# XC878CLM

## 8-Bit Single Chip Microcontroller




Microcontrollers

Never stop thinking

**Edition 2008-04**

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

**© Infineon Technologies AG 2008.**
**All Rights Reserved.**

### Legal Disclaimer

The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics (“Beschaffenheitsgarantie”). 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 your nearest Infineon Technologies Office (**www.infineon.com**).

### Warnings

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

Infineon Technologies Components may only be used in life-support devices or systems 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.

# XC878CLM

## 8-Bit Single Chip Microcontroller

Microcontrollers

Never stop thinking

**Revision History:** **2008-04** V 1.0

Previous Version:

| Page | Subjects (major changes since last revision) |
|---|---|
| | |
| | |
| | |
| | |
| | |
| | |
| | |

**We Listen to Your Comments**

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

**mcdocu.comments@infineon.com**

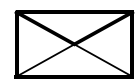


**Table of Contents** **Page**


**1 Introduction** . . . 1-1
1.1 Feature Summary . . . 1-4
1.2 Pin Configuration . . . 1-6
1.3 Pin Definitions and Functions . . . 1-7
1.4 Chip Identification Number . . . 1-18
1.5 Text Conventions . . . 1-19
1.6 Reserved, Undefined and Unimplemented Terminology . . . 1-20
1.7 Acronyms . . . 1-20

**2 Processor Architecture** . . . 2-1
2.1 Functional Description . . . 2-1
2.2 CPU Register Description . . . 2-3
2.2.1 Stack Pointer (SP) . . . 2-3
2.2.2 Data Pointer (DPTR) . . . 2-3
2.2.3 Accumulator (ACC) . . . 2-3
2.2.4 B Register . . . 2-3
2.2.5 Program Status Word . . . 2-4
2.2.6 Extended Operation (EO) . . . 2-5
2.2.7 Power Control (PCON) . . . 2-6
2.3 Memory Extension . . . 2-7
2.3.1 Memory Extension Stack . . . 2-7
2.3.2 Memory Extension Effects . . . 2-7
2.3.3 Memory Extension Registers . . . 2-9
2.4 Instruction Timing . . . 2-11

**3 Memory Organization** . . . 3-1
3.1 Program Memory . . . 3-3
3.2 Data Memory . . . 3-3
3.2.1 Internal Data Memory . . . 3-3
3.2.2 External Data Memory . . . 3-4
3.3 Memory Protection Strategy . . . 3-5
3.3.1 Flash Memory Protection . . . 3-5
3.3.2 Miscellaneous Control Register . . . 3-9
3.4 Special Function Registers . . . 3-10
3.4.1 Address Extension by Mapping . . . 3-10
3.4.1.1 System Control Register 0 . . . 3-12
3.4.2 Address Extension by Paging . . . 3-13
3.4.2.1 Page Register . . . 3-15
3.4.3 Bit-Addressing . . . 3-16
3.4.4 System Control Registers . . . 3-17
3.4.4.1 Bit Protection Scheme . . . 3-19
3.4.5 XC878 Register Overview . . . 3-21
3.4.5.1 CPU Registers . . . 3-21

**Table of Contents** **Page**


3.4.5.2 MDU Registers . . . 3-23
3.4.5.3 CORDIC Registers . . . 3-24
3.4.5.4 System Control Registers . . . 3-24
3.4.5.5 WDT Registers . . . 3-27
3.4.5.6 Port Registers . . . 3-28
3.4.5.7 ADC Registers . . . 3-30
3.4.5.8 Timer 2 Compare/Capture Unit Registers . . . 3-34
3.4.5.9 Timer 21 Registers . . . 3-36
3.4.5.10 CCU6 Registers . . . 3-37
3.4.5.11 UART1 Registers . . . 3-41
3.4.5.12 SSC Registers . . . 3-41
3.4.5.13 MultiCAN Registers . . . 3-42
3.4.5.14 OCDS Registers . . . 3-43
3.4.5.15 Flash Registers . . . 3-44
3.5 Boot ROM Operating Mode . . . 3-44
3.5.1 User Mode . . . 3-46
3.5.2 Bootstrap Loader Mode . . . 3-47
3.5.3 OCDS Mode . . . 3-47
3.5.4 User JTAG Mode . . . 3-47

**4 Flash Memory** . . . 4-1
4.1 Flash Memory Map . . . 4-2
4.2 Flash Block Pagination . . . 4-2
4.3 Operating Modes . . . 4-3
4.3.1 Read Operation . . . 4-3
4.3.2 Program Operation . . . 4-3
4.3.3 Erase Operation . . . 4-4
4.3.4 Mass Erase Operation . . . 4-4
4.3.5 Abort Operation . . . 4-5
4.4 Flash Timer . . . 4-7
4.5 Error Detection and Correction . . . 4-8
4.6 In-System Programming . . . 4-9
4.7 In-Application Programming . . . 4-10
4.7.1 Flash Programming . . . 4-10
4.7.2 Flash Page Erasing and Mass Erasing . . . 4-11
4.7.3 Get Chip Information . . . 4-12
4.8 Register Map . . . 4-14
4.9 Register Description . . . 4-14

**5 Interrupt System** . . . 5-1
5.1 Interrupt Structure . . . 5-8
5.1.1 Interrupt Structure 1 . . . 5-8
5.1.2 Interrupt Structure 2 . . . 5-9

**Table of Contents** **Page**


5.1.2.1 System Control Register 0 . . . . . 5-10
5.2 Interrupt Source and Vector . . . . . 5-11
5.3 Interrupt Priority . . . . . 5-13
5.4 Interrupt Handling . . . . . 5-14
5.5 Interrupt Response Time . . . . . 5-15
5.6 Interrupt Registers . . . . . 5-17
5.6.1 Interrupt Node Enable Registers . . . . . 5-17
5.6.2 External Interrupt Control Registers . . . . . 5-21
5.6.3 Interrupt Flag Registers . . . . . 5-27
5.6.4 Interrupt Priority Registers . . . . . 5-35
5.7 Interrupt Flag Overview . . . . . 5-38

**6 Parallel Ports** . . . . . 6-1
6.1 General Description . . . . . 6-2
6.1.1 General Register Description . . . . . 6-4
6.1.1.1 Data Register . . . . . 6-5
6.1.1.2 Direction Register . . . . . 6-6
6.1.1.3 Open Drain Control Register . . . . . 6-6
6.1.1.4 Pull-Up/Pull-Down Device Register . . . . . 6-7
6.1.1.5 Alternate Input and Output Functions . . . . . 6-8
6.1.1.6 Drive Strength Control Register . . . . . 6-9
6.2 Register Address Map . . . . . 6-10
6.3 Port 0 . . . . . 6-12
6.3.1 Functions . . . . . 6-12
6.3.2 Register Description . . . . . 6-16
6.4 Port 1 . . . . . 6-20
6.4.1 Functions . . . . . 6-20
6.4.2 Register Description . . . . . 6-24
6.5 Port 3 . . . . . 6-28
6.5.1 Functions . . . . . 6-28
6.5.2 Register Description . . . . . 6-32
6.6 Port 4 . . . . . 6-36
6.6.1 Functions . . . . . 6-36
6.6.2 Register Description . . . . . 6-40
6.7 Port 5 . . . . . 6-44
6.7.1 Functions . . . . . 6-44
6.7.2 Register Description . . . . . 6-48

**7 Power Supply, Reset and Clock Management** . . . . . 7-1
7.1 Power Supply System with Embedded Voltage Regulator . . . . . 7-1
7.2 Reset Control . . . . . 7-3
7.2.1 Types of Resets . . . . . 7-3
7.2.1.1 Power-On Reset . . . . . 7-3

**Table of Contents** **Page**


7.2.1.2 Hardware Reset . . . 7-4
7.2.1.3 Watchdog Timer Reset . . . 7-5
7.2.1.4 Power-Down Wake-Up Reset . . . 7-5
7.2.1.5 Brownout Reset . . . 7-5
7.2.2 Module Reset Behavior . . . 7-6
7.2.3 Booting Scheme . . . 7-7
7.2.4 Register Description . . . 7-8
7.3 Clock System . . . 7-10
7.3.1 Clock Generation Unit . . . 7-10
7.3.1.1 Functional Description . . . 7-11
7.3.2 Clock Source Control . . . 7-14
7.3.3 Clock Management . . . 7-15
7.3.4 Register Description . . . 7-17

**8 Power Saving Modes** . . . 8-1
8.1 Functional Description . . . 8-2
8.1.1 Idle Mode . . . 8-2
8.1.2 Slow-Down Mode . . . 8-2
8.1.3 Power-down Mode . . . 8-3
8.1.4 Peripheral Clock Management . . . 8-6
8.2 Register Description . . . 8-7

**9 Watchdog Timer** . . . 9-1
9.1 Functional Description . . . 9-2
9.1.1 Module Suspend Control . . . 9-4
9.2 Register Map . . . 9-5
9.3 Register Description . . . 9-5

**10 Multiplication/Division Unit** . . . 10-1
10.1 Functional Description . . . 10-2
10.1.1 Division Operation . . . 10-3
10.1.2 Normalize . . . 10-3
10.1.3 Shift . . . 10-3
10.1.4 Busy Flag . . . 10-4
10.1.5 Error Detection . . . 10-4
10.2 Interrupt Generation . . . 10-4
10.3 Low Power Mode . . . 10-5
10.4 Register Map . . . 10-6
10.5 Register Description . . . 10-7
10.5.1 Operand and Result Registers . . . 10-9
10.5.2 Control Register . . . 10-11
10.5.3 Status Register . . . 10-13

**11 CORDIC Coprocessor** . . . 11-1

**Table of Contents** **Page**


11.1 Features . . . 11-2
11.2 Functional Description . . . 11-3
11.2.1 Operation of the CORDIC Coprocessor . . . 11-3
11.2.2 Interrupt . . . 11-4
11.2.3 Normalized Result Data . . . 11-4
11.2.4 CORDIC Coprocessor Operating Modes . . . 11-5
11.2.4.1 Domains of Convergence . . . 11-7
11.2.4.2 Overflow Considerations . . . 11-8
11.2.5 CORDIC Coprocessor Data Format . . . 11-8
11.2.6 Accuracy of CORDIC Coprocessor . . . 11-9
11.2.7 Performance of CORDIC Coprocessor . . . 11-11
11.3 The CORDIC Coprocessor Kernel . . . 11-12
11.3.1 Arctangent and Hyperbolic Arctangent Look-Up Tables . . . 11-12
11.3.2 Linear Function Emulated Look-Up Table . . . 11-13
11.4 Low Power Mode . . . 11-14
11.5 Register Map . . . 11-15
11.6 Register Description . . . 11-16
11.6.1 Control Register . . . 11-16
11.6.2 Status and Data Control Register . . . 11-18
11.6.3 Data Registers . . . 11-19

**12 Serial Interfaces** . . . 12-1
12.1 UART . . . 12-2
12.1.1 UART Modes . . . 12-2
12.1.1.1 Mode 0, 8-Bit Shift Register, Fixed Baud Rate . . . 12-2
12.1.1.2 Mode 1, 8-Bit UART, Variable Baud Rate . . . 12-3
12.1.1.3 Mode 2, 9-Bit UART, Fixed Baud Rate . . . 12-5
12.1.1.4 Mode 3, 9-Bit UART, Variable Baud Rate . . . 12-5
12.1.2 Multiprocessor Communication . . . 12-7
12.1.3 UART Register Description . . . 12-7
12.1.4 Baud Rate Generation . . . 12-10
12.1.4.1 Fixed Clock . . . 12-10
12.1.4.2 Dedicated Baud-rate Generator . . . 12-11
12.1.4.3 Timer 1 . . . 12-22
12.1.5 Port Control . . . 12-23
12.1.6 Low Power Mode . . . 12-24
12.1.7 Register Map . . . 12-25
12.2 LIN . . . 12-26
12.2.1 LIN Protocol . . . 12-26
12.2.2 LIN Header Transmission . . . 12-28
12.2.2.1 Automatic Synchronization to the Host . . . 12-28
12.2.2.2 Baud Rate Detection of LIN . . . 12-29
12.3 High-Speed Synchronous Serial Interface . . . 12-31

**Table of Contents** **Page**

12.3.1 General Operation . . . 12-32
12.3.1.1 Operating Mode Selection . . . 12-32
12.3.1.2 Full-Duplex Operation . . . 12-33
12.3.1.3 Half-Duplex Operation . . . 12-36
12.3.1.4 Continuous Transfers . . . 12-37
12.3.1.5 Port Control . . . 12-38
12.3.1.6 Baud Rate Generation . . . 12-40
12.3.1.7 Error Detection Mechanisms . . . 12-42
12.3.2 Interrupts . . . 12-44
12.3.3 Low Power Mode . . . 12-46
12.3.4 Register Map . . . 12-46
12.3.5 Register Description . . . 12-47
12.3.5.1 Configuration Register . . . 12-47
12.3.5.2 Baud Rate Timer Reload Register . . . 12-52
12.3.5.3 Transmit and Receive Buffer Register . . . 12-53

**13 Timers** . . . 13-1
13.1 Timer 0 and Timer 1 . . . 13-2
13.1.1 Basic Timer Operations . . . 13-2
13.1.2 Timer Modes . . . 13-2
13.1.2.1 Mode 0 . . . 13-4
13.1.2.2 Mode 1 . . . 13-5
13.1.2.3 Mode 2 . . . 13-6
13.1.2.4 Mode 3 . . . 13-7
13.1.3 Port Control . . . 13-8
13.1.4 Register Map . . . 13-9
13.1.5 Register Description . . . 13-10
13.2 Timer 2 and Timer 21 . . . 13-14
13.2.1 Basic Timer Operations . . . 13-14
13.2.2 Auto-Reload Mode . . . 13-14
13.2.2.1 Up/Down Count Disabled . . . 13-14
13.2.2.2 Up/Down Count Enabled . . . 13-16
13.2.3 Capture Mode . . . 13-18
13.2.4 Count Clock . . . 13-19
13.2.5 External Interrupt Function . . . 13-20
13.2.6 Port Control . . . 13-20
13.2.7 Low Power Mode . . . 13-21
13.2.8 Module Suspend Control . . . 13-22
13.2.9 Register Map . . . 13-24
13.2.10 Register Description . . . 13-25

**14 T2CCU** . . . 14-1
14.1 Features . . . 14-1

**Table of Contents** **Page**

14.2 Requirements of Timer 2 . . . 14-3
14.3 Capture/Compare Timer of T2CCU . . . 14-3
14.3.1 Synchronized Timer Starts . . . 14-4
14.3.2 Cascading Timers For Flexible Count Rate . . . 14-5
14.3.3 Dead-time with CCT Timer . . . 14-5
14.4 T2CCU Capture/Compare Channels . . . 14-8
14.5 Capture/Compare Operation . . . 14-10
14.5.1 Compare Function . . . 14-10
14.5.1.1 Compare Mode 0 . . . 14-13
14.5.1.2 Compare Mode 1 . . . 14-15
14.5.1.3 Concurrent Compare Mode . . . 14-16
14.5.1.4 Using Interrupts in Combination with the Compare Function . . . 14-18
14.5.2 Capture Function . . . 14-19
14.6 T2CCU Registers . . . 14-21
14.6.1 Register Mapping . . . 14-22
14.6.2 T2CCU Registers Description . . . 14-24

**15 Capture/Compare Unit 6** . . . 15-1
15.1 Functional Description . . . 15-3
15.1.1 Timer T12 . . . 15-3
15.1.1.1 Timer Configuration . . . 15-4
15.1.1.2 Counting Rules . . . 15-4
15.1.1.3 Switching Rules . . . 15-5
15.1.1.4 Compare Mode of T12 . . . 15-6
15.1.1.5 Duty Cycle of 0% and 100% . . . 15-8
15.1.1.6 Dead-time Generation . . . 15-8
15.1.1.7 Capture Mode . . . 15-9
15.1.1.8 Single-Shot Mode . . . 15-10
15.1.1.9 Hysteresis-Like Control Mode . . . 15-10
15.1.2 Timer T13 . . . 15-12
15.1.2.1 Timer Configuration . . . 15-12
15.1.2.2 Compare Mode . . . 15-13
15.1.2.3 Single-Shot Mode . . . 15-13
15.1.2.4 Synchronization of T13 to T12 . . . 15-13
15.1.3 Modulation Control . . . 15-15
15.1.4 Trap Handling . . . 15-17
15.1.5 Multi-Channel Mode . . . 15-19
15.1.6 Hall Sensor Mode . . . 15-21
15.1.6.1 Sampling of the Hall Pattern . . . 15-21
15.1.6.2 Brushless-DC Control . . . 15-22
15.1.7 Interrupt Generation . . . 15-25
15.1.8 Low Power Mode . . . 15-26
15.1.9 Module Suspend Control . . . 15-27

**Table of Contents** **Page**

15.1.10 Port Connection . . . 15-28
15.1.11 Synchronous Start of Timers . . . 15-32
15.2 Register Map . . . 15-33
15.3 Register Description . . . 15-36
15.3.1 System Registers . . . 15-38
15.3.2 Timer 12 – Related Registers . . . 15-41
15.3.3 Timer 13 – Related Registers . . . 15-52
15.3.4 Capture/Compare Control Registers . . . 15-56
15.3.5 Global Modulation Control Registers . . . 15-68
15.3.6 Multi-Channel Modulation Control Registers . . . 15-74
15.3.7 Interrupt Control Registers . . . 15-80

**16 Controller Area Network (MultiCAN) Controller . . . 16-1**
16.1 MultiCAN Kernel Functional Description . . . 16-4
16.1.1 Module Structure . . . 16-4
16.1.2 Clock Control . . . 16-7
16.1.3 CAN Node Control . . . 16-8
16.1.3.1 Bit Timing Unit . . . 16-8
16.1.3.2 Bitstream Processor . . . 16-9
16.1.3.3 Error Handling Unit . . . 16-10
16.1.3.4 CAN Frame Counter . . . 16-11
16.1.3.5 CAN Node Interrupts . . . 16-11
16.1.4 Message Object List Structure . . . 16-13
16.1.4.1 Basics . . . 16-13
16.1.4.2 List of Unallocated Elements . . . 16-14
16.1.4.3 Connection to the CAN Nodes . . . 16-14
16.1.4.4 List Command Panel . . . 16-15
16.1.5 CAN Node Analysis Features . . . 16-18
16.1.5.1 Analyze Mode . . . 16-18
16.1.5.2 Loop-Back Mode . . . 16-18
16.1.5.3 Bit Timing Analysis . . . 16-19
16.1.6 Message Acceptance Filtering . . . 16-21
16.1.6.1 Receive Acceptance Filtering . . . 16-21
16.1.6.2 Transmit Acceptance Filtering . . . 16-22
16.1.7 Message Postprocessing . . . 16-23
16.1.7.1 Message Interrupts . . . 16-23
16.1.7.2 Pending Messages . . . 16-25
16.1.8 Message Object Data Handling . . . 16-27
16.1.8.1 Frame Reception . . . 16-27
16.1.8.2 Frame Transmission . . . 16-30
16.1.9 Message Object Functionality . . . 16-33
16.1.9.1 Standard Message Object . . . 16-33
16.1.9.2 Single Data Transfer Mode . . . 16-33

**Table of Contents** **Page**

16.1.9.3 Single Transmit Trial . . . 16-33
16.1.9.4 Message Object FIFO Structure . . . 16-34
16.1.9.5 Receive FIFO . . . 16-36
16.1.9.6 Transmit FIFO . . . 16-37
16.1.9.7 Gateway Mode . . . 16-38
16.1.9.8 Foreign Remote Requests . . . 16-40
16.1.10 Access Mediator . . . 16-41
16.1.11 Port Control . . . 16-43
16.1.12 Low Power Mode . . . 16-44
16.2 Registers Description . . . 16-45
16.2.1 Global Module Registers . . . 16-48
16.2.2 CAN Node Registers . . . 16-59
16.2.3 Message Object Registers . . . 16-76
16.2.4 MultiCAN Access Mediator Register . . . 16-97

**17 Analog-to-Digital Converter** . . . 17-1
17.1 Structure Overview . . . 17-2
17.2 Clocking Scheme . . . 17-3
17.2.1 Conversion Timing . . . 17-4
17.3 Low Power Mode . . . 17-7
17.4 Functional Description . . . 17-8
17.4.1 Request Source Arbiter . . . 17-9
17.4.2 Conversion Start Modes . . . 17-10
17.4.3 Channel Control . . . 17-10
17.4.4 Sequential Request Source . . . 17-11
17.4.4.1 Overview . . . 17-11
17.4.4.2 Request Source Control . . . 17-13
17.4.5 Parallel Request Source . . . 17-14
17.4.5.1 Overview . . . 17-14
17.4.5.2 Request Source Control . . . 17-15
17.4.5.3 External Trigger . . . 17-16
17.4.5.4 Software Control . . . 17-16
17.4.5.5 Autoscan . . . 17-16
17.4.6 Wait-for-Read Mode . . . 17-17
17.4.7 Result Generation . . . 17-17
17.4.7.1 Overview . . . 17-17
17.4.7.2 Limit Checking . . . 17-19
17.4.7.3 Data Reduction Filter . . . 17-20
17.4.7.4 Result Register View . . . 17-21
17.4.8 Interrupts . . . 17-23
17.4.8.1 Event Interrupts . . . 17-24
17.4.8.2 Channel Interrupts . . . 17-25
17.4.9 External Trigger Inputs . . . 17-27

**Table of Contents** **Page**

17.5 ADC Module Initialization Sequence . . . . . 17-29
17.6 Register Map . . . . . 17-31
17.7 Register Description . . . . . 17-34
17.7.1 General Function Registers . . . . . 17-34
17.7.2 Priority and Arbitration Register . . . . . 17-37
17.7.3 External Trigger Control Register . . . . . 17-39
17.7.4 Channel Control Registers . . . . . 17-40
17.7.5 Input Class Register . . . . . 17-41
17.7.6 Sequential Source Registers . . . . . 17-42
17.7.7 Parallel Source Registers . . . . . 17-50
17.7.8 Result Registers . . . . . 17-54
17.7.9 Interrupt Registers . . . . . 17-60

**18 On-Chip Debug Support** . . . . . 18-1
18.1 Features . . . . . 18-1
18.2 Functional Description . . . . . 18-1
18.3 Debugging . . . . . 18-2
18.3.1 Debug Events . . . . . 18-3
18.3.1.1 Hardware Breakpoints . . . . . 18-3
18.3.1.2 Software Breakpoints . . . . . 18-5
18.3.1.3 External Breaks . . . . . 18-5
18.3.1.4 NMI-mode priority over Debug-mode . . . . . 18-5
18.3.2 Debug Actions . . . . . 18-6
18.3.2.1 Call the Monitor Program . . . . . 18-6
18.3.2.2 Activate the MBC pin . . . . . 18-6
18.4 Debug Suspend Control . . . . . 18-7
18.5 Register Description . . . . . 18-8
18.5.1 Monitor Work Register 2 . . . . . 18-9
18.5.2 Input Select Registers . . . . . 18-10
18.6 JTAG ID . . . . . 18-11

**19 Bootstrap Loader** . . . . . 19-1
19.1 UART and LIN BSL Modes . . . . . 19-2
19.1.1 Communication Protocol . . . . . 19-3
19.1.1.1 UART Transfer Block Structure . . . . . 19-3
19.1.1.2 LIN Transfer Block Structure . . . . . 19-4
19.1.1.3 Response Code to the Host . . . . . 19-6
19.1.2 Bootstrap Loader via UART . . . . . 19-9
19.1.2.1 Communication Structure . . . . . 19-10
19.1.2.2 The Selection of Modes . . . . . 19-10
19.1.2.3 The Activation of Modes 0 and 2 . . . . . 19-11
19.1.2.4 The Activation of Modes 1, 3 and 15 . . . . . 19-12
19.1.2.5 The Activation of Mode 6 . . . . . 19-13

19.1.2.6 The Activation of Mode 10 . . . . . 19-13
19.1.2.7 The Activation of Mode 16 . . . . . 19-14
19.1.3 Bootstrap Loader via LIN . . . . . 19-15
19.1.3.1 Communication Structure . . . . . 19-16
19.1.3.2 The Selection of Modes . . . . . 19-18
19.1.3.3 The Activation of Modes 0, 2 and 8 . . . . . 19-18
19.1.3.4 The Activation of Modes 1, 3 and 9 . . . . . 19-20
19.1.3.5 The Activation of Mode 6 . . . . . 19-20
19.1.3.6 The Activation of Mode 10 . . . . . 19-21
19.1.3.7 The Activation of Mode 16 . . . . . 19-21
19.1.3.8 LIN Response Protocol to the Host . . . . . 19-21
19.1.3.9 Fast LIN BSL . . . . . 19-22
19.1.3.10 User Defined Parameter for LIN BSL . . . . . 19-23
19.2 MultiCAN BSL Mode . . . . . 19-24
19.2.1 Communication protocol . . . . . 19-24
19.2.2 CAN Message Object definition . . . . . 19-25
19.2.3 User Defined Parameter for MultiCAN BSL . . . . . 19-27
19.3 Alternate BSL Mode . . . . . 19-28

**20 Index** . . . . . 20-1
20.1 Keyword Index . . . . . 20-1
20.2 Register Index . . . . . 21-9

# 1 Introduction

The XC878 is a member of the high-performance XC800 family of 8-bit microcontrollers. It is based on the XC800 Core that is compatible with the industry standard 8051 processor.

The XC878 features both a CAN controller and LIN support integrated on a single chip to provide advance networking capabilities. The on-chip CAN module reduces the CPU load by performing most of the functions required by the networking protocol (masking, filtering and buffering of CAN frames).

The XC878 is equipped with embedded Flash memory to offer high flexibility in development and ramp-up. The XC878 memory protection strategy features read-out protection of user intellectual property (IP), along with Flash program and erase protection to prevent data corruption.

The Flash architecture supports In-Application Programming (IAP), allowing user program to modify Flash contents during program execution. In-System Programming (ISP) is available through the Boot ROM-based BootStrap Loader (BSL), enabling convenient programming and erasing of the embedded Flash via an external host (e.g., personal computer).

Other key features include a Capture/Compare Unit 6 (CCU6) for the generation of pulse width modulated signal with special modes for motor control; a 10-bit Analog-to-Digital Converter (ADC) with extended functionalities such as autoscan and result accumulation for anti-aliasing filtering or for averaging; a Multiplication/Division Unit (MDU) to support the XC800 Core in math-intensive computations in advanced motor control like Field Oriented Control; a CORDIC (COrdinate Rotation DIgital Computer) Coprocessor performing high-speed computation of trigonometric, linear or hyperbolic functions for vector rotation and transformations; and an On-Chip Debug Support (OCDS) unit for software development and debugging of XC800-based systems.

The XC878 also features an on-chip oscillator and an integrated voltage regulator to allow a single voltage supply of 3.3 or 5.0 V. For low power applications, various power saving modes are available for selection by the user. Control of the numerous on-chip peripheral functionalities is achieved by extending the Special Function Register (SFR) address range with an intelligent paging mechanism optimized for interrupt handling.

**Figure 1-1** shows the functional units of the XC878.

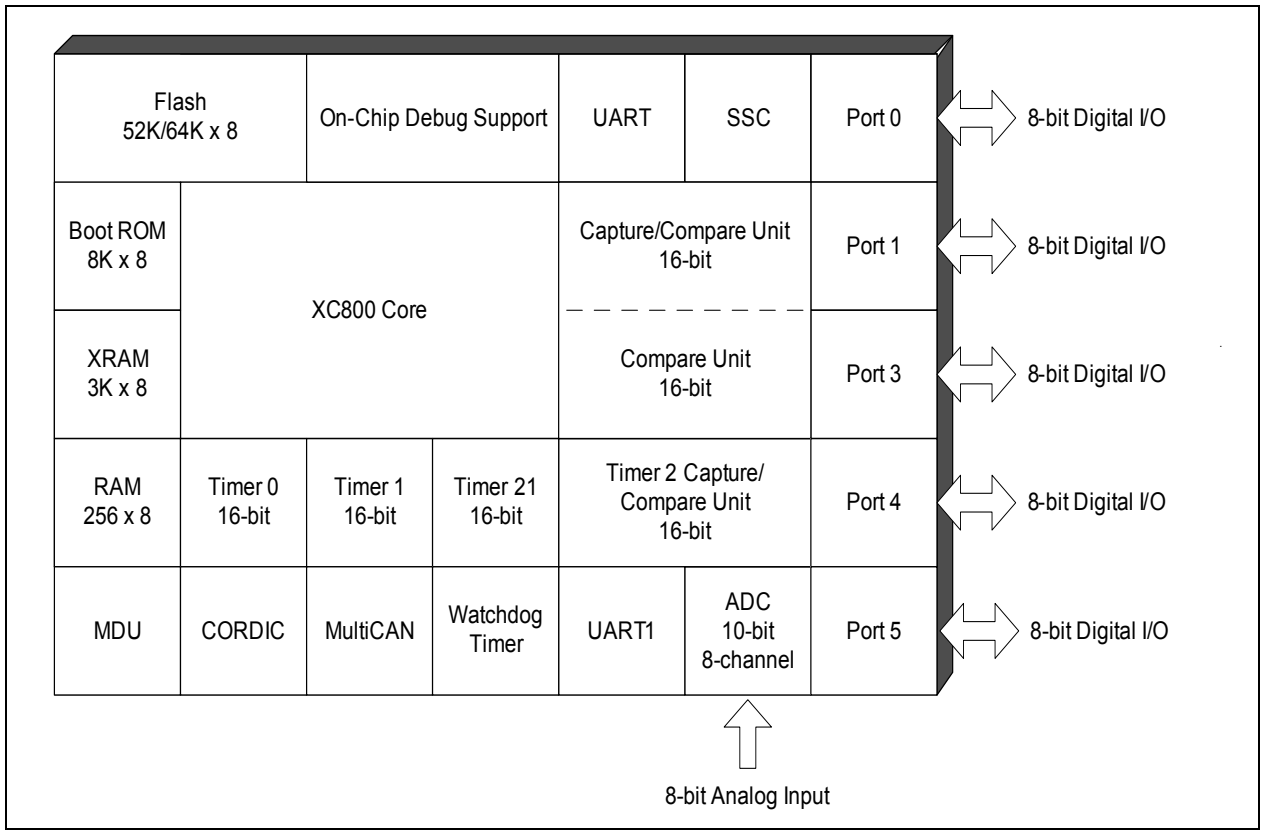



**Figure 1-1 XC878 Functional Units**

The XC878 product family features devices with different configurations and program memory sizes, power supply voltage, temperature and quality profiles (Automotive or Industrial), to offer cost-effective solutions for different application requirements.

The list of XC878 device configurations are summarized in **Table 1-1**. The type of package available is the TQFP-64.

**Table 1-1 Device Configuration**

| Device Name | CAN Module | LIN BSL Support | MDU Module |
|---|---|---|---|
| XC878 | No | No | No |
| XC878C | Yes | No | No |
| XC878L | No | Yes | No |
| XC878M | No | No | Yes |
| XC878CM | Yes | No | Yes |

From these 5 different combinations of configuration, each are further made available in many sales types, which are grouped according to device type, program memory sizes, power supply voltage, temperature and quality profiles (Automotive or Industrial), as shown in **Table 1-2**.

**Table 1-2 Device Profile**

| Sales Type | Device Type | Program Memory (Kbytes) | Power Supply (V) | Temp-erature Profile (°C) | Quality Profile |
|---|---|---|---|---|---|
| SAF-XC878-13FFI 5V | Flash | 52 | 5.0 | -40 to 85 | Industrial |
| SAF-XC878M-13FFI 5V | Flash | 52 | 5.0 | -40 to 85 | Industrial |
| SAF-XC878CM-13FFI 5V | Flash | 52 | 5.0 | -40 to 85 | Industrial |
| SAF-XC878-16FFI 5V | Flash | 64 | 5.0 | -40 to 85 | Industrial |
| SAF-XC878M-16FFI 5V | Flash | 64 | 5.0 | -40 to 85 | Industrial |
| SAF-XC878CM-16FFI 5V | Flash | 64 | 5.0 | -40 to 85 | Industrial |
| SAF-XC878-13FFI 3V3 | Flash | 52 | 3.3 | -40 to 85 | Industrial |
| SAF-XC878M-13FFI 3V3 | Flash | 52 | 3.3 | -40 to 85 | Industrial |
| SAF-XC878CM-13FFI 3V3 | Flash | 52 | 3.3 | -40 to 85 | Industrial |
| SAF-XC878-16FFI 3V3 | Flash | 64 | 3.3 | -40 to 85 | Industrial |
| SAF-XC878M-16FFI 3V3 | Flash | 64 | 3.3 | -40 to 85 | Industrial |
| SAF-XC878CM-16FFI 3V3 | Flash | 64 | 3.3 | -40 to 85 | Industrial |
| SAX-XC878-13FFA 5V | Flash | 52 | 5.0 | -40 to 105 | Automotive |
| SAX-XC878L-13FFA 5V | Flash | 52 | 5.0 | -40 to 105 | Automotive |
| SAX-XC878C-13FFA 5V | Flash | 52 | 5.0 | -40 to 105 | Automotive |
| SAX-XC878CM-13FFA 5V | Flash | 52 | 5.0 | -40 to 105 | Automotive |
| SAX-XC878-16FFA 5V | Flash | 64 | 5.0 | -40 to 105 | Automotive |
| SAX-XC878L-16FFA 5V | Flash | 64 | 5.0 | -40 to 105 | Automotive |
| SAX-XC878C-16FFA 5V | Flash | 64 | 5.0 | -40 to 105 | Automotive |
| SAX-XC878CM-16FFA 5V | Flash | 64 | 5.0 | -40 to 105 | Automotive |
| SAX-XC878-16FFA 3V3 | Flash | 64 | 3.3 | -40 to 105 | Automotive |
| SAX-XC878L-16FFA 3V3 | Flash | 64 | 3.3 | -40 to 105 | Automotive |

The term “XC878” in this document refers to all devices of the XC878 family unless stated otherwise.

## 1.1 Feature Summary

The following list summarizes the main features of the XC878:

- High-performance XC800 Core
  - compatible with standard 8051 processor
  - two clocks per machine cycle architecture (for memory access without wait state)
  - two data pointers
- On-chip memory
  - 8 Kbytes of Boot ROM
  - 256 bytes of RAM
  - 3 Kbytes of XRAM
  - 52/64 Kbytes of Flash for program code and data; (includes memory protection strategy)
- I/O port supply at 3.3 or 5.0 V and core logic supply at 2.5 V (generated by embedded voltage regulator)
- Power-on reset generation
- Brownout detection for core logic supply
- On-chip OSC and PLL for clock generation
  - Loss-of-Clock detection
- Power saving modes
  - slow-down mode
  - idle mode
  - power-down mode with wake-up capability via RXD or EXINT0
  - clock gating control to each peripheral
- Programmable 16-bit Watchdog Timer (WDT)
- Five ports
  - Up to 40 pins as digital I/O
  - 8 dedicated analog inputs used as A/D converter input
- 8-channel, 10-bit ADC
- Four 16-bit timers
  - Timer 0 and Timer 1 (T0 and T1)
  - Timer 2 and Timer 21 (T2 and T21)
- Multiplication/Division Unit for arithmetic calculation (MDU)
- CORDIC Coprocessor for computation of trigonometric, hyperbolic and linear functions
- MultiCAN with 2 nodes, 32 message objects
- Two Capture/compare units
  - Capture/compare unit 6 for PWM signal generation (CCU6)
  - Timer 2 Capture/compare unit for vaious digital signal generation (T2CCU)
- Two full-duplex serial interfaces (UART and UART1)
- Synchronous serial channel (SSC)
- On-chip debug support
  - 1 Kbyte of monitor ROM (part of the 8-Kbyte Boot ROM)

  - 64 bytes of monitor RAM
- PG-TQFP-64 pin package
- Temperature range $T_A$:
  - SAF (-40 to 85 °C)
  - SAX (-40 to 105 °C)

The block diagram of the XC878 is shown in **Figure 1-2**.

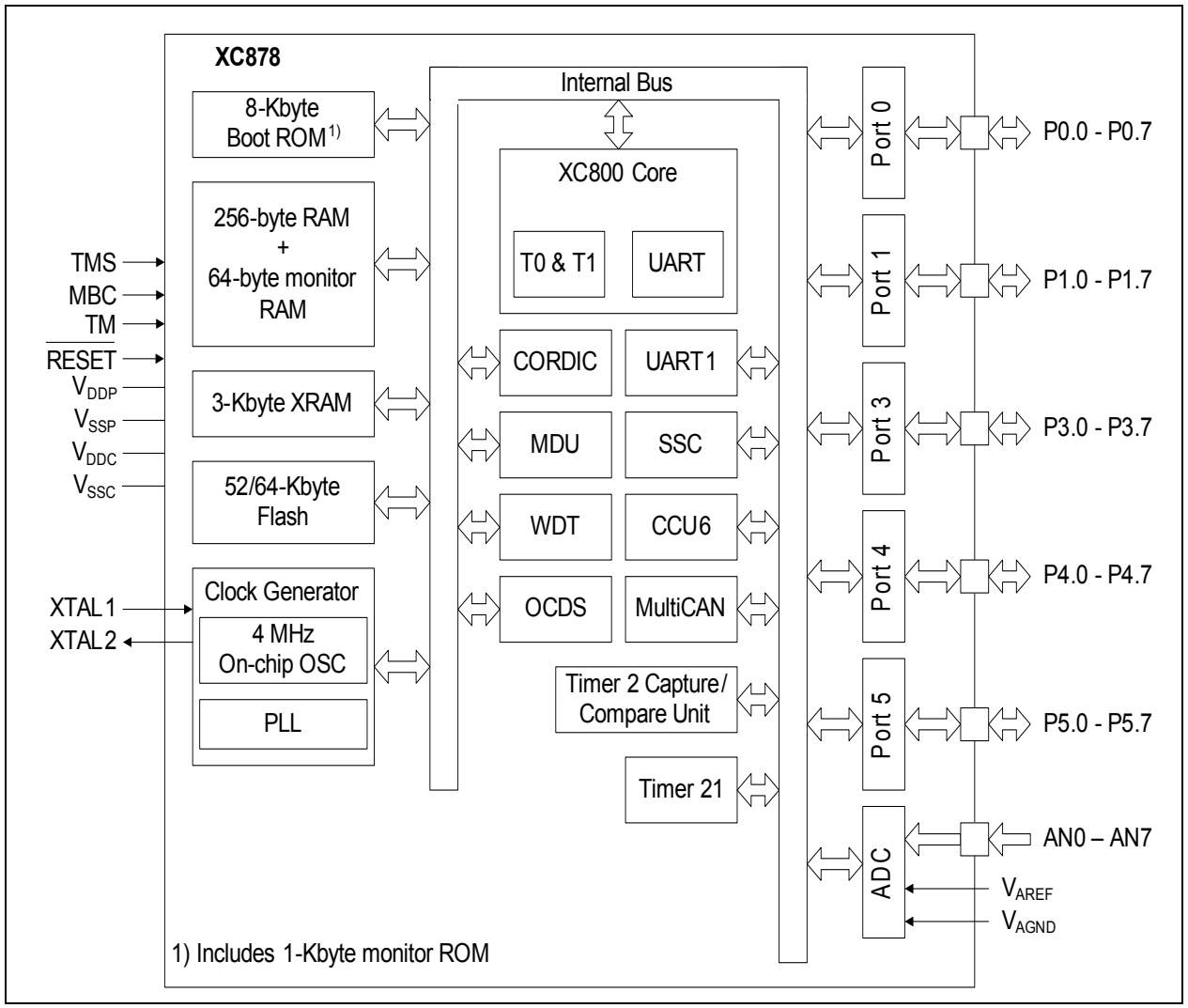



**Figure 1-2 XC878 Block Diagram**

## 1.2 Pin Configuration

The pin configuration of the XC878, which is based on the PG-TQFP-64 package, is shown in **Figure 1-3**.

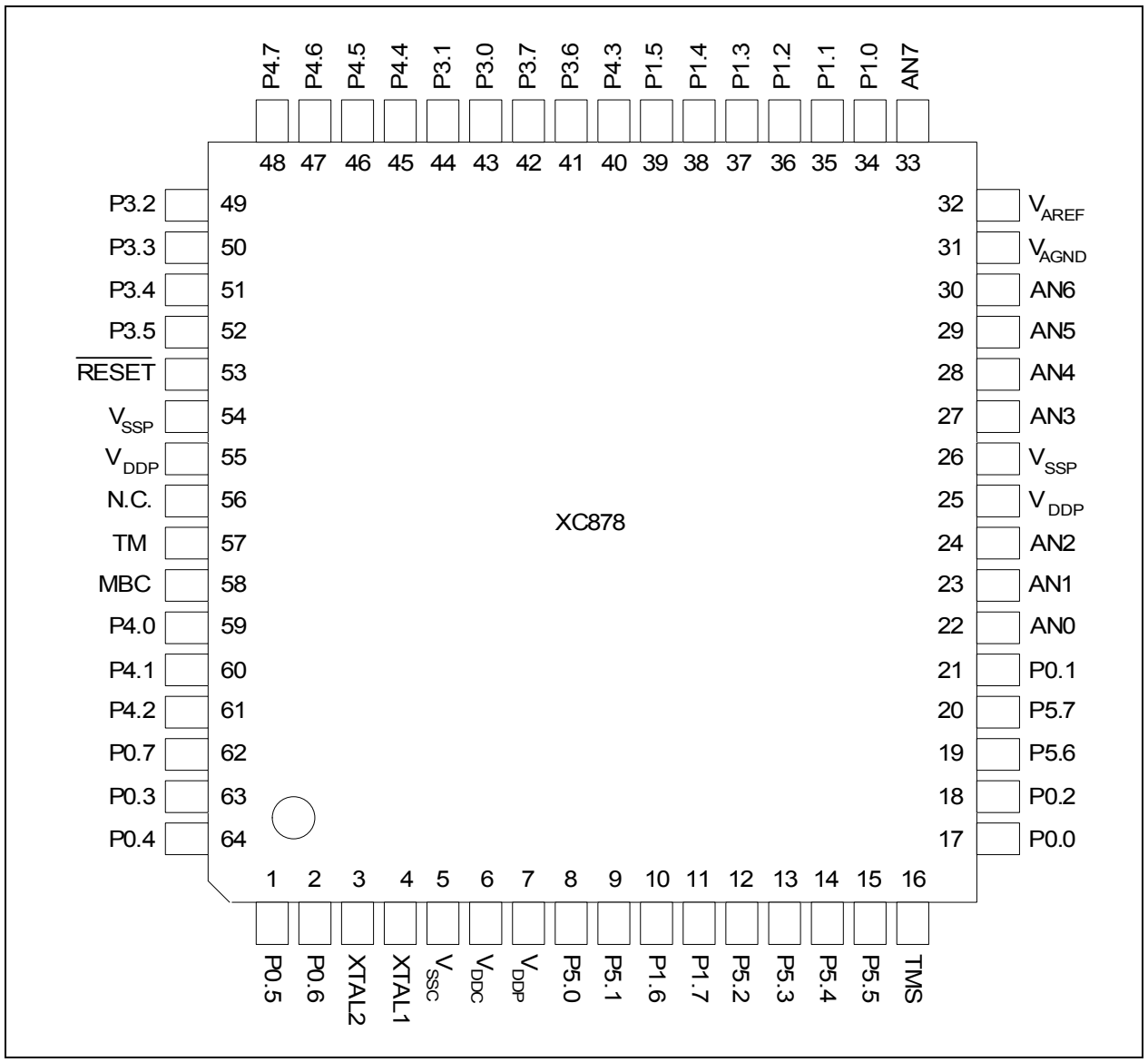



**Figure 1-3 XC878 Pin Configuration, PG-TQFP-64 Package (top view)**

## 1.3 Pin Definitions and Functions

After reset, all pins are configured as input with one of the following:

- Pull-up device enabled only (PU)
- Pull-down device enabled only (PD)
- High impedance with both pull-up and pull-down devices disabled (Hi-Z)

The functions and default states of the XC878 external pins are provided in **Table 1.3**.

**Table 1-3 Pin Definitions and Functions**

| Symbol | Pin Number (TQFP-64) | Type | Reset State | Function |
|---|---|---|---|---|
| **P0** | | I/O | | **Port 0**<br>Port 0 is an 8-bit bidirectional general purpose I/O port. It can be used as alternate functions for the JTAG, CCU6, UART, UART1, T2CCU, Timer 21, MultiCAN and SSC. |
| P0.0 | 17 | | Hi-Z | TCK_0 JTAG Clock Input<br>T12HR_1 CCU6 Timer 12 Hardware Run Input<br>CC61_1 Input/Output of Capture/Compare channel 1<br>CLKOUT_0 Clock Output<br>RXDO_1 UART Transmit Data Output |
| P0.1 | 21 | | Hi-Z | TDI_0 JTAG Serial Data Input<br>T13HR_1 CCU6 Timer 13 Hardware Run Input<br>RXD_1 UART Receive Data Input<br>RXDC1_0 MultiCAN Node 1 Receiver Input<br>COUT61_1 Output of Capture/Compare channel 1<br>EXF2_1 Timer 2 External Flag Output |
| P0.2 | 18 | | PU | $\overline{\text{CTRAP}}$_2 CCU6 Trap Input<br>TDO_0 JTAG Serial Data Output<br>TXD_1 UART Transmit Data Output/Clock Output<br>TXDC1_0 MultiCAN Node 1 Transmitter Output |
| P0.3 | 63 | | Hi-Z | SCK_1 SSC Clock Input/Output<br>COUT63_1 Output of Capture/Compare channel 3<br>RXDO1_0 UART1 Transmit Data Output |

**Table 1-3 Pin Definitions and Functions** (cont'd)

| Symbol | Pin Number (TQFP-64) | Type | Reset State | Function | |
|---|---|---|---|---|---|
| P0.4 | 64 | | Hi-Z | MTSR_1 | SSC Master Transmit Output/ Slave Receive Input |
| | | | | CC62_1 | Input/Output of Capture/Compare channel 2 |
| | | | | TXD1_0 | UART1 Transmit Data Output/Clock Output |
| P0.5 | 1 | | Hi-Z | MRST_1 | SSC Master Receive Input/Slave Transmit Output |
| | | | | EXINT0_0 | External Interrupt Input 0 |
| | | | | T2EX1_1 | Timer 21 External Trigger Input |
| | | | | RXD1_0 | UART1 Receive Data Input |
| | | | | COUT62_1 | Output of Capture/Compare channel 2 |
| P0.6 | 2 | | PU | T2CC4_1 | Compare Output Channel 4 |
| P0.7 | 62 | | PU | CLKOUT_1 | Clock Output |
| | | | | T2CC5_1 | Compare Output Channel 5 |

**Table 1-3 Pin Definitions and Functions** (cont'd)

| Symbol | Pin Number (TQFP-64) | Type | Reset State | Function | |
|---|---|---|---|---|---|
| **P1** | | I/O | | **Port 1**<br>Port 1 is an 8-bit bidirectional general purpose I/O port. It can be used as alternate functions for the JTAG, CCU6, UART, Timer 0, Timer 1, T2CCU, Timer 21, MultiCAN and SSC. | |
| P1.0 | 34 | | PU | RXD_0<br>T2EX_0<br>RXDC0_0 | UART Receive Data Input<br>Timer 2 External Trigger Input<br>MultiCAN Node 0 Receiver Input |
| P1.1 | 35 | | PU | EXINT3_0<br>T0_1<br>TXD_0<br><br>TXDC0_0 | External Interrupt Input 3<br>Timer 0 Input<br>UART Transmit Data Output/Clock Output<br>MultiCAN Node 0 Transmitter Output |
| P1.2 | 36 | | PU | SCK_0 | SSC Clock Input/Output |
| P1.3 | 37 | | PU | MTSR_0<br><br>SCK_2<br>TXDC1_3 | SSC Master Transmit Output/Slave Receive Input<br>SSC Clock Input/Output<br>MultiCAN Node 1 Transmitter Output |
| P1.4 | 38 | | PU | MRST_0<br><br>EXINT0_1<br>RXDC1_3<br>MTSR_2 | SSC Master Receive Input/ Slave Transmit Output<br>External Interrupt Input 0<br>MultiCAN Node 1 Receiver Input<br>SSC Master Transmit Output/Slave Receive Input |
| P1.5 | 39 | | PU | CCPOS0_1<br>EXINT5_0<br>T1_1<br>MRST_2<br><br>EXF2_0<br>RXDO_0 | CCU6 Hall Input 0<br>External Interrupt Input 5<br>Timer 1 Input<br>SSC Master Receive Input/ Slave Transmit Output<br>Timer 2 External Flag Output<br>UART Transmit Data Output |

**Table 1-3 Pin Definitions and Functions** (cont'd)

| Symbol | Pin Number (TQFP-64) | Type | Reset State | Function |
|---|---|---|---|---|
| P1.6 | 10 | | PU | CCPOS1_1 CCU6 Hall Input 1<br>T12HR_0 CCU6 Timer 12 Hardware Run Input<br>EXINT6_0 External Interrupt Input 6<br>RXDC0_2 MultiCAN Node 0 Receiver Input<br>T21_1 Timer 21 Input |
| P1.7 | 11 | | PU | CCPOS2_1 CCU6 Hall Input 2<br>T13HR_0 CCU6 Timer 13 Hardware Run Input<br>T2_1 Timer 2 Input<br>TXDC0_2 MultiCAN Node 0 Transmitter Output<br>P1.5 and P1.6 can be used as a software chip select output for the SSC. |

**Table 1-3 Pin Definitions and Functions** (cont'd)

| Symbol | Pin Number (TQFP-64) | Type | Reset State | Function |
|---|---|---|---|---|
| **P3** | | I/O | | **Port 3**<br>Port 3 is an 8-bit bidirectional general purpose I/O port. It can be used as alternate functions for CCU6, UART1, T2CCU, Timer 21 and MultiCAN. |
| P3.0 | 43 | | Hi-Z | CCPOS1_2 CCU6 Hall Input 1<br>CC60_0 Input/Output of Capture/Compare channel 0<br>RXDO1_1 UART1 Transmit Data Output<br>T2CC0_1/ EXINT3_2 External Interrupt Input 3/T2CCU Capture/Compare Channel 0 |
| P3.1 | 44 | | Hi-Z | CCPOS0_2 CCU6 Hall Input 0<br>CC61_2 Input/Output of Capture/Compare channel 1<br>COUT60_0 Output of Capture/Compare channel 0<br>TXD1_1 UART1 Transmit Data Output/Clock Output |
| P3.2 | 49 | | Hi-Z | CCPOS2_2 CCU6 Hall Input 2<br>RXDC1_1 MultiCAN Node 1 Receiver Input<br>RXD1_1 UART1 Receive Data Input<br>CC61_0 Input/Output of Capture/Compare channel 1<br>T2CC1_1/ EXINT4_2 External Interrupt Input 4/T2CCU Capture/Compare Channel 1 |
| P3.3 | 50 | | Hi-Z | COUT61_0 Output of Capture/Compare channel 1<br>TXDC1_1 MultiCAN Node 1 Transmitter Output<br>T2CC2_1/ EXINT5_2 External Interrupt Input 5/T2CCU Capture/Compare Channel 2 |
| P3.4 | 51 | | Hi-Z | CC62_0 Input/Output of Capture/Compare channel 2<br>RXDC0_1 MultiCAN Node 0 Receiver Input<br>T2EX1_0 Timer 21 External Trigger Input<br>T2CC3_1/ EXINT6_3 External Interrupt Input 6/T2CCU Capture/Compare Channel 3 |

**Table 1-3 Pin Definitions and Functions** (cont'd)

| Symbol | Pin Number (TQFP-64) | Type | Reset State | Function | |
|---|---|---|---|---|---|
| P3.5 | 52 | | Hi-Z | COUT62_0 | Output of Capture/Compare channel 2 |
| | | | | EXF21_0 | Timer 21 External Flag Output |
| | | | | TXDC0_1 | MultiCAN Node 0 Transmitter Output |
| P3.6 | 41 | | PU | $\overline{\text{CTRAP_0}}$ | CCU6 Trap Input |
| P3.7 | 42 | | Hi-Z | EXINT4_0 | External Interrupt Input 4 |
| | | | | COUT63_0 | Output of Capture/Compare channel 3 |

**Table 1-3 Pin Definitions and Functions** (cont'd)

| Symbol | Pin Number (TQFP-64) | Type | Reset State | Function | |
|---|---|---|---|---|---|
| **P4** | | I/O | | **Port 4**<br>Port 4 is an 8-bit bidirectional general purpose I/O port. It can be used as alternate functions for CCU6, Timer 0, Timer 1, T2CCU, Timer 21 and MultiCAN. | |
| P4.0 | 59 | | Hi-Z | RXDC0_3<br>CC60_1<br><br>T2CC0_0/<br>EXINT3_1 | MultiCAN Node 0 Receiver Input<br>Output of Capture/Compare channel 0<br>External Interrupt Input 3/T2CCU Capture/Compare Channel 0 |
| P4.1 | 60 | | Hi-Z | TXDC0_3<br><br>COUT60_1<br><br>T2CC1_0/<br>EXINT4_1 | MultiCAN Node 0 Transmitter Output<br>Output of Capture/Compare channel 0<br>External Interrupt Input 4/T2CCU Capture/Compare Channel 1 |
| P4.2 | 61 | | PU | EXINT6_1<br>T21_0 | External Interrupt Input 6<br>Timer 21 Input |
| P4.3 | 40 | | Hi-Z | T2EX_1<br>EXF21_1<br>COUT63_2 | Timer 2 External Trigger Input<br>Timer 21 External Flag Output<br>Output of Capture/Compare channel 3 |
| P4.4 | 45 | | Hi-Z | CCPOS0_3<br>T0_0<br>CC61_4<br><br>T2CC2_0/<br>EXINT5_1 | CCU6 Hall Input 0<br>Timer 0 Input<br>Output of Capture/Compare channel 1<br>External Interrupt Input 5/T2CCU Capture/Compare Channel 2 |
| P4.5 | 46 | | Hi-Z | CCPOS1_3<br>T1_0<br>COUT61_2<br><br>T2CC3_0/<br>EXINT6_2 | CCU6 Hall Input 1<br>Timer 1 Input<br>Output of Capture/Compare channel 1<br>External Interrupt Input 6/T2CCU Capture/Compare Channel 3 |

**Table 1-3 Pin Definitions and Functions** (cont'd)

| Symbol | Pin Number (TQFP-64) | Type | Reset State | Function | |
|---|---|---|---|---|---|
| P4.6 | 47 | | Hi-Z | CCPOS2_3<br>T2_0<br>CC62_2<br><br>T2CC4_0 | CCU6 Hall Input 2<br>Timer 2 Input<br>Output of Capture/Compare channel 2<br>Compare Output Channel 4 |
| P4.7 | 48 | | Hi-Z | $\overline{\text{CTRAP}}$_3<br>COUT62_2<br><br>T2CC5_0 | CCU6 Trap Input<br>Output of Capture/Compare channel 2<br>Compare Output Channel 5 |

**Table 1-3 Pin Definitions and Functions** (cont'd)

| Symbol | Pin Number (TQFP-64) | Type | Reset State | Function | |
|---|---|---|---|---|---|
| **P5** | | I/O | | **Port 5**<br>Port 5 is an 8-bit bidirectional general purpose I/O port. It can be used as alternate functions for UART, UART1, T2CCU and JTAG. | |
| P5.0 | 8 | | PU | EXINT1_1 | External Interrupt Input 1 |
| P5.1 | 9 | | PU | EXINT2_1 | External Interrupt Input 2 |
| P5.2 | 12 | | PU | RXD_2<br>T2CC2_2/<br>EXINT5_3 | UART Receive Data Input<br>External Interrupt Input 5/T2CCU Capture/Compare Channel 2 |
| P5.3 | 13 | | PU | CCPOS0_0<br>EXINT1_0<br>T12HR_2<br>CC61_3<br>TXD_2<br>T2CC5_2 | CCU6 Hall Input 0<br>External Interrupt Input 1<br>CCU6 Timer 12 Hardware Run Input<br>Input of Capture/Compare channel 1<br>UART Transmit Data Output/Clock Output<br>Compare Output Channel 5 |
| P5.4 | 14 | | PU | CCPOS1_0<br>EXINT2_0<br>T13HR_2<br>CC62_3<br>RXDO_2<br>T2CC4_2 | CCU6 Hall Input 1<br>External Interrupt Input 2<br>CCU6 Timer 13 Hardware Run Input<br>Input of Capture/Compare channel 2<br>UART Transmit Data Output<br>Compare Output Channel 4 |
| P5.5 | 15 | | PU | CCPOS2_0<br>CTRAP_1<br>CC60_3<br>TDO_1<br>TXD1_2<br>T2CC0_2/<br>EXINT3_3 | CCU6 Hall Input 2<br>CCU6 Trap Input<br>Input of Capture/Compare channel 0<br>JTAG Serial Data Output<br>UART1 Transmit Data Output/ Clock Output<br>External Interrupt Input 3/T2CCU Capture/Compare Channel 0 |

**Table 1-3 Pin Definitions and Functions** (cont'd)

| Symbol | Pin Number (TQFP-64) | Type | Reset State | Function | |
|---|---|---|---|---|---|
| P5.6 | 19 | | PU | TCK_1<br>RXDO1_2<br>T2CC1_2/<br>EXINT4_3 | JTAG Clock Input<br>UART1 Transmit Data Output<br>External Interrupt Input 4/T2CCU<br>Capture/Compare Channel 1 |
| P5.7 | 20 | | PU | TDI_1<br>RXD1_2<br>T2CC3_2/<br>EXINT6_4 | JTAG Serial Data Input<br>UART1 Receive Data Input<br>External Interrupt Input 6/T2CCU<br>Capture/Compare Channel 3 |

**Table 1-3 Pin Definitions and Functions** (cont'd)

| Symbol | Pin Number (TQFP-64) | Type | Reset State | Function |
|---|---|---|---|---|
| $V_{DDP}$ | 7, 25, 55 | – | – | **I/O Port Supply (3.3 or 5.0 V)**<br>Also used by EVR and analog modules. All pins must be connected. |
| $V_{SSP}$ | 26, 54 | – | – | **I/O Ground**<br>All pins must be connected. |
| $V_{DDC}$ | 6 | – | – | **Core Supply Monitor (2.5 V)** |
| $V_{SSC}$ | 5 | – | – | **Core Supply Ground** |
| $V_{AREF}$ | 32 | – | – | **ADC Reference Voltage** |
| $V_{AGND}$ | 31 | – | – | **ADC Reference Ground** |
| **AN0** | 22 | I | Hi-Z | **Analog Input 0** |
| **AN1** | 23 | I | Hi-Z | **Analog Input 1** |
| **AN2** | 24 | I | Hi-Z | **Analog Input 2** |
| **AN3** | 27 | I | Hi-Z | **Analog Input 3** |
| **AN4** | 28 | I | Hi-Z | **Analog Input 4** |
| **AN5** | 29 | I | Hi-Z | **Analog Input 5** |
| **AN6** | 30 | I | Hi-Z | **Analog Input 6** |
| **AN7** | 33 | I | Hi-Z | **Analog Input 7** |
| **XTAL1** | 4 | I | Hi-Z | **External Oscillator Input**<br>**(Feedback resistor required, normally NC)** |
| **XTAL2** | 3 | O | Hi-Z | **External Oscillator Output**<br>**(Feedback resistor required, normally NC)** |
| **TMS** | 16 | I | PD | **JTAG Test Mode Select** |
| **$\overline{\text{RESET}}$** | 53 | I | PU | **Reset Input** |
| **MBC** | 58 | I | PU | **Monitor & BootStrap Loader Control** |
| **TM** | 57 | – | – | **Test Mode**<br>**(External pull down device required)** |
| **NC** | 56 | – | – | **No Connection** |

## 1.4 Chip Identification Number

Each device variant of XC878 is assigned an unique chip identification number to allow easy identification of one device variant from the others. The differentiation is based on the product, variant type and device step information.

Two methods are provided to read a device variant's chip identification number:

- In-application subroutine, see **Chapter 4.7.3**;
- Bootstrap loader (BSL) mode A, see **Chapter 19.1.2.6** or **Chapter 19.1.3.6**.

## 1.5 Text Conventions

This document uses the following text conventions for named components of the XC878:

- Functional units of the XC878 are shown in upper case. For example: “The SSC can be used to communicate with shift registers.”
- Pins using negative logic are indicated by an overbar. For example: “A reset input pin $\overline{\text{RESET}}$ is provided for the hardware reset.”
- Bit fields and bits in registers are generally referenced as “Register name.Bit field” or “Register name.Bit”. Most of the register names contain a module name prefix, separated by an underscore character “_” from the actual register name. In the example of “SSC_CON”, “SSC” is the module name prefix, and “CON” is the actual register name).
- Function of the shaded bits are described in other chapters of the UM.
- Variables that are used to represent sets of processing units or registers appear in mixed-case type. For example, the register name “CC6xR” refers to multiple “CC6xR” registers with the variable x (x = 0, 1, 2). The bounds of the variables are always specified where the register expression is first used (e.g., “x = 0 - 2”), and is repeated as needed.
- The default radix is decimal. Hexadecimal constants have a suffix with the subscript letter “H” (e.g., $C0_H$). Binary constants have a suffix with the subscript letter “B” (e.g., $11_B$).
- When the extents of register fields, groups of signals, or groups of pins are collectively named in the body of the document, they are represented as “NAME[A:B]”, which defines a range, from B to A, for the named group. Individual bits, signals, or pins are represented as “NAME[C]”, with the range of the variable C provided in the text (e.g., CFG[2:0] and TOS[0]).
- Units are abbreviated as follows:
  - **MHz** = Megahertz
  - **µs** = Microseconds
  - **kBaud, kbit** = 1000 characters/bits per second
  - **MBaud, Mbit** = 1,000,000 characters/bits per second
  - **Kbyte** = 1024 bytes of memory
  - **Mbyte** = 1,048,576 bytes of memory

    In general, the k prefix scales a unit by 1000 whereas the K prefix scales a unit by 1024. Hence, the Kbyte unit scales the expression preceding it by 1024. The kBaud unit scales the expression preceding it by 1000. The M prefix scales by 1,000,000 or 1048576, and µ scales by 0.000001. For example, 1 Kbyte is 1024 bytes, 1 Mbyte is 1024 × 1024 bytes, 1 kBaud/kbit are 1000 characters/bits per second, 1 MBaud/Mbit are 1,000,000 characters/bits per second, and 1 MHz is 1,000,000 Hz.
- Data format quantities are defined as follows:
  - **Byte** = 8-bit quantity

## 1.6 Reserved, Undefined and Unimplemented Terminology

In tables where register bit fields are defined, the following conventions are used to indicate undefined and unimplemented function. Further, types of bits and bit fields are defined using the abbreviations shown in **Table 1-4**.

**Table 1-4 Bit Function Terminology**

| Function of Bits | Description |
|---|---|
| **Unimplemented** | Register bit fields named “0” indicate unimplemented functions with the following behavior.<br>Reading these bit fields returns 0.<br>Writing to these bit fields has no effect.<br>These bit fields are reserved. When writing, software should always set such bit fields to 0 in order to preserve compatibility with future products. Setting the bit fields to 1 may lead to unpredictable results. |
| **Undefined** | Certain bit combinations in a bit field can be labeled “Reserved”, indicating that the behavior of the XC878 is undefined for that combination of bits. Setting the register to undefined bit combinations may lead to unpredictable results. Such bit combinations are reserved. When writing, software must always set such bit fields to legal values as provided in the bit field description tables. |
| **rw** | The bit or bit field can be read and written. |
| **r** | The bit or bit field can only be read (read-only). |
| **w** | The bit or bit field can only be written (write-only). Reading always return 0. |
| **h** | The bit or bit field can also be modified by hardware (such as a status bit). This attribute can be combined with ‘rw’ or ‘r’ bits to ‘rwh’ and ‘rh’ bits, respectively. |

## 1.7 Acronyms

**Table 1-5** lists the acronyms used in this document.

**Table 1-5 Acronyms**

| Acronym | Description |
|---|---|
| ADC | Analog-to-Digital Converter |
| ALU | Arithmetic/Logic Unit |
| BSL | BootStrap Loader |

**Table 1-5 Acronyms** (cont'd)

| Acronym | Description |
|---|---|
| CAN | Controller Area Network |
| CCU6 | Capture/Compare Unit 6 |
| CGU | Clock Generation Unit |
| CORDIC | Cordinate Rotation Digital Computer |
| CPU | Central Processing Unit |
| ECC | Error Correction Code |
| EVR | Embedded Voltage Regulator |
| FDR | Fractional Divider |
| GPIO | General Purpose I/O |
| IAP | In-Application Programming |
| I/O | Input/Output |
| ISP | In-System Programming |
| JTAG | Joint Test Action Group |
| LIN | Local Interconnect Network |
| MDU | Multiplication/Division Unit |
| NMI | Non-Maskable Interrupt |
| OCDS | On-Chip Debug Support |
| PC | Program Counter |
| POR | Power-On Reset |
| PLL | Phase-Locked Loop |
| PSW | Program Status Word |
| PWM | Pulse Width Modulation |
| RAM | Random Access Memory |
| ROM | Read-Only Memory |
| SFR | Special Function Register |
| SPI | Serial Peripheral Interface |
| SSC | Synchronous Serial Channel |
| T2CCU | Timer 2 Capture/Compare Unit |
| UART | Universal Asynchronous Receiver/Transmitter |
| WDT | Watchdog Timer |

# 2 Processor Architecture

The XC878 is based on a high-performance 8-bit Central Processing Unit (CPU) that is compatible with the standard 8051 processor. While the standard 8051 processor is designed around a 12-clock machine cycle, the XC878 CPU uses a 2-clock machine cycle. This allows fast access to ROM, RAM and Flash memories without wait state. The instruction set consists of 45% one-byte, 41% two-byte and 14% three-byte instructions.

The XC878 CPU provides a range of debugging features, including basic stop/start, single-step execution, breakpoint support and read/write access to the data memory, program memory and Special Function Registers (SFRs).

**Features**

- Two clocks per machine cycle architecture (for memory access without wait state)
- Program memory download option
- 15-source, 4-level interrupt controller
- Two data pointers
- Power saving modes
- Dedicated debug mode and debug signals
- Two 16-bit timers (Timer 0 and Timer 1)
- Full-duplex serial port (UART)

## 2.1 Functional Description

**Figure 2-1** shows the CPU functional blocks. The CPU consists of the instruction decoder, the arithmetic section, and the program control section. Each program instruction is decoded by the instruction decoder. This instruction decoder generates internal signals that control the functions of the individual units within the CPU. The internal signals have an effect on the source and destination of data transfers and control the arithmetic/logic unit (ALU) processing.

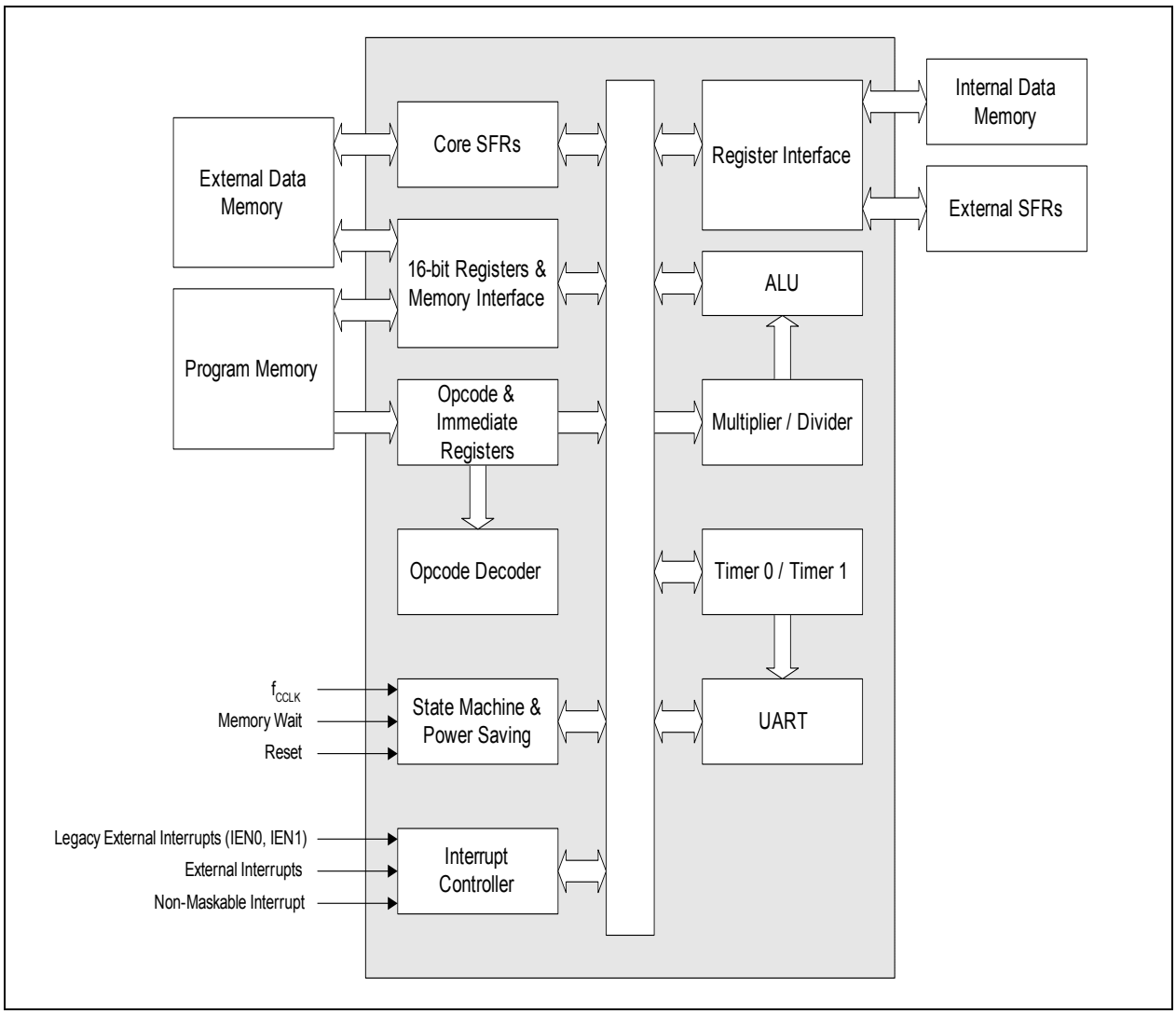



**Figure 2-1 CPU Block Diagram**

The arithmetic section of the processor performs extensive data manipulation and consists of the ALU, ACC register, B register, and PSW register.

The ALU accepts 8-bit data words from one or two sources, and generates an 8-bit result under the control of the instruction decoder. The ALU performs both arithmetic and logic operations. Arithmetic operations include add, subtract, multiply, divide, increment, decrement, BCD-decimal-add-adjust, and compare. Logic operations include AND, OR, Exclusive OR, complement, and rotate (right, left, or swap nibble (left four)). Also included is a Boolean processor performing the bit operations such as set, clear, complement, jump-if-set, jump-if-not-set, jump-if-set-and-clear, and move to/from carry. The ALU can perform the bit operations of logical AND or logical OR between any addressable bit (or its complement) and the carry flag, and place the new result in the carry flag.

The program control section controls the sequence in which the instructions stored in program memory are executed. The 16-bit Program Counter (PC) holds the address of the next instruction to be executed. The conditional branch logic enables internal and external events to the processor to cause a change in the program execution sequence.

## 2.2 CPU Register Description

The CPU registers occupy direct Internal Data Memory space locations in the range $80_H$ to $FF_H$.

### 2.2.1 Stack Pointer (SP)

The SP register contains the Stack Pointer (SP). The SP is used to load the Program Counter (PC) into Internal Data Memory during LCALL and ACALL instructions, and to retrieve the PC from memory during RET and RETI instructions. Data may also be saved on or retrieved from the stack using PUSH and POP instructions, respectively. Instructions that use the stack automatically pre-increment or post-decrement the stack pointer so that the stack pointer always points to the last byte written to the stack, i.e., the top of the stack. On reset, the SP is reset to $07_H$. This causes the stack to begin at a location = $08_H$ above register bank zero. The SP can be read or written under software control.

### 2.2.2 Data Pointer (DPTR)

The Data Pointer (DPTR) is stored in registers DPL (Data Pointer Low byte) and DPH (Data Pointer High byte) to form 16-bit addresses for External Data Memory accesses (MOVX A,@DPTR and MOVX @DPTR,A), for program byte moves (MOVC A,@A+DPTR), and for indirect program jumps (JMP @A+DPTR).

Two true 16-bit operations are allowed on the Data Pointer: load immediate (MOV DPTR,#data) and increment (INC DPTR).

### 2.2.3 Accumulator (ACC)

This register provides one of the operands for most ALU operations.

### 2.2.4 B Register

The B register is used during multiply and divide operations to provide the second operand. For other instructions, it can be treated as another scratch pad register.

### 2.2.5 Program Status Word

The Program Status Word (PSW) contains several status bits that reflect the current state of the CPU.

**PSW**
**Program Status Word Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **CY** | **AC** | **F0** | **RS1** | **RS0** | **OV** | **F1** | **P** |
| rwh | rwh | rw | rw | rw | rwh | rw | rh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **P** | 0 | rh | **Parity Flag**<br>Set/cleared by hardware after each instruction to indicate an odd/even number of “one” bits in the accumulator, i.e., even parity. |
| **F1** | 1 | rw | **General Purpose Flag** |
| **OV** | 2 | rwh | **Overflow Flag**<br>Used by arithmetic instructions |
| **RS1, RS0** | 4:3 | rw | **Register Bank Select**<br>These bits are used to select one of the four register banks.<br>00 Bank 0 selected, data address $00_H$-$07_H$<br>01 Bank 1 selected, data address $08_H$-$0F_H$<br>10 Bank 2 selected, data address $10_H$-$17_H$<br>11 Bank 3 selected, data address $18_H$-$1F_H$ |
| **F0** | 5 | rw | **General Purpose Flag** |
| **AC** | 6 | rwh | **Auxiliary Carry Flag**<br>Used by instructions that execute BCD operations |
| **CY** | 7 | rwh | **Carry Flag**<br>Used by arithmetic instructions |

### 2.2.6 Extended Operation (EO)

The instruction set includes an additional instruction MOVC @(DPTR++),A which allows program memory to be written. This instruction may be used to download code into the program memory when the CPU is initialized and subsequently, also to provide software updates. The instruction copies the contents of the accumulator to the code memory at the location pointed to by the current data pointer, and then increments the data pointer.

The instruction uses the opcode $A5_H$, which is the same as the software break instruction TRAP (see **Table 2-1**). Register bit EO.TRAP_EN is used to select the instruction executed by the opcode A5H. When TRAP_EN is 0 (default), the $A5_H$ opcode executes the MOVC instruction. When TRAP_EN is 1, the $A5_H$ opcode executes the software break instruction TRAP, which switches the CPU to debug mode for breakpoint processing.

**EO**
**Extended Operation Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **0** | | | **TRAP_EN** | **0** | | | **DPSEL0** |
| r | | | rw | r | | | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **DPSEL0** | 0 | rw | **Data Pointer Select**<br>0 DPTR0 is selected<br>1 DPTR1 is selected |
| **TRAP_EN** | 4 | rw | **TRAP Enable**<br>0 Select MOVC @(DPTR++),A<br>1 Select software TRAP instruction |
| **0** | [3:1], [7:5] | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

### 2.2.7 Power Control (PCON)

The CPU has two power-saving modes: idle mode and power-down mode. The idle mode can be entered via the PCON register. In idle mode, the clock to the CPU is stopped while the timers, serial port and interrupt controller continue to run using a half-speed clock. In power-down mode, the clock to the entire CPU is stopped.

**PCON**
**Power Control Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **SMOD** | **0** | | | **GF1** | **GF0** | **0** | **IDLE** |
| rw | r | | | rw | rw | r | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **IDLE** | 0 | rw | **Idle Mode Enable**<br>0 Do not enter idle mode<br>1 Enter idle mode |
| **GF0** | 2 | rw | **General Purpose Flag Bit 0** |
| **GF1** | 3 | rw | **General Purpose Flag Bit 1** |
| **0** | 1, [6:4] | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

## 2.3 Memory Extension

The standard amount of addressable program or external data memory (or a Bank) in an 8051 system is 64 Kbytes. The XC800 core supports memory expansion of up to 1 Mbyte and this is enabled by the availability of a Memory Management Unit (MMU) and a Memory Extension Stack. The MMU adds a set of Memory Extension registers (MEX1, MEX2, and MEX3) to control access to the extended memory space by different addressing modes. The Memory Extension Stack is used by the hardware to ‘push’ and ‘pop’ values of MEX1.

Program Code is always fetched from the 64-Kbyte block pointed to by the 4-bit Current Bank (CB) register bit field. It is updated from a 4-bit Next Bank (NB) bit field upon execution of long jump (LJMP) and call instructions. CB and NB together constitute the MEX1 register. The programmer simply writes the new bank number to NB before a jump or call instruction.

Interrupt service routines are always executed from code in the 64-Kbyte block pointed to by the Interrupt Bank (IB) register bit field. Further, memory constant data reads (in code space) and external data accesses may take place in banks other than the current bank. These banks are pointed to by the Memory Constant Bank pointer (MCB) and XRAM Bank pointer (MX). These bit fields are located in MEX2 and MEX3 registers.

### 2.3.1 Memory Extension Stack

Interrupts and Calls in Memory Extension mode make use of a Memory Extension Stack, which is updated at the same time as the standard stack.

The Memory Extension Stack is addressed using the SFR Memory Extension Stack Pointer MEXSP. This read/write register provides for a stack depth of up to 128 bytes (Bit 7 is always 0). The SFR is pre-incremented by each call instruction that is executed, and post-decremented by return instructions. MEXSP is by default reset to $7F_H$ so that the first increment selects the bottom of the stack. No indication of stack overflow is provided.

### 2.3.2 Memory Extension Effects

The following instructions can change the 64-Kbyte block pointed to: MOVC, MOVX, LJMP, LCALL, ACALL, RET and RETI.

Relative jumps (e.g. SJMP), indirect jumps (JMP @A+DPTR) and absolute jumps within 2-Kbyte regions (AJMPs), however, will in no way change the current bank. In other words, these instructions do not deselect the active 64-Kbyte bank block.

### Move Constant Instructions (MOVC)

MOVC instructions access data bytes in either the Current bank (CB19 – CB16) or a ‘Memory Constant’ bank, defined by the MCB19 – MCB16 bit field in MEX3 and MEX2. The bank selection is done by the MCM bit in MEX2 (MEX2.7).

### Move External Data Instructions (MOVX)

MOVX instructions can either access data in the Current bank or a ‘Data Memory’ bank, defined by the MX19 – MX16 bits in MEX3. The bank selection is done by the MXM bit in MEX3 (MEX3.3).

### Long Jump Instructions (LJMP)

When a jump to another bank of the Memory Extension is required, the Next Bank bits NB19 – NB16 in MEX1 (MEX1.3 – MEX1.0) must be set to the appropriate bank address before the LJMP instruction is executed. When the LJMP is encountered in the code, the Next Bank bits (NB19 – 16) are copied to the Current Bank bits CB19 – CB16 in MEX1 (MEX1.7 – MEX1.4) and appear on address bus at the beginning of the next program fetch cycle.

*Note: The Next Bank Bits (NB19 – 16) are not changed by the jump.*

### CALL Instructions (LCALL and ACALL)

Whenever an LCALL occurs, the MMU carries out the following sequence of actions:

1. The Memory Extension Stack Pointer is incremented.
2. The MEX1 register bits are made available on data bus.
3. The MEXSP register bits [6:0] are made available on address lines.
4. The Memory Extension Stack read and write signals are set for a write operation.
5. A write is performed to the Memory Extension Stack.
6. The Next Bank bits NB19 – NB16 (MEX1.3 – MEX1.0) are copied to the CB19 – CB16 bits (MEX1.7 – MEX.4).

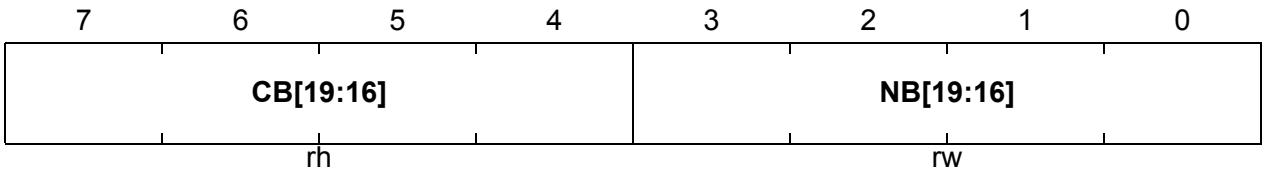
### Return Instructions (RET and RETI)

On leaving a subroutine, the MMU carries out the following sequence of actions:

1. The MEXSP register bits [6:0] are made available on address.
2. The Memory Extension Stack read and write signals are set for a read operation.
3. A read is performed on the Memory Extension Stack.
4. Memory Extension Stack data is written to the MEX1 register.
5. The Memory Extension Stack Pointer is decremented.

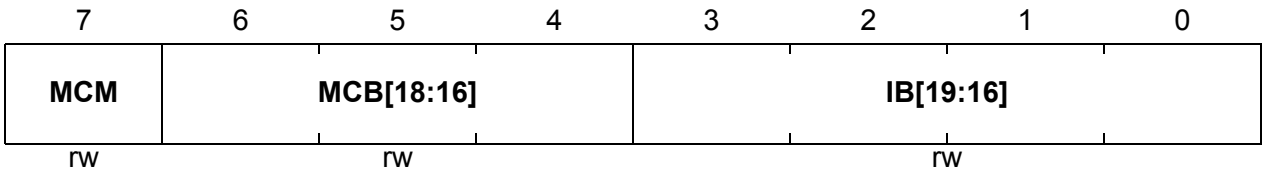
### 2.3.3 Memory Extension Registers

**MEX1**
**Memory Extension Register 1** **Reset Value: $00_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **NB[19:16]** | [3:0] | rw | **Next Bank Number** |
| **CB[19:16]** | [7:4] | rh | **Current Bank Number** |

**MEX2**
**Memory Extension Register 2** **Reset Value: $00_H$**



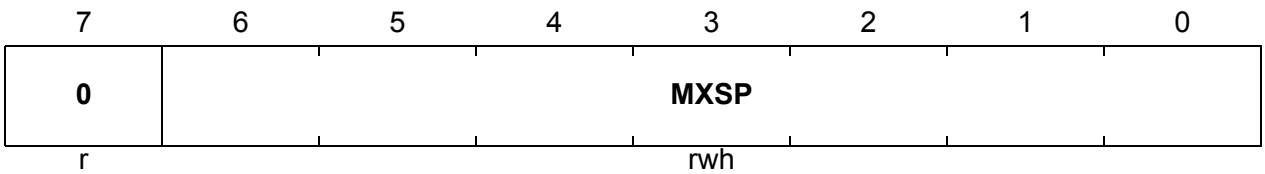

| Field | Bits | Type | Description |
|---|---|---|---|
| **IB[19:16]** | [3:0] | rw | **Interrupt Bank Number** |
| **MCB[18:16]** | [6:4] | rw | **Memory Constant Bank Number (with MEX3.7)** |
| **MCM** | 7 | rw | **Memory Constant Mode**<br>0 MOVC access data in the current bank<br>1 MOVC access data in the Memory Constant bank |

**MEX3**
**Memory Extension Register 3** **Reset Value: $00_H$**



| Field | Bits | Type | Description |
|---|---|---|---|
| **MXB[19:16]** | 4, [2:0] | rw | **XRAM Bank Number** |
| **MXM** | 3 | rw | **XRAM Bank Selector**<br>0 MOVX access data in the current bank<br>1 MOVX access data in the Memory XRAM bank |
| **MCB19** | 7 | rw | **Memory Constant Bank Number MSB** |
| **0** | [6:5] | rw | **Reserved**<br>Returns 0 if read; should be written with 0. |

**MEXSP**
**Memory Extension Stack Pointer Register** **Reset Value: $7F_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **MXSP** | [6:0] | rwh | **Memory Extension Stack Pointer**<br>It provides for a stack depth of up to 128 bytes. It is pre-incremented by call instructions and post-decremented by return instructions. |
| **0** | 7 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

## 2.4 Instruction Timing

For memory access without wait state, a CPU machine cycle comprises two input clock periods referred to as Phase 1 (P1) and Phase 2 (P2) that correspond to two different CPU states. A CPU state within an instruction is denoted by reference to the machine cycle and state number, e.g., C2P1 is the first clock period within machine cycle 2. Memory accesses take place during one or both phases of the machine cycle. SFR writes only occur at the end of P2. An instruction takes one, two or four machine cycles to execute. Registers are generally updated and the next opcode read at the end of P2 of the last machine cycle for the instruction.

With each access to the Flash memory, instruction execution times are extended by one machine cycle (one wait state), starting from either P1 or P2.

**Figure 2-2** shows the fetch/execute timing related to the internal states and phases. Execution of an instruction occurs at C1P1. For a 2-byte instruction, the second reading starts at C1P1.

**Figure 2-2** (a) shows two timing diagrams for a 1-byte, 1-cycle (1 × machine cycle) instruction. The diagram shows the instruction being executed within one machine cycle since the opcode (C1P2) is fetched from a memory without wait state.

**Figure 2-2** (b) shows two timing diagrams for a 2-byte, 1-cycle (1 × machine cycle) instruction. The diagram shows the instruction being executed within one machine cycle since the second byte (C1P1) and the opcode (C1P2) are fetched from a memory without wait state.

**Figure 2-2** (c) shows two timing diagrams of a 1-byte, 2-cycle (2 × machine cycle) instruction. The diagram shows the instruction being executed over two machine cycles with the opcode (C2P2) fetched from a memory without wait state.

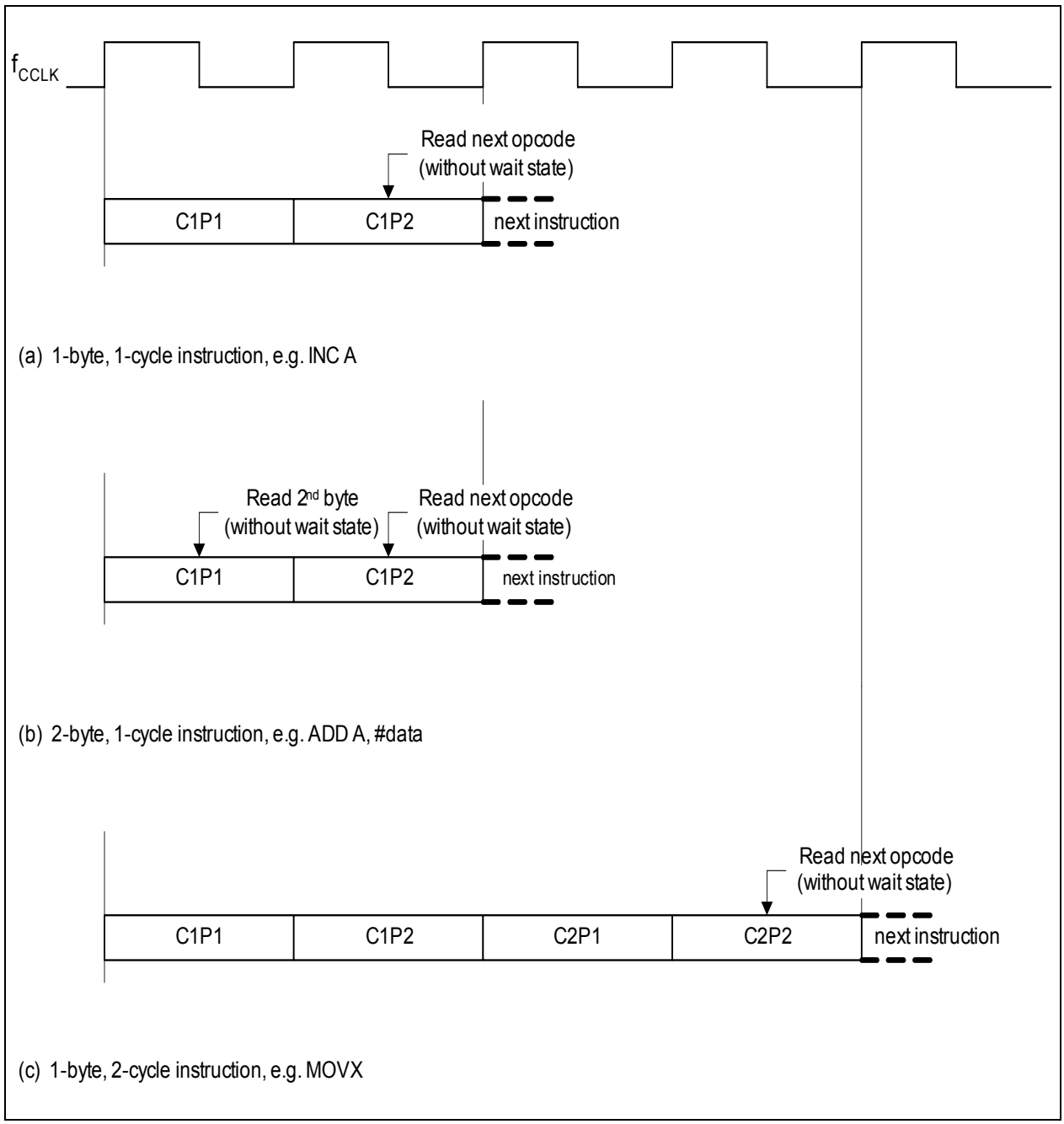



**Figure 2-2 CPU Instruction Timing**

Instructions are 1, 2 or 3 bytes long as indicated in the “Bytes” column of Table 2-1. For the XC878, the time taken for each instruction includes:

- decoding/executing the fetched opcode
- fetching the operand/s (for instructions > 1 byte)
- fetching the first byte (opcode) of the next instruction (due to XC878 CPU pipeline)

*Note: The XC878 CPU fetches the opcode of the next instruction while executing the current instruction.*

Table 2-1 provides a reference for the number of clock cycles required by each instruction. This value applies to fetching operand(s) and opcode from fast program memory (e.g., Boot ROM and XRAM) without wait state. The instruction time for the standard 8051 processor is provided in the last column for performance comparison with the XC878 CPU. From these two sets of values, the XC878 CPU is able to executes instructions faster than the standard 8051 processor by a factor of six.

**Table 2-1 CPU Instruction Timing**

| Mnemonic | Hex Code | Bytes | Number of $f_{CCLK}$ Cycles | |
|---|---|---|---|---|
| | | | XC878 (no ws) | 8051 |
| **ARITHMETIC** | | | | |
| ADD A,Rn | 28-2F | 1 | 2 | 12 |
| ADD A,dir | 25 | 2 | 2 | 12 |
| ADD A,@Ri | 26-27 | 1 | 2 | 12 |
| ADD A,#data | 24 | 2 | 2 | 12 |
| ADDC A,Rn | 38-3F | 1 | 2 | 12 |
| ADDC A,dir | 35 | 2 | 2 | 12 |
| ADDC A,@Ri | 36-37 | 1 | 2 | 12 |
| ADDC A,#data | 34 | 2 | 2 | 12 |
| SUBB A,Rn | 98-9F | 1 | 2 | 12 |
| SUBB A,dir | 95 | 2 | 2 | 12 |
| SUBB A,@Ri | 96-97 | 1 | 2 | 12 |
| SUBB A,#data | 94 | 2 | 2 | 12 |
| INC A | 04 | 1 | 2 | 12 |
| INC Rn | 08-0F | 1 | 2 | 12 |
| INC dir | 05 | 2 | 2 | 12 |
| INC @Ri | 06-07 | 1 | 2 | 12 |
| DEC A | 14 | 1 | 2 | 12 |

**Table 2-1 CPU Instruction Timing** (cont'd)

| Mnemonic | Hex Code | Bytes | Number of $f_{CCLK}$ Cycles | |
|---|---|---|---|---|
| | | | XC878 (no ws) | 8051 |
| DEC Rn | 18-1F | 1 | 2 | 12 |
| DEC dir | 15 | 2 | 2 | 12 |
| DEC @Ri | 16-17 | 1 | 2 | 12 |
| INC DPTR | A3 | 1 | 4 | 24 |
| MUL AB | A4 | 1 | 8 | 48 |
| DIV AB | 84 | 1 | 8 | 48 |
| DA A | D4 | 1 | 2 | 12 |
| **LOGICAL** | | | | |
| ANL A,Rn | 58-5F | 1 | 2 | 12 |
| ANL A,dir | 55 | 2 | 2 | 12 |
| ANL A,@Ri | 56-57 | 1 | 2 | 12 |
| ANL A,#data | 54 | 2 | 2 | 12 |
| ANL dir,A | 52 | 2 | 2 | 12 |
| ANL dir,#data | 53 | 3 | 4 | 24 |
| ORL A,Rn | 48-4F | 1 | 2 | 12 |
| ORL A,dir | 45 | 2 | 2 | 12 |
| ORL A,@Ri | 46-47 | 1 | 2 | 12 |
| ORL A,#data | 44 | 2 | 2 | 12 |
| ORL dir,A | 42 | 2 | 2 | 12 |
| ORL dir,#data | 43 | 3 | 4 | 24 |
| XRL A,Rn | 68-6F | 1 | 2 | 12 |
| XRL A,dir | 65 | 2 | 2 | 12 |
| XRL A,@Ri | 66-67 | 1 | 2 | 12 |
| XRL A,#data | 64 | 2 | 2 | 12 |
| XRL dir,A | 62 | 2 | 2 | 12 |
| XRL dir,#data | 63 | 3 | 4 | 24 |
| CLR A | E4 | 1 | 2 | 12 |
| CPL A | F4 | 1 | 2 | 12 |
| SWAP A | C4 | 1 | 2 | 12 |

**Table 2-1 CPU Instruction Timing** (cont'd)

| Mnemonic | Hex Code | Bytes | Number of $f_{CCLK}$ Cycles | |
|---|---|---|---|---|
| | | | XC878 (no ws) | 8051 |
| RL A | 23 | 1 | 2 | 12 |
| RLC A | 33 | 1 | 2 | 12 |
| RR A | 03 | 1 | 2 | 12 |
| RRC A | 13 | 1 | 2 | 12 |
| **DATA TRANSFER** | | | | |
| MOV A,Rn | E8-EF | 1 | 2 | 12 |
| MOV A,dir | E5 | 2 | 2 | 12 |
| MOV A,@Ri | E6-E7 | 1 | 2 | 12 |
| MOV A,#data | 74 | 2 | 2 | 12 |
| MOV Rn,A | F8-FF | 1 | 2 | 12 |
| MOV Rn,dir | A8-AF | 2 | 4 | 24 |
| MOV Rn,#data | 78-7F | 2 | 2 | 12 |
| MOV dir,A | F5 | 2 | 2 | 12 |
| MOV dir,Rn | 88-8F | 2 | 4 | 24 |
| MOV dir,dir | 85 | 3 | 4 | 24 |
| MOV dir,@Ri | 86-87 | 2 | 4 | 24 |
| MOV dir,#data | 75 | 3 | 4 | 24 |
| MOV @Ri,A | F6-F7 | 1 | 2 | 12 |
| MOV @Ri,dir | A6-A7 | 2 | 4 | 24 |
| MOV @Ri,#data | 76-77 | 2 | 2 | 12 |
| MOV DPTR,#data | 90 | 3 | 4 | 24 |
| MOVC A,@A+DPTR | 93 | 1 | 4 | 24 |
| MOVC A,@A+PC | 83 | 1 | 4 | 24 |
| MOVX A,@Ri | E2-E3 | 1 | 4 | 24 |
| MOVX A,@DPTR | E0 | 1 | 4 | 24 |
| MOVX @Ri,A | F2-F3 | 1 | 4 | 24 |
| MOVX @DPTR,A | F0 | 1 | 4 | 24 |
| PUSH dir | C0 | 2 | 4 | 24 |
| POP dir | D0 | 2 | 4 | 24 |

**Table 2-1 CPU Instruction Timing** (cont'd)

| Mnemonic | Hex Code | Bytes | Number of $f_{CCLK}$ Cycles | |
|---|---|---|---|---|
| | | | XC878 (no ws) | 8051 |
| XCH A,Rn | C8-CF | 1 | 2 | 12 |
| XCH A,dir | C5 | 2 | 2 | 12 |
| XCH A,@Ri | C6-C7 | 1 | 2 | 12 |
| XCHD A,@Ri | D6-D7 | 1 | 2 | 12 |
| **BOOLEAN** | | | | |
| CLR C | C3 | 1 | 2 | 12 |
| CLR bit | C2 | 2 | 2 | 12 |
| SETB C | D3 | 1 | 2 | 12 |
| SETB bit | D2 | 2 | 2 | 12 |
| CPL C | B3 | 1 | 2 | 12 |
| CPL bit | B2 | 2 | 2 | 12 |
| ANL C,bit | 82 | 2 | 4 | 24 |
| ANL C,/bit | B0 | 2 | 4 | 24 |
| ORL C,bit | 72 | 2 | 4 | 24 |
| ORL C,/bit | A0 | 2 | 4 | 24 |
| MOV C,bit | A2 | 2 | 2 | 12 |
| MOV bit,C | 92 | 2 | 4 | 24 |
| **BRANCHING** | | | | |
| ACALL addr11 | 11->F1 | 2 | 4 | 24 |
| LCALL addr16 | 12 | 3 | 4 | 24 |
| RET | 22 | 1 | 4 | 24 |
| RETI | 32 | 1 | 4 | 24 |
| AJMP addr 11 | 01->E1 | 2 | 4 | 24 |
| LJMP addr 16 | 02 | 3 | 4 | 24 |
| SJMP rel | 80 | 2 | 4 | 24 |
| JC rel | 40 | 2 | 4 | 24 |
| JNC rel | 50 | 2 | 4 | 24 |
| JB bit,rel | 20 | 3 | 4 | 24 |
| JNB bit,rel | 30 | 3 | 4 | 24 |

**Table 2-1 CPU Instruction Timing** (cont'd)

| Mnemonic | Hex Code | Bytes | Number of $f_{CCLK}$ Cycles | |
|---|---|---|---|---|
| | | | XC878 (no ws) | 8051 |
| JBC bit,rel | 10 | 3 | 4 | 24 |
| JMP @A+DPTR | 73 | 1 | 4 | 24 |
| JZ rel | 60 | 2 | 4 | 24 |
| JNZ rel | 70 | 2 | 4 | 24 |
| CJNE A,dir,rel | B5 | 3 | 4 | 24 |
| CJNE A,#d,rel | B4 | 3 | 4 | 24 |
| CJNE Rn,#d,rel | B8-BF | 3 | 4 | 24 |
| CJNE @Ri,#d,rel | B6-B7 | 3 | 4 | 24 |
| DJNZ Rn,rel | D8-DF | 2 | 4 | 24 |
| DJNZ dir,rel | D5 | 3 | 4 | 24 |
| **MISCELLANEOUS** | | | | |
| NOP | 00 | 1 | 2 | 12 |
| **ADDITIONAL INSTRUCTIONS** | | | | |
| MOVC @(DPTR++),A | A5 | 1 | 4 | – |
| TRAP | A5 | 1 | 2 | – |

# 3 Memory Organization

The XC878 CPU operates in the following five address spaces:

- 52/64 KBytes of Flash program memory
- 8 Kbytes of Boot ROM program memory
- 256 bytes of internal RAM data memory
- 3 Kbytes of XRAM memory
  (XRAM can be read/written as program memory or external data memory)
- a 128-byte Special Function Register area

**Figure 3-1** illustrates the memory address spaces of the XC878 with 64K embedded Flash.

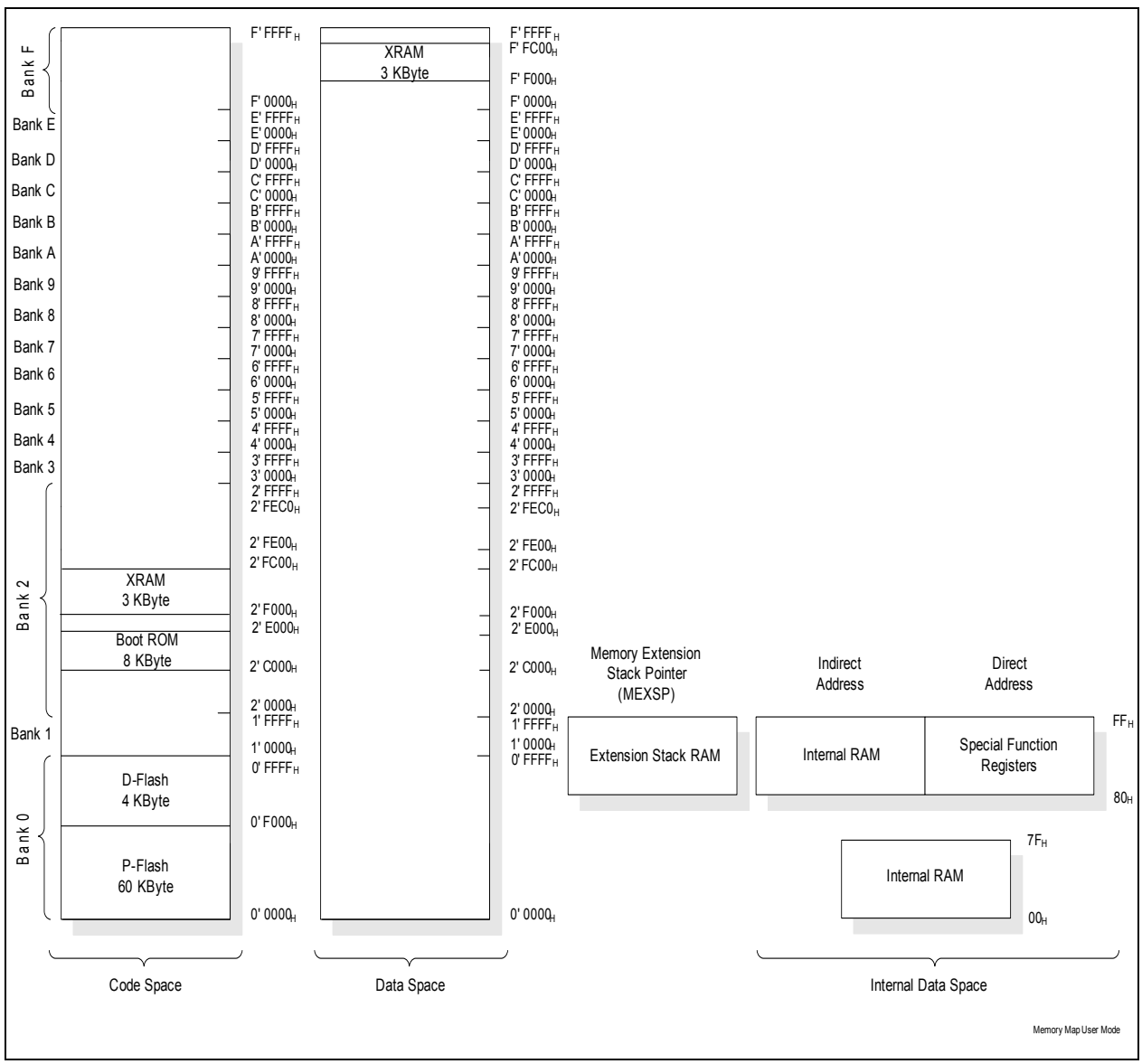



**Figure 3-1 Memory Map of XC878 with 64K Flash Memory in user mode**

**Figure 3-2** illustrates the memory address spaces of the XC878 with 52K embedded Flash.

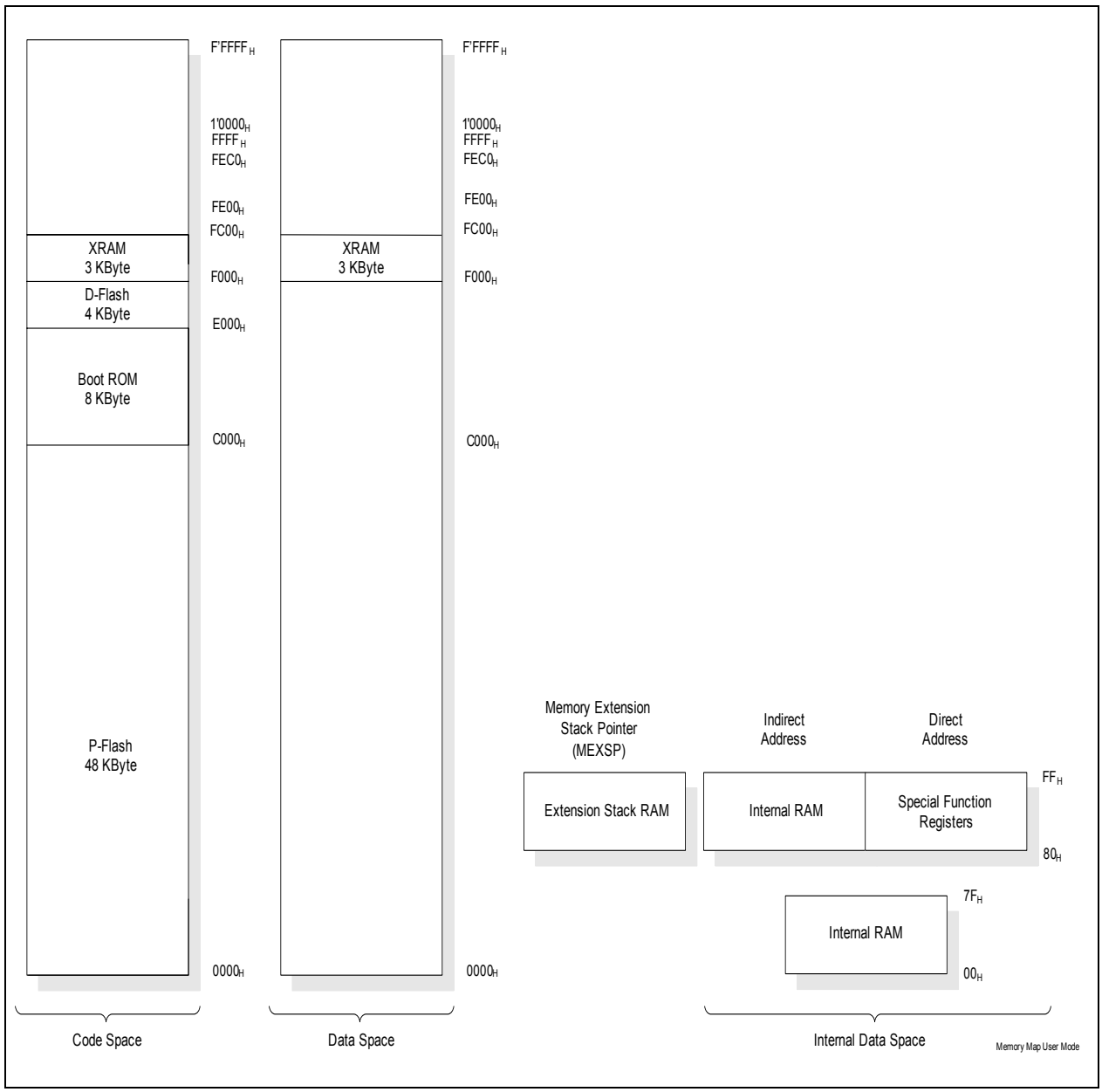



**Figure 3-2 Memory Map of XC878 with 52K Flash Memory in user mode**

## 3.1 Program Memory

The performance of the CPU is optimized with a dedicated interface for direct interfacing with the program memory without using any port pin. This means that a code fetch can occur on every rising edge of the clock. In XC878, program memory extension is accomplished with a 4-bit Current Bank pointer (CB) or the 4-bit Memory Constant Bank pointer (MC), selected by the MCM bit. The data is fetched from the 64-Kbyte block pointed to by CB or MC as described in **Chapter 2.3**. The program code is fetched from the 64-Kbyte block pointed to by CB or MC.

## 3.2 Data Memory

The data memory space consists of an internal and external memory space. The labels ‘internal’ and ‘external’ for data memory are used to distinguish between the register memory and the data space accessed using ‘MOVX’ instructions. They do not imply that the external data memory is located off-chip. In XC878, there is a 3-Kbyte of on-chip external data memory, XRAM. The internal data memory area is addressed using 8-bit addresses. The on-chip XRAM data memory are addressable by 8-bit or 16-bit indirect address with ‘MOVX’, additionally with up to 4-bit for selection of extended memory bank.

### 3.2.1 Internal Data Memory

The internal data memory is divided into three physically separate and distinct blocks: the 256-byte RAM, the 128-byte special function register (SFR) area and the 128-byte extension memory stack RAM. While the upper 128 bytes of RAM, the SFR area and the 128-byte extension memory stack RAM share the same address locations, they are accessed through different addressing modes. The lower 128 bytes of RAM can be accessed through direct or register indirect addressing, while the upper 128 bytes of RAM can be accessed through register indirect addressing only. The special function registers are accessible through direct addressing. As for the 128-byte extension memory stack RAM, they are not accessible directly. Its address is indicated via MEXSP register.

The 16 bytes of RAM that occupy addresses from $20_H$ to $2F_H$ are bit addressable. RAM occupying direct addresses from $30_H$ to $7F_H$ can be used as scratch pad registers or for used for the stack.

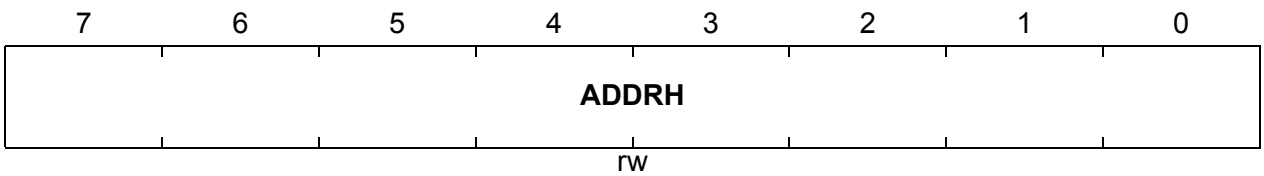
### 3.2.2 External Data Memory

The 3-Kbyte XRAM is mapped to both the external data memory area and the program memory area. It can be accessed using both ‘MOVX’ and ‘MOVC’ instructions.

The ‘MOVX’ instructions for XRAM access use either 8-bit or 16-bit indirect addresses. While the DPTR register is used for 16-bit addressing, either register R0 or R1 is used to form the 8-bit address. The upper byte of the XRAM address during execution of the 8-bit accesses is defined by the value stored in register XADDRH. Hence, the write instruction for setting the higher order XRAM address in register XADDRH must precede the ‘MOVX’ instruction.

The bank where the memories resides must also be selected with the 4-bit XRAM Bank pointer in MEX3.MX (XRAM bank) or the 4-bit Current Bank pointer in MEX1.CB (current bank), depending on bit MXM. The data is fetched from the 64-Kbyte block pointed to by CB or MX. It is further described in **Chapter 2.3**.

**XADDRH**
**On-Chip XRAM Address Higher Order** **Reset Value: $F0_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **ADDRH** | 7:0 | rw | **Higher Order of On-chip XRAM Address**<br>This value is from $F0_H$ to $FB_H$ for the XC878. |

## 3.3 Memory Protection Strategy

The XC878 memory protection strategy includes:

- Basic Protection: The user is able to block any external access via the boot option to any memory. The OCDS functionality and the functions to download code to XRAM and Flash will be disabled.
- Read-out protection: The user is able to protect the contents in the Flash memory from being read. It is used to supplement the basic protection.
- Flash program and erase protection: The user is able to protect the contents in the Flash memory from accidental erasing and programming. This feature is used together with the read-out protection.

These protection strategies are enabled by programming a valid password (16-bit non-one value) via Bootstrap Loader (BSL) mode 6.

### 3.3.1 Flash Memory Protection

As long as a valid password is available, all external access to the device, including the Flash, will be blocked.

For additional security, the Flash hardware protection can be enabled to implement a second layer of read-out protection, as well as to enable program and erase protection.

Flash hardware protection is available and comes in two modes:

- Mode 0: Only the P-Flash is protected; the D-Flash is unprotected.
- Mode 1: Both the P-Flash and D-Flash are protected.

The selection of each protection mode and the restrictions imposed are summarized in **Table 3-1**.

**Table 3-1 Flash Protection Modes**

| Flash Protection | Without hardware protection | With hardware protection | |
|---|---|---|---|
| **Hardware Protection Mode** | - | 0 | 1 |
| **Activation** | Program a valid password via BSL mode 6 | | |
| **Selection** | Bit 13 of password = 0 | Bit 13 of password = 1<br>MSB of password = 0 | Bit 13 of password = 1<br>MSB of password = 1 |
| **P-Flash contents can be read by** | Read instructions in any P-Flash | Read instructions in the P-Flash | Read instructions in the P-Flash or D-Flash |
| **External access to P-Flash** | Not possible | Not possible | Not possible |
| **P-Flash program and erase** | Possible | Possible only on the condition that (MSB - 1) of password is set to 1 | Possible only on the condition that (MSB - 1) of password is set to 1 |
| **D-Flash contents can be read by** | Read instructions in any P-Flash | Read instructions in any P-Flash | Read instructions in the P-Flash or D-Flash |
| **External access to D-Flash** | Not possible | Not possible | Not possible |
| **D-Flash program** | Possible | Possible | Possible, on the condition that (MSB - 1) of password is set to 1 |
| **D-Flash erase** | Possible | Possible, on these conditions:<br>• MISC_CON.DFLASHEN bit is set to 1 prior to each erase operation; or<br>• the (MSB - 1) of password is set to 1 | Possible, on the condition that (MSB - 1) of password is set to 1 |

In Flash hardware protection mode 0, an erase operation on the D-Flash block can proceed only if bit DFLASHEN in register MISC_CON is set to 1. At the end of each erase operation, DFLASHEN is cleared automatically by hardware. Hence, it is necessary to set DFLASHEN before each D-Flash erase operation. While the setting of DFLASHEN is taken care by the Bootstrap Loader (BSL) routine during D-Flash in-system erasing, DFLASHEN must be set by the user application code before starting each D-Flash in-application erasing. The extra step serves to prevent inadvertent destruction of the D-Flash contents.

The user programmable password must be of the format shown in **Table 3-2**.

**Table 3-2 User Programmable Password Bit Fields**

| Bits | Size | Usage | Value |
|---|---|---|---|
| 15 | 1-bit | Flash hardware protection mode selection bit | 0 Flash hardware protection mode 0 is selected.<br>1 Flash hardware protection mode 1 is selected. |
| 14 | 1-bit | Flash memory program and erase selection bit | 0 Flash memory programming and erasing is blocked.<br>1 Flash memory programming and erasing is not blocked. |
| 13 | 1-bit | Flash hardware protection enable bit | 0 Flash hardware protection will not be activated.<br>1 Flash hardware protection will be activated. |
| 12 | 1-bit | Select field for Flash memory blocks to be erased during unprotection | 0 Only P-Flash are erased during unprotection.<br>1 Both P-Flash and D-Flash are erased durng unprotection. |
| 11:0 | 12-bit | User-defined password field | This password field can be of any value that the user desire with only one condition that bit 8 to bit 11 of this password must at least contain a zero value. |

BSL mode 6, which is used for enabling Flash protection, can also be used for disabling Flash protection. Here, the programmed password must be provided by the user. To disable the flash protection, a password match is required. A password match triggers an automatic erase of the protected P-Flash and D-Flash contents, including the programmed password. With a valid password, the Flash hardware protection is then enabled or disabled upon next reset. For the other protection strategies, no reset is necessary.

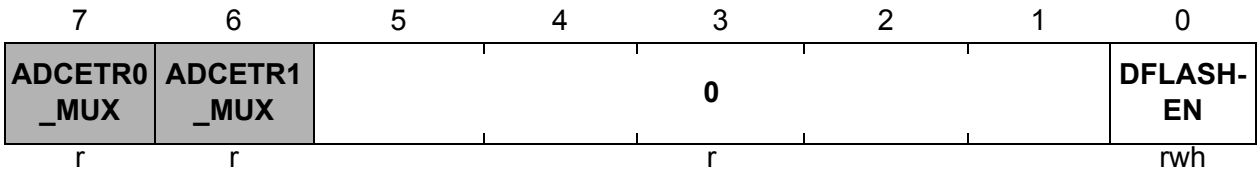
Although no protection scheme can be considered infallible, the XC878 memory protection strategy provides a very high level of protection for a general purpose microcontroller.

*Note: If ROM read-out protection is enabled, only read instructions in the ROM memory can target the ROM contents.*

### 3.3.2 Miscellaneous Control Register

The MISC_CON register contains the DFLASHEN bit to enable the erase of a D-Flash block. This bit has no effect if the Flash hardware protection is not enabled or protection mode 1 is enabled.

**MISC_CON**
**Miscellaneous Control Register** **Reset Value: $00_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **DFLASHEN** | 0 | rwh | **D-Flash Block Enable**<br>0 D-Flash block cannot be erased<br>1 D-Flash block can be erased<br>This bit is reset by hardware after each D-Flash erase operation.<br>*Note: Superfluous setting of this bit has no adverse effect on the XC878 system operation.* |
| **0** | [5:1] | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

## 3.4 Special Function Registers

The Special Function Registers (SFRs) occupy direct internal data memory space in the range $80_H$ to $FF_H$. All registers, except the program counter, reside in the SFR area. The SFRs include pointers and registers that provide an interface between the CPU and the on-chip peripherals. As the 128-SFR range is less than the total number of registers required, address extension mechanisms are required to increase the number of addressable SFRs. The address extension mechanisms include:

- Mapping
- Paging

### 3.4.1 Address Extension by Mapping

Address extension is performed at the system level by mapping. The SFR area is extended into two portions: the standard (non-mapped) SFR area and the mapped SFR area. Each portion supports the same address range $80_H$ to $FF_H$, bringing the number of addressable SFRs to 256. The extended address range is not directly controlled by the CPU instruction itself, but is derived from bit RMAP in the system control register SYSCON0 at address $8F_H$. To access SFRs in the mapped area, bit RMAP in SFR SYSCON0 must be set. However, the SFRs in the standard area can be accessed by clearing bit RMAP. **Figure 3-3** shows how the SFR area can be selected.

As long as bit RMAP is set, the mapped SFR area can be accessed. This bit is not cleared automatically by hardware. Thus, before standard/mapped registers are accessed, bit RMAP must be cleared/set, respectively, by software.

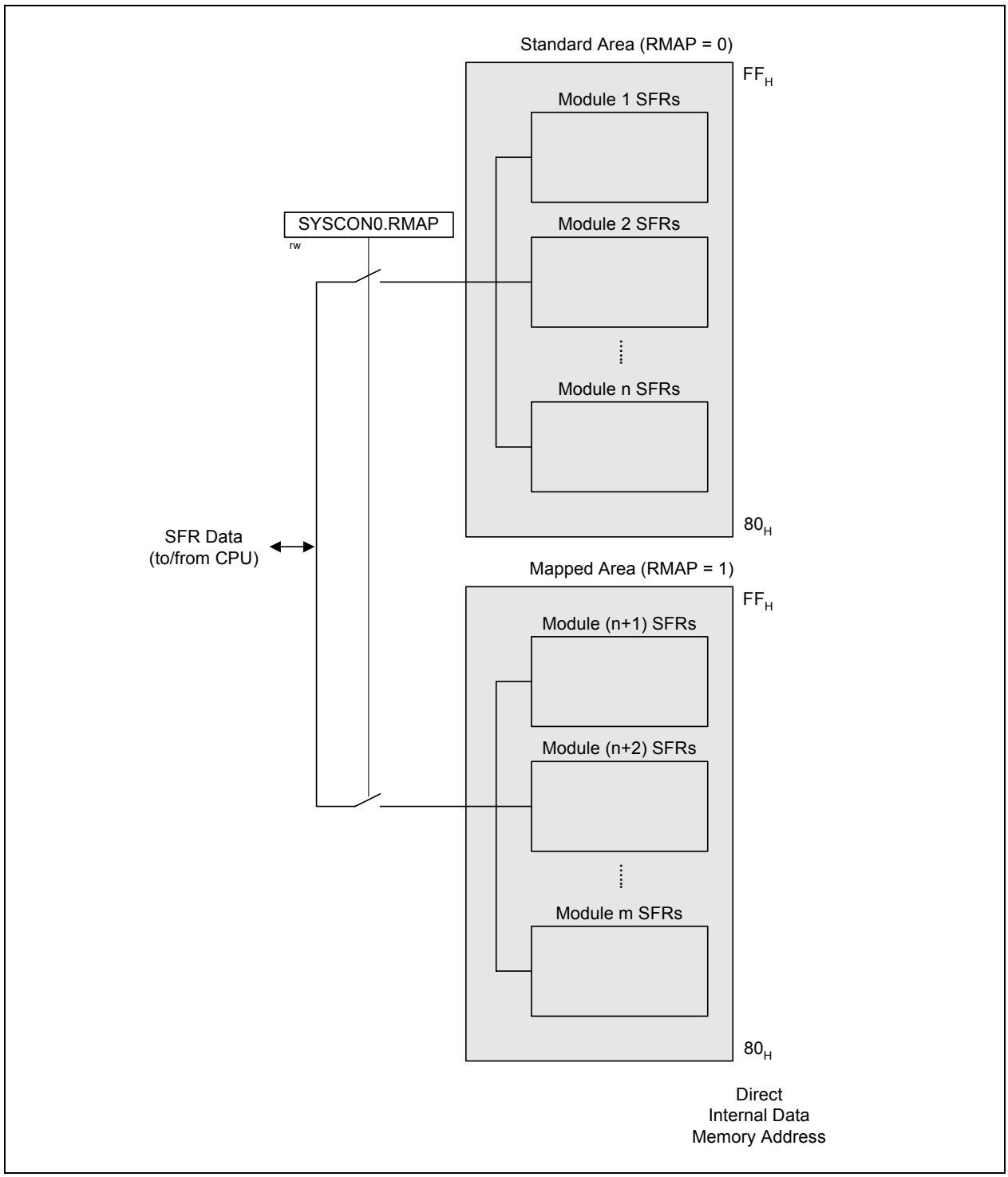



**Figure 3-3 Address Extension by Mapping**

### 3.4.1.1 System Control Register 0

The SYSCON0 register contains bits to select the SFR mapping and interrupt structure 2 mode.

**SYSCON0**
**System Control Register 0** **Reset Value: $04_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | **0** | | **IMODE** | **0** | **1** | **0** | **RMAP** |
| | r | | rw | r | r | r | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **RMAP** | 0 | rw | **Special Function Register Map Control**<br>0 The access to the standard SFR area is enabled.<br>1 The access to the mapped SFR area is enabled. |
| **0** | 2 | r | **Reserved**<br>Returns 1 if read; should be written with 1. |
| **0** | [7:5], 3,1 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

*Note: The RMAP bit should be cleared/set using ANL or ORL instructions.*

### 3.4.2 Address Extension by Paging

Address extension is further performed at the module level by paging. With the address extension by mapping, the XC878 has a 256-SFR address range. However, this is still less than the total number of SFRs needed by the on-chip peripherals. To meet this requirement, some peripherals have a built-in local address extension mechanism for increasing the number of addressable SFRs. The extended address range is not directly controlled by the CPU instruction itself, but is derived from bit field PAGE in the module page register MOD_PAGE. Hence, the bit field PAGE must be programmed before accessing the SFRs of the target module. Each module may contain a different number of pages and a different number of SFRs per page, depending on the specific requirement. Besides setting the correct RMAP bit value to select the SFR area, the user must also ensure that a valid PAGE is selected to target the desired SFRs. **Figure 3-4** shows how a page inside the extended address range can be selected.

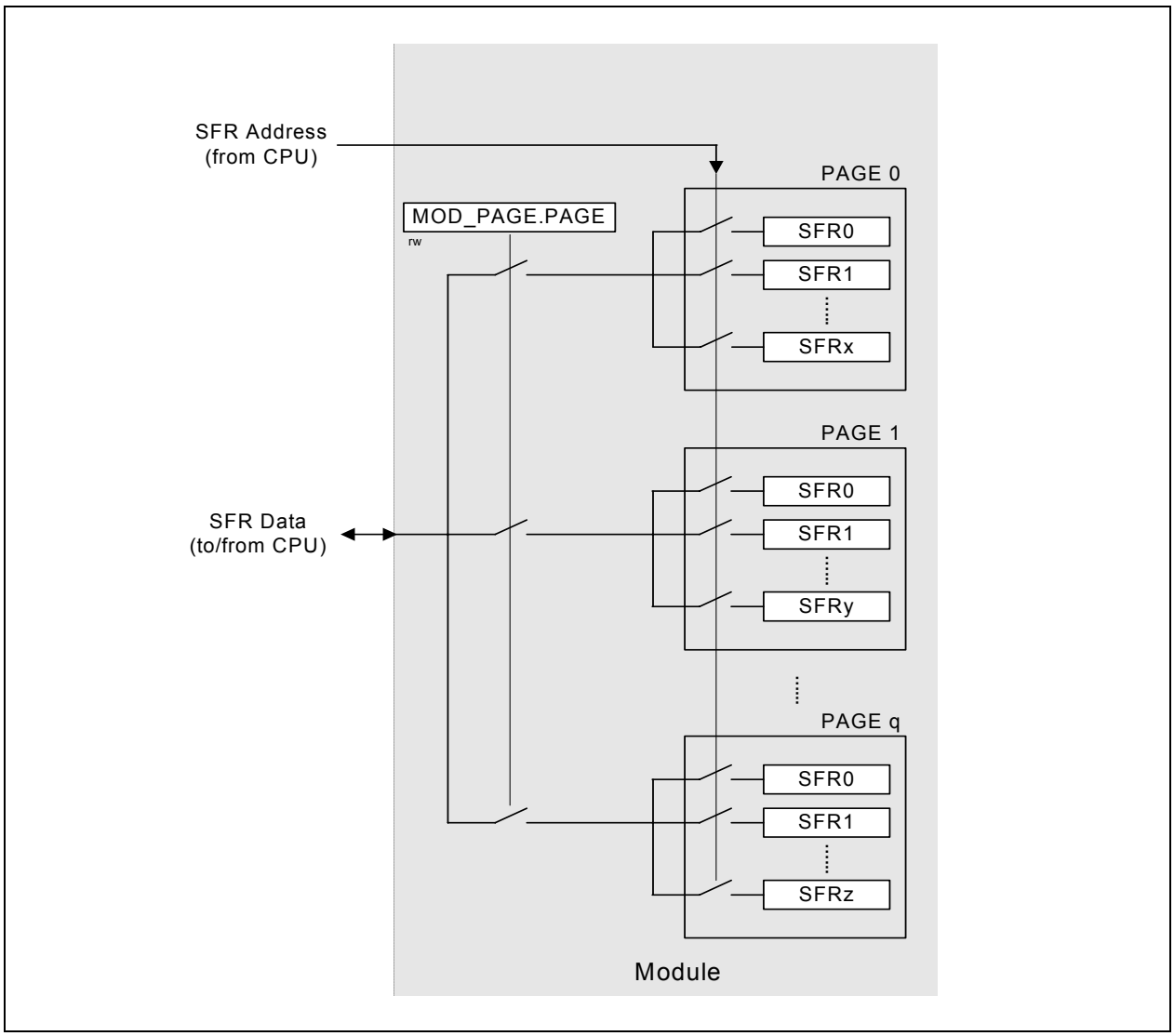



**Figure 3-4 Address Extension by Paging**

In order to access a register located in a page other than the current one, the current page must be exited. This is done by reprogramming the bit field PAGE in the page register. Only then can the desired access be performed.

If an interrupt routine is initiated between the page register access and the module register access, and the interrupt needs to access a register located in another page, the current page setting can be saved, the new one programmed, and the old page setting restored. This is possible with the storage fields STx (x = 0 - 3) for the save and restore action of the current page setting. By indicating which storage bit field should be used in parallel with the new page value, a single write operation can:

- Save the contents of PAGE in STx before overwriting with the new value
  (this is done at the beginning of the interrupt routine to save the current page setting and program the new page number); or
- Overwrite the contents of PAGE with the contents of STx, ignoring the value written to the bit positions of PAGE
  (this is done at the end of the interrupt routine to restore the previous page setting before the interrupt occurred)

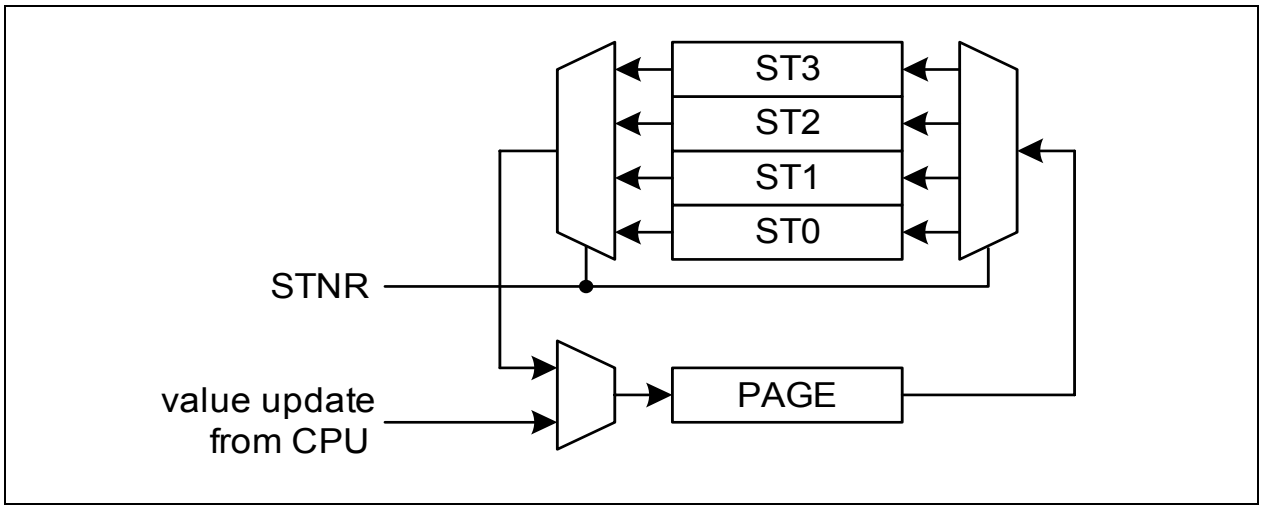



**Figure 3-5 Storage Elements for Paging**

With this mechanism, a certain number of interrupt routines (or other routines) can perform page changes without reading and storing the previously used page information. The use of only write operations makes the system simpler and faster. Consequently, this mechanism significantly improves the performance of short interrupt routines.

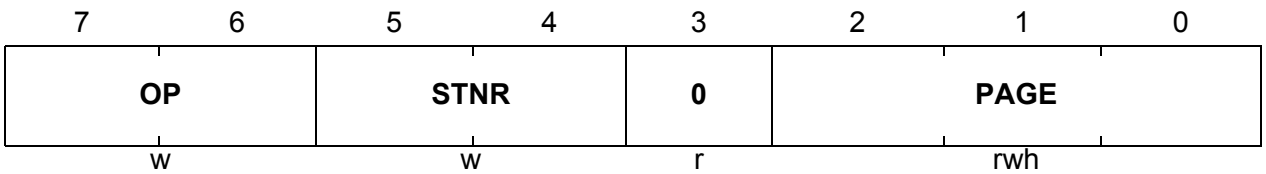
The XC878 supports local address extension for:

- Parallel Ports
- Analog-to-Digital Converter (ADC)
- Capture/Compare Unit 6 (CCU6)
- System Control Registers
- Timer 2 Capture/Compare Unit (T2CCU)

### 3.4.2.1 Page Register

The page register has the following definition:

**MOD_PAGE**
**Page Register for module MOD** **Reset Value: $00_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **PAGE** | [2:0] | rwh | **Page Bits**<br>When written, the value indicates the new page.<br>When read, the value indicates the currently active page. |
| **STNR** | [5:4] | w | **Storage Number**<br>This number indicates which storage bit field is the target of the operation defined by bit field OP.<br>If OP = $10_B$,<br>the contents of PAGE are saved in STx before being overwritten with the new value.<br>If OP = $11_B$,<br>the contents of PAGE are overwritten by the contents of STx. The value written to the bit positions of PAGE is ignored.<br><br>00 ST0 is selected.<br>01 ST1 is selected.<br>10 ST2 is selected.<br>11 ST3 is selected. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **OP** | [7:6] | w | **Operation**<br>0X Manual page mode. The value of STNR is ignored and PAGE is directly written.<br>10 New page programming with automatic page saving. The value written to the bit positions of PAGE is stored. In parallel, the previous contents of PAGE are saved in the storage bit field STx indicated by STNR.<br>11 Automatic restore page action. The value written to the bit positions PAGE is ignored and instead, PAGE is overwritten by the contents of the storage bit field STx indicated by STNR. |
| **0** | 3 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

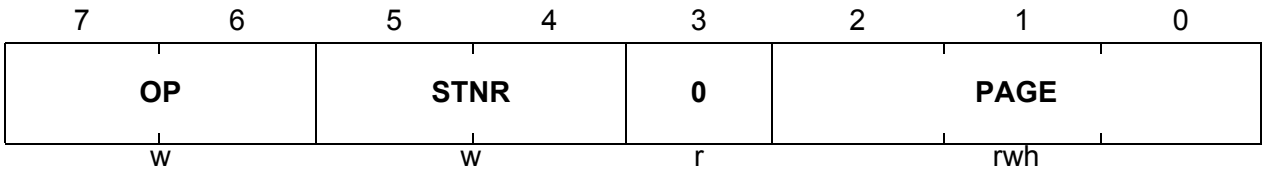
### 3.4.3 Bit-Addressing

SFRs that have addresses in the form of $1XXXX000_B$ (e.g., $80_H$, $88_H$, $90_H$, ..., $F0_H$, $F8_H$) are bitaddressable.

### 3.4.4 System Control Registers

The system control SFRs are used to control the overall system functionalities, such as interrupts, variable baud rate generation, clock management, bit protection scheme, oscillator and PLL control. The SFRs are located in the standard memory area (RMAP = 0) and are organized into 2 pages. The SCU_PAGE register is located at $BF_H$. It contains the page value and page control information.

**SCU_PAGE**
**Page Register for System Control** **Reset Value: $00_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **PAGE** | [2:0] | rwh | **Page Bits**<br>When written, the value indicates the new page.<br>When read, the value indicates the currently active page. |
| **STNR** | [5:4] | w | **Storage Number**<br>This number indicates which storage bit field is the target of the operation defined by bit field OP.<br>If OP = $10_B$,<br>the contents of PAGE are saved in STx before being overwritten with the new value.<br>If OP = $11_B$,<br>the contents of PAGE are overwritten by the contents of STx. The value written to the bit positions of PAGE is ignored.<br><br>00 ST0 is selected.<br>01 ST1 is selected.<br>10 ST2 is selected.<br>11 ST3 is selected. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **OP** | [7:6] | w | **Operation**<br>0X Manual page mode. The value of STNR is ignored and PAGE is directly written.<br>10 New page programming with automatic page saving. The value written to the bit positions of PAGE is stored. In parallel, the previous contents of PAGE are saved in the storage bit field STx indicated by STNR.<br>11 Automatic restore page action. The value written to the bit positions PAGE is ignored and instead, PAGE is overwritten by the contents of the storage bit field STx indicated by STNR. |
| **0** | 3 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

### 3.4.4.1 Bit Protection Scheme

The bit protection scheme prevents direct software writing of selected bits (i.e., protected bits) using the PASSWD register. When the bit field MODE is $11_B$, writing $10011_B$ to the bit field PASS opens access to writing of all protected bits, and writing $10101_B$ to the bit field PASS closes access to writing of all protected bits. In both cases, the value of the bit field MODE is not changed even if PASSWD register is written with $98_H$ or $A8_H$. It can only be changed when bit field PASS is written with $11000_B$, for example, writing $D0_H$ to PASSWD register disables the bit protection scheme.

Note that access is opened for maximum 32 CCLKs if the “close access” password is not written. If “open access” password is written again before the end of 32 CCLK cycles, there will be a recount of 32 CCLK cycles. The protected bits include the N-, P- and K-Divider bits, NDIV, PDIV and KDIV; the PLL output bypass control, PLLBYP; the XTAL power down control, XPD; the oscillator source select, OSCSS; the PLL power down control, PLLPD; the Watchdog Timer enable bit, WDTEN; and the power-down and slow-down enable bits, PD and SD.

**PASSWD**
**Password Register** **Reset Value: $07_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| PASS | | | | | PROTECT_S | MODE | |
| w | | | | | rh | rw | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **MODE** | [1:0] | rw | **Bit Protection Scheme Control bits**<br>00 Scheme disabled - direct access to the protected bits is allowed.<br>11 Scheme enabled - the bit field PASS has to be written with the passwords to open and close the access to protected bits. (default)<br>Others: Scheme Enabled<br>These two bits cannot be written directly. To change the value between $11_B$ and $00_B$, the bit field PASS must be written with $11000_B$; only then, will the MODE[1:0] be registered. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **PROTECT_S** | 2 | rh | **Bit Protection Signal Status bit**<br>This bit shows the status of the protection.<br>0 Software is able to write to all protected bits.<br>1 Software is unable to write to any protected bits. |
| **PASS** | [7:3] | w | **Password bits**<br>The Bit Protection Scheme only recognizes three patterns.<br>$11000_B$ Enables writing of the bit field MODE.<br>$10011_B$ Opens access to writing of all protected bits.<br>$10101_B$ Closes access to writing of all protected bits. |

### 3.4.5 XC878 Register Overview

The SFRs of the XC878 are organized into groups according to their functional units. The contents (bits) of the SFRs are summarized in **Chapter 3.4.5.1** to **Chapter 3.4.5.15**.

*Note: The addresses of the bitaddressable SFRs appear in bold typeface.*

#### 3.4.5.1 CPU Registers

The CPU SFRs can be accessed in both the standard and mapped memory areas (RMAP = 0 or 1).

**Table 3-3 CPU Register Overview**

| Addr | Register Name | Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| RMAP = 0 or 1 | | | | | | | | | | |
| $81_H$ | **SP** **Reset: $07_H$** Stack Pointer Register | Bit Field | SP | | | | | | | |
| | | Type | rw | | | | | | | |
| $82_H$ | **DPL** **Reset: $00_H$** Data Pointer Register Low | Bit Field | DPL7 | DPL6 | DPL5 | DPL4 | DPL3 | DPL2 | DPL1 | DPL0 |
| | | Type | rw | rw | rw | rw | rw | rw | rw | rw |
| $83_H$ | **DPH** **Reset: $00_H$** Data Pointer Register High | Bit Field | DPH7 | DPH6 | DPH5 | DPH4 | DPH3 | DPH2 | DPH1 | DPH0 |
| | | Type | rw | rw | rw | rw | rw | rw | rw | rw |
| $87_H$ | **PCON** **Reset: $00_H$** Power Control Register | Bit Field | SMOD | 0 | | | GF1 | GF0 | 0 | IDLE |
| | | Type | rw | r | | | rw | rw | r | rw |
| **$88_H$** | **TCON** **Reset: $00_H$** Timer Control Register | Bit Field | TF1 | TR1 | TF0 | TR0 | IE1 | IT1 | IE0 | IT0 |
| | | Type | rwh | rw | rwh | rw | rwh | rw | rwh | rw |
| $89_H$ | **TMOD** **Reset: $00_H$** Timer Mode Register | Bit Field | GATE 1 | T1S | T1M | | GATE 0 | T0S | T0M | |
| | | Type | rw | rw | rw | | rw | rw | rw | |
| $8A_H$ | **TL0** **Reset: $00_H$** Timer 0 Register Low | Bit Field | VAL | | | | | | | |
| | | Type | rwh | | | | | | | |
| $8B_H$ | **TL1** **Reset: $00_H$** Timer 1 Register Low | Bit Field | VAL | | | | | | | |
| | | Type | rwh | | | | | | | |
| $8C_H$ | **TH0** **Reset: $00_H$** Timer 0 Register High | Bit Field | VAL | | | | | | | |
| | | Type | rwh | | | | | | | |
| $8D_H$ | **TH1** **Reset: $00_H$** Timer 1 Register High | Bit Field | VAL | | | | | | | |
| | | Type | rwh | | | | | | | |
| $94_H$ | **MEX1** **Reset: $00_H$** Memory Extension Register 1 | Bit Field | CB | | | | NB | | | |
| | | Type | r | | | | rw | | | |
| $95_H$ | **MEX2** **Reset: $00_H$** Memory Extension Register 2 | Bit Field | MCM | MCB | | | IB | | | |
| | | Type | rw | rw | | | rw | | | |
| $96_H$ | **MEX3** **Reset: $00_H$** Memory Extension Register 3 | Bit Field | MCB19 | 0 | | MXB19 | MXM | MXB | | |
| | | Type | rw | rw | | rw | rw | rw | | |

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

| Addr | Register Name | Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| $97_H$ | **MEXSP** **Reset: $7F_H$**<br>Memory Extension Stack Pointer Register | Bit Field | 0 | MXSP | | | | | | |
| | | Type | r | rwh | | | | | | |
| **$98_H$** | **SCON** **Reset: $00_H$**<br>Serial Channel Control Register | Bit Field | SM0 | SM1 | SM2 | REN | TB8 | RB8 | TI | RI |
| | | Type | rw | rw | rw | rw | rw | rwh | rwh | rwh |
| $99_H$ | **SBUF** **Reset: $00_H$**<br>Serial Data Buffer Register | Bit Field | VAL | | | | | | | |
| | | Type | rwh | | | | | | | |
| $A2_H$ | **EO** **Reset: $00_H$**<br>Extended Operation Register | Bit Field | 0 | | | TRAP_EN | 0 | | | DPSEL0 |
| | | Type | r | | | rw | r | | | rw |
| **$A8_H$** | **IEN0** **Reset: $00_H$**<br>Interrupt Enable Register 0 | Bit Field | EA | 0 | ET2 | ES | ET1 | EX1 | ET0 | EX0 |
| | | Type | rw | r | rw | rw | rw | rw | rw | rw |
| **$B8_H$** | **IP** **Reset: $00_H$**<br>Interrupt Priority Register | Bit Field | 0 | | PT2 | PS | PT1 | PX1 | PT0 | PX0 |
| | | Type | r | | rw | rw | rw | rw | rw | rw |
| $B9_H$ | **IPH** **Reset: $00_H$**<br>Interrupt Priority High Register | Bit Field | 0 | | PT2H | PSH | PT1H | PX1H | PT0H | PX0H |
| | | Type | r | | rw | rw | rw | rw | rw | rw |
| **$D0_H$** | **PSW** **Reset: $00_H$**<br>Program Status Word Register | Bit Field | CY | AC | F0 | RS1 | RS0 | OV | F1 | P |
| | | Type | rwh | rwh | rw | rw | rw | rwh | rw | rh |
| **$E0_H$** | **ACC** **Reset: $00_H$**<br>Accumulator Register | Bit Field | ACC7 | ACC6 | ACC5 | ACC4 | ACC3 | ACC2 | ACC1 | ACC0 |
| | | Type | rw | rw | rw | rw | rw | rw | rw | rw |
| **$E8_H$** | **IEN1** **Reset: $00_H$**<br>Interrupt Enable Register 1 | Bit Field | ECCIP3 | ECCIP2 | ECCIP1 | ECCIP0 | EXM | EX2 | ESSC | EADC |
| | | Type | rw | rw | rw | rw | rw | rw | rw | rw |
| **$F0_H$** | **B** **Reset: $00_H$**<br>B Register | Bit Field | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 |
| | | Type | rw | rw | rw | rw | rw | rw | rw | rw |
| **$F8_H$** | **IP1** **Reset: $00_H$**<br>Interrupt Priority 1 Register | Bit Field | PCCIP3 | PCCIP2 | PCCIP1 | PCCIP0 | PXM | PX2 | PSSC | PADC |
| | | Type | rw | rw | rw | rw | rw | rw | rw | rw |
| $F9_H$ | **IPH1** **Reset: $00_H$**<br>Interrupt Priority 1 High Register | Bit Field | PCCIP3H | PCCIP2H | PCCIP1H | PCCIP0H | PXMH | PX2H | PSSCH | PADCH |
| | | Type | rw | rw | rw | rw | rw | rw | rw | rw |

### 3.4.5.2 MDU Registers

The MDU SFRs can be accessed in the mapped memory area (RMAP = 1).

**Table 3-4 MDU Register Overview**

| Addr | Register Name | Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| RMAP = 1 | | | | | | | | | | |
| $B0_H$ | **MDUSTAT** **Reset: $00_H$**<br>MDU Status Register | Bit Field | 0 | | | | | BSY | IERR | IRDY |
| | | Type | r | | | | | rh | rwh | rwh |
| $B1_H$ | **MDUCON** **Reset: $00_H$**<br>MDU Control Register | Bit Field | IE | IR | RSEL | START | OPCODE | | | |
| | | Type | rw | rw | rw | rwh | rw | | | |
| $B2_H$ | **MD0** **Reset: $00_H$**<br>MDU Operand Register 0 | Bit Field | DATA | | | | | | | |
| | | Type | rw | | | | | | | |
| $B2_H$ | **MR0** **Reset: $00_H$**<br>MDU Result Register 0 | Bit Field | DATA | | | | | | | |
| | | Type | rh | | | | | | | |
| $B3_H$ | **MD1** **Reset: $00_H$**<br>MDU Operand Register 1 | Bit Field | DATA | | | | | | | |
| | | Type | rw | | | | | | | |
| $B3_H$ | **MR1** **Reset: $00_H$**<br>MDU Result Register 1 | Bit Field | DATA | | | | | | | |
| | | Type | rh | | | | | | | |
| $B4_H$ | **MD2** **Reset: $00_H$**<br>MDU Operand Register 2 | Bit Field | DATA | | | | | | | |
| | | Type | rw | | | | | | | |
| $B4_H$ | **MR2** **Reset: $00_H$**<br>MDU Result Register 2 | Bit Field | DATA | | | | | | | |
| | | Type | rh | | | | | | | |
| $B5_H$ | **MD3** **Reset: $00_H$**<br>MDU Operand Register 3 | Bit Field | DATA | | | | | | | |
| | | Type | rw | | | | | | | |
| $B5_H$ | **MR3** **Reset: $00_H$**<br>MDU Result Register 3 | Bit Field | DATA | | | | | | | |
| | | Type | rh | | | | | | | |
| $B6_H$ | **MD4** **Reset: $00_H$**<br>MDU Operand Register 4 | Bit Field | DATA | | | | | | | |
| | | Type | rw | | | | | | | |
| $B6_H$ | **MR4** **Reset: $00_H$**<br>MDU Result Register 4 | Bit Field | DATA | | | | | | | |
| | | Type | rh | | | | | | | |
| $B7_H$ | **MD5** **Reset: $00_H$**<br>MDU Operand Register 5 | Bit Field | DATA | | | | | | | |
| | | Type | rw | | | | | | | |
| $B7_H$ | **MR5** **Reset: $00_H$**<br>MDU Result Register 5 | Bit Field | DATA | | | | | | | |
| | | Type | rh | | | | | | | |

### 3.4.5.3 CORDIC Registers

The CORDIC SFRs can be accessed in the mapped memory area (RMAP = 1).

**Table 3-5 CORDIC Register Overview**

| Addr | Register Name | Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| RMAP = 1 | | | | | | | | | | |
| $9A_H$ | **CD_CORDXL Reset: $00_H$**<br>CORDIC X Data Low Byte | Bit Field | DATAL | | | | | | | |
| | | Type | rw | | | | | | | |
| $9B_H$ | **CD_CORDXH Reset: $00_H$**<br>CORDIC X Data High Byte | Bit Field | DATAH | | | | | | | |
| | | Type | rw | | | | | | | |
| $9C_H$ | **CD_CORDYL Reset: $00_H$**<br>CORDIC Y Data Low Byte | Bit Field | DATAL | | | | | | | |
| | | Type | rw | | | | | | | |
| $9D_H$ | **CD_CORDYH Reset: $00_H$**<br>CORDIC Y Data High Byte | Bit Field | DATAH | | | | | | | |
| | | Type | rw | | | | | | | |
| $9E_H$ | **CD_CORDZL Reset: $00_H$**<br>CORDIC Z Data Low Byte | Bit Field | DATAL | | | | | | | |
| | | Type | rw | | | | | | | |
| $9F_H$ | **CD_CORDZH Reset: $00_H$**<br>CORDIC Z Data High Byte | Bit Field | DATAH | | | | | | | |
| | | Type | rw | | | | | | | |
| **$A0_H$** | **CD_STATC Reset: $00_H$**<br>CORDIC Status and Data Control Register | Bit Field | KEEPZ | KEEPY | KEEPX | DMAP | INT_EN | EOC | ERROR | BSY |
| | | Type | rw | rw | rw | rw | rw | rwh | rh | rh |
| $A1_H$ | **CD_CON Reset: $00_H$**<br>CORDIC Control Register | Bit Field | MPS | | X_USIGN | ST_MODE | ROTVEC | MODE | | ST |
| | | Type | rw | | rw | rw | rw | rw | | rwh |

### 3.4.5.4 System Control Registers

The system control SFRs can be accessed in the mapped memory area (RMAP = 0).

**Table 3-6 SCU Register Overview**

| Addr | Register Name | Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| RMAP = 0 or 1 | | | | | | | | | | |
| $8F_H$ | **SYSCON0 Reset: $04_H$**<br>System Control Register 0 | Bit Field | 0 | | | IMODE | 0 | 1 | 0 | RMAP |
| | | Type | r | | | rw | r | r | r | rw |
| RMAP = 0 | | | | | | | | | | |
| $BF_H$ | **SCU_PAGE Reset: $00_H$**<br>Page Register | Bit Field | OP | | STNR | | 0 | PAGE | | |
| | | Type | w | | w | | r | rwh | | |
| RMAP = 0, PAGE 0 | | | | | | | | | | |

**Table 3-6 SCU Register Overview** (cont'd)

| Addr | Register Name | Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| $B3_H$ | **MODPISEL** **Reset: $00_H$** Peripheral Input Select Register | Bit Field | 0 | URRISH | JTAGTDIS | JTAGTCKS | EXINT2IS | EXINT1IS | EXINT0IS | URRIS |
| | | Type | r | rw | rw | rw | rw | rw | rw | rw |
| $B4_H$ | **IRCON0** **Reset: $00_H$** Interrupt Request Register 0 | Bit Field | 0 | EXINT6 | EXINT5 | EXINT4 | EXINT3 | EXINT2 | EXINT1 | EXINT0 |
| | | Type | r | rwh | rwh | rwh | rwh | rwh | rwh | rwh |
| $B5_H$ | **IRCON1** **Reset: $00_H$** Interrupt Request Register 1 | Bit Field | 0 | CANSRC2 | CANSRC1 | ADCSR1 | ADCSR0 | RIR | TIR | EIR |
| | | Type | r | rwh | rwh | rwh | rwh | rwh | rwh | rwh |
| $B6_H$ | **IRCON2** **Reset: $00_H$** Interrupt Request Register 2 | Bit Field | 0 | | | CANSRC3 | 0 | | | CANSRC0 |
| | | Type | r | | | rwh | r | | | rwh |
| $B7_H$ | **EXICON0** **Reset: $F0_H$** External Interrupt Control Register 0 | Bit Field | EXINT3 | | EXINT2 | | EXINT1 | | EXINT0 | |
| | | Type | rw | | rw | | rw | | rw | |
| $BA_H$ | **EXICON1** **Reset: $3F_H$** External Interrupt Control Register 1 | Bit Field | 0 | | EXINT6 | | EXINT5 | | EXINT4 | |
| | | Type | r | | rw | | rw | | rw | |
| $BB_H$ | **NMICON** **Reset: $00_H$** NMI Control Register | Bit Field | 0 | NMIECC | NMIVDDP | 0 | NMIOCDS | NMIFLASH | NMIPLL | NMIWDT |
| | | Type | r | rw | rw | r | rw | rw | rw | rw |
| $BC_H$ | **NMISR** **Reset: $00_H$** NMI Status Register | Bit Field | 0 | FNMIECC | FNMIVDDP | 0 | FNMIOCDS | FNMIFLASH | FNMIPLL | FNMIWDT |
| | | Type | r | rwh | rwh | r | rwh | rwh | rwh | rwh |
| $BD_H$ | **BCON** **Reset: $20_H$** Baud Rate Control Register | Bit Field | BGSEL | | NDOVEN | BRDIS | BRPRE | | | R |
| | | Type | rw | | rw | rw | rw | | | rw |
| $BE_H$ | **BG** **Reset: $00_H$** Baud Rate Timer/Reload Register | Bit Field | BR_VALUE | | | | | | | |
| | | Type | rwh | | | | | | | |
| $E9_H$ | **FDCON** **Reset: $00_H$** Fractional Divider Control Register | Bit Field | BGS | SYNEN | ERRSYN | EOFSYN | BRK | NDOV | FDM | FDEN |
| | | Type | rw | rw | rwh | rwh | rwh | rwh | rw | rw |
| $EA_H$ | **FDSTEP** **Reset: $00_H$** Fractional Divider Reload Register | Bit Field | STEP | | | | | | | |
| | | Type | rw | | | | | | | |
| $EB_H$ | **FDRES** **Reset: $00_H$** Fractional Divider Result Register | Bit Field | RESULT | | | | | | | |
| | | Type | rh | | | | | | | |
| RMAP = 0, PAGE 1 | | | | | | | | | | |
| $B3_H$ | **ID** **Reset: $49_H$** Identity Register | Bit Field | PRODID | | | | | VERID | | |
| | | Type | r | | | | | r | | |
| $B4_H$ | **PMCON0** **Reset: $80_H$** Power Mode Control Register 0 | Bit Field | VDDPWARN | WDTRST | WKRS | WKSEL | SD | PD | WS | |
| | | Type | rh | rwh | rwh | rw | rw | rwh | rw | |

**Table 3-6 SCU Register Overview** (cont'd)

| Addr | Register Name | Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| $B5_H$ | **PMCON1** **Reset: $00_H$** Power Mode Control Register 1 | Bit Field | 0 | CDC_ DIS | CAN_ DIS | MDU_ DIS | T2CCU _DIS | CCU_ DIS | SSC_ DIS | ADC_ DIS |
| | | Type | r | rw | rw | rw | rw | rw | rw | rw |
| $B6_H$ | **OSC_CON** **Reset: $XX_H$** OSC Control Register | Bit Field | PLLRD RES | PLLBY P | PLLPD | 0 | XPD | OSC SS | EORD RES | EXTO SCR |
| | | Type | rwh | rwh | rw | r | rw | rwh | rwh | rh |
| $B7_H$ | **PLL_CON** **Reset: $18_H$** PLL Control Register | Bit Field | NDIV | | | | | | PLLR | PLL_L OCK |
| | | Type | rw | | | | | | rh | rh |
| $BA_H$ | **CMCON** **Reset: $10_H$** Clock Control Register | Bit Field | KDIV | | 0 | FCCF G | CLKREL | | | |
| | | Type | rw | | r | rw | rw | | | |
| $BB_H$ | **PASSWD** **Reset: $07_H$** Password Register | Bit Field | PASS | | | | | PROT ECT_S | MODE | |
| | | Type | w | | | | | rh | rw | |
| $BE_H$ | **COCON** **Reset: $00_H$** Clock Output Control Register | Bit Field | COUTS | | TLEN | 0 | COREL | | | |
| | | Type | rw | | rw | r | rw | | | |
| $E9_H$ | **MISC_CON** **Reset: $00_H$** Miscellaneous Control Register | Bit Field | ADCE TR0_ MUX | ADCE TR1_ MUX | 0 | | | | | DFLA SHEN |
| | | Type | rw | rw | r | | | | | rwh |
| $EA_H$ | **PLL_CON1** **Reset: $20_H$** PLL Control Register 1 | Bit Field | NDIV | | | PDIV | | | | |
| | | Type | rw | | | rw | | | | |
| $EB_H$ | **CR_MISC** **Reset: $00_H$ or $01_H$** Reset Status Register | Bit Field | CCCF G | MDUC CFG | CCUC CFG | T2CCF G | 0 | | | HDRS T |
| | | Type | rw | rw | rw | rw | r | | | rwh |
| RMAP = 0, PAGE 3 | | | | | | | | | | |
| $B3_H$ | **XADDRH** **Reset: $F0_H$** On-chip XRAM Address Higher Order | Bit Field | ADDRH | | | | | | | |
| | | Type | rw | | | | | | | |
| $B4_H$ | **IRCON3** **Reset: $00_H$** Interrupt Request Register 3 | Bit Field | 0 | | CANS RC5 | CCU6 SR1 | 0 | | CANS RC4 | CCU6 SR0 |
| | | Type | r | | rwh | rwh | r | | rwh | rwh |
| $B5_H$ | **IRCON4** **Reset: $00_H$** Interrupt Request Register 4 | Bit Field | 0 | | CANS RC7 | CCU6 SR3 | 0 | | CANS RC6 | CCU6 SR2 |
| | | Type | r | | rwh | rwh | r | | rwh | rwh |
| $B6_H$ | **MODIEN** **Reset: $07_H$** Peripheral Interrupt Enable Register | Bit Field | 0 | | | CM5E N | CM4EN | RIREN | TIREN | EIREN |
| | | Type | r | | | rw | rw | rw | rw | rw |
| $B7_H$ | **MODPISEL1** **Reset: $00_H$** Peripheral Input Select Register 1 | Bit Field | EXINT6IS | | | UR1RIS | | T21EXI S | 0 | |
| | | Type | rw | | | rw | | rw | r | |

**Table 3-6 SCU Register Overview** (cont'd)

| Addr | Register Name | Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| $BA_H$ | **MODPISEL2** **Reset: $00_H$** Peripheral Input Select Register 2 | Bit Field | 0 | | | T2EXIS | T21IS | T2IS | T1IS | T0IS |
| | | Type | r | | | rw | rw | rw | rw | rw |
| $BB_H$ | **PMCON2** **Reset: $00_H$** Power Mode Control Register 2 | Bit Field | 0 | | | | | | UART1_DIS | T21_DIS |
| | | Type | r | | | | | | rw | rw |
| $BD_H$ | **MODSUSP** **Reset: $01_H$** Module Suspend Control Register | Bit Field | 0 | | CCTSUSP | T21SUSP | T2SUSP | T13SUSP | T12SUSP | WDTSUSP |
| | | Type | r | | rw | rw | rw | rw | rw | rw |
| $BE_H$ | **MODPISEL3** **Reset: $00_H$** Peripheral Input Select Register 3 | Bit Field | 0 | | CIS | | SIS | | MIS | |
| | | Type | r | | rw | | rw | | rw | |
| $EA_H$ | **MODPISEL4** **Reset: $00_H$** Peripheral Input Select Register 4 | Bit Field | 0 | | EXINT5IS | | EXINT4IS | | EXINT3IS | |
| | | Type | r | | rw | | rw | | rw | |

### 3.4.5.5 WDT Registers

The WDT SFRs can be accessed in the mapped memory area (RMAP = 1).

**Table 3-7 WDT Register Overview**

| Addr | Register Name | Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| RMAP = 1 | | | | | | | | | | |
| $BB_H$ | **WDTCON** **Reset: $00_H$** Watchdog Timer Control Register | Bit Field | 0 | | WINBEN | WDTPR | 0 | WDTEN | WDTRS | WDTIN |
| | | Type | r | | rw | rh | r | rw | rwh | rw |
| $BC_H$ | **WDTREL** **Reset: $00_H$** Watchdog Timer Reload Register | Bit Field | WDTREL | | | | | | | |
| | | Type | rw | | | | | | | |
| $BD_H$ | **WDTWINB** **Reset: $00_H$** Watchdog Window-Boundary Count Register | Bit Field | WDTWINB | | | | | | | |
| | | Type | rw | | | | | | | |
| $BE_H$ | **WDTL** **Reset: $00_H$** Watchdog Timer Register Low | Bit Field | WDT | | | | | | | |
| | | Type | rh | | | | | | | |
| $BF_H$ | **WDTH** **Reset: $00_H$** Watchdog Timer Register High | Bit Field | WDT | | | | | | | |
| | | Type | rh | | | | | | | |

### 3.4.5.6 Port Registers

The Port SFRs can be accessed in the standard memory area (RMAP = 0).

**Table 3-8 Port Register Overview**

| Addr | Register Name | Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| RMAP = 0 | | | | | | | | | | |
| $B2_H$ | **PORT_PAGE** **Reset: $00_H$**<br>Page Register | Bit Field | OP | | STNR | | 0 | PAGE | | |
| | | Type | w | | w | | r | rwh | | |
| RMAP = 0, PAGE 0 | | | | | | | | | | |
| **$80_H$** | **P0_DATA** **Reset: $00_H$**<br>P0 Data Register | Bit Field | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| | | Type | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh |
| $86_H$ | **P0_DIR** **Reset: $00_H$**<br>P0 Direction Register | Bit Field | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| | | Type | rw | rw | rw | rw | rw | rw | rw | rw |
| **$90_H$** | **P1_DATA** **Reset: $00_H$**<br>P1 Data Register | Bit Field | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| | | Type | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh |
| $91_H$ | **P1_DIR** **Reset: $00_H$**<br>P1 Direction Register | Bit Field | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| | | Type | rw | rw | rw | rw | rw | rw | rw | rw |
| $92_H$ | **P5_DATA** **Reset: $00_H$**<br>P5 Data Register | Bit Field | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| | | Type | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh |
| $93_H$ | **P5_DIR** **Reset: $00_H$**<br>P5 Direction Register | Bit Field | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| | | Type | rw | rw | rw | rw | rw | rw | rw | rw |
| **$B0_H$** | **P3_DATA** **Reset: $00_H$**<br>P3 Data Register | Bit Field | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| | | Type | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh |
| $B1_H$ | **P3_DIR** **Reset: $00_H$**<br>P3 Direction Register | Bit Field | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| | | Type | rw | rw | rw | rw | rw | rw | rw | rw |
| **$C8_H$** | **P4_DATA** **Reset: $00_H$**<br>P4 Data Register | Bit Field | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| | | Type | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh |
| $C9_H$ | **P4_DIR** **Reset: $00_H$**<br>P4 Direction Register | Bit Field | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| | | Type | rw | rw | rw | rw | rw | rw | rw | rw |
| RMAP = 0, PAGE 1 | | | | | | | | | | |
| **$80_H$** | **P0_PUDSEL** **Reset: $FF_H$**<br>P0 Pull-Up/Pull-Down Select Register | Bit Field | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| | | Type | rw | rw | rw | rw | rw | rw | rw | rw |
| $86_H$ | **P0_PUDEN** **Reset: $C4_H$**<br>P0 Pull-Up/Pull-Down Enable Register | Bit Field | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| | | Type | rw | rw | rw | rw | rw | rw | rw | rw |
| **$90_H$** | **P1_PUDSEL** **Reset: $FF_H$**<br>P1 Pull-Up/Pull-Down Select Register | Bit Field | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| | | Type | rw | rw | rw | rw | rw | rw | rw | rw |
| $91_H$ | **P1_PUDEN** **Reset: $FF_H$**<br>P1 Pull-Up/Pull-Down Enable Register | Bit Field | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| | | Type | rw | rw | rw | rw | rw | rw | rw | rw |

**Table 3-8 Port Register Overview** (cont'd)

| Addr | Register Name | Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| $92_H$ | **P5_PUDSEL Reset: $FF_H$**<br>P5 Pull-Up/Pull-Down Select Register | Bit Field | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| | | Type | rw | rw | rw | rw | rw | rw | rw | rw |
| $93_H$ | **P5_PUDEN Reset: $FF_H$**<br>P5 Pull-Up/Pull-Down Enable Register | Bit Field | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| | | Type | rw | rw | rw | rw | rw | rw | rw | rw |
| **$B0_H$** | **P3_PUDSEL Reset: $BF_H$**<br>P3 Pull-Up/Pull-Down Select Register | Bit Field | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| | | Type | rw | rw | rw | rw | rw | rw | rw | rw |
| $B1_H$ | **P3_PUDEN Reset: $40_H$**<br>P3 Pull-Up/Pull-Down Enable Register | Bit Field | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| | | Type | rw | rw | rw | rw | rw | rw | rw | rw |
| **$C8_H$** | **P4_PUDSEL Reset: $FF_H$**<br>P4 Pull-Up/Pull-Down Select Register | Bit Field | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| | | Type | rw | rw | rw | rw | rw | rw | rw | rw |
| $C9_H$ | **P4_PUDEN Reset: $04_H$**<br>P4 Pull-Up/Pull-Down Enable Register | Bit Field | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| | | Type | rw | rw | rw | rw | rw | rw | rw | rw |
| RMAP = 0, PAGE 2 | | | | | | | | | | |
| **$80_H$** | **P0_ALTSEL0 Reset: $00_H$**<br>P0 Alternate Select 0 Register | Bit Field | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| | | Type | rw | rw | rw | rw | rw | rw | rw | rw |
| $86_H$ | **P0_ALTSEL1 Reset: $00_H$**<br>P0 Alternate Select 1 Register | Bit Field | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| | | Type | rw | rw | rw | rw | rw | rw | rw | rw |
| **$90_H$** | **P1_ALTSEL0 Reset: $00_H$**<br>P1 Alternate Select 0 Register | Bit Field | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| | | Type | rw | rw | rw | rw | rw | rw | rw | rw |
| $91_H$ | **P1_ALTSEL1 Reset: $00_H$**<br>P1 Alternate Select 1 Register | Bit Field | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| | | Type | rw | rw | rw | rw | rw | rw | rw | rw |
| $92_H$ | **P5_ALTSEL0 Reset: $00_H$**<br>P5 Alternate Select 0 Register | Bit Field | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| | | Type | rw | rw | rw | rw | rw | rw | rw | rw |
| $93_H$ | **P5_ALTSEL1 Reset: $00_H$**<br>P5 Alternate Select 1 Register | Bit Field | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| | | Type | rw | rw | rw | rw | rw | rw | rw | rw |
| **$B0_H$** | **P3_ALTSEL0 Reset: $00_H$**<br>P3 Alternate Select 0 Register | Bit Field | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| | | Type | rw | rw | rw | rw | rw | rw | rw | rw |
| $B1_H$ | **P3_ALTSEL1 Reset: $00_H$**<br>P3 Alternate Select 1 Register | Bit Field | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| | | Type | rw | rw | rw | rw | rw | rw | rw | rw |
| **$C8_H$** | **P4_ALTSEL0 Reset: $00_H$**<br>P4 Alternate Select 0 Register | Bit Field | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| | | Type | rw | rw | rw | rw | rw | rw | rw | rw |
| $C9_H$ | **P4_ALTSEL1 Reset: $00_H$**<br>P4 Alternate Select 1 Register | Bit Field | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| | | Type | rw | rw | rw | rw | rw | rw | rw | rw |
| RMAP = 0, PAGE 3 | | | | | | | | | | |
| **$80_H$** | **P0_OD Reset: $00_H$**<br>P0 Open Drain Control Register | Bit Field | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| | | Type | rw | rw | rw | rw | rw | rw | rw | rw |

**Table 3-8 Port Register Overview** (cont'd)

| Addr | Register Name | Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| $86_H$ | **P0_DS** **Reset: $FF_H$** P0 Drive Strength Control Register | Bit Field | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| | | Type | rw | rw | rw | rw | rw | rw | rw | rw |
| **$90_H$** | **P1_OD** **Reset: $00_H$** P1 Open Drain Control Register | Bit Field | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| | | Type | rw | rw | rw | rw | rw | rw | rw | rw |
| $91_H$ | **P1_DS** **Reset: $FF_H$** P1 Drive Strength Control Register | Bit Field | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| | | Type | rw | rw | rw | rw | rw | rw | rw | rw |
| $92_H$ | **P5_OD** **Reset: $00_H$** P5 Open Drain Control Register | Bit Field | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| | | Type | rw | rw | rw | rw | rw | rw | rw | rw |
| $93_H$ | **P5_DS** **Reset: $FF_H$** P5 Drive Strength Control Register | Bit Field | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| | | Type | rw | rw | rw | rw | rw | rw | rw | rw |
| **$B0_H$** | **P3_OD** **Reset: $00_H$** P3 Open Drain Control Register | Bit Field | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| | | Type | rw | rw | rw | rw | rw | rw | rw | rw |
| $B1_H$ | **P3_DS** **Reset: $FF_H$** P3 Drive Strength Control Register | Bit Field | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| | | Type | rw | rw | rw | rw | rw | rw | rw | rw |
| **$C8_H$** | **P4_OD** **Reset: $00_H$** P4 Open Drain Control Register | Bit Field | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| | | Type | rw | rw | rw | rw | rw | rw | rw | rw |
| $C9_H$ | **P4_DS** **Reset: $FF_H$** P4 Drive Strength Control Register | Bit Field | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| | | Type | rw | rw | rw | rw | rw | rw | rw | rw |

### 3.4.5.7 ADC Registers

The ADC SFRs can be accessed in the standard memory area (RMAP = 0).

**Table 3-9 ADC Register Overview**

| Addr | Register Name | Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| RMAP = 0 | | | | | | | | | | |
| $D1_H$ | **ADC_PAGE** **Reset: $00_H$** Page Register | Bit Field | OP | | STNR | | 0 | PAGE | | |
| | | Type | w | | w | | r | rwh | | |
| RMAP = 0, PAGE 0 | | | | | | | | | | |
| $CA_H$ | **ADC_GLOBCTR** **Reset: $30_H$** Global Control Register | Bit Field | ANON | DW | CTC | | 0 | | | |
| | | Type | rw | rw | rw | | r | | | |
| $CB_H$ | **ADC_GLOBSTR** **Reset: $00_H$** Global Status Register | Bit Field | 0 | | CHNR | | | 0 | SAMPLE | BUSY |
| | | Type | r | | rh | | | r | rh | rh |
| $CC_H$ | **ADC_PRAR** **Reset: $00_H$** Priority and Arbitration Register | Bit Field | ASEN1 | ASEN0 | 0 | ARBM | CSM1 | PRIO1 | CSM0 | PRIO0 |
| | | Type | rw | rw | r | rw | rw | rw | rw | rw |
| $CD_H$ | **ADC_LCBR** **Reset: $B7_H$** Limit Check Boundary Register | Bit Field | BOUND1 | | | | BOUND0 | | | |
| | | Type | rw | | | | rw | | | |

**Table 3-9 ADC Register Overview** (cont'd)

| Addr | Register Name | Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| $CE_H$ | **ADC_INPCR0 Reset: $00_H$** Input Class 0 Register | Bit Field | STC | | | | | | | |
| | | Type | rw | | | | | | | |
| $CF_H$ | **ADC_ETRCR Reset: $00_H$** External Trigger Control Register | Bit Field | SYNEN1 | SYNEN0 | ETRSEL1 | | | ETRSEL0 | | |
| | | Type | rw | rw | rw | | | rw | | |
| RMAP = 0, PAGE 1 | | | | | | | | | | |
| $CA_H$ | **ADC_CHCTR0 Reset: $00_H$** Channel Control Register 0 | Bit Field | 0 | LCC | | | 0 | | RESRSEL | |
| | | Type | r | rw | | | r | | rw | |
| $CB_H$ | **ADC_CHCTR1 Reset: $00_H$** Channel Control Register 1 | Bit Field | 0 | LCC | | | 0 | | RESRSEL | |
| | | Type | r | rw | | | r | | rw | |
| $CC_H$ | **ADC_CHCTR2 Reset: $00_H$** Channel Control Register 2 | Bit Field | 0 | LCC | | | 0 | | RESRSEL | |
| | | Type | r | rw | | | r | | rw | |
| $CD_H$ | **ADC_CHCTR3 Reset: $00_H$** Channel Control Register 3 | Bit Field | 0 | LCC | | | 0 | | RESRSEL | |
| | | Type | r | rw | | | r | | rw | |
| $CE_H$ | **ADC_CHCTR4 Reset: $00_H$** Channel Control Register 4 | Bit Field | 0 | LCC | | | 0 | | RESRSEL | |
| | | Type | r | rw | | | r | | rw | |
| $CF_H$ | **ADC_CHCTR5 Reset: $00_H$** Channel Control Register 5 | Bit Field | 0 | LCC | | | 0 | | RESRSEL | |
| | | Type | r | rw | | | r | | rw | |
| $D2_H$ | **ADC_CHCTR6 Reset: $00_H$** Channel Control Register 6 | Bit Field | 0 | LCC | | | 0 | | RESRSEL | |
| | | Type | r | rw | | | r | | rw | |
| $D3_H$ | **ADC_CHCTR7 Reset: $00_H$** Channel Control Register 7 | Bit Field | 0 | LCC | | | 0 | | RESRSEL | |
| | | Type | r | rw | | | r | | rw | |
| RMAP = 0, PAGE 2 | | | | | | | | | | |
| $CA_H$ | **ADC_RESR0L Reset: $00_H$** Result Register 0 Low | Bit Field | RESULT | | 0 | VF | DRC | CHNR | | |
| | | Type | rh | | r | rh | rh | rh | | |
| $CB_H$ | **ADC_RESR0H Reset: $00_H$** Result Register 0 High | Bit Field | RESULT | | | | | | | |
| | | Type | rh | | | | | | | |
| $CC_H$ | **ADC_RESR1L Reset: $00_H$** Result Register 1 Low | Bit Field | RESULT | | 0 | VF | DRC | CHNR | | |
| | | Type | rh | | r | rh | rh | rh | | |
| $CD_H$ | **ADC_RESR1H Reset: $00_H$** Result Register 1 High | Bit Field | RESULT | | | | | | | |
| | | Type | rh | | | | | | | |
| $CE_H$ | **ADC_RESR2L Reset: $00_H$** Result Register 2 Low | Bit Field | RESULT | | 0 | VF | DRC | CHNR | | |
| | | Type | rh | | r | rh | rh | rh | | |
| $CF_H$ | **ADC_RESR2H Reset: $00_H$** Result Register 2 High | Bit Field | RESULT | | | | | | | |
| | | Type | rh | | | | | | | |
| $D2_H$ | **ADC_RESR3L Reset: $00_H$** Result Register 3 Low | Bit Field | RESULT | | 0 | VF | DRC | CHNR | | |
| | | Type | rh | | r | rh | rh | rh | | |
| $D3_H$ | **ADC_RESR3H Reset: $00_H$** Result Register 3 High | Bit Field | RESULT | | | | | | | |
| | | Type | rh | | | | | | | |

**Table 3-9 ADC Register Overview** (cont'd)

| Addr | Register Name | Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| RMAP = 0, PAGE 3 | | | | | | | | | | |
| $CA_H$ | **ADC_RESRA0L Reset: $00_H$** Result Register 0, View A Low | Bit Field | RESULT | | | VF | DRC | CHNR | | |
| | | Type | rh | | | rh | rh | rh | | |
| $CB_H$ | **ADC_RESRA0H Reset: $00_H$** Result Register 0, View A High | Bit Field | RESULT | | | | | | | |
| | | Type | rh | | | | | | | |
| $CC_H$ | **ADC_RESRA1L Reset: $00_H$** Result Register 1, View A Low | Bit Field | RESULT | | | VF | DRC | CHNR | | |
| | | Type | rh | | | rh | rh | rh | | |
| $CD_H$ | **ADC_RESRA1H Reset: $00_H$** Result Register 1, View A High | Bit Field | RESULT | | | | | | | |
| | | Type | rh | | | | | | | |
| $CE_H$ | **ADC_RESRA2L Reset: $00_H$** Result Register 2, View A Low | Bit Field | RESULT | | | VF | DRC | CHNR | | |
| | | Type | rh | | | rh | rh | rh | | |
| $CF_H$ | **ADC_RESRA2H Reset: $00_H$** Result Register 2, View A High | Bit Field | RESULT | | | | | | | |
| | | Type | rh | | | | | | | |
| $D2_H$ | **ADC_RESRA3L Reset: $00_H$** Result Register 3, View A Low | Bit Field | RESULT | | | VF | DRC | CHNR | | |
| | | Type | rh | | | rh | rh | rh | | |
| $D3_H$ | **ADC_RESRA3H Reset: $00_H$** Result Register 3, View A High | Bit Field | RESULT | | | | | | | |
| | | Type | rh | | | | | | | |
| RMAP = 0, PAGE 4 | | | | | | | | | | |
| $CA_H$ | **ADC_RCR0 Reset: $00_H$** Result Control Register 0 | Bit Field | VFCTR | WFR | 0 | IEN | 0 | | | DRCTR |
| | | Type | rw | rw | r | rw | r | | | rw |
| $CB_H$ | **ADC_RCR1 Reset: $00_H$** Result Control Register 1 | Bit Field | VFCTR | WFR | 0 | IEN | 0 | | | DRCTR |
| | | Type | rw | rw | r | rw | r | | | rw |
| $CC_H$ | **ADC_RCR2 Reset: $00_H$** Result Control Register 2 | Bit Field | VFCTR | WFR | 0 | IEN | 0 | | | DRCTR |
| | | Type | rw | rw | r | rw | r | | | rw |
| $CD_H$ | **ADC_RCR3 Reset: $00_H$** Result Control Register 3 | Bit Field | VFCTR | WFR | 0 | IEN | 0 | | | DRCTR |
| | | Type | rw | rw | r | rw | r | | | rw |
| $CE_H$ | **ADC_VFCR Reset: $00_H$** Valid Flag Clear Register | Bit Field | 0 | | | | VFC3 | VFC2 | VFC1 | VFC0 |
| | | Type | r | | | | w | w | w | w |
| RMAP = 0, PAGE 5 | | | | | | | | | | |
| $CA_H$ | **ADC_CHINFR Reset: $00_H$** Channel Interrupt Flag Register | Bit Field | CHINF7 | CHINF6 | CHINF5 | CHINF4 | CHINF3 | CHINF2 | CHINF1 | CHINF0 |
| | | Type | rh | rh | rh | rh | rh | rh | rh | rh |
| $CB_H$ | **ADC_CHINCR Reset: $00_H$** Channel Interrupt Clear Register | Bit Field | CHINC7 | CHINC6 | CHINC5 | CHINC4 | CHINC3 | CHINC2 | CHINC1 | CHINC0 |
| | | Type | w | w | w | w | w | w | w | w |

**Table 3-9 ADC Register Overview** (cont'd)

| Addr | Register Name | Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| $CC_H$ | **ADC_CHINSR Reset: $00_H$** Channel Interrupt Set Register | Bit Field | CHINS 7 | CHINS 6 | CHINS 5 | CHINS 4 | CHINS 3 | CHINS 2 | CHINS 1 | CHINS 0 |
| | | Type | w | w | w | w | w | w | w | w |
| $CD_H$ | **ADC_CHINPR Reset: $00_H$** Channel Interrupt Node Pointer Register | Bit Field | CHINP 7 | CHINP 6 | CHINP 5 | CHINP 4 | CHINP 3 | CHINP 2 | CHINP 1 | CHINP 0 |
| | | Type | rw | rw | rw | rw | rw | rw | rw | rw |
| $CE_H$ | **ADC_EVINFR Reset: $00_H$** Event Interrupt Flag Register | Bit Field | EVINF 7 | EVINF 6 | EVINF 5 | EVINF 4 | 0 | | EVINF 1 | EVINF 0 |
| | | Type | rh | rh | rh | rh | r | | rh | rh |
| $CF_H$ | **ADC_EVINCR Reset: $00_H$** Event Interrupt Clear Flag Register | Bit Field | EVINC 7 | EVINC 6 | EVINC 5 | EVINC 4 | 0 | | EVINC 1 | EVINC 0 |
| | | Type | w | w | w | w | r | | w | w |
| $D2_H$ | **ADC_EVINSR Reset: $00_H$** Event Interrupt Set Flag Register | Bit Field | EVINS 7 | EVINS 6 | EVINS 5 | EVINS 4 | 0 | | EVINS 1 | EVINS 0 |
| | | Type | w | w | w | w | r | | w | w |
| $D3_H$ | **ADC_EVINPR Reset: $00_H$** Event Interrupt Node Pointer Register | Bit Field | EVINP 7 | EVINP 6 | EVINP 5 | EVINP 4 | 0 | | EVINP 1 | EVINP 0 |
| | | Type | rw | rw | rw | rw | r | | rw | rw |
| RMAP = 0, PAGE 6 | | | | | | | | | | |
| $CA_H$ | **ADC_CRCR1 Reset: $00_H$** Conversion Request Control Register 1 | Bit Field | CH7 | CH6 | CH5 | CH4 | 0 | | | |
| | | Type | rwh | rwh | rwh | rwh | r | | | |
| $CB_H$ | **ADC_CRPR1 Reset: $00_H$** Conversion Request Pending Register 1 | Bit Field | CHP7 | CHP6 | CHP5 | CHP4 | 0 | | | |
| | | Type | rwh | rwh | rwh | rwh | r | | | |
| $CC_H$ | **ADC_CRMR1 Reset: $00_H$** Conversion Request Mode Register 1 | Bit Field | Rsv | LDEV | CLRP ND | SCAN | ENSI | ENTR | 0 | ENGT |
| | | Type | r | w | w | rw | rw | rw | r | rw |
| $CD_H$ | **ADC_QMR0 Reset: $00_H$** Queue Mode Register 0 | Bit Field | CEV | TREV | FLUS H | CLRV | 0 | ENTR | 0 | ENGT |
| | | Type | w | w | w | w | r | rw | r | rw |
| $CE_H$ | **ADC_QSR0 Reset: $20_H$** Queue Status Register 0 | Bit Field | Rsv | 0 | EMPT Y | EV | 0 | | FILL | |
| | | Type | r | r | rh | rh | r | | rh | |
| $CF_H$ | **ADC_Q0R0 Reset: $00_H$** Queue 0 Register 0 | Bit Field | EXTR | ENSI | RF | V | 0 | REQCHNR | | |
| | | Type | rh | rh | rh | rh | r | rh | | |
| $D2_H$ | **ADC_QBUR0 Reset: $00_H$** Queue Backup Register 0 | Bit Field | EXTR | ENSI | RF | V | 0 | REQCHNR | | |
| | | Type | rh | rh | rh | rh | r | rh | | |
| $D2_H$ | **ADC_QINR0 Reset: $00_H$** Queue Input Register 0 | Bit Field | EXTR | ENSI | RF | 0 | | REQCHNR | | |
| | | Type | w | w | w | r | | w | | |

### 3.4.5.8 Timer 2 Compare/Capture Unit Registers

The Timer 2 Compare/Capture Unit SFRs can be accessed in the standard memory area (RMAP = 0).

**Table 3-10 T2CCU Register Overview**

| Addr | Register Name | Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| RMAP = 0 | | | | | | | | | | |
| $C7_H$ | **T2_PAGE** **Reset: $00_H$**<br>Page Register | Bit Field | OP | | STNR | | 0 | PAGE | | |
| | | Type | w | | w | | r | rwh | | |
| RMAP = 0, PAGE 0 | | | | | | | | | | |
| **$C0_H$** | **T2_T2CON** **Reset: $00_H$**<br>Timer 2 Control Register | Bit Field | TF2 | EXF2 | 0 | | EXEN2 | TR2 | C/$\overline{T2}$ | $\underline{CP/RL2}$ |
| | | Type | rwh | rwh | r | | rw | rwh | rw | rw |
| $C1_H$ | **T2_T2MOD** **Reset: $00_H$**<br>Timer 2 Mode Register | Bit Field | T2REGS | T2RHEN | EDGESEL | PREN | T2PRE | | | DCEN |
| | | Type | rw | rw | rw | rw | rw | | | rw |
| $C2_H$ | **T2_RC2L** **Reset: $00_H$**<br>Timer 2 Reload/Capture Register Low | Bit Field | RC2 | | | | | | | |
| | | Type | rwh | | | | | | | |
| $C3_H$ | **T2_RC2H** **Reset: $00_H$**<br>Timer 2 Reload/Capture Register High | Bit Field | RC2 | | | | | | | |
| | | Type | rwh | | | | | | | |
| $C4_H$ | **T2_T2L** **Reset: $00_H$**<br>Timer 2 Register Low | Bit Field | THL2 | | | | | | | |
| | | Type | rwh | | | | | | | |
| $C5_H$ | **T2_T2H** **Reset: $00_H$**<br>Timer 2 Register High | Bit Field | THL2 | | | | | | | |
| | | Type | rwh | | | | | | | |
| $C6_H$ | **T2_T2CON1** **Reset: $03_H$**<br>Timer 2 Control Register 1 | Bit Field | 0 | | | | | | TF2EN | EXF2EN |
| | | Type | r | | | | | | rw | rw |
| RMAP = 0, PAGE 1 | | | | | | | | | | |
| $C0_H$ | **T2CCU_CCEN** **Reset: $00_H$**<br>T2CCU Capture/Compare Enable Register | Bit Field | CCM3 | | CCM2 | | CCM1 | | CCM0 | |
| | | Type | rw | | rw | | rw | | rw | |
| $C1_H$ | **T2CCU_CCTBSELReset: $00_H$**<br>T2CCU Capture/Compare Time Base Select Register | Bit Field | CASC | CCTTOV | CCTB5 | CCTB4 | CCTB3 | CCTB2 | CCTB1 | CCTB0 |
| | | Type | rw | rwh | rw | rw | rw | rw | rw | rw |
| $C2_H$ | **T2CCU_CCTRELLReset: $00_H$**<br>T2CCU Capture/Compare Timer Reload Register Low | Bit Field | CCTREL | | | | | | | |
| | | Type | rw | | | | | | | |
| $C3_H$ | **T2CCU_CCTRELHReset: $00_H$**<br>T2CCU Capture/Compare Timer Reload Register High | Bit Field | CCTREL | | | | | | | |
| | | Type | rw | | | | | | | |
| $C4_H$ | **T2CCU_CCTL** **Reset: $00_H$**<br>T2CCU Capture/Compare Timer Register Low | Bit Field | CCT | | | | | | | |
| | | Type | rwh | | | | | | | |

**Table 3-10 T2CCU Register Overview** (cont'd)

| Addr | Register Name | Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| $C5_H$ | **T2CCU_CCTH Reset: $00_H$** T2CCU Capture/Compare Timer Register High | Bit Field | CCT | | | | | | | |
| | | Type | rwh | | | | | | | |
| $C6_H$ | **T2CCU_CCTCON Reset: $00_H$** T2CCU CaptureCcompare Timer Control Register | Bit Field | CCTPRE | | | | CCTOVF | CCTOVEN | TIMSYN | CCTST |
| | | Type | rw | | | | rwh | rw | rw | rw |
| RMAP = 0, PAGE 2 | | | | | | | | | | |
| $C0_H$ | **T2CCU_COSHDW Reset: $00_H$** T2CCU Capture/compare Enable Register | Bit Field | ENSHDW | TXOV | COOUT5 | COOUT4 | COOUT3 | COOUT2 | COOUT1 | COOUT0 |
| | | Type | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh |
| $C1_H$ | **T2CCU_CC0L Reset: $00_H$** T2CCU Capture/Compare Register 0 Low | Bit Field | CCVALL | | | | | | | |
| | | Type | rwh | | | | | | | |
| $C2_H$ | **T2CCU_CC0H Reset: $00_H$** T2CCU Capture/compare Register 0 High | Bit Field | CCVALH | | | | | | | |
| | | Type | rwh | | | | | | | |
| $C3_H$ | **T2CCU_CC1L Reset: $00_H$** T2CCU Capture/compare Register 1 Low | Bit Field | CCVALL | | | | | | | |
| | | Type | rwh | | | | | | | |
| $C4_H$ | **T2CCU_CC1H Reset: $00_H$** T2CCU Capture/compare Register 1 High | Bit Field | CCVALH | | | | | | | |
| | | Type | rwh | | | | | | | |
| $C5_H$ | **T2CCU_CC2L Reset: $00_H$** T2CCU Capture/compare Register 2 Low | Bit Field | CCVALL | | | | | | | |
| | | Type | rwh | | | | | | | |
| $C6_H$ | **T2CCU_CC2H Reset: $00_H$** T2CCU Capture/compare Register 2 High | Bit Field | CCVALH | | | | | | | |
| | | Type | rwh | | | | | | | |
| RMAP = 0, PAGE 3 | | | | | | | | | | |
| $C0_H$ | **T2CCU_COCON Reset: $00_H$** T2CCU Compare Control Register | Bit Field | CCM5 | CCM4 | CM5F | CM4F | POLB | POLA | COMOD | |
| | | Type | rw | rw | rwh | rwh | rw | rw | rw | |
| $C1_H$ | **T2CCU_CC3L Reset: $00_H$** T2CCU Capture/compare Register 3 Low | Bit Field | CCVALL | | | | | | | |
| | | Type | rwh | | | | | | | |
| $C2_H$ | **T2CCU_CC3H Reset: $00_H$** T2CCU Capture/compare Register 3 High | Bit Field | CCVALH | | | | | | | |
| | | Type | rwh | | | | | | | |
| $C3_H$ | **T2CCU_CC4L Reset: $00_H$** T2CCU Capture/compare Register 4 Low | Bit Field | CCVALL | | | | | | | |
| | | Type | rwh | | | | | | | |
| $C4_H$ | **T2CCU_CC4H Reset: $00_H$** T2CCU Capture/compare Register 4 High | Bit Field | CCVALH | | | | | | | |
| | | Type | rwh | | | | | | | |
| $C5_H$ | **T2CCU_CC5L Reset: $00_H$** T2CCU Capture/compare Register 5 Low | Bit Field | CCVALL | | | | | | | |
| | | Type | rwh | | | | | | | |
| $C6_H$ | **T2CCU_CC5H Reset: $00_H$** T2CCU Capture/compare Register 5 High | Bit Field | CCVALH | | | | | | | |
| | | Type | rwh | | | | | | | |

**Table 3-10 T2CCU Register Overview** (cont'd)

| Addr | Register Name | Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| RMAP = 0, PAGE 4 | | | | | | | | | | |
| $C2_H$ | **T2CCU_CCTDTCLReset: $00_H$** T2CCU Capture/Compare Timer Dead-Time Control Register Low | Bit Field | DTM | | | | | | | |
| | | Type | rw | | | | | | | |
| $C3_H$ | **T2CCU_CCTDTCHReset: $00_H$** T2CCU Capture/Compare Timer Dead-Time Control Register High | Bit Field | DTRES | DTR2 | DTR1 | DTR0 | DTLEV | DTE2 | DTE1 | DTE0 |
| | | Type | rwh | rh | rh | rh | rw | rw | rw | rw |

### 3.4.5.9 Timer 21 Registers

The Timer 21 SFRs can be accessed in the mapped memory area (RMAP = 1).

**Table 3-11 T21 Register Overview**

| Addr | Register Name | Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| RMAP = 1 | | | | | | | | | | |
| **$C0_H$** | **T21_T2CON Reset: $00_H$** Timer 2 Control Register | Bit Field | TF2 | EXF2 | 0 | | EXEN2 | TR2 | C/$\overline{T2}$ | CP/$\overline{RL2}$ |
| | | Type | rwh | rwh | r | | rw | rwh | rw | rw |
| $C1_H$ | **T21_T2MOD Reset: $00_H$** Timer 2 Mode Register | Bit Field | T2REGS | T2RHEN | EDGESEL | PREN | T2PRE | | | DCEN |
| | | Type | rw | rw | rw | rw | rw | rw | rw | rw |
| $C2_H$ | **T21_RC2L Reset: $00_H$** Timer 2 Reload/Capture Register Low | Bit Field | RC2 | | | | | | | |
| | | Type | rwh | | | | | | | |
| $C3_H$ | **T21_RC2H Reset: $00_H$** Timer 2 Reload/Capture Register High | Bit Field | RC2 | | | | | | | |
| | | Type | rwh | | | | | | | |
| $C4_H$ | **T21_T2L Reset: $00_H$** Timer 2 Register Low | Bit Field | THL2 | | | | | | | |
| | | Type | rwh | | | | | | | |
| $C5_H$ | **T21_T2H Reset: $00_H$** Timer 2 Register High | Bit Field | THL2 | | | | | | | |
| | | Type | rwh | | | | | | | |
| $C6_H$ | **T21_T2CON1 Reset: $03_H$** Timer 2 Control Register 1 | Bit Field | 0 | | | | | | TF2EN | EXF2EN |
| | | Type | r | | | | | | rw | rw |

### 3.4.5.10 CCU6 Registers

The CCU6 SFRs can be accessed in the standard memory area (RMAP = 0).

**Table 3-12 CCU6 Register Overview**

| Addr | Register Name | Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| RMAP = 0 | | | | | | | | | | |
| $A3_H$ | **CCU6_PAGE** **Reset: $00_H$** Page Register | Bit Field | OP | | STNR | | 0 | PAGE | | |
| | | Type | w | | w | | r | rwh | | |
| RMAP = 0, PAGE 0 | | | | | | | | | | |
| $9A_H$ | **CCU6_CC63SRL** **Reset: $00_H$** Capture/Compare Shadow Register for Channel CC63 Low | Bit Field | CC63SL | | | | | | | |
| | | Type | rw | | | | | | | |
| $9B_H$ | **CCU6_CC63SRH** **Reset: $00_H$** Capture/Compare Shadow Register for Channel CC63 High | Bit Field | CC63SH | | | | | | | |
| | | Type | rw | | | | | | | |
| $9C_H$ | **CCU6_TCTR4L** **Reset: $00_H$** Timer Control Register 4 Low | Bit Field | T12 STD | T12 STR | 0 | | DT RES | T12 RES | T12R S | T12R R |
| | | Type | w | w | r | | w | w | w | w |
| $9D_H$ | **CCU6_TCTR4H** **Reset: $00_H$** Timer Control Register 4 High | Bit Field | T13 STD | T13 STR | 0 | | | T13 RES | T13R S | T13R R |
| | | Type | w | w | r | | | w | w | w |
| $9E_H$ | **CCU6_MCMOUTSL** **Reset: $00_H$** Multi-Channel Mode Output Shadow Register Low | Bit Field | STRM CM | 0 | MCMPS | | | | | |
| | | Type | w | r | rw | | | | | |
| $9F_H$ | **CCU6_MCMOUTSH** **Reset: $00_H$** Multi-Channel Mode Output Shadow Register High | Bit Field | STRH P | 0 | CURHS | | | EXPHS | | |
| | | Type | w | r | rw | | | rw | | |
| $A4_H$ | **CCU6_ISRL** **Reset: $00_H$** Capture/Compare Interrupt Status Reset Register Low | Bit Field | RT12 PM | RT12 OM | RCC6 2F | RCC6 2R | RCC6 1F | RCC6 1R | RCC6 0F | RCC6 0R |
| | | Type | w | w | w | w | w | w | w | w |
| $A5_H$ | **CCU6_ISRH** **Reset: $00_H$** Capture/Compare Interrupt Status Reset Register High | Bit Field | RSTR | RIDLE | RWH E | RCHE | 0 | RTRP F | RT13 PM | RT13 CM |
| | | Type | w | w | w | w | r | w | w | w |
| $A6_H$ | **CCU6_CMPMODIFL** **Reset: $00_H$** Compare State Modification Register Low | Bit Field | 0 | MCC6 3S | 0 | | | MCC6 2S | MCC6 1S | MCC6 0S |
| | | Type | r | w | r | | | w | w | w |
| $A7_H$ | **CCU6_CMPMODIFH** **Reset: $00_H$** Compare State Modification Register High | Bit Field | 0 | MCC6 3R | 0 | | | MCC6 2R | MCC6 1R | MCC6 0R |
| | | Type | r | w | r | | | w | w | w |
| $FA_H$ | **CCU6_CC60SRL** **Reset: $00_H$** Capture/Compare Shadow Register for Channel CC60 Low | Bit Field | CC60SL | | | | | | | |
| | | Type | rwh | | | | | | | |
| $FB_H$ | **CCU6_CC60SRH** **Reset: $00_H$** Capture/Compare Shadow Register for Channel CC60 High | Bit Field | CC60SH | | | | | | | |
| | | Type | rwh | | | | | | | |

**Table 3-12 CCU6 Register Overview** (cont'd)

| Addr | Register Name | Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| $FC_H$ | **CCU6_CC61SRL Reset: $00_H$** Capture/Compare Shadow Register for Channel CC61 Low | Bit Field | CC61SL | | | | | | | |
| | | Type | rwh | | | | | | | |
| $FD_H$ | **CCU6_CC61SRH Reset: $00_H$** Capture/Compare Shadow Register for Channel CC61 High | Bit Field | CC61SH | | | | | | | |
| | | Type | rwh | | | | | | | |
| $FE_H$ | **CCU6_CC62SRL Reset: $00_H$** Capture/Compare Shadow Register for Channel CC62 Low | Bit Field | CC62SL | | | | | | | |
| | | Type | rwh | | | | | | | |
| $FF_H$ | **CCU6_CC62SRH Reset: $00_H$** Capture/Compare Shadow Register for Channel CC62 High | Bit Field | CC62SH | | | | | | | |
| | | Type | rwh | | | | | | | |
| RMAP = 0, PAGE 1 | | | | | | | | | | |
| $9A_H$ | **CCU6_CC63RL Reset: $00_H$** Capture/Compare Register for Channel CC63 Low | Bit Field | CC63VL | | | | | | | |
| | | Type | rh | | | | | | | |
| $9B_H$ | **CCU6_CC63RH Reset: $00_H$** Capture/Compare Register for Channel CC63 High | Bit Field | CC63VH | | | | | | | |
| | | Type | rh | | | | | | | |
| $9C_H$ | **CCU6_T12PRL Reset: $00_H$** Timer T12 Period Register Low | Bit Field | T12PVL | | | | | | | |
| | | Type | rwh | | | | | | | |
| $9D_H$ | **CCU6_T12PRH Reset: $00_H$** Timer T12 Period Register High | Bit Field | T12PVH | | | | | | | |
| | | Type | rwh | | | | | | | |
| $9E_H$ | **CCU6_T13PRL Reset: $00_H$** Timer T13 Period Register Low | Bit Field | T13PVL | | | | | | | |
| | | Type | rwh | | | | | | | |
| $9F_H$ | **CCU6_T13PRH Reset: $00_H$** Timer T13 Period Register High | Bit Field | T13PVH | | | | | | | |
| | | Type | rwh | | | | | | | |
| $A4_H$ | **CCU6_T12DTCL Reset: $00_H$** Dead-Time Control Register for Timer T12 Low | Bit Field | DTM | | | | | | | |
| | | Type | rw | | | | | | | |
| $A5_H$ | **CCU6_T12DTCH Reset: $00_H$** Dead-Time Control Register for Timer T12 High | Bit Field | 0 | DTR2 | DTR1 | DTR0 | 0 | DTE2 | DTE1 | DTE0 |
| | | Type | r | rh | rh | rh | r | rw | rw | rw |
| $A6_H$ | **CCU6_TCTR0L Reset: $00_H$** Timer Control Register 0 Low | Bit Field | CTM | CDIR | STE12 | T12R | T12PRE | T12CLK | | |
| | | Type | rw | rh | rh | rh | rw | rw | | |
| $A7_H$ | **CCU6_TCTR0H Reset: $00_H$** Timer Control Register 0 High | Bit Field | 0 | | STE13 | T13R | T13PRE | T13CLK | | |
| | | Type | r | | rh | rh | rw | rw | | |
| $FA_H$ | **CCU6_CC60RL Reset: $00_H$** Capture/Compare Register for Channel CC60 Low | Bit Field | CC60VL | | | | | | | |
| | | Type | rh | | | | | | | |
| $FB_H$ | **CCU6_CC60RH Reset: $00_H$** Capture/Compare Register for Channel CC60 High | Bit Field | CC60VH | | | | | | | |
| | | Type | rh | | | | | | | |
| $FC_H$ | **CCU6_CC61RL Reset: $00_H$** Capture/Compare Register for Channel CC61 Low | Bit Field | CC61VL | | | | | | | |
| | | Type | rh | | | | | | | |

**Table 3-12 CCU6 Register Overview** (cont'd)

| Addr | Register Name | Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| $FD_H$ | **CCU6_CC61RH** **Reset: $00_H$** Capture/Compare Register for Channel CC61 High | Bit Field | CC61VH | | | | | | | |
| | | Type | rh | | | | | | | |
| $FE_H$ | **CCU6_CC62RL** **Reset: $00_H$** Capture/Compare Register for Channel CC62 Low | Bit Field | CC62VL | | | | | | | |
| | | Type | rh | | | | | | | |
| $FF_H$ | **CCU6_CC62RH** **Reset: $00_H$** Capture/Compare Register for Channel CC62 High | Bit Field | CC62VH | | | | | | | |
| | | Type | rh | | | | | | | |
| RMAP = 0, PAGE 2 | | | | | | | | | | |
| $9A_H$ | **CCU6_T12MSELL** **Reset: $00_H$** T12 Capture/Compare Mode Select Register Low | Bit Field | MSEL61 | | | | MSEL60 | | | |
| | | Type | rw | | | | rw | | | |
| $9B_H$ | **CCU6_T12MSELH** **Reset: $00_H$** T12 Capture/Compare Mode Select Register High | Bit Field | DBYP | HSYNC | | | MSEL62 | | | |
| | | Type | rw | rw | | | rw | | | |
| $9C_H$ | **CCU6_IENL** **Reset: $00_H$** Capture/Compare Interrupt Enable Register Low | Bit Field | ENT12PM | ENT12OM | ENCC62F | ENCC62R | ENCC61F | ENCC61R | ENCC60F | ENCC60R |
| | | Type | rw | rw | rw | rw | rw | rw | rw | rw |
| $9D_H$ | **CCU6_IENH** **Reset: $00_H$** Capture/Compare Interrupt Enable Register High | Bit Field | ENSTR | ENIDLE | ENWHE | ENCHE | 0 | ENTRPF | ENT13PM | ENT13CM |
| | | Type | rw | rw | rw | rw | r | rw | rw | rw |
| $9E_H$ | **CCU6_INPL** **Reset: $40_H$** Capture/Compare Interrupt Node Pointer Register Low | Bit Field | INPCHE | | INPCC62 | | INPCC61 | | INPCC60 | |
| | | Type | rw | | rw | | rw | | rw | |
| $9F_H$ | **CCU6_INPH** **Reset: $39_H$** Capture/Compare Interrupt Node Pointer Register High | Bit Field | 0 | | INPT13 | | INPT12 | | INPERR | |
| | | Type | r | | rw | | rw | | rw | |
| $A4_H$ | **CCU6_ISSL** **Reset: $00_H$** Capture/Compare Interrupt Status Set Register Low | Bit Field | ST12PM | ST12OM | SCC62F | SCC62R | SCC61F | SCC61R | SCC60F | SCC60R |
| | | Type | w | w | w | w | w | w | w | w |
| $A5_H$ | **CCU6_ISSH** **Reset: $00_H$** Capture/Compare Interrupt Status Set Register High | Bit Field | SSTR | SIDLE | SWHE | SCHE | SWHC | STRPF | ST13PM | ST13CM |
| | | Type | w | w | w | w | w | w | w | w |
| $A6_H$ | **CCU6_PSLR** **Reset: $00_H$** Passive State Level Register | Bit Field | PSL63 | 0 | PSL | | | | | |
| | | Type | rwh | r | rwh | | | | | |
| $A7_H$ | **CCU6_MCMCTR** **Reset: $00_H$** Multi-Channel Mode Control Register | Bit Field | 0 | | SWSYN | | 0 | SWSEL | | |
| | | Type | r | | rw | | r | rw | | |
| $FA_H$ | **CCU6_TCTR2L** **Reset: $00_H$** Timer Control Register 2 Low | Bit Field | 0 | T13TED | | T13TEC | | | T13SSC | T12SSC |
| | | Type | r | rw | | rw | | | rw | rw |
| $FB_H$ | **CCU6_TCTR2H** **Reset: $00_H$** Timer Control Register 2 High | Bit Field | 0 | | | | T13RSEL | | T12RSEL | |
| | | Type | r | | | | rw | | rw | |
| $FC_H$ | **CCU6_MODCTRL** **Reset: $00_H$** Modulation Control Register Low | Bit Field | MCMEN | 0 | T12MODEN | | | | | |
| | | Type | rw | r | rw | | | | | |

**Table 3-12 CCU6 Register Overview** (cont'd)

| Addr | Register Name | Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| $FD_H$ | **CCU6_MODCTRH Reset: $00_H$** Modulation Control Register High | Bit Field | ECT13O | 0 | T13MODEN | | | | | |
| | | Type | rw | r | rw | | | | | |
| $FE_H$ | **CCU6_TRPCTRL Reset: $00_H$** Trap Control Register Low | Bit Field | 0 | | | | | TRPM2 | TRPM1 | TRPM0 |
| | | Type | r | | | | | rw | rw | rw |
| $FF_H$ | **CCU6_TRPCTRH Reset: $00_H$** Trap Control Register High | Bit Field | TRPPEN | TRPEN13 | TRPEN | | | | | |
| | | Type | rw | rw | rw | | | | | |
| RMAP = 0, PAGE 3 | | | | | | | | | | |
| $9A_H$ | **CCU6_MCMOUTL Reset: $00_H$** Multi-Channel Mode Output Register Low | Bit Field | 0 | R | MCMP | | | | | |
| | | Type | r | rh | rh | | | | | |
| $9B_H$ | **CCU6_MCMOUTH Reset: $00_H$** Multi-Channel Mode Output Register High | Bit Field | 0 | | CURH | | | EXPH | | |
| | | Type | r | | rh | | | rh | | |
| $9C_H$ | **CCU6_ISL Reset: $00_H$** Capture/Compare Interrupt Status Register Low | Bit Field | T12PM | T12OM | ICC62F | ICC62R | ICC61F | ICC61R | ICC60F | ICC60R |
| | | Type | rh | rh | rh | rh | rh | rh | rh | rh |
| $9D_H$ | **CCU6_ISH Reset: $00_H$** Capture/Compare Interrupt Status Register High | Bit Field | STR | IDLE | WHE | CHE | TRPS | TRPF | T13PM | T13CM |
| | | Type | rh | rh | rh | rh | rh | rh | rh | rh |
| $9E_H$ | **CCU6_PISEL0L Reset: $00_H$** Port Input Select Register 0 Low | Bit Field | ISTRP | | ISCC62 | | ISCC61 | | ISCC60 | |
| | | Type | rw | | rw | | rw | | rw | |
| $9F_H$ | **CCU6_PISEL0H Reset: $00_H$** Port Input Select Register 0 High | Bit Field | IST12HR | | ISPOS2 | | ISPOS1 | | ISPOS0 | |
| | | Type | rw | | rw | | rw | | rw | |
| $A4_H$ | **CCU6_PISEL2 Reset: $00_H$** Port Input Select Register 2 | Bit Field | 0 | | | | | | IST13HR | |
| | | Type | r | | | | | | rw | |
| $FA_H$ | **CCU6_T12L Reset: $00_H$** Timer T12 Counter Register Low | Bit Field | T12CVL | | | | | | | |
| | | Type | rwh | | | | | | | |
| $FB_H$ | **CCU6_T12H Reset: $00_H$** Timer T12 Counter Register High | Bit Field | T12CVH | | | | | | | |
| | | Type | rwh | | | | | | | |
| $FC_H$ | **CCU6_T13L Reset: $00_H$** Timer T13 Counter Register Low | Bit Field | T13CVL | | | | | | | |
| | | Type | rwh | | | | | | | |
| $FD_H$ | **CCU6_T13H Reset: $00_H$** Timer T13 Counter Register High | Bit Field | T13CVH | | | | | | | |
| | | Type | rwh | | | | | | | |
| $FE_H$ | **CCU6_CMPSTATL Reset: $00_H$** Compare State Register Low | Bit Field | 0 | CC63ST | CCPOS2 | CCPOS1 | CCPOS0 | CC62ST | CC61ST | CC60ST |
| | | Type | r | rh | rh | rh | rh | rh | rh | rh |
| $FF_H$ | **CCU6_CMPSTATH Reset: $00_H$** Compare State Register High | Bit Field | T13IM | COUT63PS | COUT62PS | CC62PS | COUT61PS | CC61PS | COUT60PS | CC60PS |
| | | Type | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh |

### 3.4.5.11 UART1 Registers

The UART1 SFRs can be accessed in the mapped memory area (RMAP = 1).

**Table 3-13 UART1 Register Overview**

| Addr | Register Name | Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| RMAP = 1 | | | | | | | | | | |
| $C8_H$ | **SCON** **Reset: $00_H$** Serial Channel Control Register | Bit Field | SM0 | SM1 | SM2 | REN | TB8 | RB8 | TI | RI |
| | | Type | rw | rw | rw | rw | rw | rwh | rwh | rwh |
| $C9_H$ | **SBUF** **Reset: $00_H$** Serial Data Buffer Register | Bit Field | VAL | | | | | | | |
| | | Type | rwh | | | | | | | |
| $CA_H$ | **BCON** **Reset: $00_H$** Baud Rate Control Register | Bit Field | 0 | | | | BRPRE | | | R |
| | | Type | r | | | | rw | | | rw |
| $CB_H$ | **BG** **Reset: $00_H$** Baud Rate Timer/Reload Register | Bit Field | BR_VALUE | | | | | | | |
| | | Type | rwh | | | | | | | |
| $CC_H$ | **FDCON** **Reset: $00_H$** Fractional Divider Control Register | Bit Field | 0 | | | | | NDOV | FDM | FDEN |
| | | Type | r | | | | | rwh | rw | rw |
| $CD_H$ | **FDSTEP** **Reset: $00_H$** Fractional Divider Reload Register | Bit Field | STEP | | | | | | | |
| | | Type | rw | | | | | | | |
| $CE_H$ | **FDRES** **Reset: $00_H$** Fractional Divider Result Register | Bit Field | RESULT | | | | | | | |
| | | Type | rh | | | | | | | |
| $CF_H$ | **SCON1** **Reset: $07_H$** Serial Channel Control Register 1 | Bit Field | 0 | | | | | NDOV EN | TIEN | RIEN |
| | | Type | r | | | | | rw | rw | rw |

### 3.4.5.12 SSC Registers

The SSC SFRs can be accessed in the standard memory area (RMAP = 0).

**Table 3-14 SSC Register Overview**

| Addr | Register Name | Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| RMAP = 0 | | | | | | | | | | |
| $AA_H$ | **SSC_CONL** **Reset: $00_H$** Control Register Low Programming Mode | Bit Field | LB | PO | PH | HB | BM | | | |
| | | Type | rw | rw | rw | rw | rw | | | |
| $AA_H$ | **SSC_CONL** **Reset: $00_H$** Control Register Low Operating Mode | Bit Field | 0 | | | | BC | | | |
| | | Type | r | | | | rh | | | |
| $AB_H$ | **SSC_CONH** **Reset: $00_H$** Control Register High Programming Mode | Bit Field | EN | MS | 0 | AREN | BEN | PEN | REN | TEN |
| | | Type | rw | rw | r | rw | rw | rw | rw | rw |
| $AB_H$ | **SSC_CONH** **Reset: $00_H$** Control Register High Operating Mode | Bit Field | EN | MS | 0 | BSY | BE | PE | RE | TE |
| | | Type | rw | rw | r | rh | rwh | rwh | rwh | rwh |

**Table 3-14 SSC Register Overview** (cont'd)

| Addr | Register Name | Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| $AC_H$ | **SSC_TBL** **Reset: $00_H$** Transmitter Buffer Register Low | Bit Field | TB_VALUE | | | | | | | |
| | | Type | rw | | | | | | | |
| $AD_H$ | **SSC_RBL** **Reset: $00_H$** Receiver Buffer Register Low | Bit Field | RB_VALUE | | | | | | | |
| | | Type | rh | | | | | | | |
| $AE_H$ | **SSC_BRL** **Reset: $00_H$** Baud Rate Timer Reload Register Low | Bit Field | BR_VALUE | | | | | | | |
| | | Type | rw | | | | | | | |
| $AF_H$ | **SSC_BRH** **Reset: $00_H$** Baud Rate Timer Reload Register High | Bit Field | BR_VALUE | | | | | | | |
| | | Type | rw | | | | | | | |

### 3.4.5.13 MultiCAN Registers

The MultiCAN SFRs can be accessed in the standard memory area (RMAP = 0).

**Table 3-15 CAN Register Overview**

| Addr | Register Name | Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| RMAP = 0 | | | | | | | | | | |
| **$D8_H$** | **ADCON** **Reset: $00_H$** CAN Address/Data Control Register | Bit Field | V3 | V2 | V1 | V0 | AUAD | | BSY | RWEN |
| | | Type | rw | rw | rw | rw | rw | | rh | rw |
| $D9_H$ | **ADL** **Reset: $00_H$** CAN Address Register Low | Bit Field | CA9 | CA8 | CA7 | CA6 | CA5 | CA4 | CA3 | CA2 |
| | | Type | rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh |
| $DA_H$ | **ADH** **Reset: $00_H$** CAN Address Register High | Bit Field | 0 | | | | CA13 | CA12 | CA11 | CA10 |
| | | Type | r | | | | rwh | rwh | rwh | rwh |
| $DB_H$ | **DATA0** **Reset: $00_H$** CAN Data Register 0 | Bit Field | CD | | | | | | | |
| | | Type | rwh | | | | | | | |
| $DC_H$ | **DATA1** **Reset: $00_H$** CAN Data Register 1 | Bit Field | CD | | | | | | | |
| | | Type | rwh | | | | | | | |
| $DD_H$ | **DATA2** **Reset: $00_H$** CAN Data Register 2 | Bit Field | CD | | | | | | | |
| | | Type | rwh | | | | | | | |
| $DE_H$ | **DATA3** **Reset: $00_H$** CAN Data Register 3 | Bit Field | CD | | | | | | | |
| | | Type | rwh | | | | | | | |

### 3.4.5.14 OCDS Registers

The OCDS SFRs can be accessed in the mapped memory area (RMAP = 1).

**Table 3-16 OCDS Register Overview**

| Addr | Register Name | Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| RMAP = 1 | | | | | | | | | | |
| $E9_H$ | **MMCR2** **Reset: $8U_H$** Monitor Mode Control 2 Register | Bit Field | STMODE | EXBC | DSUSP | MBCON | ALTDI | MMEP | MMODE | JENA |
| | | Type | rw | rw | rw | rwh | rw | rwh | rh | rh |
| $EA_H$ | **MEXTCR** **Reset: $0U_H$** Memory Extension Control Register | Bit Field | 0 | | | | BANKBPx | | | |
| | | Type | r | | | | rw | | | |
| $EB_H$ | **MMWR1** **Reset: $00_H$** Monitor Work Register 1 | Bit Field | MMWR1 | | | | | | | |
| | | Type | rw | | | | | | | |
| $EC_H$ | **MMWR2** **Reset: $00_H$** Monitor Work Register 2 | Bit Field | MMWR2 | | | | | | | |
| | | Type | rw | | | | | | | |
| $F1_H$ | **MMCR** **Reset: $00_H$** Monitor Mode Control Register | Bit Field | MEXIT_P | MEXIT | 0 | MSTEP | MRAMS_P | MRAMS | TRF | RRF |
| | | Type | w | rwh | r | rw | w | rwh | rh | rh |
| $F2_H$ | **MMSR** **Reset: $00_H$** Monitor Mode Status Register | Bit Field | MBCAM | MBCIN | EXBF | SWBF | HWB3F | HWB2F | HWB1F | HWB0F |
| | | Type | rw | rwh | rwh | rwh | rwh | rwh | rwh | rwh |
| $F3_H$ | **MMBPCR** **Reset: $00_H$** Breakpoints Control Register | Bit Field | SWBC | HWB3C | | HWB2C | | HWB1C | HWB0C | |
| | | Type | rw | rw | | rw | | rw | rw | |
| $F4_H$ | **MMICR** **Reset: $00_H$** Monitor Mode Interrupt Control Register | Bit Field | DVECT | DRETR | COMRST | MSTSEL | MMUIE_P | MMUIE | RRIE_P | RRIE |
| | | Type | rwh | rwh | rwh | rh | w | rw | w | rw |
| $F5_H$ | **MMDR** **Reset: $00_H$** Monitor Mode Data Transfer Register Receive | Bit Field | MMRR | | | | | | | |
| | | Type | rh | | | | | | | |
| $F6_H$ | **HWBPSR** **Reset: $00_H$** Hardware Breakpoints Select Register | Bit Field | 0 | | | BPSEL_P | BPSEL | | | |
| | | Type | r | | | w | rw | | | |
| $F7_H$ | **HWBPDR** **Reset: $00_H$** Hardware Breakpoints Data Register | Bit Field | HWBPxx | | | | | | | |
| | | Type | rw | | | | | | | |

### 3.4.5.15 Flash Registers

The Flash SFRs can be accessed in the mapped memory area (RMAP = 1).

**Table 3-17 Flash Register Overview**

| Addr | Register Name | Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| RMAP = 1 | | | | | | | | | | |
| $D1_H$ | **FCON** **Reset: $10_H$**<br>P-Flash Control Register | Bit Field | 0 | FBSY | YE | 1 | NVSTR | MAS1 | ERASE | PROG |
| | | Type | r | rh | rwh | r | rw | rw | rw | rw |
| $D2_H$ | **EECON** **Reset: $10_H$**<br>D-Flash Control Register | Bit Field | 0 | EEBSY | YE | 1 | NVSTR | MAS1 | ERASE | PROG |
| | | Type | r | rh | rwh | r | rw | rw | rw | rw |
| $D3_H$ | **FCS** **Reset: $80_H$**<br>Flash Control and Status Register | Bit Field | 1 | SBEIE | FTEN | 0 | EEDERR | EESERR | FDERR | FSERR |
| | | Type | r | rw | rwh | r | rwh | rwh | rwh | rwh |
| $D4_H$ | **FEAL** **Reset: $00_H$**<br>Flash Error Address Register, Low Byte | Bit Field | ECCEADDR | | | | | | | |
| | | Type | rh | | | | | | | |
| $D5_H$ | **FEAH** **Reset: $00_H$**<br>Flash Error Address Register, High Byte | Bit Field | ECCEADDR | | | | | | | |
| | | Type | rh | | | | | | | |
| $D6_H$ | **FTVAL** **Reset: $78_H$**<br>Flash Timer Value Register | Bit Field | MODE | OFVAL | | | | | | |
| | | Type | rw | rw | | | | | | |
| $DD_H$ | **FCS1** **Reset: $00_H$**<br>Flash Control and Status Register 1 | Bit Field | 0 | | | | | | | EEABORT |
| | | Type | r | | | | | | | rwh |

## 3.5 Boot ROM Operating Mode

After a reset, the CPU will always start by executing the Boot ROM code in active memory map 0. In active memory map 0, the Boot ROM occupies the program memory address space $0'0000_H$ – $0'1FFF_H$, $0'C000_H$ – $0'DFFF_H$ and $2'C000_H$ – $2'DFFF_H$with the remaining program memory address space disabled. The Boot ROM start-up procedure will first jump to $2'C00X_H$ before switching to active memory map 1 as shown in **Figure 3-6** and **Figure 3-7**. As a result, the Boot ROM memory formerly occupying the address range $0'0000_H$ – $0'1FFF_H$ , $0'C000_H$ – $0'DFFF_H$ and $2'C000_H$ - $2'DFFF_H$ will be mapped to only $2'C000_H$ – $2'DFFF_H$ or $0'C000_H$ - $0'DFFF_H$ depending on the Flash memory size. Also, the remaining program memory blocks (XRAM, P-Flash and D-Flash) are enabled. After the active memory map switch, the remaining Boot ROM start-up procedure will be executed from $2'C00X_H$/$0'C00X_H$. This includes checking the latched values of pins MBC, TMS and P0.0 to enter the selected Boot ROM operating modes. Refer to **Chapter 7.2.3** for the selection of different Boot ROM operating modes. The memory organization of the XC878 shown in this document is after the active memory map switch, i.e. active memory map 1, where the different operating modes are executed.

*Note: The TM pin requires a external pull down device to ensure a proper entry of the Boot ROM operating mode*

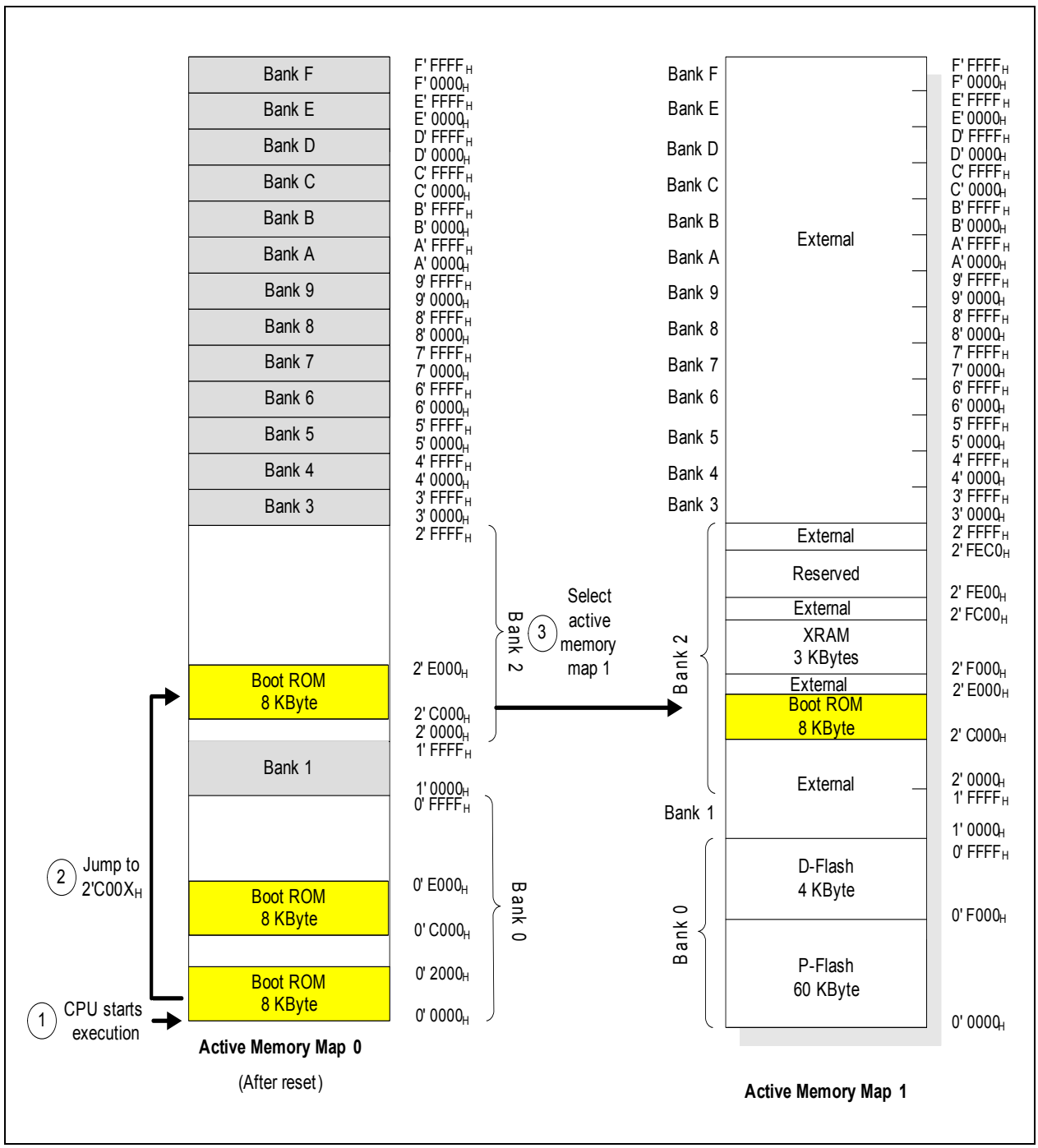



**Figure 3-6 Active Memory Map Selection for 64K Flash**

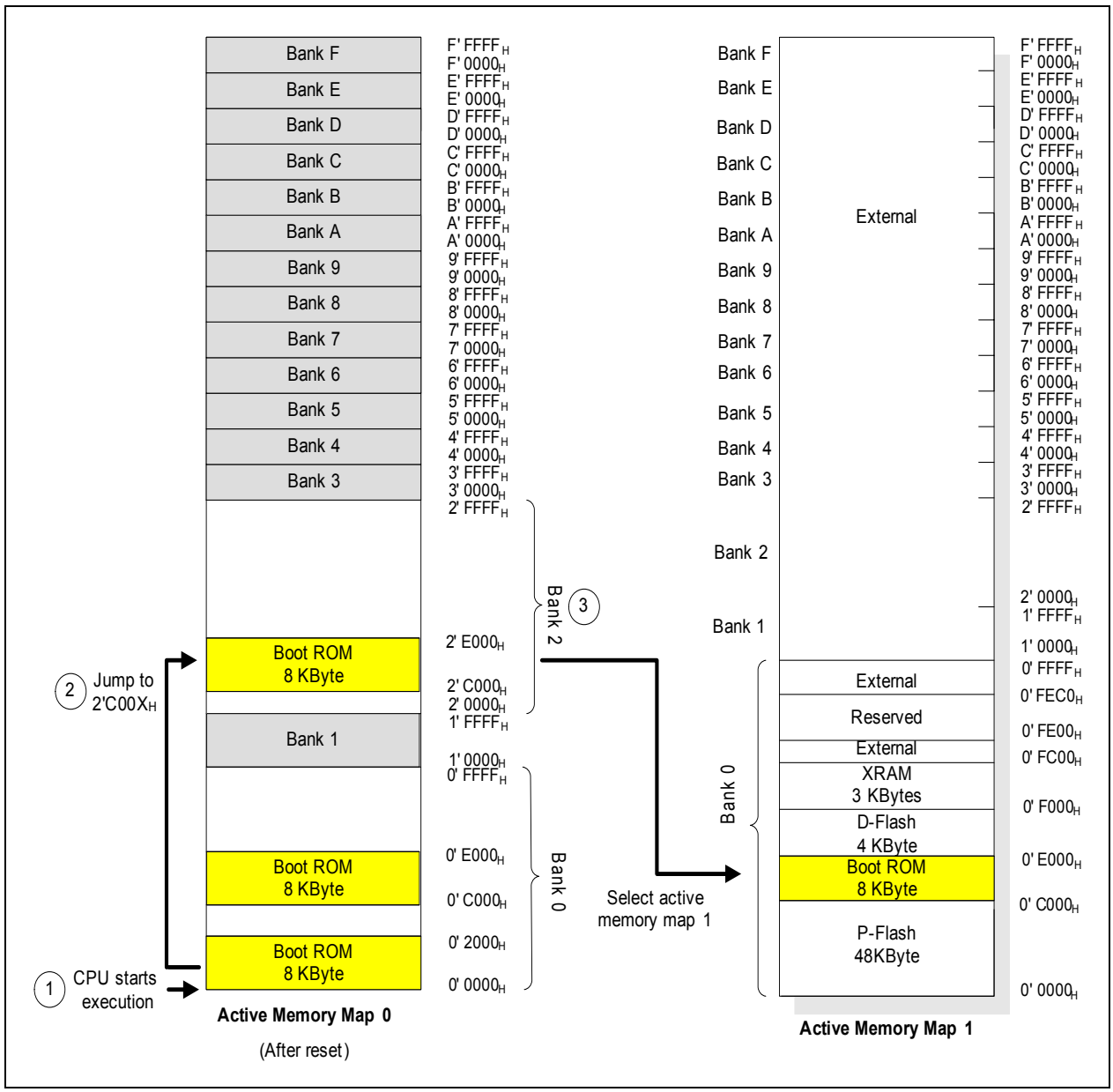



**Figure 3-7 Active Memory Map Selection for 52K Flash**

## 3.5.1 User Mode

If (MBC, TMS, P0.0) = (1, 0, x), the Boot ROM will jump to program memory address $0'0000_H$ to execute the user code in the Flash. This is the normal operating mode of the XC878 with the On-Chip Oscillator and PLL in non-bypassed mode.

However, if program memory address $0'0000_H$ contains $FF_H$, indicating the Flash memory is not yet programmed with user code, BootStrap Loader (BSL) mode will be entered instead to facilitate Flash programming.

*Note: User should prevent programming a $FF_H$ value to program memory address $0000_H$ to avoid entering BSL mode unintentionally.*

### 3.5.2 Bootstrap Loader Mode

If (MBC, TMS, P0.0) = (0, 0, x), the software routines of the BootStrap Loader (BSL) located in the Boot ROM will be executed, allowing the XRAM and Flash memory (if available) to be programmed, erased and executed. Refer to **Chapter 19** for the different BSL working modes.

### 3.5.3 OCDS Mode

If (MBC, TMS, P0.0) = (0, 1, 0), the OCDS mode will be entered for debugging program code. The OCDS hardware is initialized and a jump to program memory address $0000_H$ is performed next. The user code in the Flash is executed and the debugging process may be started.

During the OCDS mode, the lowest 64 bytes ($00_H$ – $3F_H$) in the internal data memory address range may be alternatively mapped to the 64-byte monitor RAM or the internal data RAM.

### 3.5.4 User JTAG Mode

If (MBC, TMS, P0.0) = (1, 1, x), the Boot ROM will jump to program memory address $0000_H$ to execute the user code in the Flash. This is similar to the normal user mode described in **Section 3.5.1**, with the addition that the primary JTAG port is automatically configured to allow hot-attach.

# 4 Flash Memory

The XC878 has an embedded user-programmable non-volatile Flash memory that allows for fast and reliable storage of user code and data. It is operated with a single 2.5 V supply from the Embedded Voltage Regulator (EVR) and does not require additional programming or erasing voltage.

**Features**

- In-System Programming (ISP) via UART
- In-Application Programming (IAP)
- Error Correction Code (ECC) for dynamic correction of single-bit errors
- Support for aborting erase operation
- Minimum program width of 1-byte for D-Flash and 2-byte for P-Flash
- 1-page minimum erase width
- 1-byte read access
- 1 × CCLK period read access time
- Flash is delivered in erased state (read all ones)

## 4.1 Flash Memory Map

The XC878 product family offers Flash devices with 64 Kbytes and 52 Kbytes of embedded Flash memory. Each Flash device consists of Program Flash (P-Flash) and Data Flash (D-Flash) block. The 64-Kbyte Flash device consists of 60 Kbytes of P-Flash and 4 Kbytes of D-Flash; 52-Kbyte Flash device consists of 48 Kbytes of P-Flash and 4 Kbytes of D-Flash. **Table 4-1** shows the address mappings for P-Flash and D-Flash of the 2 devices available in XC878.

**Table 4-1 Flash Memory Map for the 3 Flash devices**

| Devices | Description | Size | Address Range |
|---|---|---|---|
| 64-Kbyte Flash | P-Flash | 60 KBytes | $0'0000_H$ - $0'EFFF_H$ |
| | D-Flash | 4 KBytes | $0'F000_H$ - $0'FFFF_H$ |
| 52-Kbyte Flash | P-Flash | 48 KBytes | $0'0000_H$ - $0'BFFF_H$ |
| | D-Flash | 4 KBytes | $0'E000_H$ - $0'EFFF_H$ |

The P-Flash memory occupies program memory address starting from $0000_H$, where the reset and interrupt vectors are located. The 4 Kbytes D-Flash memory are mapped to different location in the program memory address spaces based on the type of Flash devices in the XC878.

## 4.2 Flash Block Pagination

The XC878 devices consist of two types of blocks, namely Program Flash (P-Flash) block and Data Flash (D-Flash) block. P-Flash has 120 pages of 8 wordlines per page with 64 bytes per wordline. D-Flash has 64 pages of 2 wordlines per page with 32 bytes per wordline. Both types can be used for code and data storage. The label “Data” neither implies that the D-Flash is mapped to the data memory region, nor that it can only be used for data storage. It is used to distinguish the different page width and wordline of each Flash block.

The internal structure of each Flash block represents a page architecture for flexible erase capability. The minimum erase width is always a complete page. The D-Flash block is divided into smaller size for extended erasing and reprogramming capability; even numbers for each page size are provided to allow greater flexibility and the ability to adapt to a wide range of application requirements.

For example, the user’s program can implement a buffer mechanism for each page. Double copies of each data set can be stored in separate pages of similar size to ensure that a backup copy of the data set is available in the event the actual data set is corrupted or erased.

Alternatively, the user can implement an algorithm for EEPROM emulation, which uses the D-Flash block like a circular stack memory; the latest data updates are always

programmed on top of the actual region. When the top of the page is reached, all actual data (representing the EEPROM data) is copied to the bottom area of the next page and the last page is then erased. This round robin procedure, using multifold replications of the emulated EEPROM size, significantly increases the Flash endurance. To speed up data search, the RAM can be used to contain the pointer to the valid data set.

## 4.3 Operating Modes

XC878 has five Flash operation modes: STANDBY, READ, PROGRAM, ERASE and MASS ERASE, that are applicable to both P-Flash and D-Flash. Read operation is controlled by hardware while program, erase and mass erase operations are all controlled by CPU via software routines. In general, the Flash operating modes are controlled by the BSL and Flash program/erase subroutines (see **Section 4.7**). But user also have the option to create software subroutines to perform these flash operations (see **Section 4.3.1** - **Section 4.3.5**). Various timing intervals are required during flash operations and they can be generated using the Flash Timer as describe in **Section 4.4**.

During an erase or program cycle, read operations on the same Flash array block are not allowed. If the user interrupts a program or erase cycle, the data may not be programmed or erased correctly. The software procedure as described in **Section 4.3.5** can be used to abort D-Flash program and erase operation to ensure correct behaviour of Flash macro.

When the user sets bit PMCON0.PD = 1 to enter the system power-down mode, the Flash blocks are automatically brought to its Standby state by hardware before bringing the system to power down state.

### 4.3.1 Read Operation

The read operation is controlled mainly by hardware. From the bus point of view, a read from the Flash module is done like a fetch from a program memory. Once the read signal from the CPU is active, the Flash memory block will enter the read mode and the data will be available to the CPU at the next clock cycle. No wait state is needed.

### 4.3.2 Program Operation

A program cycle writes data to a Flash array block. A write is performed by the instruction ‘MOVC @(DPTR++),A’. The data is stored in a data buffer before it is encoded by the ECC module and programmed into the Flash array block. In XC878, the program width for P-Flash memory is in multiples of 2 bytes and D-Flash memory is 1 byte.

The following steps shows a possible programming sequence in a program cycle:

1. Set the bit FCON.PROG (P-Flash) or EECON.PROG (D-Flash) to signal the start of a programming cyle.

2. Execute a “MOVC” instruction with a dummy data to any address in the same wordline as the address to be accessed.
3. Delay for a minimum of 5 us ($T_{vns}$).
4. Set the bit FCON/EECON.NVSTR for charge pump to drive high voltage.
5. Delay for a minimum of 10 us ($T_{pgs}$).
6. Execute a “MOVC” instruction to the flash address to be accessed. FCON/EECON.YE and FCS.FTEN is set by hardware at next clock cycle (YE hold time of 40 ns is needed).
7. Delay for a minimum of 20 us but not longer than 40 us ($T_{prog}$).
8. Clear the bits FCON/EECON.YE and FCS.FTEN respectively.
9. Repeat steps 6 to 8 for any further programming of data to the same row.
10. Clear the bit FCON/EECON.PROG.
11. Delay for a minimum of 5 us ($T_{nvh}$)
12. Clear the bit FCON/EECON NVSTR.
13. Delay for a minimum of 1 us ($T_{rcv}$).

*Note: The programming sequences from 1 to 12 have to be completed within 4 ms.*

### 4.3.3 Erase Operation

An erase cycle erases a Flash page to “1”.

A possible page erase sequence is as follows:

1. Set the bit FCON/EECON.ERASE and clear the bit FCON/EECON.MAS1 to trigger the start of the page erase cycle.
2. Execute a “MOVC” instruction with a dummy data to any address in the page to be erased.
3. Delay for a minimum of 5 µs ($T_{vns}$).
4. Set the bit FCON/EECON.NVSTR for charge pump to drive high voltage.
5. Delay for a minimum of 20 ms ($T_{erase}$).
6. Clear bit FCON/EECON.ERASE.
7. Delay for a minimum of 5 us ($T_{nvh}$)
8. Clear the bit FCON/EECON NVSTR.
9. Delay for a minimum of 1 us ($T_{rcv}$).

*Note: If a read is carried out on an address belonging to the page in the P-Flash which has just been erased, FDERR bit in register FCS will be set and an interrupt may be generated due to an ECC error. To prevent this, ECC interrupt generation should be disabled via the bit NMICON.NMIECC.*

### 4.3.4 Mass Erase Operation

All memory cells in either the P-Flash block or the D-Flash block will be erased to “1” in mass erase operation depending on the address specific to be erased. If a read is carried

out on the P-Flash block following a mass erase, an interrupt may be generated due to an ECC error. To prevent this, ECC interrupt generation should be disabled via the bit NMICON.NMIECC.

A possible mass erase sequence is as follows:

1. Set the bits FCON/EECON.ERASE and FCON/EECON.MAS1 to trigger the start of the mass erase cycle.
2. Execute a “MOVC” instruction with a dummy data to any address in the page to be erased.
3. Delay for a minimum of 5 µs ($T_{vns}$).
4. Set the bit FCON/EECON.NVSTR for charge pump to drive high voltage.
5. Delay for a minimum of 200 ms ($T_{me}$).
6. Clear bit FCON/EECON.ERASE.
7. Delay for a minimum of 100 us ($T_{nvhl}$)
8. Clear the bit FCON/EECON NVSTR and FCON/EECON.MAS1.
9. Delay for a minimum of 1 us ($T_{rcv}$).

### 4.3.5 Abort Operation

Program and erase operation including mass erase for D-Flash memory are allowed to be aborted by software. To ensure the correct behavior of Flash memory, software must follow the sequence shown in **Figure 4-1**. The abort operation can only be used while accessing D-Flash memory. Read, program and Erase operations are blocked once bit EEABORT in register FCS1 is set.

To ensure the timing interval required in the abort sequence is generated correctly using Flash timer, software need to clear FTEN and then set FTEN bit again. This step will reset the two counters in the Flash timer to zero count.

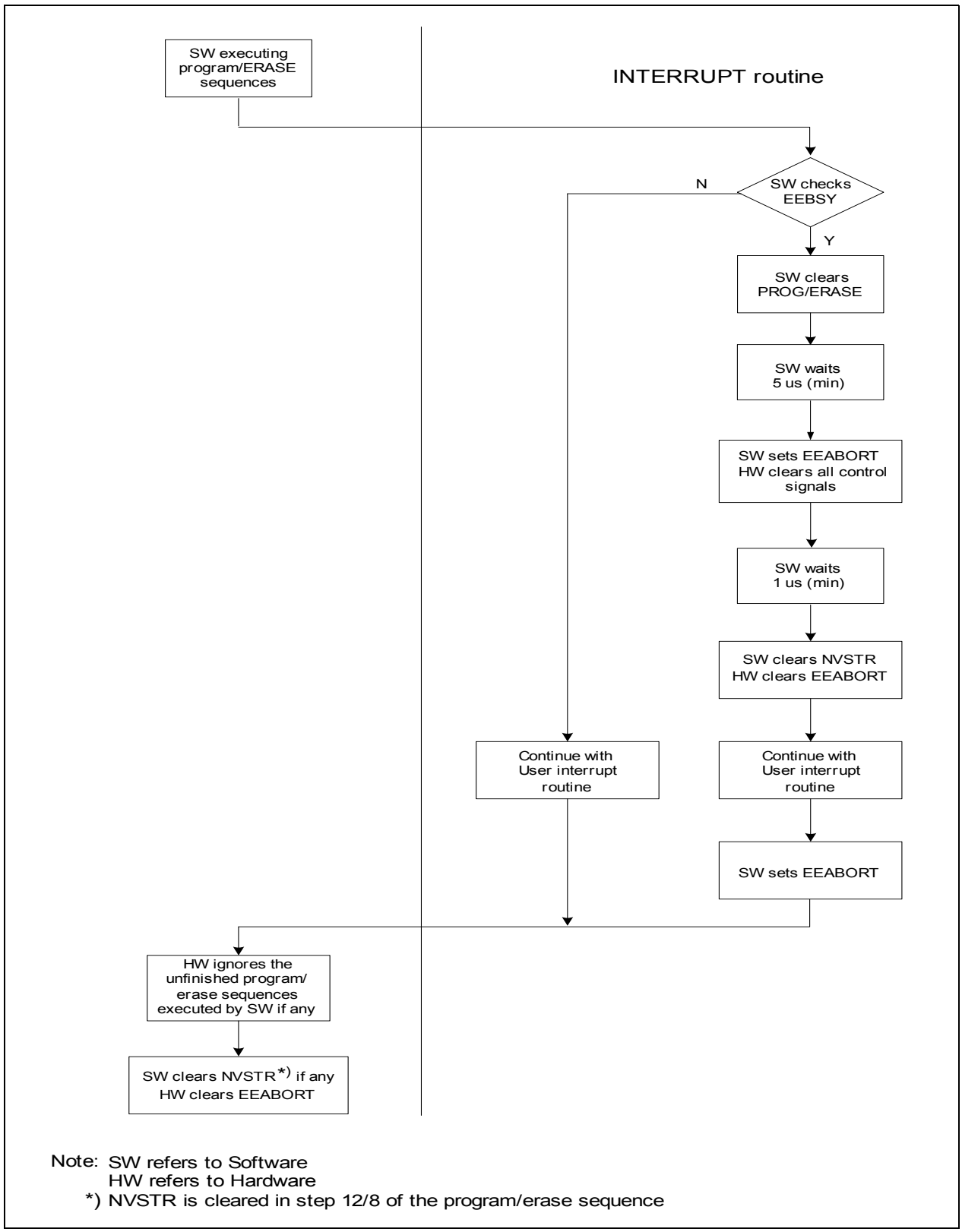



**Figure 4-1 D-Flash Program/Erase Abort Flow**

## 4.4 Flash Timer

The Flash Timer is used to generate various time intervals (e.g. for program set-up time, erase time) required during program and erase operations.

The Flash Timer has 2 operating modes, Program Timer and Erase Timer modes, which are selectable using the bit FTVAL.MODE. In Program Timer mode (FTVAL.MODE=0), a 7-bit counter is used to count up to the designated overflow value stored in FTVAL.OFVAL. Once this value is reached, the associated overflow flag FCS.FTOV is set. The 7-bit counter is then reset to zero and if the Flash timer is not disabled, resumes counting.

In Erase Timer mode (FTVAL.MODE=1), an additional 12-bit counter is used. On enabling the Flash Timer, the 12-bit counter starts counting up. Each time the 12-bit counter overflows, it generates a signal to increment the 7-bit counter by 1, before it is reset to zero and resumes counting. The timer overflow signal and flag is set only when the 7-bit counter reaches the designated overflow value.

The Flash Timer is enabled by setting the enable bit, FCS.FTEN, to ‘1’. Similarly, it is disabled by clearing the enable bit. Once disabled, both the 7-bit and 12-bit counter are reset and remain at zero.

An example of the Flash Timer waveform is shown in **Figure 4-2**.

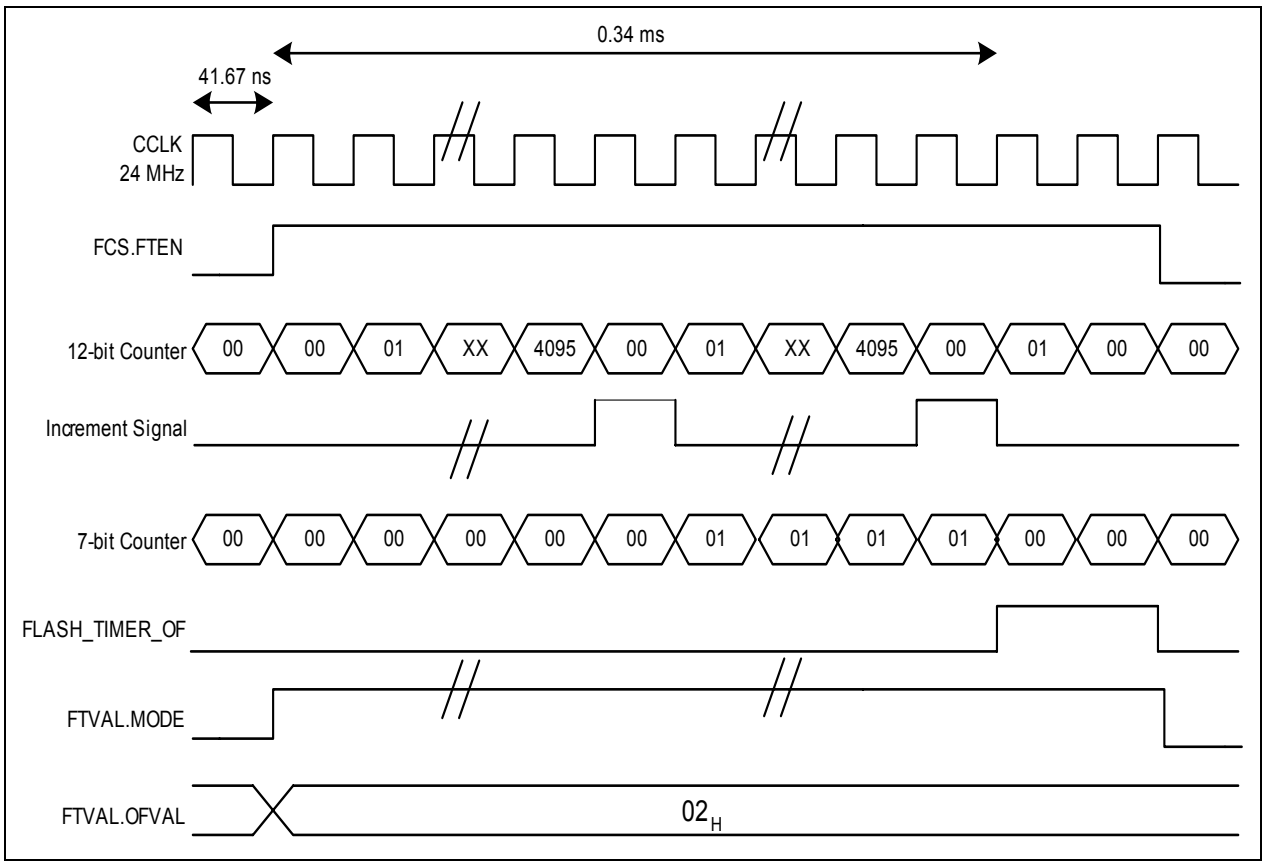



**Figure 4-2 Flash Timer Timing Diagram**

## 4.5 Error Detection and Correction

The 8-bit data from the CPU is encoded with an Error Correction Code (ECC) before being stored in the Flash memory. The ECC is generated based on a modified Hamming code and concatenated to the MSB of the data. During a read access, data is retrieved from the Flash memory and decoded for dynamic error detection and correction.

The correction algorithm (hamming code) has the capability to:

- Detect and correct all 1-bit errors
- Detect all 2-bit errors, but cannot correct

A corrected 1-bit error (result is valid) and an uncorrected 2-bit error (result is invalid) can be distinguished by the single bit error flag and double bit error flag in register FCS.

In both cases, an ECC non-maskable interrupt (NMI) event can be generated; bit FNMIECC in register NMISR is set, and if enabled via NMICON.NMIECC, an NMI to the CPU is triggered. For the case of single bit error interrupt, bit SBEIE in register FCS has to be enabled too.

The 16-bit Flash address at which the ECC error occurs is stored in the system control SFRs FEAL and FEAH as described in **Section 4.9**, and can be accessed by the interrupt service routine to determine where the error occurred.

## 4.6 In-System Programming

In-System Programming (ISP) of the Flash memory is supported via the Boot ROM-based Bootstrap Loader (BSL), allowing a blank microcontroller device mounted onto an application board to be programmed with the user code, and also a previously programmed device to be erased then reprogrammed without removal from the board. This feature offers ease-of-use and versatility for the embedded design.

ISP is supported through the microcontroller's serial interface (UART) which is connected to the personal computer host via the commonly available RS-232 serial cable. The BSL mode is selected if the latched values of the MBC and TMS pins are 0 after power-on or hardware reset. The BSL routine will first perform an automatic synchronization with the transfer speed (baud rate) of the serial communication partner (personal computer host). Communication between the BSL routine and the host is done via a transfer protocol; information is sent from the host to the microcontroller in blocks with specified block structure, and the BSL routine acknowledges the received data by returning a single acknowledge or error byte. User can program, erase or execute the P-Flash and D-Flash blocks.

The available working modes include:

- Transfer user program from host to Flash
- Execute user program in Flash
- Page Erase of P-Flash or D-Flash block from the same or different page(s).
- Mass Erase of P-Flash block and D-Flash block

## 4.7 In-Application Programming

In some applications, the Flash contents may need to be modified during program execution. In-Application Programming (IAP) is supported so that users can program, erase or mass erase the Flash memory from their Flash user program by calling some subroutines in the Boot ROM. The Flash subroutines will first perform some checks and an initialization sequence before starting the program, erase or mass erase operation. User program can only continue to run after the completion of these flash operation. A manual check on the Flash data is necessary to determine if the programming or erasing was successful via using the 'MOVC' instruction to read out the Flash contents.

*Note: The wordline in the Flash memory, where the Flash user program is executing from, cannot be targeted for any page/mass erase and program operation. For example, user program in P Flash Block Page 1 Wordline 0 can program or erase other wordline except this Wordline 0 from Page 1.*

### 4.7.1 Flash Programming

Each call of the Flash program subroutine allows the programming of 2 to 64 bytes of data for P-Flash block and 1 to 32 bytes of data for D-Flash block into the selected wordline (WL).

Before calling this subroutine, the user must ensure that the contents to be programmed are stored incrementally in the XRAM, starting from the address specified in DPTR1. R1 is also needed to indicate the number of bytes to progam into a WL.. In addition, the input DPTR0 must contain a valid Flash address in even number (addresses of a protected Flash block is considered invalid). Otherwise, PSW.CY bit will be set and no programming will occur. During each programming cycle, no crossing of a wordline boundary is allowed. For example, programming of 4-bytes of data to P-Flash block starting at address $003E_H$ is not allowed.

If valid inputs (see **Table 4-2**) are available before calling the subroutine, the microcontroller will start with the initialization sequence (includes setting up the Flash Timer in an IRAM buffer) and then execute the program sequence as describe in **Section 4.3.2**. User program code will only continue after the execution of the Flash program subroutine.

**Table 4-2 Flash Program Subroutine**

| | |
|---|---|
| **Subroutine** | $DFF6_H$: FLASH_PROGRAM |
| **Input** | DPTR0:<br>Start address of the Flash programming location |
| | DPTR1:<br>XRAM start address containing Flash data to be programmed. |
| | R1:<br>Number of bytes to be programmed |
| | EA:0 |
| **Output** | PSW.CY:<br>C = 0<br>Programs the content in R1 number of bytes of the XRAM block to the appropriate P-Flash or D-Flash[1)] location(s).<br>C = 1<br>Flash is not programmed due to one or more of the following:<br>1. DPTR0 has an invalid P-Flash or D-Flash address.<br>2. DPTR0 has an odd P-Flash address.<br>3. R1 is 0 or exceed the appropriate Flash WL size.<br>4. R1 has an odd number during P-Flash programming.<br>5. The Flash block to be programmed crosses a wordline boundary. |
| **Stack size required** | 9 |
| **Resource used/destroyed** | ACC, B, C, R0, R3, R4, R5, IRAM($37_H$ - $3E_H$) |

1) The P-Flash or D-Flash location is automatically determined by the subroutine whenever it is invoked.

### 4.7.2 Flash Page Erasing and Mass Erasing

Each call of the Flash erase subroutine for erase operation allows either Page Erase(erase a single page in either P-Flash block or the D-Flash) or Mass Erase(erase either the P-Flash block or the D-Flash block) to be carried out. Before calling the subroutine for erase operation, the ACC register must contain the type of erase operation and the input DPTR0 must contain a valid Flash address within the page or block to be erased. Also, protected Flash blocks should not be targeted for erase.

If valid inputs (see **Table 4-3**) are available before calling the subroutine, tthe microcontroller will start with the initialization sequence and then execute the erase sequence as described in **Section 4.3.3**. User program code will only continue after the execution of the Flash erase subroutine.

**Table 4-3 Flash Erase Subroutine**

| | |
|---|---|
| **Subroutine** | $DFF9_H$: FLASH_ERASE |
| **Input** | DPTR0:<br>Any address in the Flash page or block to be erased. |
| | ACC:<br>Type of Erase (0: Page Erase, 1: Mass Erase) |
| | EA:0 |
| | MISC_CON.DFLASHEN[1]:1 |
| **Output** | PSW.CY:<br>C = 0<br>Erase the appropriate P-Flash or D-Flash[2] page or block.<br>C = 1<br>Flash is not erased due to an invalid Flash address. |
| **Stack size required** | 9 |
| **Resource used/destroyed** | ACC, B, C, R0, R3, R4, R5, IRAM($37_H$ - $3C_H$) |

1) When Flash Protection Mode 0 is enabled, the DFLASHEN bit needs to be set before each erase of the D-Flash block.

2) The P-Flash or D-Flash location is automatically determined by the subroutine whenever it is invoked.

### 4.7.3 Get Chip Information

This subroutine reads out a 4-byte data that contains chip related information. In the XC878, it reads out the 4-byte chip identification number, which is used to identify the particular device variant.

**Table 4-4 Get Chip Information Subroutine**

| | |
|---|---|
| **Subroutine** | $DFE1_H$: GET_CHIP_INFO |
| **Input** | ACC:<br>$00_H$ = Chip Identification Number<br>Others = Reserved |
| | R1:<br>IRAM start address for 4-byte return data |
| | MEX2.MCM:0 |

**Table 4-4 Get Chip Information Subroutine** (cont'd)

| | |
|---|---|
| **Output** | PSW.CY:<br>C = 0<br>With a valid input (ACC = $00_H$), IRAM buffer filled with the following:<br>Byte 1 in R1 (MSB)<br>Byte 2 in R1 + 1<br>Byte 3 in R1 + 2<br>Byte 4 in R1 + 3 (LSB)<br>C = 1<br>IRAM buffer is cleared upon invalid input (ACC not equal to $00_H$) |
| **Stack size required** | 4 |
| **Resource used/destroyed** | ACC, R1, R6 |

## 4.8 Register Map

The SFRs of Flash module consist of two control registers for P-Flash and D-Flash, a common status register, a control register for D-Flash, a Flash Timer register and two address registers to store the full 16-bit address in the event of an ECC error. They can be accessed from the mapped SFR area.

Table 4-5 list the Flash module registers with their addresses.

**Table 4-5 SFR Address List**

| Address | Register |
|---|---|
| $D1_H$ | FCON |
| $D2_H$ | EECON |
| $D3_H$ | FCS |
| $D4_H$ | FEAL |
| $D5_H$ | FEAL |
| $D6_H$ | FTVAL |
| $DD_H$ | FCS1 |

## 4.9 Register Description

**FCON**
**P-Flash Control Register** **Reset Value: $10_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **0** | **FBSY** | **YE** | **1** | **NVSTR** | **MAS1** | **ERASE** | **PROG** |
| r | rh | rwh | r | rwh | rwh | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **PROG** | 0 | rw | **Program Bit**<br>0 Program operation is not selected.<br>1 Program operation is selected. |
| **ERASE** | 1 | rw | **Erase Bit**<br>0 Erase operation is not selected.<br>1 Erase operation is selected. |
| **MAS1** | 2 | rwh | **Mass Erase Bit**<br>0 Mass Erase operation is not selected.<br>1 Mass Erase operation is selected. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **NVSTR** | 3 | rwh | **Non-Volatile Store Bit**<br>0 Charge pump to drive high voltage is off.<br>1 Charge pump to drive high voltage is on. |
| **YE** | 5 | rwh | **Y-Address Enable Bit**<br>The YE bit is set by hardware only but can be cleared by both hardware and software.<br>0 FYE is disabled.<br>1 FYE is enabled. |
| **FBSY** | 6 | rh | **P-Flash Busy Bit**<br>0 Flash is not performing any Program/Erase operation.<br>1 Flash is still performing a Program/Erase operation. |
| **1** | 4 | r | **Reserved**<br>Returns 1 if read; should be written with 1. |
| **0** | 7 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

**EECON**
**D-Flash Control Register** **Reset Value: $10_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **0** | **EEBSY** | **YE** | **1** | **NVSTR** | **MAS1** | **ERASE** | **PROG** |
| r | rh | rwh | r | rwh | rwh | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **PROG** | 0 | rw | **Program Bit**<br>0 Program operation is not selected.<br>1 Program operation is selected. |
| **ERASE** | 1 | rw | **Erase Bit**<br>0 Erase operation is not selected.<br>1 Erase operation is selected. |
| **MAS1** | 2 | rwh | **Mass Erase Bit**<br>0 Mass Erase operation is not selected.<br>1 Mass Erase operation is selected. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **NVSTR** | 3 | rwh | **Non-Volatile Store Bit**<br>0 Charge pump to drive high voltage is off.<br>1 Charge pump to drive high voltage is on. |
| **YE** | 5 | rwh | **Y-Address Enable Bit**<br>The YE bit is set by hardware only but can be cleared by both hardware and software.<br>0 FYE is disabled.<br>1 FYE is enabled. |
| **EEBSY** | 6 | rh | **D-Flash Busy Bit**<br>0 Flash is not performing any Program/Erase operation.<br>1 Flash is still performing a Program/Erase operation. |
| **1** | 4 | r | **Reserved**<br>Returns 1 if read; should be written with 1. |
| **0** | 7 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

**FCS**
**Flash Control and Status Register** **Reset Value: $80_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **1** | **SBEIE** | **FTEN** | **0** | **EEDERR** | **EESERR** | **FDERR** | **FSERR** |
| r | rw | rwh | r | rwh | rwh | rwh | rwh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **FSERR** | 0 | rwh | **Flash Single-bit Error Flag**<br>The FSERR bit is set by hardware and reset by software.<br>0 No single-bit error in data has occurred.<br>1 A single-bit error in data has occurred. |
| **FDERR** | 1 | rwh | **Flash Double-bit Error Flag**<br>The FDERR bit is set by hardware and reset by software.<br>0 No double-bit error has occurred.<br>1 A double-bit error has occurred. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **EESERR** | 2 | rwh | **D-Flash Single-bit Error Flag**<br>The EESERR bit is set by hardware and reset by software.<br>0 No single-bit error in data has occurred.<br>1 A single-bit error in data has occurred. |
| **EEDERR** | 3 | rwh | **D-Flash Double-bit Error Flag**<br>The EEDERR bit is set by hardware and reset by software.<br>0 No double-bit error has occurred.<br>1 A double-bit error has occurred. |
| **FTEN** | 5 | rwh | **Flash Timer Enable Bit**<br>The FTEN bit can be set by both hardware and software but can only be reset by software.<br>0 Flash Timer is disabled.<br>1 Flash Timer is enabled. |
| **SBEIE** | 6 | rw | **Single-Bit Error Interrupt Enable**<br>0 Single-bit error interrupt is not enabled.<br>1 Single-bit error interrupt is enabled. |
| **0** | 4 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |
| **1** | 7 | r | **Reserved**<br>Returns 1 if read; should be written with 1. |

**FTVAL**
**Flash Timer Value Register** **Reset Value: $78_H$**

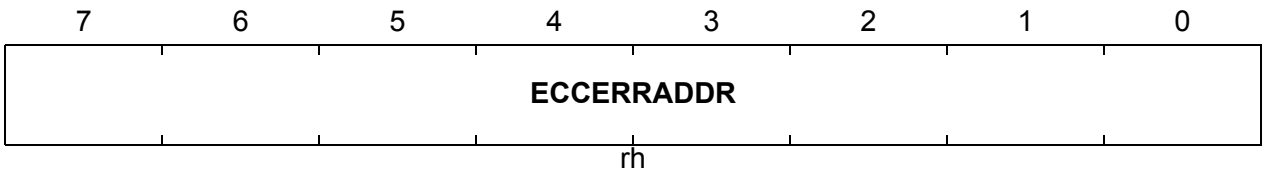
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **MODE** | **OFVAL** | | | | | | |
| rw | rw | | | | | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **OFVAL** | [6:0] | rw | **Overflow Value**<br>The bit field OFVAL stores the value at which the 7-bit counter in the Flash Timer generates an overflow. Default value is $78_H$ , giving rise to a time interval of slightly more than 5 µs at 24 MHz. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **MODE** | 7 | rw | **MODE**<br>0 Program Timer mode is selected.<br>1 Erase Timer mode is selected. |

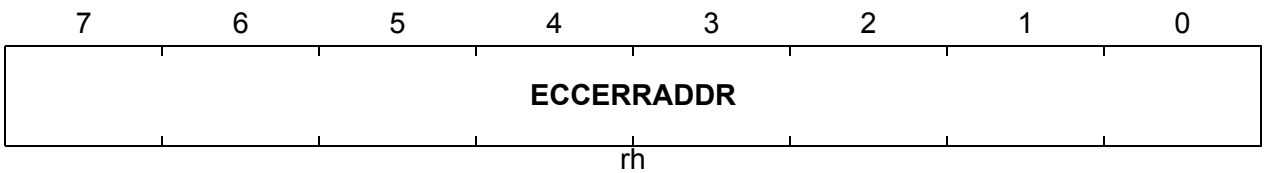
*Note: The default value of $78_H$ for bit field OFVAL is calculated based on the default core frequency of 24 MHz. For frequencies other than 24 MHz, a new overflow value has to be calculated and written to OFVAL.*

**FEAL**
**Flash Error Address Register Low** **Reset Value: $00_H$**



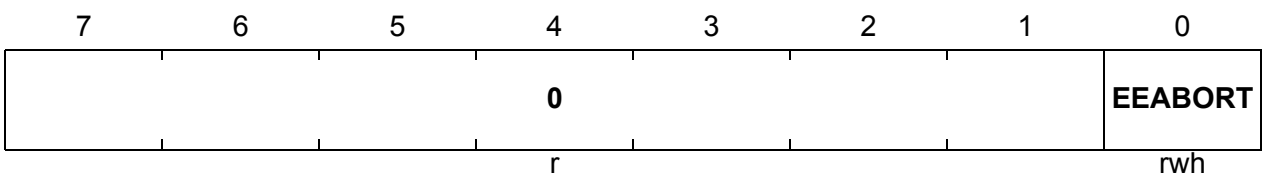

| Field | Bits | Type | Description |
|---|---|---|---|
| **ECCERRADDR** | [7:0] | rh | **ECC Error Address Value [7:0]** |

**FEAH**
**Flash Error Address Register High** **Reset Value: $00_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **ECCERRADDR** | [7:0] | rh | **ECC Error Address Value [15:8]** |

**FCS1**
**Flash Control and Status Register 1** **Reset Value: $00_H$**



| Field | Bits | Type | Description |
|---|---|---|---|
| **EEABORT** | 0 | rwf | **D-Flash Program/Erase Abort**<br>The EEABORT bit is set by software and reset by clearing EECON.NVSTR bit.<br>0 Program/Erase access is not aborted.<br>1 Program/Erase access is aborted.<br>*Note: Read, program and erase operations are blocked once EEABORT bit is set.* |
| **0** | 4 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

# 5 Interrupt System

The XC800 Core supports one non-maskable interrupt (NMI) and 14 maskable interrupt requests. In addition to the standard interrupt functions supported by the core, e.g., configurable interrupt priority and interrupt masking, the XC878 interrupt system provides extended interrupt support capabilities such as the mapping of each interrupt vector to several interrupt sources to increase the number of interrupt sources supported, and additional status registers for detecting and identifying the interrupt source.

The XC878 supports 14 interrupt vectors with four priority levels. Twelve of these interrupt vectors are assigned to the on-chip peripherals: Timer 0, Timer 1 and UART are each assigned one dedicated interrupt vector; T2CCU, Timer 21, UART1, MDU, SSC, CORDIC, MultiCAN, CORDIC, A/D Converter and the Capture/Compare Unit share eight interrupt vectors. Two of these interrupt vectors are also shared with External Interrupts 2 to 6. External interrupts 0 to 1 are each assigned one dedicated interrupt vector.

The Non-Maskable Interrupt (NMI) is similar to regular interrupts, except it has the highest priority (over other regular interrupts) when addressing important system events. In the XC878, any one of the following five events can generate an NMI:

- WDT prewarning has occurred
- The PLL has lost the lock to the external crystal
- Flash Timer overflow has occurred
- VDDP is below the prewarning voltage level (4.0 V if the external power supply is 5.0 V)
- Flash ECC error has occurred

**Figure 5-1** to **Figure 5-5** give a general overview of the interrupt sources and nodes, and their corresponding control and status flags.

**Figure 5-6** gives the corresponding overview for the NMI sources.

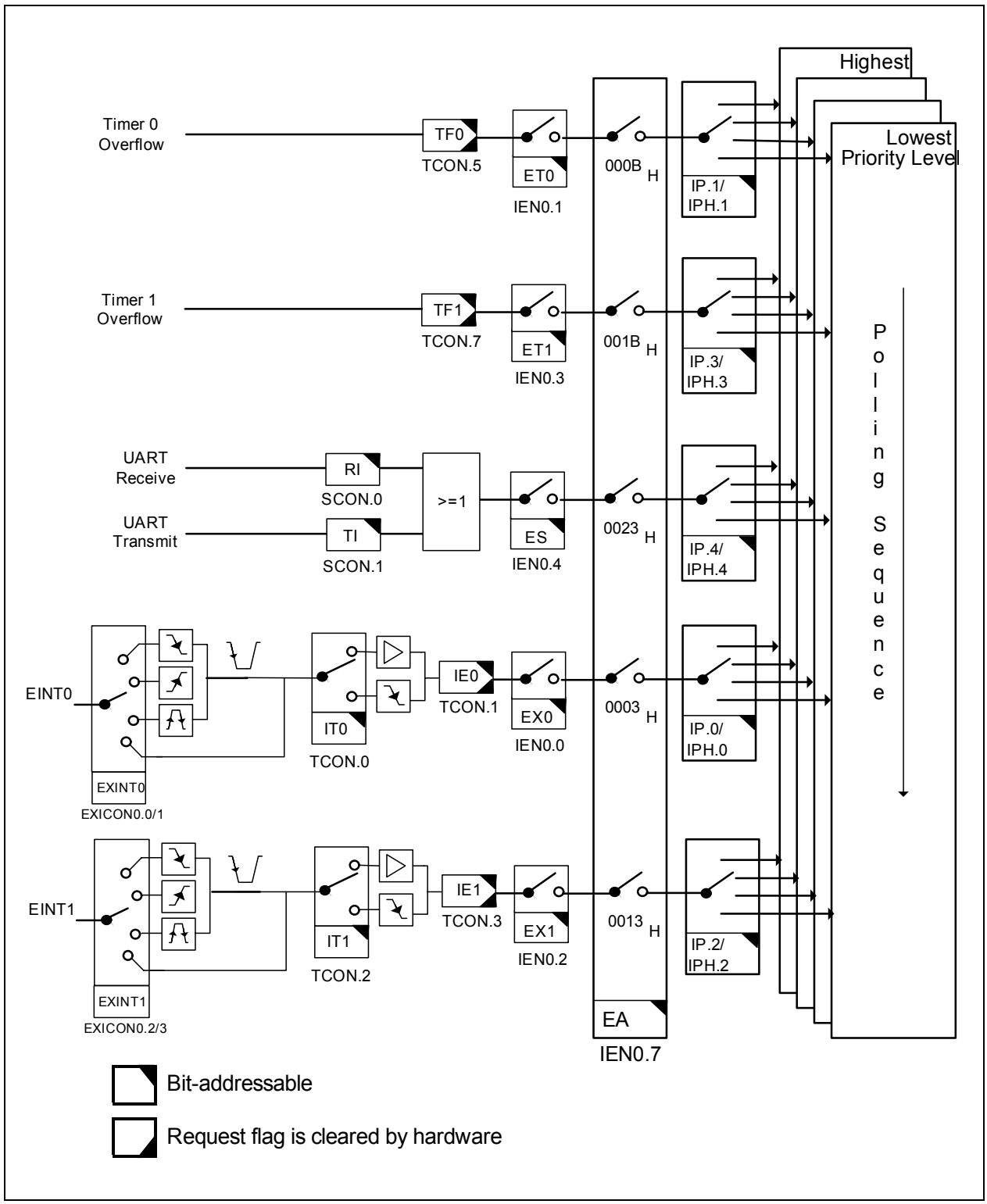



**Figure 5-1 Interrupt Request Sources (Part 1)**

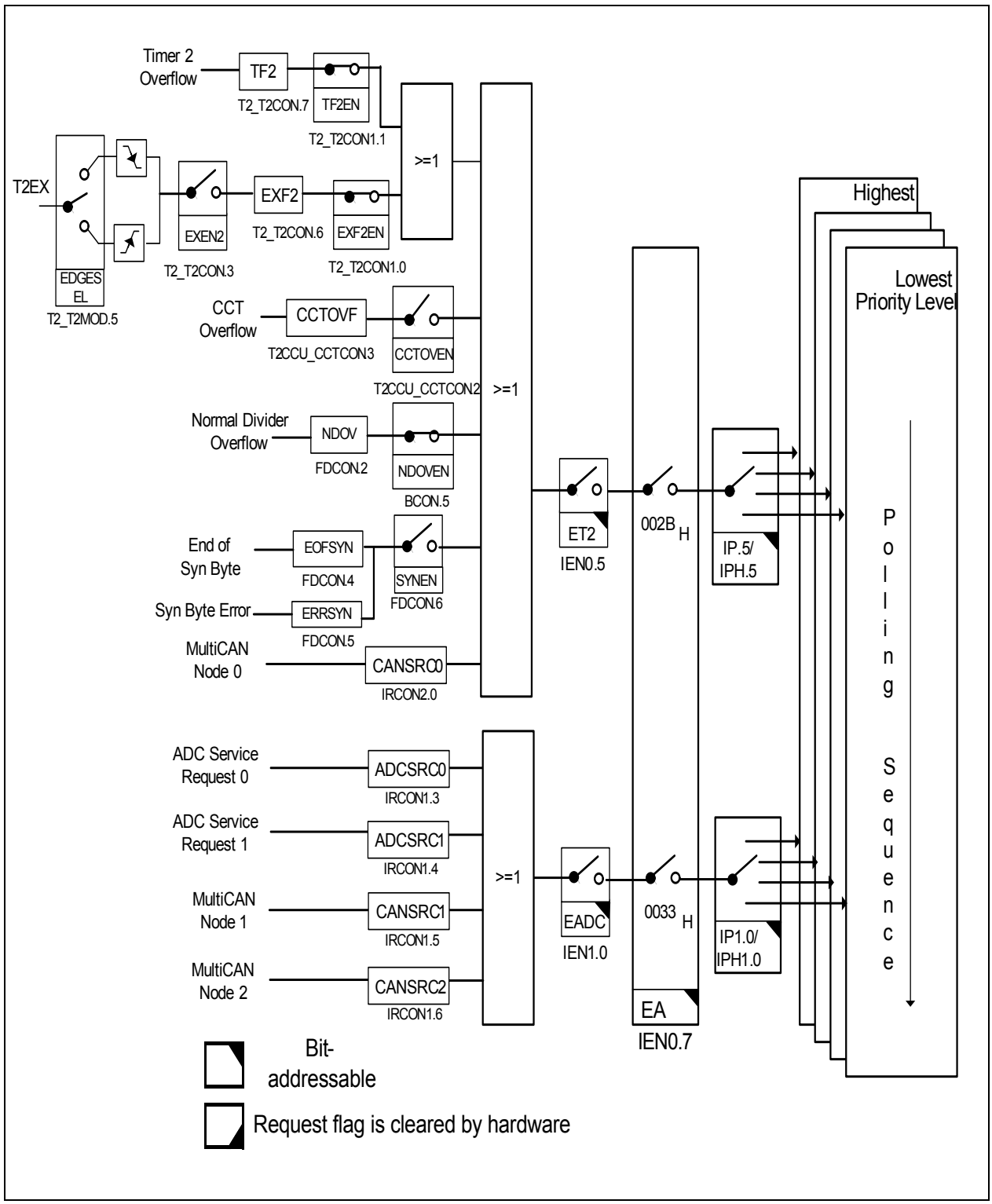



**Figure 5-2 Interrupt Request Sources (Part 2)**

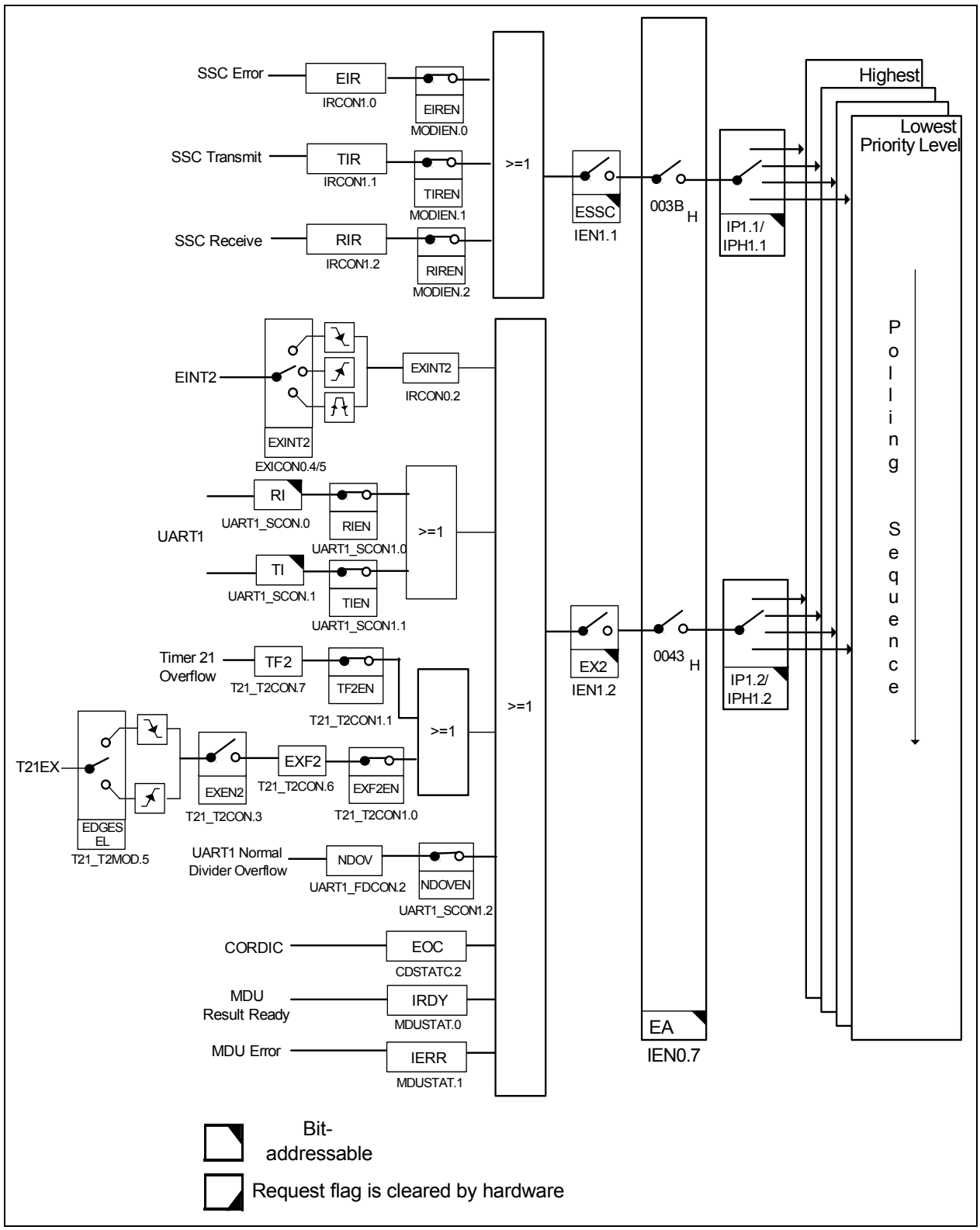



**Figure 5-3 Interrupt Request Sources (Part 3)**

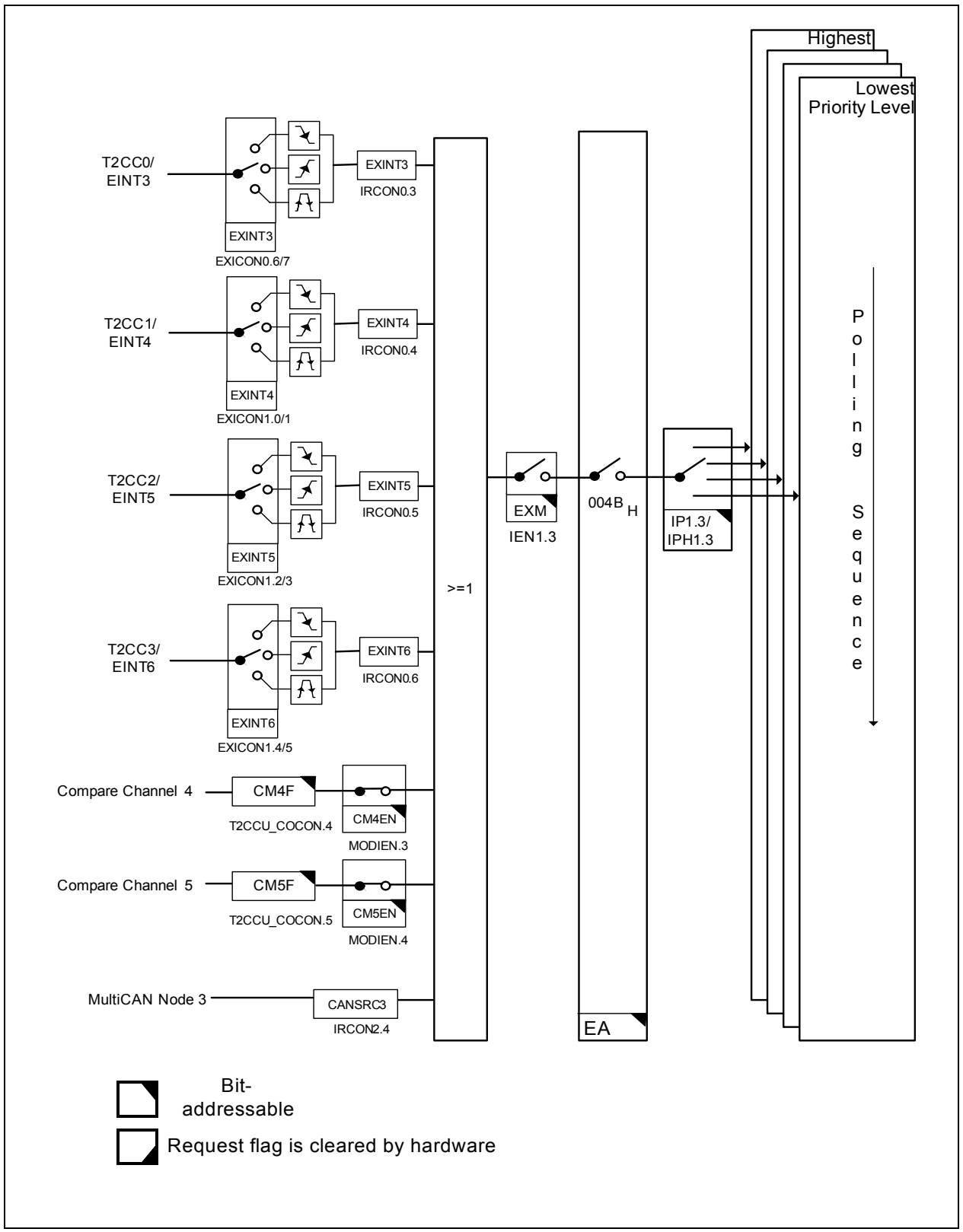



**Figure 5-4 Interrupt Request Sources (Part 4)**

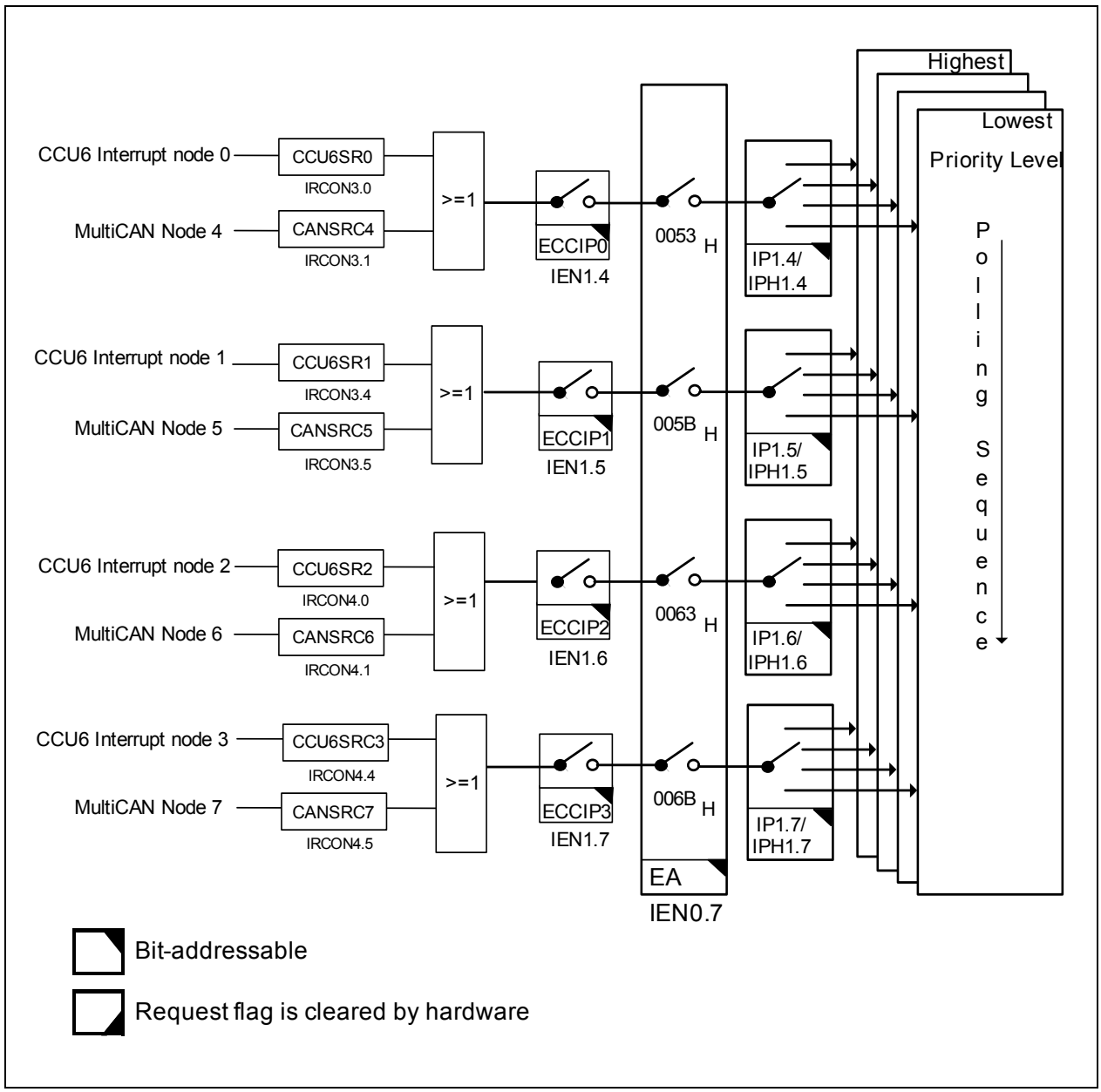



**Figure 5-5 Interrupt Request Sources (Part 5)**

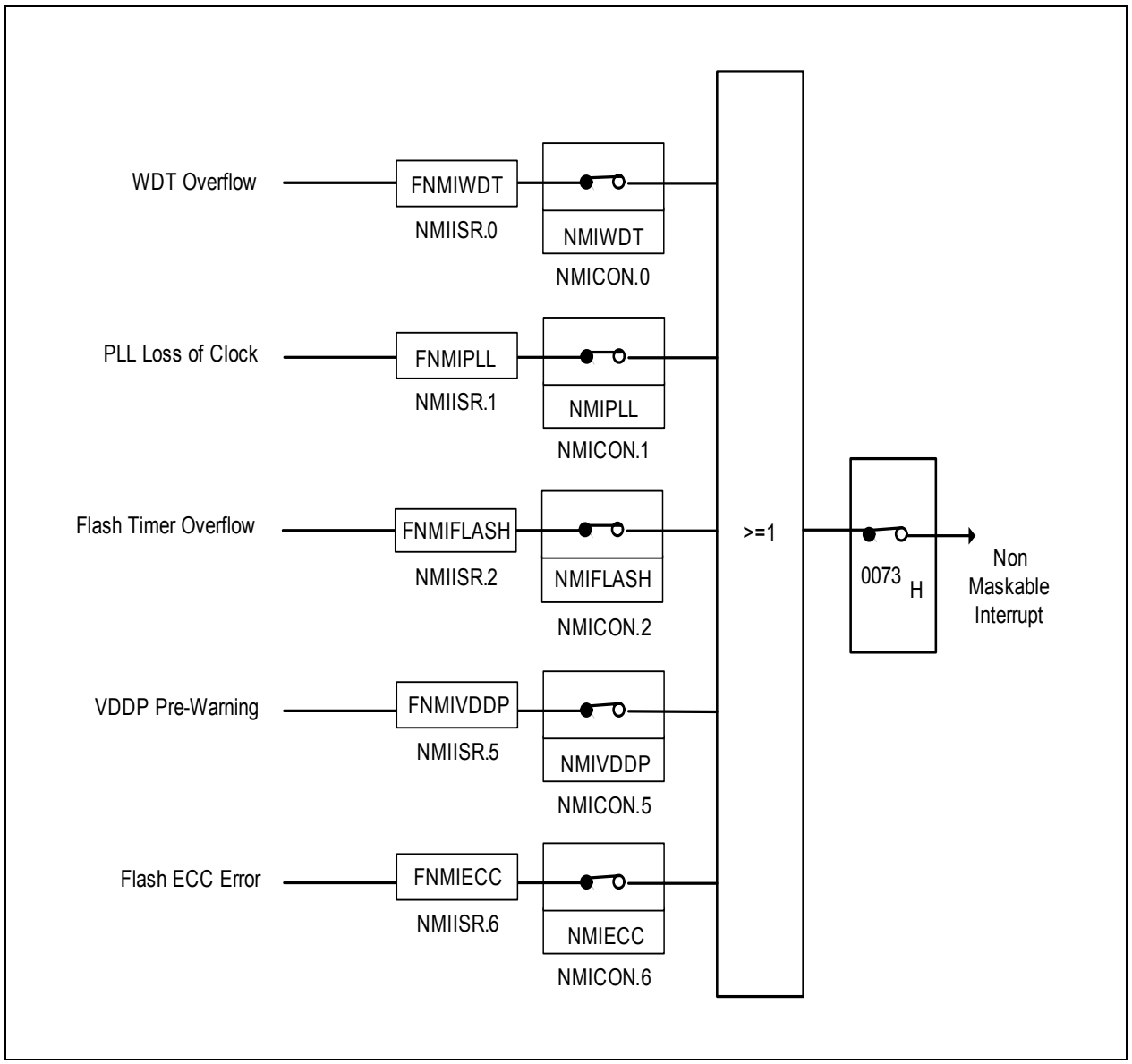



**Figure 5-6 Non-Maskable Interrupt Request Sources**

## 5.1 Interrupt Structure

An interrupt event source may be generated from the on-chip peripherals or from external. Detection of interrupt events is controlled by the respective on-chip peripherals. Interrupt status flags are available for determining which interrupt event has occurred, especially useful for an interrupt node which is shared by several event sources. Each interrupt node has a global enable/disable bit. In most cases, additional enable bits are provided for enabling/disabling particular interrupt events.

In general, the XC878 has two interrupt structures distinguished mainly by the manner in which the pending interrupt request (one per interrupt vector/source going directly to the core) is generated (due to the events) and cleared.

Common among these two interrupt structures is the interrupt masking bit, EA, which is used to globally enable or disable all interrupt requests (except NMI) to the core. Resetting bit EA to 0 only masks the pending interrupt requests from the core, but does not block the capture of incoming interrupt requests.

### 5.1.1 Interrupt Structure 1

For interrupt structure 1 in **Figure 5-7**, the interrupt event will set the interrupt status flag which doubles as a pending interrupt request to the core. An active pending interrupt request will interrupt the core only if its corresponding interrupt node is enabled. Once an interrupt node is serviced (interrupt acknowledged), its pending interrupt request (represented by the interrupt status flag) may be automatically cleared by hardware (the core).

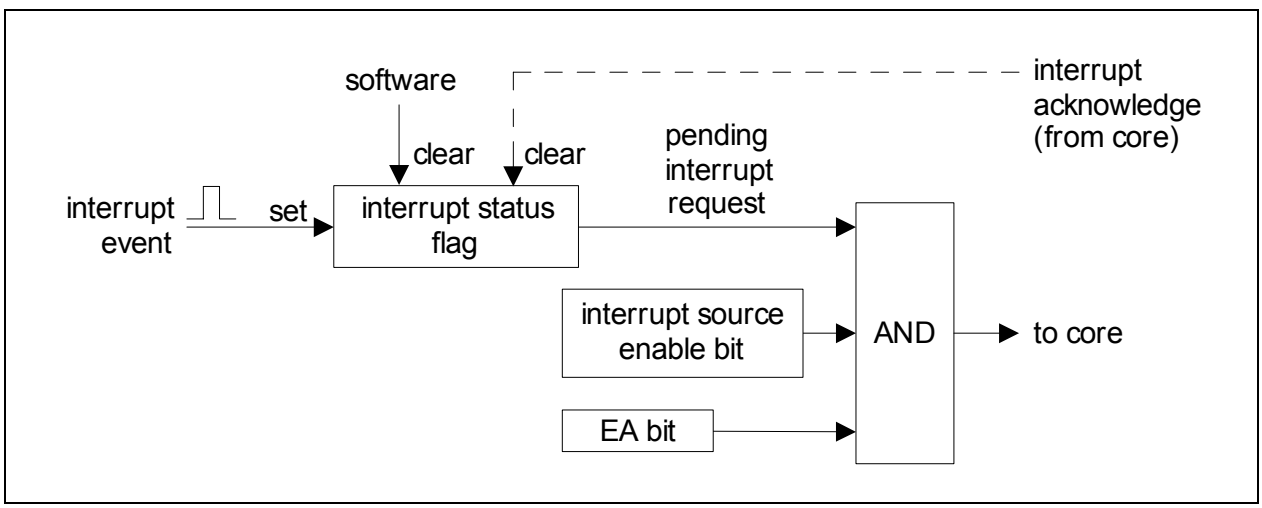



**Figure 5-7 Interrupt Structure 1**

For the XC878, interrupt sources Timer 0, Timer 1, external interrupt 0 and external interrupt 1 (each have a dedicated interrupt node) will have their respective interrupt status flags TF0, TF1, IE0 and IE1 in register TCON cleared by the core once their corresponding pending interrupt request is serviced. In the case that an interrupt node is

disabled (e.g., software polling is used), its interrupt status flag must be cleared by software since the core will not be interrupted (and therefore the interrupt acknowledge is not generated). For the UART module, interrupt status flags RI and TI in register SCON will not be cleared by the core even when its pending interrupt request is serviced. The UART module's interrupt status flags (and hence the pending interrupt request) can only be cleared by software.

### 5.1.2 Interrupt Structure 2

Interrupt structure 2 in **Figure 5-8** applies to Timer 2, UART1, LIN, external interrupts 2 to 6, ADC, SSC, CCU6, Flash, MDU and MultiCAN interrupt sources. For this structure, the interrupt status flag does not directly drive the pending interrupt request, which is latched due to an interrupt event. Further, an additional control bit IMODE in SYSCON0 register is used to select one of two defined modes of handling incoming interrupt events.

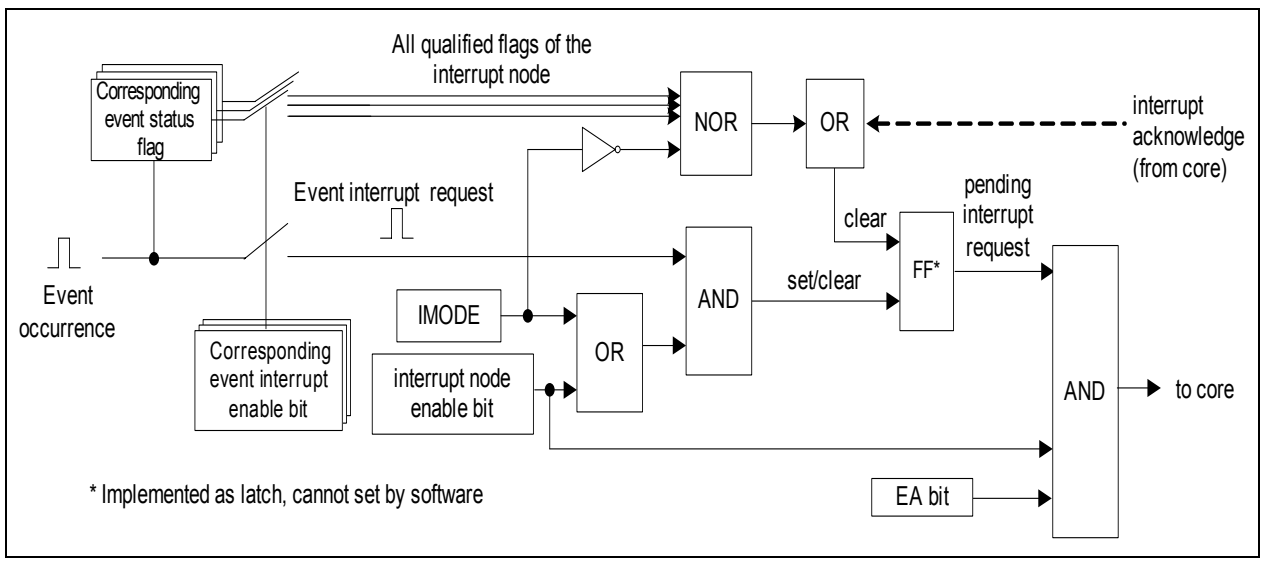



**Figure 5-8 Interrupt Structure 2**

If IMODE = 1, an event generated by its corresponding interrupt source will set the status flag, and in parallel, if the event is enabled for interrupt, activate the pending interrupt request. If IMODE = 0, an event will set the status flag, but the pending interrupt request is activated only if the event is enabled for interrupt and the interrupt node is enabled. Consider the case where interrupt event occurred while its interrupt node was disabled (assume global interrupt enable EA is set). When the interrupt node is enabled later, for IMODE = 1, previously activated pending interrupt request will now cause an active interrupt request to the core. On the contrary with IMODE = 0, there is no active interrupt request to the core due to previous events.

An active pending interrupt request interrupts the core and is automatically cleared by hardware (the core) once the interrupt node is serviced (interrupt acknowledged); the status flag remains set and must be cleared by software. A pending interrupt request can also be cleared by software; the method differs depending on the IMODE bit setting.

If IMODE = 1, only on clearing all interrupt-enabled status flags of the node will indirectly clear its pending interrupt request. Note that this is not exactly like interrupt structure 1 where the pending interrupt request is cleared directly by resetting the node's interrupt status flags. If IMODE = 0, only on clearing the interrupt node enable bit will indirectly clear its pending interrupt request.

Hence when IMODE = 0, the interrupt node enable bit additionally serves a dual function: to enable/disable the generation of pending interrupt request, and to clear an already generated pending interrupt request (by resetting enable bit to 0).

*Note: Interrupt structure 2 applies to the NMI, with the exclusion of EA bit and 'interrupt node enable bit' is replaced by OR of all NMICON bits. Therefore, NMI node is non-maskable when IMODE = 1; whereas NMI pending interrupt request may be cleared by clearing all NMICON bits when IMODE = 0*

### 5.1.2.1 System Control Register 0

The SYSCON0 register contains bits to select the SFR mapping and interrupt structure 2 mode.

**SYSCON0**
**System Control Register 0** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **0** | | | **IMODE** | **0** | **1** | **0** | **RMAP** |
| | r | | rw | r | **r** | **r** | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **IMODE** | 4 | rw | **Interrupt Structure 2 Mode Select**<br>0 Interrupt structure 2 mode 0 is selected.<br>1 Interrupt structure 2 mode 1 is selected. |
| **1** | 2 | r | **Reserved**<br>Returns 1if read; should be written with 1. |
| **0** | 1, 3, [7:5] | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

*Note: The IMODE bit should be cleared/set using ANL or ORL instructions.*

## 5.2 Interrupt Source and Vector

Each interrupt event source has an associated interrupt vector address for the interrupt node it belongs to. This vector is accessed to service the corresponding interrupt node request. The interrupt service of each interrupt node can be individually enabled or disabled via an enable bit. The assignment of the XC878 interrupt sources to the interrupt vector address and the corresponding interrupt node enable bits are summarized in **Table 5-1**.

**Table 5-1 Interrupt Vector Addresses**

| Interrupt Node | Vector Address | Assignment for XC878 | Enable Bit | SFR |
|---|---|---|---|---|
| NMI | $0073_H$ | Watchdog Timer NMI | NMIWDT | NMICON |
| | | PLL NMI | NMIPLL | |
| | | Flash Timer NMI | NMIFLASH | |
| | | $V_{DDP}$ Prewarning NMI | NMIVDDP | |
| | | Flash ECC NMI | NMIECC | |
| XINTR0 | $0003_H$ | External Interrupt 0 | EX0 | IEN0 |
| XINTR1 | $000B_H$ | Timer 0 | ET0 | |
| XINTR2 | $0013_H$ | External Interrupt 1 | EX1 | |
| XINTR3 | $001B_H$ | Timer 1 | ET1 | |
| XINTR4 | $0023_H$ | UART | ES | |
| XINTR5 | $002B_H$ | T2CCU | ET2 | |
| | | UART Fractional Divider (Normal Divider Overflow) | | |
| | | MultiCAN Node 0 | | |
| | | LIN | | |

**Table 5-1 Interrupt Vector Addresses** (cont'd)

| Interrupt Node | Vector Address | Assignment for XC878 | Enable Bit | SFR |
|---|---|---|---|---|
| XINTR6 | $0033_H$ | MultiCAN Nodes 1 and 2 | EADC | IEN1 |
| | | ADC[1:0] | | |
| XINTR7 | $003B_H$ | SSC | ESSC | |
| XINTR8 | $0043_H$ | External Interrupt 2 | EX2 | |
| | | T21 | | |
| | | CORDIC | | |
| | | UART1 | | |
| | | UART1 Fractional Divider (Normal Divider Overflow) | | |
| | | MDU[1:0] | | |
| XINTR9 | $004B_H$ | External Interrupt 3 | EXM | |
| | | External Interrupt 4 | | |
| | | External Interrupt 5 | | |
| | | External Interrupt 6 | | |
| | | T2CCU | | |
| | | MultiCAN Node 3 | | |
| XINTR10 | $0053_H$ | CCU6 INP0 | ECCIP0 | |
| | | MultiCAN Node 4 | | |
| XINTR11 | $005B_H$ | CCU6 INP1 | ECCIP1 | |
| | | MultiCAN Node 5 | | |
| XINTR12 | $0063_H$ | CCU6 INP2 | ECCIP2 | |
| | | MultiCAN Node 6 | | |
| XINTR13 | $006B_H$ | CCU6 INP3 | ECCIP3 | |
| | | MultiCAN Node 7 | | |

## 5.3 Interrupt Priority

An interrupt that is currently being serviced can only be interrupted by a higher-priority interrupt, but not by another interrupt of the same or lower priority. Hence, an interrupt of the highest priority cannot be interrupted by any other interrupt request.

If two or more requests of different priority levels are received simultaneously, the request with the highest priority is serviced first. If requests of the same priority are received simultaneously, an internal polling sequence determines which request is serviced first. Thus, within each priority level, there is a second priority structure determined by the polling sequence as shown in **Table 5-2**.

**Table 5-2 Priority Structure within Interrupt Level**

| Source | Level |
|---|---|
| Non-Maskable Interrupt (NMI) | (highest) |
| External Interrupt 0 | 1 |
| Timer 0 Interrupt | 2 |
| External Interrupt 1 | 3 |
| Timer 1 Interrupt | 4 |
| UART Interrupt | 5 |
| T2CCU,UART Normal Divider Overflow, LIN, MultiCAN Interrupt | 6 |
| ADC, MultiCAN Interrupt | 7 |
| SSC Interrupt | 8 |
| External Interrupt 2, Timer 21, UART1, UART1 Normal Divider Overflow, CORDIC, MDU Interrupt | 9 |
| External Interrupt [6:3], MultiCAN | 10 |
| CCU6 Interrupt Node Pointer 0, MultiCAN Interrupt | 11 |
| CCU6 Interrupt Node Pointer 1, MultiCAN Interrupt | 12 |
| CCU6 Interrupt Node Pointer 2, MultiCAN Interrupt | 13 |
| CCU6 Interrupt Node Pointer 3, MultiCAN Interrupt | 14 |

## 5.4 Interrupt Handling

The interrupt request signals are sampled at phase 2 in each machine cycle. The sampled requests are then polled during the following machine cycle. If one interrupt node request was active at phase 2 of the preceding cycle, the polling cycle will find it and the interrupt system will generate an LCALL to the appropriate service routine, provided this hardware-generated LCALL is not blocked by any of the following conditions:

1. An interrupt of equal or higher priority is already in progress.
2. The current (polling) cycle is not in the final cycle of the instruction in progress.
3. The instruction in progress is RETI or any write access to registers IEN0/IEN1 or IP,IPH/IP1,IP1H.

Any of these three conditions will block the generation of the LCALL to the interrupt service routine. Condition 2 ensures that the instruction in progress is completed before vectoring to any service routine. Condition 3 ensures that if the instruction in progress is RETI or any write access to registers IEN0/IEN1 or IP,IPH/IP1,IP1H, then at least one more instruction will be executed before any interrupt is vectored to; this delay guarantees that changes of the interrupt status can be observed by the CPU.

The polling cycle is repeated with each machine cycle, and the values polled are the values that were present at phase 2 of the previous machine cycle. Note that if any interrupt flag is active but was not responded to for one of the conditions already mentioned, or if the flag is no longer active at a later time when servicing the interrupt node, the corresponding interrupt source will not be serviced. In other words, the fact that the interrupt flag was once active but not serviced is not remembered. Every polling cycle interrogates only the pending interrupt requests.

The processor acknowledges an interrupt request by executing a hardware generated LCALL to the appropriate service routine. In some cases, hardware also clears the flag that generated the interrupt, while in other cases, the flag must be cleared by the user's software. The hardware-generated LCALL pushes the contents of the Program Counter (PC) onto the stack (but it does not save the PSW) and reloads the PC with an address that depends on the source of the interrupt being vectored to, as shown in the **Table 5-1**.

Program execution returns to the next instruction after calling the interrupt when the RETI instruction is encountered. The RETI instruction informs the processor that the interrupt routine is no longer in progress, then pops the two top bytes from the stack and reloads the PC. Execution of the interrupted program continues from the point where it was stopped. Note that the RETI instruction is important because it informs the processor that the program has left the current interrupt priority level. A simple RET instruction would also have returned execution to the interrupted program, but it would have left the interrupt control system on the assumption that an interrupt was still in progress. In this case, no interrupt of the same or lower priority level would be acknowledged.

## 5.5 Interrupt Response Time

Due to an interrupt event of (the various sources of) an interrupt node, its corresponding request signal will be sampled active at phase 2 in every machine cycle. The value is not polled by the circuitry until the next machine cycle. If the request is active and conditions are right for it to be acknowledged, a hardware subroutine call to the requested service routine will be the next instruction to be executed. The call itself takes two machine cycles. Thus, a minimum of three complete machine cycles will elapse from activation of the interrupt request to the beginning of execution of the first instruction of the service routine as shown in **Figure 5-9**.

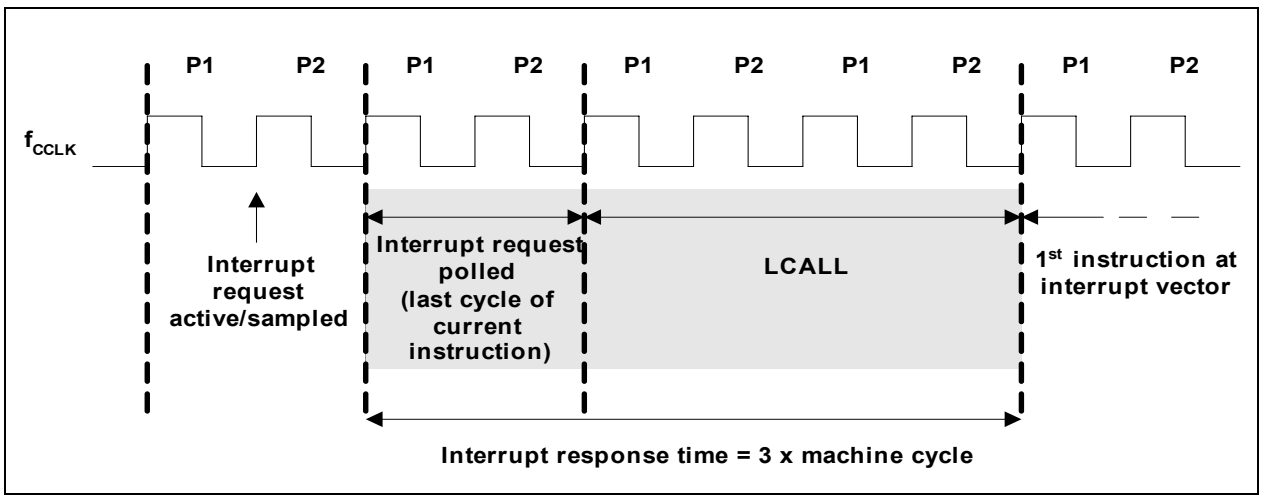



**Figure 5-9 Minimum Interrupt Response Time**

A longer response time would be obtained if the request is blocked by one of the three previously listed conditions:

1. If an interrupt of equal or higher priority is already in progress, the additional wait time will depend on the nature of the other interrupt's service routine.
2. If the instruction in progress is not in its final cycle, the additional wait time cannot be more than three machine cycles since the longest instructions (MUL and DIV) are only four machine cycles long. See **Figure 5-10**.
3. If the instruction in progress is RETI or a write access to registers IEN0, IEN1 or IP(H), IP1(H), the additional wait time cannot be more than five cycles (a maximum of one more machine cycle to complete the instruction in progress, plus four machine cycles to complete the next instruction, if the instruction is MUL or DIV). See **Figure 5-11**.

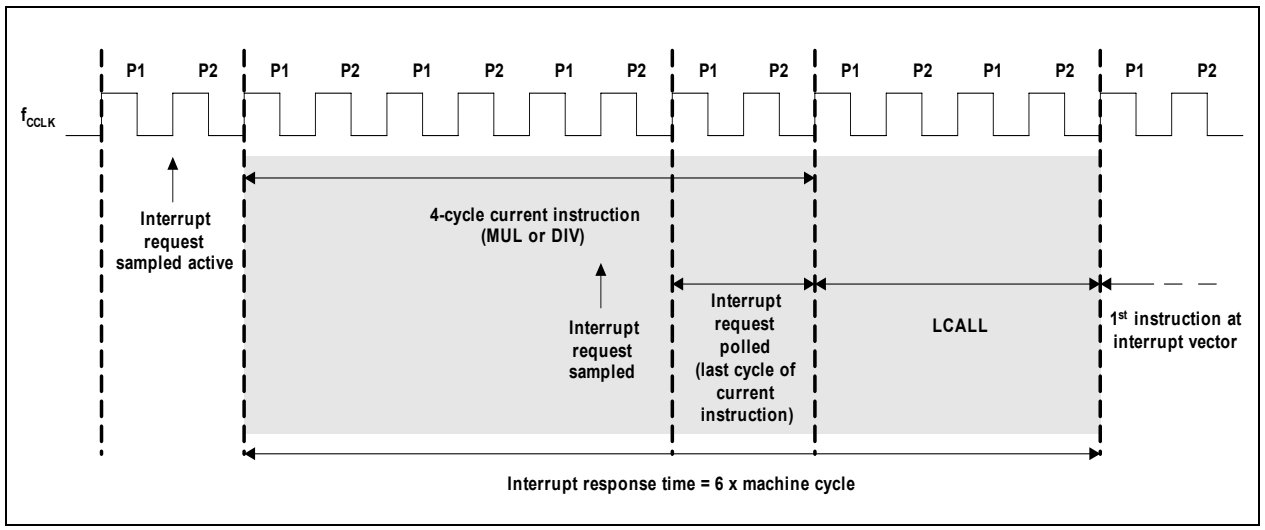



**Figure 5-10 Interrupt Response Time for Condition 2**

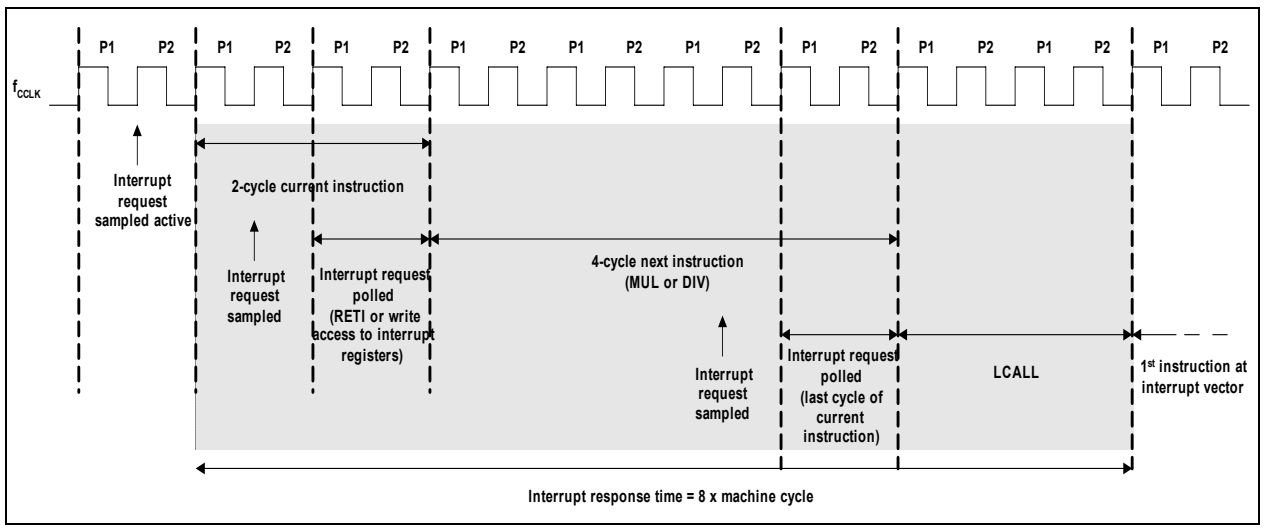



**Figure 5-11 Interrupt Response Time for Condition 3**

Thus in a single interrupt system, the response time is between three machine cycles and less than nine machine cycles if wait states are not considered. When considering wait states, the interrupt response time will be extended depending on the user instructions (except the hardware generated LCALL) being executed during the interrupt response time (shaded region in **Figure 5-10** and **Figure 5-11**).

## 5.6 Interrupt Registers

Interrupt registers are used for interrupt node enable, external interrupt control, interrupt flags and interrupt priority setting.

### 5.6.1 Interrupt Node Enable Registers

Each interrupt node can be individually enabled or disabled by setting or clearing the corresponding bit in the interrupt enable registers IEN0 or IEN1. Register IEN0 also contains the global interrupt masking bit (EA), which can be cleared to block all pending interrupt requests at once.

The NMI interrupt vector is shared by a number of sources, each of which can be enabled or disabled individually via register NMICON.

After reset, the enable bits in IEN0, IEN1 and NMICON are cleared to 0. This implies that all interrupt sources are disabled by default.

**IEN0**
**Interrupt Enable Register 0** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **EA** | **0** | **ET2** | **ES** | **ET1** | **EX1** | **ET0** | **EX0** |
| rw | r | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **EX0** | 0 | rw | **Interrupt Node XINTR0 Enable**<br>0 XINTR0 is disabled<br>1 XINTR0 is enabled |
| **ET0** | 1 | rw | **Interrupt Node XINTR1 Enable**<br>0 XINTR1 is disabled<br>1 XINTR1 is enabled |
| **EX1** | 2 | rw | **Interrupt Node XINTR2 Enable**<br>0 XINTR2 is disabled<br>1 XINTR2 is enabled |
| **ET1** | 3 | rw | **Interrupt Node XINTR3 Enable**<br>0 XINTR3 is disabled<br>1 XINTR3 is enabled |

| Field | Bits | Type | Description |
|---|---|---|---|
| **ES** | 4 | rw | **Interrupt Node XINTR4 Enable**<br>0 XINTR4 is disabled<br>1 XINTR4 is enabled |
| **ET2** | 5 | rw | **Interrupt Node XINTR5 Enable**<br>0 XINTR5 is disabled<br>1 XINTR5 is enabled |
| **EA** | 7 | rw | **Global Interrupt Mask**<br>0 All pending interrupt requests (except NMI) are blocked from the core.<br>1 Pending interrupt requests are not blocked from the core. |
| **0** | 6 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

**IEN1**
**Interrupt Enable Register 1** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **ECCIP3** | **ECCIP2** | **ECCIP1** | **ECCIP0** | **EXM** | **EX2** | **ESSC** | **EADC** |
| rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **EADC** | 0 | rw | **Interrupt Node XINTR6 Enable**<br>0 XINTR6 is disabled<br>1 XINTR6 is enabled |
| **ESSC** | 1 | rw | **Interrupt Node XINTR7 Enable**<br>0 XINTR7 is disabled<br>1 XINTR7 is enabled |
| **EX2** | 2 | rw | **Interrupt Node XINTR8 Enable**<br>0 XINTR8 is disabled<br>1 XINTR8 is enabled |
| **EXM** | 3 | rw | **Interrupt Node XINTR9 Enable**<br>0 XINTR9 is disabled<br>1 XINTR9 is enabled |

| Field | Bits | Type | Description |
|---|---|---|---|
| **ECCIP0** | 4 | rw | **Interrupt Node XINTR10 Enable**<br>0 XINTR10 is disabled<br>1 XINTR10 is enabled |
| **ECCIP1** | 5 | rw | **Interrupt Node XINTR11 Enable**<br>0 XINTR11 is disabled<br>1 XINTR11 is enabled |
| **ECCIP2** | 6 | rw | **Interrupt Node XINTR12 Enable**<br>0 XINTR12 is disabled<br>1 XINTR12 is enabled |
| **ECCIP3** | 7 | rw | **Interrupt Node XINTR13 Enable**<br>0 XINTR13 is disabled<br>1 XINTR13 is enabled |

**NMICON**
**NMI Control Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **0** | **NMIECC** | **NMIVDDP** | **0** | **NMIOCDS** | **NMIFLASH** | **NMIPLL** | **NMIWDT** |
| r | rw | rw | r | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **NMIWDT** | 0 | rw | **Watchdog Timer NMI Enable**<br>0 WDT NMI is disabled.<br>1 WDT NMI is enabled. |
| **NMIPLL** | 1 | rw | **PLL Loss of Clock NMI Enable**<br>0 PLL Loss of Clock NMI is disabled.<br>1 PLL Loss of Clock NMI is enabled. |
| **NMIFLASH** | 2 | rw | **Flash Timer NMI Enable**<br>0 Flash Timer NMI is disabled.<br>1 Flash Timer NMI is enabled. |
| **NMIOCDS** | 3 | rw | **OCDS NMI Enable**<br>0 OCDS NMI is disabled.<br>1 Reserved |

| Field | Bits | Type | Description |
|---|---|---|---|
| **NMIVDDP** | 5 | rw | **VDDP Prewarning NMI Enable**<br>0 $V_{DDP}$ NMI is disabled.<br>1 $V_{DDP}$ NMI is enabled.<br>*Note: When the external power supply is 3.3 V, the user must disable NMIVDDP.* |
| **NMIECC** | 6 | rw | **ECC NMI Enable**<br>0 ECC NMI is disabled.<br>1 ECC NMI is enabled. |
| **0** | 4, 7 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

### 5.6.2 External Interrupt Control Registers

The seven external interrupts, EXT_INT[6:0], are driven into the XC878 from the ports. External interrupts can be positive, negative, or double edge triggered. Registers EXICON0 and EXICON1 specify the active edge for the external interrupt. Among the external interrupts, external interrupt 0 and external interrupt 1 can be selected to bypass edge detection for direct feed-through to the core. This signal to the core can be further programmed to either low-level or negative transition activated, by the bits IT0 and IT1 in the TCON register. However for edge detection, TCON.IT0/1 must be set to falling edge triggered. An active edge event detected will generate internally two CCLK cycle low pulse for detection by core.

If the external interrupt is positive (negative) edge triggered, the external source must hold the request pin low (high) for at least one CCLK cycle, and then hold it high (low) for at least one CCLK cycle to ensure that the transition is recognized. If edge detection is bypassed for external interrupt 0 and external interrupt 1, the external source must hold the request pin “high” or “low” for at least two CCLK cycles.

External Interrupts 2 to 6 share their interrupt node with other interrupt sources. Therefore in addition to the corresponding interrupt node enable, each external interrupt 2 to 6 may be disabled individually, and are disabled by default after reset.

All the external interrupts support alternative input pin, selected via EXINTxIS bits in SFRs MODPISEL, MODPISEL1 and MODPISEL4. When switching inputs, the active edge/level trigger select and the level on the associated pins should be considered to prevent unintentional interrupt generation.

**EXICON0**
**External Interrupt Control Register 0** **Reset Value: $F0_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **EXINT3** | | **EXINT2** | | **EXINT1** | | **EXINT0** | |
| rw | | rw | | rw | | rw | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **EXINT0** | [1:0] | rw | **External Interrupt 0 Trigger Select**<br>00 Interrupt on falling edge<br>01 Interrupt on rising edge<br>10 Interrupt on both rising and falling edges<br>11 Bypass the edge detection. The interrupt request signal directly feeds to the core. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **EXINT1** | [3:2] | rw | **External Interrupt 1 Trigger Select**<br>00 Interrupt on falling edge<br>01 Interrupt on rising edge<br>10 Interrupt on both rising and falling edges<br>11 Bypass the edge detection.The interrupt request signal directly feeds to the core. |
| **EXINT2** | [5:4] | rw | **External Interrupt 2 Trigger Select**<br>00 Interrupt on falling edge<br>01 Interrupt on rising edge<br>10 Interrupt on both rising and falling edges<br>11 External interrupt 2 is disabled |
| **EXINT3** | [7:6] | rw | **External Interrupt 3 Trigger Select/ T2CCU Capture/Compare Channel 0**<br>00 Interrupt on falling edge<br>01 Interrupt on rising edge<br>10 Interrupt on both rising and falling edges<br>11 External interrupt 3 is disabled |

**EXICON1**
**External Interrupt Control Register 1** **Reset Value: $3F_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **0** | | **EXINT6** | | **EXINT5** | | **EXINT4** | |
| r | | rw | | rw | | rw | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **EXINT4** | [1:0] | rw | **External Interrupt 4 Trigger Select**<br>00 Interrupt on falling edge<br>01 Interrupt on rising edge<br>10 Interrupt on both rising and falling edges<br>11 External interrupt 4 is disabled |
| **EXINT5** | [3:2] | rw | **External Interrupt 5 Trigger Select**<br>00 Interrupt on falling edge<br>01 Interrupt on rising edge<br>10 Interrupt on both rising and falling edges<br>11 External interrupt 5 is disabled |

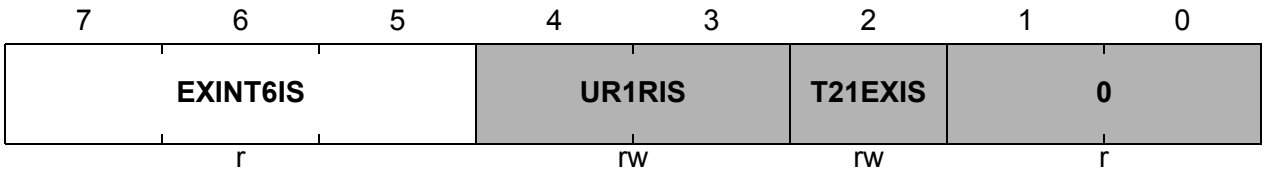
| Field | Bits | Type | Description |
|---|---|---|---|
| **EXINT6** | [5:4] | rw | **External Interrupt 6 Trigger Select**<br>00 Interrupt on falling edge<br>01 Interrupt on rising edge<br>10 Interrupt on both rising and falling edges<br>11 External interrupt 6 is disabled |
| **0** | [7:6] | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

**MODPISEL**
**Peripheral Input Select Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **0** | **URRISH** | **JTAGTDIS** | **JTAGTCKS** | **EXINT2IS** | **EXINT1IS** | **EXINT0IS** | **URRIS** |
| r | rw | rw | rw | rw | rw | rw | rw |

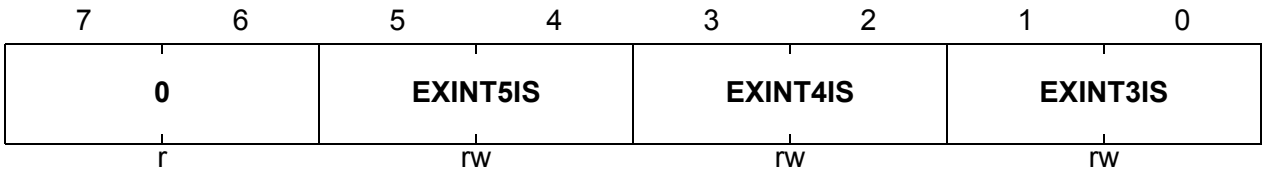
| Field | Bits | Type | Description |
|---|---|---|---|
| **EXINT0IS** | 1 | rw | **External Interrupt 0 Input Select**<br>0 External Interrupt Input EXINT0_0 is selected.<br>1 External Interrupt Input EXINT0_1 is selected. |
| **EXINT1IS** | 2 | rw | **External Interrupt 1 Input Select**<br>0 External Interrupt Input EXINT1_0 is selected.<br>1 External Interrupt Input EXINT1_1 is selected. |
| **EXINT2IS** | 3 | rw | **External Interrupt 2 Input Select**<br>0 External Interrupt Input EXINT2_0 is selected.<br>1 External Interrupt Input EXINT2_1 is selected. |
| **0** | 7 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

**MODPISEL1**
**Peripheral Input Select Register 1** **Reset Value: $00_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **EXINT6IS** | [7:5} | rw | **External Interrupt 6 Input/T2CCU Capture/Compare Channel 3 Select**<br>000 External Interrupt Input EXINT6_0 is selected.<br>001 External Interrupt Input EXINT6_1 is selected.<br>010 External Interrupt Input EXINT6_2/T2CCU Capture/Compare Channel T2CC3_0 is selected.<br>011 External Interrupt Input EXINT6_3/T2CCU Capture/Compare Channel T2CC3_1 is selected.<br>100 External Interrupt Input EXINT6_4/T2CCU Capture/Compare Channel T2CC3_2 is selected.<br>101 Reserved.<br>110 Reserved.<br>111 Reserved. |
| **0** | [1:0] | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

**MODPISEL4**
**Peripheral Input Select Register 4** **Reset Value: $00_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **EXINT3IS** | [1:0] | rw | **External Interrupt 3 Input/T2CCU Capture/Compare Channel 0 Select**<br>00 External Interrupt Input EXINT3_0 is selected.<br>01 External Interrupt Input EXINT3_1/T2CCU Capture/Compare Channel T2CC0_0 is selected.<br>10 External Interrupt Input EXINT3_2/T2CCU Capture/Compare Channel T2CC0_1 is selected<br>11 External Interrupt Input EXINT3_3/T2CCU Capture/Compare Channel T2CC0_2 is selected |
| **EXINT4IS** | [3:2] | rw | **External Interrupt 4 Input/T2CCU Capture/Compare Channel 1 Select**<br>00 External Interrupt Input EXINT4_0 is selected.<br>01 External Interrupt Input EXINT4_1/T2CCU Capture/Compare Channel T2CC1_0 is selected.<br>10 External Interrupt Input EXINT4_2/T2CCU Capture/Compare Channel T2CC1_1 is selected<br>11 External Interrupt Input EXINT4_3/T2CCU Capture/Compare Channel T2CC1_2 is selected |

| Field | Bits | Type | Description |
|---|---|---|---|
| **EXINT5IS** | [5:4] | rw | **External Interrupt 5 Input/T2CCU Capture/Compare Channel 2 Select**<br>00 External Interrupt Input EXINT5_0 is selected.<br>01 External Interrupt Input EXINT5_1/T2CCU Capture/Compare Channel T2CC2_0 is selected.<br>10 External Interrupt Input EXINT5_2/T2CCU Capture/Compare Channel T2CC2_1 is selected<br>11 External Interrupt Input EXINT5_3/T2CCU Capture/Compare Channel T2CC2_2 is selected |
| **0** | [7:6] | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

**TCON**
**Timer and Counter Control/Status Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **TF1** | **TR1** | **TF0** | **TR0** | **IE1** | **IT1** | **IE0** | **IT0** |
| rwh | rw | rwh | rw | rwh | rw | rwh | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **IT0** | 0 | rw | **External Interrupt 0 Level/Edge Trigger Control Flag**<br>0 Low-level triggered external interrupt 0 is selected.<br>1 Falling edge triggered external interrupt 0 is selected. |
| **IT1** | 2 | rw | **External Interrupt 1 Level/Edge Trigger Control Flag**<br>0 Low-level triggered external interrupt 1 is selected.<br>1 Falling edge triggered external interrupt 1 is selected. |

### 5.6.3 Interrupt Flag Registers

The interrupt flags for the different interrupt sources are located in several Special Function Registers (SFRs). In case of software and hardware access to a flag bit at the same time, hardware will have higher priority.

**IRCON0**
**Interrupt Request Register 0** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **0** | **EXINT6** | **EXINT5** | **EXINT4** | **EXINT3** | **EXINT2** | **EXINT1** | **EXINT0** |
| r | rwh | rwh | rwh | rwh | rwh | rwh | rwh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **EXINTx (x = 0 - 1)** | [1:0] | rwh | **Interrupt Flag for External Interrupt 0/1**<br>This bit is set by hardware and can only be cleared by software.<br>0 Interrupt event has not occurred.<br>1 Interrupt event has occurred.<br>These bits are set by corresponding active edge event i.e. falling/rising/both. These flags are 'dummy' and has no effect on the respective interrupt signal to core. Instead, the corresponding TCON flag is the interrupt request to the core - it is sufficient to poll and clear the TCON flag. |
| **EXINT2** | 2 | rwh | **Interrupt Flag for External Interrupt 2**<br>This bit is set by hardware and can only be cleared by software.<br>0 Interrupt event has not occurred.<br>1 Interrupt event has occurred. |
| **EXINTy (y = 3 - 6)** | [6:3] | rwh | **Interrupt Flag for External Interrupt y or T2CCU Capture/Compare Channel z(z = 0-3)**<br>This bit is set by hardware and can only be cleared by software.<br>0 Interrupt event has not occurred.<br>1 Interrupt event has occurred. |
| **0** | 7 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

**IRCON1**
**Interrupt Request Register 1** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **0** | **CANSRC2** | **CANSRC1** | **ADCSR1** | **ADCSR0** | **RIR** | **TIR** | **EIR** |
| r | rwh | rwh | rwh | rwh | rwh | rwh | rwh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **EIR** | 0 | rwh | **Error Interrupt Flag for SSC**<br>This bit is set by hardware and can only be cleared by software.<br>0 Interrupt event has not occurred.<br>1 Interrupt event has occurred. |
| **TIR** | 1 | rwh | **Transmit Interrupt Flag for SSC**<br>This bit is set by hardware and can only be cleared by software.<br>0 Interrupt event has not occurred.<br>1 Interrupt event has occurred. |
| **RIR** | 2 | rwh | **Receive Interrupt Flag for SSC**<br>This bit is set by hardware and can only be cleared by software.<br>0 Interrupt event has not occurred.<br>1 Interrupt event has occurred. |
| **ADCSR0** | 3 | rwh | **Interrupt Flag 0 for ADC**<br>This bit is set by hardware and can only be cleared by software.<br>0 Interrupt event has not occurred.<br>1 Interrupt event has occurred. |
| **ADCSR1** | 4 | rwh | **Interrupt Flag 1 for ADC**<br>This bit is set by hardware and can only be cleared by software.<br>0 Interrupt event has not occurred.<br>1 Interrupt event has occurred. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **CANSRC1** | 5 | rwh | **Interrupt Flag 1 for MultiCAN**<br>This bit is set by hardware and can only be cleared by software.<br>0 Interrupt event has not occurred.<br>1 Interrupt event has occurred. |
| **CANSRC2** | 6 | rwh | **Interrupt Flag 2 for MultiCAN**<br>This bit is set by hardware and can only be cleared by software.<br>0 Interrupt event has not occurred.<br>1 Interrupt event has occurred. |
| **0** | 7 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

**IRCON2**
**Interrupt Request Register 2** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **0** | | | **CANSRC3** | **0** | | | **CANSRC0** |
| r | | | rwh | r | | | rwh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **CANSRC0** | 0 | rwh | **Interrupt Flag 0 for MultiCAN**<br>This bit is set by hardware and can only be cleared by software.<br>0 Interrupt event has not occurred.<br>1 Interrupt event has occurred. |
| **CANSRC3** | 3 | rwh | **Interrupt Flag 3 for MultiCAN**<br>This bit is set by hardware and can only be cleared by software.<br>0 Interrupt event has not occurred.<br>1 Interrupt event has occurred. |
| **0** | [3:1], [7:5] | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

**IRCON3**
**Interrupt Request Register 3** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **0** | | **CANSRC5** | **CCU6SR1** | **0** | | **CANSRC4** | **CCU6SR0** |
| r | | rwh | rwh | r | | rwh | rwh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **CCU6SR0** | 0 | rwh | **Interrupt Flag 0 for CCU6**<br>This bit is set by hardware and can only be cleared by software.<br>0 Interrupt event has not occurred.<br>1 Interrupt event has occurred. |
| **CANSRC4** | 1 | rwh | **Interrupt Flag 4 for MultiCAN**<br>This bit is set by hardware and can only be cleared by software.<br>0 Interrupt event has not occurred.<br>1 Interrupt event has occurred. |
| **CCU6SR1** | 4 | rwh | **Interrupt Flag 1 for CCU6**<br>This bit is set by hardware and can only be cleared by software.<br>0 Interrupt event has not occurred.<br>1 Interrupt event has occurred. |
| **CANSRC5** | 5 | rwh | **Interrupt Flag 5 for MultiCAN**<br>This bit is set by hardware and can only be cleared by software.<br>0 Interrupt event has not occurred.<br>1 Interrupt event has occurred. |
| **0** | [3:2], [7:6] | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

**IRCON4**
**Interrupt Request Register 4** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **0** | | **CANSRC7** | **CCU6SR3** | **0** | | **CANSRC6** | **CCU6SR2** |
| r | | rwh | rwh | r | | rwh | rwh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **CCU6SR2** | 0 | rwh | **Interrupt Flag 2 for CCU6**<br>This bit is set by hardware and can only be cleared by software.<br>0 Interrupt event has not occurred.<br>1 Interrupt event has occurred. |
| **CANSRC6** | 1 | rwh | **Interrupt Flag 6 for MultiCAN**<br>This bit is set by hardware and can only be cleared by software.<br>0 Interrupt event has not occurred.<br>1 Interrupt event has occurred. |
| **CCU6SR3** | 4 | rwh | **Interrupt Flag 3 for CCU6**<br>This bit is set by hardware and can only be cleared by software.<br>0 Interrupt event has not occurred.<br>1 Interrupt event has occurred. |
| **CANSRC7** | 5 | rwh | **Interrupt Flag 7 for MultiCAN**<br>This bit is set by hardware and can only be cleared by software.<br>0 Interrupt event has not occurred.<br>1 Interrupt event has occurred. |
| **0** | [3:2], [7:6] | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

**TCON**
**Timer Control Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **TF1** | **TR1** | **TF0** | **TR0** | **IE1** | **IT1** | **IE0** | **IT0** |
| rwh | rw | rwh | rw | rwh | rw | rwh | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **IE0** | 1 | rwh | **External Interrupt 0 Flag**<br>Set by hardware when external interrupt 0 event is detected.<br>Cleared by hardware when processor vectors to interrupt routine. Can also be cleared by software. |
| **IE1** | 3 | rwh | **External Interrupt 1 Flag**<br>Set by hardware when external interrupt 1 event is detected.<br>Cleared by hardware when processor vectors to interrupt routine. Can also be cleared by software. |
| **TF0** | 5 | rwh | **Timer 0 Overflow Flag**<br>Set by hardware on Timer 0 overflow.<br>Cleared by hardware when processor vectors to interrupt routine. Can also be cleared by software. |
| **TF1** | 7 | rwh | **Timer 1 Overflow Flag**<br>Set by hardware on Timer 1 overflow.<br>Cleared by hardware when processor vectors to interrupt routine. Can also be cleared by software. |

## SCON
**Serial Channel Control Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **SM0** | **SM1** | **SM2** | **REN** | **TB8** | **RB8** | **TI** | **RI** |
| rw | rw | rw | rw | rw | rwh | rwh | rwh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **RI** | 0 | rwh | **Serial Interface Receiver Interrupt Flag**<br>Set by hardware if a serial data byte has been received. Must be cleared by software. |
| **TI** | 1 | rwh | **Serial Interface Transmitter Interrupt Flag**<br>Set by hardware at the end of a serial data transmission. Must be cleared by software. |

## NMISR
**NMI Status Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **0** | **FNMIECC** | **FNMI VDDP** | **0** | **FNMI OCDS** | **FNMI FLASH** | **FNMIPLL** | **FNMIWDT** |
| r | rwh | rwh | r | rwh | r | rwh | rwh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **FNMIWDT** | 0 | rwh | **Watchdog Timer NMI Flag**<br>0 No Watchdog Timer NMI has occurred.<br>1 Watchdog Timer prewarning has occurred. |
| **FNMIPLL** | 1 | rwh | **PLL NMI Flag**<br>0 No PLL NMI has occurred.<br>1 PLL loss-of-clock to the external crystal has occurred. |
| **FNMIFLASH** | 2 | rwh | **FLASH Timer NMI Flag**<br>0 No FlashTimer NMI has occurred.<br>1 Flash Timer Overflow has occurred. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **FNMIOCDS** | 3 | rwh | **OCDS NMI Flag**<br>0 No OCDS NMI has occurred.<br>1 Reserved |
| **FNMIVDDP** | 5 | rwh | **VDDP Prewarning NMI Flag**<br>0 No $V_{DDP}$ NMI occurred.<br>1 $V_{DDP}$ prewarning (drop to 4.0 V for external power supply of 5.0 V) has occurred. |
| **FNMIECC** | 6 | rwh | **ECC NMI Flag**<br>0 No ECC error has occurred.<br>1 ECC error has occurred. |
| **0** | 4,7 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

Register NMISR can only be cleared by software or reset to the default value after the power-on reset/hardware reset/brownout reset. The register value is retained on any other reset such as watchdog timer reset or power-down wake-up reset. This allows the system to detect what caused the previous NMI.

### 5.6.4 Interrupt Priority Registers

Each interrupt source can be individually programmed to one of the four available priority levels. Two pairs of interrupt priority registers are available to program the priority level of each interrupt vector. The first pair of Interrupt Priority Registers are SFRs IP and IPH. The second pair of Interrupt Priority Registers are SFRs IP1 and IPH1.

The corresponding bits in each pair of Interrupt Priority Registers select one of the four priority levels shown in **Table 5-3**.

**Table 5-3 Interrupt Priority Level Selection**

| IPH.x / IPH1.x | IP.x / IP1.x | Priority Level |
|---|---|---|
| 0 | 0 | Level 0 (lowest) |
| 0 | 1 | Level 1 |
| 1 | 0 | Level 2 |
| 1 | 1 | Level 3 (highest) |

*Note: NMI always has the highest priority (above Level 3), it does not use the level selection shown in **Table 5-3**.*

**IP**
**Interrupt Priority Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **0** | | **PT2** | **PS** | **PT1** | **PX1** | **PT0** | **PX0** |
| r | | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **PX0** | 0 | rw | **Priority Level Low Bit for Interrupt Node XINTR0** |
| **PT0** | 1 | rw | **Priority Level Low Bit for Interrupt Node XINTR1** |
| **PX1** | 2 | rw | **Priority Level Low Bit for Interrupt Node XINTR2** |
| **PT1** | 3 | rw | **Priority Level Low Bit for Interrupt Node XINTR3** |
| **PS** | 4 | rw | **Priority Level Low Bit for Interrupt Node XINTR4** |
| **PT2** | 5 | rw | **Priority Level Low Bit for Interrupt Node XINTR5** |
| **0** | [7:6] | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

## IPH
## Interrupt Priority High Register

**Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **0** | | **PT2H** | **PSH** | **PT1H** | **PX1H** | **PT0H** | **PX0H** |
| r | | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **PX0H** | 0 | rw | Priority Level High Bit for Interrupt Node XINTR0 |
| **PT0H** | 1 | rw | **Priority Level High Bit for Interrupt Node XINTR1** |
| **PX1H** | 2 | rw | **Priority Level High Bit for Interrupt Node XINTR2** |
| **PT1H** | 3 | rw | **Priority Level High Bit for Interrupt Node XINTR3** |
| **PSH** | 4 | rw | **Priority Level High Bit for Interrupt Node XINTR4** |
| **PT2H** | 5 | rw | **Priority Level High Bit for Interrupt Node XINTR5** |
| **0** | [7:6] | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

## IP1
## Interrupt Priority 1 Register

**Reset Value: $04_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **PCCIP3** | **PCCIP2** | **PCCIP1** | **PCCIP0** | **PXM** | **PX2** | **PSSC** | **PADC** |
| rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **PADC** | 0 | rw | **Priority Level Low Bit for Interrupt Node XINTR6** |
| **PSSC** | 1 | rw | **Priority Level Low Bit for Interrupt Node XINTR7** |
| **PX2** | 2 | rw | **Priority Level Low Bit for Interrupt Node XINTR8** |
| **PXM** | 3 | rw | **Priority Level Low Bit for Interrupt Node XINTR9** |
| **PCCIP0** | 4 | rw | **Priority Level Low Bit for Interrupt Node XINTR10** |
| **PCCIP1** | 5 | rw | **Priority Level Low Bit for Interrupt Node XINTR11** |
| **PCCIP2** | 6 | rw | **Priority Level Low Bit for Interrupt Node XINTR12** |

| Field | Bits | Type | Description |
|---|---|---|---|
| **PCCIP3** | 7 | rw | **Priority Level Low Bit for Interrupt Node XINTR13** |

**IPH1**
**Interrupt Priority 1 High Register** **Reset Value: $04_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **PCCIP3H** | **PCCIP2H** | **PCCIP1H** | **PCCIP0H** | **PXMH** | **PX2H** | **PSSCH** | **PADCH** |
| rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **PADCH** | 0 | rw | **Priority Level High Bit for Interrupt Node XINTR6** |
| **PSSCH** | 1 | rw | **Priority Level High Bit for Interrupt Node XINTR7** |
| **PX2H** | 2 | rw | **Priority Level High Bit for Interrupt Node XINTR8** |
| **PXMH** | 3 | rw | **Priority Level High Bit for Interrupt Node XINTR9** |
| **PCCIP0H** | 4 | rw | **Priority Level High Bit for Interrupt Node XINTR10** |
| **PCCIP1H** | 5 | rw | **Priority Level High Bit for Interrupt Node XINTR11** |
| **PCCIP2H** | 6 | rw | **Priority Level High Bit for Interrupt Node XINTR12** |
| **PCCIP3H** | 7 | rw | **Priority Level High Bit for Interrupt Node XINTR13** |

## 5.7 Interrupt Flag Overview

The interrupt events have interrupt flags that are located in different SFRs. Table 5-4 provides the corresponding SFR to which each interrupt flag belongs. Detailed information on the interrupt flags is provided in the respective peripheral chapters.

**Table 5-4 Locations of the Interrupt Request Flags**

| Interrupt Source | Interrupt Flag | SFR |
|---|---|---|
| Timer 0 Overflow | TF0 | TCON |
| Timer 1 Overflow | TF1 | TCON |
| Timer 2 Overflow | TF2 | T2_T2CON |
| Timer 2 External Event | EXF2 | T2_T2CON |
| Timer 21 Overflow | TF2 | T21_T2CON |
| Timer 21 External Event | EXF2 | T21_T2CON |
| LIN End of Syn Byte | EOFSYN | FDCON |
| LIN Syn Byte Error | ERRSYN | FDCON |
| UART Receive | RI | SCON |
| UART Transmit | TI | SCON |
| UART Normal Divider Overflow | NDOV | FDCON |
| UART1 Receive | RI | UART1_SCON |
| UART1 Transmit | TI | UART1_SCON |
| UART1 Normal Divider Overflow | NDOV | UART1_FDCON |
| External Interrupt 0 | IE0 | TCON |
| External Interrupt 1 | IE1 | TCON |
| External Interrupt 2 | EXINT2 | IRCON0 |
| External Interrupt 3/T2CC0 | EXINT3 | IRCON0 |
| External Interrupt 4/T2CC1 | EXINT4 | IRCON0 |
| External Interrupt 5/T2CC2 | EXINT5 | IRCON0 |
| External Interrupt 6/T2CC3 | EXINT6 | IRCON0 |
| T2CC4 | CM4F | T2CCU_COCON |
| T2CC5 | CM5F | T2CCU_COCON |
| T2CCU CCT Overflow | CCTOVF | T2CCU_CCTCON |
| MDU Result Ready | IRDY | MDUSTAT |
| MDU Error | IERR | MDUSTAT |

**Table 5-4 Locations of the Interrupt Request Flags** (cont'd)

| Interrupt Source | Interrupt Flag | SFR |
|---|---|---|
| A/D Converter Service Request 0 | ADCSR0 | IRCON1 |
| A/D Converter Service Request 1 | ADCSR1 | IRCON1 |
| SSC Error | EIR | IRCON1 |
| SSC Transmit | TIR | IRCON1 |
| SSC Receive | RIR | IRCON1 |
| CORDIC Interrupt | EOC | CDSTATC |
| MultiCAN Interrupt 0 | CANSRC0[1)] | IRCON2 |
| MultiCAN Interrupt 1 | CANSRC1[1)] | IRCON1 |
| MultiCAN Interrupt 2 | CANSRC2[1)] | IRCON1 |
| MultiCAN Interrupt 3 | CANSRC3[1)] | IRCON2 |
| MultiCAN Interrupt 4 | CANSRC4[1)] | IRCON3 |
| MultiCAN Interrupt 5 | CANSRC5[1)] | IRCON3 |
| MultiCAN Interrupt 6 | CANSRC6[1)] | IRCON4 |
| MultiCAN Interrupt 7 | CANSRC7[1)] | IRCON4 |
| CCU6 Node 0 Interrupt | CCU6SR0 | IRCON3 |
| CCU6 Node 1 Interrupt | CCU6SR1 | IRCON3 |
| CCU6 Node 2 Interrupt | CCU6SR2 | IRCON4 |
| CCU6 Node 3 Interrupt | CCU6SR3 | IRCON4 |
| Watchdog Timer NMI | FNMIWDT | NMISR |
| PLL NMI | FNMIPLL | NMISR |
| FLASH Timer Overflow NMI | FNMIFLASH | NMISR |
| $V_{DDP}$ Prewarning NMI | FNMIVDDP | NMISR |
| Flash ECC NMI | FNMIECC | NMISR |

[1)] Different MultiCAN interrupt can be assigned to different MultiCAN interrupt output lines [7:0] via MultiCAN registers NIPRx/MOIPRn.

# 6 Parallel Ports

The XC878 has 40 port pins organized into five parallel ports: Port 0 (P0), Port 1 (P1), Port 3 (P3), Port 4 (P4) and Port 5 (P5). Each pin has a pair of internal pull-up and pull-down devices that can be individually enabled or disabled. These ports are bidirectional and can be used as general purpose input/output (GPIO) or to perform alternate input/output functions for the on-chip peripherals. When configured as an output, the open drain mode can be selected.

**Bidirectional Port Features:**

- Configurable pin direction
- Configurable pull-up/pull-down devices
- Configurable open drain mode
- Configurable drive strength
- Transfer data through digital inputs and outputs (general purpose I/O)
- Alternate input/output for on-chip peripherals

## 6.1 General Description

**Figure 6-1** shows the block diagram of an XC878 bidirectional port pin. Each port pin is equipped with a number of control and data bits, thus enabling very flexible usage of the pin. By defining the contents of the control register, each individual pin can be configured as an input or an output. The user can also configure each pin as an open drain pin with or without internal pull-up/pull-down device.

Each bidirectional port pin can be configured for input or output operation. Switching between input and output mode is accomplished through the register Px_DIR (x = 0, 1, 3, 4 or 5), which enables or disables the output and input drivers. A port pin can only be configured as either input or output mode at any one time.

In input mode (default after reset), the output driver is switched off (high-impedance). The actual voltage level present at the port pin is translated into a logic 0 or 1 via a Schmitt-Trigger device and can be read via the register Px_DATA.

In output mode, the output driver is activated and drives the value supplied through the multiplexer to the port pin. In the output driver, each port line can be switched to open drain mode or normal mode (push-pull mode) via the register Px_OD. The drive strength of the output driver can also be configured via the register Px_DS.

The output multiplexer in front of the output driver enables the port output function to be used for different purposes. If the pin is used for general purpose output, the multiplexer is switched by software to the data register Px_DATA. Software can set or clear the bit in Px_DATA and therefore directly influence the state of the port pin. If an on-chip peripheral uses the pin for output signals, alternate output lines (AltDataOut) can be switched via the multiplexer to the output driver circuitry. Selection of the alternate function is defined in registers Px_ALTSEL0 and Px_ALTSEL1. When a port pin is used as an alternate function, its direction must be set accordingly in the register Px_DIR.

Each pin can also be programmed to activate an internal weak pull-up or pull-down device. Register Px_PUDSEL selects whether a pull-up or the pull-down device is activated while register Px_PUDEN enables or disables the pull device.

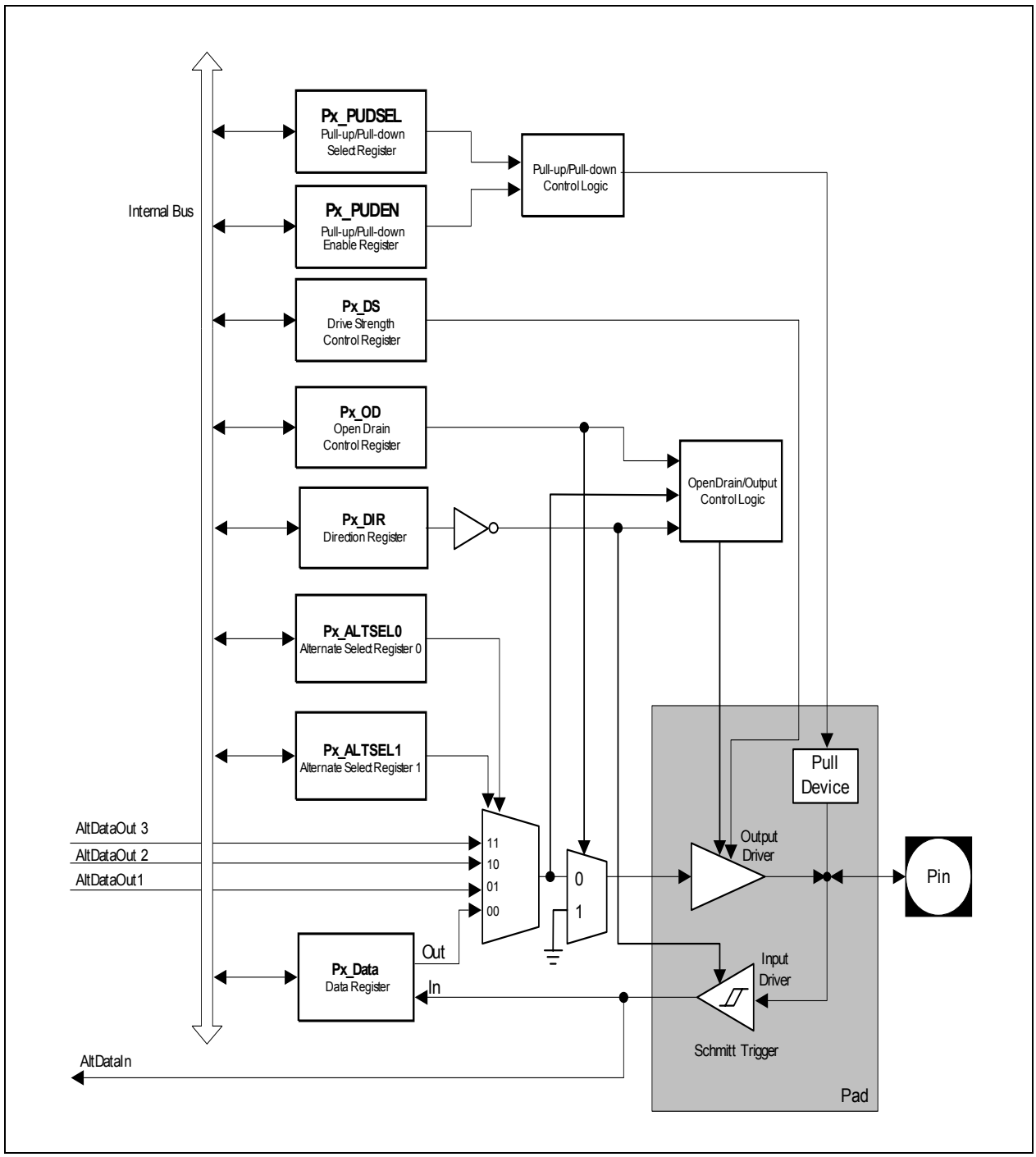



**Figure 6-1 General Structure of Bidirectional Port**

### 6.1.1 General Register Description

The individual control and data bits of each parallel port are implemented in a number of 8-bit registers. Bits with the same meaning and function are assembled together in the same register. The registers configure and use the port as general purpose I/O or alternate function input/output.

The availability and definition of registers specific to each port is defined in **Section 6.3** to **Section 6.7**. This section provides only an overview of the different port registers

**Table 6-1 Port Registers**

| Register Short Name | Register Long Name | Description |
|---|---|---|
| Px_DATA | Port x Data Register | **Page 6-5** |
| Px_DIR | Port x Direction Register | **Page 6-6** |
| Px_OD | Port x Open Drain Control Register | **Page 6-6** |
| Px_PUDSEL | Port x Pull-Up/Pull-Down Select Register | **Page 6-7** |
| Px_PUDEN | Port x Pull-Up/Pull-Down Enable Register | **Page 6-7** |
| Px_ALTSEL0 | Port x Alternate Select Register 0 | **Page 6-8** |
| Px_ALTSEL1 | Port x Alternate Select Register 1 | **Page 6-8** |
| Px_DS | Port x Drive Strength Control Register | **Page 6-6** |

### 6.1.1.1 Data Register

If a port pin is used as general purpose output, output data is written into data register Px_DATA. If a port pin is used as general purpose input, the latched value of the port pin can be read through register Px_DATA.

*Note:* A port pin that has been assigned as input will latch in the active internal pull-up/pull-down setting if it is not driven by an external source. This results in register Px_DATA being updated with the active pull value.

**Px_DATA**
**Port x Data Register**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **P7** | **P6** | **P5** | **P4** | **P3** | **P2** | **P1** | **P0** |
| rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **Pn**<br>**(n = 0 - 7)** | n | rwh | **Port x Pin n Data Value**<br>0 Port x pin n data value = 0<br>1 Port x pin n data value = 1 |

Bit Px_DATA.n can only be written if the corresponding pin is set to output (Px_DIR.n = 1) and cannot be written if the corresponding pin is set to input (Px_DIR.n = 0). The content of Px_DATA.n is output on the assigned pin if the pin is assigned as GPIO pin and the direction is switched/set to output. A read operation of Px_DATA returns the register value and not the state of the corresponding Px_DATA pin.

### 6.1.1.2 Direction Register

The direction of bidirectional port pins is controlled by the respective direction register Px_DIR.

**Px_DIR**
**Port x Direction Register**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **P7** | **P6** | **P5** | **P4** | **P3** | **P2** | **P1** | **P0** |
| rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **Pn**<br>**(n = 0 - 7)** | n | rw | **Bidirectional: Port x Pin n Direction Control**<br>0 Direction is set to input<br>1 Direction is set to output |

### 6.1.1.3 Open Drain Control Register

Each pin in output mode can be emulated to open drain mode. If driven with 1, no driver will be activated and the pin output state depends on the internal pull-up/pull-down device setting or stay at tristate mode. If driven with 0, the driver's pull-down transistor will be activated.

The open drain mode is controlled by the register Px_OD.

**Px_OD**
**Port x Open Drain Control Register**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **P7** | **P6** | **P5** | **P4** | **P3** | **P2** | **P1** | **P0** |
| rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **Pn**<br>**(n = 0 - 7)** | n | rw | **Port x Pin n Open Drain Mode**<br>0 Normal Mode; output is actively driven for 0 and 1 state<br>1 Open Drain Mode; output is actively driven only for 0 state |

### 6.1.1.4 Pull-Up/Pull-Down Device Register

Internal pull-up/pull-down devices can be optionally applied to a port pin.

This offers the possibility to configure the following input characteristics:

- tristate
- High-impedance with a weak pull-up device
- High-impedance with a weak pull-down device

and the following output characteristics:

- Push/pull (optional pull-up/pull-down)
- Open drain with internal pull-up
- Open drain with external pull-up

The pull-up/pull-down device can be fixed or controlled via the registers Px_PUDSEL and Px_PUDEN. Register Px_PUDSEL selects the type of pull-up/pull-down device, while register Px_PUDEN enables or disables it. The pull-up/pull-down device can be selected pinwise.

**Px_PUDSEL**
**Port x Pull-Up/Pull-Down Select Register**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **P7** | **P6** | **P5** | **P4** | **P3** | **P2** | **P1** | **P0** |
| rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **Pn**<br>**(n = 0 - 7)** | n | rw | **Pull-Up/Pull-Down Select Port x Bit n**<br>0 Pull-down device is selected<br>1 Pull-up device is selected |

**Px_PUDEN**
**Port x Pull-Up/Pull-Down Enable Register**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **P7** | **P6** | **P5** | **P4** | **P3** | **P2** | **P1** | **P0** |
| rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **Pn (n = 0 - 7)** | n | rw | **Pull-Up/Pull-Down Enable at Port x Bit n**<br>0 Pull-up or Pull-down device is disabled<br>1 Pull-up or Pull-down device is enabled |

### 6.1.1.5 Alternate Input and Output Functions

The number of alternate input functions that uses a pin for input is not limited. Each port control logic of an I/O pin provides several input paths of digital input value via register or direct digital input value.

Alternate output functions are selected via an output multiplexer which can select up to four output lines.

This multiplexer can be controlled by the following registers:

- Register Px_ALTSEL0
- Register Px_ALTSEL1

Selection of alternate functions is defined in registers Px_ALTSEL0 and Px_ALTSEL1.

**Px_ALTSELn(n = 0 - 1)**
**Port x Alternate Select Register**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **P7** | **P6** | **P5** | **P4** | **P3** | **P2** | **P1** | **P0** |
| rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **Pn (n = 0 - 7)** | n | rw | **Pin Output Functions**<br>Configuration of Px_ALTSEL0.Pn and Px_ALTSEL1.Pn for GPIO or alternate settings:<br>00 Normal GPIO<br>10 Alternate Select 1<br>01 Alternate Select 2<br>11 Alternate Select 3 |

*Note: Set Px_ALTSEL0.Pn and Px_ALTSEL1.Pn to select only implemented alternate output functions.*

### 6.1.1.6 Drive Strength Control Register

The drive strength of the port can be controlled by the register Px_DS. It selects either the strong or weak drive strength.

**Px_DS**
**Port x Drive Strength Control Register**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **P7** | **P6** | **P5** | **P4** | **P3** | **P2** | **P1** | **P0** |
| rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **Pn**<br>**(n = 0 - 7)** | n | rw | **Drive Strength Control Port x Bit n**<br>0 Weak drive strength<br>1 Strong drive strength |

## 6.2 Register Address Map

The Port SFRs are located in the standard memory area (RMAP = 0) and are organized into 4 pages. The PORT_PAGE register is located at address $B2_H$. It contains the page value and page control information.

The addresses of the kernel SFRs are listed in **Table 6-2**

**Table 6-2 SFR Address List for Pages 0-3**

| Address | Page 0 | Page 1 | Page 2 | Page 3 |
|---|---|---|---|---|
| 80H | P0_DATA | P0_PUDSEL | P0_ALTSEL0 | P0_OD |
| 86H | P0_DIR | P0_PUDEN | P0_ALTSEL1 | P0_DS |
| 90H | P1_DATA | P1_PUDSEL | P1_ALTSEL0 | P1_OD |
| 91H | P1_DIR | P1_PUDEN | P1_ALTSEL1 | P1_DS |
| 92H | P5_DATA | P5_PUDSEL | P5_ALTSEL0 | P5_OD |
| 93H | P5_DIR | P5_PUDEN | P5_ALTSEL1 | P5_DS |
| B0H | P3_DATA | P3_PUDSEL | P3_ALTSEL0 | P3_OD |
| B1H | P3_DIR | P3_PUDEN | P3_ALTSEL1 | P3_DS |
| C8H | P4_DATA | P4_PUDSEL | P4_ALTSEL0 | P4_OD |
| C9H | P4_DIR | P4_PUDEN | P4_ALTSEL1 | P4_DS |

**PORT_PAGE**
**Page Register for PORT** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| OP | | STNR | | 0 | PAGE | | |
| w | | w | | r | rwh | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **PAGE** | 2:0 | rwh | **Page Bits**<br>When written, the value indicates the new page address.<br>When read, the value indicates the currently active page = addr [y:x+1] |

| Field | Bits | Type | Description |
|---|---|---|---|
| **STNR** | 5:4 | w | **Storage Number**<br>This number indicates which storage bit field is the target of the operation defined by bit OP.<br>If OP = $10_B$,<br>the contents of PAGE are saved in STx before being overwritten with the new value.<br>If OP = $11_B$,<br>the contents of PAGE are overwritten by the contents of STx. The value written to the bit positions of PAGE is ignored.<br>00 ST0 is selected.<br>01 ST1 is selected.<br>10 ST2 is selected.<br>11 ST3 is selected. |
| **OP** | 7:6 | w | **Operation**<br>0X Manual page mode. The value of STNR is ignored and PAGE is directly written.<br>10 New page programming with automatic page saving. The value written to the bit positions of PAGE is stored. In parallel, the former contents of PAGE are saved in the storage bit field STx indicated by STNR.<br>11 Automatic restore page action. The value written to the bit positions PAGE is ignored and instead, PAGE is overwritten by the contents of the storage bit field STx indicated by STNR. |
| **0** | 3 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

## 6.3 Port 0

Port P0 is a 8-bit general purpose bidirectional port.

The registers of P0 are summarized in **Table 6-3**.

**Table 6-3 Port 0 Registers**

| Register Short Name | Register Long Name |
|---|---|
| P0_DATA | Port 0 Data Register |
| P0_DIR | Port 0 Direction Register |
| P0_OD | Port 0 Open Drain Control Register |
| P0_PUDSEL | Port 0 Pull-Up/Pull-Down Select Register |
| P0_PUDEN | Port 0 Pull-Up/Pull-Down Enable Register |
| P0_ALTSEL0 | Port 0 Alternate Select Register 0 |
| P0_ALTSEL1 | Port 0 Alternate Select Register 1 |
| P0_DS | Port 0 Drive Strength Control Register |

### 6.3.1 Functions

**Table 6-4** shows Port 0 input and output functions.

Alternate output functions are selected via register P0_ALTSEL0 and register P0_ALTSEL1.

**Table 6-4 Port 0 Input/Output Functions**

| Port Pin | Input/Output | Select | Connected Signal(s) | From/to Module |
|---|---|---|---|---|
| P0.0 | Input | GPI | P0_DATA.P0 | – |
| | | ALT1 | TCK_0 | JTAG |
| | | ALT2 | T12HR_1 | CCU6 |
| | | ALT3 | CC61_1 | CCU6 |
| | Output | GPO | P0_DATA.P0 | – |
| | | ALT1 | CLKOUT_0 | Clock Output |
| | | ALT2 | CC61_1 | CCU6 |
| | | ALT3 | RXDO_1 | UART |

**Table 6-4 Port 0 Input/Output Functions**

| Port Pin | Input/Output | Select | Connected Signal(s) | From/to Module |
|---|---|---|---|---|
| P0.1 | Input | GPI | P0_DATA.P1 | – |
| | | ALT1 | TDI_0 | JTAG |
| | | ALT2 | T13HR_1 | CCU6 |
| | | ALT3 | RXD_1 | UART |
| | | ALT4 | RXDC1_0 | MultiCAN |
| | Output | GPO | P0_DATA.P1 | – |
| | | ALT1 | EXF2_1 | T2CCU |
| | | ALT2 | COUT61_1 | CCU6 |
| | | ALT3 | – | – |
| P0.2 | Input | GPI | P0_DATA.P2 | – |
| | | ALT1 | – | – |
| | | ALT2 | $\overline{\text{CTRAP_2}}$ | CCU6 |
| | | ALT3 | – | – |
| | Output | GPO | P0_DATA.P2 | – |
| | | ALT1 | TDO_0 | JTAG |
| | | ALT2 | TXD_1 | UART |
| | | ALT3 | TXDC1_0 | MultiCAN |
| P0.3 | Input | GPI | P0_DATA.P3 | – |
| | | ALT1 | SCK_1 | SSC |
| | | ALT2 | – | – |
| | | ALT3 | – | – |
| | Output | GPO | P0_DATA.P3 | – |
| | | ALT1 | SCK_1 | SSC |
| | | ALT2 | COUT63_1 | CCU6 |
| | | ALT3 | RXDO1_0 | UART1 |

**Table 6-4 Port 0 Input/Output Functions**

| Port Pin | Input/Output | Select | Connected Signal(s) | From/to Module |
|---|---|---|---|---|
| P0.4 | Input | GPI | P0_DATA.P4 | – |
| | | ALT1 | MTSR_1 | SSC |
| | | ALT2 | – | – |
| | | ALT3 | CC62_1 | CCU6 |
| | Output | GPO | P0_DATA.P4 | – |
| | | ALT1 | MTSR_1 | SSC |
| | | ALT2 | CC62_1 | CCU6 |
| | | ALT3 | TXD1_0 | UART1 |
| P0.5 | Input | GPI | P0_DATA.P5 | – |
| | | ALT1 | MRST_1 | SSC |
| | | ALT2 | EXINT0_0 | External interrupt 0 |
| | | ALT3 | T2EX1_1 | Timer 21 |
| | | ALT4 | RXD1_0 | UART1 |
| | Output | GPO | P0_DATA.P5 | – |
| | | ALT1 | MRST_1 | SSC |
| | | ALT2 | COUT62_1 | CCU6 |
| | | ALT3 | – | – |
| P0.6 | Input | GPI | P0_DATA.P6 | – |
| | | ALT1 | – | – |
| | | ALT2 | – | – |
| | | ALT3 | – | – |
| | Output | GPO | P0_DATA.P6 | – |
| | | ALT1 | T2CC4_1 | T2CCU |
| | | ALT2 | – | – |
| | | ALT3 | – | – |

**Table 6-4 Port 0 Input/Output Functions**

| Port Pin | Input/Output | Select | Connected Signal(s) | From/to Module |
|---|---|---|---|---|
| P0.7 | Input | GPI | P0_DATA.P7 | – |
| | | ALT1 | – | – |
| | | ALT2 | – | – |
| | | ALT3 | – | – |
| | Output | GPO | P0_DATA.P7 | – |
| | | ALT1 | CLKOUT_1 | SCU |
| | | ALT2 | T2CC5_1 | T2CCU |
| | | ALT3 | – | – |

## 6.3.2 Register Description

**P0_DATA**
**Port 0 Data Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **P7** | **P6** | **P5** | **P4** | **P3** | **P2** | **P1** | **P0** |
| rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **Pn**<br>**(n = 0 - 7)** | n | rwh | **Port 0 Pin n Data Value**<br>0 Port 0 pin n data value = 0 (default)<br>1 Port 0 pin n data value = 1 |

**P0_DIR**
**Port 0 Direction Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **P7** | **P6** | **P5** | **P4** | **P3** | **P2** | **P1** | **P0** |
| rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **Pn**<br>**(n = 0 - 7)** | n | rw | **Port 0 Pin n Direction Control**<br>0 Direction is set to input (default).<br>1 Direction is set to output |

**P0_OD**
**Port 0 Open Drain Control Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **P7** | **P6** | **P5** | **P4** | **P3** | **P2** | **P1** | **P0** |
| rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **Pn**<br>**(n = 0 - 7)** | n | rw | **Port 0 Pin n Open Drain Mode**<br>0 Normal Mode; output is actively driven for 0 and 1 state (default).<br>1 Open Drain Mode; output is actively driven only for 0 state |

**P0_PUDSEL**
**Port 0 Pull-Up/Pull-Down Select Register** **Reset Value: $FF_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **P7** | **P6** | **P5** | **P4** | **P3** | **P2** | **P1** | **P0** |
| rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **Pn**<br>**(n = 0 - 7)** | n | rw | **Pull-Up/Pull-Down Select Port 0 Bit n**<br>0 Pull-down device is selected<br>1 Pull-up device is selected (default). |

**P0_PUDEN**
**Port 0 Pull-Up/Pull-Down Enable Register** **Reset Value: $C4_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **P7** | **P6** | **P5** | **P4** | **P3** | **P2** | **P1** | **P0** |
| rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **Pn**<br>**(n = 0 - 7)** | n | rw | **Pull-Up/Pull-Down Enable at Port 0 Bit n**<br>0 Pull-up or Pull-down device is disabled<br>1 Pull-up or Pull-down device is enabled |

**P0_ALTSELn(n = 0 - 1)**
**Port 0 Alternate Select Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **P7** | **P6** | **P5** | **P4** | **P3** | **P2** | **P1** | **P0** |
| rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **Pn**<br>**(n = 0 - 7)** | n | rw | **Pin Output Functions**<br>Configuration of P0_ALTSEL0.Pn and P0_ALTSEL1.Pn for GPIO or alternate settings:<br>00 Normal GPIO (default).<br>10 Alternate Select 1<br>01 Alternate Select 2<br>11 Alternate Select 3 |

**P0_DS**
**Port 0 Drive Strength Control Register** **Reset Value: $FF_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **P7** | **P6** | **P5** | **P4** | **P3** | **P2** | **P1** | **P0** |
| rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **Pn**<br>**(n = 0 - 7)** | n | rw | **Drive Strength Control Port 0 Bit n**<br>0 Weak drive strength<br>1 Strong drive strength (default). |

## 6.4 Port 1

Port P1 is a 8-bit general purpose bidirectional port.

The registers of P1 are summarized in **Table 6-5**.

**Table 6-5 Port 1 Registers**

| Register Short Name | Register Long Name |
|---|---|
| P1_DATA | Port 1 Data Register |
| P1_DIR | Port 1 Direction Register |
| P1_OD | Port 1 Open Drain Control Register |
| P1_PUDSEL | Port 1 Pull-Up/Pull-Down Select Register |
| P1_PUDEN | Port 1 Pull-Up/Pull-Down Enable Register |
| P1_ALTSEL0 | Port 1 Alternate Select Register 0 |
| P1_ALTSEL1 | Port 1 Alternate Select Register 1 |
| P1_DS | Port 1 Drive Strength Control Register |

### 6.4.1 Functions

**Table 6-6** shows Port 1 input and output functions.

Alternate output functions are selected via register P1_ALTSEL0 and register P1_ALTSEL1.

**Table 6-6 Port 1 Input/Output Functions**

| Port Pin | Input/Output | Select | Connected Signal(s) | From/to Module |
|---|---|---|---|---|
| P1.0 | Input | GPI | P1_DATA.P0 | – |
| | | ALT1 | RXD_0 | UART |
| | | ALT2 | T2EX_0 | T2CCU |
| | | ALT3 | RXDC0_0 | MultiCAN |
| | Output | GPO | P1_DATA.P0 | – |
| | | ALT1 | – | – |
| | | ALT2 | – | – |
| | | ALT3 | – | – |

**Table 6-6 Port 1 Input/Output Functions**

| Port Pin | Input/Output | Select | Connected Signal(s) | From/to Module |
|---|---|---|---|---|
| P1.1 | Input | GPI | P1_DATA.P1 | – |
| | | ALT1 | – | – |
| | | ALT2 | EXINT3_0 | External interrupt 3 |
| | | ALT3 | T0_1 | Timer 0 |
| | Output | GPO | P1_DATA.P1 | – |
| | | ALT1 | – | – |
| | | ALT2 | TXD_0 | UART |
| | | ALT3 | TXDC0_0 | MultiCAN |
| P1.2 | Input | GPI | P1_DATA.P2 | – |
| | | ALT1 | SCK_0 | SSC |
| | | ALT2 | – | – |
| | | ALT3 | – | – |
| | Output | GPO | P1_DATA.P2 | – |
| | | ALT1 | SCK_0 | SSC |
| | | ALT2 | – | – |
| | | ALT3 | – | – |
| P1.3 | Input | GPI | P1_DATA.P3 | – |
| | | ALT1 | MTSR_0 | SSC |
| | | ALT2 | SCK_2 | SSC |
| | | ALT3 | – | – |
| | Output | GPO | P1_DATA.P3 | – |
| | | ALT1 | MTSR_0 | SSC |
| | | ALT2 | SCK_2 | SSC |
| | | ALT3 | TXDC1_3 | MultiCAN |

**Table 6-6 Port 1 Input/Output Functions**

| Port Pin | Input/Output | Select | Connected Signal(s) | From/to Module |
|---|---|---|---|---|
| P1.4 | Input | GPI | P1_DATA.P4 | – |
| | | ALT1 | MRST_0 | SSC |
| | | ALT2 | EXINT0_1 | External interrupt 0 |
| | | ALT3 | RXDC1_3 | MultiCAN |
| | | ALT4 | MTSR_2 | SSC |
| | Output | GPO | P1_DATA.P4 | – |
| | | ALT1 | MRST_0 | SSC |
| | | ALT2 | MTSR_2 | SSC |
| | | ALT3 | – | – |
| P1.5 | Input | GPI | P1_DATA.P5 | – |
| | | ALT1 | CCPOS0_1 | CCU6 |
| | | ALT2 | EXINT5_0 | External interrupt 5 |
| | | ALT3 | T1_1 | Timer 1 |
| | | ALT4 | MRST_2 | SSC |
| | Output | GPO | P1_DATA.P5[1)] | – |
| | | ALT1 | EXF2_0 | T2CCU |
| | | ALT2 | RXDO_0 | UART |
| | | ALT3 | MRST_2 | SSC |
| P1.6 | Input | GPI | P1_DATA.P6 | – |
| | | ALT1 | CCPOS1_1 | CCU6 |
| | | ALT2 | T12HR_0 | CCU6 |
| | | ALT3 | EXINT6_0 | External interrupt 6 |
| | | ALT4 | RXDC0_2 | MultiCAN |
| | | ALT5 | T21_1 | Timer 21 |
| | Output | GPO | P1_DATA.P6[2)] | – |
| | | ALT1 | – | – |
| | | ALT2 | – | – |

**Table 6-6 Port 1 Input/Output Functions**

| Port Pin | Input/Output | Select | Connected Signal(s) | From/to Module |
|---|---|---|---|---|
| P1.7 | Input | GPI | P1_DATA.P7 | – |
| | | ALT1 | CCPOS2_1 | CCU6 |
| | | ALT2 | T13HR_0 | CCU6 |
| | | ALT3 | T2_1 | T2CCU |
| | Output | GPO | P1_DATA.P7 | – |
| | | ALT1 | – | – |
| | | ALT2 | – | – |
| | | ALT3 | TXDC0_2 | MultiCAN |

1) P1.5 can be used as a software Chip Select function for the SSC.

2) P1.6 can be used as a software Chip Select function for the SSC.

## 6.4.2 Register Description

**P1_DATA**
**Port 1 Data Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **P7** | **P6** | **P5** | **P4** | **P3** | **P2** | **P1** | **P0** |
| rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **Pn**<br>**(n = 0 - 7)** | n | rwh | **Port 1 Pin n Data Value**<br>0 Port 1 pin n data value = 0 (default)<br>1 Port 1 pin n data value = 1 |

**P1_DIR**
**Port 1 Direction Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **P7** | **P6** | **P5** | **P4** | **P3** | **P2** | **P1** | **P0** |
| rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **Pn**<br>**(n = 0 - 7)** | n | rw | **Port 1 Pin n Direction Control**<br>0 Direction is set to input (default)<br>1 Direction is set to output |

**P1_OD**
**Port 1 Open Drain Control Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **P7** | **P6** | **P5** | **P4** | **P3** | **P2** | **P1** | **P0** |
| rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **Pn**<br>**(n = 0 - 7)** | n | rw | **Port 1 Pin n Open Drain Mode**<br>0 Normal Mode; output is actively driven for 0 and 1 state (default)<br>1 Open Drain Mode; output is actively driven only for 0 state |

**P1_PUDSEL**
**Port 1 Pull-Up/Pull-Down Select Register** **Reset Value: $FF_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **P7** | **P6** | **P5** | **P4** | **P3** | **P2** | **P1** | **P0** |
| rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **Pn**<br>**(n = 0 - 7)** | n | rw | **Pull-Up/Pull-Down Select Port 1 Bit n**<br>0 Pull-down device is selected<br>1 Pull-up device is selected (default) |

**P1_PUDEN**
**Port 1 Pull-Up/Pull-Down Enable Register** **Reset Value: $FF_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **P7** | **P6** | **P5** | **P4** | **P3** | **P2** | **P1** | **P0** |
| rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **Pn**<br>**(n = 0 - 7)** | n | rw | **Pull-Up/Pull-Down Enable at Port 1 Bit n**<br>0 Pull-up or Pull-down device is disabled<br>1 Pull-up or Pull-down device is enabled(default) |

**P1_ALTSELn(n = 0 - 1)**
**Port 1 Alternate Select Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **P7** | **P6** | **P5** | **P4** | **P3** | **P2** | **P1** | **P0** |
| rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **Pn**<br>**(n = 0 - 7)** | n | rw | **Pin Output Functions**<br>Configuration of P1_ALTSEL0.Pn and P1_ALTSEL1.Pn for GPIO or alternate settings:<br>00 Normal GPIO (default)<br>10 Alternate Select 1<br>01 Alternate Select 2<br>11 Alternate Select 3 |

## P1_DS
## Port 1 Drive Strength Control Register

**Reset Value: $FF_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **P7** | **P6** | **P5** | **P4** | **P3** | **P2** | **P1** | **P0** |
| rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **Pn**<br>**(n = 0 - 7)** | n | rw | **Port 1 Bit n Drive Strength**<br>0 Weak drive strength<br>1 Strong drive strength (default) |

## 6.5 Port 3

Port P3 is a 8-bit general purpose bidirectional port.

The registers of P3 are summarized in **Table 6-7**.

**Table 6-7 Port 3 Registers**

| Register Short Name | Register Long Name |
|---|---|
| P3_DATA | Port 3 Data Register |
| P3_DIR | Port 3 Direction Register |
| P3_OD | Port 3 Open Drain Control Register |
| P3_PUDSEL | Port 3 Pull-Up/Pull-Down Select Register |
| P3_PUDEN | Port 3 Pull-Up/Pull-Down Enable Register |
| P3_ALTSEL0 | Port 3 Alternate Select Register 0 |
| P3_ALTSEL1 | Port 3 Alternate Select Register 1 |
| P3_DS | Port 3 Drive Strength Control Register |

### 6.5.1 Functions

**Table 6-8** shows Port 3 input and output functions.

Alternate output functions are selected via register P3_ALTSEL0 and register P3_ALTSEL1.

**Table 6-8 Port 3 Input/Output Functions**

| Port Pin | Input/Output | Select | Connected Signal(s) | From/to Module |
|---|---|---|---|---|
| P3.0 | Input | GPI | P3_DATA.P0 | – |
| | | ALT1 | CC60_0 | CCU6 |
| | | ALT2 | CCPOS1_2 | CCU6 |
| | | ALT3 | T2CC0_1/EXINT3_2 | T2CCU/<br>External Interrupt 3 |
| | Output | GPO | P3_DATA.P0 | – |
| | | ALT1 | CC60_0 | CCU6 |
| | | ALT2 | T2CC0_1 | T2CCU |
| | | ALT3 | RXDO1_1 | UART1 |

**Table 6-8 Port 3 Input/Output Functions**

| Port Pin | Input/Output | Select | Connected Signal(s) | From/to Module |
|---|---|---|---|---|
| P3.1 | Input | GPI | P3_DATA.P1 | – |
| | | ALT1 | – | – |
| | | ALT2 | CCPOS0_2 | CCU6 |
| | | ALT3 | CC61_2 | CCU6 |
| | Output | GPO | P3_DATA.P1 | – |
| | | ALT1 | COUT60_0 | CCU6 |
| | | ALT2 | CC61_2 | CCU6 |
| | | ALT3 | TXD1_1 | UART1 |
| P3.2 | Input | GPI | P3_DATA.P2 | – |
| | | ALT1 | CC61_0 | CCU6 |
| | | ALT2 | CCPOS2_2 | CCU6 |
| | | ALT3 | RXDC1_1 | MultiCAN |
| | | ALT4 | RXD1_1 | UART1 |
| | | ALT5 | T2CC1_1/EXINT4_2 | T2CCU/<br>External Interrupt 4 |
| | Output | GPO | P3_DATA.P2 | – |
| | | ALT1 | CC61_0 | CCU6 |
| | | ALT2 | T2CC1_1 | T2CCU |
| | | ALT3 | – | – |
| P3.3 | Input | GPI | P3_DATA.P3 | – |
| | | ALT1 | T2CC2_1/EXINT5_2 | T2CCU/<br>External Interrupt 5 |
| | | ALT2 | – | – |
| | | ALT3 | – | – |
| | Output | GPO | P3_DATA.P3 | – |
| | | ALT1 | COUT61_0 | CCU6 |
| | | ALT2 | T2CC2_1 | T2CCU |
| | | ALT3 | TXDC1_1 | MultiCAN |

**Table 6-8 Port 3 Input/Output Functions**

| Port Pin | Input/Output | Select | Connected Signal(s) | From/to Module |
|---|---|---|---|---|
| P3.4 | Input | GPI | P3_DATA.P4 | – |
| | | ALT1 | CC62_0 | CCU6 |
| | | ALT2 | T2EX1_0 | Timer 21 |
| | | ALT3 | RXDC0_1 | MultiCAN |
| | | ALT4 | T2CC3_1/EXINT6_3 | T2CCU/<br>External Interrupt 6 |
| | Output | GPO | P3_DATA.P4 | – |
| | | ALT1 | CC62_0 | CCU6 |
| | | ALT2 | T2CC3_1 | T2CCU |
| | | ALT3 | – | – |
| P3.5 | Input | GPI | P3_DATA.P5 | – |
| | | ALT1 | – | – |
| | | ALT2 | – | – |
| | | ALT3 | – | – |
| | Output | GPO | P3_DATA.P5 | – |
| | | ALT1 | COUT62_0 | CCU6 |
| | | ALT2 | EXF21_0 | Timer 21 |
| | | ALT3 | TXDC0_1 | MultiCAN |
| P3.6 | Input | GPI | P3_DATA.P6 | – |
| | | ALT1 | $\overline{\text{CTRAP_0}}$ | CCU6 |
| | | ALT2 | – | – |
| | | ALT3 | – | – |
| | Output | GPO | P3_DATA.P6 | – |
| | | ALT1 | – | – |
| | | ALT2 | – | – |
| | | ALT3 | – | – |

**Table 6-8 Port 3 Input/Output Functions**

| Port Pin | Input/Output | Select | Connected Signal(s) | From/to Module |
|---|---|---|---|---|
| P3.7 | Input | GPI | P3_DATA.P7 | – |
| | | ALT1 | – | – |
| | | ALT2 | EXINT4_0 | External interrupt 4 |
| | | ALT3 | – | – |
| | Output | GPO | P3_DATA.P7 | – |
| | | ALT1 | COUT63_0 | CCU6 |
| | | ALT2 | – | – |
| | | ALT3 | – | – |

## 6.5.2 Register Description

**P3_DATA**
**Port 3 Data Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **P7** | **P6** | **P5** | **P4** | **P3** | **P2** | **P1** | **P0** |
| rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **Pn**<br>**(n = 0 - 7)** | n | rwh | **Port 3 Pin n Data Value**<br>0 Port 3 pin n data value = 0 (default)<br>1 Port 3 pin n data value = 1 |

**P3_DIR**
**Port 3 Direction Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **P7** | **P6** | **P5** | **P4** | **P3** | **P2** | **P1** | **P0** |
| rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **Pn**<br>**(n = 0 - 7)** | n | rw | **Port 3 Pin n Direction Control**<br>0 Direction is set to input (default)<br>1 Direction is set to output |

**P3_OD**
**Port 3 Open Drain Control Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **P7** | **P6** | **P5** | **P4** | **P3** | **P2** | **P1** | **P0** |
| rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **Pn**<br>**(n = 0 - 7)** | n | rw | **Port 3 Pin n Open Drain Mode**<br>0 Normal Mode; output is actively driven for 0 and 1 state (default)<br>1 Open Drain Mode; output is actively driven only for 0 state |

**P3_PUDSEL**
**Port 3 Pull-Up/Pull-Down Select Register** **Reset Value: $BF_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **P7** | **P6** | **P5** | **P4** | **P3** | **P2** | **P1** | **P0** |
| rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **Pn**<br>**(n = 0 - 7)** | n | rw | **Pull-Up/Pull-Down Select Port 3 Bit n**<br>0 Pull-down device is selected<br>1 Pull-up device is selected (default) |

*Note: Pull down device is activated for Pin P3.6 when reset is active. In the BootROM start up procedure, the pull down device is deactivated so that Pin P3.6 becomes tristate.*

### P3_PUDEN
**Port 3 Pull-Up/Pull-Down Enable Register** **Reset Value: $40_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **P7** | **P6** | **P5** | **P4** | **P3** | **P2** | **P1** | **P0** |
| rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **Pn**<br>**(n = 0 - 7)** | n | rw | **Pull-Up/Pull-Down Enable at Port 3 Bit n**<br>0 Pull-up or Pull-down device is disabled<br>1 Pull-up or Pull-down device is enabled(default) |

### P3_ALTSEL0
**Port 3 Alternate Select 0 Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **P7** | **P6** | **P5** | **P4** | **P3** | **P2** | **P1** | **P0** |
| rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **Pn**<br>**(n = 0 - 7)** | n | rw | **Pin Output Functions**<br>Configuration of P3_ALTSEL0.Pn and P3_ALTSEL1.Pn for GPIO or alternate settings:<br>00 Normal GPIO (default)<br>10 Alternate Select 1<br>01 Alternate Select 2<br>11 Alternate Select 3 |

**P3_ALTSEL1**
**Port 3 Alternate Select 1 Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **P7** | **P6** | **P5** | **P4** | **P3** | **P2** | **P1** | **P0** |
| rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **Pn**<br>**(n = 0 - 7)** | n | rw | **Pin Output Functions**<br>Configuration of P3_ALTSEL0.Pn and P3_ALTSEL1.Pn for GPIO or alternate settings:<br>00 Normal GPIO (default)<br>10 Alternate Select 1<br>01 Alternate Select 2<br>11 Alternate Select 3 |

**P3_DS**
**Port 3 Drive Strength Control Register** **Reset Value: $FF_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **P7** | **P6** | **P5** | **P4** | **P3** | **P2** | **P1** | **P0** |
| rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **Pn**<br>**(n = 0 - 7)** | n | rw | **Port 3 Bit n Drive Strength**<br>0 Weak drive strength<br>1 Strong drive strength (default) |

## 6.6 Port 4

Port P4 is a 8-bit general purpose bidirectional port.

The registers of P4 are summarized in **Table 6-9**.

**Table 6-9 Port 4 Registers**

| Register Short Name | Register Long Name |
|---|---|
| P4_DATA | Port 4 Data Register |
| P4_DIR | Port 4 Direction Register |
| P4_OD | Port 4 Open Drain Control Register |
| P4_PUDSEL | Port 4 Pull-Up/Pull-Down Select Register |
| P4_PUDEN | Port 4 Pull-Up/Pull-Down Enable Register |
| P4_ALTSEL0 | Port 4 Alternate Select Register 0 |
| P4_ALTSEL1 | Port 4 Alternate Select Register 1 |
| P4_DS | Port 4 Drive Strength Control Register |

### 6.6.1 Functions

**Table 6-10** shows Port 4 input and output functions.

Alternate output functions are selected via register P4_ALTSEL0 and register P4_ALTSEL1.

**Table 6-10 Port 4 Input/Output Functions**

| Port Pin | Input/Output | Select | Connected Signal(s) | From/to Module |
|---|---|---|---|---|
| P4.0 | Input | GPI | P4_DATA.P0 | – |
| | | ALT1 | – | – |
| | | ALT2 | – | – |
| | | ALT3 | RXDC0_3 | MultiCAN |
| | | ALT4 | T2CC0_0/EXINT3_1 | T2CCU/<br>External Interrupt 3 |
| | Output | GPO | P4_DATA.P0 | – |
| | | ALT1 | CC60_1 | CCU6 |
| | | ALT2 | T2CC0_0 | T2CCU |
| | | ALT3 | – | – |

**Table 6-10 Port 4 Input/Output Functions**

| Port Pin | Input/Output | Select | Connected Signal(s) | From/to Module |
|---|---|---|---|---|
| P4.1 | Input | GPI | P4_DATA.P1 | – |
| | | ALT1 | T2CC1_0/EXINT4_1 | T2CCU/<br>External Interrupt 4 |
| | | ALT2 | – | – |
| | | ALT3 | – | – |
| | Output | GPO | P4_DATA.P1 | – |
| | | ALT1 | COUT60_1 | CCU6 |
| | | ALT2 | T2CC1_0 | T2CCU |
| | | ALT3 | TXDC0_3 | MultiCAN |
| P4.2 | Input | GPI | P4_DATA.P2 | – |
| | | ALT1 | T21_0 | Timer 21 |
| | | ALT2 | EXINT6_1 | External Interrupt 6 |
| | | ALT3 | – | – |
| | Output | GPO | P4_DATA.P2 | – |
| | | ALT1 | – | – |
| | | ALT2 | – | – |
| | | ALT3 | – | – |
| P4.3 | Input | GPI | P4_DATA.P3 | – |
| | | ALT1 | T2EX_1 | T2CCU |
| | | ALT2 | – | – |
| | | ALT3 | – | – |
| | Output | GPO | P4_DATA.P3 | – |
| | | ALT1 | EXF21_1 | Timer 21 |
| | | ALT2 | COUT63_2 | CCU6 |
| | | ALT3 | – | – |

**Table 6-10 Port 4 Input/Output Functions**

| Port Pin | Input/Output | Select | Connected Signal(s) | From/to Module |
|---|---|---|---|---|
| P4.4 | Input | GPI | P4_DATA.P4 | – |
| | | ALT1 | CCPOS0_3 | CCU6 |
| | | ALT2 | T0_0 | Timer 0 |
| | | ALT3 | T2CC2_0/EXINT5_1 | T2CCU/<br>External Interrupt 5 |
| | Output | GPO | P4_DATA.P4 | – |
| | | ALT1 | CC61_4 | CCU6 |
| | | ALT2 | T2CC2_0 | T2CCU |
| | | ALT3 | – | – |
| P4.5 | Input | GPI | P4_DATA.P5 | – |
| | | ALT1 | CCPOS1_3 | CCU6 |
| | | ALT2 | T1_0 | Timer 1 |
| | | ALT3 | T2CC3_0/EXINT6_2 | T2CCU/<br>External Interrupt 6 |
| | Output | GPO | P4_DATA.P5 | – |
| | | ALT1 | COUT61_2 | CCU6 |
| | | ALT2 | T2CC3_0 | T2CCU |
| | | ALT3 | – | – |
| P4.6 | Input | GPI | P4_DATA.P6 | – |
| | | ALT1 | CCPOS2_3 | CCU6 |
| | | ALT2 | T2_0 | Timer 2 |
| | | ALT3 | – | – |
| | Output | GPO | P4_DATA.P6 | – |
| | | ALT1 | CC62_2 | CCU6 |
| | | ALT2 | T2CC4_0 | T2CCU |
| | | ALT3 | – | – |

**Table 6-10 Port 4 Input/Output Functions**

| Port Pin | Input/Output | Select | Connected Signal(s) | From/to Module |
|---|---|---|---|---|
| P4.7 | Input | GPI | P4_DATA.P7 | – |
| | | ALT1 | $\overline{\text{CTRAP_3}}$ | CCU6 |
| | | ALT2 | – | – |
| | | ALT3 | – | – |
| | Output | GPO | P4_DATA.P7 | – |
| | | ALT1 | COUT62_2 | CCU6 |
| | | ALT2 | T2CC5_0 | T2CCU |
| | | ALT3 | – | – |

## 6.6.2 Register Description

**P4_DATA**
**Port 4 Data Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **P7** | **P6** | **P5** | **P4** | **P3** | **P2** | **P1** | **P0** |
| rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **Pn**<br>**(n = 0 - 7)** | n | rwh | **Port 4 Pin n Data Value**<br>0 Port 4 pin n data value = 0 (default)<br>1 Port 4 pin n data value = 1 |

**P4_DIR**
**Port 4 Direction Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **P7** | **P6** | **P5** | **P4** | **P3** | **P2** | **P1** | **P0** |
| rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **Pn**<br>**(n = 0 - 7)** | n | rw | **Port 4 Pin n Direction Control**<br>0 Direction is set to input (default)<br>1 Direction is set to output |

**P4_OD**
**Port 4 Open Drain Control Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **P7** | **P6** | **P5** | **P4** | **P3** | **P2** | **P1** | **P0** |
| rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **Pn**<br>**(n = 0 - 7)** | n | rw | **Port 4 Pin n Open Drain Mode**<br>0 Normal Mode; output is actively driven for 0 and 1 state (default)<br>1 Open Drain Mode; output is actively driven only for 0 state |

**P4_PUDSEL**
**Port 4 Pull-Up/Pull-Down Select Register** **Reset Value: $FF_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **P7** | **P6** | **P5** | **P4** | **P3** | **P2** | **P1** | **P0** |
| rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **Pn**<br>**(n = 0 - 7)** | n | rw | **Pull-Up/Pull-Down Select Port 4 Bit n**<br>0 Pull-down device is selected<br>1 Pull-up device is selected (default) |

**P4_PUDEN**
**Port 4 Pull-Up/Pull-Down Enable Register** **Reset Value: $04_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **P7** | **P6** | **P5** | **P4** | **P3** | **P2** | **P1** | **P0** |
| rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **Pn**<br>**(n = 0 - 7)** | n | rw | **Pull-Up/Pull-Down Enable at Port 4 Bit n**<br>0 Pull-up or Pull-down device is disabled<br>1 Pull-up or Pull-down device is enabled(default) |

**P4_ALTSELn(n = 0 - 1)**
**Port 4 Alternate Select Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **P7** | **P6** | **P5** | **P4** | **P3** | **P2** | **P1** | **P0** |
| rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **Pn**<br>**(n = 0 - 7)** | n | rw | **Pin Output Functions**<br>Configuration of P4_ALTSEL0.Pn and P4_ALTSEL1.Pn for GPIO or alternate settings:<br>00 Normal GPIO (default)<br>10 Alternate Select 1<br>01 Alternate Select 2<br>11 Alternate Select 3 |

## P4_DS
## Port 4 Drive Strength Control Register

**Reset Value: $FF_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **P7** | **P6** | **P5** | **P4** | **P3** | **P2** | **P1** | **P0** |
| rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **Pn**<br>**(n = 0 - 7)** | n | rw | **Port 4 Bit n Drive Strength**<br>0 Weak drive strength<br>1 Strong drive strength (default) |

## 6.7 Port 5

Port P5 is a 8-bit general purpose bidirectional port.

The registers of P5 are summarized in **Table 6-11**.

**Table 6-11 Port 5 Registers**

| Register Short Name | Register Long Name |
|---|---|
| P5_DATA | Port 5 Data Register |
| P5_DIR | Port 5 Direction Register |
| P5_OD | Port 5 Open Drain Control Register |
| P5_PUDSEL | Port 5 Pull-Up/Pull-Down Select Register |
| P5_PUDEN | Port 5 Pull-Up/Pull-Down Enable Register |
| P5_ALTSEL0 | Port 5 Alternate Select Register 0 |
| P5_ALTSEL1 | Port 5 Alternate Select Register 1 |
| P5_DS | Port 5 Drive Strength Control Register |

### 6.7.1 Functions

**Table 6-12** shows Port 5 input and output functions.

Alternate output functions are selected via register P5_ALTSEL0 and register P5_ALTSEL1.

**Table 6-12 Port 5 Input/Output Functions**

| Port Pin | Input/Output | Select | Connected Signal(s) | From/to Module |
|---|---|---|---|---|
| P5.0 | Input | GPI | P5_DATA.P0 | – |
| | | ALT1 | – | – |
| | | ALT2 | EXINT1_1 | External Interrupt 1 |
| | | ALT3 | – | – |
| | Output | GPO | P5_DATA.P0 | – |
| | | ALT1 | – | – |
| | | ALT2 | – | – |
| | | ALT3 | – | – |

**Table 6-12 Port 5 Input/Output Functions**

| Port Pin | Input/Output | Select | Connected Signal(s) | From/to Module |
|---|---|---|---|---|
| P5.1 | Input | GPI | P5_DATA.P1 | – |
| | | ALT1 | – | – |
| | | ALT2 | EXINT2_1 | External Interrupt 2 |
| | | ALT3 | – | – |
| | Output | GPO | P5_DATA.P1 | – |
| | | ALT1 | – | – |
| | | ALT2 | – | – |
| | | ALT3 | – | – |
| P5.2 | Input | GPI | P5_DATA.P2 | – |
| | | ALT1 | RXD_2 | UART |
| | | ALT2 | T2CC2_2/EXINT5_3 | T2CCU/<br>External Interrupt 5 |
| | | ALT3 | – | – |
| | Output | GPO | P5_DATA.P2 | – |
| | | ALT1 | T2CC2_2 | T2CCU |
| | | ALT2 | – | – |
| | | ALT3 | – | – |
| P5.3 | Input | GPI | P5_DATA.P3 | – |
| | | ALT1 | CCPOS0_0 | CCU6 |
| | | ALT2 | EXINT1_0 | External Interrupt 1 |
| | | ALT3 | T12HR_2 | CCU6 |
| | | ALT4 | CC61_3 | CCU6 |
| | Output | GPO | P5_DATA.P3 | – |
| | | ALT1 | T2CC5_2 | T2CCU |
| | | ALT2 | TXD_2 | UART |
| | | ALT3 | – | – |

**Table 6-12 Port 5 Input/Output Functions**

| Port Pin | Input/Output | Select | Connected Signal(s) | From/to Module |
|---|---|---|---|---|
| P5.4 | Input | GPI | P5_DATA.P4 | – |
| | | ALT1 | CCPOS1_0 | CCU6 |
| | | ALT2 | EXINT2_0 | External Interrupt 2 |
| | | ALT3 | T13HR_2 | CCU6 |
| | | ALT4 | CC62_3 | CCU6 |
| | Output | GPO | P5_DATA.P4 | – |
| | | ALT1 | T2CC4_2 | T2CCU |
| | | ALT2 | RXDO_2 | UART |
| | | ALT3 | – | – |
| P5.5 | Input | GPI | P5_DATA.P5 | – |
| | | ALT1 | CCPOS2_0 | CCU6 |
| | | ALT2 | T2CC0_2/EXINT3_3 | T2CCU/<br>External Interrupt 3 |
| | | ALT3 | $\overline{\text{CTRAP_1}}$ | CCU6 |
| | | ALT4 | CC60_3 | CCU6 |
| | Output | GPO | P5_DATA.P5 | – |
| | | ALT1 | TDO_1 | JTAG |
| | | ALT2 | TXD1_2 | UART1 |
| | | ALT3 | T2CC0_2 | T2CCU |
| P5.6 | Input | GPI | P5_DATA.P6 | – |
| | | ALT1 | TCK_1 | JTAG |
| | | ALT2 | T2CC1_2/EXINT4_3 | T2CCU/<br>External Interrupt 4 |
| | | ALT3 | – | – |
| | Output | GPO | P5_DATA.P6 | – |
| | | ALT1 | – | – |
| | | ALT2 | RXDO1_2 | UART1 |
| | | ALT3 | T2CC1_2 | T2CCU |

**Table 6-12 Port 5 Input/Output Functions**

| Port Pin | Input/Output | Select | Connected Signal(s) | From/to Module |
|---|---|---|---|---|
| P5.7 | Input | GPI | P5_DATA.P7 | – |
| | | ALT1 | TDI_1 | JTAG |
| | | ALT2 | RXD1_2 | UART1 |
| | | ALT3 | T2CC3_2/EXINT6_4 | T2CCU/<br>External Interrupt 6 |
| | Output | GPO | P5_DATA.P7 | – |
| | | ALT1 | T2CC3_2 | T2CCU |
| | | ALT2 | – | – |
| | | ALT3 | – | – |

## 6.7.2 Register Description

**P5_DATA**
**Port 5 Data Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **P7** | **P6** | **P5** | **P4** | **P3** | **P2** | **P1** | **P0** |
| rwh | rwh | rwh | rwh | rwh | rwh | rwh | rwh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **Pn**<br>**(n = 0 - 7)** | n | rwh | **Port 5 Pin n Data Value**<br>0 Port 5 pin n data value = 0 (default)<br>1 Port 5 pin n data value = 1 |

**P5_DIR**
**Port 5 Direction Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **P7** | **P6** | **P5** | **P4** | **P3** | **P2** | **P1** | **P0** |
| rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **Pn**<br>**(n = 0 - 7)** | n | rw | **Port 5 Pin n Direction Control**<br>0 Direction is set to input (default)<br>1 Direction is set to output |

**P5_OD**
**Port 5 Open Drain Control Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **P7** | **P6** | **P5** | **P4** | **P3** | **P2** | **P1** | **P0** |
| rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **Pn**<br>**(n = 0 - 7)** | n | rw | **Port 5 Pin n Open Drain Mode**<br>0 Normal Mode; output is actively driven for 0 and 1 state (default)<br>1 Open Drain Mode; output is actively driven only for 0 state |

**P5_PUDSEL**
**Port 5 Pull-Up/Pull-Down Select Register** **Reset Value: $FF_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **P7** | **P6** | **P5** | **P4** | **P3** | **P2** | **P1** | **P0** |
| rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **Pn**<br>**(n = 0 - 7)** | n | rw | **Pull-Up/Pull-Down Select Port 5 Bit n**<br>0 Pull-down device is selected<br>1 Pull-up device is selected (default) |

**P5_PUDEN**
**Port 5 Pull-Up/Pull-Down Enable Register** **Reset Value: $FF_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **P7** | **P6** | **P5** | **P4** | **P3** | **P2** | **P1** | **P0** |
| rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **Pn**<br>**(n = 0 - 7)** | n | rw | **Pull-Up/Pull-Down Enable at Port 5 Bit n**<br>0 Pull-up or Pull-down device is disabled<br>1 Pull-up or Pull-down device is enabled(default) |

**P5_ALTSELn(n = 0 - 1)**
**Port 5 Alternate Select Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **P7** | **P6** | **P5** | **P4** | **P3** | **P2** | **P1** | **P0** |
| rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **Pn**<br>**(n = 0 - 7)** | n | rw | **Pin Output Functions**<br>Configuration of P5_ALTSEL0.Pn and P5_ALTSEL1.Pn for GPIO or alternate settings:<br>00 Normal GPIO (default)<br>10 Alternate Select 1<br>01 Alternate Select 2<br>11 Alternate Select 3 |

**P5_DS**
**Port 5 Drive Strength Control Register** **Reset Value: $FF_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **P7** | **P6** | **P5** | **P4** | **P3** | **P2** | **P1** | **P0** |
| rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **Pn**<br>**(n = 0 - 7)** | n | rw | **Port 5 Bit n Drive Strength**<br>0 Weak drive strength<br>1 Strong drive strength (default) |

# 7 Power Supply, Reset and Clock Management

The XC878 provides a range of utility features for secure system performance under critical conditions (e.g., brownout).

The power supply to the core, memories and the peripherals is regulated by the Embedded Voltage Regulator (EVR) that comes with detection circuitries to ensure that the supplied voltages are within the specified operating range. The main voltage and low power voltage regulators in the EVR may be independently switched off to reduce power consumption for the different power saving modes.

At the center of the XC878 clock system is the Clock Generation Unit (CGU), which generates a master clock frequency using the Phase-Locked Loop (PLL) and oscillator units. In-phase synchronized clock signals are derived from the master clock and distributed throughout the system. A programmable clock divider is available for scaling the master clock into lower frequencies for power savings.

## 7.1 Power Supply System with Embedded Voltage Regulator

The XC878 microcontroller requires two different levels of power supply:

- 3.3 V or 5.0 V for the Embedded Voltage Regulator (EVR) and Ports
- 2.5 V for the core, memory, on-chip oscillator, and peripherals

**Figure 7-1** shows the XC878 power supply system. A power supply of 3.3 V or 5.0 V must be provided from the external power supply pin. The 2.5 V power supply for the logic is generated by the EVR. The EVR helps reduce the power consumption of the whole chip and the complexity of the application board design.

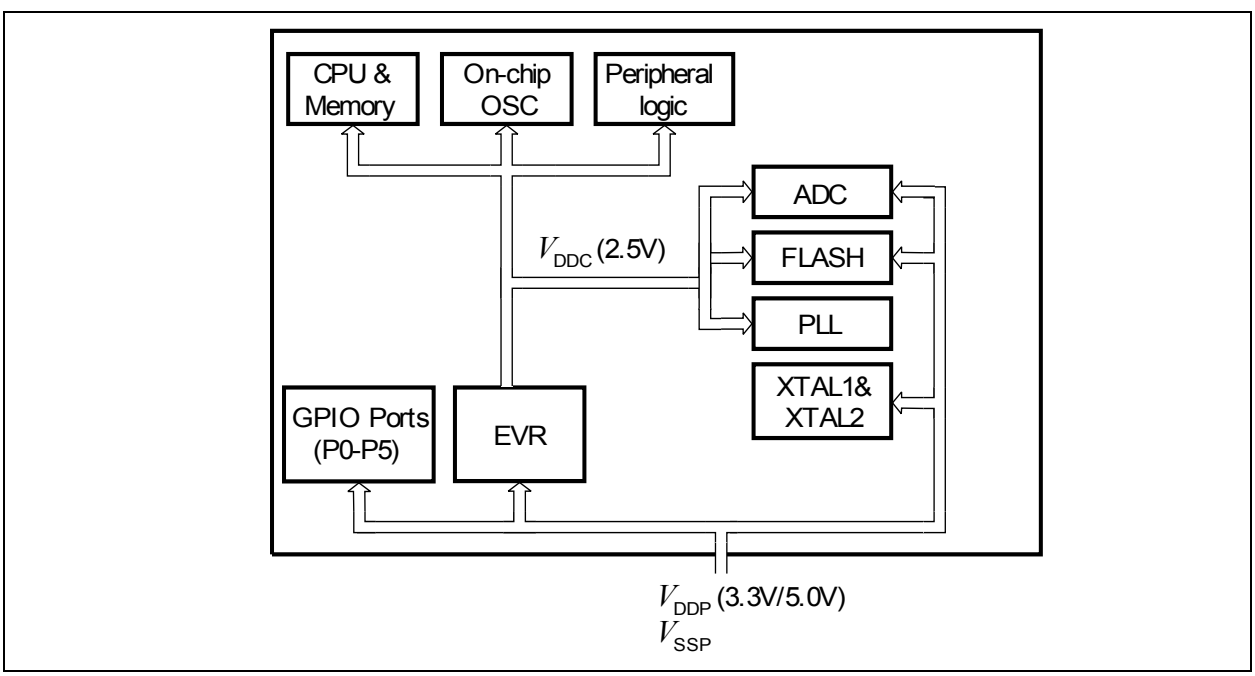



**Figure 7-1 XC878 Power Supply System**

**EVR Features:**

- Input voltage ($V_{DDP}$): 3.3 V/5.0 V
- Output voltage ($V_{DDC}$): 2.5 V +/-7.5%
- Low power voltage regulator provided in power-down mode
- $V_{DDP}$ prewarning detection
- $V_{DDC}$ brownout detection

The EVR consists of a main voltage regulator and a low power voltage regulator. In active mode, both voltage regulators are enabled. In power-down mode, the main voltage regulator is switched off, while the low power voltage regulator continues to function and provide power supply to the system with low power consumption.

The EVR has the $V_{DDC}$ and $V_{DDP}$ detectors. The $V_{DDC}$ detector has a threshold voltage levels for brownout (1.8 V). If $V_{DDC}$ is below 1.8 V, the brownout reset is activated, putting the microcontroller into a reset state. The brownout detection is enabled in both active and power down mode.

As for $V_{DDP}$, there is a prewarning threshold of 4.0 V if the external power supply is 5.0 V. When $V_{DDP}$ is below 4.0 V, the $V_{DDP}$ NMI flag NMISR.FNMIVDDP is set and an NMI request to the CPU is activated provided $V_{DDP}$ NMI is enabled (NMICON.NMIVDDP). If an external power supply of 3.3 V is used, the $V_{DDP}$ detector is switched off. In power-down mode, all detectors are disabled except for the brownout detector.

The EVR also has a power-on reset (POR) detector for $V_{DDC}$ to ensure correct power up. The voltage level detection of POR is 1.8 V. The monitoring function is used in both active mode and power-down mode. During power up, after $V_{DDC}$ exceeds 1.8 V, the reset of EVR is extended by a delay that is typically 300 µs.

## 7.2 Reset Control

The XC878 has five types of resets: power-on reset, hardware reset, watchdog timer reset, power-down wake-up reset, and brownout reset.

When theXC878 is first powered up, the status of certain pins (see **Table 7-2**) must be defined to ensure proper start operation of the device. At the end of a reset sequence, the sampled values are latched to select the desired boot option, which cannot be modified until the next power-on reset or hardware reset. This guarantees stable conditions during the normal operation of the device.

The hardware reset function can be used during normal operation or when the chip is in power-down mode. A reset input pin $\overline{\text{RESET}}$ is provided for the hardware reset.

The Watchdog Timer (WDT) module is also capable of resetting the device if it detects a malfunction in the system.

Another type of reset that needs to be detected is the reset while the device is in power-down mode (i.e., wake-up reset). While the contents of the static RAM are undefined after a power-on reset, they are well defined after a wake-up reset from power-down mode.

A brownout reset is triggered if the $V_{DDC}$ supply voltage dips below 1.8 V.

### 7.2.1 Types of Resets

#### 7.2.1.1 Power-On Reset

The supply voltage $V_{DDP}$ is used to power up the chip. The EVR is the first module in the chip to be reset, which includes:

1. Startup of the main voltage regulator and the low power voltage regulator.
2. When $V_{DDP}$ and $V_{DDC}$ reach the threshold of the $V_{DDP}$ and $V_{DDC}$ detectors, the reset of EVR becomes inactive.

A typical example of the circuitry connected to the power supply of XC878 is shown in **Figure 7-2**. For a voltage regulator with $IDD_{max}$ = 100 mA, the $V_{DDP}$ capacitor value is 10 µF. $V_{DDC}$ capacitor value has a maximum value of 220 nF.

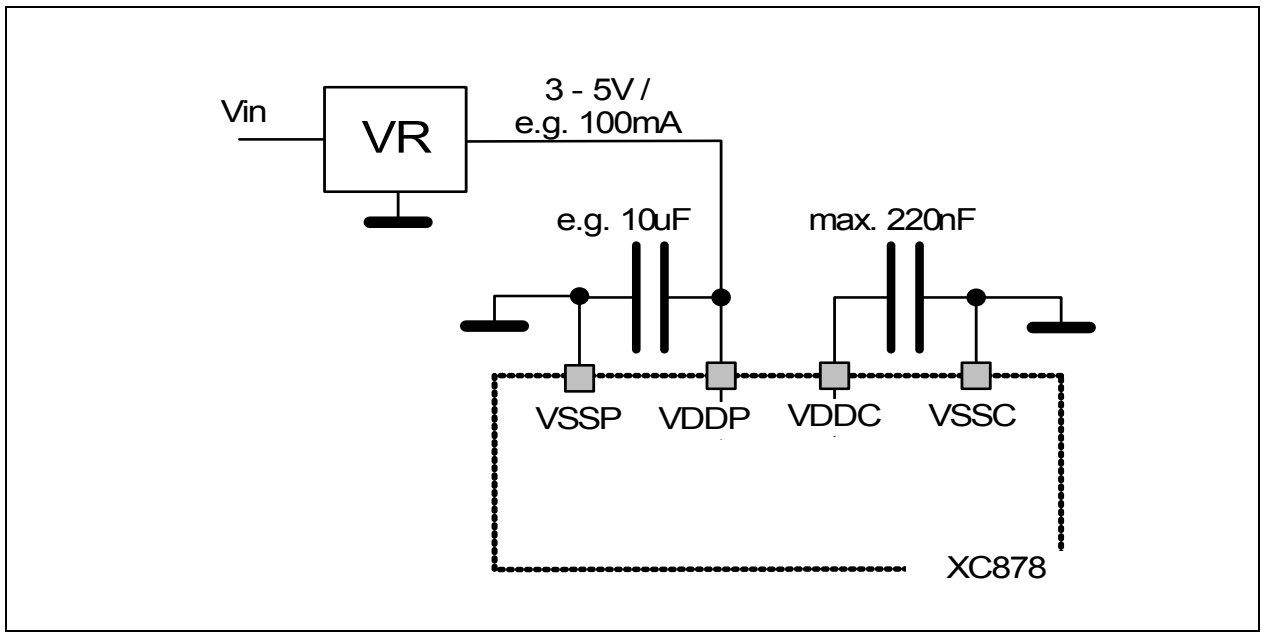



**Figure 7-2 Example of a Power Supply Circuitry**

When the system starts up, the output of PLL is disabled. Once the EVR is stable, provided the oscillator is running, the PLL is connected and the continuous lock detection ensures that PLL starts functioning. Following this, as soon as the system clock is stable, the reset to CPU is released and program execution starts.

The status of pins MBC, TMS and P0.0 is latched by the reset. The latched values are used to select the boot options (see **Section 7.2.3**). A correctly executed reset leaves the system in a defined state. The program execution starts from location $0000_H$.

**Figure 7-3** shows the power-on reset sequence.

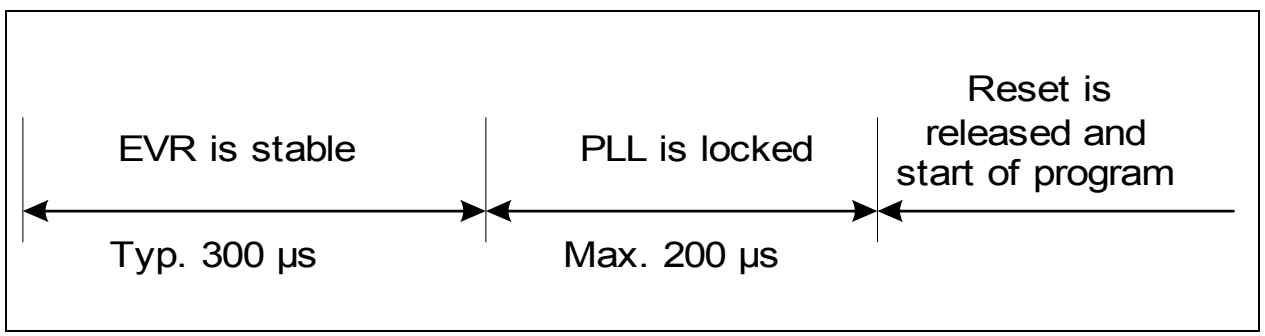



**Figure 7-3 Power-on Reset**

### 7.2.1.2 Hardware Reset

An external hardware reset sequence is started when the reset input pin $\overline{\text{RESET}}$ is asserted low. After the $\overline{\text{RESET}}$ pin is deasserted, the reset sequence is the same as the power-on reset sequence, as shown in **Figure 7-3**. A hardware reset through $\overline{\text{RESET}}$ pin will terminate the idle mode or the power-down mode.

The status of pins MBC, TMS and P0.0 is latched by the reset. The latched value is used to select the boot options (see **Section 7.2.3**).

### 7.2.1.3 Watchdog Timer Reset

The watchdog timer reset is an internal reset. The Watchdog Timer (WDT) maintains a counter that must be refreshed or cleared periodically. If the WDT is not serviced correctly and in time, it will generate an NMI request to the CPU and then reset the device after a predefined time-out period. Bit PMCON0.WDTRST is used to indicate the watchdog timer reset status.

For watchdog timer reset, as the EVR is already stable and PLL lock detection is not needed, the timing for watchdog timer reset is shorter compared to the other types of resets.

### 7.2.1.4 Power-Down Wake-Up Reset

Power is still applied to the XC878 during power-down mode, as the low power voltage regulator is still operating. If power-down mode is entered appropriately, all important system states will have been preserved in the Flash by software.

If the XC878 is in power-down mode, three options are available to awaken it:

- through RXD
- through EXINT0
- through RXD or EXINT0

Selection of these options is made via the control bit PMCON0.WS. The wake-up from power-down can be with reset or without reset; this is chosen by the PMCON0.WKSEL bit. The wake-up status (with or without reset) is indicated by the PMCON0.WKRS bit.

Once XC878 is awake, the power-down wake-up reset sequence is similar to the power-on reset sequence.

In addition to the above-mentioned three options, the power-down mode can also be exited by the hardware reset through $\overline{\text{RESET}}$ pin.

### 7.2.1.5 Brownout Reset

In active mode and power-down mode, the $V_{DDC}$ detector in EVR detects brownout when the core supply voltage $V_{DDC}$ dips below the threshold voltage $V_{DDC_TH}$ (1.8 V). The brownout will cause the device to be reset. Once the brownout reset takes place, the reset sequence is the same as the power-on reset sequence, as shown in **Figure 7-3**. All ports are at tristate during brownout.

### 7.2.2 Module Reset Behavior

Table 7-1 lists the functions of the XC878 and the various reset types that affect these functions. The symbol "■" signifies that the particular function is reset to its default state.

**Table 7-1 Effect of Reset on Device Functions**

| Module/ Function | Wake-Up Reset | Watchdog Reset | Hardware Reset | Power-On Reset | Brownout Reset |
|---|---|---|---|---|---|
| **CPU Core** | ■ | ■ | ■ | ■ | ■ |
| **Peripherals** | ■ | ■ | ■ | ■ | ■ |
| **On-Chip Static RAM** | Not affected, Reliable | Not affected, Reliable | Not affected, Reliable | Affected, un-reliable | Affected, un-reliable |
| **Oscillator, PLL** | ■ | Not affected | ■ | ■ | ■ |
| **Port Pins** | ■ | ■ | ■ | ■ | ■ |
| **EVR** | The voltage regulator is switched on | Not affected | Not affected | ■ | ■ |
| **FLASH** | ■ | ■ | ■ | ■ | ■ |
| **NMI** | ■ | ■ | ■ | ■ | ■ |

### 7.2.3 Booting Scheme

When the XC878 is reset, it must identify the type of configuration with which to start the different modes once the reset sequence is complete. Thus, boot configuration information that is required for activation of special modes and conditions needs to be applied by the external world through input pins. After power-on reset or hardware reset, the pins MBC, TMS and P0.0 collectively select the different boot options. **Table 7-2** shows the available boot options in the XC878.

**Table 7-2 XC878 Boot Selections [1)]**

| MBC | TMS | P0.0 | Type of Mode | PC Start Value |
|---|---|---|---|---|
| 1 | 0 | x | User Mode; on-chip OSC/PLL non-bypassed (normal) | $0000_H$ |
| 1 | 1 | x | User (JTAG) Mode[2)]; on-chip OSC/PLL non-bypassed (normal) | $0000_H$ |
| 0 | 0 | x | BSL Mode (LIN Mode[3)], UART/ MultiCAN Mode[4)5)] and Alternate BSL Mode[6)]); on-chip OSC/PLL non-bypassed | $0000_H$ |
| 0 | 1 | 0 | OCDS Mode; on-chip OSC/PLL non-bypassed (normal) | $0000_H$ |

1) In addition to the pins MBC, TMS and P0.0, TM pin also requires an external pull down for all the boot options.

2) Normal user mode with standard JTAG (TCK,TDI,TDO) pins for hot-attach purpose.

3) If a device is programmed as LIN, LIN BSL is always used instead of UART/MultiCAN.

4) UART or MultiCAN BSL is decoded by firmware based on the protocol for product variant with MultiCAN. If no MultiCAN and LIN variant, UART BSL is used.

5) In MultiCAN BSL mode, the clock source is switched to XTAL by firmware, bypassing the on-chip oscillator. This avoids any frequency invariance with the on-chip oscillator and allows other frequency clock input, thus ensuring accurate baud rate detection (especially at high bit rates).

6) Alternate BSL Mode is a user defined BSL code programmed in Flash. It is entered if the AltBSLPassword is valid.

*Note: The boot options are valid only with the default set of UART and JTAG pins.*

### 7.2.4 Register Description

**Table 7-3 Reset Values of Register PMCON0**

| Reset Source | Reset Value |
|---|---|
| Power-on Reset/Hardware Reset/Brownout Reset | $1000\ 0000_B$ |
| Watchdog Timer Reset | $1100\ 0000_B$ |
| Power-down Wake-up Reset | $1010\ 0000_B$ |

**Table 7-4 Reset Values of Register CR_MISC**

| Reset Source | Reset Value |
|---|---|
| Power-on Reset/Brownout Reset | $0000\ 0000_B$ |
| Hardware Reset | $0000\ 0001_B$ |

**PMCON0**
**Power Mode Control Register 0** **Reset Value: See Table 7-3**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| VDDP WARN | WDTRST | WKRS | WKSEL | SD | PD | WS | |
| rh | rwh | rwh | rw | rw | rwh | rw | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **WS** | [1:0] | rw | **Wake-Up Source Select**<br>00 No wake-up is selected.<br>01 Wake-up source RXD (falling edge trigger) is selected.<br>10 Wake-up source EXINT0 (falling edge trigger) is selected.<br>11 Wake-up source RXD (falling edge trigger) or EXINT0 (falling edge trigger) is selected. |
| **WKSEL** | 4 | rw | **Wake-Up Reset Select Bit**<br>0 Wake-up without reset<br>1 Wake-up with reset |

| Field | Bits | Type | Description |
|---|---|---|---|
| **WKRS** | 5 | rwh | **Wake-Up Indication Bit**<br>0 No wake-up occurred.<br>1 Wake-up has occurred.<br>This bit can only be set by hardware and reset by software. |
| **WDTRST** | 6 | rwh | **Watchdog Timer Reset Indication Bit**<br>0 No watchdog timer reset occurred.<br>1 Watchdog timer reset has occurred.<br>This bit can only be set by hardware and reset by software. |
| **VDDPWARN** | 7 | rh | **$V_{DDP}$ Warning Threshold Indication Bit**<br>0 Below $V_{DDP}$ warning threshold level.<br>1 Above $V_{DDP}$ warning threshold level. |

**CR_MISC**
**Clock and Reset Miscellaneous Register** **Reset Value: See Table 7-4**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **CCCFG** | **MDUCCFG** | **CCUCCFG** | **T2CCFG** | **0** | | | **HDRST** |
| rw | rw | rw | rw | r | | | rwh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **HDRST** | 7 | rwh | **Hardware Reset Indication Bit**<br>0 No hardware reset occurred.<br>1 Hardware reset occurred.<br>This bit can only be set by hardware and reset by software. |
| **0** | [3:1] | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

## 7.3 Clock System

The XC878 clock system performs the following functions:

- Acquires and buffers incoming clock signals to create a master clock frequency
- Distributes in-phase synchronized clock signals throughout the system
- Divides a system master clock frequency into lower frequencies for power saving mode

### 7.3.1 Clock Generation Unit

The Clock Generation Unit (CGU) in the XC878 consists of an oscillator circuit and a Phase-Locked Loop (PLL). In the XC878, the oscillator can be from either of these two sources: the on-chip oscillator (4 MHz) or the external oscillator (2 MHz to 20 MHz). The term “oscillator” is used to refer to both on-chip oscillator and external oscillator, unless otherwise stated. After the reset, on-chip oscillator will be used by default. The external oscillator can be selected via software. The PLL can convert a low-frequency external clock signal from the oscillator circuit to a high-speed internal clock for maximum performance.

**Figure 7-4** shows the block diagram of CGU.

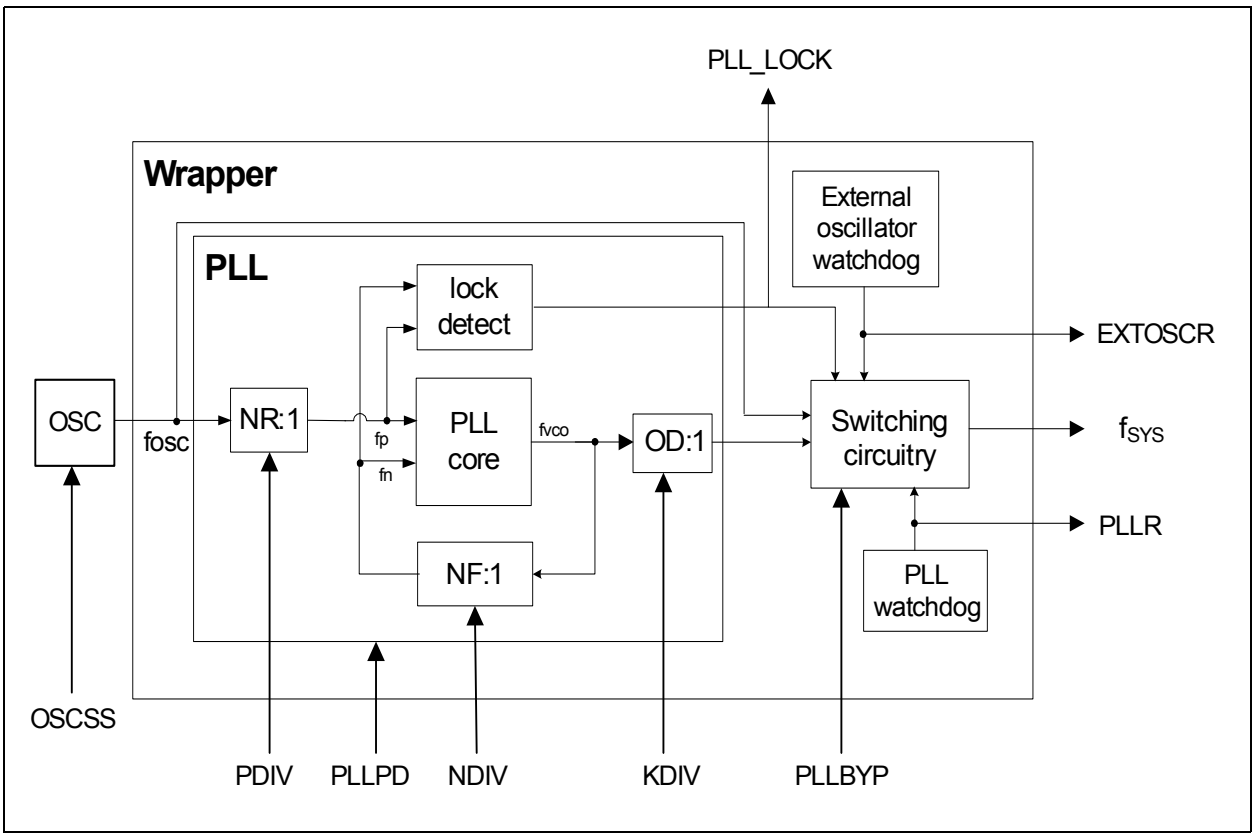



**Figure 7-4 CGU Block Diagram**

### 7.3.1.1 Functional Description

When the XC878 is powered up, the PLL output is disabled. After the EVR is stable, provided the oscillator is running, the PLL will be connected and the continuous lock detection will ensure that the PLL starts functioning. Once reset has been released, bit LOCK will be set to 1 to indicate that the PLL is locked.

#### Loss-of-Clock Operation

Loss-of-Clock may happens during one or a combination of these situations:

- PLL loses it's lock to the oscillator; or
- PLL output runs at less than 25 MHz; or
- External oscillator runs at less than 1 MHz when it is selected as the input source to PLL; or
- External oscillator runs at less than 1 MHz when it is programmed as the system frequency by setting OSC_CON.PLLBYP = 1 and OSC_CON.OSCSS = 0.

During loss-of-clock, one of the signal stated below could be deasserted:

- the PLL lock status, PLL_CON.PLL_LOCK
- the PLL run status, PLL_CON.PLLR
- the external oscillator run status, OSC_CON.EXTOSCR

When PLL loses its lock to the oscillator, the PLL Loss-of-Clock NMI flag NMISR.FNMIPLL is set and an NMI request to the CPU is activated if PLL NMI is enabled (NMICON.NMIPLL). In addition, the PLL_LOCK flag, PLLR flag and EXTOSCR flag may be reset depending on different application. While loss-of-clock remains true, the system clock is automatically switched to bypass the PLL output and the on-chip oscillator is used as the clock source. Emergency routines can be executed with the XC878 clocked at the on-chip frequency.

The XC878 remains in this loss-of-clock state until the next power-on reset, hardware reset or after a successful lock recovery has been performed. For the case of unsuccessfully loss-of-clock recovery, emergency measures must be executed. Emergency measures such as a system shut down can be carried out by the user.

If the PLL is not the system's clock source (PLLBYP = 1) or the external oscillator is not the system frequency when the loss-of-clock is detected, only the lock flag is reset (PLL_LOCK = 0) and no further action is taken. This allows the PLL parameters to be switched dynamically.

#### Loss-of-Clock Recovery

If loss-of-clock has occurred, the system clock will automatically switch to run from on-chip oscillator frequency (4 MHz). EXTOSCR, PLLR and PLL_LOCK flag must be checked and normal operation can only resumed if all bits are set. To ensure that the recovery from loss-of-clock, the following sequence must be performed.

When external oscillator is selected as the input source to PLL:

1. Restart the external oscillator watchdog by setting bit EORDRES.
2. Restart the PLL watchdog by setting bit PLLRDRES
3. Bypass the PLL output (PLLBYP = 1).
4. Select the PLL power down mode (PLLPD = 1).
5. Select the internal oscillator as the source of oscillator (OSCSS = 0).
6. Wait for 65 cycles based on internal oscillator frequency

- If bit OSC_CON.EXTOSCR is set after 65 internal oscillator clock cycles, then:

1. Select the external oscillator as the source of oscillator (OSCSS = 1).
2. Reprogram NDIV, PDIV and KDIV values is necessary.
3. Change to PLL normal operation mode (PLLPD=0).
4. Wait until the PLL_LOCK and PLLR flag has been set.
5. Disable the bypass of PLL output (PLLBYP = 0) and normal operation resumed.

When internal oscillator is selected as the input source to PLL:

1. Restart the PLL watchdog by setting bit PLLRDRES
2. Bypass the PLL output (PLLBYP = 1).
3. Select the PLL power down mode (PLLPD = 1).
4. Reprogram NDIV, PDIV and KDIV values is necessary.
5. Change to PLL normal operation mode (PLLPD=0).
6. Wait until the PLL_LOCK and PLLR flag has been set.
7. Disable the bypass of PLL output (PLLBYP = 0) and normal operation resumed.

When external oscillator is selected as system frequency:

1. Restart the external oscillator watchdog by setting bit EORDRES.
2. Select the internal oscillator as the source of oscillator (OSCSS = 0).
3. Wait for 65 cycles based on internal oscillator frequency

- If bit EXTOSCR is set after 65 internal oscillator clock cycles, then:

1. Select the external oscillator as the source of oscillator (OSCSS = 1).

### Changing PLL Parameters

To change the PLL parameters, the following sequence must be performed:

1. Bypass the PLL output (PLLBYP = 1).
2. Select the PLL power down mode (PLLPD = 1).
3. Program desired NDIV, PDIV and KDIV values.
4. Change to PLL normal operation mode (PLLPD=0).
5. Restart the PLL watchdog by setting bit PLLRDRES.
6. Wait until the PLL_LOCK and PLLR flag has been set.
7. Disable the bypass of PLL output (PLLBYP = 0).

### Select the External Oscillator as PLL input source

To select the external oscillator, the following sequence must be performed:

1. Bypass the PLL output (PLLBYP = 1); On-chip oscillator as system clock.
2. Select the PLL power down mode (PLLPD = 1).
3. Power up external oscillator (XPD = 0)
4. Wait for 1.5 ms until the external oscillator is stable (the delay time should be adjusted accroding to different external oscillators).
5. Restart the external oscillator watchdog by setting bit EORDRES.
6. Wait for 65 cycles based on internal oscillator frequency.
7. Wait until EXTOSCR is set.
8. The source of external oscillator is selected by setting bit OSCSS.; External oscillator as system clock
9. Program desired NDIV, PDIV and KDIV values.
10. Change to PLL normal operation mode (PLLPD=0).
11. Restart the PLL watchdog by setting PLLRDRES.
12. Wait untill the PLL_LOCK and PLLR flag has been set.
13. Disable the bypass of PLL output (PLLBYP = 0).

**Select the External Oscillator as system clock using Direct Drive (PLL Bypass) mode**

To select the external oscillator as system clock, the following sequence must be performed:

1. Bypass the PLL output (PLLBYP = 1); On-chip oscillator as system clock.
2. Select the PLL power down mode (PLLPD = 1).
3. Power up external oscillator (XPD = 0)
4. Wait for 1.5 ms until the external oscillator is stable (the delay time should be adjusted accroding to different external oscillators).
5. Restart the external oscillator watchdog by setting bit EORDRES.
6. Wait for 65 cycles based on internal oscillator frequency.
7. Wait until EXTOSCR is set.
8. The source of external oscillator is selected by setting bit OSCSS.; External oscillator as system clock

To minimize power consumption when the on-chip oscillator is used, the external oscillator circuitry may be powered down by setting bits XPD.

### 7.3.2 Clock Source Control

The clock system provides two ways to generate the system clock:

**Direct Drive (PLL Bypass Operation)**

During PLL bypass operation, the system clock has the same frequency as the external clock source.

(7.1)

$$f_{SYS} = f_{OSC}$$

**PLL Mode**

The CPU clock is derived from the oscillator clock, divided by the NR factor (PDIV), multiplied by the NF factor (NDIV), and divided by the OD factor (KDIV). PLL output must not be bypassed for this PLL mode. The PLL mode is used during normal system operation.

(7.2)

$$f_{SYS} = f_{OSC} \times \frac{NF}{NR \times OD}$$

In normal running mode, the system works in the PLL mode.

For the XC878, the value of NF, NR and OD can be selected by bits NDIV, PDIV and KDIV respectively for different oscillator inputs inorder to obtain the required fsys. But the combination of these factors must fulfill the following condition:

- 100 MHz < $f_{VCO}$ < 175 MHz
- 800 KHz < $f_{OSC}$ / (2 * NR) < 8 MHz

Table 7-5 provides examples on how the typical system frequency of fsys = 144 MHz can be obtained for the different oscillator sources.

**Table 7-5 System frequency ($f_{sys}$ = 144 MHz)**

| Oscillator | fosc | N | P | K | fsys |
|---|---|---|---|---|---|
| On-chip | 4 MHz | 72 | 2 | 1 | 144 MHz |
| External | 8 MHz | 72 | 4 | 1 | 144 MHz |
| | 6 MHz | 72 | 3 | 1 | 144 MHz |
| | 4 MHz | 72 | 2 | 1 | 144 MHz |

### 7.3.3 Clock Management

The Clock Management sub-module generates all clock signals required within the microcontroller from the basic clock. It consists of:

- Generate all clocks in the system
- Basic clock slow down circuitry
- Centralized enable/disable circuit for clock control

Figure 7-5 shows the clock generation from the system frequency $f_{sys}$. In normal running mode, the typical frequencies of different modules are as follows:

- CPU clock: CCLK, SCLK = 24 MHz
- Fast clock: FCLK = 48 MHz
- Peripheral clock: PCLK = 24 MHz

For the XC878, there are five modules, namely MultiCAN, MDU, CORDIC, CCU6 and T2CCU which could be clocked at 24 MHz or 48 MHz clock. The selection of the clock frequency for these five modules are done via CMCON and CR_MISC register.

Furthermore, a clock output (CLKOUT) is available on pin P(0.0 or 0.7) as an alternate output. If bit COUTS = 00, the output clock is from on-chip oscillator output frequency; if bit COUTS = 01, the clock output frequency is chosen by the bit field COREL. Under this selection, the clock output frequency can further be divided by 2 using toggle latch (bit TLEN is set to 1), so that the resulting output frequency has 50% duty cycle. If bit COUTS = 10, the PCLK clock is selected.

In idle mode, only the CPU clock CCLK is disabled. In power-down mode, CCLK, SCLK, FCLK, CCLK2 and PCLK are all disabled. If slow-down mode is enabled, the clock to the core and peripherals will be divided by a programmable factor that is selected by the bit field CMCON.CLKREL.

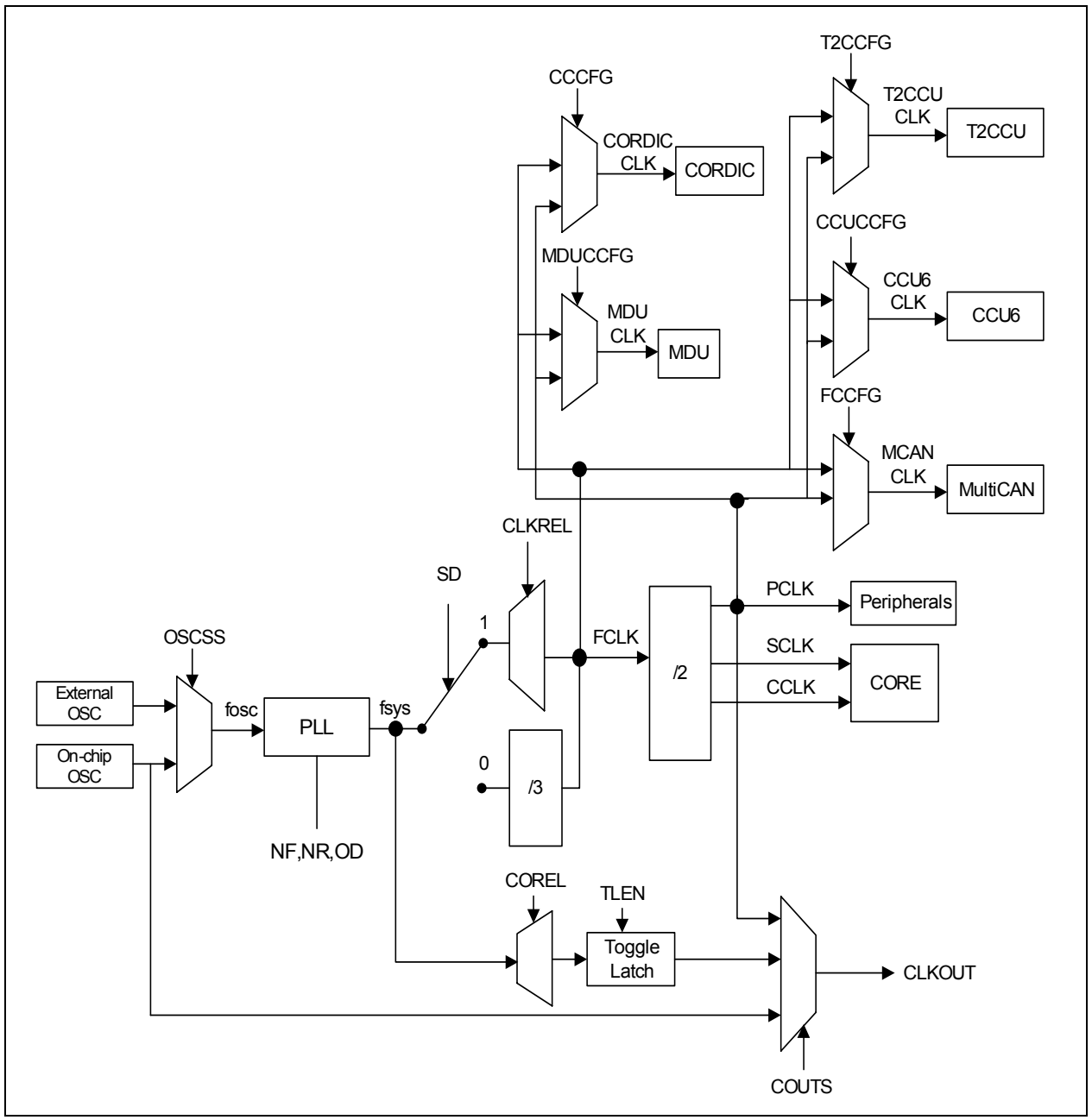



**Figure 7-5 Clock Generation from $f_{sys}$**

### 7.3.4 Register Description

**OSC_CON**
**OSC Control Register** **Reset Value: 0X00 1000$_B$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **PLL RDRES** | **PLLBYP** | **PLLPD** | **0** | **XPD** | **OSCSS** | **EORDRES** | **EXTOSCR** |
| rwh | rwh | rw | r | rw | rwh | rwh | rh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **EXTOSCR** | 0 | rh | **External Oscillator Run Status Bit**<br>This bit shows the state of the external oscillator watchdog.<br>0 External oscillator is not running.<br>1 External oscillator is running. |
| **EORDRES** | 1 | rwh | **External Oscillator Watchdog Reset**<br>0 No Operation.<br>1 External oscillator watchdog is reset and restarted.<br>This bit will be reset automatically to 0. |
| **OSCSS** | 2 | rwh | **Oscillator Source Select**<br>0 On-chip oscillator is selected.<br>1 External oscillator is selected.<br>This bit is protected by the Bit-Protection Scheme. It is updated by hardware after reset depending on boot configuration.<br>This bit is not modified by hardware due to switching to on-chip oscillator during the PLL loss-of-clock. The previous setting will be retained.<br>*Note: To make the switch clock source smoothly, before switching from on-chip oscillator to external oscillator, clock output from external oscillator must be kept stable.* |
| **XPD** | 3 | rw | **XTAL Power-down Control**<br>0 XTAL is not powered down.<br>1 XTAL is powered down.<br>This bit is protected by the Bit-Protection Scheme.<br>*Note: Before setting OSCSS bit to 1, XPD bit must be ensure to be 0.* |

| Field | Bits | Type | Description |
|---|---|---|---|
| **PLLPD** | 5 | rw | **PLL Power Down Control**<br>0 PLL is not powered down.<br>1 PLL is powered down.<br>This bit is protected by the Bit-Protection Scheme. |
| **PLLBYP** | 6 | rwh | **PLL Output Bypass Contro**<br>0 PLL output is not bypassed.<br>1 PLL output is bypassed.<br>This bit is updated by hardware after reset, depending on boot configuration. |
| **PLLPD** | 7 | rw | **PLL Watchdog reset**<br>0 No Operation.<br>1 PLL watchdog is reset and restarted.<br>This bit will be reset automatically to 0. |
| **0** | 4 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

**PLL_CON**
**PLL Control Register** **Reset Value: 0001 1000$_B$**

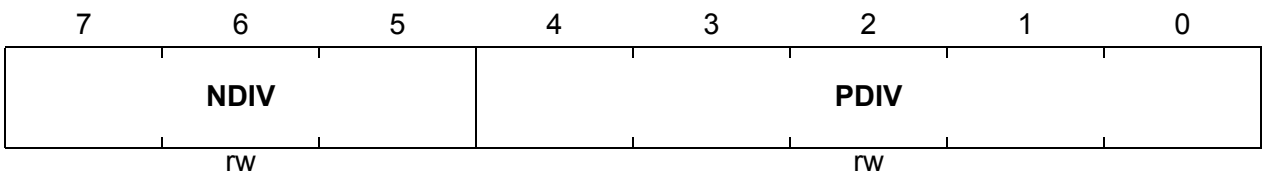
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **NDIV** | | | | | | **PLLR** | **PLL_ LOCK** |
| rw | | | | | | rh | rh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **PLL_LOCK** | 0 | rh | **PLL Lock Status Flag**<br>0 PLL is not locked.<br>1 PLL is locked. |
| **PLL_LOCK** | 1 | rh | **PLL Run Status Flag**<br>0 PLL is not running.<br>1 PLL is running. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **NDIV** | [7:2] | rw | **PLL NF-Divider together with PLL_CON1[7:5]**<br>000000000 NF = 2<br>000000001 NF = 3<br>000000010 NF = 4<br>.....<br>001000110 NF = 72 (default)<br>.....<br>111111101 NF = 511<br>111111110 NF = 512<br>111111111 NF = 513<br>The NDIV bit is a protected bit. When the Protection Scheme is activated, this bit cannot be written directly. See Protection Scheme (**Chapter 3.4.4.1**) for more details. |

*Note: The reset value of register PLL_CON is **0001 1000**$_B$. One clock cycle after reset, bit LOCK will be set to 1 if the PLL is locked, then the value **0001 1001**$_B$ will be observed.*

**PLL_CON1**
**PLL Control Register 1** **Reset Value: 0010 0000$_B$**



| Field | Bits | Type | Description |
|---|---|---|---|
| **PDIV** | [4:0] | rw | **PLL NR-Divider]**<br>00000 NR = 2 (defailt)<br>00001 NR = 3<br>00010 NR = 4<br>.....<br>11101 NR = 31<br>11110 NR = 32<br>11111 NR = 33<br>The PDIV bit is a protected bit. When the Protection Scheme is activated, this bit cannot be written directly. See Protection Scheme (**Chapter 3.4.4.1**) for more details. |
| **NDIV** | [7:5] | rw | **MSBs PLL NF-Divider**<br>Together with PLL_CON.NDIV to form a 9 bit divider. |
| **0** | 0 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

*Note: The reset value of register PLL_CON is **0010 0000**$_B$. One clock cycle after reset, bit LOCK will be set to 1 if the PLL is locked, then the value **0010 0001**$_B$ will be observed.*

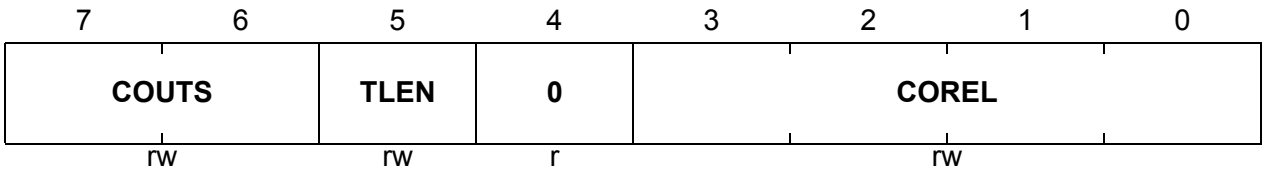
**CMCON**
**Clock Control Register** **Reset Value: 0001 0000$_B$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **KDIV** | | **0** | **FCCFG** | **CLKREL** | | | |
| rw | | r | rw | rw | | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **CLKREL** | [3:0] | rw | **Clock Divider**<br>0000 $f_{SYS}$/6<br>0001 $f_{SYS}$/12<br>0010 $f_{SYS}$/18<br>0011 $f_{SYS}$/24<br>0100 $f_{SYS}$/36<br>0101 $f_{SYS}$/48<br>0110 $f_{SYS}$/72<br>0111 $f_{SYS}$/96<br>1000 $f_{SYS}$/144<br>1001 $f_{SYS}$/192<br>1010 $f_{SYS}$/288<br>1011 $f_{SYS}$/384<br>1100 $f_{SYS}$/576<br>1101 $f_{SYS}$/768<br>1110 $f_{SYS}$/1152<br>*Note: The changes of bit field CLKREL take effect only when PMCON0.SD bit is set.*<br>*Note: The clock division listed above is inclusive of the fixed divider factor of 2. See **Table 7-5**.* |
| **FCCFG** | 4 | rw | **MultiCAN Clock Configuration**<br>0 $f_{MCAN}$ is configured to run at PCLK.<br>1 $f_{MCAN}$ is configured to run at 2 times PCLK. |
| **KDIV** | [7:6] | rw | **PLL OD-Divider**<br>00 OD = 1 (default)<br>01 OD = 2<br>10 OD = 2<br>11 OD = 4<br>The KDIV bit is a protected bit. When the Protection Scheme is activated, this bit cannot be written directly. See Protection Scheme (**Chapter 3.4.4.1**) for more details. |
| **0** | 5 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

## COCON
## Clock Output Control Register

**Reset Value: 0000 0000$_B$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **COREL** | [3:0] | rw | **Clock Output Divider**<br>0000 $f_{SYS}$/3<br>0001 $f_{SYS}$/6<br>0010 $f_{SYS}$/9<br>0011 $f_{SYS}$/12<br>0100 $f_{SYS}$/15<br>0101 $f_{SYS}$/18<br>0110 $f_{SYS}$/21<br>0111 $f_{SYS}$/24<br>1000 $f_{SYS}$/30<br>1001 $f_{SYS}$/36<br>1010 $f_{SYS}$/39<br>1011 $f_{SYS}$/42<br>1100 $f_{SYS}$/48<br>1101 $f_{SYS}$/54<br>1110 $f_{SYS}$/60 |
| **TLEN** | 5 | rw | **Toggle Latch Enable**<br>This bit is only applicable when COUTS is set to 1.<br>0 Toggle Latch is disabled. Clock output frequency is chosen by the bit field COREL.<br>1 Toggle Latch is enabled. Clock output frequency is half of the frequency that is chosen by the bit field COREL. The clock output frequency has 50% duty cycle. |
| **COUTS** | [7:6] | rw | **Clock Out Source Select**<br>00 On-chip oscillator output frequency is selected.<br>01 Clock output frequency is chosen by the bit field COREL.<br>10 PCLK output frequency is selected.<br>11 Reserved. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | 4 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

*Note: Registers OSC_CON, PLL_CON, PLL_CON1, CMCON, and COCON are not reset during the watchdog timer reset.*

**CR_MISC**
**Clock and Reset Miscellaneous Register** **Reset Value: 0000 000X$_B$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **CCCFG** | **MDUCCFG** | **CCUCCFG** | **T2CCFG** | **0** | | | **HDRST** |
| rw | rw | rw | rw | r | | | rwh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **T2CCFG** | 4 | rw | **Timer 2 Capture/Compare Unit Clock Configuration**<br>0 $f_{T2CCU}$ is configured to run at PCLK.<br>1 $f_{T2CCU}$ is configured to run at 2 times PCLK. |
| **CCUCCFG** | 5 | rw | **CCU6 Clock Configuration**<br>0 $f_{CCU6}$ is configured to run at PCLK.<br>1 $f_{CCU6}$ is configured to run at 2 times PCLK. |
| **MDUCCFG** | 6 | rw | **MDU Clock Configuration**<br>0 $f_{MDU}$ is configured to run at PCLK.<br>1 $f_{MDU}$ is configured to run at 2 times PCLK. |
| **CCCFG** | 7 | rw | **CORDIC Clock Configuration**<br>0 $f_{CORDIC}$ is configured to run at PCLK.<br>1 $f_{CORDIC}$ is configured to run at 2 times PCLK. |
| **0** | [3:1] | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

# 8 Power Saving Modes

The power saving modes in the XC878 provide flexible power consumption through a combination of techniques, including:

- Stopping the CPU clock
- Stopping the clocks of individual system components
- Reducing clock speed of some peripheral components
- Power-down of the entire system with fast restart capability

After a reset, the active mode (normal operating mode) is selected by default (see **Figure 8-1**) and the system runs in the main system clock frequency. From active mode, different power saving modes can be selected by software. They are:

- Idle mode
- Slow-down mode
- Power-down mode

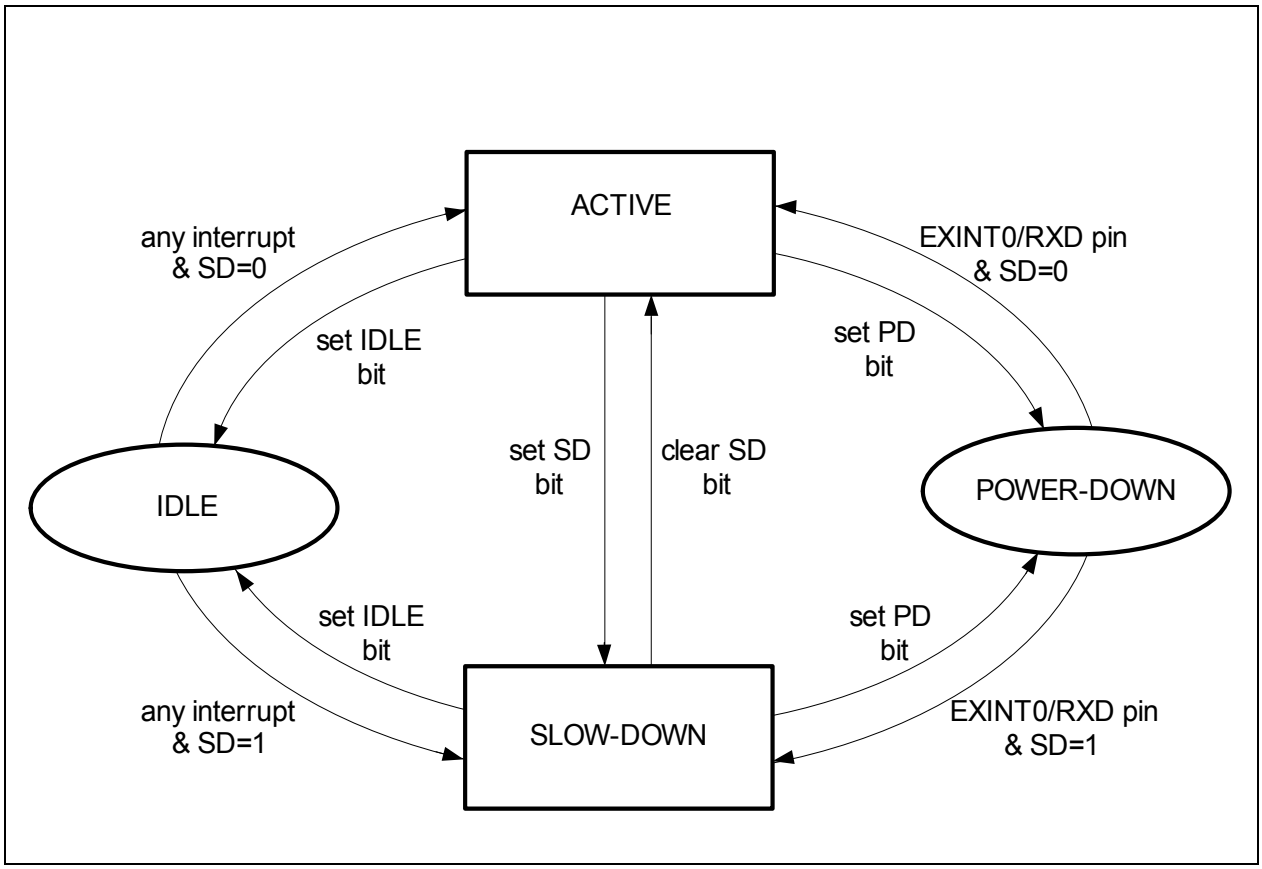



**Figure 8-1 Transition between Power Saving Modes**

## 8.1 Functional Description

This section describes the various power saving modes, their operations, and how they are entered and exited.

### 8.1.1 Idle Mode

The idle mode is used to reduce power consumption by stopping the core's clock.

In idle mode, the oscillator continues to run, but the core is stopped with its clock disabled. Peripherals whose input clocks are not disabled are still functional. The user should disable the Watchdog Timer (WDT) before the system enters the idle mode; otherwise, it will generate an internal reset when an overflow occurs and thus will disrupt the idle mode. The CPU status is preserved in its entirety; the stack pointer, program counter, program status word, accumulator, and all other registers maintain their data during idle mode. The port pins hold the logical state they had at the time the idle mode was activated.

Software requests idle mode by setting the bit PCON.IDLE to 1.

The system will return to active mode on occurrence of any of the following conditions:

- The idle mode can be terminated by activating any enabled interrupt. The CPU operation is resumed and the interrupt will be serviced. Upon RETI instruction, the core will return to execute the next instruction after the instruction that sets the IDLE bit to 1.
- An external hard reset signal ($\overline{\text{RESET}}$) is asserted.

### 8.1.2 Slow-Down Mode

The slow-down mode is used to reduce power consumption by decreasing the internal clock in the device.

The slow-down mode is activated by setting the bit SD in SFR PMCON0. The bit field CMCON.CLKREL is used to select a different slow-down frequency. The CPU and peripherals are clocked at this lower frequency. The slow-down mode is terminated by clearing bit SD.

The slow-down mode can be combined with the idle mode by performing the following sequence:

1. The slow-down mode is activated by setting the bit PMCON0.SD.
2. The idle mode is activated by setting the bit PCON.IDLE.

There are two ways to terminate the combined idle and slow-down modes:

- The idle mode can be terminated by activation of any enabled interrupt. CPU operation is resumed, and the interrupt will be serviced. The next instruction to be executed after the RETI instruction will be the one following the instruction that had set the bit IDLE. Nevertheless, the slow-down mode stays enabled and if required termination must be done by clearing the bit SD in the corresponding interrupt service

routine or at any point in the program where the user no longer requires the slow-down mode.
- The other way of terminating the combined idle and slow-down mode is through a hardware reset.

### 8.1.3 Power-down Mode

In power-down mode, the oscillator and the PLL are turned off. All the memories including Flash are put into the standby mode mode. The main voltage regulator is switched off, but the low power voltage regulator continues to operate. Therefore, all functions of the microcontroller are stopped and only the contents of the FLASH, on-chip RAM, XRAM and the SFRs are maintained. The port pins hold the logical state they had when the power-down mode was activated. For the digital ports, the user must take care from external side that the ports are not floating in power-down mode. This can be done with external pull-up/pull-down or putting the port to output. For Flash memories, user need to take care that the current flash program or erase operation has completetd before entering power down mode. Otherwise, it may need to be aborted if there is an urgent request of power down. In this case, the current operation may not be successful. **Figure 8-2** shows the flow with Flash abort during an urgent power down that triggers a NMI interrupt.

In power-down mode, the clock is turned off. Hence, it cannot be awakened by an interrupt or by the WDT. It is awakened only when it receives an external wake-up signal or reset signal.

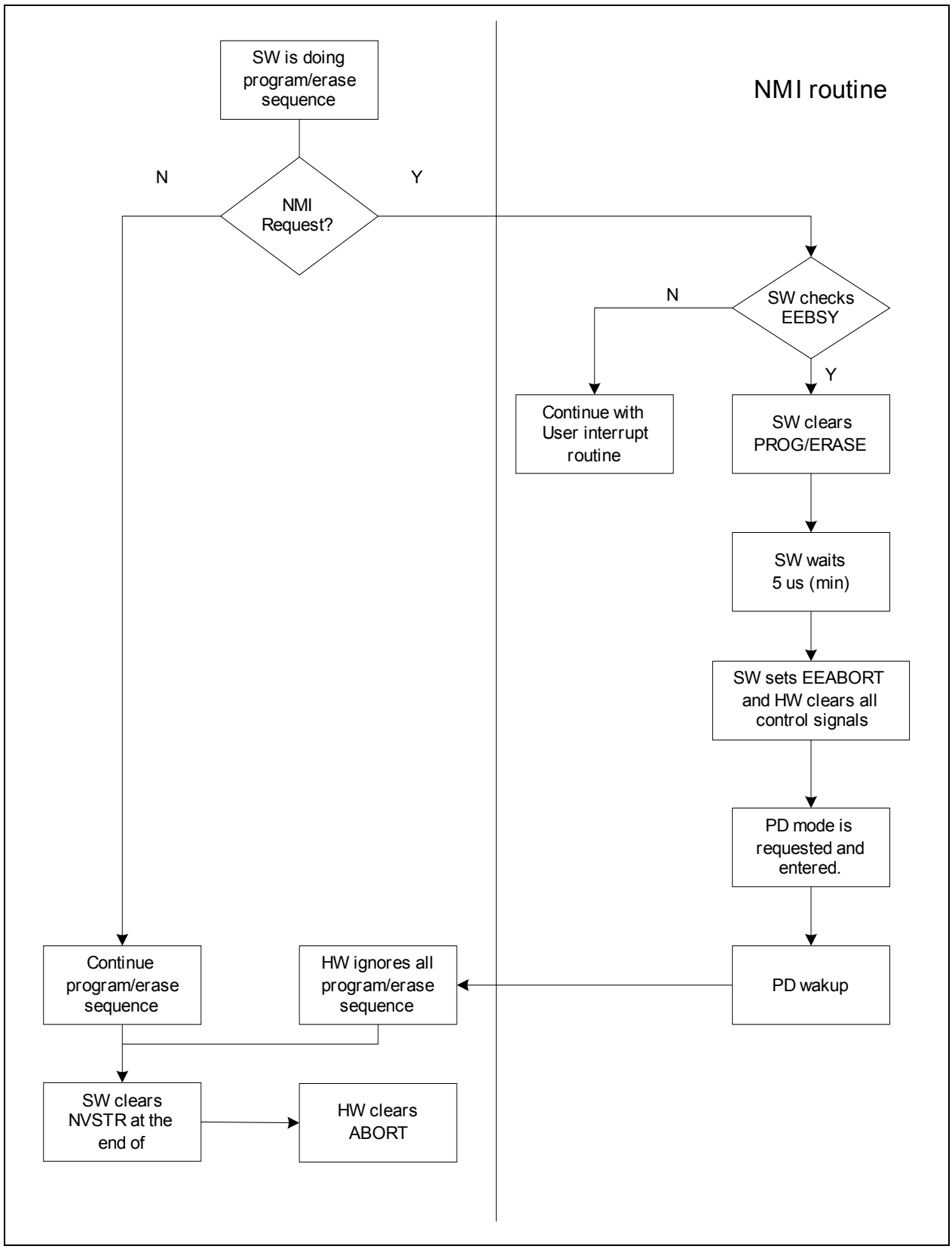



**Figure 8-2 Power Down with Flash Abort Flow**

### Entering Power-down Mode

Software requests power-down mode by setting the bit PMCON0.PD to 1.

Three NOP instructions must be inserted after the bit PMCON0.PD is set to 1. This ensures the first instruction (after three NOP instructions) is executed correctly after wake-up from power-down mode.

If the external wake-up from power-down is used, software must prepare the external environment of the XC878 to trigger one of these signals under the appropriate conditions before entering power-down mode. A wake-up circuit is used to detect a wake-up signal and activate the power-up. During power-down, this circuit remains active. It does not depend on any clocks. Exit from power-down mode can be achieved by applying a falling edge trigger to the:

- EXINT0 pin
- RXD pin
- RXD pin or EXINT0 pin

The wake-up source can be selected by the bit WS of the PMCON0 register. The wake-up with reset or without reset is selected by bit PMCON0.WKSEL. The wake-up source and wake-up type must be selected before the system enters the power-down mode.

### Exiting Power-down Mode

If power-down mode is exited via a hardware reset, the device is put into the hardware reset state.

When the wake-up source and wake-up type have been selected prior to entering power-down mode, the power-down mode can be exited via EXINT0 pin/RXD pin.

Bit MODPISEL.URRIS and MODPISEL.URRISH are used to select one of the three RXD inputs and bit MODPISEL.EXINT0IS is used to select one of the two EXINT0 inputs.

If bit WKSEL was set to 1 before entering power-down mode, the system will execute a reset sequence similar to the power-on reset sequence. Therefore, all port pins are put into their reset state and will remain in this state until they are affected by program execution.

If bit WKSEL was cleared to 0 before entering power-down mode, a fast wake-up sequence is used. The port pins continue to hold their state which was valid during power-down mode until they are affected by program execution.

The wake-up from power-down without reset undergoes the following procedure:

1. In power-down mode, EXINT0 pin/RXD pin must be held at high level.
2. Power-down mode is exited when EXINT0 pin/RXD pin goes low for at least 100 ns.
3. The main voltage regulator is switched on and takes approximately 300 µs to become stable.

4. The on-chip oscillator and the PLL are started. Typically, the on-chip oscillator takes approximately 8 µs to stabilize. The PLL will be locked within 200 µs after the on-chip oscillator clock is detected for stable nominal frequency.
5. The CPU operation is resumed. If wake-up source is EXINT0 pin, the interrupt will be serviced if EXINT0 is enabled before entering power-down mode. Upon RETI instruction, the core will return to execute the next instruction after the instruction that sets the PD bit. If wake-up source is RXD pin, the core will return to execute the next instruction after the instruction which sets the PD bit.

### 8.1.4 Peripheral Clock Management

The amount of reduction in power consumption that can be achieved by this feature depends on the number of peripherals running. Peripherals that are not required for a particular functionality can be disabled by gating off the clock inputs. For example, in idle mode, if all timers are stopped, and ADC, CCU6, MDU, MultiCAN and the serial interfaces are not running, maximum power reduction can be achieved. However, the user must take care when determining which peripherals should continue running and which must be stopped during active and idle modes.

The ADC, SSC, CCU6, CORDIC, MDU, MultiCAN, UART1, T2CCU and Timer 21 can be disabled (clock is gated off) by setting the corresponding bit in the PMCON1 register and PMCON2 register. Furthermore, the analog part of the ADC module may be disabled by resetting the GLOBCTR.ANON bit. This feature causes the generation of $f_{ADCI}$ to be stopped and allows a reduction in power consumption when no conversion is needed.

In order to save power consumption when the on-chip oscillator is used, XTAL should be powered down by setting bit OSC_CON.XPD. However, when the external oscillator is used, the on-chip oscillator cannot be powered down.

## 8.2 Register Description

**PMCON0**
**Power Mode Control Register 0** **Reset Value: $80_H$[1)]**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| VDDPWARN | WDTRST | WKRS | WKSEL | SD | PD | WS | |
| rh | rwh | rwh | rw | rw | rwh | rw | |

1) The reset value for watchdog timer reset is $C0_H$ and the reset value for power-down wake-up reset is $A0_H$.

| Field | Bits | Type | Description |
|---|---|---|---|
| **WS** | [1:0] | rw | **Wake-up Source Select**<br>00 No wake-up is selected.<br>01 Wake-up source RXD (falling edge trigger) is selected.<br>10 Wake-up source EXINT0 (falling edge trigger) is selected.<br>11 Wake-up source RXD (falling edge trigger) or EXINT0 (falling edge trigger) is selected. |
| **PD** | 2 | rwh | **Power-down Enable Bit**<br>Setting this bit will cause the chip to enter power-down mode. It is reset by wake-up circuit.<br>The PD bit is a protected bit. When the Protection Scheme (see **Chapter 3.4.4.1**) is activated, this bit cannot be written directly. |
| **SD** | 3 | rw | **Slow-down Enable Bit**<br>Setting this bit will cause the chip to enter slow-down mode. It is reset by the user.<br>The SD bit is a protected bit. When the Protection Scheme is activated, this bit cannot be written directly |
| **WKSEL** | 4 | rw | **Wake-up Reset Select Bit**<br>0 Wake-up without reset<br>1 Wake-up with reset |

| Field | Bits | Type | Description |
|---|---|---|---|
| **WKRS** | 5 | rwh | **Wake-up Indication Bit**<br>This bit can only be set by hardware and reset by software.<br>0 No wake-up occurred<br>1 Wake-up has occurred |

**PCON**
**Power Control Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **SMOD** | | **0** | | **GF1** | **GF0** | **0** | **IDLE** |
| rw | | r | | rw | rw | r | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **IDLE** | 0 | rw | **Idle Mode Enable**<br>0 Do not enter idle mode<br>1 Enter idle mode |
| **0** | 1, [6:4] | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

**MODPISEL**
**Peripheral Input Select Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **0** | **URRISH** | **JTAGTDIS** | **JTAGTCKS** | **EXINT2IS** | **EXINT1IS** | **EXINT0IS** | **URRIS** |
| r | rw | rw | rw | rw | rw | r | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **URRISH, URRIS** | 6, 0 | rw | **UART Receive Input Select**<br>00 UART Receiver Input RXD_0 is selected.<br>01 UART Receiver Input RXD_1 is selected.<br>10 UART Receiver Input RXD_2 is selected.<br>11 Reserved |

| Field | Bits | Type | Description |
|---|---|---|---|
| **EXINT0IS** | 1 | rw | **External Interrupt 0 Input Select**<br>0 External Interrupt Input EXINT0_0 is selected.<br>1 External Interrupt Input EXINT0_1 is selected. |
| **0** | 7 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

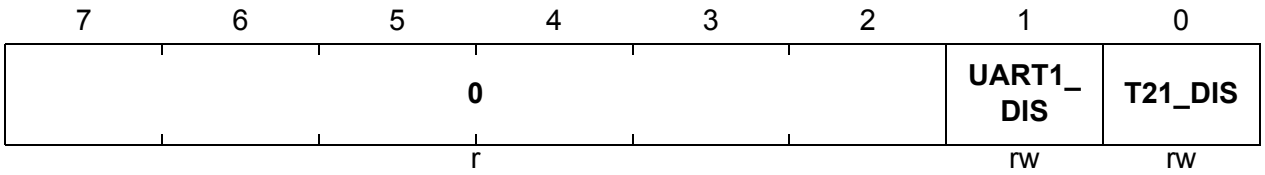
**PMCON1**
**Power Mode Control Register 1** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **0** | **CDC_DIS** | **CAN_DIS** | **MDU_DIS** | **T2CCU_DIS** | **CCU_DIS** | **SSC_DIS** | **ADC_DIS** |
| r | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **ADC_DIS** | 0 | rw | **ADC Disable Request. Active high**<br>0 ADC is in normal operation (default).<br>1 ADC is disabled. |
| **SSC_DIS** | 1 | rw | **SSC Disable Request. Active high**<br>0 SSC is in normal operation (default).<br>1 SSC is disabled. |
| **CCU_DIS** | 2 | rw | **CCU Disable Request. Active high**<br>0 CCU is in normal operation (default).<br>1 CCU is disabled. |
| **T2CCU_DIS** | 3 | rw | **T2CCU Disable Request. Active high**<br>0 T2CCU is in normal operation (default).<br>1 T2CCU is disabled. |
| **MDU_DIS** | 4 | rw | **MDU Disable Request. Active high**<br>0 MDU is in normal operation (default).<br>1 MDU is disabled. |
| **CAN_DIS** | 5 | rw | **CAN Disable Request. Active high**<br>0 CAN is in normal operation (default).<br>1 CAN is disabled. |
| **CDC_DIS** | 6 | rw | **CORDIC Disable Request. Active High**<br>0 CORDIC is in normal operation (default).<br>1 CORDIC is disabled. |

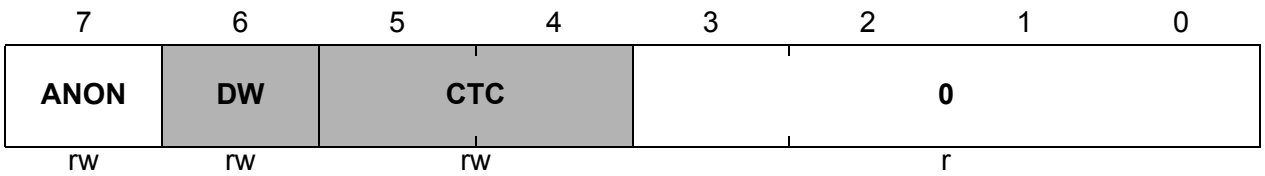
| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | 7 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

**PMCON2**
**Power Mode Control Register 2** **Reset Value: $00_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **T21_DIS** | 0 | rw | **Timer 21 Disable Request. Active high**<br>0 Timer 21 is in normal operation (default).<br>1 Timer 21 is disabled. |
| **UART1_DIS** | 1 | rw | **UART1 Disable Request. Active high**<br>0 UART1 is in normal operation (default).<br>1 UART1 is disabled. |
| **0** | [7:2] | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

**ADC_GLOBCTR**
**Global Control Register** **Reset Value: $00_H$**



| Field | Bits | Type | Description |
|---|---|---|---|
| **ANON** | 7 | rw | **Analog Part Switched On**<br>This bit enables the analog part of the ADC module and defines its operation mode.<br>0 The analog part is switched off and conversions are not possible. To achieve minimal power consumption, the internal analog circuitry is in its power-down state and the generation of $f_{ADCI}$ is stopped.<br>1 The analog part of the ADC module is switched on and conversions are possible. The automatic power-down capability of the analog part is disabled. |
| **0** | 3:0 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

**OSC_CON**
**OSC Control Register** **Reset Value: 0X00 1000$_B$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **PLLRDRES** | **PLLBYP** | **PLLPD** | **0** | **XPD** | **OSCSS** | **EORDRES** | **EXTOSCR** |
| rwh | rwh | rw | r | rw | rwh | rwh | rh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **XPD** | 3 | rw | **XTAL Power Down Control**<br>0 XTAL is not powered down.<br>1 XTAL is powered down.<br>The XPD bit is a protected bit. When the Protection Scheme is activated, this bit cannot be written directly |
| **PLLPD** | 5 | rw | **PLL Power Down Control**<br>0 PLL is not powered down.<br>1 PLL is powered down.<br>The PLLPD bit is a protected bit. When the Protection Scheme is activated, this bit cannot be written directly |
| **0** | 4 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

# 9 Watchdog Timer

The Watchdog Timer (WDT) provides a highly reliable and secure way to detect and recover from software or hardware failures. The WDT is reset at a regular interval that is predefined by the user. The CPU must service the WDT within this interval to prevent the WDT from causing an XC878 system reset. Hence, routine service of the WDT confirms that the system is functioning properly. This ensures that an accidental malfunction of the XC878 will be aborted in a user-specified time period.

The WDT is by default disabled.

In debug mode, the WDT is default suspended and stops counting (its debug suspend bit is default set i.e., MODSUSP.WDTSUSP = 1). Therefore during debugging, there is no need to refresh the WDT.

**Features**

- 16-bit Watchdog Timer
- Programmable reload value for upper 8 bits of timer
- Programmable window boundary
- Selectable input frequency of $f_{PCLK}/2$ or $f_{PCLK}/128$

## 9.1 Functional Description

The Watchdog Timer is a 16-bit timer, which is incremented by a count rate of $f_{PCLK}/2$ or $f_{PCLK}/128$. This 16-bit timer is realized as two concatenated 8-bit timers. The upper 8 bits of the Watchdog Timer can be preset to a user-programmable value via a watchdog service access in order to vary the watchdog expire time. The lower 8 bits are reset on each service access. **Figure 9-1** shows the block diagram of the watchdog timer unit.

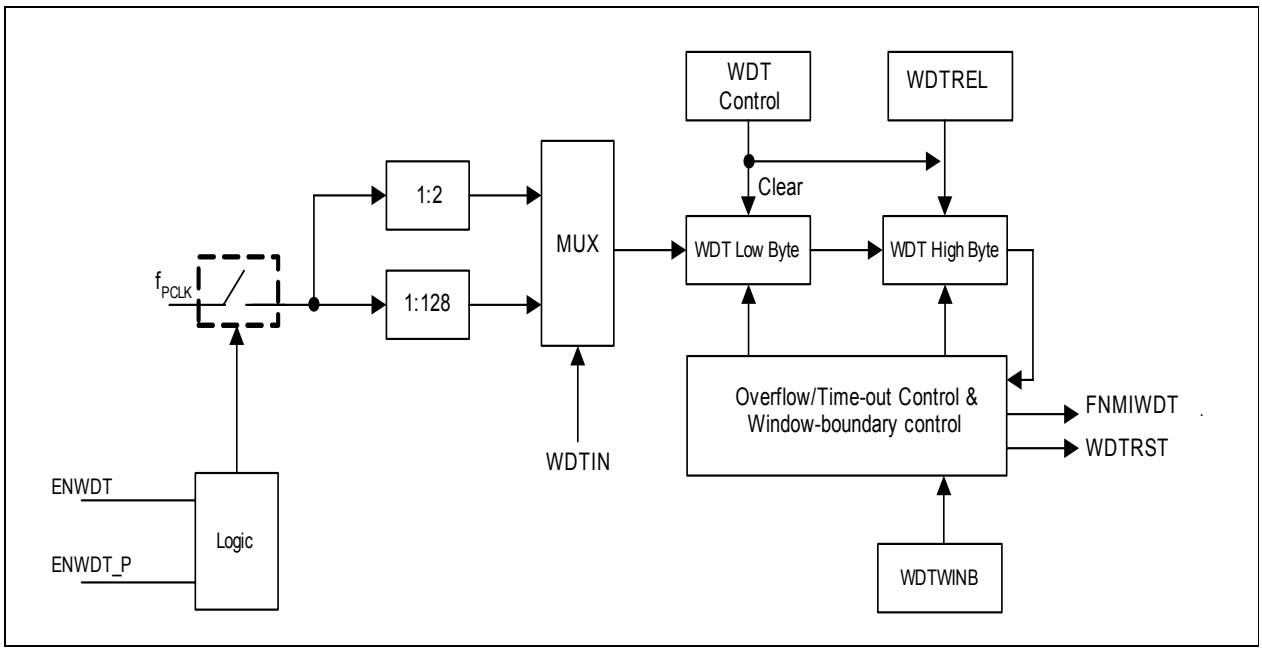



**Figure 9-1 WDT Block Diagram**

If the WDT is enabled by setting WDTEN to 1, the timer is set to a user-defined start value and begins counting up. It must be serviced before the counter overflows. Servicing is performed through refresh operation (setting bit WDTRS to 1). This reloads the timer with the start value, and normal operation continues.

If the WDT is not serviced before the timer overflows, a system malfunction is assumed and normal mode is terminated. A WDT NMI request (FNMIWDT) is then asserted and prewarning is entered. The prewarning lasts for $30_H$ count. During the prewarning period, refreshing of the WDT is ignored and the WDT cannot be disabled. A reset (WDTRST) of the XC878 is imminent and can no longer be avoided. The occurrence of a WDT reset is indicated by the bit WDTRST, which is set to 1 once hardware detects the assertion of the signal WDTRST. If refresh happens at the same time an overflow occurs, WDT will not go into prewarning period

The WDT must be serviced periodically so that its count value will not overflow. Servicing the WDT clears the low byte and reloads the high byte with the preset value in bit field WDTREL. Servicing the WDT also clears the bit WDTRS.

The WDT has a “programmable window boundary”, which disallows any refresh during the WDT’s count-up. A refresh during this window-boundary constitutes an invalid

access to the WDT and causes the WDT to activate WDTRST, although no NMI request is generated in this instance. The window boundary is from $0000_H$ to the value obtained from the concatenation of WDTWINB and $00_H$. This feature can be enabled by WINBEN.

After being serviced, the WDT continues counting up from the value (<WDTREL> * $2^8$). The time period for an overflow of the WDT is programmable in two ways:

- The input frequency to the WDT can be selected via bit WDTIN in register WDTCON to be either $f_{PCLK}/2$ or $f_{PCLK}/128$.
- The reload value WDTREL for the high byte of WDT can be programmed in register WDTREL.

The period $P_{WDT}$ between servicing the WDT and the next overflow can be determined by the following formula:

$$P_{WDT} = \frac{2^{(1+<WDTIN>*6)} * (2^{16} - WDTREL * 2^8)}{f_{PCLK}} \quad (9.1)$$

If the Window-Boundary Refresh feature of the WDT is enabled, the period $P_{WDT}$ between servicing the WDT and the next overflow is shortened if WDTWINB is greater than WDTREL. See also **Figure 9-2**. This period can be calculated by the same formula by replacing WDTREL with WDTWINB. In order for this feature to be useful, WDTWINB cannot be smaller than WDTREL.

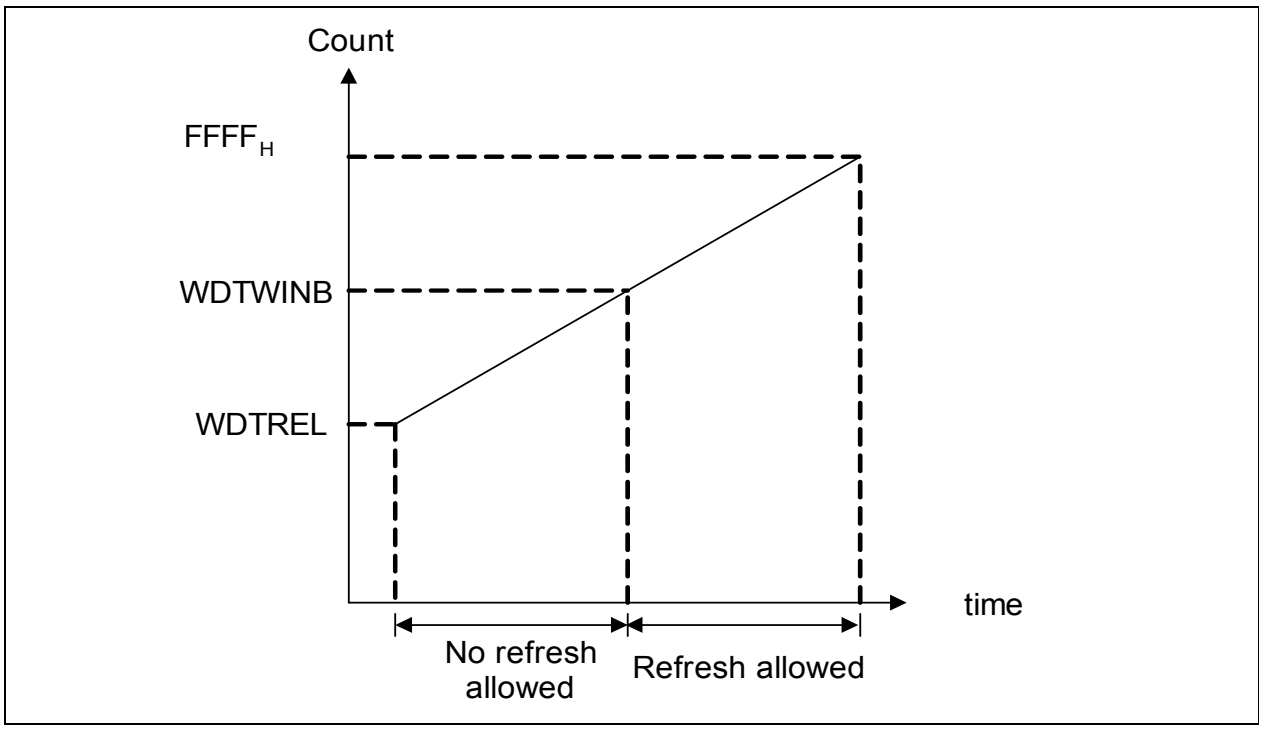



**Figure 9-2 WDT Timing Diagram**

Table 9-1 lists the possible ranges for the watchdog time which can be achieved using a certain module clock. Some numbers are rounded to 3 significant digits.

**Table 9-1 Watchdog Time Ranges**

| Reload value in WDTREL | Prescaler for $f_{WDT}$ | | | | | |
|---|---|---|---|---|---|---|
| | 2 (WDTIN = 0) | | | 128 (WDTIN = 1) | | |
| | 24 MHz | 16 MHz | 12 MHz | 24 MHz | 16 MHz | 12 MHz |
| $FF_H$ | 21.3 µs | 32.0 µs | 42.67 µs | 1.37 ms | 2.05 ms | 2.73 ms |
| $7F_H$ | 2.75 ms | 4.13 ms | 5.5 ms | 176 ms | 264 ms | 352 ms |
| $00_H$ | 5.46 ms | 8.19 ms | 10.92 ms | 350 ms | 524 ms | 699 ms |

*Note: For safety reasons, the user is advised to rewrite WDTCON each time before the WDT is serviced.*

### 9.1.1 Module Suspend Control

The WDT is by default suspended on entering debug mode. The WDT can be allowed to run in debug mode by clearing the bit WDTSUSP in SFR MODSUSP to 0.

**MODSUSP**
**Module Suspend Control Register** **Reset Value: $01_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | | CCTSUSP | T21SUSP | T2SUSP | T13SUSP | T12SUSP | WDTSUSP |
| r | | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **WDTSUSP** | 0 | rw | **WDT Debug Suspend Bit**<br>0 WDT will not be suspended.<br>1 WDT will be suspended. |
| **0** | 7:6 | r | **Reservedl**<br>Returns 0 if read; should be written with 0. |

## 9.2 Register Map

Five SFRs control the operations of the WDT. They can be accessed from the mapped SFR area.

Table 9-2 lists the addresses of these SFRs.

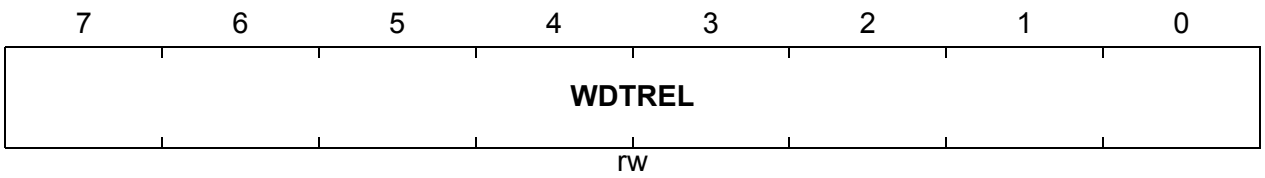
**Table 9-2 SFR Address List**

| Address | Register |
|---|---|
| $BB_H$ | WDTCON |
| $BC_H$ | WDTREL |
| $BD_H$ | WDTWINB |
| $BE_H$ | WDTL |
| $BF_H$ | WDTH |

## 9.3 Register Description

The Watchdog Timer Current Count Value is contained in the Watchdog Timer Register WDTH and WDTL, which are non-bitaddressable read-only register. The operation of the WDT is controlled by its bitaddressable WDT Control Register WDTCON. This register also selects the input clock prescaling factor. The register WDTREL specifies the reload value for the high byte of the timer.

**WDTREL**
**Watchdog Timer Reload Register** **Reset Value: $00_H$**



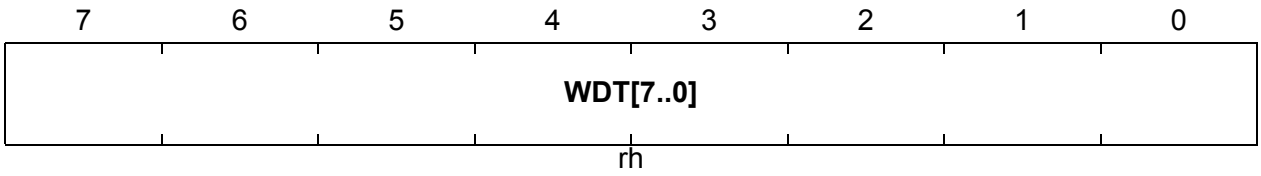

| Field | Bits | Type | Description |
|---|---|---|---|
| **WDTREL** | 7:0 | rw | Watchdog Timer Reload Value (for the high byte of WDT) |

**WDTCON**
**Watchdog Timer Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **0** | | **WINBEN** | **WDTPR** | **0** | **WDTEN** | **WDTRS** | **WDTIN** |
| r | | rw | rh | r | rw | rwh | rw |

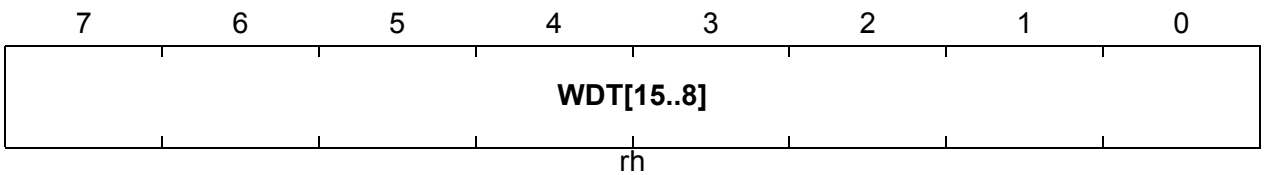
| Field | Bits | Type | Description |
|---|---|---|---|
| **WDTIN** | 0 | rw | **Watchdog Timer Input Frequency Selection**<br>0 Input frequency is $f_{PCLK}/2$<br>1 Input frequency is $f_{PCLK}/128$ |
| **WDTRS** | 1 | rwh | **WDT Refresh Start.**<br>Active high. Set to start refresh operation on the watchdog timer. Cleared by hardware automatically. |
| **WDTEN** | 2 | rw | **WDT Enable.**<br>WDTEN is a protected bit. If the Protection Scheme is activated then this bit cannot be written directly. See protection Scheme for more details.<br>0 WDT is disabled.<br>1 WDT is enabled. |
| **WDTPR** | 4 | rh | **Watchdog Prewarning Mode Flag**<br>This bit is set to 1 when a Watchdog error is detected. The Watchdog Timer has issued an NMI trap and is in Prewarning Mode. A reset of the chip occurs after the prewarning period has expired.<br>0 Normal mode (default after reset)<br>1 The Watchdog is operating in Prewarning Mode |
| **WINBEN** | 5 | rw | **Watchdog Window-Boundary Enable.**<br>0 Watchdog Window-Boundary feature is disabled (default).<br>1 Watchdog Window-Boundary feature is enabled. |
| **0** | 3, [7:6] | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

## WDTL
**Watchdog Timer, Low Byte** **Reset Value: $00_H$**



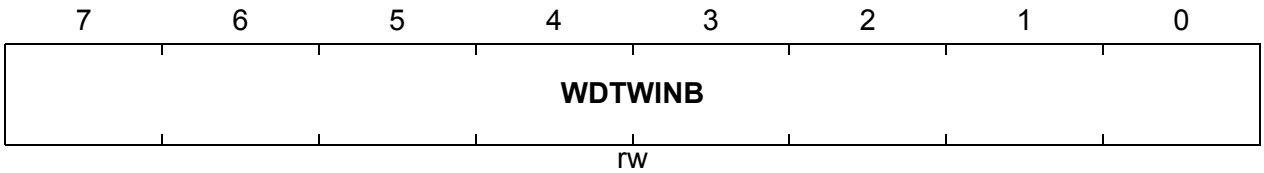

| Field | Bits | Type | Description |
|---|---|---|---|
| **WDT[7..0]** | 7:0 | rh | **Watchdog Timer Current Value** |

## WDTH
**Watchdog Timer, High Byte** **Reset Value: $00_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **WDT[15..8]** | 7:0 | rh | **Watchdog Timer Current Value** |

**WDTWINB**
**Watchdog Window-Boundary Count** **Reset Value: $00_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **WDTWINB** | 7:0 | rw | **Watchdog Window-Boundary Count Value**<br>This value is programmble. Within this Window-Boundary range from 0000H to (WDTWINB,00H), the WDT cannot do a Refresh, else it will cause a WDTRST to be asserted.<br>WDTWINB is matched to WDTH. |

**PMCON0**
**Power Mode Control Register 0** **Reset Value: $00_H$ [1)]**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **VDDP WARN** | **WDTRST** | **WKRS** | **WKSEL** | **SD** | **PD** | **WS** | |
| rh | rwh | rwh | rw | rw | rwh | rw | |

[1)] The reset value for watchdog timer reset is $40_H$ and the reset value for power-down wake-up reset is $20_H$.

| Field | Bits | Type | Description |
|---|---|---|---|
| **WDTRST** | 6 | rwh | **Watchdog Timer Reset Indication Bit**<br>0 No WDT reset has occurred.<br>1 WDT reset has occurred. |

# 10 Multiplication/Division Unit

The Multiplication/Division Unit (MDU) provides fast 16-bit multiplication, 16-bit and 32-bit division as well as shift and normalize features. It has been integrated to support the XC878 Core in real-time control applications, which require fast mathematical computations.

The MDU uses a total of 14 registers; 12 registers for data manipulation, one register to control the operation of MDU and one register for storing the status flags. These registers are memory mapped as special function registers like any other registers for peripheral control. The MDU operates concurrently with and independent of the CPU.

### Features

- Fast signed/unsigned 16-bit multiplication
- Fast signed/unsigned 32-bit divide by 16-bit and 16-bit divide by 16-bit operations
- 32-bit unsigned normalize operation
- 32-bit arithmetic/logical shift operations
- 2 selectable operating frequency: peripheral clock or double the peripheral clock.

Table 10-1 specifies the number of clock cycles used for calculation in various operations.

**Table 10-1 MDU Operation Characteristics**

| Operation | Result | Remainder | No. of Clock Cycles used for calculation |
|---|---|---|---|
| Signed 32-bit/16-bit | 32-bit | 16-bit | 33 |
| Signed 16-bit/16bit | 16-bit | 16-bit | 17 |
| Signed 16-bit x 16-bit | 32-bit | - | 16 |
| Unsigned 32-bit/16-bit | 32-bit | 16-bit | 32 |
| Unsigned 16-bit/16-bit | 16-bit | 16-bit | 16 |
| Unsigned 16-bit x 16-bit | 32-bit | - | 16 |
| 32-bit normalize | - | - | No. of shifts + 1 (Max. 32) |
| 32-bit shift L/R | - | - | No. of shifts + 1 (Max. 32) |

## 10.1 Functional Description

The MDU can be regarded as a special coprocessor for multiplication, division, normalization and shift. Its operation can be divided into three phases (see **Figure 10-1**):

**Phase one: Load MDx registers**

In this phase, the operands are loaded into the MDU Operand (MDx) registers by the CPU.

The type of calculation the MDU must perform is selected by writing a 4-bit opcode that represents the required operation into the bit field MDUCON.OPCODE.

**Phase two: Execute calculation**

This phase commences only when the start bit MDUCON.START is set, which in turn sets the busy flag. The start bit is automatically cleared in the next cycle.

During this phase, the MDU works on its own, in parallel with the CPU. The result of the calculation is made available in the MDU Result (MRx) registers at the end of this phase.

**Phase three: Read result from the MRx registers**

In this final phase, the result is fetched from the MRx registers by the CPU. The MRx registers will be overwritten at the start of the next calculation phase.

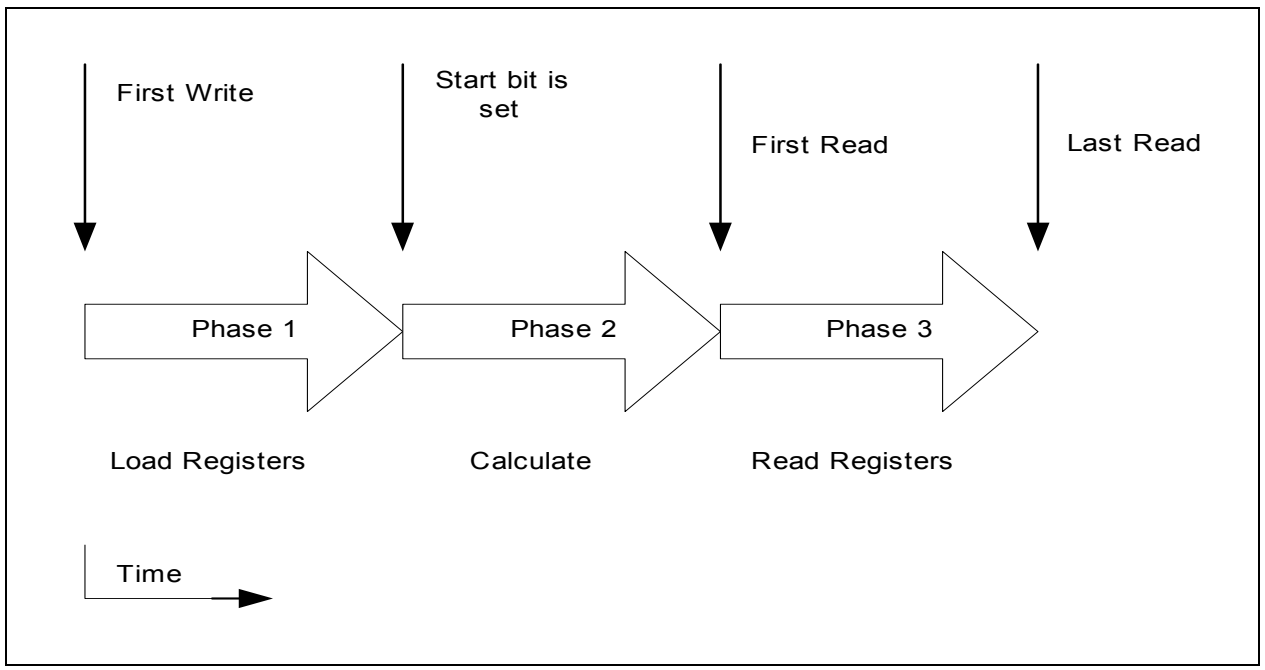



**Figure 10-1 Operating phases of the MDU**

### 10.1.1 Division Operation

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

If q = D div d

and r = D mod d

then D = q * d + r

and | r | < | d |

where “D” is the dividend, “d” is the divisor, “q” is the quotient and “r” is the remainder.

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

### 10.1.2 Normalize

The MDU supports up to 32-bit unsigned normalize.

Normalizing is done on an unsigned integer variable stored in MD0 (least significant byte) to MD3 (most significant byte). This feature is mainly meant to support applications where floating point arithmetic is used. During normalization, all leading zeros of an unsigned integer variable in registers MD0 to MD3 are removed by shift left operations. The whole operation is completed when the MSB (most significant bit) contains a 1.

After normalizing, bit field MR4.SCTR contains the number of shift left operations that were done. This number may be used later as an exponent. The maximum number of shifts in a normalize operation is 31 (= $2^5$ - 1).

### 10.1.3 Shift

The MDU implements both logical and arithmetic shifts to support up to 32-bit unsigned and signed shift operations.

During logical shift, zeros are shifted in from the left end of register MD3 or right end of register MD0. An arithmetic left shift is identical to a logical left shift, but during arithmetic right shifts, signed bits are shifted in from the left end of register MD3. For example, if the data $0101_B$ and $1010_B$ are to undergo an arithmetic shift right, the results obtained will be $0010_B$ and $1101_B$, respectively.

For any shift operation, register bit MD4.SLR specifies the shift direction, and MD4.SCTR the shift count.

*Note: The MDU does not detect overflows due to an arithmetic shift left operation. User must always ensure that the result of an arithmetic shift left is within the boundaries of MDU.*

### 10.1.4 Busy Flag

A busy flag is provided to indicate the MDU is still performing a calculation. The flag MDUSTAT.BSY is set at the start of a calculation and cleared after the calculation is completed at the end of phase two. It is also cleared when the error flag is set.

If a second operation needs to be executed, the status of the busy flag will be polled first and only when it is not set, can the start bit be written and the second operation begin. Any unauthorized write to the start bit while the busy flag is still set will be ignored.

### 10.1.5 Error Detection

The error flag MDUSTAT.IERR is provided to indicate that an error has occurred while performing a calculation. The flag is set by hardware when one of these occurs:

- Division by zero
- Writing of reserved opcodes to MDUCON register

The setting of the error flag causes the current operation to be aborted and triggers an interrupt (see **Section 10.2** below). A division by zero error does not set the error flag immediately but rather, at the end of calculation phase for a division operation. An opcode error is detected upon setting MDUCON.START to 1. Errors due to division by zero lead to the loading of a saturated value into the MRx registers.

*Note: The accuracy of any result obtained when the error flag is set is not guaranteed by MDU and hence the result should not be used.*

## 10.2 Interrupt Generation

The interrupt structure of the MDU is shown in **Figure 10-2**. There are two possible interrupt events in the MDU, and each event sets one of the two interrupt flags. The interrupt flags is reset by software by writing 0 to it.

At the end of phase two, the interrupt flag MDUSTAT.IRDY is set by hardware to indicate the successful completion of a calculation. The results can then be obtained from the MRx registers. The interrupt line INT_O0 is mapped directly to this interrupt source.

An interrupt can also be triggered when an error occurs during calculation. This is indicated by the setting of the interrupt flag MDUSTAT.IERR. In the event of a division by zero error, MDUSTAT.IERR is set only at the end of the calculation phase. Once the MDUSTAT.IERR is set, any ongoing calculation will be aborted. For a division by zero error, a saturated value is then loaded into the MRx registers. The bit MDUCON.IR determines the interrupt line to be mapped to this interrupt source.

An interrupt is only generated when interrupt enable bit MDUCON.IE is 1 and the corresponding interrupt event occurs. An interrupt request signal is always asserted positively for 2 clocks.

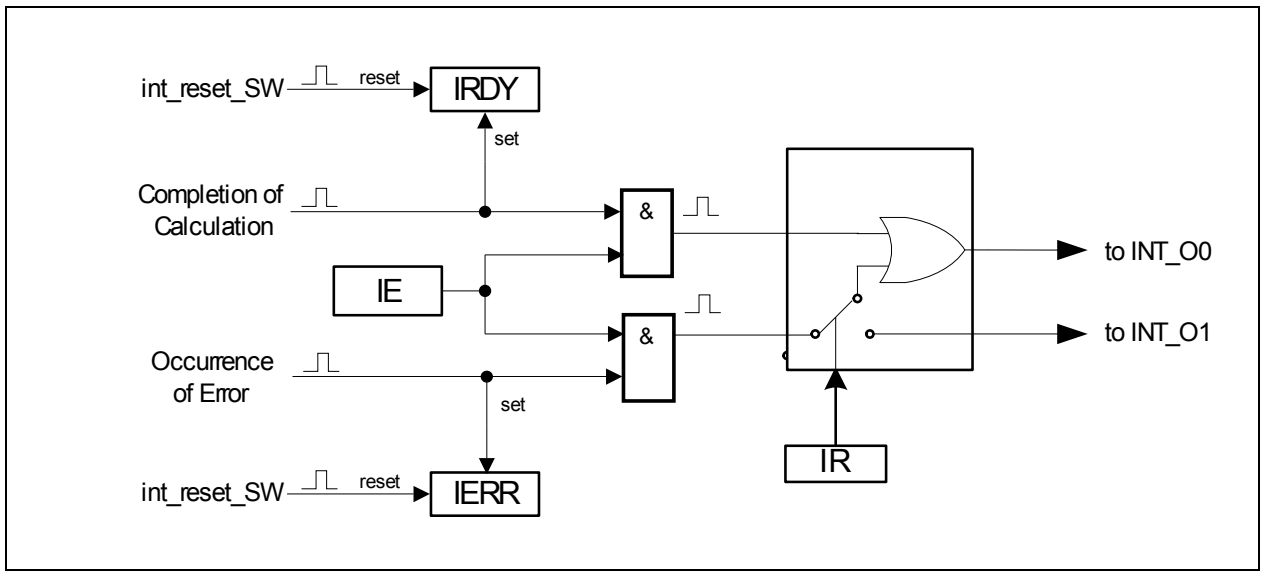



**Figure 10-2 Interrupt Generation**

## 10.3 Low Power Mode

If the MDU functionality is not required at all, it can be completely disabled by gating off its clock input for maximal power reduction. This is done by setting bit MDU_DIS in register PMCON1 as described below. Refer to **Chapter 8.1.4** for details on peripheral clock management.

**PMCON1**
**Power Mode Control Register 1** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **0** | **CDC_DIS** | **CAN_DIS** | **MDU_DIS** | **T2CCU _DIS** | **CCU_DIS** | **SSC_DIS** | **ADC_DIS** |
| r | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| MDU_DIS | 4 | rw | MDU Disable Request. Active high.<br>0 MDU is in normal operation (default).<br>1 Request to disable the MDU. |
| 0 | 7 | r | Reserved<br>Returns 0 if read; should be written with 0. |

## 10.4 Register Map

Table 10-2 lists the MDU registers with their addresses:

**Table 10-2 MDU Registers**

| SFR | Address | Name |
|---|---|---|
| MDUCON | $B1_H$ (mapped) | MDU Control Register |
| MDUSTAT | $B0_H$ (mapped) | MDU Status Register |
| MD0/MR0 | $B2_H$ (mapped) | MDU Data/Result Register 0 |
| MD1/MR1 | $B3_H$ (mapped) | MDU Data/Result Register 1 |
| MD2/MR2 | $B4_H$ (mapped) | MDU Data/Result Register 2 |
| MD3/MR3 | $B5_H$ (mapped) | MDU Data/Result Register 3 |
| MD4/MR4 | $B6_H$ (mapped) | MDU Data/Result Register 4 |
| MD5/MR5 | $B7_H$ (mapped) | MDU Data/Result Register 5 |

The MDx and MRx registers share the same address. However, since MRx registers should never be written to, any write operation to one of these addresses will be interpreted as a write to an MDx register.

In the event of a read operation, an additional bit MDUCON.RSEL is needed to select which set of registers, MDx or MRx, the read operation must be directed to. By default, the MRx registers are read.

## 10.5 Register Description

The 14 SFRs of the MDU consist of a control register MDUCON, a status register MDUSTAT and 2 sets of data registers, MD0 to MD5 (which contain the operands) and MR0 to MR5 (which contain the results).

Depending on the type of operation, the individual MDx and MRx registers assume specific roles as summarized in **Table 10-3** and **Table 10-4**. For example, in a multiplication operation, the low byte of the 16-bit multiplicator must be written to register MD4 and the high byte to MD5.

**Table 10-3 MDx Registers**

| Register | Roles of registers in operations | | | |
|---|---|---|---|---|
| | **16-bit Multiplication** | **32/16-bit Division** | **16/16-bit Division** | **Normalize and Shift** |
| MD0 | M'andL | D'endL | D'endL | OperandL |
| MD1 | M'andH | D'end | D'endH | Operand |
| MD2 | - | D'end | - | Operand |
| MD3 | - | D'endH | - | OperandH |
| MD4 | M'orL | D'orL | D'orL | Control |
| MD5 | M'orH | D'orH | D'orH | - |

**Table 10-4 MRx Registers**

| Register | Roles of registers in operations | | | |
|---|---|---|---|---|
| | **16-bit Multiplication** | **32/16-bit Division** | **16/16-bit Division** | **Normalize and Shift** |
| MR0 | PrL | QuoL | QuoL | ResultL |
| MR1 | Pr | Quo | QuoH | Result |
| MR2 | Pr | Quo | - | Result |
| MR3 | PrH | QuoH | - | ResultH |
| MR4 | M'orL | RemL | RemL | Control |
| MR5 | M'orH | RemH | RemH | - |

Abbreviations:

- D'end: Dividend, 1st operand of division

- D'or: Divisor, 2nd operand of division
- M'and: Multiplicand, 1st operand of multiplication
- M'or: Multiplicator, 2nd operand of multiplication
- Pr: Product, result of multiplication
- Rem: Remainder
- Quo: Quotient, result of division
- ...L: means that this byte is the least significant of the 16-bit or 32-bit operand
- ...H: means that this byte is the most significant of the 16-bit or 32-bit operand

The MDx registers are built with shadow registers, which are latched with data from the actual registers at the start of a calculation. This frees up the MDx registers to be written with the next set of operands while the current calculation is ongoing.

MDx and MRx registers not used in an operation are undefined to the user. For normalize and shift operations, the registers MD4 and MR4 are used as shift input and output control registers to specify the shift direction and store the number of shifts performed.

### 10.5.1 Operand and Result Registers

The MDx and MRx registers are used to store the operands and results of a calculation. MD4 and MR4 are also used as input and output control registers for shift and normalize operations.

**MDx (x = 0 - 5)**
**MDU Operand Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| DATA | | | | | | | |
| rw | | | | | | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **DATA** | 7:0 | rw | **Operand Value**<br>See **Table 10-3**. |

**MRx (x = 0 - 5)**
**MDU Result Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| DATA | | | | | | | |
| rw | | | | | | | |

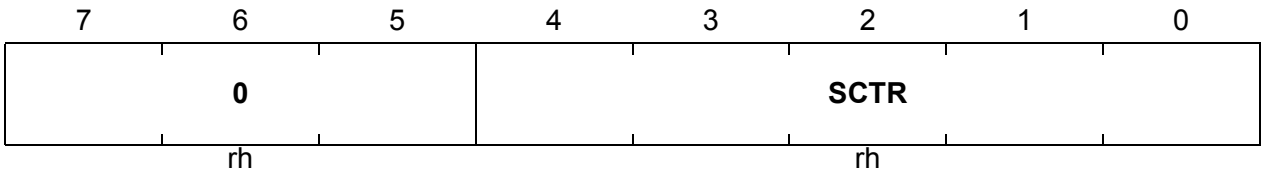
| Field | Bits | Type | Description |
|---|---|---|---|
| **DATA** | 7:0 | rh | **Result Value**<br>See **Table 10-4**. |

**MD4**
**Shift Input Control Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | | SLR | SCTR | | | | |
| rw | | rw | rw | | | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **SCTR** | 4:0 | rw | **Shift Counter**<br>The count written to SCTR determines the number of shifts to be performed during a shift operation. |
| **SLR** | 5 | rw | **Shift Direction**<br>0 Selects shift left operation.<br>1 Selects shift right operation. |
| **0** | 7:6 | rw | **Reserved**<br>Should be written with 0. Returns undefined data if read. |

**MR4**
**Shift Output Control Register** **Reset Value: $00_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **SCTR** | 4:0 | rh | **Shift Counter**<br>After a normalize operation, SCTR contains the number of normalizing shifts performed. |
| **0** | 7:5 | rh | **Reserved**<br>Returns undefined data if read. |

### 10.5.2 Control Register

Register MDUCON contains control bits that select and start the type of operation to be performed.

**MDUCON**
**MDU Control Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| IE | IR | RSEL | START | OPCODE | | | |
| rw | rw | rw | rwh | rw | | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **OPCODE** | 3:0 | rw | **Operation Code**<br>0000 Unsigned 16-bit Multiplication<br>0001 Unsigned 16-bit/16-bit Division<br>0010 Unsigned 32-bit/16-bit Division<br>0011 32-bit Logical Shift L/R<br>0100 Signed 16-bit Multiplication<br>0101 Signed 16-bit/16-bit Division<br>0110 Signed 32-bit/16-bit Division<br>0111 32-bit Arithmetic Shift L/R<br>1000 32-bit Normalize<br>Others: Reserved |
| **START** | 4 | rwh | **Start Bit**<br>The bit START is set by software and reset by hardware.<br>0 Operation is not started.<br>1 Operation is started. |
| **RSEL** | 5 | rw | **Read Select**<br>0 Read the MRx registers.<br>1 Read the MDx registers. |
| **IR** | 6 | rw | **Interrupt Routing**<br>0 The two interrupt sources have their own dedicated interrupt lines.<br>1 The two interrupt sources share one interrupt line INT_O0. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **IE** | 7 | rw | **Interrupt Enable**<br>0 The interrupt is disabled.<br>1 The interrupt is enabled. |

*Note: Write access to MDUCON is not allowed when the busy flag MDUSTAT.BSY is set during the calculation phase.*

*Note: Writing reserved opcode values to MDUCON results in an error condition when MDUCON.START bit is set to 1.*

### 10.5.3 Status Register

Register MDUSTAT contains the status flags of the MDU.

**MDUSTAT**
**MDU Status Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **0** | | | | | **BSY** | **IERR** | **IRDY** |
| r | | | | | rh | rwh | rwh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **IRDY** | 0 | rwh | **Interrupt on Result Ready**<br>The bit IRDY is set by hardware and reset by software.<br>0 No interrupt is triggered at the end of a successful operation.<br>1 An interrupt is triggered at the end of a successful operation. |
| **IERR** | 1 | rwh | **Interrupt on Error**<br>The bit IERR is set by hardware and reset by software.<br>0 No interrupt is triggered with the occurrence of an error.<br>1 An interrupt is triggered with the occurrence of an error. |
| **BSY** | 2 | rh | **Busy Bit**<br>0 The MDU is not running any calculation.<br>1 The MDU is still running a calculation. |
| **0** | 7:3 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

# 11 CORDIC Coprocessor

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

The CORDIC algorithm is an iterative process where truncation errors are inherent. Higher accuracy is achieved in the CORDIC Coprocessor with 16 iterations per calculation and kernel data width of at least 20 bits. The main advantage of using this algorithm is the low hardware costs involved compared to other complex algorithms.

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

$$x_{i+1} = x_i - m \cdot d_i \cdot y_i \cdot 2^{-i} \quad (11.1)$$

$$y_{i+1} = y_i + d_i \cdot x_i \cdot 2^{-i} \quad (11.2)$$

$$z_{i+1} = z_i - d_i \cdot e_i \quad (11.3)$$

where

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

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

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

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

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

## 11.1 Features

- Modes of operation
  - Supports all CORDIC operating modes for solving circular (trigonometric), linear (multiply-add, divide-add) and hyperbolic functions
  - Integrated look-up tables (LUTs) for all operating modes
- Circular vectoring mode: Extended support for values of initial X and Y data up to full range of $[-2^{15},(2^{15}-1)]$ for solving angle and magnitude
- Circular rotation mode: Extended support for values of initial Z data up to full range of $[-2^{15},(2^{15}-1)]$, representing angles in the range $[-\pi,((2^{15}-1)/2^{15})\pi]$ for solving trigonometry
- 2 selectable operating frequency: peripheral clock or double the peripheral clock.
- Gated clock input to support disabling of module
- 16-bit accessible data width
  - 24-bit kernel data width plus 2 overflow bits for X and Y each
  - 20-bit kernel data width plus 1 overflow bit for Z
  - With KEEP bit to retain the last value in the kernel register for a new calculation
- 16 iterations per calculation: Approximately 41 clock-cycles or less, from set of start (ST) bit to set of end-of-calculation flag, excluding time taken for write and read access of data bytes.
- Twos complement data processing
  - Only exception: X result data with user selectable option for unsigned result
- X and Y data generally accepted as integer or rational number; X and Y must be of the same data form
- Entries of LUTs are 20-bit signed integers
  - Entries of atan and atanh LUTs are integer representations (S19) of angles with the scaling such that $[-2^{15},(2^{15}-1)]$ represents the range $[-\pi,((2^{15}-1)/2^{15})\pi]$
  - Accessible Z result data for circular and hyperbolic functions is integer in data form of S15
- Emulated LUT for linear function
  - Data form is 1 integer bit and 15-bit fractional part (1.15)
  - Accessible Z result data for linear function is rational number with fixed data form of S4.11 (signed 4Q16)
- Truncation Error
  - The result of a CORDIC calculation may return an approximation due to truncation of LSBs
  - Good accuracy of the CORDIC calculated result data, especially in circular mode
- Interrupt
  - On completion of a calculation
  - Interrupt enabling and corresponding flag

## 11.2 Functional Description

The following sections describe the function of the CORDIC Coprocessor.

### 11.2.1 Operation of the CORDIC Coprocessor

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

Initialization of the kernel data register is enabled by clearing respective KEEP bits of the CD_STATC. If ST_MODE = 1, writing 1 to bit ST starts a new calculation. Otherwise, by default where ST_MODE = 0, a new calculation starts after a write access to register CD_CORDXL. Each calculation involves a fixed number of 16 iterations. Bit BSY is set while a calculation is in progress to indicate busy status. It is cleared by hardware at the end of a calculation.

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

The data registers CD_CORDxL and CD_CORDxH function as shadow registers which can be written to without affecting an ongoing calculation. Values are transferred to the kernel data registers only on valid setting of bit ST, or if ST_MODE = 0, after write access to X low byte CD_CORDXL (provided KEEP bit of corresponding data is not set). The result data must be read at the end of calculation (BSY no longer active) before starting a new calculation. The result data is read directly from the kernel data registers with bit CD_STATC.DMAP = 0. The kernel data is placed directly on the bus so the data registers which function as shadow registers are not overwritten during this operation. Alternatively, the shadow data registers are read (DMAP = 1), although this would be merely reading back the user-initialized initial data.

At the end of each calculation, CD_STATC.BSY returns to 0, the End-of-Calculation (EOC) flag is set and the interrupt request signal will be activated if interrupt is enabled by INT_EN = 1. The result data in X, Y and Z are internally checked, and in case of data overflow, the ERROR bit is set. This bit is automatically cleared on the start of a new calculation, or when read.

On starting a new calculation, the kernel data registers can no longer be expected to hold the result of the previous calculation. The kernel data registers always hold either the initial value or the (intermediate) result of the last CORDIC iteration.

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

### 11.2.2 Interrupt

The End-of-Calculation (EOC) is the only interrupt source of the CORDIC Coprocessor. If interrupt is enabled by CD_STATC.INT_EN = 1, an interrupt request signal is activated at the end of CORDIC calculation and also indicated by the CD_STATC.EOC flag. If not cleared by software, the EOC flag remains set until cleared by hardware when a read access is performed to the low byte of Z result data (DMAP = 0).

During EOC data processing, a check must be made to ensure that the ERROR flag is not set (indicates data overflow has occurred).

### 11.2.3 Normalized Result Data

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

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

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

For **linear** function, there is no additional processing of the CORDIC calculated Z data, as such it is taken directly as the result data. The accessible Z result data is a real number expressed as signed 4Q16.

For **circular** and **hyperbolic** functions, the accessible Z result data is a normalized integer value, angles in the range $[-\pi,((2^{15}-1)/2^{15})\pi]$ are represented by $[-2^{15},(2^{15}-1)]$. The CORDIC Coprocessor expects Z data to be interpreted with this scaling:

$$\text{Input Z Initial Data} = \text{Real Z Initial Value (in radians)} \times \frac{32768}{\pi}$$

$$\text{Real Z Result Value (in radians)} = \text{Z Result Data} \times \frac{\pi}{32768}$$

The CORDIC calculated data includes an inherent gain factor $K$ resulting from the rotation or vectoring. The value $K$ is different for each CORDIC function, as shown in **Table 11-1**.

**Table 11-1 CORDIC Function Inherent Gain Factor for Result Data**

| Function | Approximated Gain K |
|---|---|
| Circular | 1.64676 |
| Hyperbolic | 0.828 |
| Linear | 1 |

### 11.2.4 CORDIC Coprocessor Operating Modes

Table 11-2 gives an overview of the CORDIC Coprocessor operating modes. In this table, $X$, $Y$ and $Z$ represent the initial data, while $X_{final}$, $Y_{final}$ and $Z_{final}$ represent the final result data when all processing is complete and BSY is no longer active.

The CORDIC equations are:

$$x_{i+1} = x_i - m \cdot d_i \cdot y_i \cdot 2^{-i} \quad (11.4)$$

$$y_{i+1} = y_i + d_i \cdot x_i \cdot 2^{-i} \quad (11.5)$$

$$z_{i+1} = z_i - d_i \cdot e_i \quad (11.6)$$

**Table 11-2 CORDIC Coprocessor Operating Modes and Corresponding Result Data**

| Function | Rotation Mode | Vectoring Mode |
|---|---|---|
| | $d_i$ = sign ($z_i$), $z_i \rightarrow 0$ | $d_i$ = -sign ($y_i$), $y_i \rightarrow 0$ |
| **Circular**<br>$m$ = 1<br>$e_i$ = atan($2^{-i}$) | $X_{final}$ = K[$X$ cos($Z$) - $Y$ sin($Z$)] / MPS<br>$Y_{final}$ = K[$Y$ cos($Z$) + $X$ sin($Z$)] / MPS<br>$Z_{final}$ = 0<br>where K ≈ 1.64676 | $X_{final}$ = K sqrt($X^2$+$Y^2$) / MPS<br>$Y_{final}$ = 0<br>$Z_{final}$ = $Z$ + atan($Y$ / $X$)<br>where K ≈ 1.64676 |
| | For solving cos($Z$) and sin($Z$), set $X$ = 1 / K, $Y$ = 0.<br>Useful domain: Full range of $X$, $Y$ and $Z$ supported due to pre-processing logic. | For solving magnitude of vector (sqrt($x^2$+$y^2$)), set $X$ = x / K, $Y$ = y / K.<br>Useful domain: Full range of $X$ and $Y$ supported due to pre- and post-processing logic.<br><br>For solving atan($Y$ / $X$), set $Z$ = 0.<br>Useful domain: Full range of $X$ and $Y$, except $X$ = 0. |
| | Relationships:<br>tan(v) = sin(v) / cos(v) | Relationships:<br>acos(w) = atan[sqrt(1-$w^2$) / w]<br>asin(w) = atan[w / sqrt(1-$w^2$)] |
| **Linear**<br>$m$ = 0<br>$e_i$ = $2^{-i}$ | $X_{final}$ = $X$ / MPS<br>$Y_{final}$ = [$Y$ + $X$ $Z$] / MPS<br>$Z_{final}$ = 0 | $X_{final}$ = $X$ / MPS<br>$Y_{final}$ = 0<br>$Z_{final}$ = $Z$ + $Y$ / $X$ |
| | For solving $X \cdot Z$, set $Y$ = 0.<br>Useful domain: $\|Z\| \leq 2$. | For solving ratio $Y$ / $X$, set $Z$ = 0.<br>Useful domain: $\|Y / X\| \leq 2$, $X > 0$. |

**Table 11-2 CORDIC Coprocessor Operating Modes and Corresponding Result Data** (cont'd)

| Function | Rotation Mode | Vectoring Mode |
|---|---|---|
| **Hyperbolic**<br>$m$ = -1<br>$e_i$ = atanh($2^{-i}$) | $X_{final}$ = k[$X$ cosh($Z$) - $Y$ sinh($Z$)] / MPS<br>$Y_{final}$ = k[$Y$ cosh($Z$) + $X$ sinh($Z$)] / MPS<br>$Z_{final}$ = 0<br>where k ≈ 0.828 | $X_{final}$ = k sqrt($X^2$-$Y^2$) / MPS<br>$Y_{final}$ = 0<br>$Z_{final}$ = $Z$ + atanh($Y$ / $X$)<br>where k ≈ 0.828 |
| | For solving cosh($Z$) and sinh($Z$) and $e^Z$, set $X$ = 1 / k, $Y$ = 0.<br>Useful domain: \|$Z$\| ≤ 1.11rad, $Y$ = 0. | For solving sqrt($x^2$-$y^2$), set $X$ = x / k, $Y$ = y / k.<br>Useful domain: \|y\| < \|x\|, $X$ > 0.<br><br>For solving atanh($Y$ / $X$), set $Z$ = 0.<br>Useful domain: \|atanh($Y$ / $X$)\| ≤ 1.11rad, $X$ > 0. |
| | Relationships:<br>tanh(v) = sinh(v) / cosh(v)<br>$e^v$ = sinh(v) + cosh(v)<br>$w^t$ = $e^{t\ ln(w)}$ | Relationships:<br>ln(w) = 2 atanh[(w-1) / (w+1)]<br>sqrt(w) = sqrt((w+0.25)$^2$-(w-0.25)$^2$)<br>acosh(w) = ln[w+sqrt(1-$w^2$)]<br>asinh(w) = ln[w+sqrt(1+$w^2$)] |

### Usage Notes

- For solving the respective functions, user must initialize the CORDIC data (X, Y and Z) with meaningful initial values within domain of convergence to ensure result convergence. The 'useful domain' listed in **Table 11-2** covers the supported domain of convergence for the CORDIC algorithm and excludes the not-meaningful range(s) for the function. For details regarding the supported domain of convergence, refer to **Chapter 11.2.4.1**. For result data accuracy, refer to **Chapter 11.2.6**.
- Function limitations must be considered, e.g., setting initial X = 0 for atan(Y / X) is not meaningful. Violations of such function limitations may yield incoherent CORDIC result data.
- All data inputs are processed and handled as twos complement. Only exception is user-option for X result data (only) to be read as unsigned value.
- The only case where the result data is always positive and larger than the initial data is X result data (only) in circular vectoring mode; therefore, the user may want to use the MSB bit as data bit instead of sign bit. By setting X_USIGN = 1, X result data will be processed as unsigned data.
- For circular and hyperbolic functions, and due to the corresponding fixed LUT, the Z data is always handled as signed integer S19 (accessible as S15). The LUTs contain scaled integer values (S19) of atan($2^{-i}$) for i = 0, 1, 2, ..., 15 and atanh($2^{-i}$) for

i = 1, 2, ..., 15, such that angles in the range $[-\pi,((2^{19}-1)/2^{19})\pi]$ are represented by integer values ranging $[-2^{19},(2^{19}-1)]$. Therefore, Z data is limited (not considering domain of convergence) to represent angles $[-\pi,((2^{15}-1)/2^{15})\pi]$ for these CORDIC functions. Any calculated value of Z outside of this range will result in overflow error.

- For linear function, the Z data is always handled as signed fraction S4.15 (accessible as S4.11 in the form signed 4Q16). The emulated LUT is actually a shift register that holds data in the form 1.15 which gives the real value of $2^{-i}$. Therefore, regardless of the domain of convergence, Z data is logically only useful for values whose magnitude is smaller than 16. Overflow error is indicated by the CD_STATC.ERROR bit.
- The MPS setting has no effect on Z data. User must ensure proper initialization of Z initial data to prevent overflow and incorrect result data.
- The CORDIC Coprocessor is designed such that with correct user setting of MPS > 1, there is no internal overflow of the X and Y data and the read result data is complete. However, note that in these cases, the higher the MPS setting, the lower the resolution of the result data due to loss of LSB bit(s).
- The hyperbolic rotation mode is limited, in terms of result accuracy, in that initial Y data must be set to zero. In other words, the CORDIC Coprocessor is not able to return accurate result for cosh(Z)+/-sinh(Z) in a single calculation.

### 11.2.4.1 Domains of Convergence

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

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

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

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

**Circular Rotation Mode:** The full range of Z input $[-2^{15},(2^{15}-1)]$ representing angles $[-\pi,((2^{15}-1)/2^{15})\pi]$ is supported. No limitations on initial X and Y inputs, except for overflow considerations which can be overcome with MPS setting.

**Circular Vectoring Mode:** The full range of X and Y inputs [$-2^{15}$,($2^{15}$-1)] are supported, while Z initial value should satisfy $|Z| \leq \pi / 2$ to prevent possible Z result data overflow.

*Note: Considerations should also be given to function limitations such as the meaning of the result data, e.g. divide by zero is not meaningful. The 'useful domain' included within **Table 11-2** for each of the main functions, attempts to cover both for CORDIC convergence and useful range of the function.*

*Note: Input values may be within the domain of convergence, however, this does not guarantee a fixed level of accuracy of the CORDIC result data. Refer to **Chapter 11.2.6** for details on accuracy of the CORDIC Coprocessor.*

### 11.2.4.2 Overflow Considerations

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

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

The MPS setting has no effect on the Z data. For circular and hyperbolic functions, any value of Z outside of the range [$-\pi$,(($2^{15}$-1)/$2^{15}$)$\pi$] cannot be represented and will result in Z data overflow error. Note that kernel data Z has values in the range [$-\pi$,(($2^{19}$-1)/$2^{19}$)$\pi$] scaled to the range [$-2^{19}$,($2^{19}$-1)], so the written and read values of Z data are always normalized as such. For linear function, where Z is a real value, magnitude of Z must not exceed 4 integer bits.

## 11.2.5 CORDIC Coprocessor Data Format

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

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

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

The Z data is always handled as integer, based on the normalization factor for circular or hyperbolic function. In case of linear function, accessible Z data is a real number with

fixed input and result data form of S4.11 (signed 4Q16) which is a fraction with 11 decimal places.

Refer to **Chapter 11.2.3** for details on data normalization.

### 11.2.6 Accuracy of CORDIC Coprocessor

Each CORDIC calculation involves a fixed number of 16 CORDIC iterations starting from iteration 0. The hyperbolic function is special in this respect in that it starts from iteration 1 with repeat iterations at defined steps. The addressable data registers are 16 bits wide, while the internal kernel X and Y data registers used for calculation are each 26 bits wide (24 data bits plus 2 overflow bits) and internal kernel Z data register is 21 bits wide (20 data bits plus 1 overflow bit). For more details on the data form of the LUTs, refer to **Chapter 11.3.1** and **Chapter 11.3.2**.

For input data values within the specified useful domain (see **Table 11-2**), the result of each calculation of the CORDIC Coprocessor is guaranteed to converge, although the accuracy is not fixed per data form in each operating mode. The accuracy is a measure of the magnitude of the difference between the result data and the expected data from a high-accuracy calculator. “Normalized Deviation” (ND) is a generic term used to refer to the magnitude of deviation of the result data from the expected result. The deviation is calculated as if the input/result data is integer. In case the data is a rational number, the magnitude of deviation has to be interpreted. For example, Z for linear vectoring mode of the data form S4.11 - ND = 1 ($01_B$) means the difference from expected real data has magnitude of no more than $|2^{-11} + 2^{-11}|$; ND = 2 ($10_B$) means the difference is no more than $|2^{-10}+2^{-11}|$; ND = 3 ($11_B$) means the difference is no more than $|2^{-11}+2^{-10}+2^{-11}|$; ND = 4 ($100_B$) means the difference is no more than $|2^{-9}+2^{-11}|$, and so on. The value of $2^{-11}$ is always added to account for possible truncation error.

**Table 11-3** lists the probability of Normalized Deviation in a single calculation, obtained from simulation with approximately one million different input sets for each respective CORDIC Coprocessor operating mode, based on the input conditions specified (always within useful domain, possibly with additional conditions).

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

**Table 11-3 Normalized Deviation of a Calculation**

| Mode | X Normalized Deviation | Y or Z Normalized Deviation |
|---|---|---|
| **Circular Vectoring** | Input conditions: Useful Domain and [(1.64676/2)·sqrt($X^2$+$Y^2$) ≥ 600] | |
| | 0 : 50.8317%<br>1 : 49.1683%<br>ND for X ≤ 1 | 0 : 55.8702%<br>1 : 44.1298%<br>ND for Z ≤ 1 |
| **Circular Rotation** | Input conditions: Useful Domain (Full range of X, Y and Z) | |
| | 0 : 50.7715%<br>1 : 48.8579%<br>2 : 0.3681%<br>3 : 0.0023%<br>4 : 0.0002%<br>ND for X ≤ 4 | 0 : 51.2011%<br>1 : 48.4944%<br>2 : 0.3024%<br>3 : 0.0020%<br>4 : 0.0001%<br>ND for Y ≤ 4 |
| **Linear Vectoring** | Input conditions: Useful Domain (\|Y / X\| ≤ 2, X > 0) | |
| | 0 : 66.9170%<br>1 : 33.0830%<br>ND for X ≤ 1 | 0 : 88.5676%<br>1 : 11.4322%<br>2 : 0.0002%<br>ND for Z ≤ 2 |
| **Linear Rotation** | Input conditions: Useful Domain (\|Z\| ≤ 2) | |
| | 0 : 69.7141%<br>1 : 30.2859%<br>ND for X ≤ 1 | 0 : 62.4055%<br>1 : 37.1965%<br>2 : 0.3980%<br>ND for Y ≤ 2 |
| **Hyperbolic Vectoring** | Input conditions: Useful Domain (\|Y\| < \|X\|, X > 0, \|atanh(Y / X)\| ≤ 1.11rad) | |
| | 0 : 34.5399%<br>1 : 34.5438%<br>2 : 17.9254%<br>3 : 11.6747%<br>4 : 1.3162%<br>ND for X ≤ 4 | 0 : 58.3062%<br>1 : 41.6938%<br>ND for Z ≤ 1 |

**Table 11-3 Normalized Deviation of a Calculation** (cont'd)

| Mode | X Normalized Deviation | Y or Z Normalized Deviation |
|---|---|---|
| **Hyperbolic Rotation** | Input conditions: Useful Domain ($\|Z\| \leq 1.11$rad, Y = 0) | |
| | 0 : 14.9401%<br>1 : 31.6474%<br>2 : 23.7692%<br>3 : 14.8353%<br>4 : 7.4881%<br>5 : 4.3398%<br>6 : 2.4387%<br>7 : 0.5267%<br>8 : 0.0146%<br>ND for $X \leq 8$ | 0 : 40.4787%<br>1 : 40.6711%<br>2 : 11.9209%<br>3 : 4.6940%<br>4 : 1.7290%<br>5 : 0.4453%<br>6 : 0.0607%<br>7 : 0.0003%<br>ND for $Y \leq 7$ |

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

### 11.2.7 Performance of CORDIC Coprocessor

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

The CORDIC Coprocessor uses barrel shifters for data shifting. For a fixed number of 16 iterations per calculation, the total time from the start of calculation to the instant the EOC flag is set is approximately 41 clock cycles (or less). It should be noted that the ERROR flag is valid only after one cycle. This timing for one complete calculation is applicable also to those modes which involve additional data processing, and also to the hyperbolic modes which involve repeat iterations and an extra cycle for mode setup.

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

## 11.3 The CORDIC Coprocessor Kernel

The CORDIC Coprocessor consists of data registers for holding the X, Y and Z values, in twos complement format. Three shift registers are used to shift the values in the X and Y registers by the number of iterations and to generate the emulated LUT data for the linear function. Additionally, two look-up tables (LUT) are implemented as combinatorial logic to support the circular and hyperbolic function each. The LUT data for the selected operating mode is multiplexed and then added to the data in the Z register with the correct sign. The atan LUT contains precalculated atan($2^{-i}$) values, while the atanh LUT contains precalculated atanh($2^{-i}$) values, both in twos complement format for i = iteration count. The emulated LUT, as mentioned above, is actually a shift register that generates data by shifting. This shift register is reloaded whenever the Finite-State-Machine (FSM) switches to the setup mode on starting a new calculation. The CORDIC Coprocessor FSM controls the flow of the calculation.

### 11.3.1 Arctangent and Hyperbolic Arctangent Look-Up Tables

The LUTs are 20bits and 21bits wide respectively, for the arctangent table (atan LUT) and hyperbolic arctangent table (atanh LUT). Each entry of the atan LUT is divided into 1 sign bit (MSB) followed by 19-bit integer part. For the atanh LUT, each entry has 1 repeater bit (MSB), followed by 1 sign bit, then 19-bit integer part.

The contents of the LUTs are:

- atan LUT with data form of S19, see **Table 11-4**

**Table 11-4 Precomputed Scaled Values for atan($2^{-i}$)**

| Iteration No. | Scaled atan($2^{-i}$) in hex | Iteration No. | Scaled atan($2^{-i}$) in hex |
|---|---|---|---|
| i = 0 | 20000 | i = 8 | 28C |
| i = 1 | 12E40 | i = 9 | 146 |
| i = 2 | 9FB4 | i = 10 | A3 |
| i = 3 | 5111 | i = 11 | 51 |
| i = 4 | 28B1 | i = 12 | 29 |
| i = 5 | 145D | i = 13 | 14 |
| i = 6 | A2F | i = 14 | A |
| i = 7 | 518 | i = 15 | 5 |

- atanh LUT with data form of S19, see **Table 11-5**

**Table 11-5 Precomputed Scaled Values for atanh($2^{-i}$)**

| Iteration No. | Scaled atanh($2^{-i}$) in hex | Iteration No. | Scaled atanh($2^{-i}$) in hex |
|---|---|---|---|
| i = 0 | - | i = 8 | 28C |
| i = 1 | 16618 | i = 9 | 146 |
| i = 2 | A681 | i = 10 | A3 |
| i = 3 | 51EA | i = 11 | 51 |
| i = 4 | 28CC | i = 12 | 29 |
| i = 5 | 1461 | i = 13 | 14 |
| i = 6 | A30 | i = 14 | A |
| i = 7 | 518 | i = 15 | 5 |

The Z data is a normalized representation of the actual angle. The internal scaling is such that $[-\pi,((2^{19}-1)/2^{19})\pi]$ is equivalent to $[-2^{19},(2^{19}-1)]$. The last 4 LSB bits are truncated, as 16-bit data is transferred to the data bus when addressed. From user's point, the angles $[-\pi,((2^{15}-1)/2^{15})\pi]$ are therefore represented by the range $[-2^{15},(2^{15}-1)]$.

### 11.3.2 Linear Function Emulated Look-Up Table

The emulated LUT for linear function is actually a shift register. The emulated LUT has 1 integer bit (MSB) followed by 15-bit fractional part of the form 1Q16.

In linear function, where Z is a real number, the internal Z data is of the form signed 4Q20. The externally read data has the last 4 bits of the fractional part truncated, resulting in a sign bit followed by 4-bit integer part, and finally 11-bit fractional part.

## 11.4 Low Power Mode

If the CORDIC Coprocessor functionality is not required at all, it can be completely disabled by gating off its clock input for maximal power reduction. This is done by setting bit CDC_DIS in register PMCON1 as described below. Refer to **Chapter 8.1.4** for details on peripheral clock management.

**PMCON1**
**Power Mode Control Register 1** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **0** | **CDC_DIS** | **CAN_DIS** | **MDU_DIS** | **T2CCU _DIS** | **CCU_DIS** | **SSC_DIS** | **ADC_DIS** |
| r | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| CDC_DIS | 6 | rw | CORDIC Disable Request. Active high.<br>0 CORDIC is in normal operation (default).<br>1 Request to disable the CORDIC. |
| 0 | 7 | r | Reserved<br>Returns 0 if read; should be written with 0. |

## 11.5 Register Map

The CORDIC Coprocessor registers are located in the mapped Special Function Register (SFR) area. **Table 11-6** lists the addresses of these registers.

*Note: All CORDIC Coprocessor register names described in this section shall be referenced fully with the module name prefix "CD_".*

**Table 11-6 Register Summary for CORDIC Coprocessor**

| Name | Address (HEX) | Reset Value (HEX) | Description |
|---|---|---|---|
| CD_CORDXL | 9A | 00 | CORDIC X Data Low Byte |
| CD_CORDXH | 9B | 00 | CORDIC X Data High Byte |
| CD_CORDYL | 9C | 00 | CORDIC Y Data Low Byte |
| CD_CORDYH | 9D | 00 | CORDIC Y Data High Byte |
| CD_CORDZL | 9E | 00 | CORDIC Z Data Low Byte |
| CD_CORDZH | 9F | 00 | CORDIC Z Data High Byte |
| CD_STATC | A0 | 00 | CORDIC Status and Data Control Register |
| CD_CON | A1 | 62 | CORDIC Control Register |

## 11.6 Register Description

### 11.6.1 Control Register

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

**CD_CON**
**CORDIC Control Register** **Reset Value: $62_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| MPS | | X_USIGN | ST_MODE | ROTVEC | MODE | | ST |
| rw | | rw | rw | rw | rw | | rwh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **ST** | 0 | rwh | **Start Calculation**<br>If ST_MODE = 1, set ST to start a CORDIC calculation. Is effective only while BSY is not set. This bit may be set with the other bits of this register in one write access.<br>Cleared by hardware at the beginning of calculation. |
| **MODE** | 2:1 | rw | **Operating Mode**<br>00 Linear Mode<br>01 Circular Mode (default)<br>10 Reserved<br>11 Hyperbolic Mode |
| **ROTVEC** | 3 | rw | **Rotation Vectoring Selection**<br>0 Vectoring Mode (default)<br>1 Rotation Mode |
| **ST_MODE** | 4 | rw | **Start Method**<br>0 Auto start of calculation after write access to X low byte CD_CORDXL (default)<br>1 Start calculation only after bit ST is set |

| Field | Bits | Type | Description |
|---|---|---|---|
| **X_USIGN** | 5 | rw | **Result Data Format for X in Circular Vectoring Mode**<br>When reading the X result data with DMAP = 0,<br>X data has a data format of:<br>0 Signed, twos complement<br>1 Unsigned (default)<br>With this bit set, the MSB bit of the X result data is processed as a data bit instead of a sign bit.<br>*Note: This bit is only effective when operating in circular vectoring mode. In all other modes, X is always processed as twos complement data.*<br>*Note: X_USIGN = 1 is meaningful in circular vectoring mode because the result data is always positive and always larger than the initial data.* |
| **MPS** | 7:6 | rw | **X and Y Magnitude Prescaler**<br>After the last iteration of a calculation, the calculated value of X and Y are each divided by this factor to yield the result.<br>Proper setting of these bits is important to avoid an overflow of the result in the respective kernel data registers.<br>00 Divide by 1<br>01 Divide by 2 (default)<br>10 Divide by 4<br>11 Reserved, retain the last MPS setting |

### 11.6.2 Status and Data Control Register

The CD_STATC register is bit-addressable, and generally reflects the status of the CORDIC Coprocessor. The register also contain bits for data control, as well as for interrupt control.

**CD_STATC**
**CORDIC Status and Data Control Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **KEEPZ** | **KEEPY** | **KEEPX** | **DMAP** | **INT_EN** | **EOC** | **ERROR** | **BSY** |
| rw | rw | rw | rw | rw | rwh | rh | rh |

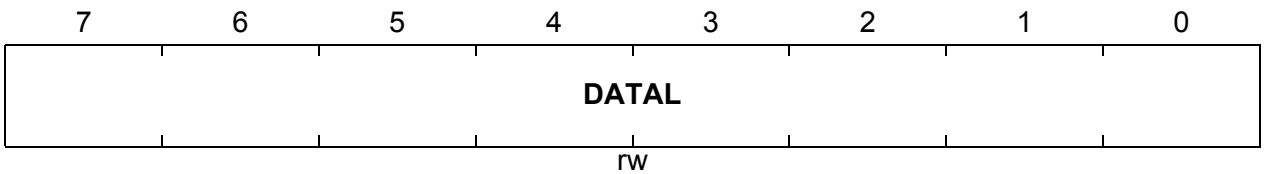
| Field | Bits | Type | Description |
|---|---|---|---|
| **BSY** | 0 | rh | **Busy Indication**<br>Indicates a running calculation when set. The flag is asserted one clock cycle after bit ST was set. It is deasserted at the end of a calculation. |
| **ERROR** | 1 | rh | **Error Indication**<br>In case of overflow error in the calculated result for X, Y or Z, this bit is set at the end of CORDIC calculation. Cleared after any read access on this register, or when a new CORDIC calculation is started. |
| **EOC** | 2 | rwh | **End of Calculation Flag**<br>Set at the end of a complete CORDIC calculation when BSY goes inactive.<br>Unless cleared by software, bit remains set until a read access is performed to the low byte of Z result data (DMAP = 0) where the bit is automatically cleared by hardware. |
| **INT_EN** | 3 | rw | **Interrupt Enable**<br>Set to enable CORDIC Coprocessor interrupt |
| **DMAP** | 4 | rw | **Data Map**<br>0 Read (result) data from kernel data registers (default)<br>1 Read (initial) data from the shadow data registers |

| Field | Bits | Type | Description |
|---|---|---|---|
| **KEEPX** | 5 | rw | **Last X Result as Initial Data for New Calculation**<br>If set, a new calculation will use as initial data, the value of the result from the previous calculation. In other words, the respective kernel data register will not be overwritten by the contents of the shadow data register at the beginning of new calculation. This bit should always be cleared for the very first calculation to load the initial X data.<br>*Note: Independent of the KEEP bit, the shadow data registers will continue to hold the last written initial data value until the next software write.*<br>*Note: If KEEPx bit is set for a multi-step calculation, the accuracy of the corresponding final x result data may be reduced and is not guaranteed as shown in* **Section 11.2.6**. |
| **KEEPY** | 6 | rw | **Last Y Result as Initial Data for New Calculation**<br><See description for KEEPX> |
| **KEEPZ** | 7 | rw | **Last Z Result as Initial Data for New Calculation**<br><See description for KEEPX> |

### 11.6.3 Data Registers

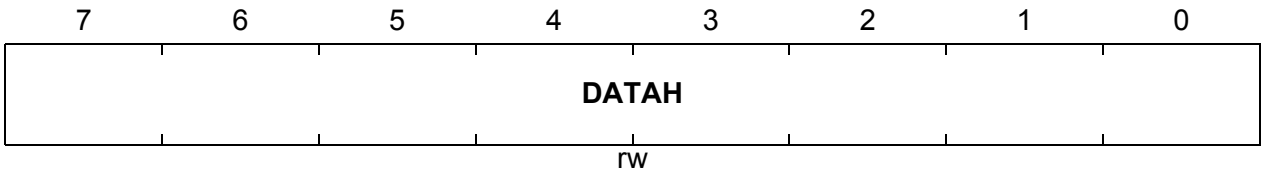
The Data registers are used to initialize the X, Y and Z parameters. The result data from CORDIC calculation can also be read (DMAP = 0). Reading of the shadow registers for initial data are also possible (DMAP = 1). Regardless of the DMAP setting for reading, these data registers always hold the last written initial value until the next user software write, or reset.

**CD_CORDxL (x = X, Y or Z)**
**CORDIC x Data Low Byte** **Reset Value: $00_H$**



| Field | Bits | Type | Description |
|---|---|---|---|
| **DATAL** | 7:0 | rw | **Low Byte Data**<br>Write to this byte always writes to the low byte of the corresponding shadow data register. New data may be written during an ongoing CORDIC calculation.<br><br>For read,<br>DMAP=0: Result data from kernel data byte<br>DMAP=1: Initial data from the shadow data byte |

**CD_CORDxH (x = X, Y or Z)**
**CORDIC x Data High Byte** **Reset Value: $00_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **DATAH** | 7:0 | rw | **High Byte Data**<br>Write to this byte always writes to the high byte of the corresponding shadow data register. New data may be written during an ongoing CORDIC calculation.<br><br>For read,<br>DMAP=0: Result data from kernel data byte<br>DMAP=1: Initial data from the shadow data byte |

# 12 Serial Interfaces

The XC878 contains three serial interfaces, which consists of two Universal Asynchronous Receivers/Transmitters (UART and UART1) and a High-Speed Synchronous Serial Interface (SSC), for serial communication with external devices. Additionally, the UART module can be used to support the Local Interconnect Network (LIN) protocol.

### UART and UART1 Features

- Full-duplex asynchronous modes
  - 8-bit or 9-bit data frames, LSB first
  - fixed or variable baud rate
- Receive buffered
- Multiprocessor communication
- Interrupt generation on the completion of a data transmission or reception

### LIN Features

- Master and slave mode operation

### SSC Features

- Master and slave mode operation
  - Full-duplex or half-duplex operation
- Transmit and receive buffered
- Flexible data format
  - Programmable number of data bits: 2 to 8 bits
  - Programmable shift direction: LSB or MSB shift first
  - Programmable clock polarity: idle low or high state for the shift clock
  - Programmable clock/data phase: data shift with leading or trailing edge of the shift clock
- Variable baud rate
- Compatible with Serial Peripheral Interface (SPI)
- Interrupt generation
  - On a transmitter empty condition
  - On a receiver full condition
  - On an error condition (receive, phase, baud rate, transmit error)

## 12.1 UART

The UART provides a full-duplex asynchronous receiver/transmitter, i.e., it can transmit and receive simultaneously. It is also receive-buffered, i.e., it can commence reception of a second byte before a previously received byte has been read from the receive register. However, if the first byte still has not been read by the time reception of the second byte is complete, one of the bytes will be lost.

*Note: The term "UART" is used to represent the serial port in general and is applicable to both UART and UART1 modules. If it is followed by the word "module" as in "UART module", it is used to represent the first UART module.*

### 12.1.1 UART Modes

The UART can be used in four different modes. In mode 0, it operates as an 8-bit shift register. In mode 1, it operates as an 8-bit serial port. In modes 2 and 3, it operates as a 9-bit serial port. The only difference between mode 2 and mode 3 is the baud rate, which is fixed in mode 2 but variable in mode 3. The variable baud rate is set by either the underflow rate on the dedicated baud-rate generator, or by the overflow rate on Timer 1.

The different modes are selected by setting bits SM0 and SM1 to their corresponding values, as shown in **Table 12-1**.

**Table 12-1 UART Modes**

| SM0 | SM1 | Operating Mode | Baud Rate |
|---|---|---|---|
| 0 | 0 | Mode 0: 8-bit shift register | $f_{PCLK}/2$ |
| 0 | 1 | Mode 1: 8-bit shift UART | Variable |
| 1 | 0 | Mode 2: 9-bit shift UART | $f_{PCLK}/64$ or $f_{PCLK}/32$[1)] |
| 1 | 1 | Mode 3: 9-bit shift UART | Variable |

1) For UART1 module, the baud rate is fixed at $f_{PCLK}/64$.

#### 12.1.1.1 Mode 0, 8-Bit Shift Register, Fixed Baud Rate

In mode 0, the serial port behaves as an 8-bit shift register. Data is shifted in through RXD, and out through RXDO, while the TXD line is used to provide a shift clock which can be used by external devices to clock data in and out.

The transmission cycle is activated by a write to SBUF. One machine cycle later, the data has been written to the transmit shift register with a 1 at the 9th bit position. For the next seven machine cycles, the contents of the transmit shift register are shifted right one position and a zero shifted in from the left so that when the MSB of the data byte is at the output position, it has a 1 and a sequence of zeros to its left. The control block then executes one last shift before setting the TI bit.

Reception is started by the condition REN = 1 and RI = 0. At the start of the reception cycle, $11111110_B$ is written to the receive shift register. In each machine cycle that follows, the contents of the shift register are shifted left one position and the value sampled on the RXD line in the same machine cycle is shifted in from the right. When the 0 of the initial byte reaches the leftmost position, the control block executes one last shift, loads SBUF and sets the RI bit.

The baud rate for the transfer is fixed at $f_{PCLK}/2$ where $f_{PCLK}$ is the input clock frequency, i.e. one bit per machine cycle.

#### 12.1.1.2 Mode 1, 8-Bit UART, Variable Baud Rate

In mode 1, the UART behaves as an 8-bit serial port. A start bit (0), 8 data bits, and a stop bit (1) are transmitted on TXD or received on RXD at a variable baud rate.

The transmission cycle is activated by a write to SBUF. The data is transferred to the transmit register and a 1 is loaded to the 9th bit position (as in mode 0). At phase 1 of the machine cycle after the next rollover in the divide-by-16 counter, the start bit is copied to TXD, and data is activated one bit time later. One bit time after the data is activated, the data starts getting shifted right with zeros shifted in from the left. When the MSB gets to the output position, the control block executes one last shift and sets the TI bit.

Reception is started by a high to low transition on RXD (sampled at 16 times the baud rate). The divide-by-16 counter is then reset and $1111\ 1111_B$ is written to the receive register. If a valid start bit (0) is then detected (based on two out of three samples), it is shifted into the register followed by 8 data bits. If the transition is not followed by a valid start bit, the controller goes back to looking for a high to low transition on RXD. When the start bit reaches the leftmost position, the control block executes one last shift, then loads SBUF with the 8 data bits, loads RB8 (SCON.2) with the stop bit, and sets the RI bit, provided RI = 0, and either SM2 = 0 (see **Section 12.1.2**) or the received stop bit = 1. If none of these conditions is met, the received byte is lost.

The associated timings for transmit/receive in mode 1 are illustrated in **Figure 12-1**.

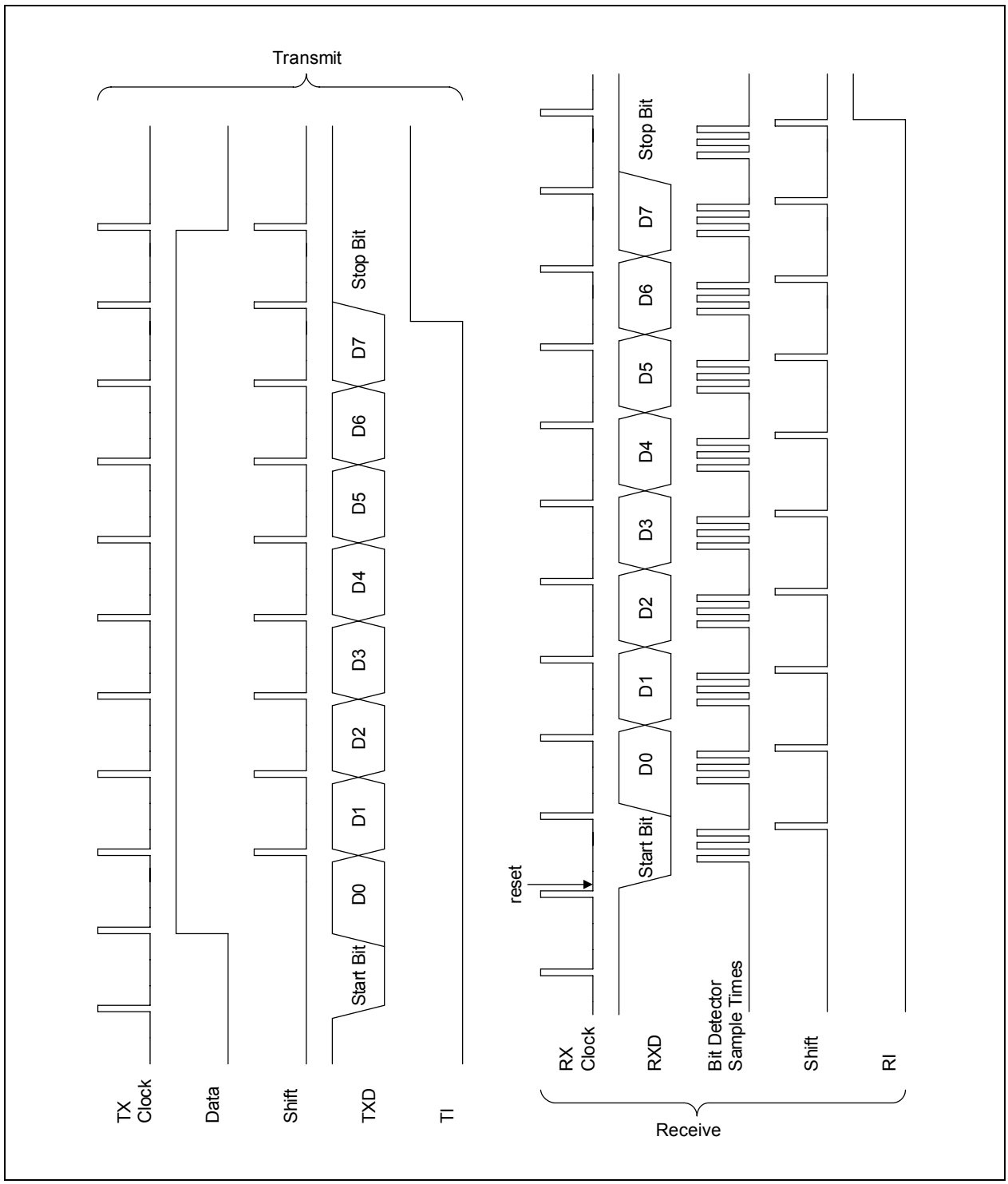



**Figure 12-1 Serial Interface, Mode 1, Timing Diagram**

### 12.1.1.3 Mode 2, 9-Bit UART, Fixed Baud Rate

In mode 2, the UART behaves as a 9-bit serial port. A start bit (0), 8 data bits plus a programmable 9th bit and a stop bit (1) are transmitted on TXD or received on RXD. The 9th bit for transmission is taken from TB8 (SCON.3) while for reception, the 9th bit received is placed in RB8 (SCON.2).

The transmission cycle is activated by a write to SBUF. The data is transferred to the transmit register and TB8 is copied into the 9th bit position. At phase 1 of the machine cycle following the next rollover in the divide-by-16 counter, the start bit is copied to TXD and data is activated one bit time later. One bit time after the data is activated, the data starts shifting right. For the first shift, a stop bit (1) is shifted in from the left and for subsequent shifts, zeros are shifted in. When the TB8 bit gets to the output position, the control block executes one last shift and sets the TI bit.

Reception is started by a high to low transition on RXD (sampled at 16 times the baud rate). The divide-by-16 counter is then reset and 1111 $1111_B$ is written to the receive register. If a valid start bit (0) is then detected (based on two out of three samples), it is shifted into the register followed by 8 data bits. If the transition is not followed by a valid start bit, the controller goes back to looking for a high to low transition on RXD. When the start bit reaches the leftmost position, the control block executes one last shift, then loads SBUF with the 8 data bits, loads RB8 (SCON.2) with the 9th data bit, and sets the RI bit, provided RI = 0, and either SM2 = 0 (see **Section 12.1.2**) or the 9th bit = 1. If none of these conditions is met, the received byte is lost.

The baud rate for the transfer is either $f_{PCLK}$/64 or $f_{PCLK}$/32 for UART module, depending on the setting of the top bit (SMOD) of the PCON (Power Control) register, which acts as a Double Baud Rate selector. For UART1 module, the baud rate is fixed at $f_{PCLK}$/64.

### 12.1.1.4 Mode 3, 9-Bit UART, Variable Baud Rate

Mode 3 is the same as mode 2 in all respects except that the baud rate is variable.

In all modes, transmission is initiated by any instruction that uses SBUF as a destination register. Reception is initiated in the modes by the incoming start bit if REN = 1.

The serial interface also provides interrupt requests when transmission or reception of the frames has been completed. The corresponding interrupt request flags are TI or RI, respectively. If the serial interrupt is not used (i.e., serial interrupt not enabled), TI and RI can also be used for polling the serial interface.

The associated timings for transmit/receive in modes 2 and 3 are illustrated in **Figure 12-2**.

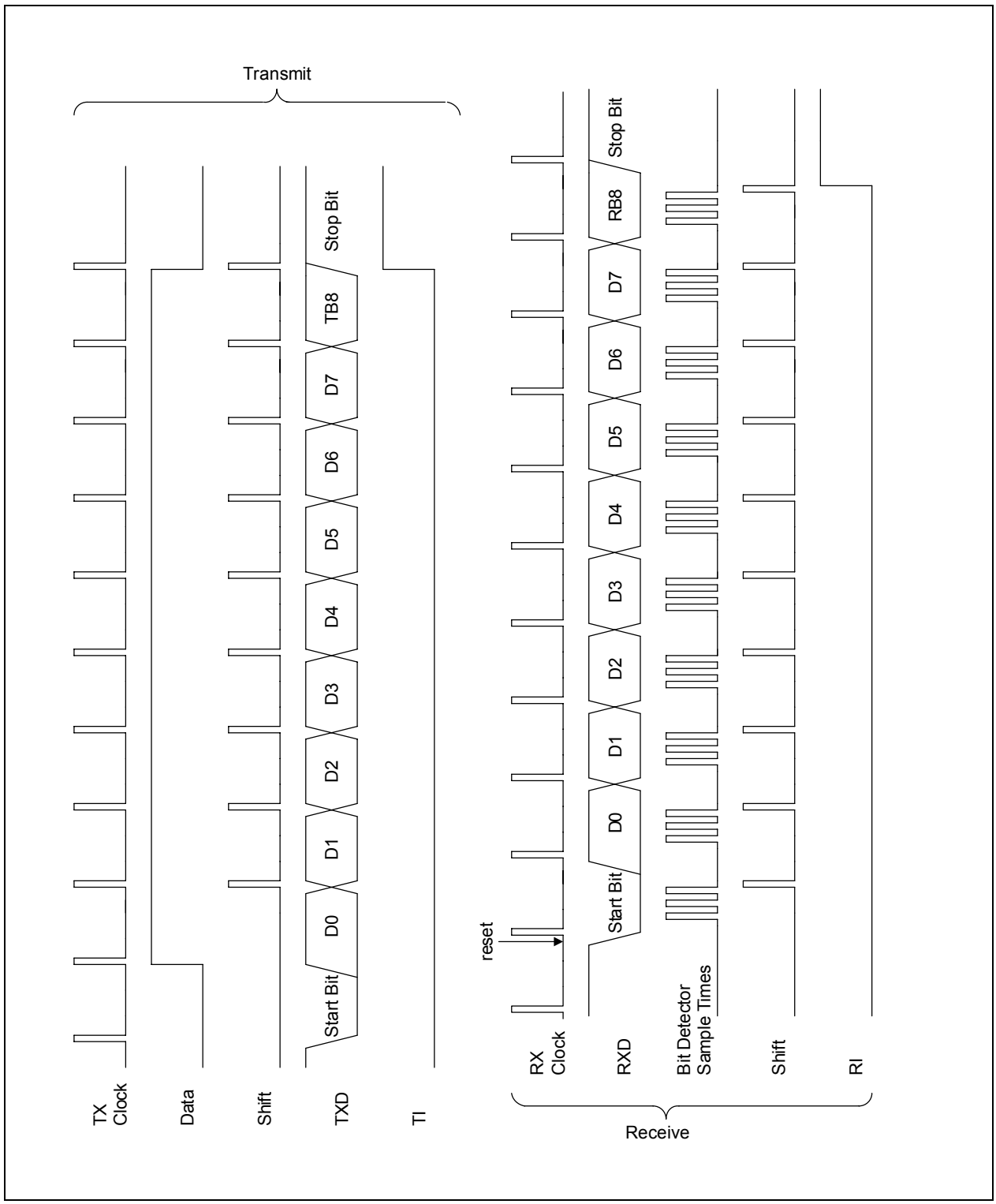



**Figure 12-2 Serial Interface, Modes 2 and 3, Timing Diagram**

### 12.1.2 Multiprocessor Communication

Modes 2 and 3 have a special provision for multiprocessor communication using a system of address bytes with bit 9 = 1 and data bytes with bit 9 = 0. In these modes, 9 data bits are received. The 9th data bit goes into RB8. The communication always ends with one stop bit. The port can be programmed such that when the stop bit is received, the serial port interrupt will be activated only if RB8 = 1.

This feature is enabled by setting bit SM2 in SCON. One of the ways to use this feature in multiprocessor systems is described in the following paragraph.

When the master processor wants to transmit a block of data to one of several slaves, it first sends out an address byte that identifies the target slave. An address byte differs from a data byte in that the 9th bit is 1 in an address byte and 0 in a data byte. With SM2 = 1, no slave will be interrupted by a data byte. An address byte, however, will interrupt all slaves, so that each slave can examine the received byte and see if it is being addressed. The addressed slave will clear its SM2 bit and prepare to receive the data bytes that will be coming. The slaves that were not being addressed retain their SM2s as set and ignore the incoming data bytes.

Bit SM2 has no effect in mode 0. SM2 can be used in mode 1 to check the validity of the stop bit. In a mode 1 reception, if SM2 = 1, the receive interrupt will not be activated unless a valid stop bit is received.

### 12.1.3 UART Register Description

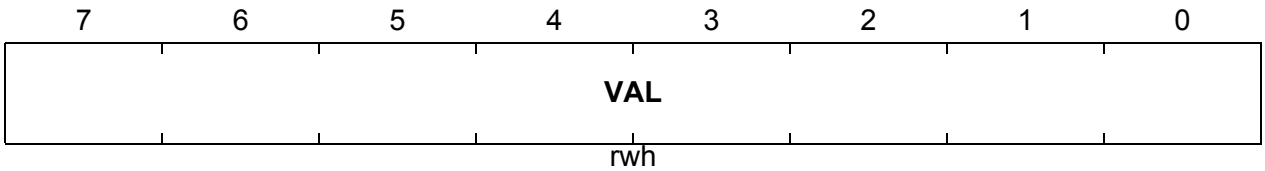
Both UART modules contain the two Special Function Registers (SFRs), SCON and SBUF. SCON is the control register and SBUF is the data register. On reset, both SCON and SBUF return $00_H$. The serial port control and status register is the SFR SCON. This register contains not only the mode selection bits, but also the 9th data bit for transmit and receive (TB8 and RB8) and the serial port interrupt bits (TI and RI).

SBUF is the receive and transmit buffer of the serial interface. Writing to SBUF loads the transmit register and initiates transmission. This register is used for both transmit and receive data. Transmit data is written to this location and receive data is read from this location, but the two paths are independent.

Reading out SBUF accesses a physically separate receive register.

UART1 module has additionally a third SFR, SCON1, to enable or disable the receive, transmit and divider overflow interrupts. Setting bit RIEN to 1, for example, enables UART1 module's receive interrupt.

## SBUF
**Serial Data Buffer** **Reset Value: $00_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| VAL | [7:0] | rwh | **Serial Interface Buffer Register** |

## SCON
**Serial Channel Control Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **SM0** | **SM1** | **SM2** | **REN** | **TB8** | **RB8** | **TI** | **RI** |
| rw | rw | rw | rw | rw | rwh | rwh | rwh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **RI** | 0 | rwh | **Receive Interrupt Flag**<br>This is set by hardware at the end of the 8th bit on mode 0, or at the half point of the stop bit in modes 1, 2, and 3. Must be cleared by software. |
| **TI** | 1 | rwh | **Transmit Interrupt Flag**<br>This is set by hardware at the end of the 8th bit in mode 0, or at the beginning of the stop bit in modes 1, 2, and 3. Must be cleared by software. |
| **RB8** | 2 | rwh | **Serial Port Receiver Bit 9**<br>In modes 2 and 3, this is the 9th data bit received.<br>In mode 1, this is the stop bit received.<br>In mode 0, this bit is not used. |
| **TB8** | 3 | rw | **Serial Port Transmitter Bit 9**<br>In modes 2 and 3, this is the 9th data bit sent. |
| **REN** | 4 | rw | **Enable Receiver of Serial Port**<br>0 Serial reception is disabled.<br>1 Serial reception is enabled. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **SM2** | 5 | rw | Enable Serial Port Multiprocessor Communication in Modes 2 and 3<br>In mode 2 or 3, if SM2 is set to 1, RI will not be activated if the received 9th data bit (RB8) is 0.<br>In mode 1, if SM2 is set to 1, RI will not be activated if a valid stop bit (RB8) was not received.<br>In mode 0, SM2 should be 0. |
| **SM1, SM0** | 6<br>7 | rw | **Serial Port Operating Mode Selection**<br>00 Mode 0: 8-bit shift register, fixed baud rate ($f_{PCLK}$/2).<br>01 Mode 1: 8-bit UART, variable baud rate.<br>10 Mode 2: 9-bit UART, fixed baud rate ($f_{PCLK}$/64 or $f_{PCLK}$/32).<br>11 Mode 3: 9-bit UART, variable baud rate. |

**SCON1**
**Serial Channel Control Register 1** **Reset Value: $07_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **0** | | | | | **NDOVEN** | **TIEN** | **RIEN** |
| r | | | | | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **RIEN** | 0 | rw | **Receive Interrupt Enable Bit**<br>0 Receive interrupt is disabled.<br>1 Receive interrupt is enabled. |
| **TIEN** | 1 | rw | **Transmit Interrupt Enable Bit**<br>0 Transmit interrupt is disabled.<br>1 Transmit interrupt is enabled. |
| **NDOVEN** | 2 | rw | **Normal Divider Overflow Enable Bitt**<br>0 Normal divider overflow interrupt is disabled.<br>1 Normal divider overflow interrupt is enabled. |
| **0** | [7:3] | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

### 12.1.4 Baud Rate Generation

There are several ways to generate the baud rate clock for the serial ports, depending on the mode in which they are operating.

The baud rates in modes 0 and 2 are fixed, so they use the

- Fixed clock, (see **Section 12.1.4.1**)

In modes 1 and 3, the variable baud rate is generated using the

- Dedicated baud-rate generator (see **Section 12.1.4.2**)

Additionally for UART module, the variable baud can also be generated using

- Timer 1 (see **Section 12.1.4.3**)

This selection between the different variable baud rate sources is performed by bit BGS in UART module's FDCON register.

#### 12.1.4.1 Fixed Clock

The baud rates in modes 0 and 2 are fixed. However, for the case of UART module, while the baud rate in mode 0 can only be $f_{PCLK}/2$, the baud rate in mode 2 can be selected as either $f_{PCLK}/64$ or $f_{PCLK}/32$ depending on bit SMOD. Bit SMOD in the PCON register acts as a double baud rate selector in modes 1, 2 and 3. In modes 1 and 3, only the variable baud rate supplied by Timer 1 is dependent on SMOD. The baud rate supplied by the dedicated baud-rate generator is independent of SMOD.

"Baud rate clock" and "baud rate" must be distinguished from each other. The serial interface requires a clock rate that is 16 times the baud rate for internal synchronization. Therefore, the dedicated baud-rate generator and Timer 1 must provide a "baud rate clock" to the serial interface where it is divided by 16 to obtain the actual "baud rate". The abbreviation $f_{PCLK}$ refers to the input clock frequency.

**PCON**
**Power Control Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **SMOD** | **0** | | | **GF1** | **GF0** | **0** | **IDLE** |
| rw | r | | | rw | rw | r | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **SMOD** | 7 | rw | **Double Baud Rate Enable**<br>0 Do not double the baud rate of serial interface in modes 1, 2 and 3.<br>1 Double the baud rate of serial interface in mode 2, and in modes 1 and 3 only if Timer 1 is used as variable baud rate source. |
| **0** | 1, [6:4] | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

**Baud rate in Mode 2**

For UART module, the baud rate in mode 2 is dependent on the value of bit SMOD in the PCON register. If SMOD = 0 (value after reset), the baud rate is 1/64 of the input clock frequency $f_{PCLK}$. If SMOD = 1, the baud rate is 1/32 of $f_{PCLK}$.

(12.1)

$$\text{Mode 2 baud rate} = \frac{2^{\text{SMOD}}}{64} \times f_{PCLK}$$

For UART1 module, the baud rate in mode 2 does not depend on the bit SMOD and is always 1/64 of the input clock frequency $f_{PCLK}$.

### 12.1.4.2 Dedicated Baud-rate Generator

Each of the UART modules has a dedicated baud-rate generator that is based on a programmable 8-bit reload value, and includes divider stages (i.e., prescaler and fractional divider) for generating a wide range of baud rates based on its input clock $f_{PCLK}$.

The baud rate timer is a count-down timer and is clocked by either the output of the fractional divider ($f_{MOD}$) if the fractional divider is enabled (FDCON.FDEN = 1), or the output of the prescaler ($f_{DIV}$) if the fractional divider is disabled (FDEN = 0). For baud rate generation, the fractional divider must be configured to fractional divider mode (FDCON.FDM = 0). This allows the baud rate control run bit BCON.R to be used to start or stop the baud rate timer. At each timer underflow, the timer is reloaded with the 8-bit reload value in register BG and one clock pulse is generated for the serial channel.

Enabling the fractional divider in normal divider mode (FDEN = 1 and FDM = 1) stops the baud rate timer and nullifies the effect of bit BCON.R.

Register BG is a dual-function Baud-rate Generator/Reload register. Reading from BG returns the timer's contents, while writing to BG causes an auto-reload of its contents into the baud rate timer if BCON.R = 1. If BCON.R = 0 at the time a write operation to BG

occurs, the auto-reload action will be delayed until the first instruction cycle after setting BCON.R.

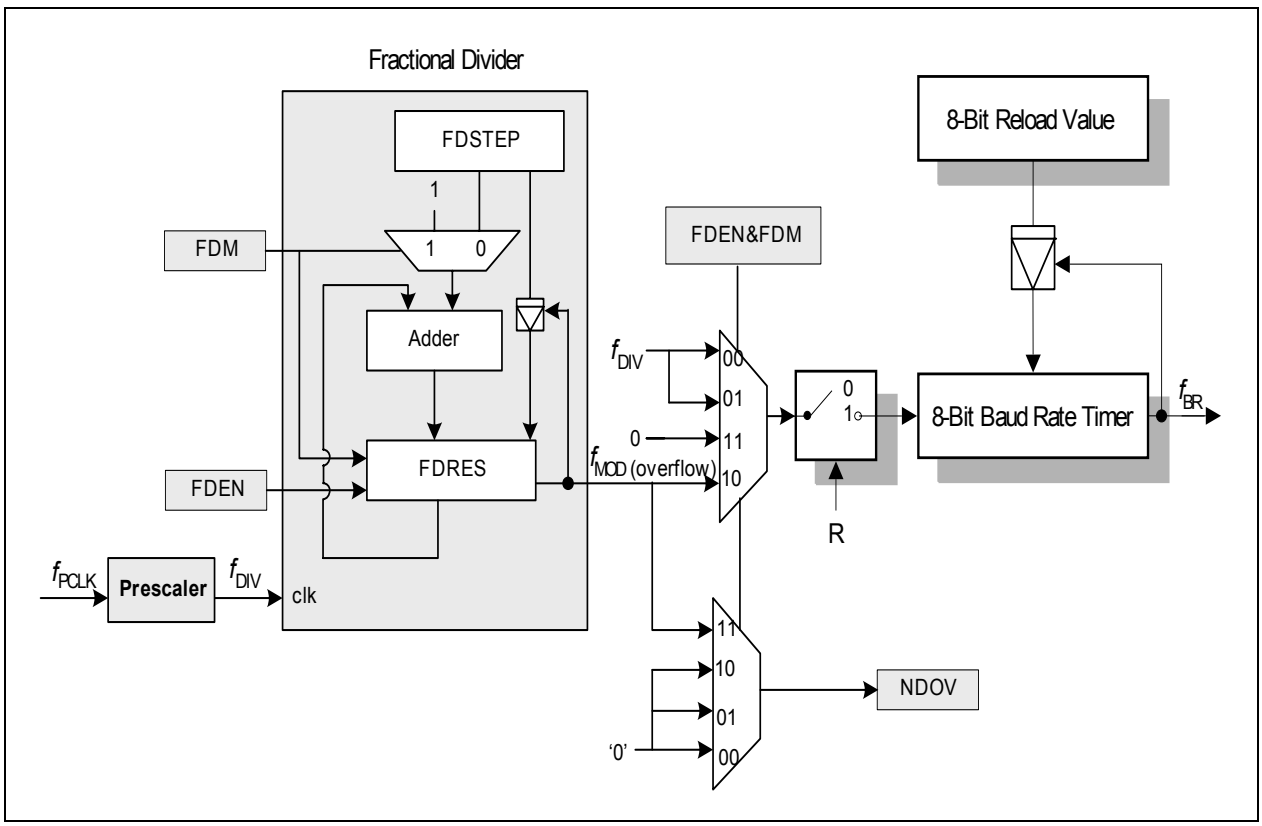



**Figure 12-3 Baud-rate Generator Circuitry**

The baud rate ($f_{BR}$) value is dependent on the following parameters:

- Input clock $f_{PCLK}$
- Prescaling factor ($2^{BRPRE}$) defined by bit field BRPRE in register BCON
- Fractional divider (STEP/256) defined by register FDSTEP
  (to be considered only if fractional divider is enabled and operating in fractional divider mode)
- 8-bit reload value (BR_VALUE) for the baud rate timer defined by register BG

The following formulas calculate the final baud rate without (see **Equation (12.3)**) and with the fractional divider (see **Equation (12.4)**), respectively:

(12.2)

$$\text{baud rate} = \frac{f_{PCLK}}{16 \times 2^{BRPRE} \times (BR_VALUE + 1)} \quad \text{where } 2^{BRPRE} \times (BR_VALUE + 1) > 1$$

(12.3)

$$\text{baud rate} = \frac{f_{PCLK}}{16 \times 2^{BRPRE} \times (BR_VALUE + 1)} \times \frac{STEP}{256}$$

The maximum baud rate that can be generated is limited to $f_{PCLK}/32$. Hence, for a module clock of 24 MHz, the maximum achievable baud rate is 0.75 MBaud.

Standard LIN protocol can support a maximum baud rate of 20 kHz, the baud rate accuracy is not critical and the fractional divider can be disabled. Only the prescaler is used for auto baud rate calculation. For LIN fast mode, which supports the baud rate of 20 kHz to 57.6 kHz, the higher baud rates require the use of the fractional divider for greater accuracy.

**Table 12-2** lists the various commonly used baud rates with their corresponding parameter settings and deviation errors. The fractional divider is disabled and a module clock of 24 MHz is used.

**Table 12-2 Typical Baud rates for UART with Fractional Divider disabled**

| Baud rate | Prescaling Factor ($2^{BRPRE}$) | Reload Value (BR_VALUE + 1) | Deviation Error |
|---|---|---|---|
| 19.2 kBaud | 1 (BRPRE=$000_B$) | 78 ($4E_H$) | 0.17 % |
| 9600 Baud | 1 (BRPRE=$000_B$) | 156 ($9C_H$) | 0.17 % |
| 4800 Baud | 2 (BRPRE=$001_B$) | 156 ($9C_H$) | 0.17 % |
| 2400 Baud | 4 (BRPRE=$010_B$) | 156 ($9C_H$) | 0.17 % |

The fractional divider allows baud rates of higher accuracy (lower deviation error) to be generated. **Table 12-3** lists the resulting deviation errors from generating a baud rate of 57.6 kHz, using different module clock frequencies. The fractional divider is enabled (fractional divider mode) and the corresponding parameter settings are shown.

**Table 12-3 Deviation Error for UART with Fractional Divider enabled**

| $f_{PCLK}$ | Prescaling Factor ($2^{BRPRE}$) | Reload Value (BR_VALUE + 1) | STEP | Deviation Error |
|---|---|---|---|---|
| 26.67 MHz | 1 | 20 ($14_H$) | 177 ($B1_H$) | +0.03 % |
| 24 MHz | 1 | 6 ($6_H$) | 59 ($3B_H$) | +0.03 % |
| 16 MHz | 1 | 4 ($4_H$) | 59 ($3B_H$) | +0.03 % |
| 13.33 MHz | 1 | 10 ($A_H$) | 177 ($B1_H$) | +0.03 % |
| 12 MHz | 1 | 3 ($3_H$) | 59 ($3B_H$) | +0.03 % |
| 8 MHz | 1 | 2 ($2_H$) | 59 ($3B_H$) | +0.03 % |
| 6.67 MHz | 1 | 5 ($5_H$) | 177 ($B1_H$) | +0.03 % |
| 6 MHz | 1 | 6 ($6_H$) | 236 ($EC_H$) | +0.03 % |

### Fractional Divider

The input clock $f_{DIV}$ to the 8-bit fractional divider is scaled either by a factor of 1/n, or n/256 to generate an output clock $f_{MOD}$ for the baud rate timer. The fractional divider has two operating modes:

- Fractional divider mode
- Normal divider mode

### Fractional Divider Mode

The fractional divider mode is selected by clearing bit FDM in register FDCON to 0. Once the fractional divider is enabled (FDEN = 1), the output clock $f_{MOD}$ of the fractional divider is derived from scaling its input clock $f_{DIV}$ by a factor of n/256, where n is defined by bit field STEP in register FDSTEP and can take any value from 0 to 255.

In fractional divider mode, the output clock pulse $f_{MOD}$ is dependent on the result of the addition FDRES.RESULT + FDSTEP.STEP; if the addition leads to an overflow over $FF_H$, a pulse is generated for $f_{MOD}$.

The average output frequency in fractional divider mode is derived as follows:

(12.4)

$$f_{MOD} = f_{DIV} x \frac{STEP}{256} \qquad \text{where } STEP = 0 - 255$$

**Figure 12-4** shows the operation in fractional divider mode with a reload value of STEP = $8D_H$ (factor of 141/256 = 0.55).

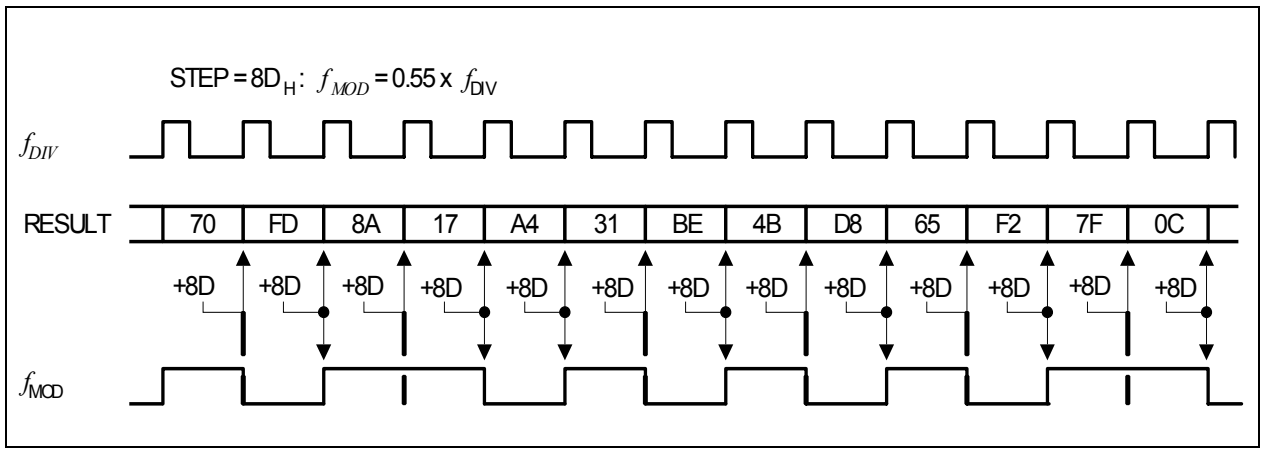



**Figure 12-4 Fractional Divider Mode Timing**

*Note: In fractional divider mode, $f_{MOD}$ will have a maximum jitter of one $f_{DIV}$ clock period.*

In general, the fractional divider mode can be used to generate an average output clock frequency with higher accuracy than the normal divider mode.

### Normal Divider Mode

Setting bit FDM in register FDCON to 1 configures the fractional divider to normal divider mode, while at the same time disables baud rate generation (see **Figure 12-3**). Once the fractional divider is enabled (FDEN = 1), it functions as an 8-bit auto-reload timer (with no relation to baud rate generation) and counts up from the reload value with each input clock pulse. Bit field RESULT in register FDRES represents the timer value, while bit field STEP in register FDSTEP defines the reload value. At each timer overflow, an overflow flag (FDCON.NDOV) will be set and an interrupt request will be generated if NDOVEN bit is set. This bit is located in BCON register for UART while it is located in SCON1 register for UART1. This gives an output clock $f_{MOD}$ that is 1/n of the input clock $f_{DIV}$, where n is defined by 256 - STEP.

The output frequency in normal divider mode is derived as follows:

(12.5)

$$f_{MOD} = f_{DIV} \times \frac{1}{256 - STEP}$$

**Figure 12-5** shows the operation in normal divider mode with a reload value of STEP = $FD_H$. In order to get $f_{MOD} = f_{DIV}$, STEP must be programmed with $FF_H$.

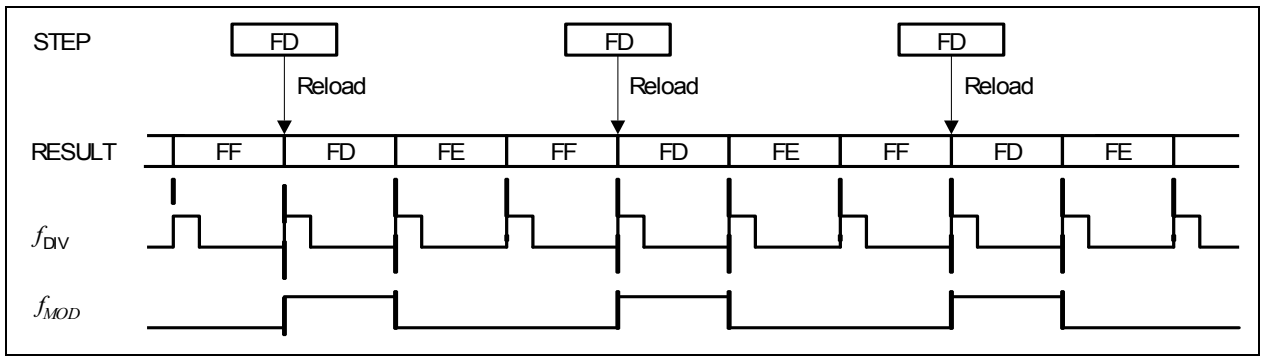



**Figure 12-5 Normal Mode Timing**

### Baud Rate Generator Registers

Both UART and UART1 module baud rate generators contain the five SFRs, BG, BCON, FDCON, FDSTEP and FDRES. The functionality of these registers are described in the following pages.

Register BCON contains the control bits for the baud-rate generator and the prescaling factor.

**BCON**
**Baud Rate Control Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| BGSEL | | NDOVEN | BRDIS | BRPRE | | | R |
| rw | | rw | rw | rw | | | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **R** | 0 | rw | **Baud-rate Generator Run Control**<br>0 Baud-rate generator is disabled.<br>1 Baud-rate generator is enabled.<br>*Note: BR_VALUE should only be written if R = 0.* |
| **BRPRE** | [3:1] | rw | **Prescaler Select**<br>000 $f_{DIV} = f_{PCLK}$<br>001 $f_{DIV} = f_{PCLK}/2$<br>010 $f_{DIV} = f_{PCLK}/4$<br>011 $f_{DIV} = f_{PCLK}/8$<br>100 $f_{DIV} = f_{PCLK}/16$<br>101 $f_{DIV} = f_{PCLK}/32$<br>Others: reserved |

| Field | Bits | Type | Description |
|---|---|---|---|
| **BRDIS** | 4 | rw | **Break/Synch Detection Disable**<br>0 Break/Synch detection is enabled.<br>1 Break/Synch detection is disabled. |
| **NDOVEN** | 5 | rw | **Normal Divider Overflow Enable Bit**<br>0 Normal divider overflow interrupt is disabled.<br>1 Normal divider overflow interrupt is enabled. |
| **BGSEL** | [7:6] | rw | **Baud Rate Select for Detection**<br>For different values of BGSEL, the baud rate range for detection is defined by the following formula:<br>$f_{PCLK}$/(2184*2^BGSEL)< baud rate range< $f_{PCLK}$/(72*2^BGSEL)<br>where BGSEL =$00_B$, $01_B$, $10_B$, $11_B$.<br>See **Table 12-4** for bit field BGSEL definition for different input frequencies. |

*Note: Bits BRDIS, BGSEL and NDOVEN are used only in UART module and not in UART1 module. Therefore, they should always be written with 0 in the BCON register in UART1 module. Setting them to 1 in the UART1 register has no effect*

*Note: Please take note that bit NDOVEN for UART1 is located in SCON1 register.*

**Table 12-4 BGSEL Bit Field Definition for Different Input Frequencies**

| $f_{PCLK}$ | BGSEL | Baud Rate Select for Detection $f_{PCLK}$/(2184*2^BGSEL) to $f_{PCLK}$/(72*2^BGSEL) |
|---|---|---|
| 24 MHz | $00_B$ | 11 kHz to 333.3 kHz |
| | $01_B$ | 5.5 kHz to 166.6 kHz |
| | $10_B$ | 2.8 kHz to 83.3 kHz |
| | $11_B$ | 1.4 kHz to 41.6 kHz |
| 12 MHz | $00_B$ | 5.5 kHz to 166.6 kHz |
| | $01_B$ | 2.8 kHz to 83.3 kHz |
| | $10_B$ | 1.4 kHz to 41.6 kHz |
| | $11_B$ | 0.7 kHz to 20.8 kHz |

**Table 12-4 BGSEL Bit Field Definition for Different Input Frequencies** (cont'd)

| $f_{PCLK}$ | BGSEL | Baud Rate Select for Detection $f_{PCLK}$/(2184*2^BGSEL) to $f_{PCLK}$/(72*2^BGSEL) |
|---|---|---|
| 2 MHz | $00_B$ | 0.92 kHz to 27.7 kHz |
| | $01_B$ | 0.46 kHz to 13.8 kHz |
| | $10_B$ | 0.23 kHz to 6.9 kHz |
| | $11_B$ | 0.12 kHz to 3.4 kHz |

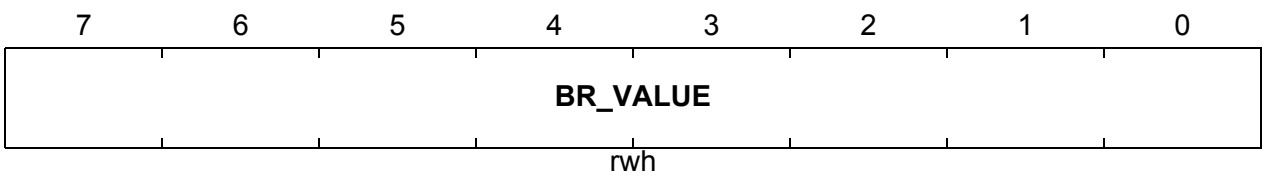
When $f_{PCLK}$=24 MHz, the baud rate range between 1.4 kHz to 333.3 kHz can be detected. In order to increase the detection accuracy of the baud rate, the following examples serve as a guide to select BGSEL value:

- If the baud rate falls in the range of 1.4 kHz to 2.8 kHz, selected BGSEL value is "$11_B$".
- If the baud rate falls in the range of 2.8 kHz to 5.5 kHz, selected BGSEL value is "$10_B$".
- If the baud rate falls in the range of 5.5 kHz to 11 kHz, selected BGSEL value is "$01_B$".
- If the baud rate falls in the range of 11 kHz to 333.3 kHz, selected BGSEL value is "$00_B$". If the baud rate is 20kHz, the possible values of BGSEL that can be selected are "$00_B$", "$01_B$", "$10_B$", and "$11_B$". However, it is advisable to select "$00_B$" for better detection accuracy.

The baud rate can also be detected when the system is in the slow-down mode. For detection of the standard LIN baud rate, the required minimum $f_{PCLK}$ is 2 MHz, for which the baud rate range that can be detected is between 0.12 kHz to 27.7 kHz.

Register BG contains the 8-bit reload value for the baud rate timer.

**BG**
**Baud Rate Timer/Reload Register** **Reset Value: $00_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **BR_VALUE** | [7:0] | rwh | **Baud rate Timer/Reload Value**<br>Reading returns the 8-bit content of the baud rate timer; writing loads the baud rate timer/reload value.<br>*Note: BG should only be written if R = 0.* |

Register FDCON contains the control and status bits for the fractional divider, and also the status flags used in LIN protocol support (see **Section 12.2.1**).

**FDCON**
**Fractional Divider Control Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **BGS** | **SYNEN** | **ERRSYN** | **EOFSYN** | **BRK** | **NDOV** | **FDM** | **FDEN** |
| rw | rw | rwh | rwh | rwh | rwh | rw | rw |

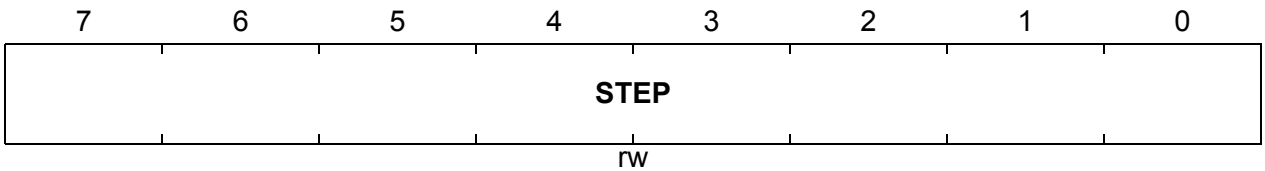
| Field | Bits | Type | Description |
|---|---|---|---|
| **FDEN** | 0 | rw | **Fractional Divider Enable Bit**<br>0 Fractional Divider is disabled, only prescaler is considered.<br>1 Fractional Divider is enabled. |
| **FDM** | 1 | rw | **Fractional Divider Mode Select**<br>0 Fractional Divider Mode is selected.<br>1 Normal Divider Mode is selected. |
| **NDOV** | 2 | rwh | **Overflow Flag in Normal Divider Mode**<br>This bit is set by hardware and can only be cleared by software.<br>0 Interrupt request is not active.<br>1 Interrupt request is active. |
| **BRK** | 3 | rwh | **Break Field Flag**<br>This bit is set by hardware and can only be cleared by software.<br>0 Break Field is not detected.<br>1 Break Field is detected. |
| **EOFSYN** | 4 | rwh | **End of SYN Byte Flag**<br>This bit is set by hardware and can only be cleared by software.<br>0 End of SYN Byte is not detected.<br>1 End of SYN Byte is detected. |
| **ERRSYN** | 5 | rwh | **SYN Byte Error Flag**<br>This bit is set by hardware and can only be cleared by software.<br>0 Error is not detected in SYN Byte.<br>1 Error is detected in SYN Byte. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **SYNEN** | 6 | rw | **End of SYN Byte and SYN Byte Error Interrupts Enable**<br>0 End of SYN Byte and SYN Byte Error Interrupts are not enabled.<br>1 End of SYN Byte and SYN Byte Error Interrupts are enabled. |
| **BGS** | 7 | rw | **Baud-rate Generator Select**<br>0 Baud-rate generator is selected.<br>1 Timer 1 is selected. |

*Note: Bits 3 to 7 are used only in UART module and not in UART1 module. Therefore, they should always be written with 0 in the FDCON register in UART1 module. Setting them to 1 in the UART1 register has no effect.*

Register FDSTEP contains the 8-bit STEP value for the fractional divider.

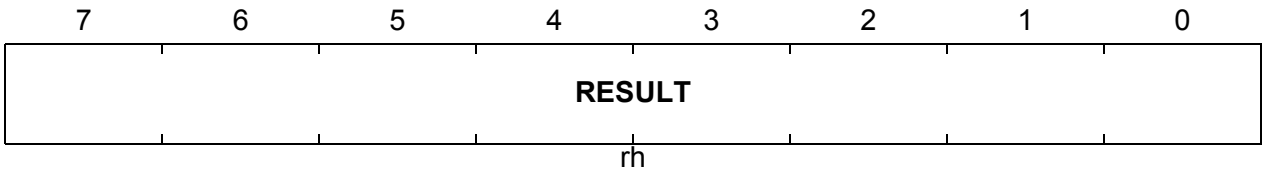
**FDSTEP**
**Fractional Divider Reload Register** **Reset Value: $00_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **STEP** | [7:0] | rw | **STEP Value**<br>In normal divider mode, STEP contains the reload value for RESULT.<br>In fractional divider mode, this bit field defines the 8-bit value that is added to the RESULT with each input clock cycle. |

Register FDRES contains the 8-bit RESULT value for the fractional divider.

**FDRES**
**Fractional Divider Result Register** **Reset Value: $00_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **RESULT** | [7:0] | rh | **RESULT Value**<br>In normal divider mode, RESULT acts as reload counter (addition +1).<br>In fractional divider mode, this bit field contains the result of the addition RESULT+STEP.<br>If FDEN bit is changed from “0” to “1”, RESULT is loaded with FF. |

### 12.1.4.3 Timer 1

In modes 1 and 3 of UART module, Timer 1 can be used for generating the variable baud rates. In theory, this timer could be used in any of its modes. But in practice, it should be set into auto-reload mode (Timer 1 mode 2), with its high byte set to the appropriate value for the required baud rate. The baud rate is determined by the Timer 1 overflow rate and the value of SMOD as follows:

(12.6)

$$\text{Mode 1, 3 baud rate} = \frac{2^{\text{SMOD}} \text{ x } f_{\text{PCLK}}}{32 \text{ x } 2 \text{ x } (256 - \text{TH1})}$$

Alternatively, for a given baud rate, the value of Timer 1 high byte can be derived:

(12.7)

$$\text{TH1} = 256 - \frac{2^{\text{SMOD}} \text{ x } f_{\text{PCLK}}}{32 \text{ x } 2 \text{ x Mode 1, 3 baud rate}}$$

*Note: Timer 1 can neither indicate an overflow nor generate an interrupt if Timer 0 is in mode 3; Timer 1 is halted while Timer 0 takes over the use of its control bits and overflow flag. Hence, the baud rate supplied to the UART module is defined by Timer 0 and not Timer 1. User should avoid using Timer 0 and Timer 1 in mode 3 for baud rate generation.*

*Note: Timer 1 cannot be used to generate the variable baud rate in UART1.*

### 12.1.5 Port Control

The UART modules shift in data through RXD which can be selected from three different sources, RXD_0, RXD_1 and RXD_2. This selection is performed by the SFR bits MODPISEL.URRIS and MODPISEL.URRISH in UART module, and MODPISEL1.UR1RIS in UART1 module.

**MODPISEL**
**Peripheral Input Select Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **0** | **URRISH** | **JTAGTDIS** | **JTAGTCKS** | **EXINT2IS** | **EXINT1IS** | **EXINT0IS** | **URRIS** |
| r | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **URRISH, URRIS** | 6,0 | rw | **UART Receive Input Select [6,0]**<br>00 UART Receiver Input RXD_0 is selected.<br>01 UART Receiver Input RXD_1 is selected.<br>10 UART Receiver Input RXD_2 is selected.<br>11 Reserved |
| **0** | 7 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

**MODPISEL1**
**Peripheral Input Select Register 1** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **EXINT6IS** | | | **UR1RIS** | | **T21EXIS** | **0** | |
| rw | | | rw | | rw | r | |

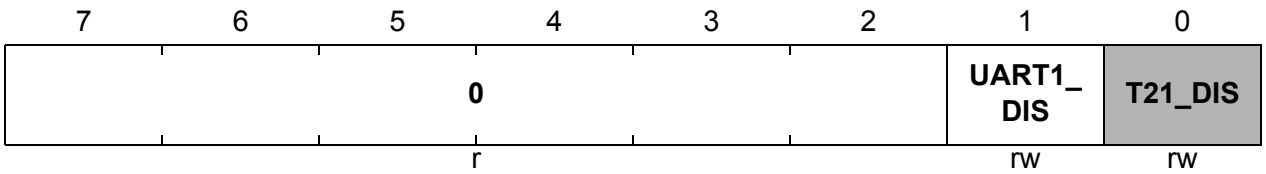
| Field | Bits | Type | Description |
|---|---|---|---|
| **UR1RIS** | [4:3] | rw | **UART1 Receive Input Select**<br>00 UART1 Receiver Input RXD_0 is selected.<br>01 UART1 Receiver Input RXD_1 is selected.<br>10 UART1 Receiver Input RXD_2 is selected.<br>11 Reserved |

| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | [1:0] | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

### 12.1.6 Low Power Mode

If the UART1 module functionality is not required at all, it can be completely disabled by gating off its clock input for maximal power reduction. This is done by setting bit UART1_DIS in register PMCON2 as described below. Refer to **Chapter 8.1.4** for details on peripheral clock management.

**PMCON2**
**Power Mode Control Register 1** **Reset Value: $00_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| UART1_DIS | 1 | rw | UART1 Module Disable Request. Active high.<br>0 UART1 module is in normal operation (default).<br>1 Request to disable the UART1 module. |
| **0** | [7:2] | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

*Note: The Low Power Mode option is not available in UART module.*

### 12.1.7 Register Map

All UART1 module register names described in the previous sections are referenced in other chapters of this document with the module name prefix “UART1_”, e.g., UART1_SCON. However, all UART module registers are not referenced by any prefix.

Besides the SCON and SBUF registers, which can be accessed from both the standard (non-mapped) and mapped SFR area, the rest of the UART module’s SFRs are located in SCU page 0 of the standard area. The UART1 module SFRs are all located in the mapped SFR area.

**Table 12-5** lists the addresses of these SFRs.

**Table 12-5 UART Module SFR Address List**

| UART Module | | UART1 Module | |
|---|---|---|---|
| **Address** | **Register** | **Address** | **Register** |
| $98_H$ | SCON | $C8_H$ | SCON |
| $99_H$ | SBUF | $C9_H$ | SBUF |
| $BD_H$ | BCON | $CA_H$ | BCON |
| $BE_H$ | BG | $CB_H$ | BG |
| $E9_H$ | FDCON | $CC_H$ | FDCON |
| $EA_H$ | FDSTEP | $CD_H$ | FDSTEP |
| $EB_H$ | FDRES | $CE_H$ | FDRES |
| | | $CF_H$ | SCON1 |

## 12.2 LIN

The UART module can be used to support the Local Interconnect Network (LIN) protocol for both master and slave operations. The LIN baud rate detection feature provides the capability to detect the baud rate within LIN protocol using Timer 2. This allows the UART module to be synchronized to the LIN baud rate for data transmission and reception.

### 12.2.1 LIN Protocol

LIN is a holistic communication concept for local interconnected networks in vehicles. The communication is based on the SCI (UART) data format, a single-master/multiple-slave concept, a clock synchronization for nodes without stabilized time base. An attractive feature of LIN is self-synchronization of the slave nodes without a crystal or ceramic resonator, which significantly reduces the cost of hardware platform. Hence, the baud rate must be calculated and returned with every message frame.

The structure of a LIN frame is shown in **Figure 12-6**. The frame consists of the:

- header, which comprises a Break (13-bit time low), Synch Byte ($55_H$), and ID field
- response time
- data bytes (according to UART protocol)
- checksum

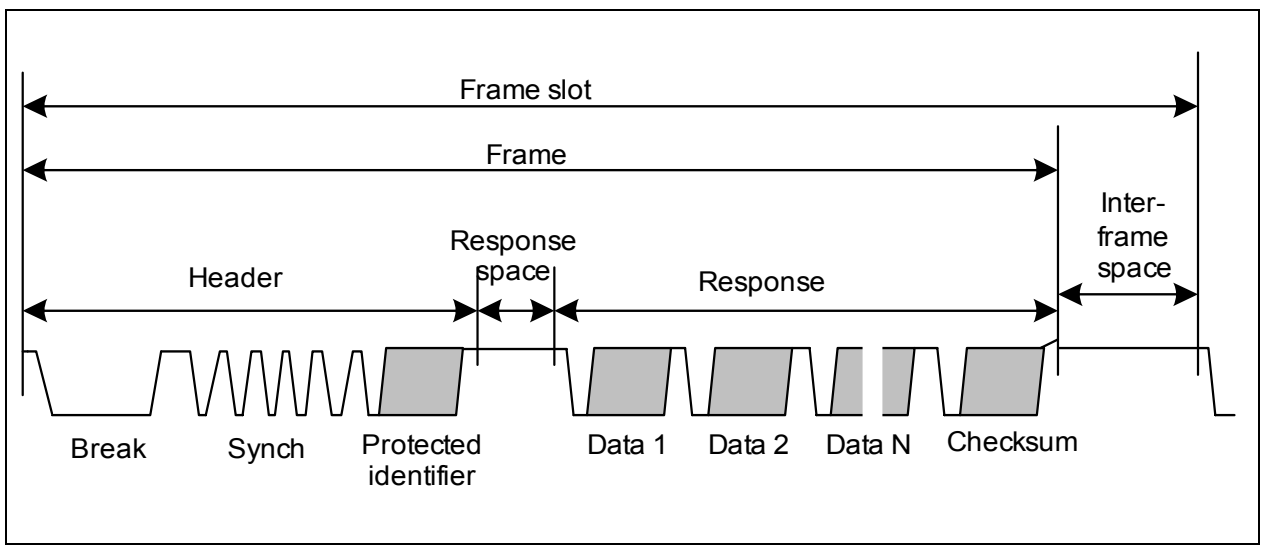



**Figure 12-6 The Structure of LIN Frame**

Each byte field is transmitted as a serial byte, as shown in **Figure 12-7**. The LSB of the data is sent first and the MSB is sent last. The start bit is encoded as a bit with value zero (dominant) and the stop bit is encoded as a bit with value one (recessive).

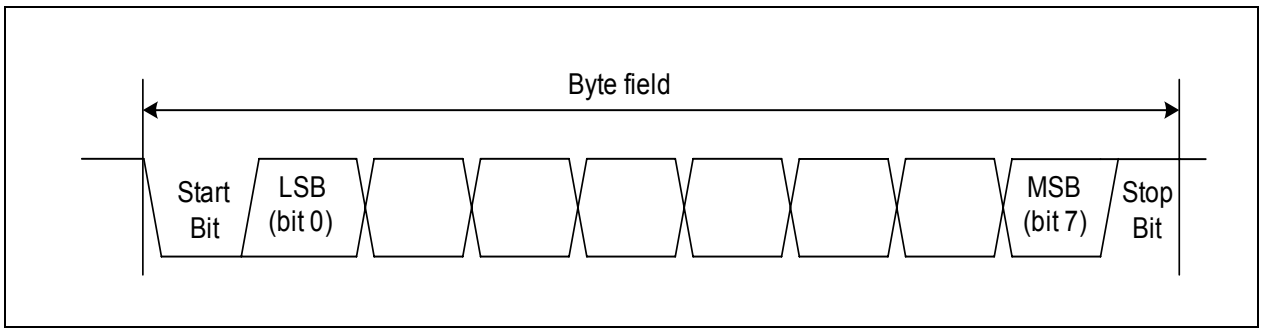



**Figure 12-7 The Structure of Byte Field**

The break is used to signal the beginning of a new frame. It is the only field that does not comply with **Figure 12-7**. A break is always generated by the master task (in the master mode) and it must be at least 13 bits of dominant value, including the start bit, followed by a break delimiter, as shown in **Figure 12-8**. The break delimiter will be at least one nominal bit time long.

A slave node will use a break detection threshold of 11 nominal bit times.




**Figure 12-8 The Break Field**

Synch Byte is a specific pattern for determination of time base. The byte field is with the data value $55_H$, as shown in **Figure 12-9**.

A slave task is always able to detect the Break/Synch sequence, even if it expects a byte field (assuming the byte fields are separated from each other). If this happens, detection of the Break/Synch sequence will abort the transfer in progress and processing of the new frame will commence.

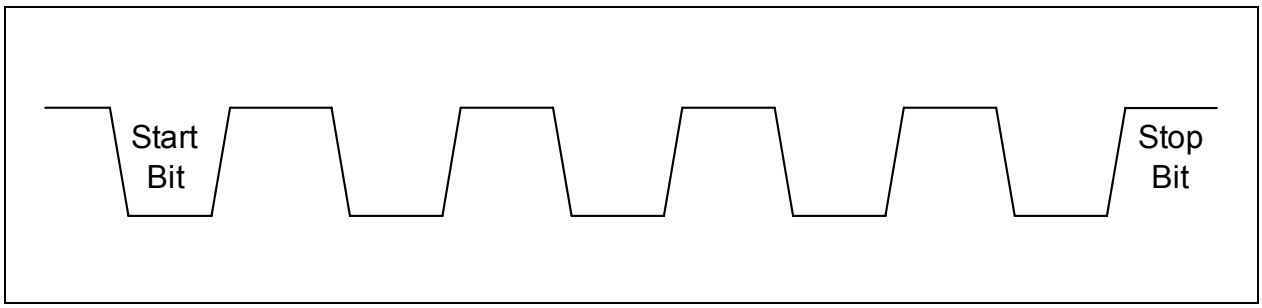



**Figure 12-9 The Synch Byte Field**

The slave task will receive and transmit data when an appropriate ID is sent by the master:

1. Slave waits for Synch Break
2. Slave synchronizes on Synch Byte
3. Slave snoops for ID
4. According to ID, slave determines whether to receive or transmit data, or do nothing
5. When transmitting, the slave sends 2, 4 or 8 data bytes, followed by check byte

### 12.2.2 LIN Header Transmission

LIN header transmission is only applicable in master mode. In the LIN communication, a master task decides when and which frame is to be transferred on the bus. It also identifies a slave task to provide the data transported by each frame. The information needed for the handshaking between the master and slave tasks is provided by the master task through the header portion of the frame.

The header consists of a break and synch pattern followed by an identifier. Among these three fields, only the break pattern cannot be transmitted as a normal 8-bit UART data. The break must contain a dominant value of 13 bits or more to ensure proper synchronization of slave nodes.

In the LIN communication, a slave task is required to be synchronized at the beginning of the protected identifier field of frame. For this purpose, every frame starts with a sequence consisting of a break field followed by a synch byte field. This sequence is unique and provides enough information for any slave task to detect the beginning of a new frame and be synchronized at the start of the identifier field.

#### 12.2.2.1 Automatic Synchronization to the Host

Upon entering LIN communication, a connection is established and the transfer speed (baud rate) of the serial communication partner (host) is automatically synchronized in the following steps that are to be included in user software:

STEP 1: Initialize interface for reception and timer for baud rate measurement

STEP 2: Wait for an incoming LIN frame from host

STEP 3: Synchronize the baud rate to the host

STEP 4: Enter for Master Request Frame or for Slave Response Frame

The next section, **Section 12.2.2.2**, provides some hints on setting up the microcontroller for baud rate detection of LIN.

*Note: Re-synchronization and setup of baud rate are always done for **every** Master Request Header or Slave Response Header LIN frame.*

### 12.2.2.2 Baud Rate Detection of LIN

The LIN baud rate detection feature provides the capability to detect the baud rate within the LIN protocol using Timer 2. Initialization consists of:

- Serial port of the microcontroller set to Mode 1 (8-bit UART, variable baud rate) for communication.
- Provide the baud rate range via bit field BCON.BGSEL.
- Timer 2 is set to capture mode with falling edge trigger at pin T2EX. Bit T2MOD.EDGESEL is set to 0 by default and bit T2CON.CP/$\overline{RL2}$ is set to 1.
- Timer 2 external events are enabled. T2CON. EXEN2 is set to 1. (EXF2 flag is set when a negative transition occurs at pin T2EX)
- $f_{T2}$ can be configured by bit field T2MOD.T2PRE.

The baud rate detection for LIN is shown in **Figure 12-10**, the Header LIN frame consists of the:

- SYN Break (13 bit times low)
- SYN byte ($55_H$)
- Protected ID field

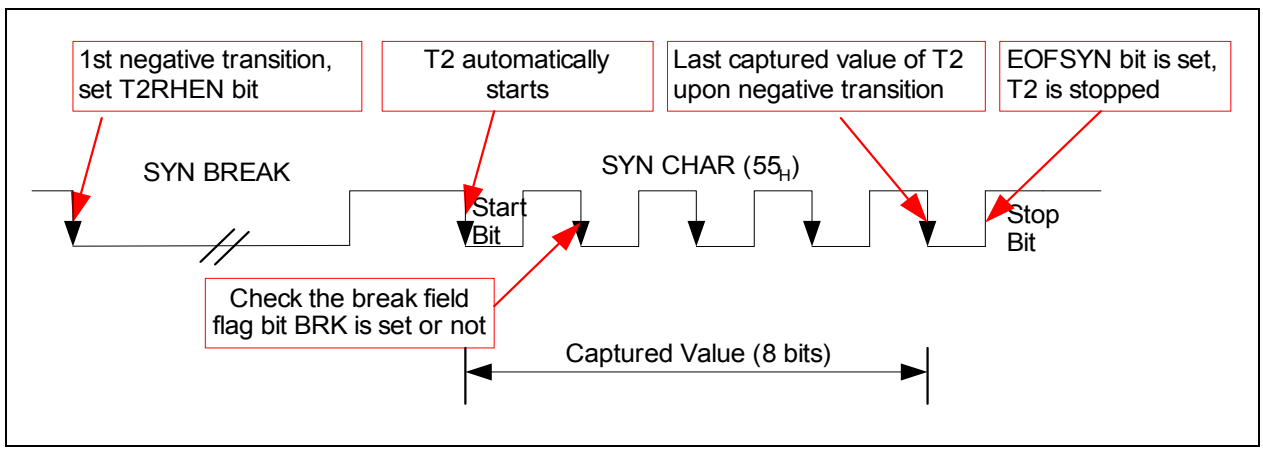



**Figure 12-10 LIN Auto Baud Rate Detection**

With the first falling edge:

- The Timer 2 External Start Enable bit (T2MOD.T2RHEN) is set. The falling edge at pin T2EX is selected by default for Timer 2 External Start (bit T2MOD.T2REGS is 0).

With the second falling edge:

- Start Timer 2 by the hardware.

With the third falling edge:

- Timer 2 captures the timing of 2 bits of SYN byte.
- Check the Break Field Flag bit FDCON.BRK.

If the Break Field Flag FDCON.BRK is set, software may continue to capture 4/6/8 bits of SYN byte. Finally, the End of SYN Byte Flag (FDCON.EOFSYN) is set, Timer 2 is

stopped. T2 Reload/Capture register (RC2H/L) is the time taken for 2/4/6/8 bits according to the implementation. Then the LIN routine calculates the actual baud rate, sets the PRE and BG values if the UART module uses the baud-rate generator for baud rate generation.

After the third falling edge, the software may discard the current operation and continue to detect the next header LIN frame if the following conditions were detected:

- The Break Field Flag FDCON.BRK is not set, or
- The SYN Byte Error Flag FDCON.ERRSYN is set, or
- The Break Field Flag FDCON.BRK is set, but the End of SYN Byte Flag FDCON.EOFSYN and the SYN Byte Error Flag FDCON.ERRSYN are not set.

## 12.3 High-Speed Synchronous Serial Interface

The SSC supports full-duplex and half-duplex synchronous communication. The serial clock signal can be generated by the SSC internally (master mode) using its own 16-bit baud-rate generator, or can be received from an external master (slave mode). Data width, shift direction, clock polarity and phase are programmable. This allows communication with SPI-compatible devices or devices using other synchronous serial interfaces.

Data is transmitted or received on lines TXD and RXD, which are normally connected to the pins MTSR (Master Transmit/Slave Receive) and MRST (Master Receive/Slave Transmit). The clock signal is output via line MS_CLK (Master Serial Shift Clock) or input via line SS_CLK (Slave Serial Shift Clock). Both lines are normally connected to the pin SCLK. Transmission and reception of data are double-buffered.

**Figure 12-11** shows the block diagram of the SSC.

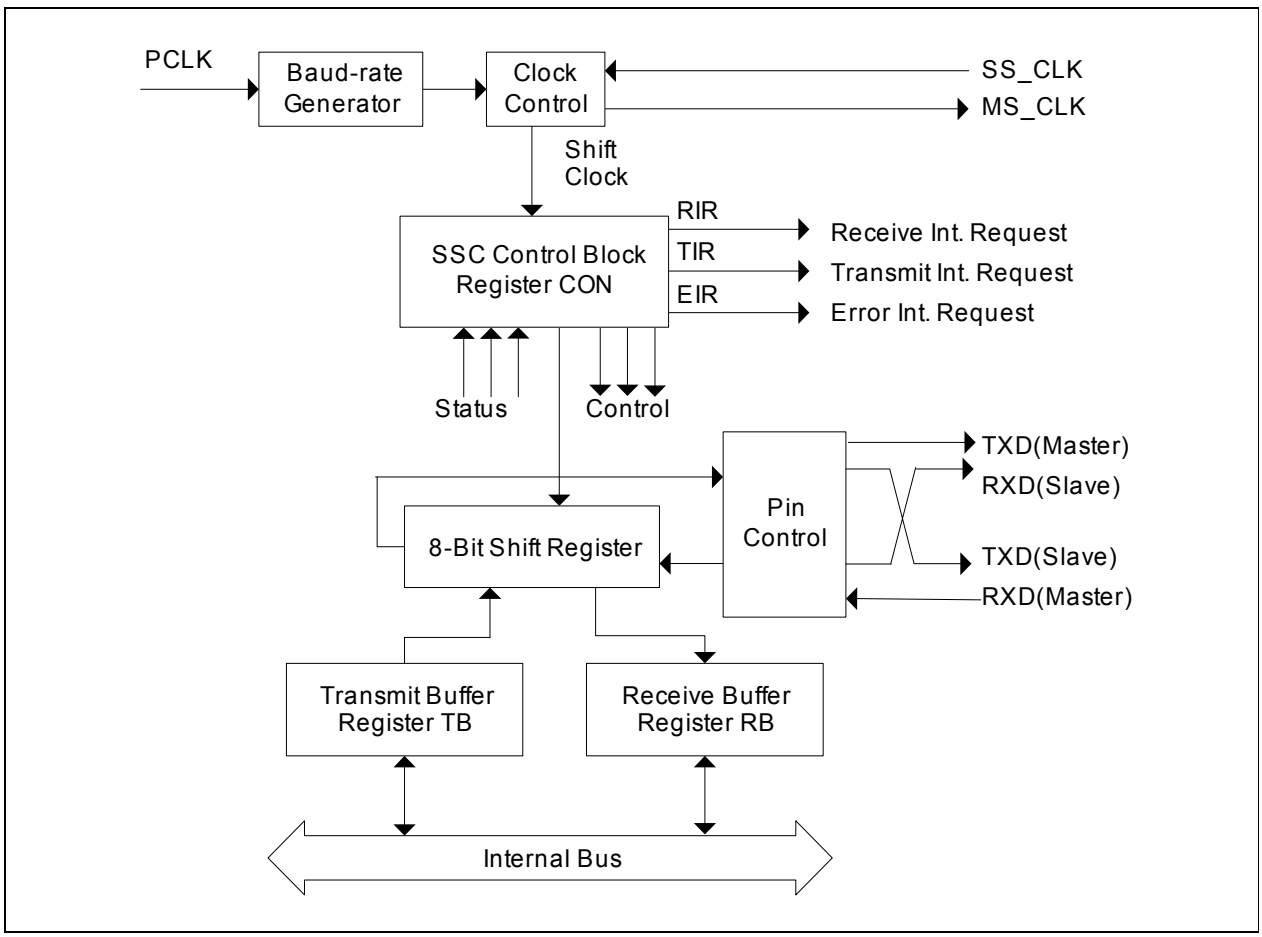



**Figure 12-11 Synchronous Serial Channel SSC Block Diagram**

## 12.3.1 General Operation

### 12.3.1.1 Operating Mode Selection

The operating mode of the serial channel SSC is controlled by its control register CON. This register has a double function:

- During programming (SSC disabled by CON.EN = 0), it provides access to a set of control bits
- During operation (SSC enabled by CON.EN = 1), it provides access to a set of status flags.

The shift register of the SSC is connected to both the transmit lines and the receive lines via the pin control logic. Transmission and reception of serial data are synchronized and take place at the same time, i.e., the same number of transmitted bits is also received. Transmit data is written into the Transmitter Buffer register (TB) and is moved to the shift register as soon as this is empty. An SSC master (CON.MS = 1) immediately begins transmitting, while an SSC slave (CON.MS = 0) will wait for an active shift clock. When the transfer starts, the busy flag CON.BSY is set and the Transmit Interrupt Request line (TIR) will be activated to indicate that register TB may be reloaded again. When the programmed number of bits (2...8) have been transferred, the contents of the shift register are moved to the Receiver Buffer register (RB) and the Receive Interrupt Request line (RIR) will be activated. If no further transfer is to take place (TB is empty), CON.BSY will be cleared at the same time. Software should not modify CON.BSY, as this flag is hardware controlled.

*Note: The SSC starts transmission and sets CON.BSY minimum two clock cycles after transmit data is written into TB. Therefore, it is not recommended to poll CON.BSY to indicate the start and end of a single transmission. Instead, interrupt service routine should be used if interrupts are enabled, or the interrupt flags IRCON1.TIR and IRCON1.RIR should be polled if interrupts are disabled.*

*Note: Only one SSC can be the master at a given time.*

The transfer of serial data bits can be programmed in a number of ways:

- The data width can be specified from 2 to 8 bits
- A transfer may start with either the LSB or the MSB
- The shift clock may be idle low or idle high
- The data bits may be shifted with the leading edge or the trailing edge of the shift clock signal
- The baud rate may be set within a certain range depending on the module clock
- The shift clock can be generated (MS_CLK) or can be received (SS_CLK)

These features allow the SSC to be adapted to a wide range of applications requiring serial data transfer.

The Data Width Selection supports the transfer of frames of any data length, from 2-bit “characters” up to 8-bit “characters”. Starting with the LSB (CON.HB = 0) allows communication with SSC devices in synchronous mode or with serial interfaces such as the one in 8051. Starting with the MSB (CON.HB = 1) allows operation compatible with the SPI interface.

Regardless of the data width selected and whether the MSB or the LSB is transmitted first, the transfer data is always right-aligned in registers TB and RB, with the LSB of the transfer data in bit 0 of these registers. The data bits are rearranged for transfer by the internal shift register logic. The unselected bits of TB are ignored; the unselected bits of RB will not be valid and should be ignored by the receiver service routine.

The Clock Control allows the transmit and receive behavior of the SSC to be adapted to a variety of serial interfaces. A specific shift clock edge (rising or falling) is used to shift out transmit data, while the other shift clock edge is used to latch in receive data. Bit CON.PH selects the leading edge or the trailing edge for each function. Bit CON.PO selects the level of the shift clock line in the idle state. Thus, for an idle-high clock, the leading edge is a falling one, a 1 - to - 0 transition (see **Figure 12-12**).

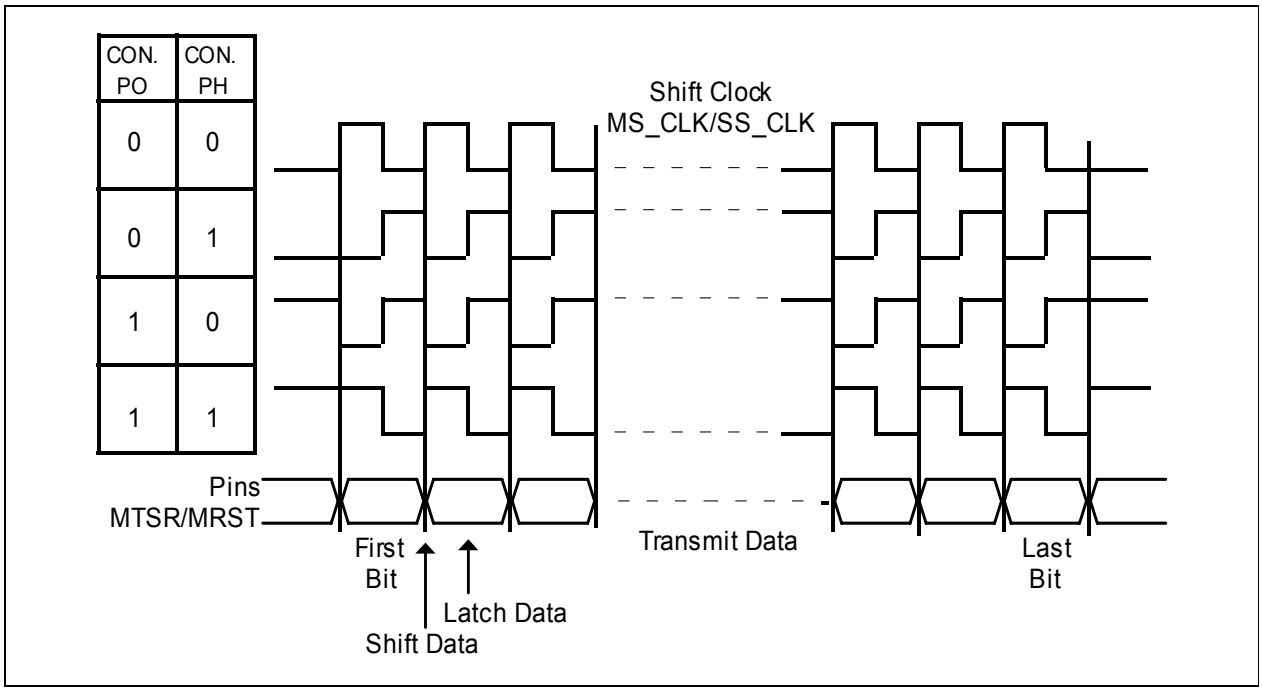



**Figure 12-12 Serial Clock Phase and Polarity Options**

When initializing the devices for serial communication, one device must be selected for master operation while all other devices must be programmed for slave operation.

### 12.3.1.2 Full-Duplex Operation

The various devices are connected through three lines. The definition of these lines is always determined by the master: the line connected to the master’s data output line

TXD is the transmit line; the receive line is connected to its data input line RXD; the shift clock line is either MS_CLK or SS_CLK. Only the device selected for master operation generates and outputs the shift clock on line MS_CLK. Since all slaves receive this clock, their pin SCLK must be switched to input mode. The external connections are hard-wired, and the function and direction of these pins are determined by the master or slave operation of the individual device.

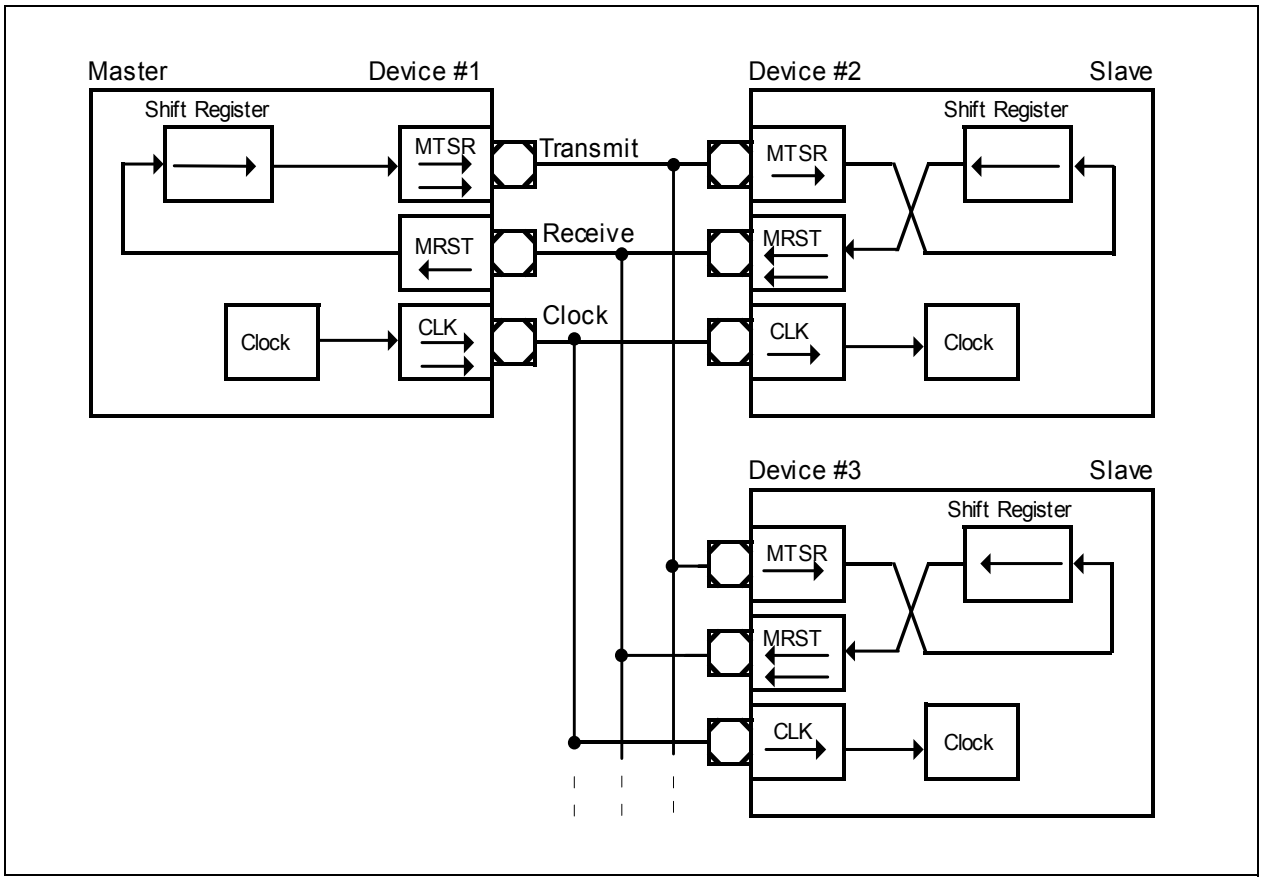



**Figure 12-13 SSC Full-Duplex Configuration**

The data output pins MRST of all slave devices are connected together onto the single receive line in the configuration shown in **Figure 12-13**. During a transfer, each slave shifts out data from its shift register. There are two ways to avoid collisions on the receive line due to different slave data:

- Only one slave drives the line, i.e., enables the driver of its MRST pin. All the other slaves must have their MRST pins programmed as input so only one slave can put its data onto the master's receive line. Only the receiving of data from the master is possible. The master selects the slave device from which it expects data either by separate select lines, or by sending a special command to this slave. The selected slave then switches its MRST line to output until it gets a de-selection signal or command.

- The slaves use open drain output on MRST. This forms a wired-AND connection. The receive line needs an external pull-up in this case. Corruption of the data on the receive line sent by the selected slave is avoided when all slaves not selected for transmission to the master send ones only. Because this high level is not actively driven onto the line, but only held through the pull-up device, the selected slave can pull this line actively to a low-level when transmitting a zero bit. The master selects the slave device from which it expects data either by separate select lines or by sending a special command to this slave.

After performing the necessary initialization of the SSC, the serial interfaces can be enabled. For a master device, the clock line will now go to its programmed polarity. The data line will go to either 0 or 1 until the first transfer starts. After a transfer, the data line will always remain at the logic level of the last transmitted data bit.

When the serial interfaces are enabled, the master device can initiate the first data transfer by writing the transmit data into register TB. This value is copied into the shift register (assumed to be empty at this time), and the selected first bit of the transmit data will be placed onto the TXD line on the next clock from the baud-rate generator (transmission starts only if CON.EN = 1). Depending on the selected clock phase, a clock pulse will also be generated on the MS_CLK line. At the same time, with the opposite clock edge, the master latches and shifts in the data detected at its input line RXD. This “exchanges” the transmit data with the receive data. Because the clock line is connected to all slaves, their shift registers will be shifted synchronously with the master’s shift register—shifting out the data contained in the registers, and shifting in the data detected at the input line.

With the start of the transfer, the busy flag CON.BSY is set and the TIR will be activated to indicate that register TB may be reloaded again. After the preprogrammed number of clock pulses (via the data width selection), the data transmitted by the master is contained in all the slaves’ shift registers, while the master’s shift register holds the data of the selected slave. In the master and all slaves, the contents of the shift register are copied into the receive buffer RB and the RIR is activated. If no further transfer is to take place (TB is empty), CON.BSY will be cleared at the same time. Software should not modify CON.BSY, as this flag is hardware controlled.

When configured as a slave device, the SSC will immediately output the selected first bit (MSB or LSB of the transfer data) at the output pin once the contents of the transmit buffer are copied into the slave's shift register. Bit CON.BSY is not set until the first clock edge at SS_CLK appears.

*Note: On the SSC, a transmission and a reception take place at the same time, regardless of whether valid data has been transmitted or received.*

*Note: The initialization of the CLK pin on the master requires some attention in order to avoid undesired clock transitions, which may disturb the other devices. Before the clock pin is switched to output via the related direction control register, the clock output level will be selected in the control register CON and the alternate output*

*be prepared via the related ALTSEL register, or the output latch must be loaded with the clock idle level.*

### 12.3.1.3 Half-Duplex Operation

In a half-duplex mode, only one data line is necessary for both receiving and transmitting of data. The data exchange line is connected to both the MTSR and MRST pins of each device, the shift clock line is connected to the SCLK pin.

The master device controls the data transfer by generating the shift clock, while the slave devices receive it. Due to the fact that all transmit and receive pins are connected to one data exchange line, serial data may be moved between arbitrary stations.

As in full-duplex mode, there are two ways to avoid collisions on the data exchange line:

- only the transmitting device may enable its transmit pin driver
- the non-transmitting devices use open drain output and send only ones.

Since the data inputs and outputs are connected together, a transmitting device will clock in its own data at the input pin (MRST for a master device, MTSR for a slave). By this method, any corruptions on the common data exchange line are detected if the received data is not equal to the transmitted data.

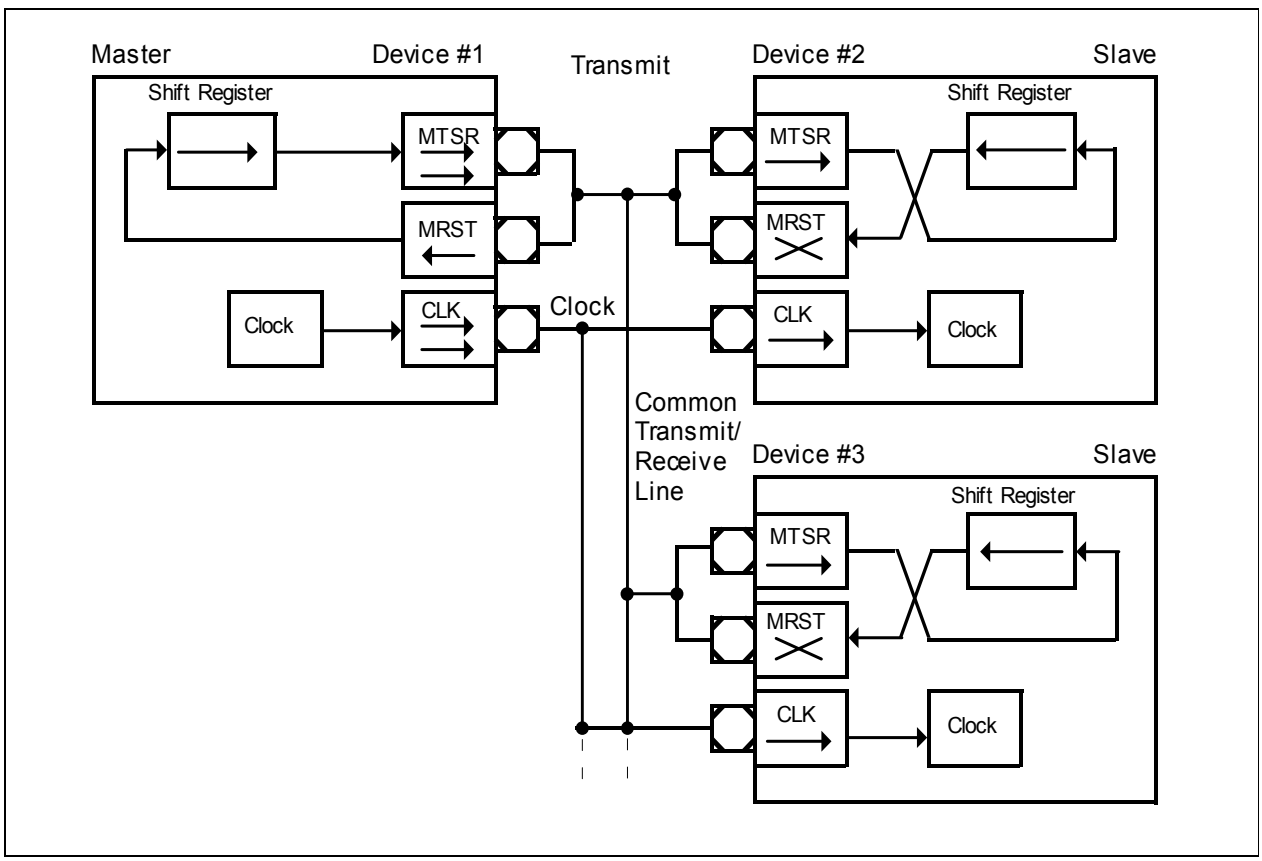



**Figure 12-14 SSC Half-Duplex Configuration**

#### 12.3.1.4 Continuous Transfers

When the transmit interrupt request flag is set, it indicates that the transmit buffer TB is empty and ready to be loaded with the next transmit data. If TB has been reloaded by the time the current transmission is finished, the data is immediately transferred to the shift register and the next transmission will start without any additional delay. On the data line, there is no gap between the two successive frames. For example, two byte transfers would look the same as one word transfer. This feature can be used to interface with devices that can operate with or require more than 8 data bits per transfer. It is just a matter of software specifying the total data frame length. This option can also be used to interface with byte-wide and word-wide devices.

*Note: This feature allows only multiples of the selected basic data width, because it would require disabling/enabling of the SSC to reprogram the basic data width on-the-fly.*

### 12.3.1.5 Port Control

The SSC uses three lines to communicate with the external world as shown in **Figure 12-5**. Pin SCLK serves as the clock line, while pins MRST and MTSR serve as the serial data input/output lines.

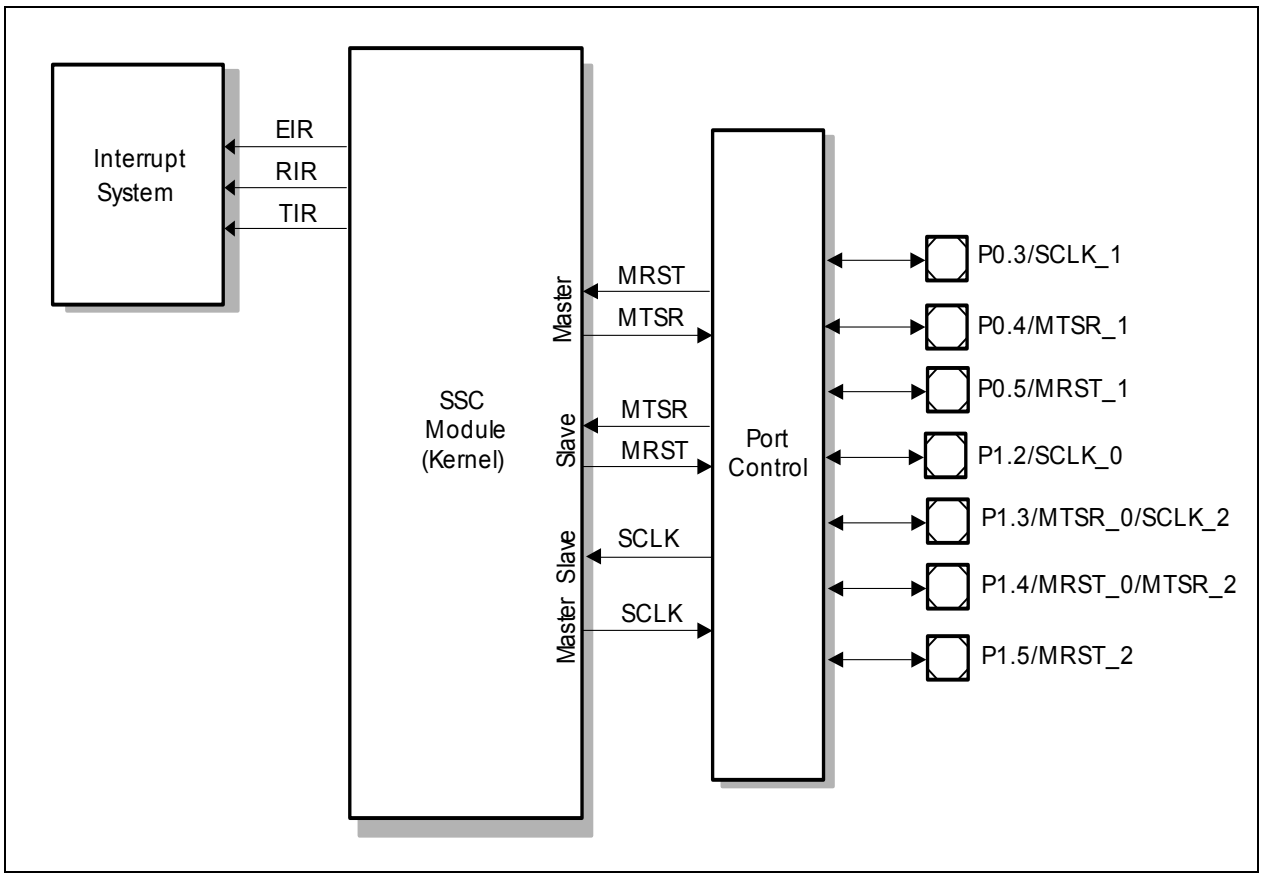
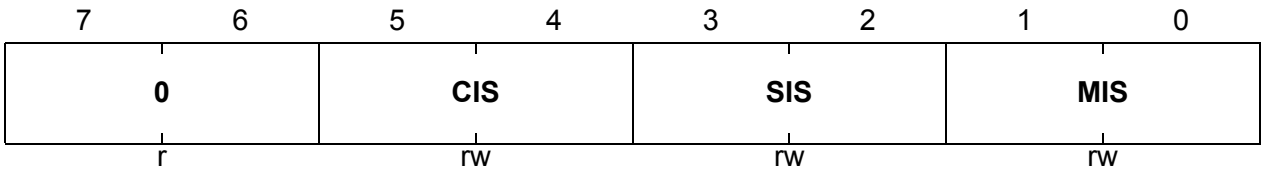



**Figure 12-15 SSC Module I/O Interface**

Operation of the SSC I/O lines depends on the selected operating mode (master or slave). The direction of the port lines depends on the operating mode. The SSC will automatically use the correct kernel output or kernel input line of the ports when switching modes.

Since the SSC I/O lines are connected with the bidirectional lines of the general purpose I/O ports, software I/O control is used to control the port pins assigned to these lines. The port registers must be programmed for alternate output and input selection. When switching between master and slave modes, port registers must be reprogrammed. As for switching between different receiver inputs, register MODPISEL3 is used.

**MODPISEL3**
**Peripheral Input Select Register 3** **Reset Value: $00_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **MIS** | [1:0] | rw | **Master Mode Receiver Input Select**<br>00 Receiver input 0 is selected.<br>01 Receiver input 1 is selected.<br>10 Receiver input 2 is selected.<br>11 Reserved |
| **SIS** | [3:2] | rw | **Slave Mode Receiver Input Select**<br>00 Receiver input 0 is selected.<br>01 Receiver input 1 is selected.<br>10 Receiver input 2 is selected.<br>11 Reserved |
| **CIS** | [5:4] | rw | **Slave Mode Clock Input Select**<br>00 Clock input 0 is selected.<br>01 Clock input 1 is selected.<br>10 Clock input 2 is selected.<br>11 Reserved |
| **0** | [7:6] | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

### 12.3.1.6 Baud Rate Generation

The serial channel SSC has its own dedicated 16-bit baud-rate generator with 16-bit reload capability, allowing baud rate generation independent of the timers. **Figure 12-16** shows the baud-rate generator.

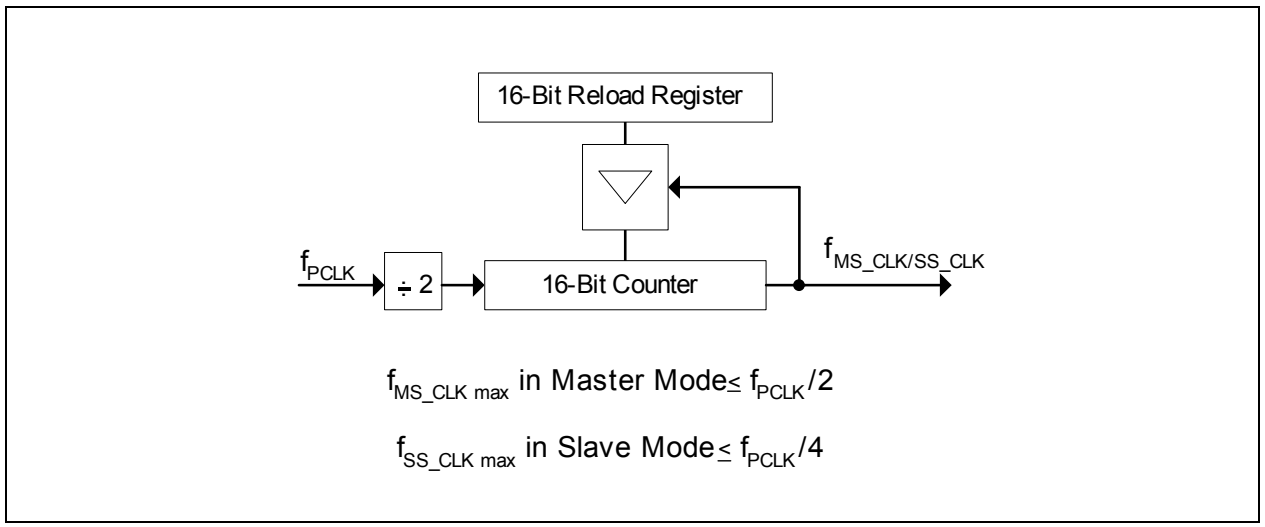



**Figure 12-16 SSC Baud-rate Generator**

The baud-rate generator is clocked with the module clock $f_{PCLK}$. The timer counts downwards. Register BR is the dual-function Baud-rate Generator/Reload register. Reading BR, while the SSC is enabled, returns the contents of the timer. Reading BR, while the SSC is disabled, returns the programmed reload value. In this mode, the desired reload value can be written to BR.

*Note: Never write to BR while the SSC is enabled.*

The formulas below calculate either the resulting baud rate for a given reload value, or the required reload value for a given baud rate:

$$\text{Baud rate} = \frac{f_{PCLK}}{2 \times (<BR> + 1)} \qquad BR = \frac{f_{PCLK}}{2 \times \text{Baud rate}} - 1$$

<BR> represents the contents of the reload register, taken as an unsigned 16-bit integer, while baud rate is equal to $f_{MS_CLK/SS_CLK}$ as shown in **Figure 12-16**.

The maximum baud rate that can be achieved when using a module clock of 24 MHz is 12 MBaud in master mode (with <BR> = $0000_H$) or 6 MBaud in slave mode (with <BR> = $0001_H$).

**Table 12-6** lists some possible baud rates together with the required reload values and the resulting deviation errors, assuming a module clock frequency of 24 MHz.

**Table 12-6 Typical Baud Rates of the SSC ($f_{hw_clk}$ = 24 MHz)**

| Reload Value | Baud Rate (= $f_{MS_CLK/SS_CLK}$) | Deviation |
|---|---|---|
| $0000_H$ | 12 MBaud (only in Master mode) | 0.0% |
| $0001_H$ | 6 MBaud | 0.0% |
| $0008_H$ | 1.3 MBaud | 0.0% |
| $000B_H$ | 1 MBaud | 0.0% |
| $000F_H$ | 750 kBaud | 0.0% |
| $0011_H$ | 666.7 kBaud | 0.0% |
| $0013_H$ | 600 kBaud | 0.0% |
| $0017_H$ | 500 kBaud | 0.0% |
| $002C_H$ | 266.7 kBaud | 0.0% |
| $003B_H$ | 200 kBaud | 0.0% |
| $0059_H$ | 133.3 kBaud | 0.0% |
| $0077_H$ | 100 kBaud | 0.0% |
| $FFFF_H$ | 183.11 Baud | 0.0% |

### 12.3.1.7 Error Detection Mechanisms

The SSC is able to detect four different error conditions. Receive Error and Phase Error are detected in all modes; Transmit Error and Baud Rate Error apply only to slave mode. When an error is detected, the respective error flag is/can be set and an error interrupt request will be generated by activating the Error Interrupt Request line (EIR) (see **Figure 12-17**). The error interrupt handler may then check the error flags to determine the cause of the error interrupt. The error flags are not reset automatically, but rather must be cleared by software after servicing. This allows servicing of error conditions to be done via interrupt if their enable bits are set, or via polling by software if their enable bits are not set.

*Note: The error interrupt handler must clear the associated (enabled) error flag(s) to prevent repeated interrupt requests.*

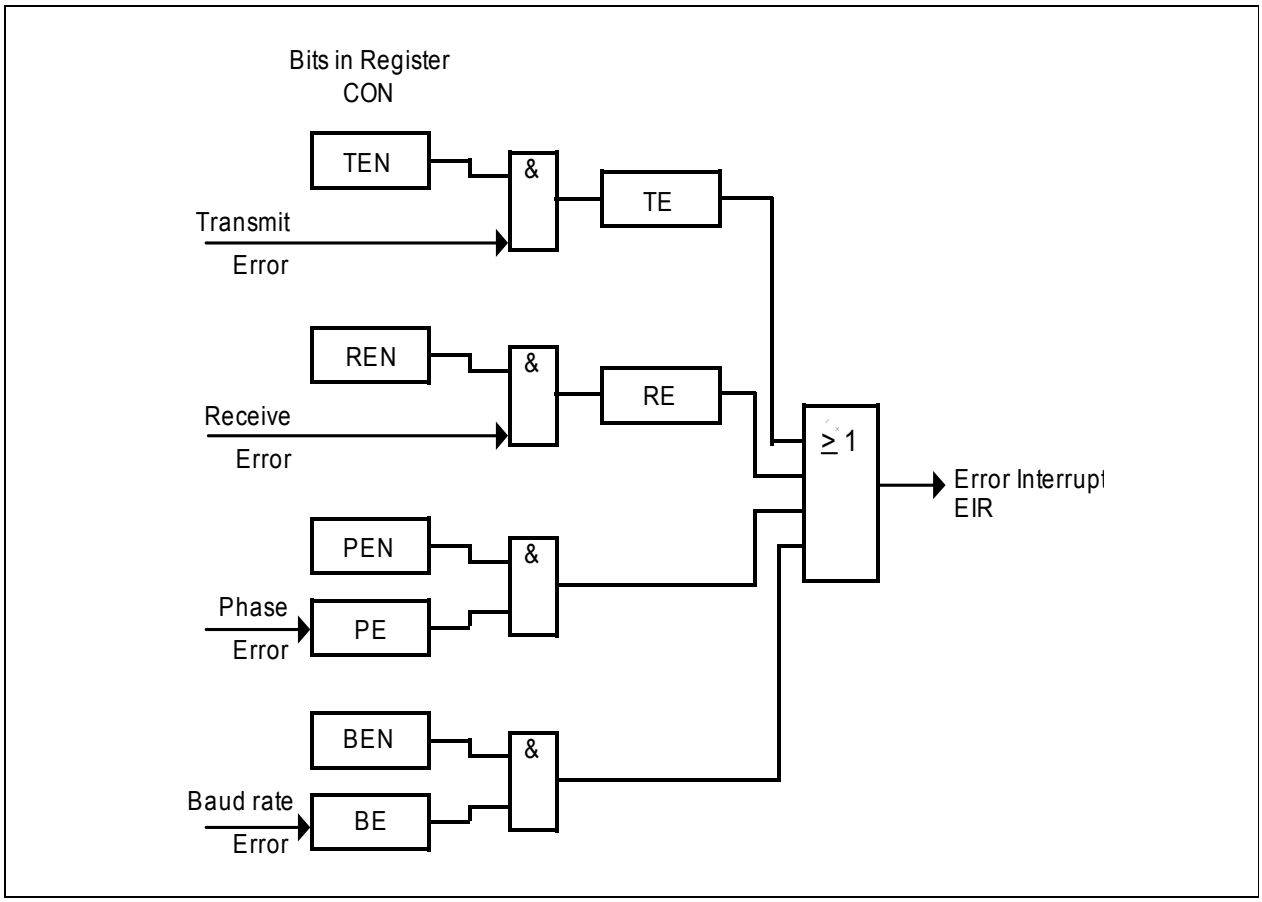



**Figure 12-17 SSC Error Interrupt Control**

A **Receive Error** (master or slave mode) is detected when a new data frame is completely received, but the previous data was not read out of the register RB. This condition sets the error flag CON.RE and the EIR, when enabled via CON.REN. The old data in the receive buffer RB will be overwritten with the new value and this lost data is irretrievable.

A **Phase Error** (master or slave mode) is detected when the incoming data at pin MRST (master mode) or MTSR (slave mode), sampled with the same frequency as the module clock, changes between one cycle before and two cycles after the latching edge of the shift clock signal SCLK. This condition sets the error flag CON.PE and, when enabled via CON.PEN, sets the EIR.

*Note: When receiving and transmitting data in parallel, phase error occurs if the baud rate is configured to $f_{hw_clk}/2$.*

A **Baud Rate Error** (slave mode) is detected when the incoming clock signal deviates from the programmed baud rate by more than 100%, i.e., it is either more than double or less than half the expected baud rate. This condition sets the error flag CON.BE and, when enabled via CON.BEN, sets the EIR. Using this error detection capability requires that the slave's baud-rate generator be programmed to the same baud rate as the master device. This feature detects false, additional or missing pulses on the clock line (within a certain frame).

*Note: If this error condition occurs and bit CON.REN = 1, an automatic reset of the SSC will be performed. This is done to re-initialize the SSC if too few or too many clock pulses have been detected.*

*Note: This error can occur after any transfer if the communication is stopped. This is the case due to the fact that the SSC module supports back-to-back transfers for multiple transfers. In order to handle this, the baud rate detector expects immediately after a finished transfer, the next clock cycle for a new transfer.*

A **Transmit Error** (slave mode) is detected when a transfer was initiated by the master (SS_CLK gets active), but the transmit buffer TB of the slave had not been updated since the last transfer. This condition sets the error flag CON.TE and the EIR, when enabled via CON.TEN. If a transfer starts without the transmit buffer having been updated, the slave will shift out the 'old' contents of the shift register, which normally is the data received during the last transfer. This may lead to corruption of the data on the transmit/receive line in half-duplex mode (open drain configuration) if this slave is not selected for transmission. This mode requires that slaves not selected for transmission only shift out ones; that is, their transmit buffers must be loaded with '$FFFF_H$' prior to any transfer.

*Note: A slave with push/pull output drivers not selected for transmission, will normally have its output drivers switched off. However, in order to avoid possible conflicts or misinterpretations, it is recommended to always load the slave's transmit buffer prior to any transfer.*

The cause of an error interrupt request (receive, phase, baud rate or transmit error) can be identified by the error status flags in control register CON.

*Note: The error status flags CON.TE, CON.RE, CON.PE, and CON.BE are not reset automatically upon entry into the error interrupt service routine, but must be cleared by software.*

### 12.3.2 Interrupts

There are 3 interrupt sources in SSC that will trigger an interrupt: TIR, RIR and EIR.

An overview of the various interrupts in SSC is provided in **Table 12-7**.

**Table 12-7 SSC Interrupt Sources**

| Interrupt | Signal | Description |
|---|---|---|
| Transmission starts | TIR | Indicates that the transmit buffer can be reloaded with new data. |
| Transmission ends | RIR | The configured number of bits have been transmitted and shifted to the receive buffer. |
| Receive Error | EIR | This interrupt occurs if a new data frame is completely received and the last data in the receive buffer was not read. |
| Phase Error | EIR | This interrupt is generated if the incoming data changes between one cycle before and two cycles after the latching edge of the shift clock signal SCLK. |
| Baud Rate Error (Slave mode only) | EIR | This interrupt is generated when the incoming clock signal deviates from the programmed baud rate by more than 100%. |
| Transmit Error (Slave mode only) | EIR | This interrupt is generated when TB was not updated since the last transfer if a transfer is initiated by a master. |

SSC module uses interrupt structure 2 which is described in **Chapter 5.1.2** for the 3 interrupt sources. 3 bits define in register MODIEN are used for the enabling and disabling of these interrupt event.

**MODIEN**
**Peripheral Interrupt Enable Register** **Reset Value: $07_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **0** | | | **CM5EN** | **CM4EN** | **RIREN** | **TIREN** | **EIREN** |
| r | | | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **EIREN** | 0 | rw | **Error Interrupt Enable Bit for SSC**<br>0 Error interrupt is disabled.<br>1 Error interrupt is enabled. |
| **TIREN** | 1 | rw | **Transmit Interrupt Enable Bit for SSC**<br>0 Transmit interrupt is disabled.<br>1 Transmit interrupt is enabled. |
| **RIREN** | 2 | rw | **Receive Interrupt Enable Bit for SSC**<br>0 Receive interrupt is disabled.<br>1 Receive interrupt is enabled. |
| **0** | [7:5] | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

### 12.3.3 Low Power Mode

If the SSC functionality is not required at all, it can be completely disabled by gating off its clock input for maximal power reduction. This is done by setting bit SSC_DIS in register PMCON1 as described below. Refer to **Chapter 8.1.4** for details on peripheral clock management.

**PMCON1**
**Power Mode Control Register 1** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **0** | **CDC_DIS** | **CAN_DIS** | **MDU_DIS** | **T2CCU _DIS** | **CCU_DIS** | **SSC_DIS** | **ADC_DIS** |
| r | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **SSC_DIS** | 1 | rw | **SSC Disable Request. Active high.**<br>0 SSC is in normal operation (default).<br>1 Request to disable the SSC. |
| **0** | 7 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

### 12.3.4 Register Map

The addresses of the kernel SFRs are listed in **Table 12-8**.

**Table 12-8 SFR Address List**

| Address | Register |
|---|---|
| $AA_H$ | CONL |
| $AB_H$ | CONH |
| $AC_H$ | TBL |
| $AD_H$ | RBL |
| $AE_H$ | BRL |
| $AF_H$ | BRH |

### 12.3.5 Register Description

All SSC register names described in this section are referenced in other chapters of this document with the module name prefix “SSC_”, e.g., SSC_CONL.

#### 12.3.5.1 Configuration Register

The operating mode of the serial channel SSC is controlled by the control register CON. This register contains control bits for mode and error check selection, and status flags for error identification. Depending on bit EN, either control functions or status flags and master/slave control are enabled.

**CON.EN = 0: Programming Mode**

**CONL**
**Control Register Low** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **LB** | **PO** | **PH** | **HB** | **BM** | | | |
| rw | rw | rw | rw | rw | | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **BM** | [3:0] | rw | **Data Width Selection**<br>0000 Reserved. Do not use this combination.<br>0001 -<br>0111 Transfer Data Width is 2...8 bits (<BM>+1)<br>*Note: BM[3] is fixed to 0.* |
| **HB** | 4 | rw | **Heading Control**<br>0 Transmit/Receive LSB First<br>1 Transmit/Receive MSB First |
| **PH** | 5 | rw | **Clock Phase Control**<br>0 Shift transmit data on the leading clock edge, latch on trailing edge<br>1 Latch receive data on leading clock edge, shift on trailing edge |

| Field | Bits | Type | Description |
|---|---|---|---|
| **PO** | 6 | rw | **Clock Polarity Control**<br>0 Idle clock line is low, leading clock edge is low-to-high transition<br>1 Idle clock line is high, leading clock edge is high-to-low transition |
| **LB** | 7 | rw | **Loop Back Control**<br>0 Normal output<br>1 Receive input is connected with transmit output (half-duplex mode) |

**CONH**
**Control Register High** **Reset Value: $00_H$**

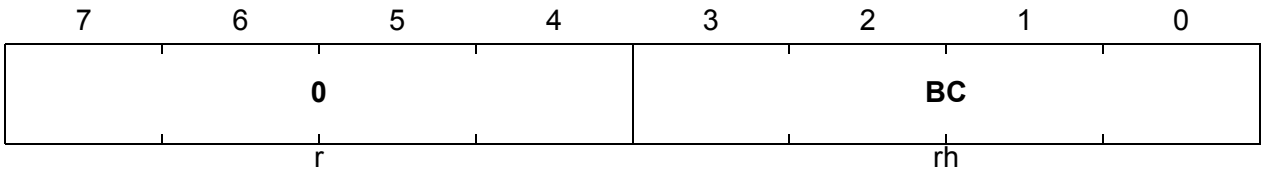
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **EN** | **MS** | **0** | **AREN** | **BEN** | **PEN** | **REN** | **TEN** |
| rw | rw | r | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **TEN** | 0 | rw | **Transmit Error Interrupt Enable**<br>0 Transmit error interrupt is disabled<br>1 Transmit error interrupt is enabled |
| **REN** | 1 | rw | **Receive Error Enable**<br>0 Receive error interrupt is disabled<br>1 Receive error interrupt is enabled |
| **PEN** | 2 | rw | **Phase Error Enable**<br>0 Phase error interrupt is disabled<br>1 Phase error interrupt is enabled |
| **BEN** | 3 | rw | **Baud Rate Error Enable**<br>0 Baud rate error interrupt is disabled<br>1 Baud rate error interrupt is enabled |
| **AREN** | 4 | rw | **Automatic Reset Enable**<br>0 No additional action upon a baud rate error<br>1 The SSC is automatically reset upon a baud rate error. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **MS** | 6 | rw | **Master Select**<br>0 Slave mode. Operate on shift clock received via SCLK.<br>1 Master mode. Generate shift clock and output it via SCLK. |
| **EN** | 7 | rw | **Enable Bit = 0**<br>Transmission and reception disabled. Access to control bits. |
| **0** | 5 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

### CON.EN = 1: Operating Mode

**CONL**
**Control Register Low** **Reset Value: $00_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **BC** | [3:0] | rh | **Bit Count Field**<br>0001 -<br>1111 Shift counter is updated with every shifted bit |
| **0** | [7:4] | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

**CONH**
**Control Register High** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **EN** | **MS** | **0** | **BSY** | **BE** | **PE** | **RE** | **TE** |
| rw | rw | r | rh | rwh | rwh | rwh | rwh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **TE** | 0 | rwh | **Transmit Error Flag**<br>0 No error<br>1 Transfer starts with the slave's transmit buffer not being updated |
| **RE** | 1 | rwh | **Receive Error Flag**<br>0 No error<br>1 Reception completed before the receive buffer was read |
| **PE** | 2 | rwh | **Phase Error Flag**<br>0 No error<br>1 Received data changes around sampling clock edge |

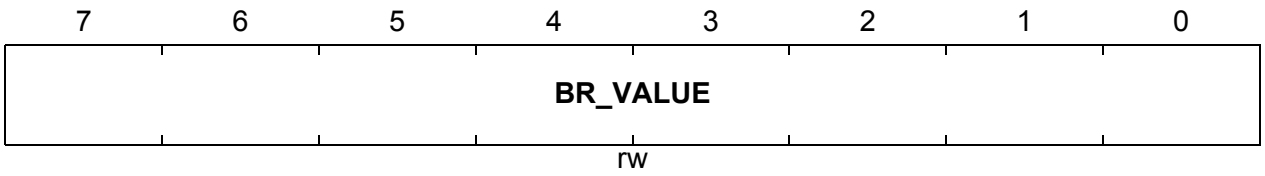
| Field | Bits | Type | Description |
|---|---|---|---|
| **BE** | 3 | rwh | **Baud rate Error Flag**<br>0 No error<br>1 More than factor 2 or 0.5 between slave's actual and expected baud rate |
| **BSY** | 4 | rh | **Busy Flag**<br>Set while a transfer is in progress |
| **MS** | 6 | rw | **Master Select Bit**<br>0 Slave mode. Operate on shift clock received via SCLK.<br>1 Master mode. Generate shift clock and output it via SCLK. |
| **EN** | 7 | rw | **Enable Bit = 1**<br>Transmission and reception enabled. Access to status flags and Master/Slave control. |
| **0** | 5 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

*Note: The target of an access to CON (control bits or flags) is determined by the state of CON.EN prior to the access; that is, writing $C057_H$ to CON in programming mode (CON.EN = 0) will initialize the SSC (CON.EN was 0) and then turn it on (CON.EN = 1). When writing to CON, ensure that reserved locations receive zeros.*

### 12.3.5.2 Baud Rate Timer Reload Register

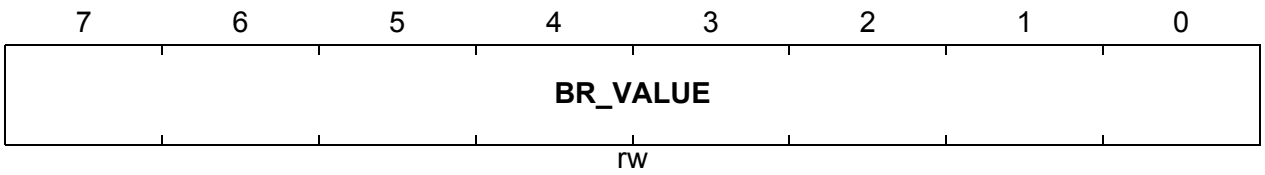
The SSC baud rate timer reload register BR contains the 16-bit reload value for the baud rate timer.

**BRL**
**Baud Rate Timer Reload Register Low** **Reset Value: $00_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **BR_VALUE** | [7:0] | rw | **Baud Rate Timer/Reload Register Value [7:0]** Reading BR returns the 16-bit contents of the baud rate timer. Writing to BR loads the baud rate timer reload register with BR_VALUE. |

**BRH**
**Baud Rate Timer Reload Register High** **Reset Value: $00_H$**

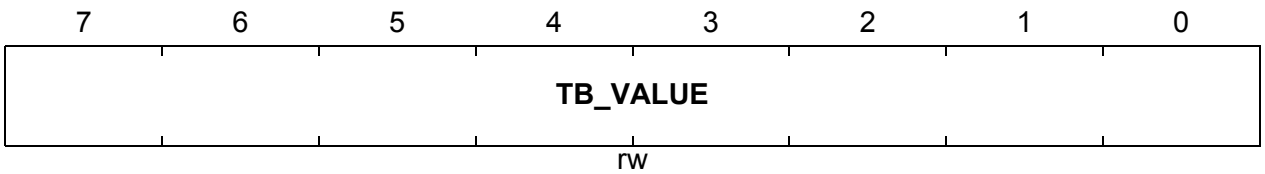



| Field | Bits | Type | Description |
|---|---|---|---|
| **BR_VALUE** | [7:0] | rw | **Baud Rate Timer/Reload Register Value [15:8]** Reading BR returns the 16-bit contents of the baud rate timer. Writing to BR loads the baud rate timer reload register with BR_VALUE. |

### 12.3.5.3 Transmit and Receive Buffer Register

The SSC transmitter buffer register TB contains the transmit data value.

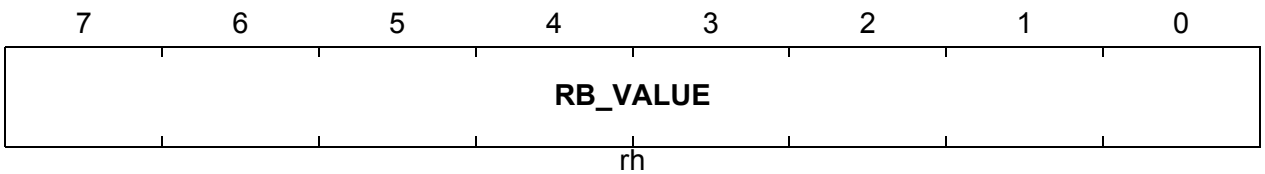
**TBL**
**Transmitter Buffer Register Low** **Reset Value: $00_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **TB_VALUE** | [7:0] | rw | **Transmit Data Register Value**<br>TB_VALUE is the data value to be transmitted. Unselected bits of TB are ignored during transmission. |

The SSC receiver buffer register RB contains the receive data value.

**RBL**
**Receiver Buffer Register Low** **Reset Value: $00_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **RB_VALUE** | [7:0] | rh | **Receive Data Register Value**<br>RB contains the received data value RB_VALUE. Unselected bits of RB will not be valid and should be ignored. |

# 13 Timers

The XC878 provides four 16-bit timers, Timer 0, Timer 1, Timer 2 and Timer 21. They are useful in many timing applications such as measuring the time interval between events, counting events and generating signals at regular intervals. In particular, Timer 1 can be used as the baud-rate generator for the on-chip serial port.

**Timer 0 and Timer 1 Features:**

- Four operational modes :
  - Mode 0: 13-bit timer/counter
  - Mode 1: 16-bit timer/counter
  - Mode 2: 8-bit timer/counter with auto-reload
  - Mode 3: Two 8-bit timers/counters

**Timer 2 and Timer 21 Features:**

- Selectable up/down counting
- 16-bit auto-reload mode
- 1 channel, 16-bit capture mode

*Note: Timer 2 refers to the 16-bit general purpose timer in T2CCU modules(Timer 2 Capture/Compare Unit). For the detailed CCU operation of T2CCU, see* ***Chapter 14****.*

## 13.1 Timer 0 and Timer 1

Timer 0 and Timer 1 can function as both timers or counters. When functioning as a timer, Timer 0 and Timer 1 are incremented every machine cycle, i.e. every 2 input clocks (or 2 PCLKs). When functioning as a counter, Timer 0 and Timer 1 are incremented in response to a 1-to-0 transition (falling edge) at their respective external input pins, T0 or T1.

### 13.1.1 Basic Timer Operations

The operations of the two timers are controlled using the Special Function Registers (SFRs) TCON and TMOD. To enable a timer, i.e., allow the timer to run, its control bit TCON.TRx is set. To select the timer input to be either from internal system clock or external pin, the input selector bit TMOD is used.

*Note: The "x" (e.g., TCON.TRx) in this chapter denotes either 0 or 1.*

Each timer consists of two 8-bit registers - TLx (low byte) and THx (high byte) which defaults to $00_H$ on reset. Setting or clearing TCON.TRx does not affect the timer registers.

#### Timer Overflow

When a timer overflow occurs, the timer overflow flag, TCON.TFx, is set, and an interrupt may be raised if the interrupt enable control bit, IEN0.ETx, is set. The overflow flag is automatically cleared when the interrupt service routine is entered.

When Timer 0 operates in mode 3, the Timer 1 control bits, TR1, TF1 and ET1 are reserved for TH0, see **Section 13.1.2.4**.

#### External Control

In addition to pure software control, the timers can also be enabled or disabled through external port control. When external port control is used, SFR EXICON0 must first be configured to bypass the edge detection circuitry for EXINTx to allow direct feed-through. When the timer is enabled (TCON.TRx = 1) and TMOD.GATEx is set, the respective timer will only run if the core external interrupt EXINTx = 1. This facilitates pulse width measurements. However, this is not applicable for Timer 1 in mode 3.

If TMOD.GATEx is cleared, the timer reverts to pure software control.

### 13.1.2 Timer Modes

Timers 0 and 1 are fully compatible and can be configured in four different operating modes, as shown in **Table 13-1**. The bit field TxM in register TMOD selects the operating mode to be used for each timer.

In modes 0, 1 and 2, the two timers operate independently, but in mode 3, their functions are specialized.

**Table 13-1 Timer 0 and Timer 1 Modes**

| Mode | Operation |
|---|---|
| **0** | **13-bit timer/counter**<br>The timer is essentially an 8-bit counter with a divide-by-32 prescaler. This mode is included solely for compatibility with Intel 8048 devices. |
| **1** | **16-bit timer/counter**<br>The timer registers, TLx and THx, are concatenated to form a 16-bit timer/counter. |
| **2** | **8-bit timer/counter with auto-reload**<br>The timer register TLx is reloaded with a user-defined 8-bit value in THx upon overflow. |
| **3** | **Timer 0 operates as two 8-bit timers/counters**<br>The timer registers, TL0 and TH0, operate as two separate 8-bit counters. Timer 1 is halted and retains its count even if enabled. |

### 13.1.2.1 Mode 0

Putting either Timer 0 or Timer 1 into mode 0 configures it as an 8-bit timer/counter with a divide-by-32 prescaler. **Figure 13-1** shows the mode 0 operation.

In this mode, the timer register is configured as a 13-bit register. As the count rolls over from all 1s to all 0s, it sets the timer overflow flag TFx. The overflow flag TFx can then be used to request an interrupt. The counted input is enabled for the timer when TRx = 1 and either GATEx = 0 or EXINTx = 1 (setting GATEx = 1 allows the timer to be controlled by external input EXINTx to facilitate pulse width measurements). TRx is a control bit in the register TCON; bit GATEx is in register TMOD..

The 13-bit register consists of all the 8 bits of THx and the lower 5 bits of TLx. The upper 3 bits of TLx are indeterminate and should be ignored. Setting the run flag (TRx) does not clear the registers..

Mode 0 operation is the same for Timer 0 and Timer 1.

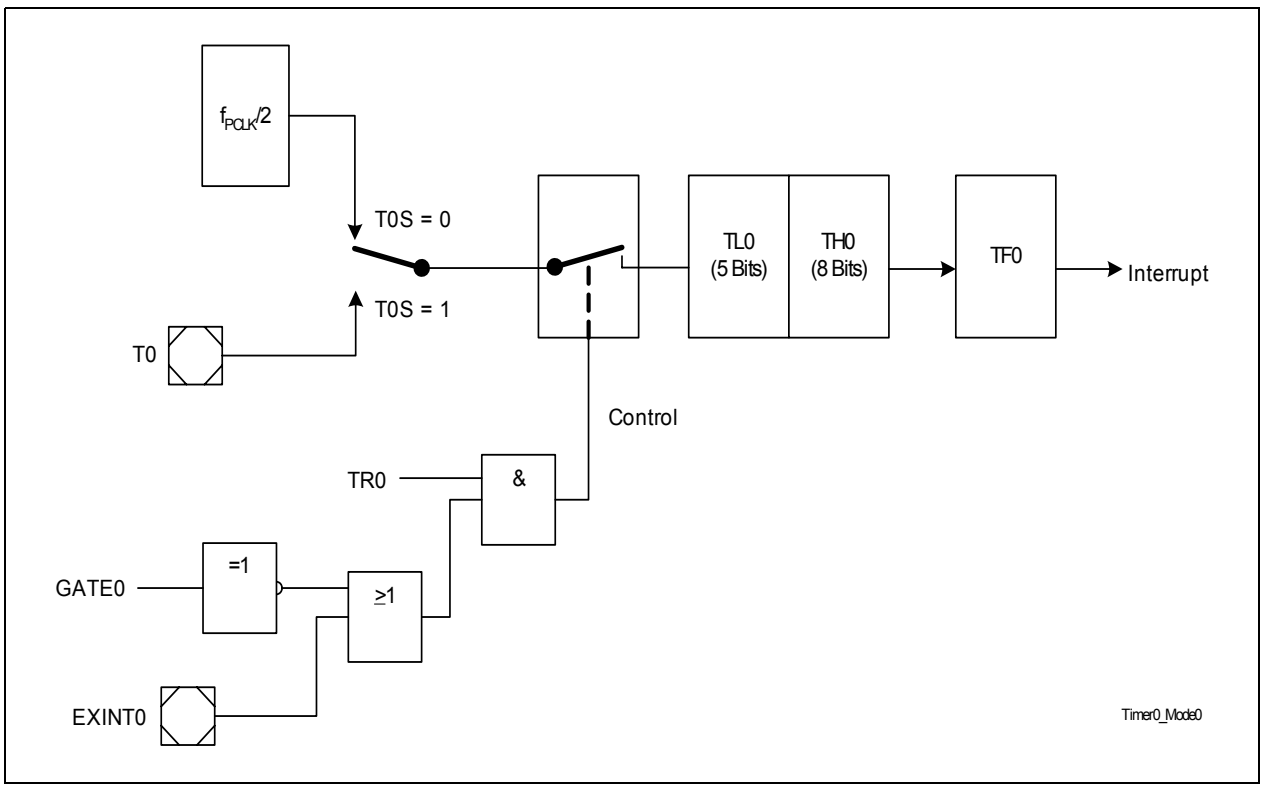



**Figure 13-1 Timer 0, Mode 0: 13-Bit Timer/Counter**

### 13.1.2.2 Mode 1

Mode 1 operation is similar to that of mode 0, except that the timer register runs with all 16 bits. Mode 1 operation for Timer 0 is shown in **Figure 13-2**.

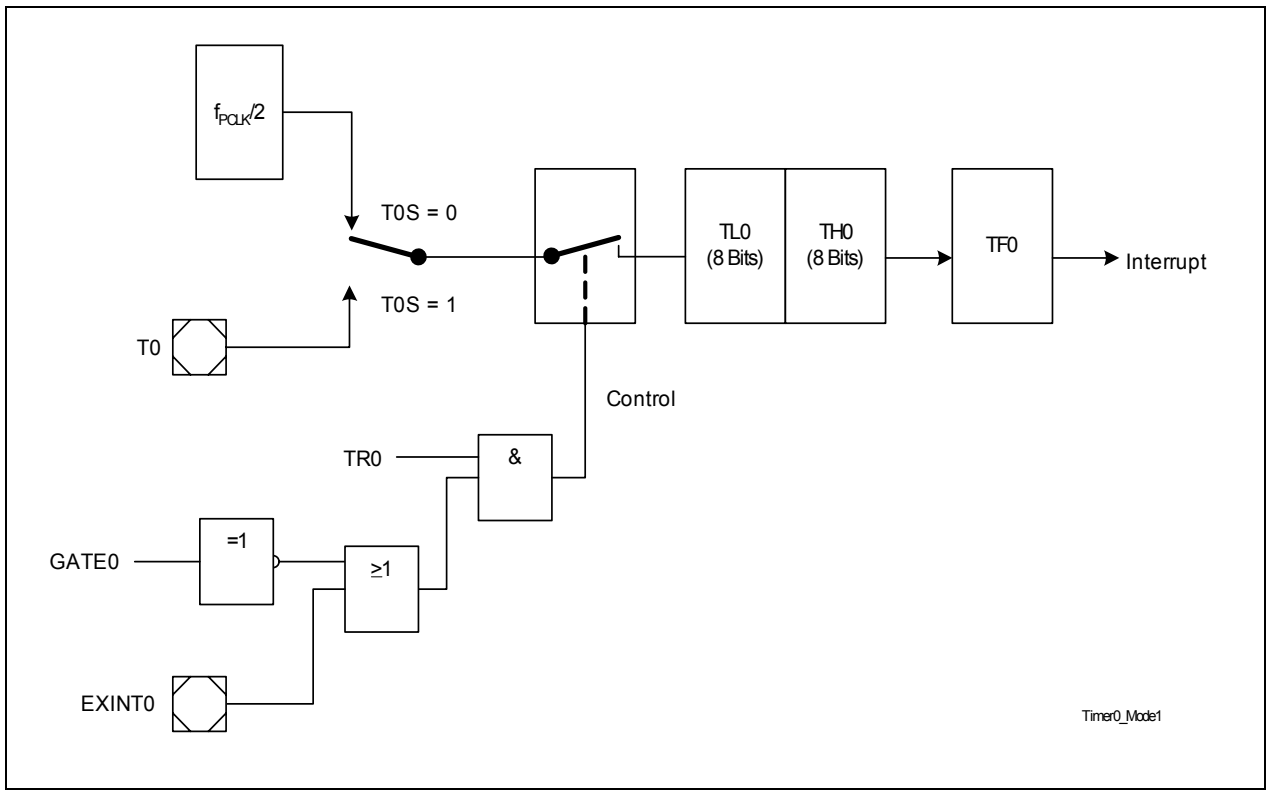



**Figure 13-2 Timer 0, Mode 1: 16-Bit Timer/Counter**

### 13.1.2.3 Mode 2

In Mode 2 operation, the timer is configured as an 8-bit counter (TLx) with automatic reload, as shown in **Figure 13-3** for Timer 0.

An overflow from TLx not only sets TFx, but also reloads TLx with the contents of THx that has been preset by software. The reload leaves THx unchanged.

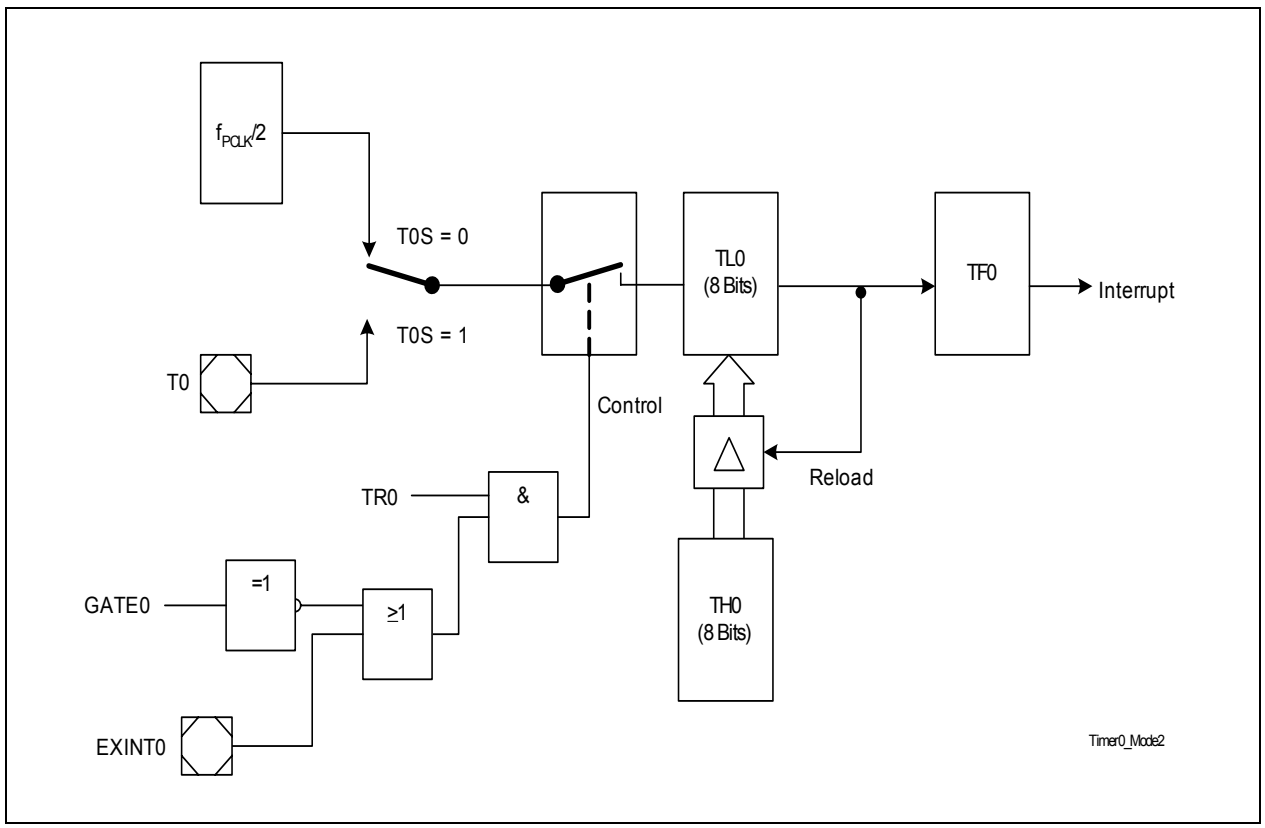



**Figure 13-3 Timer 0, Mode 2: 8-Bit Timer/Counter with Auto-Reload**

### 13.1.2.4 Mode 3

In mode 3, Timer 0 and Timer 1 behave differently. Timer 0 in mode 3 establishes TL0 and TH0 as two separate counters. Timer 1 in mode 3 simply holds its count. The effect is the same as setting TR1 = 0

The logic for mode 3 operation for Timer 0 is shown in **Figure 13-4**. TL0 uses the Timer 0 control bits GATE0, TR0 and TF0, while TH0 is locked into a timer function (counting machine cycles) and takes over the use of TR1 and TF1 from Timer 1. Thus, TH0 now sets TF1 upon overflow and generates an interrupt if ET1 is set.

Mode 3 is provided for applications requiring an extra 8-bit timer. When Timer 0 is in mode 3 and TR1 is set, Timer 1 can be turned on by switching it to any of the other modes and turned off by switching it into mode 3.

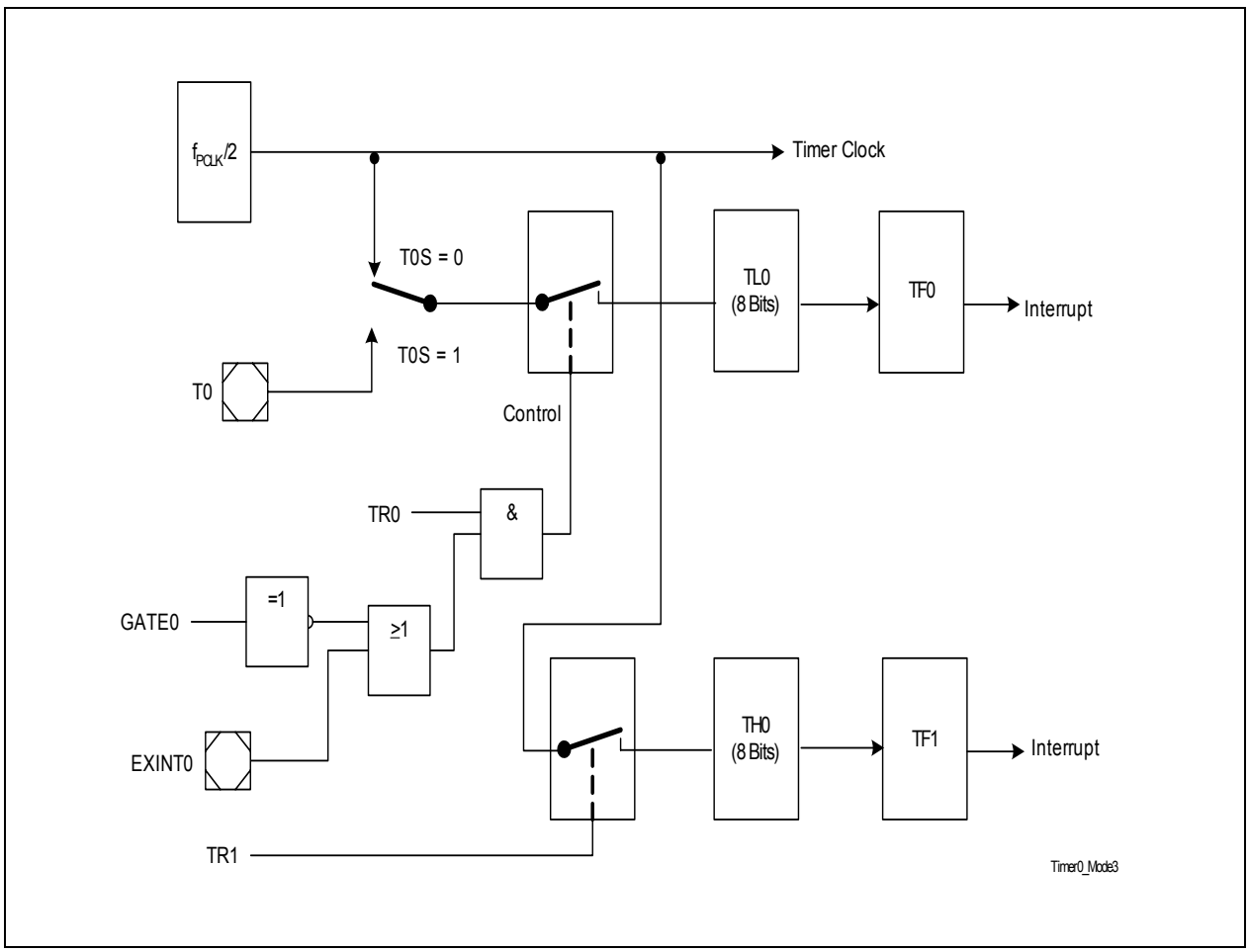
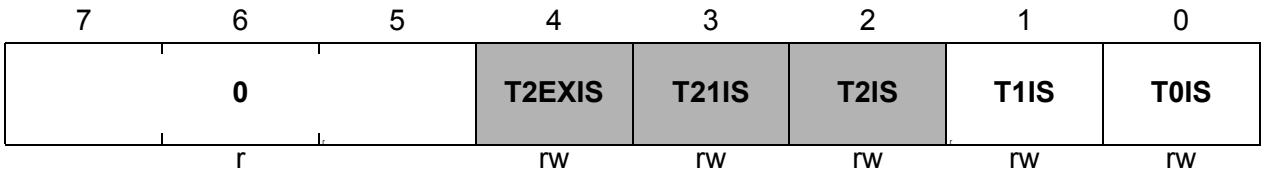



**Figure 13-4 Timer 0, Mode 3: Two 8-Bit Timers/Counters**

### 13.1.3 Port Control

When functioning as an event counter, Timer 0 and 1 count 1-to-0 transitions at their external input pins, T0 and T1, which can be selected from two different sources, T0_0 and T0_1 for Timer 0, and T1_0 and T1_1 for Timer 1. This selection is performed by the SFR bits MODPISEL2.T0IS and MODPISEL2.T1IS.

**MODPISEL2**
**Peripheral Input Select Register 2** **Reset Value: $00_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **T0IS** | 0 | rw | **T0 Input Select**<br>0 Timer 0 Input T0_0 is selected.<br>1 Timer 0 Input T0_1 is selected. |
| **T1IS** | 1 | rw | **T1 Input Selectt**<br>0 Timer 1 Input T1_0 is selected.<br>1 Timer 1 Input T1_1 is selected. |
| **0** | [7:5] | r | **Reservedl**<br>Returns 0 if read; should be written with 0. |

### 13.1.4 Register Map

Seven SFRs control the operations of Timer 0 and Timer 1. They can be accessed from both the standard (non-mapped) and mapped SFR area.

Table 13-2 lists the addresses of these SFRs.

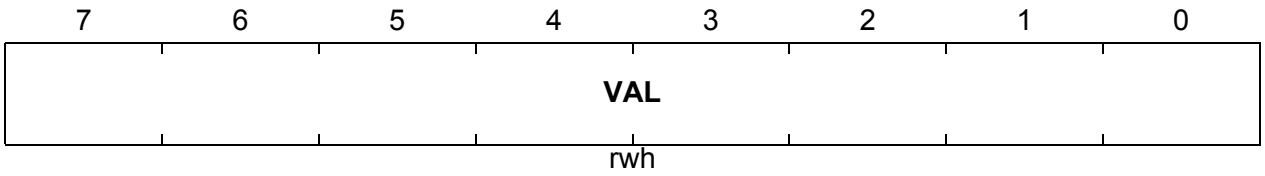
**Table 13-2 Register Map**

| Address | Register |
|---|---|
| $88_H$ | TCON |
| $89_H$ | TMOD |
| $8A_H$ | TL0 |
| $8B_H$ | TL1 |
| $8C_H$ | TH0 |
| $8D_H$ | TH1 |
| $A8_H$ | IEN0 |

### 13.1.5 Register Description

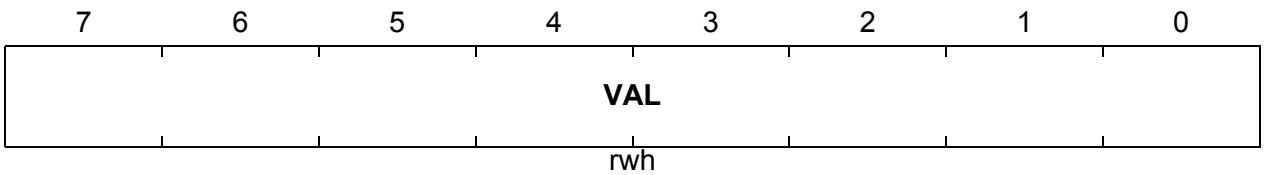
The low bytes(TL0, TL1) and high bytes(TH0, TH1)of both Timer 0 and Timer 1 can be combined to a one-timer configuration depending on the mode used. Register TCON controls the operations of Timer 0 and Timer 1. The operating modes of both timers are selected using register TMOD. Register IEN0 contains bits that enable interrupt operations in Timer 0 and Timer 1.

**TLx (x = 0 - 1)**
**Timer x, Low Byte** **Reset Value: $00_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **TLx.VAL(x = 0, 1)** | [7:0] | rwh | **Timer 0/1 Low Register**<br>00 TLx holds the 5-bit prescaler value.<br>01 TLx holds the lower 8-bit part of the 16-bit timer value.<br>10 TLx holds the 8-bit timer value.<br>11 TL0 holds the 8-bit timer value; TL1 is not used. |

**THx (x = 0 - 1)**
**Timer x, High Byte** **Reset Value: $00_H$**



| Field | Bits | Type | Description |
|---|---|---|---|
| **THx.VAL(x = 0, 1)** | [7:0] | rwh | **Timer 0/1 High Register**<br>00 THx holds the 8-bit timer value.<br>01 THx holds the higher 8-bit part of the 16-bit timer value.<br>10 THx holds the 8-bit reload value.<br>11 TH0 holds the 8-bit timer value; TH1 is not used. |

**TCON**
**Timer 0/1 Control Registers** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **TF1** | **TR1** | **TF0** | **TR0** | **IE1** | **IT1** | **IE0** | **IT0** |
| rwh | rw | rwh | rw | rwh | rw | rwh | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **TR0** | 4 | rw | **Timer 0 Run Control**<br>0 Timer is halted<br>1 Timer runs |
| **TF0** | 5 | rwh | **Timer 0 Overflow Flag**<br>Set by hardware when Timer 0 overflows.<br>Cleared by hardware when the processor calls the interrupt service routine. |
| **TR1** | 6 | rw | **Timer 1 Run Control**<br>0 Timer is halted<br>1 Timer runs<br>Note: Timer 1 Run Control affects TH0 also if Timer 0 operates in Mode 3. |
| **TF1** | 7 | rwh | **Timer 1 Overflow Flag**<br>Set by hardware when Timer 1[1)] overflows.<br>Cleared by hardware when the processor calls the interrupt service routine. |

[1)] TF1 is set by TH0 instead if Timer 0 operates in Mode 3.

**TMOD**
**Timer Mode Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **GATE1** | **T1S** | **T1M** | | **GATE0** | **T0S** | **T0M** | |
| rw | rw | rw | | rw | rw | rw | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **T0M[1:0]** | [1:0] | rw | **Mode select bits**<br>00 13-bit timer (M8048 compatible mode)<br>01 16-bit timer<br>10 8-bit auto-reload timer<br>11 Timer 0 is split into two halves. TL0 is an 8-bit timer controlled by the standard Timer 0 control bits, and TH0 is the other 8-bit timer controlled by the standard Timer 1 control bits. TH1 and TL1 of Timer 1 are held (Timer 1 is stopped). |
| **T1M[1:0]** | [5:4] | rw | **Mode select bits**<br>00 13-bit timer (M8048 compatible mode)<br>01 16-bit timer<br>10 8-bit auto-reload timer<br>11 Timer 0 is split into two halves. TL0 is an 8-bit timer controlled by the standard Timer 0 control bits, and TH0 is the other 8-bit timer controlled by the standard Timer 1 control bits. TH1 and TL1 of Timer 1 are held (Timer 1 is stopped). |
| **T0S** | 2 | rw | **Timer 0 Selector**<br>0 Input is from internal system clock<br>1 Input is from T0 pin |
| **GATE0** | 3 | rw | **Timer 0 Gate Flag**<br>0 Timer 0 will only run if TCON.TR0 = 1 (software control)<br>1 Timer 0 will only run if EXINT0 pin = 1 (hardware control) and TCON.TR0 is set |

| Field | Bits | Type | Description |
|---|---|---|---|
| **T1S** | 6 | rw | **Timer 1 Selector**<br>0 Input is from internal system clock<br>1 Input is from T1 pin |
| **GATE1** | 7 | rw | **Timer Gate Flag**<br>0 Timer 1 will only run if TCON.TR1 = 1 (software control)<br>1 Timer 1 will only run if EXINT1 pin = 1 (hardware control) and TCON.TR1 is set |

**IEN0**
**Interrupt Enable Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **EA** | **0** | **ET2** | **ES** | **ET1** | **EX1** | **ET0** | **EX0** |
| rw | r | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **ET0** | 1 | rw | **Timer 0 Overflow Interrupt Enable**<br>0 Timer 0 interrupt is disabled<br>1 Timer 0 interrupt is enabled |
| **ET1** | 3 | rw | **Timer 1 Overflow Interrupt Enable**<br>0 Timer 1 interrupt is disabled<br>1 Timer 1 interrupt is enabled<br>Note: When Timer 0 operates in Mode 3, this interrupt indicates an overflow in the Timer 0 register, TH0. |

## 13.2 Timer 2 and Timer 21

Timer 2(refers to the Timer 2 in T2CCU module) and Timer 21 are 16-bit general purpose timers that are functionally identical. Timer 2 is running at T2CCU clock frequency, $f_{T2CCU}$, which could be PCLK or 2 times PCLK frequency. As for Timer 21, it is running at PCLK frequency. Both timers have two modes of operation, a 16-bit auto-reload mode and a 16-bit one channel capture mode and can function as a timer or counter in each of its modes. As a timer, the timers count with an input clock of $f_{T2CCU}/12$ (if prescaler is disabled) for Timer 2 and $f_{PCLK}/12$ for Timer 21. As a counter, they count 1-to-0 transitions on pin T2 for Timer 2 or on pin T21 for Timer 21. In the counter mode, the maximum resolution for the count is $f_{T2CCU}/24$ (if prescaler is disabled) for Timer 2 and $f_{PCLK}/24$ for Timer 21.

*Note: Subsequent sections describe the functionalities of Timer 2, which is valid also for Timer 21 unless otherwise stated. As for the description of the CCU functions in T2CCU module, please refer to* ***Chapter 14****.*

### 13.2.1 Basic Timer Operations

Timer 2 can be started by using TR2 bit by hardware or software. Timer 2 can be started by setting TR2 bit by software. If bit T2RHEN is set, Timer 2 can be started by hardware. Bit T2REGS defines the event on pin T2EX, falling edge or rising edge, that can set the run bit TR2 by hardware. Timer 2 can only be stopped by resetting TR2 bit by software.

### 13.2.2 Auto-Reload Mode

The auto-reload mode is selected when the bit CP/$\overline{RL2}$ in register T2CON is zero. In this mode, Timer 2 counts to an overflow value and then reloads its register contents with a 16-bit start value for a fresh counting sequence. The overflow condition is indicated by setting bit TF2 in the T2CON register. At the same time, an interrupt request to the core will be generated (if interrupt is enabled). The overflow flag TF2 must be cleared by software.

The auto-reload mode is further classified into two categories depending upon the DCEN control bit in register T2MOD.

#### 13.2.2.1 Up/Down Count Disabled

If DCEN = 0, the up-down count selection is disabled. The timer, therefore, functions as a pure up counting timer only. The operational block diagram is shown in **Figure 13-5**.

If the T2CON register bit EXEN2 = 0, the timer starts to count up to a maximum of $FFFF_H$ once the timer is started by setting the bit TR2 in register T2CON to 1. Upon overflow, bit TF2 is set and the timer register is reloaded with the 16-bit reload value of the RC2 register. This reload value is chosen by software, prior to the occurrence of an overflow condition. A fresh count sequence is started and the timer counts up from this reload value as in the previous count sequence.

If EXEN2 = 1, the timer counts up to a maximum of $FFFF_H$ once TR2 is set. A 16-bit reload of the timer registers from register RC2 is triggered either by an overflow condition or by a negative/positive edge (chosen by the bit EDGESEL in register T2MOD) at input pin T2EX. If an overflow caused the reload, the overflow flag TF2 is set. If a negative/positive transition at pin T2EX caused the reload, bit EXF2 in register T2CON is set. In either case, an interrupt is generated to the core if the related interrupt enable bit EXF2EN/TF2EN in register T2CON1 is enabled and the timer proceeds to its next count sequence. The EXF2 flag, similar to the TF2, must be cleared by software.

If bit T2RHEN is set, Timer 2 is started by first falling edge/rising edge at pin T2EX, which is defined by bit T2REGS. If bit EXEN2 is set, bit EXF2 is also set at the same point when Timer 2 is started with the same falling edge/rising edge at pin T2EX, which is defined by bit EDGESEL. The reload will happen with the following negative/positive transitions at pin T2EX, which is defined by bit EDGESEL.

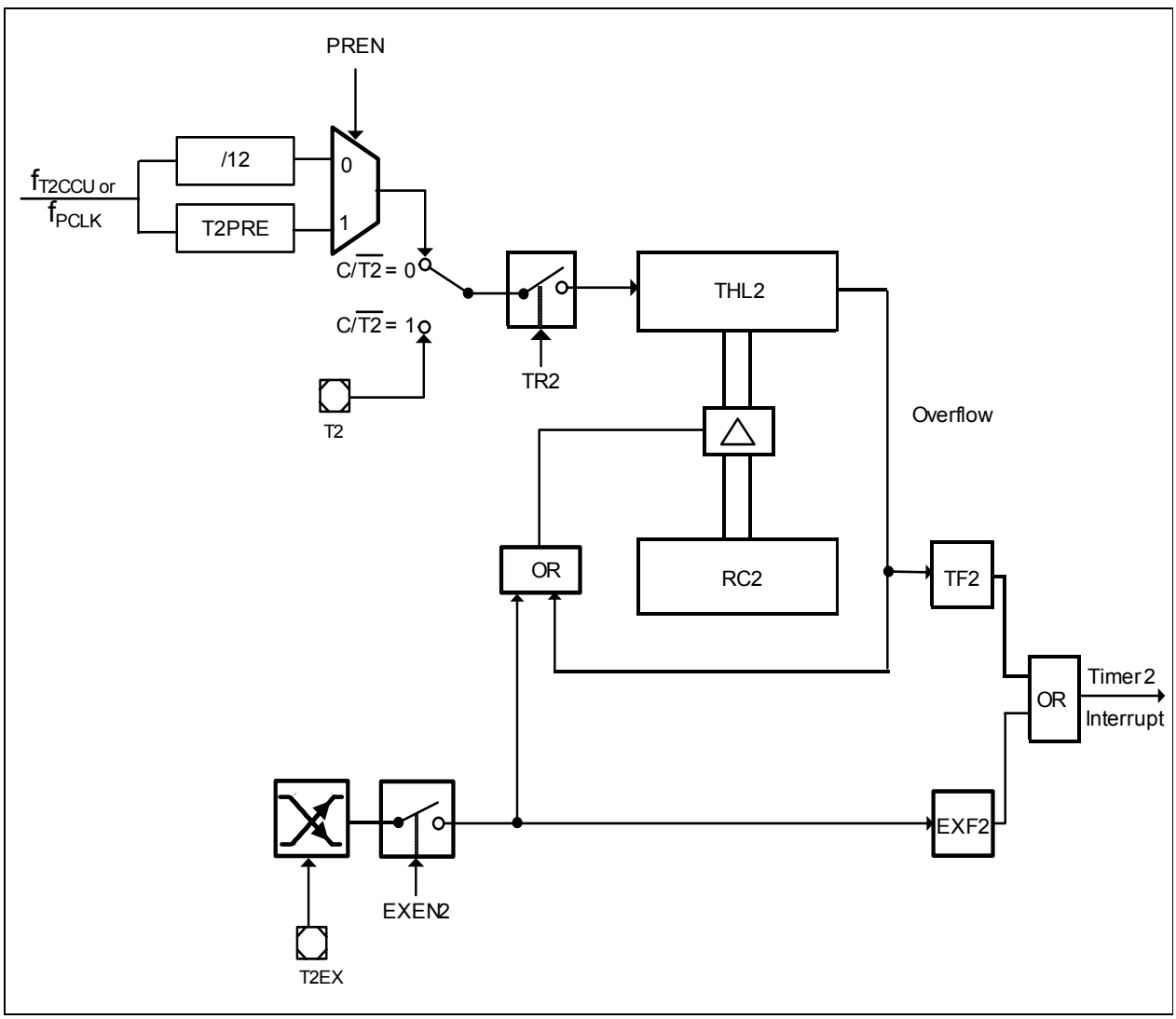



**Figure 13-5 Auto-Reload Mode (DCEN = 0)**

### 13.2.2.2 Up/Down Count Enabled

If DCEN = 1, the up-down count selection is enabled. The direction of count is determined by the level at input pin T2EX. The operational block diagram is shown in **Figure 13-6**.

A logic 1 at pin T2EX sets the Timer 2 to up counting mode. The timer, therefore, counts up to a maximum of $FFFF_H$. Upon overflow, bit TF2 is set and the timer register is reloaded with a 16-bit reload value of the RC2 register. A fresh count sequence is started and the timer counts up from this reload value as in the previous count sequence. This reload value is chosen by software, prior to the occurrence of an overflow condition.

A logic 0 at pin T2EX sets the Timer 2 to down counting mode. The timer counts down and underflows when the THL2 value reaches the value stored at register RC2. The underflow condition sets the TF2 flag and causes $FFFF_H$ to be reloaded into the THL2 register. A fresh down counting sequence is started and the timer counts down as in the previous counting sequence.

If bit T2RHEN is set, Timer 2 can only be started either by rising edge (T2REGS = 1) at pin T2EX and then proceed with the up counting, or be started by falling edge (T2REGS = 0) at pin T2EX and then proceed with the down counting.

In this mode, bit EXF2 toggles whenever an overflow or an underflow condition is detected. This flag, however, does not generate an interrupt request.

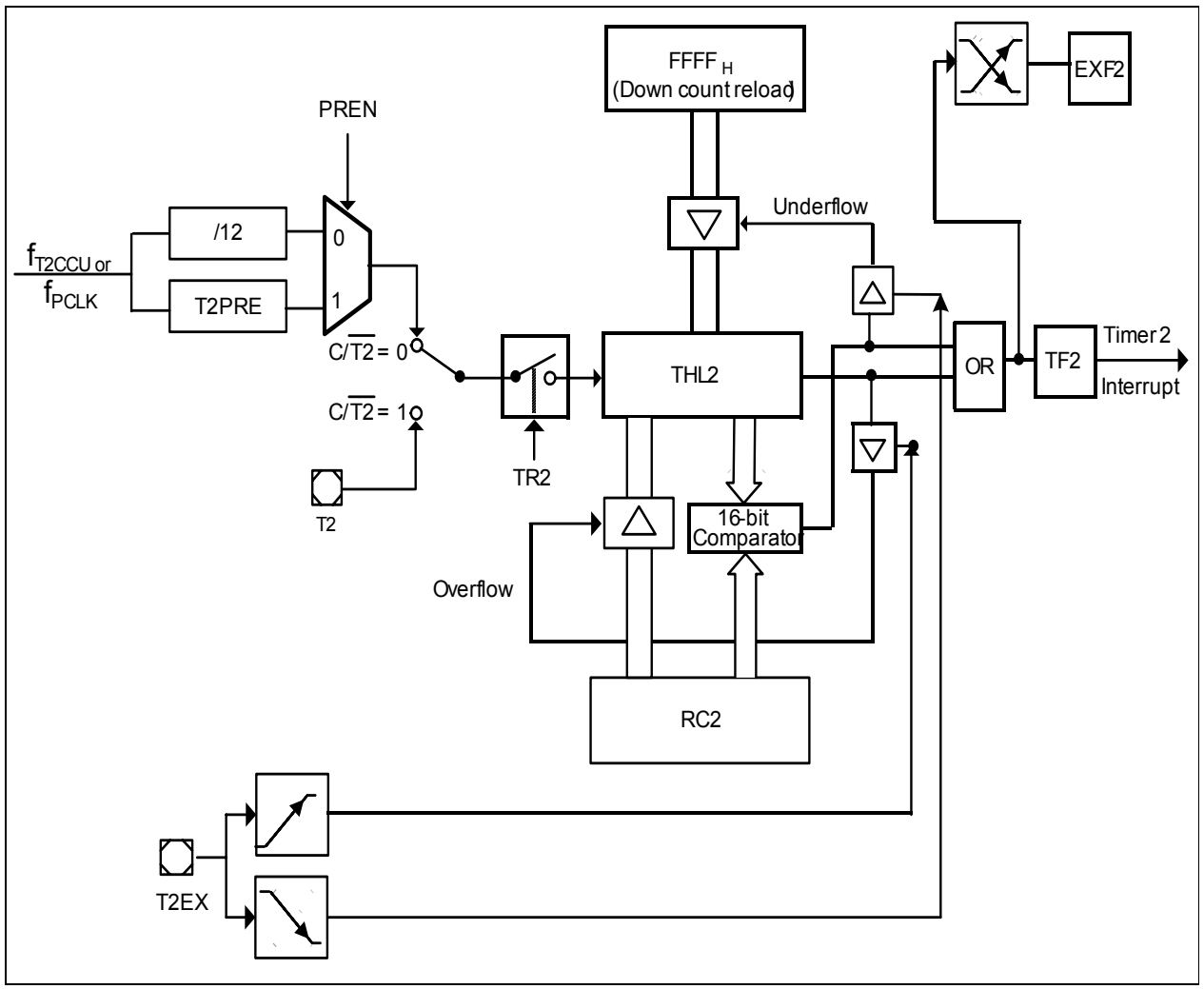



**Figure 13-6 Auto-Reload Mode (DCEN = 1)**

### 13.2.3 Capture Mode

In order to enter the 16-bit capture mode, bits CP/$\overline{RL2}$ and EXEN2 in register T2CON must be set. In this mode, the down count function must remain disabled. The timer functions as a 16-bit timer and always counts up to $FFFF_H$, after which, an overflow condition occurs. Upon overflow, bit TF2 is set and the timer reloads its registers with $0000_H$. The setting of TF2 generates an interrupt request to the core.

Additionally, with a falling/rising edge (chosen by T2MOD.EDGESEL) on pin T2EX, the contents of the timer register (THL2) are captured into the RC2 register. The external input is sampled in every T2CCU clock cycle. When a sampled input shows a low (high) level in one T2CCU clock cycle and a high (low) in the next T2CCU clock cycle, a transition is recognized. If the capture signal is detected while the counter is being incremented, the counter is first incremented before the capture operation is performed. This ensures that the latest value of the timer register is always captured.

If bit T2RHEN is set, Timer 2 is started by first falling edge/rising edge at pin T2EX, which is defined by bit T2REGS. If bit EXEN2 is set, bit EXF2 is also set at the same point when Timer 2 is started with the same falling edge/rising edge at pin T2EX, which is defined by bit EDGESEL. The capture will happen with the following negative/positive transitions at pin T2EX, which is defined by bit EDGESEL.

When the capture operation is completed, bit EXF2 is set and can be used to generate an interrupt request. **Figure 13-7** describes the capture function of Timer 2.

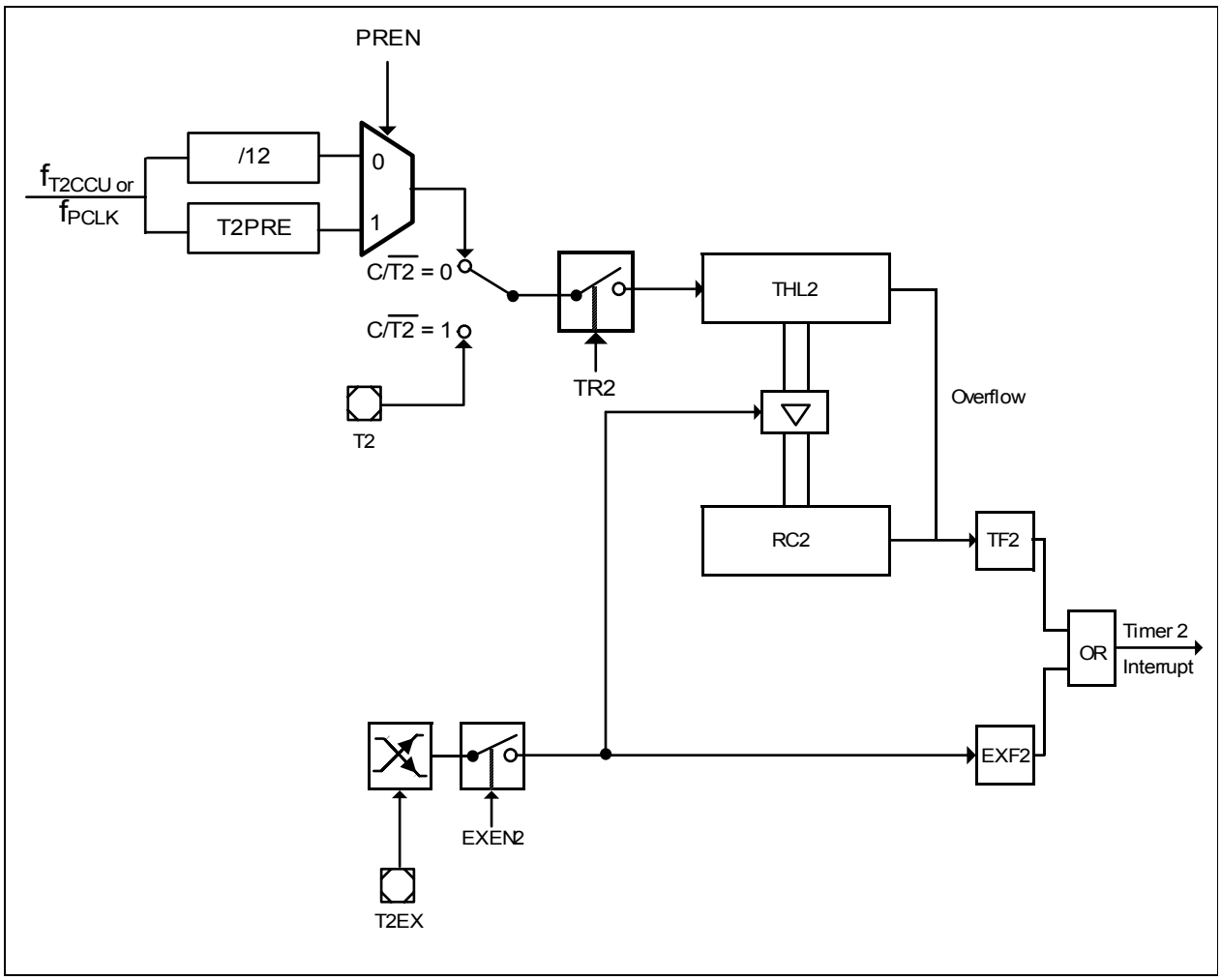



**Figure 13-7 Capture Mode**

## 13.2.4 Count Clock

The count clock for the auto-reload mode is chosen by the bit C/$\overline{T2}$ in register T2CON. If C/$\overline{T2}$ = 0, a count clock of $f_{T2CCU}$/12 (if prescaler is disabled) is used for the count operation.

If C/$\overline{T2}$ = 1, Timer 2 behaves as a counter that counts 1-to-0 transitions of input pin T2. The counter samples pin T2 over 2 T2CCU clock cycles. If a 1 was detected during the first clock and a 0 was detected in the following clock, then the counter increments by one. Therefore, the input levels should be stable for at least 1 clock.

If bit T2RHEN is set, Timer 2 can be started by the falling edge/rising edge on pin T2EX, which is defined by bit T2REGS.

*Note: The C501 compatible feature requires a count resolution of at least 24 clocks.*

### 13.2.5 External Interrupt Function

While the timer/counter function is disabled (TR2 = 0), it is still possible to generate a Timer 2 interrupt to the core via an external event at T2EX, as long as Timer 2 remains enabled (PMCON1.T2_DIS = 0). To achieve this, bit EXEN2 in register T2CON must be set. As a result, any transition on T2EX will cause either a dummy reload or a dummy capture, depending on the CP/ RL2 bit selection.

By disabling the timer/counter function, T2EX can be alternatively used to provide an edge-triggered (rising or falling) external interrupt function, with bit EXF2 serving as the external interrupt flag.

### 13.2.6 Port Control

When functioning as an event counter, Timer 2 and Timer 21 count 1-to-0 transitions at their external input pin, T2 and T21, which can be selected from two different sources, T2_0 and T2_1 for Timer 2, and T21_0 and T21_1 for Timer 21. This selection is performed by the SFR bits MODPISEL2.T2IS and MODPISEL2.T21IS. In addition, there are two sources of T2EX pin for Timer 2 and T21EX pin for Timer 21. They can be selected by MODPISEL2.T2EXIS and MODPISEL1.T21EXIS

**MODPISEL2**
**Peripheral Input Select Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **0** | | | **T2EXIS** | **T21IS** | **T2IS** | **T1IS** | **T0IS** |
| r | | | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **T2IS** | 2 | rw | **T2 Input Select**<br>0 Timer 2 Input T2_0 is selected.<br>1 Timer 2 Input T2_1 is selected. |
| **T21IS** | 3 | rw | **T21 Input Select**<br>0 Timer 21 Input T21_0 is selected.<br>1 Timer 21 Input T21_1 is selected. |
| **T2EXIS** | 4 | rw | **T2 External Input Select**<br>0 Timer 2 Input T2EX_0 is selected.<br>1 Timer 2 Input T2EX_1 is selected. |
| **0** | [7:5] | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

**MODPISEL1**
**Peripheral Input Select Register 1** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| EXINT6IS | | | UR1RIS | | T21EXIS | 0 | |
| rw | | | rw | | rw | r | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **T21EXIS** | 2 | rw | **T21 External Input Select**<br>0 Timer 21 Input T21EX_0 is selected.<br>1 Timer 21 Input T21EX_1 is selected. |
| **0** | [1:0] | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

## 13.2.7 Low Power Mode

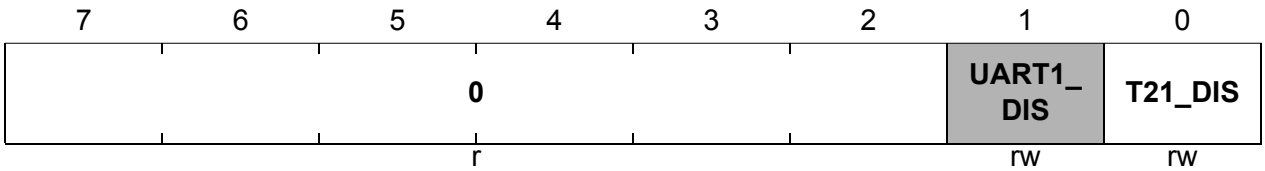
If the Timer 2 together with the Capture/Compare and Timer 21 functionalities are not required at all, they can be completely disabled by gating off their clock inputs for maximal power reduction. This is done by setting bits T2CCU_DIS in register PMCON1 and T21_DIS in register PMCON2 as described below. Refer to **Table 8.1.4** for details on peripheral clock management.

**PMCON1**
**Power Mode Control Register 1** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | CDC_DIS | CAN_DIS | MDU_DIS | T2CCU _DIS | CCU_DIS | SSC_DIS | ADC_DIS |
| r | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **T2CCU_DIS** | 3 | rw | **Timer 2 Capture/Compare Unit Disable Request. Active high.**<br>0 T2CCU is in normal operation (default).<br>1 Request to disable the T2CCU. |
| **0** | 7 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

**PMCON2**
**Power Mode Control Register 2** **Reset Value: $00_H$**



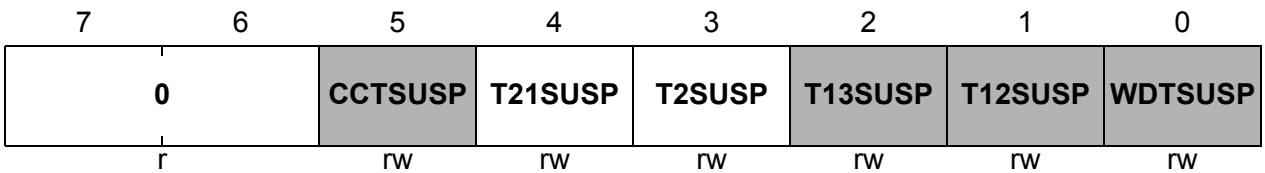

| Field | Bits | Type | Description |
|---|---|---|---|
| **T21_DIS** | 0 | rw | **Timer 21 Disable Request. Active high.**<br>0 Timer 21 is in normal operation (default).<br>1 Request to disable the Timer 21. |
| **0** | [7:2] | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

## 13.2.8 Module Suspend Control

Timer 2 can be configured to stop their counting when the OCDS enters monitor mode (see **Chapter 18.3**) by setting their respective module suspend bits, T2SUSP, in SFR MODSUSP.

While the timers are suspended, their module settings and configuration are not to be changed.

**MODSUSP**
**Module Suspend Control Register** **Reset Value: $01_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **T2SUSP** | 3 | rw | **Timer 2 Debug Suspend Bit**<br>0 Timer 2 will not be suspended.<br>1 Timer 2 will be suspended. |
| **T21SUSP** | 4 | rw | **Timer 21 Debug Suspend Bit**<br>0 Timer 21 will not be suspended.<br>1 Timer 21 will be suspended. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | [7:6] | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

### 13.2.9 Register Map

Timer 2 and Timer 21 contain an identical set of SFRs.

All Timer 2 register names described in the following sections are referenced in other chapters of this document with the module name prefix “T2_”, e.g., T2_T2CON, while those of Timer 21 are referenced with “T21_”, e.g., T21_T2CON.

The Timer 2 SFRs are located in Page 0 of the standard (non-mapped) SFR area. The corresponding set of SFRs for Timer 21 are assigned the same address as the Timer 2 SFRs, except that they are located instead in the mapped area. **Table 13-3** lists these addresses.

**Table 13-3 SFR Address List**

| Address | Register |
|---|---|
| $C0_H$ | T2CON |
| $C1_H$ | T2MOD |
| $C2_H$ | RC2L |
| $C3_H$ | RC2H |
| $C4_H$ | T2L |
| $C5_H$ | T2H |
| $C6_H$ | T2CON1 |

### 13.2.10 Register Description

Register T2MOD is used to configure Timer 2 for the various modes of operation.

**T2MOD**
**Timer 2 Mode Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **T2REGS** | **T2RHEN** | **EDGESEL** | **PREN** | **T2PRE** | | | **DCEN** |
| rw | rw | rw | rw | rw | | | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **DCEN** | 0 | rw | **Up/Down Counter Enable**<br>0 Up/Down Counter function is disabled.<br>1 Up/Down Counter function is enabled and controlled by pin T2EX (Up = 1, Down = 0). |
| **T2PRE** | [3:1] | rw | **Timer 2 Prescaler Bit**<br>Selects the input clock for Timer 2 which is derived from the T2CCU clock.<br>000 $f_{T2} = f_{T2CCU}$<br>001 $f_{T2} = f_{T2CCU}/2$<br>010 $f_{T2} = f_{T2CCU}/4$<br>011 $f_{T2} = f_{T2CCU}/8$<br>100 $f_{T2} = f_{T2CCU}/16$<br>101 $f_{T2} = f_{T2CCU}/32$<br>110 $f_{T2} = f_{T2CCU}/64$<br>111 $f_{T2} = f_{T2CCU}/128$<br>*Note: T2CCU clock could be running at PCLK or 2 times PCLK frequency. As for T21, it is derived based on PCLK clock only.* |
| **PREN** | 4 | rw | **Prescaler Enable**<br>0 Prescaler is disabled and the divider 12 takes effect.<br>1 Prescaler is enabled (see T2PRE bit) and the divider 12 is bypassed. |
| **EDGESEL** | 5 | rw | **Edge Select in Capture Mode/Reload Mode**<br>0 The falling edge at pin T2EX is selected.<br>1 The rising edge at pin T2EX is selected. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **T2RHEN** | 6 | rw | **Timer 2 External Start Enable**<br>0 Timer 2 External Start is disabled.<br>1 Timer 2 External Start is enabled. |
| **T2REGS** | 7 | rw | **Edge Select for Timer 2 External Start**<br>0 The falling edge at Pin T2EX is selected.<br>1 The rising edge at Pin T2EX is selected. |

Register T2CON controls the operating modes of Timer 2. In addition, it contains the status flags for interrupt generation.

**T2CON**
**Timer 2 Control Register** **Reset Value: $00_H$**

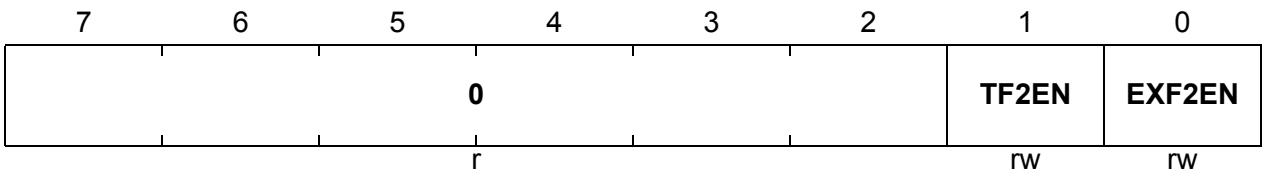
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **TF2** | **EXF2** | **0** | | **EXEN2** | **TR2** | **C/$\overline{T2}$** | **CP/$\overline{RL2}$** |
| rwh | rwh | r | | rw | rwh | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **CP/$\overline{RL2}$** | 0 | rw | **Capture/Reload Select**<br>0 Reload upon overflow or upon negative/positive transition at pin T2EX (when EXEN2 = 1).<br>1 Capture Timer 2 data register contents on the negative/positive transition at pin T2EX, provided EXEN2 = 1.<br>The negative or positive transition at pin T2EX is selected by bit EDGESEL. |
| **C/$\overline{T2}$** | 1 | rw | **Timer or Counter Select**<br>0 Timer function selected<br>1 Count upon negative edge at pin T2 |
| **TR2** | 2 | rwh | **Timer 2 Start/Stop Control**<br>0 Stop Timer 2<br>1 Start Timer 2 |
| **EXEN2** | 3 | rw | **Timer 2 External Enable Control**<br>0 External events are disabled.<br>1 External events are enabled in capture/reload mode. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **EXF2** | 6 | rwh | **Timer 2 External Flag**<br>In capture/reload mode, this bit is set by hardware when a negative/positive transition occurs at pin T2EX, if bit EXEN2 = 1. This bit must be cleared by software.<br>*Note: When bit DCEN = 1 in auto-reload mode, no interrupt request to the core is generated.* |
| **TF2** | 7 | rwh | **Timer 2 Overflow/Underflow Flag**<br>Set by a Timer 2 overflow/underflow. Must be cleared by software. |
| **0** | [5:4] | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

Register T2CON1is used to enable the external interrupt and the overflow interrupt.

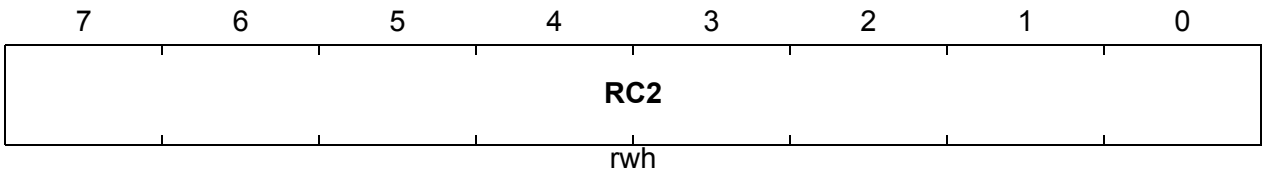
**T2CON1**
**Timer 2 Control Register 1** **Reset Value: $03_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **EXF2EN** | 0 | rw | **External Interrupt Enable**<br>0 External interrupt is disabled.<br>1 External interrupt is enabled. |
| **TF2EN** | 1 | rw | **Overflow/Underflow Interrupt Enable**<br>0 Overflow/underflow interrupt is disabled<br>1 Overflow/underflow interrupt is enabled |
| **0** | [7:2] | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

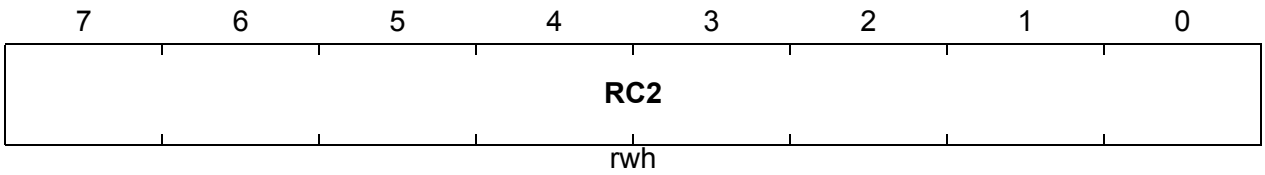
Register RC2 is used for a 16-bit reload of the timer count upon overflow or a capture of current timer count depending on the mode selected.

**RC2L**
**Timer 2 Reload/Capture Register Low** **Reset Value: $00_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **RC2** | [7:0] | rwh | **Reload/Capture Value [7:0]**<br>If CP/RL2 = 0, these contents are loaded into the timer register upon an overflow condition.<br>If CP/RL2 = 1, this register is loaded with the current timer count upon a negative/positive transition at pin T2EX when EXEN2 = 1. |

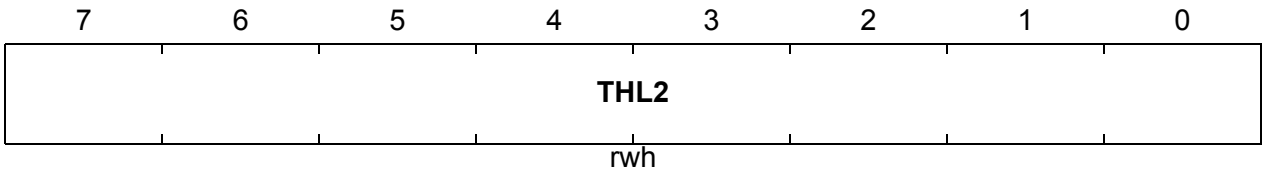
**RC2H**
**Timer 2 Reload/Capture Register High** **Reset Value: $00_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **RC2** | [7:0] | rwh | **Reload/Capture Value [15:8]**<br>If CP/RL2 = 0, these contents are loaded into the timer register upon an overflow condition.<br>If CP/RL2 = 1, this register is loaded with the current timer count upon a negative/positive transition at pin T2EX when EXEN2 = 1. |

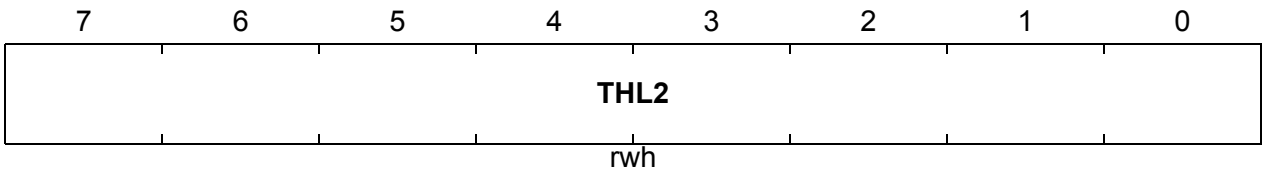
Register T2 holds the current 16-bit value of the Timer 2 count.

**T2L**
**Timer 2 Register Low** **Reset Value: $00_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **THL2** | [7:0] | rwh | **Timer 2 Value [7:0]**<br>These bits indicate the current timer value. |

**T2H**
**Timer 2 Register High** **Reset Value: $00_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **THL2** | [7:0] | rwh | **Timer 2 Value [15:8]**<br>These bits indicate the current timer value. |

# 14 T2CCU

The block diagram of the T2CCU module is shown in **Figure 14-1**. It consists of the standard Timer 2 and a Capture/compare unit (CCU). The Capture/Compare Timer (CCT) is part of the CCU. Control is available in the CCU to select individually for each of its 16-bit capture/compare channel, either the Timer 2 or its own Capture/Compare Timer (CCT) as the time base. Both timers have a resolution of 16 bits. The clock frequency of T2CCU, $f_{T2CCU}$, could be set at PCLK frequency or 2 times the PCLK frequency.

The T2CCU can be used for various digital signal generation and event capturing like pulse generation, pulse width modulation, pulse width measuring etc. Target applications include various automotive control as well as industrial (frequency generation, digital-to-analog conversion, process control etc.).

## 14.1 Features

T2CCU has the following features:

- Option to select individually for each channel, either Timer 2 or Capture/Compare Timer as time base
- Extremely flexible Capture/Compare Timer count rate by cascading with Timer 2
- Capture/Compare Timer may be ‘reset’ immediately by triggering overflow event
- 16-bit resolution; maximum frequency = 2 x peripheral clock frequency
- Six compare channels in total
- Four capture channels multiplexed with the compare channels, in total
- Shadow register for each compare register
  - Transfer via software control or on timer overflow.
- Compare Mode 0: Compare output signal changes from the inactive level to active level on compare match. Returns to inactive level on timer overflow.
  - Active level can be defined by register bit for channel groups A and B.
  - Support of 0% to 100% duty cycle in compare mode 0.
- Compare Mode 1: Full control of the software on the compare output signal level, for the next compare match.
- Concurrent Compare Mode with channel 0
- Capture Mode 0: Capture on any external event (rising/falling/both edge) at the 4 pins T2CC0 to T2CC3.
- Capture Mode 1: Capture upon writing to the low byte of the corresponding channel capture register.
- Capture mode 0 or 1 can be established independently on the 4 capture channels.

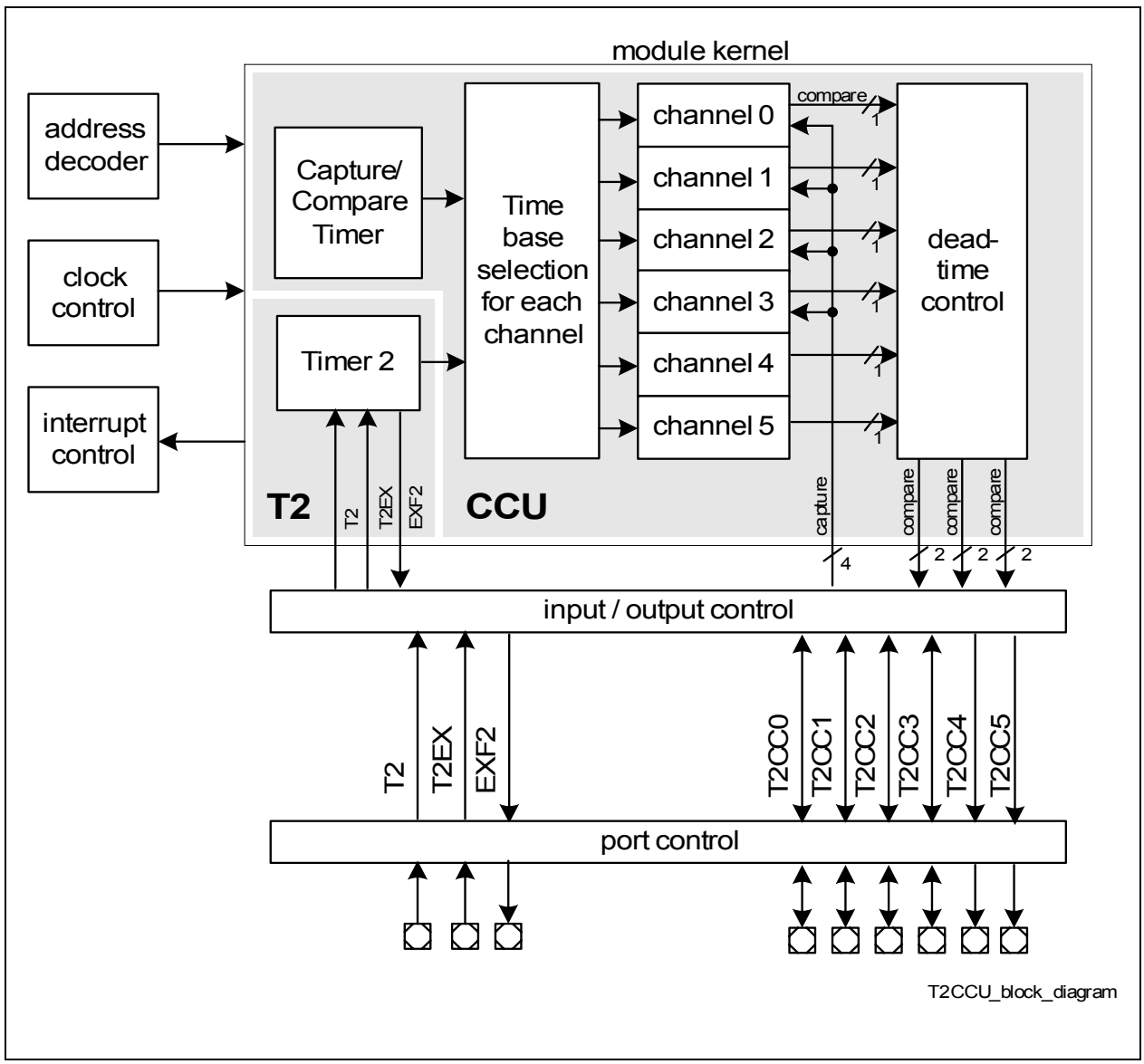



**Figure 14-1 T2CCU Block Diagram**

## 14.2 Requirements of Timer 2

By default, all T2CCU channels select the 16-bit Timer 2 as the time base. Timer 2 can be started in timer function (C/$\overline{T2}$), by setting bit TR2 or by external start via T2EX. Prior to this, the Timer 2 reload value on overflow should be initialized in the registers RC2H/L. The Up/Down Counter function of Timer 2 must be disabled (DCEN = 0), so that Timer 2 functions as a pure up timer only. T2PRE should be set to select the timer operating frequency, this setting depends on the application requirement.

Timer 2 supports operational frequencies of $f_{T2CCU}$, $f_{T2CCU}/2$, $f_{T2CCU}/4$, $f_{T2CCU}/8$, $f_{T2CCU}/16$, $f_{T2CCU}/32$, $f_{T2CCU}/64$, or $f_{T2CCU}/128$. Thus, the 16-bit timer is incremented in the fastest case, twice in every machine cycle and in the slowest case, once in every sixty-four machine cycles if T2CCU is clocked at PCLK frequency.

The timer overflow flag TF2 is set when there is an overflow of the timer contents, and can generate an interrupt if enabled.

Timer 2 may be stopped any time by clearing bit TR2.

*Note: **Chapter 13.2** describes the detailed Timer 2 operations.*

## 14.3 Capture/Compare Timer of T2CCU

The Capture/Compare Timer (CCT) is part of the Capture/Compare Unit (CCU) in T2CCU and is a dedicated time base for the T2CCU capture/compare operations.

CCT is basically a free-running counter that is started by setting bit T2CCU_CCTCON.CCTST. When this bit is set, CCT will start counting from the value in the CCTH/L timer register. The timer counts up to $FFFF_H$ and on overflow, the timer will be reloaded with the value initialized in the registers CCTRELH/L, and count up from this value. CCT can be clocked at frequencies of $f_{T2CCU}$, $f_{T2CCU}/2$, $f_{T2CCU}/4$, $f_{T2CCU}/8$, $f_{T2CCU}/16$, $f_{T2CCU}/32$, $f_{T2CCU}/64$, $f_{T2CCU}/128$, $f_{T2CCU}/256$, $f_{T2CCU}/512$, $f_{T2CCU}/1024$, or $f_{T2CCU}/2048$. Thus, CCT is incremented in the fastest case, twice in every machine cycle and in the slowest case, once in every 1024 machine cycles if T2CCU is clocked at PCLK frequency.

The timer overflow flag CCTOVF is set on overflow of the timer contents, and can generate an interrupt if enabled by bit CCTOVEN.

The CCT timer registers CCTH/L may be written any time to (re-)initialize the timer value, even when it is running. When CCT is not running, write to CCTL/H updates the corresponding byte of timer. However when CCT is running, CCTH has to be written first. On write to CCTL, the values in CCTH and CCTL will be updated to the timer which starts counting from this value.

CCT may be stopped any time by clearing bit CCTST. The timer registers will hold the last timer count.

### Triggered CCT Timer Overflow

A special feature of the CCT timer is the possibility to ‘reset’ the timer immediately by a Triggered Timer Overflow event. By ‘reset’, CCT timer is reloaded and start counting from the value of the reload registers, while all actions associated with the operation mode of the timer channels on base timer overflow will be executed. On setting bit T2CCU_CCTBSEL.CCTTOV, the CCT timer overflow event is triggered asynchronously to accomplish all the above. An example usage for a channel operating in compare mode 0 is illustrated in **Figure 14-2**. For more details on compare mode 0, refer to **Section 14.5.1.1**.

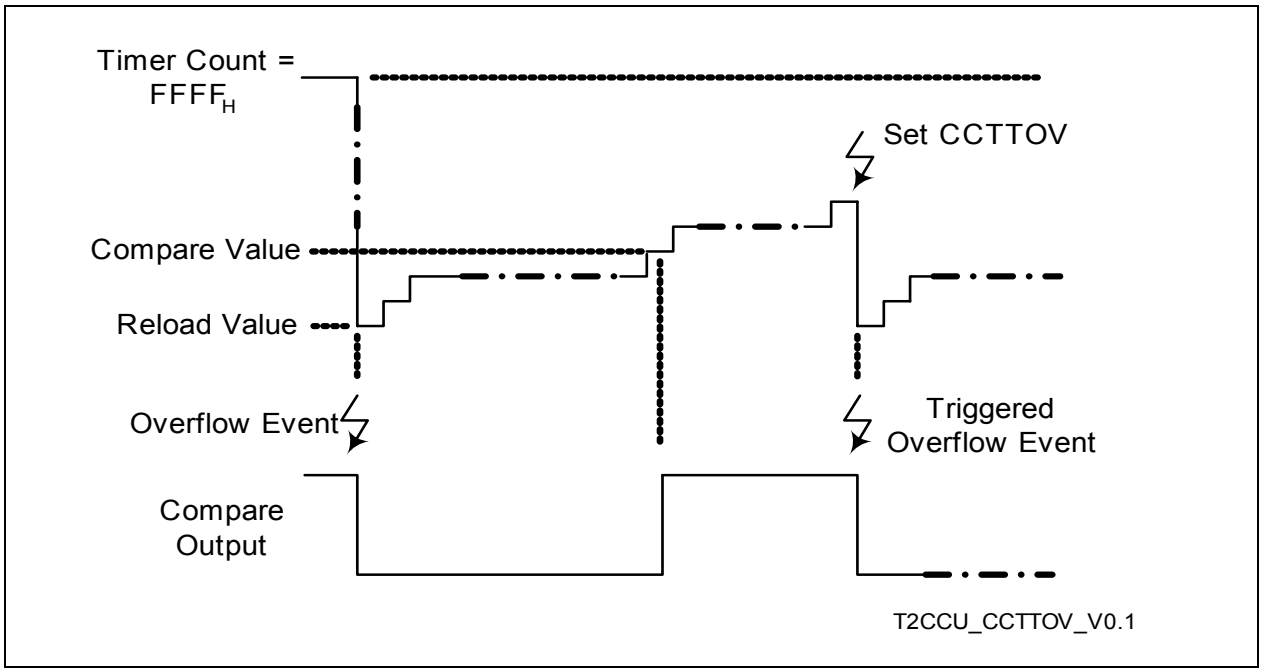



**Figure 14-2 Example of Channel in Compare Mode 0 on ‘Triggered Timer Overflow’ Event**

## 14.3.1 Synchronized Timer Starts

As described in above sections, it is possible to individually start/stop the timers: Timer 2 by TR2 bit; Capture/Compare Timer by CCTST bit. If the user wants to synchronize the timer starts, then the procedure required is:

1. Ensure CCT timer is not already running,
2. Clear bit TR2 (stop Timer 2) and ensure timers are properly initialized,
3. Set bit TIMSYN to enable synchronized timer starts,
4. Set bit CCTST.

Step 3 and 4 can be combined in a single write operation to the sfr T2CCU_CCTCON.

Synchronized timer start is defined for the case where CCT timer is not already running. When T2CCU_CCTCON bits TIMSYN = 1 and CCTST = 0, setting CCTST sets also the

Timer 2 T2CON.TR2 bit. Therefore, both timers are started simultaneously. In the case Timer 2 is already running when CCTST is set, Timer 2 is not affected and continues normal counting operation.

To stop the timers, the respective timer bits can be cleared independently.

### 14.3.2 Cascading Timers For Flexible Count Rate

Especially for applications requiring the base timer to run at flexible count rate, it is allowed to cascade the Timer 2 with the CCT timer. If the bit T2CCU_CCTBSEL.CASC is set, CCT timer will count at the overflow rate of Timer 2. Refer to **Figure 14-3** where Timer 2 is configured as a timer.

An example usage could modify the reload value of Timer 2 for the next overflow, whenever required by the application. If it fits the application, Timer 2 may still be used by the channels as the time base for capture or compare operations.

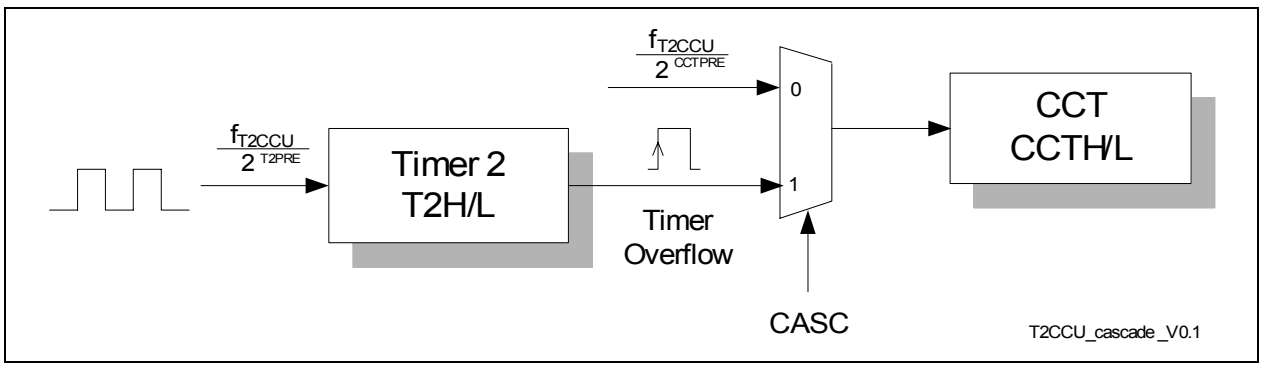



**Figure 14-3 Cascading Timer 2 to CCT Timer**

*Note: Timer 2 may be configured in timer mode or as an external event counter via pin T2.*

### 14.3.3 Dead-time with CCT Timer

The generation of (complementary) signals with dead-time for the highside and the lowside switches of one power inverter phase is based on twin grouping of the consecutive compare channels – one from group A and the other from group B. The three compare channel pairs therefore comprise of channel 0 with 1, channel 2 with 3, and channel 4 with 5. All channels must be enabled to run in compare mode 0 with CCT timer as the time base. The compare value must be set the same for the channels of a channel pair. It is also required that all the highside and lowside switches have the same active polarity. If the highside switch should be active while the CCT timer value is above the compare value, then the lowside switch should be active while the CCT timer value is below the compare value. The polarity on compare match is controlled via the bits T2CCU_COCON.POLA and POLB for compare group A and B respectively – settings of which must be inverted.

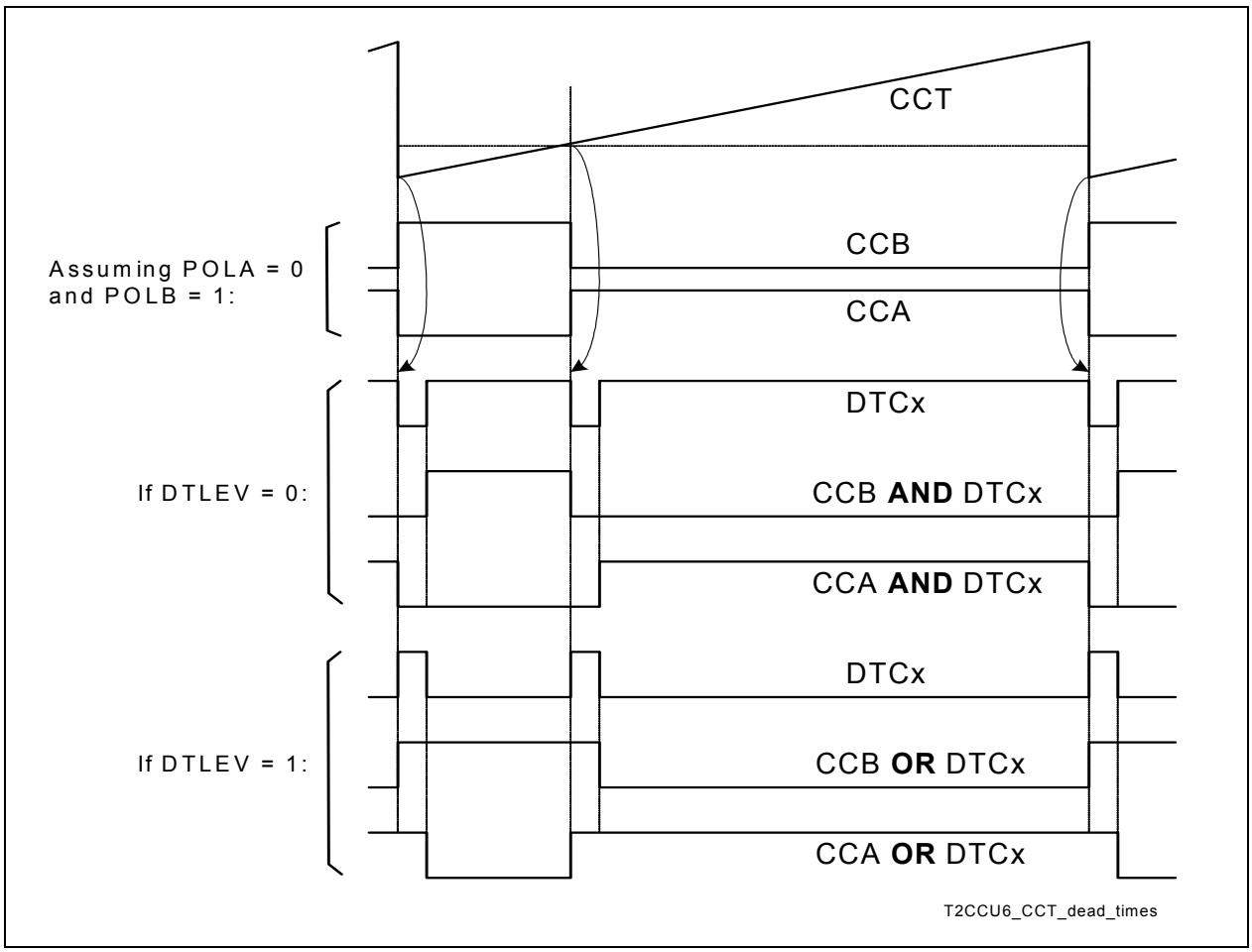



**Figure 14-4 PWM-signals with Dead-time Generation (Illustrated for One Channel Pair)**

In most cases, the switching behavior of the connected power switches is not symmetrical concerning the times needed to switch on and to switch off. A general problem arises if the time taken to switch on is less than the time to switch off the power device. This leads to a short-circuit in the inverter bridge leg, which may damage the complete system. In order to solve this problem by hardware, T2CCU contains a programmable dead-time counter, which delays the passive to active edge of the switching signals (the active to passive edge is not delayed); see **Figure 14-4**.

The dead-time generation logic (see **Figure 14-5**) is built in a similar way for all three channel pairs. Each instance of the CCT timer overflow or compare match event (of channel from group A) triggers the corresponding dead-time counter (8-bit down counter, clocked at CCT count rate) and toggles DTCx to the active level. The trigger pulse (DTCx_rl) leads to a reload of the dead-time counter with the value programmed in bit field T2CCU_CCTDTCL.DTM. This reload can only take place if the dead-time feature is enabled by bit T2CCU_CCTDTCH.DTEx and while the counter is zero.

While counting down (zero is not yet reached), the control signal DTCx stays at active level. On reaching the counter value zero, the dead-time counter stops counting and the signal DTCx toggles again to the inactive level. The DTCx signal is logically manipulated with the CCT modulation signals CCA and CCB, leading to a delay of the passive to active edge of the resulting compare output signal, which is shown in **Figure 14-4**. The active level of the DTCx and the manipulation of the modulation signals is defined and handled differently depending on setting of T2CCU_CCTDTCH.DTLEV (defines the passive level of the switches).

*Note: Signal DTCx toggles only if dead-time is enabled by T2CCU_CCTDTCH.DTEx. When DTEx is set from the clear state, DTCx is held at inactive level according to setting of DTLEV until the next dead-time event.*

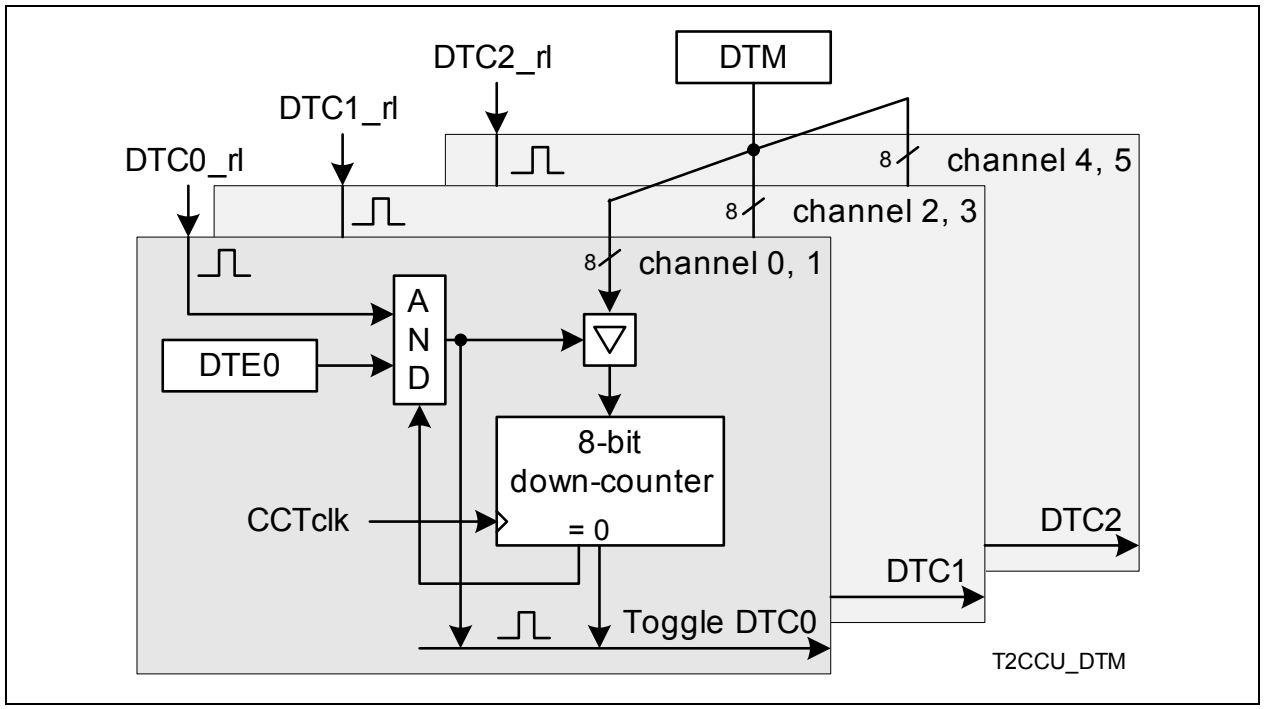



**Figure 14-5 Dead-time Counter**

Each of the three channel pairs works independently with its own dead-time counter, the trigger and enable signals. The value of bit field DTM is valid for all three channel pairs.

## 14.4 T2CCU Capture/Compare Channels

The T2CCU allows the following operating modes:

- Compare:Up to 6 PWM output signals with 65536 steps at maximum
- Capture:Up to 4 high speed capture inputs

The capture/compare resolution for each channel is per the timer base operating frequency.

**Figure 14-6** shows a diagram of Timer 2 and the Capture/Compare Unit(CCU) for a channel. The I/O pins are generally assigned as alternate function to general purpose I/O port. As shown in **Table 14-1**, the I/Os T2CC0, 1, 2, 3, 4 always share the same pin as the external interrupts EXINT3, 4, 5 and 6 respectively. The corresponding external interrupt request flags are set for the capture/compare function. The respective timer overflow flags are also set on timer overflow.

**Table 14-1 Capture and Compare I/Os of T2CCU**

| Pins | Function |
|---|---|
| T2CC0 / EXINT3 | Compare output/Capture input for channel 0 |
| T2CC1 / EXINT4 | Compare output/Capture input for channel 1 |
| T2CC2 / EXINT5 | Compare output/Capture input for channel 2 |
| T2CC3 / EXINT6 | Compare output/Capture input for channel 3 |
| T2CC4 | Compare output for channel 4 |
| T2CC5 | Compare output for channel 5 |

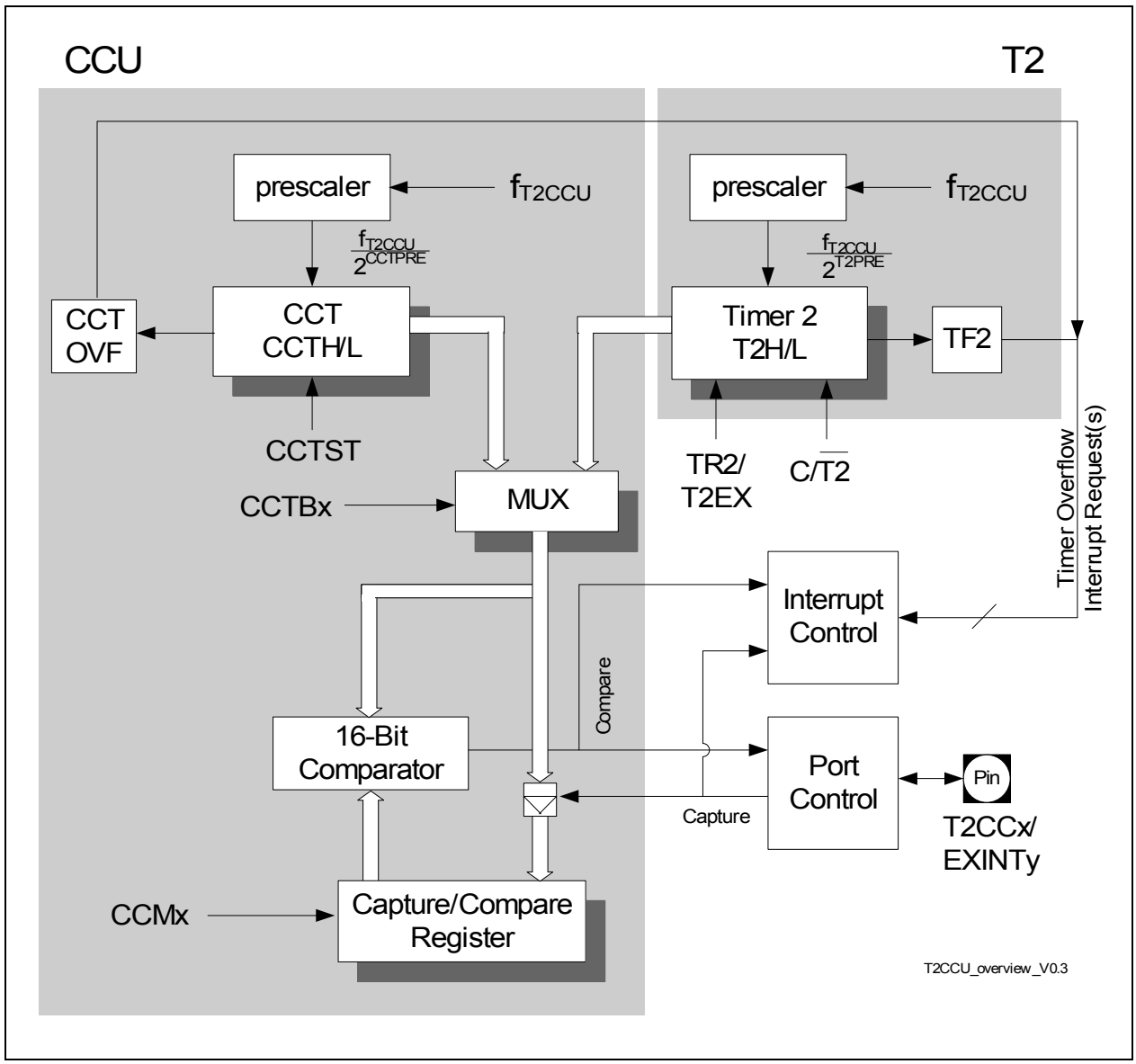



**Figure 14-6 Timer 2 Capture/Compare Function for Channel x**

## 14.5 Capture/Compare Operation

There are altogether six compare channels and altogether four capture channels, which can be individually enabled. The capture channels can be independently selected for capture mode 0 or 1. A general selection of the capture/compare function for each channel is done in register T2CCU_CCEN.

In case of compare mode selected, the register T2CCU_COCON allows to configure the compare mode for all compare channels. The compare channels are further divided into two groups - A and B. If the channels are configured for compare mode respectively, group A consists of the channels CC0, CC2 and CC4 while group B consists of the channels CC1, CC3 and CC5. In this way, the level of the output signal on compare match till timer overflow (active level) can be configured separately for the channels of the two groups.

In case of capture mode selected, the external event for the capture can be configured by the external interrupt SFR bits. The active trigger could be selected individually for each channel for rising, falling or both edge triggered.

*Note: For defined behavior, it is required of the user to initialize properly all the T2CCU related SFRs before enabling any T2CCU capture or compare mode.*

### 14.5.1 Compare Function

The T2CCU compare output is only actively driven on enabling the compare function. When the compare function is enabled by writing to a channel's CCMx bits, the respective compare output is latched (as initial value) based on the settings to the channel SFR bits.

In compare mode, the 16-bit value stored in a compare register is compared with the contents of the timer. When the timer count matches the compare register value, the corresponding channel's output signal is toggled at the pin, and an interrupt is requested if enabled.

The contents of a compare register can be regarded as a "time stamp" at which the corresponding output reacts in a predefined way (with either a positive or negative transition). Variation of this "time stamp" changes the waveform of a rectangular output signal at the pin. As a variation of the duty cycle of a periodic signal, this may be used for pulse width modulation as well as for a continually controlled generation of any kind of square waveforms. Two compare modes plus one concurrent compare mode are available to cover a wide range of possible applications.

The active compare mode is selected by bit T2CCU_COCON.COMOD, and applies to all enabled compare channels. In all compare modes, the new output value arrives at the port pin within the same T2CCU clock cycle in which the internal compare signal is activated.

The compare match signal stays high for as long as the comparator inputs are equal (at least one T2CCU clock cycle). The rising edge of the compare match signal is the active edge for all compare match operations.

### Shadow Registers for Compare Registers

The compare registers of the T2CCU cannot be accessed directly by user code. Update of channel compare value is by shadow transfer from the corresponding SFR T2CCU_CCx. The user may choose to select one out of two events to start a shadow transfer to the internal compare registers.

1. By setting the SFR bit T2CCU_COSHDW.TXOV = 1, shadow transfer is started for the compare channels on the overflow of their corresponding timer base. In this case, the update of the compare registers is controlled by hardware and is therefore independent from any service delay.
2. By setting the SFR bit T2CCU_COSHDW.ENSHDW = 1, shadow transfer is started for each channel when the corresponding SFR bit COOUTx is set under user control. COOUTx will be cleared automatically by hardware at the next T2CCU clock cycle, after being set.

In both cases, the rising edge of the event pulse signal is the active edge for the shadow transfer of the compare register(s). The advantage of the shadow transfer mechanism is that it allows glitch-free pulse width modulation, which can occur due to the two instructions needed to update the 16-bit compare registers.

Note that it is not allowed to set both bits TXOV and ENSHDW at the same time. If bit TXOV is set to enable shadow transfer on timer overflow, bit ENSHDW will be cleared by internal hardware to disable shadow transfer by software.

**Figure 14-7** illustrates the shadow transfer mechanism.

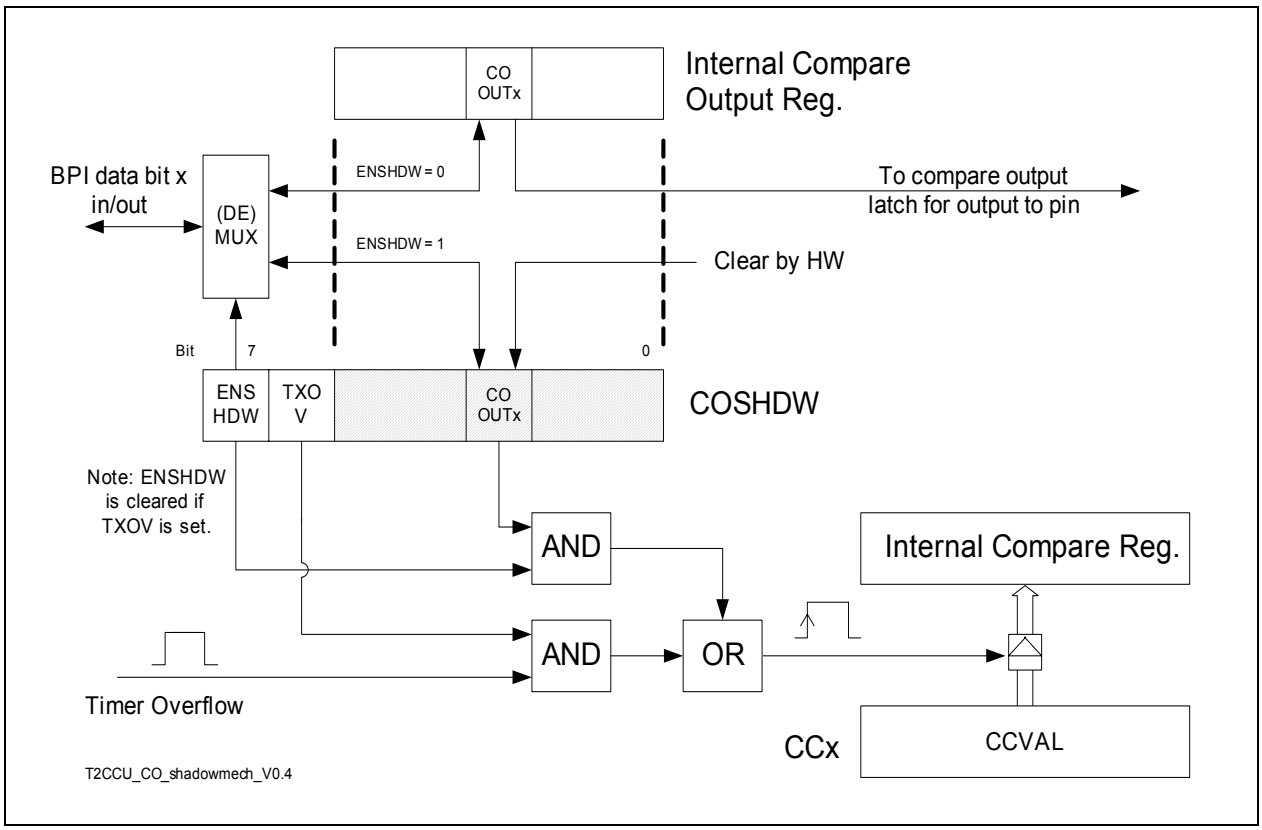



**Figure 14-7 Compare Value Shadow Transfer Mechanism and The Dual Function of Bits COOUTx**

### Mapped Compare Output Bits

To enable dual function on the COOUTx bits of SFR T2CCU_COSHDW, bit ENSHDW is used for the control of access to these bits. When ENSHDW = 1, the SFR bits COOUTx can be directly accessed for control of shadow transfer of compare value. When ENSHDW = 0, writing or reading to COOUTx bits actually access an internal compare output register instead of the COSHDW bits. The corresponding bits of this internal compare output register control the output polarity of the compare channels in compare mode 1 and concurrent compare mode.

**Figure 14-7** illustrates the dual function of the bits COOUTx.

### 14.5.1.1 Compare Mode 0

When compare mode 0 is selected, on setting CCMx to enable compare function, the compare output is latched,

- To the active level if the base timer content are equal or bigger than the compare register content, or,
- To the inactive level if the base timer content is less than the compare register content.

The active level is defined for each channel by its compare group A or B, via the SFR bit T2CCU_COCON.POLA or POLB.

In compare mode 0, when the timer and compare register contents match, the output signal changes from the inactive level to active level. The compare output signal goes back to inactive level on timer overflow.

Compare mode 0 is ideal for generating pulse width modulated output signals, which in turn can be used for digital-to-analog conversion via a filter network or by the controlled device (e.g. the inductance of a DC or AC motor). Mode 0 may also be used for providing output clocks with initially defined period and duty cycle. This usage needs the least CPU time. Once set up, the output goes on oscillating without any CPU intervention.

**Figure 14-8** illustrates the function of compare mode 0. In this function, the compare output signal is directly controlled by the setting of SFR bit T2CCU_COCON.POLA/B and the events: timer overflow and compare match.

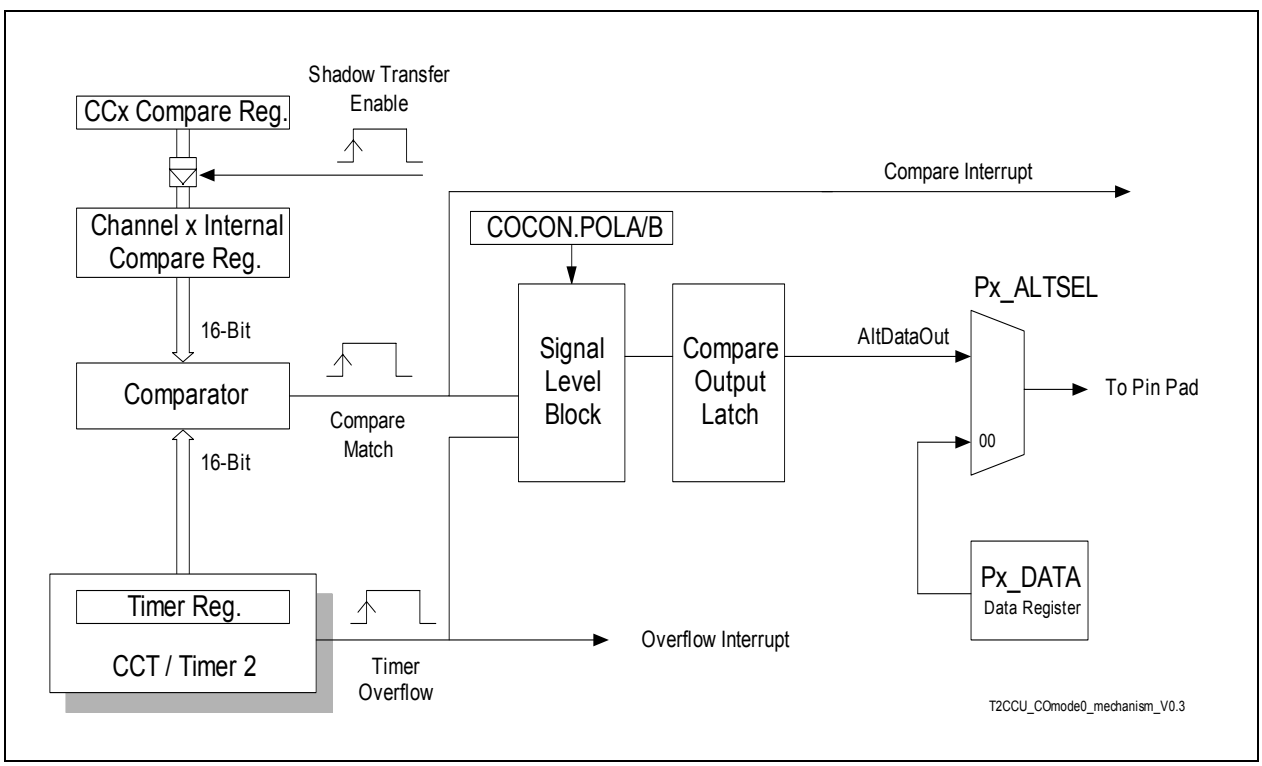



**Figure 14-8 Functional Diagram of Compare Mode 0**

**Figure 14-9** illustrates the channel output when POLA/B is set to ‘0’. Interrupts may be generated on timer overflow or compare match, and update the next compare value etc., in the interrupt service routine.

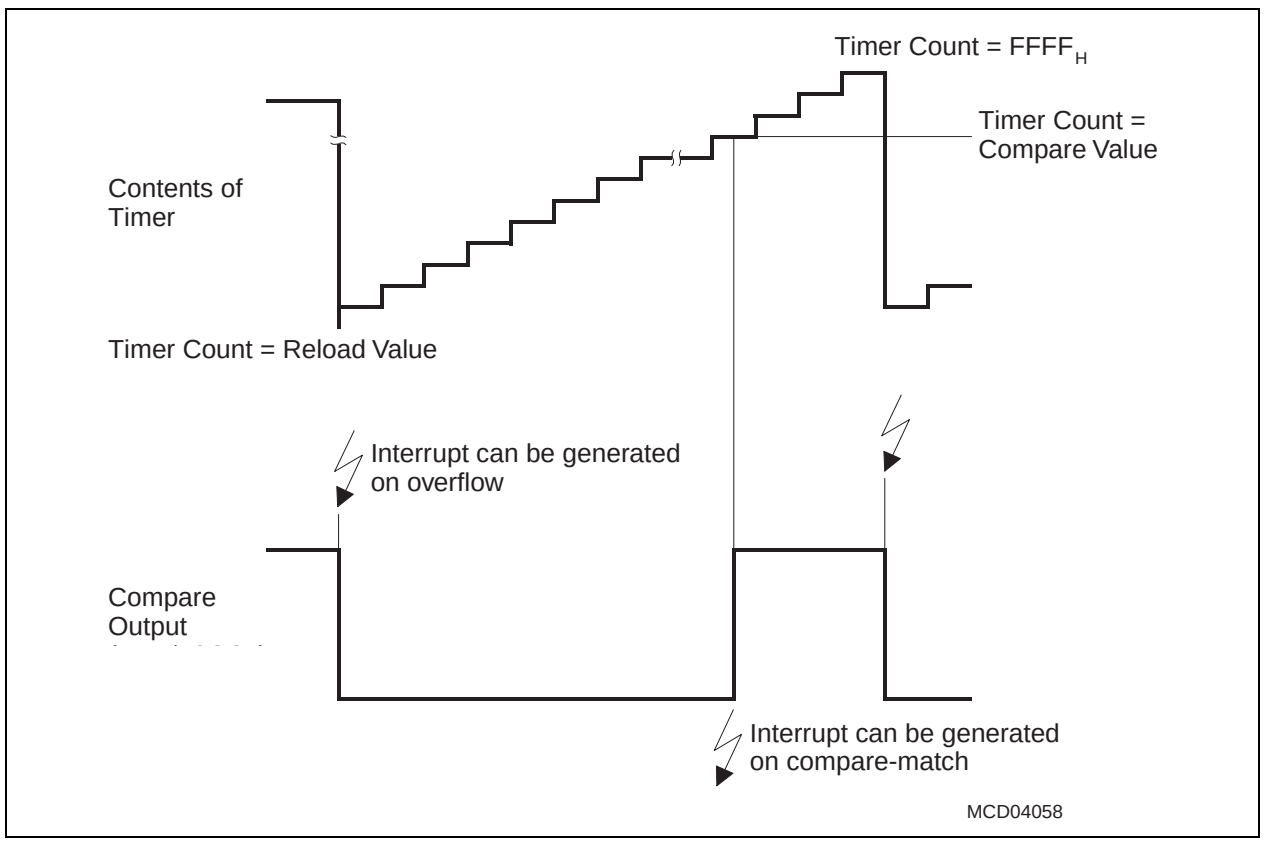



**Figure 14-9 Channel Output in Compare Mode 0 (POLA/B = 0)**

### PWM Signal from 0% to 100% Duty Cycle in Compare Mode 0

The active level of a PWM signal refers to the compare output signal level from the time of compare match till the next timer overflow.

To support a 100% duty cycle for a compare channel with T2CCU_COCON.POLA/B = 0 (active level = high), the 16-bit channel x compare value must be set equal to the timer overflow reload value. In this case, the internal hardware will not cause the compare output signal to the inactive level on timer overflow, and the output stays at the active level until the next timer overflow when the compare value is no more equal to the timer reload value.

To support a 0% duty cycle for a compare channel with T2CCU_COCON.POLA/B = 0, the channel x compare value must be smaller than the timer overflow reload value. In this case, the timer reload value must not be zero. The compare match event will not happen and the compare output stays at the inactive level (low).

The above mechanism applies for compare channel with T2CCU_COCON.POLA/B = 1 correspondingly.

### 14.5.1.2 Compare Mode 1

When compare mode 1 is selected, on setting CCMx to enable compare function, the compare output is latched with the value initialized to bit COOUTx of the internal compare output register.

In compare mode 1, the software adaptively determines the transition of the output signal. It is commonly used when output signals are not related to a constant signal period (as in a standard PWM Generation) but must be controlled very precisely with high resolution and without jitter. In compare mode 1, the compare output signal level is under the full control of the software, for the next compare match.

If compare mode 1 is enabled and the software writes to COOUTx bit(s) of the internal compare output register (in this case, bit T2CCU_COSHDW.ENSHDW must be cleared), the new value will not appear at the output pin until the next compare match occurs. Thus, one can choose independently for each compare channel, whether the output signal is to toggle (1-to-0 or 0-to-1) or to remain at the same level, when the channel timer count matches the contents of the channel internal compare register.

**Figure 14-10** illustrates the function of compare mode 1. In this function, the bit COOUTx of the internal compare output register will only be transferred to the compare output latch as alternate data out, on the next compare match event.

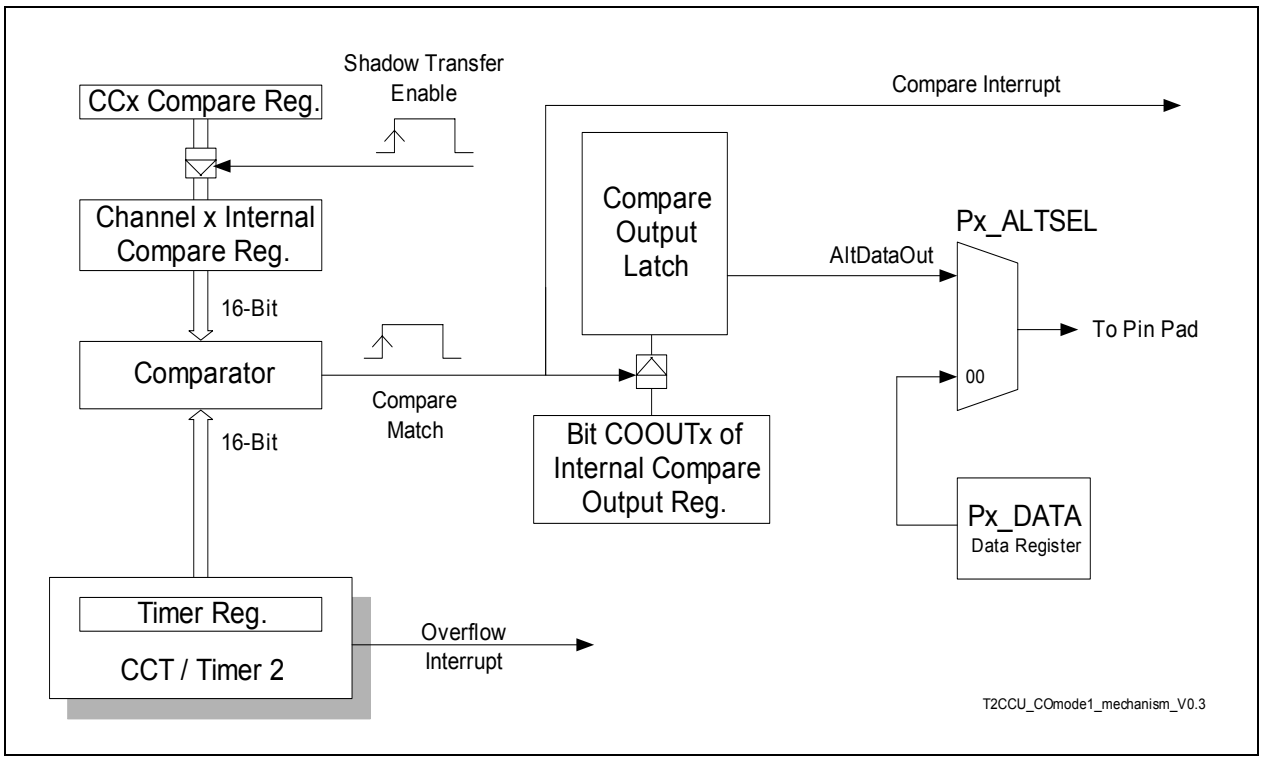



**Figure 14-10 Functional Diagram of Compare Mode 1**

### 14.5.1.3 Concurrent Compare Mode

When concurrent compare mode is selected, on setting CCMx to enable compare function, the compare outputs are latched with the corresponding values initialized to the COOUTx bits of the internal compare output register.

Concurrent Compare is similar to compare mode 1, with the additional capability to manipulate the output at more than one compare channel. Like in compare mode 1, the software updates the COOUTx bits of the internal compare output register (bit T2CCU_COSHDW.ENSHDW must be cleared), so that up to six compare channel outputs are updated concurrently when channel 0 compare match occurs. When this happens, all enabled compare channel outputs will be updated with the corresponding value in the internal compare output register.

Concurrent compare is an ideal and effective mode for applications where more than one synchronous output signal is to be generated. Such applications could include for e.g., a multiple-phase stepper motor control, or control of ignition coils of a car engine. These applications generally require predefined bit-patterns to be put to an output port at precisely predefined moment.

**Figure 14-11** illustrates the function of concurrent compare mode. Compare channel 0 must be enabled and initialized for proper operation.

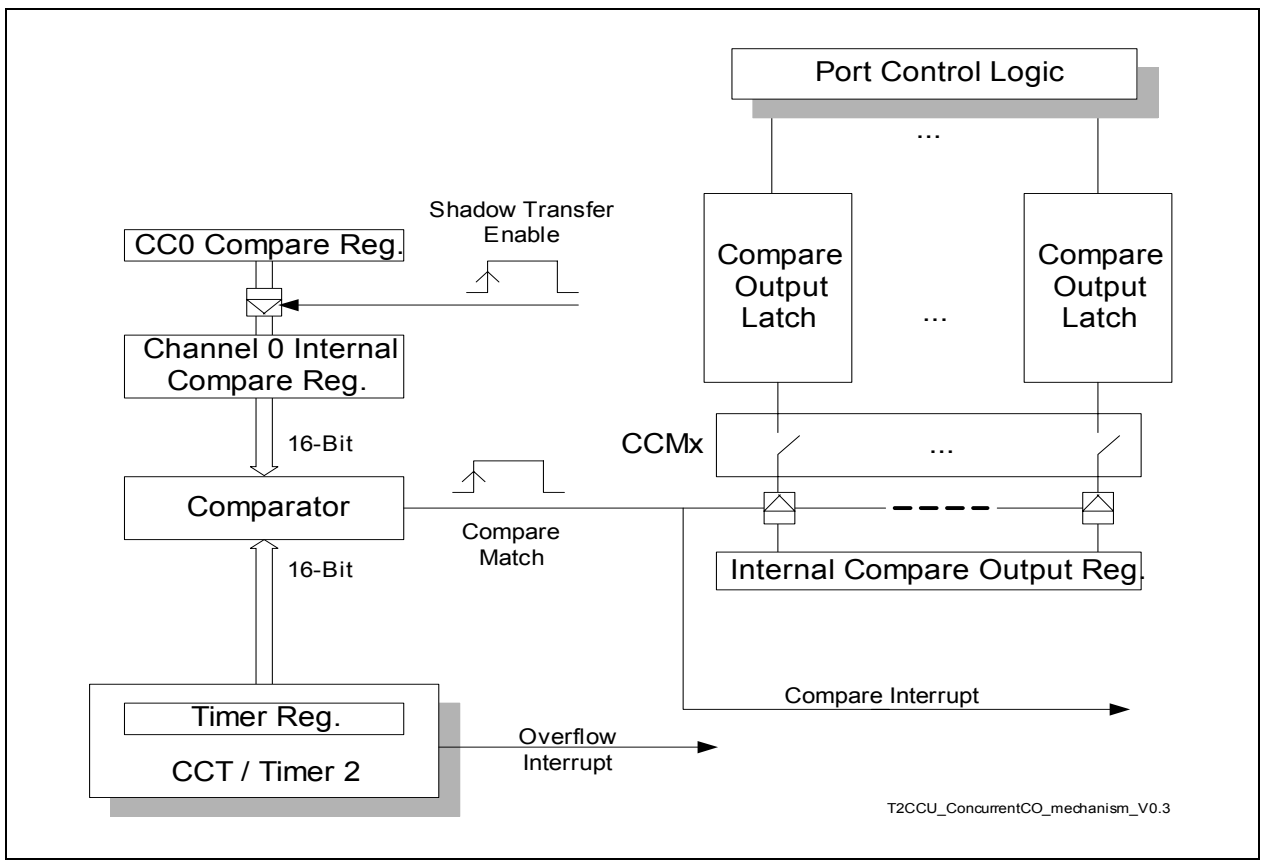



**Figure 14-11 Functional Diagram of Concurrent Compare Mode**

**Figure 14-12** illustrates two examples of how four rectangular waveforms can be generated at the port using a pattern table. The patterns are written to the internal compare output register (with T2CCU_COSHDW.ENSHDW = 0) before the corresponding timer count is reached. The (future) timer count at which the pattern shall be available at the port must be loaded to the compare shadow registers T2CCU_CC0H and T2CCU_CC0L, and transferred to the internal compare registers. To manage the waveform generation, one example uses a time schedule table with list of compare values, the other example makes use of a fixed compare value (such as $FFF0_H$) with shorter timer period. In these ways, each channel can be controlled at predefined time(s) by the user on whether to toggle or not.

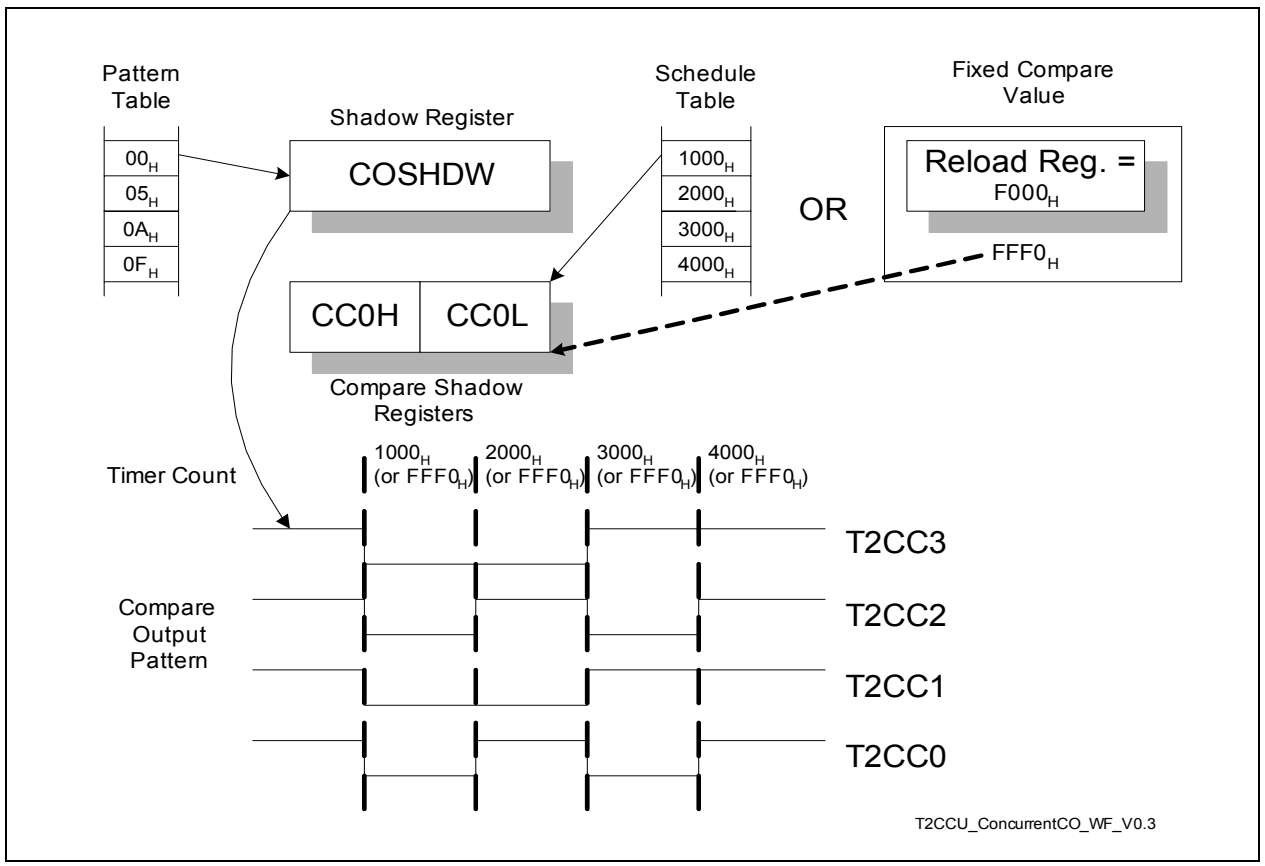



**Figure 14-12 Two Examples of Waveform Generation in Concurrent Compare Mode (by Schedule Table of Compare Values or Fixed Compare Value)**

### 14.5.1.4 Using Interrupts in Combination with the Compare Function

**Reserved External Interrupts**

Each of the compare channels T2CCU_CC0-3 share the same port pin with an external interrupt as alternate functions at port pins. When compare mode is enabled and the corresponding port pin is set to output mode for use as compare output, the input driver of the port pin is disabled. Therefore the corresponding external interrupt inputs cannot be used for any purpose.

However, the interrupt function of the external interrupts can still be enabled for the compare match interrupts. In this case, the corresponding external interrupt lines are connected directly to the internal compare match signal.

**Compare Match Interrupt**

The compare match interrupt (including that for channels 4 and 5) is enabled together with the capture interrupt by the IEN1.EXM bit for all channels. A channel compare match occurs when timer count matches the channel register contents. Besides manipulating the corresponding channel compare output, the corresponding compare match flag IRCON0.EXINTx or T2CCU_COMOD.CMxF is set for the interrupt request. For this, the interrupt edge trigger for compare match must be set to rising edge in the corresponding EXICON register. The compare match interrupt service routine can be used to set up the parameters for the next channel compare event, such as change the contents of the compare registers or to determine the level of the compare port outputs for the next “compare match”.

The main advantage in using compare match interrupts is to avoid unintentional overwriting of compare/compare output registers before a match has been reached. This could happen when the CPU writes to the compare registers without knowing about the actual timer count.

*Note: As an alternative, the timer overflow interrupt may be used in place of the compare match interrupt.*

### 14.5.2 Capture Function

This mode allows the measurement of external signal pulse widths through the Capture/Compare registers CC0-3. Each of the 16-bit registers can be used to latch the current 16-bit count of the timer base of the corresponding channel. Two capture modes are provided for this function.

In mode 0, any external event (rising/falling/both edge) at pins T2CC0 to T2CC3 latches the channel timer contents to the corresponding channel capture register. The capture edge is defined by the corresponding external interrupt set up (each capture input share the same pin with an external interrupt). The external input(s) is sampled in every CCLK cycle. When a sampled input shows a low (high) level in one CCLK cycle and a high (low) in the next CCLK cycle, a transition is recognized. The corresponding external interrupt request flag(s) (EXINT3 to EXINT6) is set. If the interrupt(s) is enabled, the CPU will vector to the appropriate interrupt service routine. The timer contents are latched to the corresponding capture register in the T2CCU clock cycle following the one in which the transition was identified.

In mode 1, a capture will occur upon writing to the low byte of the corresponding channel capture register. The software is able to read the contents of the channel timer “on-the-fly”. The value written to the capture register is irrelevant for this function. The timer contents will be latched into the channel capture register following the write instruction. In this mode no interrupt request will be generated.

**Figure 14-13** shows the capture operation of the Capture/Compare channels.

The two capture modes can be established individually for each capture channel by bits in SFR T2CCU_CCEN (Capture/Compare enable register). That means, in contrast to the compare modes, it is possible to simultaneously select mode 0 for one capture channel and mode 1 for another channel.

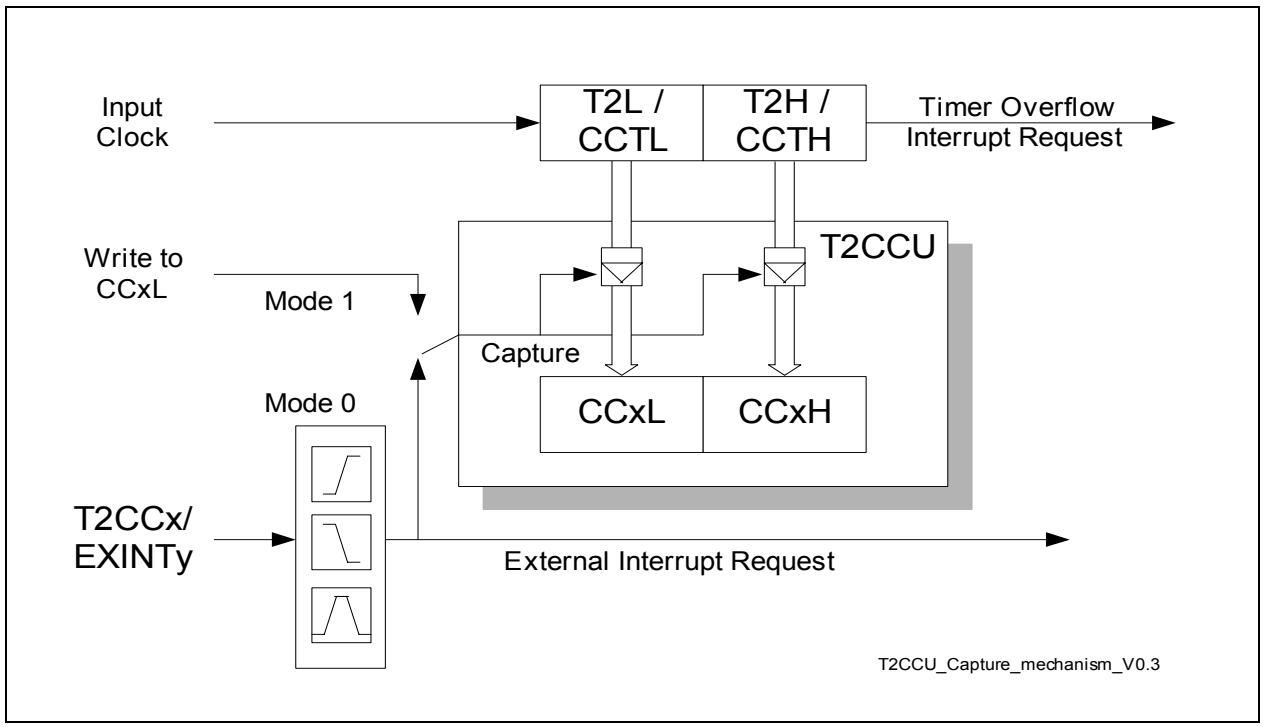



**Figure 14-13 Functional Diagram of Capture Mode**

## 14.6 T2CCU Registers

This section describes all T2CCU special function registers. The Timer 2 and interrupt related SFRs are also listed, as shown in **Table 14-2**.

*Note: All T2CCU register names shall be referenced fully with the module name prefix "T2CCU_".*

**Table 14-2 Registers Overview**

| Name | Description |
|---|---|
| **T2CCU Special Function Registers** | |
| CCTBSEL | Capture/Compare Time Base Select Register |
| CCTRELL | Capture/Compare Timer Reload Register Low |
| CCTRELH | Capture/Compare Timer Reload Register High |
| CCTL | Capture/Compare Timer Register Low |
| CCTH | Capture/Compare Timer Register High |
| CCTCON | Capture/Compare Timer Control Register |
| CCTDTCL | Capture/Compare Timer Dead-Time Control Register Low |
| CCTDTCH | Capture/Compare Timer Dead-Time Control Register High |
| CCEN | Capture/Compare Enable Register |
| COSHDW | Compare Shadow Register |
| COCON | Compare Control Register |
| CC0L | Capture/Compare Register 0 Low |
| CC0H | Capture/Compare Register 0 High |
| CC1L | Capture/Compare Register 1 Low |
| CC1H | Capture/Compare Register 1 High |
| CC2L | Capture/Compare Register 2 Low |
| CC2H | Capture/Compare Register 2 High |
| CC3L | Capture/Compare Register 3 Low |
| CC3H | Capture/Compare Register 3 High |
| CC4L | Compare Register 4 Low |
| CC4H | Compare Register 4 High |
| CC5L | Compare Register 5 Low |

**Table 14-2 Registers Overview** (cont'd)

| Name | Description |
|---|---|
| CC5H | Compare Register 5 High |
| **Related Timer 2 Special Function Registers** | |
| T2_T2CON | Timer 2 Control Register |
| T2_T2CON1 | Timer 2 Control Register 1 |
| T2_T2MOD | Timer 2 Mode Register |
| T2_RC2L | Timer 2 Reload Register, Low Byte |
| T2_RC2H | Timer 2 Reload Register, High Byte |
| T2_T2L | Timer 2, Low Byte |
| T2_T2H | Timer 2, High Byte |
| **Related Interrupt Special Function Registers** | |
| IEN0 | Interrupt Enable Register 0 |
| IEN1 | Interrupt Enable Register 1 |
| EXICON0 | External Interrupt Control Register 0 |
| EXICON1 | External Interrupt Control Register 1 |
| IRCON0 | Interrupt Request Register 0 |

### 14.6.1 Register Mapping

The BPI of the T2CCU supports local address extension mechanism. The SFRs of the Timer 2 are accessed in Page 0, while SFRs of the T2CCU are accessed from Page 1 to Page 3. Please refer to **Chapter 4** for details on the local address extension.

The addresses (non-mapped) of the SFRs are listed in **Table 14-3**.

**Table 14-3 SFR Address List for Pages 0-4**

| Address | Page 0 | Page 1 | Page 2 | Page 3 | Page 4 |
|---|---|---|---|---|---|
| $C0_H$ | T2CON | CCEN | COSHDW | COCON | |
| $C1_H$ | T2MOD | CCTBSEL | CC0L | CC3L | |
| $C2_H$ | RC2L | CCTRELL | CC0H | CC3H | CCTDTCL |
| $C3_H$ | RC2H | CCTRELH | CC1L | CC4L | CCTDTCH |
| $C4_H$ | T2L | CCTL | CC1H | CC4H | |

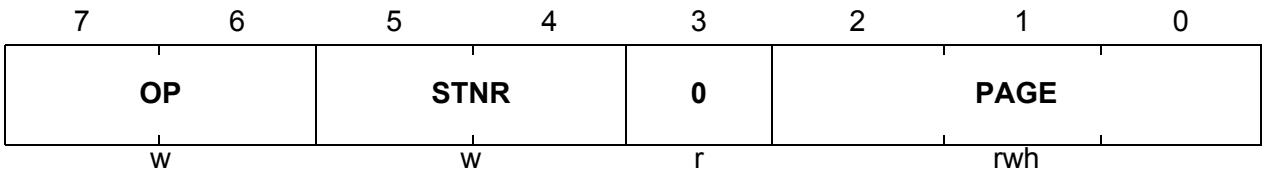
**Table 14-3 SFR Address List for Pages 0-4** (cont'd)

| Address | Page 0 | Page 1 | Page 2 | Page 3 | Page 4 |
|---|---|---|---|---|---|
| $C5_H$ | T2H | CCTH | CC2L | CC5L | |
| $C6_H$ | T2CON1 | CCTCON | CC2H | CC5H | |

*Note: The Timer 2 SFRs in T2CCU must be accessed via the paging mechanism at Page 0.*

SFR T2_PAGE, at address $C7_H$, contains the page value and the page control information.

**T2_PAGE**
**Page Register for T2CCU** **Reset Value: $00_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **PAGE** | [2:0] | rwh | **Page Bits**<br>When written, the value indicates the new page address.<br>When read, the value indicates the currently active page = addr [y:x+1] |
| **STNR** | [5:4] | w | **Storage Number**<br>This number indicates which storage bit field is the target of the operation defined by bit OP.<br>If OP = $10_B$,<br>the contents of PAGE are saved in STx before being overwritten with the new value.<br>If OP = $11_B$,<br>the contents of PAGE are overwritten by the contents of STx. The value written to the bit positions of PAGE is ignored.<br><br>00 ST0 is selected.<br>01 ST1 is selected.<br>10 ST2 is selected.<br>11 ST3 is selected. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **OP** | [7:6] | w | **Operation**<br>0X Manual page mode. The value of STNR is ignored and PAGE is directly written.<br>10 New page programming with automatic page saving. The value written to the bit positions of PAGE is stored. In parallel, the former contents of PAGE are saved in the storage bit field STx indicated by STNR.<br>11 Automatic restore page action. The value written to the bit positions PAGE is ignored and instead, PAGE is overwritten by the contents of the storage bit field STx indicated by STNR. |
| **0** | 3 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

## 14.6.2 T2CCU Registers Description

The following describes the Special Function Registers of the T2CCU kernel.

**T2CCU_CCTBSEL**
**T2CCU Capture/Compare Time Base Select Register** **Reset Value: $00_H$**

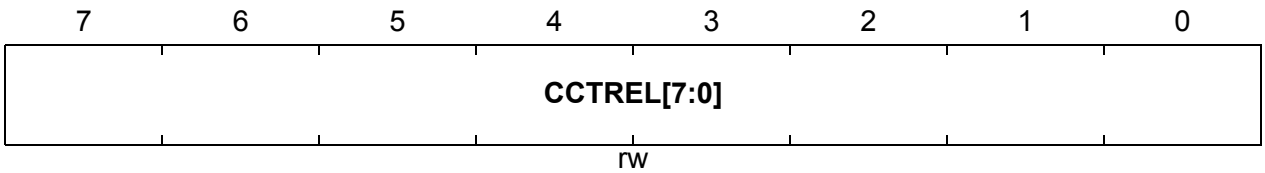
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **CASC** | **CCTTOV** | **CCTB5** | **CCTB4** | **CCTB3** | **CCTB2** | **CCTB1** | **CCTB0** |
| rw | rwh | rw | rw | rw | rw | rw | rw |

| Field | Bits | Typ | Description |
|---|---|---|---|
| **CCTBx**<br>**(x = 0-5)** | [5:0] | rw | **Channel x Time Base Select**<br>0 Selects Timer 2<br>1 Selects Capture/Compare Timer (CCT) |
| **CCTTOV** | 6 | rwh | **Trigger CCT Timer Overflow Event**<br>0 No action<br>1 CCT overflow event is triggered immediately<br>On set, this bit is held for one T2CCU clock cycle, then cleared by hardware.<br>Reading this bit always return 0. |

| Field | Bits | Typ | Description |
|---|---|---|---|
| **CASC** | 7 | rw | **Cascade Timers**<br>0 Timer 2 and CCT timer operate independently<br>1 Timer 2 is cascaded with CCT timer |

**T2CCU_CCTRELL**

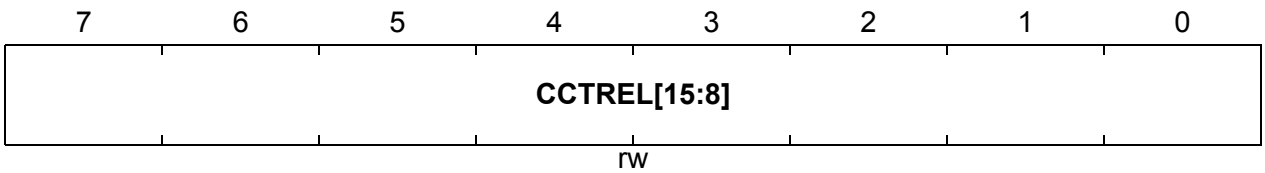
**T2CCU Capture/Compare Timer Reload Register Low** **Reset Value: $00_H$**




| Field | Bits | Typ | Description |
|---|---|---|---|
| **CCTREL[7:0]** | 7:0 | rw | **Capture/Compare Timer Low Byte Reload Value**<br>The contents of the registers are loaded into the CCT timer low byte register upon an overflow condition. |

**T2CCU_CCTRELH**

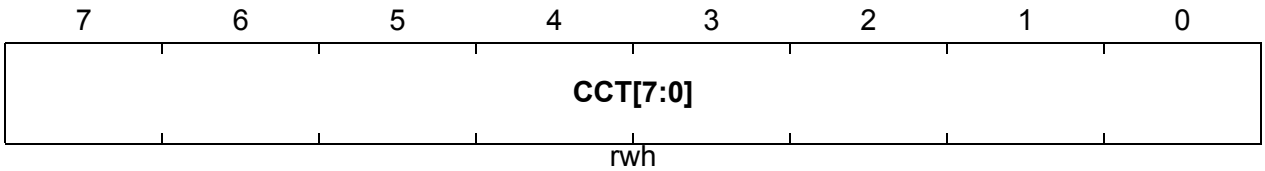
**T2CCU Capture/Compare Timer Reload Register High** **Reset Value: $00_H$**




| Field | Bits | Typ | Description |
|---|---|---|---|
| **CCTREL[15:8]** | 7:0 | rw | **Capture/Compare Timer High Byte Reload Value**<br>The contents of the registers are loaded into the CCT timer high byte register upon an overflow condition. |

### T2CCU_CCTL
### T2CCU Capture/Compare Timer Register Low

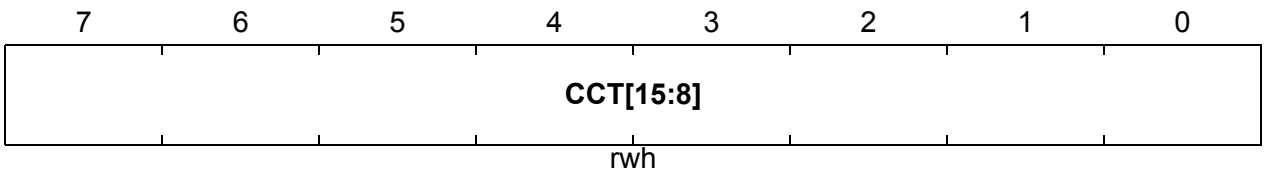
**Reset Value: $00_H$**




| Field | Bits | Typ | Description |
|---|---|---|---|
| **CCT[7:0]** | 7:0 | rwh | **Capture/Compare Timer Low Byte Value**<br>The register represents the low byte counter value of the Capture/Compare Timer.<br>When CCT is not running, write to CCTL updates the lower byte of timer.<br>To (re-)initialize CCT when it is running, CCTH has to be written first. On write to CCTL, the values CCTH and CCTL will be updated to the timer which counts from this value. |

### T2CCU_CCTH
### T2CCU Capture/Compare Timer Register High

**Reset Value: $00_H$**




| Field | Bits | Typ | Description |
|---|---|---|---|
| **CCT[15:8]** | 7:0 | rwh | **Capture/Compare Timer High Byte Value**<br>The register represents the high byte counter value of the Capture/Compare Timer.<br>When CCT is not running, write to CCTH updates the upper byte of timer.<br>To (re-)initialize CCT when it is running, CCTH has to be written first. On write to CCTL, the values CCTH and CCTL will be updated to the timer which counts from this value. |

## T2CCU_CCTCON
**T2CCU Capture/Compare Timer Control Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| CCTPRE | | | | CCTOVF | CCTOVEN | TIMSYN | CCTST |
| rw | | | | rwh | rw | rw | rw |

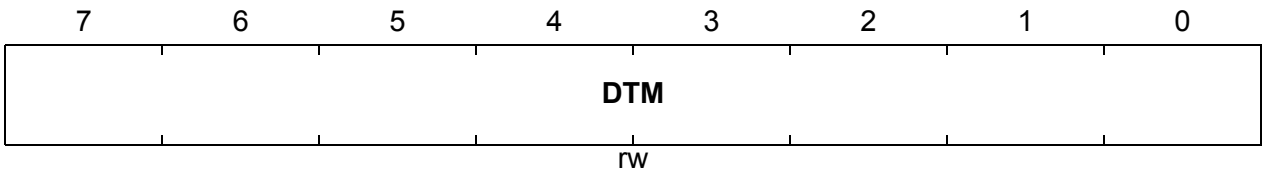
| Field | Bits | Typ | Description |
|---|---|---|---|
| **CCTST** | 0 | rw | **Capture/Compare Timer Start/Stop Control**<br>0 Stop Capture/Compare Timer<br>1 Start Capture/Compare Timer.<br>If bit TIMSYN = 1 and CCTST was 0, start also Timer 2 (if it was not already running).<br>*Note: Setting CCTST while the bit was already set will not disturb the running CCT timer operation.* |
| **TIMSYN** | 1 | rw | **Enable Synchronized Timer Starts**<br>0 Disable synchronized timer starts<br>1 Enable synchronized timer starts |
| **CCTOVEN** | 2 | rw | **Capture/Compare Timer Overflow Interrupt Enable**<br>Set to enable CCT overflow interrupt. |
| **CCTOVF**[1)] | 3 | rwh | **Capture/Compare Timer Overflow Flag**<br>Set by hardware on CCT overflow.<br>This bit has to be cleared by software. |

| Field | Bits | Typ | Description |
|---|---|---|---|
| **CCTPRE** | [7:4] | rw | **Compare/Compare Timer Prescaler Bit**<br>Selects the input clock for CCT, which is derived from the peripheral clock.<br>0000 $f_{CCT} = f_{T2CCU}$<br>0001 $f_{CCT} = f_{T2CCU}/2$<br>0010 $f_{CCT} = f_{T2CCU}/4$<br>0011 $f_{CCT} = f_{T2CCU}/8$<br>0100 $f_{CCT} = f_{T2CCU}/16$<br>0101 $f_{CCT} = f_{T2CCU}/32$<br>0110 $f_{CCT} = f_{T2CCU}/64$<br>0111 $f_{CCT} = f_{T2CCU}/128$<br>1000 $f_{CCT} = f_{T2CCU}/256$<br>1001 $f_{CCT} = f_{T2CCU}/512$<br>1010 $f_{CCT} = f_{T2CCU}/1024$<br>1011 $f_{CCT} = f_{T2CCU}/2048$<br>Others Reserved<br>*Note: It is not recommended to change CCTPRE when CCTST = 1.* |

1) In case of software and hardware access to a flag bit at the same time, hardware will have higher priority.

**T2CCU_CCTDTCL**

**T2CCU Capture/Compare Timer Dead-Time Control Register Low Reset Value: $00_H$**




| Field | Bits | Typ | Description |
|---|---|---|---|
| **DTM** | [7:0] | rw | **Dead-Time**<br>This 8-bit value defines 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. |

**T2CCU_CCTDTCH**
**T2CCU Capture/Compare Timer Dead-Time Control Register HighReset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **DTRES** | **DTR2** | **DTR1** | **DTR0** | **DTLEV** | **DTE2** | **DTE1** | **DTE0** |
| rwh | rh | rh | rh | rw | rw | rw | rw |

| Field | Bits | Typ | Description |
|---|---|---|---|
| **DTEx (x = 0-2)** | [2:0] | rw | **Dead-Time Enable for Channel Pair x**<br>0 Dead-time generation is disabled.<br>1 Dead-time generation is enabled for channel pair x. The corresponding outputs switch from the passive state to the active state (according to compare status) with the delay programmed in DTM.<br>*Note: This bit shall only be set after enabling the corresponding channels in compare mode 0 with CCT timer as the time base.* |
| **DTLEV** | 3 | rw | **Dead-Time Level**<br>This bit defines the active level of all DTCx during dead-time counting.<br>0 Low (switches are active high).<br>1 High (switches are active low). |
| **DTRx (x = 4-6)** | [6:4] | rh | **Dead-Time Run Indication Bits**<br>0 The value of the corresponding dead-time counter is 0.<br>1 The value of the corresponding dead-time counter is not 0. |
| **DTRES** | 7 | rwh | **Reset Dead-Time Counters**<br>0 No action.<br>1 The three dead-time counters are reset to 0.<br>On set, this bit is held for one T2CCU clock cycle, then cleared by hardware.<br>Reading this bit always return 0. |

**T2CCU_CCEN**
**T2CCU Capture/Compare Enable Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| CCM3 | | CCM2 | | CCM1 | | CCM0 | |
| rw | | rw | | rw | | rw | |

| Field | Bits | Typ | Description |
|---|---|---|---|
| **CCM0** | [1:0] | rw | **Capture/Compare Enable for Channel 0**<br>00 Capture/Compare disabled<br>01 Capture on active edge at pin T2CC0<br>10 Compare enabled<br>11 Capture on write operation into register CC0L |
| **CCM1** | [3:2] | rw | **Capture/Compare Enable for Channel 1**<br>00 Capture/Compare disabled<br>01 Capture on active edge at pin T2CC1<br>10 Compare enabled<br>11 Capture on write operation into register CC1L |
| **CCM2** | [5:4] | rw | **Capture/Compare Enable for Channel 2**<br>00 Capture/Compare disabled<br>01 Capture on active bedge at pin T2CC2<br>10 Compare enabled<br>11 Capture on write operation into register CC2L |
| **CCM3** | [7:6] | rw | **Capture/Compare Enable for Channel 3**<br>00 Capture/Compare disabled<br>01 Capture on active edge at pin T2CC3<br>10 Compare enabled<br>11 Capture on write operation into register CC3L |

*Note: The T2CCU output is only actively driven on enabling the compare function.*

**T2CCU_COSHDW**
**T2CCU Compare Shadow Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| ENSHDW | TXOV | COOUT5 | COOUT4 | COOUT3 | COOUT2 | COOUT1 | COOUT0 |
| rwh | rw | rwh | rwh | rwh | rwh | rwh | rwh |

| Field | Bits | Typ | Description |
|---|---|---|---|
| **COOUTx (x = 0-5)** | [5:0] | rwh | **Compare Channel x Output**<br>ENSHDW = 0<br>Any access to bit COOUTx accesses the corresponding bit of the internal 6-bit compare output register.<br>Write to bit COOUTx will change the corresponding bit of the internal compare output register.<br>Read to bit COOUTx returns the value of the bit in the internal compare output register.<br>The content of the internal compare output register is output on the corresponding T2CCx pins when a valid compare match occurs for a channel in compare mode 1 or concurrent compare mode.<br>*Note: A concurrent write operation to clear bit ENSHDW and set/clear bit(s) COOUTx is effective for initializing the internal compare output register.*<br><br>ENSHDW = 1<br>In this case, accessing bit COOUTx accesses the actual SFR bits.<br>If COOUTx is set, the contents of register CCx will be transferred to corresponding channel x internal compare register.<br>On set, COOUTx is held for one T2CCU clock cycle, then cleared automatically by hardware.<br>*Note: A concurrent write operation to set bit ENSHDW and bit(s) COOUTx (and clear bit TXOV) is effective for starting shadow transfer(s).* |
| **TXOV** | 6 | rw | **Enable Shadow Transfer On Timer Overflow**<br>Set to enable for all compare channels, transfer of shadow value in T2CCU_CCx to the corresponding internal compare registers on overflow of their respective timer base. |

| Field | Bits | Typ | Description |
|---|---|---|---|
| **ENSHDW** | 7 | rwh | **Enable Shadow Transfer With COOUTx**<br>Clear to enable access of the internal compare output register via the COOUTx bits.<br>Set to enable use of COOUTx bits for transfer of shadow value in T2CCU_CCx to the corresponding channel x internal compare register.<br>*Note: When setting bit TXOV, bit ENSHDW will be cleared by internal hardware. To ensure bit ENSHDW is set, user must ensure bit TXOV is cleared.* |

**T2CCU_COCON**
**T2CCU Compare Control Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **CCM5** | **CCM4** | **CM5F** | **CM4F** | **POLB** | **POLA** | **COMOD** | |
| rw | rw | rwh | rwh | rw | rw | rw | |

| Field | Bits | Typ | Description |
|---|---|---|---|
| **COMOD** | [1:0] | rw | **Compare Mode Control (for compare channels)**<br>00 Compare Mode 0 selected<br>01 Compare Mode 1 selected<br>10 Concurrent Compare mode selected<br>Others Reserved |
| **POLA** | 2 | rw | **Compare Active Level for Channel Group A**<br>In compare mode 0, the bit defines the active level of the outputs of the compare channels belonging to the group.<br>0 High on Compare Match<br>1 Low on Compare Match |
| **POLB** | 3 | rw | **Compare Active Level for Channel Group B**<br>In compare mode 0, the bit defines the active level of the outputs of the compare channels belonging to the group.<br>0 High on Compare Match<br>1 Low on Compare Match |

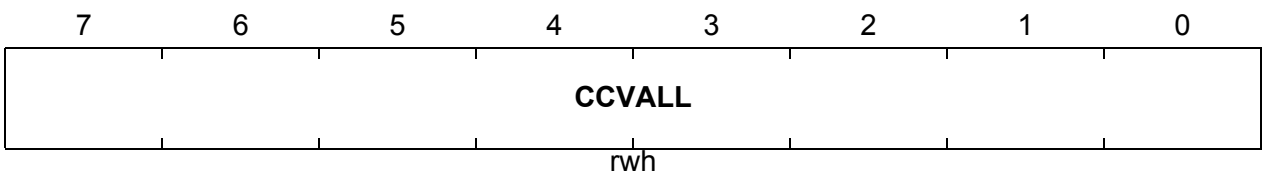
| Field | Bits | Typ | Description |
|---|---|---|---|
| **CM4F**[1)] | 4 | rwh | **Compare Channel 4 Interrupt Flag**<br>Set on a compare match event on compare channel 4.<br>This bit has to be cleared by software. |
| **CM5F**[1)] | 5 | rwh | **Compare Channel 5 Interrupt Flag**<br>Set on a compare match event on compare channel 5.<br>This bit has to be cleared by software. |
| **CCM4** | 6 | rw | **Compare Enable for Channel 4**<br>0 Compare disabled<br>1 Compare enabled<br>See note below. |
| **CCM5** | 7 | rw | **Compare Enable for Channel 5**<br>0 Compare disabled<br>1 Compare enabled<br>See note below. |

1) In case of software and hardware access to a flag bit at the same time, hardware will have higher priority.

*Note: Compare channels 4 and 5 should be enabled in a separate step after setting the compare mode via bits COMOD (and active level POLA/B if compare mode 0). This is to ensure defined compare output according to the selected mode on starting the respective compare channel.*

*Note: The interrupt enable for compare channels 4 and 5 is controlled by the respective bit of SCU SFR MODIEN.CM4EN and CM5EN.*

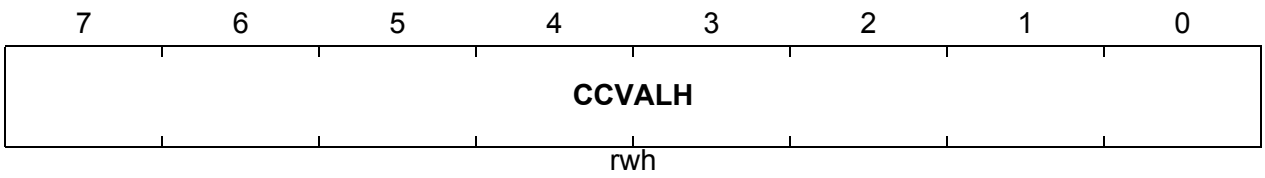
**T2CCU_CCxL**
**T2CCU Capture/Compare Register x Low** **Reset Value: $00_H$**



| Field | Bits | Typ | Description |
|---|---|---|---|
| **CCVALL** | [7:0] | rwh | **Capture/Compare Low Byte Value for Channel x**<br>Capture<br>On read, CCVALL returns the low byte of the last captured value.<br><br>Compare<br>CCVALL holds the low byte written. CCx acts as a shadow register to the channel x internal compare register. The value written is only transferred to the internal compare register when corresponding bit T2CCU_COSHDW.COOUTx is being set (with ENSHDW = 1), or on overflow of the channel timer base.<br>The read value always return the last written value.<br><br>*Note: With T2CCU_CC4 and T2CCU_CC5, only the compare function is available.* |

**T2CCU_CCxH**

**T2CCU Capture/Compare Register x High** **Reset Value: $00_H$**



| Field | Bits | Typ | Description |
|---|---|---|---|
| **CCVALH** | [7:0] | rwh | **Capture/Compare High Byte Value for Channel x**<br>Capture<br>On read, CCVALH returns the high byte of the last captured value.<br><br>Compare<br>CCVALH holds the high byte written. CCx acts as a shadow register to the channel x internal compare register. The value written is only transferred to the internal compare register when corresponding bit T2CCU_COSHDW.COOUTx is being set (with ENSHDW = 1), or on overflow of the channel timer base.<br>The read value always return the last written value.<br><br>*Note: With T2CCU_CC4 and T2CCU_CC5, only the compare function is available.* |

# 15 Capture/Compare Unit 6

The Capture/Compare Unit 6 (CCU6) provides two independent timers (T12, T13), which can be used for Pulse Width Modulation (PWM) generation, especially for AC-motor control. The CCU6 also supports special control modes for block commutation and multi-phase machines. The block diagram of the CCU6 module is shown in **Figure 15-1**.

The timer T12 can function in capture and/or compare mode for its three channels. The timer T13 can work 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.

**Timer T12 Features:**

- Three capture/compare channels, each channel can be used either as a capture or as a compare channel
- Supports generation of a three-phase PWM (six outputs, individual signals for highside and lowside switches)
- 16-bit resolution
- 2 selectable operating frequency: peripheral clock or double the peripheral clock.
- Dead-time control for each channel to avoid short-circuits in the power stage
- Concurrent update of the required T12/13 registers
- Generation of center-aligned and edge-aligned PWM
- Supports single-shot mode
- Supports many interrupt request sources
- Hysteresis-like control mode

**Timer T13 Features:**

- One independent compare channel with one output
- 16-bit resolution, maximum count frequency = 2 x peripheral clock frequency
- Can be synchronized to T12
- Interrupt generation at period-match and compare-match
- Supports single-shot mode

**Additional Features:**

- Implements block commutation for Brushless DC-drives
- Position detection via Hall-sensor pattern
- Automatic rotational speed measurement 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

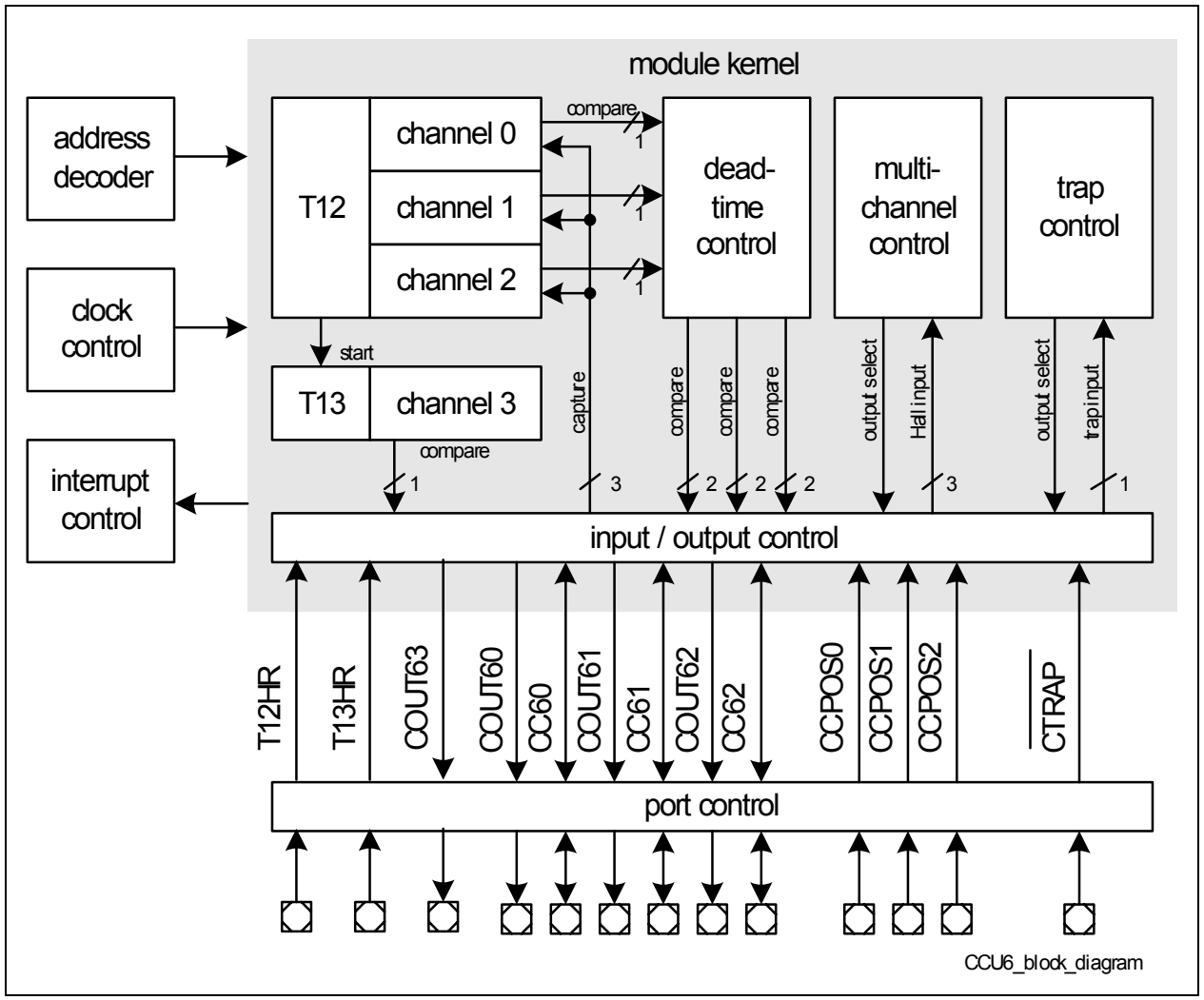



**Figure 15-1 CCU6 Block Diagram**

## 15.1 Functional Description

### 15.1.1 Timer T12

The timer T12 is built with three channels in capture/compare mode. The input clock for timer T12 can be from $f_{CCU6}$ to a maximum of $f_{CCU6}/128$ and is configured by bit field T12CLK. In order to support higher clock frequencies, an additional prescaler factor of 1/256 can be enabled for the prescaler of T12 if bit T12PRE = 1.

The timer period, compare values, passive state selects bits and passive levels bits are written to shadow registers and not directly to the actual registers, while the read access targets the registers actually used (except for the three compare channels, where both the actual and the shadow registers can be read). The transfer from the shadow registers to the actual registers is enabled by setting the shadow transfer enable bit STE12.

If this transfer is enabled, the shadow registers are copied to the respective registers as soon as the associated timer reaches the value zero the next time (being cleared in edge-aligned mode or counting down to 1 in center-aligned mode). When timer T12 is operating in center-aligned mode, it will also copy the registers (if enabled by STE12) if it reaches the currently programmed period value (counting up).

When timer T12 is stopped, the shadow transfer takes place immediately if the corresponding bit STE12 is set. Once the transfer is complete, the respective bit STE12 is cleared automatically.

**Figure 15-2** shows an overview of Timer T12.

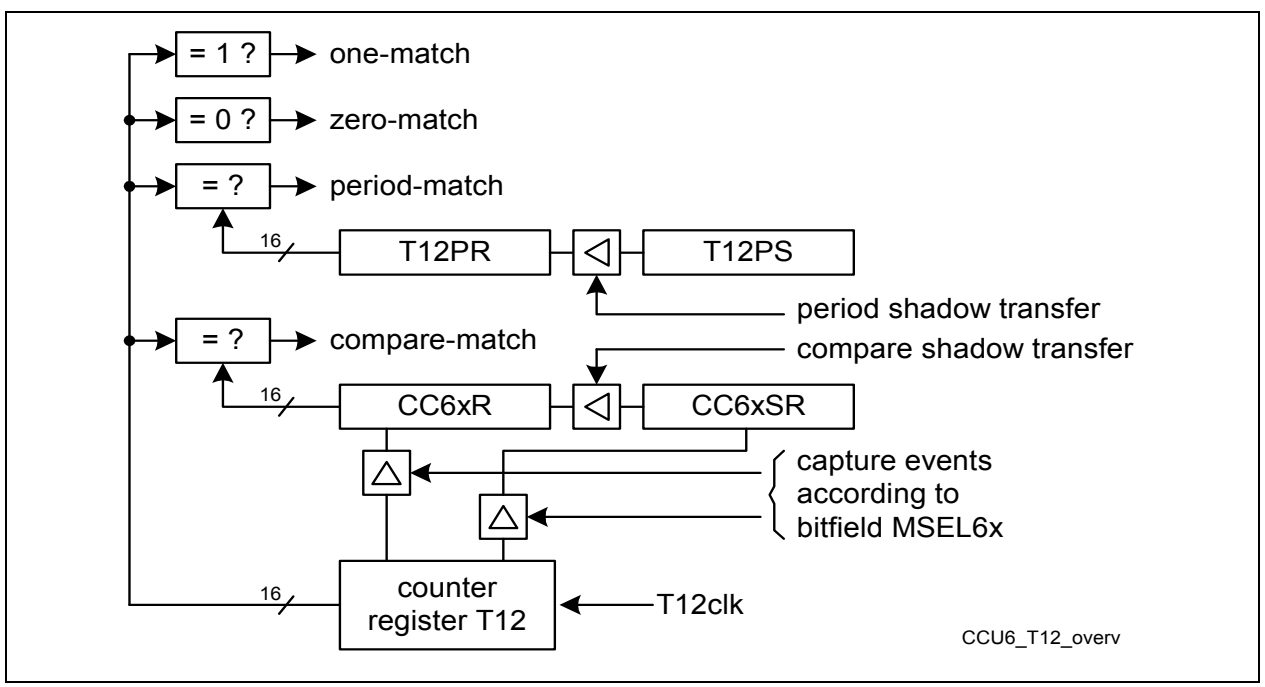



**Figure 15-2 T12 Overview**

#### 15.1.1.1 Timer Configuration

Register T12 represents the counting value of timer T12. It can be written only while timer T12 is stopped. Write actions while T12 is running are not taken into account. Register T12 can always be read by software.

In edge-aligned mode, T12 only counts up, whereas in center-aligned mode, T12 can count up and down.

Timer T12 can be started and stopped by using bit T12R by hardware or software.

- Bit field T12RSEL defines the event on pin T12HR: rising edge, falling edge, or either of these two edges, that can set the run bit T12R by hardware.
- If bit field T12RSEL = $00_B$, the external setting of T12R is disabled and the timer run bit can only be controlled by software. Bit T12R is set/reset by software by setting bit T12RS or T12RR.
- In single-shot mode, bit T12R is reset by hardware according to the function defined by bit T12SSC. If bit T12SSC = 1, the bit T12R is reset by hardware when:
  – T12 reaches its period value in edge-aligned mode
  – T12 reaches the value 1 while counting down in center-aligned mode

Register T12 can be reset to zero by setting bit T12RES. Setting of T12RES has no impact on run bit T12R.

#### 15.1.1.2 Counting Rules

With reference to the T12 input clock, the counting sequence is defined by the following counting rules:

**T12 in edge-aligned mode (Bit CTM = 0)**

The count direction is set to counting up (CDIR = 0). The counter is reset to zero if a period-match is detected, and the T12 shadow register transfer takes place if STE12 = 1.

**T12 in center-aligned mode (Bit CTM = 1)**

- The count direction is set to counting up (CDIR = 0) if a one-match is detected while counting down.
- The count direction is set to counting down (CDIR = 1) if a period-match is detected while counting up.
- If STE12 = 1, shadow transfer takes place when:
  – a period-match is detected while counting up
  – a one-match is detected while counting down

The timer T12 prescaler is reset when T12 is not running to ensure reproducible timings and delays.

### 15.1.1.3 Switching Rules

Compare actions take place in parallel for the three compare channels. Depending on the count direction, the compare matches have different meanings. In order to get the PWM information independent of the output levels, two different states have been introduced for the compare actions: the active state and the passive state. Both these states are used to generate the desired PWM as a combination of the control by T13, the trap control unit and the multi-channel control unit. If the active state is interpreted as a 1 and the passive state as a 0, the state information is combined with a logical AND function.

- active AND active = active
- active AND passive = passive
- passive AND passive = passive

The compare states change with the detected compare-matches and are indicated by the CC6xST bits. The compare states of T12 are defined as follows:

- passive if the counter value is below the compare value
- active if the counter value is above the compare value

This leads to the following switching rules for the compare states:

- set to the active state when the counter value reaches the compare value while counting up
- reset to the passive state when the counter value reaches the compare value while counting down
- reset to the passive state in case of a zero-match without compare-match while counting up
- set to the active state in case of a zero-match with a parallel compare-match while counting up

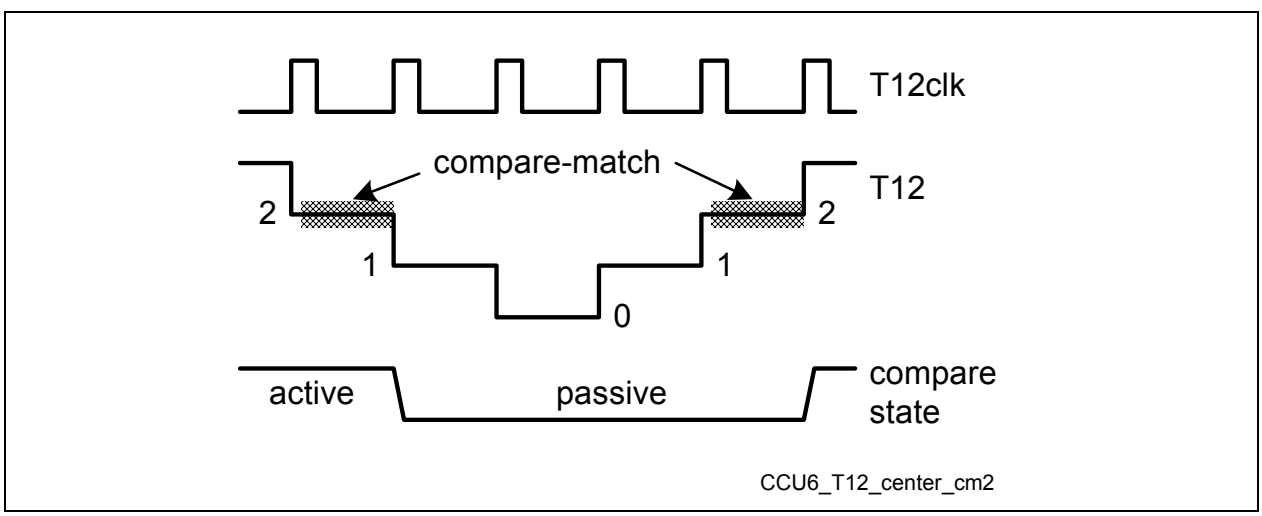



**Figure 15-3 Compared States for Compare Value = 2**

The switching rules are considered only while the timer is running. As a result, write actions to the timer registers while the timer is stopped do not lead to compare actions.

#### 15.1.1.4 Compare Mode of T12

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. The register CC6xR can only be read by software and the modification of the value is done by a shadow register transfer from register CC6xSR.

Register T12PR contains the period value for timer T12. The period value is compared to the actual counter value of T12 and the resulting counter actions depend on the defined counting rules.

**Figure 15-4** shows an example in the center-aligned mode without dead-time. The bit CC6xST indicates the occurrence of a capture or compare event of the corresponding channel. It can be set (if it is 0) by the following events:

- a software set (MCC6xS)
- a compare set event (T12 counter value above the compare value) if the T12 runs and if the T12 set event is enabled
- upon a capture set event

The bit CC6xST can be reset (if it is 1) by the following events:

- a software reset (MCC6xR)
- a compare reset event (T12 counter value below the compare value) if the T12 runs and if the T12 reset event is enabled (including in single-shot mode at the end of the T12 period)
- a reset event in the hysteresis-like control mode

The bit CC6xPS represents passive state select bit. The timer T12's two output lines (CC6x, COUT6x) can be selected to be in the passive state while CC6xST is 0 (with CC6xPS = 0) or while CC6xST is 1 (with CC6xPS = 1).

The output level that is driven while the output is in the passive state is defined by the corresponding bit in bit field PSL.

**Figure 15-5** shows the settings of CC6xPS/COUT6xPS and PSL for different applications. The examples are in the center-aligned mode with dead-time.

Hardware modifications of the compare state bits are only possible while timer T12 is running. Therefore, the bit T12R can be used to enable/disable the modification by hardware.

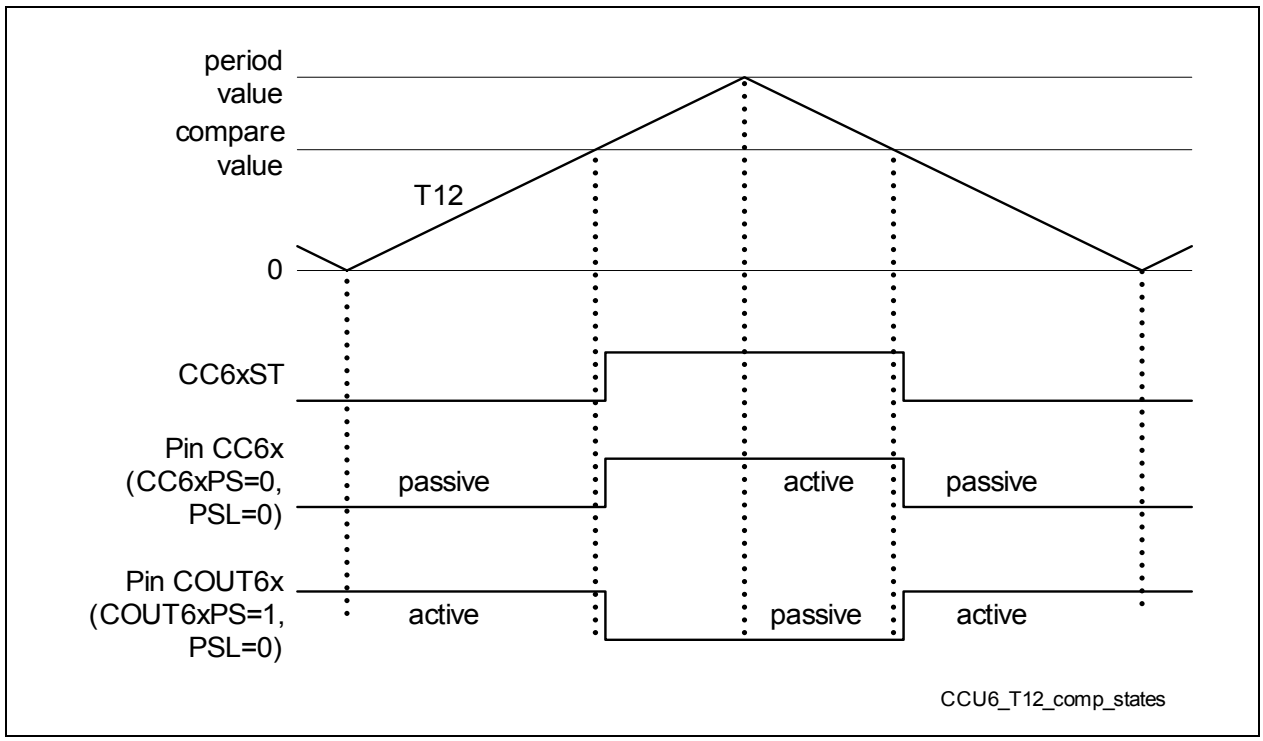



**Figure 15-4 Compare States of Timer T12**

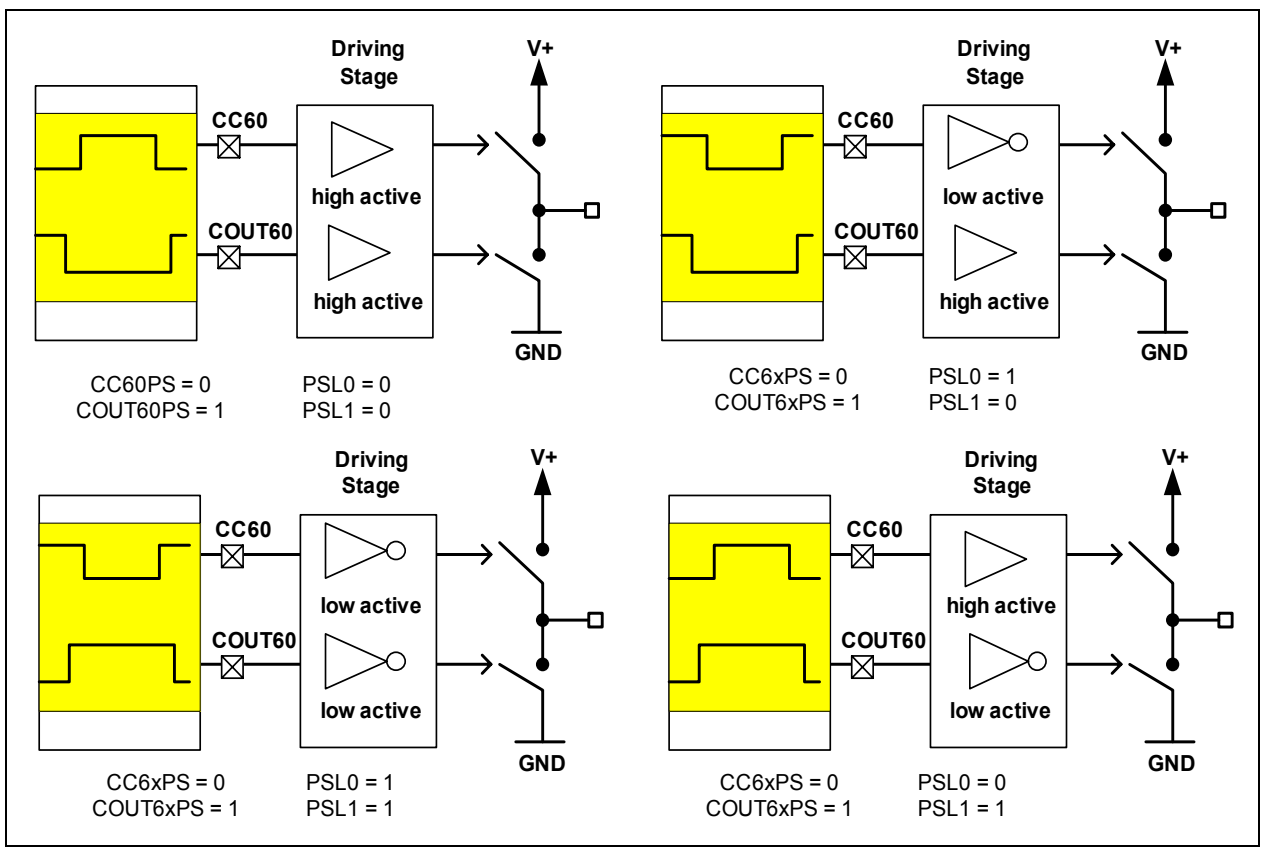



**Figure 15-5 Different settings of CC6xPS/COUT6xPS and PSL**

For the hysteresis-like compare mode (MSEL6x = $1001_B$) (see **Section 15.1.1.9**), the setting of the compare state bit is possible only while the corresponding input CCPOSx = 1 (inactive).

If the hall sensor mode (MSEL6x = $1000_B$) is selected (see **Section 15.1.6**), the compare state bits of the compare channels 1 and 2 are modified by the timer T12 in order to indicate that a programmed time interval has elapsed.

The set is only generated when bit CC6xST is reset; a reset can only take place when the bit is set. Thus, the events triggering the set and reset actions of the CC6xST bit must be combined. This OR-combination of the resulting set and reset permits the reload of the dead-time counter to be triggered (see **Figure 15-6**). This is triggered only if bit CC6xST is changed, permitting a correct PWM generation with dead-time and the complete duty cycle range of 0% to 100% in edge-aligned and center-aligned modes.

### 15.1.1.5 Duty Cycle of 0% and 100%

These counting and switching rules ensure a PWM functionality in the full range between 0% and 100% duty cycle (duty cycle = active time/total PWM period). In order to obtain a duty cycle of 0% (compare state never active), a compare value of T12P+1 must be programmed (for both compare modes). A compare value of 0 will lead to a duty cycle of 100% (compare state always active).

### 15.1.1.6 Dead-time Generation

In most cases, the switching behavior of the connected power switches is not symmetrical with respect to the times needed to switch on and to switch off. A general problem arises if the time taken to switch on is less than the time to switch off the power device. This leads to a short-circuit in the inverter bridge leg, which may damage the entire system. In order to solve this problem by hardware, the CCU6 contains a programmable dead-time counter, which delays the passive to active edge of the switching signals (the active to passive edge is not delayed).

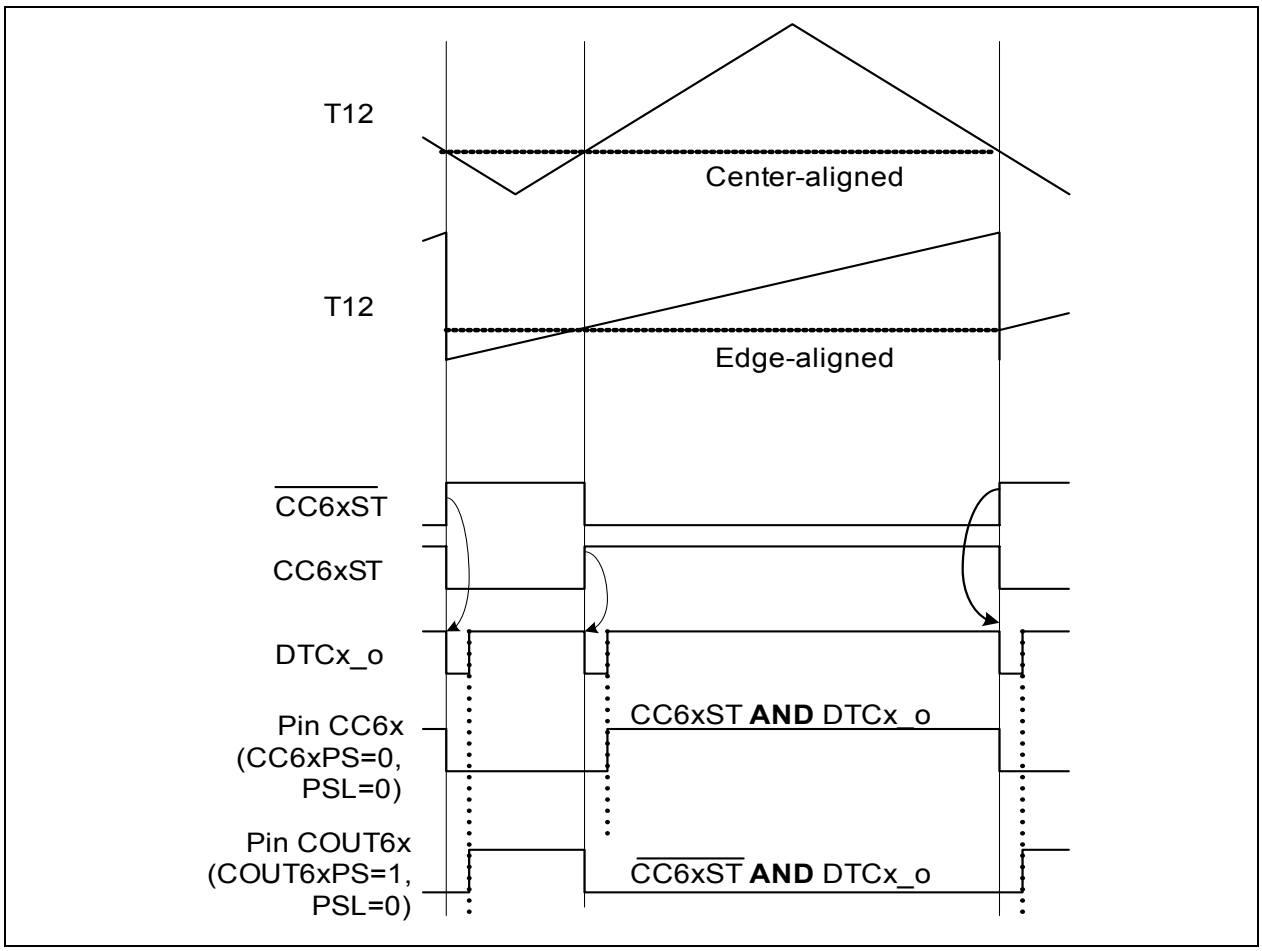



**Figure 15-6 PWM-signals with Dead-time Generation**

Register T12DTC controls the dead-time generation for the timer T12 compare channels. Each channel can be independently enabled/disabled for dead-time generation by bit DTEx. If enabled, the transition from passive state to active state is delayed by the value defined by bit field DTM (8-bit down counter, clocked with T12CLK). The dead-time counter can only be reloaded when it is zero.

Each of the three channels works independently with its own dead-time counter, trigger and enable signals. The value of bit field DTM is valid for all three channels.

### 15.1.1.7 Capture Mode

In capture mode, the bits CC6xST indicate the occurrence of the selected capture event according to the bit fields MSEL6x.

- MSEL6x = $01XX_B$, double register capture mode (see **Table 15-6**)
- MSEL6x = $101X_B$ or $11XX_B$, multi-input capture modes (see **Table 15-8**)

A rising and/or a falling edge on the pins CC6x or CCPOSx can be selected as the capture event that is used to transfer the contents of timer T12 to the CC6xR and

CC6xSR registers. In order to work in capture mode, the capture pins must be configured as inputs.

There are several ways to store the captured values in the registers. For example, in double register capture mode, the timer value is stored in the channel shadow register CC6xSR. The value previously stored in this register is simultaneously copied to the channel register CC6xR. The software can then check the newly captured value while still preserving the possibility of reading the value captured earlier.

*Note: In capture mode, a shadow transfer can be requested according to the shadow transfer rules, except for the capture/compare registers that are left unchanged.*

### 15.1.1.8 Single-Shot Mode

The single-shot mode of timer T12 is selected when bit T12SSC is set to 1. In single-shot mode, the timer T12 stops automatically at the end of its counting period. **Figure 15-7** shows the functionality at the end of the timer period in edge-aligned and center-aligned modes. If the end of period event is detected while bit T12SSC is set, the bit T12R and all CC6xST bits are reset.

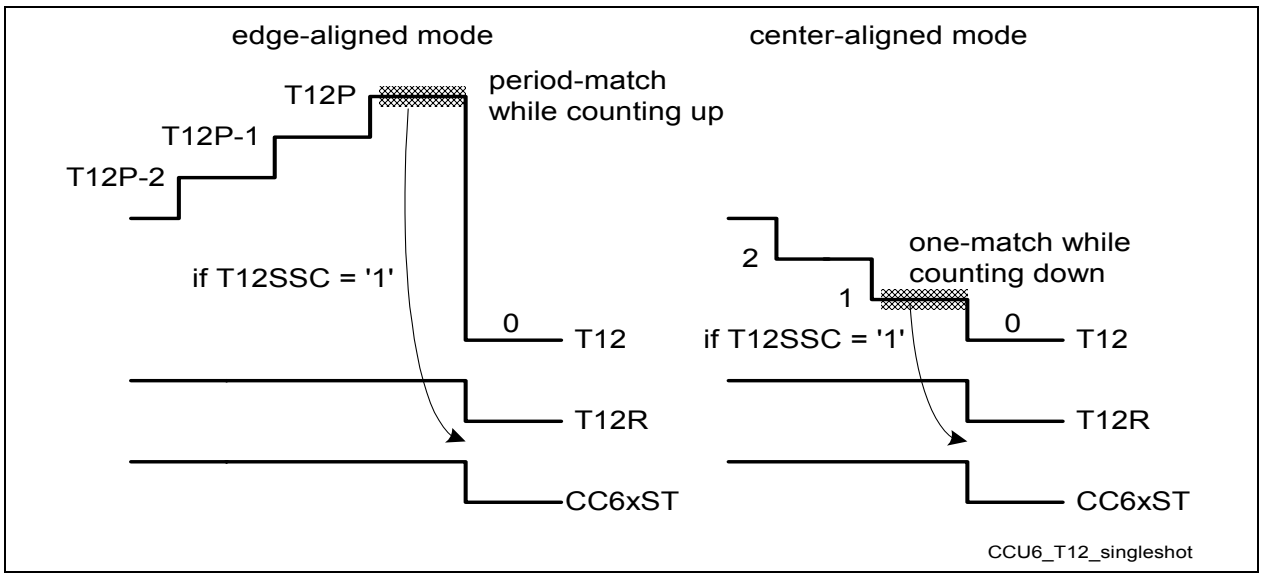



**Figure 15-7 End of Single-Shot Mode of T12**

### 15.1.1.9 Hysteresis-Like Control Mode

The hysteresis-like control mode (MSEL6x = $1001_B$) offers the possibility of switching off the PWM output, if the input CCPOSx becomes 0, by resetting bit CC6xST. This can be used as a simple motor control feature by using a comparator to indicate, for example, over-current. While CCPOSx = 0, the PWM outputs of the corresponding channel are driving their passive levels. The setting of bit CC6xST is only possible while CCPOSx = 1. **Figure 15-8** shows an example of hysteresis-like control mode.

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 may change constantly.

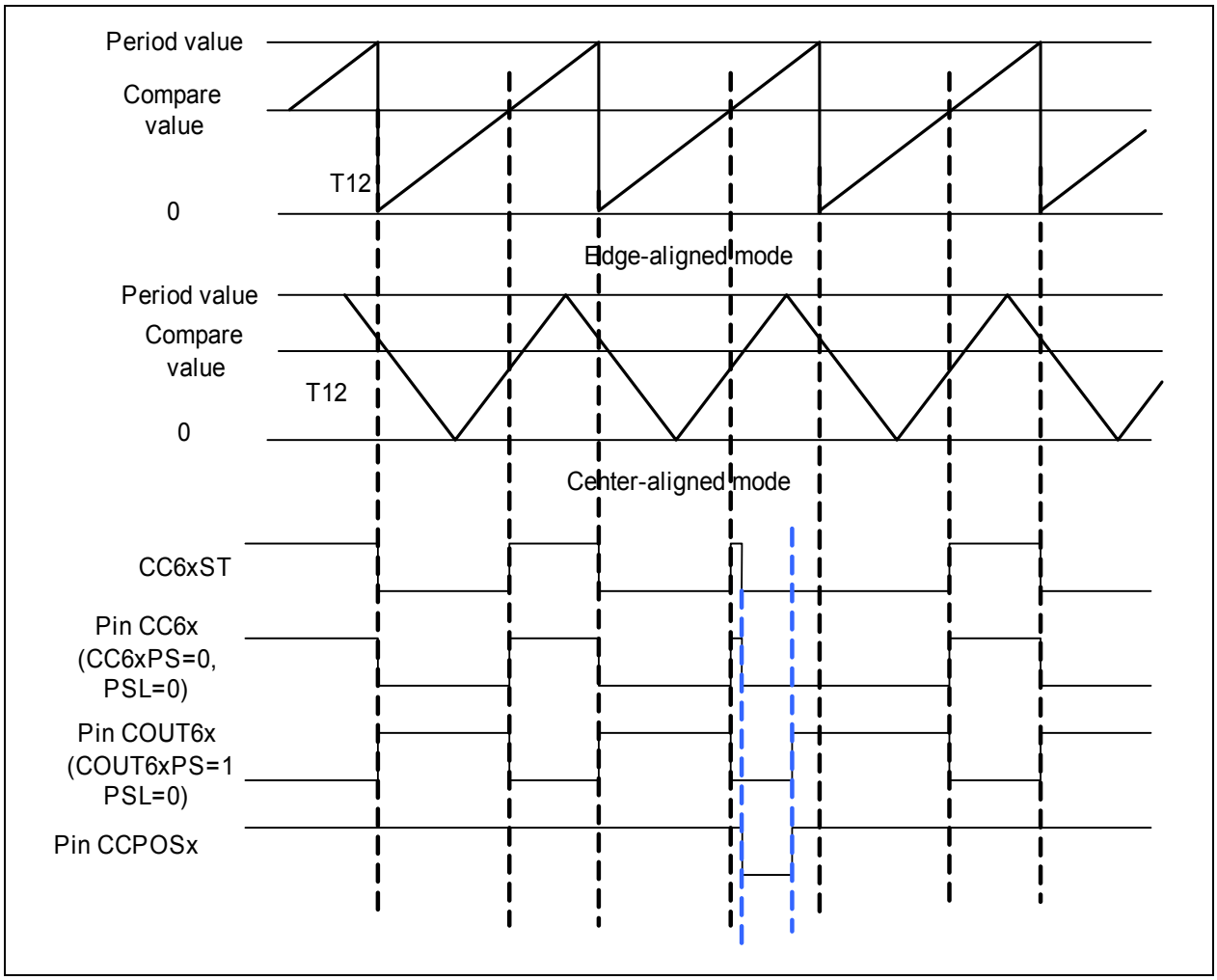



**Figure 15-8 Hysteresis-Like Control Mode**

### 15.1.2 Timer T13

The timer T13 is similar to timer T12, except that it has only one channel in compare mode. The counter can only count up (similar to the edge-aligned mode of T12). The input clock for timer T13 can be from $f_{CCU6}$ to a maximum of $f_{CCU6}/128$ and is configured by bit field T13CLK. In order to support higher clock frequencies, an additional prescaler factor of 1/256 can be enabled for the prescaler of T13 if bit T13PRE = 1.

The T13 shadow transfer, in case of a period-match, is enabled by bit STE13. During the T13 shadow transfer, the contents of register CC63SR are transferred to register CC63R. Both registers can be read by software, while only the shadow register can be written by software.

The bits CC63PS, T13IM and PSL63 have shadow bits. The contents of these shadow bits are transferred to the actually used bits during the T13 shadow transfer. Write actions target the shadow bits, while read actions deliver the value of the actually used bits.

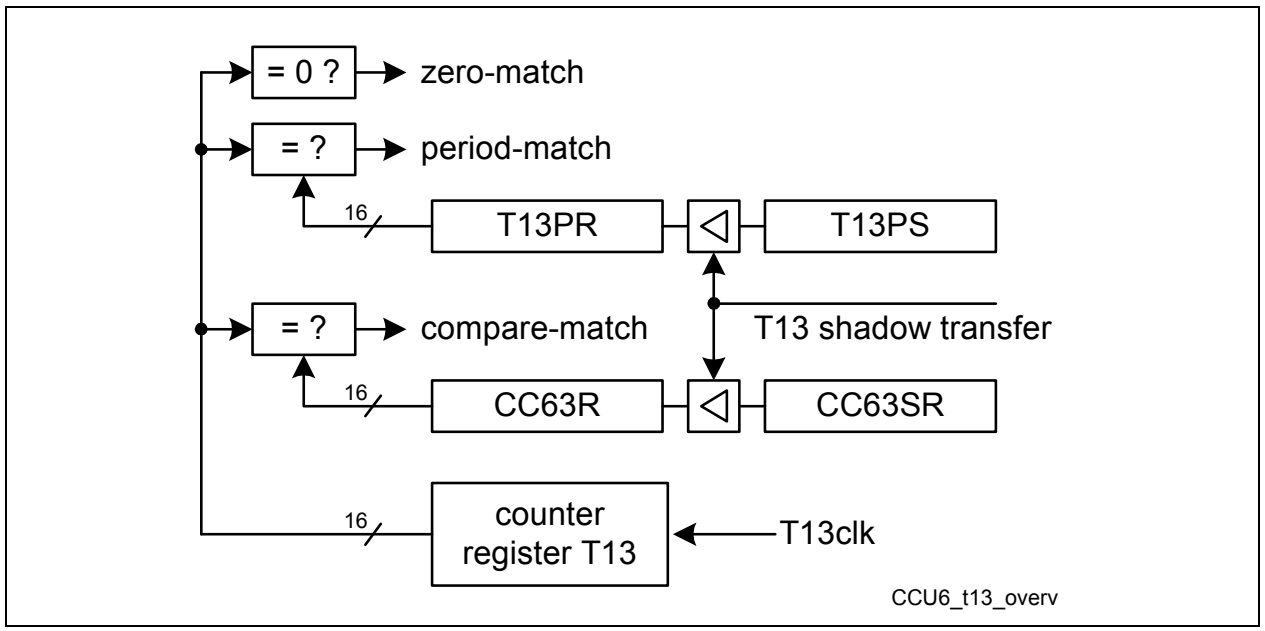



**Figure 15-9 T13 Overview**

Timer T13 counts according to the same counting and switching rules as timer T12 in edge-aligned mode. **Figure 15-9** shows an overview of Timer T13.

#### 15.1.2.1 Timer Configuration

Register T13 represents the counting value of timer T13. It can be written only while the timer T13 is stopped. Write actions are not taken into account while T13 is running. Register T13 can always be read by software. Timer T13 supports only edge-aligned mode (counting up).

Timer T13 can be started and stopped by using bit T13R by hardware or software.

- Bit T13R is set/reset by software by setting bit T13RS or T13RR.
- In single-shot mode, if bit T13SSC = 1, the bit T13R is reset by hardware when T13 reaches its period value.
- Bit fields T13TEC and T13TED select the trigger event that will set bit T13R for synchronization of different T12 compare events.

The T13 counter register can be reset to zero by setting bit T13RES. Setting of T13RES has no impact on bit T13R.

### 15.1.2.2 Compare Mode

Register CC63R is the actual compare register for T13. The value stored in CC63R is compared to the counter value of T13. The register CC63R can only be read by software and 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 software.

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.

The bit CC63ST indicates the occurrence of a compare event of the corresponding channel. It can be set (if it is 0) by the following events:

- a software set (MCC63S)
- a compare set event (T13 counter value above the compare value) if the T13 runs and if the T13 set event is enabled

The bit CC63ST can be reset (if it is 1) by the following events:

- a software reset (MCC63R)
- a compare reset event (T13 counter value below the compare value) if the T13 runs and if the T13 reset event is enabled (including in single-shot mode at the end of the T13 period)

Timer T13 is used to modulate the other output signals with a T13 PWM. In order to decouple COUT63 from the internal modulation, the compare state can be selected independently by bits T13IM and COUT63PS.

### 15.1.2.3 Single-Shot Mode

The single-shot mode of timer T13 is selected when bit T13SSC is set to 1. In single-shot mode, the timer T13 stops automatically at the end of its counting period. If the end of period event is detected while bit T13SSC is set, the bit T13R and the bit CC63ST are reset.

### 15.1.2.4 Synchronization of T13 to T12

The timer T13 can be synchronized on a T12 event. The events include:

- a T12 compare event on channel 0
- a T12 compare event on channel 1
- a T12 compare event on channel 2
- any T12 compare event on channel 0, 1, or 2
- a period-match of T12
- a zero-match of T12 (while counting up)
- any edge of inputs CCPOSx

The bit fields T13TEC and T13TED select the event that is used to start timer T13. This event sets bit T13R by hardware and T13 starts counting. Combined with the single-shot mode, this can be used to generate a programmable delay after a T12 event.

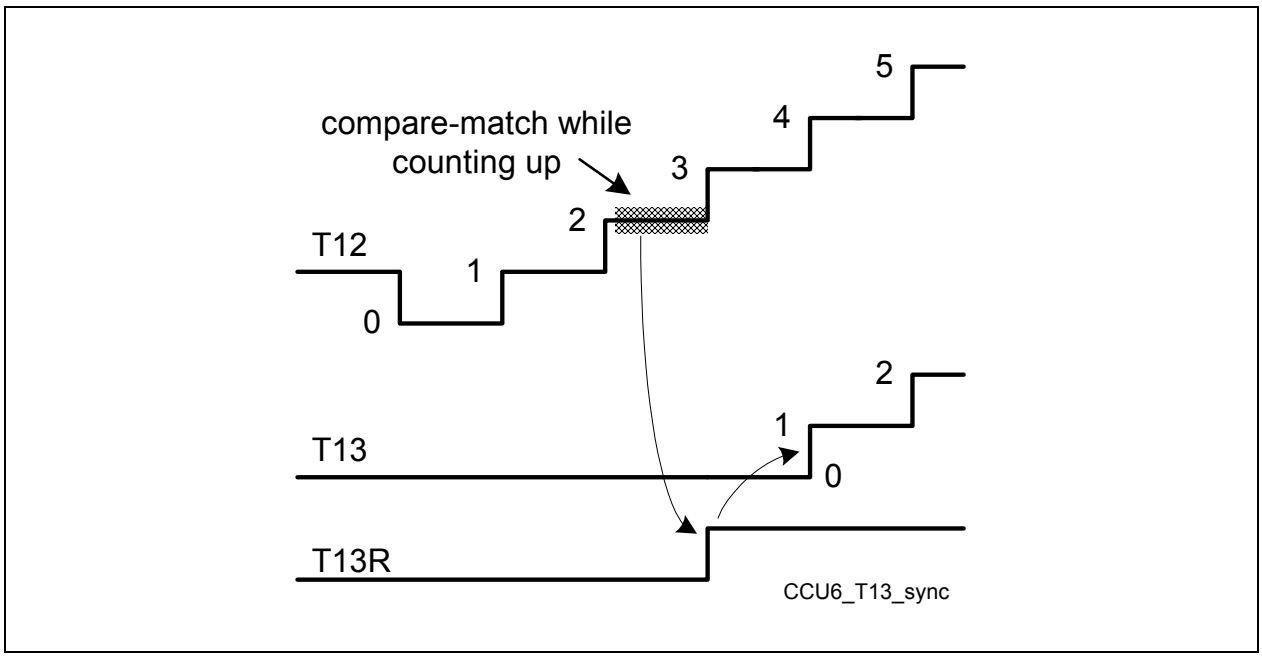



**Figure 15-10 Synchronization of T13 to T12**

**Figure 15-10** shows the synchronization of T13 to a T12 event. The selected event in this example is a compare-match (compare value = 2) while counting up. The clocks of T12 and T13 can be different (use other prescaler factor), but in this example T12CLK is shown as equal to T13CLK for the sake of simplicity.

### 15.1.3 Modulation Control

The modulation control part combines the different modulation sources (CC6x_T12_o and COUT6x_T12_o are the output signals that are configured with CC6xPS/COUT6xPS; MOD_T13_o is the output signal after T13 Inverted Modulation (T13IM)). Each modulation source can be individually enabled per output line. Furthermore, the trap functionality is taken into account to disable the modulation of the corresponding output line during the trap state (if enabled).

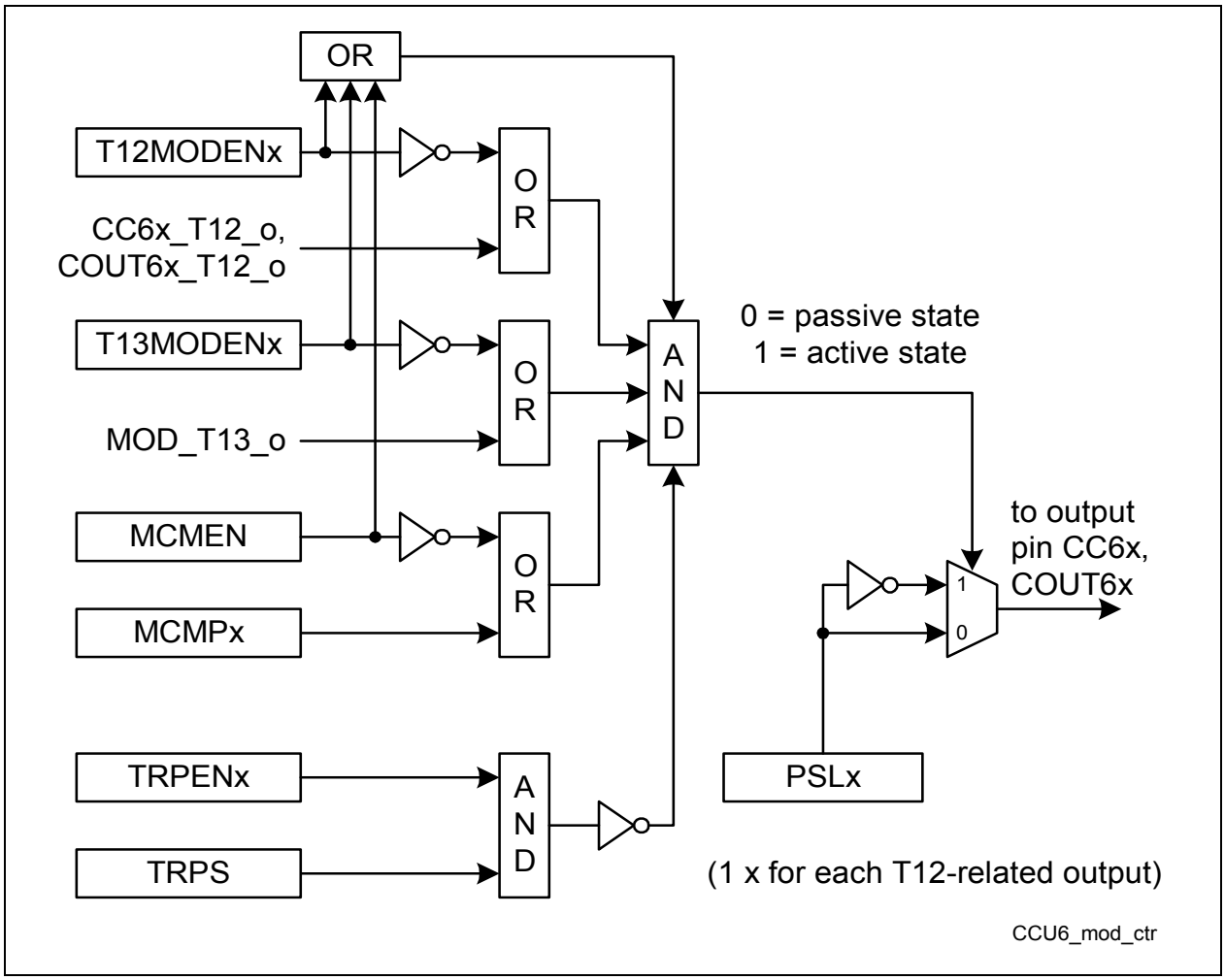



**Figure 15-11 Modulation Control of T12-related Outputs**

For each of the six T12-related output lines (represented by “x”) in the **Figure 15-11**:

- T12MODENx enables the modulation by a PWM pattern generated by timer T12
- T13MODENx enables the modulation by a PWM pattern generated by timer T13
- MCMPx chooses the multi-channel patterns
- TRPENx enables the trap functionality
- PSLx defines the output level that is driven while the output is in the passive state

As shown in **Figure 15-12**, the modulation control part for the T13-related output COUT63 combines the T13 output signal (COUT63_T13_o is the output signal that is configured by COUT63PS) and the enable bit ECT13O with the trap functionality. The output level of the passive state is selected by bit PSL63.

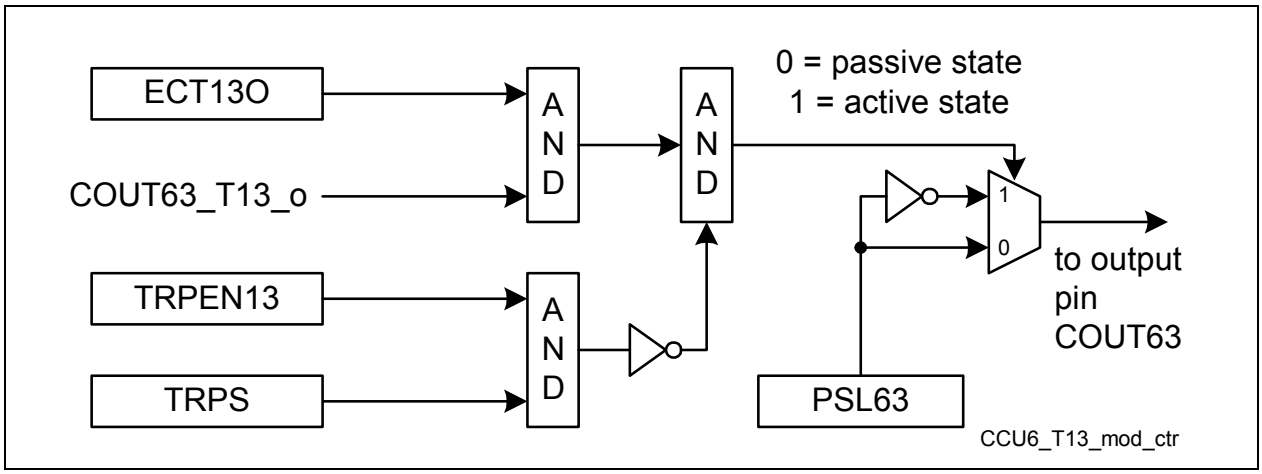



**Figure 15-12 Modulation Control of the T13-related Output COUT63**

Figure 15-13 shows a modulation control example for CC60 and COUT60.

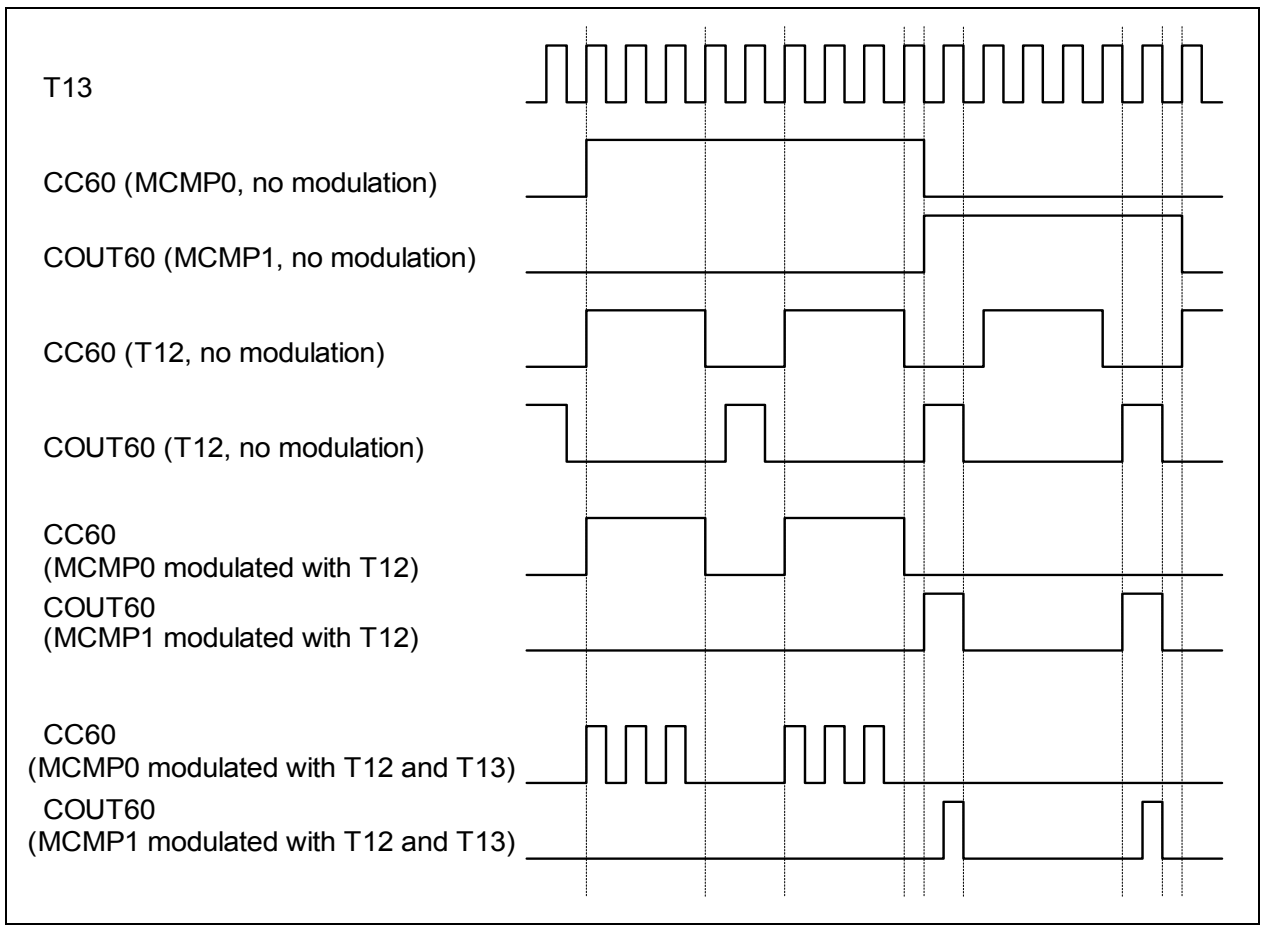



**Figure 15-13 Modulation Control Example for CC60 and COUT60**

### 15.1.4 Trap Handling

The trap functionality permits the PWM outputs to react to the state of the input pin CTRAP. This functionality can be used to switch off the power devices if the trap input becomes active (e.g., as emergency stop).

During the trap state, the selected outputs are forced into the passive state and no active modulation is possible. The trap state is entered immediately by hardware if the CTRAP input signal becomes active and the trap function is enabled by bit TRPPEN. It can also be entered by software by setting bit TRPF (trap input flag), thus leading to TRPS = 1 (trap state indication flag). The trap state can be left when the input is inactive by software control and synchronized to the following events:

- TRPF is automatically reset after CTRAP becomes inactive (if TRPM2 = 0)
- TRPF must be reset by software after CTRAP becomes inactive (if TRPM2 = 1)
- synchronized to T12 PWM after TRPF is reset
  (T12 period-match in edge-aligned mode or one-match while counting down in center-aligned mode)

- synchronized to T13 PWM after TRPF is reset (T13 period-match)
- no synchronization to T12 or T13

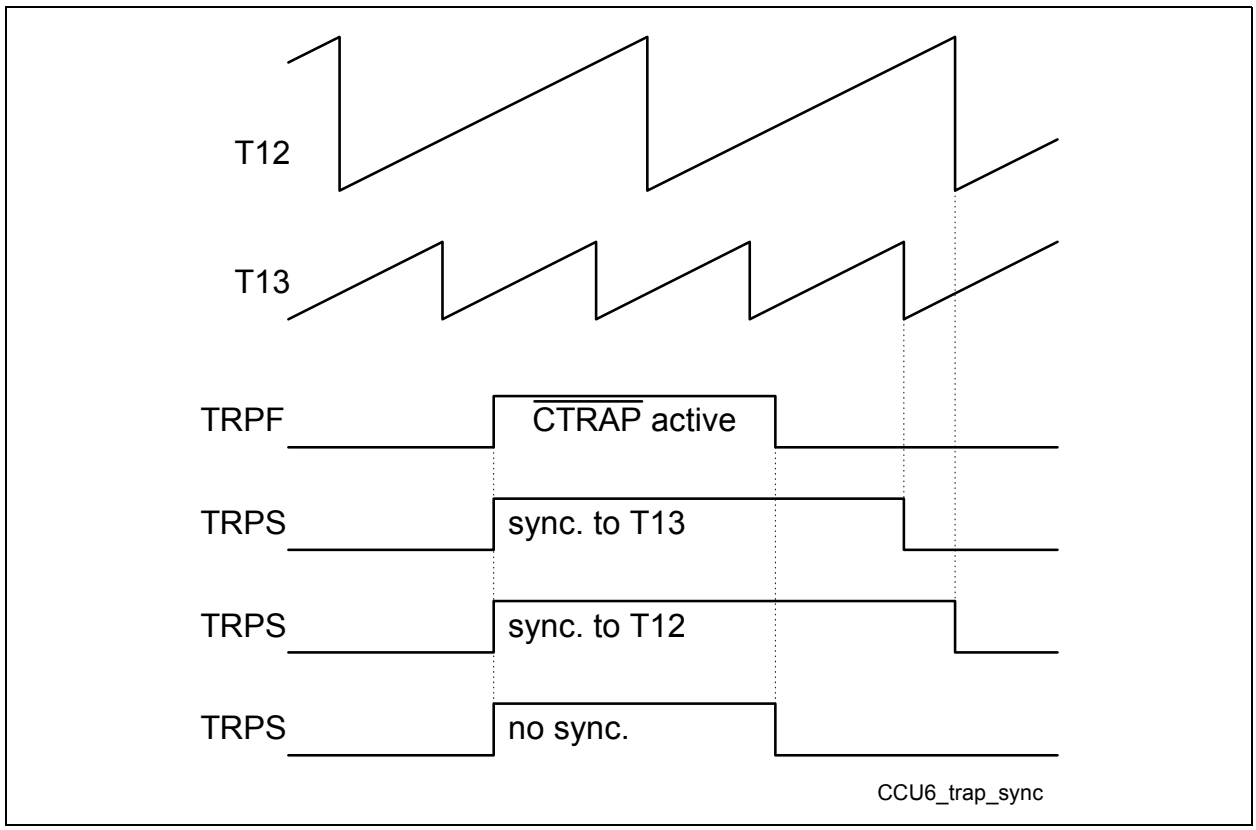



**Figure 15-14 Trap State Synchronization (with TRPM2 = 0)**

### 15.1.5 Multi-Channel Mode

The multi-channel mode offers the possibility of modulating all six T12-related outputs. The bits in bit field MCMP are used to select the outputs that may become active. If the multi-channel mode is enabled (bit MCMEN = 1), only those outputs that have a 1 at the corresponding bit positions in bit field MCMP may become active.

This bit field has its own shadow bit field MCMPS, which 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, which is then taken into account by the hardware at a well-defined moment and synchronized to a PWM period. This avoids unintended pulses due to unsynchronized modulation sources (T12, T13, SW).

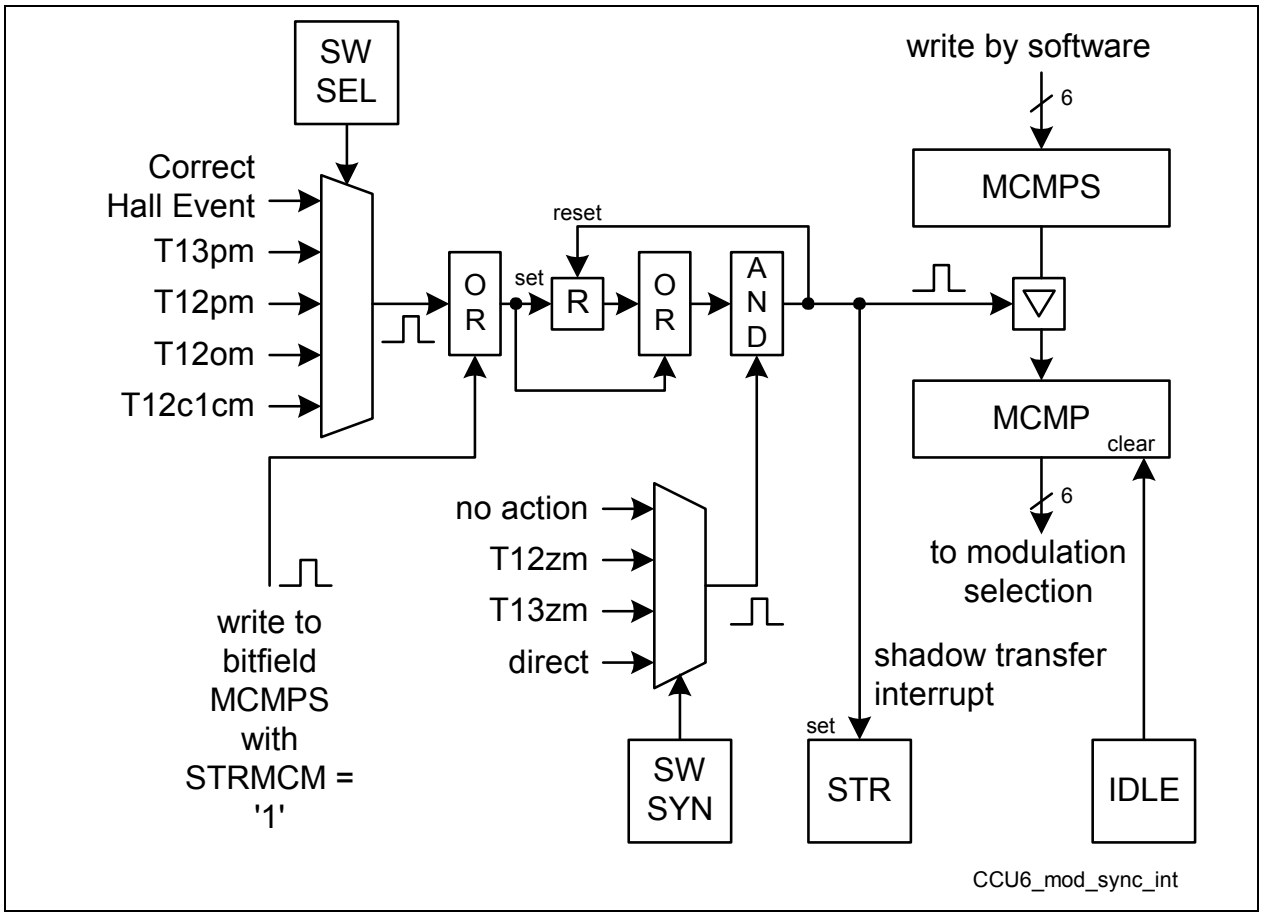



**Figure 15-15 Modulation Selection and Synchronization**

Figure 15-15 shows the modulation selection for the multi-channel mode. The event that triggers the update of bit field MCMP is chosen by SWSEL. If the selected switching event occurs, the reminder flag R is set. This flag monitors the update request and it is automatically reset when the update takes place. 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, which leads to the transfer from MCMPS to MCMP. Due to this structure, an update takes place with a new PWM period.

The update can also be requested by software by writing to bit field MCMPS with the shadow transfer request bit STRMCM set. If this bit is set during the write action to the register, the flag R is automatically set. By using this, the update takes place completely under software control.

A shadow transfer interrupt can be generated when the shadow transfer takes place. The possible hardware request events are:

- a T12 period-match while counting up (T12pm)
- a T12 one-match while counting down (T12om)
- a T13 period-match (T13pm)
- a T12 compare-match of channel 1 (T12c1cm)
- a correct Hall event

The possible hardware synchronization events are:

- a T12 zero-match while counting up (T12zm)
- a T13 zero-match (T13zm)

### 15.1.6 Hall Sensor Mode

In **Brushless-DC** motors, the next multi-channel state values depend on the pattern of the Hall inputs. There is a strong correlation between the **Hall pattern** (CURH) and the **modulation pattern** (MCMP). Because of different machine types, the modulation pattern for driving the motor can vary. Therefore, it is beneficial to have wide flexibility in defining the correlation between the Hall pattern and the corresponding modulation pattern. The CCU6 offers this by having a register which contains the actual Hall pattern (CURHS), the next expected Hall pattern (EXPHS), and its output pattern (MCMPS). At every correct Hall event, a new Hall pattern with its corresponding output pattern can be loaded (from a predefined table) by software into the register MCMOUTS. This shadow register can also be loaded by a write action on MCMOUTS with bit STRHP = 1. In case of a phase delay (generated by T12 channel 1), a new pattern can be loaded when the multi-channel mode shadow transfer (indicated by bit STR) occurs.

#### 15.1.6.1 Sampling of the Hall Pattern

The Hall pattern (on CCPOSx) is sampled with the module clock $f_{CCU6}$. By using the dead-time counter DTC0 (mode MSEL6x = $1000_B$), a hardware **noise filter** can be implemented to suppress spikes on the Hall inputs. In case of a Hall event, the DTC0 is reloaded, and it starts counting and generates a delay between the detected event and the sampling point. After the counter value of 1 is reached, the CCPOSx inputs are sampled (without noise and spikes) and are compared to the current Hall pattern (CURH) and to the expected Hall pattern (EXPH). If the sampled pattern equals to the current pattern, it means that the edge on CCPOSx was due to a noise spike and no action will be triggered (implicit noise filter by delay). If the sampled pattern equals to the next expected pattern, the edge on CCPOSx was a correct Hall event, and the bit CHE is set which causes an interrupt.

If it is required that the multi-channel mode and the Hall pattern comparison work independently of timer T12, the delay generation by DTC0 can be bypassed. In this case, timer T12 can be used for other purposes.

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. The hall compare action can also be triggered by software by writing a 1 to bit SWHC. The triggering sources for the sampling by hardware include:

- Any edge at one of the inputs CCPOSx (x = 0 - 2)
- A T13 compare-match
- A T13 period-match
- A T12 period-match (while counting up)
- A T12 one-match (while counting down)
- A T12 compare-match of channel 0 (while counting up)
- A T12 compare-match of channel 0 (while counting down)

This correct Hall event can be used as a transfer request event for register MCMOUTS. The transfer from MCMOUTS to MCMOUT transfers the new CURH-pattern as well as the next EXPH-pattern. In case the sampled Hall inputs were neither the current nor the expected Hall pattern, the bit WHE (wrong Hall event) is set, which can also cause an interrupt and set the IDLE mode to clear MCMP (modulation outputs are inactive). To restart from IDLE, the transfer request of MCMOUTS must be initiated by software (bit STRHP and bit fields SWSEL/SWSYN).

### 15.1.6.2 Brushless-DC Control

For **Brushless-DC** motors, there is a special mode (MSEL6x = $1000_B$) which is triggered by a change of the Hall inputs (CCPOSx). In this case, T12's channel 0 acts in capture function, channel 1 and 2 act in compare function (without output modulation), and the multi-channel-block is used to trigger the output switching together with a possible modulation of T13.

After the detection of a valid Hall edge, the T12 count value is captured to channel 0 (representing the actual motor speed) and the T12 is reset. When the timer reaches the compare value in channel 1, the next multi-channel state is switched by triggering the shadow transfer of bit field MCMP. This trigger event can be combined with several conditions which are necessary to implement noise filtering (correct Hall event) and to synchronize the next multi-channel state to the modulation sources (avoiding spikes on the output lines). This compare function of channel 1 can be used as a phase delay for the position input to the output switching which is necessary if a sensorless back-EMF technique is used instead of Hall sensors. The compare value in channel 2 can be used as a time-out trigger (interrupt) indicating that the motor's destination speed is far below the desired value (which can be caused by an abnormal load change). In this mode, the modulation of T12 must be disabled (T12MODENx = 0).

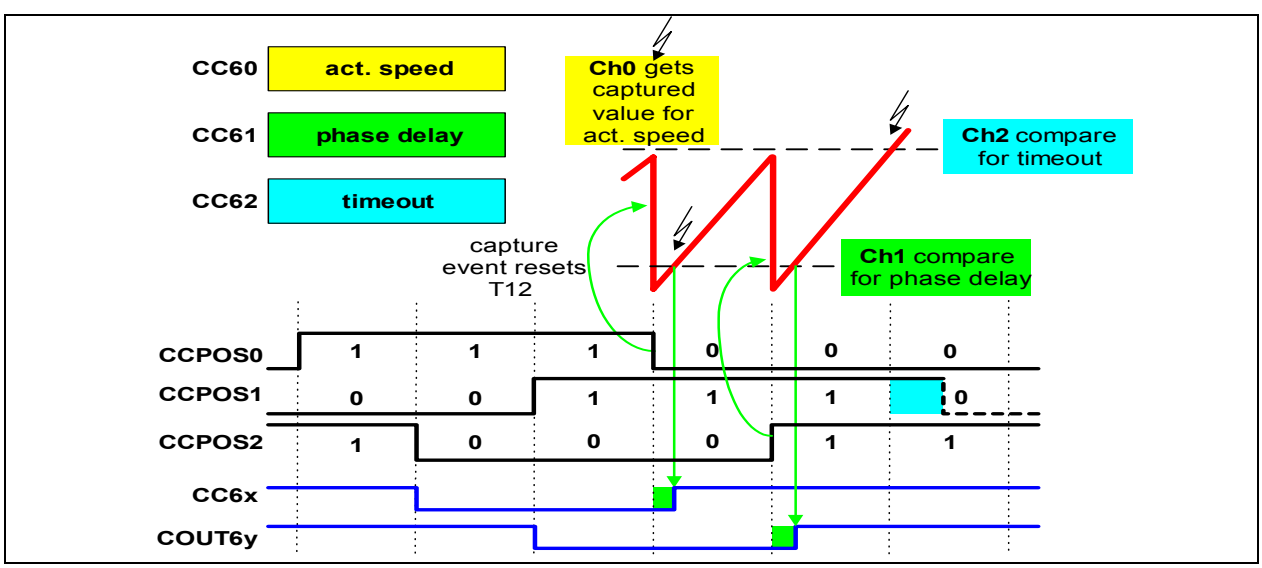



**Figure 15-16 Timer T12 Brushless-DC Mode (all MSEL6x = $1000_B$)**

Table 15-1 lists an example of block commutation in BLDC motor control. If the input signal combination CCPOS0-CCPOS2 changes its state, the outputs CC6x and COUT6x are set to their new states.

Figure 15-17 shows the block commutation in rotate left mode and Figure 15-18 shows the block commutation in rotate right mode. These figures are derived directly from Table 15-1.

**Table 15-1 Block Commutation Control Table**

| Mode | CCPOS0-CCPOS2 Inputs | | | CC60 - CC62 Outputs | | | COUT60 - COUT62 Outputs | | |
|---|---|---|---|---|---|---|---|---|---|
| | CCPOS0 | CCPOS1 | CCPOS2 | CC60 | CC61 | CC62 | COUT60 | COUT61 | COUT62 |
| Rotate left, 0° phase shift | 1 | 0 | 1 | inactive | inactive | active | inactive | active | inactive |
| | 1 | 0 | 0 | inactive | inactive | active | active | inactive | inactive |
| | 1 | 1 | 0 | inactive | active | inactive | active | inactive | inactive |
| | 0 | 1 | 0 | inactive | active | inactive | inactive | inactive | active |
| | 0 | 1 | 1 | active | inactive | inactive | inactive | inactive | active |
| | 0 | 0 | 1 | active | inactive | inactive | inactive | active | inactive |
| Rotate right | 1 | 1 | 0 | active | inactive | inactive | inactive | active | inactive |
| | 1 | 0 | 0 | active | inactive | inactive | inactive | inactive | active |
| | 1 | 0 | 1 | inactive | active | inactive | inactive | inactive | active |
| | 0 | 0 | 1 | inactive | active | inactive | active | inactive | inactive |
| | 0 | 1 | 1 | inactive | inactive | active | active | inactive | inactive |
| | 0 | 1 | 0 | inactive | inactive | active | inactive | active | inactive |
| Slow-down | X | X | X | inactive | inactive | inactive | active | active | active |
| Idle[1)] | X | X | X | inactive | inactive | inactive | inactive | inactive | inactive |

[1)] In case the sampled Hall inputs were neither the current nor the expected Hall pattern, the bit WHE (Wrong Hall Event) is set, which can also cause an interrupt and set the IDLE mode to clear MCMP (modulation outputs are inactive).

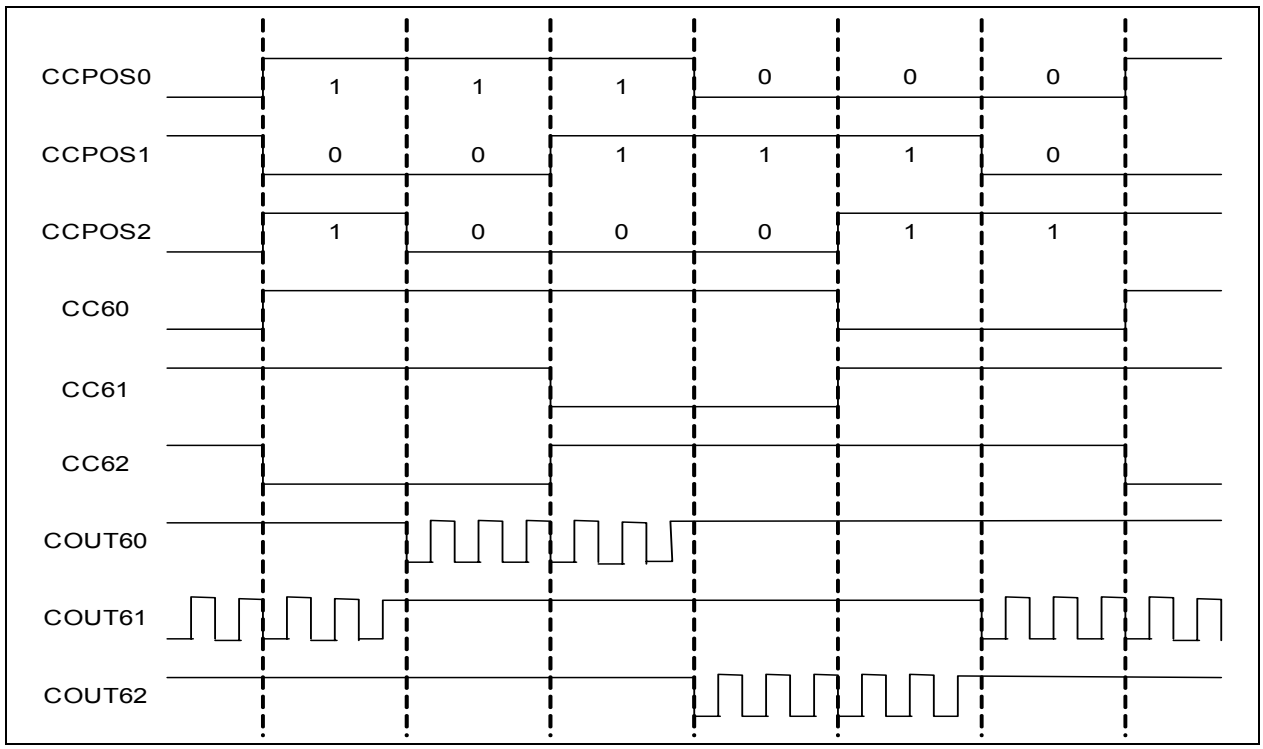



**Figure 15-17 Block Commutation in Rotate Left Mode**

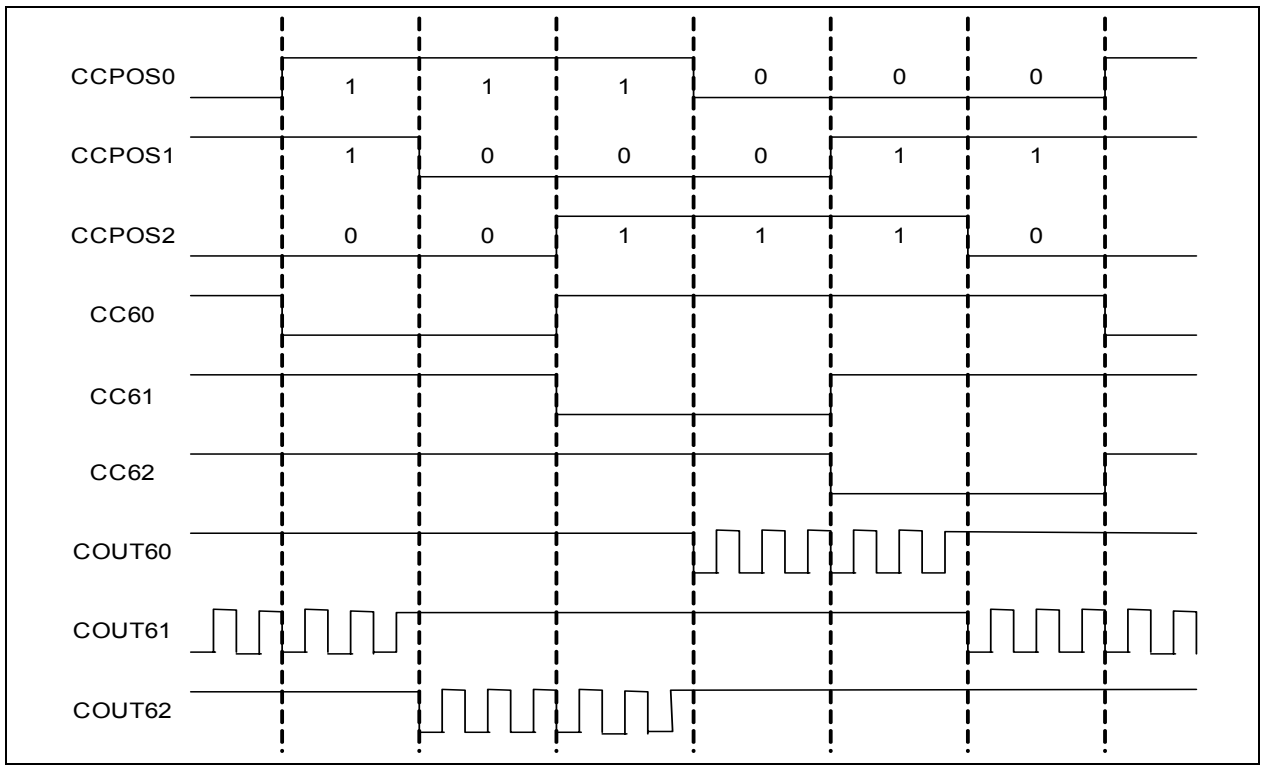



**Figure 15-18 Block Commutation in Rotate Right Mode**

### 15.1.7 Interrupt Generation

The interrupt generation can be triggered by the interrupt event or the setting of the corresponding interrupt bit in register IS by software. The interrupt is generated independently of the interrupt flag in register IS. Register IS can only be read; write actions have no impact on the contents of this register. The software can set or reset the bits individually by writing to register ISS or register ISR, respectively.

If enabled by the related interrupt enable bit in register IEN, an interrupt will be generated. The interrupt sources of the CCU6 module can be mapped to four interrupt output lines by programming the interrupt node pointer register INP.

### 15.1.8 Low Power Mode

If the CCU6 functionality is not required at all, it can be completely disabled by gating off its clock input for maximal power reduction. This is done by setting bit CCU_DIS in register PMCON1 as described below. Refer to **Chapter 8.1.4** for details on peripheral clock management.

**PMCON1**
**Power Mode Control Register 1** **Reset Value: $00_H$**

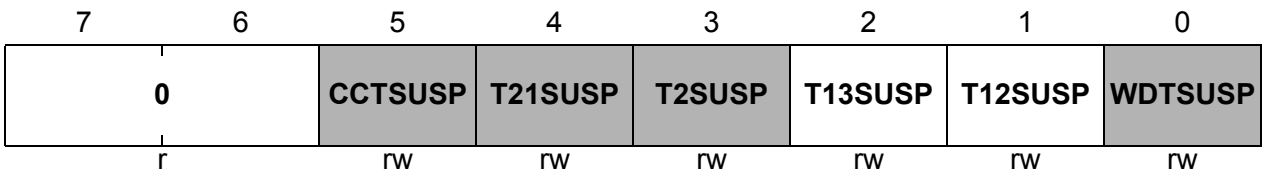
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **0** | **CDC_DIS** | **CAN_DIS** | **MDU_DIS** | **T2CCU _DIS** | **CCU_DIS** | **SSC_DIS** | **ADC_DIS** |
| r | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **CCU_DIS** | 2 | rw | **CCU6 Disable Request. Active high.**<br>0 CCU6 is in normal operation (default).<br>1 Request to disable the CCU6. |
| **0** | 7 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

### 15.1.9 Module Suspend Control

The timers of CCU6, Timer 12 and Timer 13, can be configured to stop their counting when the OCDS enters monitor mode (see **Chapter 18.3**) by setting their respective module suspend bits, T12SUSP and T13SUSP, in SFR MODSUSP.

**MODSUSP**
**Module Suspend Control Register** **Reset Value: $01_H$**




| Field | Bits | Typ | Description |
|---|---|---|---|
| **T12SUSP** | 1 | rw | **Timer 12 Debug Suspend Bit**<br>0 Timer 12 will not be suspended.<br>1 Timer 12 will be suspended. |
| **T13SUSP** | 2 | rw | **Timer 13 Debug Suspend Bit**<br>0 Timer 13 will not be suspended.<br>1 Timer 13 will be suspended. |
| **0** | [7:6] | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

### 15.1.10 Port Connection

Table 15-2 shows how bits and bit fields must be programmed for the required I/O functionality of the CCU6 I/O lines. This table also shows the values of the peripheral input select registers.

**Table 15-2 CCU6 I/O Control Selection**

| Port Lines | PISEL Register Bit | Input/Output Control Register Bits | I/O |
|---|---|---|---|
| P3.6/$\overline{\text{CTRAP_0}}$ | ISTRP = $00_B$ | P3_DIR.P6 = 0 | Input |
| P5.5/$\overline{\text{CTRAP_1}}$ | ISTRP = $01_B$ | P5_DIR.P5 = 0 | Input |
| P0.2/$\overline{\text{CTRAP_2}}$ | ISTRP = $10_B$ | P0_DIR.P2 = 0 | Input |
| P4.7/$\overline{\text{CTRAP_3}}$ | ISTRP = $11_B$ | P4_DIR.P7 = 0 | Input |
| P5.3/CCPOS0_0 | ISPOS0 = $00_B$ | P5_DIR.P3 = 0 | Input |
| P1.5/CCPOS0_1 | ISPOS0 = $01_B$ | P1_DIR.P5 = 0 | Input |
| P3.1/CCPOS0_2 | ISPOS0 = $10_B$ | P3_DIR.P1 = 0 | Input |
| P4.4/CCPOS0_3 | ISPOS0 = $11_B$ | P4_DIR.P4 = 0 | Input |
| P5.4/CCPOS1_0 | ISPOS1 = $00_B$ | P5_DIR.P4 = 0 | Input |
| P1.6/CCPOS1_1 | ISPOS1 = $01_B$ | P1_DIR.P6 = 0 | Input |
| P3.0/CCPOS1_2 | ISPOS1 = $10_B$ | P3_DIR.P0 = 0 | Input |
| P4.5/CCPOS1_3 | ISPOS1 = $11_B$ | P4_DIR.P5 = 0 | Input |
| P5.5/CCPOS2_0 | ISPOS2 = $00_B$ | P5_DIR.P5 = 0 | Input |
| P1.7/CCPOS2_1 | ISPOS2 = $01_B$ | P1_DIR.P7 = 0 | Input |
| P3.2/CCPOS2_2 | ISPOS2 = $10_B$ | P3_DIR.P2 = 0 | Input |
| P4.6/CCPOS2_3 | ISPOS2 = $11_B$ | P4_DIR.P6 = 0 | Input |
| P3.0/CC60_0 | ISCC60 = $00_B$ | P3_DIR.P0 = 0 | Input |
| | – | P3_DIR.P0 = 1 | Output |
| | | P3_ALTSEL0.P0 = 1 | |
| | | P3_ALTSEL1.P0 = 0 | |
| P4.0/CC60_1 | – | P4_DIR.P0 = 1 | Output |
| | | P4_ALTSEL0.P0 = 1 | |
| | | P4_ALTSEL1.P0 = 0 | |
| P5.5/CC60_3 | ISCC60 = $11_B$ | P5_DIR.P5 = 0 | Input |

**Table 15-2 CCU6 I/O Control Selection** (cont'd)

| Port Lines | PISEL Register Bit | Input/Output Control Register Bits | I/O |
|---|---|---|---|
| P3.1/COUT60_0 | – | P3_DIR.P1 = 1 | Output |
| | | P3_ALTSEL0.P1 = 1 | |
| | | P3_ALTSEL1.P1 = 0 | |
| P4.1/COUT60_1 | – | P4_DIR.P1 = 1 | Output |
| | | P4_ALTSEL0.P1 = 1 | |
| | | P4_ALTSEL1.P1 = 0 | |
| P3.2/CC61_0 | ISCC61 = $00_B$ | P3_DIR.P2 = 0 | Input |
| | – | P3_DIR.P2 = 1 | Output |
| | | P3_ALTSEL0.P2 = 1 | |
| | | P3_ALTSEL1.P2 = 0 | |
| P0.0/CC61_1 | ISCC61 = $01_B$ | P0_DIR.P0 = 0 | Input |
| | – | P0_DIR.P0 = 1 | Output |
| | | P0_ALTSEL0.P0 = 0 | |
| | | P0_ALTSEL1.P0 = 1 | |
| P3.1/CC61_2 | ISCC61 = $10_B$ | P3_DIR.P1 = 0 | Input |
| | – | P3_DIR.P1 = 1 | Output |
| | | P3_ALTSEL0.P1 = 0 | |
| | | P3_ALTSEL1.P1 = 1 | |
| P5.3/CC61_3 | ISCC61 = $11_B$ | P5_DIR.P3 = 0 | Input |
| P4.4/CC61_4 | – | P4_DIR.P4 = 1 | Output |
| | | P4_ALTSEL0.P4 = 1 | |
| | | P4_ALTSEL1.P4 = 0 | |
| P3.3/COUT61_0 | – | P3_DIR.P3 = 1 | Output |
| | | P3_ALTSEL0.P3 = 1 | |
| | | P3_ALTSEL1.P3 = 0 | |
| P0.1/COUT61_1 | – | P0_DIR.P1 = 1 | Output |
| | | P0_ALTSEL0.P1 = 0 | |
| | | P0_ALTSEL1.P1 = 1 | |

**Table 15-2 CCU6 I/O Control Selection** (cont'd)

| Port Lines | PISEL Register Bit | Input/Output Control Register Bits | I/O |
|---|---|---|---|
| P4.5/COUT61_2 | – | P4_DIR.P5 = 1 | Output |
| | | P4_ALTSEL0.P5 = 1 | |
| | | P4_ALTSEL1.P5 = 0 | |
| P3.4/CC62_0 | ISCC62= $00_B$ | P3_DIR.P4 = 0 | Input |
| | – | P3_DIR.P4 = 1 | Output |
| | | P3_ALTSEL0.P4 = 1 | |
| | | P3_ALTSEL1.P4 = 0 | |
| P0.4/CC62_1 | ISCC62 = $01_B$ | P0_DIR.P4 = 0 | Input |
| | – | P0_DIR.P4 = 1 | Output |
| | | P0_ALTSEL0.P4 = 0 | |
| | | P0_ALTSEL1.P4 = 1 | |
| P4.6/CC62_2 | – | P4_DIR.P6 = 1 | Output |
| | | P4_ALTSEL0.P6 = 1 | |
| | | P4_ALTSEL1.P6 = 0 | |
| P5.4/CC62_3 | ISCC62 = $11_B$ | P5_DIR.P4 = 0 | Input |
| P3.5/COUT62_0 | – | P3_DIR.P5 = 1 | Output |
| | | P3_ALTSEL0.P5 = 1 | |
| | | P3_ALTSEL1.P5 = 0 | |
| P0.5/COUT62_1 | – | P0_DIR.P5 = 1 | Output |
| | | P0_ALTSEL0.P5 = 0 | |
| | | P0_ALTSEL1.P5 = 1 | |
| P4.7/COUT62_2 | – | P4_DIR.P7 = 1 | Output |
| | | P4_ALTSEL0.P7 = 1 | |
| | | P4_ALTSEL1.P7 = 0 | |
| P3.7/COUT63_0 | – | P3_DIR.P7 = 1 | Output |
| | | P3_ALTSEL0.P7 = 1 | |
| | | P3_ALTSEL1.P7 = 0 | |
| P0.3/COUT63_1 | – | P0_DIR.P3 = 1 | Output |
| | | P0_ALTSEL0.P3 = 0 | |
| | | P0_ALTSEL1.P3 = 1 | |

**Table 15-2 CCU6 I/O Control Selection** (cont'd)

| Port Lines | PISEL Register Bit | Input/Output Control Register Bits | I/O |
|---|---|---|---|
| P4.3/COUT63_2 | – | P4_DIR.P3 = 1 | Output |
| | | P4_ALTSEL0.P3 = 0 | |
| | | P4_ALTSEL1.P3 = 1 | |
| P1.6/T12HR_0 | IST12HR = $00_B$ | P1_DIR.P6 = 0 | Input |
| P0.0/T12HR_1 | IST12HR = $01_B$ | P0_DIR.P0 = 0 | Input |
| P5.3/T12HR_2 | IST12HR = $10_B$ | P5_DIR.P3 = 0 | Input |
| P1.7/T13HR_0 | IST13HR = $00_B$ | P1_DIR.P7 = 0 | Input |
| P0.1/T13HR_1 | IST13HR = $01_B$ | P0_DIR.P1 = 0 | Input |
| P5.4/T13HR_2 | IST13HR = $10_B$ | P5_DIR.P4 = 0 | Input |

### 15.1.11 Synchronous Start of Timers

Timer 12 and Timer 13 can be started synchronously to the CCT timer of the T2CCU module. This is enabled by selecting T12HR_3 as the T12HR input and T13HR_3 as the T13HR input. In XC878, CCTST signal from T2CCU module is connected to both T12HR_3 and T13HR_3. If rising edge on T12HR_3 is selected as active edge by T12RSEL bit in TCTR2H register, the settng of bit CCTST in T2CCU_CCTCON register from 0 sets T12R bit. T13 can also be started synchronously with T12 and CCT as described in **Table 15-3**.

**Table 15-3 Selection for Synchronous Start of Timer**

| CCU Timer Input | PISEL Register Bit | Timer Run Selection |
|---|---|---|
| T12HR_3 | IST12HR = $11_B$ | TCTR2H.T12RSEL |
| T13HR_3 | IST13HR = $11_B$ | TCTR2H.T13RSEL |

## 15.2 Register Map

The CCU6 SFRs are located in the standard memory area (RMAP = 0) and are organized into 4 pages. The CCU6_PAGE register is located at address $A3_H$. It contains the page value and the page control information.

All CCU6 register names described in the following sections are referenced in other chapters of this document with the module name prefix “CCU6_”, e.g., CCU6_CC63SRL.

The addresses (non-mapped) of the kernel SFRs are listed in **Table 15-4**.

**Table 15-4 SFR Address List for Pages 0-3**

| Address | Page 0 | Page 1 | Page 2 | Page 3 |
|---|---|---|---|---|
| $9A_H$ | CC63SRL | CC63RL | T12MSELL | MCMOUTL |
| $9B_H$ | CC63SRH | CC63RH | T12MSELH | MCMOUTH |
| $9C_H$ | TCTR4L | T12PRL | IENL | ISL |
| $9D_H$ | TCTR4H | T12PRH | IENH | ISH |
| $9E_H$ | MCMOUTSL | T13PRL | INPL | PISEL0L |
| $9F_H$ | MCMOUTSH | T13PRH | INPH | PISEL0H |
| $A4_H$ | ISRL | T12DTCL | ISSL | PISEL2 |
| $A5_H$ | ISRH | T12DTCH | ISSH | |
| $A6_H$ | CMPMODIFL | TCTR0L | PSLR | |
| $A7_H$ | CMPMODIFH | TCTR0H | MCMCTR | |
| $FA_H$ | CC60SRL | CC60RL | TCTR2L | T12L |
| $FB_H$ | CC60SRH | CC60RH | TCTR2H | T12H |
| $FC_H$ | CC61SRL | CC61RL | MODCTRL | T13L |
| $FD_H$ | CC61SRH | CC61RH | MODCTRH | T13H |
| $FE_H$ | CC62SRL | CC62RL | TRPCTRL | CMPSTATL |
| $FF_H$ | CC62SRH | CC62RH | TRPCTRH | CMPSTATH |

## CCU6_PAGE
## Page Register for CCU6

**Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **OP** | | **STNR** | | **0** | **PAGE** | | |
| w | | w | | r | rwh | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **PAGE** | [2:0] | rwh | **Page Bits**<br>When written, the value indicates the new page address.<br>When read, the value indicates the currently active page = addr [y:x+1]. |
| **STNR** | [5:4] | w | **Storage Number**<br>This number indicates which storage bit field is the target of the operation defined by bit field OP.<br>If OP = $10_B$,<br>the contents of PAGE are saved in STx before being overwritten with the new value.<br>If OP = $11_B$,<br>the contents of PAGE are overwritten by the contents of STx. The value written to the bit positions of PAGE is ignored.<br><br>00 ST0 is selected.<br>01 ST1 is selected.<br>10 ST2 is selected.<br>11 ST3 is selected. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **OP** | [7:6] | w | **Operation**<br>0X Manual page mode. The value of STNR is ignored and PAGE is directly written.<br>10 New page programming with automatic page saving. The value written to the bit positions of PAGE is stored. In parallel, the previous contents of PAGE are saved in the storage bit field STx indicated by STNR.<br>11 Automatic restore page action. The value written to the bit positions PAGE is ignored and instead, PAGE is overwritten by the contents of the storage bit field STx indicated by STNR. |
| **0** | 3 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

## 15.3 Register Description

Table 15-5 shows all registers associated with the CCU6 module.

For all CCU6 registers, the write-only bit positions (indicated by “w”) always deliver the value of 0 when they are read out. If a hardware and a software request to modify a bit occur simultaneously, the software wins.

**Table 15-5 Registers Overview**

| Register Short Name | Register Long Name | Description see |
|---|---|---|
| **System Registers** | | |
| **PISEL0L** | Port Input Select Register 0 Low | Page 15-38 |
| **PISEL0H** | Port Input Select Register 0 High | Page 15-39 |
| **PISEL2** | Port Input Select Register 2 | Page 15-40 |
| **Timer T12 Registers** | | |
| **T12L** | Timer T12 Counter Register Low | Page 15-47 |
| **T12H** | Timer T12 Counter Register High | Page 15-47 |
| **T12PRL** | Timer T12 Period Register Low | Page 15-48 |
| **T12PRH** | Timer T12 Period Register High | Page 15-48 |
| **CC6xRL** | Capture/Compare Register for Channel CC6x Low | Page 15-49 |
| **CC6xRH** | Capture/Compare Register for Channel CC6x High | Page 15-49 |
| **CC6xSRL** | Compare Shadow Register for Channel CC6x Low | Page 15-49 |
| **CC6xSRH** | Compare Shadow Register for Channel CC6x High | Page 15-50 |
| **T12DTCL** | Timer T12 Dead-Time Control Register Low | Page 15-51 |
| **T12DTCH** | Timer T12 Dead-Time Control Register High | Page 15-51 |
| **Timer T13 Registers** | | |
| **T13L** | Timer T13 Counter Register Low | Page 15-52 |
| **T13H** | Timer T13 Counter Register High | Page 15-53 |
| **T13PRL** | Timer T13 Period Register Low | Page 15-53 |
| **T13PRH** | Timer T13 Period Register High | Page 15-54 |
| **CC63RL** | Capture/Compare Register for Channel CC63 Low | Page 15-54 |

**Table 15-5 Registers Overview** (cont'd)

| Register Short Name | Register Long Name | Description see |
|---|---|---|
| **CC63RH** | Capture/Compare Register for Channel CC63 High | **Page 15-54** |
| **CC63SRL** | Capture/Compare Shadow Register for Channel CC63 Low | **Page 15-55** |
| **CC63SRH** | Capture/Compare Shadow Register for Channel CC63 High | **Page 15-55** |
| **CCU6 Control Registers** | | |
| **CMPSTATL** | Compare State Register High | **Page 15-56** |
| **CMPSTATH** | Compare State Register High | **Page 15-57** |
| **CMPMODIFL** | Compare State Modification Register Low | **Page 15-59** |
| **CMPMODIFH** | Compare State Modification Register High | **Page 15-59** |
| **TCTR0L** | Timer Control Register 0 Low | **Page 15-60** |
| **TCTR0H** | Timer Control Register 0 High | **Page 15-61** |
| **TCTR2L** | Timer Control Register 2 Low | **Page 15-63** |
| **TCTR2H** | Timer Control Register 2 High | **Page 15-65** |
| **TCTR4L** | Timer Control Register 4 Low | **Page 15-66** |
| **TCTR4H** | Timer Control Register 4 High | **Page 15-67** |
| **Modulation Control Registers** | | |
| **MODCTRL** | Modulation Control Register Low | **Page 15-68** |
| **MODCTRH** | Modulation Control Register High | **Page 15-69** |
| **TRPCTRL** | Trap Control Register Low | **Page 15-70** |
| **TRPCTRH** | Trap Control Register High | **Page 15-72** |
| **PSLR** | Passive State Level Register | **Page 15-73** |
| **MCMOUTSL** | Multi_Channel Mode Output Shadow Register Low | **Page 15-74** |
| **MCMOUTSH** | Multi_Channel Mode Output Shadow Register High | **Page 15-75** |
| **MCMOUTL** | Multi_Channel Mode Output Register Low | **Page 15-76** |
| **MCMOUTH** | Multi_Channel Mode Output Register High | **Page 15-78** |
| **MCMCTR** | Multi_Channel Mode Control Register | **Page 15-79** |

**Table 15-5 Registers Overview** (cont'd)

| Register Short Name | Register Long Name | Description see |
|---|---|---|
| **T12MSELL** | T12 Capture/Compare Mode Select Register Low | **Page 15-43** |
| **T12MSELH** | T12 Capture/Compare Mode Select Register High | **Page 15-45** |
| **Interrupt Control Registers** | | |
| **ISL** | Interrupt Status Register Low | **Page 15-80** |
| **ISH** | Interrupt Status Register High | **Page 15-81** |
| **ISSL** | Interrupt Status Set Register Low | **Page 15-84** |
| **ISSH** | Interrupt Status Set Register High | **Page 15-85** |
| **ISRL** | Interrupt Status Reset Register Low | **Page 15-86** |
| **ISRH** | Interrupt Status Reset Register High | **Page 15-87** |
| **IENL** | Interrupt Enable Register Low | **Page 15-88** |
| **IENH** | Interrupt Enable Register High | **Page 15-90** |
| **INPL** | Interrupt Node Pointer Register Low | **Page 15-91** |
| **INPH** | Interrupt Node Pointer Register High | **Page 15-93** |

### 15.3.1 System Registers

Registers PISEL0 and PISEL2 contain bit fields that select the actual input port for the module inputs. This permits the adaptation of the pin functionality of the device to the application's requirements. The output pins are chosen according to the registers in the ports.

**PISEL0L**
**Port Input Select Register 0 Low** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **ISTRP** | | **ISCC62** | | **ISCC61** | | **ISCC60** | |
| **rw** | | **rw** | | **rw** | | **rw** | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **ISCC60** | 1:0 | rw | **Input Select for CC60**<br>This bit field defines the port pin that is used for the CC60 capture input signal.<br>00 The input pin for CC60_0.<br>01 Reserved<br>10 Reserved<br>11 The input pin for CC60_3. |
| **ISCC61** | 3:2 | rw | **Input Select for CC61**<br>This bit field defines the port pin that is used for the CC61 capture input signal.<br>00 The input pin for CC61_0.<br>01 The input pin for CC61_1.<br>10 The input pin for CC61_2.<br>11 The input pin for CC61_3. |
| **ISCC62** | 5:4 | rw | **Input Select for CC62**<br>This bit field defines the port pin that is used for the CC62 capture input signal.<br>00 The input pin for CC62_0.<br>01 The input pin for CC62_1.<br>10 Reserved<br>11 The input pin for CC62_3. |
| **ISTRP** | 7:6 | rw | **Input Select for CTRAP**<br>This bit field defines the port pin that is used for the CTRAP input signal.<br>00 The input pin for $\overline{\text{CTRAP_0}}$.<br>01 The input pin for $\overline{\text{CTRAP_1}}$.<br>10 The input pin for $\overline{\text{CTRAP_2}}$.<br>11 The input pin for $\overline{\text{CTRAP_3}}$ |

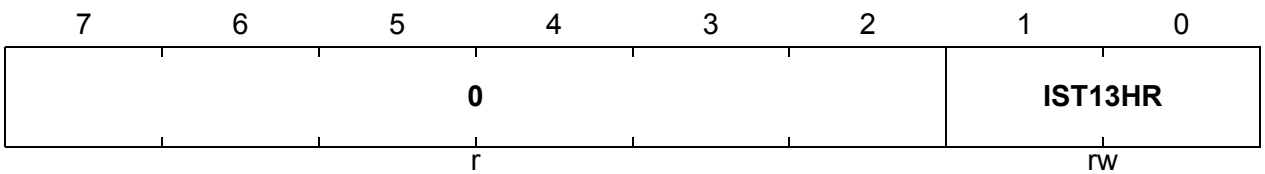
**PISEL0H**

**Port Input Select Register 0 High** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **IST12HR** | | **ISPOS2** | | **ISPOS1** | | **ISPOS0** | |
| **rw** | | **rw** | | **rw** | | **rw** | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **ISPOS0** | 1:0 | rw | **Input Select for CCPOS0**<br>This bit field defines the port pin that is used for the CCPOS0 input signal.<br>00 The input pin for CCPOS0_0.<br>01 The input pin for CCPOS0_1.<br>10 The input pin for CCPOS0_2.<br>11 The input pin for CCPOS0_3. |
| **ISPOS1** | 3:2 | rw | **Input Select for CCPOS1**<br>This bit field defines the port pin that is used for the CCPOS1 input signal.<br>00 The input pin for CCPOS1_0.<br>01 The input pin for CCPOS1_1.<br>10 The input pin for CCPOS1_2.<br>11 The input pin for CCPOS1_3 |
| **ISPOS2** | 5:4 | rw | **Input Select for CCPOS2**<br>This bit field defines the port pin that is used for the CCPOS2 input signal.<br>00 The input pin for CCPOS2_0.<br>01 The input pin for CCPOS2_1.<br>10 The input pin for CCPOS2_2.<br>11 The input pin for CCPOS2_3 |
| **IST12HR** | 7:6 | rw | **Input Select for T12HR**<br>This bit field defines the port pin that is used for the T12HR input signal.<br>00 The input pin for T12HR_0.<br>01 The input pin for T12HR_1.<br>10 The input pin for T12HR_2.<br>11 The input pin for T12HR_3. |

**PISEL2**
**Port Input Select Register 2** **Reset Value: $00_H$**



| Field | Bits | Type | Description |
|---|---|---|---|
| **IST13HR** | 1:0 | rw | **Input Select for T13HR**<br>This bit field defines the port pin that is used for the T13HR input signal.<br>00 The input pin for T13HR_0.<br>01 The input pin for T13HR_1.<br>10 The input pin for T13HR_2.<br>11 The input pin for T13HR_3. |
| **0** | 7:2 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

### 15.3.2 Timer 12 – Related Registers

The generation of the patterns for a 3-channel PWM is based on timer T12. The registers related to timer T12 can be concurrently updated (with well-defined conditions) in order to ensure consistency of the three PWM channels.

Timer T12 supports capture and compare modes, which can be independently selected for the three channels CC60, CC61, and CC62.

Register T12MSEL contains control bits to select the capture/compare functionality of the three channels of timer T12. **Table 15-6**, **Table 15-7** and **Table 15-8** define and elaborate some of the capture/compare modes selectable. Refer to the following register description for the selection.

**Table 15-6 Double-Register Capture Modes**

| Description | |
|---|---|
| 0100 | The contents of T12 are stored in CC6nR after a rising edge and in CC6nSR after a falling edge on the input pin CC6n. |
| 0101 | The value stored in CC6nSR is copied to CC6nR after a rising edge on the input pin CC6n. The actual timer value of T12 is simultaneously stored in the shadow register CC6nSR. This feature is useful for time measurements between consecutive rising edges on pins CC6n. COUT6n is I/O. |

**Table 15-6 Double-Register Capture Modes** (cont'd)

| | Description |
|---|---|
| 0110 | The value stored in CC6nSR is copied to CC6nR after a falling edge on the input pin CC6n. The actual timer value of T12 is simultaneously stored in the shadow register CC6nSR. This feature is useful for time measurements between consecutive falling edges on pins CC6n. COUT6n is I/O. |
| 0111 | The value stored in CC6nSR is copied to CC6nR after any edge on the input pin CC6n. The actual timer value of T12 is simultaneously stored in the shadow register CC6nSR. This feature is useful for time measurements between consecutive edges on pins CC6n. COUT6n is I/O. |

**Table 15-7 Combined T12 Modes**

| | Description |
|---|---|
| 1000 | Hall Sensor mode:<br>Capture mode for channel 0, compare mode for channels 1 and 2. The contents of T12 are captured into CC60 at a valid hall event (which is a reference to the actual speed). CC61 can be used for a phase delay function between hall event and output switching. CC62 can act as a time-out trigger if the expected hall event comes too late. The value $1000_B$ must be programmed to MSEL0, MSEL1 and MSEL2 if the hall signals are used. In this mode, the contents of timer T12 are captured in CC60 and T12 is reset after the detection of a valid hall event. In order to avoid noise effects, the dead-time counter channel 0 is started after an edge has been detected at the hall inputs. On reaching the value of $000001_B$, the hall inputs are sampled and the pattern comparison is done. |
| 1001 | Hysteresis-like control mode with dead-time generation:<br>The negative edge of the CCPOSx input signal is used to reset bit CC6nST. As a result, the output signals can be switched to passive state immediately and switch back to active state (with dead-time) if the CCPOSx is high and the bit CC6nST is set by a compare event. |

**Table 15-8 Multi-Input Capture Modes**

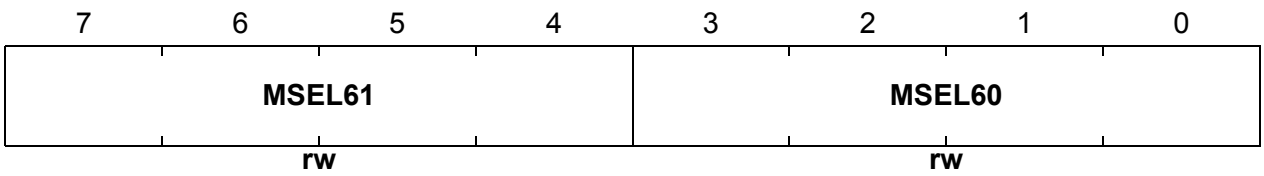
| | Description |
|---|---|
| 1010 | The timer value of T12 is stored in CC6nR after a rising edge at the input pin CC6n. The timer value of T12 is stored in CC6nSR after a falling edge at the input pin CCPOSx. |
| 1011 | The timer value of T12 is stored in CC6nR after a falling edge at the input pin CC6n. The timer value of T12 is stored in CC6nSR after a rising edge at the input pin CCPOSx. |

**Table 15-8 Multi-Input Capture Modes**

| | Description |
|---|---|
| 1100 | The timer value of T12 is stored in CC6nR after a rising edge at the input pin CC6n. The timer value of T12 is stored in CC6nSR after a rising edge at the input pin CCPOSx. |
| 1101 | The timer value of T12 is stored in CC6nR after a falling edge at the input pin CC6n. The timer value of T12 is stored in CC6nSR after a falling edge at the input pin CCPOSx. |
| 1110 | The timer value of T12 is stored in CC6nR after any edge at the input pin CC6n. The timer value of T12 is stored in CC6nSR after any edge at the input pin CCPOSx. |
| 1111 | reserved (no capture or compare action) |

**T12MSELL**

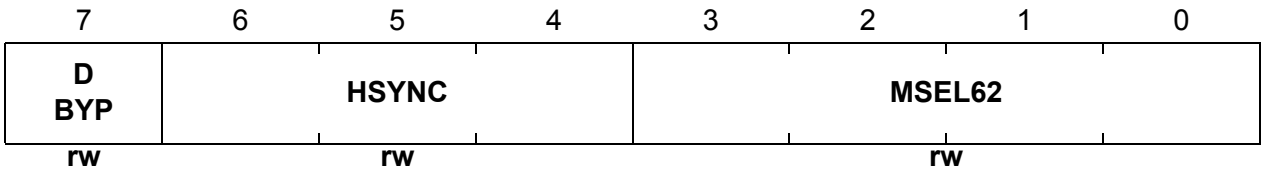
**T12 Capture/Compare Mode Select Register Low** **Reset Value: $00_H$**



| Field | Bits | Type | Description |
|---|---|---|---|
| **MSEL60, MSEL61** | 3:0, 7:4 | rw | **Capture/Compare Mode Selection**<br>These bit fields select the operating mode of the three timer T12 capture/compare channels. Each channel (n = 0, 1, 2) can be programmed individually either for compare or capture operation according to:<br>0000 Compare outputs disabled, pins CC6n and COUT6n can be used for I/O. No capture action.<br>0001 Compare output on pin CC6n, pin COUT6n can be used for I/O. No capture action.<br>0010 Compare output on pin COUT6n, pin CC6n can be used for I/O. No capture action.<br>0011 Compare output on pins COUT6n and CC6n.<br>01XX Double-Register Capture modes, see **Table 15-6**.<br>1000 Hall Sensor mode, see **Table 15-7**. In order to enable the hall edge detection, all three MSEL6x must be programmed to Hall Sensor mode.<br>1001 Hysteresis-like mode, see **Table 15-7**.<br>101X Multi-Input Capture modes, see **Table 15-8**.<br>11XX Multi-Input Capture modes, see **Table 15-8**. |

**T12MSELH**

**T12 Capture/Compare Mode Select Register High** **Reset Value: $00_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **MSEL62** | 3:0 | rw | **Capture/Compare Mode Selection**<br>These bit fields select the operating mode of the three timer T12 capture/compare channels. Each channel (n = 0, 1, 2) can be programmed individually either for compare or capture operation according to:<br>0000 Compare outputs disabled, pins CC6n and COUT6n can be used for I/O. No capture action.<br>0001 Compare output on pin CC6n, pin COUT6n can be used for I/O. No capture action.<br>0010 Compare output on pin COUT6n, pin CC6n can be used for I/O. No capture action.<br>0011 Compare output on pins COUT6n and CC6n.<br>01XX Double-Register Capture modes, see **Table 15-6**.<br>1000 Hall Sensor mode, see **Table 15-7**. In order to enable the hall edge detection, all three MSEL6x must be programmed to Hall Sensor mode.<br>1001 Hysteresis-like mode, see **Table 15-7**.<br>101X Multi-Input Capture modes, see **Table 15-8**.<br>11XX Multi-Input Capture modes, see **Table 15-8**. |

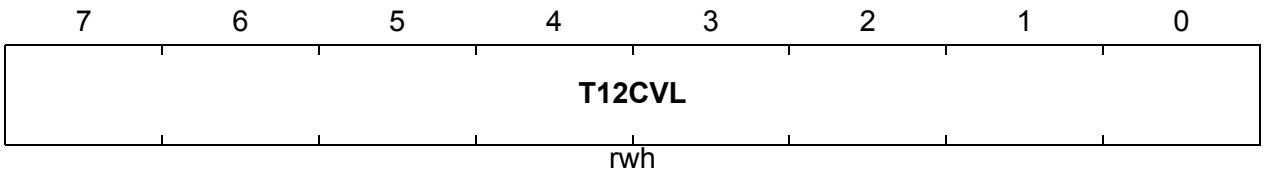
| Field | Bits | Type | Description |
|---|---|---|---|
| **HSYNC** | 6:4 | rw | **Hall Synchronization**<br>Bit field HSYNC defines the source for the sampling of the Hall input pattern and the comparison to the current and the expected Hall pattern bit fields. In all modes, a trigger by software by writing a 1 to bit SWHC is possible.<br>000 Any edge at one of the inputs CCPOSx (x = 0, 1, 2) triggers the sampling.<br>001 A T13 compare-match triggers the sampling.<br>010 A T13 period-match triggers the sampling.<br>011 The Hall sampling triggered by hardware sources is switched off.<br>100 A T12 period-match (while counting up) triggers the sampling.<br>101 A T12 one-match (while counting down) triggers the sampling.<br>110 A T12 compare-match of channel 0 (while counting up) triggers the sampling.<br>111 A T12 compare-match of channel 0 (while counting down) triggers the sampling. |
| **DBYP** | 7 | rw | **Delay Bypass**<br>Bit DBYP defines if the source signal for the sampling of the Hall input pattern (selected by HSYNC) uses the dead-time counter DTC0 of timer T12 as additional delay or if the delay is bypassed.<br>0 The delay bypass is not active. The dead-time counter DTC0 is generating a delay after the source signal becomes active.<br>1 The delay bypass is active. The dead-time counter DTC0 is not used by the sampling of the Hall pattern. |

*Note: In the capture modes, all edges at the CC6x inputs lead to the setting of the corresponding interrupt status flags in register IS. In order to monitor the selected capture events at the CCPOSx inputs in the multi-input capture modes, the CC6xST bits of the corresponding channel are set when detecting the selected event. The interrupt status bits and the CC6xST bits must be reset by software.*

Register T12 represents the counting value of timer T12. It can only be written while the timer T12 is stopped. Write actions while T12 is running are not taken into account. Register T12 can always be read by software.

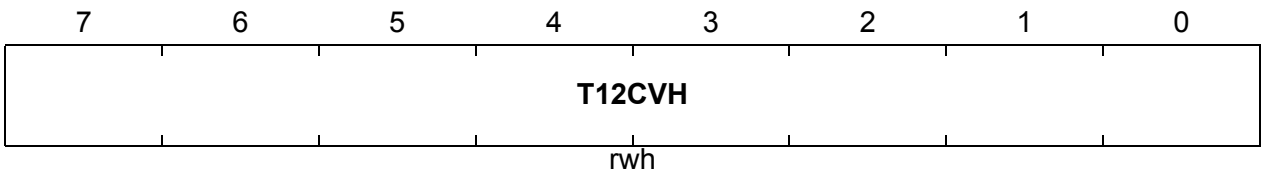
In edge-aligned mode, T12 only counts up, whereas in center-aligned mode, T12 can count up and down.

**T12L**
**Timer T12 Counter Register Low** **Reset Value: $00_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **T12CVL** | 7:0 | rwh | **Timer T12 Counter Value Low Byte**<br>This register represents the lower 8-bit counter value of timer T12. |

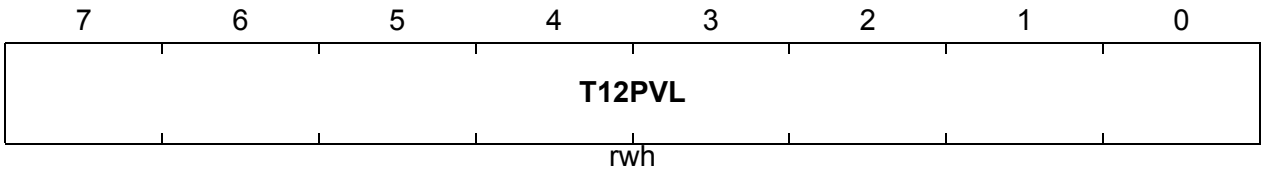
**T12H**
**Timer T12 Counter Register High** **Reset Value: $00_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **T12CVH** | 7:0 | rwh | **Timer T12 Counter Value High Byte**<br>This register represents the upper 8-bit counter value of timer T12. |

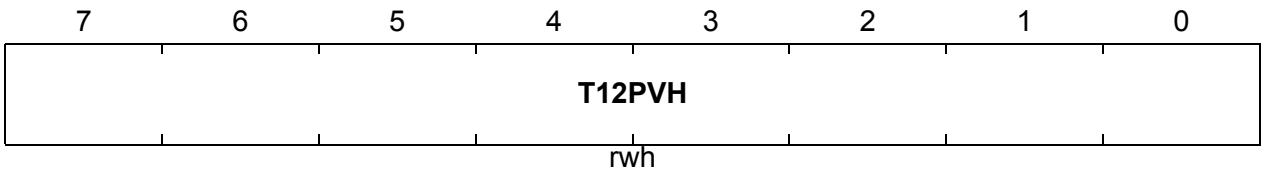
*Note: While timer T12 is stopped, the internal clock divider is reset in order to ensure reproducible timings and delays.*

**T12PRL**
**Timer T12 Period Register Low** **Reset Value: $00_H$**



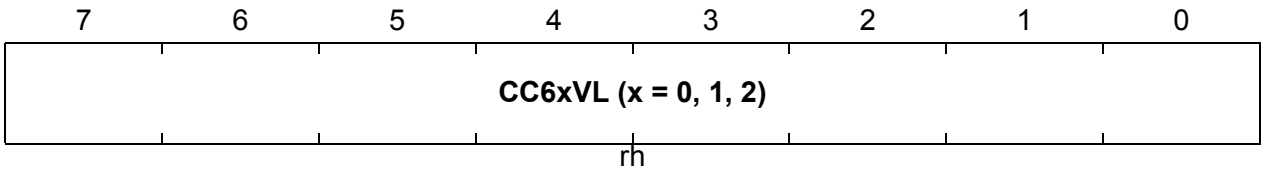

| Field | Bits | Type | Description |
|---|---|---|---|
| **T12PVL** | 7:0 | rwh | **T12 Period Value Low Byte**<br>The value T12PV defines the counter value for T12, which leads to a period-match. On reaching this value, the timer T12 is set to zero (edge-aligned mode) or changes its count direction to down counting (center-aligned mode). |

**T12PRH**
**Timer T12 Period Register High** **Reset Value: $00_H$**



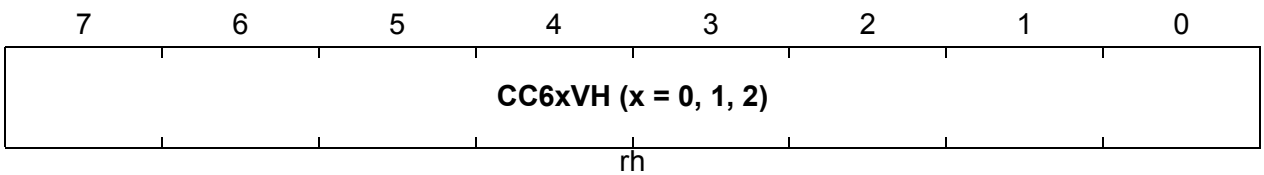

| Field | Bits | Type | Description |
|---|---|---|---|
| **T12PVH** | 7:0 | rwh | **T12 Period Value High Byte**<br>The value T12PV defines the counter value for T12, which leads to a period-match. On reaching this value, the timer T12 is set to zero (edge-aligned mode) or changes its count direction to down counting (center-aligned mode). |

**CC6xRL (x = 0, 1, 2)**
**Capture/Compare Register for Channel CC6x Low** **Reset Value: $00_H$**



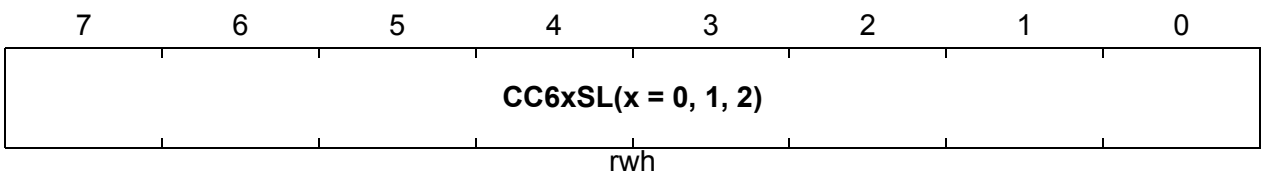

| Field | Bits | Type | Description |
|---|---|---|---|
| **CC6xVL (x = 0, 1, 2)** | 7:0 | rh | **Channel x Capture/Compare Value Low Byte** In compare mode, the bit fields CC6xV contain the values that are compared to the T12 counter value. In capture mode, the captured value of T12 can be read from these registers. |

**CC6xRH (x = 0, 1, 2)**
**Capture/Compare Register for Channel CC6x High** **Reset Value: $00_H$**



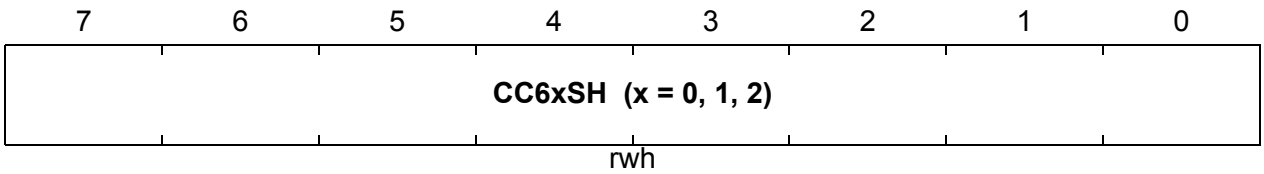

| Field | Bits | Type | Description |
|---|---|---|---|
| **CC6xVH (x = 0, 1, 2)** | 7:0 | rh | **Channel x Capture/Compare Value High Byte** In compare mode, the bit fields CC6xV contain the values that are compared to the T12 counter value. In capture mode, the captured value of T12 can be read from these registers. |

**CC6xSRL (x = 0, 1, 2)**
**Capture/Compare Shadow Register for Channel CC6x Low** **Reset Value: $00_H$**



| Field | Bits | Type | Description |
|---|---|---|---|
| **CC6xSL (x = 0, 1, 2)** | 7:0 | rwh | **Shadow Register for Channel x Capture/Compare Value Low Byte**<br>In compare mode, the contents of bit field CC6xS are transferred to the bit field CC6xV during a shadow transfer. In capture mode, the captured value of T12 can be read from these registers. |

**CC6xSRH (x = 0, 1, 2)**
**Capture/Compare Shadow Register for Channel CC6x High** **Reset Value: $00_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **CC6xSH (x = 0, 1, 2)** | 7:0 | rwh | **Shadow Register for Channel x Capture/Compare Value High Byte**<br>In compare mode, the contents of bit field CC6xS are transferred to the bit field CC6xV during a shadow transfer. In capture mode, the captured value of T12 can be read from these registers. |

**T12DTCL**
**Dead-Time Control Register for Timer T12 Low** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| DTM | | | | | | | |
| rw | | | | | | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **DTM** | 7:0 | rw | **Dead-Time**<br>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. |

**T12DTCH**
**Dead-Time Control Register for Timer T12 High** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | DTR2 | DTR1 | DTR0 | 0 | DTE2 | DTE1 | DTE0 |
| r | rh | rh | rh | r | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **DTEx**<br>**(x = 0, 1, 2)** | 2:0 | rw | **Dead-Time Enable Bits**<br>Bits DTE0..DTE2 enable and disable the dead-time generation for each compare channel (0, 1, 2) of timer T12.<br>0 Dead-time generation is disabled. The corresponding outputs switch from the passive state to the active state (according to the actual compare status) without any delay.<br>1 Dead-time generation 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. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **DTRx (x = 0, 1, 2)** | 6:4 | rh | **Dead-Time Run Indication Bits**<br>Bits DTR0..DTR2 indicate the status of the dead-time generation for each compare channel (0, 1, 2) of timer T12.<br>0 The value of the corresponding dead-time counter channel is 0.<br>1 The value of the corresponding dead-time counter channel is not 0. |
| **0** | 3, 7 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

*Note: 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.*

### 15.3.3 Timer 13 – Related Registers

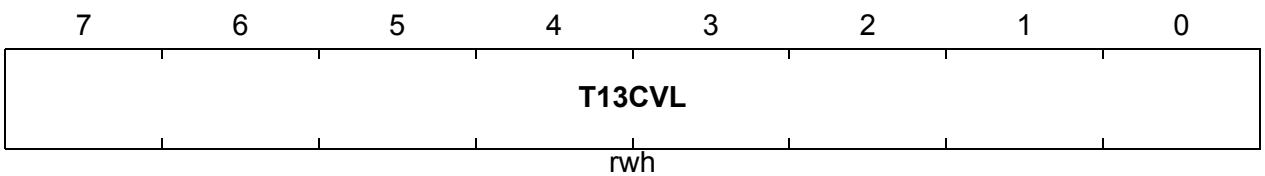
The generation of the patterns for a single channel PWM is based on timer T13. The registers related to timer T13 can be concurrently updated (with well-defined conditions) in order to ensure consistency of the PWM signal. T13 can be synchronized to several timer T12 events.

Timer T13 supports only 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 software.

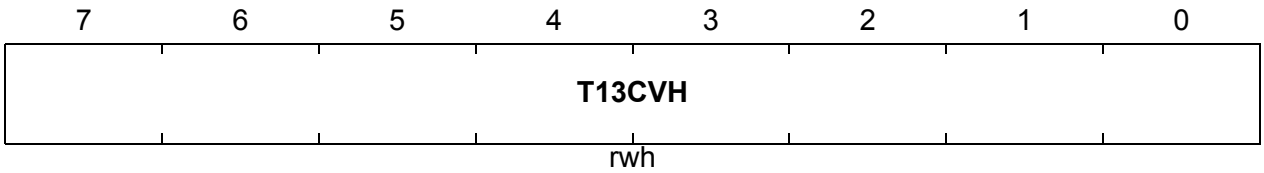
Timer T13 supports only edge-aligned mode (counting up).

**T13L**
**Timer T13 Counter Register Low** **Reset Value: $00_H$**



| Field | Bits | Type | Description |
|---|---|---|---|
| **T13CVL** | 7:0 | rwh | **Timer T13 Counter Value Low Byte**<br>This register represents the lower 8-bit counter value of timer T13. |

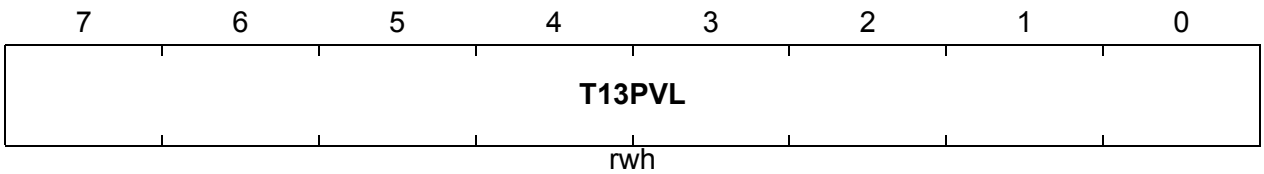
**T13H**
**Timer T13 Counter Register High** **Reset Value: $00_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **T13CVH** | 7:0 | rwh | **Timer T13 Counter Value High Byte**<br>This register represents the upper 8-bit counter value of timer T13. |

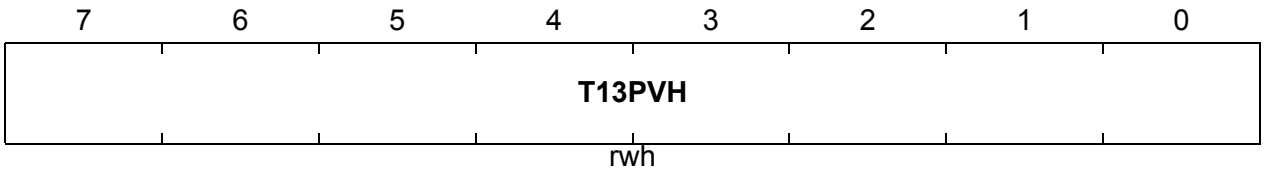
*Note: While timer T13 is stopped, the internal clock divider is reset in order to ensure reproducible timings and delays.*

**T13PRL**
**Timer T13 Period Register Low** **Reset Value: $00_H$**



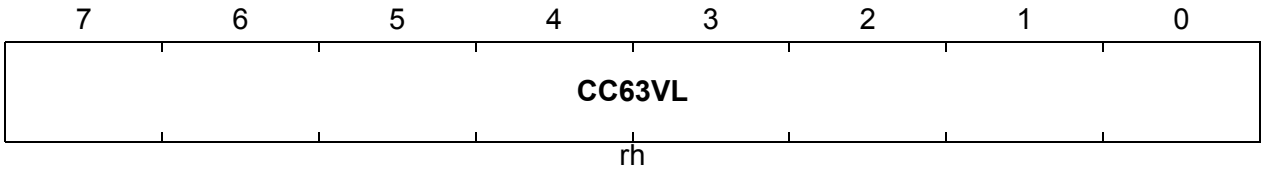

| Field | Bits | Type | Description |
|---|---|---|---|
| **T13PVL** | 7:0 | rwh | **T13 Period Value Low Byte**<br>The value T13PV defines the counter value for T13, which leads to a period-match. On reaching this value, the timer T13 is set to zero. |

**T13PRH**
**Timer T13 Period Register High** **Reset Value: $00_H$**



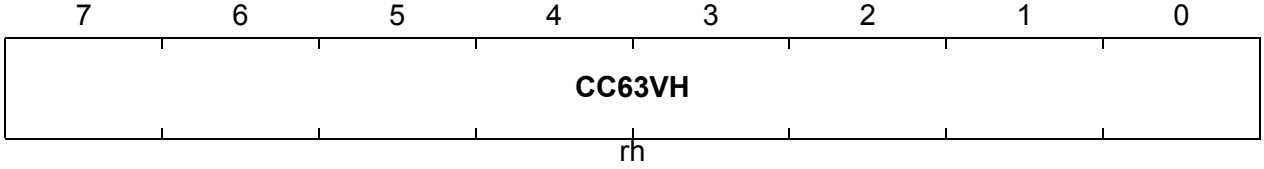

| Field | Bits | Type | Description |
|---|---|---|---|
| **T13PVH** | 7:0 | rwh | **T13 Period Value High Byte**<br>The value T13PV defines the counter value for T13, which leads to a period-match. On reaching this value, the timer T13 is set to zero. |

**CC63RL**
**Capture/Compare Register for Channel CC63 Low** **Reset Value: $00_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **CC63VL** | 7:0 | rh | **Channel CC63 Compare Value Low Byte**<br>The bit field CC63V contains the value that is compared to the T13 counter value. |

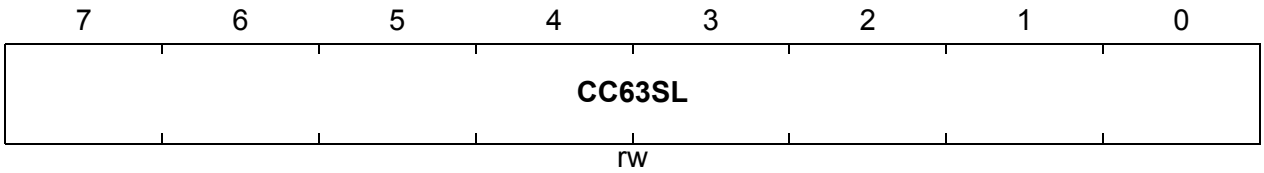
**CC63RH**
**Capture/Compare Register for Channel CC63 High** **Reset Value: $00_H$**



| Field | Bits | Type | Description |
|---|---|---|---|
| **CC63VH** | 7:0 | rh | **Channel CC63 Compare Value High Byte**<br>The bit field CC63V contains the value that is compared to the T13 counter value. |

**CC63SRL**

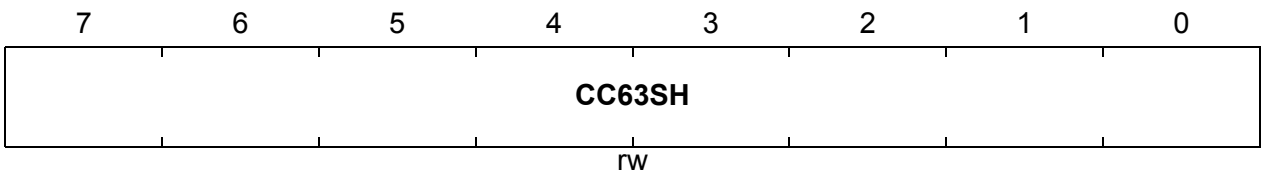
**Capture/Compare Shadow Register for Channel CC63 Low** **Reset Value: $00_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **CC63SL** | 7:0 | rw | **Shadow Register for Channel CC63 Compare Value Low Byte**<br>The contents of bit field CC63S are transferred to the bit field CC63V during a shadow transfer. |

**CC63SRH**

**Capture/Compare Shadow Register for Channel CC63 High** **Reset Value: $00_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **CC63SH** | 7:0 | rw | **Shadow Register for Channel CC63 Compare Value High Byte**<br>The contents of bit field CC63S are transferred to the bit field CC63V during a shadow transfer. |

### 15.3.4 Capture/Compare Control Registers

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.

**CMPSTATL**
**Compare State Register Low** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **0** | **CC 63ST** | **CC POS 2** | **CC POS 1** | **CC POS 0** | **CC 62ST** | **CC 61ST** | **CC 60ST** |
| **r** | **rh** | **rh** | **rh** | **rh** | **rh** | **rh** | **rh** |

| Field | Bits | Type | Description |
|---|---|---|---|
| **CC6xST (x = 0, 1, 2, 3)** | 0, 1, 2, 6 | rh | **Capture/Compare State Bits**<br>Bits CC6xST monitor the state of the capture/compare channels. Bits CC6xST are related to T12; bit CC63ST is related to T13.<br>0 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 reset by software the last time.<br>1 In compare mode, the counter value is greater than or equal to the compare value. In capture mode, the selected edge has been detected.<br>These bits are set and reset according to the T12 and T13 switching rules. |
| **CCPOSx (x = 0, 1, 2)** | 3, 4, 5 | rh | **Sampled Hall Pattern Bits**<br>Bits CCPOSx indicate 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.<br>0 The input CCPOSx has been sampled as 0.<br>1 The input CCPOSx has been sampled as 1. |
| **0** | 7 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

**CMPSTATH**
**Compare State Register High** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **T13 IM** | **C OUT63PS** | **C OUT62PS** | **CC 62PS** | **C OUT61PS** | **CC 61PS** | **C OUT60PS** | **CC 60PS** |
| **rwh** | **rwh** | **rwh** | **rwh** | **rwh** | **rwh** | **rwh** | **rwh** |

| Field | Bits | Type | Description |
|---|---|---|---|
| **CC6xPS (x = 0, 1, 2)**<br><br>**COUT6xPS (x = 0, 1, 2, 3)** | 0, 2, 4<br><br>1, 3, 5, 6 | rwh | **Passive State Select for Compare Outputs**<br>Bits CC6xPS, COUT6xPS select the state of the corresponding compare channel, which 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 COUT63PS is related to T13.<br>0 The corresponding compare output drives passive level while CC6xST is 0.<br>1 The corresponding compare output drives passive level while CC6xST is 1.<br>These bits have shadow bits and are updated in parallel to the capture/compare registers of T12 and T13, respectively. A read action targets the actually used values, whereas a write action targets the shadow bits.<br>In capture mode, these bits are not used. |
| **T13IM** | 7 | rwh | **T13 Inverted Modulation**<br>Bit T13IM inverts the T13 signal for the modulation of the CC6x and COUT6x (x = 0, 1, 2) signals.<br>0 T13 output is not inverted.<br>1 T13 output is inverted for further modulation.<br>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. |

The Compare Status Modification Register contains control bits allowing for modification by software of the Capture/Compare state bits.

**CMPMODIFL**
**Compare State Modification Register Low** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **0** | **MCC 63S** | **0** | | | **MCC 62S** | **MCC 61S** | **MCC 60S** |
| **r** | **w** | **r** | | | **w** | **w** | **w** |

| Field | Bits | Type | Description |
|---|---|---|---|
| **MCC6xS (x = 0, 1, 2, 3)** | 0, 1, 2, 6 | w | **Capture/Compare Status Modification Bits (Set)**<br>These bits are used to set the corresponding CC6xST bits by software.<br>This feature allows the user to individually change the status of the output lines by software, e.g. when the corresponding compare timer is stopped. This allows a bit manipulation of CC6xST-bits by a single data write action.<br>The following functionality of a write access to bits concerning the same capture/compare state bit is provided:<br>MCC6xR, MCC6xS =<br>0,0 Bit CC6xST is not changed.<br>0,1 Bit CC6xST is set.<br>1,0 Bit CC6xST is reset.<br>1,1 Reserved (toggle) |
| **0** | 5:3,7 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

**CMPMODIFH**
**Compare State Modification Register High** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **0** | **MCC 63R** | **0** | | | **MCC 62R** | **MCC 61R** | **MCC 60R** |
| **r** | **w** | **r** | | | **w** | **w** | **w** |

| Field | Bits | Type | Description |
|---|---|---|---|
| **MCC6xR (x = 0, 1, 2, 3)** | 0, 1, 2, 6 | w | **Capture/Compare Status Modification Bits (Reset)**<br>These bits are used to reset the corresponding CC6xST bits by software.<br>This feature allows the user to individually change the status of the output lines by software, e.g. when the corresponding compare timer is stopped. This allows a bit manipulation of CC6xST-bits by a single data write action.<br>The following functionality of a write access to bits concerning the same capture/compare state bit is provided:<br>MCC6xR, MCC6xS =<br>0,0 Bit CC6xST is not changed.<br>0,1 Bit CC6xST is set.<br>1,0 Bit CC6xST is reset.<br>1,1 Reserved (toggle) |
| **0** | 5:3,7 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

Register TCTR0 controls the basic functionality of both timers T12 and T13.

**TCTR0L**
**Timer Control Register 0 Low** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| CTM | CDIR | STE12 | T12R | T12 PRE | T12CLK | | |
| rw | rh | rh | rh | rw | rw | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **T12CLK** | 2:0 | rw | **Timer T12 Input Clock Select**<br>Selects the input clock for timer T12 which is derived from the peripheral clock according to the equation $f_{T12} = f_{CCU}/2^{<T12CLK>}$.<br>000 $f_{T12} = f_{CCU}$<br>001 $f_{T12} = f_{CCU}/2$<br>010 $f_{T12} = f_{CCU}/4$<br>011 $f_{T12} = f_{CCU}/8$<br>100 $f_{T12} = f_{CCU}/16$<br>101 $f_{T12} = f_{CCU}/32$<br>110 $f_{T12} = f_{CCU}/64$<br>111 $f_{T12} = f_{CCU}/128$ |
| **T12PRE** | 3 | rw | **Timer T12 Prescaler Bit**<br>In order to support higher clock frequencies, an additional prescaler factor of 1/256 can be enabled for the prescaler for T12.<br>0 The additional prescaler for T12 is disabled.<br>1 The additional prescaler for T12 is enabled. |
| **T12R** | 4 | rh | **Timer T12 Run Bit**<br>T12R starts and stops timer T12. It is set/reset by software by setting bits T12RS or T12RR, or it is reset by hardware according to the function defined by bit field T12SSC.<br>0 Timer T12 is stopped.<br>1 Timer T12 is running.<br>A concurrent set/reset action on T12R (from T12SSC, T12RR or T12RS) will have no effect. The bit T12R will remain unchanged. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **STE12** | 5 | rh | **Timer T12 Shadow Transfer Enable**<br>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.<br>A T12 shadow transfer event is a period-match while counting up or a one-match while counting down.<br>0 The shadow register transfer is disabled.<br>1 The shadow register transfer is enabled. |
| **CDIR** | 6 | rh | **Count Direction of Timer T12**<br>This bit is set/reset according to the counting rules of T12.<br>0 T12 counts up.<br>1 T12 counts down. |
| **CTM** | 7 | rw | **T12 Operating Mode**<br>0 Edge-aligned Mode:<br>T12 always counts up and continues counting from zero after reaching the period value.<br>1 Center-aligned Mode:<br>T12 counts down after detecting a period-match and counts up after detecting a one-match. |

**TCTR0H**
**Timer Control Register 0 High** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **0** | | **STE13** | **T13R** | **T13PRE** | **T13CLK** | | |
| **r** | | **rh** | **rh** | **rw** | **rw** | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **T13CLK** | 2:0 | rw | **Timer T13 Input Clock Select**<br>Selects the input clock for timer T13 which is derived from the peripheral clock according to the equation $f_{T13} = f_{CCU}/2^{<T13CLK>}$.<br>000 $f_{T13} = f_{CCU}$<br>001 $f_{T13} = f_{CCU}/2$<br>010 $f_{T13} = f_{CCU}/4$<br>011 $f_{T13} = f_{CCU}/8$<br>100 $f_{T13} = f_{CCU}/16$<br>101 $f_{T13} = f_{CCU}/32$<br>110 $f_{T13} = f_{CCU}/64$<br>111 $f_{T13} = f_{CCU}/128$ |
| **T13PRE** | 3 | rw | **Timer T13 Prescaler Bit**<br>In order to support higher clock frequencies, an additional prescaler factor of 1/256 can be enabled for the prescaler for T13.<br>0 The additional prescaler for T13 is disabled.<br>1 The additional prescaler for T13 is enabled. |
| **T13R** | 4 | rh | **Timer T13 Run Bit**<br>T13R starts and stops timer T13. It is set/reset by software by setting bits T13RS or T13RR or it is set/reset by hardware according to the function defined by bit fields T13SSC, T13TEC and T13TED.<br>0 Timer T13 is stopped.<br>1 Timer T13 is running.<br>A concurrent set/reset action on T13R (from T13SSC, T13TEC, T13RR or T13RS) will have no effect. The bit T13R will remain unchanged. |
| **STE13** | 5 | rh | **Timer T13 Shadow Transfer Enable**<br>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.<br>A T13 shadow transfer event is a period-match.<br>0 The shadow register transfer is disabled.<br>1 The shadow register transfer is enabled. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | 7:6 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

*Note: A write action to the bit fields T12CLK or T12PRE is only taken into account when the timer T12 is not running (T12R = 0). A write action to the bit fields T13CLK or T13PRE is only taken into account when the timer T13 is not running (T13R = 0).*

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. For example, this feature can be used to trigger AD conversions, after a specified delay (to avoid problems due to switching noise), synchronously to a PWM event.

**TCTR2L**
**Timer Control Register 2 Low** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **0** | **T13 TED** | | | **T13 TEC** | | **T13 SSC** | **T12 SSC** |
| **r** | **rw** | | | **rw** | | **rw** | **rw** |

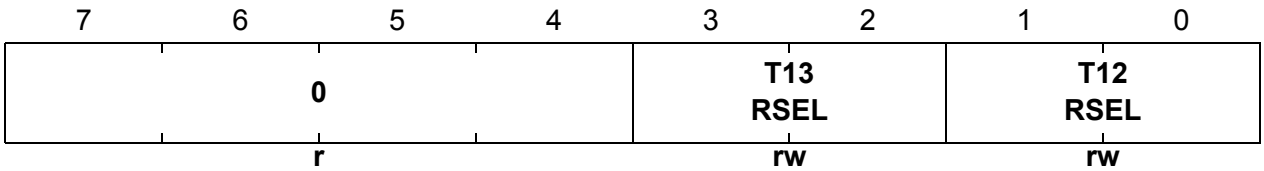
| Field | Bits | Type | Description |
|---|---|---|---|
| **T12SSC** | 0 | rw | **Timer T12 Single Shot Control**<br>This bit controls the single shot-mode of T12.<br>0 The single-shot mode is disabled, no hardware action on T12R.<br>1 The single shot mode is enabled, the bit T12R is reset by hardware if:<br>–T12 reaches its period value in edge-aligned mode<br>–T12 reaches the value 1 while down counting in center-aligned mode.<br>In parallel to the reset action of bit T12R, the bits CC6xST (x = 0, 1, 2) are reset. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **T13SSC** | 1 | rw | **Timer T13 Single Shot Control**<br>This bit controls the single shot-mode of T13.<br>0 No hardware action on T13R<br>1 The single-shot mode is enabled, the bit T13R is reset by hardware if T13 reaches its period value.<br>In parallel to the reset action of bit T13R, the bit CC63ST is reset. |
| **T13TEC** | 4:2 | rw | **T13 Trigger Event Control**<br>Bit field T13TEC selects the trigger event to start T13 (automatic set of T13R for synchronization to T12 compare signals) according to following combinations:<br>000 no action<br>001 set T13R on a T12 compare event on channel 0<br>010 set T13R on a T12 compare event on channel 1<br>011 set T13R on a T12 compare event on channel 2<br>100 set T13R on any T12 compare event on the channels 0, 1, or 2<br>101 set T13R upon a period-match of T12<br>110 set T13R upon a zero-match of T12 (while counting up)<br>111 set T13R on any edge of inputs CCPOSx |
| **T13TED** | 6:5 | rw | **Timer T13 Trigger Event Direction**<br>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.<br>00 no action<br>01 while T12 is counting up<br>10 while T12 is counting down<br>11 independent on the count direction of T12 |
| **0** | 7 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

Example:
If the timer T13 is intended to start at any compare event on T12 (T13TEC = $100_B$), 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 = $01_B$ or $11_B$.

**TCTR2H**
**Timer Control Register 2 High** **Reset Value: $00_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **T12RSEL** | 1:0 | rw | **Timer T12 External Run Selection**<br>Bit field T12RSEL defines the event of signal T12HR that can set the run bit T12R by hardware.<br>00 The external setting of T12R is disabled.<br>01 Bit T12R is set if a rising edge of signal T12HR is detected.<br>10 Bit T12R is set if a falling edge of signal T12HR is detected.<br>11 Bit T12R is set if an edge of signal T12HR is detected. |
| **T13RSEL** | 3:2 | rw | **Timer T13 External Run Selection**<br>Bit field T13RSEL defines the event of signal T13HR that can set the run bit T13R by hardware.<br>00 The external setting of T13R is disabled.<br>01 Bit T13R is set if a rising edge of signal T13HR is detected.<br>10 Bit T13R is set if a falling edge of signal T13HR is detected.<br>11 Bit T13R is set if an edge of signal T13HR is detected. |
| **0** | 7:4 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

Register TCTR4 allows the software control of the run bits T12R and T13R by independent set and reset conditions. Furthermore, the timers can be reset (while running) and the bits STE12 and STE13 can be controlled by software.

**TCTR4L**
**Timer Control Register 4 Low** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| T12 STD | T12 STR | 0 | | DT RES | T12 RES | T12 RS | T12 RR |
| w | w | r | | w | w | w | w |

| Field | Bits | Type | Description |
|---|---|---|---|
| **T12RR** | 0 | w | **Timer T12 Run Reset**<br>Setting this bit resets the T12R bit.<br>0 T12R is not influenced.<br>1 T12R is cleared, T12 stops counting. |
| **T12RS** | 1 | w | **Timer T12 Run Set**<br>Setting this bit sets the T12R bit.<br>0 T12R is not influenced.<br>1 T12R is set, T12 counts. |
| **T12RES** | 2 | w | **Timer T12 Reset**<br>0 No effect on T12.<br>1 The T12 counter register is reset to zero. The switching of the output signals is according to the switching rules. Setting of T12RES has no impact on bit T12R. |
| **DTRES** | 3 | w | **Dead-Time Counter Reset**<br>0 No effect on the dead-time counters.<br>1 The three dead-time counter channels are reset to zero. |
| **T12STR** | 6 | w | **Timer T12 Shadow Transfer Request**<br>0 No action<br>1 STE12 is set, enabling the shadow transfer. |
| **T12STD** | 7 | w | **Timer T12 Shadow Transfer Disable**<br>0 No action<br>1 STE12 is reset without triggering the shadow transfer. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | 5:4 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

**TCTR4H**
**Timer Control Register 4 High** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **T13 STD** | **T13 STR** | **0** | | | **T13 RES** | **T13 RS** | **T13 RR** |
| **w** | **w** | | **r** | | **w** | **w** | **w** |

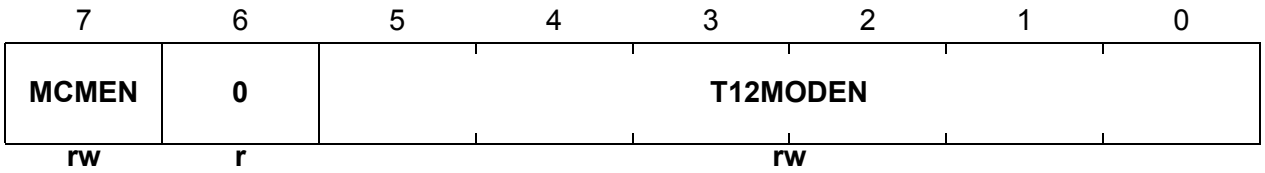
| Field | Bits | Type | Description |
|---|---|---|---|
| **T13RR** | 0 | w | **Timer T13 Run Reset**<br>Setting this bit resets the T13R bit.<br>0 T13R is not influenced.<br>1 T13R is cleared, T13 stops counting. |
| **T13RS** | 1 | w | **Timer T13 Run Set**<br>Setting this bit sets the T13R bit.<br>0 T13R is not influenced.<br>1 T13R is set, T13 counts. |
| **T13RES** | 2 | w | **Timer T13 Reset**<br>0 No effect on T13.<br>1 The T13 counter register is reset to zero. The switching of the output signals is according to the switching rules. Setting of T13RES has no impact on bit T13R. |
| **T13STR** | 6 | w | **Timer T13 Shadow Transfer Request**<br>0 No action<br>1 STE13 is set, enabling the shadow transfer. |
| **T13STD** | 7 | w | **Timer T13 Shadow Transfer Disable**<br>0 No action<br>1 STE13 is reset without triggering the shadow transfer. |
| **0** | 5:3 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

*Note: A simultaneous write of a 1 to bits which set and reset the same bit will trigger no action. The corresponding bit will remain unchanged.*

### 15.3.5 Global Modulation Control Registers

Register MODCTR contains control 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.

**MODCTRL**
**Modulation Control Register Low** **Reset Value: $00_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **T12MODEN** | 5:0 | rw | **T12 Modulation Enable**<br>Setting these bits enables the modulation of the corresponding compare channel by a PWM pattern generated by timer T12. The bit positions are corresponding to the following output signals:<br>Bit 0 modulation of CC60<br>Bit 1 modulation of COUT60<br>Bit 2 modulation of CC61<br>Bit 3 modulation of COUT61<br>Bit 4 modulation of CC62<br>Bit 5 modulation of COUT62<br>The enable feature of the modulation is defined as follows:<br>0 The modulation of the corresponding output signal by a T12 PWM pattern is disabled.<br>1 The modulation of the corresponding output signal by a T12 PWM pattern is enabled. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **MCMEN** | 7 | rw | **Multi-Channel Mode Enable**<br>0 The modulation of the corresponding output signal by a multi-channel pattern according to bit field MCMPis disabled.<br>1 The modulation of the corresponding output signal by a multi-channel pattern according to bit field MCMP is enabled. |
| **0** | 6 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

**MODCTRH**
**Modulation Control Register High** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **ECT 13O** | **0** | **T13MODEN** | | | | | |
| **rw** | **r** | **rw** | | | | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **T13MODEN** | 5:0 | rw | **T13 Modulation Enable**<br>Setting these bits enables the modulation of the corresponding compare channel by a PWM pattern generated by timer T13. The bit positions are corresponding to the following output signals:<br>Bit 0 modulation of CC60<br>Bit 1 modulation of COUT60<br>Bit 2 modulation of CC61<br>Bit 3 modulation of COUT61<br>Bit 4 modulation of CC62<br>Bit 5 modulation of COUT62<br>The enable feature of the modulation is defined as follows:<br>0 The modulation of the corresponding output signal by a T13 PWM pattern is disabled.<br>1 The modulation of the corresponding output signal by a T13 PWM pattern is enabled. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **ECT13O** | 7 | rw | **Enable Compare Timer T13 Output**<br>0 The alternate output function COUT63 is disabled.<br>1 The alternate output function COUT63 is enabled for the PWM signal generated by T13. |
| **0** | 6 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

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 $\overline{\text{CTRAP}}$ input pin, which is monitored (inverted level) by bit TRPF (in register IS). While TRPF = 1 (trap input active), the trap state bit TRPS (in register IS) is set to 1.

**TRPCTRL**
**Trap Control Register Low** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | | **0** | | | **TRPM2** | **TRPM1** | **TRPM0** |
| | | **r** | | | **rw** | **rw** | **rw** |

| Field | Bits | Type | Description |
|---|---|---|---|
| **TRPM0, TRPM1** | 1:0 | rw | **Trap Mode Control Bits 1, 0**<br>These two bits define the behavior of the selected outputs when leaving the trap state after the trap condition has become inactive again.<br>A synchronization to the timer driving the PWM pattern permits to avoid unintended short pulses when leaving the trap state. The combination (TRPM1, TRPM0) leads to:<br>00 The trap state is left (return to normal operation according to TRPM2) when a zero-match of T12 (while counting up) is detected (synchronization to T12).<br>01 The trap state is left (return to normal operation according to TRPM2) when a zero-match of T13 is detected (synchronization to T13).<br>10 reserved<br>11 The trap state is left (return to normal operation according to TRPM2) immediately without any synchronization to T12 or T13. |
| **TRPM2** | 2 | rw | **Trap Mode Control Bit 2**<br>0 The trap state can be left (return to normal operation = bit TRPS = 0) as soon as the input CTRAP becomes inactive. Bit TRPF is automatically cleared by hardware if the input pin CTRAP becomes 1. Bit TRPS is automatically cleared by hardware if bit TRPF is 0 and if the synchronization condition (according to TRPM0,1) is detected.<br>1 The trap state can be left (return to normal operation = bit TRPS = 0) as soon as bit TRPF is reset by software after the input CTRAP becomes inactive (TRPF is not cleared by hardware). Bit TRPS is automatically cleared by hardware if bit TRPF = 0 and if the synchronization condition (according to TRPM0,1) is detected. |
| **0** | 7:3 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

**TRPCTRH**
**Trap Control Register High** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| TRPPEN | TRPEN13 | TRPEN | | | | | |
| rw | rw | rw | | | | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **TRPEN** | 5:0 | rw | **Trap Enable Control**<br>Setting these bits enables the trap functionality for the following corresponding output signals:<br>Bit 0 trap functionality of CC60<br>Bit 1 trap functionality of COUT60<br>Bit 2 trap functionality of CC61<br>Bit 3 trap functionality of COUT61<br>Bit 4 trap functionality of CC62<br>Bit 5 trap functionality of COUT62<br>The enable feature of the trap functionality is defined as follows:<br>0 The trap functionality of the corresponding output signal is disabled. The output state is independent from bit TRPS.<br>1 The trap functionality of the corresponding output signal is enabled. The output is set to the passive state while TRPS = 1. |
| **TRPEN13** | 6 | rw | **Trap Enable Control for Timer T13**<br>0 The trap functionality for T13 is disabled. Timer T13 (if selected and enabled) provides PWM functionality even while TRPS = 1.<br>1 The trap functionality for T13 is enabled. The timer T13 PWM output signal is set to the passive state while TRPS = 1. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **TRPPEN** | 7 | rw | **Trap Pin Enable**<br>0 The trap functionality based on the input pin CTRAP is disabled. A trap can only be generated by software by setting bit TRPF.<br>1 The trap functionality based on the input pin CTRAP is enabled. A trap can be generated by software by setting bit TRPF or by CTRAP = 0. |

Register PSLR defines the passive state level driven by the output pins of the module. The passive state level is the value that is driven by the port pin during the passive state of the output. During the active state, the corresponding output pin drives the active state level, which is the inverted passive state level. The passive state level permits the adaptation of the driven output levels to the driver polarity (inverted, not inverted) of the connected power stage.

**PSLR**
**Passive State Level Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **PSL 63** | **0** | **PSL** | | | | | |
| **rwh** | **r** | **rwh** | | | | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **PSL** | 5:0 | rwh | **Compare Outputs Passive State Level**<br>The bits of this bit field define the passive level driven by the module outputs during the passive state. The bit positions are:<br>Bit 0 passive level for output CC60<br>Bit 1 passive level for output COUT60<br>Bit 2 passive level for output CC61<br>Bit 3 passive level for output COUT61<br>Bit 4 passive level for output CC62<br>Bit 5 passive level for output COUT62<br>The value of each bit position is defined as:<br>0 The passive level is 0.<br>1 The passive level is 1. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **PSL63** | 7 | rwh | **Passive State Level of Output COUT63**<br>This bit field defines the passive level of the output pin COUT63.<br>0 The passive level is 0.<br>1 The passive level is 1. |
| **0** | 6 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

*Note: Bit field PSL has a shadow register to allow for updates without undesired pulses on the output lines. The bits are updated with the T12 shadow transfer. A read action targets the actually used values, whereas a write action targets the shadow bits.*

*Note: Bit field PSL63 has a shadow register to allow for updates without undesired pulses on the output line. The bit is updated with the T13 shadow transfer. A read action targets the actually used values, whereas a write action targets the shadow bits.*

### 15.3.6 Multi-Channel Modulation Control Registers

Register MCMOUTS contains bits controlling the output states for multi-channel mode. Furthermore, the appropriate signals for the block commutation by Hall sensors can be selected. This register is a shadow register (that can be written) for register MCMOUT, which indicates the currently active signals.

**MCMOUTSL**
**Multi-Channel Mode Output Shadow Register Low** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **STR MCM** | **0** | **MCMPS** | | | | | |
| **w** | **r** | **rw** | | | | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **MCMPS** | 5:0 | rw | **Multi-Channel PWM Pattern Shadow**<br>Bit field MCMPS is the shadow bit field for bit field MCMP. The multi-channel shadow transfer is triggered according to the transfer conditions defined by register MCMCTR. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **STRMCM** | 7 | w | **Shadow Transfer Request for MCMPS**<br>Setting this bit during a write action leads to an immediate update of bit field MCMP by the value written to bit field MCMPS. This functionality permits an update triggered by software. When read, this bit always delivers 0.<br>0 Bit field MCMP is updated according to the defined hardware action. The write access to bit field MCMPS does not modify bit field MCMP.<br>1 Bit field MCMP is updated by the value written to bit field MCMPS. |
| **0** | 6 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

**MCMOUTSH**

**Multi-Channel Mode Output Shadow Register High** **Reset Value: $00_H$**

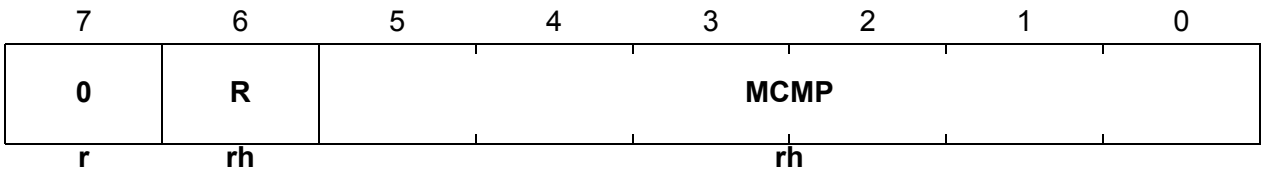
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **STRHP** | **0** | **CURHS** | | | **EXPHS** | | |
| **w** | **r** | **rw** | | | **rw** | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **EXPHS** | 2:0 | rw | **Expected Hall Pattern Shadow**<br>Bit field EXPHS is the shadow bit field for bit field EXPH. The bit field is transferred to bit field EXPH if an edge on the hall input pins CCPOSx (x = 0, 1, 2) is detected. |
| **CURHS** | 5:3 | rw | **Current Hall Pattern Shadow**<br>Bit field CURHS is the shadow bit field for bit field CURH. The bit field is transferred to bit field CURH if an edge on the hall input pins CCPOSx (x = 0, 1, 2) is detected. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **STRHP** | 7 | w | **Shadow Transfer Request for the Hall Pattern**<br>Setting these bits during a write action leads to an immediate update of bit fields CURH and EXPH by the value written to bit fields CURHS and EXPHS. This functionality permits an update triggered by software. When read, this bit always delivers 0.<br>0 The bit fields CURH and EXPH are updated according to the defined hardware action. The write access to bit fields CURHS and EXPHS does not modify the bit fields CURH and EXPH.<br>1 The bit fields CURH and EXPH are updated by the value written to the bit fields CURHS and EXPHS. |
| **0** | 6 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

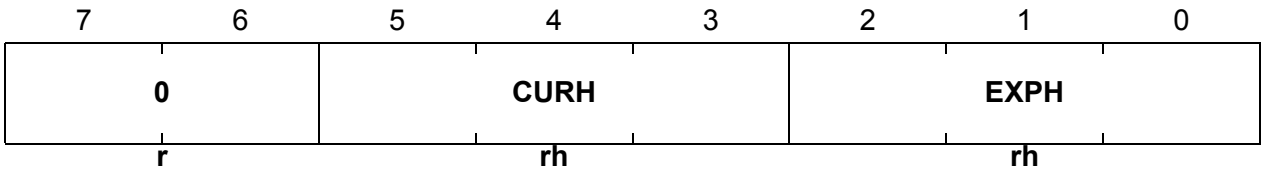
Register MCMOUT shows the multi-channel control bits that are currently used. Register MCMOUT is defined as follows:

**MCMOUTL**
**Multi-Channel Mode Output Register Low** **Reset Value: $00_H$**



| Field | Bits | Type | Description |
|---|---|---|---|
| **MCMP** | 5:0 | rh | **Multi-Channel PWM Pattern**<br>Bit field MCMP is written by a shadow transfer from bit field MCMPS. It contains the output pattern for the multi-channel mode. If this mode is enabled by bit MCMEN in register MODCTR, the output state of the following output signal can be modified:<br>Bit 0 multi-channel state for output CC60<br>Bit 1 multi-channel state for output COUT60<br>Bit 2 multi-channel state for output CC61<br>Bit 3 multi-channel state for output COUT61<br>Bit 4 multi-channel state for output CC62<br>Bit 5 multi-channel state for output COUT62<br>The multi-channel patterns can set the related output to the passive state.<br>0 The output is set to the passive state. The PWM generated by T12 or T13 is not taken into account.<br>1 The output can deliver the PWM generated by T12 or T13 (according to register MODCTR).<br>While IDLE = 1, bit field MCMP is cleared. |
| **R** | 6 | rh | **Reminder Flag**<br>This reminder flag indicates that the shadow transfer from bit field MCMPS to MCMP has been requested by the selected trigger source. This bit is cleared when the shadow transfer takes place and while MCMEN = 0.<br>0 Currently, no shadow transfer from MCMPS to MCMP is requested.<br>1 A shadow transfer from MCMPS to MCMP has been requested by the selected trigger source, but it has not yet been executed, because the selected synchronization condition has not yet occurred. |
| **0** | 7 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

**MCMOUTH**
**Multi-Channel Mode Output Register High** **Reset Value: $00_H$**

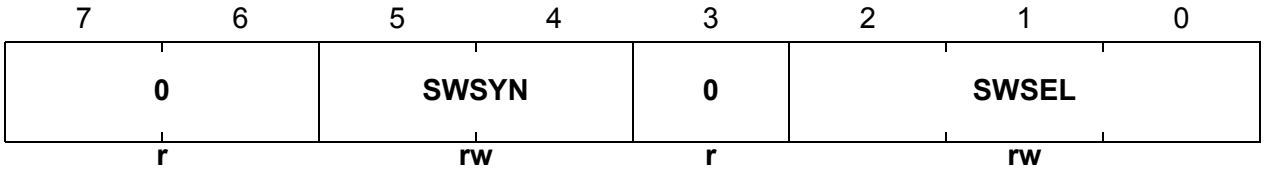



| Field | Bits | Type | Description |
|---|---|---|---|
| **EXPH** | 2:0 | rh | **Expected Hall Pattern**<br>Bit field EXPH is written by a shadow transfer from bit field EXPHS. The contents are compared after every detected edge at the hall input pins with the pattern at the hall input pins in order to detect the occurrence of the next desired (=expected) hall pattern or a wrong pattern.<br>If the current hall pattern at the hall input pins is equal to the bit field EXPH, bit CHE (correct hall event) is set and an interrupt request is generated (if enabled by bit ENCHE).<br>If the current hall pattern at the hall input pins is not equal to the bit fields CURH or EXPH, bit WHE (wrong hall event) is set and an interrupt request is generated (if enabled by bit ENWHE). |
| **CURH** | 5:3 | rh | **Current Hall Pattern**<br>Bit field CURH is written by a shadow transfer from bit field CURHS.The contents are compared after every detected edge at the hall input pins with the pattern at the hall input pins in order to detect the occurrence of the next desired (=expected) hall pattern or a wrong pattern.<br>If the current hall input pattern is equal to bit field CURH, the detected edge at the hall input pins has been an invalid transition (e.g. a spike). |
| **0** | 7:6 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

*Note: The bits in the bit fields EXPH and CURH correspond to the hall patterns at the input pins CCPOSx (x = 0, 1, 2) in the following order (EXPH.2, EXPH.1, EXPH.0), (CURH.2, CURH.1, CURH.0), (CCPOS2, CCPOS.1, CCPOS0).*

Register MCMCTR contains control bits for the multi-channel functionality.

**MCMCTR**
**Multi-Channel Mode Control Register** **Reset Value: $00_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **SWSEL** | 2:0 | rw | **Switching Selection**<br>Bit field SWSEL selects one of the following trigger request sources (next multi-channel event) for the shadow transfer 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.<br>000 no trigger request will be generated<br>001 correct hall pattern on CCPOSx detected<br>010 T13 period-match detected (while counting up)<br>011 T12 one-match (while counting down)<br>100 T12 channel 1 compare-match detected (phase delay function)<br>101 T12 period match detected (while counting up)<br>else reserved, no trigger request will be generated |

| Field | Bits | Type | Description |
|---|---|---|---|
| **SWSYN** | 5:4 | rw | **Switching Synchronization**<br>Bit field SWSYN triggers the shadow transfer between MCMPS and MCMP if it has been requested before (flag R set by an event selected by SWSEL). This feature permits the synchronization of the outputs to the PWM source, that is used for modulation (T12 or T13).<br>00 direct; the trigger event directly causes the shadow transfer<br>01 T13 zero-match triggers the shadow transfer<br>10 a T12 zero-match (while counting up) triggers the shadow transfer<br>11 reserved; no action |
| **0** | 3, 6, 7 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

*Note: The generation of the shadow transfer request by hardware is only enabled if bit MCMEN = 1.*

### 15.3.7 Interrupt Control Registers

**ISL**
**Capture/Compare Interrupt Status Register Low** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **T12 PM** | **T12 OM** | **ICC 62F** | **ICC 62R** | **ICC 61F** | **ICC 61R** | **ICC 60F** | **ICC 60R** |
| **rh** | **rh** | **rh** | **rh** | **rh** | **rh** | **rh** | **rh** |

| Field | Bits | Type | Description |
|---|---|---|---|
| **ICC6xR**<br>**(x = 0, 1, 2)** | 0, 2, 4 | rh | **Capture, Compare-Match Rising Edge Flag**<br>In compare mode, a compare-match has been detected while T12 was counting up. In capture mode, a rising edge has been detected at the input CC6x.<br>0 The event has not yet occurred since this bit has been reset for the last time.<br>1 The event described above has been detected. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **ICC6xF (x = 0, 1, 2)** | 1, 3, 5 | rh | **Capture, Compare-Match Falling Edge Flag**<br>In compare mode, a compare-match has been detected while T12 was counting down. In capture mode, a falling edge has been detected at the input CC6x.<br>0 The event has not yet occurred since this bit has been reset for the last time.<br>1 The event described above has been detected. |
| **T12OM** | 6 | rh | **Timer T12 One-Match Flag**<br>0 A timer T12 one-match (while counting down) has not yet been detected since this bit has been reset for the last time.<br>1 A timer T12 one-match (while counting down) has been detected. |
| **T12PM** | 7 | rh | **Timer T12 Period-Match Flag**<br>0 A timer T12 period-match (while counting up) has not yet been detected since this bit has been reset for the last time.<br>1 A timer T12 period-match (while counting up) has been detected. |

**ISH**
**Capture/Compare Interrupt Status Register High** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **STR** | **IDLE** | **WHE** | **CHE** | **TRPS** | **TRPF** | **T13PM** | **T13CM** |
| **rh** | **rh** | **rh** | **rh** | **rh** | **rh** | **rh** | **rh** |

| Field | Bits | Type | Description |
|---|---|---|---|
| **T13CM** | 0 | rh | **Timer T13 Compare-Match Flag**<br>0 A timer T13 compare-match has not yet been detected since this bit has been reset for the last time.<br>1 A timer T13 compare-match has been detected. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **T13PM** | 1 | rh | **Timer T13 Period-Match Flag**<br>0 A timer T13 period-match has not yet been detected since this bit has been reset for the last time.<br>1 A timer T13 period-match has been detected. |
| **TRPF** | 2 | rh | **Trap Flag**<br>The trap flag TRPF will be set by hardware if TRPPEN = 1 and CTRAP = 0 or by software. If TRPM2 = 0, bit TRPF is reset by hardware if the input CTRAP becomes inactive (TRPPEN = 1). If TRPM2 = 1, bit TRPF must be reset by software in order to leave the trap state.<br>0 The trap condition has not been detected.<br>1 The trap condition has been detected (input CTRAP has been 0 or by software). |
| **TRPS** | 3 | rh | **Trap State**<br>0 The trap state is not active.<br>1 The trap state is active. Bit TRPS is set while bit TRPF = 1. It is reset according to the mode selected in register TRPCTR.<br>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. Bit TRPS = 1 and TRPF = 0 can occur if the trap condition is no longer active but the selected synchronization has not yet taken place. |
| **CHE** | 4 | rh | **Correct Hall Event**<br>On every valid hall edge, the contents of EXPH are compared with the pattern on pin CCPOSx and if equal bit CHE is set.<br>0 A transition to a correct (=expected) hall event has not yet been detected since this bit has been reset for the last time.<br>1 A transition to a correct (=expected) hall event has been detected. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **WHE** | 5 | rh | **Wrong Hall Event**<br>On every valid hall edge, the contents of EXPH are compared with the pattern on pin CCPOSx. If both comparisons (CURH and EXPH with CCPOSx) are not true, bit WHE (wrong hall event) is set.<br>0 A transition to a wrong hall event (not the expected one) has not yet been detected since this bit has been reset for the last time.<br>1 A transition to a wrong hall event (not the expected one) has been detected. |
| **IDLE** | 6 | rh | **IDLE State**<br>This bit is set together with bit WHE (wrong hall event) and it must be reset by software.<br>0 No action.<br>1 Bit field MCMP is cleared and held to 0, the selected outputs are set to passive state. |
| **STR** | 7 | rh | **Multi-Channel Mode Shadow Transfer Request**<br>This bit is set when a shadow transfer from MCMOUTS to MCMOUT takes places in multi-channel mode.<br>0 The shadow transfer has not yet taken place.<br>1 The shadow transfer has taken place. |

*Note: Not all bits in register IS can generate an interrupt. Other status bits have been added, which have a similar structure for their set and reset actions.*

*Note: 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 hardware or software) for the corresponding bit in register IS.*

*Note: In compare mode (and hall mode), the timer-related interrupts are only generated while the timer is running (TxR = 1). In capture mode, the capture interrupts are also generated while the timer T12 is stopped.*

Register ISS contains the individual interrupt request set bits required to generate a CCU6 interrupt request by software.

**ISSL**

**Capture/Compare Interrupt Status Set Register Low** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **SST12PM** | **SST12OM** | **SCC62F** | **SCC62R** | **SCC61F** | **SCC61R** | **SCC60F** | **SCC60R** |
| **w** | **w** | **w** | **w** | **w** | **w** | **w** | **w** |

| Field | Bits | Type | Description |
|---|---|---|---|
| **SCC60R** | 0 | w | **Set Capture, Compare-Match Rising Edge Flag**<br>0 No action<br>1 Bit ICC60R in register IS will be set. |
| **SCC60F** | 1 | w | **Set Capture, Compare-Match Falling Edge Flag**<br>0 No action<br>1 Bit ICC60F in register IS will be set. |
| **SCC61R** | 2 | w | **Set Capture, Compare-Match Rising Edge Flag**<br>0 No action<br>1 Bit ICC61R in register IS will be set. |
| **SCC61F** | 3 | w | **Set Capture, Compare-Match Falling Edge Flag**<br>0 No action<br>1 Bit ICC61F in register IS will be set. |
| **SCC62R** | 4 | w | **Set Capture, Compare-Match Rising Edge Flag**<br>0 No action<br>1 Bit ICC62R in register IS will be set. |
| **SCC62F** | 5 | w | **Set Capture, Compare-Match Falling Edge Flag**<br>0 No action<br>1 Bit ICC62F in register IS will be set. |
| **ST12OM** | 6 | w | **Set Timer T12 One-Match Flag**<br>0 No action<br>1 Bit T12OM in register IS will be set. |
| **ST12PM** | 7 | w | **Set Timer T12 Period-Match Flag**<br>0 No action<br>1 Bit T12PM in register IS will be set. |

*Note: If the setting by hardware of the corresponding flags leads to an interrupt, the setting by software has the same effect.*

**ISSH**
**Capture/Compare Interrupt Status Set Register High** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **S STR** | **S IDLE** | **S WHE** | **S CHE** | **S WHC** | **S TRPF** | **S T13 PM** | **S T13 CM** |
| **w** | **w** | **w** | **w** | **w** | **w** | **w** | **w** |

| Field | Bits | Type | Description |
|---|---|---|---|
| **ST13CM** | 0 | w | **Set Timer T13 Compare-Match Flag**<br>0 No action<br>1 Bit T13CM in register IS will be set. |
| **ST13PM** | 1 | w | **Set Timer T13 Period-Match Flag**<br>0 No action<br>1 Bit T13PM in register IS will be set. |
| **STRPF** | 2 | w | **Set Trap Flag**<br>0 No action<br>1 Bits TRPF and TRPS in register IS will be set. |
| **SWHC** | 3 | w | **Software Hall Compare**<br>0 No action<br>1 The Hall compare action is triggered. |
| **SCHE** | 4 | w | **Set Correct Hall Event Flag**<br>0 No action<br>1 Bit CHE in register IS will be set. |
| **SWHE** | 5 | w | **Set Wrong Hall Event Flag**<br>0 No action<br>1 Bit WHE in register IS will be set. |
| **SIDLE** | 6 | w | **Set IDLE Flag**<br>0 No action<br>1 Bit IDLE in register IS will be set. |
| **SSTR** | 7 | w | **Set STR Flag**<br>0 No action<br>1 Bit STR in register IS will be set. |

Register ISR contains the individual interrupt request reset bits to reset the corresponding flags by software.

**ISRL**

**Capture/Compare Interrupt Status Reset Register Low** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| R<br>T12<br>PM | R<br>T12<br>OM | R<br>CC<br>62F | R<br>CC<br>62R | R<br>CC<br>61F | R<br>CC<br>61R | R<br>CC<br>60F | R<br>CC<br>60R |
| w | w | w | w | w | w | w | w |

| Field | Bits | Type | Description |
|---|---|---|---|
| **RCC60R** | 0 | w | **Reset Capture, Compare-Match Rising Edge Flag**<br>0 No action<br>1 Bit ICC60R in register IS will be reset. |
| **RCC60F** | 1 | w | **Reset Capture, Compare-Match Falling Edge Flag**<br>0 No action<br>1 Bit ICC60F in register IS will be reset. |
| **RCC61R** | 2 | w | **Reset Capture, Compare-Match Rising Edge Flag**<br>0 No action<br>1 Bit ICC61R in register IS will be reset. |
| **RCC61F** | 3 | w | **Reset Capture, Compare-Match Falling Edge Flag**<br>0 No action<br>1 Bit ICC61F in register IS will be reset. |
| **RCC62R** | 4 | w | **Reset Capture, Compare-Match Rising Edge Flag**<br>0 No action<br>1 Bit ICC62R in register IS will be reset. |
| **RCC62F** | 5 | w | **Reset Capture, Compare-Match Falling Edge Flag**<br>0 No action<br>1 Bit ICC62F in register IS will be reset. |
| **RT12OM** | 6 | w | **Reset Timer T12 One-Match Flag**<br>0 No action<br>1 Bit T12OM in register IS will be reset. |
| **RT12PM** | 7 | w | **Reset Timer T12 Period-Match Flag**<br>0 No action<br>1 Bit T12PM in register IS will be reset. |

**ISRH**

**Capture/Compare Interrupt Status Reset Register High** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **R STR** | **R IDLE** | **R WHE** | **R CHE** | **0** | **R TRPF** | **R T13 PM** | **R T13 CM** |
| **w** | **w** | **w** | **w** | **r** | **w** | **w** | **w** |

| Field | Bits | Type | Description |
|---|---|---|---|
| **RT13CM** | 0 | w | **Reset Timer T13 Compare-Match Flag**<br>0 No action<br>1 Bit T13CM in register IS will be reset. |
| **RT13PM** | 1 | w | **Reset Timer T13 Period-Match Flag**<br>0 No action<br>1 Bit T13PM in register IS will be reset. |
| **RTRPF** | 2 | w | **Reset Trap Flag**<br>0 No action<br>1 Bit TRPF in register IS will be reset (not taken into account while input $\overline{\text{CTRAP}}$ = 0 and TRPPEN = 1. |
| **RCHE** | 4 | w | **Reset Correct Hall Event Flag**<br>0 No action<br>1 Bit CHE in register IS will be reset. |
| **RWHE** | 5 | w | **Reset Wrong Hall Event Flag**<br>0 No action<br>1 Bit WHE in register IS will be reset. |
| **RIDLE** | 6 | w | **Reset IDLE Flag**<br>0 No action<br>1 Bit IDLE in register IS will be reset. |
| **RSTR** | 7 | w | **Reset STR Flag**<br>0 No action<br>1 Bit STR in register IS will be reset. |
| **0** | 3 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

## IENL
**Capture/Compare Interrupt Enable Register Low** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **ENT12PM** | **ENT12OM** | **ENCC62F** | **ENCC62R** | **ENCC61F** | **ENCC61R** | **ENCC60F** | **ENCC60R** |
| **rw** | **rw** | **rw** | **rw** | **rw** | **rw** | **rw** | **rw** |

| Field | Bits | Type | Description |
|---|---|---|---|
| **ENCC60R** | 0 | rw | **Capture, Compare-Match Rising Edge Interrupt Enable for Channel 0**<br>0 No interrupt will be generated if the set condition for bit ICC60R in register IS occurs.<br>1 An interrupt will be generated if the set condition for bit ICC60R in register IS occurs. The interrupt line that will be activated is selected by bit field INPCC60. |
| **ENCC60F** | 1 | rw | **Capture, Compare-Match Falling Edge Interrupt Enable for Channel 0**<br>0 No interrupt will be generated if the set condition for bit ICC60F in register IS occurs.<br>1 An interrupt will be generated if the set condition for bit ICC60F in register IS occurs. The interrupt line that will be activated is selected by bit field INPCC60. |
| **ENCC61R** | 2 | rw | **Capture, Compare-Match Rising Edge Interrupt Enable for Channel 1**<br>0 No interrupt will be generated if the set condition for bit ICC61R in register IS occurs.<br>1 An interrupt will be generated if the set condition for bit ICC61R in register IS occurs. The interrupt line that will be activated is selected by bit field INPCC61. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **ENCC61F** | 3 | rw | **Capture, Compare-Match Falling Edge Interrupt Enable for Channel 1**<br>0 No interrupt will be generated if the set condition for bit ICC61F in register IS occurs.<br>1 An interrupt will be generated if the set condition for bit ICC61F in register IS occurs. The interrupt line that will be activated is selected by bit field INPCC61. |
| **ENCC62R** | 4 | rw | **Capture, Compare-Match Rising Edge Interrupt Enable for Channel 2**<br>0 No interrupt will be generated if the set condition for bit ICC62R in register IS occurs.<br>1 An interrupt will be generated if the set condition for bit ICC62R in register IS occurs. The interrupt line that will be activated is selected by bit field INPCC62. |
| **ENCC62F** | 5 | rw | **Capture, Compare-Match Falling Edge Interrupt Enable for Channel 2**<br>0 No interrupt will be generated if the set condition for bit ICC62F in register IS occurs.<br>1 An interrupt will be generated if the set condition for bit ICC62F in register IS occurs. The interrupt line that will be activated is selected by bit field INPCC62. |
| **ENT12OM** | 6 | rw | **Enable Interrupt for T12 One-Match**<br>0 No interrupt will be generated if the set condition for bit T12OM in register IS occurs.<br>1 An interrupt will be generated if the set condition for bit T12OM in register IS occurs. The interrupt line that will be activated is selected by bit field INPT12. |
| **ENT12PM** | 7 | rw | **Enable Interrupt for T12 Period-Match**<br>0 No interrupt will be generated if the set condition for bit T12PM in register IS occurs.<br>1 An interrupt will be generated if the set condition for bit T12PM in register IS occurs. The interrupt line that will be activated is selected by bit field INPT12. |

## IENH
## Capture/Compare Interrupt Enable Register High

**Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **EN STR** | **EN IDLE** | **EN WHE** | **EN CHE** | **0** | **EN TRPF** | **EN T13 PM** | **EN T13 CM** |
| **rw** | **rw** | **rw** | **rw** | **r** | **rw** | **rw** | **rw** |

| Field | Bits | Type | Description |
|---|---|---|---|
| **ENT13CM** | 0 | rw | **Enable Interrupt for T13 Compare-Match**<br>0 No interrupt will be generated if the set condition for bit T13CM in register IS occurs.<br>1 An interrupt will be generated if the set condition for bit T13CM in register IS occurs. The interrupt line that will be activated is selected by bit field INPT13. |
| **ENT13PM** | 1 | rw | **Enable Interrupt for T13 Period-Match**<br>0 No interrupt will be generated if the set condition for bit T13PM in register IS occurs.<br>1 An interrupt will be generated if the set condition for bit T13PM in register IS occurs. The interrupt line that will be activated is selected by bit field INPT13. |
| **ENTRPF** | 2 | rw | **Enable Interrupt for Trap Flag**<br>0 No interrupt will be generated if the set condition for bit TRPF in register IS occurs.<br>1 An interrupt will be generated if the set condition for bit TRPF in register IS occurs. The interrupt line that will be activated is selected by bit field INPERR. |
| **ENCHE** | 4 | rw | **Enable Interrupt for Correct Hall Event**<br>0 No interrupt will be generated if the set condition for bit CHE in register IS occurs.<br>1 An interrupt will be generated if the set condition for bit CHE in register IS occurs. The interrupt line that will be activated is selected by bit field INPCHE. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **ENWHE** | 5 | rw | **Enable Interrupt for Wrong Hall Event**<br>0 No interrupt will be generated if the set condition for bit WHE in register IS occurs.<br>1 An interrupt will be generated if the set condition for bit WHE in register IS occurs. The interrupt line that will be activated is selected by bit field INPERR. |
| **ENIDLE** | 6 | rw | **Enable Idle**<br>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.<br>0 The bit IDLE is not automatically set when a wrong hall event is detected.<br>1 The bit IDLE is automatically set when a wrong hall event is detected. |
| **ENSTR** | 7 | rw | **Enable Multi-Channel Mode Shadow Transfer Interrupt**<br>0 No interrupt will be generated if the set condition for bit STR in register IS occurs.<br>1 An interrupt will be generated if the set condition for bit STR in register IS occurs. The interrupt line that will be activated is selected by bit field INPCHE. |
| **0** | 3 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

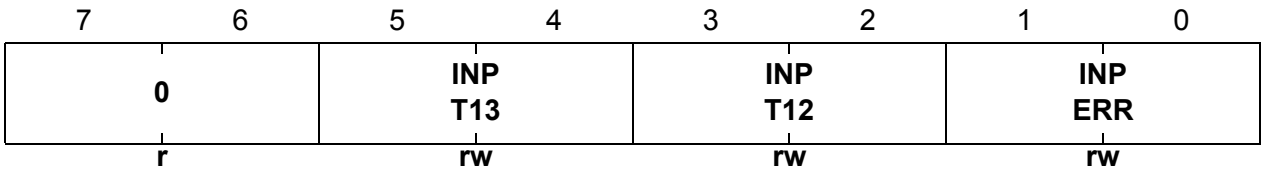
**INPL**

**Capture/Compare Interrupt Node Pointer Register Low** **Reset Value: $40_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **INPCHE** | | **INPCC62** | | **INPCC61** | | **INPCC60** | |
| **rw** | | **rw** | | **rw** | | **rw** | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **INPCC60** | 1:0 | rw | **Interrupt Node Pointer for Channel 0 Interrupts**<br>This bit field defines the interrupt output line, which is activated due to a set condition for bit ICC60R (if enabled by bit ENCC60R) or for bit ICC60F (if enabled by bit ENCC60F).<br>00 Interrupt output line SR0 is selected.<br>01 Interrupt output line SR1 is selected.<br>10 Interrupt output line SR2 is selected.<br>11 Interrupt output line SR3 is selected. |
| **INPCC61** | 3:2 | rw | **Interrupt Node Pointer for Channel 1 Interrupts**<br>This bit field defines the interrupt output line, which is activated due to a set condition for bit ICC61R (if enabled by bit ENCC61R) or for bit ICC61F (if enabled by bit ENCC61F).<br>00 Interrupt output line SR0 is selected.<br>01 Interrupt output line SR1 is selected.<br>10 Interrupt output line SR2 is selected.<br>11 Interrupt output line SR3 is selected. |
| **INPCC62** | 5:4 | rw | **Interrupt Node Pointer for Channel 2 Interrupts**<br>This bit field defines the interrupt output line, which is activated due to a set condition for bit ICC62R (if enabled by bit ENCC62R) or for bit ICC62F (if enabled by bit ENCC62F).<br>00 Interrupt output line SR0 is selected.<br>01 Interrupt output line SR1 is selected.<br>10 Interrupt output line SR2 is selected.<br>11 Interrupt output line SR3 is selected. |
| **INPCHE** | 7:6 | rw | **Interrupt Node Pointer for the CHE Interrupt**<br>This bit field defines the interrupt output line, which is activated due to a set condition for bit CHE (if enabled by bit ENCHE) or for bit STR (if enabled by bit ENSTR).<br>00 Interrupt output line SR0 is selected.<br>01 Interrupt output line SR1 is selected.<br>10 Interrupt output line SR2 is selected.<br>11 Interrupt output line SR3 is selected. |

**INPH**
**Capture/Compare Interrupt Node Pointer Register High** **Reset Value: $39_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **INPERR** | 1:0 | rw | **Interrupt Node Pointer for Error Interrupts**<br>This bit field defines the interrupt output line, which is activated due to a set condition for bit TRPF (if enabled by bit ENTRPF) or for bit WHE (if enabled by bit ENWHE).<br>00 Interrupt output line SR0 is selected.<br>01 Interrupt output line SR1 is selected.<br>10 Interrupt output line SR2 is selected.<br>11 Interrupt output line SR3 is selected. |
| **INPT12** | 3:2 | rw | **Interrupt Node Pointer for Timer T12 Interrupts**<br>This bit field defines the interrupt output line, which is activated due to a set condition for bit T12OM (if enabled by bit ENT12OM) or for bit T12PM (if enabled by bit ENT12PM).<br>00 Interrupt output line SR0 is selected.<br>01 Interrupt output line SR1 is selected.<br>10 Interrupt output line SR2 is selected.<br>11 Interrupt output line SR3 is selected. |
| **INPT13** | 5:4 | rw | **Interrupt Node Pointer for Timer T13 Interrupts**<br>This bit field defines the interrupt output line, which is activated due to a set condition for bit T13CM (if enabled by bit ENT13CM) or for bit T13PM (if enabled by bit ENT13PM).<br>00 Interrupt output line SR0 is selected.<br>01 Interrupt output line SR1 is selected.<br>10 Interrupt output line SR2 is selected.<br>11 Interrupt output line SR3 is selected. |
| **0** | 7:6 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

# 16 Controller Area Network (MultiCAN) Controller

The MultiCAN module contains 2 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 V2.0 B active. Each CAN node can receive and transmit standard frames with 11-bit identifiers as well as extended frames with 29-bit identifiers.

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

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

The bit timings for the CAN nodes are derived from the module clock ($f_{CAN}$) and are programmable up to a data rate of 1 Mbit/s. External bus transceivers are connected with a CAN node via a pair of receive and transmit pins.

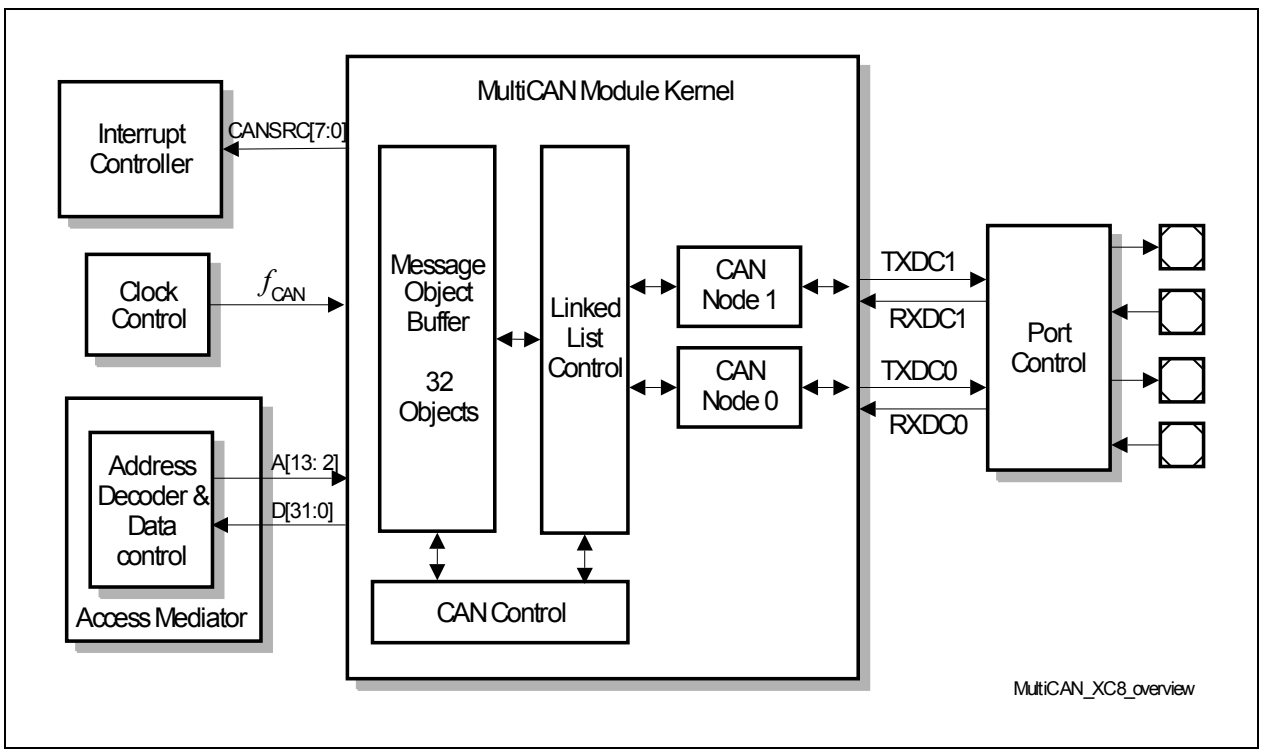



**Figure 16-1 Overview of the MultiCAN Module**

## Features

- Compliant with ISO 11898
- CAN functionality according to CAN specification V2.0 B active
- Dedicated control registers for each CAN node
- Data transfer rates up to 1 Mbit/s
- Flexible and powerful message transfer control and error handling capabilities
- Advanced CAN bus bit timing analysis and baud rate detection for each CAN node via a frame counter
- Full-CAN functionality: A set of 32 message objects can be individually
  - Allocated (assigned) to any CAN node
  - Configured as transmit or receive object
  - Set up to handle frames with 11-bit or 29-bit identifier
  - Identified by a timestamp via a frame counter
  - Configured to remote monitoring mode
- Advanced acceptance filtering
  - Each message object provides an individual acceptance mask to filter incoming frames
  - A message object can be configured to accept standard or extended frames or to accept both standard and extended frames
  - Message objects can be grouped into four priority classes for transmission and reception
  - The selection of the message to be transmitted first can be based on frame identifier, IDE bit and RTR bit according to CAN arbitration rules, or according to its order in the list
- Advanced message object functionality
  - Message objects can be combined to build FIFO message buffers of arbitrary size, limited only by the total number of message objects
  - Message objects can be linked to form a gateway that automatically transfers frames between two different CAN buses. A single gateway can link any two CAN nodes. An arbitrary number of gateways can be defined.
- Advanced data management
  - The message objects are organized in double-chained lists
  - List reorganizations can be performed at any time, even during full operation of the CAN nodes
  - A powerful, command-driven list controller manages the organization of the list structure and ensures consistency of the list
  - Message FIFOs are based on the list structure and can easily be scaled in size during CAN operation
  - Static allocation commands offer compatibility with TwinCAN applications that are not list-based
- Advanced interrupt handling

– Up to 8 interrupt output lines are available. Interrupt requests can be individually routed to one of the 8 interrupt output lines
– Message post-processing notifications can be combined flexibly into a dedicated register field of 64 notification bits

## 16.1 MultiCAN Kernel Functional Description

This section describes the functionality of the MultiCAN module.

### 16.1.1 Module Structure

**Figure 16-2** shows the general structure of the MultiCAN module.

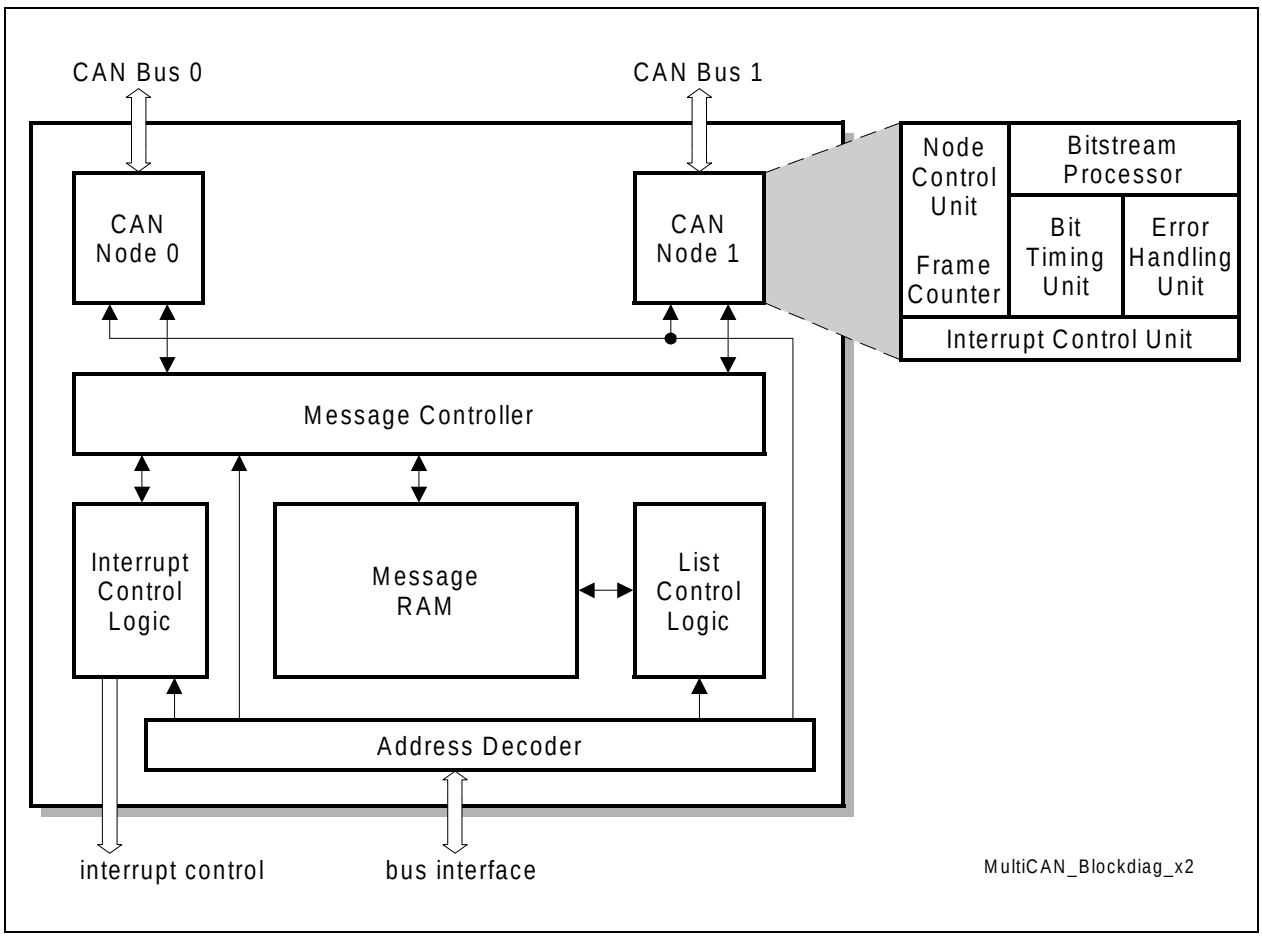



**Figure 16-2 MultiCAN Block Diagram**

#### CAN Nodes

Each CAN node consists of several sub-units.

- **Bitstream Processor**
  The Bitstream Processor performs data, remote, error and overload frame processing according to the ISO 11898 standard. This includes conversion between the serial data stream and the input/output registers.
- **Bit Timing Unit**
  The Bit Timing Unit 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 re-synchronization.

- **Error Handling Unit**
  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.
- **Node Control Unit**
  The Node Control Unit coordinates the operation of the CAN node:
  – Enable/disable CAN transfer of the node
  – Enable/disable and generate node-specific events that lead to an interrupt request (CAN bus errors, successful frame transfers etc.)
  – Administration of the Frame Counter
- **Interrupt Control Unit**
  The Interrupt Control Unit in the CAN node controls the interrupt generation for the different conditions that can occur in the CAN node.

### Message Controller

The Message Controller handles the exchange of CAN frames between the CAN nodes and the message objects that are stored in the Message RAM. The Message Controller performs several functions:

- Receive acceptance filtering to determine the correct message object for storing of a received CAN frame
- Transmit acceptance filtering to determine the message object to be transmitted first, individually for each CAN node
- Transfer contents between message objects and the CAN nodes, taking into account the status/control bits of the message objects
- Handling of the FIFO buffering and gateway functionality
- Aggregation of message-pending notification bits

### List Controller

The List Controller performs all operations that lead to a modification of the double-chained message object lists. Only the list controller is allowed to modify the list structure. The allocation/deallocation or reallocation of a message object can be requested via a user command interface (command panel). The list controller state machine then performs the requested command autonomously.

### Interrupt Control

The general interrupt structure is shown in **Figure 16-3**. 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 software by writing a 0 to it.

If enabled by the related interrupt enable bit in the interrupt enable register, an interrupt pulse can be generated at one of the 8 interrupt output lines CANSRCm of the MultiCAN

module. If more than one interrupt source is connected to the same interrupt node pointer (in the interrupt node pointer register), the requests are combined to one common line.

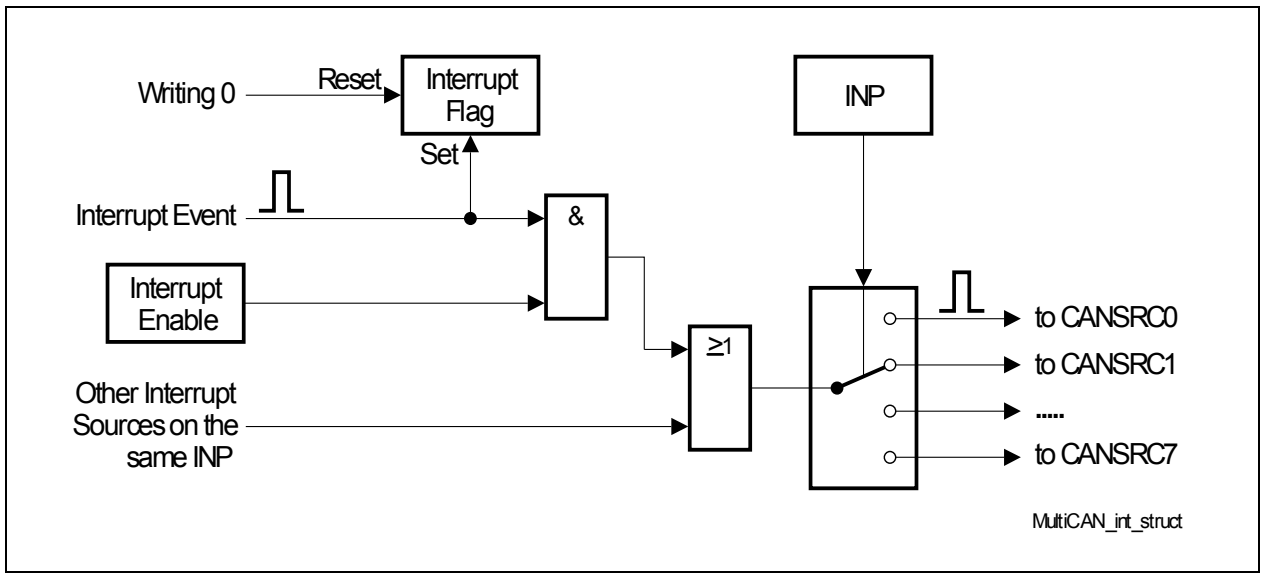



**Figure 16-3 General Interrupt Structure**

### 16.1.2 Clock Control

Table 16-1 indicates the minimum operating frequencies in MHz for $f_{CAN}$ that are required for a baud rate of 1 Mbit/s for the active CAN nodes. 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).

The values imply that the CPU executes maximum access to the MultiCAN module. The values may contain rounding effects.

**Table 16-1 Minimum Operating Frequencies [MHz]**

| Number of Allocated Message Objects[1)] | with 1 CAN Node Active | with 2 CAN Nodes Active |
|---|---|---|
| **16 Message Objects** | 12 | 19 |
| **32 Message Objects** | 15 | 23 |

1) Only those message objects that are allocated to a CAN node must be taken into account. The unallocated message objects have no influence on the minimum operating frequency.

### 16.1.3 CAN Node Control

Each CAN node may be configured and run independently from the other CAN nodes. Each CAN node is equipped with an individual set of SFR registers to control and to monitor the CAN node.

*Note: In the following descriptions, index "x" stands for the node number and index "n" represents the message object number.*

#### 16.1.3.1 Bit Timing Unit

According to the ISO 11898 standard, a CAN bit time is subdivided into different segments (**Figure 16-4**). Each segment consists of multiples of a time quantum $t_q$. The magnitude of $t_q$ is adjusted by bit fields NBTRx.BRP and NBTRx.DIV8, both controlling the baud rate prescaler. The baud rate prescaler is driven by the module clock $f_{CAN}$.

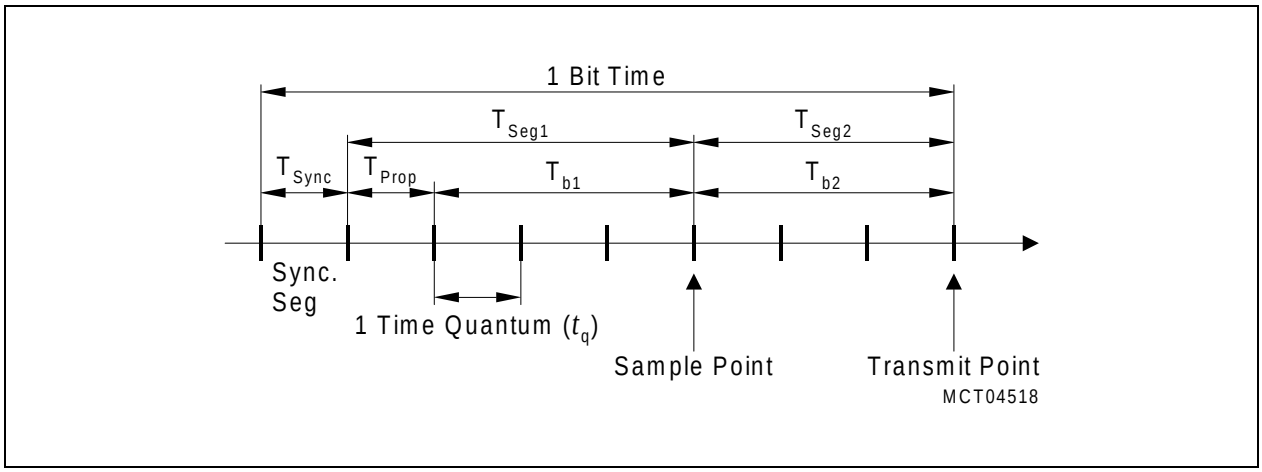



**Figure 16-4 CAN Bus Bit Timing Standard**

The Synchronization Segment ($T_{Sync}$) allows a phase synchronization between transmitter and receiver time base. The Synchronization Segment length is always one $t_q$. The Propagation Time Segment ($T_{Prop}$) takes into account the physical propagation delay in the transmitter output driver on the CAN bus line and in the transceiver circuit. For a working collision detection mechanism, $T_{Prop}$ must be two times the sum of all propagation delay quantities rounded up to a multiple of $t_q$. The phase buffer segments 1 and 2 ($T_{b1}$, $T_{b2}$) before and after the signal sample point are used to compensate for a mismatch between transmitter and receiver clock phases detected in the synchronization segment.

The maximum number of time quanta allowed for re-synchronization is defined by bit field NBTRx.SJW. The Propagation Time Segment and the Phase Buffer Segment 1 are combined to parameter $T_{Seg1}$, which is defined by the value NBTRx.TSEG1. A minimum of 3 time quanta is requested by the ISO standard. Parameter $T_{Seg2}$, which is defined by the value of NBTRx.TSEG2, covers the Phase Buffer Segment 2. A minimum of 2 time

quanta is requested by the ISO standard. According to ISO standard, a CAN bit time, calculated as the sum of $T_{Sync}$, $T_{Seg1}$ and $T_{Seg2}$, must not fall below 8 time quanta.

Calculation of the bit time:

| | | |
|---|---|---|
| $t_q$ | = (BRP + 1) / $f_{CAN}$ | if DIV8 = 0 |
| | = 8 ×(BRP+1) / $f_{CAN}$ | if DIV8 = 1 |
| $T_{Sync}$ | = 1 × $t_q$ | |
| $T_{Seg1}$ | = (TSEG1 + 1) × $t_q$ | (min. 3 $t_q$) |
| $T_{Seg2}$ | = (TSEG2 + 1) × $t_q$ | (min. 2 $t_q$) |
| bit time | = $T_{Sync}$ + $T_{Seg1}$ + $T_{Seg2}$ | (min. 8 $t_q$) |

To compensate phase shifts between clocks of different CAN controllers, the CAN controller must synchronize on any edge from the recessive to the dominant bus level. If the hard synchronization is enabled (at the start of frame), the bit time is restarted at the synchronization segment. Otherwise, the re-synchronization jump width $T_{SJW}$ defines the maximum number of time quanta, a bit time may be shortened or lengthened by one re-synchronization. The value of SJW is defined by bit field NBTRx.SJW.

| | |
|---|---|
| $T_{SJW}$ | = (SJW + 1) × $t_q$ |
| $T_{Seg1}$ | ≥ $T_{SJW}$ + $T_{prop}$ |
| $T_{Seg2}$ | ≥ $T_{SJW}$ |

The maximum relative tolerance for $f_{CAN}$ depends on the Phase Buffer Segments and the re-synchronization jump width.

| | | |
|---|---|---|
| d$f_{CAN}$ | ≤ min ($T_{b1}$, $T_{b2}$) / 2 × (13 × bit time - $T_{b2}$) | AND |
| d$f_{CAN}$ | ≤ $T_{SJW}$ / 20 × bit time | |

A valid CAN bit timing must be written to the register NBTR before resetting the bit NCRx.INIT, i.e., before enabling the operation of the CAN node. The register NBTRx may be written only if bit NCRx.CCE (Configuration Change Enable) is set.

### 16.1.3.2 Bitstream Processor

Based on the message objects in the message buffer, the Bit Stream Processor generates the remote and data frames to be transmitted via the CAN bus. It controls the CRC generator and adds the checksum information to the new remote or data frame. After including the ‘Start of Frame Bit’ and the ‘End of Frame Field’, the Bit Stream

Processor starts the CAN bus arbitration procedure and continues with the frame transmission when the bus was found in idle state. While the data transmission is running, the Bit Stream Processor monitors continuously the I/O line. If (outside the CAN bus arbitration phase or the acknowledge slot) a mismatch is detected between the voltage level on the I/O line and the logic state of the bit currently sent out by the transmit shift register, a ‘Last Error’ interrupt request is generated and the error code is indicated by the bit field NSRx.LEC.

The data consistency of an incoming frame is verified by checking the associated CRC field. When an error has been detected, the ‘Last Error’ interrupt request is generated and the error code is indicated by the bit field NSRx.LEC. Furthermore, an error frame is generated and transmitted on the CAN bus. After decomposing a faultless frame into identifier and data portion, the received information is transferred to the message buffer executing remote and data frame handling, interrupt generation and status processing.

### 16.1.3.3 Error Handling Unit

The Error Handling Unit of a CAN node x is responsible for the fault confinement of the CAN device. Its two counters, the Receive Error Counter NECNTx.REC and the Transmit Error Counter NECNTx.TEC 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 for the respective CAN node x in register NECNTx. 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. The CAN node is in error passive state, if at least one of the error counters is equal or greater than 128.

The “bus-off” state is activated if the Transmit Error Counter is equal or greater than the “bus-off” limit of 256. This state is reported by flag NSRx.BOFF. The device remains in this state, until the “bus-off” recovery sequence is finished. Additionally, bit NSRx.EWRN is set when at least one of the error counters is equal or greater than the error warning limit defined by bit field NECNTx.EWRNLVL. Bit NSRx.EWRN is reset if both error counters fall below the error warning limit again.

### 16.1.3.4 CAN Frame Counter

Each CAN node is equipped with a frame counter which enables the counting of transmitted/received CAN frames or helps obtain information on the time instant when a frame has started to transmit or received by the CAN node. CAN frame counting/bit time counting is performed by a 16-bit counter which is controlled by register NFCRx. Bit field NFCRx.CFSEL 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 stored to the bit field NFCRx.CFC and copied to the bit field MOIPRn.CFCVAL 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 bit field NFCRx.CFC. After the successful transfer of the frame, the captured value is copied to the bit field MOIPRn.CFCVAL of the message object involved in the transfer.
- **Bit Timing Mode:**
  Used for baud rate detection and analysis of the bit timing (**Chapter 16.1.5.3**).

### 16.1.3.5 CAN Node Interrupts

Each CAN node is equipped with four interrupt sources to generate an interrupt request upon:

- the successful transmission/reception of a frame
- a CAN protocol error with a last error code
- an alert condition occurs: transmit/receive error counters reach the warning limit, bus-off state changes, a list length error occurs, or a list object error occurs
- an overflow of the frame counter

Besides the hardware generated interrupts, software initiated interrupts can be generated using the register MITR. Writing a 1 to bit n of bit field MITR.IT generates an interrupt request signal on the corresponding interrupt output line CANSRCm. When writing MITR.IT more than one bit can be set resulting in the activation of multiple CANSRCm interrupt output lines at the same time.

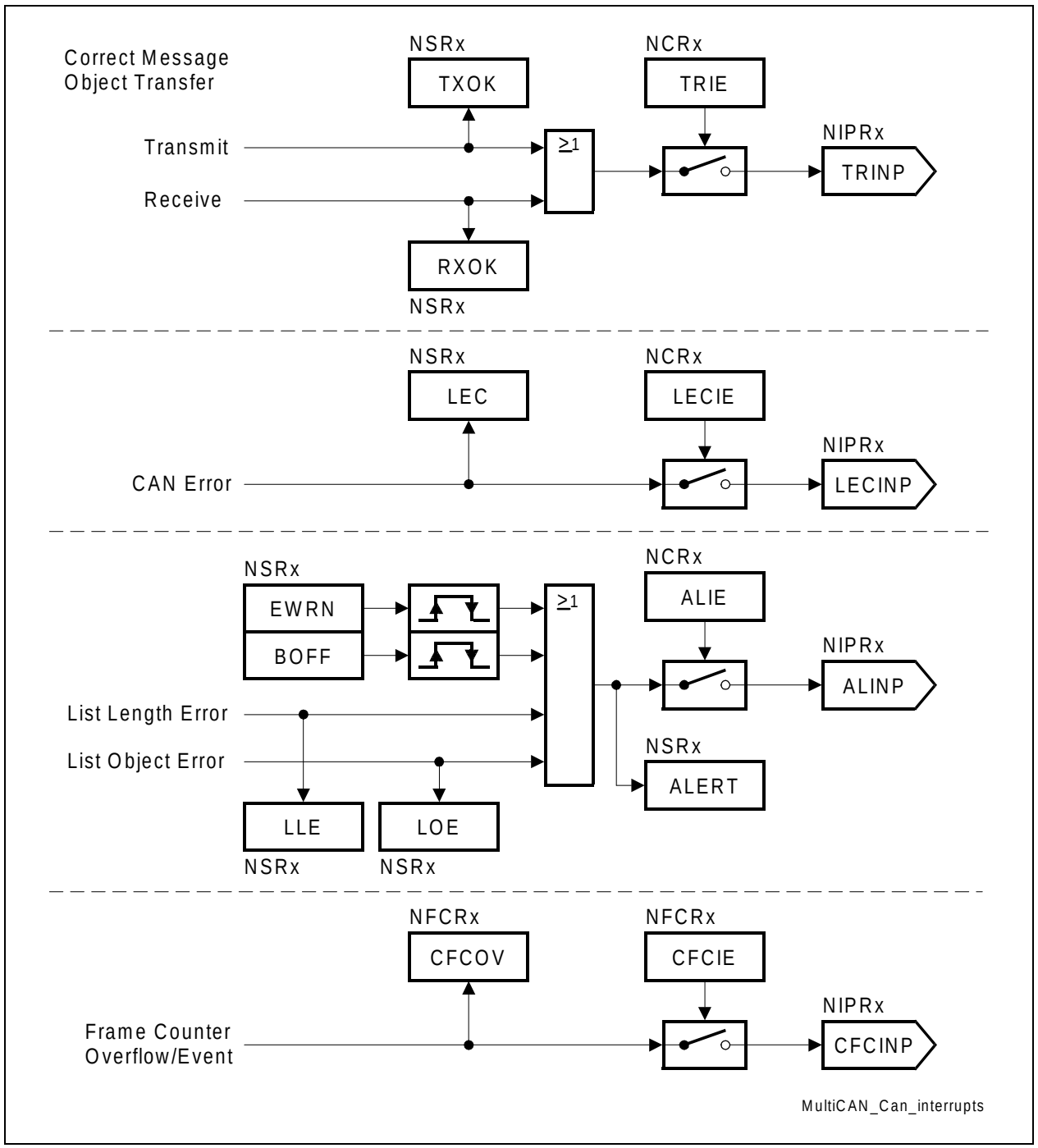



**Figure 16-5 CAN Node Interrupts**

### 16.1.4 Message Object List Structure

This section describes the structure of the message object lists in the MultiCAN module.

#### 16.1.4.1 Basics

The message objects of the MultiCAN module are organized in double-chained lists, where each message object has a pointer to the previous message object in the list as well as a pointer to the next message object in the list. The MultiCAN module provides eight lists. Each message object is allocated to one of these lists. In the example in **Figure 16-6**, the three message objects (3, 5, and 16) are allocated to the list with index 2 (List Register LIST2).

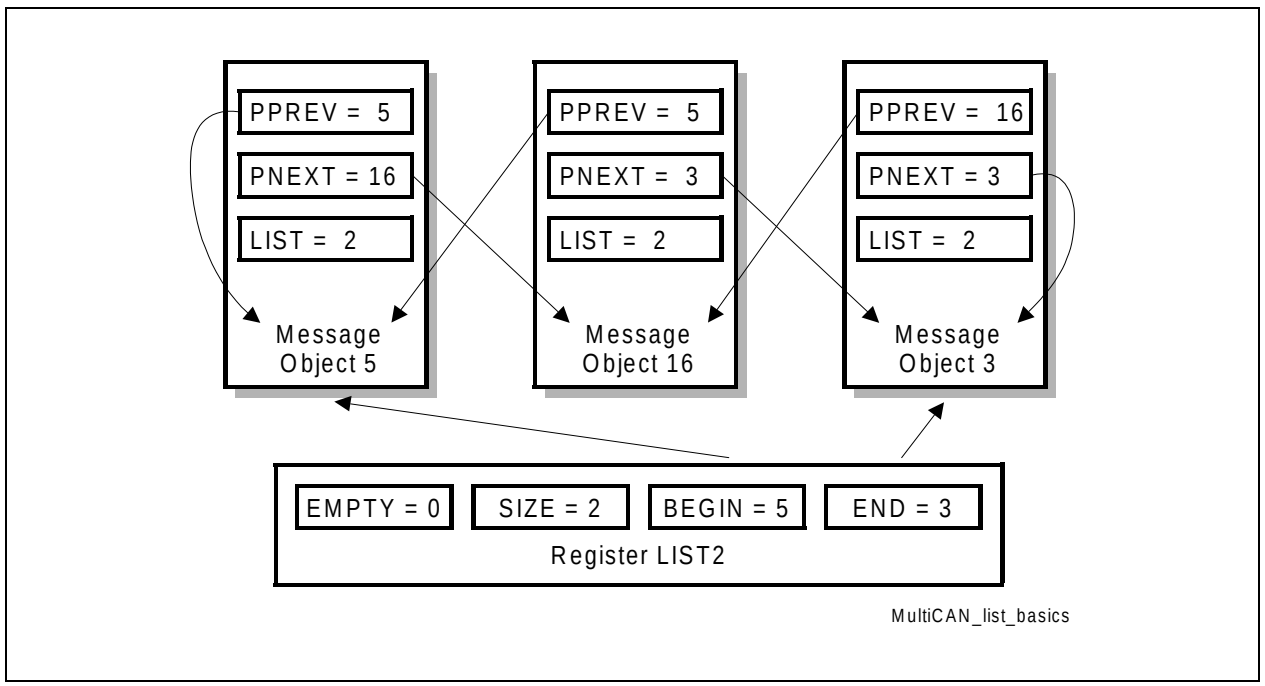



**Figure 16-6 Example Allocation of Message Objects to a List**

Bit field LIST.BEGIN points to the first element in the list (object 5 in the example), and bit field LIST.END points to the last element in the list (object 3 in the example). The number of elements in the list is indicated by bit field LIST.SIZE (SIZE = number of list elements - 1, thus SIZE = 2 for the 3 elements in the example). The bit LIST.EMPTY indicates whether a list is empty or not (EMPTY = 0 in the example, because list 2 is not empty).

Each message object n has a pointer MOCTRn.PNEXT that points to the next message object in the list and a pointer MOCTRn.PPREV that points to the previous message object in the list. PPREV of the first message object points to the message object itself because the first message object has no predecessor (in the example message object 5 is the first message object in the list, indicated by PPREV = 5). PNEXT of the last message object also points to the message object itself because the last message object

has no successor (in the example object 3 is the last message object in the list, indicated by PNEXT = 3).

Bit field MOCTRn.LIST indicates the list index number to which the message object is currently allocated. The message object of the example are allocated to list 2. Therefore, all LIST bit fields for the message objects assigned to list 2 are set to LIST = 2.

### 16.1.4.2 List of Unallocated Elements

The list with list index 0 has a special meaning: it is the list of all unallocated elements. An element is called unallocated if it belongs to list 0 (MOCTRn.LIST = 0). It is called allocated if it belongs to a list with an index not equal to 0 (MOCTRn.LIST > 0).

After reset, all message objects are unallocated. This means that they are assigned to the list of unallocated elements with MOCTRn.LIST = 0. After this initial allocation of the message objects caused by reset, the list of all unallocated message objects is ordered by message number (predecessor of message object n is object n-1, successor of object n is object n+1).

### 16.1.4.3 Connection to the CAN Nodes

Each CAN node is linked to one unique list of message objects. A CAN node performs message transfer only with the message objects that are allocated to the list of the CAN node. This is illustrated in **Figure 16-7**. Frames that are received on a CAN node may only be stored in one of the message objects that belongs to the CAN node; frames to be transmitted on a CAN node are selected only from the message objects that are allocated to that node, as indicated by the vertical arrows.

There are more lists (eight) than CAN nodes (two). This means that some lists are not linked to one of the CAN nodes. A message object that is allocated to one of these unlinked lists cannot receive messages directly from a CAN node and it may not transmit messages.

FIFO and gateway mechanisms refer to message object numbers and not directly to a specific list. The user must take care that the message objects targeted by FIFO/gateway belong to the desired list. The mechanisms allow working with lists that do not belong to this CAN node.

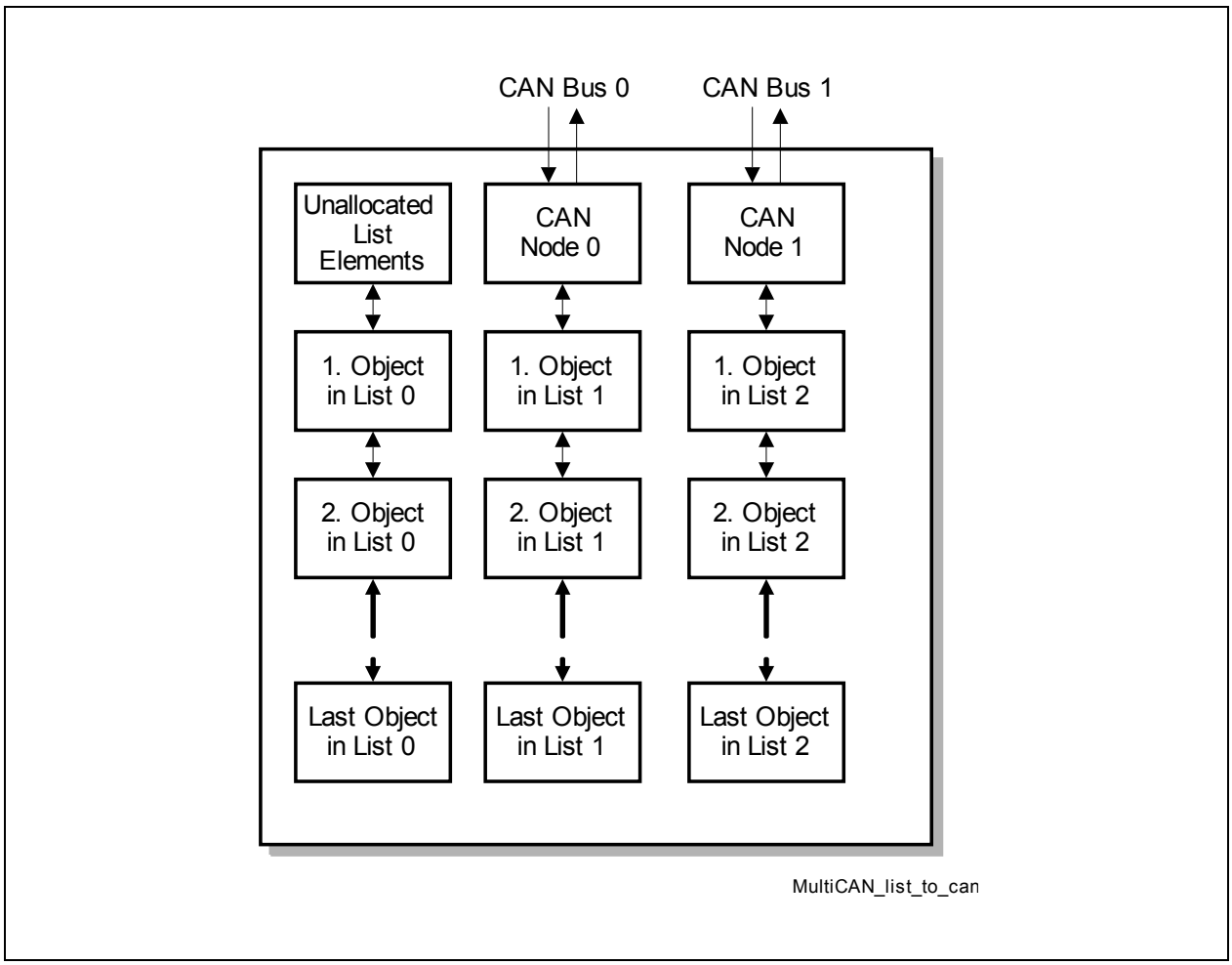



**Figure 16-7 Message Objects Linked to CAN Nodes**

### 16.1.4.4 List Command Panel

The list structure cannot be modified directly by means of write accesses to the LIST registers and the PPREV, PNEXT and LIST bit fields in the register MOSTATn as they are read-only. The management of the list structure is performed by and limited to the list controller inside the MultiCAN module. The list controller is controlled via a command panel allowing the user to issue list allocation commands to the list controller. The list controller basically serves two purposes:

1. Ensure that all operations that modify the list structure result in a consistent list structure.
2. Present flexibility to the user.

The list controller and the associated command panel allows the programmer to concentrate on the final properties of the list, which are characterized by the allocation of message objects to a CAN node, and the ordering relation between objects that are allocated to the same list. The process of list (re-)building is done in the list controller.

Table 16-2 gives an overview on the available panel commands while Table 16-7 describes the panel commands in more detail.

**Table 16-2 Panel Commands Overview**

| Command Name | Description |
|---|---|
| **No Operation** | No new command is started. |
| **Initialize Lists** | Run the initialization sequence to reset the CTRL and LIST field of all message objects. |
| **Static Allocate** | Allocate message object to a list. |
| **Dynamic Allocate** | Allocate the first message object of the list of unallocated objects to the selected list. |
| **Static Insert Before** | Remove a message object (source object) from the list that it currently belongs to, and insert it before a given destination object into the list structure of the destination object. |
| **Dynamic Insert Before** | Insert a new message object before a given destination object. |
| **Static Insert Behind** | Remove a message object (source object) from the list that it currently belongs to, and insert it behind a given destination object into the list structure of the destination object. |
| **Dynamic Insert Behind** | Insert a new message object behind a given destination object. |

A panel command is started by writing the respective command code to the bit field PANCTR.PANCMD. The corresponding command arguments must be written to bit fields PANCTR.PANAR1 and PANCTR.PANAR2 before writing the command code or together with the command code in a single 32-bit write access to the PANCTR Register.

With the write operation of a valid command code, the PANCTR.BUSY flag is set and further write accesses to the Panel Control Register are ignored. The BUSY flag remains active and the control panel remains locked until the execution of the requested command has been completed. After a reset, the list controller builds up list 0. During this operation, BUSY is set and other accesses to the CAN RAM are forbidden. The CAN RAM can be accessed again when BUSY becomes inactive.

*Note: The CAN RAM is automatically initialized after reset by the list controller in order to ensure correct list pointers in each message object. The end of this CAN RAM initialization is indicated by bit PANCTR.BUSY becoming inactive.*

In case of a dynamic allocation command that takes an element from the list of unallocated objects, the PANCTR.RBUSY bit becomes set together with the BUSY bit (RBUSY = BUSY = 1). This indicates that bit fields PANCTR.PANAR1 and PANCTR.PANAR2 are going to be updated by the list controller in the following way:

1. The message number of the message object taken from the list of unallocated elements is written to PANAR1.
2. If ERR (bit 7 of PANAR2) is set to 1, the list of unallocated elements was empty and the command is aborted. If ERR is 0, the list was not empty and the command will be performed successfully.

The results of a dynamic allocation command are written before the list controller starts the actual allocation process. As soon as the results are available, RBUSY becomes inactive (RBUSY = 0) again, while BUSY still remains active until completion of the command. This allows the user to set up the new message object while it is still in the process of list allocation. The access to message objects is not limited during ongoing list operations. However, any access to a register resource located inside the RAM delays the ongoing allocation process by one access cycle.

As soon as the command is finished, the BUSY flag becomes inactive (BUSY = 0) and write accesses to the Panel Control Register are enabled again. Additionally, the “No Operation” command code is automatically written to the bit field PANCTR.PANCMD. A new command may be started any time when BUSY = 0.

All fields of the register PANCTR except BUSY and RBUSY may be written by the user. This allows the register PANCTR to be saved and restored if the Command Panel is 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 memory location before issuing a command. At the end of the interrupt service routine, it should restore PANCTR from the memory location.

Before a message object that is allocated to the list of an active CAN node is moved to another list or to another position within the same list, bit MOCTRn.MSGVAL (“Message Valid”) of message object n must be cleared.

### 16.1.5 CAN Node Analysis Features

This section describes the CAN node analysis capabilities of the MultiCAN module.

#### 16.1.5.1 Analyze Mode

The CAN analyze mode allows the CAN traffic to be monitored without affecting the logical state of the CAN bus. The CAN analyze mode is selected by setting bit NCRx.CALM.

In CAN analyze mode, the transmit pin of a CAN node is held on recessive level permanently. The CAN node may receive frames (data, remote, and error frames) but is not allowed to transmit. Received data/remote frames are not acknowledged (i.e., acknowledge slot is sent recessive) but will be received and stored in matching message objects as long as there is any other node that acknowledges the frame. The complete message object functionality is available but no transmit request will be executed.

#### 16.1.5.2 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. With the switch, each CAN node can be connected either to the internal CAN bus (loop-back mode activated) or the external CAN bus, respectively to its transmit or receive pin (normal operation). The CAN bus which is currently not selected is driven recessive, this means the transmit pin is held at 1 and the receive pin is ignored by the CAN nodes that are in loop-back mode.

The loop-back mode is selected by setting bit NPCRx.LBM. All CAN nodes that are in loop-back mode may communicate together via the internal CAN bus without affecting the normal operation of the other CAN nodes that are not in loop-back mode.

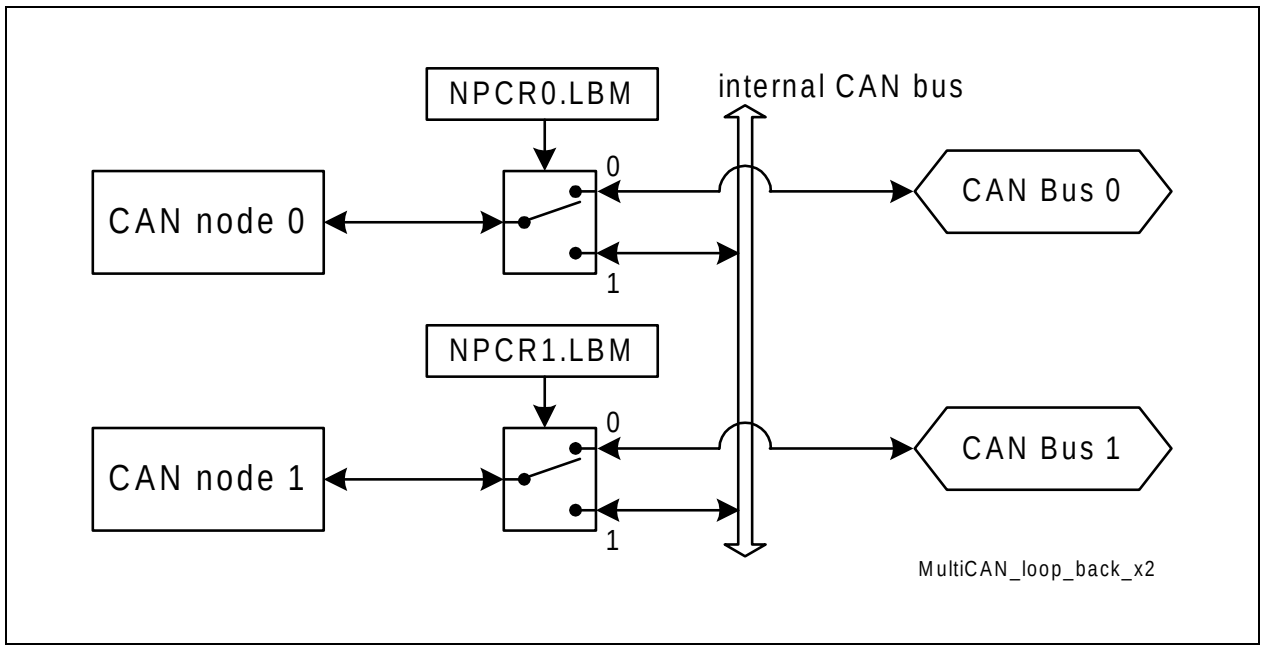



**Figure 16-8 Loop-Back Mode**

### 16.1.5.3 Bit Timing Analysis

Detailed analysis of the bit timing can be performed for each CAN node using the analysis modes of the CAN frame counter. The bit timing analysis functionality of the frame counter may be used for automatic detection of the CAN baud rate as well as for the analysis of the timing of the CAN network.

Bit timing analysis is selected by NFCRx.CFMOD = $10_B$. Bit timing analysis does not affect the operation of the CAN node. The bit timing measurement results are written into the NFCRx.CFC bit field. Whenever NFCRx.CFC is updated in bit timing analysis mode, the bit NFCRx.CFCOV is set to indicate the CFC update event. If NFCRx.CFCIE is set, an interrupt request can be generated (see **Figure 16-5**).

#### Automatic Baud Rate Detection

For automatic baud rate detection, the time between the observation of subsequent dominant edges on the CAN bus must be measured. This measurement is automatically performed if bit field NFCRx.CFSEL = $000_B$. With each dominant edge monitored on the CAN receive input line, the time (measured in $f_{CAN}$ clock cycles) between this edge and the most recent dominant edge is stored in the NFCRx.CFC bit field.

### Synchronization Analysis

The bit time synchronization is monitored if NFCRx.CFSEL = $010_B$. The time between the first dominant edge and the sample point is measured and stored in the NFCRx.CFC bit field. The bit timing synchronization offset may be derived from this time as the first edge after the sample point triggers synchronization and there is only one synchronization between consecutive sample points.

Synchronization analysis can be used, for example, for fine tuning of the baud rate during reception of the first CAN frame with the measured baud rate.

### Driver Delay Measurement

The delay between a transmitted edge and the corresponding received edge is measured when NFCRx.CFSEL = $011_B$ (dominant to dominant) and NFCRx.CFSEL = $100_B$ (recessive to recessive). These delays indicate the time needed to represent a new bit value on the physical implementation of the CAN bus.

### 16.1.6 Message Acceptance Filtering

This section describes the Message Acceptance Filtering capabilities of the MultiCAN module.

#### 16.1.6.1 Receive Acceptance Filtering

When a CAN frame is received by a CAN node, a unique message object is determined in which the received frame is stored after successful frame reception. A message object is qualified for reception of a frame if the following six conditions are fulfilled.

- The message object is allocated to the message object list of the CAN node by which the frame is received.
- Bit MOSTATn.MSGVAL is set.
- Bit MOSTATn.RXEN is set.
- Bit MOSTATn.DIR is equal to bit RTR of the received frame.
  If bit MOSTATn.DIR = 1 (transmit object), the message object accepts only remote frames. If bit MOSTATn.DIR = 0 (receive object), the message object accepts only data frames.
- If bit MOAMRn.MIDE = 1, the IDE bit of the received frame is evaluated in the following way: If MOARn.IDE = 1, the IDE bit of the received frame must be set (indicates extended identifier). If MOARn.IDE = 0, the IDE bit of the received frame must be cleared (indicates standard identifier).
  If bit MOAMRn.MIDE = 0, the IDE bit of the received frame is “don’t care”. In this case, message objects with standard and extended frames are accepted.
- The identifier of the received frame matches the identifier stored in the register MOARn as qualified by the acceptance mask in the MOAMRn register. This means that each bit of the received message object identifier is equal to the bit field MOARn.ID, except those bits for which the corresponding acceptance mask bits in bit field MOAMRn.AM are cleared. These identifier bits are “don’t care” for reception.

Among all messages that fulfill all six qualifying criteria the message object with the highest receive priority wins receive acceptance filtering and becomes selected to store the received frame. All other message objects lose receive acceptance filtering.

The following priority scheme is defined for the message objects:

A message object a (MOa) has higher receive priority than a message object b (MOb) if the following two conditions are fulfilled (see **Page 16-93**):

1. MOa has a higher priority class than MOb. This means, the 2-bit priority bit field MOARa.PRI must be equal or less than bit field MOARb.PRI.
2. If both message objects have the same priority class (MOARa.PRI = MOARb.PRI), MOb is a list successor of MOa. This means that MOb can be reached by means of successively stepping forward in the list, starting from a.

### 16.1.6.2 Transmit Acceptance Filtering

A message is requested for transmission by setting a transmit request in the message object that holds the message. If more than one message object have a valid transmit request for the same CAN node, one of these message objects is chosen for transmission, because only a single message object can be transmitted at one time on a CAN bus.

A message object is qualified for transmission on a CAN node if the following four conditions are are fulfilled.

1. The message object is allocated to the message object list of the CAN node.
2. Bit MOSTATn.MSGVAL is set.
3. Bit MOSTATn.TXRQ is set.
4. Bit MOSTATn.TXEN0 and MOSTATn.TXEN1 are set.

A priority scheme determines which of all qualifying message objects is transmitted first. The following assumption is made: message object a (MOa) and message object b (MOb) are two message objects qualified for transmission. MOb is a list successor of MOa. This means, MOb can be reached by means of successively stepping forward in the list, starting from a.

If both message objects belong to a different priority class (different value of bit field MOARn.PRI), then the message object with lower MOAR.PRI value has higher transmit priority and will be transmitted first.

If both message objects belong to the same priority class (identical PRI bit field in register MOARn), MOa has a higher transmit priority than MOb if one of the following conditions is fulfilled.

- PRI = $10_B$ and CAN message MOa has higher or equal priority than CAN message MOb with respect to CAN arbitration rules (see **Table 16-13**).
- PRI = $01_B$ or PRI = $11_B$ (priority by list order).

The message object that is qualified for transmission and has highest transmit priority wins the transmit acceptance filtering, and will be transmitted first. All other message objects lose the current transmit acceptance filtering round. They get a new chance in subsequent acceptance filtering rounds.

The priority rules are valid for normal CAN operation.

### 16.1.7 Message Postprocessing

After a message object has successfully received or transmitted a frame, the CPU can be notified to perform a message postprocessing on the message object. The postprocessing of the MultiCAN module consists of two elements:

1. Message interrupts to trigger postprocessing.
2. Message pending registers to collect pending message interrupts into a common structure for postprocessing.

#### 16.1.7.1 Message Interrupts

When the storage of a received frame into a message object or the successful transmission of a frame is completed, a message interrupt can be issued. For each message object, a transmit and a receive interrupt can be generated and routed to one of the eight CAN interrupt output lines (see **Figure 16-9**). A receive interrupt occurs also after a frame storage event has been induced by a FIFO or a gateway action. The status bits MOSTATn.TXPND and MOSTATn.RXPND are always set after a successful transmission/reception, regardless if the respective message interrupt is enabled or not.

A FIFO full interrupt condition of a message object is provided. If bit field MOFCRn.OVIE is set, the FIFO full interrupt will become activated depending on the actual message object type.

In case of a Receive FIFO Base Object (MOFCRn.MMC = $0001_B$), the FIFO full interrupt is routed to the interrupt output line CANSRCm as defined by the transmit interrupt node pointer MOIPRn.TXINP.

In case of a Transmit FIFO Base Object (MOFCRn.MMC = $0010_B$), the FIFO full interrupt is routed to the interrupt output line CANSRCm as defined by the receive interrupt node pointer MOIPRn.RXINP.

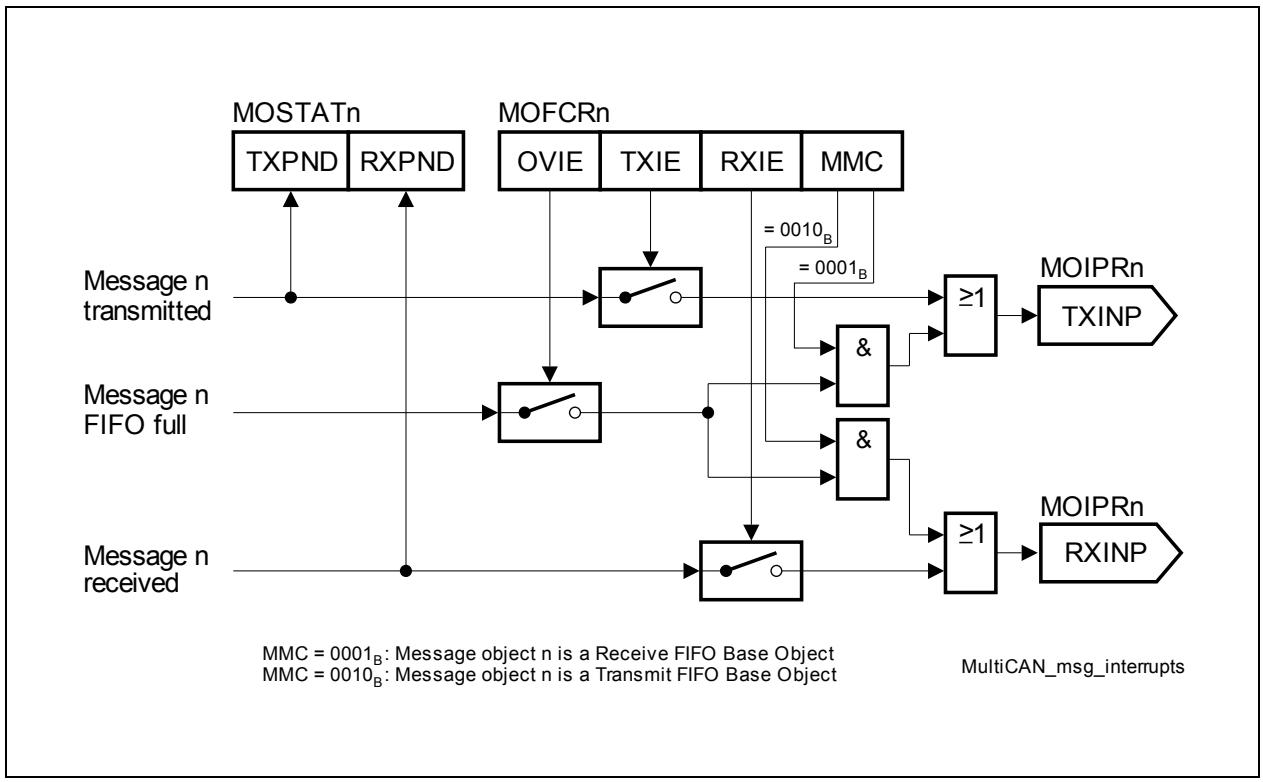



**Figure 16-9 Message Interrupt Request Routing**

### 16.1.7.2 Pending Messages

With a message interrupt request generation, a message pending bit is set in one of the Message Pending Registers. There are two Message Pending Registers MSPNDk (k = 1-0) with 32 pending bits available to each, resulting in 64 pending bits. **Figure 16-10** shows the allocation of the message pending bits.

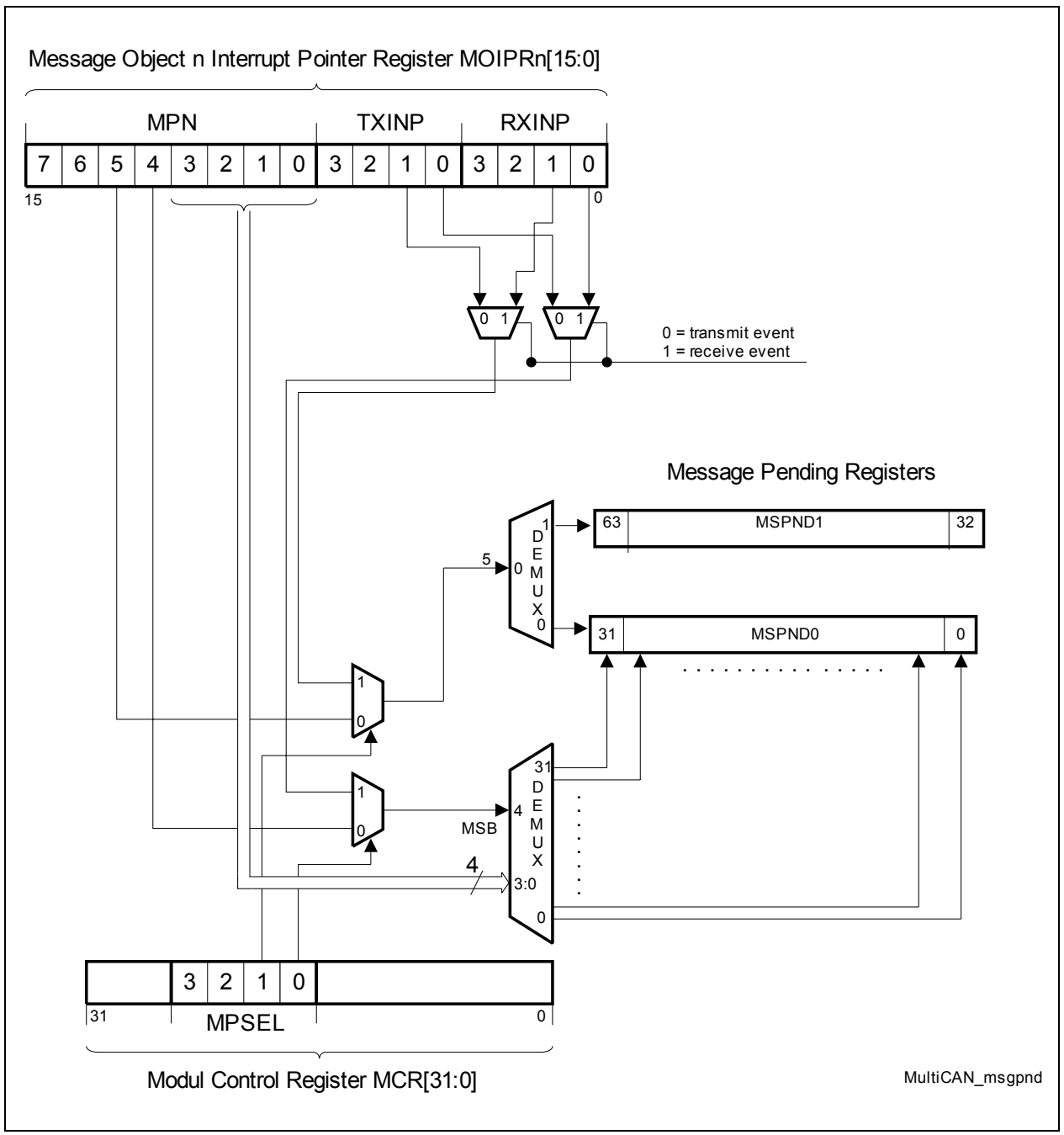



**Figure 16-10 Message Pending Bit Allocation**

The location of a pending bit is defined by two demultiplexers selecting the number k of the MSPNDk registers (1-bit demux), and the bit location within the corresponding MSPNDk register (5-bit demux).

### Allocation Case 1

In this allocation case, bit field MCR.MPSEL = $0000_B$. Here, the location selection consists of two parts:

- The bit 5 of MOIPRn.MPN (MPN[5]) select the number k [k=1-0] of a Message Pending Register MSPNDk in which the pending bit will be set.
- The lower five bits of MOIPRn.MPN (MPN[4:0]) select the bit position (31-0) in MSPNDk for the pending bit to be set.

### Allocation Case 2

In this allocation case, bit field MCR.MPSEL is taken into account for pending bit allocation. Bit field MCR.MPSEL allows the inclusion of the interrupt request node pointer for reception (MOIPRn.RXINP) or transmission (MOIPRn.TXINP) for pending bit allocation in a way that different target locations for the pending bits are used in receive and transmit cases. If MPSEL = $1111_B$, the location selection operates in the following way:

- At a transmit event, the bit 1 of TXINP define the number k of a Pending Register MSPNDk in which the pending bit will be set. At a receive event, the bit 1 of RXINP define the number k.
- The bit position (31-0) in MSPNDk for the pending bit to be set is selected by the lowest bit of TXINP or RXINP and the four least significant bits of MPN.

### General Hints

The Message Pending Registers MSPNDk can be written by software. Bits that are written with 1 are left unchanged and bits which are written with 0 are cleared. This allows individual MSPNDk bits to be cleared with a single register write access. Therefore, access conflicts are avoided when the MultiCAN module (hardware) sets another pending bit at the same time when software writes to the register.

Each Message Pending Register MSPNDk is associated with a Message Index Register MSIDk which indicates the lowest bit position of all set (1) bits in Message Pending Register k. The MSIDk register is a read-only register which is updated immediately when a value in the corresponding Message Pending Register k is changed.

### 16.1.8 Message Object Data Handling

This section describes the handling capabilities for the Message Object Data of the MultiCAN module.

#### 16.1.8.1 Frame Reception

After the reception of a message, it is stored in a message object according to the scheme shown in **Figure 16-11**. The MultiCAN module not only copies the received data into the message object, but it provides advanced features to enable consistent data exchange between MultiCAN and CPU.

**MSGVAL**

During the frame reception, information is stored only in the message object when MOSTATn.MSGVAL = 1. If bit MSGVAL is reset by the CPU, the MultiCAN module stops all ongoing write accesses to the message object so that the message object can be reconfigured by the CPU with subsequent write accesses to it without being disturbed by the MultiCAN.

**RTSEL**

When the CPU re-configures a message object during CAN operation (for example, clears MSGVAL, modifies the message object and sets MSGVAL again), the following scenario can occur:

1. The message object wins receive acceptance filtering.
2. The CPU clears MSGVAL to re-configure the message object.
3. The CPU sets MSGVAL again after re-configuration.
4. The end of the received frame is reached. As MSGVAL is set, the received data is stored in the message object, a message interrupt request is generated, gateway and FIFO actions are processed, etc.

After the re-configuration of the message object (after step 3 above) the storage of further received data may be undesirable. This can be achieved through bit MOCTRn.RTSEL (“Receive/Transmit Selected”) that allows a message object to be disconnected from an ongoing frame reception.

When a message object wins the receive acceptance filtering, its RTSEL bit is set by the MultiCAN module to indicate an upcoming frame delivery. The MultiCAN module checks RTSEL whether it is set on successful frame reception to verify that the object is still ready for receiving the frame. The received frame is then stored in the message object (along with all subsequent actions such as message interrupts, FIFO & gateway actions, flag updates) only if RTSEL = 1.

When a message object is invalidated during CAN operation (resetting bit MSGVAL), RTSEL should be cleared before setting MSGVAL again (latest with the same write access that sets MSGVAL) to prevent the storage of a frame that belongs to the old

context of the message object. Therefore, a message object re-configuration should consist of the following steps:

1. Clear MSGVAL bit
2. Re-configure the message object while MSGVAL = 0
3. Clear RTSEL bit and set MSGVAL again

### RXEN

Bit MOSTATn.RXEN enables a message object for frame reception. A message object can receive CAN messages from the CAN bus only if RXEN = 1. The MultiCAN module evaluates RXEN only during receive acceptance filtering. After receive acceptance filtering, RXEN is ignored and has no further influence on the actual storage of a received message in a message object.

Bit RXEN enables the “soft phase out” of a message object: after clearing RXEN, a currently received CAN message for which the message object has won acceptance filtering is still stored in the message object but for subsequent messages the message object no longer wins receive acceptance filtering.

### RXUPD, NEWDAT and MSGLST

An ongoing frame storage process is indicated by the bit MOSTATn.RXUPD (“Receive Updating”). RXUPD is set with the start and cleared with the end of a message object update (which consists of frame storage as well as flag updates).

After storing the received frame (identifier, IDE bit, DLC and the data field for data frames as well) the bit MOSTATn.NEWDAT (“New Data”) is set. If NEWDAT was already set before it becomes set again, bit MOSTATn.MSGLST (“Message Lost”) is set to indicate a data loss condition.

The RXUPD and NEWDAT flags can help to read consistent frame data from the message object during an ongoing CAN operation. The following steps are recommended to be executed:

1. Clear NEWDAT bit.
2. Read message content (identifier, data etc.) from the message object.
3. Check that both NEWDAT and RXUPD are cleared. If this is not the case, go back to step 1.
4. As step 3 was successful, the message object content is consistent, i.e., has not been updated by the MultiCAN module while reading.

Bits RXUPD, NEWDAT and MSGLST have the same behavior for the reception of data as well as remote frames.

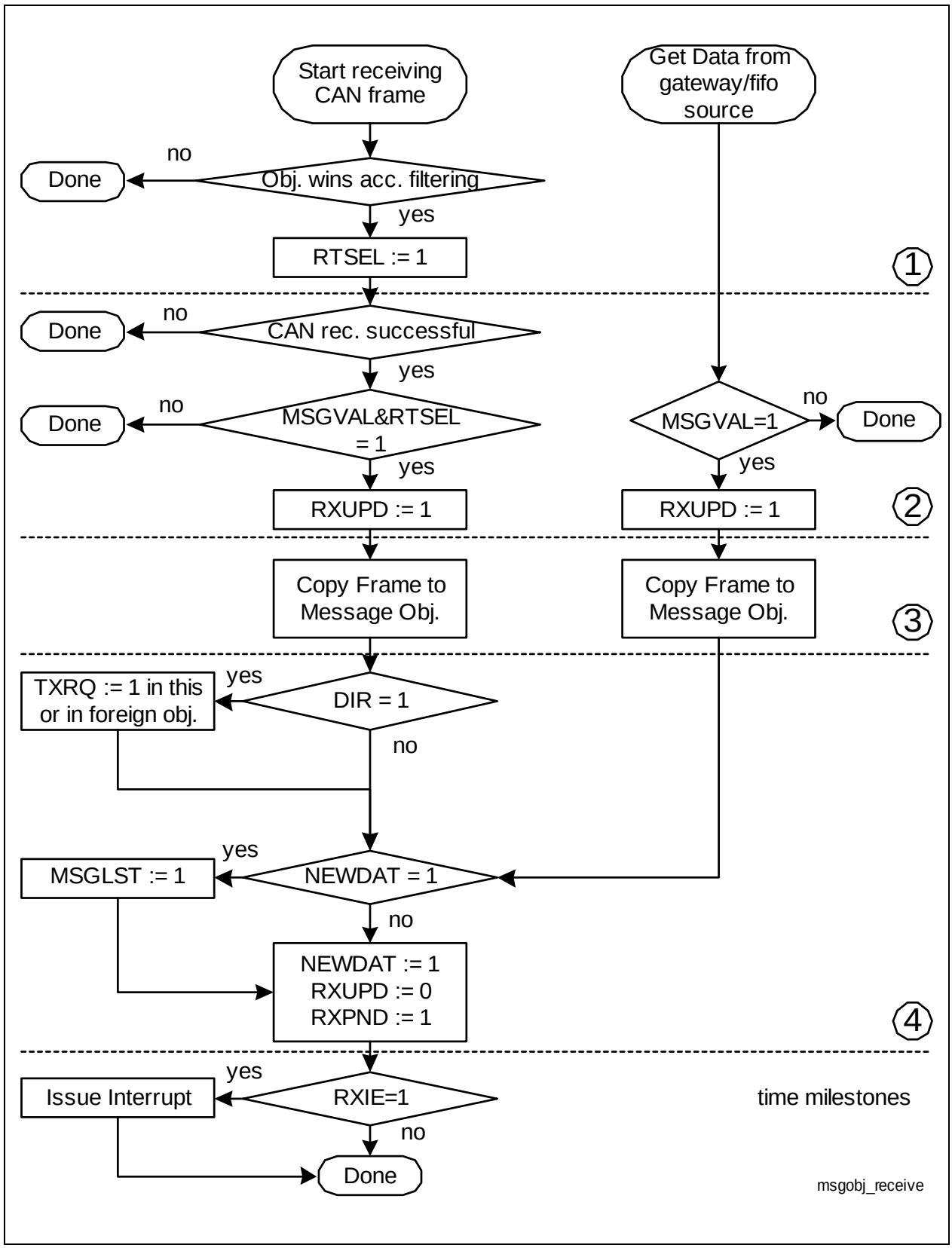



**Figure 16-11 Reception of a Message Object**

### 16.1.8.2 Frame Transmission

The process of a message object transmission is shown in **Figure 16-12**. With the copy of the message object content to be transmitted (identifier, IDE bit, RTR = DIR bit, DLC, and for data frames also the data field) into the internal transmit buffer of the assigned CAN node, also several status flags are served and monitored to control consistent data handling.

The transmission process of a message object starting after the transmit acceptance filtering is identical for remote and data frames.

#### MSGVAL, TXRQ, TXEN0, TXEN1

A message can only be transmitted if all four bits in MOSTATn Register MSGVAL (“Message Valid”), TXRQ (“Transmit Request”), TXEN0 (“Transmit Enable 0”), TXEN1 (“Transmit Enable 1”) are set. Although these bits are equivalent with respect to the transmission process, they have different semantics:

**Table 16-3 Message Transmission Bit Definitions**

| Bit | Description |
|---|---|
| **MSGVAL** | **Message Valid**<br>This is the main switch bit of the message object. |
| **TXRQ** | **Transmit Request**<br>This is the standard transmit request bit. This bit must be set whenever a message object is to be transmitted. TXRQ is cleared by hardware at the end of a successful transmission, except when there is new data (indicated by NEWDAT = 1) to be transmitted.<br>When bit MOFCRn.STT (“Single Transmit Trial”) is set, TXRQ is already cleared when the content of the message object is copied into the transmit frame buffer of the CAN node.<br>A received remote request (after a remote frame reception) sets bit TXRQ to request the transmission of the requested data frame. |
| **TXEN0** | **Transmit Enable 0**<br>This bit can be temporarily cleared by software to suppress the transmission of this message object when it writes new content to the data field. This avoids transmission of inconsistent frames that consist of a mixture of old and new data.<br>Remote requests are still accepted when TXEN0 = 0, but transmission of the data frame is suspended until transmission is re-enabled by software (setting TXEN0). |

**Table 16-3 Message Transmission Bit Definitions** (cont'd)

| Bit | Description |
|---|---|
| **TXEN1** | **Transmit Enable 1**<br>This bit is used in transmit FIFOs to select the message object that is transmit active within the FIFO structure.<br>For message objects that are not transmit FIFO elements, TXEN1 can either be set permanently to 1 or can be used as a second independent transmission enable bit. |

**RTSEL**

When a message object has been identified after transmission acceptance filtering to be transmitted next, bit MOCTRn.RTSEL ("Receive/Transmit Selected") becomes set.

When the message object is copied into the internal transmit buffer, bit RTSEL is checked, and the message is only transmitted if RTSEL = 1. After the successful transmission of the message, bit RTSEL is checked again and the message postprocessing is only executed if RTSEL = 1.

For a complete re-configuration of a valid message object, the following steps should be executed:

1. Clear MSGVAL bit
2. Re-configure the message object while MSGVAL = 0
3. Clear RTSEL and set MSGVAL

Clearing of RTSEL ensures that the message object is disconnected from an ongoing/scheduled transmission and no message object processing (copying message to transmit buffer including clearing NEWDAT, clearing TXRQ, time stamp update, message interrupt, etc.) within the old context of the object can occur after the message object becomes valid again, but within a new context.

**NEWDAT**

When the content of a message object has been transferred to the internal transmit buffer of the CAN node, bit MOSTATn.NEWDAT (New Data) is cleared by hardware to indicate that the transmit message object data is no longer new.

When the transmission of the frame is successful and NEWDAT is still cleared (if no new data has been copied into the message object meanwhile), TXRQ (Transmit Request) is cleared automatically by hardware.

If, however, the NEWDAT bit has been set again by the software (because a new frame is to be transmitted), TXRQ is not cleared to enable the transmission of the new data.

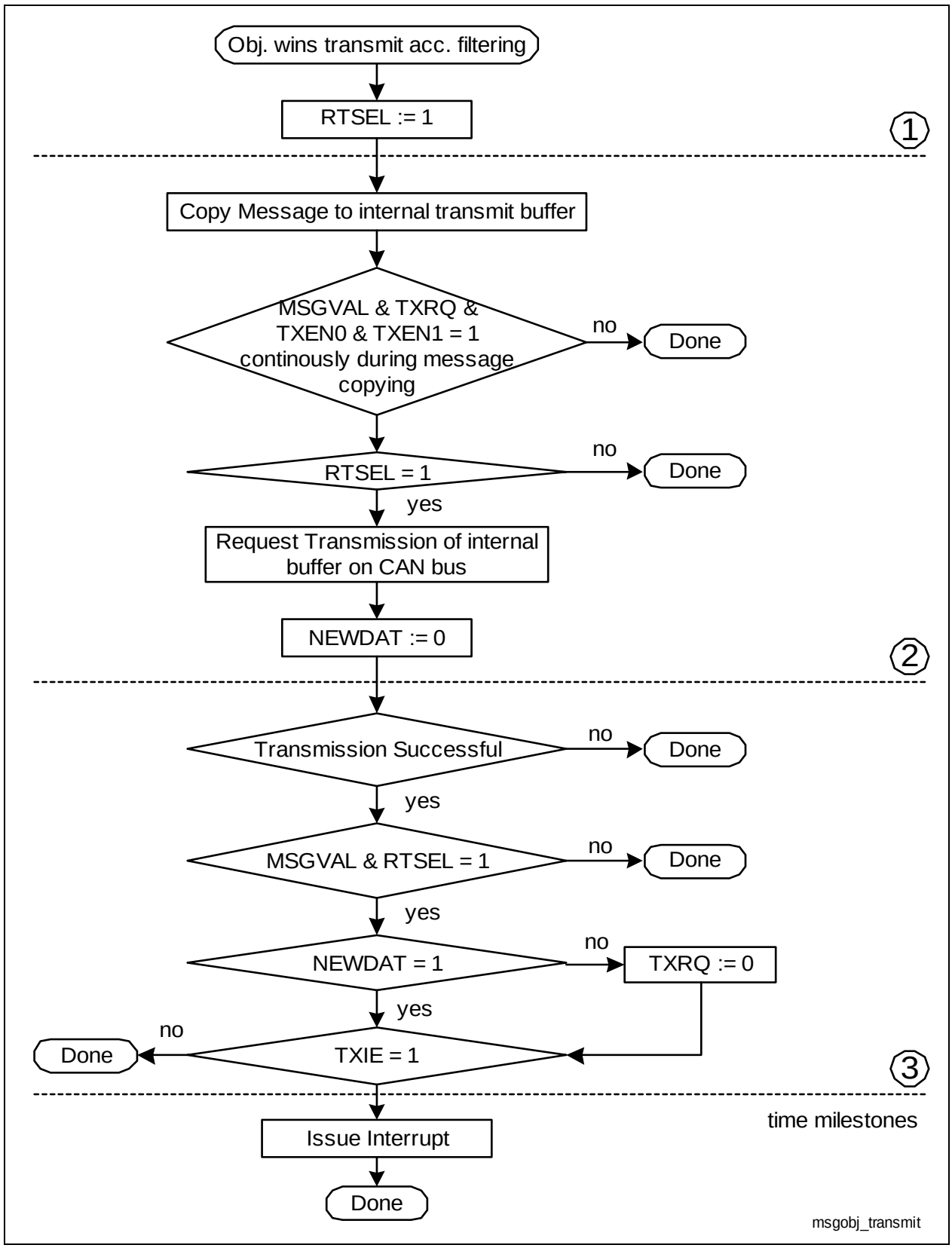



**Figure 16-12 Transmission of a Message Object**

### 16.1.9 Message Object Functionality

This section describes the functionality of the Message Objects in the MultiCAN module.

#### 16.1.9.1 Standard Message Object

A message object is selected as Standard Message Object when bit field MOFCRn.MMC = $0000_B$. The Standard Message Object can 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 following sections) are available and can be individually selected.

#### 16.1.9.2 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 MOFCRn.SDT.

**Message Reception**

When a received message stored in a message object is overwritten by a new received message, the content of the first message gets lost and is replaced with the content of the new received message (indicated by MSGLST = 1).

In single data transfer mode (SDT = 1), bit MSGVAL of the message object is automatically cleared by hardware after the storage of a received data frame. This prevents the reception of further messages.

After the reception of a remote frame, bit MSGVAL is not automatically cleared.

**Message Transmission**

When a message object receives a series of multiple remote requests, then it transmits several data frames in response to the remote requests. If the data within the message object has not been updated in the time between the transmissions, the same data can be sent more than once on the CAN bus.

In single data transfer mode (SDT = 1), this is avoided because MSGVAL is automatically cleared after the successful transmission of a data frame.

After the transmission of a remote frame, bit MSGVAL is not automatically cleared.

#### 16.1.9.3 Single Transmit Trial

If bit MOFCRn.STT is set, then the transmission request is cleared (TXRQ = 0) when the frame content of the message 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.

### 16.1.9.4 Message Object FIFO Structure

In case of high CPU load it may be difficult to process a series of CAN frames in time. This may happen if multiple messages are received or must be transmitted in short time.

Therefore, a FIFO buffer structure is available to avoid loss of incoming messages and to minimize the setup time for outgoing messages. The FIFO structure can also be used to automate the reception or transmission of a series of CAN messages and to generate a single message interrupt when the whole CAN frame series is done.

There can be several FIFOs in parallel. The number of FIFOs and their size are only limited by the number of available message objects. A FIFO can be installed, resized and de-installed at any time, even during CAN operation.

The basic structure of a FIFO is shown in **Figure 16-13**. A FIFO consists of one base object and n slave objects. The slave objects are chained together in a list structure (similar as in message object lists). The base object may be allocated to any list. Although **Figure 16-13** shows the base object as a separate part beside the slave objects, it is also possible to integrate the base object at any place into the chain of slave objects. This means that the base object is slave object, too (not possible for gateways). The absolute object numbers of the message objects have no impact on the operation of the FIFO.

The base object need not to be allocated to the same list as the slave objects. Only the slave object must be allocated to a common list (as they are chained together). Several pointers (BOT, CUR and TOP) that are located in the Register MOFGPRn link the base object to the slave objects, regardless whether the base object is allocated to the same or to another **list** than the slave objects.

The smallest FIFO would be a single message object which is both FIFO base and FIFO slave (not very useful). The biggest possible FIFO structure would include all message objects of the MultiCAN module. Any FIFO sizes between these limits are possible.

In the FIFO base object, the FIFO boundaries are defined. Bit field MOFGPRn.BOT of the base object points to (includes the number of) the bottom slave object in the FIFO structure. The MOFGPRn.TOP bit field points to (includes the number of) the top slave object in the FIFO structure. The MOFGPRn.CUR bit field points to (includes the number of) the slave object that is actually selected by the MultiCAN module for message transfer. When a message transfer occurs with this object, CUR is set to the next message object in the list structure of the slave objects (CUR = PNEXT of current object). If CUR was equal to TOP (top of the FIFO reached), the next update of CUR will result in CUR = BOT (wrapped around from the top to the bottom of the FIFO). This scheme represents a circular FIFO structure where the bit fields BOT and TOP establish the link from the last to the first element.

Bit field MOFGPRn.SEL of the base object can be used for monitoring purposes. It allows a slave object to be defined within the list at which a message interrupt is generated whenever the CUR pointer reaches the value of the SEL pointer. Thus, SEL

allows the end of a predefined message transfer series to be detected or to issue a warning interrupt when the FIFO becomes full.

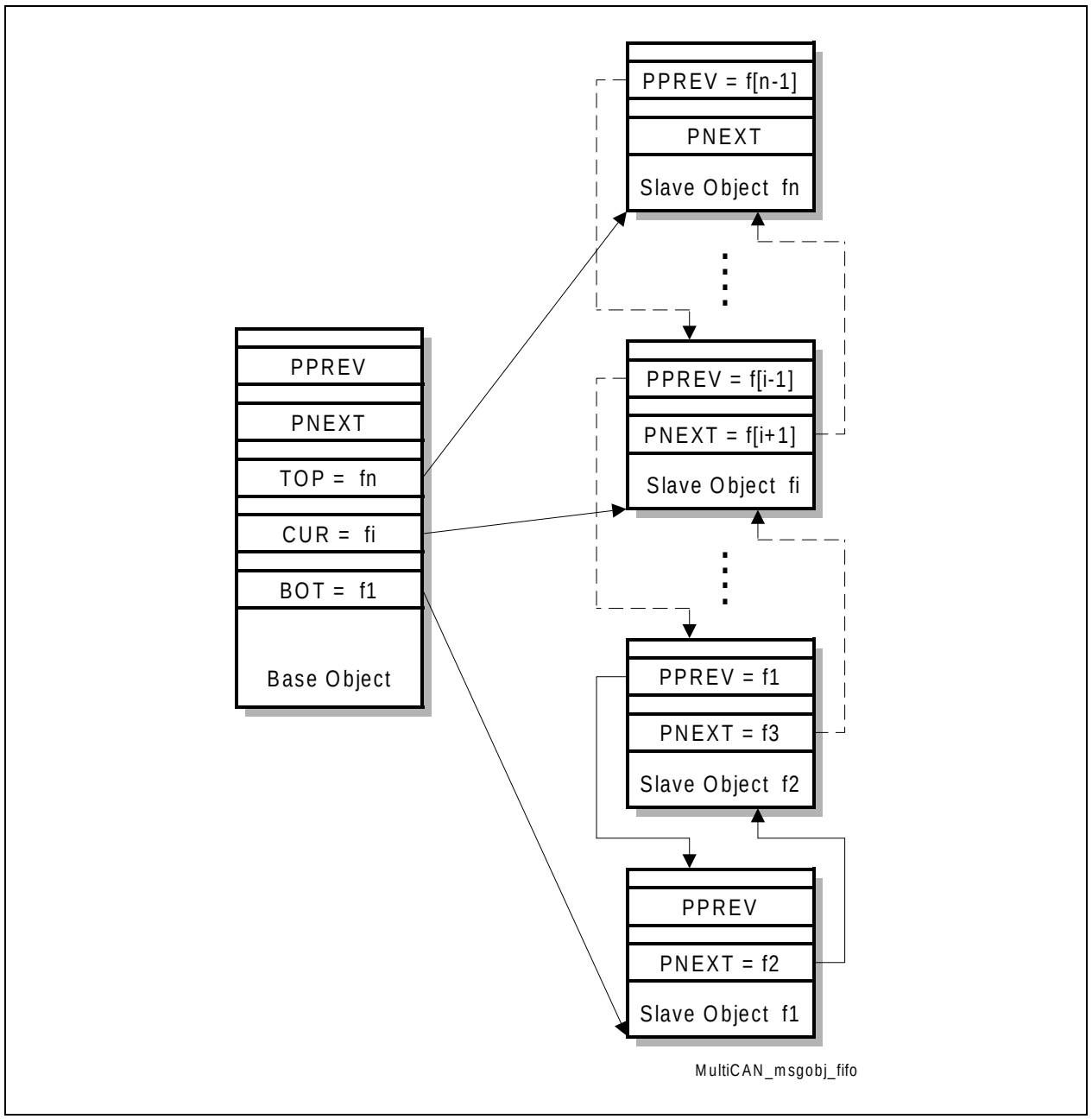



**Figure 16-13 FIFO Structure with FIFO Base Object and n FIFO Slave Objects**

### 16.1.9.5 Receive FIFO

The Receive FIFO structure is used to buffer incoming (received) remote or data frames.

A Receive FIFO is selected by setting MOFCRn.MMC = $0001_B$ in the FIFO base object. This MMC code automatically designates a message object as FIFO base object. The message modes of the FIFO slave objects are not relevant for the operation of the Receive FIFO.

When the FIFO base object receives a frame from the CAN node it belongs to, the frame is not stored in the base object itself but in the message object that is selected by the base object's MOFGPRn.CUR pointer. This message object receives the CAN message as if it is the direct receiver of the message. However, MOFCRn.MMC = $0000_B$ is implicitly assumed for the FIFO slave object, and a standard message delivery is performed. The actual message mode (MMC setting) of the FIFO slave object is ignored. For the slave object, no acceptance filtering takes place that checks the received frame for a match with the identifier, IDE bit, and DIR bit.

With the reception of a CAN frame, the current pointer CUR of the base object is set to the number of the next message object in the FIFO structure. This message object will then be used to store the next incoming message.

If bit field MOFCRn.OVIE ("Overflow Interrupt Enable") of the FIFO base object is set and the current pointer MOFGPRn.CUR becomes equal to MOFGPRn.SEL, a FIFO overflow interrupt request is generated. This interrupt request is generated on interrupt node TXINP of the base object immediately after the storage of the received frame in the slave object. Transmit interrupts are still generated if TXIE is set.

A CAN message is stored in FIFO base and slave object only if MSGVAL = 1.

### 16.1.9.6 Transmit FIFO

The Transmit FIFO structure is used to buffer a series of data or remote frames that must be transmitted.

A Transmit FIFO is selected by setting MOFCRn.MMC = $0010_B$ in the FIFO base object. Unlike the Receive FIFO, slave objects assigned to the Transmit FIFO are required to set explicitly their bit fields MOFCRn.MMC = $0011_B$. The CUR pointer in all slave objects must point back to the Transmit FIFO Base Object (to be initialized by software).

The MOSTATn.TXEN1 bits (Transmit Enable 1) of all message objects except the one which is selected by the CUR pointer of the base object must be cleared by software. TXEN1 of the message (slave) object selected by CUR must be set. CUR (of the base object) may be initialized to any FIFO slave object.

When tagging the message objects of the FIFO as valid to start the operation of the FIFO, then the base object must be tagged valid (MSGVAL = 1) first.

Before a Transmit FIFO becomes de-installed during operation, its slave objects must be tagged invalid (MSGVAL = 0).

The Transmit FIFO uses the bit MOCTRn.TXEN1 of all FIFO elements to select the actual message for transmission. Transmit acceptance filtering evaluates TXEN1 for each message object and a message object can win transmit acceptance filtering only if its TXEN1 bit is set. When a FIFO object has transmitted a message, the hardware clears its TXEN1 bit in addition to standard transmit postprocessing (clear TXRQ, transmit interrupt etc.) and moves the CUR pointer to the next message object to be transmitted. TXEN1 is set automatically (by hardware) in the next message object. Thus, TXEN1 moves along the Transmit FIFO structure like a token that selects the active element.

If bit field MOFCRn.OVIE (“Overflow Interrupt Enable”) of the FIFO base object is set and the current pointer CUR becomes equal to MOFGPRn.SEL, a FIFO overflow interrupt request is generated. The interrupt request is generated on interrupt node RXINP of the base object after postprocessing of the received frame. Receive interrupts are still generated for the Transmit FIFO base object if bit RXIE is set.

### 16.1.9.7 Gateway Mode

The gateway mode allows an automatic information transfer to be established between two independent CAN buses without CPU interaction.

The gateway mode operates on message object level. In gateway mode, information is transferred between two message objects, resulting in an information transfer between the two CAN nodes to which the message objects are allocated. A gateway may be established with any pair of CAN nodes, and there can be as many gateways as there are message objects available to build the gateway structure.

Gateway mode is selected by setting MOFCRs.MMC = $0100_B$ of the gateway source object s. The gateway destination object d is selected by the MOFGPRd.CUR pointer of the source object. The gateway destination object only needs to be valid (its MSGVAL = 1). All other settings are not relevant for the information transfer from the source object to the destination object.

A gateway source object s behaves like a standard message object except some additional actions are performed by the MultiCAN module when a CAN frame has been received and stored in the source object (see **Figure 16-14**):

1. If bit MOFCRs.DLCC is set, the data length code MOFCRs.DLC is copied from the gateway source object to the gateway destination object.
2. If bit MOFCRs.IDC is set, the identifier MOARs.ID and the identifier extension MOARs.IDE are copied from the gateway source object to the gateway destination object.
3. If bit MOFCRs.DATC is set, the data bytes stored in the two data registers MODATALs and MODATAHs are copied from the gateway source object to the gateway destination object. All 8 data bytes are copied, even if MOFCRs.DLC indicates less than 8 data bytes.
4. If bit MOFCRs.GDFS is set, the transmit request flag MOSTATd.TXRQ is set in the gateway destination object.
5. The receive pending bit MOSTATd.RXPND and the new data bit MOSTATd.NEWDAT are set in the gateway destination object.
6. A message interrupt request is generated for the gateway destination object if its MOSTATd.RXIE is set.
7. The current object pointer MOFGPRs.CUR of the gateway source object is moved to the next destination object according to the FIFO rules as described on **Page 16-34**. A gateway with a single (static) destination object is obtained by setting MOFGPRs.TOP = MOFGPRs.BOT = MOFGPRs.CUR = destination object.

The link from the gateway source object to the gateway destination object works in the same way as the link from a FIFO base to a FIFO slave. This means that a gateway with an integrated destination FIFO may be created; in **Figure 16-13**, where the object on the left is the gateway source object and the message object on the right side is the gateway destination objects.

The gateway operates 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).

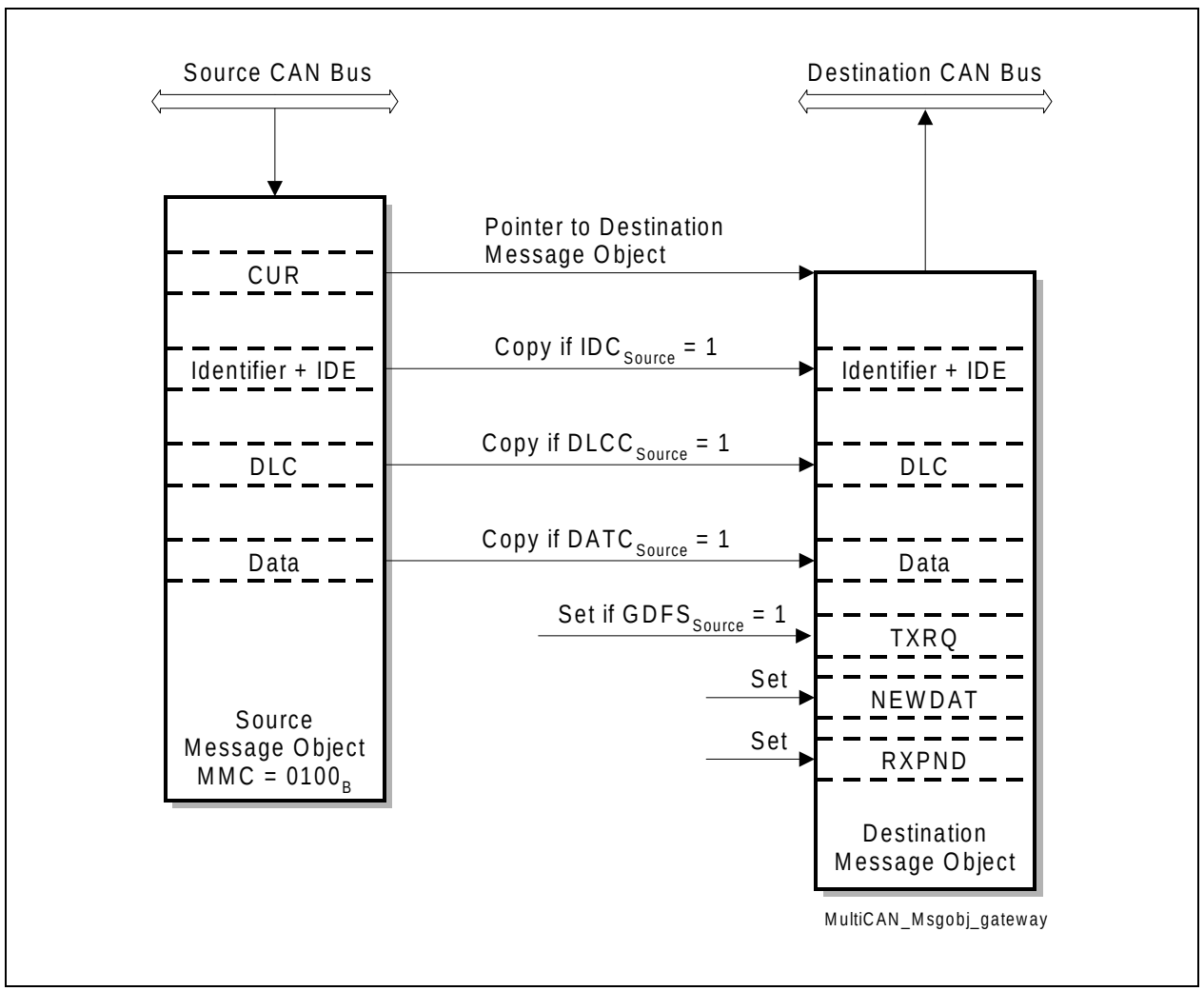



**Figure 16-14 Gateway Transfer from Source to Destination**

### 16.1.9.8 Foreign Remote Requests

When a remote frame has been received on a CAN node and is stored in a message object, a transmit request is set to trigger the answer (transmission of a data frame) to the request or to automatically issue a secondary request. If the Foreign Remote Request Enable bit MOFCRn.FRREN is cleared in the message object in which the remote request is stored, MOSTATn.TXRQ is set in the same message object.

If bit FRREN is set, TXRQ is set in the message object that is referenced by pointer MOFGPRn.CUR. The value of CUR is, however, not changed by this feature.

Although the foreign remote request feature works independently of the selected message mode, it is especially useful for gateways to issue a remote request on the source bus of a gateway after the reception of a remote request on the gateway destination bus. According to the setting of FRREN in the gateway destination object, there are two capabilities to handle remote requests that appear on the destination side (assuming that the source object is a receive object and the destination is a transmit object, i.e. $DIR_{source}$ = 0 and $DIR_{destination}$ = 1):

**FRREN = 0 in the Gateway Destination Object**

1. A remote frame is received by gateway destination object.
2. TXRQ is set automatically in the gateway destination object.
3. A data frame with the current data stored in the destination object is transmitted on the destination bus.

**FRREN = 1 in the Gateway Destination Object**

1. A remote frame is received by gateway destination object.
2. TXRQ is set automatically in the gateway source object (must be referenced by CUR pointer of the destination object).
3. A remote request is transmitted by the source object (which is a receive object) on the source CAN bus.
4. The receiver of the remote request responds with a data frame on the source bus.
5. The data frame is stored in the source object.
6. The data frame is copied to the destination object (gateway action).
7. TXRQ is set in the destination object (assuming $GDFS_{source}$ = 1).
8. The new data stored in the destination object is transmitted on the destination bus, as response to the initial remote request on the destination bus.

### 16.1.10 Access Mediator

The MultiCAN needs to cover a maximum of 16 Kbytes SFR kernel address range, which is much greater than the XC878 can provide. To meet this demand, an address extension decoding mechanism is built in the unit called “Access Mediator” to decode the SFRs in the MultiCAN kernel. The address lines are not directly controlled by the CPU instruction itself, but they are derived from register bits that have to be programmed before accessing the MultiCAN kernel.

To decode the address of the MultiCAN kernel registers, at least 14-bit address line is needed. As the MultiCAN registers are 32-bit wide (4 Bytes), then the address lines A[1:0] are not needed for decoding and are tied to “00”. The address lines A[13:2] are implemented and they are programmed from the register bits CA2 to CA9 in the register CAN_ADL and CA10 to CA13 in the register CAN_ADH. The address registers need to be programmed before accessing the MultiCAN registers.

The data bus are 32 bit (D[31:0]) between the Access Mediator and MultiCAN kernel. Four data registers CAN_DATAn (n = 3-0) are implemented in the Access Mediator. Each register in the MultiCAN kernel is read and written via these 4 data registers.

When writing to MultiCAN kernel, the data in the registers CAN_DATAn (n = 3-0) are set valid or not valid by configuring the register bits Vn (n = 3-0) in the register CAN_ADCON. Only the valid data (bytes) are sent during the write process. The register bits Vn (n = 3-0) has no effect on the read process. During the read process, 32-bit data will be read from the MultiCAN kernel.

The register bit CAN_ADCON.BSY is used to indicate if the transmission is complete or not. When the BSY register bit is set, the data registers and address registers will not accept any read/write access. The write/read action to the MultiCAN kernel only takes place when writing the CAN_ADCON register. The write/read action to the MultiCAN kernel is defined by the bit CAN_ADCON.RWEN. Reading the CAN_ADCON register has no effect on write/read data to/from the MultiCAN kernel. Each write/read action to the MultiCAN kernel only writes/reads data once.

Furthermore, there is an additional functionality for auto increment/decrement the address by configuring the bit field CAN_ADCON.AUAD. The address can be auto incremented/decremented by 1 or auto incremented by 8 (which is useful when programming the message objects). If this function is enabled, after a read/write process is finished, the address pointer will automatically point to the next register address. The address registers CAN_ADL and CAN_ADH also reflect the address that the address pointer pointed to. The next read/write action to the next register can be taken immediately without writing the address to the registers CAN_ADL and CAN_ADH again.

#### Write Process to the MultiCAN Kernel

- Write the address of the MultiCAN kernel register to the CAN_ADL and CAN_ADH registers.

- Write the data to the CAN_DATA0/CAN_DATA1/CAN_DATA2/CAN_DATA3 registers.
- Write the register CAN_ADCON, including setting the valid bit of the data registers and setting register bit RWEN to 1.
- The valid data will be written to the MultiCAN kernel only once. Register bit BSY will become 1.
- When Register bit BSY becomes 0, the transmission is finished.

**Read Process to the MultiCAN Kernel**

- Write the address of the MultiCAN kernel register to the CAN_ADL and CAN_ADH registers.
- Write the register CAN_ADCON, setting register bit RWEN to 0.
- The 32-bit data will be read from the MultiCAN kernel only once. Register bit BSY will become 1.
- When register bit BSY becomes 0, the transmission is finished.
- Read the data from the CAN_DATA0/CAN_DATA1/CAN_DATA2/CAN_DATA3 registers.

*Note: The address registers and data registers should be only written/read when register bit BSY is 0.*

### 16.1.11 Port Control

The interconnections between the MultiCAN module and the port I/O lines are controlled in the port logics. In addition to the I/O control selection, the selection of a CAN node's receive input line is configured by a bit field RXSEL in its node port control register NPCRx (x = 1-0).

**Table 16-4** shows how bits and bit fields must be programmed for the required I/O functionality of the CAN I/O lines.

**Table 16-4 CAN I/O Control Selection**

| Port Lines | PISEL Register Bit | Input/Output Control Register Bits | I/O |
|---|---|---|---|
| P1.0/RXDC0_0 | NPCR0.RXSEL = $000_B$ | P1_DIR.P0 = $0_B$ | Input |
| P1.1/TXDC0_0 | – | P1_DIR.P1 = $1_B$ | Output |
| | | P1_ALTSEL0.P1 = $1_B$ | |
| | | P1_ALTSEL1.P1 = $1_B$ | |
| P3.4/RXDC0_1 | NPCR0.RXSEL = $001_B$ | P3_DIR.P4 = $0_B$ | Input |
| P3.5/TXDC0_1 | – | P3_DIR.P5 = $1_B$ | Output |
| | | P3_ALTSEL0.P5 = $1_B$ | |
| | | P3_ALTSEL1.P5 = $1_B$ | |
| P1.6/RXDC0_2 | NPCR0.RXSEL = $010_B$ | P1_DIR.P6 = $0_B$ | Input |
| P1.7/TXDC0_2 | – | P1_DIR.P7 = $1_B$ | Output |
| | | P1_ALTSEL0.P7 = $1_B$ | |
| | | P1_ALTSEL1.P7 = $1_B$ | |
| P4.0/RXDC0_3 | NPCR0.RXSEL = $011_B$ | P4_DIR.P0 = $0_B$ | Input |
| P4.1/TXDC0_3 | – | P4_DIR.P1 = $1_B$ | Output |
| | | P4_ALTSEL0.P1 = $1_B$ | |
| | | P4_ALTSEL1.P1 = $1_B$ | |
| P0.1/RXDC1_0 | NPCR1.RXSEL = $000_B$ | P0_DIR.P1 = $0_B$ | Input |
| P0.2/TXDC1_0 | – | P0_DIR.P2 = $1_B$ | Output |
| | | P0_ALTSEL0.P2 = $1_B$ | |
| | | P0_ALTSEL1.P2= $1_B$ | |
| P3.2/RXDC1_1 | NPCR1.RXSEL = $001_B$ | P3_DIR.P2 = $0_B$ | Input |

**Table 16-4 CAN I/O Control Selection** (cont'd) (cont'd)

| Port Lines | PISEL Register Bit | Input/Output Control Register Bits | I/O |
|---|---|---|---|
| P3.3/TXDC1_1 | – | P3_DIR.P3 = $1_B$ | Output |
| | | P3_ALTSEL0.P3 = $1_B$ | |
| | | P3_ALTSEL1.P3 = $1_B$ | |
| P1.4/RXDC1_3 | NPCR1.RXSEL = $001_B$ | P1_DIR.P4 = $0_B$ | Input |
| P1.3/TXDC1_3 | – | P1_DIR.P3 = $1_B$ | Output |
| | | P1_ALTSEL0.P3 = $1_B$ | |
| | | P1_ALTSEL1.P3 = $1_B$ | |

### 16.1.12 Low Power Mode

If the MultiCAN functionality is not required at all, it can be completely disabled by gating off its clock input for maximal power reduction. This is done by setting bit CAN_DIS in register PMCON1 as described below. Refer to **Chapter 8.1.4** for details on peripheral clock management.

**PMCON1**
**Power Mode Control Register 1** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **0** | **UART1_DIS** | **CAN_DIS** | **MDU_DIS** | **T2_DIS** | **CCU_DIS** | **SSC_DIS** | **ADC_DIS** |
| r | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **CAN_DIS** | 5 | rw | **CAN Disable Request. Active high**<br>0 CAN is in normal operation (default).<br>1 CAN is disabled. |
| **0** | 7 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

## 16.2 Registers Description

This section describes the registers of the MultiCAN module. All MultiCAN register names described in this section are also referenced in other parts of the User's Manual by the module name prefix "CAN_".

### MultiCAN Kernel Register Overview

The MultiCAN Kernel include three blocks of registers:

- Global Module Registers
- Node Registers, for each CAN node x
- Message Object Registers, for each message object n

**Table 16-5 Registers Overview - MultiCAN Kernel Registers**

| Register Short Name | Register Long Name | Offset Address[1)] | Description see |
|---|---|---|---|
| **Global Module Registers** | | | |
| LISTm | List Register m | $0100_H + m \times 4_H$ | Page 16-54 |
| MSPNDk | Message Pending Register k | $0120_H + k \times 4_H$ | Page 16-56 |
| MSIDk | Message Index Register k | $0140_H + k \times 4_H$ | Page 16-57 |
| MSIMASK | Message Index Mask Register | $01C0_H$ | Page 16-58 |
| PANCTR | Panel Control Register | $01C4_H$ | Page 16-48 |
| MCR | Module Control Register | $01C8_H$ | Page 16-52 |
| MITR | Module Interrupt Trigger Reg. | $01CC_H$ | Page 16-53 |
| **Node Registers** | | | |
| NCRx | Node x Control Register | $0200_H + x \times 100_H$ | Page 16-59 |
| NSRx | Node x Status Register | $0204_H + x \times 100_H$ | Page 16-63 |
| NIPRx | Node x Interrupt Pointer Reg. | $0208_H + x \times 100_H$ | Page 16-66 |
| NPCRx | Node x Port Control Register | $020C_H + x \times 100_H$ | Page 16-68 |
| NBTRx | Node x Bit Timing Register | $0210_H + x \times 100_H$ | Page 16-69 |
| NECNTx | Node x Error Counter Register | $0214_H + x \times 100_H$ | Page 16-71 |
| NFCRx | Node x Frame Counter Register | $0218_H + x \times 100_H$ | Page 16-72 |
| **Message Object Registers** | | | |
| MOFCRn | Message Object n Function Control Register | $1000_H + n \times 20_H$ | Page 16-86 |
| MOFGPRn | Message Object n FIFO/Gateway Pointer Register | $1004_H + n \times 20_H$ | Page 16-90 |

**Table 16-5 Registers Overview - MultiCAN Kernel Registers** (cont'd)

| Register Short Name | Register Long Name | Offset Address[1)] | Description see |
|---|---|---|---|
| MOIPRn | Message Object n Interrupt Pointer Register | $1008_H + n \times 20_H$ | Page 16-84 |
| MOAMRn | Message Object n Acceptance Mask Register | $100C_H + n \times 20_H$ | Page 16-91 |
| MODATALn | Message Object n Data Register Low | $1010_H + n \times 20_H$ | Page 16-95 |
| MODATAHn | Message Object n Data Register High | $1014_H + n \times 20_H$ | Page 16-96 |
| MOARn | Message Object n Arbitration Register | $1018_H + n \times 20_H$ | Page 16-92 |
| MOCTRn<br>MOSTATn | Message Object n Control Reg.<br>Message Object n Status Reg. | $101C_H + n \times 20_H$ | Page 16-76<br>Page 16-79 |

1) *The following ranges for parameters m, k, x, and n are valid:m = 7-0, k = 1-0, x = 1-0, n = 31-0*

### MultiCAN Access Mediator Register Overview

Table 16-6 shows the addresses (non-mapped) of the following MultiCAN Access Mediator SFRs.

**Table 16-6 MultiCAN Register Mapping**

| Register Name | Physical Address | Description See |
|---|---|---|
| **CAN_DATA3** | $DE_H$ (non mapped) | Page 16-100 |
| **CAN_DATA2** | $DD_H$ (non mapped) | Page 16-99 |
| **CAN_DATA1** | $DC_H$ (non mapped) | Page 16-99 |
| **CAN_DATA0** | $DB_H$ (non mapped) | Page 16-99 |
| **CAN_ADH** | $DA_H$ (non mapped) | Page 16-98 |
| **CAN_ADL** | $D9_H$ (non mapped) | Page 16-98 |
| **CAN_ADCON** | $D8_H$ (non mapped) | Page 16-97 |

**Figure 16-15** shows the MultiCAN kernel register address map.

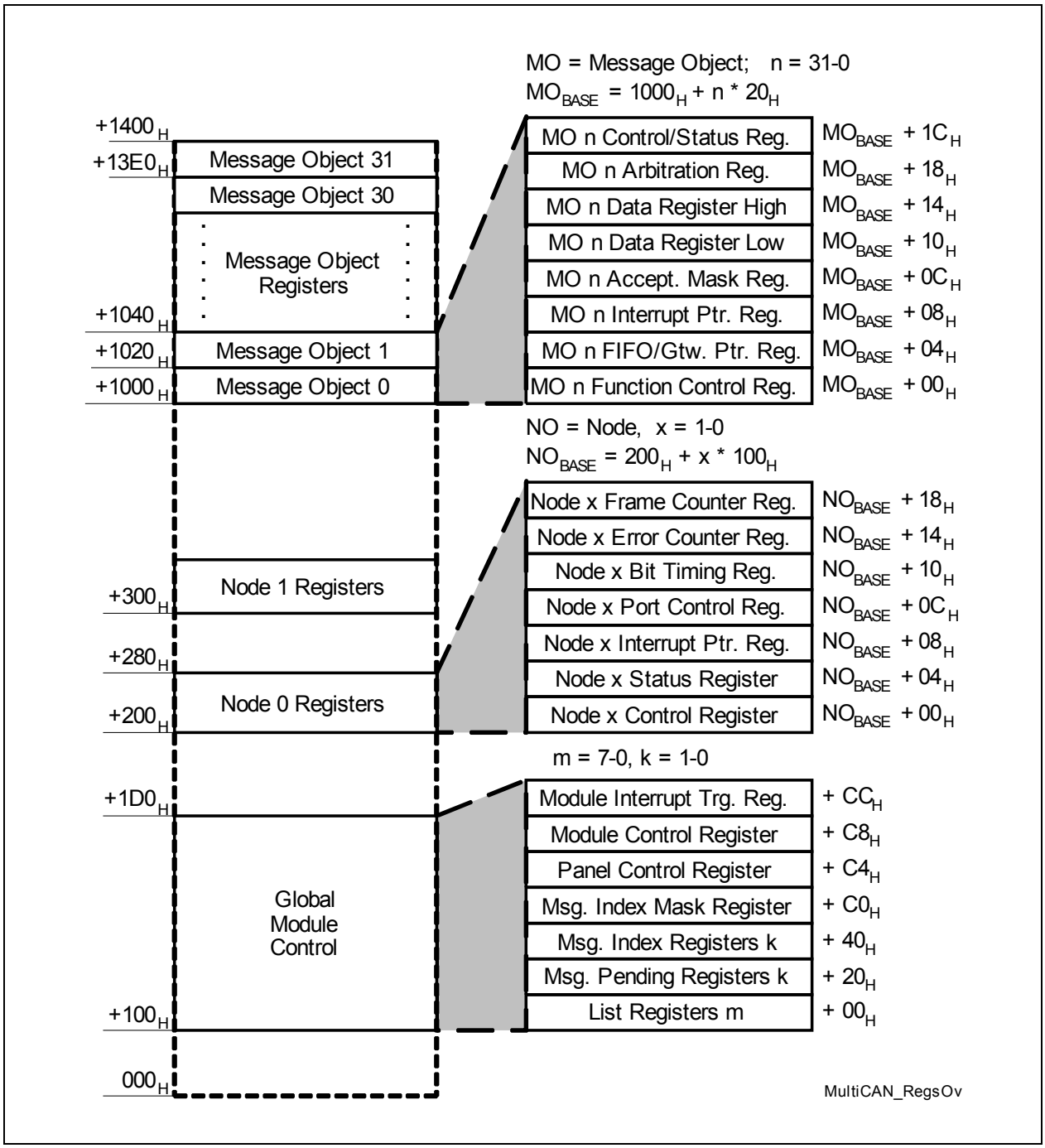


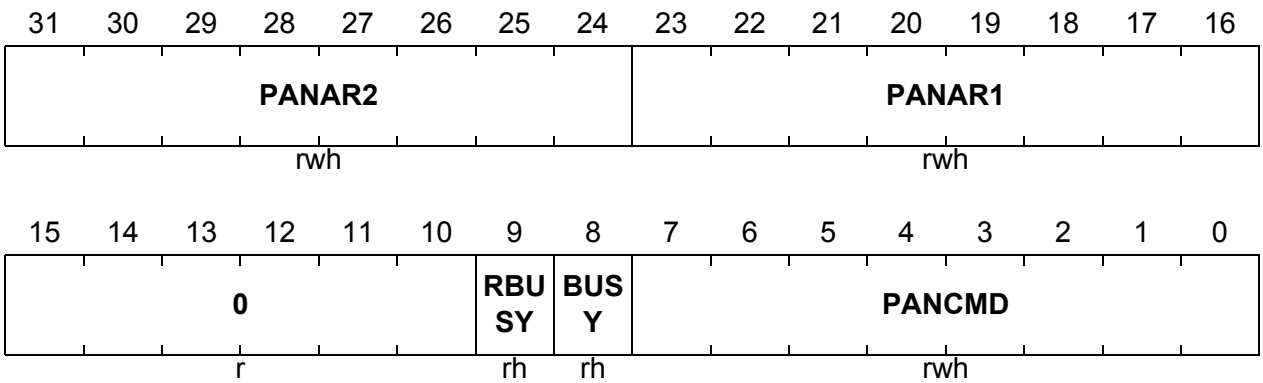

**Figure 16-15 MultiCAN Kernel Register Address Map**

### 16.2.1 Global Module Registers

All list operations such as allocation, de-allocation and relocation of message objects within the list structure are performed via the Command Panel. It is not possible to modify the list structure directly by software by writing to the message objects and the LIST registers.

The Panel Control Register PANCTR is used to start a new command by writing the command arguments and the command code into its bit fields.

**PANCTR**
**Panel Control Register** **Reset Value: 0000 0301$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **PANCMD** | [7:0] | rwh | **Panel Command**<br>This bit field is used to start a new command by writing a panel command code into it. At the end of a panel command, the NOP (no operation) command code is automatically written into PANCMD. The coding of PANCMD is defined in **Table 16-7**. |
| **BUSY** | 8 | rh | **Panel Busy Flag**<br>0 Panel has finished command and is ready to accept a new command.<br>1 Panel operation is in progress. |
| **RBUSY** | 9 | rh | **Result Busy Flag**<br>0 No update of PANAR1 and PANAR2 is scheduled by the list controller.<br>1 A list command is running (BUSY = 1) that will write results to PANAR1 and PANAR2, but the results are not yet available. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **PANAR1** | [23:16] | rwh | **Panel Argument 1**<br>See **Table 16-7**. |
| **PANAR2** | [31:24] | rwh | **Panel Argument 2**<br>See **Table 16-7**. |
| **0** | [15:10] | r | **Reserved**<br>Read as 0; should be written with 0. |

### Panel Commands

A panel operation consists of a command code (PANCMD) and up to two panel arguments (PANAR1, PANAR2). Commands that have a return value deliver it to the PANAR1 bit field. Commands that return an error flag deliver it to bit 31 of the Panel Control Register, this means bit 7 of PANAR2.

**Table 16-7 Panel Commands**

| PANCMD | PANAR2 | PANAR1 | Command Description |
|---|---|---|---|
| $00_H$ | – | – | **No Operation**<br>Writing $00_H$ to PANCMD has no effect. No new command is started. |
| $01_H$ | **Result:**<br>Bit 7: ERR<br>Bit 6-0: undefined | – | **Initialize Lists**<br>Run the initialization sequence to reset the CTRL and LIST fields of all message objects. List registers LIST[7:0] are set to their reset values. This results in the de-allocation of all message objects.<br>The initialization command requires that bits NCRx.INIT and NCRx.CCE are set for all CAN nodes (x = 0-1).<br>Bit 7 of PANAR2 (ERR) reports the success of the operation:<br>0 Initialization was successful<br>1 Not all NCRx.INIT and NCRx.CCE bits are set. Therefore, no initialization is performed.<br>The initialized list command is automatically performed with each reset of the MultiCAN module, but with the exception that all message object registers are reset. |

**Table 16-7 Panel Commands** (cont'd)

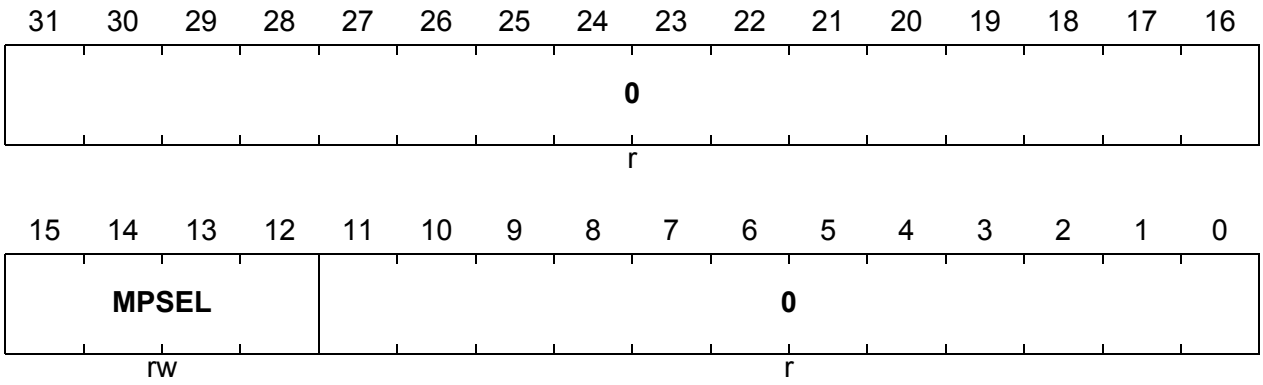
| PANCMD | PANAR2 | PANAR1 | Command Description |
|---|---|---|---|
| $02_H$ | **Argument:**<br>List Index | **Argument:**<br>Message Object Number | **Static Allocate**<br>Allocate message object to a list. The message object is removed from the list that it currently belongs to and appended to the end of the list given by PANAR2. This command is also used to deallocate a message object. In this case, the target list is the list of unallocated elements (PANAR2 = 0). |
| $03_H$ | **Argument:**<br>List Index<br>**Result:**<br>Bit 7: ERR<br>Bit 6-0: undefined | **Result:**<br>Message Object Number | **Dynamic Allocate**<br>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.<br>An ERR bit (bit 7 of PANAR2) reports the success of the operation:<br>0 Success.<br>1 The operation has not been performed because the list of unallocated elements was empty. |
| $04_H$ | **Argument:**<br>Destination Object Number | **Argument:**<br>Source Object Number | **Static Insert Before**<br>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.<br>The source object thus becomes the predecessor of the destination object. |

**Table 16-7 Panel Commands** (cont'd)

| PANCMD | PANAR2 | PANAR1 | Command Description |
|---|---|---|---|
| $05_H$ | **Argument:**<br>Destination Object Number<br>**Result:**<br>Bit 7: ERR<br>Bit 6-0: undefined | **Result:**<br>Object Number of inserted object | **Dynamic Insert Before**<br>Insert a new message object before a given destination object. The new object is taken from the list of unallocated elements (the first element is chosen). The number of the new object is delivered as a result to PANAR1.<br>An ERR bit (bit 7 of PANAR2) reports the success of the operation:<br>0 Success.<br>1 The operation has not been performed because the list of unallocated elements was empty. |
| $06_H$ | **Argument:**<br>Destination Object Number | **Argument:**<br>Source Object Number | **Static Insert Behind**<br>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.<br>The source object thus becomes the successor of the destination object. |
| $07_H$ | **Argument:**<br>Destination Object Number<br>**Result:**<br>Bit 7: ERR<br>Bit 6-0: undefined | **Result:**<br>Object Number of inserted object | **Dynamic Insert Behind**<br>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.<br>An ERR bit (bit 7 of PANAR2) reports the success of the operation:<br>0 Success.<br>1 The operation has not been performed because the list of unallocated elements was empty. |
| $08_H$ - $FF_H$ | – | – | **Reserved** |

The Module Control Register MCR contains basic settings that define the operation of the MultiCAN module.

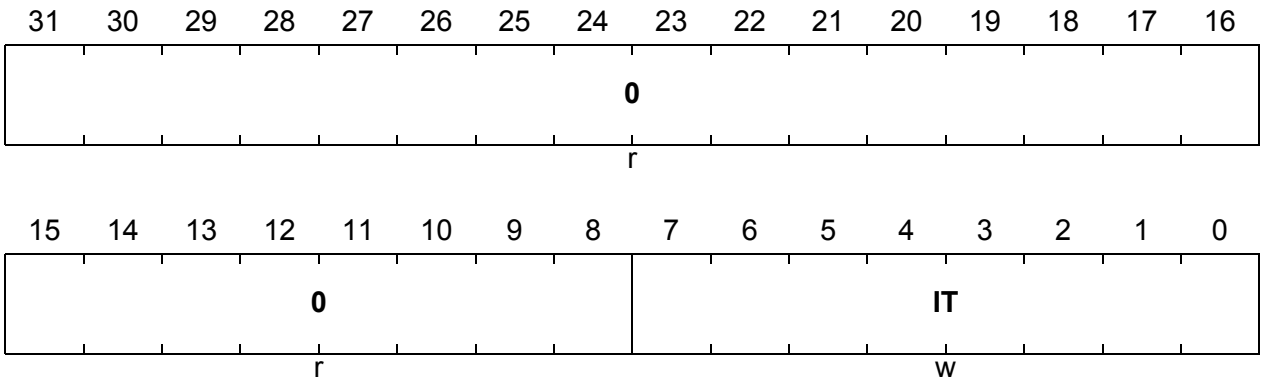
**MCR**
**Module Control Register** **Reset Value: 0000 0000$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **MPSEL** | [15:12] | rw | **Message Pending Selector**<br>Bit field MPSEL allows the bit position of the message pending bit to be selected after a message reception/transmission by a mixture of the MOIPRn register bit fields RXINP, TXINP, and MPN. Selection details are given in **Figure 16-10** on **Page 16-25**. |
| **0** | [31:16], [11:0] | r | **Reserved**<br>Read as 0; should be written with 0. |

The Interrupt Trigger Register ITR allows interrupt requests to be triggered on each interrupt output line by software.

**MITR**
**Module Interrupt Trigger Register** **Reset Value: 0000 0000$_H$**



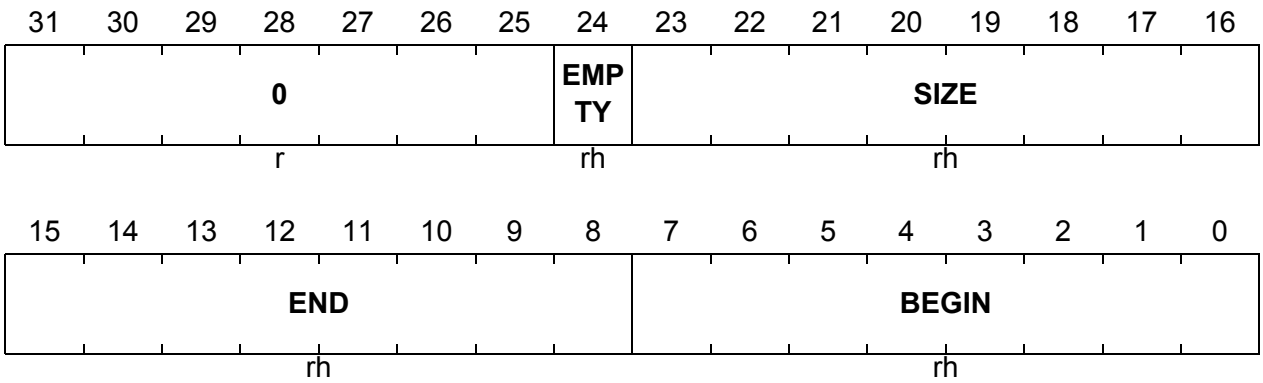

| Field | Bits | Type | Description |
|---|---|---|---|
| **IT** | [7:0] | w | **Interrupt Trigger**<br>Writing a 1 to IT[n] (n = 0-7) generates an interrupt request on interrupt output line CANSRC[n]. Writing a 0 to IT[n] has no effect. Bit field IT is always read as 0. Multiple interrupt requests can be generated with a single write operation to MITR by writing a 1 to several bit positions of IT. |
| **0** | [31:8] | r | **Reserved**<br>Read as 0; should be written with 0. |

### List Pointer and List Register

Each of the two CAN nodes has a list which defines the allocated message objects. Additionally, a list of all unallocated objects is available. Further, general purpose lists are available which are not associated to a CAN node. The List Registers are assigned in the following way:

- LIST0 defines the list of all unallocated objects
- LIST1 defines the list for CAN node 0
- LIST2 defines the list for CAN node 1
- LIST[7:3] are not associated to a CAN node (free lists)

**LIST0**
**List Register 0** **Reset Value: 001F 1F00$_H$**
**LISTm (m = 1-7)**
**List Register m** **Reset Value: 0100 0000$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **BEGIN** | [7:0] | rh | **List Begin**<br>BEGIN indicates the number of the first message object in list m. |
| **END** | [15:8] | rh | **List End**<br>END indicates the number of the last message object in list m. |
| **SIZE** | [23:16] | rh | **List Size**<br>SIZE indicates the number of elements in the list m.<br>SIZE = number of list elements - 1 |
| **EMPTY** | 24 | rh | **List Empty Indication**<br>0 At least one message object is allocated to list m.<br>1 No message object is allocated to the list m. List m is empty. |

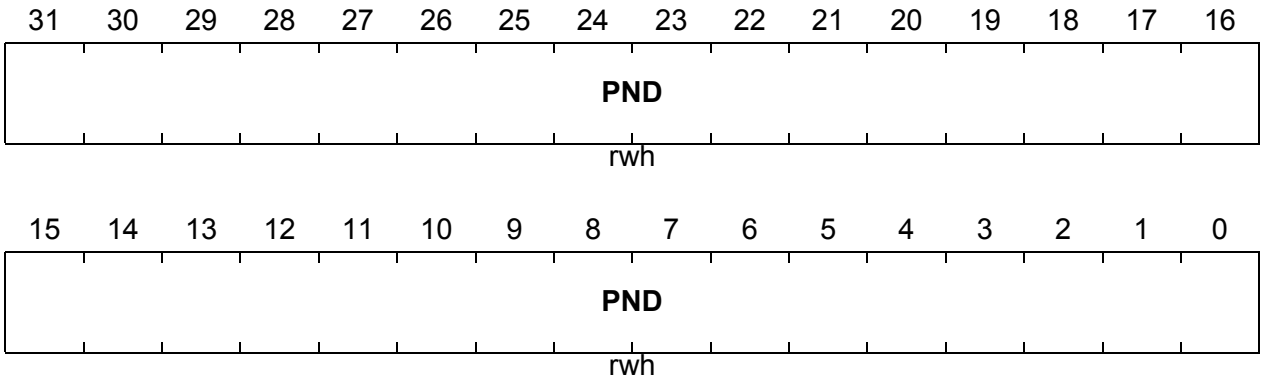
| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | [31:25] | r | **Reserved**<br>ead as 0; should be written with 0. |

### Message Notifications

When a message object n generates an interrupt request upon the transmission or reception of a message, then the request is routed to the interrupt output line selected by the bit field MOIPRn.TXIPND or MOIPRn.RXIPND of the message object n. As there are more message objects than interrupt output lines, an interrupt routine typically processes requests from more than one message object. Therefore, a priority selection mechanism is implemented in the MultiCAN module to select the highest priority object within a collection of message objects.

The Message Pending Register MSPNDk contains the pending interrupt notification of list m.

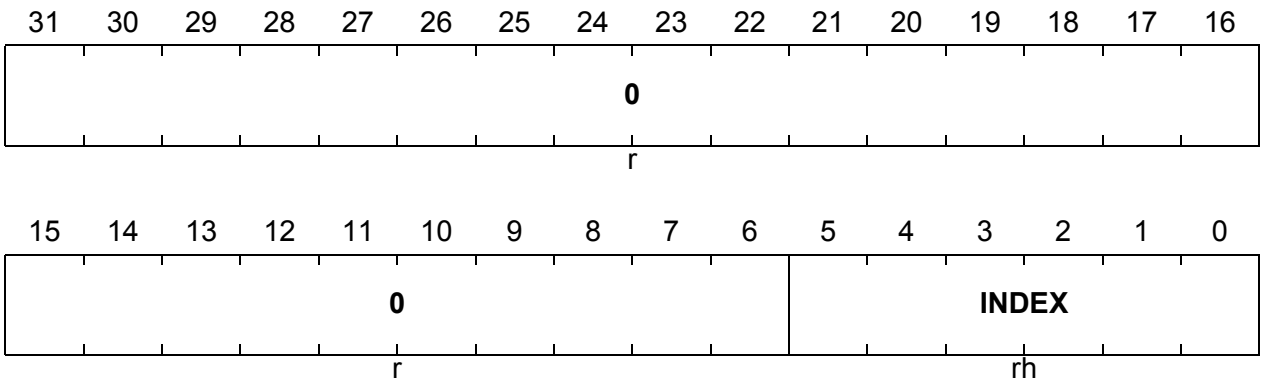
**MSPNDk (k = 0-1)**
**Message Pending Register k** **Reset Value: 0000 0000$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **PND** | [31:0] | rwh | **Message Pending**<br>When a message interrupt occurs, the message object sets a bit in one of the MSPND register, where the bit position is given by the MPN[4:0] field of the IPR register of the message object. The register selection k is given by the bit 5 of MPN.<br>The register bits can be cleared by software (write 0). Writing a 1 has no effect. |

Each Message Pending Register has a Message Index Register MSIDk associated with it. The Message Index Register shows the active (set) pending bit with lowest bit position within groups of pending bits.

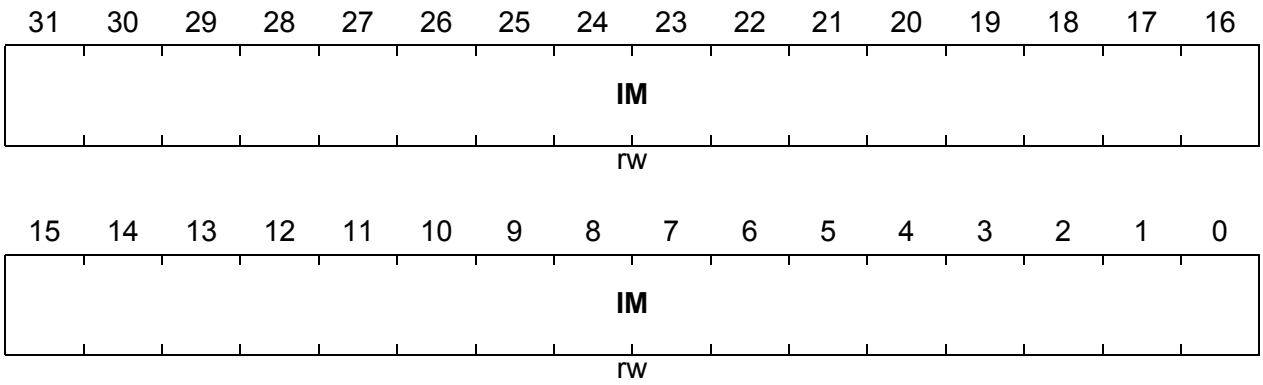
**MSIDk (k = 0-1)**
**Message Index Register k** **Reset Value: 0000 0020$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **INDEX** | [5:0] | rh | **Message Pending Index**<br>The value of INDEX is given by the bit position i of the pending bit of MSPNDk with the following properties:<br>1. MSPNDk[i] & IM[i] = 1<br>2. i = 0 or MSPNDk[i-1:0] & IM[i-1:0] = 0<br>If no bit of MSPNDk satisfies these conditions then INDEX reads $100000_B$.<br>Thus INDEX shows the position of the first pending bit of MSPNDk, in which only those bits of MSPNDk that are selected in the Message Index Mask Register are taken into account. |
| **0** | [31:6] | r | **Reserved**<br>Read as 0; should be written with 0. |

The Message Index Mask Register MSIMASK selects individual bits for the calculation of the Message Pending Index. The Message Index Mask Register is used commonly for all Message Pending registers and their associated Message Index registers.

**MSIMASK**
**Message Index Mask Register** **Reset Value: 0000 0000$_H$**



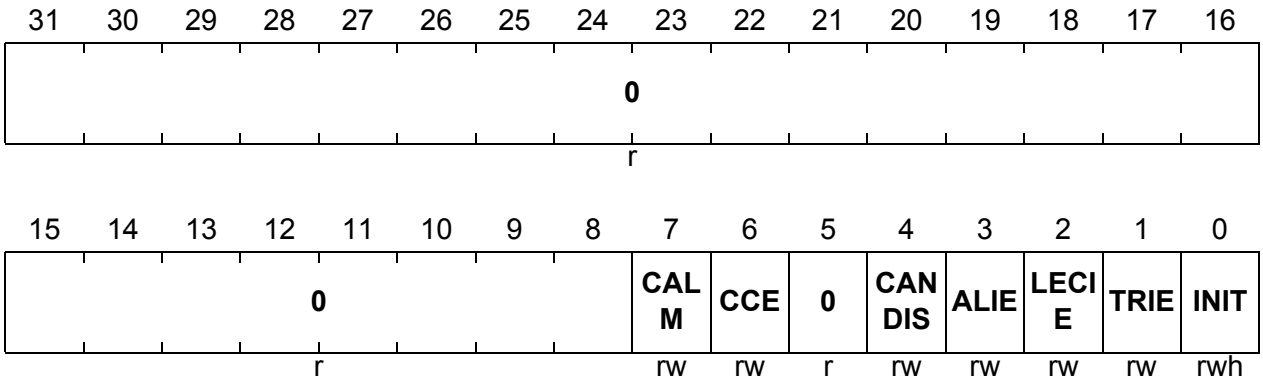

| Field | Bits | Type | Description |
|---|---|---|---|
| **IM** | [31:0] | rw | **Message Index Mask**<br>Only those bits in MSPNDk for which the corresponding Index Mask bits are set contribute to the calculation of the Message Index. |

### 16.2.2 CAN Node Registers

The CAN node registers are built in for each CAN node of the MultiCAN module. They contain information that is directly related to the operation of the CAN nodes and are shared among the nodes.

The Node Control Register NCRx contains basic settings that define the operation of the CAN node.

**NCRx (x = 0-1)**
**Node x Control Register** **Reset Value: 0000 0001$_H$**



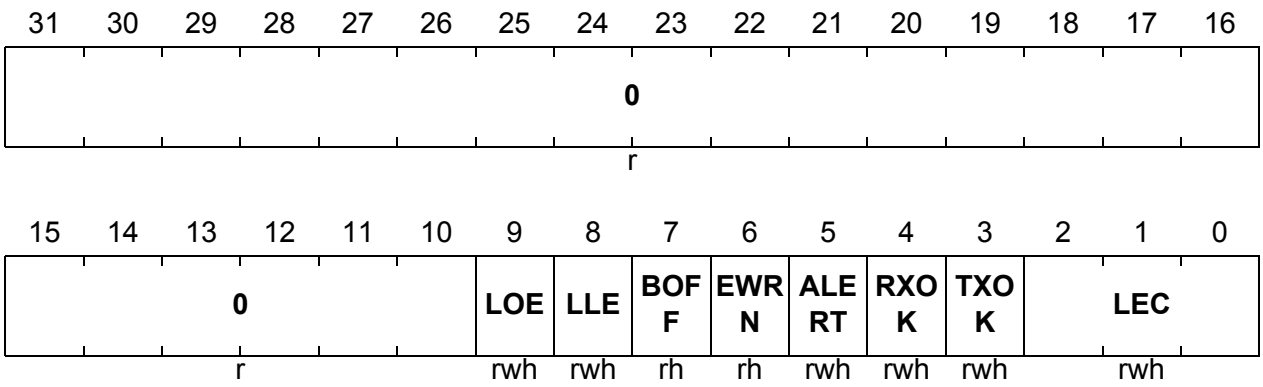
| Field | Bits | Type | Description |
|---|---|---|---|
| **INIT** | 0 | rwh | **Node Initialization**<br>0 Resetting bit INIT enables the participation of the node in the CAN traffic.<br>If the CAN node is in the bus-off state then the ongoing bus-off recovery (which does not depend on the INIT bit) is continued. With the end of the bus-off recovery sequence, the CAN node is allowed to take part in the CAN traffic.<br>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.<br>1 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.<br>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 $\times$ 1) then the CAN node leaves the bus-off state but remains inactive as long as INIT remains set.<br>Bit INIT is automatically set when the CAN node enters the bus-off state. |
| **TRIE** | 1 | rw | **Transfer Interrupt Enable**<br>TRIE enables the transfer interrupt of CAN node x. This interrupt is generated after the successful reception or transmission of a CAN frame in node x.<br>0 Transfer interrupt is disabled.<br>1 Transfer interrupt is enabled.<br>Bit field NIPRx.TRINP selects the interrupt output line which becomes activated at this type of interrupt. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **LECIE** | 2 | rw | **LEC Indicated Error Interrupt Enable**<br>LECIE enables the last error code interrupt of CAN node x. This interrupt is generated with each update of bit field NSRx.LEC with LEC > 0 (CAN protocol error).<br>0 Last error code interrupt is disabled.<br>1 Last error code interrupt is enabled.<br>Bit field NIPRx.LECINP selects the interrupt output line which becomes activated at this type of interrupt. |
| **ALIE** | 3 | rw | **Alert Interrupt Enable**<br>ALIE enables the alert interrupt of CAN node x. This interrupt is generated by any one of the following events:<br>• A change of bit NSRx.BOFF<br>• A change of bit NSRx.EWRN<br>• A List Length Error, which also sets bit NSRx.LLE<br>• A List Object Error, which also sets bit NSRx.LOE<br>• A Bit INIT is set by hardware<br>0 Alert interrupt is disabled.<br>1 Alert interrupt is enabled.<br>Bit field NIPRx.ALINP selects the interrupt output line which becomes activated at this type of interrupt. |
| **CANDIS** | 4 | rw | **CAN Disable**<br>Setting this bit disables the CAN node. The CAN node first waits until it is bus-idle or bus-off. Then bit INIT is automatically set, and an alert interrupt is generated if bit ALIE is set. |
| **CCE** | 6 | rw | **Configuration Change Enable**<br>0 The Bit Timing Register, the Port Control Register, and the Error Counter Register may only be read. All attempts to modify them are ignored.<br>1 The Bit Timing Register, the Port Control Register, and the Error Counter Register may be read and written. |
| **CALM** | 7 | rw | **CAN Analyze Mode**<br>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.<br>Bit CALM can be written only while bit INIT is set. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | [31:8], 5 | r | **Reserved**<br>Read as 0; should be written with 0. |

The Node Status Register NSRx reports errors as well as successfully transferred CAN frames.

**NSRx (x = 0-1)**
**Node x Status Register** **Reset Value: 0000 0000$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **LEC** | [2:0] | rwh | **Last Error Code**<br>This bit field indicates the type of the last (most recent) CAN error. The encoding of this bit field is described in **Table 16-8**. |
| **TXOK** | 3 | rwh | **Message Transmitted Successfully**<br>0 No successful transmission since last (most recent) flag reset.<br>1 A message has been transmitted successfully (error-free and acknowledged by at least another node).<br>TXOK must be reset by software (write 0). Writing 1 has no effect. |
| **RXOK** | 4 | rwh | **Message Received Successfully**<br>0 No successful reception since last (most recent) flag reset.<br>1 A message has been received successfully.<br>RXOK must be reset by software (write 0). Writing 1 has no effect. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **ALERT** | 5 | rwh | **Alert Warning**<br>The ALERT bit is set upon the occurrence of one of the following events (the same events which also trigger an alert interrupt if NCRx.ALIE is set):<br>• A change of bit NSRx.BOFF<br>• A change of bit NSRx.EWRN<br>• A List Length Error, which also sets bit NSRx.LLE<br>• A List Object Error, which also sets bit NSRx.LOE<br>• Bit INIT has been set by hardware<br>ALERT must be reset by software (write 0). Writing 1 has no effect. |
| **EWRN** | 6 | rh | **Error Warning Status**<br>0 No warning limit exceeded.<br>1 One of the error counters NECNTx.REC or NECNTx.TEC reached the warning limit NECNTx.EWRNLVL. |
| **BOFF** | 7 | rh | **Bus-off Status**<br>0 CAN controller is not in the bus-off state.<br>1 CAN controller is in the bus-off state. |
| **LLE** | 8 | rwh | **List Length Error**<br>0 No List Length Error since last (most recent) flag reset.<br>1 A List Length Error has been detected during message acceptance filtering. The number of elements in the list that belongs to this CAN node differs from the list SIZE given in the list termination pointer.<br>LLE must be reset by software (write 0). Writing 1 has no effect. |
| **LOE** | 9 | rwh | **List Object Error**<br>0 No List Object Error since last (most recent) flag reset.<br>1 A List Object Error has been detected during message acceptance filtering. A message object with wrong LIST index entry in the Message Object Control Register has been detected.<br>LOE must be reset by software (write 0). Writing 1 has no effect. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | [31:10] | r | **Reserved**<br>Read as 0; should be written with 0. |

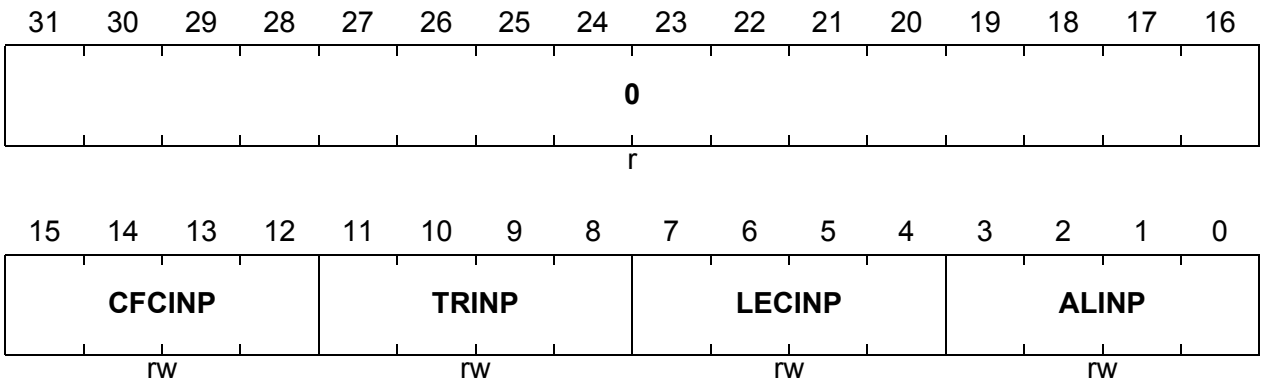
### Encoding of the LEC Bit Field

**Table 16-8 Encoding of the LEC Bit Field**

| LEC Value | Signification |
|---|---|
| $000_B$ | **No Error:**<br>No error was detected for the last (most recent) message on the CAN bus. |
| $001_B$ | **Stuff Error:**<br>More than 5 equal bits in a sequence have occurred in a part of a received message where this is not allowed. |
| $010_B$ | **Form Error:**<br>A fixed format part of a received frame has the wrong format. |
| $011_B$ | **Ack Error:**<br>The transmitted message was not acknowledged by another node. |
| $100_B$ | **Bit1 Error:**<br>During a message transmission, the CAN node tried to send a recessive level (1) outside the arbitration field and the acknowledge slot, but the monitored bus value was dominant. |
| $101_B$ | **Bit0 Error:**<br>Two different conditions are signaled by this code:<br>1. During transmission of a message (or acknowledge bit, active-error flag, overload flag), the CAN node tried to send a dominant level (0), but the monitored bus value was recessive.<br>2. During bus-off recovery, this code is set each time a sequence of 11 recessive bits has been monitored. The CPU may use this code as indication that the bus is not continuously disturbed. |
| $110_B$ | **CRC Error:**<br>The CRC checksum of the received message was incorrect. |
| $111_B$ | **CPU write to LEC:**<br>Whenever the CPU writes the value $111_B$ to LEC, it takes the value $111_B$. Whenever the CPU writes another value to LEC, the written LEC value is ignored. |

The four interrupt pointers in the NIPR register select one out of the eight interrupt outputs individually for each type of CAN node interrupt. See also Page 16-11 for more CAN node interrupt details.

**NIPRx (x = 0-1)**
**Node x Interrupt Pointer Register** **Reset Value: 0000 0000$_H$**

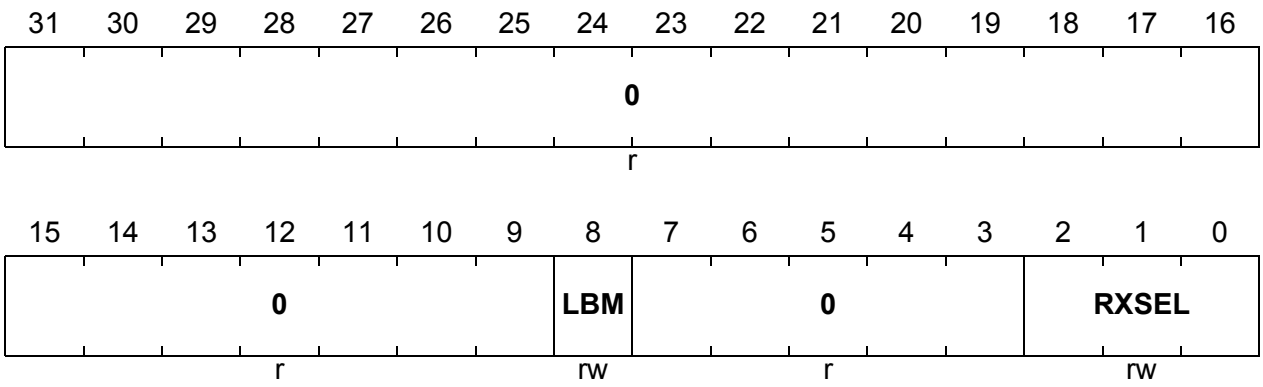



| Field | Bits | Type | Description |
|---|---|---|---|
| **ALINP** | [3:0] | rw | **Alert Interrupt Node Pointer**<br>ALINP selects the interrupt output line CANSRCm (m = 0-7) for an alert interrupt of CAN Node x.<br>$0000_B$ Interrupt output line CANSRC0 is selected.<br>$0001_B$ Interrupt output line CANSRC1 is selected.<br>… …<br>$0111_B$ Interrupt output line CANSRC7 is selected.<br>$1000_B$-$1111_B$ Reserved |
| **LECINP** | [7:4] | rw | **Last Error Code Interrupt Node Pointer**<br>LECINP selects the interrupt output line CANSRCm (m = 0-7) for an LEC interrupt of CAN Node x.<br>$0000_B$ Interrupt output line CANSRC0 is selected.<br>$0001_B$ Interrupt output line CANSRC1 is selected.<br>… …<br>$0111_B$ Interrupt output line CANSRC7 is selected.<br>$1000_B$-$1111_B$IReserved |

| Field | Bits | Type | Description |
|---|---|---|---|
| **TRINP** | [11:8] | rw | **Transfer OK Interrupt Node Pointer**<br>TRINP selects the interrupt output line CANSRCm (m = 0-7) for a transfer OK interrupt of CAN Node x.<br>$0000_B$ Interrupt output line CANSRC0 is selected.<br>$0001_B$ Interrupt output line CANSRC1 is selected.<br>… …<br>$0111_B$ Interrupt output line CANSRC7 is selected.<br>$1000_B$-$1111_B$IReserved |
| **CFCINP** | [15:12] | rw | **Frame Counter Interrupt Node Pointer**<br>CFCINP selects the interrupt output line CANSRCm (m = 0-7) for a frame counter overflow interrupt of CAN Node x.<br>$0000_B$ Interrupt output line CANSRC0 is selected.<br>$0001_B$ Interrupt output line CANSRC1 is selected.<br>… …<br>$0111_B$ Interrupt output line CANSRC7 is selected.<br>$1000_B$-$1111_B$Reserved |
| **0** | [31:16] | r | **Reserved**<br>Read as 0; should be written with 0. |

The Node Port Control Register NPCRx configures the CAN bus transmit/receive ports. NPCRx can be written only if bit NCRx.CCE is set.

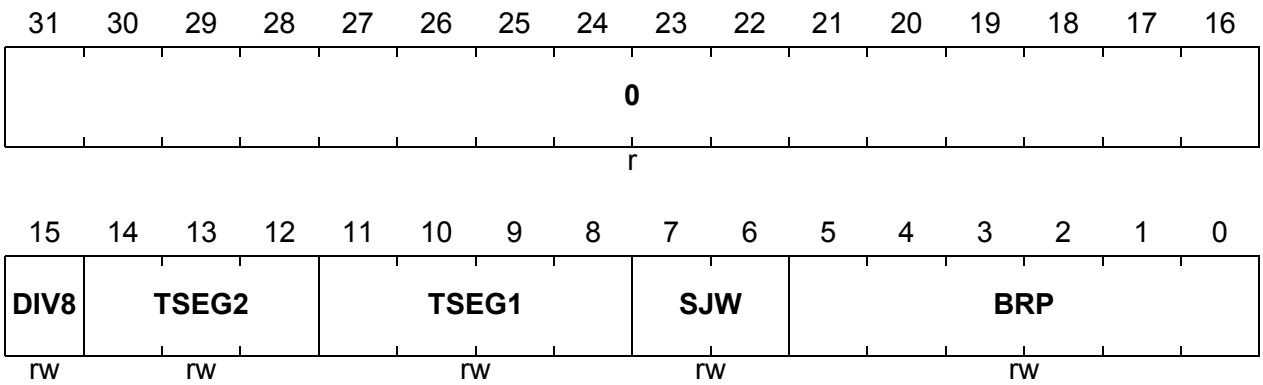
**NPCRx (x = 0-1)**
**Node x Port Control Register** **Reset Value: 0000 0000$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **RXSEL** | [2:0] | rw | **Receive Select**<br>RXSEL selects one out of 8 possible receive inputs. The CAN receive signal is performed only through the selected input.<br>*Note: In XC878, only specific combinations of RXSEL are available (see also **Page 16-43**).* |
| **LBM** | 8 | rw | **Loop-Back Mode**<br>0 Loop-Back Mode is disabled.<br>1 Loop-Back Mode is enabled. This node is connected to an internal (virtual) loop-back CAN bus. All CAN nodes which are in Loop-Back Mode are connected to this virtual CAN bus so that they can communicate with each other internally. The external transmit line is forced recessive in Loop-Back Mode. |
| **0** | [7:3], [31:9] | r | **Reserved**<br>Read as 0; should be written with 0. |

The Node Bit Timing Register NBTRx contains all parameters to set up the bit timing for the CAN transfer. NBTRx can be written only if bit NCRx.CCE is set.

**NBTRx (x = 0-1)**
**Node x Bit Timing Register** **Reset Value: 0000 0000$_H$**

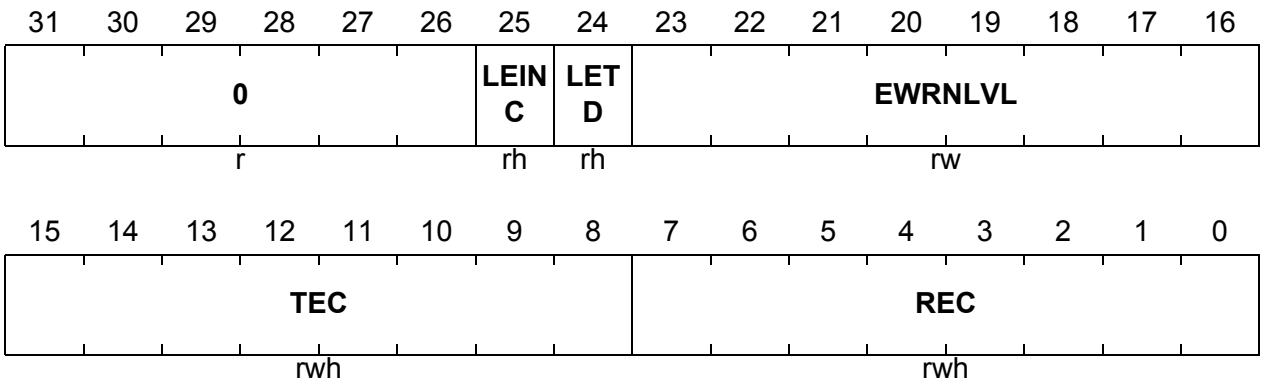



| Field | Bits | Type | Description |
|---|---|---|---|
| **BRP** | [5:0] | rw | **Baud Rate Prescaler**<br>The duration of one time quantum is given by (BRP + 1) clock cycles if DIV8 = 0.<br>The duration of one time quantum is given by 8 × (BRP + 1) clock cycles if DIV8 = 1. |
| **SJW** | [7:6] | rw | **(Re) Synchronization Jump Width**<br>(SJW + 1) time quanta are allowed for re-synchronization. |
| **TSEG1** | [11:8] | rw | **Time Segment Before Sample Point**<br>(TSEG1 + 1) time quanta is the user-defined nominal time between the end of the synchronization segment and the sample point. It includes the propagation segment, which takes into account signal propagation delays. The time segment may be lengthened due to re-synchronization.<br>Valid values for TSEG1 are 2 to 15. |
| **TSEG2** | [14:12] | rw | **Time Segment After Sample Point**<br>(TSEG2 + 1) time quanta is the user-defined nominal time between the sample point and the start of the next synchronization segment. It may be shortened due to re-synchronization.<br>Valid values for TSEG2 are 1 to 7. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **DIV8** | 15 | rw | **Divide Prescaler Clock by 8**<br>0 A time quantum lasts (BRP+1) clock cycles.<br>1 A time quantum lasts 8 × (BRP+1) clock cycles. |
| **0** | [31:16] | r | **Reserved**<br>Read as 0; should be written with 0. |

The Node Error Counter Register NECNTx contains the CAN receive and transmit error counter as well as some additional bits to ease error analysis. NECNTx can be written only if bit NCRx.CCE is set.

**NECNTx (x = 0-1)**
**Node x Error Counter Register** **Reset Value: 0060 0000$_H$**

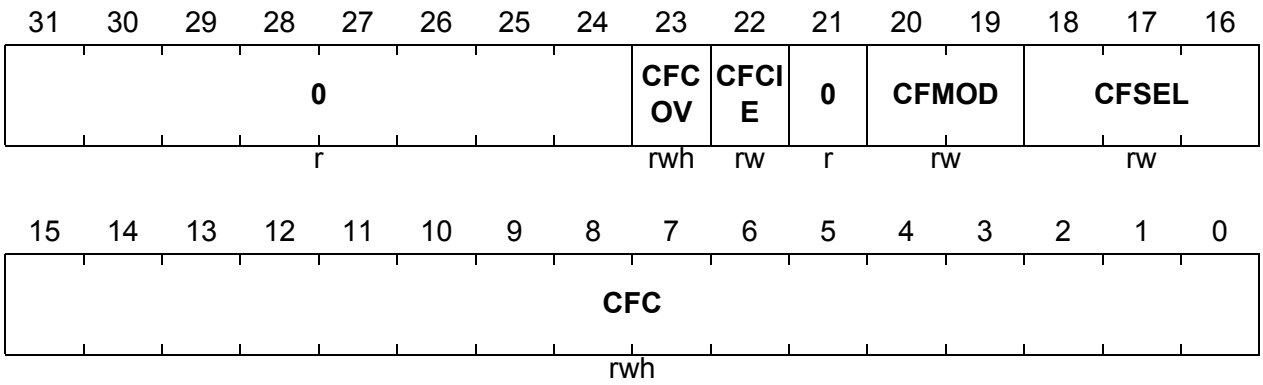



| Field | Bits | Type | Description |
|---|---|---|---|
| **REC** | [7:0] | rwh | **Receive Error Counter**<br>Bit field REC contains the value of the receive error counter of CAN node x. |
| **TEC** | [15:8] | rwh | **Transmit Error Counter**<br>Bit field TEC contains the value of the transmit error counter of CAN node x. |
| **EWRNLVL** | [23:16] | rw | **Error Warning Level**<br>Bit field EWRNLVL defines the threshold value (warning level, default 96) to be reached in order to set the corresponding error warning bit NSRx.EWRN. |
| **LETD** | 24 | rh | **Last Error Transfer Direction**<br>0 The last error occurred while the CAN node x was receiver (REC has been incremented).<br>1 The last error occurred while the CAN node x was transmitter (TEC has been incremented). |
| **LEINC** | 25 | rh | **Last Error Increment**<br>0 The last error led to an error counter increment of 1.<br>1 The last error led to an error counter increment of 8. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | [31:26] | r | **Reserved**<br>Read as 0; should be written with 0. |

The Node Frame Counter Register NFCRx contains the actual value of the frame counter as well as control and status bits of the frame counter.

**NFCRx (x = 0-1)**
**Node x Frame Counter Register** **Reset Value: 0000 0000$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **CFC** | [15:0] | rwh | **CAN Frame Counter**<br>In Frame Count Mode (CFMOD = $00_B$), this bit field contains the frame count value.<br>In Time Stamp Mode (CFMOD = $01_B$), this bit field contains the captured bit time count value, captured with the start of a new frame.<br>In all Bit Timing Analysis Modes (CFMOD = $10_B$), CFC always displays the number of $f_{CAN}$ clock cycles (measurement result) minus 1. Example: a CFC value of 34 in measurement mode CFSEL = $000_B$ means that 35 $f_{CAN}$ clock cycles have been elapsed between the most recent two dominant edges on the receive input. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **CFSEL** | [18:16] | rw | **CAN Frame Count Selection**<br>This bit field selects the function of the frame counter for the chosen frame count mode.<br>**Frame Count Mode**<br>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.<br>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.<br>Bit 2 If Bit 2 of CFSEL is set, then CFC is incremented each time a frame has been transmitted successfully by the node.<br>**Time Stamp Mode**<br>$000_B$ The frame counter is incremented (internally) at the beginning of a new bit time. The value is sampled during the SOF bit of a new frame. The sampled value is visible in the CFC field.<br>**Bit Timing Mode**<br>The available bit timing measurement modes are shown in Table 16-9. If CFCIE is set, then an interrupt on request node x (where x is the CAN node number) is generated with a CFC update. |
| **CFMOD** | [20:19] | rw | **CAN Frame Counter Mode**<br>This bit field determines the operation mode of the frame counter.<br>$00_B$ Frame Count Mode: The frame counter is incremented upon the reception and transmission of frames.<br>$01_B$ Time Stamp Mode: The frame counter is used to count bit times.<br>$10_B$ Bit Timing Mode: The frame counter is used for analysis of the bit timing.<br>$11_B$ Reserved. |
| **CFCIE** | 22 | rw | **CAN Frame Count Interrupt Enable**<br>CFCIE enables the CAN frame counter overflow interrupt of CAN node x.<br>0 CAN frame counter overflow interrupt is disabled.<br>1 CAN frame counter overflow interrupt is enabled.<br>Bit field NIPRx.CFCINP selects the interrupt output line that is activated at this type of interrupt. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **CFCOV** | 23 | rwh | **CAN Frame Counter Overflow Flag**<br>Flag CFCOV is set upon a frame counter overflow (transition from $FFFF_H$ to $0000_H$). In bit timing analysis mode, CFCOV is set upon an update of CFC. An interrupt request is generated if CFCIE = 1.<br>0 No overflow has occurred since last flag reset.<br>1 An overflow has occurred since last flag reset.<br>CFCOV must be reset by software. |
| **0** | 21, [31:24] | r | **Reserved**<br>Read as 0; should be written with 0. |

### Bit Timing Analysis Modes

**Table 16-9 Bit Timing Analysis Modes (CFMOD = 10)**

| CFSEL | Measurement |
|---|---|
| $000_B$ | Whenever a dominant edge (transition from 1 to 0) is monitored on the receive input, the time (measured in clock cycles) between this edge and the most recent dominant edge is stored in CFC. |
| $001_B$ | Whenever a recessive edge (transition from 0 to 1) is monitored on the receive input, the time (measured in clock cycles) between this edge and the most recent dominant edge is stored in CFC. |
| $010_B$ | Whenever a dominant edge is received as a result of a transmitted dominant edge, the time (clock cycles) between both edges is stored in CFC. |
| $011_B$ | Whenever a recessive edge is received as a result of a transmitted recessive edge, the time (clock cycles) between both edges is stored in CFC. |
| $100_B$ | Whenever a dominant edge that qualifies for synchronization is monitored on the receive input, the time (measured in clock cycles) between this edge and the most recent sample point is stored in CFC. |
| $101_B$ | With each sample point, the time (measured in clock cycles) between the start of the new bit time and the start of the previous bit time is stored in CFC[11:0].<br>Additional information is written to CFC[15:12] at each sample point:<br>CFC[15]: Transmit value of actual bit time<br>CFC[14]: Receive sample value of actual bit time<br>CFC[13:12]: CAN bus information (see **Table 16-10**) |
| $111_B$ | Reserved, do not use this combination. |

**Table 16-10 CAN Bus State Information**

| CFC[13:12] | CAN Bus State |
|---|---|
| $00_B$ | **NoBit**<br>The CAN bus is idle, performs bit (de-) stuffing or is in one of the following frame segments:<br>SOF, reserved bits, SRR, CRC, delimiters, first 6 EOF bits, IFS. |
| $01_B$ | **NewBit**<br>This code represents the first bit of a new frame segment.<br>The current bit is the first bit in one of the following frame segments:<br>Bit 10 (MSB) of standard ID (transmit only), RTR, IDE, DLC(MSB), bit 7 (MSB) in each data byte and the first bit of the ID extension. |
| $10_B$ | **Bit**<br>This code represents a bit inside a frame segment with a length of more than one bit (not the first bit of those frame segments that is indicated by NewBit).<br>The current bit is processed within one of the following frame segments:<br>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**<br>The current bit is in one of the following frame segments:<br>Acknowledge slot, last bit of EOF, active/passive-error frame, overload frame.<br>Two or more directly consecutive Done codes signal an Error Frame. |

### 16.2.3 Message Object Registers

The Message Object Control Register MOCTRn and the Message Object Status Register MOSTATn are located at the same address offset within a message object address block (offset address $1C_H$). The MOCTRn is a write-only register that makes it possible to set/reset CAN transfer related control bits through software.

**MOCTR0**
**Message Object 0 Control Register** **Reset Value: 0100 0000$_H$**
**MOCTR31**
**Message Object 31 Control Register** **Reset Value: 1F1E 0000$_H$**
**MOCTRn (n = 1-30)**
**Message Object n Control Register**
**Reset Value: ((n+1)*01000000$_H$)+((n-1)*00010000$_H$)**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | SETDIR | SETTXEN1 | SETTXEN0 | SETTXRQ | SETRXEN | SETRTSEL | SETMSGVAL | SETMSGLST | SETNEWDAT | SETRXUPD | SETTXPND | SETRXPND |
| w | | | | w | w | w | w | w | w | w | w | w | w | w | w |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | RESDIR | RESTXEN1 | RESTXEN0 | RESTXRQ | RESRXEN | RESRTSEL | RESMSGVAL | RESMSGLST | RESNEWDAT | RESRXUPD | RESTXPND | RESRXPND |
| w | | | | w | w | w | w | w | w | w | w | w | w | w | w |

| Field | Bits | Type | Description |
|---|---|---|---|
| **RESRXPND**<br>**SETRXPND** | 0<br>16 | w<br>w | **Reset/Set Receive Pending**<br>These bits control the set/reset condition for RXPND (see **Table 16-11**). |
| **RESTXPND**<br>**SETTXPND** | 1<br>17 | w<br>w | **Reset/Set Transmit Pending**<br>These bits control the set/reset condition for TXPND (see **Table 16-11**). |
| **RESRXUPD**<br>**SETRXUPD** | 2<br>18 | w<br>w | **Reset/Set Receive Updating**<br>These bits control the set/reset condition for RXUPD (see **Table 16-11**). |
| **RESNEWDAT**<br>**SETNEWDAT** | 3<br>19 | w<br>w | **Reset/Set New Data**<br>These bits control the set/reset condition for NEWDAT (see **Table 16-11**). |

| Field | Bits | Type | Description |
|---|---|---|---|
| **RESMSGLST**<br>**SETMSGLST** | 4<br>20 | w<br>w | **Reset/Set Message Lost**<br>These bits control the set/reset condition for MSGLST (see **Table 16-11**). |
| **RESMSGVAL**<br>**SETMSGVAL** | 5<br>21 | w<br>w | **Reset/Set Message Valid**<br>These bits control the set/reset condition for MSGVAL (see **Table 16-11**). |
| **RESRTSEL**<br>**SETRTSEL** | 6<br>22 | w<br>w | **Reset/Set Receive/Transmit Selected**<br>These bits control the set/reset condition for RTSEL (see **Table 16-11**). |
| **RESRXEN**<br>**SETRXEN** | 7<br>23 | w<br>w | **Reset/Set Receive Enable**<br>These bits control the set/reset condition for RXEN (see **Table 16-11**). |
| **RESTXRQ**<br>**SETTXRQ** | 8<br>24 | w<br>w | **Reset/Set Transmit Request**<br>These bits control the set/reset condition for TXRQ (see **Table 16-11**). |
| **RESTXEN0**<br>**SETTXEN0** | 9<br>25 | w<br>w | **Reset/Set Transmit Enable 0**<br>These bits control the set/reset condition for TXEN0 (see **Table 16-11**). |
| **RESTXEN1**<br>**SETTXEN1** | 10<br>26 | w<br>w | **Reset/Set Transmit Enable 1**<br>These bits control the set/reset condition for TXEN1 (see **Table 16-11**). |
| **RESDIR**<br>**SETDIR** | 11<br>27 | w<br>w | **Reset/Set Message Direction**<br>These bits control the set/reset condition for DIR (see **Table 16-11**). |
| **0** | [15:12],<br>[31:28] | w | **Reserved**<br>Should be written with 0. |

**Table 16-11 Reset/Set Conditions for Bits in Register MOCTRn**

| RESy Bit[1)] | SETy Bit | Action on Write |
|---|---|---|
| Write 0 | Write 0 | Leave element unchanged |
| | No write | |
| No write | Write 0 | |
| Write 1 | Write 1 | |

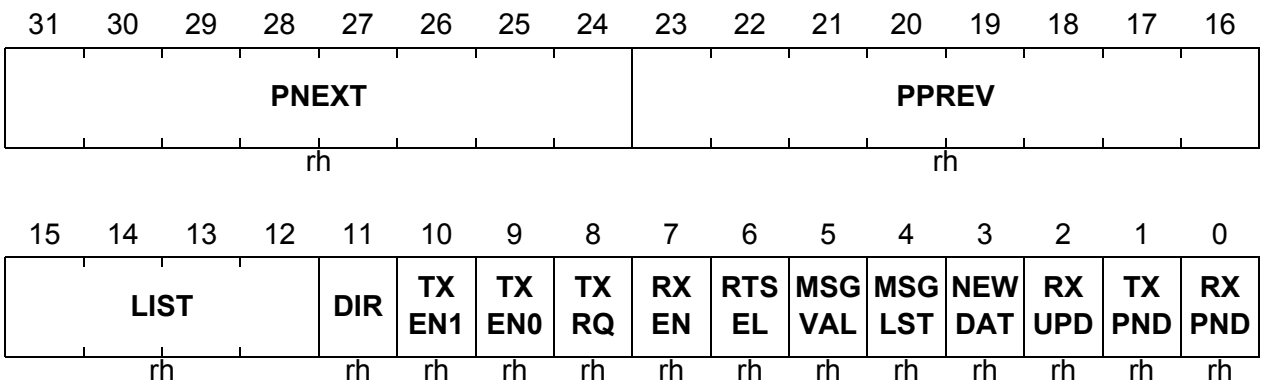
**Table 16-11 Reset/Set Conditions for Bits in Register MOCTRn** (cont'd)

| RESy Bit[1)] | SETy Bit | Action on Write |
|---|---|---|
| Write 1 | Write 0 | Reset element |
| | No write | |
| Write 0 | Write 1 | Set element |
| No write | | |

1) The parameter "y" stands for the second part of the bit name ("RXPND", "TXPND", … up to "DIR").

The MOSTATn is a read-only register that indicates message object list status information such as the number of the current message object predecessor and successor message object, as well as the list number to which the message object is assigned.

**MOSTAT0**
**Message Object 0 Status Register** **Reset Value: 0100 0000$_H$**
**MOSTAT31**
**Message Object 31Status Register** **Reset Value: 1F1E 0000$_H$**
**MOSTATn (n = 1-30)**
**Message Object n Status Register**
**Rest Value: ((n+1)*01000000$_H$)+((n-1)*00010000$_H$)**




| Field | Bits | Type | Description |
|---|---|---|---|
| **RXPND** | 0 | rh | **Receive Pending**<br>0 No CAN message has been received.<br>1 A CAN message has been received by the message object n, either directly or via gateway copy action.<br>RXPND is not reset by hardware but must be reset by software. |
| **TXPND** | 1 | rh | **Transmit Pending**<br>0 No CAN message has been transmitted.<br>1 A CAN message from message object n has been transmitted successfully over the CAN bus.<br>TXPND is not reset by hardware but must be reset by software. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **RXUPD** | 2 | rh | **Receive Updating**<br>0 No receive update ongoing.<br>1 Message identifier, DLC, and data of the message object are currently updated. |
| **NEWDAT** | 3 | rh | **New Data**<br>0 No update of the message object n since last flag reset.<br>1 Message object n has been updated.<br>NEWDAT is set by hardware after a received CAN frame has been stored in message object n.<br>NEWDAT is cleared by hardware when a CAN transmission of message object n has been started.<br>NEWDAT should be set by software after the new transmit data has been stored in message object n to prevent the automatic reset of TXRQ at the end of an ongoing transmission. |
| **MSGLST** | 4 | rh | **Message Lost**<br>0 No CAN message is lost.<br>1 A CAN message is lost because NEWDAT has become set again when it has been already set. |
| **MSGVAL** | 5 | rh | **Message Valid**<br>0 Message object n is not valid.<br>1 Message object n is valid.<br>Only a valid message object takes part in CAN transfers. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **RTSEL** | 6 | rh | **Receive/Transmit Selected**<br>0 Message object n is not selected for receive or transmit operation.<br>1 Message object n is selected for receive or transmit operation.<br>**Frame Reception:**<br>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.<br>**Frame Transmission:**<br>RTSEL is set by hardware when message object n has been identified to be transmitted next. It is checked that 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.<br>RTSEL needs to be checked only when the context of message object n changes and interference with an ongoing frame transfer will be avoided. In all other cases, RTSEL can be ignored. RTSEL has no impact on message acceptance filtering. RTSEL is not cleared by hardware. |
| **RXEN** | 7 | rh | **Receive Enable**<br>0 Message object n is not enabled for frame reception.<br>1 Message object n is enabled for frame reception.<br>RXEN is only evaluated for receive acceptance filtering . |

| Field | Bits | Type | Description |
|---|---|---|---|
| **TXRQ** | 8 | rh | **Transmit Request**<br>0 No transmission of message object n is requested.<br>1 Transmission of message object n on the CAN bus is requested.<br>The transmit request becomes valid only if TXRQ, TXEN0, TXEN1 and MSGVAL are set. TXRQ is set by hardware if a matching remote frame has been received correctly. TXRQ is reset by hardware if message object n has been transmitted successfully and NEWDAT is not set again by software. |
| **TXEN0** | 9 | rh | **Transmit Enable 0**<br>0 Message object n is not enabled for frame transmission.<br>1 Message object n is enabled for frame transmission.<br>Message object n can be transmitted only if both bits, TXEN0 and TXEN1, are set.<br>The user may clear TXEN0 in order to inhibit the transmission of a message that is currently updated, or to disable automatic response of remote frames. |
| **TXEN1** | 10 | rh | **Transmit Enable 1**<br>0 Message object n is not enabled for frame transmission.<br>1 Message object n is enabled for frame transmission.<br>Message object n can be transmitted only if both bits, TXEN0 and TXEN1, are set.<br>TXEN1 is used by the MultiCAN module for selecting the active message object in the transmit FIFOs. |

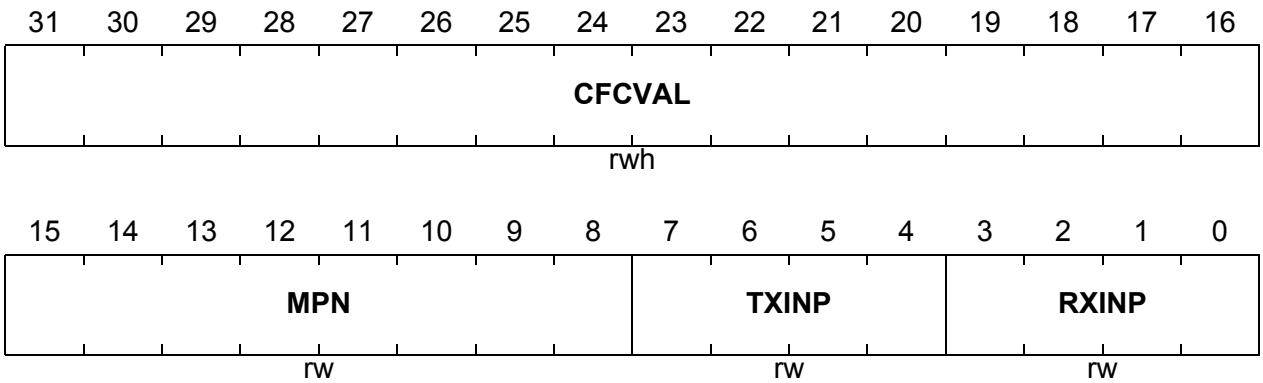
| Field | Bits | Type | Description |
|---|---|---|---|
| **DIR** | 11 | rh | **Message Direction**<br>0 Receive Object selected:<br>With TXRQ = 1, a remote frame with the identifier of message object n is scheduled for transmission. On reception of a data frame with matching identifier, the message is stored in message object n.<br>1 Transmit Object selected:<br>If TXRQ = 1, message object n is scheduled for transmission of a data frame. On reception of a remote frame with matching identifier, bit TXRQ is set. |
| **LIST** | [15:12] | rh | **List Allocation**<br>LIST indicates the number of the message list to which message object n is allocated. LIST is updated by hardware when the list allocation of the object is modified by a panel command. |
| **PPREV** | [23:16] | rh | **Pointer to Previous Message Object**<br>PPREV holds the message object number of the previous message object in a message list structure. |
| **PNEXT** | [31:24] | rh | **Pointer to Next Message Object**<br>PNEXT holds the message object number of the next message object in a message list structure. |

**Table 16-12 MOSTATn Reset Values**

| Message Object | PNEXT | PPREV | Reset Value |
|---|---|---|---|
| 0 | 1 | 0 | $0100\ 0000_H$ |
| 1 | 2 | 0 | $0200\ 0000_H$ |
| 2 | 3 | 1 | $0301\ 0000_H$ |
| 3 | 4 | 2 | $0402\ 0000_H$ |
| … | … | … | … |
| 28 | 29 | 27 | $1D1B\ 0000_H$ |
| 29 | 30 | 28 | $1E1C\ 0000_H$ |
| 30 | 31 | 29 | $1F1D\ 0000_H$ |
| 31 | 31 | 30 | $1F1E\ 0000_H$ |

The Message Object Interrupt Pointer Register MOIPRn holds the message interrupt pointers, the message pending number, and the frame counter value of message object n.

**MOIPRn (n = 0-31)**
**Message Object n Interrupt Pointer Register** **Reset Value: 0000 0000$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **RXINP** | [3:0] | rw | **Receive Interrupt Node Pointer**<br>RXINP selects the interrupt output line CANSRCm (m = 0-7) for a receive interrupt event of message object n. RXINP can also be taken for message pending bit selection (see **Page 16-25**).<br>$0000_B$ Interrupt output line CANSRC0 is selected.<br>$0001_B$ Interrupt output line ICANSRC1 is selected.<br>… …<br>$0110_B$ Interrupt output line CANSRC6 is selected.<br>$0111_B$ Interrupt output line CANSRC7 is selected.<br>$1000_B$-$1111_B$Reserved |
| **TXINP** | [7:4] | rw | **Transmit Interrupt Node Pointer**<br>TXINP selects the interrupt output line CANSRCm (m = 0-7) for a transmit interrupt event of message object n. TXINP can also be taken for message pending bit selection (see **Page 16-25**).<br>$0000_B$ Interrupt output line CANSRC0 is selected.<br>$0001_B$ Interrupt output line CANSRC1 is selected.<br>… …<br>$0110_B$ Interrupt output line CANSRC6 is selected.<br>$0111_B$ Interrupt output line CANSRC7 is selected.<br>$1000_B$-$1111_B$Reserved |

| Field | Bits | Type | Description |
|---|---|---|---|
| **MPN** | [15:8] | rw | **Message Pending Number**<br>This bit field selects the bit position of the bit in the Message Pending Register that is set upon a message object n receive/transmit interrupt. |
| **CFCVAL** | [31:16] | rwh | **CAN Frame Counter Value**<br>When a message is stored in message object n or message object n has been successfully transmitted, the CAN frame counter value NFCRx.CFC is then copied to CFCVAL. |

The Message Object Function Control Register MOFCRn contains bits that select and configure the function of the message object. It also holds the CAN data length code.

**MOFCRn (n = 0-31)**
**Message Object n Function Control Register** **Reset Value: 0000 0000$_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | DLC | | | | STT | SDT | RMM | FRR EN | 0 | OVIE | TXIE | RXIE |
| rw | | | | rwh | | | | rw | rw | rw | rw | rw | rw | rw | rw |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | DATC | DLCC | IDC | GDFS | 0 | | | | MMC | | | |
| rw | | | | rw | rw | rw | rw | rw | | | | rw | | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **MMC** | [3:0] | rw | **Message Mode Control**<br>MMC controls the message mode of message object n.<br>$0000_B$ Standard Message Object<br>$0001_B$ Receive FIFO Base Object<br>$0010_B$ Transmit FIFO Base Object<br>$0011_B$ Transmit FIFO Slave Object<br>$0100_B$ Gateway Source Object<br>Others Reserved |
| **GDFS** | 8 | rw | **Gateway data frame Send**<br>0 TXRQ is unchanged in the destination object.<br>1 TXRQ is set in the gateway destination object after the transfer of a data frame from the gateway source to the gateway destination object.<br>Applicable only to a gateway source object; ignored in other nodes. |

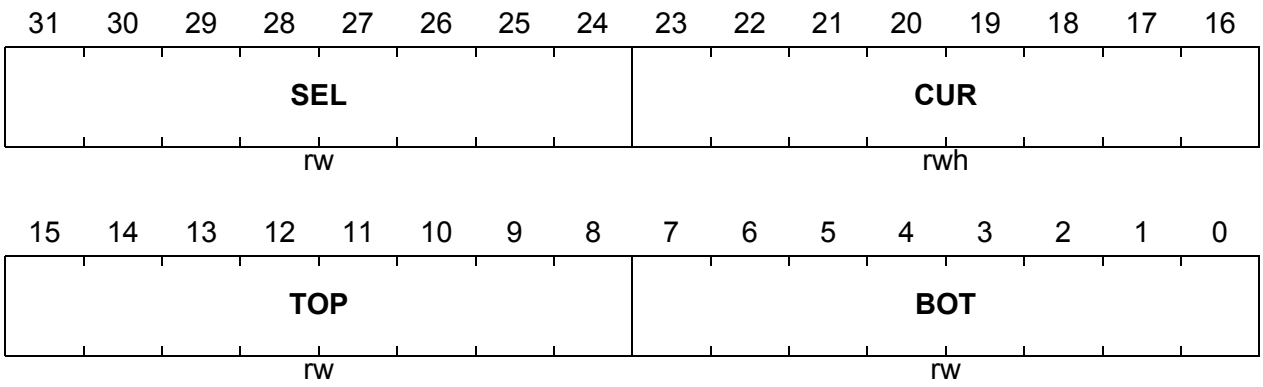
| Field | Bits | Type | Description |
|---|---|---|---|
| **IDC** | 9 | rw | **Identifier Copy**<br>0 The identifier of the gateway source object is not copied.<br>1 The identifier of the gateway source object (after storing the received frame in the source) is copied to the gateway destination object.<br>Applicable only to a gateway source object; ignored in other nodes. |
| **DLCC** | 10 | rw | **Data Length Code Copy**<br>0 Data length code is not copied.<br>1 Data length code of the gateway source object (after storing the received frame in the source) is copied to the gateway destination object.<br>Applicable only to a gateway source object; ignored in other nodes. |
| **DATC** | 11 | rw | **Data Copy**<br>0 Data fields are not copied.<br>1 Data fields in registers MODATALn and MODATAHn of the gateway source object (after storing the received frame in the source) are copied to the gateway destination.<br>Applicable only to a gateway source object; ignored in other nodes. |
| **RXIE** | 16 | rw | **Receive Interrupt Enable**<br>RXIE enables the message receive interrupt of message object n. This interrupt is generated after reception of a CAN message (independent of whether the CAN message is received directly or indirectly via a gateway action).<br>0 Message receive interrupt is disabled.<br>1 Message receive interrupt is enabled.<br>Bit field MOIPRn.RXINP selects the interrupt output line which becomes activated at this type of interrupt. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **TXIE** | 17 | rw | **Transmit Interrupt Enable**<br>TXIE enables the message transmit interrupt of message object n. This interrupt is generated after the transmission of a CAN message.<br>0 Message transmit interrupt is disabled.<br>1 Message transmit interrupt is enabled.<br>Bit field MOIPRn.TXINP selects the interrupt output line which becomes activated at this type of interrupt. |
| **OVIE** | 18 | rw | **Overflow Interrupt Enable**<br>OVIE enables the FIFO full interrupt of message object n. This interrupt is generated when the pointer to the current message object (CUR) reaches the value of SEL in the FIFO/Gateway Pointer Register.<br>0 FIFO full interrupt is disabled.<br>1 FIFO full interrupt is enabled.<br>If message object n is a Receive FIFO base object, bit field MOIPRn.TXINP selects the interrupt output line which becomes activated at this type of interrupt.<br>If message object n is a Transmit FIFO base object, bit field MOIPRn.RXINP selects the interrupt output line which becomes activated at this type of interrupt.<br>For all other message object modes, bit OVIE has no effect. |
| **FRREN** | 20 | rw | **Foreign Remote Request Enable**<br>Specifies whether the TXRQ bit is set in message object n or in a foreign message object referenced by the pointer CUR.<br>0 TXRQ of message object n is set on reception of a matching remote frame.<br>1 TXRQ of the message object referenced by the pointer CUR is set on reception of a matching remote frame. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **RMM** | 21 | rw | **Transmit Object Remote Monitoring**<br>0 Remote monitoring is disabled: Identifier, IDE bit, and DLC of message object n remain unchanged upon the reception of a matching remote frame.<br>1 Remote monitoring is enabled: Identifier, IDE bit, and DLC of a matching remote frame are copied to transmit object n in order to monitor incoming remote frames.<br>Bit RMM applies only to transmit objects and has no effect on receive objects. |
| **SDT** | 22 | rw | **Single Data Transfer**<br>If SDT = 1 and message object n is not a FIFO base object, then MSGVAL is reset when this object has taken part in a successful data transfer (receive or transmit).<br>If SDT = 1 and message object n is a FIFO base object, then MSGVAL is reset when the pointer to the current object CUR reaches the value of SEL in the FIFO/Gateway Pointer Register.<br>With SDT = 0, bit MSGVAL is not affected. |
| **STT** | 23 | rw | **Single Transmit Trial**<br>If this bit is set, then TXRQ is cleared on transmission start of message object n. Thus, no transmission retry is performed in case of transmission failure. |
| **DLC** | [27:24] | rwh | **Data Length Code**<br>Bit field determines the number of data bytes for message object n. Valid values for DLC are 0 to 8. A value of DLC > 8 results in a data length of 8 data bytes, but the DLC code is not truncated upon reception or transmission of CAN frames. |
| **0** | [7:4], [15:12], 19, [31:28] | rw | **Reserved**<br>Read as 0 after reset; value last written is read back; should be written with 0. |

The Message Object FIFO/Gateway Pointer register MOFGPRn contains a set of message object link pointers that are used for FIFO and gateway operations.

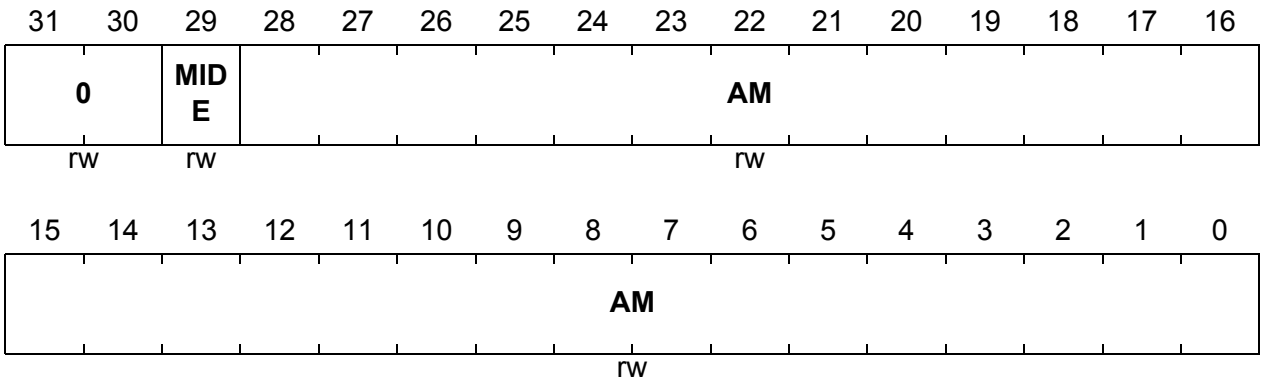
**MOFGPRn (n = 0-31)**
**Message Object n FIFO/Gateway Pointer Register** **Reset Value: 0000 0000$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **BOT** | [7:0] | rw | **Bottom Pointer**<br>Bit field BOT points to the first element in a FIFO structure. |
| **TOP** | [15:8] | rw | **Top Pointer**<br>Bit field TOP points to the last element in a FIFO structure. |
| **CUR** | [23:16] | rwh | **Current Object Pointer**<br>Bit field CUR points to the actual target object within a FIFO/Gateway structure.<br>After a FIFO/gateway operation, CUR is updated with the message number of the next message object in the list structure (given by PNEXT of the message control register) until it reaches the FIFO top element (given by TOP) when it is reset to the bottom element (given by BOT). |
| **SEL** | [31:24] | rw | **Object Select Pointer**<br>Bit field SEL is the second (software) pointer to complement the hardware pointer CUR in the FIFO structure. SEL is used for monitoring purposes (FIFO interrupt generation). |

Message Object n Acceptance Mask Register MOAMRn contains the mask bits for the acceptance filtering of the message object n.

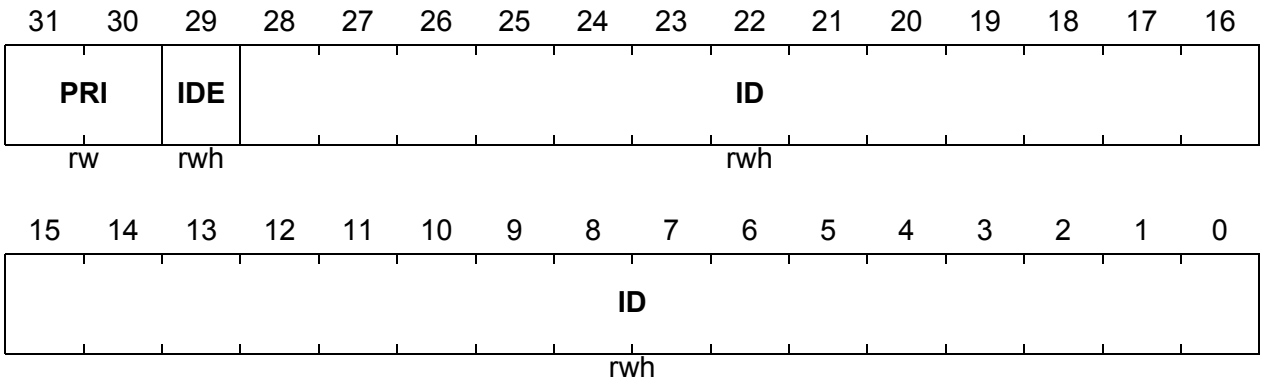
**MOAMRn (n = 0-31)**
**Message Object n Acceptance Mask Register** **Reset Value: 3FFF FFFF$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **AM** | [28:0] | rw | **Acceptance Mask for Message Identifier**<br>Bit field AM is the 29-bit mask for filtering incoming messages with standard identifiers (AM[28:18]) or extended identifiers (AM[28:0]). For standard identifiers, bits AM[17:0] are “don’t care”. |
| **MIDE** | 29 | rw | **Acceptance Mask Bit for Message IDE Bit**<br>0 Message object n accepts the reception of both, standard and extended frames.<br>1 Message object n receives frames only with matching IDE bit. |
| **0** | [31:30] | rw | **Reserved**<br>Read as 0 after reset; value last written is read back; should be written with 0. |

Message Object n Arbitration Register MOARn contains the CAN identifier of the message object.

**MOARn (n = 0-31)**
**Message Object n Arbitration Register** **Reset Value: 0000 0000$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **ID** | [28:0] | rwh | **CAN Identifier of Message Object n**<br>Identifier of a standard message (ID[28:18]) or an extended message (ID[28:0]). For standard identifiers, bits ID[17:0] are “don’t care”. |
| **IDE** | 29 | rwh | **Identifier Extension Bit of Message Object n**<br>0 Message object n handles standard frames with 11-bit identifier.<br>1 Message object n handles extended frames with 29-bit identifier. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **PRI** | [31:30] | rw | **Priority Class**<br>PRI assigns one of the four priority classes 0, 1, 2, 3 to message object n. A lower PRI number defines a higher priority. Message objects with lower PRI value always win acceptance filtering for frame reception and transmission over message objects with higher PRI value. Acceptance filtering based on identifier/mask and list position is performed only between message objects of the same priority class. PRI also determines the acceptance filtering method for transmission:<br>$00_B$ Reserved.<br>$01_B$ Transmit acceptance filtering is based on the list order. This means that message object n is considered for transmission only if there is no other message object with valid transmit request (MSGVAL & TXEN0 & TXEN1 = 1) somewhere before this object in the list.<br>$10_B$ Transmit acceptance filtering is based on the CAN identifier. This means, message object n is considered for transmission only if there is no other message object with higher priority identifier + IDE + DIR (with respect to CAN arbitration rules) somewhere in the list (see **Table 16-13**).<br>$11_B$ Transmit acceptance filtering is based on the list order (as PRI = $01_B$). |

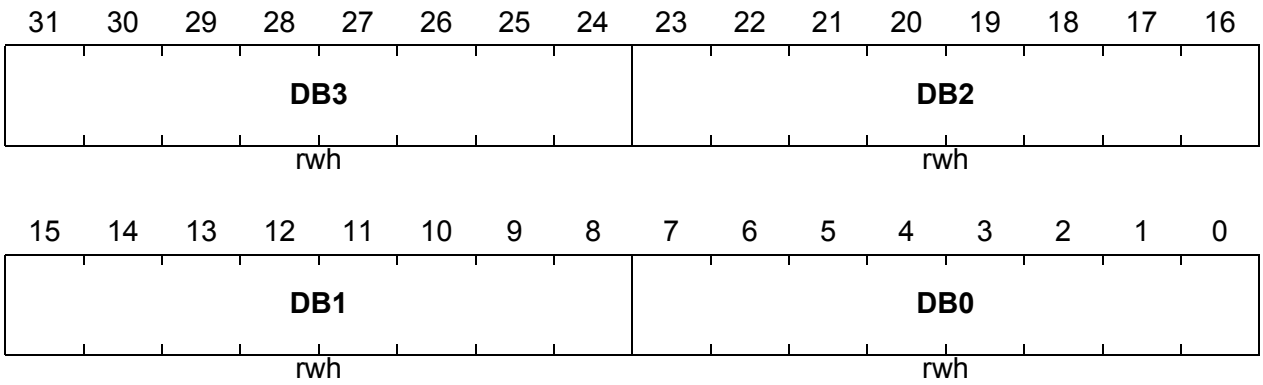
### Transmit Priority of Msg. Objects based on CAN Arbitration Rules

**Table 16-13 Transmit Priority of Msg. Objects Based on CAN Arbitration Rules**

| Settings of Arbitrarily Chosen Message Objects A and B, (A has higher transmit priority than B) | Comment |
|---|---|
| A.MOAR[28:18] < B.MOAR[28:18] (11-bit standard identifier of A less than 11-bit standard identifier of B) | Messages with lower standard identifier have higher priority than messages with higher standard identifier. MOAR[28] is the most significant bit (MSB) of the standard identifier. MOAR[18] is the least significant bit of the standard identifier. |
| A.MOAR[28:18] = B.MOAR[28:18]<br>A.MOAR.IDE = 0 (send Standard Frame)<br>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]<br>A.MOAR.IDE = B.MOAR.IDE = 0<br>A.MOCTR.DIR = 1 (send data frame)<br>B.MOCTR.DIR = 0 (send Remote Fame) | Standard data frames have higher transmit priority than standard remote frames with equal identifier. |
| A.MOAR[28:0] = B.MOAR[28:0]<br>A.MOAR.IDE = B.MOAR.IDE = 1<br>A.MOCTR.DIR = 1 (send data frame)<br>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]<br>A.MOAR.IDE = B.MOAR.IDE = 1<br>(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. |

Message Object n Data Register Low MODATALn contains the lowest four data bytes of message object n. Unused data bytes are set to zero upon reception and ignored for transmission.

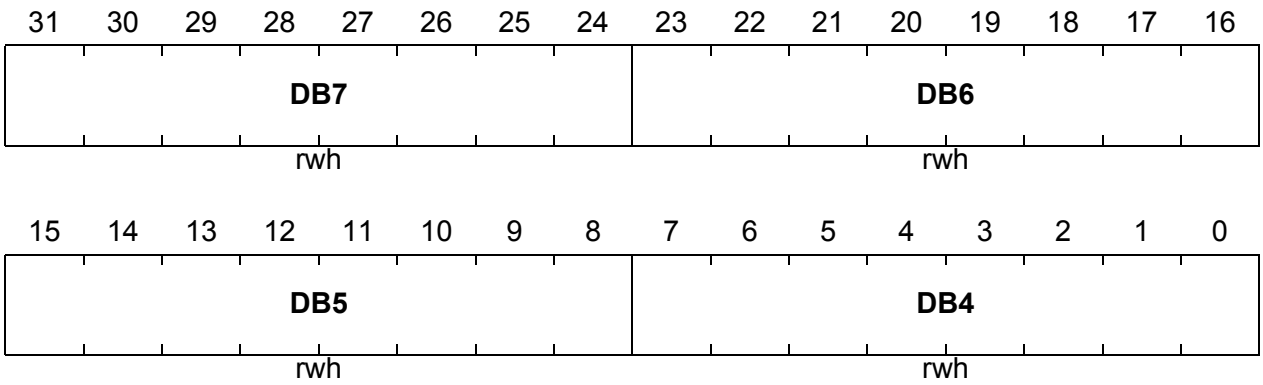
**MODATALn (n = 0-31)**
**Message Object n Data Register Low** **Reset Value: 0000 0000$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **DB0** | [7:0] | rwh | **Data Byte 0 of Message Object n** |
| **DB1** | [15:8] | rwh | **Data Byte 1 of Message Object n** |
| **DB2** | [23:16] | rwh | **Data Byte 2 of Message Object n** |
| **DB3** | [31:24] | rwh | **Data Byte 3 of Message Object n** |

Message Object n Data Register High MODATAH contains the highest four data bytes of message object n. Unused data bytes are set to zero upon reception and ignored for transmission.

**MODATAHn (n = 0-31)**
**Message Object n Data Register High** **Reset Value: 0000 0000$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **DB4** | [7:0] | rwh | **Data Byte 4 of Message Object n** |
| **DB5** | [15:8] | rwh | **Data Byte 5 of Message Object n** |
| **DB6** | [23:16] | rwh | **Data Byte 6 of Message Object n** |
| **DB7** | [31:24] | rwh | **Data Byte 7 of Message Object n** |

### 16.2.4 MultiCAN Access Mediator Register

**CAN_ADCON**
**CAN Address/ Data Control Register** **Reset Value: 0000 0000$_B$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **V3** | **V2** | **V1** | **V0** | **AUAD** | | **BSY** | **RWEN** |
| rw | rw | r | r | rw | | rh | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **RWEN** | 0 | rw | **Read/Write Enable**<br>0 Read is enabled<br>1 Write is enabled. |
| **BSY** | 1 | rh | **Data Transmission Busy**<br>0 Data Transimission is finished.<br>1 Data Transimission is in progress. |
| **AUAD** | [3:2] | rw | **Auto Increment/Decrement the Address**<br>00 No increment/decrement the address.<br>01 Auto increment the current address (+1 )<br>10 Auto decrement the current address (-1 )<br>11 Auto increment the current address (+8) |
| **V0** | 4 | rw | **CAN Data 0 Valid**<br>0 Data in CAN_DATA0 register is not valid for transmission.<br>1 Data in CAN_DATA0 register is valid for transmission. |
| **V1** | 5 | rw | **CAN Data 1 Valid**<br>0 Data in CAN_DATA1 register is not valid for transmission.<br>1 Data in CAN_DATA1 register is valid for transmission. |
| **V2** | 6 | rw | **CAN Data 2 Valid**<br>0 Data in CAN_DATA2 register is not valid for transmission.<br>1 Data in CAN_DATA2 register is valid for transmission. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **V3** | 7 | rw | **CAN Data 3 Valid**<br>0 Data in CAN_DATA3 register is not valid for transmission.<br>1 Data in CAN_DATA3 register is valid for transmission. |

**CAN_ADL**
**Can Address Register Low** **Reset Value: 0000 0000$_B$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **CA9** | **CA8** | **CA7** | **CA6** | **CA5** | **CA4** | **CA3** | **CA2** |
| rwh | | | | | | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **CAn (n=2 to 9)** | n-2 | rwh | **CAN Address Bit n** |

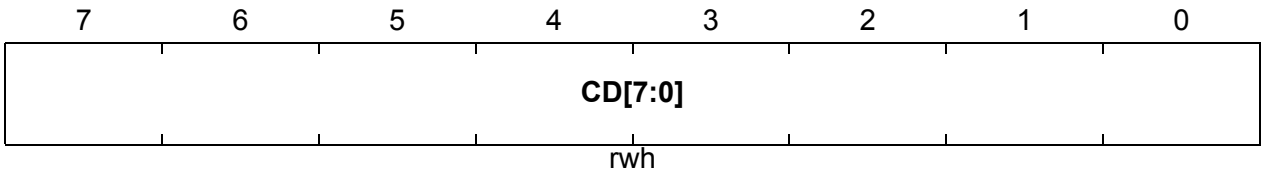
**CAN_ADH**
**CAN Address Register High** **Reset Value: 0000 0000$_B$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | | | | **CA13** | **CA12** | **CA11** | **CA10** |
| r | | | | rwh | rwh | rwh | rwh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **CA10** | 0 | rwh | **CAN Address Bit 10** |
| **CA11** | 1 | rwh | **CAN Address Bit 11** |
| **CA12** | 2 | rwh | **CAN Address Bit 12** |
| **CA13** | 3 | rwh | **CAN Address Bit 13** |
| **0** | [7:4] | r | **Reserved;** read as 0; should be written with 0. |

## CAN_DATA0
## CAN Data Register 0

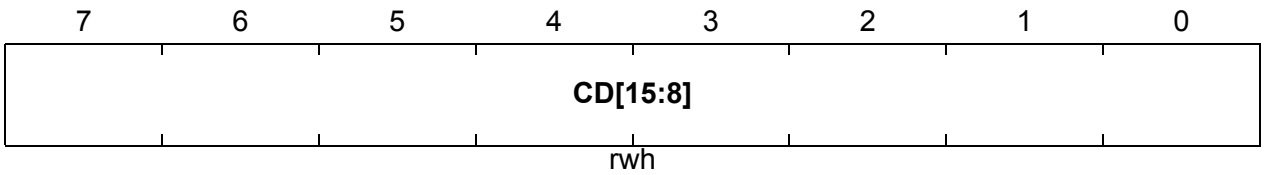
**Reset Value: 0000 0000$_B$**




| Field | Bits | Type | Description |
|---|---|---|---|
| CD | [7:0] | rwh | CAN Data Byte 0 |

## CAN_DATA1
## CAN Data Register 1

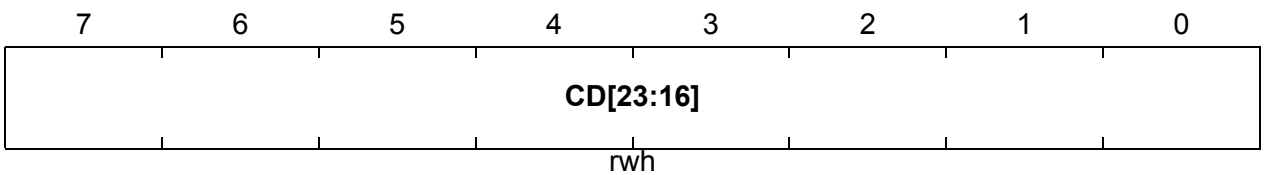
**Reset Value: 0000 0000$_B$**




| Field | Bits | Type | Description |
|---|---|---|---|
| CD | [7:0] | rwh | CAN Data Byte 1 |

## CAN_DATA2
## CAN Data Register 2

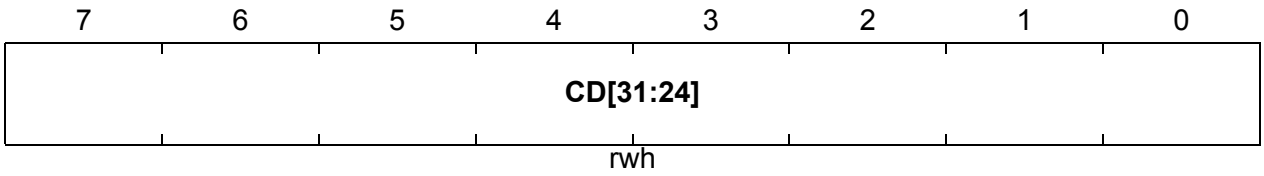
**Reset Value: 0000 0000$_B$**




| Field | Bits | Type | Description |
|---|---|---|---|
| CD | [7:0] | rwh | CAN Data Byte 2 |

## CAN_DATA3
## CAN Data Register 3

**Reset Value: 0000 0000$_B$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **CD** | [7:0] | rwh | **CAN Data Byte 3** |

# 17 Analog-to-Digital Converter

The XC878 includes a high-performance 10-bit Analog-to-Digital Converter (ADC) with eight multiplexed analog input channels. The ADC uses a successive approximation technique to convert the analog voltage levels from up to eight different sources.

**Features**

- Successive approximation
- 8-bit or 10-bit resolution
- Eight analog channels
- Four independent result registers
- Result data protection for slow CPU access (wait-for-read mode)
- Single conversion mode
- Autoscan functionality
- Limit checking for conversion results
- Data reduction filter (accumulation of up to 2 conversion results)
- Two independent conversion request sources with programmable priority
- Selectable conversion request trigger
- Flexible interrupt generation with configurable service nodes
- Programmable sample time
- Programmable clock divider
- Cancel/restart feature for running conversions
- Integrated sample and hold circuitry
- Compensation of offset errors
- Low power modes

## 17.1 Structure Overview

The ADC module consists of two main parts, i.e., analog and digital, with each containing independent building blocks.

The analog part includes:

- Analog input multiplexer
  (for selecting the channel to be converted)
- Analog converter stage
  (e.g., capacitor network and comparator as part of the ADC)
- Digital control part of the analog converter stage
  (for controlling the analog-to-digital conversion process and generating the conversion result)

The digital part defines and controls the overall functionality of the ADC module, and includes:

- Digital data and conversion request handling
  (for controlling the conversion trigger mechanisms and handling the conversion results)
- Bus interface to the device-internal data bus
  (for controlling the interrupts and register accesses)

The block diagram of the ADC module is shown in **Figure 17-1**. The analog input channel x (x = 0 - 7) is available at port pin ANx.

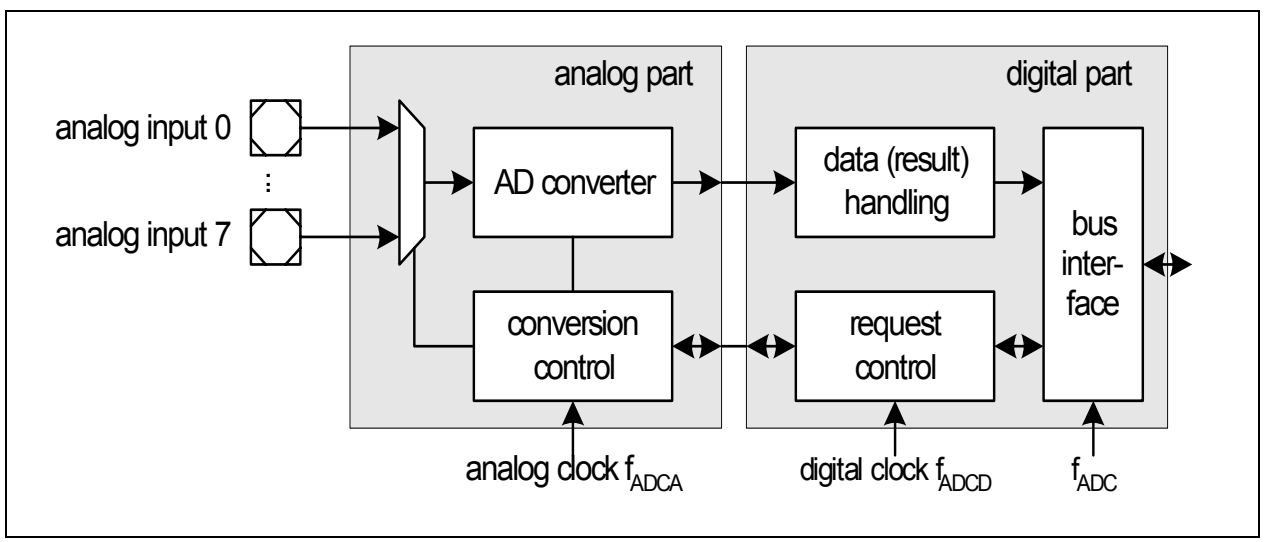



**Figure 17-1 Overview of ADC Building Blocks**

## 17.2 Clocking Scheme

A common module clock $f_{ADC}$ generates the various clock signals used by the analog and digital parts of the ADC module:

- $f_{ADCA}$ is input clock for the analog part.
- $f_{ADCI}$ is internal clock for the analog part (defines the time base for conversion length and the sample time). This clock is generated internally in the analog part, based on the input clock $f_{ADCA}$ to generate a correct duty cycle for the analog components.
- $f_{ADCD}$ is input clock for the digital part. This clock is used for the arbiter (defines the duration of an arbitration round) and other digital control structures (e.g., registers and the interrupt generation).

The internal clock for the analog part $f_{ADCI}$ is limited to a maximum frequency of 10 MHz. Therefore, the ADC clock prescaler must be programmed to a value that ensures $f_{ADCI}$ does not exceed 10 MHz. The prescaler ratio is selected by bit field CTC in register GLOBCTR. A prescaling ratio of 32 can be selected when the maximum performance of the ADC is not required.

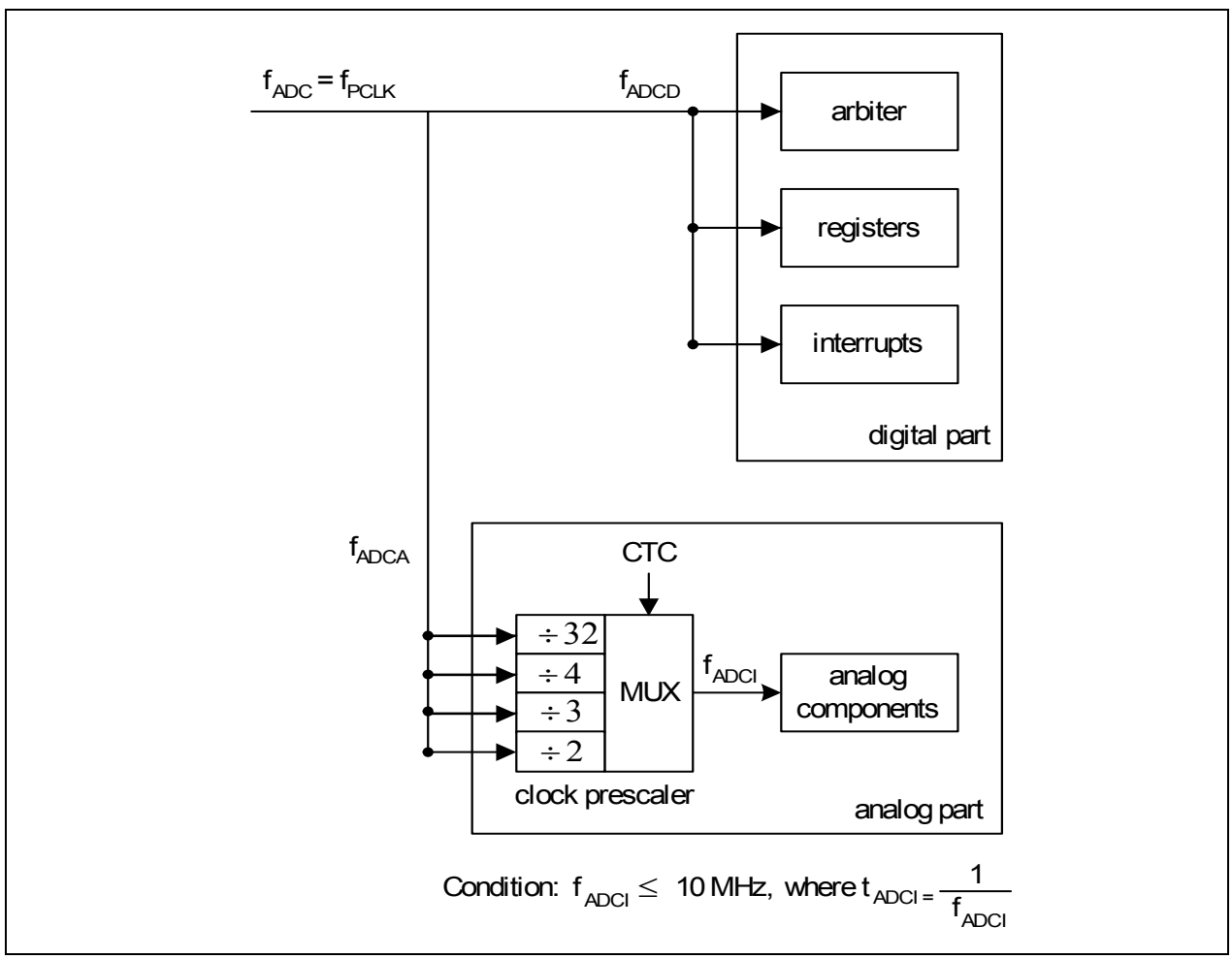



**Figure 17-2 Clocking Scheme**

For module clock $f_{ADC}$ = 24 MHz, the analog clock $f_{ADCI}$ frequency can be selected as shown in **Table 17-1**.

**Table 17-1 $f_{ADCI}$ Frequency Selection**

| Module Clock $f_{ADC}$ | CTC | Prescaling Ratio | Analog Clock $f_{ADCI}$ |
|---|---|---|---|
| 24 MHz | $00_B$ | ÷ 2 | 12 MHz (N.A) |
| | $01_B$ | ÷ 3 | 8 MHz |
| | $10_B$ | ÷ 4 | 6 MHz |
| | $11_B$ (default) | ÷ 32 | 750 kHz |

As $f_{ADCI}$ cannot exceed 10 MHz, bit field CTC should not be set to $00_B$ when $f_{ADC}$ is 24 MHz. During slow-down mode where $f_{ADC}$ may be reduced to 12 MHz, 6 MHz etc., CTC can be set to $00_B$ as long as the divided analog clock $f_{ADCI}$ does not exceed 10 MHz. However, it is important to note that the conversion error could increase due to loss of charges on the capacitors, if $f_{ADC}$ becomes too low during slow-down mode.

### 17.2.1 Conversion Timing

The analog-to-digital conversion procedure consists of the following phases:

- Synchronization phase ($t_{SYN}$)
- Sample phase ($t_S$)
- Conversion phase
- Write result phase ($t_{WR}$)

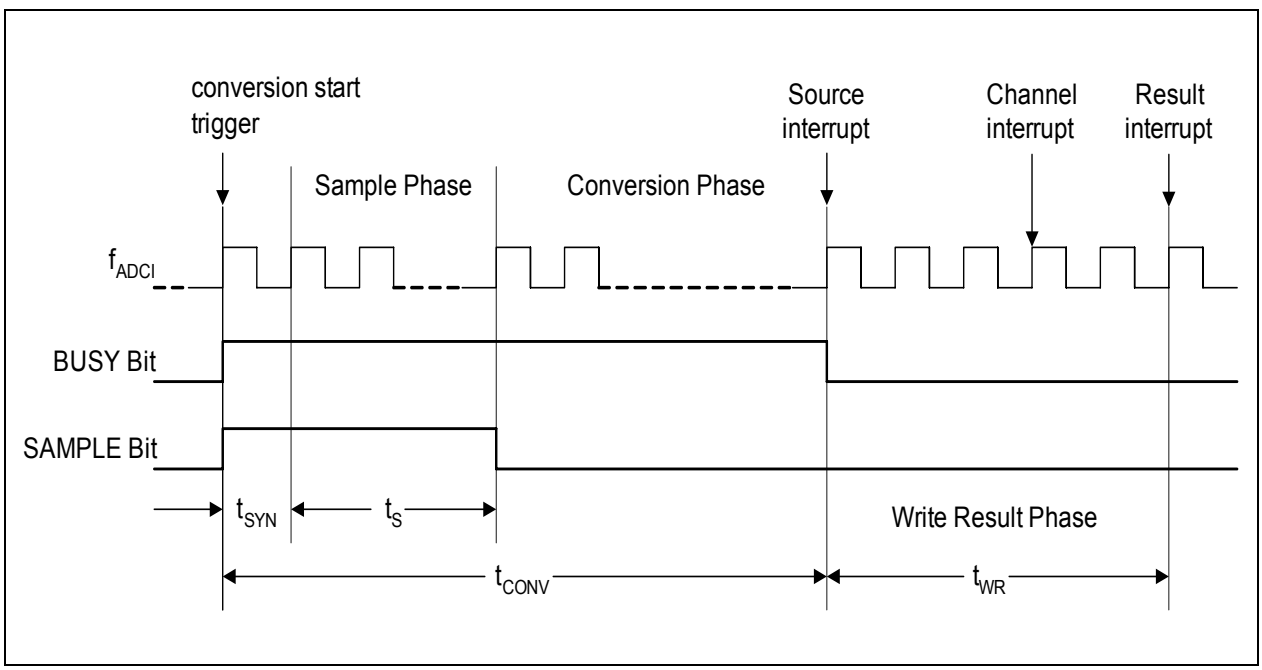



**Figure 17-3 Conversion Timing**

### Synchronization Phase $t_{SYN}$

One $f_{ADCI}$ period is required for synchronization between the conversion start trigger (from the digital part) and the beginning of the sample phase (in the analog part). The BUSY and SAMPLE bits will be set with the conversion start trigger.

### Sample Phase $t_S$

During this period, the analog input voltage is sampled. The internal capacitor array is connected to the selected analog input channel and is loaded with the analog voltage to be converted. The analog voltage is internally fed to a voltage comparator. With the beginning of the sampling phase, the SAMPLE and BUSY flags in register GLOBSTR are set. The duration of this phase is common to all analog input channels and is controlled by bit field STC in register INPCR0:

$$t_S = (2 + STC) \times t_{ADCI} \quad (17.1)$$

### Conversion Phase

During the conversion phase, the analog voltage is converted into an 8-bit or 10-bit digital value using the successive approximation technique with a binary weighted capacitor network. At the beginning of the conversion phase, the SAMPLE flag is reset (to indicate the sample phase is over), while the BUSY flag continues to be asserted. The BUSY flag is deasserted only at the end of the conversion phase with the corresponding source interrupt (of the source that started the conversion) asserted.

### Write Result Phase $t_{WR}$

At the end of the conversion phase, the corresponding channel interrupt (of the converted channel) is asserted three $f_{ADCI}$ periods later, after the limit checking has been performed. The result interrupt is asserted, once the conversion result has been written into the target result register.

### Total Conversion Time $t_{CONV}$

The total conversion time (synchronizing + sampling + charge redistribution) $t_{CONV}$ is given by:

$$t_{CONV} = t_{ADC} \times (1 + r \times (3 + n + STC)) \quad (17.2)$$

where

r = CTC + 2 for CTC = $00_B$, $01_B$ or $10_B$,

r = 32 for CTC = $11_B$,

CTC = Conversion Time Control,

STC = Sample Time Control,

n = 8 or 10 (for 8-bit and 10-bit conversion, respectively),

$t_{ADC} = 1 / f_{ADC}$

Example:

STC = $00_H$,

CTC = $01_B$,

$f_{ADC}$ = 26.7 MHz,

n = 10,

$t_{CONV} = t_{ADC} \times (1 + 3 \times (3 + 10 + 0)) = 1.5\ \mu s$

## 17.3 Low Power Mode

The ADC module may be disabled, either partially or completely, when no conversion is required in order to reduce power consumption.

The analog part of the ADC module may be disabled by resetting the ANON bit. This causes the generation of $f_{ADCI}$ to be stopped and results in a reduction in power consumption. Conversions are possible only by enabling the analog part (ANON = 1) again. The wake-up time is approximately 100 ns.
Refer to **Section 17.7.1** for register description of disabling the ADC analog part.

If the ADC functionality is not required at all, it can be completely disabled by gating off its clock input ($f_{ADC}$) for maximal power reduction. This is done by setting bit ADC_DIS in register PMCON1. Refer to **Chapter 8.1.4** for details on peripheral clock management.

**PMCON1**
**Power Mode Control Register 1 ($B5_H$) Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **0** | **CDC_DIS** | **CAN_DIS** | **MDU_DIS** | **T2CCU _DIS** | **CCU_DIS** | **SSC_DIS** | **ADC_DIS** |
| r | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **ADC_DIS** | 0 | rw | **ADC Disable Request. Active high.**<br>$0_B$ ADC is in normal operation (default)<br>$1_B$ Request to disable the ADC |
| **0** | 7 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

## 17.4 Functional Description

The ADC module functionality includes:

- Two different conversion request sources (sequential and parallel) with independent registers. The request sources are used to trigger conversions due to external events (synchronization to PWM signals), sequencing schemes, etc.
- An arbiter that regularly scans the request sources to find the channel with the highest priority for the next conversion. The priority of each source can be programmed individually to obtain the required flexibility to cover the desired range of applications.
- Control registers for each of the eight channels that define the behavior of each analog input (such as the interrupt behavior, a pointer to a result register, a pointer to a channel class, etc.).
- An input class register that delivers general channel control information (sample time) from a centralized location.
- Four result registers (instead of one result register per analog input channel) for storing the conversion results and controlling the data reduction.
- A decimation stage for conversion results, adding the incoming result to the value already stored in the targeted result register. This stage allows fast consecutive conversions without the risk of data loss for slow CPU clock frequency.

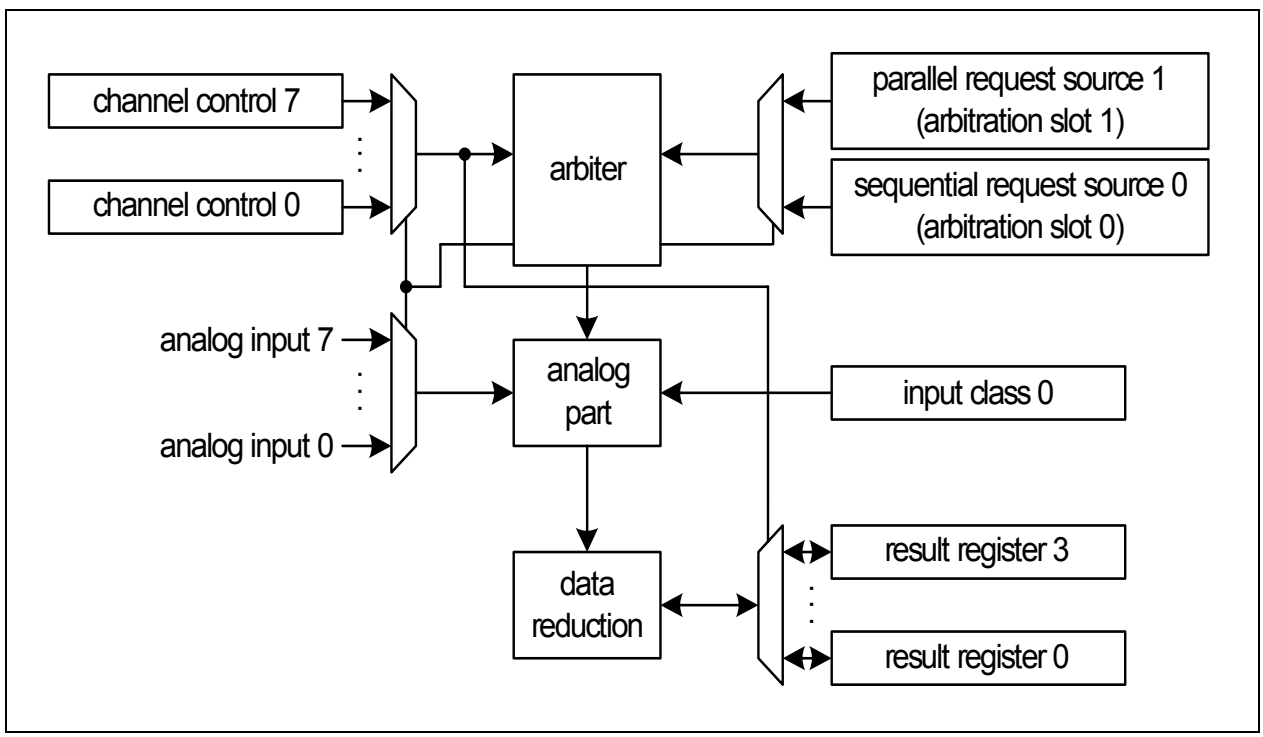



**Figure 17-4 ADC Block Diagram**

### 17.4.1 Request Source Arbiter

The arbiter can operate in two modes that are selectable by bit ARBM:

- Permanent arbitration:
  In this mode, the arbiter will continuously poll the request sources even when there is no pending conversion request.
- Arbitration started by pending conversion request:
  In this mode, the arbiter will start polling the request sources only if there is at least one conversion pending request.

Once started, the arbiter polls the two request sources (source x at slot x, x = 0 - 1) to find the analog channel with the highest priority that must be converted. For each arbitration slot, the arbiter polls the request pending signal (REQPND) and the channel number valid signal (REQCHNRV) of one request source. The sum of all arbitration slots is called an arbitration round. An arbitration slot must be enabled (ASENx = 1) before it can take part in the arbitration.

Each request source has a source priority that can be programmed via bit PRIOx. Starting with request source 0 (arbitration slot 0), the arbiter checks if a request source has a pending request (REQPND = 1) for a conversion. If more than one request source is found with the same programmed priority level and a pending conversion request, the channel specified by the request source that was found first is selected. The REQCHNRV signal is also checked by the arbiter and a conversion can only be started if REQCHNRV = 1 (and REQPND = 1). If both request sources are programmed with the same priority, the channel number specified by request source 0 will be converted first since it is connected to arbitration slot 0.

The period $t_{ARB}$ of a complete arbitration round is fixed at:

$$t_{ARB} = 4 * t_{ADCD} \quad (17.3)$$

Refer to **Section 17.7.2** for register description of priority and arbitration control.

### 17.4.2 Conversion Start Modes

At the end of each arbitration round, the arbiter would have found the request source with the highest priority and a pending conversion request. It stores the arbitration result, namely the channel number, the sample time and the targeted result register for further actions.

If the analog part is idle, a conversion can be started immediately. If a conversion is currently running, the arbitration result is compared to the priority of the currently running conversion. If the current conversion has the same or a higher priority, it will continue to completion. Immediately after its completion, the next conversion can begin. As soon as the analog part is idle and the arbiter has output a conversion request, the conversion will start.

In case the new conversion request has a higher priority than the current conversion, two conversion start modes exist (selectable by bit CSMx, x = 0 - 1):

- Wait-for-Start:
  In this mode, the current conversion is completed normally. The pending conversion request will be treated immediately after the conversion is completed. The conversion start takes place as soon as possible.
- Cancel-Inject-Repeat:
  In this mode, the current conversion is aborted immediately if a new request with a higher priority has been found. The new conversion is started as soon as possible after the abort action. The aborted conversion request is restored in the request source that has requested the aborted conversion. As a result, it takes part in the next arbitration round. The priority of an active request source (including pending or active conversion) must not be changed by software. The abort will not be accepted during the last 3 clock cycles of a running conversion.

Refer to **Section 17.7.2** for register description relating to conversion start control.

### 17.4.3 Channel Control

Each channel has its own control information that defines the target result register for the conversion result (see **Section 17.7.4**). The only control information that is common to all channels is the sampling time defined by the input class register (see **Section 17.7.5**).

### 17.4.4 Sequential Request Source

A sequential request source requests one conversion after the other. The amount of channels requested for conversion depends on the length of the sequential buffer queue (number of queue stages).
The sequential source register description can be found in **Section 17.7.6**.

#### 17.4.4.1 Overview

The sequential request source at arbitration slot 0 requests one conversion after another for channel numbers between 0 and 7. The queue stage stores the requested channel number and some additional control information. As a result, the order in which the channels are to be converted is freely programmable without restrictions in the sequence. The additional control information is used to enable the request source interrupt (when the requested channel conversion is completed) and to enable the automatic refill process.

A sequential source consists of 4 queue stages, one backup stage (QBUR0) and a mode control register (QMR0). The backup stage stores the information about the latest conversion requested after it has been aborted. If the backup register contains an aborted request (V = 1), it is treated before the entries in the queue stage. This implies that only the bit V in the backup register is cleared when the requested conversion is started. If the bit V in the backup register is not set, the bit V in the queue stage 0 is reset when the requested conversion is started. The request source can take part in the source arbitration if the backup stage or queue stage contains a valid request (V = 1).

*Note: Of the 4 queue stages, only the register queue 0 can be read, the register of the other stages are internal.*

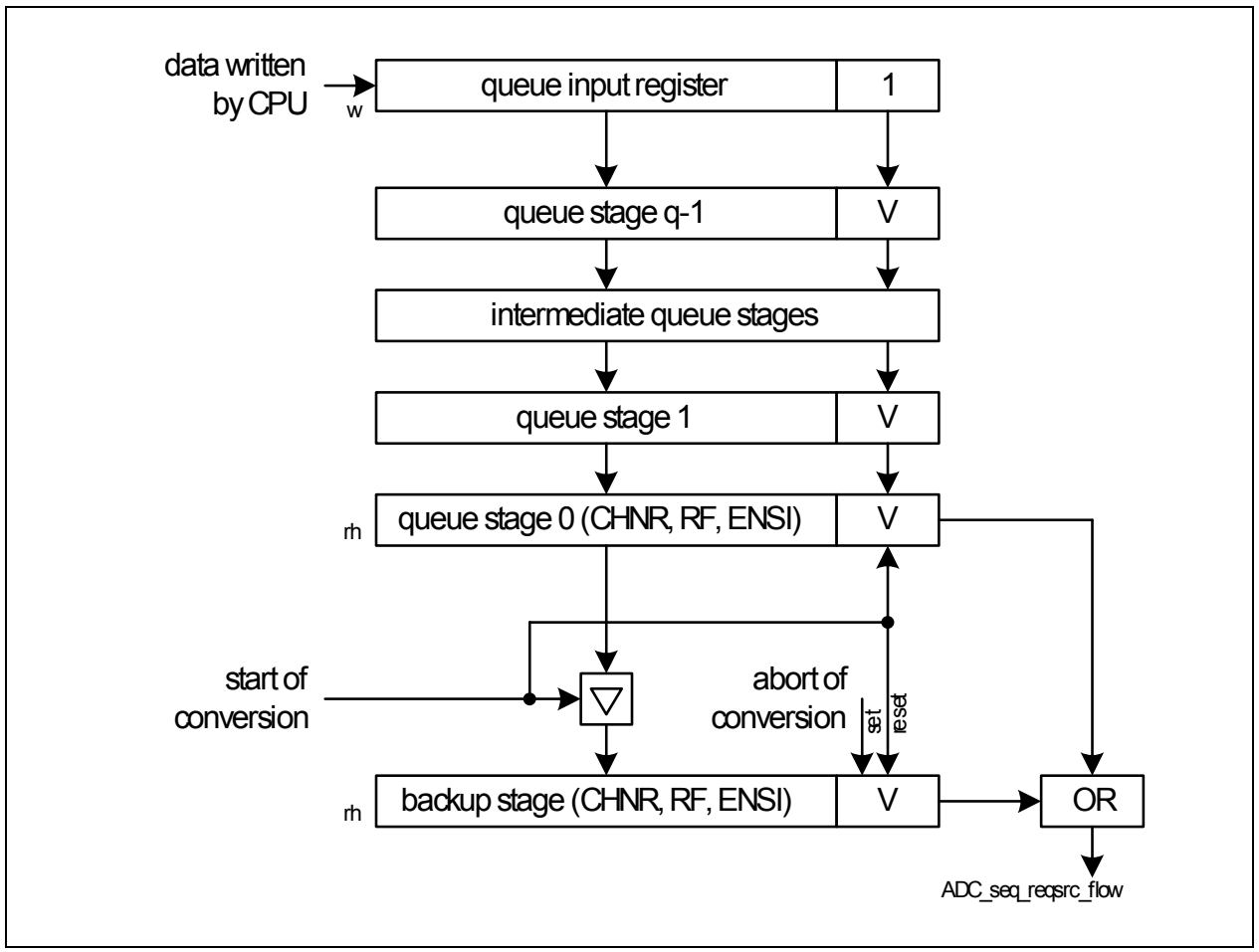



**Figure 17-5 Multi-Stage Queue**

The automatic refill feature can be activated (RF = 1) to allow automatic re-insertion of the pending request into the queue stage after a successful execution (conversion start). Otherwise, the pending request will be discarded once it is executed. While the automatic refill feature is enabled, software should not write data to the queue input register.

The write address in which to enter a conversion request is given by the write-only queue input register (QINR0). If there is still an empty stage (V=0) in the queue, the written value will be stored there (bit V becomes set), or else the write action is ignored. In the event that a requested conversion is aborted after its start, its setting is stored in the backup register (bit V becomes set).

Refer to **Section 17.7.6** for description of the sequential request source registers.

### 17.4.4.2 Request Source Control

If the conversion requested by the source is not related to an external trigger event (EXTR = 0), the valid bit V = 1 directly requests the conversion by setting signals REQPND and REQCHNRV to 1. In this case, no conversion will be requested if V = 0. A gating mechanism allows the user to enable/disable conversion requests according to bit ENGT.

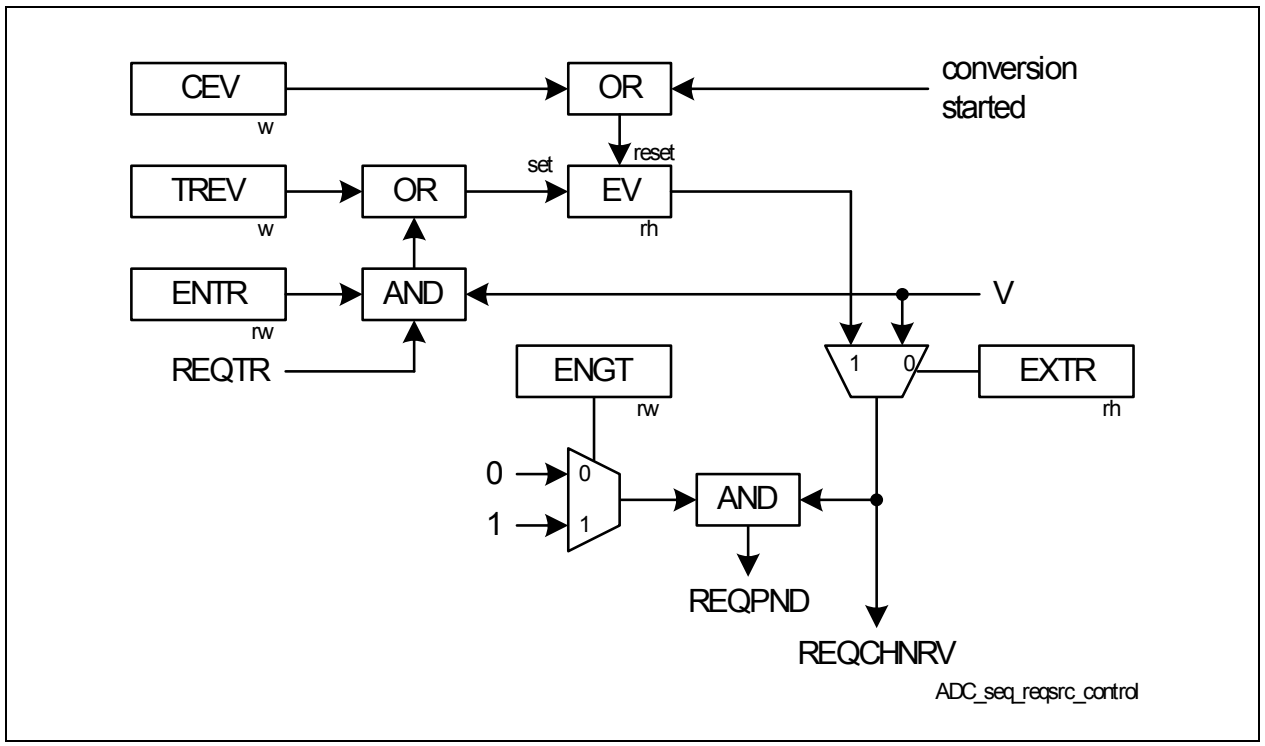



**Figure 17-6 Sequential Request Source Control**

If the requested conversion is sensitive to an external trigger event (EXTR = 1), the signal REQTR can be taken into account (with ENTR = 1) or the software can write TREV = 1. Both actions set the event flag EV. The event flag EV = 1 indicates that an external event has taken place and a conversion can be requested (EV can be set only if a conversion request is valid with V = 1). In this case, the signal REQCHNRV is derived from bit EV.

In the queue backup register, bit EXTR is always considered as 0. If a queue controlled conversion has been started and aborted due to a higher priority conversion, the aborted conversion will be restarted without waiting for a new trigger event.

### 17.4.5 Parallel Request Source

A parallel request source generates one or more channel conversion requests in parallel. The requests are always treated one after the other in a pre-defined sequence (higher channel numbers before lower channel numbers).
The parallel source register description can be found in **Section 17.7.7**.

#### 17.4.5.1 Overview

The parallel request source at arbitration slot 1 generates one or more conversion requests for channel numbers between 4 and 7 in parallel. The requests are always treated one after the other (in separate arbitration rounds) in a predefined sequence (higher channel numbers before lower channel numbers).

The parallel request source consists of a conversion request control register (CRCR1), a conversion request pending register (CRPR1) and a conversion request mode register (CRMR1). The contents of the conversion request control register are copied (overwrite) to the conversion request pending register when a selected load event (LDE) occurs. The type of the event defines the behavior and the trigger of the request source.

The activation of a conversion request to the arbiter may be started if the content of the conversion pending register is not 0. The highest bit position number among the pending bits with values equal to 1 specifies the channel number for conversion. To take part in the source arbitration, both the REQCHNRV and REQPND signals must be 1.

Refer to **Section 17.7.7** for description of the parallel request source registers.

### 17.4.5.2 Request Source Control

All conversion pending bits are ORed together to deliver an intermediate signal PND for generating REQCHNRV and REQPND. The signal PND is gated with bit ENGT, allowing the user to enable/disable conversion requests. See **Figure 17-7**.

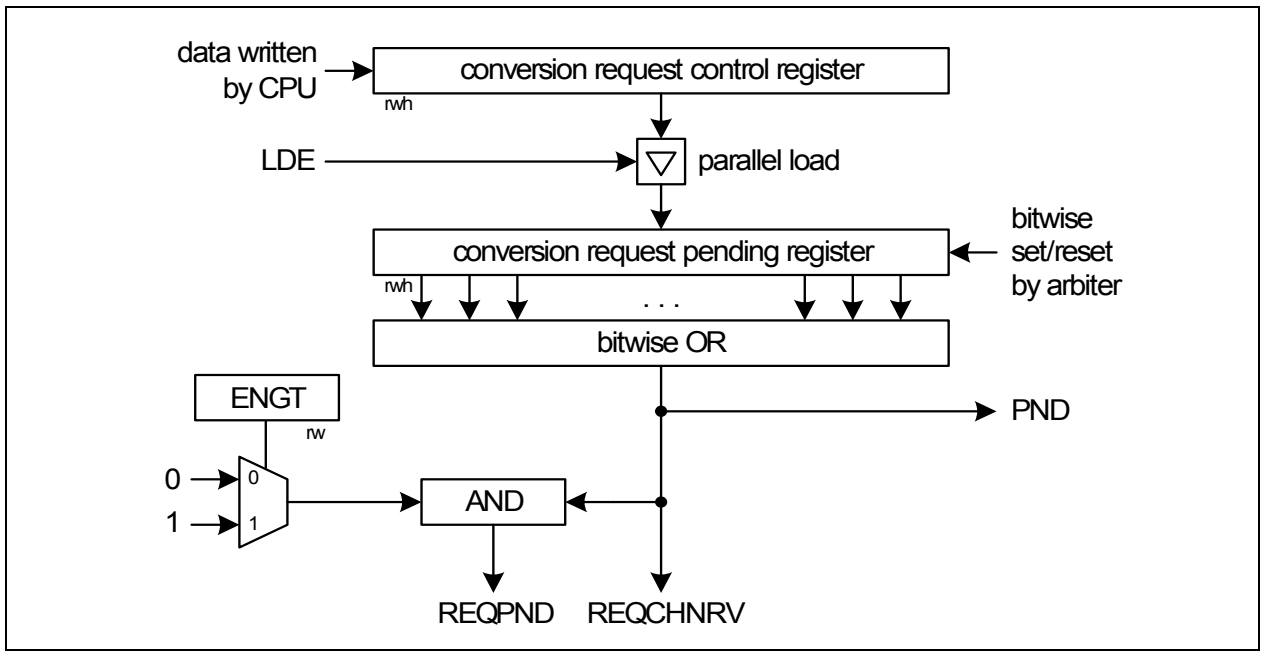



**Figure 17-7 Parallel Request Source Control**

The load event for a parallel load can be:

- External trigger at the input line REQTR. See **Section 17.4.5.3**.
- Write operation to a specific address of the conversion request control register. See **Section 17.4.5.4**.
- Write operation with LDEV = 1 to the request source mode register. See **Section 17.4.5.4**.
- Source internal action (conversion completed and PND = 0 for autoscan mode). See **Section 17.4.5.5**.

Each bit (bit x, x = 4 - 7) in the conversion request control/pending registers corresponds to one analog input channel. The bit position directly defines the channel number. The bits in the conversion request pending register can be set or reset bitwisely by the arbiter:

- The corresponding bit in the conversion request pending register is automatically reset when the arbiter indicates the start of conversion for this channel.
- The bit is automatically set when the arbiter indicates that the conversion has been aborted.

A source interrupt can be generated (if enabled) when a conversion (requested by this source) is completed while PND = 0. These rules apply only if the request source has triggered the conversion.

#### 17.4.5.3 External Trigger

The conversion request for the parallel source (and also the sequential source) can be synchronized to an external trigger event. For the parallel source, this is done by coupling the reload event to a request trigger input, REQTR.

#### 17.4.5.4 Software Control

The load event for the parallel source can also be generated under software control in two ways:

- The conversion request control register can be written at two different addresses (CRCR1 and CRPR1). Accessed at CRCR1, the write action changes only the bits in this register. Accessed at CRPR1, a load event will take place one clock cycle after the write access. This automatic load event can be used to start conversions with a single move operation. In this case, the information about the channels to be converted is given as an argument in the move instruction.
- Bit LDEV can be written with 1 by software to trigger the load event. In this case, the load event does not contain any information about the channels to be converted, but always takes the contents of the conversion request control register. This allows the conversion request control register to be written at a second address without triggering the load event.

#### 17.4.5.5 Autoscan

The autoscan is a functionality of the parallel source. If autoscan mode is enabled, the load event takes place when the conversion is completed while PND = 0, provided the parallel request source has triggered the conversion. This automatic reload feature allows channels 4 to 7 to be constantly scanned for pending conversion requests without the need for external trigger or software action.

### 17.4.6 Wait-for-Read Mode

The wait-for-read mode can be used for all request sources to allow the CPU to treat each conversion result independently without the risk of data loss. Data loss can occur if the CPU does not read a conversion result in a result register before a new result overwrites the previous one.

In wait-for-read mode, the conversion request generated by a request source for a specific channel will be disabled (and conversion not possible) if the targeted result register contains valid data (indicated by its valid flag being set). Conversion of the requested channel will not start unless the valid flag of the targeted result register is cleared (data is invalid). The wait-for-read mode for a result register can be enabled by setting bit WFR (see **Section 17.7.8**).

### 17.4.7 Result Generation

The result generation part handles the storage of the conversion result, data decimation, limit checking and interrupt generation.

#### 17.4.7.1 Overview

The result generation of the ADC module consists of several parts:

- A limit checking unit, comparing the conversion result to two selected boundary values (BOUND0 and BOUND1). A channel interrupt can be generated according to the limit check result.
- A data reduction filter, accumulating the conversion results. The accumulation is done by adding the new conversion result to the value stored in the selected result register.
- Four result registers, storing the conversion results. The software can read the conversion result from the result registers. The result register used to store the conversion result is selected individually for each input channel.

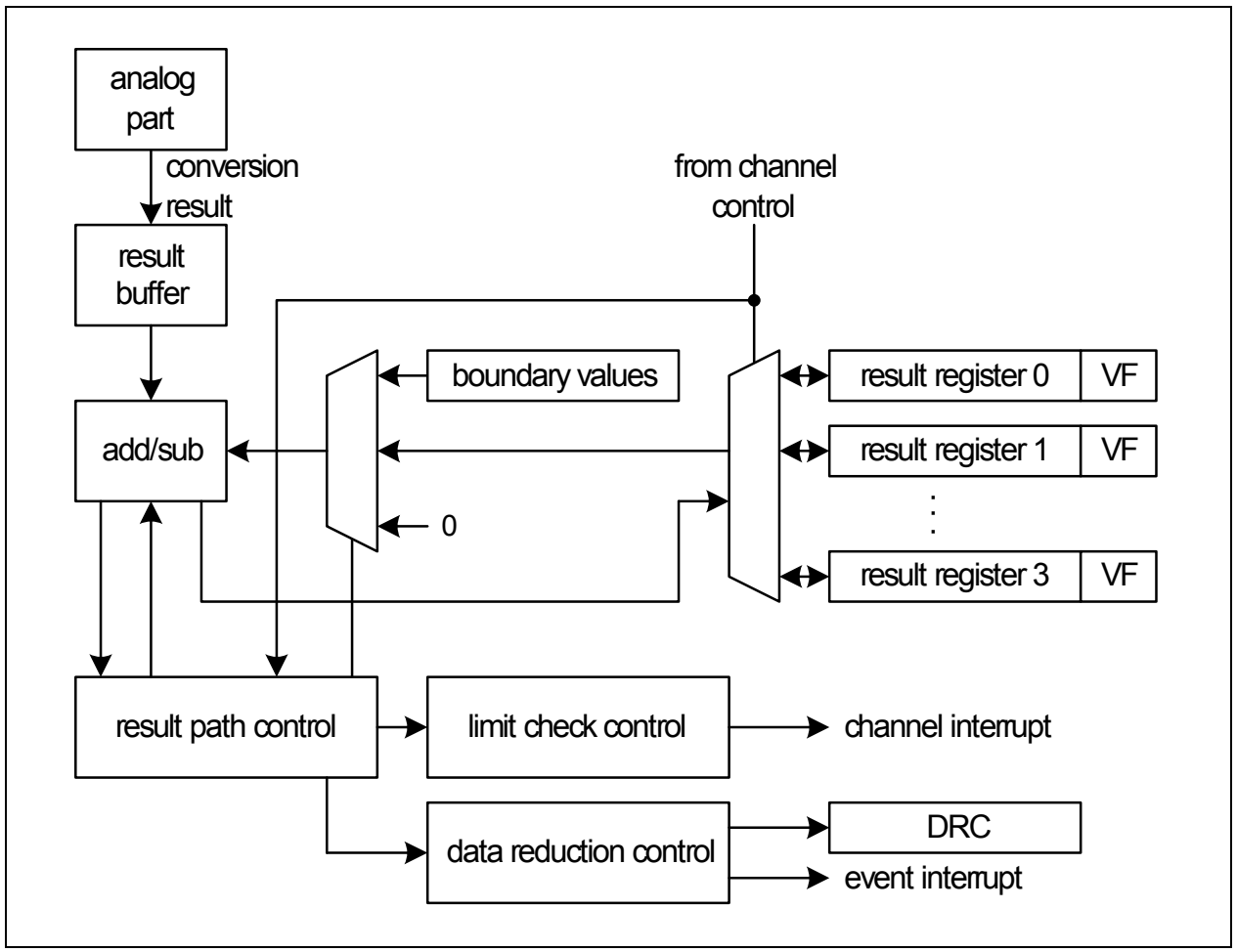



**Figure 17-8 Result Path**

Refer to **Section 17.7.8** for description of the result generation registers.

### 17.4.7.2 Limit Checking

The limit checking and the data reduction filter are based on a common add/subtract structure. The incoming result is compared with BOUND0, then with BOUND1. Depending on the result flags (lower-than compare), the limit checking unit can generate a channel interrupt. It can become active when the valid result of the data reduction filter is stored in the selected result register.

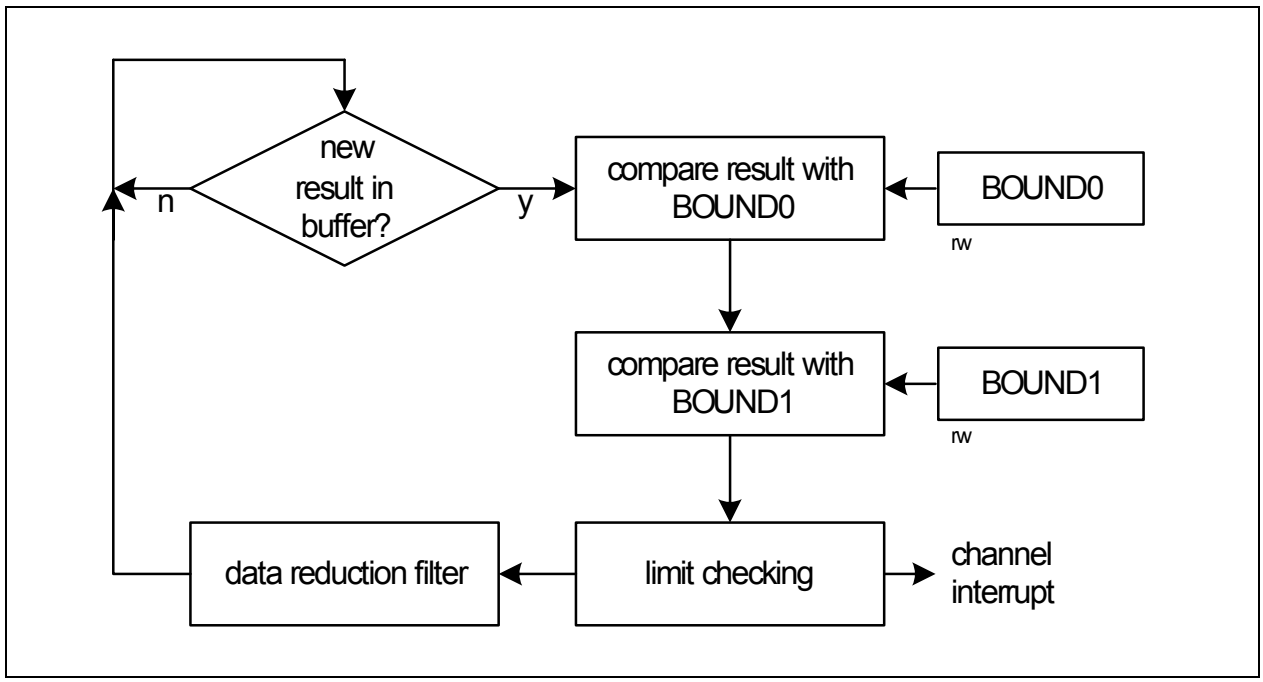



**Figure 17-9 Limit Checking Flow**

### 17.4.7.3 Data Reduction Filter

Each result register can be controlled to enable or disable the data reduction filter. The data reduction block allows the accumulation of conversion results for anti-aliasing filtering or for averaging.

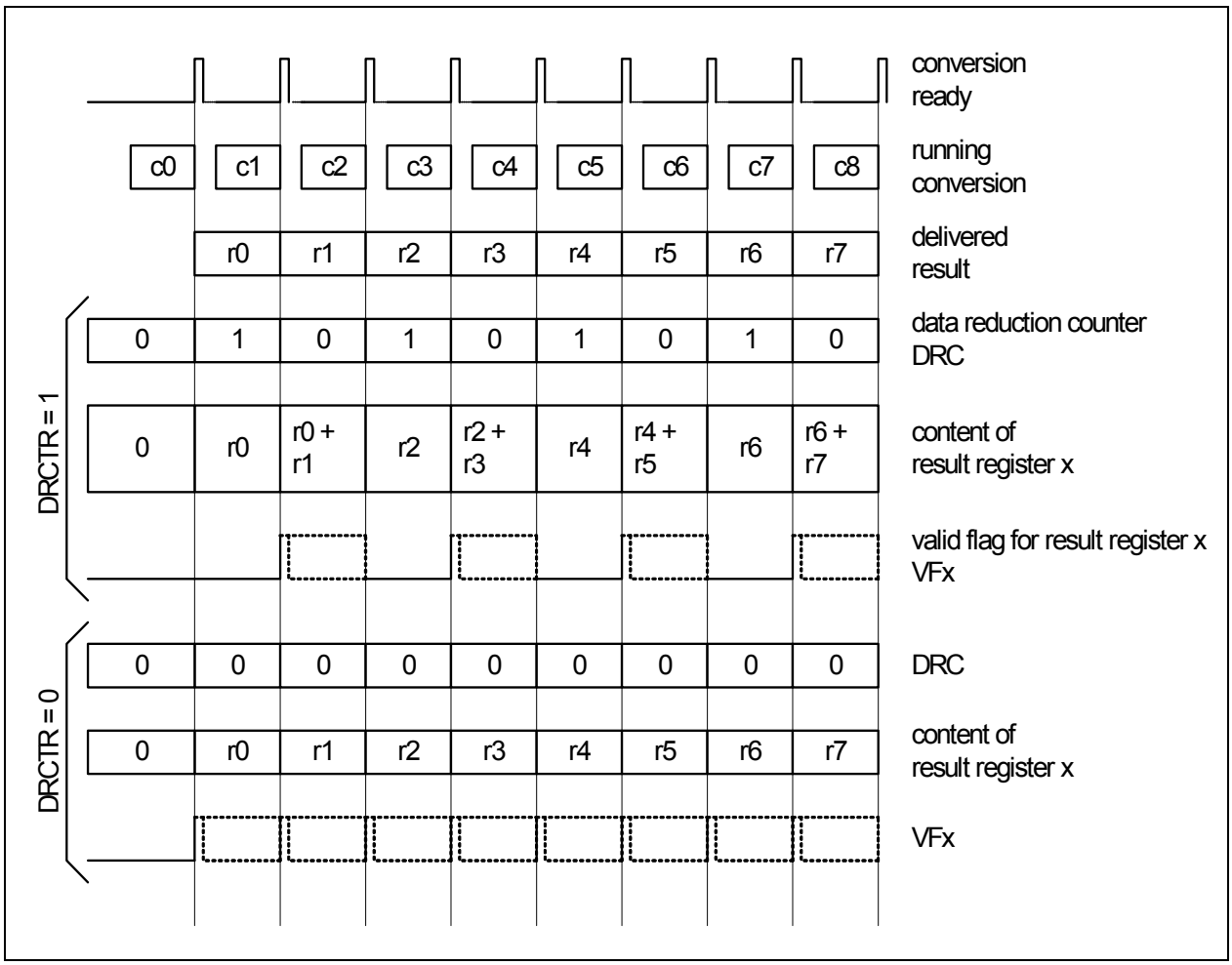



**Figure 17-10 Data Reduction Flow**

If DRC is 0 and a new conversion result comes in, DRC is reloaded with its reload value (defined by bit DRCTR in the result control register) and the value of 0 is added to the conversion result (instead of the previous result register content). Then, the complete result is stored in the selected result register. If the reload value is 0 (data reduction filter disabled), accumulation is done over one conversion. Hence, a result event is generated and the valid bit (VF) for the result register becomes set. If the reload value is 1 (data reduction filter enabled), accumulation is done over two conversions. In this case, neither a result event is generated nor the valid bit is set.

If DRC is 1 and a new conversion result comes in, the data reduction filter adds the incoming result to the value already stored in the result register and decrements DRC.

After this addition, the complete result is stored in the selected result register. The result event is generated and the valid bit becomes set.

It is possible to have an identical cycle behavior of the path to the result register, with the data reduction filter being enabled or disabled. Furthermore, an overflow of the result register is avoided, because a maximum of 2 conversion results are added (a 10-bit result added twice delivers a maximum of 11 bits).

#### 17.4.7.4 Result Register View

In order to cover a wide range of applications, the content of result register x (x = 0 - 3) is available as different read views at different addresses (see **Figure 17-11**):

- Normal read view RESRxL/H:
  This view delivers the 8-bit or 10-bit conversion result.
- Accumulated read view RESRAxL/H:
  This view delivers the accumulated 9-bit or 11-bit conversion result.

All conversion results (with or without accumulation) are stored in the result registers, but can be viewed at either RESRxL/H or RESRAxL/H which shows different data alignment and width.

When the data reduction filter is enabled (DRCTR = 1), read access should be performed on RESRAxL/H as it shows the full 9-bit (R8:R0) or 11-bit (R10:R0) accumulated conversion result. Reading from RESRxL/H gives the appended (MSB unavailable) accumulated result.

When the data reduction filter is disabled (DRCTR = 0), the user can read the 8-bit or 10-bit conversion result from either RESRxL/H or RESRAxL/H. In particular, for 8-bit conversion (without accumulation), the result can be read from RESRxH with a single instruction. Hence, depending on the application requirement, the user can choose to read from the different views.

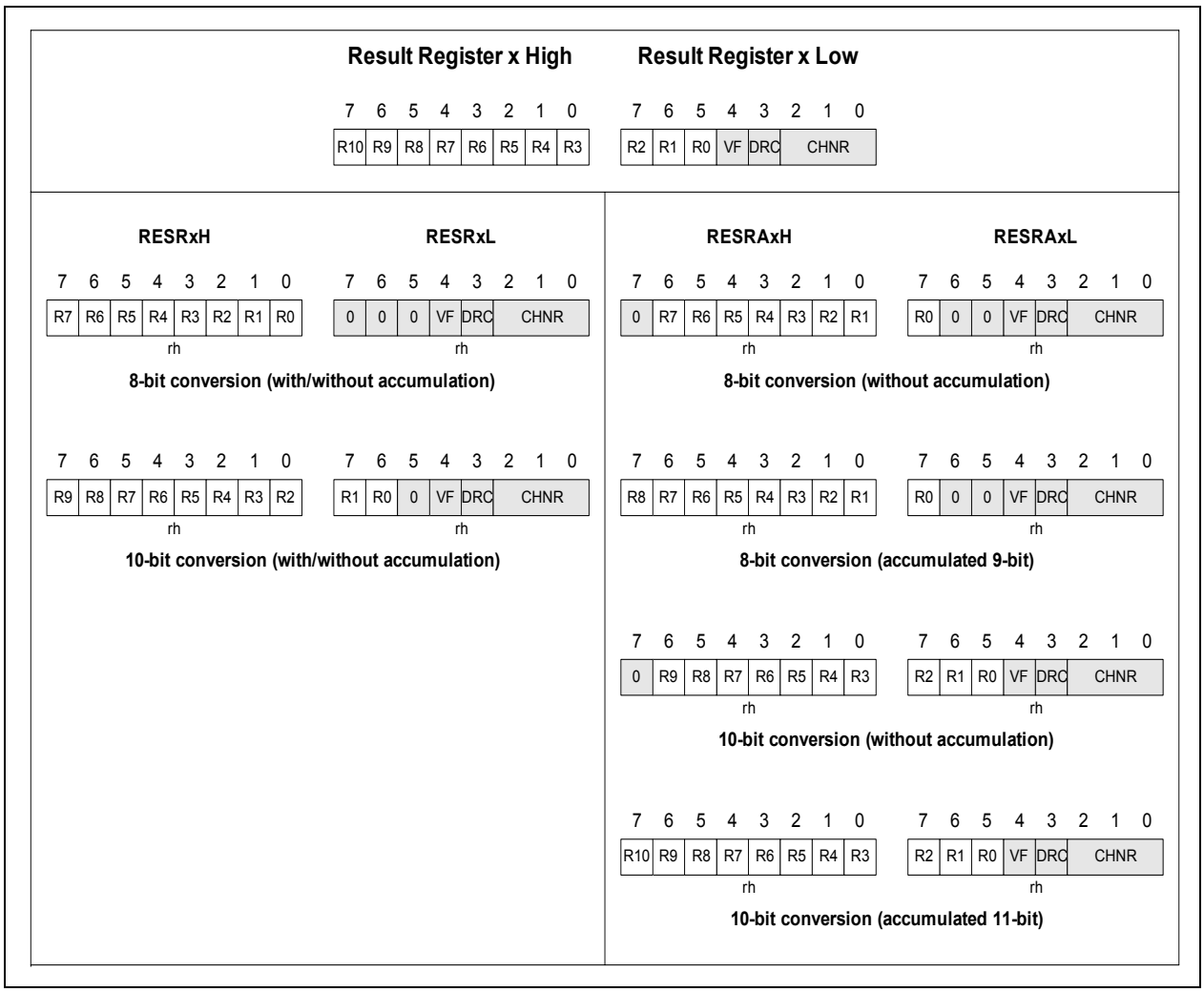



**Figure 17-11 Result Register View**

### 17.4.8 Interrupts

The ADC module provides 2 service request outputs SR[1:0] that can be activated by different interrupt sources.

The interrupt structure of the ADC supports two different types of interrupt sources:

- Event Interrupts: Activated by events of the request sources (source interrupts) or result registers (result interrupts).
- Channel Interrupts: Activated by the completion of any input channel conversion. They are enabled according to the control bits for the limit checking. The settings are defined individually for each input channel.

The interrupt compressor is an OR-combination of all incoming interrupt pulses for each of the SR lines.

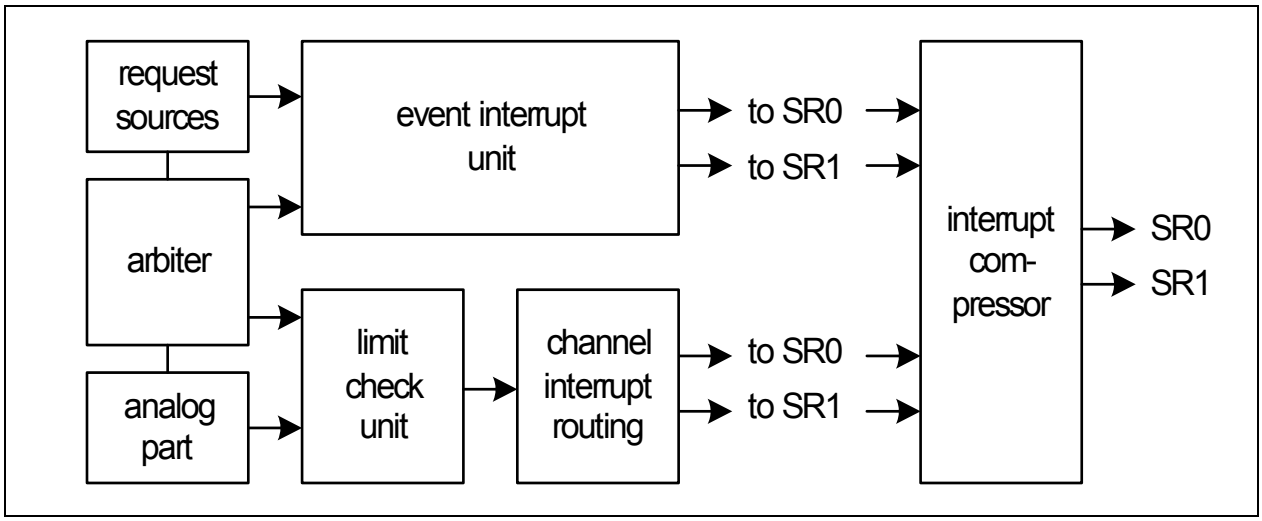



**Figure 17-12 Interrupt Overview**

Refer to **Section 17.7.9** for description of the interrupt registers.

### 17.4.8.1 Event Interrupts

Event interrupts can be generated by the request sources and the result registers. The event interrupt enable bits are located in the request sources (ENSI) and result register control (IEN). An interrupt node pointer (EVINP) for each event allows the selection of the targeted service output line.

A request source event is generated when the requested channel conversion is completed:

- Event 0: Request source event of sequential request source 0 (arbitration slot 0)
- Event 1: Request source event of parallel request source 1 (arbitration slot 1)

A result event is generated according to the data reduction control (see **Section 17.4.7.3**):

- Event 4: Result register event of result register 0
- Event 5: Result register event of result register 1
- Event 6: Result register event of result register 2
- Event 7: Result register event of result register 3

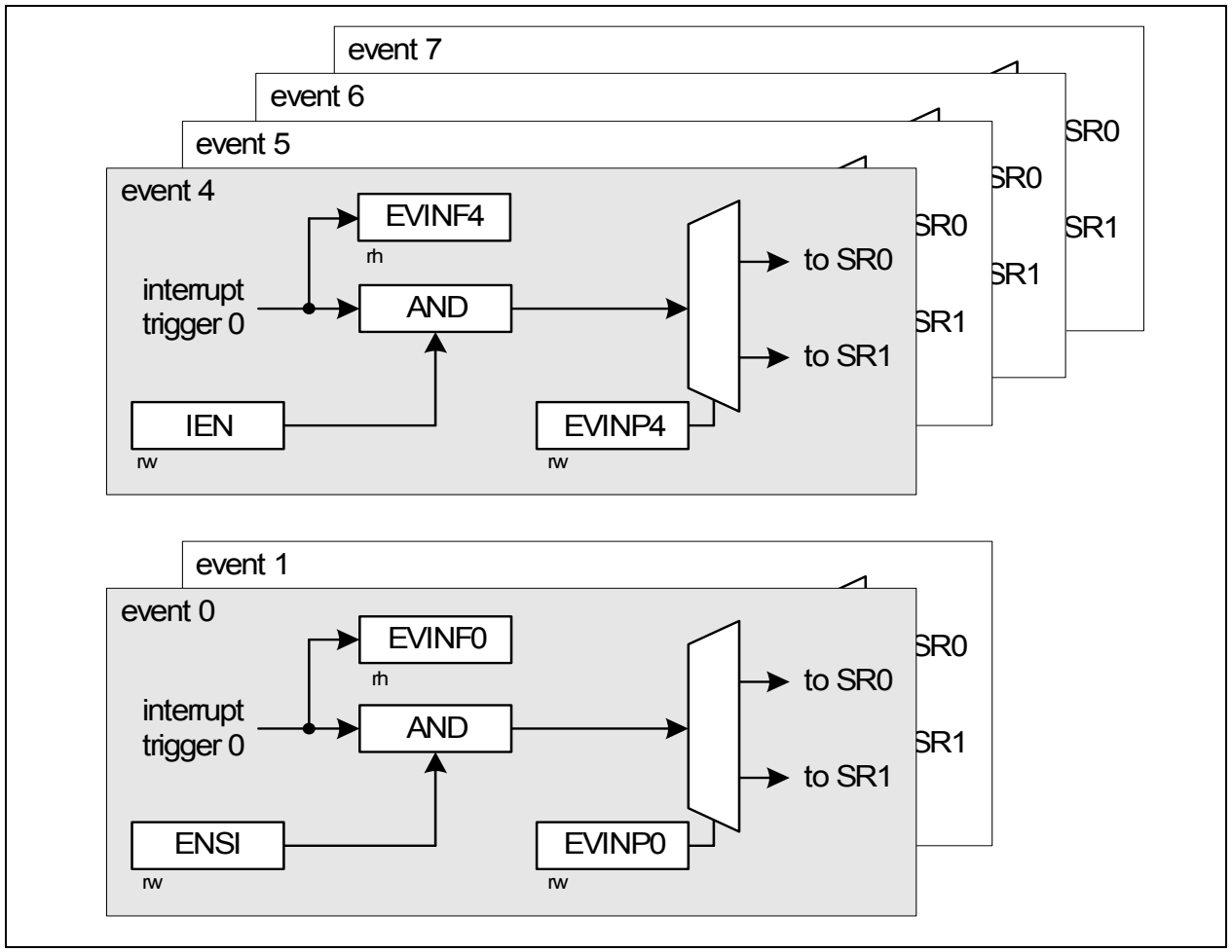



**Figure 17-13 Event Interrupt Structure**

### 17.4.8.2 Channel Interrupts

The channel interrupts occur when a conversion is completed and the selected limit checking condition is met. As a result, only one channel interrupt can be activated at a time. An interrupt can be triggered according to the limit checking result by comparing the conversion result with two selectable boundaries for each channel.

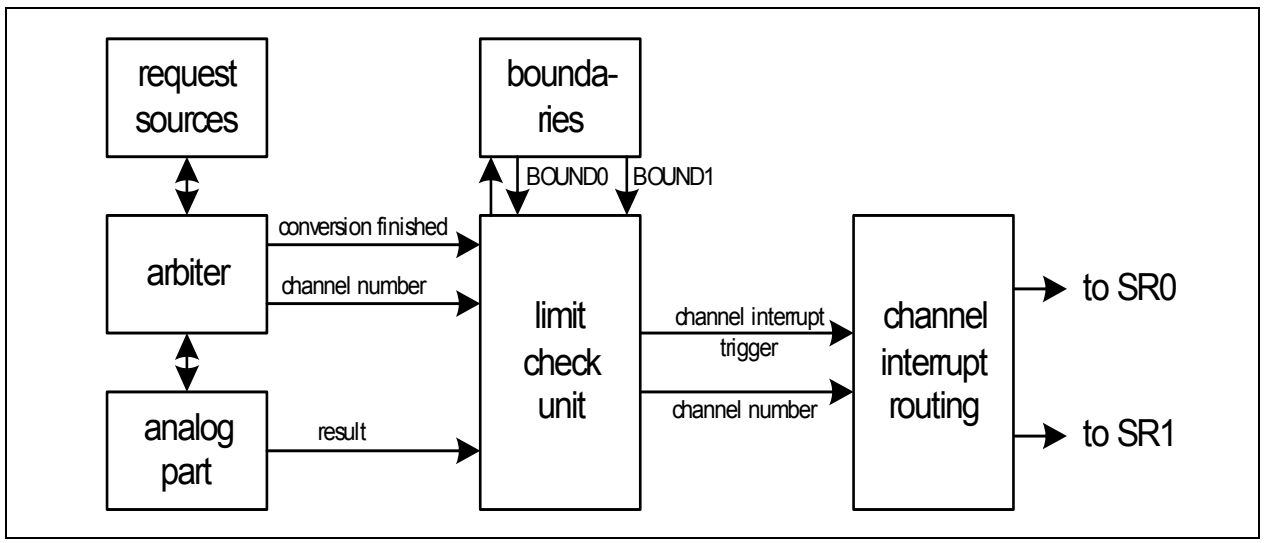



**Figure 17-14 Channel Interrupt Overview**

The limit checking unit uses two boundaries (BOUND0 and BOUND1) to compare with the conversion result. With these two boundaries, the conversion result space is split into three areas:

- Area I: The conversion result is below both boundaries.
- Area II: The conversion result is between the two boundaries, or is equal to one of the boundaries.
- Area III: The conversion result is above both boundaries.

After a conversion has been completed, a channel interrupt can be triggered according to the following conditions (selected by the limit check control bit field LCC):

- LCC = 000: No trigger, the channel interrupt is disabled.
- LCC = 001: A channel interrupt is generated if the conversion result is not in area I.
- LCC = 010: A channel interrupt is generated if the conversion result is not in area II.
- LCC = 011: A channel interrupt is generated if the conversion result is not in area III.
- LCC = 100: A channel interrupt is always generated (regardless of the boundaries).
- LCC = 101: A channel interrupt is generated if the conversion result is in area I.
- LCC = 110: A channel interrupt is generated if the conversion result is in area II.
- LCC = 111: A channel interrupt is generated if the conversion result is in area III.

The channel-specific interrupt node pointer CHINPx (x = 0 - 7) selects the service request output (SR[1:0]) that will be activated upon a channel interrupt trigger. See **Figure 17-15**.

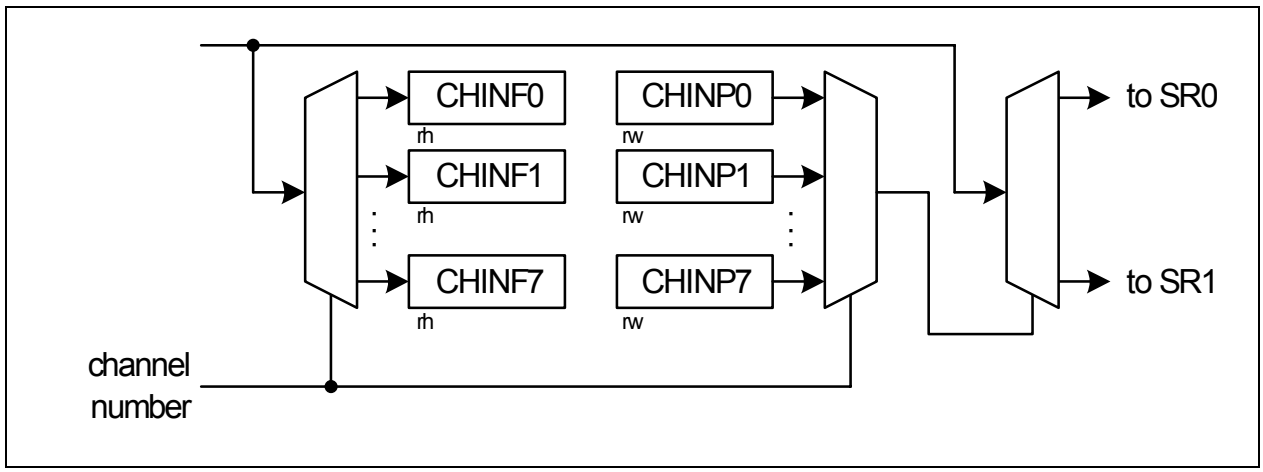



**Figure 17-15 Channel Interrupt Routing**

### 17.4.9 External Trigger Inputs

The sequential and parallel request sources has one request trigger input REQTRx (x = 0 - 1) each, through which a conversion request can be started. The input to REQTRx is selected from eight external trigger inputs (ETRx0 to ETRx7) via a multiplexer depending on bit field ETRSELx. It is possible to bypass the synchronization stages for external trigger requests that come synchronous to ADC. This selection is done via bit SYNENx.

Refer to **Section 17.7.9** for description of the external trigger control registers.

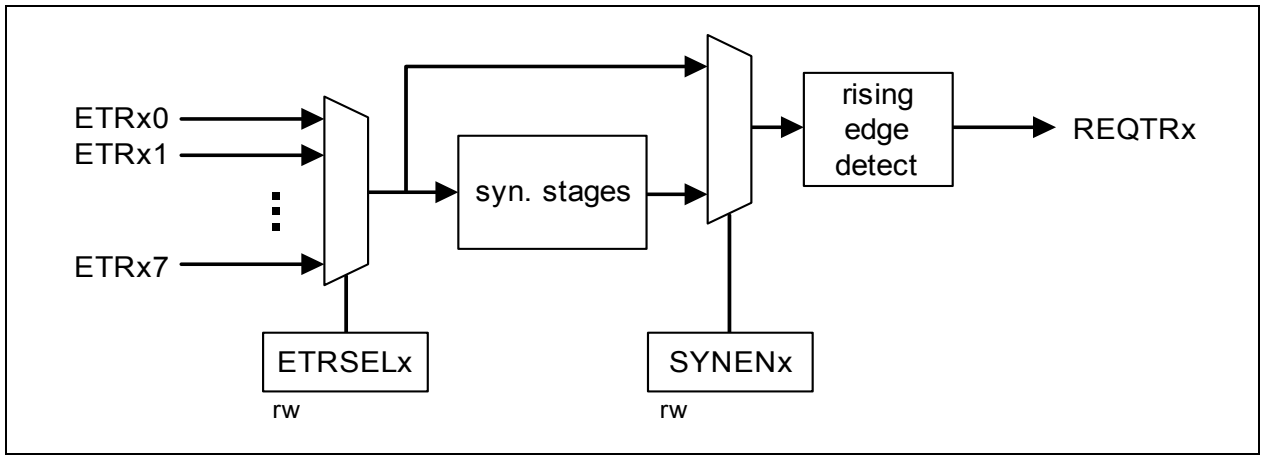



**Figure 17-16 External Trigger Input**

The external trigger inputs to the ADC module are driven by events occuring in the CCU6 module or T2CCU module. See **Table 17-2**. Inputs to ETRx0/1 are selectable by user via the respective bits in the MISC_CON register.

**Table 17-2 External Trigger Input Source**

| External Trigger Input | CCU6/T2CCU Event |
|---|---|
| ETRx0 | T13 period-match or CCT overflow |
| ETRx1 | T13 compare-match or T2CC5 compare-match |
| ETRx2 | T12 period-match |
| ETRx3 | T12 compare-match for channel 0 |
| ETRx4 | T12 compare-match for channel 1 |
| ETRx5 | T12 compare-match for channel 2 |
| ETRx6 | Shadow transfer event for multi-channel mode |
| ETRx7 | Correct hall event for multi-channel mode |

## MISC_CON
## Miscellaneous Control Register

**Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **ADCETR0_MUX** | **ADCETR1_MUX** | **0** | | | | | **DFLASHEN** |
| w | rw | r | | | | | rwh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **ADCETR1_MUX** | 6 | rwh | **ADC External Trigger Input 1 Event Select**<br>$0_B$ T13CM from CCU6 (default).<br>$1_B$ CM5F from T2CCU. |
| **ADCETR0_MUX** | 7 | w | **ADC External Trigger Input 0 Event Select**<br>$0_B$ T13PM from CCU6 (default).<br>$1_B$ CCTOVF from T2CCU. |
| **0** | [5:1] | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

## 17.5 ADC Module Initialization Sequence

The following steps is meant to provide a general guideline on how to initialize the ADC module. Some steps may be varied or omitted depending on the application requirements:

- Configure global control functions:
  - Select conversion width (GLOBCTR.DW)
  - Select analog clock $f_{ADCI}$ divider ratio (GLOBCTR.CTC)
- Configure arbitration control functions:
  - Select priority level for request source x (PRAR.PRIOx)
  - Select conversion start mode for request source x (PRAR.CSMx)
  - Enable arbitration slot x (PRAR.ASENx)
  - Select arbitration mode (PRAR.ARBM)
- Configure channel control information:
  - Select limit check control for channel x (CHCTRx.LCC)
  - Select target result register for channel x (CHCTRx.RESRSEL)
  - Select sample time for all channels (INPCR0.STC)
- Configure result control information:
  - Enable/disable data reduction for result register x (RCRx.DRCTR)
  - Enable/disable event interrupt for result register x (RCRx.IEN)
  - Enable/disable wait-for-read mode for result register x (RCRx.WFR)
  - Enable/disable valid flag reset by read access for result register x (RCRx.VFCTR)
- Configure interrupt control functions:
  - Select channel x interrupt node pointer (CHINPR.CHINPx)
  - Select event x interrupt node pointer (EVINPR.EVINPx)
- Configure limit check boundaries:
  - Select limit check boundaries for all channels (LCBR.BOUND0, LCBR.BOUND1)
- Configure external trigger control functions:
  - Select source x external trigger input (ETRCR.ETRSELx)
  - Enable/disable source x external trigger input synchronization (ETRCR.SYNENx)
- Setup sequential source:
  - Enable conversion request (QMR0.ENGT)
  - Enable/disable external trigger (QMR0.ENTR)
- Setup parallel source:
  - Enable conversion request (CRMR1.ENGT)
  - Enable/disable external trigger (CRMR1.ENTR)
  - Enable/disable source interrupt (CRMR1.ENSI)
  - Enable/disable autoscan (CRMR1.SCAN)
- Turn on analog part:
  - Set GLOBCTR.ANON (wait for 100 ns)
- Start sequential request:
  - Write to QINR0 (with information such as REQCHNR, RF, ENSI and EXTR)

  - Generate a pending conversion request using any method described in **Section 17.4.4.2**
- Start parallel request:
  - Write to CRCR1 (no load event) or CRPR1 (automatic load event) the channels to be converted.
  - Generate a load event (if not already available) to trigger a pending conversion request, using any method described in **Section 17.4.5.2**
- Wait for ADC conversion to be completed:
  - The source interrupt indicates that the conversion requested by the source is completed.
  - The channel interrupt indicates that the corresponding channel conversion is completed (with limit check performed).
  - The result interrupt indicates that the result (with/without accumulation) in the corresponding result register is ready and can be read.
- Read ADC result

## 17.6 Register Map

All ADC register names described in the following sections are referenced in other chapters of this document with the module name prefix “ADC_”, e.g., ADC_GLOBCTR.

The addresses of the ADC SFRs are listed in **Table 17-3** and **Table 17-4**

**Table 17-3 SFR Address List for Pages 0 - 3**

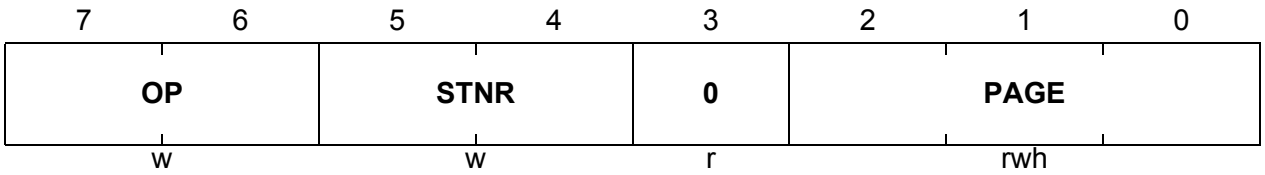
| Address | Page 0 | Page 1 | Page 2 | Page 3 |
|---|---|---|---|---|
| $CA_H$ | GLOBCTR | CHCTR0 | RESR0L | RESRA0L |
| $CB_H$ | GLOBSTR | CHCTR1 | RESR0H | RESRA0H |
| $CC_H$ | PRAR | CHCTR2 | RESR1L | RESRA1L |
| $CD_H$ | LCBR | CHCTR3 | RESR1H | RESRA1H |
| $CE_H$ | INPCR0 | CHCTR4 | RESR2L | RESRA2L |
| $CF_H$ | ETRCR | CHCTR5 | RESR2H | RESRA2H |
| $D2_H$ | – | CHCTR6 | RESR3L | RESRA3L |
| $D3_H$ | – | CHCTR7 | RESR3H | RESRA3H |

**Table 17-4 SFR Address List for Pages 4 - 7**

| Address | Page 4 | Page 5 | Page 6 | Page 7 |
|---|---|---|---|---|
| $CA_H$ | RCR0 | CHINFR | CRCR1 | – |
| $CB_H$ | RCR1 | CHINCR | CRPR1 | – |
| $CC_H$ | RCR2 | CHINSR | CRMR1 | – |
| $CD_H$ | RCR3 | CHINPR | QMR0 | – |
| $CE_H$ | VFCR | EVINFR | QSR0 | – |
| $CF_H$ | – | EVINCR | Q0R0 | – |
| $D2_H$ | – | EVINSR | QBUR0/QINR0 | – |
| $D3_H$ | – | EVINPR | – | – |

The ADC SFRs are located in the standard memory area (RMAP = 0) and are organized into 7 pages. The ADC_PAGE register is located at address $D1_H$. It contains the page value and page control information.

**ADC_PAGE**
**Page Register for ADC** **($D1_H$)** **Reset Value: $00_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **PAGE** | [2:0] | rwh | **Page Bits**<br>When written, the value indicates the new page address.<br>When read, the value indicates the currently active page. |
| **STNR** | [5:4] | w | **Storage Number**<br>This number indicates which storage bit field is the target of the operation defined by bit OP.<br>If OP = $10_B$,<br>the contents of PAGE are saved in STx before being overwritten with the new value.<br>If OP = $11_B$,<br>the contents of PAGE are overwritten by the contents of STx. The value written to the bit positions of PAGE is ignored.<br>$00_B$ ST0 is selected.<br>$01_B$ ST1 is selected.<br>$10_B$ ST2 is selected.<br>$11_B$ ST3 is selected. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **OP** | [7:6] | w | **Operation**<br>$0X_B$ Manual page mode. The value of STNR is ignored and PAGE is directly written.<br>$10_B$ New page programming with automatic page saving. The value written to the bit positions of PAGE is stored. In parallel, the former contents of PAGE are saved in the storage bit field STx indicated by STNR.<br>$11_B$ Automatic restore page action. The value written to the bit positions PAGE is ignored and instead, PAGE is overwritten by the contents of the storage bit field STx indicated by STNR. |
| **0** | 3 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

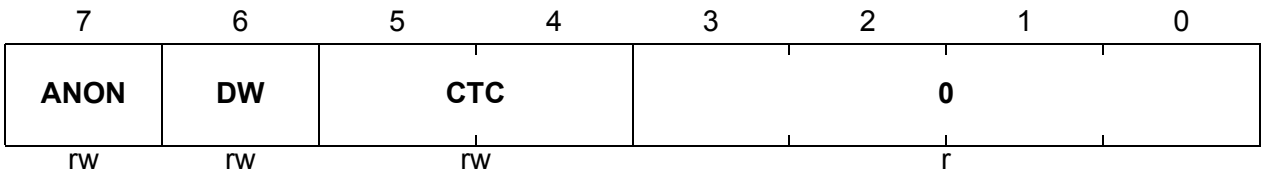
## 17.7 Register Description

This section describes all the registers which are associated with the functionalities of the ADC module.

### 17.7.1 General Function Registers

Register GLOBCTR contains bits that control the analog converter and the conversion delay.

**GLOBCTR**
**Global Control Register** **($CA_H$)** **Reset Value: $30_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **CTC** | [5:4] | w | **Conversion Time Control**<br>This bit field defines the divider ratio for the divider stage of the internal analog clock $f_{ADCI}$. This clock provides the internal time base for the conversion and sample time calculations.<br>$00_B$ $f_{ADCI}$ = 1/2 × $f_{ADCA}$<br>$01_B$ $f_{ADCI}$ = 1/3 × $f_{ADCA}$<br>$10_B$ $f_{ADCI}$ = 1/4 × $f_{ADCA}$<br>$11_B$ $f_{ADCI}$ = 1/32 × $f_{ADCA}$ (default) |
| **DW** | 6 | rw | **Data Width**<br>This bit defines the conversion resolution.<br>$0_B$ The result is 10 bits wide (default).<br>$1_B$ The result is 8 bits wide. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **ANON** | 7 | rw | **Analog Part Switched On**<br>This bit enables the analog part of the ADC module and defines its operation mode.<br>$0_B$ The analog part is switched off and conversions are not possible. To achieve minimal power consumption, the internal analog circuitry is in its power-down state and the generation of $f_{ADCI}$ is stopped.<br>$1_B$ The analog part of the ADC module is switched on and conversions are possible. The automatic power-down capability of the analog part is disabled. |
| **0** | [3:0] | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

Register GLOBSTR contains bits that indicate the current status of a conversion.

**GLOBSTR**
**Global Status Register** **($CB_H$)** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **0** | | **CHNR** | | | **0** | **SAMPLE** | **BUSY** |
| r | | rh | | | r | rh | rh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **BUSY** | 0 | rh | **Analog Part Busy**<br>This bit indicates that a conversion is currently active.<br>$0_B$ The analog part is idle.<br>$1_B$ A conversion is currently active. |
| **SAMPLE** | 1 | rh | **Sample Phase**<br>This bit indicates that an analog input signal is currently sampled.<br>$0_B$ The analog part is not in the sampling phase.<br>$1_B$ The analog part is in the sampling phase. |
| **CHNR** | [5:3] | rh | **Channel Number**<br>This bit field indicates which analog input channel is currently converted. This information is updated when a new conversion is started. |
| **0** | 2, [7:6] | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

### 17.7.2 Priority and Arbitration Register

Register PRAR contains bits that define the request source priority and the conversion start mode. It also contains bits that enable/disable the conversion request treatment in the arbitration slots.

**PRAR**
**Priority and Arbitration Register ($CC_H$) Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **ASEN1** | **ASEN0** | **0** | **ARBM** | **CSM1** | **PRIO1** | **CSM0** | **PRIO0** |
| rw | rw | r | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **PRIO0** | 0 | rw | **Priority of Request Source 0**<br>This bit defines the priority of the sequential request source 0.<br>$0_B$ Low priority<br>$1_B$ High priority |
| **CSM0** | 1 | rw | **Conversion Start Mode of Request Source 0**<br>This bit defines the conversion start mode of the sequential request source 0.<br>$0_B$ The wait-for-start mode is selected.<br>$1_B$ The cancel-inject-repeat mode is selected. |
| **PRIO1** | 2 | rw | **Priority of Request Source 1**<br>This bit defines the priority of the parallel request source 1.<br>$0_B$ Low priority<br>$1_B$ High priority |
| **CSM1** | 3 | rw | **Conversion Start Mode of Request Source 1**<br>This bit defines the conversion start mode of the parallel request source 1.<br>$0_B$ The wait-for-start mode is selected.<br>$1_B$ The cancel-inject-repeat mode is selected. |
| **ARBM** | 4 | rw | **Arbitration Mode**<br>This bit defines which arbitration mode is selected.<br>$0_B$ Permanent arbitration (default).<br>$1_B$ Arbitration started by pending conversion request |

| Field | Bits | Type | Description |
|---|---|---|---|
| **ASENx (x = 0 - 1)** | [7:6] | rw | **Arbitration Slot x Enable**<br>Each bit enables an arbitration slot of the arbiter round. ASEN0 enables arbitration slot 0, ASEN1 enables slot 1.<br>If an arbitration slot is disabled, a pending conversion request of a request source connected to this slot is not taken into account for arbitration.<br>$0_B$ The corresponding arbitration slot is disabled. Conversions are not requested, pending conversion request of a request source connected to slot x is not taken into account for arbitration. If the arbiter shall not be running continously (ARBM = 1), no conversion request of the request source for arbitration slot x must be active. Clear conversion requests of the related request source before disabling an arbitration slot.<br>$1_B$ The corresponding arbitration slot is enabled. Conversions are requested for the request source with pending request bit(s). |
| **0** | 5 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

### 17.7.3 External Trigger Control Register

Register ETRCR contains bits that select the external trigger input signal source and enable synchronization of the external trigger input.

**ETRCR**
**External Trigger Control Register ($CF_H$) Reset Value: $00_H$**

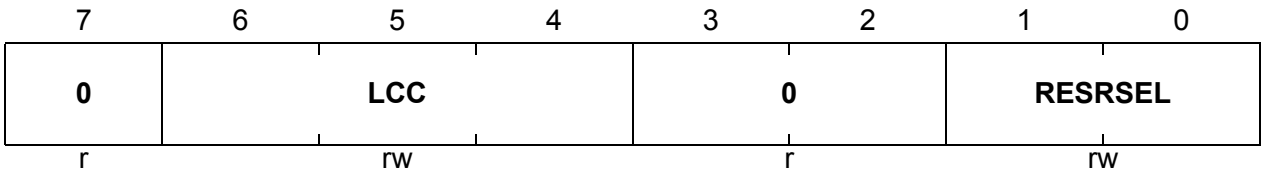
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **SYNEN1** | **SYNEN0** | **ETRSEL1** | | | **ETRSEL0** | | |
| rw | rw | rw | | | rw | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **ETRSELx (x = 0 - 1)** | [2:0], [5:3] | rw | **External Trigger Selection for Request Source x**<br>This bit field defines which external trigger input signal is selected.<br>$000_B$ The trigger input ETRx0 is selected.<br>$001_B$ The trigger input ETRx1 is selected.<br>$010_B$ The trigger input ETRx2 is selected.<br>$011_B$ The trigger input ETRx3 is selected.<br>$100_B$ The trigger input ETRx4 is selected.<br>$101_B$ The trigger input ETRx5 is selected.<br>$110_B$ The trigger input ETRx6 is selected.<br>$111_B$ The trigger input ETRx7 is selected. |
| **SYNENx (x = 0 - 1)** | [7:6] | rw | **Synchronization Enable**<br>$0_B$ Synchronizing stage is not in external trigger input REQTRx path.<br>$1_B$ Synchronizing stage is in external trigger input REQTRx path. |

### 17.7.4 Channel Control Registers

The channel control registers contain bits that select the targeted result register and control the limit check mechanism. Register CHCTRx defines the settings for the input channel x.

**CHCTRx (x = 0 - 7)**
**Channel Control Register x** **($CA_H$ + x * 1)** **Reset Value: $00_H$**

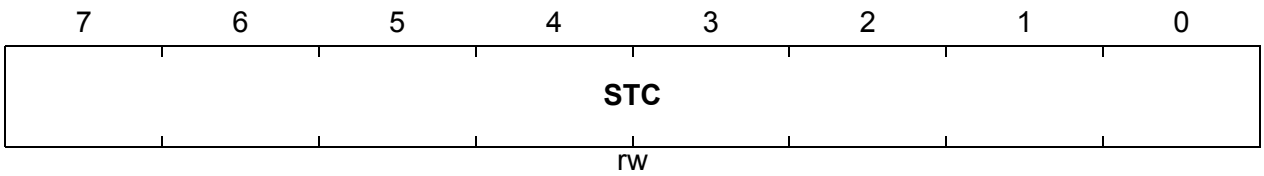



| Field | Bits | Type | Description |
|---|---|---|---|
| **RESRSEL** | [1:0] | rw | **Result Register Selection**<br>This bit field defines which result register will be the target of a conversion of this channel.<br>$00_B$ The result register 0 is selected.<br>$01_B$ The result register 1 is selected.<br>$10_B$ The result register 2 is selected.<br>$11_B$ The result register 3 is selected. |
| **LCC** | [6:4] | rw | **Limit Check Control**<br>This bit field defines the behavior of the limit checking mechanism.<br>See coding in **Section 17.4.8.2**. |
| **0** | [3:2], 7 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

### 17.7.5 Input Class Register

Register INPCR0 contains bits that control the sample time for the input channels.

**INPCR0**
**Input Class 0 Register** **($CE_H$)** **Reset Value: $00_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **STC** | [7:0] | rw | **Sample Time Control**<br>This bit field defines the additional length of the sample time, given in terms of $f_{ADCI}$ clock cycles.<br>A sample time of 2 analog clock cycles is extended by the programmed value. |

### 17.7.6 Sequential Source Registers

These registers contain the control and status bits of sequential request source 0.

Register QMR0 contains bits that are used to set the sequential request source in the desired mode.

**QMR0**
**Queue Mode Register** **($CD_H$)** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **CEV** | **TREV** | **FLUSH** | **CLRV** | **0** | **ENTR** | **0** | **ENGT** |
| w | w | w | w | r | rw | r | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **ENGT** | 0 | rw | **Enable Gate**<br>This bit enables the gating functionality for the request source.<br>$0_B$ The gating line is permanently 0. The source is switched off.<br>$1_B$ The gating line is permanently 1. The source is switched on. |
| **ENTR** | 2 | rw | **Enable External Trigger**<br>This bit enables the external trigger possibility. If enabled, bit EV is set if a rising edge is detected at the external trigger input REQTR when at least one V bit is set in register Q0R0 or QBUR0.<br>$0_B$ The external trigger is disabled.<br>$1_B$ The external trigger is enabled. |
| **CLRV** | 4 | w | **Clear V Bits**<br>$0_B$ No action<br>$1_B$ The bit V in register Q0R0 or QBUR0 is reset. If QBUR0.V = 1, then QBUR0.V is reset. If QBUR0.V = 0, then Q0R0.V is reset. |
| **FLUSH** | 5 | w | **Flush Queue**<br>$0_B$ No action<br>$1_B$ All bits V in the queue registers and bit EV are reset. The queue contains no more valid entry. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **TREV** | 6 | w | **Trigger Event**<br>$0_B$ No action<br>$1_B$ A trigger event is generated by software. If the source waits for a trigger event, a conversion request is started. |
| **CEV** | 7 | w | **Clear Event Bit**<br>$0_B$ No action<br>$1_B$ Bit EV is cleared. |
| **0** | 1, 3 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

Register QSR0 contains bits that indicate the status of the sequential source.

**QSR0**
**Queue Status Register** **($CE_H$)** **Reset Value: $20_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Rsv | 0 | EMPTY | EV | 0 | | FILL | |
| r | r | rh | rh | r | | rh | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **FILL** | [1:0] | rh | **Filling Level**<br>This bit field indicates how many entries are valid in the sequential-sourced queue. It is incremented each time a new entry is written to QINR0, decremented each time a requested conversion has been finished. A new entry is ignored if the filling level has reached its maximum value. If EMPTY bit = 1, there are no valid entries in the queue.<br>$00_B$ If EMPTY bit = 0, there is 1 valid entry in the queue.<br>$01_B$ If EMPTY bit = 0, there are 2 valid entries in the queue.<br>$10_B$ If EMPTY bit = 0, there is 3 valid entry in the queue.<br>$11_B$ If EMPTY bit = 0, there are 4 valid entries in the queue. |
| **EV** | 4 | rh | **Event Detected**<br>This bit indicates that an event has been detected while V = 1. Once set, this bit is reset automatically when the requested conversion is started.<br>$0_B$ An event has not been detected.<br>$1_B$ An event has been detected. |
| **EMPTY** | 5 | rh | **Queue Empty**<br>This bit indicates if the sequential source contains valid entries. A new entry is ignored if the queue is filled (EMPTY = 0).<br>$0_B$ The queue is filled with 'FILL+1' valid entries in the queue.<br>$1_B$ The queue is empty, no valid entries are present in the queue. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **Rsv** | 7 | r | **Reserved**<br>Returns 1 if read; should be written with 0.<br>*Note: This bit is initialized to 0 immediately after reset, but is updated by hardware to 1 (and remains as 1) shortly after.* |
| **0** | [3:0], 6 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

Register Q0R0 contains bits that monitor the status of the current sequential request.

**Q0R0**
**Queue 0 Register 0** **($CF_H$)** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **EXTR** | **ENSI** | **RF** | **V** | **0** | **REQCHNR** | | |
| rh | rh | rh | rh | r | rh | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **REQCHNR** | [2:0] | rh | **Request Channel Number**<br>This bit field indicates the channel number that will be or is currently requested. |
| **V** | 4 | rh | **Request Channel Number Valid**<br>This bit indicates if the data in REQCHNR, RF, ENSI and EXTR is valid. Bit V is set when a valid entry is written to the queue input register QINR0 (or by an update by intermediate queue registers).<br>$0_B$ The data is not valid.<br>$1_B$ The data is valid. |
| **RF** | 5 | rh | **Refill**<br>This bit indicates if the pending request is discarded after being executed (conversion start) or if it is automatically refilled in the top position of the request queue.<br>$0_B$ The request is discarded after conversion start.<br>$1_B$ The request is refilled in the queue after conversion start. |
| **ENSI** | 6 | rh | **Enable Source Interrupt**<br>This bit indicates if a source interrupt will be generated when the conversion is completed. The interrupt trigger becomes activated if the conversion requested by the source has been completed and ENSI = 1.<br>$0_B$ The source interrupt generation is disabled.<br>$1_B$ The source interrupt generation is enabled. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **EXTR** | 7 | rh | **External Trigger**<br>This bit defines if the conversion request is sensitive to an external trigger event.<br>The event flag (bit EV) indicates if an external event has taken place and a conversion can be requested.<br>$0_B$ Bit EV is not used to start conversion request.<br>$1_B$ Bit EV is used to start conversion request. |
| **0** | 3 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

The registers QBUR0 and QINR0 share the same register address. A read operation at this register address will deliver the ‘rh’ bits of the QBUR0 register, while a write operation to the same address will target the ‘w’ bits of the QINR0 register.

Register QBUR0 contains bits that monitor the status of an aborted sequential request.

**QBUR0**
**Queue Backup Register 0** **($D2_H$)** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **EXTR** | **ENSI** | **RF** | **V** | **0** | **REQCHNR** | | |
| rh | rh | rh | rh | r | rh | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **REQCHNR** | [2:0] | rh | **Request Channel Number**<br>This bit field is updated by bit field Q0R0.REQCHNR when the conversion requested by Q0R0 is started. |
| **V** | 4 | rh | **Request Channel Number Valid**<br>This bit indicates if the data in REQCHNR, RF, ENSI, and EXTR is valid. Bit V is set if a running conversion is aborted. It is reset when the conversion is started.<br>$0_B$ The backup register does not contain valid data, because the conversion described by this data has not been aborted.<br>$1_B$ The data is valid. The aborted conversion is requested before taking into account what is requested by Q0R0. |
| **RF** | 5 | rh | **Refill**<br>This bit is updated by bit Q0R0.RF when the conversion requested by Q0R0 is started. |
| **ENSI** | 6 | rh | **Enable Source Interrupt**<br>This bit is updated by bit Q0R0.ENSI when the conversion requested by Q0R0 is started. |
| **EXTR** | 7 | rh | **External Trigger**<br>This bit is updated by bit Q0R0.EXTR when the conversion requested by Q0R0 is started. |
| **0** | 3 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

Register QINR0 is the entry register for sequential requests.

**QINR0**
**Queue Input Register 0** **($D2_H$)** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **EXTR** | **ENSI** | **RF** | **0** | | **REQCHNR** | | |
| w | w | w | r | | w | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **REQCHNR** | [2:0] | w | **Request Channel Number**<br>This bit field defines the requested channel number. |
| **RF** | 5 | w | **Refill**<br>This bit defines the refill functionality. |
| **ENSI** | 6 | w | **Enable Source Interrupt**<br>This bit defines the source interrupt functionality. |
| **EXTR** | 7 | w | **External Trigger**<br>This bit defines the external trigger functionality. |
| **0** | [4:3] | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

### 17.7.7 Parallel Source Registers

These registers contain the control and status bits of parallel request source 1.

Register CRCR1 contains the bits that are copied to the pending register (CRPR1) when the load event occurs. This register can be accessed at two different addresses (one read view, two write views). The first address for read and write access is the address given for CRCR1. The second address for write actions is given for CRPR1. A write operation to CRPR1 leads to a data write to the bits in CRCR1 with an automatic load event one clock cycle later.

**CRCR1**
**Conversion Request Control Register 1($CA_H$)** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **CH7** | **CH6** | **CH5** | **CH4** | **0** | | | |
| rwh | rwh | rwh | rwh | r | | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **CHx (x = 4 - 7)** | x | rwh | **Channel Bit x**<br>Each bit corresponds to one analog channel, the channel number x is defined by the bit position in the register. The corresponding bit x in the conversion request pending register will be overwritten by this bit when the load event occurs.<br>$0_B$ The analog channel x will not be requested for conversion by the parallel request source.<br>$1_B$ The analog channel x will be requested for conversion by the parallel request source. |
| **0** | [3:0] | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

Register CRPR1 contains bits that request a conversion of the corresponding analog channel. The bits in this register have only a read view. A write operation to this address leads to a data write to CRCR1 with an automatic load event one clock cycle later.

**CRPR1**
**Conversion Request Pending Register 1($CB_H$)** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **CHP7** | **CHP6** | **CHP5** | **CHP4** | **0** | | | |
| rwh | rwh | rwh | rwh | r | | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **CHPx (x = 4 - 7)** | x | rwh | **Channel Pending Bit x**<br>Write view:<br>A write to this address targets the bits in register CRCR1.<br>Read view:<br>Each bit corresponds to one analog channel; the channel number x is defined by the bit position in the register.<br>The arbiter automatically resets (at start of conversion) or sets it again (at abort of conversion) for the corresponding analog channel.<br>$0_B$ The analog channel x is not requested for conversion by the parallel request source.<br>$1_B$ The analog channel x is requested for conversion by the parallel request source. |
| **0** | [3:0] | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

*Note: The bits that can be read from this register location are generally ‘rh’. They cannot be modified directly by a write operation. A write operation modifies the bits in CRCR1 (that is why they are marked ‘rwh’) and leads to a load event one clock cycle later.*

Register CRMR1 contains bits that are used to set the request source in the desired mode.

**CRMR1**
**Conversion Request Mode Register 1 ($CC_H$)** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **Rsv** | **LDEV** | **CLRPND** | **SCAN** | **ENSI** | **ENTR** | **0** | **ENGT** |
| r | w | w | rw | rw | rw | r | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **ENGT** | 0 | rw | **Enable Gate**<br>This bit enables the gating functionality for the request source.<br>$0_B$ The gating line is permanently 0. The source is switched off.<br>$1_B$ The gating line is permanently 1. The source is switched on. |
| **ENTR** | 2 | rw | **Enable External Trigger**<br>This bit enables the external trigger possibility. If enabled, the load event takes place if a rising edge is detected at the external trigger input REQTR.<br>$0_B$ The external trigger is disabled.<br>$1_B$ The external trigger is disabled. |
| **ENSI** | 3 | rw | **Enable Source Interrupt**<br>This bit enables the request source interrupt. This interrupt can be generated when the last pending conversion is completed for this source (while PND = 0).<br>$0_B$ The source interrupt is disabled.<br>$1_B$ The source interrupt is enabled. |
| **SCAN** | 4 | rw | **Autoscan Enable**<br>This bit enables the autoscan functionality. If enabled, the load event is automatically generated when a conversion (requested by this source) is completed and PND = 0.<br>$0_B$ The autoscan functionality is disabled.<br>$1_B$ The autoscan functionality is enabled. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **CLRPND** | 5 | w | **Clear Pending Bits**<br>$0_B$ No action<br>$1_B$ The bits in register CRPR1 are reset. |
| **LDEV** | 6 | w | **Generate Load Event**<br>$0_B$ No action<br>$1_B$ The load event is generated. |
| **Rsv** | 7 | r | **Reserved**<br>Returns 1 if read; should be written with 0.<br>*Note: This bit is initialized to 0 immediately after reset, but is updated by hardware to 1 (and remains as 1) shortly after.* |
| **0** | 1 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

### 17.7.8 Result Registers

The result registers deliver the conversion results and, optionally, the channel number that has lead to the latest update of the result register. The result registers are available as different read views at different addresses. The following bit fields can be read from the result registers, depending on the selected read address. For details on the conversion result alignment and width, see **Section 17.4.7.4**.

#### Normal Read View RESRx

This view delivers the 8-bit or 10-bit conversion result and a 3-bit channel number. The corresponding valid flag is cleared when the high byte of the register is accessed by a read command, provided that bit RCRx.VFCTR is set.

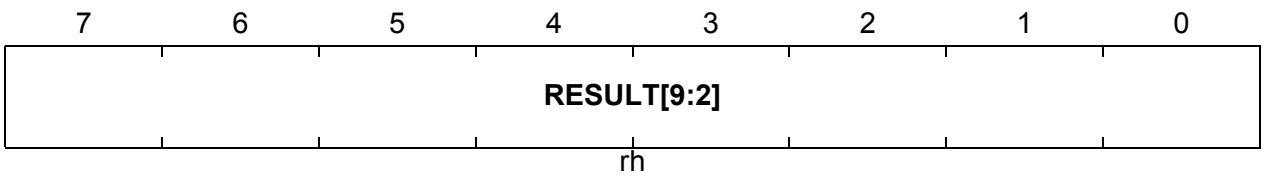
**RESRxL (x = 0 - 3)**
**Result Register x Low** **($CA_H$ + x * 2)** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| RESULT[1:0] | | 0 | VF | DRC | CHNR | | |
| rh | | r | rh | rh | rh | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **CHNR** | [2:0] | rh | **Channel Number**<br>This bit field contains the channel number of the latest register update. |
| **DRC** | 3 | rh | **Data Reduction Counter**<br>This bit field indicates how many conversion results have still to be accumulated to generate the final result for data reduction.<br>$0_B$ The final result is available in the result register.The valid flag is automatically set when this bit field is set to 0.<br>$1_B$ 1 more conversion result must be added to obtain the final result in the result register.The valid flag is automatically reset when this bit field is set to 1. |

| Field | Bits | Type | Description |
|---|---|---|---|
| **VF** | 4 | rh | **Valid Flag for Result Register x**<br>This bit indicates that the contents of the result register x are valid.<br>$0_B$ The result register x does not contain valid data.<br>$1_B$ The result register x contains valid data. |
| **RESULT[1:0]** | [7:6] | rh | **Conversion Result**<br>This bit field contains the conversion result or the result of the data reduction filter. |
| **0** | 5 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

**RESRxH (x = 0 - 3)**
**Result Register x High** **($CB_H$ + x * 2)** **Reset Value: $00_H$**

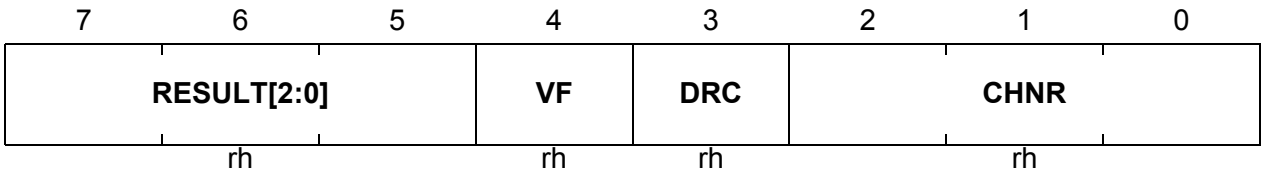



| Field | Bits | Type | Description |
|---|---|---|---|
| **RESULT[9:2]** | [7:0] | rh | **Conversion Result**<br>This bit field contains the conversion result or the result of the data reduction filter. |

### Accumulated Read View RESRAx

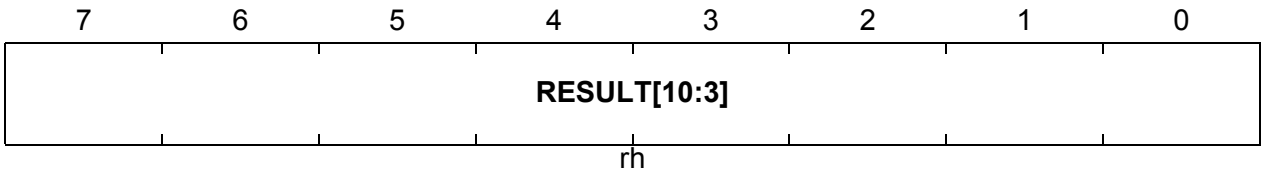
This view delivers the accumulated 9-bit or 11-bit conversion result and a 3-bit channel number. The corresponding valid flag is cleared when the high byte of the register is accessed by a read command, provided that bit RCRx.VFCTR is set.

**RESRAxL (x = 0 - 3)**
**Result Register x, View A Low** **($CA_H$ + x * 2)** **Reset Value: $00_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **CHNR** | [2:0] | rh | **Channel Number**<br>This bit field contains the channel number of the latest register update. |
| **DRC** | 3 | rh | **Data Reduction Counter**<br>This bit field indicates how many conversion results have still to be accumulated to generate the final result for data reduction.<br>$0_B$ The final result is available in the result register.The valid flag is automatically set when this bit field is set to 0.<br>$1_B$ 1 more conversion result must be added to obtain the final result in the result register.The valid flag is automatically reset when this bit field is set to 1. |
| **VF** | 4 | rh | **Valid Flag for Result Register x**<br>This bit indicates that the contents of the result register x are valid.<br>$0_B$ The result register x does not contain valid data.<br>$1_B$ The result register x contains valid data. |
| **RESULT[2:0]** | [7:5] | rh | **Conversion Result**<br>This bit field contains the conversion result or the result of the data reduction filter. |

**RESRAxH (x = 0 - 3)**
**Result Register x, View A High** **($CB_H$ + x * 2)** **Reset Value: 00H**




| Field | Bits | Type | Description |
|---|---|---|---|
| **RESULT[10:3]** | [7:0] | rh | **Conversion Result**<br>This bit field contains the conversion result or the result of the data reduction filter. |

Writing a 1 to a bit position in register VFCR clears the corresponding valid flag in registers RESRx/RESRAx. If a hardware event triggers the setting of a bit VFx and VFCx = 1, the bit VFx is cleared (software overrules hardware).

**VFCR**
**Valid Flag Clear Register** **($CE_H$)** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **0** | | | | **VFC3** | **VFC2** | **VFC1** | **VFC0** |
| r | | | | w | w | w | w |

| Field | Bits | Type | Description |
|---|---|---|---|
| **VFCx(x = 0 - 3)** | x | w | **Clear Valid Flag for Result Register x**<br>$0_B$ No action<br>$1_B$ Bit VFCx is reset. |
| **0** | [7:4] | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

The result control registers RCRx contain bits that control the behavior of the result registers and monitor their status.

**RCRx (x = 0 - 3)**
**Result Control Register x** **($CA_H$ + x * 1)** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **VFCTR** | **WFR** | **0** | **IEN** | | **0** | | **DRCTR** |
| rw | rw | r | rw | | r | | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **DRCTR** | 0 | rw | **Data Reduction Control**<br>This bit defines how many conversion results are accumulated for data reduction. It defines the reload value for bit DRC.<br>$0_B$ The data reduction filter is disabled. The reload value for DRC is 0, so the accumulation is done over 1 conversion.<br>$1_B$ The data reduction filter is enabled. The reload value for DRC is 1, so the accumulation is done over 2 conversions. |
| **IEN** | 4 | rw | **Interrupt Enable**<br>This bit enables the event interrupt related to the result register x. An event interrupt can be generated when DRC is set to 0 (after decrementing or by reload).<br>$0_B$ The event interrupt is disabled.<br>$1_B$ The event interrupt is enabled. |
| **WFR** | 6 | rw | **Wait-for-Read Mode**<br>This bit enables the wait-for-read mode for result register x.<br>$0_B$ The wait-for-read mode is disabled.<br>$1_B$ The wait-for-read mode is enabled. |
| **VFCTR** | 7 | rw | **Valid Flag Control**<br>This bit enables the reset of valid flag (by read access to high byte) for result register x.<br>$0_B$ VF unchanged by read access to RESRxH/RESRAxH. (default)<br>$1_B$ VF reset by read access to RESRxH/RESRAxH. |
| **0** | [3:1], 5 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

### 17.7.9 Interrupt Registers

Register CHINFR monitors the activated channel interrupt flags.

**CHINFR**
**Channel Interrupt Flag Register ($CA_H$) Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **CHINF7** | **CHINF6** | **CHINF5** | **CHINF4** | **CHINF3** | **CHINF2** | **CHINF1** | **CHINF0** |
| rh | rh | rh | rh | rh | rh | rh | rh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **CHINFx (x = 0 - 7)** | x | rh | **Interrupt Flag for Channel x**<br>This bit monitors the status of the channel interrupt x.<br>$0_B$ A channel interrupt for channel x has not occurred.<br>$1_B$ A channel interrupt for channel x has occurred. |

Writing a 1 to a bit position in register CHINCR clears the corresponding channel interrupt flag in register CHINFR. If a hardware event triggers the setting of a bit CHINFx and CHINCx = 1, the bit CHINFx is cleared (software overrules hardware).

**CHINCR**
**Channel Interrupt Clear Register ($CB_H$) Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **CHINC7** | **CHINC6** | **CHINC5** | **CHINC4** | **CHINC3** | **CHINC2** | **CHINC1** | **CHINC0** |
| w | w | w | w | w | w | w | w |

| Field | Bits | Type | Description |
|---|---|---|---|
| **CHINCx (x = 0 - 7)** | x | w | **Clear Interrupt Flag for Channel x**<br>$0_B$ No action<br>$1_B$ Bit CHINFR.x is reset. |

Writing a 1 to a bit position in register CHINSR sets the corresponding channel interrupt flag in register CHINFR and generates an interrupt pulse.

**CHINSR**
**Channel Interrupt Set Register ($CC_H$) Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **CHINS7** | **CHINS6** | **CHINS5** | **CHINS4** | **CHINS3** | **CHINS2** | **CHINS1** | **CHINS0** |
| w | w | w | w | w | w | w | w |

| Field | Bits | Type | Description |
|---|---|---|---|
| **CHINSx (x = 0 - 7)** | x | w | **Set Interrupt Flag for Channel x**<br>$0_B$ No action<br>$1_B$ Bit CHINFR.x is set and an interrupt pulse is generated. |

The bits in register CHINPR define the service request output line, SRx (x = 0 or 1), that is activated if a channel interrupt is generated.

**CHINPR**
**Channel Interrupt Node Pointer Register($CD_H$) Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **CHINP7** | **CHINP6** | **CHINP5** | **CHINP4** | **CHINP3** | **CHINP2** | **CHINP1** | **CHINP0** |
| rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **CHINPx (x = 0 - 7)** | x | rw | **Interrupt Node Pointer for Channel x**<br>This bit defines which SR lines becomes activated if the channel x interrupt is generated.<br>$0_B$ The line SR0 becomes activated.<br>$1_B$ The line SR1 becomes activated. |

Register EVINFR monitors the activated event interrupt flags.

**EVINFR**
**Event Interrupt Flag Register ($CE_H$) Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| EVINF7 | EVINF6 | EVINF5 | EVINF4 | 0 | | EVINF1 | EVINF0 |
| rh | rh | rh | rh | r | | rh | rh |

| Field | Bits | Type | Description |
|---|---|---|---|
| **EVINFx (x = 0 - 1, 4 - 7)** | [1:0], [7:4] | rh | **Interrupt Flag for Event x**<br>This bit monitors the status of the event interrupt x.<br>$0_B$ An event interrupt for event x has not occurred.<br>$1_B$ An event interrupt for event x has occurred. |
| **0** | [3:2] | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

Writing a 1 to a bit position in register EVINCR clears the corresponding event interrupt flag in register EVINFR. If a hardware event triggers the setting of a bit EVINFx and EVINCx = 1, the bit EVINFx is cleared (software overrules hardware).

**EVINCR**
**Event Interrupt Clear Flag Register ($CF_H$) Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| EVINC7 | EVINC6 | EVINC5 | EVINC4 | 0 | | EVINC1 | EVINC0 |
| w | w | w | w | r | | w | w |

| Field | Bits | Type | Description |
|---|---|---|---|
| **EVINCx (x = 0 - 1, 4 - 7)** | [1:0], [7:4] | w | **Clear Interrupt Flag for Event x**<br>$0_B$ No action<br>$1_B$ Bit EVINFR.x is reset. |
| **0** | [3:2] | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

Writing a 1 to a bit position in register EVINSR sets the corresponding event interrupt flag in register EVINFR and generates an interrupt pulse (if the interrupt is enabled).

**EVINSR**
**Event Interrupt Set Flag Register (D2$_H$) Reset Value: 00$_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| EVINS7 | EVINS6 | EVINS5 | EVINS4 | 0 | | EVINS1 | EVINS0 |
| w | w | w | w | r | | w | w |

| Field | Bits | Type | Description |
|---|---|---|---|
| **EVINSx (x = 0 - 1, 4 - 7)** | [1:0], [7:4] | w | **Set Interrupt Flag for Event x**<br>$0_B$ No action<br>$1_B$ Bit EVINFR.x is set. |
| **0** | [3:2] | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

The bits in register EVINPR define the service request output line, SRx (x = 0 or 1), that is activated if an event interrupt is generated.

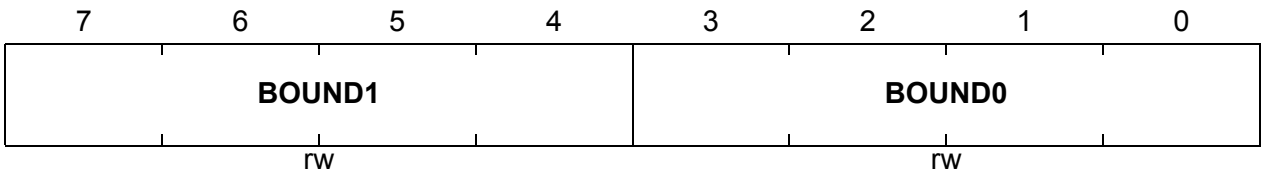
**EVINPR**
**Event Interrupt Node Pointer Register (D3$_H$) Reset Value: 00$_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| EVINP7 | EVINP6 | EVINP5 | EVINP4 | 0 | | EVINP1 | EVINP0 |
| rw | rw | rw | rw | r | | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **EVINPx (x = 0 - 1, 4 - 7)** | [1:0], [7:4] | rw | **Interrupt Node Pointer for Event x**<br>This bit defines which SR lines becomes activated if the event x interrupt is generated.<br>$0_B$ The line SR0 becomes activated.<br>$1_B$ The line SR1 becomes activated. |
| **0** | [3:2] | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

The bit fields in register LCBR define the four MSB of the compare values (boundaries) used by the limit checking unit. The values defined in bit fields BOUND0 and BOUND1 are concatenated with either four (8-bit conversion) or six (10-bit conversion) 0s at the end to form the final value used for comparison with the converted result. For example, the reset value of BOUND1 ($B_H$) will translate into $B0_H$ for an 8-bit comparison, and $2C0_H$ for a 10-bit comparison.

**LCBR**
**Limit Check Boundary Register ($CD_H$) Reset Value: $B7_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **BOUNDx (x = 0 - 1)** | [3:0], [7:4] | rw | **Boundary for Limit Checking** This bit field defines the four MSB of the compare value used by the limit checking unit. The result of the limit check is used for interrupt generation. |

# 18 On-Chip Debug Support

The On-Chip Debug Support (OCDS) provides the basic functionality required for software development and debugging of XC800-based systems.

The OCDS design is based on these principles:

- Use the built-in debug functionality of the XC800 Core
- Add a minimum of hardware overhead
- Provide support for most of the operations by a Monitor Program
- Use standard interface to communicate with the Host (a Debugger)

## 18.1 Features

The main debug features supported are:

- Set breakpoints on instruction address and on address range within the Program Memory
- Set breakpoints on Internal RAM address range
- Support unlimited software breakpoints in Flash/RAM code region
- Process external breaks via JTAG and upon activating a dedicated pin
- Step through the program code

## 18.2 Functional Description

The OCDS functional blocks are shown in **Figure 18-1**. The Monitor Mode Control (MMC) block at the center of OCDS system brings together control signals and supports the overall functionality. The MMC communicates with the XC800 Core, primarily via the Debug Interface, and also receives reset and clock signals.

After processing memory address and control signals from the core, the MMC provides proper access to the dedicated extra-memories: a Monitor ROM (holding the firmware code) and a Monitor RAM (for work-data and Monitor-stack).

The OCDS system is accessed through the JTAG[1)], which is an interface dedicated exclusively for testing and debugging activities and is not normally used in an application. The dedicated MBC pin is used for external configuration and debugging control.

1) The pins of the JTAG port can be assigned to either Port 0 (primary) or Ports 1 and 2 (secondary set one) or Port 5 (secondary set two).
User must set the JTAG pins (TCK and TDI) as input during connection with the OCDS system.

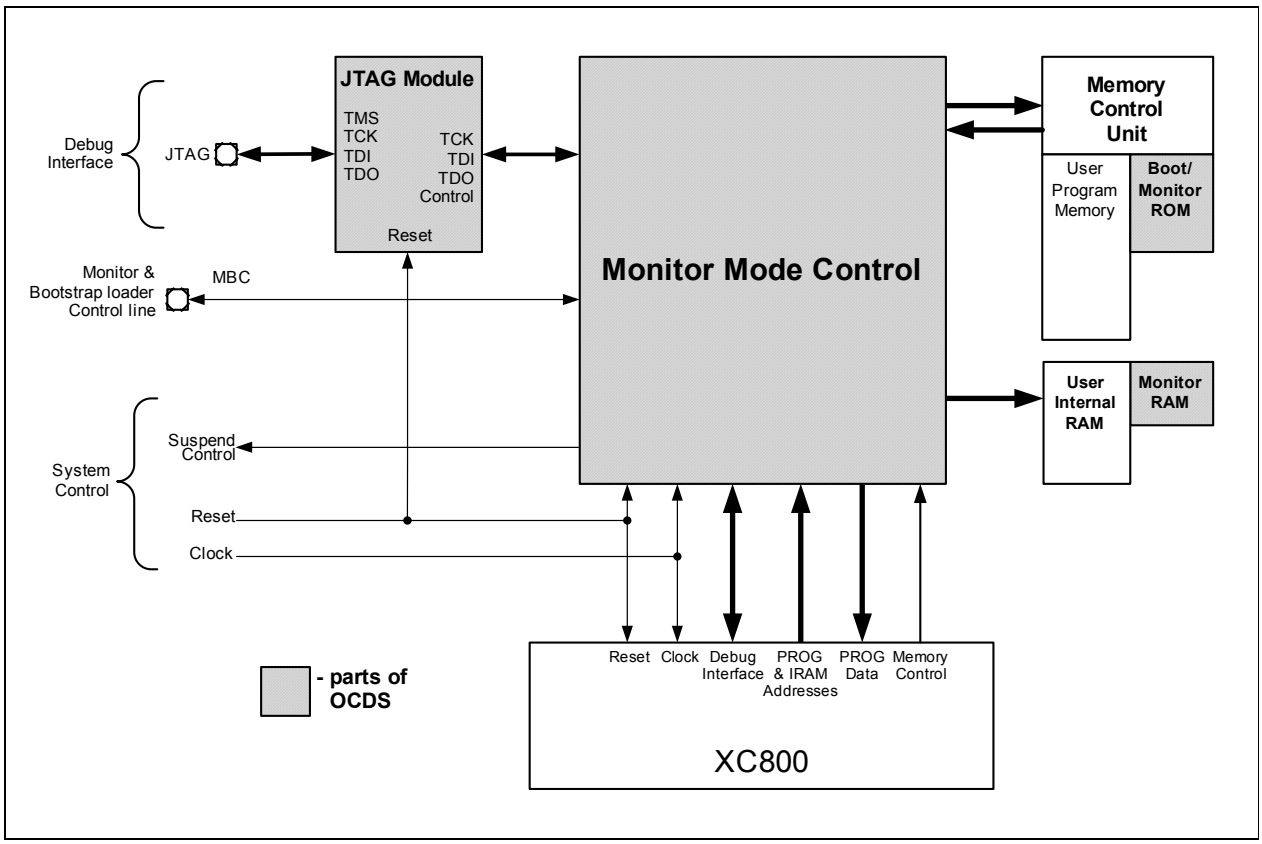



**Figure 18-1 XC878 OCDS: Block Diagram**

*Note: All the debug functionality described here can normally be used only after XC878 has been started in OCDS mode.*
*For more information on boot configuration options, see **Chapter 7.2.3**.*

***Attention: As long as the OCDS is actively used, the application software should not change the TRAP_EN bit within Extended Operation (EO) register!***

## 18.3 Debugging

The on-chip debug system functionality can be described in two parts. The first part covers the generation of Debug Events and the second part describes the Debug Actions that are taken when a debug event is generated.

- Debug events:
  - **Hardware Breakpoints**
  - **Software Breakpoints**
  - **External Breaks**
- Debug event actions:
  - **Call the Monitor Program**
  - **Activate the MBC pin**

The XC878 debug operation is based on close interaction between the OCDS hardware and a specialized software called the Monitor program.

### 18.3.1 Debug Events

The OCDS system recognizes a number of different debug events, which are also called breakpoints or simply breaks.

Depending on how the events are processed in time, they can be classified into three types of breaks:

- Break Before Make
  The break happens just before the break instruction (i.e. the instruction causing the break) is executed. Therefore, the break instruction itself will be the next instruction from the user program flow but executed only after the relevant debug action has been taken.
- Break After Make
  The break happens immediately after the instruction causing it has been executed. Therefore, the break instruction itself has already been executed when the relevant debug action is taken.
- Break Now
  The events of this type are asynchronous to the code execution inside the XC878 and there is no “instruction causing the debug event” in this case. The debug action is performed by OCDS “as soon as possible” once the debug event is raised.

#### 18.3.1.1 Hardware Breakpoints

Hardware breakpoints are generated by observing certain address buses within the XC878 system. The bus relevant to the hardware breakpoint type is continuously compared against certain registers where addresses for the breakpoints have been programmed.

The hardware breakpoints can be classified into different types:

- Depending on the address bus supervised
  - **Breakpoints on Instruction Address**
    Program Memory Address (PROGA) is observed
  - **Breakpoints on IRAM Address**
    Internal Data Memory Addresses for read/write (SOURCE_A, DESTIN_A) are observed
- Depending on the way comparison is done
  - Equal breakpoints
    Comparison is done only against one value; the break event is raised when just this value is matched.
  - Range breakpoints
    Comparison is done against two values; the break event is raised when a value

observed is found belonging to the range between two programmed values (inclusively).

### Breakpoints on Instruction Address

These Instruction Pointer (IP) breakpoints are generated when a break address is matched for the first byte of an instruction that is going to be executed i.e., for the address within Program Memory where an instruction opcode is fetched from.

*Note: In case of 2- and 3-byte instructions, the break will not be generated for addresses of the second and third instruction bytes.*

The IP breakpoints are of Break Before Make type, therefore the instruction at the breakpoint is executed only after the proper debug action is taken.

The OCDS in XC878 supports both equal breakpoints and range breakpoints on Instruction address (see **“Configurations of Hardware Breakpoints” on Page 18-4**).

### Breakpoints on IRAM Address

These breakpoints are generated when an instruction performs read or write access to a location within a defined address range from the Internal Data Memory (IRAM).

The IRAM breakpoints are of Break After Make type, therefore the proper debug action is taken immediately after the operation to the breakpoint address is performed.

The OCDS in XC878 supports only range breakpoints on IRAM address.

The OCDS differentiates between a breakpoint on read and a breakpoint on write operation to the IRAM.

### Configurations of Hardware Breakpoints

The OCDS allows setting of up to 4 hardware breakpoints. In XC878, the Program Memory address is 20-bit wide, while the Internal Data Memory address (both for Read and Write) is 8-bit wide. For setting of breakpoint on instruction address, BANKBPx defines the 4-bit bank and HWBPx defines the 16-bit address within the selected bank. For setting of breakpoint on IRAM address, HWBP2/3L and HWBP2/3H define the 8-bit IRAM address range.

The configurations supported are:

- Breakpoint 0
- Breakpoint 1
  - Two equal breakpoints on
    Instruction Address = BANKBP0 & HWBP0 and
    Instruction Address = BANKBP1 & HWBP1 or
  - One range breakpoint on
    BANKBP0 & HWBP0 <= Instruction Address <= BANKBP1 & HWBP1
- Breakpoint 2

  - One equal breakpoint on Instruction Address = BANKBP2 & HWBP2, or
  - One range breakpoint on HWBP2L <= IRAM Read Address <= HWBP2H
- Breakpoint 3
  - One equal breakpoint on Instruction Address = BANKBP3 & HWBP3, or
  - One range breakpoint on HWBP3L <= IRAM Write Address <= HWBP3H

Setting both values for a range breakpoint to the same address leads to generation of an equal breakpoint.

#### 18.3.1.2 Software Breakpoints

These breakpoints use the XC800-specific (not 8051-standard) TRAP instruction, decoded by the core while at the same time the TRAP_EN bit within the Extended Operation (EO) register is set to 1.

Upon fetching a TRAP instruction, a Break Before Make breakpoint is generated and the relevant Break Action is taken.

The software breakpoints are in fact similar in behavior to the equal breakpoints on Instruction address, except that they are raised by a program code instead of specialized (compare) logic.

An unlimited number of software breakpoints can be set by replacing the original instruction opcodes in the user program. However, this is possible only at addresses where a writable memory (RAM/Flash) is implemented.

*Note: In order to continue user program execution after the debug event, an external Debugger must restore the original opcode at the address of the current software breakpoint.*

#### 18.3.1.3 External Breaks

These debug events are of Break Now type and can be raised in two ways:

- By a request via the JTAG interface - using a special sequence, an external device connected to the JTAG can break the user program running on XC878 and start a debug session;
- By asserting low the dedicated Monitor and BootStrap loader Control line (MBC) while the XC878 is running and this type of break is enabled - used for reaction to asynchronous events from the external world.

#### 18.3.1.4 NMI-mode priority over Debug-mode

While the core is in NMI-mode (after an NMI-request has been accepted and before the RETI instruction is executed, i.e. the time during a NMI-servicing routine), certain debug functions are blocked/restricted:

1. No external break is possible while the core is servicing an NMI.
   External break requested inside a NMI-servicing routine will be taken only after RETI is executed.
2. A breakpoint into NMI-servicing routine is taken, but single-step is not possible afterwards.
   If a step is requested, the servicing routine will run as coded and monitor mode will be invoked again only after a RETI is executed.

Hardware breakpoints and software breakpoints proceed as normal while CPU is in NMI-mode.

### 18.3.2 Debug Actions

In case of a debug event, the OCDS system can respond in two ways depending on the current configuration.

#### 18.3.2.1 Call the Monitor Program

XC878 comes with an on-chip Monitor program, factory-stored into the non-volatile Monitor ROM (see **Figure 18-1**). Activating this program is the primary and basic OCDS reaction to recognized debug events.

The OCDS hardware ensures that the Monitor is always safely started, and fully independent of the current system status at the moment when the debug action is taken. Also, interrupt requests optionally raised during Monitor-entry will not disturb the firmware functioning.

Once started, the Monitor runs with own stack- and data- memory (see Monitor RAM in **Figure 18-1**), which guarantees that all of the core and memory resources will be found untouched when returning control back to the user program. Therefore the OCDS-debugging in XC878 is fully non-destructive.

The functions of the XC878 Monitor include:

- Communication with an external Debugger via the JTAG interface
- Read/write access to arbitrary memory locations and Special Function Registers (SFRs), including the Instruction Pointer and password-protected bits
- Configuring OCDS and setting/removing breakpoints
- Executing a single instruction (step-mode)

*Note: Detailed descriptions of the Monitor program functionality and the JTAG communication protocol are not provided in this document.*

#### 18.3.2.2 Activate the MBC pin

The MBC pin can be driven actively low in reaction to debug events, if respective settings have been done in OCDS.

This functionality allows two alternative configurations:

- As an action additional to the Monitor program start - in such a case MBC pin is activated for up to 77 system clock (SCLK) cycles;
- As the only OCDS action while temporarily suspending the core activity - MBC pin is driven low for 4 SCLK cycles only as a fastest reaction to the program flow (breakpoint match).

## 18.4 Debug Suspend Control

Next to the basic debug functionality - setting breakpoints and halting the execution of user software - XC878 OCDS supports also an additional feature: module suspend during debugging.

As long as the device is in monitor mode (i.e. while the user software is not running but in break) and if debug suspend functionality is generally enabled by on-chip software (Monitor or Bootcode) OCDS activates a signal to a number of counter modules, namely:

- Watchdog Timer (WDT)
- Timer 2
- Timer 12 and Timer 13 in Capture/Compare Unit 6 (CCU6)

The Module Suspend Control Register (MODSUSP) holds control bits for these timers. When some control bit is set - the respective timer will be stopped while the monitor mode is active.

This feature could be quite useful, especially regarding the Watchdog Timer: it allows to prevent XC878 from unintentional WDT-resets while the user software is not executed and respectively - not able to service the Watchdog.

Also suspending the other timer-modules makes sense for debugging: once the application is not running, stopping counters helps for a more complete “freeze” of the device-status during a break.

It must be noted, in XC878 all of the debug suspend control bits (global enable in OCDS and individual selections in SCU) have values 0 after reset, i.e. by default no module will be suspended upon a break. But normally, for debugging the device will be started in OCDS mode and then the monitor will be invoked before to start any user code. Then it is possible using a debugger to configure suspend-controls as desired and only afterwards start the debug-session.

*Note: For more information on debug-suspend, refer to the individual modules’ section on Module Suspend Control.*

## 18.5 Register Description

From a programmer's point of view, OCDS is represented in C868-1R, C868-1S by a total of 10 register-addresses (see **Table 18-1**), all located within the mapped SFR area.

**Table 18-1 OCDS Directly Addressable Registers**

| Register Short Name | Address (mapped) | Register Full Name |
|---|---|---|
| MMCR | $F1_H$ | Monitor Mode Control Register |
| MMCR2 | $E9_H$ | Monitor Mode Control Register 2 |
| MMSR | $F2_H$ | Monitor Mode Status Register |
| MMBPCR | $F3_H$ | Monitor Mode Breakpoints Control Register |
| MMICR | $F4_H$ | Monitor Mode Interrupt Control Register |
| MMDR | $F5_H$ | Monitor Mode Data Register |
| HWBPSR | $F6_H$ | Hardware Breakpoints Select Register |
| HWBPDR | $F7_H$ | Hardware Breakpoints Data Register |
| MEXTCR | $EA_H$ | Memory Extension control Register |
| MMWR1 | $EB_H$ | Monitor Work Register 1 |
| MMWR2 | $EC_H$ | Monitor Work Register 2 |

Additionally, there are overall 12 indirectly accessible OCDS registers, in two groups:

- 8 Hardware Breakpoint registers, accessible via HWBPSR (Register Select) and HWBPDR (Data)

**Table 18-2 Hardware Breakpoint Registers (8/16-bit Addresses)**

| Register Short Name | Register Full Name |
|---|---|
| HWBP0L | Hardware Breakpoint 0 Low Register |
| HWBP0H | Hardware Breakpoint 0 High Register |
| HWBP1L | Hardware Breakpoint 1 Low Register |
| HWBP1H | Hardware Breakpoint 1 High Register |
| HWBP2L | Hardware Breakpoint 2 Low Register |
| HWBP2H | Hardware Breakpoint 2 High Register |

**Table 18-2 Hardware Breakpoint Registers (8/16-bit Addresses)** (cont'd)

| Register Short Name | Register Full Name |
|---|---|
| HWBP3L | Hardware Breakpoint 3 Low Register |
| HWBP3H | Hardware Breakpoint 3 High Register |

- 4 Hardware Breakpoint Bank registers,accessible via HWBPSR (Register Select) and MEXTCR (Data)

**Table 18-3 Hardware Breakpoint Bank Registers**

| Register Short Name | Register Full Name |
|---|---|
| BANKBP0 | Hardware Breakpoint 0 Bank Register |
| BANKBP1 | Hardware Breakpoint 1 Bank Register |
| BANKBP2 | Hardware Breakpoint 2 Bank Register |
| BANKBP3 | Hardware Breakpoint 3 Bank Register |

The OCDS registers are exclusively dedicated to the on-chip Monitor program and the user should not write into them. Anyway a big part of these registers or separate bits/fields are protected and can not be written by user software but only by the firmware in two modes of C868-1R, C868-1S:

- Startup mode - while the Bootcode is executed after reset, the user code is still not started
- Monitor mode - while the Monitor program is running, the user code is in break.

Therefore an unintentional access to OCDS registers by the user software can not disturb the normal debug functionality.

### 18.5.1 Monitor Work Register 2

Only one register - **MMWR2** - can be used for general purposes when no debug-session is possible: if the XC878 is not started in OCDS mode and no external device is connected to the JTAG interface.

**MMWR2**
**Monitor Work Register 2** **mapped SFR ($EC_H$)** **Reset value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| MMWR2 | | | | | | | |
| rw | | | | | | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **MMWR2** | 7:0 | rw | **Work Register 2**<br>Work location 2 for the Monitor Program. |

## 18.5.2 Input Select Registers

Bits MODPISEL.JTAGTCKS is used to select one of the two TCK inputs while bits MODPISEL.JTAGTDIS is used to select one of the two TDI inputs.

**MODPISEL**
**Peripheral Input Select Register** **Reset Value: $00_H$**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **0** | **URRISH** | **JTAGTDIS** | **JTAGTCKS** | **EXINT2IS** | **EXINT1IS** | **EXINT0IS** | **URRIS** |
| r | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **JTAGTCKS** | 4 | rw | **JTAG TCK Input Select**<br>0 JTAG TCK Input TCK_0 is selected.<br>1 JTAG TCK Input TCK_1 is selected. |
| **JTAGTDIS** | 5 | rw | **JTAG TDI Input Select**<br>0 JTAG TDI Input TDI_0 is selected.<br>1 JTAG TDI Input TDI_1 is selected. |
| **0** | 7 | r | **Reserved**<br>Returns 0 if read; should be written with 0. |

## 18.6 JTAG ID

This is a read-only register located inside the JTAG module, and is used to recognize the device(s) connected to the JTAG interface. Its content is shifted out when INSTRUCTION register contains the IDCODE command (opcode $04_H$), and the same is also true immediately after reset.

The JTAG ID for the XC878 devices is given in **Table 18-4**.

**Table 18-4 JTAG ID Summary**

| Device Type | Device Name | JTAG ID |
|---|---|---|
| Flash | XC878*-13FF | $1018\ 3083_H$ |
| | XC878*-16FF | $1018\ 2083_H$ |

*Note: The asterisk (*) above denotes the device configuration letters from **Table 1-1** . **Table 1-2** shows the device profiles available for XC878.*

# 19 Bootstrap Loader

The XC878 includes a Bootstrap Loader (BSL) Mode that can be entered with the pin configuration during hardware reset, shown in **Table 19-1**. The main purpose of BSL Mode is to allow easy and quick programming/erasing of the Flash and XRAM via serial interface. The XC878 supports four device BSL modes:

- UART BSL
- LIN BSL
- MultiCAN BSL
- Alternate BSL

If a device is programmed as UART/MultiCAN, the entry to the respective BSL (UART or MultiCAN) is decided based on their initial header frames.

*Note: UART BSL is supported only via UART module and not UART1.*

*Note: For BSL modes, only the default set of receive/transmit pins of UART and MultiCAN node 0 (P1.0/P1.1) can be used.*

**Table 19-1 Pin Configuration to Enter BSL Mode**

| MBC[1)] | TMS[1)] | MODE / Comment |
|---|---|---|
| 0 | 0 | BSL Mode (LIN Mode[2)], UART/ MultiCAN Mode[3)4)] and Alternate BSL Mode[5)]); on-chip OSC/PLL non-bypassed |

1) Latched pin values

2) If a device is programmed as LIN, LIN BSL is always used instead of UART/MultiCAN.

3) UART or MultiCAN BSL is decoded by firmware based on the protocol for product variant with MultiCAN. If no MultiCAN and LIN variant, UART BSL is used.

4) In MultiCAN BSL mode, the clock source is switched to XTAL by firmware, bypassing the on-chip oscillator. This avoids any frequency invariance with the on-chip oscillator and allows other frequency clock input, thus ensuring accurate baud rate detection (especially at high bit rates).

5) Alternate BSL Mode is a user defined BSL mode. The user BSL code is located in Flash memory. It is entered if the AltBSLPassword is valid. For an invalid password, LIN or UART/MultiCAN BSL will be entered based on the product variants.

**Section 19.1** describes the UART and LIN BSL modes, **Section 19.2** describes the MultiCAN BSL mode and Alternate BSL mode is described in **Section 19.3**.

## 19.1 UART and LIN BSL Modes

The UART and LIN BSL Modes have three functional parts represented by the three phases described below:

- **Phase I**: Establish a serial connection and automatically synchronize to the transfer speed (baud rate) of the serial communication partner (host).
- **Phase II**: Perform serial communication with the host. The host controls and sends a special header information which selects one of the modes, described in **Table 19-2**.
- **Phase III**: Response to host to indicate successful/failure transfer. See **Section 19.1.1.3**.

**Table 19-2 Serial Communication Modes of the UART and LIN BSL Modes**

| Mode | Description |
|---|---|
| **0** ($00_H$) | Transfer a user program from the host to XRAM[1)] |
| **1** ($01_H$) | Execute a user program in the XRAM at start address[2)] |
| **2** ($02_H$) | Transfer a user program from the host to P-Flash[1)] |
| **3** ($03_H$) | Execute a user program in the P-Flash[2)] |
| **6** ($06_H$) | Flash Protection Mode enabling/disabling scheme[1)] |
| **8** ($08_H$) | Transfer a user program from the host to XRAM[1)3)] |
| **9** ($09_H$) | Execute a user program in the XRAM[2)3)] |
| **10** ($0A_H$) | Get 4-byte chip information |
| **15** ($0F_H$) | Enter OCDS UART Mode[2)] |
| **16** ($10_H$) | Erase/Mass erase P-Flash or D-Flash Memory[1)] |

1) The microcontroller would return to the beginning of Phase I/II and wait for the next command from the host

2) BSL Mode is exited and the serial communication is not established.

3) Mode 8 and Mode 9 are supported in BSL Mode via LIN only. It is the similar to Mode 0 and Mode 1.

Basic serial communication protocol such as transfer block structure and the various response code to host for both BSL Mode via UART and LIN are described in **Section 19.1.1** while implementation details of BSL Mode via both UART and LIN protocols will be covered in **Section 19.1.2** and **Section 19.1.3** respectively.

### 19.1.1 Communication Protocol

Once baud rate is established, the host sends a block of information to the microcontroller to select the desired mode. All blocks follow the specified block structure as shown in **Section 19.1.1.1** for UART and **Section 19.1.1.2** for LIN. The microcontroller respond to host by sending specific response code as shown in **Section 19.1.1.3**.

#### 19.1.1.1 UART Transfer Block Structure

A UART transfer block consists of three parts:

| **Block Type** (1 byte) | **Data Area** (XX bytes) | **Checksum** (1 byte) |
|---|---|---|

- **Block Type**: the type of block, which determines how the data area is interpreted. Implemented block types are:

  **$00_H$** type "**HEADER**"

  Header Block has a fixed length of 8 bytes. Special information is contained in the data area of the Header Block, which is used to select different modes.

  **$01_H$** type "**DATA**"

  Data Block is used in Mode 0 and Mode 2 to transfer a portion of program code. The program code is in the data area of the Data Block.[1)]

  **$02_H$** type "**END OF TRANSMISSION" (EOT)**

  EOT Block is the last block in data transmission in Mode 0 and Mode 2. The last program code to be transferred is in the data area of the EOT Block.[1)]

- **Data Area**: Data size is 6 bytes for Header Block and cannot exceed 96 bytes for both Data and EOT Blocks.
- **Checksum**: the XOR checksum of the block type and data area sent by the host. BSL routine calculates the checksum of the received bytes (block type and data area) and compares it with received checksum.

1) The length of Data and EOT Blocks is defined as Block_Length in the Header Block.

### 19.1.1.2 LIN Transfer Block Structure

A LIN transfer block, 9 bytes long (fixed), consists of four parts:

| NAD (1 byte) | Block Type (1 byte) | Data Area (6 bytes) | Checksum (1 byte) |
|---|---|---|---|

- **NAD**: Node Address for Diagnostic, which specifies the address of the active slave node

  **$01_H$ to $7E_H$** Valid Slave Address

  **$80_H$ to $FF_H$** Valid Slave Address

  **$7F_H$** Broadcast Address (For Master nodes to all Slave nodes)

  **$00_H$** Invalid Slave Address (Reserved for go-to-sleep-command)
- **Block Type**: The type of block, which determines how the data area is interpreted. See **Section 19.1.1.1**.

  **$00_H$** “HEADER” type

  **$01_H$** “DATA” type

  **$02_H$** “END OF TRANSMISSION” (EOT) type
- **Data Area**: Fixed size of 6 bytes which represent the data of the block. For Header Block, one byte will indicate the Mode selected and 5 bytes for Mode data. For Data and EOT Blocks, data area consists of the program code.
- **Checksum**: The Programming Checksum or LIN Checksum contains the non-inverted or inverted eight bit sum with carry[1)] over NAD, Block Type and Data Area.

1) Eight bit sum with carry equivalent to sum all values and subtract 255 every time the sum is greater or equal to 256 (which is not the same as modulo-255 or modulo-256).

Diagnostic LIN frame always uses classic checksum where checksum calculation is over the data bytes only. It is used for communication with LIN 1.3 slaves. The Classic Checksum contains the inverted eight bit sum with carry over all data bytes.

A non-LIN standard checksum, also known as Programming Checksum, is implemented to differentiate an XC878 Programming LIN frame from a normal LIN frame and to allow other slaves (non-Programming), which are on the LIN bus to ignore this Programming frame. XC878 supports both the LIN Classic Checksum and Programming Checksum where Programming Checksum contains the eight bit sum with carry over all 8 data bytes.

An illustration on the Programming Checksum and LIN Checksum calculation is provided in **Table 19-3** for data of $4A_H$, $55_H$, $93_H$ and $E5_H$.

**Table 19-3 LIN Frame - Programming Checksum**

| Addition of data | HEX | Result | CARRY | Addition with CARRY |
|---|---|---|---|---|
| **$4A_H$** | $4A_H$ | $4A_H$ | 0 | $4A_H$ |
| ($4A_H$) + **$55_H$** | $9F_H$ | $9F_H$ | 0 | $9F_H$ |
| ($9F_H$) + **$93_H$** | $0132_H$ | $32_H$ | 1 | $33_H$ |
| ($33_H$) + **$E5_H$** | $0118_H$ | $18_H$ | 1 | **$19_H$** |

The Programming Checksum is $19_H$. An inversion of the Programming Checksum yields the standard LIN Checksum (Classic Checksum (i.e., $E6_H$)).

Both Programming and LIN Checksum are supported and indicated in respective modes.

### 19.1.1.3 Response Code to the Host

The microcontroller would let the host know whether a block has been successfully received by sending out a response code.

Table 19-4 tabulates the possible responses from the microcontroller upon reception of a Header, Data or EOT block for each working mode.

**Table 19-4 Possible Responses for Various Block Types**

| Mode | Header Block | Data Block | EOT Block |
|---|---|---|---|
| 0, 8 | Acknowledge, Block Error, Checksum Error, Protection Error | Acknowledge, Block Error, Checksum Error | Acknowledge, Block Error, Checksum Error |
| 1, 9 | Acknowledge, Block Error, Checksum Error | - | - |
| 2 | Acknowledge, Block Error, Checksum Error, Protection Error | Acknowledge, Block Error, Checksum Error | Acknowledge, Block Error, Checksum Error |
| 3 | Acknowledge, Block Error, Checksum Error | - | - |
| 6 | Acknowledge, Block Error, Checksum Error, Protection Error | - | - |
| 10 | Acknowledge, Block Error, Checksum Error | - | - |
| 15 | Acknowledge, Block Error, Checksum Error | - | - |
| 16 | Acknowledge, Block Error, Checksum Error, Protection Error | - | - |

If a block is received correctly, an Acknowledge Code ($55_H$) is sent. In case of failure, it may be a wrong block type error or checksum error. Block type error is caused by two conditions; (i) The microcontroller receives a block type other than the implemented ones; (ii) The microcontroller receives the transfer blocks in wrong sequence. In both error cases, the BSL routine awaits the actual block from the host again.

When program and erase operations of Flash are restricted due to Flash Protection Mode 0 or 1 being enabled, protection error code will be sent to the host. This will indicate that Flash is protected, and hence, it cannot be programmed or erased. In this error case, the BSL routine will wait for the next header block from the host again.

Table 19-5 lists the responses with the possible reasons and/or implcations for error and suggests the possible corrective actions that the host can take upon notification of the error.

**Table 19-5 Definition of Responses**

| Response | Value | Description | | | |
|---|---|---|---|---|---|
| | | Block Type | BSL Mode | Reasons / Implications | Corrective Action |
| Acknow-ledge | $55_H$ | Header | 1, 3, 9, 15 | The requested operation will be performed once the response is sent. | - |
| | | | 6, 16 | The requested operation has been performed and is successful. | |
| | | EOT | 0, 2, 8 | | |
| | | All others | | Reception of the block is successful. Transmission of 4-byte data follows in Mode 10. Ready to receive the next block. | |
| Block Error | $FF_H$ | Header | 2 | P-Flash start address is odd. | P-Flash address should be even. |
| | | | | Exceed the length of the block. | Reduce the length of the block to be within limits. |
| | | Data, EOT | | The length of P-Flash code is odd. | Length of P-Flash code should be even. |
| | | All others | | Either the block type is undefined or the communication structure is invalid. | Retransmit a valid block. |
| Checksum Error | $FE_H$ | All | | Mismatch exists between the calculated and received Checksum. | Retransmit the block |

**Table 19-5 Definition of Responses**

| Response | Value | Description | | | |
|---|---|---|---|---|---|
| | | Block Type | BSL Mode | Reasons / Implications | Corrective Action |
| Protection Error | $FD_H$ | Header | 0, 2, 8, 16 | Protection against external access is enabled, i.e. FPASSWD is valid. | - |
| | | | 6 | Fail to disable the protection, i.e. FPASSWD does not match user password. | - |

### 19.1.2 Bootstrap Loader via UART

Upon entering UART BSL, a serial connection is established and the transfer speed (baud rate) of the serial communication partner (host) is automatically synchronized in the following steps:

- STEP 1: Initialize serial interface for reception and timer for baud rate measurement
- STEP 2: Wait for test byte ($80_H$) from host
- STEP 3: Synchronize the baud rate to the host
- STEP 4: Send Acknowledge byte ($55_H$) to the host
- STEP 5: Enter Phase II

Baud rate is established once in the beginning of UART BSL. Until next hardware reset, subsequent communication between host and the microcontroller will follow this baud rate.

The serial port of the microcontroller is set to Mode 1 (8-bit UART, variable baud rate), while Timer 2 is configured to auto-reload mode (16-bit timer) for baud rate measurement. The PC host sends test byte ($80_H$) to start the synchronization flow. The timer is started on reception of the start bit (0) and stopped on reception of the last bit of the test byte (1). Then the UART BSL routine calculates the actual baud rate, sets the PRE and BG values and activates Baud Rate Generator. When the synchronization is done, the microcontroller sends back the Acknowledge byte ($55_H$) to the host. The baud rate supported ranges from 4800 Baud to 38400 Baud.

If the synchronization fails, the Acknowledge code from the microcontroller cannot be received correctly by the host. In this case, on the host side, the host software may display a message to the user, e.g., requesting the user to repeat the synchronization procedure, see **Section 19.1.1.3** for Response code.

On the microcontroller side, the UART BSL routine cannot determine whether the synchronization is correct or not. It always enters Phase II after sending the acknowledge byte. Therefore, if synchronization fails, a reset of the microcontroller has to be invoked, to restart the microcontroller for a new synchronization attempt.

### 19.1.2.1 Communication Structure

There are two types of transfer flow of the Header Block, Data Block, EOT Block, and the Response Code, as shown in **Figure 19-1**. One is adopted by Mode 0 and Mode 2, while the other is adopted by the rest of the modes. Data and EOT Blocks are transferred only in Mode 0 and 2.

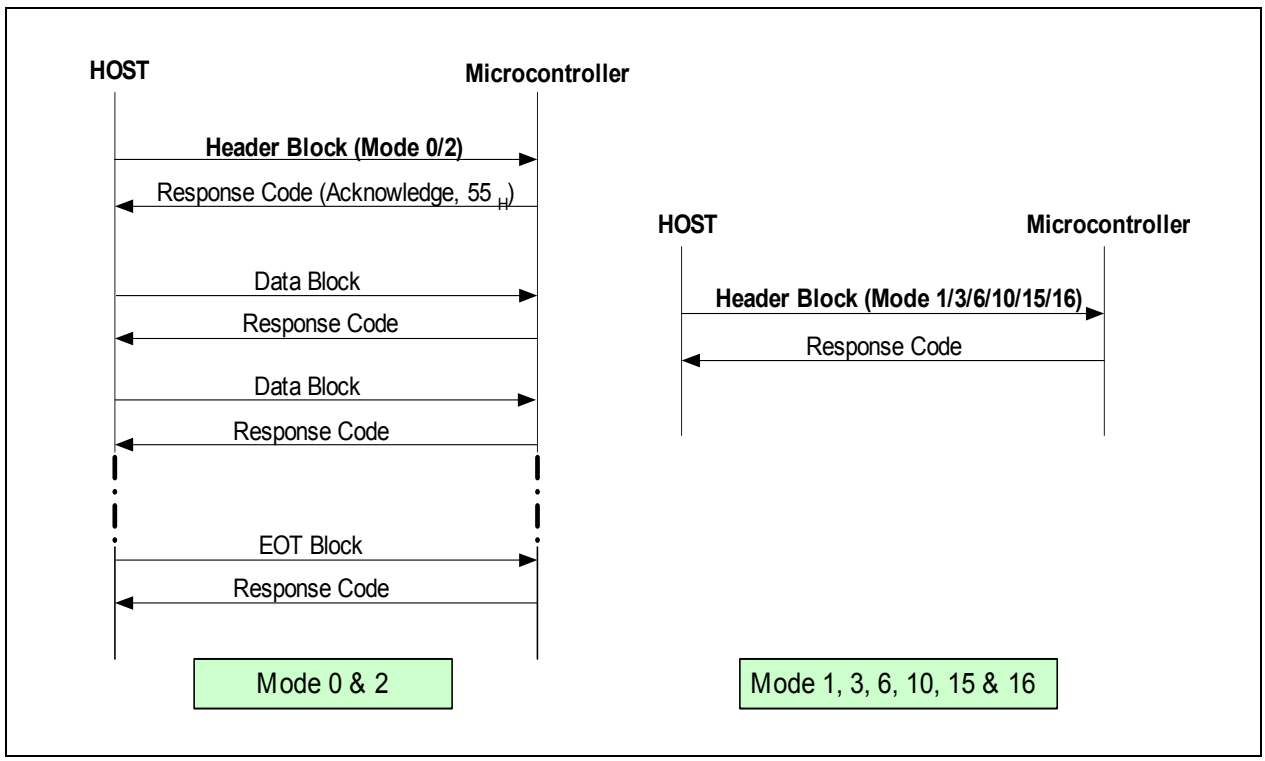



**Figure 19-1 Communication Structure of the UART BSL Modes**

### 19.1.2.2 The Selection of Modes

When UART BSL routine enters Phase II, it first awaits for an 8-byte Header Block, from the host which contains the information for the selection of the modes, as shown below.

| **Block Type** $00_H$ (Header Block) | **Data Area** | | **Checksum** (1 byte) |
|---|---|---|---|
| | **Mode** (1 byte) | **Mode Data** (5 bytes) | |

Description:

- $00_H$: The block type, which marks the block as a **Header Block**
- **Mode**: The mode to be selected. Mode 0 - 6 are supported. See **Table 19-2**
- **Mode Data**: Five bytes of special information to activate corresponding mode.
- **Checksum**: The checksum of the header block. XOR of all 7 bytes.

### 19.1.2.3 The Activation of Modes 0 and 2

Mode 0 and Mode 2[1] are used to transfer a user program from the host to the XRAM and Flash of the microcontroller respectively. The header block has the following structure:

**The Header Block**

| $00_H$ (Header Block) | $00_H/02_H$ (Mode 0/2) | Mode Data | | | | Checksum |
|---|---|---|---|---|---|---|
| | | **StartAddr High** (1 byte) | **StartAddr Low** (1 byte) | **Block_ Length** (1 byte) | **Not Used** (2 bytes) | |

Mode Data Description:

**Start Addr High, Low**: 16-bit Start Address of the 20-bit Start Address[2], which determines where to copy the received program code in the XRAM/Flash[3]

**Block_Length**: The whole length (block type, data area and checksum) of the following Data or EOT Blocks.[4][5]

**Not used**: 2 bytes, these bytes are not used and will be ignored in Mode 0/2.

After the header block is successfully received, the microcontroller enters Mode 0/2, during which the program code is transmitted from the host to the microcontroller by Data Block and EOT Block, which are described below.

**The Data Block**

| **$01_H$ (Data Block)** (1 byte) | **Program Code** ((**Block_Length** - 2) bytes) | **Checksum** (1 byte) |
|---|---|---|

Description:

**Program Code**: The program code has a length of (**Block_Length**-2) byte, where the Block_Length is provided in the previous Header Block.

1) Contents in the XRAM could be changed when Mode 2 is activated.
2) The upper 4 bits, or the bank, is dynamically assigned by the device based on the Flash device variant.
3) During each programming cycle, no crossing of a wordline boundary is allowed. For P-Flash programming, the Start Address must a valid address in even number.
4) When the Block_Length is defined in Header Block, the subsequent Data or EOT Block must be of this length. To redefine the Block_Length, it must be accompanied by a new Header Block.
5) The Block_Length is 3 - 128 for Mode 0. The Block_Length is 3 - 34 bytes for Mode 2 if D-Flash is targeted. For P-Flash, the Block_Length is always 4 - 66 bytes.

*Note: No empty Data Block is allowed.*

**The EOT Block**

| $02_H$ (EOT Block) (1 byte) | Last_Codelength (1 byte) | Program Code | Not Used | Checksum (1 byte) |
|---|---|---|---|---|

Description:

**Last_Codelength**: This byte indicates the length of the program code in this EOT Block.

**Program Code**: The last program code to be sent to the microcontroller

**Not used**: The length is (**Block_Length**-3-**Last_Codelength**). These bytes are not used and they can be set to any value.

### 19.1.2.4 The Activation of Modes 1, 3 and 15

Mode 1 is used to execute a user program in the XRAM of the microcontroller at 0'$F000_H$ and 2'$F000_H$ depending on the Flash variant. As for Mode 3, it used to execute a user program in the Flash of the microcontroller at 0'$0000_H$. Mode F is used to enter OCDS UART Mode. The header block has the following structure:

**The Header Block**

| $00_H$ (Header Block) | $01_H/03_H/0F_H$ (Mode 1/3/15) | Mode Data | Checksum (1 byte) |
|---|---|---|---|
| | | Not Used (5 Bytes) | |

Mode Data Description:

**Not used**: The five bytes are not used and will be ignored in Mode 1/3/15.

For Modes 1, 3 and 15, the header block is the only transfer block to be sent by the host, no further serial communication is necessary. The microcontroller will then exit the BSL Mode and jump to the XRAM address (Mode 1), jump to Flash address at 0'$0000_H$ (Mode 3) and/or start to communicate with the OCDS UART debugger (Mode 15).

### 19.1.2.5 The Activation of Mode 6

Mode 6[1)] is used to enable or disable Flash protection via the given user-password. The header block for this mode has the following structure:

**The Header Block**

| | | | | |
|---|---|---|---|---|
| **$00_H$** (Header Block) | **$06_H$** (Mode 6) | **Mode Data** (5 bytes) | | **Checksum** |
| | | **User-Password** (2 byte) | **Not Used** (3 bytes) | |

Mode Data Description:

**User-Password**: This byte is given by user to enable or disable Flash protection. For a description of the user-password, see **Chapter 3.3.1**. Transferring of the 16-bits user-password follows network byte order which means that the upper byte is the first to be transferred.

**Not used**: The four bytes are not used and will be ignored in Mode 6.

In Mode 6, the header block is the only transfer block to be sent by the host. This mode is used when user wants to (i) enable Flash protection; (ii) disable Flash protection.

When Flash is not protected yet, the microcontroller will enable the various Flash protection scheme based on the 3 MSB of the user-password. The selected Flash protection mode will be activated once the Header block is received and microcontroller identifies this user-password as the program-password for future operations.

When Flash is already protected, the microcontroller will deactivate all Flash Protection if the user-password byte matches the program-password. **Protected P-Flash Block will be erased and D-Flash block will depend on bit 12 of the user-password**. After that, the program-password is reset. For the case of Flash hardware protection as described in **Chapter 3.3.1**, it will only be deactivated at the next power-up or hardware reset.

### 19.1.2.6 The Activation of Mode 10

Mode 10 ($0A_H$) is used to obtain a 4-byte data. The contents of the 4-byte data is determined by the Option byte in the header block. The header block for this mode has the following structure:

1) Contents in the XRAM could be changed when Mode 6 is activated.

**The Header Block**

| $00_H$ (Header Block) | $0A_H$ (Mode 10) | **Mode Data** (5 bytes) | | **Checksum** |
|---|---|---|---|---|
| | | **Not Used** (4 bytes) | **Option** (1 byte) | |

Mode Data Description:

**Option**: This byte will determine the 4 bytes data to be sent to the host. Only option $00_H$ is available to return the chip identification number, which is used to identify the particular device variant.

$00_H$ - Chip Identification Number (MSB byte 1... LSB byte 4)

In Mode 10, the header block is the only transfer block to be sent by the host. The microcontroller will return an acknowledgement followed by 4 bytes of data to the host if the header block is received successfully. If an invalid option is received, the microcontroller will return 4 bytes of $00_H$.

### 19.1.2.7 The Activation of Mode 16

Mode 16 ($10_H$) is used to erase a page in P-Flash or D-Flash, or mass erase all memories in P-Flash block and D-Flash block. If code protection is enabled, this mode will be unavaliable and Protection Error is returned. The selection of the type of erase is controlled through the Erase/MassErase byte in the Mode Data block.

**The Header Block**

| $00_H$ (Header Block) | $10_H$ (Mode 16) | **Mode Data** (5 bytes) | | | | **Checksum** |
|---|---|---|---|---|---|---|
| | | **Start Addr High (1 byte)** | **Start Addr Low (1 byte)** | **Erase/ MassErase (1 byte)** | **Not Used (2 bytes)** | |

Mode Data Description:

**Start Addr High, Low**: Lower 16-bit of any 20-bit Start Address[1)] within the targeted page or block.

**Erase / Mass Erase**: 0 for (Page) Erase or 1 for Mass Erase.

**Not used**: 2 bytes, these bytes are not used and will be ignored in Mode 16.

1) The upper 4 bits, or the bank, is dynamically assigned by the device based on the Flash device variant

### 19.1.3 Bootstrap Loader via LIN

Standard LIN protocol can support a maximum baud rate of 20 kHz. However, the XC878L device has an enhanced feature which supports a baud rate of up to 57.6 kHz. LIN BSL is implemented to support the baud rate of 20 kHz and below using standard LIN protocol, while Fast LIN BSL is introduced to support the baud rate of 20 kHz to 57.6 kHz via a single-wire UART using UART protocol. See **Section 19.1.3.9**.

LIN BSL supports Fast Programming through Mode 0, Mode 2 or Mode 8 with the selection of Fast Programming Option. Refer to **Section 19.1.3.3** for more details.

Features of LIN BSL are:

- Re-synchronization of the transfer speed (baud rate) of the communication partner upon receiving every LIN frame
- Use of Diagnostic Frame (Master Request and Slave Response)
- User-preloaded NAD stored in uppermost P-Flash Block. (Default Broadcast NAD used if value not present or valid)
- Save LIN frame into XRAM and jump to User Mode if first frame received is an invalid LIN Frame
- Programming and LIN Checksum supported
- Fast LIN BSL using BSL Mode protocol on single-wire UART (LIN)

Re-synchronization and setup of baud rate (Phase I) are always performed prior to the entry of Phase II and III. Thus different baud rates can be supported. Phase II is entered when its Master Request Header is received, otherwise Phase III is entered (Slave Response Header). The Master Request Header has a Protected ID of $3C_H$ while the Slave Response Header has a Protected ID of $7D_H$. The microcontroller responds to the host only after a Slave Response Header is received. The Command and Response LIN frames are identified as Diagnostic LIN frame which has a standard 8 data byte structure (instead of 2 or 4).

Upon entering LIN BSL, a connection is established and the transfer speed (baud rate) of the serial communication partner (host) is automatically synchronized in the following steps:

- STEP 1: Initialize interface for reception and timer for baud rate measurement
- STEP 2: Wait for an incoming LIN frame from the host
- STEP 3: Synchronize the baud rate to the host
- STEP 4: Enter Phase II (for Master Request Frame) or
- Phase III (for Slave Response Frame)

*Note: Re-synchronization and setup of baud rate are always done for **every** Master Request Header or Slave Response Header LIN frame.*

A Header LIN frame consists of the:

- Synch (SYN) Break (13 bit times low)
- Synch (SYN) byte ($55_H$)
- Protected Identifier (ID) field ($3C_H$ or $7D_H$)

The Break is used to indicate the beginning of a new frame and it must be at least 13 bits of dominant value. When a negative transition is detected at pin T2EX at the beginning of Break, the Timer 2 External Start Enable bit (T2MOD.T2RHEN) is set. This will then automatically start Timer 2 at the next negative transition of pin T2EX. Finally, the End of SYN Byte Flag (FDCON.EOFSYN) is polled. When this flag is set, Timer 2 is stopped. The time taken for the transfer (8 bits) is captured in the T2 Reload/Capture register (RC2H/L). Then the LIN BSL routine calculates the actual baud rate, sets the PRE and BG values and activates the Baud Rate Generator. The baud rate detection for LIN is shown in **Figure 19-2**.

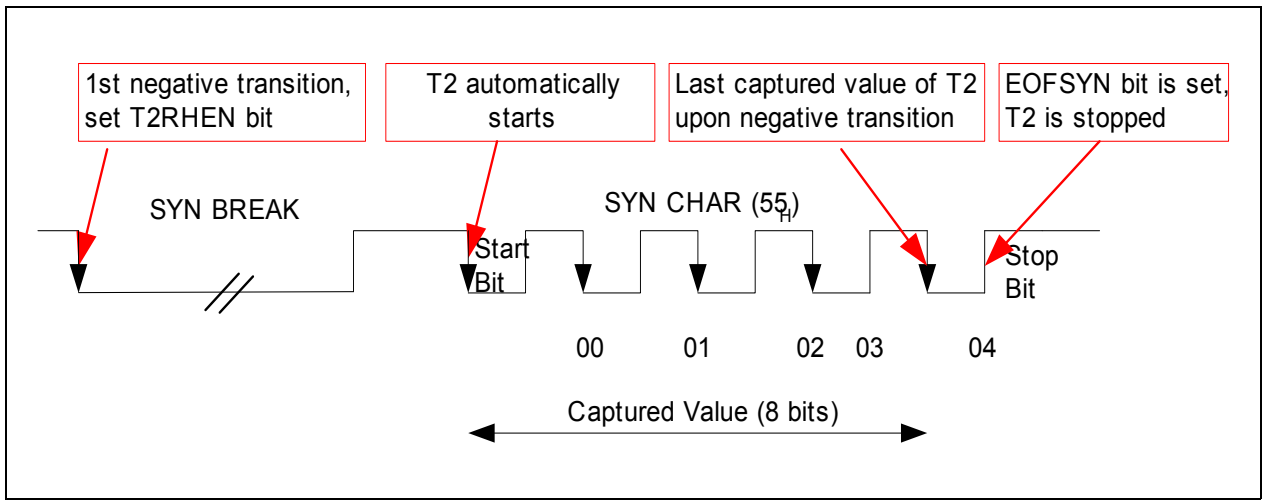



**Figure 19-2 LIN Auto Baud Rate Detection for Header LIN Frame**

### 19.1.3.1 Communication Structure

The transfer between the PC host and the microcontroller for the 3 phases is shown in **Figure 19-3** while **Figure 19-4** shows the Master Request Header, Slave Response Header, Command and Response LIN frames.

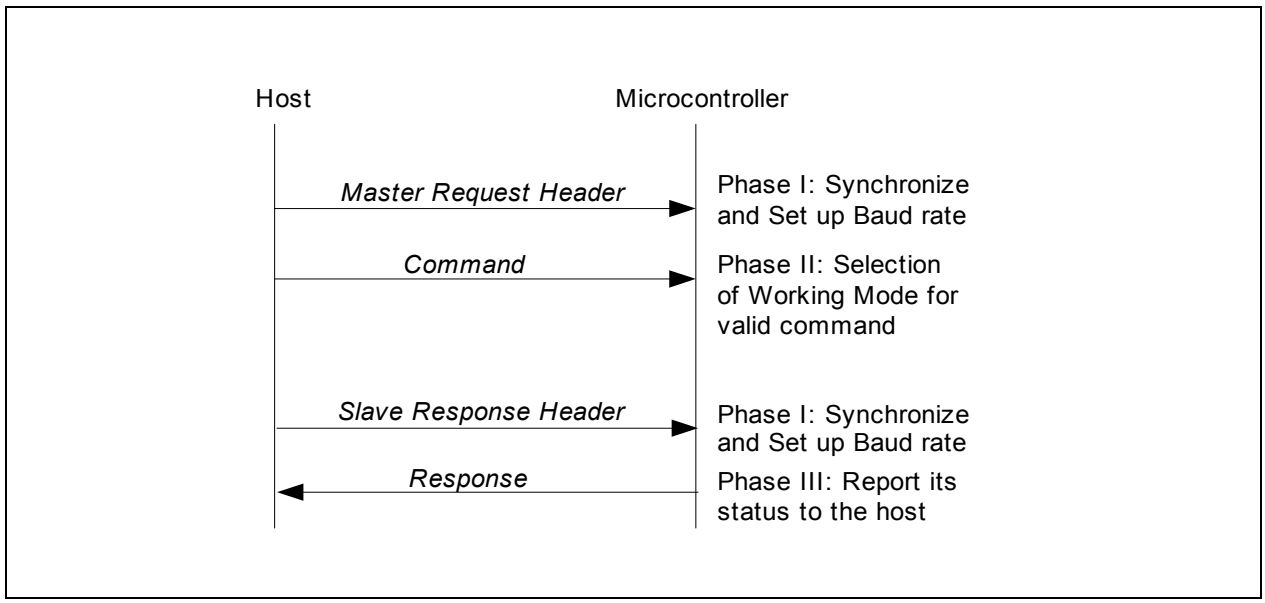



**Figure 19-3 LIN BSL - Phases I, II and III**

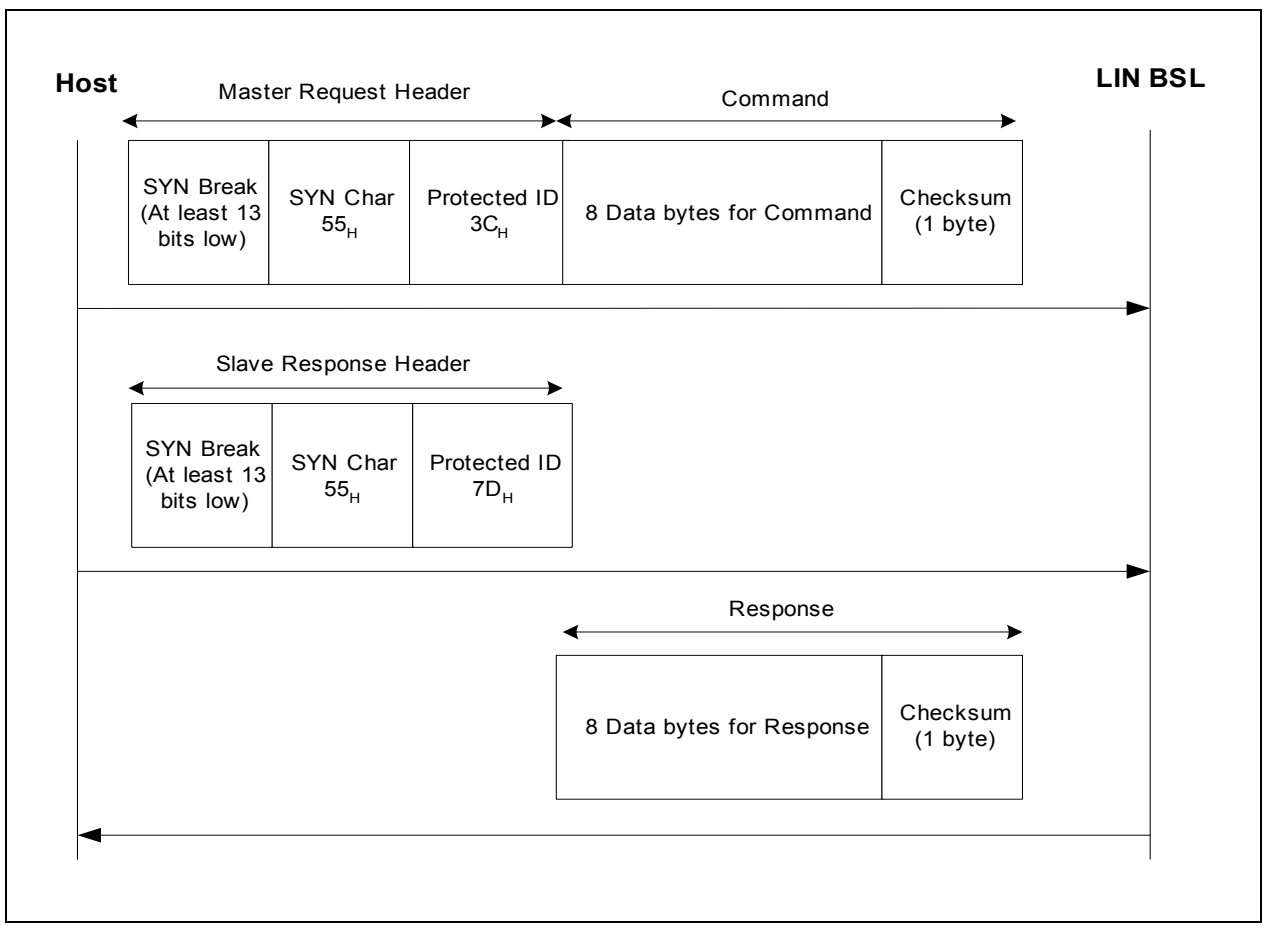



**Figure 19-4 LIN BSL Frames**

### 19.1.3.2 The Selection of Modes

When the LIN BSL routine enters Phase II, it first awaits for a 9-byte Header Block, from the host which contains the information for the selection of the modes, as shown below.

| NAD (1 byte) | Block Type $00_H$ (Header Block) | Data Area | | Checksum (1 byte) |
|---|---|---|---|---|
| | | Mode (1 byte) | Mode Data (5 bytes) | |

Description:

- **NAD**: Node Address for Diagnostic
- **$00_H$**: The block type, which marks the block as a **Header Block**
- **Mode**: The mode to be selected. Mode 0, 1, 2, 3, 6, 8, 9, 10, 15 and 16 are supported. See **Table 19-2**.
- **Mode Data**: Five bytes of special information to activate corresponding mode.
- **Checksum**: The Programming Checksum or LIN Checksum of the header block.

*Note: Mode 8 and Mode 9 support LIN Checksum while other modes support Programming Checksum.*

### 19.1.3.3 The Activation of Modes 0, 2 and 8

Mode 0, as well as Mode 8, and Mode 2[1)] are used to transfer a user program from the host to the XRAM and Flash of the microcontroller respectively. The header block has the following structure:

**The Header Block**

| NAD (1 byte) | $00_H$ (Header Block) | $00_H/02_H/08_H$ (Mode 0/2/8) | Mode Data | | | | | Checksum |
|---|---|---|---|---|---|---|---|---|
| | | | StartAddr High (1 byte) | StartAddr Low (1 byte) | Data Count (1 byte) | Not Used (2 bytes) | Fast_ Prog (1 byte) | |

Mode Data Description:

**Start Addr High, Low**: Lower 16-bit of the 20-bit Start Address[2)], which determines where to copy the received program code in the XRAM/Flash[3)].

1) Contents in the XRAM could be changed when Mode 2 is activated.

2) The upper 4 bits, or the bank, is dynamically assigned by the device based on the Flash device variant.

3) During each programming cycle, no crossing of a wordline boundary is allowed. For P-Flash programming, the Start Address must a valid address in even number.

**Data Count**: The number of data blocks[1] for this mode. The range is from 0 to 255.

**Not used**: 2 bytes, these bytes are not used and will be ignored in Mode 0/2/8.

**Fast_Prog**: Indication byte to enter Fast LIN BSL

- $01_H$: Enter Fast LIN BSL
- Other values: Ignored. Fast LIN BSL is not entered.

*Note: The **Block-Length** used in UART BSL is not implemented here, as a Diagnostic LIN frame has a standard 8 data bytes structure, followed by the checksum.*

When this Command LIN frame (Header Block) is used for entering Fast LIN BSL, no other Master Request Header and Command LIN frames (for Data Block or EOT Block) should be received. Instead, the microcontroller will receive a Slave Response Header LIN frame and send a Response LIN frame to acknowledge receiving correct header block to enter Fast LIN BSL where UART BSL protocol is used. See **Section 19.1.3.9**.

On successfully receipt of the Header Block, the microcontroller enters Mode 0/2/8, whereby the program code is transmitted from the host to the microcontroller by Data Block and EOT Block, which are described below.

### The Data Block

| NAD<br>(1 byte) | Data Block<br>$01_H$ | Program Code<br>(6 bytes) | Checksum<br>(1 byte) |
|---|---|---|---|

Description:

**Program Code**: The program code has a fixed length of 6 bytes per Data Block.

*Note: No empty Data Block is allowed.*

### The EOT Block

| NAD<br>(1 byte) | EOT Block<br>$02_H$ | Last_Codelength<br>(1 byte) | Program Code | Not Used | Checksum<br>(1 byte) |
|---|---|---|---|---|---|

Description:

**Last_Codelength**: This byte indicates the length of the program code in this EOT Block.

**Program Code:** The last program code (valid data) to be sent to the microcontroller.

**Not used**: The length is (LIN_Block_Length[2]-4-Last_Codelength). These bytes are not used and they can be set to any value.

1) Each data block is embedded in a Master Request Frame.

2) LIN_Block_Length is always 9 bytes, inclusive of a NAD and a checksum.

Internally, the microcontroller will transfer the valid data (6 bytes) of the Data Block into a buffer, and count the number of data bytes received. Microcontroller will program the data once the maximum buffer size is reached. If an EOT Block is received before maximum bytes are reached, then the remaining data bytes are programmed.

### 19.1.3.4 The Activation of Modes 1, 3 and 9

Mode 1, as well as Mode 9, and Mode 3 are used to execute a user program in the XRAM/Flash of the microcontroller at $0'F000_H$/$2'F000_H$(depending on Flash variant) and $0'0000_H$ respectively. The header block for this mode has the following structure:

**The Header Block**

| **NAD** (1 byte) | **$00_H$** (Header Block) | **$01_H$/$03_H$/$09_H$** (Mode 1/3/9) | **Mode Data** | **Checksum** (1 byte) |
|---|---|---|---|---|
| | | | Not Used (5 Bytes) | |

Mode Data Description:

**Not used**: The five bytes are not used and will be ignored in Mode 1/3/9.

For Modes 1, 3 and 9, the header block is the only transfer block to be sent by the host, no further serial communication is necessary. The microcontroller will exit the LIN BSL and jump to the XRAM address at $0'F000_H$/$2'F000_H$ (Mode 1 and Mode 9) and/or jump to Flash address at $0'0000_H$ (Mode 3).

### 19.1.3.5 The Activation of Mode 6

Mode 6[1)] is used to enable or disable Flash protection via the given user-password. The header block for this mode has the following structure:

**The Header Block**

| **NAD** (1 byte) | **$00_H$** (Header Block) | **$06_H$** (Mode 6) | **Mode Data** (5 bytes) | | **Checksum** (1 byte) |
|---|---|---|---|---|---|
| | | | **User-Password** (2 bytes) | **Not Used** (3 bytes) | |

Mode data description can be referred at **Section 19.1.2.5**.

1) Contents in the XRAM could be changed when Mode 6 is activated.

### 19.1.3.6 The Activation of Mode 10

Mode 10 ($0A_H$) is used to get 4 bytes data determined by the Option byte in the header block. The header block for this mode has the following structure:

**The Header Block**

| **NAD** (1 byte) | **$00_H$** (Header Block) | **$0A_H$** (Mode 10) | **Mode Data** (5 bytes) | | **Checksum** (1 byte) |
|---|---|---|---|---|---|
| | | | **Not Used** (4 bytes) | **Option** (1 byte) | |

Mode data description can be referred at **Section 19.1.2.6**.

### 19.1.3.7 The Activation of Mode 16

Mode 16 ($10_H$) is used to erase a page in P-Flash or D-Flash, or mass erase all memories in P-Flash block or D-Flash block. The selection of the type of erase is controlled through the Erase/MassErase byte in the Mode Data block.

**The Header Block**

| **NAD** (1 byte) | **$00_H$** (Header Block) | **$10_H$** (Mode 16) | **Mode Data** (5 bytes) | | | | **Checksum** (1 byte) |
|---|---|---|---|---|---|---|---|
| | | | **Start Addr High (1 byte)** | **Start Addr Low (1 byte)** | **Erase/ MassErase (1 byte)** | **Not Used (2 bytes)** | |

Mode data description can be referred at **Section 19.1.2.7**.

### 19.1.3.8 LIN Response Protocol to the Host

The microcontroller replies with a Response Block indicating its status when the host sends a Slave Response Header LIN frame. A Response transfer block, 9 bytes long (fixed), consists of four parts:

| **NAD** (1 byte) | **Response** (1 byte) | **Not Used** (6 bytes) | **Checksum** (1 byte) |
|---|---|---|---|

- **NAD**: Node Address for Diagnostic, which specifies the address of the active slave node
- **Response**: Acknowledgement or Error Status indication byte. See **Section 19.1.1.3**
- **Not Used**: These 6 bytes are ignored and are set to $00_H$

- **Checksum**: The LIN Checksum contains the eight bit sum with carry over NAD, Response and Not Used. All responses will adopt LIN Checksum regardless of modes

### 19.1.3.9 Fast LIN BSL

Fast LIN BSL is an enhanced feature in XC878 device, supporting higher baud rate up to 57.6 KHz. This is higher than Standard LIN, which supports only a baud rate of up to 20 kHz. This mode is especially useful during back-end programming, where faster programming time is desirable.

Fast LIN BSL is entered when the last byte of the Mode Data of Command LIN frame is $01_H$ (header block for LIN Modes 0, 2 and 8). See **Section 19.1.3.3**. When Fast LIN BSL Master Request Header and Command LIN frames are received, the microcontroller will wait for the Slave Response Header LIN frame before sending back the Response LIN frame. The host will then send the header block using BSL UART protocol at the calculated high baud rate. See **Figure 19-5**. Microcontroller will stay at Fast LIN BSL, and the communication structure and selection of modes will be like BSL Mode via UART as shown in **Section 19.1.2.1** and **Section 19.1.2.2**.

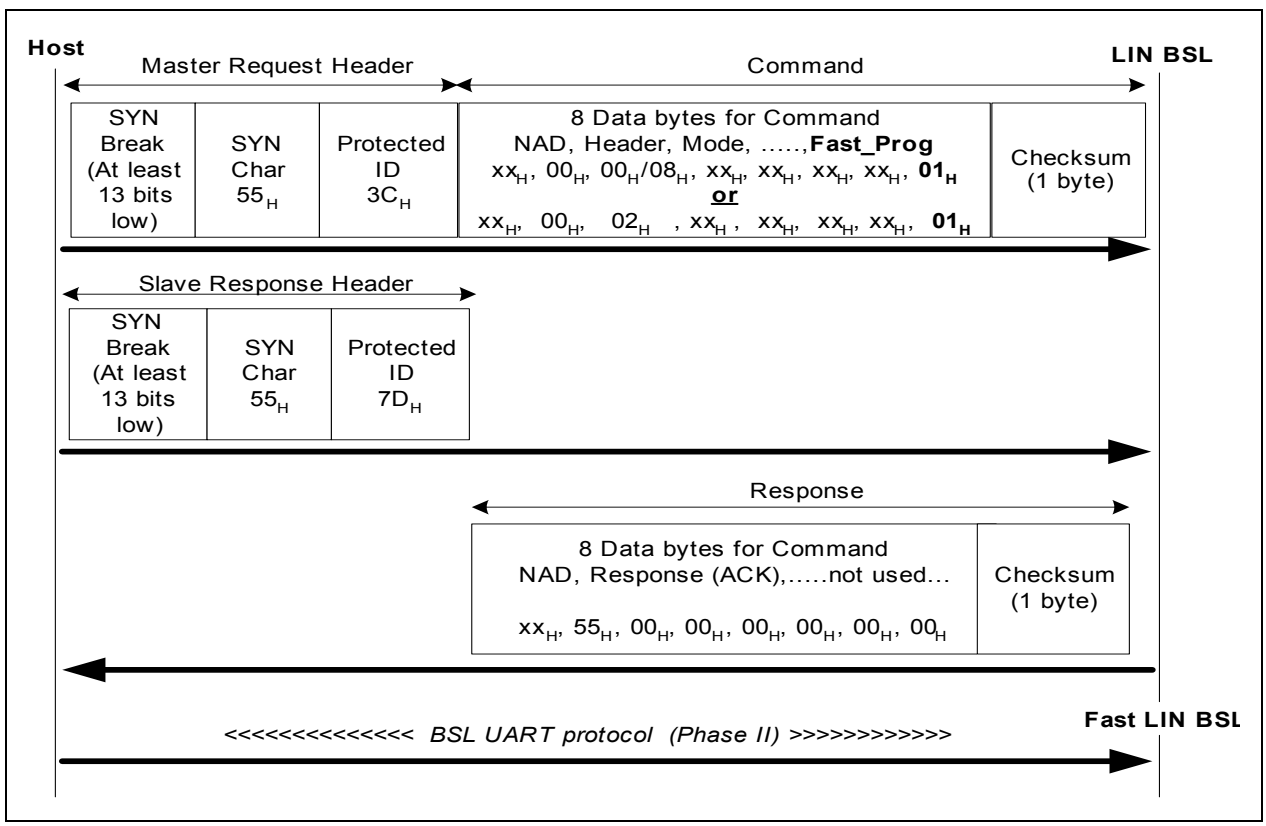



**Figure 19-5 Fast LIN BSL Frames**

### 19.1.3.10 User Defined Parameter for LIN BSL

The NAD (Node Address for Diagnostic) value, which specifies the address of the active slave node for the LIN modes, is programmed into the uppermost P-Flash block. This parameter is specified by the user.

User needs to program the NAD in the format shown in **Table 19-6**. To ensure the validity of the parameter, the inverted NAD value is required to be programmed together with the actual value. If an invalid NAD is programmed, the default NAD value is assumed.

**Table 19-6 User Defined Parameters**

| Address | User Defined Value | Criteria / Range | Default |
|---|---|---|---|
| $0'BFFE_H$ [1)]<br>$0'EFFE_H$ [2)] | NAD | $01_H$ – $0FF_H$ ($00_H$ is reserved) | $7F_H$ |
| $0'BFFF_H$ [1)]<br>$0'EFFF_H$ [2)] | $\overline{NAD}$ | - | - |

1) For a Flash device that has 52 Kbytes of Flash memory.

2) For a Flash device that has 64 Kbytes of Flash memory.

*Note: For a variant device with LIN BSL support, it must be ensured that a valid NAD is programmed before protecting the device. Device access is not granted without having the correct NAD.*

## 19.2 MultiCAN BSL Mode

MultiCAN BSL can be entered only when Flash is not protected, else user mode is entered instead and code from memory address location $0000_H$ will be executed. The MultiCAN BSL protocol is divided into two sections, hardware initialisation and software communication.

In the hardware initialisation section, XC878 is configured to use an external oscillator and CAN node 0 for communication. The use of external oscillator is to ensure an optimal performance on CAN applications, which requires the oscillator to have a frequency deviation of less than 1.5 %. XC878 supports four oscillator frequency values, which the user can enter at the top address of the P-Flash block. The usage for user defined parameter is described in **Section 19.2.3**.

In the software communication section, three main phases have been identified, namely the Autobaud, Acknowledgement and Data Reception phases. All three phases involves the transmission and reception of CAN Message Objects[1)].

The Autobaud phase is started on entry to MultiCAN BSL where the host sends a Host Command Message to the microcontroller. The microcontroller will determine the current CAN network baud rate and configure the baud rate of the CAN node accordingly to enable the communication channel. In the Acknowledgement Phase, the microcontroller sends an Acknowledge Message to the host to establish the communication channel. With the communication channel established, the Data Reception Phase can now be started. The host sends Data Message Objects to download the code into XRAM and execute the code from there. In the XC878, there are 2 Kbytes of XRAM available for program execution.

The following assumptions are introduced to keep the MultiCAN BSL implementation simple:

- Host and the XC878 are the only CAN node in the CAN network (Point to Point Connection)
- CAN Node 0 (P1.0/P1.1) on the XC878 is used for this mode
- XC878 expects to receive a standard CAN frame with message identifier of $555_H$.

### 19.2.1 Communication protocol

Data is exchanged using Message Objects implemented with the standard CAN data frame (11 bit identifier) as shown in **Figure 19-6**. Message Objects with other message identifiers are ignored by XC878. The data field in a standard CAN message is used to implement the communication protocol.

1) CAN Message Object refers to a standard CAN data frame as defined in BOSCH CAN Specification 2.0B

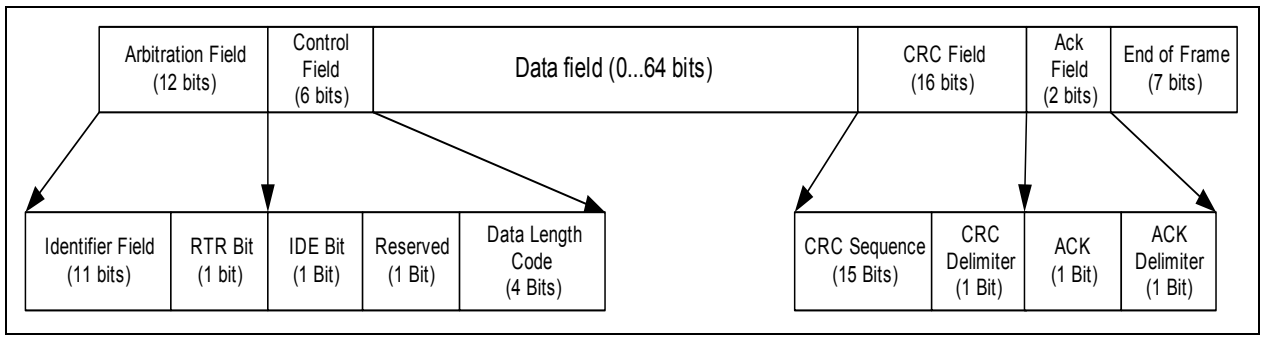



**Figure 19-6 Standard CAN frame format**

Communication is initiated by the host, which continuously sends a Host Command Message Object until it receives an Acknowledgement Message Object from the microcontroller.

After the baud rate is determined and the acknowledgement is received by the host, the host can activate the MultiCAN BSL operational mode by sending the Data Message Object. All messages received from this point on will have their data bytes sequentially written into the XRAM. The size of the internal XRAM is 2 kbytes which results in a maximum of 2048 8-bit instructions.

Once all messages have been received, the CAN module will be reinitialized. The bootstrap loader then terminates its sequence and transfers program execution to the user code by jumping to location 0'F000$_H$/ 2'F000$_H$ (i.e. the first loaded instruction). The program that was loaded into the XRAM from the host will now be executed.

*Note: The bootstrap loader assumes all message data is valid. The host should send its code/data sequentially in multiples of 8 code/data bytes. The user is limited to sending a maximum of 256 messages.*

## 19.2.2 CAN Message Object definition

### Host Command Message Object

In the Autobaud phase, the Host Command message is sent by the host and used for automatic baud rate detection. Since there are no other nodes (Point-to-Point) on the bus, the host will continually send the message. The host will transmit this message and wait for the microcontroller to acknowledge it.

The Host Command message data field contains 8 bytes of information for enabling the BSL mode. The first 2 data bytes, Byte 0 and 1, contain the value 0x5555. The next 2 data bytes, Bytes 2 and 3, contain the identifier for an acknowledge message that the microcontroller sends back to the host. Bytes 4 and 5, contain the 16-bit value for the number of messages to be received. The final 2 data bytes, bytes 6 and 7 contain the identifier for the data messages that the host will send to the XC878 device.

The message identifier is $555_H$ and the data length code is set to 8.

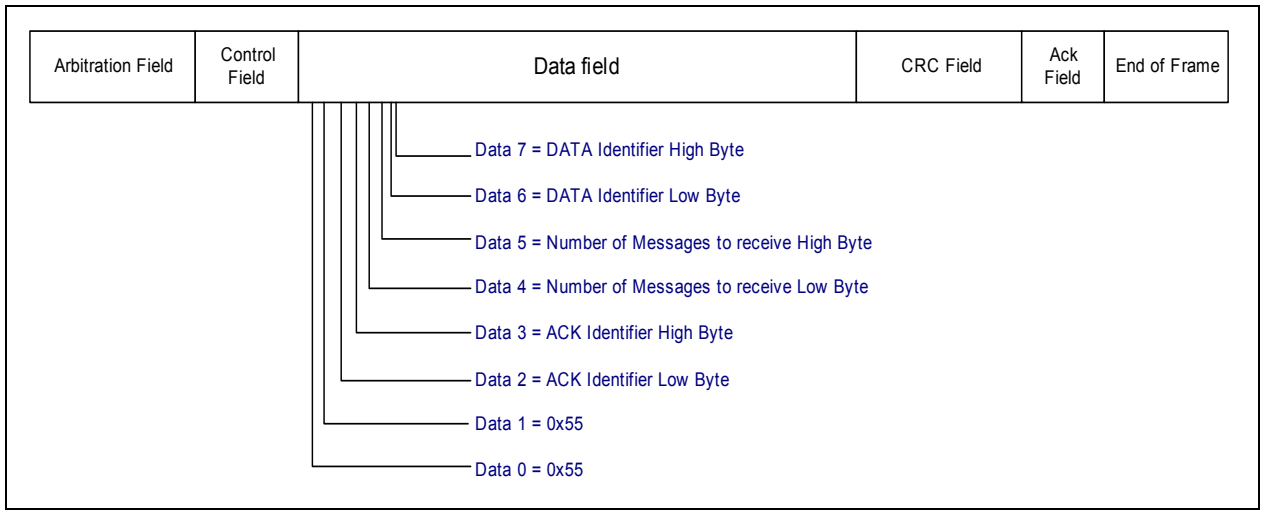



**Figure 19-7 Host Command Message Format**

### Acknowledgement Message Object

In the Acknowledgement phase, this message is sent by the microcontroller after successfully determining the CAN network baud rate. The message identifier used is specified by the host and determined from the Host message (Data bytes 2 and 3) received. The data length code is set to 4.

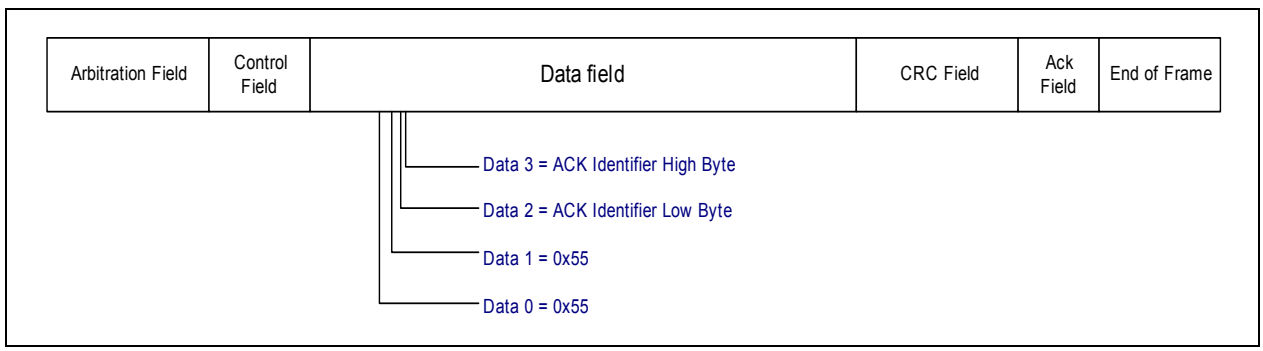



**Figure 19-8 Acknowledgement Message Format**

### Data Message Object

In the Data Reception phase, this message is sent by the host with a host specified Data Identifier, which is defined in data bytes 6 and 7 of the Host Command message. The data field contains user code/data that is required for the BSL Mode. The data received is then loaded to the XRAM.

### 19.2.3 User Defined Parameter for MultiCAN BSL

The OSC value, which specifies the oscillator frequency connected to the device, is programmed into the uppermost P-Flash block. This parameter is specified by the user.

**Table 19-7** shows the address, supported values and default value of the user defined parameter for unprotected flash. To ensure the validity of the parameter, the inverted values are required to be programmed together with the actual values. A check is done to verify whether the addition of the inverted value, actual value and $01_H$, will give $00_H$.

**Table 19-7 User Defined Parameter for MultiCAN BSL**

| Address[1)] | Parameter | Value |
|---|---|---|
| $0'EFF9_H$ | OSC | Byte indicate external oscillator frequency<br>$00_H$: 4 MHz<br>$01_H$: 6 MHz<br>$02_H$: 8 MHz<br>$03_H$: 12 MHz<br>Others: 8 MHz (default) |
| $0'EFFA_H$ | $\overline{OSC}$ | Byte indicate external oscillator frequency<br>$FF_H$: 4 MHz<br>$FE_H$: 6 MHz<br>$FD_H$: 8 MHz<br>$FC_H$: 12 MHz<br>Others: 8 MHz (default) |

1) The address shown in the table assumes a device with 64 Kbytes of Flash memory. For a device with 52 Kbytes of Flash memory, OSC and $\overline{OSC}$ adress is $0'BFF9_H$ and $0'BFFA_H$ respectively.

## 19.3 Alternate BSL Mode

Alternate BSL mode is a mode with user defined BSL code. The BSL code is programmed in the Flash memory with the start address specific by JumpAddrH (Jump Address High byte) and JumpAddrL (Jump Address Low Byte). It is entered via the BSL mode with a vaild AltBSL password as shown in **Table 19-1**. These parameters are specified by the user in the uppermost P-Flash block.

User needs to program the AltBSLPassword, JumpAddrH and JumpAddrL in the format shown in **Table 19-6**. To ensure the validity of the parameter, the inverted AltBSLPassword value is required to be programmed together with the actual value. In addition, the LJMP opcode ($02_H$) needs to be programmed in the address as shown in **Table 19-6**.

**Table 19-8 User Defined Parameters in relation with Unprotected Flash**

| Address | User Defined Value | Criteria / Range |
|---|---|---|
| $0'BFFC_H$ [1] <br> $0'EFFC_H$ [2] | AltBSLPassword | $00_H$ – $0FF_H$ |
| $0'BFFD_H$ [1] <br> $0'EFFD_H$ [2] | $\overline{\text{AltBSLPassword}}$ | - |
| $0'BFFA_H$ [1] <br> $0'EFFA_H$ [2] | JumpAddrH | $00_H$ – $0FF_H$ [3] |
| $0'BFFB_H$ [1] <br> $0'EFFB_H$ [2] | JumpAddrL | $00_H$ – $0FF_H$ [3] |
| $0'BFF9_H$ [4] <br> $0'EFF9_H$ [4] | $02_H$ (LJMP Opcode) | - |

1) For a Flash device that has 52 Kbytes of Flash memory.

2) For a Flash device that has 64 Kbytes of Flash memory.

3) User need to ensure that the Flash address is valid based on the Flash variants.

4) User needs to ensure that address $0'xFF8_H$ is also programmed because the start address of P-Flash needs to be even.

For the case of an invalid AltBSLPassword being programmed, the chip will then entered the LIN or UART/MultiCAN BSL mode based on the product variants.

# 20 Index

## 20.1 Keyword Index

This section lists a number of keywords which refer to specific details of the XC878 in terms of its architecture, its functional units, or functions.


## A

Accumulator 2-3
Alternate functions 6-8
  Input 6-8
  Output 6-8
Analog input clock 17-3
Analog-to-Digital Converter 17-1
  Interrupt 17-23
    Channel 17-25
    Event 17-24
    Node pointer 17-25
  Low power mode 17-7
  Module clock 17-3
  Register description 17-34
  Register map 17-31
Arbitration round 17-9
Arbitration slot 17-9
Arithmetic 2-1
Automatic refill 17-12
Autoscan 17-16

## B

B Register 2-3
Baud-rate generator 12-11
  Fractional divider
    Fractional divider mode 12-14
    Normal divider mode 12-15
Bit protection scheme 3-19
Boot options 7-7
  BSL mode 7-7, 19-1
  OCDS mode 7-7
  User mode 7-7
Boot ROM 3-1
Boot ROM operating mode 3-44
  BootStrap Loader Mode 3-47
  OCDS mode 3-47
  User JTAG mode 3-47
  User mode 3-46
Booting scheme 7-7
Bootstrap loader 3-47, 4-9, 19-1
  Fast LIN BSL 19-22
  LIN BSL 19-15
    Communication structure 19-16
    Mode selection 19-18
  MultiCAN BSL 19-24
    Communication protocol 19-24
    Message object 19-25
  Response code 19-6
  UART BSL 19-9
    Communication structure 19-10
    Mode selection 19-10
Brownout reset 7-5
Buffer mechanism 4-2

## C

CAN
  Block diagram 16-1
  MultiCAN
    Bit timing 16-8
    Block diagram 16-4
    Interrupt structure 16-6
    Low Power Mode 16-44
    Message acceptance filtering 16-21
    Message object data handling 16-27

Message object FIFO 16-34
Message object functionality 16-33
Message object interrupts 16-23
Message object lists 16-13
Node control 16-8
Node interrupts 16-11
Register map 16-46
Registers
Offset addresses 16-45
MultiCAN registers
LISTi **16-54**
MCR **16-52**
MITR **16-53**
MOAMRn **16-91**
MOARn **16-92**
MOCTRn **16-76**
MODATAHn **16-96**
MODATALn **16-95**
MOFCRn **16-86**
MOFGPRn **16-90**
MOIPRn **16-84**
MOSTATn **16-79**
MSIDk **16-57**
MSIMASK **16-58**
MSPNDk **16-56**
NBTRx **16-69**
NCRx **16-59**
NECNTx **16-71**
NFCRx **16-72**
NIPRx **16-66**
NPCRx **16-68**
NSRx **16-63**
PANCTR **16-48**
Cancel-Inject-Repeat 17-10
Capture/Compare Unit 6 15-1
Low Power Mode 15-26
Module Suspend Control 15-27
Register description 15-36
Register map 15-33
Central Processing Unit 2-1
Chip identification number 1-18
Circular stack memory 4-2
Clock management 7-15
Clock source 7-14
Clock system 7-10
Register description 7-17
Conversion error 17-4
Conversion phase 17-5
CORDIC 11-1
Accuracy 11-9
Normalized Deviation 11-10
Calculated Data 11-1
CORDIC equations 11-1
Data Format 11-8
Data Overflow 11-8
Domains of Convergence 11-7
Features 11-2
Functional Description 11-3
Gain Factor 11-4
Initial Data 11-1
Interrupt 11-4
Look-Up Tables
atan 11-12
atanh 11-12
linear 11-13
Low power mode 11-14
Normalized Result Data 11-4
Operating Modes 11-5
Circular Function 11-5
Hyperbolic Function 11-6
Linear Function 11-5
Rotation Mode 11-5
Usage Notes 11-6
Vectoring Mode 11-5
Performance 11-11
Register description 11-16
Register map 11-15
Result Data 11-1
Correction algorithm 4-8
Count Clock 13-19
Counter 13-2, 13-14
CPU 2-1
CPU Registers
Extended Operation 2-5
Power Control 2-6

## D

Data Flash 4-2
Data memory 3-3
Data pointer 2-3
Data reduction 17-19
  Counter 17-20
Debug 18-2
  Events 18-3
Debug suspend control 18-7
D-Flash 4-2
DFLASHEN 3-9
Digital input clock 17-3
Direct Drive 7-14
Division operation 10-3
Document
  Acronyms 1-20
  Terminology 1-20
  Textual convention 1-7
  Textual conventions 1-19
Dynamic error detection 4-8

## E

EEPROM emulation 4-2
Embedded voltage regulator 7-1
  Features 7-2
  Low power voltage regulator 7-2
  Main voltage regulator 7-2
  Threshold voltage levels 7-2
Enter BSL mode 19-1
Error Correction Code 4-8
Extended operation 2-5
External breaks 18-5
  Break now 18-5
External data memory 3-4
External oscillator 7-10, 7-12, 7-13

## F

Fast LIN BSL 19-22
Flash 4-1
  Endurance 4-3
  Non-volatile 4-1
  Operating modes 4-3
    Abort Operation 4-5
    Erase Operation 4-4
    Mass Erase Operation 4-4
    Program Operation 4-3
    Read Operation 4-3
Flash abort 8-3
Flash address mapping 4-2
Flash memory
  Register description 4-14
  Register map 4-14
Flash memory protection 3-5
Flash Pagination 4-2
Flash program memory 3-1
Flash Timer 4-7

## G

GPIO 6-1, 6-5

## H

Hall sensor mode
  Actual hall pattern 15-21
  Block commutation 15-23
  Brushless-DC 15-21, 15-22
  Correct hall event 15-21
  Expected Hall pattern 15-21
  Hall pattern 15-21
  Modulation pattern 15-21
  Noise filter 15-21
Hamming code 4-8
Hardware breakpoints 18-3
Hardware reset 7-4
High-impedance 6-2

## I

Idle mode 7-15, 8-2
In-Application Programming 4-10
  Flash page erasing and mass erasing 4-11
  Flash programming 4-10
  Get chip information 4-12
Input class 17-8
Instruction decoder 2-1
Instruction timing 2-11, 2-13

CPU state 2-11
Mnemonic 2-13
Wait state 2-11
In-System Programming 4-9
Internal analog clock 17-3
Maximum frequency 17-3
Internal data memory 3-3
Internal RAM 3-1
Interrupt handling 5-14
Interrupt request flags 5-38
Interrupt response time 5-15
Interrupt source and vector 5-11
Interrupt structure 5-8
Interrupt system 5-1
Register description 5-17

## J

JTAG ID 18-11

## L

Limit checking 17-19
LIN 12-26–12-30
Break field 12-27
Header transmission 12-28
LIN frame 12-26
LIN protocol 12-26
Synch byte 12-27
LIN BSL 19-15

## M

Maskable interrupt 5-1
Memory Extension 2-7
Memory Extension Effect 2-7
Memory Extension Stack 2-7
Memory organization 3-1
Special Function Registers 3-10
Address extension by mapping 3-10
Mapped 3-10
Standard 3-10
Address extension by paging 3-13
Local address extension 3-13
Save and restore 3-14
Memory protection 3-5
Minimum erase width 4-2
Minimum program width 4-1
Modulation 15-15
Monitor mode control 18-1
Monitor RAM 18-1
Data 18-6
Stack 18-6
Monitor ROM 18-1
MultiCAN BSL 19-24
Multi-Channel Mode 15-19
Multiplication/Division Unit 10-1
Error detection 10-4
Interrupt generation 10-4
Low power mode 10-5
Register description 10-7
Register map 10-6

## N

Non-maskable interrupt
Events 5-1
Normalize operation 10-3

## O

On-Chip Debug Support 18-1
Register description 18-8
Register map 18-8
On-chip oscillator 7-10

## P

P0 register description 6-16
P1 register description 6-24
P3 register description 6-32
P4 register description 6-40
P5 register description 6-48
Parallel ports 6-1
Bidirectional port structure 6-3
Driver 6-2, 6-6
General port structure 6-3
General register description 6-4
Kernel registers 6-4
Direction control register 6-6

Offset addresses 6-10
Open drain control register 6-6
Normal mode 6-2
Open drain mode 6-2
Parallel request source 17-14
Password 3-7
Peripheral clock management 8-6
Permanent arbitration 17-9
Personal computer host 4-9
P-Flash 4-2
Phase-Locked Loop 7-10
Changing PLL parameters 7-12
Pin
Configuration 1-6
PLL
Loss-of-clock 7-11
Startup 7-11
PLL mode 7-14
Power control 2-6
Power saving modes 8-1
Power supply system 7-1
Power-down mode 7-15, 8-3
Entering power-down mode 8-5
Exiting power-down mode 8-5
Power-down wake-up reset 7-5
Power-on reset 7-2, 7-3
Prewarning period 9-2
Processor architecture 2-1
Instruction timing
Machine cycle 2-11
Register description 2-3
Program control 2-1
Program counter 2-3
Program Flash 4-2
Program memory 3-3
Program status word 2-4
Pull-down device 6-7
Pull-up device 6-7
Pulse width modulation 15-1

## R

Read access time 4-1
Read-out protection 3-5
Request gating 17-13
Request trigger 17-13, 17-15, 17-16, 17-27
CCU6 Event 17-27
Reset control 7-3
Module behavior 7-6
Result read view 17-21
Accumulated 17-21
Normal 17-21
RS-232 4-9

## S

Sample phase 17-5
Schmitt-Trigger 6-2
Sequential request source 17-11
Serial interfaces 12-1–12-30
Shift operation 10-3
Slow-down mode 7-15, 8-2
Software breakpoints 18-5
Break before make 18-5
Source priority 17-9
Special Function Register area 3-1
Stack pointer 2-3
Synchronization phase 17-5
Synchronous serial interface 12-31
Baud rate generation 12-40
Continuous transfer operation 12-37
Data width 12-33
Error detection 12-42
Baud rate error 12-43
Phase error 12-43
Receive error 12-42
Transmit error 12-43
Full-duplex operation 12-33
Half-duplex operation 12-36
Interrupts 12-42
Low power mode 12-46
Master mode 12-31
Operating mode 12-32
Port control 12-38
Register description 12-47
Register map 12-46
Right-aligned 12-33
Slave mode 12-31

## T

Timer 0 and Timer 1 13-1
- Counter 13-2
- External control 13-2
- Mode 0, 13-bit timer 13-4
- Mode 1, 16-bit timer 13-5
- Mode 2, 8-bit automatic reload timer 13-6
- Mode 3, two 8-bit timers 13-7
- Port control 13-8
- Register description 13-10
- Register map 13-9
- Timer operations 13-2
- Timer overflow 13-2

Timer 2 13-14–13-29
- Auto-Reload mode 13-14
  - Up/Down Count Disabled 13-14
  - Up/Down Count Enabled 13-16
- Capture mode 13-18
- Counter 13-14
- External interrupt function 13-20
- Low power mode 13-21
- Module suspend control 13-22
- Port control 13-20
- Register description 13-25
- Register map 13-24
- Timer operations 13-14

Timer T12 15-3
- Capture mode 15-9
- Center-aligned mode 15-4
- Compare mode 15-6
- Dead-time 15-8
- Duty cycle 15-8
- Edge-aligned mode 15-4
- Hysteresis-like control mode 15-10
- Shadow transfer 15-3
- Single-shot mode 15-10
- Three-phase PWM 15-1

Timer T13 15-12
- Compare mode 15-13
- Shadow transfer 15-12
- Single-shot mode 15-13

Timer/counter
- Timer/counter 2 14-8–??
  - Block diagram 14-9
  - Capture function 14-19–??
  - Compare function 14-10–??
  - Compare mode 0 14-13–??
  - Compare mode 1 14-15–??
  - Compare mode interrupts 14-18
  - Port functions 14-8
  - Registers 14-21–??

Total conversion time 17-6
Trap handling 15-17
Tristate 6-7

## U

UART BSL 19-9
UART/UART1 module 12-2–12-25
- Baud rate generation 12-10
- Interrupt requests 12-5
- Mode 1, 8-bit UART 12-3
- Mode 2, 9-bit UART 12-5
- Mode 3, 9-bit UART 12-5
- Modes 12-2
- Port control 12-23
- Receive-buffered 12-2
- Register map 12-25

UART1 module
- Low power mode 12-24

User programmable password 3-7

## W

Wait-for-read mode 17-17
Wait-for-Start 17-10
Watchdog timer 9-1
- Input frequency 9-3
- Module suspend control 9-4
- Register description 9-5
- Servicing 9-2
- Time period 9-3

Watchdog timer reset 7-5
Window boundary 9-2
Write result phase 17-5

# X

XC878
- Device configuration 1-2
- Device Profile 1-3
- Feature summary 1-4
- Functional units 1-2

XRAM 3-1

## 20.2 Register Index

This section lists the references to the Special Function Registers of the XC878.


### A

ACC 2-3
ADC_PAGE 17-32
ADCON 16-97
ADH 16-98
ADL 16-98

### B

B 2-3
BCON 12-16
BG 12-18
BRH 12-52
BRL 12-52

### C

CAN_LISTi 16-54
CAN_MCR 16-52
CAN_MITR 16-53
CAN_MOAMRn 16-91
CAN_MOARn 16-92
CAN_MOCTRn 16-76
CAN_MODATAHn 16-96
CAN_MODATALn 16-95
CAN_MOFCRn 16-86
CAN_MOFGPRn 16-90
CAN_MOIPRn 16-84
CAN_MOSTATn 16-79
CAN_MSIDk 16-57
CAN_MSIMASK 16-58
CAN_MSPNDk 16-56
CAN_NBTRx 16-69
CAN_NCRx 16-59
CAN_NECNTx 16-71
CAN_NFCRx 16-72
CAN_NIPRx 16-66
CAN_NPCRx 16-68
CAN_NSRx 16-63
CAN_PANCTR 16-48
CC63RH 15-54
CC63RL 15-54
CC63SRH 15-55
CC63SRL 15-55
CC6xRH 15-49
CC6xRL 15-49
CC6xSRH 15-50
CC6xSRL 15-49
CCU6_PAGE 15-34
CD_CON 11-16
CD_CORDxH (x = X, Y or Z) 11-20
CD_CORDxL (x = X, Y or Z) 11-19
CD_STATC 11-18
CHCTRx (x = 0 - 7) 17-40
CHINCR 17-60
CHINFR 17-60
CHINPR 17-61
CHINSR 17-61
CMCON 7-20
CMPMODIFH 15-59
CMPMODIFL 15-58
CMPSTATH 15-57
CMPSTATL 15-56
COCON 7-22
CONH 12-48, 12-50
CONL 12-47, 12-50
CR_MISC 7-9, 7-23
CRCR1 17-50
CRMR1 17-52
CRPR1 17-51

### D

DATA0 16-99
DATA1 16-99
DATA2 16-99
DATA3 16-100
DPH 2-3

DPL 2-3

## E

EECON 4-15
EO 2-5
ETRCR 17-39
EVINCR 17-62
EVINFR 17-62
EVINPR 17-63
EVINSR 17-63
EXICON0 5-21
EXICON1 5-22

## F

FCON 4-14
FCS 4-16
FCS1 4-18
FDCON 12-19
FDRES 12-21
FDSTEP 12-20
FEAH 4-18
FEAL 4-18
FTVAL 4-17

## G

GLOBCTR 8-10, 17-34
GLOBSTR 17-36

## I

IEN0 5-17, 13-13
IEN1 5-18
IENH 15-90
IENL 15-88
INPCR0 17-41
INPH 15-93
INPL 15-91
IP 5-35
IP1 5-36
IPH 5-36
IPH1 5-37
IRCON0 5-27
IRCON1 5-28
IRCON2 5-29
IRCON3 5-30
IRCON4 5-31
ISH 15-81
ISL 15-80
ISRH 15-87
ISRL 15-86
ISSH 15-85
ISSL 15-84

## L

LCBR 17-64

## M

MCMCTR 15-79
MCMOUTH 15-78
MCMOUTL 15-76
MCMOUTSH 15-75
MCMOUTSL 15-74
MD4 10-9
MDUCON 10-11
MDUSTAT 10-13
MDx (x = 0 - 5) 10-9
MEX1 2-9
MEX2 2-9
MEX3 2-9
MEXSP 2-10
MISC_CON 3-9, 17-28
MMWR2 18-10
MODCTRH 15-69
MODCTRL 15-68
MODIEN 12-45
MODPISEL 5-23, 8-8, 12-23, 18-10
MODPISEL1 5-24, 12-23, 13-21
MODPISEL2 13-8, 13-20
MODPISEL3 12-39
MODPISEL4 5-25
MODSUSP 9-4, 13-22, 15-27
MR4 10-10
MRx (x = 0 - 5) 10-9

## N

NMICON 5-19
NMISR 5-33

## O

OSC_CON 7-17, 8-11

## P

P0_ALTSEL0 6-18
P0_ALTSEL1 6-18
P0_DATA 6-16
P0_DIR 6-16
P0_DS 6-19
P0_OD 6-17
P0_PUDEN 6-18
P0_PUDSEL 6-17
P1_ALTSEL0 6-26
P1_ALTSEL1 6-26
P1_DATA 6-24
P1_DIR 6-24
P1_DS 6-27
P1_OD 6-25
P1_PUDEN 6-26
P1_PUDSEL 6-25
P3_ALTSEL0 6-34
P3_ALTSEL1 6-35
P3_DATA 6-32
P3_DIR 6-32
P3_DS 6-35
P3_OD 6-33
P3_PUDEN 6-34
P3_PUDSEL 6-33
P4_ALTSEL0 6-42
P4_ALTSEL1 6-42
P4_DATA 6-40
P4_DIR 6-40
P4_DS 6-43
P4_OD 6-41
P4_PUDEN 6-42
P4_PUDSEL 6-41
P5_ALTSEL0 6-50
P5_ALTSEL1 6-50
P5_DATA 6-48
P5_DIR 6-48
P5_DS 6-51
P5_OD 6-49
P5_PUDEN 6-50
P5_PUDSEL 6-49
PASSWD 3-19
PCON 2-6, 8-8, 12-10
PISEL0H 15-39
PISEL0L 15-38
PISEL2 15-40
PLL_CON 7-18
PLL_CON1 7-19
PMCON0 7-8, 8-7, 9-8
PMCON1 8-9, 10-5, 11-14, 12-46, 13-21, 15-26, 16-44, 17-7
PMCON2 8-10, 12-24, 13-22
PORT_PAGE 6-10
PRAR 17-37
PSLR 15-73
PSW 2-4
Px_ALTSELn 6-8
Px_DATA 6-5
Px_DIR 6-6
Px_DS 6-9
Px_OD 6-6
Px_PUDEN 6-7
Px_PUDSEL 6-7

## Q

Q0R0 17-46
QBUR0 17-48
QINR0 17-49
QMR0 17-42
QSR0 17-44

## R

RBL 12-53
RC2H 13-28
RC2L 13-28
RCRx (x = 0 - 3) 17-58
RESRAxH (x = 0 - 3) 17-57
RESRAxL (x = 0 - 3) 17-56
RESRxH (x = 0 - 3) 17-55
RESRxL (x = 0 - 3) 17-54

## S

SBUF 12-8
SCON 5-33, 12-8
SCON1 12-9
SCU_PAGE 3-17
SP 2-3
SYSCON0 3-12, 5-10

## T

T12DTCH 15-51
T12DTCL 15-51
T12H 15-47
T12L 15-47
T12MSELH 15-45
T12MSELL 15-43
T12PRH 15-48
T12PRL 15-48
T13H 15-53
T13L 15-52
T13PRH 15-54
T13PRL 15-53
T2_PAGE 14-23
T2CCU_CCEN 14-30
T2CCU_CCTBSEL 14-24
T2CCU_CCTCON 14-27
T2CCU_CCTDTCH 14-29
T2CCU_CCTDTCL 14-28
T2CCU_CCTH 14-26
T2CCU_CCTL 14-26
T2CCU_CCTRELH 14-25
T2CCU_CCTRELL 14-25
T2CCU_CCxH 14-34
T2CCU_CCxL 14-33
T2CCU_COCON 14-32
T2CCU_COSHDW 14-30
T2CON 13-26
T2CON1 13-27
T2H 13-29
T2L 13-29
T2MOD 13-25
TBL 12-53
TCON 5-26, 5-32, 13-11
TCTR2H 15-65
TCTR2L 15-63
TCTR4H 15-67
TCTR4L 15-66
TCTROH 15-61
TCTROL 15-60
THx (x = 0 - 1) 13-10
TLx (x = 0 - 1) 13-10
TMOD 13-12
TRPCTRH 15-72
TRPCTRL 15-70

## V

VFCR 17-58

## W

WDTCON 9-6
WDTH 9-7
WDTL 9-7
WDTREL 9-5
WDTWINB 9-8

## X

XADDRH 3-4

w w w . i n f i n e o n . c o m

Published by Infineon Technologies AG