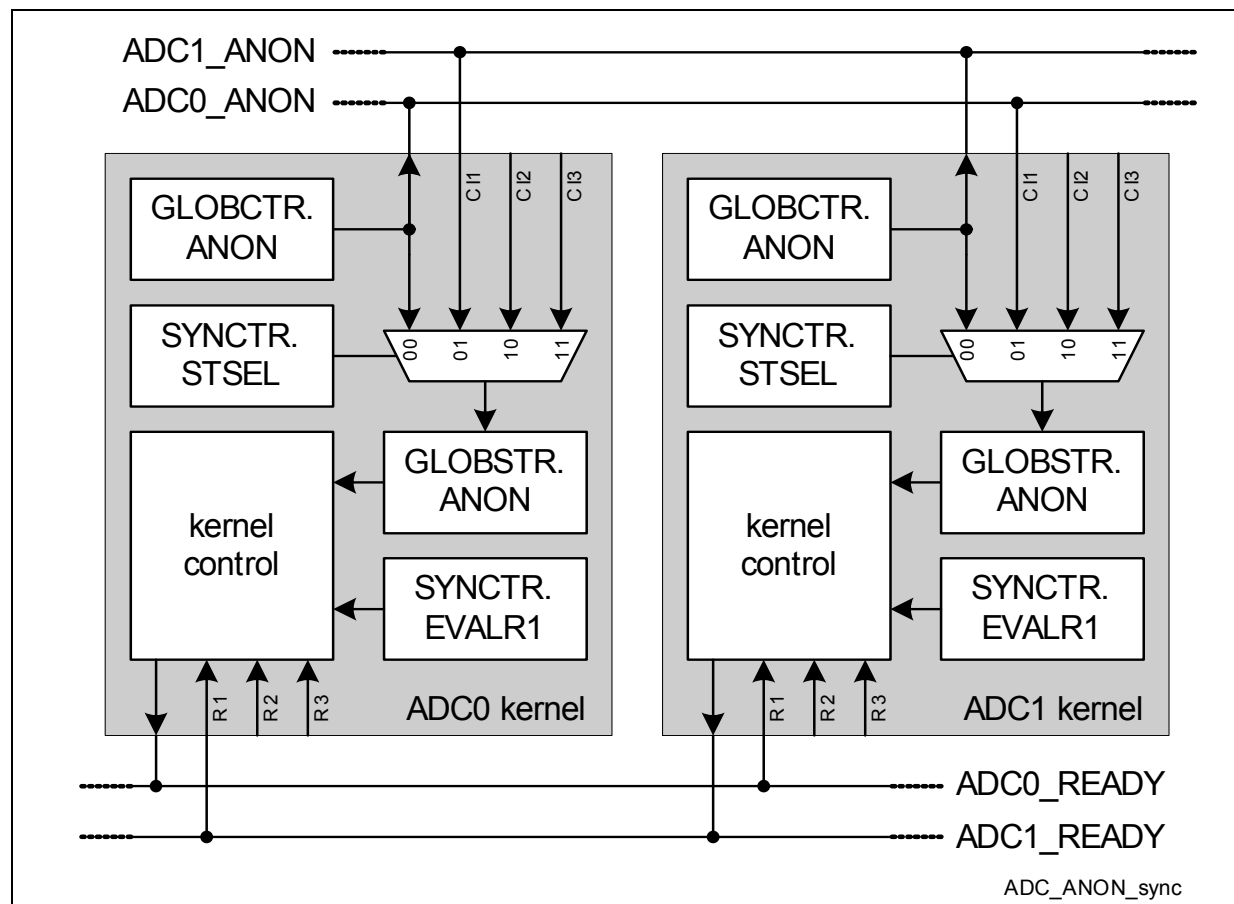
The synchronization request issuing mechanism of the master to the slave kernel(s) is based on bit field **GLOBSTR.ANON**. The information given by **GLOBCTR.ANON** is distributed by the synchronization master to the slave kernel(s) in the conversion group (the bit fields **SYNCTR.STSEL** of all kernels must be programmed in a way that all kernels refer to the same information). In addition to the ANON information, the master delivers the requested channel number to the slave (not explicitly shown in **Figure 18-25**).

The start of the converters of all kernels of the conversion group is based on signals indicating when a kernel is ready and can start the sample phase of a parallel conversion. Bit **SYNCTR.EVALR1** defines if a kernel has to wait for the other kernel(s) (to allow parallel conversions) or can start without waiting (no parallel conversions possible). To support parallel conversions, the ready signal of the kernel of a conversion group has to be considered.

The alias feature is independent of synchronized conversions. All kernels of a conversion group request the same channel number (defined by the master), but can convert analog signals from different inputs. The requested channel number can be redirected by its alias setting. For example, if the channel number requested in the conversion group is channel CH0, but for a kernel, an alternative reference is connected

to this input, the actually converted analog input can be changed to any value. This can be done by programming bit field ALIAS0 accordingly.

*Note: A parallel conversion in the slave ADC should not target a result register that is already used for data reduction of other channels.*



### Figure 18-25 Synchronization via ANON and Ready Signals

### Table 18-4 SYNCTR Setting for Kernel Synchronization

<b>Operating Mode</b>	<b>SYNCTR.EVALR1</b>	<b>SYNCTR.STSEL</b>
-----------------------	----------------------	---------------------

### ADC0 Kernel (values to be programmed to ADC0\_SYNCNTR)

no sync	0 <sub>B</sub>	00 <sub>B</sub>
master of ADC1	1 <sub>B</sub>	00 <sub>B</sub>
slave of ADC1	1 <sub>B</sub>	01 <sub>B</sub>

### ADC1 Kernel (values to be programmed to ADC1\_SYNCTR)

no sync	$0_B$	$00_B$
---------	-------	--------



**Table 18-4 SYNCTR Setting for Kernel Synchronization (cont'd)**

<b>Operating Mode</b>	<b>SYNCTR.EVALR1</b>	<b>SYNCTR.STSEL</b>
master of ADC0	1 <sub>B</sub>	00 <sub>B</sub>
slave of ADC0	1 <sub>B</sub>	01 <sub>B</sub>

### **18.2.19 Equidistant Sampling**

Each ADC kernel supports equidistant sampling of one (or more) analog input channels, e.g. for audio purposes or digital filters.

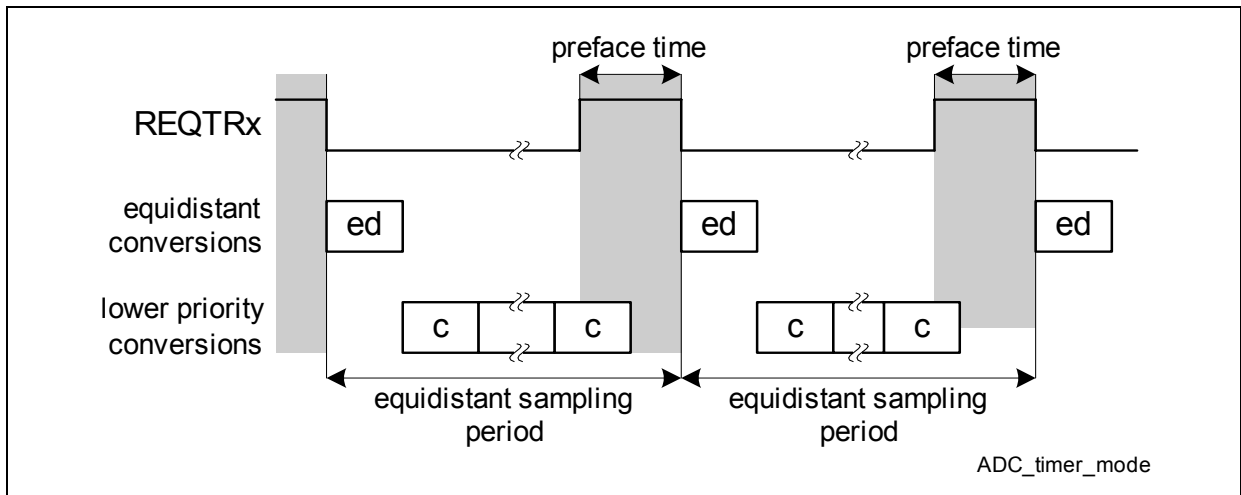
Therefore, each request source can be programmed to take part in the arbitration round and to win the arbitration (depending on the programmed priority levels), but without starting the conversion immediately. The exact start point of the conversion is given by a control signal (generated outside the ADC module, e.g. by a timer module) that is selected as trigger input REQTRx of request source x. Equidistant sampling is ensured if the REQTRx signal is generated synchronously to the arbiter timing, mainly for the arbiter. Each ADC kernel provides an output ARBCNT, that is activated once per arbitration round to count the arbiter cycles as timing base for the equidistant sampling by a timer located outside the ADC module.

A requested equidistant conversion can start its sampling phase if the converter is idle and the arbiter has decided which channel to convert. To ensure that the converter is idle, the arbiter decides which channel to convert (winner of the arbitration round), but it waits for the timer control signal to really start the measurement (preface time). If the request source selected for equidistant sampling has been programmed with the highest priority, no other request source can disturb the equidistant sampling.

The interpretation of the trigger signal REQTRx for equidistant sampling is enabled by selecting timer mode in the corresponding request source input register (RSIRx.TMEN = 1). The frequency of signal REQTRx defines the sampling rate and its high time defines the length of the preface time interval where the corresponding request source takes part in the arbitration. During the preface time, the currently running conversion can be finished. It has to be programmed to a value allowing the converter to become idle.

If signal ARBCNT is used as counting input signal for a timer, the arbiter has to be programmed to run permanently (GLOBCTR.ARBm = 0). If the timer has an independent time base, the arbiter can be stopped while no requests are pending. The preface time has to be longer than one arbitration round.

Depending on the request source requesting equidistant sampling, one or more channels can be converted one after the other. The order of the requested channels being fixed by the request source, the equidistant sampling is also supported for several 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 18-26 Timer Mode for Equidistant Sampling**

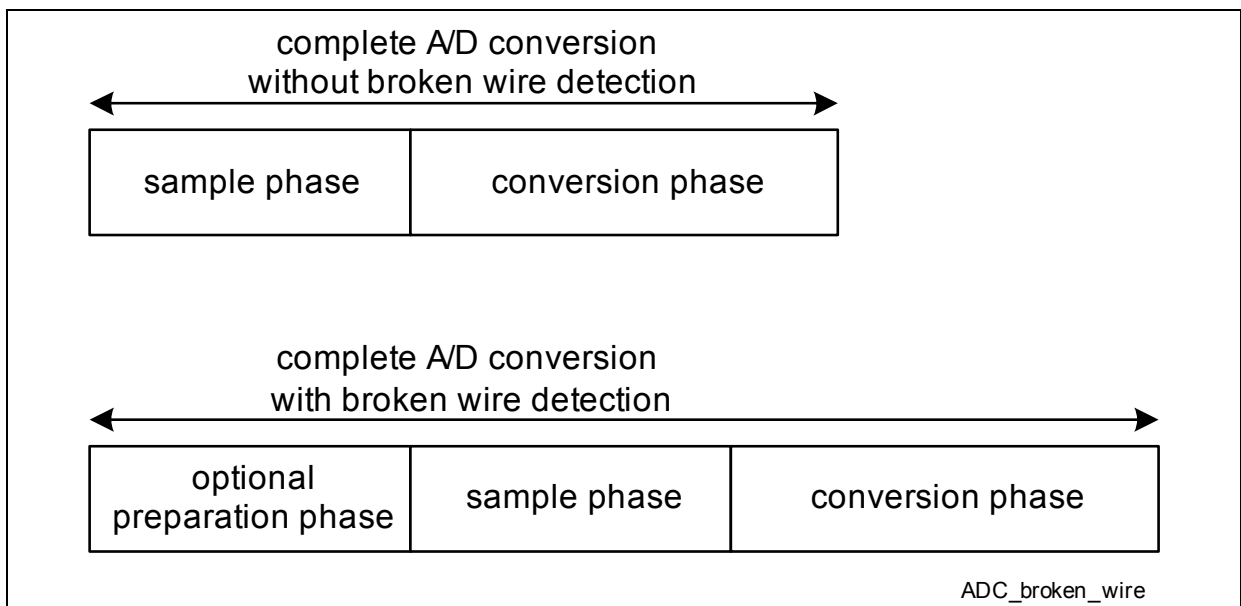
### 18.2.20 Broken Wire Detection

To support self-test in safety-critical applications, each ADC kernel provides a broken wire detection mechanism to check the connection of sensors or other voltage sources to the analog inputs of the ADC kernels.

This mechanism allows to prepare the capacitor field  $C_{AIN}$  before starting the sample phase and the conversion phase. A preparation phase is added to each conversion of an input channel with **BWDENR**.ENx = 1 (the broken wire detection can be individually enabled for each input channel CH0 to CH15).

An analog to digital conversion consists of the following phases:

- **Optional preparation phase:**  
 If a channel is enabled for broken wire detection, the capacitor field  $C_{AIN}$  is connected to the analog input CHx defined by **BWDCFGR**.CHP before the sample phase starts. The preparation phase length is identical to the sample phase length for this conversion.  
 If a channel is disabled for broken wire detection, the preparation phase is omitted (default setting).
- **Sample phase:**  
 During this phase, the capacitor field  $C_{AIN}$  is connected to one of the analog inputs CHx via an input multiplexer (see [Section 18.1.8.1](#)). The request sources and the arbiter define which analog input has the highest priority.
- **Conversion phase:**  
 During this phase, the capacitor field  $C_{AIN}$  is not connected to an analog input and the analog to digital conversion takes place. At the end of this phase,  $C_{AIN}$  is loaded to about  $V_{AREF}/2$ .



**Figure 18-27 Broken Wire Detection**

**Analog to Digital Converter**

The broken wire detection mechanism allows to apply a voltage outside the expected measurement value range of the connected sensor. If the actual digital conversion result is located outside the expected measurement range (e.g. by using limit checking) with enabled broken wire detection, a defective connection has been detected. It is recommended to ensure enough margin between the voltage applied during the preparation phase and the sensors output range to minimize the effects of parasitics and leakage.

Input channels CH16 ( $V_{AGND}$ ) and CH17 ( $V_{AREF}$ ) have been especially introduced to allow the selection of the maximum or the minimum voltage of the measurement range.

*Note: The length of the complete analog to digital conversion is increased by the length of the preparation phase if the broken wire detection is enabled. This influences the timing of conversion sequences.*

## 18.2.21 Additional Feature Registers

### 18.2.21.1 External Multiplexer Enable Register

The external multiplexer enable register defines which analog input channel is used to control the settings of an external analog multiplexer and defines its operating mode. It also contains bit MTM7 to control the multiplexer test mode for CH7.

#### EMENR

#### External Multiplexer Enable Register

XSFR(D6 <sub>H</sub> )											Reset Value: 0000 <sub>H</sub>				
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>MTM 7</b>				<b>0</b>				<b>EMU XEN</b>	<b>SCA NEN</b>	<b>TRO EN</b>	<b>0</b>			<b>EMUX CHNR</b>	
rw				r				rw	rw	rw	r			rw	

Field	Bits	Type	Description
<b>EMUXCHNR</b>	[3:0]	rw	<b>Channel Number for External Multiplexer</b> If external multiplexer control is enabled (EMUXEN = 1), this bit field defines the analog ADC input channel connected to the external analog multiplexer.
<b>TROEN</b>	5	rw	<b>Trigger Option Enable</b> This bit selects the scan mode behavior of the external multiplexer (if enabled). Description see <a href="#">Section 18.2.17</a> . 0 <sub>B</sub> Single-input scan is selected. The trigger option is disabled (no automatic trigger of more conversions of CHx). 1 <sub>B</sub> Multi-input scan is selected. The trigger option is enabled leading to an automatic scan through the externally connected multiplexer inputs by automatically triggering additional conversions of CHx until EMUX = 0.

Field	Bits	Type	Description
<b>SCANEN</b>	6	rw	<p><b>Scan Enable</b>  This bit enables/disables the automatic scan of the inputs of the external multiplexer for conversions of the channel selected by bit field EMUXCHNR (taken into account only if EMUXEN=1).</p> <p>0<sub>B</sub> The scan mode is disabled. Bit field EMUX is updated by bit field SETEMUX at the beginning of a conversion of the selected channel. If bit EMUX is changed, the value of EMSAMPLE is applied.</p> <p>1<sub>B</sub> The scan mode is enabled. Bit field EMUX is decremented by 1 for each conversion of the selected channel. After reaching 0, bit field EMUX is updated by bit field SETEMUX. The value of EMSAMPLE is always applied for the selected channel.</p> <p>It is recommended to write the start value of the first scan sequence to SETEMUX while EMUXEN=0.</p>
<b>EMUXEN</b>	7	rw	<p><b>External Multiplexer Control Enable</b>  This bit enables/disables the automatic control of the external multiplexer.</p> <p>0<sub>B</sub> The external multiplexer control by HW is disabled. Bit field EMUX is immediately updated under SW control by writing to SETEMUX. The settings of SCANEN and TROEN are ignored.</p> <p>1<sub>B</sub> The external multiplexer control is enabled. The update of EMUX is under HW control respecting the conversion timings.</p>
<b>MTM7</b>	15	rw	<p><b>Multiplexer Test Mode CH7</b>  This bit enables/disables multiplexer test mode for input CH7 (see <a href="#">Section 18.2.16</a>).</p> <p>0<sub>B</sub> The multiplexer test mode is disabled.</p> <p>1<sub>B</sub> The multiplexer test mode is enabled.</p>
<b>0</b>	4, [14:8]	r	<p><b>Reserved</b>  Read as 0; should be written with 0.</p>

### 18.2.21.2 External Multiplexer Control Register

The external multiplexer control register defines the settings of an external analog multiplexer and the alternative sample phase length.

#### EMCTR

#### External Multiplexer Control Register

XSFR(D0 <sub>H</sub> )										Reset Value: 0000 <sub>H</sub>					
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>EMSAMPLE</b>								<b>0</b>	<b>EMUX</b>		<b>0</b>	<b>SETEMUX</b>			
rw								r	rh		r	rw			

Field	Bits	Type	Description
<b>SETEMUX</b>	[2:0]	rw	<b>Setting of External Multiplexer</b> If the external multiplexer control is disabled, EMUX is loaded with the SETEMUX value. If enabled, the following two options are available: <b>Scan Mode disabled:</b> This bit field defines the input of the external multiplexer that will be selected for the next conversion of the channel selected by EMUXCHNR. Bit field EMUX will be updated by SETEMUX at the beginning of the next conversion of this channel. <b>Scan Mode enabled:</b> This bit field defines the start value of the scan of the external multiplexer inputs. The scan starts with the programmed input down to input 0. Bit field EMUX is updated by SETEMUX at the end of the conversion of this channel if EMUX = 0.



Field	Bits	Type	Description
EMUX	[6:4]	rh	<b>Current Setting for External Multiplexer</b> This bit field defines the input of the external multiplexer selected for conversion. Its value is available at the output lines EMUX[2:0]. If the external multiplexer control is disabled, EMUX is loaded with the SETEMUX value. If enabled, the following two options are available: <b>Scan Mode disabled:</b> This bit field becomes updated by SETEMUX at the beginning of the conversion of the channel selected by EMUXCHNR. <b>Scan Mode enabled:</b> This bit field is decremented by 1 at the end of the conversion of the channel selected by EMUXCHNR. After reaching 0, it is reloaded with the value of bit field SETEMUX.
EMSAMPLE	[15:8]	rw	<b>External Multiplexer Sampling Time</b> This bit field defines the alternative sample phase length in the case the external multiplexer setting has changed with the start of a conversion with enabled external multiplexer (the value given by the selected input class is not taken into account). A minimum sample phase of 2 analog clock cycles is extended by the programmed value. $\text{sample phase length} = (2 + \text{EMSAMPLE}) / f_{\text{ADC1}}$
0	3, 7	r	<b>Reserved</b> returns 0 if read; should be written with 0;

### 18.2.21.3 Synchronization Control Register

The synchronization control register contains bits controlling the synchronization between the kernels for parallel conversions. The programming of register SYNCTR in the kernels of the conversion group has to be done while the bit field **GLOBSTR**.ANON = 00<sub>B</sub> in the ADC kernels of the conversion group. Bit field ANON of the synchronization master can be set to 11<sub>B</sub> afterwards. It is recommended to avoid power saving modes (ANON = 01<sub>B</sub> or 10<sub>B</sub>) for parallel conversions.

The bits EVALRx are only taken into account if a synchronized, parallel conversion is requested by a master. This ensures that the conversions of the ADC kernels of the synchronization group are started at the same time for parallel sampling (although a kernel might be idle, the master and its connected slave have to wait for all of them being ready).

## SYNCTR

**Synchronization Control Register XSFR(1A<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0											0	EVALR1	0	STSEL	
r											rw	rw	r	rw	

Field	Bits	Type	Description
<b>STSEL</b>	[1:0]	rw	<b>Start Selection</b> This bit field controls the synchronization mechanism of the ADC kernel. 00 <sub>B</sub> The kernel is a synchronization master. The kernel's own bit field GLOBCTR.ANON is taken into account. 01 <sub>B</sub> The kernel is a synchronization slave. The control information at input CI1 is taken into account instead (see <a href="#">Figure 18-25</a> ). 10 <sub>B</sub> Reserved, do not use (kernel is switched off) 11 <sub>B</sub> Reserved, do not use (kernel is switched off)
<b>EVALR1</b>	4	rw	<b>Evaluate Ready Input R1</b> This bit defines if a kernel is considered to be part of the conversion group. Parallel conversions can only be started if the synchronization master and the slave of the conversion group indicate that they are ready to start a parallel conversion. 0 <sub>B</sub> The ready input R1 is not considered for the start of a parallel conversion of this conversion group. 1 <sub>B</sub> The ready input R1 is considered for the start of a parallel conversion of this conversion group.
<b>0</b>	[6:5]	r	<b>Reserved for Future Use</b> returns 0 if read; must be written with 0;
<b>0</b>	[3:2], [15:7]	r	<b>Reserved</b> returns 0 if read; should be written with 0;

### 18.2.21.4 Broken Wire Detection Enable Register

The broken wire detection enable register defines if a channel is enabled for broken wire detection by introducing an additional preparation phase to the sample phase. The channel number refers to the arbitration winner (can be directed to another input by the alias feature).

#### **BWDENR**

#### **Broken Wire Detection Enable Register**

**XSFR(C8<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>EN</b>	<b>EN</b>	<b>EN</b>	<b>EN</b>	<b>EN</b>	<b>EN</b>	<b>EN</b>	<b>EN</b>	<b>EN</b>	<b>EN</b>	<b>EN</b>	<b>EN</b>	<b>EN</b>	<b>EN</b>	<b>EN</b>	<b>EN</b>
<b>15</b>	<b>14</b>	<b>13</b>	<b>12</b>	<b>11</b>	<b>10</b>	<b>9</b>	<b>8</b>	<b>7</b>	<b>6</b>	<b>5</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>1</b>	<b>0</b>
rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW

Field	Bits	Type	Description
<b>ENx</b> <b>(x=0-15)</b>	x	rW	<b>Broken Wire Detection Enable for Channel CHx</b> This bit defines if the broken wire detection is enabled for CHx. 0 <sub>B</sub> The broken wire detection is disabled. 1 <sub>B</sub> The broken wire detection is enabled.

### 18.2.21.5 Broken Wire Detection Configuration Register

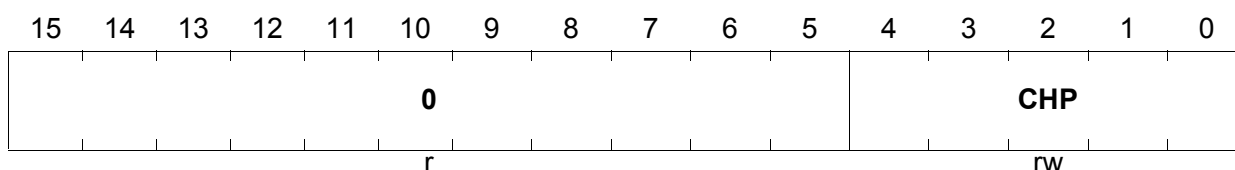
The broken wire detection configuration register defines which channel number is used for the additional preparation phase.

#### **BWDCFGR**

#### **Broken Wire Detection Configuration Register**

**XSFR(CA<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
<b>CHP</b>	[4:0]	rw	<b>Channel Number for Preparation Phase</b> This bit field defines which input channel is used for the preparation phase for the broken wire detection.
<b>0</b>	[15:5]	r	<b>Reserved</b> returns 0 if read; should be written with 0;

## 18.3 Implementation

This section describes the implementation of the ADC kernels in the XE16xyM device.

- Address map (see [Section 18.3.1](#))
- Interrupt control registers (see [Section 18.3.2](#))
- Analog connections of ADC0 (see [Section 18.3.3.1](#))
- Analog connections of ADC1 (see [Section 18.3.3.2](#))
- Digital connections of ADC0 (see [Section 18.3.4.1](#))
- Digital connections of ADC1 (see [Section 18.3.4.2](#))

### 18.3.1 Address Map

The ADC kernels ADC0 and ADC1 are available at the following base addresses. The exact register address is given by the offset of the register (given in [Table 18-2](#)) plus the kernel base address (given in [Table 18-5](#)) of the module.

**Table 18-5 Registers Address Space**

Module	Base Address	End Address	Note
ADC0	E000 <sub>H</sub>	E0FF <sub>H</sub>	
ADC1	E100 <sub>H</sub>	E1FF <sub>H</sub>	

**Table 18-6 Registers Overview**

Register Short Name	Register Long Name	Offset Address	Page Number
please refer to register table in <a href="#">Section 18.2.1</a>		H	

### 18.3.2 Interrupt Control Registers

The interrupt control registers are located in the SFR area. They are described in the general interrupt chapter.

**Table 18-7 ADC Interrupt Control Registers**

Short Name	Description
ADC_0IC	Interrupt Control Register for SR0 of ADC0
ADC_1IC	Interrupt Control Register for SR1 of ADC0
ADC_2IC	Interrupt Control Register for SR2 of ADC0
ADC_3IC	Interrupt Control Register for SR3 of ADC0

**Table 18-7    ADC Interrupt Control Registers (cont'd)**

<b>Short Name</b>	<b>Description</b>
<b>ADC_4IC</b>	Interrupt Control Register for SR0 of ADC1
<b>ADC_5IC</b>	Interrupt Control Register for SR1 of ADC1
<b>ADC_6IC</b>	Interrupt Control Register for SR2 of ADC1
<b>ADC_7IC</b>	Interrupt Control Register for SR3 of ADC1

### 18.3.3 Analog Connections

The input channels of both ADC kernels are distributed as follows:

- 16 channels (CH0 to CH15) of ADC0 are connected to P5
- 8 channels (CH0 to CH7) of ADC1 are connected to P15
- 4 channels (CH8 to CH11) of ADC1 are overlaid with ADC0 channels (CH8 to CH11) on P5
- CH16 of each ADC kernel is connected to its  $V_{AGND}$  input
- CH17 of each ADC kernel is connected to its  $V_{AREF}$  input

Each ADC kernel has its own reference input lines  $V_{AGND}$  and  $V_{AREF}$ . Depending on the package, these input lines can be available as independent pins for high pin count packages or can be combined for low pin count packages.

The respective voltage supply lines of both converters are connected together.

#### 18.3.3.1 Analog Connections of ADC0

The table below lists the analog connections of ADC0.

**Table 18-8 ADC0 Analog Connections in XE16xyM**

Signal	from/to Module	I/O to ADC0	Can be used to/as, connected to
Power supply and standard reference			
V <sub>DDPA</sub>	see pinning chapter	I	analog power supply
V <sub>SS</sub>		I	analog power supply
V <sub>AREF0</sub>		I	positive analog reference
V <sub>AGND</sub>		I	negative analog reference
Analog input channels			
CH0	P5.0	I	analog input channel 0
CH1	P5.1	I	analog input channel 1
CH2	P5.2	I	analog input channel 2
CH3	P5.3	I	analog input channel 3
CH4	P5.4	I	analog input channel 4
CH5	P5.5	I	analog input channel 5
CH6	P5.6	I	analog input channel 6
CH7	P5.7	I	analog input channel 7
CH8	P5.8	I	analog input channel 8 overlaid with ADC1 channel 8

**Table 18-8 ADC0 Analog Connections in XE16xyM (cont'd)**

Signal	from/to Module	I/O to ADC0	Can be used to/as, connected to
CH9	P5.9	I	analog input channel 9 overlaid with ADC1 channel 9
CH10	P5.10	I	analog input channel 10 overlaid with ADC1 channel 10
CH11	P5.11	I	analog input channel 11 overlaid with ADC1 channel 11
CH12	P5.12	I	analog input channel 12
CH13	P5.13	I	analog input channel 13
CH14	P5.14	I	analog input channel 14
CH15	P5.15	I	analog input channel 15
CH16	V <sub>AGND</sub>	I	analog input channel 16, internally connected to the V <sub>AGND</sub> input of ADC0
CH17	V <sub>AREF</sub>	I	analog input channel 17 internally connected to the V <sub>AREF</sub> input of ADC0
CH18 .. 31	n.c.	I	not available, do not request for conversion

### 18.3.3.2 Analog Connections of ADC1

The table below lists the analog connections of ADC1.

**Table 18-9 ADC1 Analog Connections in XE16xyM**

Signal	from/to Module	I/O to ADC1	Can be used to/as, connected to
Power supply and standard reference			
V <sub>DDPA</sub>	see pinning chapter	I	analog power supply
V <sub>SS</sub>		I	
V <sub>AREF1</sub>		I	positive analog reference
V <sub>AGND</sub>		I	negative analog reference
Analog input channels			
CH0	P15.0	I	analog input channel 0
CH1	P15.1	I	analog input channel 1
CH2	P15.2	I	analog input channel 2
CH3	P15.3	I	analog input channel 3



**Table 18-9    ADC1 Analog Connections in XE16xyM (cont'd)**

<b>Signal</b>	<b>from/to Module</b>	<b>I/O to ADC1</b>	<b>Can be used to/as, connected to</b>
CH4	P15.4	I	analog input channel 4
CH5	P15.5	I	analog input channel 5
CH6	P15.6	I	analog input channel 6
CH7	P15.7	I	analog input channel 7
CH8	P5.8	I	analog input channel 8 overlaid with ADC0 channel 8
CH9	P5.9	I	analog input channel 9 overlaid with ADC0 channel 9
CH10	P5.10	I	analog input channel 10 overlaid with ADC0 channel 10
CH11	P5.11	I	analog input channel 11 overlaid with ADC0 channel 11
CH16	V <sub>AGND</sub>	I	analog input channel 16, internally connected to the V <sub>AGND</sub> input of ADC1
CH17	V <sub>AREF</sub>	I	analog input channel 17, internally connected to the V <sub>AREF</sub> input of ADC1
CH12 .. 15, 18 .. 31	n.c.	I	not available, do not request for conversion

### 18.3.4 Digital Connections

The following table shows the digital connections of the ADC kernels with other modules or pins in the XE16xyM device.

The following sections refer to the inter-module connections, whereas the connections of the service request outputs SR[3:0] of each kernel to the interrupt control registers is given in [Section 18.3.2](#).

*Note: The functional inputs of the ADC that are marked “I(s)” are additionally synchronized to  $f_{SYS}$  before they can affect the module internal logic. The resulting delay of  $2/f_{SYS}$  and an uncertainty of  $1/f_{SYS}$  have to be taken into account for precise timing calculation. An edge of an input signal can only be correctly detected if the high phase and the low phase of the input signal are both longer than  $1/f_{SYS}$ .*

*The functional inputs of the ADC that are marked “I” are already considered as synchronous to  $f_{SYS}$ .*

#### 18.3.4.1 Digital Connections of ADC0

The table below lists the digital connections of ADC0.

**Table 18-10 ADC0 Digital Connections in XE16xyM**

Signal	from/to Module	I/O to ADC0	Can be used to/as, connected to
<b>Arbiter Timing</b>			
ARBCNT	CCU60_T12HRE, CCU60_T13HRE, CCU61_T12HRE, CCU61_T13HRE	O	time base for equidistant sampling for CCU60, CCU61
<b>External multiplexer control</b>			
EMUX[0]	P6.0	O	control of external analog multiplexer(s)
EMUX[1]	P6.1	O	control of external analog multiplexer(s)
EMUX[2]	P6.2	O	control of external analog multiplexer(s)
<b>Request Source 0</b>			
REQGT0A	CCU60_COUT63	I	CCU60
REQGT0B	CCU61_COUT63	I	CCU61
REQGT0C	0	I	request gating signal for source 0
REQGT0D	0	I	request gating signal for source 0
REQGT0E	ERU_PDOUT0	I (s)	ERU

**Table 18-10 ADC0 Digital Connections in XE16xyM (cont'd)**

<b>Signal</b>	<b>from/to Module</b>	<b>I/O to ADC0</b>	<b>Can be used to/as, connected to</b>
REQGT0F	ERU_PDOUT1	I (s)	ERU
REQGT0G	P6.0	I (s)	external pin
REQGT0H	CCU60_CC60	I (s)	CCU60
REQTR0A	CC2_CC16	I	CC2
REQTR0B	ERU_TOUT1	I	ERU
REQTR0C	CCU61_SR3	I	CCU61
REQTR0D	0	I	request trigger signal for source 0
REQTR0E	P6.1	I (s)	external pin
REQTR0F	P6.3	I (s)	external pin
REQTR0G	ADC0_REQGT0	I (s)	extend input selection for triggering by using gating inputs (with ENGT = 0X)
REQTR0H	ADC0_SR3	I (s)	service request output 3 of ADC0
REQTR0	-	O	selected trigger signal for source 0
REQGT0	ADC0_REQTR0G	O	selected gating signal for source 0

**Request Source 1**

REQGT1A	CCU60_COUT63	I	CCU60
REQGT1B	CCU61_COUT63	I	CCU61
REQGT1C	0	I	request gating signal for source 1
REQGT1D	0	I	request gating signal for source 1
REQGT1E	ERU_PDOUT0	I (s)	ERU
REQGT1F	ERU_PDOUT1	I (s)	ERU
REQGT1G	P6.0	I (s)	external pin
REQGT1H	CCU60_CC61	I (s)	CCU60
REQTR1A	CC2_CC17	I	CC2
REQTR1B	ERU_TOUT1	I	ERU
REQTR1C	CCU61_SR3	I	CCU61
REQTR1D	0	I	request trigger signal for source 1
REQTR1E	P6.1	I (s)	external pin
REQTR1F	P6.3	I (s)	external pin

**Table 18-10 ADC0 Digital Connections in XE16xyM (cont'd)**

<b>Signal</b>	<b>from/to Module</b>	<b>I/O to ADC0</b>	<b>Can be used to/as, connected to</b>
REQTR1G	ADC0_REQGT1	I (s)	extend input selection for triggering by using gating inputs (with ENGT = 0X)
REQTR1H	ADC0_SR3	I (s)	service request output 3 of ADC0
REQTR1	-	O	selected trigger signal for source 1
REQGT1	ADC0_REQTR1G	O	selected gating signal for source 1

**Request Source 2**

REQGT2A	CCU60_COUT63	I	CCU60
REQGT2B	CCU61_COUT63	I	CCU61
REQGT2C	0	I	request gating signal for source 2
REQGT2D	0	I	request gating signal for source 2
REQGT2E	ERU_PDOUT0	I (s)	ERU
REQGT2F	ERU_PDOUT1	I (s)	ERU
REQGT2G	P6.0	I (s)	external pin
REQGT2H	CCU60_CC62	I (s)	CCU60
REQTR2A	CC2_CC18	I	CC2
REQTR2B	ERU_TOUT1	I	ERU
REQTR2C	CCU61_SR3	I	CCU61
REQTR2D	0	I	request trigger signal for source 2
REQTR2E	P6.1	I (s)	external pin
REQTR2F	P6.3	I (s)	external pin
REQTR2G	ADC0_REQGT2	I (s)	extend input selection for triggering by using gating inputs (with ENGT = 0X)
REQTR2H	ADC0_SR3	I (s)	service request output 3 of ADC0
REQTR2	-	O	selected trigger signal for source 2
REQGT2	ADC0_REQTR2G	O	selected gating signal for source 2

**Service Request Outputs**

SR3	CCU60_CCPOS2C	O	CCU60 Hall input trigger
-----	---------------	---	--------------------------

### 18.3.4.2 Digital Connections of ADC1

The table below lists the digital connections of ADC1.

**Table 18-11 ADC1 Digital Connections in XE16xyM**

Signal	from/to Module	I/O to ADC1	Can be used to/as, connected to
<b>Arbiter Timing</b>			
ARBCNT	-	O	arbiter timing
<b>External multiplexer control</b>			
EMUX[0]	P7.2	O	control of external analog multiplexer(s)
EMUX[1]	P7.3	O	control of external analog multiplexer(s)
EMUX[2]	P7.4	O	control of external analog multiplexer(s)
<b>Request source 0</b>			
REQGT0A	CCU60_COUT63	I	CCU60
REQGT0B	CCU61_COUT63	I	CCU61
REQGT0C	0	I	request gating signal for source 0
REQGT0D	0	I	request gating signal for source 0
REQGT0E	ERU_PDOUT0	I (s)	ERU
REQGT0F	ERU_PDOUT1	I (s)	ERU
REQGT0G	P6.0	I (s)	external pin
REQGT0H	0	I (s)	request gating signal for source 0
REQTR0A	CC2_CC24	I	CC2
REQTR0B	ERU_TOUT1	I	ERU
REQTR0C	0	I	request trigger signal for source 0
REQTR0D	0	I	request trigger signal for source 0
REQTR0E	P6.1	I (s)	external pin
REQTR0F	P6.3	I (s)	external pin
REQTR0G	ADC1_REQGT0	I (s)	extend input selection for triggering by using gating inputs (with ENGTT = 0X)
REQTR0H	ADC1_SR3	I (s)	service request output 3 of ADC1
REQTR0	-	O	selected trigger signal for source 0
REQGT0	ADC1_REQTR0G	O	selected gating signal for source 0
<b>Request source 1</b>			

**Table 18-11 ADC1 Digital Connections in XE16xyM (cont'd)**

<b>Signal</b>	<b>from/to Module</b>	<b>I/O to ADC1</b>	<b>Can be used to/as, connected to</b>
REQGT1A	CCU60_COUT63	I	CCU60
REQGT1B	CCU61_COUT63	I	CCU61
REQGT1C	0	I	request gating signal for source 1
REQGT1D	0	I	request gating signal for source 1
REQGT1E	ERU_PDOUT0	I (s)	ERU
REQGT1F	ERU_PDOUT1	I (s)	ERU
REQGT1G	P6.0	I (s)	external pin
REQGT1H	0	I (s)	request gating signal for source 1
REQTR1A	CC2_CC25	I	CC2
REQTR1B	ERU_TOUT1	I	ERU
REQTR0C	0	I	request trigger signal for source 1
REQTR1D	0	I	request trigger signal for source 1
REQTR1E	P6.1	I (s)	external pin
REQTR1F	P6.3	I (s)	external pin
REQTR1G	ADC1_REQGT1	I (s)	extend input selection for triggering by using gating inputs (with ENG1 = 0X)
REQTR1H	ADC1_SR3	I (s)	service request output 3 of ADC1
REQTR1	-	O	selected trigger signal for source 1
REQGT1	ADC1_REQTR1G	O	selected gating signal for source 1

**Request source 2**

REQGT2A	CCU60_COUT63	I	CCU60
REQGT2B	CCU61_COUT63	I	CCU61
REQGT2C	0	I	request gating signal for source 2
REQGT2D	0	I	request gating signal for source 2
REQGT2E	ERU_PDOUT0	I (s)	ERU
REQGT2F	ERU_PDOUT1	I (s)	ERU
REQGT2G	P6.0	I (s)	external pin
REQGT2H	0	I (s)	request gating signal for source 2
REQTR2A	CC2_CC26	I	CC2
REQTR2B	ERU_TOUT1	I	ERU

**Table 18-11 ADC1 Digital Connections in XE16xyM (cont'd)**

<b>Signal</b>	<b>from/to Module</b>	<b>I/O to ADC1</b>	<b>Can be used to/as, connected to</b>
REQTR2C	0	I	request trigger signal for source 2
REQTR2D	0	I	request trigger signal for source 2
REQTR2E	P6.1	I (s)	external pin
REQTR2F	P6.3	I (s)	external pin
REQTR2G	ADC1_REQGT2	I (s)	extend input selection for triggering by using gating inputs (with ENGT = 0X)
REQTR2H	ADC1_SR3	I (s)	service request output 3 of ADC1
REQTR2	-	O	selected trigger signal for source 2
REQGT2	ADC1_REQTR2G	O	selected gating signal for source 2

**Service Request Outputs**

SR3	CCU61_CCPOS2C	O	CCU61 Hall input trigger
-----	---------------	---	--------------------------

## 19 Capture/Compare Unit

The XE16xyM provides a Capture/Compare (CAPCOM2) unit which provides 16 capture/compare channels, which interact with 2 timers. A CAPCOM channel can **capture** the contents of a timer on specific internal or external events, or it can **compare** a timer's contents with given values, and modify output signals in case of a match.

Data Registers	Control Registers	Interrupt Control
CC2_T7/T7REL	CC2_T78CON	CC2_T7IC
CC2_T8/T8REL		CC2_T8IC
CC16-CC19	CC2_M4	CC16IC-CC19IC
CC20-CC23	CC2_M5	CC20IC-CC23IC
CC24-CC27	CC2_M6	CC24IC-CC27IC
CC28-CC31	CC2_M7	CC28IC-CC31IC
	CC2_SEE	
	CC2_SEM	
	CC2_DRM	
	CC2_IOC	
CC2_OUT	CC2_ID	
	CC2_KSCCFG	
CC2_CC16...31	CAPCOM2 Capture/Compare Register 16...31	
CC2_CC16IC...31IC	CAPCOM2 Interrupt Control Register 16...31	
CC2_M4...7	CAPCOM2 Mode Control Register 4...7	
CC2_T78CON	CAPCOM2 Timer Control Register	
CC2_T7, T8	CAPCOM2 Timer Register	
CC2_T7/8REL	CAPCOM2 Timer Reload Register	
CC2_T7IC, T8IC	CAPCOM2 Timer x Interrupt Control Register	
CC2_SEE	CAPCOM2 Single Event Enable Register	
CC2_SEM	CAPCOM2 Single Event Mode Register	
CC2_DRM	CAPCOM2 Double-Register Compare Mode Register	
CC2_OUT	CAPCOM2 Output Register	
CC2_IOC	CAPCOM2 Input/Output Control Register	
CC2_KSCCFG	CAPCOM2 Kernel State Configuration Register	
CC2_ID	CAPCOM2 Module Identification Register	
	cc2_registers.vsd	

**Figure 19-1 SFRs Associated with the CAPCOM2 Unit**

The two timers of CAPCOM2 are named T7 and T8 and the 16 channels of CAPCOM2 are named CC16...31.



With this mechanism, the CAPCOM2 unit supports generation and control of timing sequences on up to 16 channels with a minimum of software intervention.

From the programmer's point of view, the term 'CAPCOM unit' refers to a set of registers which are associated with this peripheral (see also [Figure 19-1](#)), including the port pins that may be used for alternate input/output functions, and their control bits.

## **19.1 Functional Overview**

The CAPCOM2 unit is typically used to handle high speed IO tasks such as pulse and waveform generation, pulse width modulation, or recording of the time when a specific event occurs. It also supports the implementation of up to 16 software-controlled interrupt events.

The CAPCOM2 Unit consists of two 16-bit timers (T7/T8), each with its own reload register (TxREL), and a bank of sixteen dual-purpose 16-bit capture/compare registers (CCy).

The input clock for the CAPCOM timers is programmable to several prescaled values of the module input clock ( $f_{CC}$ ), or it can be derived from the overflow/underflow of timer T6. T7 may also operate in counter mode (from an external input), clocked by external events.

Each capture/compare register may be programmed individually for capture or compare operation, and each register may be allocated to either of the two timers. Each capture/compare register has one signal associated with it, which serves as an input signal for the capture operation or as an output signal for the compare operation.

The capture operation causes the current timer contents to be copied into the respective capture/compare register, triggered by an event (transition) on the associated input signal. This event also activates the associated interrupt request line.

The compare operation may cause an output signal transition on the associated output signal, when the allocated timer increments to the value stored in a capture/compare register. The compare match event also activates the associated interrupt request line. In Double-register compare mode a pair of registers controls one common output signal.

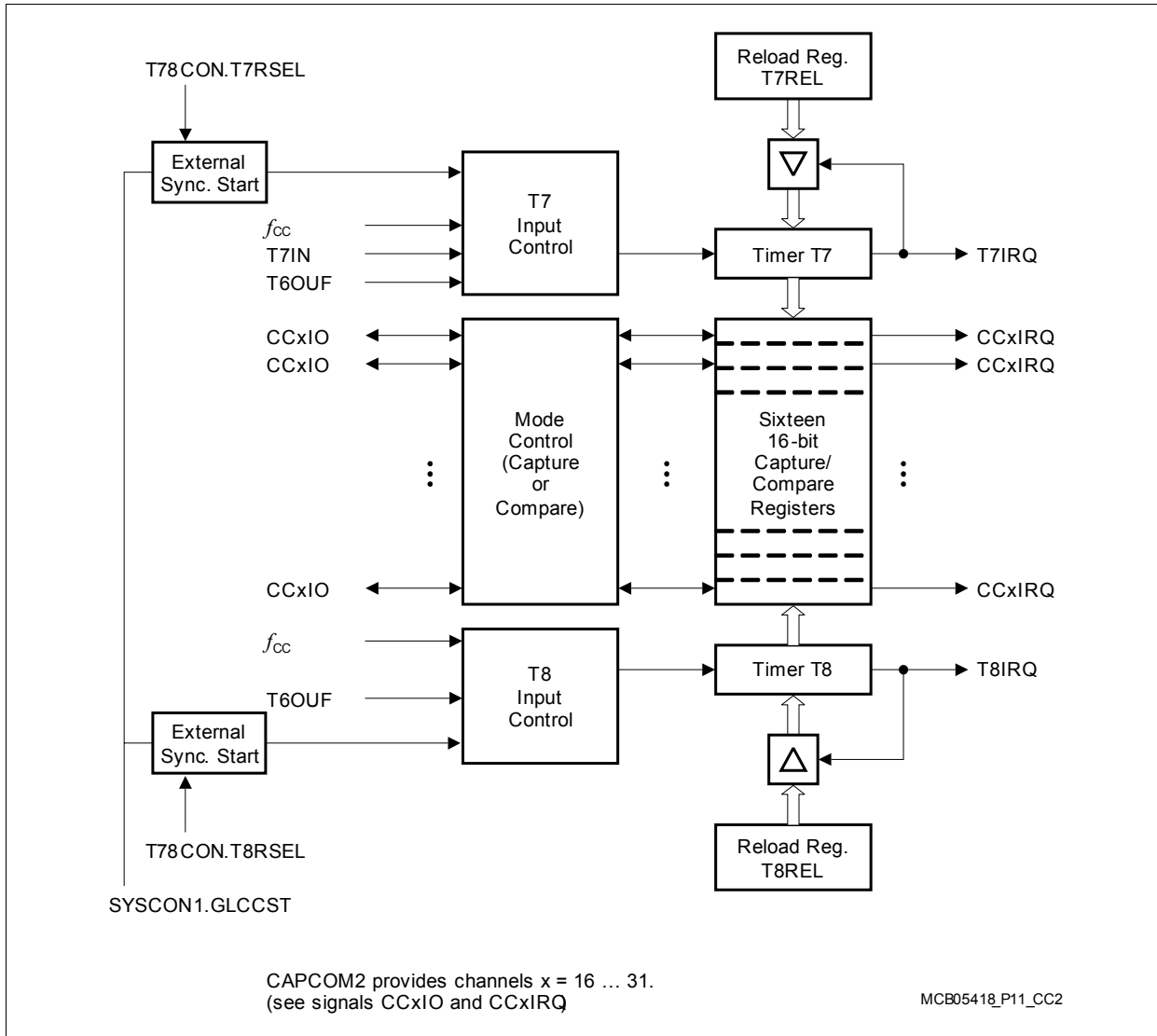
The compare output signals are available via a dedicated output register. The output path can be selected.

For the switching of the output signals two timing schemes (see [Section 19.1.10](#)) can be selected:

In **Staggered Mode** the output signals are switched consecutively in 8 steps, which distributes the switching steps over a certain time. In staggered mode, the maximum resolution is  $8 t_{CC}$ .

In **Non-Staggered Mode** the output signals are switched immediately at the same time. In non-staggered mode, the maximum resolution is  $1 t_{CC}$ .

[Figure 19-2](#) shows the basic structure of the CAPCOM2 unit.



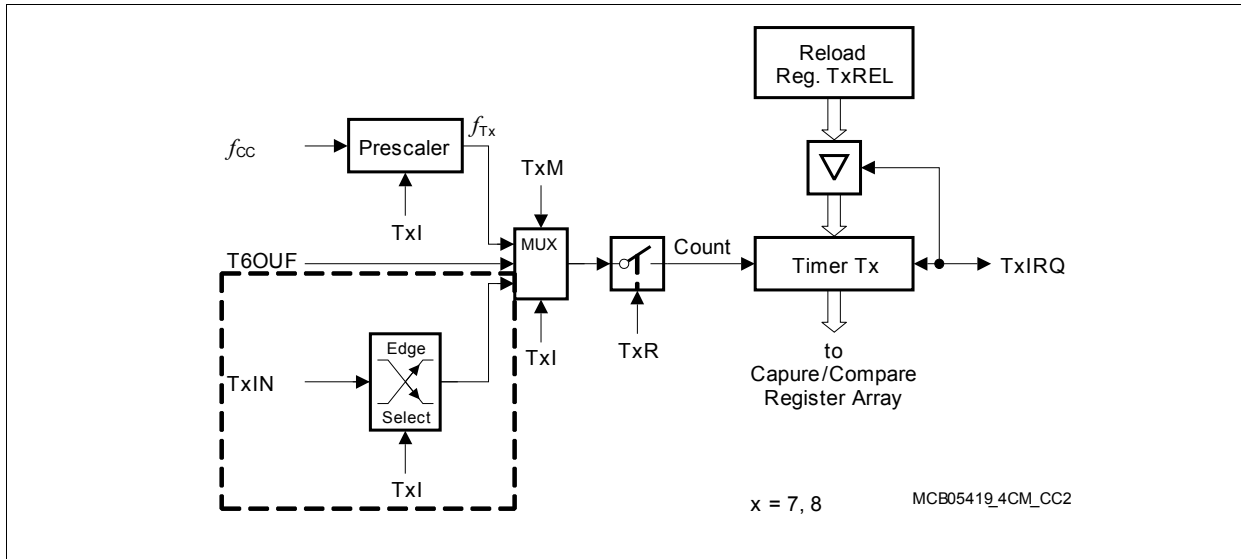
**Figure 19-2 CAPCOM2 Unit Block Diagram**

There is a possibility to start both timers T7 and T8 synchronously with the CAPCOM6 timers, by setting the bit SYS CON1.GLCCST.

### 19.1.1 The CAPCOM Timers

The primary use of the timers T7 and T8 is to provide two independent time bases for the capture/compare channels of each unit. The maximum resolution is  $8 t_{CC}$  in staggered mode, and  $1 t_{CC}$  in non-staggered mode.

The basic structure of the timers, illustrated in [Figure 19-3](#), is identical, except for the input pin (see mark).



**Figure 19-3 Block Diagram of a CAPCOM Timer**

The functions of the CAPCOM timers are controlled via the bit-addressable control register **CC2\_T78CON**. The high-byte of CC2\_T78CON controls T8, the low-byte of CC2\_T78CON controls T7. The control options are identical for all timers (except for external input).

In all modes, the timers are always counting upward. The current timer values are accessible for the CPU in the timer registers Tx, which are non bit-addressable registers. When the CPU writes to a register Tx in the state immediately before the respective timer increment or reload is to be performed, the CPU write operation has priority and the increment or reload is disabled to guarantee correct timer operation.

The timer run flags TxR allow the starting and stopping of the timers. The following description of the timer modes and operation always applies to the enabled state of the timers, i.e. the respective run flag is assumed to be set.

### Timer Mode

In Timer Mode ( $TxM = 0$ ), the input clock for a CAPCOM timer is derived from  $f_{CC}$ , divided by a programmable prescaler. Each timer has its own individual prescaler, controlled through the individual bitfields TxI in the timer control register CC2\_T78CON. The input frequency  $f_{Tx}$  for a timer Tx and its resolution  $r_{Tx}$  are determined by the following formulas:

Staggered Mode:

$$f_{Tx}[\text{MHz}] = \frac{f_{CC}[\text{MHz}]}{2^{(<TxI> + 3)}} \quad r_{Tx}[\mu\text{s}] = \frac{2^{(<TxI> + 3)}}{f_{CC}[\text{MHz}]} \quad (19.1)$$

Non-Staggered Mode:

$$f_{Tx}[MHz] = \frac{f_{CC}[MHz]}{2^{<TxI>}} \quad r_{Tx}[\mu s] = \frac{2^{<TxI>}}{f_{CC}[MHz]} \quad (19.2)$$

When a timer overflows from FFFF<sub>H</sub> to 0000<sub>H</sub>, it is reloaded with the value stored in its respective reload register TxREL. The reload value determines the period P<sub>Tx</sub> between two consecutive overflows of Tx as follows:

Staggered Mode:

$$P_{Tx}[\mu s] = \frac{(2^{16} - <TxREL>) \times 2^{(<TxI> + 3)}}{f_{CC}[MHz]} \quad (19.3)$$

Non-Staggered Mode:

$$P_{Tx}[\mu s] = \frac{(2^{16} - <TxREL>) \times 2^{<TxI>}}{f_{CC}[MHz]} \quad (19.4)$$

After a timer has been started by setting its run flag (TxR), the first increment will occur within the time interval which is defined by the selected timer resolution. All further increments occur exactly after the time defined by the timer resolution.

Examples for timer input frequencies, resolution and periods, which result from the selected prescaler option in TxI when using a 40 MHz clock, are listed in [Table 19-1](#) below. The numbers for the timer periods are based on a reload value of 0000<sub>H</sub>. Note that some numbers may be rounded.

**Table 19-1 Timer Tx Input Clock Selection for Timer Mode,  $f_{CC} = 40$  MHz**

<b>Txl</b>	<b>Prescaler</b>	<b>Input Frequency</b>	<b>Resolution</b>	<b>Period</b>
<b>Staggered Mode</b>				
000 <sub>B</sub>	8	5 MHz	200 ns	13.11 ms
001 <sub>B</sub>	16	2.5 MHz	400 ns	26.21 ms
010 <sub>B</sub>	32	1.25 MHz	800 ns	52.43 ms
011 <sub>B</sub>	64	625 kHz	1.6 $\mu$ s	104.86 ms
100 <sub>B</sub>	128	312.5 kHz	3.2 $\mu$ s	209.72 ms
101 <sub>B</sub>	256	156.25 kHz	6.4 $\mu$ s	419.43 ms
110 <sub>B</sub>	512	78.125 kHz	12.8 $\mu$ s	838.86 ms
111 <sub>B</sub>	1024	39.0625 kHz	25.6 $\mu$ s	1677.72 ms
<b>Non-Staggered Mode</b>				
000 <sub>B</sub>	1	40 MHz	25 ns	1.6384 ms
001 <sub>B</sub>	2	20 MHz	50 ns	3.2768 ms
010 <sub>B</sub>	4	10 MHz	100 ns	6.5536 ms
011 <sub>B</sub>	8	5 MHz	200 ns	13.11 ms
100 <sub>B</sub>	16	2.5 MHz	400 ns	26.21 ms
101 <sub>B</sub>	32	1.25 MHz	800 ns	52.43 ms
110 <sub>B</sub>	64	625 kHz	1.6 $\mu$ s	104.86 ms
111 <sub>B</sub>	128	312.5 kHz	3.2 $\mu$ s	209.72 ms

### Counter Mode

In Counter Mode ( $TxM = 1$ ), the input clock of a CAPCOM timer is either derived from an associated external input pin, T7IN, or from the over-/underflows of GPT timer T6.

Using an external signal connected to pin TxIN as a counting signal is only possible for timer T7. The only counter option for timer T8 is using the over-/underflows of the GPT timer T6 (selected by  $Txl = 000_B$ ).

Bitfields T7I are used to select either a positive, a negative, or both a positive and a negative transition of the external signal at pin T7IN to trigger an increment of timer T7. Please note that certain criteria must be met for the external signal and the port pin programming for this mode in order to operate properly. These conditions are detailed in [Chapter 19.1.11](#).

### **Timer Overflow and Reload**

When a CAPCOM timer contains the value  $FFFF_H$  at the time a new count trigger occurs, a timer interrupt request is generated, and the timer is loaded with the contents of its associated reload register TxREL. The timer then resumes incrementing with the next count trigger starting from the reloaded value.

The reload registers TxREL are not bit-addressable. After reset, they contain the value  $0000_H$ .

#### **19.1.2 Timer Interrupt**

Upon a timer overflow the corresponding timer interrupt request flag TxIR for the respective timer will be set. This flag can be used to generate an interrupt or trigger a PEC service request, when enabled by the respective interrupt enable bit TxIE.

Each timer has its own bit-addressable interrupt control register CC2\_TxIC and its own interrupt vector. The organization of the interrupt control registers TxIC is identical with the other interrupt control registers.

### 19.1.3 Capture/Compare Channels

The 16-bit capture/compare registers **CC2\_CCy (y=16-31)** are used as data registers for capture or compare operations with respect to timers T7 and T8. The capture/compare registers are not bit-addressable.

The functions of the 16 capture/compare registers of a unit are controlled by 4 bit-addressable 16-bit mode control registers, named **CC2\_M4 ... CC2\_M7**, which are all organized identically. Each register contains the bits for mode selection and timer allocation for four capture/compare registers.

Each of the registers CCy may be individually programmed for capture mode or for one of 4 different compare modes, and may be allocated individually to one of the two timers of the CAPCOM unit.

A special double-register compare mode combines two registers to act on one common output signal. When capture or compare operations are disabled for one of the CCy registers, it may be used for general purpose variable storage.

**Table 19-2 Selection of Capture Modes and Compare Modes**

Mode	MODy	Selected Operating Mode
Disabled	000 <sub>B</sub>	<b>Disable Capture and Compare Modes</b> The respective CAPCOM register may be used for general variable storage.
Capture	001 <sub>B</sub>	<b>Capture on Positive Transition (Rising Edge)</b> at Pin CCyIO
	010 <sub>B</sub>	<b>Capture on Negative Transition (Falling Edge)</b> at Pin CCyIO
	011 <sub>B</sub>	<b>Capture on Positive and Negative Transition (Both Edges)</b> at Pin CCyIO
Compare	100 <sub>B</sub>	<b>Compare Mode 0:</b> Interrupt Only Several interrupts per timer period. Can enable double-register compare mode for Bank2 registers.
	101 <sub>B</sub>	<b>Compare Mode 1:</b> Toggle Output Pin on each Match Several compare events per timer period. Can enable double-register compare mode for Bank1 registers.
	110 <sub>B</sub>	<b>Compare Mode 2:</b> Interrupt Only Only one interrupt per timer period.
	111 <sub>B</sub>	<b>Compare Mode 3:</b> Set Output Pin on each Match Reset output pin on each timer overflow; only one interrupt per timer period.

## Capture/Compare Unit

The detailed discussion of the capture and compare modes is valid for all the capture/compare channels, so registers, bits and pins are only referenced by a placeholder.

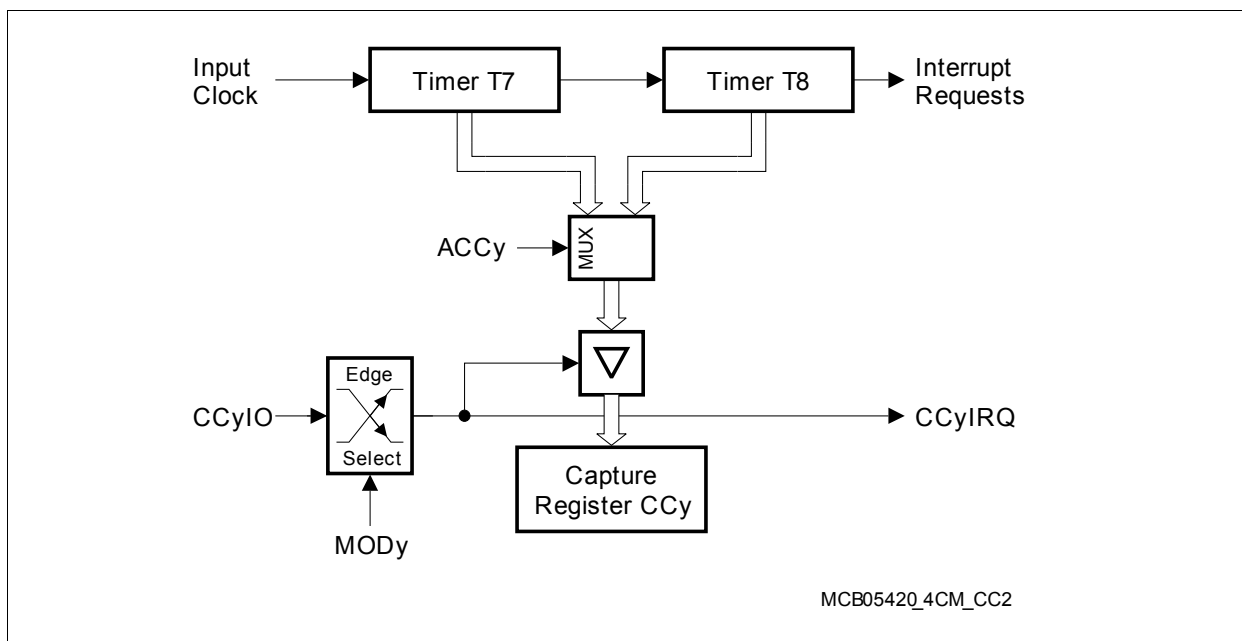
### 19.1.4 Capture Mode

In Capture Mode, the current contents of a CAPCOM timer are copied (captured) into the respective capture/compare register in response to an external event. This is used, for example, to record the time at which an external event has occurred, or to measure the distance between two external events in timer increments.

The event to cause a capture of a timer's contents can be programmed to be either the positive, the negative, or both the positive and the negative transition of the external signal connected to the input pin. This triggering transition is selected by bitfield MODy in the respective mode control register. When the selected external signal transition occurs, the selected timer's contents is copied into the capture/compare register and the respective interrupt request line CCyIRQ is activated. This can cause an interrupt or PEC service request, when enabled.

*Note: A capture input can be used as an additional external interrupt input. The capture operation can be disregarded in this case.*

Either the contents of timer T7 or T8 can be captured, selected by the timer allocation control bit ACCy in the respective mode control register.



**Figure 19-4 Capture Mode Block Diagram**

For capture operation, the respective pin must be programmed for input. To ensure that a transition of the input signal is recognized correctly, its level must be held high or low



for a minimum number of module clock cycles before it changes. This information can be found in [Section 19.1.11](#).

### **19.1.5 Compare Modes**

The compare modes allow triggering of events (interrupts and/or output signal transitions) or generation of pulse trains with minimum software overhead. In all compare modes, the 16-bit value stored in a capture/compare register CCy (in the following also referred to as 'compare value') is continuously compared with the contents of the allocated timer (T7 or T8). If the current timer contents match the compare value, the interrupt request line associated with register CCy is activated and, depending on the compare mode, an output signal can be generated at the corresponding output pin CCyIO.

Four different compare modes are available, which can be selected individually for each of the capture/compare registers by bitfield MODy in the respective mode control register. Modes 0 and 2 do not influence the output signals. In the following, each mode is described in detail.

In addition to these 'single-register' modes, a 'double-register' compare mode enables two registers to operate on the same pin. This feature can further reduce software overhead, as two different compare values can be programmed to control a sequence of transitions for a signal. See [Section 19.1.6](#) for details for this operation.

In all Compare Modes, the comparator performs an 'equal to' comparison. This means, a match is only detected when the timer contents are equal to the contents of a compare register. In addition, the comparator is only enabled in the clock cycle directly after the timer was incremented by hardware. This is done to prevent repeated matches if the timer does not operate with the highest possible input clock (either in timer or counter mode). In this case, the timer contents would remain at the same value for several or up to thousands of cycles. This operation has the side-effect, that software modifications of the timer contents will have no effect regarding the comparator. If a timer is set by software to the same value stored in one of the compare registers, no match will be detected. If a compare register is set to a value smaller than the current timer contents, no action will take place.

For the exact operation of the port output function, please see [Section 19.1.8](#).

When two or more compare registers are programmed to the same compare value<sup>1)</sup>, their corresponding interrupt request flags will be set and the selected output signals will be generated after the allocated timer is incremented to this compare value. Further compare events on the same compare value are disabled<sup>2)</sup> until the timer is incremented

1) In staggered mode these interrupts and output signals are generated sequentially (see [Section 19.1.10](#)).

2) Even if more compare cycles are executed before the timer increments (lower timer frequency) a given compare value only results in one single compare event.

again or written to by software. After a reset, compare events for register CCy will only become enabled, if the allocated timer has been incremented or written to by software and one of the compare modes described in the following has been selected for this register.

#### **19.1.5.1 Compare Mode 0**

This is an interrupt-only mode which can be used for software timing purposes. In this mode, the interrupt request line CCyIRQ is activated each time a match is detected between the contents of the compare register CCy and the allocated timer. A match means, the contents of the timer are equal to ('=') the contents of the compare register. Several of these compare events are possible within a single timer period, if the compare value in register CCy is updated during the timer period. The corresponding port signal CCyIO is not affected by compare events in this mode and can be used as general purpose IO.

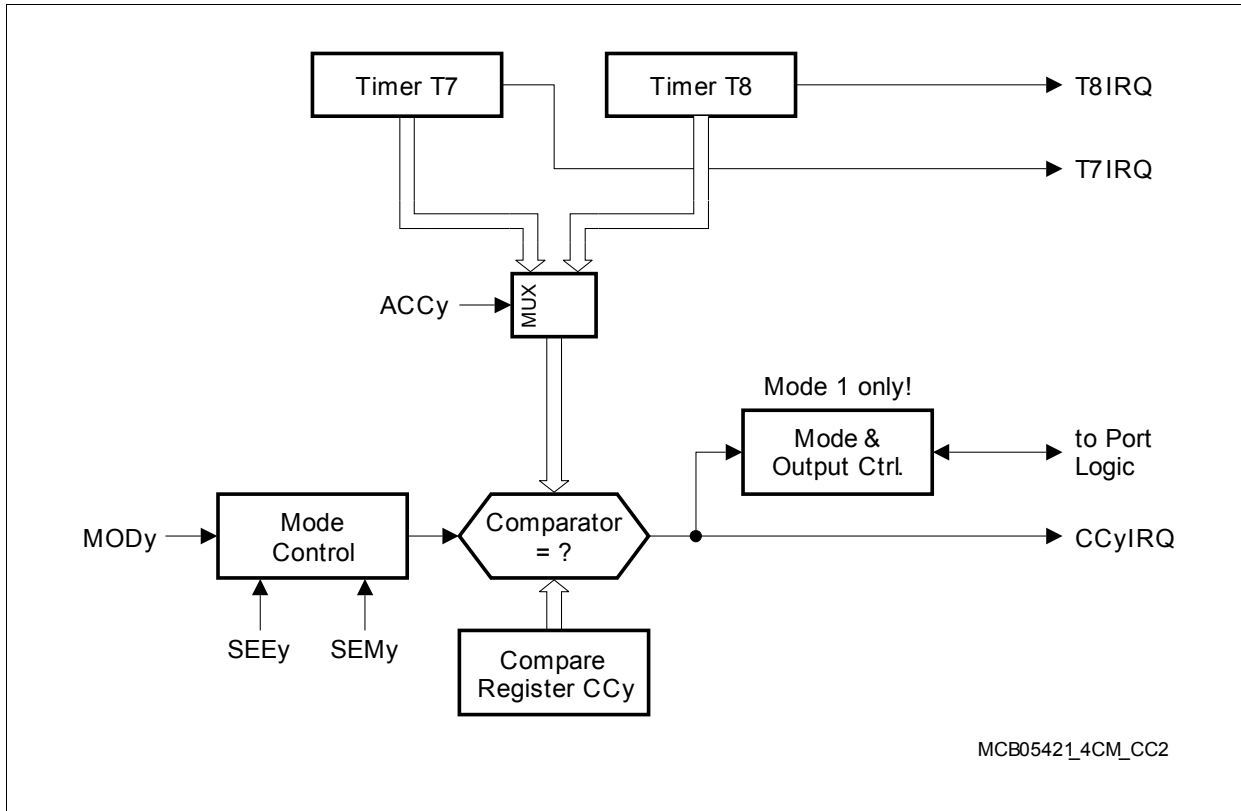
*Note: If compare mode 0 is programmed for one of the bank2 registers the double-register compare mode may be enabled for this register (see [Chapter 19.1.6](#)).*

#### **19.1.5.2 Compare Mode 1**

This is a compare mode which influences the associated output signal. Besides this, the basic operation is as in compare mode 0. Each time a match is detected between the contents of the compare register CCy and the allocated timer, the interrupt request line CCyIRQ is activated. In addition, the associated output signal is toggled. Several of these compare events are possible within a single timer period, if the compare value in register CCy is updated during the timer period.

*Note: If compare mode 1 is programmed for one of the bank1 registers the double-register compare mode may be enabled for this register (see [Section 19.1.6](#)).*

For the exact operation of the port output signal, please see [Section 19.1.8](#).



**Figure 19-5 Compare Mode 0 and 1 Block Diagram**

*Note: The signal remains unaffected in compare mode 0.*

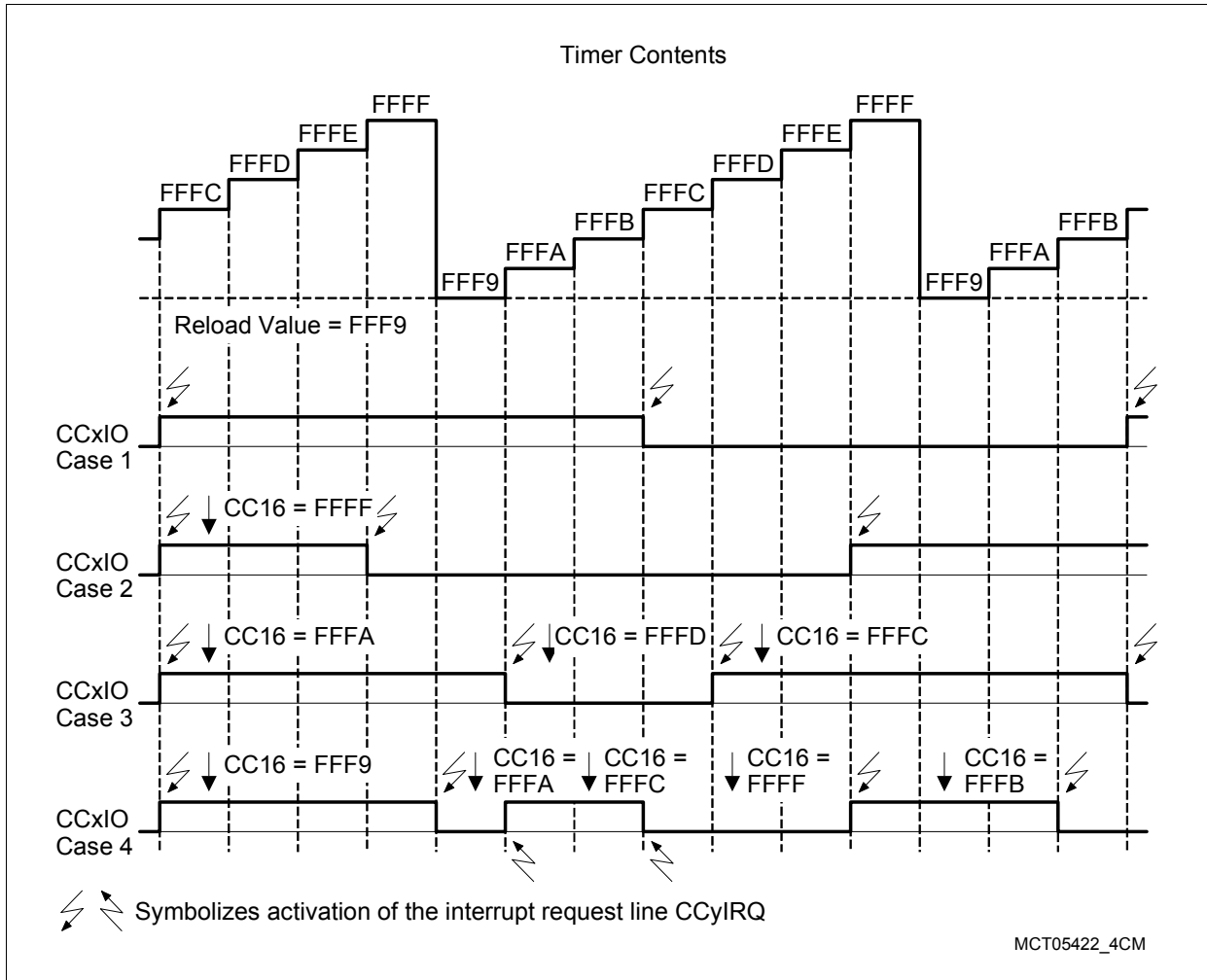
**Figure 19-6** illustrates a few example cases for compare modes 0 and 1.

In all examples, the reload value of the used timer is set to  $FFF9_H$ . When the timer overflows, it starts counting from this value upwards.

**In Case 1**, register CCy contains the value  $FFFC_H$ . When the timer reaches this value, a match is detected, and the interrupt request line CCyIRQ is activated. In compare mode 0, this is all that will happen. In compare mode 1, additionally the associated port output is toggled, causing an inversion of the output signal. If the contents of register CCy are not changed, this operation will take place each time the timer reaches the programmed compare value.

**In Case 2**, software reloads the compare register CCy with  $FFFF_H$  after the first match with  $FFFC_H$  has occurred. As the timer continues to count up, it finally reaches this new compare value, and a new match is detected, activating the interrupt request line (both modes) and toggling the output signal (compare mode 1). If then the compare value is left unchanged, the next match will occur when the timer reaches  $FFFF_H$  again.

This example illustrates, that further compare matches are possible within the current timer period (this is in contrast to compare modes 2 and 3).



**Figure 19-6 Examples for Compare Modes 0 and 1**

**In Case 3**, a new compare value, higher than the current timer contents, causes a new match within the current timer period. The compare register is reloaded with  $FFFA_H$  after the first match (at  $FFFC_H$ ). However, the timer has already passed this value. Thus, it will take until the timer reaches  $FFFA_H$  in the following timer period to cause the desired compare match. Reloading register CCy now with a value higher than the current timer contents will cause the next match within this period.

**In Case 4**, the compare values are equal to the timer reload value or to the maximum count value,  $FFFF_H$ .

### **19.1.5.3 Compare Mode 2**

Compare mode 2 is an interrupt-only mode similar to compare mode 0. The main difference is that only one compare match, corresponding to one interrupt request, is possible within a given timer period.

When a match is detected in compare mode 2 for the first time within a count period of the allocated timer, the interrupt request line CCyIRQ is activated. In addition, all further compare matches within the current timer period are disabled, even if a new compare value, higher than the current timer contents, would be written to the register. This blocking is only released when the allocated timer overflows. A new compare value written to the compare register after the first match will only go into effect within the following timer period.

### **19.1.5.4 Compare Mode 3**

Compare mode 3 is based on compare mode 2, but additionally influences the associated port pin. Only one compare event is possible within one timer period.

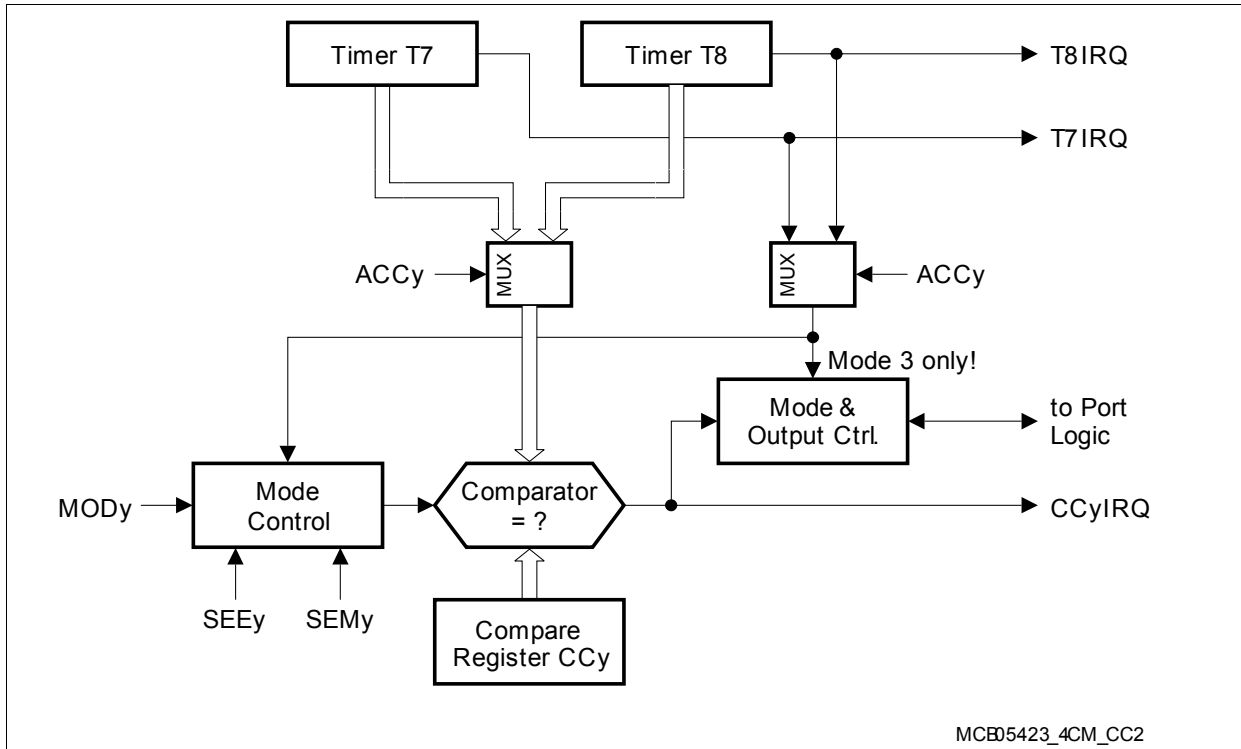
When a match is detected in compare mode 3 for the first time within a count period of the allocated timer, the interrupt request line CCyIRQ is activated, and the associated output signal is set to 1. In addition, all further compare matches within the current timer period are disabled, even if a new compare value, higher than the current timer contents, would be written to the register. This blocking is only released when the allocated timer overflows. A new compare value written to the compare register after the first match will only go into effect within the following timer period.

The overflow signal is also used to reset the associated output signal to 0.

Special attention has to be paid when the compare value is set equal to the timer reload value. In this case, the compare match signal would try to set the output signal, while the timer overflow tries to reset the output signal. This conflict is avoided such that the state of the output signal is left unchanged in this case.

*Note: When the compare value is changed from a value above the current timer contents to a value below the current timer contents, the new value is not recognized before the next timer period.*

For the exact operation of the port output signal, please see [Section 19.1.8](#).



**Figure 19-7 Compare Mode 2 and 3 Block Diagram**

*Note: The port signal remains unaffected in compare mode 2.*

**Figure 19-8** illustrates a few timing examples for compare modes 2 and 3.

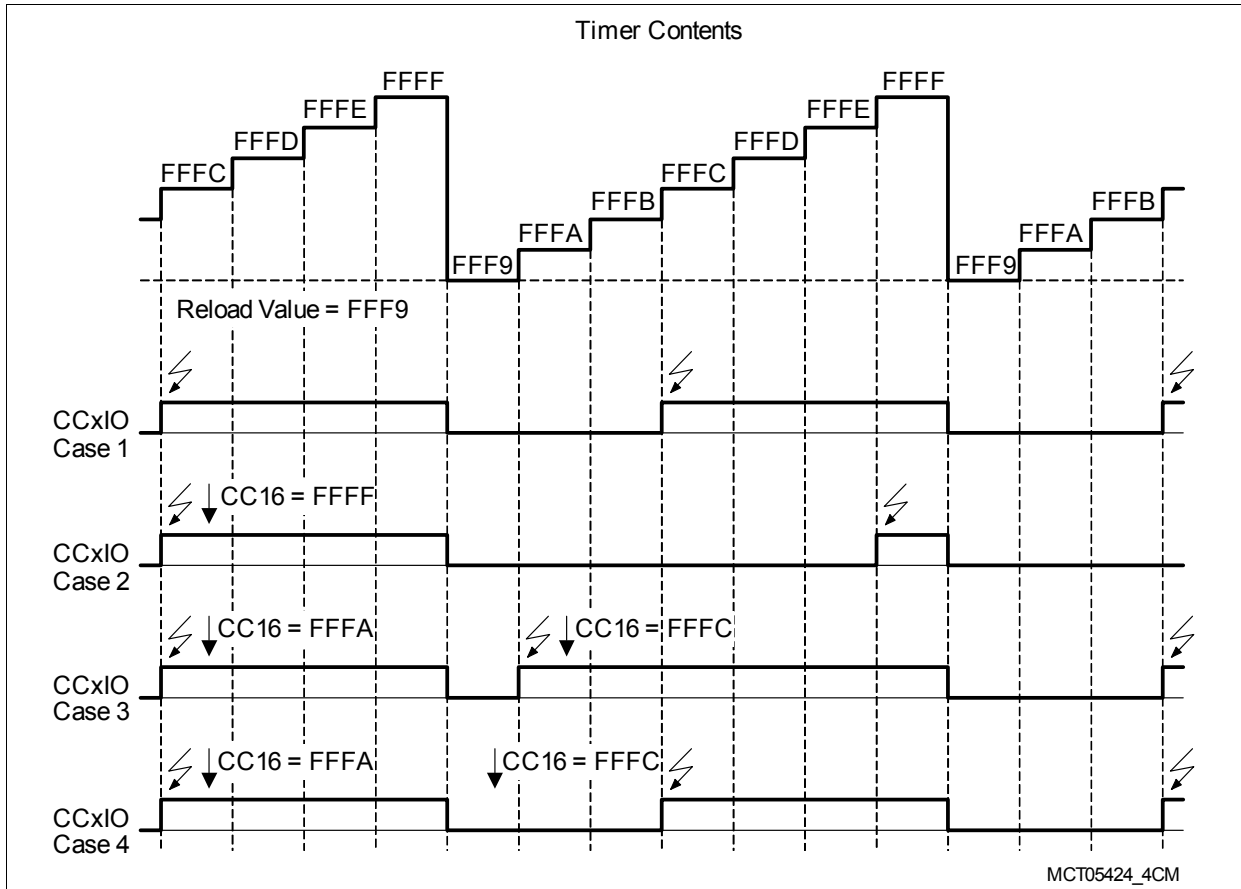
In all examples, the reload value of the used timer is set to  $FFF9_H$ . When the timer overflows, it starts counting from this value upwards.

**In Case 1**, register CCy contains the value  $FFFC_H$ . When the timer reaches this value, a match is detected, and the interrupt request line CCyIRQ is activated. In compare mode 2, this is all that will happen. In compare mode 3, additionally the associated port output is set to 1. The timer continues to count, and finally reaches its overflow. At this point, the port output is reset to 0 again. Note that, although not shown in the diagrams, the overflow signal of the timer also activates the associated interrupt request line TxIRQ. If the contents of register CCy are not changed, the port output will be set again during the following timer period, and reset again when the timer overflows. This operation is ideal for the generation of a pulse width modulated (PWM) signal with a minimum of software overhead. The pulse width is varied by changing the compare value accordingly.

**In Case 2**, the compare operation is blocked after the first match within a timer period. After the first match at  $FFFC_H$ , the interrupt request is generated and the port output is set. In addition, further compare matches are disabled. If now a new compare value is written to register CCy, no interrupt request and no port output influence will take place, although the new compare value is higher than the current timer contents. Only after the

## Capture/Compare Unit

overflow of the timer, the compare logic is enabled again, and the next match will be detected at  $FFFF_H$ . One can see, that this operation is ideal for PWM generation, as software can write a new compare value regardless of whether this value is higher or lower than the current timer contents. It is assured that the new value (usually written to the compare register in the appropriate interrupt service routine) will only go into effect during the following timer period.



**Figure 19-8 Timing Example for Compare Modes 2 and 3**

*Note: In compare mode 2, only interrupt requests are generated, in mode 3, also the output signals are generated.*

**In Case 3,** further examples for the operation of the compare match blocking are illustrated.

**In Case 4,** a new compare value is written to a compare register before the first match within the timer period. One can see that, of course, the originally programmed compare match (at  $FFFA_H$ ) will not take place. The first match will be detected at  $FFFC_H$ . However, it is important to note that the reprogramming of the compare register took place asynchronously - this means, the register was written to without any regard to the current contents of the timer. This is dangerous in the sense that the effect of such an asynchronous reprogramming is not easily predictable. If the timer would have already

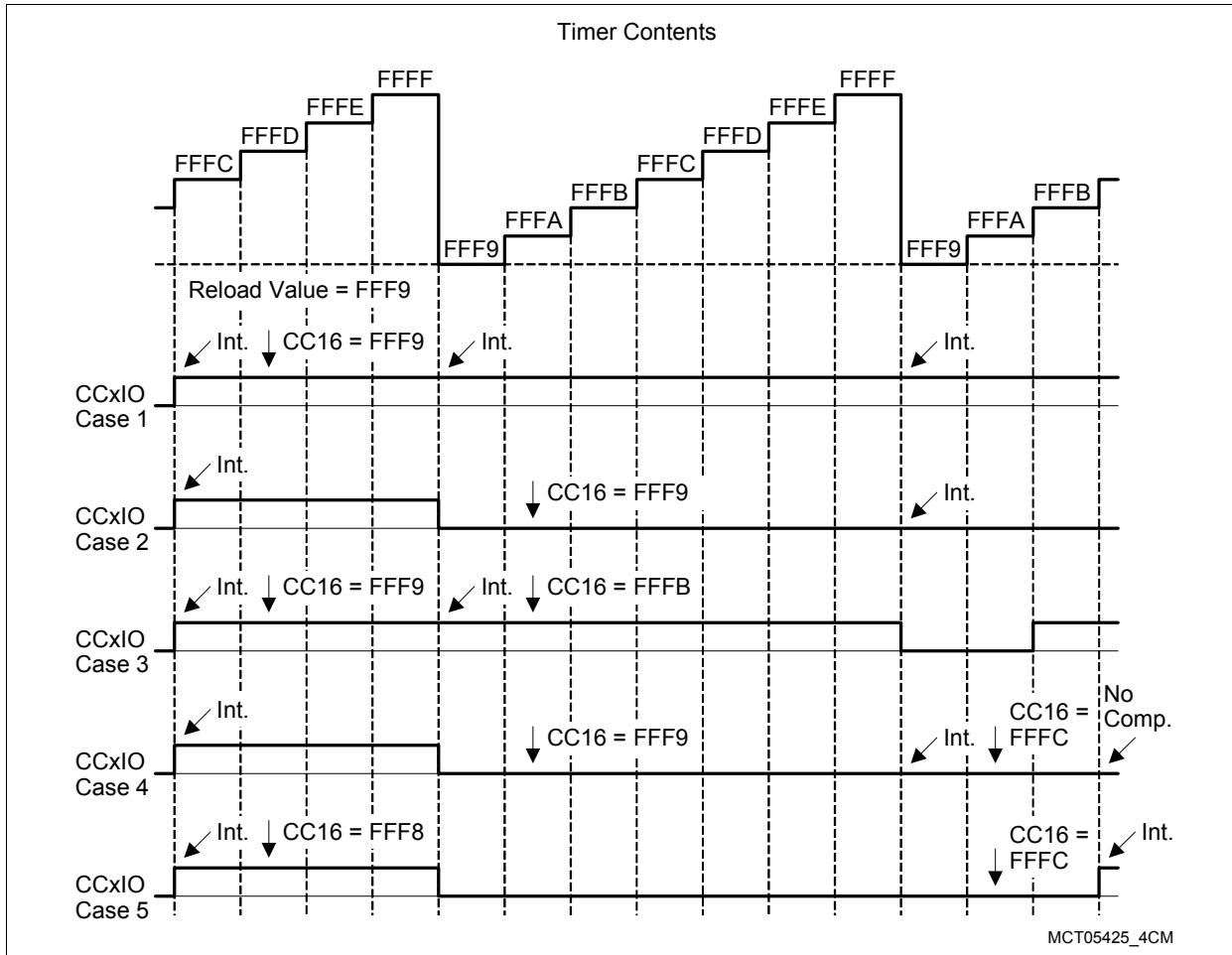
### **Capture/Compare Unit**

reached the originally programmed compare value of  $FFFA_H$  by the time the software wrote to the register, a match would have been detected and the reprogramming would go into effect during the next timer period.

The examples in [Figure 19-9](#) show special cases for compare modes 2 and 3. Case 1 illustrates the effect when the compare value is equal to the reload value of the timer. An interrupt is generated in both modes. In mode 3, the output signal is not affected - it remains at the high level. Setting the compare value equal to the reload value easily enables a 100% duty cycle signal for PWM generation. The important advantage here is that the compare interrupt is still generated and can be used to reload the next compare value. Thus, no special treatment is required for this case (see Case 3).

Cases 2, 4, and 5 show different options for the generation of a 0% duty cycle signal. Case 2 shows an asynchronous reprogramming of the compare value equal to the reload value. At the end of the current timer period, a compare interrupt will be generated, which enables software to set the next compare value. The disadvantage of this method is that at least two timer periods will pass until a new regular compare value can go into effect. The compare match with the reload value  $FFF9_H$  will block further compare matches during that timer period. This is additionally illustrated by Case 4.





**Figure 19-9 Special Cases in Compare Modes 2 and 3**

Case 5 shows an option to get around this problem. Here, the compare register is reloaded with FFF8<sub>H</sub>, a value which is lower than the timer reload value. Thus, the timer will never reach this value, and no compare match will be detected. The output signal will be set to 0 after the first timer overflow. However, after the second overflow, software now reloads the compare register with a regular compare value. As no compare blocking has taken place (since there was no compare match), the newly written compare value will go into effect during the current timer period.

### 19.1.6 Double-Register Compare Mode

The Double-Register Compare Mode makes it possible to further reduce software overhead for a number of applications. In this mode, two compare registers work together to control one output. This mode is selected via the DRM register, or by a special combination of compare modes for the two registers.

For double-register compare mode, the 16 capture/compare registers of a CAPCOM unit are regarded as two banks of 8 registers each. The lower eight registers form bank1,

**Capture/Compare Unit**

while the upper eight registers form bank2. For double-register mode, a bank1 register and a bank2 register form a register pair. Both registers of this register pair operate on the pin associated with the bank1 register.

The relationship between the bank1 and bank2 register of a pair and the effected output pins for double-register compare mode is listed in [Table 19-3](#).

**Table 19-3 CAPCOM2 Register Pairs for Double-Register Compare Mode**

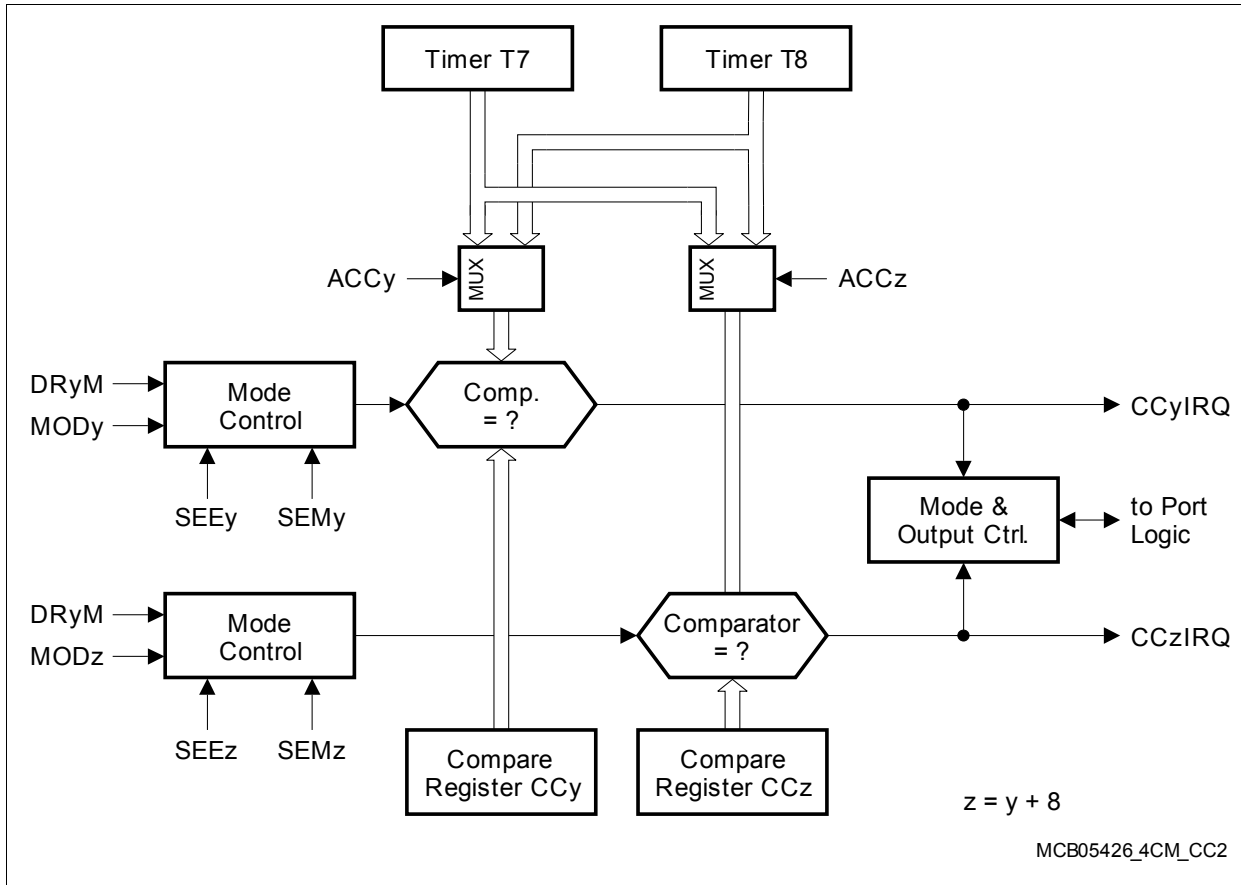
Register Pair		Used Output Pin	Control Bitfield in <b>CC2_DRM</b>
Bank 1	Bank 2		
CC16	CC24	CC16IO	DR0M
CC17	CC25	CC17IO	DR1M
CC18	CC26	CC18IO	DR2M
CC19	CC27	CC19IO	DR3M
CC20	CC28	CC20IO	DR4M
CC21	CC29	CC21IO	DR5M
CC22	CC30	CC22IO	DR6M
CC23	CC31	CC23IO	DR7M

The double-register compare mode can be programmed individually for each register pair. Double-register compare mode can be selected via a certain combination of compare modes for the two registers of a pair. The bank1 register must be programmed for mode 1 (with port influence), while the bank2 register must be programmed for mode 0 (interrupt-only).

Double-register compare mode can be controlled (this means, enabled or disabled) for each register pair via the associated control bitfield DRxM in register CC2\_DRM.

Double-register compare mode can be controlled individually for each of the register pairs.

In the block diagram of the double-register compare mode ([Figure 19-10](#)), a bank2 register will be referred to as CCz, while the corresponding bank1 register will be referred to as CCy.



**Figure 19-10 Double-Register Compare Mode Block Diagram**

When a match is detected for one of the two registers in a register pair (**CCy** or **CCz**), the associated interrupt request line (**CCyIRQ** or **CCzIRQ**) is activated, and pin **CCyIO**, corresponding to the bank1 register **CCy**, is toggled. The generated interrupt always corresponds to the register that caused the match.

*Note: If a match occurs simultaneously for both register **CCy** and register **CCz** of the register pair, pin **CCyIO** will be toggled only once, but two separate compare interrupt requests will be generated.*

Each of the two registers of a pair can be individually allocated to one of the two timers in the CAPCOM unit. This offers a wide variety of applications, as the two timers can run in different modes with different resolution and frequency. However, this might require sophisticated software algorithms to handle the different timer periods.

*Note: The signals **CCzIO** (which do not serve for double-register compare mode) may be used for general purpose IO.*

### 19.1.7 CAPCOM Interrupts

Upon a capture or compare event, the interrupt request flag CCyIR for the respective capture/compare register CCy is automatically set. This flag can be used to generate an interrupt or trigger a PEC service request when enabled by the interrupt enable bit CCyIE.

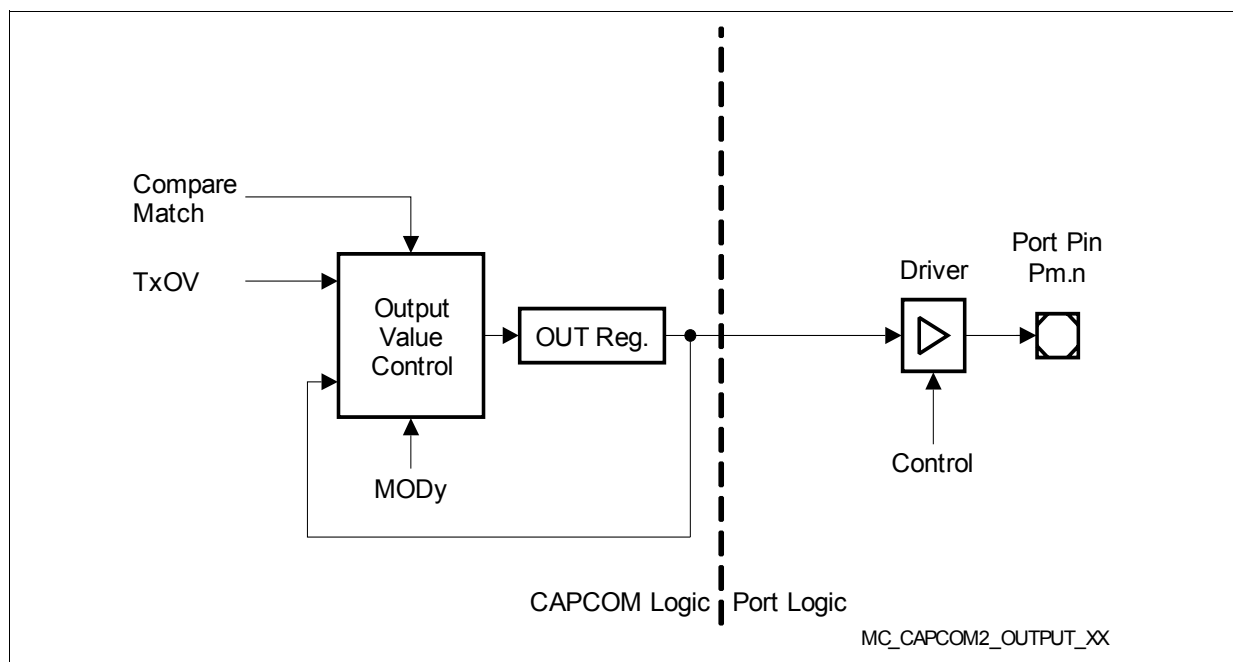
Capture interrupts can be regarded as external interrupt requests with the additional feature of recording the time at which the triggering event occurred.

Each of the capture/compare registers (CCy) has its own bitaddressable interrupt control register CC2\_CCyIC and its own interrupt vector allocated. These registers are organized in the same way as all other interrupt control registers.

### 19.1.8 Compare Output Signal Generation

This section discusses the interaction between the CAPCOM Unit and the Port Logic. The block diagram illustrated in [Figure 19-11](#) details the logic of the block “Mode & Output Control”, shown in [Figure 19-5](#), [Figure 19-7](#), and [Figure 19-10](#).

Each output signal is stored in its associated bit of the compare output register CC2\_OUT. The individual bits are updated each time an associated compare event occurs. The bits of these registers are connected to the respective port pins as an alternate output function of a port line.



**Figure 19-11 Port Output Block Diagram for Compare Modes**

*Note: A compare output signal is visible at the pin only in compare modes 1 or 3.*

The output signal of a compare event can either be a 1, a 0, the complement of the current level, or the previous level. The block 'Output Value Control' determines the correct new level based on the compare event, the timer overflow signal, and the current state of the OUT register bit. For the output toggle function (e.g. in compare mode 1), the state of the OUT register bit is read, inverted, and then written back.

### **19.1.9 Single Event Mode**

If an application requires that one and only one compare event needs to take place (within a certain time frame), single event operation helps to reduce software overhead and to eliminate the need for fast reaction upon events.

In order to achieve a single event operation without this feature, software would have to either disable the compare mode or write a new value, which is outside of the count range of the timer, into the compare register, after the programmed compare match has taken place. Thus, usually an interrupt service routine is required to perform this operation. Interrupt response time may be critical if the timer period is very short - the disable operation needs to be completed before the timer would reach the same value again.

The single event operation eliminates the need for software to react after the first compare match. The complete operation can be set up before the event, and no action is required after the event. The hardware takes care of generating only one event, and then disabling all further compare matches.

This option is programmed via the Single Event Mode register **CC2\_SEM** and the Single Event Enable register **CC2\_SEE**. Each register provides one bit for each CCy register of a unit.

To setup a single event operation for a CCy register, software first programs the desired compare operation and compare value, and then sets the respective bit in register **CC2\_SEM** to enable the single event mode. At last, the respective event enable bit in register **CC2\_SEE** is set.

When the programmed compare match occurs, all operations of the selected compare mode take place. In addition, hardware automatically disables all further compare matches and reset the event enable bit in register **CC2\_SEE** to 0. As long as this bit is cleared, any compare operation is disabled. To setup a new event, this bit must first be set again.

### **19.1.10 Staggered and Non-Staggered Operation**

The CAPCOM2 unit can run in one of two basic operation modes: Staggered Mode and Non-Staggered Mode. The selection between these modes is performed via register **IOC**.

In staggered mode, a CAPCOM operation cycle consists of 8 module clock cycles, and the outputs of the compare events of the different registers are staggered, that is, the

## **Capture/Compare Unit**

outputs for compare matches with the same compare value are not switched at the same time, but with a fixed time delay. This operation helps to reduce noise and peak power consumption caused by simultaneous switching outputs.

In non-staggered Mode, a CAPCOM operation cycle is equal to one module clock cycle, and all compare outputs for compare events with the same compare value are switched in the same clock cycle. This mode offers a faster operation and increased resolution of the CAPCOM unit, 8 times higher than in staggered mode.

### **Staggered Mode**

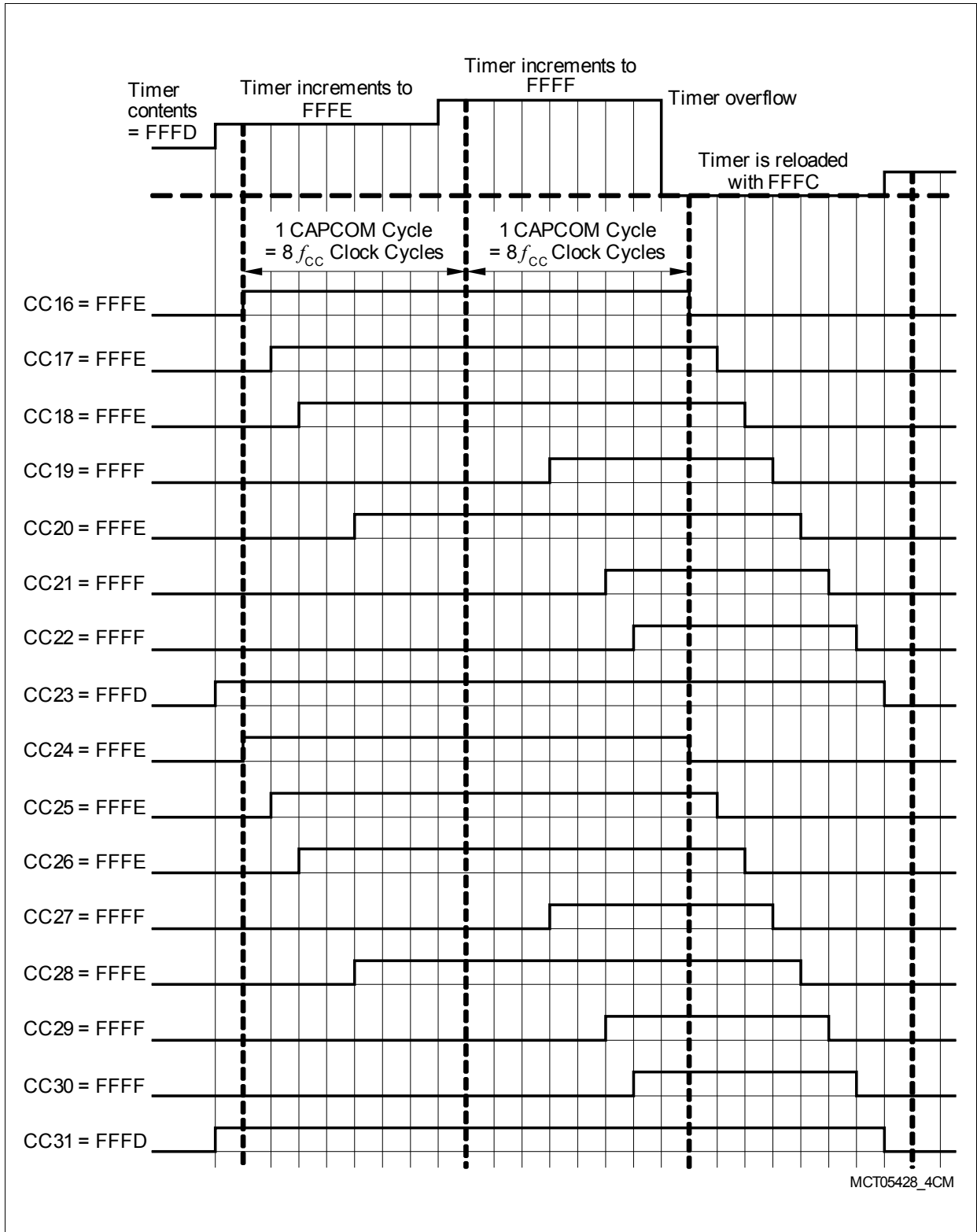
**Figure 19-12** illustrates the staggered mode operation for CAPCOM2. In this example, all CCy registers are programmed for compare mode 3.

Registers CC16, CC17, and CC18 are all programmed for a compare value of  $FFFE_H$ . When the timer increments to  $FFFE_H$ , the comparator detects a match for all of the three registers. The output CC16IO of register CC16 is switched to 1 one cycle after the comparator match. However, the outputs CC17IO and CC18IO are not switched at the same time, but one, respectively two cycles later. This staggering of the outputs continues for all registers including register CC23. The number of the register indicates the delay of the output signal in clock cycles - the output of register CC23 is switched 7 cycles later than the one of register CC16. In the example, the compare value for register CC13 is set to  $FFFD_H$ . Thus, the output is switched in the last clock cycle of the CAPCOM cycle in which the timer reached  $FFFD_H$ .

When the timer overflows, all compare outputs are reset to 0 (compare mode 3). Again, the staggering of the output signals can be seen from **Figure 19-12**.

Looking at registers CC24 through CC31 shows that their outputs are switched in parallel to the respective outputs of registers CC16 through CC23. In fact, the staggering is performed in parallel for the upper and the lower register bank. In this way, it is assured, that both compare signals of a register pair in double-register compare mode operate simultaneously.

*Note: This is a general description and only refers to channels connected to pins.*



**Figure 19-12 Staggered Mode Operation**

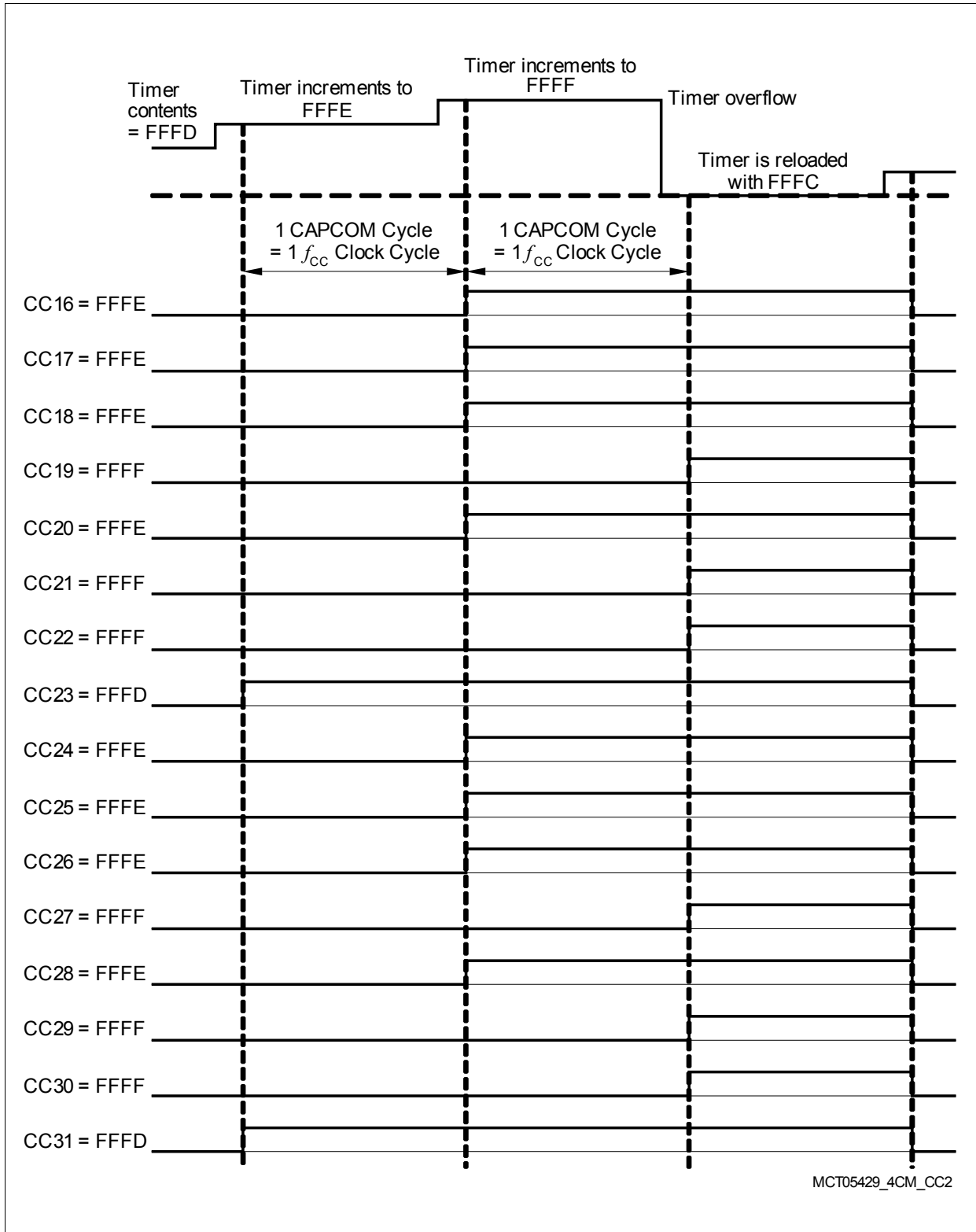
### **Non-Staggered Mode**

To gain maximum speed and resolution with a CAPCOM unit, it can be switched to non-staggered mode. In this mode, one CAPCOM operation cycle is equal to one module clock cycle. Timer increment and the comparison of its new contents with the contents of the compare register takes place within one clock cycle. The appropriate output signals are switched in the following clock cycle (in parallel to the next possible timer increment and comparison).

**Figure 19-13** illustrates the non-staggered mode for CAPCOM2 unit. Note that when the timer overflows, it also takes one additional clock cycle to switch the output signals.

*Note: This is a general description and only refers to channels connected to pins.*





**Figure 19-13 Non-Staggered Mode Operation**

### 19.1.11 External Input Signal Requirements

The external input signals of a CAPCOM2 unit are sampled by the CAPCOM2 logic based on the module clock and the basic operation mode (staggered or non-staggered mode). To assure that a signal level is recognized correctly, its high or low level must be held active for at least one complete sampling period.

The duration of a sampling period is one module clock cycle in non-staggered mode, and 8 module clock cycles in staggered mode. To recognize a signal transition, the signal needs to be sampled twice. If the level of the first sampling is different to the level detected during the second sampling, a transition is recognized. Therefore, a minimum of two sampling periods are required for the sampling of an external input signal. Thus, the maximum frequency of an input signal must not be higher than half the module clock frequency in non-staggered mode, and a  $1/16^{\text{th}}$  of the module clock frequency in staggered mode.

**Table 19-4** summarizes the requirements and limits for external input signals.

**Table 19-4 CAPCOM2 External Input Signal Limits**

	<b>Non-Staggered Mode</b>	<b>Staggered Mode</b>
Maximum Input Frequency	$f_{CC} / 2$	$f_{CC} / 16$
Minimum Input Signal Level Duration	$1 / f_{CC}$	$8 / f_{CC}$

In order to use an external signal as a count or capture input, the port pin to which it is connected must be configured as input.

*Note: For example for test purposes a pin used as a count or capture input may be configured as output. Software or an other peripheral may control the respective signal and thus trigger count or capture events.*

In order to cause a compare output signal to be seen by the external world, the associated port pin must be configured as output. For compare output signals the output of register CC2\_OUT is used as an alternate output function of a port.

## 19.2 CAPCOM2 Registers

The following table presents a summary of the registers provided in the CAPCOM2 module.

**Table 19-5 CAPCOM2 Module Register Summary**

Name	Description	Address		Reset Value
		16-Bit	8-Bit	

### Capture / Compare Unit 2 (CAPCOM2)

<b>CC2_ID</b>	CAPCOM2 Identification Register	FFEE <sub>H</sub>	-	50XX <sub>H</sub>
<b>CC2_M4</b>	CAPCOM2 Mode Control Register 4	FF22 <sub>H</sub>	91 <sub>H</sub>	0000 <sub>H</sub>
<b>CC2_M5</b>	CAPCOM2 Mode Control Register 5	FF24 <sub>H</sub>	92 <sub>H</sub>	0000 <sub>H</sub>
<b>CC2_M6</b>	CAPCOM2 Mode Control Register 6	FF26 <sub>H</sub>	93 <sub>H</sub>	0000 <sub>H</sub>
<b>CC2_M7</b>	CAPCOM2 Mode Control Register 7	FF28 <sub>H</sub>	94 <sub>H</sub>	0000 <sub>H</sub>
<b>CC2_SEE</b>	CAPCOM2 Single Event Enable Register	FE2A <sub>H</sub>	15 <sub>H</sub>	0000 <sub>H</sub>
<b>CC2_SEM</b>	CAPCOM2 Single Event Mode Register	FE28 <sub>H</sub>	14 <sub>H</sub>	0000 <sub>H</sub>
<b>CC2_DRM</b>	CAPCOM2 Double Register Mode Register	FF2A <sub>H</sub>	95 <sub>H</sub>	0000 <sub>H</sub>
<b>CC2_OUT</b>	CAPCOM2 Output Register	FF2C <sub>H</sub>	96 <sub>H</sub>	0000 <sub>H</sub>
<b>CC2_T7</b>	CAPCOM2 Timer 7 Register	F050 <sub>H</sub>	28 <sub>H</sub>	0000 <sub>H</sub>
<b>CC2_T8</b>	CAPCOM2 Timer 8 Register	F052 <sub>H</sub>	29 <sub>H</sub>	0000 <sub>H</sub>
<b>CC2_T7REL</b>	CAPCOM2 Timer 7 Reload Register	F054 <sub>H</sub>	2A <sub>H</sub>	0000 <sub>H</sub>
<b>CC2_T8REL</b>	CAPCOM2 Timer 8 Reload Register	F056 <sub>H</sub>	2B <sub>H</sub>	0000 <sub>H</sub>
<b>CC2_T78CON</b>	CAPCOM2 Timer 7/8 Control Register	FF20 <sub>H</sub>	90 <sub>H</sub>	0000 <sub>H</sub>
<b>CC2_IOC</b>	CAPCOM2 I/O Control Register	F066 <sub>H</sub>	33 <sub>H</sub>	0000 <sub>H</sub>
<b>CC2_CC16</b>	CAPCOM2 Register 16	FE60 <sub>H</sub>	30 <sub>H</sub>	0000 <sub>H</sub>
<b>CC2_CC17</b>	CAPCOM2 Register 17	FE62 <sub>H</sub>	31 <sub>H</sub>	0000 <sub>H</sub>
<b>CC2_CC18</b>	CAPCOM2 Register 18	FE64 <sub>H</sub>	32 <sub>H</sub>	0000 <sub>H</sub>
<b>CC2_CC19</b>	CAPCOM2 Register 19	FE66 <sub>H</sub>	33 <sub>H</sub>	0000 <sub>H</sub>
<b>CC2_CC20</b>	CAPCOM2 Register 20	FE68 <sub>H</sub>	34 <sub>H</sub>	0000 <sub>H</sub>
<b>CC2_CC21</b>	CAPCOM2 Register 21	FE6A <sub>H</sub>	35 <sub>H</sub>	0000 <sub>H</sub>
<b>CC2_CC22</b>	CAPCOM2 Register 22	FE6C <sub>H</sub>	36 <sub>H</sub>	0000 <sub>H</sub>
<b>CC2_CC23</b>	CAPCOM2 Register 23	FE6E <sub>H</sub>	37 <sub>H</sub>	0000 <sub>H</sub>
<b>CC2_CC24</b>	CAPCOM2 Register 24	FE70 <sub>H</sub>	38 <sub>H</sub>	0000 <sub>H</sub>

**Table 19-5 CAPCOM2 Module Register Summary (cont'd)**

Name	Description	Address		Reset Value
		16-Bit	8-Bit	
<b>CC2_CC25</b>	CAPCOM2 Register 25	FE72 <sub>H</sub>	39 <sub>H</sub>	0000 <sub>H</sub>
<b>CC2_CC26</b>	CAPCOM2 Register 26	FE74 <sub>H</sub>	3A <sub>H</sub>	0000 <sub>H</sub>
<b>CC2_CC27</b>	CAPCOM2 Register 27	FE76 <sub>H</sub>	3B <sub>H</sub>	0000 <sub>H</sub>
<b>CC2_CC28</b>	CAPCOM2 Register 28	FE78 <sub>H</sub>	3C <sub>H</sub>	0000 <sub>H</sub>
<b>CC2_CC29</b>	CAPCOM2 Register 29	FE7A <sub>H</sub>	3D <sub>H</sub>	0000 <sub>H</sub>
<b>CC2_CC30</b>	CAPCOM2 Register 30	FE7C <sub>H</sub>	3E <sub>H</sub>	0000 <sub>H</sub>
<b>CC2_CC31</b>	CAPCOM2 Register 31	FE7E <sub>H</sub>	3F <sub>H</sub>	0000 <sub>H</sub>
<b>CC2_T7IC</b>	CAPCOM2 Timer 7 Interrupt Control Register	FF6C <sub>H</sub>	BD <sub>H</sub>	0000 <sub>H</sub>
<b>CC2_T8IC</b>	CAPCOM2 Timer 8 Interrupt Control Register	FF6E <sub>H</sub>	BE <sub>H</sub>	0000 <sub>H</sub>
<b>CC2_CC16IC</b>	CAPCOM2 Register 16 Interrupt Control Register Shared Interrupt node, see ISSR register	F1C0 <sub>H</sub>	B0 <sub>H</sub>	0000 <sub>H</sub>
<b>CC2_CC17IC</b>	CAPCOM2 Register 17 Interrupt Control Register Shared Interrupt node, see ISSR register	F1C2 <sub>H</sub>	B1 <sub>H</sub>	0000 <sub>H</sub>
<b>CC2_CC18IC</b>	CAPCOM2 Register 18 Interrupt Control Register Shared Interrupt node, see ISSR register	F1C4 <sub>H</sub>	B2 <sub>H</sub>	0000 <sub>H</sub>
<b>CC2_CC19IC</b>	CAPCOM2 Register 19 Interrupt Control Register Shared Interrupt node, see ISSR register	F1C6 <sub>H</sub>	B3 <sub>H</sub>	0000 <sub>H</sub>
<b>CC2_CC20IC</b>	CAPCOM2 Register 20 Interrupt Control Register Shared Interrupt node, see ISSR register	F1C8 <sub>H</sub>	B4 <sub>H</sub>	0000 <sub>H</sub>
<b>CC2_CC21IC</b>	CAPCOM2 Register 21 Interrupt Control Register Shared Interrupt node, see ISSR register	F1CA <sub>H</sub>	B5 <sub>H</sub>	0000 <sub>H</sub>
<b>CC2_CC22IC</b>	CAPCOM2 Register 22 Interrupt Control Register Shared Interrupt node, see ISSR register	F1CC <sub>H</sub>	B6 <sub>H</sub>	0000 <sub>H</sub>
<b>CC2_CC23IC</b>	CAPCOM2 Register 23 Interrupt Control Register Shared Interrupt node, see ISSR register	F1CE <sub>H</sub>	B7 <sub>H</sub>	0000 <sub>H</sub>

**Table 19-5 CAPCOM2 Module Register Summary (cont'd)**

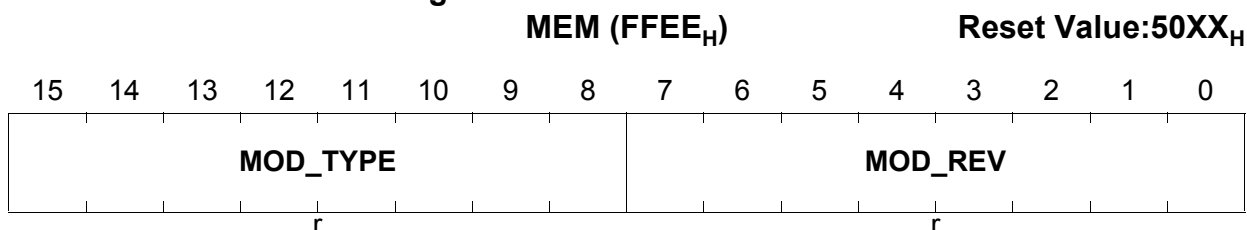
Name	Description	Address		Reset Value
		16-Bit	8-Bit	
<b>CC2_CC24IC</b>	CAPCOM2 Register 24 Interrupt Control Register Shared Interrupt node, see ISSR register	F1D0 <sub>H</sub>	B8 <sub>H</sub>	0000 <sub>H</sub>
<b>CC2_CC25IC</b>	CAPCOM2 Register 25 Interrupt Control Register Shared Interrupt node, see ISSR register	F1D2 <sub>H</sub>	B9 <sub>H</sub>	0000 <sub>H</sub>
<b>CC2_CC26IC</b>	CAPCOM2 Register 26 Interrupt Control Register Shared Interrupt node, see ISSR register	F1D4 <sub>H</sub>	BA <sub>H</sub>	0000 <sub>H</sub>
<b>CC2_CC27IC</b>	CAPCOM2 Register 27 Interrupt Control Register Shared Interrupt node, see ISSR register	F1D6 <sub>H</sub>	BB <sub>H</sub>	0000 <sub>H</sub>
<b>CC2_CC28IC</b>	CAPCOM2 Register 28 Interrupt Control Register Shared Interrupt node, see ISSR register	F1D8 <sub>H</sub>	BC <sub>H</sub>	0000 <sub>H</sub>
<b>CC2_CC29IC</b>	CAPCOM2 Register 29 Interrupt Control Register Shared Interrupt node, see ISSR register	F1DA <sub>H</sub>	C2 <sub>H</sub>	0000 <sub>H</sub>
<b>CC2_CC30IC</b>	CAPCOM2 Register 30 Interrupt Control Register Shared Interrupt node, see ISSR register	F1DC <sub>H</sub>	C6 <sub>H</sub>	0000 <sub>H</sub>
<b>CC2_CC31IC</b>	CAPCOM2 Register 31 Interrupt Control Register Shared Interrupt node, see ISSR register	F1DE <sub>H</sub>	CA <sub>H</sub>	0000 <sub>H</sub>

### 19.2.1 Identification Register

For module type and revision identification the CAPCOM2 unit provides a specific read-only identification register.

#### CC2\_ID

#### CAPCOM2 Identification Register



Field	Bits	Typ	Description
MOD_REV	[7:0]	r	<b>Module Revision Number</b> Defines the revision number. The value of a module revision starts with 01 <sub>H</sub> (first revision).
MOD_TYPE	[15:8]	r	<b>Module Identification Number</b> Defines the module identification number (50 <sub>H</sub> = CAPCOM2).

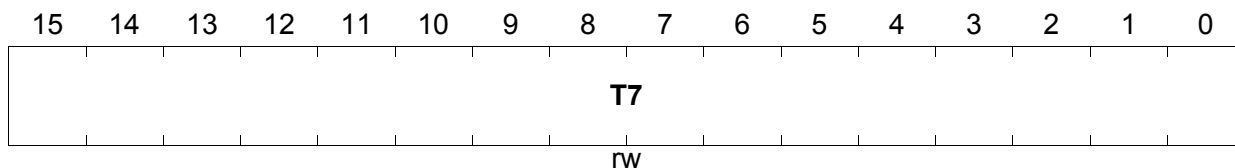
## 19.2.2 Timer 7/8 Registers

### CC2\_T7

**CAPCOM2 Timer 7 Register**

**ESFR (F050<sub>H</sub>/28<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



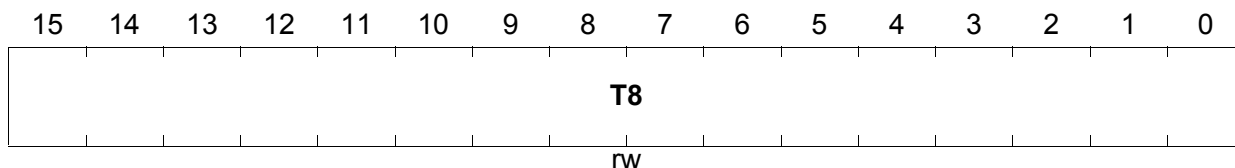
Field	Bits	Typ	Description
T7	[15:0]	rw	<b>Timer 7 Current Value</b> Current value of the Timer 7

### CC2\_T8

**CAPCOM2 Timer 8 Register**

**ESFR (F052<sub>H</sub>/29<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



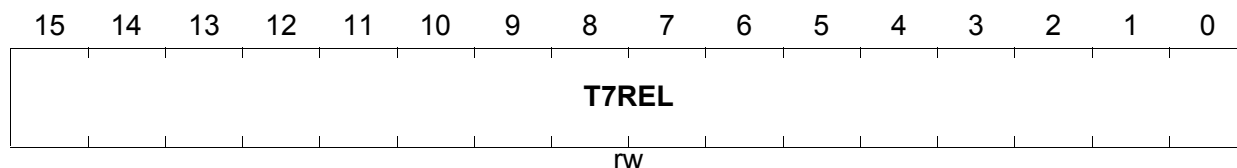
Field	Bits	Typ	Description
T8	[15:0]	rw	<b>Timer 8 Current Value</b> Current value of the Timer 8

**CC2\_T7REL**

**CAPCOM2 Timer 7 Reload Register**

**ESFR (F054<sub>H</sub>/2A<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



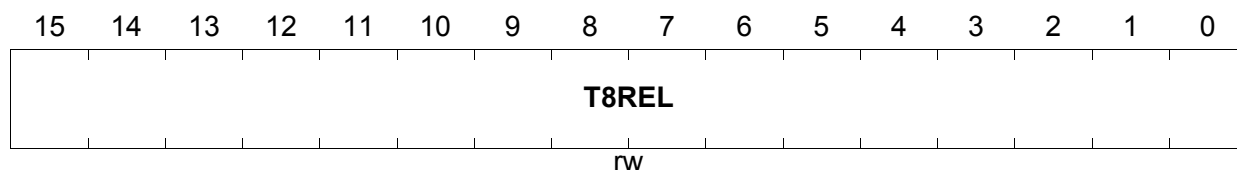
Field	Bits	Typ	Description
<b>T7REL</b>	[15:0]	rw	<b>Timer 7 Reload Value</b> Reload value of the Timer 7

**CC2\_T8REL**

**CAPCOM2 Timer 8 Reload Register**

**ESFR (F056<sub>H</sub>/2B<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



Field	Bits	Typ	Description
<b>T8REL</b>	[15:0]	rw	<b>Timer 8 Reload Value</b> Reload value of the Timer 8



## 19.2.3 Timer 7/8 Control Register

**CC2\_T78CON**

**CAPCOM2 Timer 7/8 Control Register**

**SFR (FF20<sub>H</sub>/90<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	T8R	T8RSEL	T8M			T8I		-	T7R	T7RSEL	T7M			T7I	
-	rw	rw	rw			rw		-	rw	rw	rw			rw	

Field	Bits	Typ	Description
<b>T7I, T8I</b>	[2:0], [10:8]	rw	<b>Timer/Counter Tx Input Selection</b> <b>Timer Mode</b> (TxM = 0): Input frequency $f_{Tx} = f_{CC}/2^{(<TxI>+3)}$ or $f_{CC}/2^{(<TxI>)}$ , depending on (non-)staggered mode, see <a href="#">Table 19-1</a> <b>Counter Mode</b> (TxM = 1): 000 <sub>B</sub> Overflow/Underflow of GPT Timer T6 001 <sub>B</sub> Positive (rising) edge on pin TxIN 010 <sub>B</sub> Negative (falling) edge on pin TxIN 011 <sub>B</sub> Any edge (rising and falling) on pin TxIN 1XX <sub>B</sub> Reserved. Do not use this combination! <i>Note: For timer T8 the only option in counter mode is 000<sub>B</sub>. T8 stop in all other cases.</i>
<b>T7M, T8M</b>	3, 11	rw	<b>Timer / Counter x Mode Selection</b> 0 <sub>B</sub> Timer Mode 1 <sub>B</sub> Counter Mode
<b>T7R, T8R</b>	6, 14	rw	<b>Timer / Counter x Run Control</b> 0 <sub>B</sub> Timer/Counter x is disabled. 1 <sub>B</sub> Timer/Counter x is enabled.
<b>T7RSEL</b>	[5:4]	rw	<b>Timer T7 External Run Selection</b> Bit field T7RSEL defines the event of signal T7HR that can set the run bit T7R by HW. 00 <sub>B</sub> The external setting of T7R is disabled. 01 <sub>B</sub> Bit T7R is set if a rising edge of signal T7HR is detected. 10 <sub>B</sub> Bit T7R is set if a falling edge of signal T7HR is detected. 11 <sub>B</sub> Bit T7R is set if an edge of signal T7HR is detected.

Field	Bits	Typ	Description
<b>T8RSEL</b>	[13:12]	rw	<b>Timer T8 External Run Selection</b> Bit field T8RSEL defines the event of signal T8HR that can set the run bit T8R by HW. 00 <sub>B</sub> The external setting of T8R is disabled. 01 <sub>B</sub> Bit T8R is set if a rising edge of signal T8HR is detected. 10 <sub>B</sub> Bit T8R is set if a falling edge of signal T8HR is detected. 11 <sub>B</sub> Bit T8R is set if an edge of signal T8HR is detected.

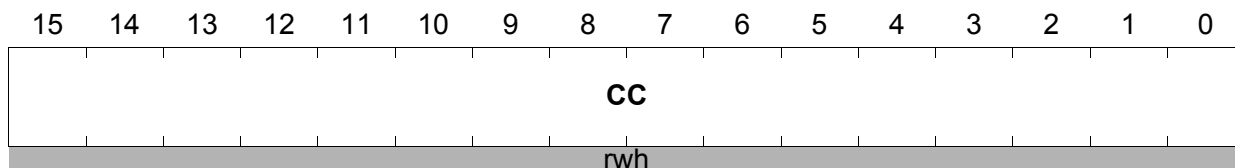
## 19.2.4 Capture/Compare Registers

**CC2\_CCy (y=16-31)**

**CAPCOM2 Capture/Compare Register y**

**SFR (FE60<sub>H</sub>-32+2\*y / 30<sub>H</sub>-16+y)**

**Reset Value: 0000<sub>H</sub>**



Field	Bits	Typ	Description
CC	[15:0]	rwh	<b>Capture Register Value</b> Current value of the Capture/Compare register y

## 19.2.5 Capture/Compare Mode Registers

### CC2\_M4

#### CAPCOM2 Mode Control Register 4

**SFR (FF22<sub>H</sub>/91<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>ACC 19</b>	<b>MOD19</b>			<b>ACC 18</b>	<b>MOD18</b>			<b>ACC 17</b>	<b>MOD17</b>			<b>ACC 16</b>	<b>MOD16</b>		
rw	rw			rw	rw			rw	rw			rw	rw		

Field	Bits	Type	Description
<b>ACCy</b> (y=16-19)	4*y-61	rw	<b>Allocation Bit for CAPCOM Register CCy</b> (y = 16-19) 0 <sub>B</sub> CCy allocated to Timer T7 1 <sub>B</sub> CCy allocated to Timer T8
<b>MODy</b> (y=16-19)	[4*y-62:4*y-64]	rw	<b>Mode Selection for CAPCOM Register CCy</b> (y = 16-19) See <a href="#">Table 19-2</a> .

### CC2\_M5

#### CAPCOM2 Mode Control Register 5

**SFR (FF24<sub>H</sub>/92<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>ACC 23</b>	<b>MOD23</b>			<b>ACC 22</b>	<b>MOD22</b>			<b>ACC 21</b>	<b>MOD21</b>			<b>ACC 20</b>	<b>MOD20</b>		
rw	rw			rw	rw			rw	rw			rw	rw		

Field	Bits	Type	Description
<b>ACCy</b> (y=20-23)	4*y-77	rw	<b>Allocation Bit for CAPCOM Register CCy</b> (y = 20-23) 0 <sub>B</sub> CCy allocated to Timer T7 1 <sub>B</sub> CCy allocated to Timer T8
<b>MODy</b> (y=20-23)	[4*y-78:4*y-80]	rw	<b>Mode Selection for CAPCOM Register CCy</b> (y = 20-23) See <a href="#">Table 19-2</a> .

## CC2\_M6

### CAPCOM2 Mode Control Register 6

**SFR (FF26<sub>H</sub>/93<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ACC 27	MOD27			ACC 26	MOD26			ACC 25	MOD25			ACC 24	MOD24		
rw	rw			rw	rw			rw	rw			rw	rw		

Field	Bits	Type	Description
<b>ACC<sub>y</sub></b> (y=24-27)	4*y-93	rw	<b>Allocation Bit for CAPCOM Register CC<sub>y</sub></b> (y = 24-28) 0 <sub>B</sub> CC <sub>y</sub> allocated to Timer T7 1 <sub>B</sub> CC <sub>y</sub> allocated to Timer T8
<b>MOD<sub>y</sub></b> (y=24-27)	[4*y-94:4*y-96]	rw	<b>Mode Selection for CAPCOM Register CC<sub>y</sub></b> (y = 24-27) See <a href="#">Table 19-2</a> .

## CC2\_M7

### CAPCOM2 Mode Control Register 7

**SFR (FF28<sub>H</sub>/94<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ACC 31	MOD31			ACC 30	MOD30			ACC 29	MOD29			ACC 28	MOD28		
rw	rw			rw	rw			rw	rw			rw	rw		

Field	Bits	Type	Description
<b>ACC<sub>y</sub></b> (y=28-31)	4*y-109	rw	<b>Allocation Bit for CAPCOM Register CC<sub>y</sub></b> (y = 28-31) 0 <sub>B</sub> CC <sub>y</sub> allocated to Timer T7 1 <sub>B</sub> CC <sub>y</sub> allocated to Timer T8
<b>MOD<sub>y</sub></b> (y=28-31)	[4*y-110:4*y-112]	rw	<b>Mode Selection for CAPCOM Register CC<sub>y</sub></b> (y = 28-31) See <a href="#">Table 19-2</a> .

## 19.2.6 Compare Output Register

The CAPCOM2's compare output serves two registers in parallel, the port output register for binary compatibility and a separate one for enhanced functionality. The CAPCOM2 compare output and the port output latch is muxed in the port logic.

Compare output is visible at the pin if compare mode 1 or 3 is programmed in the CAPCOM2.

### CC2\_OUT

#### CAPCOM2 Compare Output Register

**SFR (FF2C<sub>H</sub>/96<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

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

Field	Bits	Type	Description
<b>CCyIO</b> <b>(y=16-31)</b>	y-16	rwh	<b>Compare Output for Channel y</b> Alternative port output for the associated port pin

## 19.2.7 Double-Register Compare Mode Register

**CC2\_DRM**

**CAPCOM2 Double-Register Compare Mode Register**

**SFR (FF2A<sub>H</sub>/95<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>DR7M</b>		<b>DR6M</b>		<b>DR5M</b>		<b>DR4M</b>		<b>DR3M</b>		<b>DR2M</b>		<b>DR1M</b>		<b>DR0M</b>	
rw		rw		rw		rw		rw		rw		rw		rw	

Field	Bits	Type	Description
<b>DRxM</b> <b>(x=0-7)</b>	[2*x+1:2*x]	rw	<b>Double Register x Compare Mode Selection</b> 00 <sub>B</sub> DRM is controlled via the combination of compare modes 1 and 0 (compatibility mode) 01 <sub>B</sub> DRM disabled regardless of compare modes 10 <sub>B</sub> DRM enabled regardless of compare modes 11 <sub>B</sub> Reserved <i>Note: "x" indicates the register pair index in a bank.</i>

## 19.2.8 IOC Register

### CC2\_IOC

**CAPCOM2 I/O Control Register ESFR (F066<sub>H</sub>/33<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

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

Field	Bits	Typ	Description
<b>0</b>	1	rw	<b>Reserved</b> read as '0', do not set this bit
<b>STAG</b>	2	rw	<b>Staggered Mode Control</b> 0 <sub>B</sub> CAPCOM operates in Staggered Mode 1 <sub>B</sub> CAPCOM operates in Non-Staggered Mode
<b>0</b>	3	r	<b>Reserved</b> read as '0', do not set this bit
<b>0</b>	[15:4]	r	<b>Reserved</b> read as '0'



## 19.2.9 Single Event Mode Register

### CC2\_SEM

#### CAPCOM2 Single Event Mode Control Register

**SFR (FE28<sub>H</sub>/14<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

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

Field	Bits	Type	Description
<b>SEMy</b> <b>(y = 16-31)</b>	y-16	rw	<b>Single Event Mode Control</b> 0 <sub>B</sub> Single Event Mode disabled for channel y 1 <sub>B</sub> Single Event Mode enabled for channel y

### CC2\_SEE

#### CAPCOM2 Single Event Enable Register

**SFR (FE2A<sub>H</sub>/15<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

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

Field	Bits	Type	Description
<b>SEEy</b> <b>(y = 16-31)</b>	y-16	rwh	<b>Single Event Enable Control</b> 0 <sub>B</sub> Single Event disabled for channel y 1 <sub>B</sub> Single Event enabled for channel y <i>Note: This bit is cleared by hardware after the event has occurred.</i>

## 19.2.10 KSCCFG Register

### CC2\_KSCCFG

#### CAPCOM2 Kernel State Configuration Register

**SFR(FE24<sub>H</sub>/12<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>BP COM</b>	<b>0</b>	<b>COMCFG</b>	<b>BP SUM</b>	<b>0</b>	<b>SUMCFG</b>	<b>BP NOM</b>	<b>0</b>	<b>NOMCFG</b>	<b>0</b>	<b>BP MOD EN</b>	<b>MOD EN</b>				
w	r	rw	w	r	rw	w	r	rw		r	w	rw			

Field	Bits	Type	Description
<b>MODEN</b>	0	rw	<p><b>Module Enable</b></p> <p>This bit enables the module kernel clock and the module functionality.</p> <p>0<sub>B</sub> The module is switched off. It does not react on mode control actions and the module clock is switched off immediately (without stop condition). The module does not react on read accesses and ignores write accesses.</p> <p>1<sub>B</sub> 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 CAPCOM2 registers.</p> <p><i>Note: This bit is reset by an application reset.</i></p>
<b>BPMODEN</b>	1	w	<p><b>Bit Protection for MODEN</b></p> <p>This bit enables the write access to the bit MODEN. It always reads 0. It is only active during the write access cycle.</p> <p>0<sub>B</sub> MODEN is not changed.</p> <p>1<sub>B</sub> MODEN is updated with the written value.</p> <p><i>Note: This bit is reset by an application reset.</i></p>

<b>Field</b>	<b>Bits</b>	<b>Type</b>	<b>Description</b>
<b>NOMCFG</b>	[5:4]	rw	<b>Normal Operation Mode Configuration</b> This bit field defines the kernel mode applied in normal operation mode. 0X <sub>B</sub> The module is switched on. 1X <sub>B</sub> The module is switched off. This field is taken into account for CR = 00 or 11. <i>Note: This bit is reset by an application reset.</i>
<b>BPNOM</b>	7	w	<b>Bit Protection for NOMCFG</b> This bit enables the write access to the bit field NOMCFG. It always reads 0. It is only active during the write access cycle. 0 <sub>B</sub> NOMCFG is not changed. 1 <sub>B</sub> NOMCFG is updated with the written value. <i>Note: This bit is reset by an application reset.</i>
<b>SUMCFG</b>	[9:8]	rw	<b>Suspend Mode Configuration</b> This bit field defines the kernel mode applied in suspend mode. 0X <sub>B</sub> The module is switched on. 1X <sub>B</sub> The module is switched off. This field is taken into account for CR = 01. <i>Note: This bit is reset by a debug reset.</i>
<b>BPSUM</b>	11	w	<b>Bit Protection for SUMCFG</b> This bit enables the write access to the bit field SUMCFG. It always reads 0. It is only active during the write access cycle. 0 <sub>B</sub> SUMCFG is not changed. 1 <sub>B</sub> SUMCFG is updated with the written value. <i>Note: This bit is reset by a debug reset.</i>
<b>COMCFG</b>	[13:12]	rw	<b>Clock Off Mode Configuration</b> This bit field defines the kernel mode applied in clock off mode. 0X <sub>B</sub> The module is switched on. 1X <sub>B</sub> The module is switched off. This field is taken into account for CR = 10. <i>Note: This bit is reset by an application reset.</i>

Field	Bits	Type	Description
<b>BPCOM</b>	15	w	<b>Bit Protection for COMCFG</b> This bit enables the write access to the bit field COMCFG. It always reads 0. It is only active during the write access cycle. 0 <sub>B</sub> COMCFG is not changed. 1 <sub>B</sub> COMCFG is updated with the written value. <i>Note: This bit is reset by an application reset.</i>
<b>0</b>	[3:2], 6, 10, 14	r	<b>Reserved;</b> returns 0 if read; should be written with 0;

## **19.3 Module Implementation**

This section describes the connections of the CAPCOM unit to its environment.

### **19.3.1 Interfaces of the CAPCOM2 Unit**

The CAPCOM2 unit is connected to its environment in different ways. These connections are summarized in [Table 19-6](#).

#### **Internal Connections**

The overflow/underflow signal T6OFL of GPT2 timer T6 is connected to the CAPCOM2 input T6OUF, providing an optional clock source for the CAPCOM timers.

Synchronous starting is supported by bit SCU\_SYSCON1.GLCCST.

Compare output signals can trigger A/D conversions, trigger serial transmissions (USIC), and generate request signals for the external request unit (ERU).

The 18 interrupt request lines of the CAPCOM2 unit are connected to the interrupt control block. The channel interrupt request lines share interrupt nodes with other sources. The selection is done using register SCU\_ISSR.

The CAPCOM2 module is clocked with the XE16xyM system clock, so  $f_{CC} = f_{SYS}$ .

#### **External Connections**

Sixteen (twelve in 100-pin package) capture/compare signals of the CAPCOM2 unit are connected with input/output ports of the XE16xyM. Depending on the selected direction, these ports may accept capture trigger signals from the external system or issue compare output signals to external circuitry.

*Note: Capture trigger signals may also be derived from output pins. In this case, software can generate the trigger edges, for example.*

Timer T7 can be clocked by an external signal.

**Table 19-6 CAPCOM2 Connections in XE16xyM**

<b>Signal</b>	<b>from/to Module</b>	<b>I/O to CAPCOM2</b>	<b>Can be used to/as</b>
T7IN	P5.9	I	Timer 7 input from port
T6OUF	T6OFL (GPT12)	I	GPT12 timer T6 overflow
T7HR	GLCCST (SCU)	I	Global CAPCOM start
T8HR	GLCCST (SCU)	I	Global CAPCOM start

**Table 19-6 CAPCOM2 Connections in XE16xyM (cont'd)**

<b>Signal</b>	<b>from/to Module</b>	<b>I/O to CAPCOM2</b>	<b>Can be used to/as</b>
CC16IO	P2.3	I	Capture/Compare input
	P2.3 ADC0_REQTR0A	O	Capture/Compare output ADC0 request trigger
CC17IO	P2.4	I	Capture/Compare input
	P2.4 ADC0_REQTR1A	O	Capture/Compare output ADC0 request trigger
CC18IO	P2.5	I	Capture/Compare input
	P2.5 ADC0_REQTR2A	O	Capture/Compare output ADC0 request trigger
CC19IO	P2.6	I	Capture/Compare input
	P2.6	O	Capture/Compare output
CC20IO	P2.7	I	Capture/Compare input
	P2.7	O	Capture/Compare output
CC21IO	P2.8	I	Capture/Compare input
	P2.8	O	Capture/Compare output
CC22IO	P2.9	I	Capture/Compare input
	P2.9	O	Capture/Compare output
CC23IO	P2.10	I	Capture/Compare input
	P2.10	O	Capture/Compare output
CC24IO	P4.0	I	Capture/Compare input
	P4.0 U0C0_DX2E ADC1_REQTR0A	O	Capture/Compare output USIC0 Channel 0 time slot ADC1 request trigger
CC25IO	P4.1	I	Capture/Compare input
	P4.1 U1C0_DX2E ADC1_REQTR1A	O	Capture/Compare output USIC1 Channel 0 time slot ADC1 request trigger
CC26IO	P4.2	I	Capture/Compare input
	P4.2 U2C0_DX2E ADC1_REQTR2A	O	Capture/Compare output USIC2 Channel 0 time slot ADC1 request trigger

**Table 19-6 CAPCOM2 Connections in XE16xyM (cont'd)**

<b>Signal</b>	<b>from/to Module</b>	<b>I/O to CAPCOM2</b>	<b>Can be used to/as</b>
CC27IO	P4.3	I	Capture/Compare input
	P4.3 U3C0_DX2E	O	Capture/Compare output USIC3 Channel 0 time slot

## **20 Capture/Compare Unit 6 (CCU6)**

The CCU6 is a high-resolution 16-bit capture and compare unit with application specific modes, mainly for AC drive control. Special operating modes support the control of Brushless DC-motors using Hall sensors or Back-EMF detection. Furthermore, block commutation and control mechanisms for multi-phase machines are supported. It also supports inputs to start several timers synchronously, an important feature in devices with several CCU6 modules.

This chapter is structured as follows:

- Introduction (see [Section 20.1](#))  
including the register overview (see [Section 20.1.3](#))
- Operating T12 (see [Section 20.2](#))  
including T12 related registers (see [Section 20.2.8](#))  
and capture/compare control registers (see [Section 20.2.9](#))
- Operating T13 (see [Section 20.3](#))  
including T13 related registers (see [Section 20.3.6](#))
- Trap handling (see [Section 20.4](#))
- Multi-Channel mode (see [Section 20.5](#))
- Hall sensor mode (see [Section 20.6](#))
- Modulation control registers (see [Section 20.7](#))
- Interrupt handling (see [Section 20.8](#))  
including interrupt registers (see [Section 20.8.2](#))
- General module operation (see [Section 20.9](#))  
including general registers (see [Section 20.9.3](#))
- Module implementation (see [Section 20.10](#))

### **20.1 Introduction**

The CCU6 unit is made up of a Timer T12 Block with three capture/compare channels and a Timer T13 Block with one compare channel. The T12 channels can independently generate PWM signals or accept capture triggers, or they can jointly generate control signal patterns to drive AC-motors or inverters.

A rich set of status bits, synchronized updating of parameter values via shadow registers, and flexible generation of interrupt request signals provide means for efficient software-control.

*Note: The capture/compare module itself is named CCU6 (capture/compare unit 6).  
A capture/compare channel inside this module is named CC6x.*



## **20.1.1 Feature Set Overview**

This section gives an overview over the different building blocks and their main features.

### **Timer 12 Block Features**

- Three capture/compare channels, each channel can be used either as capture or as compare channel
- Generation of a three-phase PWM supported (six outputs, individual signals for high-side and low-side switches)
- 16-bit resolution, maximum count frequency = peripheral clock
- Dead-time control for each channel to avoid short-circuits in the power stage
- Concurrent update of T12 registers
- Center-aligned and edge-aligned PWM can be generated
- Single-shot mode supported
- Start can be controlled by external events
- Capability of counting external events
- Many interrupt request sources
- Hysteresis-like control mode

### **Timer 13 Block Features**

- One independent compare channel with one output
- 16-bit resolution, maximum count frequency = peripheral clock
- Concurrent update of T13 registers
- Can be synchronized to T12
- Interrupt generation at period-match and compare-match
- Single-shot mode supported
- Start can be controlled by external events
- Capability of counting external events

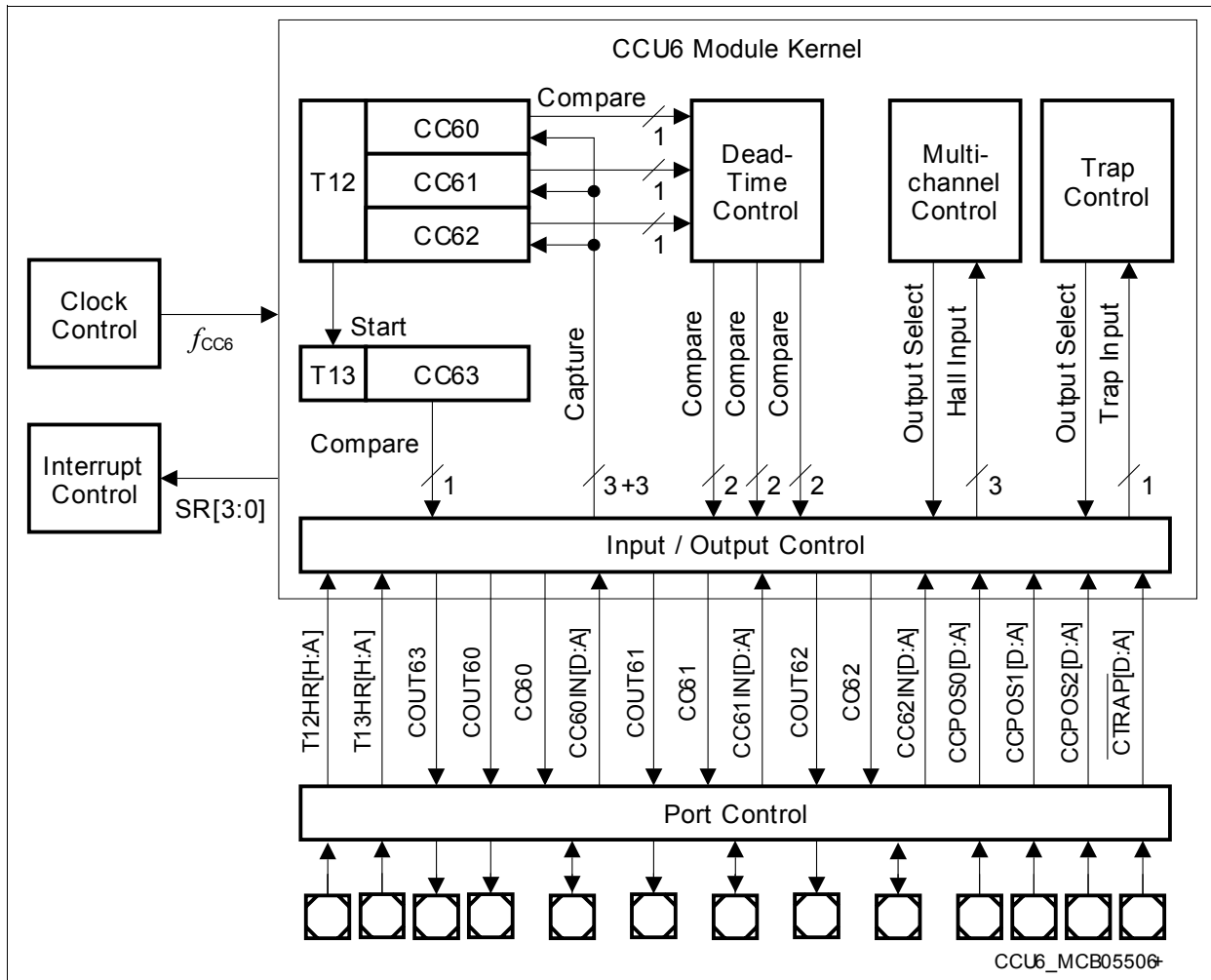
### **Additional Specific Functions**

- Block commutation for Brushless DC-drives implemented
- Position detection via Hall-sensor pattern
- Noise filter supported for position input signals
- Automatic rotational speed measurement and commutation control for block commutation
- Integrated error handling
- Fast emergency stop without CPU load via external signal ( $\overline{\text{CTRAP}}$ )
- Control modes for multi-channel AC-drives
- Output levels can be selected and adapted to the power stage

**Capture/Compare Unit 6 (CCU6)**

**20.1.2 Block Diagram**

The Timer T12 can operate in capture and/or compare mode for its three channels. The modes can also be combined (e.g. a channel operates in compare mode, whereas another channel operates in capture mode). The Timer T13 can operate in compare mode only. The multi-channel control unit generates output patterns which can be modulated by T12 and/or T13. The modulation sources can be selected and combined for the signal modulation.



**Figure 20-1 CCU6 Block Diagram**

### 20.1.3 Register Overview

For the generation of the overall register table, the prefix “CCU6x\_” has to be added to the register names in this table to identify the registers of different CCU6 modules that are implemented. In this naming convention, x indicates the module number.

**Table 20-1** shows all registers required for programming of a CCU6 module. It summarizes the CCU6 kernel registers and defines the offset and the reset values. 8-bit short addresses are not available for this module.

T12 related Registers	Cap/Com Control Registers	Interrupt Status/ Control Registers	General Registers
<b>T12</b>	<b>CMPSTAT</b>	<b>IS</b>	<b>KSCFG</b>
<b>T12PR</b>	<b>CMPMODIF</b>	<b>ISS</b>	<b>KSCSR</b>
<b>T12DTC</b>	<b>T12MSEL</b>	<b>ISR</b>	<b>PISELH</b>
<b>CC60R</b>	<b>TCTR0</b>	<b>INP</b>	<b>PISELL</b>
<b>CC60SR</b>	<b>TCTR2</b>	<b>IEN</b>	<b>ID</b>
<b>CC61R</b>	<b>TCTR4</b>	<b>0IC</b>	
<b>CC61SR</b>		<b>1IC</b>	
<b>CC62R</b>		<b>2IC</b>	
<b>CC62SR</b>		<b>3IC</b>	
	Modulation Control Registers		
	<b>MODCTR</b>		
	<b>TRPCTR</b>		
	<b>PSLR</b>		
	<b>MCMCTR</b>		
	<b>MCMOUTS</b>		
	<b>MCMOUT</b>		
T13 related Registers			
<b>T13</b>			
<b>T13PR</b>			
<b>CC63R</b>			
<b>CC63SR</b>			

CCU6\_regs2

**Figure 20-2 CCU6 Registers**

**Table 20-1 CCU6 Module Register Summary**

Short Name	Description	Offset	Reset Value	See Page
------------	-------------	--------	-------------	----------

**General Registers**

<b>ID</b>	Module Identification Register	08 <sub>H</sub>	54XX <sub>H</sub>	<a href="#">Page 20-107</a>
<b>PISELL</b>	Module Port Input Select Register	04 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 20-107</a>
<b>PISELH</b>	Module Port Input Select Register	06 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 20-109</a>
<b>KSCFG</b>	Kernel State Configuration Register	00 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 20-112</a>
<b>KSCSR</b>	Kernel State Control Sensitivity Register	0E <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 20-114</a>

**Timer T12 related Registers**

<b>T12</b>	Timer 12 Counter Register	10 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 20-32</a>
<b>T12PR</b>	Timer 12 Period Register	12 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 20-32</a>
<b>T12DTC</b>	Dead-Time Control Register for Timer T12	14 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 20-35</a>
<b>CC60R</b>	Capture/Compare Register Channel CC60	18 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 20-33</a>
<b>CC61R</b>	Capture/Compare Register Channel CC61	1A <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 20-33</a>
<b>CC62R</b>	Capture/Compare Register Channel CC62	1C <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 20-33</a>
<b>CC60SR</b>	Capture/Compare Shadow Register Channel CC60	20 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 20-34</a>
<b>CC61SR</b>	Capture/Compare Shadow Register Channel CC61	22 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 20-34</a>
<b>CC62SR</b>	Capture/Compare Shadow Register Channel CC62	24 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 20-34</a>

**Capture/Compare Control Registers**

<b>CMPSTAT</b>	Compare State Register	28 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 20-37</a>
<b>CMPMODIF</b>	Compare State Modification Register	2A <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 20-39</a>
<b>T12MSEL</b>	T12 Capture/Compare Mode Select Register	46 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 20-40</a>
<b>TCTR0</b>	Timer Control Register 0	2C <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 20-41</a>

**Table 20-1 CCU6 Module Register Summary (cont'd)**

<b>Short Name</b>	<b>Description</b>	<b>Offset</b>	<b>Reset Value</b>	<b>See Page</b>
<b>TCTR2</b>	Timer Control Register 2	2E <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 20-44</a>
<b>TCTR4</b>	Timer Control Register 4	26 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 20-47</a>

**Timer T13 related Registers**

<b>T13</b>	Timer 13 Counter Register	30 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 20-62</a>
<b>T13PR</b>	Timer 13 Period Register	32 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 20-63</a>
<b>CC63R</b>	Compare Register for Timer 13	34 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 20-64</a>
<b>CC63SR</b>	Compare Shadow Register for Timer 13	36 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 20-64</a>

**Modulation Control Registers**

<b>MODCTR</b>	Modulation Control Register	40 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 20-78</a>
<b>TRPCTR</b>	Trap Control Register	42 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 20-80</a>
<b>PSLR</b>	Passive State Level Register	44 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 20-83</a>
<b>MCMOUTS</b>	Multi-Channel Mode Output Shadow Register	4A <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 20-86</a>
<b>MCMOUT</b>	Multi-Channel Mode Output Register	4C <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 20-87</a>
<b>MCMCTR</b>	Multi-Channel Mode Control Register	4E <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 20-84</a>

**Interrupt Status and Node Registers**

<b>IS</b>	Interrupt Status Register	50 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 20-91</a>
<b>ISS</b>	Interrupt Status Set Register	52 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 20-94</a>
<b>ISR</b>	Interrupt Status Reset Register	54 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 20-96</a>
<b>INP</b>	Interrupt Node Pointer Register	56 <sub>H</sub>	3940 <sub>H</sub>	<a href="#">Page 20-101</a>
<b>IEN</b>	Interrupt Node Pointer Register	58 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 20-98</a>

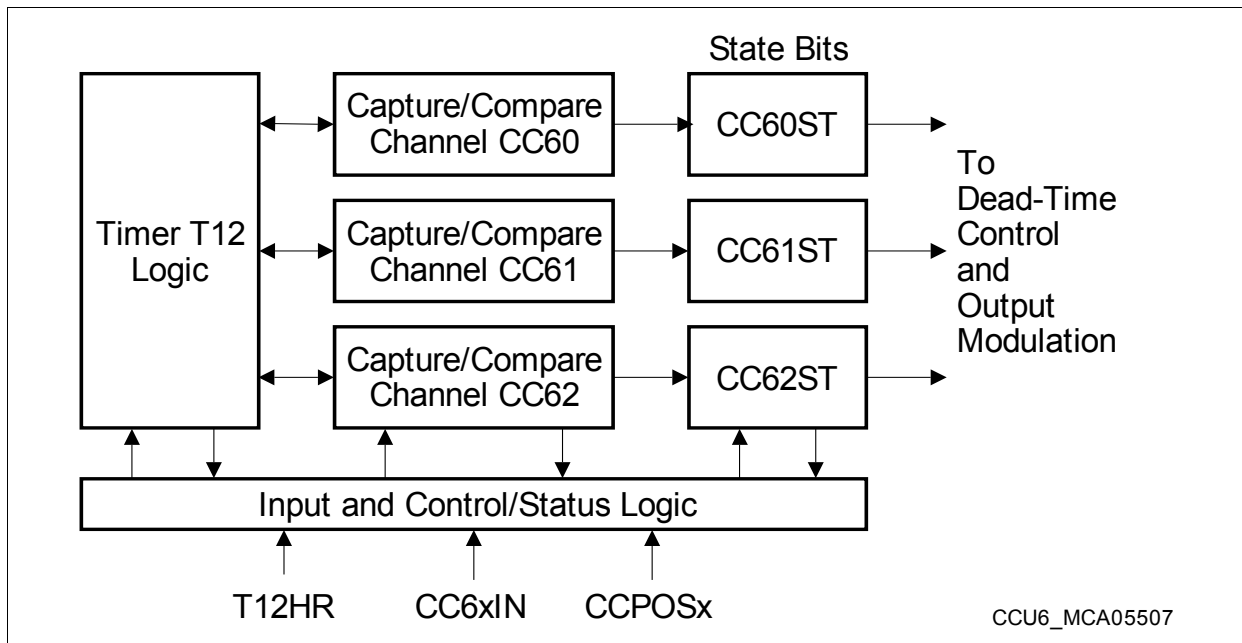
*Note: In the case of a write access to addresses inside the address range (that is covered by the same chip select signal), but that are not the addresses explicitly mentioned for the module, the write access is not taken into account for the module. The same principle is valid for read accesses. In case of a read access to another address, the module does not react.*

## 20.2 Operating Timer T12

The timer T12 block is the main unit to generate the 3-phase PWM signals. A 16-bit counter is connected to 3 channel registers via comparators, that generate a signal when the counter contents match one of the channel register contents. A variety of control functions facilitate the adaptation of the T12 structure to different application needs. Besides the 3-phase PWM generation, the T12 block offers options for individual compare and capture functions, as well as dead-time control and hysteresis-like compare mode.

This section provides information about:

- T12 overview (see [Section 20.2.1](#))
- Counting scheme (see [Section 20.2.2](#))
- Compare modes (see [Section 20.2.3](#))
- Compare mode output path (see [Section 20.2.4](#))
- Capture modes (see [Section 20.2.5](#))
- Shadow transfer (see [Section 20.2.6](#))
- T12 operating mode selection (see [Section 20.2.7](#))
- T12 counter register description (see [Section 20.2.8](#))

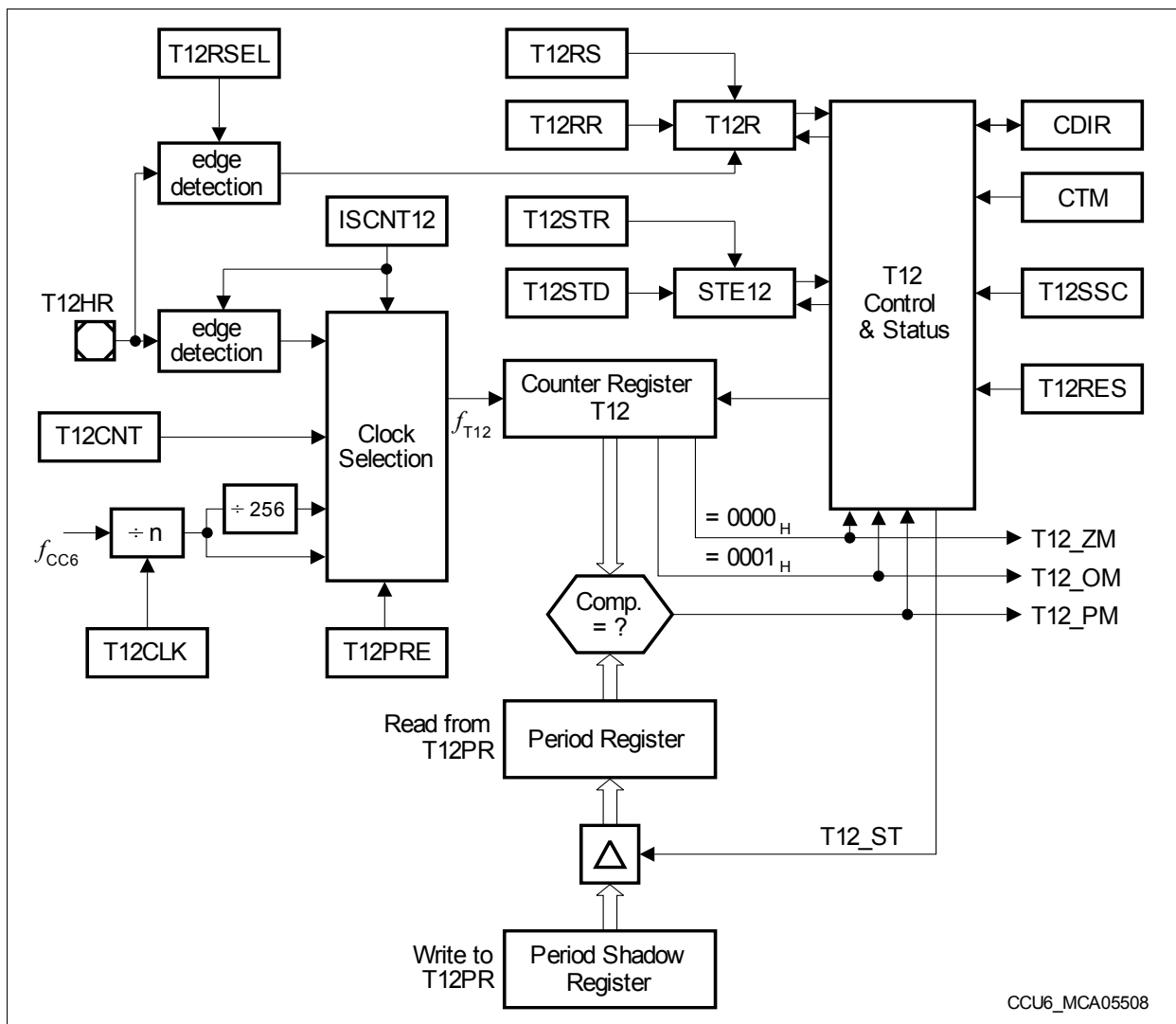


**Figure 20-3 Overview Diagram of the Timer T12 Block**

### 20.2.1 T12 Overview

**Figure 20-4** shows a detailed block diagram of Timer T12. The functions of the timer T12 block are controlled by bits in registers **TCTR0**, **TCTR2**, **TCTR4**, and **PISELL**.

Timer T12 receives its input clock ( $f_{T12}$ ) from the module clock  $f_{CC6}$  via a programmable prescaler and an optional 1/256 divider or from an input signal T12HR. These options are controlled via bit fields T12CLK and T12PRE (see [Table 20-2](#)). T12 can count up or down, depending on the selected operation mode. A direction flag, CDIR, indicates the current counting direction.



### Figure 20-4 Timer T12 Logic and Period Comparators

Via a comparator, the T12 counter register **T12** is connected to a Period Register **T12PR**. This register determines the maximum count value for T12.

In Edge-Aligned mode, T12 is cleared to 0000<sub>H</sub> after it has reached the period value defined by T12PR. In Center-Aligned mode, the count direction of T12 is set from 'up' to

## **Capture/Compare Unit 6 (CCU6)**

‘down’ after it has reached the period value (please note that in this mode, T12 exceeds the period value by one before counting down). In both cases, signal T12\_PM (T12 Period Match) is generated. The Period Register receives a new period value from its Shadow Period Register.

A read access to T12PR delivers the current period value at the comparator, whereas a write access targets the Shadow Period Register to prepare another period value. The transfer of a new period value from the Shadow Period Register into the Period Register (see [Section 20.2.6](#)) is controlled via the ‘T12 Shadow Transfer’ control signal, T12\_ST. The generation of this signal depends on the operating mode and on the shadow transfer enable bit STE12. Providing a shadow register for the period value as well as for other values related to the generation of the PWM signal allows a concurrent update by software for all relevant parameters.

Two further signals indicate whether the counter contents are equal to 0000<sub>H</sub> (T12\_ZM = zero match) or 0001<sub>H</sub> (T12\_OM = one match). These signals control the counting and switching behavior of T12.

The basic operating mode of T12, either Edge-Aligned mode ([Figure 20-5](#)) or Center-Aligned mode ([Figure 20-6](#)), is selected via bit CTM. A Single-Shot control bit, T12SSC, enables an automatic stop of the timer when the current counting period is finished (see [Figure 20-7](#) and [Figure 20-8](#)).

The start or stop of T12 is controlled by the Run bit T12R that can be modified by bits in register **TCTR4**. The run bit can be set/cleared by software via the associated set/clear bits T12RS or T12RR, it can be set by a selectable edge of the input signal T12HR (**TCTR2**.T12RSEL), or it is cleared by hardware according to preselected conditions.

The timer T12 run bit T12R must not be set while the applied T12 period value is zero. Timer T12 can be cleared via control bit T12RES. Setting this write-only bit does only clear the timer contents, but has no further effects, for example, it does not stop the timer.

The generation of the T12 shadow transfer control signal, T12\_ST, is enabled via bit STE12. This bit can be set or reset by software indirectly through its associated set/clear control bits T12STR and T12STD.

While Timer T12 is running, write accesses to the count register T12 are not taken into account. If T12 is stopped and the Dead-Time counters are 0, write actions to register T12 are immediately taken into account.



## 20.2.2 T12 Counting Scheme

This section describes the clocking and counting capabilities of T12.

### 20.2.2.1 Clock Selection

In **Timer Mode** (**PISELH.ISCNT12** = 00<sub>B</sub>), the input clock  $f_{T12}$  of Timer T12 is derived from the internal module clock  $f_{CC6}$  through a programmable prescaler and an optional 1/256 divider. The resulting prescaler factors are listed in **Table 20-2**. The prescaler of T12 is cleared while T12 is not running (**TCTR0.T12R** = 0) to ensure reproducible timings and delays.

**Table 20-2 Timer T12 Input Frequency Options**

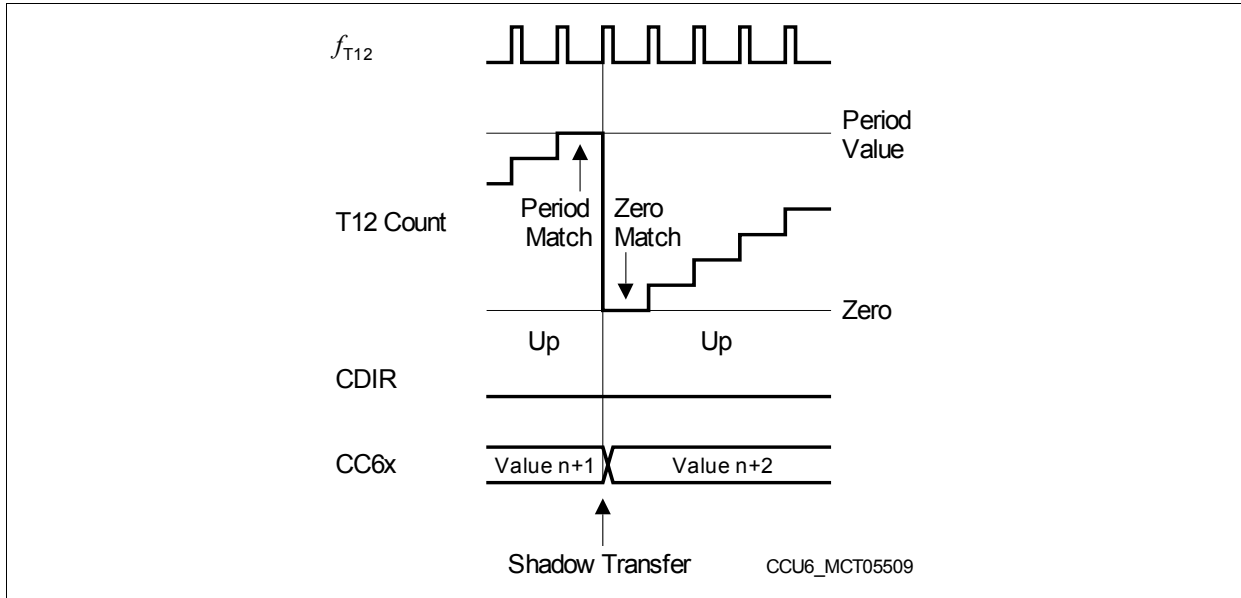
<b>T12CLK</b>	<b>Resulting Input Clock <math>f_{T12}</math> Prescaler Off (T12PRE = 0)</b>	<b>Resulting Input Clock <math>f_{T12}</math> Prescaler On (T12PRE = 1)</b>
000 <sub>B</sub>	$f_{CC6}$	$f_{CC6} / 256$
001 <sub>B</sub>	$f_{CC6} / 2$	$f_{CC6} / 512$
010 <sub>B</sub>	$f_{CC6} / 4$	$f_{CC6} / 1024$
011 <sub>B</sub>	$f_{CC6} / 8$	$f_{CC6} / 2048$
100 <sub>B</sub>	$f_{CC6} / 16$	$f_{CC6} / 4096$
101 <sub>B</sub>	$f_{CC6} / 32$	$f_{CC6} / 8192$
110 <sub>B</sub>	$f_{CC6} / 64$	$f_{CC6} / 16384$
111 <sub>B</sub>	$f_{CC6} / 128$	$f_{CC6} / 32768$

In **Counter Mode**, timer T12 counts one step:

- If a 1 is written to **TCTR4.T12CNT** and **PISELH.ISCNT12** = 01<sub>B</sub>
- If a rising edge of input signal T12HR is detected and **PISELH.ISCNT12** = 10<sub>B</sub>
- If a falling edge of input signal T12HR is detected and **PISELH.ISCNT12** = 11<sub>B</sub>

### 20.2.2.2 Edge-Aligned / Center-Aligned Mode

In **Edge-Aligned Mode** (CTM = 0), timer T12 is always counting upwards (CDIR = 0). When reaching the value given by the period register (period-match T12\_PM), the value of T12 is cleared with the next counting step (saw tooth shape).



**Figure 20-5 T12 Operation in Edge-Aligned Mode**

As a result, in Edge-Aligned mode, the timer period is given by:

$$T12_{PER} = \text{<Period-Value>} + 1; \text{ in } T12 \text{ clocks } (f_{T12}) \quad (20.1)$$

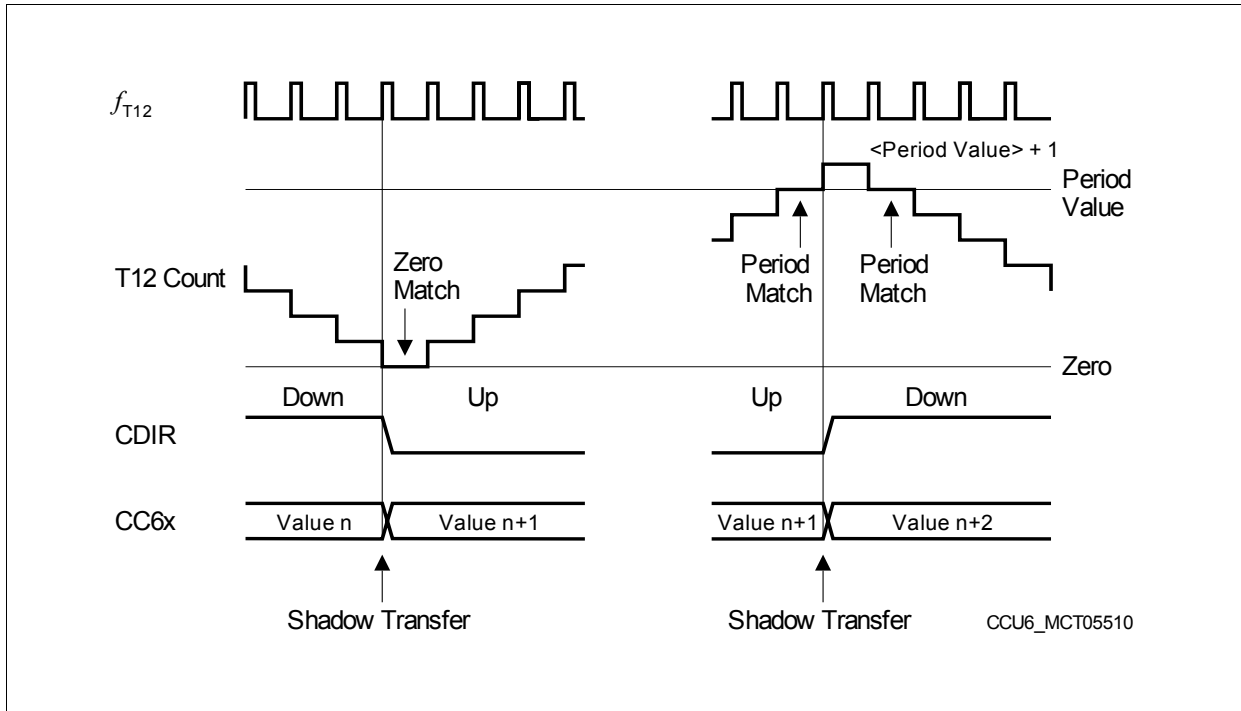
In **Center-Aligned Mode** (CTM = 1), timer T12 is counting upwards or downwards (triangular shape). When reaching the value given by the period register (period-match T12\_PM) while counting upwards (CDIR = 0), the counting direction control bit CDIR is changed to downwards (CDIR = 1) with the next counting step.

When reaching the value 0001<sub>H</sub> (one-match T12\_OM) while counting downwards, the counting direction control bit CDIR is changed to upwards with the next counting step.

As a result, in Center.Aligned mode, the timer period is given by:

$$T12_{PER} = (\text{<Period-Value>} + 1) \times 2; \text{ in } T12 \text{ clocks } (f_{T12}) \quad (20.2)$$

- With the next clock event of  $f_{T12}$  the count direction is set to counting up (CDIR = 0) when the counter reaches 0001<sub>H</sub> while counting down.
- With the next clock event of  $f_{T12}$  the count direction is set to counting down (CDIR = 1) when the Period-Match is detected while counting up.
- With the next clock event of  $f_{T12}$  the counter counts up while CDIR = 0 and it counts down while CDIR = 1.



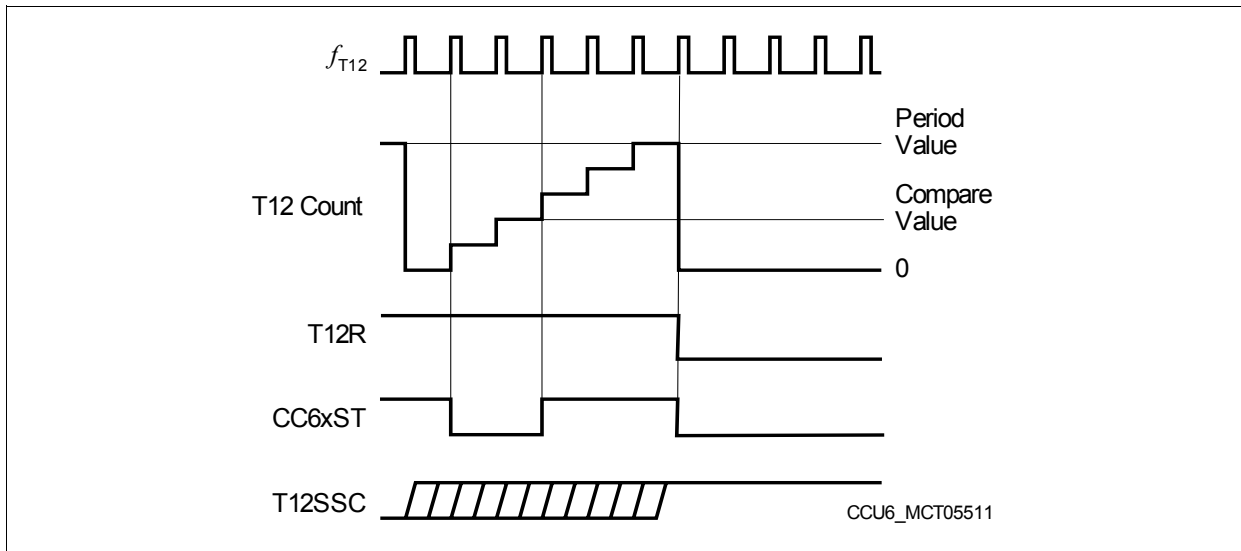
**Figure 20-6 T12 Operation in Center-Aligned Mode**

*Note: Bit CDIR changes with the next timer clock event after the one-match or the period-match. Therefore, the timer continues counting in the previous direction for one cycle before actually changing its direction (see [Figure 20-6](#)).*

### 20.2.2.3 Single-Shot Mode

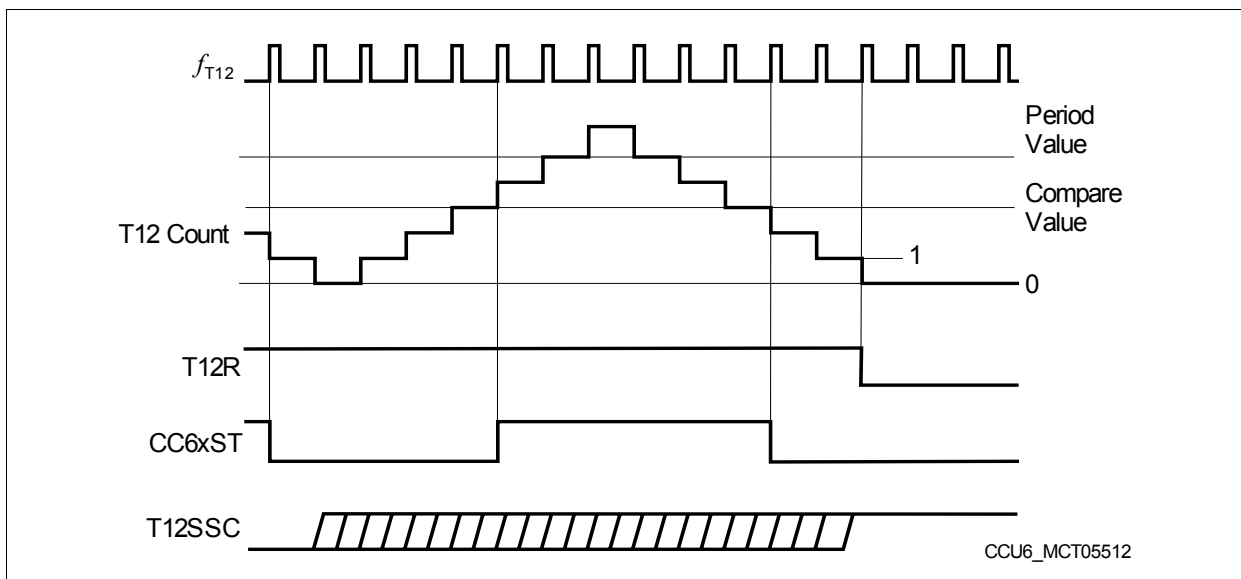
In Single-Shot Mode, the timer run bit T12R is cleared by hardware. If bit T12SSC = 1, the timer T12 will stop when the current timer period is finished.

In Edge-Aligned mode, T12R is cleared when the timer becomes zero after having reached the period value (see [Figure 20-7](#)).



**Figure 20-7 Single-Shot Operation in Edge-Aligned Mode**

In Center-Aligned mode, the period is finished when the timer has counted down to zero (one clock cycle after the one-match while counting down, see [Figure 20-8](#)).



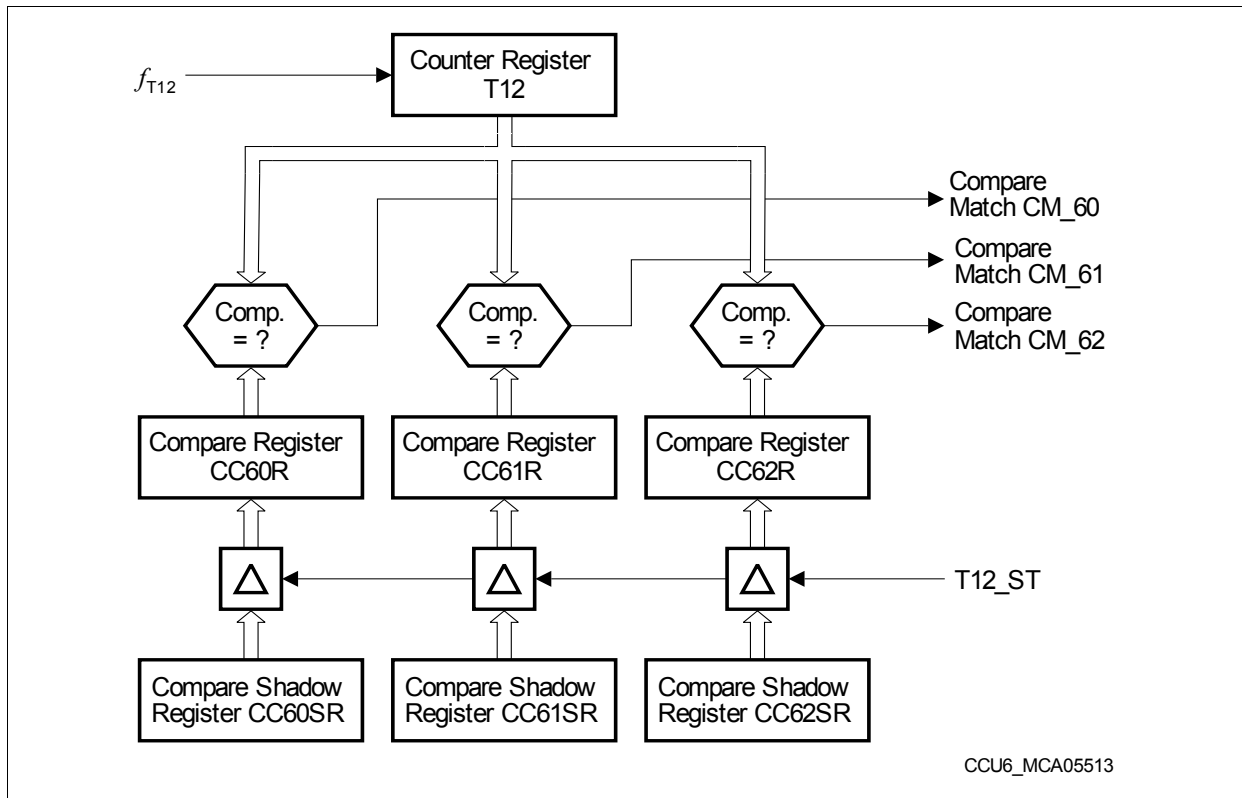
**Figure 20-8 Single-Shot Operation in Center-Aligned Mode**

## 20.2.3 T12 Compare Mode

Associated with Timer T12 are three individual capture/compare channels, that can perform compare or capture operations with regard to the contents of the T12 counter. The capture functions are explained in [Section 20.2.5](#).

### 20.2.3.1 Compare Channels

In Compare Mode (see [Figure 20-9](#)), the three individual compare channels CC60, CC61, and CC62 can generate a three-phase PWM pattern.

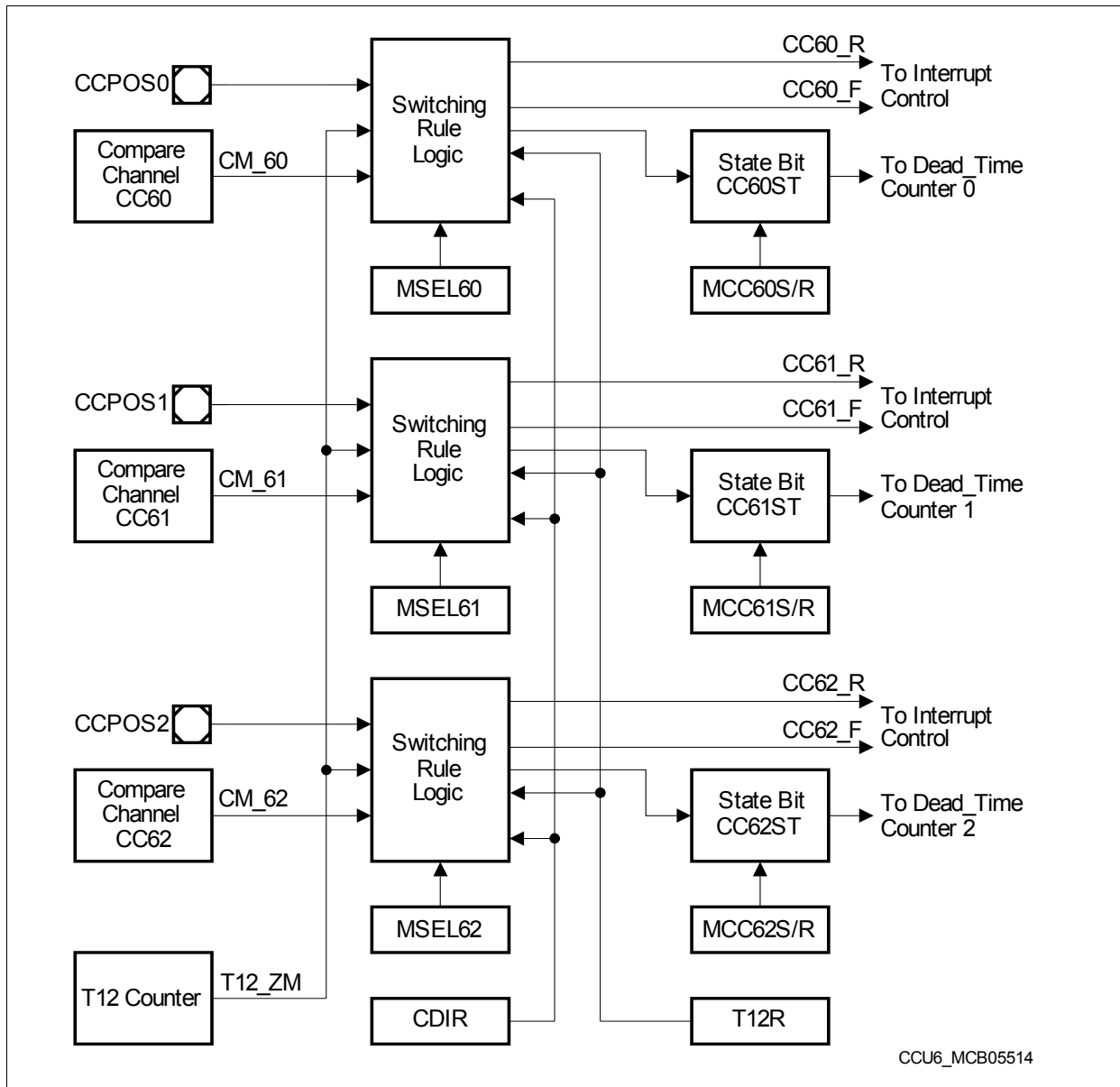


**Figure 20-9 T12 Channel Comparators**

Each compare channel is connected to the T12 counter register via its individual equal-to comparator, generating a match signal when the contents of the counter matches the contents of the associated compare register. Each channel consists of the comparator and a double register structure - the actual compare register CC6xR, feeding the comparator, and an associated shadow register CC6xSR, that is preloaded by software and transferred into the compare register when signal T12 shadow transfer, T12\_ST, gets active. Providing a shadow register for the compare value as well as for other values related to the generation of the PWM signal facilitates a concurrent update by software for all relevant parameters of a three-phase PWM.

### 20.2.3.2 Channel State Bits

Associated with each (compare) channel is a State Bit, **CMPSTAT**.CC6xST, holding the status of the compare (or capture) operation (see [Figure 20-10](#)). In compare mode, the State Bits are modified according to a set of switching rules, depending on the current status of timer T12.



**Figure 20-10 Compare State Bits for Compare Mode**

The inputs to the switching rule logic for the CC6xST bits are the timer direction (CDIR), the timer run bit (T12R), the timer T12 zero-match signal (T12\_ZM), and the actual individual compare-match signals CM\_6x as well as the mode control bits, **T12MSEL**.MSEL6x.

## Capture/Compare Unit 6 (CCU6)

In addition, each state bit can be set or cleared by software via the appropriate set and reset bits in register **CMPMODIF**, MCC6xS and MCC6xR. The input signals CCPOSx are used in hysteresis-like compare mode, whereas in normal compare mode, these inputs are ignored.

*Note: In Hall Sensor, single shot or capture modes, additional/different rules are taken into account (see related sections).*

A compare interrupt event CC6x\_R is signaled when a compare match is detected while counting upwards, whereas the compare interrupt event CC6x\_F is signaled when a compare match is detected while counting down. The actual setting of a State Bit has no influence on the interrupt generation in compare mode.

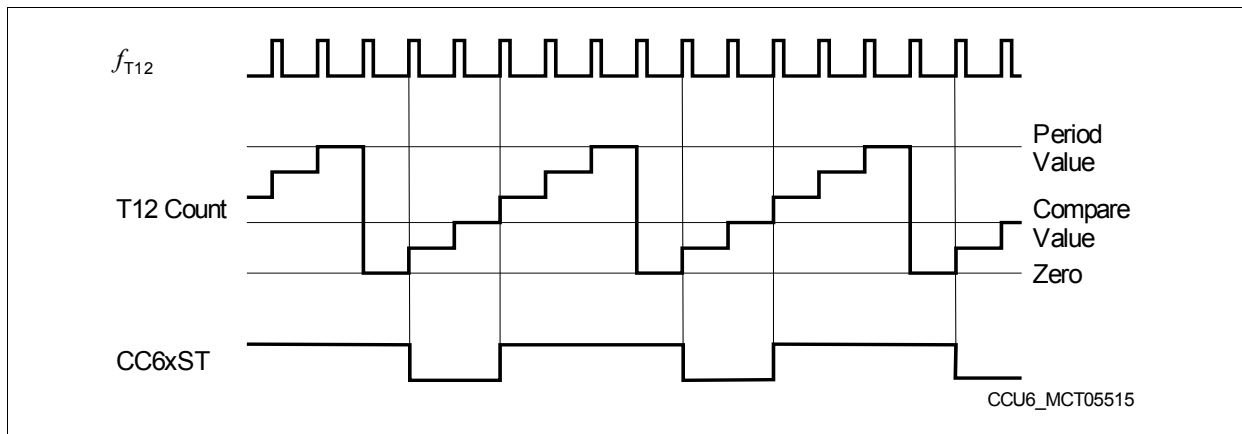
A modification of a State Bit CC6xST by the switching rule logic due to a compare action is only possible while Timer T12 is running (T12R = 1). If this is the case, the following switching rules apply for setting and clearing the State Bits in Compare Mode (illustrated in **Figure 20-11** and **Figure 20-12**):

A State Bit **CC6xST** is **set** to 1:

- with the next T12 clock ( $f_{T12}$ ) after a compare-match when T12 is counting up (i.e., when the counter is incremented above the compare value);
- with the next T12 clock ( $f_{T12}$ ) after a zero-match AND a parallel compare-match when T12 is counting up.

A State Bit **CC6xST** is **cleared** to 0:

- with the next T12 clock ( $f_{T12}$ ) after a compare-match when T12 is counting down (i.e., when the counter is decremented below the compare value in center-aligned mode);
- with the next T12 clock ( $f_{T12}$ ) after a zero-match AND NO parallel compare-match when T12 is counting up.

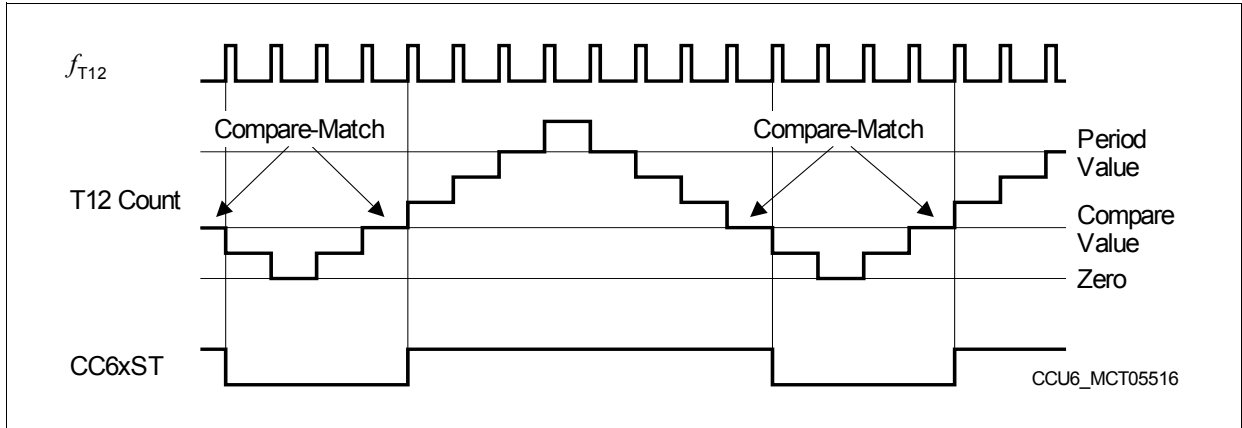


**Figure 20-11 Compare Operation, Edge-Aligned Mode**

**Figure 20-13** illustrates some more examples for compare waveforms. It is important to note that in these examples, it is assumed that some of the compare values are changed

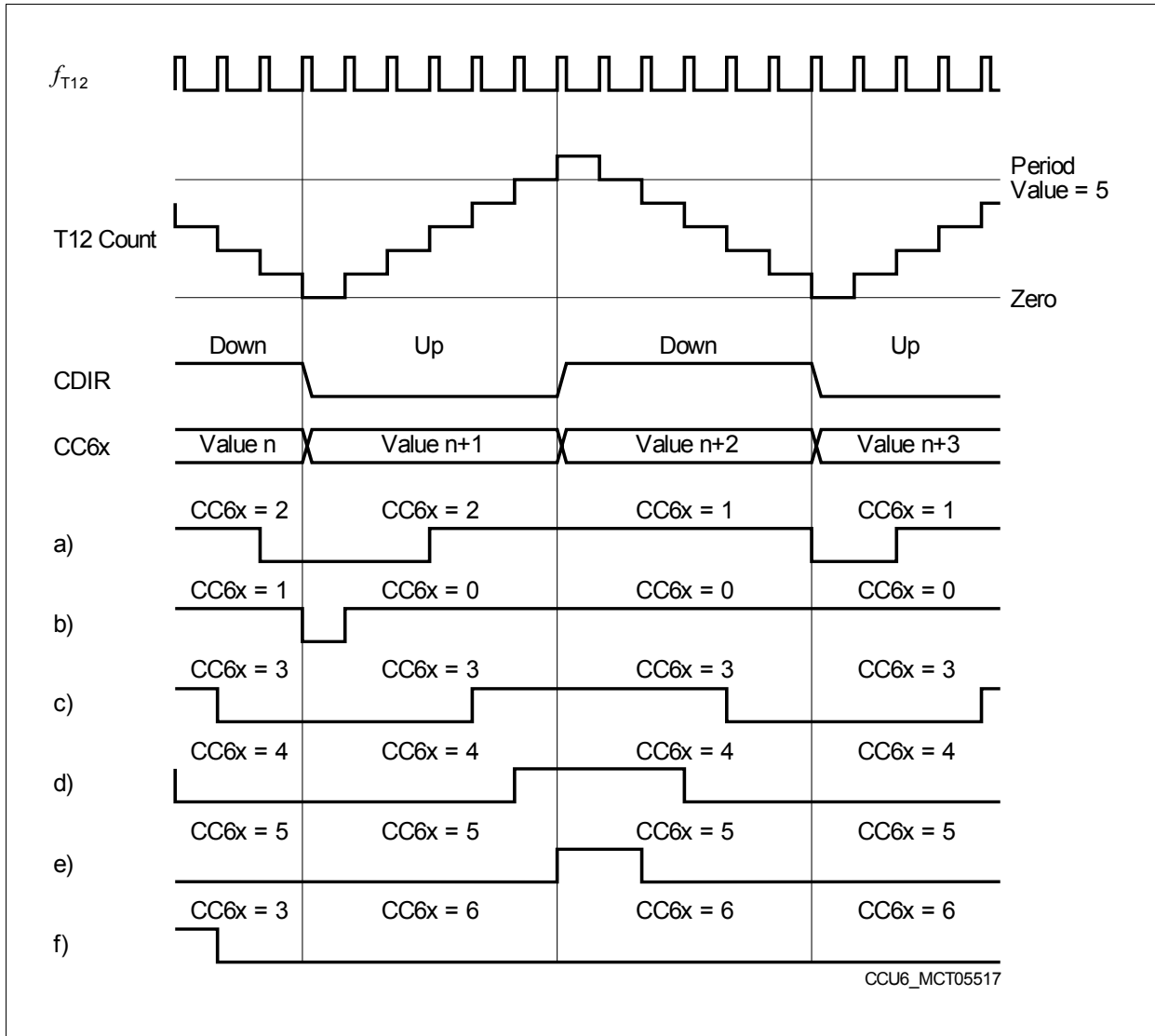
**Capture/Compare Unit 6 (CCU6)**

while the timer is running. This change is performed via a software preload of the Shadow Register, CC6xSR. The value is transferred to the actual Compare Register CC6xR with the T12 Shadow Transfer signal, T12\_ST, that is assumed to be enabled.



**Figure 20-12 Compare Operation, Center-Aligned Mode**



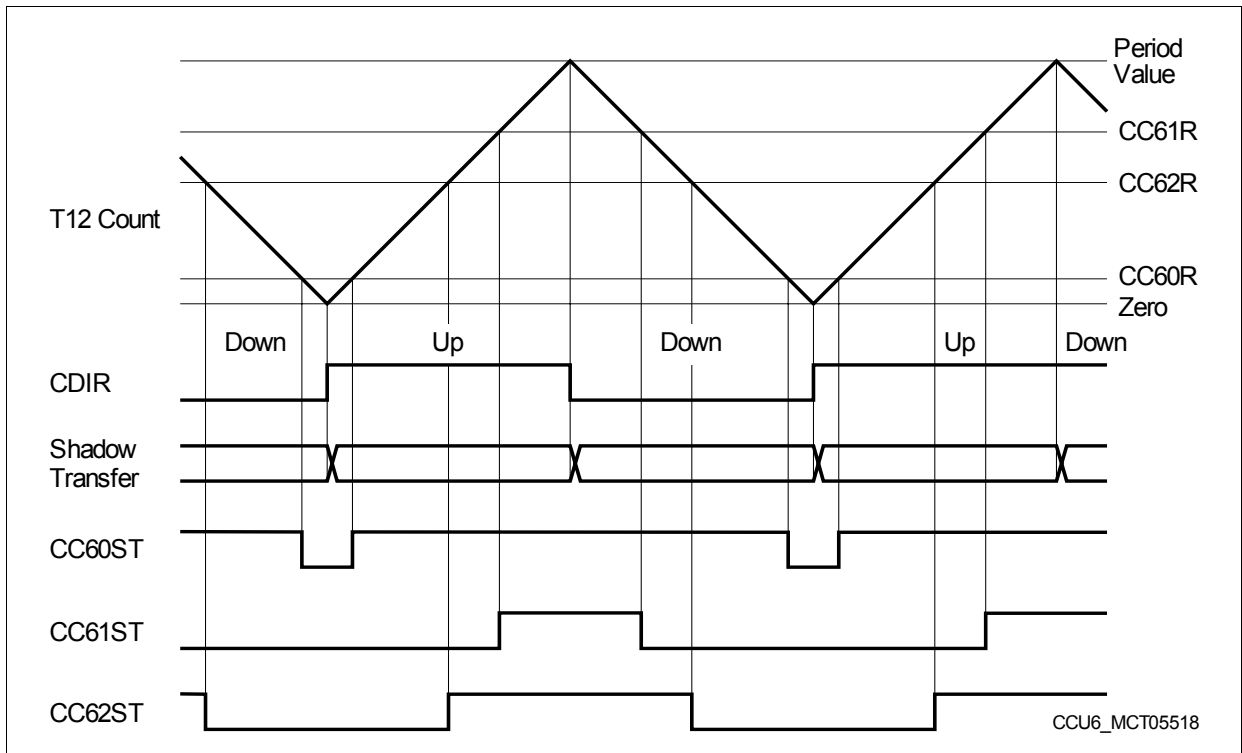


**Figure 20-13 Compare Waveform Examples**

Example b) illustrates the transition to a duty cycle of 100%. First, a compare value of  $0001_H$  is used, then changed to  $0000_H$ . Please note that a low pulse with the length of one T12 clock is still produced in the cycle where the new value  $0000_H$  is in effect; this pulse originates from the previous value  $0001_H$ . In the following timer cycles, the State Bit CC6xST remains at 1, producing a 100% duty cycle signal. In this case, the compare rule 'zero-match AND compare-match' is in effect.

Example f) shows the transition to a duty cycle of 0%. The new compare value is set to  $\langle \text{Period-Value} \rangle + 1$ , and the State Bit CC6ST remains cleared.

**Figure 20-14** illustrates an example for the waveforms of all three channels. With the appropriate dead-time control and output modulation, a very efficient 3-phase PWM signal can be generated.



**Figure 20-14 Three-Channel Compare Waveforms**

### **20.2.3.3 Hysteresis-Like Control Mode**

The hysteresis-like control mode (**T12MSEL**.MSEL6x = 1001<sub>B</sub>) offers the possibility to switch off the PWM output if the input CCPOSx becomes 0 by clearing the State Bit CC6xST. This can be used as a simple motor control feature by using a comparator indicating, e.g., overcurrent. While CCPOSx = 0, the PWM outputs of the corresponding channel are driving their passive levels, because the setting of bit CC6xST is only possible while CCPOSx = 1.

As long as input CCPOSx is 0, the corresponding State Bit is held 0. When CCPOSx is at high level, the outputs can be in active state and are determined by bit CC6xST (see **Figure 20-10** for the state bit logic and **Figure 20-15** for the output paths).

The CCPOSx inputs are evaluated with  $f_{CC6}$ .

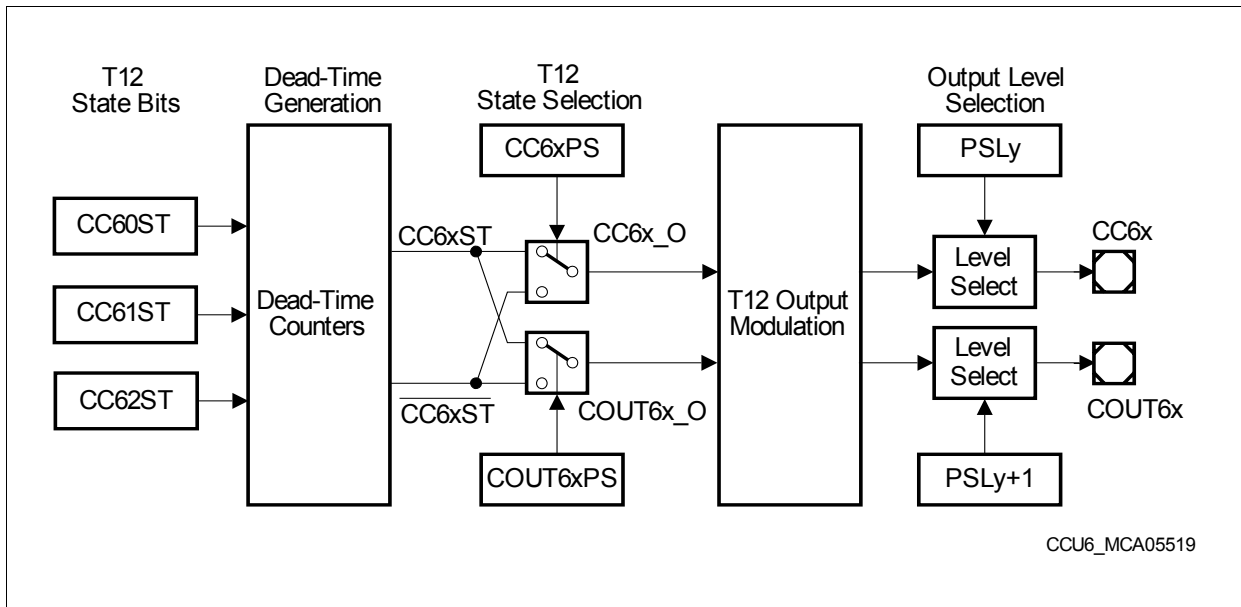
This mode can be used to introduce a timing-related behavior to a hysteresis controller. A standard hysteresis controller detects if a value exceeds a limit and switches its output according to the compare result. Depending on the operating conditions, the switching frequency and the duty cycle are not fixed, but change permanently.

If (outer) time-related control loops based on a hysteresis controller in an inner loop should be implemented, the outer loops show a better behavior if they are synchronized to the inner loops. Therefore, the hysteresis-like mode can be used, that combines timer-related switching with a hysteresis controller behavior. For example, in this mode, an output can be switched on according to a fixed time base, but it is switched off as soon as a falling edge is detected at input CCPOSx.

This mode can also be used for standard PWM with overcurrent protection. As long as there is no low level signal at pin CCPOSx, the output signals are generated in the normal manner as described in the previous sections. Only if input CCPOSx shows a low level, e.g. due to the detection of overcurrent, the outputs are shut off to avoid harmful stress to the system.

## 20.2.4 Compare Mode Output Path

**Figure 20-15** gives an overview on the signal path from a channel State Bit to its output pin in its simplest form. As illustrated, a user has a variety of controls to determine the desired output signal switching behavior in relation to the current state of the State Bit, CC6xST. Please refer to **Section 20.2.4.3** for details on the output modulation.



**Figure 20-15 Compare Mode Simplified Output Path Diagram**

The output path is based on signals that are defined as active or passive. The terms active and passive are not related to output levels, but to internal actions. This mainly applies for the modulation, where T12 and T13 signals are combined with the multi-channel signals and the trap function. The Output level Selection allows the user to define the output level at the output pin for the passive state (inverted level for the active state). It is recommended to configure this block in a way that an external power switch is switched off while the CCU6 delivers an output signal in the passive state.

### 20.2.4.1 Dead-Time Generation

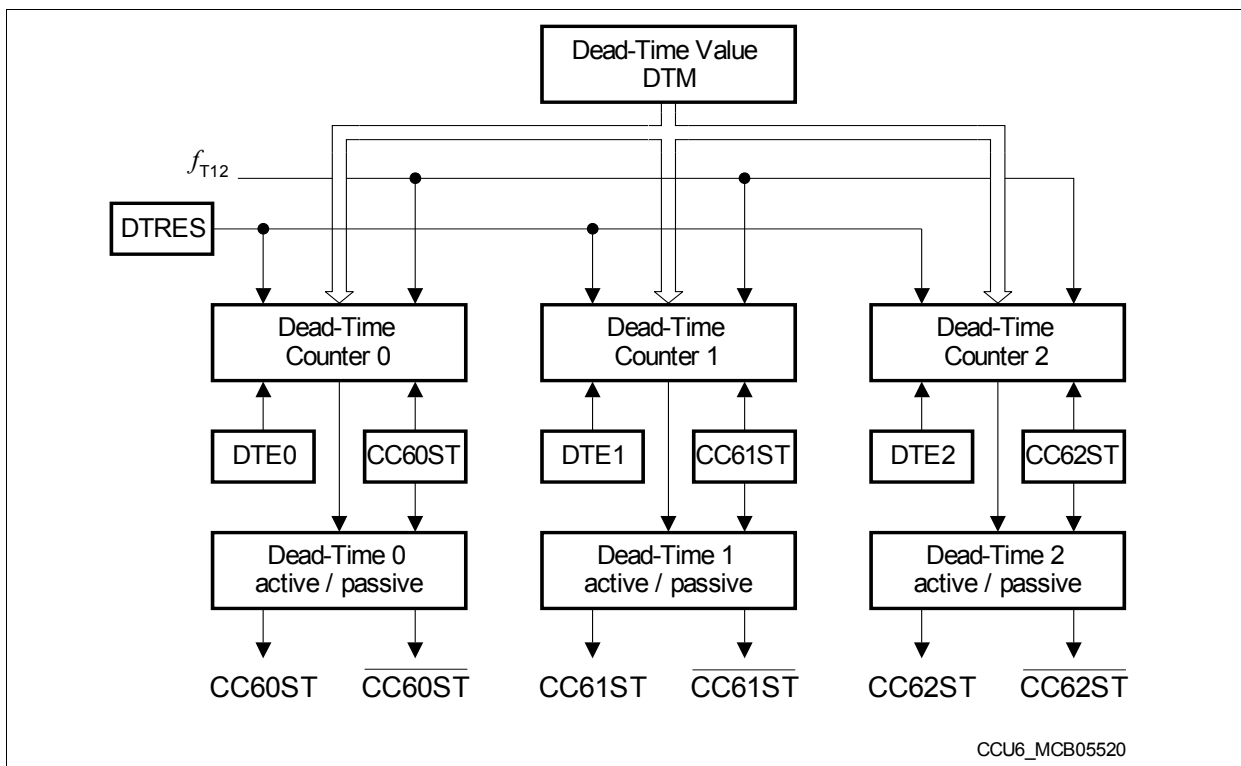
The generation of (complementary) signals for the high-side and the low-side switches of one power inverter phase is based on the same compare channel. For example, if the high-side switch should be active while the T12 counter value is above the compare value (State Bit = 1), then the low-side switch should be active while the counter value is below the compare value (State Bit = 0).

In most cases, the switching behavior of the connected power switches is not symmetrical concerning the switch-on and switch-off times. A general problem arises if the time for switch-on is smaller than the time for switch-off of the power device. In this case, a short-circuit can occur in the inverter bridge leg, which may damage the complete system. In order to solve this problem by HW, this capture/compare unit

**Capture/Compare Unit 6 (CCU6)**

contains a programmable Dead-Time Generation Block, that delays the passive to active edge of the switching signals by a programmable time (the active to passive edge is not delayed).

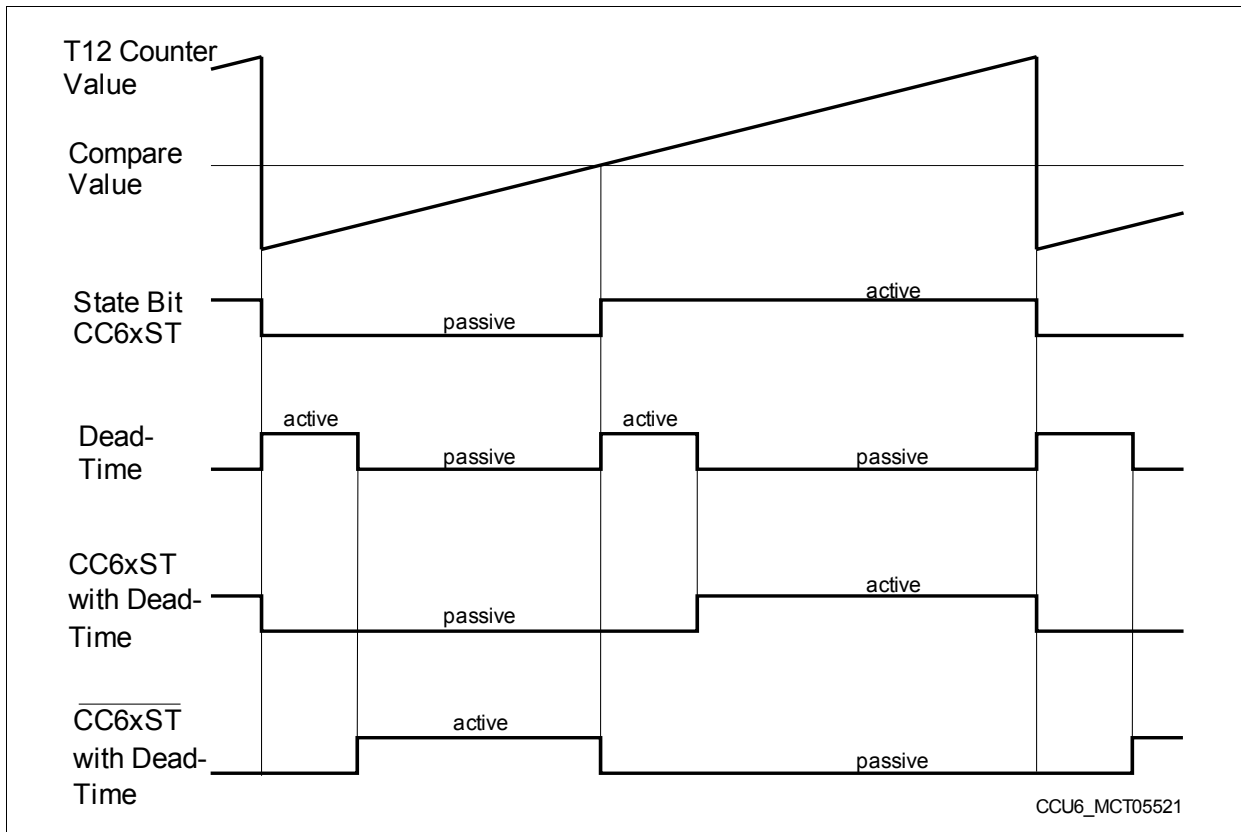
The Dead-Time Generation Block, illustrated in **Figure 20-16**, is built in a similar way for all three channels of T12. It is controlled by bits in register **T12DTC**. Any change of a CC6xST State Bit activates the corresponding Dead-Time Counter, that is clocked with the same input clock as T12 ( $f_{T12}$ ). The length of the dead-time can be programmed by bit field DTM. This value is identical for all three channels. Writing **TCTR4.DTRES = 1** sets all dead-times to passive.



**Figure 20-16 Dead-Time Generation Block Diagram**

Each of the three dead-time counters has its individual dead-time enable bit, DTE<sub>x</sub>. An enabled dead-time counter generates a dead-time delaying the passive-to-active edge of the channel output signal. The change in a State Bit CC6xST is not taken into account while the dead-time generation of this channel is currently in progress (active). This avoids an unintentional additional dead-time if a State Bit CC6xST changes too early. A disabled dead-time counter is always considered as passive and does not delay any edge of CC6xST.

Based on the State Bits CC6xST, the Dead-Time Generation Block outputs a direct signal CC6xST and an inverted signal  $\overline{\text{CC6xST}}$  for each compare channel, each masked with the effect of the related Dead-Time Counters (waveforms illustrated in **Figure 20-17**).



**Figure 20-17 Dead-Time Generation Waveforms**

#### 20.2.4.2 State Selection

To support a wide range of power switches and drivers, the state selection offers the flexibility to define when an output can be active and can be modulated, especially useful for **complementary or multi-phase PWM** signals.

The state selection is based on the signals  $\overline{\text{CC6xST}}$  and  $\text{CC6xST}$  delivered by the dead-time generator (see [Figure 20-15](#)). Both signals are never active at the same time, but can be passive at the same time. This happens during the dead-time of each compare channel after a change of the corresponding State Bit  $\text{CC6xST}$ .

The user can select independently for each output signal  $\text{CC6xO}$  and  $\text{COUT6xO}$  if it should be active before or after the compare value has been reached (see register [CMPSTAT](#)). With this selection, the active (conducting) phases of complementary power switches in a power inverter bridge leg can be positioned with respect to the compare value (e.g. signal  $\text{CC6xO}$  can be active before, whereas  $\text{COUT6xO}$  can be active after the compare value is reached). Like this, the output modulation, the trap logic and the output level selection can be programmed independently for each output signal, although two output signals are referring to the same compare channel.

### 20.2.4.3 Output Modulation and Level Selection

The last block of the data path is the Output Modulation block. Here, all the modulation sources and the trap functionality are combined and control the actual level of the output pins (controlled by the modulation enable bits T1xMODENy and MCMEN in register **MODCTR**). The following signal sources can be combined here **for each T12 output signal** (see **Figure 20-18** for compare channel CC60):

- A **T12 related compare signal** CC6x\_O (for outputs CC6x) or COUT6x\_O (for outputs COUT6x) delivered by the T12 block (state selection with dead-time) with an individual enable bit T12MODENy per output signal (y = 0, 2, 4 for outputs CC6x and y = 1, 3, 5 for outputs COUT6x)
- The **T13 related compare signal** CC63\_O delivered by the T13 state selection with an individual enable bit T13MODENy per output signal (y = 0, 2, 4 for outputs CC6x and y = 1, 3, 5 for outputs COUT6x)
- A **multi-channel output signal** MCMPy (y = 0, 2, 4 for outputs CC6x and y = 1, 3, 5 for outputs COUT6x) with a common enable bit MCMEN
- The **trap state** TRPS with an individual enable bit TRPENy per output signal (y = 0, 2, 4 for outputs CC6x and y = 1, 3, 5 for outputs COUT6x)

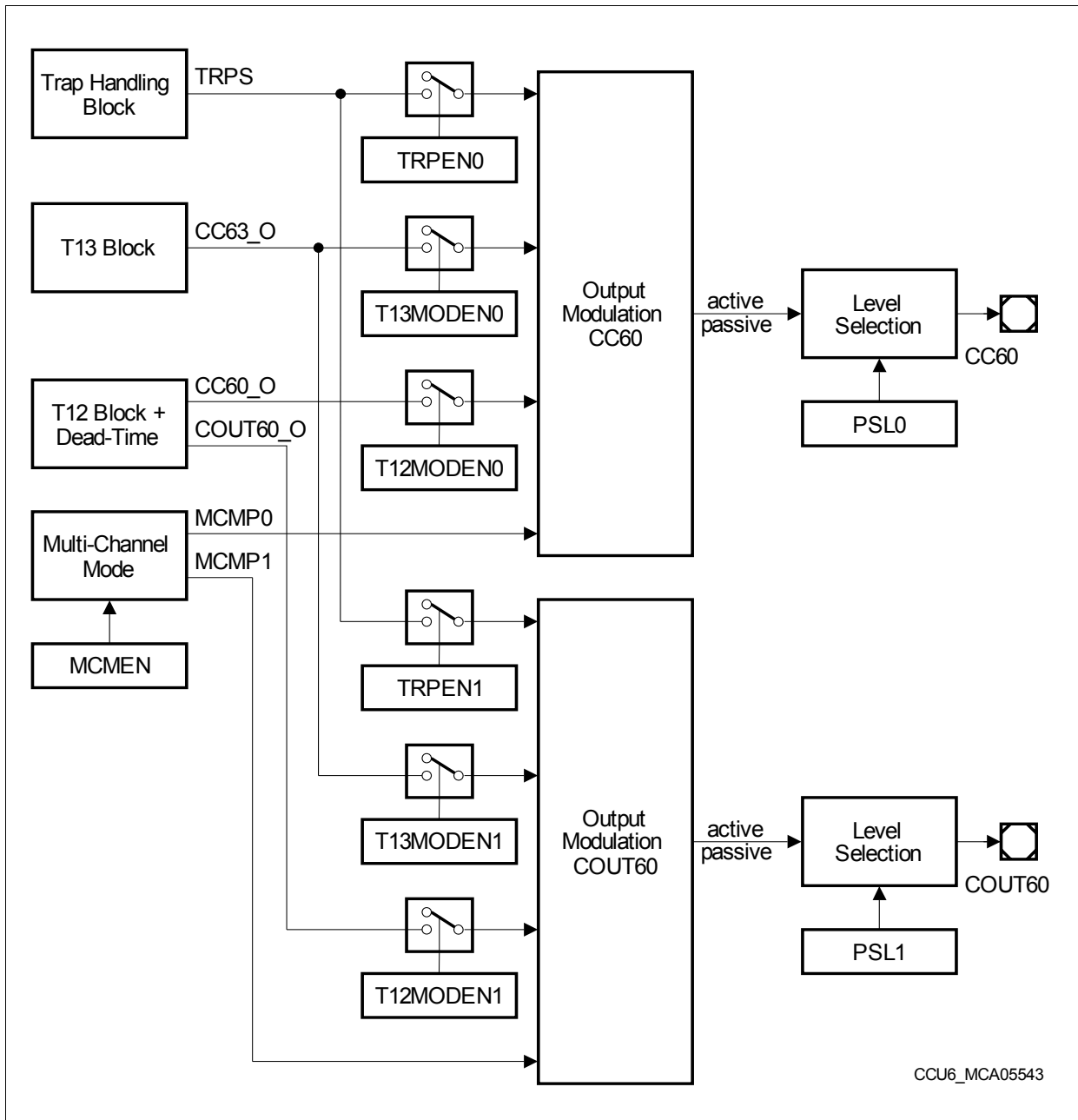
If one of the modulation input signals CC6x\_O/COUT6x\_O, CC63\_O, or MCMPy of an output modulation block is enabled and is at passive state, the modulated is also in passive state, regardless of the state of the other signals that are enabled. Only if all enabled signals are in active state the modulated output shows an active state. If no modulation input is enabled, the output is in passive state.

If the Trap State is active (TRPS = 1), then the outputs that are enabled for the trap signal (by TRPENy = 1) are set to the passive state.

The output of each of the modulation control blocks is connected to a level select block that is configured by register **PSLR**. It offers the option to determine the actual output level of a pin, depending on the state of the output line (decoupling of active/passive state and output polarity) as specified by the Passive State Select bit PSLy. If the modulated output signal is in the passive state, the level specified directly by PSLy is output. If it is in the active state, the inverted level of PSLy is output. This allows the user to adapt the polarity of an active output signal to the connected circuitry.

The PSLy bits have shadow registers to allow for updates without undesired pulses on the output lines. The bits related to CC6x and COUT6x (x = 0, 1, 2) are updated with the T12 shadow transfer signal (T12\_ST). A read action returns the actually used values, whereas a write action targets the shadow bits. Providing a shadow register for the PSL value as well as for other values related to the generation of the PWM signal facilitates a concurrent update by software for all relevant parameters.

**Figure 20-18** shows the output modulation structure for compare channel CC60 (output signals CC60 and COUT60). A similar structure is implemented for the other two compare channels CC61 and CC62.



**Figure 20-18 Output Modulation for Compare Channel CC60**



### 20.2.5 T12 Capture Modes

Each of the three channels of the T12 Block can also be used to capture T12 time information in response to an external signal CC6xIN.

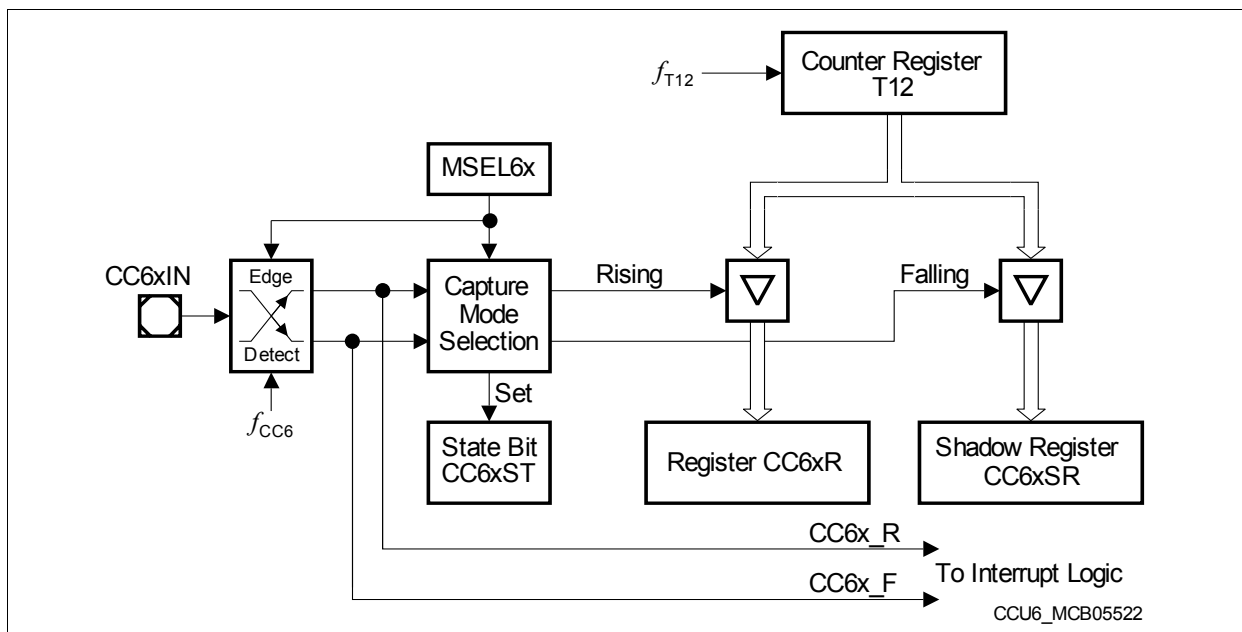
In capture mode, the interrupt event CC6x\_R is detected when a rising edge is detected at the input CC6xIN, whereas the interrupt event CC6x\_F is detected when a falling edge is detected.

There are a number of different modes for capture operation. In all modes, both of the registers of a channel are used. The selection of the capture modes is done via the **T12MSEL**.MSEL6x bit fields and can be selected individually for each of the channels.

**Table 20-3 Capture Modes Overview**

MSEL6x	Mode	Signal	Active Edge	CC6nSR Stored in	T12 Stored in
0100 <sub>B</sub>	1	CC6xIN	Rising	–	CC6xR
		CC6xIN	Falling	–	CC6xSR
0101 <sub>B</sub>	2	CC6xIN	Rising	CC6xR	CC6xSR
0110 <sub>B</sub>	3	CC6xIN	Falling	CC6xR	CC6xSR
0111 <sub>B</sub>	4	CC6xIN	Any	CC6xR	CC6xSR

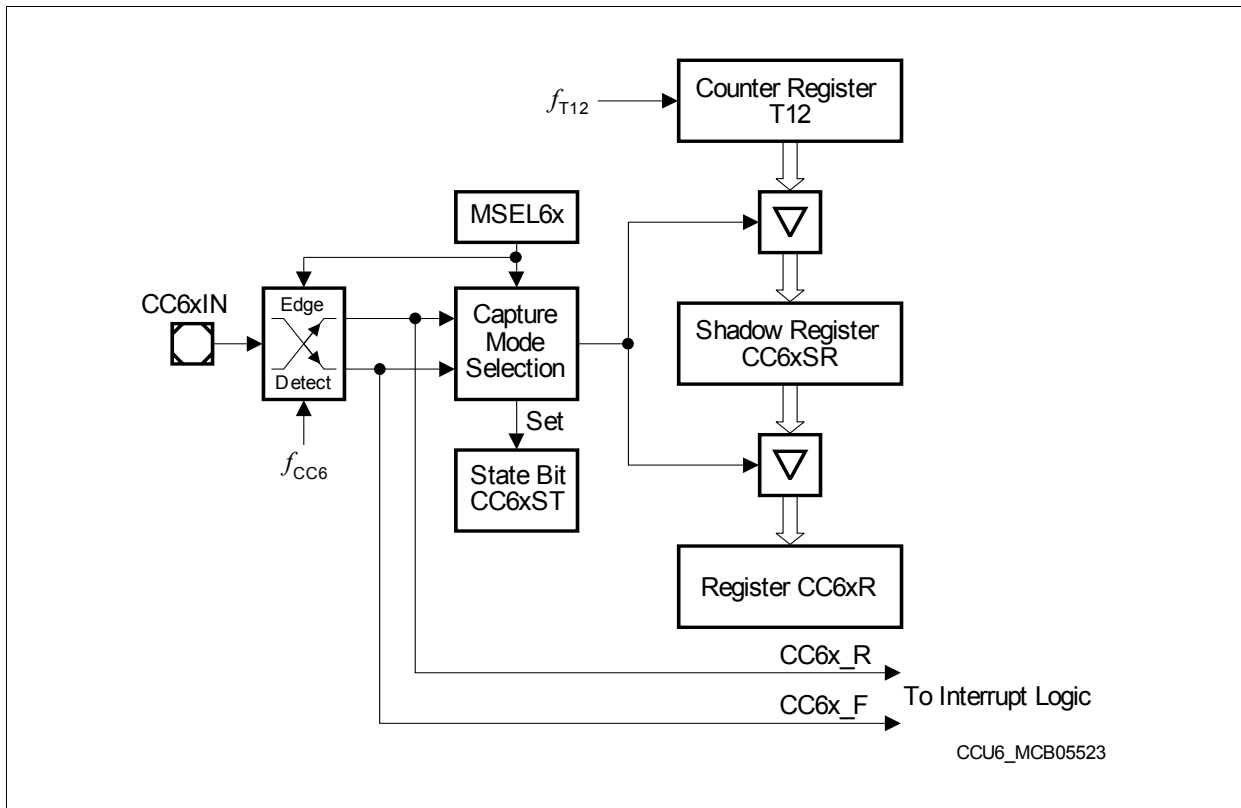
**Figure 20-19** illustrates **Capture Mode 1**. When a rising edge (0-to-1 transition) is detected at the corresponding input signal CC6xIN, the current contents of Timer T12 are captured into register CC6xR. When a falling edge (1-to-0 transition) is detected at the input signal CC6xIN, the contents of Timer T12 are captured into register CC6xSR.



**Figure 20-19 Capture Mode 1 Block Diagram**

**Capture/Compare Unit 6 (CCU6)**

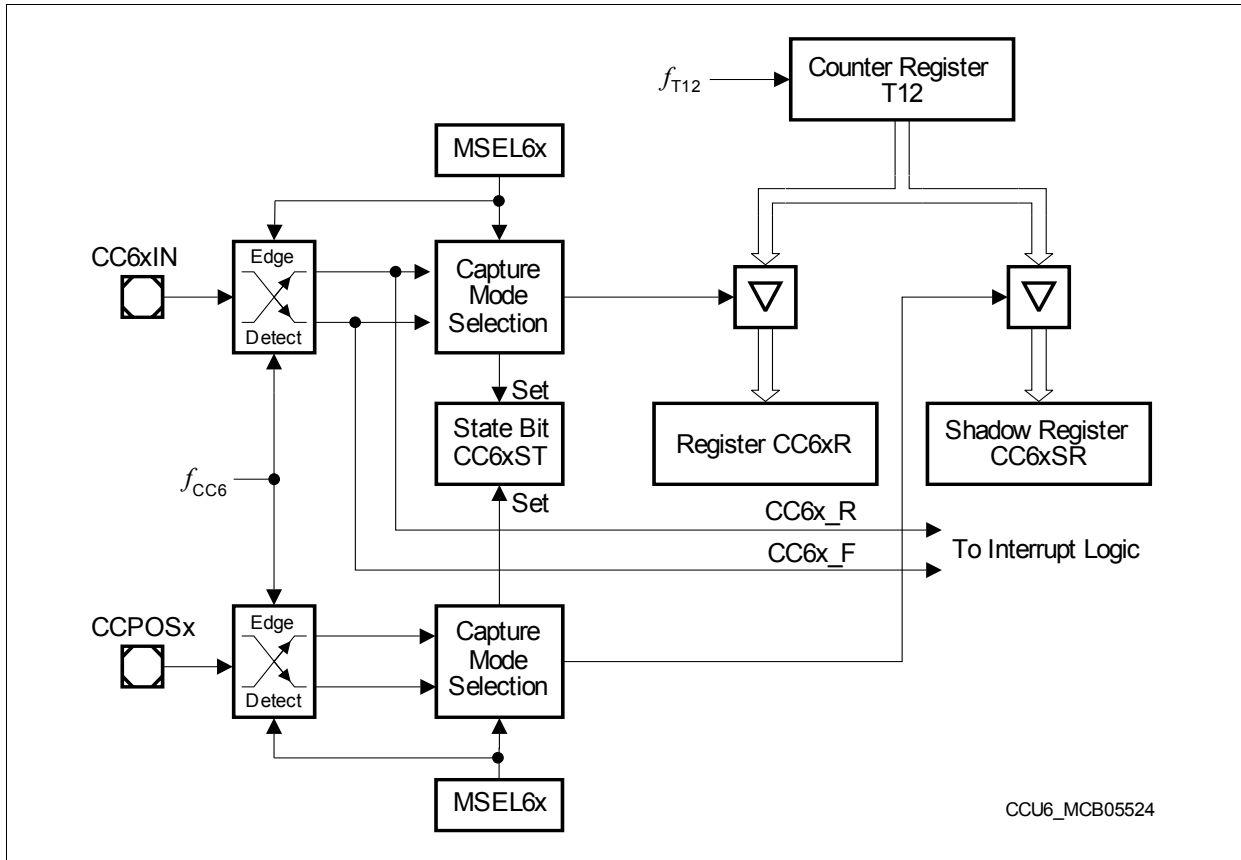
**Capture Modes 2, 3 and 4** are shown in [Figure 20-20](#). They differ only in the active edge causing the capture operation. In each of the three modes, when the selected edge is detected at the corresponding input signal CC6xIN, the current contents of the shadow register CC6xSR are transferred into register CC6xR, and the current Timer T12 contents are captured in register CC6xSR (simultaneous transfer). The active edge is a rising edge of CC6xIN for Capture Mode 2, a falling edge for Mode 3, and both, a rising or a falling edge for Capture Mode 4, as shown in [Table 20-3](#). These capture modes are very useful in cases where there is little time between two consecutive edges of the input signal.



**Figure 20-20 Capture Modes 2, 3 and 4 Block Diagram**

**Capture/Compare Unit 6 (CCU6)**

Five further capture modes are called **Multi-Input Capture Modes**, as they use two different external inputs, signal CC6xIN and signal CCPOSx.



**Figure 20-21 Multi-Input Capture Modes Block Diagram**

In each of these modes, the current T12 contents are captured in register CC6xR in response to a selected event at signal CC6xIN, and in register CC6xSR in response to a selected event at signal CCPOSx. The possible events can be opposite input transitions, or the same transitions, or any transition at the two inputs. The different options are detailed in [Table 20-4](#).

In each of the various capture modes, the Channel State Bit, CC6xST, is set to 1 when the selected capture trigger event at signal CC6xIN or CCPOSx has occurred. The State Bit is not cleared by hardware, but can be cleared by software.

In addition, appropriate signal lines to the interrupt logic are activated, that can generate an interrupt request to the CPU. Regardless of the selected active edge, all edges detected at signal CC6xIN can lead to the activation of the appropriate interrupt request line (see also [Section 20.8](#)).

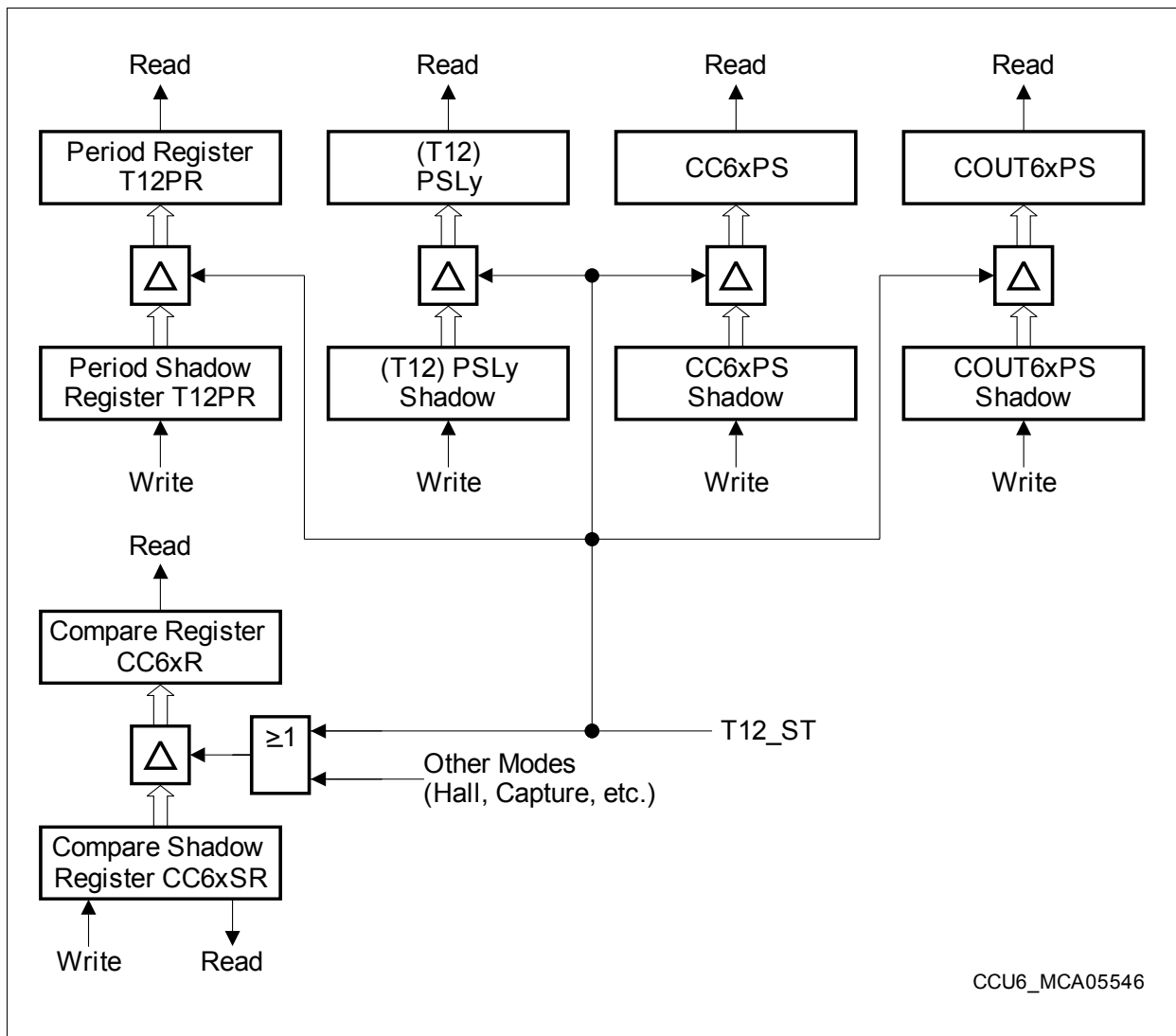
**Table 20-4 Multi-Input Capture Modes Overview**

<b>MSEL6x</b>	<b>Mode</b>	<b>Signal</b>	<b>Active Edge</b>	<b>T12 Stored in</b>
1010 <sub>B</sub>	5	CC6xIN	Rising	CC6xR
		CCPOSx	Falling	CC6xSR
1011 <sub>B</sub>	6	CC6xIN	Falling	CC6xR
		CCPOSx	Rising	CC6xSR
1100 <sub>B</sub>	7	CC6xIN	Rising	CC6xR
		CCPOSx	Rising	CC6xSR
1101 <sub>B</sub>	8	CC6xIN	Falling	CC6xR
		CCPOSx	Falling	CC6xSR
1110 <sub>B</sub>	9	CC6xIN	Any	CC6xR
		CCPOSx	Any	CC6xSR
1111 <sub>B</sub>	–	reserved (no capture or compare action)		

## 20.2.6 T12 Shadow Register Transfer

A special shadow transfer signal (T12\_ST) can be generated to facilitate updating the period and compare values of the compare channels CC60, CC61, and CC62 synchronously to the operation of T12. Providing a shadow register for values defining one PWM period facilitates a concurrent update by software for all relevant parameters. The next PWM period can run with a new set of parameters. The generation of this signal is requested by software via bit **TCTR0.STE12** (set by writing 1 to the write-only bit **TCTR4.T12STR**, cleared by writing 1 to the write-only bit **TCTR4.T12STD**).

**Figure 20-22** shows the shadow register structure and the shadow transfer signals, as well as on the read/write accessibility of the various registers.



**Figure 20-22 T12 Shadow Register Overview**

A T12 shadow register transfer takes place (T12\_ST active):

- while timer T12 is not running (T12R = 0), or
- STE12 = 1 and a Period-Match is detected while counting up, or
- STE12 = 1 and a One-Match is detected while counting down

When signal T12\_ST is active, a shadow register transfer is triggered with the next cycle of the T12 clock. Bit STE12 is automatically cleared with the shadow register transfer.

### 20.2.7 Timer T12 Operating Mode Selection

The operating mode for the T12 channels are defined by the bit fields **T12MSEL**.MSEL6x.

**Table 20-5 T12 Capture/Compare Modes Overview**

<b>MSEL6x</b>	<b>Selected Operating Mode</b>
0000 <sub>B</sub> , 1111 <sub>B</sub>	Capture/Compare modes switched off
0001 <sub>B</sub> , 0010 <sub>B</sub> , 0011 <sub>B</sub>	Compare mode, see <a href="#">Section 20.2.3</a> same behavior for all three codings
01XX <sub>B</sub>	Double-Register Capture modes, see <a href="#">Section 20.2.5</a>
1000 <sub>B</sub>	Hall Sensor Mode, see <a href="#">Section 20.6</a> In order to properly enable this mode, all three MSEL6x fields have to be programmed to Hall Sensor mode.
1001 <sub>B</sub>	Hysteresis-like compare mode, see <a href="#">Section 20.2.3.3</a>
1010 <sub>B</sub> , 1011 <sub>B</sub> , 1100 <sub>B</sub> , 1101 <sub>B</sub> , 1110 <sub>B</sub>	Multi-Input Capture modes, see <a href="#">Section 20.2.5</a>

The clocking and counting scheme of the timers are controlled by the timer control registers **TCTR0** and **TCTR2**. Specific actions are triggered by write operations to register **TCTR4**.



Field	Bits	Type	Description
<b>T12PV</b>	[15:0]	rwh	<b>T12 Period Value</b> The value T12PV defines the counter value for T12 leading to a period-match. When reaching this value, the timer T12 is set to zero (edge-aligned mode) or changes its count direction to down counting (center-aligned mode).

### 20.2.8.3 Capture/Compare Registers

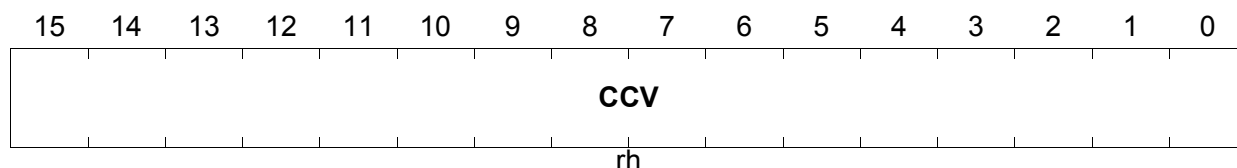
In compare mode, the registers CC6xR (x = 0 - 2) are the actual compare registers for T12. The values stored in CC6xR are compared (all three channels in parallel) to the counter value of T12. In capture mode, the current value of the T12 counter register is captured by registers CC6xR if the corresponding capture event is detected.

#### CC6xR (x = 0-2)

##### Capture/Compare Register for Channel CC6x

**XSFR(18<sub>H</sub> + 2\*x)**

**Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
<b>CCV</b>	[15:0]	rh	<b>Capture/Compare Value</b> In compare mode, the bit fields CCV contain the values, that are compared to the T12 counter value. In capture mode, the captured value of T12 can be read from these registers.



#### 20.2.8.4 Capture/Compare Shadow Registers

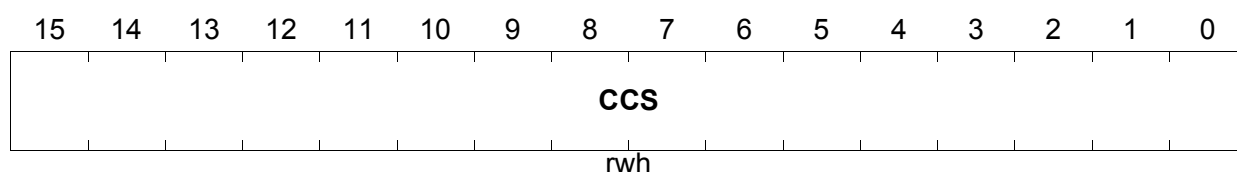
The registers CC6xR can only be read by SW, the modification of the value is done by a shadow register transfer from register CC6xSR. The corresponding shadow registers CC6xSR can be read and written by SW. In capture mode, the value of the T12 counter register can also be captured by registers CC6xSR if the selected capture event is detected (depending on the selected capture mode).

##### **CC6xSR (x=0-2)**

##### **Capture/Compare Shadow Reg. for Channel CC6x**

**XSFR(20<sub>H</sub>+2\*x)**

**Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
CCS	[15:0]	rwh	<b>Shadow Register for Channel x Capture/Compare Value</b> In compare mode, the bit fields contents of CCS are transferred to the bit fields CCV for the corresponding channel during a shadow transfer. In capture mode, the captured value of T12 can be read from these registers.

*Note: The shadow registers can also be written by SW in capture mode. In this case, the HW capture event wins over the SW write if both happen in the same cycle (the SW write is discarded).*

### 20.2.8.5 Dead-time Control Register

Register T12DTC controls the dead-time generation for the timer T12 compare channels. Each channel can be independently enabled/disabled for dead-time generation. If enabled, the transition from passive state to active state is delayed by the value defined by bit field DTM.

The dead time counters are clocked with the same frequency as T12.

This structure allows symmetrical dead-time generation in center-aligned and in edge-aligned PWM mode. A duty cycle of 50% leads to CC6x, COUT6x switched on for: 0.5 \* period - dead time.

*Note: The dead-time counters are not reset by bit T12RES, but by bit DTRES.*

#### T12DTC

##### Dead-Time Control Register for Timer12

XSFR(14 <sub>H</sub> )								Reset Value: 0000 <sub>H</sub>							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	DTR 2	DTR 1	DTR 0	0	DTE 2	DTE 1	DTE 0	DTM							
r	rh	rh	rh	r	rw	rw	rw	rw							

Field	Bits	Type	Description
<b>DTM</b>	[7:0]	rw	<b>Dead-Time</b> Bit field DTM determines the programmable delay between switching from the passive state to the active state of the selected outputs. The switching from the active state to the passive state is not delayed.
<b>DTE2, DTE1, DTE0</b>	10, 9, 8	rw	<b>Dead Time Enable Bits</b> Bits DTE0..DTE2 enable and disable the dead time generation for each compare channel (0, 1, 2) of timer T12. 0 <sub>B</sub> Dead-Time Counter x is disabled. The corresponding outputs switch from the passive state to the active state (according to the actual compare status) without any delay. 1 <sub>B</sub> Dead-Time Counter x is enabled. The corresponding outputs switch from the passive state to the active state (according to the compare status) with the delay programmed in bit field DTM.

**Capture/Compare Unit 6 (CCU6)**

Field	Bits	Type	Description
<b>DTR2, DTR1, DTR0</b>	14, 13, 12	rh	<b>Dead Time Run Indication Bits</b> Bits DTR0..DTR2 indicate the status of the dead time generation for each compare channel (0, 1, 2) of timer T12. 0 <sub>B</sub> Dead-Time Counter x is currently in the passive state. 1 <sub>B</sub> Dead-Time Counter x is currently in the active state.
<b>0</b>	15, 11	r	<b>reserved;</b> returns 0 if read; should be written with 0;

## 20.2.9 Capture/Compare Control Registers

### 20.2.9.1 Channel State Bits

The Compare State Register CMPSTAT contains status bits monitoring the current capture and compare state and control bits defining the active/passive state of the compare channels.

#### **CMPSTAT**

**Compare State Register**

**XSFR(28<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>T13 IM</b>	<b>C OUT 63PS</b>	<b>C OUT 62PS</b>	<b>CC 62PS</b>	<b>C OUT 61PS</b>	<b>CC 61PS</b>	<b>C OUT 60PS</b>	<b>CC 60PS</b>	<b>0</b>	<b>CC 63ST</b>	<b>CC POS 62</b>	<b>CC POS 61</b>	<b>CC POS 60</b>	<b>CC 62ST</b>	<b>CC 61ST</b>	<b>CC 60ST</b>
rw	rw	rw	rw	rw	rw	rw	rw	r	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
<b>CC60ST, CC61ST, CC62ST, CC63ST</b> 1)	0, 1, 2, 6	rh	<b>Capture/Compare State Bits</b> Bits CC6xST monitor the state of the capture/compare channels. Bits CC6xST (x = 0, 1, 2) are related to T12, bit CC63ST is related to T13. 0 <sub>B</sub> In compare mode, the timer count is less than the compare value. In capture mode, the selected edge has not yet been detected since the bit has been cleared by SW the last time. 1 <sub>B</sub> In compare mode, the counter value is greater than or equal to the compare value. In capture mode, the selected edge has been detected.
<b>CCPOS60, CCPOS61, CCPOS62</b>	3, 4, 5	rh	<b>Sampled Hall Pattern Bits</b> Bits CCPOS6x (x = 0, 1, 2) are indicating the value of the input Hall pattern that has been compared to the current and expected value. The value is sampled when the event HCRDY (Hall Compare Ready) occurs. 0 <sub>B</sub> The input CCPOS6x has been sampled as 0. 1 <sub>B</sub> The input CCPOS6x has been sampled as 1.

**Capture/Compare Unit 6 (CCU6)**

Field	Bits	Type	Description
<b>CC60PS, CC61PS, CC62PS, COUT60PS, COUT61PS, COUT62PS, COUT63PS</b> <sup>2)</sup>	8, 10, 12, 9, 11, 13, 14	rwh	<b>Passive State Select for Compare Outputs</b> Bits CC6xPS, COUT6xPS select the state of the corresponding compare channel, that is considered to be the passive state. During the passive state, the passive level (defined in register PSLR) is driven by the output pin. Bits CC6xPS, COUT6xPS (x = 0, 1, 2) are related to T12, bit CC63PS is related to T13. 0 <sub>B</sub> The corresponding compare signal is in passive state while CC6xST is 0. 1 <sub>B</sub> The corresponding compare signal is in passive state while CC6xST is 1. In capture mode, these bits are not used.
<b>T13IM</b> <sup>3)</sup>	15	rwh	<b>T13 Inverted Modulation</b> Bit T13IM inverts the T13 signal for the modulation of the CC6x and COUT6x (x = 0, 1, 2) signals. 0 <sub>B</sub> T13 output CC63_O is equal to CC63ST. 1 <sub>B</sub> T13 output CC63_O is equal to $\overline{\text{CC63ST}}$ .
<b>0</b>	7	r	<b>reserved;</b> returns 0 if read; should be written with 0;

- 1) These bits are set and cleared according to the T12, T13 switching rules
- 2) These bits have shadow bits and are updated in parallel to the capture/compare registers of T12, T13 respectively. A read action targets the actually used values, whereas a write action targets the shadow bits.
- 3) This bit has a shadow bit and is updated in parallel to the compare and period registers of T13. A read action targets the actually used values, whereas a write action targets the shadow bit.

**Capture/Compare Unit 6 (CCU6)**

The Compare Status Modification Register CMPMODIF provides software-control (independent set and clear conditions) for the channel state bits CC6xST. This feature enables the user to individually change the status of the output lines by software, for example when the corresponding compare timer is stopped.

**CMPMODIF**

**Compare State Modification Register**

**XSFR(2A<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	MCC 63R		0		MCC 62R	MCC 61R	MCC 60R	0	MCC 63S		0		MCC 62S	MCC 61S	MCC 60S
r	w		r		w	w	w	r	w		r		w	w	w

Field	Bits	Type	Description
MCC60S, MCC61S, MCC62S, MCC63S, MCC60R, MCC61R, MCC62R, MCC63R	0, 1, 2, 6, 8, 9, 10, 14	w	<b>Capture/Compare Status Modification Bits</b> These bits are used to bits to set (MCC6xS) or to clear (MCC6xR) the corresponding bits CC6xST by SW. This feature allows the user to individually change the status of the output lines by SW, e.g. when the corresponding compare timer is stopped. This allows a bit manipulation of CC6xST-bits by a single data write action. The following functionality of a write access to bits concerning the same capture/compare state bit is provided: [MCC6xR, MCC6xS] = 00 <sub>B</sub> Bit CC6xST is not changed. 01 <sub>B</sub> Bit CC6xST is set. 10 <sub>B</sub> Bit CC6xST is cleared. 11 <sub>B</sub> reserved
0	[5:3], 7, [13:11], 15	r	<b>reserved;</b> returns 0 if read; should be written with 0;

### 20.2.9.2 T12 Mode Control Register

Register T12MSEL contains control bits to select the capture/compare functionality of the three channels of Timer T12.

#### T12MSEL

##### T12 Mode Select Register

**XSFR (46<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>D BYP</b>	<b>HSYNC</b>			<b>MSEL62</b>			<b>MSEL61</b>			<b>MSEL60</b>					
rw	rw			rw			rw			rw			rw		

Field	Bits	Type	Description
<b>MSEL60, MSEL61, MSEL62</b>	[3:0], [7:4], [11:8]	rw	<b>Capture/Compare Mode Selection</b> These bit fields select the operating mode of the three T12 capture/compare channels. Each channel (x = 0, 1, 2) can be programmed individually for one of these modes (except for Hall Sensor Mode). Coding see <a href="#">Table 20-5</a> .
<b>HSYNC</b>	[14:12]	rw	<b>Hall Synchronization</b> Bit field HSYNC defines the source for the sampling of the Hall input pattern and the comparison to the current and the expected Hall pattern bit fields. Coding see <a href="#">Table 20-11</a> .
<b>DBYP</b>	15	rw	<b>Delay Bypass</b> DBYP controls whether the source signal for the sampling of the Hall input pattern (selected by HSYNC) is delayed by the Dead-Time Counter 0. <div> <div>0<sub>B</sub></div> <div>The bypass is not active.</div> <div>Dead-Time Counter 0 is generating a delay after the source signal becomes active.</div> </div> <div> <div>1<sub>B</sub></div> <div>The bypass is active.</div> <div>Dead-Time Counter 0 is not used for a delay.</div> </div>

### 20.2.9.3 Timer Control Registers

Register TCTR0 controls the basic functionality of both timers, T12 and T13.

*Note: A write action to the bit fields T12CLK or T12PRE is only taken into account while the timer T12 is not running (T12R=0). A write action to the bit fields T13CLK or T13PRE is only taken into account while the timer T13 is not running (T13R=0).*

#### TCTR0

**Timer Control Register 0**

**XSFR(2C<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		STE 13	T13R	T13 PRE	T13CLK			CTM	CDIR	STE 12	T12R	T12 PRE	T12CLK		
r		rh	rh	rw	rw			rw	rh	rh	rh	rw	rw		

Field	Bits	Type	Description
<b>T12CLK</b>	[2:0]	rw	<b>Timer T12 Input Clock Select</b> Selects the input clock for timer T12 that is derived from the peripheral clock according to the equation $f_{T12} = f_{CC6} / 2^{<T12CLK>}$ . 000 <sub>B</sub> $f_{T12} = f_{CC6}$ 001 <sub>B</sub> $f_{T12} = f_{CC6} / 2$ 010 <sub>B</sub> $f_{T12} = f_{CC6} / 4$ 011 <sub>B</sub> $f_{T12} = f_{CC6} / 8$ 100 <sub>B</sub> $f_{T12} = f_{CC6} / 16$ 101 <sub>B</sub> $f_{T12} = f_{CC6} / 32$ 110 <sub>B</sub> $f_{T12} = f_{CC6} / 64$ 111 <sub>B</sub> $f_{T12} = f_{CC6} / 128$
<b>T12PRE</b>	3	rw	<b>Timer T12 Prescaler Bit</b> In order to support higher clock frequencies, an additional prescaler factor of 1/256 can be enabled for the prescaler for T12. 0 <sub>B</sub> The additional prescaler for T12 is disabled. 1 <sub>B</sub> The additional prescaler for T12 is enabled.
<b>T12R</b>	4	rh	<b>Timer T12 Run Bit<sup>1)</sup></b> T12R starts and stops timer T12. It is set/cleared by SW by setting bits T12RR or T12RS or it is cleared by HW according to the function defined by bit field T12SSC. 0 <sub>B</sub> Timer T12 is stopped. 1 <sub>B</sub> Timer T12 is running.



Field	Bits	Type	Description
<b>STE12</b>	5	rh	<b>Timer T12 Shadow Transfer Enable</b> Bit STE12 enables or disables the shadow transfer of the T12 period value, the compare values and passive state select bits and levels from their shadow registers to the actual registers if a T12 shadow transfer event is detected. Bit STE12 is cleared by hardware after the shadow transfer. A T12 shadow transfer event is a period-match while counting up or a one-match while counting down. 0 <sub>B</sub> The shadow register transfer is disabled. 1 <sub>B</sub> The shadow register transfer is enabled.
<b>CDIR</b>	6	rh	<b>Count Direction of Timer T12</b> This bit is set/cleared according to the counting rules of T12. 0 <sub>B</sub> T12 counts up. 1 <sub>B</sub> T12 counts down.
<b>CTM</b>	7	rw	<b>T12 Operating Mode</b> 0 <sub>B</sub> Edge-aligned Mode: T12 always counts up and continues counting from zero after reaching the period value. 1 <sub>B</sub> Center-aligned Mode: T12 counts down after detecting a period-match and counts up after detecting a one-match.
<b>T13CLK</b>	[10:8]	rw	<b>Timer T13 Input Clock Select</b> Selects the input clock for timer T13 that is derived from the peripheral clock according to the equation $f_{T13} = f_{CC6} / 2^{<T13CLK>}$ . 000 <sub>B</sub> $f_{T13} = f_{CC6}$ 001 <sub>B</sub> $f_{T13} = f_{CC6} / 2$ 010 <sub>B</sub> $f_{T13} = f_{CC6} / 4$ 011 <sub>B</sub> $f_{T13} = f_{CC6} / 8$ 100 <sub>B</sub> $f_{T13} = f_{CC6} / 16$ 101 <sub>B</sub> $f_{T13} = f_{CC6} / 32$ 110 <sub>B</sub> $f_{T13} = f_{CC6} / 64$ 111 <sub>B</sub> $f_{T13} = f_{CC6} / 128$

**Capture/Compare Unit 6 (CCU6)**

Field	Bits	Type	Description
<b>T13PRE</b>	11	rw	<b>Timer T13 Prescaler Bit</b> In order to support higher clock frequencies, an additional prescaler factor of 1/256 can be enabled for the prescaler for T13. 0 <sub>B</sub> The additional prescaler for T13 is disabled. 1 <sub>B</sub> The additional prescaler for T13 is enabled.
<b>T13R</b>	12	rh	<b>Timer T13 Run Bit<sup>2)</sup></b> T13R starts and stops timer T13. It is set/cleared by SW by setting bits T13RR or T13RS or it is set/cleared by HW according to the function defined by bit fields T13SSC, T13TEC and T13TED. 0 <sub>B</sub> Timer T13 is stopped. 1 <sub>B</sub> Timer T13 is running.
<b>STE13</b>	13	rh	<b>Timer T13 Shadow Transfer Enable</b> Bit STE13 enables or disables the shadow transfer of the T13 period value, the compare value and passive state select bit and level from their shadow registers to the actual registers if a T13 shadow transfer event is detected. Bit STE13 is cleared by hardware after the shadow transfer. A T13 shadow transfer event is a period-match. 0 <sub>B</sub> The shadow register transfer is disabled. 1 <sub>B</sub> The shadow register transfer is enabled.
<b>0</b>	[15: 14]	r	<b>reserved;</b> returns 0 if read; should be written with 0;

1) A concurrent set/clear action on T12R (from T12SSC, T12RR or T12RS) will have no effect. The bit T12R will remain unchanged.

2) A concurrent set/cleared action on T13R (from T13SSC, T13TEC, T13RR or T13RS) will have no effect. The bit T12R will remain unchanged.

**Capture/Compare Unit 6 (CCU6)**

Register TCTR2 controls the single-shot and the synchronization functionality of both timers T12 and T13. Both timers can run in single-shot mode. In this mode they stop their counting sequence automatically after one counting period with a count value of zero. The single-shot mode and the synchronization feature of T13 to T12 allow the generation of events with a programmable delay after well-defined PWM actions of T12.

**TCTR2**

**Timer Control Register 2**

**XSFR(2E<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>0</b>				<b>T13 RSEL</b>		<b>T12 RSEL</b>		<b>0</b>	<b>T13 TED</b>		<b>T13 TEC</b>		<b>T13 SSC</b>	<b>T12 SSC</b>	
r				rw		rw		r	rw		rw		rw	rw	

Field	Bits	Type	Description
<b>T12SSC</b>	0	rw	<b>Timer T12 Single Shot Control</b> This bit controls the single shot-mode of T12. 0 <sub>B</sub> The single-shot mode is disabled, no HW action on T12R. 1 <sub>B</sub> The single shot mode is enabled, the bit T12R is cleared by HW if - T12 reaches its period value in edge-aligned mode - T12 reaches the value 1 while down counting in center-aligned mode. In parallel to the clear action of bit T12R, the bits CC6xST (x=0, 1, 2) are cleared.
<b>T13SSC</b>	1	rw	<b>Timer T13 Single Shot Control</b> This bit controls the single shot-mode of T13. 0 <sub>B</sub> No HW action on T13R 1 <sub>B</sub> The single-shot mode is enabled, the bit T13R is cleared by HW if T13 reaches its period value. In parallel to the clear action of bit T13R, the bit CC63ST is cleared.

Field	Bits	Type	Description
<b>T13TEC</b>	[4:2]	rw	<b>T13 Trigger Event Control</b> bit field T13TEC selects the trigger event to start T13 (automatic set of T13R for synchronization to T12 compare signals) according to following combinations: 000 <sub>B</sub> no action 001 <sub>B</sub> set T13R on a T12 compare event on channel 0 010 <sub>B</sub> set T13R on a T12 compare event on channel 1 011 <sub>B</sub> set T13R on a T12 compare event on channel 2 100 <sub>B</sub> set T13R on any T12 compare event (ch. 0, 1, 2) 101 <sub>B</sub> set T13R upon a period-match of T12 110 <sub>B</sub> set T13R upon a zero-match of T12 (while counting up) 111 <sub>B</sub> set T13R on any edge of inputs CCPOSx
<b>T13TED</b>	[6:5]	rw	<b>Timer T13 Trigger Event Direction<sup>1)</sup></b> Bit field T13TED delivers additional information to control the automatic set of bit T13R in the case that the trigger action defined by T13TEC is detected. 00 <sub>B</sub> reserved, no action 01 <sub>B</sub> while T12 is counting up 10 <sub>B</sub> while T12 is counting down 11 <sub>B</sub> independent on the count direction of T12
<b>T12RSEL</b>	[9:8]	rw	<b>Timer T12 External Run Selection</b> Bit field T12RSEL defines the event of signal T12HR that can set the run bit T12R by HW. 00 <sub>B</sub> The external setting of T12R is disabled. 01 <sub>B</sub> Bit T12R is set if a rising edge of signal T12HR is detected. 10 <sub>B</sub> Bit T12R is set if a falling edge of signal T12HR is detected. 11 <sub>B</sub> Bit T12R is set if an edge of signal T12HR is detected.

Field	Bits	Type	Description
<b>T13RSEL</b>	[11:10]	rw	<b>Timer T13 External Run Selection</b> Bit field T13RSEL defines the event of signal T13HR that can set the run bit T13R by HW. 00 <sub>B</sub> The external setting of T13R is disabled. 01 <sub>B</sub> Bit T13R is set if a rising edge of signal T13HR is detected. 10 <sub>B</sub> Bit T13R is set if a falling edge of signal T13HR is detected. 11 <sub>B</sub> Bit T13R is set if an edge of signal T13HR is detected.
<b>0</b>	7, [15: 12]	r	<b>reserved;</b> returns 0 if read; should be written with 0;

1) Example:

If the timer T13 is intended to start at any compare event on T12 (T13TEC=100) the trigger event direction can be programmed to

- counting up >> a T12 channel 0, 1, 2 compare match triggers T13R only while T12 is counting up
- counting down >> a T12 channel 0, 1, 2 compare match triggers T13R only while T12 is counting down
- independent from bit CDIR >> each T12 channel 0, 1, 2 compare match triggers T13R

The timer count direction is taken from the value of bit CDIR. As a result, if T12 is running in edge-aligned mode (counting up only), T13 can only be started automatically if bit field T13TED=01 or 11.

**Capture/Compare Unit 6 (CCU6)**

Register TCTR4 provides software-control (independent set and clear conditions) for the run bits T12R and T13R. Furthermore, the timers can be reset (while running) and bits STE12 and STE13 can be controlled by software. Reading these bits always returns 0.

**TCTR4**

**Timer Control Register 4**

**XSFR(26<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>T13 STD</b>	<b>T13 STR</b>	<b>T13 CNT</b>	<b>0</b>		<b>T13 RES</b>	<b>T13 RS</b>	<b>T13 RR</b>	<b>T12 STD</b>	<b>T12 STR</b>	<b>T12 CNT</b>	<b>0</b>	<b>DT RES</b>	<b>T12 RES</b>	<b>T12 RS</b>	<b>T12 RR</b>
w	w	w	r		w	w	w	w	w	w	r	w	w	w	w

Field	Bits	Type	Description
<b>T12RR</b>	0	w	<b>Timer T12 Run Reset</b> Setting this bit clears the T12R bit. 0 <sub>B</sub> T12R is not influenced. 1 <sub>B</sub> T12R is cleared, T12 stops counting.
<b>T12RS</b>	1	w	<b>Timer T12 Run Set</b> Setting this bit sets the T12R bit. 0 <sub>B</sub> T12R is not influenced. 1 <sub>B</sub> T12R is set, T12 starts counting.
<b>T12RES</b>	2	w	<b>Timer T12 Reset</b> 0 <sub>B</sub> No effect on T12. 1 <sub>B</sub> The T12 counter register is cleared to zero. The switching of the output signals is according to the switching rules. Setting of T12RES has no impact on bit T12R.
<b>DTRES</b>	3	w	<b>Dead-Time Counter Reset</b> 0 <sub>B</sub> No effect on the dead-time counters. 1 <sub>B</sub> The three dead-time counter channels are cleared to zero.
<b>T12CNT</b>	5	w	<b>Timer T12 Count Event</b> 0 <sub>B</sub> No action 1 <sub>B</sub> If enabled (PISELH), timer T12 counts one step.
<b>T12STR</b>	6	w	<b>Timer T12 Shadow Transfer Request</b> 0 <sub>B</sub> No action 1 <sub>B</sub> STE12 is set, enabling the shadow transfer.

**Capture/Compare Unit 6 (CCU6)**

Field	Bits	Type	Description
<b>T12STD</b>	7	w	<b>Timer T12 Shadow Transfer Disable</b> 0 <sub>B</sub> No action 1 <sub>B</sub> STE12 is cleared without triggering the shadow transfer.
<b>T13RR</b>	8	w	<b>Timer T13 Run Reset</b> Setting this bit clears the T13R bit. 0 <sub>B</sub> T13R is not influenced. 1 <sub>B</sub> T13R is cleared, T13 stops counting.
<b>T13RS</b>	9	w	<b>Timer T13 Run Set</b> Setting this bit sets the T13R bit. 0 <sub>B</sub> T13R is not influenced. 1 <sub>B</sub> T13R is set, T13 starts counting.
<b>T13RES</b>	10	w	<b>Timer T13 Reset</b> 0 <sub>B</sub> No effect on T13. 1 <sub>B</sub> The T13 counter register is cleared to zero. The switching of the output signals is according to the switching rules. Setting of T13RES has no impact on bit T13R.
<b>T13CNT</b>	13	w	<b>Timer T13 Count Event</b> 0 <sub>B</sub> No action 1 <sub>B</sub> If enabled (PISELH), timer T13 counts one step.
<b>T13STR</b>	14	w	<b>Timer T13 Shadow Transfer Request</b> 0 <sub>B</sub> No action 1 <sub>B</sub> STE13 is set, enabling the shadow transfer.
<b>T13STD</b>	15	w	<b>Timer T13 Shadow Transfer Disable</b> 0 <sub>B</sub> No action 1 <sub>B</sub> STE13 is cleared without triggering the shadow transfer.
<b>0</b>	4, [12:11]	r	<b>reserved;</b> returns 0 if read; should be written with 0;

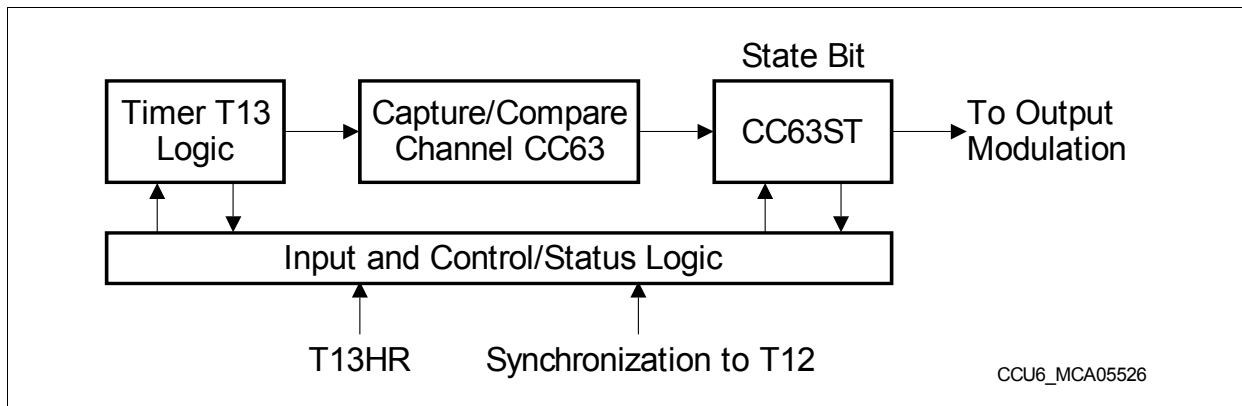
*Note: A simultaneous write of a 1 to bits that set and clear the same bit will trigger no action. The corresponding bit will remain unchanged.*

## 20.3 Operating Timer T13

Timer T13 is implemented similarly to Timer T12, but only with one channel in compare mode. A 16-bit up-counter is connected to a channel register via a comparator, that generates a signal when the counter contents match the contents of the channel register. A variety of control functions facilitate the adaptation of the T13 structure to different application needs. In addition, T13 can be started synchronously to timer T12 events.

This section provides information about:

- T13 overview (see [Section 20.3.1](#))
- Counting scheme (see [Section 20.3.2](#))
- Compare mode (see [Section 20.3.3](#))
- Compare output path (see [Section 20.3.4](#))
- Shadow register transfer (see [Section 20.3.5](#))
- T13 counter register description (see [Section 20.3.6](#))



**Figure 20-23 Overview Diagram of the Timer T13 Block**

### 20.3.1 T13 Overview

**Figure 20-24** shows a detailed block diagram of Timer T13. The functions of the timer T12 block are controlled by bits in registers **TCTR0**, **TCTR2**, **TCTR4**, and **PISELH**. Timer T13 receives its input clock,  $f_{T13}$ , from the module clock  $f_{CC6}$  via a programmable prescaler and an optional 1/256 divider or from an input signal T13HR. T13 can only count up (similar to the Edge-Aligned mode of T12).

Via a comparator, the timer T13 Counter Register **T13** is connected to the Period Register **T13PR**. This register determines the maximum count value for T13. When T13 reaches the period value, signal T13\_PM (T13 Period Match) is generated and T13 is cleared to 0000<sub>H</sub> with the next T13 clock edge. The Period Register receives a new period value from its Shadow Period Register, T13PS, that is loaded via software. The transfer of a new period value from the shadow register into T13PR is controlled via the 'T13 Shadow Transfer' control signal, T13\_ST. The generation of this signal depends on the associated control bit STE13. Providing a shadow register for the period value as

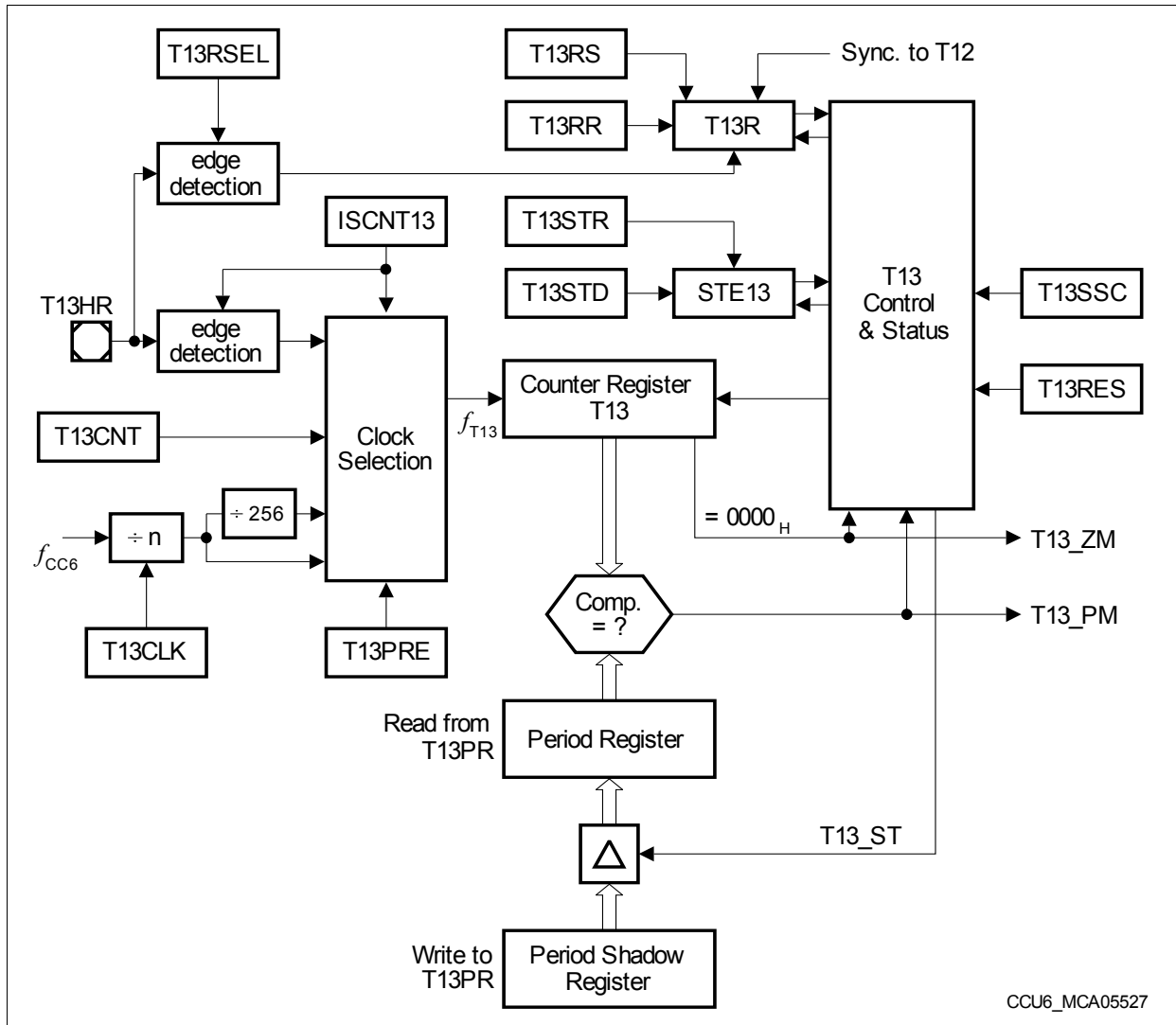


**Capture/Compare Unit 6 (CCU6)**

well as for other values related to the generation of the PWM signal facilitates a concurrent update by software for all relevant parameters (refer to [Table 20.3.5](#)).

Another signal indicates whether the counter contents are equal to 0000<sub>H</sub> (T13\_ZM).

A Single-Shot control bit, T13SSC, enables an automatic stop of the timer when the current counting period is finished (see [Figure 20-26](#)).



**Figure 20-24 T13 Counter Logic and Period Comparators**

The start or stop of T13 is controlled by the Run bit, T13R. This control bit can be set by software via the associated set/clear bits T13RS or T13RR in register [TCTR4](#), or it is cleared by hardware according to preselected conditions (single-shot mode).

The timer T13 run bit T13R must not be set while the applied T13 period value is zero. Bit T13R can be set automatically if an event of T12 is detected to synchronize T13 timings to T12 events, e.g. to generate a programmable delay via T13 after an edge of a T12 compare channel before triggering an AD conversion (T13 can trigger ADC

conversions).

Timer T13 can be cleared to 0000<sub>H</sub> via control bit T13RES. Setting this write-only bit only clears the timer contents, but has no further effects, e.g., it does not stop the timer.

The generation of the T13 shadow transfer control signal, T13\_ST, is enabled via bit STE13. This bit can be set or cleared by software indirectly through its associated set/reset control bits T13STR and T13STD.

Two bit fields, T13TEC and T13TED, control the synchronization of T13 to Timer T12 events. T13TEC selects the trigger event, while T13TED determines for which T12 count direction the trigger should be active.

While Timer T13 is running, write accesses to the count register T13 are not taken into account. If T13 is stopped, write actions to register T13 are immediately taken into account.

*Note: The T13 Period Register and its associated shadow register are located at the same physical address. A write access to this address targets the Shadow Register, while a read access reads from the actual period register.*

## 20.3.2 T13 Counting Scheme

This section describes the clocking and the counting capabilities of T13.

### 20.3.2.1 Clock Selection

In **Timer Mode** (**PISELH.ISCNT13** = 00<sub>B</sub>), the input clock  $f_{T13}$  of Timer T13 is derived from the internal module clock  $f_{CC6}$  through a programmable prescaler and an optional 1/256 divider. The resulting prescaler factors are listed in **Table 20-6**. The prescaler of T13 is cleared while T13 is not running (**TCTR0.T13R** = 0) to ensure reproducible timings and delays.

**Table 20-6 Timer T13 Input Clock Options**

<b>T13CLK</b>	<b>Resulting Input Clock <math>f_{T13}</math> Prescaler Off (T13PRE = 0)</b>	<b>Resulting Input Clock <math>f_{T13}</math> Prescaler On (T13PRE = 1)</b>
000 <sub>B</sub>	$f_{CC6}$	$f_{CC6} / 256$
001 <sub>B</sub>	$f_{CC6} / 2$	$f_{CC6} / 512$
010 <sub>B</sub>	$f_{CC6} / 4$	$f_{CC6} / 1024$
011 <sub>B</sub>	$f_{CC6} / 8$	$f_{CC6} / 2048$
100 <sub>B</sub>	$f_{CC6} / 16$	$f_{CC6} / 4096$
101 <sub>B</sub>	$f_{CC6} / 32$	$f_{CC6} / 8192$
110 <sub>B</sub>	$f_{CC6} / 64$	$f_{CC6} / 16384$
111 <sub>B</sub>	$f_{CC6} / 128$	$f_{CC6} / 32768$

In **Counter Mode**, timer T13 counts one step:

- If a 1 is written to **TCTR4.T13CNT** and **PISELH.ISCNT13** = 01<sub>B</sub>
- If a rising edge of input signal T13HR is detected and **PISELH.ISCNT13** = 10<sub>B</sub>
- If a falling edge of input signal T13HR is detected and **PISELH.ISCNT13** = 11<sub>B</sub>

### 20.3.2.2 T13 Counting

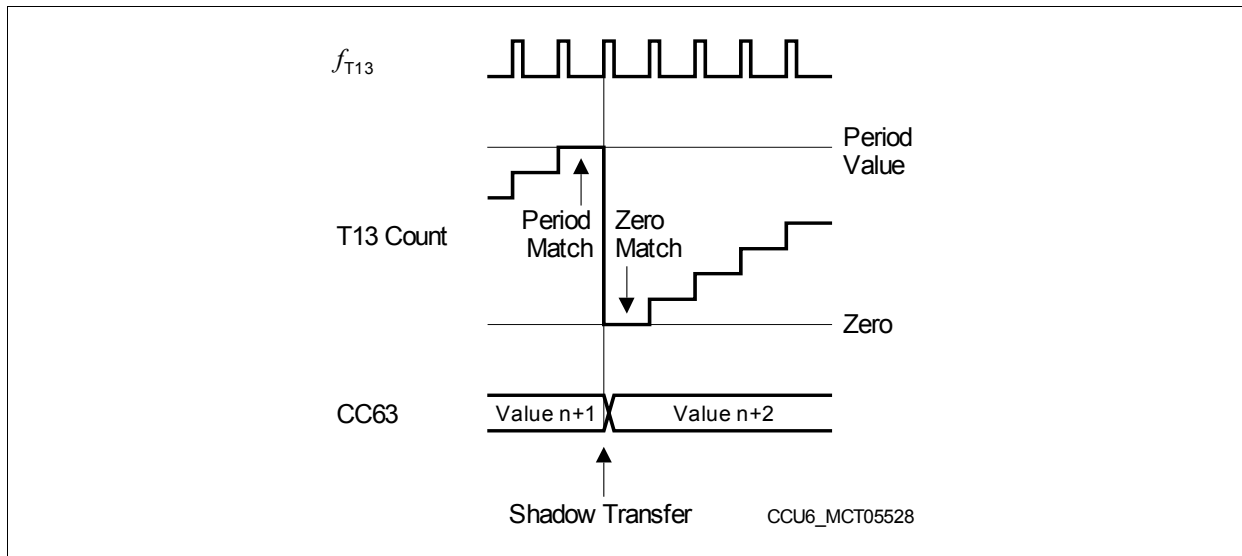
The period of the timer is determined by the value in the period Register T13PR according to the following formula:

$$T13_{PER} = \text{<Period-Value>} + 1; \text{ in } T13 \text{ clocks } (f_{T13}) \quad (20.3)$$

Timer T13 can only count up, comparable to the Edge-Aligned mode of T12. This leads to very simple 'counting rule' for the T13 counter:

- The counter is cleared with the next T13 clock edge if a Period-Match is detected. The counting direction is always upwards.

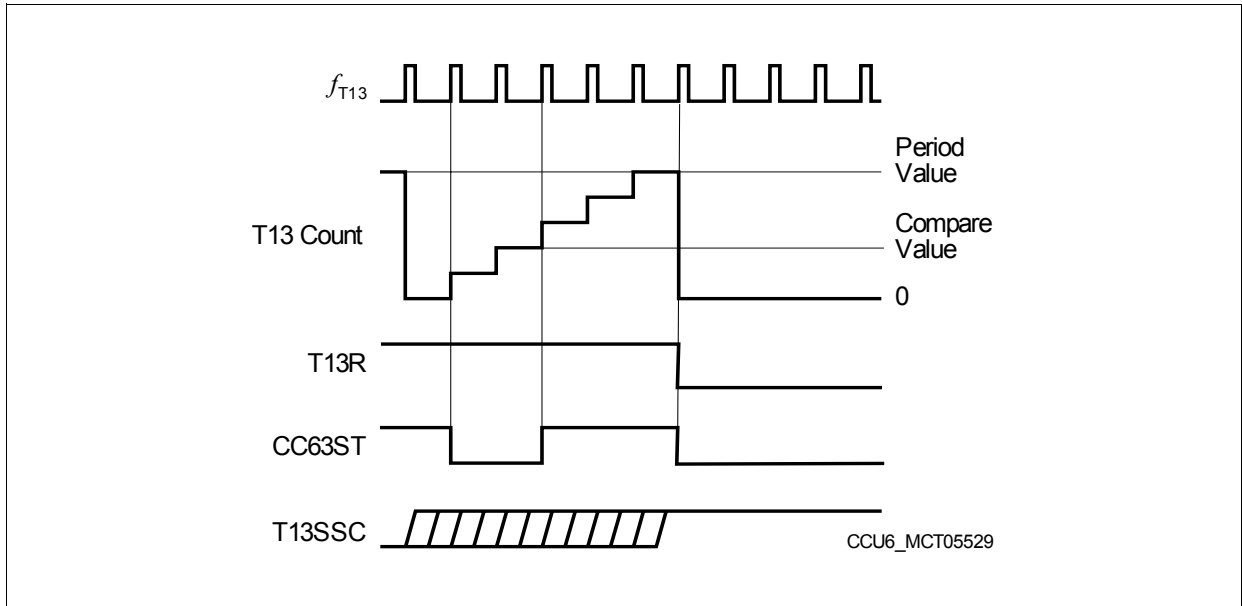
The behavior of T13 is illustrated in **Figure 20-25**.



**Figure 20-25 T13 Counting Sequence**

### 20.3.2.3 Single-Shot Mode

In Single-Shot Mode, the timer run bit T13R is cleared by hardware. If bit T13SSC = 1, the timer T13 will stop when the current timer period is finished.

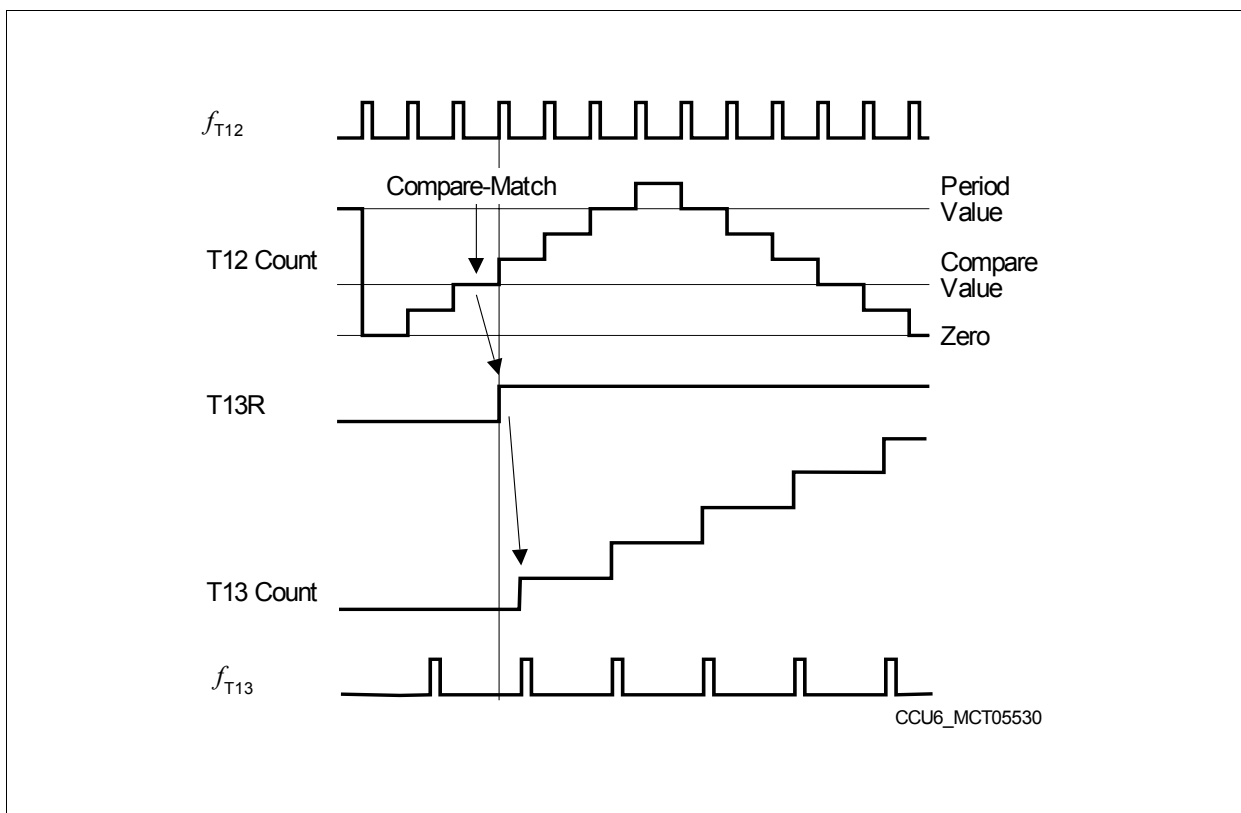


**Figure 20-26 Single-Shot Operation of Timer T13**

### 20.3.2.4 Synchronization to T12

Timer T13 can be synchronized to a T12 event. Bit fields T13TEC and T13TED select the event that is used to start Timer T13. The selected event sets bit T13R via HW, and T13 starts counting. Combined with the Single-Shot mode, this feature can be used to generate a programmable delay after a T12 event.

**Figure 20-27** shows an example for the synchronization of T13 to a T12 event. Here, the selected event is a compare-match (compare value = 2) while counting up. The clocks of T12 and T13 can be different (other prescaler factor); the figure shows an example in which T13 is clocked with half the frequency of T12.



**Figure 20-27 Synchronization of T13 to T12 Compare Match**

Bit field T13TEC selects the trigger event to start T13 (automatic set of T13R for synchronization to T12 compare signals) according to the combinations shown in [Table 20-7](#). Bit field T13TED additionally specifies for which count direction of T12 the selected trigger event should be regarded (see [Table 20-8](#)).

**Table 20-7 T12 Trigger Event Selection**

<b>T13TEC</b>	<b>Selected Event</b>
000 <sub>B</sub>	None
001 <sub>B</sub>	T12 Compare Event on Channel 0 (CM_CC60)
010 <sub>B</sub>	T12 Compare Event on Channel 1 (CM_CC61)
011 <sub>B</sub>	T12 Compare Event on Channel 2 (CM_CC62)
100 <sub>B</sub>	T12 Compare Event on any Channel (0, 1, 2)
101 <sub>B</sub>	T12 Period-Match (T12_PM)
110 <sub>B</sub>	T12 Zero-Match while counting up (T12_ZM and CDIR = 0)
111 <sub>B</sub>	Any Hall State Change

**Table 20-8 T12 Trigger Event Additional Specifier**

<b>T13TED</b>	<b>Selected Event Specifier</b>
00 <sub>B</sub>	Reserved, no action
01 <sub>B</sub>	Selected event is active while T12 is counting up (CDIR = 0)
10 <sub>B</sub>	Selected event is active while T12 is counting down (CDIR = 1)
11 <sub>B</sub>	Selected event is active independently of the count direction of T12

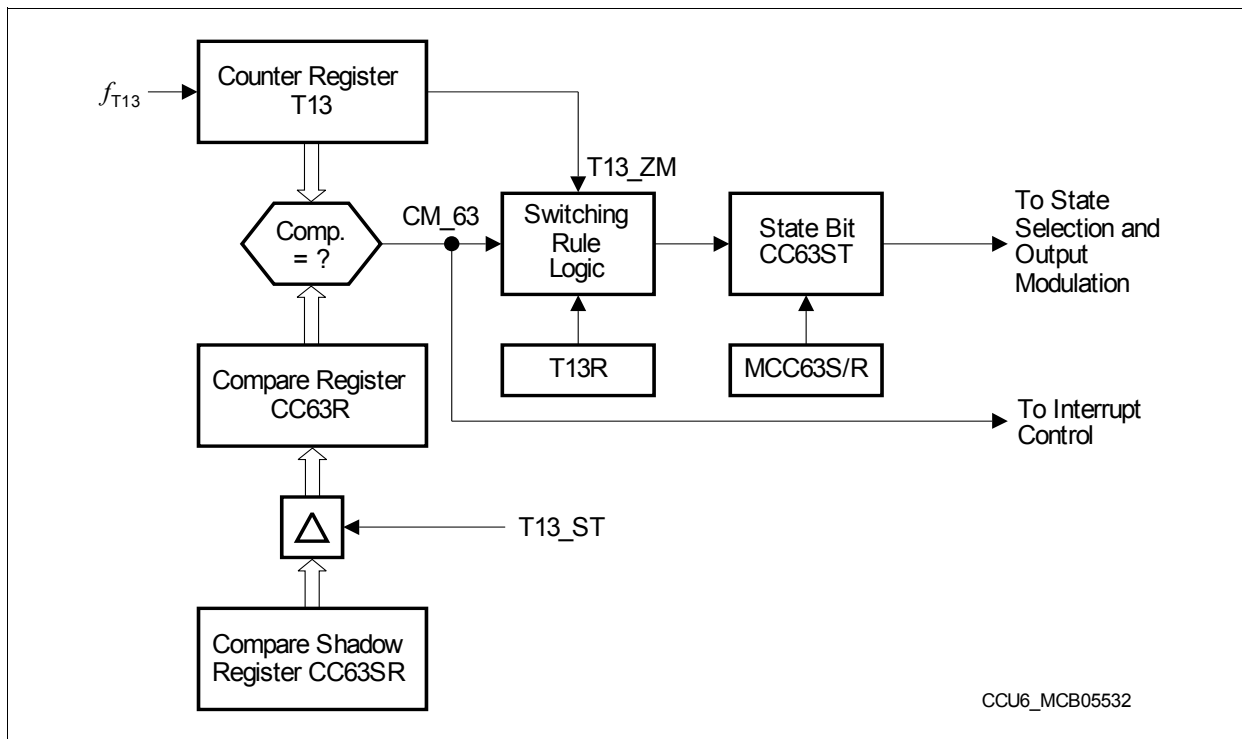
### 20.3.3 T13 Compare Mode

Associated with Timer T13 is one compare channel, that can perform compare operations with regard to the contents of the T13 counter.

**Figure 20-23** gives an overview on the T13 channel in Compare Mode. The channel is connected to the T13 counter register via an equal-to comparator, generating a compare match signal when the contents of the counter matches the contents of the compare register.

The channel consists of the comparator and a double register structure - the actual compare register, **CC63R**, feeding the comparator, and an associated shadow register, **CC63SR**, that is preloaded by software and transferred into the compare register when signal T13 shadow transfer, T13\_ST, gets active. Providing a shadow register for the compare value as well as for other values related to the generation of the PWM signal facilitates a concurrent update by software for all relevant parameters.

Associated with the channel is a State Bit, **CMPSTAT.CC63ST**, holding the status of the compare operation. **Figure 20-28** gives an overview on the logic for the State Bit.



**Figure 20-28 T13 State Bit Block Diagram**

A compare interrupt event **CM\_63** is signaled when a compare match is detected. The actual setting of a State Bit has no influence on the interrupt generation.

The inputs to the switching rule logic for the **CC63ST** bit are the timer run bit (**T13R**), the timer zero-match signal (**T13\_ZM**), and the actual individual compare-match signal **CM\_63**. In addition, the state bit can be set or cleared by software via bits **MCC63S** and



MCC63R in register **CMPMODIF**.

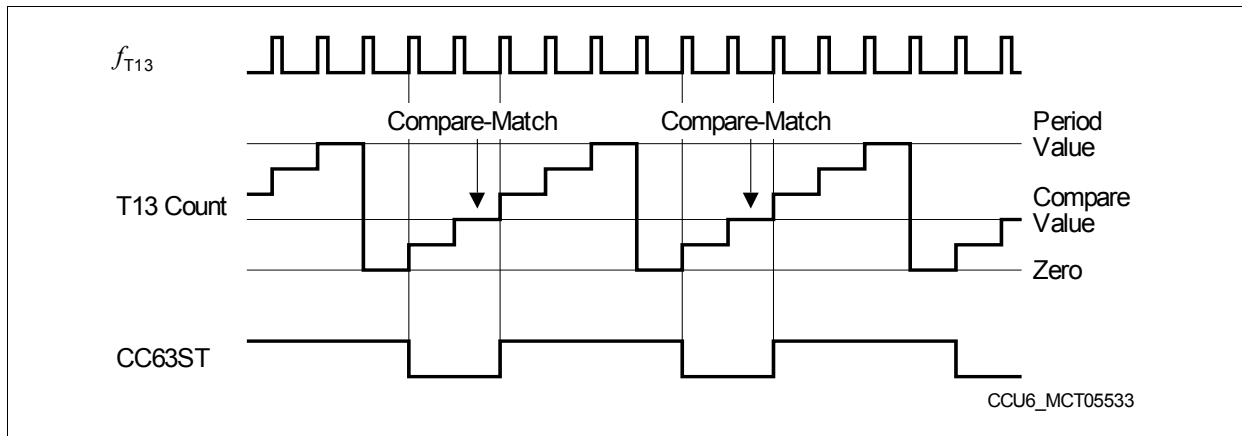
A modification of the State Bit CC63ST by hardware is only possible while Timer T13 is running ( $T13R = 1$ ). If this is the case, the following switching rules apply for setting and resetting the State Bit in Compare Mode:

State Bit **CC63ST** is **set** to 1

- with the next T13 clock ( $f_{T13}$ ) after a compare-match (T13 is always counting up) (i.e., when the counter is incremented above the compare value);
- with the next T13 clock ( $f_{T13}$ ) after a zero-match AND a parallel compare-match.

State Bit **CC63ST** is **cleared** to 0

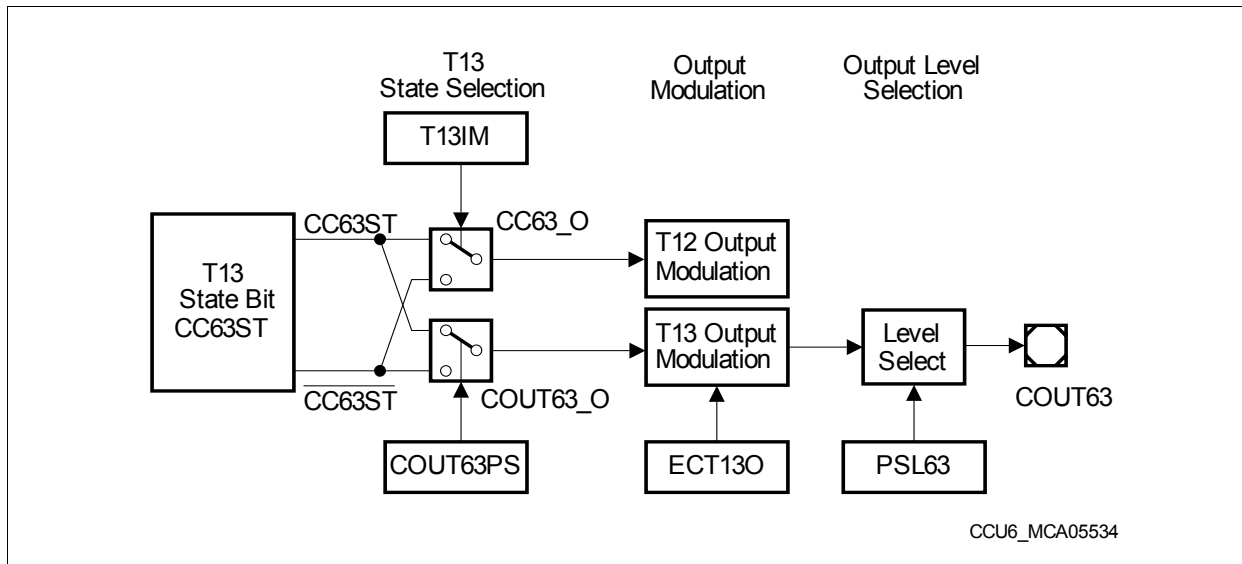
- with the next T13 clock ( $f_{T13}$ ) after a zero-match AND NO parallel compare-match.



**Figure 20-29 T13 Compare Operation**

### 20.3.4 Compare Mode Output Path

**Figure 20-30** gives an overview on the signal path from the channel State Bit CC63ST to its output pin COUT63. As illustrated, a user can determine the desired output behavior in relation to the current state of CC63ST. Please refer to **Section 20.2.4.3** for detailed information on the output modulation for T12 signals.



**Figure 20-30 Channel 63 Output Path**

The output line COUT63\_O can generate a T13 PWM at the output pin COUT63. The signal CC63\_O can be used to modulate the T12-related output signals with a T13 PWM. In order to decouple COUT63 from the internal modulation, the compare state leading to an active signal can be selected independently by bits T13IM and COUT63PS.

The last block of the data path is the Output Modulation block. Here, the modulation source T13 and the trap functionality are combined and control the actual level of the output pin COUT63 (see **Figure 20-31**):

- The **T13 related compare signal** COUT63\_O delivered by the T13 state selection with the enable bit **MODCTR.ECT13O**
- The **trap state** TRPS with an individual enable bit **TRPCTR.TRPEN13**

If the modulation input signal COUT63\_O is enabled (ECT13O = 1) and is at passive state, the modulated is also in passive state. If the modulation input is not enabled, the output is in passive state.

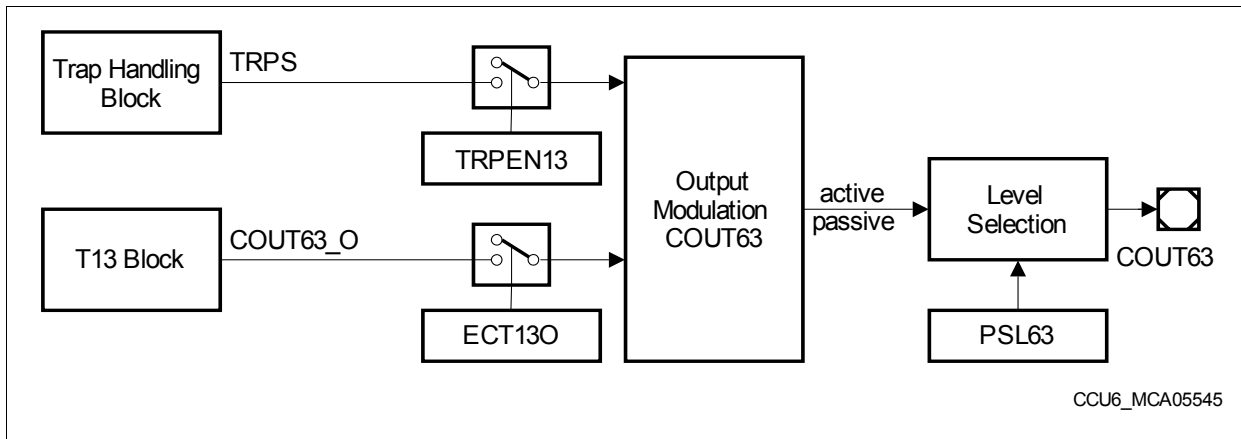
If the Trap State is active (TRPS = 1), then the output enabled for the trap signal (by TRPEN13 = 1) is set to the passive state.

The output of the modulation control block is connected to a level select block. It offers the option to determine the actual output level of a pin, depending on the state of the output line (decoupling of active/passive state and output polarity) as specified by the Passive State Select bit **PSLR.PSL63**. If the modulated output signal is in the passive

## Capture/Compare Unit 6 (CCU6)

state, the level specified directly by PSL63 is output. If it is in the active state, the inverted level of PSL63 is output. This allows the user to adapt the polarity of an active output signal to the connected circuitry.

The PSL63 bit has a shadow register to allow for updates with the T13 shadow transfer signal (T13\_ST) without undesired pulses on the output lines. A read action returns the actually used value, whereas a write action targets the shadow bit. Providing a shadow register for the PSL value as well as for other values related to the generation of the PWM signal facilitates a concurrent update by software for all relevant parameters.



**Figure 20-31 T13 Output Modulation**

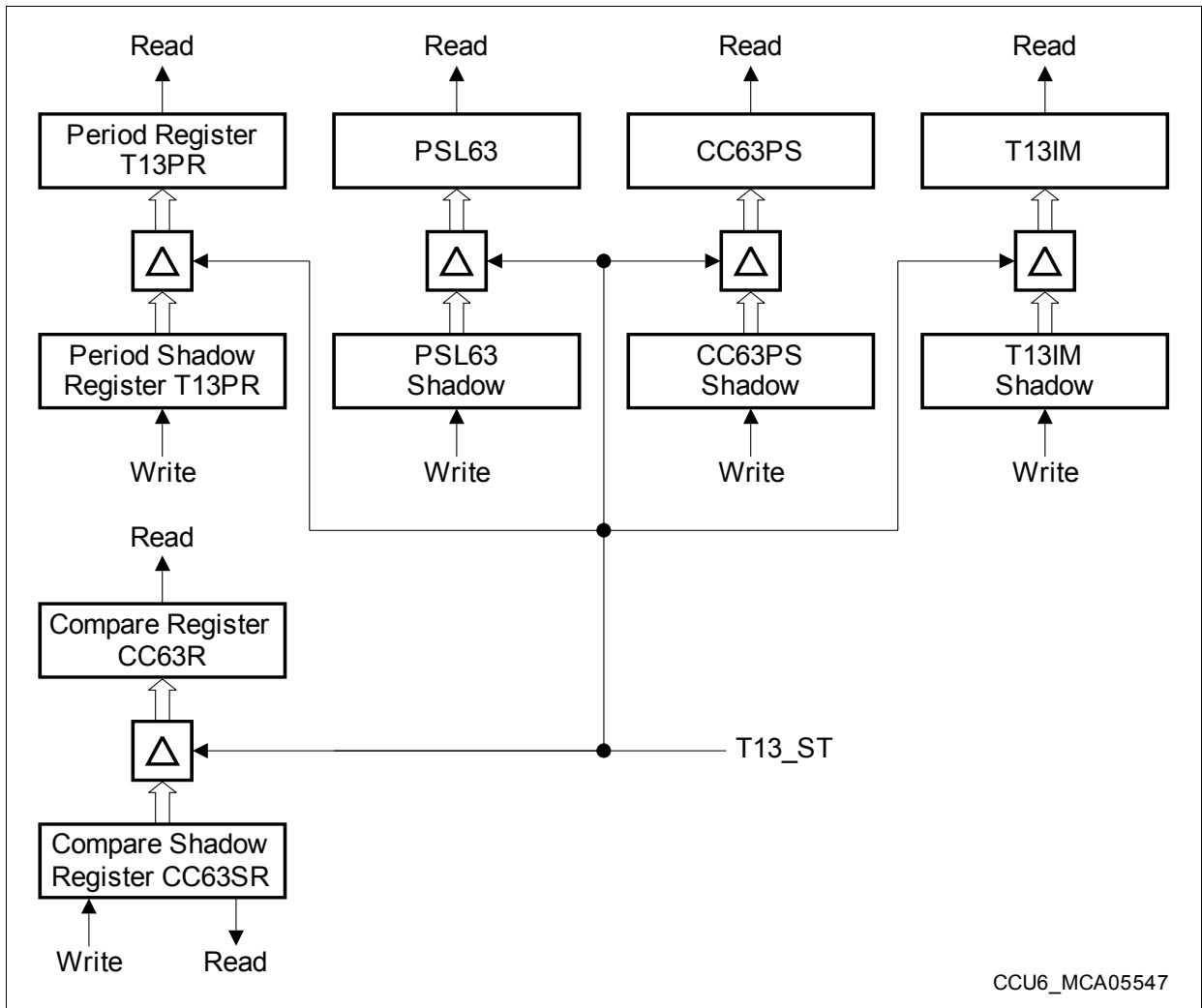
### 20.3.5 T13 Shadow Register Transfer

A special shadow transfer signal (T13\_ST) can be generated to facilitate updating the period and compare values of the compare channel CC63 synchronously to the operation of T13. Providing a shadow register for values defining one PWM period facilitates a concurrent update by software for all relevant parameters. The next PWM period can run with a new set of parameters. The generation of this signal is requested by software via bit **TCTR0.STE13** (set by writing 1 to the write-only bit **TCTR4.T13STR**, cleared by writing 1 to the write-only bit **TCTR4.T13STD**).

When signal T13\_ST is active, a shadow register transfer is triggered with the next cycle of the T13 clock. Bit STE13 is automatically cleared with the shadow register transfer.

A T13 shadow register transfer takes place (T13\_ST active):

- while timer T13 is not running (T13R = 0), or
- STE13 = 1 and a Period-Match is detected while T13R = 1



**Figure 20-32 T13 Shadow Register Overview**

## **20.3.6 T13 related Registers**

### **20.3.6.1 T13 Counter Register**

The generation of the patterns for a single channel pulse width modulation (PWM) is based on timer T13. The registers related to timer T13 can be concurrently updated (with well-defined conditions) in order to ensure consistency of the PWM signal. T13 can be synchronized to several timer T12 events.

Timer T13 only supports compare mode on its compare channel CC63.

Register T13 represents the counting value of timer T13. It can only be written while the timer T13 is stopped. Write actions while T13 is running are not taken into account. Register T13 can always be read by SW.

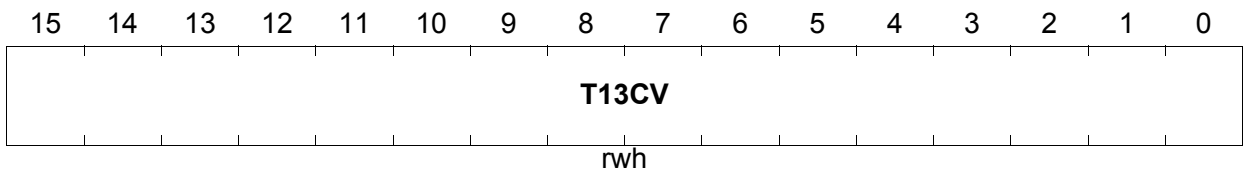
Timer T13 only supports edge-aligned mode (counting up).

#### **T13**

**Timer T13 Counter Register**

**XSFR(30<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
<b>T13CV</b>	[15:0]	rwh	<b>Timer 13 Counter Value</b> This register represents the 16-bit counter value of Timer13.

*Note: While timer T13 is stopped, the internal clock divider is reset in order to ensure reproducible timings and delays.*

### 20.3.6.2 Period Register

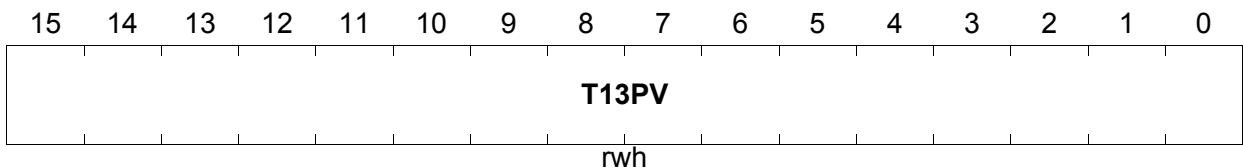
Register T13PR contains the period value for timer T13. The period value is compared to the actual counter value of T13 and the resulting counter actions depend on the defined counting rules. This register has a shadow register and the shadow transfer is controlled by bit STE13. A read action by SW delivers the value currently used for the compare action, whereas the write action targets a shadow register. The shadow register structure allows a concurrent update of all T13-related values.

#### T13PR

**Timer 13 Period Register**

**XSFR(32<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
T13PV	[15:0]	rwh	<b>T13 Period Value</b> The value T13PV defines the counter value for T13 leading to a period-match. When reaching this value, the timer T13 is set to zero.

### 20.3.6.3 Compare Register

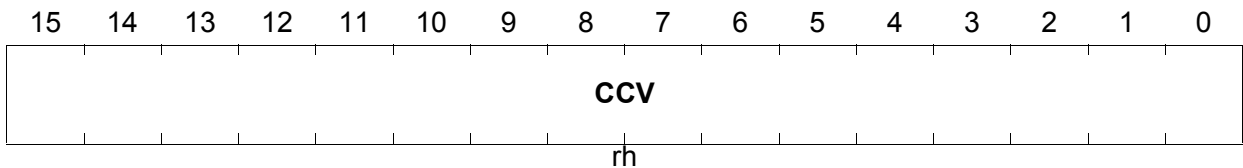
Registers CC63R is the actual compare register for T13. The values stored in CC63R is compared to the counter value of T13. The State Bit CC63ST is located in register **CMPSTAT**.

#### CC63R

**Compare Register for T13**

**XSFR(34<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
CCV	[15:0]	rh	<b>Channel CC63 Compare Value</b> The bit field CCV contains the value, that is compared to the T13 counter value.

### 20.3.6.4 Compare Shadow Register

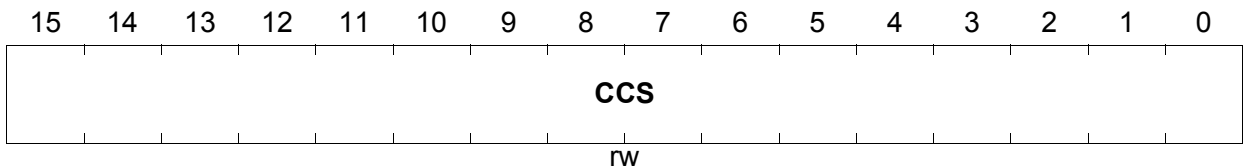
The register CC63R can only be read by SW, the modification of the value is done by a shadow register transfer from register CC63SR. The corresponding shadow register CC63SR can be read and written by SW.

#### CC63SR

**Compare Shadow Register for T13**

**XSFR(36<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
CCS	[15:0]	rw	<b>Shadow Register for Channel CC63 Compare Value</b> The bit field contents of CCS is transferred to the bit field CCV during a shadow transfer.

## 20.4 Trap Handling

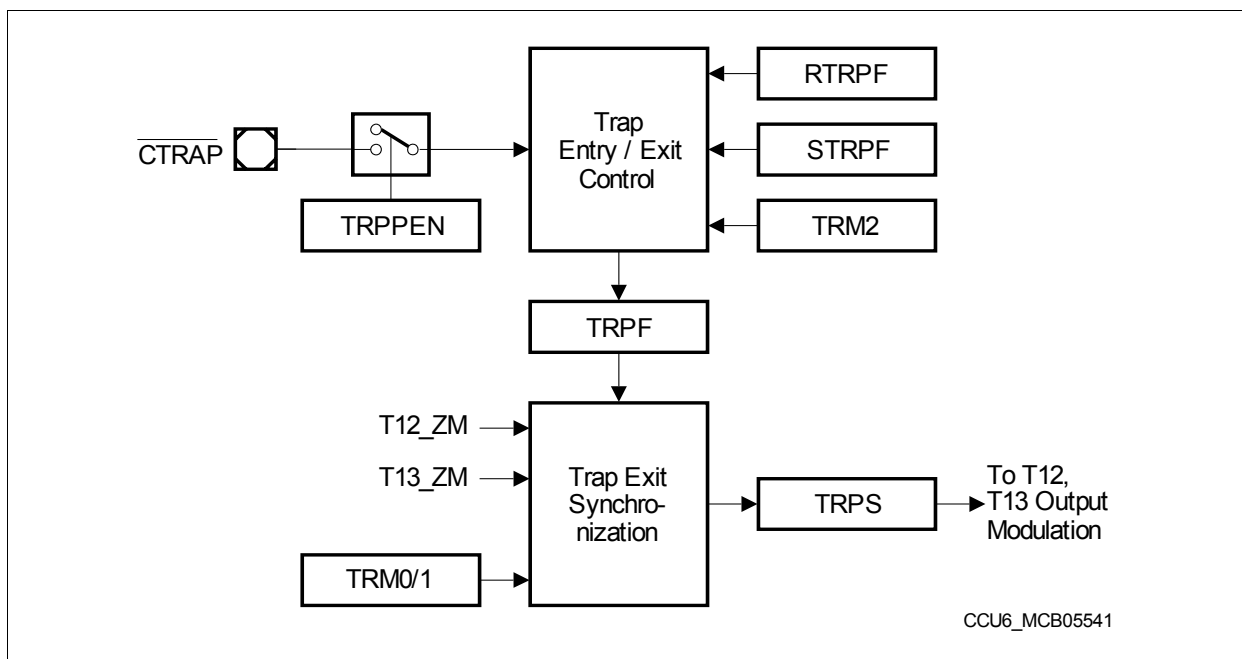
The trap functionality permits the PWM outputs to react on the state of the input signal  $\overline{\text{CTRAP}}$ . This functionality can be used to switch off the power devices if the trap input becomes active (e.g. to perform an emergency stop). The trap handling and the effect on the output modulation are controlled by the bits in the trap control register **TRPCTR**. The trap flags TRPF and TRPS are located in register **IS** and can be set/cleared by SW by writing to registers **ISS** and **ISR**.

**Figure 20-33** gives an overview on the trap function.

The Trap Flag TRPF monitors the trap input and initiates the entry into the Trap State. The Trap State Bit TRPS determines the effect on the outputs and controls the exit of the Trap State.

When a trap condition is detected ( $\overline{\text{CTRAP}} = 0$ ) and the input is enabled (TRPPEN = 1), both, the Trap Flag TRPF and the Trap State Bit TRPS, are set to 1 (trap state active). The output of the Trap State Bit TRPS leads to the Output Modulation Blocks (for T12 and for T13) and can there deactivate the outputs (set them to the passive state). Individual enable control bits for each of the six T12-related outputs and the T13-related output facilitate a flexible adaptation to the application needs.

There are a number of different ways to exit the Trap State. This offers SW the option to select the best operation for the application. Exiting the Trap State can be done either immediately when the trap condition is removed ( $\overline{\text{CTRAP}} = 1$  or TRPPEN = 0), or under software control, or synchronously to the PWM generated by either Timer T12 or Timer T13.



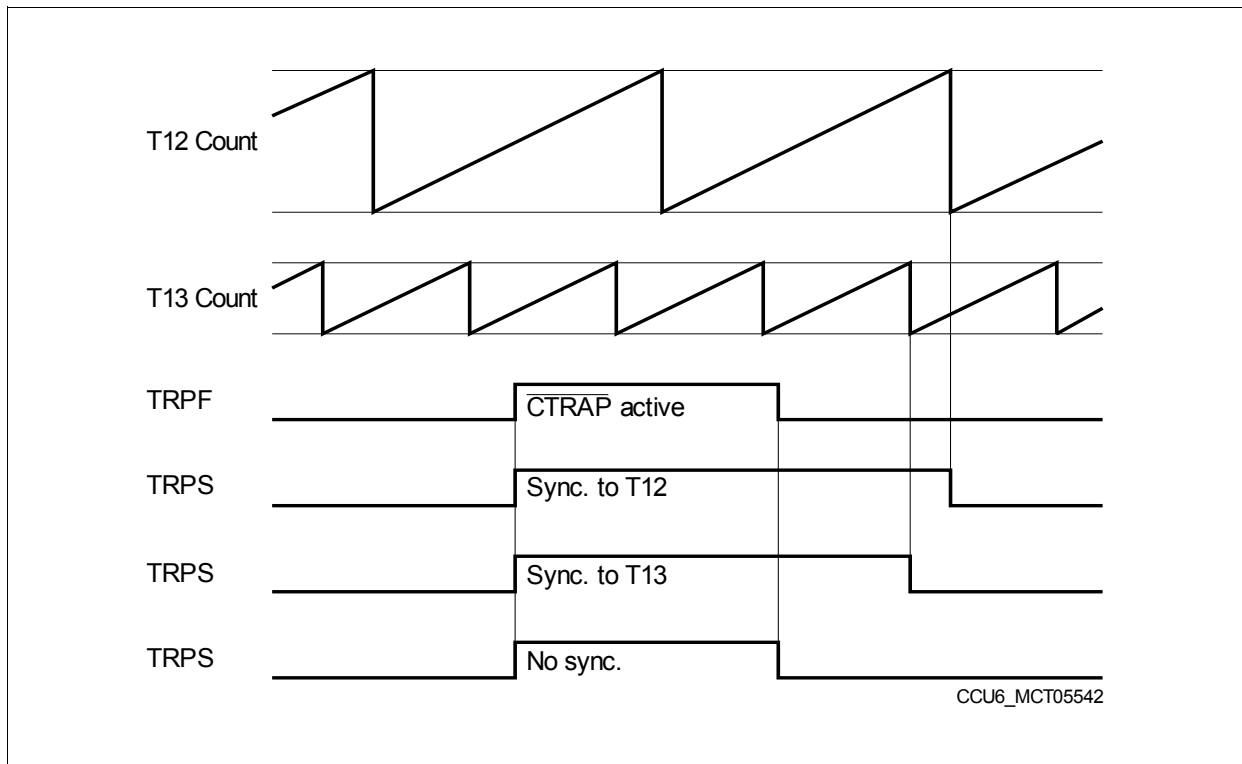
**Figure 20-33 Trap Logic Block Diagram**



**Capture/Compare Unit 6 (CCU6)**

Clearing of TRPF is controlled by the mode control bit TRPM2. If  $\text{TRPM2} = 0$ , TRPF is automatically cleared by HW when  $\text{CTRAP}$  returns to the inactive level ( $\text{CTRAP} = 1$ ) or if the trap input is disabled ( $\text{TRPPEN} = 0$ ). When  $\text{TRPM2} = 1$ , TRPF must be reset by SW after  $\text{CTRAP}$  has become inactive.

Clearing of TRPS is controlled by the mode control bits TRPM1 and TRPM0 (located in the Trap Control Register TRPCTR). A reset of TRPS terminates the Trap State and returns to normal operation. There are three options selected by TRPM1 and TRPM0. One is that the Trap State is left immediately when the Trap Flag TRPF is cleared, without any synchronization to timers T12 or T13. The other two options facilitate the synchronization of the termination of the Trap State to the count periods of either Timer T12 or Timer T13. **Figure 20-34** gives an overview on the associated operation.

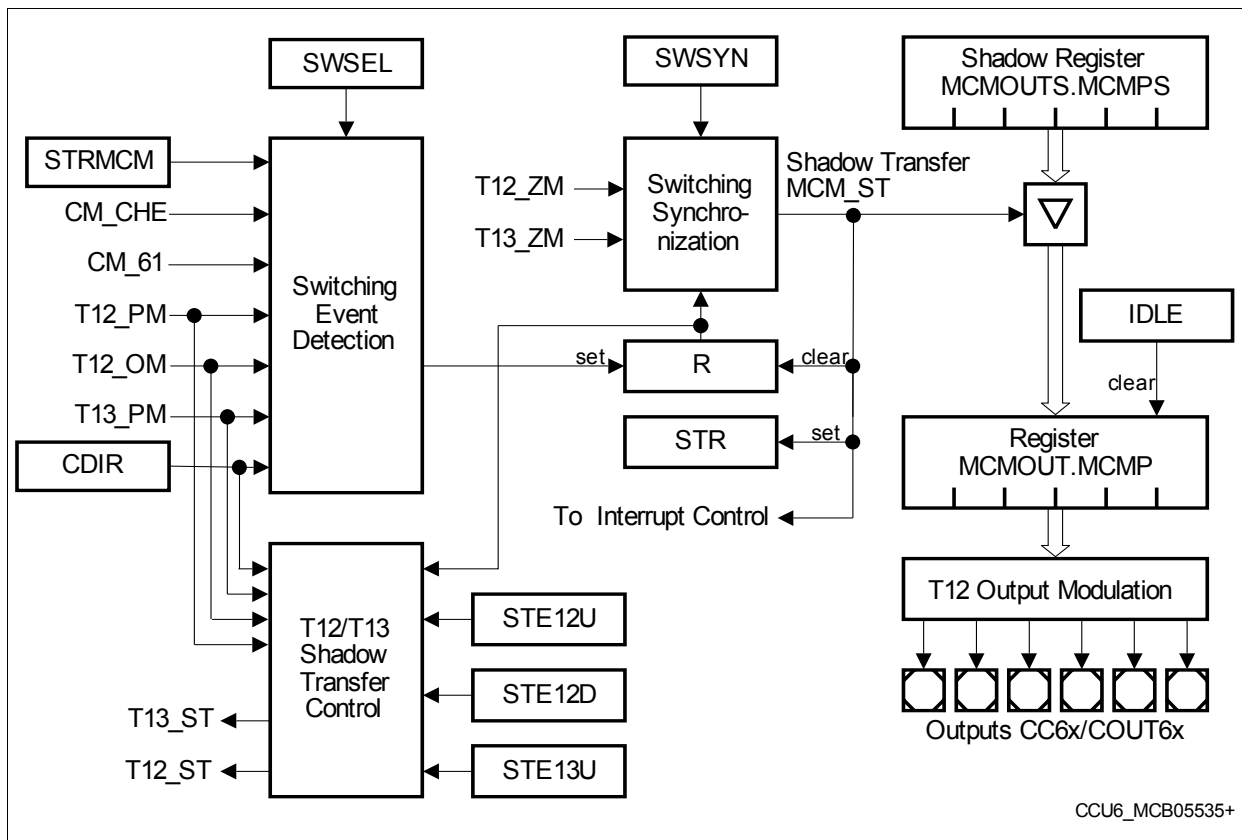


**Figure 20-34 Trap State Synchronization (with TRM2 = 0)**

## 20.5 Multi-Channel Mode

The Multi-Channel mode offers the possibility to modulate all six T12-related output signals with one instruction. The bits in bit field **MCMOUT**.MCMP are used to specify the outputs that may become active. If Multi-Channel mode is enabled (bit **MODCTR**.MCMEN = 1), only those outputs may become active, that have a 1 at the corresponding bit position in bit field MCMP.

This bit field has its own shadow bit field **MCMOUTS**.MCMPS, that can be written by software. The transfer of the new value in MCMPS to the bit field MCMP can be triggered by, and synchronized to, T12 or T13 events. This structure permits the software to write the new value, that is then taken into account by the hardware at a well-defined moment and synchronized to a PWM signal. This avoids unintended pulses due to unsynchronized modulation sources.



**Figure 20-35 Multi-Channel Mode Block Diagram**

**Figure 20-35** shows the functional blocks for the Multi-Channel operation, controlled by bit fields in register **MCMCTR**. The event that triggers the update of bit field MCMP is chosen by SWSEL. In order to synchronize the update of MCMP to a PWM generated by T12 or T13, bit field SWSYN allows the selection of the synchronization event leading to the transfer from MCMPS to MCMP. Due to this structure, an update takes place with a new PWM period. A reminder flag R is set when the selected switching event occurs

**Capture/Compare Unit 6 (CCU6)**

(the event is not necessarily synchronous to the modulating PWM), and is cleared when the transfer takes place. This flag can be monitored by software to check for the status of this logic block. If the shadow transfer from MCMPS to MCMP takes place, bit **IS**.STR becomes set and an interrupt can be generated.

In addition to the Multi-Channel shadow transfer event MCM\_ST, the shadow transfers for T12 (T12\_ST) and T13 (T13\_ST) can be generated to allow concurrent updates of applied duty cycles for T12 and/or T13 modulation and Multi-Channel patterns.

If it is explicitly desired, the update takes place immediately with the occurrence of the selected event when the direct synchronization mode is selected. The update can also be requested by software by writing to bit field MCMPS with the shadow transfer request bit STRMCM = 1. The option to trigger an update by SW is possible for all settings of SWSEL.

By using the direct mode and bit STRMCM = 1, the update takes place completely under software control.

The event selection and synchronization options are summarized in [Table 20-9](#) and [Table 20-10](#).

**Table 20-9 Multi-Channel Mode Switching Event Selection**

<b>SWSEL</b>	<b>Selected Event (see register <a href="#">MCMCTR</a>)</b>
000 <sub>B</sub>	No automatic event detection
001 <sub>B</sub>	Correct Hall Event (CM_CHE) detected at input signals CCPOSx without additional delay
010 <sub>B</sub>	T13 Period-Match (T13_PM)
011 <sub>B</sub>	T12 One-Match while counting down (T12_OM and CDIR = 1)
100 <sub>B</sub>	T12 Compare Channel 1 Event while counting up (CM_61 and CDIR = 0) to support the phase delay function by CC61 for block commutation mode.
101 <sub>B</sub>	T12 Period-Match while counting up (T12_PM and CDIR = 0)
110 <sub>B</sub> , 111 <sub>B</sub>	Reserved, no action

**Table 20-10 Multi-Channel Mode Switching Synchronization**

<b>SWSYN</b>	<b>Synchronization Event (see register <a href="#">MCMCTR</a>)</b>
00 <sub>B</sub>	Direct Mode: the trigger event directly causes the shadow transfer
01 <sub>B</sub>	T13 Zero-Match (T13_ZM), the MCM shadow transfer is synchronized to a T13 PWM

**Table 20-10 Multi-Channel Mode Switching Synchronization (cont'd)**

<b>SWSYN</b>	<b>Synchronization Event (see register <a href="#">MCMCTR</a>)</b>
10 <sub>B</sub>	T12 Zero-Match (T12_ZM), the MCM shadow transfer is synchronized to a T12 PWM
11 <sub>B</sub>	Reserved, no action

## **20.6 Hall Sensor Mode**

For Brushless DC-Motors in block commutation mode, the Multi-Channel Mode has been introduced to provide efficient means for switching pattern generation. These patterns need to be output in relation to the angular position of the motor. For this, usually Hall sensors or Back-EMF sensing are used to determine the angular rotor position. The CCU6 provides three inputs, CCPOS0, CCPOS1, and CCPOS2, that can be used as inputs for the Hall sensors or the Back-EMF detection signals.

There is a strong correlation between the motor position and the output modulation pattern. When a certain position of the motor has been reached, indicated by the sampled Hall sensor inputs (the Hall pattern), the next, pre-determined Multi-Channel Modulation pattern has to be output. Because of different machine types, the modulation pattern for driving the motor can vary. Therefore, it is wishful to have a wide flexibility in defining the correlation between the Hall pattern and the corresponding Modulation pattern. Furthermore, a hardware mechanism significantly reduces the CPU for block-commutation.

The CCU6 offers the flexibility by having a register containing the currently assumed Hall pattern (CURH), the next expected Hall pattern (EXPH) and the corresponding output pattern (MCMOUT). A new Modulation pattern is output when the sampled Hall inputs match the expected ones (EXPH). To detect the next rotation phase (segment for block commutation), the CCU6 monitors the Hall inputs for changes. When the next expected Hall pattern is detected, the next corresponding Modulation pattern is output.

To increase for noise immunity (to a certain extend), the CCU6 offers the possibility to introduce a sampling delay for the Hall inputs. Some changes of the Hall inputs are not leading to the expected Hall pattern, because they are only short spikes due to noise. The Hall pattern compare logic compares the Hall inputs to the next expected pattern and also to the currently assumed pattern to filter out spikes.

For the Hall and Modulation output patterns, a double-register structure is implemented. While register **MCMOUT** holds the actually used values, its shadow register **MCMOUTS** can be loaded by software from a pre-defined table, holding the appropriate Hall and Modulation patterns for the given motor control.

A transfer from the shadow register into register MCMOUT can take place when a correct Hall pattern change is detected. Software can then load the next values into register MCMOUTS. It is also possible by software to force a transfer from MCMOUTS into MCMOUT.

*Note: The Hall input signals CCPOSx and the CURH and EXPH bit fields are arranged in the following order:*

*CCPOS0 corresponds to CURH.0 (LSB) and EXPH.0 (LSB)*

*CCPOS1 corresponds to CURH.1 and EXPH.1*

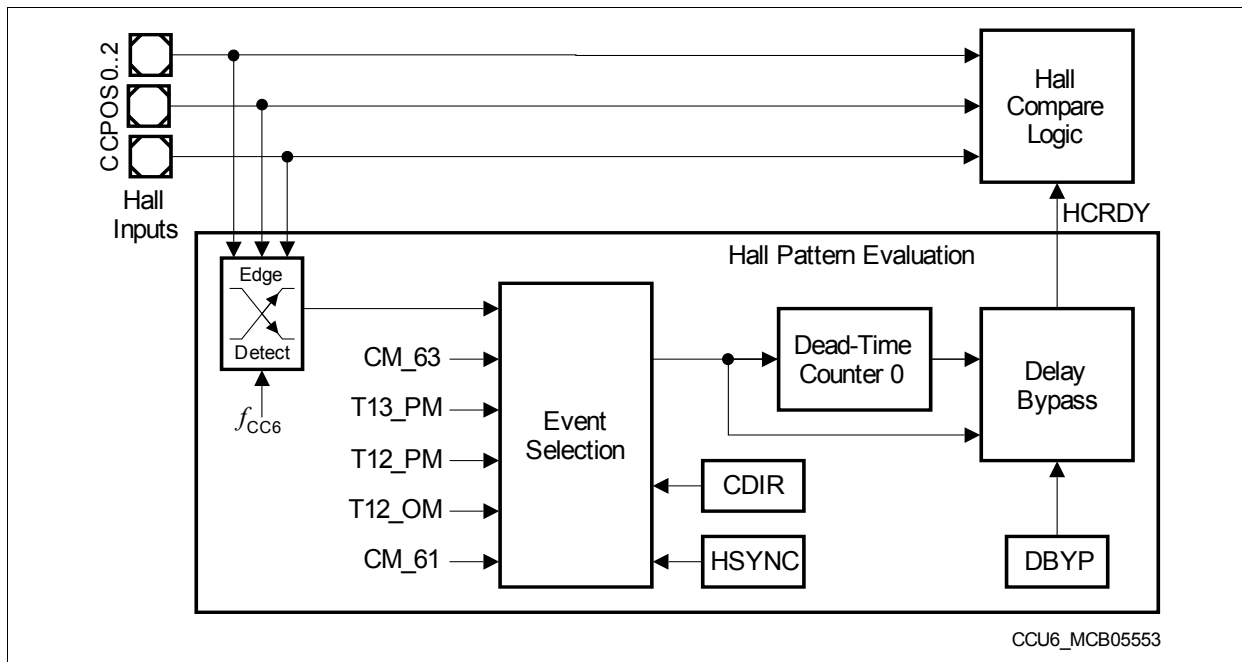
*CCPOS2 corresponds to CURH.2 (MSB) and EXPH.2 (MSB)*

### 20.6.1 Hall Pattern Evaluation

The Hall sensor inputs CCPOSx can be permanently monitored via an edge detection block (with the module clock  $f_{CC6}$ ). In order to suppress spikes on the Hall inputs due to noise in rugged inverter environment, two optional noise filtering methods are supported by the Hall logic (both methods can be combined).

- **Noise filtering with delay:**  
 For this function, the mode control bit fields MSEL6x for all T12 compare channels must be programmed to 1000<sub>B</sub> and DBYP = 0. The selected event triggers Dead-Time Counter 0 to generate a programmable delay (defined by bit field DTM). When the delay has elapsed, the evaluation signal HCRDY becomes activated. Output modulation with T12 PWM signals is not possible in this mode.
- **Noise filtering by synchronization to PWM:**  
 The Hall inputs are not permanently monitored by the edge detection block, but samples are taken only at defined points in time during a PWM period. This can be used to sample the Hall inputs when the switching noise (due to PWM) does not disturb the Hall input signals.

If neither the delay function of Dead-Time Counter 0 is not used for the Hall pattern evaluation nor the Hall mode for Brushless DC-Drive control is enabled, the timer T12 block is available for PWM generation and output modulation.



**Figure 20-36 Hall Pattern Evaluation**

If the evaluation signal HCRDY (Hall Compare Ready, see [Figure 20-37](#)) becomes activated, the Hall inputs are sampled and the Hall compare logic starts the evaluation of the Hall inputs.

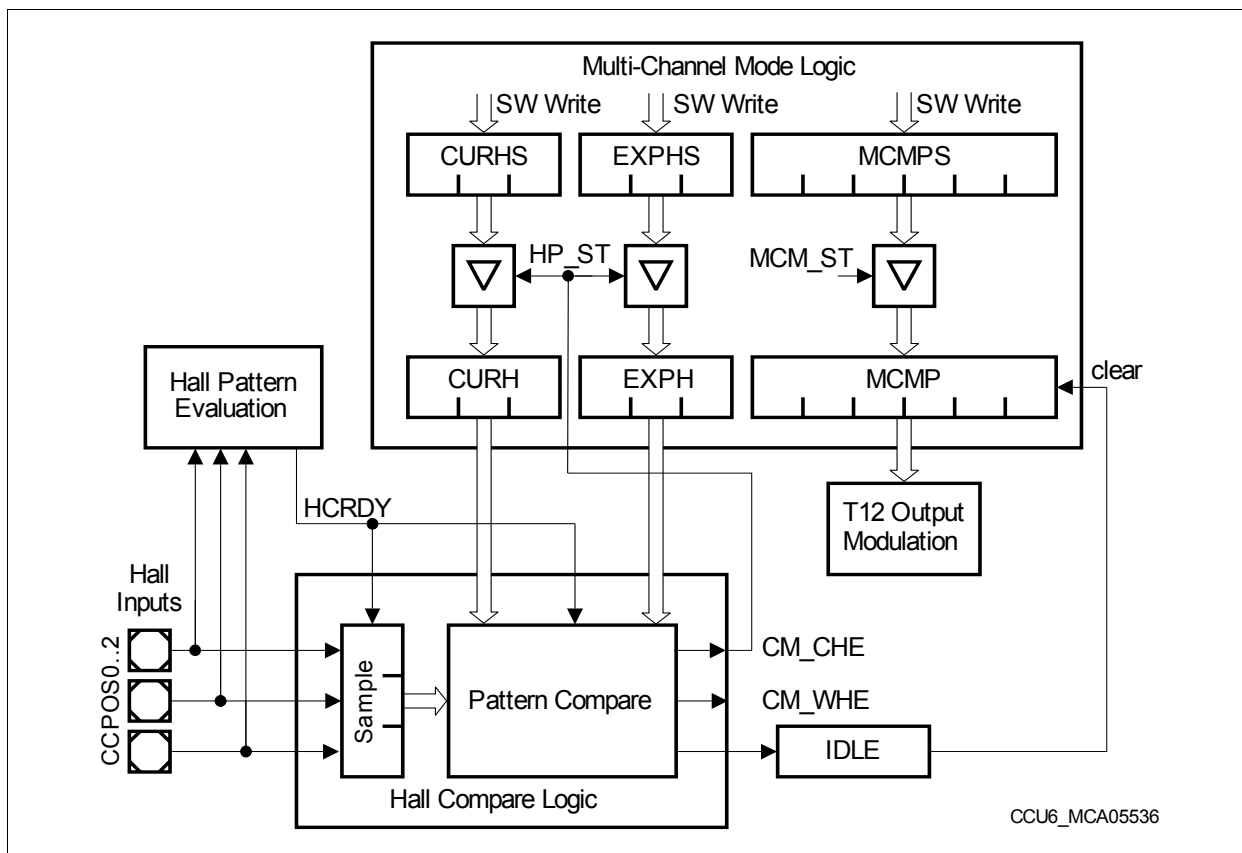
**Figure 20-36** illustrates the events for Hall pattern evaluation and the noise filter logic, **Table 20-11** summarizes the selectable trigger input signals.

**Table 20-11 Hall Sensor Mode Trigger Event Selection**

<b>HSYNC</b>	<b>Selected Event (see register <a href="#">T12MSEL</a>)</b>
000 <sub>B</sub>	Any edge at any of the inputs CCPOSx, independent from any PWM signal (permanent check).
001 <sub>B</sub>	A T13 Compare-Match (CM_63).
010 <sub>B</sub>	A T13 Period-Match (T13_PM).
011 <sub>B</sub>	Hall sampling triggered by HW sources is switched off.
100 <sub>B</sub>	A T12 Period-Match while counting up (T12_PM and CDIR = 0).
101 <sub>B</sub>	A T12 One-Match while counting down (T12_OM and CDIR = 1).
110 <sub>B</sub>	A T12 Compare-Match of compare channel CC61 while counting up (CM_61 and CDIR = 0).
111 <sub>B</sub>	A T12 Compare-Match of compare channel CC61 while counting down (CM_61 and CDIR = 1).

## 20.6.2 Hall Pattern Compare Logic

**Figure 20-37** gives an overview on the double-register structure and the pattern compare logic. Software writes the next modulation pattern (MCMPS) and the corresponding current (CURHS) and expected (EXPHS) Hall patterns into the shadow register MCMOUTS. Register MCMOUT holds the actually used values CURH and EXPH. The modulation pattern MCMPS is provided to the T12 Output Modulation block. The current (CURH) and expected (EXPH) Hall patterns are compared to the sampled Hall sensor inputs (visible in register **CMPSTAT**). Sampling of the inputs and the evaluation of the comparator outputs is triggered by the evaluation signal HCRDY (Hall Compare Ready), that is detailed in the next section.



**Figure 20-37 Hall Pattern Compare Logic**

- If the sampled Hall pattern matches the value programmed in CURH, the detected transition was a spike (no Hall event) and no further actions are necessary.
- If the sampled Hall pattern matches the value programmed in EXPH, the detected transition was the expected event (correct Hall event CM\_CHE) and the MCM value has to change.
- If the sampled Hall pattern matches neither CURH nor EXPH, the transition was due to a major error (wrong Hall event CM\_CWE) and can lead to an emergency shut down (IDLE).



At every correct Hall event (CM\_CHE), the next Hall patterns are transferred from the shadow register MCMOUTS into MCMOUT (Hall pattern shadow transfer HP\_ST), and a new Hall pattern with its corresponding output pattern can be loaded (e.g. from a predefined table in memory) by software into MCMOUTS. For the Modulation patterns, signal MCM\_ST is used to trigger the transfer.

Loading this shadow register can also be done by writing MCMOUTS.STRHP = 1 (for EXPH and CURH) or MCMOUTS.STRMCMP = 1 (for MCMP).

### 20.6.3 Hall Mode Flags

Depending on the Hall pattern compare operation, a number of flags are set in order to indicate the status of the module and to trigger further actions and interrupt requests.

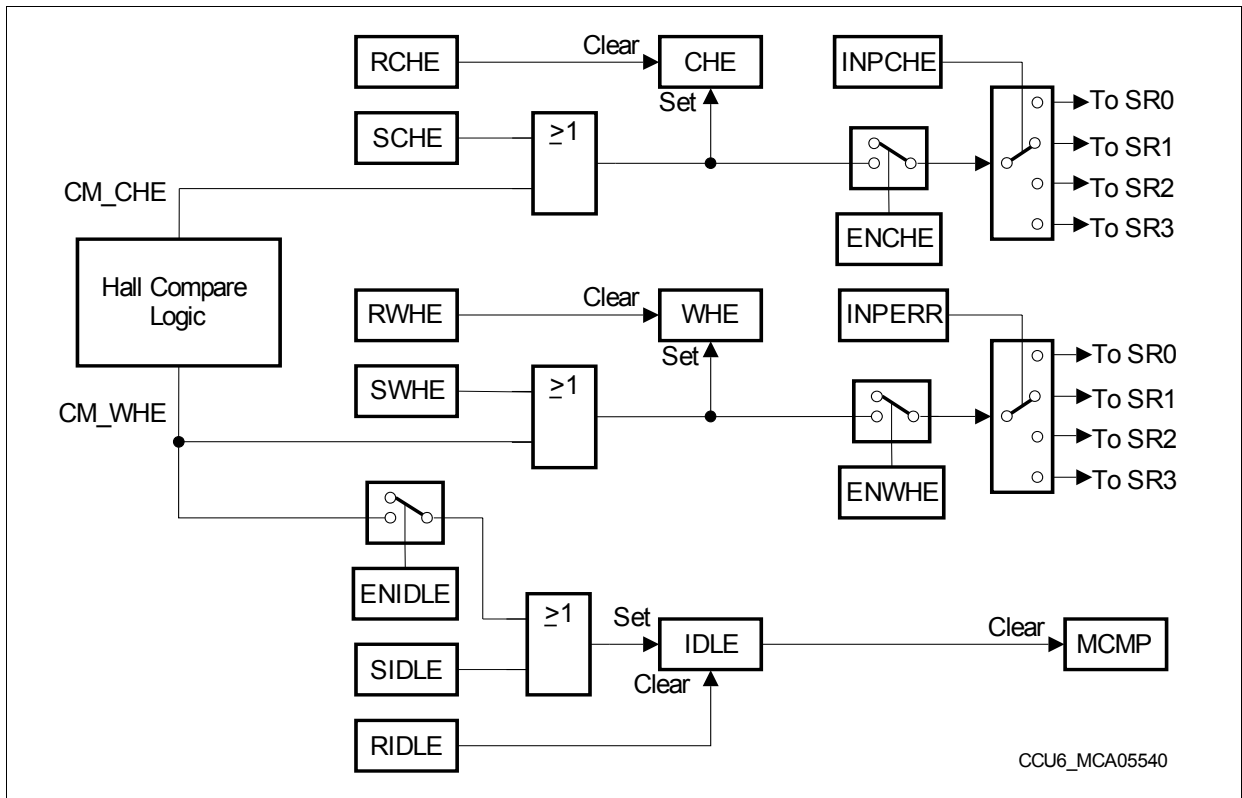
Flag **IS.CHE** (Correct Hall Event) is set by signal CM\_CHE when the sampled Hall pattern matches the expected one (EXPH). This flag can also be set by SW by setting bit **ISS.SCHE** = 1. If enabled by bit **IEN.ENCHE** = 1, the set signal for CHE can also generate an interrupt request to the CPU. Bit field **INP.INPCHE** defines which service request output becomes activated in case of an interrupt request. To clear flag CHE, SW needs to write **ISR.RCHE** = 1.

Flag **IS.WHE** indicates a Wrong Hall Event. Its handling for flag setting and resetting as well as interrupt request generation are similar to the mechanism for flag CHE.

The implementation of flag STR is done in the same way as for CHE and WHE. This flag is set by HW by the shadow transfer signal MCM\_ST (see also [Figure 20-35](#)).

Please note that for flags CHE, WHE, and STR, the interrupt request generation is triggered by the set signal for the flag. That means, a request can be generated even if the flag is already set. There is no need to clear the flag in order to enable further interrupt requests.

The implementation for the IDLE flag is different. It is set by HW through signal CM\_WHE if enabled by bit ENIDLE. Software can also set the flag via bit SIDLE. As long as bit IDLE is set, the modulation pattern field MCMP is cleared to force the outputs to the passive state. Flag IDLE must be cleared by software by writing RIDLE = 1 in order to return to normal operation. To fully restart from IDLE mode, the transfer requests for the bit fields in register MCMOUTS to register MCMOUT have to be initiated by software via bits STRMCM and STRHP in register MCMOUTS. In this way, the release from IDLE mode is under software control, but can be performed synchronously to the PWM signal.

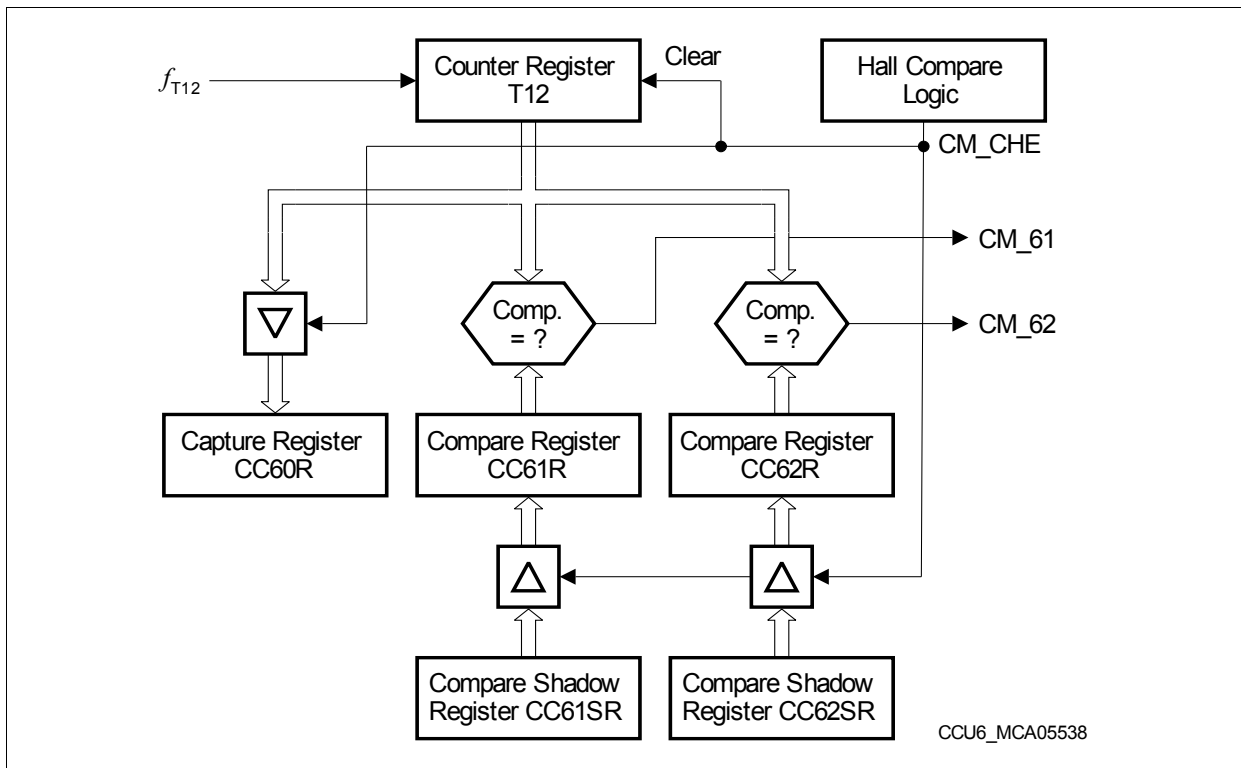


**Figure 20-38 Hall Mode Flags**

#### 20.6.4 Hall Mode for Brushless DC-Motor Control

The CCU6 provides a mode for the Timer T12 Block especially targeted for convenient control of block commutation patterns for Brushless DC-Motors. This mode is selected by setting all **T12MSEL**.MSEL6x bit fields of the three T12 Channels to 1000<sub>B</sub>.

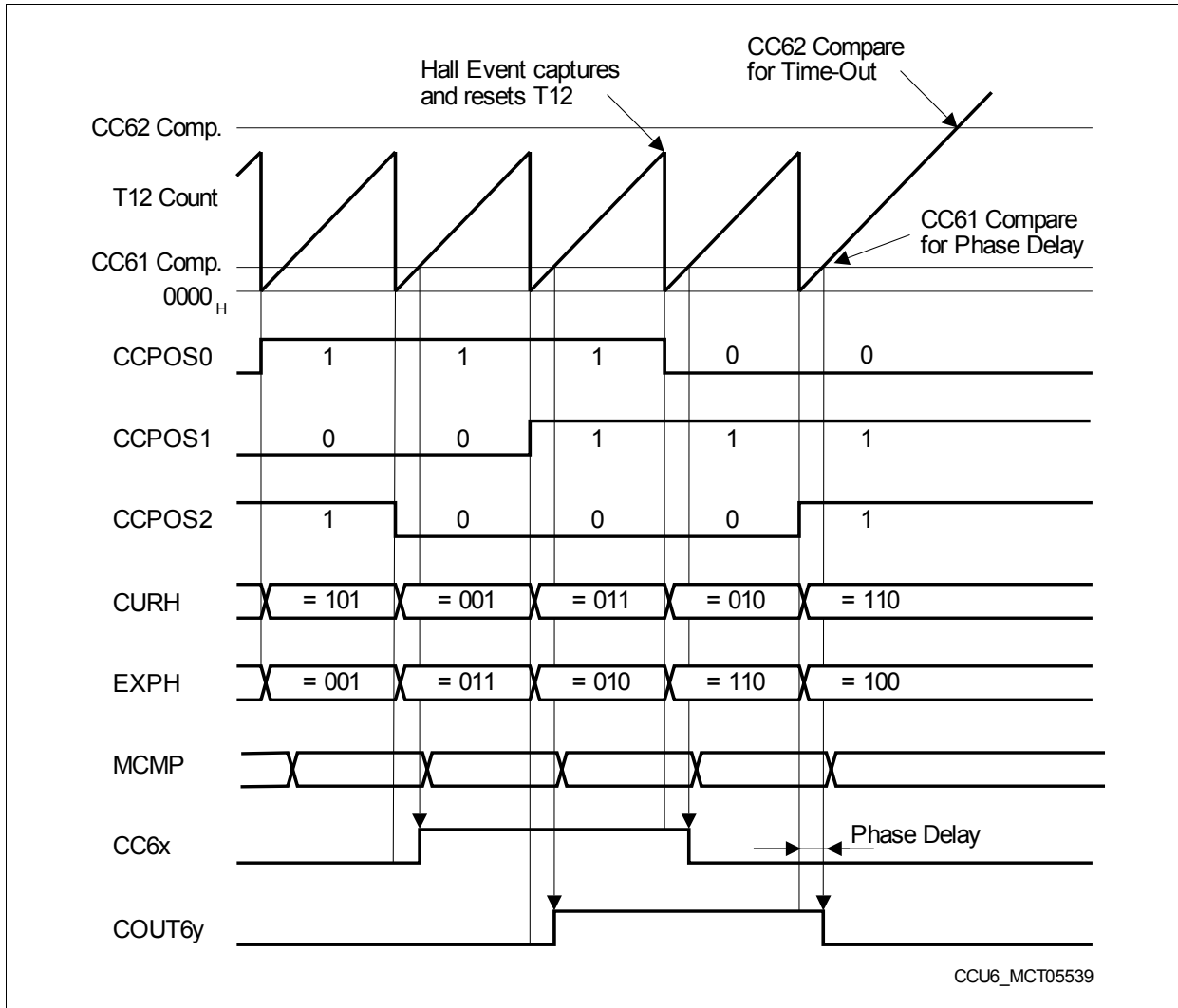
In this mode, illustrated in **Figure 20-39**, channel CC60 is placed in capture mode to measure the time elapsed between the last two correct Hall events, channel CC61 in compare mode to provide a programmable phase delay between the Hall event and the application of a new PWM output pattern, and channel CC62 also in compare mode as first time-out criterion. A second time-out criterion can be built by the T12 period match event.



**Figure 20-39 T12 Block in Hall Sensor Mode**

The signal CM\_CHE from the Hall compare logic is used to transfer the new compare values from the shadow registers CC6xSR into the actual compare registers CC6xR, performs the shadow transfer for the T12 period register, to capture the current T12 contents into register CC60R, and to clear T12.

*Note: In this mode, the shadow transfer signal T12\_ST is not generated. Not all shadow bits, such as the PSLy bits, will be transferred to their main registers. To program the main registers, SW needs to write to these registers while Timer T12 is stopped. In this case, a SW write actualizes both registers.*



**Figure 20-40 Brushless DC-Motor Control Example (all MSEL6x = 1000<sub>B</sub>)**

After the detection of an expected Hall pattern (CM\_CHE active), the T12 count value is captured into channel CC60 (representing the actual rotor speed by measuring the elapsed time between the last two correct Hall events), and T12 is reset. When the timer reaches the compare value in channel CC61, the next multi-channel state is switched by triggering the shadow transfer of bit field MCMP (if enabled in bit field **SWEN**). This trigger event can be combined with the synchronization of the next multi-channel state to the PWM source (to avoid spikes on the output lines, see **Section 20.5**). This compare function of channel CC61 can be used as a phase delay from the position sensor input signals to the switching of the output signals, that is necessary if a sensorless back-EMF technique or Hall sensors are used. The compare value in channel CC62 can be used as a time-out trigger (interrupt), indicating that the actual motor speed is far below the desired destination value. An abnormal load change can be detected with this feature and PWM generation can be disabled.

## 20.7 Modulation Control Registers

### 20.7.1 Modulation Control

This register contains bits enabling the modulation of the corresponding output signal by PWM pattern generated by the timers T12 and T13. Furthermore, the multi-channel mode can be enabled as additional modulation source for the output signals.

#### MODCTR

**Modulation Control Register**

**XSFR(40<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>ECT 130</b>	<b>0</b>							<b>MCM EN</b>	<b>0</b>						
rw	r							rw	r						

Field	Bits	Type	Description
<b>T12MODEN</b>	[5:0]	rw	<b>T12 Modulation Enable</b> These bits enable the modulation of the corresponding output signal by a PWM pattern generated by timer T12. T12MODEN0 = MODCTR.0 for output CC60 T12MODEN1 = MODCTR.1 for output COUT60 T12MODEN2 = MODCTR.2 for output CC61 T12MODEN3 = MODCTR.3 for output COUT61 T12MODEN4 = MODCTR.4 for output CC62 T12MODEN5 = MODCTR.5 for output COUT62 0 <sub>B</sub> The modulation of the corresponding output signal by a T12 PWM pattern is disabled. 1 <sub>B</sub> The modulation of the corresponding output signal by a T12 PWM pattern is enabled.
<b>MCMEN</b>	7	rw	<b>Multi-Channel Mode Enable</b> 0 <sub>B</sub> The modulation of the corresponding output signal by a multi-channel pattern according to bit field MCMOUT is disabled. 1 <sub>B</sub> The modulation of the corresponding output signal by a multi-channel pattern according to bit field MCMOUT is enabled.

Field	Bits	Type	Description
<b>T13MODEN</b>	[13:8]	rw	<b>T13 Modulation Enable</b> These bits enable the modulation of the corresponding output signal by the PWM pattern CC63_O generated by timer T13. T13MODEN0 = MODCTR.8 for output CC60 T13MODEN1 = MODCTR.9 for output COUT60 T13MODEN2 = MODCTR.10 for output CC61 T13MODEN3 = MODCTR.11 for output COUT61 T13MODEN4 = MODCTR.12 for output CC62 T13MODEN5 = MODCTR.13 for output COUT62 0 <sub>B</sub> The modulation of the corresponding output signal by a T13 PWM pattern is disabled. 1 <sub>B</sub> The modulation of the corresponding output signal by a T13 PWM pattern is enabled.
<b>ECT13O</b>	15	rw	<b>Enable Compare Timer T13 Output</b> 0 <sub>B</sub> The output COUT63 is in the passive state. 1 <sub>B</sub> The output COUT63 is enabled for the PWM signal generated by T13.
<b>0</b>	6, 14	r	<b>reserved;</b> returns 0 if read; should be written with 0;

## 20.7.2 Trap Control Register

The register TRPCTR controls the trap functionality. It contains independent enable bits for each output signal and control bits to select the behavior in case of a trap condition. The trap condition is a low level on the CTRAP input pin, that is monitored (inverted level) by bit IS.TRPF. While TRPF=1 (trap input active), the trap state bit IS.TRPS is set to 1.

### TRPCTR

#### Trap Control Register

**XSFR(42<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TRP PEN	TRP EN 13	TRPEN						0					TRP M2	TRP M1	TRP M0
rw	rw	rw						r					rw	rw	rw

Field	Bits	Type	Description
TRPM1, TRPM0	1, 0	rw	<b>Trap Mode Control Bits 1, 0</b> These two bits define the behavior of the selected outputs when leaving the trap state after the trap condition has become inactive again. A synchronization to the timer driving the PWM pattern avoids unintended pulses when leaving the trap state. The combination [TRPM1, TRPM0] leads to: 00 <sub>B</sub> The trap state is left (return to normal operation) after TRPF has become 0 again when a zero-match of T12 (while counting up) is detected (synchronization to T12). 01 <sub>B</sub> The trap state is left (return to normal operation) after TRPF has become 0 again when a zero-match of T13 is detected (synchronization to T13). 10 <sub>B</sub> reserved 11 <sub>B</sub> The trap state is left (return to normal operation) immediately after TRPF has become 0 again without any synchronization to T12 or T13.

Field	Bits	Type	Description
<b>TRPM2</b>	2	rw	<b>Trap Mode Control Bit 2</b> This bit defines how the trap flag TRPF can be cleared after the trap input condition ( $\overline{\text{CTRAP}} = 0$ and $\text{TRPPEN} = 1$ ) is no longer valid (either by $\overline{\text{CTRAP}} = 1$ or by $\text{TRPPEN} = 0$ ). $0_B$ Automatic Mode: Bit TRPF is cleared by HW if the trap input condition is no longer valid. $1_B$ Manual Mode: Bit TRPF stays 0 after the trap input condition is no longer valid. It has to be cleared by SW by writing $\text{ISR.RTRPF} = 1$ .
<b>TRPEN</b>	[13:8]	rw	<b>Trap Enable Control</b> Setting a bit enables the trap functionality for the following corresponding output signals: $\text{TRPEN0} = \text{TRPCTR.8}$ for output CC60 $\text{TRPEN1} = \text{TRPCTR.9}$ for output COUT60 $\text{TRPEN2} = \text{TRPCTR.10}$ for output CC61 $\text{TRPEN3} = \text{TRPCTR.11}$ for output COUT61 $\text{TRPEN4} = \text{TRPCTR.12}$ for output CC62 $\text{TRPEN5} = \text{TRPCTR.13}$ for output COUT62 $0_B$ The trap functionality of the corresponding output signal is disabled. The output state is independent from bit IS.TRPS. $1_B$ The trap functionality of the corresponding output signal is enabled. The output state is set to the passive while $\text{IS.TRPS}=1$ .
<b>TRPEN13</b>	14	rw	<b>Trap Enable Control for Timer T13</b> $0_B$ The trap functionality for output COUT63 is disabled. The output state is independent from bit IS.TRPS. $1_B$ The trap functionality for output COUT63 is enabled. The output state is set to the passive while $\text{IS.TRPS}=1$ .



**Capture/Compare Unit 6 (CCU6)**

Field	Bits	Type	Description
TRPPEN	15	rw	<b>Trap Pin Enable</b> This bit enables the input (pin) function for the trap generation. An interrupt can <u>only be generated</u> if a falling edge is detected at pin CTRAP while TRPPEN = 1. 0 <sub>B</sub> The CCU6 trap functionality based on the input CTRAP is disabled. A CCU6 trap can only be generated by SW by setting bit TRPF. 1 <sub>B</sub> The CCU6 trap functionality based on the input CTRAP is enabled. A CCU6 trap can be generated by SW by setting bit TRPF or by CTRAP=0.
0	[7:3]	r	<b>reserved;</b> returns 0 if read; should be written with 0;

### 20.7.3 Passive State Level Register

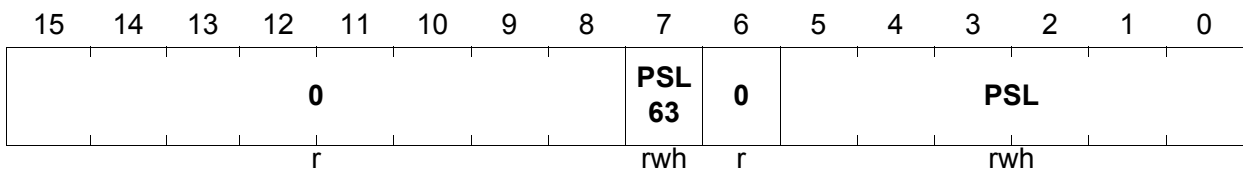
Register PSLR defines the passive state level of the PWM outputs of the module. The passive state level is the value that is driven during the passive state of the output. During the active state, the corresponding output pin drives the active state level, that is the inverted passive state level. The passive state level permits to adapt the driven output levels to the driver polarity (inverted, not inverted) of the connected power stage. The bits in this register have shadow bit fields to permit a concurrent update of all PWM-related parameters (bit field PSL is updated with T12\_ST, whereas PSL63 is updated with T13\_ST). The actually used values can be read (attribute “rh”), whereas the shadow bits can only be written (attribute “w”).

#### PSLR

##### Passive State Level Register

**XSFR(44<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
PSL	[5:0]	rwh	<b>Compare Outputs Passive State Level</b> These bits define the passive level driven by the module outputs during the passive state. PSL0 = PSLR.0 for output CC60 PSL1 = PSLR.1 for output COUT60 PSL2 = PSLR.2 for output CC61 PSL3 = PSLR.3 for output COUT61 PSL4 = PSLR.4 for output CC62 PSL5 = PSLR.5 for output COUT62 0 <sub>B</sub> The passive level is 0. 1 <sub>B</sub> The passive level is 1.
PSL63	7	rwh	<b>Passive State Level of Output COUT63</b> This bit defines the passive level driven by the module output COUT63 during the passive state. 0 <sub>B</sub> The passive level is 0. 1 <sub>B</sub> The passive level is 1.
0	6, [15:8]	r	<b>reserved;</b> returns 0 if read; should be written with 0;

## 20.7.4 Multi-Channel Mode Registers

Register MCMCTR contains control bits for the multi-channel functionality.

### MCMCTR

#### Multi-Channel Mode Control Register

**XSFR(4E<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0					<b>STE 13U</b>	<b>STE 12D</b>	<b>STE 12U</b>	0		<b>SWSYN</b>		0	<b>SWSEL</b>		
r					rw	rw	rw	r		rw		r	rw		

Field	Bits	Type	Description
<b>SWSEL</b>	[2:0]	rw	<b>Switching Selection</b> Bit field SWSEL selects one of the following trigger request sources (next multi-channel event) for the shadow transfer MCM_ST from MCMPS to MCMP. The trigger request is stored in the reminder flag R until the shadow transfer is done and flag R is cleared automatically with the shadow transfer. The shadow transfer takes place synchronously with an event selected in bit field SWSYN. 000 <sub>B</sub> No trigger request will be generated 001 <sub>B</sub> Correct Hall pattern detected (CM_CHE) 010 <sub>B</sub> T13 period-match detected (while counting up) 011 <sub>B</sub> T12 one-match (while counting down) 100 <sub>B</sub> T12 channel 1 compare-match detected (phase delay function) 101 <sub>B</sub> T12 period match detected (while counting up) 110 <sub>B</sub> reserved, no trigger request will be generated 111 <sub>B</sub> reserved, no trigger request will be generated

Field	Bits	Type	Description
<b>SWSYN</b>	[5:4]	rw	<b>Switching Synchronization</b> Bit field SWSYN defines the synchronization mechanism of the shadow transfer event MCM_ST if it has been requested before (flag R set by an event selected by SWSEL) and if MCMEN = 1. This feature permits the synchronization of the outputs to the PWM source, that is used for modulation (T12 or T13). 00 <sub>B</sub> Direct; the trigger event immediately leads to the shadow transfer 01 <sub>B</sub> A T13 zero-match triggers the shadow transfer 10 <sub>B</sub> A T12 zero-match (while counting up) triggers the shadow transfer 11 <sub>B</sub> reserved; no action
<b>STE12U</b>	8	rw	<b>Shadow Transfer Enable for T12 Upcounting</b> This bit enables the shadow transfer T12_ST if flag MCMOUT.R is set or becomes set while a T12 period match is detected while counting up. 0 <sub>B</sub> No action 1 <sub>B</sub> The T12_ST shadow transfer mechanism is enabled if MCMEN = 1.
<b>STE12D</b>	9	rw	<b>Shadow Transfer Enable for T12 Downcounting</b> This bit enables the shadow transfer T12_ST if flag MCMOUT.R is set or becomes set while a T12 one match is detected while counting down. 0 <sub>B</sub> No action 1 <sub>B</sub> The T12_ST shadow transfer mechanism is enabled if MCMEN = 1.
<b>STE13U</b>	10	rw	<b>Shadow Transfer Enable for T13 Upcounting</b> This bit enables the shadow transfer T13_ST if flag MCMOUT.R is set or becomes set while a T13 period match is detected. 0 <sub>B</sub> No action 1 <sub>B</sub> The T13_ST shadow transfer mechanism is enabled if MCMEN = 1.
<b>0</b>	3, [7:6], [15:11]	r	<b>reserved;</b> returns 0 if read; should be written with 0;

**Capture/Compare Unit 6 (CCU6)**

Register MCMOUTS contains bits used as pattern input for the multi-channel mode and the Hall mode. This register is a shadow register (that can be read and written) for register MCMOUT, indicating the currently active signals.

**MCMOUTS**

**Multi-Channel Mode Output Shadow Register**

**XSFR(4A<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>STR HP</b>	<b>0</b>	<b>CURHS</b>			<b>EXPHS</b>			<b>STR MCM</b>	<b>0</b>	<b>MCMPs</b>					
w	r	rw			rw			w	r	rw					

Field	Bits	Type	Description
<b>MCMPs</b>	[5:0]	rw	<b>Multi-Channel PWM Pattern Shadow</b> Bit field MCMPs is the shadow bit field for bit field MCMP. The multi-channel shadow transfer is triggered by MCM_ST according to the transfer conditions defined by register MCMCTR.
<b>STRMCM</b>	7	w	<b>Shadow Transfer Request for MCMPs</b> Writing STRMCM = 1 leads to an immediate activation of MCM_ST to update bit field MCMP by the value of MCMPs. When read, this bit always delivers 0. 0 <sub>B</sub> No action. 1 <sub>B</sub> Bit field MCMP is updated.
<b>EXPHS</b>	[10:8]	rw	<b>Expected Hall Pattern Shadow</b> Bit field EXPHS is the shadow bit field for bit field EXPH. The shadow transfer takes place when a correct Hall event is detected (CM_CHE).
<b>CURHS</b>	[13:11]	rw	<b>Current Hall Pattern Shadow</b> Bit field CURHS is the shadow bit field for bit field CURH. The shadow transfer takes place when a correct Hall event is detected (CM_CHE).

Field	Bits	Type	Description
<b>STRHP</b>	15	w	<b>Shadow Transfer Request for the Hall Pattern</b> Writing STRHP = 1 leads to an immediate activation of HP_ST to update bit fields EXPH and CURH by EXPHS and CURHS. When read, this bit always delivers 0. 0 <sub>B</sub> No action. 1 <sub>B</sub> Bit fields EXPH and CURH are updated.
<b>0</b>	6, 14	r	<b>reserved;</b> returns 0 if read; should be written with 0;

## MCMOUT

### Multi-Channel Mode Output Register

**XSFR(4C<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>0</b>	<b>CURH</b>			<b>EXPH</b>			<b>0</b>	<b>R</b>	<b>MCMP</b>						
r	rh			rh			r	rh	rh						

Field	Bits	Type	Description
<b>MCMP</b>	[5:0]	rh	<b>Multi-Channel PWM Pattern</b> Bit field MCMP defines the output pattern for the multi-channel mode. If this mode is enabled by MODCTR.MCMEN = 1, the output state of all T12 related PWM outputs can be modified. This bit field is 0 while IS.IDLE = 1. MCMP0 = MCMOUT.0 for output CC60 MCMP1 = MCMOUT.1 for output COUT60 MCMP2 = MCMOUT.2 for output CC61 MCMP3 = MCMOUT.3 for output COUT61 MCMP4 = MCMOUT.4 for output CC62 MCMP5 = MCMOUT.5 for output COUT62 0 <sub>B</sub> The output is set to the passive state. A PWM generated by T12 or T13 are not taken into account. 1 <sub>B</sub> The output can be in the active state, depending on the enabled PWM modulation signals generated by T12, T13 and the trap state.

**Capture/Compare Unit 6 (CCU6)**

Field	Bits	Type	Description
<b>R</b>	6	rh	<b>Reminder Flag</b> This flag indicates that the shadow transfer from MCMPS to MCMC has been requested by the selected trigger source. It is cleared when the shadow transfer takes place or while MCMEN=0. 0 <sub>B</sub> A shadow transfer MCM_ST is not requested. 1 <sub>B</sub> A shadow transfer MCM_ST is requested, but has not yet been executed, because the selected synchronization condition has not yet occurred.
<b>EXPH</b>	[10:8]	rh	<b>Expected Hall Pattern</b> Bit field EXPH is updated by a shadow transfer HP_ST from bit field EXPHS. If HCRDY = 1, EXPH is compared to the sampled CCPOSx inputs in order to detect the occurrence of the next desired (=expected) hall pattern or a wrong pattern. If the sampled hall pattern at the hall input pins is equal to bit field EXPH, a correct Hall event has been detected (CM_CHE).
<b>CURH</b>	[13:11]	rh	<b>Current Hall Pattern</b> Bit field CURH is updated by a shadow transfer HP_ST from bit field CURHS. If HCRDY = 1, CURH is compared to the sampled CCPOSx inputs in order to detect a spike. If the sampled Hall pattern at the Hall input pins is equal to bit field CURH, no Hall event has been detected. If the sampled Hall input pattern is neither equal to CURH nor equal to EXPH, the Hall event was not the desired one and may be due to a fatal error (e.g. blocked rotor, etc.). In this case, a wrong Hall event has been detected (CM_WHE).
<b>0</b>	7, [15:14]	r	<b>reserved;</b> returns 0 if read; should be written with 0;

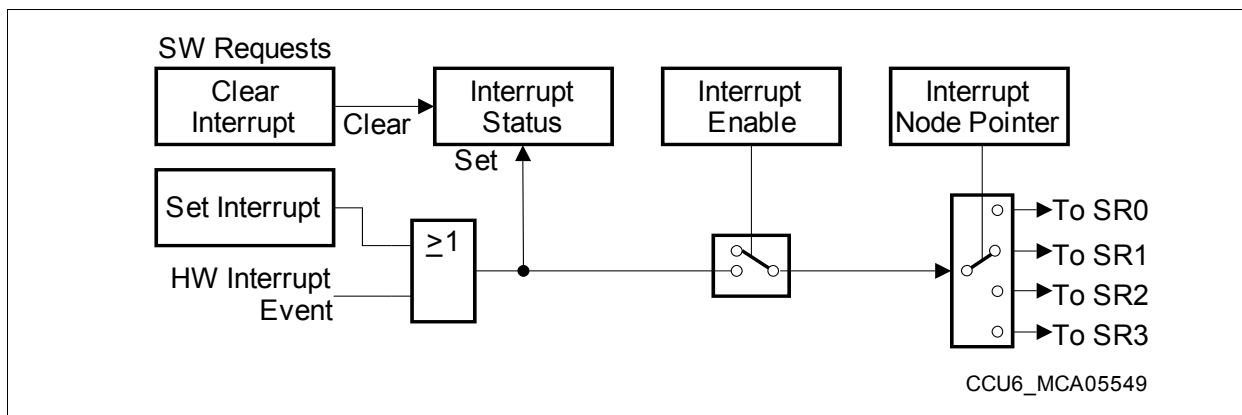
## 20.8 Interrupt Handling

This section describes the interrupt handling of the CCU6 module.

### 20.8.1 Interrupt Structure

The HW interrupt event or the SW setting of the corresponding interrupt set bit (in register ISS) sets the event indication flags (in register IS) and can trigger the interrupt generation. The interrupt pulse is generated independently from the interrupt status flag in register IS (it is not necessary to clear the related status bit to be able to generate another interrupt). The interrupt flag can be cleared by SW by writing to the corresponding bit in register ISR.

If enabled by the related interrupt enable bit in register IEN, an interrupt pulse can be generated on one of the four service request outputs (SR0 to SR3) of the module. If more than one interrupt source is connected to the same interrupt node pointer (in register INP), the requests are logically OR-combined to one common service request output (see [Figure 20-41](#)).

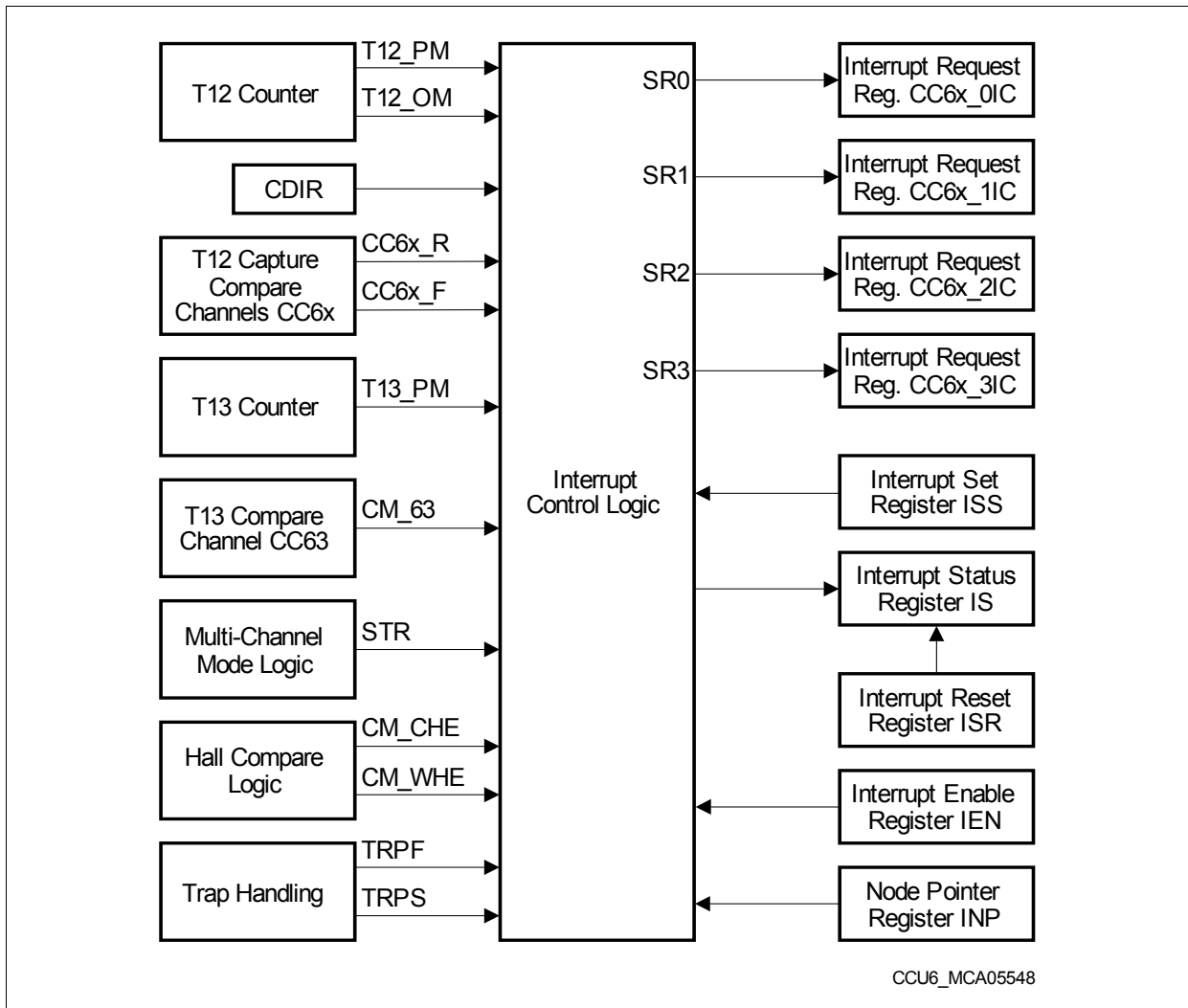


**Figure 20-41 General Interrupt Structure**

The available interrupt events in the CCU6 are shown in [Figure 20-42](#).



**Capture/Compare Unit 6 (CCU6)**



**Figure 20-42 Interrupt Sources and Events**

## 20.8.2 Interrupt Registers

### 20.8.2.1 Interrupt Status Register

Register IS contains the individual interrupt request bits. This register can only be read, write actions have no impact on the contents of this register. The SW can set or clear the bits individually by writing to the registers ISS (to set the bits) or to register ISR (to clear the bits).

The interrupt generation is independent from the value of the bits in register IS, e.g. the interrupt will be generated (if enabled) even if the corresponding bit is already set. The trigger for an interrupt generation is the detection of a set condition (by HW or SW) for the corresponding bit in register IS.

In compare mode (and hall mode), the timer-related interrupts are only generated while the timer is running ( $T1xR=1$ ). In capture mode, the capture interrupts are also generated while the timer T12 is stopped.

*Note: Not all bits in register IS can generate an interrupt. Other status bits have been added, that have a similar structure for their set and clear actions. It is recommended that SW checks the interrupt bits bit-wisely (instead of common OR over the bits).*

#### IS

##### Interrupt Status Register

**XSFR(50<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>STR</b>	<b>IDLE</b>	<b>WHE</b>	<b>CHE</b>	<b>TRP S</b>	<b>TRP F</b>	<b>T13 PM</b>	<b>T13 CM</b>	<b>T12 PM</b>	<b>T12 OM</b>	<b>ICC 62F</b>	<b>ICC 62R</b>	<b>ICC 61F</b>	<b>ICC 61R</b>	<b>ICC 60F</b>	<b>ICC 60R</b>
rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
<b>ICC60R, ICC61R, ICC62R</b>	0, 2, 4	rh	<b>Capture, Compare-Match Rising Edge Flag</b> This bit indicates that event CC6x_R has been detected. This event occurs in compare mode when a compare-match is detected while T12 is counting up ( $CM\_6x$ and $CDIR = 0$ ) and in capture mode when a rising edge is detected at the related input CC6xIN. 0 <sub>B</sub> The event has not yet been detected. 1 <sub>B</sub> The event has been detected.

**Capture/Compare Unit 6 (CCU6)**

Field	Bits	Type	Description
ICC60F, ICC61F, ICC62F	1, 3, 5	rh	<b>Capture, Compare-Match Falling Edge Flag</b> This bit indicates that event CC6x_F has been detected. This event occurs in compare mode when a compare-match is detected while T12 is counting down (CM_6x and CDIR = 1) and in capture mode when a falling edge is detected at the related input CC6xIN. 0 <sub>B</sub> The event has not yet been detected. 1 <sub>B</sub> The event has been detected.
T12OM	6	rh	<b>Timer T12 One-Match Flag</b> This bit indicates that a timer T12 one-match while counting down (T12_OM and CDIR = 1) has been detected. 0 <sub>B</sub> The event has not yet been detected. 1 <sub>B</sub> The event has been detected.
T12PM	7	rh	<b>Timer T12 Period-Match Flag</b> This bit indicates that a timer T12 period-match while counting up (T12_PM and CDIR = 0) has been detected. 0 <sub>B</sub> The event has not yet been detected. 1 <sub>B</sub> The event has been detected.
T13CM	8	rh	<b>Timer T13 Compare-Match Flag</b> This bit indicates that a timer T13 compare-match (CM_63) has been detected. 0 <sub>B</sub> The event has not yet been detected. 1 <sub>B</sub> The event has been detected.
T13PM	9	rh	<b>Timer T13 Period-Match Flag</b> This bit indicates that a timer T13 period-match (T13_PM) has been detected. 0 <sub>B</sub> The event has not yet been detected. 1 <sub>B</sub> The event has been detected.
TRPF	10	rh	<b>Trap Flag</b> This bit indicates if a trap condition (input $\overline{\text{CTRAP}}$ = 0 or by SW) is / has been detected. If TRM2= 0, it becomes cleared automatically if $\overline{\text{CTRAP}}$ = 1 or TRPPEN = 0, whereas if TRM2 = 1, it has to be cleared by writing RTRPF = 1. 0 <sub>B</sub> The trap condition has not been detected. 1 <sub>B</sub> The trap condition is / has been detected.

**Capture/Compare Unit 6 (CCU6)**

Field	Bits	Type	Description
<b>TRPS</b>	11	rh	<b>Trap State<sup>1)</sup></b> This bit indicates the actual trap state. It is set if TRPF = 1 and becomes cleared according to the mode selected in register TRPCTR. 0 <sub>B</sub> The trap state is not active. 1 <sub>B</sub> The trap state is active.
<b>CHE</b>	12	rh	<b>Correct Hall Event</b> This bit indicates that a correct Hall event (CM_CHE) has been detected. 0 <sub>B</sub> The event has not yet been detected. 1 <sub>B</sub> The event has been detected.
<b>WHE</b>	13	rh	<b>Wrong Hall Event</b> This bit indicates that a wrong Hall event (CM_WHE) has been detected. 0 <sub>B</sub> The event has not yet been detected. 1 <sub>B</sub> The event has been detected.
<b>IDLE</b>	14	rh	<b>IDLE State</b> If enabled by ENIDLE = 1, this bit is set together with bit WHE and it has to be cleared by SW. 0 <sub>B</sub> No action. 1 <sub>B</sub> Bit field MCMP is cleared, the selected outputs are set to passive state.
<b>STR</b>	15	rh	<b>Multi-Channel Mode Shadow Transfer Request</b> This bit indicates that a shadow transfer from MCMPS to MCMP (MCM_ST) has taken place. 0 <sub>B</sub> The event has not yet been detected. 1 <sub>B</sub> The event has been detected.

1) During the trap state, the selected outputs are set to the passive state. The logic level driven during the passive state is defined by the corresponding bit in register PSLR. Bits TRPS=1 and TRPF=0 can occur if the trap condition is no longer active but the selected synchronization has not yet taken place.

### 20.8.2.2 Interrupt Status Set Register

Register ISS contains individual interrupt request set bits to generate a CCU6 interrupt request by software. Writing a 1 sets the bit(s) in register IS at the corresponding bit position(s) and can generate an interrupt event (if available and enabled). All bit positions read as 0.

#### ISS

#### Interrupt Status Set Register

**XSFR(52<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>S</b> <b>STR</b>	<b>S</b> <b>IDLE</b>	<b>S</b> <b>WHE</b>	<b>S</b> <b>CHE</b>	<b>S</b> <b>WHC</b>	<b>S</b> <b>TRP</b> <b>F</b>	<b>S</b> <b>T13</b> <b>PM</b>	<b>S</b> <b>T13</b> <b>CM</b>	<b>S</b> <b>T12</b> <b>PM</b>	<b>S</b> <b>T12</b> <b>OM</b>	<b>S</b> <b>CC</b> <b>62F</b>	<b>S</b> <b>CC</b> <b>62R</b>	<b>S</b> <b>CC</b> <b>61F</b>	<b>S</b> <b>CC</b> <b>61R</b>	<b>S</b> <b>CC</b> <b>60F</b>	<b>S</b> <b>CC</b> <b>60R</b>
W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W

Field	Bits	Type	Description
<b>SCC60R,</b> <b>SCC61R,</b> <b>SCC62R</b>	0, 2, 4	w	<b>Set Capture, Compare-Match Rising Edge Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit CC6xR will be set.
<b>SCC60F,</b> <b>SCC61F,</b> <b>SCC62F</b>	1, 3, 5	w	<b>Set Capture, Compare-Match Falling Edge Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit CC6xF will be set.
<b>ST12OM</b>	6	w	<b>Set Timer T12 One-Match Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit T12OM will be set.
<b>ST12PM</b>	7	w	<b>Set Timer T12 Period-Match Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit T12PM will be set.
<b>ST13CM</b>	8	w	<b>Set Timer T13 Compare-Match Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit T13CM will be set.
<b>ST13PM</b>	9	w	<b>Set Timer T13 Period-Match Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit T13PM will be set.
<b>STRPF</b>	10	w	<b>Set Trap Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bits TRPF and TRPS will be set.

**Capture/Compare Unit 6 (CCU6)**

<b>Field</b>	<b>Bits</b>	<b>Type</b>	<b>Description</b>
<b>SWHC</b>	11	w	<b>Software Hall Compare</b> 0 <sub>B</sub> No action 1 <sub>B</sub> The Hall compare action is triggered.
<b>SCHE</b>	12	w	<b>Set Correct Hall Event Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit CHE will be set.
<b>SWHE</b>	13	w	<b>Set Wrong Hall Event Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit WHE will be set.
<b>SIDLE</b>	14	w	<b>Set IDLE Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit IDLE will be set.
<b>SSTR</b>	15	w	<b>Set STR Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit STR will be set.

### 20.8.2.3 Status Reset Register

Register ISR contains bits to individually clear the interrupt event flags by software. Writing a 1 clears the bit(s) in register IS at the corresponding bit position(s). All bit positions read as 0.

#### ISR

**Interrupt Status Reset Register      XSFR(54<sub>H</sub>)      Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R STR	R IDLE	R WHE	R CHE	0	R TRP F	R T13 PM	R T13 CM	R T12 PM	R T12 OM	R CC 62F	R CC 62R	R CC 61F	R CC 61R	R CC 60F	R CC 60R
W	W	W	W	r	W	W	W	W	W	W	W	W	W	W	W

Field	Bits	Type	Description
<b>RCC60R, RCC61R, RCC62R</b>	0, 2, 4	w	<b>Reset Capture, Compare-Match Rising Edge Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit CC6xR will be cleared.
<b>RCC60F, RCC61F, RCC62F</b>	1, 3, 5	w	<b>Reset Capture, Compare-Match Falling Edge Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit CC6xF will be cleared.
<b>RT12OM</b>	6	w	<b>Reset Timer T12 One-Match Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit T12OM will be cleared.
<b>RT12PM</b>	7	w	<b>Reset Timer T12 Period-Match Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit T12PM IS will be cleared.
<b>RT13CM</b>	8	w	<b>Reset Timer T13 Compare-Match Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit T13CM will be cleared.
<b>RT13PM</b>	9	w	<b>Reset Timer T13 Period-Match Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit T13PM will be cleared.
<b>RTRPF</b>	10	w	<b>Reset Trap Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit TRPF will be cleared (not taken into account while input CTRAP=0 and TRPPEN=1.

**Capture/Compare Unit 6 (CCU6)**

<b>Field</b>	<b>Bits</b>	<b>Type</b>	<b>Description</b>
<b>RCHE</b>	12	w	<b>Reset Correct Hall Event Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit CHE will be cleared.
<b>RWHE</b>	13	w	<b>Reset Wrong Hall Event Flag</b> 1 <sub>B</sub> No action 0 <sub>B</sub> Bit WHE will be cleared.
<b>RIDLE</b>	14	w	<b>Reset IDLE Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit IDLE will be cleared.
<b>RSTR</b>	15	w	<b>Reset STR Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Bit STR will be cleared.
<b>0</b>	11	r	<b>reserved;</b> returns 0 if read; should be written with 0;



### 20.8.2.4 Interrupt Enable Register

Register IEN contains the interrupt enable bits and a control bit to enable the automatic idle function in the case of a wrong hall pattern.

#### IEN

#### Interrupt Enable Register

**XSFR(58<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>EN STR</b>	<b>EN IDLE</b>	<b>EN WHE</b>	<b>EN CHE</b>	<b>0</b>	<b>EN TRP F</b>	<b>EN T13 PM</b>	<b>EN T13 CM</b>	<b>EN T12 PM</b>	<b>EN T12 OM</b>	<b>EN CC 62F</b>	<b>EN CC 62R</b>	<b>EN CC 61F</b>	<b>EN CC 61R</b>	<b>EN CC 60F</b>	<b>EN CC 60R</b>
rw	rw	rw	rw	r	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
<b>ENCC60R, ENCC61R, ENCC62R</b>	0, 2, 4	rw	<b>Capture, Compare-Match Rising Edge Interrupt Enable for Channel CC6x</b> 0 <sub>B</sub> No interrupt will be generated if the set condition for bit CC6xR in register IS occurs. 1 <sub>B</sub> An interrupt will be generated if the set condition for bit CC6xR in register IS occurs. The service request output that will be activated is selected by bit field INPCC6x.
<b>ENCC60F, ENCC61F, ENCC62F</b>	1, 3, 5	rw	<b>Capture, Compare-Match Falling Edge Interrupt Enable for Channel CC6x</b> 0 <sub>B</sub> No interrupt will be generated if the set condition for bit CC6xF in register IS occurs. 1 <sub>B</sub> An interrupt will be generated if the set condition for bit CC6xF in register IS occurs. The service request output that will be activated is selected by bit field INPCC6x.
<b>ENT12OM</b>	6	rw	<b>Enable Interrupt for T12 One-Match</b> 0 <sub>B</sub> No interrupt will be generated if the set condition for bit T12OM in register IS occurs. 1 <sub>B</sub> An interrupt will be generated if the set condition for bit T12OM in register IS occurs. The service request output that will be activated is selected by bit field INPT12.

**Capture/Compare Unit 6 (CCU6)**

<b>Field</b>	<b>Bits</b>	<b>Type</b>	<b>Description</b>
<b>ENT12PM</b>	7	rw	<b>Enable Interrupt for T12 Period-Match</b> 0 <sub>B</sub> No interrupt will be generated if the set condition for bit T12PM in register IS occurs. 1 <sub>B</sub> An interrupt will be generated if the set condition for bit T12PM in register IS occurs. The service request output that will be activated is selected by bit field INPT12.
<b>ENT13CM</b>	8	rw	<b>Enable Interrupt for T13 Compare-Match</b> 0 <sub>B</sub> No interrupt will be generated if the set condition for bit T13CM in register IS occurs. 1 <sub>B</sub> An interrupt will be generated if the set condition for bit T13CM in register IS occurs. The service request output that will be activated is selected by bit field INPT13.
<b>ENT13PM</b>	9	rw	<b>Enable Interrupt for T13 Period-Match</b> 0 <sub>B</sub> No interrupt will be generated if the set condition for bit T13PM in register IS occurs. 1 <sub>B</sub> An interrupt will be generated if the set condition for bit T13PM in register IS occurs. The service request output that will be activated is selected by bit field INPT13.
<b>ENTRPF</b>	10	rw	<b>Enable Interrupt for Trap Flag</b> 0 <sub>B</sub> No interrupt will be generated if the set condition for bit TRPF in register IS occurs. 1 <sub>B</sub> An interrupt will be generated if the set condition for bit TRPF in register IS occurs. The service request output that will be activated is selected by bit field INPERR.
<b>ENCHE</b>	12	rw	<b>Enable Interrupt for Correct Hall Event</b> 0 <sub>B</sub> No interrupt will be generated if the set condition for bit CHE in register IS occurs. 1 <sub>B</sub> An interrupt will be generated if the set condition for bit CHE in register IS occurs. The service request output that will be activated is selected by bit field INPCHE.

**Capture/Compare Unit 6 (CCU6)**

<b>Field</b>	<b>Bits</b>	<b>Type</b>	<b>Description</b>
<b>ENWHE</b>	13	rw	<b>Enable Interrupt for Wrong Hall Event</b> 0 <sub>B</sub> No interrupt will be generated if the set condition for bit WHE in register IS occurs. 1 <sub>B</sub> An interrupt will be generated if the set condition for bit WHE in register IS occurs. The service request output that will be activated is selected by bit field INPERR.
<b>ENIDLE</b>	14	rw	<b>Enable Idle</b> This bit enables the automatic entering of the idle state (bit IDLE will be set) after a wrong hall event has been detected (bit WHE is set). During the idle state, the bit field MCMP is automatically cleared. 0 <sub>B</sub> The bit IDLE is not automatically set when a wrong hall event is detected. 1 <sub>B</sub> The bit IDLE is automatically set when a wrong hall event is detected.
<b>ENSTR</b>	15	rw	<b>Enable Multi-Channel Mode Shadow Transfer Interrupt</b> 0 <sub>B</sub> No interrupt will be generated if the set condition for bit STR in register IS occurs. 1 <sub>B</sub> An interrupt will be generated if the set condition for bit STR in register IS occurs. The service request output that will be activated is selected by bit field INPCHE.
<b>0</b>	11	r	<b>reserved;</b> returns 0 if read; should be written with 0;

### 20.8.2.5 Interrupt Node Pointer Register

Register INP contains the interrupt node pointers allowing a flexible interrupt handling. These bit fields define which service request output will be activated if the corresponding interrupt event occurs and the interrupt generation for this event is enabled.

#### INP

**Interrupt Node Pointer Register      XSFR(56<sub>H</sub>)      Reset Value: 3940<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>0</b>		<b>INP T13</b>		<b>INP T12</b>		<b>INP ERR</b>		<b>INP CHE</b>		<b>INP CC62</b>		<b>INP CC61</b>		<b>INP CC60</b>	
r		rw		rw		rw		rw		rw		rw		rw	

Field	Bits	Type	Description
<b>INPCC60, INPCC61, INPCC62</b>	[1:0], [3:2], [5:4]	rw	<b>Interrupt Node Pointer for Channel CC6x Interrupts</b> This bit field defines the service request output activated due to a set condition for bit CC6xR (if enabled by bit ENCC6xR) or for bit CC6xF (if enabled by bit ENCC6xF). 00 <sub>B</sub> Service request output SR0 is selected. 01 <sub>B</sub> Service request output SR1 is selected. 10 <sub>B</sub> Service request output SR2 is selected. 11 <sub>B</sub> Service request output SR3 is selected.
<b>INPCHE</b>	[7:6]	rw	<b>Interrupt Node Pointer for the CHE Interrupt</b> This bit field defines the service request output activated due to a set condition for bit CHE (if enabled by bit ENCHE) or for bit STR (if enabled by bit ENSTR). Coding see INPCC6x.
<b>INPERR</b>	[9:8]	rw	<b>Interrupt Node Pointer for Error Interrupts</b> This bit field defines the service request output activated due to a set condition for bit TRPF (if enabled by bit ENTRPF) or for bit WHE (if enabled by bit ENWHE). Coding see INPCC6x.

**Capture/Compare Unit 6 (CCU6)**

<b>Field</b>	<b>Bits</b>	<b>Type</b>	<b>Description</b>
<b>INPT12</b>	[11:10]	rw	<b>Interrupt Node Pointer for Timer12 Interrupts</b> This bit field defines the service request output activated due to a set condition for bit T12OM (if enabled by bit ENT12OM) or for bit T12PM (if enabled by bit ENT12PM). Coding see INPCC6x.
<b>INPT13</b>	[13:12]	rw	<b>Interrupt Node Pointer for Timer13 Interrupt</b> This bit field defines the service request output activated due to a set condition for bit T13CM (if enabled by bit ENT13CM) or for bit T13PM (if enabled by bit ENT13PM). Coding see INPCC6x.
<b>0</b>	[15:14]	r	<b>reserved;</b> returns 0 if read; should be written with 0;

## **20.9 General Module Operation**

This section provides information about the:

- Configuration of the behavior of the different device operating modes (see mode control description in [Section 20.9.1](#))
- Input selection (see [Section 20.9.2](#))
- General register description (see [Section 20.9.3](#))

### **20.9.1 Mode Control**

The mode control concept for system control tasks, such as power saving, or suspend request for debugging, allows to program the module behavior under different device operating conditions. The behavior of a CCU6 kernel can be programmed for each of the device operating modes, that are requested by the global state control part of the SCU. Therefore, a CCU6 module provides a kernel state configuration register **KSCFG** defining the behavior in the following device operating modes:

- **Normal operation:**  
This operating mode is the default operating mode when neither a suspend request nor a clock-off request are pending. The module clock is not switched off and the CCU6 registers can be read or written. The kernel behavior is defined by KSCFG.NOMCFG.
- **Suspend mode:**  
This operating mode is requested when a suspend request (issued by a debugger) is pending in the device. The module clock is not switched off and the CCU6 registers can be read or written. The kernel behavior is defined by KSCFG.SUMCFG.
- **Clock-off mode:**  
This operating mode is requested for power saving purposes. The module clock is switched off automatically when all kernels of the CCU6 module reached their specified state in a stop mode. In this case, CCU6 registers can not be accessed. The kernel behavior is defined by KSCFG.COMCFG.

The kernel distinguishes four different blocks (T12, T13, Hall logic, and trap logic). These blocks can be individually enabled for the request of stop mode 0 and stop mode 1 by the sensitivity bits **KSCSR.SBx**. If the request sensitivity is disabled, the block continues normal operation. If the request sensitivity is enabled, the block operates as specified for the selected stop mode.

The complete CCU6 acknowledge is given to the GSC when all four blocks have reached their defined end condition.

**Table 20-12 CCU6 Functional Blocks**

Block	Function	Sensitivity Bit
0	<b>Timer T12:</b> A functional enable is delivered until the specified stop condition is reached. Then, T12 stops counting and the CC6xIN input stages are frozen.	KSCSR.SB0
1	<b>Timer T13:</b> A functional enable is delivered until the specified stop condition is reached. Then, T13 stops counting.	KSCSR.SB1
2	<b>Hall Logic:</b> The hall logic is stopped immediately and the CCPOSx input stages are frozen.	KSCSR.SB2
3	<b>Trap Logic:</b> The trap logic is stopped immediately and the CTRAP input stage is frozen.	KSCSR.SB3

The behavior of the CCU6 kernel can be programmed for each of the device operating modes (normal operation, suspend mode, clock-off mode). Therefore, it supports four kernel modes, as shown in [Table 20-13](#).

**Table 20-13 CCU6 Kernel Behavior**

Kernel Mode	Kernel Behavior	Code
run mode 0	kernel operation as specified, no impact on CCU6 operation (same behavior for run mode 0 and run mode 1)	00 <sub>B</sub>
run mode 1		01 <sub>B</sub>

**Table 20-13 CCU6 Kernel Behavior (cont'd)**

<b>Kernel Mode</b>	<b>Kernel Behavior</b>	<b>Code</b>
stop mode 0	<p>The sensitivity bits are taken into account for:</p> <p><b>T12 block:</b> Timer T12 continues normal operation (if running) until they reach the end of the PWM period and then it stops (same stop condition as in single shot mode). When the timer stops, the CC6xIN inputs are frozen.</p> <p><b>T13 block:</b> Timer T13 continues normal operation (if running) until they reach the end of the PWM period and then it stops (same stop condition as in single shot mode).</p> <p><b>Hall logic block:</b> The CCPOSx input values are frozen.</p> <p><b>Trap logic block:</b> The CTRAP input value is frozen.</p>	10 <sub>B</sub>
stop mode 1	<p>The output lines enabled for the trap condition are set to their passive values (similar to a trap state). The sensitivity bits are taken into account for:</p> <p><b>T12 block:</b> Timer T12 stops immediately and CC6xIN inputs are frozen.</p> <p><b>T13 block:</b> Timer T13 stops.</p> <p><b>Hall logic block:</b> The CCPOSx input values are frozen.</p> <p><b>Trap logic block:</b> The CTRAP input value is frozen.</p>	11 <sub>B</sub>

Generally, bit field KSCFG.NOMCFG should be configured for run mode 0 as default setting for standard operation. If a CCU6 kernel should not react to a suspend request (and to continue operation as in normal mode), bit field KSCFG.SUMCFG has to be configured with the same value as KSCFG.NOMCFG. If a CCU6 kernel 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 has to be written to KSCFG.SUMCFG.

A similar mechanism applies for the clock-off mode with the possibility to program the desired behavior by bit field KSCFG.COMCFG.

*Note: The stop mode selection strongly depends on the application needs and it is very unlikely that different stop modes are required in parallel in the same application. As a result, only one stop mode type (either 0 or 1) should be used in the bit fields in register KSCFG. Do not mix stop mode 0 and stop mode 1 and avoid transitions from stop mode 0 to stop mode 1 (or vice versa) for the CCU6 module.*

If the module clock is disabled by KSCFG.MODEN = 0 or in clock-off mode when the stop condition is reached (in stop mode 0 or 1), the module can not be accessed by read or write operations (except register KSCFG that can always be accessed). As a consequence, it can not be configured.

Please note that bit KSCFG.MODEN should only be set by SW while all configuration fields are configured for run mode 0.



### **20.9.2 Input Selection**

Each CCU6 input signal can be selected from a vector of four or eight possible inputs by programming the port input select registers **PISELL** and **PISELH**. This permits to adapt the pin functionality of the device to the application requirements.

The output pins for the module output signals are chosen in the ports.

Naming convention:

The input vector CC60IN[D:A] for input signal CC60IN is composed of the signals CC60INA to CC60IND.

*Note: All functional inputs of the CCU6 are synchronized to  $f_{CC6}$  before they affect the module internal logic. The resulting delay of  $2/f_{CC6}$  and for asynchronous signals an additional uncertainty of  $1/f_{CC6}$  have to be taken into account for precise timing calculation. An edge of an input signal can only be correctly detected if the high phase and the low phase of the input signal are both longer than  $1/f_{CC6}$ .*

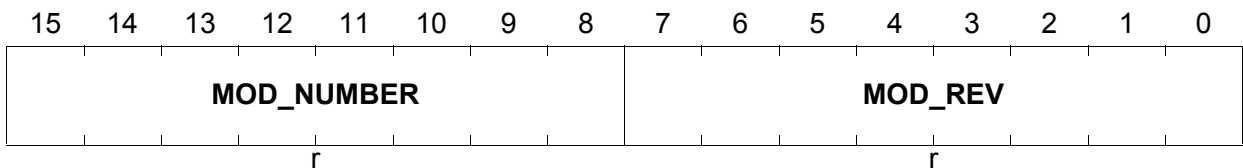
## 20.9.3 General Registers

### 20.9.3.1 ID Register

The ID register is a read-only register used for CCU6 module identification purposes. It provides 8 bits for module identification and 8 bits for revision numbering.

#### ID

**Module Identification Register**      **XSFR(08<sub>H</sub>)**      **Reset Value: 54XX<sub>H</sub>**



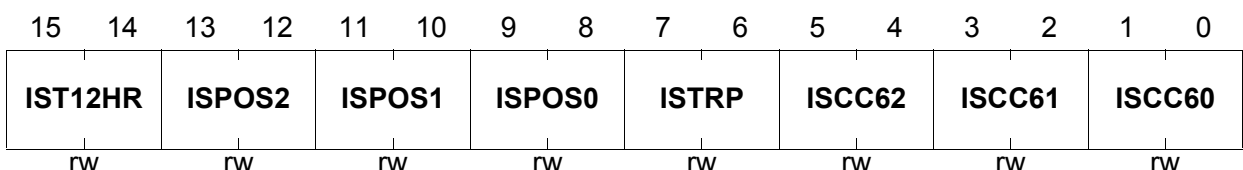
Field	Bits	Type	Description
MOD_REV	[7:0]	r	<b>Module Revision Number Value</b> Bits 7-0 bits are used for module revision numbering. The value of the module revision number starts with 01 <sub>H</sub> (first revision), 02 <sub>H</sub> , 03 <sub>H</sub> , ... up to FF <sub>H</sub> .
MOD_NUMBER	[15:8]	r	<b>Module Identification Number Value</b> Bits 15-8 are used for module identification. The CCU6 has the module number 54 <sub>H</sub> .

### 20.9.3.2 Port Input Select Registers

Registers PISELL and PISELH contain bit fields selecting the actual input signal for the module inputs.

#### PISELL

**Port Input Select Register Low**      **XSFR(04<sub>H</sub>)**      **Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
<b>ISCC60</b>	[1:0]	rw	<b>Input Select for CC60</b> This bit field defines the input signal used as CC60 capture input. 00 <sub>B</sub> The signal CC60INA is selected. 01 <sub>B</sub> The signal CC60INB is selected. 10 <sub>B</sub> The signal CC60INC is selected. 11 <sub>B</sub> The signal CC60IND is selected.
<b>ISCC61</b>	[3:2]	rw	<b>Input Select for CC61</b> This bit field defines the input signal used as CC61 capture input. 00 <sub>B</sub> The signal CC61INA is selected. 01 <sub>B</sub> The signal CC61INB is selected. 10 <sub>B</sub> The signal CC61INC is selected. 11 <sub>B</sub> The signal CC61IND is selected.
<b>ISCC62</b>	[5:4]	rw	<b>Input Select for CC62</b> This bit field defines the input signal used as CC62 capture input. 00 <sub>B</sub> The signal CC62INA is selected. 01 <sub>B</sub> The signal CC62INB is selected. 10 <sub>B</sub> The signal CC62INC is selected. 11 <sub>B</sub> The signal CC62IND is selected.
<b>ISTRP</b>	[7:6]	rw	<b>Input Select for CTRAP</b> This bit field defines the input signal used as CTRAP input. 00 <sub>B</sub> The signal CTRAPA is selected. 01 <sub>B</sub> The signal CTRAPB is selected. 10 <sub>B</sub> The signal CTRAPC is selected. 11 <sub>B</sub> The signal CTRAPD is selected.
<b>ISPOS0</b>	[9:8]	rw	<b>Input Select for CCPOS0</b> This bit field defines the input signal used as CCPOS0 input. 00 <sub>B</sub> The signal CCPOS0A is selected. 01 <sub>B</sub> The signal CCPOS0B is selected. 10 <sub>B</sub> The signal CCPOS0C is selected. 11 <sub>B</sub> The signal CCPOS0D is selected.

**Capture/Compare Unit 6 (CCU6)**

Field	Bits	Type	Description
<b>ISPOS1</b>	[11:10]	rw	<b>Input Select for CCPOS1</b> This bit field defines the input signal used as CCPOS1 input. 00 <sub>B</sub> The signal CCPOS1A is selected. 01 <sub>B</sub> The signal CCPOS1B is selected. 10 <sub>B</sub> The signal CCPOS1C is selected. 11 <sub>B</sub> The signal CCPOS1D is selected.
<b>ISPOS2</b>	[13:12]	rw	<b>Input Select for CCPOS2</b> This bit field defines the input signal used as CCPOS2 input. 00 <sub>B</sub> The signal CCPOS2A is selected. 01 <sub>B</sub> The signal CCPOS2B is selected. 10 <sub>B</sub> The signal CCPOS2C is selected. 11 <sub>B</sub> The signal CCPOS2D is selected.
<b>IST12HR</b>	[15:14]	rw	<b>Input Select for T12HR</b> This bit field defines the input signal used as T12HR input. 00 <sub>B</sub> Either signal T12HRA (if T12EXT = 0) or T12HRE (if T12EXT = 1) is selected. 01 <sub>B</sub> Either signal T12HRB (if T12EXT = 0) or T12HRF (if T12EXT = 1) is selected. 10 <sub>B</sub> Either signal T12HRC (if T12EXT = 0) or T12HRG (if T12EXT = 1) is selected. 11 <sub>B</sub> Either signal T12HRD (if T12EXT = 0) or T12HRH (if T12EXT = 1) is selected.

**PISELH**

**Port Input Select Register High**

**XSFR(06<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0								T13 EXT	T12 EXT	ISCNT13		ISCNT12		IST13HR	
r								rw	rw	rw		rw		rw	

Field	Bits	Type	Description
<b>IST13HR</b>	[1:0]	rw	<b>Input Select for T13HR</b> This bit field defines the input signal used as T13HR input. 00 <sub>B</sub> Either signal T13HRA (if T13EXT = 0) or T13HRE (if T13EXT = 1) is selected. 01 <sub>B</sub> Either signal T13HRB (if T13EXT = 0) or T13HRF (if T13EXT = 1) is selected. 10 <sub>B</sub> Either signal T13HRC (if T13EXT = 0) or T13HRG (if T13EXT = 1) is selected. 11 <sub>B</sub> Either signal T13HRD (if T13EXT = 0) or T13HRH (if T13EXT = 1) is selected.
<b>ISCNT12</b>	[3:2]	rw	<b>Input Select for T12 Counting Input</b> This bit field defines the input event leading to a counting action of T12. 00 <sub>B</sub> The T12 prescaler generates the counting events. Bit TCTR4.T12CNT is not taken into account. 01 <sub>B</sub> Bit TCTR4.T12CNT written with 1 is a counting event. The T12 prescaler is not taken into account. 10 <sub>B</sub> The timer T12 is counting each rising edge detected in the selected T12HR signal. 11 <sub>B</sub> The timer T12 is counting each falling edge detected in the selected T12HR signal.
<b>ISCNT13</b>	[5:4]	rw	<b>Input Select for T13 Counting Input</b> This bit field defines the input event leading to a counting action of T13. 00 <sub>B</sub> The T13 prescaler generates the counting events. Bit TCTR4.T13CNT is not taken into account. 01 <sub>B</sub> Bit TCTR4.T13CNT written with 1 is a counting event. The T13 prescaler is not taken into account. 10 <sub>B</sub> The timer T13 is counting each rising edge detected in the selected T13HR signal. 11 <sub>B</sub> The timer T13 is counting each falling edge detected in the selected T13HR signal.

**Capture/Compare Unit 6 (CCU6)**

Field	Bits	Type	Description
<b>T12EXT</b>	6	rw	<b>Extension for T12HR Inputs</b> This bit extends the 2-bit field IST12HR. 0 <sub>B</sub> One of the signals T12HR[D:A] is selected. 1 <sub>B</sub> One of the signals T12HR[H:E] is selected.
<b>T13EXT</b>	7	rw	<b>Extension for T13HR Inputs</b> This bit extends the 2-bit field IST13HR. 0 <sub>B</sub> One of the signals T13HR[D:A] is selected. 1 <sub>B</sub> One of the signals T13HR[H:E] is selected.
<b>0</b>	[15:8]	r	<b>reserved;</b> returns 0 if read; should be written with 0;

### 20.9.3.3 Kernel State Configuration Register

The kernel state configuration register KSCFG allows the selection of the desired kernel modes for the different device operating modes.

Bit fields KSCFG.NOMCFG and KSCFG.COMCFG are reset by an application reset. Bit field KSCFG.SUMCFG is reset by a debug reset.

*Note: The coding of the bit fields NOMCFG, SUMCFG and COMCFG is described in [Table 20-13](#).*

#### KSCFG

##### Kernel State Configuration Register

XSFR(00 <sub>H</sub> )											Reset Value: 0000 <sub>H</sub>				
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BP COM	0	COMCFG		BP SUM	0	SUMCFG		BP NOM	0	NOMCFG		0		BP MOD EN	MOD EN
w	r	rw		w	r	rw		w	r	rw		r		w	rw

Field	Bits	Type	Description
<b>MODEN</b>	0	rw	<b>Module Enable</b> This bit enables the module kernel clock and the module functionality. 0 <sub>B</sub> The module is switched off immediately (without respecting a stop condition). It does not react on mode control actions and the module clock is switched off. The module does not react on read accesses and ignores write accesses (except to KSCFG). 1 <sub>B</sub> 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 CCU6 registers.
<b>BPMODEN</b>	1	w	<b>Bit Protection for MODEN</b> This bit enables the write access to the bit MODEN. It always reads 0. 0 <sub>B</sub> MODEN is not changed. 1 <sub>B</sub> MODEN is updated with the written value.

Field	Bits	Type	Description
<b>NOMCFG</b>	[5:4]	rw	<b>Normal Operation Mode Configuration</b> This bit field defines the kernel mode applied in normal operation mode. 00 <sub>B</sub> Run mode 0 is selected. 01 <sub>B</sub> Run mode 1 is selected. 10 <sub>B</sub> Stop mode 0 is selected. 11 <sub>B</sub> Stop mode 1 is selected.
<b>BPNO</b>	7	w	<b>Bit Protection for NOMCFG</b> This bit enables the write access to the bit field NOMCFG. It always reads 0. 0 <sub>B</sub> NOMCFG is not changed. 1 <sub>B</sub> NOMCFG is updated with the written value.
<b>SUMCFG</b>	[9:8]	rw	<b>Suspend Mode Configuration</b> This bit field defines the kernel mode applied in suspend mode. Coding like NOMCFG.
<b>BPSUM</b>	11	w	<b>Bit Protection for SUMCFG</b> This bit enables the write access to the bit field SUMCFG. It always reads 0. 0 <sub>B</sub> SUMCFG is not changed. 1 <sub>B</sub> SUMCFG is updated with the written value.
<b>COMCFG</b>	[13:12]	rw	<b>Clock Off Mode Configuration</b> This bit field defines the kernel mode applied in clock-off mode. Coding like NOMCFG.
<b>BPCOM</b>	15	w	<b>Bit Protection for COMCFG</b> This bit enables the write access to the bit field COMCFG. It always reads 0. 0 <sub>B</sub> COMCFG is not changed. 1 <sub>B</sub> COMCFG is updated with the written value.
<b>0</b>	[3:2], 6, 10, 14	r	<b>Reserved</b> returns 0 if read; should be written with 0;

*Note: The bit protection bits BPxxx allow partly modification of the configuration bits with a single write operation (without the need of a read-modify-write mechanism handled by the CPU).*



### 20.9.3.4 Kernel State Sensitivity Control Register

The kernel state control sensitivity register bits define which internal block is effected by stop modes 0 and 1.

#### KSCSR

#### Kernel State Control Sensitivity Register

**XSFR(0E<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0												<b>SB3</b>	<b>SB2</b>	<b>SB1</b>	<b>SB0</b>
r												rw	rw	rw	rw

Field	Bits	Type	Description
<b>SB0, SB1, SB2, SB3</b>	0, 1, 2, 3	rw	<b>Sensitivity Block x</b> This bit defines if block x of the CCU6 kernel is sensitive to stop mode 0 or stop mode 1. The functional definition of the blocks is given in <a href="#">Table 20-12</a> . 0 <sub>B</sub> Block x is not sensitive to stop mode 0 or stop mode 1 and behaves like in run mode 0. It continues normal operation without respecting the defined stop condition. 1 <sub>B</sub> Block x is sensitive to stop mode 0 or stop mode 1. It is respecting the defined stop condition.
<b>0</b>	[15:4]	r	<b>reserved;</b> returns 0 if read; should be written with 0;

## **20.10 Implementation**

This section describes the implementation of the CCU6 modules in the XE16xyM device.

- Address map (see [Section 20.10.1](#))
- Interrupt control registers (see [Section 20.10.2](#))
- Synchronous start (see [Section 20.10.3](#))
- Connections of CCU60 (see [Section 20.10.4.1](#))
- Connections of CCU61 (see [Section 20.10.4.2](#))
- Connections of CCU62 (see [Section 20.10.4.3](#))
- Connections of CCU63 (see [Section 20.10.4.4](#))

### **20.10.1 Address Map**

The four CCU6 modules in the XE16xyM, named CCU60 to CCU63, can be accessed in the following address ranges.

The exact register address is given by the offset of the register (given in [Table 20-1](#)) plus the kernel base address (given in [Table 20-14](#)) of the module.

**Table 20-14 Registers Address Space**

Module	Base Address	End Address	Note
CCU60	EA00 <sub>H</sub>	EA7E <sub>H</sub>	
CCU61	EA80 <sub>H</sub>	EAFE <sub>H</sub>	
CCU62	EB00 <sub>H</sub>	EB7E <sub>H</sub>	
CCU63	EB80 <sub>H</sub>	EBFE <sub>H</sub>	

**Table 20-15 Registers Overview**

Register Short Name	Register Long Name	Offset Address	Page Number
please refer to register table in <a href="#">Section 20.1.3</a>		H	

## **20.10.2 Interrupt Control Registers**

The interrupt control registers are located in the SFR area. They are described in the general interrupt chapter.

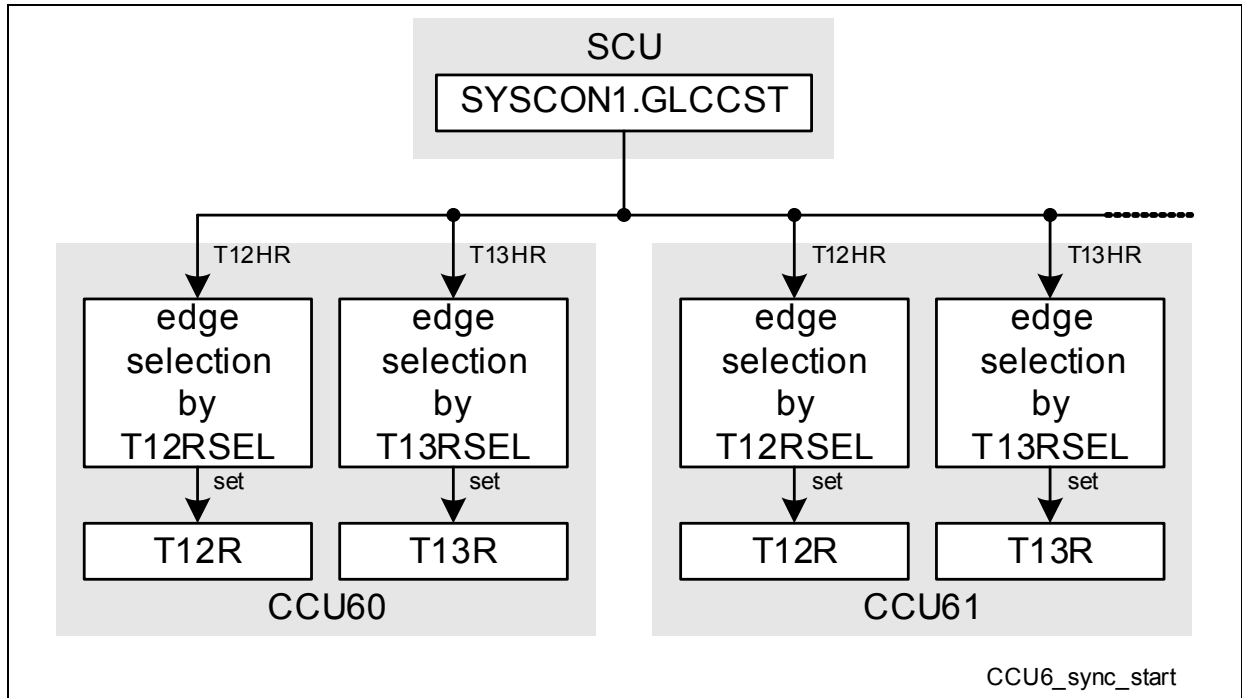
**Table 20-16 CCU6 Interrupt Control Registers**

<b>Short Name</b>	<b>Description</b>
<b>CCU60_0IC</b>	Interrupt Control Register for SR0 of CCU60
<b>CCU60_1IC</b>	Interrupt Control Register for SR1 of CCU60
<b>CCU60_2IC</b>	Interrupt Control Register for SR2 of CCU60
<b>CCU60_3IC</b>	Interrupt Control Register for SR3 of CCU60
<b>CCU61_0IC</b>	Interrupt Control Register for SR0 of CCU61
<b>CCU61_1IC</b>	Interrupt Control Register for SR1 of CCU61
<b>CCU61_2IC</b>	Interrupt Control Register for SR2 of CCU61
<b>CCU61_3IC</b>	Interrupt Control Register for SR3 of CCU61
<b>CCU62_0IC</b>	Interrupt Control Register for SR0 of CCU62
<b>CCU62_1IC</b>	Interrupt Control Register for SR1 of CCU62
<b>CCU62_2IC</b>	Interrupt Control Register for SR2 of CCU62
<b>CCU62_3IC</b>	Interrupt Control Register for SR3 of CCU62
<b>CCU63_0IC</b>	Interrupt Control Register for SR0 of CCU63
<b>CCU63_1IC</b>	Interrupt Control Register for SR1 of CCU63
<b>CCU63_2IC</b>	Interrupt Control Register for SR2 of CCU63
<b>CCU63_3IC</b>	Interrupt Control Register for SR3 of CCU63

### 20.10.3 Synchronous Start Feature

Synchronous start is supported by bit SYSCON1.GLCCST (global capture/compare start) in the SCU module that is connected to the T12HR and T13HR inputs of all CCU6x modules.

The same signal can also be connected to other capture/compare units in order to allow a synchronous start of the capture/compare timers.



**Figure 20-43 Synchronization Concept**

## 20.10.4 Digital Connections

The following tables show the digital connections of the CCU6x modules with other modules or pins in the XE16xyM device.

Each input signal can be selected among 4 or 8 possible input lines, e.g. the input vector for input signal CC60IN is composed of CC60IN[D:A], whereas the input vectors for T12HR and T13HR are composed of T12HR[H:A] and T13HR[H:A].

The following sections refer to the interface signals, whereas the connections of the service request outputs SR[3:0] to the interrupt control registers of each CCU6x to the interrupt control registers is given in [Section 20.10.2](#).

The CCU6x modules are clocked with the XE16xyM system clock, so  $f_{CC6} = f_{SYS}$ .

*Note: All functional inputs of the CCU6 are synchronized to  $f_{CC6}$  before they can affect the module internal logic. The resulting delay of  $2/f_{CC6}$  and an uncertainty of  $1/f_{CC6}$  have to be taken into account for precise timing calculation.*

*An edge of an input signal can only be correctly detected if both, the high phase and the low phase of the input signal are each longer than  $1/f_{CC6}$ .*

### 20.10.4.1 Connections of CCU60

This table describes the module interconnections of CCU60.

**Table 20-17 CCU60 Digital Connections in XE16xyM**

Signal	from/to Module	I/O to CCU60	Can be used to/as
CC60INA	P10.0	I	input signals for capture event on channel CC60
CC60INB	P8.0	I	
CC60INC	0	I	
CC60IND	RTC interrupt	I	
CC61INA	P10.1	I	input signals for capture event on channel CC61
CC61INB	P8.1	I	
CC61INC	0	I	
CC61IND	WUT trigger (SCU)	I	
CC62INA	P10.2	I	input signals for capture event on channel CC62
CC62INB	P8.2	I	
CC62INC	0	I	
CC62IND	0	I	

**Table 20-17 CCU60 Digital Connections in XE16xyM (cont'd)**

<b>Signal</b>	<b>from/to Module</b>	<b>I/O to CCU60</b>	<b>Can be used to/as</b>
CTRAPA	P10.6	I	input signals for CTRAP, the ESRx input refers to the synchronized input signal, which can be filtered (if enabled)
CTRAPB	P8.6	I	
CTRAPC	ESR2	I	
CTRAPD	ERU_PDOUT2	I	
CCPOS0A	P10.7	I	input signals for CCPOS0
CCPOS0B	P9.7	I	
CCPOS0C	CCU61_SR3	I	
CCPOS0D	0	I	
CCPOS1A	P10.8	I	input signals for CCPOS1
CCPOS1B	P9.6	I	
CCPOS1C	CCU63_SR3	I	
CCPOS1D	0	I	
CCPOS2A	P10.9	I	input signals for CCPOS2
CCPOS2B	P9.5	I	
CCPOS2C	ADC0_SR3	I	
CCPOS2D	0	I	
T12HRA	CCU63_MCM_ST	I	input signals for T12HR
T12HRB	P5.5	I	
T12HRC	P5.8	I	
T12HRD	SYSCON.GLCCST	I	
T12HRE	ADC0_ARBCNT	I	
T12HRF	0	I	
T12HRG	0	I	
T12HRH	0	I	

**Table 20-17 CCU60 Digital Connections in XE16xyM (cont'd)**

<b>Signal</b>	<b>from/to Module</b>	<b>I/O to CCU60</b>	<b>Can be used to/as</b>
T13HRA	EXTCLK (SCU)	I	input signals for T13HR
T13HRB	CCU60_T12_ZM	I	
T13HRC	P5.8	I	
T13HRD	SYSCON.GLCCST	I	
T13HRE	ADC0_ARBCNT	I	
T13HRF	0	I	
T13HRG	U0C0_SR3	I	
T13HRH	0	I	
CC60	P10.0 P8.0	O	compare outputs of channel CC60
CC60	ADC0_REQGT0H	O	
COUT60	P10.3 P8.3	O	
CC61	P10.1 P8.1	O	compare outputs of channel CC61
CC61	ADC0_REQGT1H	O	
COUT61	P10.4 P8.4	O	
CC62	P10.2 P8.2	O	compare outputs of channel CC62
CC62	ADC0_REQGT2H	O	
COUT62	P10.5 P8.5	O	
COUT63	P10.7 P10.10 P8.6 U0C0_DX2F U0C1_DX2F	O	compare output of channel CC63
COUT63	ADCx_REQGTyA	O	ADC triggers
T12_ZM	CCU60_T13HRB	O	T12 zero match
T13_PM	ERU_OGU02	O	T13 period match

**Table 20-17 CCU60 Digital Connections in XE16xyM (cont'd)**

<b>Signal</b>	<b>from/to Module</b>	<b>I/O to CCU60</b>	<b>Can be used to/as</b>
MCM_ST	CCU61_T12HRA ERU_OGU01	O	MCM shadow transfer
SR3	CCU61_CCPOS0C	O	CCU61 trigger
SR3	CCU63_CCPOS1C	O	CCU63 trigger



### 20.10.4.2 Connections of CCU61

This table describes the module interconnections of CCU61.

**Table 20-18 CCU61 Digital Connections in XE16xyM**

Signal	from/to Module	I/O to CC61	Can be used to/as
CC60INA	P0.0	I	input signals for capture event on channel CC60
CC60INB	P11.5	I	
CC60INC	0	I	
CC60IND	0	I	
CC61INA	P0.1	I	input signals for capture event on channel CC61
CC61INB	P11.2	I	
CC61INC	0	I	
CC61IND	0	I	
CC62INA	P0.2	I	input signals for capture event on channel CC62
CC62INB	P11.4	I	
CC62INC	0	I	
CC62IND	0	I	
CTRAPA	P0.6	I	input signals for CTRAP, the ESRx input refers to the synchronized input signal, that can be filtered (if enabled)
CTRAPB	P0.7	I	
CTRAPC	ESR2	I	
CTRAPD	P11.1	I	
CCPOS0A	P4.5	I	input signals for CCPOS0
CCPOS0B	0	I	
CCPOS0C	CCU60_SR3	I	
CCPOS0D	0	I	
CCPOS1A	P4.6	I	input signals for CCPOS1
CCPOS1B	0	I	
CCPOS1C	CCU62_SR3	I	
CCPOS1D	0	I	

**Table 20-18 CCU61 Digital Connections in XE16xyM (cont'd)**

Signal	from/to Module	I/O to CC61	Can be used to/as
CCPOS2A	P4.7	I	input signals for CCPOS2
CCPOS2B	0	I	
CCPOS2C	ADC1_SR3	I	
CCPOS2D	0	I	
T12HRA	CCU60_MCM_ST	I	input signals for T12HR
T12HRB	P1.2	I	
T12HRC	P5.8	I	
T12HRD	SYSCON.GLCCST	I	
T12HRE	ADC0_ARBCNT	I	
T12HRF	0	I	
T12HRG	0	I	
T12HRH	0	I	
T13HRA	P5.10	I	input signals for T13HR
T13HRB	CCU61_T12_ZM	I	
T13HRC	P5.8	I	
T13HRD	SYSCON.GLCCST	I	
T13HRE	ADC0_ARBCNT	I	
T13HRF	P11.3	I	
T13HRG	U1C0_SR3	I	
T13HRH	0	I	
CC60	P0.0 P11.5	O	compare outputs of channel CC60
COUT60	P0.3 P11.0	O	
CC61	P0.1 P11.2	O	compare outputs of channel CC61
COUT61	P0.4 P11.1	O	

**Table 20-18 CCU61 Digital Connections in XE16xyM (cont'd)**

<b>Signal</b>	<b>from/to Module</b>	<b>I/O to CC61</b>	<b>Can be used to/as</b>
CC62	P0.2 P11.4	O	compare outputs of channel CC62
COUT62	P0.5 P11.3	O	
COUT63	P0.6 P11.3 P11.5 U1C0_DX2F U1C1_DX2F	O	compare output of channel CC63
COUT63	ADCx_REQGTyB	O	ADC triggers
T12_ZM	CCU61_T13HRB	O	T12 zero match
T13_PM	ERU_OGU12	O	T13 period match
MCM_ST	CCU62_T12HRA ERU_OGU11	O	MCM shadow transfer
SR3	CCU60_CCPOS0C	O	CCU60 trigger
SR3	CCU62_CCPOS1C	O	CCU62 trigger
SR3	ADC0_REQTRyC	O	ADC0 trigger

### 20.10.4.3 Connections of CCU62

This table describes the module interconnections of CCU62.

**Table 20-19 CCU62 Digital Connections in XE16xyM**

Signal	from/to Module	I/O to CCU62	Can be used to/as
CC60INA	P1.7	I	input signals for capture event on channel CC60
CC60INB	P8.3	I	
CC60INC	0	I	
CC60IND	0	I	
CC61INA	P1.6	I	input signals for capture event on channel CC61
CC61INB	P8.4	I	
CC61INC	0	I	
CC61IND	0	I	
CC62INA	P1.2	I	input signals for capture event on channel CC62
CC62INB	P8.5	I	
CC62INC	0	I	
CC62IND	0	I	
CTRAPA	P7.1	I	input signals for CTRAP, the ESRx input refers to the synchronized input signal, which can be filtered (if enabled)
CTRAPB	P1.0	I	
CTRAPC	ESR2	I	
CTRAPD	P8.6	I	
CCPOS0A	P7.2	I	input signals for CCPOS0
CCPOS0B	P4.1	I	
CCPOS0C	CCU63_SR3	I	
CCPOS0D	0	I	
CCPOS1A	P7.3	I	input signals for CCPOS1
CCPOS1B	P4.2	I	
CCPOS1C	CCU61_SR3	I	
CCPOS1D	0	I	

**Table 20-19 CCU62 Digital Connections in XE16xyM (cont'd)**

Signal	from/to Module	I/O to CCU62	Can be used to/as
CCPOS2A	P7.4	I	input signals for CCPOS2
CCPOS2B	P4.3	I	
CCPOS2C	ADC0_SR3	I	
CCPOS2D	0	I	
T12HRA	CCU61_MCM_ST	I	input signals for T12HR
T12HRB	P1.3	I	
T12HRC	P5.8	I	
T12HRD	SYSCON.GLCCST	I	
T12HRE	ADC1_ARBCNT	I	
T12HRF	0	I	
T12HRG	0	I	
T12HRH	0	I	
T13HRA	CAN_INT_O15	I	input signals for T13HR
T13HRB	CCU62_T12_ZM	I	
T13HRC	P5.8	I	
T13HRD	SYSCON.GLCCST	I	
T13HRE	ADC1_ARBCNT	I	
T13HRF	0	I	
T13HRG	U2C0_SR3	I	
T13HRH	0	I	
CC60	P1.7 P8.3	O	compare outputs of channel CC60
COUT60	P1.5 P9.7	O	
CC61	P1.6 P8.4	O	compare outputs of channel CC61
COUT61	P1.4 p9.6	O	

**Table 20-19 CCU62 Digital Connections in XE16xyM (cont'd)**

<b>Signal</b>	<b>from/to Module</b>	<b>I/O to CCU62</b>	<b>Can be used to/as</b>
CC62	P1.2 P8.5	O	compare outputs of channel CC62
COUT62	P1.1 P9.5	O	
COUT63	P1.3 P9.4 P9.7 U2C0_DX2F U2C1_DX2F	O	compare output of channel CC63
COUT63	ADCx_REQGTyC	O	ADC triggers
T12_ZM	CCU62_T13HRB	O	T12 zero match
T13_PM	ERU_OGU22	O	T13 period match
MCM_ST	CCU63_T12HRA ERU_OGU21	O	MCM shadow transfer
SR3	CCU61_CCPOS1C	O	CCU61 trigger
SR3	CCU63_CCPOS0C	O	CCU63 trigger
SR3	ADC1_REQTRyC	O	ADC1 trigger

#### 20.10.4.4 Connections of CCU63

This table describes the module interconnections of CCU63.

**Table 20-20 CCU63 Digital Connections in XE16xyM**

Signal	from/to Module	I/O to CCU63	Can be used to/as
CC60INA	P9.0	I	input signals for capture event on channel CC60
CC60INB	P2.0	I	
CC60INC	0	I	
CC60IND	0	I	
CC61INA	P9.1	I	input signals for capture event on channel CC61
CC61INB	P2.1	I	
CC61INC	0	I	
CC61IND	0	I	
CC62INA	P9.2	I	input signals for capture event on channel CC62
CC62INB	P2.2	I	
CC62INC	0	I	
CC62IND	0	I	
CTRAPA	P9.6	I	input signals for CTRAP, the ESRx input refers to the synchronized input signal, which can be filtered (if enabled)
CTRAPB	P9.7	I	
CTRAPC	ESR2	I	
CTRAPD	ERU_PDOUT3	I	
CCPOS0A	P11.0	I	input signals for CCPOS0
CCPOS0D	0	I	
CCPOS0C	CCU62_SR3	I	
CCPOS0D	0	I	
CCPOS1A	P11.1	I	input signals for CCPOS1
CCPOS1B	0	I	
CCPOS1C	CCU60_SR3	I	
CCPOS1D	0	I	

**Table 20-20 CCU63 Digital Connections in XE16xyM (cont'd)**

<b>Signal</b>	<b>from/to Module</b>	<b>I/O to CCU63</b>	<b>Can be used to/as</b>
CCPOS2A	P11.2	I	input signals for CCPOS2
CCPOS2B	0	I	
CCPOS2C	ADC1_SR3	I	
CCPOS2D	0	I	
T12HRA	CCU62_MCM_ST	I	input signals for T12HR
T12HRB	P5.4	I	
T12HRC	P5.8	I	
T12HRD	SYSCON.GLCCST	I	
T12HRE	ADC1_ARBCNT	I	
T12HRF	0	I	
T12HRG	0	I	
T12HRH	0	I	
T13HRA	CAN_INT_O15	I	input signals for T13HR
T13HRB	CCU63_T12_ZM	I	
T13HRC	P5.8	I	
T13HRD	SYSCON.GLCCST	I	
T13HRE	ADC1_ARBCNT	I	
T13HRF	P5.13	I	
T13HRG	U3C0_SR3	I	
T13HRH	0	I	
CC60	P9.0 P2.0	O	compare outputs of channel CC60
CC60	ADC1_REQGT0H	O	
COUT60	P9.3	O	
CC61	P9.1 P2.1	O	compare outputs of channel CC61
CC61	ADC1_REQGT1H	O	
COUT61	P9.4	O	



**Table 20-20 CCU63 Digital Connections in XE16xyM (cont'd)**

<b>Signal</b>	<b>from/to Module</b>	<b>I/O to CCU63</b>	<b>Can be used to/as</b>
CC62	P9.2 P2.2	O	compare outputs of channel CC62
CC62	ADC1_REQGT2H	O	
COUT62	P9.5 P9.6	O	
COUT63	P9.6 P2.3 U3C0_DX2F U3C1_DX2F	O	compare output of channel CC63
COUT63	ADCx_REQGTyD	O	ADC triggers
T12_ZM	CCU63_T13HRB	O	T12 zero match
T13_PM	ERU_OGU32	O	T13 period match
MCM_ST	CCU60_T12HRA ERU_OGU31	O	MCM shadow transfer
SR3	CCU60_CCPOS1C	O	CCU60 trigger
SR3	CCU62_CCPOS0C	O	CCU62 trigger

## **21 Universal Serial Interface Channel**

The **Universal Serial Interface Channel** module (USIC) is a flexible interface module covering several serial communication protocols. A USIC module contains two independent communication channels named UxC0 and UxC1, with x being the number of the USIC module (e.g. channel y of USIC module x is referenced as UxCy). The user can program during run-time which protocol will be handled by each communication channel and which pins are used.

This chapter is structured as follows:

- Introduction (see [Page 21-1](#))
- Operating the USIC (see [Page 21-13](#))
- ASC protocol for UART and LIN (see [Page 21-110](#))
- SSC protocol (see [Page 21-131](#))
- IIC protocol (see [Page 21-161](#))
- IIS protocol (see [Page 21-185](#))
- Module implementation in XE16xyM (see [Page 21-205](#))

### **21.1 Introduction**

This section gives an overview about the feature set of the USIC and introduces the USIC structure. It describes the:

- Feature set overview (see [Page 21-2](#))
- Channel structure (see [Page 21-5](#))
- Input stages (see [Page 21-6](#))
- Output signals (see [Page 21-7](#))
- Baud rate generator (see [Page 21-8](#))
- Channel events and interrupts (see [Page 21-9](#))
- Data shifting and handling (see [Page 21-9](#))

### **21.1.1 Feature Set Overview**

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  $f_{\text{SYS}} / 4$
  - Wide baud rate range down to single-digit baud rates
  - Number of data bits per data frame: 1 to 63
  - MSB or LSB first
- **LIN** Support by hardware (Local Interconnect Network)
  - Data transfers based on ASC protocol
  - Baud rate detection possible by built-in capture event of baud rate generator
  - Checksum generation under software control for higher flexibility
- **SSC/SPI** (synchronous serial channel with or without slave select lines)
  - Module capability: maximum baud rate  $f_{\text{SYS}} / 2$ , limited by loop delay
  - Number of data bits per data frame 1 to 63, more with explicit stop condition
  - 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  $f_{\text{SYS}} / 2$

*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).

## **Universal Serial Interface Channel**

In addition to the FIFO buffer, a bypass mechanism allows the introduction of high-priority data without flushing the FIFO buffer.

- **Transmit control information**

For each data word to be transmitted, a 5-bit transmit control information has been added to automatically control some transmission parameters, such as word length, frame length, or the slave select control for the SPI protocol. The transmit control information is generated automatically by analyzing the address where the user software has written the data word to be transmitted (32 input locations =  $2^5 = 5$  bit transmit control information).

This feature allows individual handling of each data word, e.g. the transmit control information associated to the data words stored in a transmit FIFO can automatically modify the slave select outputs to select different communication targets (slave devices) without CPU load. Alternatively, it can be used to control the frame length.

- **Flexible frame length control**

The number of bits to be transferred within a data frame is independent of the data word length and can be handled in two different ways. The first option allows automatic generation of frames up to 63 bits with a known length. The second option supports longer frames (even unlimited length) or frames with a dynamically controlled length.

- **Interrupt capability**

The events of each USIC channel can be individually routed to one of 4 service request outputs SR[3:0], 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 an 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. at 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

### **Universal Serial Interface Channel**

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.

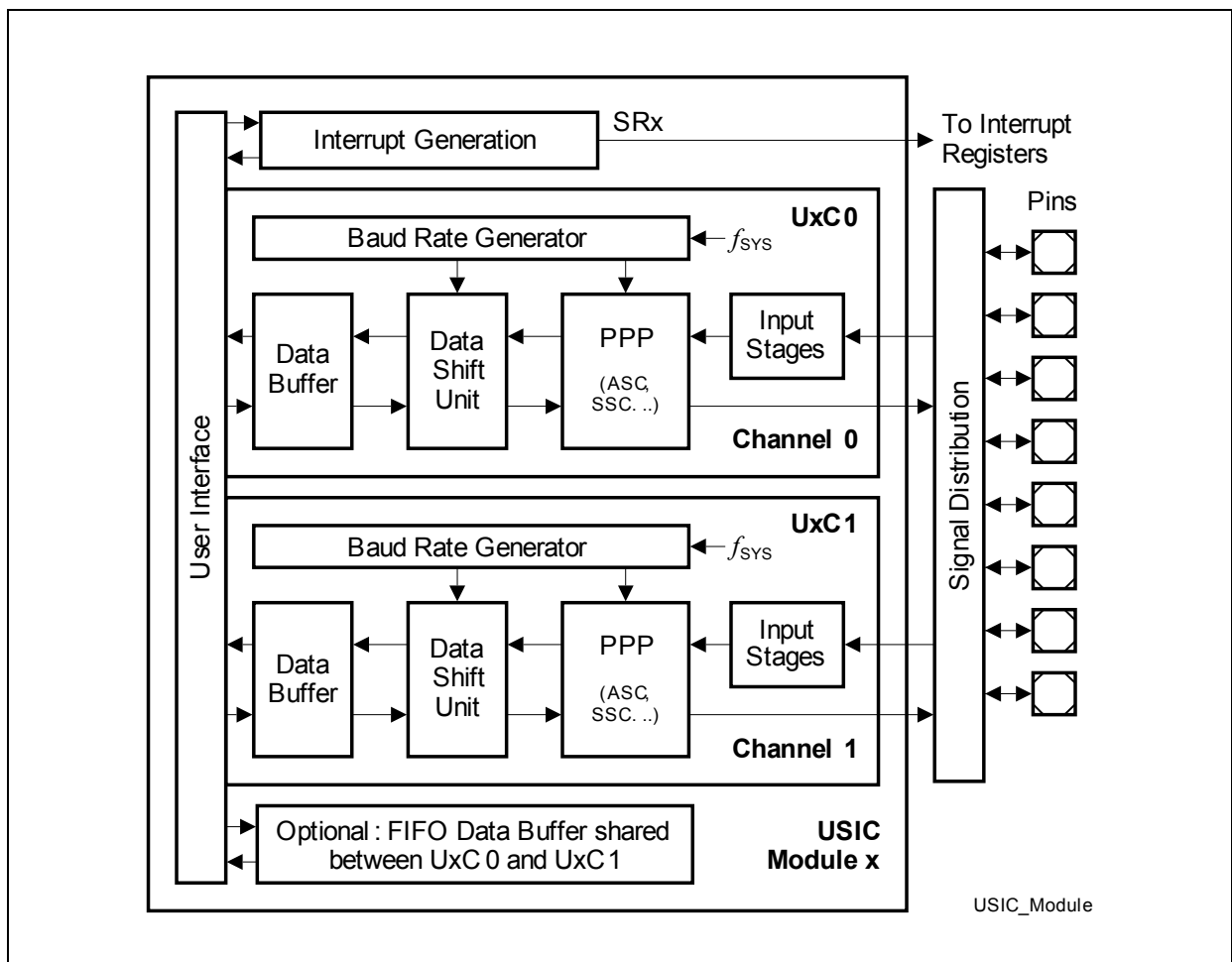
*Note: Depending on the selected additional functions (such as digital filters, input synchronization stages, sample point adjustment, data structure, etc.), the maximum reachable baud rate can be limited. Please also take care about additional delays, such as (internal or external) propagation delays and driver delays (e.g. for collision detection in ASC mode, for IIC, etc.).*

### 21.1.2 Channel Structure

The USIC module contains two independent communication channels, with a structure as shown in [Figure 21-1](#).

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. This FIFO data buffer is not necessarily available in all devices (see [Section 21.7.2](#)).

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.



**Figure 21-1 USIC Module/Channel Structure**

### 21.1.3 Input Stages

For each protocol up to three input signals are available, the number of actually used inputs depends on the selected protocol. Each input signal is handled by an input stage (called DX0, DX1, DX2) 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 21-1](#).

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, and for the transmit data validation to the data handling section.

**Table 21-1 Input Signals for Different Protocols**

<b>Selected Protocol</b>	<b>Shift Data Input (handled by DX0)</b>	<b>Shift Clock Input (handled by DX1)</b>	<b>Shift Control Input (handled by DX2)</b>
<b>ASC, LIN</b>	RXD	optional: external frequency input or TXD collision detection	optional: transmit data validation
<b>SSC, SPI (Master)</b>	DIN (MRST, MISO)	optional: external frequency input or delay compensation	optional: transmit data validation or delay compensation
<b>SSC, SPI (Slave)</b>	DIN (MSTR, MOSI)	SCLKIN	SELIN
<b>IIC</b>	SDA	SCL	optional: transmit data validation
<b>IIS (Master)</b>	DIN	optional: external frequency input or delay compensation	optional: transmit data validation or delay compensation
<b>IIS (Slave)</b>	DIN	SCLKIN	WAIN

*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 implementation chapter, see [Page 21-212](#).*

### 21.1.4 Output Signals

For each protocol up to eleven 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 21-2](#).

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 21-2 Output Signals for Different Protocols**

<b>Selected Protocol</b>	<b>Shift Data Output DOUT</b>	<b>Shift Clock Output SCLKOUT</b>	<b>Shift Control Outputs SELO[7:0]</b>	<b>Master Clock Output MCLKOUT</b>
<b>ASC, LIN</b>	TXD	not used	not used	optional: master time base
<b>SSC, SPI (master)</b>	DOUT (MTSR, MOSI)	master shift clock	slave select, chip select	optional: master time base
<b>SSC, SPI (slave)</b>	DOUT (MRST, MISO)	optional: independent clock output	not used	optional: independent clock output
<b>IIC</b>	SDA	SCL	not used	optional: master time base
<b>IIS (master)</b>	DOUT	master shift clock	WA	optional: master time base
<b>IIS (slave)</b>	DOUT	optional: independent clock output	not used	optional: independent clock 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 implementation chapter, see [Page 21-212](#).*

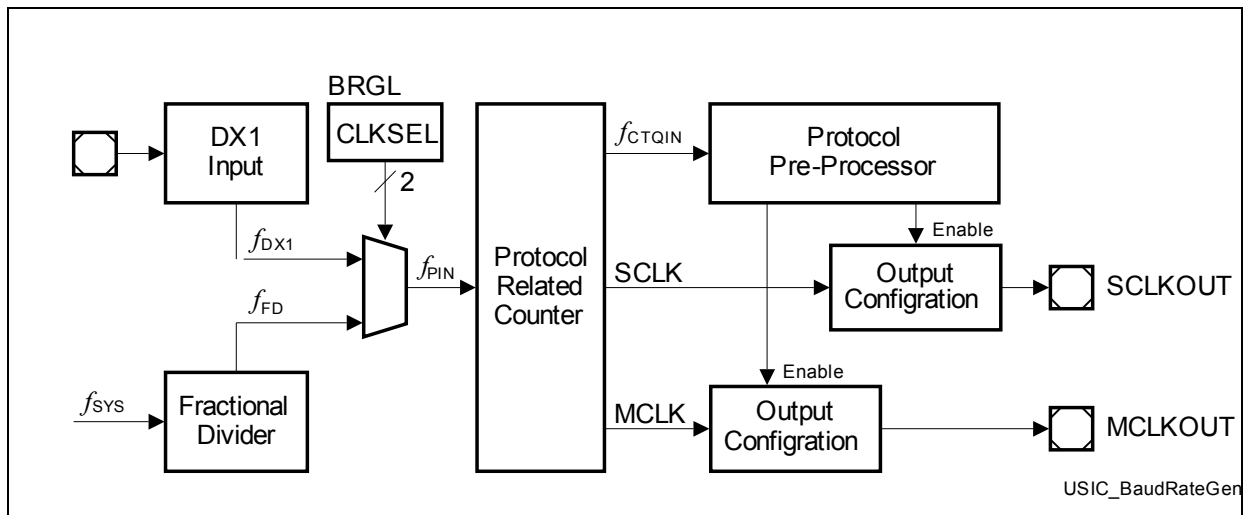


### 21.1.5 Baud Rate Generator

Each USIC Channel contains a baud rate generator structured as shown in **Figure 21-2**. 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  $f_{SYS}$ .
- The DX1 input to generate the input frequency  $f_{PIN} = f_{DX1}$  for baud rate generation based on an external signal.
- A protocol-related counter to provide the master clock signal MCLK, the shift clock signal SCLK, and other protocol-related signals. It can also be used 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 implementation section.



**Figure 21-2 Baud Rate Generator**

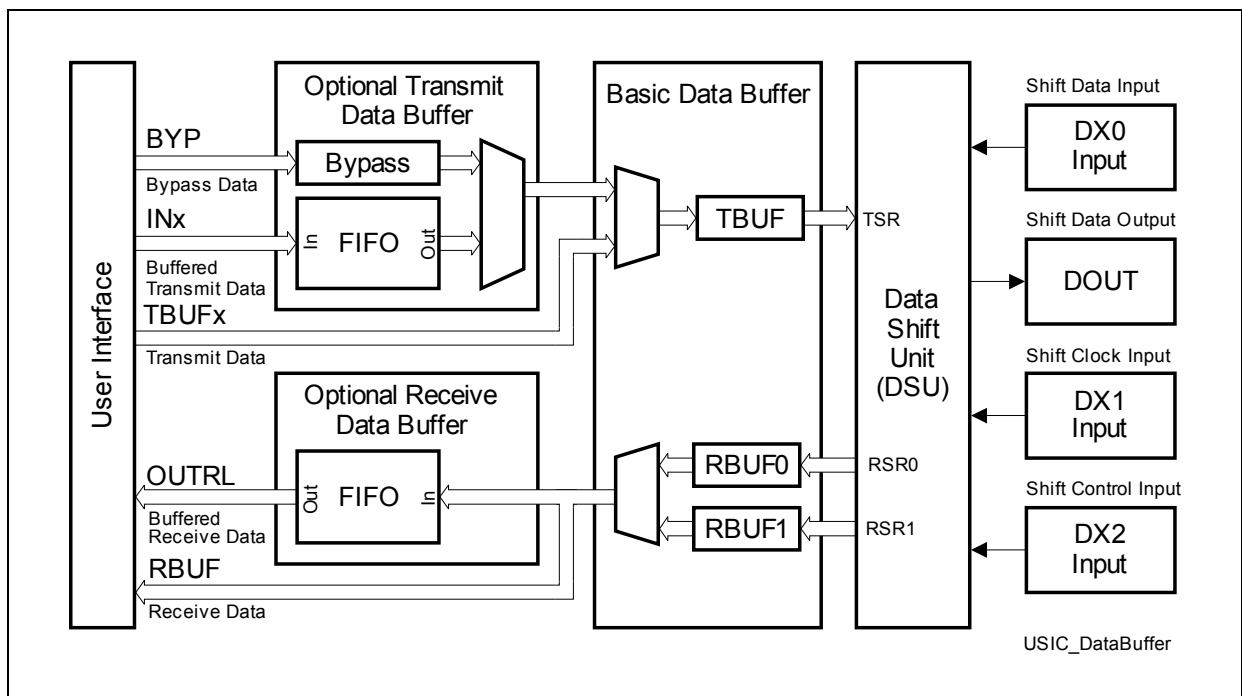
### 21.1.6 Channel Events and Interrupts

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.

### 21.1.7 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), the shift clock (handled by the input stage DX1), and the shift control (handled by the input stage DX2). The signal DOUT represents the shift data output.



**Figure 21-3 Principle of Data Buffering**

The principle of data handling comprises:

- A transmitter with a transmit shift register (TSR) 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 receive shift registers (RSR0 and RSR1) in the DSU and a double receive buffer structure (RBUF0, RBUF1). The alternating receive shift

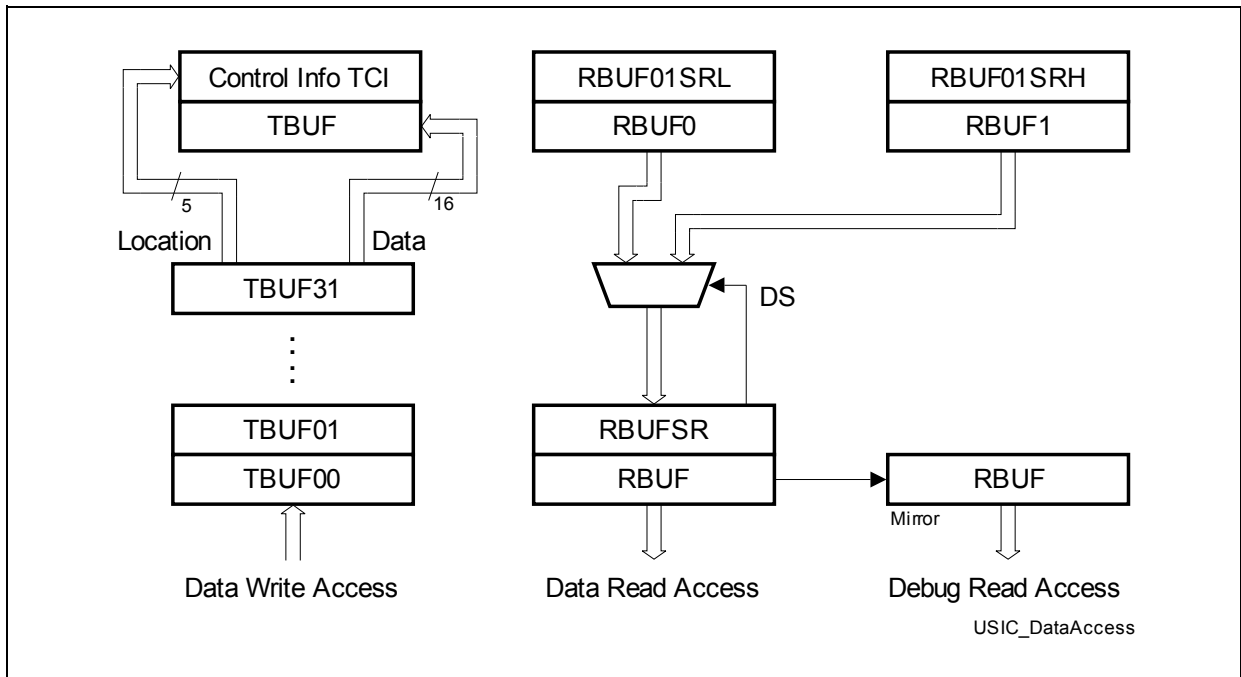
registers support the reception of data streams and data frames longer than one data word.

- Optional transmit and receive data buffers according to the first-in-first-out principle (FIFO), that are not necessarily available in all devices (see [Section 21.7.2](#))
- A user interface to handle data, interrupts, and status and control information.

### 21.1.7.1 Basic Data Buffer Structure

The read access to received data and the write access of data to be transmitted can be handled by a basic data buffer structure.

The received data stored in the receiver buffers RBUF0/RBUF1 can be read directly from these registers. In this case, the user has to take care about the reception sequence to read these registers in the correct order. To simplify the use of the receive buffer structure, register RBUF has been introduced. A read action from this register delivers the data word received first (oldest data) to respect the reception sequence. With a read access from at least the low byte of RBUF, the data is automatically declared to be no longer new and the next received data word becomes visible in RBUF and can be read out next.



**Figure 21-4 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 registers RBUF01SRL/H (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 TBUF<sub>x</sub> (x = 00-31), consisting of 32 consecutive addresses. The data written to one of these input locations is stored in the transmit buffer TBUF. Additionally, the address of the written location is evaluated and can be used for additional control purposes. This 5-bit wide information (named **Transmit Control Information TCI**) can be used for different purposes in different protocols.

### **21.1.7.2 FIFO Buffer Structure**

To allow easier data setup and handling, an additional data buffering mechanism can be optionally supported. The data buffer is based on the first-in-first-out principle (FIFO) that ensures that the sequence of transferred data words is respected.

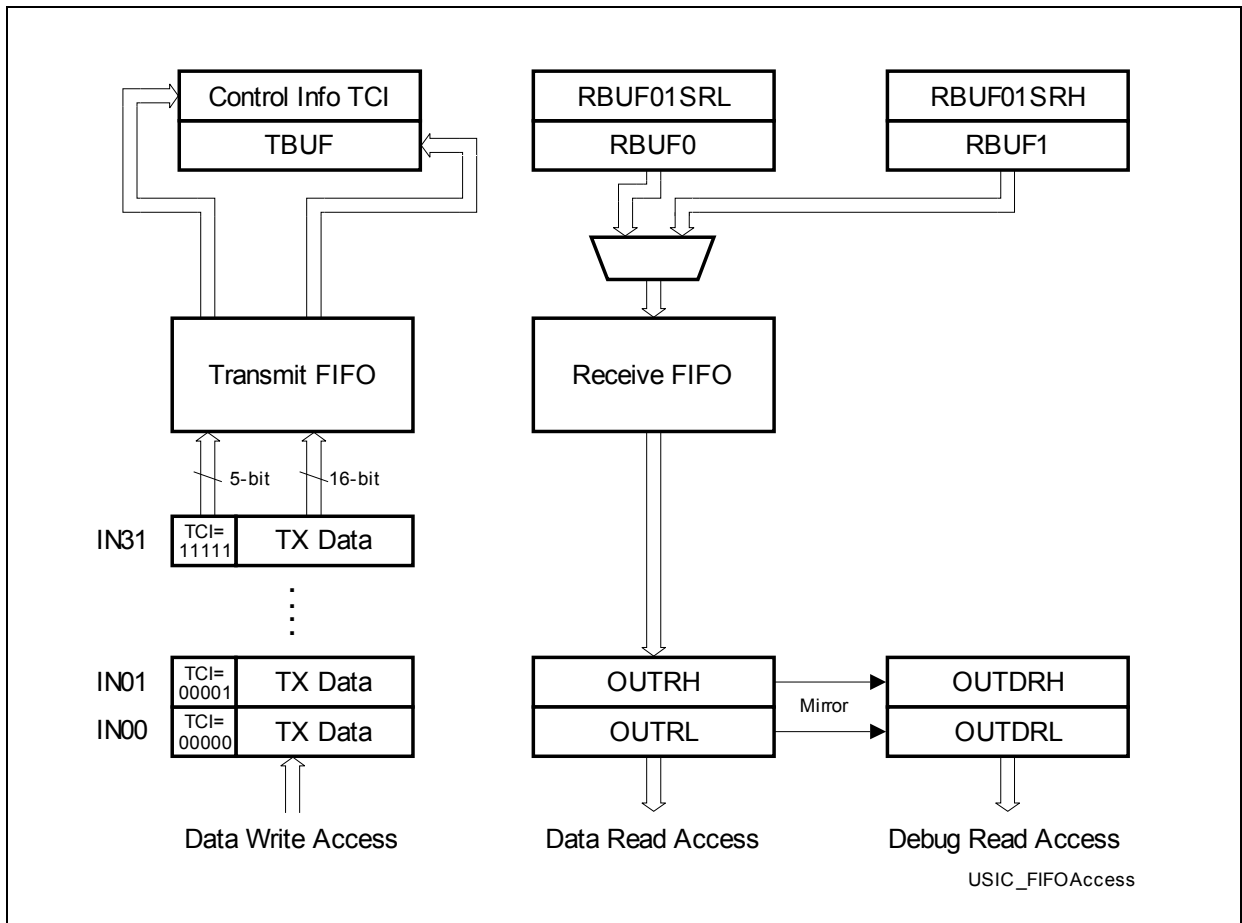
If a FIFO buffer structure is used, the data handling scheme (data with associated control information) is similar to the one without FIFO. The additional FIFO buffer can be independently enabled/disabled for transmission and reception (e.g. if data FIFO buffers are available for a specific USIC channel, it is possible to configure the transmit data path without and the receive data path with FIFO buffering).

The transmit FIFO buffer is addressed by using 32 consecutive address locations for IN<sub>x</sub> instead of TBUF<sub>x</sub> (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 OUTDRL 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 (OUTDRL) 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: For more information on operating the FIFO buffers, see [Section 21.2.13](#).*



**Figure 21-5 Data Access Structure with FIFO**

## **21.2 Operating the USIC**

This section describes how to operate the USIC communication channel.

It describes:

- Register Overview (see [Page 21-13](#))
- General channel operation (see [Page 21-18](#))
- Channel control and configuration registers (see [Page 21-25](#))
- Protocol related registers (see [Page 21-33](#))
- Input stages (see [Page 21-36](#))
- Input stage control registers (see [Page 21-38](#))
- Baud rate generation (see [Page 21-41](#))
- Baud rate and shift control registers (see [Page 21-46](#))
- Operating the transmit path (see on [Page 21-51](#))
- Operating the receive path (see [Page 21-55](#))
- Transfer control and status registers (see [Page 21-57](#))
- Data buffer registers (see [Page 21-69](#))
- Operating the FIFO data buffer (see [Page 21-79](#))
- FIFO buffer and bypass registers (see [Page 21-89](#))

### **21.2.1 Register Overview**

The module itself being 32-bit wide, some registers have been split up in two parts for the 16-bit implementation. Both parts keep the same name as the former 32-bit register, with an additional index. The lower part ends with the index L, whereas the upper (higher) part ends with the index H. Former 32-bit registers consisting of only 16 used bits keep their name (without additional index), because only the used bits appear in the register map.

**Table 21-3** 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.

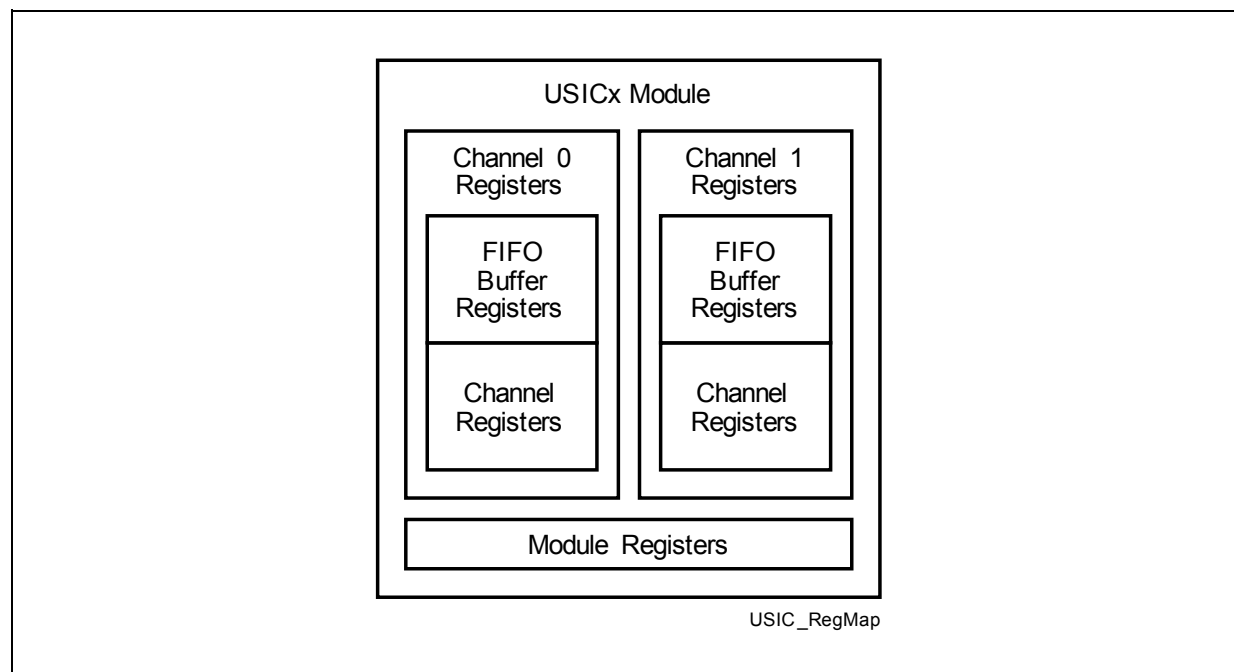
Please note that all registers can be accessed with any access width (8-bit, 16-bit), independent of the described width. Short addressing is not supported.

All USIC registers (except bit field KSCFG.SUMCFG) are always reset by an application 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.*

## Universal Serial Interface Channel

**Figure 21-6** 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 “UxC0\_” and “UxC1\_”.



**Figure 21-6 USIC Module and Channel Registers**

**Table 21-3 USIC Kernel-Related and Kernel Registers**

Register Short Name	Register Long Name	Offset Addr.	Reset Value	Description see
<b>Module Registers<sup>1)</sup></b>				
IDL	Module Identification Register L	008 <sub>H</sub>	C0XX <sub>H</sub>	<a href="#">Page 21-208</a>
IDH	Module Identification Register H	00A <sub>H</sub>	003A <sub>H</sub>	<a href="#">Page 21-209</a>
<b>Channel Registers</b>				
FDRL	Fractional Divider Register L	004 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 21-46</a>
FDRH	Fractional Divider Register H	006 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 21-47</a>
KSCFG	Kernel State Configuration Register	00C <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 21-29</a>
CCR	Channel Control Register	010 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 21-25</a>
INPRL	Interrupt Node Pointer Register L	014 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 21-31</a>
INPRH	Interrupt Node Pointer Register H	016 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 21-32</a>
CCFG	Channel Configuration Register	018 <sub>H</sub>	00CF <sub>H</sub>	<a href="#">Page 21-28</a>
BRGL	Baud Rate Generator Register L	01C <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 21-48</a>

**Universal Serial Interface Channel**

**Table 21-3 USIC Kernel-Related and Kernel Registers (cont'd)**

<b>Register Short Name</b>	<b>Register Long Name</b>	<b>Offset Addr.</b>	<b>Reset Value</b>	<b>Description see</b>
BRGH	Baud Rate Generator Register H	01E <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 21-50</a>
DX0CR	Input Control Register 0	020 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 21-38</a>
DX1CR	Input Control Register 1	024 <sub>H</sub>	0000 <sub>H</sub>	
DX2CR	Input Control Register 2	028 <sub>H</sub>	0000 <sub>H</sub>	
SCTRL	Shift Control Register L	030 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 21-57</a>
SCTRH	Shift Control Register H	032 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 21-59</a>
FMRL	Flag Modification Register L	038 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 21-67</a>
FMRH	Flag Modification Register H	03A <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 21-68</a>
TCSRL	Transmit Control/Status Register L	03C <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 21-60</a>
TCSRH	Transmit Control/Status Register H	03E <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 21-65</a>
PCRL	Protocol Control Register L	040 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 21-33</a> <sup>2)</sup>
				<a href="#">Page 21-123</a> <sup>3)</sup>
				<a href="#">Page 21-152</a> <sup>4)</sup>
				<a href="#">Page 21-179</a> <sup>5)</sup>
				<a href="#">Page 21-199</a> <sup>6)</sup>
PCRH	Protocol Control Register H	042 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 21-33</a> <sup>2)</sup>
				<a href="#">Page 21-126</a> <sup>3)</sup>
				<a href="#">Page 21-154</a> <sup>4)</sup>
				<a href="#">Page 21-179</a> <sup>5)</sup>
				<a href="#">Page 21-201</a> <sup>6)</sup>



**Universal Serial Interface Channel**

**Table 21-3 USIC Kernel-Related and Kernel Registers (cont'd)**

<b>Register Short Name</b>	<b>Register Long Name</b>	<b>Offset Addr.</b>	<b>Reset Value</b>	<b>Description see</b>
PSR	Protocol Status Register	044 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 21-34</a> <sup>2)</sup>
				<a href="#">Page 21-127</a> <sup>3)</sup>
				<a href="#">Page 21-156</a> <sup>4)</sup>
				<a href="#">Page 21-182</a> <sup>5)</sup>
				<a href="#">Page 21-202</a> <sup>6)</sup>
PSCR	Protocol Status Clear Register	048 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 21-35</a>
RBUFD	Receiver Buffer Register for Debugger	04C <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 21-77</a>
RBUF0	Receiver Buffer Register 0	050 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 21-70</a>
RBUF1	Receiver Buffer Register 1	054 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 21-73</a>
RBUFSR	Receiver Buffer Status Register	058 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 21-78</a>
RBUF	Receiver Buffer Register	05C <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 21-76</a>
RBUF01SRL	Receiver Buffer 01 Status Register L	060 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 21-70</a>
RBUF01SRH	Receiver Buffer 01 Status Register H	062 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 21-73</a>
–	Reserved; do not access this location.	06C <sub>H</sub>	–	–
–	Reserved; do not access this location.	06E <sub>H</sub>	–	–
TBUFx	Transmit Buffer Input Location x (x = 00-31)	080 <sub>H</sub> + x*4	0000 <sub>H</sub>	<a href="#">Page 21-69</a>
<b>FIFO Buffer Registers</b>				
BYP	Bypass Data Register	100 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 21-89</a>
BYPCLR	Bypass Control Register L	104 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 21-89</a>
BYPCHR	Bypass Control Register H	106 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 21-91</a>
TRBPTRL	Transmit/Receive Buffer Pointer Register L	108 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 21-108</a>
TRBPTRH	Transmit/Receive Buffer Pointer Register H	10A <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 21-109</a>

**Universal Serial Interface Channel**

**Table 21-3 USIC Kernel-Related and Kernel Registers (cont'd)**

<b>Register Short Name</b>	<b>Register Long Name</b>	<b>Offset Addr.</b>	<b>Reset Value</b>	<b>Description see</b>
TBCTRL	Transmit Buffer Control Register L	110 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 21-98</a>
TBCTRH	Transmit Buffer Control Register H	112 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 21-99</a>
RBCTRL	Receive Buffer Control Register L	114 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 21-101</a>
RBCTRH	Receive Buffer Control Register H	116 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 21-102</a>
TRBSRL	Transmit/Receive Buffer Status Register L	118 <sub>H</sub>	0808 <sub>H</sub>	<a href="#">Page 21-92</a>
TRBSRH	Transmit/Receive Buffer Status Register H	11A <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 21-95</a>
TRBSCR	Transmit/Receive Buffer Status Clear Register	11C <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 21-96</a>
OUTRL	Receive Buffer Output Register L	120 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 21-106</a>
OUTRH	Receive Buffer Output Register H	122 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 21-106</a>
OUTDRL	Receive Buffer Output Register L for Debugger	124 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 21-107</a>
OUTDRH	Receive Buffer Output Register H for Debugger	126 <sub>H</sub>	0000 <sub>H</sub>	<a href="#">Page 21-107</a>
INx	Transmit FIFO Buffer Input Location x (x = 00-31)	180 <sub>H</sub> + x*4	0000 <sub>H</sub>	<a href="#">Page 21-105</a>

- 1) Details of the module identification registers are described in the implementation section (see [Page 21-208](#)).
- 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.

### **21.2.2 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 21-19](#)).
- 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.

#### **21.2.2.1 Protocol Control and Status**

The protocol-related control and status information are located in the protocol control registers PCRL and PCRH 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 PCRL/H Bits**

The signification of the bits in registers PCRL/PCRH is indicated by the protocol-related alias names for the different protocols.

- PCRL/PRCH for the ASC protocol (see [Page 21-123](#))
- PCRL/PRCH for the SSC protocol (see [Page 21-152](#))
- PCRL/PRCH for the IIC protocol (see [Page 21-179](#))
- PCRL/PRCH for the IIS protocol (see [Page 21-199](#))

##### **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 21-127](#))
- PSR flags for the SSC protocol (see [Page 21-156](#))
- PSR flags for the IIC protocol (see [Page 21-182](#))
- PSR flags for the IIS protocol (see [Page 21-202](#))

### 21.2.2.2 Mode Control

The mode control concept for system control tasks, such as power saving, or 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, that are requested by the global state control part of the SCU. 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 neither a suspend request nor a clock-off request are 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.
- Clock-off mode:  
This operating mode is requested for power saving purposes. The module clock is switched off automatically when all channels of the USIC module reached their specified state in a stop mode. In this case, USIC registers can not be accessed. The channel behavior is defined by KSCFG.COMCFG.

The behavior of a USIC communication channel can be programmed for each of the device operating modes (normal operation, suspend mode, clock-off mode). Therefore, the USIC communication channel provides four kernel modes, as shown in [Table 21-4](#).

**Table 21-4 USIC Communication Channel Behavior**

Kernel Mode	Channel Behavior	KSCFG. NOMCFG
Run mode 0	Channel operation as specified, no impact on data transfer	00 <sub>B</sub>
Run mode 1		01 <sub>B</sub>
Stop mode 0	Explicit stop condition as described in the protocol chapters	10 <sub>B</sub>
Stop mode 1		11 <sub>B</sub>

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.

## Universal Serial Interface Channel

A similar mechanism applies for the clock-off mode with the possibility to program the desired behavior by bit field KSCFG.COMCFG.

The stop conditions are defined for the selected protocol (see mode control description in the protocol section).

*Note: The stop mode selection strongly depends on the application needs and it is very unlikely that different stop modes are required in parallel in the same application. As a result, only one stop mode type (either 0 or 1) should be used in the bit fields in register KSCFG. Do not mix stop mode 0 and stop mode 1 and avoid transitions from stop mode 0 to stop mode 1 (or vice versa) for the same communication channel.*

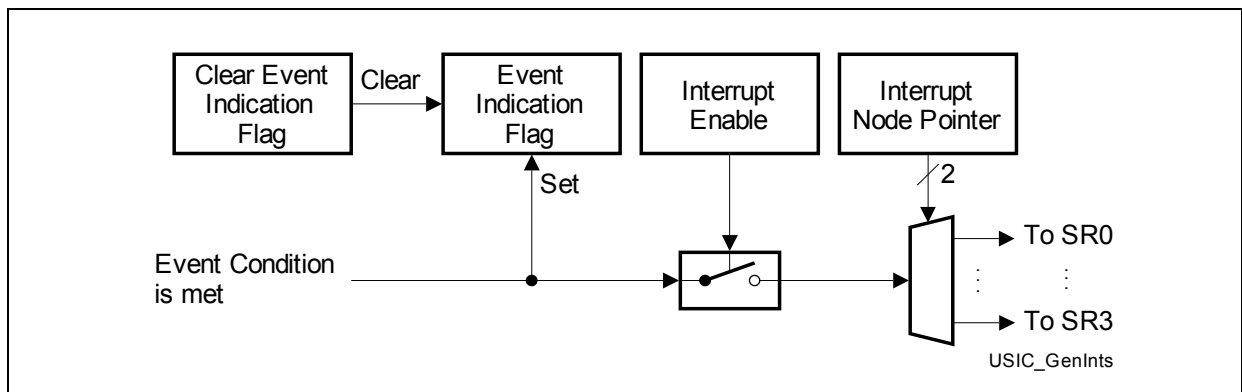
If the module clock is disabled by KSCFG.MODEN = 0 or in clock-off mode when the stop condition is reached (in stop mode 0 or 1), the module can not be accessed by read or write operations (except register KSCFG that can always be accessed).

### 21.2.2.3 General Channel Events and Interrupts

The general event and interrupt structure is shown in [Figure 21-7](#). 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 21-210](#).



**Figure 21-7 General Event and Interrupt Structure**

#### **21.2.2.4 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 RBUF, 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 TCSRL.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 RBUF, 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 TCSRL.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)

### Universal Serial Interface Channel

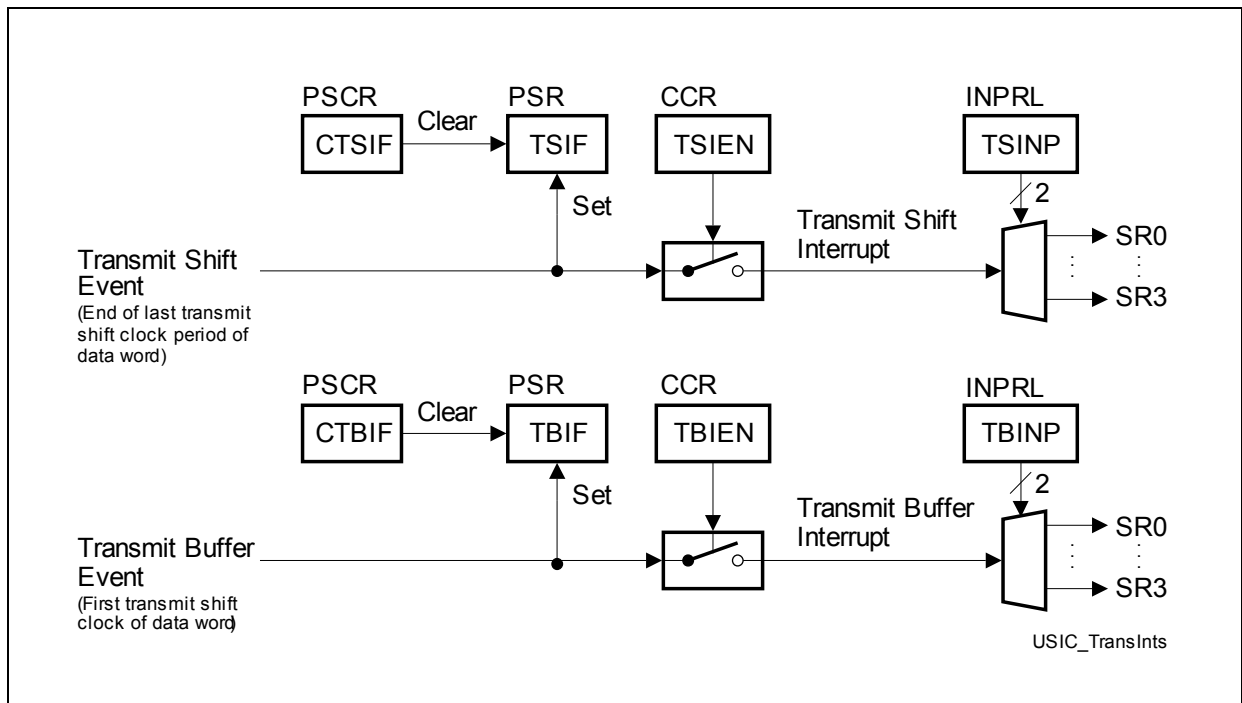
has not been read out before it becomes overwritten with new incoming data, this event occurs. It is indicated by flag PSR.DLIF and, if enabled, leads to a protocol interrupt.

**Table 21-5** shows the registers, bits and bit fields indicating the data transfer events and controlling the interrupts of a USIC channel.

**Table 21-5 Data Transfer Events and Interrupt Handling**

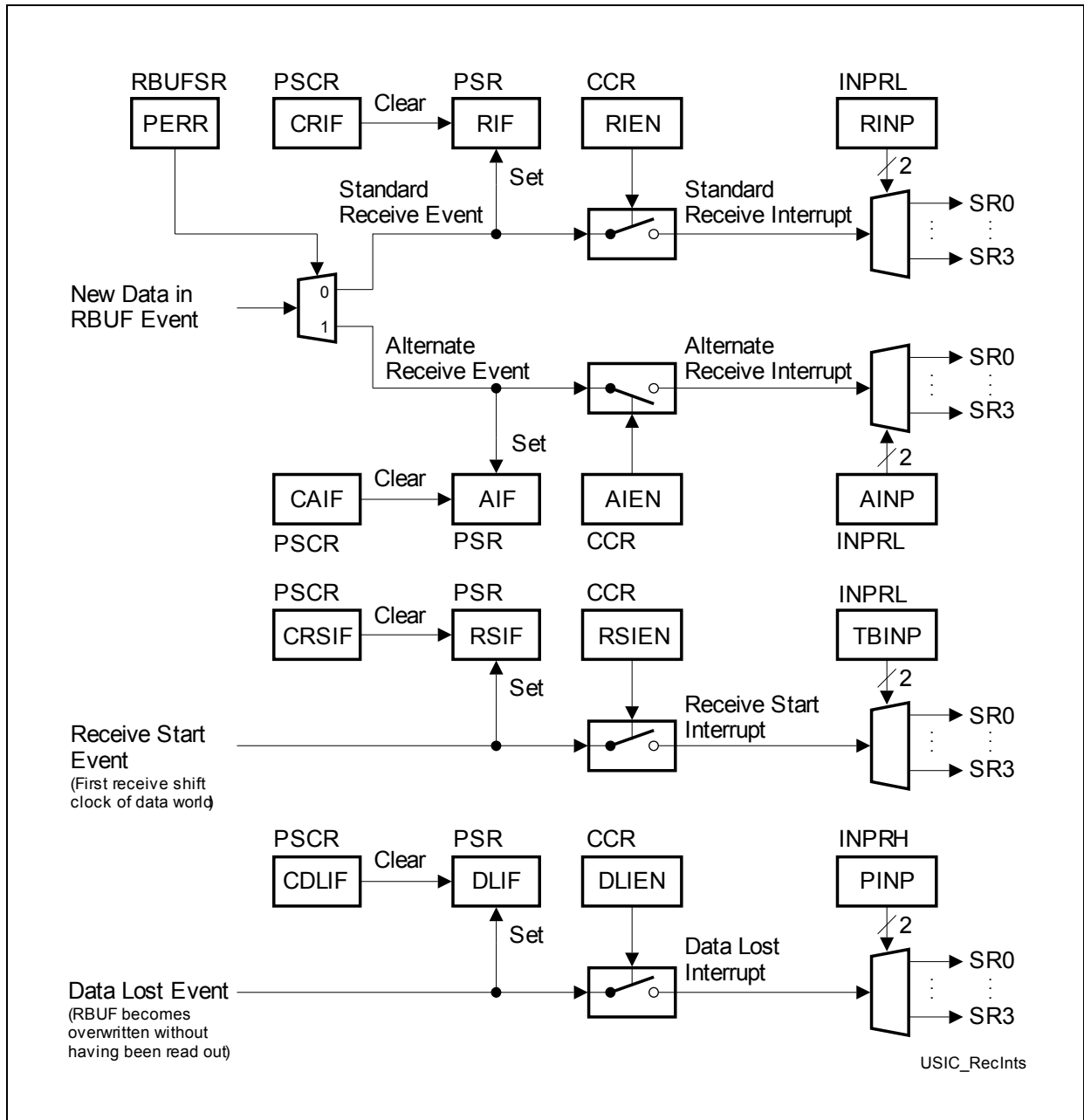
Event	Indication Flag	Indication cleared by	Interrupt enabled by	SRx Output selected by
Standard receive event	PSR.RIF	PSCR.CRIF	CCR.RIEN	INPRL.RINP
Receive start event	PSR.RSIF	PSCR.CRSIF	CCR.RSIEN	INPRL.TBINP
Alternative receive event	PSR.AIF	PSCR.CAIF	CCR.AIEN	INPRL.AINP
Transmit shift event	PSR.TSIF	PSCR.CTSIF	CCR.TSIEN	INPRL.TSINP
Transmit buffer event	PSR.TBIF	PSCR.CTBIF	CCR.TBIEN	INPRL.TBINP
Data lost event	PSR.DLIF	PSCR.CDLIF	CCR.DLIEN	INPRH.PINP

**Figure 21-8** shows the two transmit events and interrupts.



**Figure 21-8 Transmit Events and Interrupts**

**Figure 21-9** shows the receive events and interrupts.



**Figure 21-9 Receive Events and Interrupts**



### 21.2.2.5 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
- 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 21-6 Protocol-specific Events and Interrupt Handling**

<b>Event</b>	<b>Indication Flag</b>	<b>Indication cleared by</b>	<b>Interrupt enabled by</b>	<b>SRx Output selected by</b>
Protocol-specific events in ASC mode	PSR.ST[8:2]	PSCR.CST[8:2]	PCRL.CTR[7:3]	INPRH.PINP
Protocol-specific events in SSC mode	PSR.ST[3:2]	PSCR.CST[3:2]	PCRL.CTR[15:14]	INPRH.PINP
Protocol-specific events in IIC mode	PSR.ST[8:1]	PSCR.CST[8:1]	PCRH.CTR[24:18]	INPRH.PINP
Protocol-specific events in IIS mode	PSR.ST[6:3]	PSCR.CST[6:3]	PCRL.CTR[6:4], PCRL.CTR[15]	INPRH.PINP

## 21.2.3 Channel Control and Configuration Registers

### 21.2.3.1 Channel Control Register

The channel control register contains the enable/disable bits for interrupt generation on channel events, the control of the parity generation and the protocol selection of a USIC channel.

#### CCR

#### Channel Control Register

(10<sub>H</sub>)

Reset Value: 0000<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>AI</b> <b>EN</b>	<b>RI</b> <b>EN</b>	<b>TBI</b> <b>EN</b>	<b>TSI</b> <b>EN</b>	<b>DLI</b> <b>EN</b>	<b>RSI</b> <b>EN</b>	<b>PM</b>		<b>0</b>			<b>MODE</b>				
rw	rw	rw	rw	rw	rw	rw		r			rw				

Field	Bits	Type	Description
<b>MODE</b>	[3:0]	rw	<b>Operating Mode</b> 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 PCRH, PCRL, and PSR have to be cleared or updated by software. 0 <sub>H</sub> The USIC channel is disabled. All protocol-related state machines are set to an idle state. 1 <sub>H</sub> The SSC (SPI) protocol is selected. 2 <sub>H</sub> The ASC (SCI, UART) protocol is selected. 3 <sub>H</sub> The IIS protocol is selected. 4 <sub>H</sub> The IIC protocol is selected. Other bit combinations are reserved.

**Universal Serial Interface Channel**

<b>Field</b>	<b>Bits</b>	<b>Type</b>	<b>Description</b>
<b>PM</b>	[9:8]	rw	<b>Parity Mode</b> This bit field defines the parity generation of the sampled input values. 00 <sub>B</sub> The parity generation is disabled. 01 <sub>B</sub> Reserved 10 <sub>B</sub> Even parity is selected (parity bit = 1 on odd number of 1s in data, parity bit = 0 on even number of 1s in data). 11 <sub>B</sub> Odd parity is selected (parity bit = 0 on odd number of 1s in data, parity bit = 1 on even number of 1s in data).
<b>RSIEN</b>	10	rw	<b>Receiver Start Interrupt Enable</b> This bit enables the interrupt generation in case of a receiver start event. 0 <sub>B</sub> The receiver start interrupt is disabled. 1 <sub>B</sub> The receiver start interrupt is enabled. In case of a receiver start event, the service request output SRx indicated by INPRL.TBINP is activated.
<b>DLIEN</b>	11	rw	<b>Data Lost Interrupt Enable</b> This bit enables the interrupt generation in case of a data lost event (data received in RBUFx while RDVx = 1). 0 <sub>B</sub> The data lost interrupt is disabled. 1 <sub>B</sub> The data lost interrupt is enabled. In case of a data lost event, the service request output SRx indicated by INPRH.PINP is activated.
<b>TSIEN</b>	12	rw	<b>Transmit Shift Interrupt Enable</b> This bit enables the interrupt generation in case of a transmit shift event. 0 <sub>B</sub> The transmit shift interrupt is disabled. 1 <sub>B</sub> The transmit shift interrupt is enabled. In case of a transmit shift interrupt event, the service request output SRx indicated by INPRL.TSINP is activated.

**Universal Serial Interface Channel**

<b>Field</b>	<b>Bits</b>	<b>Type</b>	<b>Description</b>
<b>TBIEN</b>	13	rw	<b>Transmit Buffer Interrupt Enable</b> This bit enables the interrupt generation in case of a transmit buffer event. 0 <sub>B</sub> The transmit buffer interrupt is disabled. 1 <sub>B</sub> The transmit buffer interrupt is enabled. In case of a transmit buffer event, the service request output SRx indicated by INPRL.TBINP is activated.
<b>RIEN</b>	14	rw	<b>Receive Interrupt Enable</b> This bit enables the interrupt generation in case of a receive event. 0 <sub>B</sub> The receive interrupt is disabled. 1 <sub>B</sub> The receive interrupt is enabled. In case of a receive event, the service request output SRx indicated by INPRL.RINP is activated.
<b>AIEN</b>	15	rw	<b>Alternative Receive Interrupt Enable</b> This bit enables the interrupt generation in case of a alternative receive event. 0 <sub>B</sub> The alternative receive interrupt is disabled. 1 <sub>B</sub> The alternative receive interrupt is enabled. In case of an alternative receive event, the service request output SRx indicated by INPRL.AINP is activated.
<b>0</b>	[7:4]	r	<b>Reserved</b> Read as 0; should be written with 0.

### 21.2.3.2 Channel Configuration Register

The channel configuration register contains indicates the functionality that is available in the USIC channel.

#### CCFG

#### Channel Configuration Register

(18<sub>H</sub>)

Reset Value: 00CF<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0								TB	RB	0		IIS	IIC	ASC	SSC
r								r	r	r		r	r	r	r

Field	Bits	Type	Description
<b>SSC</b>	0	r	<b>SSC Protocol Available</b> This bit indicates if the SSC protocol is available. 0 <sub>B</sub> The SSC protocol is not available. 1 <sub>B</sub> The SSC protocol is available.
<b>ASC</b>	1	r	<b>ASC Protocol Available</b> This bit indicates if the ASC protocol is available. 0 <sub>B</sub> The ASC protocol is not available. 1 <sub>B</sub> The ASC protocol is available.
<b>IIC</b>	2	r	<b>IIC Protocol Available</b> This bit indicates if the IIC functionality is available. 0 <sub>B</sub> The IIC protocol is not available. 1 <sub>B</sub> The IIC protocol is available.
<b>IIS</b>	3	r	<b>IIS Protocol Available</b> This bit indicates if the IIS protocol is available. 0 <sub>B</sub> The IIS protocol is not available. 1 <sub>B</sub> The IIS protocol is available.
<b>RB</b>	6	r	<b>Receive FIFO Buffer Available</b> This bit indicates if an additional receive FIFO buffer is available. 0 <sub>B</sub> A receive FIFO buffer is not available. 1 <sub>B</sub> A receive FIFO buffer is available.
<b>TB</b>	7	r	<b>Transmit FIFO Buffer Available</b> This bit indicates if an additional transmit FIFO buffer is available. 0 <sub>B</sub> A transmit FIFO buffer is not available. 1 <sub>B</sub> A transmit FIFO buffer is available.

Field	Bits	Type	Description
<b>0</b>	[5:4], [15:8]	r	<b>Reserved</b> Read as 0; should be written with 0.

### 21.2.3.3 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<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BP COM	0	COMCFG		BP SUM	0	SUMCFG		BP NOM	0	NOMCFG		0	0	BP MOD EN	MOD EN
w	r	rw		w	r	rw		w	r	rw		r	r	w	rw

Field	Bits	Type	Description
<b>MODEN</b>	0	rw	<b>Module Enable</b> This bit enables the module kernel clock and the module functionality. 0 <sub>B</sub> The module is switched off immediately (without respecting a stop condition). It does not react on mode control actions and the module clock is switched off. The module does not react on read accesses and ignores write accesses (except to KSCFG). 1 <sub>B</sub> 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.
<b>BPMODEN</b>	1	w	<b>Bit Protection for MODEN</b> This bit enables the write access to the bit MODEN. It always reads 0. 0 <sub>B</sub> MODEN is not changed. 1 <sub>B</sub> MODEN is updated with the written value.

**Universal Serial Interface Channel**

<b>Field</b>	<b>Bits</b>	<b>Type</b>	<b>Description</b>
<b>NOMCFG</b>	[5:4]	rw	<b>Normal Operation Mode Configuration</b> This bit field defines the kernel mode applied in normal operation mode. 00 <sub>B</sub> Run mode 0 is selected. 01 <sub>B</sub> Run mode 1 is selected. 10 <sub>B</sub> Stop mode 0 is selected. 11 <sub>B</sub> Stop mode 1 is selected.
<b>BPNOM</b>	7	w	<b>Bit Protection for NOMCFG</b> This bit enables the write access to the bit field NOMCFG. It always reads 0. 0 <sub>B</sub> NOMCFG is not changed. 1 <sub>B</sub> NOMCFG is updated with the written value.
<b>SUMCFG</b>	[9:8]	rw	<b>Suspend Mode Configuration</b> This bit field defines the kernel mode applied in suspend mode. Coding like NOMCFG.
<b>BPSUM</b>	11	w	<b>Bit Protection for SUMCFG</b> This bit enables the write access to the bit field SUMCFG. It always reads 0. 0 <sub>B</sub> SUMCFG is not changed. 1 <sub>B</sub> SUMCFG is updated with the written value.
<b>COMCFG</b>	[13:12]	rw	<b>Clock Off Mode Configuration</b> This bit field defines the kernel mode applied in clock-off mode. Coding like NOMCFG.
<b>BPCOM</b>	15	w	<b>Bit Protection for COMCFG</b> This bit enables the write access to the bit field COMCFG. It always reads 0. 0 <sub>B</sub> COMCFG is not changed. 1 <sub>B</sub> COMCFG is updated with the written value.
<b>0</b>	[3:2], 6, 10, 14	r	<b>Reserved</b> Read as 0; should be written with 0.

### 21.2.3.4 Interrupt Node Pointer Registers

The interrupt node pointer registers define the service request output SR<sub>x</sub> that is activated if the corresponding event occurs and interrupt generation is enabled.

#### INPRL

**Interrupt Node Pointer Register L (14<sub>H</sub>)** **Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>0</b>	<b>AINP</b>		<b>0</b>	<b>RINP</b>		<b>0</b>	<b>TBINP</b>		<b>0</b>	<b>TSINP</b>					
r	rw		r	rw		r	rw		r	rw					

Field	Bits	Type	Description
<b>TSINP</b>	[1:0]	rw	<b>Transmit Shift Interrupt Node Pointer</b> This bit field defines which service request output SR <sub>x</sub> becomes activated in case of a transmit shift interrupt. 00 <sub>B</sub> Output SR0 becomes activated. 01 <sub>B</sub> Output SR1 becomes activated. 10 <sub>B</sub> Output SR2 becomes activated. 11 <sub>B</sub> Output SR3 becomes activated.
<b>TBINP</b>	[5:4]	rw	<b>Transmit Buffer Interrupt Node Pointer</b> This bit field defines which service request output SR <sub>x</sub> will be activated in case of a transmit buffer interrupt or a receive start interrupt. Coding like TSINP.
<b>RINP</b>	[9:8]	rw	<b>Receive Interrupt Node Pointer</b> This bit field defines which service request output SR <sub>x</sub> will be activated in case of a receive interrupt. Coding like TSINP.
<b>AINP</b>	[13:12]	rw	<b>Alternative Receive Interrupt Node Pointer</b> This bit field defines which service request output SR <sub>x</sub> will be activated in case of a alternative receive interrupt. Coding like TSINP.
<b>0</b>	[3:2], [7:6], [11:10], [15:14]	r	<b>Reserved</b> Read as 0; should be written with 0.



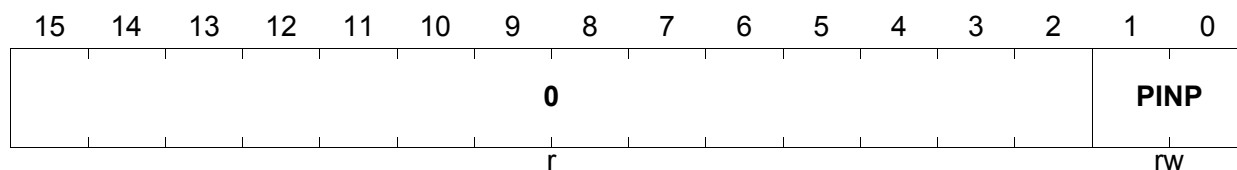
**Universal Serial Interface Channel**

**INPRH**

**Interrupt Node Pointer Register H**

**(16<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
<b>PINP</b>	[1:0]	rw	<b>Protocol Interrupt Node Pointer</b> This bit field defines which service request output SRx becomes activated in case of a protocol interrupt. 00 <sub>B</sub> Output SR0 becomes activated. 01 <sub>B</sub> Output SR1 becomes activated. 10 <sub>B</sub> Output SR2 becomes activated. 11 <sub>B</sub> Output SR3 becomes activated.
<b>0</b>	[15:2]	r	<b>Reserved</b> Read as 0; should be written with 0.

## 21.2.4 Protocol Related Registers

### 21.2.4.1 Protocol Control Registers

The bits in the protocol control registers 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.

#### PCRL

**Protocol Control Register L** (40<sub>H</sub>) **Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>CTR 15</b>	<b>CTR 14</b>	<b>CTR 13</b>	<b>CTR 12</b>	<b>CTR 11</b>	<b>CTR 10</b>	<b>CTR 9</b>	<b>CTR 8</b>	<b>CTR 7</b>	<b>CTR 6</b>	<b>CTR 5</b>	<b>CTR 4</b>	<b>CTR 3</b>	<b>CTR 2</b>	<b>CTR 1</b>	<b>CTR 0</b>
rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW

Field	Bits	Type	Description
<b>CTR<sub>x</sub></b> (x = 0-15)	x	rW	<b>Protocol Control Bit x</b> This bit is a protocol control bit.

#### PCRH

**Protocol Control Register H** (42<sub>H</sub>) **Reset Value: 0000<sub>H</sub>**

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

Field	Bits	Type	Description
<b>CTR<sub>x</sub></b> (x = 16-30)	x - 16	rwh	<b>Protocol Control Bit x</b> This bit is a protocol control bit that can be overwritten by protocol-specific information.
<b>CTR31</b>	15	rwh	<b>Protocol Control Bit 31</b> In the various protocols, this bit controls the start and the stop of the MCLK signal. 0 <sub>B</sub> Signal MCLK is not generated (MCLK = 0). 1 <sub>B</sub> Signal MCLK generation is enabled.

### 21.2.4.2 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 doesn't 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

(44<sub>H</sub>)

Reset Value: 0000<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>AIF</b>	<b>RIF</b>	<b>TBIF</b>	<b>TSIF</b>	<b>DLIF</b>	<b>RSIF</b>	<b>ST9</b>	<b>ST8</b>	<b>ST7</b>	<b>ST6</b>	<b>ST5</b>	<b>ST4</b>	<b>ST3</b>	<b>ST2</b>	<b>ST1</b>	<b>ST0</b>
rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
<b>STx</b> (x = 0-9)	x	rwh	<b>Protocol Status Flag x</b> See protocol specific description.
<b>RSIF</b>	10	rwh	<b>Receiver Start Indication Flag</b> 0 <sub>B</sub> A receiver start event has not occurred. 1 <sub>B</sub> A receiver start event has occurred.
<b>DLIF</b>	11	rwh	<b>Data Lost Indication Flag</b> 0 <sub>B</sub> A data lost event has not occurred. 1 <sub>B</sub> A data lost event has occurred.
<b>TSIF</b>	12	rwh	<b>Transmit Shift Indication Flag</b> 0 <sub>B</sub> A transmit shift event has not occurred. 1 <sub>B</sub> A transmit shift event has occurred.
<b>TBIF</b>	13	rwh	<b>Transmit Buffer Indication Flag</b> 0 <sub>B</sub> A transmit buffer event has not occurred. 1 <sub>B</sub> A transmit buffer event has occurred.
<b>RIF</b>	14	rwh	<b>Receive Indication Flag</b> 0 <sub>B</sub> A receive event has not occurred. 1 <sub>B</sub> A receive event has occurred.
<b>AIF</b>	15	rwh	<b>Alternative Receive Indication Flag</b> 0 <sub>B</sub> An alternative receive event has not occurred. 1 <sub>B</sub> An alternative receive event has occurred.

### 21.2.4.3 Protocol Status Clear Register

Read accesses to this register always deliver 0 at all bit positions.

#### PSCR

#### Protocol Status Clear Register

(48<sub>H</sub>)

Reset Value: 0000<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>C</b> <b>AIF</b>	<b>C</b> <b>RIF</b>	<b>C</b> <b>TBIF</b>	<b>C</b> <b>TSIF</b>	<b>C</b> <b>DLIF</b>	<b>C</b> <b>RSIF</b>	<b>C</b> <b>ST9</b>	<b>C</b> <b>ST8</b>	<b>C</b> <b>ST7</b>	<b>C</b> <b>ST6</b>	<b>C</b> <b>ST5</b>	<b>C</b> <b>ST4</b>	<b>C</b> <b>ST3</b>	<b>C</b> <b>ST2</b>	<b>C</b> <b>ST1</b>	<b>C</b> <b>ST0</b>
W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W

Field	Bits	Type	Description
<b>CSTx</b> (x = 0-9)	x	w	<b>Clear Status Flag x in PSR</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Flag PSR.STx is cleared.
<b>CRSIF</b>	10	w	<b>Clear Receiver Start Indication Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Flag PSR.RSIF is cleared.
<b>CDLIF</b>	11	w	<b>Clear Data Lost Indication Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Flag PSR.DLIF is cleared.
<b>CTSIF</b>	12	w	<b>Clear Transmit Shift Indication Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Flag PSR.TSIF is cleared.
<b>CTBIF</b>	13	w	<b>Clear Transmit Buffer Indication Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Flag PSR.TBIF is cleared.
<b>CRIF</b>	14	w	<b>Clear Receive Indication Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Flag PSR.RIF is cleared.
<b>CAIF</b>	15	w	<b>Clear Alternative Receive Indication Flag</b> 0 <sub>B</sub> No action 1 <sub>B</sub> Flag PSR.AIF is cleared.

## 21.2.5 Operating the Input Stages

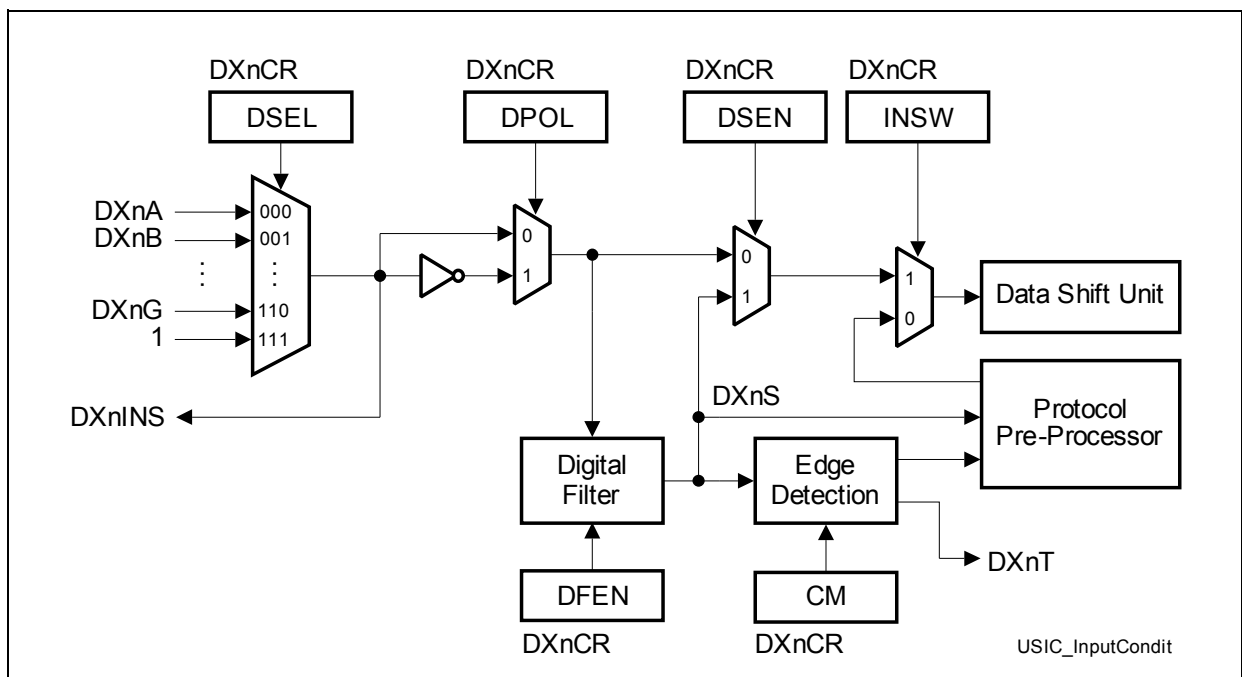
All three input stages offer the same feature set. They are used for all protocols, because the signal conditioning can be adapted in a very flexible way and the digital filters can be switched on and off separately.

### 21.2.5.1 General Input Structure

All input stages are built in a similar way as shown in [Figure 21-10](#). All enable/disable functions and selections are controlled independently for each input stage by bits in the registers DX0CR, DX1CR, and DX2CR.

The desired input signal can be selected among the input lines DXnA to DXnG and a permanent 1-level by programming bit field DSEL. Please refer to the implementation chapter 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  $f_{SYS}$ ) 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  $f_{SYS}$ .



**Figure 21-10 Input Conditioning**

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.

#### **21.2.5.2 Digital Filter**

The digital filter can be enabled to reduce noise on the input signals. Before being filtered, the input signal becomes synchronized to  $f_{SYS}$ . 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  $f_{SYS}$ . 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  $f_{SYS}$ , an enabled filter adds a delay of up to two filter sampling periods between the selected input and signal  $DXnS$ .

#### **21.2.5.3 Edge Detection**

The synchronized (and optionally filtered) signal  $DXnS$  can be used as input to the data shift unit and is also an input to the selected protocol pre-processor. If the protocol pre-processor does not use the  $DXnS$  signal for protocol-specific handling,  $DXnS$  can be used for other tasks, e.g. to control data transmissions in master mode (a data word can be tagged valid for transmission, see chapter about data buffering).

A programmable edge detection indicates that the desired event has occurred by activating the trigger signal  $DXnT$  (introducing a delay of one period of  $f_{SYS}$  before a reaction to this event can take place).

#### **21.2.5.4 Selected Input Monitoring**

The selected input signal of each input stage has been made available with the signals  $DX0INS$ ,  $DX1INS$ , and  $DX2INS$ . These signals can be used in the system to trigger other actions, e.g. to generate interrupts.

#### **21.2.5.5 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

## Universal Serial Interface Channel

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.

### 21.2.6 Input Stage Register

#### 21.2.6.1 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 (20<sub>H</sub>) Reset Value: 0000<sub>H</sub>

##### DX1CR

Input Control Register 1 (24<sub>H</sub>) Reset Value: 0000<sub>H</sub>

##### DX2CR

Input Control Register 2 (28<sub>H</sub>) Reset Value: 0000<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DXS	0			CM		SF SEL	D POL	0	DS EN	DF EN	IN SW	0	DSEL		
rh	r			rw		rw	rw	r	rw	rw	rw	r	rw		

Field	Bits	Type	Description
DSEL	[2:0]	rw	<b>Data Selection for Input Signal</b> This bit field defines the input data signal for the corresponding input line for protocol pre-processor. The selection can be made from the input vector DXn[G:A]. 000 <sub>B</sub> The data input DXnA is selected. 001 <sub>B</sub> The data input DXnB is selected. 010 <sub>B</sub> The data input DXnC is selected. 011 <sub>B</sub> The data input DXnD is selected. 100 <sub>B</sub> The data input DXnE is selected. 101 <sub>B</sub> The data input DXnF is selected. 110 <sub>B</sub> The data input DXnG is selected. 111 <sub>B</sub> The data input is always 1.

**Universal Serial Interface Channel**

Field	Bits	Type	Description
<b>INSW</b>	4	rw	<b>Input Switch</b> This bit defines if the data shift unit input is derived from the input data path DXn or from the selected protocol pre-processors. $0_B$ The input of the data shift unit is controlled by the protocol pre-processor. $1_B$ The input of the data shift unit is connected to the selected data input line. This setting is used if the signals are directly derived from an input pin without treatment by the protocol pre-processor.
<b>DFEN</b>	5	rw	<b>Digital Filter Enable</b> This bit enables/disables the digital filter for signal DXnS. $0_B$ The input signal is not digitally filtered. $1_B$ The input signal is digitally filtered.
<b>DSEN</b>	6	rw	<b>Data Synchronization Enable</b> This bit selects if the asynchronous input signal or the synchronized (and optionally filtered) signal DXnS can be used as input for the data shift unit. $0_B$ The un-synchronized signal can be taken as input for the data shift unit. $1_B$ The synchronized signal can be taken as input for the data shift unit.
<b>DPOL</b>	8	rw	<b>Data Polarity for DXn</b> This bit defines the signal polarity of the input signal. $0_B$ The input signal is not inverted. $1_B$ The input signal is inverted.
<b>SFSEL</b>	9	rw	<b>Sampling Frequency Selection</b> This bit defines the sampling frequency of the digital filter for the synchronized signal DXnS. $0_B$ The sampling frequency is $f_{SYS}$ . $1_B$ The sampling frequency is $f_{FD}$ .



**Universal Serial Interface Channel**

Field	Bits	Type	Description
<b>CM</b>	[11:10]	rw	<b>Combination Mode</b> This bit field selects which edge of the synchronized (and optionally filtered) signal DXnS activates the trigger output DXnT of the input stage. 00 <sub>B</sub> The trigger activation is disabled. 01 <sub>B</sub> A rising edge activates DXnT. 10 <sub>B</sub> A falling edge activates DXnT. 11 <sub>B</sub> Both edges activate DXnT.
<b>DXS</b>	15	rh	<b>Synchronized Data Value</b> This bit indicates the value of the synchronized (and optionally filtered) input signal. 0 <sub>B</sub> The current value of DXnS is 0. 1 <sub>B</sub> The current value of DXnS is 1.
<b>0</b>	3, 7, [14:12]	r	<b>Reserved</b> Read as 0; should be written with 0.

## 21.2.7 Operating the Baud Rate Generator

The following blocks can be configured to operate the baud rate generator, see also **Figure 21-2**.

### 21.2.7.1 Fractional Divider

The fractional divider generates its output frequency  $f_{FD}$  by dividing the input frequency  $f_{SYS}$  either by an integer factor  $n$  or by multiplication by  $n/1024$ . It has two operating modes:

- Normal divider mode (FDRL.DM = 01<sub>B</sub>):  
 In this mode, the output frequency  $f_{FD}$  is derived from the input clock  $f_{SYS}$  by an integer division by a value between 1 and 1024. The division is based on a counter FDRH.RESULT that is incremented by 1 with  $f_{SYS}$ . After reaching the value 3FF<sub>H</sub>, the counter is loaded with FDRL.STEP and then continues counting. In order to achieve  $f_{FD} = f_{SYS}$ , the value of STEP has to be programmed with 3FF<sub>H</sub>.  
 The output frequency in normal divider mode is defined by the equation:

$$f_{FD} = f_{SYS} \times \frac{1}{n} \quad \text{with } n = 1024 - \text{STEP} \quad (21.1)$$

- Fractional divider mode (FDRL.DM = 10<sub>B</sub>):  
 In this mode, the output frequency  $f_{FD}$  is derived from the input clock  $f_{SYS}$  by a fractional multiplication by  $n/1024$  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  $f_{SYS}$  period. This jitter is not accumulated over several cycles.  
 The frequency  $f_{FD}$  is generated by an addition of FDRL.STEP to FDRH.RESULT with  $f_{SYS}$ . The frequency  $f_{FD}$  is based on the overflow of the addition result over 3FF<sub>H</sub>.  
 The output frequency in fractional divider mode is defined by the equation:

$$f_{FD} = f_{SYS} \times \frac{n}{1024} \quad \text{with } n = \text{STEP} \quad (21.2)$$

The output frequency  $f_{FD}$  of the fractional divider is selected for baud rate generation by BRGL.CLKSEL = 00<sub>B</sub>.

### 21.2.7.2 External Frequency Input

The baud rate can be generated referring to an external frequency input (instead of to  $f_{SYS}$ ) 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  $f_{SYS}$ . It can be optionally filtered

## Universal Serial Interface Channel

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 BRGL.CLKSEL = 10<sub>B</sub>, 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 BRGL.CLKSEL = 11<sub>B</sub>, 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  $f_{SYS}$  to be used for baud rate generation.

### 21.2.7.3 Protocol-Related Counter in Divider Mode

In divider mode, the protocol-related 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  $f_{SYS}$  can also appear in these signals. The outputs frequencies of this divider is controlled by registers BRGL and BRGH.

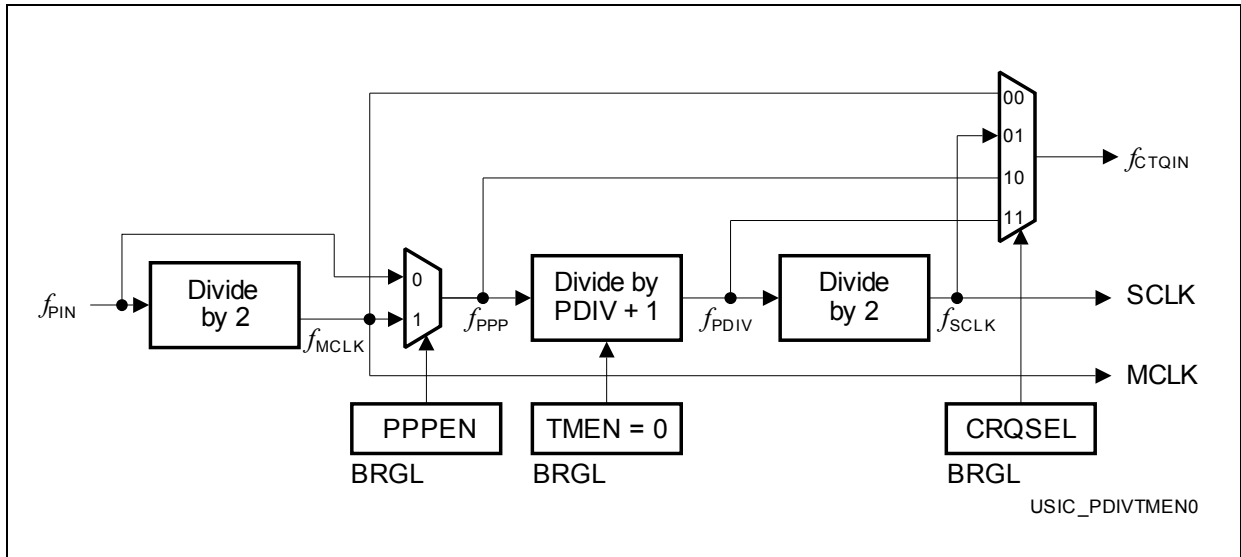
In order to define a frequency ratio between the master clock MCLK and the shift clock SCLK, the divider stage for MCLK is located in front of the divider by PDIV+1, whereas the divider stage for SCLK is located at the output of this divider.

$$f_{MCLK} = \frac{f_{PIN}}{2} \quad (21.3)$$

$$f_{SCLK} = \frac{f_{PDIV}}{2} \quad (21.4)$$

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.

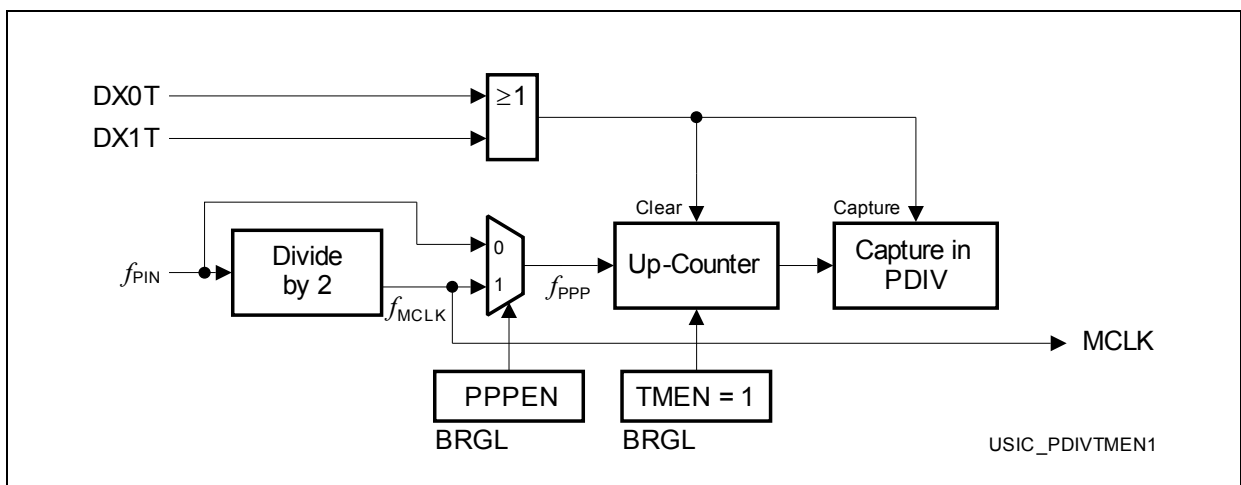
$$\begin{aligned} f_{PDIV} &= f_{PIN} \times \frac{1}{PDIV + 1} && \text{if } PPPEN = 0 \\ f_{PDIV} &= f_{MCLK} \times \frac{1}{PDIV + 1} && \text{if } PPPEN = 1 \end{aligned} \quad (21.5)$$



### Figure 21-11 Protocol-Related Counter (Divider Mode)

#### 21.2.7.4 Protocol-Related Counter in Capture Mode

In capture mode, the protocol-related counter stage can be used for time interval measurement (BRGL.TMEN = 1). In this case, the frequency division is disabled (reception and transmission are not possible) and the counter is working as capture timer by counting  $f_{\text{PPP}}$  periods. When reaching its maximum value, the counter stops counting. If an event is indicated by DX0T or DX1T, the actual counter value is captured into bit field BRGH.PDIV and the counter restarts from 0. Additionally, a transmit shift interrupt event is generated (bit PSRL.TSIF becomes set).



### Figure 21-12 Protocol-Related Counter (Capture Mode)

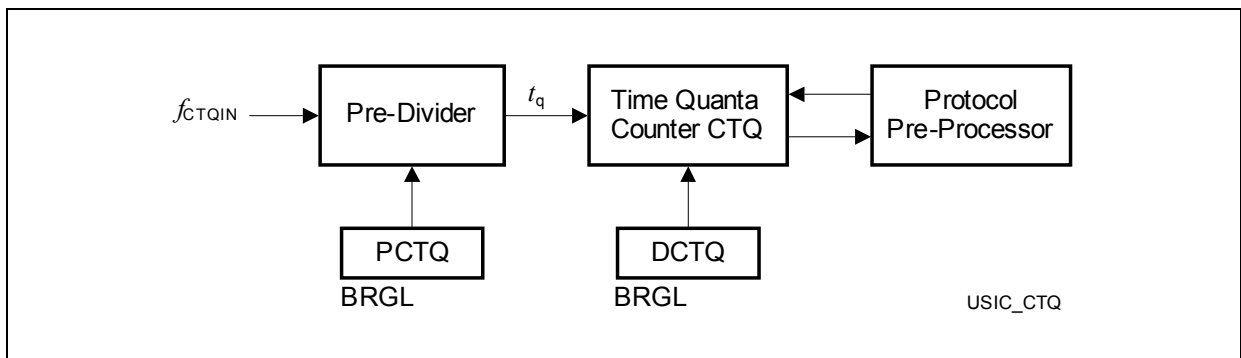
The capture mode can be used to measure the baud rate in slave mode before starting data transfers, e.g. to measure the time between two edges of a data signal (by DX0T)

or of a shift clock signal (by DX1T). The conditions to activate the DXnT trigger signals can be configured in each input stage.

### 21.2.7.5 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.



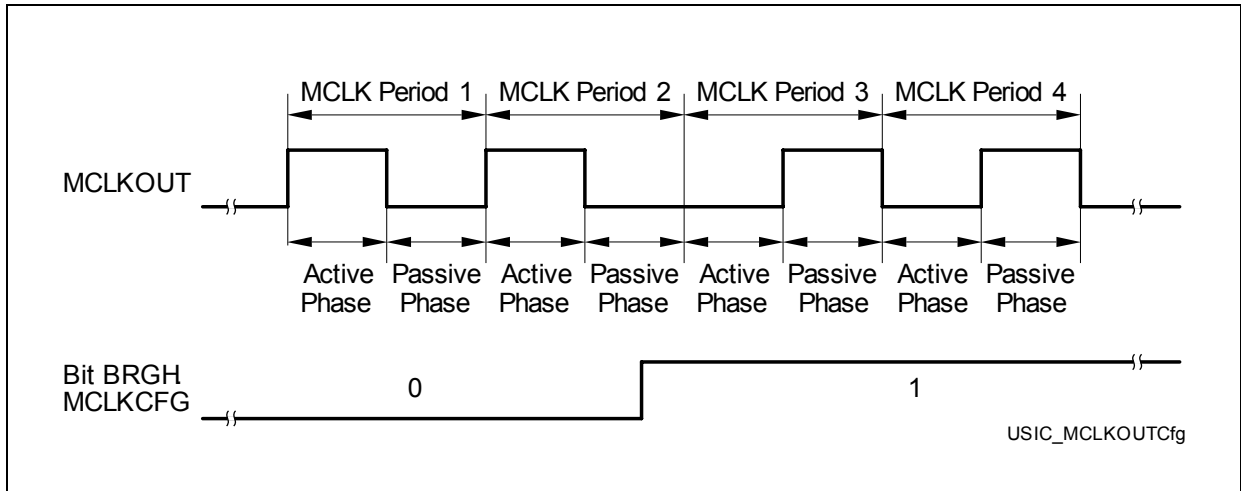
**Figure 21-13 Time Quanta Counter**

### 21.2.7.6 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 BRGH.MCLKCFG, evaluated at the start of the active phase. The polarity of the MCLKOUT signal during the passive phase is defined by bit BRGH.MCLKCFG, evaluated at the start of the passive phase. If bit BRGH.MCLKOUT 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 21-14](#), the value of BRGH.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 PCRH.MCLK. After this bit has become set, signal MCLKOUT is generated with the next active phase of the MCLK period. If PCRH.MCLK = 0 (MCLKOUT generation disabled), the level for the passive phase is also applied for active phase.



**Figure 21-14 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 BRGH.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.

## 21.2.8 Baud Rate Generator Registers

### 21.2.8.1 Fractional Divider Registers

The fractional divider registers FDRL and FDRH allow the generation of the internal frequency  $f_{FD}$ , that is derived from the system clock  $f_{SYS}$ .

#### FDRL

**Fractional Divider Register L**

**(04<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>DM</b>		<b>0</b>				<b>STEP</b>									
rw		r				rw									

Field	Bits	Type	Description
<b>STEP</b>	[9:0]	rw	<b>Step Value</b> In normal divider mode STEP contains the reload value for RESULT after RESULT has reached 3FF <sub>H</sub> . In fractional divider mode STEP defines the value added to RESULT with each input clock cycle.
<b>DM</b>	[15:14]	rw	<b>Divider Mode</b> This bit fields defines the functionality of the fractional divider block. 00 <sub>B</sub> The divider is switched off, $f_{FD} = 0$ . 01 <sub>B</sub> Normal divider mode selected. 10 <sub>B</sub> Fractional divider mode selected. 11 <sub>B</sub> The divider is switched off, $f_{FD} = 0$ .
<b>0</b>	[13:10]	r	<b>Reserved</b> Read as 0; should be written with 0.

**Universal Serial Interface Channel**

**FDRH**

**Fractional Divider Register H**

**(06<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>0</b>		<b>0</b>				<b>RESULT</b>									
rw		r				rh									

Field	Bits	Type	Description
<b>RESULT</b>	[9:0]	rh	<b>Result Value</b> In normal divider mode this bit field is updated with $f_{SYS}$ according to: $RESULT = RESULT + 1$ In fractional divider mode this bit field is updated with $f_{SYS}$ according to: $RESULT = RESULT + STEP$ If bit field DM is written with 01 <sub>B</sub> or 10 <sub>B</sub> , RESULT is loaded with a start value of 3FF <sub>H</sub> .
<b>0</b>	[15:14]	rw	<b>Reserved for Future Use</b> Must be written with 0 to allow correct fractional divider operation.
<b>0</b>	[13:10]	r	<b>Reserved</b> Read as 0; should be written with 0.



### 21.2.8.2 Baud Rate Generator Registers

The protocol-related divider for baud rate generation is controlled by the registers BRGL and BRGH.

#### BRGL

**Baud Rate Generator Register L (1C<sub>H</sub>)** **Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	DCTQ				PCTQ		CTQSEL		0	PP EN	TM EN	0	CLKSEL		
r	rw				rw		rw		r	rw	rw	r	rw		

Field	Bits	Type	Description
<b>CLKSEL</b>	[1:0]	rw	<b>Clock Selection</b> This bit field defines the input frequency $f_{PIN}$ 00 <sub>B</sub> The fractional divider frequency $f_{FD}$ is selected. 01 <sub>B</sub> Reserved, no action 10 <sub>B</sub> The trigger signal DX1T defines $f_{PIN}$ . Signal MCLK toggles with $f_{PIN}$ . 11 <sub>B</sub> Signal MCLK corresponds to the DX1S signal and the frequency $f_{PIN}$ is derived from the rising edges of DX1S.
<b>TMEN</b>	3	rw	<b>Timing Measurement Enable</b> This bit defines the functionality of the protocol-related divider. 0 <sub>B</sub> Divider mode: $f_{PDIV} = f_{PPP} / (PDIV + 1)$ Data transfers are possible and the trigger signals DX0T and DX1T are ignored. 1 <sub>B</sub> Capture mode: 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 PDIV, the counter is cleared and a transmit shift event is generated. Data transfers are not possible.

**Universal Serial Interface Channel**

Field	Bits	Type	Description
<b>PPPEN</b>	4	rw	<b>Enable 2:1 Divider for <math>f_{PPP}</math></b> This bit defines the input frequency $f_{PPP}$ . $0_B$ The 2:1 divider for $f_{PPP}$ is disabled. $f_{PPP} = f_{PIN}$ $1_B$ The 2:1 divider for $f_{PPP}$ is enabled. $f_{PPP} = f_{MCLK} = f_{PIN} / 2$ .
<b>CTQSEL</b>	[7:6]	rw	<b>Input Selection for CTQ</b> This bit defines the length of a time quantum for the protocol pre-processor. $00_B$ $f_{CTQIN} = f_{PDIV}$ $01_B$ $f_{CTQIN} = f_{PPP}$ $10_B$ $f_{CTQIN} = f_{SCLK}$ $11_B$ $f_{CTQIN} = f_{MCLK}$
<b>PCTQ</b>	[9:8]	rw	<b>Pre-Divider for Time Quanta Counter</b> This bit field defines length of a time quantum $t_q$ for the time quanta counter in the protocol pre-processor. $t_Q = (PCTQ + 1) / f_{CTQIN}$
<b>DCTQ</b>	[14:10]	rw	<b>Denominator for Time Quanta Counter</b> This bit field defines the number of time quanta $t_q$ taken into account by the time quanta counter in the protocol pre-processor.
<b>0</b>	2, 5, 15	r	<b>Reserved</b> Read as 0; should be written with 0.

**Universal Serial Interface Channel**

**BRGH**

**Baud Rate Generator Register H**

**(1E<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>SCLKCFG</b>		<b>M CLK CFG</b>	<b>0</b>			<b>PDIV</b>									
rw		rw	r			rwh									

Field	Bits	Type	Description
<b>PDIV</b>	[9:0]	rwh	<b>Divider Mode: Divider Factor to Generate <math>f_{PDIV}</math></b> This bit field defines the ratio between the input frequency $f_{PPP}$ and the divider frequency $f_{PDIV}$ . <b>Capture Mode: Captured Time Interval</b> 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.
<b>MCLKCFG</b>	13	rw	<b>Master Clock Configuration</b> This bit field defines the level of the passive phase of the MCLKOUT signal. 0 <sub>B</sub> The passive level is 0. 1 <sub>B</sub> The passive level is 1.
<b>SCLKCFG</b>	[15:14]	rw	<b>Shift Clock Output Configuration</b> This bit field defines the level of the passive phase of the SCLKOUT signal and enables/disables a delay of half of a SCLK period. 00 <sub>B</sub> The passive level is 0 and the delay is disabled. 01 <sub>B</sub> The passive level is 1 and the delay is disabled. 10 <sub>B</sub> The passive level is 0 and the delay is enabled. 11 <sub>B</sub> The passive level is 1 and the delay is enabled.
<b>0</b>	[12:10]	r	<b>Reserved</b> Read as 0; should be written with 0.

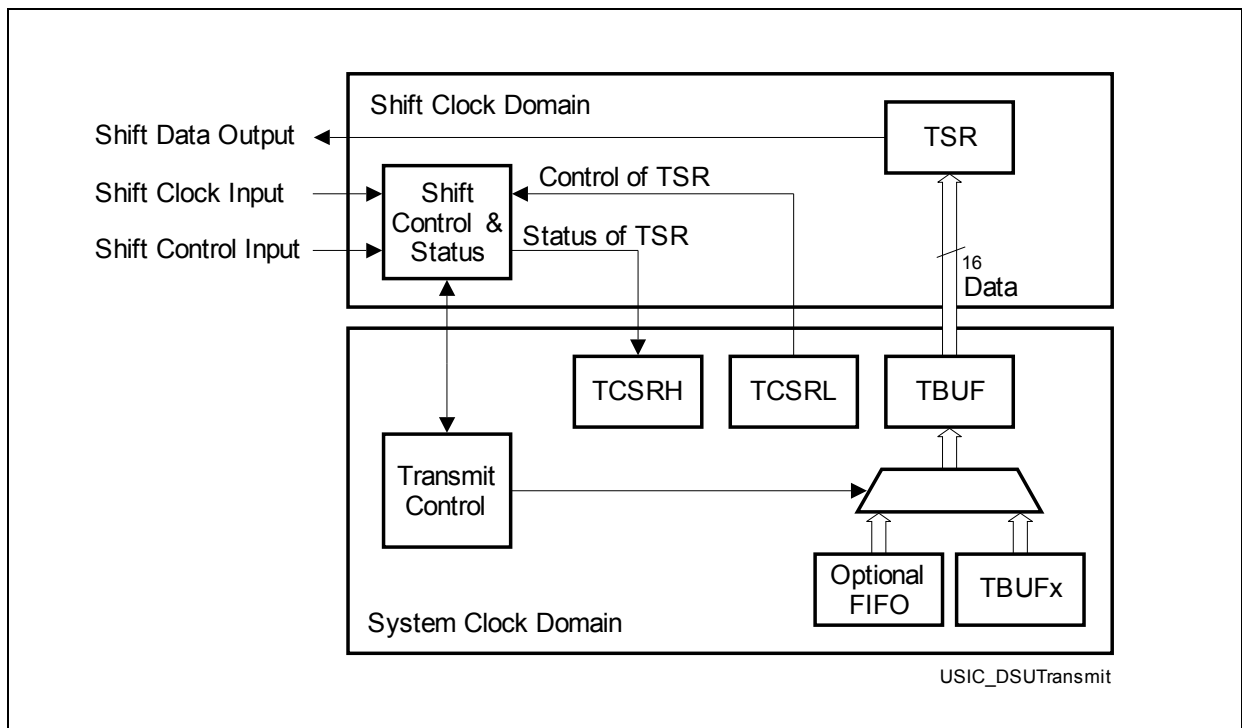
## 21.2.9 Operating the Transmit Data Path

The transmit data path is based on a 16-bit wide transmit shift register TSR 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 registers. Register TCSRL mainly controls the transmit data handling, whereas register TCSRH monitors the transmit status.

A change of the value of the data shift output signal DOUT 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 DOUT until the next data word begins with the next corresponding edge of the shift clock.

### 21.2.9.1 Transmit Buffering

The transmit shift register TSR can not be directly accessed by software, because it is 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 21-52](#)) or, optionally, by a FIFO buffer stage (see [Page 21-79](#)).



**Figure 21-15 Transmit Data Path**

### **21.2.9.2 Transmit Control Information**

The transmit control information TCI can be used as additional control parameter for data transfers. The TCI is derived from the address x of the written TBUFx transmit buffer input location.

It can be used 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 bits WLEMD, FLEMD, SELMD, and WAMD in register TCSRL. Please note that not all possible settings lead to useful system behavior.

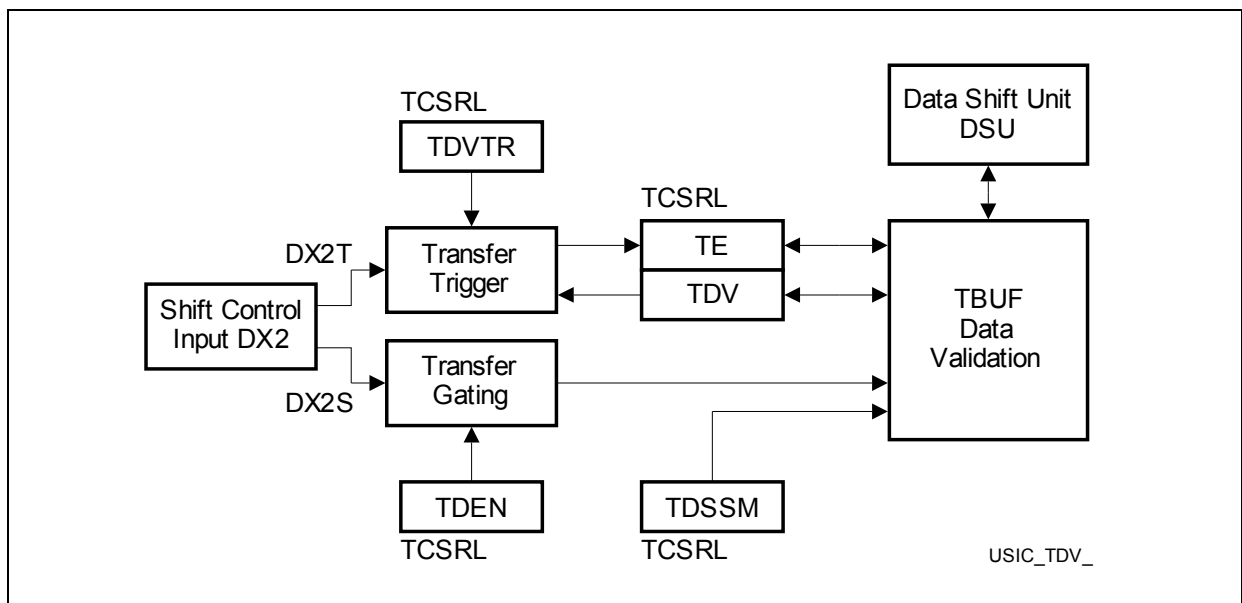
- **Word length control:**  
If TCSRL.WLEMD = 1, bit field SCTRH.WLE is updated with TCI[3:0] if a transmit buffer input location TBUFx is written. This function can be used in all protocols to dynamically change the data word length between 1 and 16 data bits per data word. Additionally, bit TCSRL.EOF 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 TCSRL.FLEMD = TCSRL.SELMD = 0.
- **Frame length control:**  
If TCSRL.FLEMD = 1, bit field SCTRH.FLE[4:0] is updated with TCI[4:0] and SCTRH.FLE[5] becomes 0 if a transmit buffer input location TBUFx is written. This function can be used in all protocols to dynamically change the data frame length between 1 and 32 data bits per data frame. It is recommended to program TCSRL.SELMD = TCSRL.WLEMD = TCSRL.WAMD = 0.
- **Select output control:**  
If TCSRL.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 TCSRL.WLEMD = TCSRL.FLEMD = TCSRL.WAMD = 0.
- **Word address control:**  
If TCSRL.WAMD = 1, bit TCSRL.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 TCSRL.SELMD = TCSRL.FLEMD = 0.

### 21.2.9.3 Transmit Data Validation

The data word in the transmit buffer TBUF can be tagged valid or invalid for transmission by bit TCSRL.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 SCTRL.PDL (if the content of TBUF has not been valid at the transmission start). In both cases, the content of TBUF is not changed.

The control and status bits for the data validation are located in registers TCSRL or TCSRH. The data validation is based on the logic blocks shown in **Figure 21-16**.



**Figure 21-16 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

## Universal Serial Interface Channel

shifting, signal DX2S can be used for gating purposes. The transfer gating logic is controlled by bit field TCSRL.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 TCSRL.TDVTR and the occurrence of a trigger event is indicated by bit TCSRH.TE.
- 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 TCSRL.TDV = 1, and bit TCSRH.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 TCSRL.TDV by writing FMRL.MTDV = 10<sub>B</sub> before updating the data. Bit TCSRL.TDV becomes automatically set when TBUF is updated with new data. Another possibility are the interrupts TBI (for ASC and IIC) or RSI (for SSC and IIS) indicating that a transmission has started. While a transmission is in progress, TBUF can be loaded with new data. In this case the user has to take care that an update of the TBUF content takes place before a new transmission starts.

With this structure, the following data transfer functionality can be achieved:

- If bit TCSRL.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 TCSRL.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 TCSRL.TDSSM has to be 1.
- Bit TCSRL.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 FMRL.MTDV allows software driven modification (set or clear) of bit TCSRL.TDV. Together with the gating control bit field TCSRL.TDEN, the user can set up the transmit data word without starting the transmission. A possible program

## Universal Serial Interface Channel

sequence could be: clear TCSRL.TDEN = 00<sub>B</sub>, write data to TBUFx, clear TCSRL.TDV by writing FMRL.MTDV = 10<sub>B</sub>, re-enable the gating with TCSRL.TDEN = 01<sub>B</sub> and then set TCSRL.TDV under software control by writing FMRL.MTDV = 01<sub>B</sub>.

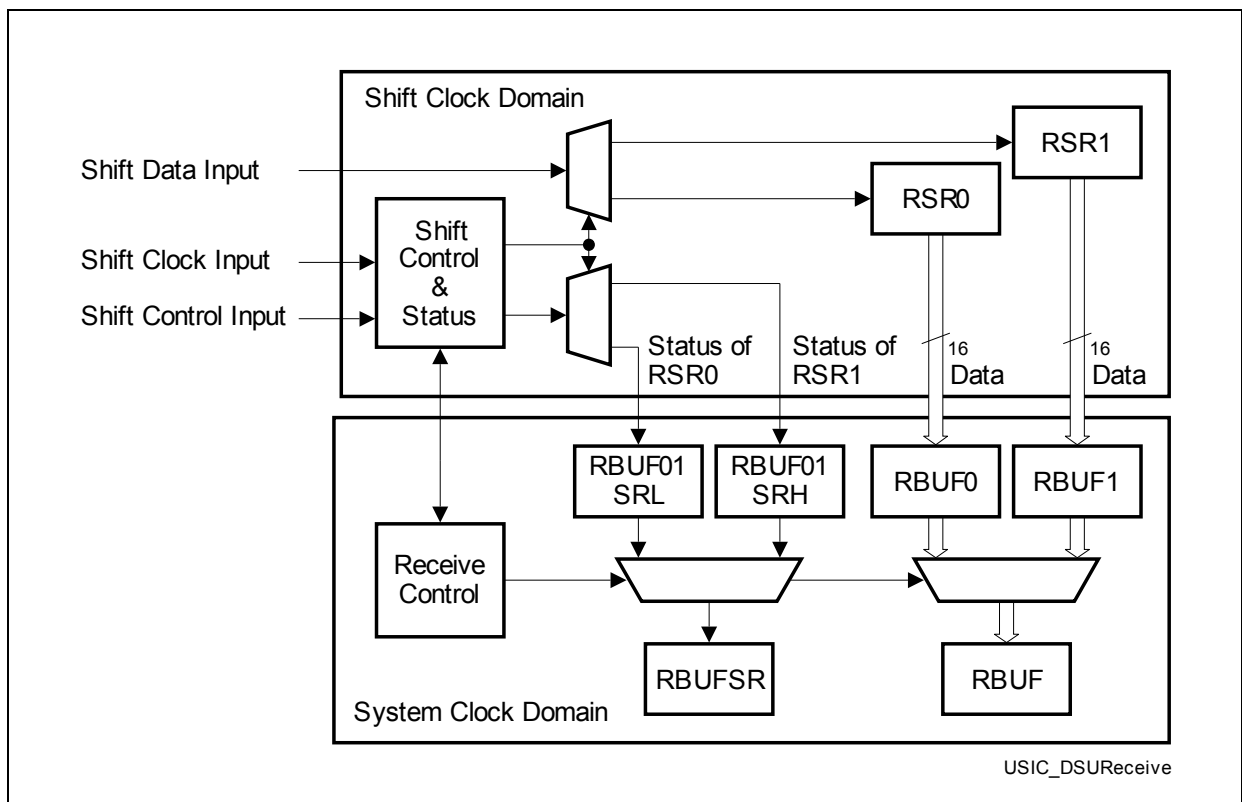
### 21.2.10 Operating the Receive Data Path

The receive data path is based on two 16-bit wide receive shift registers RSR0 and RSR1 and a receive buffer for each of them (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 RBUF01SRL monitors the status of RBUF0 and register RBUF01SRH of RBUF1.

#### 21.2.10.1 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 21-79](#)).



**Figure 21-17 Receive Data Path**



### **21.2.10.2 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  $f_{\text{SYS}}$ .

- A received data word in a receiver shift register RSRx must be held constant for at least 4 periods of  $f_{\text{SYS}}$  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  $f_{\text{SYS}}$  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  $f_{\text{SYS}}$  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. The setup and hold times are defined in the Data Sheet.

## 21.2.11 Transfer Control and Status Registers

### 21.2.11.1 Shift Control Registers

The data shift unit is controlled by the registers defined in this section. The values in these registers are applied for data transmission and reception.

Please note that the shift control settings SDIR, WLE, and FLE 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).

#### **SCTRL**

##### **Shift Control Register L**

**(30<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0						TRM		DOCFG		0			PDL		S DIR
r						rw		rw		r			rw		rw

Field	Bits	Type	Description
<b>SDIR</b>	0	rw	<b>Shift Direction</b> This bit defines the shift direction of the data words for transmission and reception. 0 <sub>B</sub> Shift LSB first. The first data bit of a data word is located at bit position 0. 1 <sub>B</sub> Shift MSB first. The first data bit of a data word is located at the bit position given by bit field SCTRLH.WLE.
<b>PDL</b>	1	rw	<b>Passive Data Level</b> This bit defines the output level at the shift data output signal when no data is available for transmission. The PDL level is output with the first relevant transmit shift clock edge of a data word. 0 <sub>B</sub> The passive data level is 0. 1 <sub>B</sub> The passive data level is 1.
<b>DOCFG</b>	[7:6]	rw	<b>Data Output Configuration</b> This bit defines the relation between the internal shift data value and the data output signal DOUT. X0 <sub>B</sub> DOUT = shift data value X1 <sub>B</sub> DOUT = inverted shift data value

**Universal Serial Interface Channel**

Field	Bits	Type	Description
<b>TRM</b>	[9:8]	rw	<b>Transmission Mode</b> This bit field describes how the shift control signal is interpreted by the DSU. Data transfers are only possible while the shift control signal is active. 00 <sub>B</sub> The shift control signal is considered as inactive and data frame transfers are not possible. 01 <sub>B</sub> The shift control signal is considered active if it is at 1-level. This is the setting to be programmed to allow data transfers. 10 <sub>B</sub> The shift control signal is considered active if it is at 0-level. It is recommended to avoid this setting and to use the inversion in the DX2 stage in case of a low-active signal. 11 <sub>B</sub> 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.
<b>0</b>	[5:2], [15:10]	r	<b>Reserved</b> Read as 0; should be written with 0.

**Universal Serial Interface Channel**

**SCTRH**

**Shift Control Register H**

**(32<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>0</b>				<b>WLE</b>				<b>0</b>		<b>FLE</b>					
r				rwh				r		rwh					

Field	Bits	Type	Description
<b>FLE</b>	[5:0]	rwh	<b>Frame Length</b> This bit field defines how many bits are transferred within a data frame. A data frame can consist of several concatenated data words. If TCSRL.FLEMD = 1, the value can be updated automatically by the data handler.
<b>WLE</b>	[11:8]	rwh	<b>Word Length</b> This bit field defines the data word length (amount of bits that are transferred in each data word) for reception and transmission. The data word is always right-aligned in the data buffer at the bit positions [WLE down to 0]. If TCSRL.WLEMD = 1, the value can be updated automatically by the data handler. 0 <sub>H</sub> The data word contains 1 data bit located at bit position 0. 1 <sub>H</sub> The data word contains 2 data bits located at bit positions [1:0]. ... E <sub>H</sub> The data word contains 15 data bits located at bit positions [14:0]. F <sub>H</sub> The data word contains 16 data bits located at bit positions [15:0].
<b>0</b>	[7:6], [15:12]	r	<b>Reserved</b> Read as 0; should be written with 0.

## 21.2.11.2 Transmission Control and Status Registers

The data transmission is controlled by register TCSRL.

### TCSRL

**Transmit Control/Status Register L (3C<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>0</b>	<b>WA</b>	<b>TD VTR</b>	<b>TDEN</b>			<b>0</b>	<b>TD SSM</b>	<b>TDV</b>	<b>EOF</b>	<b>SOF</b>	<b>0</b>	<b>WA MD</b>	<b>FLE MD</b>	<b>SEL MD</b>	<b>WLE MD</b>
r	rwh	rw	rw			r	rw	rh	rwh	rwh	r	rw	rw	rw	rw

Field	Bits	Type	Description
<b>WLEMD</b>	0	rw	<b>WLE Mode</b> This bit enables the data handler to automatically update the bit field SCTRH.WLE by the transmit control information TCI[3:0] and bit TCSR.EOF by TCI[4] (see <a href="#">Page 21-52</a> ). If enabled, an automatic update takes place when new data is loaded to register TBUF, either by writing to one of the transmit buffer input locations TBUFx or by an optional data buffer. 0 <sub>B</sub> The automatic update of SCTRH.WLE and TCSR.EOF is disabled. 1 <sub>B</sub> The automatic update of SCTR.WLE and TCSR.EOF is enabled.
<b>SELMD</b>	1	rw	<b>Select Mode</b> This bit can be used mainly for the SSC protocol. It enables the data handler to automatically update bit field PCRH.CTR[20:16] by the transmit control information TCI[4:0] and clear bit field PCRH.CTR[23:21] (see <a href="#">Page 21-52</a> ). If enabled, an automatic update takes place when new data is loaded to register TBUF, either by writing to one of the transmit buffer input locations TBUFx or by an optional data buffer. 0 <sub>B</sub> The automatic update of PCRH.CTR[23:16] is disabled. 1 <sub>B</sub> The automatic update of PCRH.CTR[23:16] is disabled.

**Universal Serial Interface Channel**

Field	Bits	Type	Description
<b>FLEMD</b>	2	rw	<b>FLE Mode</b> This bit enables the data handler to automatically update bits SCTR.H.FLE[4:0] by the transmit control information TCI[4:0] and to clear bit SCTR.H.FLE[5] (see <a href="#">Page 21-52</a> ). If enabled, an automatic update takes place when new data is loaded to register TBUF, either by writing to one of the transmit buffer input locations TBUFx or by an optional data buffer. 0 <sub>B</sub> The automatic update of FLE is disabled. 1 <sub>B</sub> The automatic update of FLE is enabled.
<b>WAMD</b>	3	rw	<b>WA Mode</b> This bit can be used mainly for the IIS protocol. It enables the data handler to automatically update bit TCSR.L.WA by the transmit control information TCI[4] (see <a href="#">Page 21-52</a> ). If enabled, an automatic update takes place when new data is loaded to register TBUF, either by writing to one of the transmit buffer input locations TBUFx or by an optional data buffer. 0 <sub>B</sub> The automatic update of bit WA is disabled. 1 <sub>B</sub> The automatic update of bit WA is enabled.
<b>SOF</b>	5	rw	<b>Start Of Frame</b> This bit is only taken into account for the SSC protocol, otherwise it is ignored. It indicates that the data word in TBUF is considered as the first word of a new SSC frame if it is valid for transmission (TCSR.L.TDV = 1). This bit becomes cleared when the TBUF data word is transferred to the transmit shift register. 0 <sub>B</sub> The data word in TBUF is not considered as first word of a frame. 1 <sub>B</sub> 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).

**Universal Serial Interface Channel**

Field	Bits	Type	Description
<b>EOF</b>	6	rwh	<p><b>End Of Frame</b></p> <p>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.</p> <p>0<sub>B</sub> The data word in TBUF is not considered as last word of an SSC frame.</p> <p>1<sub>B</sub> The data word in TBUF is considered as last word of an SSC frame.</p>
<b>TDV</b>	7	rh	<p><b>Transmit Data Valid</b></p> <p>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).</p> <p>0<sub>B</sub> The data word in TBUF is not valid for transmission.</p> <p>1<sub>B</sub> 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.</p>

**Universal Serial Interface Channel**

Field	Bits	Type	Description
<b>TDSSM</b>	8	rw	<p><b>TBUF Data Single Shot Mode</b></p> <p>This bit defines if the data word TBUF data is considered as permanently valid or if the data should only be transferred once.</p> <p>0<sub>B</sub> 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.</p> <p>1<sub>B</sub> 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.</p> <p>TDSSM = 1 has to be programmed if an optional data buffer is used.</p>
<b>TDEN</b>	[11:10]	rw	<p><b>TBUF Data Enable</b></p> <p>This bit field controls the gating of the transmission start of the data word in the transmit buffer TBUF.</p> <p>00<sub>B</sub> A transmission start of the data word in TBUF is disabled. If a transmission is started, the passive data level is sent out.</p> <p>01<sub>B</sub> A transmission of the data word in TBUF can be started if TDV = 1.</p> <p>10<sub>B</sub> A transmission of the data word in TBUF can be started if TDV = 1 while DX2S = 0.</p> <p>11<sub>B</sub> A transmission of the data word in TBUF can be started if TDV = 1 while DX2S = 1.</p>
<b>TDVTR</b>	12	rw	<p><b>TBUF Data Valid Trigger</b></p> <p>This bit enables the transfer trigger unit to set bit TCSRH.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.</p> <p>0<sub>B</sub> Bit TCSRH.TE is permanently set.</p> <p>1<sub>B</sub> Bit TCSRH.TE is set if DX2T becomes active while TDV = 1.</p>



**Universal Serial Interface Channel**

Field	Bits	Type	Description
<b>WA</b>	13	rwh	<b>Word Address</b> This bit is only taken into account for the IIS protocol, otherwise it is ignored. It can be modified automatically by the data handler if bit WAMD = 1. Bit WA defines for which channel the data stored in TBUF will be transmitted. $0_B$ The data word in TBUF will be transmitted after a falling edge of WA has been detected (referring to PSR.WA). $1_B$ The data word in TBUF will be transmitted after a rising edge of WA has been detected (referring to PSR.WA).
<b>0</b>	4, 9, [15:14]	r	<b>Reserved</b> Read as 0; should be written with 0.

**Universal Serial Interface Channel**

The data transmission status is monitored by register TCSRH.

**TCSRH**

**Transmit Control/Status Register H (3E<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		TE	TVC	TV	0	T SOF	0								
r		rh	rh	rh	r	rh	r								

Field	Bits	Type	Description
<b>TSOF</b>	8	rh	<b>Transmitted Start Of Frame</b> This bit indicates if the latest start of a data word transmission has taken place for the first data word of a new data frame. This bit is updated with the transmission start of each data word. 0 <sub>B</sub> The latest data word transmission has not been started for the first word of a data frame. 1 <sub>B</sub> The latest data word transmission has been started for the first word of a data frame.
<b>TV</b>	10	rh	<b>Transmission Valid</b> This bit represents the transmit buffer underflow and indicates if the latest start of a data word transmission has taken place with a valid data word from the transmit buffer TBUF. This bit is updated with the transmission start of each data word. 0 <sub>B</sub> 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 (SCTRL.PDL) has been started. 1 <sub>B</sub> The latest start of a data word transmission has taken place with valid data from TBUF.
<b>TVC</b>	11	rh	<b>Transmission Valid Cumulated</b> This bit cumulates the transmit buffer underflow indication TV. It is cleared automatically together with bit TV and has to be set by writing FMRL.ATVC = 1. 0 <sub>B</sub> Since TVC has been set, at least one data buffer underflow condition has occurred. 1 <sub>B</sub> Since TVC has been set, no data buffer underflow condition has occurred.

**Universal Serial Interface Channel**

Field	Bits	Type	Description
<b>TE</b>	12	rh	<b>Trigger Event</b> If the transfer trigger mechanism is enabled, this bit indicates that a trigger event has been detected (DX2T = 1) while TCSRL.TDV = 1. If the event trigger mechanism is disabled, the bit TE is permanently set. It is cleared by writing FMRL.MTDV = 10 <sub>B</sub> or when the data word located in TBUF is loaded into the shift register. 0 <sub>B</sub> The trigger event has not yet been detected. A transmission of the data word in TBUF can not be started. 1 <sub>B</sub> The trigger event has been detected (or the trigger mechanism is switched off) and a transmission of the data word in TBUF can not be started.
<b>0</b>	[7:0], 9, [15:13]	r	<b>Reserved</b> Read as 0; should be written with 0.

### 21.2.11.3 Flag Modification Registers

The flag modification registers FMRL, FMRH allow the modification of control and status flags related to data handling by using only write accesses. Read accesses to FMRL, FMRH 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).

#### FMRL

**Flag Modification Register L** (38<sub>H</sub>) **Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>C</b>	<b>C</b>										<b>A</b>				
<b>RDV</b>	<b>RDV</b>					<b>0</b>					<b>TVC</b>	<b>0</b>		<b>MTDV</b>	
<b>1</b>	<b>0</b>														
w	w					r					w	r		w	

Field	Bits	Type	Description
<b>MTDV</b>	[1:0]	w	<b>Modify Transmit Data Valid</b> Writing to this bit field can modify bits TCSR.L.TDV and TCSR.H.TE to control the start of a data word transmission by software. 00 <sub>B</sub> No action. 01 <sub>B</sub> Bit TDV is set, TE is unchanged. 10 <sub>B</sub> Bits TDV and TE are cleared. 11 <sub>B</sub> Reserved
<b>ATVC</b>	4	w	<b>Activate Bit TVC</b> Writing to this bit can set bit TCSR.H.TVC to start a new cumulation of the transmit buffer underflow condition. 0 <sub>B</sub> No action. 1 <sub>B</sub> Bit TCSR.H.TVC is set.
<b>CRDV0</b>	14	w	<b>Clear Bits RDV for RBUF0</b> Writing 1 to this bit clears bits RBUF0.SRL.RDV00 and RBUF0.SRH.RDV10 to declare the received data in RBUF0 as no longer valid (to emulate a read action). 0 <sub>B</sub> No action. 1 <sub>B</sub> Bits RBUF0.SRL.RDV00 and RBUF0.SRH.RDV10 are cleared.

**Universal Serial Interface Channel**

Field	Bits	Type	Description
<b>CRDV1</b>	15	w	<b>Clear Bit RDV for RBUF1</b> Writing 1 to this bit clears bits RBUF01SRL.RDV01 and RBUF01SRH.RDV11 to declare the received data in RBUF1 as no longer valid (to emulate a read action). 0 <sub>B</sub> No action. 1 <sub>B</sub> Bits RBUF01SRL.RDV01 and RBUF01SRH.RDV11 are cleared.
<b>0</b>	[3:2], [13:5]	r	<b>Reserved</b> Read as 0; should be written with 0.

**FMRH**

**Flag Modification Register H**

**(3A<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0												SI O3	SI O2	SI O1	SI O0
r												w	w	w	w

Field	Bits	Type	Description
<b>SIO0, SIO1, SIO2, SIO3</b>	0, 1, 2, 3	w	<b>Set Interrupt Output SRx</b> Writing a 1 to this bit field activates the service request output SRx of this USIC channel. It has no impact on service request outputs of other USIC channels. 0 <sub>B</sub> No action. 1 <sub>B</sub> The service request output SRx is activated.
<b>0</b>	[15:4]	r	<b>Reserved</b> Read as 0; should be written with 0.

## 21.2.12 Data Buffer Registers

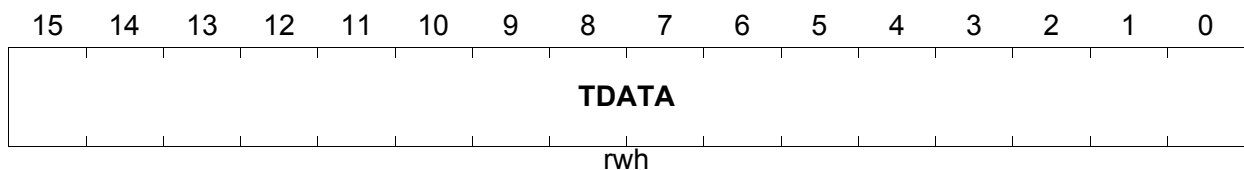
### 21.2.12.1 Transmit Buffer Locations

The 32 independent data input locations TBUF00 to TBUF31 are address locations that can be used as data entry locations for the transmit buffer. Data written to one of these locations will appear in a common register TBUF. Additionally, the 5 bit coding of the number [31:0] of the addressed data input location represents the transmit control information TCI (please refer to the protocol sections for more details).

The internal transmit buffer register TBUF contains the data that will be loaded to the transmit shift register for the next transmission of a data word. It can be read out at all TBUF00 to TBUF31 addresses.

#### TBUFx (x = 00-31)

**Transmit Buffer Input Location x**     **(80<sub>H</sub> + x\*4)**     **Reset Value: 0000<sub>H</sub>**



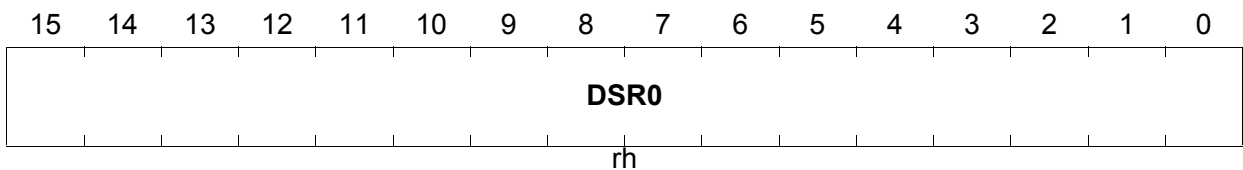
Field	Bits	Type	Description
TDATA	[15:0]	rwh	<b>Transmit Data</b> This bit field contains the data to be transmitted (read view). A data write action to at least the low byte of TDATA sets TCSRL.TDV.

### 21.2.12.2 Receive Buffer Registers RBUF0, RBUF1

The receive buffer register RBUF0 contains the data received from RSR0. 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** (50<sub>H</sub>) **Reset Value: 0000<sub>H</sub>**

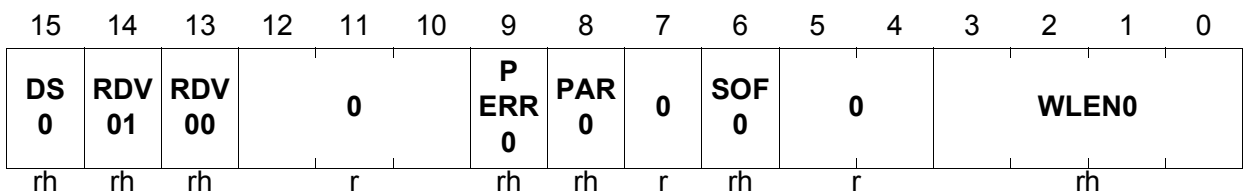


Field	Bits	Type	Description
DSR0	[15:0]	rh	Data of Shift Register 0

The receive buffer status register RBUF01SRL provides the status of the data in receive buffer RBUF0.

#### RBUF01SRL

**Receiver Buffer 01 Status Register L** (60<sub>H</sub>) **Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
WLEN0	[3:0]	rh	<b>Received Data Word Length in RBUF0</b> 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. 0 <sub>H</sub> One bit has been received. ... F <sub>H</sub> Sixteen bits have been received.

**Universal Serial Interface Channel**

<b>Field</b>	<b>Bits</b>	<b>Type</b>	<b>Description</b>
<b>SOF0</b>	6	rh	<b>Start of Frame in RBUF0</b> This bit indicates whether the data word in RBUF0 has been the first data word of a data frame. 0 <sub>B</sub> The data in RBUF0 has not been the first data word of a data frame. 1 <sub>B</sub> The data in RBUF0 has been the first data word of a data frame.
<b>PAR0</b>	8	rh	<b>Protocol-Related Argument in RBUF0</b> This bit indicates the value of the protocol-related argument. This value is elaborated depending on the selected protocol and adds additional information to the data word in RBUF0. The meaning of this bit is described in the corresponding protocol chapter.
<b>PERR0</b>	9	rh	<b>Protocol-related Error in RBUF0</b> This bit indicates if the value of the protocol-related argument meets an expected value. This value is elaborated depending on the selected protocol and adds additional information to the data word in RBUF0. The meaning of this bit is described in the corresponding protocol chapter. 0 <sub>B</sub> The received protocol-related argument PAR matches the expected value. The reception of the data word sets bit PSR.RIF and can generate a receive interrupt. 1 <sub>B</sub> 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.



**Universal Serial Interface Channel**

Field	Bits	Type	Description
<b>RDV00</b>	13	rh	<b>Receive Data Valid in RBUF0</b> This bit indicates the status of the data content of register RBUF0. This bit is identical to bit RBUF01SRH.RDV10 and allows consisting reading of information for the receive buffer registers. It is set when a new data word is stored in RBUF0 and automatically cleared if it is read out via RBUF. 0 <sub>B</sub> Register RBUF0 does not contain data that has not yet been read out. 1 <sub>B</sub> Register RBUF0 contains data that has not yet been read out.
<b>RDV01</b>	14	rh	<b>Receive Data Valid in RBUF1</b> This bit indicates the status of the data content of register RBUF1. This bit is identical to bit RBUF01SRH.RDV11 and allows consisting reading of information for the receive buffer registers. It is set when a new data word is stored in RBUF1 and automatically cleared if it is read out via RBUF. 0 <sub>B</sub> Register RBUF1 does not contain data that has not yet been read out. 1 <sub>B</sub> Register RBUF1 contains data that has not yet been read out.
<b>DS0</b>	15	rh	<b>Data Source</b> 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 RBUF01SRH.DS1 and allows consisting reading of information for the receive buffer registers. 0 <sub>B</sub> The register RBUF contains the data of RBUF0 (same for associated status information). 1 <sub>B</sub> The register RBUF contains the data of RBUF1 (same for associated status information).
<b>0</b>	[5:4], 7, [12:10]	r	<b>Reserved</b> Read as 0; should be written with 0.

**Universal Serial Interface Channel**

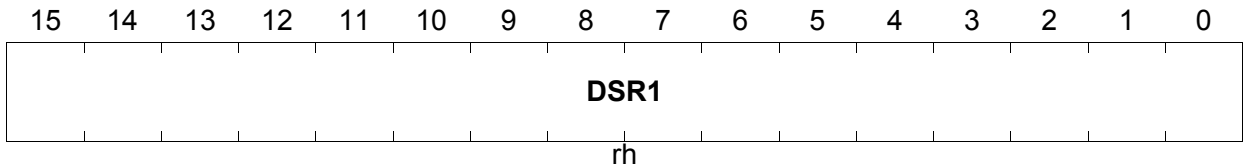
The receive buffer register RBUF1 contains the data received from RSR1. A read action does not change the status of the receive data from “not yet read = valid” to “already read = not valid”.

**RBUF1**

**Receiver Buffer Register 1**

**(54<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



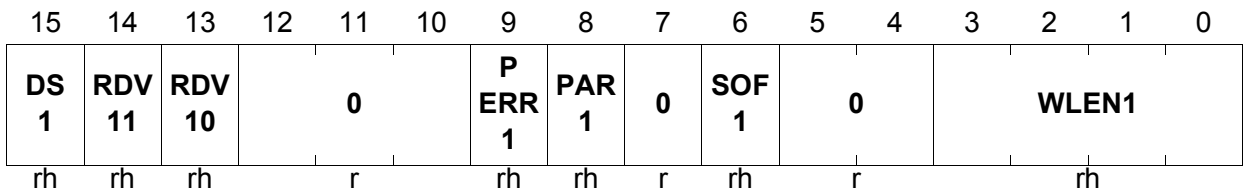
Field	Bits	Type	Description
DSR1	[15:0]	rh	Data of Shift Register 1

The receive buffer status register RBUF01SRH provides the status of the data in receive buffer RBUF1.

**RBUF01SRH**

**Receiver Buffer 01 Status Register H (62<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
WLEN1	[3:0]	rh	<b>Received Data Word Length in RBUF1</b> 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. 0 <sub>H</sub> One bit has been received. ... F <sub>H</sub> Sixteen bits have been received.

**Universal Serial Interface Channel**

Field	Bits	Type	Description
<b>SOF1</b>	6	rh	<b>Start of Frame in RBUF1</b> This bit indicates whether the data word in RBUF1 has been the first data word of a data frame. 0 <sub>B</sub> The data in RBUF1 has not been the first data word of a data frame. 1 <sub>B</sub> The data in RBUF1 has been the first data word of a data frame.
<b>PAR1</b>	8	rh	<b>Protocol-Related Argument in RBUF1</b> This bit indicates the value of the protocol-related argument. This value is elaborated depending on the selected protocol and adds additional information to the data word in RBUF1. The meaning of this bit is described in the corresponding protocol chapter.
<b>PERR1</b>	9	rh	<b>Protocol-related Error in RBUF1</b> This bit indicates if the value of the protocol-related argument meets an expected value. This value is elaborated depending on the selected protocol and adds additional information to the data word in RBUF1. The meaning of this bit is described in the corresponding protocol chapter. 0 <sub>B</sub> The received protocol-related argument PAR matches the expected value. The reception of the data word sets bit PSR.RIF and can generate a receive interrupt. 1 <sub>B</sub> 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.
<b>RDV10</b>	13	rh	<b>Receive Data Valid in RBUF0</b> This bit indicates the status of the data content of register RBUF0. This bit is identical to bit RBUF01SRL.RDV00 and allows consisting reading of information for the receive buffer registers. 0 <sub>B</sub> Register RBUF0 does not contain data that has not yet been read out. 1 <sub>B</sub> Register RBUF0 contains data that has not yet been read out.

**Universal Serial Interface Channel**

Field	Bits	Type	Description
<b>RDV11</b>	14	rh	<b>Receive Data Valid in RBUF1</b> This bit indicates the status of the data content of register RBUF1. This bit is identical to bit RBUF01SRL.RDV01 and allows consisting reading of information for the receive buffer registers. 0 <sub>B</sub> Register RBUF1 does not contain data that has not yet been read out. 1 <sub>B</sub> Register RBUF1 contains data that has not yet been read out.
<b>DS1</b>	15	rh	<b>Data Source</b> This bit indicates which receive buffer register (RBUF0 or RBUF1) is currently visible in registers RBUF(D) and in RBUF SR 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 RBUF01SRL.DS0 and allows consisting reading of information for the receive buffer registers. 0 <sub>B</sub> The register RBUF contains the data of RBUF0 (same for associated status information). 1 <sub>B</sub> The register RBUF contains the data of RBUF1 (same for associated status information).
<b>0</b>	[5:4], 7, [12:10]	r	<b>Reserved</b> Read as 0; should be written with 0.

### 21.2.12.3 Receive Buffer Registers RBUF, RBUFD, RBUFSR

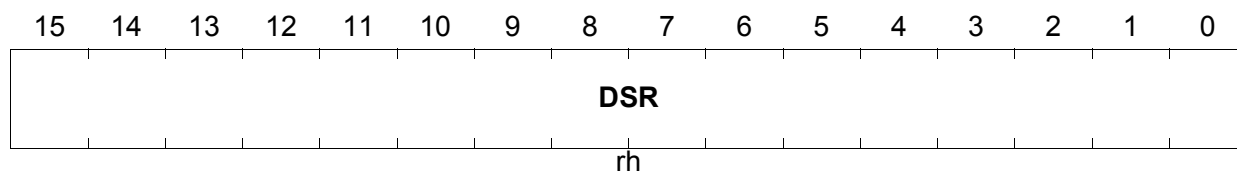
The receiver buffer register RBUF shows the content of the either RBUF0 or RBUF1, depending on the order of reception. Always the oldest data (the data word that has been received first) from both receive buffers can be read from RBUF. It is recommended to read out the received data from RBUF instead 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**

**(5C<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
DSR	[15:0]	rh	<b>Received Data</b> This bit field monitors the content of either RBUF0 or RBUF1, depending on the reception sequence.

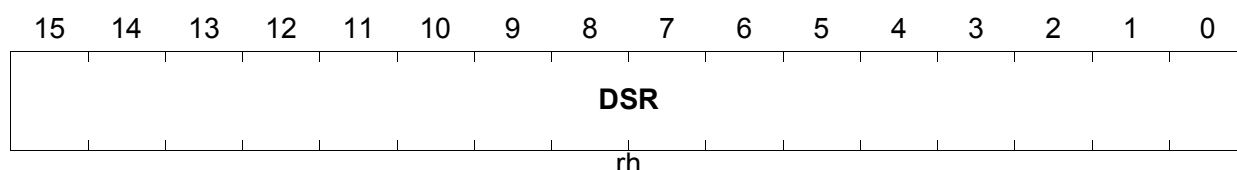
### Universal Serial Interface Channel

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 (4C<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
DSR	[15:0]	rh	<b>Data from Shift Register</b> Same as RBUF.DSR, but without releasing the buffer after a read action.

**Universal Serial Interface Channel**

The receive buffer status register RBUFSR provides the status of the data in receive buffers RBUF and RBUFD. If bits RBUF01SRL.DS0 (or RBUF01SRH.DS1) are 0, the content of RBUF01SRL is monitored in RBUFSR, otherwise the content of RBUF01SRH is shown.

**RBUFSR**

**Receiver Buffer Status Register**

**(58<sub>H</sub>)**

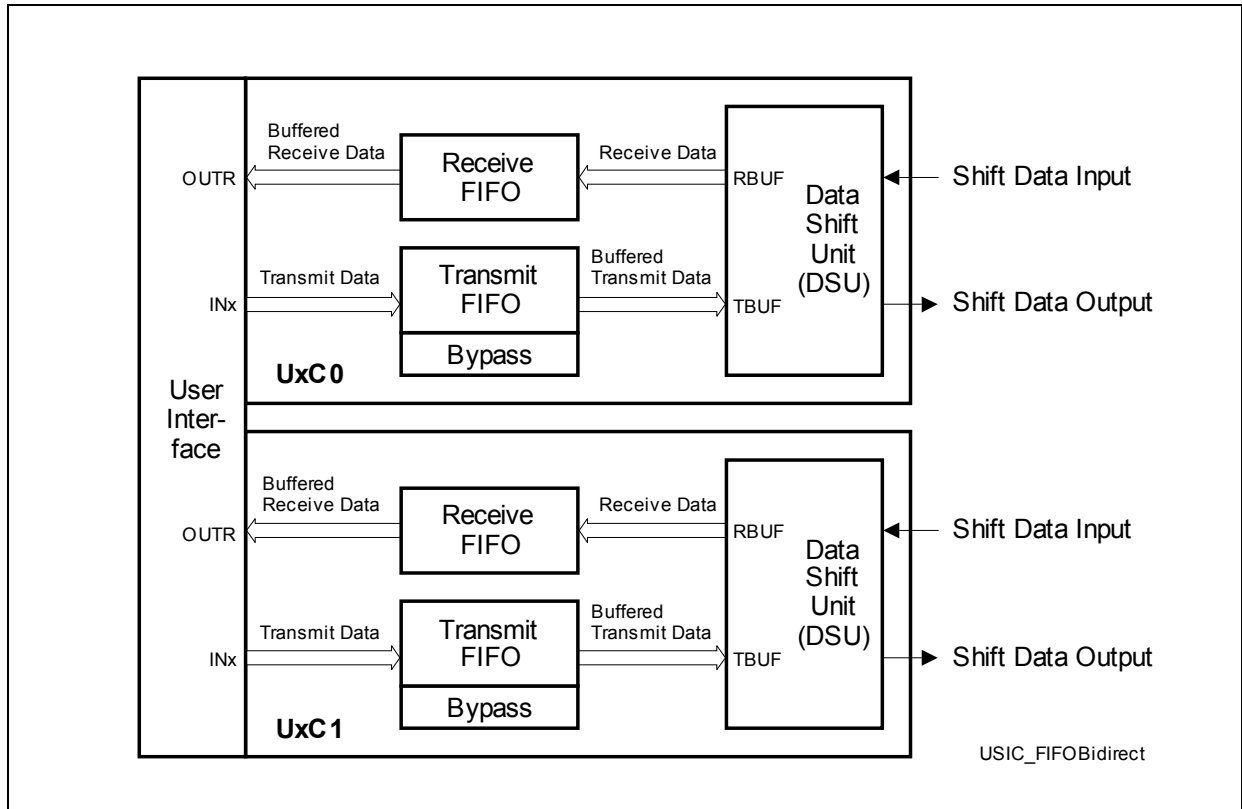
**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>DS</b>	<b>RDV</b> <b>1</b>	<b>RDV</b> <b>0</b>		<b>0</b>		<b>P</b> <b>ERR</b>	<b>PAR</b>	<b>0</b>	<b>SOF</b>	<b>0</b>			<b>WLEN</b>		
rh	rh	rh		r		rh	rh	r	rh	r			rh		

Field	Bits	Type	Description
<b>WLEN</b>	[3:0]	rh	<b>Received Data Word Length in RBUF or RBUFD</b> Description see RBUF01SRL.WLEN0 or RBUF01SRH.WLEN1.
<b>SOF</b>	6	rh	<b>Start of Frame in RBUF or RBUFD</b> Description see RBUF01SRL.SOF0 or RBUF01SRH.SOF1.
<b>PAR</b>	8	rh	<b>Protocol-Related Argument in RBUF or RBUFD</b> Description see RBUF01SRL.PAR0 or RBUF01SRH.PAR1.
<b>PERR</b>	9	rh	<b>Protocol-related Error in RBUF or RBUFD</b> Description see RBUF01SRL.PERR0 or RBUF01SRH.PERR1.
<b>RDV0</b>	13	rh	<b>Receive Data Valid in RBUF or RBUFD</b> Description see RBUF01SRL.RDV00 or RBUF01SRH.RDV10.
<b>RDV1</b>	14	rh	<b>Receive Data Valid in RBUF or RBUFD</b> Description see RBUF01SRL.RDV01 or RBUF01SRH.RDV11.
<b>DS</b>	15	rh	<b>Data Source of RBUF or RBUFD</b> Description see RBUF01SRL.DS0 or RBUF01SRH.DS1.
<b>0</b>	[5:4], 7, [12:10]	r	<b>Reserved</b> Read as 0; should be written with 0.

### 21.2.13 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 XE16xyM, totally 64 buffer entries can be distributed among the transmit or receive FIFO buffers of both channels of the USIC module.



**Figure 21-18 FIFO Buffer Overview**

In order to operate the FIFO data buffers, the following issues have to be considered:

- FIFO buffer available and selected:  
 The transmit FIFO buffer and the bypass structure are only available if CCFG.TB = 1, whereas the receive FIFO buffer is only available if CCFG.RB = 1.  
 It is recommended to configure all buffer parameters while there is no data traffic for this USIC channel and the FIFO mechanism is disabled by TBCTRH.SIZE = 0 (for transmit buffer) or RBCTRH.SIZE = 0 (for receive buffer). The allocation of a buffer area by writing TBCTRL or RBCTRL 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 21-86](#).

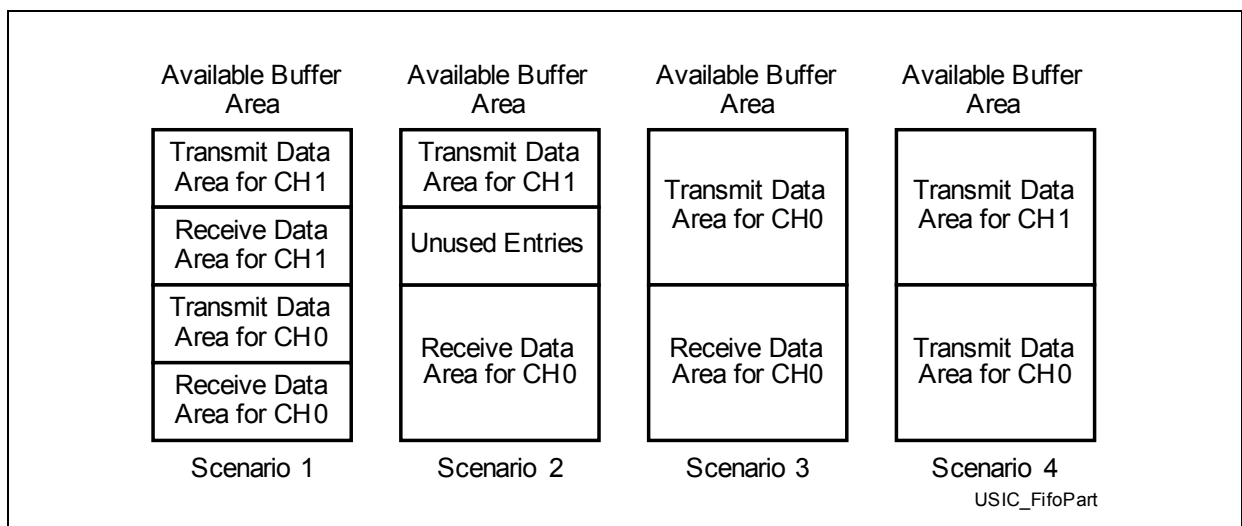
### 21.2.13.1 FIFO Buffer Partitioning

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 21-19](#).

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,  $n$ ,  $2 \times n$ ,  $3 \times n$ , etc. and consists of the FIFO entries  $[x \times n, (x+1) \times n - 1]$ , with  $x$  being an integer number (incl. 0). It is not possible to have “holes” with unused FIFO entries within a FIFO buffer, whereas there can be unused FIFO entries between two FIFO buffers.



**Figure 21-19 FIFO Buffer Partitioning**

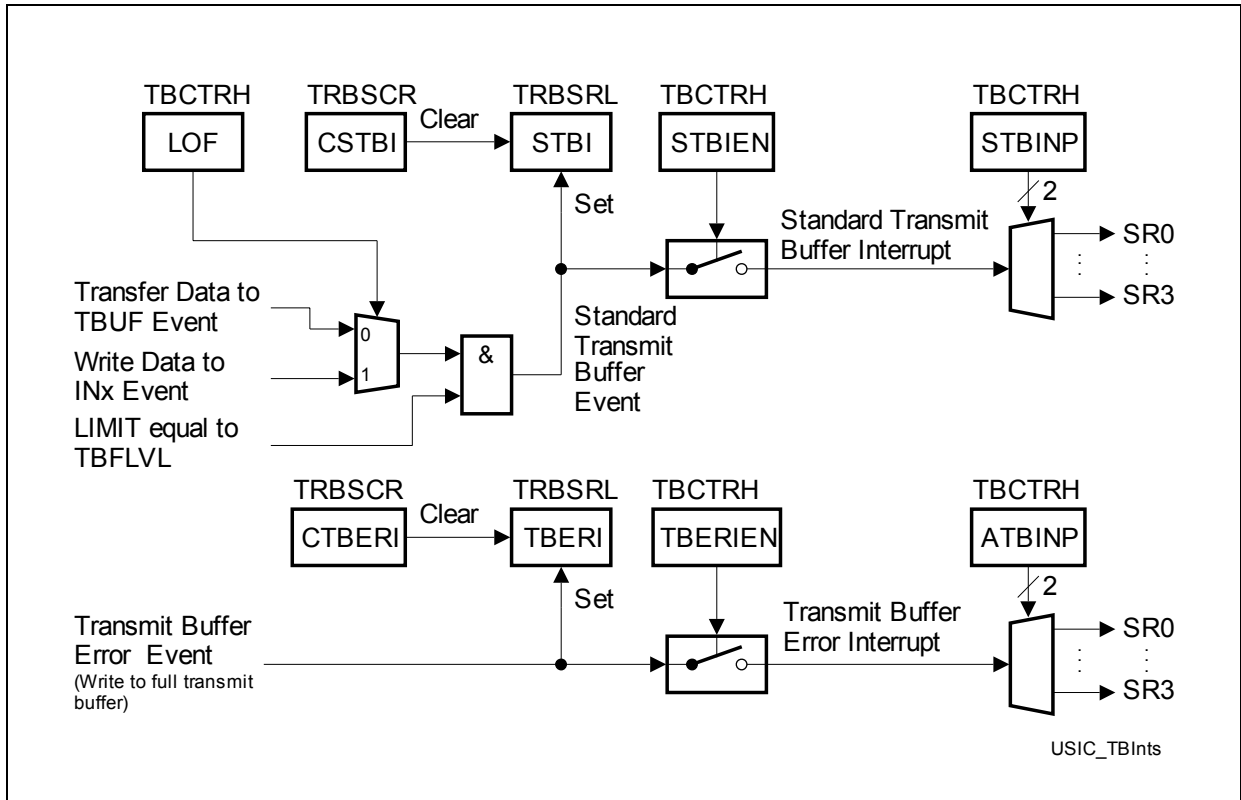
## **Universal Serial Interface Channel**

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 RBCTRL (for a receive FIFO buffer) or TBCTRL (for a transmit FIFO buffer) while the related bit field RBCTRH.SIZE=0 (or TBCTRH.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.

### **21.2.13.2 Data Buffer Events and Interrupts**

The transmit FIFO buffer mechanism detects the following events, that can lead to interrupts (if enabled).

- Standard transmit buffer event:  
The filling level of the transmit buffer (given by TRBSRH.TBFLVL) exceeds (TBCTRH.LOF = 1) or falls below (TBCTRH.LOF = 0) a programmed limit (TBCTRL.LIMIT). The trigger of this event is the transition from equal to below or bigger, not the fact of being below or above.  
If the standard transmit buffer event is used to indicate that new data has to be written to one of the INx locations, TBCTRH.LOF = 0 should be programmed.
- Transmit buffer error event:  
The software has written to a full buffer. The written value is ignored.



**Figure 21-20 Transmit Buffer Events**

The receive FIFO buffer mechanism detects the following events, that can lead to an interrupt (if enabled). 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 OUTRL.

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 OUTRL, 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 RBCTRH.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 (RBCTRH.RNM = 0):  
 The filling level of the receive buffer (given by TRBSRH.RBFLVL) exceeds (RBCTRH.LOF = 1) or falls below (RBCTRH.LOF = 0) a programmed limit (RBCTRL.LIMIT). The trigger of this event is the transition from equal to below or

**Universal Serial Interface Channel**

greater, not the fact of being below or above.

If the standard receive buffer event is used to indicate that new data has to be read from OUTRL, RBCTRH.LOF = 1 should be programmed.

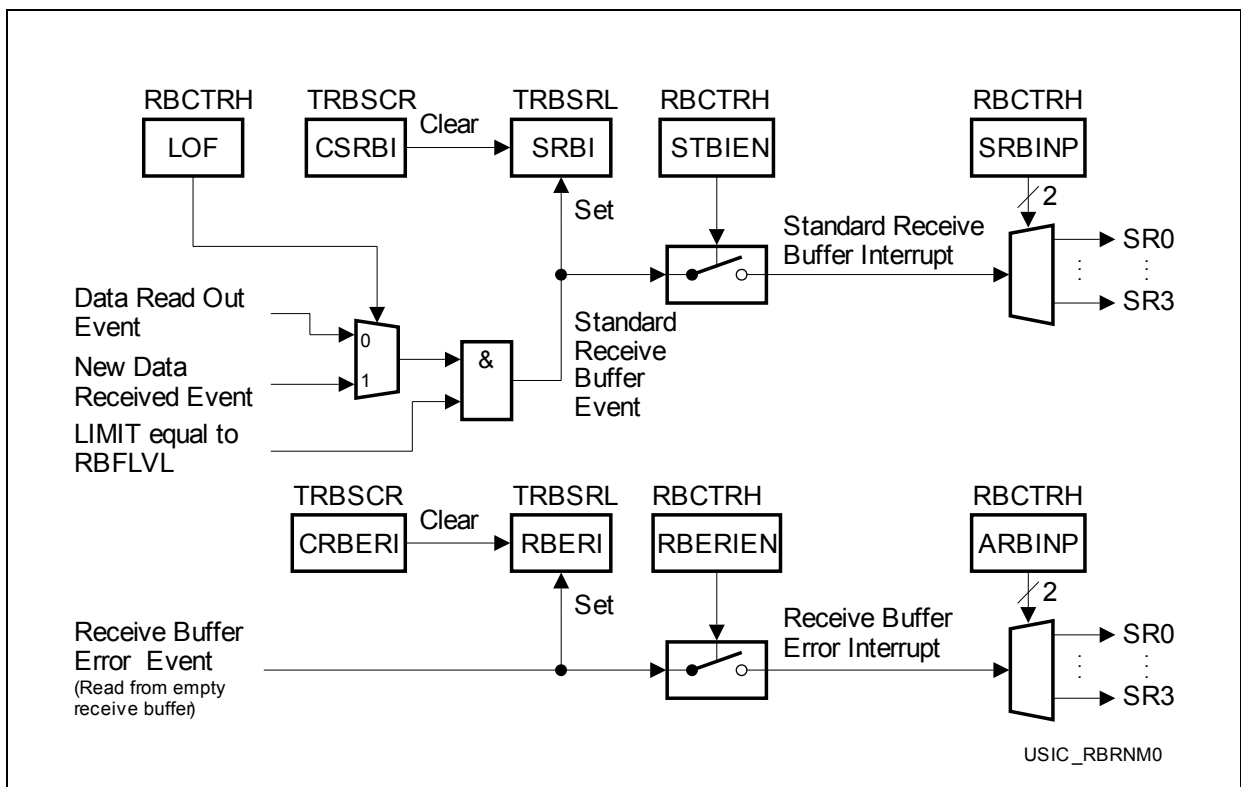
- Standard receive buffer event in RCI mode (RBCTRH.RNM = 1):  
If the OUTR stage is updated with a new data value with RCI[4] = 0.
- Alternative receive buffer event in filling level mode (RBCTRH.RNM = 0): not used
- Alternative receive buffer event in RCI mode (RBCTRH.RNM = 1):  
If the OUTR stage is updated with a new value with RCI[4] = 1.

- Receive buffer error event:

The software reads from an empty buffer. The read data is invalid.

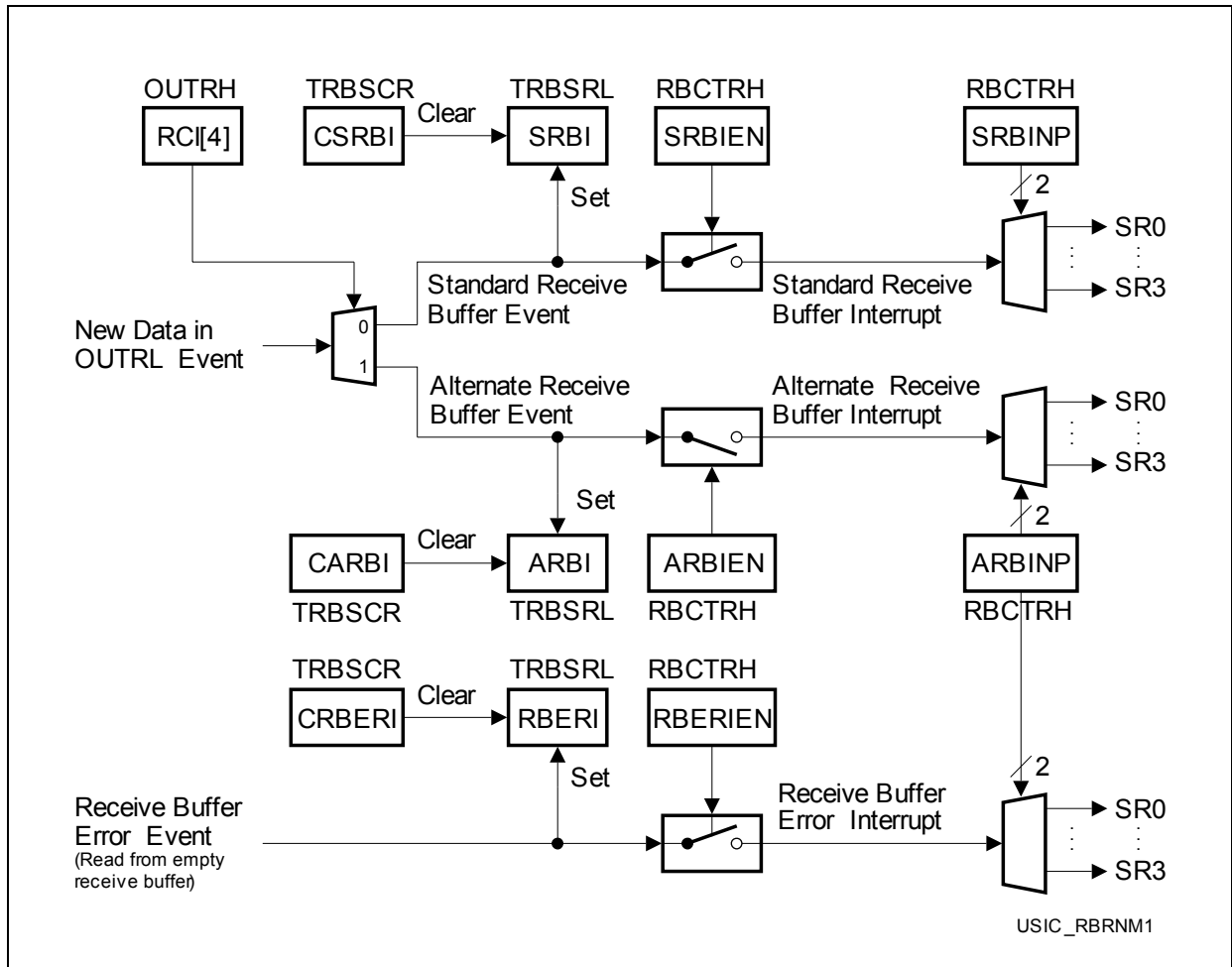
**Figure 21-21** shows the receiver buffer events and interrupts in filling level mode.

*Note: A buffer event in filling level mode occurs only when the filling level transitions away from the threshold value. Transitions starting with a filling level other than the threshold level generate no trigger event.*



**Figure 21-21 Receiver Buffer Events in Filling Level Mode**

**Figure 21-22** shows the receiver buffer events and interrupts in RCI mode.



**Figure 21-22 Receiver Buffer Events in RCI Mode**

**Table 21-7** shows the registers, bits and bit fields to indicate the buffer events and to control the interrupts related to the FIFO buffers (transmit and the receive) of a USIC channel.

**Table 21-7 Buffer Events and Interrupt Handling**

Event	Indication Flag	Indication cleared by	Interrupt enabled by	SRx Output selected by
Standard transmit buffer event	TRBSRL. STBI	TRBSCR. CSTBI	TBCTRH. STBIEN	TBCTRH. STBINP
Transmit buffer error event	TRBSRL. TBERI	TRBSCR. CTBERI	TBCTRH. TBERIEN	TBCTRH. ATBINP
Standard receive buffer event	TRBSRL. SRBI	TRBSCR. CSRBI	RBCTRH. SRBIEN	RBCTRH. SRBINP

**Universal Serial Interface Channel**

**Table 21-7 Buffer Events and Interrupt Handling**

<b>Event</b>	<b>Indication Flag</b>	<b>Indication cleared by</b>	<b>Interrupt enabled by</b>	<b>SRx Output selected by</b>
Alternative receive buffer event	TRBSRL. ARBI	TRBSCR. CARBI	RBCTRH. ARBIEN	RBCTRH. ARBINP
Receive buffer error event	TRBSRL. RBERI	TRBSCR. CRBERI	RBCTRH. RBERIEN	RBCTRH. ARBINTXDP

### **21.2.13.3 FIFO Buffer Bypass**

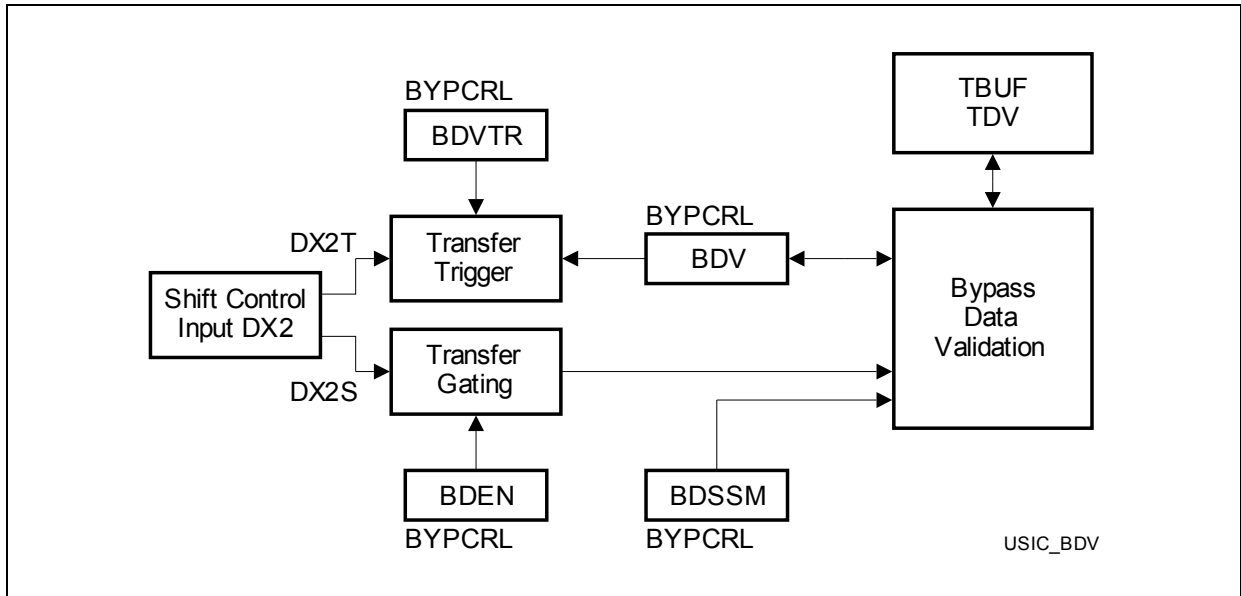
The data bypass mechanism is part of the transmit FIFO control block. It allows to introduce a data word in the data stream without modifying the transmit FIFO buffer contents, e.g. to send a high-priority message. The bypass structure consists of a bypass data word of maximum 16 bits in register BYP and some associated control information in registers BYPCRL and BYPCRH. 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 BYRCRL.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 TCSRL.TDV = 0 (TBUF is empty).
- Bypass data can only be transferred to TBUF if the bypass is enabled by BYPCRL.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 21-23](#).

- A transfer gating logic enables or disables the bypass data word transfer to TBUF under software or under hardware control. If the input stage DX2 is not needed for data shifting, signal DX2S can be used for gating purposes. The transfer gating logic is controlled by bit field BYPCRL.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 BYPCRL.BDVTR.
- A bypass data validation logic combining the inputs from the gating logic, the triggering logic and TCSRL.TDV.



**Figure 21-23 Bypass Data Validation**

With this structure, the following bypass data transfer functionality can be achieved:

- Bit BYPCRL.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 BYPCRL.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).

#### 21.2.13.4 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.

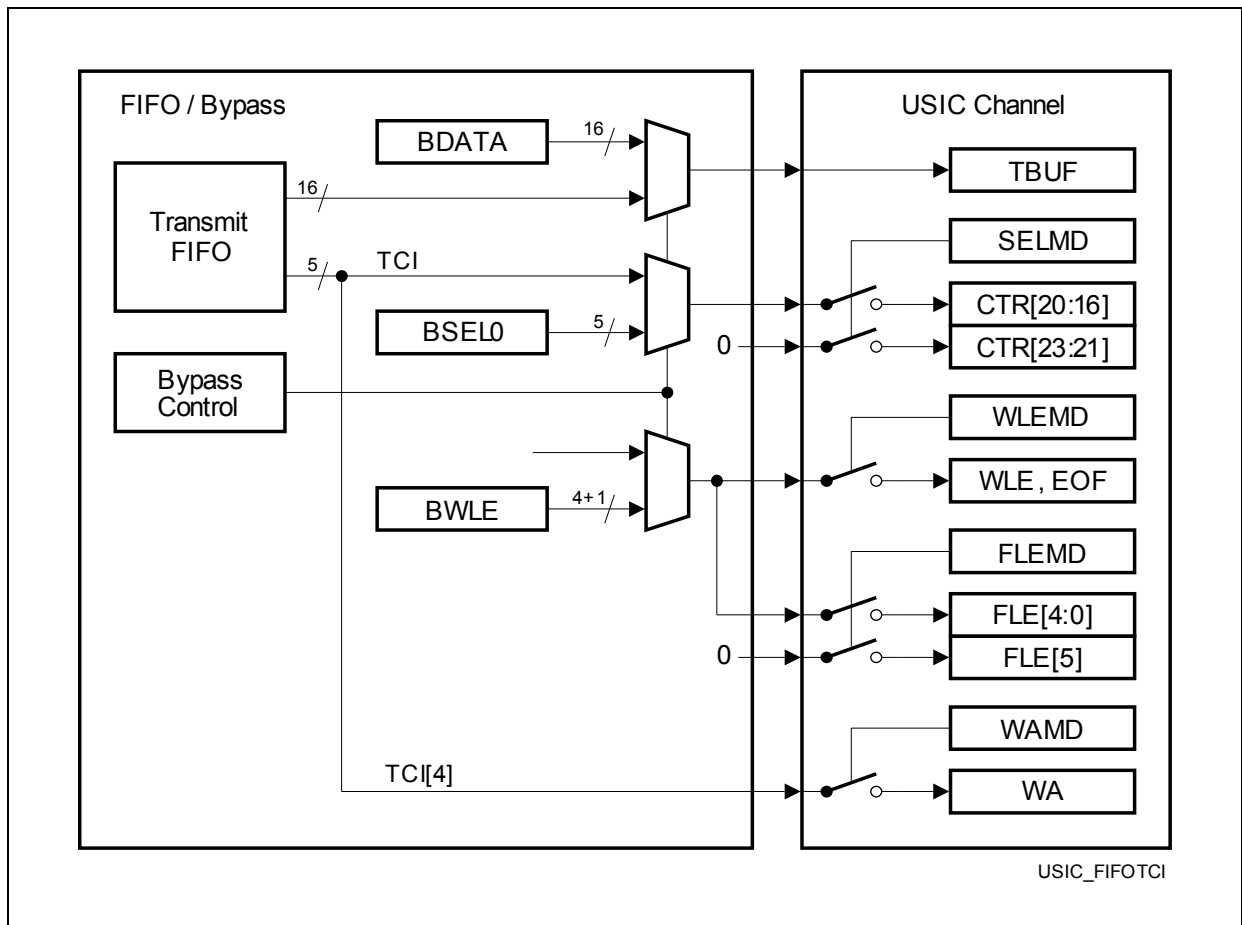
In order to avoid data loss and stalling of the CPU due to delayed software accesses, the baud rate, the word length and the software access mechanism have to be taken into account. Each access to the FIFO data buffer area by software or by hardware takes one period of  $f_{SYS}$ . Especially a continuous flow of very short, consecutive data words can lead to an access limitation.



### 21.2.13.5 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.

- TCSRL.SELMD = 1: update of PCRH.CTR[20:16] by FIFO TCI or BYPCRH.BSELO with additional clear of PCRH.CTR[23:21]
- TCSRL.WLEMD = 1: update of SCTR.H.WLE and TCSRL.EOF by FIFO TCI or BYPCRL.BWLE (if the WLE information is overwritten by TCI or BWLE, the user has to take care that FLE is set accordingly)
- TCSRL.FLEMD = 1: update of SCTR.H.FLE[4:0] by FIFO TCI or BYPCRL.BWLE with additional clear of SCTR.H.FLE[5]
- TCSRL.WAMD = 1: update of TCSRL.WA by FIFO TCI[4]



**Figure 21-24 TCI Handling with FIFO / Bypass**

## 21.2.14 FIFO Buffer and Bypass Registers

### 21.2.14.1 Bypass Registers

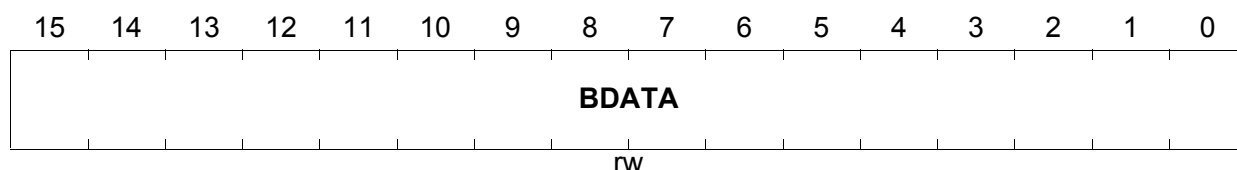
A write action to at least the low byte of the bypass data register sets BYPCRL.BDV = 1 (bypass data tagged valid).

#### **BYP**

**Bypass Data Register**

**(100<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



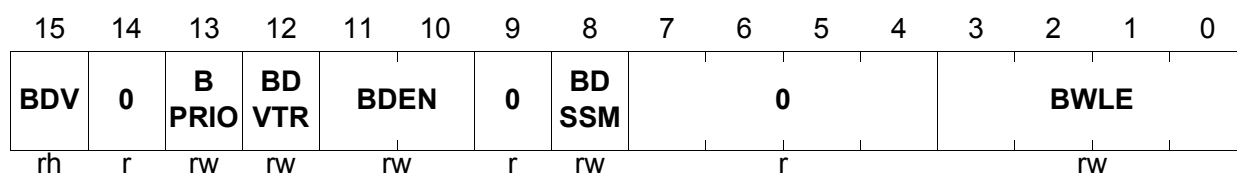
Bit (Field)	Width	Type	Description
<b>BDATA</b>	[15:0]	rw	<b>Bypass Data</b> This bit field contains the bypass data.

#### **BYPCRL**

**Bypass Control Register L**

**(104<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
<b>BWLE</b>	[3:0]	rw	<b>Bypass Word Length</b> This bit field defines the word length of the bypass data. The word length is given by BWLE + 1 with the data word being right-aligned in the data buffer at the bit positions [BWLE down to 0]. The bypass data word is always considered as an own frame with the length of BWLE. Same coding as SCTR.H.WLE.

**Universal Serial Interface Channel**

<b>Field</b>	<b>Bits</b>	<b>Type</b>	<b>Description</b>
<b>BDSSM</b>	8	rw	<b>Bypass Data Single Shot Mode</b> This bit defines if the bypass data is considered as permanently valid or if the bypass data is only transferred once (single shot mode). 0 <sub>B</sub> The bypass data is still considered as valid after it has been loaded into TBUF. The loading of the data into TBUF does not clear BDV. 1 <sub>B</sub> The bypass data is considered as invalid after it has been loaded into TBUF. The loading of the data into TBUF clears BDV.
<b>BDEN</b>	[11:10]	rw	<b>Bypass Data Enable</b> This bit field defines if and how the transfer of bypass data to TBUF is enabled. 00 <sub>B</sub> The transfer of bypass data is disabled. 01 <sub>B</sub> The transfer of bypass data to TBUF is possible. Bypass data will be transferred to TBUF according to its priority if BDV = 1. 10 <sub>B</sub> Gated bypass data transfer is enabled. Bypass data will be transferred to TBUF according to its priority if BDV = 1 and while DX2S = 0. 11 <sub>B</sub> Gated bypass data transfer is enabled. Bypass data will be transferred to TBUF according to its priority if BDV = 1 and while DX2S = 1.
<b>BDVTR</b>	12	rw	<b>Bypass Data Valid Trigger</b> This bit enables the bypass data for being tagged valid when DX2T is active (for time framing or time-out purposes). 0 <sub>B</sub> Bit BDV is not influenced by DX2T. 1 <sub>B</sub> Bit BDV is set if DX2T is active.
<b>BPRIO</b>	13	rw	<b>Bypass Priority</b> This bit defines the priority between the bypass data and the transmit FIFO data. 0 <sub>B</sub> The transmit FIFO data has a higher priority than the bypass data. 1 <sub>B</sub> The bypass data has a higher priority than the transmit FIFO data.

**Universal Serial Interface Channel**

Field	Bits	Type	Description
<b>BDV</b>	15	rh	<b>Bypass Data Valid</b> This bit defines if the bypass data is valid for a transfer to TBUF. This bit is set automatically by a write access to at least the low-byte of register BYP. It can be cleared by software by writing TRBSCR.CBDV. 0 <sub>B</sub> The bypass data is not valid. 1 <sub>B</sub> The bypass data is valid.
<b>0</b>	[7:4], 9, 14	r	<b>Reserved</b> Read as 0; should be written with 0.

**BYPCR H**

**Bypass Control Register H**

**(106<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

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

Field	Bits	Type	Description
<b>BSELO</b>	[4:0]	rw	<b>Bypass Select Outputs</b> This bit field contains the value that is written to PCRH.CTR[20:16] if bypass data is transferred to TBUF while TCSRL.SELMD = 1. In the SSC protocol, this bit field can be used to define which SELOx output line will be activated when bypass data is transmitted.
<b>0</b>	[15:5]	r	<b>Reserved</b> Read as 0; should be written with 0.

### 21.2.14.2 General FIFO Buffer Control Registers

The transmit and receive FIFO status information of UxCy is given in registers UxCy\_TRBSRL/H.

The bits related to the transmitter buffer in this register can only be 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 TRBSRL can be cleared by writing a 1 to the corresponding bit position in register TRBSCR, whereas writing a 0 has no 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”.

#### TRBSRL

##### Transmit/Receive Buffer Status Register L

(118<sub>H</sub>)

Reset Value: 0808<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	T BUS	T FUL L	T EMP TY	0	TB ERI	ST BI	0	R BUS	R FUL L	R EMP TY	AR BI	RB ERI	SR BI		
r	rh	rh	rh	r	rwh	rwh	r	rh	rh	rh	rwh	rwh	rwh		

Field	Bits	Type	Description
SRBI	0	rwh	<b>Standard Receive Buffer Event</b> This bit indicates that a standard receive buffer event has been detected. It is cleared by writing TRBSCR.CSRBI = 1. If enabled by RBCTRH.SRBIEN, the service request output SRx selected by RBCTRH.SRBINP becomes activated if a standard receive buffer event is detected. 0 <sub>B</sub> A standard receive buffer event has not been detected. 1 <sub>B</sub> A standard receive buffer event has been detected.

**Universal Serial Interface Channel**

Field	Bits	Type	Description
<b>RBERI</b>	1	rwh	<p><b>Receive Buffer Error Event</b></p> <p>This bit indicates that a receive buffer error event has been detected. It is cleared by writing TRBSCR.CRBERI = 1.</p> <p>If enabled by RBCTRH.RBERIEN, the service request output SRx selected by RBCTRH.ARBINP becomes activated if a receive buffer error event is detected.</p> <p>0<sub>B</sub> A receive buffer error event has not been detected.</p> <p>1<sub>B</sub> A receive buffer error event has been detected.</p>
<b>ARBI</b>	2	rwh	<p><b>Alternative Receive Buffer Event</b></p> <p>This bit indicates that an alternative receive buffer event has been detected. It is cleared by writing TRBSCR.CARBI = 1.</p> <p>If enabled by RBCTRH.ARBIEEN, the service request output SRx selected by RBCTRH.ARBINP becomes activated if an alternative receive buffer event is detected.</p> <p>0<sub>B</sub> An alternative receive buffer event has not been detected.</p> <p>1<sub>B</sub> An alternative receive buffer event has been detected.</p>
<b>REMPY</b>	3	rh	<p><b>Receive Buffer Empty</b></p> <p>This bit indicates whether the receive buffer is empty.</p> <p>0<sub>B</sub> The receive buffer is not empty.</p> <p>1<sub>B</sub> The receive buffer is empty.</p>
<b>RFULL</b>	4	rh	<p><b>Receive Buffer Full</b></p> <p>This bit indicates whether the receive buffer is full.</p> <p>0<sub>B</sub> The receive buffer is not full.</p> <p>1<sub>B</sub> The receive buffer is full.</p>

**Universal Serial Interface Channel**

<b>Field</b>	<b>Bits</b>	<b>Type</b>	<b>Description</b>
<b>RBUS</b>	5	rh	<b>Receive Buffer Busy</b> This bit indicates whether the receive buffer is currently updated by the FIFO handler. 0 <sub>B</sub> The receive buffer information has been completely updated. 1 <sub>B</sub> The OUTRL/H update from the FIFO memory is ongoing. A read from OUTRL/H will be delayed. FIFO pointers from the previous read are not yet updated.
<b>STBI</b>	8	rwh	<b>Standard Transmit Buffer Event</b> This bit indicates that a standard transmit buffer event has been detected. It is cleared by writing TRBSCR.CSTBI = 1. If enabled by TBCTRH.STBIEN, the service request output SRx selected by TBCTRH.STBINP becomes activated if a standard transmit buffer event is detected. 0 <sub>B</sub> A standard transmit buffer event has not been detected. 1 <sub>B</sub> A standard transmit buffer event has been detected.
<b>TBERI</b>	9	rwh	<b>Transmit Buffer Error Event</b> This bit indicates that a transmit buffer error event has been detected. It is cleared by writing TRBSCR.CTBERI = 1. If enabled by TBCTRH.TBERIEN, the service request output SRx selected by TBCTRH.ATBINP becomes activated if a transmit buffer error event is detected. 0 <sub>B</sub> A transmit buffer error event has not been detected. 1 <sub>B</sub> A transmit buffer error event has been detected.
<b>EMPTY</b>	11	rh	<b>Transmit Buffer Empty</b> This bit indicates whether the transmit buffer is empty. 0 <sub>B</sub> The transmit buffer is not empty. 1 <sub>B</sub> The transmit buffer is empty.

**Universal Serial Interface Channel**

Field	Bits	Type	Description
<b>TFULL</b>	12	rh	<b>Transmit Buffer Full</b> This bit indicates whether the transmit buffer is full. 0 <sub>B</sub> The transmit buffer is not full. 1 <sub>B</sub> The transmit buffer is full.
<b>TBUS</b>	13	rh	<b>Transmit Buffer Busy</b> This bit indicates whether the transmit buffer is currently updated by the FIFO handler. 0 <sub>B</sub> The transmit buffer information has been completely updated. 1 <sub>B</sub> 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.
<b>0</b>	[7:6], 10, [15:14]	r	<b>Reserved</b> Read as 0; should be written with 0.

**TRBSRH**

**Transmit/Receive Buffer Status Register H**

(11A<sub>H</sub>)

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>0</b>	<b>TBFLVL</b>						<b>0</b>	<b>RBFLVL</b>							
r	rh						r	rh							

Field	Bits	Type	Description
<b>RBFLVL</b>	[6:0]	rh	<b>Receive Buffer Filling Level</b> This bit field indicates the filling level of the receive buffer, starting with 0 for an empty buffer.
<b>TBFLVL</b>	[14:8]	rh	<b>Transmit Buffer Filling Level</b> This bit field indicates the filling level of the transmit buffer, starting with 0 for an empty buffer.
<b>0</b>	7, 15	r	<b>Reserved</b> Read as 0; should be written with 0.



## Universal Serial Interface Channel

The bits in register TRBSCR are used to clear the notification bits in register TRBSRL 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

(11C<sub>H</sub>)

Reset Value: 0000<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FLU SH TB	FLU SH RB		0		C BDV	C TB ERI	C ST BI			0			C AR BI	C RB ERI	C SR BI
w	w		r		w	w	w			r			w	w	w

Field	Bits	Type	Description
CSRBI	0	w	<b>Clear Standard Receive Buffer Event</b> 0 <sub>B</sub> No effect. 1 <sub>B</sub> Clear TRBSRL.SRBI.
CRBERI	1	w	<b>Clear Receive Buffer Error Event</b> 0 <sub>B</sub> No effect. 1 <sub>B</sub> Clear TRBSRL.RBERI.
CARBI	2	w	<b>Clear Alternative Receive Buffer Event</b> 0 <sub>B</sub> No effect. 1 <sub>B</sub> Clear TRBSRL.ARBI.
CSTBI	8	w	<b>Clear Standard Transmit Buffer Event</b> 0 <sub>B</sub> No effect. 1 <sub>B</sub> Clear TRBSRL.STBI.
CTBERI	9	w	<b>Clear Transmit Buffer Error Event</b> 0 <sub>B</sub> No effect. 1 <sub>B</sub> Clear TRBSRL.TBERI.
CBDV	10	w	<b>Clear Bypass Data Valid</b> 0 <sub>B</sub> No effect. 1 <sub>B</sub> Clear BYPCRL.BDV.
FLUSHRB	14	w	<b>Flush Receive Buffer</b> 0 <sub>B</sub> No effect. 1 <sub>B</sub> 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.

**Universal Serial Interface Channel**

Field	Bits	Type	Description
<b>FLUSHTB</b>	15	w	<b>Flush Transmit Buffer</b> $0_B$ No effect. $1_B$ The transmit FIFO buffer is cleared (filling level is cleared and output pointer is set to input pointer value). Should only be used while the FIFO buffer is not taking part in data traffic.
<b>0</b>	[7:3], [13:11]	r	<b>Reserved</b> Read as 0; should be written with 0.

### 21.2.14.3 Transmit FIFO Buffer Control Registers

The transmit FIFO buffer is controlled by registers TBCTRL and TBCTRH. These registers can only be written if the transmit buffer functionality is enabled by CCFG.TB = 1, otherwise write accesses are ignored.

#### TBCTRL

##### Transmitter Buffer Control Register L (110<sub>H</sub>)

**Reset Value: 0000<sub>H</sub>**

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

Field	Bits	Type	Description
<b>DPTR</b>	[5:0]	w	<b>Data Pointer</b> 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 TRBPTRL 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.
<b>LIMIT</b>	[13:8]	rw	<b>Limit For Interrupt Generation</b> This bit field defines the target filling level of the transmit FIFO buffer that is used for the standard transmit buffer event detection.
<b>0</b>	[7:6], [15:14]	r	<b>Reserved</b> Read as 0; should be written with 0.

**Universal Serial Interface Channel**

**TBCTRH**

**Transmitter Buffer Control Register H (112<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>TB ERI EN</b>	<b>ST BI EN</b>	<b>0</b>	<b>LOF</b>	<b>0</b>	<b>SIZE</b>			<b>0</b>			<b>ATBINP</b>		<b>0</b>	<b>STBINP</b>	
rw	rw	r	rw	r	rw	rw	rw	rw	r	rw	rw	rw	r	rw	rw

Field	Bits	Type	Description
<b>STBINP</b>	[1:0]	rw	<b>Standard Transmit Buffer Interrupt Node Pointer</b> This bit field defines which service request output SRx becomes activated in case of a standard transmit buffer event. 00 <sub>B</sub> Output SR0 becomes activated. 01 <sub>B</sub> Output SR1 becomes activated. 10 <sub>B</sub> Output SR2 becomes activated. 11 <sub>B</sub> Output SR3 becomes activated.
<b>ATBINP</b>	[4:3]	rw	<b>Alternative Transmit Buffer Interrupt Node Pointer</b> This bit field define which service request output SRx will be activated in case of a transmit buffer error event. 00 <sub>B</sub> Output SR0 becomes activated. 01 <sub>B</sub> Output SR1 becomes activated. 10 <sub>B</sub> Output SR2 becomes activated. 11 <sub>B</sub> Output SR3 becomes activated.
<b>SIZE</b>	[10:8]	rw	<b>Buffer Size</b> This bit field defines the number of FIFO entries assigned to the transmit FIFO buffer. 000 <sub>B</sub> The FIFO mechanism is disabled. The buffer does not accept any request for data. 001 <sub>B</sub> The FIFO buffer contains 2 entries. 010 <sub>B</sub> The FIFO buffer contains 4 entries. 011 <sub>B</sub> The FIFO buffer contains 8 entries. 100 <sub>B</sub> The FIFO buffer contains 16 entries. 101 <sub>B</sub> The FIFO buffer contains 32 entries. 110 <sub>B</sub> The FIFO buffer contains 64 entries. 111 <sub>B</sub> Reserved

**Universal Serial Interface Channel**

Field	Bits	Type	Description
<b>LOF</b>	12	rw	<b>Buffer Event on Limit Overflow</b> This bit defines which relation between filling level and programmed limit leads to a standard transmit buffer event. $0_B$ A standard transmit buffer event occurs when the filling level equals the limit value and gets lower due to transmission of a data word. $1_B$ A standard transmit buffer interrupt event occurs when the filling level equals the limit value and gets bigger due to a write access to a data input location INx.
<b>STBIEN</b>	14	rw	<b>Standard Transmit Buffer Interrupt Enable</b> This bit enables/disables the generation of a standard transmit buffer interrupt in case of a standard transmit buffer event. $0_B$ The standard transmit buffer interrupt generation is disabled. $1_B$ The standard transmit buffer interrupt generation is enabled.
<b>TBERIEN</b>	15	rw	<b>Transmit Buffer Error Interrupt Enable</b> This bit enables/disables the generation of a transmit buffer error interrupt in case of a transmit buffer error event (software writes to a full transmit buffer). $0_B$ The transmit buffer error interrupt generation is disabled. $1_B$ The transmit buffer error interrupt generation is enabled.
<b>0</b>	2, [7:5], 11, 13	r	<b>Reserved</b> Read as 0; should be written with 0.

#### 21.2.14.4 Receive FIFO Buffer Control Registers

The receive FIFO buffer is controlled by registers RBCTRL and RBCTRH. These registers can only be written if the receive buffer functionality is enabled by CCFG.RB = 1, otherwise write accesses are ignored.

##### **RBCTRL**

**Receiver Buffer Control Register L (114<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>0</b>		<b>LIMIT</b>						<b>0</b>		<b>DPTR</b>					
r		rw						r		w					

Field	Bits	Type	Description
<b>DPTR</b>	[5:0]	w	<b>Data Pointer</b> 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 TRBPTRH 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.
<b>LIMIT</b>	[13:8]	rw	<b>Limit For Interrupt Generation</b> This bit field defines the target filling level of the receive FIFO buffer that is used for the standard receive buffer event detection.
<b>0</b>	[7:6], [15:14]	r	<b>Reserved</b> Read as 0; should be written with 0.

**Universal Serial Interface Channel**

**RBCTRH**

**Receiver Buffer Control Register H (116<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>RB ERI EN</b>	<b>SR BI EN</b>	<b>AR BI EN</b>	<b>LOF</b>	<b>RNM</b>	<b>SIZE</b>			<b>RCIM</b>		<b>0</b>	<b>ARBINP</b>		<b>0</b>	<b>SRBINP</b>	
rw	rw	rw	rw	rw	rw	rw	rw	rw	r	r	rw	r	rw	rw	rw

Field	Bits	Type	Description
<b>SRBINP</b>	[1:0]	rw	<b>Standard Receive Buffer Interrupt Node Pointer</b> This bit field defines which service request output SRx becomes activated in case of a standard receive buffer event. 00 <sub>B</sub> Output SR0 becomes activated. 01 <sub>B</sub> Output SR1 becomes activated. 10 <sub>B</sub> Output SR2 becomes activated. 11 <sub>B</sub> Output SR3 becomes activated.
<b>ARBINP</b>	[4:3]	rw	<b>Alternative Receive Buffer Interrupt Node Pointer</b> 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. 00 <sub>B</sub> The output SR0 becomes activated. 01 <sub>B</sub> The output SR1 becomes activated. 10 <sub>B</sub> The output SR2 becomes activated. 11 <sub>B</sub> The output SR3 becomes activated.
<b>RCIM</b>	[7:6]	rw	<b>Receiver Control Information Mode</b> 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)RH. 00 <sub>B</sub> RCI[4] = PERR, RCI[3:0] = WLEN 01 <sub>B</sub> RCI[4] = SOF, RCI[3:0] = WLEN 10 <sub>B</sub> RCI[4] = 0, RCI[3:0] = WLEN 11 <sub>B</sub> RCI[4] = PERR, RCI[3] = PAR, RCI[2:1] = 00 <sub>B</sub> , RCI[0] = SOF

**Universal Serial Interface Channel**

Field	Bits	Type	Description
<b>SIZE</b>	[10:8]	rw	<b>Buffer Size</b> This bit field defines the number of FIFO entries assigned to the receive FIFO buffer. 000 <sub>B</sub> The FIFO mechanism is disabled. The buffer does not accept any request for data. 001 <sub>B</sub> The FIFO buffer contains 2 entries. 010 <sub>B</sub> The FIFO buffer contains 4 entries. 011 <sub>B</sub> The FIFO buffer contains 8 entries. 100 <sub>B</sub> The FIFO buffer contains 16 entries. 101 <sub>B</sub> The FIFO buffer contains 32 entries. 110 <sub>B</sub> The FIFO buffer contains 64 entries. 111 <sub>B</sub> Reserved
<b>RNM</b>	11	rw	<b>Receiver Notification Mode</b> This bit defines the receive buffer event mode. The receive buffer error event is not affected by RNM. 0 <sub>B</sub> Filling level mode: A standard receive buffer event occurs when the filling level equals the limit value and changes, either due to a read access from OUTRL (LOF = 0) or due to a new received data word (LOF = 1). 1 <sub>B</sub> RCI mode: A standard receive buffer event occurs when register OUTRL is updated with a new value if the corresponding value in OUTRH.RCI[4] = 0. If OUTRH.RCI[4] = 1, an alternative receive buffer event occurs instead of the standard receive buffer event.
<b>LOF</b>	12	rw	<b>Buffer Event on Limit Overflow</b> This bit defines which relation between filling level and programmed limit leads to a standard receive buffer event in filling level mode (RNM = 0). In RCI mode (RNM = 1), bit fields LIMIT and LOF are ignored. 0 <sub>B</sub> A standard receive buffer event occurs when the filling level equals the limit value and gets lower due to a read access from OUTRL. 1 <sub>B</sub> 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.



**Universal Serial Interface Channel**

<b>Field</b>	<b>Bits</b>	<b>Type</b>	<b>Description</b>
<b>ARBIEN</b>	13	rw	<b>Alternative Receive Buffer Interrupt Enable</b> This bit enables/disables the generation of an alternative receive buffer interrupt in case of an alternative receive buffer event. $0_B$ The alternative receive buffer interrupt generation is disabled. $1_B$ The alternative receive buffer interrupt generation is enabled.
<b>SRBIEN</b>	14	rw	<b>Standard Receive Buffer Interrupt Enable</b> This bit enables/disables the generation of a standard receive buffer interrupt in case of a standard receive buffer event. $0_B$ The standard receive buffer interrupt generation is disabled. $1_B$ The standard receive buffer interrupt generation is enabled.
<b>RBERIEN</b>	15	rw	<b>Receive Buffer Error Interrupt Enable</b> This bit enables/disables the generation of a receive buffer error interrupt in case of a receive buffer error event (the software reads from an empty receive buffer). $0_B$ The receive buffer error interrupt generation is disabled. $1_B$ The receive buffer error interrupt generation is enabled.
<b>0</b>	2, 5	r	<b>Reserved</b> Read as 0; should be written with 0.

### 21.2.14.5 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.

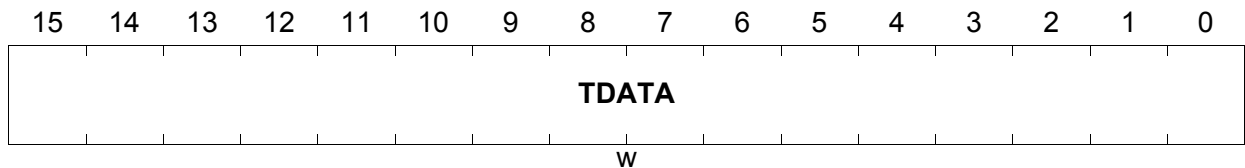
If the FIFO is already full and new data is written to it, the write access is ignored and a transmit buffer error event is signaled.

**INx (x = 00-31)**

**Transmit FIFO Buffer Input Location x**

**(180<sub>H</sub> + x \*4)**

**Reset Value: 0000<sub>H</sub>**



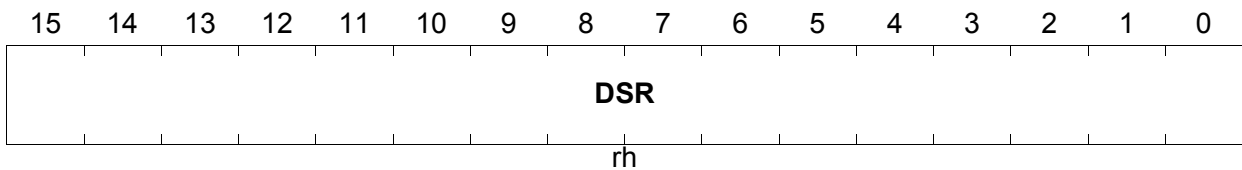
Field	Bits	Type	Description
<b>TDATA</b>	[15:0]	w	<b>Transmit Data</b> This bit field contains the data to be transmitted (write view), read actions deliver 0. A write action to at least the low byte of TDATA triggers the data storage in the FIFO.

### Universal Serial Interface Channel

The receiver FIFO buffer output register OUTRL shows the oldest received data word in the FIFO buffer. 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. Register OUTRH contains the receiver control information RCI containing the information selected by RBCTRH.RCIM. Write accesses to OUTRL/H are ignored.

#### OUTRL

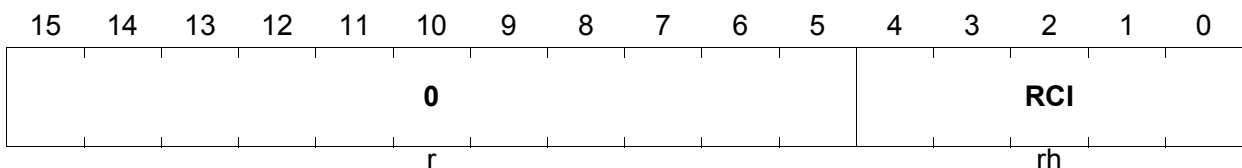
**Receiver Buffer Output Register L (120<sub>H</sub>)** **Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
DSR	[15:0]	rh	<b>Received Data</b> This bit field monitors the content of the oldest data word in the receive FIFO. Reading at least the low byte releases the buffer entry currently shown in DSR.

#### OUTRH

**Receiver Buffer Output Register H (122<sub>H</sub>)** **Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
RCI	[4:0]	rh	<b>Receiver Control Information</b> This bit field monitors the receiver control information associated to DSR. The bit structure of RCI depends on bit field RBCTRH.RCIM.
0	[15:5]	r	<b>Reserved</b> Read as 0; should be written with 0.

### Universal Serial Interface Channel

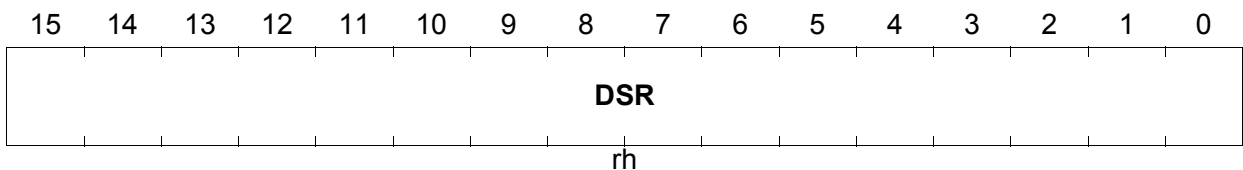
If a debugger should be used to monitor the received data in the FIFO buffer, the FIFO mechanism must not be activated in order to guaranty data consistency. Therefore, a second address set is available, named OUTDRL/H (D like debugger), having the same bit fields like the original buffer output register OUTRL/H, 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 OUTDRL/H are ignored.

#### OUTDRL

##### Receiver Buffer Output Register L for Debugger

(124<sub>H</sub>)

Reset Value: 0000<sub>H</sub>



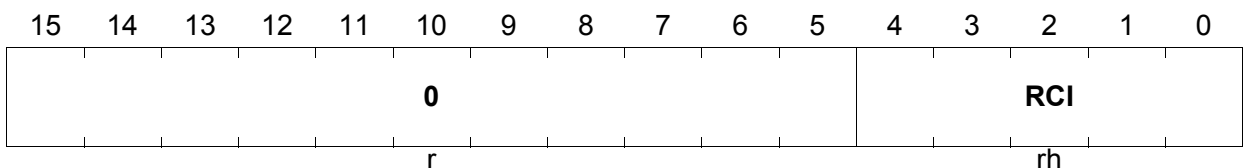
Field	Bits	Type	Description
DSR	[15:0]	rh	<b>Data from Shift Register</b> Same as OUTRL.DSR, but without releasing the buffer after a read action.

#### OUTDRH

##### Receiver Buffer Output Register H for Debugger

(126<sub>H</sub>)

Reset Value: 0000<sub>H</sub>



Field	Bits	Type	Description
RCI	[4:0]	rh	<b>Receive Control Information from Shift Register</b> Same as OUTRH.RCI.
0	[15:5]	r	<b>Reserved</b> Read as 0; should be written with 0.

### 21.2.14.6 FIFO Buffer Pointer Registers

The pointers for FIFO handling of the transmit and receive FIFO buffers are located in registers TRBPTRL (for the transmit buffer) and TRBPTRH (for the receive buffer). 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.

#### TRBPTRL

##### Transmit/Receive Buffer Pointer Register L

(108<sub>H</sub>)

Reset Value: 0000<sub>H</sub>

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

Field	Bits	Type	Description
TDIPTR	[5:0]	rh	<b>Transmitter Data Input Pointer</b> 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	<b>Transmitter Data Output Pointer</b> This bit field indicates the buffer entry that will be used for the next transmit data to be output to TBUF.
0	[7:6], [15:14]	r	<b>Reserved</b> Read as 0; should be written with 0.

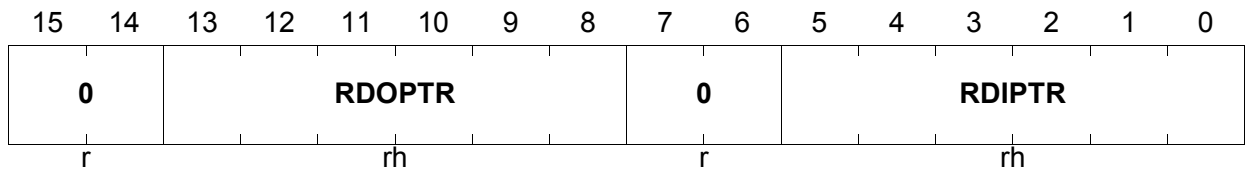
**Universal Serial Interface Channel**

**TRBPTRH**

**Transmit/Receive Buffer Pointer Register H**

**(10A<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
<b>RDIPTR</b>	[5:0]	rh	<b>Receiver Data Input Pointer</b> This bit field indicates the buffer entry that will be used for the next receive data coming from RBUF.
<b>RDOPTR</b>	[13:8]	rh	<b>Receiver Data Output Pointer</b> This bit field indicates the buffer entry that will be used for the next receive data to be output at the OUT(D)RL addresses.
<b>0</b>	[7:6], [15:14]	r	<b>Reserved</b> Read as 0; should be written with 0.

## 21.3 Asynchronous Serial Channel (ASC = UART)

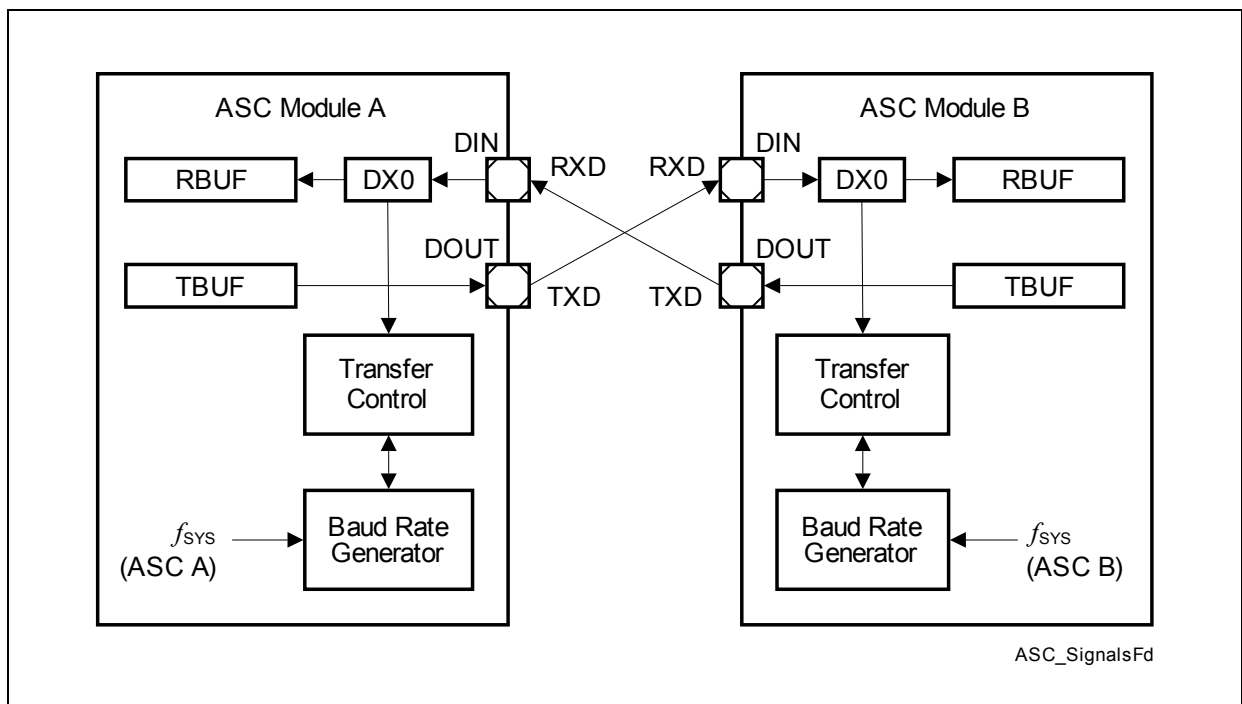
The asynchronous serial channel ASC covers the reception and the transmission of asynchronous data frames and provides a hardware LIN support. The receiver and transmitter being independent, frames can start at different points in time for transmission and reception. The ASC mode is selected by  $CCR.MODE = 0010_B$  with  $CCFG.ASC = 1$  (ASC mode available).

This chapter contains the following sections:

- Signal description (see [Page 21-110](#))
- Frame format (see [Page 21-111](#))
- Bit timing (see [Page 21-115](#))
- Operating the ASC (see [Page 21-114](#))
- Protocol registers (see [Page 21-123](#))
- Hardware LIN support (see [Page 21-129](#))

### 21.3.1 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.



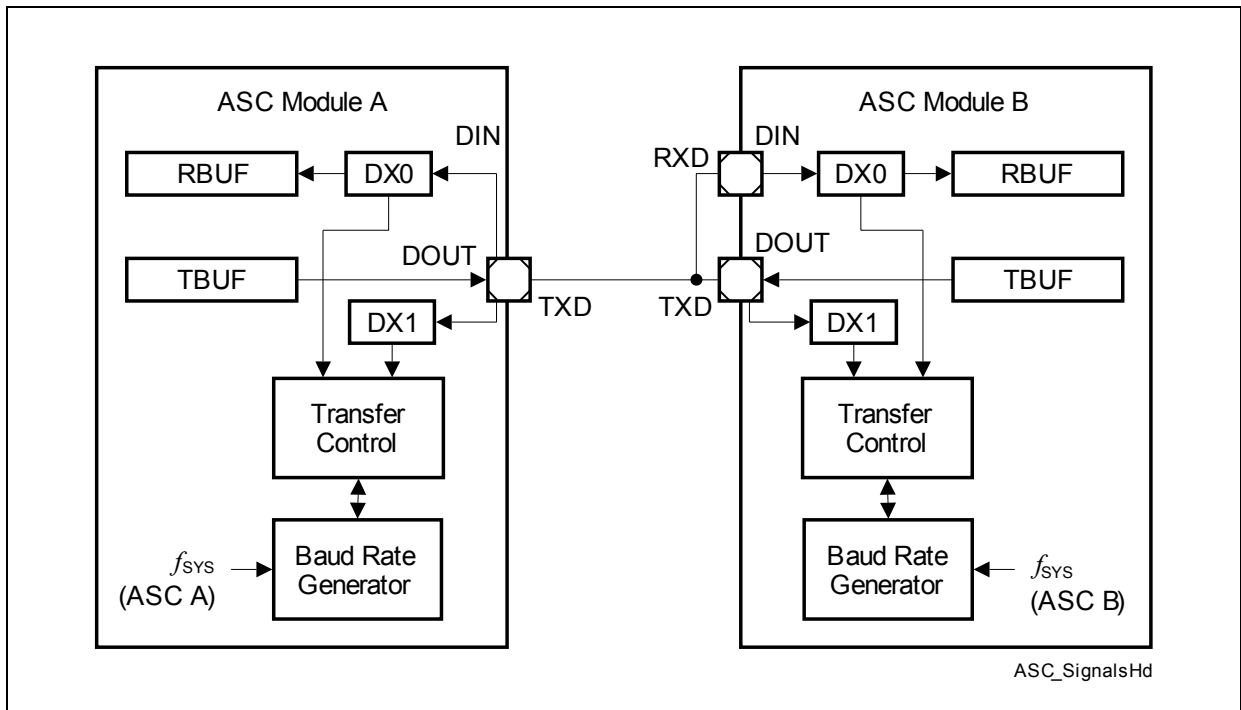
**Figure 21-25 ASC Signal Connections for Full-Duplex Communication**

For full-duplex communication, an independent communication line is needed for each transfer direction. [Figure 21-25](#) shows an example with a point-to-point full-duplex connection between two communication partners ASC A and ASC B.

## Universal Serial Interface Channel

For half-duplex or multi-transmitter communication, a single communication line is shared between the communication partners. **Figure 21-26** 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 DIN to the transmitter output DOUT. 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.



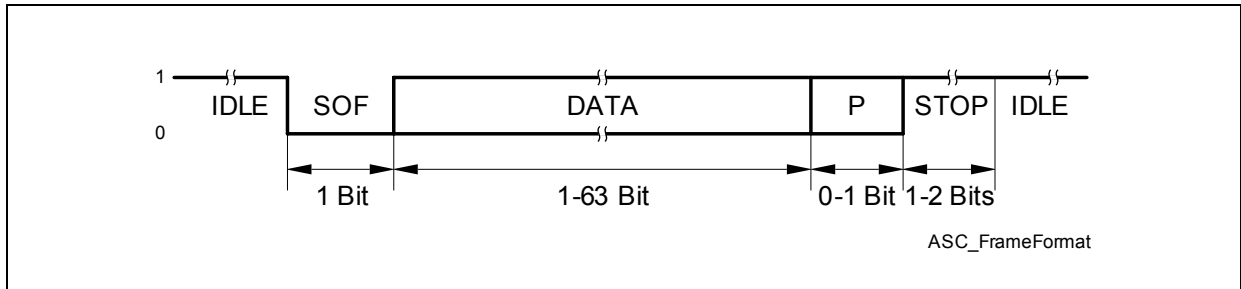
**Figure 21-26 ASC Signal Connections for Half-Duplex Communication**

### 21.3.2 Frame Format

A standard ASC frame is shown in **Figure 21-27**. It consists of:

- An idle time with the signal level 1.
- One start of frame bit (SOF) with the signal level 0.
- A data field containing a programmable number of data bits (1-63).
- A parity bit (P), programmable for either even or odd parity. It is optionally possible to handle frames without parity bit.
- One or two stop bits with the signal level 1.





**Figure 21-27 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.

### 21.3.2.1 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 PCRL.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 PCRL.IDM = 0.

In the case that the complete idle detection is enabled by PCRL.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).

### **21.3.2.2 Start Bit Detection**

The receiver input signal DIN (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.

### **21.3.2.3 Data Field**

The length of the data field (number of data bits) can be programmed by bit field SCTR.H.FLE. It can vary between 1 and 63 data bits, corresponding to values of SCTR.H.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 (SCTRL.PDL) is sent out.

The shift direction can be programmed by SCTRL.SDIR. The standard setting for ASC frames with LSB first is achieved with the default setting SDIR = 0.

### **21.3.2.4 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 as receiver buffer status information. These registers contain bits to monitor a protocol-related argument (PAR) and protocol-related error indication (PERR).

### **21.3.2.5 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.

### **21.3.3 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  $SCTRL.TRM = 01_B$  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 DIN) with  $DX0CR.INSW = 0$  and configure a transmit data output pin (signal DOUT). 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.
- **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 registers  $SCTRL$  and  $SCTRH$ . If required by the application, the data input and output signals can be inverted. Additionally, the parity mode has to be configured ( $CCR.PM$ ).

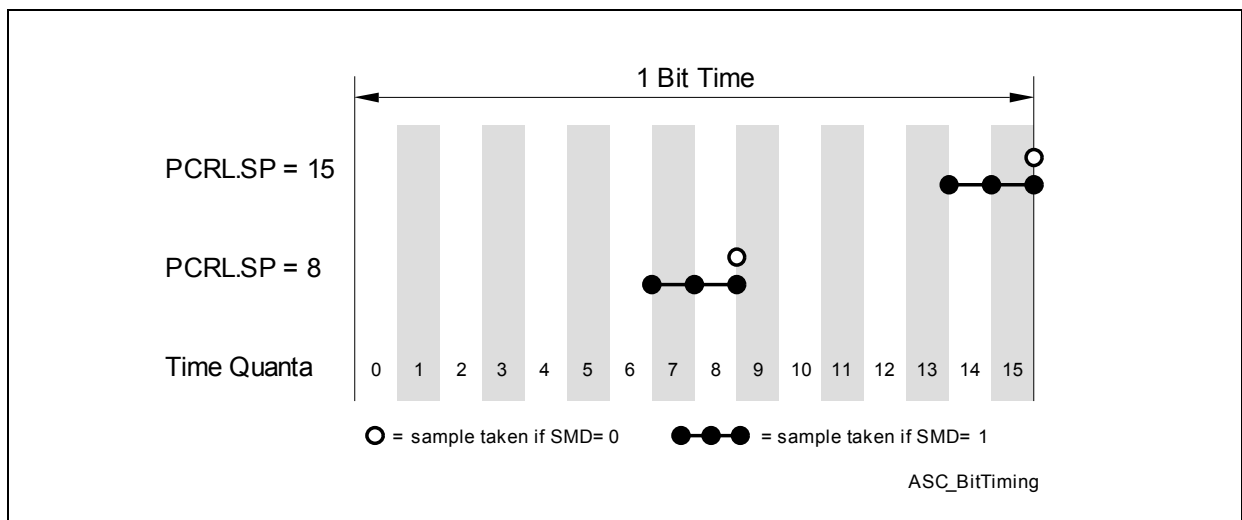
### 21.3.3.1 Bit Timing

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 BRGL.DCTQ and the length of a time quantum is given by BRGL.PCTQ.

In the example given in **Figure 21-28**, one bit time is composed of 16 time quanta (BRGL.DCTQ = 15). It is not recommended to program less than 4 time quanta per bit time.

Bit field PCRL.SP determines the position of the sampling point for the bit value. The value of PCRL.SP must not be set to a value greater than BRGL.DCTQ. It is possible to sample the bit value only once per bit time or to take the average of samples. Depending on bit PCRL.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 21-28 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.

### 21.3.3.2 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 BRGL define the baud rate setting:

- BRGL.CTQSEL  
to define the input frequency  $f_{CTQIN}$  for the time quanta generation
- BRGL.PCTQ  
to define the length of a time quantum (division of  $f_{CTQIN}$  by 1, 2, 3, or 4)
- BRGL.DCTQ  
to define the number of time quanta per bit time

The standard setting is given by CTQSEL = 00<sub>B</sub> ( $f_{CTQIN} = f_{PDIV}$ ) and PPPEN = 0 ( $f_{PPP} = f_{PIN}$ ). Under these conditions, the baud rate is given by:

$$f_{ASC} = f_{PIN} \times \frac{1}{PDIV + 1} \times \frac{1}{PCTQ + 1} \times \frac{1}{DCTQ + 1} \quad (21.6)$$

In order to generate slower frequencies, two additional divide-by-2 stages can be selected by CTQSEL = 10<sub>B</sub> ( $f_{CTQIN} = f_{SCLK}$ ) and PPPEN = 1 ( $f_{PPP} = f_{MCLK}$ ), leading to:

$$f_{ASC} = \frac{f_{PIN}}{2 \times 2} \times \frac{1}{PDIV + 1} \times \frac{1}{PCTQ + 1} \times \frac{1}{DCTQ + 1} \quad (21.7)$$

### 21.3.3.3 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 PCRL.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 PCRL.RNIEN.

### 21.3.3.4 Collision Detection

In some applications, such as data transfer over a single data line shared by several sending devices (see [Figure 21-26](#)), 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.

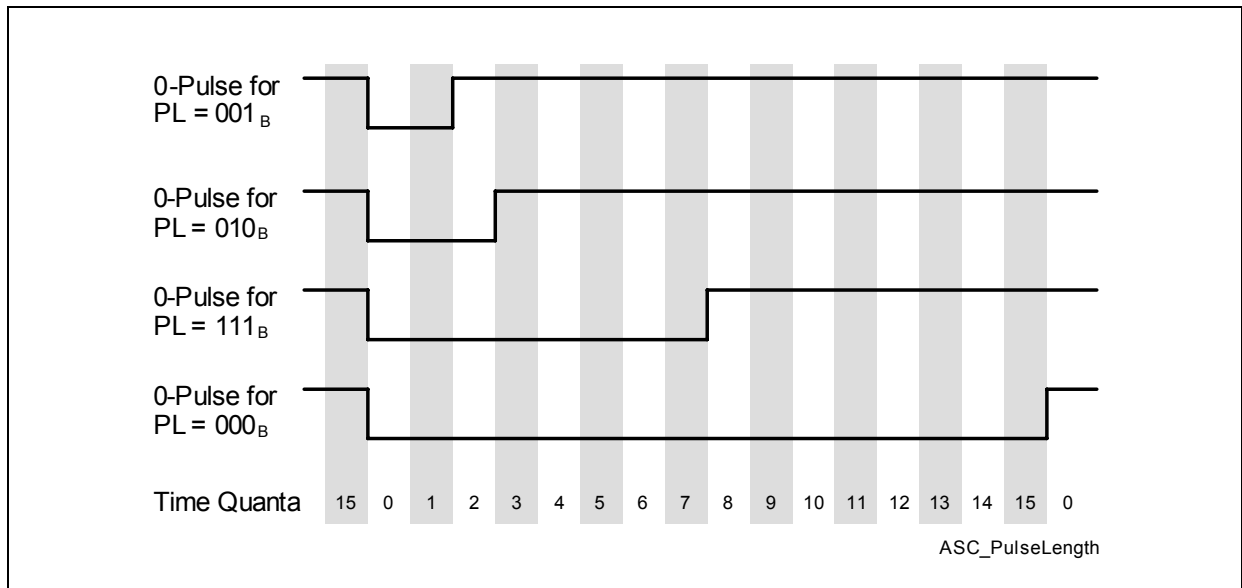
The data value read at the TXD input at the DX1 stage and the transmitted data bit value are compared after the sampling of each bit value. If enabled by PCRL.CDEN = 1 and a bit sent is not equal to the bit read back, a collision is detected and bit PSR.COL is set. If enabled, bit PSR.COL = 1 disables 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.

### 21.3.3.5 Pulse Shaping

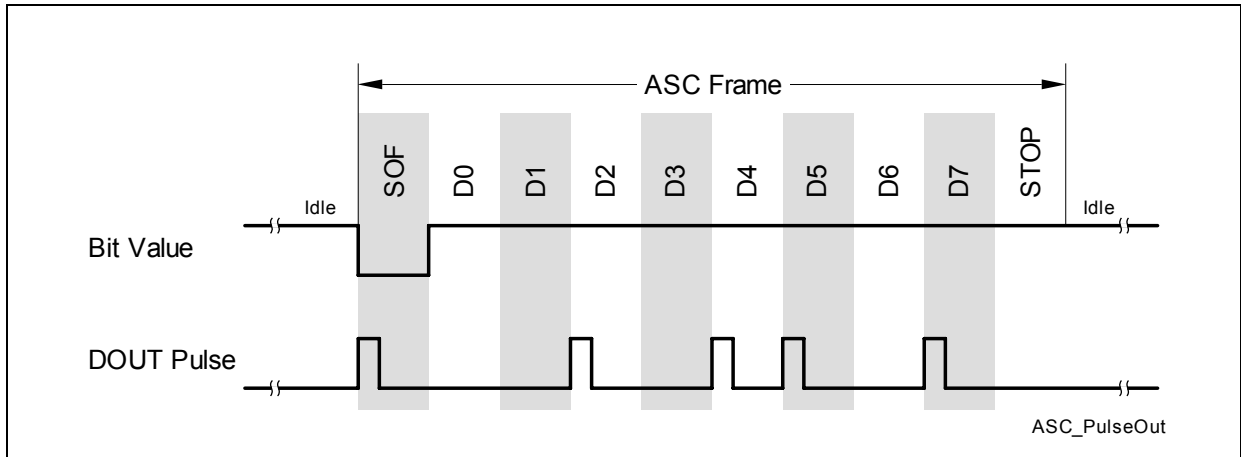
For some applications, the 0 level of transmitted bits with the bit value 0 is not applied at the transmit output during the complete bit time. Instead of driving the original 0 level, only a 0 pulse is generated and the remaining time quanta of the bit time are driven with 1 level. The length of a bit 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  $PCR.H.PL = 000_B$ ). In the case  $PCR.H.PL > 000_B$ , the transmit output signal becomes 0 for the number of time quanta defined by  $PCR.H.PL$ . 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 21-29 Transmitter Pulse Length Control**

**Figure 21-30** 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  $SCTRL.DOCFG = 01_B$ .



**Figure 21-30 Pulse Output Example**

### 21.3.3.6 Automatic Shadow Mechanism

The contents of the protocol control registers PCRL and PCRH, as well as bit field SCTR.H.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.H.WLE and SCTRL.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 SCTRL.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.H.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.H.WLE can be modified while a data transfer is in progress after the RSI event has been detected.

### 21.3.3.7 End of Frame Control

The number of bits per ASC frame is defined by bit field SCTR.H.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 TCSRL.FLEMD = 1 (in this case, bits TCSRL.WLEMD, SELMD, and WAMD have to be cleared).

If enabled, the transmit control information TCI automatically overwrites the bit field TCSRL.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).

### **21.3.3.8 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.

### **21.3.3.9 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.

### **21.3.3.10 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 (DOUT) 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 21-116](#).
- 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 DOUT 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



## **Universal Serial Interface Channel**

PSR.RFF becomes set at the end of the last stop bit. The DIN 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.

### **21.3.3.11 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 TCSRL.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.

### **21.3.3.12 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.

### **21.3.3.13 Receive Buffer Handling**

If a receive FIFO buffer is available (CCFG.RB = 1) and enabled for data handling (RBCTRH.SIZE > 0), it is recommended to set RBCTRH.RCIM = 11<sub>B</sub> 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 OUTRH.RCI[0] = 1, a parity error is indicated by OUTRH.RCI[4] = 1, and the received parity bit value is given by OUTRH.RCI[3].

The standard receive buffer event and the alternative receive buffer event can be used for the following operations in RCI mode (RBCTRH.RNM = 1):

- A standard receive buffer event indicates that a data word can be read from OUTRL that has been received without parity error.
- An alternative receive buffer event indicates that a data word can be read from OUTRL that has been received with parity error.

### **21.3.3.14 Sync-Break Detection**

The receiver permanently checks the DIN signal for a certain number of consecutive bit times with 0 level. The number is given by the number of programmed bits per frame (SCTRH.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 PCRL.SBD = 1). The counting restarts from 0 each time a falling edge is found at input DIN. This feature can be used for the detection of a synchronization break for slave devices in a LIN bus system (the master doesn't check for sync break).

For example, in a configuration for 8 data bits without parity generation, bit PCRL.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).

### **21.3.3.15 Transfer Status Indication**

The receiver status can be monitored by flag PSR[9] = BUSY if bit PCRH.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.

### **Universal Serial Interface Channel**

The transmitter status can be monitored by flag PSR[9] = BUSY if bit PCRH.CTR[17] (transmitter status enable TSTEN) is set. In this case, bit BUSY is set during a complete frame transmission from the beginning of the start of frame bit to the end of the last stop bit.

If both bits RSTEN and TSTEN are set, flag BUSY indicates the logical OR-combination of the receiver and the transmitter status. If both bits are cleared, flag BUSY is not modified depending on the transfer status (status changes are ignored).

## 21.3.4 ASC Protocol Registers

In ASC mode, the registers PCRH, PCRL and PSR handle ASC related information.

### 21.3.4.1 ASC Protocol Control Registers

In ASC mode, the PCRL/PCRH register bits or bit fields are defined as described in this section.

#### PCRL

##### Protocol Control Register L [ASC Mode]

(40<sub>H</sub>)

Reset Value: 0000<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PL			SP				FFI EN	FEI EN	RNI EN	CD EN	SBI EN	IDM	STP B	SMD	
rw			rw				rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
<b>SMD</b>	0	rw	<b>Sample Mode</b> This bit field defines the sample mode of the ASC receiver. The selected data input signal can be sampled only once per bit time or three times (in consecutive time quanta). When sampling three times, the bit value shifted in the receiver shift register is given by a majority decision among the three sampled values. 0 <sub>B</sub> Only one sample is taken per bit time. The current input value is sampled. 1 <sub>B</sub> Three samples are taken per bit time and a majority decision is made.
<b>STPB</b>	1	rw	<b>Stop Bits</b> This bit defines the number of stop bits in an ASC frame. 0 <sub>B</sub> The number of stop bits is 1. 1 <sub>B</sub> The number of stop bits is 2.

**Universal Serial Interface Channel**

Field	Bits	Type	Description
<b>IDM</b>	2	rw	<b>Idle Detection Mode</b> This bit defines if the idle detection is switched off or based on the frame length. 0 <sub>B</sub> The bus idle detection is switched off and bits PSR.TXIDLE and PSR.RXIDLE are set automatically to enable data transfers without checking the inputs before. 1 <sub>B</sub> The bus is considered as idle after a number of consecutive passive bit times defined by SCTRH.FLE plus 2 (in the case without parity bit) or plus 3 (in the case with parity bit).
<b>SBIEN</b>	3	rw	<b>Synchronization Break Interrupt Enable</b> This bit enables the generation of a protocol interrupt if a synchronization break is detected. The automatic detection is always active, so bit SBD can be set independently of SBIEN. 0 <sub>B</sub> The interrupt generation is disabled. 1 <sub>B</sub> The interrupt generation is enabled.
<b>CDEN</b>	4	rw	<b>Collision Detection Enable</b> This bit enables the reaction of a transmitter to the collision detection. 0 <sub>B</sub> The collision detection is disabled. 1 <sub>B</sub> If a collision is detected, the transmitter stops its data transmission, outputs a 1, sets bit PSR.COL and generates a protocol interrupt. In order to allow data transmission again, PSR.COL has to be cleared by software.
<b>RNIEN</b>	5	rw	<b>Receiver Noise Detection Interrupt Enable</b> 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 PCRL.RNIEN. 0 <sub>B</sub> The interrupt generation is disabled. 1 <sub>B</sub> The interrupt generation is enabled.

**Universal Serial Interface Channel**

Field	Bits	Type	Description
<b>FEIEN</b>	6	rw	<b>Format Error Interrupt Enable</b> 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 PCRL.FEIEN. 0 <sub>B</sub> The interrupt generation is disabled. 1 <sub>B</sub> The interrupt generation is enabled.
<b>FFIEN</b>	7	rw	<b>Frame Finished Interrupt Enable</b> 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 PCRL.FFIEN. 0 <sub>B</sub> The interrupt generation is disabled. 1 <sub>B</sub> The interrupt generation is enabled.
<b>SP</b>	[12:8]	rw	<b>Sample Point</b> This bit field defines the sample point of the bit value. The sample point must not be located outside the programmed bit timing ( $PCRL.SP \leq BRGL.DCTQ$ ).
<b>PL</b>	[15:13]	rw	<b>Pulse Length</b> This bit field defines the length of a 0 data bit, counted in time quanta, starting with the time quantum 0 of each bit time. Each bit value that is a 0 can lead to a 0 pulse that is shorter than a bit time, e.g. for IrDA applications. The length of a bit time is not changed by PL, only the length of the 0 at the output signal. The pulse length must not be longer than the programmed bit timing ( $PCRH.PL \leq BRGL.DCTQ$ ). This bit field is only taken into account by the transmitter and is ignored by the receiver. 000 <sub>B</sub> The pulse length is equal to the bit length (no shortened 0). 001 <sub>B</sub> The pulse length of a 0 bit is 2 time quanta. 010 <sub>B</sub> The pulse length of a 0 bit is 3 time quanta. ... 111 <sub>B</sub> The pulse length of a 0 bit is 8 time quanta.

**PCRH**

**Protocol Control Register H [ASC Mode]**

**(42<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>M CLK</b>							<b>0</b>							<b>TST EN</b>	<b>RST EN</b>
rw							r							rw	rw

Field	Bits	Type	Description
<b>RSTEN</b>	0	rw	<b>Receiver Status Enable</b> This bit enables the modification of flag PSR[9] = BUSY according to the receiver status. 0 <sub>B</sub> Flag PSR[9] is not modified depending on the receiver status. 1 <sub>B</sub> Flag PSR[9] is set during the complete reception of a frame.
<b>TSTEN</b>	1	rw	<b>Transmitter Status Enable</b> This bit enables the modification of flag PSR[9] = BUSY according to the transmitter status. 0 <sub>B</sub> Flag PSR[9] is not modified depending on the transmitter status. 1 <sub>B</sub> Flag PSR[9] is set during the complete transmission of a frame.
<b>0</b>	[14:2]	r	<b>Reserved</b> Returns 0 if read; not modified in ASC mode.
<b>MCLK</b>	15	rw	<b>Master Clock Enable</b> This bit enables the generation of the master clock MCLK. 0 <sub>B</sub> The MCLK generation is disabled and the MCLK signal is 0. 1 <sub>B</sub> The MCLK generation is enabled.

### 21.3.4.2 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 doesn't lead to further actions (no interrupt generation). Writing a 0 has no effect. The PSR flags should be cleared by software before enabling a new protocol.

#### PSR

#### Protocol Status Register [ASC Mode] (44<sub>H</sub>)

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>AIF</b>	<b>RIF</b>	<b>TBIF</b>	<b>TSIF</b>	<b>DLIF</b>	<b>RSIF</b>	<b>BU SY</b>	<b>TFF</b>	<b>RFF</b>	<b>FER 1</b>	<b>FER 0</b>	<b>RNS</b>	<b>COL</b>	<b>SBD</b>	<b>RX IDLE</b>	<b>TX IDLE</b>
rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
<b>TXIDLE</b>	0	rwh	<b>Transmission Idle</b> This bit shows if the transmit line (DX1) has been idle. A frame transmission can only be started if TXIDLE is set. 0 <sub>B</sub> The transmitter line has not yet been idle. 1 <sub>B</sub> The transmitter line has been idle and frame transmission is possible.
<b>RXIDLE</b>	1	rwh	<b>Reception Idle</b> This bit shows if the receive line (DX0) has been idle. A frame reception can only be started if RXIDLE is set. 0 <sub>B</sub> The receiver line has not yet been idle. 1 <sub>B</sub> The receiver line has been idle and frame reception is possible.
<b>SBD</b>	2	rwh	<b>Synchronization Break Detected<sup>1)</sup></b> This bit is set if a programmed number of consecutive bit values with level 0 has been detected (called synchronization break, e.g. in a LIN bus system). 0 <sub>B</sub> A synchronization break has not yet been detected. 1 <sub>B</sub> A synchronization break has been detected.



**Universal Serial Interface Channel**

Field	Bits	Type	Description
<b>COL</b>	3	rwh	<b>Collision Detected<sup>1)</sup></b> This bit is set if a collision has been detected (with PCRL.CDEN = 1). 0 <sub>B</sub> A collision has not yet been detected and frame transmission is possible. 1 <sub>B</sub> A collision has been detected and frame transmission is not possible.
<b>RNS</b>	4	rwh	<b>Receiver Noise Detected<sup>1)</sup></b> This bit is set if receiver noise has been detected. 0 <sub>B</sub> Receiver noise has not been detected. 1 <sub>B</sub> Receiver noise has been detected.
<b>FER0</b>	5	rwh	<b>Format Error in Stop Bit 0<sup>1)</sup></b> This bit is set if a 0 has been sampled in the stop bit 0 (called format error 0). 0 <sub>B</sub> A format error 0 has not been detected. 1 <sub>B</sub> A format error 0 has been detected.
<b>FER1</b>	6	rwh	<b>Format Error in Stop Bit 1<sup>1)</sup></b> This bit is set if a 0 has been sampled in the stop bit 1 (called format error 1). 0 <sub>B</sub> A format error 1 has not been detected. 1 <sub>B</sub> A format error 1 has been detected.
<b>RFF</b>	7	rwh	<b>Receive Frame Finished<sup>1)</sup></b> This bit is set if the receiver has finished the last stop bit. 0 <sub>B</sub> The received frame is not yet finished. 1 <sub>B</sub> The received frame is finished.
<b>TFF</b>	8	rwh	<b>Transmitter Frame Finished<sup>1)</sup></b> This bit is set if the transmitter has finished the last stop bit. 0 <sub>B</sub> The transmitter frame is not yet finished. 1 <sub>B</sub> The transmitter frame is finished.
<b>BUSY</b>	9	r	<b>Transfer Status BUSY</b> This bit indicates the receiver status (if PCRH.RSTEN = 1) or the transmitter status (if PCRH.TSTEN = 1) or the logical OR combination of both (if PCRH.RSTEN = PCRH.TSTEN = 1). 0 <sub>B</sub> A data transfer does not take place. 1 <sub>B</sub> A data transfer currently takes place.
<b>RSIF</b>	10	rwh	<b>Receiver Start Indication Flag</b> 0 <sub>B</sub> A receiver start event has not occurred. 1 <sub>B</sub> A receiver start event has occurred.

Field	Bits	Type	Description
<b>DLIF</b>	11	rwh	<b>Data Lost Indication Flag</b> $0_B$ A data lost event has not occurred. $1_B$ A data lost event has occurred.
<b>TSIF</b>	12	rwh	<b>Transmit Shift Indication Flag</b> $0_B$ A transmit shift event has not occurred. $1_B$ A transmit shift event has occurred.
<b>TBIF</b>	13	rwh	<b>Transmit Buffer Indication Flag</b> $0_B$ A transmit buffer event has not occurred. $1_B$ A transmit buffer event has occurred.
<b>RIF</b>	14	rwh	<b>Receive Indication Flag</b> $0_B$ A receive event has not occurred. $1_B$ A receive event has occurred.
<b>AIF</b>	15	rwh	<b>Alternative Receive Indication Flag</b> $0_B$ An alternative receive event has not occurred. $1_B$ An alternative receive event has occurred.

1) This status bit can generate a protocol interrupt (see [Page 21-24](#)). The general interrupt status flags are described in the general interrupt chapter.

### 21.3.5 Hardware LIN Support

In order to support the LIN protocol, bit TCSRL.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 (SCTRH.FLE = 7<sub>H</sub>). For both, master and slave devices, the parity generation has to be switched off (CCR.PM = 00<sub>B</sub>) and transfers take place with LSB first (SCTRL.SDIR = 0) and 1 stop bit (PCRL.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 (or IN11) 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.

## Universal Serial Interface Channel

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  $BRGL.TMEN = 1$ ,  $DX0CR.CM = 10_H$  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  $BRGH.PDIV$ . 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 has to be disabled ( $BRGL.TMEN = 0$ ) and  $BRGH.PDIV$  can be programmed with the formerly captured value divided by twice the number of time quanta per bit (assuming  $BRGL.PCTQ = 00_B$ ).

In order to avoid a  $PDIV$  overflow during baud rate measurement, the prescaler settings of the fractional divider must be set in a way that leads to a target value of  $PDIV$  well below  $1024 / (2 \times \text{number of time quanta per bit time})$ . As this procedure leads to low  $PDIV$  target values, the baud rate measurement accuracy becomes limited. Therefore, the following procedure is recommended:

- Slowing down the fractional divider for baud rate measurement by  $2 \times \text{number of time quanta per bit time}$ .
- Writing the current value of  $FDRL.DM$  again to restart the fraction divider.
- Switching-on the baud rate measurement by writing  $BRGL.TMEN = 1$  (note that the synchronization break detection is not possible when baud rate measurement is enabled).
- Restoring the fractional divider to its original settings.
- Switching-off the baud rate measurement by writing  $BRGL.TMEN = 0$ .
- The measurement result in  $BRGH.PDIV$  can now be directly used as baud rate setting.

- 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, no transmission can take place by the ASC transmitter of the same USIC channel.

## 21.4 Synchronous Serial Channel (SSC)

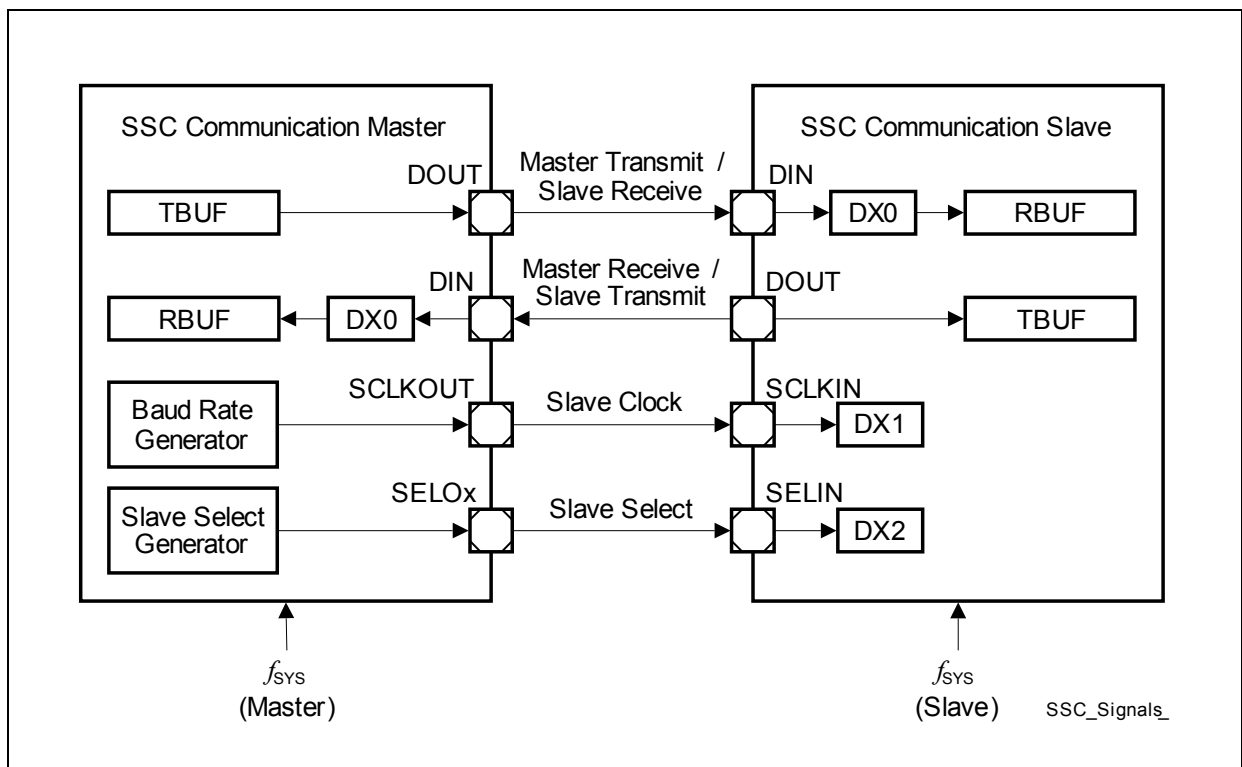
The synchronous serial channel SSC covers the data transfer function of an SPI-like module. It can handle reception and transmission of synchronous data frames between a device operating in master mode and at least one device in slave mode. The SSC mode is selected by  $CCR.MODE = 0001_B$  with  $CCFG.SSC = 1$  (SSC mode is available).

This chapter contains the following sections:

- Signal description (see [Page 21-131](#))
- General SSC issues (see [Page 21-139](#))
- Master mode operation (see [Page 21-143](#))
- Slave mode operation (see [Page 21-150](#))
- Protocol registers (see [Page 21-152](#))
- Timing considerations (see [Page 21-158](#))

### 21.4.1 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 21-31 SSC Signals for 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

### Universal Serial Interface Channel

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 SELO<sub>x</sub>, with x = 0-7) and 1 slave select input SELIN for slave mode. In most applications, the slave select signals are active low.

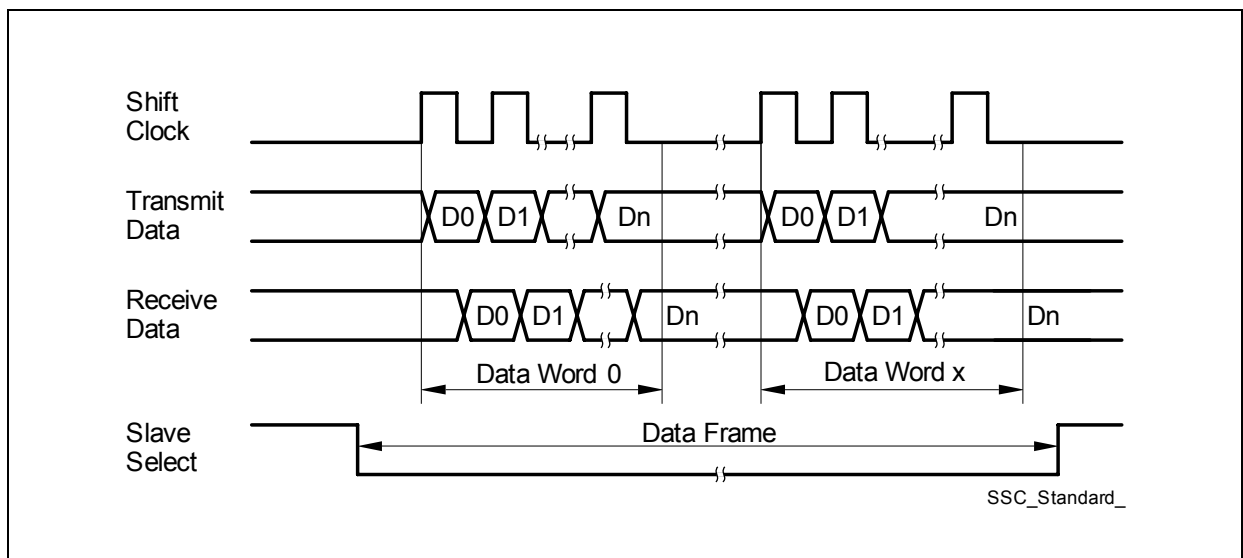
A device operating in master mode controls the start and end of a data frame, as well as the generation of the shift clock and slave select signals. This comprises the baud rate setting for the shift clock and the delays between the shift clock and the slave select output signals. If several SSC modules 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 DX0, DX1, and DX2 please refer to [Page 21-36](#).

**Table 21-8 SSC Communication Signals**

SSC Mode	Receive Data	Transmit Data	Shift Clock	Slave Select(s)
Master	MRST <sup>1)</sup> , input DIN, handled by DX0	MTSR <sup>2)</sup> , Output DOUT	Output SCLKOUT	Output(s) SELO <sub>x</sub>
Slave	MTSR, input DIN, handled by DX0	MRST, Output DOUT	Input SCLKIN, handled by DX1	input SELIN, handled by DX2

1) MRST = master receive slave transmit, also known as MISO = master in slave out

2) MTSR = master transmit slave receive, also known as MOSI = master out slave in

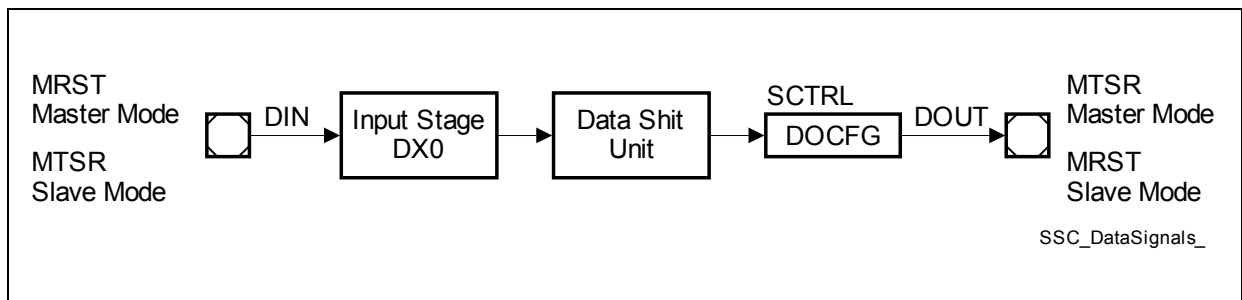


**Figure 21-32 4-Wire SSC Standard Communication Signals**

### 21.4.1.1 Transmit and Receive Data Signals

In 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 or by enabling/disabling push/pull output drivers). 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 21-31](#).

The receive data input signal DIN 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 DIN is taken from an input pin. The signal polarity of DOUT (data output) with respect to the data bit value can be configured in block DOCFG (data output configuration) by bit field SCTRL.DOCFG.

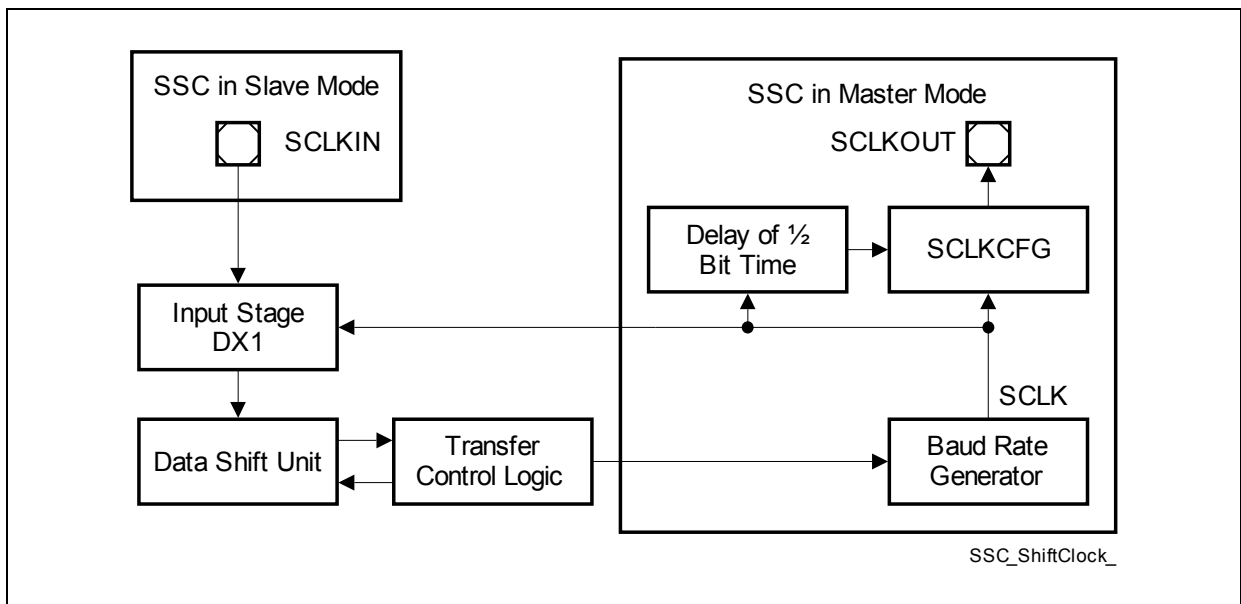


**Figure 21-33 SSC Data Signals**

### 21.4.1.2 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.



**Figure 21-34 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 BRGH.SCLKCFG, allowing 4 possible settings, as shown in [Figure 21-35](#).

- No delay, no polarity inversion (SCLKCFG = 00<sub>B</sub>, 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<sub>B</sub>):  
 The inactive level of SCLKOUT is 1, while no data frame is transferred. The first data

## **Universal Serial Interface Channel**

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.

- SCLKOUT is delayed by 1/2 shift clock period, no polarity inversion (SCLKCFG = 10<sub>B</sub>):

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).

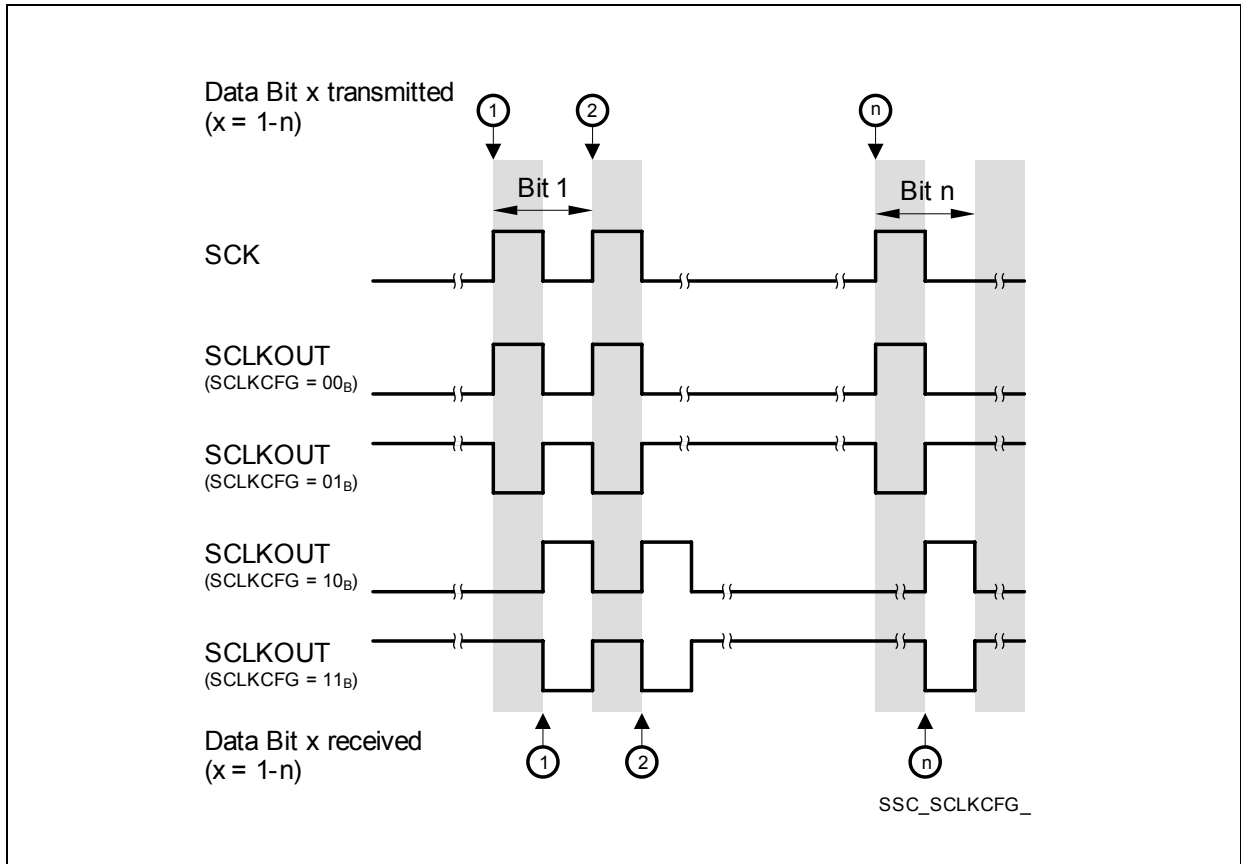
- SCLKOUT is delayed by 1/2 shift clock period, polarity inversion (SCLKCFG = 11<sub>B</sub>):

The inactive level of SCLKOUT is 1, while no data frame is transferred.

The first data bit of a new data frame is transmitted 1/2 shift clock period before the first falling clock edge of SCLKOUT. Due to the delay, the next data bits seem to be transmitted with the rising edges of SCLKOUT. The last data bit of a data frame is transmitted 1/2 period of SCLKOUT before the last falling clock edge of SCLKOUT. The first data bit is received 1/2 shift clock period before the first rising edge of SCLKOUT. Due to the delay, the next data bits seem to be received with the falling edges of SCLKOUT. The last data bit is received 1/2 period of SCLKOUT before the last rising clock edge of SCLKOUT.

This setting can be used only in master mode and not in slave mode (the connected slave has to provide the first data bit before the first SCLKOUT edge, e.g. as soon as it is addressed by its slave select).





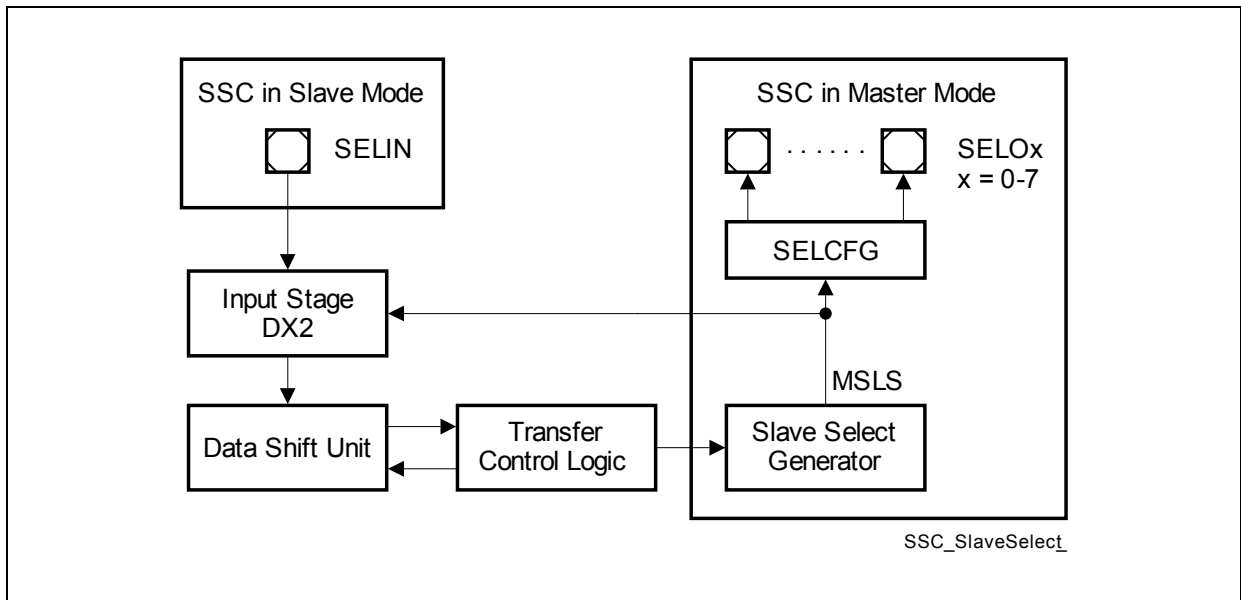
**Figure 21-35 SCLKOUT Configuration in SSC Master Mode**

*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.*

### 21.4.1.3 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 SELO<sub>x</sub> output signals that can be configured by the block SELCFG (select configuration).



**Figure 21-36 SSC Slave Select Signals**

The control of the SELCFG block is based on protocol specific bits and bit fields in the protocol control register parts PCRL and PCRH. For the generation of the MSLS signal please refer to [Section 21.4.3.2](#).

- PCRL.SELCTR to chose between direct and coded select mode
- PCRL.SELINV to invert the SELO<sub>x</sub> outputs
- PCRH.SELO[7:0] as individual value for each SELO<sub>x</sub> line

The SELCFG block supports the following configurations of the SELO<sub>x</sub> output signals:

- Direct Select Mode (SELCTR = 1):  
 Each SELO<sub>x</sub> line (with  $x = 0-7$ ) can be directly connected to an external slave device. If bit  $x$  in bit field SELO is 0, the SELO<sub>x</sub> output is permanently inactive. A SELO<sub>x</sub> output becomes active while the internal signal MSLS is active (see

## **Universal Serial Interface Channel**

**Section 21.4.3.2**) 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  $f_{SYS}$  compared to MSLS to allow the other SELOx lines to stabilize before enabling the address decoder.

## **21.4.2 Operating the SSC**

This chapter contains SSC issues, that are of general interest and not directly linked to either master mode or slave mode.

### **21.4.2.1 Automatic Shadow Mechanism**

The contents of the baud rate control registers BRGL and BRGH, bit field SCTR.H.FLE as well as the protocol control registers PCRL and PCRH 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.H.WLE and SCTRL.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 SCTRL.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 SCTRL.WLE. If TCSRL.WLEMD = 1, it is recommended to update TCSRL and TBUFxx after the receiver start interrupt has been generated.

### **21.4.2.2 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 TCSRL.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.

### **21.4.2.3 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.ACK = 1, the SSC mode can be disabled.

### **21.4.2.4 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 SCTRL.WLE) is evaluated for each new data word, whereas the frame length information (defined by SCTRL.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.

### **21.4.2.5 Parity Mode**

Parity generation is not supported in SSC mode and bit field CCR.PM = 00<sub>B</sub> has to be programmed.

### **21.4.2.6 Transfer Mode**

In SSC mode, bit field SCTRL.TRM = 01<sub>B</sub> has to be programmed to allow data transfers. Setting SCTRL.TRM = 00<sub>B</sub> disables and stops the data transfer immediately.

### **21.4.2.7 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 TCSRL.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 RBUFSSR.SOF = 0. Bit PSR.RIF is set after the reception of the last data bit of a data word if RBUFSSR.SOF = 0.  
Bit RBUFSSR.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 RBUFSSR.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.

### **21.4.2.8 Protocol-Related Argument and Error**

The protocol-related argument (RBUFSSR.PAR) and the protocol-related error (RBUFSSR.PERR) are two flags that are assigned to each received data word in the corresponding receiver buffer status registers.

In SSC mode, RBUFSSR.PAR is always 0. The received start of frame indication is monitored by the protocol-related error indication RBUFSSR.PERR (0 = received word is not the first word of a frame, 1 = received word is the first word of a new frame).

### **21.4.2.9 Receive Buffer Handling**

If a receive FIFO buffer is available (CCFG.RB = 1) and enabled for data handling (RBCTRH.SIZE > 0), it is recommended to set RBCTRH.RCIM = 01<sub>B</sub> 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 OUTRH.RCI[4] = 1, and the word length of the received data is given by OUTRH.RCI[3:0].

The standard receive buffer event and the alternative receive buffer event can be used for the following operation in RCI mode (RBCTRH.RNM = 1):

- A standard receive buffer event indicates that a data word can be read from OUTRL that has not been the first word of a data frame.

**Universal Serial Interface Channel**

- An alternative receive buffer event indicates that the first data word of a new data frame can be read from OUTRL.

### **21.4.3 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  $SCTRL.TRM = 01_B$  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 input stage DX0 with the selected receive data input pin (DIN) with  $DX0CR.INSW = 1$  and configure a transmit data output pin (DOUT).
- **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  $PCRL.MSLSEN = 1$  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, and the shift direction have to be set up according to the application requirements by programming the registers SCTRL and SCTRLH.

*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.*

#### **21.4.3.1 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 ( $BRGL.PPPEN = 0$ ). 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 ( $BRGL.PPPEN = 1$ ).

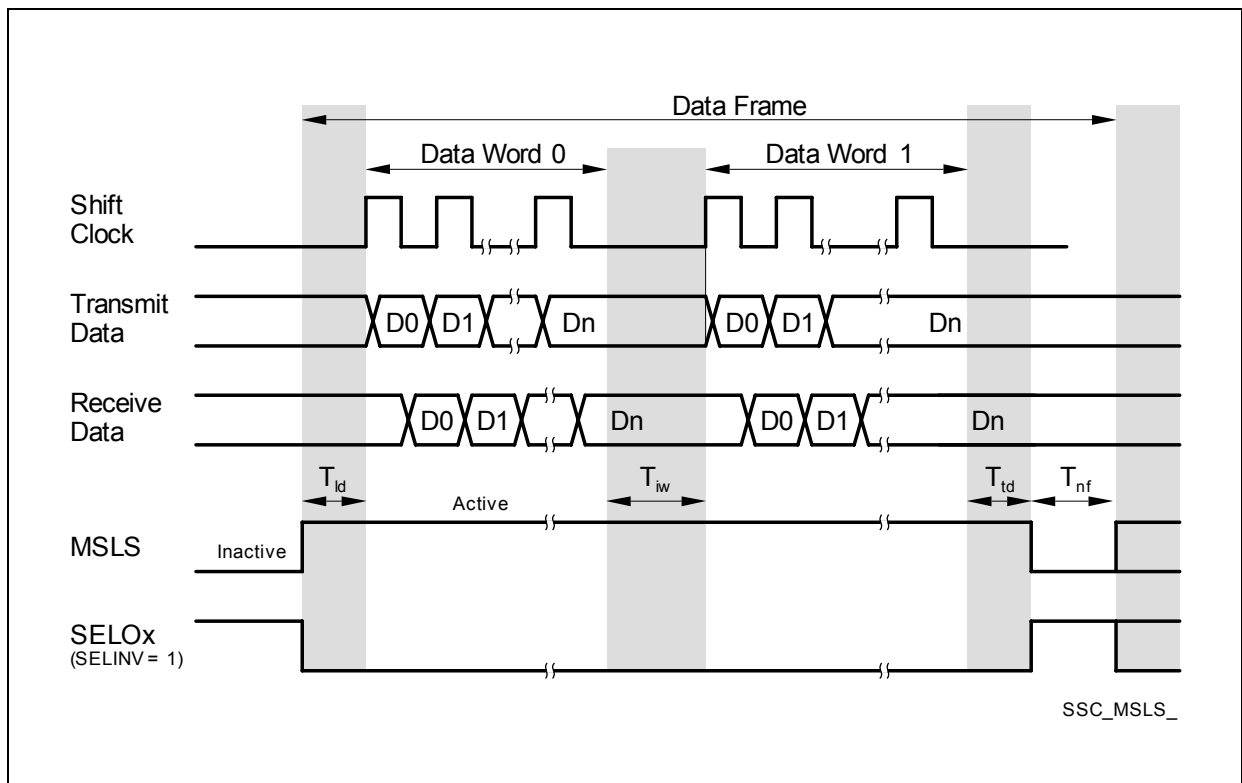


The adjustable divider factor is defined by bit field BRGH.PDIV.

$$\begin{aligned}
 f_{\text{SCLK}} &= \frac{f_{\text{PIN}}}{2} \times \frac{1}{\text{PDIV} + 1} && \text{if } \text{PPPEN} = 0 \\
 f_{\text{SCLK}} &= \frac{f_{\text{PIN}}}{2 \times 2} \times \frac{1}{\text{PDIV} + 1} && \text{if } \text{PPPEN} = 1
 \end{aligned}
 \tag{21.8}$$

### 21.4.3.2 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  $T_{\text{ld}}$ ), 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  $T_{\text{td}}$ ) 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  $T_{\text{nf}}$ ). 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  $T_{\text{iw}}$ ).



**Figure 21-37 MSLS Generation in SSC Master Mode**

In SSC master mode, the slave select delays are defined as follows:

- Leading delay  $T_{ld}$ :  
The leading delay starts if valid data is available for transmission. The internal signal MSLS becomes active with the start of the leading delay. The first shift clock edge (rising edge) of SCLK is generated by the baud rate generator after the leading delay has elapsed.
- Trailing delay  $T_{td}$ :  
The trailing delay starts at the end of the last SCLK cycle of a data frame. The internal signal MSLS becomes inactive with the end of the trailing delay.
- Inter-word delay  $T_{iw}$ :  
This delay is optional and can be enabled/disabled by PCRH.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.

### **21.4.3.3 Automatic Slave Select Update**

If the number of bits per SSC frame and the word length are defined by bit fields SCTR.H.FLE and SCTR.H.WLE, the transmit control information TCI can be used to update the slave select setting PCRH.CTR[23:16] to control the SELOx select outputs. The automatic update mechanism is enabled by TCSRL.SELMD = 1 (bits TCSRL.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 21-118](#)).

#### 21.4.3.4 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.

- BRGL.CTQSEL  
to define the input frequency  $f_{CTQIN}$  for the time quanta generation for  $T_{ld}$  and  $T_{td}$
- BRGL.PCTQ  
to define the length of a time quantum (division of  $f_{CTQIN}$  by 1, 2, 3, or 4) for  $T_{ld}$  and  $T_{td}$
- BRGL.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.

- PCRL.CTQSEL1  
to define the input frequency  $f_{CTQIN1}$  for the time quanta generation for  $T_{nf}$  and  $T_{iw}$
- PCRL.PCTQ1  
to define the length of a time quantum (division of  $f_{CTQIN1}$  by 1, 2, 3, or 4) for  $T_{nf}$  and  $T_{iw}$
- PCRL.DCTQ1  
to define the number of time quanta for the delay generation for  $T_{nf}$  and  $T_{iw}$
- PCRH.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:

$$T_{ld} = T_{td} = \frac{(PCTQ + 1) \times (DCTQ + 1)}{f_{CTQIN}} \quad (21.9)$$

$$T_{iw} = T_{nf} = \frac{(PCTQ1 + 1) \times (DCTQ1 + 1)}{f_{CTQIN1}}$$

### **21.4.3.5 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 PCRL.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 PCRL.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.

### **21.4.3.6 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 `SCTRH.FLE < 63`), the following alternative mechanisms for end of frame handling are supported. It is recommended to set `SCTRH.FLE = 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 TCSRL.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 `TCSRL.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 TCSRL.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 `TCSRL.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 `TCSRL.WLEMD = 1`, the address of the written `TBUF[31:0]` is used as transmit control information `TCI[4:0]` to update `SCTRH.WLE (= TCI[3:0])` and `TCSRL.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 `TCSRL.WLEMD = 1`, bits `TCSRL.EOF` and `SOF`, as well as `SCTRH.WLE` must not be written by software after writing data to a TBUF location. Furthermore, it is recommended to clear bits `TCSRL.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

**Universal Serial Interface Channel**

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 PCRL.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 (TCSRL.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 PCRL.FEM = 1, signal MSLS stays active while the transmit buffer is waiting for new data (TCSRL.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  $T_{nf}$ .

#### **21.4.4 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  $SCTRL.TRM = 01_B$  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 a connection of input stage DX0 with the selected receive data input pin (signal DIN) with  $DX0CR.INSW = 1$  and configure a transmit data output pin (signal DOUT).  
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 DOUT signal outputs the passive data level defined by  $SCTRL.PDL$ .
- **Baud rate generation:**  
The baud rate generator is not needed and can be switched off by the fractional divider.
- **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 registers  $PCRL/PCRH$  are not necessary and can be programmed to 0.

##### **21.4.4.1 Protocol Interrupts**

The following protocol-related events generated in SSC mode and can lead to a protocol interrupt. They are related to the start and the end of a data frame. After the start of a data frame a new setting could be programmed for the next data frame and after the end of a data frame the SSC connections to pins can be changed.

Please note that the bits in register PSR are not all automatically cleared by hardware and have to be cleared by software in order to monitor new incoming events.

- **MSLS event:**  
The MSLS generation being switched off, this event is not available.
- **DX2T event:**  
The slave select input signal SELIN is handled by the DX2 stage and the edges of the selected signal can generate a protocol interrupt. This interrupt allows to indicate

that a data frame has started and/or that a data frame has been completely finished. A programmable edge detection for the DX2 input signal activates DX2T, sets bit PSR.DX2TEV and additionally, a protocol interrupt can be generated if PCRL.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.

#### **21.4.4.2 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.H.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.H.FLE can be either programmed to the word length SCTR.H.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.H.FLE should be programmed to its maximum value to disable the slave internal frame length evaluation.



## 21.4.5 SSC Protocol Registers

In SSC mode, the registers PCRL, PCRH and PSR handle SSC related information.

### 21.4.5.1 SSC Protocol Control Registers

In SSC mode, the PCRL/PCRH register bits or bit fields are defined as described in this section.

#### PCRL

##### Protocol Control Register L [SSC Mode]

(40<sub>H</sub>)

Reset Value: 0000<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>DX2 TIEN</b>	<b>MSL SIEN</b>	<b>0</b>	<b>DCTQ1</b>				<b>PCTQ1</b>		<b>CTQSEL1</b>		<b>FEM</b>	<b>SE INV</b>	<b>SEL CTR</b>	<b>MSL SEN</b>	
rw		rw	rw				rw		rw		rw	rw	rw	rw	rw

Field	Bits	Type	Description
<b>MSLSEN</b>	0	rw	<b>MSLS Enable</b> This bit enables/disables the generation of the master slave select signal MSLS. If the SSC is a transfer slave, the SLS information is read from a pin and the internal generation is not needed. If the SSC is a transfer master, it has to provide the MSLS signal. 0 <sub>B</sub> The MSLS generation is disabled (MSLS = 0). This is the setting for SSC slave mode. 1 <sub>B</sub> The MSLS generation is enabled. This is the setting for SSC master mode.
<b>SELCTR</b>	1	rw	<b>Select Control</b> This bit selects the operating mode for the SELO[7:0] outputs. 0 <sub>B</sub> The coded select mode is enabled. 1 <sub>B</sub> The direct select mode is enabled.
<b>SELINV</b>	2	rw	<b>Select Inversion</b> This bit defines if the polarity of the SELO[7:0] outputs in relation to the master slave select signal MSLS. 0 <sub>B</sub> The SELO outputs have the same polarity as the MSLS signal (active high). 1 <sub>B</sub> The SELO outputs have the inverted polarity to the MSLS signal (active low).

**Universal Serial Interface Channel**

Field	Bits	Type	Description
<b>FEM</b>	3	rw	<b>Frame End Mode</b> This bit defines if a transmit buffer content that is not valid for transmission is considered as an end of frame condition for the slave select generation. $0_B$ The current data frame is considered as finished when the last bit of a data word has been sent out and the transmit buffer TBUF does not contain new data (TDV = 0). $1_B$ The MSLS signal is kept active also while no new data is available and no other end of frame condition is reached. In this case, the software can accept delays in delivering the data without automatic deactivation of MSLS in multi-word data frames.
<b>CTQSEL1</b>	[5:4]	rw	<b>Input Frequency Selection</b> This bit field defines the input frequency $f_{CTQIN}$ for the generation of the slave select delays $T_{iw}$ and $T_{nf}$ . $00_B$ $f_{CTQIN} = f_{PDIV}$ $01_B$ $f_{CTQIN} = f_{PPP}$ $10_B$ $f_{CTQIN} = f_{SCLK}$ $11_B$ $f_{CTQIN} = f_{MCLK}$
<b>PCTQ1</b>	[7:6]	rw	<b>Divider Factor PCTQ1 for <math>T_{iw}</math> and <math>T_{nf}</math></b> This bit field represents the divider factor PCTQ1 (range = 0 - 3) for the generation of the inter-word delay and the next-frame delay. $T_{iw} = T_{nf} = 1/f_{CTQIN} \times (PCTQ1 + 1) \times (DCTQ1 + 1)$
<b>DCTQ1</b>	[12:8]	rw	<b>Divider Factor DCTQ1 for <math>T_{iw}</math> and <math>T_{nf}</math></b> This bit field represents the divider factor DCTQ1 (range = 0 - 31) for the generation of the inter-word delay and the next-frame delay. $T_{iw} = T_{nf} = 1/f_{CTQIN} \times (PCTQ1 + 1) \times (DCTQ1 + 1)$
<b>MSLSIEN</b>	14	rw	<b>MSLS Interrupt Enable</b> This bit enables/disables the generation of a protocol interrupt if the state of the MSLS signal changes (indicated by PSR.MSLSEV = 1). $0_B$ A protocol interrupt is not generated if a change of signal MSLS is detected. $1_B$ A protocol interrupt is generated if a change of signal MSLS is detected.

**Universal Serial Interface Channel**

Field	Bits	Type	Description
<b>DX2TIEN</b>	15	rw	<b>DX2T Interrupt Enable</b> This bit enables/disables the generation of a protocol interrupt if the DX2T signal becomes activated (indicated by PSR.DX2TEV = 1). $0_B$ A protocol interrupt is not generated if DX2T is activated. $1_B$ A protocol interrupt is generated if DX2T is activated.
<b>0</b>	13	rw	<b>Reserved</b> Returns 0 if read; should be written with 0.

**PCRH**

**Protocol Control Register H [SSC Mode]**

**(42<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>M CLK</b>			<b>0</b>				<b>TIW EN</b>				<b>SELO</b>				
rw			rw				rw				rw				

Field	Bits	Type	Description
<b>SELO</b>	[7:0]	rw	<b>Select Output</b> This bit field defines the setting of the SELO[7:0] output lines. $0_B$ The corresponding SELOx line cannot be activated. $1_B$ The corresponding SELOx line can be activated (according to the mode selected by SELCTR).
<b>TIWEN</b>	8	rw	<b>Enable Inter-Word Delay <math>T_{iw}</math></b> This bit enables/disables the inter-word delay $T_{iw}$ after the transmission of a data word. $0_B$ No delay between data words of the same frame. $1_B$ The inter-word delay $T_{iw}$ is enabled and introduced between data words of the same frame.

**Universal Serial Interface Channel**

Field	Bits	Type	Description
<b>MCLK</b>	15	rw	<b>Master Clock Enable</b> This bit enables/disables the generation of the master clock output signal MCLK, independent from master or slave mode. 0 <sub>B</sub> The MCLK generation is disabled and output MCLK = 0. 1 <sub>B</sub> The MCLK generation is enabled.
<b>0</b>	[14:9]	rw	<b>Reserved</b> Returns 0 if read; not modified in SSC mode.

### 21.4.5.2 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 doesn't 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] (44<sub>H</sub>)

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>AIF</b>	<b>RIF</b>	<b>TBIF</b>	<b>TSIF</b>	<b>DLIF</b>	<b>RSIF</b>	<b>0</b>						<b>DX2 TEV</b>	<b>MSL SEV</b>	<b>DX2 S</b>	<b>MSL S</b>
rwh	rwh	rwh	rwh	rwh	rwh	r						rwh	rwh	rwh	rwh

Field	Bits	Type	Description
<b>MSLS</b>	0	rwh	<b>MSLS Status</b> This bit indicates the current status of the MSLS signal. It must be cleared by software to stop a running frame. 0 <sub>B</sub> The internal signal MSLS is inactive (0). 1 <sub>B</sub> The internal signal MSLS is active (1).
<b>DX2S</b>	1	rwh	<b>DX2S Status</b> This bit indicates the current status of the DX2S signal that can be used as slave select input SELIN. 0 <sub>B</sub> DX2S is 0. 1 <sub>B</sub> DX2S is 1.
<b>MSLSEV</b>	2	rwh	<b>MSLS Event Detected<sup>1)</sup></b> This bit indicates that the MSLS signal has changed its state since MSLSEV has been cleared. Together with the MSLS status bit, the activation/deactivation of the MSLS signal can be monitored. 0 <sub>B</sub> The MSLS signal has not changed its state. 1 <sub>B</sub> The MSLS signal has changed its state.
<b>DX2TEV</b>	3	rwh	<b>DX2T Event Detected<sup>1)</sup></b> This bit indicates that the DX2T trigger signal has been activated since DX2TEV has been cleared. 0 <sub>B</sub> The DX2T signal has not been activated. 1 <sub>B</sub> The DX2T signal has been activated.

**Universal Serial Interface Channel**

Field	Bits	Type	Description
<b>0</b>	[9:4]	r	<b>Reserved</b> Returns 0 if read; not modified in SSC mode.
<b>RSIF</b>	10	rwh	<b>Receiver Start Indication Flag</b> 0 <sub>B</sub> A receiver start event has not occurred. 1 <sub>B</sub> A receiver start event has occurred.
<b>DLIF</b>	11	rwh	<b>Data Lost Indication Flag</b> 0 <sub>B</sub> A data lost event has not occurred. 1 <sub>B</sub> A data lost event has occurred.
<b>TSIF</b>	12	rwh	<b>Transmit Shift Indication Flag</b> 0 <sub>B</sub> A transmit shift event has not occurred. 1 <sub>B</sub> A transmit shift event has occurred.
<b>TBIF</b>	13	rwh	<b>Transmit Buffer Indication Flag</b> 0 <sub>B</sub> A transmit buffer event has not occurred. 1 <sub>B</sub> A transmit buffer event has occurred.
<b>RIF</b>	14	rwh	<b>Receive Indication Flag</b> 0 <sub>B</sub> A receive event has not occurred. 1 <sub>B</sub> A receive event has occurred.
<b>AIF</b>	15	rwh	<b>Alternative Receive Indication Flag</b> 0 <sub>B</sub> An alternative receive event has not occurred. 1 <sub>B</sub> An alternative receive event has occurred.

- 1) This status bit can generate a protocol interrupt in SSC mode (see [Page 21-24](#)). The general interrupt status flags are described in the general interrupt chapter.

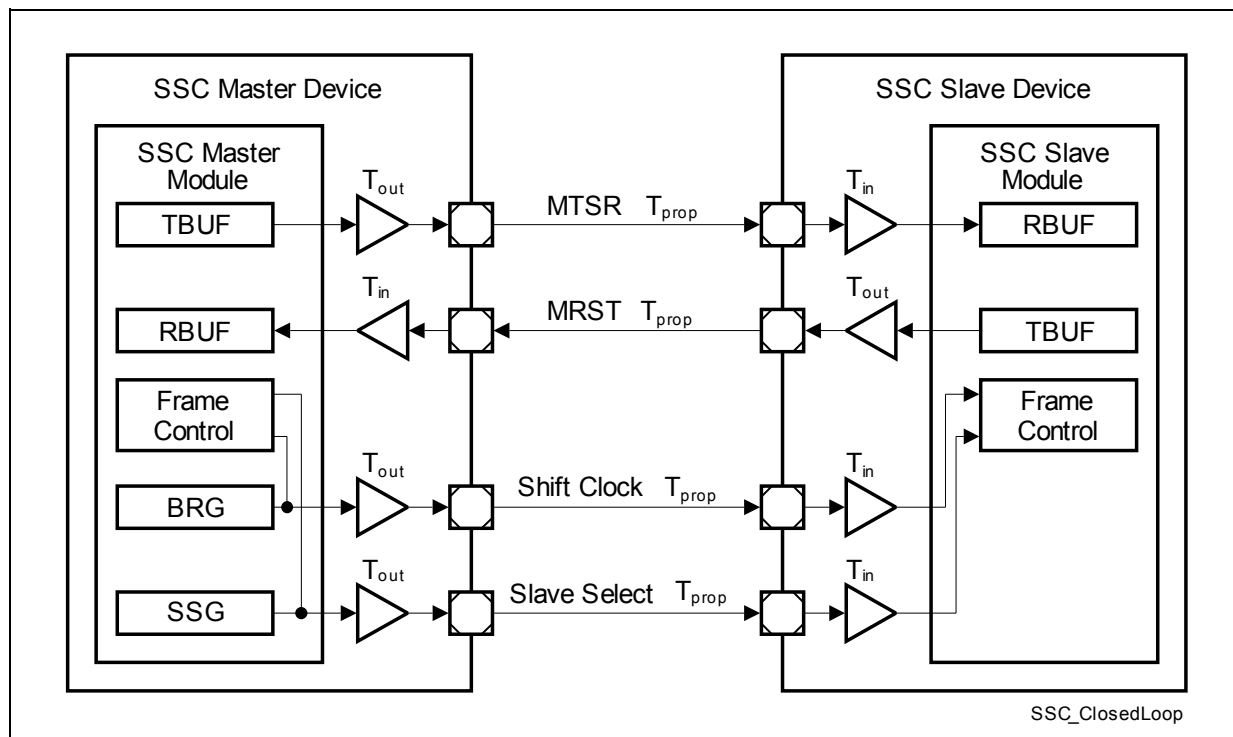
## 21.4.6 SSC Timing Considerations

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.

### 21.4.6.1 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 21-38 SSC Closed-loop Delay**

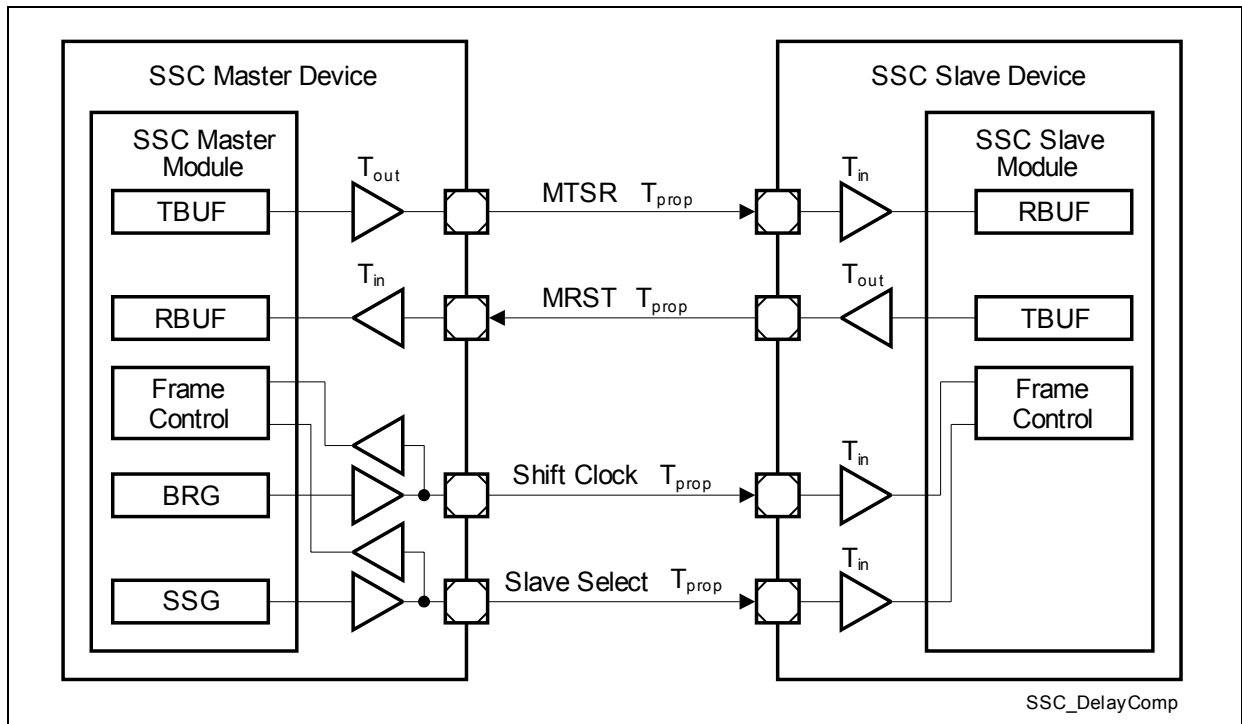
## Universal Serial Interface Channel

The signal path between the SSC modules of the master and the slave device includes the master's output driver, the wiring to the slave device and the slave device's input stage. With the received shift clock edges, the slave device receives the master's transmit data and transmits its own data back to the master device, passing by a similar signal path in the other direction. The master module receives the slave's transmit data related to its internal shift clock edges. In order to ensure correct data reception in the master device, the slave's transmit data has to be stable (respecting setup and hold times) as master receive data with the next shift clock edge of the master (generally 1/2 shift clock period). To avoid data corruption, the accumulated delays of the input and output stages, the signal propagation on the wiring and the reaction times of the transmitter/receiver have to be carefully considered, especially at high baud rates.

In the given example, the time between the generation of the shift clock signal and the evaluation of the receive data by the master SSC module is given by the sum of  $T_{out\_master} + 2 \times T_{prop} + T_{in\_slave} + T_{out\_slave} + T_{in\_master} + \text{module reaction times} + \text{input setup times}$ . 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.

### 21.4.6.2 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 21-39 SSC Master Mode with Delay Compensation**



**Universal Serial Interface Channel**

If the 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.

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 ( $DXnCR.INSW = 1$ ). 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.

## **21.5 Inter-IC Bus Protocol (IIC)**

The IIC protocol of the USIC refers to the IIC bus specification version 2.1, january 2000 from Philips Semiconductors. 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).

This chapter contains the following sections:

- Introduction (see [Page 21-161](#))
- Operating the IIC protocol (see [Page 21-165](#))
- Symbol timing and programming (see [Page 21-171](#))
- Data flow handling (see [Page 21-174](#))
- IIC protocol registers (see [Page 21-179](#))

### **21.5.1 Introduction**

USIC IIC Features:

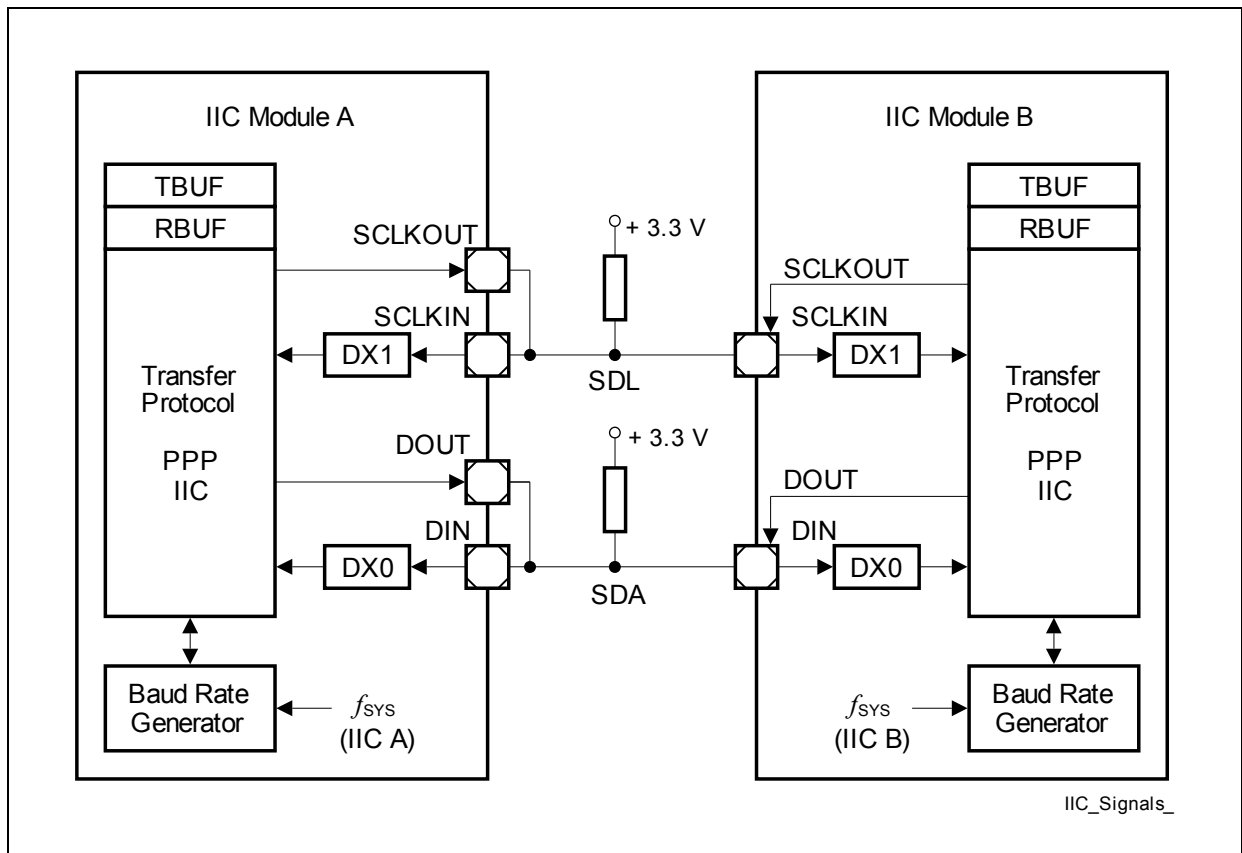
- Two-wire interface, with one line for shift clock transfer and synchronization (shift clock SCL), the other one for the data transfer (shift data SDA)
- Communication in standard mode (100 kBit/s) or in fast mode (up to 400 kBit/s)
- Support of 7-bit addressing, as well as 10-bit addressing
- Master mode operation,  
where the IIC controls the bus transactions and provides the clock signal.
- Slave mode operation,  
where an external master controls the bus transactions and provides the clock signal.
- Multi-master mode operation,  
where several masters can be connected to the bus and bus arbitration can take place, i.e. the IIC module can be master or slave. The master/slave operation of an IIC bus participant can change from frame to frame.
- Efficient frame handling (low software effort), also allowing PEC transfers
- Powerful interrupt handling due to multitude of indication flags
- Compensation support for input delays

### 21.5.1.1 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 DOUT
- Shift clock SCL: input handled by DX1 stage, output signal SCLKOUT

**Figure 21-25** 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 DOUT and a DX0 stage input for the same pin (similar for SCLKOUT and DX1). The pin assignment of module IIC B shows the connection of DOUT and a DX0 input at the same pin, also for SCLKOUT and a DX1 input.



**Figure 21-40 IIC Signal Connections**

### **21.5.1.2 Symbols**

A symbol is a sequence of edges on the lines SDA and SCL. Symbols contain 10 or 25 time quanta  $t_q$ , depending on the selected baud rate. The baud rate generator determines the length of the time quanta  $t_q$ , the sequence of edges in a symbol is handled by the IIC protocol pre-processor, and the sequence of symbols can be programmed by the user according to the application needs.

The following symbols are defined:

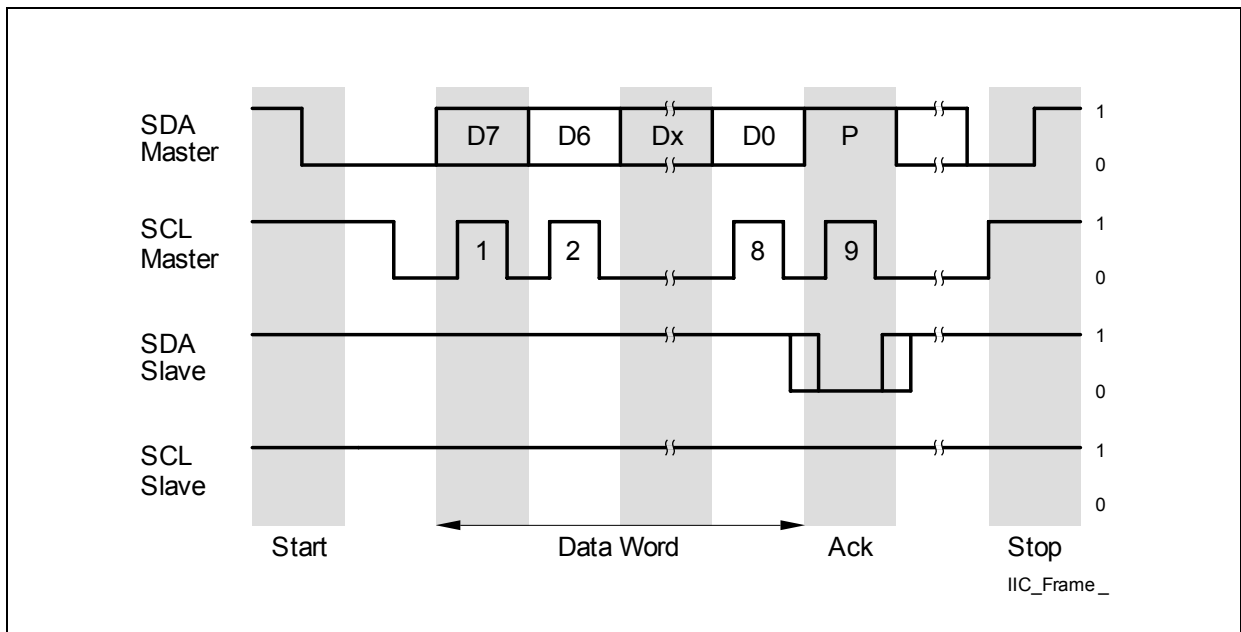
- Bus idle:  
SDA and SCL are high. No data transfer takes place currently.
- Data bit symbol:  
SDA stable during the high phase of SCL. SDA then represents the transferred bit value. There is one clock pulse on SCL for each transferred bit of data. During data transfers SDA may only change while SCL is low.
- Start symbol:  
Signal SDA being high followed by a falling edge of SDA while SCL is high indicates a start condition. This start condition initiates a data transfer over the IIC bus after the bus has been idle.
- Repeated start symbol:  
This start condition initiates a data transfer over the bus after a data symbol when the bus has not been idle. Therefore, SDA is set high and SCL low, followed by a start symbol.
- Stop symbol:  
A rising edge on SDA while SCL is high indicates a stop condition. This stop condition terminates a data transfer to release the bus to idle state. Between a start condition and a stop condition an arbitrary number of bytes may be transferred.

### 21.5.1.3 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 21-41 IIC Frame Example (simplified)**

## 21.5.2 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  $SCTRL.TRM = 11_B$  should 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 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 DIN) with  $DX0CR.INSW = 0$  and configure the transmit data output signal DOUT (with  $SCTRL.DOCFG = 00_B$ ) 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  $BRGH.SCLKCFG = 00_B$ ) 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 basic I/O port configuration for the IIC I/O pins must also setup correctly before the IIC mode becomes enabled by  $CCR.MODE = 0100_B$ .
- **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  $1/f_{PP}$  if another IIC participant lengthens the SCL low phase on the bus.  
 More details are given in [Section 21.5.3](#).
- **Data format configuration:**  
 The data format has to be configured for 8 data bits ( $SCTRH.WLE = 7$ ), unlimited data flow ( $SCTRH.FLE = 3FF_H$ ), and MSB shifted first ( $SCTRL.SDIR = 1$ ). The parity generation has to be disabled ( $CCR.PM = 00_B$ ).
- **General hints:**  
 The IIC slave module becomes active (for reception or transmission) if it is selected by the address sent by the master. In the case that the slave sends data to the master, it uses the transmit path. So a master must not request to read data from the slave address defined for its own channel in order to avoid collisions.  
 The built-in error detection mechanisms are only activated while the IIC module is

taking part in IIC bus traffic.

If the slave can not deal with too high frequencies, it can lengthen the low phase of the SCL signal.

For data transfers according to the IIC specification, the shift data line SDA shall only change while SCL = 0 (defined by IIC bus specification).

### **21.5.2.1 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 21-25** 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  $f_{SYS}$  and  $f_{PPP}$ .

### **21.5.2.2 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.

### **21.5.2.3 Baud Rate Update**

The baud rate setting can be changed from frame to frame. The BRGL/H register setting and PCR.STIM are sampled (shadowed) while the IIC bus is idle. A new setting of these bits can be programmed while a frame is running. The new setting will be taken into account with the start of the next frame. In order to minimize the risk of inconsistencies when changing baud rate setting (several registers have to be updated), it is recommended to avoid baud rate changes while the IIC protocol is enabled, especially for slave devices.

### **21.5.2.4 Master Arbitration**

During the address and data transmission, the master transmitter checks at the rising edge of SCL for each data bit if the value it is sending is equal to the value read on the SDA line. If yes, the next data bit values can be 0. If this is not the case (transmitted

value = 1, value read = 0), the master has lost the transmit arbitration. This is indicated by status flag PSR.ARL and can generate a protocol interrupt if enabled by PCRH.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.

#### **21.5.2.5 Release of TBUF**

In case of a non-acknowledge or an error, the content of TBUF becomes invalid. In both cases, the software has to flush the transmit buffer and to set it up again with appropriate values to react on the previous event.

#### **21.5.2.6 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 TCSRL.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, TCSRL.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 TCSRL.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.



### **21.5.2.7 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.

- **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 TCSRL.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 address 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.
- **Protocol interrupt events:**  
The IIC protocol related interrupt events are either indicating the reception of symbols or the detection of frame errors (common indication PSR.ERR) or unexpected/wrong TDF codes (common indication PSR.WTDF).
  - 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)
  - non-acknowledge received (PSR.NACK)
  - start condition not at the expected position in a frame (PSR.ERR)
  - stop condition not at the expected position in a frame (PSR.ERR)
  - as slave, 10-bit address interrupted by a stop condition after the first address byte (PSR.ERR)
  - TDF slave code in master mode (PSR.WTDF)
  - TDF master code in slave mode (PSR.WTDF)
  - Reserved TDF code found (PSR.WDTF)
  - Start condition code during a running frame in master mode (PSR.WTDF)
  - Data byte transmission code after transfer direction has been changed to reception (master read) in master mode (PSR.WTDF)

If a wrong TDF code is found in TBUF, the error event is active until the TDF value is either corrected or invalidated. If the related interrupt is enabled, the interrupt handler should check PSR.WDTF first and correct or invalidate TBUF, before dealing with the other possible interrupt events.

### **21.5.2.8 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 a non-acknowledge (SDA = 1). In addition to the match of the programmed address, an other address byte value has to be answered with an acknowledge if the slave is capable to handle the corresponding requests. The address byte 00<sub>H</sub> indicates a general call address, that can be acknowledged. The value 01<sub>H</sub> 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<sub>H</sub> is acknowledged if bit PCRH.ACK00 is set.
- The address byte 01<sub>H</sub> 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<sub>B</sub>, the slave device waits for a second address byte and compares it also to PCR.SLAD[7:0] and sends an acknowledge accordingly to cover the 10 bit addressing mode. The user has to take care about reserved addresses (refer to IIC specification for more detailed description). Only the address 1111 0XX<sub>B</sub> 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.

### **21.5.2.9 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.

### **21.5.2.10 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 1 at this bit position indicates that the data byte has been received when the device has been in slave mode, whereas a 0 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.

### 21.5.3 Symbol Timing

The symbol timing of the IIC is determined by the master stimulating the shift clock line SCL. It is different for standard and fast IIC mode.

- 100 kBaud standard mode (PCR.H.STIM = 0):  
The symbol timing is based on 10 time quanta  $t_q$  per symbol. A minimum module clock frequency  $f_{SYS} = 2$  MHz is required.
- 400 kBaud standard mode (PCR.H.STIM = 1):  
The symbol timing is based on 25 time quanta  $t_q$  per symbol. A minimum module clock frequency  $f_{SYS} = 10$  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 BRGL define the length of a time quantum  $t_q$  that is given by one period of  $f_{PCTQ}$ .

- BRGL.CTQSEL  
to define the input frequency  $f_{CTQIN}$  for the time quanta generation
- BRGL.PCTQ  
to define the length of a time quantum (division of  $f_{CTQIN}$  by 1, 2, 3, or 4)
- BRGL.DCTQ  
to define the number of time quanta per symbol (number of  $t_q = DCTQ + 1$ )

The standard setting is given by CTQSEL = 00<sub>B</sub> ( $f_{CTQIN} = f_{PDIV}$ ) and PPPEN = 0 ( $f_{PPP} = f_{IN}$ ). Under these conditions, the frequency  $f_{PCTQ}$  is given by:

$$f_{PCTQ} = f_{PIN} \times \frac{1}{PDIV + 1} \times \frac{1}{PCTQ + 1} \quad (21.10)$$

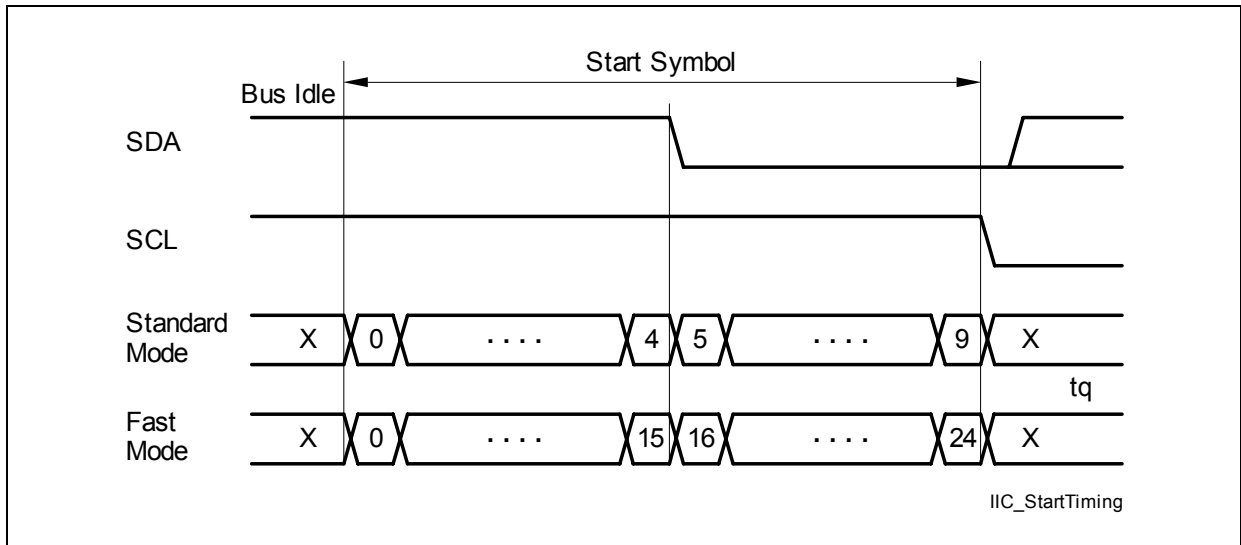
To respect the specified SDA hold time of 300 ns after a falling edge of signal SCL, a hold delay  $t_{HDEL}$  has been introduced. It also prevents an erroneous detection of a start or a stop condition. The length of this delay can be programmed by bit field PCR.HDEL. Taking into account the input sampling and output update, bit field HDEL can be programmed according to:

$$\begin{aligned} HDEL &\geq 300 \text{ ns} \times f_{PPP} - \left( 3 \times \frac{f_{PPP}}{f_{SYS}} \right) + 1 && \text{with digital filter and } HDEL_{min} = 2 \\ HDEL &\geq 300 \text{ ns} \times f_{PPP} - \left( 3 \times \frac{f_{PPP}}{f_{SYS}} \right) + 2 && \text{without digital filter and } HDEL_{min} = 1 \end{aligned} \quad (21.11)$$

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 won't be treated as a start or stop condition.

### 21.5.3.1 Start Symbol

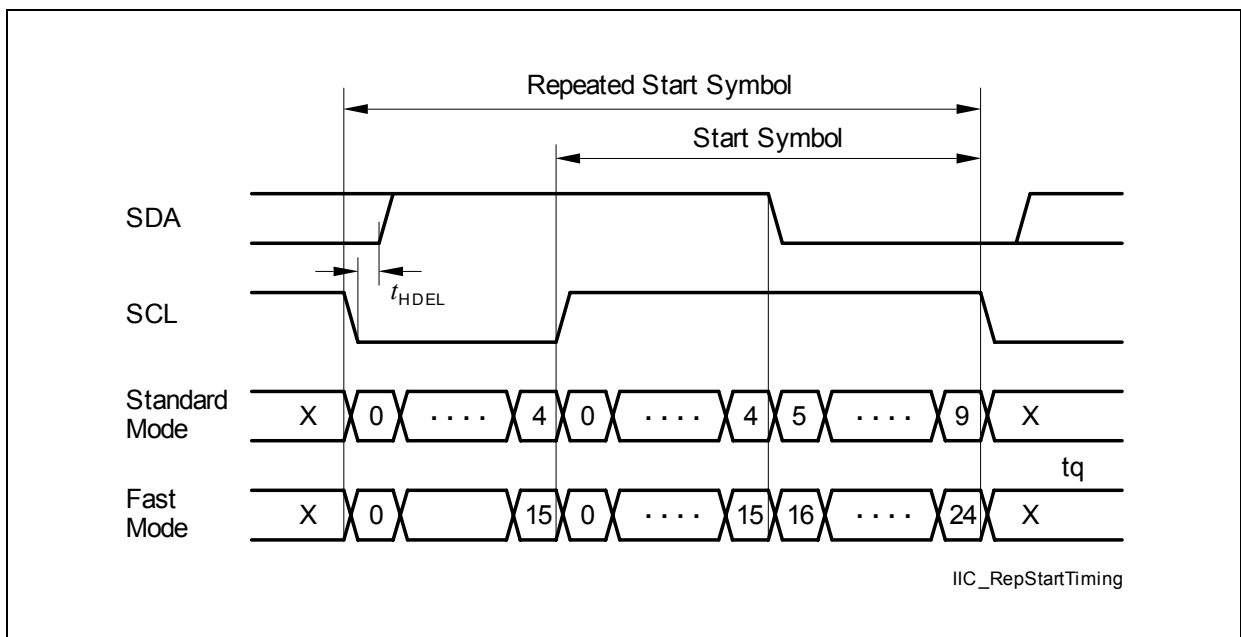
**Figure 21-42** shows the general start symbol timing.



**Figure 21-42 Start Symbol Timing**

### 21.5.3.2 Repeated Start Symbol

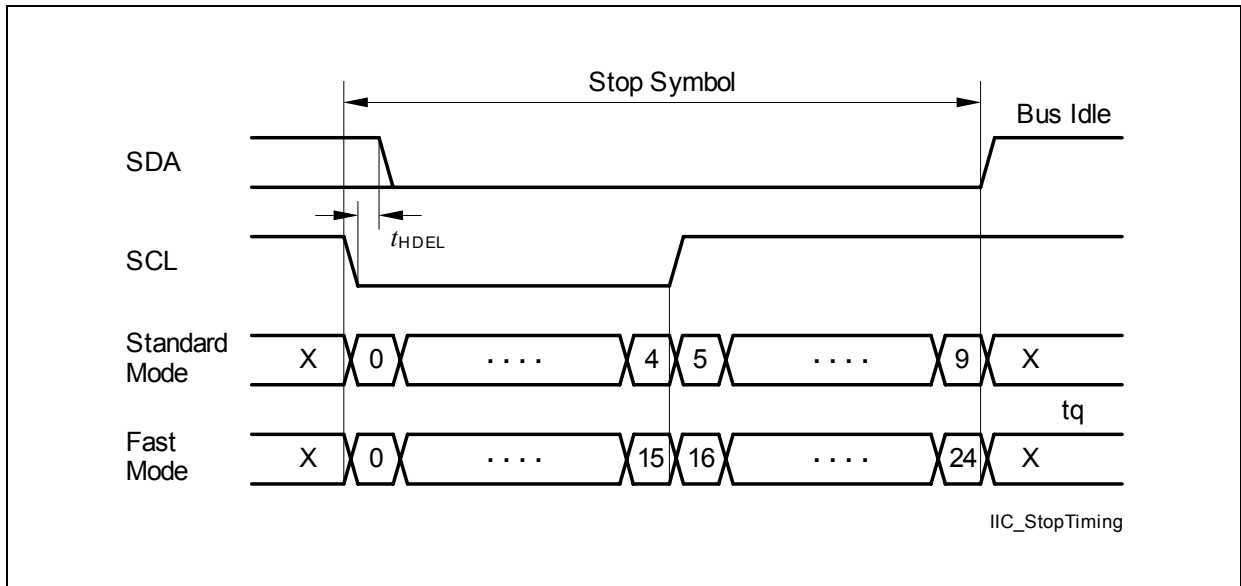
During the first part of a repeated start symbol, an SCL low value is driven for the specified number of time quanta. Then a high value is output. After the detection of a rising edge at the SCL input, a normal start symbol is generated, as shown in **Figure 21-43**.



**Figure 21-43 Repeated Start Symbol Timing**

### 21.5.3.3 Stop Symbol

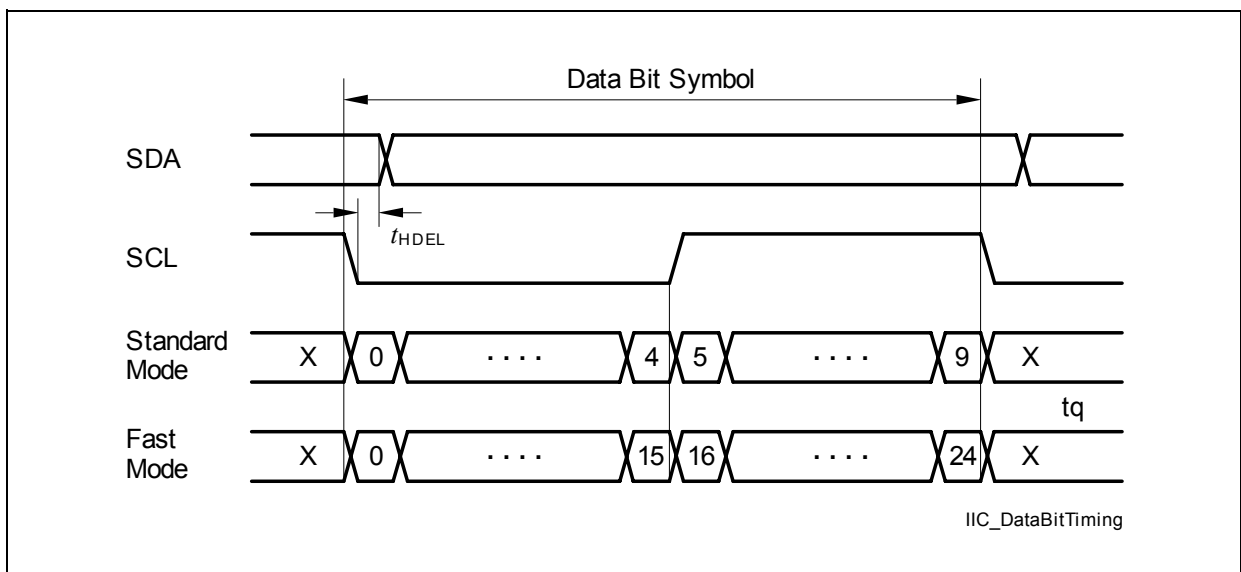
**Figure 21-44** shows the stop symbol timing.



**Figure 21-44 Stop Symbol Timing**

### 21.5.3.4 Data Bit Symbol

**Figure 21-45** shows the general data bit symbol timing.



**Figure 21-45 Data Bit Symbol**

Output SDA changes after the time  $t_{HDEL}$  defined by PCRH.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  $f_{\text{SYS}}$  to the path. The minimum module input frequency has to be selected properly to ensure the SDA setup time to SCL rising edge.

### **21.5.4 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.

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.

#### **21.5.4.1 Transmit Data Formats**

The following transmit data formats are available in master mode:

- Send data byte as master (TDF = 000<sub>B</sub>):  
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.
- Receive data byte and send acknowledge 0 (TDF = 010<sub>B</sub>):  
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.
- Receive data byte and send acknowledge 1 (TDF = 011<sub>B</sub>):  
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.

**Universal Serial Interface Channel**

- Send start condition ( $TDF = 100_B$ ):  
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).
- Send repeated start condition ( $TDF = 101_B$ ):  
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).
- Send stop condition ( $TDF = 110_B$ ):  
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.
- $TDF = 111_B$ :  
Reserved and 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):

- Send data byte as slave ( $TDF = 001_B$ ):  
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.



### 21.5.4.2 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 won't be set. Therefore the dummy read won't generate a receive interrupt and the data byte won't 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<sub>B</sub>) is not possible and won't be accepted (leading to a wrong TDF Code indication).

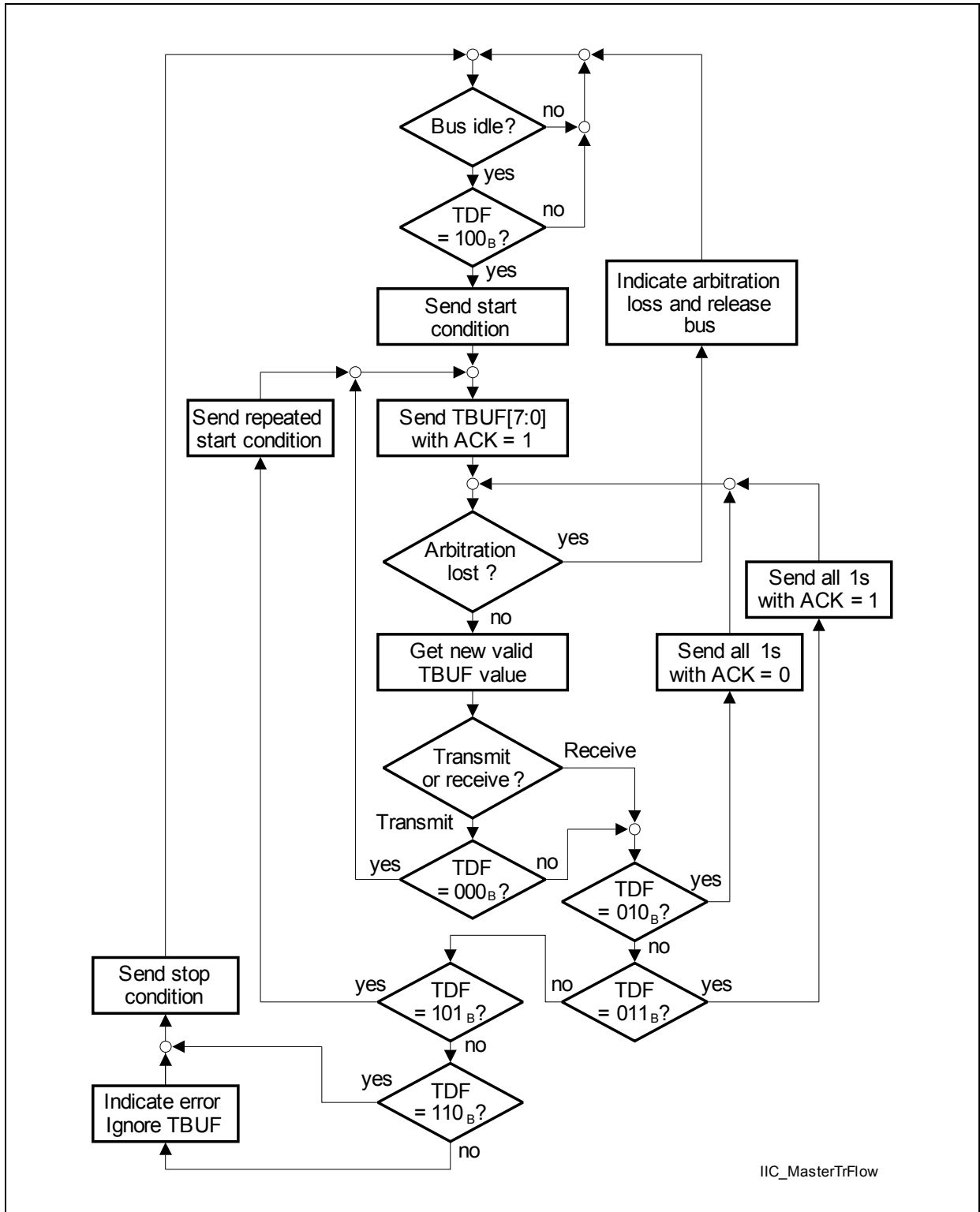
**Table 21-9 Valid TDF Codes Overview**

Frame Position	Valid TDF Codes
First TDF code (master idle)	Start (100 <sub>B</sub> )
Read transfer: second TDF code (after start or repeated start)	Receive with acknowledge (010 <sub>B</sub> ) or receive with not-acknowledge (011 <sub>B</sub> )
Write transfer: second TDF code (after start or repeated start)	Transmit (000 <sub>B</sub> ), repeated start (101 <sub>B</sub> ), or stop (110 <sub>B</sub> )
Read transfer: third and subsequent TDF code after acknowledge	Receive with acknowledge (010 <sub>B</sub> ) or receive with not-acknowledge (011 <sub>B</sub> )
Read transfer: third and subsequent TDF code after not-acknowledge	Repeated start (101 <sub>B</sub> ) or stop (110 <sub>B</sub> )
Write transfer: third and subsequent TDF code	Transmit (000 <sub>B</sub> ), repeated start (101 <sub>B</sub> ), or stop (110 <sub>B</sub> )

- First TDF code:  
A master transfer starts with the TDF start code (100<sub>B</sub>). All other codes are ignored, but no WTDF error will be indicated.
- TDF code after a start (100<sub>B</sub>) or repeated start code (101<sub>B</sub>) 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<sub>B</sub> and 011<sub>B</sub> 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.

### **Universal Serial Interface Channel**

- TDF code after a start ( $100_B$ ) or repeated start code ( $101_B$ ) in case of a write access:  
If a master-write transfer is started (determined by the LSB of the address byte = 0), the master still owns the SDA line. In this case, the transmit ( $000_B$ ), repeated start ( $101_B$ ) and stop ( $110_B$ ) codes are valid. The other codes are considered as wrong. To abort the transfer in case of a wrong code, the STOP condition is generated immediately.
- TDF code of the third and subsequent command in case of a read access with acknowledged previous data byte:  
If a master-read transfer is started (determined by the LSB of the address byte), the transfer direction of SDA changes and the slave will actively drive the data line. To force the slave to release the SDA line, the master has to not-acknowledge a byte transfer. In this case, only the receive codes  $010_B$  and  $011_B$  are valid. To abort the transfer in case of a wrong code, a dummy read must be performed by the master before the STOP condition can be generated.
- TDF code of the third and subsequent command in case of a read access with a not-acknowledged previous data byte:  
If a master-read transfer is started (determined by the LSB of the address byte), the transfer direction of SDA changes and the slave will actively drive the data line. To force the slave to release the SDA line, the master has to not-acknowledge a byte transfer. In this case, only the restart ( $101_B$ ) and stop code ( $110_B$ ) are valid. To abort the transfer in case of a wrong code, the STOP condition is generated immediately.
- TDF code of the third and subsequent command in case of a write access:  
If a master-write transfer is started (determined by the LSB of the address byte), the master still owns the SDA line. In this case, the transmit ( $000_B$ ), repeated start ( $101_B$ ) and stop ( $110_B$ ) codes are valid. The other codes are considered as wrong. To abort the transfer in case of a wrong code, the STOP condition is generated immediately.
- After a master device has received a non-acknowledge from a slave device, a stop condition will be sent out automatically, except if the following TDF code requests a repeated start condition. In this case, the TDF code is taken into account, whereas all other TDF codes are ignored.



**Figure 21-46 IIC Master Transmission**

## 21.5.5 IIC Protocol Registers

In IIC mode, the registers PCRH, PCRL and PSR handle IIC related information.

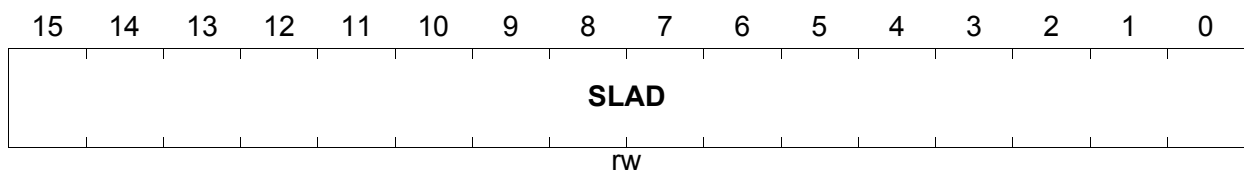
### 21.5.5.1 IIC Protocol Control Registers

In IIC mode, the PCRL/PCRH register bits or bit fields are defined as described in this section.

#### PCRL

**Protocol Control Register L [IIC Mode] (40<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

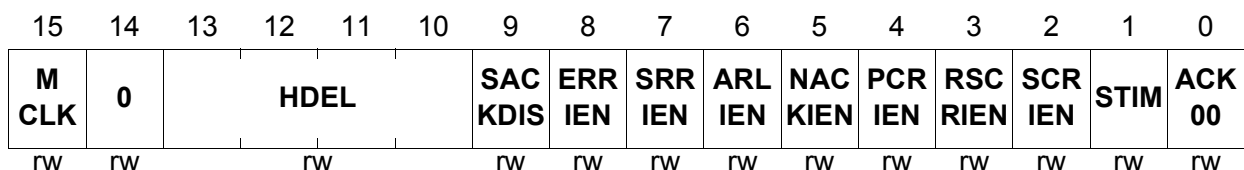


Field	Bits	Type	Description
<b>SLAD</b>	[15:0]	rw	<b>Slave Address</b> This bit field contains the programmed slave address. The corresponding bits in the first received address byte are compared to the bits SLAD[15:9] to check for address match. If SLAD[15:11] = 11110 <sub>B</sub> , then the second address byte is also compared to SLAD[7:0].

#### PCRH

**Protocol Control Register H [IIC Mode] (42<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
<b>ACK00</b>	0	rw	<b>Acknowledge 00<sub>H</sub></b> This bit defines if a slave device should be sensitive to the slave address 00 <sub>H</sub> . 0 <sub>B</sub> The slave device is not sensitive to this address. 1 <sub>B</sub> The slave device is sensitive to this address.

**Universal Serial Interface Channel**

<b>Field</b>	<b>Bits</b>	<b>Type</b>	<b>Description</b>
<b>STIM</b>	1	rw	<b>Symbol Timing</b> This bit defines how many time quanta are used in a symbol. $0_B$ A symbol contains 10 time quanta. The timing is adapted for standard mode (100 kBaud). $1_B$ A symbol contains 25 time quanta. The timing is adapted for fast mode (400 kBaud).
<b>SCRIEN</b>	2	rw	<b>Start Condition Received Interrupt Enable</b> This bit enables the generation of a protocol interrupt if a start condition is detected. $0_B$ The start condition interrupt is disabled. $1_B$ The start condition interrupt is enabled.
<b>RSCRIEN</b>	3	rw	<b>Repeated Start Condition Received Interrupt Enable</b> This bit enables the generation of a protocol interrupt if a repeated start condition is detected. $0_B$ The repeated start condition interrupt is disabled. $1_B$ The repeated start condition interrupt is enabled.
<b>PCRIEN</b>	4	rw	<b>Stop Condition Received Interrupt Enable</b> This bit enables the generation of a protocol interrupt if a stop condition is detected. $0_B$ The stop condition interrupt is disabled. $1_B$ The stop condition interrupt is enabled.
<b>NACKIEN</b>	5	rw	<b>Non-Acknowledge Interrupt Enable</b> This bit enables the generation of a protocol interrupt if a non-acknowledge is detected by a master. $0_B$ The non-acknowledge interrupt is disabled. $1_B$ The non-acknowledge interrupt is enabled.
<b>ARLIEN</b>	6	rw	<b>Arbitration Lost Interrupt Enable</b> This bit enables the generation of a protocol interrupt if an arbitration lost event is detected. $0_B$ The arbitration lost interrupt is disabled. $1_B$ The arbitration lost interrupt is enabled.
<b>SRRIEN</b>	7	rw	<b>Slave Read Request Interrupt Enable</b> This bit enables the generation of a protocol interrupt if a slave read request is detected. $0_B$ The slave read request interrupt is disabled. $1_B$ The slave read request interrupt is enabled.

**Universal Serial Interface Channel**

Field	Bits	Type	Description
<b>ERRIEN</b>	8	rw	<b>Error Interrupt Enable</b> This bit enables the generation of a protocol interrupt if an IIC error condition is detected (indicated by PSR.ERR or PSR.WTDF). 0 <sub>B</sub> The error interrupt is disabled. 1 <sub>B</sub> The error interrupt is enabled.
<b>SACKDIS</b>	9	rw	<b>Slave Acknowledge Disable</b> This bit disables the generation of an active acknowledge signal for a slave device (active acknowledge = 0 level). Once set by software, it is automatically cleared with each (repeated) start condition. If this bit is set after a byte has been received (indicated by an interrupt) but before the next acknowledge bit has started, the next acknowledge bit will be sent with passive level. This would indicate that the receiver does not accept more bytes. As a result, a minimum of 2 bytes will be received if the first receive interrupt is used to set this bit. 0 <sub>B</sub> The generation of an active slave acknowledge is enabled (slave acknowledge with 0 level = more bytes can be received). 1 <sub>B</sub> The generation of an active slave acknowledge is disabled (slave acknowledge with 1 level = reception stopped).
<b>HDEL</b>	[13:10]	rw	<b>Hardware Delay</b> This bit field defines the delay used to compensate the internal treatment of the SCL signal (see <a href="#">Page 21-171</a> ) in order to respect the SDA hold time specified for the IIC protocol.
<b>0</b>	14	rw	<b>Reserved</b> Returns 0 if read; should be written with 0.
<b>MCLK</b>	15	rw	<b>Master Clock Enable</b> This bit enables generation of the master clock MCLK (not directly used for IIC protocol, can be used as general frequency output). 0 <sub>B</sub> The MCLK generation is disabled and MCLK is 0. 1 <sub>B</sub> The MCLK generation is enabled.

### 21.5.5.2 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 doesn't 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] (44<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>AIF</b>	<b>RIF</b>	<b>TBIF</b>	<b>TSIF</b>	<b>DLIF</b>	<b>RSIF</b>	<b>0</b>	<b>ERR</b>	<b>SRR</b>	<b>ARL</b>	<b>N ACK</b>	<b>PCR</b>	<b>R SCR</b>	<b>SCR</b>	<b>W TDF</b>	<b>SL SEL</b>
rwh	rwh	rwh	rwh	rwh	rwh	r	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
<b>SLSEL</b>	0	rwh	<b>Slave Select</b> This bit indicates that this device has been selected as slave. 0 <sub>B</sub> The device is not selected as slave. 1 <sub>B</sub> The device is selected as slave.
<b>WTDF</b>	1	rwh	<b>Wrong TDF Code Found<sup>1)</sup></b> This bit indicates that an unexpected/wrong TDF code has been found. A protocol interrupt can be generated if PCRH.ERRIEN = 1. 0 <sub>B</sub> A wrong TDF code has not been found. 1 <sub>B</sub> A wrong TDF code has been found.
<b>SCR</b>	2	rwh	<b>Start Condition Received<sup>1)</sup></b> This bit indicates that a start condition has been detected on the IIC bus lines. A protocol interrupt can be generated if PCRH.SCRIEN = 1. 0 <sub>B</sub> A start condition has not yet been detected. 1 <sub>B</sub> A start condition has been detected.

**Universal Serial Interface Channel**

Field	Bits	Type	Description
<b>RSCR</b>	3	rwh	<b>Repeated Start Condition Received<sup>1)</sup></b> This bit indicates that a repeated start condition has been detected on the IIC bus lines. A protocol interrupt can be generated if PCRH.RSCRIEN = 1. 0 <sub>B</sub> A repeated start condition has not yet been detected. 1 <sub>B</sub> A repeated start condition has been detected.
<b>PCR</b>	4	rwh	<b>Stop Condition Received<sup>1)</sup></b> This bit indicates that a stop condition has been detected on the IIC bus lines. A protocol interrupt can be generated if PCRH.PCRIEN = 1. 0 <sub>B</sub> A stop condition has not yet been detected. 1 <sub>B</sub> A stop condition has been detected.
<b>NACK</b>	5	rwh	<b>Non-Acknowledge Received<sup>1)</sup></b> 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 PCRH.NACKIEN = 1. 0 <sub>B</sub> A non-acknowledge has not been received. 1 <sub>B</sub> A non-acknowledge has been received.
<b>ARL</b>	6	rwh	<b>Arbitration Lost<sup>1)</sup></b> This bit indicates that an arbitration has been lost. A protocol interrupt can be generated if PCRH.ARLIEN = 1. 0 <sub>B</sub> An arbitration has not been lost. 1 <sub>B</sub> An arbitration has been lost.
<b>SRR</b>	7	rwh	<b>Slave Read Request<sup>1)</sup></b> 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 PCRH.SRRIEN = 1. 0 <sub>B</sub> A slave read request has not been detected. 1 <sub>B</sub> A slave read request has been detected.



**Universal Serial Interface Channel**

Field	Bits	Type	Description
<b>ERR</b>	8	rwh	<b>Error<sup>1)</sup></b> This bit indicates that an IIC error (frame format or TDF code) has been detected. A protocol interrupt can be generated if PCRH.ERRIEN = 1. 0 <sub>B</sub> An IIC error has not been detected. 1 <sub>B</sub> An IIC error has been detected.
<b>0</b>	9	r	<b>Reserved</b> Returns 0 if read; not modified in IIC mode.
<b>RSIF</b>	10	rwh	<b>Receiver Start Indication Flag</b> 0 <sub>B</sub> A receiver start event has not occurred. 1 <sub>B</sub> A receiver start event has occurred.
<b>DLIF</b>	11	rwh	<b>Data Lost Indication Flag</b> 0 <sub>B</sub> A data lost event has not occurred. 1 <sub>B</sub> A data lost event has occurred.
<b>TSIF</b>	12	rwh	<b>Transmit Shift Indication Flag</b> 0 <sub>B</sub> A transmit shift event has not occurred. 1 <sub>B</sub> A transmit shift event has occurred.
<b>TBIF</b>	13	rwh	<b>Transmit Buffer Indication Flag</b> 0 <sub>B</sub> A transmit buffer event has not occurred. 1 <sub>B</sub> A transmit buffer event has occurred.
<b>RIF</b>	14	rwh	<b>Receive Indication Flag</b> 0 <sub>B</sub> A receive event has not occurred. 1 <sub>B</sub> A receive event has occurred.
<b>AIF</b>	15	rwh	<b>Alternative Receive Indication Flag</b> 0 <sub>B</sub> An alternative receive event has not occurred. 1 <sub>B</sub> An alternative receive event has occurred.

1) This status bit can generate a protocol interrupt (see [Page 21-24](#)). The general interrupt status flags are described in the general interrupt chapter.

## **21.6 IIS Protocol**

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 = 0011B` with `CCFG.IIS = 1` (IIS mode is available).

This chapter contains the following sections:

- Introduction (see [Page 21-185](#))
- General IIS issues (see [Page 21-189](#))
- Master mode operation (see [Page 21-194](#))
- Slave mode operation (see [Page 21-198](#))
- Protocol registers (see [Page 21-199](#))

### **21.6.1 Introduction**

The IIS protocol is a synchronous serial communication protocol mainly for audio and infotainment applications and refers to the Philips specification, 1986, revised June 5, 1996.

#### **21.6.1.1 Signal Description**

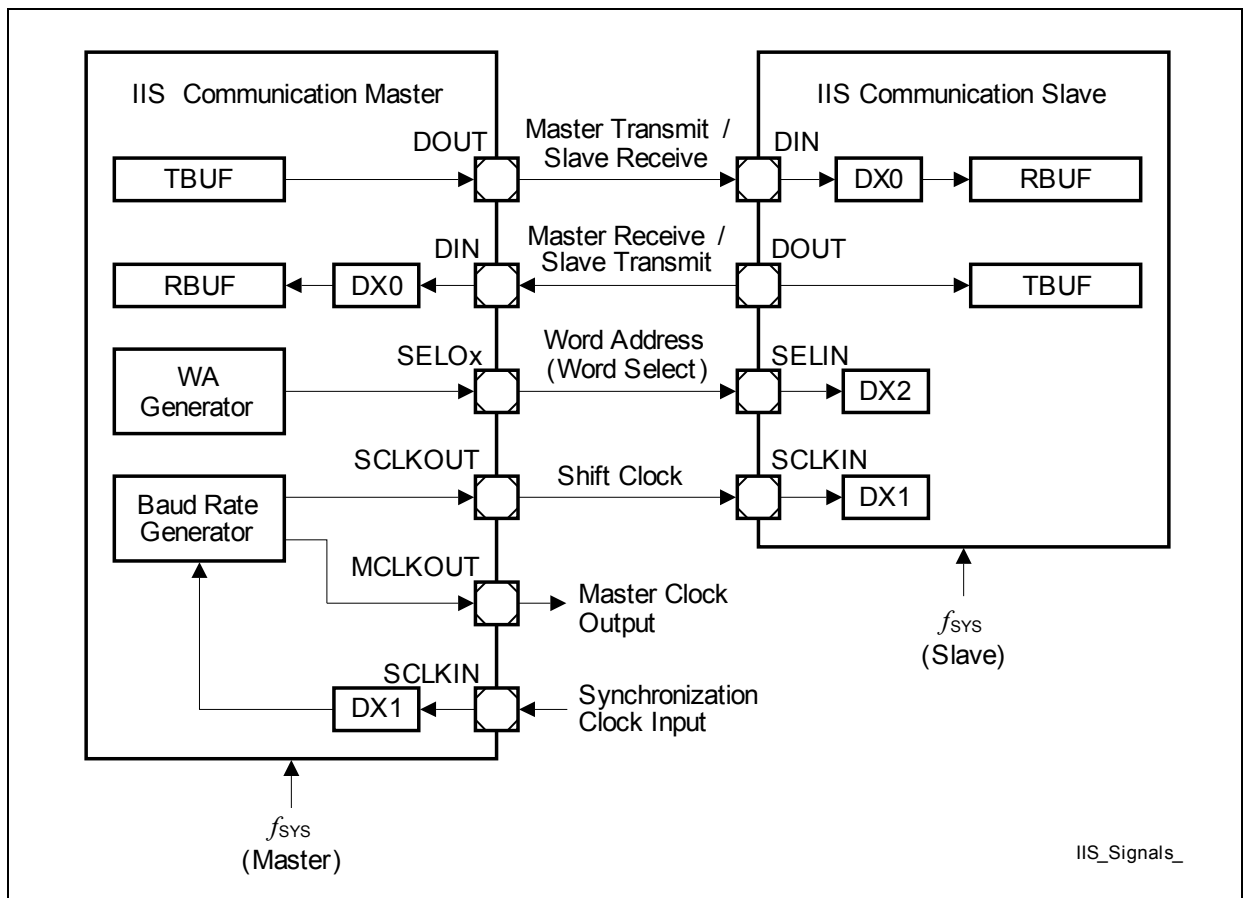
A connection between an IIS master and an IIS slave is based on the following signals:

- 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 21-10 IIS IO Signals**

IIS Mode	Receive Data	Transmit Data	Shift Clock	Word Address
Master	Input DIN, handled by DX0	Output DOUT	Output SCLKOUT	Output(s) SELOx
Slave	Input DIN, handled by DX0	Output DOUT	Input SCLKIN, handled by DX1	Input SELIN, handled by DX2



**Figure 21-47 IIS Signals**

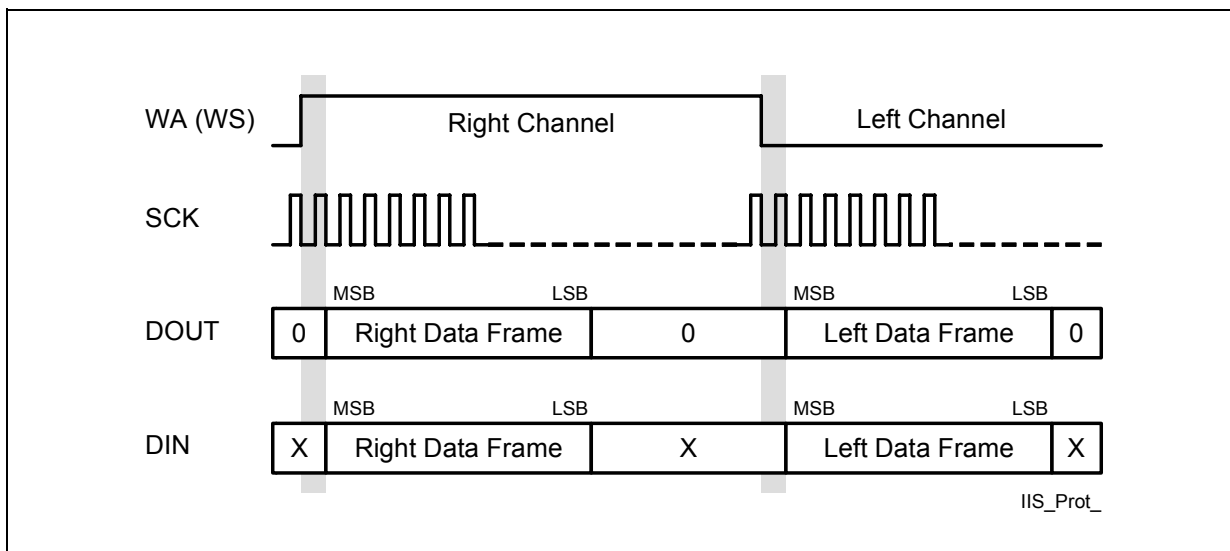
Two additional signals are available for the USIC IIS communication master:

- 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  $f_{SYS}$  of the communication master. It can be used as master clock input of a communication network with synchronized IIS connections.

### 21.6.1.2 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.

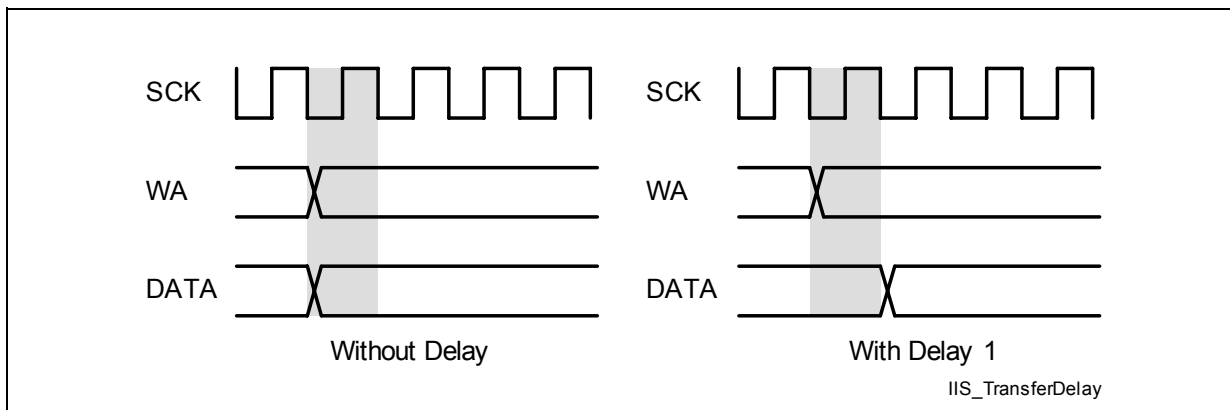
In a USIC communication channel, data words are tagged for being transmitted for the left or for the right channel. Also the received data words contain a tag identifying the WA state when the data has been received.



**Figure 21-48 Protocol Overview**

### 21.6.1.3 Transfer Delay

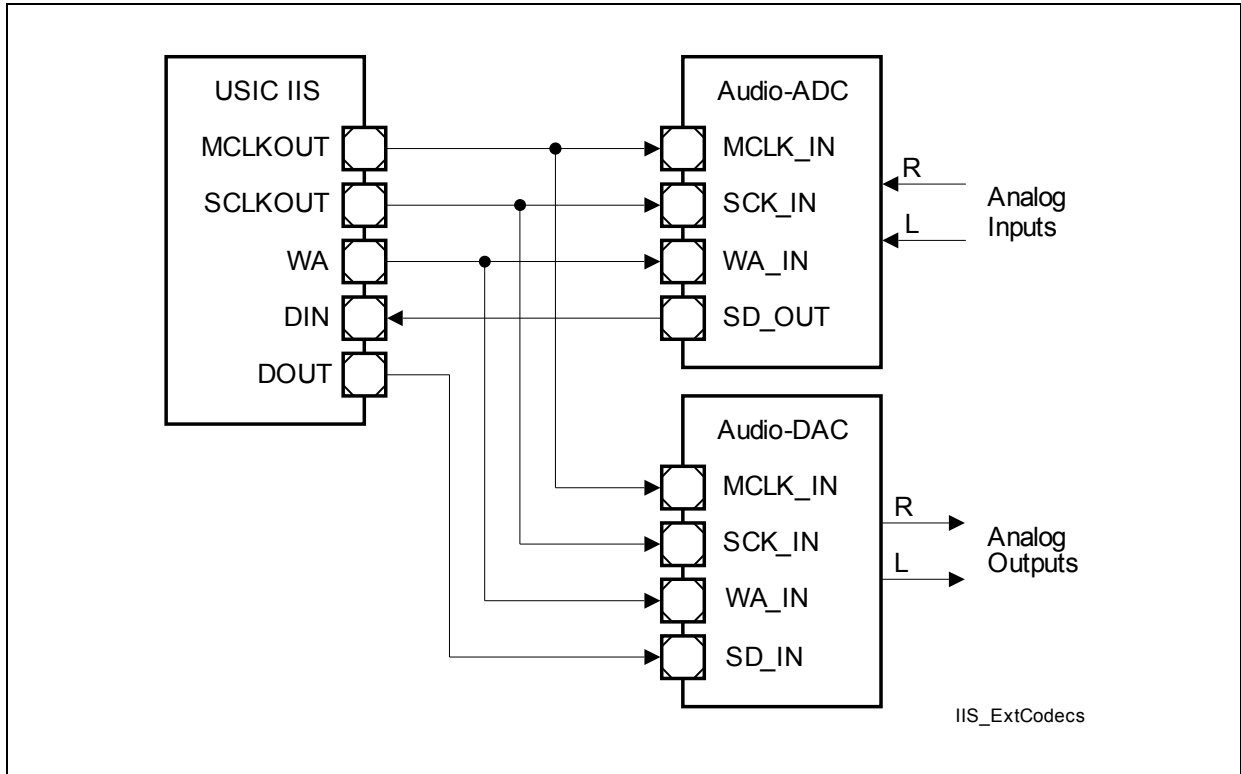
The transfer delay feature allows the transfer of data (transmission and reception) with a programmable delay (counted in shift clock periods).



**Figure 21-49 Transfer Delay for IIS**

#### 21.6.1.4 Connection of External Audio Components

The IIS signals can be used to communicate with external audio devices (such as Codecs) or other audio data sources/destinations.



**Figure 21-50 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).

## **21.6.2 Operating the IIS**

This chapter contains IIS issues, that are of general interest and not directly linked to master mode or slave mode.

### **21.6.2.1 Frame Length and Word Length Configuration**

After each change of the WA signal, a complete data frame is intended to be transferred (frame length  $\leq$  system word length). The number of data bits transferred after a change of signal WA is defined by SCTR.H.FLE. A data frame can consist of several data words with a data word length defined by SCTR.H.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.H.FLE, the additional bits are transmitted with passive data level (SCTRL.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 TCSRL to 0.

### **21.6.2.2 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.

### **21.6.2.3 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 PCRL.WAGEN is internally considered as 0 (the bit itself is not changed). If WAGEN = 1, then the current system word cycle is finished and then 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.H.TDEL.  
When leaving a stop mode with WAGEN = 1, the WA generation starts from the beginning.

### 21.6.2.4 Transfer Delay

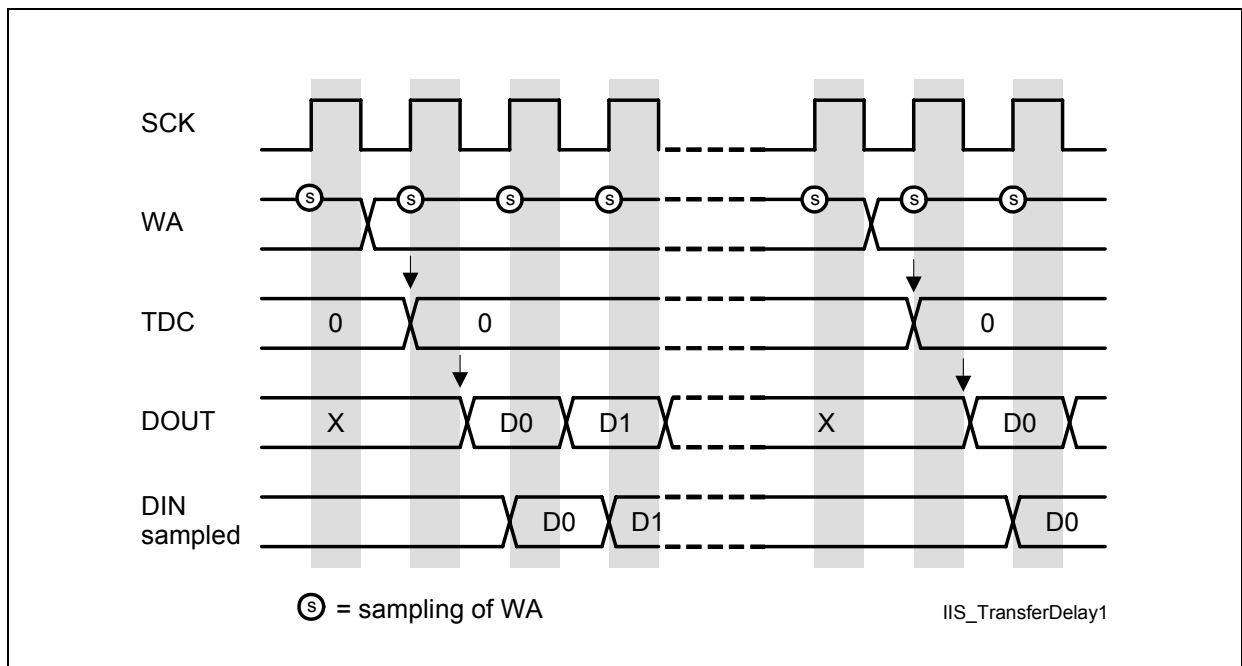
The transfer delay can be used to synchronize a data transfer to an event (e.g. a change of the WA signal). This event has to be synchronously generated to the falling edge of the shift clock SCK (like the change of the transmit data), because the input signal for the event is directly sampled in the 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 21-49](#), 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.H.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.H.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.H.TDEL+1.

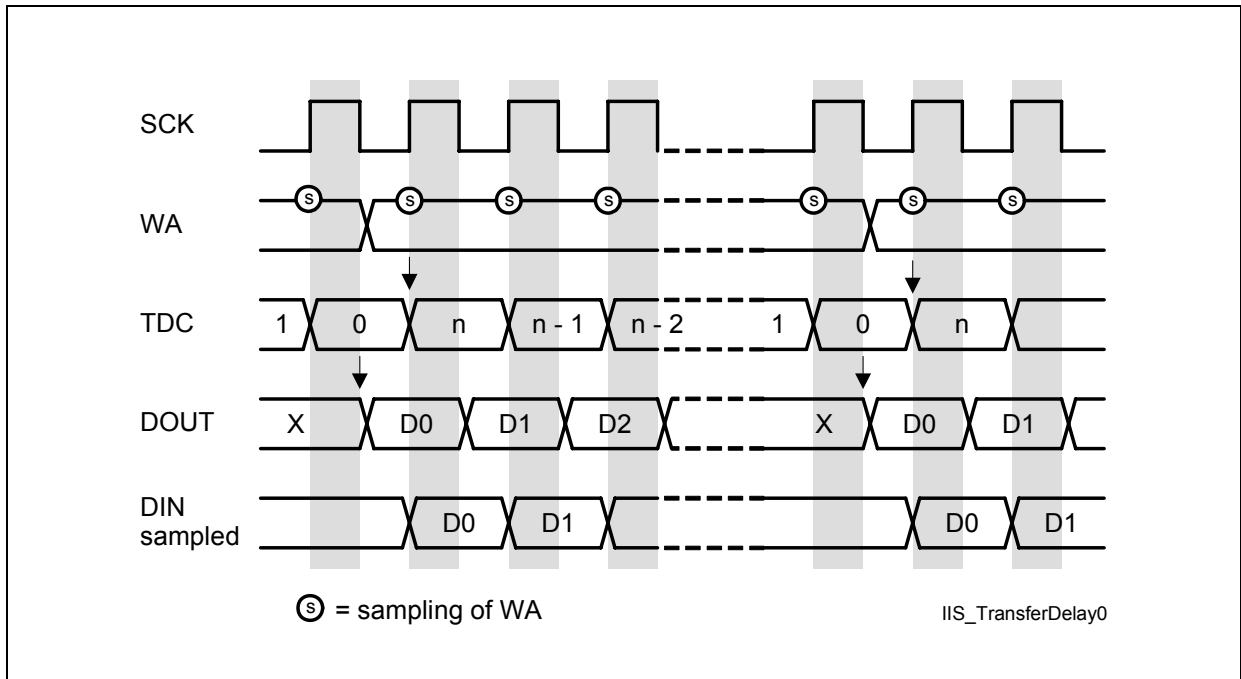
In the example in [Figure 21-51](#), the reload value PCR.H.TDEL for TDC is 0. When the samples taken on receiver side show the change of the WA signal, the counter TDC is reloaded. If the reload value is 0, the data transfer starts with 1 shift clock cycle delay compared to the change of WA.



**Figure 21-51 Transfer Delay with Delay 1**

## Universal Serial Interface Channel

The ideal case without any transfer delay is shown in [Figure 21-52](#). 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 21-52 Transfer Delay with 0 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 PCRH.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 PCRH.TDEL is set to a value bigger than the system word length, no transfer takes place.

### 21.6.2.5 Parity Mode

Parity generation is not supported in IIS mode and bit field CCR.PM = 00<sub>B</sub> has to be programmed.

### 21.6.2.6 Transfer Mode

In IIS mode, bit field SCTRL.TRM = 11<sub>B</sub> has to be programmed to allow data transfers. Setting SCTRL.TRM = 00<sub>B</sub> disables and stops the data transfer immediately.



### **21.6.2.7 Data Transfer Interrupt Handling**

The data transfer interrupts indicate events related to IIS frame handling.

- Transmit buffer interrupt TBI:  
Bit PSR.TBIF is set after the start of first data bit of a data word.
- Transmit shift interrupt TSI:  
Bit PSR.TSIF is set after the start of the last data bit of a data word.
- Receiver start interrupt RSI:  
Bit PSR.RSIF is set after the reception of the first data bit of a data word.  
With this event, bit TCSRL.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.

### **21.6.2.8 Protocol-Related Argument and Error**

In order to distinguish between data words received for the left or the right channel, the IIS protocol pre-processor samples the level of the WA input (just after the WA transition) and propagates it as protocol-related error (although it is not an error, but an indication) to the receive buffer status register at the bit position RBUFSR[9]. This bit position defines if either a standard receive interrupt (if RBUFSR[9] = 0) or an alternative receive interrupt (if RBUFSR[9] = 1) becomes activated when a new data word has been received. Incoming data can be handled by different interrupts or DMA mechanisms for the left and the right channel if the corresponding events are directed to different interrupt nodes. Flag PAR is always 0.

### **21.6.2.9 Transmit Data Handling**

The IIS protocol pre-processor allows to distinguish between the left and the right channel for data transmission. Therefore, bit TCSRL.WA indicates on which channel the data in the buffer will be transmitted. If TCSRL.WA = 0, the data will be transmitted after a falling edge of WA. If TCSRL.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 TCSRL.WA can be automatically updated by the transmit control information TCI[4] for each data word if TCSRL.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.

#### **21.6.2.10 Receive Buffer Handling**

If a receive FIFO buffer is available (CCFG.RB = 1) and enabled for data handling (RBCTRH.SIZE > 0), it is recommended to set RBCTRH.RCIM = 11<sub>B</sub> 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 OUTRH.RCI[0] = 1, and the channel indication by the sampled WA value is given by OUTRH.RCI[4].

The standard receive buffer event and the alternative receive buffer event can be used for the following operation in RCI mode (RBCTRH.RNM = 1):

- A standard receive buffer event indicates that a data word can be read from OUTRL that belongs to a data frame started when WA = 0.
- An alternative receive buffer event indicates that a data word can be read from OUTRL that belongs to a data frame started when WA = 1.

#### **21.6.2.11 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.

### **21.6.3 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  $SCTRL.TRM = 11_B$  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 (DIN) with  $DX0CR.INSW = 1$ . Configure a transmit data output pin (DOUT) 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 DOUT signal to a pin).
- **Baud rate generation:**  
The desired baud rate setting has to be selected, comprising the fractional divider and the baud rate generator. Bit  $DX1CR.INSW = 0$  has to be programmed to use the baud rate generator output SCLK directly as input for the data shift unit. Configure a shift clock output pin with the inverted signal SCLKOUT without additional delay ( $BRGH.SCLKCFG = 01_B$ ).
- **Word address WA generation:**  
The WA generation has to be enabled by setting  $PCRL.WAGEN = 1$  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 registers SCTRL and SCTRH. Generally, the MSB is shifted first ( $SCTRL.SDIR = 1$ ).  
Bit  $TCSRL.WAMD$  can be set to use the transmit control information  $TCI[4]$  to distinguish the data words for transmission while  $WA = 0$  or while  $WA = 1$ .

#### **21.6.3.1 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  $f_{SYS}$  for  $f_{PIN}$ . This uncertainty does not accumulate over several SCLK cycles.

## Universal Serial Interface Channel

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  $f_{SYS}$ .

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  $f_{PIN}$  (BRGL.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 (BRGL.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 21-53](#).

The adjustable integer divider factor is defined by bit field BRGH.PDIV.

$$\begin{aligned} f_{SCLK} &= \frac{f_{PIN}}{2} \times \frac{1}{PDIV + 1} && \text{if PPPEN} = 0 \\ f_{SCLK} &= \frac{f_{PIN}}{2 \times 2} \times \frac{1}{PDIV + 1} && \text{if PPPEN} = 1 \end{aligned} \quad (21.12)$$

*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.*

### 21.6.3.2 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.

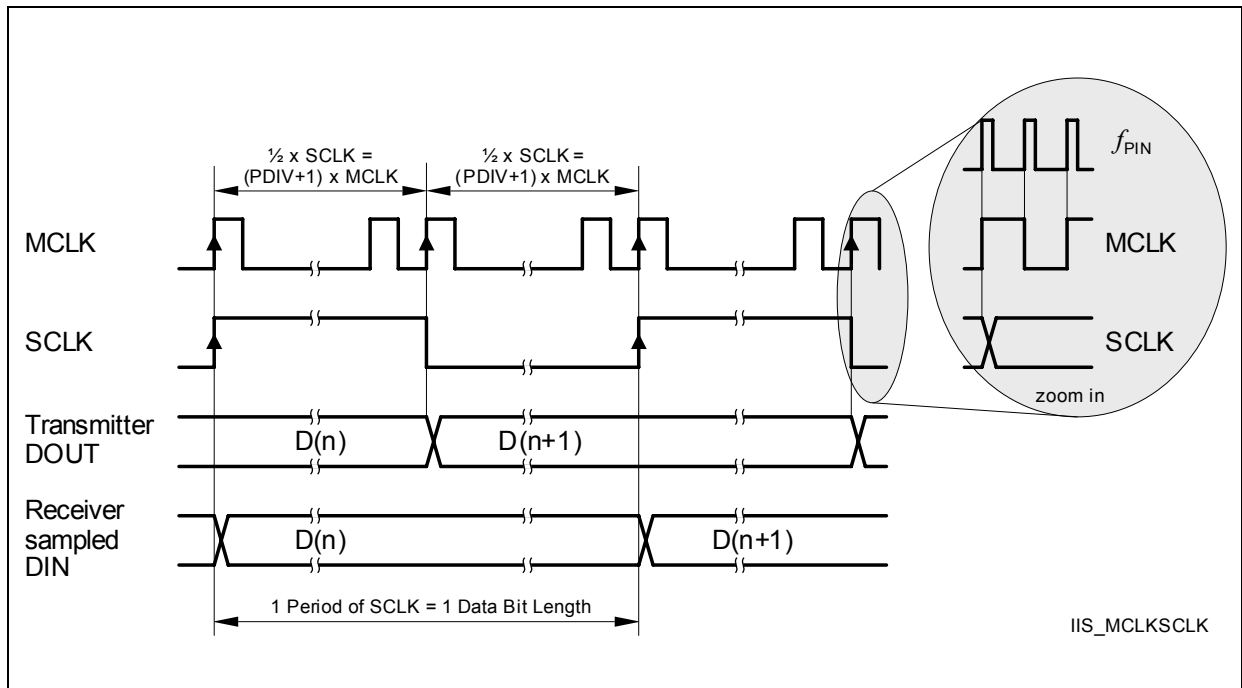
In IIS master mode, the system word length is defined by:

- BRGL.CTQSEL = 10<sub>B</sub>  
to base the WA toggling on SCLK
- BRGL.PCTQ  
to define the number N of SCLK cycles per system word length
- BRGL.DCTQ  
to define the number N of SCLK cycles per system word length

$$N = (PCTQ + 1) \times (DCTQ + 1) \quad (21.13)$$

### 21.6.3.3 Master Clock Output

The master clock signal MCLK can be generated by the master of the IIS transfer (BRGL.PPPEN = 1). It is used especially to connect external Codec devices. It can be configured by bit BRGH.MCLKCFG in its polarity to become the output signal MCLKOUT.



**Figure 21-53 MCLK and SCLK for IIS**

#### **21.6.3.4 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 PCRL.WAFEIEN = 1 for the falling edge, similar for PCRL.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 PCRL.WAGEN = 0. A protocol interrupt can be generated if PCRL.ENDIEN = 1.
- **DX2T event:**  
An activation of the trigger signal DX2T is indicated by PSR.DX2TEV = 1 and can generate a protocol interrupt if PCRL.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.

### **21.6.4 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  $SCTRL.TRM = 11_B$  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 (DIN) with  $DX0CR.INSW = 1$ . Configure a transmit data output pin (DOUT) 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 DOUT signal to a pin).
- **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 ( $PCRL.WAGEN = 0$ ).

#### **21.6.4.1 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  $PCRL.DX2TIEN = 1$ .

## 21.6.5 IIS Protocol Registers

In IIS mode, the registers PCRL, PCRH and PSR handle IIS related information.

### 21.6.5.1 IIS Protocol Control Registers

In IIS mode, the PCRL/PCRH register bits or bit fields are defined as described in this section.

#### PCRL

##### Protocol Control Register L [IIS Mode]

(40<sub>H</sub>)

Reset Value: 0000<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>DX2 TIEN</b>									<b>END IEN</b>	<b>WAR EIEN</b>	<b>WAF EIEN</b>	<b>0</b>	<b>SELI NV</b>	<b>DT EN</b>	<b>WAG EN</b>
rw									rw	rw	rw	r	rw	rw	rw

Field	Bits	Type	Description
<b>WAGEN</b>	0	rw	<b>WA Generation Enable</b> This bit enables/disables the generation of word address control output signal WA. 0 <sub>B</sub> 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 PCRH.MCLK. 1 <sub>B</sub> 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 PCRH.MCLK. After clearing WAGEN, the USIC module stops the generation of the WA signal within the next 4 WA periods.
<b>DTEN</b>	1	rw	<b>Data Transfers Enable</b> This bit enables/disables the transfer of IIS frames as a reaction to changes of the input word address control line WA. 0 <sub>B</sub> The changes of the WA input signal are ignored and no transfers take place. 1 <sub>B</sub> Transfers are enabled.



**Universal Serial Interface Channel**

Field	Bits	Type	Description
<b>SELINV</b>	2	rw	<b>Select Inversion</b> This bit defines if the polarity of the SELOx outputs in relation to the internally generated word address signal WA. 0 <sub>B</sub> The SELOx outputs have the same polarity as the WA signal. 1 <sub>B</sub> The SELOx outputs have the inverted polarity to the WA signal.
<b>WAFEIEN</b>	4	rw	<b>WA Falling Edge Interrupt Enable</b> This bit enables/disables the activation of a protocol interrupt when a falling edge of WA has been generated. 0 <sub>B</sub> A protocol interrupt is not activated if a falling edge of WA is generated. 1 <sub>B</sub> A protocol interrupt is activated if a falling edge of WA is generated.
<b>WAREIEN</b>	5	rw	<b>WA Rising Edge Interrupt Enable</b> This bit enables/disables the activation of a protocol interrupt when a rising edge of WA has been generated. 0 <sub>B</sub> A protocol interrupt is not activated if a rising edge of WA is generated. 1 <sub>B</sub> A protocol interrupt is activated if a rising edge of WA is generated.
<b>ENDIEN</b>	6	rw	<b>END Interrupt Enable</b> This bit enables/disables the activation of a protocol interrupt when the WA generation stops after clearing PCR.WAGEN (complete system word length is processed before stopping). 0 <sub>B</sub> A protocol interrupt is not activated. 1 <sub>B</sub> A protocol interrupt is activated.
<b>DX2TIEN</b>	15	rw	<b>DX2T Interrupt Enable</b> This bit enables/disables the generation of a protocol interrupt if the DX2T signal becomes activated (indicated by PSR.DX2TEV = 1). 0 <sub>B</sub> A protocol interrupt is not generated if DX2T is active. 1 <sub>B</sub> A protocol interrupt is generated if DX2T is active.
<b>0</b>	3, [14:7]	rw	<b>Reserved</b> Returns 0 if read; should be written with 0;

**PCRH**

**Protocol Control Register H [IIS Mode]**

**(42<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>M CLK</b>					<b>0</b>							<b>TDEL</b>			
rw					rw							rw			

Field	Bits	Type	Description
<b>TDEL</b>	[5:0]	rw	<b>Transfer Delay</b> 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.
<b>0</b>	[14:6]	rw	<b>Reserved</b> Returns 0 if read; should be written with 0.
<b>MCLK</b>	15	rw	<b>Master Clock Enable</b> This bit enables generation of the master clock MCLK (not directly used for IIC protocol, can be used as general frequency output). 0 <sub>B</sub> The MCLK generation is disabled and MCLK is 0. 1 <sub>B</sub> The MCLK generation is enabled.

### 21.6.5.2 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 doesn't 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] (44<sub>H</sub>)** **Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>AIF</b>	<b>RIF</b>	<b>TBIF</b>	<b>TSIF</b>	<b>DLIF</b>	<b>RSIF</b>	<b>0</b>		<b>END</b>	<b>WA RE</b>	<b>WA FE</b>	<b>DX2 TEV</b>	<b>0</b>		<b>DX2 S</b>	<b>WA</b>
rwh	rwh	rwh	rwh	rwh	rwh	r		rwh	rwh	rwh	rwh	r		rwh	rwh

Field	Bits	Type	Description
<b>WA</b>	0	rwh	<b>Word Address</b> 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 RBUFSTR[9] to distinguish between data received for the right and the left channel. 0 <sub>B</sub> WA has been sampled 0. 1 <sub>B</sub> WA has been sampled 1.
<b>DX2S</b>	1	rwh	<b>DX2S Status</b> This bit indicates the current status of the DX2S signal, which is used as word address signal WA. 0 <sub>B</sub> DX2S is 0. 1 <sub>B</sub> DX2S is 1.
<b>DX2TEV</b>	3	rwh	<b>DX2T Event Detected<sup>1)</sup></b> This bit indicates that the DX2T signal has been activated. In IIS slave mode, an activation of DX2T generates a protocol interrupt if PCRL.DX2TIEN = 1. 0 <sub>B</sub> The DX2T signal has not been activated. 1 <sub>B</sub> The DX2T signal has been activated.

**Universal Serial Interface Channel**

<b>Field</b>	<b>Bits</b>	<b>Type</b>	<b>Description</b>
<b>WAFE</b>	4	rwh	<b>WA Falling Edge Event<sup>1)</sup></b> This bit indicates that a falling edge of the WA output signal has been generated. This event generates a protocol interrupt if PCRL.WAFEIEN = 1. 0 <sub>B</sub> A WA falling edge has not been generated. 1 <sub>B</sub> A WA falling edge has been generated.
<b>WARE</b>	5	rwh	<b>WA Rising Edge Event<sup>1)</sup></b> This bit indicates that a rising edge of the WA output signal has been generated. This event generates a protocol interrupt if PCRL.WAREIEN = 1. 0 <sub>B</sub> A WA rising edge has not been generated. 1 <sub>B</sub> A WA rising edge has been generated.
<b>END</b>	6	rwh	<b>WA Generation End<sup>1)</sup></b> This bit indicates that the WA generation has ended after clearing PCRL.WAGEN. This bit should be cleared by software before clearing WAGEN. 0 <sub>B</sub> The WA generation has not yet ended (if it is running and WAGEN has been cleared). 1 <sub>B</sub> The WA generation has ended (if it has been running).
<b>RSIF</b>	10	rwh	<b>Receiver Start Indication Flag</b> 0 <sub>B</sub> A receiver start event has not occurred. 1 <sub>B</sub> A receiver start event has occurred.
<b>DLIF</b>	11	rwh	<b>Data Lost Indication Flag</b> 0 <sub>B</sub> A data lost event has not occurred. 1 <sub>B</sub> A data lost event has occurred.
<b>TSIF</b>	12	rwh	<b>Transmit Shift Indication Flag</b> 0 <sub>B</sub> A transmit shift event has not occurred. 1 <sub>B</sub> A transmit shift event has occurred.
<b>TBIF</b>	13	rwh	<b>Transmit Buffer Indication Flag</b> 0 <sub>B</sub> A transmit buffer event has not occurred. 1 <sub>B</sub> A transmit buffer event has occurred.
<b>RIF</b>	14	rwh	<b>Receive Indication Flag</b> 0 <sub>B</sub> A receive event has not occurred. 1 <sub>B</sub> A receive event has occurred.
<b>AIF</b>	15	rwh	<b>Alternative Receive Indication Flag</b> 0 <sub>B</sub> An alternative receive event has not occurred. 1 <sub>B</sub> An alternative receive event has occurred.

**Universal Serial Interface Channel**

Field	Bits	Type	Description
0	2, [9:7]	r	<b>Reserved</b> Returns 0 if read; not modified in IIS mode.

- 1) This status bit can generate a protocol interrupt (see [Page 21-24](#)). The general interrupt status flags are described in the general interrupt chapter.

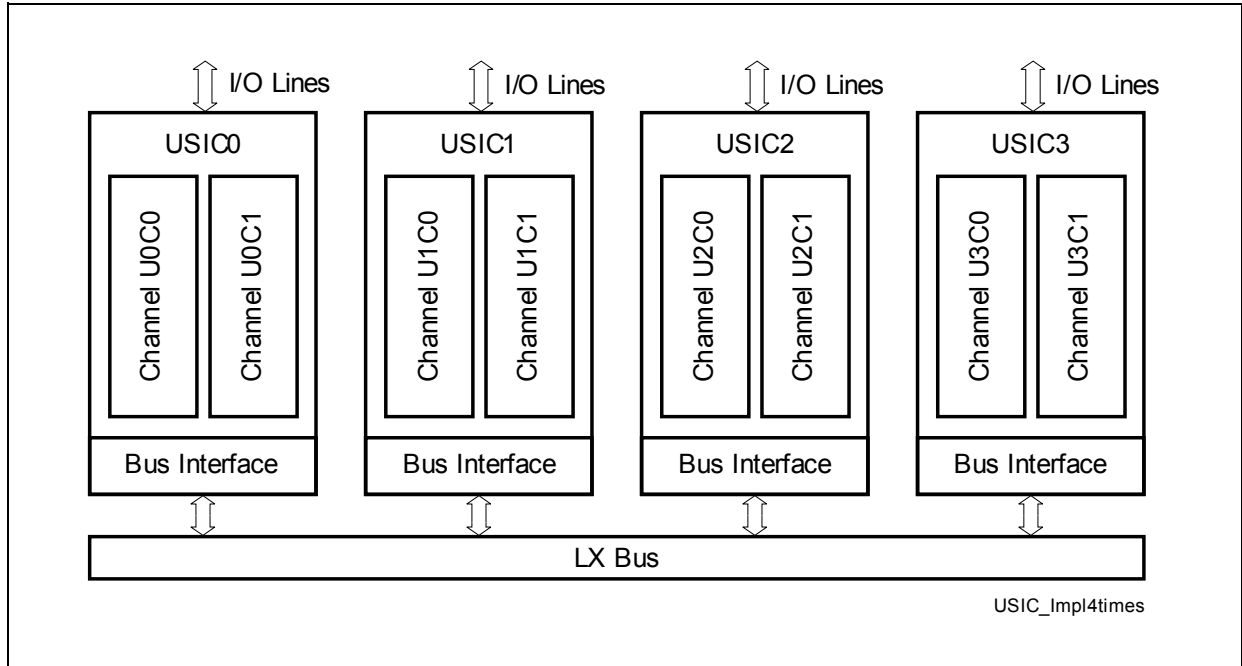
## 21.7 USIC Implementation in XE16xyM

This section describes the implementation specific details of the USIC modules in the XE16xyM. It contains details about:

- Implementation Overview (see [Page 21-205](#))
- Channel Features (see [Page 21-206](#))
- Address Map (see [Page 21-207](#))
- Module Identification Registers (see [Page 21-208](#))
- Interrupt Control Registers (see [Page 21-210](#))
- Input/Output Connections (see [Page 21-212](#))
- USIC Module 0 I/O Lines (see [Page 21-213](#))
- USIC Module 1 I/O Lines (see [Page 21-216](#))
- USIC Module 2 I/O Lines (see [Page 21-219](#))
- USIC Module 3 I/O Lines (see [Page 21-222](#))

### 21.7.1 Implementation Overview

The XE16xyM devices can contain up to four identical USIC modules (USIC0, USIC1, USIC2, and USIC3) with 2 communication channels each. Which modules are available for a particular product, is described in the corresponding data sheet.



**Figure 21-54 USIC Module Structure in XE16xyM**

## 21.7.2 Channel Features

The USIC channels in the XE16xyM support the following functionality:

**Table 21-11 USIC Module Feature Set**

Channel	ASC Protocol	LIN Support	SSC Protocol	IIC Protocol	IIS Protocol	FIFO Buffer Entries	SELOx <sup>1)</sup>
<b>U0C0</b>	yes	yes	yes	yes	yes	64 shared	8
<b>U0C1</b>	yes	yes	yes	yes	yes		4
<b>U1C0</b>	yes	yes	yes	yes	yes	64 shared	8
<b>U1C1</b>	yes	yes	yes	yes	yes		5
<b>U2C0</b>	yes	yes	yes	yes	yes	64 shared	6
<b>U2C1</b>	yes	yes	yes	yes	yes		3
<b>U3C0</b>	yes	yes	yes	yes	yes	64 shared	4
<b>U3C1</b>	yes	yes	yes	yes	yes		2

1) This number refers to the maximum number of signals available in the 144-pin package. Please note that some of these signals may overlap with others. As a result, not all signals are necessarily available in parallel.

### 21.7.3 Address Map

The registers of the USIC communication channels are available at the following base addresses. The exact register address is given by the relative address of the register (given in [Table 21-3](#)) plus the channel base address (given in [Table 21-12](#)).

**Table 21-12 Registers Address Space**

Module	Base Address	End Address	Note
U0C0	204000 <sub>H</sub>	2041FF <sub>H</sub>	Standard locations
U0C1	204200 <sub>H</sub>	2043FF <sub>H</sub>	Standard locations
U1C0	204800 <sub>H</sub>	2049FF <sub>H</sub>	Standard locations
U1C1	204A00 <sub>H</sub>	204BFF <sub>H</sub>	Standard locations
U2C0	205000 <sub>H</sub>	2051FF <sub>H</sub>	Standard locations
U2C1	205200 <sub>H</sub>	2053FF <sub>H</sub>	Standard locations
U3C0	205800 <sub>H</sub>	2059FF <sub>H</sub>	Standard locations
U3C1	205A00 <sub>H</sub>	205BFF <sub>H</sub>	Standard locations
U0C0A	20B000 <sub>H</sub>	20B1FF <sub>H</sub>	Alternate locations
U0C1A	20B200 <sub>H</sub>	20B3FF <sub>H</sub>	Alternate locations
U1C0A	20B400 <sub>H</sub>	20B5FF <sub>H</sub>	Alternate locations
U1C1A	20B600 <sub>H</sub>	20B7FF <sub>H</sub>	Alternate locations
U2C0A	20B800 <sub>H</sub>	20B9FF <sub>H</sub>	Alternate locations
U2C1A	20BA00 <sub>H</sub>	20BBFF <sub>H</sub>	Alternate locations
U3C0A	20BC00 <sub>H</sub>	20BDFF <sub>H</sub>	Alternate locations
U3C1A	20BE00 <sub>H</sub>	20BFFF <sub>H</sub>	Alternate locations



## 21.7.4 Module Identification Registers

The module identification registers indicate the function and the design step of the USIC modules.

### USIC0\_IDL

Module Identification Register L

(204008<sub>H</sub>)

Reset Value: C0XX<sub>H</sub>

### USIC1\_IDL

Module Identification Register L

(204808<sub>H</sub>)

Reset Value: C0XX<sub>H</sub>

### USIC2\_IDL

Module Identification Register L

(205008<sub>H</sub>)

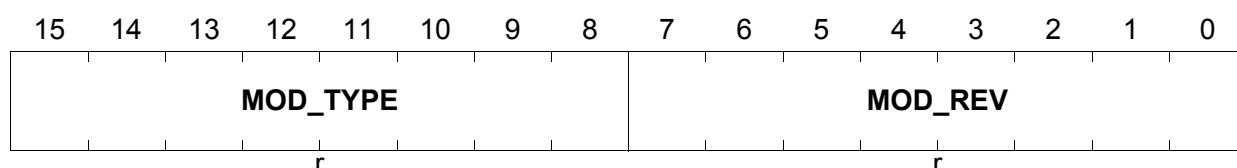
Reset Value: C0XX<sub>H</sub>

### USIC3\_IDL

Module Identification Register L

(205808<sub>H</sub>)

Reset Value: C0XX<sub>H</sub>



Field	Bits	Type	Description
<b>MOD_REV</b>	[7:0]	r	<b>Module Revision Number</b> MOD_REV defines the revision number. The value of a module revision starts with 01 <sub>H</sub> (first revision).
<b>MOD_TYPE</b>	[15:8]	r	<b>Module Type</b> This bit field is C0 <sub>H</sub> . It defines the module as a 32-bit module.

**Universal Serial Interface Channel**

**USIC0\_IDH**

**Module Identification Register H**

(20400A<sub>H</sub>)

**Reset Value: 003A<sub>H</sub>**

**USIC1\_IDH**

**Module Identification Register H**

(20480A<sub>H</sub>)

**Reset Value: 003A<sub>H</sub>**

**USIC2\_IDH**

**Module Identification Register H**

(20500A<sub>H</sub>)

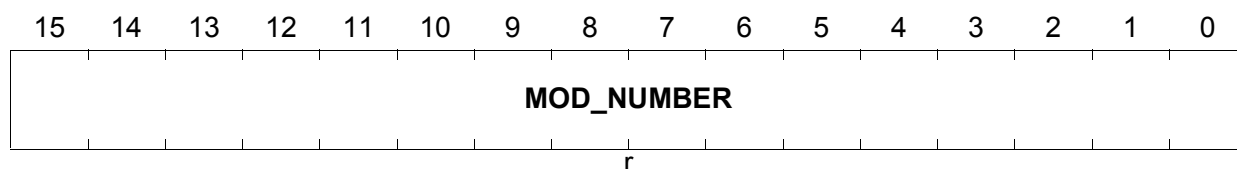
**Reset Value: 003A<sub>H</sub>**

**USIC3\_IDH**

**Module Identification Register H**

(20580A<sub>H</sub>)

**Reset Value: 003A<sub>H</sub>**



Field	Bits	Type	Description
<b>MOD_NUMBER</b>	[15:0]	r	<b>Module Number Value</b> This bit field defines the USIC module identification number (003A <sub>H</sub> = USIC).

### 21.7.5 Interrupt Control Registers

Each USIC channel provides 4 service request outputs SR[3:0] (not all of them are necessarily connected to independent interrupt registers UxCy\_nIC). **Table 21-13** shows the assignment of the service request outputs to the interrupt control registers.

Each USIC communication channel is connected to 3 dedicated interrupt control registers (connected to UxCy\_SR[2:0], e.g. one for transmission, one for reception, the third one for protocol or error handling, or for the alternative receive events). A fourth interrupt control register per communication channel (connected to UxCy\_SR3) is shared with module CC2.

The interrupt control registers are located in the SFR area. They are described in the general interrupt chapter.

**Table 21-13 USIC Interrupt Control Registers**

<b>Service Request Output Line</b>	<b>Interrupt Control Register/Bit</b>
SR0 of USIC0 channel 0	U0C0_0IC
SR1 of USIC0 channel 0	U0C0_1IC
SR2 of USIC0 channel 0	U0C0_2IC
SR3 of USIC0 channel 0	SCU_ISSR.4, shared with CC2
SR0 of USIC0 channel 1	U0C1_0IC
SR1 of USIC0 channel 1	U0C1_1IC
SR2 of USIC0 channel 1	U0C1_2IC
SR3 of USIC0 channel 1	SCU_ISSR.5, shared with CC2
SR0 of USIC1 channel 0	U1C0_0IC
SR1 of USIC1 channel 0	U1C0_1IC
SR2 of USIC1 channel 0	U1C0_2IC
SR3 of USIC1 channel 0	SCU_ISSR.6, shared with CC2
SR0 of USIC1 channel 1	U1C1_0IC
SR1 of USIC1 channel 1	U1C1_1IC
SR2 of USIC1 channel 1	U1C1_2IC
SR3 of USIC1 channel 1	SCU_ISSR.7, shared with CC2
SR0 of USIC2 channel 0	U2C0_0IC
SR1 of USIC2 channel 0	U2C0_1IC
SR2 of USIC2 channel 0	U2C0_2IC
SR3 of USIC2 channel 0	SCU_ISSR.12, shared with CC2

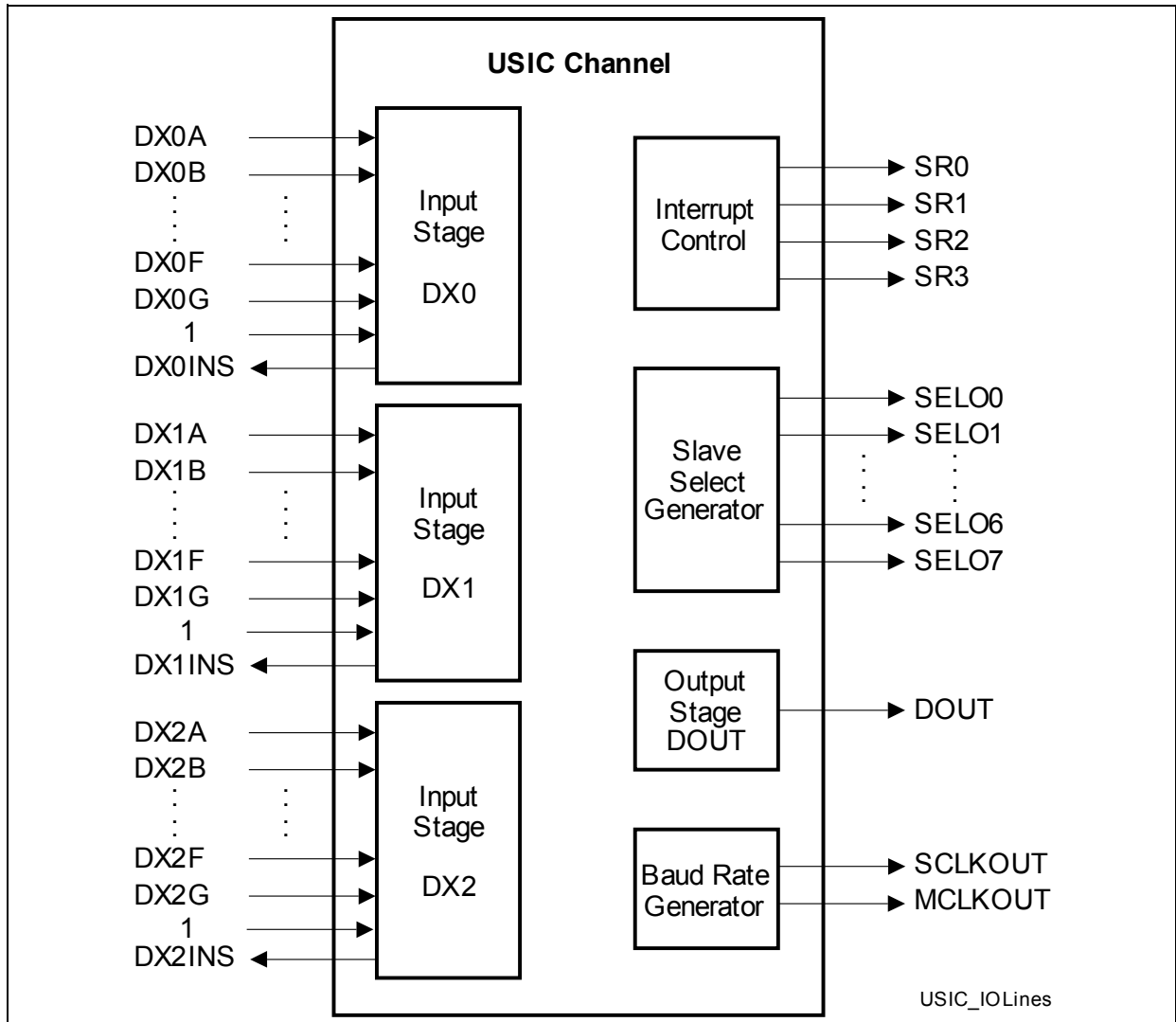
**Universal Serial Interface Channel**

**Table 21-13 USIC Interrupt Control Registers (cont'd)**

<b>Service Request Output Line</b>	<b>Interrupt Control Register/Bit</b>
SR0 of USIC2 channel 1	U2C1_0IC
SR1 of USIC2 channel 1	U2C1_1IC
SR2 of USIC2 channel 1	U2C1_2IC
SR3 of USIC2 channel 1	SCU_ISSR.13, shared with CC2
SR0 of USIC3 channel 0	U3C0_0IC
SR1 of USIC3 channel 0	U3C0_1IC
SR2 of USIC3 channel 0	U3C0_2IC
SR3 of USIC3 channel 0	SCU_ISSR.2, shared with CC2
SR0 of USIC3 channel 1	U3C1_0IC
SR1 of USIC3 channel 1	U3C1_1IC
SR2 of USIC3 channel 1	U3C1_2IC
SR3 of USIC3 channel 1	SCU_ISSR.3, shared with CC2

## 21.7.6 Input/Output Connections

**Figure 21-1** 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 XE16xyM device. Naming convention: UxCy refers to USIC module x channel y.



**Figure 21-55 USIC Channel I/O Lines**

The connections of the service request outputs SR[3:0] to the interrupt control registers are defined in [Table 21-13](#) on [Page 21-210](#).

### 21.7.6.1 USIC Module 0 I/O Lines

The signals of USIC module 0 have the prefix “U0C0\_” for channel 0 and “U0C1\_” for channel 1.

**Table 21-14 I/O Connections of USIC0**

Signal	Used as	From/To	
		Channel 0 – U0C0	Channel 1 – U0C1
Data Inputs			
DX0A	Shift data input	P10.0	P10.0
DX0B	Shift data input	P10.1	P10.7
DX0C	Shift data input	P10.6	P10.14
DX0D	Shift data input	P7.4	P2.3
DX0E	Shift data input	P2.3	P2.10
DX0F	Shift data input	P2.4	P7.3
DX0G	Loop back data shift input	U0C0_DOUT	U0C1_DOUT
Clock Inputs			
DX1A	Shift clock input	P10.1	P10.10
DX1B	Shift clock input	P10.2	P10.5
DX1C	Shift clock input	P10.8	P10.15
DX1D	Shift clock input	P2.5	P2.8
DX1E	Shift clock input	0	P7.4
DX1F	Input for single wire ASC collision detection	U0C0_DX0INS	U0C1_DX0INS
DX1G	Loop back shift clock input	U0C0_SCLKOUT	U0C1_SCLKOUT
Control Inputs			
DX2A	Shift control input	P10.3	P10.3
DX2B	Shift control input	P10.4	P10.4
DX2C	Shift control input	P10.10	P2.7
DX2D	Shift control input	P2.6	0
DX2E	Input for transmit data validation	CC2_CC24	RTC_T14INT
DX2F	Input for transmit data validation	CCU60_T13_PM	CCU60_T13_PM
DX2G	Loop back shift control input	U0C0_SELO0	U0C1_SELO0

**Universal Serial Interface Channel**

**Table 21-14 I/O Connections of USIC0 (cont'd)**

Signal	Used as	From/To	
		Channel 0 – U0C0	Channel 1 – U0C1
Data Outputs			
DOUT	Shift data output	P2.3	P2.9
		P7.3	P2.10
		P10.1	P7.3
		P10.6	P7.4
		–	P10.0
		–	P10.7
		–	P10.14
		–	P10.15
		–	P4.3
		–	P2.4
Clock Outputs			
MCLKOUT	Master clock output, e.g. for IIS	P10.8	P10.9
SCLKOUT	Shift clock output	P2.5	P2.8
		P10.2	P7.4
		–	P10.5
Control Outputs			
SELO0	Shift control output 0	P2.6	P2.7
		P10.10	P10.8
		U0C0_DX2G	U0C1_DX2G
SELO1	Shift control output 1	P2.7	P2.6
SELO2	Shift control output 2	P2.11	P2.11
SELO3	Shift control output 3	P2.10	P2.12
		P10.4	–
SELO4	Shift control output 5	P2.12	–
		P3.4	–
		P10.9	–
SELO5	Shift control output 5	P3.5	–
SELO6	Shift control output 6	P3.6	–

**Universal Serial Interface Channel**

**Table 21-14 I/O Connections of USIC0 (cont'd)**

Signal	Used as	From/To	
		Channel 0 – U0C0	Channel 1 – U0C1
SELO7	Shift control output 7	P3.7	–
<b>System Related Outputs</b>			
DX0INS	External interrupt input for ERU (SCU)	ERU_0A2	ERU_0B2
	Single wire ASC collision detection	U0C0_DX1F	U0C1_DX1F
DX1INS	–	–	–
DX2INS	External interrupt input for ERU (SCU)	ERU_0A3	ERU_0B3
<b>Loop Back Outputs</b>			
DOUT	Loop back shift data output	U0C0_DX0G	U0C1_DX0G
SCLKOUT	Loop back shift clock output	U0C0_DX1G	U0C1_DX1G
SELO0	Loop back shift control output	U0C0_DX2G	U0C1_DX2G



### 21.7.6.2 USIC Module 1 I/O Lines

The signals of USIC module 1 have the prefix “U1C0\_” for channel 0 and “U1C1\_” for channel 1.

**Table 21-15 I/O Connections of USIC1**

Signal	Used as	From/To	
		Channel 0 – U1C0	Channel 1 – U1C1
Data Inputs			
DX0A	Shift data input	P0.0	P0.6
DX0B	Shift data input	P0.1	P0.7
DX0C	Shift data input	P10.14	ESR1
DX0D	Shift data input	P2.3	ESR2
DX0E	Shift data input	ESR0	P6.0
DX0F	Shift data input	ESR1	CAN1INS
DX0G	Loop back data shift input	U1C0_DOUT	U1C1_DOUT
Clock Inputs			
DX1A	Shift clock input	P0.1	P0.5
DX1B	Shift clock input	P0.2	P0.6
DX1C	Shift clock input	P0.5	P6.2
DX1D	Shift clock input	P10.11	0
DX1E	Shift clock input	P10.12	0
DX1F	Input for single wire ASC collision detection	U1C0_DX0INS	U1C1_DX0INS
DX1G	Loop back shift clock input	U1C0_SCLKOUT	U1C1_SCLKOUT
Control Inputs			
DX2A	Shift control input	P0.3	P0.4
DX2B	Shift control input	ESR0	ESR1
DX2C	Shift control input	ESR1	ESR2
DX2D	Shift control input	P10.6	P6.3
DX2E	Input for transmit data validation	CC2_CC25	RTC_T14INT
DX2F	Input for transmit data validation	CCU61_T13_PM	CCU61_T13_PM
DX2G	Loop back shift control input	U1C0_SELO0	U1C1_SELO0

**Universal Serial Interface Channel**

**Table 21-15 I/O Connections of USIC1 (cont'd)**

Signal	Used as	From/To	
		Channel 0 – U1C0	Channel 1 – U1C1
Data Outputs			
DOUT	Shift data output	P0.0	P0.6
		P0.1	P0.7
		P10.12	P6.1
		P10.13	P8.2
		P10.15	–
Clock Outputs			
MCLKOUT	Master clock output, e.g. for IIS	P1.0	P1.7
SCLKOUT	Shift clock output	P0.2	P0.5
		P10.11	P6.2
Control Outputs			
SELO0	Shift control output 0	P0.3	P0.4
		P10.6	P6.3
SELO1	Shift control output 1	P0.4	P0.3
		P10.14	–
SELO2	Shift control output 2	P0.5	P1.6
		P10.15	–
SELO3	Shift control output 3	P0.7	P1.5
		P10.13	–
SELO4	Shift control output 5	P1.0	P1.4
SELO5	Shift control output 5	P1.1	–
SELO6	Shift control output 6	P1.2	–
SELO7	Shift control output 7	P1.3	–
System Related Outputs			
DX0INS	External interrupt input for ERU (SCU)	ERU_1A2	ERU_1B2
	Single wire ASC collision detection	U1C0_DX1F	U1C1_DX1F
DX1INS	External interrupt input for ERU (SCU)	ERU_3B0	–

**Universal Serial Interface Channel**

**Table 21-15 I/O Connections of USIC1 (cont'd)**

Signal	Used as	From/To	
		Channel 0 – U1C0	Channel 1 – U1C1
DX2INS	External interrupt input for ERU (SCU)	ERU_1A3	ERU_1B3
<b>Loop Back Outputs</b>			
DOUT	Loop back shift data output	U1C0_DX0G	U1C1_DX0G
SCLKOUT	Loop back shift clock output	U1C0_DX1G	U1C1_DX1G
SELO0	Loop back shift control output	U1C0_DX2G	U1C1_DX2G

### 21.7.6.3 USIC Module 2 I/O Lines

The signals of USIC module 2 have the prefix “U2C0\_” for channel 0 and “U2C1\_” for channel 1.

**Table 21-16 I/O Connections of USIC2**

Signal	Used as	From/To	
		Channel 0 – U2C0	Channel 1 – U2C1
Data Inputs			
DX0A	Shift data input	P3.0	P3.6
DX0B	Shift data input	P3.1	P3.7
DX0C	Shift data input	P1.5	P1.1
DX0D	Shift data input	P1.6	P1.2
DX0E	Shift data input	P9.5	ESR2
DX0F	Shift data input	P5.8	P5.10
DX0G	Loop back data shift input	U2C0_DOUT	U2C1_DOUT
Clock Inputs			
DX1A	Shift clock input	P3.0	P3.5
DX1B	Shift clock input	P3.2	P3.6
DX1C	Shift clock input	P1.7	P1.2
DX1D	Shift clock input	P9.7	0
DX1E	Shift clock input	0	0
DX1F	Input for single wire ASC collision detection	U2C0_DX0INS	U2C1_DX0INS
DX1G	Loop back shift clock input	U2C0_SCLKOUT	U2C1_SCLKOUT
Control Inputs			
DX2A	Shift control input	P3.3	P3.4
DX2B	Shift control input	P1.4	ESR2
DX2C	Shift control input	0	ESR1
DX2D	Shift control input	0	0
DX2E	Input for transmit data validation	CC2_CC26	RTC_T14INT
DX2F	Input for transmit data validation	CCU62_T13_PM	CCU62_T13_PM
DX2G	Loop back shift control input	U2C0_SELO0	U2C1_SELO0

**Universal Serial Interface Channel**

**Table 21-16 I/O Connections of USIC2 (cont'd)**

Signal	Used as	From/To	
		Channel 0 – U2C0	Channel 1 – U2C1
Data Outputs			
DOUT	Shift data output	P3.0	P3.6
		P3.1	P3.7
		P1.6	P1.1
		P9.4	P10.8
		P9.5	–
		P10.5	–
Clock Outputs			
MCLKOUT	Master clock output, e.g. for IIS	–	–
SCLKOUT	Shift clock output	P3.2	P3.5
		P1.7	P1.2
Control Outputs			
SELO0	Shift control output 0	P3.3	P3.4
SELO1	Shift control output 1	P3.4	P3.3
SELO2	Shift control output 2	P3.5	P2.13
SELO3	Shift control output 3	P3.7	–
SELO4	Shift control output 4	P1.3	–
SELO5	Shift control output 5	P1.4	–
SELO6	Shift control output 6	–	–
SELO7	Shift control output 7	–	–
System Related Outputs			
DX0INS	External interrupt input for ERU (SCU)	ERU_2A2	ERU_2B2
	Single wire ASC collision detection	U2C0_DX1F	U2C1_DX1F
DX1INS	External interrupt input for ERU (SCU)	ERU_2B0	–
DX2INS	External interrupt input for ERU (SCU)	ERU_2A3	ERU_2B3

**Universal Serial Interface Channel**

**Table 21-16 I/O Connections of USIC2 (cont'd)**

Signal	Used as	From/To	
		Channel 0 – U2C0	Channel 1 – U2C1
Loop Back Outputs			
DOUT	Loop back shift data output	U2C0_DX0G	U2C1_DX0G
SCLKOUT	Loop back shift clock output	U2C0_DX1G	U2C1_DX1G
SELO0	Loop back shift control output	U2C0_DX2G	U2C1_DX2G

#### 21.7.6.4 USIC Module 3 I/O Lines

The signals of USIC module 3 have the prefix “U3C0\_” for channel 0 and “U3C1\_” for channel 1.

**Table 21-17 I/O Connections of USIC3**

Signal	Used as	From/To	
		Channel 0 – U3C0	Channel 1 – U3C1
Data Inputs			
DX0A	Shift data input	P10.3	P2.10
DX0B	Shift data input	P4.5	P11.4
DX0C	Shift data input	0	0
DX0D	Shift data input	0	0
DX0E	Shift data input	0	0
DX0F	Shift data input	0	0
DX0G	Loop back data shift input	U3C0_DOUT	U3C1_DOUT
Clock Inputs			
DX1A	Shift clock input	P10.14	P11.0
DX1B	Shift clock input	P4.2	0
DX1C	Shift clock input	0	0
DX1D	Shift clock input	0	0
DX1E	Shift clock input	0	0
DX1F	Input for single wire ASC collision detection	U3C0_DX0INS	U3C1_DX0INS
DX1G	Loop back shift clock input	U3C0_SCLKOUT	U3C1_SCLKOUT
Control Inputs			
DX2A	Shift control input	P10.11	P11.1
DX2B	Shift control input	P10.2	P11.5
DX2C	Shift control input	P4.4	0
DX2D	Shift control input	0	0
DX2E	Input for transmit data validation	CC2_CC27	RTC_T14INT
DX2F	Input for transmit data validation	CCU63_T13_PM	CCU62_T13_PM
DX2G	Loop back shift control input	U3C0_SELO0	U3C1_SELO0

**Universal Serial Interface Channel**

**Table 21-17 I/O Connections of USIC3 (cont'd)**

Signal	Used as	From/To	
		Channel 0 – U3C0	Channel 1 – U3C1
Data Outputs			
DOUT	Shift data output	P10.4	P2.11
		P4.5	P11.4
		P4.6	P11.2
		–	–
		–	–
Clock Outputs			
MCLKOUT	Master clock output, e.g. for IIS	–	–
SCLKOUT	Shift clock output	P10.14	P11.0
		P4.2	–
Control Outputs			
SELO0	Shift control output 0	P10.11	P11.1
SELO1	Shift control output 1	P10.2	P11.5
SELO2	Shift control output 2	P4.4	–
SELO3	Shift control output 3	P4.1	–
SELO4	Shift control output 4	–	–
SELO5	Shift control output 5	–	–
SELO6	Shift control output 6	–	–
SELO7	Shift control output 7	–	–
System Related Outputs			
DX0INS	External interrupt input for ERU (SCU)	–	–
	Single wire ASC collision detection	U3C0_DX1F	U3C1_DX1F
DX1INS	External interrupt input for ERU (SCU)	–	–
DX2INS	External interrupt input for ERU (SCU)	–	–



**Universal Serial Interface Channel**

**Table 21-17 I/O Connections of USIC3 (cont'd)**

Signal	Used as	From/To	
		Channel 0 – U3C0	Channel 1 – U3C1
Loop Back Outputs			
DOUT	Loop back shift data output	U3C0_DX0G	U3C1_DX0G
SCLKOUT	Loop back shift clock output	U3C0_DX1G	U3C1_DX1G
SELO0	Loop back shift control output	U3C0_DX2G	U3C1_DX2G

## 22 Controller Area Network (MultiCAN) Controller

This chapter describes the MultiCAN controller of the XE16xyM. It contains the following sections:

- Overview of the MultiCAN Kernel (see [Section 22.1](#))
- Functional description of the MultiCAN Kernel (see [Section 22.2](#))
- XE16xyM implementation specific details and registers of the MultiCAN controller (port connections and control, interrupt control, address decoding, clock control, see [Section 22.5](#)).

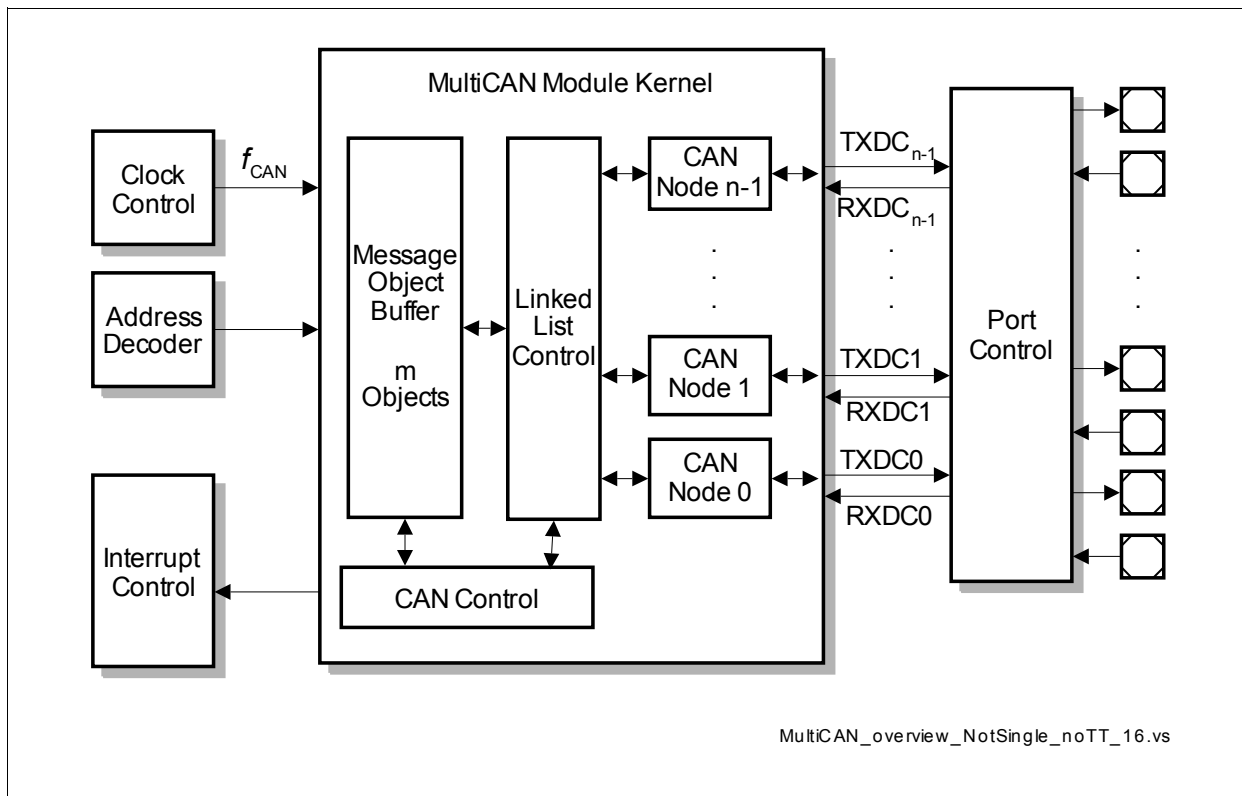
*Note: The MultiCAN kernel register names described in this chapter will be referenced in the XE16xyM User's Manual by the module name prefix "CAN\_".*

### 22.1 MultiCAN Short Description

This section describes the serial communication interfaces CAN (Controller Area Network) of the communication module MultiCAN of the XE16xyM.

#### 22.1.1 Overview

The MultiCAN module contains up to 6 independent CAN nodes, representing the communication interfaces.



**Figure 22-1 Overview of the MultiCAN Module**

## **22.1.2 CAN Features**

Several key features contribute to the high performance of the MultiCAN module:

- CAN functionality conforms to CAN specification V2.0 B active for each CAN node (compliant to ISO 11898)
- Up to 6 independent CAN nodes available
- Up to 256 independent message objects (shared by the CAN nodes)
- Dedicated control registers for each CAN node
- Data transfer rate up to 1MBaud, individually programmable for each node
- Flexible and powerful message transfer control and error handling capabilities
- Full-CAN functionality: message objects can be individually
  - Assigned to one of the 6 CAN nodes
  - Configured as transmit or receive object
  - Participate in a message buffer with FIFO algorithm
  - Set up to handle frames with 11-bit or 29-bit identifiers
  - Provided with programmable acceptance mask register for filtering
  - Monitored via a frame counter
  - Configured to Remote Monitoring Mode
- Automatic gateway mode support
- 16 individually programmable interrupt outputs
- CAN Analyzer Mode for bus monitoring
- SRAMs in MultiCAN module optionally parity error protected

## 22.2 CAN Functional Description

This section describes the core features of the CAN module.

### 22.2.1 Conventions and Definitions

**Table 22-1** defines constants that are used throughout the MultiCAN specification. These are fixed maximum values for a given MultiCAN implementation. Nevertheless, in different products, some objects and nodes may be disabled, depending on the product configuration.

**Table 22-1 Fixed Module Constants**

Constant	Value	Description
<b>n_objects</b>	256	<b>Number of Message Objects</b> n_objects denotes the total amount of message objects available.
<b>n_interrupts</b>	16	<b>Number of Interrupt Output Lines</b> n_interrupts denotes the total number of interrupt outputs available.
<b>n_pendings</b>	256	<b>Number of Message Pending Bits</b> n_pendings denotes the number of message pending bits available. The number of message pending registers is given by $n\_pendings/32$ .
<b>n_lists</b>	8	<b>Number of Lists</b> n_lists denotes the total number of lists available for allocation of message number.
<b>n_nodes</b>	6	<b>Number of CAN Nodes Available</b> n_nodes denotes the total number of CAN nodes available. As each CAN node has it's own list in addition to the list of un-allocated elements, the relation $n\_nodes < n\_lists$ is true.

### 22.2.2 Introduction

The MultiCAN module contains 6 Full-CAN nodes operating independently or exchanging data and remote frames via a gateway function. Transmission and reception of CAN frames is handled in accordance to CAN specification V2.0part B (active). Each CAN node can receive and transmit standard frames with 11-bit identifiers as well as extended frames with 29-bit identifiers.

### **22.2.2.1 Feature Overview**

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

The message objects are organized in double chained lists, where each CAN node has it's own list of message objects. A CAN node stores frames only into message objects that are allocated to the list of the CAN node. It only transmits messages from objects of this list.

A powerful, command driven list controller performs all list operations.

The bit timings for the CAN nodes are derived from the peripheral clock ( $f_{CAN}$ ) and are programmable up to a data rate of 1 MBaud. A pair of receive and transmit pins connects each CAN node to a bus transceiver.

#### **Features**

- Compliant to ISO 11898.
- CAN functionality according to CAN specification V2.0 B active.
- Dedicated control registers are provided for each CAN node.
- A data transfer rate up to 1 MBaud is supported.
- Flexible and powerful message transfer control and error handling capabilities are implemented.
- Advanced CAN bus bit timing analysis and baud rate detection can be performed for each CAN node via the frame counter.
- Full-CAN functionality: A set of 256 message objects can be individually
  - allocated (assigned) to any CAN node
  - configured as transmit or receive object
  - setup to handle frames with 11-bit or 29-bit identifier
  - counted or assigned a timestamp via a frame counter
  - configured to remote monitoring mode
- Advanced Acceptance Filtering:
  - Each message object provides an individual acceptance mask to filter incoming frames.
  - A message object can be configured to accept only standard or only extended frames or to accept both standard and extended frames.
  - Message objects can be grouped into 4 priority classes.
  - The selection of the message to be transmitted first can be performed on the basis of frame identifier, IDE bit and RTR bit according to CAN arbitration rules.
- Advanced Message Object Functionality:
  - Message Objects can be combined to build FIFO message buffers of arbitrary size, which is only limited by the total number of message objects.

---

**Controller Area Network (MultiCAN) Controller**

- Message objects can be linked to form a gateway to automatically transfer frames between 2 different CAN buses. A single gateway can link any two CAN nodes. An arbitrary number of gateways may be defined.
- Advanced Data Management:
  - The Message objects are organized in double chained lists.
  - List reorganizations may be performed any time, even during full operation of the CAN nodes.
  - A powerful, command driven list controller manages the organization of the list structure and ensures consistency of the list.
  - Message FIFOs are based on the list structure and can easily be scaled in size during CAN operation.
  - Static Allocation Commands offer compatibility with TwinCAN applications, which are not list based.
- Advanced Interrupt Handling:
  - Up to 16 interrupt output lines are available. Most interrupt requests can be individually routed to one of the 16 interrupt output lines.
  - Message postprocessing notifications can be flexibly aggregated into a dedicated register field of 256 notification bits.



## Controller Area Network (MultiCAN) Controller

### CAN Nodes

Each CAN node consists of several sub-units as described in [Table 22-2](#):

**Table 22-2 Subunits of CAN Nodes**

Subunit	Description
<b>Bit Stream Processor</b>	The Bit Stream Processor performs data, remote, error and overload frame processing according to the ISO 11898 standard. This includes conversion between the serial data stream and the input/output shift registers.
<b>Bit Timing Unit</b>	The Bit Timing Unit defines the length of a bit time and the location of the sample point according to the user settings, taking into account propagation delays and phase shift errors. The Bit Timing Unit also performs resynchronization.
<b>Error Handling Unit</b>	The Error Handling Unit manages the receive and transmit error counter. According to the contents of both counters the CAN node is set into an "Error Active", "Error Passive" or "Bus-Off" state.
<b>Node Control Unit</b>	The Node Control Unit coordinates the operation of the CAN node: <ul style="list-style-type: none"> <li>• Enables/disable CAN transfer of the node</li> <li>• Enable/Disable and generate node specific events that lead to an interrupt request (CAN bus errors, successful frame transfers etc.)</li> <li>• Administration of the Frame Counter</li> </ul>

### Message Controller

The message controller handles the exchange of CAN frames between the CAN nodes and the message objects which are stored in the Message RAM. It performs:

- Receive Acceptance filtering to determine the correct message object for storing of a received CAN frame.
- Transmit Acceptance Filtering to determine the message object to be transmitted first, individually for each CAN node.
- Content transfer between message objects and the CAN nodes, taking into account the status/control bits of the message objects.
- Handling of the FIFO buffering and Gateway functionality.
- Aggregation of message pending notification bits.



### **List Controller**

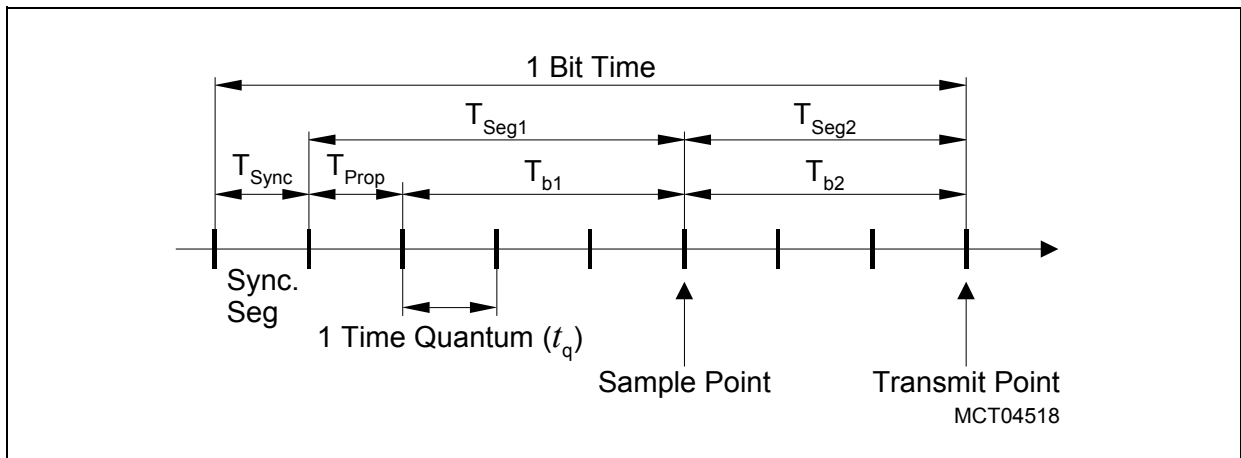
The list controller performs all operations that lead to a modification of the double chained message object lists. Only the list controller is allowed to modify the list structure. The allocation/deallocation or reallocation of a message object can be requested via a user command interface (command panel). The list controller state machine then performs the requested command autonomously.

## 22.2.3 CAN Node Control

Each CAN node may be configured and run independently from the other CAN nodes. To this end each CAN node is equipped with an individual set of SFR registers to control and to monitor the CAN node.

### 22.2.3.1 Bit Timing

According to ISO 11898 standard, a CAN bit time is subdivided into different segments (**Figure 22-3**). Each segment consists of multiples of a time quantum  $t_q$ . The magnitude of  $t_q$  is adjusted by the bit field BRP and by bit DIV8, both controlling the baud rate prescaler (see bit timing register NBTR). The baud rate prescaler is driven by the MultiCAN module clock  $f_{CAN}$ .



**Figure 22-3 CAN Bus Bit Timing Standard**

The Synchronization Segment ( $T_{Sync}$ ) allows a phase synchronization between transmitter and receiver time base. The Synchronization Segment length is always  $1 t_q$ . The Propagation Time Segment ( $T_{Prop}$ ) takes into account the physical propagation delay in the transmitter output driver, on the CAN bus line and in the transceiver circuit. For a working collision detect mechanism,  $T_{Prop}$  has to be two times the sum of all propagation delay quantities rounded up to a multiple of  $t_q$ . The Phase Buffer Segments 1 and 2 ( $T_{b1}$ ,  $T_{b2}$ ) before and after the signal sample point are used to compensate a mismatch between transmitter and receiver clock phase detected in the synchronization segment.

The maximum number of time quanta allowed for resynchronization is defined by bit field SJW in the CAN Node Bit Timing register NBTR. The Propagation Time Segment and the Phase Buffer Segment 1 are combined to parameter TSeg1, which is defined by the value TSEG1 in the respective CAN Node Bit Timing register NBTR. A minimum of 3 time quanta is requested by the ISO standard. Parameter TSeg2, which is defined by the value of TSEG2 in the CAN Node Bit Timing Register NBTR, covers the Phase Buffer Segment 2. A minimum of 2 time quanta is requested by the ISO standard. According

### Controller Area Network (MultiCAN) Controller

ISO standard, a CAN bit time, calculated as the sum of  $T_{\text{Sync}}$ ,  $T_{\text{Seg1}}$  and  $T_{\text{Seg2}}$ , must not fall below 8 time quanta.

Calculation of the bit time:

$$\begin{aligned}
 t_q &= (\text{BRP}+1) / f_{\text{CAN}} && \text{if DIV8} = 0 \\
 &= 8 \times (\text{BRP}+1) / f_{\text{CAN}} && \text{if DIV8} = 1 \\
 T_{\text{Sync}} &= 1 \ t_q \\
 T_{\text{Seg1}} &= (\text{TSEG1} + 1) \times t_q && (\text{min. } 3 \ t_q) \\
 T_{\text{Seg2}} &= (\text{TSEG2} + 1) \times t_q && (\text{min. } 2 \ t_q) \\
 \text{bit time} &= T_{\text{Sync}} + T_{\text{Seg1}} + T_{\text{Seg2}} && (\text{min. } 8 \ t_q)
 \end{aligned}$$

To compensate phase shifts between clocks of different CAN controllers, the CAN controller has to synchronize on any edge from the recessive to the dominant bus level. If the hard synchronization is enabled (at the start of frame), the bit time is restarted at the synchronization segment. Otherwise, the resynchronization jump width  $T_{\text{SJW}}$  defines the maximum number of time quanta a bit time may be shortened or lengthened by one resynchronization. The value of SJW is programmed in the CAN Node Bit Timing Register.

$$\begin{aligned}
 T_{\text{SJW}} &= (\text{SJW} + 1) \times t_q \\
 T_{\text{Seg1}} &\geq T_{\text{SJW}} + T_{\text{prop}} \\
 T_{\text{Seg2}} &\geq T_{\text{SJW}}
 \end{aligned}$$

The maximum relative tolerance for  $f_{\text{CAN}}$  depends on the Phase Buffer Segments and the resynchronization jump width.

$$\begin{aligned}
 \text{dfCAN} &\leq \min(\text{Tb1}, \text{Tb2}) / 2 \times (13 \times \text{bit time} - \text{Tb2}) \quad \text{AND} \\
 \text{dfCAN} &\leq \text{TSJW} / 20 \times \text{bit time}
 \end{aligned}$$

A valid CAN bit timing must be written to the CAN Node Bit Timing Register NBTR before resetting the INIT bit in the Node Control Register, i.e. before enabling the operation of the CAN node.

The Node Bit Timing Register may be written only if bit CCE (Configuration Change Enable) is set in the corresponding Node Control Register.

### **22.2.3.2 CAN Error Handling**

The Error Handling Unit of the CAN node is responsible for the fault confinement of the CAN device. Its two counters, the Receive Error Counter and the Transmit Error Counter (control register NECNT), are incremented and decremented by commands from the Bit Stream Processor. If the Bit Stream Processor itself detects an error while a transmit operation is running, the Transmit Error Counter is incremented by 8. An increment of 1 is used, when the error condition was reported by an external CAN node via an error frame generation. For error analysis, the transfer direction of the disturbed message and the node, recognizing the transfer error, are indicated in the control register NECNT of the respective CAN node. According to the values of the error counters, the CAN node is set into the states "error active", "error passive" and "bus-off".

The CAN node is in error active state, if both error counters are below the error passive limit of 128. It is in error passive state, if at least one of the error counters equals or exceeds 128.

The bus-off state is activated if the Transmit Error Counter equals or exceeds the bus-off limit of 256. This state is reported by flag BOFF in the NSR status register of the CAN node. The device remains in this state, until the bus-off recovery sequence is finished. Additionally, there is the bit EWRN in the NSR status register, which is set, if at least one of the error counters equals or exceeds the error warning limit defined by bit field EWRNLVL in the control registers NECNT of the CAN node. Bit EWRN is reset if both error counters fall below the error warning limit again (see [Page 22-63](#)).

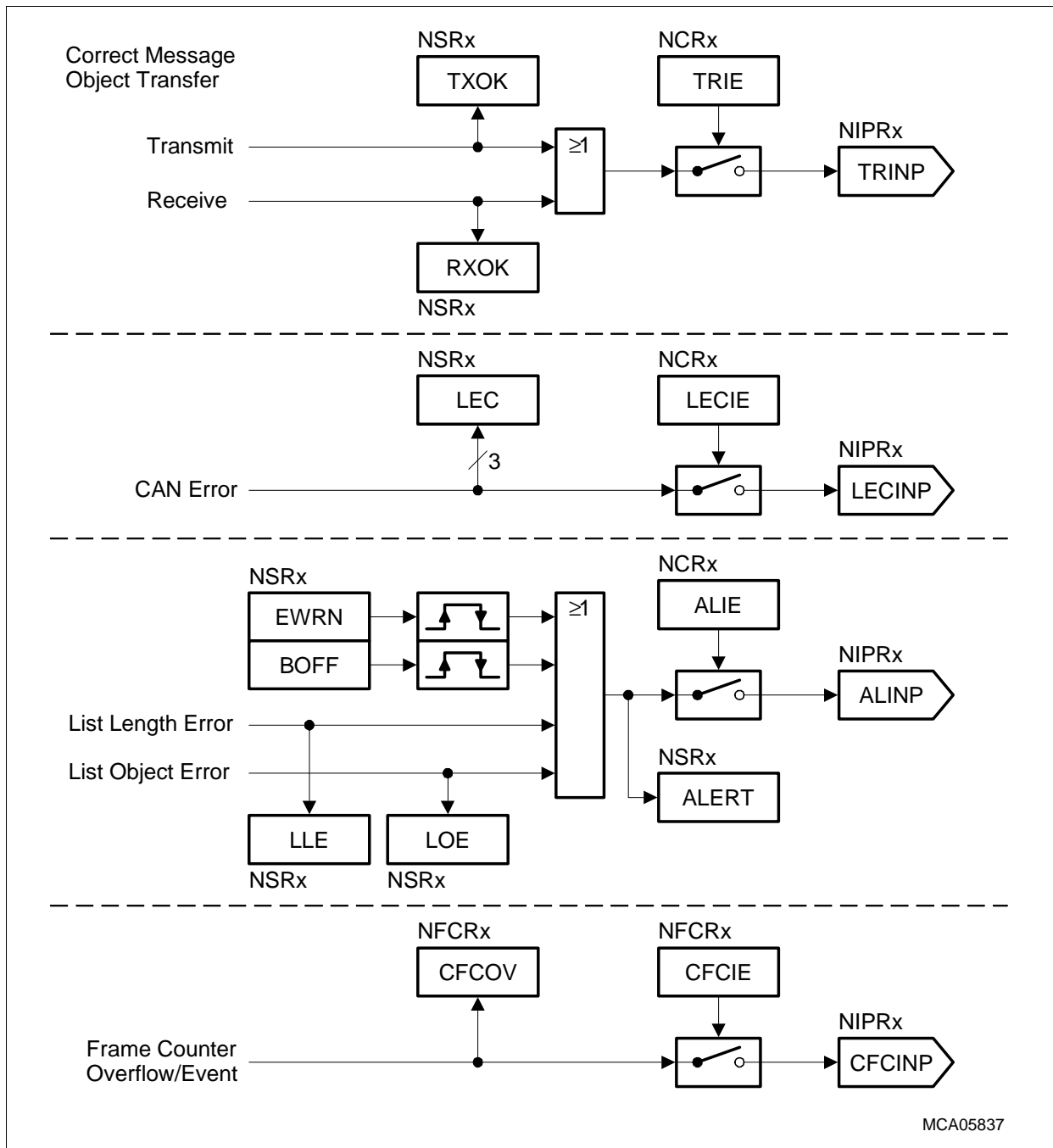
### **22.2.3.3 CAN Frame Counter**

Each CAN node is equipped with a frame counter which allows to count transmitted/received CAN frames or to obtain information about the time instant when a frame has been started to transmit or being received by the CAN node. CAN frame counting/bit time counting is performed by a 16 bit counter which is controlled by register NFCR of the respective CAN node. Bit field CFSEL of register NFCR defines the operation mode of the frame counter:

- **Frame Count Mode:** The frame counter is incremented after the successful transmission and/or reception of a CAN frame. The incremented value is copied to the CFC field of the Interrupt Pointer Register of the message object involved in the transfer.
- **Time Stamp Mode:** The frame counter is incremented with the beginning of a new bit time. When the transmission/reception of a frame starts, the value of the frame counter is captured and stored to the CFC field of register NFCR. After the successful transfer of the frame the captured value is copied to the CFC field of the Interrupt Pointer Register of the message object involved in the transfer.
- **Bit Timing Mode:** Used for baud rate detection and analysis of the bit timing ([Chapter 22.2.5.3](#)).

### 22.2.3.4 CAN Node Interrupts

Each CAN node is equipped with four interrupt sources.



**Figure 22-4 CAN Node Interrupts**

An interrupt request is generated upon:

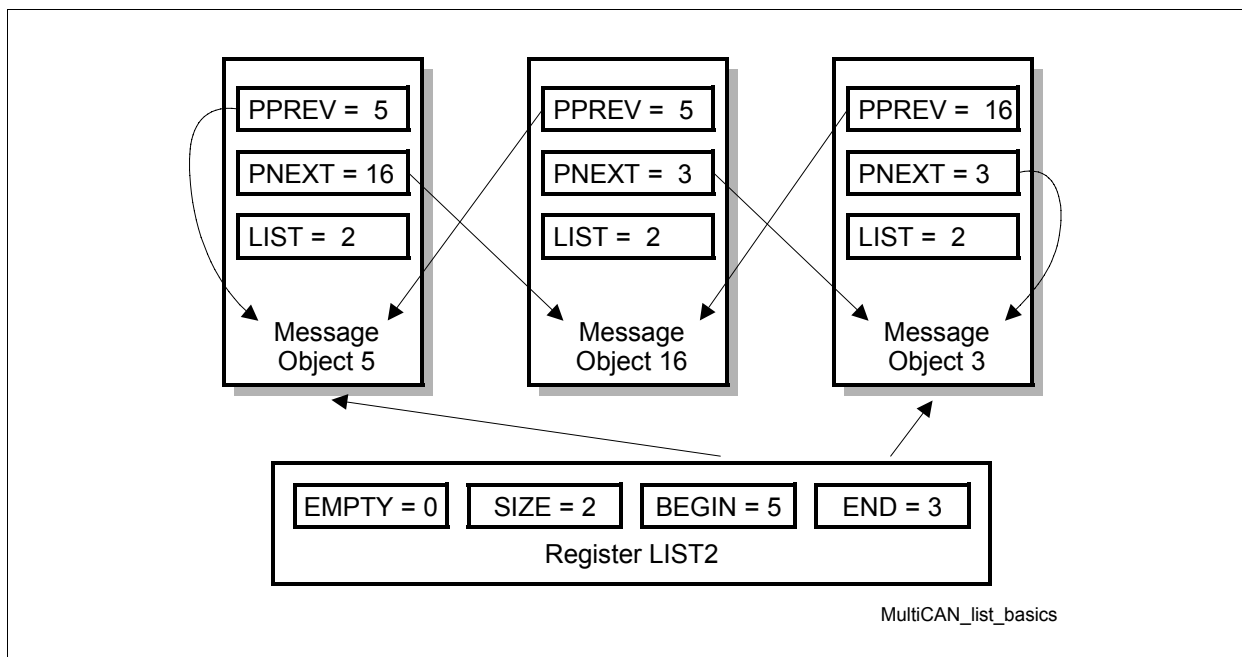
- The successful transmission/reception of a frame,
- An overflow of the frame counter (frame count mode/time stamp mode) or a bit timing measurement event (bit timing mode),
- An error related to the CAN node.

## 22.2.4 Message Object List Structure

The message objects of the MultiCAN module are organized in double chained lists, where each message object has a pointer to the previous message object in the list as well as a pointer to the next message object in the list.

### 22.2.4.1 Basics

The MultiCAN module provides 8 different lists, where each object is allocated to one of these lists. A 4 bit LIST bit field in the Message Object Control Register indicates the list to which the respective message object is currently allocated. In the example of **Figure 22-5** three message objects are allocated to the list with list index 2.



**Figure 22-5 Example Allocation of Message Objects to a List**

The BEGIN field of the List Register points to the first element in the list (object 5 in the example) whereas the END field points to the last element in the list (object 3 in the example). The number of elements in the list is indicated in the SIZE field of the List Register ( $\#elements = SIZE + 1$ , thus  $SIZE = 2$  for the 3 elements of the example). The

## **Controller Area Network (MultiCAN) Controller**

EMPTY bit indicates a list with no elements (EMPTY = 0 in the example, as the list is not empty).

Each message object has a pointer PNEXT (located in the Message Object Control Register) that points to the next message object in the list and a pointer PREV that points to the previous message object in the list. PPREV of the first message object points to the object itself because the first object has no predecessor (in the example object 5 is the first object, indicated by PPREV = 5). PNEXT of the last message object also points to the object itself because the last element has no successor (in the example object 3 is the last object, indicated by PNEXT = 3).

Each message object also has a 4 bit LIST field (located in the Message Object Control Register) which shows list index of the list to which the object is currently allocated (the objects of the example are allocated to list 2, thus LIST = 2).

### **22.2.4.2 List of Unallocated Elements**

The list with list index 0 has a special meaning: It is the list of all unallocated elements. An element is called unallocated if and only if it belongs to list zero. It is called allocated if and only if it belongs to one of the other lists.

After reset all message objects are unallocated, i.e. belong to the list of unallocated elements. The initial allocation of the message objects within the list of unallocated objects is ordered by message number, i.e. the predecessor of message object n is object n-1 and the successor of object n is object n+1.

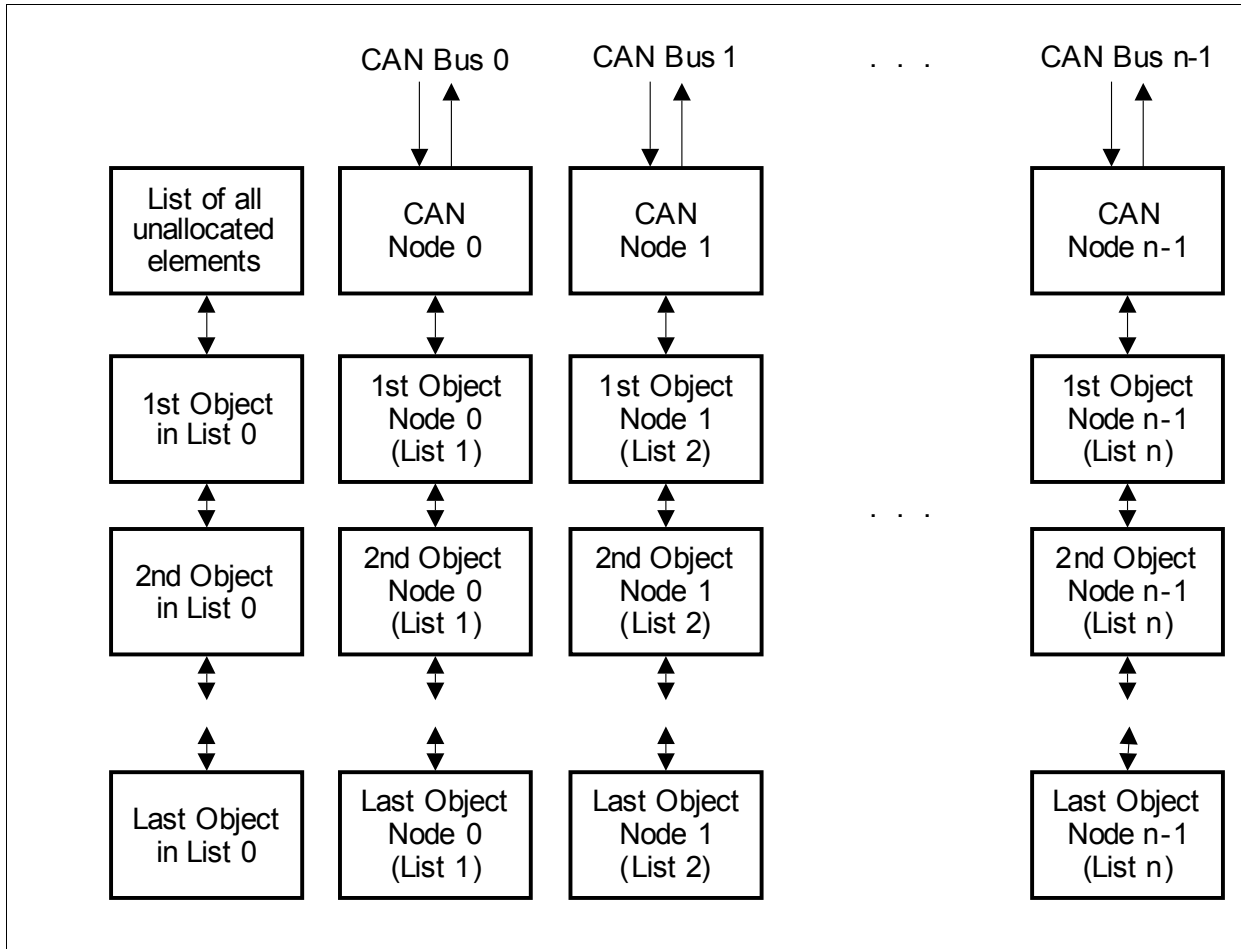
### **22.2.4.3 Connection to the CAN Nodes**

One CAN node is linked to exactly one unique list of message objects..

**Table 22-3 List Indices**

<b>List Index</b>	<b>Description</b>
<b>0</b>	List of unallocated elements
<b>1 to n_nodes</b>	Lists associated to a CAN node. List index i belongs to CAN node i -1.
<b>n_nodes+1 to n_lists-1</b>	Free user lists, which are not associated to a CAN node.

**Controller Area Network (MultiCAN) Controller**



**Figure 22-6 Message Objects Linked to several CAN Nodes**



#### **22.2.4.4 List Command Panel**

The list structure may not be modified directly by means of write accesses to the LIST registers and the PPREV, PNEXT and LIST fields in the message objects as they are read only. The management of the list structure is performed by and limited to the list controller unit inside the MultiCAN module. The list controller is controlled via a command panel which allows the user to issue list allocation commands to the list controller. The list controller basically serves two purposes:

1. Ensure that all operations that modify the list structure result in a consistent list structure.
2. Present maximum comfort and flexibility to the user.

The list controller and the associated command panel allows the programmer to concentrate on the final properties of the list, which are characterized by the allocation of message objects to a CAN node and the ordering relation between objects which are allocated to the same list. The process of list (re-)building is left to the list controller.

A panel command is started by writing the respective command code (see [Table 22-9 “Panel Commands” on Page 22-52](#)) into the PANCMD field of the panel control register. The corresponding command arguments must be written to PANAR1 and PANAR2 before writing the command code or latest together with the command code in a single 32 bit write access to the panel control register (only possible within 32 bit system environments).

With the write of a valid command code the BUSY flag in the Panel Control Register becomes active (BUSY = 1) and the control panel registers are locked, which means that write accesses to the Panel Control Register are ignored. The BUSY flag remains active and the control panel remains locked until the execution of the requested command is completed.

When the issued command is a dynamic allocation which takes an element from the list of unallocated objects, then also the RBUSY bit becomes active together with the BUSY bit (RBUSY = BUSY = 1) to indicate that PANAR1 and PANAR2 are going to be updated by the list controller:

1. The message number of the message object taken from the list of unallocated elements is written to PANAR1.
2. An error status is posted to bit 7 of PANAR2 (Bit 7 = ERR). If ERR = 1 then the list of unallocated elements was empty and the command is aborted. If ERR = 0 then the list was not empty and the command will be performed successfully.

The results are written before the list controller starts the actual allocation process. As soon as the results are available, RBUSY becomes inactive (RBUSY = 0) again, while BUSY still remains active until completion of the command. This allows the user to setup the new message object while it is still in the process of list allocation. The access to message objects is not limited during ongoing list operations. However, any access to a

---

**Controller Area Network (MultiCAN) Controller**

register resource located inside the RAM delays the ongoing allocation process by one access cycle.

As soon as the command is done the BUSY flag becomes inactive (BUSY = 0) and write accesses to the Panel Control Register are enabled again. Also the NOP command code is automatically written to the CMD field of the Panel Control Register. A new command may be started any time during BUSY inactive.

All fields of the Panel Control Register except BUSY and RBUSY may be written by the user. This allows to save and restore the Panel Control Register if the Command Panel shall be used within independent (mutually interruptible) interrupt routines. If this is the case then any task that uses the Command Panel and that may interrupt another task also using the Command Panel should poll the BUSY flag until it becomes inactive and save the whole PANCTR register to a save memory location before issuing a command. At the end it should restore PANCTR from the said memory location.

Before a message object which is allocated to the list of an active CAN node shall be moved to another list or to another position within the same list, bit MSGVAL ("Message Valid") should be cleared in the Message Object Control Register of the message object.

## **22.2.5 CAN Node Analysis Features**

CAN Analyze Mode allows to monitor the CAN traffic without affecting the logical state of the CAN bus.

### **22.2.5.1 Analyze Mode**

CAN Analyze Mode is selected by setting bit CALM in the Node Control Register. CAN Analyze Mode may be selected for each CAN node individually.

In CAN Analyze Mode the transmit pin of the CAN node is held on recessive level. The CAN node may receive frames (data-, remote-, and error frames) but is not allowed to transmit. Active error frames are sent recessive. Received data/remote frames are not acknowledged (i.e. acknowledge slot is sent recessive), but will be received and stored in matching message objects as long as there is any other node that acknowledges the frame.

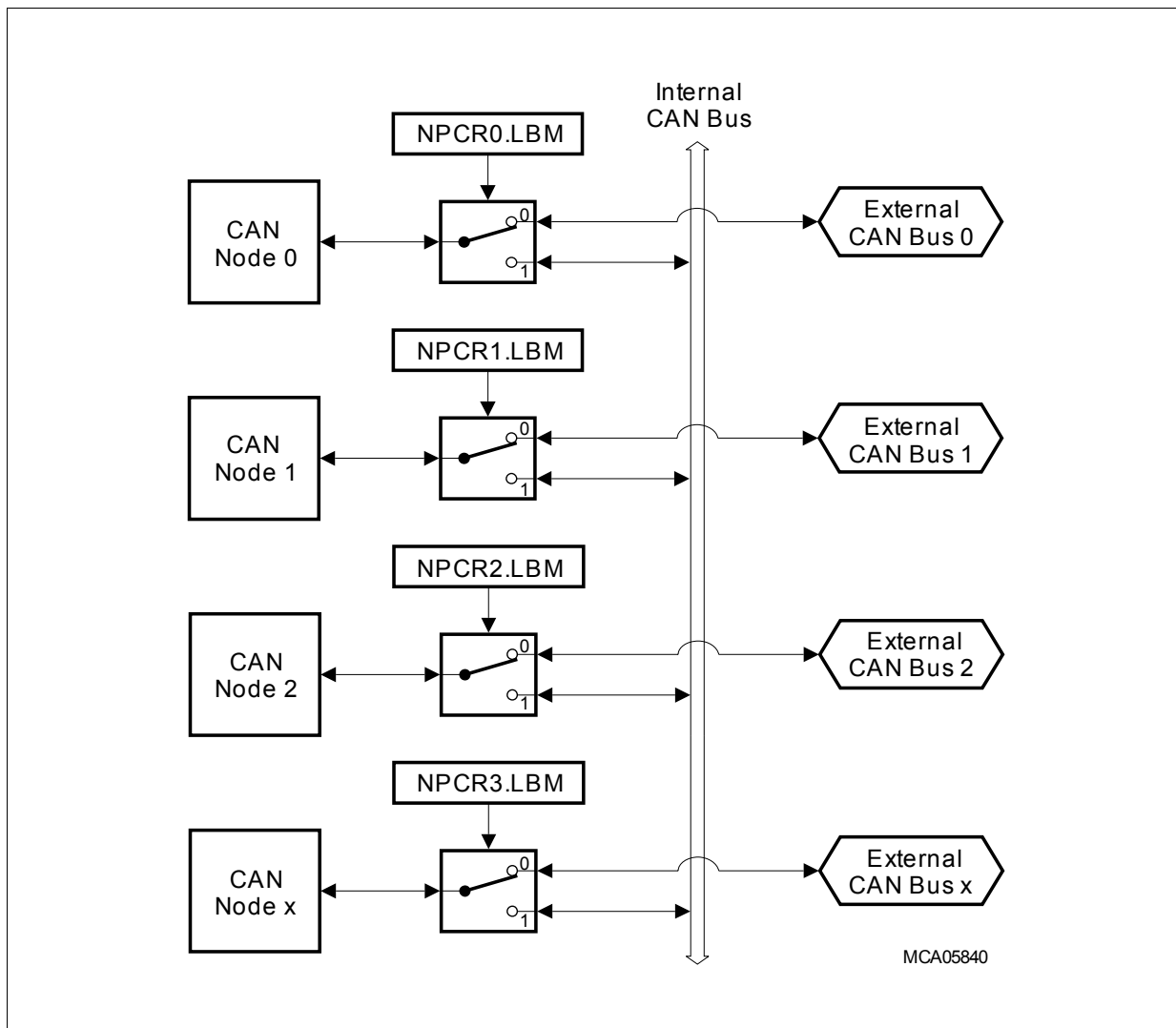
All message object functionality is available, but no transmit request will be executed.

### **22.2.5.2 Loop-back Mode**

The MultiCAN module provides a loop-back mode to enable an in-system test of the MultiCAN module as well as the development of CAN driver software without access to an external CAN bus.

The loop-back feature consists of an internal CAN bus (inside the MultiCAN module) and a bus select switch for each CAN node (**Figure 22-7**). With the switch each CAN node can be wired either to the internal CAN bus (loop-back mode activated) or the external CAN bus, i.e. the transmit- and receive pins (normal operation). The CAN bus which is currently not selected is driven recessive, i.e. the transmit pin is held at 1 and the receive pin is ignored by the CAN nodes which are in loop-back mode.

Loop-back Mode is selected individually for each CAN node by setting bit LBM in the respective Node Port Control Register. All CAN nodes that are in loop-back mode may communicate on the internal CAN bus without affecting the normal operation of the other CAN nodes which are not in loop-back mode.



**Figure 22-7 Loop-back Mode for several CAN Nodes**

### **22.2.5.3 Bit Timing Analysis**

For each CAN node detailed analysis of the bit timing can be performed by means of using dedicated analysis modes of the CAN frame counter. The bit timing analysis functionality of the frame counter may be used for automatic detection of the CAN baud rate as well as for the analysis of the timing of the CAN network.

Bit timing analysis for a CAN node is selected by  $CFMOD = 10_B$  (Bit Timing Mode) in the CAN Node Frame Counter Register.

Bit timing analysis does not affect the operation of the CAN node.

The measurement results are written to the CFC field. Whenever CFC is updated in Bit Timing Mode, then also the CFCOV bit is set in order to indicate the update event. If CFCIE is set then also an interrupt request is generated, where for the CAN node  $i = 0$  to 5 the interrupt request is generated on the interrupt output line  $i$ .

### **Automatic Baud Rate Detection**

Automatic baud rate detection requires to measure the time between the observation of subsequent dominant edges on the CAN bus. This measurement is automatically performed if  $CFSE = 000_B$  in the CAN Node Frame Counter Register. With each dominant edge monitored on the CAN receive input the time (measured in clock cycles) between this edge and the most recent dominant edge is stored in the CFC field.

### **Synchronization Analysis**

The bit time synchronization is monitored if  $CFSEL = 010_B$ . The time between the first dominant edge and the sample point is measured and stored in CFC. The bit timing synchronization offset may be derived from this time as the first edge after the sample point triggers synchronization and there is only one synchronization between consecutive sample points.

Synchronization Analysis may be used to fine tune the baud rate during reception of the first CAN frame with the measured baud rate.

### **Driver Delay Measurement**

The delay between a transmitted edge and the corresponding received edge is measured with  $CFSEL = 011_B$  (dominant to dominant) and  $CFSEL = 001_B$  (recessive to recessive). These delays indicate the time needed to represent a new bit value on the physical implementation of the CAN bus.

## **22.2.6 Message Acceptance Filtering**

The message acceptance filtering includes receive and transmit filtering.

### **22.2.6.1 Receive Acceptance Filtering**

When a message object is received on a CAN node, then a unique message object is determined in which the received frame will be stored upon successful frame reception. A message object qualifies for the reception of a frame if and only if the following conditions are fulfilled:

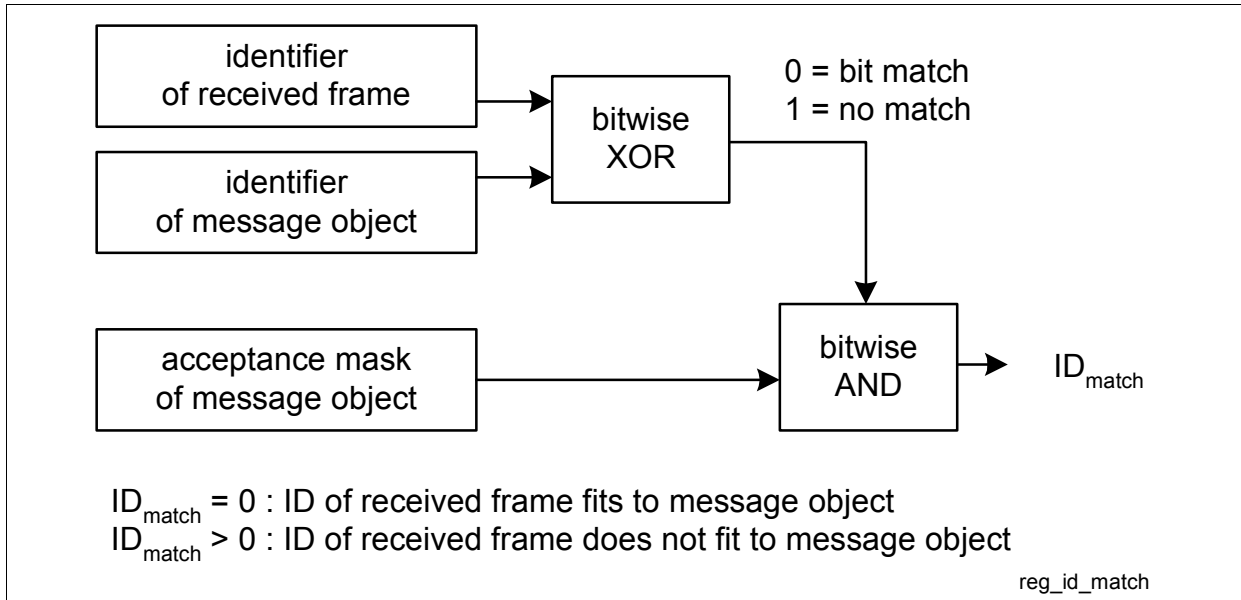
1. The message object is allocated to the list of the CAN node on which the frame is received.
2. MSGVAL is set in the Message Control Register
3. RXEN is set in the Message Control Register
4. The DIR bit in the Message Control Register equals the RTR bit of the received frame. If DIR = 1 (transmit object) then the message object only accepts remote frames. If DIR = 0 (receive object) then the message object only accepts data frames.
5. If MIDE = 1 in the Acceptance Mask Register (MOAMR) then the IDE bit of the received frame equals the IDE bit in the Arbitration Register (MOAR). If MOAR.IDE = 1 then the message object only accepts frames with extended identifier. If MOAR.IDE = 0 then the message object only accepts standard frames. If MOAMR.MIDE = 0 then the IDE bit of the received frame is don't care, i.e. the message object accepts both standard and extended frames.
6. The identifier of the received frame matches the identifier stored in the Arbitration Register of the message object with respect to the acceptance mask in the MOAMR register. This means that each bit of the received identifier is equal to the corresponding identifier bit in the Acceptance Register, except those bits for which the corresponding mask bits in MOAMR are cleared. These identifier bits are don't care. **Figure 22-8** illustrates this identifier check.

A priority ordering relation is defined for the message objects:

A message object A has higher receive priority than a message object B if and only if the following conditions are fulfilled:

1. A belongs to a higher priority class than B, i.e. MOAR.PRI of A must be less than or equal to MOAR.PRI of B.
2. If both objects belong to the same priority class (PRI of A = PRI of B) then message object B is a list successor of A, i.e. B can be reached by means of successively stepping forward in the list, starting from A.

Among all messages that fulfill all 6 qualifying criteria the unique message object with highest receive priority wins acceptance filtering, i.e. is selected for storage of the received frame. All other message objects loose receive acceptance filtering.



**Figure 22-8 Received Message Identifier Acceptance Check**

### 22.2.6.2 Transmit Acceptance Filtering

A message is requested for transmission by means of setting a transmit request in the message object which holds the message. If more than one message object has a valid transmit request for the same CAN node, then a single message object is chosen for actual transmission from the candidates, because only a single message object may be transmitted at the same time on a single CAN bus.

A message object qualifies for transmission on a given CAN node if and only if it meets the following criteria (**Figure 22-9**):

1. The message object is allocated to the list of the CAN node considered.
2. MSGVAL is set in the Message Object Control Register.
3. TXRQ is set in the Message Object Control Register.
4. TXEN0 and TXEN1 are set in the Message Object Control Register.

A priority order relation is defined for all qualifying objects to determine the message to be transmitted first: Let A and B be two message objects qualifying for transmission, where without loss of generality object B is assumed to be a list successor of A, i.e. B can be reached by means of successively stepping forward in the list, starting from A. For both message objects associated CAN messages  $CAN_A$  and  $CAN_B$  are defined, where identifier, IDE and RTR bit are taken from MOAR.ID, MOAR.IDE and MOCTR.DIR.

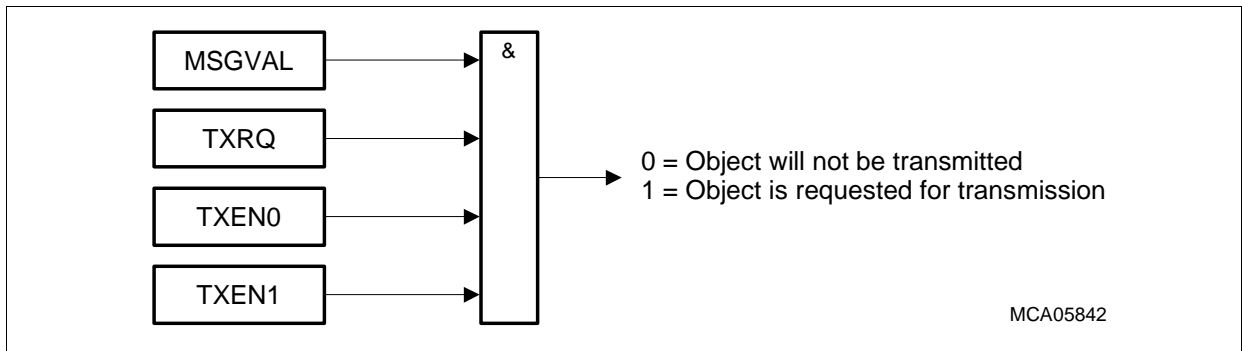
If both message objects belong to a different priority class (different value of bit field PRI in the Message Object Arbitration Register MOAR) then the message object with lower PRI value has higher transmit priority and will be transmitted first.

**Controller Area Network (MultiCAN) Controller**

If both message objects belong to the same priority class (equal value of bit field MOAR.PRI), then message object A has higher transmit priority than object B if and only if one of the following conditions is fulfilled:

1. PRI = 10 and CAN message CAN<sub>A</sub> has higher or equal priority than CAN message CAN<sub>B</sub> with respect to CAN arbitration rules (see [Table 22-13](#)).
2. PRI = 01 or PRI = 11 (priority by list order).
3. PRI = 00 is reserved. Transmit objects with PRI = 00 are not taken into account for the transmit acceptance filtering.

The unique message object that qualifies for transmission and has highest transmit priority wins transmit acceptance filtering, i.e. will be transmitted first. All other message objects lose the current transmit acceptance filtering round. They get a new chance in subsequent filtering rounds.



**Figure 22-9 Effective Transmit Request of Message Object**



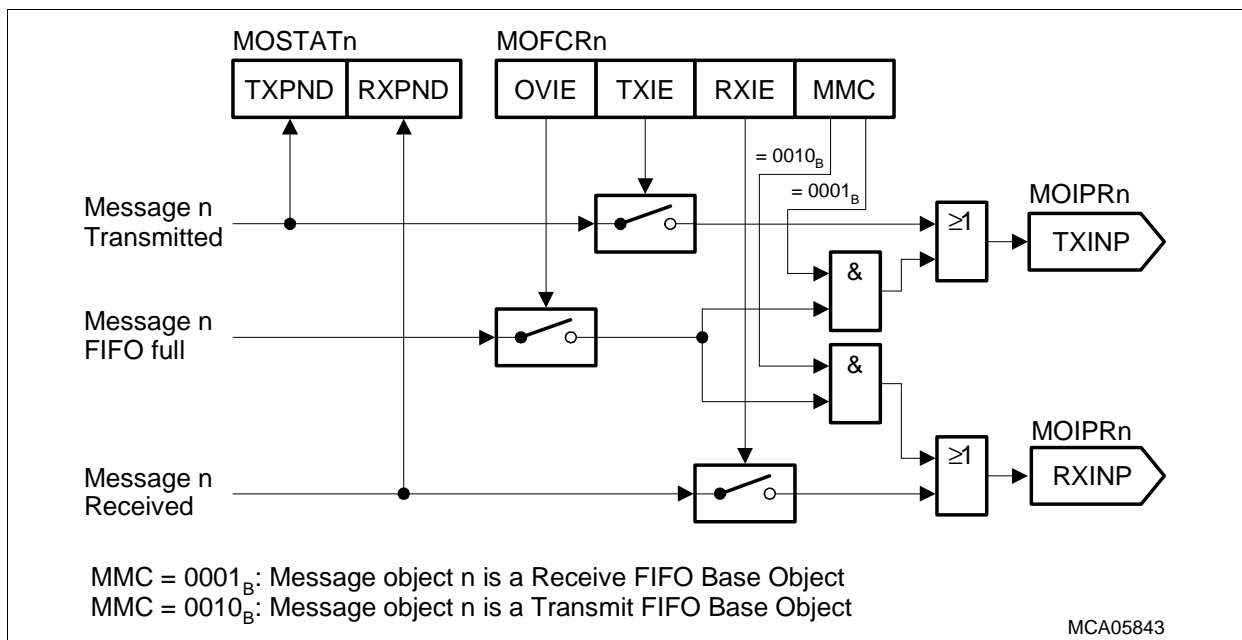
## 22.2.7 Message Postprocessing Interface

When a message object has received or transmitted a frame successfully then the CPU may be notified to perform message postprocessing on the message object. The postprocessing interface of the MultiCAN module consists of two elements:

1. Message Interrupts to trigger postprocessing.
2. Message Pending Registers to aggregate the pending message interrupts into a common structure for postprocessing.

### 22.2.7.1 Message Interrupts

When the storage of a received frame into a message object or the successful transmission of a frame is completed then a message interrupt may be requested. For each message object both transmit and receive interrupts may be routed individually to one of the available interrupt output lines, as illustrated in [Table 22-10](#). A receive interrupt is not restricted to the direct storage of a received frame from the CAN node the message object belongs to. It also occurs upon frame storage induced by FIFO or gateway action. The TXPND and RXPND bits are set whenever a successful transmission/reception takes place, no matter if the respective interrupt is enabled or not.



**Figure 22-10 Message Interrupt Request Routing**

### **22.2.7.2 Message Pending**

When a message interrupt request is generated then also a message pending bit is set in one of the Message Pending Register. To this end the pending bit selection field MPN is defined in the Message Object Interrupt Pointer Register. The value of MPN is combined with TXINP and RXINP to yield the effective bit position of the Pending bit, as illustrated in [Figure 22-11](#). The bit position consists of 2 parts:

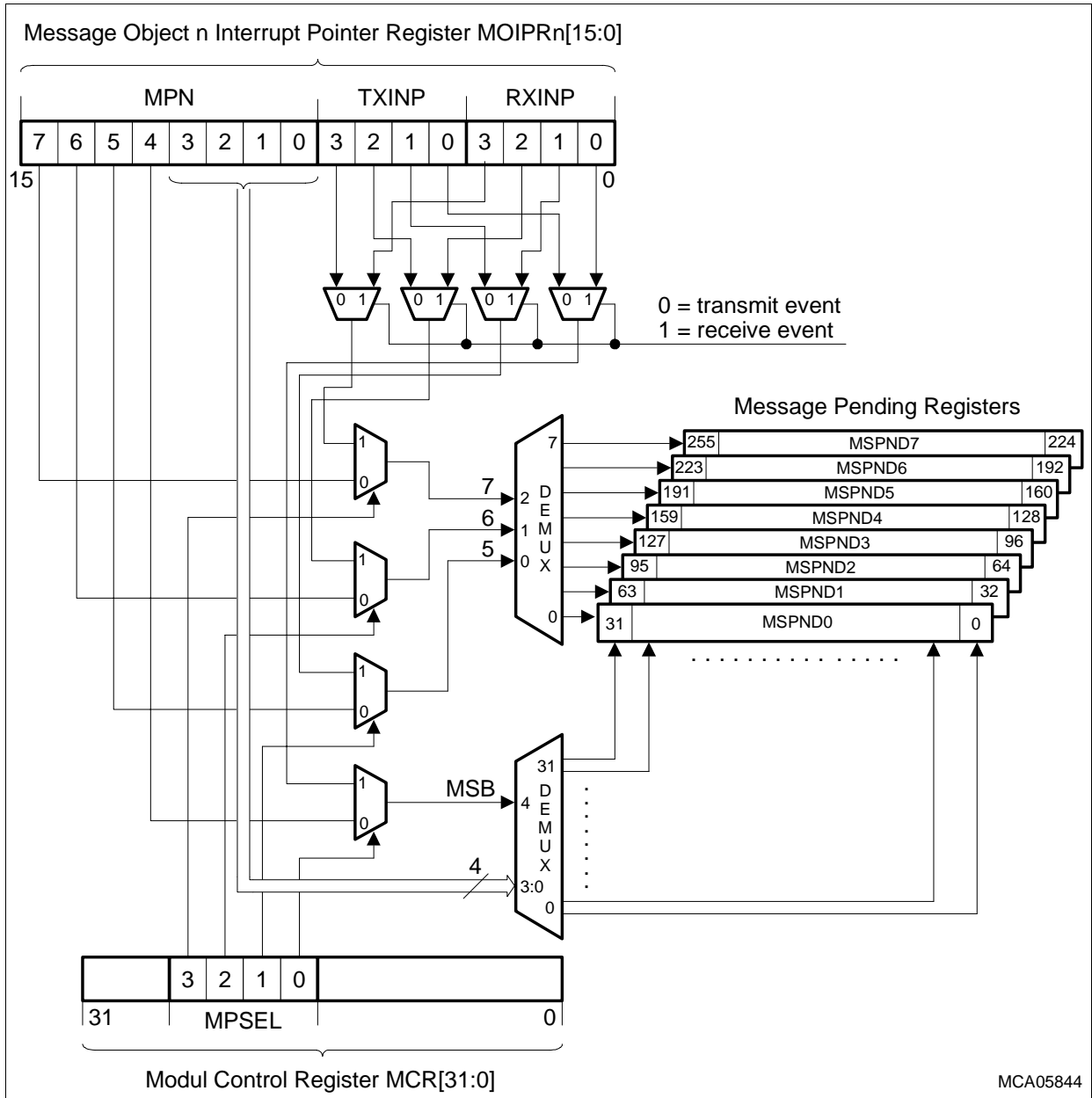
1. The high part (bits [7:5]) of the calculated position selects the Message Pending Register in which the pending bit will be set.
2. The low part (bits [4:0]) of the calculated position selects the position (0-31) of the pending bit within the 32 bit Message Pending Register.

The MPSEL bit field in the MultiCAN Control Register allows to include the interrupt request node pointer (RXINP for reception, TXINP for transmission) so as to implement different target location of the pending bit for receive and transmit.

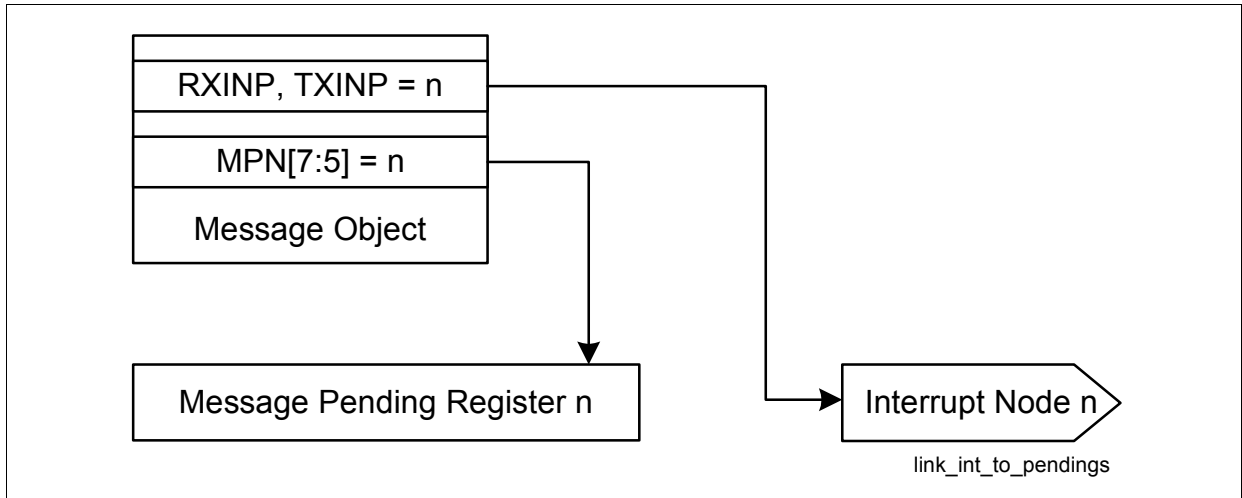
The Message Pending Registers may be written by the CPU, but those bits that are written 1 are left unchanged and only those bits which are written 0 are cleared. This allows to clear individual bits with a single write access instead of a read/modify/write-back access. Thus there is no access conflict when the MultiCAN module sets another pending bit in the same register at the same time.

Each Message Pending Register is linked to an individual Message Index Register which displays the lowest bit position of all set (1) bits in the Message Pending Register. The Message Index Register is read only and is updated immediately when the value of the corresponding Message Pending Register changes.

There is no direct link between the Message Pending Registers and the interrupt request nodes. Such a link may, however, be established by the application. For example, each interrupt request node could be linked to a unique Message Pending Register. The example shown in [Figure 22-12](#) links message Pending Register n to interrupt output line n (n = 0-7).



**Figure 22-11 Target Location of the Message Pending Bit (Transmit/Receive)**



**Figure 22-12 Example Link of Message Pending Registers to Interrupt Output Lines**

## **22.2.8 Message Object Data Handling**

The following section describes the actions taken during a frame reception and during a frame transmission.

### **22.2.8.1 Frame Reception**

When a message is received on the CAN bus then the storage of the message into a message object is prepared and performed according to the scheme shown in **Figure 22-13**. The MultiCAN module not just copies the received data into the message object, but it provides advanced features to enable consistent data exchange between MultiCAN and CPU.

#### **MSGVAL**

The MSGVAL ("Message Valid") bit in the Message Object Control Register is the main switch of the message object. The MultiCAN module only stores information in the message object during the frame reception process when MSGVAL is set (MSGVAL = 1).

Whenever MSGVAL is reset (MSGVAL = 0) by the CPU then the MultiCAN module stops all ongoing write accesses to the message object so that the message object may be reconfigured by the CPU in subsequent write accesses to the message object without being disturbed by the MultiCAN.

#### **RTSEL**

When the CPU re-configures a message object (i.e. clears MSGVAL, modifies the message object and sets MSGVAL again) during CAN operation then the following scenario can occur:

1. The message object wins receive acceptance filtering.
2. The CPU clears MSGVAL to reconfigure the message object.
3. The CPU sets MSGVAL again after reconfiguration.
4. The end of the received frame is reached. As MSGVAL is set, the received data are stored in the message object, a message interrupt request is generated, gateway and fifo actions are processed etc.

The storage of the received data may be undesirable if the context of the message object has changed, because the old message object configuration has been used for acceptance filtering of the message.

Bit MOCTR.RTSEL ("Receive/Transmit Selected") allows to disconnect a message object from an ongoing frame reception:

When a message object wins receive acceptance filtering then bit RTSEL is set (RTSEL = 1) by the MultiCAN in order to indicate an upcoming frame delivery. The MultiCAN checks RTSEL for value 1 upon successful frame reception in order to verify

**Controller Area Network (MultiCAN) Controller**

that the object is still ready for receiving the frame. The received frame is stored in the message object (along with all subsequent actions such as message interrupts, FIFO & gateway actions, flag updates) only if RTSEL = 1.

When the user invalidates a message object during CAN operation (MSGVAL → 0) then the user should clear RTSEL before setting MSGVAL again (latest with the same write access that sets MSGVAL) in order to prevent the storage of a frame that belongs to the old context of the message object. Thus message object reconfiguration should consist of the following sequence of steps:

1. Clear MSGVAL.
2. Reconfigure message object while MSGVAL = 0.
3. Clear RTSEL and set MSGVAL.

**RXEN**

Bit MOCTR.RXEN enables a message object for frame reception. A message object can receive CAN messages from the CAN bus only if RXEN = 1. The MultiCAN evaluates RXEN only during receive acceptance filtering. After receive acceptance filtering RXEN is ignored, i.e. the value of RXEN has no influence on the actual storage of a received message in a message object.

Bit RXEN enables a “soft phase out” of a message object: When the user clears RXEN then a currently received CAN message for which the message object has won acceptance filtering is still stored in the message object, but for subsequent messages the message object no longer wins receive acceptance filtering.

**RXUPD, NEWDAT and MSGLST**

An ongoing frame storage process is indicated with the RXUPD (“Receive Updating”) bit in the Message Object Control Register. The MultiCAN module sets RXUPD with the start and clears RXUPD with the end of a message object update (which consists of frame storage as well as flag updates).

After storing the received frame (identifier, IDE bit, DLC and for data frames also the data field) in the message object NEWDAT (“New Data”) is set by the MultiCAN. If NEWDAT was already set then also MSGLST (“Message Lost”) is set in order to indicate data loss.

The RXUPD and NEWDAT flags may be used by the CPU to read consistent frame data from the message object during ongoing CAN operation. The following steps are recommended:

1. Clear NEWDAT
2. Read message content (identifier, data etc.) from message object
3. Read Message Object Control Register and check that both NEWDAT and RXUPD are cleared. If this is not the case then goto back to step 1.
4. As step 3 was successful, the read message content is consistent, i.e. has not been updated by the MultiCAN while reading.

## Controller Area Network (MultiCAN) Controller

The bits RXUPD, NEWDAT and MSGLST work in the same fashion for the reception of data as well as remote frames.

### 22.2.8.2 Frame Transmission

The process of message object transmission is illustrated in [Figure 22-14](#). In addition to copying the message content (identifier, IDE bit, RTR = DIR bit, DLC and for data frames also the data field) to the internal transmit buffer of the CAN node that the message object belongs to, also several status flags are served and monitored in order to enable consistent data handling.

The transmission process (after transmit acceptance filtering) of a given message object makes no difference between remote and data frames.

#### MSGVAL, TXRQ, TXEN0, TXEN1

For the MSGVAL bit the section [“MSGVAL” on Page 22-28](#) for frame reception is also valid for transmission.

A message may only be transmitted if all four bits MSGVAL (“Message Valid”), TXRQ (“Transmit Request”), TXEN0 (“Transmit Enable 0”), TXEN1 (“Transmit Enable 1”) of the Message Object Control Register are set (1) (see also [Figure 22-9](#)). Although these bits are equivalent with respect to the transmission process, they have different semantics:

**Table 22-4 Bits to set (1) in MOCTR for message transmission**

Bit	Description
<b>MSGVAL</b>	<b>Message Valid</b> Main Switch of the Message Object
<b>TXRQ</b>	<b>Transmit Request</b> Standard Transmit Request bit. The CPU should set this bit whenever a message object shall be transmitted. TXRQ is cleared automatically at the end of the successful transmission, except when there are new data (indicated by NEWDAT = 1) to be transmitted. When the single transmit trial bit is set (STT = 1) in the Message Object Function Register then TXRQ is already cleared by the MultiCAN when the content of the message object is copied to the transmit frame buffer of the CAN node. A received remote request (i.e. remote frame received on CAN bus) sets bit TXRQ to request the transmission of the corresponding data frame.

**Controller Area Network (MultiCAN) Controller**

**Table 22-4 Bits to set (1) in MOCTR for message transmission (cont'd)**

<b>Bit</b>	<b>Description</b>
<b>TXEN0</b>	<p><b>Transmit Enable 0</b></p> <p>This bit may be temporarily cleared by the CPU to suppress the transmission of this object when it writes new content to the data field. This avoids transmission of inconsistent frames which consist of a mixture of old and new data.</p> <p>Remote requests are still accepted during TXEN0 = 0, but transmission of the data frame is suspended until the CPU re-enables transmission (TXEN0 = 1).</p>
<b>TXEN1</b>	<p><b>Transmit Enable 1</b></p> <p>This bit is used in transmit FIFOs to select the message object which is transmit active within the FIFO structure.</p> <p>For message objects which are not transmit FIFO elements TXEN1 may either be set to 1 permanently or be used as a second, independent transmission enable bit.</p>

## **RTSEL**

When a message object has been identified to be transmitted next (by acceptance filtering) then the MultiCAN set bit MOCTR.RTSEL ("Receive/Transmit Selected").

When the MultiCAN copies the message object to the transmit buffer it checks bit RTSEL and the message is transmitted only if RTSEL = 1.

After the successful transmission of the message bit RTSEL is checked again and message postprocessing is only performed if RTSEL = 1.

A complete reconfiguration of an operating message object should be done by means of the following steps:

1. Clear MSGVAL ("Message Valid").
2. Reconfigure message object while MSGVAL = 0.
3. Clear RTSEL and set MSGVAL.

Here clearing RTSEL ensures that the message object is disconnected from an ongoing/scheduled transmission and no message object processing (copying message to transmit buffer incl. clearing NEWDAT, clearing TXRQ, time stamp update, message interrupt etc.) within the old context of the object may occur after the message object becomes valid again, but within a new context.

## **NEWDAT**

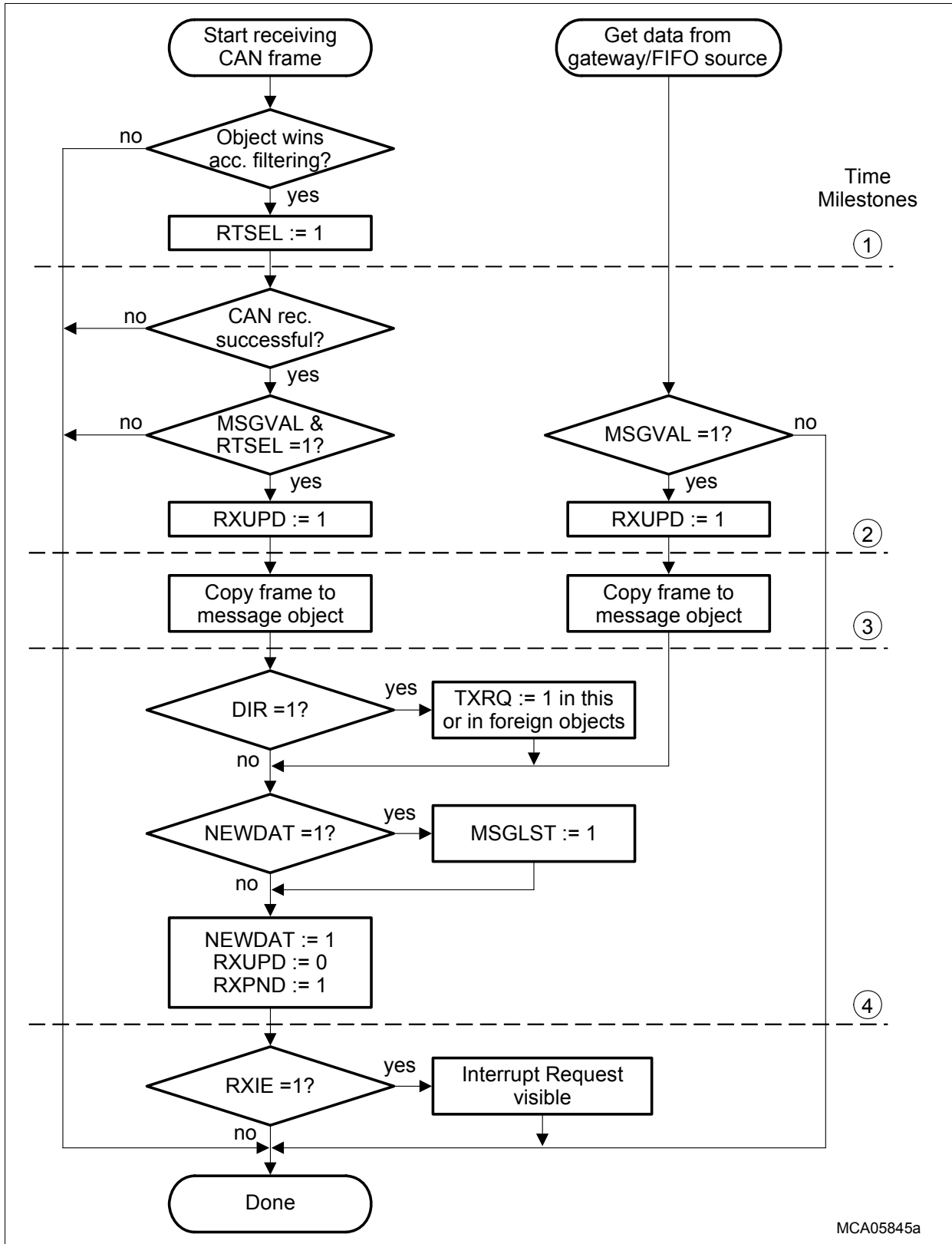
When the content of a message object has been transferred to the transmit buffer of the CAN node then bit NEWDAT ("New Data") is cleared to indicate that the transmit data of the message object are no longer new.



**Controller Area Network (MultiCAN) Controller**

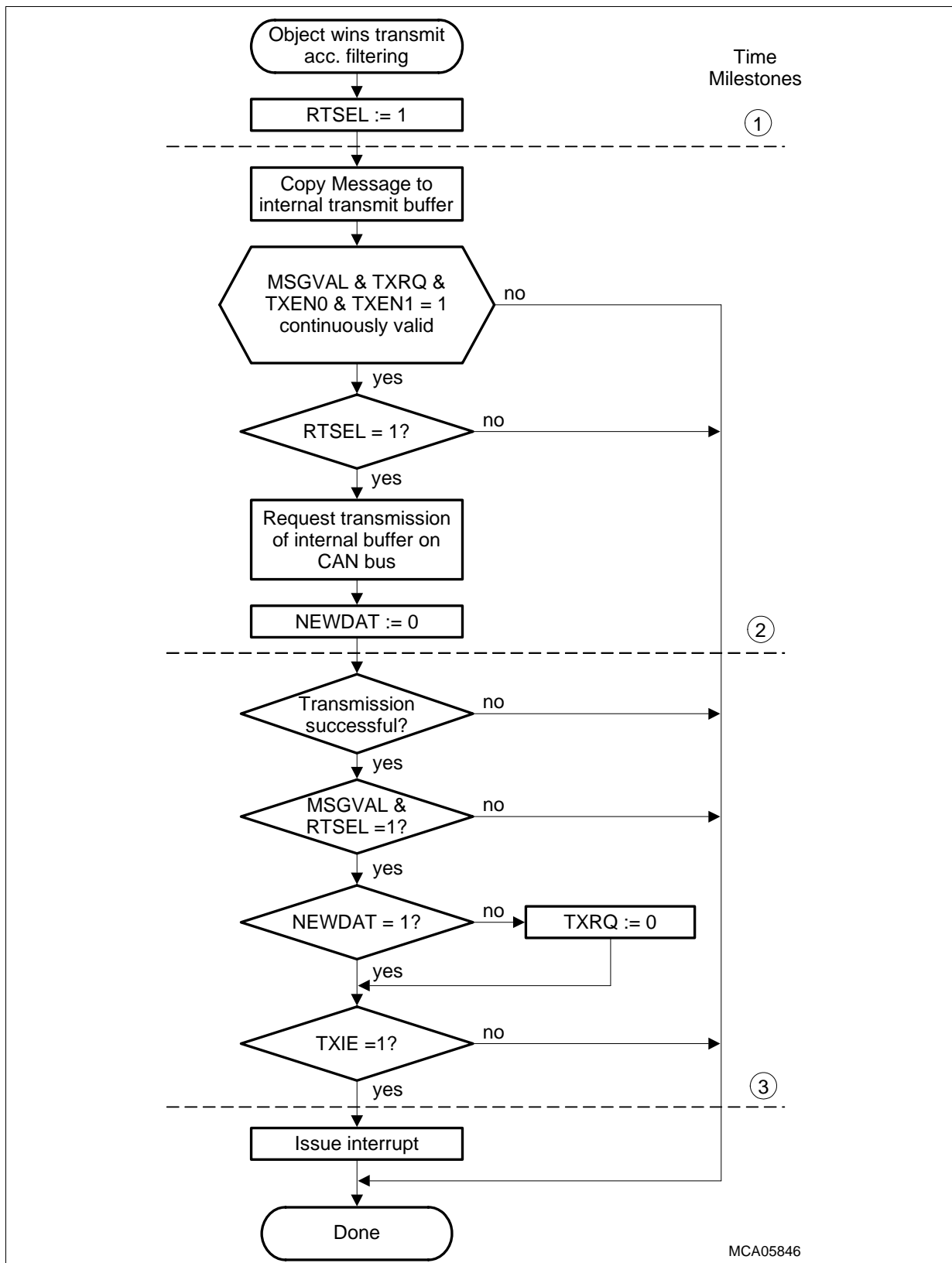
When the CAN transmission of the frame is successful and NEWDAT is still cleared (i.e. no new data have been copied to the message object in the meantime) then TXRQ ("Transmit Request") is cleared automatically.

If, however, the NEWDAT bit has been set again by the CPU (because a new frame shall be transmitted) then TXRQ is not cleared in order to enable the transmission of the new data.



MCA05845a

**Figure 22-13 Data Delivery to Message Object by MultiCAN Module**



**Figure 22-14 Transmission of a Message Object**

## **22.2.9 Message Object Functionality**

This section describes the functionality that is related to each individual Message Object.

### **22.2.9.1 Standard Message Object Mode**

Standard message mode is selected via MMC = 0000<sub>B</sub> in the Message Object Function Control Register of the message object. In this mode a message object may transmit and receive CAN frames according to the basic rules as described in the previous sections. Additional services such as Single Data Transfer Mode or Single Transmit Trial (see sections below) are available and may be selected individually by the user.

### **22.2.9.2 Single Data Transfer Mode**

Single Data Transfer Mode is a useful feature in order to broadcast data over the CAN bus without unintended doubling of information. Single Data Transfer Mode is selected via bit SDT in the Message Object Function Register of the message object.

#### **Message Reception**

When a received message is stored in a message object and further messages are stored in the same message object before the CPU reads the first message object, then the content of the first message gets lost and is replaced with the content of the subsequent messages (indicated by MSGLST = 1).

If SDT = 1 (Single Data Transfer Mode activated) then the MultiCAN controller automatically clears the MSGVAL bit of the message object after the storage of a received data frame to prevent the reception of further messages.

The reception of a remote frame does not lead to the clearance of MSGVAL.

#### **Message Transmission**

When a message object receives a series of multiple remote requests then it transmit several data frames in response to the requests. If the data within the message object has not been updated in the time between the transmissions, the same data may be represented more than once on the CAN bus.

In Single Data Transfer Mode (SDT = 1) this is avoided because the MultiCAN controller automatically clears MSGVAL after the successful transmission of a data frame.

The transmission of a remote frame does not clear MSGVAL.

### **22.2.9.3 Single Transmit Trial**

If the bit STT in the message object function register is set (STT = 1) then the transmission request is cleared (TXRQ := 0) when the frame content of the message

**Controller Area Network (MultiCAN) Controller**

object has been copied to the internal transmit buffer of the CAN node. Thus the transmission of the message object is not tried again when it fails due to CAN bus errors.

**22.2.9.4 Message Object FIFO Structure**

In case of high CPU load it may be difficult to process a series of CAN frames in time. This may happen for the short term reception of multiple messages as well as the transmission of a series with tight due date.

Therefore a FIFO buffer structure has been implemented in order to avoid loss of incoming messages and to minimize the setup time for outgoing messages. The FIFO structure may also be used to automate the reception or transmission of a series of CAN messages and to generate a single message interrupt when the whole series is done.

There may be as many FIFOs in parallel as are required by the application. The number of FIFOs and their size are only limited by the number of message objects available. A FIFO may be installed, resized and deinstalled any time, even during CAN operation.

The basic structure of a FIFO is shown in [Figure 22-15](#). A FIFO consists of a single base object (shown on the left side) and several slave objects (shown on the right side). The slave objects are chained together in the same list structure. The base object may be allocated to any list. Although [Figure 22-15](#) shows the base object as a separate item apart from the slave objects, it is also possible to integrate the base object at any place into the chain of slave objects, so that the base object is slave object, too (not possible for gateways). The FIFO structure fully relies on the list structure. The absolute object numbers of the message objects have no impact on the operation of the FIFO.

The base object needs not be allocated to the same list as the slave objects. Only the slave object must be allocated to a common list (as they are chained together). The BOT, CUR and TOP pointer link the base object to the slave objects, no matter whether the base object is allocated to the same or to another list than the slave objects.

The absolute minimum FIFO would consist of a single message object which is both FIFO base and FIFO slave (not very useful). The biggest possible FIFO would use all message objects of the MultiCAN module. Any sizes between these extremes are possible.

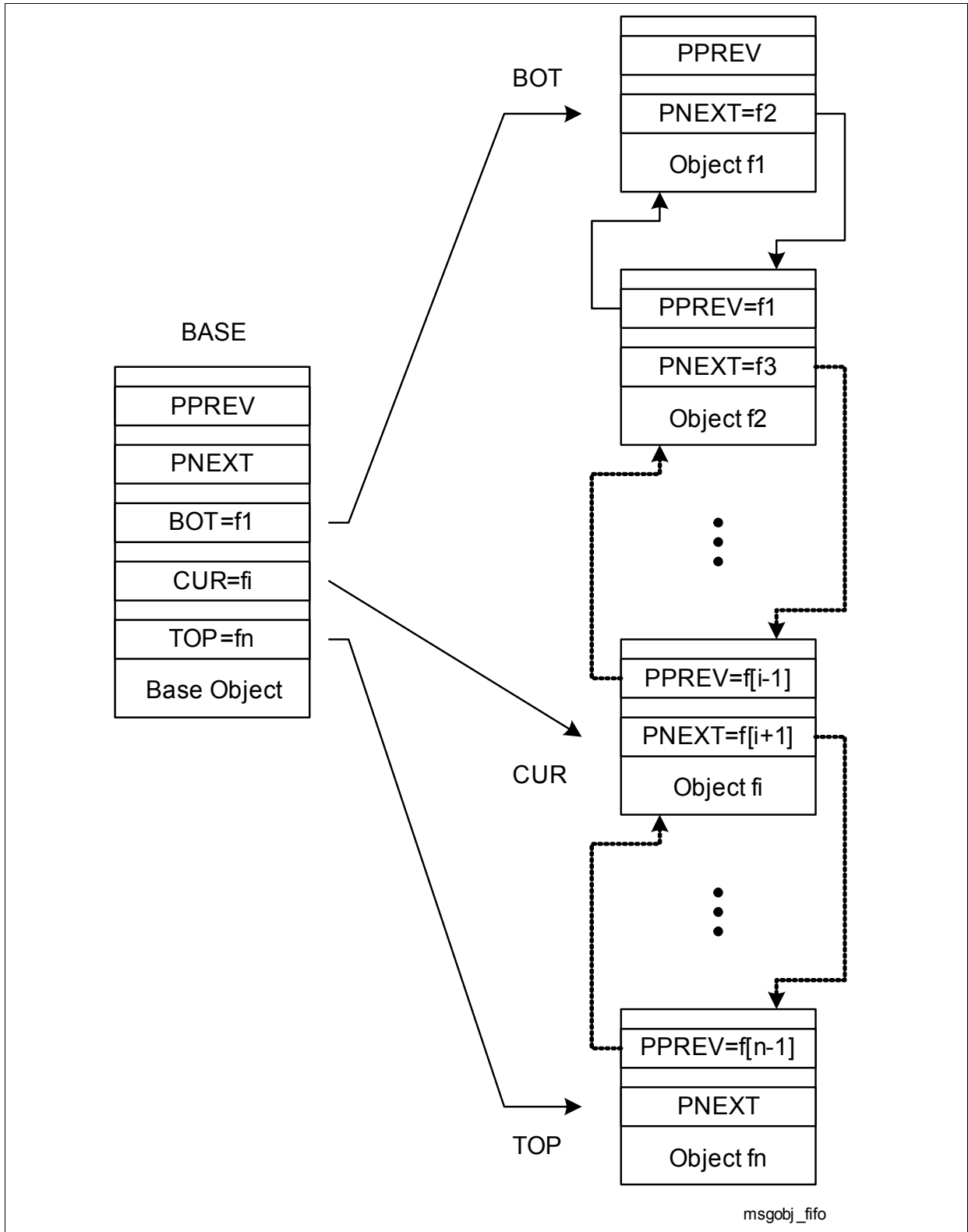
In the FIFO base object the boundaries of the FIFO are defined. The BOT field in the FIFO/Gateway Pointer Register of the base object points to the first (bottom) slave element in the FIFO. The TOP field in the FIFO/Gateway Pointer Register of the base object points to the last (top) slave element.

The CUR field in the FIFO/Gateway Pointer Register of the FIFO base object points to the actual slave object selected by the MultiCAN for message transfer. When a message transfer takes place with this object then CUR is moved to the next position. If CUR has already reached the top of the FIFO (CUR = TOP) then it is wrapped around to the bottom of the FIFO (CUR := BOT). Otherwise CUR is moved to the next message object in the list structure of the slave objects (CUR := PNEXT of current object). This scheme

**Controller Area Network (MultiCAN) Controller**

yields a circular FIFO structure where the fields BOT and TOP just establish the link from the last to the first element, which is missing in the linear structure.

The SEL field in the FIFO/Gateway Pointer register of the base object may be used for monitoring purposes. It allows to select any slave object and to generate a message interrupt if the CUR pointer reaches the value of SEL. Thus SEL offers a convenient way to determine the end of a predefined series of message transfers, or it may be used to issue a warning to the CPU when the FIFO gets full.



**Figure 22-15 FIFO Structure with FIFO Base and n FIFO Destinations (Slaves)**

### **22.2.9.5 Receive FIFO**

The Receive FIFO structure is used to buffer incoming (received) remote or data frames. A Receive FIFO is selected via  $MMC = 0001_B$  in the Message Object Function Control Register of the FIFO base object. This MMC code automatically designates the message object as FIFO base object. The message mode of the FIFO slave objects are not relevant for the operation of the Receive FIFO.

When the FIFO base object receives a frame from the CAN node it belongs to, then the frame is not stored in the base object. Instead the message is stored in the message object that is selected by the CUR pointer in the FIFO/Gateway Pointer Register of the FIFO base object.

The message object selected by CUR receives the CAN message as if it were the direct receiver of the message. However,  $MMC = 0000_B$  is implicitly assumed for the FIFO slave, i.e. a standard message delivery is performed. The actual message mode (MMC) of the FIFO slave is ignored. There is also no extra acceptance filtering to match the received frame against the identifier, IDE bit and DIR bit of the slave object.

When the FIFO base object receives a CAN frame then the MultiCAN moves the current pointer CUR to the next message object in the FIFO structure, which will then be used to store the next incoming message. The old value of CUR is used for the current transfer.

IF bit OVIE is set in the Message Object Function Register of the FIFO base object and the pointer CUR reaches the value stored in SEL then a FIFO overflow interrupt request is generated. The interrupt request is generated on interrupt output line TXINP (TXINP of the base object) immediately after the storage of the received frame into the slave object. Transmit interrupts are still generated if TXIE is set.

A CAN message is stored in a FIFO slave only if MSGVAL = 1 in both FIFO base and slave object.

In order to avoid direct reception of a message by a slave message object, as if it was an independent message object and not a part of a FIFO, the bit RXEN of each slave object must be cleared. The setting of the bit RXEN is “don’t care” only if the slave object is located in a list not assigned to a CAN node.



### **22.2.9.6 Transmit FIFO**

The Transmit FIFO structure is used to buffer a series of data or remote frames to be transmitted. A transmit FIFO consists of one base message object and one or more slave message objects.

A Transmit FIFO base object is selected via  $MMC = 0010_B$  in the Message Object Function Control Register of the FIFO base object. Unlike the Receive FIFO the Transmit FIFO requires the explicit declaration of the FIFO slave objects via  $MMC = 0011_B$ . The CUR pointer of all slave objects must point back to the Transmit FIFO Base Object (to be initialized by user).

The TXEN1 bits of all message objects except the one which is selected by the CUR pointer of the base object must be cleared (to be initialized by user). TXEN1 of the message object selected by CUR must be set. CUR may be initialized to any FIFO slave object.

When tagging the message objects of the FIFO valid to start the operation of the FIFO then the base object must be tagged valid ( $MSGVAL := 1$ ) first.

When a Transmit FIFO shall be deinstalled during operation, then the slave objects must be tagged invalid ( $MSGVAL := 0$ ) first.

The Transmit FIFO uses the TXEN1 bit in the Message Object Control Register of all FIFO elements to select the actual message for transmission. Transmit acceptance filtering evaluates TXEN1 for each message object and a message object may win transit acceptance filtering only if TXEN1 is set. When a FIFO element has transmitted a message then in addition to standard transmit postprocessing (clear TXRQ, transmit interrupt etc.) the MultiCAN clears TXEN1 in that message object and moves the CUR pointer in the corresponding FIFO base object to the next message object to be transmitted. TXEN1 is set automatically in the next message object. Thus TXEN1 moves along the FIFO structure like a token to select the active element.

IF bit OVIE is set in the Message Object Function Register of the FIFO base object and the pointer CUR reaches the value stored in SEL then a FIFO overflow interrupt request is generated. The interrupt request is generated on interrupt output line as defined by RXINP (RXINP of the base object) when postprocessing of the received frame is done. Receive interrupts are still generated for the Transmit FIFO base object if bit RXIE is set.

### **22.2.9.7 Gateway Mode**

The Gateway Mode allows to establish an automatic information transfer between two independent CAN bus systems without CPU interaction.

The Gateway Mode operates on message object level. In Gateway mode, information is transferred between two message objects, resulting in an information transfer between the two CAN nodes to which the message objects are allocated. A gateway may be established between any pair of CAN nodes and there may be as many gateways as there are message objects available to build the gateway structure.

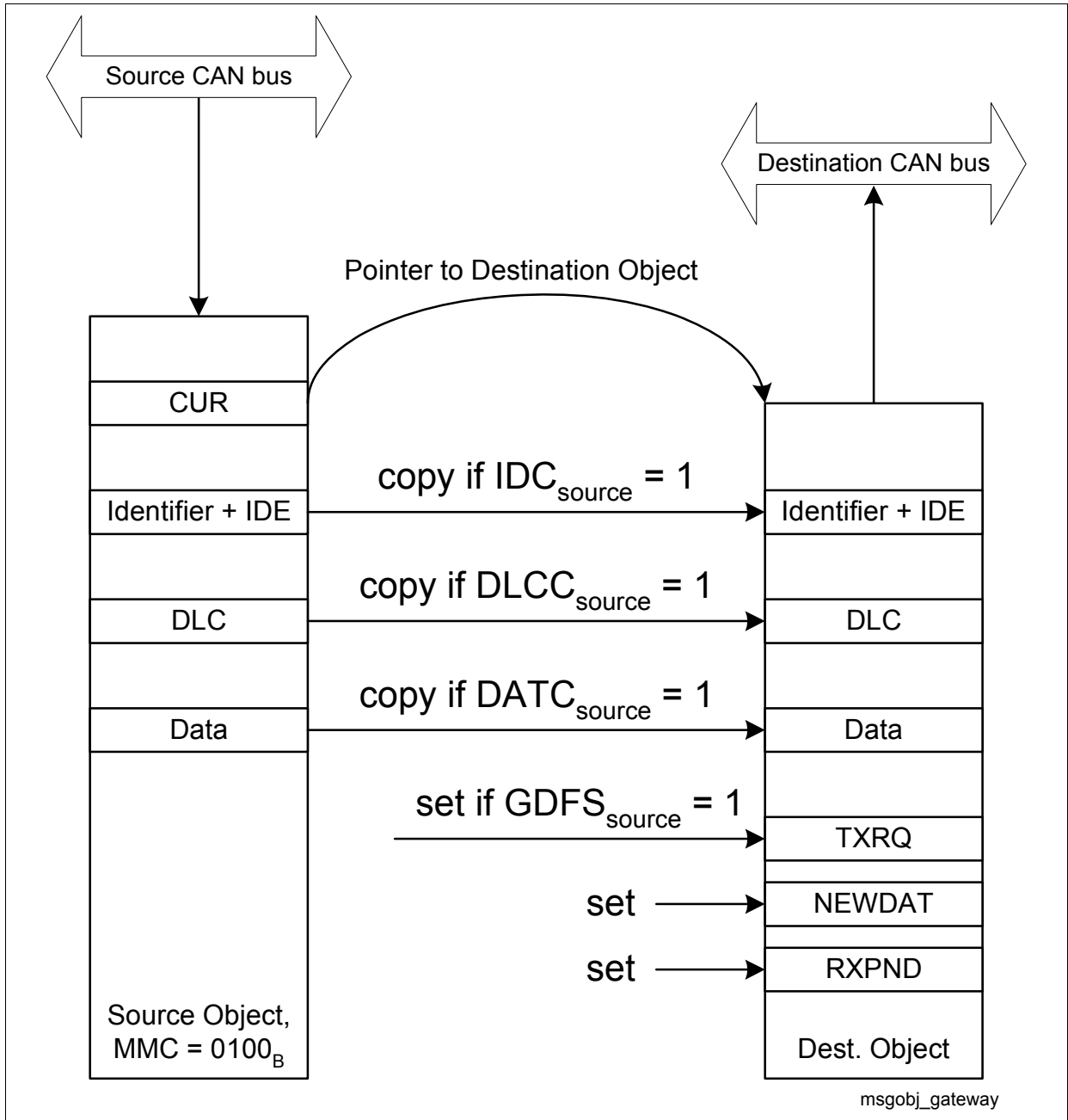
Gateway Mode is selected via  $MMC = 0100_B$  in the Message Object Function Control Register of the gateway source object. The gateway destination object is selected by the CUR pointer in the FIFO/Gateway Pointer Register of the source object. The gateway destination object just needs to be valid ( $MSGVAL = 1$ ), all other settings are not relevant for the information transfer from the source object to the destination object.

A gateway source object behaves like a standard message objects, but when a CAN frame has been received and stored in the source object, some additional actions are performed by the MultiCAN (**Figure 22-16**):

1. If bit DLCC is set in the Message Object Function Register of the source object, then the DLC code is copied from the source object to the destination object.
2. If bit IDC is set in the Message Object Function Register of the source object, then the identifier and the IDE bit are copied from the source object to the destination object.
3. If bit DATC is set in the Message Object Function Register of the source object, then the data field is copied from the source object to the destination object.
4. If bit GDFS is set in the Message Object Function Register of the source object, then TXRQ is set in the Message Object Control Register of the destination object.
5. RXPND and NEWDAT are set in the Message Object Control Register of the destination object.
6. A message interrupt request is generated for the destination object if RXIE is set in the Message Object Control Register of the destination object.
7. The current pointer CUR in the FIFO/Gateway Pointer Register of the source object is moved to the next destination object according to the FIFO rules as described in **Chapter 22.2.9.4**. A gateway with a single (static) destination object is obtained by means of setting  $TOP = BOT = CUR = \text{destination object}$ .

The link from the source to the destination object works in the same way as the link from a FIFO source to a FIFO slave. This means that a gateway with an integrated destination FIFO may be created (**Figure 22-15**), where the object on the left in **Figure 22-15** is the gateway source object and the message objects on the right side are the gateway destination objects.

The gateway works in the same way for the reception of data frames (source object is receive object, i.e.  $DIR = 0$ ) as well as for the reception of remote frames (source object is transmit object).



**Figure 22-16 Gateway Transfer from Source to Destination**

### **22.2.9.8 Foreign Remote Requests**

When a remote frame received on a CAN node is stored in a message object, then a transmit request is set in order to trigger the answer (data frame transmission) to the request or to automatically issue a secondary request. If bit FRREN is cleared (FRREN = 0) in the Function Control register of the message object where the remote request is stored, then TXRQ is set in the Control Register of the same message object.

If bit FRREN is set (FRREN = 1: foreign remote request enabled) then TXRQ is set in the message object that is referenced by pointer CUR in the FIFO/Gateway Pointer Register. The value of CUR is, however, not changed by this feature.

Although the foreign remote request feature works independently from the selected message mode, it is especially useful for gateways to issue a remote request on the source of a gateway upon the reception of a remote request on the gateway destination. According to the setting of FRREN in the gateway destination object there are two ways to handle remote requests that appear on the destination side (assuming that the source object is a receive object and the destination is a transmit object, i.e.  $DIR_{source} = 0$  and  $DIR_{destination} = 1$ ):

#### **FRREN = 0 in the Gateway Destination Object**

1. A remote frame is received by gateway destination.
2. TXRQ is set automatically in the gateway destination object.
3. A data frame with the current data stored in the destination object is transmitted on the destination bus.

#### **FRREN = 1 in the Gateway Destination Object**

1. A remote frame is received by gateway destination.
2. TXRQ is set automatically in the gateway source object (must be referenced by CUR pointer of the destination object).
3. A remote request is transmitted by the source object (which is a receive object) on the source CAN bus.
4. The receiver of the remote request responds with a data frame on the source bus.
5. The data frame is stored in the source object.
6. The data frame is copied to the destination object (gateway action).
7. TXRQ is set in the destination object (assuming  $GDFS_{source} = 1$ ).
8. The new data stored in the destination object is transmitted on the destination bus, as response to the initial remote request on the destination bus.

## 22.3 MultiCAN Kernel Registers

The register set of the MultiCAN module consists of three distinct subsets:

1. The **Global Module Registers** apply to the whole MultiCAN module and exist only once.
2. The **CAN Node Registers** apply to a single CAN node and thus exist once for each CAN node.
3. The collection of **Message Object Registers** defines a single message object and thus exists once for each message object.

### 22.3.1 Register Address Map

**Table 22-5** shows the address map of the MultiCAN module with respect to the base address of the MultiCAN module.

**Table 22-5 MultiCAN Address Map (Relative to MultiCAN Base Address)**

Register Group	Start address	Total Range
Global Module Registers	+100 <sub>H</sub>	+100 <sub>H</sub> to + 1FF <sub>H</sub>
CAN Node Registers for CANx, x = 0 - 5	+200 <sub>H</sub>	+200 <sub>H</sub> to + 7FF <sub>H</sub>
Message Objects n = 0 - 255	+1000 <sub>H</sub>	+1000 <sub>H</sub> to + 2FFF <sub>H</sub>

**Controller Area Network (MultiCAN) Controller**

**Global Module Registers**

The global module registers exist only once. They are listed in [Table 22-6](#) with their relative address with respect to the start address of the Global Module Registers.

**Table 22-6 Relative Addresses of Global Module Registers**

<b>Register</b>	<b>Rel. Address</b>	<b>Full Name of Register</b>
LIST0L	100 <sub>H</sub>	List Registers 0 Low
LIST0H	102 <sub>H</sub>	List Registers 0 High
LIST1L	104 <sub>H</sub>	List Registers 1 Low
LIST1H	106 <sub>H</sub>	List Registers 1 High
LIST2L	108 <sub>H</sub>	List Registers 2 Low
LIST2H	10A <sub>H</sub>	List Registers 2 High
LIST3L	10C <sub>H</sub>	List Registers 3 Low
LIST3H	10E <sub>H</sub>	List Registers 3High
LIST4L	110 <sub>H</sub>	List Registers 4 Low
LIST4H	112 <sub>H</sub>	List Registers 4 High
LIST5L	114 <sub>H</sub>	List Registers 5 Low
LIST5H	116 <sub>H</sub>	List Registers 5 High
LIST6L	118 <sub>H</sub>	List Registers 6 Low
LIST6H	11A <sub>H</sub>	List Registers 6 High
LIST7L	11C <sub>H</sub>	List Registers 7 Low
LIST7H	11E <sub>H</sub>	List Registers 7 High
MSPND0L	140 <sub>H</sub>	Message Pending Registers 0 Low
MSPND0H	142 <sub>H</sub>	Message Pending Registers 0 High
MSPND1L	144 <sub>H</sub>	Message Pending Registers 1 Low
MSPND1H	146 <sub>H</sub>	Message Pending Registers 1 High
MSPND2L	148 <sub>H</sub>	Message Pending Registers 2 Low
MSPND2H	14A <sub>H</sub>	Message Pending Registers 2 High
MSPND3L	14C <sub>H</sub>	Message Pending Registers 3 Low
MSPND3H	14E <sub>H</sub>	Message Pending Registers 3 High
MSPND4L	150 <sub>H</sub>	Message Pending Registers 4 Low
MSPND4H	152 <sub>H</sub>	Message Pending Registers 4 High
MSPND5L	154 <sub>H</sub>	Message Pending Registers 5 Low

**Controller Area Network (MultiCAN) Controller**

**Table 22-6 Relative Addresses of Global Module Registers**

<b>Register</b>	<b>Rel. Address</b>	<b>Full Name of Register</b>
MSPND5H	156 <sub>H</sub>	Message Pending Registers 5 High
MSPND6L	158 <sub>H</sub>	Message Pending Registers 6 Low
MSPND6H	15A <sub>H</sub>	Message Pending Registers 6 High
MSPND7L	15C <sub>H</sub>	Message Pending Registers 7 Low
MSPND7H	146 <sub>H</sub>	Message Pending Registers 7 High
MSID0	180 <sub>H</sub>	Message Index Registers 0
MSID1	184 <sub>H</sub>	Message Index Registers 1
MSID2	188 <sub>H</sub>	Message Index Registers 2
MSID3	18C <sub>H</sub>	Message Index Registers 3
MSID4	190 <sub>H</sub>	Message Index Registers 4
MSID5	194 <sub>H</sub>	Message Index Registers 5
MSID6	198 <sub>H</sub>	Message Index Registers 6
MSID7	19C <sub>H</sub>	Message Index Registers 7
MSIMASKL	1C0 <sub>H</sub>	Message Index Mask Register Low
MSIMASKH	1C2 <sub>H</sub>	Message Index Mask Register High
PANCTRL	1C4 <sub>H</sub>	Panel Control Register Low
PANCTRH	1C6 <sub>H</sub>	Panel Control Register High
MCR	1C8 <sub>H</sub>	Module Control Register
MITR	1CC <sub>H</sub>	Module Interrupt Trigger Register
-	+120 <sub>H</sub> ... +13E <sub>H</sub> +148 <sub>H</sub> ... +17E <sub>H</sub> +188 <sub>H</sub> ... +1BE <sub>H</sub> +1CE <sub>H</sub> ... +1FE <sub>H</sub>	Reserved

### **CAN Node Registers**

The registers of a CAN node are located at consecutive 32 bit addresses according to [Table 22-7](#) which shows the relative address of the 32 bit CAN Node Registers with respect to the base address of CAN node register block. The CAN Node Register block exists once for each CAN node.

**Controller Area Network (MultiCAN) Controller**

**Table 22-7 Relative Addresses of CAN Node Registers**

Register	Rel. Address	Full Name of Register
NCR	+00 <sub>H</sub>	CAN Node Control Register
NSR	+04 <sub>H</sub>	CAN Node Status Register
NIPR	+08 <sub>H</sub>	CAN Node Interrupt Pointer Register
NPCR	+0C <sub>H</sub>	CAN Node Port Control Register
NBTRL	+10 <sub>H</sub>	CAN Node Bit Timing Register Low
NBTRH	+12 <sub>H</sub>	CAN Node Bit Timing Register High
NECNTL	+14 <sub>H</sub>	CAN Node Error Counter Register Low
NECNTH	+16 <sub>H</sub>	CAN Node Error Counter Register High
NFCRL	+18 <sub>H</sub>	CAN Node Frame Counter Register Low
NFCRH	+1A <sub>H</sub>	CAN Node Frame Counter Register High
-	+1C <sub>H</sub> to +FE <sub>H</sub>	Reserved

**Message Object Registers**

The registers of a message object are located at consecutive 32 bit addresses according to [Table 22-8](#) which shows the relative address of the 32 bit Message Object Registers with respect to the base address of the Message Object.

**Table 22-8 Relative Addresses of Message Object Registers**

Register	Rel. Address	Full Name of Register
MOFCRL	+00 <sub>H</sub>	Message Object Function Control Register Low
MOFCRH	+02 <sub>H</sub>	Message Object Function Control Register High
MOFGPRL	+04 <sub>H</sub>	Message Object FIFO/Gateway Pointer Reg. Low
MOFGPRH	+06 <sub>H</sub>	Message Object FIFO/Gateway Pointer Reg. High
MOIPRL	+08 <sub>H</sub>	Message Object Interrupt Pointer Register Low
MOIPRH	+0A <sub>H</sub>	Message Object Interrupt Pointer Register High
MOAMRL	+0C <sub>H</sub>	Message Object Acceptance Mask Register Low
MOAMRH	+0E <sub>H</sub>	Message Object Acceptance Mask Register High
MODATALL	+10 <sub>H</sub>	Message Object Data Register Low Low
MODATALH	+12 <sub>H</sub>	Message Object Data Register Low High
MODATAHL	+14 <sub>H</sub>	Message Object Data Register High Low



**Controller Area Network (MultiCAN) Controller**

**Table 22-8 Relative Addresses of Message Object Registers**

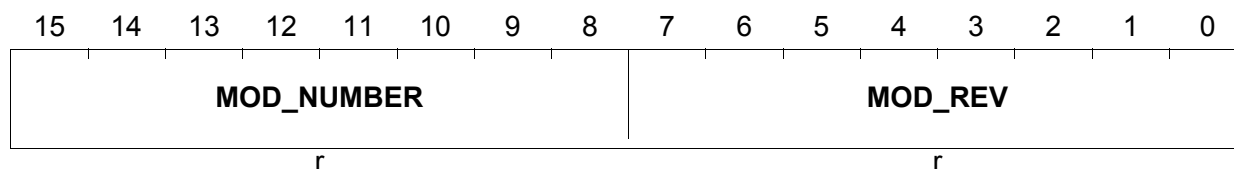
<b>Register</b>	<b>Rel. Address</b>	<b>Full Name of Register</b>
MODATAHH	+16 <sub>H</sub>	Message Object Data Register High High
MOARL	+18 <sub>H</sub>	Message Object Arbitration Register Low
MOARH	+1A <sub>H</sub>	Message Object Arbitration Register High
MOCTRL MOSTATL	+1C <sub>H</sub>	Message Object Control Register Low Message Object Status Register Low
MOCTRH MOSTATH	+1E <sub>H</sub>	Message Object Control Register High Message Object Status Register Low

## 22.3.2 Global MultiCAN Registers

### 22.3.2.1 Module Identification Register

ID

**Module Identification Register** (08<sub>H</sub>) **Reset Value: 4001<sub>H</sub>**



Field	Bits	Type	Description
MOD_REV	[7:0]	r	<b>Module Revision Number Value</b> Bits 7-0 bits are used for module revision numbering. The value of the module revision number starts with 01 <sub>H</sub> (first revision), 02 <sub>H</sub> , 03 <sub>H</sub> , ... up to FF <sub>H</sub> .
MOD_NUMBER	[15:8]	r	<b>Module Identification Number Value</b> Bits 15-8 are used for module identification. The MultiCAN has the module number 40 <sub>H</sub> .

## Controller Area Network (MultiCAN) Controller

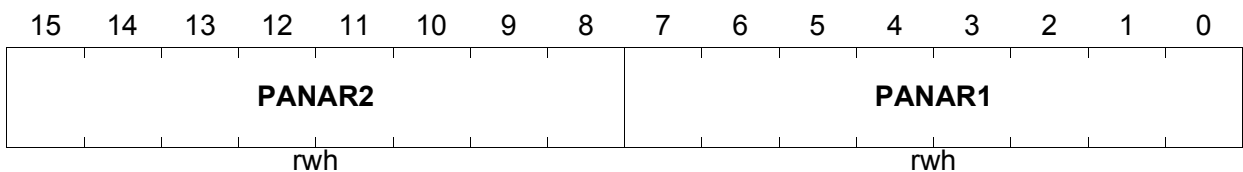
### 22.3.2.2 Command Panel

All list operations such as allocation, deallocation and relocation of message objects within the list structure are performed via the Command Panel. It is not possible to modify the list structure directly by means of writing to the message objects and the LIST registers.

A new command is started by means of writing the command arguments and the command code to the Panel Control Register.

#### PANCTR

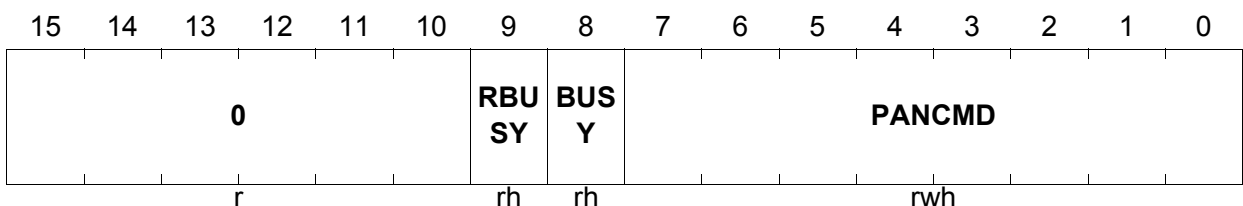
**Panel Control Register** (1C6<sub>H</sub>) **Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
PANAR1	[7:0]	rwh	Panel Argument 1
PANAR2	[15:8]	rwh	Panel Argument 2

#### PANCTRL

**Panel Control Register** (1C4<sub>H</sub>) **Reset Value: 0301<sub>H</sub>**



Field	Bits	Type	Description
PANCMD	[7:0]	rwh	<b>Panel Command</b> A new command is started by means of writing the command number to PANCMD. At the end of a panel command the NOP (no operation) command code is automatically written to PANCMD.

**Controller Area Network (MultiCAN) Controller**

Field	Bits	Type	Description
<b>BUSY</b>	8	rh	<b>Panel Busy</b> 0 <sub>B</sub> Panel has finished command and is ready to accept a new command. 1 <sub>B</sub> Panel operation is in progress.
<b>RBUSY</b>	9	rh	<b>Result Busy</b> 0 <sub>B</sub> No update of PANAR1 and PANAR2 is scheduled by the list controller. 1 <sub>B</sub> A list command is running (BUSY = 1) that will write results to PANAR1 and PANAR2, but the results are not yet available.
<b>0</b>	[10:15]	r	<b>reserved;</b> returns '0' if read; should be written with '0';

**Controller Area Network (MultiCAN) Controller**

**Panel Commands**

A panel operation consists of a command code to be written to PANCMD and up to 2 panel arguments (PANAR1, PANAR2). Commands that have a return value deliver it to the PANAR1 field. Commands that deliver an error flag post it to bit 7 of PANAR2.

**Table 22-9 Panel Commands**

Code	PANAR2	PANAR1	Command Description
0			<b>No Operation</b> Writing value 0 to PANCMD has no effect. No new command is started.
1	<b>Result:</b> Bit 7 : ERR Bit 6-0 : undefined		<b>Initialize Lists</b> Run the initialization sequence to reset the CTRL and LIST field of all message objects and the list registers LIST[7:0] to their reset values. This results in the deallocation of all message objects. The initialization command requires that bits INIT and CCE are set in the Node Control Register of all CAN nodes 0-5. An ERR bit (bit 7 of PANAR2) reports the success of the operation: 0Success 1Not all INIT and CCE bits are set. Thus no initialization is performed. The initialization command is automatically performed with each reset of the MultiCAN module, but with the exception that all message object registers are reset.
2	<b>Argument:</b> List Index	<b>Argument:</b> Message Object Number	<b>Static Allocate</b> Allocate a given message object to a list. The message object is removed from the list that it currently belongs to and appended to the end of the list. given by PANAR2. This command is also used to deallocate a message object. In this case the target list is the list of unallocated elements. (PANAR2 = 0).

**Controller Area Network (MultiCAN) Controller**

**Table 22-9 Panel Commands**

<b>Code</b>	<b>PANAR2</b>	<b>PANAR1</b>	<b>Command Description</b>
<b>3</b>	<b>Argument:</b> List Index <b>Result:</b> Bit 7 : ERR Bit 6-0 : undefined	<b>Result:</b> Message Object Number	<b>Dynamic Allocate</b> Allocate the first message object of the list of unallocated objects to the selected list. The message object is appended to the end of the list. The message number of the message object is returned in PANAR1. An ERR bit (bit 7 of PANAR2) reports the success of the operation: 0Success. 1The operation has not been performed because the list of unallocated elements was empty.
<b>4</b>	<b>Argument:</b> Destination Object Number	<b>Argument:</b> Source Object Number	<b>Static Insert Before</b> Remove a message object (source object) from the list that it currently belongs to and insert it before a given destination object into the list structure of the destination object. The source object thus becomes the predecessor of the destination object.
<b>5</b>	<b>Argument:</b> Destination Object Number <b>Result:</b> Bit 7 : ERR Bit 6-0 : undefined	<b>Result:</b> Object Number of inserted object	<b>Dynamic Insert Before</b> Insert a new message object before a given destination object. The new object is taken from the list of unallocated elements (the first element is chosen). The number of the new object is delivered as result to PANAR1.  An ERR bit (bit 7 of PANAR2) reports the success of the operation: 0Success. 1The operation has not been performed because the list of unallocated elements was empty.
<b>6</b>	<b>Argument:</b> Destination Object Number	<b>Argument:</b> Source Object Number	<b>Static Insert Behind</b> Remove a message object (source object) from the list that it currently belongs to and insert it behind a given destination object into the list structure of the destination object. The source object thus becomes the successor of the destination object.

**Controller Area Network (MultiCAN) Controller**

**Table 22-9 Panel Commands**

<b>Code</b>	<b>PANAR2</b>	<b>PANAR1</b>	<b>Command Description</b>
<b>7</b>	<b>Argument:</b> Destination Object Number <b>Result:</b> Bit 7 : ERR Bit 6-0 : undefined	<b>Result:</b> Object Number of inserted object	<b>Dynamic Insert Behind</b> Insert a new message object behind a given destination object. The new object is taken from the list of unallocated elements (the first element is chosen). The number of the new object is delivered as result to PANAR1.  An ERR bit (bit 7 of PANAR2) reports the success of the operation: 0Success. 1The operation has not been performed because the list of unallocated elements was empty.
<b>8 - 255</b>	-	-	<b>Reserved</b>

**Controller Area Network (MultiCAN) Controller**

### 22.3.2.3 Module Control Register

The Module Control Register contains basic settings to define the operation of the module.

#### MCR

#### Module Control Register

(1C8<sub>H</sub>)

Reset Value: 0000<sub>H</sub>

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

Field	Bits	Type	Description
MPSEL	[15:12]	rw	<b>Message Pending Selector</b> MPSEL allows to calculate the bit position of the message pending bit to be set after a message reception/transmission interrupt from a mixture of RXINP, TXINP and MPN (Message Pending Number). With the definitions INP ... RXINP upon message reception, TXINP upon message transmission MPN ... 8 bit message pending number the effective position of the message pending bit is calculated according to the formula $POS = ( (INP \& MPSEL) \ll 4 ) \mid (MPN \& (\sim MPSEL \ll 4)) \mid (MPN \& = 0x0F_H)$  If MPSEL = 0 then the position is simply given by the message pending number MPN. If MPSEL = 1111 <sub>B</sub> then the upper 4 bits of the position is given by the interrupt output line pointer INP and the lower 4 bits are taken from MPN.
0	[11:0]	r	<b>reserved;</b> returns '0' if read; should be written with '0';

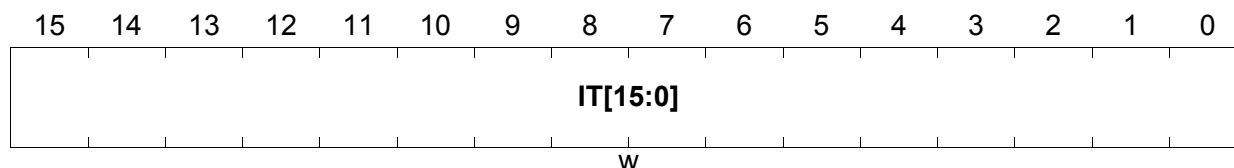


### 22.3.2.4 Interrupt Trigger Register ITR

The Interrupt Trigger Register ITR allows to trigger interrupt requests on each interrupt output line by software.

#### MITR

**Module Interrupt Trigger Register (1CC<sub>H</sub>)** **Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
IT[15:0]	[15:0]	w	<b>Interrupt Trigger</b> Writing value 1 to bit n (n = 15-0) generates an interrupt request on interrupt output line n. Writing value 0 has no effect. Reading delivers always 0. More than one interrupt request may be generated at the same time by means of writing 1 to several bit positions of IT with a single write access.

### 22.3.2.5 List Pointer

Each CAN node has an own list which defines the message objects that are allocated to the respective node. In addition to that there is the list of all unallocated objects and finally a general purpose user list which is not associated to a CAN node. Each list is assigned a list index according to [Table 22-3 “List Indices” on Page 22-14](#).

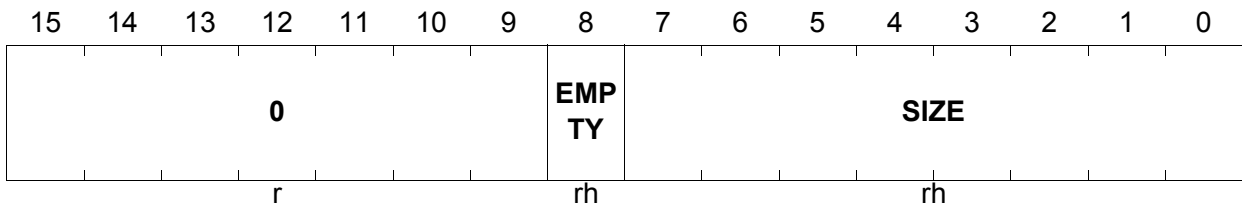
Each list is terminated with a List Register which defines the first and the last element in the list.

#### LIST0H

**List Register 0 High** (102<sub>H</sub>) **Reset Value: 007F<sub>H</sub>**

**LISTyH (y = 1-7)**

**List Register y High** (102<sub>H</sub>+y\*4) **Reset Value: 0100<sub>H</sub>**



Field	Bits	Type	Description
<b>SIZE</b>	[7:0]	rh	<b>Size of List</b> The number of elements in the list l is given by #elements = SIZE + 1, provided the list is not empty. If the list l is empty, the value of SIZE is zero.
<b>EMPTY</b>	8	rh	<b>List Empty Indication</b> 0 <sub>B</sub> At least one message object is allocated to list l. 1 <sub>B</sub> No message object is allocated to the list l.
<b>0</b>	[15:9]	r	<b>Reserved;</b> read as 0; should be written with 0.

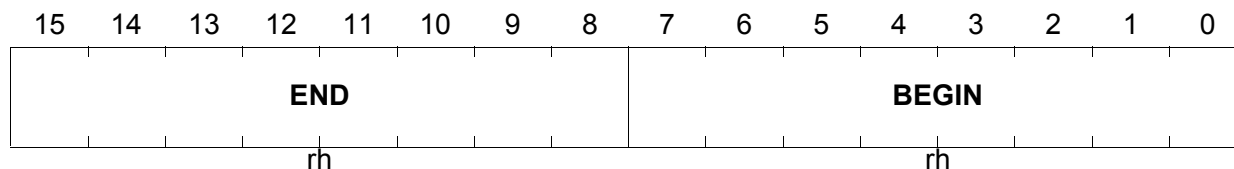
**Controller Area Network (MultiCAN) Controller**

**LIST0L**

**List Register 0 Low** (100<sub>H</sub>) **Reset Value: 7F00<sub>H</sub>**

**LISTxL (x = 1-7)**

**List Register x Low** (100<sub>H</sub>+x\*4) **Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
<b>BEGIN</b>	[7:0]	rh	<b>List Begin</b> Pointer to the first message object in the list l.
<b>END</b>	[15:8]	rh	<b>END Pointer</b> Pointer to the last message object in the list l.

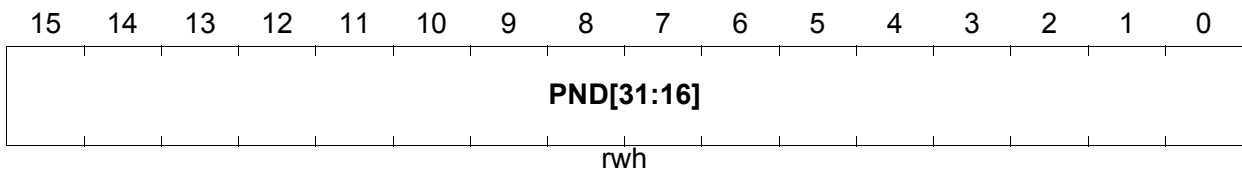
### 22.3.2.6 Message Notifications

When a message object generates an interrupt request upon the transmission or reception of a message, then the request is routed to the interrupt output line selected by TXINP or RXINP of the message object. As there are more message objects than interrupt output lines, an interrupt routine typically processes requests from more than one message object. Therefore a priority selection mechanism is implemented in the MultiCAN module to select the highest priority object within a collection of message objects. The Message Pending Register contains the interrupt pending.

#### MSPNDkH (k = 0-7)

**Message Pending Register k High (142<sub>H</sub>+k\*4)**

**Reset Value: 0000<sub>H</sub>**

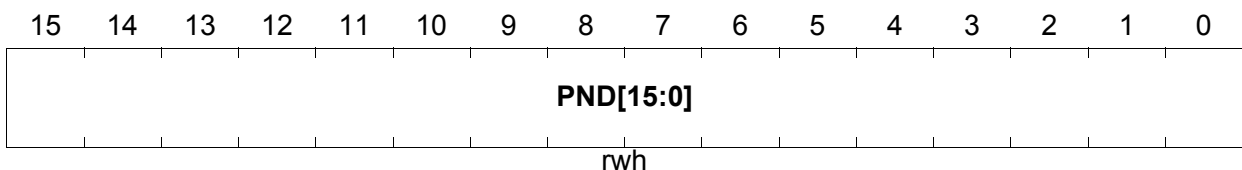


Field	Bits	Type	Description
PND[31:16]	[15:0]	rwh	<b>Message Pending</b> When a message interrupt occurs then the message object sets a bit in one of the MSPND register, where the bit position is given by the MPN[4:0] field of the IPR register of the message object. The register selection n is given by the higher bits of MPN. The register bits may be cleared by SW (write 0), but writing 1 has no effect.

#### MSPNDkL (k = 0-7)

**Message Pending Register k Low (140<sub>H</sub>+k\*4)**

**Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
PND[15:0]	[15:0]	rwh	<b>Message Pending</b> The same as PND[31:16]

## Controller Area Network (MultiCAN) Controller

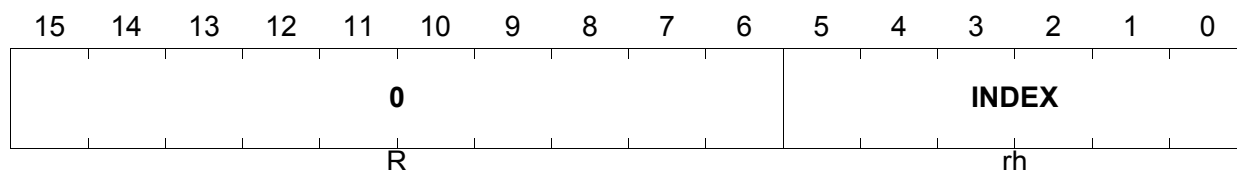
Each Message Pending Register has a Message Index Register associated to it. The Message Index Register shows the active (set) pending bit with lowest bit position within groups of pending bits.

**MSIDk (k = 0-7)**

**Message Index Register k**

**(180<sub>H</sub>+k\*4)**

**Reset Value: 0020<sub>H</sub>**



Field	Bits	Type	Description
INDEX	[5:0]	rh	<b>Message Pending Index</b> The value of INDEX is given by the bit position i of the pending bit of MSPNDk with the following properties: 1. MSPNDk[i] & IM[i] = 1 2. i = 0 or MSPNDk[i-1:0] & IM[i-1:0] = 0 If no bit of MSPNDk satisfies these conditions then INDEX reads 100000 <sub>B</sub> . Thus INDEX shows the position of the first pending bit of MSPNDk, where only those bits of MSPNDk which are selected in the Message Index Mask Register are taken into account.
0	[15:6]	r	<b>Reserved:</b> read as 0; should be written with 0.

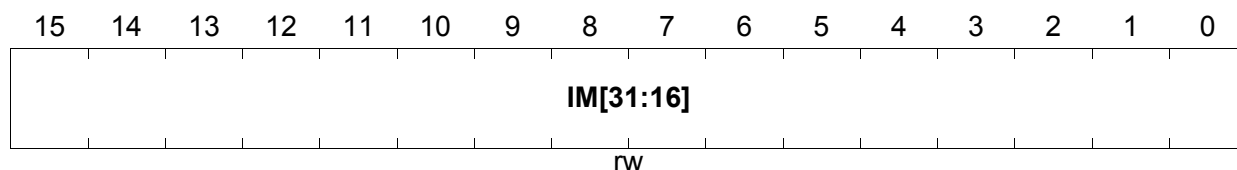
## Controller Area Network (MultiCAN) Controller

The Message Index Mask Register selects individual bits for the calculation of the Message Pending Index. The Message Index Mask Register is used commonly for all Message Pending registers and their associated Message Index registers.

### MSIMASKH

**Message Index Mask Register High (1C2<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

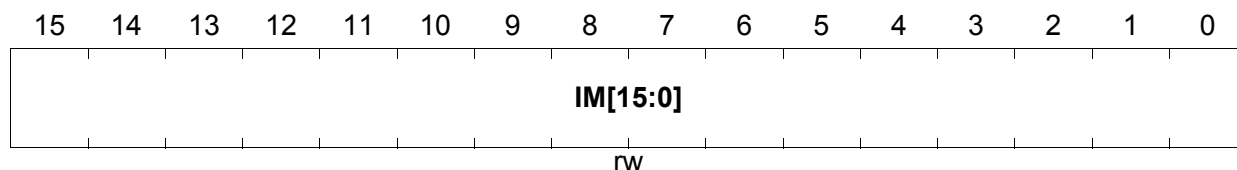


Field	Bits	Type	Description
IM[31:16]	[15:0]	rw	<b>Message Index Mask</b> Only those bits in MSPNDk for which the corresponding Index Mask bits are set contribute to the calculation of the Message Index.

### MSIMASKL

**Message Index Mask Register Low (1C0<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
IM[15:0]	[15:0]	rw	<b>Message Index Mask</b> Only those bits in MSPNDk for which the corresponding Index Mask bits are set contribute to the calculation of the Message Index.

## Controller Area Network (MultiCAN) Controller

### 22.3.3 CAN Node Specific Registers

The CAN node specific registers exist once for each CAN node of the MultiCAN module. They contain information that is directly related to the operation of the CAN nodes and which may not be shared among the nodes.

#### 22.3.3.1 Node Control Register

The Node Control Register contains basic settings that define the operation of the CAN node and the interaction of the CAN node with the message objects.

**NCRx (x = 0-5)**

**Node x Control Register**

**(200H+x\*100<sub>H</sub>)**

**Reset Value: 0001<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0							<b>SUS EN</b>	<b>CAL M</b>	<b>CCE</b>	<b>0</b>	<b>CAN DIS</b>	<b>ALIE</b>	<b>LECI E</b>	<b>TRIE</b>	<b>INIT</b>
r							rw	rw	rw	r	rw	rw	rw	rw	rwh

**Controller Area Network (MultiCAN) Controller**

Field	Bits	Type	Description
INIT	0	rwh	<p><b>Node Initialization</b></p> <p>0<sub>B</sub> Resetting bit INIT enables the participation of the node in the CAN traffic.  If the CAN node is in the bus off state then the ongoing bus off recovery (which does not depend on the INIT bit) is continued. With the end of the bus off recovery sequence the CAN node is allowed to take part in the CAN traffic.  If the CAN node is not in the bus off state a sequence of 11 consecutive recessive bits must be detected before the node is allowed to take part in the CAN traffic.</p> <p>1<sub>B</sub> Setting this bit terminates the participation of this node in the CAN traffic. Any ongoing frame transfer is cancelled and the transmit line goes recessive.  If the CAN node is in the bus off state then the running bus off recovery sequence is continued. If the INIT bit is still set after the successful completion of the bus off recovery sequence, i.e. after detecting 128 sequences of 11 consecutive recessive bits (11 x 1) then the CAN node leaves the bus off state but remains inactive as long as INIT remains set.</p> <p>Bit INIT is automatically set when the CAN node becomes 'bus off' (see <a href="#">Page 22-11</a>).</p>
TRIE	1	rw	<p><b>Transfer Interrupt Enable</b></p> <p>If this bit is set, then an interrupt request is generated upon the successful reception or transmission of a CAN frame. The interrupt output line is selected by TRINP in the CAN Node Interrupt Pointer Register.</p>
LECIE	2	rw	<p><b>LEC indicated Error Interrupt Enable</b></p> <p>If this bit is set, then an interrupt request is generated upon each update of the LEC field in the Node Status Register leading to LEC &gt; 0 (CAN protocol error). The interrupt output line is selected by LECINP in the CAN Node Interrupt Pointer Register.</p>



**Controller Area Network (MultiCAN) Controller**

Field	Bits	Type	Description
<b>ALIE</b>	3	rw	<b>Alert Interrupt Enable</b> If this bit is set then an alert interrupt is generated on one of the following events: <ol style="list-style-type: none"> <li>1) change of bit BOFF in the CAN Node Status Register.</li> <li>2) change of bit EWRN in the CAN Node Status Register.</li> <li>3) List Length Error, which also sets bit LLE in the CAN Node Status Register.</li> <li>4) List Object Error, which also sets bit LOE in the CAN Node Status Register.</li> <li>5) Bit INIT has been set by the MultiCAN.</li> </ol> The interrupt is requested on the interrupt output line selected by ALINP in the CAN Node Interrupt Pointer Register.
<b>CANDIS</b>	4	rw	<b>CAN Disable</b> Setting this bit disables the CAN node. The CAN node first waits until it is BUS IDLE or BUS OFF. Then bit INIT is automatically set and an alert interrupt is generated if bit ALIE is set.
<b>CCE</b>	6	rw	<b>Configuration Change Enable</b> 0 <sub>B</sub> The Bit Timing Register, the Port Control Register and the Error Counter Register may only be read. All attempts to modify them are ignored. 1 <sub>B</sub> The Bit Timing Register, the Port Control Register and the Error Counter Register may be read and written.
<b>CALM</b>	7	rw	<b>Can Analyze Mode</b> If this bit is set then the CAN node operates in analyze mode. This means that messages may be received, but not transmitted. No acknowledge is sent on the CAN bus upon frame reception. Active error flags are sent recessive instead of dominant. The transmit line is continuously held at recessive (1) level. Bit CALM can be written only while bit INIT is set.

**Controller Area Network (MultiCAN) Controller**

Field	Bits	Type	Description
<b>SUSEN</b>	8	rw	<b>Suspend Enable</b> This bit allows to set the CAN node into suspend mode via OCDS (on chip debug support): 0 <sub>B</sub> An OCDS suspend trigger is ignored by the CAN node. 1 <sub>B</sub> An OCDS suspend trigger disables the CAN node: As soon as the CAN node becomes BUS IDLE or BUS OFF bit INIT is internally forced to '1' to disable the CAN node. The actual value of bit INIT remains unchanged. Bit SUSEN is reset via a debug reset.
<b>0</b>	5, [15:9]	r	<b>Reserved;</b> read as 0; should be written with 0.

### 22.3.3.2 Node Status Register

The Node Status Register reports errors as well as successfully transferred CAN frames.

**NSRx (x = 0-5)**

**Node x Status Register**

**(204H+x\*100<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0					SUS ACK	LOE	LLE	BOF F	EWR N	ALE RT	RXO K	TXO K	LEC		
r					rh	rwh	rwh	rh	rh	rwh	rwh	rwh	rwh		

Field	Bits	Type	Description
LEC	[2:0]	rwh	<b>Last Error Code</b> The encoding of this bit field is detailed in <a href="#">Table 22-10</a> .
TXOK	3	rwh	<b>Message Transmitted Successfully</b> 0 <sub>B</sub> No successful transmission since last flag reset. 1 <sub>B</sub> A message has been transmitted successfully (error free and acknowledged by at least another node). TXOK must be reset by software (write 0). Writing 1 has no effect.
RXOK	4	rwh	<b>Message Received Successfully</b> 0 <sub>B</sub> No successful reception since last flag reset. 1 <sub>B</sub> A message has been received successfully. RXOK must be reset by software (write 0). Writing 1 has no effect.

**Controller Area Network (MultiCAN) Controller**

Field	Bits	Type	Description
<b>ALERT</b>	5	rwh	<b>Alert Warning</b> The ALERT bit is set upon the occurrence of one of the following events (the same events which also trigger an alert interrupt if ALIE is set): <ol style="list-style-type: none"> <li>1) Change of bit BOFF in the CAN Node Status Register.</li> <li>2) Change of bit EWRN in the CAN Node Status Register.</li> <li>3) List Length Error, which also sets bit LLE in the CAN Node Status Register.</li> <li>4) List Object Error, which also sets bit LOE in the CAN Node Status Register.</li> <li>5) Bit INIT has been set by the MultiCAN.</li> </ol> ALERT must be reset by software (write 0). Writing 1 has no effect.
<b>EWRN</b>	6	rh	<b>Error Warning Status</b> 0 <sub>B</sub> No warning limit exceeded. 1 <sub>B</sub> One of the error counters REC or TEC reached the warning limit EWRNLVL.
<b>BOFF</b>	7	rh	<b>Bus-off Status</b> 0 <sub>B</sub> CAN controller is not in the bus-off state. 1 <sub>B</sub> CAN controller is in the bus-off state.
<b>LLE</b>	8	rwh	<b>List Length Error</b> 0 <sub>B</sub> No list length error since last flag reset. 1 <sub>B</sub> List length error has been detected during message acceptance filtering. The number of elements in the list that belongs to this CAN node differs from the list SIZE given in the list termination pointer. LLE must be reset by software (write 0). Writing 1 has no effect.
<b>LOE</b>	9	rwh	<b>List Object Error</b> 0 <sub>B</sub> No list object error since last flag reset. 1 <sub>B</sub> List object error has been detected during message acceptance filtering. A message object with wrong LIST index entry in the Message Object Control Register has been detected. LOE must be reset by software (write 0). Writing 1 has no effect.

**Controller Area Network (MultiCAN) Controller**

Field	Bits	Type	Description
<b>SUSACK</b>	10	rh	<b>Suspend Acknowledge</b> $0_B$ The CAN node is not in suspend mode or a suspend request is pending, but the CAN node has not yet reached BUS IDLE or BUS OFF. $1_B$ The CAN node is in suspend mode: The CAN node is inactive (bit NCR.INIT internally forced to '1') due to an OCDS suspend request.
<b>0</b>	[15:11]	r	<b>Reserved;</b> read as 0; should be written with 0.

## Encoding of the LEC Bitfield

**Table 22-10 Encoding of the LEC Bit Field**

<b>LEC Value</b>	<b>Signification</b>
000 <sub>B</sub>	<u>No Error:</u> No error was detected for the last message on the CAN bus.
001 <sub>B</sub>	<u>Stuff Error:</u> More than 5 equal bits in a sequence have occurred in a part of a received message where this is not allowed.
010 <sub>B</sub>	<u>Form Error:</u> A 'fixed format part' of a received frame has the wrong format.
011 <sub>B</sub>	<u>Ack Error:</u> The transmitted message was not acknowledged by another node.
100 <sub>B</sub>	<u>Bit1 Error:</u> During a message transmission the CAN node tried to send a recessive level (1) outside the arbitration field and the acknowledge slot, but the monitored bus value was dominant.
101 <sub>B</sub>	<u>Bit0 Error:</u> Two different conditions are signalled by this code: a) During transmission of a message (or acknowledge bit, active error flag, overload flag) the CAN node tried to send a dominant level (0), but the monitored bus value was recessive. b) During bus-off recovery this code is set each time a sequence of 11 recessive bits has been monitored. The CPU may use this code as indication that the bus is not continuously disturbed.
110 <sub>B</sub>	<u>CRC Error:</u> The CRC checksum of the received message was incorrect.
111 <sub>B</sub>	<u>CPU write to LEC:</u> Whenever the the CPU writes the value 111 to LEC, it takes the value 111. Whenever the CPU writes another value to LEC, the written LEC value is ignored.

### 22.3.3.3 Node Interrupt Pointer Register

The Node Interrupt Pointer Register connects each interrupt request source of the CAN node to one of the up to 16 available interrupt output lines.

**NIPRx (x = 0-5)**

**Node x Interrupt Pointer Register (208H+x\*100<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>CFCINP</b>				<b>TRINP</b>				<b>LECINP</b>				<b>ALINP</b>			
rw				rw				rw				rw			

Field	Bits	Type	Description
<b>ALINP</b>	[3:0]	rw	<b>Alert Interrupt Node Pointer</b> Number of interrupt output line INT_Om (m=0-15) reporting the “Alert Interrupt Request”, if enabled by ALIE = 1.
<b>LECINP</b>	[7:4]	rw	<b>Last Error Code Interrupt Node Pointer</b> Number of interrupt output line INT_Om (m=0-15) reporting the “Last Error Interrupt Request”, if enabled by LECIE = 1.
<b>TRINP</b>	[11:8]	rw	<b>Transfer OK Interrupt Node Pointer</b> Number of interrupt output line INT_Om (m=0-15) reporting the “Transfer Interrupt Request”, if enabled by TRIE.
<b>CFCINP</b>	[15:12]	rw	<b>Frame Counter Interrupt Node Pointer</b> Number of interrupt output line INT_Om (m=0-15) reporting the “Frame Counter Overflow Interrupt Request”, if enabled by CFCIE = 1.

**Controller Area Network (MultiCAN) Controller**

### 22.3.3.4 Node Port Control Register

The Node Port Control Register configures the CAN bus transmit/receive ports. NPCRx may be written only if bit NCRx.CCE is set.

**NPCR<sub>x</sub> (x = 0-5)**

**Node x Port Control Register (20CH+x\*100<sub>H</sub>)** **Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>0</b>							<b>LBM</b>	<b>0</b>				<b>RXSEL</b>			
r							rw	r				rw			

Field	Bits	Type	Description
<b>RXSEL</b>	[2:0]	rw	<b>Receive Select</b> RXSEL selects one out of 8 possible receive inputs. CAN traffic is performed through the selected input. The other inputs are ignored. See also "Receive Input Selection" Section
<b>LBM</b>	8	rw	<b>Loop Back Mode</b> 0 <sub>B</sub> Loop back mode is disabled. 1 <sub>B</sub> Loop back mode is enabled. This node is connected to an internal (virtual) loop back CAN bus. All CAN nodes which are in loop back mode are connected to this virtual CAN bus so that they can communicate with each other internally. The external transmit line is forced recessive in loop back mode.
<b>0</b>	[7:3], [15:9]	r	<b>Reserved;</b> read as 0; should be written with 0.



### 22.3.3.5 Node Bit Timing Register

The Node Bit Timing Register contains all parameters to setup the bit timing for the CAN transfer. NBTRx may be written only if bit NCRx.CCE is set.

#### NBTRxH (x = 0-5)

**Node x Bit Timing Register High (212H+x\*100<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

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

Field	Bits	Type	Description
FTX	0	rw	<b>Fast Transmit (TTC only)</b> When a message is requested for transmission on the CAN bus, then the start of frame (SOF) symbol is sent with the beginning of a new bit time.  If the CAN bus is in the idle state and bit FTX is set (FTX = 1) then a new bit time is started immediately with the transmit trigger of a new message. This eliminates the variable delay between the transmit trigger of a message and the actual SOF signal on the transmit output. Such a variable delay occurs when transmit triggers occur at different positions within a CAN bit time.
0	[15:1]	r	<b>reserved;</b> returns '0' if read; should be written with '0';

#### NBTRxL (x = 0-5)

**Node x Bit Timing Register Low (210H+x\*100<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DIV8		TSEG2			TSEG1			SJW		BRP					
rw		rw			rw			rw		rw					

**Controller Area Network (MultiCAN) Controller**

Field	Bits	Type	Description
<b>BRP</b>	[5:0]	rw	<b>Baud Rate Prescaler</b> The duration of one time quantum is given by (BRP + 1) clock cycles if DIV8 = 0. The duration of one time quantum is given by 8 × (BRP + 1) clock cycles if DIV8 = 1.
<b>SJW</b>	[7:6]	rw	<b>(Re)Synchronization Jump Width</b> (SJW + 1) time quanta are allowed for resynchronization.
<b>TSEG1</b>	[11:8]	rw	<b>Time Segment Before Sample Point</b> (TSEG1 + 1) time quanta is the user defined nominal time between the end of the synchronization segment and the sample point. It includes the propagation segment, which takes into account signal propagation delays. The time segment may be lengthened due to resynchronization. Valid values for TSEG1 are 2 to 15.
<b>TSEG2</b>	[14:12]	rw	<b>Time Segment After Sample Point</b> (TSEG2 + 1) time quanta is the user defined nominal time between the sample point and the start of the next synchronization segment. It may be shortened due to resynchronization. Valid values for TSEG2 are 1 to 7.
<b>DIV8</b>	15	rw	<b>Divide Prescaler Clock by 8</b> 0 <sub>B</sub> A time quantum lasts (BRP+1) clock cycles. 1 <sub>B</sub> A time quantum lasts 8 × (BRP+1) clock cycles.

### 22.3.3.6 Node Error Counter Register

**NECNTxH (x = 0-5)**

**Node x Error Counter Register High(216H+x\*100<sub>H</sub>)**

**Reset Value: 0060<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>0</b>						<b>LEIN C</b>	<b>LET D</b>	<b>EWRNLVL</b>							
r						rh	rh	rw							

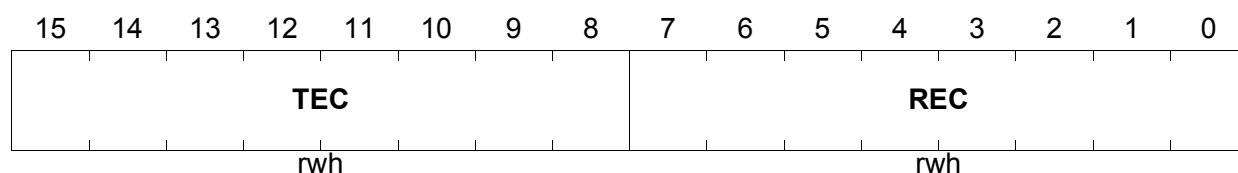
**Controller Area Network (MultiCAN) Controller**

Field	Bits	Type	Description
<b>EWRNLVL</b>	[7:0]	rw	<b>Error Warning Level</b> Bit field EWRNLVL defines the threshold value (warning level, default 96) to be reached in order to set the corresponding error warning bit EWRN.
<b>LETD</b>	8	rh	<b>Last Error Transfer Direction</b> $0_B$ The last error occurred while the CAN node was receiver (REC has been incremented). $1_B$ The last error occurred while the CAN node was transmitter (TEC has been incremented).
<b>LEINC</b>	9	rh	<b>Last Error Increment</b> $0_B$ The last error led to an error counter increment of 1. $1_B$ The last error led to an error counter increment of 8.
<b>0</b>	[15:10]	r	<b>Reserved;</b> read as 0; should be written with 0.

**NECNTxL (x = 0-5)**

**Node x Error Counter Register Low(214H+x\*100<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
<b>REC</b>	[7:0]	rwh	<b>Receive Error Counter</b> Bit field REC contains the value of the receive error counter of the CAN node.
<b>TEC</b>	[15:8]	rwh	<b>Transmit Error Counter</b> Bit field TEC contains the value of the transmit error counter of the CAN node.

## Controller Area Network (MultiCAN) Controller

### 22.3.3.7 Node Frame Counter Register

The Node Frame Counter Register contains the actual value of the frame counter as well as control and status bits of the frame counter.

**NFCRxH (x = 0-5)**

**Node x Frame Counter Register High(21AH+x\*100<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0								CFC OV	CFCI E	0	CFMOD	CFSEL			
r								rwh	rw	r	rw	rw			

Field	Bits	Type	Description
CFSEL	[2:0]	rw	<p><b>CAN Frame Count Selection</b></p> <p>This bit field selects the function of the frame counter for the chosen frame count mode.</p> <p><b>Frame Count Mode</b></p> <p>Bit 0 If Bit 0 of CFSEL is set then CFC is incremented each time a foreign frame (i.e. a frame not matching to a message object) has been received on the CAN bus.</p> <p>Bit 1 If Bit 1 of CFSEL is set then CFC is incremented each time a frame matching to a message object has been received on the CAN bus.</p> <p>Bit 2 If Bit 2 of CFSEL is set then CFC is incremented each time a frame has been transmitted successfully by the node.</p> <p><b>Time Stamp Mode</b></p> <p>The frame counter is incremented (internally) with the beginning of a new bit time. Its value is permanently sampled in the CFC field while the bus is idle. The value sampled just before the SOF bit of a new frame is detected is written to the corresponding message object. When the treatment of a message object is finished, the sampling continues.</p> <p><b>Bit Timing Mode</b></p> <p>The available bit timing measurement modes are shown in <a href="#">Table 22-11</a>. If CFCIE is set then an interrupt on request node x (where x is the CAN node index) is generated with a CFC update.</p>

**Controller Area Network (MultiCAN) Controller**

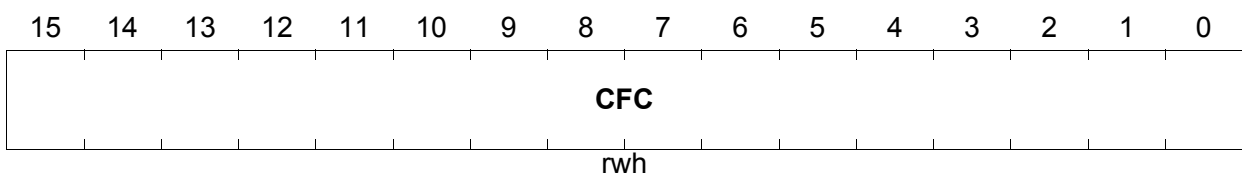
Field	Bits	Type	Description
<b>CFMOD</b>	[4:3]	rw	<b>CAN Frame Counter Mode</b> This bit field defines the operation mode of the frame counter. 00 <sub>B</sub> Frame Count Mode: The frame counter is incremented upon the reception and transmission of frames. 01 <sub>B</sub> Time Stamp Mode: The frame counter is used to count CAN bit times. 10 <sub>B</sub> Bit Timing Mode: The frame counter is used for analysis of the bit timing. <sup>1)</sup> 11 <sub>B</sub> reserved
<b>CFCIE</b>	6	rw	<b>CAN Frame Count Interrupt Enable</b> 0 <sub>B</sub> CAN Frame Counter Overflow interrupt request is disabled. 1 <sub>B</sub> CAN Frame Counter Overflow interrupt request is enabled.
<b>CFCOV</b>	7	rwh	<b>CAN Frame Counter Overflow Flag</b> Flag CFCOV is set upon a frame counter overflow (transition from FFFF <sub>H</sub> to 0000 <sub>H</sub> ). In bit timing analysis mode CFCOV is set upon an update of CFC. An interrupt request is generated if CFCIE = 1. 0 <sub>B</sub> No overflow has occurred since last flag reset. 1 <sub>B</sub> An overflow has occurred since last flag reset. CFCOV must be cleared by software.
<b>0</b>	5, [15:8]	r	<b>reserved;</b> returns '0' if read; should be written with '0';

<sup>1)</sup> For all bit timing analysis modes, the count value of NFCRx.CFC always displays the measured value minus 1.  
 Example: A CFC value of 34 in mode CFSEL = 000 indicates that 35 have been elapsed between the most recent 2 dominant edges on the receive input.

**NFCRxL (x = 0-5)**

**Node x Frame Counter Register Low(218H+x\*100<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
<b>CFC</b>	[15:0]	rwh	<b>CAN Frame Counter</b> In Frame Count Mode this bit field contains the frame count value. In TimeStamp Mode this bit field contains the captured bit time count value, captured with the start of a new frame.

### Bit Timings Analysis Modes and States

**Table 22-11 Bit Timing Analysis Modes (CFMOD = 10)**

CFSEL	Measurement
000	Whenever a dominant edge (transition from 1 to 0) is monitored on the receive input the time (measured in clock cycles) between this edge and the most recent dominant edge is stored in CFC.
001	Whenever a recessive edge (transition from 0 to 1) is monitored on the receive input the time (measured in clock cycles) between this edge and the most recent dominant edge is stored in CFC.
010	Whenever a dominant edge is received as a result of a transmitted dominant edge the time (clock cycles) between both edges is stored in CFC.
011	Whenever a recessive edge is received as a result of a transmitted recessive edge the time (clock cycles) between both edges is stored in CFC.
100	Whenever a dominant edge that qualifies for synchronization is monitored on the receive input the time (measured in clock cycles) between this edge and the most recent sample point is stored in CFC.
101	With each sample point, the time (measured in clock cycles) between the start of the new bit time and the start of the previous bit time is stored in CFC[11:0]. Additional information is written to CFC[15:12] at each sample point: CFC[15] : Transmit value of actual bit time CFC[14] : Receive sample value of actual bit time CFC[13:12] : CAN bus information (see <a href="#">Table 22-12</a> )
110	reserved
111	reserved

**Table 22-12 CAN Bus State Information**

<b>CFC[13:12]</b>	<b>CAN bus state</b>
00	<b>NoBit</b> The CAN bus is idle, performs bit (de-) stuffing or is in one of the following frame segments: SOF, SRR, CRC, delimiters, first 6 EOF bits, IFS
01	<b>NewBit</b> This code represents the first bit of a new frame segment. The current bit is the first bit in one of the following frame segments: bit 10 (MSB) of standard ID (transmit only), RTR, reserved bits, IDE, DLC(MSB), bit 7 (MSB) in each data byte and the first bit of the ID extension
10	<b>Bit</b> This code represents a bit inside a frame segment with a length of more than one bit (not the first bit of those frame segments which is indicated by NewBit). The current bit is processed within one of the following frame segments: ID bits (except first bit of standard ID for transmission and first bit of ID extension), DLC (3 LSB) and bits 6-0 in each data byte
11	<b>Done</b> The current bit is in one of the following frame segments: Acknowledge slot, last bit of EOF, active/passive error frame, overload frame. Two or more directly consecutive Done codes signal an error frame.

## 22.3.4 Message Object Registers

### 22.3.4.1 Message Object Control Register

The Message Object Control Register contains control bits for the CAN transfer and the message object link pointer. Each control bit has a corresponding bit in the CTRL field. A control bit is set by writing 1 to the corresponding bit in CTRL. It is cleared by writing 1 to the control bit directly. Any other combination leaves the control bit unchanged. After reset initialization the pointer PNEXT (read value of MOCTRnH[15:8]) points to message object n+1 (PNEXT = n+1), except for PNEXT of message object 255, which terminates the initial list (PNEXT = 255). Pointer PREV (read value of MOCTRnH[7:0]) initially points to message object n-1 (PPREV = n-1), except for PPREV of message object 0 which indicates the start of the initial list (PPREV = 0). This reset initialization means that all message objects initially belong to the list of unallocated elements.

#### MOCTR0H

**Message Object 0 Control Register High (101E<sub>H</sub>)**

**Reset Value: 0100<sub>H</sub>**

#### MOCTR255H

**Message Object 255 Control Register High (2FFE<sub>H</sub>)**

**Reset Value: FFFE<sub>H</sub>**

#### MOCTRnH (n = 1-254)

**Message Object n Control Register High (101E<sub>H</sub>+n\*20<sub>H</sub>)**

**Reset Value:**

**((n+1)\*0100<sub>H</sub>)+((n-1)\*0001<sub>H</sub>)**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				SET DIR	SET TXE N1	SET TXE N0	SET TXR Q	SET RXE N	SET RTS EL	SET MSG VAL	SET MSG LST	SET NEW DAT	SET RXU PD	SET TXP ND	SET RXP ND
W				W	W	W	W	W	W	W	W	W	W	W	W

Field	Bits	Type	Description
SETRXPND	0	w	<b>Set Receive Pending</b> This bit sets the RXPND
SETTXPND	1	w	<b>Set Transmit Pending</b> This bit sets the TXPND
SETRXUPD	2	w	<b>Set Receive Updating</b> This bit sets the RXUPD
SETNEWDAT	3	w	<b>Set New Data</b> This bit sets the NEWDAT



**Controller Area Network (MultiCAN) Controller**

Field	Bits	Type	Description
<b>SETMSGGLST</b>	4	w	<b>Set Message Lost</b> This bit sets the MSGGLST
<b>SETMSGVAL</b>	5	w	<b>Set Message Valid</b> This bit sets the MSGVAL
<b>SETRTSEL</b>	6	w	<b>Set Receive/Transmit Selected</b> This bit sets the RTSEL
<b>SETRXEN</b>	7	w	<b>Set Receive Enable</b> This bit sets the RXEN
<b>SETTXRQ</b>	8	w	<b>Set Transmit Request</b> This bit sets the TXRQ
<b>SETTXEN0</b>	9	w	<b>Set Transmit Enable 0</b> This bit sets the TXEN0
<b>SETTXEN1</b>	10	w	<b>Set Transmit Enable 1</b> This bit sets the TXEN1
<b>SETDIR</b>	11	w	<b>Set Message Direction</b> This bit sets the DIR
<b>0</b>	[15:12]	w	<b>Reserved</b> Should be written with 0.

**MOCTRnL (n = 0-255)**

**Message Object n Control Register Low(101C<sub>H</sub>+n\*20<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				<b>RES DIR</b>	<b>RES TXE N1</b>	<b>RES TXE N0</b>	<b>RES TXR Q</b>	<b>RES RXE N</b>	<b>RES RTS EL</b>	<b>RES MSG VAL</b>	<b>RES MSG LST</b>	<b>RES NEW DAT</b>	<b>RES RXU PD</b>	<b>RES TXP ND</b>	<b>RES RXP ND</b>
W				W	W	W	W	W	W	W	W	W	W	W	W

Field	Bits	Type	Description
<b>RESRXPND</b>	0	w	<b>Reset Receive Pending</b> This bit resets the RXPND
<b>RESTXPND</b>	1	w	<b>Reset Transmit Pending</b> This bit resets the TXPND
<b>RESRXUPD</b>	2	w	<b>Reset Receive Updating</b> This bit resets the RXUPD

**Controller Area Network (MultiCAN) Controller**

<b>Field</b>	<b>Bits</b>	<b>Type</b>	<b>Description</b>
<b>RESNEWDAT</b>	3	w	<b>Reset New Data</b> This bit resets the NEWDAT
<b>RESMSGLST</b>	4	w	<b>Reset Message Lost</b> This bit resets the MSGLST
<b>RESMSGVAL</b>	5	w	<b>Reset Message Valid</b> This bit resets the MSGVAL
<b>RESRTSEL</b>	6	w	<b>Reset Receive/Transmit Selected</b> This bit resets the RTSEL
<b>RESRXEN</b>	7	w	<b>Reset Receive Enable</b> This bit resets the RXEN
<b>RESTXRQ</b>	8	w	<b>Reset Transmit Request</b> This bit resets the TXRQ
<b>RESTXEN0</b>	9	w	<b>Reset Transmit Enable 0</b> This bit resets the TXEN0
<b>RESTXEN1</b>	10	w	<b>Reset Transmit Enable 1</b> This bit resets the TXEN1
<b>RESDIR</b>	11	w	<b>Reset Message Direction</b> This bit resets the DIR
<b>0</b>	[15:12]	w	<b>Reserved</b> Should be written with 0.

### 22.3.4.2 Message Object Status Register

#### **MOSTAT0H**

**Message Object 0 Status Register High (101E<sub>H</sub>)**

**Reset Value: 0100<sub>H</sub>**

#### **MOSTAT255H**

**Message Object 255 Status Register High (2FFE<sub>H</sub>)**

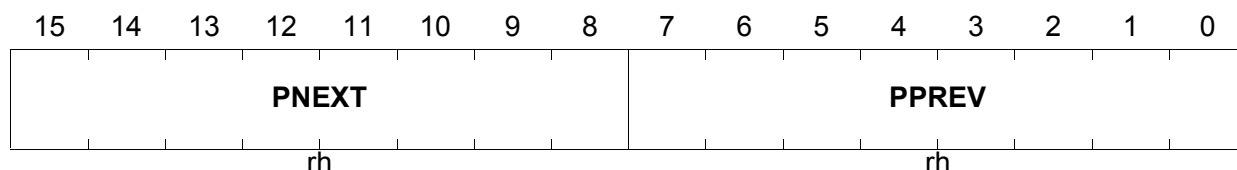
**Reset Value: FFFE<sub>H</sub>**

#### **MOSTATnH (n = 1-254)**

**Message Object n Status Register High (101E<sub>H</sub>+n\*20<sub>H</sub>)**

**Reset Value:**

**((n+1)\*0100<sub>H</sub>)+((n-1)\*0001<sub>H</sub>)**

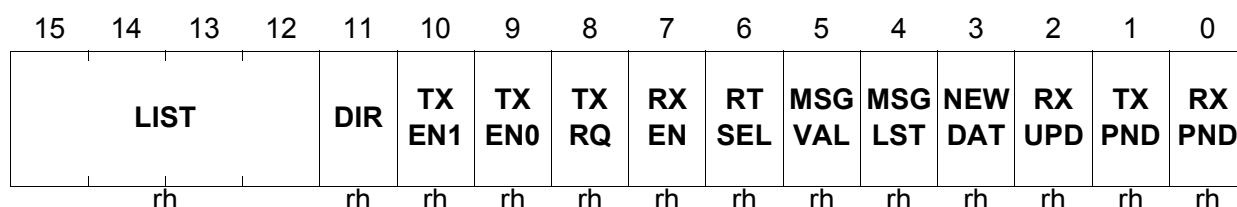


Field	Bits	Type	Description
<b>PNEXT</b>	[15:8]	rh	<b>Pointer to Previous Message Object</b> PPREV holds the message object number of the previous message object in a message list structure.
<b>PPREV</b>	[7:0]	rh	<b>Pointer to Next Message Object</b> PNEXT holds the message object number of the next message object in a message list structure.

#### **MOSTATnL (n = 0-255)**

**Message Object n Status Register Low(101C<sub>H</sub>+n\*20<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



**Controller Area Network (MultiCAN) Controller**

Field	Bits	Type	Description
<b>RXPND</b>	0	rh	<b>Receive Pending</b> 0 <sub>B</sub> No CAN message has been received. 1 <sub>B</sub> A CAN message has been received by the message object n, either directly or via gateway copy action. RXPND is not reset by hardware but must be reset by software.
<b>TXPND</b>	1	rh	<b>Transmit Pending</b> 0 <sub>B</sub> No CAN message has been transmitted. 1 <sub>B</sub> A CAN message from message object n has been transmitted successfully over the CAN bus. TXPND is reset by hardware but must be reset by software.
<b>RXUPD</b>	2	rh	<b>Receive Updating</b> 0 <sub>B</sub> No receive update ongoing. 1 <sub>B</sub> Message identifier, DLC, and data of the message object are currently updated.
<b>NEWDAT</b>	3	rh	<b>New Data</b> 0 <sub>B</sub> No update of the message object n since last flag reset. 1 <sub>B</sub> Message object n has been updated. NEWDAT is set by hardware after a received CAN frame has been stored in message object n. NEWDAT is cleared by hardware when a CAN transmission of message object n has been started. NEWDAT should be set by software after the new transmit data has been stored in message object n to prevent the automatic reset of TXRQ at the end of an ongoing transmission.
<b>MSGLST</b>	4	rh	<b>Message Lost</b> 0 <sub>B</sub> No CAN message is lost. 1 <sub>B</sub> A CAN message is lost because NEWDAT has become set again when it has already been set.

**Controller Area Network (MultiCAN) Controller**

Field	Bits	Type	Description
<b>MSGVAL</b>	5	rh	<b>Message Valid</b> 0 <sub>B</sub> Message object n is not valid. 1 <sub>B</sub> Message object n is valid. Only a valid message object takes part in CAN transfers.
<b>RTSEL</b>	6	rh	<b>Receive/Transmit Selected</b> 0 <sub>B</sub> Message object n is not selected for receive or transmit operation. 1 <sub>B</sub> Message object n is selected for receive or transmit operation. <b>Frame Reception:</b> RTSEL is set by hardware when message object n has been identified for storage of a CAN frame that is currently received. Before a received frame becomes finally stored in message object n, a check is performed to determine if RTSEL is set. Thus the CPU can suppress a scheduled frame delivery to this message object n by clearing RTSEL by software. <b>Frame Transmission:</b> RTSEL is set by hardware when message object n has been identified to be transmitted next. A check is performed to determine if RTSEL is still set before message object n is actually set up for transmission and bit NEWDAT is cleared. It is also checked that RTSEL is still set before its message object n is verified due to the successful transmission of a frame. RTSEL needs to be checked only when the context of message object n changes, and a conflict with an ongoing frame transfer shall be avoided. In all other cases, RTSEL can be ignored. RTSEL has no impact on message acceptance filtering. RTSEL is not cleared by hardware.
<b>RXEN</b>	7	rh	<b>Receive Enable</b> 0 <sub>B</sub> Message object n is not enabled for frame reception. 1 <sub>B</sub> Message object n is enabled for frame reception. RXEN is evaluated for receive acceptance filtering only.

**Controller Area Network (MultiCAN) Controller**

Field	Bits	Type	Description
<b>TXRQ</b>	8	rh	<p><b>Transmit Request</b></p> <p>0<sub>B</sub> No transmission of message object n is requested.</p> <p>1<sub>B</sub> Transmission of message object n on the CAN bus is requested.</p> <p>The transmit request becomes valid only if TXRQ, TXEN0, TXEN1 and MSGVAL are set. TXRQ is set by hardware if a matching Remote Frame has been received correctly. TXRQ is reset by hardware if message object n has been transmitted successfully and NEWDAT is not set again by software.</p>
<b>TXEN0</b>	9	rh	<p><b>Transmit Enable 0</b></p> <p>0<sub>B</sub> Message object n is not enabled for frame transmission.</p> <p>1<sub>B</sub> Message object n is enabled for frame transmission.</p> <p>Message object n can be transmitted only if both bits, TXEN0 and TXEN1, are set.</p> <p>The user may clear TXEN0 in order to inhibit the transmission of a message that is currently updated, or to disable automatic response of Remote Frames.</p>
<b>TXEN1</b>	10	rh	<p><b>Transmit Enable 1</b></p> <p>0<sub>B</sub> Message object n is not enabled for frame transmission.</p> <p>1<sub>B</sub> Message object n is enabled for frame transmission.</p> <p>Message object n can be transmitted only if both bits, TXEN0 and TXEN1, are set.</p> <p>TXEN1 is used by the MultiCAN module for selecting the active message object in the Transmit FIFOs.</p>

**Controller Area Network (MultiCAN) Controller**

Field	Bits	Type	Description
<b>DIR</b>	11	rh	<b>Message Direction</b> $0_B$ Receive Object selected: With TXRQ = 1, a Remote Frame with the identifier of message object n is scheduled for transmission. On reception of a Data Frame with matching identifier, the message is stored in message object n. $1_B$ Transmit Object selected: If TXRQ = 1, message object n is scheduled for transmission of a Data Frame. On reception of a Remote Frame with matching identifier, bit TXRQ is set.
<b>LIST</b>	[15:12]	rh	<b>List Allocation</b> LIST indicates the number of the message list to which message object n is allocated. LIST is updated by hardware when the list allocation of the object is modified by a panel command.

**Controller Area Network (MultiCAN) Controller**

### 22.3.4.3 Message Object Interrupt Pointer Register

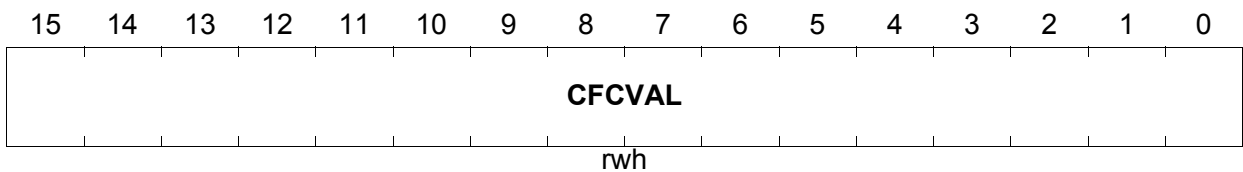
The Message Object Interrupt Pointer Registers MOIPR H/L hold various pointers related to message interrupts as well as the frame counter value.

**MOIPRnH (n = 0-255)**

**Message Object n Interrupt Pointer Register High**

**(100A<sub>H</sub>+n\*20<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



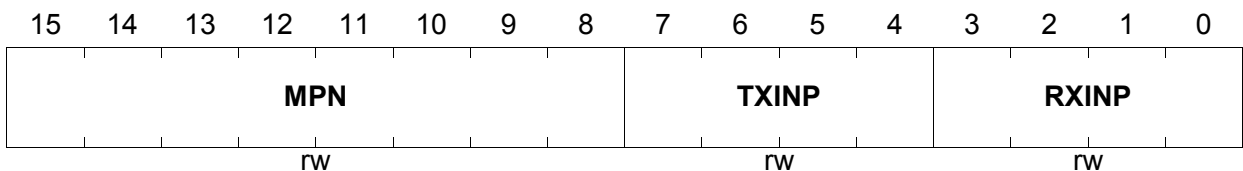
Field	Bits	Type	Description
CFCVAL	[15:0]	rwh	<b>CAN Frame Counter Value</b> When a message is stored in this message object or this message object has been successfully transmitted then the CAN frame counter value CFC of the CAN Node Frame Counter Register (NFCR) is copied to CFCVAL.

**MOIPRnL (n = 0-255)**

**Message Object n Interrupt Pointer Register Low**

**(1008<sub>H</sub>+n\*20<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
RXINP	[3:0]	rw	<b>Receive Interrupt Node Pointer</b> Select the interrupt output line INT_Om (m=0-15) for receive interrupts.
TXINP	[7:4]	rw	<b>Transmit Interrupt Node Pointer</b> Select the interrupt output line INT_Om (m=0-15) for transmit interrupts.



**Controller Area Network (MultiCAN) Controller**

Field	Bits	Type	Description
MPN	[15:8]	rw	<b>Message Pending Number</b> This field selects the bit position of the bit in the message pending register to be set upon a receive/transmit interrupt.

### 22.3.4.4 Message Object Function Control Register

The Message Object Function Control Registers High / Low contain bits to select and to configure the function of the message object. It also holds the CAN data length code.

**MOFCRnH (n = 0-255)**

**Message Object n Function Control Register High**

**(1002<sub>H</sub>+n\*20<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>OMSC</b>				<b>DLC</b>				<b>STT</b>	<b>SDT</b>	<b>RMM</b>	<b>FRR EN</b>	<b>0</b>	<b>OVIE</b>	<b>TXIE</b>	<b>RXIE</b>
rwh				rwh				rw	rw	rw	rw	r	rw	rw	rw

Field	Bits	Type	Description
<b>RXIE</b>	0	rw	<b>Receive Interrupt Enable</b> If RXIE is set then a message interrupt request is generated with the reception of a CAN message, no matter whether the CAN message is received directly or indirectly via a gateway action. The interrupt is requested on interrupt output line as defined by RXINP.
<b>TXIE</b>	1	rw	<b>Transmit Interrupt Enable</b> If TXIE is set then a message interrupt request is generated when this message object successfully transmitted a message over the CAN bus. The interrupt is requested on interrupt output line as defined by TXINP.

**Controller Area Network (MultiCAN) Controller**

Field	Bits	Type	Description
<b>OVIE</b>	2	rw	<b>Overflow Interrupt Enable</b> IF OVIE = 1 then a FIFO full interrupt is generated when the pointer to the current object CUR reaches the value of SEL in the FIFO/Gateway Pointer Register. If this object is a receive FIFO base object then the FIFO full interrupt is requested on interrupt output line as defined by TXINP. If this object is a transmit FIFO base object then the FIFO full interrupt is requested on interrupt output line as defined by RXINP. For all other message object modes OVIE has no effect.
<b>FRREN</b>	4	rw	<b>Foreign Remote Request Enable</b> Specifies if the TXRQ bit is set in this message object or in a foreign object referenced by the pointer CUR. 0 <sub>B</sub> TXRQ of this message object is set upon the reception of a matching remote frame. 1 <sub>B</sub> TXRQ of the message object referenced by the pointer CUR is set upon the reception of a matching remote frame.
<b>RMM</b>	5	rw	<b>Transmit Object Remote Monitoring</b> 0 <sub>B</sub> Remote monitoring disabled: The identifier, IDE bit and DLC of the message object remain unchanged upon the reception of a matching remote frame. 1 <sub>B</sub> Remote monitoring enabled: The identifier, DLC and IDE bit of a matching remote frame are copied to this transmit object in order to monitor incoming remote frames. Bit RMM only applies to transmit objects and has no impact on receive objects.

**Controller Area Network (MultiCAN) Controller**

Field	Bits	Type	Description
<b>SDT</b>	6	rw	<b>Single Data Transfer</b> If SDT = 1 and this object is not a FIFO base object then MSGVAL is reset when this object has taken part in a successful data transfer (receive or transmit). If SDT = 1 and this object is a FIFO base object then MSGVAL is reset when the pointer to the current object CUR reaches the value of SEL in the FIFO/Gateway Pointer Register. With SDT = 0, bit MSGVAL is not affected.
<b>STT</b>	7	rw	<b>Single Transmit Trial</b> If this bit is set then TXRQ is cleared upon transmission start of this message object. Thus no transmission retry is performed in case of transmission failure.
<b>DLC</b>	[11:8]	rwh	<b>Data Length Code</b> Valid values for the data length are 0 to 8. DLC>8 leads to 8 data bytes, but the DLC code is not truncated upon reception or transmission of CAN frames.
<b>0</b>	[15:12]	r	<b>Reserved;</b> read as 0; should be written with 0.
<b>0</b>	3	r	<b>Reserved;</b> read as 0; should be written with 0.

**MOFCRnL (n = 0-255)**

**Message Object n Function Control Register Low**

**(1000<sub>H</sub>+n\*20<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				DAT C	DLC C	IDC	GDF S	0				MMC			
r				rw	rw	rw	rw	r				rw			

**Controller Area Network (MultiCAN) Controller**

Field	Bits	Type	Description
<b>MMC</b>	[3:0]	rw	<b>Message Mode Control</b> Bit field MMC controls the functionality of the message object. 0000 <sub>B</sub> Standard Message Object 0001 <sub>B</sub> Receive FIFO Base Object 0010 <sub>B</sub> Transmit FIFO Base Object 0011 <sub>B</sub> Transmit FIFO Slave Object 0100 <sub>B</sub> Gateway Source Object else <sub>B</sub> Reserved
<b>GDFS</b>	8	rw	<b>Gateway Data Frame Send</b> 1 <sub>B</sub> TXRQ is set in the gateway destination object after the transfer of a data frame from the gateway source to the gateway destination. 0 <sub>B</sub> TXRQ is not set in the destination object. Applicable only to Gateway Source Object.
<b>IDC</b>	9	rw	<b>Identifier Copy</b> IF IDC = 1 then the identifier of the gateway source object (after storing the received frame in the source) is copied to the gateway destination. Applicable only to Gateway Source Object.
<b>DLCC</b>	10	rw	<b>Data Length Code Copy</b> If DLCC = 1 then the data length code of the gateway source object (after storing the received frame in the source) is copied to the gateway destination. Applicable only to Gateway Source Object.
<b>DATC</b>	11	rw	<b>Data Copy</b> If DATC = 1 then the data field (registers MODATA0 and MODATA4) of the gateway source object (after storing the received frame in the source) is copied to the gateway destination. Applicable only to Gateway Source Object.
<b>0</b>	[7:4], [15:12]	r	<b>reserved;</b> returns '0' if read; should be written with '0';

**Controller Area Network (MultiCAN) Controller**

### 22.3.4.5 Message Object FIFO/Gateway Pointer Register

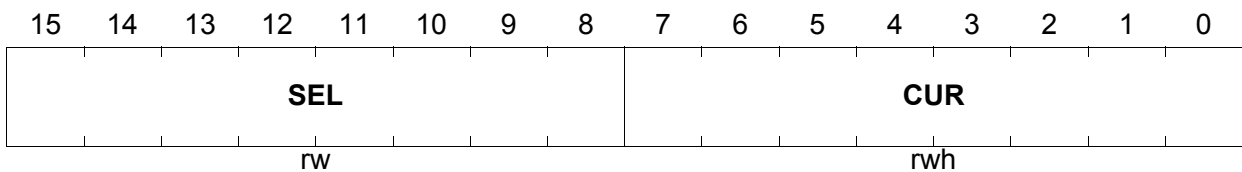
The Message Object FIFO/Gateway Pointer Registers H/L contain a set of message object link pointer used for FIFO and gateway functionality

**MOFGPRnH (n = 0-255)**

**Message Object n FIFO/Gateway Pointer Register High**

**(1006<sub>H</sub>+n\*20<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



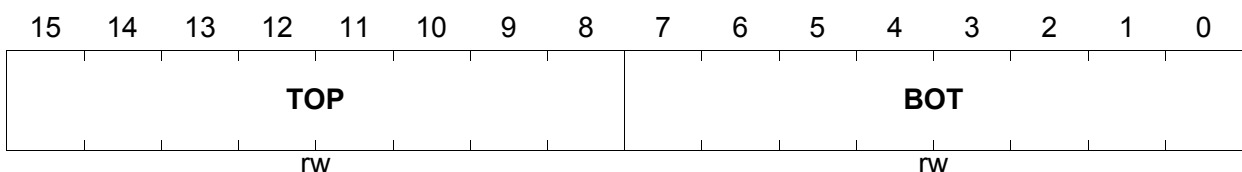
Field	Bits	Type	Description
<b>CUR</b>	[7:0]	rwh	<b>Current Object Pointer</b> The Current Object Pointer links to the actual target object within a FIFO/Gateway structure. After a FIFO/gateway operation CUR is updated with the message number of the next message object in the list structure (given by PNEXT of the message control register) until it reaches the FIFO top element (given by TOP) when it is reset to the bottom element (given by BOT).
<b>SEL</b>	[15:8]	rw	<b>Object Select Pointer</b> The Object Select Pointer is the second (software) pointer to complement the hardware pointer CUR in the FIFO structure. SEL is used for monitoring purposes only.

**MOFGPRnL (n = 0-255)**

**Message Object n FIFO/Gateway Pointer Register Low**

**(1004<sub>H</sub>+n\*20<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
<b>BOT</b>	[7:0]	rw	<b>Bottom Pointer</b> The Bottom Pointer points to the first element in a FIFO structure.
<b>TOP</b>	[15:8]	rw	<b>Top Pointer</b> The TOP pointer points to the last element in a FIFO structure.

*Note: The pointers in this register must be set to objects assigned to the same CAN node. It is forbidden to refer to objects that are not in the linked list for the same CAN node.*

**Controller Area Network (MultiCAN) Controller**

### 22.3.4.6 Message Object Acceptance Mask Register

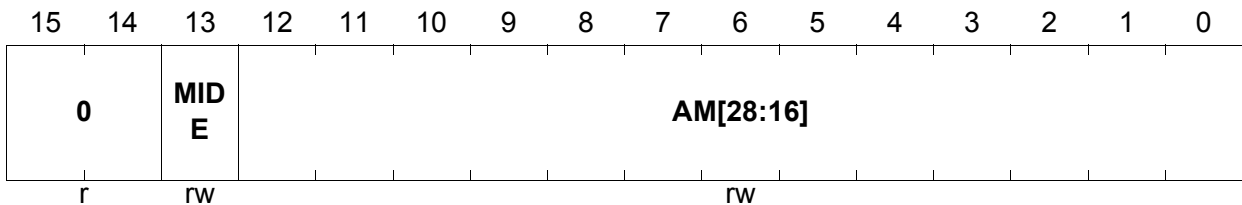
Registers MOAMR H/L contain the mask bits for the acceptance filtering of the message object.

**MOAMRnH (n = 0-255)**

**Message Object n Acceptance Mask Register High**

**(100E<sub>H</sub>+n\*20<sub>H</sub>)**

**Reset Value: 3FFF<sub>H</sub>**



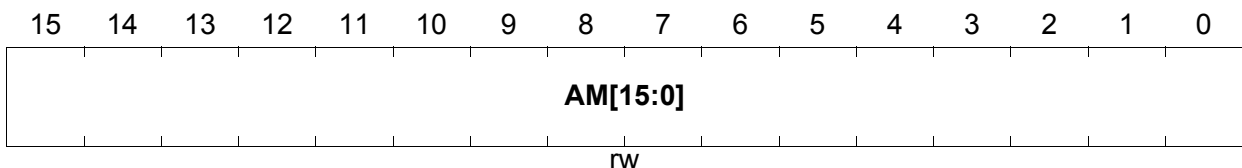
Field	Bits	Type	Description
<b>AM[28:16]</b>	[12:0]	rw	<b>Acceptance Mask for Message Identifier High</b> see description of MOAMRnL.AM[15:0]
<b>MIDE</b>	13	rw	<b>Acceptance Mask bit for Message IDE bit</b> <div> <div>0<sub>B</sub></div> <div>This message objects accepts the reception of both standard and extended frames.</div> <div>1<sub>B</sub></div> <div>This message object only receives frames with matching IDE bit.</div> </div>
<b>0</b>	[15:14]	r	<b>Reserved;</b> read as 0; should be written with 0.

**MOAMRnL (n = 0-255)**

**Message Object n Acceptance Mask Register Low**

**(100C<sub>H</sub>+n\*20<sub>H</sub>)**

**Reset Value: FFFF<sub>H</sub>**





Field	Bits	Type	Description
<b>AM[15:0]</b>	[15:0]	rw	<b>Acceptance Mask for Message Identifier</b> Mask to filter incoming messages with standard identifiers (AM[28:18]) or extended identifiers (AM[28:0]). For standard identifiers bits AM[17:0] are "don't care".

### 22.3.4.7 Message Object Arbitration Register

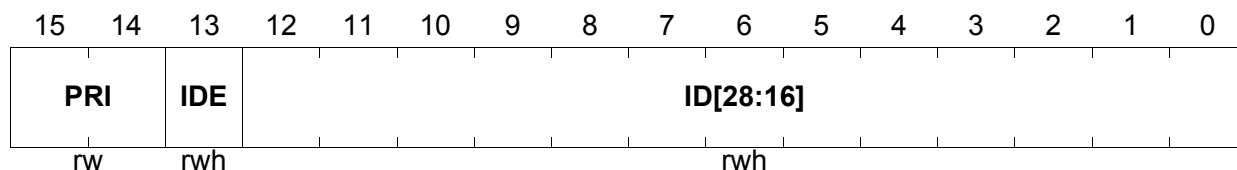
Registers MOAR H/L contain the CAN identifier of the message object.

**MOARnH (n = 0-255)**

**Message Object n Arbitration Register High**

**(101A<sub>H</sub>+n\*20<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
<b>ID[28:16]</b>	[12:0]	rwh	<b>CAN Identifier of Message Object</b> Identifier of a standard message (ID[28:18]) or an extended message (ID[28:0]). For standard identifiers bits ID[17:0] are “don’t care”.
<b>IDE</b>	13	rwh	<b>CAN IDE bit of Message Object</b> 0 <sub>B</sub> Standard frame with 11-bit identifier 1 <sub>B</sub> Extended frame with 29-bit identifier

**Controller Area Network (MultiCAN) Controller**

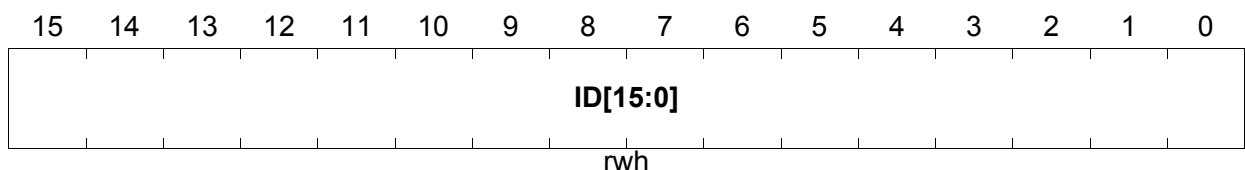
Field	Bits	Type	Description
PRI	[15:14]	rw	<b>Priority Class</b> PRI assigns one of the four priority classes 0, 1, 2, 3 to the message object, with lower PRI number meaning higher priority. Message objects with lower PRI value always win acceptance filtering for frame reception and transmission over message objects with higher PRI value. Acceptance filtering based on identifier/mask and list position is only performed between message objects of the same priority class. PRI also defines the acceptance filtering method for transmission: 00 <sub>B</sub> Reserved. Transmit objects with PRI = 00 are not taken into account for the transmit acceptance filtering. 01 <sub>B</sub> Transmit acceptance filtering is based on the list order, i.e. this message object is considered for transmission only if there is no other message object with valid transmit request (MSGVAL & TXRQ & TXEN0 & TXEN1 = 1) somewhere before this object in the list. 10 <sub>B</sub> Transmit acceptance filtering is based on the CAN identifier, i.e. this message object is considered for transmission only if there is no other message object with higher priority identifier+IDE+DIR (with respect to CAN arbitration rules) somewhere in the list. 11 <sub>B</sub> Transmit acceptance filtering is based on the list order (like PRI = 01).

**MOARnL (n = 0-255)**

**Message Object n Arbitration Register Low**

**(1018<sub>H</sub>+n\*20<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
<b>ID[15:0]</b>	[15:0]	rwh	<b>CAN Identifier of Message Object Low</b> Identifier of a standard message (ID[28:18]) or an extended message (ID[28:0]). For standard identifiers bits ID[17:0] are “don’t care”.

### Transmit Priority

**Table 22-13 Transmit Priority based on CAN Arbitration Rules**

Settings of arbitrarily chosen message objects A and B, where A has higher transmit priority than B	Comment
A.MOAR[28:18] < B.MOAR[28:18] (11 bit standard identifier of A less than 11 bit standard identifier of B)	Messages with lower standard identifier have higher priority than messages with higher standard identifier. MOAR[28] is the most significant bit (MSB) of the standard identifier. MOAR[18] is the least significant bit of the standard identifier.
A.MOAR[28:18] = B.MOAR[28:18] A.MOAR.IDE = 0 (send standard frame) B.MOAR.IDE = 1 (send extended frame)	Standard frames have higher transmit priority than extended frames with equal standard identifier.
A.MOAR[28:18] = B.MOAR[28:18] A.MOAR.IDE = B.MOAR.IDE = 0 A.MOCTR.DIR = 1 (send data frame) B.MOCTR.DIR = 0 (send remote frame)	Standard data frames have higher transmit priority than standard remote frames with equal identifier.
A.MOAR[28:0] = B.MOAR[28:0] A.MOAR.IDE = B.MOAR.IDE = 1 A.MOCTR.DIR = 1 (send data frame) B.MOCTR.DIR = 0 (send remote frame)	Extended data frames have higher transmit priority than extended remote frames with equal identifier.
A.MOAR[28:0] < B.MOAR[28:0] A.MOAR.IDE = B.MOAR.IDE = 1 (29 bit identifier)	Extended frames with lower identifier have higher transmit priority than extended frames with higher identifier. MOAR[28] is the most significant bit (MSB) of the overall identifier (standard identifier MOAR[28:18] and identifier extension MOAR[17:0]). MOAR[0] is the least significant bit (LSB) of the overall identifier.

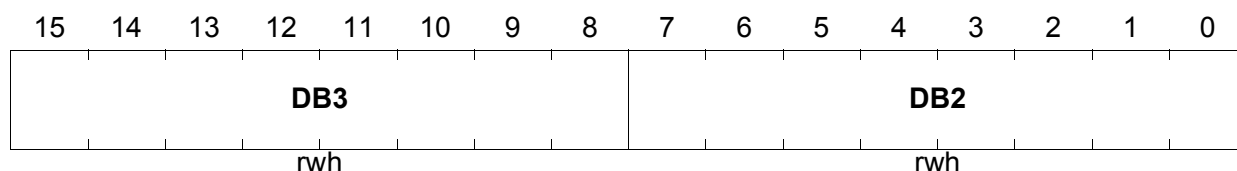
### 22.3.4.8 Message Object Data Registers

**MODATANLH (n = 0-255)**

**Message Object n Data Register Low High**

**(1012<sub>H</sub>+n\*20<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



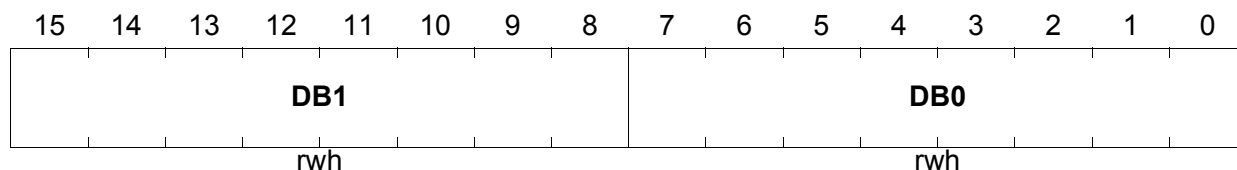
Field	Bits	Type	Description
<b>DB2</b>	[7:0]	rwh	<b>CAN Data Byte 2</b>
<b>DB3</b>	[15:8]	rwh	<b>CAN Data Byte 3</b>

**MODATANLL (n = 0-255)**

**Message Object n Data Register Low Low**

**(1010<sub>H</sub>+n\*20<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
<b>DB0</b>	[7:0]	rwh	<b>CAN Data Byte 0</b>
<b>DB1</b>	[15:8]	rwh	<b>CAN Data Byte 1</b>

## Controller Area Network (MultiCAN) Controller

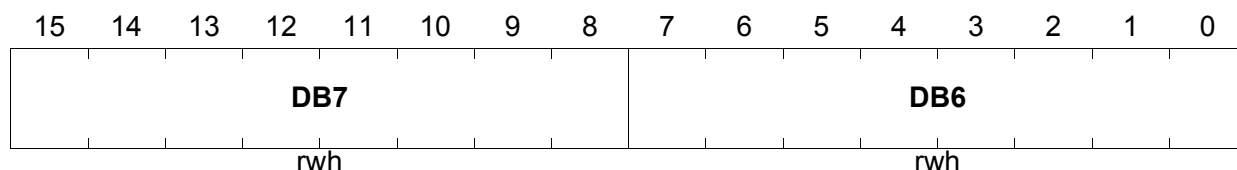
Registers MODATAH H/L contain the highest four CAN data bytes. Unused data bytes are padded zero upon reception and ignored for transmission .

### MODATANHH (n = 0-255)

#### Message Object n Data Register High High

$(1016_H + n * 20_H)$

Reset Value: 0000<sub>H</sub>



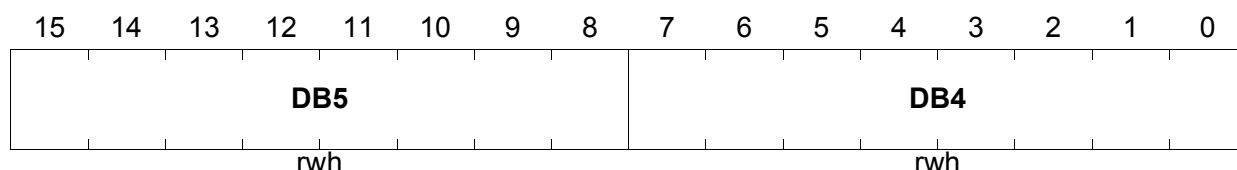
Field	Bits	Type	Description
DB6	[7:0]	rwh	CAN Data Byte 6
DB7	[15:8]	rwh	CAN Data Byte 7

### MODATANHL (n = 0-255)

#### Message Object n Data Register High Low

$(1014_H + n * 20_H)$

Reset Value: 0000<sub>H</sub>



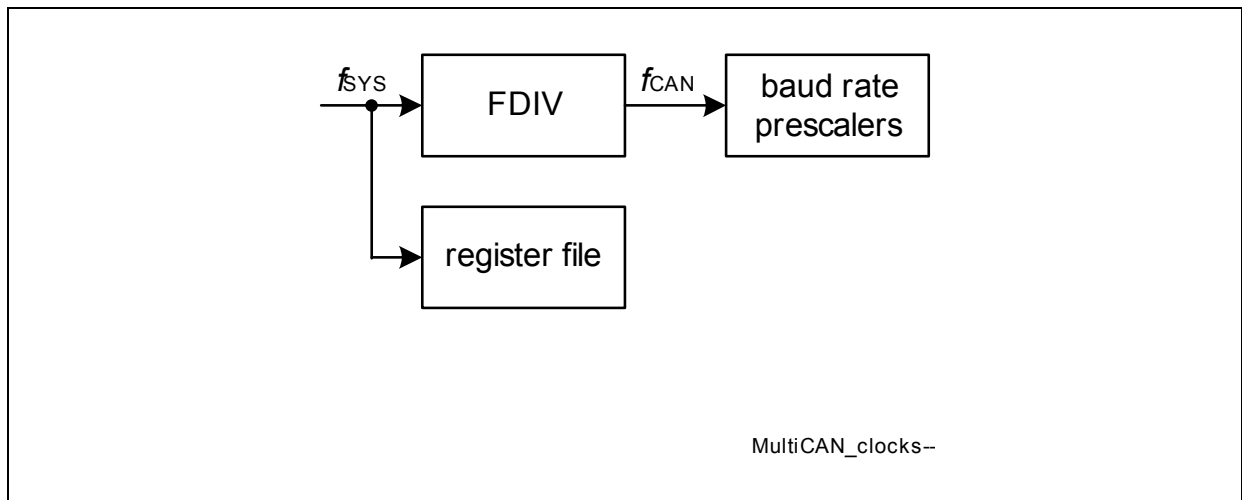
Field	Bits	Type	Description
DB4	[7:0]	rwh	CAN Data Byte 4
DB5	[15:8]	rwh	CAN Data Byte 5

## 22.4 General Control and Status

The following section describes the general clock, debug and interrupt topics.

### 22.4.1 Clock Control

The CAN clock frequency  $f_{CAN}$  of the functional blocks of the MultiCAN module is derived from the system clock  $f_{SYS}$  (= clock on the system bus). The fractional divider FDIV in the module is used to generate the CAN clock frequency for the bit timing calculation. This frequency is identical for all CAN nodes. The scheduler itself is in the  $f_{SYS}$  domain. The clock generation can be enabled/disabled by the fractional divider control bit field FDR.DM.



**Figure 22-17 MultiCAN Clock Generation**

The fractional divider FDIV output  $f_{CAN}$  is based on the system clock  $f_{SYS}$ , but only every n-th clock pulse is taken. The register file is in the system frequency domain. The suspend signal (coming as acknowledge from the module as answer to the OCDS suspend request) freezes or resets the fractional divider.

*Note: The receive input line contains a synchronization stage to ensure stable input data. Together with the internal CAN state machine, this leads to a minimum reaction time of at least 3 clock cycles of  $f_{SYS}$  between CAN input and output. The switching delay of the input stages can be generally neglected, whereas the rise/fall times of the port output drivers (programmable values) should be taken into account, especially for higher baud rates.*

The table below indicates the minimum frequencies of  $f_{SYS}$  in MHz for MultiCAN module operation (acceptance filtering, MO handling, etc.), that are required for a baud rate of 1 Mbit/s for the active CAN nodes (the highest CAN baud rate of the activated CAN nodes has to be taken into account). If less baud rate is desired, the values can be scaled linearly (e.g. for a maximum of 500 kbit/s, 50% of the indicated value are required).

## Controller Area Network (MultiCAN) Controller

The values imply that the CPU (or PEC) executes a maximum of accesses to the MultiCAN module. The values may contain rounding effects.

**Table 22-14 Minimum Operating Frequencies [MHz]**

<b>Number of allocated message objects MO<sup>1)</sup></b>	<b>1 CAN node active</b>	<b>2 CAN nodes active</b>	<b>3 CAN nodes active</b>	<b>4 CAN nodes active</b>	<b>5 CAN nodes active</b>	<b>6 CAN nodes active</b>
<b>16 MO</b>	12	19	26	33	40	47
<b>32 MO</b>	15	23	30	37	44	52
<b>64 MO</b>	21	28	37	46	53	61
<b>128 MO</b>	40	45	50	55	61	70
<b>144 MO</b>	42	47	52	57	62	70
<b>160 MO</b>	46	51	56	61	66	72
<b>176 MO</b>	50	55	60	66	71	76
<b>192 MO</b>	54	59	65	70	75	80
<b>208 MO</b>	58	64	69	74	79	84
<b>224 MO</b>	63	68	73	78	83	89
<b>240 MO</b>	67	72	77	82	88	93
<b>256 MO</b>	71	76	81	87	92	97

<sup>1)</sup> Only those message objects have to be taken into account that are allocated to a CAN node. The unallocated message objects have no influence on the minimum operating frequency.

The baud rate generation of the MultiCAN being based on  $f_{SYS}$ , this frequency has to be chosen carefully to allow correct CAN bit timing. The required value of  $f_{SYS}$  is given by an integer multiple (n) of the CAN baud rate multiplied by the number of time quanta per CAN bit time. For example, to reach 1 Mbit/s with 20 tq per bit time, possible values of  $f_{SYS}$  are given by formula  $[n \times 20]$  MHz, with n being an integer value, starting at 1. In order to minimize jitter, it is not recommended to use the fractional divider mode for high baud rates.

### 22.4.2 Port Input Control

For each CAN node there is possibility to select which pin will be used as RXDCAN input. The selected pin (one out of several possible) is connected to the CAN node.



### **22.4.3 Suspend Mode**

The suspend mode can be triggered by the OCDS in order to freeze the state of the module and to have access to the registers (at least for read actions). There are several aspects related to the suspend mode:

- All actions are immediately stopped ("hard suspend"):  
The module clock is switched off as soon as the suspend line becomes active. This mode is supported by the fast switch off feature of the BPI. Write actions to the module are not supported and only combinatorial read actions deliver the desired data (the CAN RAM and the CAN registers can not be accessed).  
In this mode, all further module actions are disabled and there is a very high probability that the communication with other devices is made impossible and that the CAN bus is blocked by the device in hard suspend mode (e.g. if the suspended CAN just sends a dominant level). A normal continuation when the suspend mode is left is not always possible and reset must be activated.
- The current action is finished ("soft suspend"):  
The module functions are stopped (clock is still running!) automatically after internal actions have been finished, for example after a CAN frame has been sent out. Due to this behavior, the communication network is not blocked due to the suspend mode of one communication partner. Furthermore, all registers are accessible for read and write actions. As a result, the debugger can stop the module actions and modify registers. These modifications are taken into account after the suspend mode is left.  
This mode is designed to be able to modify registers or to read them by the OCDS while the rest of the systems is still running and not corrupted by the suspend mode.

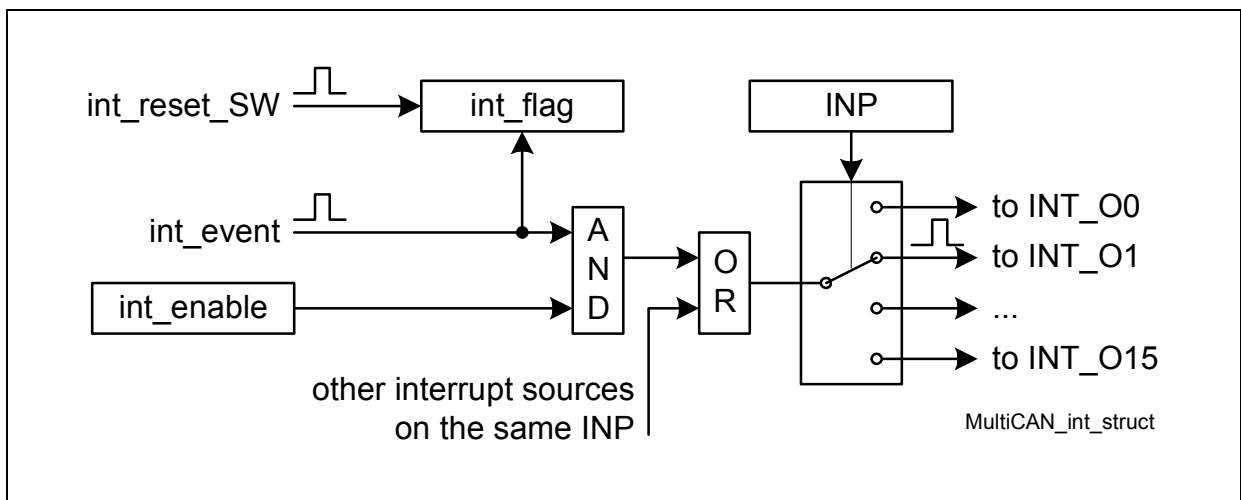
In the MultiCAN module a suspend mechanism is implemented allowing the individual freeze of CAN nodes. The fast switch off feature (hard suspend) of the BPI must not be activated by the user in order to support the soft suspend mode. In order to allow the required flexibility for the system, each CAN node can be individually enabled for the soft suspend mode.

The hard suspend feature can be enabled/disabled for the complete MultiCAN module, whereas the soft suspend feature can be enabled/disabled independently for each CAN node. The fractional divider disables the CAN clock only if all CAN nodes signal that they can be suspended. A CAN node that is not active can always be suspended.

#### 22.4.4 Interrupt Structure

The general interrupt structure is shown in the figure below. The interrupt event can trigger the interrupt generation. The interrupt pulse is generated independently from the interrupt flag in the interrupt status register. The interrupt flag can be reset by SW by writing a 0 to it.

If enabled by the related interrupt enable bit in the interrupt enable register, an interrupt pulse can be generated at one of the 16 interrupt output lines INT\_Ox of the module. If more than one interrupt source is connected to the same interrupt node pointer (in the interrupt node pointer register), the requests are combined to one common line.



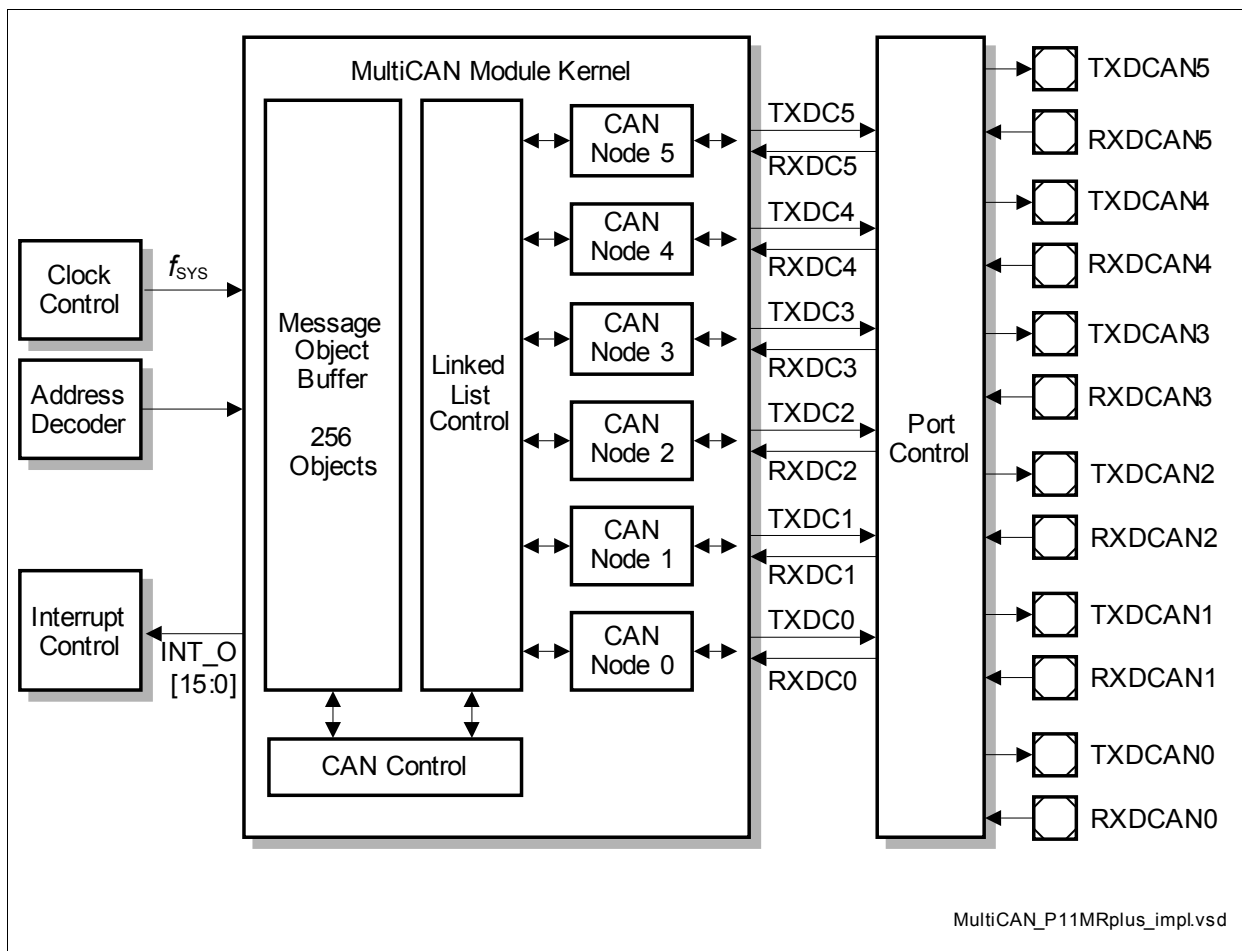
**Figure 22-18 General Interrupt Structure**

## 22.5 MultiCAN Module Implementation

This section describes CAN module interfaces with the clock control, port connections, interrupt control, and address decoding.

### 22.5.1 Interfaces of the CAN Module

**Figure 22-19** shows the XE16xyM specific implementation details and interconnections of the CAN module. The I/O lines of the CAN module kernel (two I/O lines of each CAN node) are connected to the ports as described in **Table 22-18**. The CAN module is further supplied by clock control, interrupt control, and address decoding logic.



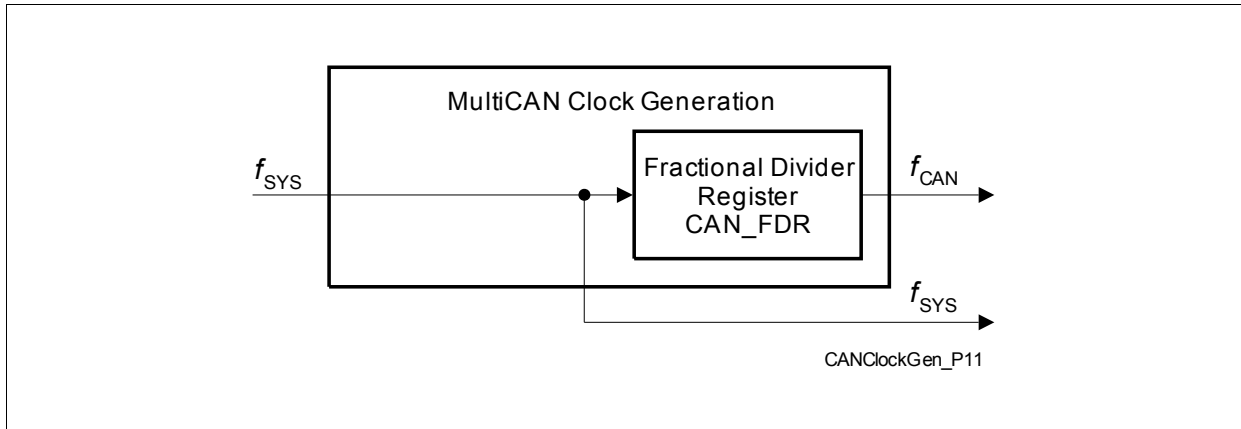
**Figure 22-19 CAN Module Implementation and Interconnections**

The MultiCAN interrupt control register x is connected to the CAN interrupt output line INT\_Ox, with x = 15 - 0. Additionally, the signal INT\_O15 can be used to start timers.

## 22.5.2 Module Clock Generation

As shown in **Figure 22-20**, the clock signals for the MultiCAN module are generated and controlled by a clock generation unit. This clock generation unit is responsible for the enable/disable control, the clock frequency adjustment, and the debug clock control.

The frequency control of the module timer clock  $f_{CAN}$  is performed via the CAN\_FDR register.



**Figure 22-20 MultiCAN Module Clock Generation**

The module control clock  $f_{SYS}$  is used inside the MultiCAN module kernel for control purposes such as e.g. for clocking of control logic and register operations. The frequency of  $f_{SYS}$  is identical to the system clock frequency  $f_{SYS}$ .

The module timer clock  $f_{CAN}$  is used inside the MultiCAN module kernels as input clock for all timing relevant operations.

The frequency of  $f_{CAN}$  is defined by:

$$f_{CAN} = f_{SYS} \times \frac{1}{n} \quad \text{with } n = 1024 - \text{CAN\_FDR.STEP}$$

$$\text{or } f_{CAN} = f_{SYS} \times \frac{n}{1024} \quad \text{with } n = 0-1023$$

*Note: The upper formula applies to normal divider mode of the fractional divider (CAN\_FDR.DM = 01<sub>B</sub>). The lower formula applies to fractional divider mode (CAN\_FDR.DM = 10<sub>B</sub>).*

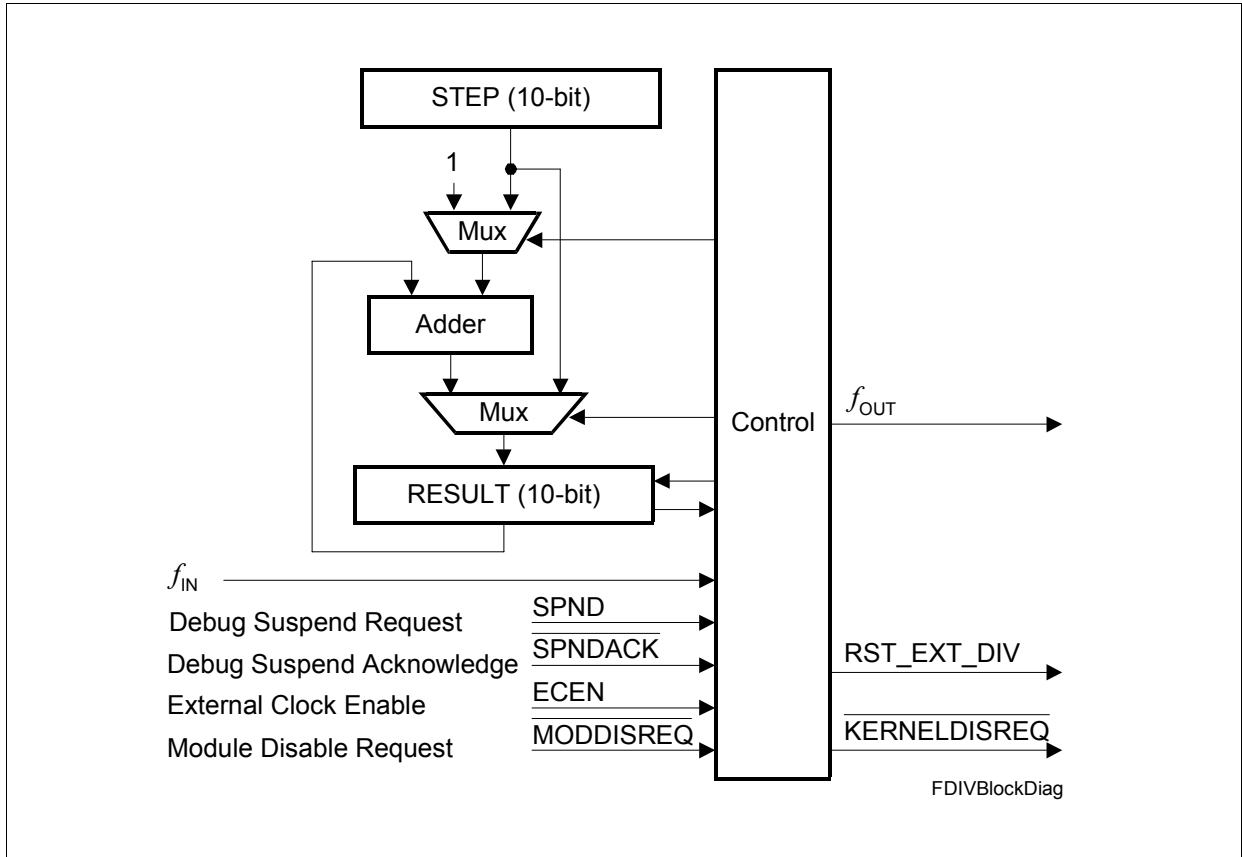
*Note: Input signal ECEN of the MultiCAN fractional divider is wired to 0.*

### 22.5.2.1 Fractional Divider Overview

The fractional divider allows to generate output clocks from an input clock using a programmable divider. The fractional divider divides an input clock  $f_{IN}$  either by the factor

**Controller Area Network (MultiCAN) Controller**

1/n or by a fraction of n/1024 for any value of n from 0 to 1023 and outputs the clock signals,  $f_{OUT}$ . The clock generation can be enabled/disabled by the fractional divider register control bit field FDR.DM.



**Figure 22-21 Fractional Divider Block Diagram**

The clock generation in the fractional divider is further controlled by four input signals.

**Table 22-15 Fractional Divider I/O Lines**

Signal	I/O	Description
SPND	Input	Suspend Request Input is controlled by the debug system suspend request signal. It becomes active when a general suspend request is issued from the debug system to the on-chip modules.
SPNDACK		Suspend Acknowledge Input is driven with the disable acknowledge signal from the module kernel. This signal is activated by the module kernel as a response to a suspend request that has been issued by the fractional divider via $\overline{\text{KERNELDISREQ}} = 0$ .
MODDISREQ		Module Disable Request Input is connected to the disable request output from the CLC logic. An active signal at this input results in the activation of output signal $\overline{\text{KERNELDISREQ}}$ .
ECEN		External Clock Enable Signal ECEN can be used to synchronize the fractional divider clock generation to external events.
KERNELDISREQ	Output	Kernel Disable Request This output signal becomes active when either $\overline{\text{MODDISREQ}}$ is activated or when SPND becomes active.
RST_EXT_DIV		Reset External Divider This output signal allows to control (stop/reset) external divider stages for $f_{\text{OUT}}$ .
$f_{\text{OUT}}$		Module Clock Enable Signal $f_{\text{OUT}}$ is the enable signal for the module clock. The module clock itself is built by and-ing the $f_{\text{OUT}}$ enable signal with $f_{\text{IN}}$ . Module clock frequency references mostly refer to the AND combination of $f_{\text{OUT}}$ with $f_{\text{IN}}$ .

The fractional divider has two operating modes:

- Normal divider mode
- Fractional divider mode

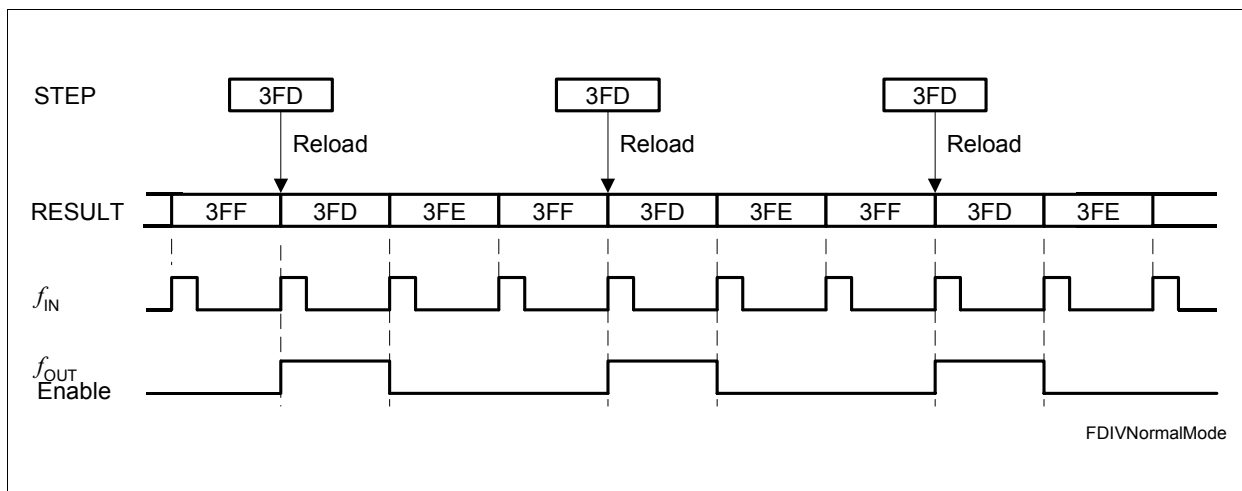
### Normal Divider Mode

In normal divider mode ( $FDR.DM = 01_B$ ) the fractional divider behaves like a reload counter (addition of +1) that generates an output clock pulse at  $f_{OUT}$  on the transition from  $3FF_H$  to  $000_H$ .  $FDR.RESULT$  represents the counter value and  $FDR.STEP$  defines the reload value.

The output frequencies in normal divider mode are defined according the following formulas:

$$f_{OUT} = f_{IN} \times \frac{1}{n} \quad \text{with } n = 1024 - STEP$$

In order to get  $f_{OUT} = f_{IN}$  STEP must be programmed with  $3FF_H$ . **Figure 22-22** shows the operation of the normal divider mode with a reload value of  $FDR.STEP = 3FD_H$ . The clock frequency of  $f_{OUT}$  is represented by and-ing the  $f_{OUT}$  enable signal with  $f_{IN}$ .



**Figure 22-22 Normal Mode Timing**

### Fractional Divider Mode

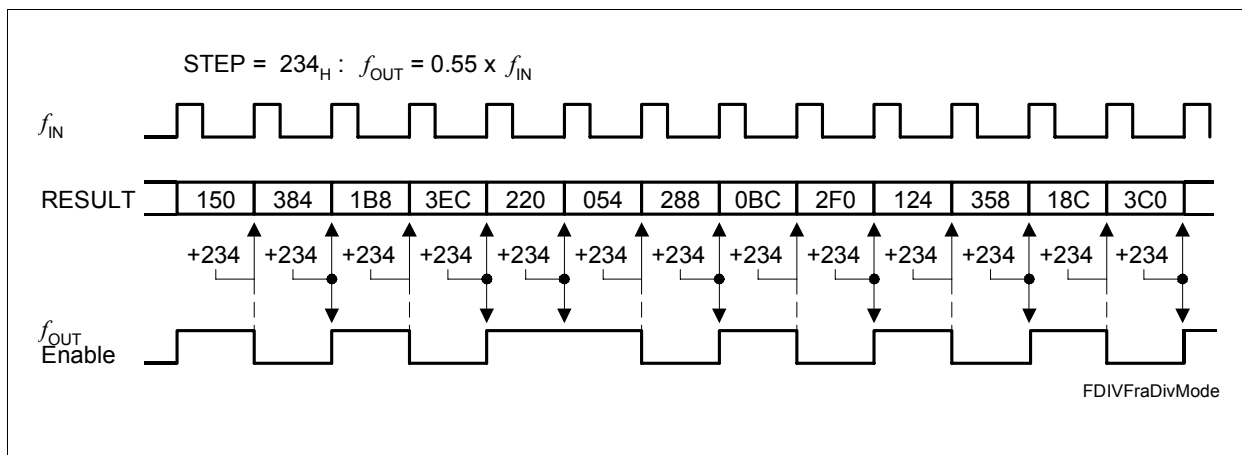
When the fractional divider mode is selected ( $FDR.DM = 10_B$ ), the output clock  $f_{OUT}$  is derived from the input clock  $f_{IN}$  by division of a fraction of  $n/1024$  for any value of  $n$  from 0 to 1023. In general, the fractional divider mode allows to program the average output clock frequency with a higher accuracy than in normal divider mode.

In fractional divider mode an output clock pulse at  $f_{OUT}$  is generated dependent on the result of the addition  $FDR.RESULT + FDR.STEP$ . If the addition leads to an overflow over  $3FF_H$  a pulse is generated at  $f_{OUT}$ . Note that in fractional divider mode the clock  $f_{OUT}$  can have a maximum period jitter of one  $f_{IN}$  clock period.

The output frequencies in fractional divider mode are defined according the following formulas:

$$f_{OUT} = f_{IN} \times \frac{n}{1024} \quad \text{with } n = 0-1023$$

**Figure 22-23** shows the operation of the fractional divider mode with a reload value of  $FDR.STEP = 234_H$  (= factor  $564/1024 = 0.55$ ). The clock frequency of  $f_{OUT}$  is represented by and-ing the  $f_{OUT}$  enable signal with  $f_{IN}$ .



**Figure 22-23 Fractional Divider Mode Timing**

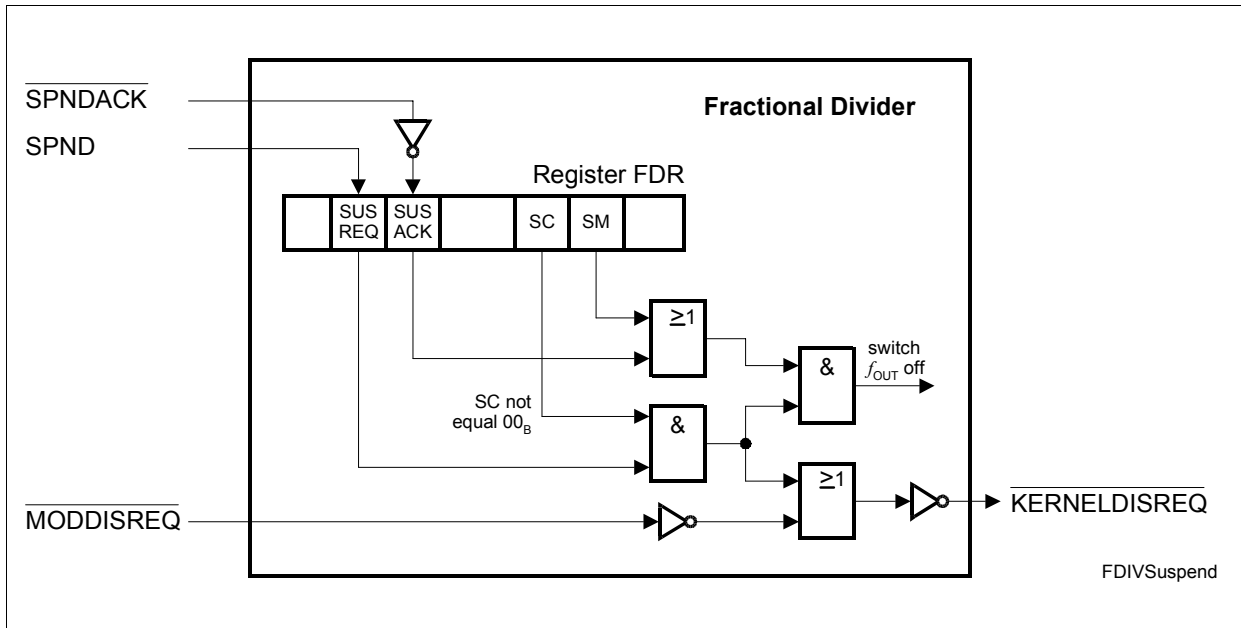
### Suspend Mode Control

The fractional divider allows to control its operation according to the input Suspend Request (SPND). This input is activated in suspend mode by the on-chip debug control logic. In suspend mode, the module registers are accessible for read and write actions, but the other module internal functions are frozen. Suspend mode is requested by  $SPND = 1$ . Suspend mode is entered one  $f_{IN}$  clock cycle after the suspend mode request has been acknowledged by setting  $SPNDACK$  to 0 (granted suspend mode) and



## Controller Area Network (MultiCAN) Controller

FDR.SC is not equal  $00_B$  (clock output signal disabled). Suspend mode is immediately entered when bit SM is set to 1 and FDR.SC is not equal  $00_B$  (immediate suspend mode). The state of signals SPND and  $\overline{\text{SPNDACK}}$  is latched in two status flags of register FDR, SUSREQ and SUSACK. SPND and ( $\overline{\text{SPNDACK}}$  or bit SM) must remain set both to maintain the suspend mode.



**Figure 22-24 Suspend Mode Configuration**

The Kernel Disable Request signal  $\overline{\text{KERNELDISREQ}}$  becomes always active when  $\overline{\text{MODDISREQ}}$  is activated, independently of the suspend mode settings in the fractional divider logic.

### External Clock Enable

When the module clock generation has been disabled by software (setting  $\text{FDR.DISCLK} = 1$ ), the disable state can be left via input  $\text{ECEN} = 1$  (hardware controlled). This feature is enabled when  $\text{FDR.ENHW} = 1$ . In the MultiCAN module, signal ECEN is tied to 0.

### Registers Overview

#### Fractional Divider Registers

The fractional divider contains two registers, FDRL (lower 16 bits) and FDRH (higher 16 bits).

**Controller Area Network (MultiCAN) Controller**

**FDRL**

**Fractional Divider Register L**

**(0C<sub>H</sub>)**

**Reset Value: 0000H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>DM</b>		<b>SC</b>		<b>SM</b>	<b>0</b>	<b>STEP</b>									
rw		rw		rw	r	rw									

Field	Bits	Type	Description
<b>STEP</b>	[9:0]	rw	<b>Step Value</b> In normal divider mode STEP contains the reload value for RESULT. In fractional divider mode this bit field defines the 10-bit value that is added to the RESULT with each input clock cycle.
<b>SM</b>	11	rw	<b>Suspend Mode</b> SM selects between granted or immediate suspend mode. 0     Granted suspend mode selected 1     Immediate suspend mode selected
<b>SC</b>	[13:12]	rw	<b>Suspend Control</b> This bit field defines the behavior of the fractional divider in suspend mode (bit SUSREQ and SUSACK set). 00    Clock generation continues. 01    Clock generation is stopped and the clock output signals are not generated. RESULT is not changed except when writing bit field DM with 01B or 10B. 10    Clock generation is stopped and the clock output signals are not generated. RESULT is loaded with 3FF <sub>H</sub> . 11    Same as SC = 10B but RST_EXT_DIV is 1 (independently of bit field DM).

**Controller Area Network (MultiCAN) Controller**

Field	Bits	Type	Description
<b>DM</b>	[15:14]	rw	<b>Divider Mode</b> This bit fields defines the functionality of the fractional divider block. 00 Fractional divider is switched off; no output clock is generated. RST_EXT_DIV is 1. RESULT is not updated (default after reset). 01 Normal divider mode selected. 10 Fractional divider mode selected. 11 Fractional divider is switched off; no output clock is generated. RESULT is not updated.
<b>0</b>	others	r	<b>Reserved</b> read as 0; should be written with 0.

**FDRH**

**Fractional Divider Register H**

**(0E<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>DIS CLK</b>	<b>EN HW</b>	<b>SUS REQ</b>	<b>SUS ACK</b>	<b>0</b>		<b>RESULT</b>									
rw	rw	rh	rh	r		rh									

Field	Bits	Type	Description
<b>RESULT</b>	[9:0]	rh	<b>Result Value</b> In normal divider mode RESULT acts as reload counter (addition +1). In fractional divider mode this bit field contains the result of the addition RESULT+STEP. If DM is written with 01 <sub>B</sub> or 10 <sub>B</sub> , RESULT is loaded with 3FF <sub>H</sub> .
<b>SUSACK</b>	12	rh	<b>Suspend Mode Acknowledge</b> 0 Suspend mode is not acknowledged. 1 Suspend mode is acknowledged. Suspend mode is entered when SUSACK and SUSREQ are set.

**Controller Area Network (MultiCAN) Controller**

Field	Bits	Type	Description
<b>SUSREQ</b>	13	rh	<b>Suspend Mode Request</b> 0 Suspend mode is not requested. 1 Suspend mode is requested. Suspend mode is entered when SUSREQ and SUSACK are set.
<b>ENHW</b>	14	rw	<b>Enable Hardware Clock Control</b> 0 Bit DISCLK cannot be reset by HW by a high level at input signal ECEN. 1 Bit DISCLK is reset by hardware while input signal ECEN is at high level.
<b>DISCLK</b>	15	rwh	<b>Disable Clock</b> 0 Clock generation of $f_{OUT}$ is enabled according to the setting of bit field DM. 1 Fractional divider is stopped. The enable signal $f_{OUT}$ becomes inactive. No change except when writing bit field DM.
<b>0</b>	others	r	<b>Reserved</b> read as 0; should be written with 0.

### Fractional Divider Operation Modes

**Table 22-16 Fractional Divider Function Table**

Mode	SC	DM	RES_EXT_DIV	RESULT	$f_{OUT}$	Operation of Fractional Divider
Normal Mode	—	00	1	unchanged	inactive	switched off
		01	0	continuously updated <sup>1)</sup>	active	normal divider mode
		10				fractional divider mode
		11		unchanged	inactive	switched off

**Controller Area Network (MultiCAN) Controller**

**Table 22-16 Fractional Divider Function Table**

Mode	SC	DM	RES_EXT_DIV	RESULT	$f_{OUT}$	Operation of Fractional Divider
Suspend Mode	00	00	1	unchanged	inactive	switched off
		01	0	continuously updated <sup>1)</sup>	active	normal divider mode
		10				fractional divider mode
		11			unchanged	inactive
	01	00	1	unchanged	inactive	switched off
		01	0	unchanged <sup>1)</sup>		halted
		10				
		11				unchanged
	10	00	1	loaded with 3FF <sub>H</sub>	inactive	switched off
		01	0			halted
		10				
		11				
	11	—	1	loaded with 3FF <sub>H</sub>	inactive	switched off

<sup>1)</sup> Each write operation to FDR with DM = 01<sub>B</sub> or 10<sub>B</sub> sets RESULT to 3FF<sub>H</sub>.

### 22.5.3 Mode Control Behavior

The MultiCAN module provides two mechanisms to stop participation in CAN traffic:

- **Suspend Mode:**  
The suspend mode request is issued by the OCDS block. The sensitivity of a CAN node to a suspend request can be individually enabled/disabled for each CAN node. In suspend mode, a CAN node correctly finishes a running CAN frame, but does not start a new one.
- **Immediate Stop Mode:**  
The immediate stop mode is entered when a stop mode is requested by the mode control of the device, configured by CAN\_KSCCFG. If an immediate stop is requested, the CAN module immediately stops all CAN activity (even within a running frame) and sets all transmit outputs to 1. In order to allow CAN operation, bit field NOMCFG has to be set to a run mode. To support suspend mode (see description above), bit field SUMCFG has to be set to run mode to avoid immediate stop mode.

#### **22.5.4 Mode Control**

The mode control concept for system control tasks, such as power saving, or suspend request for debugging, allows to program the module behavior under different device operating conditions. The behavior of the MultiCAN kernel can be programmed for each of the device operating modes, that are requested by the global state control part of the SCU. MultiCAN has an associated register **CAN\_KSCCFG** defining the behavior of the kernel of the module in the following device operating modes:

- **Normal operation:**  
This operating mode is the default operating mode when neither a suspend request nor a clock-off request are pending. The module clock is not switched off and the MultiCAN registers can be read or written. The kernel behavior is defined by KSCCFG.NOMCFG.
- **OCDS suspend mode:**  
This operating mode is requested when a suspend request (issued by a debugger) is pending in the device. The module clock is not switched off and the MultiCAN registers can be read or written. The KSCCFG.SUMCFG = 00, and the OCDS registers are properly configured.
- **Clock-off mode:**  
This operating mode is requested for power saving purposes. The module clock is switched off .

For the MultiCAN module, the following internal actions can be influenced by mode control:

- A current transmission of a CAN message:  
If there is a pending request, it can be started. This start has to be enabled by the mode control. If the current kernel mode allows the start (run modes 0 and 1), it will be executed. If the kernel mode does not allow a start (stop modes 0 and 1), the request is not started. The start request is not cancelled, but frozen. A “frozen” request is started as programmed if the kernel mode is changed to a run mode again.

The behavior of the MultiCAN kernel can be programmed for each of the device operating modes (normal operation, suspend mode, clock-off mode), as shown in **Table 22-17**.

**Controller Area Network (MultiCAN) Controller**

**Table 22-17 MultiCAN Kernel Behavior**

<b>Kernel Mode</b>	<b>Kernel Behavior</b>	<b>Code</b>
run mode 0	kernel operation as specified, no impact on data transfer (same behavior for run mode 0 and run mode 1)	00 <sub>B</sub>
run mode 1		01 <sub>B</sub>
stop mode 0	The module is stopped after finishing some internal actions which may take several clock cycles. Pending CAN transfers are not completed. The device is driving recessive level on the external bus. No read / write access to the registers is possible.	10 <sub>B</sub>
stop mode 1		11 <sub>B</sub>

Generally, bit field KSCCFG.NOMCFG should be configured for run mode 0 as default setting for standard operation. If the MultiCAN kernel should not react to a suspend request (and to continue operation as in normal mode), bit field KSCCFG.SUMCFG has to be configured with the same value as KSCCFG.NOMCFG. If the MultiCAN kernel 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 has to be written to KSCCFG.SUMCFG.

A similar mechanism applies for the clock-off mode with the possibility to program the desired behavior by bit field KSCCFG.COMCFG.

*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 KSCCFG. Do not mix stop mode 0 and stop mode 1 and avoid transitions from stop mode 0 to stop mode 1 (or vice versa) for the MultiCAN module.*

Please note that bit KSCCFG.MODEN should only be set by SW while all configuration fields are configured for run mode 0.

## 22.5.5 Mode Control Register Description

### 22.5.5.1 Kernel State Configuration Register

The kernel state configuration register KSCCFG allows the selection of the desired kernel modes for the different device operating modes.

The bit fields KSCCFG.NOMCFG and KSCCFG.COMCFG are reset by an application reset, whereas the bit field KSCCFG.SUMCFG is reset by a debug reset.

*Note: The coding of the bit fields NOMCFG, SUMCFG and COMCFG is described in [Table 22-17](#).*

#### CAN\_KSCCFG

##### Kernel State Configuration Register

SFR(FE1E <sub>H</sub> )											Reset Value: 0000 <sub>H</sub>				
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BP COM	0	COMCFG		BP SUM	0	SUMCFG		BP NOM	0	NOMCFG		0		BP MOD EN	MOD EN
w	r	rw		w	r	rw		w	r	rw		r		w	rw

Field	Bits	Type	Description
<b>MODEN</b>	0	rw	<p><b>Module Enable</b></p> <p>This bit enables the module kernel clock and the module functionality.</p> <p>0<sub>B</sub> 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 KSCCFG).</p> <p>1<sub>B</sub> The module is switched on and can operate. After writing 1 to MODEN, it is recommended to read register KSCCFG to avoid pipeline effects in the control block before accessing other MultiCAN registers.</p> <p><i>Note: This bit is reset by an application reset.</i></p>



**Controller Area Network (MultiCAN) Controller**

Field	Bits	Type	Description
<b>BPMODEN</b>	1	w	<b>Bit Protection for MODEN</b> This bit enables the write access to the bit MODEN. It always reads 0. 0 <sub>B</sub> MODEN is not changed. 1 <sub>B</sub> MODEN is updated with the written value.
<b>NOMCFG</b>	[5:4]	rw	<b>Normal Operation Mode Configuration</b> This bit field defines the kernel mode applied in normal operation mode. 0X <sub>B</sub> The module is switched on. 1X <sub>B</sub> The module is switched off. This field is taken into account for CR = 00 or 11. <i>Note: This bit is reset by an application reset.</i>
<b>BPNUM</b>	7	w	<b>Bit Protection for NOMCFG</b> This bit enables the write access to the bit field NOMCFG. It always reads 0. 0 <sub>B</sub> NOMCFG is not changed. 1 <sub>B</sub> NOMCFG is updated with the written value.
<b>SUMCFG</b>	[9:8]	rw	<b>Suspend Mode Configuration</b> This bit field defines the kernel mode applied in suspend mode. 0X <sub>B</sub> The module is switched on. This is the recommended setting in order to have soft suspend behavior. The kernel is suspended by the OCDS module. 1X <sub>B</sub> The module is switched off. This field is taken into account for CR = 01. <i>Note: This bit is reset by a debug reset.</i>
<b>BPSUM</b>	11	w	<b>Bit Protection for SUMCFG</b> This bit enables the write access to the bit field SUMCFG. It always reads 0. 0 <sub>B</sub> SUMCFG is not changed. 1 <sub>B</sub> SUMCFG is updated with the written value.

**Controller Area Network (MultiCAN) Controller**

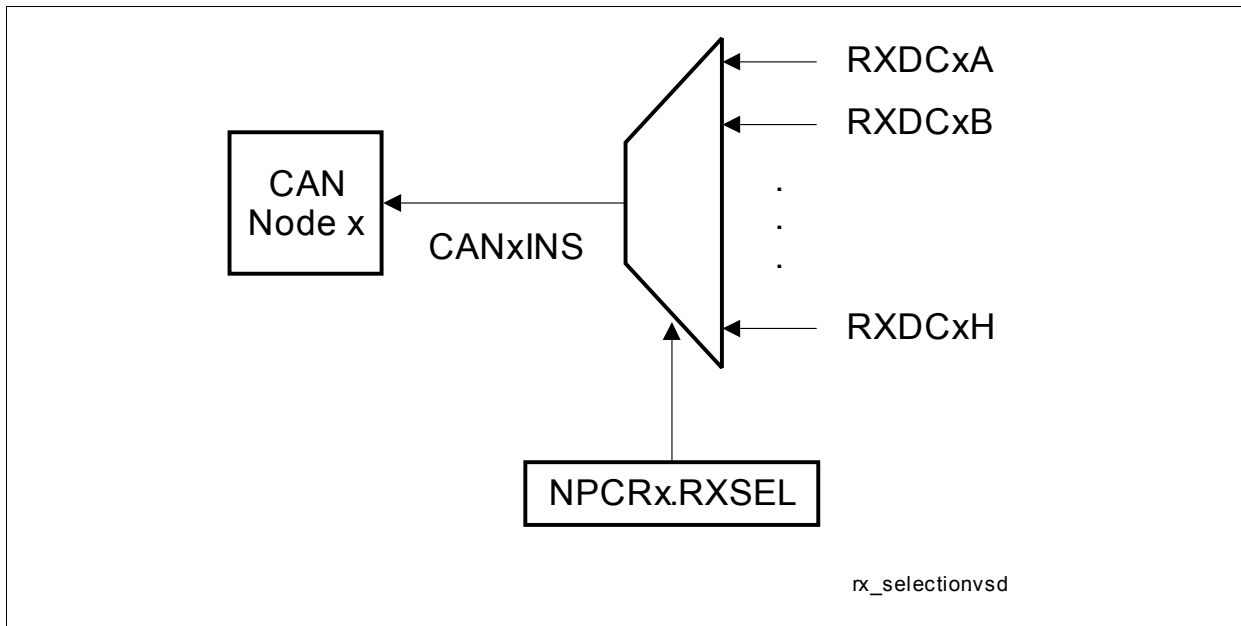
Field	Bits	Type	Description
<b>COMCFG</b>	[13:12]	rw	<b>Clock Off Mode Configuration</b> This bit field defines the kernel mode applied in clock off mode. 0X <sub>B</sub> The module is switched on. 1X <sub>B</sub> The module is switched off. This field is taken into account for CR = 10. <i>Note: This bit is reset by an application reset.</i>
<b>BPCOM</b>	15	w	<b>Bit Protection for COMCFG</b> This bit enables the write access to the bit field COMCFG. It always reads 0. 0 <sub>B</sub> COMCFG is not changed. 1 <sub>B</sub> COMCFG is updated with the written value.
<b>0</b>	[3:2], 6, 10, 14	r	<b>Reserved</b> returns 0 if read; should be written with 0;

*Note: The bit protection bits BPxxx allow partly modification of the configuration bits with a single write operation (without the need of a read-modify-write mechanism handled by the CPU).*

### 22.5.6 Connection of External Signals

The following table shows the digital connections of the MultiCAN signals with other modules or pins in the XE16xyM device.

The selected input signal (selected by bit field NPCRx.RXSEL) for each CAN node is made available by internal signal CANxINS (CAN node x input signal, with  $x = 4 - 0$ ).



**Figure 22-25 CAN Module Receive Input Selection**

**Controller Area Network (MultiCAN) Controller**

**Table 22-18 MultiCAN Connections in XE16xyM**

Signal	from/to Module	I/O to CAN	Can be used to/as
MultiCAN Node 0 Signals			
RXDC0A	P2.3	I	receive input A (NPCR0.RXSEL = 000 <sub>B</sub> )
RXDC0B	P0.3		receive input B (NPCR0.RXSEL = 001 <sub>B</sub> )
RXDC0C	P2.0		receive input C (NPCR0.RXSEL = 010 <sub>B</sub> )
RXDC0D	P2.6		receive input D (NPCR0.RXSEL = 011 <sub>B</sub> )
RXDC0E	ESR1		receive input E (NPCR0.RXSEL = 100 <sub>B</sub> )
RXDC0F	P11.0		receive input F (NPCR0.RXSEL = 101 <sub>B</sub> )
RXDC0G	1		receive input G (NPCR0.RXSEL = 110 <sub>B</sub> )
RXDC0H	0		receive input H (NPCR0.RXSEL = 111 <sub>B</sub> )
TXDC0	P0.1	O	transmit output
	P0.2		
	P2.1		
	P2.4		
	P2.5		
	P11.1		
MultiCAN Node 1 Signals			
RXDC1A	P2.4	I	receive input A (NPCR1.RXSEL = 000 <sub>B</sub> )
RXDC1B	P0.4		receive input B (NPCR1.RXSEL = 001 <sub>B</sub> )
RXDC1C	P2.7		receive input C (NPCR1.RXSEL = 010 <sub>B</sub> )
RXDC1D	CAN0INS		receive input D (NPCR1.RXSEL = 011 <sub>B</sub> )
RXDC1E	ESR2		receive input E (NPCR1.RXSEL = 100 <sub>B</sub> )
RXDC1F	P8.1		receive input F (NPCR1.RXSEL = 101 <sub>B</sub> )
RXDC1G	1		receive input G (NPCR1.RXSEL = 110 <sub>B</sub> )
RXDC1H	0		receive input H (NPCR1.RXSEL = 111 <sub>B</sub> )
CAN1INS	U1C1_DX0F	O	
TXDC1	P0.6	O	transmit output
	P2.2		
	P2.9		
	P8.2		

**Controller Area Network (MultiCAN) Controller**

**Table 22-18 MultiCAN Connections in XE16xyM (cont'd)**

Signal	from/to Module	I/O to CAN	Can be used to/as
MultiCAN Node 2 Signals			
RXDC2A	P4.3	I	receive input A (NPCR2.RXSEL = 000 <sub>B</sub> )
RXDC2B	P10.11		receive input B (NPCR2.RXSEL = 001 <sub>B</sub> )
RXDC2C	CAN1INS		receive input C (NPCR2.RXSEL = 010 <sub>B</sub> )
RXDC2D	P2.13		receive input D (NPCR2.RXSEL = 011 <sub>B</sub> )
RXDC2E	P6.1		receive input E (NPCR2.RXSEL = 100 <sub>B</sub> )
RXDC2F	P5.15		receive input F (NPCR2.RXSEL = 101 <sub>B</sub> )
RXDC2G	1		receive input F (NPCR2.RXSEL = 110 <sub>B</sub> )
RXDC2H	0		receive input H (NPCR2.RXSEL = 111 <sub>B</sub> )
TXDC2	P4.1	O	transmit output
	P4.2		
	P10.12		
	P6.0		
	P2.12		
MultiCAN Node 3 Signals			
RXDC3A	P3.3	I	receive input A (NPCR3.RXSEL = 000 <sub>B</sub> )
RXDC3B	P3.0		receive input B (NPCR3.RXSEL = 001 <sub>B</sub> )
RXDC3C	P10.14		receive input C (NPCR3.RXSEL = 010 <sub>B</sub> )
RXDC3D	CAN2INS		receive input D (NPCR3.RXSEL = 011 <sub>B</sub> )
RXDC3E	P0.5		receive input E (NPCR3.RXSEL = 100 <sub>B</sub> )
RXDC3F	-		receive input F (NPCR3.RXSEL = 101 <sub>B</sub> )
RXDC3G	1		receive input G (NPCR3.RXSEL = 110 <sub>B</sub> )
RXDC3H	0		receive input H (NPCR3.RXSEL = 111 <sub>B</sub> )
TXDC3	P3.1	O	transmit output
	P3.2		
	P10.13		
	P0.7		

**Controller Area Network (MultiCAN) Controller**

**Table 22-18 MultiCAN Connections in XE16xyM (cont'd)**

Signal	from/to Module	I/O to CAN	Can be used to/as
MultiCAN Node 4 Signals			
RXDC4A	P3.4	I	receive input A (NPCR4.RXSEL = 000 <sub>B</sub> )
RXDC4B	P7.0		receive input B (NPCR4.RXSEL = 001 <sub>B</sub> )
RXDC4C	P10.7		receive input C (NPCR4.RXSEL = 010 <sub>B</sub> )
RXDC4D	CAN3INS		receive input D (NPCR4.RXSEL = 011 <sub>B</sub> )
RXDC4E	P1.7		receive input E (NPCR4.RXSEL = 100 <sub>B</sub> )
RXDC4F	-		receive input F (NPCR4.RXSEL = 101 <sub>B</sub> )
RXDC4G	1		receive input G (NPCR4.RXSEL = 110 <sub>B</sub> )
RXDC4H	0		receive input H (NPCR4.RXSEL = 111 <sub>B</sub> )
TXDC4	P3.6	O	transmit output
	P7.1		
	P7.2		
	P10.6		
MultiCAN Node 5 Signals			
RXDC5A	P1.4	I	receive input A (NPCR4.RXSEL = 000 <sub>B</sub> )
RXDC5B	P11.4		receive input B (NPCR4.RXSEL = 001 <sub>B</sub> )
RXDC5C	P2.1		receive input C (NPCR4.RXSEL = 010 <sub>B</sub> )
RXDC5D	CAN4INS		receive input D (NPCR4.RXSEL = 011 <sub>B</sub> )
RXDC5E	-		receive input E (NPCR4.RXSEL = 100 <sub>B</sub> )
RXDC5 [G:F]	1		receive inputs [G:F] (NPCR4.RXSEL = 101 <sub>B</sub> to 110 <sub>B</sub> )
RXDC5H	0		receive input H (NPCR4.RXSEL = 111 <sub>B</sub> )
TXDC5	P2.0	O	transmit output
	P7.2		
	P11.3		
General MultiCAN Signals			
INT_O[15:0]	interrupt controller	O	interrupt output lines (service requests) <sup>1)</sup>
	INT_O15 to CCU62 and CCU63		notification for timer start on.

<sup>1)</sup> CAN interrupts are shared with interrupts of other modules. See the ISSRx register description.

## 22.5.7 MultiCAN Module Register Address Map

In the XE16xyM, the registers of the MultiCAN module are located in the following address range:

**Table 22-19 Registers Address Space**

Module	Base Address	End Address	Note
CAN	200000 <sub>H</sub>	203FFF <sub>H</sub>	16 kBytes
CANa	208000 <sub>H</sub>	20AFFF <sub>H</sub>	12 kBytes

*Note: The complete and detailed address map of the MultiCAN modules is described in the chapter "Register Overview" of the XE16xyM System Units User's Manual.*

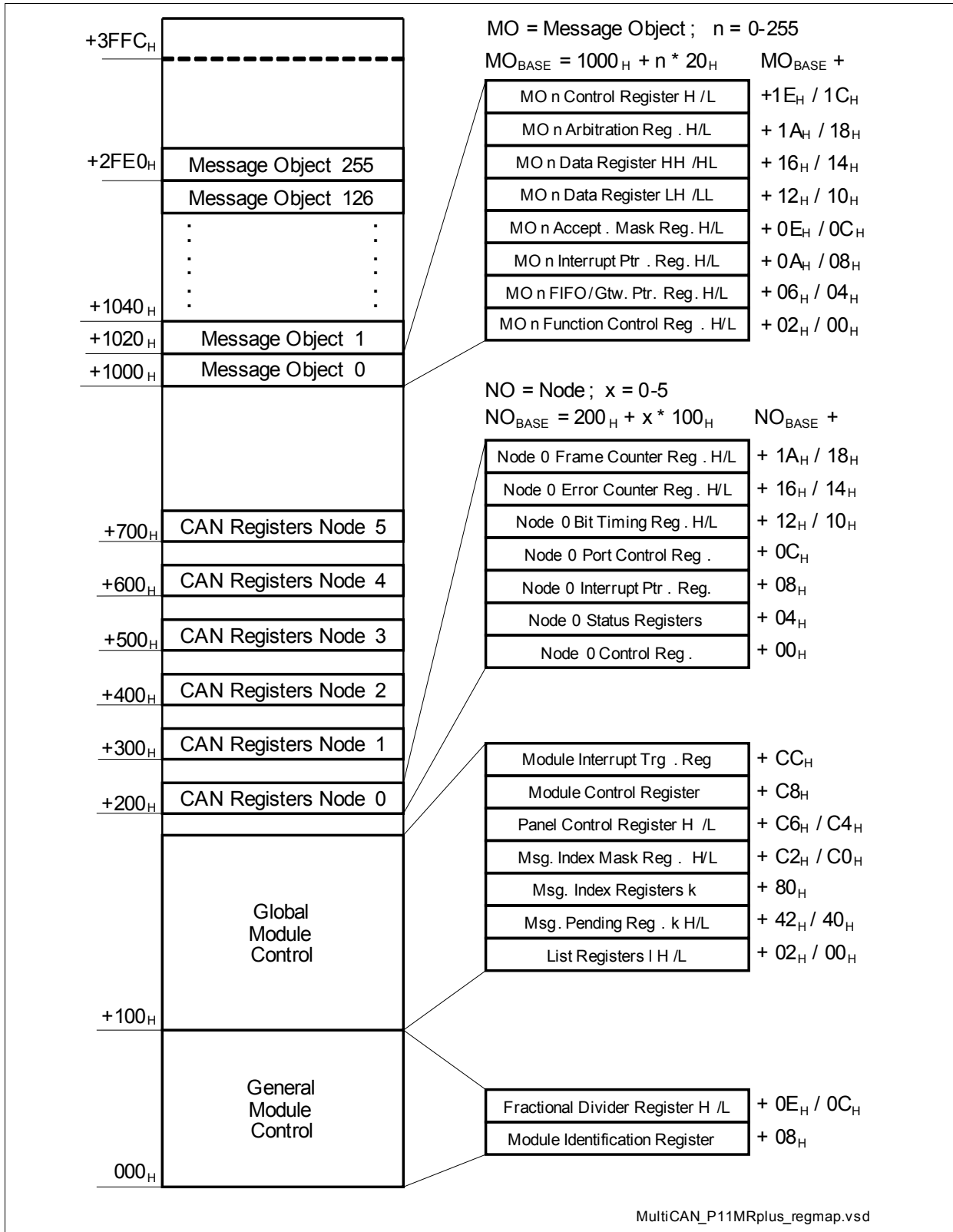
In the MultiCAN module address range, the following register blocks are located at the offset start addresses, see [Figure 22-26](#):

- 0000<sub>H</sub>      General registers for clock control, fractional divider, ID
- 0100<sub>H</sub>      Global Module Control registers
- 0200<sub>H</sub>      CAN node 0 registers
- 1000<sub>H</sub>      Message object memory (32 bytes for each object)

The CAN RAM is automatically initialized after reset by the list controller in order to ensure correct list pointers in each message object. The end of this CAN RAM initialization is indicated by bit PANCTR.BUSY becoming inactive. Before the end of the initialization sequence, the CAN module must not be accessed with other instructions than polling for bit PANCTR.BUSY.

The CAN RAM can be optionally enabled for parity detection. The feature is controlled in the SCU.

**Controller Area Network (MultiCAN) Controller**



**Figure 22-26 MultiCAN Module Register Map**



## **23 Appendix: Functional and Operational Updates**

The XE16xyM devices are the successors of the 0-series devices within the XE166 Family of microcontrollers. This new series of product types provides a number of enhancements.

Some aspects of the existing features had to be adapted to realize these enhancements. Those features are, therefore, controlled in a different way, while providing comparable functions.

This appendix summarizes both types of updates:

- **Functional updates:** new features have been incorporated to add functionality to applications.
- **Operational updates:** existing features are controlled in a different way.

*Note: These hints help to exploit additional functionalities, to avoid malfunctions when upgrading to the new series of microcontrollers, and to design software that can run on both types of products.*

### **Flash Memory**

The XE16xyM provides an additional 64-Kbyte Flash module. This Flash module can be used for EEPROM emulation as well as for code and constant storage. The maximum available Flash size, therefore, is 832 Kbytes. The logical sectors have been adapted. Individual ECC status and trap control supports parallel programming of Flash modules. The program Flash interrupt signal is routed to an SCU interrupt node.

### **RAM Areas**

The maximum available program SRAM (PSRAM) is 32 Kbytes, compared to 64 Kbytes. The XE16xyM offers 8 Kbytes of standby memory (SBRAM).

### **MPU**

The new memory protection unit (MPU) of the XE16xyM supports applications with dedicated isolated memory regions for different tasks.

### **MCHK**

The new memory checker unit (MCHK) is a hardware CRC32 generator supporting safety-oriented applications.

### **ECC**

The error correction mechanism with single-error detection and correction (ECC) provides enhanced protection for the RAMs of the XE16xyM. The known parity protection can be used alternatively, for compatibility reasons.

## **Appendix: Functional and Operational Updates**

### **Startup Configuration**

The hardware startup configuration via Port 10 has been reworked to support additional user-selectable startup modes. This configuration is ignored when  $\overline{\text{TRST}}$  is low, so in this standard case no configuration at all is required.

### **Debugging via DAP**

The new device access port (DAP) offers a two-wire debug interface in addition to the standard JTAG debug interface.

### **Simplified EVR Control**

Because the on-chip EVRs can fully supply the XE16xyM, the options selected by pin TRef have been removed. This reduces the configuration effort and also provides one additional IO pin, P2.13.

### **Clock Features**

In the XE16xyM, the clock input pin CLKIN1 can also feed the on-chip PLL.  
The oscillator counter indicates stable operation of the oscillator, indicated by bit OSCSTAB.  
The crystal oscillator gain is now programmable.

### **System Timer**

The new system timer (STM) supports applications requiring calender/clock functions even during power reduction phases.

### **Serial Channels**

The XE16xyM offers up to 8 serial channels, compared to up to 6.  
A busy flag provides additional information in UART mode.  
A start-of-frame flag provides additional information in SSC mode.  
Additional optional port connections enhance the flexibility.  
Requests from the additional module USIC3 can be selected instead of CAPCOM2 requests via bits ISS2/ISS3 in register ISSR.

### **A/D Converters**

The ADC modules are equipped with additional features:

- Broken wire detection and multiplexer test mode increase the reliability
- Equidistant sampling provides optimized input data for filter algorithms
- Enhanced trigger/gating capabilities (request source registers RSIRx instead of PISEL) optimize the interaction of on-chip modules
- Improved hardware control of external multiplexers

**Appendix: Functional and Operational Updates****MultiCAN**

The MultiCAN module in the XE16xyM offers up to 6 CAN nodes (compared to 5) and up to 256 message objects (compared to 128). Additional optional port connections enhance the flexibility. A series of errata has been fixed in this version of the MultiCAN module (a new certification report is available).

**GPT12E**

The timer module (GPT12E) has been equipped with a port input select register (PISEL) and additional port connections.

**CCU6x**

The CCU6 units feature 8 hardware run inputs (compared to 4). Internal connections to and from the ADC modules have been improved. The shadow transfer capabilities have been improved.

Trigger signals to and from the USIC modules have been improved.  
Additional optional port connections enhance the flexibility.

**ERU**

The four interrupt request lines of the external request unit (ERU) are assigned to dedicated interrupt nodes. They are no more shared with CAPCOM2 interrupts. Bits ISS0 ... ISS3 and ISS8 ... ISS11 in register ISSR no more select ERU requests.

**Wake-up Timer**

The wake-up timer (WUT) can generate periodical trigger signals. For this purpose the clock source is now selectable and the prescaler is programmable. The counter register WICR has been replaced with the reload register WUTREL. Therefore, the counter cannot be read directly by software.

**Watchdog Timer**

The watchdog timer (WDT) puts the XE16xyM into a permanent reset state after expiring for the second time. The status flag indicating the first occurrence must explicitly be cleared by writing to a bit. It is not automatically cleared upon servicing the WDT.

**LXBus Address Range**

The address range for the LXBus has been increased to accommodate the alternate address range for modules MultiCAN, USIC0, USIC1, and USIC2.

**Appendix: Functional and Operational Updates**

**Miscellaneous**

Additional pins can be selected as wake-up trigger input for the ESR logic.

Fast startup mode supports intermittent operation without Flash memory.

Register ESRDAT is protected by the register security mechanism.

*Note: Please refer to the corresponding Errata Sheets for a summary of the errata that have been fixed in the XE16xyM.*

## Keyword Index

This section lists a number of keywords which refer to specific details of the XE16xyM in terms of its architecture, its functional units or functions. This helps to quickly find the answer to specific questions about the XE16xyM.

*Note: Registers are listed in a separate index: [Register Index](#).*

### A

Acronyms 1-9

ADC

Equidistant Sampling **18-106**

Address Windows (External Bus) 11-9

Addressing Modes

Code Addressing 5-33

CoREG Addressing Mode 5-47

DSP Addressing Modes 5-43

Indirect Addressing Modes 5-41

Long Addressing Modes 5-38

Short Addressing Modes 5-36

ALU 5-54

Atomic Instruction 5-8

### B

Baudrate

Bootstrap Loader 12-25

Bit

Manipulation 5-58

Manipulation Instructions 14-2

Protected 3-16

Protection 5-58

reserved 2-17

Block Diagram ITC / PEC 7-4

Bootstrap Loader 12-18

Branch Prediction 5-7

Bus Phases (External Bus) 11-2

Byte Write Configuration (EBC) 11-19

### C

C166S-V2 5-1

CAN

Block diagram 22-1

Clock control 22-102

Features 22-2

Functional description 22-3

Interrupt structure 22-105

Module implementation 22-106

MultiCAN

Analysis mode 22-18

Bit timing 22-9

Block diagram 22-6

Error handling 22-11

Gateway mode 22-41

Interrupts 22-12

Message acceptance filtering  
22-21

Message object FIFO 22-36

Message object lists 22-13

Node control 22-9

Overview 22-4

Registers

LISTiH **22-57**

LISTiL **22-58**

MCR **22-55**

MITR **22-56**

MOAMRnH **22-95**

MOAMRnL **22-95**

MOARnH **22-97**

MOARnL **22-98**

MOCTRnH **22-79, 22-82**

MOCTRnL **22-80, 22-82**

MODATAnHH **22-101**

MODATAnHL **22-101**

MODATAnLH **22-100**

MODATAnLL **22-100**

MOFCRnH **22-89**

MOFCRnL **22-91**  
 MOFGPRnH **22-93**  
 MOFGPRnL **22-93**  
 MOIPRnH **22-87**  
 MOIPRnL **22-87**  
 MSIDk **22-60**  
 MSIMASKH **22-61**  
 MSIMASKL **22-61**  
 MSPNDkH **22-59**  
 MSPNDkL **22-59**  
 NBTRxH **22-72**  
 NBTRxL **22-72**  
 NCRx **22-62**  
 NECNTxH **22-73**  
 NECNTxL **22-74**  
 NFCRxH **22-75**  
 NFCRxL **22-76**  
 NIPRx **22-70**  
 NPCRx **22-71**  
 NSRx **22-66**  
 PANCTRH **22-50**  
 PANCTRL **22-50**  
 CAPCOM12  
   Capture Mode 19-9  
   Counter Mode 19-6  
 CAPCOM2 2-17  
 Capture Mode  
   GPT1 16-28  
   GPT2 (CAPREL) 16-50  
 CCU6 2-19  
 Clock  
   generation 2-33  
   output signal 8-24  
 Clock System  
   Main oscillator 8-5  
   Oscillator run detection 8-15  
 Clock system  
   Clock source 8-8  
   Gain control 8-6  
   PLL, see "PLL"  
 Concatenation of Timers 16-24, 16-49  
 Context Switch 5-28  
 Count direction 16-6, 16-38

Counter 16-22, 16-47  
 Counter Mode (GPT1) 16-10, 16-42  
 CPU 2-2, 5-1  
 CPU Stack Pointer 5-49

## **D**

Data Management Unit (Introduction) 2-9  
 Data Page 5-39  
 Demultiplexed External Bus 11-5  
 Development Support 1-8  
 Direction  
   count 16-6, 16-38  
 Division 5-60  
 Double-Register Compare 19-18  
 DPP 5-39  
 DSP Processing 5-62

## **E**

EBC  
   Address Windows 11-9  
   Arbitration 11-13  
   Bus Phases 11-2  
   Byte Write Configuration 11-19  
   Idle State 11-17  
   Interface 11-1  
   Master Slave Connection 11-16  
   Ready Control 11-11  
   Register Description 11-18  
   Shutdown Control 11-24  
   Timing 11-2

ESR 8-77

ESRx 7-1

EXTBUS 11-1

Extend Instruction 5-8

External

  Bus 2-14

  Interrupts 7-45

External Bus **11-1**

External Bus Arbitration 11-13

## **F**

Flags 5-53–5-56

Flash

## Keyword Index

- Command Sequences 3-26
  - Change Read Margin 3-28
  - Clear Status 3-28
  - Disable Read Protection 3-35
  - Disable Write Protection 3-36
  - Enter Page Mode 3-29
  - Enter Security Page Mode 3-30
  - Erase Page 3-34
  - Erase Sector 3-33
  - Erase Security Page 3-34
  - Load Page Word 3-31
  - Program Page 3-32
  - Re-Enable Read/Write Protection 3-37
  - Reset to Read 3-28
- Concurrent Program/Erase 3-38
- Definitions 3-20
  - Array 3-21
  - Block 3-21
  - Drain Disturb 3-21
  - Endurance 3-20
  - Erasing 3-20
  - Memory 3-21
  - Page 3-21
  - Programming 3-20
  - Retention 3-20
  - Sector 3-21
- ECC 3-41
- EEPROM Emulation 3-54
- Interrupt Generation 3-56
- Linear Code Pre-Fetch 3-24
- Logical Sectors 3-43
- Margin Reads 3-42
- Operating Modes 3-22
  - Command Mode 3-23
  - Page Mode 3-23
  - Read Mode 3-22
- Protection
  - Details 3-45
  - Determining RPA and WPA 3-48
  - Effective Read Security 3-49
  - Effective Write Security 3-49
  - Examples 3-52
  - Lower Layer "Physical State" 3-46
  - Middle Layer "Flash State" 3-47
  - Overview 3-42
  - Security Pages 3-51
  - Upper Layer "Protection State" 3-48
- Recommendations
  - EEPROM Emulation 3-57
  - Programming Code and Constant Data 3-56
  - Recommendations 3-56
  - Sequence Errors 3-37
  - Wait States 3-24, 3-61
- Fractional divider
  - Block diagram 22-108
  - Operating modes 22-110
  - Suspend mode 22-111
- Frequency
  - output signal 8-24
- G**
  - Gated timer mode (GPT1) 16-9
  - Gated timer mode (GPT2) 16-41
  - GPRs
    - General Purpose Registers 5-24
  - GPT 2-20
  - GPT1 16-2
  - GPT12E
    - Register Table 16-67
  - GPT2 16-34
- I**
  - Idle State (External Bus) 11-17
  - IMB 3-58
    - Block Diagram 3-58
    - Error Reporting Summary 3-80
    - Overview 3-58
    - Registers
      - Overview 3-60
    - Startup, Shutdown 3-79
  - Incremental Interface Mode (GPT1) 16-11
  - Instruction 14-1
    - Bit Manipulation 14-2

Pipeline 5-9  
 Pointer 5-34  
   protected 14-6  
 Interface  
   External Bus 11-1  
 Interrupt  
   External 7-45  
   Latency 7-48  
   Priority 7-9  
   RTC 17-13  
   System 2-8, 7-3  
 IO Area  
   CPU Pipeline behaviour **5-10**  
  
**L**  
 Latency  
   Interrupt, PEC 7-48  
 LXBUS 11-1  
 LXBus 2-14  
  
**M**  
 MAC Unit 5-62  
 Mask Protection 5-58  
 Master Slave Connection (EBC) 11-16  
 MCHK  
   Functionality 4-2  
   LFSR 4-2  
   Linear Feedback Shift Register 4-2  
   MISR 4-6  
   Multiple Input Shift Register 4-6  
 Memory 2-10, 3-1  
   Address Space Overview 3-1  
   CPU behavior when accessing IO Area  
   **5-10**  
   Data Retention Memories 3-82  
   Data SRAM (DSRAM) 3-10  
   Dual-Port RAM (DPRAM) 3-10  
   External Memory Space 3-18  
   Flash 3-14, **3-20**  
   Flash Emulation 3-13  
   IMB 3-58  
   IO Areas 3-17  
   Little Endian 3-2

Marker Memory (MKMEM) 3-11  
 Memory Map 3-3  
 On-Chip Program Memory Map 3-12  
 Program/Data SRAM (PSRAM) 3-13  
 Protected Bits 3-16  
 Register Areas 3-5  
 Standby RAM (SBRAM) 3-11  
   Accesses 3-82  
   System Stack 3-15  
 Memory Checker  
   Functionality 4-2  
   LFSR 4-2  
   Linear Feedback Shift Register 4-2  
   MISR 4-6  
   Multiple Input Shift Register 4-6  
 MPU  
   Registers Overview 6-3  
 Multiplexed External Bus 11-6  
 Multiplication 5-60  
  
**N**  
 Non-Segmented Mode 5-33  
  
**O**  
 OCDS  
   Requests 7-47  
  
**P**  
 PD+ Bus  
   Bit Protection 5-58  
 PEC 2-10  
   Latency 7-48  
 Peripheral  
   Summary 2-15  
 Pins 10-1  
 Pipeline 5-9  
 PLL **8-7**  
   Functionality 8-7  
   Switching parameters 8-16  
 Port 2-31  
   Temperature compensation 8-176  
 Ports  
   Configuring a Pin 9-14



Output register Pn\_OUT 9-9

Pad driver control 9-6

Structure

    Analog 9-4

    Hardware Override 9-3

    Standard 9-2

Prefetch 5-5

Program Management Unit (Introduction)  
    2-9

Protected

    instruction 14-6

Protected Bits 3-16, 5-58

## **R**

Ready Control (External Bus) 11-11

Real Time Clock (->RTC) 2-22, 17-1

Registers

    EBC 11-18

Reserved bits 2-17

Reset 8-57

    Module behavior 8-65

RTC 2-22, 17-1

    Registers

        T14 17-8

        T14REL 17-8

## **S**

Segmented Mode 5-33

Shutdown Control (EBC) 11-24

System Stack 5-49

## **T**

Temperature compensation 8-176

Timer 16-2, 16-34

    Auxiliary Timer 16-15, 16-43

    Concatenation 16-24, 16-49

    Core Timer 16-4, 16-36

    Counter Mode (GPT1) 16-10, 16-42

    Gated Mode (GPT1) 16-9

    Gated Mode (GPT2) 16-41

    Incremental Interface Mode (GPT1)  
    16-11

    Mode (GPT1) 16-8

    Mode (GPT2) 16-40

Timing (External Bus) 11-2

Tools 1-8

## **U**

USIC

    ASC mode 21-110

        Automatic shaping 21-118

        Baud rate 21-116

        Bit timing 21-115

        Collision detection 21-116

        Data transfer interrupts 21-120

        EOF control 21-118

        Frame format 21-111

        Noise detection 21-116

        Protocol interrupts 21-119

        Protocol registers 21-123

        Pulse shaping 21-117

        Receive buffer 21-121

        Signals 21-110

        Sync-break detection 21-121

        Transfer status 21-121

Baud rate 21-8

Channel structure 21-5

Data buffer 21-10

Data shifting and handling 21-9

Data transfer interrupts 21-21

External frequency 21-41

Feature set 21-2

FIFO buffer 21-11

FIFO data buffer 21-79

Fractional divider 21-41

General interrupts 21-20

IIC mode 21-161

    Baud rate 21-166

    Byte stretching 21-166

    Data bit symbol 21-173

    Data flow handling 21-174

    Frame format 21-164

    Master arbitration 21-166

    Master transmission 21-178

    Mode control 21-167

    Protocol interrupts 21-168

- Protocol registers 21-179
- Receiver address acknowledge 21-169
- Receiver handling 21-169
- Receiver status 21-170
- Signals 21-162
- Start symbol 21-172
- Stop symbol 21-173
- Symbol timing 21-171
- Transmission chain 21-166
- Transmit data 21-174
- IIS mode 21-185
  - Baud rate 21-194
  - Connection of Audio devices 21-188
  - Data interrupts 21-192
  - Frame and word length 21-189
  - Mode control 21-189
  - Protocol interrupts 21-197, 21-198
  - Protocol overview 21-187
  - Protocol registers 21-199
  - Receive data 21-193
  - Signals 21-185
  - Slave mode operation 21-198
  - Transfer delay 21-187, 21-190
  - Transmit data 21-192
  - WA generation 21-195, 21-196
- Implementation
  - Address map 21-207
  - Channels 21-206
  - I/O lines of USIC0 21-213
  - I/O lines of USIC1 21-216
  - I/O lines of USIC2 21-219
  - I/O lines of USIC3 21-222
  - Interrupt registers 21-210
  - Overview 21-205
- Input stages 21-6, 21-36
- Kernel registers
  - Baud rate registers 21-46
  - BRGH 21-50
  - BRGL 21-48
  - BYP 21-89
  - BYPCLR 21-91
  - BYPCLR 21-89
  - CCFG 21-28
  - CCR 21-25
  - Channel control and configuration registers 21-25
  - Data buffer registers 21-69
  - DX0CR 21-38
  - DX1CR 21-38
  - DX2CR 21-38
  - FDRH 21-47
  - FDRL 21-46
  - FIFO buffer and bypass registers 21-89
  - FMRH 21-68
  - FMRL 21-67
  - INPRH 21-32
  - INPRL 21-31
  - Input stage register 21-38
  - INx 21-105
  - KSCFG 21-29
  - OUTDRH 21-107
  - OUTDRL 21-107
  - OUTRH 21-106
  - OUTRL 21-106
  - Overview 21-14
  - PCRH 21-33, 21-126, 21-154, 21-179, 21-201
  - PCRL 21-33, 21-123, 21-152, 21-179, 21-199
  - Protocol registers 21-33
  - PSCR 21-35
  - PSR 21-34, 21-127, 21-156, 21-182, 21-202
  - RBCTRH 21-102
  - RBCTRL 21-101
  - RBUF 21-76
  - RBUF0 21-70
  - RBUF01SRH 21-73
  - RBUF01SRL 21-70
  - RBUF1 21-73
  - RBUFD 21-77
  - RBUFSR 21-78
  - SCTRH 21-59

SCTRL 21-57  
TBCTRH 21-99  
TBCTRL 21-98  
TBUFx 21-69  
TCSRH 21-65  
TCSRL 21-60  
Transfer control/status registers  
21-57  
TRBPTRH 21-109  
TRBPTRL 21-108  
TRBSCR 21-96  
TRBSRH 21-95  
TRBSRL 21-92  
Mode control 21-19  
Module registers  
  USIC0\_IDH 21-209  
  USIC0\_IDL 21-208  
  USIC1\_IDH 21-209  
  USIC1\_IDL 21-208  
  USIC2\_IDH 21-209  
  USIC2\_IDL 21-208  
  USIC3\_IDH 21-209  
  USIC3\_IDL 21-208  
Output signals 21-7  
Protocol control and status 21-18  
Protocol interrupts 21-24  
Protocol related counter 21-42  
Receive buffering 21-55  
Registers overview 21-14  
SSC mode 21-131  
  Automatic Shadow mechanism  
  21-139  
  Baud rate 21-143  
  Data frame control 21-140  
  EOF control 21-148, 21-151  
  Master mode 21-143  
  Protocol interrupts 21-147, 21-150  
  Protocol registers 21-152  
  Receive buffer 21-141  
  Signals 21-131  
  Slave mode 21-150  
  Slave select delay 21-146  
  Slave select generation 21-144

Time quanta counter 21-44  
Transmit buffering 21-51

## **W**

Watchdog 2-30  
Watchdog Timer 8-180  
  Kernel Registers 8-185  
  Modes of operation  
    Disable Mode 8-183  
    Normal Mode 8-182  
    Prewarning Mode 8-183  
  Period calculation 8-181

## **Z**

Zero-Cycle Jump 5-7

## Register Index

This section lists the registers of the XE16xyM. This helps to quickly find the reference to the description of the respective register.

*Note: Keywords are listed in a separate index: [Keyword Index](#).*

### A

ADC0\_KSCFG 18-24  
 ADCx\_ALR0 18-79  
 ADCx\_ASENr 18-39  
 ADCx\_BWDCFG 18-116  
 ADCx\_BWDENr 18-115  
 ADCx\_CHCTR 18-70  
 ADCx\_CHINCR 18-75  
 ADCx\_CHINFR 18-74  
 ADCx\_CHINPR 18-76  
 ADCx\_CRCR 18-45  
 ADCx\_CRM 18-47  
 ADCx\_CRPR 18-46  
 ADCx\_EMCTR 18-112  
 ADCx\_EMENr 18-110  
 ADCx\_EVINCR 18-95  
 ADCx\_EVINFR 18-94  
 ADCx\_EVINPR 18-96  
 ADCx\_GLOBCTR 18-27  
 ADCx\_GLOBSTR 18-29  
 ADCx\_ID 18-26  
 ADCx\_INPR 18-72  
 ADCx\_LCB 18-73  
 ADCx\_Q0R 18-59  
 ADCx\_QBUR 18-61  
 ADCx\_QINR 18-63  
 ADCx\_QMR 18-54  
 ADCx\_QSR 18-57  
 ADCx\_RCR 18-92  
 ADCx\_RESRAV 18-89  
 ADCx\_RESRA 18-89  
 ADCx\_RESRV 18-88  
 ADCx\_RESR 18-88  
 ADCx\_RSIR 18-32  
 ADCx\_RSPR 18-40

ADCx\_RSSR 18-90  
 ADCx\_SYNCTR 18-114  
 ADCx\_VFR 18-91  
 ADDRSEL7 11-28  
 ADDRSELx 11-23

### C

CAN\_LISTiH 22-57  
 CAN\_LISTiL 22-58  
 CAN\_MCR 22-55  
 CAN\_MITR 22-56  
 CAN\_MOAMRnH 22-95  
 CAN\_MOAMRnL 22-95  
 CAN\_MOARnH 22-97  
 CAN\_MOARnL 22-98  
 CAN\_MOCTRnH 22-79, 22-82  
 CAN\_MOCTRnL 22-80  
 CAN\_MODATAnHH 22-101  
 CAN\_MODATAnHL 22-101  
 CAN\_MODATAnLH 22-100  
 CAN\_MODATAnLL 22-100  
 CAN\_MOFCRnH 22-89  
 CAN\_MOFCRnL 22-91  
 CAN\_MOFGPRnH 22-93  
 CAN\_MOFGPRnL 22-93  
 CAN\_MOIPRnH 22-87  
 CAN\_MOIPRnL 22-87  
 CAN\_MOSTATnL 22-82  
 CAN\_MSIDk 22-60  
 CAN\_MSIMASKH 22-61  
 CAN\_MSIMASKL 22-61  
 CAN\_MSPNDkH 22-59  
 CAN\_MSPNDkL 22-59  
 CAN\_NBTRxH 22-72  
 CAN\_NBTRxL 22-72

CAN\_NCRx 22-62  
 CAN\_NECNTxH 22-73  
 CAN\_NECNTxL 22-74  
 CAN\_NFCRxH 22-75  
 CAN\_NFCRxL 22-76  
 CAN\_NIPRx 22-70  
 CAN\_NPCRx 22-71  
 CAN\_NSRx 22-66  
 CAN\_PANCTRH 22-50  
 CAN\_PANCTRL 22-50  
 CAPREL 16-59  
 CC2\_DRM 19-40  
 CC2\_KSCCFG 17-15, 19-43  
 CC2\_M4/5/6/7 19-37  
 CC2\_OUT 19-39  
 CC2\_SEE 19-42  
 CC2\_SEM 19-42  
 CCU6x\_CC63R 20-64  
 CCU6x\_CC63SR 20-64  
 CCU6x\_CC6xR 20-33  
 CCU6x\_CC6xSR 20-34  
 CCU6x\_CMPMODIF 20-39  
 CCU6x\_CMPSTAT 20-37  
 CCU6x\_IEN 20-98  
 CCU6x\_INP 20-101  
 CCU6x\_IS 20-91  
 CCU6x\_ISR 20-96  
 CCU6x\_ISS 20-94  
 CCU6x\_KSCCFG 20-112  
 CCU6x\_KSCSR 20-114  
 CCU6x\_MCMOUT 20-87  
 CCU6x\_MCMOUTS 20-86  
 CCU6x\_MODCTR 20-78  
 CCU6x\_PISELH 20-109  
 CCU6x\_PISELL 20-107  
 CCU6x\_PSLR 20-83  
 CCU6x\_T12 20-32  
 CCU6x\_T12DTC 20-35  
 CCU6x\_T12MSEL 20-40  
 CCU6x\_T12PR 20-32  
 CCU6x\_T13 20-62  
 CCU6x\_T13PR 20-63  
 CCU6x\_TCTR0 20-41

CCU6x\_TCTR2 20-44  
 CCU6x\_TCTR4 20-47  
 CCU6x\_TRPCTR 20-80  
 CP 5-31  
 CPUCON1 5-21  
 CPUCON2 5-22  
 CSP 5-34

## **D**

DPP0/1/2/3 5-39

## **E**

EBCMOD0 11-18  
 EBCMOD1 11-20

## **F**

FCONCS7 11-27  
 FCONCSx 11-22  
 FL\_KSCCFG 3-76

## **G**

GPT12E\_CAPREL 16-59  
 GPT12E\_KSCCFG 16-63  
 GPT12E\_T2,-T3,-T4 16-32  
 GPT12E\_T2CON 16-15  
 GPT12E\_T3CON 16-4  
 GPT12E\_T4CON 16-17  
 GPT12E\_T5,-T6 16-59  
 GPT12E\_T5CON 16-43  
 GPT12E\_T6CON 16-36

## **I**

IDX0/1 5-43  
 IMB module registers 3-60  
 IMB\_ECC\_STAT 3-73  
 IMB\_ECC\_TRAP 3-71  
 IMB\_FSR\_BUSY 3-65  
 IMB\_FSR\_OP 3-66  
 IMB\_FSR\_PROT 3-68  
 IMB\_IMBCTRH 3-62  
 IMB\_IMBCTRL 3-61  
 IMB\_INTCTR 3-64  
 IMB\_MAR0 3-69

IMB\_PROCONx 3-70  
 IP 5-34

## **M**

MAH 5-67  
 MAL 5-66  
 MCHK\_COUNT 4-18  
 MCHK\_IR 4-15  
 MCHK\_RRH 4-16  
 MCHK\_RRL 4-16  
 MCHK\_TPRH 4-19  
 MCHK\_TPRL 4-19  
 MCW 5-63  
 MDC 5-61  
 MDH 5-60  
 MDL 5-61  
 MEM\_KSCCFG 3-75  
 MPU\_PM0 6-5  
 MPU\_PMx 6-7  
 MPU\_PRA 6-9  
 MPU\_PRD 6-7  
 MPU\_PRLx 6-4  
 MPU\_PRUX 6-4  
 MRW 5-70  
 MSW 5-68

## **O**

ONES 5-72

## **P**

Pn\_DIDIS  
     P15 9-16  
     P5 9-16  
 Pn\_IN 9-12  
 Pn\_IOCRx 9-13  
 Pn\_OMRH  
     P10 9-10  
     P2 9-10  
 Pn\_OMRL 9-10  
 Pn\_OUT 9-9  
 Pn\_POCON 9-7  
 Ports  
     Pn\_IN 9-12

Pn\_IOCRx 9-13  
 Pn\_OMR 9-10  
 PSW 5-53

## **Q**

QR0/1 5-42  
 QX0/1 5-44

## **R**

RELH/L 17-10  
 RTC\_CON 17-6  
 RTC\_IC 17-14  
 RTC\_ISNC 17-14  
 RTC\_RELH/L 17-10  
 RTC\_RTCH/L 17-9  
 RTC\_T14 17-8  
 RTC\_T14REL 17-8  
 RTCH/L 17-9

## **S**

SBRAM module registers 3-84  
 SBRAM\_DATA0 3-87  
 SBRAM\_DATA1 3-88  
 SBRAM\_RADD 3-84  
 SBRAM\_WADD 3-85  
 SP 5-50  
 SPSEG 5-50  
 STKOV 5-52  
 STKUN 5-52

## **T**

T14 17-8  
 T14REL 17-8  
 T2, T3, T4 16-32  
 T2CON 16-15  
 T3CON 16-4  
 T4CON 16-17  
 T5, T6 16-59  
 T5CON 16-43  
 T6CON 16-36  
 TCONCS7 11-25  
 TCONCSx 11-21

## **U**

USIC0\_IDH 21-209  
 USIC0\_IDL 21-208  
 USIC1\_IDH 21-209  
 USIC1\_IDL 21-208  
 USIC2\_IDH 21-209  
 USIC2\_IDL 21-208  
 USIC3\_IDH 21-209  
 USIC3\_IDL 21-208  
 UxCy\_BRGH 21-50  
 UxCy\_BRGL 21-48  
 UxCy\_BYP 21-89  
 UxCy\_BYPCRH 21-91  
 UxCy\_BYPCRL 21-89  
 UxCy\_CCFG 21-28  
 UxCy\_CCR 21-25  
 UxCy\_DX0CR 21-38  
 UxCy\_DX1CR 21-38  
 UxCy\_DX2CR 21-38  
 UxCy\_FDRH 21-47  
 UxCy\_FDRL 21-46  
 UxCy\_FMRH 21-68  
 UxCy\_FMRL 21-67  
 UxCy\_INPRH 21-32  
 UxCy\_INPRL 21-31  
 UxCy\_INx 21-105  
 UxCy\_KSCFG 21-29  
 UxCy\_OUTDRH 21-107  
 UxCy\_OUTDRL 21-107  
 UxCy\_OUTRH 21-106  
 UxCy\_OUTRL 21-106  
 UxCy\_PCRH 21-33, 21-126, 21-154,  
     21-179, 21-201  
 UxCy\_PCRL 21-33, 21-123, 21-152,  
     21-179, 21-199  
 UxCy\_PSCR 21-35  
 UxCy\_PSR 21-34, 21-127, 21-156,  
     21-182, 21-202  
 UxCy\_RBCTRH 21-102  
 UxCy\_RBCTRL 21-101  
 UxCy\_RBUF 21-76  
 UxCy\_RBUF0 21-70

UxCy\_RBUF01SRH 21-73  
 UxCy\_RBUF01SRL 21-70  
 UxCy\_RBUF1 21-73  
 UxCy\_RBUFD 21-77  
 UxCy\_RBUFSR 21-78  
 UxCy\_SCTRH 21-59  
 UxCy\_SCTRL 21-57  
 UxCy\_TBCTRH 21-99  
 UxCy\_TBCTRL 21-98  
 UxCy\_TBUFx 21-69  
 UxCy\_TCSRH 21-65  
 UxCy\_TCSRL 21-60  
 UxCy\_TRBPTRH 21-109  
 UxCy\_TRBPTRL 21-108  
 UxCy\_TRBSCR 21-96  
 UxCy\_TRBSRH 21-95  
 UxCy\_TRBSRL 21-92

## **Z**

ZEROS 5-72

[www.infineon.com](http://www.infineon.com)