XC161

16-Bit Single-Chip Microcontroller with C166SV2 Core

Volume 2 (of 2): Peripheral Units

Microcontrollers

Never stop thinking.

Edition 2004-01

Published by Infineon Technologies AG,
St.-Martin-Strasse 53,
81669 München, Germany

© Infineon Technologies AG 2004.
All Rights Reserved.

Attention please!

The information herein is given to describe certain components and shall not be considered as a guarantee of characteristics.

Terms of delivery and rights to technical change reserved.

We hereby disclaim any and all warranties, including but not limited to warranties of non-infringement, regarding circuits, descriptions and charts stated herein.

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.

XC161

16-Bit Single-Chip Microcontroller with C166SV2 Core

Volume 2 (of 2): Peripheral Units

Microcontrollers

Never stop thinking.

XC161	**Volume 2 (of 2): Peripheral Units**
Revision History:	**V2.2, 2004-01**
Previous Version:	V2.2, 2003-09 (Pre-release) V2.1, 2003-06 V2.0, 2003-03 V1.1, 2002-02 (Draft Manual) V1.0, 2001-04 (Draft Manual)

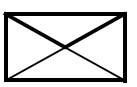
Page	Subjects (major changes since version V2.1)[1)]
14-1ff	Timer block description introduced
14-7	Phrasing improved
14-26	Figure corrected
14-27	Prescaler table reworked
14-29	Timer register description added
14-35	Phrasing improved
14-50	Prescaler table reworked
14-53	Timer register description added
17-1ff	Register names adapted

1) In order to create the current version V2.2 of this manual, the layout of several graphics and text structures has been adapted to company documentation rules. The contents have not been changed otherwise, except for the Pre-release note on page 1-2 or obvious typographical errors.

Controller Area Network (CAN): License of Robert Bosch GmbH

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

This User's Manual consists of two Volumes, "System Units" and "Peripheral Units". For your convenience this table of contents (and also the keyword index) lists both volumes, so can immediately find the reference to the desired section in the corresponding document ([1] or [2]).

1 Introduction . . . 1-1 [1]
1.1 Members of the 16-bit Microcontroller Family . . . 1-3 [1]
1.2 Summary of Basic Features . . . 1-5 [1]
1.3 Abbreviations . . . 1-9 [1]
1.4 Naming Conventions . . . 1-10 [1]

2 Architectural Overview . . . 2-1 [1]
2.1 Basic CPU Concepts and Optimizations . . . 2-2 [1]
2.1.1 High Instruction Bandwidth/Fast Execution . . . 2-4 [1]
2.1.2 Powerful Execution Units . . . 2-5 [1]
2.1.3 High Performance Branch-, Call-, and Loop-Processing . . . 2-6 [1]
2.1.4 Consistent and Optimized Instruction Formats . . . 2-7 [1]
2.1.5 Programmable Multiple Priority Interrupt System . . . 2-8 [1]
2.1.6 Interfaces to System Resources . . . 2-9 [1]
2.2 On-Chip System Resources . . . 2-10 [1]
2.3 On-Chip Peripheral Blocks . . . 2-14 [1]
2.4 Clock Generation . . . 2-29 [1]
2.5 Power Management Features . . . 2-29 [1]
2.6 On-Chip Debug Support (OCDS) . . . 2-31 [1]
2.7 Protected Bits . . . 2-32 [1]

3 Memory Organization . . . 3-1 [1]
3.1 Address Mapping . . . 3-3 [1]
3.2 Special Function Register Areas . . . 3-4 [1]
3.3 Data Memory Areas . . . 3-8 [1]
3.4 Program Memory Areas . . . 3-10 [1]
3.5 System Stack . . . 3-12 [1]
3.6 IO Areas . . . 3-13 [1]
3.7 External Memory Space . . . 3-14 [1]
3.8 Crossing Memory Boundaries . . . 3-15 [1]
3.9 The On-Chip Program Flash Module . . . 3-16 [1]
3.9.1 Flash Operating Modes . . . 3-18 [1]
3.9.2 Command Sequences . . . 3-19 [1]
3.9.3 Error Correction and Data Integrity . . . 3-25 [1]
3.9.4 Protection and Security Features . . . 3-27 [1]
3.9.5 Flash Status Information . . . 3-32 [1]
3.9.6 Operation Control and Error Handling . . . 3-35 [1]

Table of Contents **Page**

3.10 Program Memory Control . . . . . 3-37 [1]
3.10.1 Address Map . . . . . 3-38 [1]
3.10.2 Flash Memory Access . . . . . 3-39 [1]
3.10.3 IMB Control Functions . . . . . 3-41 [1]

4 Central Processing Unit (CPU) . . . . . 4-1 [1]
4.1 Components of the CPU . . . . . 4-4 [1]
4.2 Instruction Fetch and Program Flow Control . . . . . 4-5 [1]
4.2.1 Branch Detection and Branch Prediction Rules . . . . . 4-7 [1]
4.2.2 Correctly Predicted Instruction Flow . . . . . 4-7 [1]
4.2.3 Incorrectly Predicted Instruction Flow . . . . . 4-9 [1]
4.3 Instruction Processing Pipeline . . . . . 4-11 [1]
4.3.1 Pipeline Conflicts Using General Purpose Registers . . . . . 4-13 [1]
4.3.2 Pipeline Conflicts Using Indirect Addressing Modes . . . . . 4-15 [1]
4.3.3 Pipeline Conflicts Due to Memory Bandwidth . . . . . 4-17 [1]
4.3.4 Pipeline Conflicts Caused by CPU-SFR Updates . . . . . 4-20 [1]
4.4 CPU Configuration Registers . . . . . 4-26 [1]
4.5 Use of General Purpose Registers . . . . . 4-29 [1]
4.5.1 GPR Addressing Modes . . . . . 4-31 [1]
4.5.2 Context Switching . . . . . 4-33 [1]
4.6 Code Addressing . . . . . 4-37 [1]
4.7 Data Addressing . . . . . 4-39 [1]
4.7.1 Short Addressing Modes . . . . . 4-39 [1]
4.7.2 Long Addressing Modes . . . . . 4-41 [1]
4.7.3 Indirect Addressing Modes . . . . . 4-45 [1]
4.7.4 DSP Addressing Modes . . . . . 4-47 [1]
4.7.5 The System Stack . . . . . 4-53 [1]
4.8 Standard Data Processing . . . . . 4-57 [1]
4.8.1 16-bit Adder/Subtracter, Barrel Shifter, and 16-bit Logic Unit . . . . 4-61 [1]
4.8.2 Bit Manipulation Unit . . . . . 4-61 [1]
4.8.3 Multiply and Divide Unit . . . . . 4-63 [1]
4.9 DSP Data Processing (MAC Unit) . . . . . 4-65 [1]
4.9.1 Representation of Numbers and Rounding . . . . . 4-66 [1]
4.9.2 The 16-bit by 16-bit Signed/Unsigned Multiplier and Scaler . . . . . 4-67 [1]
4.9.3 Concatenation Unit . . . . . 4-67 [1]
4.9.4 One-bit Scaler . . . . . 4-67 [1]
4.9.5 The 40-bit Adder/Subtracter . . . . . 4-67 [1]
4.9.6 The Data Limiter . . . . . 4-68 [1]
4.9.7 The Accumulator Shifter . . . . . 4-68 [1]
4.9.8 The 40-bit Signed Accumulator Register . . . . . 4-69 [1]
4.9.9 The MAC Unit Status Word MSW . . . . . 4-70 [1]
4.9.10 The Repeat Counter MRW . . . . . 4-72 [1]
4.10 Constant Registers . . . . . 4-74 [1]

Table of Contents **Page**

5 Interrupt and Trap Functions . . . . . 5-1 [1]
5.1 Interrupt System Structure . . . . . 5-2 [1]
5.2 Interrupt Arbitration and Control . . . . . 5-4 [1]
5.3 Interrupt Vector Table . . . . . 5-10 [1]
5.4 Operation of the Peripheral Event Controller Channels . . . . . 5-18 [1]
5.4.1 The PEC Source and Destination Pointers . . . . . 5-22 [1]
5.4.2 PEC Transfer Control . . . . . 5-24 [1]
5.4.3 Channel Link Mode for Data Chaining . . . . . 5-26 [1]
5.4.4 PEC Interrupt Control . . . . . 5-27 [1]
5.5 Prioritization of Interrupt and PEC Service Requests . . . . . 5-29 [1]
5.6 Context Switching and Saving Status . . . . . 5-31 [1]
5.7 Interrupt Node Sharing . . . . . 5-34 [1]
5.8 External Interrupts . . . . . 5-35 [1]
5.9 OCDS Requests . . . . . 5-40 [1]
5.10 Service Request Latency . . . . . 5-41 [1]
5.11 Trap Functions . . . . . 5-43 [1]

6 General System Control Functions . . . . . 6-1 [1]
6.1 System Reset . . . . . 6-2 [1]
6.1.1 Reset Sources and Phases . . . . . 6-3 [1]
6.1.2 Status After Reset . . . . . 6-6 [1]
6.1.3 Application-Specific Initialization Routine . . . . . 6-11 [1]
6.1.4 System Startup Configuration . . . . . 6-14 [1]
6.1.5 Hardware Configuration in External Start Mode . . . . . 6-18 [1]
6.1.6 Default Configuration in Single-Chip Mode . . . . . 6-23 [1]
6.1.7 Reset Behavior Control . . . . . 6-24 [1]
6.2 Clock Generation . . . . . 6-26 [1]
6.2.1 Oscillators . . . . . 6-27 [1]
6.2.2 Clock Generation and Frequency Control . . . . . 6-30 [1]
6.2.3 Clock Distribution . . . . . 6-37 [1]
6.2.4 Oscillator Watchdog . . . . . 6-38 [1]
6.2.5 Interrupt Generation . . . . . 6-38 [1]
6.2.6 Generation of an External Clock Signal . . . . . 6-39 [1]
6.3 Central System Control Functions . . . . . 6-43 [1]
6.3.1 Status Indication . . . . . 6-45 [1]
6.3.2 Reset Source Indication . . . . . 6-46 [1]
6.3.3 Peripheral Shutdown Handshake . . . . . 6-47 [1]
6.3.4 Flexible Peripheral Management . . . . . 6-47 [1]
6.3.5 Debug System Control . . . . . 6-49 [1]
6.3.6 Register Security Mechanism . . . . . 6-51 [1]
6.4 Watchdog Timer (WDT) . . . . . 6-55 [1]
6.5 Identification Control Block . . . . . 6-60 [1]

Table of Contents **Page**

7 Parallel Ports . . . . . 7-1 [1]
7.1 Input Threshold Control . . . . . 7-2 [1]
7.2 Output Driver Control . . . . . 7-3 [1]
7.3 Alternate Port Functions . . . . . 7-8 [1]
7.4 PORT0 . . . . . 7-9 [1]
7.5 PORT1 . . . . . 7-13 [1]
7.6 Port 2 . . . . . 7-24 [1]
7.7 Port 3 . . . . . 7-29 [1]
7.8 Port 4 . . . . . 7-41 [1]
7.9 Port 5 . . . . . 7-51 [1]
7.10 Port 6 . . . . . 7-54 [1]
7.11 Port 7 . . . . . 7-65 [1]
7.12 Port 9 . . . . . 7-72 [1]
7.13 Port 20 . . . . . 7-82 [1]

8 Dedicated Pins . . . . . 8-1 [1]

9 The External Bus Controller EBC . . . . . 9-1 [1]
9.1 External Bus Signals . . . . . 9-3 [1]
9.2 Timing Principles . . . . . 9-4 [1]
9.2.1 Basic Bus Cycle Protocols . . . . . 9-4 [1]
9.2.1.1 Demultiplexed Bus . . . . . 9-5 [1]
9.2.1.2 Multiplexed Bus . . . . . 9-6 [1]
9.2.2 Bus Cycle Phases . . . . . 9-7 [1]
9.2.2.1 A Phase - $\overline{CS}$ Change Phase . . . . . 9-7 [1]
9.2.2.2 B Phase - Address Setup/ALE Phase . . . . . 9-7 [1]
9.2.2.3 C Phase - Delay Phase . . . . . 9-7 [1]
9.2.2.4 D Phase - Write Data Setup/MUX Tristate Phase . . . . . 9-7 [1]
9.2.2.5 E Phase - $\overline{RD}$/$\overline{WR}$ Command Phase . . . . . 9-7 [1]
9.2.2.6 F Phase - Address/Write Data Hold Phase . . . . . 9-8 [1]
9.2.3 Bus Cycle Examples: Fastest Access Cycles . . . . . 9-8 [1]
9.3 Functional Description . . . . . 9-10 [1]
9.3.1 Configuration Register Overview . . . . . 9-10 [1]
9.3.2 The EBC Mode Register 0 . . . . . 9-12 [1]
9.3.3 The EBC Mode Register 1 . . . . . 9-14 [1]
9.3.4 The Timing Configuration Registers TCONCSx . . . . . 9-15 [1]
9.3.5 The Function Configuration Registers FCONCSx . . . . . 9-16 [1]
9.3.6 The Address Window Selection Registers ADDRSELx . . . . . 9-18 [1]
9.3.6.1 Definition of Address Areas . . . . . 9-18 [1]
9.3.6.2 Address Window Arbitration . . . . . 9-20 [1]
9.3.7 Ready Controlled Bus Cycles . . . . . 9-21 [1]
9.3.7.1 General . . . . . 9-21 [1]
9.3.7.2 The Synchronous/Asynchronous READY . . . . . 9-22 [1]

Table of Contents **Page**

9.3.7.3 Combining the READY Function with Predefined Wait States . 9-22 [1]
9.3.8 Access Control to TwinCAN . . . . . 9-23 [1]
9.3.9 External Bus Arbitration . . . . . 9-24 [1]
9.3.9.1 Initialization of Arbitration . . . . . 9-24 [1]
9.3.9.2 Arbitration Master Scheme . . . . . 9-24 [1]
9.3.9.3 Arbitration Slave Scheme . . . . . 9-26 [1]
9.3.9.4 Bus Lock Function . . . . . 9-27 [1]
9.3.9.5 Direct Master Slave Connection . . . . . 9-27 [1]
9.3.10 Shutdown Control . . . . . 9-28 [1]
9.4 LXBus Access Control and Signal Generation . . . . . 9-29 [1]
9.5 EBC Register Table . . . . . 9-29 [1]

10 The Bootstrap Loader . . . . . 10-1 [1]
10.1 Entering the Bootstrap Loader . . . . . 10-2 [1]
10.2 Loading the Startup Code . . . . . 10-4 [1]
10.3 Exiting Bootstrap Loader Mode . . . . . 10-4 [1]
10.4 Choosing the Baudrate for the BSL . . . . . 10-5 [1]

11 Debug System . . . . . 11-1 [1]
11.1 Introduction . . . . . 11-1 [1]
11.2 Debug Interface . . . . . 11-2 [1]
11.3 OCDS Module . . . . . 11-3 [1]
11.3.1 Debug Events . . . . . 11-5 [1]
11.3.2 Debug Actions . . . . . 11-6 [1]
11.4 Cerberus . . . . . 11-7 [1]
11.4.1 Functional Overview . . . . . 11-7 [1]
11.5 Emulation Device . . . . . 11-9 [1]

12 Instruction Set Summary . . . . . 12-1 [1]

13 Device Specification . . . . . 13-1 [1]

14 The General Purpose Timer Units . . . . . 14-1 [2]
14.1 Timer Block GPT1 . . . . . 14-2 [2]
14.1.1 GPT1 Core Timer T3 Control . . . . . 14-4 [2]
14.1.2 GPT1 Core Timer T3 Operating Modes . . . . . 14-8 [2]
14.1.3 GPT1 Auxiliary Timers T2/T4 Control . . . . . 14-15 [2]
14.1.4 GPT1 Auxiliary Timers T2/T4 Operating Modes . . . . . 14-18 [2]
14.1.5 GPT1 Clock Signal Control . . . . . 14-27 [2]
14.1.6 GPT1 Timer Registers . . . . . 14-29 [2]
14.1.7 Interrupt Control for GPT1 Timers . . . . . 14-30 [2]
14.2 Timer Block GPT2 . . . . . 14-31 [2]
14.2.1 GPT2 Core Timer T6 Control . . . . . 14-33 [2]
14.2.2 GPT2 Core Timer T6 Operating Modes . . . . . 14-36 [2]

Table of Contents **Page**

14.2.3 GPT2 Auxiliary Timer T5 Control . . . . . 14-39 [2]
14.2.4 GPT2 Auxiliary Timer T5 Operating Modes . . . . . 14-41 [2]
14.2.5 GPT2 Register CAPREL Operating Modes . . . . . 14-45 [2]
14.2.6 GPT2 Clock Signal Control . . . . . 14-50 [2]
14.2.7 GPT2 Timer Registers . . . . . 14-53 [2]
14.2.8 Interrupt Control for GPT2 Timers and CAPREL . . . . . 14-54 [2]
14.3 Interfaces of the GPT Module . . . . . 14-55 [2]

15 Real Time Clock . . . . . 15-1 [2]
15.1 Defining the RTC Time Base . . . . . 15-2 [2]
15.2 RTC Run Control . . . . . 15-5 [2]
15.3 RTC Operating Modes . . . . . 15-7 [2]
15.3.1 48-bit Timer Operation . . . . . 15-10 [2]
15.3.2 System Clock Operation . . . . . 15-10 [2]
15.3.3 Cyclic Interrupt Generation . . . . . 15-11 [2]
15.4 RTC Interrupt Generation . . . . . 15-12 [2]

16 The Analog/Digital Converter . . . . . 16-1 [2]
16.1 Mode Selection . . . . . 16-3 [2]
16.1.1 Compatibility Mode . . . . . 16-3 [2]
16.1.2 Enhanced Mode . . . . . 16-5 [2]
16.2 ADC Operation . . . . . 16-8 [2]
16.2.1 Fixed Channel Conversion Modes . . . . . 16-11 [2]
16.2.2 Auto Scan Conversion Modes . . . . . 16-12 [2]
16.2.3 Wait for Read Mode . . . . . 16-13 [2]
16.2.4 Channel Injection Mode . . . . . 16-14 [2]
16.3 Automatic Calibration . . . . . 16-17 [2]
16.4 Conversion Timing Control . . . . . 16-18 [2]
16.5 A/D Converter Interrupt Control . . . . . 16-21 [2]
16.6 Interfaces of the ADC Module . . . . . 16-22 [2]

17 Capture/Compare Units . . . . . 17-1 [2]
17.1 The CAPCOM Timers . . . . . 17-4 [2]
17.2 CAPCOM Timer Interrupts . . . . . 17-9 [2]
17.3 Capture/Compare Channels . . . . . 17-10 [2]
17.4 Capture Mode Operation . . . . . 17-13 [2]
17.5 Compare Mode Operation . . . . . 17-14 [2]
17.5.1 Compare Mode 0 . . . . . 17-15 [2]
17.5.2 Compare Mode 1 . . . . . 17-15 [2]
17.5.3 Compare Mode 2 . . . . . 17-18 [2]
17.5.4 Compare Mode 3 . . . . . 17-18 [2]
17.5.5 Double-Register Compare Mode . . . . . 17-22 [2]
17.6 Compare Output Signal Generation . . . . . 17-25 [2]
17.7 Single Event Operation . . . . . 17-27 [2]

Table of Contents **Page**

17.8 Staggered and Non-Staggered Operation . . . . . 17-29 [2]
17.9 CAPCOM Interrupts . . . . . 17-34 [2]
17.10 External Input Signal Requirements . . . . . 17-36 [2]
17.11 Interfaces of the CAPCOM Units . . . . . 17-37 [2]

18 Asynchronous/Synchronous Serial Interface (ASC) . . . . . 18-1 [2]
18.1 Operational Overview . . . . . 18-3 [2]
18.2 Asynchronous Operation . . . . . 18-5 [2]
18.2.1 Asynchronous Data Frames . . . . . 18-6 [2]
18.2.2 Asynchronous Transmission . . . . . 18-9 [2]
18.2.3 Transmit FIFO Operation . . . . . 18-9 [2]
18.2.4 Asynchronous Reception . . . . . 18-12 [2]
18.2.5 Receive FIFO Operation . . . . . 18-12 [2]
18.2.6 FIFO Transparent Mode . . . . . 18-15 [2]
18.2.7 IrDA Mode . . . . . 18-16 [2]
18.2.8 RxD/TxD Data Path Selection in Asynchronous Modes . . . . . 18-17 [2]
18.3 Synchronous Operation . . . . . 18-19 [2]
18.3.1 Synchronous Transmission . . . . . 18-20 [2]
18.3.2 Synchronous Reception . . . . . 18-20 [2]
18.3.3 Synchronous Timing . . . . . 18-20 [2]
18.4 Baudrate Generation . . . . . 18-22 [2]
18.4.1 Baudrate in Asynchronous Mode . . . . . 18-22 [2]
18.4.2 Baudrate in Synchronous Mode . . . . . 18-26 [2]
18.5 Autobaud Detection . . . . . 18-27 [2]
18.5.1 General Operation . . . . . 18-27 [2]
18.5.2 Serial Frames for Autobaud Detection . . . . . 18-28 [2]
18.5.3 Baudrate Selection and Calculation . . . . . 18-29 [2]
18.5.4 Overwriting Registers on Successful Autobaud Detection . . . . . 18-33 [2]
18.6 Hardware Error Detection Capabilities . . . . . 18-34 [2]
18.7 Interrupts . . . . . 18-35 [2]
18.8 Registers . . . . . 18-39 [2]
18.9 Interfaces of the ASC Modules . . . . . 18-56 [2]

19 High-Speed Synchronous Serial Interface (SSC) . . . . . 19-1 [2]
19.1 Introduction . . . . . 19-1 [2]
19.2 Operational Overview . . . . . 19-1 [2]
19.2.1 Operating Mode Selection . . . . . 19-3 [2]
19.2.2 Full-Duplex Operation . . . . . 19-8 [2]
19.2.3 Half-Duplex Operation . . . . . 19-11 [2]
19.2.4 Continuous Transfers . . . . . 19-12 [2]
19.2.5 Baudrate Generation . . . . . 19-12 [2]
19.2.6 Error Detection Mechanisms . . . . . 19-14 [2]
19.2.7 SSC Register Summary . . . . . 19-16 [2]

Table of Contents **Page**

19.2.8 Port Configuration Requirements . . . . . 19-17 [2]
19.3 Interfaces of the SSC Modules . . . . . 19-18 [2]

20 IIC-Bus Module . . . . . 20-1 [2]
20.1 Overview . . . . . 20-2 [2]
20.2 Register Description . . . . . 20-5 [2]
20.3 IIC-Bus Module Operation . . . . . 20-12 [2]
20.3.1 Operation in Single-Master Mode . . . . . 20-12 [2]
20.3.2 Operation in Multimaster Mode . . . . . 20-12 [2]
20.3.3 Operation in Slave Mode . . . . . 20-13 [2]
20.3.4 Transmit/Receive Buffer . . . . . 20-14 [2]
20.3.5 Baud Rate Generation . . . . . 20-15 [2]
20.3.6 Notes for Programming the IIC-Bus Module . . . . . 20-16 [2]
20.4 Interrupt Request Operation . . . . . 20-17 [2]
20.5 Port Connection and Configuration . . . . . 20-19 [2]
20.6 Interfaces of the IIC-Bus Module . . . . . 20-21 [2]
20.7 IIC-Bus Overview . . . . . 20-22 [2]

21 TwinCAN Module . . . . . 21-1 [2]
21.1 Kernel Description . . . . . 21-1 [2]
21.1.1 Overview . . . . . 21-1 [2]
21.1.2 TwinCAN Control Shell . . . . . 21-4 [2]
21.1.2.1 Initialization Processing . . . . . 21-4 [2]
21.1.2.2 Interrupt Request Compressor . . . . . 21-5 [2]
21.1.2.3 Global Control and Status Logic . . . . . 21-6 [2]
21.1.3 CAN Node Control Logic . . . . . 21-7 [2]
21.1.3.1 Overview . . . . . 21-7 [2]
21.1.3.2 Timing Control Unit . . . . . 21-9 [2]
21.1.3.3 Bitstream Processor . . . . . 21-11 [2]
21.1.3.4 Error Handling Logic . . . . . 21-11 [2]
21.1.3.5 Node Interrupt Processing . . . . . 21-12 [2]
21.1.3.6 Message Interrupt Processing . . . . . 21-13 [2]
21.1.3.7 Interrupt Indication . . . . . 21-13 [2]
21.1.4 Message Handling Unit . . . . . 21-15 [2]
21.1.4.1 Arbitration and Acceptance Mask Register . . . . . 21-16 [2]
21.1.4.2 Handling of Remote and Data Frames . . . . . 21-17 [2]
21.1.4.3 Handling of Transmit Message Objects . . . . . 21-18 [2]
21.1.4.4 Handling of Receive Message Objects . . . . . 21-21 [2]
21.1.4.5 Single Data Transfer Mode . . . . . 21-23 [2]
21.1.5 CAN Message Object Buffer (FIFO) . . . . . 21-24 [2]
21.1.5.1 Buffer Access by the CAN Controller . . . . . 21-26 [2]
21.1.5.2 Buffer Access by the CPU . . . . . 21-27 [2]
21.1.6 Gateway Message Handling . . . . . 21-28 [2]

Table of Contents **Page**

21.1.6.1 Normal Gateway Mode . . . . . 21-29 [2]
21.1.6.2 Normal Gateway with FIFO Buffering . . . . . 21-33 [2]
21.1.6.3 Shared Gateway Mode . . . . . 21-36 [2]
21.1.7 Programming the TwinCAN Module . . . . . 21-40 [2]
21.1.7.1 Configuration of CAN Node A/B . . . . . 21-40 [2]
21.1.7.2 Initialization of Message Objects . . . . . 21-40 [2]
21.1.7.3 Controlling a Message Transfer . . . . . 21-41 [2]
21.1.8 Loop-Back Mode . . . . . 21-44 [2]
21.1.9 Single Transmission Try Functionality . . . . . 21-45 [2]
21.1.10 Module Clock Requirements . . . . . 21-46 [2]
21.2 TwinCAN Register Description . . . . . 21-47 [2]
21.2.1 Register Map . . . . . 21-47 [2]
21.2.2 CAN Node A/B Registers . . . . . 21-49 [2]
21.2.3 CAN Message Object Registers . . . . . 21-64 [2]
21.2.4 Global CAN Control/Status Registers . . . . . 21-80 [2]
21.3 XC161 Module Implementation Details . . . . . 21-82 [2]
21.3.1 Interfaces of the TwinCAN Module . . . . . 21-82 [2]
21.3.2 TwinCAN Module Related External Registers . . . . . 21-83 [2]
21.3.2.1 System Registers . . . . . 21-84 [2]
21.3.2.2 Port Registers . . . . . 21-85 [2]
21.3.2.3 Interrupt Registers . . . . . 21-90 [2]
21.3.3 Register Table . . . . . 21-91 [2]

22 Serial Data Link Module SDLM . . . . . 22-1 [2]
22.1 Overview . . . . . 22-1 [2]
22.2 SDLM Kernel Description . . . . . 22-2 [2]
22.2.1 J1850 Concept . . . . . 22-2 [2]
22.2.1.1 Frame Format Basics . . . . . 22-3 [2]
22.2.1.2 J1850 Bits and Symbols . . . . . 22-5 [2]
22.2.1.3 Frame Arbitration . . . . . 22-6 [2]
22.2.2 Block Diagram . . . . . 22-6 [2]
22.2.2.1 4x Mode . . . . . 22-8 [2]
22.2.2.2 Break Operation . . . . . 22-8 [2]
22.2.3 Interrupt Handling . . . . . 22-9 [2]
22.2.3.1 Message Operating Mode . . . . . 22-10 [2]
22.2.3.2 Receive Operation . . . . . 22-10 [2]
22.2.3.3 Transmit Operation . . . . . 22-11 [2]
22.2.4 In-Frame Response (IFR) Operation . . . . . 22-12 [2]
22.2.5 Block Mode . . . . . 22-13 [2]
22.2.6 Bus Access in FIFO Mode . . . . . 22-15 [2]
22.2.7 Flowcharts . . . . . 22-16 [2]
22.2.7.1 Overview . . . . . 22-16 [2]
22.2.7.2 Transmission Control . . . . . 22-17 [2]

Table of Contents **Page**

22.2.7.3 Read Operations . . . . . 22-20 [2]
22.2.8 IFR Handling . . . . . 22-22 [2]
22.2.8.1 IFR Types 1, 2 via IFRVAL . . . . . 22-22 [2]
22.3 SDLM Register Description . . . . . 22-23 [2]
22.3.1 Global Control and Timing Registers . . . . . 22-24 [2]
22.4 Control and Status Registers . . . . . 22-29 [2]
22.4.1 Transmission Related Registers . . . . . 22-39 [2]
22.4.2 Reception Related Registers . . . . . 22-43 [2]
22.5 SDLM Module Register Table . . . . . 22-50 [2]
22.6 XC161 Module Implementation Details . . . . . 22-51 [2]
22.6.1 Interfaces of the SDLM Module . . . . . 22-51 [2]
22.6.2 SDLM Module Related External Registers . . . . . 22-53 [2]
22.6.2.1 System Registers . . . . . 22-54 [2]
22.6.2.2 Port Registers . . . . . 22-55 [2]
22.6.2.3 Interrupt Registers . . . . . 22-60 [2]

23 Register Set . . . . . 23-1 [2]
23.1 PD+BUS Peripherals . . . . . 23-1 [2]
23.2 LXBUS Peripherals . . . . . 23-16 [2]

Keyword Index . . . . . i-1 [1+2]

14 The General Purpose Timer Units

The General Purpose Timer Unit blocks GPT1 and GPT2 have very flexible multifunctional timer structures which may be used for timing, event counting, pulse width measurement, pulse generation, frequency multiplication, and other purposes. They incorporate five 16-bit timers that are grouped into the two timer blocks GPT1 and GPT2. Each timer in each block may operate independently in a number of different modes such as gated timer or counter mode, or may be concatenated with another timer of the same block. Each block has alternate input/output functions and specific interrupts associated with it.

Block GPT1 contains three timers/counters: The core timer T3 and the two auxiliary timers T2 and T4. The maximum resolution is $f_{GPT}/4$. The auxiliary timers of GPT1 may optionally be configured as reload or capture registers for the core timer. These registers are listed in **Section 14.1.6**.

- $f_{GPT}/4$ maximum resolution
- 3 independent timers/counters
- Timers/counters can be concatenated
- 4 operating modes:
  - Timer Mode
  - Gated Timer Mode
  - Counter Mode
  - Incremental Interface Mode
- Reload and Capture functionality
- Separate interrupt lines

Block GPT2 contains two timers/counters: The core timer T6 and the auxiliary timer T5. The maximum resolution is $f_{GPT}/2$. An additional Capture/Reload register (CAPREL) supports capture and reload operation with extended functionality. These registers are listed in **Section 14.2.7**. The core timer T6 may be concatenated with timers of the CAPCOM units (T0, T1, T7, and T8).

The following list summarizes the features which are supported:

- $f_{GPT}/2$ maximum resolution
- 2 independent timers/counters
- Timers/counters can be concatenated
- 3 operating modes:
  - Timer Mode
  - Gated Timer Mode
  - Counter Mode
- Extended capture/reload functions via 16-bit capture/reload register CAPREL
- Separate interrupt lines

14.1 Timer Block GPT1

From a programmer's point of view, the GPT1 block is composed of a set of SFRs as summarized below. Those portions of port and direction registers which are used for alternate functions by the GPT1 block are shaded.

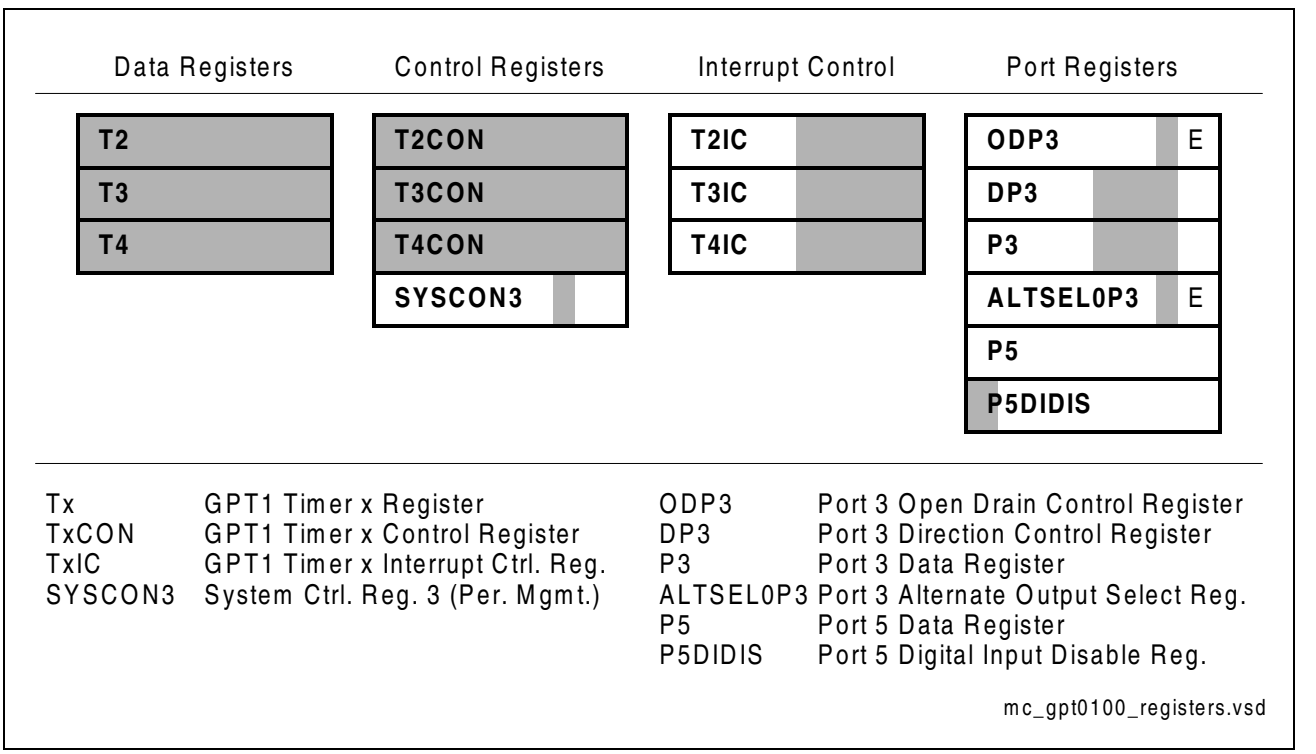


Figure 14-1 SFRs Associated with Timer Block GPT1

All three timers of block GPT1 (T2, T3, T4) can run in one of 4 basic modes: Timer Mode, Gated Timer Mode, Counter Mode, or Incremental Interface Mode. All timers can count up or down. Each timer of GPT1 is controlled by a separate control register TxCON.

Each timer has an input pin TxIN (alternate pin function) associated with it, which serves as the gate control in gated timer mode, or as the count input in counter mode. The count direction (up/down) may be programmed via software or may be dynamically altered by a signal at the External Up/Down control input TxEUD (alternate pin function). An overflow/underflow of core timer T3 is indicated by the Output Toggle Latch T3OTL, whose state may be output on the associated pin T3OUT (alternate pin function). The auxiliary timers T2 and T4 may additionally be concatenated with the core timer T3 (through T3OTL) or may be used as capture or reload registers for the core timer T3.

The current contents of each timer can be read or modified by the CPU by accessing the corresponding timer count registers T2, T3, or T4, located in the non-bitaddressable SFR space (see **Section 14.1.6**). When any of the timer registers is written to by the CPU in the state immediately preceding a timer increment, decrement, reload, or capture operation, the CPU write operation has priority in order to guarantee correct results.

The interrupts of GPT1 are controlled through the Interrupt Control Registers TxIC. These registers are not part of the GPT1 block. The input and output lines of GPT1 are connected to pins of ports P3 and P5. The control registers for the port functions are located in the respective port modules.

*Note: The timing requirements for external input signals can be found in **Section 14.1.5**, **Section 14.3** summarizes the module interface signals, including pins.*

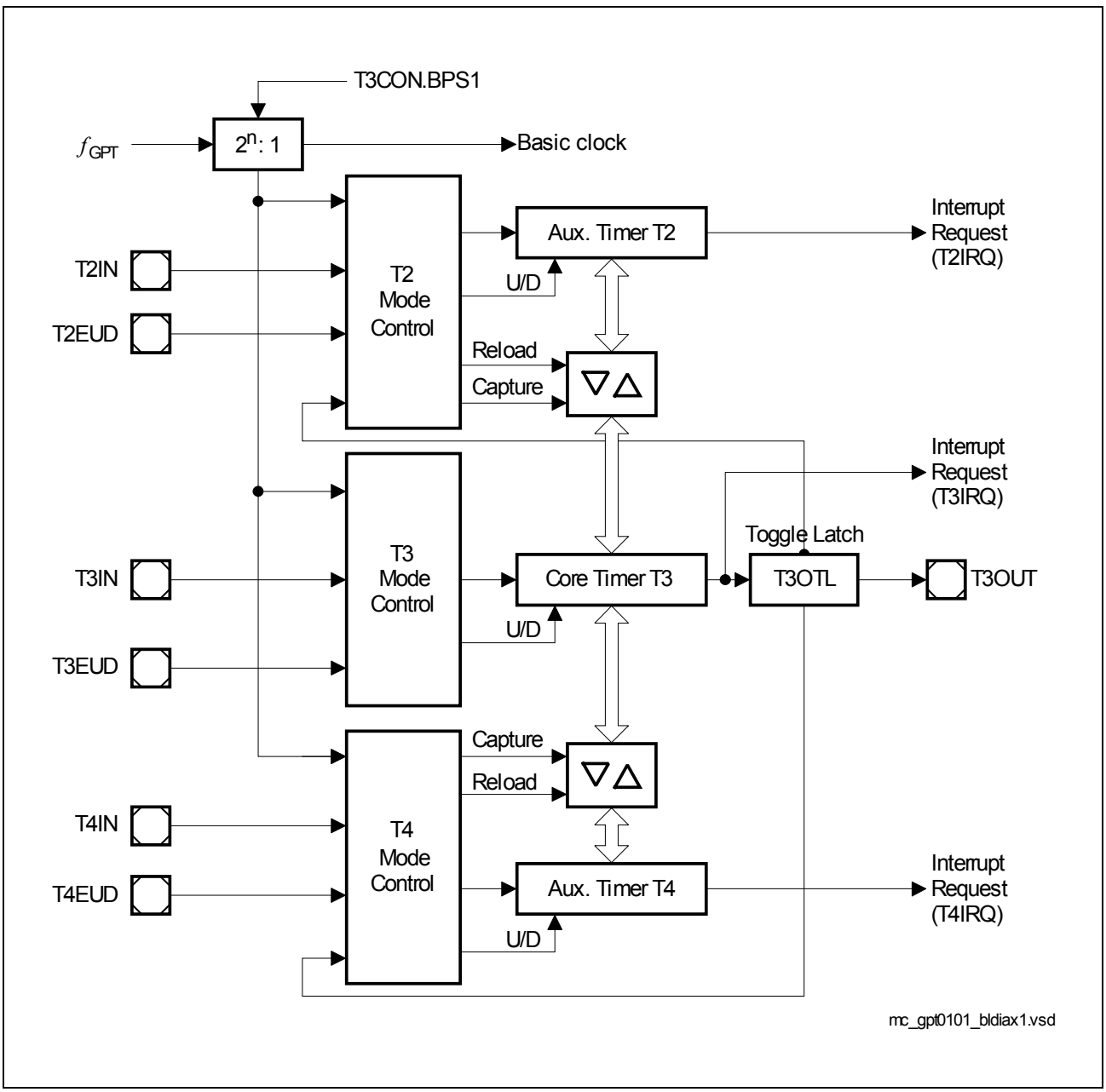


Figure 14-2 GPT1 Block Diagram (n = 2 … 5)

14.1.1 GPT1 Core Timer T3 Control

The current contents of the core timer T3 are reflected by its count register T3. This register can also be written to by the CPU, for example, to set the initial start value.

The core timer T3 is configured and controlled via its bitaddressable control register T3CON.

GPT12E_T3CON
Timer 3 Control Register **SFR ($FF42_H/A1_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T3 R DIR	T3 CH DIR	T3 ED GE	BPS1		T3 OTL	T3 OE	T3 UDE	T3 UD	T3R	T3M			T3I		
rh	rwh	rwh	rw		rwh	rw	rw	rw	rw	rw			rw		

Field	Bits	Typ	Description
T3RDIR	15	rh	**Timer T3 Rotation Direction Flag** 0 Timer T3 counts up 1 Timer T3 counts down
T3CHDIR	14	rwh	**Timer T3 Count Direction Change Flag** This bit is set each time the count direction of timer T3 changes. T3CHDIR must be cleared by SW. 0 No change of count direction was detected 1 A change of count direction was detected
T3EDGE	13	rwh	**Timer T3 Edge Detection Flag** The bit is set each time a count edge is detected. T3EDGE must be cleared by SW. 0 No count edge was detected 1 A count edge was detected
BPS1	[12:11]	rw	**GPT1 Block Prescaler Control** Selects the basic clock for block GPT1 (see also **Section 14.1.5**) 00 $f_{GPT}/8$ 01 $f_{GPT}/4$ 10 $f_{GPT}/32$ 11 $f_{GPT}/16$
T3OTL	10	rwh	**Timer T3 Overflow Toggle Latch** Toggles on each overflow/underflow of T3. Can be set or reset by software (see separate description)

Field	Bits	Typ	Description
T3OE	9	rw	**Overflow/Underflow Output Enable** 0 Alternate Output Function Disabled 1 State of T3 toggle latch is output on pin T3OUT
T3UDE	8	rw	**Timer T3 External Up/Down Enable** 0 Input T3EUD is disconnected 1 Direction influenced by input T3EUD[1)]
T3UD	7	rw	**Timer T3 Up/Down Control**[1)]
T3R	6	rw	**Timer T3 Run Bit** 0 Timer T3 stops 1 Timer T3 runs
T3M	[5:3]	rw	**Timer T3 Mode Control** (Basic Operating Mode) 000 Timer Mode 001 Counter Mode 010 Gated Timer Mode with gate active low 011 Gated Timer Mode with gate active high 100 Reserved. Do not use this combination. 101 Reserved. Do not use this combination. 110 Incremental Interface Mode (Rotation Detection Mode) 111 Incremental Interface Mode (Edge Detection Mode)
T3I	[2:0]	rw	**Timer T3 Input Parameter Selection** Depends on the operating mode, see respective sections for encoding: **Table 14-7** for Timer Mode and Gated Timer Mode **Table 14-2** for Counter Mode **Table 14-3** for Incremental Interface Mode

1) See **Table 14-1** for encoding of bits T3UD and T3EUD.

Timer T3 Run Control

The core timer T3 can be started or stopped by software through bit T3R (Timer T3 Run Bit). This bit is relevant in all operating modes of T3. Setting bit T3R will start the timer, clearing bit T3R stops the timer.

In gated timer mode, the timer will only run if T3R = 1 and the gate is active (high or low, as programmed).

Note: When bit T2RC or T4RC in timer control register T2CON or T4CON is set, bit T3R will also control (start and stop) the auxiliary timer(s) T2 and/or T4.

Count Direction Control

The count direction of the GPT1 timers (core timer and auxiliary timers) can be controlled either by software or by the external input pin TxEUD (Timer Tx External Up/Down Control Input). These options are selected by bits TxUD and TxUDE in the respective control register TxCON. When the up/down control is provided by software (bit TxUDE = 0), the count direction can be altered by setting or clearing bit TxUD. When bit TxUDE = 1, pin TxEUD is selected to be the controlling source of the count direction. However, bit TxUD can still be used to reverse the actual count direction, as shown in **Table 14-1**. The count direction can be changed regardless of whether or not the timer is running.

Note: When pin TxEUD is used as external count direction control input, it must be configured as input (its corresponding direction control bit must be cleared).

Table 14-1 GPT1 Timer Count Direction Control

Pin TxEUD	Bit TxUDE	Bit TxUD	Count Direction	Bit TxRDIR
X	0	0	Count Up	0
X	0	1	Count Down	1
0	1	0	Count Up	0
1	1	0	Count Down	1
0	1	1	Count Down	1
1	1	1	Count Up	0

Timer 3 Output Toggle Latch

The overflow/underflow signal of timer T3 is connected to a block named ‘Toggle Latch’, shown in the timer mode diagrams. **Figure 14-3** illustrates the details of this block. An overflow or underflow of T3 will clock two latches: The first latch represents bit T3OTL in control register T3CON. The second latch is an internal latch toggled by T3OTL’s output. Both latch outputs are connected to the input control blocks of the auxiliary timers T2 and T4. The output level of the shadow latch will match the output level of T3OTL, but is delayed by one clock cycle. When the T3OTL value changes, this will result in a temporarily different output level from T3OTL and the shadow latch, which can trigger the selected count event in T2 and/or T4.

When software writes to T3OTL, both latches are set or cleared simultaneously. In this case, both signals to the auxiliary timers carry the same level and no edge will be detected. Bit T3OE (overflow/underflow output enable) in register T3CON enables the state of T3OTL to be monitored via an external pin T3OUT. When T3OTL is linked to an external port pin (must be configured as output), T3OUT can be used to control external HW. If T3OE = 1, pin T3OUT outputs the state of T3OTL. If T3OE = 0, pin T3OUT outputs a high level (as long as the T3OUT alternate function is selected for the port pin).

The trigger signals can serve as an input for the counter function or as a trigger source for the reload function of the auxiliary timers T2 and T4.

As can be seen from **Figure 14-3**, when latch T3OTL is modified by software to determine the state of the output line, also the internal shadow latch is set or cleared accordingly. Therefore, no trigger condition is detected by T2/T4 in this case.

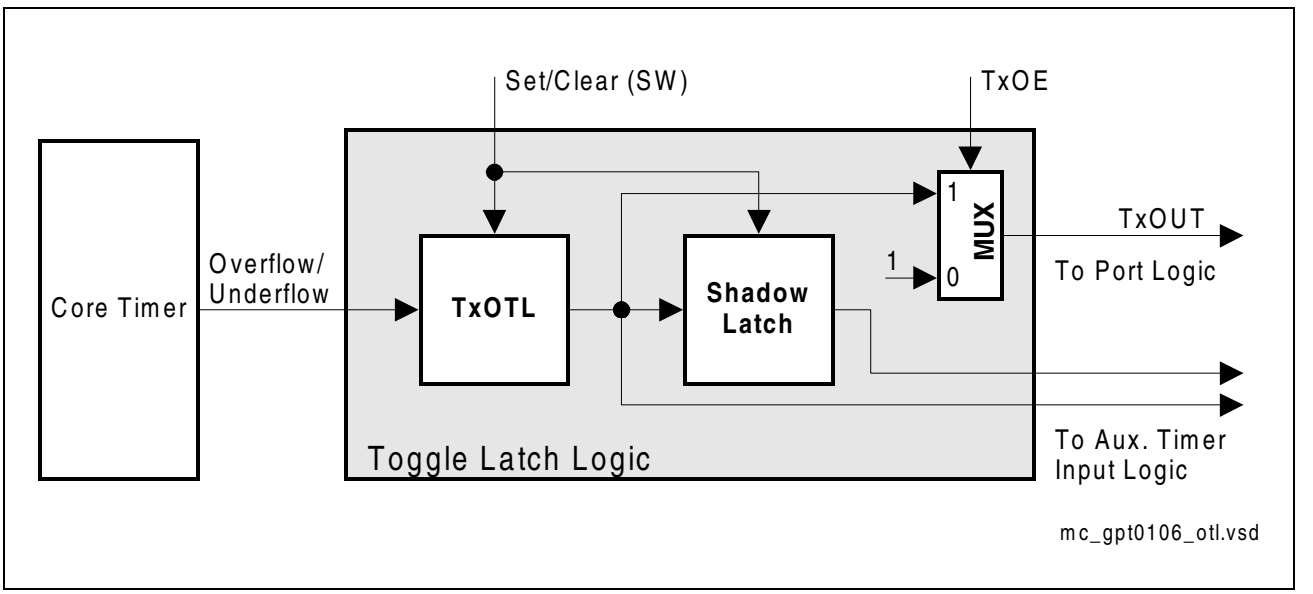


Figure 14-3 Block Diagram of the Toggle Latch Logic of Core Timer T3

14.1.2 GPT1 Core Timer T3 Operating Modes

Timer 3 in Timer Mode

Timer mode for the core timer T3 is selected by setting bitfield T3M in register T3CON to 000_B. In timer mode, T3 is clocked with the module's input clock f_{GPT} divided by two programmable prescalers controlled by bitfields BPS1 and T3I in register T3CON. Please see **Section 14.1.5** for details on the input clock options.

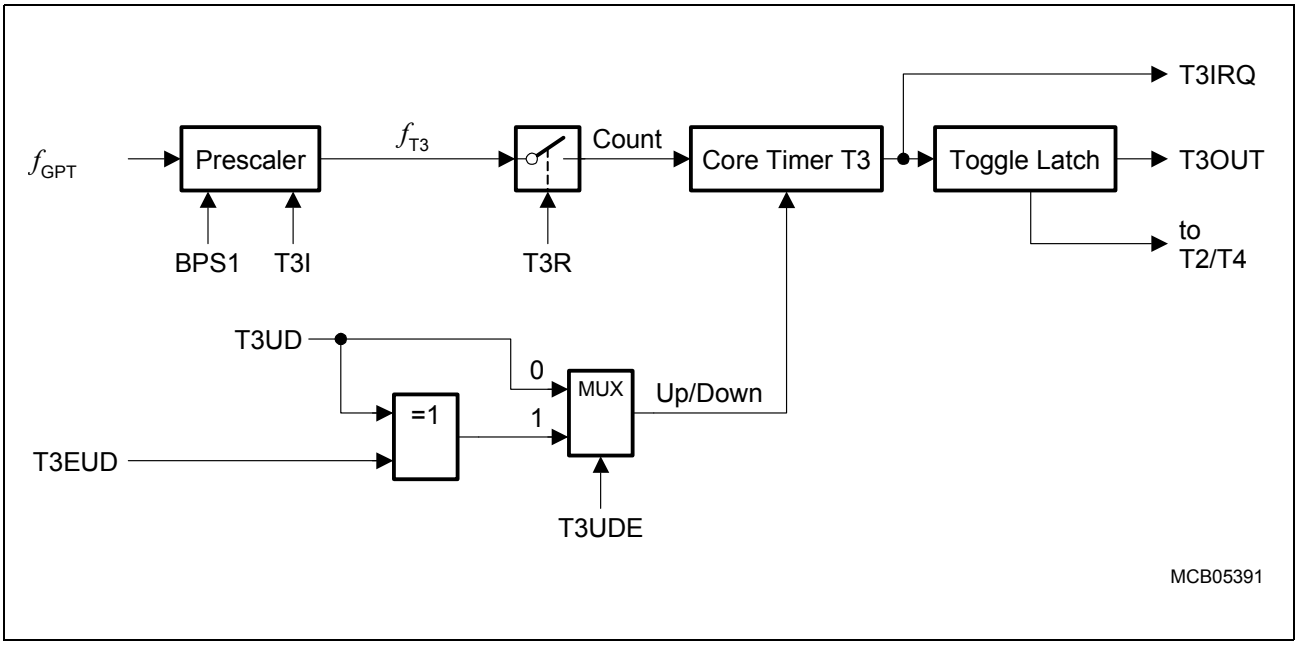


Figure 14-4 Block Diagram of Core Timer T3 in Timer Mode

Gated Timer Mode

Gated timer mode for the core timer T3 is selected by setting bitfield T3M in register T3CON to 010_B or 011_B. Bit T3M.0 (T3CON.3) selects the active level of the gate input. The same options for the input frequency are available in gated timer mode as in timer mode (see **Section 14.1.5**). However, the input clock to the timer in this mode is gated by the external input pin T3IN (Timer T3 External Input).
To enable this operation, the associated pin T3IN must be configured as input, that is, the corresponding direction control bit must contain 0.

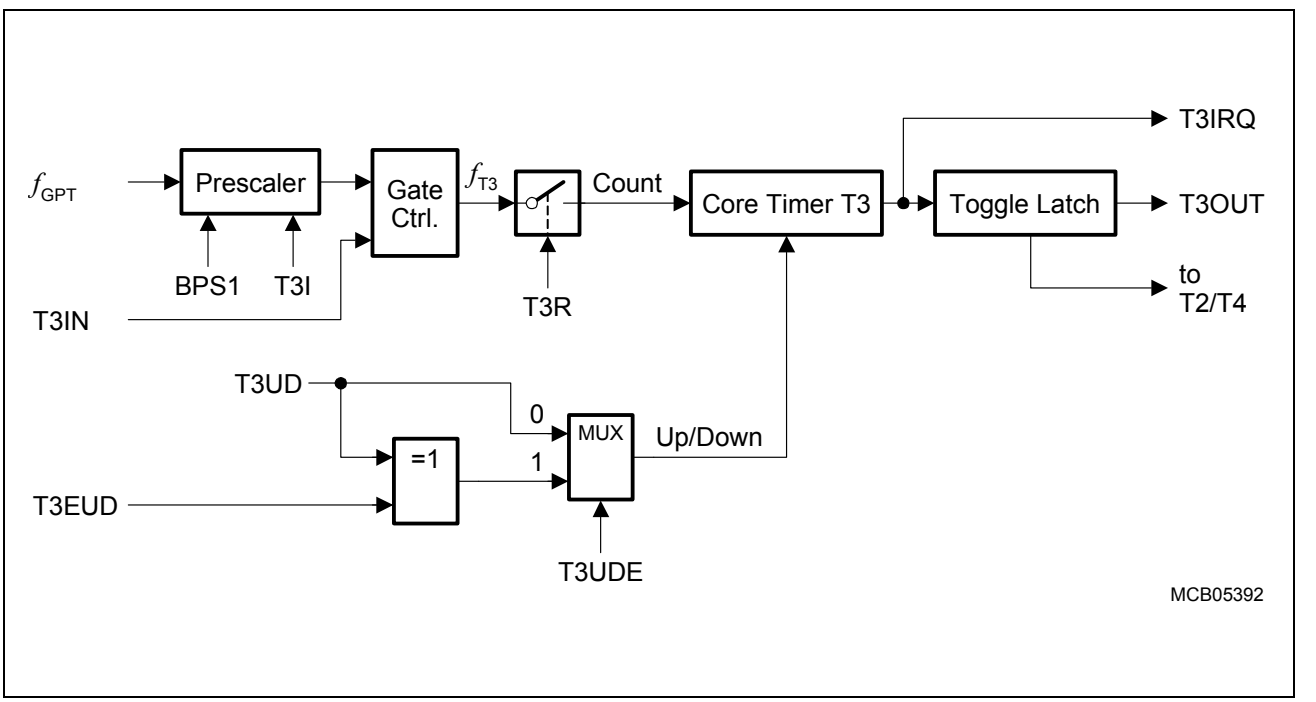


Figure 14-5 Block Diagram of Core Timer T3 in Gated Timer Mode

If T3M = 010_B, the timer is enabled when T3IN shows a low level. A high level at this line stops the timer. If T3M = 011_B, line T3IN must have a high level in order to enable the timer. Additionally, the timer can be turned on or off by software using bit T3R. The timer will only run if T3R is 1 and the gate is active. It will stop if either T3R is 0 or the gate is inactive.

Note: A transition of the gate signal at pin T3IN does not cause an interrupt request.

Counter Mode

Counter Mode for the core timer T3 is selected by setting bitfield T3M in register T3CON to 001_B. In counter mode, timer T3 is clocked by a transition at the external input pin T3IN. The event causing an increment or decrement of the timer can be a positive, a negative, or both a positive and a negative transition at this line. Bitfield T3I in control register T3CON selects the triggering transition (see **Table 14-2**).

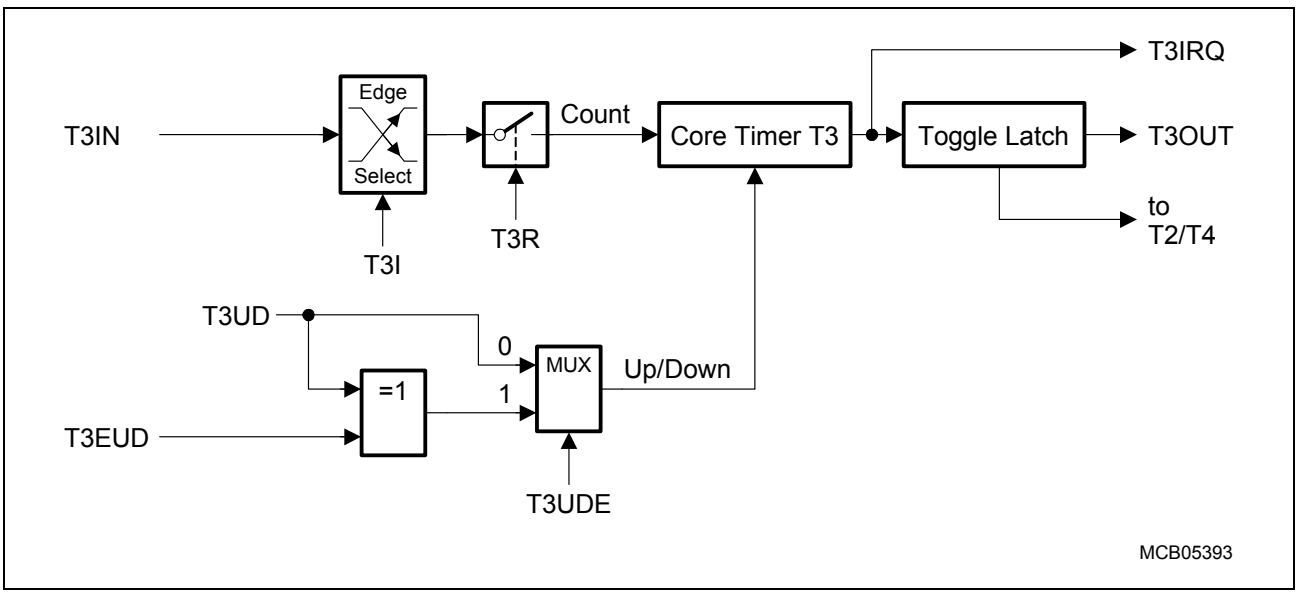


Figure 14-6 Block Diagram of Core Timer T3 in Counter Mode

Table 14-2 GPT1 Core Timer T3 (Counter Mode) Input Edge Selection

T3I	Triggering Edge for Counter Increment/Decrement
0 0 0	None. Counter T3 is disabled
0 0 1	Positive transition (rising edge) on T3IN
0 1 0	Negative transition (falling edge) on T3IN
0 1 1	Any transition (rising or falling edge) on T3IN
1 X X	Reserved. Do not use this combination

For counter mode operation, pin T3IN must be configured as input (the respective direction control bit DPx.y must be 0). The maximum input frequency allowed in counter mode depends on the selected prescaler value. To ensure that a transition of the count input signal applied to T3IN is recognized correctly, its level must be held high or low for a minimum number of module clock cycles before it changes. This information can be found in **Section 14.1.5**.

Incremental Interface Mode

Incremental interface mode for the core timer T3 is selected by setting bitfield T3M in register T3CON to 110_B or 111_B. In incremental interface mode, the two inputs associated with core timer T3 (T3IN, T3EUD) are used to interface to an incremental encoder. T3 is clocked by each transition on one or both of the external input pins to provide 2-fold or 4-fold resolution of the encoder input.

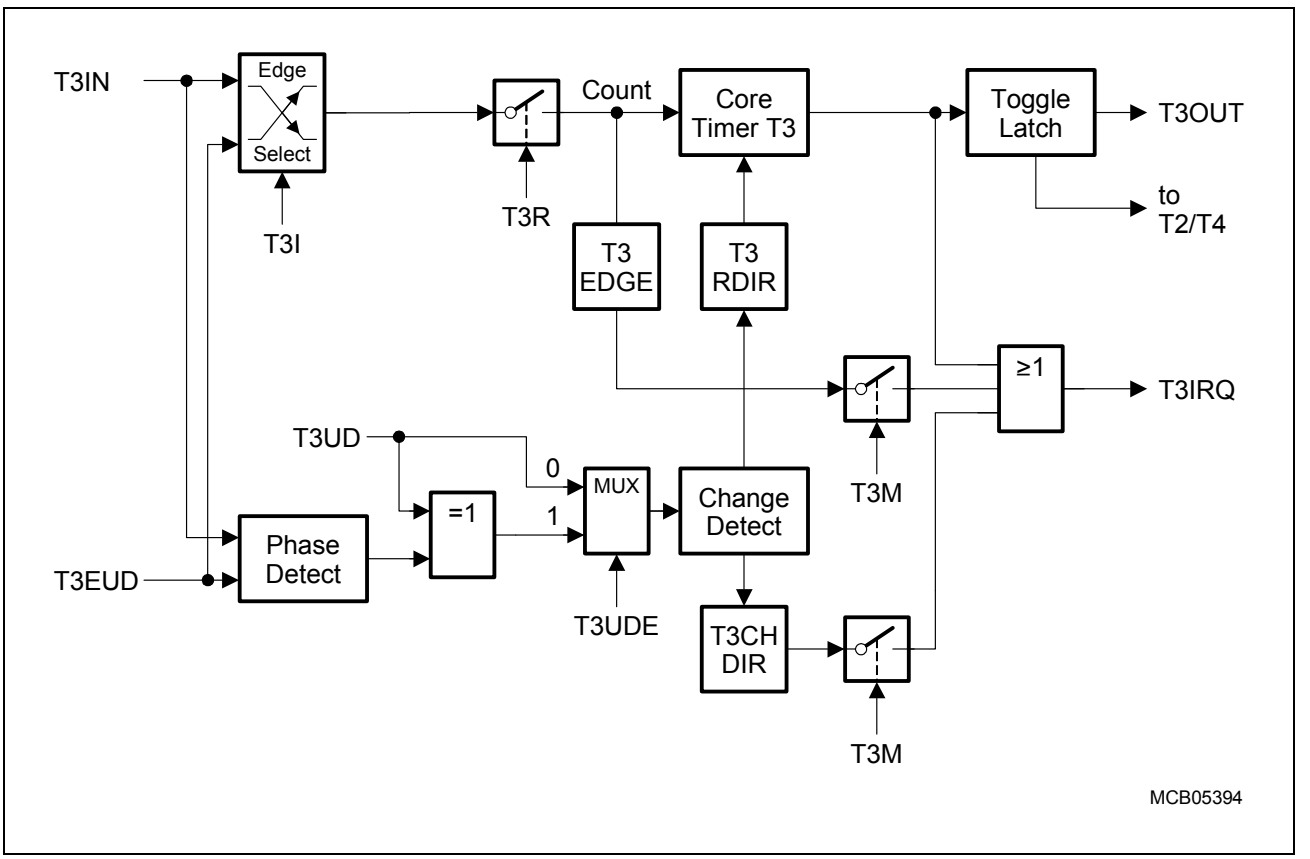


Figure 14-7 Block Diagram of Core Timer T3 in Incremental Interface Mode

Bitfield T3I in control register T3CON selects the triggering transitions (see **Table 14-3**). The sequence of the transitions of the two input signals is evaluated and generates count pulses as well as the direction signal. So T3 is modified automatically according to the speed and the direction of the incremental encoder and, therefore, its contents always represent the encoder's current position.

The interrupt request (T3IRQ) generation mode can be selected: In Rotation Detection Mode (T3M = 110_B), an interrupt request is generated each time the count direction of T3 changes. In Edge Detection Mode (T3M = 111_B), an interrupt request is generated each time a count edge for T3 is detected. Count direction, changes in the count direction, and count requests are monitored by status bits T3RDIR, T3CHDIR, and T3EDGE in register T3CON.

Table 14-3 Core Timer T3 (Incremental Interface Mode) Input Edge Selection

T3I	Triggering Edge for Counter Increment/Decrement
0 0 0	None. Counter T3 stops.
0 0 1	Any transition (rising or falling edge) on T3IN.
0 1 0	Any transition (rising or falling edge) on T3EUD.
0 1 1	Any transition (rising or falling edge) on any T3 input (T3IN or T3EUD).
1 X X	Reserved. Do not use this combination.

The incremental encoder can be connected directly to the XC161 without external interface logic. In a standard system, however, comparators will be employed to convert the encoder's differential outputs (such as A, $\overline{A}$) to digital signals (such as A). This greatly increases noise immunity.

Note: The third encoder output T0, which indicates the mechanical zero position, may be connected to an external interrupt input and trigger a reset of timer T3 (for example via PEC transfer from ZEROS).

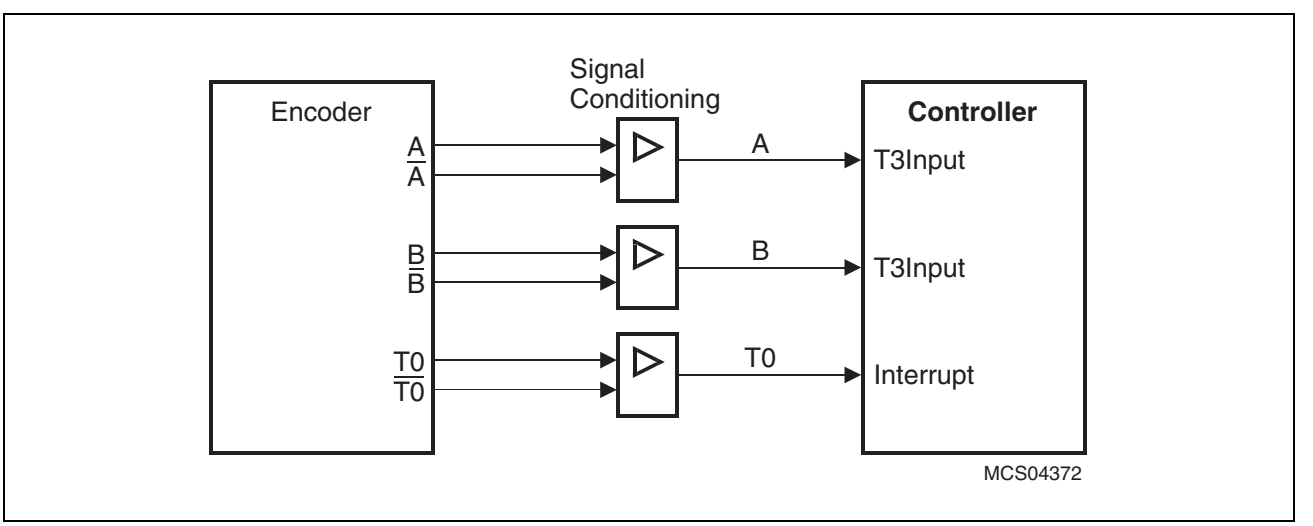


Figure 14-8 Connection of the Encoder to the XC161

For incremental interface operation, the following conditions must be met:

- Bitfield T3M must be 110_B or 111_B.
- Both pins T3IN and T3EUD must be configured as input, i.e. the respective direction control bits must be 0.
- Bit T3UDE must be 1 to enable automatic external direction control.

The maximum count frequency allowed in incremental interface mode depends on the selected prescaler value. To ensure that a transition of any input signal is recognized correctly, its level must be held high or low for a minimum number of module clock cycles before it changes. This information can be found in **Section 14.1.5**.

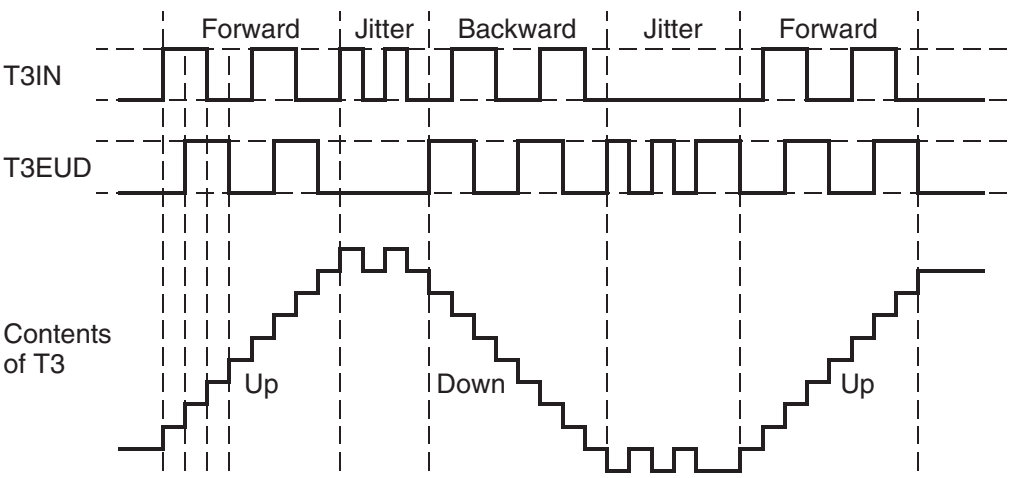
As in incremental interface mode two input signals with a 90° phase shift are evaluated, their maximum input frequency can be half the maximum count frequency.

In incremental interface mode, the count direction is automatically derived from the sequence in which the input signals change, which corresponds to the rotation direction of the connected sensor. **Table 14-4** summarizes the possible combinations.

Table 14-4 GPT1 Core Timer T3 (Incremental Interface Mode) Count Direction

Level on Respective other Input	T3IN Input		T3EUD Input	
	Rising ↗	Falling ↘	Rising ↗	Falling ↘
High	Down	Up	Up	Down
Low	Up	Down	Down	Up

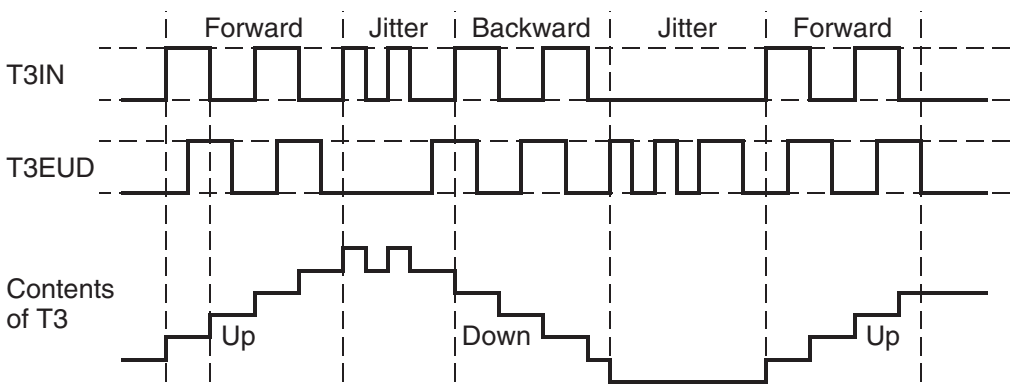
Figure 14-9 and **Figure 14-10** give examples of T3's operation, visualizing count signal generation and direction control. They also show how input jitter is compensated, which might occur if the sensor rests near to one of its switching points.



Note: This example shows the timer behaviour assuming that T3 counts upon any transition on input, i.e. T3I = '011 $_B$'.

MCT04373

Figure 14-9 Evaluation of Incremental Encoder Signals, 2 Count Inputs



Note: This example shows the timer behaviour assuming that T3 counts upon any transition on input T3IN, i.e. T3I = '001 $_B$'.

MCT04374

Figure 14-10 Evaluation of Incremental Encoder Signals, 1 Count Input

*Note: Timer T3 operating in incremental interface mode automatically provides information on the sensor's current position. Dynamic information (speed, acceleration, deceleration) may be obtained by measuring the incoming signal periods. This is facilitated by an additional special capture mode for timer T5 (see **Section 14.2.5**).*

14.1.3 GPT1 Auxiliary Timers T2/T4 Control

Auxiliary timers T2 and T4 have exactly the same functionality. They can be configured for timer mode, gated timer mode, counter mode, or incremental interface mode with the same options for the timer frequencies and the count signal as the core timer T3. In addition to these 4 counting modes, the auxiliary timers can be concatenated with the core timer, or they may be used as reload or capture registers in conjunction with the core timer. The start/stop function of the auxiliary timers can be remotely controlled by the T3 run control bit. Several timers may thus be controlled synchronously.

The current contents of an auxiliary timer are reflected by its count register T2 or T4, respectively. These registers can also be written to by the CPU, for example, to set the initial start value.

The individual configurations for timers T2 and T4 are determined by their bitaddressable control registers T2CON and T4CON, which are organized identically. Note that functions which are present in all 3 timers of block GPT1 are controlled in the same bit positions and in the same manner in each of the specific control registers.

Note: The auxiliary timers have no output toggle latch and no alternate output function.

GPT12E_T2CON
Timer 2 Control Register **SFR ($FF40_H/A0_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T2RDIR	T2CHDIR	T2EDGE	T2IRDIS	-	-	T2RC	T2UDE	T2UD	T2R	T2M			T2I		
rh	rwh	rwh	rw	-	-	rw	rw	rw	rw	rw			rw		

GPT12E_T4CON
Timer 4 Control Register **SFR ($FF44_H/A2_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T4RDIR	T4CHDIR	T4EDGE	T4IRDIS	-	-	T4RC	T4UDE	T4UD	T4R	T4M			T4I		
rh	rwh	rwh	rw	-	-	rw	rw	rw	rw	rw			rw		

Field	Bits	Typ	Description
TxRDIR	15	rh	**Timer Tx Rotation Direction** 0 Timer x counts up 1 Timer x counts down

Field	Bits	Typ	Description
TxCHDIR	14	rwh	**Timer Tx Count Direction Change** This bit is set each time the count direction of timer Tx changes. TxCHDIR must be cleared by SW. 0 No change in count direction was detected 1 A change in count direction was detected
TxEDGE	13	rwh	**Timer Tx Edge Detection** The bit is set each time a count edge is detected. TxEDGE must be cleared by SW. 0 No count edge was detected 1 A count edge was detected
TxIRDIS	12	rw	**Timer Tx Interrupt Request Disable** 0 Interrupt generation for TxCHDIR and TxEDGE interrupts in Incremental Interface Mode is enabled 1 Interrupt generation for TxCHDIR and TxEDGE interrupts in Incremental Interface Mode is disabled
TxRC	9	rw	**Timer Tx Remote Control** 0 Timer Tx is controlled by its own run bit TxR 1 Timer Tx is controlled by the run bit T3R of core timer 3, not by bit TxR
TxUDE	8	rw	**Timer Tx External Up/Down Enable** 0 Input TxEUD is disconnected 1 Direction influenced by input TxEUD[1)]
TxUD	7	rw	**Timer Tx Up/Down Control[1)]**
TxR	6	rw	**Timer Tx Run Bit** 0 Timer Tx stops 1 Timer Tx runs *Note: This bit only controls timer Tx if bit TxRC = 0.*
TxM	[5:3]	rw	**Timer Tx Mode Control** (Basic Operating Mode) 000 Timer Mode 001 Counter Mode 010 Gated Timer Mode with gate active low 011 Gated Timer Mode with gate active high 100 Reload Mode 101 Capture Mode 110 Incremental Interface Mode (Rotation Detect.) 111 Incremental Interface Mode (Edge Detection)

Field	Bits	Typ	Description
TxI	[2:0]	rw	**Timer Tx Input Parameter Selection** Depends on the operating mode, see respective sections for encoding: **Table 14-7** for Timer Mode and Gated Timer Mode **Table 14-2** for Counter Mode **Table 14-3** for Incremental Interface Mode

1) See **Table 14-1** for encoding of bits TxUD and TxEUD.

Timer T2/T4 Run Control

Each of the auxiliary timers T2 and T4 can be started or stopped by software in two different ways:

- Through the associated timer run bit (T2R or T4R). In this case it is required that the respective control bit TxRC = 0.
- Through the core timer's run bit (T3R). In this case the respective remote control bit must be set (TxRC = 1).

The selected run bit is relevant in all operating modes of T2/T4. Setting the bit will start the timer, clearing the bit stops the timer.

In gated timer mode, the timer will only run if the selected run bit is set and the gate is active (high or low, as programmed).

Note: If remote control is selected T3R will start/stop timer T3 and the selected auxiliary timer(s) synchronously.

Count Direction Control

The count direction of the GPT1 timers (core timer and auxiliary timers) is controlled in the same way, either by software or by the external input pin TxEUD. Please refer to the description in **Table 14-1**.

Note: When pin TxEUD is used as external count direction control input, it must be configured as input (its corresponding direction control bit must be cleared).

14.1.4 GPT1 Auxiliary Timers T2/T4 Operating Modes

The operation of the auxiliary timers in the basic operating modes is almost identical with the core timer's operation, with very few exceptions. Additionally, some combined operating modes can be selected.

Timers T2 and T4 in Timer Mode

Timer mode for an auxiliary timer Tx is selected by setting its bitfield TxM in register TxCON to 000_B.

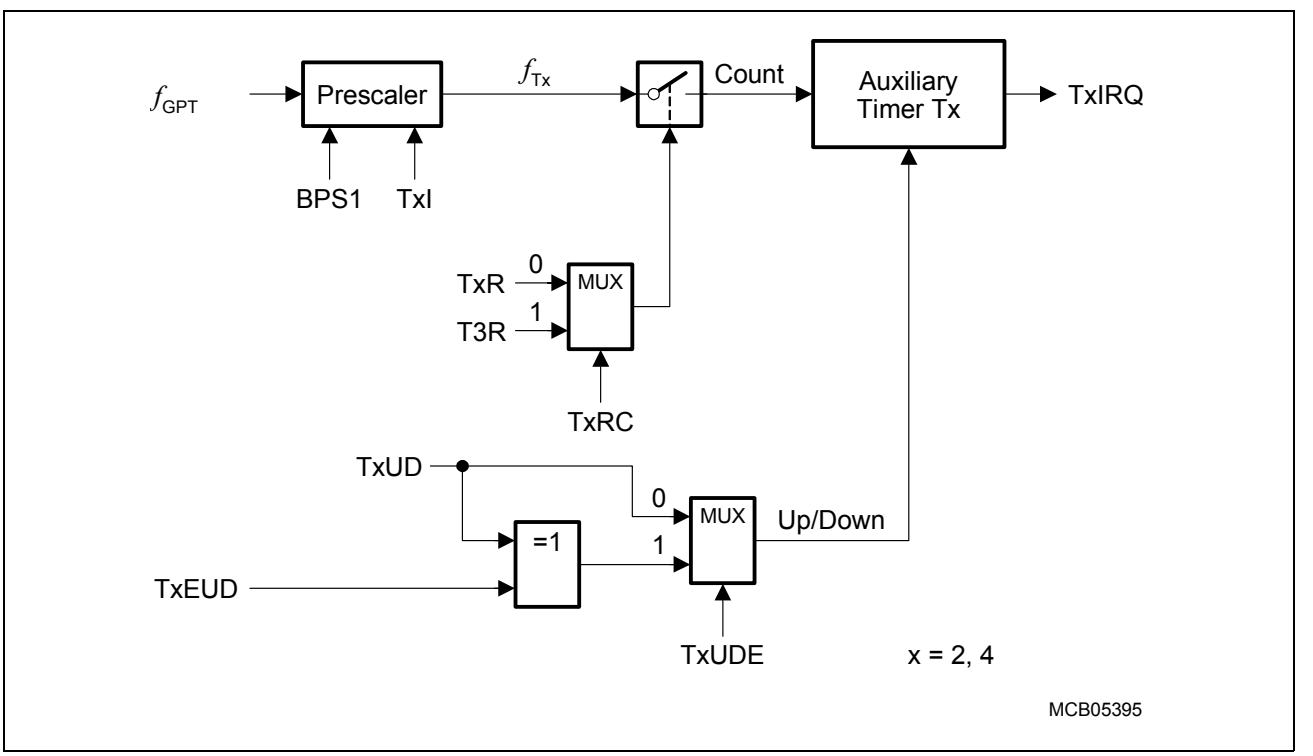


Figure 14-11 Block Diagram of an Auxiliary Timer in Timer Mode

Timers T2 and T4 in Gated Timer Mode

Gated timer mode for an auxiliary timer Tx is selected by setting bitfield TxM in register TxCON to 010_B or 011_B. Bit TxM.0 (TxCON.3) selects the active level of the gate input.

Note: A transition of the gate signal at line TxIN does not cause an interrupt request.

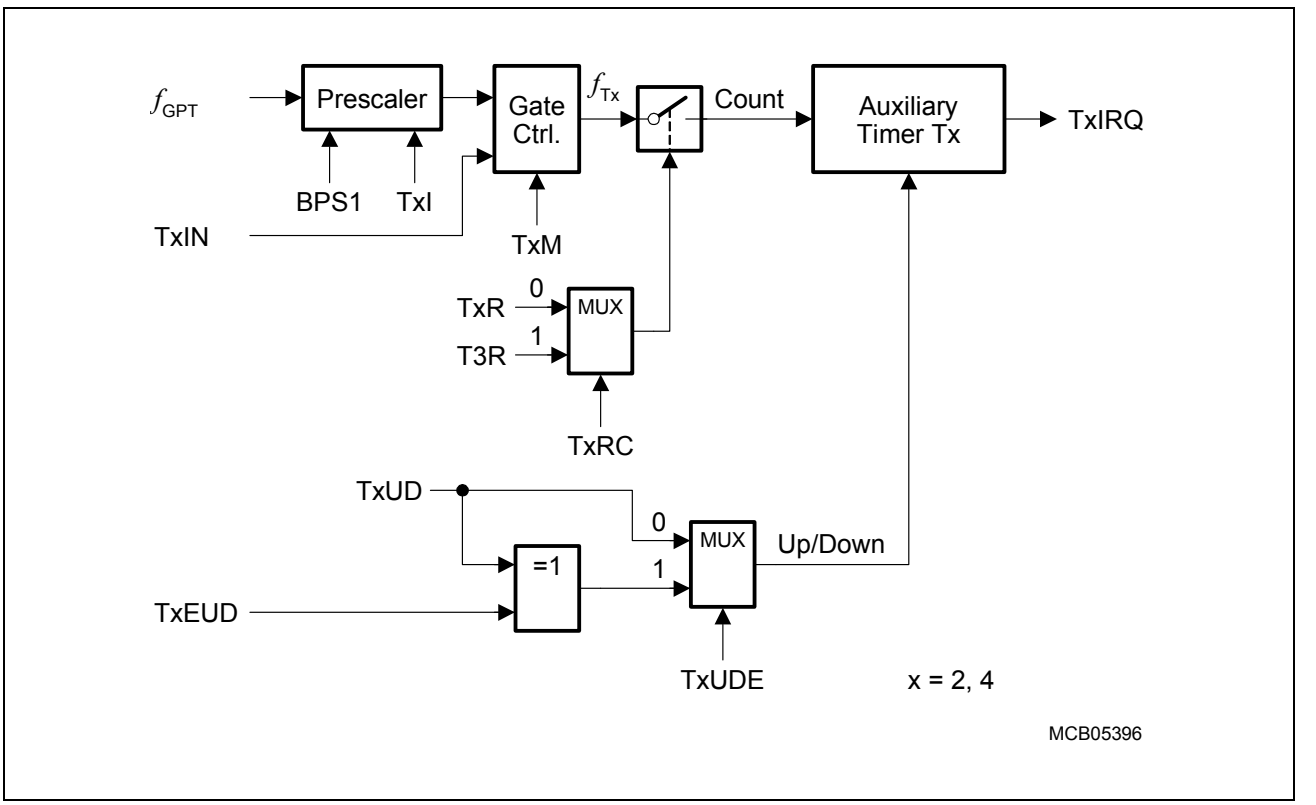


Figure 14-12 Block Diagram of an Auxiliary Timer in Gated Timer Mode

Note: There is no output toggle latch for T2 and T4.
Start/stop of an auxiliary timer can be controlled locally or remotely.

Timers T2 and T4 in Counter Mode

Counter mode for an auxiliary timer Tx is selected by setting bitfield TxM in register TxCON to 001_B. In counter mode, an auxiliary timer can be clocked either by a transition at its external input line TxIN, or by a transition of timer T3's toggle latch T3OTL. The event causing an increment or decrement of a timer can be a positive, a negative, or both a positive and a negative transition at either the respective input pin or at the toggle latch. Bitfield TxI in control register TxCON selects the triggering transition (see **Table 14-5**).

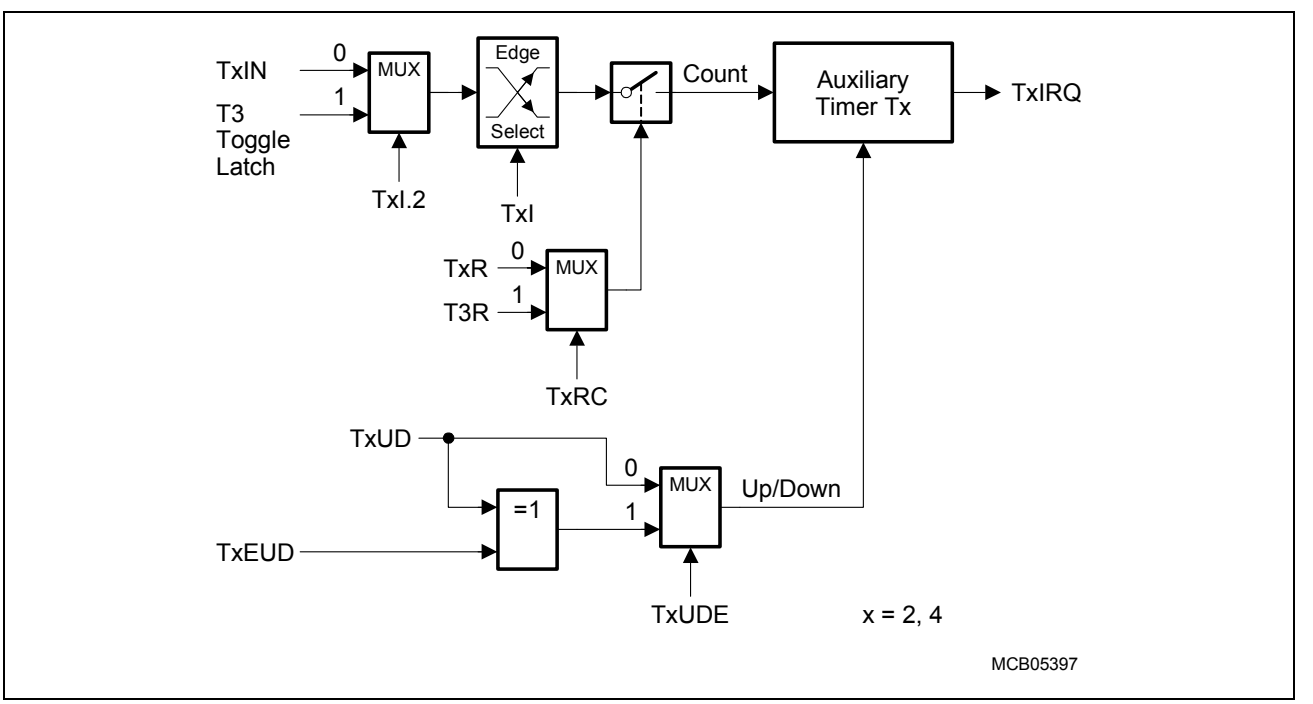


Figure 14-13 Block Diagram of an Auxiliary Timer in Counter Mode

Table 14-5 GPT1 Auxiliary Timer (Counter Mode) Input Edge Selection

T2I/T4I	Triggering Edge for Counter Increment/Decrement
X 0 0	None. Counter Tx is disabled
0 0 1	Positive transition (rising edge) on TxIN
0 1 0	Negative transition (falling edge) on TxIN
0 1 1	Any transition (rising or falling edge) on TxIN
1 0 1	Positive transition (rising edge) of T3 toggle latch T3OTL
1 1 0	Negative transition (falling edge) of T3 toggle latch T3OTL
1 1 1	Any transition (rising or falling edge) of T3 toggle latch T3OTL

Note: Only state transitions of T3OTL which are caused by the overflows/underflows of T3 will trigger the counter function of T2/T4. Modifications of T3OTL via software will NOT trigger the counter function of T2/T4.

For counter operation, pin TxIN must be configured as input (the respective direction control bit DPx.y must be 0). The maximum input frequency allowed in counter mode depends on the selected prescaler value. To ensure that a transition of the count input signal applied to TxIN is recognized correctly, its level must be held high or low for a minimum number of module clock cycles before it changes. This information can be found in **Section 14.1.5**.

Timers T2 and T4 in Incremental Interface Mode

Incremental interface mode for an auxiliary timer Tx is selected by setting bitfield TxM in the respective register TxCON to 110_B or 111_B. In incremental interface mode, the two inputs associated with an auxiliary timer Tx (TxIN, TxEUD) are used to interface to an incremental encoder. Tx is clocked by each transition on one or both of the external input pins to provide 2-fold or 4-fold resolution of the encoder input.

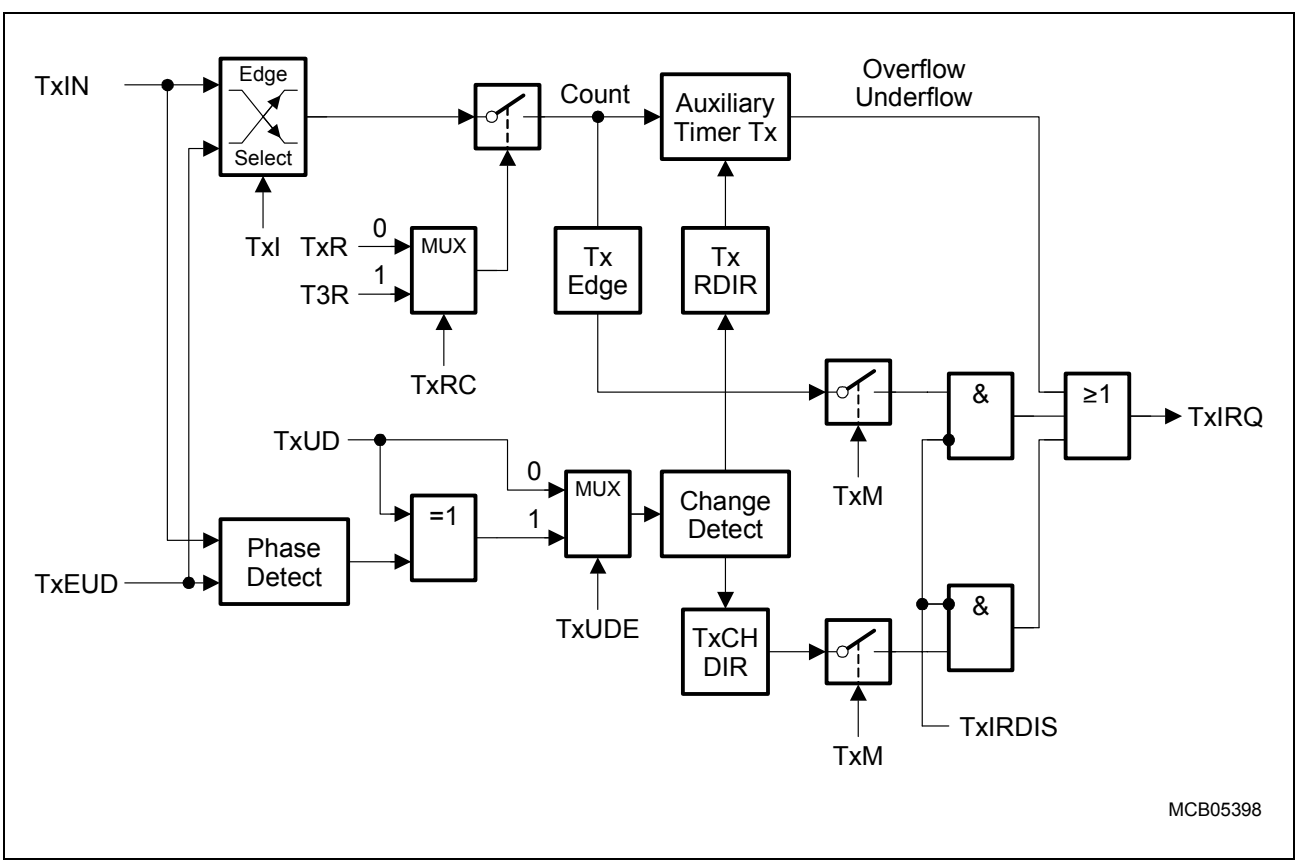


Figure 14-14 Block Diagram of an Auxiliary Timer in Incremental Interface Mode

The operation of the auxiliary timers T2 and T4 in incremental interface mode and the interrupt generation are the same as described for the core timer T3. The descriptions, figures and tables apply accordingly.

Timer Concatenation

Using the toggle bit T3OTL as a clock source for an auxiliary timer in counter mode concatenates the core timer T3 with the respective auxiliary timer. This concatenation forms either a 32-bit or a 33-bit timer/counter, depending on which transition of T3OTL is selected to clock the auxiliary timer.

- **32-bit Timer/Counter:** If both a positive and a negative transition of T3OTL are used to clock the auxiliary timer, this timer is clocked on every overflow/underflow of the core timer T3. Thus, the two timers form a 32-bit timer.
- **33-bit Timer/Counter:** If either a positive or a negative transition of T3OTL is selected to clock the auxiliary timer, this timer is clocked on every second overflow/underflow of the core timer T3. This configuration forms a 33-bit timer (16-bit core timer + T3OTL + 16-bit auxiliary timer).
  As long as bit T3OTL is not modified by software, it represents the state of the internal toggle latch, and can be regarded as part of the 33-bit timer.

The count directions of the two concatenated timers are not required to be the same. This offers a wide variety of different configurations.

T3, which represents the low-order part of the concatenated timer, can operate in timer mode, gated timer mode or counter mode in this case.

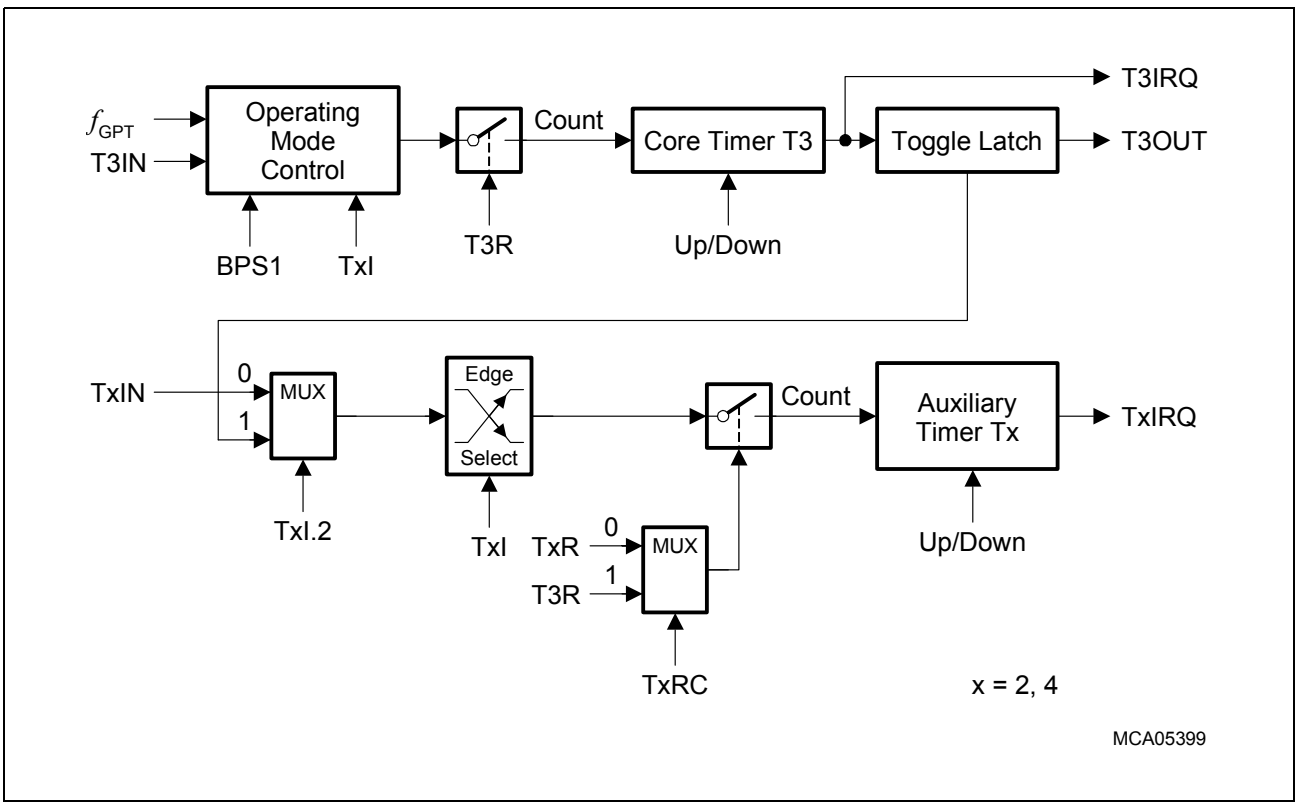


Figure 14-15 Concatenation of Core Timer T3 and an Auxiliary Timer

Auxiliary Timer in Reload Mode

Reload Mode for an auxiliary timer Tx is selected by setting bitfield TxM in the respective register TxCON to 100_B. In reload mode, the core timer T3 is reloaded with the contents of an auxiliary timer register, triggered by one of two different signals. The trigger signal is selected the same way as the clock source for counter mode (see **Table 14-5**), i.e. a transition of the auxiliary timer's input TxIN or the toggle latch T3OTL may trigger the reload.

Note: When programmed for reload mode, the respective auxiliary timer (T2 or T4) stops independently of its run flag T2R or T4R.
The timer input pin TxIN must be configured as input if it shall trigger a reload operation.

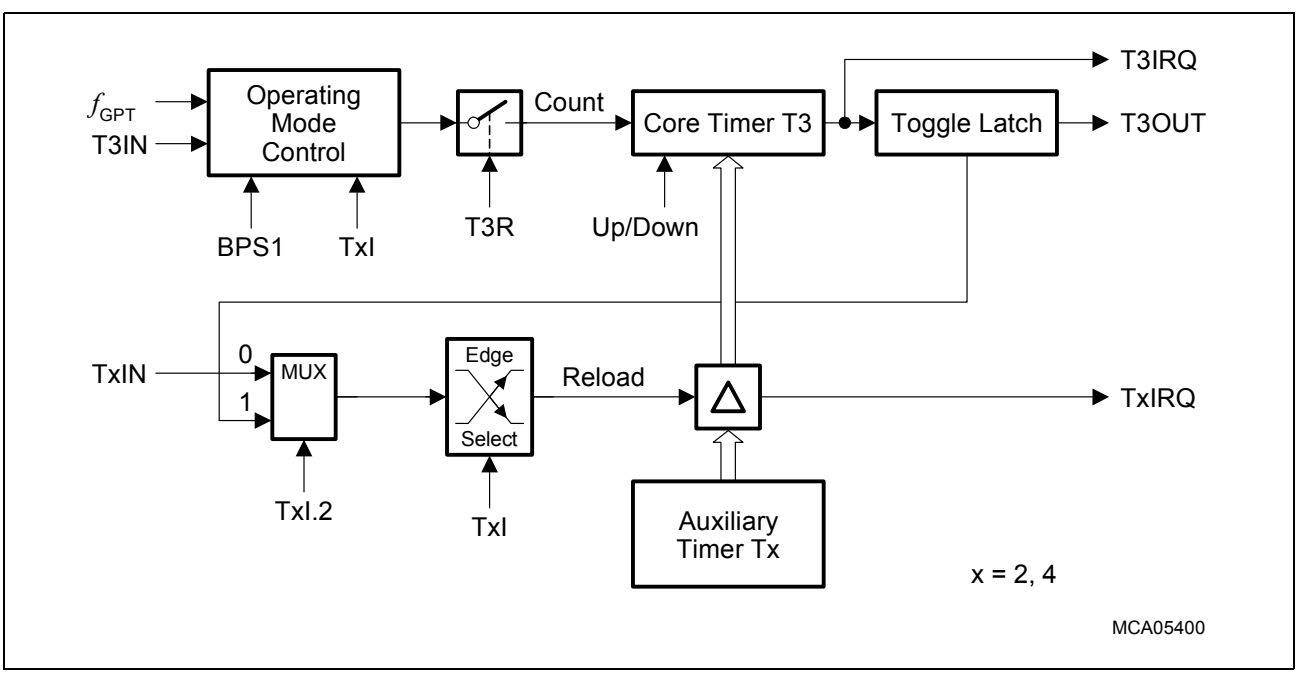


Figure 14-16 GPT1 Auxiliary Timer in Reload Mode

Upon a trigger signal, T3 is loaded with the contents of the respective timer register (T2 or T4) and the respective interrupt request flag (T2IR or T4IR) is set.

Note: When a T3OTL transition is selected for the trigger signal, the interrupt request flag T3IR will also be set upon a trigger, indicating T3's overflow or underflow. Modifications of T3OTL via software will NOT trigger the counter function of T2/T4.

To ensure that a transition of the reload input signal applied to TxIN is recognized correctly, its level must be held high or low for a minimum number of module clock cycles, detailed in **Section 14.1.5**.

The reload mode triggered by the T3 toggle latch can be used in a number of different configurations. The following functions can be performed, depending on the selected active transition:

- If both a positive and a negative transition of T3OTL are selected to trigger a reload, the core timer will be reloaded with the contents of the auxiliary timer each time it overflows or underflows. This is the standard reload mode (reload on overflow/underflow).
- If either a positive or a negative transition of T3OTL is selected to trigger a reload, the core timer will be reloaded with the contents of the auxiliary timer on every second overflow or underflow.
- Using this “single-transition” mode for both auxiliary timers allows to perform very flexible Pulse Width Modulation (PWM). One of the auxiliary timers is programmed to reload the core timer on a positive transition of T3OTL, the other is programmed for a reload on a negative transition of T3OTL. With this combination the core timer is alternately reloaded from the two auxiliary timers.

Figure 14-17 shows an example for the generation of a PWM signal using the “single-transition” reload mechanism. T2 defines the high time of the PWM signal (reloaded on positive transitions) and T4 defines the low time of the PWM signal (reloaded on negative transitions). The PWM signal can be output on pin T3OUT if T3OE = 1. With this method, the high and low time of the PWM signal can be varied in a wide range.

Note: The output toggle latch T3OTL is accessible via software and may be changed, if required, to modify the PWM signal.
However, this will NOT trigger the reloading of T3.

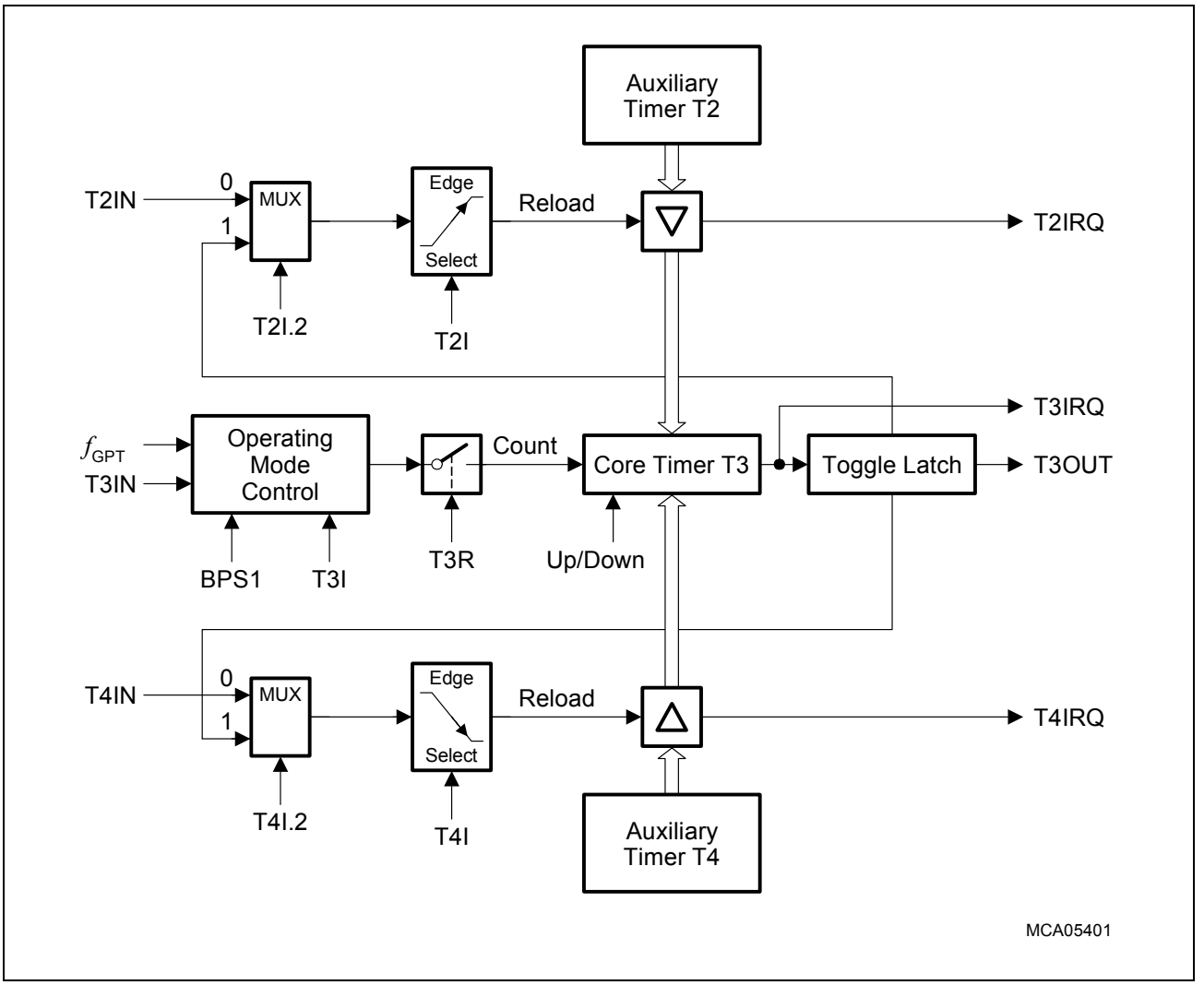


Figure 14-17 GPT1 Timer Reload Configuration for PWM Generation

Note: Although possible, selecting the same reload trigger event for both auxiliary timers should be avoided. In such a case, both reload registers would try to load the core timer at the same time. If this combination is selected, T2 is disregarded and the contents of T4 is reloaded.

Auxiliary Timer in Capture Mode

Capture mode for an auxiliary timer Tx is selected by setting bitfield TxM in the respective register TxCON to 101_B. In capture mode, the contents of the core timer T3 are latched into an auxiliary timer register in response to a signal transition at the respective auxiliary timer's external input pin TxIN. The capture trigger signal can be a positive, a negative, or both a positive and a negative transition.

The two least significant bits of bitfield TxI select the active transition (see **Table 14-5**). Bit 2 of TxI is irrelevant for capture mode and must be cleared (TxI.2 = 0).

Note: When programmed for capture mode, the respective auxiliary timer (T2 or T4) stops independently of its run flag T2R or T4R.

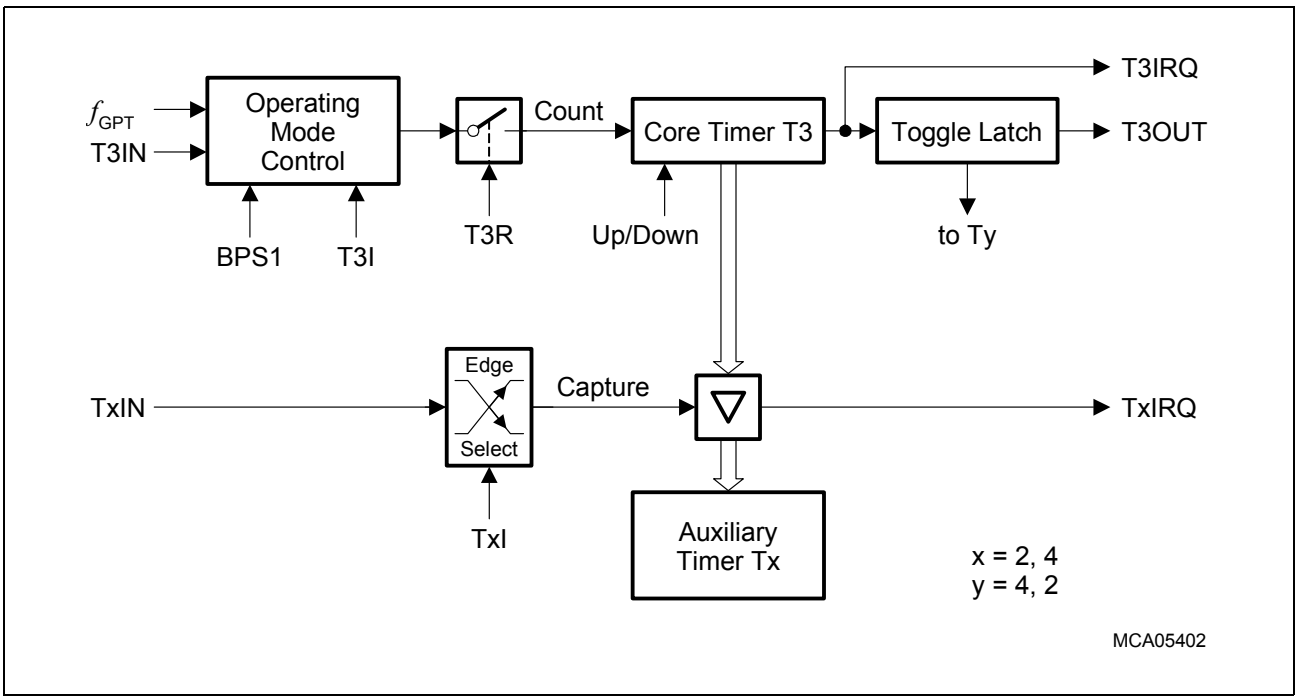


Figure 14-18 GPT1 Auxiliary Timer in Capture Mode

Upon a trigger (selected transition) at the corresponding input pin TxIN the contents of the core timer are loaded into the auxiliary timer register and the associated interrupt request flag TxIR will be set.

For capture mode operation, the respective timer input pin TxIN must be configured as input. To ensure that a transition of the capture input signal applied to TxIN is recognized correctly, its level must be held high or low for a minimum number of module clock cycles, detailed in **Section 14.1.5**.

14.1.5 GPT1 Clock Signal Control

All actions within the timer block GPT1 are triggered by transitions of its basic clock. This basic clock is derived from the system clock by a basic block prescaler, controlled by bitfield BPS1 in register T3CON (see **Figure 14-2**). The count clock can be generated in two different ways:

- **Internal count clock**, derived from GPT1's basic clock via a programmable prescaler, is used for (gated) timer mode.
- **External count clock**, derived from the timer's input pin(s), is used for counter mode.

For both ways, the basic clock determines the maximum count frequency and the timer's resolution:

Table 14-6 Basic Clock Selection for Block GPT1

Block Prescaler[1]	BPS1 = 01_B	BPS1 = 00_B[2]	BPS1 = 11_B	BPS1 = 10_B
Prescaling Factor for GPT1: F(BPS1)	F(BPS1) = 4	F(BPS1) = 8	F(BPS1) = 16	F(BPS1) = 32
Maximum External Count Frequency	$f_{GPT}/8$	$f_{GPT}/16$	$f_{GPT}/32$	$f_{GPT}/64$
Input Signal Stable Time	$4 \times t_{GPT}$	$8 \times t_{GPT}$	$16 \times t_{GPT}$	$32 \times t_{GPT}$

1) Please note the non-linear encoding of bitfield BPS1.

2) Default after reset.

Internal Count Clock Generation

In timer mode and gated timer mode, the count clock for each GPT1 timer is derived from the GPT1 basic clock by a programmable prescaler, controlled by bitfield TxI in the respective timer's control register TxCON.

The count frequency f_{Tx} for a timer Tx and its resolution r_{Tx} are scaled linearly with lower clock frequencies, as can be seen from the following formula:

$$f_{Tx} = \frac{f_{GPT}}{F(BPS1) \times 2^{<TxI>}} \qquad r_{Tx}[\mu s] = \frac{F(BPS1) \times 2^{<TxI>}}{f_{GPT}[MHz]} \tag{14.1}$$

The effective count frequency depends on the common module clock prescaler factor F(BPS1) as well as on the individual input prescaler factor $2^{<TxI>}$. **Table 14-7** summarizes the resulting overall divider factors for a GPT1 timer that result from these cascaded prescalers.

Table 14-8 lists a timer's parameters (such as count frequency, resolution, and period) resulting from the selected overall prescaler factor and the applied system frequency. Note that some numbers may be rounded.

Table 14-7 GPT1 Overall Prescaler Factors for Internal Count Clock

Individual Prescaler for Tx	Common Prescaler for Module Clock[1)]			
	BPS1 = 01_B	BPS1 = 00_B	BPS1 = 11_B	BPS1 = 10_B
Txl = 000_B	4	8	16	32
Txl = 001_B	8	16	32	64
Txl = 010_B	16	32	64	128
Txl = 011_B	32	64	128	256
Txl = 100_B	64	128	256	512
Txl = 101_B	128	256	512	1024
Txl = 110_B	256	512	1024	2048
Txl = 111_B	512	1024	2048	4096

1) Please note the non-linear encoding of bitfield BPS1.

Table 14-8 GPT1 Timer Parameters

System Clock = 10 MHz			Overall Divider Factor	System Clock = 40 MHz		
Frequency	Resolution	Period		Frequency	Resolution	Period
2.5 MHz	400 ns	26.21 ms	4	10.0 MHz	100 ns	6.55 ms
1.25 MHz	800 ns	52.43 ms	8	5.0 MHz	200 ns	13.11 ms
625.0 kHz	1.6 µs	104.9 ms	16	2.5 MHz	400 ns	26.21 ms
312.5 kHz	3.2 µs	209.7 ms	32	1.25 MHz	800 ns	52.43 ms
156.25 kHz	6.4 µs	419.4 ms	64	625.0 kHz	1.6 µs	104.9 ms
78.125 kHz	12.8 µs	838.9 ms	128	312.5 kHz	3.2 µs	209.7 ms
39.06 kHz	25.6 µs	1.678 s	256	156.25 kHz	6.4 µs	419.4 ms
19.53 kHz	51.2 µs	3.355 s	512	78.125 kHz	12.8 µs	838.9 ms
9.77 kHz	102.4 µs	6.711 s	1024	39.06 kHz	25.6 µs	1.678 s
4.88 kHz	204.8 µs	13.42 s	2048	19.53 kHz	51.2 µs	3.355 s
2.44 kHz	409.6 µs	26.84 s	4096	9.77 kHz	102.4 µs	6.711 s

External Count Clock Input

The external input signals of the GPT1 block are sampled with the GPT1 basic clock (see **Figure 14-2**). To ensure that a signal is recognized correctly, its current level (high or low) must be held active for at least one complete sampling period, before changing. A signal transition is recognized if two subsequent samples of the input signal represent different levels. Therefore, a minimum of two basic clock periods are required for the sampling of an external input signal. Thus, the maximum frequency of an input signal must not be higher than half the basic clock.

Table 14-9 summarizes the resulting requirements for external GPT1 input signals.

Table 14-9 GPT1 External Input Signal Limits

System Clock = 10 MHz		Input Frequ. Factor	GPT1 Divider BPS1	Input Phase Duration	System Clock = 40 MHz	
Max. Input Frequency	Min. Level Hold Time				Max. Input Frequency	Min. Level Hold Time
1.25 MHz	400 ns	$f_{GPT}/8$	01_B	$4 \times t_{GPT}$	5.0 MHz	100 ns
625.0 kHz	800 ns	$f_{GPT}/16$	00_B	$8 \times t_{GPT}$	2.5 MHz	200 ns
312.5 kHz	1.6 µs	$f_{GPT}/32$	11_B	$16 \times t_{GPT}$	1.25 MHz	400 ns
156.25 kHz	3.2 µs	$f_{GPT}/64$	10_B	$32 \times t_{GPT}$	625.0 kHz	800 ns

These limitations are valid for all external input signals to GPT1, including the external count signals in counter mode and incremental interface mode, the gate input signals in gated timer mode, and the external direction signals.

14.1.6 GPT1 Timer Registers

GPT12E_Tx
Timer x Count Register **SFR ($FE4x_H/2y_H$)** **Reset Value: 0000_H**

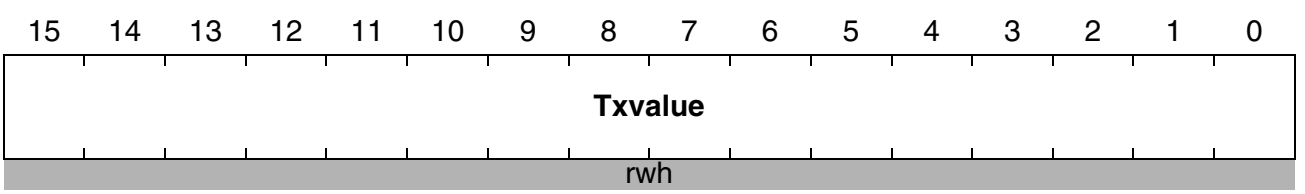


Table 14-10 GPT1 Timer Register Locations

Timer Register	Physical Address	8-Bit Address
T3	$FE42_H$	21_H
T2	$FE40_H$	20_H
T4	$FE44_H$	22_H

14.1.7 Interrupt Control for GPT1 Timers

When a timer overflows from $FFFF_H$ to 0000_H (when counting up), or when it underflows from 0000_H to $FFFF_H$ (when counting down), its interrupt request flag (T2IR, T3IR or T4IR) in register TxIC will be set. This will cause an interrupt to the respective timer interrupt vector (T2INT, T3INT or T4INT) or trigger a PEC service, if the respective interrupt enable bit (T2IE, T3IE or T4IE in register TxIC) is set. There is an interrupt control register for each of the three timers.

GPT12E_T2IC
Timer 2 Intr. Ctrl. Reg. **SFR ($FF60_H$/$B0_H$)** **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	GPX	T2IR	T2IE	ILVL				GLVL	
-	-	-	-	-	-	-	rw	rwh	rw	rw				rw	

GPT12E_T3IC
Timer 3 Intr. Ctrl. Reg. **SFR ($FF62_H$/$B1_H$)** **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	GPX	T3IR	T3IE	ILVL				GLVL	
-	-	-	-	-	-	-	rw	rwh	rw	rw				rw	

GPT12E_T4IC
Timer 4 Intr. Ctrl. Reg. **SFR ($FF64_H$/$B2_H$)** **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	GPX	T4IR	T4IE	ILVL				GLVL	
-	-	-	-	-	-	-	rw	rwh	rw	rw				rw	

Note: Please refer to the general Interrupt Control Register description for an explanation of the control fields.

14.2 Timer Block GPT2

From a programmer's point of view, the GPT2 block is represented by a set of SFRs as summarized below. Those portions of port and direction registers which are used for alternate functions by the GPT2 block are shaded.

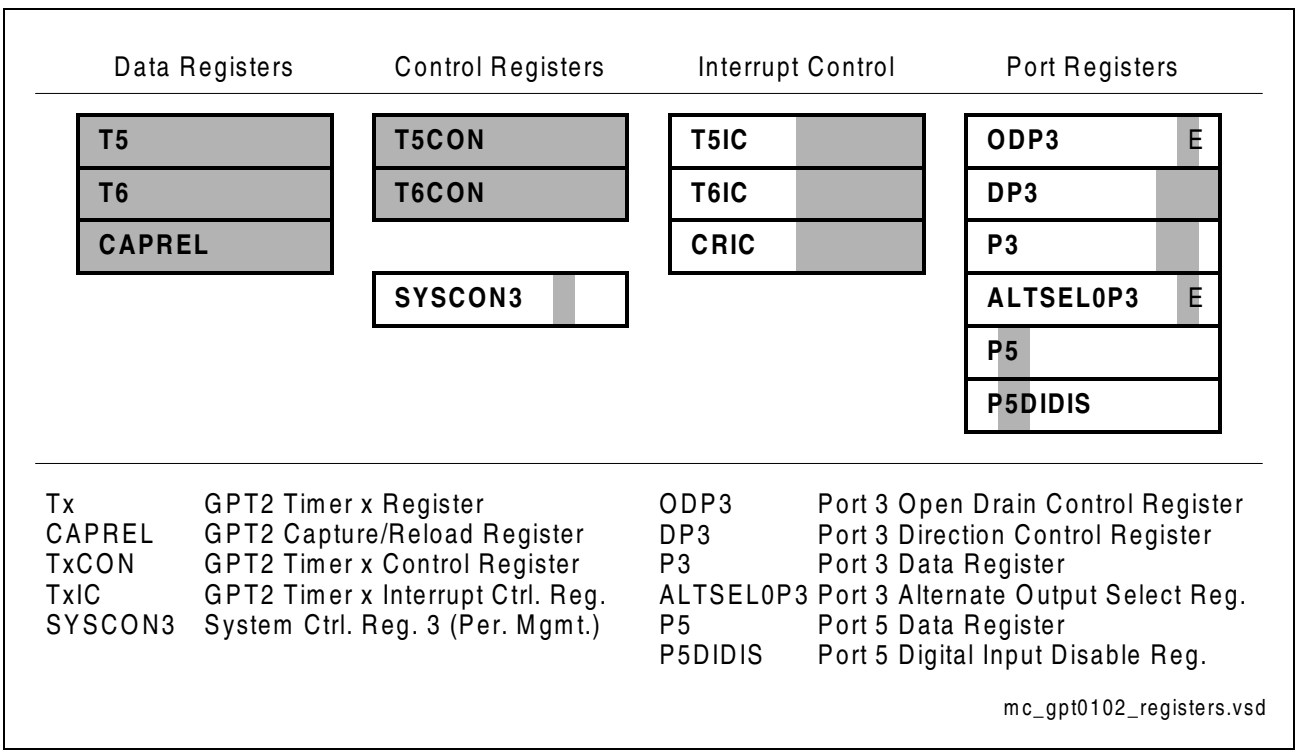


Figure 14-19 SFRs Associated with Timer Block GPT2

Both timers of block GPT2 (T5, T6) can run in one of 3 basic modes: Timer Mode, Gated Timer Mode, or Counter Mode. All timers can count up or down. Each timer of GPT2 is controlled by a separate control register TxCON.

Each timer has an input pin TxIN (alternate pin function) associated with it, which serves as the gate control in gated timer mode, or as the count input in counter mode. The count direction (up/down) may be programmed via software. An overflow/underflow of core timer T6 is indicated by the Output Toggle Latch T6OTL, whose state may be output on the associated pin T6OUT (alternate pin function). The auxiliary timer T5 may additionally be concatenated with the core timer T6 (through T6OTL).

The Capture/Reload register CAPREL can be used to capture the contents of timer T5, or to reload timer T6. A special mode facilitates the use of register CAPREL for both functions at the same time. This mode allows frequency multiplication. The capture function is triggered by the input pin CAPIN, or by GPT1 timer's T3 input lines T3IN and T3EUD. The reload function is triggered by an overflow or underflow of timer T6. Overflows/underflows of timer T6 may also clock the timers of the CAPCOM units.

The current contents of each timer can be read or modified by the CPU by accessing the corresponding timer count registers T5 or T6, located in the non-bitaddressable SFR

space (see **Section 14.2.7**). When any of the timer registers is written to by the CPU in the state immediately preceding a timer increment, decrement, reload, or capture operation, the CPU write operation has priority in order to guarantee correct results.

The interrupts of GPT2 are controlled through the Interrupt Control Registers TxIC. These registers are not part of the GPT2 block. The input and output lines of GPT2 are connected to pins of Ports P3 and P5. The control registers for the port functions are located in the respective port modules.

*Note: The timing requirements for external input signals can be found in **Section 14.2.6**, **Section 14.3** summarizes the module interface signals, including pins.*

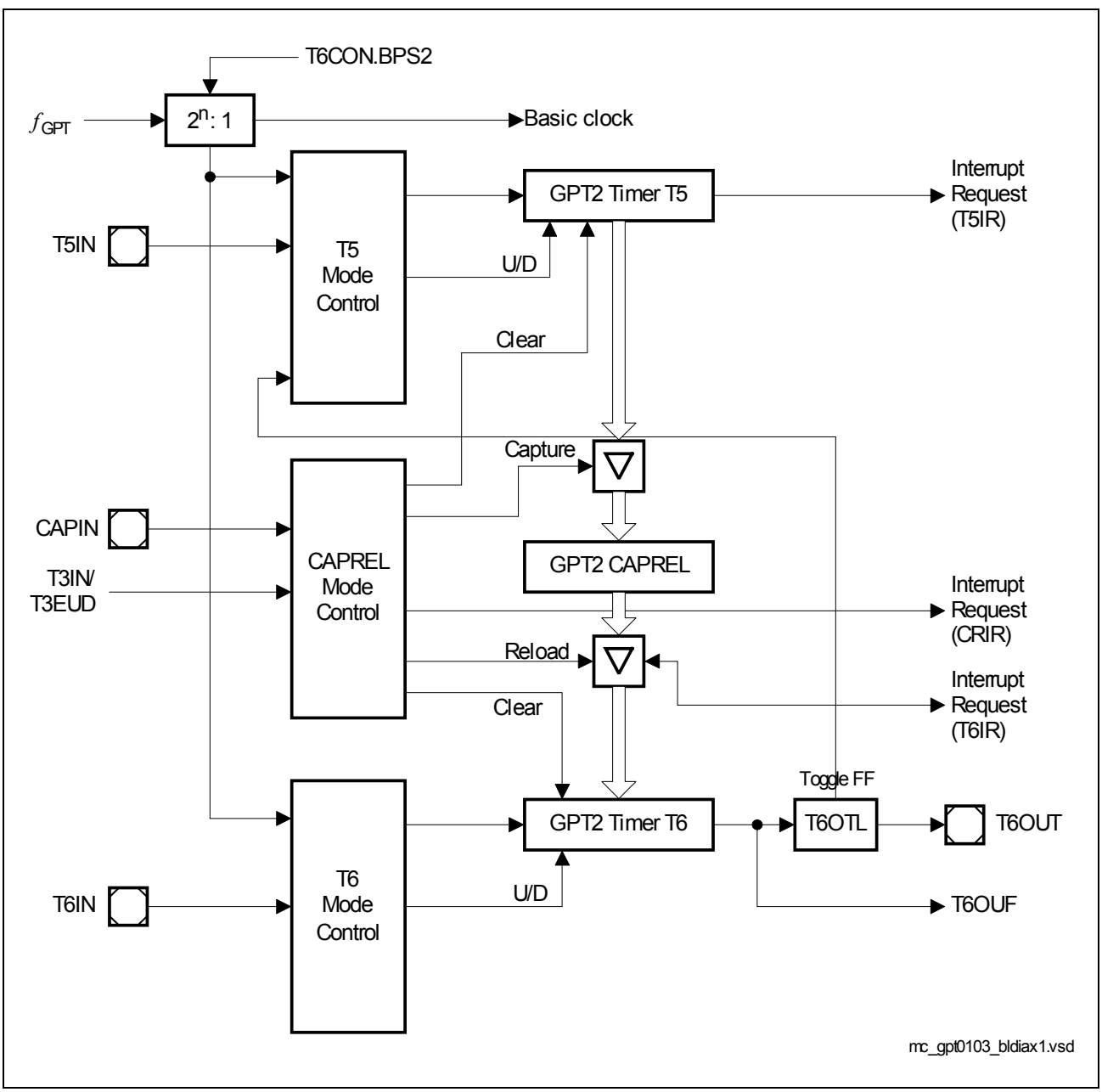


Figure 14-20 GPT2 Block Diagram

14.2.1 GPT2 Core Timer T6 Control

The current contents of the core timer T6 are reflected by its count register T6. This register can also be written to by the CPU, for example, to set the initial start value.

The core timer T6 is configured and controlled via its bitaddressable control register T6CON.

GPT12E_T6CON
Timer 6 Control Register **SFR ($FF48_H/A4_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T6 SR	T6 CLR	-	BPS2		T6 OTL	T6 OE	-	T6 UD	T6R	T6M			T6I		
rw	rw	-	rw		rwh	rw	-	rw	rw	rw			rw		

Field	Bits	Typ	Description
T6SR	15	rw	**Timer 6 Reload Mode Enable** 0 Reload from register CAPREL Disabled 1 Reload from register CAPREL Enabled
T6CLR	14	rw	**Timer T6 Clear Enable Bit** 0 Timer T6 is not cleared on a capture event 1 Timer T6 is cleared on a capture event
BPS2	[12:11]	rw	**GPT2 Block Prescaler Control** Selects the basic clock for block GPT2 (see also **Section 14.2.6**) 00 $f_{GPT}/4$ 01 $f_{GPT}/2$ 10 $f_{GPT}/16$ 11 $f_{GPT}/8$
T6OTL	10	rwh	**Timer T6 Overflow Toggle Latch** Toggles on each overflow/underflow of T6. Can be set or reset by software (see separate description)
T6OE	9	rw	**Overflow/Underflow Output Enable** 0 Alternate Output Function Disabled 1 State of T6 toggle latch is output on pin T6OUT
T6UD	7	rw	**Timer T6 Up/Down Control** 0 Timer T6 counts up 1 Timer T6 counts down

Field	Bits	Typ	Description
T6R	6	rw	**Timer T6 Run Bit** 0 Timer T6 stops 1 Timer T6 runs
T6M	[5:3]	rw	**Timer T6 Mode Control** (Basic Operating Mode) 000 Timer Mode 001 Counter Mode 010 Gated Timer Mode with gate active low 011 Gated Timer Mode with gate active high 100 Reserved. Do not use this combination. 101 Reserved. Do not use this combination. 110 Reserved. Do not use this combination. 111 Reserved. Do not use this combination.
T6I	[2:0]	rw	**Timer T6 Input Parameter Selection** Depends on the operating mode, see respective sections for encoding: **Table 14-15** for Timer Mode and Gated Timer Mode **Table 14-11** for Counter Mode

Timer T6 Run Control

The core timer T6 can be started or stopped by software through bit T6R (timer T6 run bit). This bit is relevant in all operating modes of T6. Setting bit T6R will start the timer, clearing bit T6R stops the timer.

In gated timer mode, the timer will only run if T6R = 1 and the gate is active (high or low, as programmed).

Note: When bit T5RC in timer control register T5CON is set, bit T6R will also control (start and stop) the Auxiliary Timer T5.

Count Direction Control

The count direction of the GPT2 timers (core timer and auxiliary timer) can be controlled by software. The count direction can be altered by setting or clearing bit TxUD. The count direction can be changed regardless of whether or not the timer is running.

Timer 6 Output Toggle Latch

The overflow/underflow signal of timer T6 is connected to a block named ‘Toggle Latch’, shown in the timer mode diagrams. **Figure 14-21** illustrates the details of this block. An overflow or underflow of T6 will clock two latches: The first latch represents bit T6OTL in control register T6CON. The second latch is an internal latch toggled by T6OTL’s output. Both latch outputs are connected to the input control block of the auxiliary timer T5. The output level of the shadow latch will match the output level of T6OTL, but is delayed by one clock cycle. When the T6OTL value changes, this will result in a temporarily different output level from T6OTL and the shadow latch, which can trigger the selected count event in T5.

When software writes to T6OTL, both latches are set or cleared simultaneously. In this case, both signals to the auxiliary timers carry the same level and no edge will be detected. Bit T6OE (overflow/underflow output enable) in register T6CON enables the state of T6OTL to be monitored via an external pin T6OUT. When T6OTL is linked to an external port pin (must be configured as output), T6OUT can be used to control external HW. If T6OE = 1, pin T6OUT outputs the state of T6OTL. If T6OE = 0, pin T6OUT outputs a high level (while it selects the timer output signal).

As can be seen from **Figure 14-21**, when latch T6OTL is modified by software to determine the state of the output line, also the internal shadow latch is set or cleared accordingly. Therefore, no trigger condition is detected by T5 in this case.

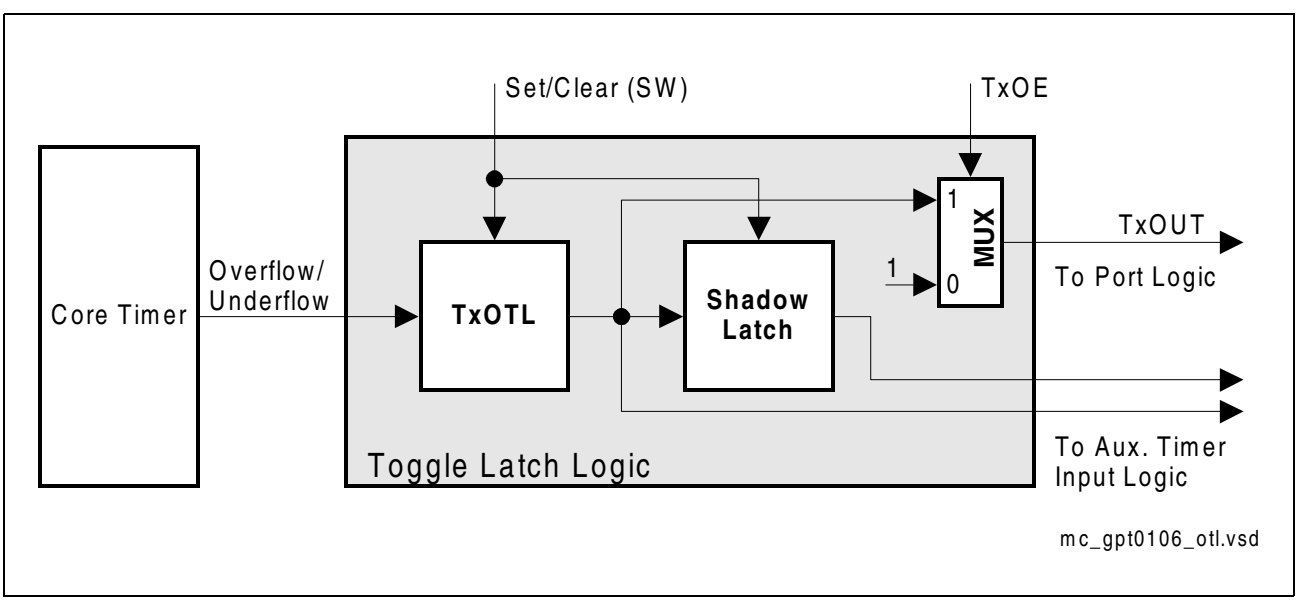


Figure 14-21 Block Diagram of the Toggle Latch Logic of Core Timer T6

Note: T6 is also used to clock the timers in the CAPCOM units. For this purpose, there is a direct internal connection between the T6 overflow/underflow line and the CAPCOM timers (signal T6OUF).

14.2.2 GPT2 Core Timer T6 Operating Modes

Timer 6 in Timer Mode

Timer mode for the core timer T6 is selected by setting bitfield T6M in register T6CON to 000_B. In this mode, T6 is clocked with the module's input clock f_{GPT} divided by two programmable prescalers controlled by bitfields BPS2 and T6I in register T6CON. Please see **Section 14.2.6** for details on the input clock options.

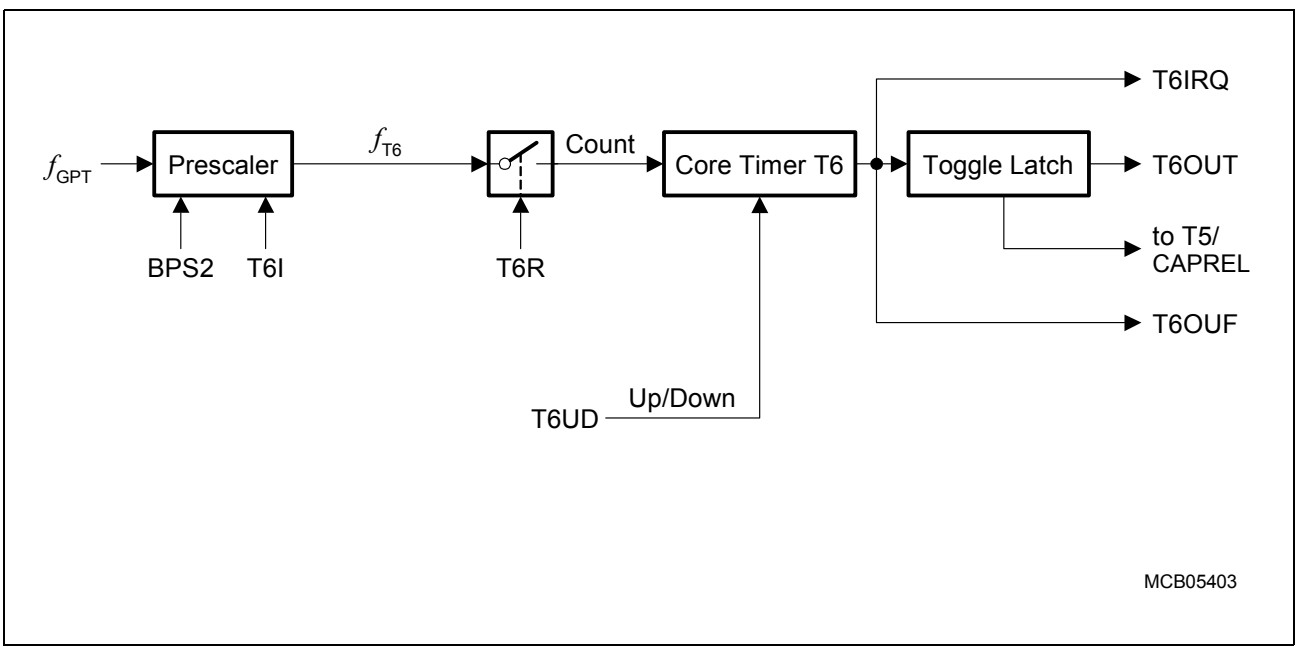


Figure 14-22 Block Diagram of Core Timer T6 in Timer Mode

Gated Timer Mode

Gated timer mode for the core timer T6 is selected by setting bitfield T6M in register T6CON to 010_B or 011_B. Bit T6M.0 (T6CON.3) selects the active level of the gate input. The same options for the input frequency are available in gated timer mode as in timer mode (see **Section 14.2.6**). However, the input clock to the timer in this mode is gated by the external input pin T6IN (Timer T6 External Input).
To enable this operation, the associated pin T6IN must be configured as input (the corresponding direction control bit must contain 0).

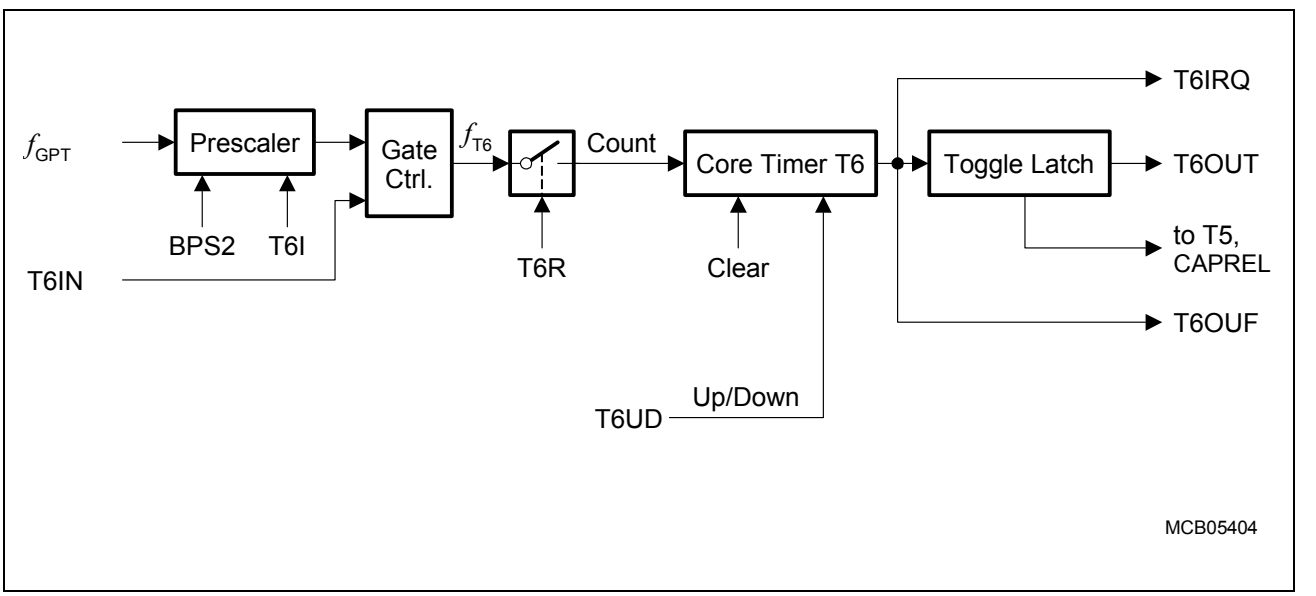


Figure 14-23 Block Diagram of Core Timer T6 in Gated Timer Mode

If T6M = 010_B, the timer is enabled when T6IN shows a low level. A high level at this line stops the timer. If T6M = 011_B, line T6IN must have a high level in order to enable the timer. Additionally, the timer can be turned on or off by software using bit T6R. The timer will only run if T6R is 1 and the gate is active. It will stop if either T6R is 0 or the gate is inactive.

Note: A transition of the gate signal at pin T6IN does not cause an interrupt request.

Counter Mode

Counter mode for the core timer T6 is selected by setting bitfield T6M in register T6CON to 001_B. In counter mode, timer T6 is clocked by a transition at the external input pin T6IN. The event causing an increment or decrement of the timer can be a positive, a negative, or both a positive and a negative transition at this line. Bitfield T6I in control register T6CON selects the triggering transition (see **Table 14-11**).

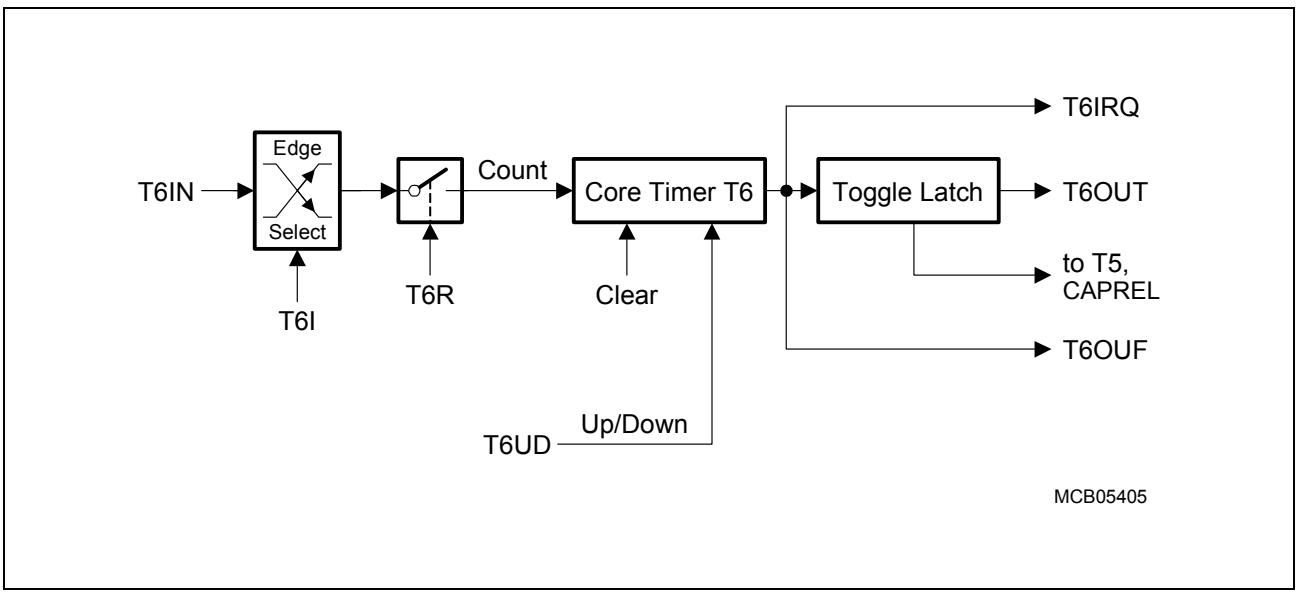


Figure 14-24 Block Diagram of Core Timer T6 in Counter Mode

Table 14-11 GPT2 Core Timer T6 (Counter Mode) Input Edge Selection

T6I	Triggering Edge for Counter Increment/Decrement
0 0 0	None. Counter T6 is disabled
0 0 1	Positive transition (rising edge) on T6IN
0 1 0	Negative transition (falling edge) on T6IN
0 1 1	Any transition (rising or falling edge) on T6IN
1 X X	Reserved. Do not use this combination

For counter mode operation, pin T6IN must be configured as input (the respective direction control bit DPx.y must be 0). The maximum input frequency allowed in counter mode depends on the selected prescaler value. To ensure that a transition of the count input signal applied to T6IN is recognized correctly, its level must be held high or low for a minimum number of module clock cycles before it changes. This information can be found in **Section 14.2.6**.

14.2.3 GPT2 Auxiliary Timer T5 Control

Auxiliary timer T5 can be configured for timer mode, gated timer mode, or counter mode with the same options for the timer frequencies and the count signal as the core timer T6. In addition to these 3 counting modes, the auxiliary timer can be concatenated with the core timer. The contents of T5 may be captured to register CAPREL upon an external or an internal trigger. The start/stop function of the auxiliary timers can be remotely controlled by the T6 run control bit. Several timers may thus be controlled synchronously.

The current contents of the auxiliary timer are reflected by its count register T5. This register can also be written to by the CPU, for example, to set the initial start value.

The individual configurations for timer T5 are determined by its bitaddressable control register T5CON. Some bits in this register also control the function of the CAPREL register. Note that functions which are present in all timers of block GPT2 are controlled in the same bit positions and in the same manner in each of the specific control registers.

Note: The auxiliary timer has no output toggle latch and no alternate output function.

GPT12E_T5CON
Timer 5 Control Register **SFR ($FF46_H/A3_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T5 SC	**T5 CLR**	**CI**		**T5 CC**	**CT3**	**T5 RC**	**-**	**T5 UD**	**T5R**	**T5M**			**T5I**		
rw	rw	rw		rw	rw	rw	-	rw	rw	rw			rw		

Field	Bits	Typ	Description
T5SC	15	rw	**Timer 5 Capture Mode Enable** 0 Capture into register CAPREL Disabled 1 Capture into register CAPREL Enabled
T5CLR	14	rw	**Timer T5 Clear Enable Bit** 0 Timer T5 is not cleared on a capture event 1 Timer T5 is cleared on a capture event
CI	[13:12]	rw	**Register CAPREL Capture Trigger Selection** (depending on bit CT3) 00 Capture disabled 01 Positive transition (rising edge) on CAPIN or any transition on T3IN 10 Negative transition (falling edge) on CAPIN or any transition on T3EUD 11 Any transition (rising or falling edge) on CAPIN or any transition on T3IN or T3EUD

Field	Bits	Typ	Description
T5CC	11	rw	**Timer T5 Capture Correction** 0 T5 is just captured without any correction 1 T5 is decremented by 1 before being captured
CT3	10	rw	**Timer T3 Capture Trigger Enable** 0 Capture trigger from input line CAPIN 1 Capture trigger from T3 input lines T3IN and/or T3EUD
T5RC	9	rw	**Timer T5 Remote Control** 0 Timer T5 is controlled by its own run bit T5R 1 Timer T5 is controlled by the run bit T6R of core timer 6, not by bit T5R
T5UD	7	rw	**Timer T5 Up/Down Control** 0 Timer T5 counts up 1 Timer T5 counts down
T5R	6	rw	**Timer T5 Run Bit** 0 Timer T5 stops 1 Timer T5 runs *Note: This bit only controls timer T5 if bit T5RC = 0.*
T5M	[5:3]	rw	**Timer T5 Mode Control** (Basic Operating Mode) 000 Timer Mode 001 Counter Mode 010 Gated Timer Mode with gate active low 011 Gated Timer Mode with gate active high 1XX Reserved. Do not use this combination
T5I	[2:0]	rw	**Timer T5 Input Parameter Selection** Depends on the operating mode, see respective sections for encoding: **Table 14-15** for Timer Mode and Gated Timer Mode **Table 14-11** for Counter Mode

Timer T5 Run Control

The auxiliary timer T5 can be started or stopped by software in two different ways:

- Through the associated timer run bit (T5R). In this case it is required that the respective control bit T5RC = 0.
- Through the core timer's run bit (T6R). In this case the respective remote control bit must be set (T5RC = 1).

The selected run bit is relevant in all operating modes of T5. Setting the bit will start the timer, clearing the bit stops the timer.

In gated timer mode, the timer will only run if the selected run bit is set and the gate is active (high or low, as programmed).

Note: If remote control is selected T6R will start/stop timer T6 and the auxiliary timer T5 synchronously.

14.2.4 GPT2 Auxiliary Timer T5 Operating Modes

The operation of the auxiliary timer in the basic operating modes is almost identical with the core timer's operation, with very few exceptions. Additionally, some combined operating modes can be selected.

Timer T5 in Timer Mode

Timer Mode for the auxiliary timer T5 is selected by setting its bitfield T5M in register T5CON to 000_B.

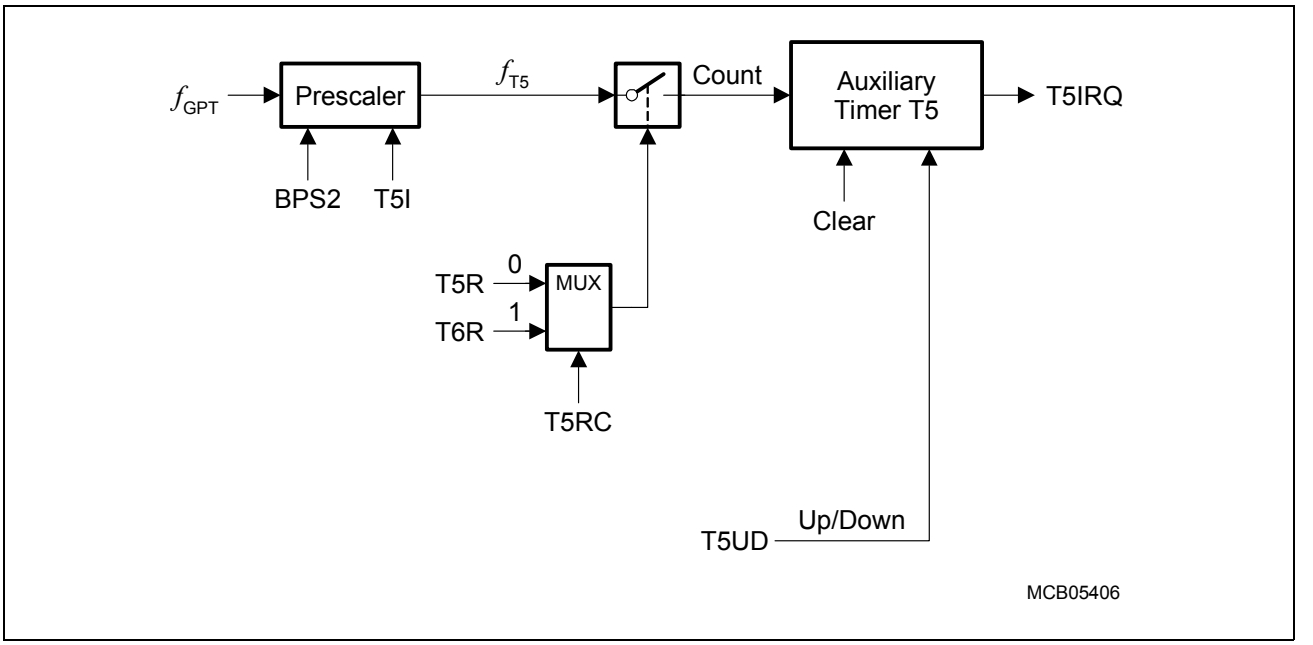


Figure 14-25 Block Diagram of Auxiliary Timer T5 in Timer Mode

Timer T5 in Gated Timer Mode

Gated timer mode for the auxiliary timer T5 is selected by setting bitfield T5M in register T5CON to 010_B or 011_B. Bit T5M.0 (T5CON.3) selects the active level of the gate input.

Note: A transition of the gate signal at line T5IN does not cause an interrupt request.

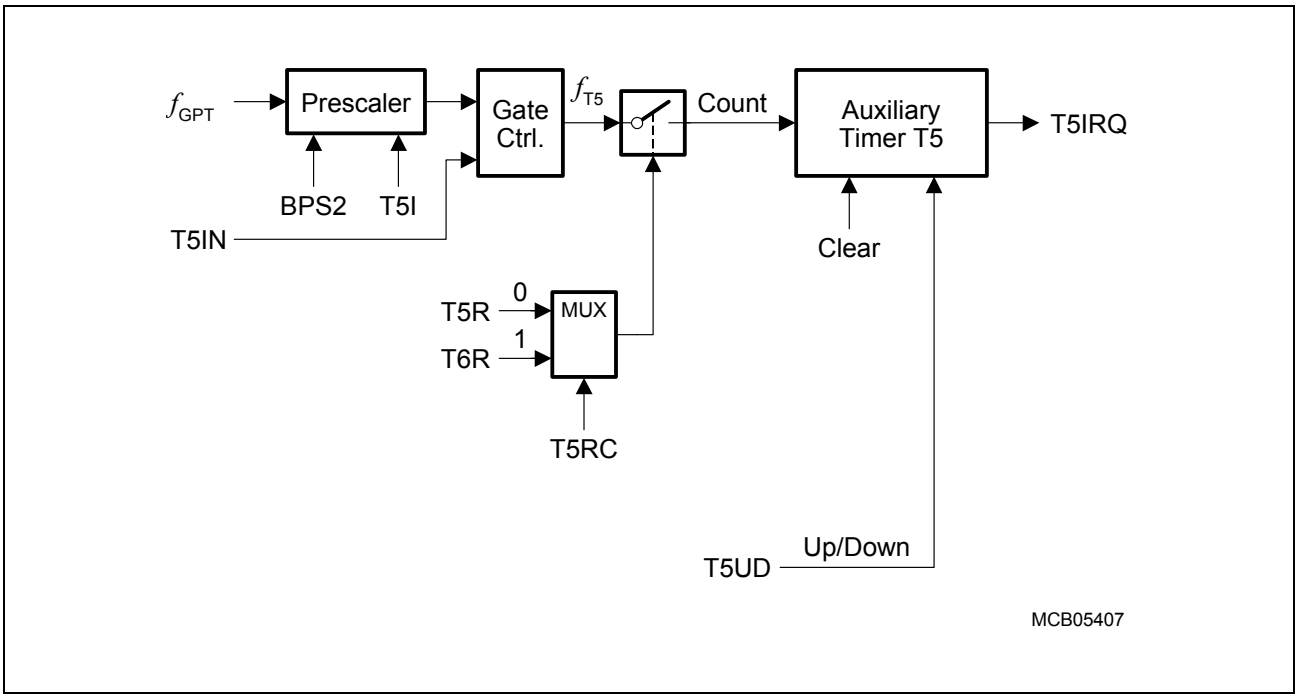


Figure 14-26 Block Diagram of Auxiliary Timer T5 in Gated Timer Mode

Note: There is no output toggle latch for T5.
Start/stop of the auxiliary timer can be controlled locally or remotely.

Timer T5 in Counter Mode

Counter mode for auxiliary timer T5 is selected by setting bitfield T5M in register T5CON to 001_B. In counter mode, the auxiliary timer can be clocked either by a transition at its external input line T5IN, or by a transition of timer T6's toggle latch T6OTL. The event causing an increment or decrement of a timer can be a positive, a negative, or both a positive and a negative transition at either the respective input pin or at the toggle latch. Bitfield T5I in control register T5CON selects the triggering transition (see **Table 14-12**).

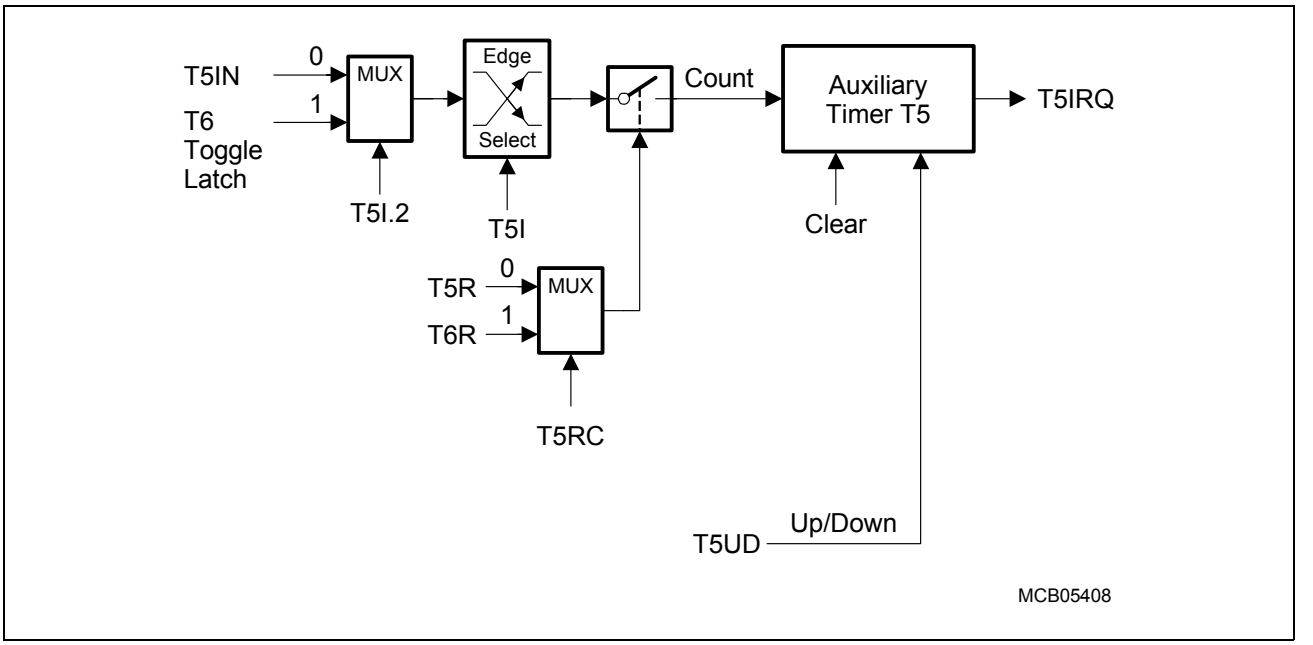


Figure 14-27 Block Diagram of Auxiliary Timer T5 in Counter Mode

Table 14-12 GPT2 Auxiliary Timer (Counter Mode) Input Edge Selection

T5I	Triggering Edge for Counter Increment/Decrement
X 0 0	None. Counter T5 is disabled
0 0 1	Positive transition (rising edge) on T5IN
0 1 0	Negative transition (falling edge) on T5IN
0 1 1	Any transition (rising or falling edge) on T5IN
1 0 1	Positive transition (rising edge) of T6 toggle latch T6OTL
1 1 0	Negative transition (falling edge) of T6 toggle latch T6OTL
1 1 1	Any transition (rising or falling edge) of T6 toggle latch T6OTL

Note: Only state transitions of T6OTL which are caused by the overflows/underflows of T6 will trigger the counter function of T5. Modifications of T6OTL via software will NOT trigger the counter function of T5.

For counter operation, pin T5IN must be configured as input (the respective direction control bit DPx.y must be 0). The maximum input frequency allowed in counter mode depends on the selected prescaler value. To ensure that a transition of the count input signal applied to T5IN is recognized correctly, its level must be held high or low for a minimum number of module clock cycles before it changes. This information can be found in **Section 14.2.6**.

Timer Concatenation

Using the toggle bit T6OTL as a clock source for the auxiliary timer in counter mode concatenates the core timer T6 with the auxiliary timer T5. This concatenation forms either a 32-bit or a 33-bit timer/counter, depending on which transition of T6OTL is selected to clock the auxiliary timer.

- **32-bit Timer/Counter:** If both a positive and a negative transition of T6OTL are used to clock the auxiliary timer, this timer is clocked on every overflow/underflow of the core timer T6. Thus, the two timers form a 32-bit timer.
- **33-bit Timer/Counter:** If either a positive or a negative transition of T6OTL is selected to clock the auxiliary timer, this timer is clocked on every second overflow/underflow of the core timer T6. This configuration forms a 33-bit timer (16-bit core timer + T6OTL + 16-bit auxiliary timer).
  As long as bit T6OTL is not modified by software, it represents the state of the internal toggle latch, and can be regarded as part of the 33-bit timer.

The count directions of the two concatenated timers are not required to be the same. This offers a wide variety of different configurations.

T6, which represents the low-order part of the concatenated timer, can operate in timer mode, gated timer mode or counter mode in this case.

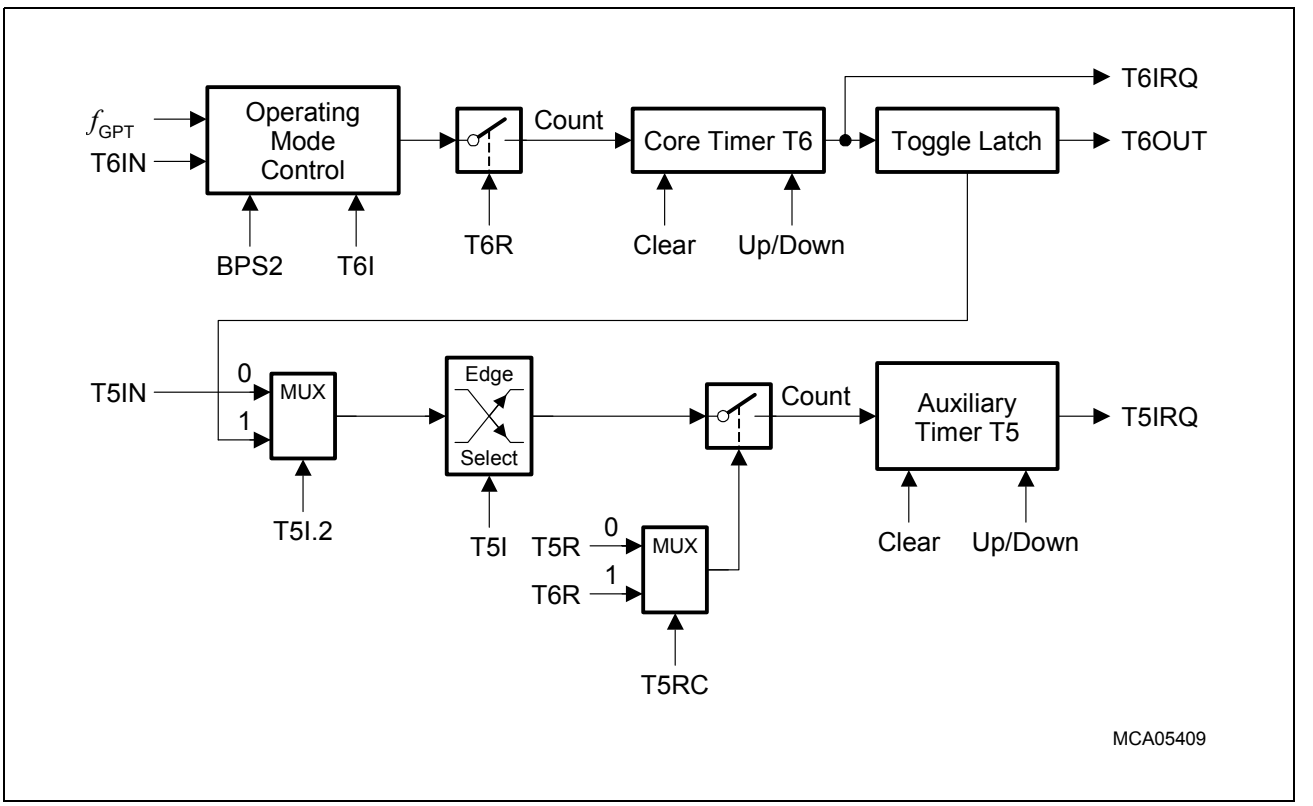


Figure 14-28 Concatenation of Core Timer T6 and Auxiliary Timer T5

14.2.5 GPT2 Register CAPREL Operating Modes

The Capture/Reload register CAPREL can be used to capture the contents of timer T5, or to reload timer T6. A special mode facilitates the use of register CAPREL for both functions at the same time. This mode allows frequency multiplication. The capture function is triggered by the input pin CAPIN, or by GPT1 timer's T3 input lines T3IN and T3EUD. The reload function is triggered by an overflow or underflow of timer T6.

In addition to the capture function, the capture trigger signal can also be used to clear the contents of timers T5 and T6 individually.

The functions of register CAPREL are controlled via several bit(field)s in the timer control registers T5CON and T6CON.

GPT2 Capture/Reload Register CAPREL in Capture Mode

Capture mode for register CAPREL is selected by setting bit T5SC in control register T5CON (set bitfield CI in register T5CON to a non-zero value to select a trigger signal). In capture mode, the contents of the auxiliary timer T5 are latched into register CAPREL in response to a signal transition at the selected external input pin(s). Bit CT3 selects the external input line CAPIN or the input lines T3IN and/or T3EUD of GPT1 timer T3 as the source for a capture trigger. Either a positive, a negative, or both a positive and a negative transition at line CAPIN can be selected to trigger the capture function, or transitions on input T3IN or input T3EUD or both inputs, T3IN and T3EUD. The active edge is controlled by bitfield CI in register T5CON. **Table 14-13** summarizes these options.

Table 14-13 CAPREL Register Input Edge Selection

CT3	CI	Triggering Signal/Edge for Capture Mode
X	0 0	None. Capture Mode is disabled.
0	0 1	Positive transition (rising edge) on CAPIN.
0	1 0	Negative transition (falling edge) on CAPIN.
0	1 1	Any transition (rising or falling edge) on CAPIN.
1	0 1	Any transition (rising or falling edge) on T3IN.
1	1 0	Any transition (rising or falling edge) on T3EUD.
1	1 1	Any transition (rising or falling edge) on T3IN or T3EUD.

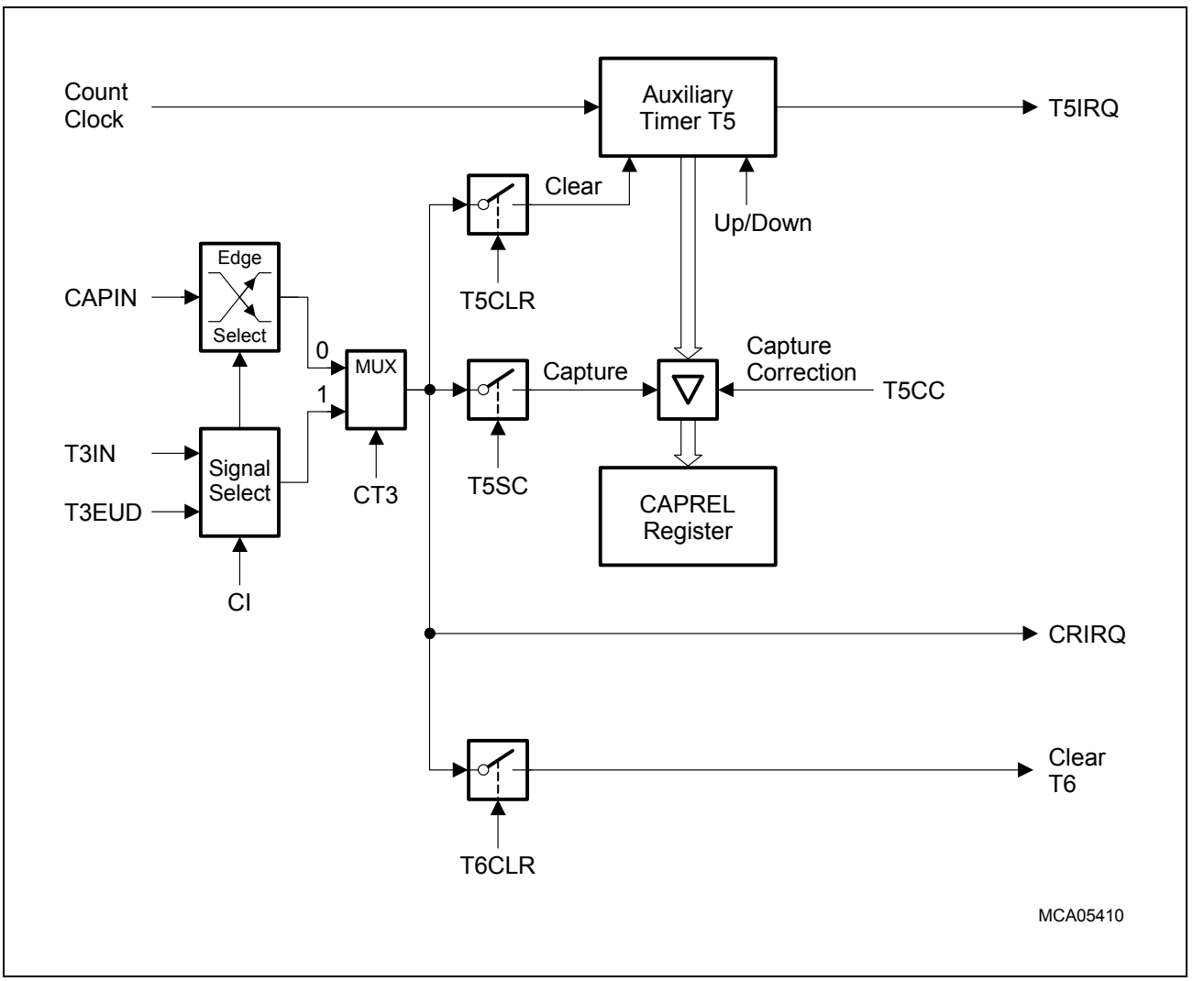


Figure 14-29 GPT2 Register CAPREL in Capture Mode

When a selected trigger is detected, the contents of the auxiliary timer T5 are latched into register CAPREL and the interrupt request line CRIRQ is activated. The same event can optionally clear timer T5 and/or timer T6. This option is enabled by bit T5CLR in register T5CON and bit T6CLR in register T6CON, respectively. If TxCLR = 0 the contents of timer Tx is not affected by a capture. If TxCLR = 1 timer Tx is cleared after the current timer T5 value has been latched into register CAPREL.

Note: Bit T5SC only controls whether or not a capture is performed. If T5SC is cleared the external input pin(s) can still be used to clear timer T5 and/or T6, or as external interrupt input(s). This interrupt is controlled by the CAPREL interrupt control register CRIC.

When capture triggers T3IN or T3EUD are enabled (CT3 = 1), register CAPREL captures the contents of T5 upon transitions of the selected input(s). These values can be used to measure T3's input signals. This is useful, for example, when T3 operates in

incremental interface mode, in order to derive dynamic information (speed, acceleration) from the input signals.

For capture mode operation, the selected pins CAPIN, T3IN, or T3EUD must be configured as input. To ensure that a transition of a trigger input signal applied to one of these inputs is recognized correctly, its level must be held high or low for a minimum number of module clock cycles, detailed in **Section 14.2.6**.

GPT2 Capture/Reload Register CAPREL in Reload Mode

Reload mode for register CAPREL is selected by setting bit T6SR in control register T6CON. In reload mode, the core timer T6 is reloaded with the contents of register CAPREL, triggered by an overflow or underflow of T6. This will not activate the interrupt request line CRIRQ associated with the CAPREL register. However, interrupt request line T6IRQ will be activated, indicating the overflow/underflow of T6.

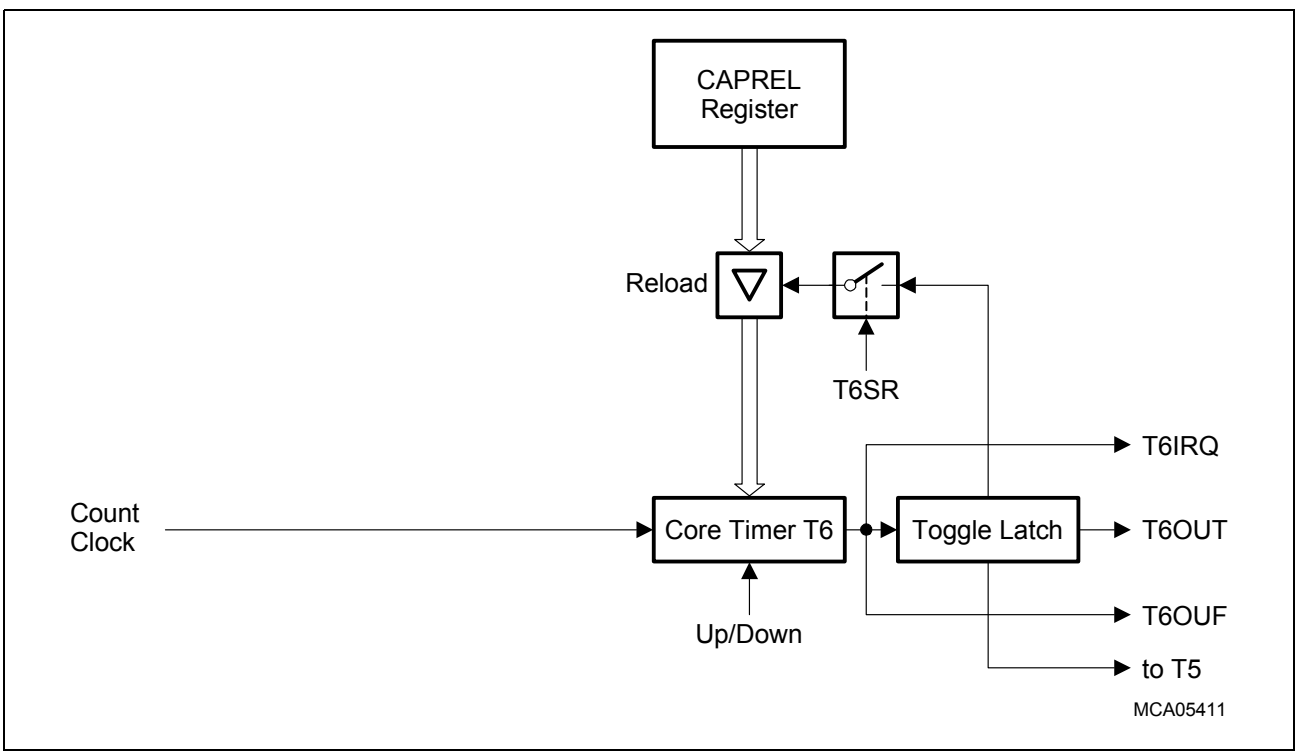


Figure 14-30 GPT2 Register CAPREL in Reload Mode

GPT2 Capture/Reload Register CAPREL in Capture-And-Reload Mode

Since the reload function and the capture function of register CAPREL can be enabled individually by bits T5SC and T6SR, the two functions can be enabled simultaneously by setting both bits. This feature can be used to generate an output frequency that is a multiple of the input frequency.

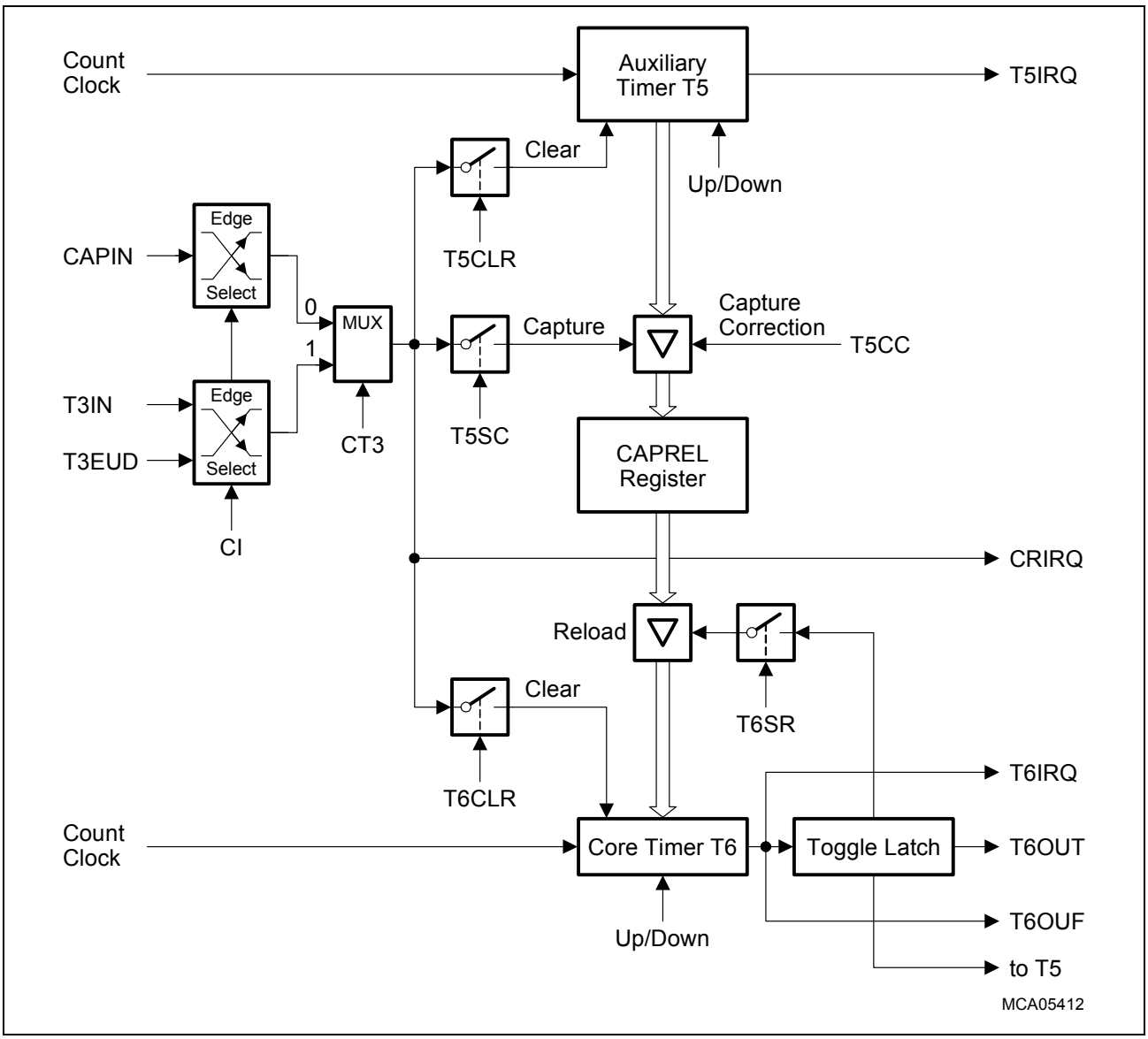


Figure 14-31 GPT2 Register CAPREL in Capture-And-Reload Mode

This combined mode can be used to detect consecutive external events which may occur aperiodically, but where a finer resolution, that means, more ‘ticks’ within the time between two external events is required.

For this purpose, the time between the external events is measured using timer T5 and the CAPREL register. Timer T5 runs in timer mode counting up with a frequency of e.g. $f_{GPT}/32$. The external events are applied to pin CAPIN. When an external event occurs,

the contents of timer T5 are latched into register CAPREL and timer T5 is cleared (T5CLR = 1). Thus, register CAPREL always contains the correct time between two events, measured in timer T5 increments. Timer T6, which runs in timer mode counting down with a frequency of e.g. $f_{GPT}/4$, uses the value in register CAPREL to perform a reload on underflow. This means, the value in register CAPREL represents the time between two underflows of timer T6, now measured in timer T6 increments. Since (in this example) timer T6 runs 8 times faster than timer T5, it will underflow 8 times within the time between two external events. Thus, the underflow signal of timer T6 generates 8 'ticks'. Upon each underflow, the interrupt request line T6IRQ will be activated and bit T6OTL will be toggled. The state of T6OTL may be output on pin T6OUT. This signal has 8 times more transitions than the signal which is applied to pin CAPIN.

Note: The underflow signal of Timer T6 can furthermore be used to clock one or more of the timers of the CAPCOM units, which gives the user the possibility to set compare events based on a finer resolution than that of the external events. This connection is accomplished via signal T6OUF.

Capture Correction

A certain deviation of the output frequency is generated by the fact that timer T5 will count actual time units (e.g. T5 running at 1 MHz will count up to the value $64_H/100_D$ for a 10 kHz input signal), while T6OTL will only toggle upon an underflow of T6 (i.e. the transition from 0000_H to $FFFF_H$). In the above mentioned example, T6 would count down from 64_H, so the underflow would occur after 101 timing ticks of T6. The actual output frequency then is 79.2 kHz, instead of the expected 80 kHz.

This deviation can be compensated for by activating the Capture Correction (T5CC = 1). If capture correction is active, the contents of T5 are decremented by 1 before being captured. The described deviation is eliminated (in the example, T5 would count up to the value $64_H/100_D$, but the CAPREL register will capture the decremented value $63_H/99_D$, T6 would count exactly 100 ticks, and the output frequency is 80 kHz).

14.2.6 GPT2 Clock Signal Control

All actions within the timer block GPT2 are triggered by transitions of its basic clock. This basic clock is derived from the system clock by a basic block prescaler, controlled by bitfield BPS2 in register T6CON (see **Figure 14-20**). The count clock can be generated in two different ways:

- **Internal count clock**, derived from GPT2's basic clock via a programmable prescaler, is used for (gated) timer mode.
- **External count clock**, derived from the timer's input pin(s), is used for counter mode.

For both ways, the basic clock determines the maximum count frequency and the timer's resolution:

Table 14-14 Basic Clock Selection for Block GPT2

Block Prescaler[1]	BPS2 = 01_B	BPS2 = 00_B[2]	BPS2 = 11_B	BPS2 = 10_B
Prescaling Factor for GPT2: F(BPS2)	F(BPS2) = 2	F(BPS2) = 4	F(BPS2) = 8	F(BPS2) = 16
Maximum External Count Frequency	$f_{GPT}/4$	$f_{GPT}/8$	$f_{GPT}/16$	$f_{GPT}/32$
Input Signal Stable Time	$2 \times t_{GPT}$	$4 \times t_{GPT}$	$8 \times t_{GPT}$	$16 \times t_{GPT}$

1) Please note the non-linear encoding of bitfield BPS2.

2) Default after reset.

Internal Count Clock Generation

In timer mode and gated timer mode, the count clock for each GPT2 timer is derived from the GPT2 basic clock by a programmable prescaler, controlled by bitfield TxI in the respective timer's control register TxCON.

The count frequency f_{Tx} for a timer Tx and its resolution r_{Tx} are scaled linearly with lower clock frequencies, as can be seen from the following formula:

$$f_{Tx} = \frac{f_{GPT}}{F(BPS2) \times 2^{<TxI>}} \qquad r_{Tx}[\mu s] = \frac{F(BPS2) \times 2^{<TxI>}}{f_{GPT}[MHz]} \tag{14.2}$$

The effective count frequency depends on the common module clock prescaler factor F(BPS2) as well as on the individual input prescaler factor $2^{<TxI>}$. **Table 14-15** summarizes the resulting overall divider factors for a GPT2 timer that result from these cascaded prescalers.

Table 14-15 GPT2 Overall Prescaler Factors for Internal Count Clock

Individual Prescaler for Tx	Common Prescaler for Module Clock[1]			
	BPS2 = 01_B	BPS2 = 00_B	BPS2 = 11_B	BPS2 = 10_B
TxI = 000_B	2	4	8	16
TxI = 001_B	4	8	16	32
TxI = 010_B	8	16	32	64
TxI = 011_B	16	32	64	128
TxI = 100_B	32	64	128	256
TxI = 101_B	64	128	256	512
TxI = 110_B	128	256	512	1024
TxI = 111_B	256	512	1024	2048

1) Please note the non-linear encoding of bitfield BPS2.

Table 14-16 lists a timer's parameters (such as count frequency, resolution, and period) resulting from the selected overall prescaler factor and the applied system frequency. Note that some numbers may be rounded.

Table 14-16 GPT2 Timer Parameters

System Clock = 10 MHz			Overall Divider Factor	System Clock = 40 MHz		
Frequency	Resolution	Period		Frequency	Resolution	Period
5.0 MHz	200 ns	13.11 ms	2	20.0 MHz	50 ns	3.28 ms
2.5 MHz	400 ns	26.21 ms	4	10.0 MHz	100 ns	6.55 ms
1.25 MHz	800 ns	52.43 ms	8	5.0 MHz	200 ns	13.11 ms
625.0 kHz	1.6 µs	104.9 ms	16	2.5 MHz	400 ns	26.21 ms
312.5 kHz	3.2 µs	209.7 ms	32	1.25 MHz	800 ns	52.43 ms
156.25 kHz	6.4 µs	419.4 ms	64	625.0 kHz	1.6 µs	104.9 ms
78.125 kHz	12.8 µs	838.9 ms	128	312.5 kHz	3.2 µs	209.7 ms
39.06 kHz	25.6 µs	1.678 s	256	156.25 kHz	6.4 µs	419.4 ms
19.53 kHz	51.2 µs	3.355 s	512	78.125 kHz	12.8 µs	838.9 ms
9.77 kHz	102.4 µs	6.711 s	1024	39.06 kHz	25.6 µs	1.678 s
4.88 kHz	204.8 µs	13.42 s	2048	19.53 kHz	51.2 µs	3.355 s

External Count Clock Input

The external input signals of the GPT2 block are sampled with the GPT2 basic clock (see **Figure 14-20**). To ensure that a signal is recognized correctly, its current level (high or low) must be held active for at least one complete sampling period, before changing. A signal transition is recognized if two subsequent samples of the input signal represent different levels. Therefore, a minimum of two basic clock periods are required for the sampling of an external input signal. Thus, the maximum frequency of an input signal must not be higher than half the basic clock.

Table 14-17 summarizes the resulting requirements for external GPT2 input signals.

Table 14-17 GPT2 External Input Signal Limits

System Clock = 10 MHz		Input Frequ. Factor	GPT2 Divider BPS1	Input Phase Duration	System Clock = 40 MHz	
Max. Input Frequency	Min. Level Hold Time				Max. Input Frequency	Min. Level Hold Time
2.5 MHz	200 ns	$f_{GPT}/4$	01_B	$2 \times t_{GPT}$	10.0 MHz	50 ns
1.25 MHz	400 ns	$f_{GPT}/8$	00_B	$4 \times t_{GPT}$	5.0 MHz	100 ns
625.0 kHz	800 ns	$f_{GPT}/16$	11_B	$8 \times t_{GPT}$	2.5 MHz	200 ns
312.5 kHz	1.6 µs	$f_{GPT}/32$	10_B	$16 \times t_{GPT}$	1.25 MHz	400 ns

These limitations are valid for all external input signals to GPT2, including the external count signals in counter mode and the gate input signals in gated timer mode.

14.2.7 GPT2 Timer Registers

GPT12E_Tx
Timer x Count Register **SFR ($FE4x_H/2y_H$)** **Reset Value: 0000_H**

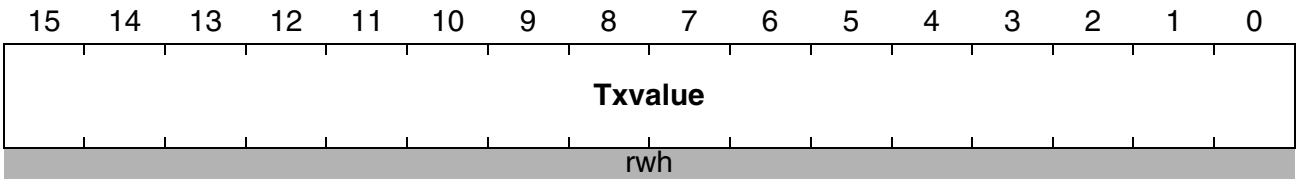
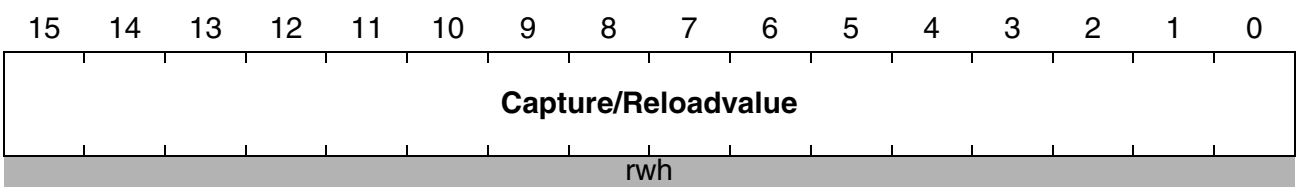


Table 14-18 GPT1 Timer Register Locations

Timer Register	Physical Address	8-Bit Address
T5	$FE46_H$	23_H
T6	$FE48_H$	24_H

GPT12E_CAPREL
Capture/Reload Register **SFR ($FE4A_H/25_H$)** **Reset Value: 0000_H**



14.2.8 Interrupt Control for GPT2 Timers and CAPREL

When a timer overflows from $FFFF_H$ to 0000_H (when counting up), or when it underflows from 0000_H to $FFFF_H$ (when counting down), its interrupt request flag (T5IR or T6IR) in register TxIC will be set. Whenever a transition according to the selection in bit field CI is detected at pin CAPIN, interrupt request flag CRIR in register CRIC is set. Setting any request flag will cause an interrupt to the respective timer or CAPREL interrupt vector (T5INT, T6INT or CRINT) or trigger a PEC service, if the respective interrupt enable bit (T5IE or T6IE in register TxIC, CRIE in register CRIC) is set. There is an interrupt control register for each of the two timers and for the CAPREL register.

GPT12E_T5IC
Timer 5 Intr. Ctrl. Reg. **SFR ($FF66_H/B3_H$)** **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	GPX	T5IR	T5IE	ILVL				GLVL	
-	-	-	-	-	-	-	rw	rwh	rw	rw				rw	

GPT12E_T6IC
Timer 6 Intr. Ctrl. Reg. **SFR ($FF68_H/B4_H$)** **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	GPX	T6IR	T6IE	ILVL				GLVL	
-	-	-	-	-	-	-	rw	rwh	rw	rw				rw	

GPT12E_CRIC
CAPREL Intr. Ctrl. Reg. **SFR ($FF6A_H/B5_H$)** **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	GPX	CRIR	CRIE	ILVL				GLVL	
-	-	-	-	-	-	-	rw	rwh	rw	rw				rw	

Note: Please refer to the general Interrupt Control Register description for an explanation of the control fields.

14.3 Interfaces of the GPT Module

Besides the described intra-module connections, the timer unit blocks GPT1 and GPT2 are connected to their environment in two basic ways (see **Figure 14-32**):

- **Internal connections** interface the timers with on-chip resources such as clock generation unit, interrupt controller, or other timers.
- **External connections** interface the timers with external resources via port pins.

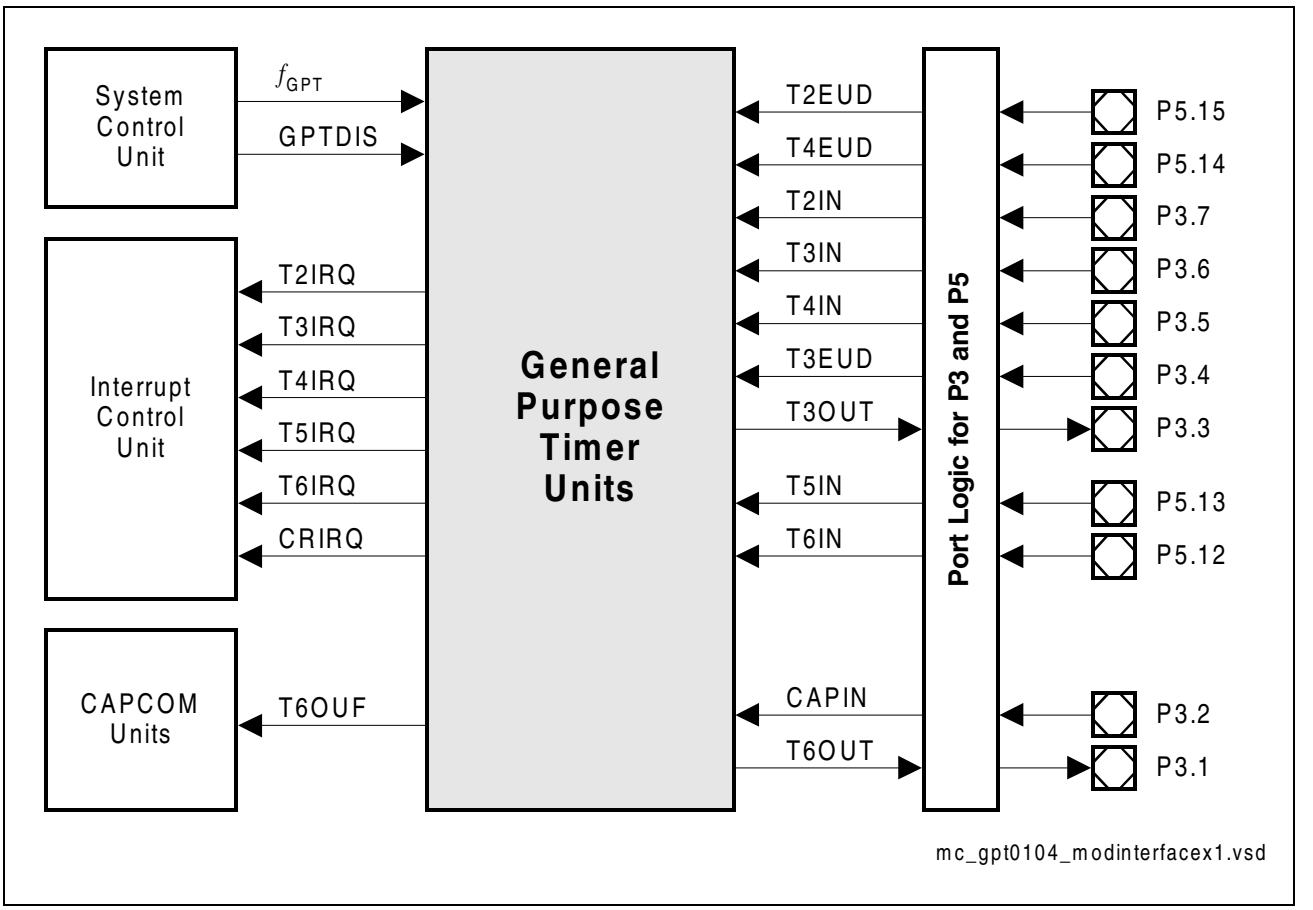


Figure 14-32 GPT Module Interfaces

Port pins to be used for timer input signals must be switched to input, the respective direction control bits must be cleared (DPx.y = 0).

Port pins to be used for timer output signals must be switched to output, the respective direction control bits must be set (DPx.y = 1). The alternate timer output signal must be selected for these pins via the respective alternate select registers (see **Chapter 7**).

Interrupt nodes to be used for timer interrupt requests must be enabled and programmed to a specific interrupt level.

15 Real Time Clock

The Real Time Clock (RTC) module of the XC161 basically consists of a chain of prescalers and timers. Its count clock is derived from the auxiliary oscillator or from the prescaled main oscillator. The RTC serves various purposes:

- 48-bit timer for long term measurements
- System clock to determine the current time and date
  (the RTC's structure supports the direct representation of time and date)
- Cyclic time based interrupt (can be generated by any timer of the chain)

A number of programming options as well as interrupt request signals adjust the operation of the RTC to the application's requirements. The RTC can continue its operation while the XC161 is in a power-saving mode, such that real time date and time information is provided.

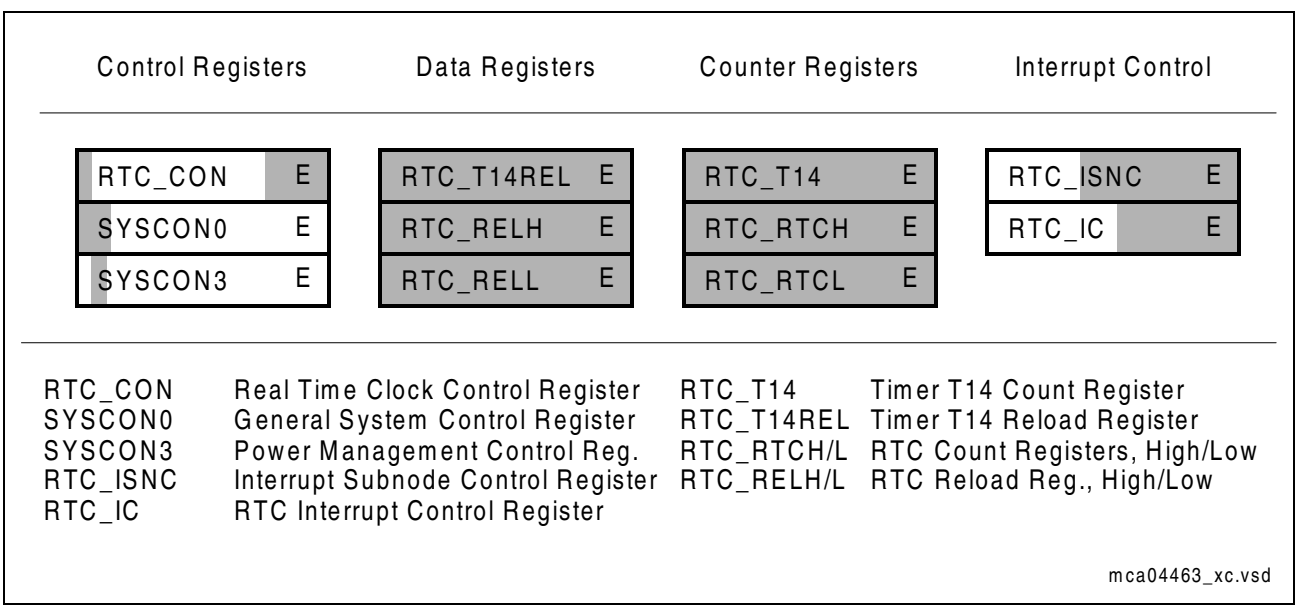


Figure 15-1 SFRs Associated with the RTC Module

The RTC module consists of a chain of 3 divider blocks:

- a selectable 8:1 divider (on - off)
- the reloadable 16-bit timer T14
- the 32-bit RTC timer block (accessible via RTC_RTCH and RTC_RTCL), made of:
  – the reloadable 10-bit timer CNT0
  – the reloadable 6-bit timer CNT1
  – the reloadable 6-bit timer CNT2
  – the reloadable 10-bit timer CNT3

All timers count upwards. Each of the five timers can generate an interrupt request. All requests are combined to a common node request.

Note: The RTC registers are not affected by a system reset in order to maintain the correct system time even when intermediate resets are executed.

15.1 Defining the RTC Time Base

The timer chain of the RTC is clocked with the count clock signal f_{RTC} which is derived from the auxiliary oscillator or from the prescaled main oscillator (see **Figure 15-2** and **Figure 15-3**). Optionally prescaled by a factor of 8, this is the basic RTC clock. Depending on the operating mode, timer T14 may provide the count increments used by the application and thus determine the input frequency of the RTC timer, that is, the RTC time base (see also **Table 15-3**).

The RTC is also supplied with the system clock f_{SYS} of the XC161. This clock signal is used to control the RTC's logic blocks and its bus interface. To synchronize properly to the count clock, the system clock must run at least four times faster than the count clock, this means $f_{SYS} \geq 4 \times f_{CNT}$.

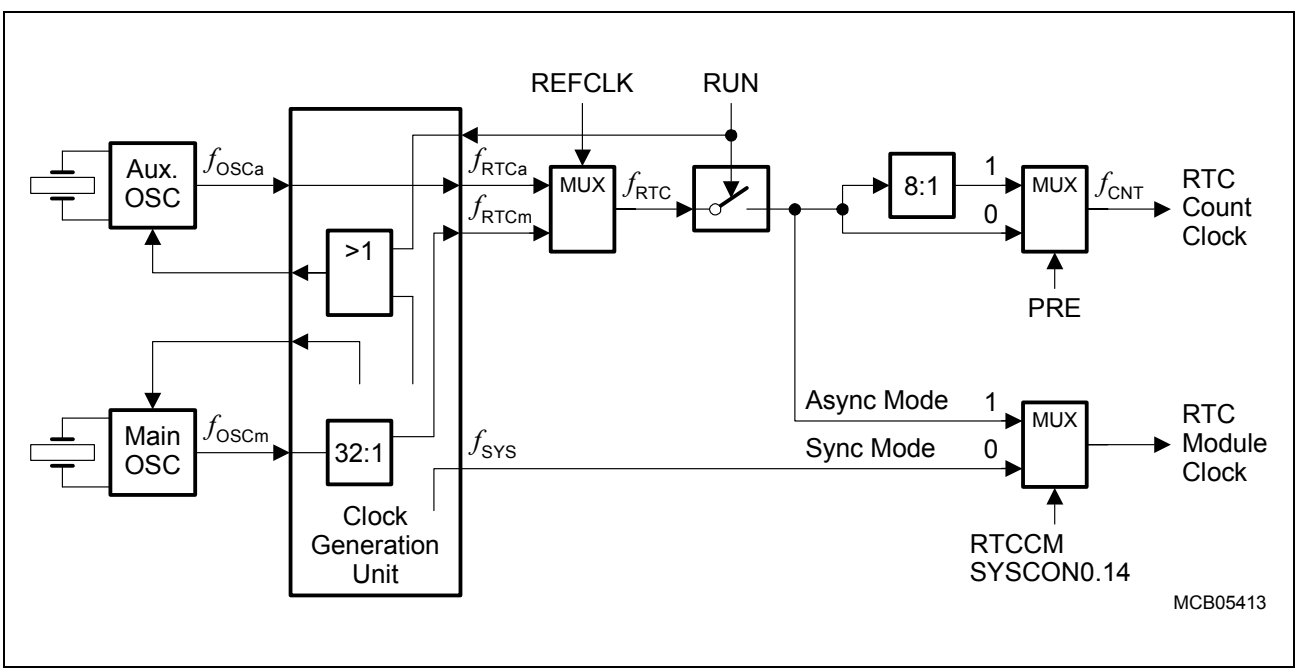


Figure 15-2 RTC Clock Supply Block Diagram

For an example, **Table 15-1** lists the interrupt period range and the T14 reload values (for a time base of 1 s and 1 ms):

Table 15-1 RTC Time Base Examples

Oscillator Frequency	T14 Intr. Period		Reload Value A		Reload Value B	
	Min.	Max.	T14REL	Base	T14REL	Base
32.768 kHz	30.52 µs	16.0 s	8000_H/$F000_H$	1.000 s	$FFDF_H$/ $FFFC_H$	1.007 ms/ 0.977 ms

Note: Select one value from the reload value pairs, depending if the 8:1 prescaler is disabled/enabled.

Asynchronous Operation

When the system clock frequency becomes lower than $4 \times f_{CNT}$ proper synchronization is not possible and count events may be missed. When the XC161 enters e.g. sleep mode the system clock stops completely and the RTC would stop counting.

In these cases the RTC can be switched to Asynchronous Mode (by setting bit RTCCM in register SYSCON0). In this mode the count registers are directly controlled by the count clock independent of the system clock (hence the name). Asynchronous operation ensures correct time-keeping even during sleep mode or powerdown mode.

However, as no synchronization between the count registers and the bus interface can be maintained in asynchronous mode, the RTC registers cannot be written. Read accesses may interfere with count events and, therefore, must be verified (e.g. by reading the same value with three consecutive read accesses).

Note: The access restrictions in asynchronous mode are only meaningful if the system clock is not switched off, of course.

Switching Clocking Modes

The clocking mode of the RTC (synchronous or asynchronous) is selected via bit RTCCM in register SYSCON0. After reset, the RTC operates in Synchronous Mode (RTCCM = 0) with the 8:1 prescaler enabled.

The selected clocking mode also affects the access to RTC registers. Bit ACCPOS in register RTC_CON indicates if full register access is possible (ACCPOS = 1, default after reset) or not (ACCPOS = 0). This also indicates the current clocking mode.

Attention: Software should poll bit ACCPOS to determine the proper transition to the intended clocking mode.

After switching to Asynchronous Mode (RTCCM = 1), bit ACCPOS = 0 indicates proper operation in Asynchronous Mode. In this case the system clock can be stopped or reduced.

After switching to Synchronous Mode, (RTCCM = 0), bit ACCPOS = 1 indicates proper operation in Synchronous Mode. In this case the RTC registers can again be accessed properly (read and write).

Note: The RTC might lose a counting event (edge of f_{CNT}) when switching from synchronous mode to asynchronous mode while the 8:1 prescaler is disabled. For these applications it is, therefore, recommended to set up the RTC with the 8:1 prescaler enabled.

Increased RTC Accuracy through Software Correction

The accuracy of the XC161's RTC is determined by the oscillator frequency and by the respective prescaling factor (excluding or including T14 and the 8:1 prescaler). The accuracy limit generated by the prescaler is due to the quantization of a binary counter (where the average is zero), while the accuracy limit generated by the oscillator frequency is due to the difference between the ideal and real frequencies (and therefore accumulates over time). This effect is predictable and can be compensated. The total accuracy of the RTC can be further increased via software for specific applications that demand a high time accuracy.

The key to the improved accuracy is knowledge of the exact oscillator frequency. The relation of this frequency to the expected ideal frequency is a measure of the RTC's deviation. The number of cycles, N, after which this deviation causes an error of ±1 cycle can be easily computed. So, the only action is to correct the count by ±1 after each series of N cycles. The correction may be made cyclically, for instance, within an interrupt service routine, or by evaluating a formula when the RTC registers are read (for this the respective "last" RTC value must be available somewhere).

Note: For the majority of applications, however, the standard accuracy provided by the RTC's structure will be more than sufficient.

Adjusting the current RTC value would require reading and then writing the complete 48-bit value. This can only be accomplished by three successive accesses each. To avoid the hassle of reading/writing multi-word values, the RTC incorporates a correction option to simply add or suppress one count pulse.

This is done by setting bit T14INC or T14DEC, respectively, in register RTC_CON. This will add an extra count pulse (T14INC) upon the next count event, or suppress the next count event (T14DEC). The respective bit remains set until its associated action has been performed and is automatically cleared by hardware after this event.

Note: Setting both bits, T14INC and T14DEC, at the same time will have no effect on the count values.

15.2 RTC Run Control

If the RTC shall operate bit RUN in register RTC_CON must be set (default after reset). Bit RUN can be cleared, for example, to exclude certain operation phases from time keeping. The RTC can be completely disabled by setting the corresponding bit RTCDIS in register SYSCON3.

Note: A valid count clock is required for proper RTC operation, of course.

A reset for the RTC is triggered via software by setting bit RTCRST in register SYSCON0. In this case all RTC registers are set to their initial values and bit RTCRST is cleared automatically. A normal system reset does not affect the RTC registers and its operation (RTC_IC will be reset, however). The initialization software must ensure the proper RTC operating mode.

The RTC control register RTC_CON selects the basic operation of the RTC module.

RTC_CON
Control Register **ESFR ($F110_H/88_H$)** **Reset Value: 8003_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ACC POS	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**REF CLK**	**T14 INC**	**T14 DEC**	**PRE**	**RUN**
rh	-	-	-	-	-	-	-	-	-	-	rw	rwh	rwh	rw	rw

Field	Bits	Type	Description
ACCPOS	15	rh	**RTC Register Access Possible** 0 No write access is possible, only asynchronous reads 1 Registers can be read and written
REFCLK	4	rw	**Reference Clock Source** 0 The RTC count clock is derived from the auxiliary oscillator (f_{OSCa}) 1 The RTC count clock is derived from the main oscillator (f_{OSCm}/32)
T14INC	3	rwh	**Increment Timer T14 Value** Setting this bit to 1 adds one count pulse upon the next count event, thus incrementing T14. This bit is cleared by hardware after incrementation.
T14DEC	2	rwh	**Decrement Timer T14 Value** Setting this bit to 1 suppresses the next count event, thus decrementing T14. This bit is cleared by hardware after decrementation.

Field	Bits	Type	Description
PRE	1	rw	**RTC Input Source Prescaler Enable** 0 Prescaler disabled, T14 clocked with f_{RTC} 1 Prescaler enabled, T14 clocked with $f_{RTC}/8$
RUN	0	rw	**RTC Run Bit** 0 RTC stopped 1 RTC runs

15.3 RTC Operating Modes

The RTC can be configured for several operating modes according to the purpose it is meant to serve. These operating modes are configured by selecting appropriate reload values and interrupt signals.

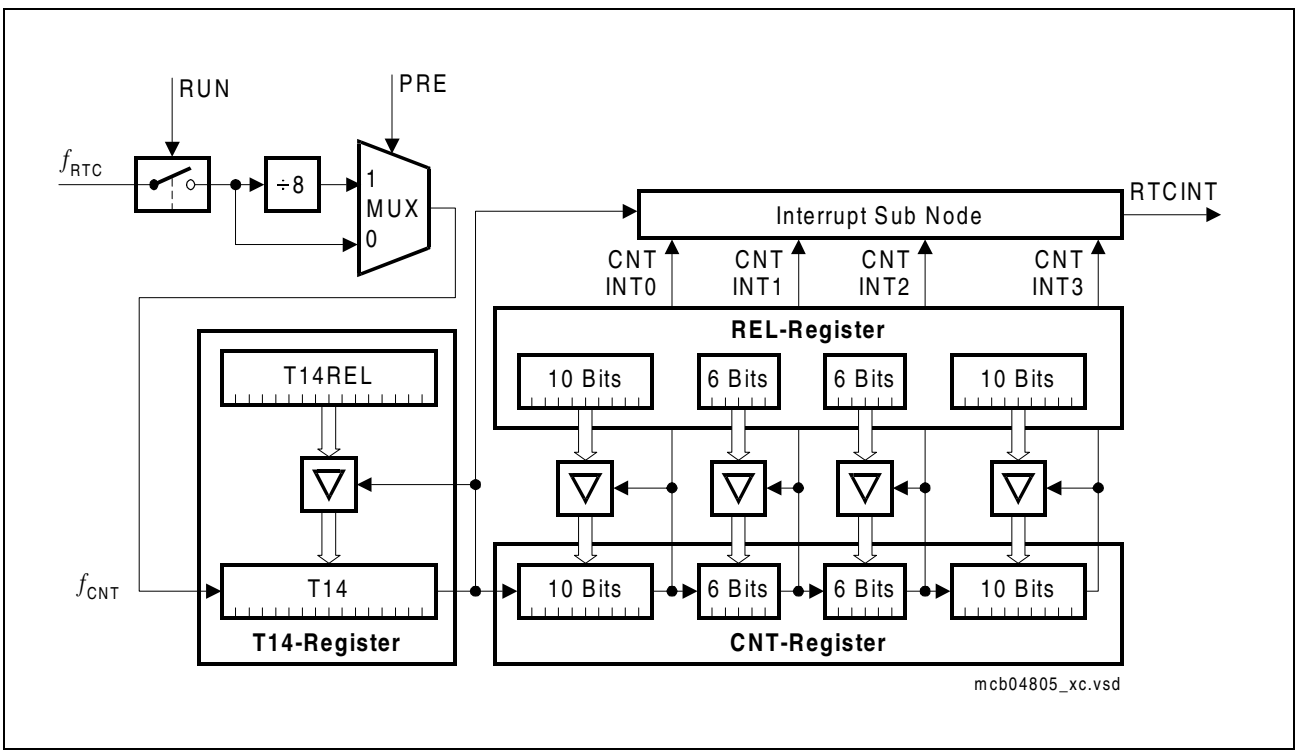


Figure 15-3 RTC Block Diagram

RTC Register Access

The actual value of the RTC is indicated by the three registers T14, RTCL, and RTCH. As these registers are concatenated to build the RTC counter chain, internal overflows occur while the RTC is running. When reading or writing the RTC value, such internal overflows must be taken into account to avoid reading/writing corrupted values.

Care must be taken, when reading the timer(s), as this requires up to three read accesses to the different registers with an inherent time delay between the accesses. An overflow from T14 to RTCL and/or from RTCL to RTCH might occur between the accesses, which needs to be taken into account appropriately.

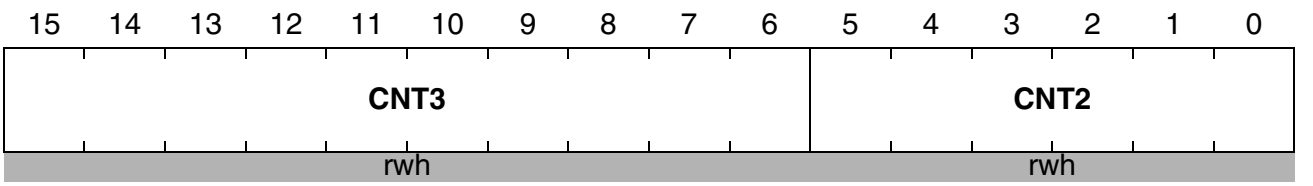
For example, reading/writing 0000_H to RTCH and then accessing RTCL could produce a corrupted value as RTCL may overflow before it can be accessed. In this case, RTCH would be 0001_H. The same precautions must be taken for T14 and T14REL.

Timer T14 and its reload register are accessed via dedicated locations. The four RTC counters CNT3 … CNT0 are accessed via the two 16-bit RTC timer registers, RTCH and RTCL. The associated four reload values REL3 … REL0 are accessed via the two 16-bit RTC reload registers, RELH and RELL.

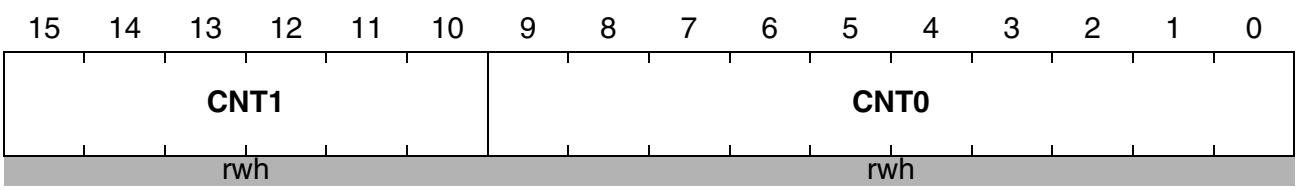
Table 15-2 Register Locations for Timer T14

Register Name	Long/Short Address	Reset Value	Notes
RTC_T14	$F0D2_H/69_H$	0000_H	16-bit timer, can be used as prescaler for the RTC block
RTC_T14REL	$F0D0_H/68_H$	0000_H	Timer T14 reload register

RTC_RTCH
RTC Timer High Register **ESFR ($F0D6_H/6B_H$)** **Reset Value: 0000_H**



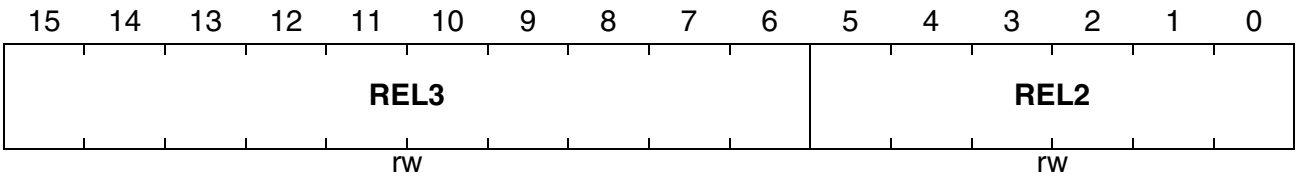
RTC_RTCL
RTC Timer Low Register **ESFR ($F0D4_H/6A_H$)** **Reset Value: 0000_H**



Field	Bits	Type	Description
CNTx (x = 3 … 0)	[15:6] [5:0] [15:10] [9:0]	rwh	**RTC Timer Count Section CNTx** An overflow of this bitfield triggers a count pulse to the next count section CNTx+1 (except for CNT3) followed by a reload of CNTx from bitfield RELx. In addition, an interrupt request is triggered.

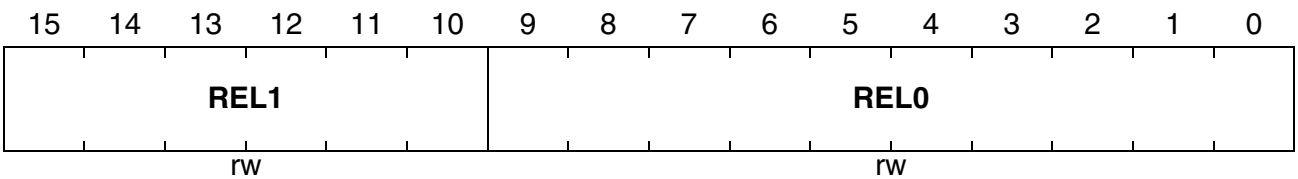
RTC_RELH

RTC Reload High Register **ESFR ($F0CE_H$/67_H)** **Reset Value: 0000_H**



RTC_RELL

RTC Reload Low Register **ESFR ($F0CC_H$/66_H)** **Reset Value: 0000_H**



Field	Bits	Type	Description
RELx **(x = 3 … 0)**	[15:6] [5:0] [15:10] [9:0]	rw	**RTC Reload Value RELx** This bitfield is copied to bitfield CNTx upon an overflow of count section CNTx.

Note: The registers of the RTC receive their reset values only upon a specific RTC reset. This reset is not triggered upon a system reset, but via software.

15.3.1 48-bit Timer Operation

The concatenation of timers T14 and COUNT0 … COUNT3 can be regarded as a 48-bit timer which is clocked with the RTC input frequency, optionally divided by the prescaler. The reload registers T14REL, RELL, and RELH must be cleared to produce a true binary 48-bit timer. However, any other reload value may be used. Reload values other than zero must be used carefully, due to the individual sections of the RTC timer with their own individual overflows and reload values.

The maximum usable timespan is 2^{48} ($\approx 10^{14}$) T14 input clocks (assuming no prescaler), which would equal more than 200 years at an oscillator frequency of 32 kHz.

15.3.2 System Clock Operation

A real time system clock can be maintained that keeps on running also during power saving modes (optionally) and indicates the current time and date. This is possible because the RTC module is not affected by a system reset[1)].

The resolution for this clock information is determined by the input clock of timer T14. By selecting appropriate reload values each cascaded timer can represent directly a part of the current time and/or date. Due to its width, T14 can adjust the RTC to the intended range of operation (time or date). The maximum usable timespan is achieved when T14REL is loaded with 0000_H and so T14 divides by 2^{16}.

System Clock Example

The RTC count clock is f_{OSCa} (8:1 prescaler off). By selecting appropriate reload values the RTC timers directly indicate the current time (see **Figure 15-4** and **Table 15-3**).

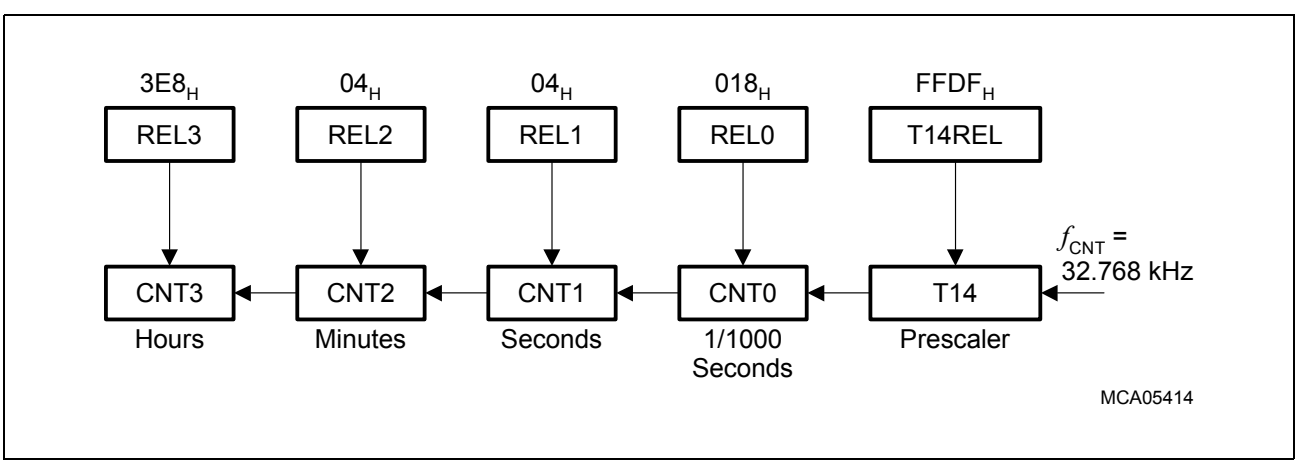


Figure 15-4 RTC Configuration Example

Note: This setup can generate an interrupt request every millisecond, every second, every minute, every hour, or every day.

1) After a power on reset, however, the RTC registers are undefined.

Each timer in the chain divides the clock by (2<timer_width> - <reload_value>) : 1, as the timers count up. **Table 15-3** shows the reload values which must be chosen for a specific scenario (i.e. operating mode of the RTC).

Table 15-3 Reload Value Scenarios

		REL3	REL2	REL1	REL0	T14REL
Time of Day (Figure 15-4)	Formula	2^{10} - 24	2^{6} - 60	2^{6} - 60	2^{10} - 1000	2^{16} - 33
	Rel. Value	$3E8_H$	04_H	04_H	018_H	$FFDF_H$
	Function	h	m	s	1/1000 s	Prescaler
	Intr. Period	day	hour	minute	second	millisec.[1]
Day of the Week	Formula	2^{10} - 7	2^{6} - 24	2^{6} - 60	2^{10} - 60	2^{16} - 32768
	Rel. Value	$3F9_H$	28_H	04_H	$3C4_H$	8000_H
	Function	day	h	m	s	Prescaler
	Intr. Period	week	day	hour	minute	second

1) T14 errors in the first example (ms) can be compensated either by choosing an adapted value for REL0, or by using software correction.

15.3.3 Cyclic Interrupt Generation

The RTC module can generate an interrupt request whenever one of the timers overflows and is reloaded. This interrupt request may be used, for example, to provide a system time tick independent of the CPU frequency without loading the general purpose timers, or to wake up regularly from sleep mode. The interrupt cycle time can be adjusted by choosing appropriate reload values and by enabling the appropriate interrupt request.

In this mode, the other operating modes can be combined. For example, a reload value of T14REL = $F9C0_H$ (2^{16} - 1600) generates a T14 interrupt request every 50 ms to wake-up the system regularly. Still the subsequent timers can be configured to represent the time or build a binary counter, however with a different time base.

15.4 RTC Interrupt Generation

The overflow signals of each timer of the RTC timer chain can generate an interrupt request. The RTC's interrupt subnode control register ISNC combines these requests to activate the common RTC interrupt request line RTC_IRQ.

Each timer overflow sets its associated request flag in register ISNC. Individual enable bits for each request flag determine whether this request also activates the common interrupt line. The enabled requests are ORed together on this line (see **Figure 15-5**).

The interrupt handler can determine the source of an interrupt request via the specific request flags and must clear them after appropriate processing (not cleared by hardware). The common node request bit is automatically cleared when the interrupt handler is vectored to.

Note: If only one source is enabled, no additional software check is required, of course. Both the individual request and the common interrupt node must be enabled.

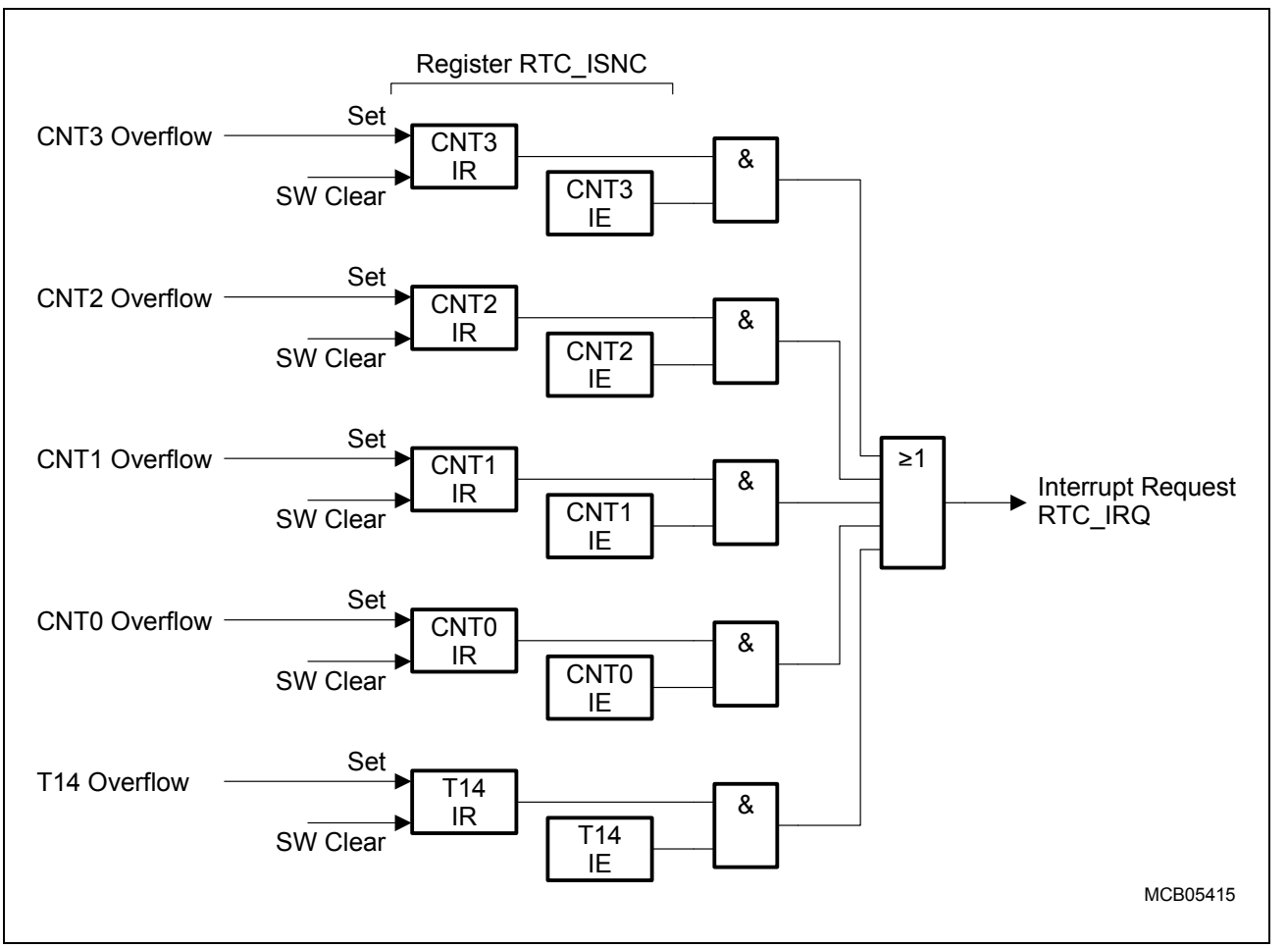


Figure 15-5 Interrupt Block Diagram

RTC_ISNC
Interrupt Subnode Ctrl. Reg. **ESFR ($F10C_H/86_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	CNT 3IR	CNT 3IE	CNT 2IR	CNT 2IE	CNT 1IR	CNT 1IE	CNT 0IR	CNT 0IE	T14 IR	T14 IE
-	-	-	-	-	-	rwh	rw	rwh	rw	rwh	rw	rwh	rw	rwh	rw

Field	Bits	Type	Description
CNTxIR (x = 3 … 0)	9, 7, 5, 3	rwh	**Section CNTx Interrupt Request Flag** 0 No request pending 1 This source has raised an interrupt request
CNTxIE (x = 3 … 0)	8, 6, 4, 2	rw	**Section CNTx Interrupt Enable Control Bit** 0 Interrupt request is disabled 1 Interrupt request is enabled
T14IR	1	rwh	**T14 Overflow Interrupt Request Flag** 0 No request pending 1 This source has raised an interrupt request
T14IE	0	rw	**T14 Overflow Interrupt Enable Control Bit** 0 Interrupt request is disabled 1 Interrupt request is enabled

Note: The interrupt request flags in register ISNC must be cleared by software. They are not cleared automatically when the service routine is entered.

RTC_IC
RTC Interrupt Ctrl. Reg. **ESFR ($F1A0_H/D0_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	GPX	RTC IR	RTC IE	ILVL				GLVL	
-	-	-	-	-	-	-	rw	rwh	rw	rw				rw	

Note: Please refer to the general Interrupt Control Register description for an explanation of the control fields.
Register RTC_IC is not part of the RTC module and is reset with any system reset.

16 The Analog/Digital Converter

The XC161 provides an Analog/Digital Converter with 8-bit or 10-bit resolution and a sample & hold circuit on-chip. An input multiplexer selects between up to 12 analog input channels (alternate functions of Port 5) either via software (fixed channel modes) or automatically (auto scan modes).

To fulfill most requirements of embedded control applications the ADC supports the following conversion modes:

- **Fixed Channel Single Conversion**
  produces just one result from the selected channel
- **Fixed Channel Continuous Conversion**
  repeatedly converts the selected channel
- **Auto Scan Single Conversion**
  produces one result from each of a selected group of channels
- **Auto Scan Continuous Conversion**
  repeatedly converts the selected group of channels
- **Wait for ADDAT Read Mode**
  start a conversion automatically when the previous result was read
- **Channel Injection Mode**
  start a conversion when a hardware trigger occurs,
  can insert the conversion of a specific channel into a group conversion (auto scan)

A set of SFRs and port pins provide access to control functions and results of the ADC. The enhanced-mode registers provide more detailed control functions for the ADC.

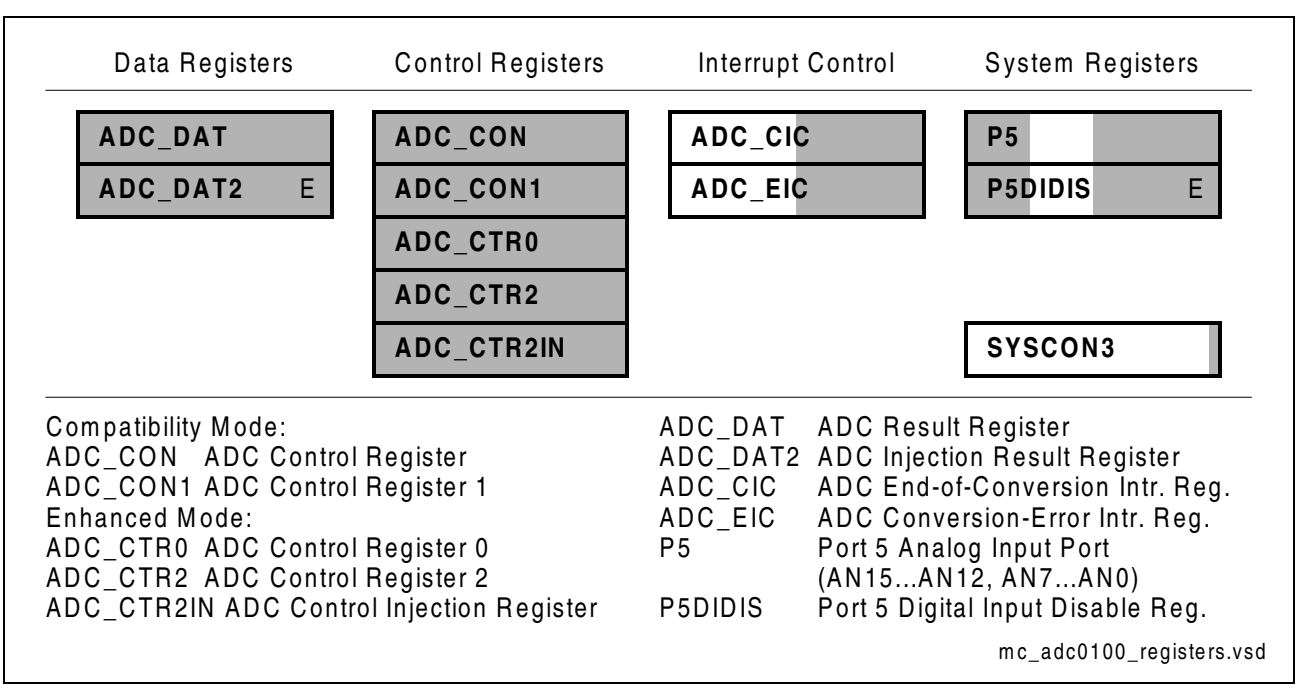


Figure 16-1 SFRs and Port Pins Associated with the A/D Converter

The external analog reference voltages V_{AREF} and V_{AGND} are fixed. The separate supply for the ADC reduces the interference with other digital signals. The reference voltages must be stable during the reset calibration phase and during an entire conversion, to achieve a maximum of accuracy.

The sample time as well as the conversion time is programmable, so the ADC can be adjusted to the internal resistances of the analog sources and/or the analog reference voltage supply (you may also want to refer to application note AP2428).

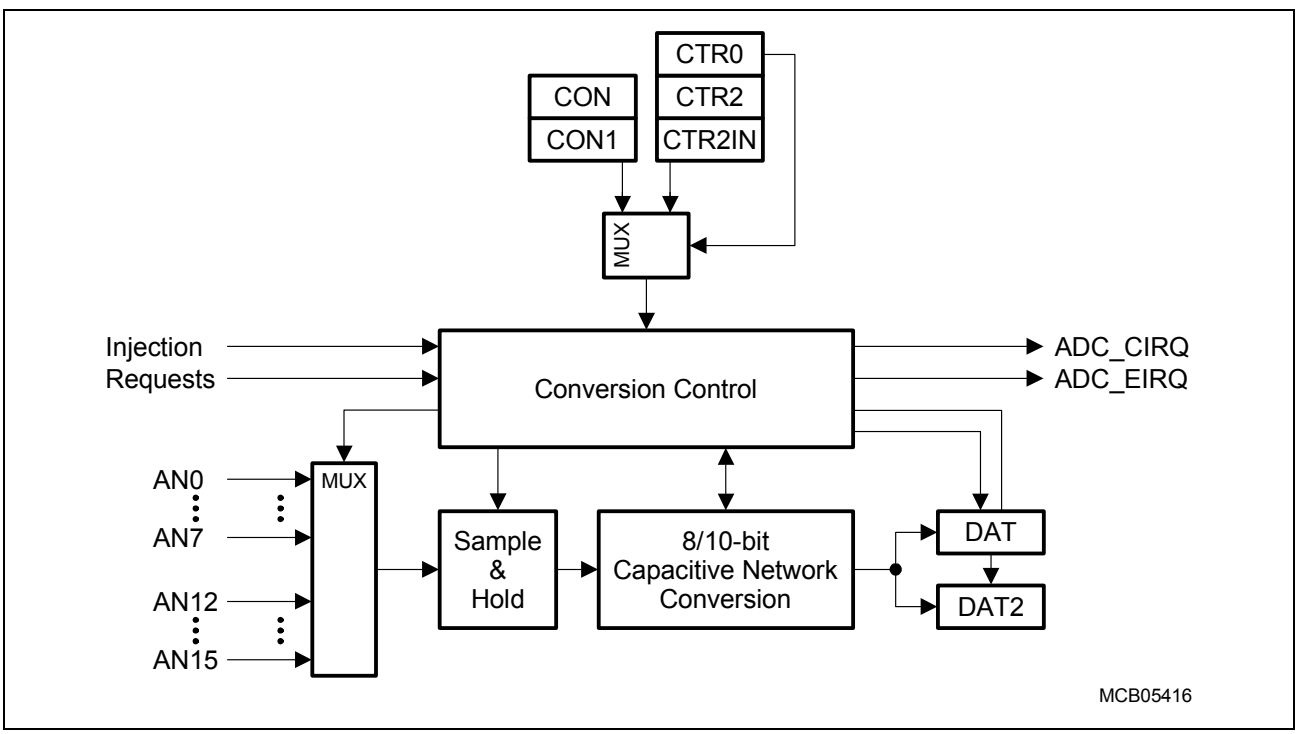


Figure 16-2 Analog/Digital Converter Block Diagram

The ADC is implemented as a capacitive network using successive approximation conversion. A conversion consists of 3 phases.

- During the sample phase, the capacitive network is connected to the selected analog input and is charged or discharged to the voltage of the analog signal.
- During the actual conversion phase, the network is disconnected from the analog input and is repeatedly charged or discharged via V_{AREF} during the steps of successive approximation.
- After the (optional) post-calibration phase (to adjust the network to changing conditions such as temperature) the result is written to the result register and an interrupt request is generated.

There are two sets of control, data, and status registers, one set for compatibility mode and one set for enhanced mode. Only one of these register sets may be active at a given time. As most of the bits and bitfields of the registers of the two sets control the same functionality or control the functionality in a very similar way, the following description is organized according to the functionality, not according to the two register sets.

16.1 Mode Selection

The analog input channels AN15 … AN12, AN7 … AN0 are alternate functions of Port 5 which is an input-only port. The Port 5 lines may either be used as analog or digital inputs. For pins that shall be used as analog inputs it is recommended to disable the digital input stage via register P5DIDIS. This avoids undesired cross currents and switching noise while the (analog) input signal level is between V_{IL} and V_{IH}.

The functions of the A/D converter are controlled by two sets of bit-addressable control registers. In compatibility mode, registers ADC_CON and ADC_CON1 are used, in enhanced mode, registers ADC_CTR0, ADC_CTR2, and ADC_CTR2IN are used. Their bitfields specify the analog channel to be acted upon, the conversion mode, and also reflect the status of the converter.

16.1.1 Compatibility Mode

In compatibility mode (MD = 0), registers ADC_CON and ADC_CON1 select the basic functions. The register layout is compatible with previous versions of the ADC module, while providing limited options.

ADC_CON
ADC Control Register **SFR ($FFA0_H/D0_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADCTC		ADSTC		AD CRQ	AD CIN	AD WR	AD BSY	AD ST	-	ADM		ADCH			
rw		rw		rwh	rw	rw	rwh	rwh	-	rw		rw			

Field	Bits	Type	Function
ADCTC	[15:14]	rw	**ADC Conversion Time Control** (Defines the ADC basic conversion clock f_{BC}) 00 $f_{BC} = f_{ADC}/4$ 01 $f_{BC} = f_{ADC}/2$ 10 $f_{BC} = f_{ADC}/16$ 11 $f_{BC} = f_{ADC}/8$
ADSTC	[13:12]	rw	**ADC Sample Time Control** (Defines the ADC sample time in a certain range) 00 $t_{BC} \times 8$ 01 $t_{BC} \times 16$ 10 $t_{BC} \times 32$ 11 $t_{BC} \times 64$
ADCRQ	11	rwh	**ADC Channel Injection Request Flag**
ADCIN	10	rw	**ADC Channel Injection Enable**

Field	Bits	Type	Function
ADWR	9	rw	**ADC Wait for Read Control**
ADBSY	8	rh	**ADC Busy Flag** 0 ADC is idle 1 A conversion is active
ADST	7	rwh	**ADC Start Bit** 0 Stop a running conversion 1 Start conversion(s)
ADM	[5:4]	rw	**ADC Mode Selection** 00 Fixed Channel Single Conversion 01 Fixed Channel Continuous Conversion 10 Auto Scan Single Conversion 11 Auto Scan Continuous Conversion
ADCH	[3:0]	rw	**ADC Analog Channel Input Selection** Selects the (first) ADC channel which is to be converted.

ADC_CON1
ADC Control Register 1 **SFR ($FFA6_H/D3_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ICST	SAMPLE	CAL	RES	ADCTC						ADSTC					
rw	rh	rh	rw	rw						rw					

Field	Bits	Type	Description
ICST	15	rw	**Improved Conversion and Sample Timing** Selects the active timing control bitfields 0 Standard conversion and sample time control, 2-bit fields in ADC_CON (default after reset) 1 Improved conversion and sample time control, 6-bit fields in ADC_CON1
SAMPLE	14	rh	**Sample Phase Status Flag** 0 A/D Converter is not in sampling 1 A/D Converter is currently in the sample phase
CAL	13	rh	**Reset Calibration Phase Status Flag** 0 A/D Converter is not in calibration phase 1 A/D Converter is in calibration phase

Field	Bits	Type	Description
RES	12	rw	**Conversion Resolution Control** 0 10-bit resolution (default after reset) 1 8-bit resolution
ADCTC	[11:6]	rw	**ADC Conversion Time Control** Defines the ADC basic conversion clock: $f_{BC} = f_{ADC}$ / (<ADCTC> + 1)
ADSTC	[5:0]	rw	**ADC Sample Time Control** Defines the ADC sample time: $t_S = t_{BC} \times 4 \times$ (<ADSTC> + 1)

Note: The limit values for f_{BC} (see data sheet) must not be exceeded when selecting ADCTC and f_{ADC}.

16.1.2 Enhanced Mode

In enhanced mode (MD = 1), registers ADC_CTR0, ADC_CTR2, and ADC_CTR2IN select the basic functions. The register layout differs from the compatibility-mode layout, but this mode provides more options.

Conversion timing is selected via registers ADC_CTR2(IN), where ADC_CTR2 controls standard conversions and ADC_CTR2IN controls injected conversions.

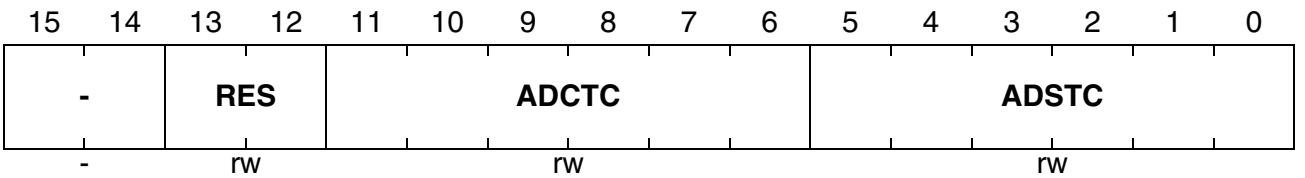
ADC_CTR0
ADC Control Register 0 **SFR ($FFBE_H/DF_H$)** **Reset Value: 1000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MD	**SAMPLE**	**ADCTS**		**ADCRQ**	**ADCIN**	**ADWR**	**ADBSY**	**ADST**	**ADM**		**CALOFF**	**ADCH**			
rw	rh	rw		rwh	rw	rw	rh	rwh	rw		rw	rw			

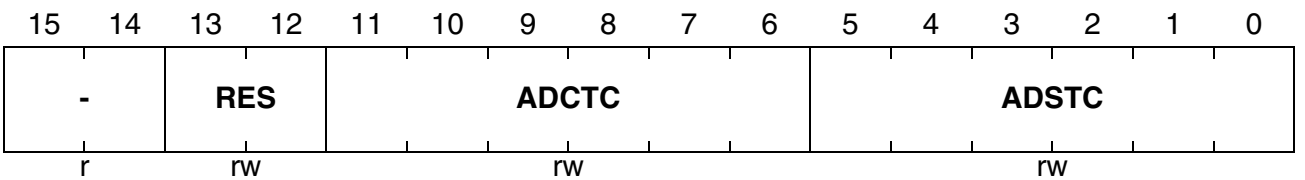
Field	Bits	Type	Description
MD	15	rw	**Mode Control** 0 Compatibility Mode 1 Enhanced Mode *Note: Any modification of control bit MD is forbidden while a conversion is currently running. User software must take care.*
SAMPLE	14	rh	**Sample Phase Status Flag** 0 A/D Converter is not in sample phase 1 A/D Converter in sample phase

Field	Bits	Type	Description
ADCTS	[13:12]	rw	**Channel Injection Trigger Input Select** 00 Channel injection trigger input disabled 01 Trigger input CAPCOM selected 10 Reserved 11 Reserved *Note: Reset value of bitfield ADCTS is 01_B for compatibility purposes.*
ADCRQ	11	rwh	**Channel Injection Request Flag**
ADCIN	10	rw	**Channel Injection Enable Control**
ADWR	9	rw	**Wait for Read Control**
ADBSY	8	rh	**Busy Flag** 0 ADC is idle 1 A conversion is active
ADST	7	rwh	**ADC Start/Stop Control** 0 Stop a running conversion 1 Start conversion(s)
ADM	[6:5]	rw	**Mode Selection Control** 00 Fixed Channel Single Conversion 01 Fixed Channel Continuous Conversion 10 Auto Scan Single Conversion 11 Auto Scan Continuous Conversion
CALOFF	4	rw	**Calibration Disable Control** 0 Calibration cycles are executed 1 Calibration is disabled (off) *Note: This control bit is active in both compatibility and enhanced mode.*
ADCH	[3:0]	rw	**Analog Input Channel Selection** Selects the (first) ADC channel which is to be converted

ADC_CTR2
ADC Control Register 2 **ESFR ($F09C_H/4E_H$)** **Reset Value: 0000_H**



ADC_CTR2IN
Injection Control Register 2 **ESFR ($F09E_H/4F_H$)** **Reset Value: 0000_H**



Field	Bits	Type	Description
RES	[13:12]	rw	**Converter Resolution Control** 00 10-bit resolution (default after reset) 01 8-bit resolution 1x Reserved
ADCTC	[11:6]	rw	**ADC Conversion Time Control** Defines the ADC basic conversion clock: $f_{BC} = f_{ADC}$ / (<ADCTC> + 1)
ADSTC	[5:0]	rw	**ADC Sample Time Control** Defines the ADC sample time: $t_S = t_{BC} \times 4 \times$ (<ADSTC> + 1)

Note: The limit values for f_{BC} (see data sheet) must not be exceeded when selecting ADCTC and f_{ADC}.

16.2 ADC Operation

Channel Selection, ADCH

Bitfield ADCH controls the input channel multiplexer logic. In the Single Channel Modes, it specifies the analog input channel which is to be converted. In the Auto Scan Modes, it specifies the highest channel number to be converted in the auto scan round.

ADCH may be changed while a conversion is in progress. The new value will go into effect after the current conversion is finished in the fixed channel modes, or after the current conversion round is finished in the auto scan modes.

ADC Flags, ADBSY, SAMPLE

The ADC Busy Status Flag is set when the ADC is started (by setting ADST) and remains set as long as the ADC performs conversions or calibration cycles.

ADBSY is cleared when the ADC is idle, meaning there are no conversion or calibration operations in progress.

Bit SAMPLE is set during the sample phase.

ADC Start/Stop Control, ADST

Bit ADST is used to start or to stop the ADC. A single conversion or a conversion sequence is started by setting bit ADST.

The busy flag ADBSY will be set and the converter then selects and samples the input channel, which is specified by the channel selection field ADCH. The sampled level will then be held internally during the conversion. When the conversion of this channel is complete, the result together with the number of the converted channel is transferred into the result register and the interrupt request is generated. The conversion result is placed into bitfield ADRES.

ADST remains set until cleared either by hardware or by software. Hardware clears the bit dependent on the conversion mode:

- In Fixed Channel Single Conversion mode, ADST is cleared after the conversion of the specified channel is finished.
- In Auto Scan Single Conversion mode, ADST is cleared after the conversion of channel 0 is finished.

Note: In the continuous conversion modes, ADST is never cleared by hardware.

Stopping the ADC via software is performed by clearing bit ADST. The reaction of the ADC depends on the conversion mode:

- In Fixed Channel Single Conversion mode, the ADC finishes the conversion and then stops. There is no difference to the operation if ADST was not cleared by software.
- In Fixed Channel Continuous Conversion mode, the ADC finishes the current conversion and then stops. This is the usual way to terminate this conversion mode.

- In Auto Scan Single Conversion mode, the ADC continues the auto scan round until the conversion of channel 0 is finished, then it stops. There is no difference to the operation if ADST was not cleared by software.
- In Auto Scan Continuous Conversion mode, the ADC continues the auto scan round until the conversion of channel 0 is finished, then it stops. This is the usual way to terminate this conversion mode.

A restart of the ADC can be performed by clearing and then setting bit ADST. This sequence will abort the current conversion and restart the ADC with the new parameters given in the control registers.

Conversion Mode Selection, ADM

Bitfield ADM selects the conversion mode of the A/D converter, as listed in **Table 16-1**.

Table 16-1 A/D Converter Conversion Mode

ADM	Description
00	Fixed Channel Single Conversion
01	Fixed Channel Continuous Conversion
10	Auto Scan Single Conversion
11	Auto Scan Continuous Conversion

While a conversion is in progress, the mode selection field ADM and the channel selection field ADCH may be changed. ADM will be evaluated after the current conversion. ADCH will be evaluated after the current conversion (fixed channel modes) or after the current conversion sequence (auto scan modes).

Conversion Resolution Control, RES

The ADC can produce either a 10-bit result (RES = 0) or an 8-bit result (RES = 1). Depending on the application's needs a higher conversion speed (an 8-bit conversion requires less conversion time) or a higher resolution can be chosen.

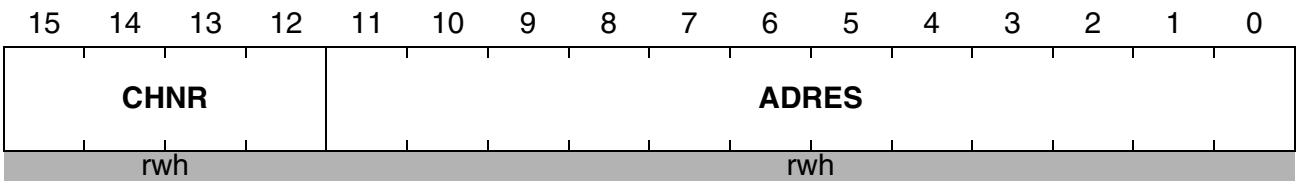
Conversion Result

The result of a conversion is stored in the result register ADC_DAT, or in register ADC_DAT2 for an injected conversion.

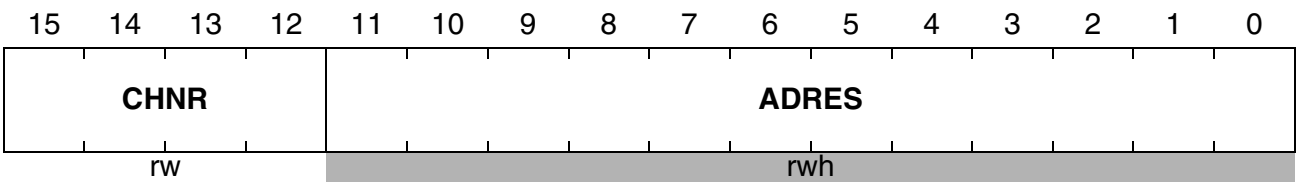
The position of the result depends on the basic operating mode (compatibility or enhanced) and on the selected resolution (8-bit or 10-bit).

Note: Bitfield CHNR of register ADC_DAT is loaded by the ADC to indicate, which channel the result refers to. Bitfield CHNR of register ADC_DAT2 is loaded by the CPU to select the analog channel, which is to be injected.

ADC_DAT
ADC Result Register **SFR ($FEA0_H/50_H$)** **Reset Value: 0000_H**



ADC_DAT2
ADC Chan. Inj. Result Reg. **ESFR ($F0A0_H/50_H$)** **Reset Value: 0000_H**



Field	Bits	Type	Function
CHNR	[15:12]	rw[h]	**Channel Number** (identifies the converted analog channel)
ADRES	[11:0]	rwh	**A/D Conversion Result** The digital result of the most recent conversion. In compatibility mode, the result is placed as follows: 8-bit: ADRES[9:2] 10-bit: ADRES[9:0] In enhanced mode, the result is placed as follows: 8-bit: ADRES[11:4] 10-bit: ADRES[11:2] *Note: Unused bits of ADRES are always set to 0.*

16.2.1 Fixed Channel Conversion Modes

These modes are selected by programming the mode selection bitfield ADM to 00_B (single conversion) or to 01_B (continuous conversion). After starting the converter through bit ADST the busy flag ADBSY will be set and the channel specified in bitfield ADCH will be converted. After the conversion is complete, the interrupt request flag ADCIR will be set.

In Single Conversion Mode the converter will automatically stop and reset bits ADBSY and ADST.

In Continuous Conversion Mode the converter will automatically start a new conversion of the channel specified in ADCH. ADCIR will be set after each completed conversion.

When bit ADST is reset by software, while a conversion is in progress, the converter will complete the current conversion and then stop and reset bit ADBSY.

16.2.2 Auto Scan Conversion Modes

These modes are selected by programming the mode selection field ADM to 10_B (single conversion) or to 11_B (continuous conversion). Auto Scan modes automatically convert a sequence of analog channels, beginning with the channel specified in bitfield ADCH and ending with channel 0, without requiring software to change the channel number. After starting the converter through bit ADST, the busy flag ADBSY will be set and the channel specified in bitfield ADCH will be converted. After the conversion is complete, the interrupt request flag ADCIR will be set and the converter will automatically start a new conversion of the next lower channel. ADCIR will be set after each completed conversion. After conversion of channel 0 the current sequence is complete.

In Single Conversion Mode the converter will automatically stop and reset bits ADBSY and ADST.

In Continuous Conversion Mode the converter will automatically start a new sequence beginning with the conversion of the channel specified in ADCH.

When bit ADST is reset by software, while a conversion is in progress, the converter will complete the current sequence (including conversion of channel 0) and then stop and reset bit ADBSY.

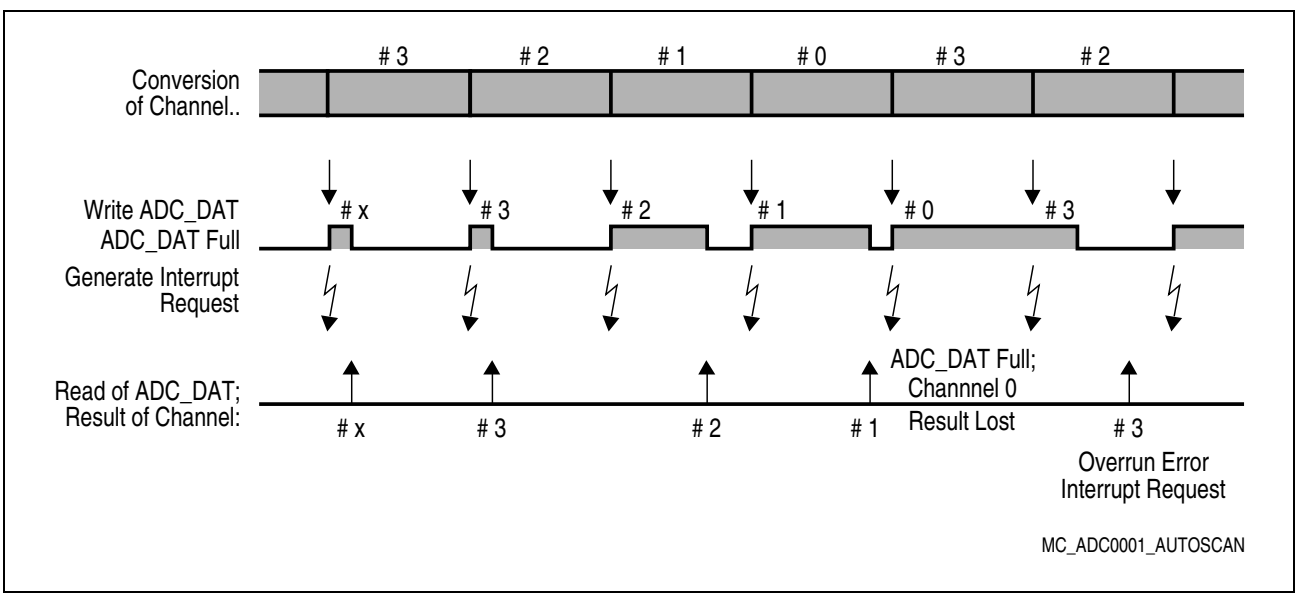


Figure 16-3 Auto Scan Conversion Mode Example

Note: Auto Scan sequences that begin with channel numbers above 7 will generate (up to) 4 invalid results from channels 11 … 8 which are not connected to input pins. Starting an Auto Scan sequence with ADCH = E_H will generate the following 15 results: 14, 13, 12, x, x, x, x, 7, 6, 5, 4, 3, 2, 1, 0.
Starting a sequence with ADCH = B_H … 8_H generates 4 … 1 invalid results at the beginning of the sequence and therefore makes no sense in an application.

16.2.3 Wait for Read Mode

If in default mode of the ADC a previous conversion result has not been read out of the result register by the time a new conversion is complete, the previous result is lost because it is overwritten by the new value, and the A/D overrun error interrupt request flag ADEIR will be set.

In order to avoid error interrupts and the loss of conversion results especially when using continuous conversion modes, the ADC can be switched to “Wait for Read Mode” by setting bit ADWR.

If the result value has not been read by the time the current conversion is complete, the new result is stored in a temporary buffer and the next conversion is suspended (ADST and ADBSY will remain set in the meantime, but no end-of-conversion interrupt will be generated). After reading the previous value the temporary buffer is copied into ADC_DAT (generating an ADCIR interrupt) and the suspended conversion is started. This mechanism applies to both single and continuous conversion modes.

Note: While in standard mode continuous conversions are executed at a fixed rate (determined by the conversion time), in “Wait for Read Mode” there may be delays due to suspended conversions. However, this only affects the conversions, if the CPU (or PEC) cannot keep track with the conversion rate.

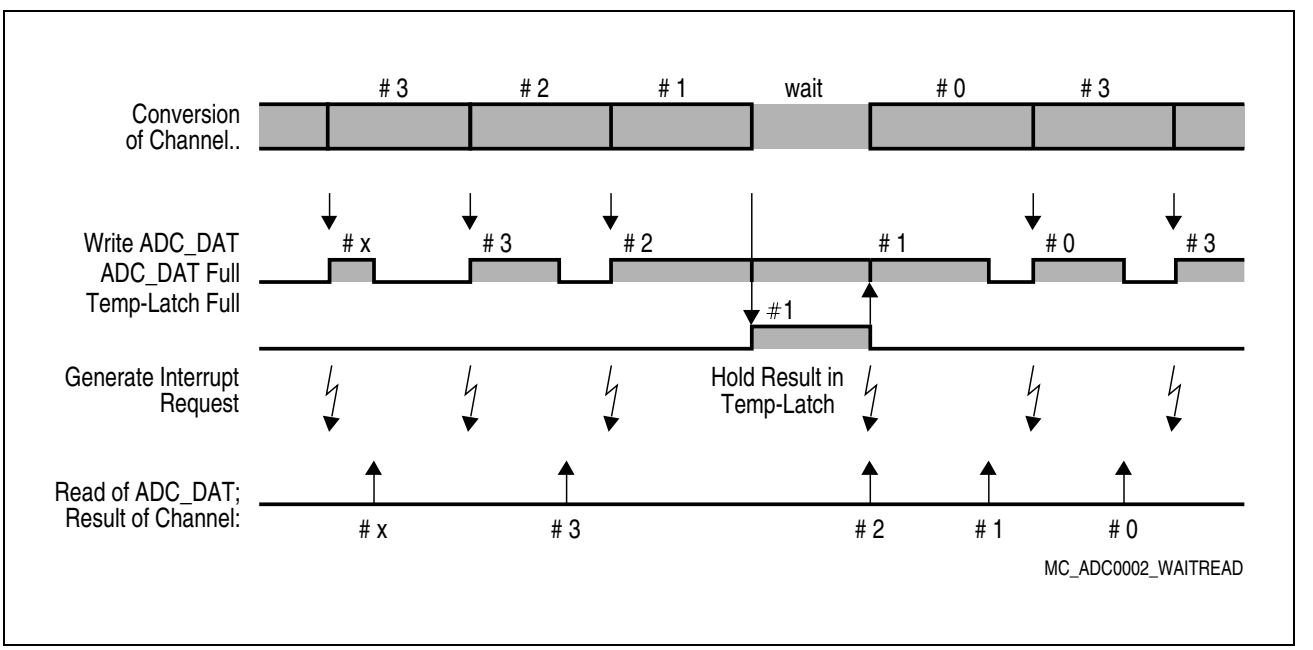


Figure 16-4 Wait for Read Mode Example

16.2.4 Channel Injection Mode

Channel Injection Mode allows the conversion of a specific analog channel (also while the ADC is running in a continuous or auto scan mode) without changing the current operating mode. After the conversion of this specific channel the ADC continues with the original operating mode.

Channel Injection mode is enabled by setting bit ADCIN and requires the Wait for Read Mode (ADWR = 1). The channel to be converted in this mode is specified in bitfield CHNR of register ADC_DAT2.

Note: Bitfield CHNR in ADC_DAT2 is not modified by the A/D converter, but only the ADRES bitfield. Since the channel number for an injected conversion is not buffered, bitfield CHNR of ADC_DAT2 must never be modified during the sample phase of an injected conversion, otherwise the input multiplexer will switch to the new channel. It is recommended to only change the channel number with no injected conversion running.

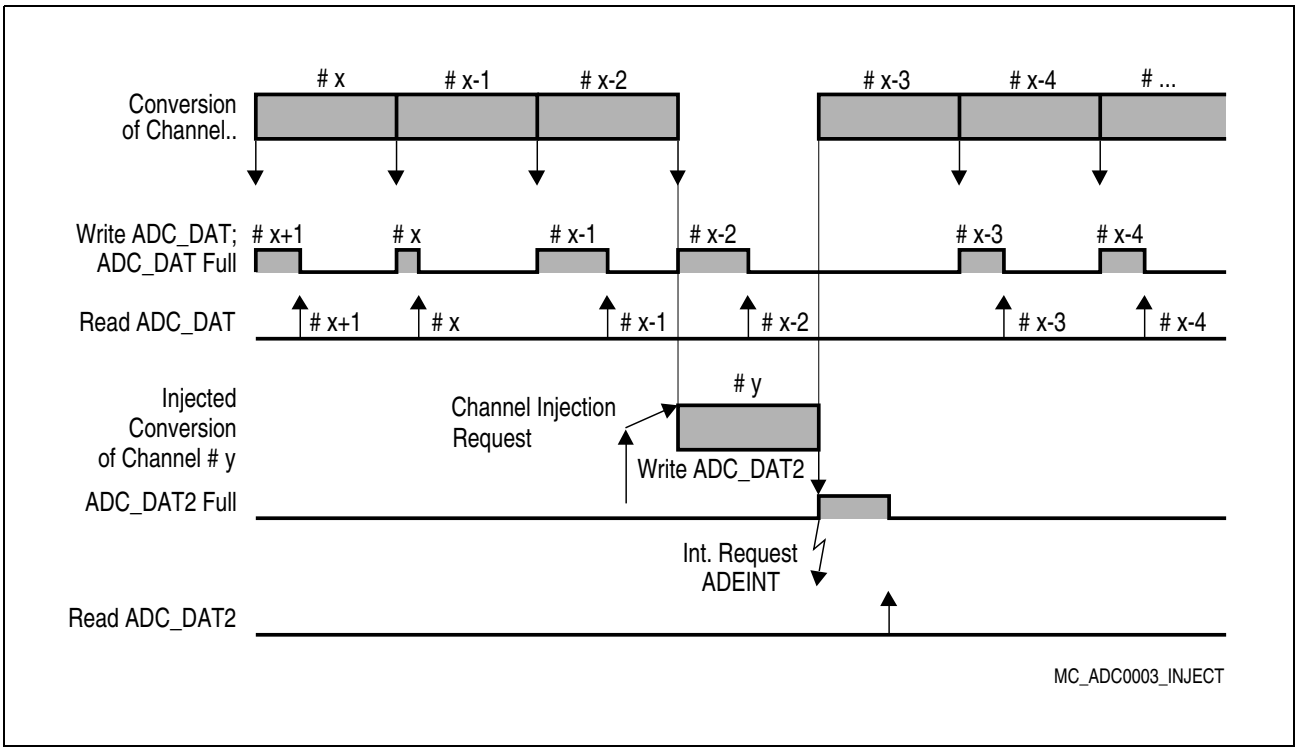


Figure 16-5 Channel Injection Example

A channel injection can be triggered in two ways:

- setting of the Channel Injection Request bit ADCRQ via software
- a compare or a capture event of Capture/Compare register CC31 of the CAPCOM2 unit, which also sets bit ADCRQ.

The second method triggers a channel injection at a specific time, on the occurrence of a predefined count value of the CAPCOM timers or on a capture event of register CC31. This can be either the positive, the negative, or both the positive and the negative edge of an external signal. In addition, this option allows recording the time of occurrence of this signal.

Note: The channel injection request bit ADCRQ will be set on any interrupt request of CAPCOM2 channel CC31, regardless whether the channel injection mode is enabled or not. It is recommended to always clear bit ADCRQ before enabling the channel injection mode.

After the completion of the current conversion (if any is in progress) the converter will start (inject) the conversion of the specified channel. When the conversion of this channel is complete, the result will be placed into the alternate result register ADC_DAT2, and a Channel Injection Complete Interrupt request will be generated, which uses the interrupt request flag ADEIR (for this reason the Wait for Read Mode is required).

Note: The result of an injected conversion is directly written to ADC_DAT2. If the previous result has not been read in the meantime, it is overwritten.
Standard conversions are suspended if the temporary buffer is full.

Arbitration of Conversions

Conversion requests that are activated while the ADC is idle immediately trigger the respective conversion. If a conversion is requested while another conversion is currently in progress the operation of the A/D converter depends on the kind of the involved conversions (standard or injected).

Note: A conversion request is activated if the respective control bit (ADST or ADCRQ) is toggled from 0 to 1, i.e. the bit must have been zero before being set.

Table 16-2 summarizes the ADC operation in the possible situations.

Table 16-2 Conversion Arbitration

Conversion in Progress	New Requested Conversion	
	Standard	**Injected**
Standard	Abort running conversion, and start requested new conversion.[1)]	Complete running conversion, start requested conversion after that.
Injected	Complete running conversion, start requested conversion after that.	Complete running conversion, start requested conversion after that. Bit ADCRQ will be 0 for the second conversion, however.

1) If an injected conversion is pending when a standard conversion is re-started, the injected conversion is executed before the newly started standard conversion.

16.3 Automatic Calibration

The ADC of the XC161 features automatic self-calibration. This calibration corrects gain errors, which are mainly due to process variation, and offset errors, which are mainly due to temperature changes.

Two types of calibration are supported:

- Reset calibration performs a thorough basic calibration of the ADC after a reset. In particular this is required after a power-on reset.
- Post-calibration performs one small calibration step after each conversion.

Reset Calibration

After a reset, a thorough power-up calibration is performed automatically to correct gain and offset errors of the A/D converter. To achieve best calibration results, the reference voltages as well as the supply voltages must be stable during the power-up calibration. During the calibration sequence a series of calibration cycles is executed, where the step width for adjustments is reduced gradually. The total number of executed calibration cycles depends on the actual properties of the respective ADC module. The maximum duration of the power-up calibration is 11,696 cycles of the basic clock f_{BC}.

Status flag CAL is set as long as this power-up calibration takes place.

Post-Calibration

After each conversion a small calibration step can be executed. For 8-bit and 10-bit conversions post-calibration is not mandatory in order not to exceed the total unadjusted error (TUE) specified in the data sheet. Post-calibration can be disabled by setting bit CALOFF in register ADC_CTR0. When disabled, the post-calibration cycles are skipped which reduces the total conversion time.

Note: Calibration may be disabled only after the reset calibration is complete.

16.4 Conversion Timing Control

When a conversion is started, first the capacitances of the converter are loaded via the respective analog input pin to the current analog input voltage. The time to load the capacitances is referred to as sample time. Next the sampled voltage is converted to a digital value in successive steps, which correspond to the resolution of the ADC. During these phases (except for the sample time) the internal capacitances are repeatedly charged and discharged via pins V_{AREF} and V_{AGND}.

The current that has to be drawn from the sources for sampling and changing charges depends on the time that each respective step takes, because the capacitors must reach their final voltage level within the given time, at least with a certain approximation. The maximum current, however, that a source can deliver, depends on its internal resistance.

The time that the two different actions during conversion take (sampling, and converting) can be programmed within a certain range in the XC161 relative to the CPU clock. The absolute time that is consumed by the different conversion steps therefore is independent from the general speed of the controller. This allows adjusting the A/D converter of the XC161 to the properties of the system:

Fast Conversion can be achieved by programming the respective times to their absolute possible minimum. This is preferable for scanning high frequency signals. The internal resistance of analog source and analog supply must be sufficiently low, however.

High Internal Resistance can be achieved by programming the respective times to a higher value, or the possible maximum. This is preferable when using analog sources and supply with a high internal resistance in order to keep the current as low as possible. The conversion rate in this case may be considerably lower, however.

Control Bitfields

For the timing control of the conversion and the sample phase two mechanisms are provided:

- **Standard timing control** uses two 2-bit fields in register ADC_CON to select prescaler values for the general conversion timing and the duration of the sample phase. This provides compact control, while limiting the prescaler factors to a few steps.
- **Improved timing control** uses two 6-bit fields in register ADC_CON1 (compatibility mode) or register ADC_CTR2/ADC_CTR2IN (enhanced mode). This provides a wide range of prescaler factors, so the ADC can be better adjusted to the internal and external system circumstances.

Improved timing control is selected by setting bit ICST in register ADC_CON1 in compatibility mode, or by selecting enhanced mode.

Standard Timing Control

Standard timing control is performed by using two 2-bit fields in register ADC_CON. Bitfield ADCTC (conversion time control) selects the basic conversion clock (f_{BC}), used for the operation of the A/D converter. The sample time is derived from this conversion clock and controlled by bitfield ADSTC. The sample time is always a multiple of 8 f_{BC} periods. **Table 16-3** lists the possible combinations.

Table 16-3 Standard Conversion and Sample Timing Control

ADC_CON.15|14 (ADCTC)	A/D Converter Basic Clock f_{BC} [1)]	ADC_CON.13|12 (ADSTC)	Sample Time t_S
00	$f_{ADC}/4$	00	$t_{BC} \times 8$
01	$f_{ADC}/2$	01	$t_{BC} \times 16$
10	$f_{ADC}/16$	10	$t_{BC} \times 32$
11	$f_{ADC}/8$	11	$t_{BC} \times 64$

Improved Timing Control

To provide a finer resolution for programming of the timing parameters, wider bitfields have been implemented for timing control (the 2-bit bitfields in register ADC_CON are disregarded in all cases).

In compatibility mode (with bit ICST = 1), the bitfields in register ADC_CON1 are used for all conversions.
In enhanced mode (bit MD = 1), the bitfields in register ADC_CTR2 are used for standard conversions. Injected conversions use the bitfields in register ADC_CTR2IN.

Bitfield ADCTC (conversion time control) selects the basic conversion clock (f_{BC}), used for the operation of the A/D converter. The sample time is derived from this conversion clock and controlled by bitfield ADSTC. The sample time is always a multiple of 4 f_{BC} periods. **Table 16-4** lists the possible combinations.

Table 16-4 Improved Conversion and Sample Timing Control

ADCTC	A/D Converter Basic Clock f_{BC} [1)]	ADSTC	Sample Time t_S
$00'0000_B = 00_H$	$f_{ADC}/1$	$00'0000_B = 00_H$	$t_{BC} \times 8$
$00'0001_B = 01_H$	$f_{ADC}/2$	$00'0001_B = 01_H$	$t_{BC} \times 12$
$00'0010_B = 02_H$	$f_{ADC}/3$	$00'0010_B = 02_H$	$t_{BC} \times 16$
…	$f_{ADC}/(ADCTC + 1)$	…	$t_{BC} \times 4 \times (ADSTC + 2)$
$11'1111_B = 3F_H$	$f_{ADC}/64$	$11'1111_B = 3F_H$	$t_{BC} \times 260$

1) The limit values for f_{BC} (see data sheet) must not be exceeded when selecting ADCTC and f_{ADC}.

Total Conversion Time Examples

The time for a complete conversion includes the sample time t_S, the conversion itself (successive approximation and calibration), and the time required to transfer the digital value to the result register as shown in the example below (standard conversion timing).

The timings refer to module clock cycles, where $t_{ADC} = 1/f_{ADC}$.

- Assumptions: f_{ADC} = 40 MHz (i.e. t_{ADC} = 25 ns), ADCTC = 01_B, ADSTC = 00_B
- Basic clock: $f_{BC} = f_{ADC}/2$ = 20 MHz, i.e. t_{BC} = 50 ns
- Sample time: $t_S = t_{BC} \times 8$ = 400 ns

Conversion 10-bit:

- With post-calibr.: $t_{C10P} = t_S + 52 \times t_{BC} + 6 \times t_{ADC}$ = (2600 + 400 + 150) ns = 3.15 µs
- Post-calibr. off: $t_{C10} = t_S + 40 \times t_{BC} + 6 \times t_{ADC}$ = (2000 + 400 + 150) ns = 2.55 µs

Conversion 8-bit:

- With post-calibr.: $t_{C8P} = t_S + 44 \times t_{BC} + 6 \times t_{ADC}$ = (2200 + 400 + 150) ns = 2.75 µs
- Post-calibr. off: $t_{C8} = t_S + 32 \times t_{BC} + 6 \times t_{ADC}$ = (1600 + 400 + 150) ns = 2.15 µs

Note: For the exact specification please refer to the data sheet of the selected derivative.

16.5 A/D Converter Interrupt Control

At the end of each conversion, interrupt request flag ADCIR in interrupt control register ADC_CIC is set. This end-of-conversion interrupt request may cause an interrupt to vector ADCINT, or it may trigger a PEC data transfer which reads the conversion result from register ADC_DAT, e.g. to store it into a table in internal RAM for later evaluation.

The interrupt request flag ADEIR in register ADC_EIC will be set either, if a conversion result overwrites a previous value in register ADC_DAT (error interrupt in standard mode), or if the result of an injected conversion has been stored into ADC_DAT2 (end-of-injected-conversion interrupt). This interrupt request may be used to cause an interrupt to vector ADEINT, or it may trigger a PEC data transfer.

ADC_CIC
ADC Conversion Intr. Ctrl. Reg. SFR ($FF98_H/CC_H$) Reset Value: - - 00_H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-							GPX	ADC IR	ADC IE	ILVL				GLVL	
-	-	-	-	-	-	-	rw	rwh	rw	rw				rw	

ADC_EIC
ADC Error Intr. Ctrl. Reg. SFR ($FF9A_H/CD_H$) Reset Value: - - 00_H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-							GPX	ADE IR	ADE IE	ILVL				GLVL	
-	-	-	-	-	-	-	rw	rwh	rw	rw				rw	

Note: Please refer to the general Interrupt Control Register description for an explanation of the control fields.

16.6 Interfaces of the ADC Module

The ADC is connected to its environment in different ways.

Internal Connections

The capture/compare signal CC31IO of the CAPCOM2 unit is connected to the ADC, providing an optional trigger source for injected conversions.

The 2 interrupt request lines of the ADC are connected to the interrupt control block.

External Connections

The analog input signals for the ADC are connected with Port 5 of the XC161 (input only). Two dedicated pins (V_{AREF} and V_{AGND}) provide the analog reference voltage for the conversion mechanism.

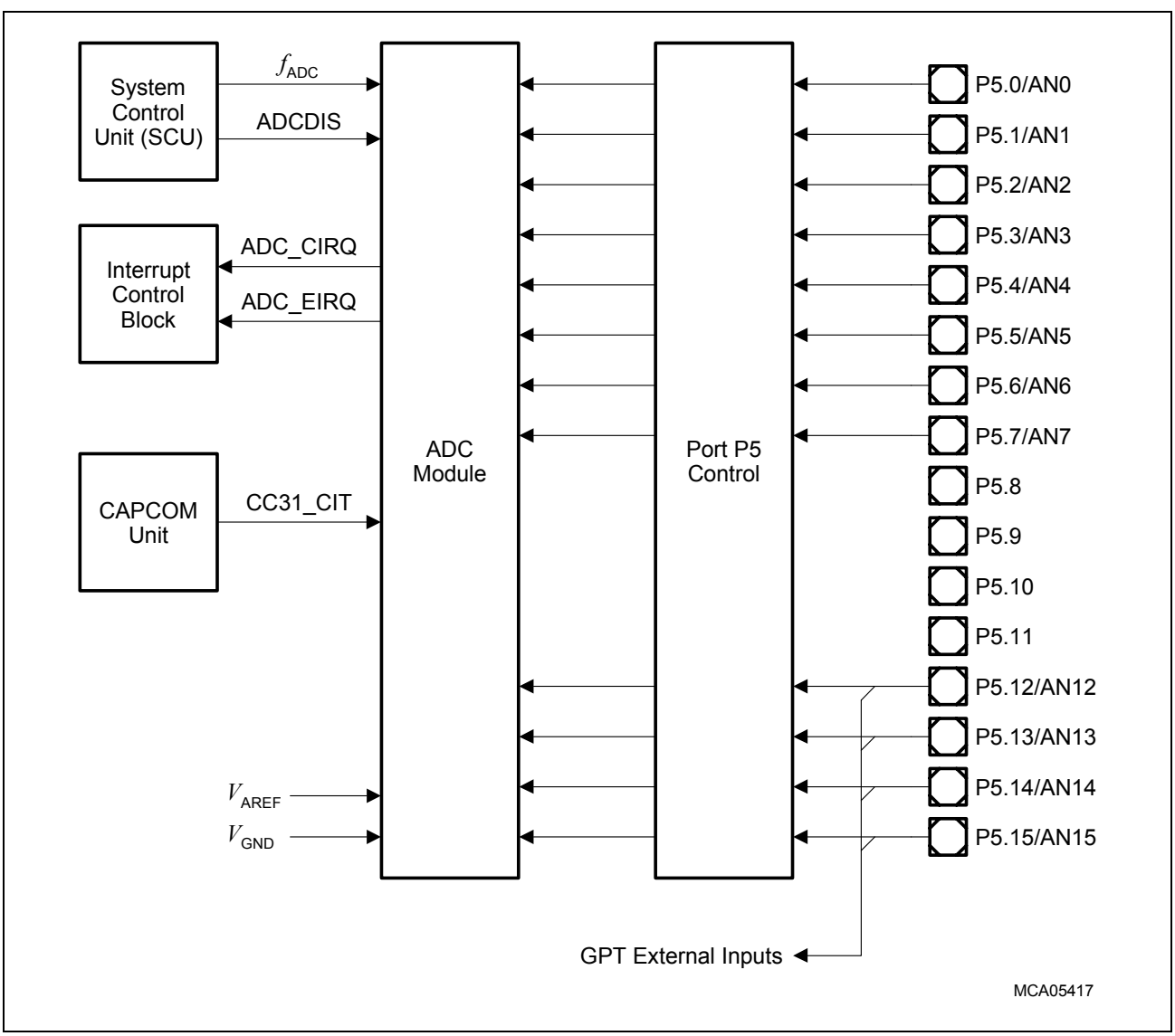


Figure 16-6 ADC Module IO Interface

17 Capture/Compare Units

The XC161 provides two, almost identical, Capture/Compare (CAPCOM) units, which only differ in the way they are connected to the pins. Each CAPCOM unit provides 16 capture/compare channels, which interact with 2 timers. A CAPCOM channel can **capture** the contents of a timer on specific internal or external events, or it can **compare** a timer's contents with given values, and modify output signals in case of a match.

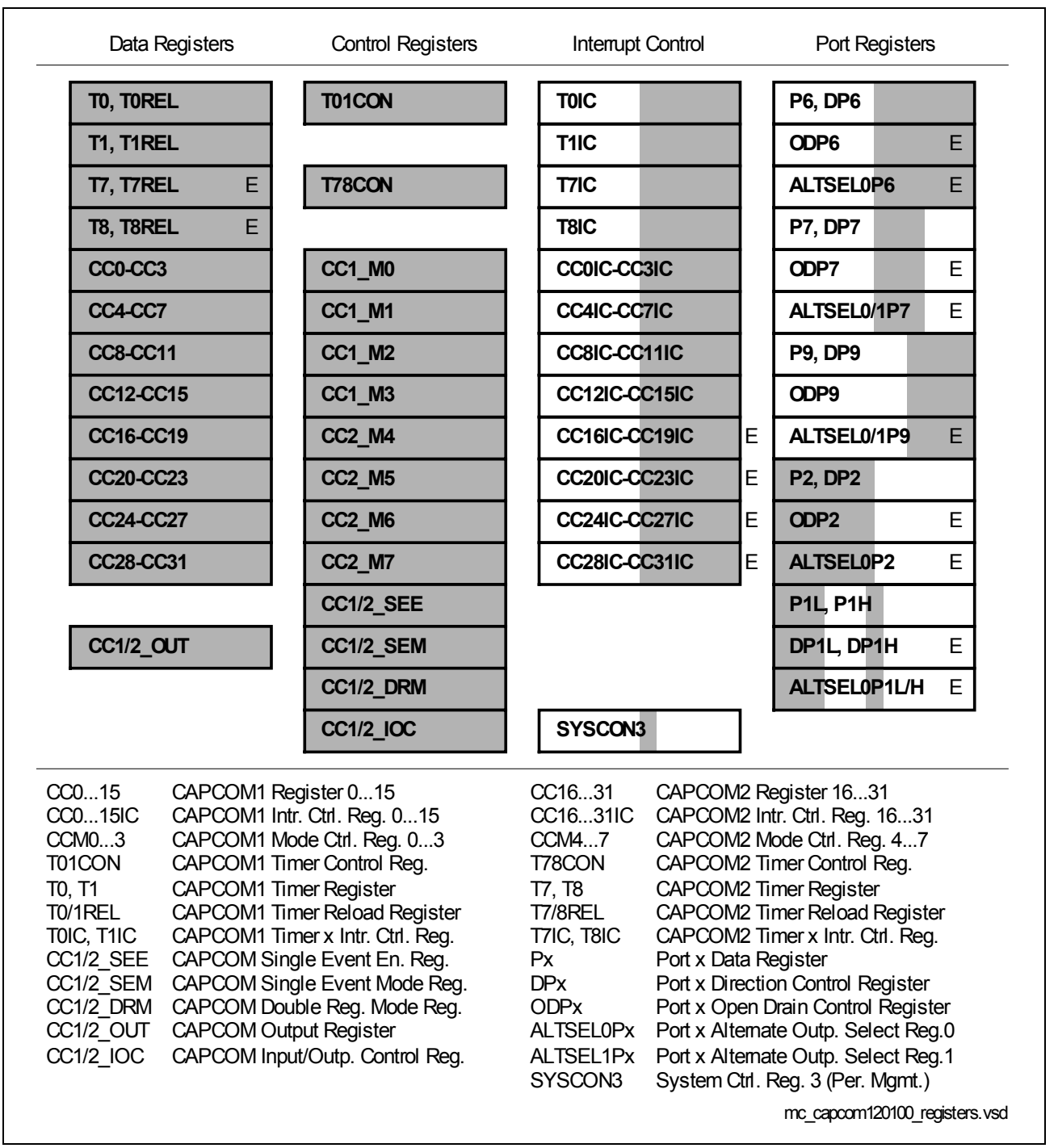


Figure 17-1 SFRs Associated with the CAPCOM Units

With this mechanism, each CAPCOM unit supports generation and control of timing sequences on up to 16 channels with a minimum of software intervention.

From the programmer's point of view, the term 'CAPCOM unit' refers to a set of registers which are associated with this peripheral, including the port pins which may be used for alternate input/output functions, and their direction control bits (see also **Figure 17-1**).

A CAPCOM unit is typically used to handle high speed IO tasks such as pulse and waveform generation, pulse width modulation, or recording of the time when a specific event occurs. It also supports the implementation of up to 16 software-controlled interrupt events.

Each CAPCOM Unit consists of two 16-bit timers (T0/T1, T7/T8), each with its own reload register (TxREL), and a bank of sixteen dual-purpose 16-bit capture/compare registers (CCy).

The input clock for the CAPCOM timers is programmable to several prescaled values of the module input clock (f_{CC}), or it can be derived from the overflow/underflow of timer T6. T0/T7 may also operate in counter mode (from an external input), clocked by external events.

Each capture/compare register may be programmed individually for capture or compare operation, and each register may be allocated to either of the two timers. Each capture/compare register has one signal associated with it, which serves as an input signal for the capture operation or as an output signal for the compare operation.

The capture operation causes the current timer contents to be latched into the respective capture/compare register, triggered by an event (transition) on the associated input signal. This event also activates the associated interrupt request line.

The compare operation may cause an output signal transition on the associated output signal, when the allocated timer increments to the value stored in a capture/compare register. The compare match event also activates the associated interrupt request line. In Double-register compare mode a pair of registers controls one common output signal.

The compare output signals are available via a dedicated output register, and may also control the output latches of the connected port pins. The output path can be selected.

For the switching of the output signals two timing schemes (see **Section 17.8**) can be selected:

In **Staggered Mode**[1] the output signals are switched consecutively in 8 steps, which distributes the switching steps over a certain time. In staggered mode, the maximum resolution is 8 t_{CC}.

In **Non-Staggered Mode** the output signals are switched immediately at the same time. In non-staggered mode, the maximum resolution is 1 t_{CC}.

Figure 17-2 shows the basic structure of a CAPCOM unit.

1) Staggered mode is compatible with the CAPCOM units of previous 16-bit controllers.

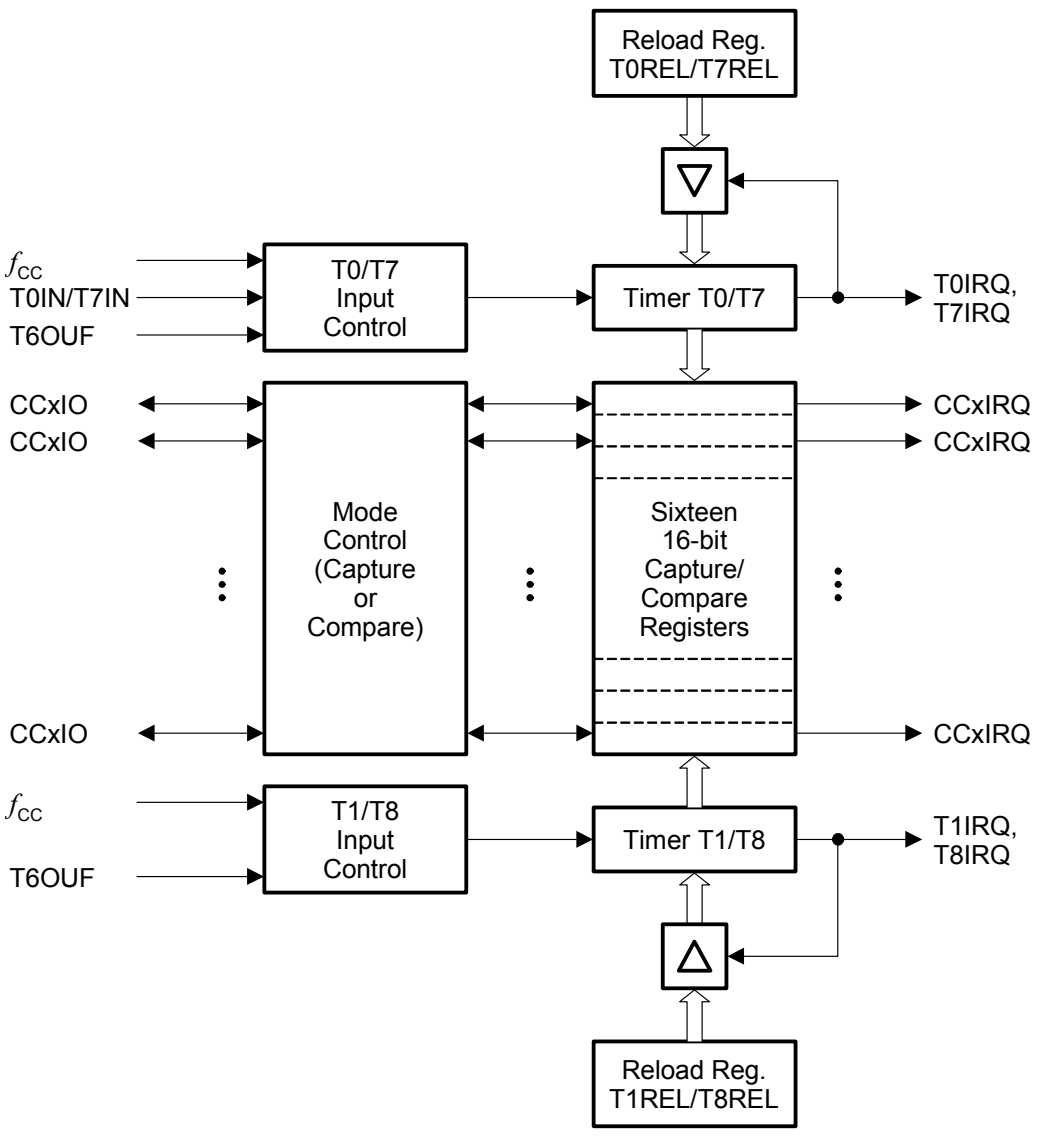


Figure 17-2 CAPCOM Unit Block Diagram

17.1 The CAPCOM Timers

The primary use of the timers T0/T7 and T1/T8 is to provide two independent time bases for the capture/compare channels of each unit. The maximum resolution is 8 t_{CC} in staggered mode, and 1 t_{CC} in non-staggered mode.

The basic structure of the two timers, illustrated in **Figure 17-3**, is identical, except for the input pin (see mark).

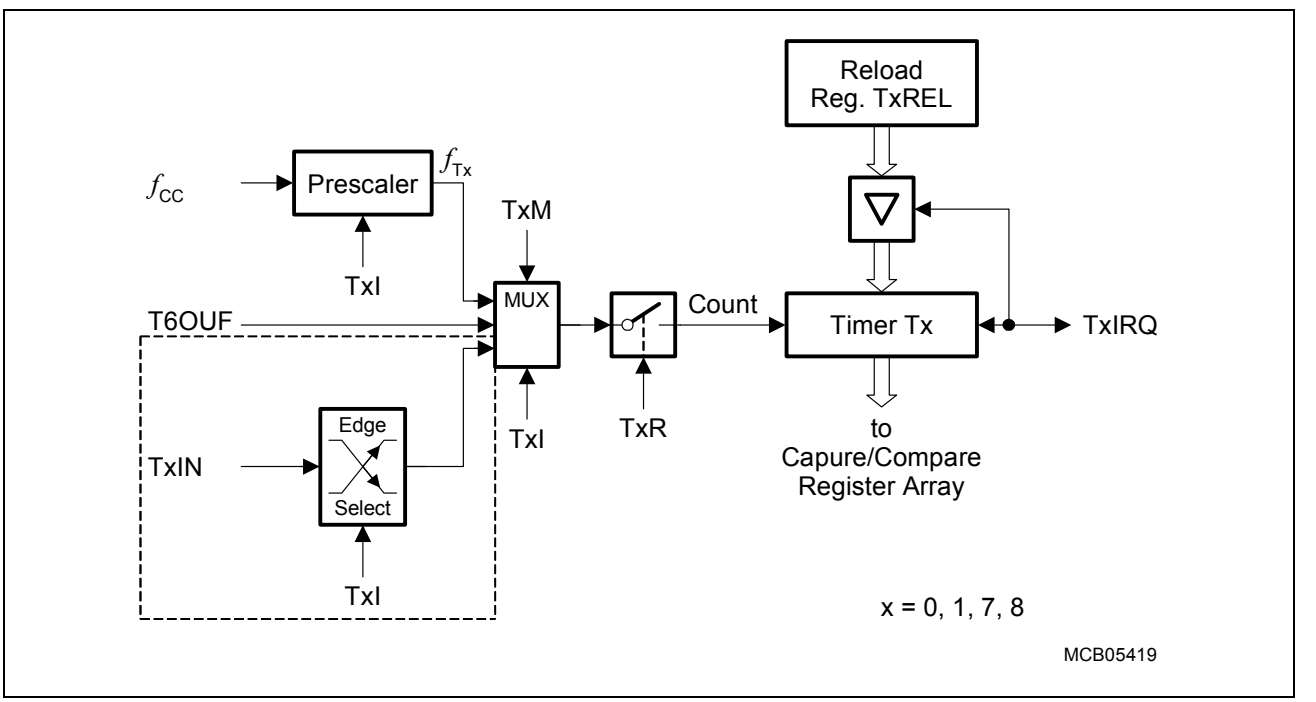


Figure 17-3 Block Diagram of a CAPCOM Timer

Note: When an external input signal is connected to the input lines of both T0 and T7, these timers count the input signal synchronously. Thus, the two timers can be regarded as one timer whose contents can be compared with 32 compare registers.

The functions of the CAPCOM timers are controlled via the bit-addressable control registers T01CON and T78CON. The high-byte of T01CON controls T1, the low-byte of T01CON controls T0. The high-byte of T78CON controls T8, the low-byte of T78CON controls T7. The control options are identical for all four timers (except for external input).

In all modes, the timers are always counting upward. The current timer values are accessible for the CPU in the timer registers Tx, which are non bit-addressable registers. When the CPU writes to a register Tx in the state immediately before the respective timer increment or reload is to be performed, the CPU write operation has priority and the increment or reload is disabled to guarantee correct timer operation.

CC1_T01CON
Timer 0/1 Control Register **SFR ($FF50_H/A8_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	T1R	-		T1M		T1I		-	T0R	-		T0M		T0I	
-	rw	-		rw		rw		-	rw	-		rw		rw	

CC2_T78CON
Timer 7/8 Control Register **SFR ($FF20_H/90_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	T8R	-		T8M		T8I		-	T7R	-		T7M		T7I	
-	rw	-		rw		rw		-	rw	-		rw		rw	

Field	Bits	Type	Description
TxR	14, 6	rw	**Timer/Counter Tx Run Control** 0 Timer/Counter Tx is disabled 1 Timer/Counter Tx is enabled
TxM	11, 3	rw	**Timer/Counter Tx Mode Selection** 0 Timer Mode 1 Counter Mode
TxI	[10:8], [2:0]	rw	**Timer/Counter Tx Input Selection** **Timer Mode** (TxM = 0): Input frequency $f_{Tx} = f_{CC}/2^{(<TxI>+3)}$ or $f_{CC}/2^{(<TxI>)}$, depending on (non-)staggered mode, see **Table 17-1** **Counter Mode** (TxM = 1): 000 Overflow/Underflow of GPT Timer T6 001 Positive (rising) edge on pin TxIN 010 Negative (falling) edge on pin TxIN 011 Any edge (rising and falling) on pin TxIN 1XX Reserved. Do not use this combination! *Note: For timers T1 and T8 the only option in counter mode is 000_B. T1 and T8 stop in other cases.*

The timer run flags TxR allow the starting and stopping of the timers. The following description of the timer modes and operation always applies to the enabled state of the timers, i.e. the respective run flag is assumed to be set.

Timer Mode

In Timer Mode (TxM = 0), the input clock for a CAPCOM timer is derived from f_{CC}, divided by a programmable prescaler. Each timer has its own individual prescaler, controlled through the individual bitfields TxI in the timer control registers T01CON and T78CON.

The input frequency f_{Tx} for a timer Tx and its resolution r_{Tx} are determined by the following formulas:

Staggered Mode:

$$f_{Tx}[MHz] = \frac{f_{CC}[MHz]}{2^{(<TxI> + 3)}} \qquad r_{Tx}[\mu s] = \frac{2^{(<TxI> + 3)}}{f_{CC}[MHz]} \tag{17.1}$$

Non-Staggered Mode:

$$f_{Tx}[MHz] = \frac{f_{CC}[MHz]}{2^{<TxI>}} \qquad r_{Tx}[\mu s] = \frac{2^{<TxI>}}{f_{CC}[MHz]} \tag{17.2}$$

When a timer overflows from $FFFF_H$ to 0000_H, it is reloaded with the value stored in its respective reload register TxREL. The reload value determines the period P_{Tx} between two consecutive overflows of Tx as follows:

Staggered Mode:

$$P_{Tx}[\mu s] = \frac{(2^{16} - <TxREL>) \times 2^{(<TxI> + 3)}}{f_{CC}[MHz]} \tag{17.3}$$

Non-Staggered Mode:

$$P_{Tx}[\mu s] = \frac{(2^{16} - <TxREL>) \times 2^{<TxI>}}{f_{CC}[MHz]} \tag{17.4}$$

After a timer has been started by setting its run flag (TxR), the first increment will occur within the time interval which is defined by the selected timer resolution. All further increments occur exactly after the time defined by the timer resolution.

Examples for timer input frequencies, resolution and periods, which result from the selected prescaler option in TxI when using a 40 MHz clock, are listed in **Table 17-1** below. The numbers for the timer periods are based on a reload value of 0000_H. Note that some numbers may be rounded.

Table 17-1 Timer Tx Input Clock Selection for Timer Mode, f_{CC} = 40 MHz

TxI	Prescaler	Input Frequency	Resolution	Period
Non-Staggered Mode				
000_B	8	5 MHz	200 ns	13.11 ms
001_B	16	2.5 MHz	400 ns	26.21 ms
010_B	32	1.25 MHz	800 ns	52.43 ms
011_B	64	625 kHz	1.6 µs	104.86 ms
100_B	128	312.5 kHz	3.2 µs	209.72 ms
101_B	256	156.25 kHz	6.4 µs	419.43 ms
110_B	512	78.125 kHz	12.8 µs	838.86 ms
111_B	1024	39.0625 kHz	25.6 µs	1677.72 ms
Non-Staggered Mode				
000_B	1	40 MHz	25 ns	1.6384 ms
001_B	2	20 MHz	50 ns	3.2768 ms
010_B	4	10 MHz	100 ns	6.5536 ms
011_B	8	5 MHz	200 ns	13.11 ms
100_B	16	2.5 MHz	400 ns	26.21 ms
101_B	32	1.25 MHz	800 ns	52.43 ms
110_B	64	625 kHz	1.6 µs	104.86 ms
111_B	128	312.5 kHz	3.2 µs	209.72 ms

Counter Mode

In Counter Mode (TxM = 1), the input clock of a CAPCOM timer is either derived from an associated external input pin, T0IN/T7IN, or from the over-/underflows of GPT timer T6.

Using an external signal connected to pin TxIN as a counting signal is only possible for timers T0 and T7. The only counter option for timers T1 and T8 is using the over-/underflows of the GPT timer T6 (selected by TxI = 000_B).

Bitfields T0I/T7I are used to select either a positive, a negative, or both a positive and a negative transition of the external signal at pin T0IN/T7IN to trigger an increment of timer T0/T7. Please note that certain criteria must be met for the external signal and the port pin programming for this mode in order to operate properly. These conditions are detailed in **Chapter 17.10**.

Timer Overflow and Reload

When a CAPCOM timer contains the value $FFFF_H$ at the time a new count trigger occurs, a timer interrupt request is generated, and the timer is loaded with the contents of its associated reload register TxREL. The timer then resumes incrementing with the next count trigger starting from the reloaded value.

The reload registers TxREL are not bitaddressable. After reset, they contain the value 0000_H.

17.2 CAPCOM Timer Interrupts

Upon a timer overflow the corresponding timer interrupt request flag TxIR for the respective timer will be set. This flag can be used to generate an interrupt or trigger a PEC service request, when enabled by the respective interrupt enable bit TxIE.

Each timer has its own bitaddressable interrupt control register and its own interrupt vector. The organization of the interrupt control registers TxIC is identical with the other interrupt control registers.

CC1_T0IC
CAPCOM T0 Intr. Ctrl. Reg. SFR ($FF9C_H/CE_H$) Reset Value: - - 00_H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-							GPX	T0IR	T0IE	ILVL				GLVL	
-	-	-	-	-	-	-	rw	rwh	rw	rw				rw	

CC1_T1IC
CAPCOM T1 Intr. Ctrl. Reg. SFR ($FF9E_H/CF_H$) Reset Value: - - 00_H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-							GPX	T1IR	T1IE	ILVL				GLVL	
-	-	-	-	-	-	-	rw	rwh	rw	rw				rw	

CC2_T7IC
CAPCOM T7 Intr. Ctrl. Reg. ESFR ($F17A_H/BE_H$) Reset Value: - - 00_H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-							GPX	T7IR	T7IE	ILVL				GLVL	
-	-	-	-	-	-	-	rw	rwh	rw	rw				rw	

CC2_T8IC
CAPCOM T8 Intr. Ctrl. Reg. ESFR ($F17C_H/BF_H$) Reset Value: - - 00_H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-							GPX	T8IR	T8IE	ILVL				GLVL	
-	-	-	-	-	-	-	rw	rwh	rw	rw				rw	

Note: Please refer to the general Interrupt Control Register description for an explanation of the control fields.

17.3 Capture/Compare Channels

The 16-bit capture/compare registers CC0 through CC15 (CC16 through CC31) are used as data registers for capture or compare operations with respect to timers T0/T7 and T1/T8. The capture/compare registers are not bit-addressable.

The functions of the 16 capture/compare registers of a unit are controlled by 4 bit-addressable 16-bit mode control registers, named CC1_M0 … CC1_M3 (CC2_M4 … CC2_M7), which are all organized identically (see description below). Each register contains the bits for mode selection and timer allocation for four capture/comp. registers.

Capture/Compare Registers for the CAPCOM1 Unit (CC15 … CC0)

CC1_M0
CAPCOM Mode Ctrl. Reg. 0 SFR ($FF52_H/A9_H$) Reset Value: 0000_H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ACC 3		MOD3		ACC 2		MOD2		ACC 1		MOD1		ACC 0		MOD0	
rw		rw		rw		rw		rw		rw		rw		rw	

CC1_M1
CAPCOM Mode Ctrl. Reg. 1 SFR ($FF54_H/AA_H$) Reset Value: 0000_H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ACC 7		MOD7		ACC 6		MOD6		ACC 5		MOD5		ACC 4		MOD4	
rw		rw		rw		rw		rw		rw		rw		rw	

CC1_M2
CAPCOM Mode Ctrl. Reg. 2 SFR ($FF56_H/AB_H$) Reset Value: 0000_H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ACC 11		MOD11		ACC 10		MOD10		ACC 9		MOD9		ACC 8		MOD8	
rw		rw		rw		rw		rw		rw		rw		rw	

CC1_M3
CAPCOM Mode Ctrl. Reg. 3 SFR ($FF58_H/AC_H$) Reset Value: 0000_H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ACC 15		MOD15		ACC 14		MOD14		ACC 13		MOD13		ACC 12		MOD12	
rw		rw		rw		rw		rw		rw		rw		rw	

Capture/Compare Registers for the CAPCOM2 Unit (CC31 … CC16)

CC2_M4
CAPCOM Mode Ctrl. Reg. 4 **SFR ($FF22_H/91_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ACC19	MOD19			ACC18	MOD18			ACC17	MOD17			ACC16	MOD16		
rw	rw			rw	rw			rw	rw			rw	rw		

CC2_M5
CAPCOM Mode Ctrl. Reg. 5 **SFR ($FF24_H/92_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ACC23	MOD23			ACC22	MOD22			ACC21	MOD21			ACC20	MOD20		
rw	rw			rw	rw			rw	rw			rw	rw		

CC2_M6
CAPCOM Mode Ctrl. Reg. 6 **SFR ($FF26_H/93_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ACC27	MOD27			ACC26	MOD26			ACC25	MOD25			ACC24	MOD24		
rw	rw			rw	rw			rw	rw			rw	rw		

CC2_M7
CAPCOM Mode Ctrl. Reg. 7 **SFR ($FF28_H/94_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ACC31	MOD31			ACC30	MOD30			ACC29	MOD29			ACC28	MOD28		
rw	rw			rw	rw			rw	rw			rw	rw		

Field	Bits	Type	Description
ACCy	15, 11, 7, 3	rw	**Allocation Bit for CAPCOM Register CCy** 0 CCy allocated to Timer T0 or T7, respectively 1 CCy allocated to Timer T1 or T8, respectively
MODy	[14:12], [10:8], [6:4], [2:0]	rw	**Mode Selection for CAPCOM Register CCy** See **Table 17-2**.

Each of the registers CCy may be individually programmed for capture mode or for one of 4 different compare modes, and may be allocated individually to one of the two timers of the respective CAPCOM unit. A special double-register compare mode combines two registers to act on one common output signal. When capture or compare operations are disabled for one of the CCy registers, it may be used for general purpose variable storage.

Table 17-2 Selection of Capture Modes and Compare Modes

Mode	MODy	Selected Operating Mode
Disabled	**0 0 0**	**Disable Capture and Compare Modes** The respective CAPCOM register may be used for general variable storage.
Capture	**0 0 1**	**Capture on Positive Transition (Rising Edge)** at Pin CCyIO
	0 1 0	**Capture on Negative Transition (Falling Edge)** at Pin CCyIO
	0 1 1	**Capture on Positive and Negative Transition (Both Edges)** at Pin CCyIO
Compare	**1 0 0**	**Compare Mode 0:** Interrupt Only Several interrupts per timer period. Can enable double-register compare mode for Bank2 registers.
	1 0 1	**Compare Mode 1:** Toggle Output Pin on each Match Several compare events per timer period. Can enable double-register compare mode for Bank1 registers.
	1 1 0	**Compare Mode 2:** Interrupt Only Only one interrupt per timer period.
	1 1 1	**Compare Mode 3:** Set Output Pin on each Match Reset output pin on each timer overflow; only one interrupt per timer period.

The detailed discussion of the capture and compare modes is valid for all the capture/compare channels, so registers, bits and pins are only referenced by a placeholder.

Note: A capture or compare event on channel 31 may be used to trigger a channel injection on the XC161's A/D converter if enabled.

17.4 Capture Mode Operation

In Capture Mode, the current contents of a CAPCOM timer are latched (captured) into the respective capture/compare register in response to an external event. This is used, for example, to record the time at which an external event has occurred, or to measure the distance between two external events in timer increments.

The event to cause a capture of a timer's contents can be programmed to be either the positive, the negative, or both the positive and the negative transition of the external signal connected to the input pin. This triggering transition is selected by bitfield MODy in the respective mode control register. When the selected external signal transition occurs, the selected timer's contents is latched into the capture/compare register and the respective interrupt request line CCyIRQ is activated. This can cause an interrupt or PEC service request, when enabled.

Note: A capture input can be used as an additional external interrupt input. The capture operation can be disregarded in this case.

Either the contents of timer T0/T7 or T1/T8 can be captured, selected by the timer allocation control bit ACCy in the respective mode control register.

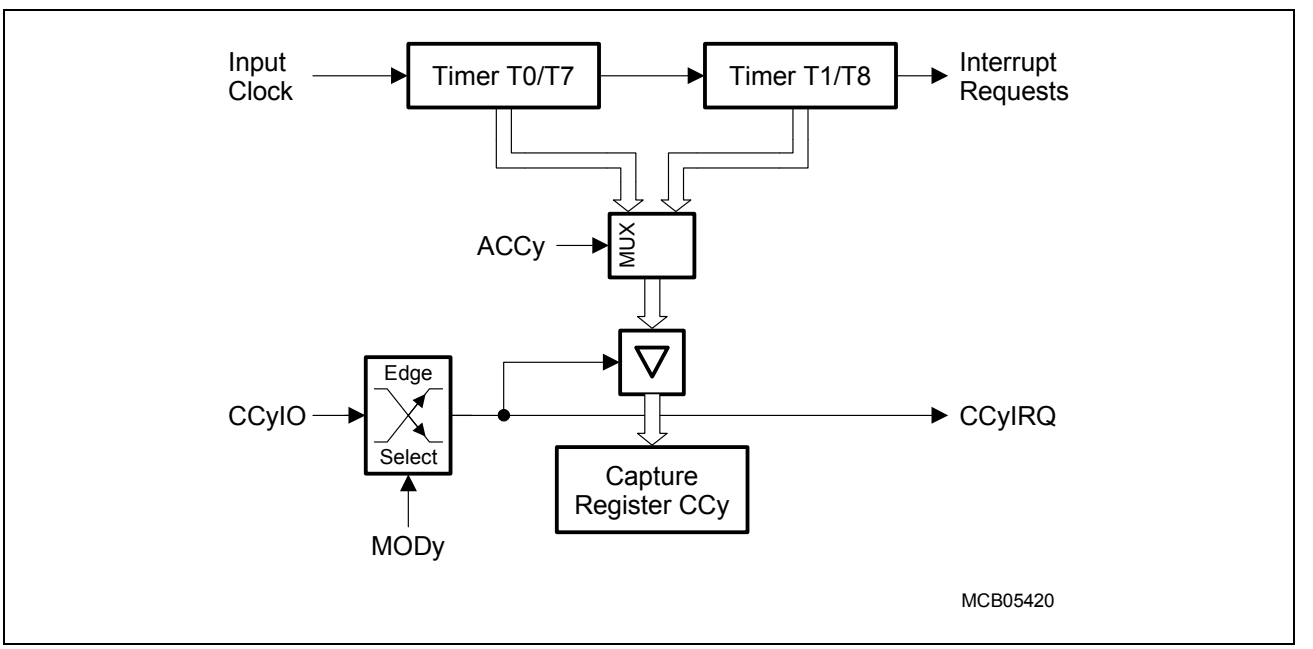


Figure 17-4 Capture Mode Block Diagram

For capture operation, the respective pin must be programmed for input. To ensure that a transition of the input signal is recognized correctly, its level must be held high or low for a minimum number of module clock cycles before it changes. This information can be found in **Section 17.10**.

17.5 Compare Mode Operation

The compare modes allow triggering of events (interrupts and/or output signal transitions) or generation of pulse trains with minimum software overhead. In all compare modes, the 16-bit value stored in a capture/compare register CCy (in the following also referred to as ‘compare value’) is continuously compared with the contents of the allocated timer (T0/T7 or T1/T8). If the current timer contents match the compare value, the interrupt request line associated with register CCy is activated and, depending on the compare mode, an output signal can be generated at the corresponding output pin CCyIO.

Four different compare modes are available, which can be selected individually for each of the capture/compare registers by bitfield MODy in the respective mode control register. Modes 0 and 2 do not influence the output signals. In the following, each mode is described in detail.

In addition to these ‘single-register’ modes, a ‘double-register’ compare mode enables two registers to operate on the same pin. This feature can further reduce software overhead, as two different compare values can be programmed to control a sequence of transitions for a signal. See **Section 17.5.5** for details for this operation.

In all Compare Modes, the comparator performs an ‘equal to’ comparison. This means, a match is only detected when the timer contents are equal to the contents of a compare register. In addition, the comparator is only enabled in the clock cycle directly after the timer was incremented by hardware. This is done to prevent repeated matches if the timer does not operate with the highest possible input clock (either in timer or counter mode). In this case, the timer contents would remain at the same value for several or up to thousands of cycles. This operation has the side-effect, that software modifications of the timer contents will have no effect regarding the comparator. If a timer is set by software to the same value stored in one of the compare registers, no match will be detected. If a compare register is set to a value smaller than the current timer contents, no action will take place.

For the exact operation of the port output function, please see **Section 17.6**.

When two or more compare registers are programmed to the same compare value[1)], their corresponding interrupt request flags will be set and the selected output signals will be generated after the allocated timer is incremented to this compare value. Further compare events on the same compare value are disabled[2)] until the timer is incremented again or written to by software. After a reset, compare events for register CCy will only become enabled, if the allocated timer has been incremented or written to by software and one of the compare modes described in the following has been selected for this register.

1) In staggered mode these interrupts and output signals are generated sequentially (see **Section 17.8**).

2) Even if more compare cycles are executed before the timer increments (lower timer frequency) a given compare value only results in one single compare event.

17.5.1 Compare Mode 0

This is an interrupt-only mode which can be used for software timing purposes. In this mode, the interrupt request line CCyIRQ is activated each time a match is detected between the contents of the compare register CCy and the allocated timer. A match means, the contents of the timer are equal to (‘=’) the contents of the compare register. Several of these compare events are possible within a single timer period, if the compare value in register CCy is updated during the timer period. The corresponding port signal CCyIO is not affected by compare events in this mode and can be used as general purpose IO.

*Note: If compare mode 0 is programmed for one of the bank2 registers the double-register compare mode may be enabled for this register (see **Chapter 17.5.5**).*

17.5.2 Compare Mode 1

This is a compare mode which influences the associated output signal. Besides this, the basic operation is as in compare mode 0. Each time a match is detected between the contents of the compare register CCy and the allocated timer, the interrupt request line CCyIRQ is activated. In addition, the associated output signal is toggled. Several of these compare events are possible within a single timer period, if the compare value in register CCy is updated during the timer period.

*Note: If compare mode 1 is programmed for one of the bank1 registers the double-register compare mode may be enabled for this register (see **Section 17.5.5**).*

For the exact operation of the port output signal, please see **Section 17.6**.

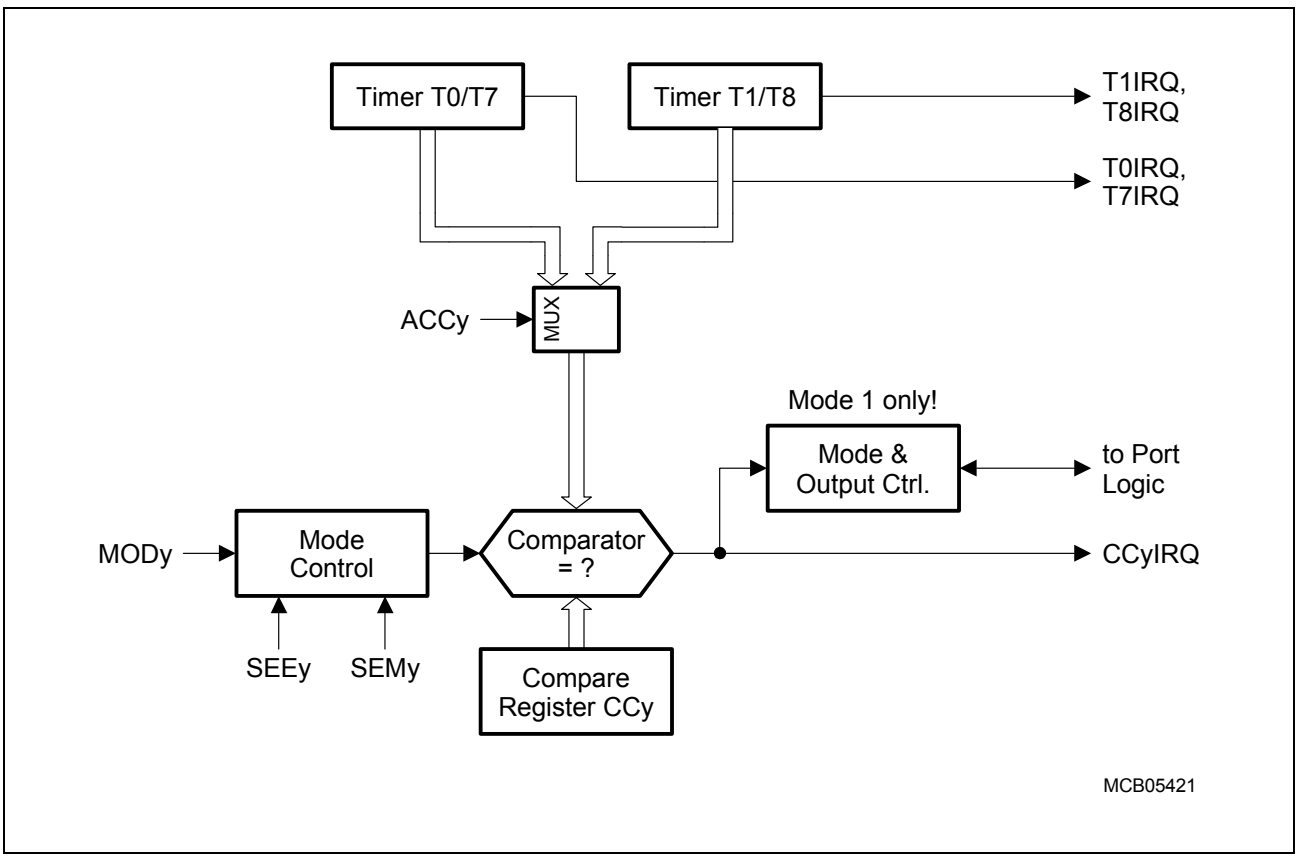


Figure 17-5 Compare Mode 0 and 1 Block Diagram

Note: The signal remains unaffected in compare mode 0.

Figure 17-6 illustrates a few example cases for compare modes 0 and 1.

In all examples, the reload value of the used timer is set to $FFF9_H$. When the timer overflows, it starts counting from this value upwards.

In Case 1, register CCy contains the value $FFFC_H$. When the timer reaches this value, a match is detected, and the interrupt request line CCyIRQ is activated. In compare mode 0, this is all that will happen. In compare mode 1, additionally the associated port output is toggled, causing an inversion of the output signal. If the contents of register CCy are not changed, this operation will take place each time the timer reaches the programmed compare value.

In Case 2, software reloads the compare register CCy with $FFFF_H$ after the first match with $FFFC_H$ has occurred. As the timer continues to count up, it finally reaches this new compare value, and a new match is detected, activating the interrupt request line (both modes) and toggling the output signal (compare mode 1). If then the compare value is left unchanged, the next match will occur when the timer reaches $FFFF_H$ again.
This example illustrates, that further compare matches are possible within the current timer period (this is in contrast to compare modes 2 and 3).

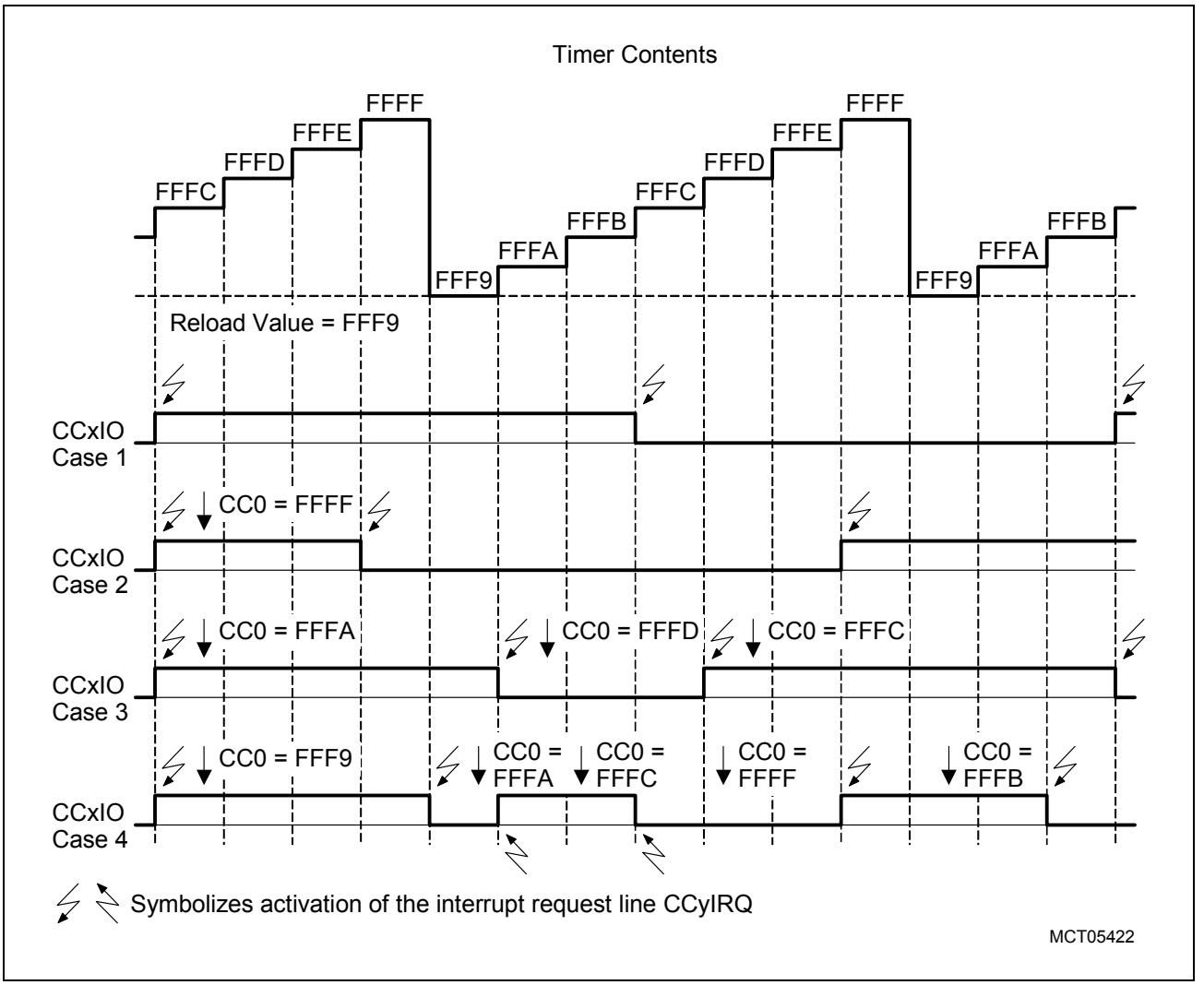


Figure 17-6 Examples for Compare Modes 0 and 1

In Case 3, a new compare value, higher than the current timer contents, causes a new match within the current timer period. The compare register is reloaded with $FFFA_H$ after the first match (at $FFFC_H$). However, the timer has already passed this value. Thus, it will take until the timer reaches $FFFA_H$ in the following timer period to cause the desired compare match. Reloading register CCy now with a value higher than the current timer contents will cause the next match within this period.

In Case 4, the compare values are equal to the timer reload value or to the maximum count value, $FFFF_H$.

17.5.3 Compare Mode 2

Compare mode 2 is an interrupt-only mode similar to compare mode 0. The main difference is that only one compare match, corresponding to one interrupt request, is possible within a given timer period.

when a match is detected in compare mode 2 for the first time within a count period of the allocated timer, the interrupt request line CCyIRQ is activated. In addition, all further compare matches within the current timer period are disabled, even if a new compare value, higher than the current timer contents, would be written to the register. This blocking is only released when the allocated timer overflows. A new compare value written to the compare register after the first match will only go into effect within the following timer period.

17.5.4 Compare Mode 3

Compare mode 3 is based on compare mode 2, but additionally influences the associated port pin. Only one compare event is possible within one timer period.

When a match is detected in compare mode 3 for the first time within a count period of the allocated timer, the interrupt request line CCyIRQ is activated, and the associated output signal is set to 1. In addition, all further compare matches within the current timer period are disabled, even if a new compare value, higher than the current timer contents, would be written to the register. This blocking is only released when the allocated timer overflows. A new compare value written to the compare register after the first match will only go into effect within the following timer period.

The overflow signal is also used to reset the associated output signal to 0.

Special attention has to be paid when the compare value is set equal to the timer reload value. In this case, the compare match signal would try to set the output signal, while the timer overflow tries to reset the output signal. This conflict is avoided such that the state of the output signal is left unchanged in this case.

Note: When the compare value is changed from a value above the current timer contents to a value below the current timer contents, the new value is not recognized before the next timer period.

For the exact operation of the port output signal, please see **Section 17.6**.

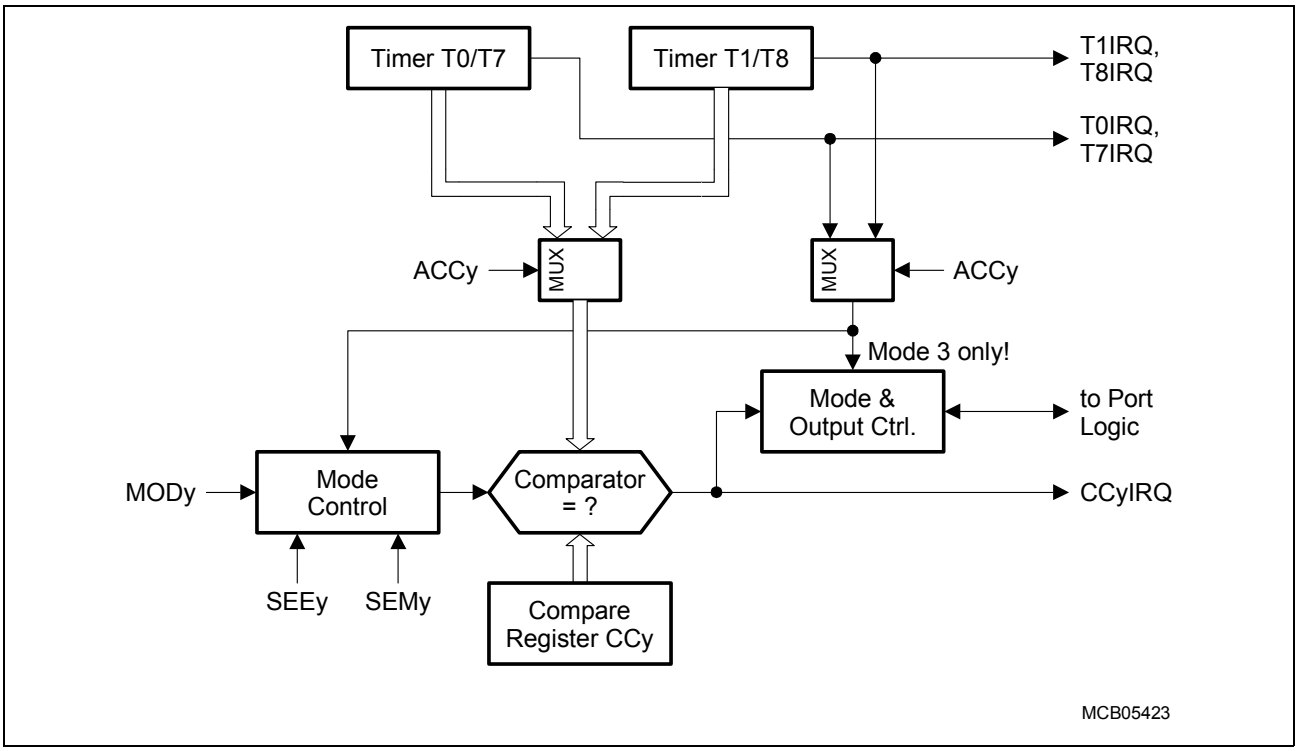


Figure 17-7 Compare Mode 2 and 3 Block Diagram

Note: The port latch and signal remain unaffected in compare mode 2.

Figure 17-8 illustrates a few timing examples for compare modes 2 and 3.

In all examples, the reload value of the used timer is set to $FFF9_H$. When the timer overflows, it starts counting from this value upwards.

In Case 1, register CCy contains the value $FFFC_H$. When the timer reaches this value, a match is detected, and the interrupt request line CCyIRQ is activated. In compare mode 2, this is all that will happen. In compare mode 3, additionally the associated port output is set to 1. The timer continues to count, and finally reaches its overflow. At this point, the port output is reset to 0 again. Note that, although not shown in the diagrams, the overflow signal of the timer also activates the associated interrupt request line TxIRQ. If the contents of register CCy are not changed, the port output will be set again during the following timer period, and reset again when the timer overflows. This operation is ideal for the generation of a pulse width modulated (PWM) signal with a minimum of software overhead. The pulse width is varied by changing the compare value accordingly.

In Case 2, the compare operation is blocked after the first match within a timer period. After the first match at $FFFC_H$, the interrupt request is generated and the port output is set. In addition, further compare matches are disabled. If now a new compare value is written to register CCy, no interrupt request and no port output influence will take place, although the new compare value is higher than the current timer contents. Only after the overflow of the timer, the compare logic is enabled again, and the next match will be

detected at $FFFF_H$. One can see, that this operation is ideal for PWM generation, as software can write a new compare value regardless of whether this value is higher or lower than the current timer contents. It is assured that the new value (usually written to the compare register in the appropriate interrupt service routine) will only go into effect during the following timer period.

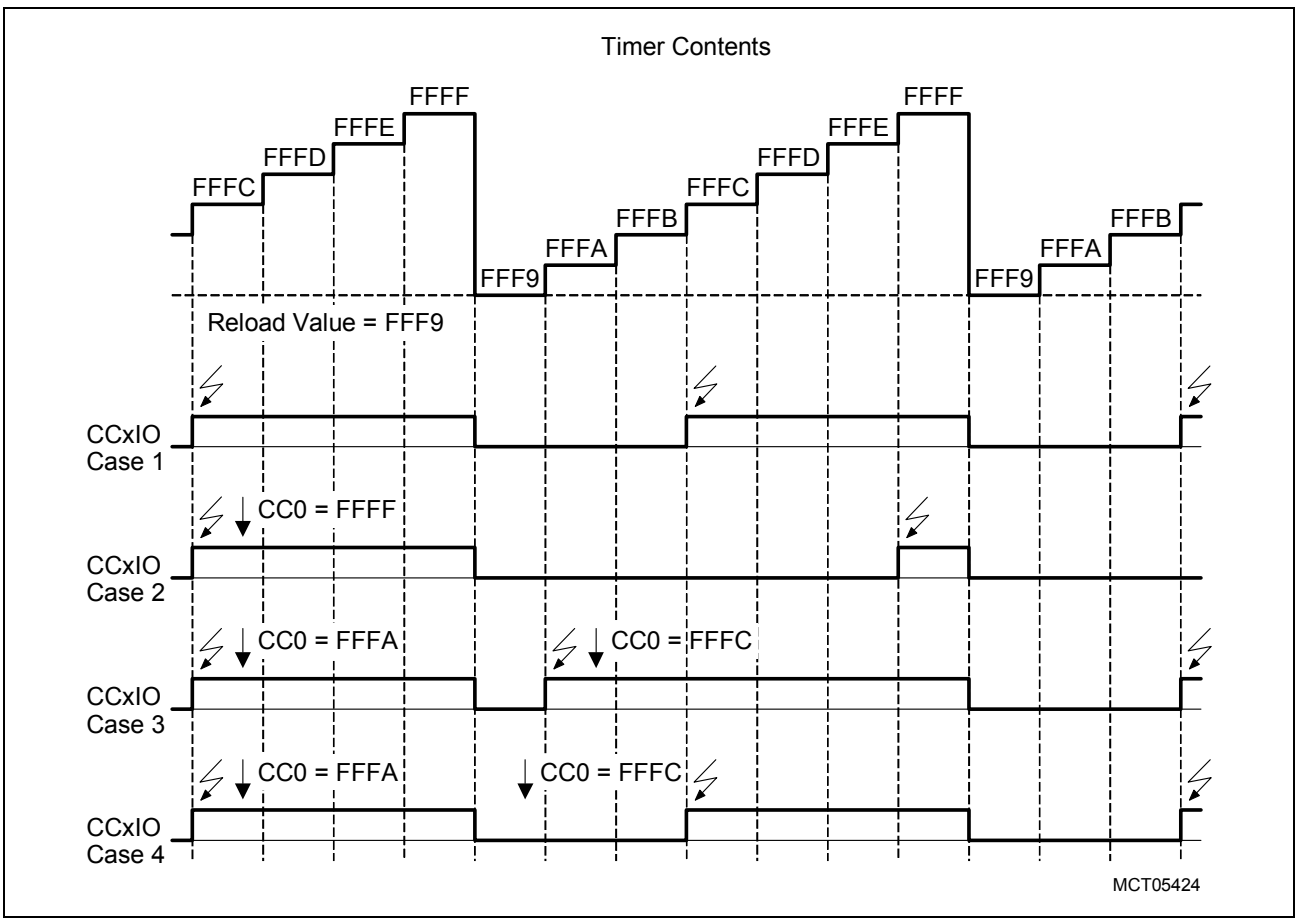


Figure 17-8 Timing Example for Compare Modes 2 and 3

Note: In compare mode 2, only interrupt requests are generated, in mode 3, also the output signals are generated.

In Case 3, further examples for the operation of the compare match blocking are illustrated.

In Case 4, a new compare value is written to a compare register before the first match within the timer period. One can see that, of course, the originally programmed compare match (at $FFFA_H$) will not take place. The first match will be detected at $FFFC_H$. However, it is important to note that the reprogramming of the compare register took place asynchronously - this means, the register was written to without any regard to the current contents of the timer. This is dangerous in the sense that the effect of such an asynchronous reprogramming is not easily predictable. If the timer would have already reached the originally programmed compare value of $FFFA_H$ by the time the software

wrote to the register, a match would have been detected and the reprogramming would go into effect during the next timer period.

The examples in **Figure 17-9** show special cases for compare modes 2 and 3. Case 1 illustrates the effect when the compare value is equal to the reload value of the timer. An interrupt is generated in both modes. In mode 3, the output signal is not affected - it remains at the high level. Setting the compare value equal to the reload value easily enables a 100% duty cycle signal for PWM generation. The important advantage here is that the compare interrupt is still generated and can be used to reload the next compare value. Thus, no special treatment is required for this case (see Case 3).

Cases 2, 4, and 5 show different options for the generation of a 0% duty cycle signal. Case 2 shows an asynchronous reprogramming of the compare value equal to the reload value. At the end of the current timer period, a compare interrupt will be generated, which enables software to set the next compare value. The disadvantage of this method is that at least two timer periods will pass until a new regular compare value can go into effect. The compare match with the reload value $FFF9_H$ will block further compare matches during that timer period. This is additionally illustrated by Case 4.

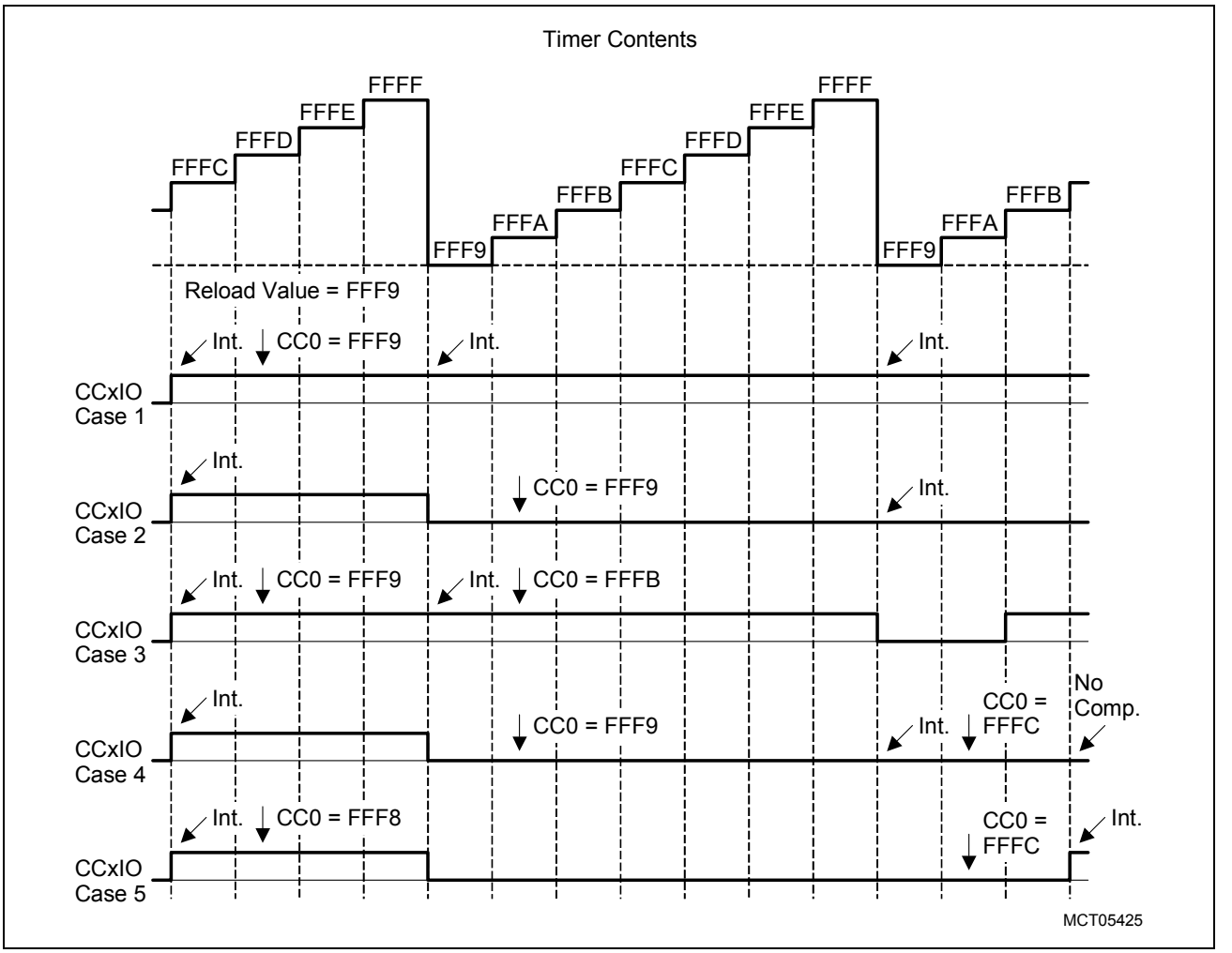


Figure 17-9 Special Cases in Compare Modes 2 and 3

Case 5 shows an option to get around this problem. Here, the compare register is reloaded with $FFF8_H$, a value which is lower than the timer reload value. Thus, the timer will never reach this value, and no compare match will be detected. The output signal will be set to 0 after the first timer overflow. However, after the second overflow, software now reloads the compare register with a regular compare value. As no compare blocking has taken place (since there was no compare match), the newly written compare value will go into effect during the current timer period.

17.5.5 Double-Register Compare Mode

The Double-Register Compare Mode makes it possible to further reduce software overhead for a number of applications. In this mode, two compare registers work together to control one output. This mode is selected via the DRM register, or by a special combination of compare modes for the two registers.

For double-register compare mode, the 16 capture/compare registers of a CAPCOM unit are regarded as two banks of 8 registers each. The lower eight registers form bank1, while the upper eight registers form bank2. For double-register mode, a bank1 register and a bank2 register form a register pair. Both registers of this register pair operate on the pin associated with the bank1 register.

The relationship between the bank1 and bank2 register of a pair and the effected output pins for double-register compare mode is listed in **Table 17-3**.

Table 17-3 Register Pairs for Double-Register Compare Mode

CAPCOM1 Unit				CAPCOM2 Unit			
Register Pair		Used Output Pin	Control Bitfield in CC1DRM	Register Pair		Used Output Pin	Control Bitfield in CC2DRM
Bank 1	Bank 2			Bank 1	Bank 2		
CC0	CC8	CC0IO	DR0M	CC16	CC24	CC16IO	DR0M
CC1	CC9	CC1IO	DR1M	CC17	CC25	CC17IO	DR1M
CC2	CC10	CC2IO	DR2M	CC18	CC26	CC18IO	DR2M
CC3	CC11	CC3IO	DR3M	CC19	CC27	CC19IO	DR3M
CC4	CC12	CC4IO	DR4M	CC20	CC28	CC20IO	DR4M
CC5	CC13	CC5IO	DR5M	CC21	CC29	CC21IO	DR5M
CC6	CC14	CC6IO	DR6M	CC22	CC30	CC22IO	DR6M
CC7	CC15	CC7IO	DR7M	CC23	CC31	CC23IO	DR7M

The double-register compare mode can be programmed individually for each register pair. Double-register compare mode can be selected via a certain combination of compare modes for the two registers of a pair. The bank1 register must be programmed

for mode 1 (with port influence), while the bank2 register must be programmed for mode 0 (interrupt-only).

Double-register compare mode can be controlled (this means, enabled or disabled) for each register pair via the associated control bitfield DRxM in register CC1_DRM or CC2_DRM, respectively.

CC1_DRM
Double-Reg. Cmp. Mode Reg. SFR ($FF5A_H$/AD_H) Reset Value: 0000_H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DR7M		DR6M		DR5M		DR4M		DR3M		DR2M		DR1M		DR0M	
rw		rw		rw		rw		rw		rw		rw		rw	

CC2_DRM
Double-Reg. Cmp. Mode Reg. SFR ($FF2A_H$/95_H) Reset Value: 0000_H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DR7M		DR6M		DR5M		DR4M		DR3M		DR2M		DR1M		DR0M	
rw		rw		rw		rw		rw		rw		rw		rw	

Field	Bits	Type	Description
DRxM	[1:0], [3:2], [5:4], [7:6], [9:8], [11:10], [13:12], [15:14]	rw	**Double Register x Compare Mode Selection** 00 DRM is controlled via the combination of compare modes 1 and 0 (compatibility mode) 01 DRM disabled regardless of compare modes 10 DRM enabled regardless of compare modes **11 Reserved** *Note: "x" indicates the register pair index in a bank.*

Double-register compare mode can be controlled individually for each of the register pairs.

In the block diagram of the double-register compare mode (**Figure 17-10**), a bank2 register will be referred to as CCz, while the corresponding bank1 register will be referred to as CCy.

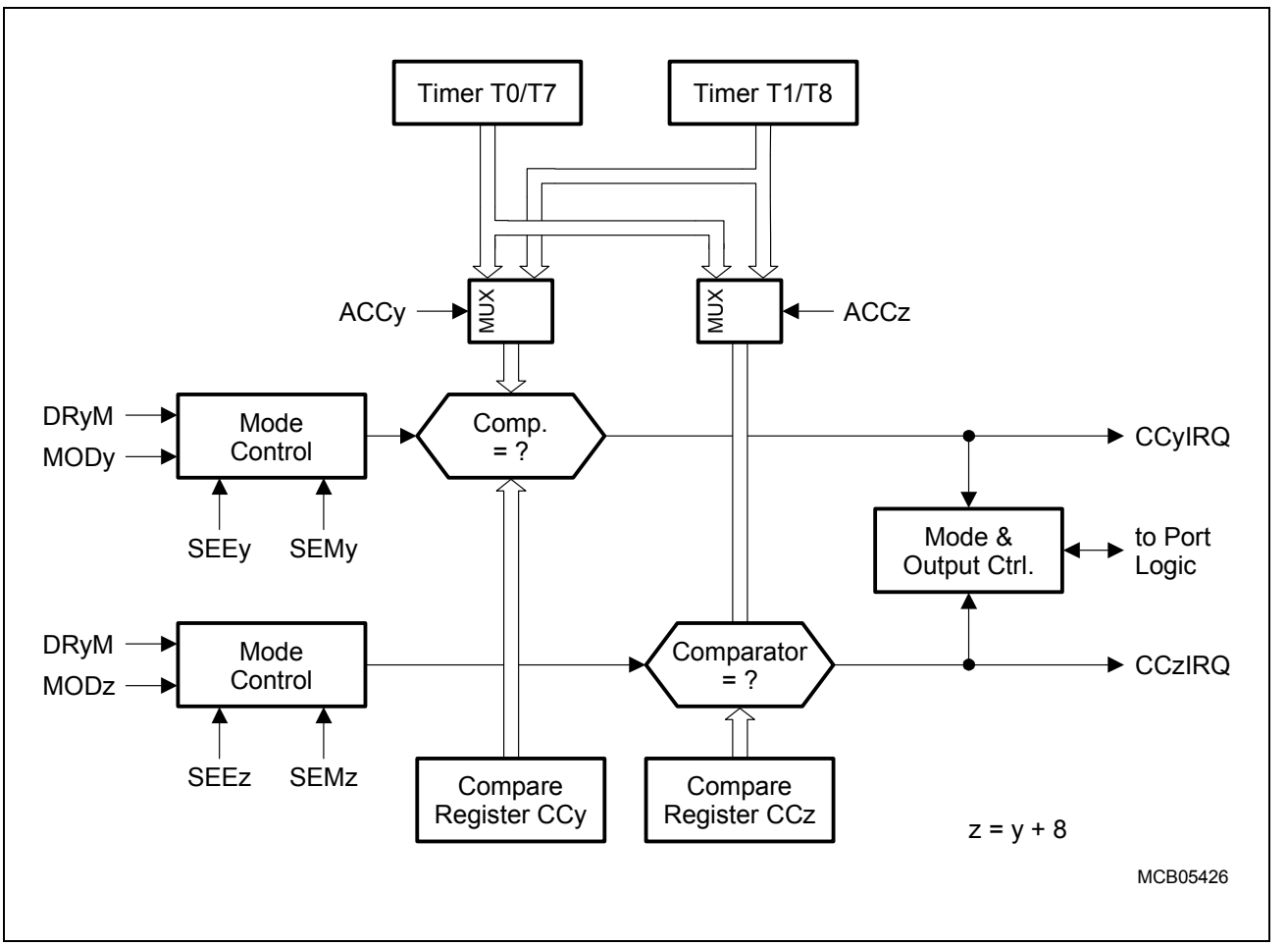


Figure 17-10 Double-Register Compare Mode Block Diagram

When a match is detected for one of the two registers in a register pair (CCy or CCz), the associated interrupt request line (CCyIRQ or CCzIRQ) is activated, and pin CCyIO, corresponding to the bank1 register CCy, is toggled. The generated interrupt always corresponds to the register that caused the match.

Note: If a match occurs simultaneously for both register CCy and register CCz of the register pair, pin CCyIO will be toggled only once, but two separate compare interrupt requests will be generated.

Each of the two registers of a pair can be individually allocated to one of the two timers in the CAPCOM unit. This offers a wide variety of applications, as the two timers can run in different modes with different resolution and frequency. However, this might require sophisticated software algorithms to handle the different timer periods.

Note: The signals CCzIO (which do not serve for double-register compare mode) may be used for general purpose IO.

17.6 Compare Output Signal Generation

This section discusses the interaction between the CAPCOM Unit and the Port Logic. The block diagram illustrated in **Figure 17-11** details the logic of the block “Mode & Output Control”, shown in **Figure 17-5**, **Figure 17-7**, and **Figure 17-10**.

Each output signal is latched in its associated bit of the respective output latch register CCx_OUT. The individual bits are updated each time an associated compare event occurs. The bits of these registers are connected to the respective port pins as an alternate output function of a port line.

Compare signals can also directly affect the associated port output latch Px. In this case, the port latch must be selected for the respective pin. The direct port latch option is disabled in non-staggered mode or it can be disabled by setting bit PL in register CCx_IOC.

Register CCx_OUT is always updated in parallel to the update of the port output latch.

CC1_OUT
Compare Output Reg. **SFR ($FF5C_H$/AE_H)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CC15IO	**CC14IO**	**CC13IO**	**CC12IO**	**CC11IO**	**CC10IO**	**CC9IO**	**CC8IO**	**CC7IO**	**CC6IO**	**CC5IO**	**CC4IO**	**CC3IO**	**CC2IO**	**CC1IO**	**CC0IO**
rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

CC2_OUT
Compare Output Reg. **SFR ($FF2C_H$/96_H)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CC15IO	**CC14IO**	**CC13IO**	**CC12IO**	**CC11IO**	**CC10IO**	**CC9IO**	**CC8IO**	**CC7IO**	**CC6IO**	**CC5IO**	**CC4IO**	**CC3IO**	**CC2IO**	**CC1IO**	**CC0IO**
rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
CCyIO	15 … 0	rwh	**Compare Output for Channel y** Alternative port output for the associated port pin.

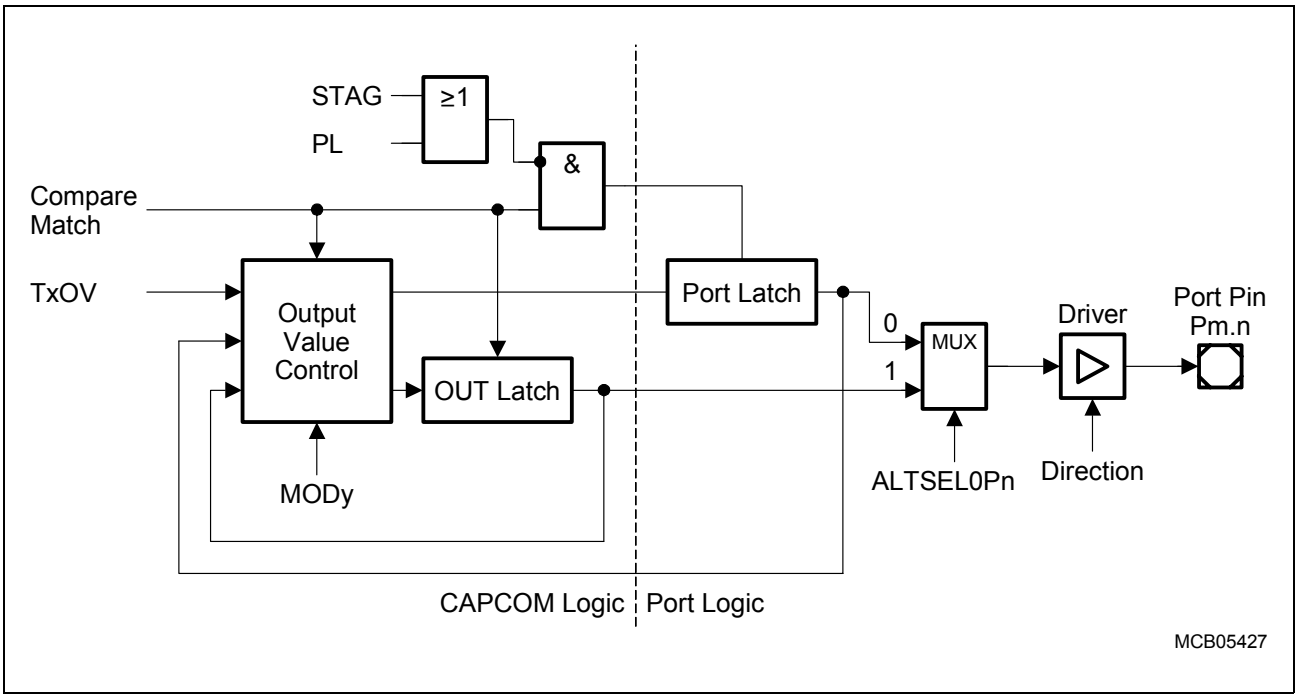


Figure 17-11 Port Output Block Diagram for Compare Modes

Note: A compare output signal is visible at the pin only in compare modes 1 or 3.

The output signal of a compare event can either be a 1, a 0, the complement of the current level, or the previous level. The block ‘Output Value Control’ determines the correct new level based on the compare event, the timer overflow signal, and the current states of the Port and OUT latches. For the output toggle function (e.g. in compare mode 1), the state of the output latch is read, inverted, and then written back.

The associated output pins either drives the port latch signal or the OUT signal, selected by register ALTSEL (see **Figure 17-11**).

Note: If the port output latch is written to by software at the same time it would be altered by a compare event, the software write will have priority. In this case the hardware-triggered change will not become effective.

17.7 Single Event Operation

If an application requires that one and only one compare event needs to take place (within a certain time frame), single event operation helps to reduce software overhead and to eliminate the need for fast reaction upon events.

In order to achieve a single event operation without this feature, software would have to either disable the compare mode or write a new value, which is outside of the count range of the timer, into the compare register, after the programmed compare match has taken place. Thus, usually an interrupt service routine is required to perform this operation. Interrupt response time may be critical if the timer period is very short - the disable operation needs to be completed before the timer would reach the same value again.

The single event operation eliminates the need for software to react after the first compare match. The complete operation can be set up before the event, and no action is required after the event. The hardware takes care of generating only one event, and then disabling all further compare matches.

This option is programmed via the Single Event Mode register CCx_SEM and the Single Event Enable register CCx_SEE. Each register provides one bit for each CCy register of a unit.

CC1_SEM
Single Event Mode Ctrl. Reg. **SFR ($FE28_H/14_H$)** **Reset Value: 0000_H**

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

CC2_SEM
Single Event Mode Ctrl. Reg. **SFR ($FE2C_H/16_H$)** **Reset Value: 0000_H**

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

Field	Bits	Type	Description
SEMy	15 … 0	rw	**Single Event Mode Control** 0 Single Event Mode disabled for channel y 1 Single Event Mode enabled for channel y

CC1_SEE
Single Event Enable Reg. **SFR ($FE2E_H/17_H$)** **Reset Value: 0000_H**

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

CC2_SEE
Single Event Enable Reg. **SFR ($FE2A_H/15_H$)** **Reset Value: 0000_H**

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

Field	Bits	Type	Description
SEEy	15 … 0	rw	**Single Event Enable Control** 0 Single Event disabled for channel y 1 Single Event enabled for channel y *Note: This bit is cleared by hardware after the event.*

To setup a single event operation for a CCy register, software first programs the desired compare operation and compare value, and then sets the respective bit in register CCx_SEM to enable the single event mode. At last, the respective event enable bit in register CCx_SEE is set.

When the programmed compare match occurs, all operations of the selected compare mode take place. In addition, hardware automatically disables all further compare matches and reset the event enable bit in register CCx_SEE to 0. As long as this bit is cleared, any compare operation is disabled. To setup a new event, this bit must first be set again.

17.8 Staggered and Non-Staggered Operation

The CAPCOM units can run in one of two basic operation modes: Staggered Mode and Non-Staggered Mode. The selection between these modes is performed via register IOC.

CC1_IOC
I/O Control Register **ESFR ($F062_H/31_H$)** **Reset Value: 0000_H**

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

CC2_IOC
I/O Control Register **ESFR ($F066_H/33_H$)** **Reset Value: 0000_H**

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

Field	Bits	Type	Description
STAG	2	rw	**Staggered Mode Control** 0 CAPCOM operates in Staggered Mode 1 CAPCOM operates in Non-Staggered Mode
PL	1	rw	**Port Lock Control** 0 Compare output signals affect the associated port output latch 1 Direct influence of the port output latch by the compare output signals is disabled

Note: Whenever Non-Staggered Mode is enabled (STAG = 1) or Port Lock is activated (PL = 1), the port output registers are not changed by the CAPCOM unit.

In staggered mode, a CAPCOM operation cycle consists of 8 module clock cycles, and the outputs of the compare events of the different registers are staggered, that is, the outputs for compare matches with the same compare value are not switched at the same time, but with a fixed time delay. This operation helps to reduce noise and peak power consumption caused by simultaneous switching outputs.

In non-staggered Mode, a CAPCOM operation cycle is equal to one module clock cycle, and all compare outputs for compare events with the same compare value are switched

in the same clock cycle. This mode offers a faster operation and increased resolution of the CAPCOM unit, 8 times higher than in staggered mode.

Staggered Mode

Figure 17-12 illustrates the staggered mode operation. In this example, all CCy registers are programmed for compare mode 3.

Registers CC0, CC1, and CC2 are all programmed for a compare value of $FFFE_H$. When the timer increments to $FFFE_H$, the comparator detects a match for all of the three registers. The output CC0IO of register CC0 is switched to 1 one cycle after the comparator match. However, the outputs CC1IO and CC2IO are not switched at the same time, but one, respectively two cycles later. This staggering of the outputs continues for all registers including register CC7. The number of the register indicates the delay of the output signal in clock cycles - the output of register CC7 is switched 7 cycles later than the one of register CC0. In the example, the compare value for register CC7 is set to $FFFD_H$. Thus, the output is switched in the last clock cycle of the CAPCOM cycle in which the timer reached $FFFD_H$.

When the timer overflows, all compare outputs are reset to 0 (compare mode 3). Again, the staggering of the output signals can be seen from **Figure 17-12**.

Looking at registers CC8 through CC15 shows that their outputs are switched in parallel to the respective outputs of registers CC0 through CC15. In fact, the staggering is performed in parallel for the upper and the lower register bank. In this way, it is assured, that both compare signals of a register pair in double-register compare mode operate simultaneously.

In staggered mode direct port latch switching (see **Section 17.6**) is possible. However, it is possible to use the alternate output function option of the associated port pins to output the compare signals.

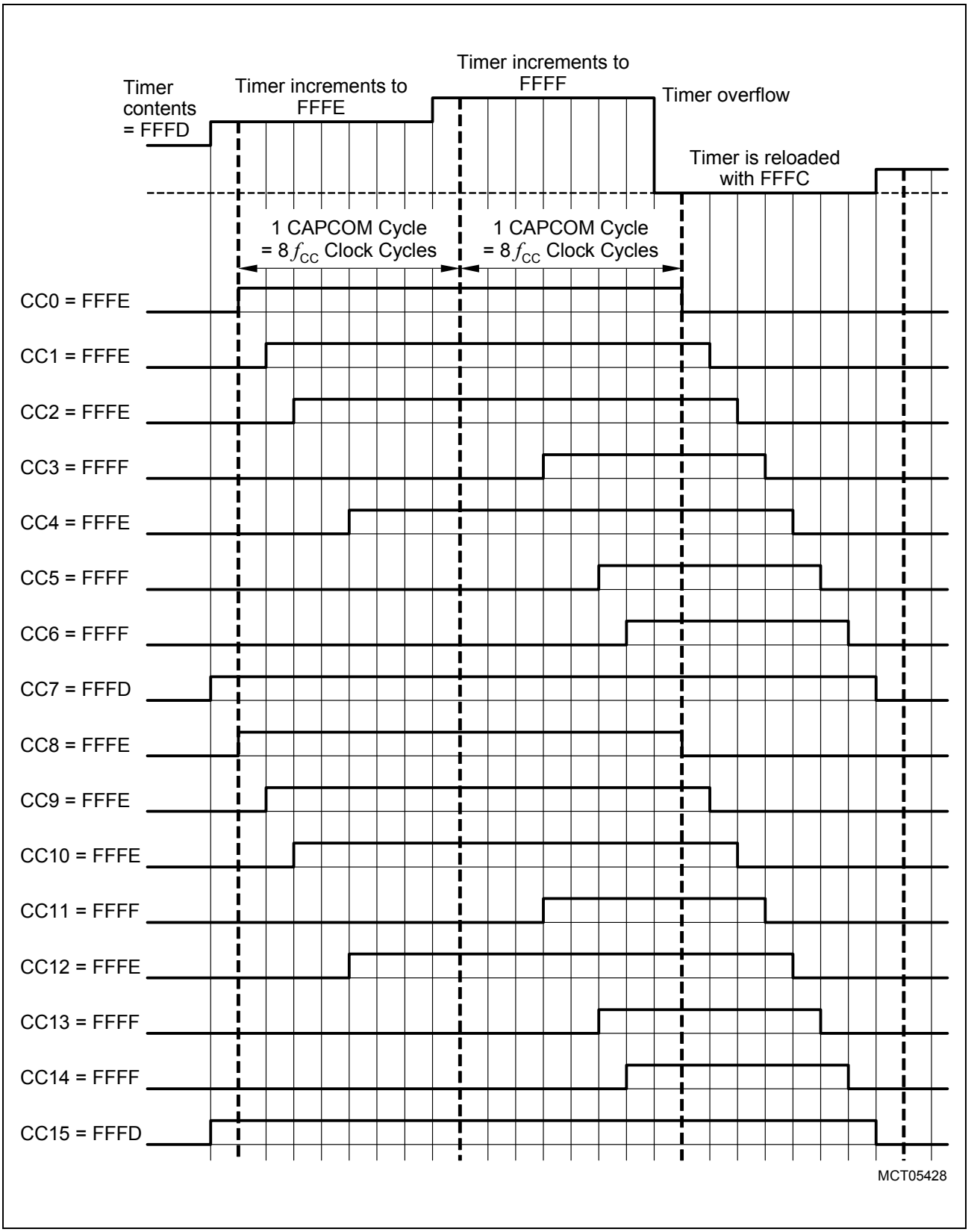


Figure 17-12 Staggered Mode Operation

Non-Staggered Mode

To gain maximum speed and resolution with the CAPCOM unit, it can be switched to non-staggered mode. In this mode, one CAPCOM operation cycle is equal to one module clock cycle. Timer increment and the comparison of its new contents with the contents of the compare register takes place within one clock cycle. The appropriate output signals are switched in the following clock cycle (in parallel to the next possible timer increment and comparison).

Figure 17-13 illustrates the non-staggered mode. Note that when the timer overflows, it also takes one additional clock cycle to switch the output signals.

Note: In non-staggered mode, direct port latch switching is disabled.

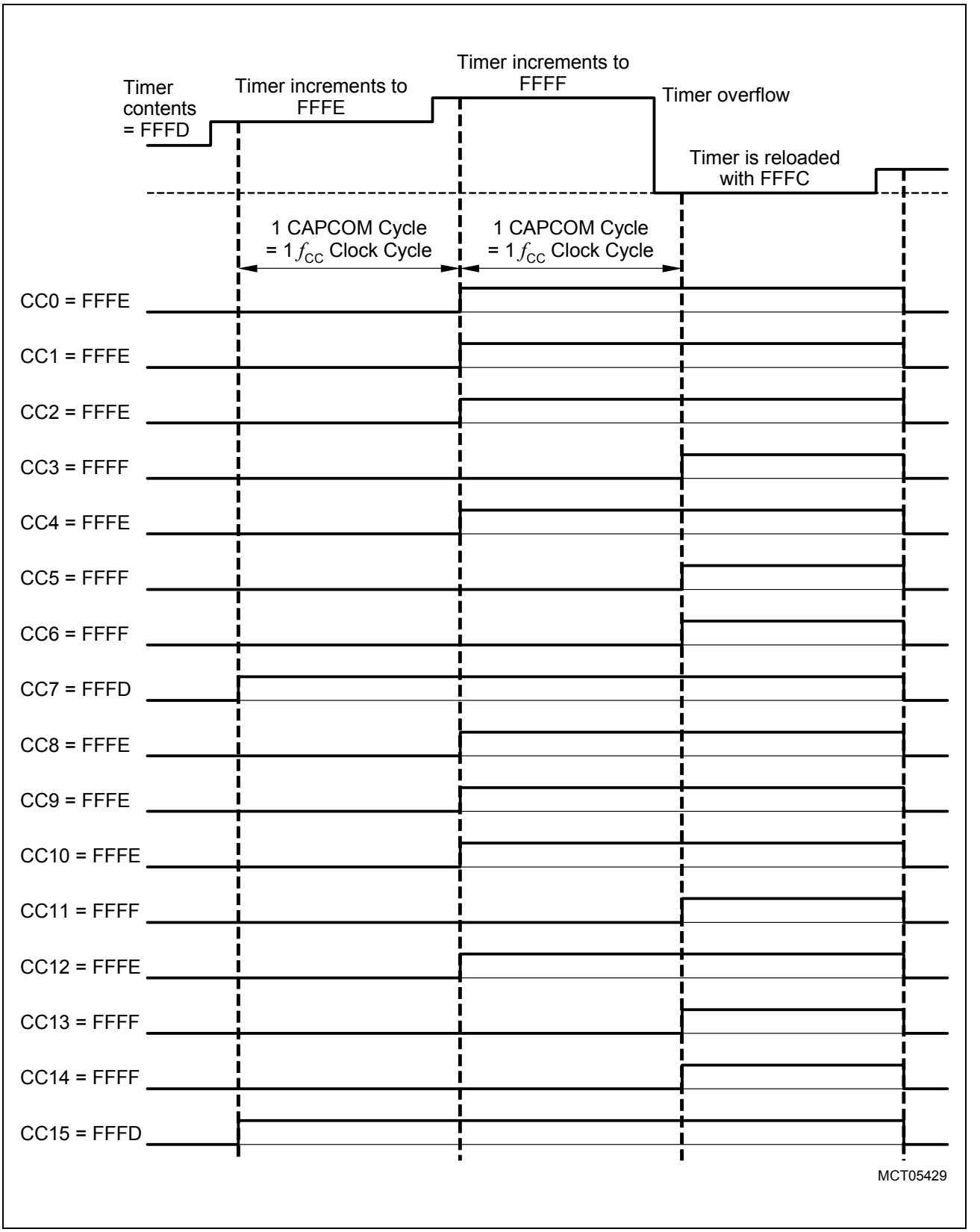


Figure 17-13 Non-Staggered Mode Operation

17.9 CAPCOM Interrupts

Upon a capture or compare event, the interrupt request flag CCxIR for the respective capture/compare register CCx is automatically set. This flag can be used to generate an interrupt or trigger a PEC service request when enabled by the interrupt enable bit CCxIE. Capture interrupts can be regarded as external interrupt requests with the additional feature of recording the time at which the triggering event occurred.

Each of the capture/compare registers has its own bitaddressable interrupt control register and its own interrupt vector allocated. These registers are organized in the same way as all other interrupt control registers. The basic register layout is shown below, **Table 17-4** lists the associated addresses.

CCx_CCyIC
CAPCOM Intr. Ctrl. Reg. **(E)SFR (Table 17-4)** **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-							GPX	CCy IR	CCy IE	ILVL				GLVL	
-	-	-	-	-	-	-	rw	rwh	rw	rw				rw	

Note: Please refer to the general Interrupt Control Register description for an explanation of the control fields.

Table 17-4 CAPCOM Unit Interrupt Control Register Addresses

CAPCOM1 Unit			CAPCOM2 Unit		
Register Name	**Address**	**Reg. Space**	**Register Name**	**Address**	**Reg. Space**
CC1_CC0IC	$FF78_H/BC_H$	SFR	CC2_CC16IC	$F160_H/B0_H$	ESFR
CC1_CC1IC	$FF7A_H/BD_H$	SFR	CC2_CC17IC	$F162_H/B1_H$	ESFR
CC1_CC2IC	$FF7C_H/BE_H$	SFR	CC2_CC18IC	$F164_H/B2_H$	ESFR
CC1_CC3IC	$FF7E_H/BF_H$	SFR	CC2_CC19IC	$F166_H/B3_H$	ESFR
CC1_CC4IC	$FF80_H/C0_H$	SFR	CC2_CC20IC	$F168_H/B4_H$	ESFR
CC1_CC5IC	$FF82_H/C1_H$	SFR	CC2_CC21IC	$F16A_H/B5_H$	ESFR
CC1_CC6IC	$FF84_H/C2_H$	SFR	CC2_CC22IC	$F16C_H/B6_H$	ESFR
CC1_CC7IC	$FF86_H/C3_H$	SFR	CC2_CC23IC	$F16E_H/B7_H$	ESFR
CC1_CC8IC	$FF88_H/C4_H$	SFR	CC2_CC24IC	$F170_H/B8_H$	ESFR
CC1_CC9IC	$FF8A_H/C5_H$	SFR	CC2_CC25IC	$F172_H/B9_H$	ESFR
CC1_CC10IC	$FF8C_H/C6_H$	SFR	CC2_CC26IC	$F174_H/BA_H$	ESFR
CC1_CC11IC	$FF8E_H/C7_H$	SFR	CC2_CC27IC	$F176_H/BB_H$	ESFR
CC1_CC12IC	$FF90_H/C8_H$	SFR	CC2_CC28IC	$F178_H/BC_H$	ESFR
CC1_CC13IC	$FF92_H/C9_H$	SFR	CC2_CC29IC	$F184_H/C2_H$	ESFR
CC1_CC14IC	$FF94_H/CA_H$	SFR	CC2_CC30IC	$F18C_H/C6_H$	ESFR
CC1_CC15IC	$FF96_H/CB_H$	SFR	CC2_CC31IC	$F194_H/CA_H$	ESFR

17.10 External Input Signal Requirements

The external input signals of a CAPCOM unit are sampled by the CAPCOM logic based on the module clock and the basic operation mode (staggered or non-staggered mode). To assure that a signal level is recognized correctly, its high or low level must be held active for at least one complete sampling period.

The duration of a sampling period is one module clock cycle in non-staggered mode, and 8 module clock cycles in staggered mode. To recognize a signal transition, the signal needs to be sampled twice. If the level of the first sampling is different to the level detected during the second sampling, a transition is recognized. Therefore, a minimum of two sampling periods are required for the sampling of an external input signal. Thus, the maximum frequency of an input signal must not be higher than half the module clock frequency in non-staggered mode, and a 1/16th of the module clock frequency in staggered mode.

Table 17-5 summarizes the requirements and limits for external input signals.

Table 17-5 CAPCOM External Input Signal Limits

	Non-Staggered Mode	Staggered Mode
Maximum Input Frequency	$f_{CC}/2$	$f_{CC}/16$
Minimum Input Signal Level Duration	$1/f_{CC}$	$8/f_{CC}$

In order to use an external signal as a count or capture input, the port pin to which it is connected must be configured as input.

Note: For example for test purposes a pin used as a count or capture input may be configured as output. Software or an other peripheral may control the respective signal and thus trigger count or capture events.

In order to cause a compare output signal to be seen by the external world, the associated port pin must be configured as output. Compare output signals can either directly switch the port latch, or the output of the CCx_OUT latch is used as an alternate output function of a port.

17.11 Interfaces of the CAPCOM Units

The CAPCOM units CAPCOM1 (see **Figure 17-14**) and CAPCOM2 (see **Figure 17-15**) are connected to their environment in different ways.

Internal Connections

The overflow/underflow signal T6OUF of GPT2 timer T6 is connected to the CAPCOM units, providing an optional clock source for the CAPCOM timers.

The 18 interrupt request lines of each CAPCOM unit are connected to the interrupt control block.

Note: The input lines from Port 2, connected with the CAPCOM1 unit, can also be used as individual external interrupt inputs.

External Connections

The capture/compare signals of both CAPCOM units are connected with input/output ports of the XC161. Depending on the selected direction, these ports may provide capture trigger signals from the external system or issue compare output signals to external circuitry.

Note: Capture trigger signals may also be derived from output pins. In this case, software can generate the trigger edges, for example.

Timers T0 and T7 can be clocked by an external signal. CAPCOM2's timer input signal T7IN shares a port pin with CAPCOM1's input/output pin CC15IO (see **Figure 17-15**). Operations in both CAPCOM units can so be combined:

- the CAPCOM1 compare output can be used to clock CAPCOM2 timer T7, or
- the CAPCOM2 count input signal can be recorded by a CAPCOM1 capture function.

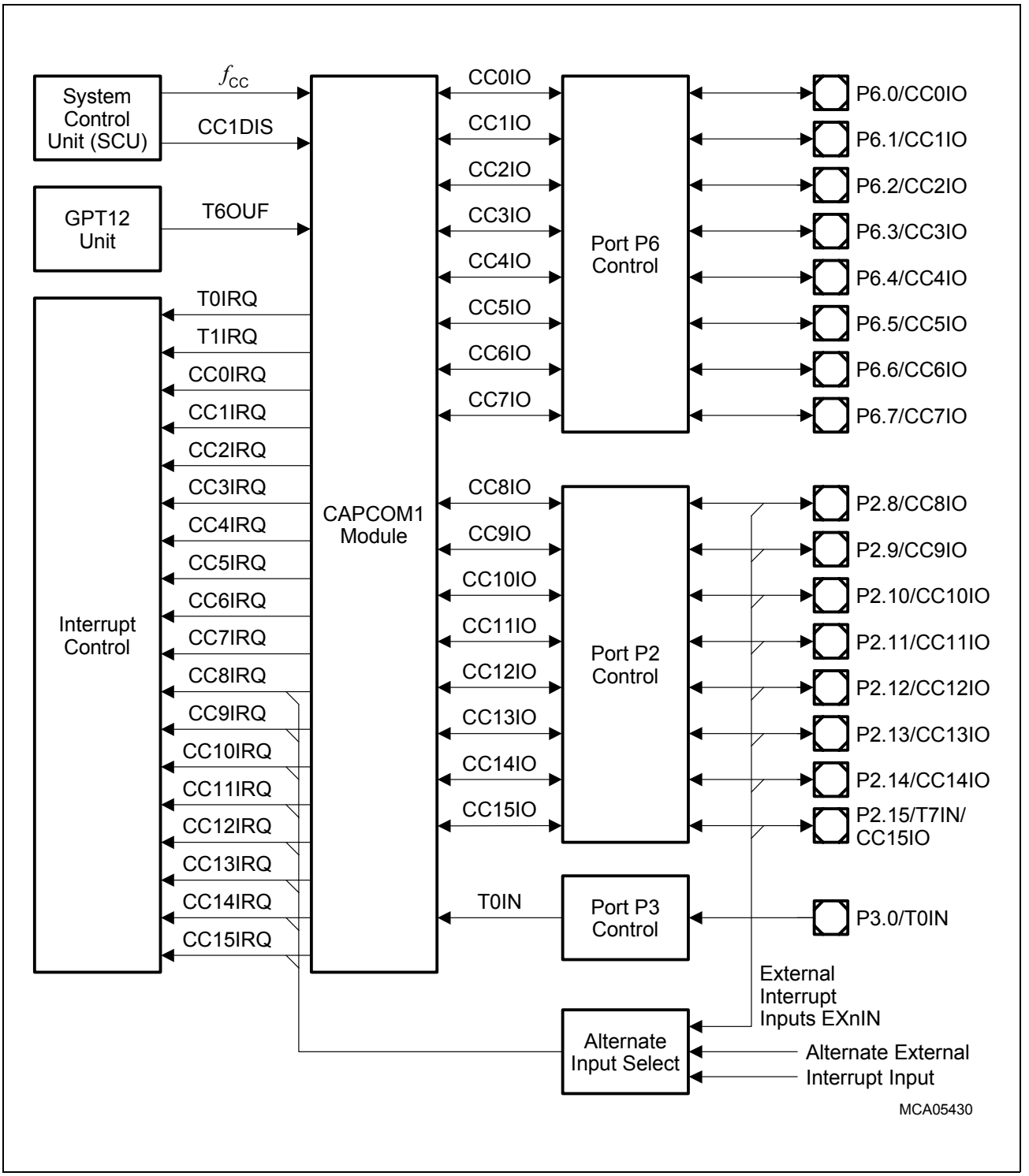


Figure 17-14 CAPCOM1 Unit Interfaces

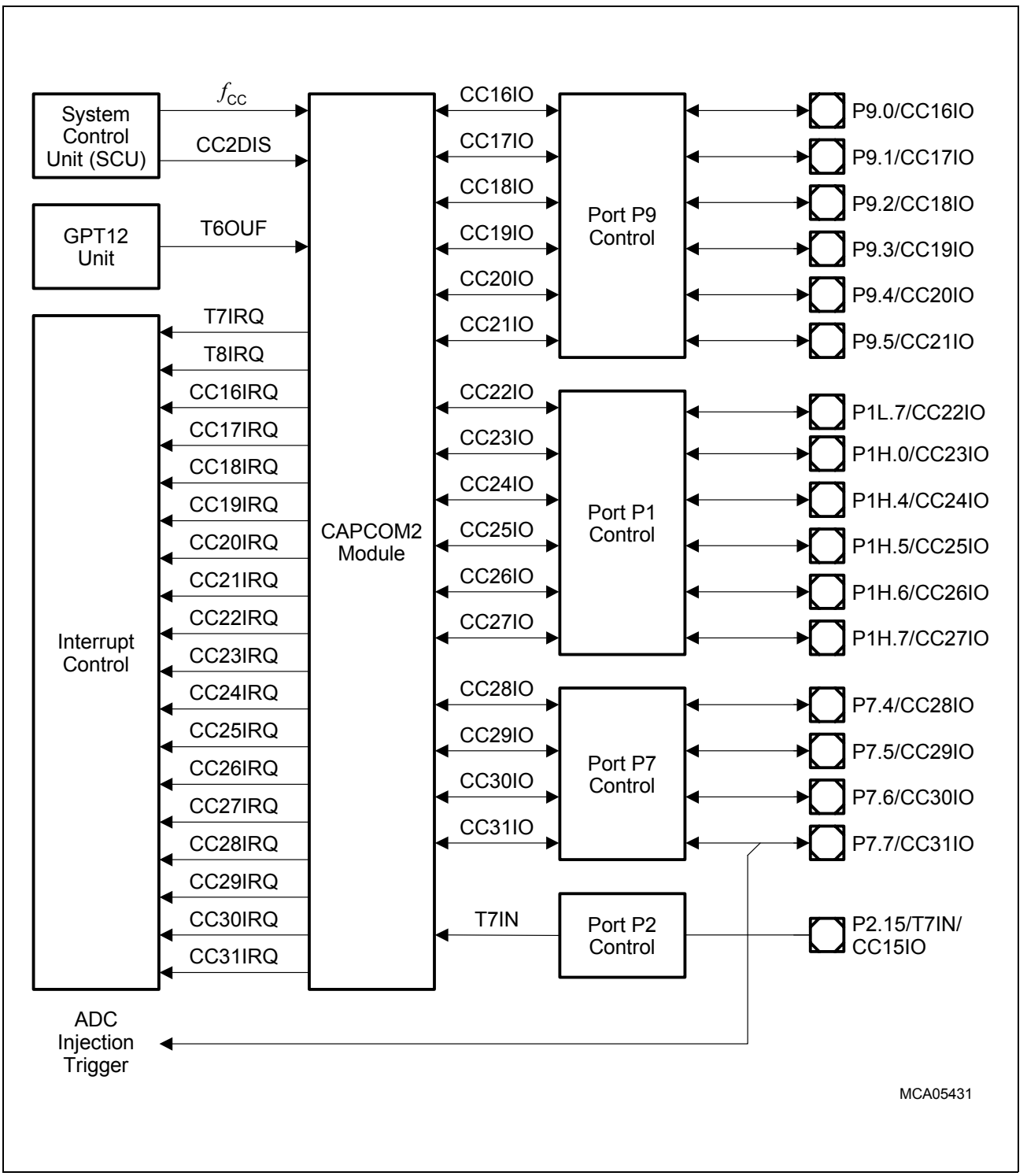


Figure 17-15 CAPCOM2 Unit Interfaces

18 Asynchronous/Synchronous Serial Interface (ASC)

The XC161 contains two Asynchronous/Synchronous Serial Interfaces, ASC0 and ASC1. The following sections present the general features and operations of such an ASC module. The final section describes the actual implementation of the two ASC modules including their interconnections with other on-chip modules.

The ASC supports full-duplex asynchronous communication and half-duplex synchronous communication. The ASC provides the following features and functions.

Features and Functions

- Full-duplex asynchronous operating modes
  - 8- or 9-bit data frames, LSB first
  - Parity bit generation/checking
  - One or two stop bits
  - Baudrate from 2.5 Mbit/s to 50 bit/s (@ 40 MHz module clock f_{ASC})
- Multiprocessor Mode for automatic address/data byte detection
- Loopback capability
- Support for IrDA data transmission up to 115.2 kbit/s maximum
- Half-duplex 8-bit synchronous operating mode
  - Baudrate from 5 Mbit/s to 202 bit/s (@ 40 MHz module clock f_{ASC})
- Double buffered transmitter/receiver
- Interrupt generation
  - On a transmitter buffer empty condition
  - On a transmit last bit of a frame condition
  - On a receiver buffer full condition
  - On an error condition (frame, parity, overrun error)
- Autobaud detection unit for asynchronous operating modes
  - Detection of standard baudrates
    1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, and 230400 bit/s
  - Detection of non-standard baudrates
  - Detection of Asynchronous Modes
  - 7 bit, even parity; 7 bit, odd parity; 8 bit, even parity; 8 bit, odd parity; 8 bit, no parity
  - Automatic initialization of control bits and baudrate generator after detection
  - Detection of a serial two-byte ASCII character frame
- FIFO
  - 8-stage receive FIFO (RXFIFO), 8-stage transmit FIFO (TXFIFO)
  - Independent control of RXFIFO and TXFIFO
  - 9-bit FIFO data width
  - Programmable Receive/Transmit Interrupt Trigger Level
  - Receive and transmit FIFO filling level indication
  - Overrun and Underflow error generation

Figure 18-1 shows all functional relevant interfaces associated with the ASC Kernel.

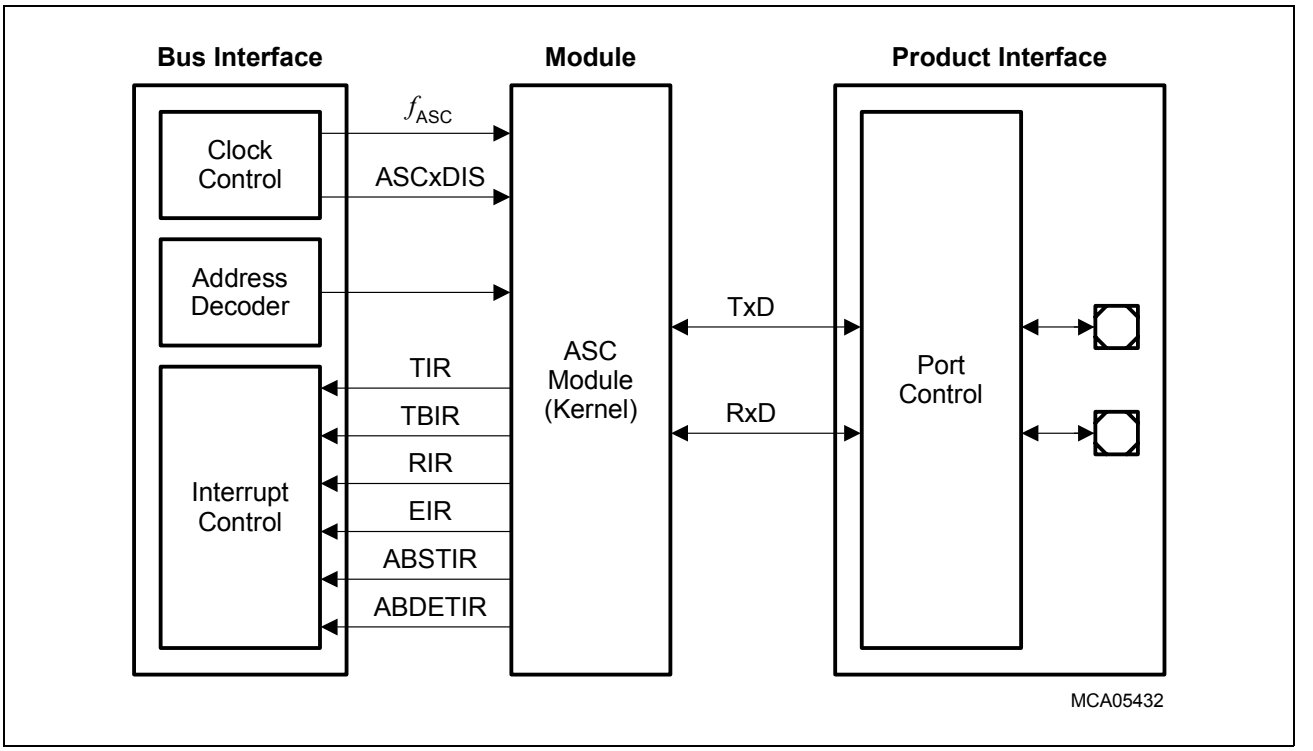


Figure 18-1 ASC Interface Diagram

18.1 Operational Overview

Figure 18-2 shows a block diagram of the ASC with its operating modes (Asynchronous and Synchronous Mode).

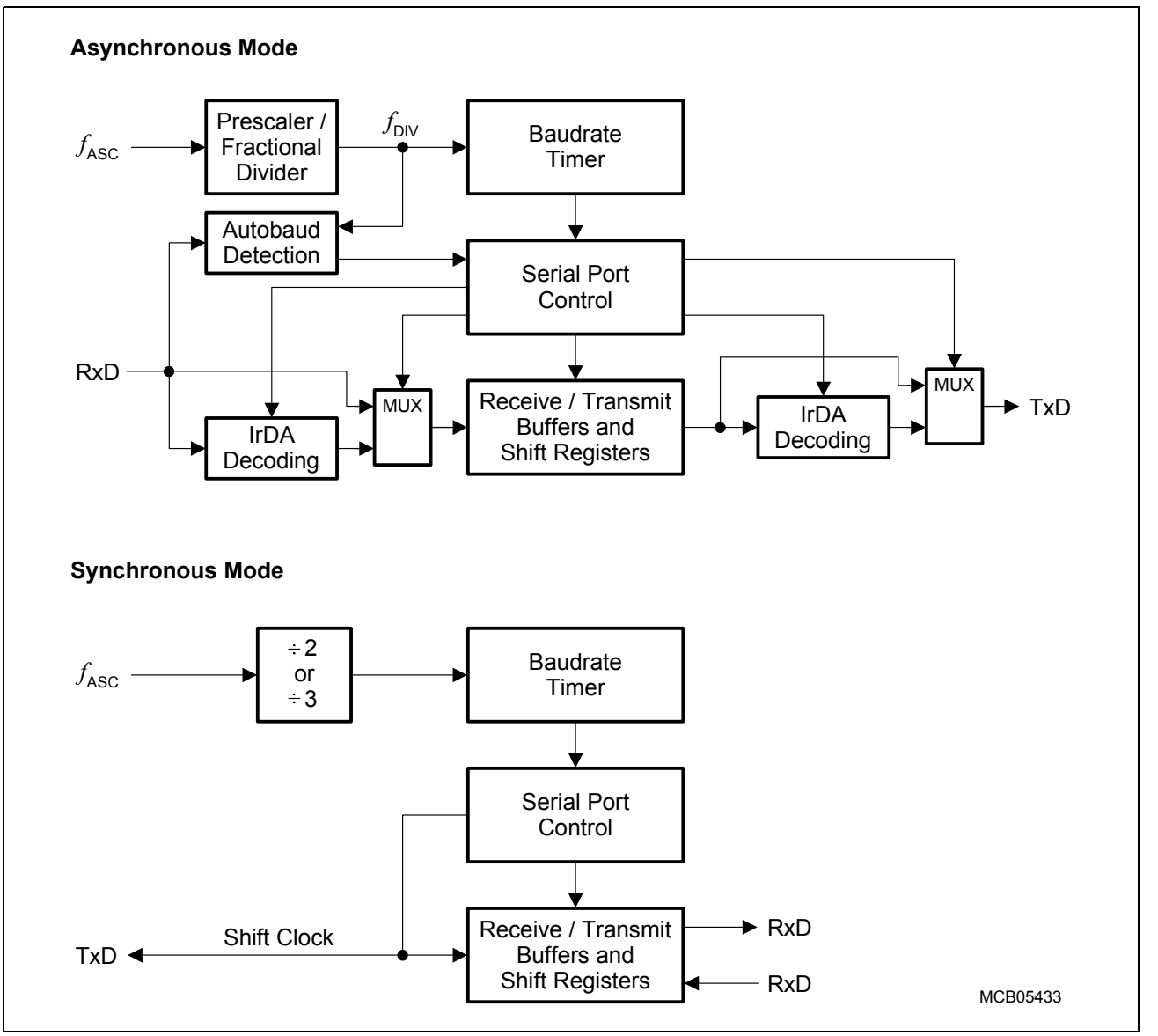


Figure 18-2 Block Diagram of the ASC

The ASC supports full-duplex asynchronous communication with up to 2.5 Mbit/s and half-duplex synchronous communication with up to 5 Mbit/s (@ 40 MHz module clock). In Synchronous Mode, data are transmitted or received synchronous to a shift clock that is generated by the microcontroller. In Asynchronous Mode, either 8- or 9-bit data transfer, parity generation, and the number of stop bits can be selected. Parity, framing, and overrun error detection is provided to increase the reliability of data transfers. Transmission and reception of data is double-buffered. For multiprocessor communication, a mechanism is provided to distinguish address bytes from data bytes.

Testing is supported by a loop-back option. A 13-bit baudrate timer with a versatile input clock divider circuitry provides the serial clock signal. In a Special Asynchronous Mode, the ASC supports IrDA data transmission up to 115.2 kbit/s with fixed or programmable IrDA pulse width. Autobaud Detection allows to detect asynchronous data frames with its baudrate and mode with automatic initialization of the baudrate generator and the mode control bits.

A transmission is started by writing to the transmit buffer register TBUF. The selected operating mode determines the number of data bits that will actually be transmitted, so that, bits written to positions 9 through 15 of register TBUF are always insignificant. Data transmission is double-buffered, so a new character may be written to the transmit buffer register before the transmission of the previous character is complete. This allows the transmission of characters back-to-back without gaps.

Data reception is enabled by the Receiver Enable Bit REN. After reception of a character has been completed, the received data can be read from the (read-only) receive buffer register RBUF; the received parity bit can also be read if provided by the selected operating mode. Bits in the upper half of RBUF that are not valid in the selected operating mode will be read as zeros.
Data reception is double-buffered, so that reception of a second character may already begin before the previously received character has been read out of the receive buffer register. In all modes, receive overrun error detection can be selected through bit OEN. When enabled, the overrun error status flag OE and the error interrupt request line EIR will be activated when the receive buffer register has not been read by the time reception of a ninth character is complete. The previously received character in the receive buffer is overwritten.

The Loopback Mode (selected by bit LB) allows the data currently being transmitted to be received simultaneously in the receive buffer. This may be used to test serial communication routines at an early stage without having to provide an external network.

Note: In Loopback Mode, the alternate input/output functions of the associated port pins are not necessary.

Note: Serial data transmission or reception is only possible when the Baudrate Generator Run bit R is set. Otherwise, the serial interface is idle.

Note: Do not program the Mode Control bitfield M to one of the reserved combinations to avoid unpredictable behavior of the serial interface.

The operating mode of the serial channel ASC is controlled by its control register ASCx_CON. This register contains control bits for mode and error check selection, and status flags for error identification.

18.2 Asynchronous Operation

Asynchronous Mode supports full-duplex communication in which both transmitter and receiver use the same data frame format and the same baudrate. Data is transmitted on line TxD and received on line RxD. IrDA data transmission/reception is supported up to 115.2 kbit/s. **Figure 18-3** shows the block diagram of the ASC when operating in Asynchronous Mode.

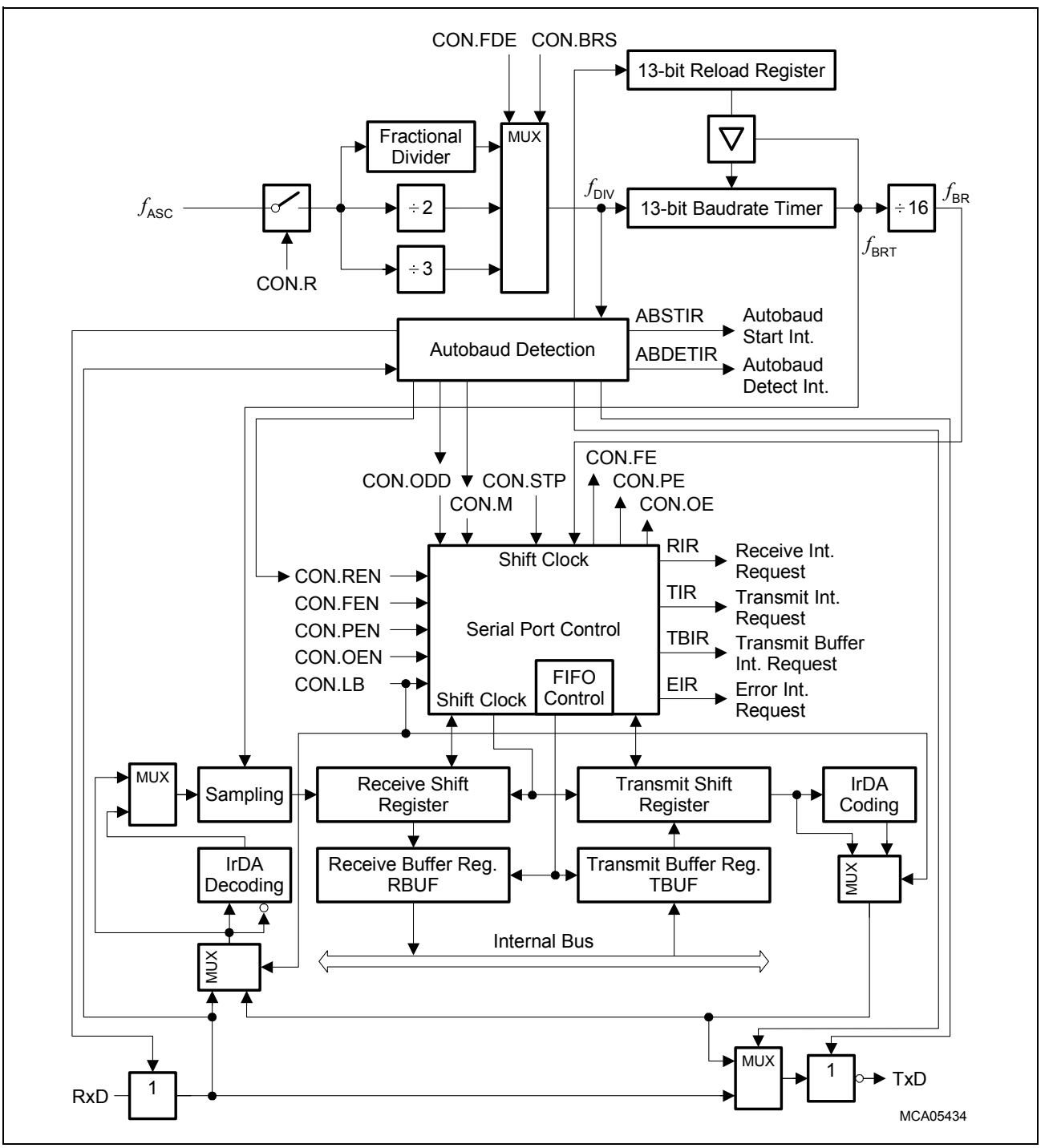


Figure 18-3 Asynchronous Mode of Serial Channel ASC

18.2.1 Asynchronous Data Frames

8-Bit Data Frames

8-bit data frames consist of either eight data bits D7 … D0 (M = 001_B), or seven data bits D6 … D0 plus an automatically generated parity bit (M = 011_B). Parity may be odd or even, depending on bit ODD. An even parity bit will be set if the modulo-2-sum of the 7 data bits is 1. An odd parity bit will be cleared in this case. Parity checking is enabled via bit PEN (always OFF in 8-bit data mode). The parity error flag PE will be set, along with the error interrupt request flag, if a wrong parity bit is received. The parity bit itself will be stored in bit RBUF.7.

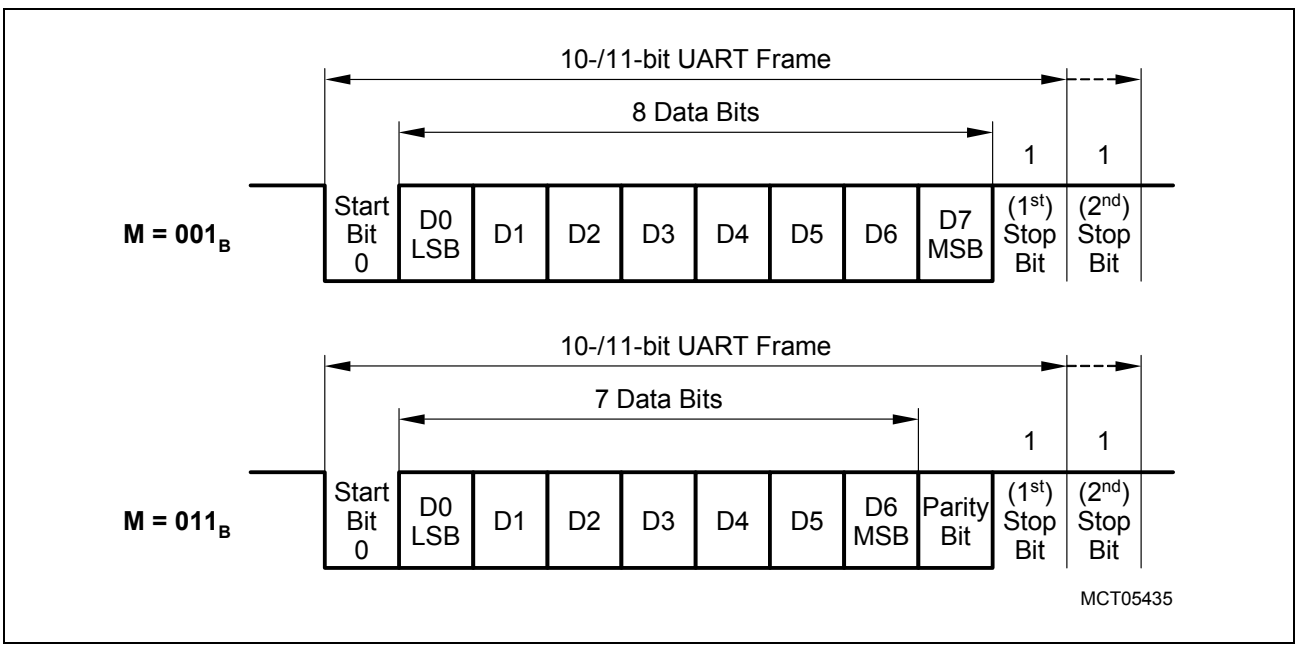


Figure 18-4 Asynchronous 8-Bit Frames

9-Bit Data Frames

9-bit data frames consist of either nine data bits D8 … D0 (M = 100_B), eight data bits D7 … D0 plus an automatically generated parity bit (M = 111_B), or eight data bits D7 … D0 plus wake-up bit (M = 101_B). Parity may be odd or even, depending on bit ODD. An even parity bit will be set if the modulo-2-sum of the 8 data bits is 1. An odd parity bit will be cleared in this case. Parity checking is enabled via bit PEN (always OFF in 9-bit data and wake-up mode). The parity error flag PE will be set, along with the error interrupt request flag, if a wrong parity bit is received. The parity bit itself will be stored in bit RBUF.8.

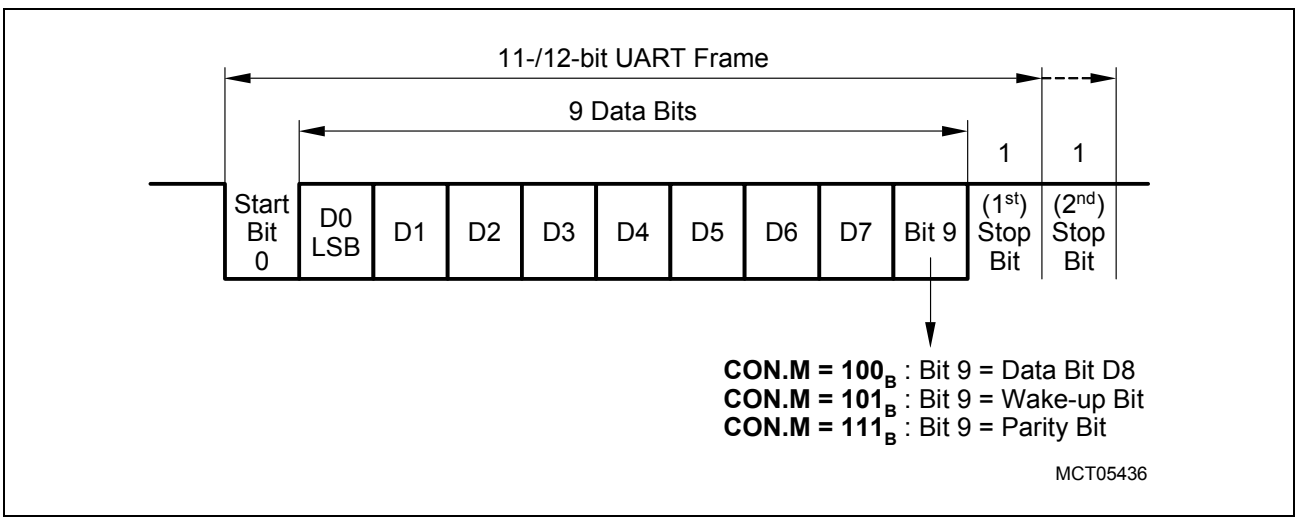


Figure 18-5 Asynchronous 9-Bit Frames

In wake-up mode, received frames are transferred to the receive buffer register only if the 9th bit (the wake-up bit) is 1. If this bit is 0, no receive interrupt request will be activated and no data will be transferred.

This feature may be used to control communication in a multi-processor system:
When the master processor wants to transmit a block of data to one of several slaves, it first sends out an address byte to identify the target slave. An address byte differs from a data byte in that the additional 9th bit is a 1 for an address byte, but is a 0 for a data byte; so, no slave will be interrupted by a data ‘byte’. An address ‘byte’ will interrupt all slaves (operating in 8-bit data + wake-up bit mode), so each slave can examine the eight LSBs of the received character (the address). The addressed slave will switch to 9-bit data mode (such as by clearing bit M[0]), to enable it to also receive the data bytes that will be coming (having the wake-up bit cleared). The slaves not being addressed remain in 8-bit data + wake-up bit mode, ignoring the following data bytes.

IrDA Frames

The modulation schemes of IrDA are based on standard asynchronous data transmission frames. The asynchronous data format in IrDA Mode (M = 010_B) is defined as follows:

1 start bit/8 data bits/1 stop bit

The coding/decoding of/to the asynchronous data frames is shown in **Figure 18-6**. In general, during IrDA transmissions, UART frames are encoded into IR frames and vice versa. A low level on the IR frame indicates an “LED off” state. A high level on the IR frame indicates an “LED on” state.

For a 0-bit in the UART frame, a high pulse is generated. For a 1-bit in the UART frame, no pulse is generated. The high pulse starts in the middle of a bit cell and has a fixed width of 3/16 of the bit time. The ASC also allows the length of the IrDA high pulse to be programmed. Further, the polarity of the received IrDA pulse can be inverted in IrDA Mode. **Figure 18-6** shows the non-inverted IrDA pulse scheme.

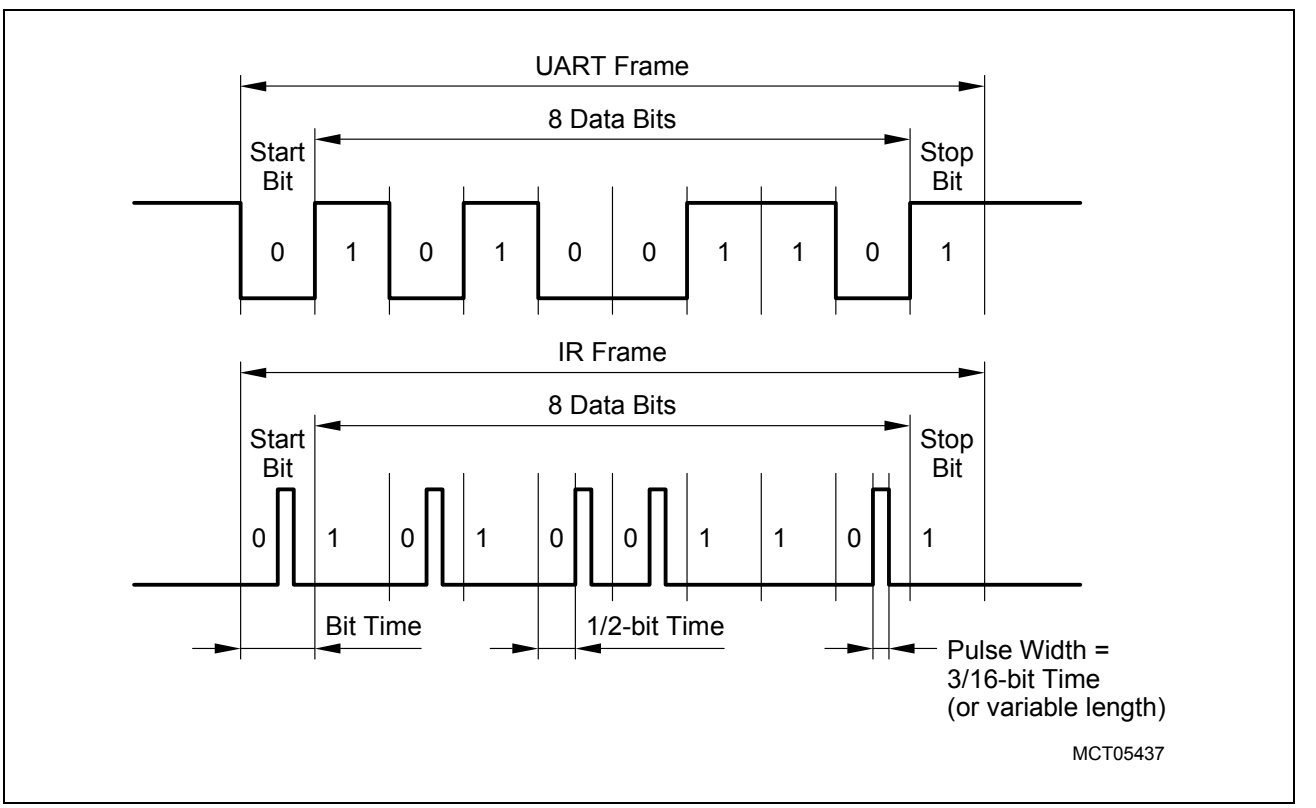


Figure 18-6 IrDA Frame Encoding/Decoding

The ASC IrDA pulse mode/width register PMW contains the 8-bit IrDA pulse width value and the IrDA pulse width mode select bit. This register is required in the IrDA operating mode only.

18.2.2 Asynchronous Transmission

Asynchronous transmission begins at the next overflow of the divide-by-16 baudrate timer (transition of the baudrate clock f_{BR}), if bit R is set and data has been loaded into TBUF. The transmitted data frame consists of three basic elements:

- Start bit
- Data field (eight or nine bits, LSB first, including a parity bit, if selected)
- Delimiter (one or two stop bits)

Data transmission is double-buffered. When the transmitter is idle, the transmit data loaded in the transmit buffer register is immediately moved to the transmit shift register, thus freeing the transmit buffer for the next data to be sent. This is indicated by the transmit buffer interrupt request line TBIR being activated. TBUF may now be loaded with the next data, while transmission of the previous data continues.

The transmit interrupt request line TIR will be activated before the last bit of a frame is transmitted, that is, before the first or the second stop bit is shifted out of the transmit shift register.

Note: The transmitter output pin TxD must be configured for alternate data output.

18.2.3 Transmit FIFO Operation

The transmit FIFO (TXFIFO) provides the following functionality:

- Enable/disable control
- Programmable filling level for transmit interrupt generation
- Filling level indication
- FIFO clear (flush) operation
- FIFO overflow error generation

The 8-stage transmit FIFO is controlled by the TXFCON control register. When bit TXFEN is set, the transmit FIFO is enabled. The interrupt trigger level defined by TXFITL defines the filling level of the TXFIFO at which a transmit buffer interrupt TBIR or a transmit interrupt TIR is generated. These interrupts are always generated when the filling level of the transmit FIFO is equal to or less than the value stored in TXFITL.

Bitfield TXFFL in the FIFO status register ASCx_FSTAT indicates the number of entries that are actually written (valid) in the TXFIFO. Therefore, the software can verify, in the interrupt service routine, for instance, how many bytes can still be written into the transmit FIFO via register TBUF without getting an overrun error.

The transmit FIFO cannot be accessed directly. All data write operations into the TXFIFO are executed by writing into the TBUF register.

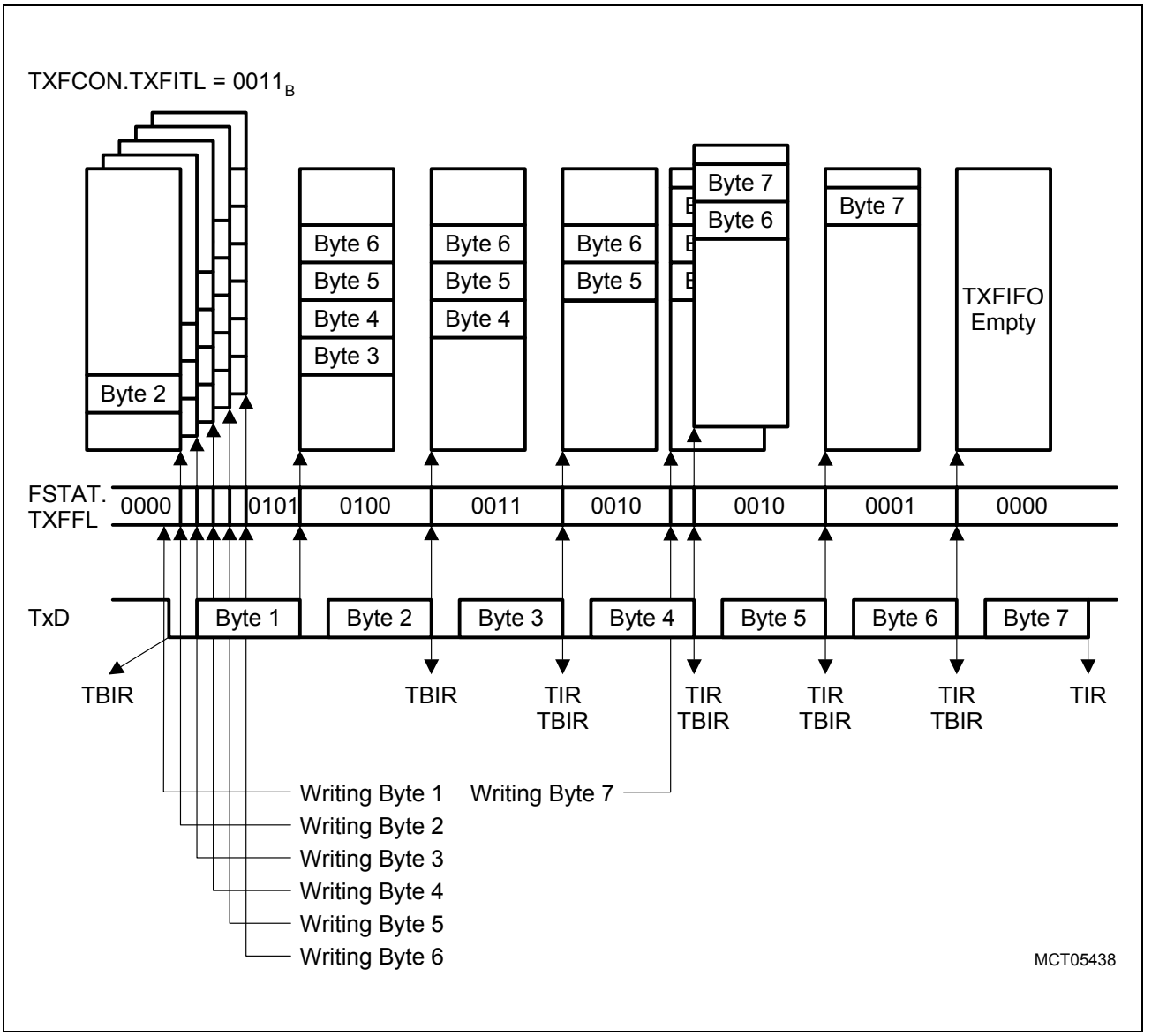


Figure 18-7 Transmit FIFO Operation Example

The example in **Figure 18-7** shows a typical 8-stage transmit FIFO operation. In this example seven bytes are transmitted via the TxD output line. The transmit FIFO interrupt trigger level TXFITL is set to 0011_B. The first byte written into the empty TXFIFO via TBUF is directly transferred into the transmit shift register and is not written into the FIFO. A transmit buffer interrupt will be generated in this case. After byte 1, bytes 2 to 6 are written into the transmit FIFO.

After the transfer of byte 3 from the TXFIFO into the transmit shift register of the ASC, 3 bytes remain in the TXFIFO. Therefore, the value of TXFITL is reached and a transmit buffer interrupt will be generated at the beginning and a transmit interrupt at the end of the byte 3 serial transmission. During the serial transmission of byte 4, another byte (byte 7) is written into the TXFIFO (TBUF write operation). Finally, after the start of the serial transmission of byte 7, the TXFIFO is again empty.

If the TXFIFO is full and additional bytes are written into TBUF, the error interrupt will be generated with bit OE set. In this case, the data byte that was last written into the transmit FIFO is overwritten and the transmit FIFO filling level TXFFL is set to maximum.

The TXFIFO can be flushed or cleared by setting bit TXFFLU in register ASCx_TXFCON. After this TXFIFO flush operation, the TXFIFO is empty and the transmit FIFO filling level TXFFL is set to 0000_B. A running serial transmission is not aborted by a receive FIFO flush operation

Note: The TXFIFO is flushed automatically with a reset operation of the ASC module and if the TXFIFO becomes disabled (resetting bit TXFEN) after it was previously enabled.

18.2.4 Asynchronous Reception

Asynchronous reception is initiated by a falling edge (1-to-0 transition) on line RxD, provided that bits R and REN are set. The receive data input line RxD is sampled at 16 times the rate of the selected baudrate. A majority decision of the 7th, 8th, and 9th sample determines the effective bit value. This avoids erroneous results that may be caused by noise.

If the detected value is not a 0 when the start bit is sampled, the receive circuit is reset and waits for the next 1-to-0 transition at line RxD. If the start bit proves valid, the receive circuit continues sampling and shifts the incoming data frame into the receive shift register.

When the last stop bit has been received, the content of the receive shift register are transferred to the receive data buffer register RBUF. Simultaneously, the receive interrupt request line RIR is activated after the 9th sample in the last stop bit time slot (as programmed), regardless of whether valid stop bits have been received or not. The receive circuit then waits for the next start bit (1-to-0 transition) at the receive data input line.

Note: The receiver input pin RxD must be configured for input.

Asynchronous reception is stopped by clearing bit REN. A currently received frame is completed including the generation of the receive interrupt request and an error interrupt request, if appropriate. Start bits that follow this frame will not be recognized.

Note: In wake-up mode, received frames are transferred to the receive buffer register only if the 9th bit (the wake-up bit) is 1. If this bit is 0, no receive interrupt request will be activated and no data will be transferred.

18.2.5 Receive FIFO Operation

The receive FIFO (RXFIFO) provides the following functionality:

- Enable/disable control
- Programmable filling level for receive interrupt generation
- Filling level indication
- FIFO clear (flush) operation
- FIFO overflow error generation

The 8-stage receive FIFO is controlled by the RXFCON control register. When bit RXFEN is set, the receive FIFO is enabled. The interrupt trigger level defined by RXFITL defines the filling level of RXFIFO at which a receive interrupt RIR is generated. RIR is always generated when the filling level of the receive FIFO is equal to or greater than the value stored in RXFITL.

Bitfield RXFFL in the FIFO status register ASCx_FSTAT indicates the number of bytes that have been actually written into the FIFO and can be read out of the FIFO by a user program.

The receive FIFO cannot be accessed directly. All data read operations from the RXFIFO are executed by reading the RBUF register.

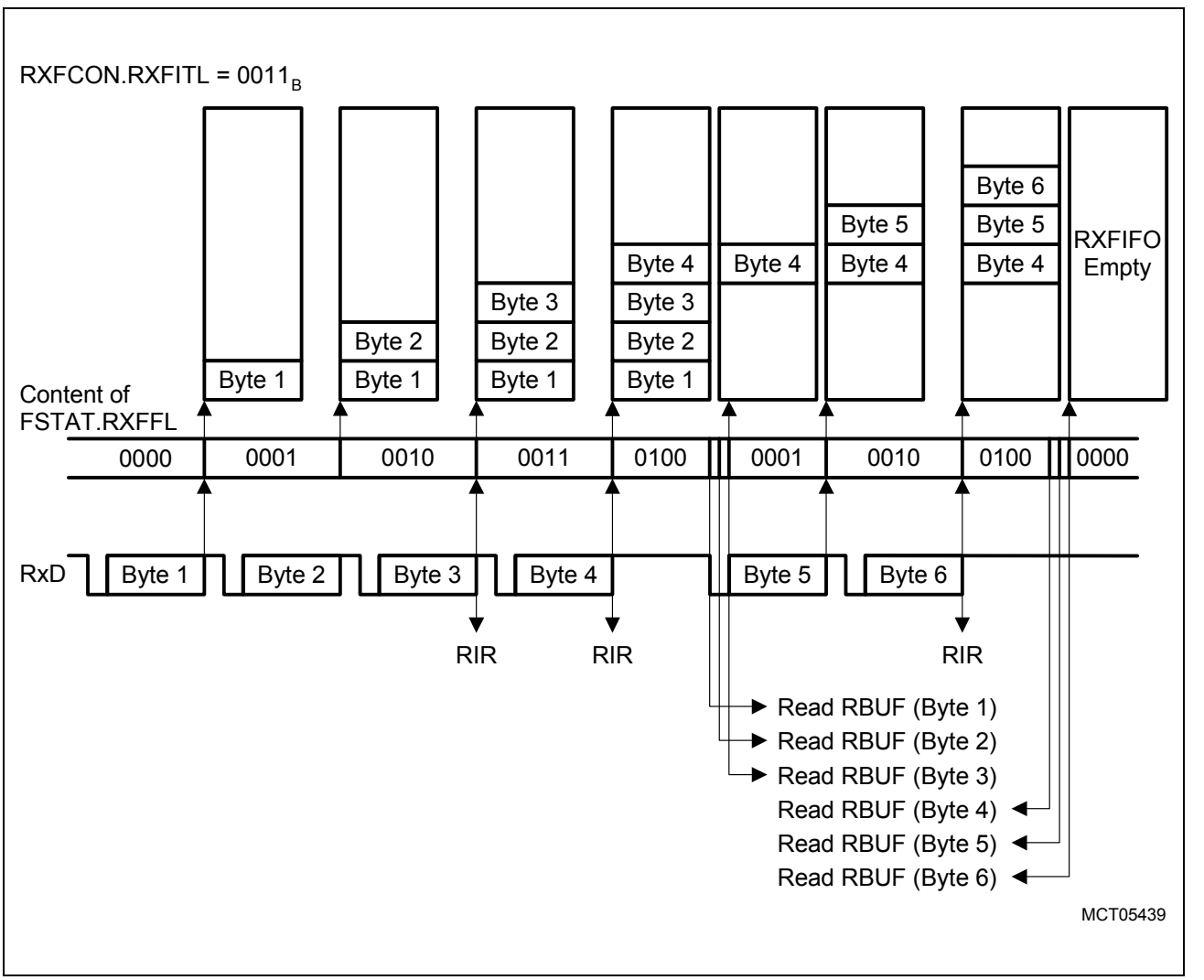


Figure 18-8 Receive FIFO Operation Example

The example in **Figure 18-8** shows a typical 8-stage receive FIFO operation. In this example, six bytes are received via the RxD input line. The receive FIFO interrupt trigger level RXFITL is set to 0011_B. Therefore, the first receive interrupt RIR is generated after the reception of byte 3 (RXFIFO is filled with three bytes).

After the reception of byte 4, three bytes are read out of the receive FIFO. After this read operation, the RXFIFO still contains one byte. RIR becomes again active after two more bytes (byte 5 and 6) have been received (RXFIFO filled again with 3 bytes). Finally, the FIFO is cleared after three read operation.

If the RXFIFO is full and additional bytes are received, the receive interrupt RIR and the error interrupt EIR will be generated with bit OE set. In this case, the data byte last written into the receive FIFO is overwritten. With the overrun condition, the receive FIFO filling level RXFFL is set to maximum. If a RBUF read operation is executed with the RXFIFO

enabled but empty, an error interrupt EIR will be generated as well with bit OE set. In this case, the receive FIFO filling level RXFFL is set to 0000_B.

If the RXFIFO is available but disabled (RXFEN = 0) and the receive operation is enabled (REN = 1), the asynchronous receive operation is functionally equivalent to the asynchronous receive operation of the ASC module.

The RXFIFO can be flushed or cleared by setting bit RXFFLU in register RXFCON. After this RXFIFO flush operation, the RXFIFO is empty and the receive FIFO filling level RXFFL is set to 0000_B.

The RXFIFO is flushed automatically with a reset operation of the ASC module and if the RXFIFO becomes disabled (resetting bit RXFEN) after it was previously enabled. Resetting bit REN without resetting RXFEN does not affect (reset) the RXFIFO state. This means that the receive operation of the ASC is stopped, in this case, without changing the content of the RXFIFO. After setting REN again, the RXFIFO with its content is again available.

Note: After a successful autobaud detection sequence (if implemented), the RXFIFO should be flushed before data is received.

18.2.6 FIFO Transparent Mode

In Transparent Mode, a specific interrupt generation mechanism is used for receive and transmit buffer interrupts. In general, in Transparent Mode, receive interrupts are always generated if data bytes are available in the RXFIFO. Transmit buffer interrupts are always generated if the TXFIFO is not full. The relevant conditions for interrupt generation in Transparent Mode are:

- FIFO filling levels
- Read/write operations on the RBUF/TBUF data register

Interrupt generation for the receive FIFO depends on the RXFIFO filling level and the execution of read operations of register RBUF (see **Figure 18-9**). Transparent Mode for the RXFIFO is enabled when bits RXTMEN and RXFEN in register ASCx_RXFCON are set.

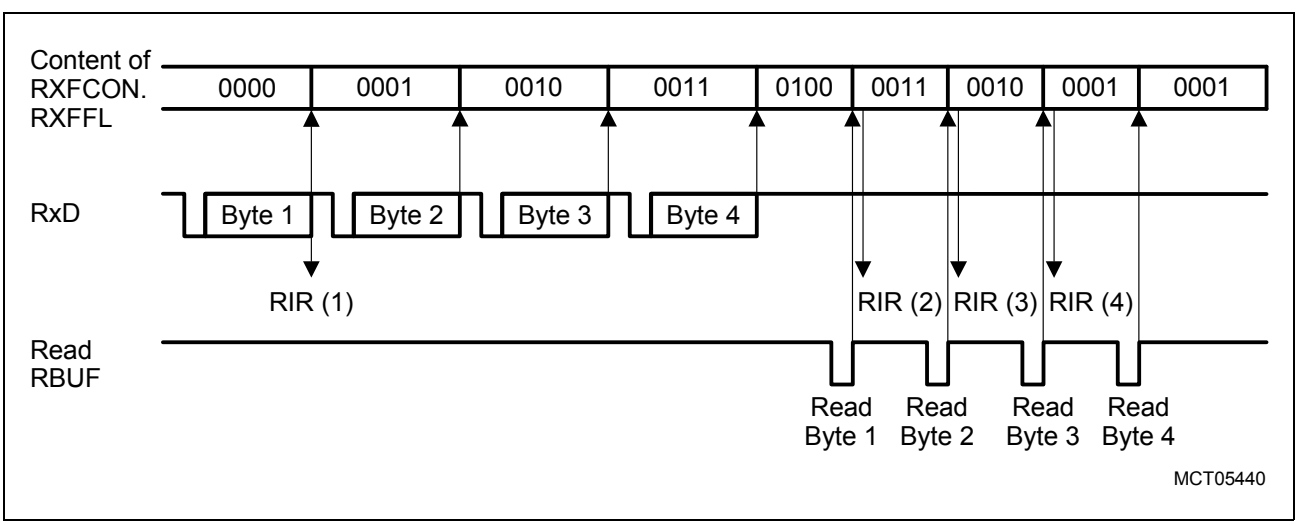


Figure 18-9 Transparent Mode Receive FIFO Operation

If the RXFIFO is empty, a receive interrupt RIR is always generated when the first byte is written into an empty RXFIFO (RXFFL changes from 0000_B to 0001_B). If the RXFIFO is filled with at least one byte, the occurrence of further receive interrupts depends on the read operations of register RBUF. The receive interrupt RIR will always be activated after a RBUF read operation if the RXFIFO still contains data (RXFFL is not equal to 0000_B). If the RXFIFO is empty after a RBUF read operation, no further receive interrupt will be generated.

If the RXFIFO is full (RXFFL = maximum) and additional bytes are received, an error interrupt EIR will be generated with bit OE set. In this case, the data byte last written into the receive FIFO is overwritten. If a RBUF read operation is executed with the RXFIFO enabled but empty (underflow condition), an error interrupt EIR will be generated as well, with bit OE set.

If the RXFIFO is flushed in Transparent Mode, the software must take care that a previous pending receive interrupt is ignored.

Note: The Receive FIFO Interrupt Trigger Level bitfield RXFITL is a don't care in Transparent Mode.

Interrupt generation for the transmit FIFO depends on the TXFIFO filling level and the execution of write operations to the register TBUF. Transparent Mode for the TXFIFO is enabled when bits TXTMEN and TXFEN are set.

A transmit buffer interrupt TBIR is always generated when the TXFIFO is not full (TXFFL not equal to maximum) after a byte has been written into register ASCx_TBUF. TBIR is also activated after a TXFIFO flush operation or when the TXFIFO becomes enabled (TXTMEN and TXFEN set) when it was previously disabled. In these cases, the TXFIFO is empty and ready to be filled with data.

If the TXFIFO is full (TXFFL = maximum) and an additional byte is written into TBUF, no further transmit buffer interrupt will be generated after the TBUF write operation. In this case the data byte last written into the transmit FIFO is overwritten and an overrun error interrupt (EIR) will be generated with bit OE set.

Note: The Transmit FIFO Interrupt Trigger Level bitfield TXFITL is a don't care in Transparent Mode.

18.2.7 IrDA Mode

The duration of the IrDA pulse is normally 3/16 of a bit period. The IrDA standard also allows the pulse duration to be independent of the baudrate or bit period. In this case, the width of the transmitted pulse always corresponds to the 3/16 pulse width at 115.2 kbit/s, which is 1.627 µs. Either fixed or bit-period-dependent IrDA pulse width generation can be selected. The IrDA pulse width mode is selected by bit IRPW.

In case of fixed IrDA pulse width generation, the lower eight bits in register PMW are used to adapt the IrDA pulse width to a fixed value such as 1.627 µs. The fixed IrDA pulse width is generated by a programmable timer as shown in **Figure 18-10**.

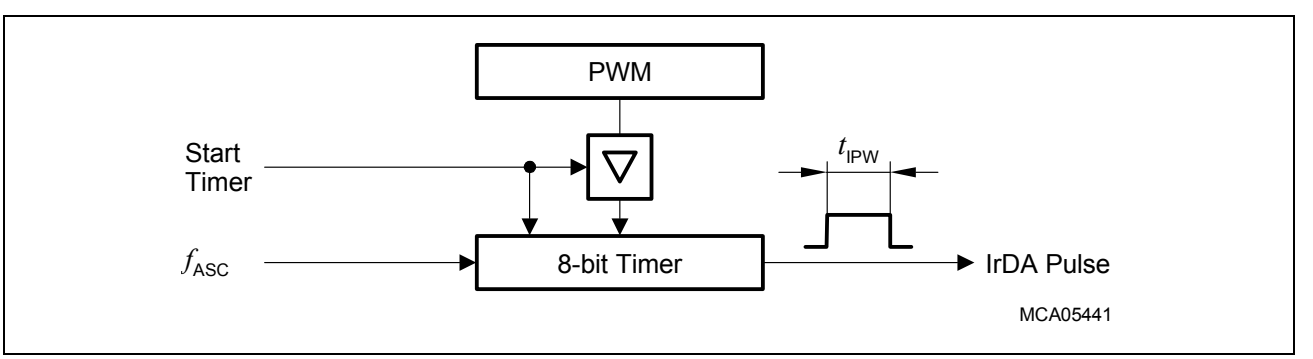


Figure 18-10 Fixed IrDA Pulse Generation

The IrDA pulse width can be calculated according the formulas given in **Table 18-1**.

*Note: The name PMW in the formulas of **Table 18-1** represents the contents of the pulse mode/width register PMW (PW_VALUE), taken as an unsigned 8-bit integer.*

Table 18-1 Formulas for IrDA Pulse Width Calculation

PMW	PMW_IPMW	Formulas	
1 … 255	0	$t_{IPW} = \frac{3}{16 \times Baudrate}$	$t_{IPW\ min} = \frac{(PMW >> 1)}{f_{ASC}}$
	1	$t_{IPW} = \frac{PMW}{f_{ASC}}$	

The contents of PW_VALUE further define the minimum IrDA pulse width ($t_{IPW\ min}$) that is still recognized as a valid IrDA pulse during a receive operation. This function is independent of the selected IrDA pulse width mode (fixed or variable) which is defined by bit IRPW. The minimum IrDA pulse width is calculated by a shift right operation of PMW bit 7-0 by one bit divided by the module clock f_{ASC}.

Note: If IRPW is cleared (fixed IrDA pulse width), PW_VALUE must be a value which assures that $t_{IPW} > t_{IPW\ min}$.

Table 18-2 gives three examples for typical frequencies of f_{ASC}.

Table 18-2 IrDA Pulse Width Adaption to 1.627 µs

f_{ASC}	PMW	t_{IPW}	Error	$t_{IPW\ min}$
20 MHz	33	1.65 µs	-1.1%	0.8 µs
50 MHz	81	1.62 µs	-1.0%	0.8 µs
66 MHz	107	1.621 µs	-1.0%	0.81 µs

18.2.8 RxD/TxD Data Path Selection in Asynchronous Modes

The data paths for the serial input and output data in Asynchronous Mode are affected by several control bits in the registers CON and ABCON as shown in **Figure 18-11**. The Synchronous Mode operation is not affected by these data path selection capabilities.

The input signal from RxD passes an inverter which is controlled by bit RXINV. The output signal of this inverter is used for the Autobaud Detection and may bypass the logic in the Echo Mode (controlled by bit ABEM). Further, two multiplexers are in the RxD input signal path for providing the Loopback Mode capability (controlled by bit LB) and the IrDA receive pulse inversion capability (controlled by bit RxDI).

Depending on the Asynchronous Mode (controlled by bitfield M), output signal or the RxD input signal in Echo Mode (controlled by bit ABEM) is switched to the TxD output via an inverter (controlled by bit TXINV).

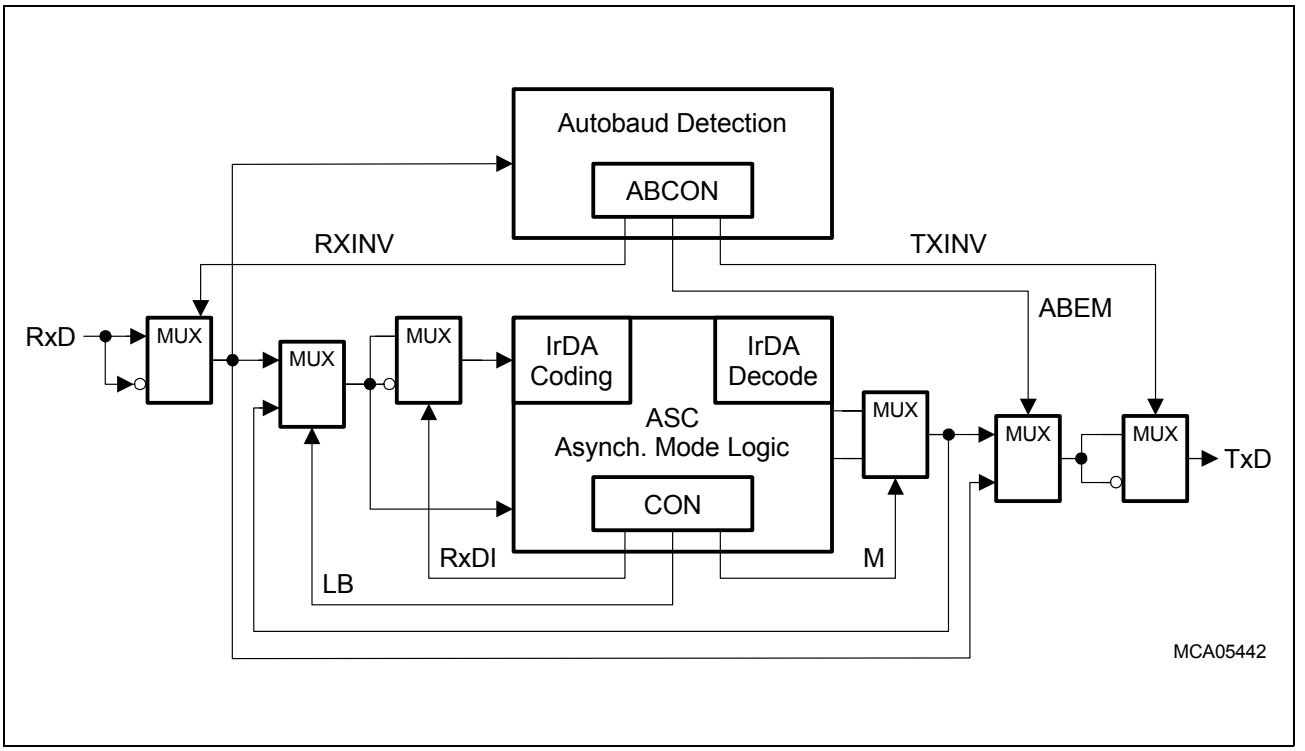


Figure 18-11 RxD/TxD Data Path in Asynchronous Modes

*Note: In Echo Mode the transmit output signal is blocked by the Echo Mode output multiplexer. **Figure 18-11** shows that it is not possible to use an IrDA coded receiver input signal for Autobaud Detection.*

18.3 Synchronous Operation

Synchronous Mode supports half-duplex communication, basically for simple I/O expansion via shift registers. Data is transmitted and received via line RxD while line TxD outputs the shift clock.

Synchronous Mode is selected with bitfield M = 000_B.

Eight data bits are transmitted or received synchronous to a shift clock generated by the internal baudrate generator. The shift clock is active only as long as data bits are transmitted or received.

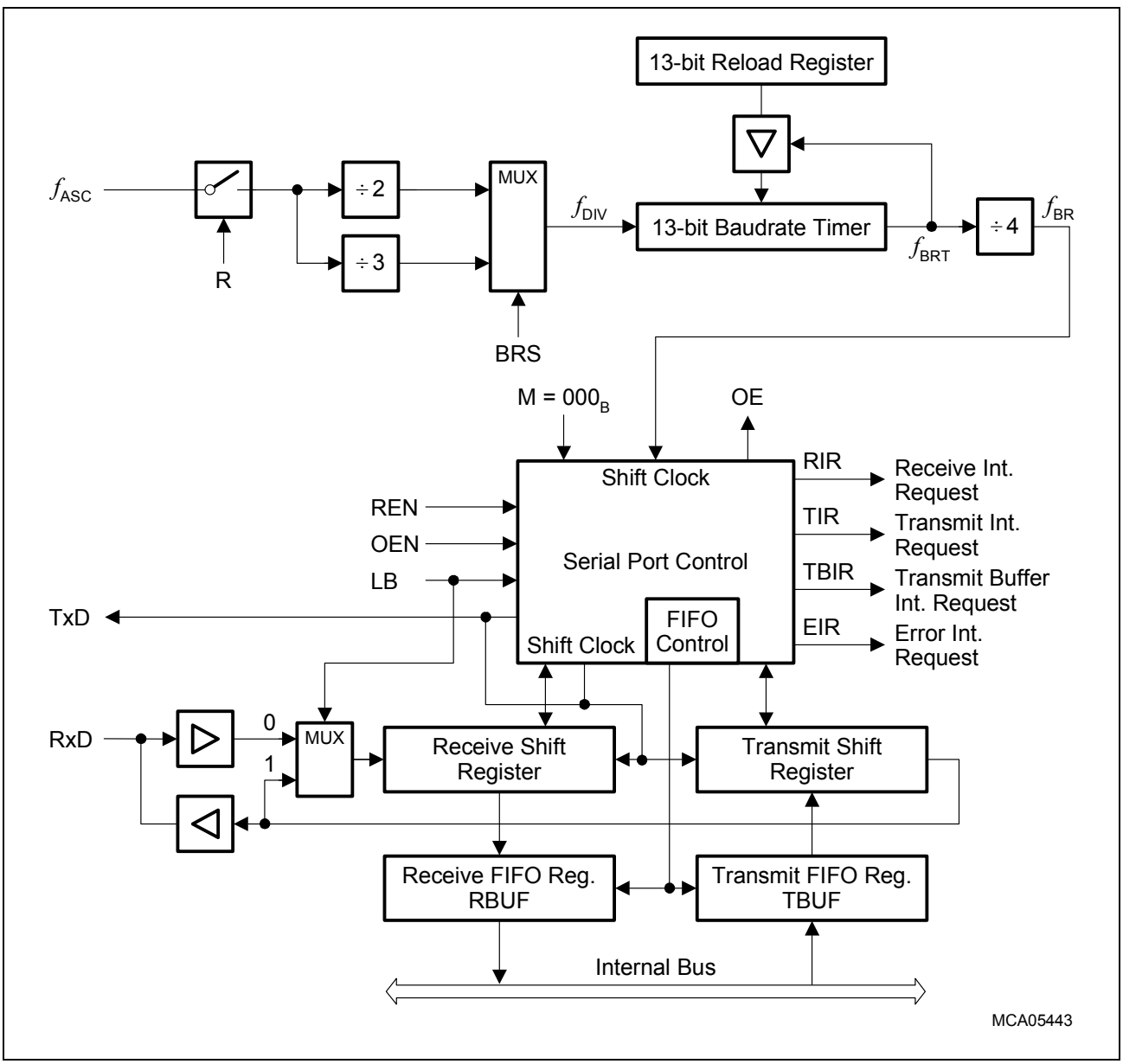


Figure 18-12 Synchronous Mode of Serial Channel ASC

18.3.1 Synchronous Transmission

Synchronous transmission begins within four state times after data has been loaded into TBUF, provided that bit R is set and bit REN is cleared (half-duplex, no reception). Exception: in Loopback Mode (bit LB set), REN must be set for reception of the transmitted byte. Data transmission is double-buffered. When the transmitter is idle, the transmit data loaded into TBUF is immediately moved to the transmit shift register, thus freeing TBUF for more data. This is indicated by the transmit buffer interrupt request line TBIR being activated. TBUF may now be loaded with the next data, while transmission of the previous continuous. The data bits are transmitted synchronous with the shift clock. After the bit time for the eighth data bit, both the TxD and RxD lines will go high, the transmit interrupt request line TIR is activated, and serial data transmission stops.

Note: Pin TxD must be configured for alternate data output in order to provide the shift clock. Pin RxD must also be configured for output during transmission.

18.3.2 Synchronous Reception

Synchronous reception is initiated by setting bit REN. If bit R is set, the data applied at RxD is clocked into the receive shift register synchronous to the clock that is output at TxD. After the eighth bit has been shifted in, the contents of the receive shift register are transferred to the receive data buffer RBUF, the receive interrupt request line RIR is activated, the receiver enable bit REN is reset, and serial data reception stops.

Note: Pin TxD must be configured for alternate data output in order to provide the shift clock. Pin RxD must be configured as alternate data input.

Synchronous reception is stopped by clearing bit REN. A currently received byte is completed, including the generation of the receive interrupt request and an error interrupt request, if appropriate. Writing to the transmit buffer register while a reception is in progress has no effect on reception and will not start a transmission.

If a previously received byte has not been read out of a full receive buffer at the time the reception of the next byte is complete, both the error interrupt request line EIR and the overrun error status flag OE will be activated/set, provided the overrun check has been enabled by bit OEN.

18.3.3 Synchronous Timing

Figure 18-13 shows timing diagrams of the ASC Synchronous Mode data reception and data transmission. In idle state, the shift clock level is high. With the beginning of a synchronous transmission of a data byte, the data is shifted out at RxD with the falling edge of the shift clock. If a data byte is received through RxD, data is latched with the rising edge of the shift clock.

Between two consecutive receive or transmit data bytes, one shift clock cycle (f_{BR}) delay is inserted.

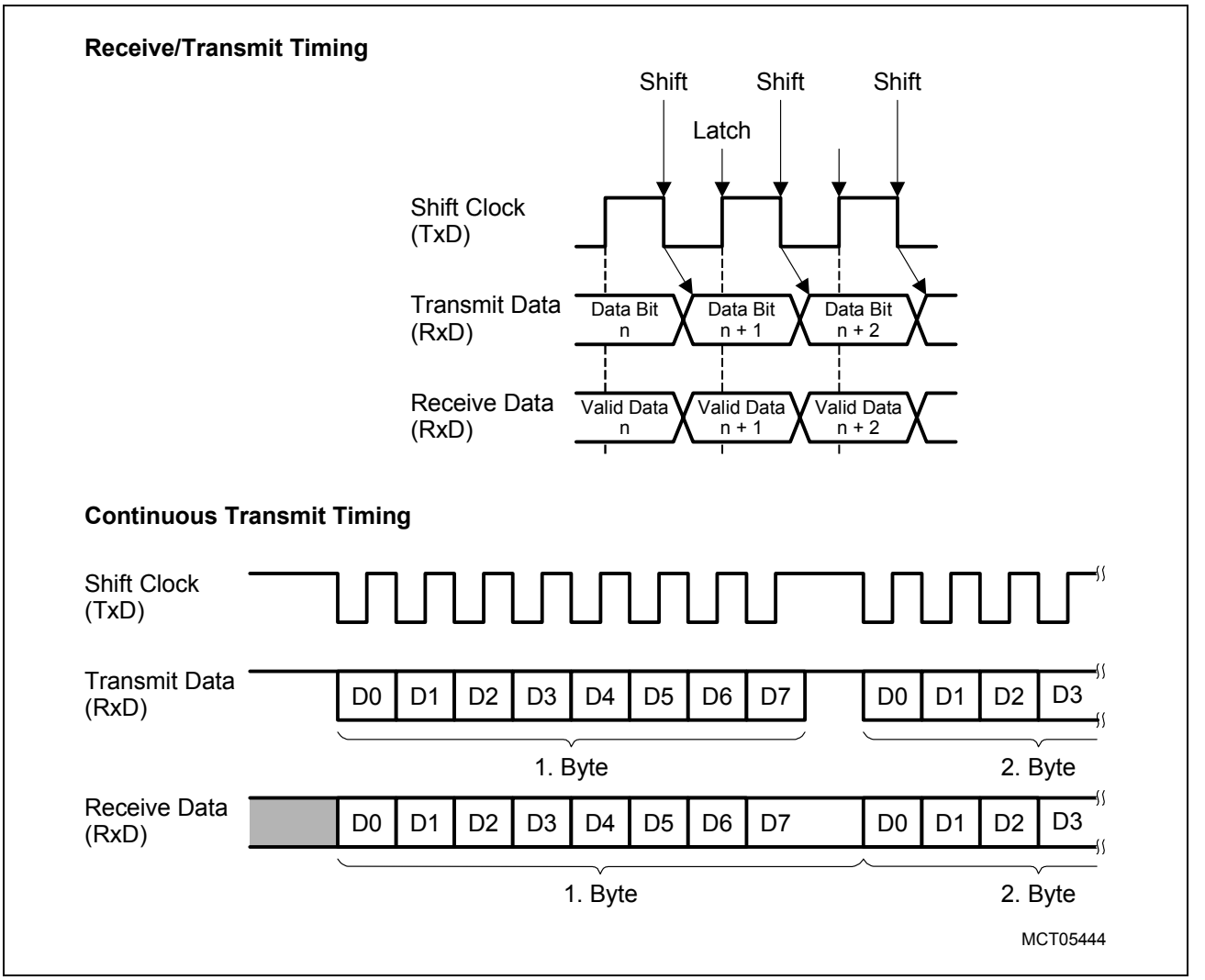


Figure 18-13 ASC Synchronous Mode Waveforms

18.4 Baudrate Generation

The serial channel ASC has its own dedicated 13-bit baudrate generator with reload capability, allowing baudrate generation independent of other timers.

The baudrate generator is clocked with a clock (f_{DIV}) derived via a prescaler from the ASC input clock f_{ASC}. The baudrate timer counts downwards and can be started or stopped through the baudrate generator run bit R. Each underflow of the timer provides one clock pulse to the serial channel. The timer is reloaded with the value stored in its 13-bit reload register each time it underflow. The resulting clock f_{BRT} is again divided by a factor for the baudrate clock (16 in Asynchronous Modes and 4 in Synchronous Mode). The prescaler is selected by the bits BRS and FDE. In addition to the two fixed dividers, a fractional divider prescaler unit is available in the Asynchronous Modes that allows selection of prescaler divider ratios of n/512 with n = 0 … 511. Therefore, the baudrate of ASC is determined by the module clock, the content of FDV, the reload value of BG, and the operating mode (asynchronous or synchronous).

Register ASCx_BG is the dual-function Baudrate Generator/Reload register. Reading ASCx_BG returns the contents of the timer BR_VALUE (bits 15 … 13 return zero), while writing to BG always updates the reload register (bits 15 … 13 are insignificant).

An autoreload of the timer with the contents of the reload register is performed each time ASCx_BG is written to. However, if bit R is cleared at the time a write operation to ASCx_BG is performed, the timer will not be reloaded until the first instruction cycle after bit R was set. For a clean baudrate initialization, ASCx_BG should be written only if R = 0. If ASCx_BG is written while R = 1, unpredictable behavior of the ASC may occur during running transmit or receive operations.

The ASC baudrate timer reload register ASCx_BG contains the 13-bit reload value for the baudrate timer in Asynchronous and Synchronous modes.

18.4.1 Baudrate in Asynchronous Mode

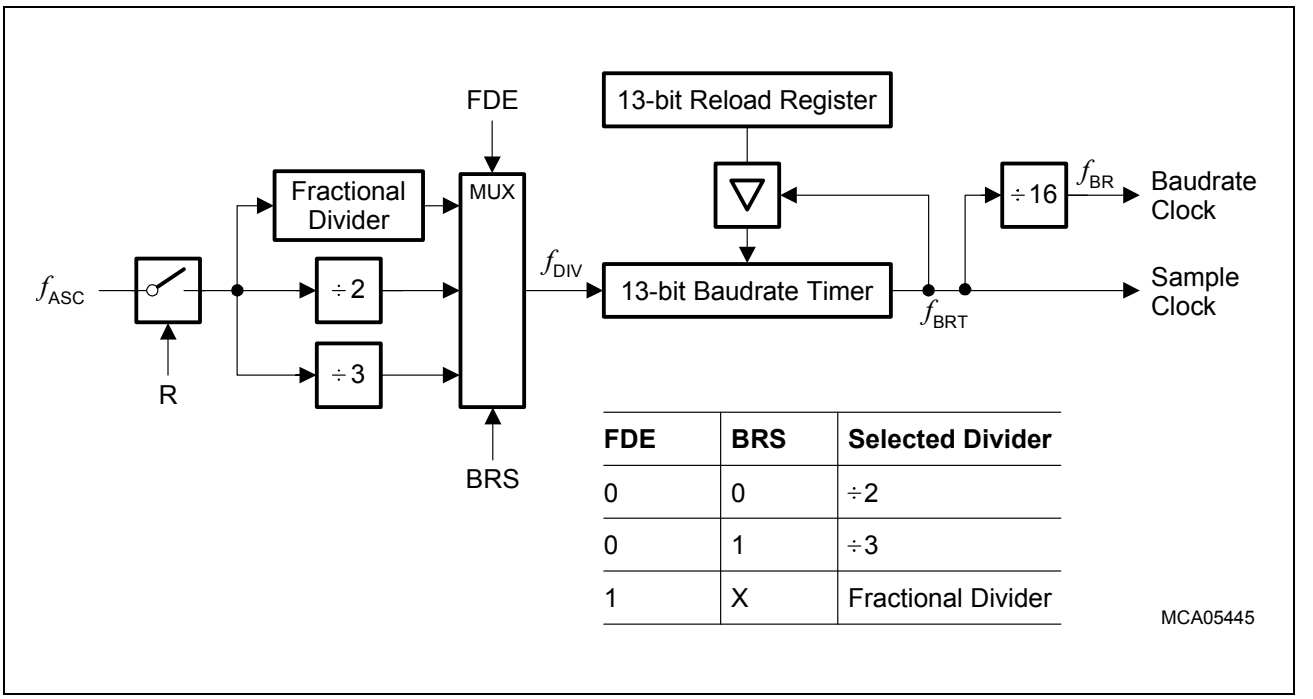
For Asynchronous Mode, the baudrate generator provides a clock f_{BRT} with sixteen times the rate of the established baudrate. Every received bit is sampled at the 7^{th}, 8^{th}, and 9^{th} cycle of this clock. The clock divider circuitry, which generates the input clock for the 13-bit baudrate timer, is extended by a fractional divider circuitry that allows adjustment for more accurate baudrate and the extension of the baudrate range.

The baudrate of the baudrate generator depends on the following bits and register values:

- Input clock f_{ASC}
- Selection of the baudrate timer input clock f_{DIV} by bits FDE and BRS
- If bit FDE is set (fractional divider): value of register ASCx_FDV
- Value of the 13-bit reload register ASCx_BG

The output clock of the baudrate timer with the reload register is the sample clock in the Asynchronous Modes of the ASC. For baudrate calculations, this baudrate clock f_{BR} is derived from the sample clock f_{DIV} by a division by 16.

The ASC fractional divider register ASCx_FDV contains the 9-bit divider value for the fractional divider (Asynchronous Mode only). It is also used for reference clock generation of the autobaud detection unit.



FDE	BRS	Selected Divider
0	0	÷2
0	1	÷3
1	X	Fractional Divider

Figure 18-14 ASC Baudrate Generator Circuitry in Asynchronous Modes

Using the Fixed Input Clock Divider

The baudrate for asynchronous operation of serial channel ASC when using the fixed input clock divider ratios (FDE = 0) and the required reload value for a given baudrate can be determined by the following formulas:

BG represents the contents of the reload bitfield BR_VALUE, taken as unsigned 13-bit integer.

The maximum baudrate that can be achieved for the Asynchronous Modes when using the two fixed clock divider and a module clock of 40 MHz is 1.25 Mbit/s. **Table 18-4** lists various commonly used baudrates together with the required reload values and the deviation errors compared to the intended baudrate.

*Note: FDE must be 0 to achieve the baudrates in **Table 18-3**. The deviation errors given in the table are rounded. Using a baudrate crystal will provide correct baudrates without deviation errors.*

Table 18-3 Asynchronous Baudrate Formulas Using the Fixed Input Clock Dividers

FDE	BRS	BG	Formula
0	0	0 ... 8191	$\text{Baudrate} = \frac{f_{ASC}}{32 \times (BG + 1)}$ $BG = \frac{f_{ASC}}{32 \times \text{Baudrate}} - 1$
	1		$\text{Baudrate} = \frac{f_{ASC}}{48 \times (BG + 1)}$ $BG = \frac{f_{ASC}}{48 \times \text{Baudrate}} - 1$

Table 18-4 Typical Asynchronous Baudrates Using the Fixed Input Clock Dividers

Baudrate	BRS = 0, f_{ASC} = 40 MHz		BRS = 1, f_{ASC} = 40 MHz	
	Deviation Error	Reload Value	Deviation Error	Reload Value
1.25 Mbit/s	---	0000_H	NA	NA
19.2 kbit/s	+0.1% / -1.3%	0040_H / 0041_H	+0.9% / -1.3%	$002A_H$ / $002B_H$
9600 bit/s	+0.1% / -0.6%	0081_H / 0082_H	+0.9% / -0.2%	0055_H / 0056_H
4800 bit/s	+0.1% / -0.2%	0103_H / 0104_H	+0.3% / -0.2%	$00AC_H$ / $00AD_H$
2400 bit/s	+0.1% / -0.0%	0207_H / 0208_H	+0.0% / -0.2%	$015A_H$ / $015B_H$
1200 bit/s	+0.0% / -0.0%	0410_H / 0411_H	+0.0% / -0.0%	$02B5_H$ / $02B6_H$

Using the Fractional Divider

When the fractional divider is selected, the input clock f_{DIV} for the baudrate timer is derived from the module clock f_{ASC} by a programmable divider. If bit FDE is set, the fractional divider is activated. It divides f_{ASC} by a fraction of n/512 for any value of n from 0 to 511. If n = 0, the divider ratio is 1, which means that $f_{DIV} = f_{ASC}$. In general, the fractional divider allows the baudrate to be programmed with much more accuracy than with the two fixed prescaler divider stages.

Note: BG represents the contents of the reload bitfield BR_VALUE, taken as an unsigned 13-bit integer.

Note: FDV represents the contents of the fractional divider register FD_VALUE taken as an unsigned 9-bit integer.

Table 18-5 Async. Baudrate Formulas Using the Fractional Input Clock Divider

FDE	BRS	BG	FDV	Formula
1	–	1 ... 8191	1 ... 511	$\text{Baudrate} = \frac{\text{FDV}}{512} \times \frac{f_{ASC}}{16 \times (\text{BG} + 1)}$
			0	$\text{Baudrate} = \frac{f_{ASC}}{16 \times (\text{BG} + 1)}$

Table 18-6 Typical Asynchronous Baudrates Using the Fractional Input Clock Divider

f_{ASC}	Desired Baudrate	BG	FDV	Resulting Baudrate	Deviation
40 MHz	115.2 kbit/s	04_H	076_H	115.234 kbit/s	0.02%
	57.6 kbit/s	04_H	$03B_H$	57.617 kbit/s	0.02%
	38.4 kbit/s	$0E_H$	076_H	38.411 kbit/s	0.02%
	19.2 kbit/s	$0E_H$	$03B_H$	19.206 kbit/s	0.02%

18.4.2 Baudrate in Synchronous Mode

For synchronous operation, the baudrate generator provides a clock with four times the rate of the established baudrate (see **Figure 18-15**).

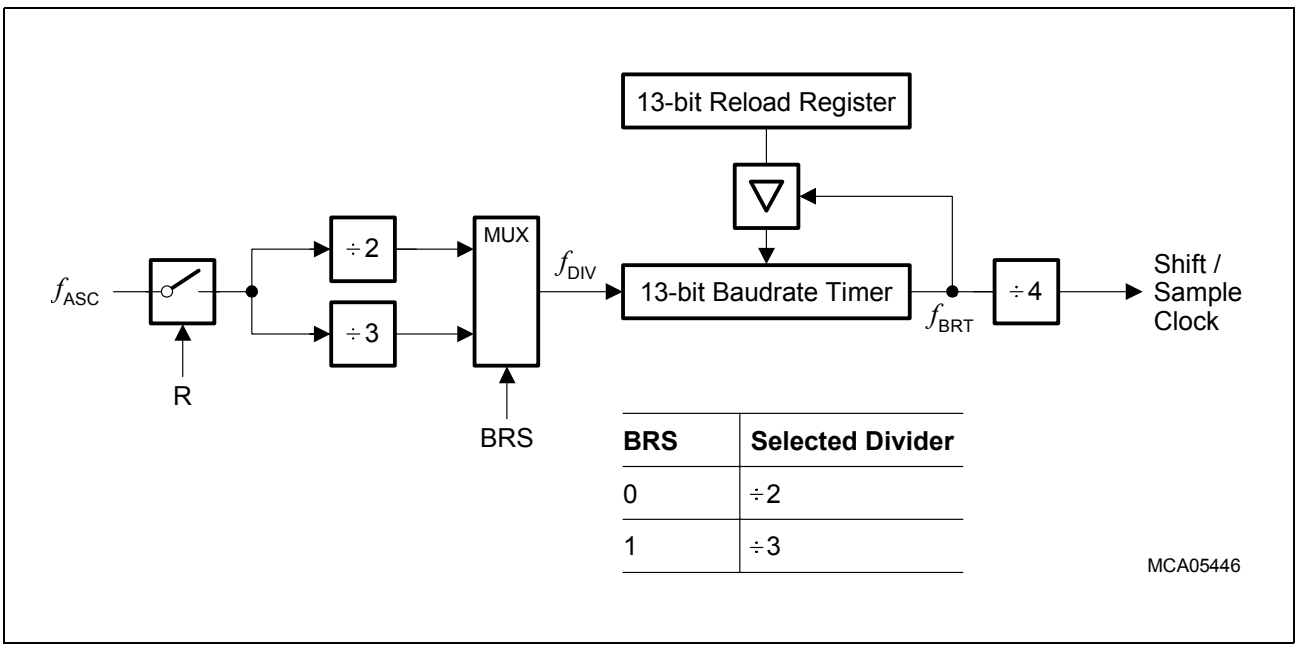


Figure 18-15 ASC Baudrate Generator Circuitry in Synchronous Mode

The baudrate for synchronous operation of serial channel ASC can be determined by the formulas as shown in **Table 18-7**.

Table 18-7 Synchronous Baudrate Formulas

BRS	BG	Formula
0	0 … 8191	$Baudrate = \frac{f_{ASC}}{8 \times (BG + 1)}$ $BG = \frac{f_{ASC}}{8 \times Baudrate} - 1$
1		$Baudrate = \frac{f_{ASC}}{12 \times (BG + 1)}$ $BG = \frac{f_{ASC}}{12 \times Baudrate} - 1$

Note: BG represents the contents of the reload bitfield BR_VALUE, taken as an unsigned 13-bit integers.

The maximum baudrate that can be achieved in Synchronous Mode when using a module clock of 40 MHz is 5 Mbit/s.

18.5 Autobaud Detection

18.5.1 General Operation

Autobaud Detection provides a capability to recognize the mode and the baudrate of an asynchronous input signal at RxD. Generally, the baudrates to be recognized must be known by the application. With this knowledge always a set of nine baudrates can be detected. The Autobaud Detection is not designed to calculate a baudrate of an unknown asynchronous frame.

Figure 18-16 shows how the Autobaud Detection is integrated into its Asynchronous Mode configuration. The RxD data line is an input to the autobaud detection unit. The clock f_{DIV}, generated by the fractional divider, is used by the autobaud detection unit as time base. After successful recognition of baudrate and Asynchronous Mode of the RxD data input signal, bits in register ASCx_CON and the value of register ASCx_BG in the baudrate timer are set to the appropriate values, and the ASC can start immediately with the reception of serial input data.

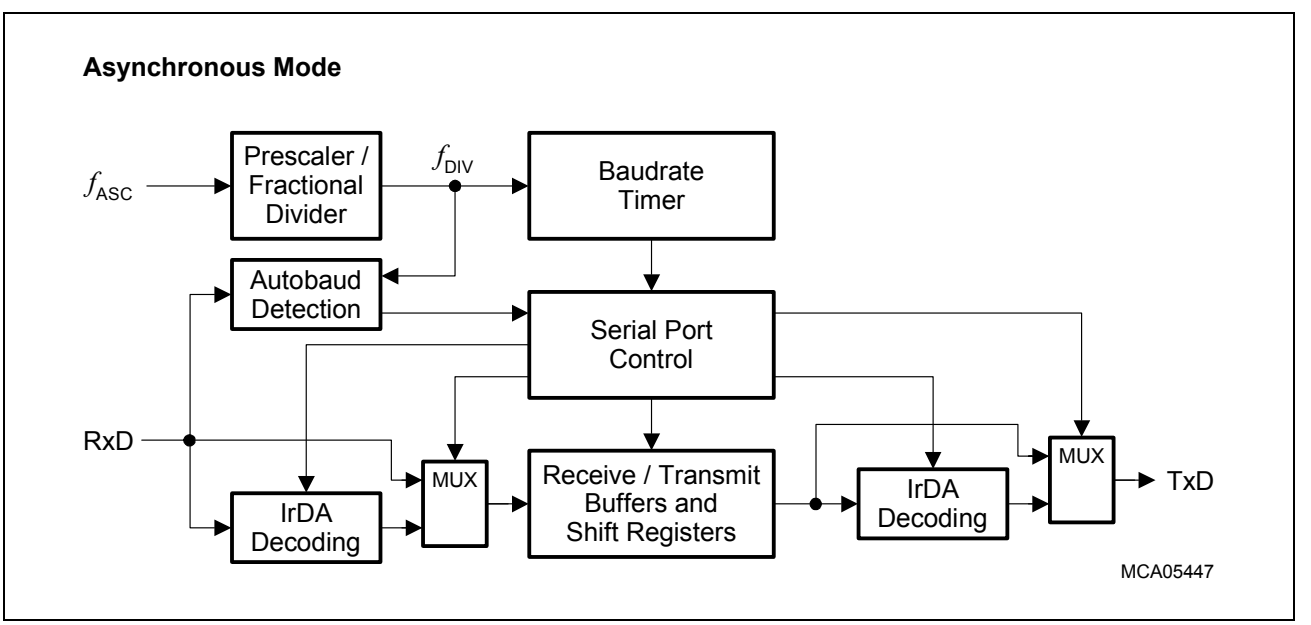


Figure 18-16 Asynchronous Mode Block Diagram

Note: Autobaud detection is not available in Synchronous Mode.

The following sequence must be executed to start the autobaud detection unit:

- Definition of the baudrates to be detected: standard or non-standard baudrates
- Programming of the prescaler/fractional divider to select a specific value of f_{DIV}
- Starting the prescaler/fractional divider (setting bit R)
- Preparing the interrupt system
- Enabling the autobaud detection (setting bit EN and the interrupt enable bits in ABCON for interrupt generation, if required)
- Polling interrupt request flag or waiting for the autobaud detection interrupt

18.5.2 Serial Frames for Autobaud Detection

The Autobaud Detection is based on the serial reception of a specific two-byte serial frame. This serial frame is build up by the two ASCII bytes “at” or “AT” (“aT” or “At” are not allowed). Both byte combinations can be detected in five types of asynchronous frames. **Figure 18-17** and **Figure 18-18** show the serial frames which are detected at least.

Note: Some other two-byte combinations will be defined too.

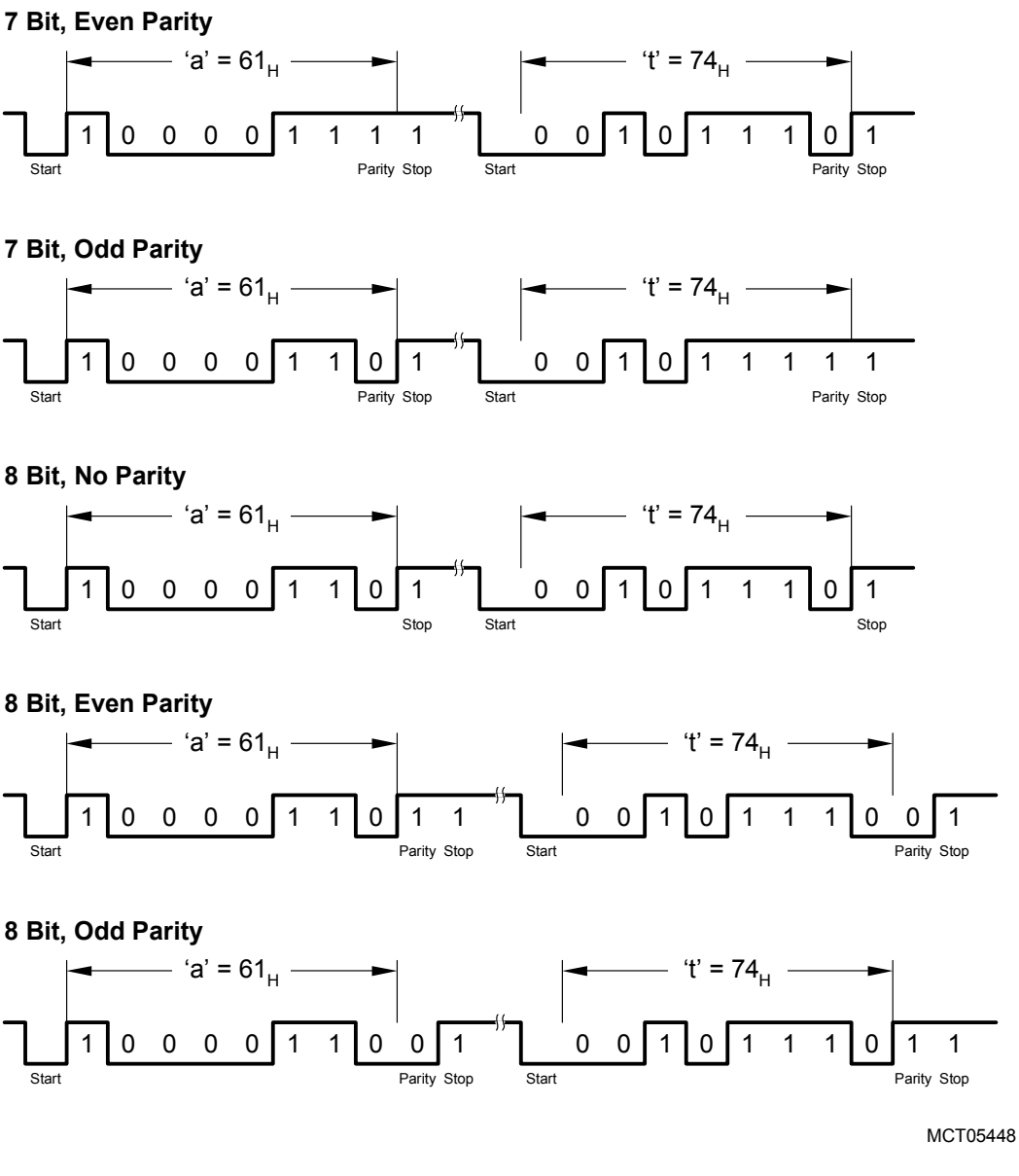


Figure 18-17 Two-Byte Serial Frames with ASCII ‘at’

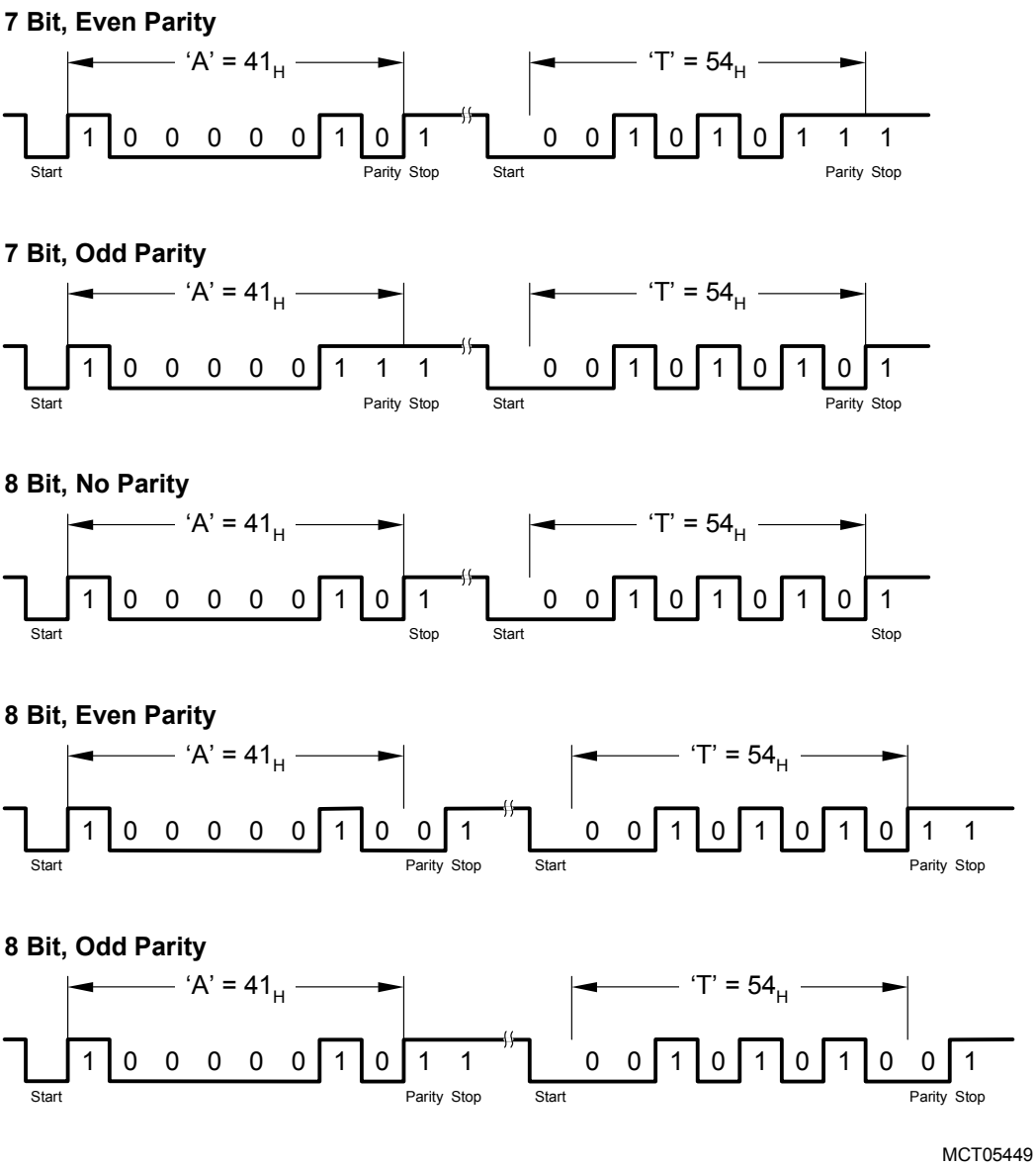


Figure 18-18 Two-Byte Serial Frames with ASCII 'AT'

18.5.3 Baudrate Selection and Calculation

Autobaud Detection requires some calculations concerning the programming of the baudrate generator and the baudrates to be detected. Two steps must be considered:

- Defining the baudrate(s) to be detected
- Programming of the baudrate timer prescaler - setup of the clock rate of f_{DIV}

In general, the baudrate generator in Asynchronous Mode is build up by two parts (see also **Figure 18-14**):

- The clock prescaler part which derives f_{DIV} from f_{ASC}
- The baudrate timer part which generates the sample clock f_{BRT} and the baudrate clock f_{BR}

Prior to an Autobaud Detection the prescaler part has to be set up by the CPU while the baudrate timer (register ASCx_BG) is initialized with a 13-bit value (BR_VALUE) automatically after a successful autobaud detection. For the following calculations, the fractional divider is used (FDE = 1).

Note: It is also possible to use the fixed divide-by-2 or divide-by-3 prescaler. But the fractional divider allows the much more precise adaption of f_{DIV} to the required value.

Standard Baudrates

For standard baudrate detection the baudrates as shown in **Table 18-8** can be e.g. detected. Therefore, the output frequency f_{DIV} of the baudrate generator must be set to a frequency derived from the module clock f_{ASC} in a way that it is equal to 11.0592 MHz. The value to be written into register FDV is the nearest integer value which is calculated according the following formula:

$$FDV = \frac{512 \times 11.0592\ MHz}{f_{ASC}} \quad (18.1)$$

Table 18-8 defines the nine standard baudrates (Br0 - Br8) which can be detected for f_{DIV} = 11.0592 MHz.

Table 18-8 Autobaud Detection Using Standard Baudrates (f_{DIV} = 11.0592 MHz)

Baudrate Numbering	Detectable Standard Baudrate	Divide Factor d_f	BG is Loaded after Detection with Value
Br0	230.400 kbit/s	48	$2 = 002_H$
Br1	115.200 kbit/s	96	$5 = 005_H$
Br2	57.600 kbit/s	192	$11 = 00B_H$
Br3	38.400 kbit/s	288	$17 = 011_H$
Br4	19.200 kbit/s	576	$35 = 023_H$
Br5	9600 bit/s	1152	$71 = 047_H$
Br6	4800 bit/s	2304	$143 = 08F_H$
Br7	2400 bit/s	4608	$287 = 11F_H$
Br8	1200 bit/s	9216	$575 = 23F_H$

According to **Table 18-8** a baudrate of 9600 bit/s is achieved when register ASCx_BG is loaded with a value of 047_H, assuming that f_{DIV} has been set to 11.0592 MHz.
Table 18-8 also lists a divide factor d_f which is defined with the following formula:

$$\text{Baudrate} = \frac{f_{DIV}}{d_f} \quad (18.2)$$

This divide factor d_f defines a **fixed** relationship between the prescaler output frequency f_{DIV} and the baudrate to be detected during the Autobaud Detection operation. This means, changing f_{DIV} results in a totally different baudrate table in means of baudrate values. For the baudrates to be detected, the following relations are always valid:

Br0 = $f_{DIV}/48_D$, Br1 = $f_{DIV}/96_D$, … up to Br8 = $f_{DIV}/9216_D$

A requirement for detecting standard baudrates up to 230.400 kbit/s is the f_{DIV} minimum value of 11.0592 MHz. With the value FD_VALUE the fractional divider f_{DIV} is adapted to the module clock frequency f_{ASC}. **Table 18-9** defines the deviation of the standard baudrates when using autobaud detection depending on the module clock f_{ASC}.

Table 18-9 Standard Baudrates - Deviations and Errors for Autobaud Detection

f_{ASC}	FDV	Error in f_{DIV}
10 MHz	not possible	
12 MHz	472	+0.03%
13 MHz	436	+0.1%
16 MHz	354	+0.03%
18 MHz	315	+0.14%
18.432 MHz	307	-0.07%
20 MHz	283	-0.04%
24 MHz	236	+0.03%
25 MHz	226	-0.22%
30 MHz	189	+0.14%
33 MHz	172	+0.24%
40 MHz	142	+0.31%

Note: If the deviation of the baudrate after autobaud detection is to high, the baudrate generator (fractional divider FDV and reload register ASCx_BG) can be reprogrammed if required to get a more precise baudrate with less error.

Non-Standard Baudrates

Due to the relationship between Br0 to Br8 in **Table 18-8** concerning the divide factor d_f other baudrates than the standard baudrates can be also selected. E.g. if a baudrate of 50 kbit/s has to be detected, Br2 is e.g. defined as baudrate for the 50 kbit/s selection. This further results in:

f_{DIV} = 50 kbit/s × d_f@Br2 = 50 kbit/s × 192 = 9.6 MHz

Therefore, depending on the module clock frequency f_{ASC}, the value of the fractional divider (register FDV must be set in this example according to the formula:

$$FDV = \frac{512 \times f_{DIV}}{f_{ASC}} \quad \text{with } f_{DIV} = 9.6 \text{ MHz} \tag{18.3}$$

Using this selection (f_{DIV} = 9.6 MHz), the detectable baudrates start at 200 kbit/s (Br0) down to 1042 bit/s (Br8). **Table 18-10** shows the baudrate table for this example.

Table 18-10 Autobaud Detection Using Non-Standard Baudrates (f_{DIV} = 9.6 MHz)

Baudrate Numbering	Detectable Non-Standard Baudrates	Divide Factor d_f	BG is Loaded after Detection with Value
Br0	200.000 kbit/s	48	$2 = 002_H$
Br1	100.000 kbit/s	96	$5 = 005_H$
Br2	50 kbit/s	192	$11 = 00B_H$
Br3	33.333 kbit/s	288	$17 = 011_H$
Br4	16.667 kbit/s	576	$35 = 023_H$
Br5	8333 bit/s	1152	$71 = 047_H$
Br6	4167 bit/s	2304	$143 = 08F_H$
Br7	2083 bit/s	4608	$287 = 11F_H$
Br8	1047 bit/s	9216	$575 = 23F_H$

18.5.4 Overwriting Registers on Successful Autobaud Detection

With a successful Autobaud Detection some bits in registers ASCx_CON and ASCx_BG are automatically set to a value which corresponds to the mode and baudrate of the detected serial frame conditions (see **Table 18-11**). In control register ASCx_CON the mode control bits M and the parity select bit ODD are overwritten. Register ASCx_BG is loaded with the 13-bit reload value for the baudrate timer.

Table 18-11 Autobaud Detection Overwrite Values for the CON Register

Detected Parameters		M	ODD	BR_VALUE
Operating Mode	7 bit, even parity 7 bit, odd parity 8 bit, even parity 8 bit, odd parity 8 bit, no parity	0 1 1 0 1 1 1 1 1 1 1 1 0 0 1	0 1 0 1 0	–
Baudrate	Br0 Br1 Br2 Br3 Br4 Br5 Br6 Br7 Br8	–	–	$2 = 002_H$ $5 = 005_H$ $11 = 00B_H$ $17 = 011_H$ $35 = 023_H$ $71 = 047_H$ $143 = 08F_H$ $287 = 11F_H$ $575 = 23F_H$

*Note: The autobaud detection interrupts are described in **Section 18.7**.*

18.6 Hardware Error Detection Capabilities

To improve the safety of serial data exchange, the serial channel ASC provides an error interrupt request flag to indicate the presence of an error, and three (selectable) error status flags in register ASCx_CON to indicate which error has been detected during reception. Upon completion of a reception, the error interrupt request line EIR will be activated simultaneously with the receive interrupt request line RIR, if one or more of the following conditions are met:

- If the framing error detection enable bit FEN is set and any of the expected stop bits is not high, the framing error flag FE is set, indicating that the error interrupt request is due to a framing error (Asynchronous Mode only).
- If the parity error detection enable bit PEN is set in the modes where a parity bit is received, and the parity check on the received data bits proves false, the parity error flag PE is set, indicating that the error interrupt request is due to a parity error (Asynchronous Mode only).
- If the overrun error detection enable bit OEN is set and the last character received was not read out of the receive buffer by software or by a DMA transfer at the time the reception of a new frame is complete, the overrun error flag OE is set indicating that the error interrupt request is due to an overrun error (Asynchronous and Synchronous Mode).

18.7 Interrupts

Six interrupt sources are provided for serial channel ASC. Line TIR indicates a transmit interrupt, TBIR indicates a transmit buffer interrupt, RIR indicates a receive interrupt and EIR indicates an error interrupt of the serial channel. The autobaud detection unit provides two additional interrupts, the ABSTIR start of autobaud operation interrupt and the ABDETIR autobaud detected interrupt. The interrupt output lines TBIR, TIR, RIR, EIR, ABSTIR, and ABDETIR are activated (active state) for two periods of the module clock f_{ASC}.

The cause of an error interrupt request (framing, parity, overrun error) can be identified by the error status flags FE, PE, and OE. For the two autobaud detection interrupts register ABSTAT provides status information.

Note: In contrary to the error interrupt request line EIR, the error status flags FE/PE/OE are not reset automatically but must be cleared by software.

For normal operation (i.e. besides the error interrupt) the ASC provides three interrupt requests to control data exchange via this serial channel:

- TBIR is activated when data is moved from TBUF to the transmit shift register.
- TIR is activated before the last bit of an asynchronous frame is transmitted, or after the last bit of a synchronous frame has been transmitted.
- RIR is activated when the received frame is moved to RBUF.

Note: While the receive task is handled by a single interrupt handler, the transmitter is serviced by two interrupt handlers. This provides advantages for the servicing software.

For single transfers it is sufficient to use the transmitter interrupt (TIR), which indicates that the previously loaded data has been transmitted, except for the last bit of an asynchronous frame. For multiple back-to-back transfers it is necessary to load the following piece of data at last until the time the last bit of the previous frame has been transmitted. In Asynchronous Mode this leaves just one bit-time for the handler to respond to the transmitter interrupt request, in Synchronous Mode it is impossible at all.

Using the transmit buffer interrupt (TBIR) to reload transmit data gives the time to transmit a complete frame for the service routine, as TBUF may be reloaded while the previous data is still being transmitted.

The start of autobaud operation interrupt ABSTIR is generated whenever the autobaud detection unit is enabled (ABEN and ABDETEN and ABSTEN are set), and a start bit has been detected at RxD. In this case ABSTIR is generated during Autobaud Detection whenever a start bit is detected.

The autobaud detected interrupt ABDETIR is always generated after recognition of the second character of the two-byte frame, this means after a successful Autobaud Detection. If FCDETEN is set the autobaud detected interrupt ABDETIR is also generated after the recognition of the **first** character of the two-byte frame.

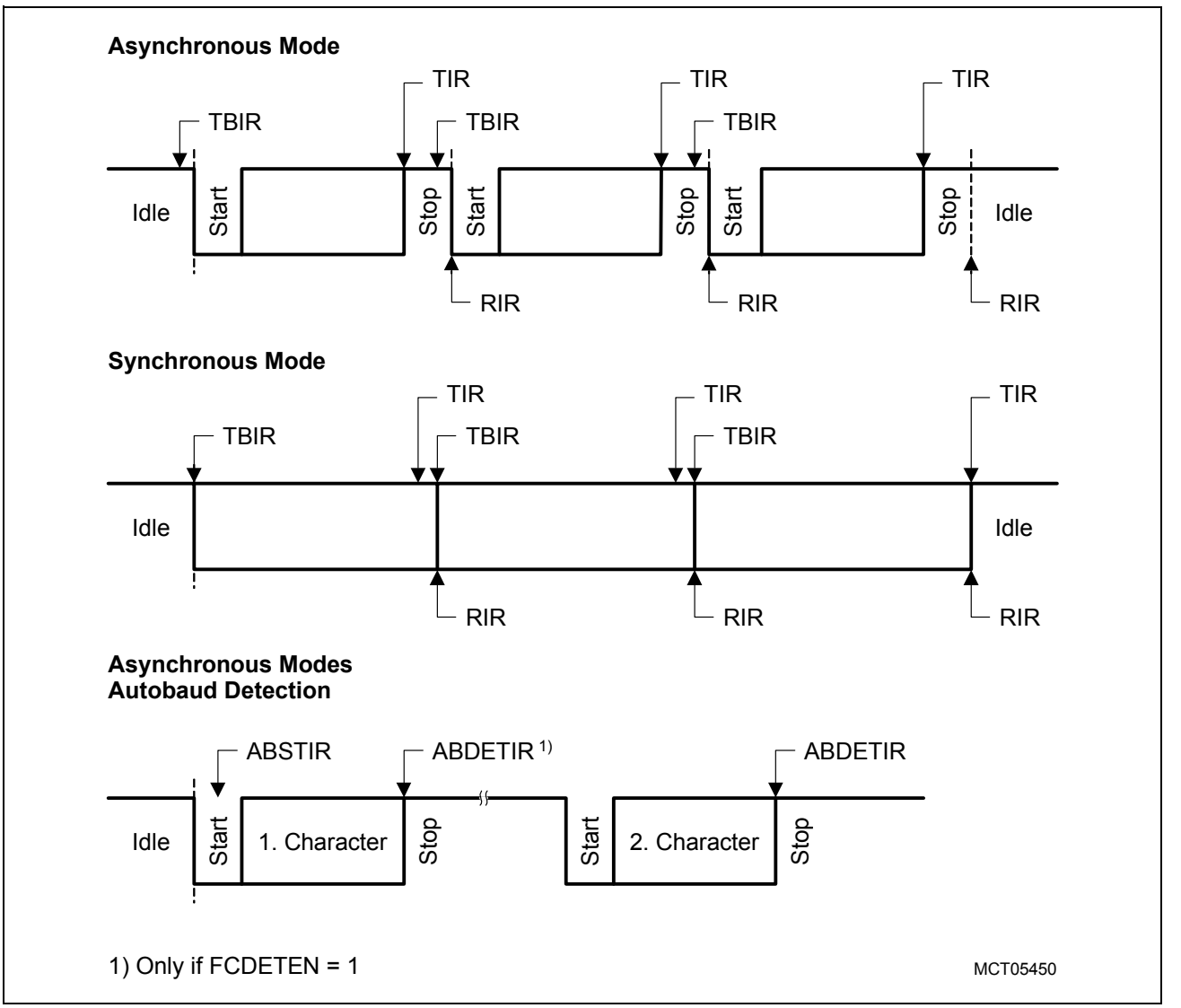


Figure 18-19 ASC Interrupt Generation

As shown in **Figure 18-19**, TBIR is an early trigger for the reload routine, while TIR indicates the completed transmission. Therefore, software using handshake should rely on TIR at the end of a data block to ensure that all data has actually been transmitted.

The six interrupts of the ASC0 and of the ASC1 module are controlled by the following service request control registers:

- ASC0_ABIC, ASC1_ABIC: control the autobaud interrupts
- ASC0_TIC, ASC1_TIC: control the transmit interrupts
- ASC0_RIC, ASC1_RIC: control the receive interrupts
- ASC0_EIC, ASC1_EIC: control the error interrupts
- ASC0_TBIC, ASC1_TBIC: control the transmit buffer empty interrupt

Note: Please refer to the general Interrupt Control Register description for an explanation of the control fields.

The two autobaud interrupt request lines (start of autobaud detection and end of autobaud detection) in each ASC module are ‘ORed’ together; the ‘ORed’ output signal is connected to the interrupt control register. This is shown in **Figure 18-20**.

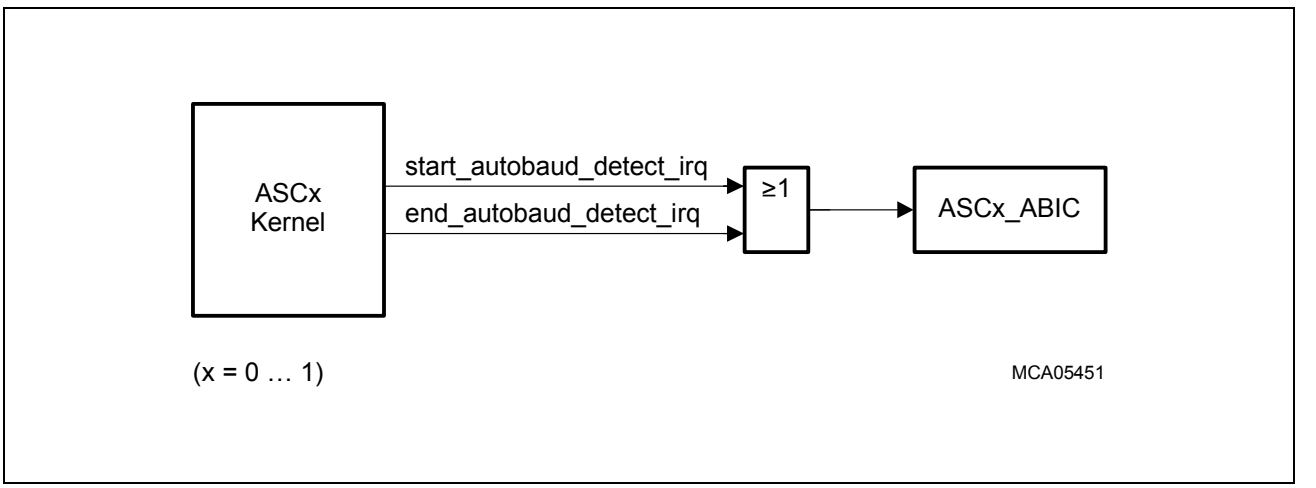


Figure 18-20 Wiring of Autobaud Interrupts

Table 18-12 summarizes all interrupt sources:

Table 18-12 ASC Interrupt Sources

Interrupt	Signal	Description
TBUF Action	TBIR	A write action to the transmit shift register from the transmit buffer register ASCx_TBUF. If a FIFO is configured for the ASC and bit TXTMEN is cleared, TXFIFL defines when the interrupt is generated depending on the FIFO fill state.
Transmit Interrupt	TIR	The interrupt is generated after the last (eight) data bit of a transmission frame is send via line TxD by the transmit shift register. *Note: Only for Synchronous Mode*
Transmit Interrupt	TIR	The interrupt is generated just before the last bit of a transmission frame is send via line TxD by the transmit shift register. If a FIFO is configured for the ASC and bit XTMEN is cleared, TXFIFL defines when the interrupt is generated depending on the FIFO fill state. *Note: Only for Asynchronous Modes*
Receive Interrupt	RIR	The interrupt is generated when the received frame is copied from the receive shift register to the receive buffer register. *Note: Only for Synchronous Mode*

Table 18-12 ASC Interrupt Sources (cont'd)

Interrupt	Signal	Description
Receive Interrupt	RIR	The interrupt is generated when the received frame is copied from the receive shift register to the receive buffer register. If a FIFO is configured for the ASC and bit RXTMEN is cleared, RXFIFL defines when the interrupt is generated depending on the FIFO fill state. *Note: Only for Asynchronous Modes*
Receive Error Interrupt	RIR and EIR	The interrupt is generated when the received frame is copied from the receive shift register to the receive buffer register and the receive buffer contains already valid data. *Note: Only for Synchronous Mode*
Receive Overflow	RIR and EIR	If an additional frame is received when the FIFO is completely full an overflow error occurs. Both interrupts are generated and the previously received frame is overwritten in the FIFO and therefore lost.
Read to empty FIFO	EIR	A read operation from the CPU to an empty receive FIFO generates this interrupt.
Transparent Read Operation	RIR	In Transparent Mode a receive interrupt is always generated on a read operation from the CPU to the receive FIFO if the FIFO is not empty after this operation.
Flush Action	TBIR	A transmit buffer interrupt is generated when the transmit FIFO is flushed.
FIFO Enable	TBIR	A transmit buffer interrupt is generated when the transmit FIFO is enabled by setting bits TXTMEN and TXFEN when it was previously disabled in Transparent Mode.
Transmit Overflow	EIR	If an additional frame is written to the transmit FIFO when it is completely full an overflow error occurred. The interrupt is generated and the previously written frame is overwritten and therefor lost in the FIFO.
Frame Error	RIR and EIR	An expected stop bit is not high. *Note: Asynchronous Mode only*
Parity Error	RIR and EIR	When a parity bit is received that does not fit to the parity of the received data. *Note: Asynchronous Mode only*

18.8 Registers

Table 18-13 shows all registers which are required for programming the ASC modules. It summarizes the ASC kernel registers and the interrupt control registers and lists their addresses.

Table 18-13 ASC Module Register Summary

Name	Description	ASC0 Addresses		Reg. Area	ASC1 Addresses	
		16-Bit	8-Bit		16-Bit	8-Bit
ASCx_CON	Control Register	$FFB0_H$	$D8_H$	SFR	$FFB8_H$	DC_H
ASCx_TBUF	Transmit Buffer Register	$FEB0_H$	58_H	SFR	$FEB8_H$	$5C_H$
ASCx_RBUF	Receive Buffer Register	$FEB2_H$	59_H	SFR	$FEBA_H$	$5D_H$
ASCx_ABCON	Autobaud Control Register	$F1B8_H$	DC_H	ESFR	$F1BC_H$	DE_H
ASCx_ABSTAT	Autobaud Status Register	$F0B8_H$	$5C_H$	ESFR	$F0BC_H$	$5E_H$
ASCx_BG	Baudrate Timer Reload Register	$FEB4_H$	$5A_H$	SFR	$FEBC_H$	$5E_H$
ASCx_FDV	Fractional Divider Register	$FEB6_H$	$5B_H$	SFR	$FEBE_H$	$5F_H$
ASCx_PMW	IrDA Pulse Mode and Width Register	$FEAA_H$	55_H	SFR	$FEAC_H$	56_H
ASCx_RXFCON	Receive FIFO Control Register	$F0C6_H$	63_H	ESFR	$F0A6_H$	53_H
ASCx_TXFCON	Transmit FIFO Control Register	$F0C4_H$	62_H	ESFR	$F0A4_H$	52_H
ASCx_FSTAT	FIFO Status Register	$F0BA_H$	$5D_H$	ESFR	$F0BE_H$	$5F_H$
ASCx_ABIC	Autobaud Interrupt Control Register	$F15C_H$	AE_H	ESFR	$F1BA_H$	DD_H
ASCx_TIC	Transmit Interrupt Control Register	$FF6C_H$	$B6_H$	SFR/ ESFR	$F182_H$	$C1_H$
ASCx_RIC	Receive Interrupt Control Register	$FF6E_H$	$B7_H$	SFR/ ESFR	$F18A_H$	$C5_H$
ASCx_EIC	Error Interrupt Control Register	$FF70_H$	$B8_H$	SFR/ ESFR	$F192_H$	$C9_H$
ASCx_TBIC	Transmit Buffer Interrupt Control Register	$F19C_H$	CE_H	ESFR	$F150_H$	$A8_H$

Control Register

The operating mode of the serial channel ASC is controlled by its control register CON. This register contains control bits for mode and error check selection, and status flags for error identification.

ASCx_CON
Control Register **SFR (Table 18-13)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	**LB**	**BRS**	**ODD**	**FDE**	**OE**	**FE**	**PE**	**OEN**	**FEN**	**PEN/ RxDI**	**REN**	**STP**	**M**		
rw	rw	rw	rw	rw	rwh	rwh	rwh	rw	rw	rw	rwh	rw		rw	

Field	Bits	Type	Description
R	15	rw	**Baudrate Generator Run Control Bit** 0 Baudrate generator disabled (ASC inactive) 1 Baudrate generator enabled *Note: BR_VALUE should only be written if R = 0.*
LB	14	rw	**Loopback Mode Enabled** 0 Loopback Mode disabled. Standard transmit/receive Mode 1 Loopback Mode enabled
BRS	13	rw	**Baudrate Selection** 0 Baud rate timer prescaler divide-by-2 selected 1 Baud rate timer prescaler divide-by-3 selected *Note: BRS is don't care if FDE = 1 (fractional divider selected).*
ODD	12	rw	**Parity Selection** 0 Even parity selected (parity bit of 1 is included in data stream on odd number of 1 and parity bit of 0 is included in data stream on even number of 1) 1 Odd parity selected (parity bit of 1 is included in data stream on even number of 1 and parity bit of 0 is included in data stream on odd number of 1)

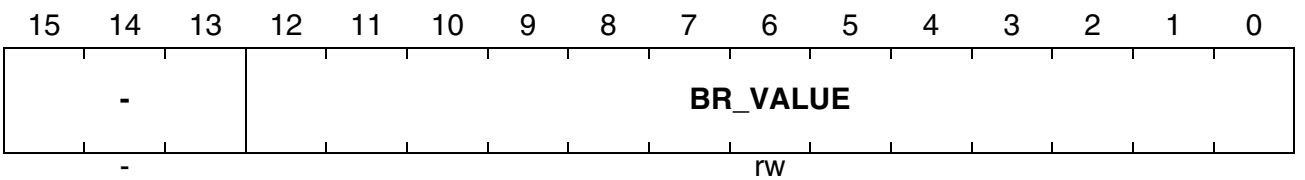
Field	Bits	Type	Description
FDE	11	rw	**Fractional Divider Enable** 0 Fractional divider disabled 1 Fractional divider enabled and used as prescaler for baudrate generator (bit BRS is don't care)
OE	10	rwh	**Overrun Error Flag** Set by hardware on an overrun/underflow error (OEN = 1). Must be cleared by software.
FE	9	rwh	**Framing Error Flag** Set by hardware on a framing error (FEN = 1). Must be cleared by software.
PE	8	rwh	**Parity Error Flag** Set by hardware on a parity error (PEN = 1). Must be cleared by software.
OEN	7	rw	**Overrun Check Enable** 0 Ignore overrun errors 1 Check overrun errors
FEN	6	rw	**Framing Check Enable** (Asynchronous Mode only) 0 Ignore framing errors 1 Check framing errors
PEN / RxDI	5	rw	**Parity Check Enable/RxDI Invert in IrDA Mode** All Asynchronous Modes without IrDA Mode (PEN): 0 Ignore parity 1 Check parity Only in IrDA Mode (RxDI): 0 RxD input is not inverted 1 RxD input is inverted
REN	4	rwh	**Receiver Enable Bit** 0 Receiver disabled 1 Receiver enabled *Note: REN is cleared by hardware after reception of a byte in synchronous mode.*
STP	3	rw	**Number of Stop Bits Selection** 0 One stop bit 1 Two stop bits

Field	Bits	Type	Description
M	[2:0]	rw	**Mode Control** 000 8-bit-data for synchronous operation 001 8-bit-data for asynchronous operation 010 8-bit-data IrDA Mode for asynchronous operation 011 7-bit-data and parity for asynchronous operation 100 9-bit-data for asynchronous operation 101 8-bit-data and wake up bit for asynchronous operation 110 Reserved. Do not use this combination 111 8-bit-data and parity for asynchronous operation

Baudrate Register

The ASC baudrate timer reload register BG contains the 13-bit reload value for the baudrate timer in Asynchronous and Synchronous Mode.

ASCx_BG
Baudrate Timer/Reload Reg. SFR (Table 18-13) Reset Value: 0000_H

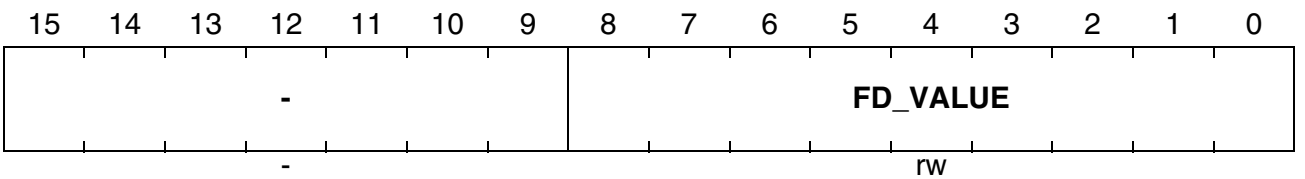


Field	Bits	Type	Description
BR_VALUE	[12:0]	rw	**Baudrate Timer/Reload Value** Reading returns the 13-bit content of the baudrate timer; writing loads the baudrate timer/reload value. *Note: BG should only be written if R = 0.*

Fractional Divider Register

The ASC fractional divider register FDV contains the 9-bit divider value for the fractional divider (Asynchronous Mode only). It is also used for reference clock generation of the autobaud detection unit.

ASCx_FDV
Fractional Divider Register **SFR (Table 18-13)** **Reset Value: 0000_H**

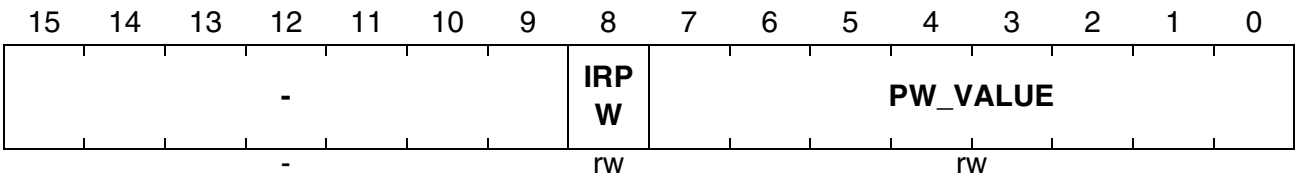


Field	Bits	Type	Description
FD_VALUE	[8:0]	rw	**Fractional Divider Register Value** FD_VALUE contains the 9-bit value of the fractional divider which defines the fractional divider ratio n/512 (n = 0 … 511). With n = 0, the fractional divider is switched off (input = output frequency, $f_{DIV} = f_{ASC}$, see **Figure 18-14**).

IrDA Pulse Mode/Width Register

The ASC IrDA pulse mode and width register PMW contains the 8-bit IrDA pulse width value and the IrDA pulse width mode select bit. This register is only required in the IrDA operating mode.

ASCx_PMW
IrDA Pulse Mode/Width Reg. **SFR (Table 18-13)** **Reset Value: 0000_H**

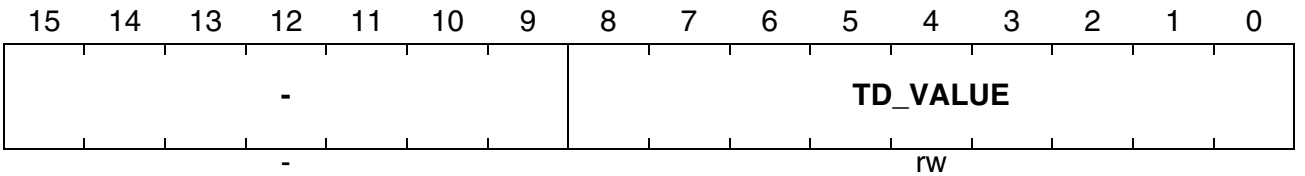


Field	Bits	Type	Description
IRPW	8	rw	**IrDA Pulse Width Selection** 0 IrDA pulse width is 3/16 bit time 1 IrDA pulse width is defined by PW_VALUE
PW_VALUE	[7:0]	rw	**IrDA Pulse Width Value** PW_VALUE is the 8-bit value n, which defines the variable pulse width of an IrDA pulse. Depending on the ASC input frequency f_{ASC}, this value can be used to adjust the IrDA pulse width to value which is not equal 3/16 bit time (e.g. 1.6 ms).

Transmitter Buffer Register

The ASC transmitter buffer register TBUF contains the transmit data value in Asynchronous and Synchronous Mode.

ASCx_TBUF
Transmit Buffer Register **SFR (Table 18-13)** **Reset Value: 0000_H**

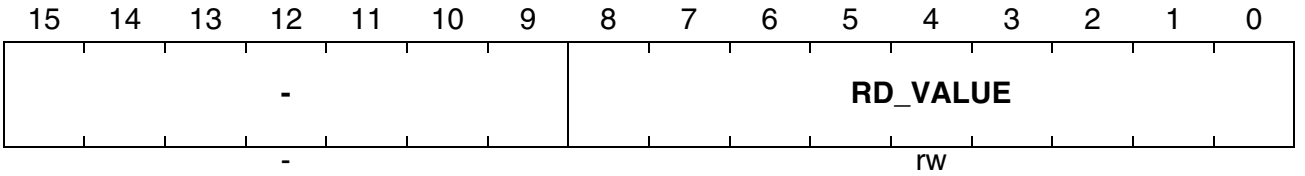


Field	Bits	Type	Description
TD_VALUE	[8:0]	rw	**Transmit Data Register Value** TBUF contains the data to be transmitted in asynchronous and synchronous operating mode of the ASC. Data transmission is double buffered. Therefore, a new value can be written to TBUF before the transmission of the previous value is complete.

Receiver Buffer Register

The ASC Receiver buffer register RBUF contains the transmit data value in Asynchronous and Synchronous Modes.

ASCx_RBUF
Receive Buffer Register **SFR (Table 18-13)** **Reset Value: 0000_H**



Field	Bits	Type	Description
RD_VALUE	[8:0]	rw	**Receive Data Register Value** RBUF contains the received data bits and, depending on the selected mode, the parity bit in asynchronous and synchronous operating mode of the ASC. In asynchronous operating mode with M = 011 (7-bit data + parity) the received parity bit is written into RD7. In asynchronous operating mode with M = 111 (8-bit data + parity) the received parity bit is written into RD8.

Autobaud Control Register

The autobaud control register ABCON of the ASC module is used to control the autobaud detection operation. It contains its general enable bit, the interrupt enable control bits, and data path control bits.

ASCx_ABCON
Autobaud Control Register **ESFR (Table 18-13)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-				RX INV	TX INV	ABEM		-			FC DET EN	AB DET EN	ABS T EN	AUR EN	AB EN
-				rw	rw	rw		-			rw	rw	rw	rw	rwh

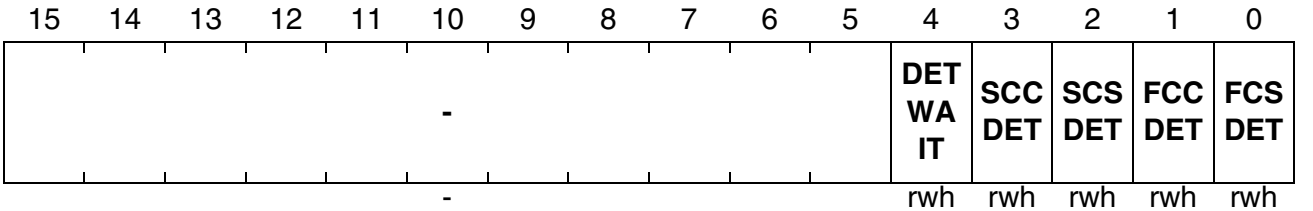
Field	Bits	Type	Description
RXINV	11	rw	**Receive Inverter Enable** 0 Receive inverter disabled 1 Receive inverter enabled
TXINV	10	rw	**Transmit Inverter Enable** 0 Transmit inverter disabled 1 Transmit inverter enabled
ABEM	[9:8]	rw	**Autobaud Echo Mode Enable** In Echo Mode the serial data at RxD is switched to TxD output. 00 Echo Mode disabled 01 Echo Mode is enabled during Autobaud Detection 10 Echo Mode is always enabled 11 Reserved; do not use this combination
FCDETEN	4	rw	**First Character of Two-Byte Frame Detected Enable** 0 Autobaud Detection interrupt ABDETIR becomes active after the two-byte frame recognition 1 Autobaud Detection interrupt ABDETIR becomes active after detection of the first **and** second byte of the two-byte frame
ABDETEN	3	rw	**Autobaud Detection Interrupt Enable** 0 Autobaud Detection interrupt disabled 1 Autobaud Detection interrupt enabled

Field	Bits	Type	Description
ABSTEN	2	rw	**Start of Autobaud Detection Interrupt Enable** 0 Start of Autobaud Detection interrupt disabled 1 Start of Autobaud Detection interrupt enabled
AUREN	1	rw	**Automatic Autobaud Control of CON.REN** 0 CON.REN is not affected during autobaud detection 1 CON.REN is cleared (receiver disabled) when ABEN and AUREN are set together. CON.REN is set (receiver enabled) after a successful Autobaud Detection (with the stop bit detection of the second character)
ABEN	0	rwh	**Autobaud Detection Enable** 0 Autobaud detection is disabled 1 Autobaud detection is enabled *Note: ABEN is reset by hardware after a successful Autobaud Detection; (with the stop bit detection of the second character). Resetting ABEN by software if it was set aborts the Autobaud Detection.*

Autobaud Status Register

The autobaud status register ABSTAT of the ASC module indicates the status of the autobaud detection operation.

ASCx_ABSTAT
Autobaud Status Register **ESFR (Table 18-13)** **Reset Value: 0000_H**



Field	Bits	Type	Description
DETWAIT	4	rwh	**Autobaud Detection is Waiting** 0 Either character ‘a’, ‘A’, ‘t’, or ‘T’ has been detected 1 The autobaud detection unit waits for the first ‘a’ or ‘A’ Bit is cleared when either FCSDET or FCCDET is set (‘a’ or ‘A’ detected). Bit can be also cleared by software. DETWAIT is set by hardware when ABEN is set.
SCCDET	3	rwh	**Second Character with Capital Letter Detected** 0 No capital ‘T’ character detected 1 Capital ‘T’ character detected Bit is cleared by hardware when ABEN is set or if FCSDET or FCCDET or SCSDET is set. Bit can be also cleared by software.
SCSDET	2	rwh	**Second Character with Small Letter Detected** 0 No small ‘t’ character detected 1 Small ‘t’ character detected Bit is cleared by hardware when ABEN is set or if FCSDET or FCCDET or SCCDET is set. Bit can be also cleared by software.

Field	Bits	Type	Description
FCCDET	1	rwh	**First Character with Capital Letter Detected** 0 No capital ‘A’ character detected 1 Capital ‘A’ character detected Bit is cleared by hardware when ABEN is set or if FCSDET or SCSDET or SCCDET is set. Bit can be also cleared by software.
FCSDET	0	rwh	**First Character with Small Letter Detected** 0 No small ‘a’ character detected 1 Small ‘a’ character detected Bit is cleared by hardware when ABEN is set or if FCCDET or SCSDET or SCCDET is set. Bit can be also cleared by software.

*Note: SCSDET or SCCDET are set when the second character has been recognized. ABEN is reset and ABDETIR set **after** SCSDET or SCCDET have been set.*

Receive FIFO Control Register

ASCx_RXFCON
Receive FIFO Control Reg. ESFR (Table 18-13) Reset Value: 0100_H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-				RXFITL				-					RXTMEN	RXFFLU	RXFEN
-				rw				-					rw	rw	rw

Field	Bits	Type	Description
RXFITL	[11:8]	rw	**Receive FIFO Interrupt Trigger Level** Defines a receive FIFO interrupt trigger level. A receive interrupt request (RIR) is generated after the reception of a byte when the filling level of the receive FIFO is equal to or greater than RXFITL. 0000 Reserved. Do not use this combination 0001 Interrupt trigger level is set to one 0010 Interrupt trigger level is set to two … … 0111 Interrupt trigger level is set to seven 1000 Interrupt trigger level is set to eight *Note: In Transparent Mode this bitfield is don't care.* *Note: Combinations defining an interrupt trigger level greater than the FIFO size should not be used.*
RXTMEN	2	rw	**Receive FIFO Transparent Mode Enable** 0 Receive FIFO Transparent Mode is disabled 1 Receive FIFO Transparent Mode is enabled *Note: This bit is don't care if the receive FIFO is disabled (RXFEN = 0).*

Field	Bits	Type	Description
RXFFLU	1	rw	**Receive FIFO Flush** 0 No operation 1 Receive FIFO is flushed *Note: Setting RXFFLU clears bitfield RXFFL in register FSTAT. RXFFLU is always read as 0.*
RXFEN	0	rw	**Receive FIFO Enable** 0 Receive FIFO is disabled 1 Receive FIFO is enabled *Note: Resetting RXFEN automatically flushes the receive FIFO.*

Note: After a successful autobaud detection sequence, the RXFIFO should be flushed before data is received.

Transmit FIFO Control Register

ASCx_TXFCON
Transmit FIFO Control Reg. ESFR (Table 18-13) Reset Value: 0100_H

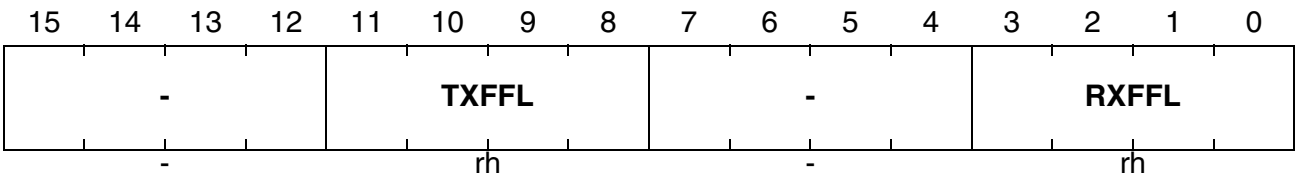
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-				TXFITL				-					TXTMEN	TXFFLU	TXFEN
-				rw				-					rw	rw	rw

Field	Bits	Type	Description
TXFITL	[11:8]	rw	**Transmit FIFO Interrupt Trigger Level** Defines a transmit FIFO interrupt trigger level. A transmit interrupt request (TIR) is generated after the transfer of a byte when the filling level of the transmit FIFO is equal to or lower than TXFITL. 0000 Reserved. Do not use this combination 0001 Interrupt trigger level is set to one 0010 Interrupt trigger level is set to two … … 0111 Interrupt trigger level is set to seven 1000 Interrupt trigger level is set to eight *Note: In Transparent Mode this bitfield is don't care.* *Note: Combinations defining an interrupt trigger level greater than the FIFO size should not be used.*
TXTMEN	2	rw	**Transmit FIFO Transparent Mode Enable** 0 Transmit FIFO Transparent Mode is disabled 1 Transmit FIFO Transparent Mode is enabled *Note: This bit is don't care if the receive FIFO is disabled (TXFEN = 0).*

Field	Bits	Type	Description
TXFFLU	1	rw	**Transmit FIFO Flush** 0 No operation 1 Transmit FIFO is flushed *Note: Setting TXFFLU clears bitfield TXFFL in register ASCx_FSTAT. TXFFLU is always read as 0.*
TXFEN	0	rw	**Transmit FIFO Enable** 0 Transmit FIFO is disabled 1 Transmit FIFO is enabled *Note: Resetting TXFEN automatically flushes the transmit FIFO.*

FIFO Status Register

ASCx_FSTAT
FIFO Status Register **ESFR (Table 18-13)** **Reset Value: 0000_H**



Field	Bits	Type	Description
TXFFL	[11:8]	rh	**Transmit FIFO Filling Level** 0000 Transmit FIFO is filled with zero bytes 0001 Transmit FIFO is filled with one byte … … 0111 Transmit FIFO is filled with seven bytes 1000 Transmit FIFO is filled with eight bytes *Note: TXFFL is cleared after a receive FIFO flush operation.*
RXFFL	[3:0]	rh	**Receive FIFO Filling Level** 0000 Receive FIFO is filled with zero bytes 0001 Receive FIFO is filled with one byte … … 0111 Receive FIFO is filled with seven bytes 1000 Receive FIFO is filled with eight bytes *Note: RXFFL is cleared after a receive FIFO flush operation.*

18.9 Interfaces of the ASC Modules

In the XC161 the ASC modules are connected to IO ports and other internal modules according to **Figure 18-21** and **Figure 18-22**.

The input/output lines of ASC0 and ASC1 are connected to pins of Ports P3. The 6 interrupt request lines of each module are connected to the Interrupt Control Block.

Clock control and emulation control of the SSC Module is handled by the System Control Unit, SCU.

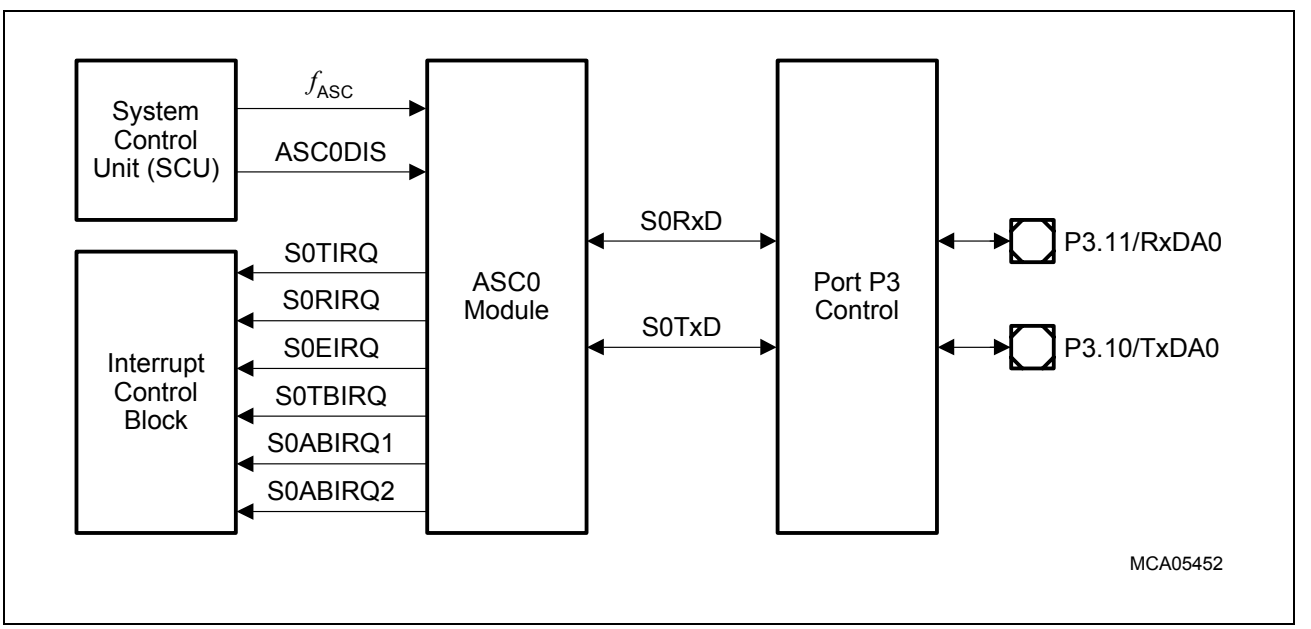


Figure 18-21 ASC0 Module Interfaces

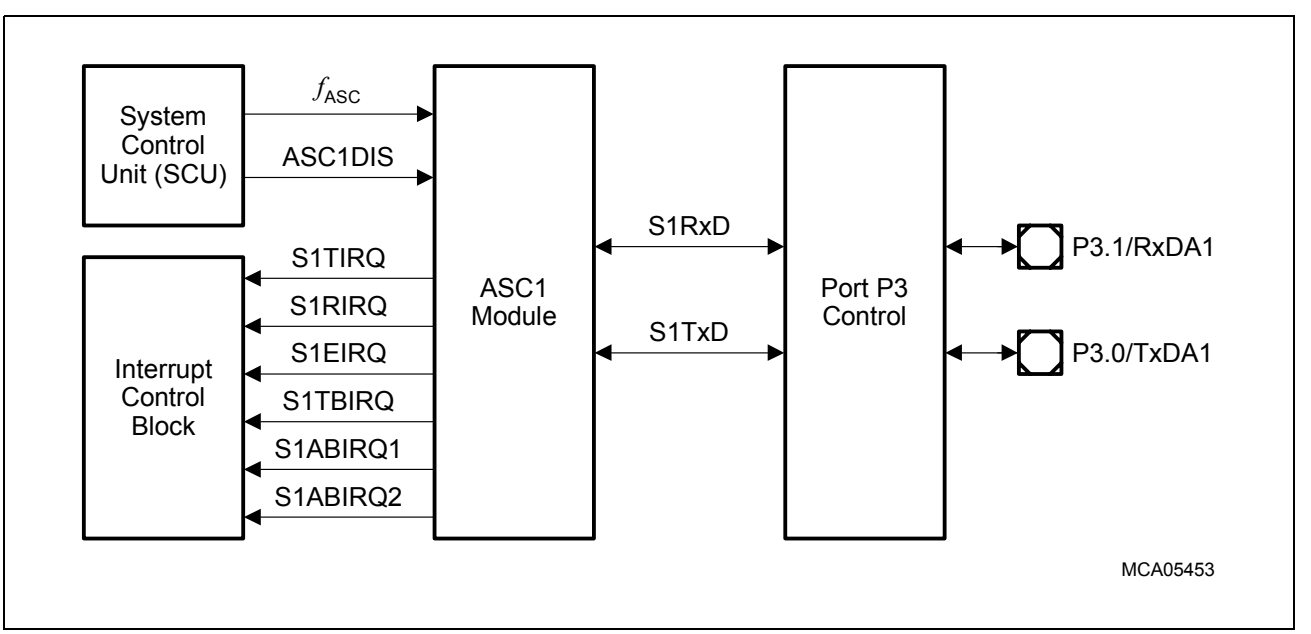


Figure 18-22 ASC1 Module Interfaces

Note: In synchronous operating mode, the direction of the RxD pin is not automatically set by the ASC modules; it must be switched by software via the corresponding bit in register DP3, depending on the selected mode (receive or transmit data).

Note: To select RxDA1 as alternate output function for P3.1, it is sufficient to set bit 1 of register ALTSEL0P3, the corresponding bit in register ALTSEL1P3 is “don’t care”.

19 High-Speed Synchronous Serial Interface (SSC)

The XC161 contains two High-Speed Synchronous Serial Interfaces, SSC0 and SSC1. The following sections present the general features and operations of such an SSC module. The final section describes the actual implementation of the two SSC modules including their interconnections with other on-chip modules.

19.1 Introduction

The High-Speed Synchronous Serial Interface (SSC) supports both full-duplex and half-duplex serial synchronous communication up to 20 Mbit/s (@ 40 MHz module clock). The serial clock signal can be generated by the SSC itself (Master Mode) or can be received from an external master (Slave Mode). Data width, shift direction, clock polarity, and phase are programmable. This supports communication with SPI-compatible devices. Transmission and reception of data is double-buffered. A 16-bit baudrate generator provides the SSC with a separate serial clock signal.

Features and Functions

- Master and Slave Mode operation
  - Full-duplex or half-duplex operation
- Flexible data format
  - Programmable number of data bits: 2 to 16 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
- Baudrate generation from 20 Mbit/s to 306.6 bit/s (@ 40 MHz module clock)
- Interrupt generation
  - On a Transmitter-Empty condition
  - On a Receiver-Full condition
  - On an Error condition (receive, phase, baudrate, transmit error)

19.2 Operational Overview

The high-speed synchronous serial interface can be configured in a very flexible way, so it can be used with other synchronous serial interfaces, can serve for master/slave or multimaster interconnections or can operate compatible with the popular SPI interface. Thus, the SSC can be used to communicate with shift registers (IO expansion), peripherals (e.g. EEPROMs, etc.) or other controllers (networking). The SSC supports half-duplex and full-duplex communication. Data is transmitted on lines MTX/STX or received on lines MRX/SRX, connected with pins MTSR (Master Transmit/Slave Receive) and MRST (Master Receive/Slave Transmit). The clock signal is output via line

MSCLK (Master Serial Shift Clock) or input via line SSCLK (Slave Serial Shift Clock). Both lines are connected to pin SCLK. These pins are alternate functions of port pins.

A block diagram of the SSC Module is shown in **Figure 19-2**.

From the programmer's point of view, the term 'SSC unit' refers to a set of registers (see **Figure 19-1**) which are associated with this peripheral, including the port pins which may be used for alternate input/output functions, and including their direction control bits.

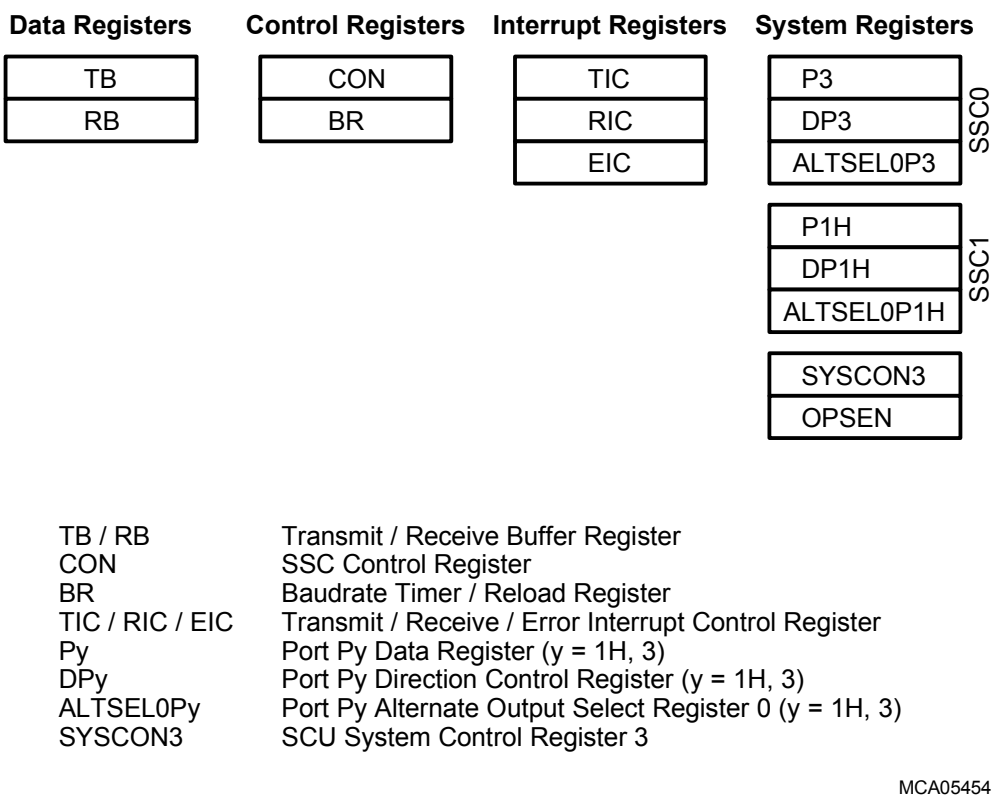


Figure 19-1 SFRs Associated with the SSC Unit

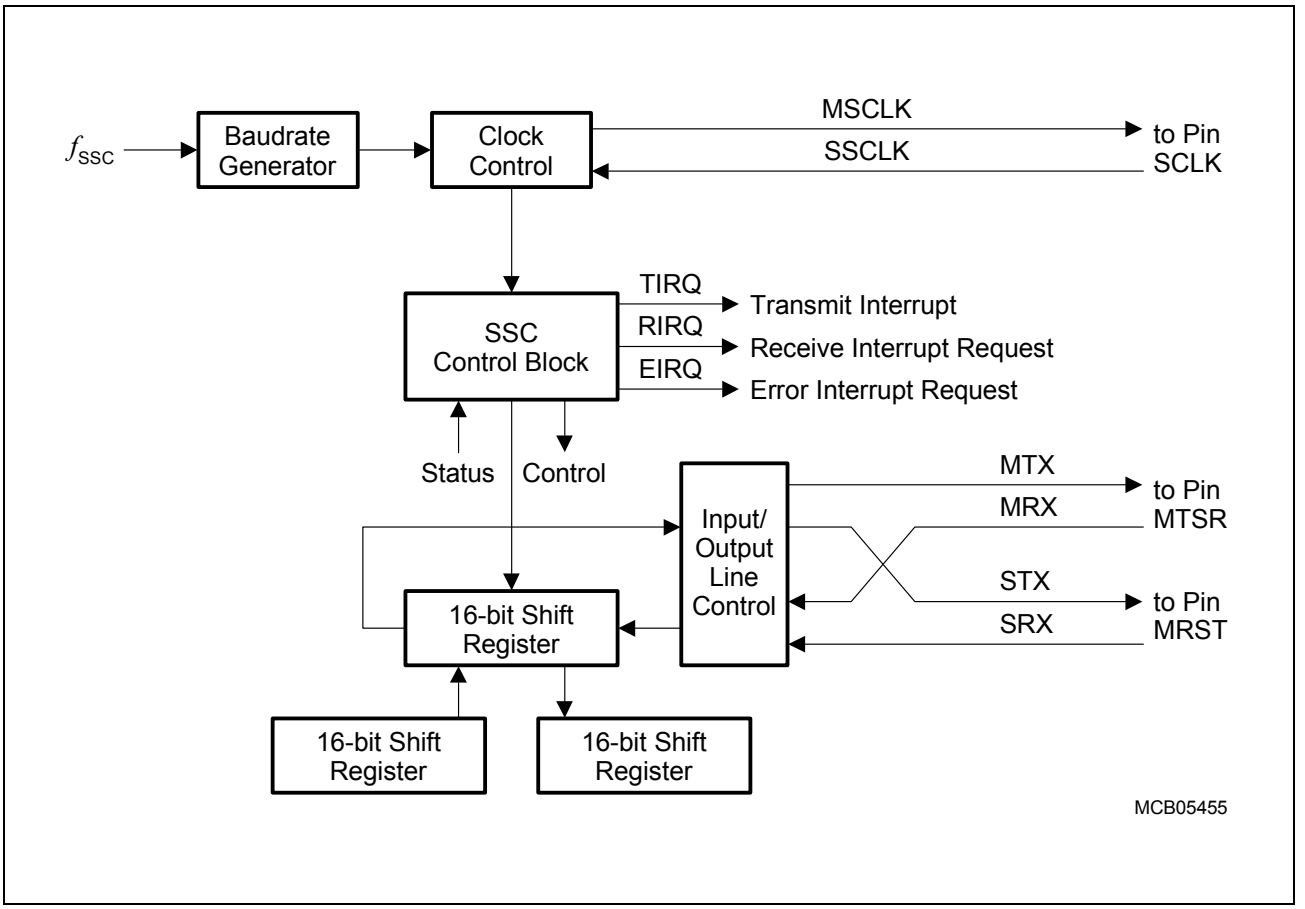


Figure 19-2 Synchronous Serial Channel (SSC) Block Diagram

19.2.1 Operating Mode Selection

The operating mode of the SSC module is controlled by its control register SSCx_CON. This register has a double function:

- During programming (SSC disabled by SSCx_CON.EN = 0), it provides access to a set of control bits
- During operation (SSC enabled by SSCx_CON.EN = 1), it provides access to a set of status flags

In the following, the layout of register CON is shown for both functions.

SSC Control Register (SSCx_CON.EN = 0: Programming Mode)

SSCx_CON
SSC Control Register **SFR (Table 19-2)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EN = 0	MS	-	A REN	BEN	PEN	REN	TEN	LB	PO	PH	HB	BM			
rw	rw	-	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw			

Field	Bits	Type	Description
EN	15	rw	**Enable Bit = 0** Transmission and reception disabled. Access to control bits.
MS	14	rw	**Master Select** 0 Slave Mode. Operate on shift clock received via SCLK. 1 Master Mode. Generate shift clock and output it via SCLK.
AREN	12	rw	**Automatic Reset Enable** 0 No additional action upon a baudrate error 1 The SSC is automatically reset upon a baudrate error
BEN	11	rw	**Baudrate Error Enable** 0 Ignore baudrate errors 1 Check baudrate errors
PEN	10	rw	**Phase Error Enable** 0 Ignore phase errors 1 Check phase errors
REN	9	rw	**Receive Error Enable** 0 Ignore receive errors 1 Check receive errors
TEN	8	rw	**Transmit Error Enable** 0 Ignore transmit errors 1 Check transmit errors
LB	7	rw	**Loop Back Control** 0 Normal output 1 Receive input is connected with transmit output (half-duplex mode)

Field	Bits	Type	Description
PO	6	rw	**Clock Polarity Control** 0 Idle clock line is low, leading clock edge is low-to-high transition. 1 Idle clock line is high, leading clock edge is high-to-low transition.
PH	5	rw	**Clock Phase Control** 0 Shift transmit data on the leading clock edge, latch on trailing edge. 1 Latch receive data on leading clock edge, shift on trailing edge.
HB	4	rw	**Heading Control** 0 Transmit/Receive LSB First 1 Transmit/Receive MSB First
BM	[3:0]	rw	**Data Width Selection** 0000 Reserved. Do not use this combination. 0001 Transfer Data Width is 2 bits … Transfer Data Width is (<BM> + 1) 1111 Transfer Data Width is 16 bits

SSC Control Register (SSCx_CON.EN = 1: Operating Mode)

SSCx_CON
SSC Control Register **SFR (Table 19-2)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EN = 1	**MS**	**-**	**BSY**	**BE**	**PE**	**RE**	**TE**	**-**	**-**	**-**	**-**	**BC**			
rw	rw	-	rh	rwh	rwh	rwh	rwh	-	-	-	-	rw			

Field	Bits	Type	Description
EN	15	rw	**Enable Bit = 1** Transmission and reception enabled. Access to status flags and M/S control.
MS	14	rw	**Master/Slave Selection** 0 Slave Mode. Operate on shift clock received via SCLK. 1 Master Mode. Generate shift clock and output it via SCLK.

Field	Bits	Type	Description
BSY	12	rh	**Busy Flag** Set while a transfer is in progress. **Do not write to!!!**
BE	11	rwh	**Baudrate Error Flag** 0 No error 1 More than factor 2 or 0.5 between slave's actual and expected baudrate
PE	19	rwh	**Phase Error Flag** 0 No error 1 The received data has changed around sampling clock edge
RE	9	rwh	**Receive Error Flag** 0 No error 1 A reception was completed before the receive buffer was read
TE	8	rwh	**Transmit Error Flag** 0 No error 1 A transfer has started with the slave's transmit buffer not being updated
BC	[3:0]	rh	**Bit Count Field** Shift counter is updated with every shifted bit. **Do not write to!!!**

Note: The target of an access to SSCx_CON (control bits or flags) is determined by the state of bit EN prior to the access; that is, writing $C057_H$ to SSCx_CON in programming mode (EN = 0) will initialize the SSC (EN was 0) and then turn it on (EN = 1). When writing to SSCx_CON, ensure that reserved locations receive zeros.

Transmitter Buffer Register

The SSC Transmit Buffer Register SSCx_TB (see **Table 19-2**) contains the transmit data value. Unselected bits of SSCx_TB are ignored during transmission. The transmit value must be right-aligned regardless of MSB or LSB first operation.

Receiver Buffer Register

The SSC Receive Buffer Register SSCx_RB (see **Table 19-2**) contains the receive data value. Unselected bits of SSCx_RB will be not valid and should be ignored. The received value is always right-aligned regardless of MSB or LSB first operation.

The shift register of the SSC is connected to both, the transmit lines and the receive lines via the pin control logic (see block diagram in **Figure 19-2**). 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.

To prepare for a transfer, the transmit data is written into the Transmit Buffer (SSCx_TB) by software. It 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 TIRQ will be activated to indicate that register SSCx_TB may be reloaded again. When the programmed number of bits (2 … 16) has been transferred, the contents of the shift register are moved to the Receive Buffer SSCx_RB and the Receive Interrupt Request line RIRQ is activated. If no further transfer is to take place (SSCx_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: Only one SSC can be master at a given time in a serial system.

The transfer of serial data bits can be programmed in many respects:

- The data width can be specified from 2 bits to 16 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 baudrate may be set from 306.6 bit/s up to 20 Mbit/s (@ 40 MHz module clock)
- The shift clock can be generated (MSCLK) or can be received (SSCLK)

These features allow the adaptation of the SSC to a wide range of applications in which serial data transfer is required.

The Data Width Selection supports the transfer of frames of any data length, from 2-bit “characters” up to 16-bit “characters”. Starting with the LSB (CON.HB = 0) enables communication with SSC devices in Synchronous Mode or with 8051-like serial interfaces, for example. Starting with the MSB (CON.HB = 1) enables 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 SSCx_TB and SSCx_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 SSCx_TB are ignored; the unselected bits of SSCx_RB will not be valid and should be ignored by the receiver service routine.

The Clock Control allows the adaptation of transmit and receive behavior of the SSC 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 PH selects the leading edge or the trailing edge for each function. Bit 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 edge, a 1-to-0 transition (see **Figure 19-3**).

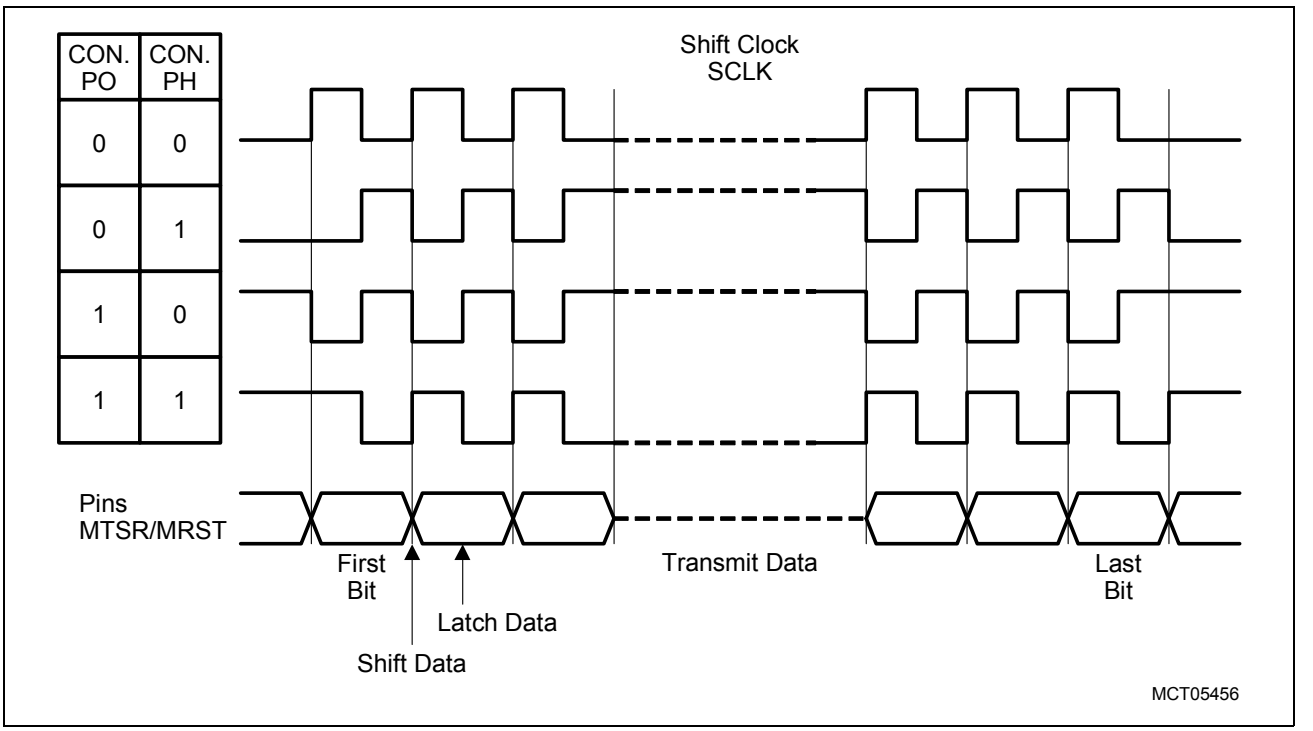


Figure 19-3 Serial Clock Phase and Polarity Options

19.2.2 Full-Duplex Operation

In a Full-Duplex serial configuration, illustrated in **Figure 19-4**, the various devices are connected via three lines. The definition of these lines is always determined by the master: The line connected to the master's data output line MTSR is the transmit line; the receive line is connected to its data input line MRST; the shift clock line is SCLK. Only the device selected for master operation generates and outputs the shift clock on line SCLK. All slaves receive this clock; thus, their SCLK pin must be switched to input mode. The output of the master's shift register is connected to the external transmit line, which in turn is connected to the slaves' shift register inputs. The outputs of the slaves' shift register are connected to the external receive line in order to enable the master to receive the data shifted out of the slaves. The external connections are hard-wired, the function and direction of these pins is determined by the master or slave operation of the individual device.

*Note: The shift direction shown in **Figure 19-4** applies for MSB-first operation as well as for LSB-first operation.*

When initializing the devices in this configuration, one device must be selected for master operation while all other devices must be programmed for slave operation. Initialization includes the operating mode of the device's SSC and also the function of the respective port lines.

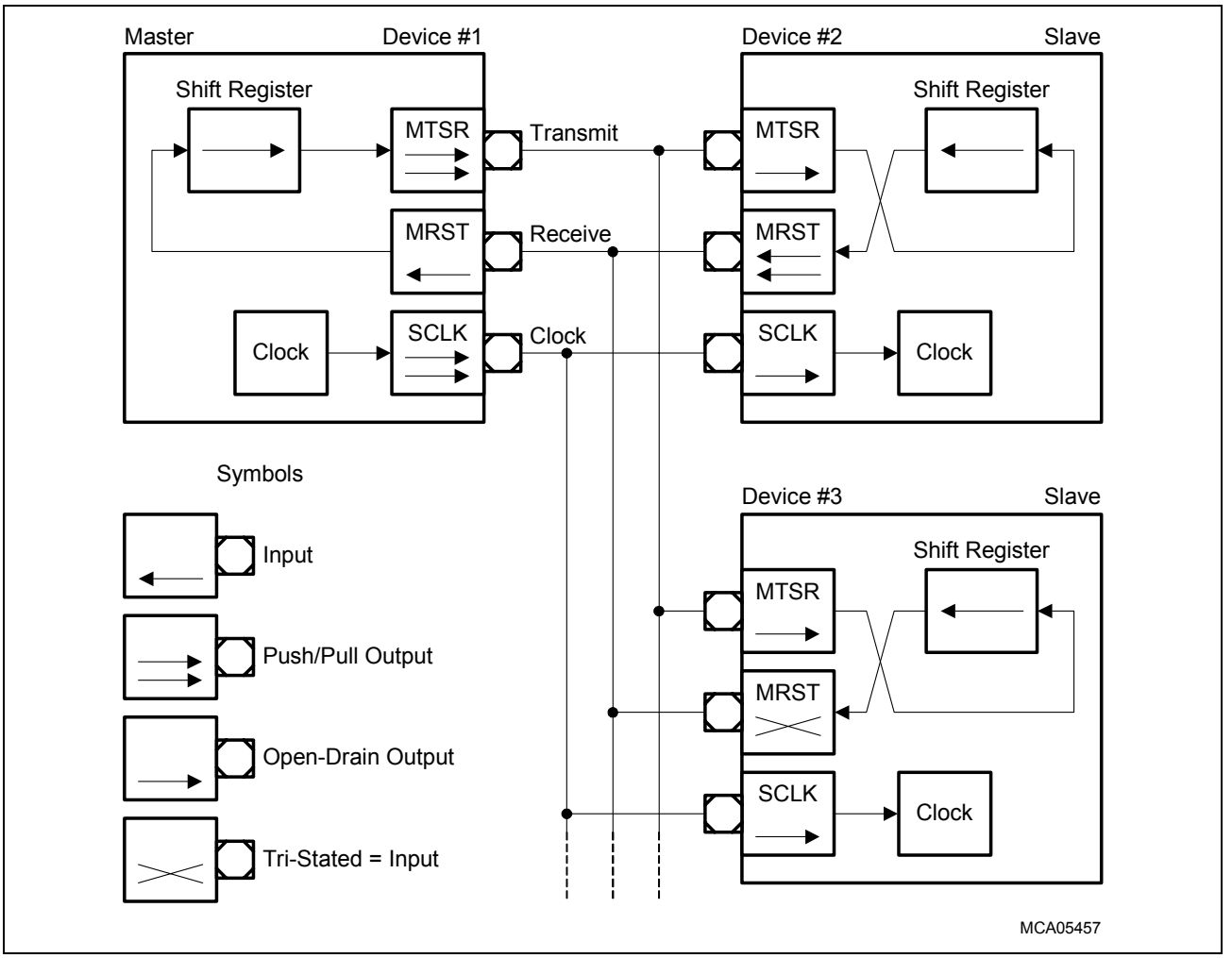


Figure 19-4 SSC Full-Duplex Configuration

The data output pins MRST of all slave devices are connected together onto the one receive line in the configuration shown in **Figure 19-4**. 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 receiving 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.
  In the configuration depicted in **Figure 19-4**, Device #2 is the slave which has its output driver enabled as push/pull output. Device #3 is an inactive slave, it needs to disable its output driver by programming the pin to input mode.
- The slaves use their MRST outputs in open-drain mode. 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 only send ones (1). 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 MSCLK will now go to its programmed polarity. The output data line MTX will go to either 0 or 1 until the first transfer will start. After a transfer, the data line MTX 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 SSCx_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 transmit line MTSR on the next clock from the baudrate generator (transmission starts only if bit EN = 1). Depending on the selected clock phase, a clock pulse will also be generated on the SCLK line. At the same time, with the opposite clock edge, the master latches and shifts in the data detected at its input line MRST. 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. 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 SSCx_RB and the receive interrupt line RIRQ is activated.

A slave device will immediately output the selected first bit (MSB or LSB of the transfer data) at line MRST when the contents of the transmit buffer are copied into the slave’s shift register. Bit BSY is not set until the first clock edge at SCLK appears. The slave device will not wait for the next clock from the baudrate generator, as the master does. The reason for this is that, depending on the selected clock phase, the first clock edge generated by the master may already be used to clock in the first data bit. Thus, the slave's first data bit must already be valid at this time.

*Note: On the SSC, a transmission **and** a reception takes 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 shall be selected in the control register SSCx_CON and the alternate output be prepared via the related ALTSEL register, or the output latch must be loaded with the clock idle level.

19.2.3 Half-Duplex Operation

In a Half-Duplex configuration, 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 the one data exchange line, serial data may be moved between arbitrary stations.

Similar to 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 outputs and send only ones (1s).

Because 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.

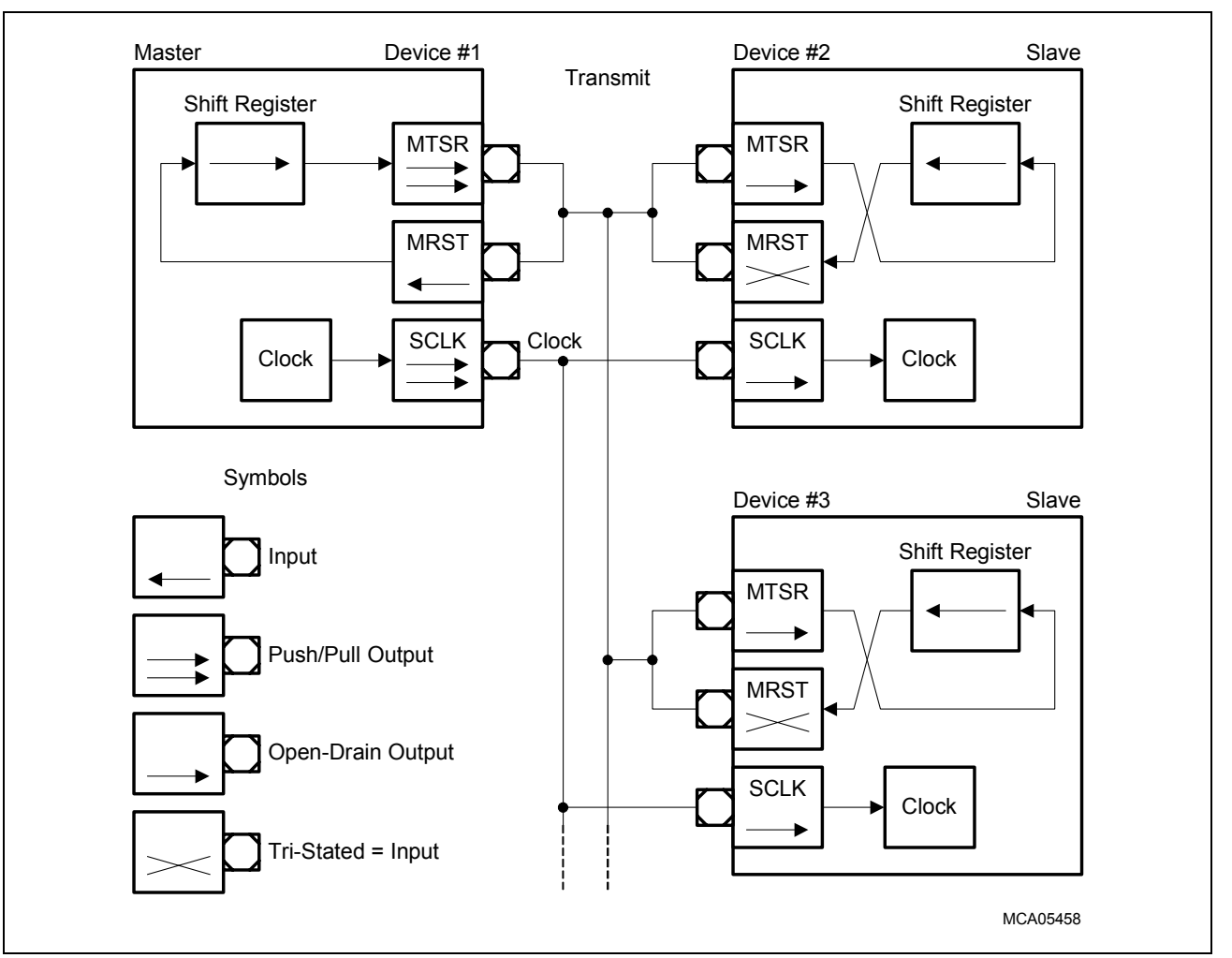


Figure 19-5 SSC Half-Duplex Configuration

19.2.4 Continuous Transfers

When the transmit interrupt request flag is set, it indicates that the transmit buffer SSCx_TB is empty and ready to be loaded with the next transmit data. If SSCx_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 8-bit transfers would look the same as one 16-bit transfer. This feature can be used to interface with devices that can operate with or require more than 16 data bits per transfer. It is just a matter of software, how long a total data frame length can be. This option can also be used to interface to byte-wide and word-wide devices on the same serial bus, for instance.

Note: Of course, this can happen only in 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.

19.2.5 Baudrate Generation

The serial channel SSC has its own dedicated 16-bit Baudrate Generator with 16-bit reload capability, facilitating baudrate generation independent of the timers. **Figure 19-6** shows the baudrate generator of the SSC in more detail.

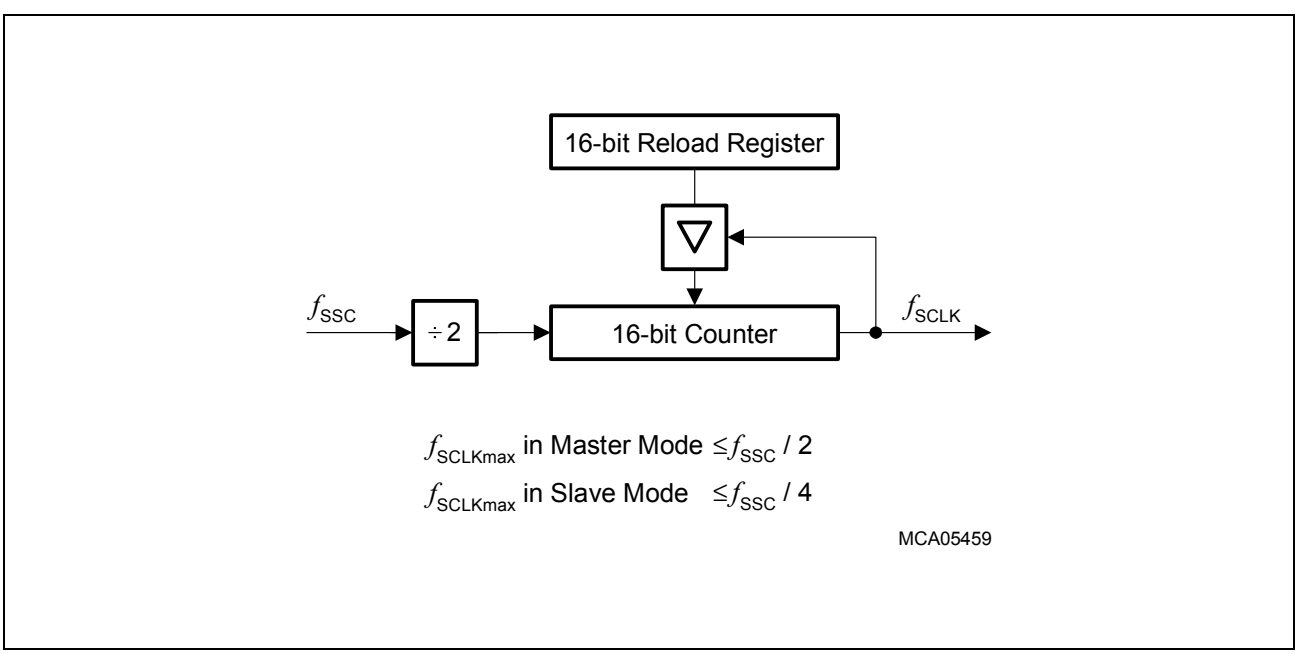


Figure 19-6 SSC Baudrate Generator

The Baudrate Generator is clocked with the module clock f_{SSC}. The counter counts downwards. Access to the Baudrate Generator is performed via one register, SSCx_BR, described below.

Baudrate Timer/Reload Register

The SSC Baudrate Timer/Reload Register SSCx_BR has a double function.

While the SSC is disabled, it serves as the reload register for the baudrate timer. Writing to it loads the timer reload register with the written reload value. Reading returns the current reload value.

While the SSC is enabled, this register reflects the current baudrate timer contents. Writing to this register is not allowed while the SSC is enabled.

Baudrate Calculation

The timer is loaded with the reload value and starts counting immediately when the SSC is enabled. The formulas below calculate either the resulting baudrate for a given reload value, or the required reload value for a given baudrate:

$$\text{Baudrate} = \frac{f_{SSC}}{2 \times (
 + 1)} \qquad BG = \frac{f_{SSC}}{2 \times \text{Baudrate}} - 1 \tag{19.1}$$

 represents the contents of the reload register, taken as unsigned 16-bit integer; while baudrate is equal to f_{SCLK} as shown in **Figure 19-6**.

The maximum baudrate that can be achieved when using a module clock of 40 MHz is 20 Mbit/s in Master Mode (with
 = 0000_H) or 10 Mbit/s in Slave Mode (with
 = 0001_H).

Table 19-1 lists some possible baudrates together with the required reload values and the resulting bit times, assuming a module clock of 40 MHz.

Table 19-1 Typical Baudrates of the SSC (f_{SSC} = 40 MHz)

Reload Value	Baudrate (= f_{SCLK})	Deviation
0000_H	20 Mbit/s (only in Master Mode)	0.0%
0001_H	10 Mbit/s	0.0%
0009_H	2 Mbit/s	0.0%
0013_H	1 Mbit/s	0.0%
$001A_H$	750 kbit/s	-1.25%
0027_H	500 kbit/s	0.0%
0063_H	200 kbit/s	0.0%
$00C7_H$	100 kbit/s	0.0%
$FFFF_H$	306.6 bit/s	0.0%

19.2.6 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 Baudrate Error only apply to Slave Mode. When an error is detected, the respective error flag in register SSCx_CON is set and an error interrupt request will be generated by activating the EIRQ line (see **Figure 19-7**). 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 some error conditions via interrupt, while the others may be polled by software.

Note: The error interrupt handler must clear the associated (enabled) error flag(s) to prevent repeated interrupt requests.

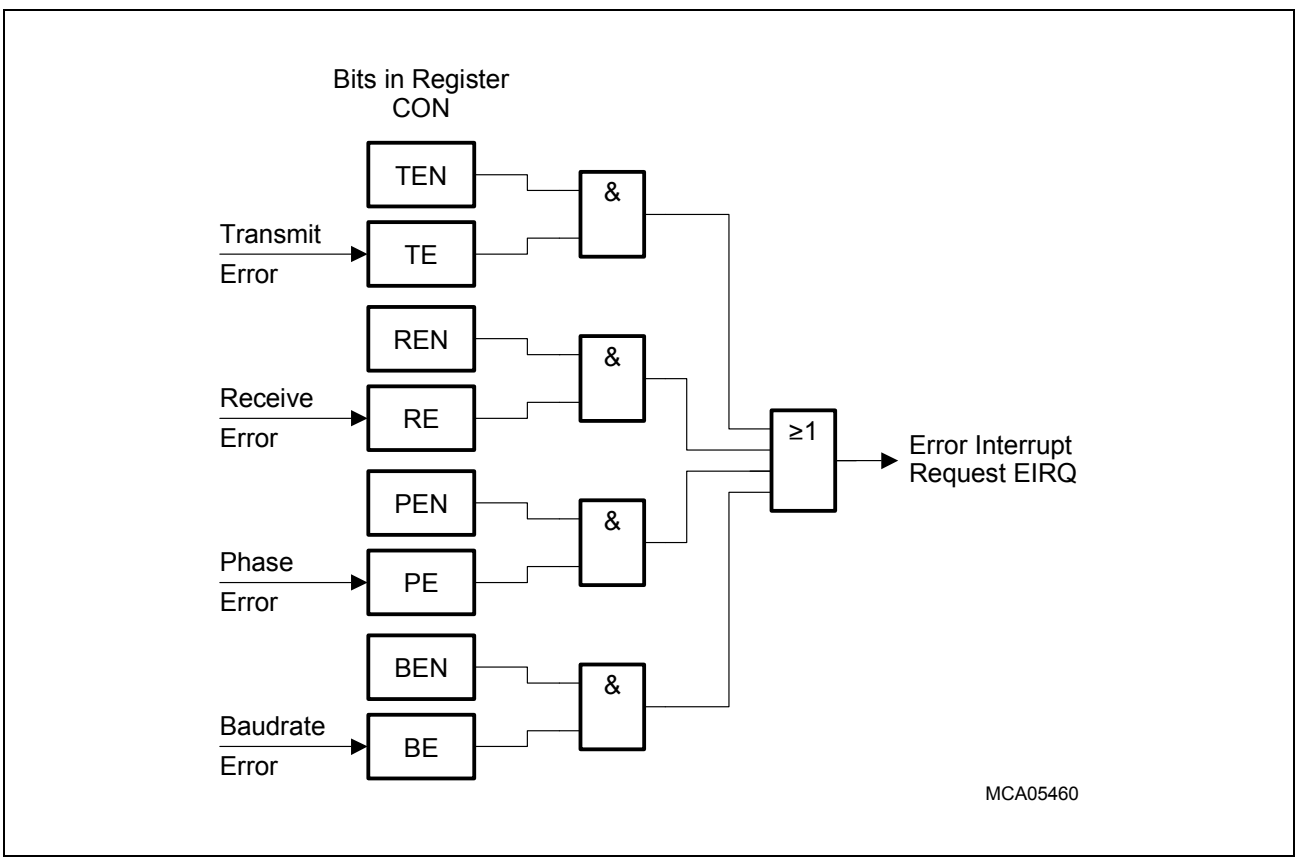


Figure 19-7 SSC Error Interrupt Control

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 receive buffer register SSCx_RB. This condition sets the error flag RE and, when enabled via bit REN, the error interrupt request line EIRQ. The old data in the receive buffer SSCx_RB will be overwritten with the new value and is irretrievably lost.

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 PE and, when enabled via bit PEN, the error interrupt request line EIRQ.

A **Baudrate Error** (Slave Mode) is detected when the incoming clock signal deviates from the programmed baudrate by more than 100%, i.e. it either is more than double or less than half the expected baudrate. This condition sets the error flag BE and, when enabled via bit BEN, the error interrupt request line EIRQ. Using this error detection capability requires that the slave's baudrate generator is programmed to the same baudrate 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 AREN = 1, an automatic reset of the SSC will be performed in case of this error. This is done to re-initialize the SSC if too few or too many clock pulses have been detected.

A **Transmit Error** (Slave Mode) is detected when a transfer was initiated by the master (SCLK gets active), but the transmit buffer SSCx_TB of the slave was not updated since the last transfer. This condition sets the error flag TE and, when enabled via bit TEN, the error interrupt request line EIRQ. If a transfer starts while the transmit buffer is not updated, the slave will shift out the 'old' contents of the shift register, which usually 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 usually 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, baudrate, transmit error) can be identified by the error status flags in control register SSCx_CON.

Note: The error status flags TE, RE, PE, and BE, are not reset automatically upon entry into the error interrupt service routine, but must be cleared by software.

19.2.7 SSC Register Summary

Table 19-2 SSC Module Register Summary

Name	Description	SSC0 Addresses		Reg. Area	SSC1 Addresses	
		16-Bit	8-Bit		16-Bit	8-Bit
SSCx_CON	Control Register	$FFB2_H$	$D9_H$	SFR	$FF5E_H$	AF_H
SSCx_BR	Baudrate Timer Reload Register	$F0B4_H$	$5A_H$	ESFR	$F05E_H$	$2F_H$
SSCx_TB	Transmit Buffer Register	$F0B0_H$	58_H	ESFR	$F05A_H$	$2D_H$
SSCx_RB	Receive Buffer Register	$F0B2_H$	59_H	ESFR	$F05C_H$	$2E_H$
SSCx_TIC	Transmit Interrupt Control Register	$FF72_H$	$B9_H$	SFR/ ESFR	$F1AA_H$	$D5_H$
SSCx_RIC	Receive Interrupt Control Register	$FF74_H$	BA_H	SFR/ ESFR	$F1AC_H$	$D6_H$
SSCx_EIC	Error Interrupt Control Register	$FF76_H$	BB_H	SFR/ ESFR	$F1AE_H$	$D7_H$

19.2.8 Port Configuration Requirements

Table 19-3 shows the required register setting to configure the IO lines of the SSC modules for master or slave mode operation.

Table 19-3 SSC0/SSC1 IO Selection and Setup

Module	Mode	Port Lines	Alternate Select Register	Direction and Port Output Register	IO
SSC0	Master	P3.8 / MRST0	ALTSEL0P3.P8 = 1	DP3.P8 = 0	Input
		P3.9 / MTSR0	ALTSEL0P3.P9 = 1	DP3.P9 = 1 and P3.P9 = 1	Output
		P3.13 / SCLK0	ALTSEL0P3.P13 = 1	DP3.P13 = 1 and P3.P13 = 1	Output
	Slave	P3.8 / MRST0	ALTSEL0P3.P8 = 1	DP3.P8 = 1 and P3.P8 = 1	Output
		P3.9 / MTSR0	ALTSEL0P3.P9 = 1	DP3.P9 = 0	Input
		P3.13 / SCLK0	ALTSEL0P3.P13 = 1	DP3.P13 = 0	Input
SSC1	Master	P1H.1 / MRST1	ALTSEL0P1H.P1 = 1	DP1H.P1 = 0	Input
		P1H.2 / MTSR1	ALTSEL0P1H.P2 = 1	DP1H.P2 = 1	Output
		P1H.3 / SCLK1	ALTSEL0P1H.P3 = 1	DP1H.P3 = 1	Output
	Slave	P1H.1 / MRST1	ALTSEL0P1H.P1 = 1	DP1H.P1 = 1	Output
		P1H.2 / MTSR1	ALTSEL0P1H.P2 = 1	DP1H.P2 = 0	Input
		P1H.3 / SCLK1	ALTSEL0P1H.P3 = 1	DP1H.P3 = 0	Input

Note: The direction control bits in registers DP3 or DP1H must be set or cleared by software depending on the mode of operation selected (master or slave mode). They are not controlled automatically by the SSC modules.

19.3 Interfaces of the SSC Modules

In the XC161 the SSC modules are connected to IO ports and other internal modules according to **Figure 19-8** and **Figure 19-9**.

The input/output lines of SSC0 are connected to pins of Ports P3, while the input/output lines of SSC1 are connected to pins of Ports P1H. The three interrupt request lines of each module are connected to the Interrupt Control Block.

Clock control and emulation control of the SSC Module is handled by the System Control Unit, SCU.

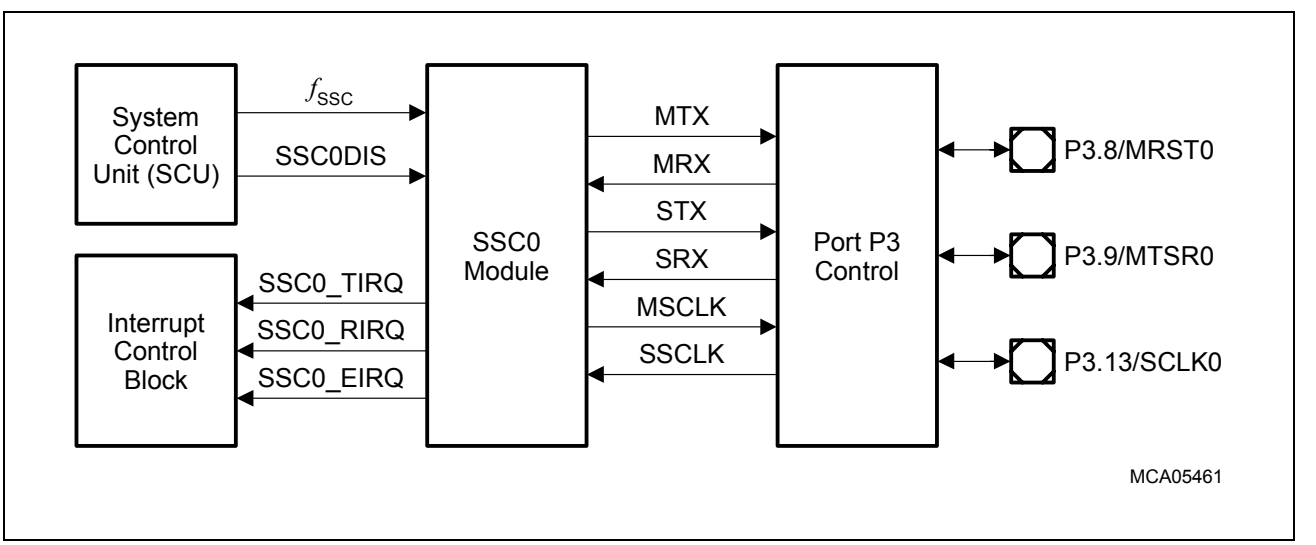


Figure 19-8 SSC0 Module Interfaces

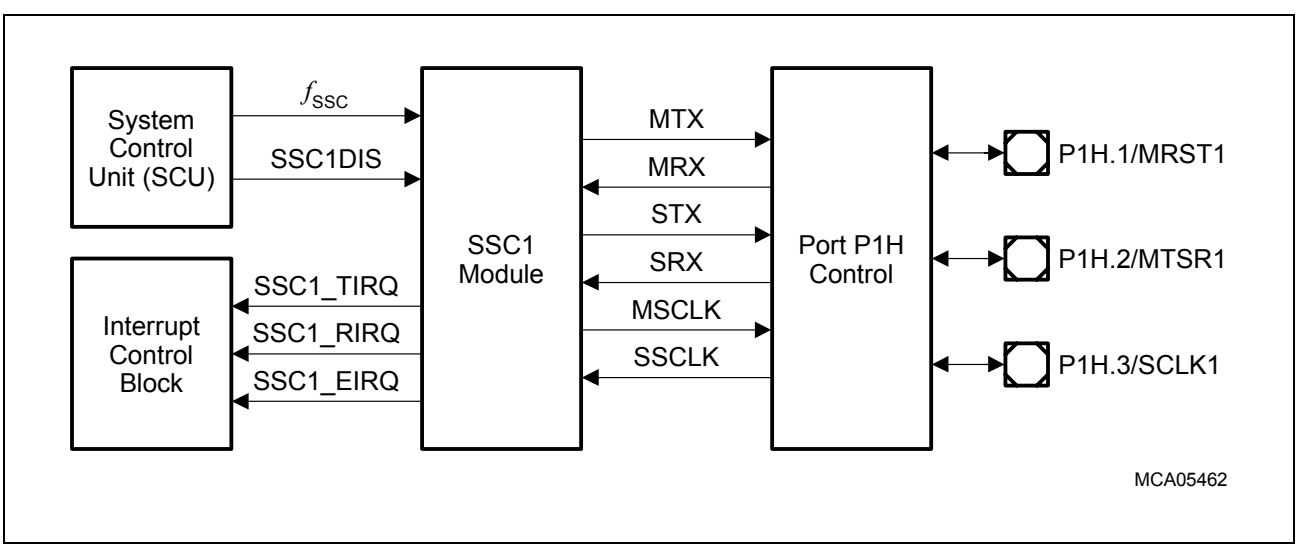


Figure 19-9 SSC1 Module Interfaces

20 IIC-Bus Module

The IIC-Bus supports a defined protocol to enable devices to communicate directly with each other via a simple two-wire serial interface. One line is responsible for clock transfer and synchronization (SCL), the other is responsible for the data transfer (SDA).

The on-chip IIC-Bus Module connects the XC161 to other external controllers and/or peripherals via the two-line serial IIC-Bus interface. The IIC-Bus Module provides communication at data rates of up to 400 kbit/s and features 7-bit addressing as well as 10-bit addressing. This module is fully compatible to the IIC bus protocol.

The module can operate in three different modes:

Master mode, where the IIC-Bus Module controls the bus transactions and provides the clock signal.

Slave mode, where an external master controls the bus transactions and provides the clock signal.

Multimaster mode, where several masters can be connected to the bus, i.e. the IIC-Bus Module can be master or slave.

The on-chip IIC-Bus Module allows efficient communication via the common IIC-Bus. The module unloads the CPU of low level tasks like

- Serialization/De-serialization of bus data
- Generation of start and stop conditions
- Monitoring of the bus lines
- Evaluation of the device address in slave mode
- Bus access arbitration in multi-master mode

Features

- Extended buffer allows up to 4 transmit/receive data bytes to be stored
- Selectable baudrate generation
- Support of standard 100 kbit/s and extended 400 kbit/s data rates
- Operation in 7-bit addressing or 10-bit addressing mode
- Flexible control via interrupt service routines or by polling
- Dynamic access to up to 3 physical IIC buses

Applications

- EEPROMs
- 7-Segment Displays
- Keyboard Controllers
- On-Screen Display
- Audio Processors

20.1 Overview

A block diagram of the XC161 IIC-Bus Module is shown in Figure 20-1, while Figure 20-2 illustrates a possible serial interface system.

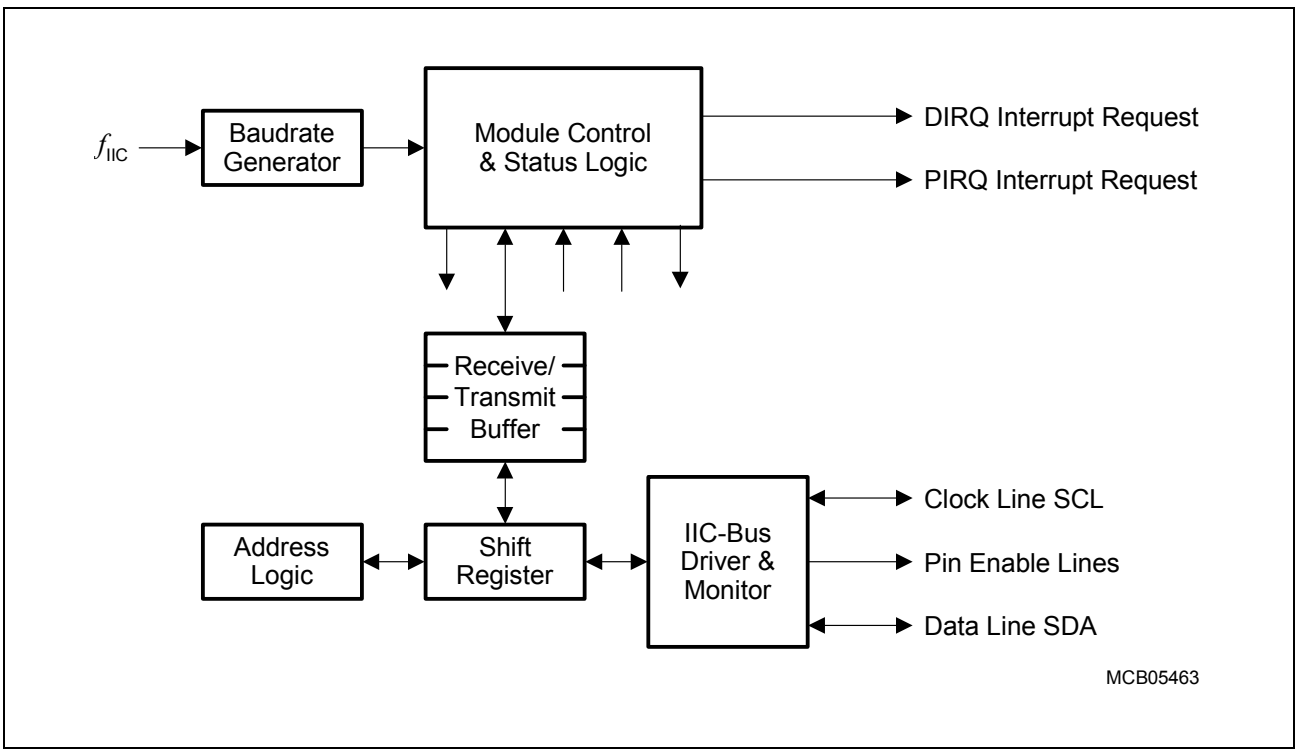


Figure 20-1 IIC-Bus Module Block Diagram

The IIC-Bus Module has its own flexible Baudrate Generator. A 4-byte Receive/Transmit Buffer enables software to write or read longer message and eliminates the need to react after each received/transmitted byte. Serialization and de-serialization of the byte data is performed via an 8-bit Shift Register. The Address Logic analyzes the received slave address and informs the Control Logic when the device has been contacted by another station in the system. The Control and Status Logic controls the entire module and provides a number of status signals and flags, reflecting the conditions of the module to the software.

To operate in an IIC-Bus system, it is not only necessary for a station to be able to drive the clock and data lines of the IIC-Bus, but also to monitor the actual levels on these lines and to detect special conditions, such as the start and stop conditions, and to perform clock synchronization as well as bus arbitration. This is handled by the IIC-Bus Driver and Monitor block. In addition, this block provides the port pin enable control for the three possible SCL/SDA signal pairs.

Due to the feature that the IIC-Bus Module of the XC161 can control up to three SCL/SDA signal pairs, it is possible to build a system with separate IIC-buses as shown in Figure 20-2.

Note: Per definition, an IIC-Bus system is a Wired-AND configuration. The active (dominant) level is the low level, while the high level is not actively driven by the stations (or nodes), but held through external pull-up devices. For this purpose, the respective pin drivers must be switched to open drain mode.

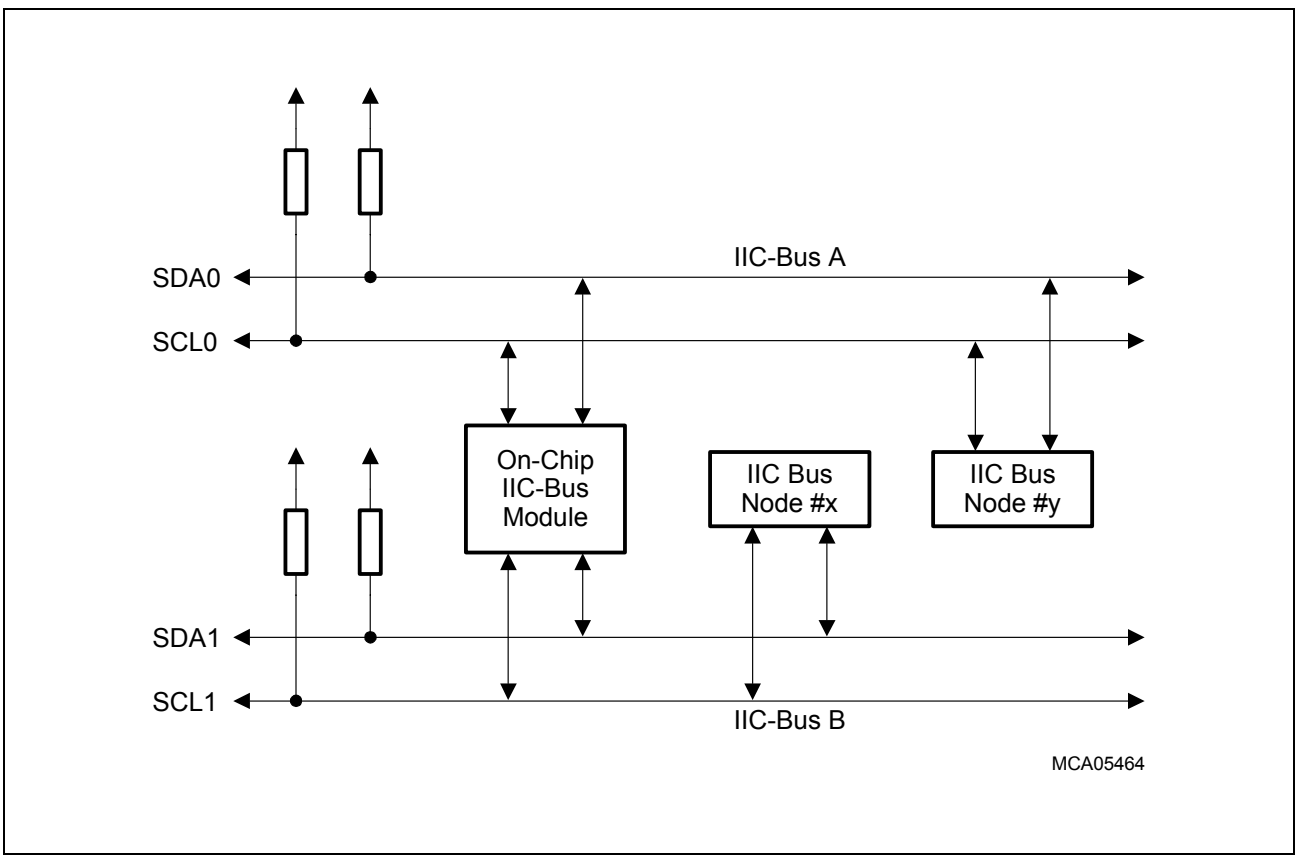


Figure 20-2 IIC-Bus Configuration Example

In an IIC-Bus system, a station may be able to play different roles: Master-Transmitter (a master device which is sending data to one or more slaves), Master-Receiver (a master which is receiving data from a slave), Slave-Transmitter (a slave which is sending data to a master) and Slave-Receiver.

From the programmer's point of view, the term 'IIC-Bus Module' refers to a set of registers which are associated with this peripheral, including the port pins which may be used for alternate input/output functions, and including their direction control bits.

Figure 20-3 shows the Special Function registers (SFRs) associated with the IIC-Bus Module.

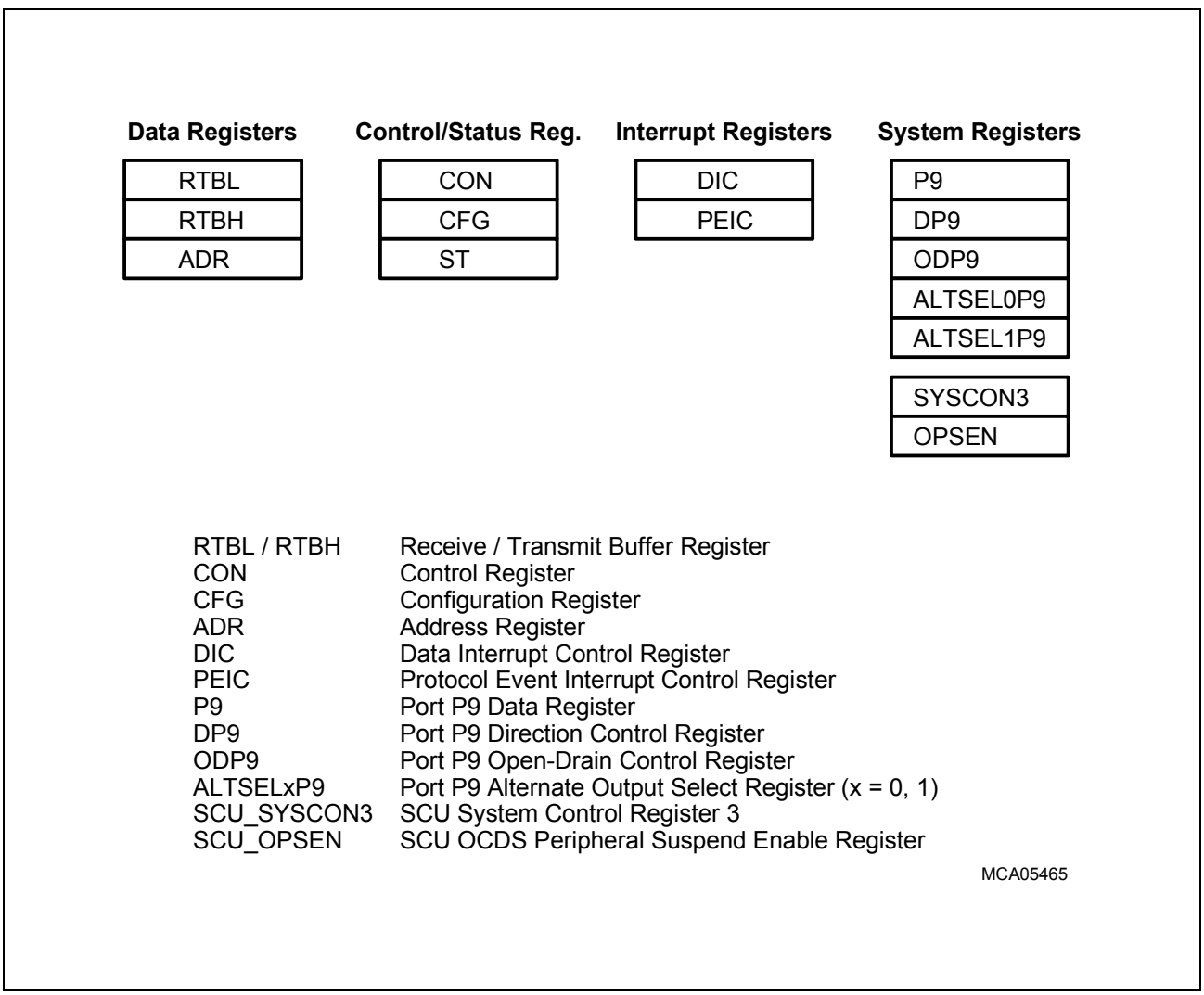


Figure 20-3 SFRs Associated with the IIC-Bus Module

20.2 Register Description

In the following, the registers of the IIC-Bus Module are described in detail.

IIC_CON
Control Register **XSFR ($E602_H$/--)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	CI		STP	IGE	TRX	INT	ACK DIS	BUM	MOD		RSC	M10
-	-	-	-	rw		rwh	rw	rwh	rw	rwh	rwh	rw		rwh	rw

Field	Bits	Type	Description
CI	[11:10]	rw	**Transmit Buffer Length Control** 00 1 byte (RTB0) 01 2 bytes (RTB1 … RTB0) 10 3 bytes (RTB2 … RTB0) 11 4 bytes (RTB3 … RTB0)
STP	9	rwh	**Master Stop Control** 0 No action 1 Setting bit STP generates a stop condition after the next transmission. Bit BUM is cleared. *Note: STP is automatically cleared by a stop condition.*
IGE	8	rw	**Ignore End-of-Transmission (IRQE) Interrupt** 0 The IIC is stopped at IRQE interrupt 1 The IIC ignores the IRQE interrupt
TRX	7	rwh	**Transmit Select** 0 No data is transmitted to the IIC bus 1 Data is transmitted to the IIC bus *Note: TRX is set automatically when writing to the transmit buffer. TRX is automatically cleared after the last byte as a slave transmitter.*
INT	6	rw	**Interrupt Flag Clear Control** 0 Interrupt flag IRQD is cleared by a read/write access to RTB0 … 3 1 Interrupt flag IRQD is not cleared by a read/write access to RTB0 … 3

Field	Bits	Type	Description
ACKDIS	5	rwh	**Acknowledge Pulse Disable** 0 An acknowledge pulse is generated for each received byte 1 No acknowledge pulse is generated *Note: ACKDIS is automatically cleared by a stop condition.*
BUM	4	rwh	**Busy Master** 0 Clearing bit BUM immediately generates a stop condition 1 Setting bit BUM generates a start condition in (multi-) Master mode *Note: Setting bit BUM while the bus is busy (BB = 1) generates an arbitration lost situation. In this case, BUM is cleared and bit AL is set. BUM cannot be set in slave mode.*
MOD	[3:2]	rw	**Basic Operating Mode** 00 IIC module is disabled and initialized (Init-Mode). Transmissions in progress will be aborted. 01 Slave mode 10 Single-Master mode 11 Multi-Master mode
RSC	1	rwh	**Repeated Start Condition Trigger** 0 No operation 1 Generate a repeated start condition in (multi-) master mode. RSC cannot be set in slave mode. *Note: RSC is cleared automatically after the repeated start condition has been sent.*
M10	0	rw	**Slave Address Width Selection** 0 7-bit slave address, using ICA[7:1] 1 10-bit slave address, using ICA[9:0]

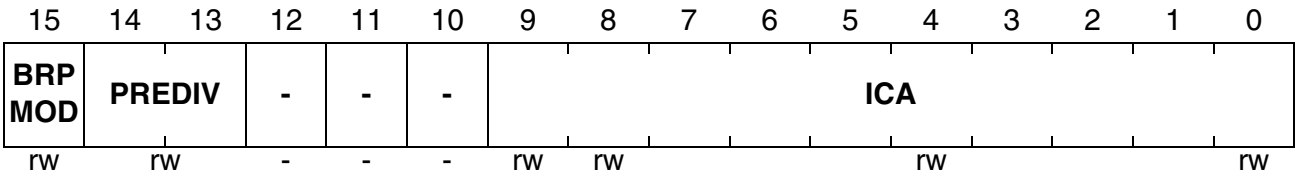
IIC_ST
Status Register **XSFR ($E604_H$/--)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	CO			IRQE	IRQP	IRQD	BB	LRB	SLA	AL	ADR
-	-	-	-	-	rh			rwh	rwh	rwh	rh	rh	rh	rwh	rh

Field	Bits	Type	Description
CO	[10:8]	rh	**Transmit Byte Counter** Displays the number of correctly transferred bytes. See **Section 20.3.4** for details. 000 0 bytes 001 1 byte 010 2 bytes 011 3 bytes 100 4 bytes 1xx Reserved
IRQE	7	rwh	**End-of-Data-Transmission Interrupt Req. Flag** 0 No interrupt request pending 1 An End-Of-Data-Transmission interrupt request is pending See **Section 20.4** for details.
IRQP	6	rwh	**Protocol Event Interrupt Request Flag** 0 No interrupt request pending 1 A Protocol Event interrupt request is pending See **Section 20.4** for details.
IRQD	5	rwh	**Data Transfer Event Interrupt Request Flag** 0 No interrupt request pending 1 A Data Transfer Event interrupt request is pending See **Section 20.4** for details.
BB	4	rh	**Bus Busy Flag** 0 The IIC-Bus is idle 1 The IIC-Bus is busy *Note: Bit BB is always 0 while the IIC module is disabled.*

Field	Bits	Type	Description
LRB	3	rh	**Last Received Bit** Bit LRB represents the last bit (i.e. the acknowledge bit) of the last transferred byte. It is automatically cleared by a read/write access to the buffer RTB0 … 3. *Note: If LRB is high (no acknowledge) in slave mode, bit TRX is set automatically to select slave transmit mode.*
SLA	2	rh	**Slave Select Flag** 0 The IIC-Bus Module is not addressed in Slave mode, or the module is in Master mode. 1 The IIC-Bus Module has been addressed as a slave (own slave address or general address, 00_H, was received).
AL	1	rwh	**Arbitration Lost Flag** Bit AL is set when the IIC-Bus Module has tried to become master on the bus but has lost arbitration. Operation is continued until the 9^{th} clock pulse. If multi-master mode is selected, the IIC module temporarily switches to Slave mode after a lost arbitration. Bit IRQP is set along with bit AL. AL must be cleared via software.
ADR	0	rh	**Address Phase Flag** Bit ADR is set after a start condition in Slave mode until the complete address has been received (1 byte in 7-bit address mode, 2 bytes in 10-bit address mode).

IIC_ADR
Address Control Register **XSFR ($E606_H$/--)** **Reset Value: 0000_H**



Field	Bits	Type	Description
BRPMOD	15	rw	**Baudrate Generator Mode Control** 0 Mode 0: Reciprocal Divider 1 Mode 1: Fractional Divider
PREDIV	[14:13]	rw	**Pre-Divider for Baudrate Generation** 00 Pre-divider is disabled 01 Pre-divider factor is 8 10 Pre-divider factor is 64 11 Reserved, do not use
ICA	[9:0]	rw	**Own Slave Address** Specifies the slave address of the IIC-Bus module 7-bit address mode (CON.M10 = 0): address stored in ICA[7:1] (ICA[9:8] and ICA[0] are read-only, read as 0) 10-bit address mode (CON.M10 = 1): address stored in ICA[9:0]

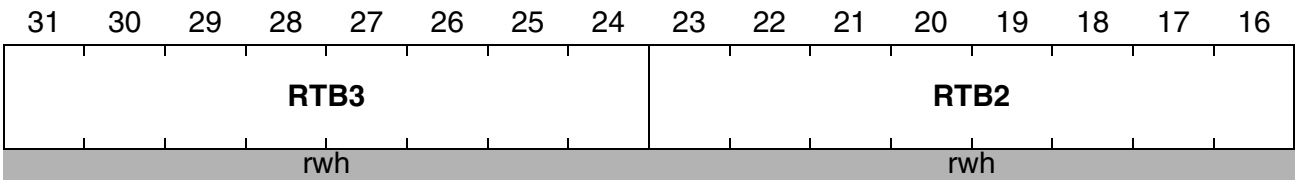
IIC_CFG
Configuration Control Register XSFR ($E600_H$/--) Reset Value: 0000_H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BRP								-	SCL EN2	SCL EN1	SCL EN0	-	SDA EN2	SDA EN1	SDA EN0
rw								-	rw	rw	rw	-	rw	rw	rw

Field	Bits	Type	Description
BRP	[15:8]	rw	**Baudrate Prescaler Value** Determines the baudrate for the IIC-Bus module together with bit ADR.BRPMOD and bitfield ADR.PREDIV
SCLENx (x = 2 … 0)	6, 5, 4	rw	**Enable Bit for SCLx Clock Line** These bits determine to which pins the IIC clock line is connected. 0 SCLx pin is disconnected 1 SCLx pin is connected with IIC clock line
SDAENx (x = 2 … 0)	2, 1, 0	rw	**Enable Bit for SDAx Data Line** These bits determine to which pins the IIC data line is connected. 0 SDAx pin is disconnected 1 SDAx pin is connected with IIC data line

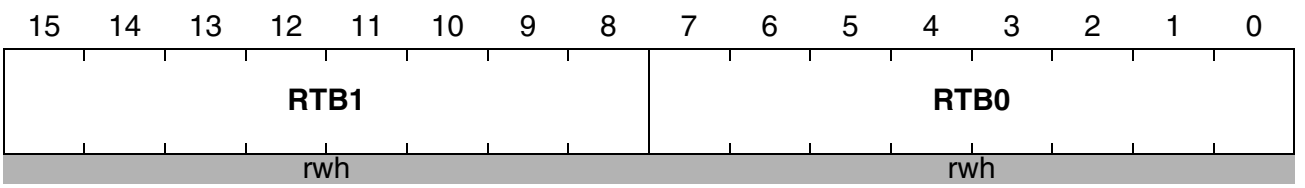
IIC_RTBH

Receive/Transmit Buffer High **XSFR ($E60A_H$/--)** **Reset Value: 0000_H**



IIC_RTBL

Receive/Transmit Buffer Low **XSFR ($E608_H$/--)** **Reset Value: 0000_H**



Field	Bits	Type	Description
RTBx (x = 3 … 0)	[31:24], [23:16], [15:8], [7:0]	rwh	**Receive/Transmit Buffer Bytes** The buffers contain the data to be sent or which have been received. The buffer size can be selected via bitfield CI, from 1 up to 4 bytes. The contents of RTB0 are sent/received first.

Note: If bit INT is set to zero and all bytes (specified in CI) of RTB0 … 3 are read/written (dependent on bit TRX), IRQD will be cleared by hardware after completion of this access (this supports PEC operation).

20.3 IIC-Bus Module Operation

The following sections describe the operation of the IIC-Bus Module in the three different modes. In addition, detailed information on the Receive/Transmit Buffer as well as the Baudrate Generator is provided.

20.3.1 Operation in Single-Master Mode

In Single-Master Mode, the IIC-Bus Module of the XC161 is the only master controlling the external IIC-Bus, thus, the master can always assume that the bus is free to use. Under normal conditions, there is no possibility for this master to loose arbitration.

Software initializes the IIC-Bus Module according to the master operation. There is no need to specify an own slave address in register ADR, as the master can never be addressed by another station.

To start a transfer, the master first writes the address of the slave to be contacted (or the general call address to access all stations) into the receive/transmit buffer. In 7-bit address mode, the address is written to bitfield RTB0, bits [7:1]. In 10-bit address mode, the address is written to bitfields RTB0 and RTB1. Bit 0 of RTB0 is the read/write bit R/$\overline{W}$, which informs the slave whether the master wants to read from or write to the slave.

Then the master sets bit BUM in register CON. This generates a start condition on the bus, the busy bit BB is set, and the transmission of the buffer contents begins.

To start a new transfer or to change the transfer direction, the master can generate a repeated start condition. This eliminates the need to first stop bus transactions, and then start again. The repeated start is performed by setting bit RSC in register CON. The busy bit BB remains set. Bit RSC is cleared automatically after the repeated start condition has been generated.

When the master is finished with the current bus transaction, it generates a stop condition on the bus by clearing bit BUM.

20.3.2 Operation in Multimaster Mode

In Multi-Master Mode, the XC161 is not the only master on the bus and must share IIC-Bus usage with other masters. This requires bus arbitration, as only one master may control the bus at a given time.

Thus, when a master tries to take control of the IIC-Bus, it might be that the bus is already in use or that another master is trying to claim the bus at the same time. To detect such situations, each master monitors the bus activity by comparing the level which it wants to output onto the SDA line with the level it reads from the external SDA line. If it finds the case that it wants to output a high level (inactively driven by the master, but usually held through external pull-up devices), but the actual level on the SDA line is a low level, then it recognizes this case as an ‘arbitration lost’ condition, and it needs to backoff.

It is not only necessary for the loosing master to release the bus in order to allow the other master to control the bus, but it also needs to receive the message from the other master, as it might be addressed as a slave.

When the XC161 wants to use the IIC-Bus, it prepares to start a transfer as in Single-Master Mode. The next recommended step is to poll bit BB to check whether the bus is busy. If BB = 0, then the start condition can be generated by setting bit BUM. If the bus is still free after that, operation continues as in Single-Master Mode. If the bus is already in use, indicated by BB = 1, the master can not take control of the bus and needs to act as a slave (acting as a slave is automatically done in hardware); bit BUM should not be set in this case. If bit BUM is set although the bus is already in use, the Arbitration Lost flag AL is set.

However, if testing bit BB showed that the bus is free, and software sets the BUM bit, but at the same time another master tries to get onto the bus, the bus arbitration needs to take place. This is performed such that the master which first detects a mismatch between its intended output level and the actual level on the SDA line looses the arbitration. The Arbitration Lost flag AL is set in this case, the Transmit Selection bit TRX is cleared to 0 (= reception), and the master automatically switches to slave mode to receive the address information. At the end of the address phase, hardware automatically compares the received address with the own station address stored in register ADR. If the two addresses match or if the general call address (00_H) has been received, the Slave Select flag SLA in register ST is set to indicate that the device has been contacted. Operation is then continued as described in Slave Mode.

Together with bit AL, the Protocol Event interrupt flag IRQP is set, and the respective interrupt request line is activated.

Note that a master which has lost arbitration has written its transmit message to the receive/transmit buffer before it has tried to take control of the IIC-Bus. However, after it has lost arbitration, it has switched to Slave Mode, and was therefore receiving the message sent over the bus. This message is then stored in the receive/transmit buffer, overwriting the previous transmit message.

Due to the fact that a master must also act as a slave in a multi-master system, the actual implicit default operating mode in Multi-Master Mode is Slave Mode.

20.3.3 Operation in Slave Mode

When the XC161 is intended to purely operate as a slave on the IIC-Bus, Slave Mode needs to be selected via bitfield MOD in register CON.

The IIC-Bus Module is selected by another master when it receives either its own device address or the general call address during the address phase of a transmission (the byte(s) following a start or repeated start condition). If this is the case, bit SLA in register ST is set, the Protocol Event interrupt flag IRQP is set, and the respective interrupt request line is activated.

If the device has not been selected, it remains idle in Slave Mode.

If the device has been selected, the read/write bit R/$\overline{W}$, which has been received together with the address information, needs to be checked by software to determine the further actions. If this bit is 0, the slave remains in receive mode, and can read the incoming message from the buffer RTB0 … 3.

If bit R/$\overline{W}$ = 1, the master wants to read from the slave device. For this, the slave needs to prepare the data to be transferred to the master. The data is written to the buffer RTB0 … 3. Writing to the buffer automatically sets the transfer mode bit TRX to one (= transmission).

In both cases, operation can only continue when all interrupt flags, IRQD, IRQE, and IRQP, are cleared. Otherwise, the device holds the SCL clock line low to prevent further transactions on the IIC-Bus. In this way, a slave is able to suspend bus activities until it is ready to proceed.

When a stop condition or a repeated start condition is detected, bit SLA is cleared (it will be set again if the slave is contacted again at the end of the address phase of the new transaction).

20.3.4 Transmit/Receive Buffer

The IIC-Bus Module has a transmit/receive buffer which can be set to a depth of one to four bytes. Access to this buffer is performed via the two registers RTBL and RTBH, each of these represents two bytes of the buffer. The depth of the buffer is specified via bitfield CI in register CON (1, 2, 3 or 4 bytes).

For a transmission, the bytes to be transferred are written to the respective buffer bytes, and then transmission is initiated. The data interrupt IRQD is activated when all bytes of the specified buffer have been transmitted.

In receive mode, the data interrupt IRQD is activated when all bytes of the specified buffer have been filled with incoming data.

A byte counter, CO in the status register ST, counts the bytes which have been transferred from the buffer to the IIC-Bus or vice versa. The contents of this counter is especially of interest in Slave-Transmitter Mode, if the bus transactions have been terminated by an external master before all bytes of the buffer have been transmitted.Software can determine the number of correctly transmitted bytes by reading bitfield CO.

In receive mode, bitfield CO needs to be read in case the transactions have been terminated (which activates the Protocol Event interrupt request, IRQP), as it represents the number of correctly received bytes.

Bitfield CO is always cleared to 0 by the correct number (defined by bitfield CI) of read/write accesses to the buffer registers.

20.3.5 Baud Rate Generation

In order to give the user high flexibility in selection of CPU frequency and IIC-Bus baudrate without constraints to baudrate accuracy, a flexible baudrate generator has been implemented. It uses two different modes and an additional pre-divider. Low baudrates may be configured at high precision in mode 0, which is compatible with previous implementations of the IIC-Bus module. High baudrates may be configured precisely in mode 1.

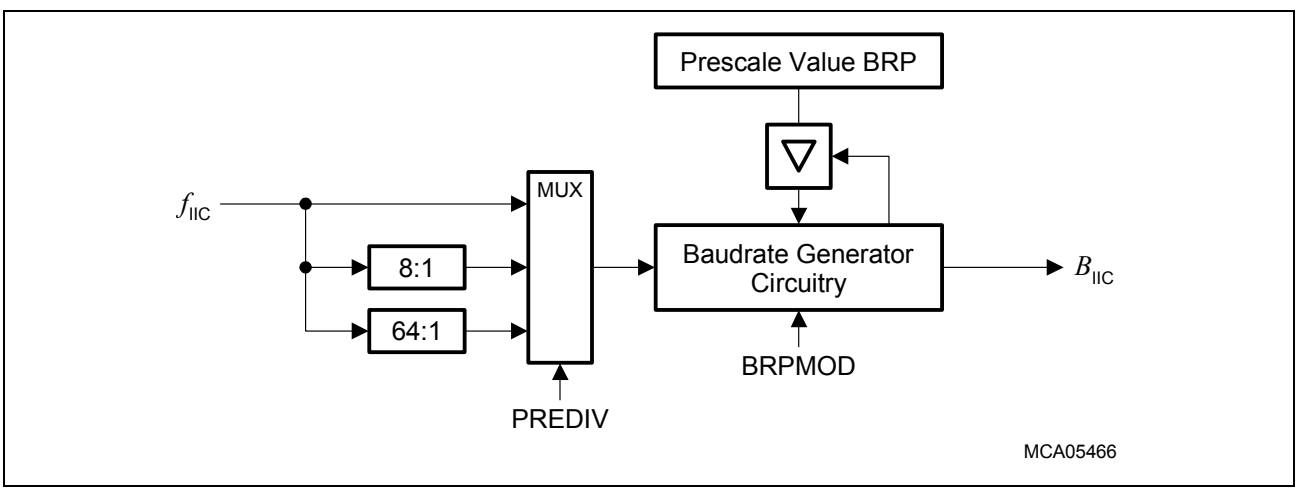


Figure 20-4 IIC-Bus Module Baudrate Generator

Reciprocal Divider Mode (BRPMOD = 0)

The resulting baudrate is:

$$B0_{IIC} = \frac{f_{IIC}}{4 \times 2^{<PREDIV> \times 3} \times (<BRP> + 1)} \qquad BRP = \frac{f_{IIC}}{4 \times 2^{<PREDIV> \times 3} \times B0_{IIC}} - 1 \qquad (20.1)$$

Table 20-1 IIC-Bus Baudrate Examples for Mode 0

BRPMOD = 0	BRP @ 100 kbit/s		BRP @ 400 kbit/s	
f_{IIC} [MHz]	PREDIV = 00_B	PREDIV = 01_B	PREDIV = 00_B	PREDIV = 01_B
40	63_H	$0B_H$	18_H	02_H
24	$3B_H$	06_H	$0E_H$	–
20	31_H	05_H	$0B_H$	–
16	27_H	04_H	09_H	–
10	18_H	02_H	05_H	–
8	13_H	01_H	04_H	–

Fractional Divider Mode (BRPMOD = 1)

The resulting baudrate is:

$$B1_{IIC} = \frac{f_{IIC} \times <BRP>}{1024 \times 2^{<PREDIV> \times 3}} \qquad BRP = \frac{1024 \times 2^{<PREDIV> \times 3} \times B1_{IIC}}{f_{IIC}} \tag{20.2}$$

Table 20-2 IIC-Bus Baudrate Examples for Mode 1

BRPMOD = 1	BRP @ 100 kbit/s		BRP @ 400 kbit/s	
f_{IIC} [MHz]	PREDIV = 00_B	PREDIV = 01_B	PREDIV = 00_B	PREDIV = 01_B
40	03_H	14_H	$0A_H$	51_H
24	04_H	22_H	11_H	88_H
20	05_H	28_H	14_H	$A4_H$
16	06_H	33_H	$1A_H$	CD_H
10	$0A_H$	51_H	29_H	–
8	$0D_H$	66_H	33_H	–

20.3.6 Notes for Programming the IIC-Bus Module

It is strictly recommended not to write to the IIC-Bus Module registers while the module is busy with transfers, except when interrupt requests have been generated.

In Master Mode (and if operating as active master in Multi-Master Mode), the module is busy as long as the BUM bit is set. In Slave Mode (and if operating as a slave in Multi-Master Mode), the module is busy from a start condition (or repeated start condition) until a stop condition is detected. This is indicated by the busy bit BB.

Access to the module's registers should only be performed after appropriate interrupt requests are generated by the module, indicating a pause in or the termination of ongoing transfers. During initialization mode (MOD = 00), all registers can be accessed freely.

A change of the transfer direction is only allowed after a protocol interrupt.

When operating as a master, software can examine the level of the acknowledge bit returned by the slave via bit LRB (Last Received Bit) in the status register ST. Note that this bit represents the acknowledge bit of the last byte which was transferred before an interrupt request was generated.

20.4 Interrupt Request Operation

The IIC-Bus Module can generate three different interrupt requests, each with its own request flag. However, due to the nature of these requests, it is sufficient to use only two interrupt nodes to process the requests. As the data interrupt request IRQD and the end-of-data-transmission interrupt request IRQE both deal with the end of a transfer of a block of data, their are combined onto one interrupt request line and node, IIC_DIRQ, as shown in **Figure 20-5**.

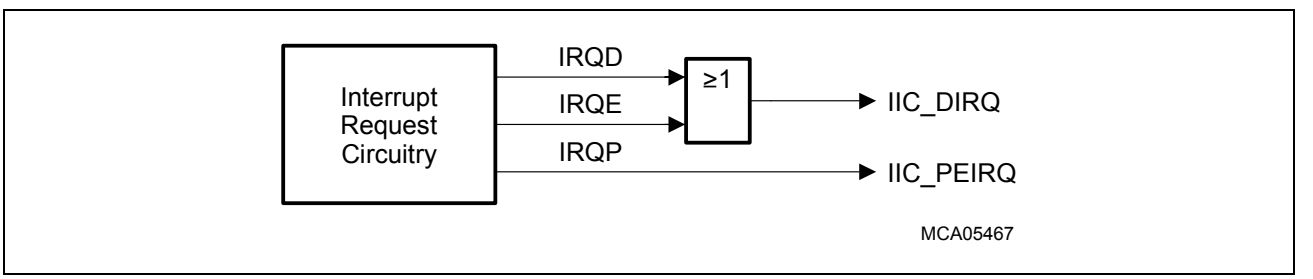


Figure 20-5 IIC-Bus Module Interrupt Wiring

The request flags for the three possible interrupt sources are located in the status register ST. The conditions for the activation of the requests and for handling of the request flags are detailed below.

As long as one or more of the interrupt request flags are set, and the IIC-Bus Module operates in Master Mode or has been selected as a slave, the clock line SCL is held at low level to prevent further transactions on the bus. The clock line is released again when all three flags are set to 0. Then, further transactions can take place on the IIC bus.

This operation can also be used to control IIC-Bus transactions by setting or clearing the request flags by software.

Data Transfer Event Interrupt, IRQD

This request is activated and the flag is set when the specified buffer is either empty (in transmit mode) or full (in receive mode). For example, when the buffer size is set to 3 bytes (via CI) and all three buffer locations, RTB0, RTB1, and RTB2, have been written with transmit values, then the request will be activated when the last byte in RTB2 has been sent via the IIC-Bus.

IRQD is also activated in Slave-Transmitter Mode, when a transfer was terminated by the current master before all data in the slave's transmit buffer has been sent. This is in addition to the activation of interrupt request IRQE.

If the automatic interrupt flag clear operation is selected (bit CON.INT = 0), then flag IRQD is automatically cleared by hardware upon a complete read or write access to the buffer(s) RTB0 … 3. If CON.INT = 1, then flag IRQD must be cleared by software.

End-of-Data-Transmission Interrupt, IRQE

This request is activated and the flag is set when the current data transfer is terminated either by a repeated start, by a stop, or by a missing acknowledge.

In the case of Slave-Transmitter Mode, additionally the Data Transfer interrupt request IRQD will be activated.

Flag IRQE must be cleared by software.

Protocol Event Interrupt, IRQP

This request is activated and the flag is set in Multi-Master mode when the module has lost arbitration. Additionally, the arbitration lost flag AL is set.

In Multi-Master and in Slave Mode, this request is activated when either the general call address or the device's own address has been received.

Flag IRQP must be cleared by software.

Interrupt Nodes

The three interrupt request lines are connected to two interrupt nodes (see **Figure 20-5**):

IIC_DIC
IIC Data Interrupt Ctrl. Reg. **ESFR ($F186_H$/$C3_H$)** **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-							GPX	ICD IR	ADC IE	ILVL				GLVL	
-	-	-	-	-	-	-	rw	rwh	rw	rw				rw	

IIC_PEIC
IIC Protocol Intr. Ctrl. Reg. **ESFR ($F18E_H$/$C7_H$)** **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-							GPX	ICP IR	ADE IE	ILVL				GLVL	
-	-	-	-	-	-	-	rw	rwh	rw	rw				rw	

Note: Please refer to the general Interrupt Control Register description for an explanation of the control fields.

20.5 Port Connection and Configuration

The IIC-Bus Module can provide up to three SCL/SDA signal pairs, which can be connected to different pins of the XC161. The individual enable control bits for these options are located in the configuration register CFG. **Figure 20-6** illustrates this feature.

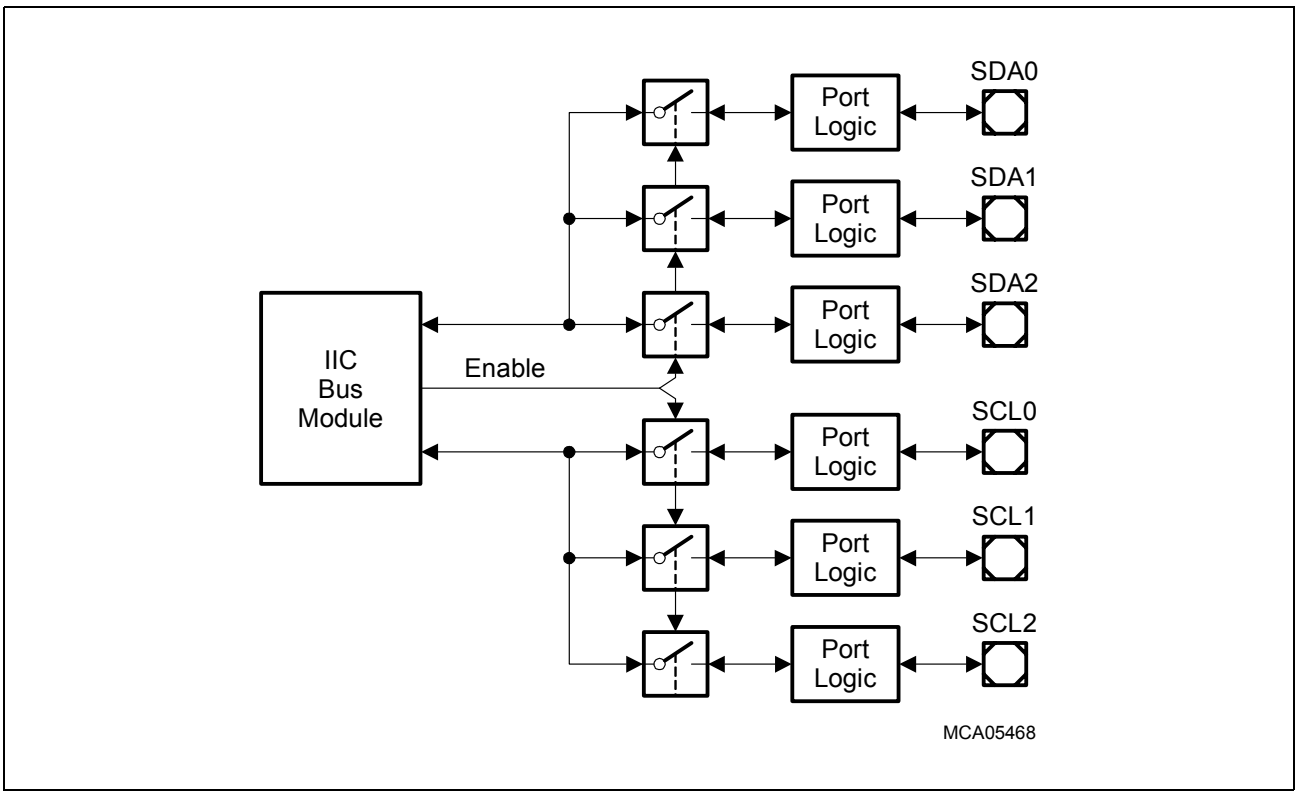


Figure 20-6 IIC-Bus Module Port Pin Connection Options

Pin Configuration

Due to the Wired-AND configuration of an IIC-Bus system, the port drivers for the SCL and SDA signal lines need to be operating in open-drain mode (no upper transistor). The high level on these lines are held via external pull-up devices (approx. 10 kΩ for operation at 100 kbit/s, 2 kΩ for operation at 400 kbit/s).

All pins of the XC161 that are to be used for IIC-Bus communication provide open-drain drivers, and must be programmed to output operation, and their alternate function must be enabled (by setting the respective port output latch to 1), before any communication can be established.

The input lines from the SCL/SDA pins are always connected to the IIC-Bus Module, and do not require special programming. The inputs feature digital input filters in order to improve the rejection of noise from the external bus lines.

Table 20-3 shows the required register setting to configure the IO lines of the IIC-Bus Module for master and slave mode operation. Please note that all lines must be configured for open-drain output operation. This is required, e.g., to enable a slave module to actively hold the SCL line low as long as it cannot accept further bus transactions. The IIC-Bus Module deactivates output lines by setting the line to high level, which results in a passive level at the open-drain output.

Table 20-3 IIC IO Selection and Setup

Port Lines	Alternate Select Register	Direction Control Register	Open Drain Control Register
Bus A:			
P9.0 / SDA0	ALTSEL0P9.P0 = 1 and ALTSEL1P9.P0 = X	DP9.P0 = 1	ODP9.P0 = 1
P9.1 / SCL0	ALTSEL0P9.P1 = 1 and ALTSEL1P9.P1 = 0	DP9.P1 = 1	ODP9.P1 = 1
Bus B:			
P9.2 / SDA1	ALTSEL0P9.P2 = 1 and ALTSEL1P9.P2 = 0	DP9.P2 = 1	ODP9.P2 = 1
P9.3 / SCL1	ALTSEL0P9.P3 = 1 and ALTSEL1P9.P3 = 0	DP9.P3 = 1	ODP9.P3 = 1
Bus C:			
P9.4 / SDA2	ALTSEL0P9.P4 = 1 and ALTSEL1P9.P4 = X	DP9.P4 = 1	ODP9.P4 = 1
P9.5 / SCL2	ALTSEL0P9.P5 = 1 and ALTSEL1P9.P5 = X	DP9.P5 = 1	ODP9.P5 = 1

20.6 Interfaces of the IIC-Bus Module

In the XC161, the IIC-Bus Module is connected to IO ports and other internal modules according to **Figure 20-7**.

The input/output lines of the module are connected to pins of Ports P9. The 2 interrupt request lines are connected to the Interrupt Control Block. Please note that two of the thee possible interrupt sources in the IIC-Bus Module or ORed together onto the request line IIC_DIRQ (see **Section 20.4**).

Clock control and emulation control of the IIC-Bus Module is handled by the System Control Unit, SCU.

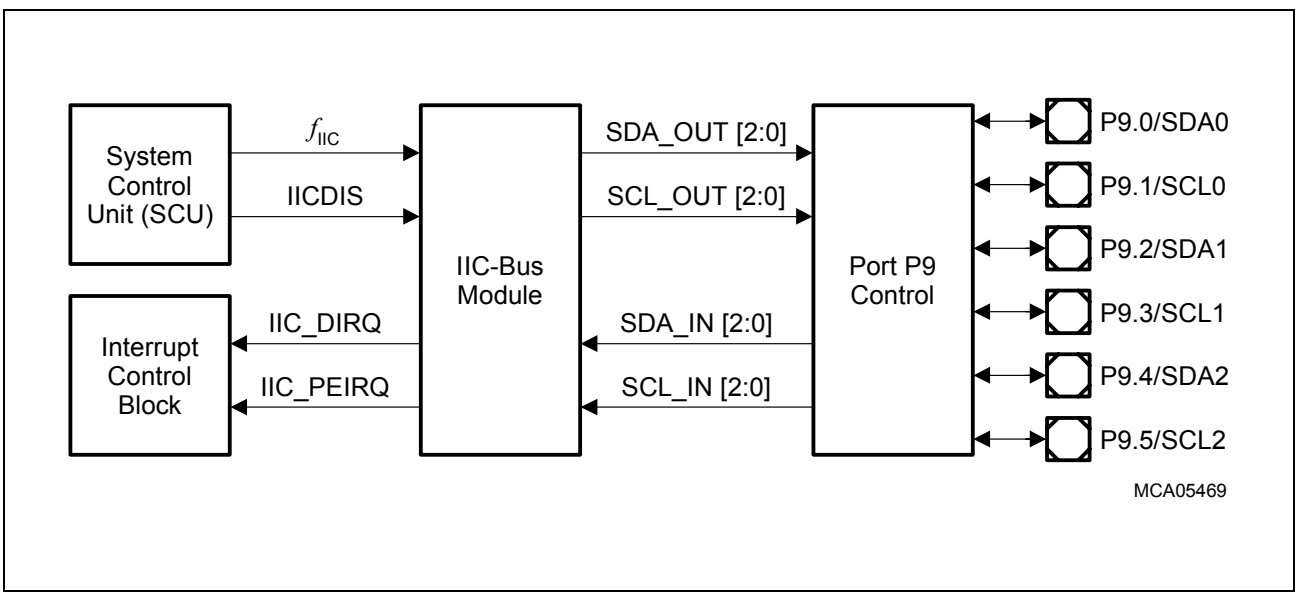


Figure 20-7 IIC-Bus Module IO Interface

20.7 IIC-Bus Overview

Figure 20-8 gives a brief overview of the major definitions of the IIC-Bus operation.

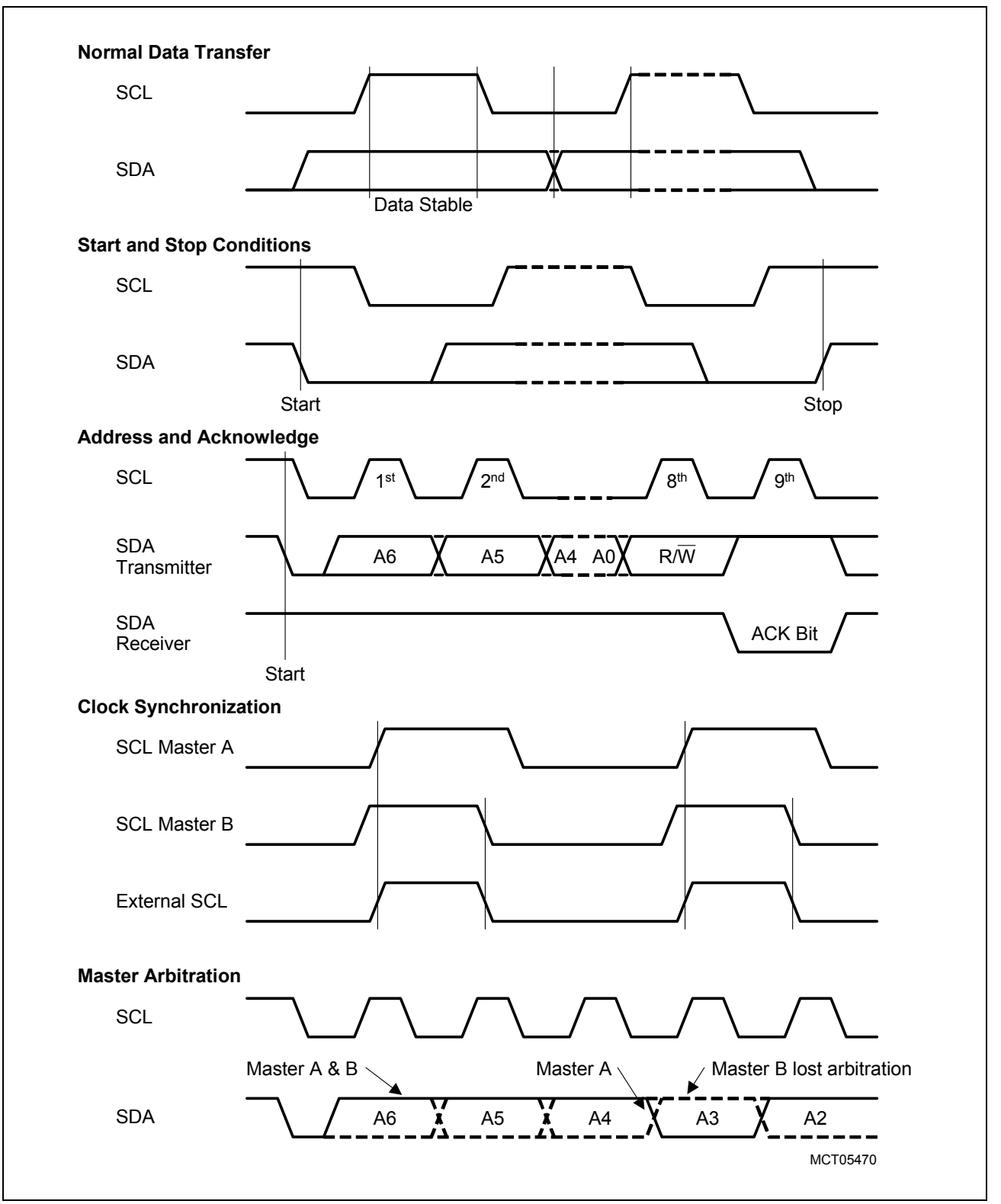


Figure 20-8 IIC-Bus Characteristics

21 TwinCAN Module

21.1 Kernel Description

21.1.1 Overview

The TwinCAN module contains two 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 part B (active). Each of the two Full-CAN nodes can receive and transmit standard frames with 11-bit identifiers as well as extended frames with 29-bit identifiers.

Both CAN nodes share the TwinCAN module's resources in order to optimize the CAN bus traffic handling and to minimize the CPU load. The flexible combination of Full-CAN functionality and FIFO architecture reduces the efforts to fulfill the real-time requirements of complex embedded control applications. Improved CAN bus monitoring functionality as well as the increased number of message objects permit precise and comfortable CAN bus traffic handling.

Depending on the application, each of the 32 message objects can be individually assigned to one of the two CAN nodes. Gateway functionality allows automatic data exchange between two separate CAN bus systems, which reduces CPU load and improves the real time behavior of the entire system.

The bit timings for both CAN nodes are derived from the peripheral clock (f_{CAN}) and are programmable up to a data rate of 1 MBaud. A pair of receive and transmit pins connect each CAN node to a bus transceiver.

Features

- CAN functionality according to CAN specification V2.0 B active.
- Dedicated control registers are provided for each CAN node.
- A data transfer rate up to 1 MBaud is supported.
- Flexible and powerful message transfer control and error handling capabilities are implemented.
- Full-CAN functionality: 32 message objects can be individually
  - assigned to one of the two CAN nodes,
  - configured as transmit or receive object,
  - participate in a 2, 4, 8, 16 or 32 message buffer with FIFO algorithm,
  - setup to handle frames with 11-bit or 29-bit identifiers,
  - provided with programmable acceptance mask register for filtering,
  - monitored via a frame counter,
  - configured to Remote Monitoring Mode.
- Up to eight individually programmable interrupt nodes can be used.
- CAN Analyzer Mode for bus monitoring is implemented.

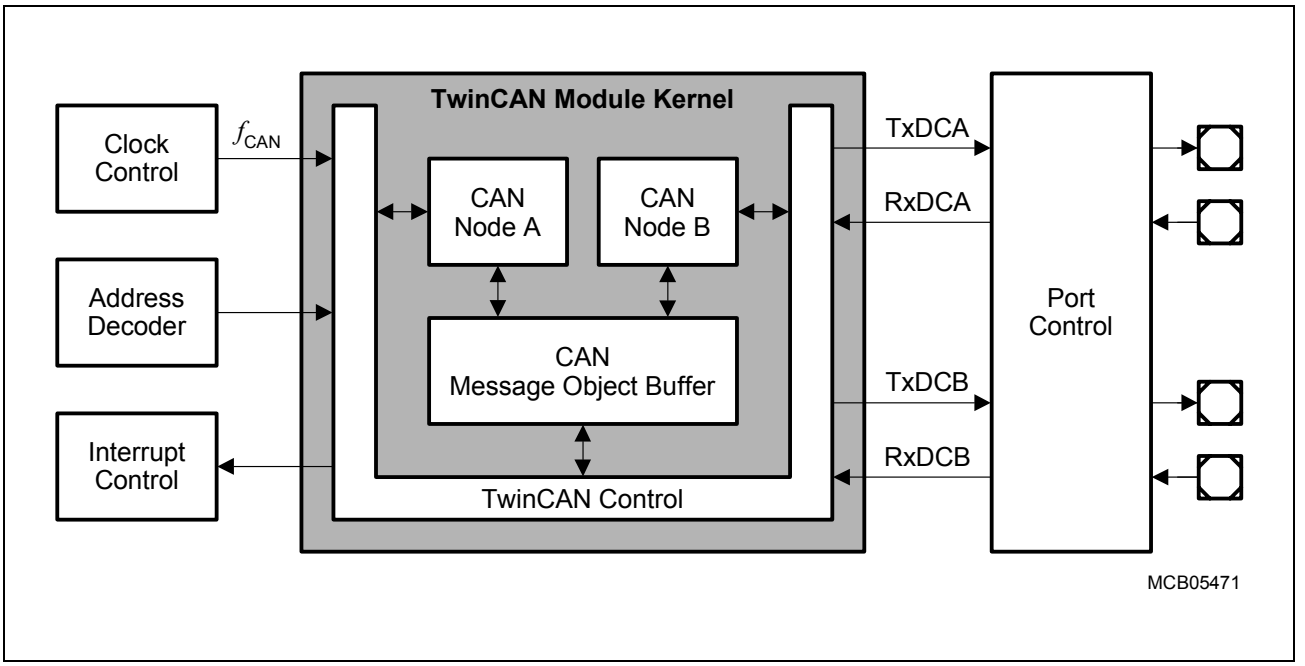


Figure 21-1 General Block Diagram of the TwinCAN Module

The CAN kernel (**Figure 21-2**) is split into

- A global control shell, subdivided into the initialization logic, the global control and status logic and the interrupt request compressor.
  - The initialization logic sets up all submodules after power-on or reset. After finishing the initialization of the node control logic and its associated message objects, the respective CAN node is synchronized with the connected CAN bus.
  - The global control and status logic informs the CPU about pending object transmit and receive interrupts and about the recent transfer history.
  - The interrupt request compressor condenses the interrupt requests from 72 sources, belonging to CAN node A and B, to 8 interrupt nodes.
- A message buffer unit, containing the message buffers, the FIFO buffer management, the gateway control logic and a message-based interrupt request generation unit.
  - The message buffer unit stores up to 32 message objects of 8 bytes maximum data length. Each object has an identifier and its own set of control and status bits. After initialization, the message buffer unit can handle reception and transmission of data without CPU supervision.
  - The FIFO buffer management stores the incoming and outgoing messages in a circular buffer and determines the next message to be processed by the CAN controller.
  - The gateway control logic transfers a message from CAN node A to CAN node B or vice versa.
  - The interrupt request generation unit indicates message-specifically the reception or transmission of an object.

- Two separate CAN nodes, subdivided into a bit stream processor, a bit timing control unit, an error handling logic, an interrupt request generation unit and a node control logic:
  - The bit stream processor performs data, remote, error and overload frames according to the ISO-DIS 11898 standard. The serial data flow between the CAN bus line, the input/output shift register and the CRC register is controlled as well as the parallel data flow between the I/O shift register and the message buffer unit.
  - The bit timing control unit defines the sampling point in respect to propagation time delays and phase shift errors and performs the resynchronization.
  - The error handling control logic manages the receive and the transmit error counter. According the contents in both timers, the CAN controller is set into an error-active, error-passive or bus-off state.
  - The interrupt request generation unit signals globally the successful end of a message transmit or receive operation, all kinds of transfer problems like bit stuffing errors, format, acknowledge, CRC or bit state errors, every change of the CAN bus warning level or of the bus-off state.
  - The node control logic enables and disables the node specific interrupt sources, enters the CAN analyzer mode and administrates a global frame counter.

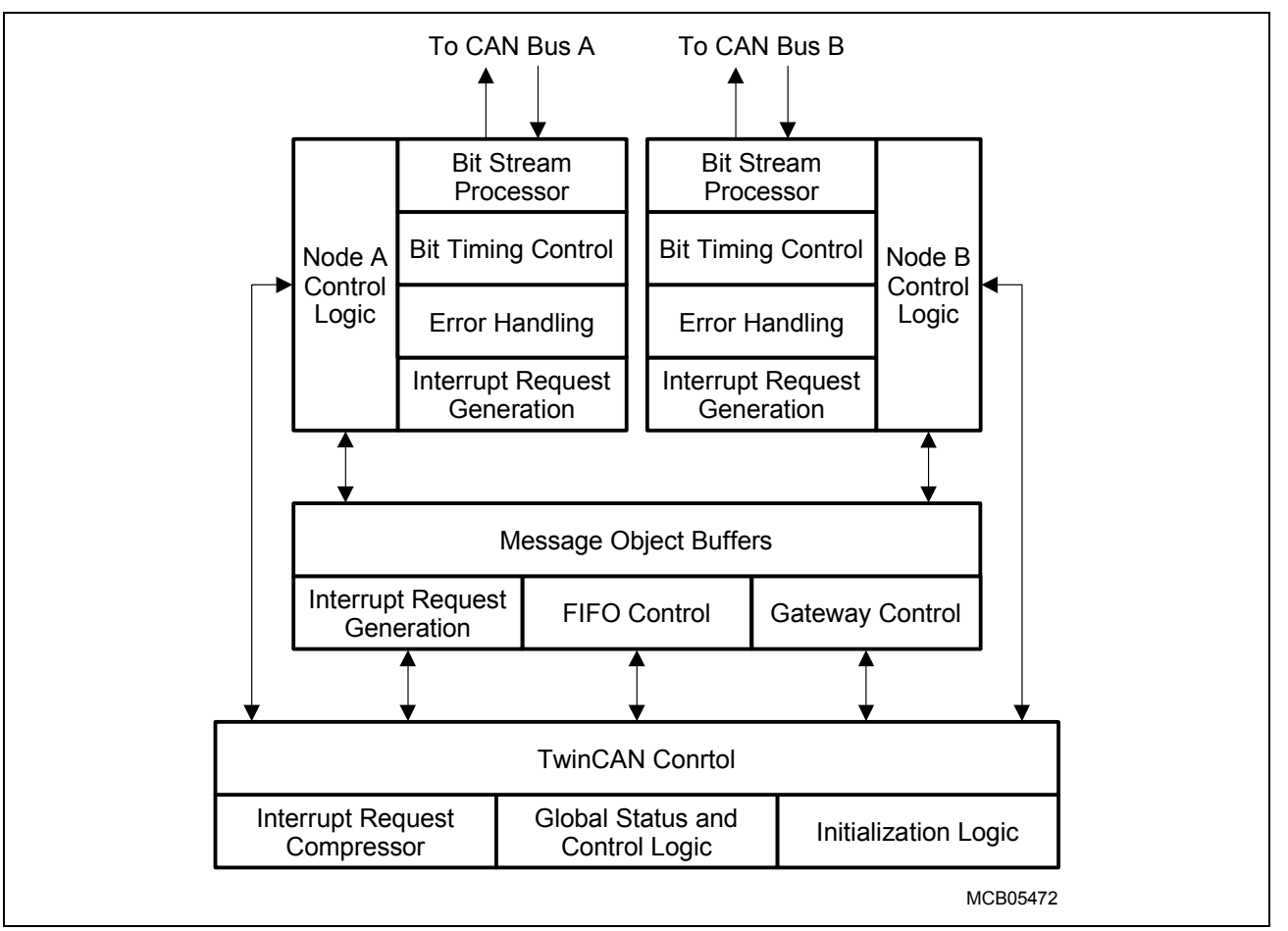


Figure 21-2 Detailed Block Diagram of the TwinCAN Kernel

21.1.2 TwinCAN Control Shell

21.1.2.1 Initialization Processing

After an external hardware reset or while it is bus-off, the respective CAN controller node is logically disconnected from the associated CAN bus and does not participate in any message transfer. This is indicated by the ACR/BCR control register bit INIT = ‘1’, which is automatically set in case of a reset or while the CAN node is bus-off. Furthermore, the CAN node will be disconnected by setting bit INIT to ‘1’ via software. While INIT is active, all message transfers between the affected CAN node controller and its associated CAN bus are stopped and the bus output pin (TXDC) is held on ‘1’ level (recessive state).

After an external hardware reset, all control and message object registers are reset to their associated reset values. During the bus-off-state or after a write access to register ACR/BCR with INIT = ‘1’, all respective control and message object registers hold their current values (except the error counters).

Resetting bit INIT to ‘0’ without being in the bus-off-state starts the synchronization sequence (= connection to the CAN bus), which has to monitor at least one bus-idle event (11 consecutive ‘recessive’ bits) on the associated CAN bus before the node is allowed to take part in CAN traffic again.

During the bus-off recovery sequence:

- The receive and the transmit error counter within the error handling logic are reset.
- 128 bus-idle events (11 consecutive ‘recessive’ bits) have to be detected, before the synchronization sequence can be initiated. The monitoring of the bus idle events is immediately started by hardware after entering the bus-off state. The number of already detected bus-idle events is counted and indicated by the receive error counter.
- The reconnect procedure tests bit INIT by hardware after 128 bus-idle events. If INIT is still set, the affected CAN node controller waits until INIT is cleared and at least one bus-idle event is detected on the CAN bus, before the node takes part in CAN traffic again. If INIT has been already cleared, the message transfer between the affected CAN node controller and its associated CAN bus is immediately enabled.

21.1.2.2 Interrupt Request Compressor

The CAN module is equipped with 32×2 message object specific interrupt request sources and 2×4 node control interrupt request sources. A request compressor condenses these 72 sources to 8 CAN interrupt nodes reporting the interrupt requests of the CAN module. Each request source is provided with an interrupt node pointer, selecting the interrupt node to start the associated service routine in order to increase flexibility in interrupt processing. Each of the 8 CAN interrupt nodes can trigger an independent interrupt routine with its own interrupt vector and its own priority.

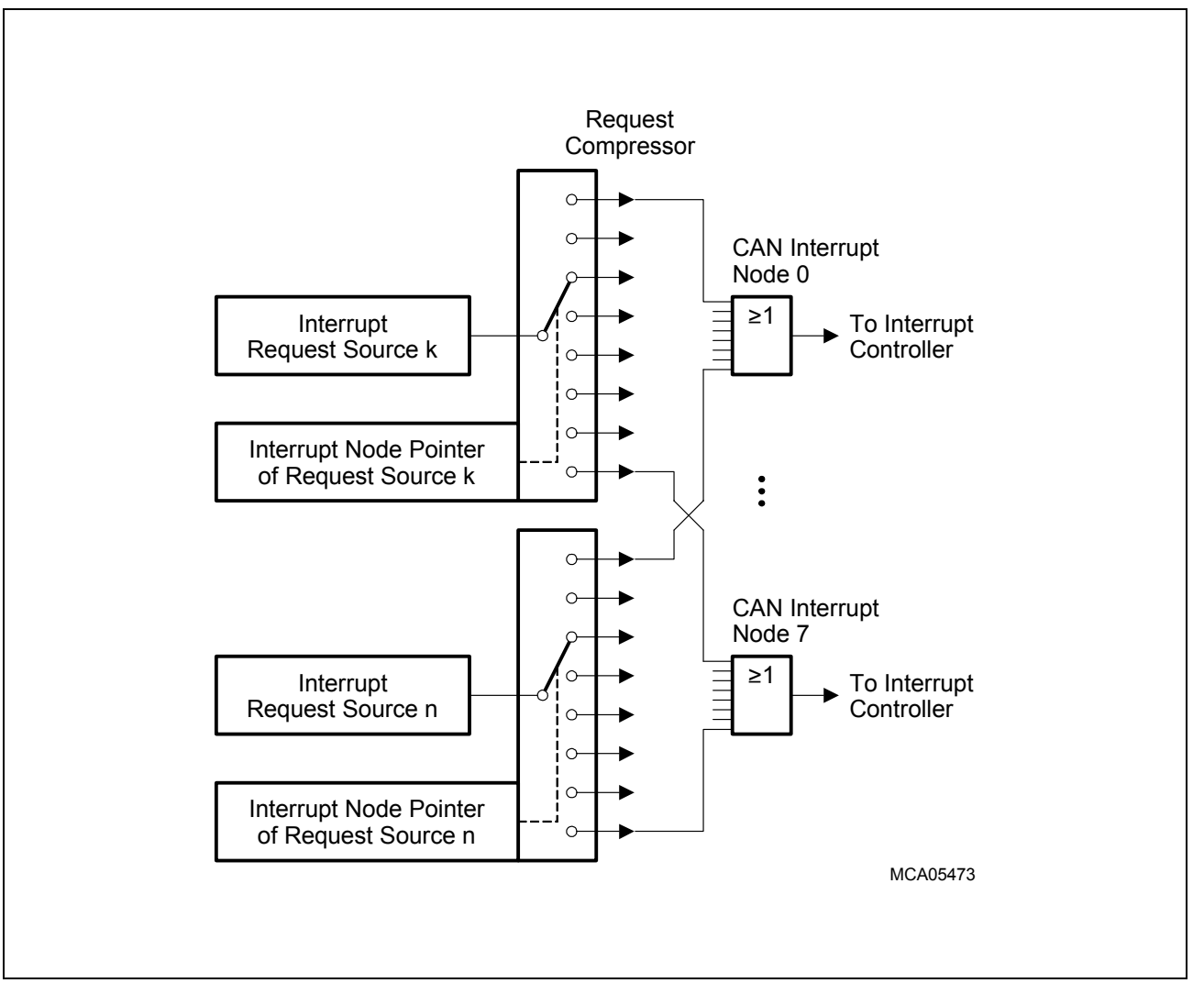


Figure 21-3 Interrupt Node Pointer and Interrupt Request Compressor

Note: All interrupts are event-oriented. The event sets the corresponding indication flag and can generate an interrupt to the system. An interrupt event occurring while its corresponding indication flag is still set, can generate a new interrupt.

21.1.2.3 Global Control and Status Logic

The receive interrupt pending register RXIPND contains 32 individual flags indicating a pending receive interrupt for the associated message objects. Flag RXIPNDn is set by hardware if the corresponding message object has correctly received a data or remote frame and the correlated interrupt request generation has been enabled by RXIEn = ‘10’. RXIPNDn can be cleared by software by resetting bit INTPNDn in the corresponding message object control register MSGCTRn.

The transmit interrupt pending register TXIPND has a similar functionality as the RXIPND register and provides identical information about pending transmit interrupts.

21.1.3 CAN Node Control Logic

21.1.3.1 Overview

Each node is equipped with an individual node control logic configuring the global behavior and providing status information.

The configuration mode is activated when the ACR/BCR register bit CCE is set to ‘1’. This mode allows modifying the CAN bit timing parameters and the error counter registers.

The CAN analyzer mode is activated when bit CALM in control register ACR/BCR is set to ‘1’. In this operation mode, data and remote frames are monitored without an active participation in any CAN transfer (CAN transmit pin is held on recessive level). Incoming remote frames are stored in a corresponding transmit message object, while arriving data frames are saved in a matching receive message object.

In CAN analyzer mode, the entire configuration information of the received frame is stored in the corresponding message object and can be evaluated by the CPU concerning their identifier, XTD bit information and data length code. If the remote monitoring mode is active by RMM = ‘1’, this information is also available for received remote frames. Incoming frames are not acknowledged and no error frames are generated. Neither remote frames are answered by the corresponding data frame nor data frames can be transmitted by setting TXRQ, if CAN analyzer mode is enabled. Receive interrupts are generated (if enabled) for all correctly received frames and the respective remote pending RMTPNDn is set in case of received remote frames.

The node specific interrupt configuration is also defined by the node control logic via the ACR/BCR register bits SIE, EIE and LECIE:

- If control bit SIE is set to ‘1’, a status change interrupt occurs when the ASR/BSR register has been updated (by each successfully completed message transfer).
- If control bit EIE is set to ‘1’, an error interrupt is generated when a bus-off condition has been recognized or the error warning level has been exceeded or underrun.
- If control bit LECIE is set to ‘1’, a last error code interrupt is generated when an error code is set in bitfield LEC in the status registers ASR or BSR.

The status register (ASR/BSR) provides an overview about the current state of the respective TwinCAN node:

- Flag TXOK is set when a message has been transmitted successfully and acknowledged by at least one other CAN node,
- flag RXOK indicates an error-free reception of a CAN bus message,
- bitfield LEC indicates the last error occurred on the CAN bus. Stuff, form, and CRC errors as well as bus arbitration errors (Bit0, Bit1) are reported,
- bit EWRN is set when at least one of the error counters in the error handling logic has reached the error warning limit (default value 96),

- bit BOFF is set when the transmit error counter exceeded the error limit of 255 and the respective CAN node controller has been logically disconnected from the associated CAN bus.

The CAN frame counter can be used to check the transfer sequence of message objects or to obtain information about the time instant, a frame has been transmitted or received from the associated CAN bus. CAN frame counting is performed by a 16-bit counter, which is controlled by register AFCR/BFCR. Bitfield CFCMD defines the operation mode and the trigger event incrementing the frame counter:

- After correctly transmitted frames,
- after correctly received frames,
- after a foreign frame on the CAN bus (not transmitted/received by the CAN node itself),
- at beginning of a new bit time.

The captured frame counter value is copied to the CFCVAL field of the associated MSGCTRn register at the end of the monitored frame transfer. Flag CFCOV is set on a frame counter overflow condition ($FFFF_H$ to 0000_H) and an interrupt request is generated if bit CFCIE is set to ‘1’.

21.1.3.2 Timing Control Unit

According to ISO-DIS 11898 standard, a CAN bit time is subdivided into different segments (**Figure 21-4**). Each segment consists of multiples of a time quantum t_q. The magnitude of t_q is adjusted by the bitfield BRP and by bit DIV8X, both controlling the baud rate prescaler (see bit timing register ABTR/BBTR). The baud rate prescaler is driven by the CAN module clock f_{CAN}.

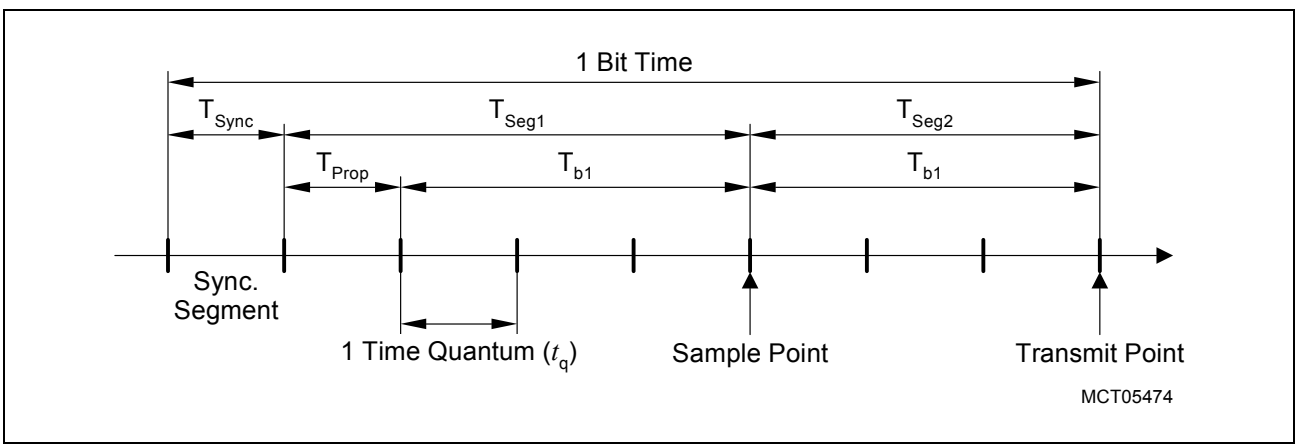


Figure 21-4 CAN Bus Bit Timing Standard

The synchronization segment (T_{Sync}) allows a phase synchronization between transmitter and receiver time base. The synchronization segment length is always 1 t_q. The propagation time segment (T_{Prop}) takes into account the physical propagation delay in the transmitter output driver, on the CAN bus line and in the transceiver circuit. For a working collision detect mechanism, T_{Prop} has to be two times the sum of all propagation delay quantities rounded up to a multiple of t_q. The phase buffer segments 1 and 2 (T_{b1}, T_{b2}) before and after the signal sample point are used to compensate a mismatch between transmitter and receiver clock phase detected in the synchronization segment.

The maximum number of time quanta allowed for resynchronization is defined by bitfield SJW in bit timing register ABTR/BBTR. The propagation time segment and the phase buffer segment 1 are combined to parameter T_{Seg1}, which is defined by the value TSEG1 in the respective bit timing register ABTR/BBTR. A minimum of 3 time quanta is requested by the ISO standard. Parameter T_{Seg2}, which is defined by the value of TSEG2 in the bit timing register ABTR/BBTR, covers the phase buffer segment 2. A minimum of 2 time quanta is requested by the ISO standard. According 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.

Note: The access to bit timing register ABTR/BBTR is only enabled if bit CCE in control register ACR/BCR is set to ‘1’.

Calculation of the Bit Time

t_q = (BRP + 1) / f_{CAN}, if DIV8X = '0'

= (BRP + 1) / 8 × f_{CAN}, if DIV8X = '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 has to synchronize on any edge from the recessive to the dominant bus level. If the hard synchronization is enabled (at the start of frame), the bit time is restarted at the synchronization segment. Otherwise, the resynchronization jump width T_{SJW} defines the maximum number of time quanta a bit time may be shortened or lengthened by one resynchronization. The value of SJW is programmed in the ABTR/BBTR registers.

T_{SJW} = (SJW + 1) × t_q

$T_{Seg1} \geq T_{SJW} + T_{prop}$

$T_{Seg2} \geq T_{SJW}$

The maximum relative tolerance for f_{CAN} depends on the phase buffer segments and the resynchronization jump width.

$df_{CAN} \leq \min(T_{b1}, T_{b2}) / 2 \times (13 \times \text{bit time} - T_{b2})$ AND

$df_{CAN} \leq T_{SJW} / 20 \times \text{bit time}$

Calculation of the Baudrate

Baudrate = f_{CAN} / ((BRP + 1) × (1 + T_{Seg1} + T_{Seg2}))

21.1.3.3 Bitstream Processor

Based on the objects in the message buffer, the bitstream processor generates the remote and data frames to be transmitted via the CAN bus. It controls the CRC generator and adds the checksum information to the new remote or data frame. After including the start of frame bit SOF and the end of frame field EOF, the bitstream processor starts the CAN bus arbitration procedure and continues with the frame transmission when the bus was found in idle state. While the data transmission is running, the bitstream processor 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 bitfield LEC in status register ASR/BSR.

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 associated error code is presented in status register ASR/BSR. 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.

21.1.3.4 Error Handling Logic

The error handling logic is responsible for the fault confinement of the CAN device. Its two counters, the receive error counter and the transmit error counter (control registers AECNT, BECNT), are incremented and decremented by commands from the bit stream processor. If the bit stream processor itself detects an error while a transmit operation is running, the transmit error counter is incremented by 8. An increment of 1 is used, when the error condition was reported by an external CAN node via an error frame generation. For error analysis, the transfer direction of the disturbed message and the node, recognizing the transfer error, are indicated in the control registers AECNT, BECNT. According to the values of the error counters, the CAN controller is set into the states error-active, error-passive or bus-off.

The CAN controller is in error-active state, if both error counters are below the error-passive limit of 128. It is in error-passive state, if at least one of the error counters equals or exceeds 128.

The bus-off state is activated if the transmit error counter equals or exceeds the bus-off limit of 256. This state is reported by flag BOFF in the ASR/BSR status register. The device remains in this state, until the bus-off recovery sequence is finished. Additionally, there is the bit EWRN in the ASR/BSR status register, which is set if at least one of the error counters equals or exceeds the error warning limit defined by bitfield EWRNLVL in the control registers AECNT, BECNT. Bit EWRN is reset if both error counters fall below the error warning limit again.

21.1.3.5 Node Interrupt Processing

Each CAN node is equipped with 4 interrupt sources supporting the

- global transmit/receive logic,
- CAN frame counter,
- error reporting system.

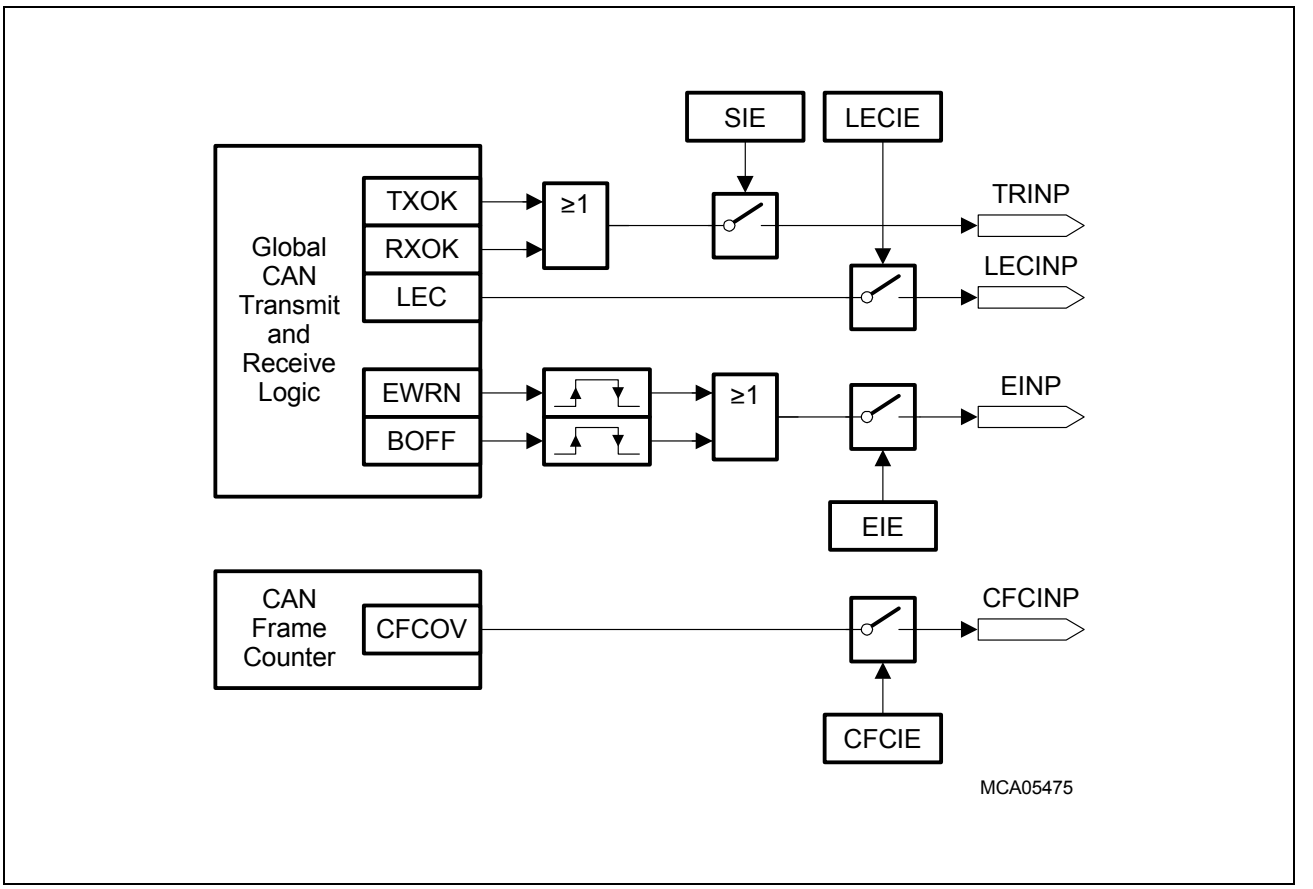


Figure 21-5 Node Specific Interrupt Control

If enabled by bit SIE = ‘1’ in the ACR/BCR register, the global transmit/receive logic generates an interrupt request, if the node status register (ASR/BSR) is updated after finishing a faultless transmission or reception of a message object. The associated interrupt node pointer is defined by bitfield TRINP in control register AGINP/BGINP.

An error is reported by a last error code interrupt request, if activated by LECIE = ‘1’ in the ACR/BCR register. The corresponding interrupt node pointer is defined by bitfield LECINP in control register AGINP/BGINP.

The CAN frame counter creates an interrupt request upon an overflow, when the AFCR/BFCR control register bit CFCIE is set to ‘1’. Bitfield CFCINP, located also in the AGINP/BGINP control register, selects the corresponding interrupt node pointer.

The error logic monitors the number of CAN bus errors and sets or resets the error warning bit EWRN according to the value in the error counters. If bit EIE in control

register ACR/BCR is set to ‘1’, an interrupt request is generated on any modification of bits EWRN and BOFF. The associated interrupt node pointer is defined by bitfield EINP in control register AGINP/BGINP.

21.1.3.6 Message Interrupt Processing

Each message object is equipped with 2 interrupt request sources indicating the successful end of a message transmission or reception.

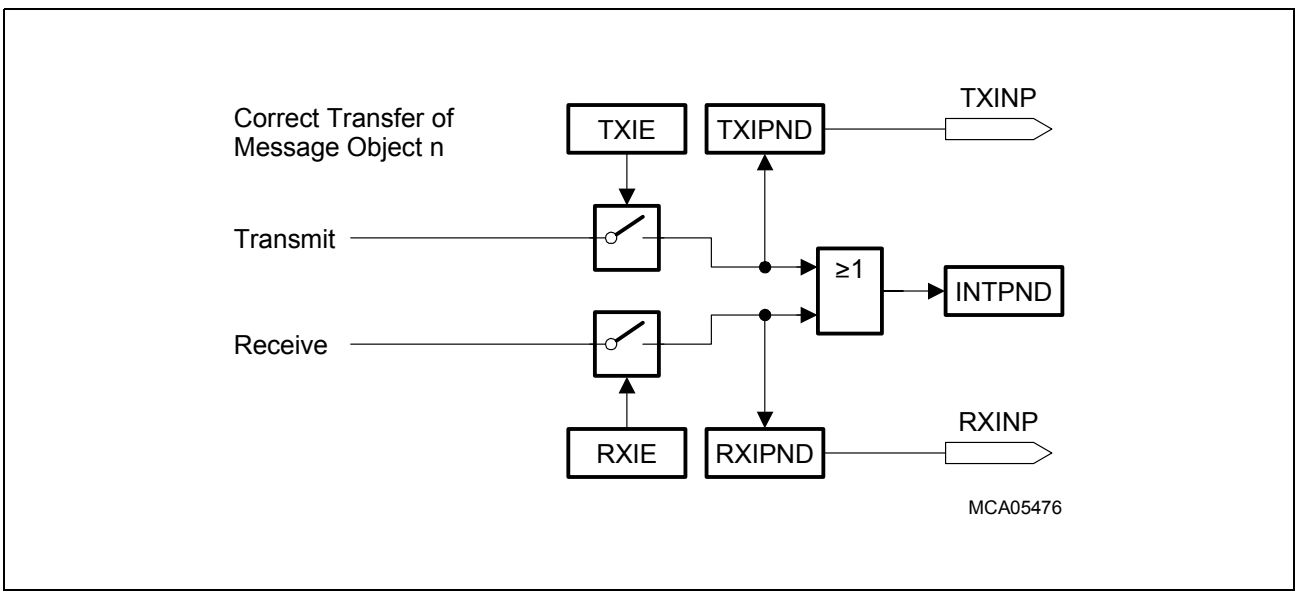


Figure 21-6 Message Specific Interrupt Control

The message based transfer interrupt sources are enabled, if bit TXIE or RXIE in the associated message control register MSGCTRn are set to ‘10’. The associated interrupt node pointers are defined by bitfields RXINP and TXINP in message configuration register MSGCFGn.

21.1.3.7 Interrupt Indication

The AIR/BIR register provides an INTID bitfield indicating the source of the pending interrupt request with the highest internal priority (lowest message object number). The type of the monitored interrupt requests, taken into account by bitfield INTID, can be selected by registers AIMR0/AIMR4 and BIMR0/BIMR4 containing a mask bit for each interrupt source. If no interrupt request is pending, all bits of AIR/BIR are cleared. The interrupt requests INTPNDn have to be cleared by software.

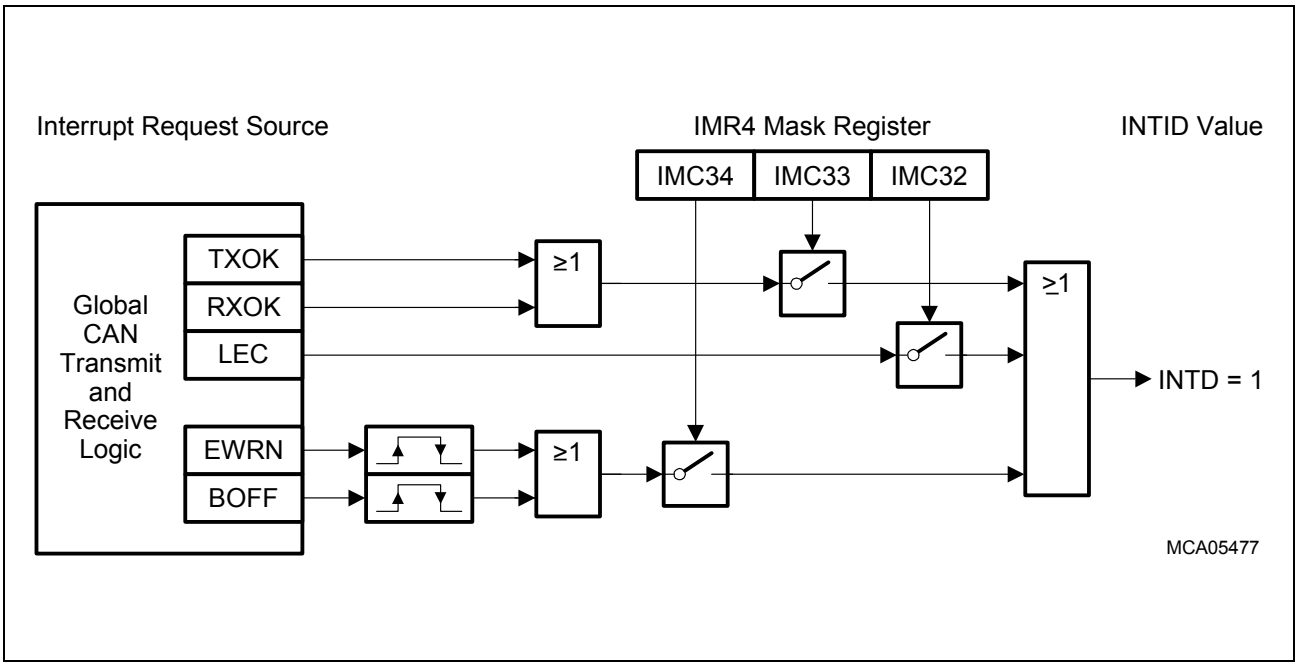


Figure 21-7 INTID Mask for Global Interrupt Request Sources

Registers AIMR0/4 and BIMR0/4 contain a mask bit for each interrupt source (AIMR0/BIMR0 for message specific interrupt sources and AIMR4/BIMR4 for the node specific interrupt sources). If a mask bit is reset, the corresponding interrupt source is not taken into account for the generation of the INTID value.

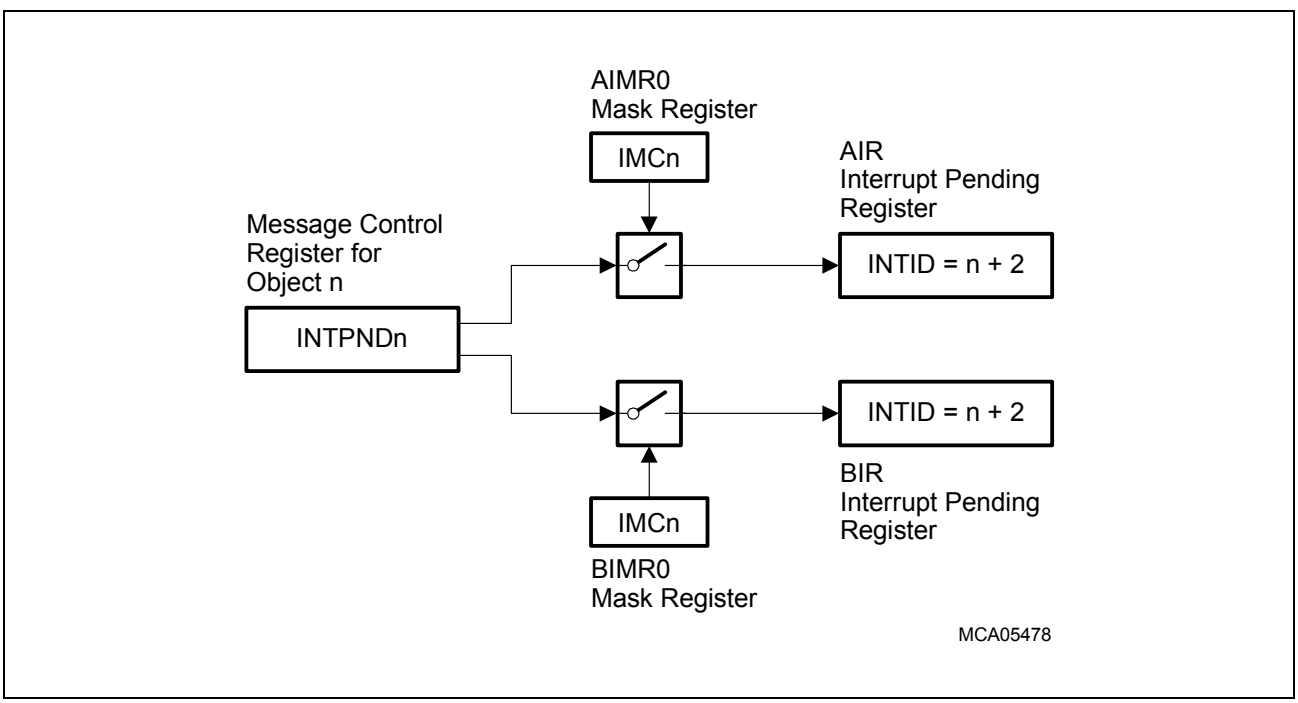


Figure 21-8 INTID Mask for Message Interrupt Request Sources

21.1.4 Message Handling Unit

A message object is the basic information unit exchanged between the CPU and the CAN controller. 32 message objects are provided by the internal CAN memory. Each of these objects has an identifier, its own set of control and status bits and a separate data area. Each message object covers 32 bytes of internal memory subdivided into control registers and data storage as illustrated in **Figure 21-9**.

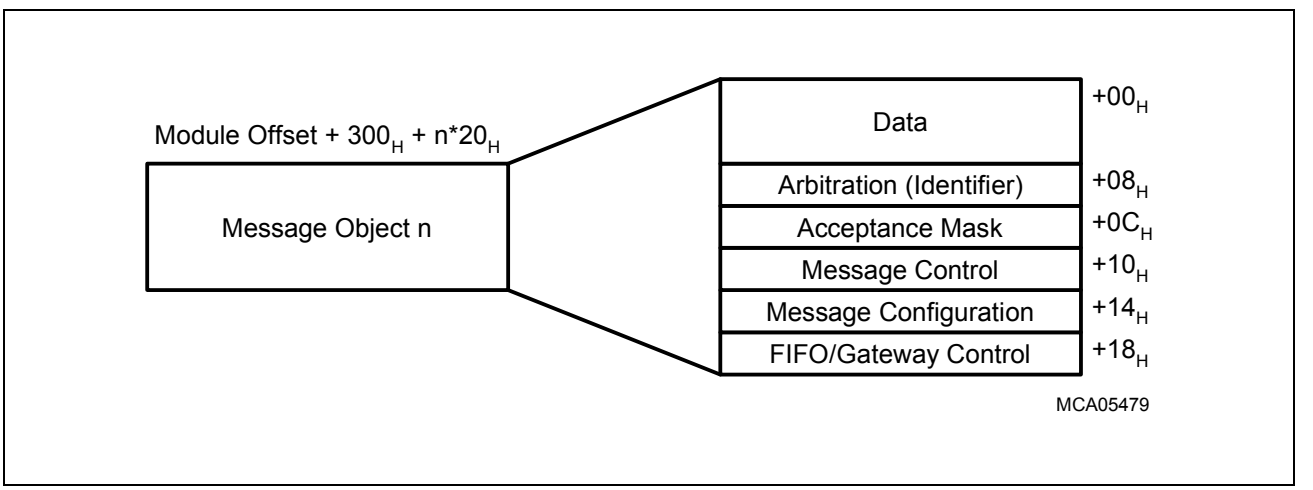


Figure 21-9 Structure of a Message Object

In normal operation mode, each message object is associated with one CAN node. Only in shared gateway mode, a message object can be accessed by both CAN nodes (according to the corresponding bitfield NODE).

In order to be taken into account by the respective CAN node control logic, the message object must be declared valid in its associated message control register (bit MSGVAL).

When a message object is initialized by the CPU, bitfield MSGVAL in message control register MSGCTRn should be reset, inhibiting a read or write access of the CAN node controller to the associated register and data buffer storage. Afterwards, the message identifier and operation mode (transmit, receive) must be defined. If a successful transmission and/or reception of a message object should be followed by the execution of an interrupt service routine, the respective bitfields TXIE and RXIE have to be set and the interrupt pending indicator (bitfield INTPND) should be reset.

If the automatic response of an incoming remote frame with matching identifier is not requested, the respective transmission message object should be configured with CPUUPD = ‘10’.

As soon as bitfield MSGVAL is set to ‘10’, the respective message object is operable and taken into account by the associated CAN node controller.

21.1.4.1 Arbitration and Acceptance Mask Register

The arbitration register MSGARn is used to filter the incoming messages and to provide the outgoing messages with an identifier. The acceptance mask register MSGAMRn may be used to disable some identifier bits of an incoming message for the acceptance test.

The identifier of a received message is compared (bitwise XOR) to the identifiers of all message objects stored in the internal CAN controller memory. The compare operation starts at object 0 and takes into account all objects with

- a valid message flag (MSGVAL = ‘10’),
- a suitable NODE declaration (register MSGCFGn),
- a cleared DIR control bit (receive message object) for data frame reception,
- DIR = ‘1’ (transmit message object) for remote frame reception,
- a matching identifier length declaration (XTD = ‘1’ marks extended 29-bit identifiers, XTD = ‘0’ indicates standard 11 bit identifiers).

The result of the compare operation is bit-by-bit ANDED with the contents of the acceptance mask register (**Figure 21-10**). If concordance is detected, the received message is stored into the CAN controller’s message object. The compare operation is finished after analyzing message object 31.

Note: Depending on the allocated identifiers and the corresponding mask register contents, multiple message objects may fulfill the selection criteria described above. In this case, the received frame is stored in the fitting message object with the lowest message number.

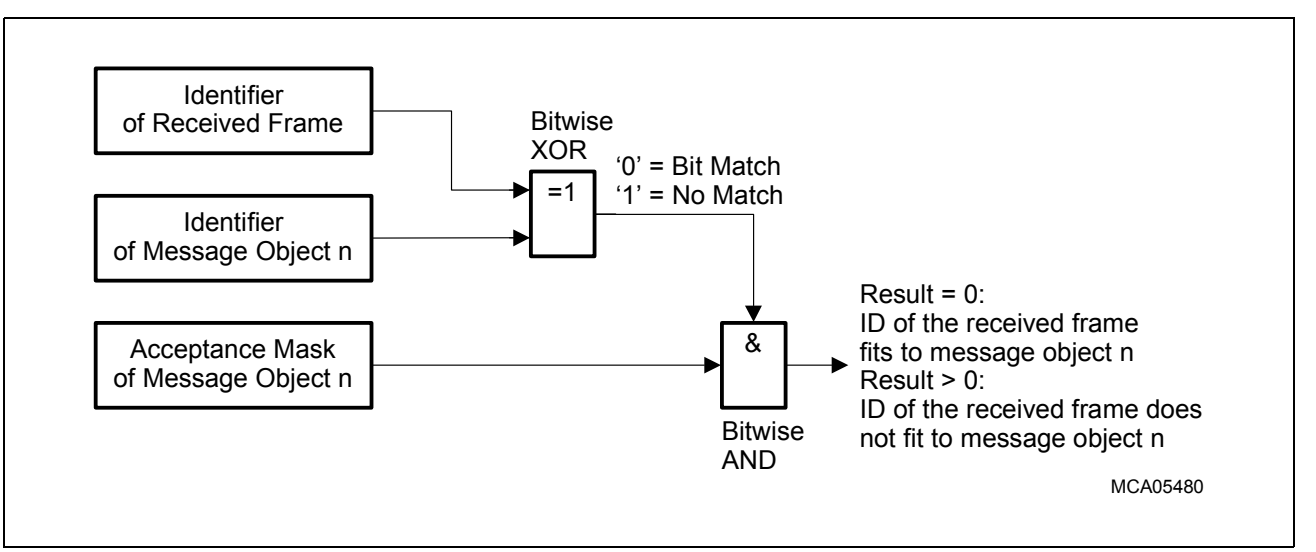


Figure 21-10 Acceptance Filtering for Received Message Identifiers

21.1.4.2 Handling of Remote and Data Frames

Message objects can be set up for transmit or receive operation according to the selected value for control bit DIR. The impact of the message object type on the associated CAN node controller concerning the generation or reception of remote and data frames is illustrated in **Table 21-1**.

Table 21-1 Handling of Remote and Data Frames

	A transmission request (TXRQ = ‘10’) for this message object generates …	**If a data frame with matching identifier is received …**	**If a remote frame with matching identifier is received …**
Receive Object (receives data frames, transmits remote frames, control bit DIR = ‘0’)	… a remote frame. The requested data frame is stored in this message object on reception.	… the data frame is stored in this message object.	… the remote frame is NOT taken into account.
Transmit Object (transmits data frames, receives remote frames, control bit DIR = ‘1’)	… a data frame based upon the information stored in this message object.	… the data frame is NOT stored.	… the remote frame is stored in this message object and RMTPND and TXRQ are set to ‘10’. A data frame, based upon the information stored in this message object, is automatically generated if CPUUPD is set to ‘01’.

21.1.4.3 Handling of Transmit Message Objects

A message object with direction flag DIR = ‘1’ (message configuration register MSGCFGn) is handled as transmit object.

All message objects with bitfield MSGVAL = ‘10’ are operable and taken into account by the CAN node controller operation described below.

During the initialization phase, the transmit request bitfield (TXRQ), the new information bitfield (NEWDAT) should be reset to ‘01’ and the update in progress by CPU bitfield (CPUUPD) in register MSGCTRn should be reset to ‘10’. The message bytes to be transmitted are written into the data partition of the message object (MSGDRn0, MSGDRn4). The number of message bytes to be transmitted has to be written to bitfield DLC in register MSGCFGn. The selected identifier has to be written to register MSGARn. Then, bitfield NEWDAT in register MSGCTRn should be set to ‘10’ and bitfield CPUUPD should be reset to ‘01’ by the CPU.

When the remote monitoring mode is enabled (RMM = ‘1’ in MSGCFGn), the identifier and the data length code of a received remote frame will be copied to the corresponding transmit message object, if a matching identifier was found during the compare and mask operation with all CAN message objects. The copy procedure may change the identifier in the transmit message object, if some MSGAMRn mask register bits have been set to ‘0’.

As long as bitfield MSGVAL in register MSGCTRn is set to ‘10’, the reception of a remote frame with matching identifier automatically sets bitfield TXRQ to ‘10’. Simultaneously, bitfield RMTPND in register MSGCTRn is set to ‘10’ in order to indicate the reception of an accepted remote frame. Alternatively, TXRQ may be set by the CPU via a write access to register MSGCTRn. If the transmit request bitfield TXRQ is found at ‘10’ (while MSGVAL = ‘10’ and CPUUPD = ‘01’) by the appropriate CAN controller node, a data frame based upon the information stored in the respective transmit message object is generated and automatically transferred when the associated CAN bus is idle.

If bitfield CPUUPD in register MSGCTRn is set to ‘10’, the automatic transmission of a message object is prohibited and flag TXRQ is not evaluated by the respective CAN node controller. The CPU can release the pending transmission by clearing CPUUPD. This allows the user to listen on the bus and to answer remote frames under software control.

When the data partition of a transmit message object has to be updated by the CPU, bitfield CPUUPD in message control register MSGCTRn should be set to ‘10’, inhibiting a read or write access of the associated CAN node controller. If a remote frame with an accepted identifier arrives during the update of a message object’s data storage, bitfields TXRQ and RMTPND are automatically set to ‘10’ and the transmission of the corresponding data frame is pending until CPUUPD is reset again.

If several valid message objects with pending transmission request are noticed by the associated CAN node controller, the contents of the message object with the lowest message number is transmitted first.

Bitfield NEWDAT is internally reset by the respective CAN node controller when the contents of the selected message object's data registers is copied to the bitstream processor. Bitfields RMTPND and TXRQ are automatically reset when the message object has been successfully transmitted.

The captured value of the frame counter is copied to bitfield CFCVAL in register MSGCTRn and a transmit interrupt request is generated (INTPNDn and TXIPNDn are set) if enabled by TXIE = '10'. Then the Frame Counter is incremented by one if enabled in control register AFCR/BFCR.

When a data frame with matching identifier is received, it is ignored by the respective transmit object and not indicated by any interrupt request.

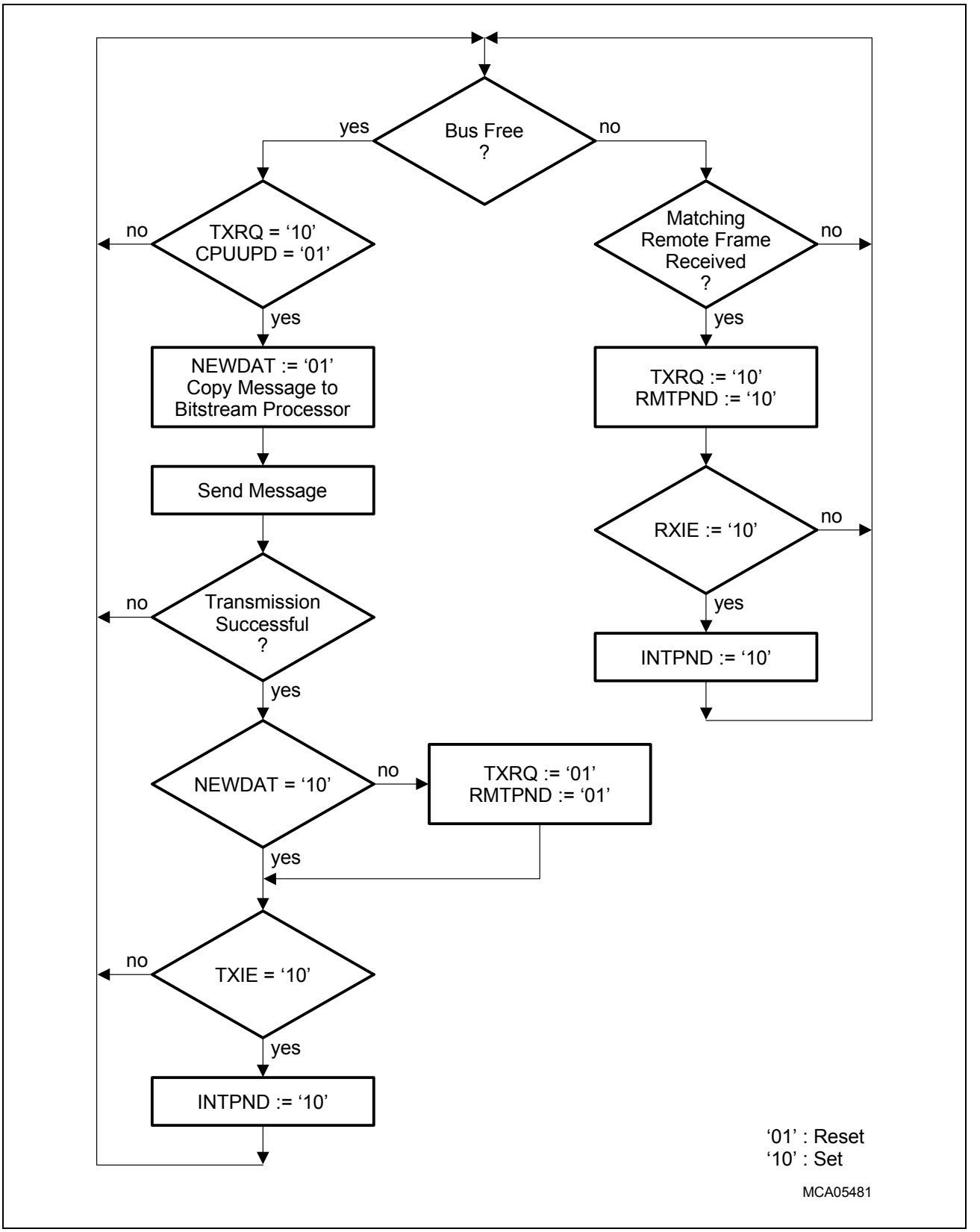


Figure 21-11 Handling of Message Objects with Direction = ‘1’ = Transmit by the CAN Controller Node Hardware

21.1.4.4 Handling of Receive Message Objects

A message object with direction flag DIR = ‘0’ (message configuration register MSGCFGn) is handled as receive object.

In the initialization phase, the transmit request bitfield (TXRQ), the message lost bitfield (MSGLST) and the NEWDAT bitfield in register MSGCTR should be reset.

All message objects with bitfield MSGVAL = ‘10’ are operable and taken into account by the CAN node controller operation described below.

When a data frame has been received, the new information is stored in the data partition of the message object (MSGDRn0, MSGDRn4) and the bitfield DLC in register MSGCFG is updated with the number of received bytes. Unused message bytes will be overwritten by non-specified values. If the NEWDAT bitfield in register MSGCTR is still set, the CAN controller assumes an overwrite of the previously stored message and signals a data loss by setting bitfield MSGLST. In any case, bitfield NEWDAT is automatically set to ‘10’ reporting an update of the data register by the CAN controller. The captured value of the frame counter is copied to bitfield CFCVAL in register MSGCTRn and a receive interrupt request is generated (INTPNDn and RXIPNDn are set) if enabled by RXIE = ‘10’. Then the frame counter is incremented by one if enabled in control register AFCR/BFCR.

When a receive object is marked to be transmitted (TXRQ = ‘10’), bit MSGLST changes automatically to CPUUPD. If CPUUPD is reset to ‘01’, the CAN controller generates a remote frame which is emitted to the other communication partners via CAN bus. In case of CPUUPD = ‘10’, the remote frame transfer is prohibited until the CPU releases the pending transmission by resetting CPUUPD to ‘01’. RMTPND and TXRQ are automatically reset, when the remote frame has been successfully transmitted. Finally, a transmit interrupt request is generated if enabled by TXIE = ‘10’.

When a remote frame with matching identifier is received, it is not answered and not indicated by an interrupt request.

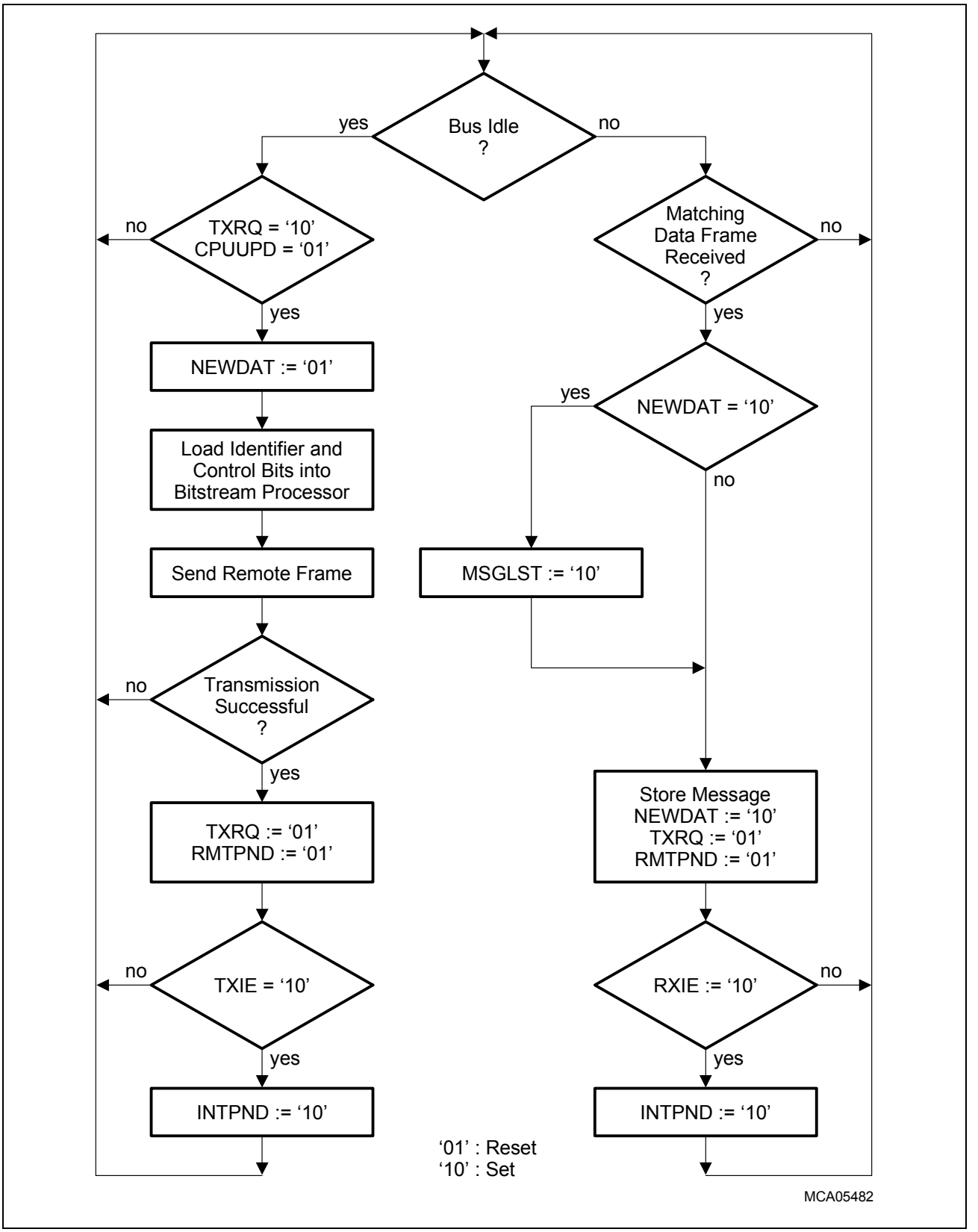


Figure 21-12 Handling of Message Objects with Direction = '0' = Receive by the CAN Controller Node Hardware

21.1.4.5 Single Data Transfer Mode

The single data transfer mode is a useful feature in order to broadcast data over the CAN bus without unintended doubling of information. The single data transfer mode is selected via bit SDT in the FIFO/Gateway control register MSGFGCRn.

Each received data frame with matching identifier is automatically stored in the corresponding receive message object if MSGVAL is set to ‘10’. When data frames addressing the same message object are received within a short time interval, information might get lost (indicated by MSGLST = ‘10’), if the CPU has not processed the former message object contents in time.
Each arriving remote frame with matching identifier is answered by a data frame based on the contents of the corresponding message object. This behavior may lead to multiple generation and transmission of identical data frames according to the number of accepted remote requests.

If SDT is set to ‘1’, the CAN node controller automatically resets bit MSGVAL in a message object after receiving a data frame with corresponding identifier. All following data frames, addressing the disabled message object, are ignored until MSGVAL is set again by the CPU.
If SDT is set to ‘1’, the CAN node controller automatically resets bit MSGVAL in the addressed message object, when the transmission of the corresponding data frame has been finished successfully. In consequence, all following remote requests concerning the disabled message object are ignored until MSGVAL is set again by the CPU. This feature allows for transmitting data in a consecutive manner without unintended doubling of any information.

If SDT is cleared, control bitfield MSGVAL is not reset by the CAN node controller.

21.1.5 CAN Message Object Buffer (FIFO)

In case of a high CPU load, it may be difficult to process an incoming data frame before the corresponding message object is overwritten with the next input data stream provided by the CAN node controller. Depending on the application, it could be also necessary to ensure a minimum data frame generation rate to fulfill external real time requirements.

Therefore, a message buffer facility has been implemented in order to avoid a loss of incoming messages and to minimize the setup time for outgoing messages. Some message objects can be configured as a base object using succeeding slave message objects as individual buffer storage (building a circular buffer used as message FIFO).

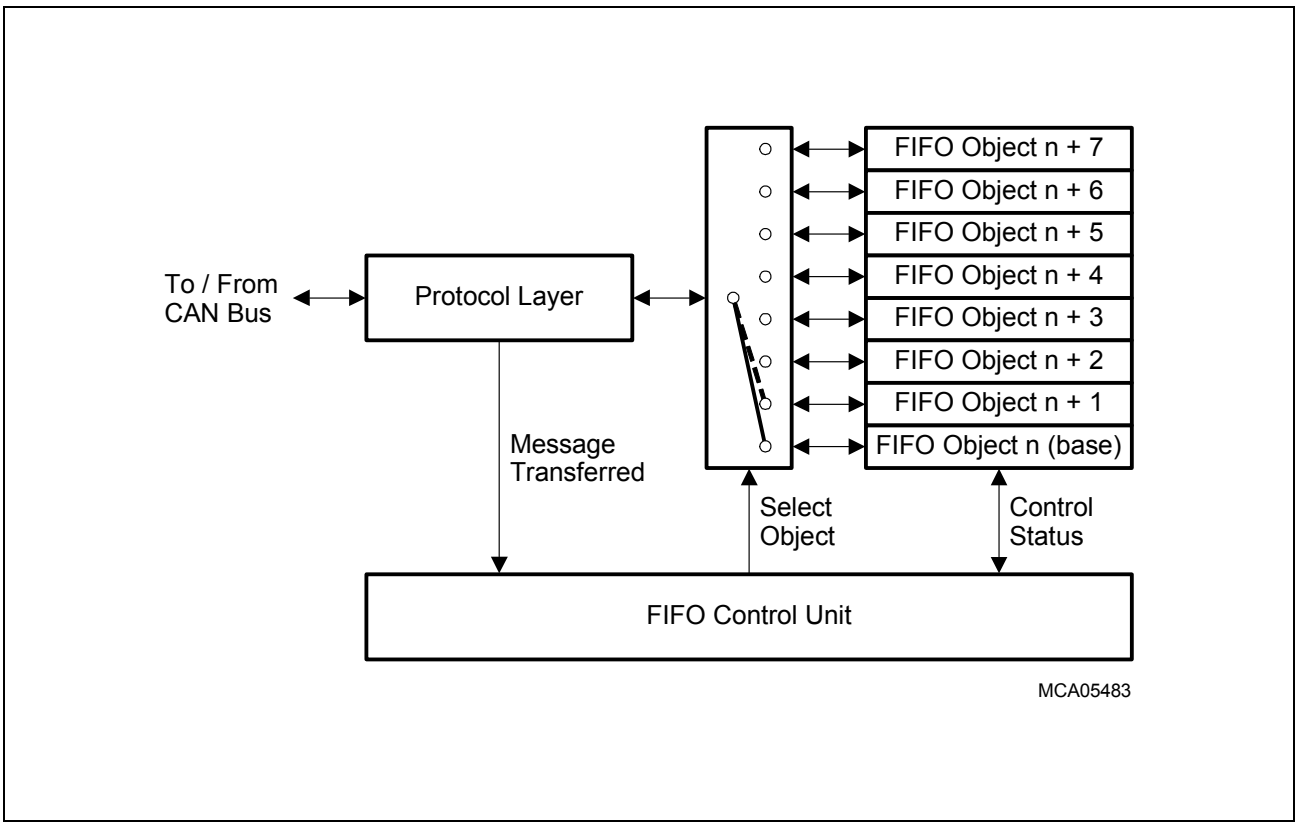


Figure 21-13 FIFO Buffer Control Structure

The number of base and slave message objects, combined to a buffer, has to be a power of two (2, 4, 8 etc.) and the buffer base address has to be an integer multiple of the buffer length (e.g. a buffer containing 8 messages can use object 0, 8, 16 or 24 as base object as illustrated in **Table 21-2**).

A base object is defined by setting bitfield MMC to ‘010’ in control register MSGFGCRn and the requested buffer size is determined by selecting an appropriate value for FSIZE. A slave object is defined by setting bitfield MMC to ‘011’. Bitfield FSIZE has to be equal in all FIFO elements in the same FIFO.

Table 21-2 Message Objects Providing FIFO Base Functionality

Msg. Object n > FIFO Size	0	2	4	6	8	10	12	14	16	18	…	30
2 stage FIFO	X	X	X	X	X	X	X	X	X	X		X
4 stage FIFO	X	–	X	–	X	–	X	–	X	–		–
8 stage FIFO	X	–	–	–	X	–	–	–	X	–		–
16 stage FIFO	X	–	–	–	–	–	–	–	X	–		–
32 stage FIFO	X	–	–	–	–	–	–	–	–	–		–

The identifiers and corresponding acceptance masks have to be identical in all FIFO elements belonging to the same buffer in case of a receive FIFO (DIR = ‘0’). In case of a transmit FIFO (DIR = ‘1’) the identifier of the currently addressed message object is taken into account for transmission.

Each member of a buffer configuration keeps its individual MSGVAL, NEWDAT, CPUUPD or MSGLST, TXRQ and RMTPND flag and its separate interrupt control configuration. Inside a FIFO buffer, all elements must be

- assigned to the same CAN node (control bit NODE in register MSGCFGn),
- programmed for the same transfer direction (control bit DIR),
- set up to the same identifier length (control bit XTD),
- programmed to the same FIFO length (bitfield FSIZEn) and
- set up with the same value for the FIFO direction (bit FD in register MSGFGCRn).
- The slave’s CANPTR has to point to the FIFO base object.

The base object’s CANPTR has to be initialized with the message number of the base object, the CANPTR pointers of the slave objects have to be set up with the message number of the base object. The CANPTR of the base object addresses the next FIFO element to be accessed for information transfer and its value can be calculated according the following rule:

CANPTRn(new) := CANPTRn(old) & ~FSIZEn | (CANPTRn(old) + 1) & FSIZEn

Control bit FD defines which transfer action (reception or transmission) leads to an update of the CANPTR bitfield. Bit FD works independently from the direction bit DIR of the FIFO elements. The reception of a data frame (DIR = ‘0’) or the reception of a remote frame (DIR = ‘1’) are receive actions leading to an update of CANPTR if FD = ‘0’. The transmission of a data frame (DIR = ‘1’) or the transmission of a remote frame (DIR = ‘0’) are transmit actions initiating an increment of CANPTR if FD = ‘1’.

Note: The overall message object storage size is not affected by the configuration of buffer structures. The available storage size may be used for 32 message objects without buffering or for one message object with a buffer depth of 32 elements. Additionally, any combination of buffered and unbuffered message objects

according to the FIFO rules is allowed as long as the limit of 32 message objects is not exceeded.

21.1.5.1 Buffer Access by the CAN Controller

The data transfer between the message buffer and the CAN bus is managed by the associated CAN controller. Each buffer is controlled by a FIFO algorithm (First In, First Out = First Overwritten) storing messages, delivered by the CAN controller, in a circular order.

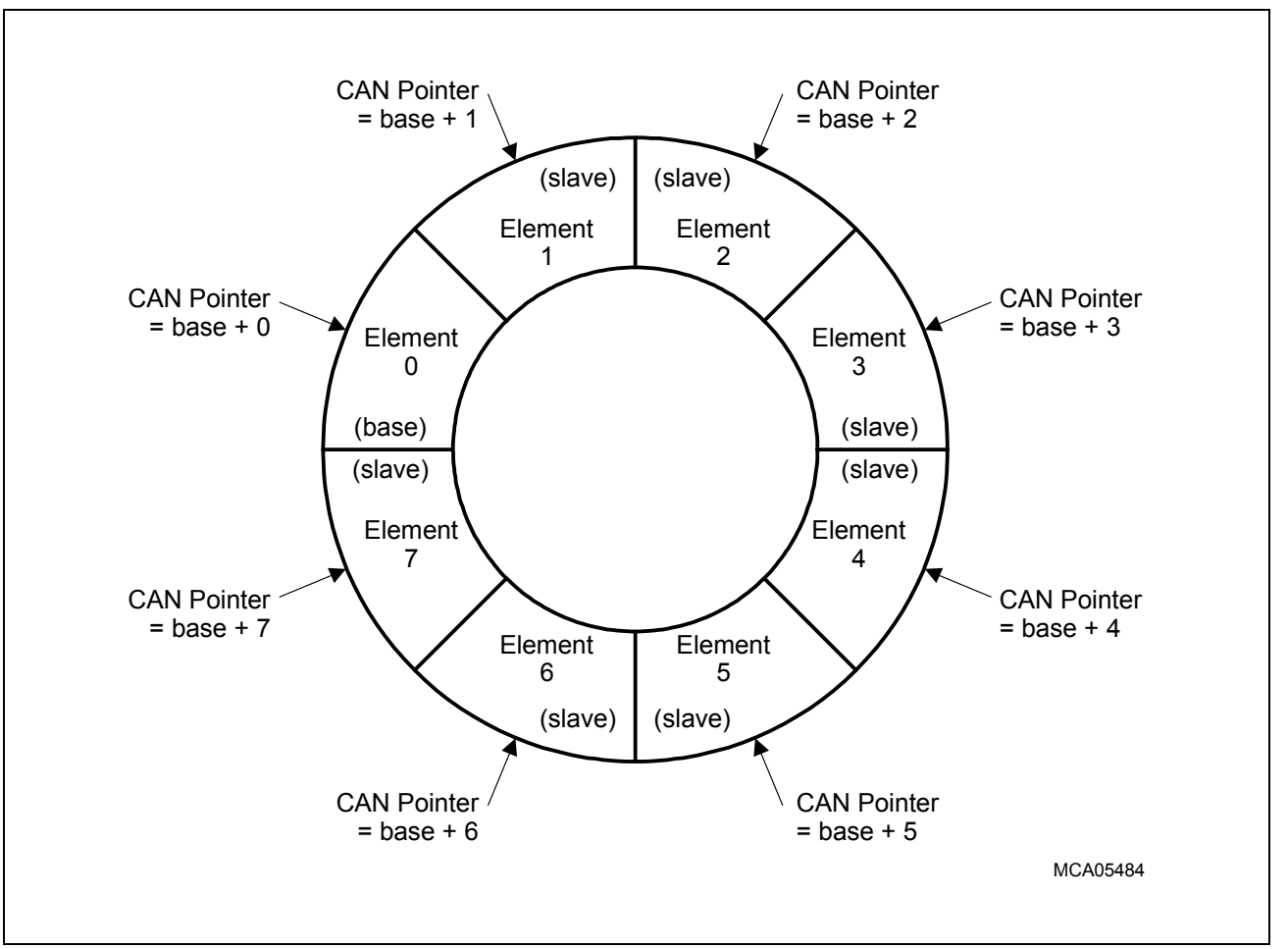


Figure 21-14 Structure of a FIFO Buffer with one Base Object and Seven Slave Objects

If the FIFO buffer was initialized with receive objects, the first accepted message is stored in the base message object (number n), the second message is written to buffer element (n+1) and so on. The number of the element, used to store the next input message, is indicated by bitfield CANPTR in control register MSGFGCRn of the base object. If the reserved buffer space has been used up, the base message object (followed by the consecutive slave objects) is addressed again to store the next incoming message. When a message object was not read out on time by the CPU, the previous

message data is overwritten, which is indicated by flag MSGLST in the corresponding MSGCTR register.

If the FIFO buffer was initialized with transmit message objects, the CAN controller starts the transfer with the contents of buffer element 0 (FIFO base object) and increments bitfield CANPTR in control register MSGFGCRn, pointing to the next element to be transmitted.

If the message object, which is currently addressed by the base object's CANPTR, is not valid (MSGVAL = '01'), the FIFO is not enabled for data transfer. In this case, the MSGVAL bitfields of the other FIFO elements (including the base element if not currently addressed) are not taken into account.

In the case that the MSGVAL bitfields are set to '10' for the FIFO base object and '01' for the currently addressed FIFO slave object, the data will not be delivered to the slave object, whereas the bitfield CANPTR in the FIFO base object is incremented according to FIFO rules.

If the FIFO is set up for the transmission of data frames and a matching remote frame is detected for one of the elements of the FIFO, the transmit request and remote pending bits will be set automatically in the corresponding message object. The transmission of the requested data frame is handled according to the FIFO rules and the value of the CANPTR bitfield in the FIFO base object.

21.1.5.2 Buffer Access by the CPU

The message transfer between a buffer and the CPU has to be managed by software. All message objects, combined to a buffer, can be accessed directly by the CPU. Bitfield CANPTR in control register MSGFGCRn is not automatically modified by a CPU access to the message object registers.

21.1.6 Gateway Message Handling

The CAN module supports an automatic information transfer between two independent CAN bus systems without CPU interaction.

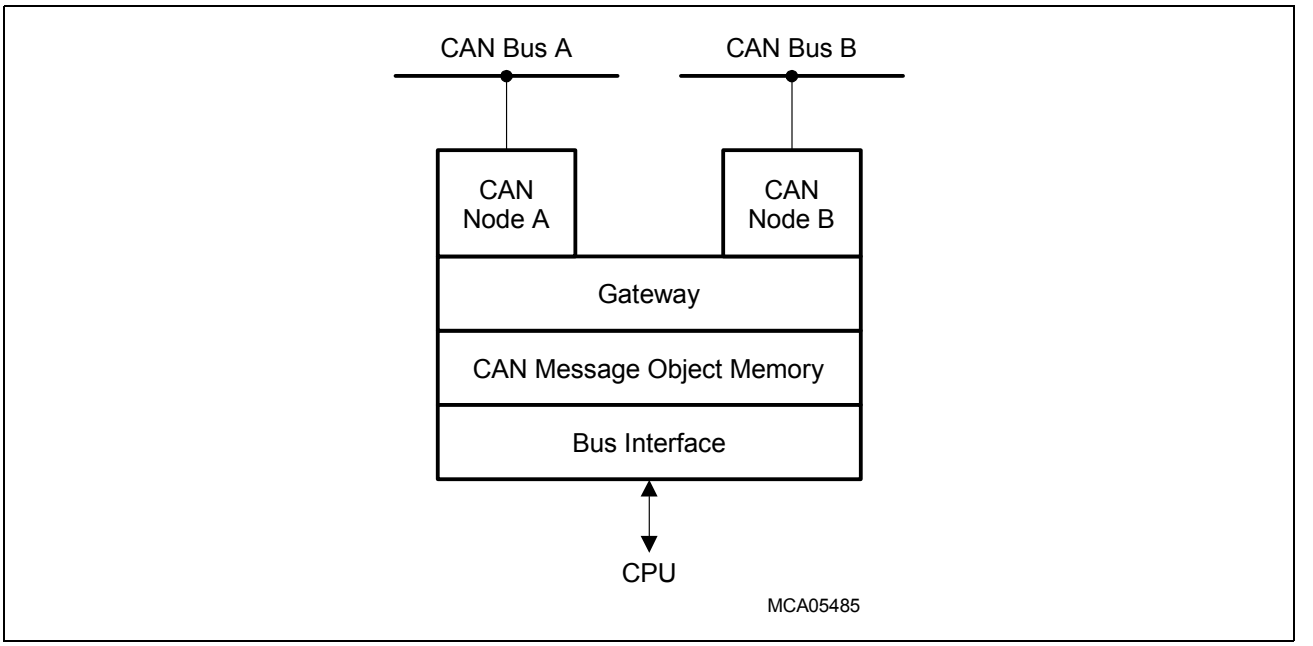


Figure 21-15 TwinCAN Gateway Functionality

The gateway functionality is handled via the CAN message object memory shared by both CAN nodes. Each object stored in the message memory is associated to node A or to node B via bit NODE in the message configuration register MSGCFGn. The information exchange between both CAN nodes can be handled by coupling two message objects (normal gateway mode) or by sharing one common message object (shared gateway mode).

In the following paragraphs, the gateway side receiving data frames is named “source” (indicated by <s>) and the side transmitting the data frames, which passed the gateway, is called “destination” (indicated by <d>). In concordance to this notation, remote frames passing the gateway are received on the destination side and transmitted on the source side.

The gateway function of a message object and the requested information transfer mode are defined by bitfield MMC in the FIFO/Gateway control register MSGFGCRn.

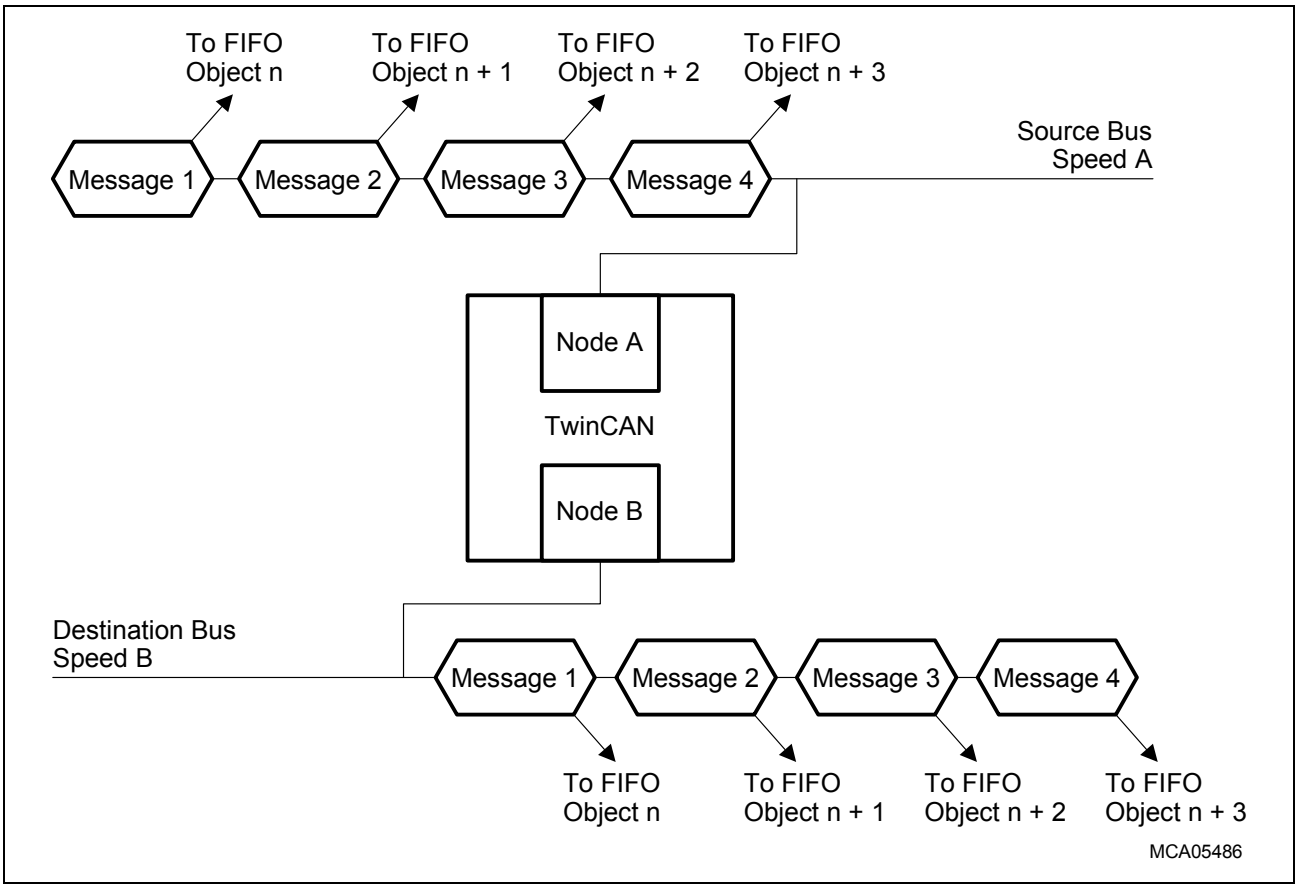


Figure 21-16 Message Burst in Case of FIFO/Gateway

21.1.6.1 Normal Gateway Mode

The normal gateway mode consumes two message objects to transfer a message from the source to the destination node. In this mode, different identifiers can be used for the same message data. Details of the message transfer through the normal gateway are controlled by the respective $MSGFGCR_{<s>}$ and $MSGFGCR_{<d>}$ registers. All 8 data bytes from the source object (even if not all bytes are valid) are copied to the destination object.

The object receiving the information from the source node has to be configured as receive message object (DIR = 0) and must be associated to the source CAN bus via bit NODE. Register $MSGFGCR_{<s>}$ should be initialized according the following enumeration:

- Bitfield $MMC_{<s>}$ has to be set to ‘100’ indicating a normal mode gateway for incoming (data) frames.
- Bitfield $CANPTR_{<s>}$ must be initialized with the number of the message object used as destination for the data copy process.
- If no FIFO functionality is required on the destination side, bitfield $FSIZE_{<s>}$ has to be filled with ‘00000’. When FIFO capabilities are needed, bitfield $FSIZE_{<s>}$ must contain the FIFO buffer length, which has to be identical with the content of the FIFO base object’s FSIZE bitfield on the destination side.

- When bit $IDC_{<s>}$ is set, the identifier of the source message object is copied to the destination message object. Otherwise, the identifier of the destination message object is not modified.
- If $DLCC_{<s>}$ is set, the data length code of the source message is copied to the destination object.
- Bit $GDFS_{<s>}$ decides, whether the transmit request flag on the destination side is set ($TXRQ_{<d>}$ = ‘10’ if $GDFS_{<s>}$ = ‘1’) after finishing the data copy process. An automatic transmission of the copied data frame on the destination side takes place, if control bit $CPUUPD_{<d>}$ is reset to ‘01’.

The destination message object, addressed by $CANPTR_{<s>}$, has to be configured for transmit operation (DIR = 1). Depending on the required functionality, the destination message object can be set up in three different operating modes:

- With $MMC_{<d>}$ = ‘000’, the destination message object is declared as standard message object. In this case, data frames, received on the source side, can be automatically emitted on the destination side if enabled by the respective control bits $CPUUPD_{<d>}$ and $GDFS_{<s>}$. Remote frames, received on the destination side, are not transferred to the source side, but can be directly answered by the destination message object if $CPUUPD_{<d>}$ is reset to ‘01’.
- With $MMC_{<d>}$ = ‘100’, the destination message object is declared as normal mode gateway for incoming (remote) frames. Data frames, received on the source side, can be automatically emitted on the destination side if enabled ($CPUUPD_{<d>}$, $GDFS_{<s>}$) and remote frames, received on the destination side, are transmitted on the source side if enabled by $SRREN_{<d>}$ = ‘1’.
- With $MMC_{<d>}$ = ‘01x’, the destination message object is set up as an element of a FIFO buffering the data frames transferred from the source side through the gateway. Remote frames, received on the destination side, are not transferred to the source side, but can be directly answered by the currently addressed FIFO element if $CPUUPD_{<d>}$ is reset (bits $SRREN_{<d>}$ have to be cleared).
- Remote frame handling is completely done on the destination side according to FIFO rules.

$MMC_{<d>}$ = '000':

The operation with a standard message object on the destination side is illustrated in Figure 21-17.

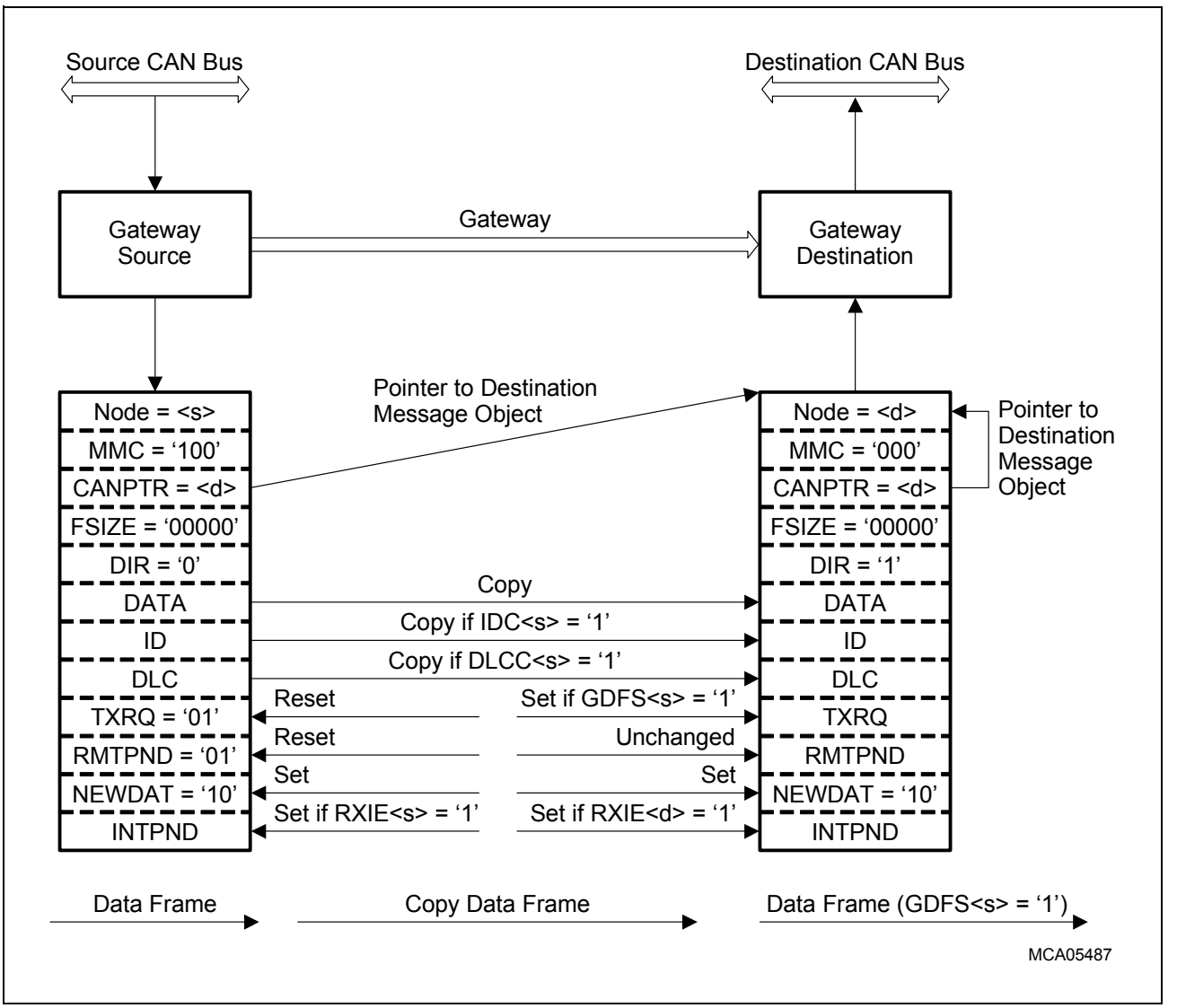


Figure 21-17 Data Frame Reception in Normal Gateway Mode with a Standard Destination Message Object ($MMC_{<d>}$ = '000')

A matching data frame, arrived at the source node, is automatically copied to the destination node's message object addressed by $CANPTR_{<s>}$. Bitfield $CANPTR_{<d>}$ is loaded with the destination message object number. Regardless of control bit $SRREN_{<d>}$, remote frames, received on the destination node, are not transferred to the source side, but can be directly answered by the destination message object. For this purpose, control bitfields $TXRQ_{<d>}$ and $RMTPND_{<d>}$ are set to '10', which immediately initiates a data frame transmission on the destination CAN bus if $CPUUPD_{<d>}$ is reset to '01'.

$MMC_{<d>}$ = '100':

The operation with a normal mode gateway message object for incoming (remote) frames on the destination side is illustrated in **Figure 21-18**.

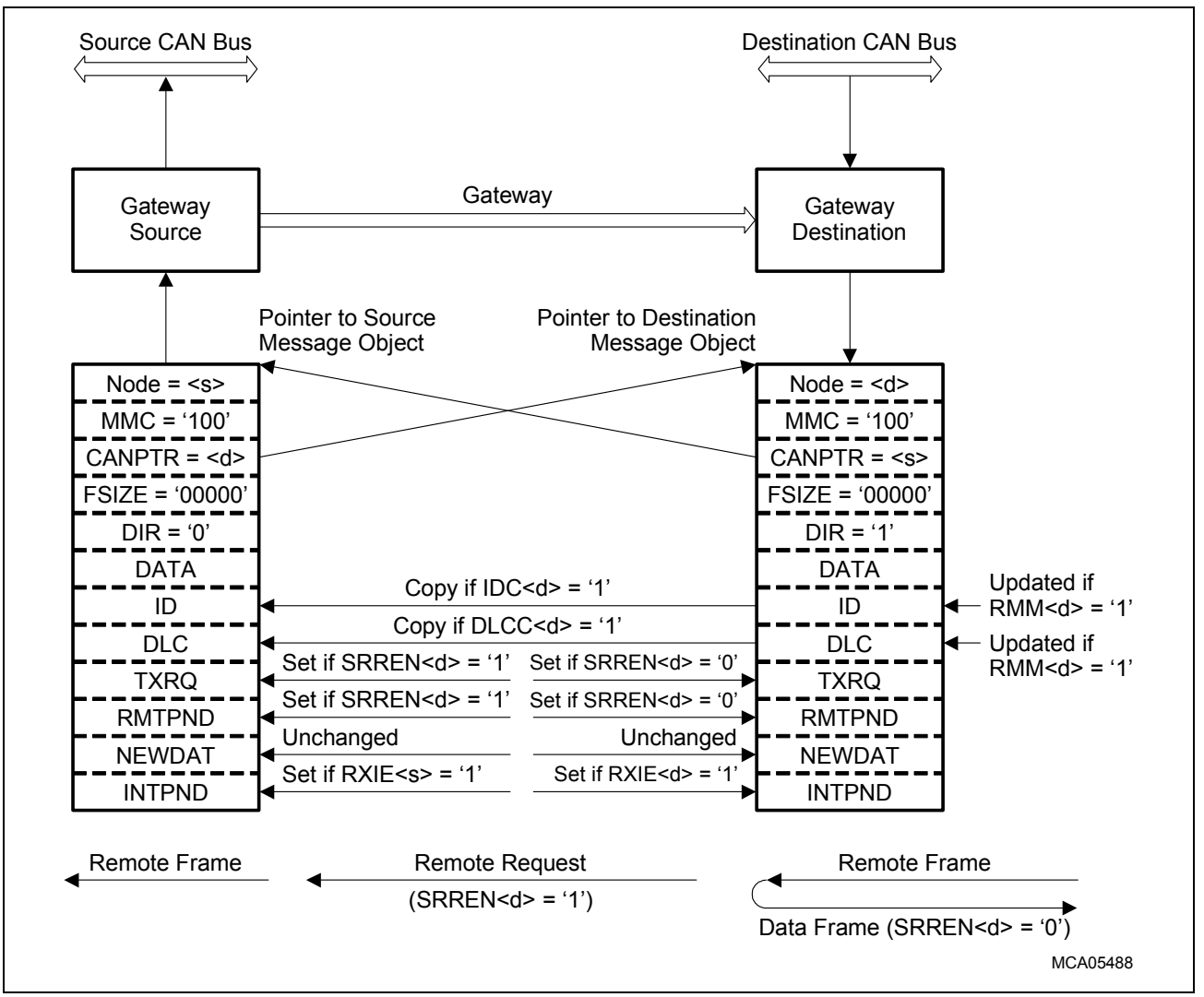


Figure 21-18 Remote Frame Transfer in Normal Gateway Mode, $MMC_{<d>}$ = '100'

The gateway object on the destination side, setup as transmit object, can receive remote frames. If bit $SRREN_{<d>}$ in the associated gateway control register MSGFGCRn is cleared, a remote frame with matching identifier is directly answered by the CAN destination node controller. For this purpose, control bits $TXRQ_{<d>}$ and $RMTPND_{<d>}$ are set to '10', which immediately initiates a data frame transmission on the destination CAN bus if $CPUUPD_{<d>}$ is reset. When bit $SRREN_{<d>}$ is set to '1', a remote frame received on the destination side is transferred via the gateway and transmitted again by the CAN source node controller.

A transmit request for the gateway message object on the source side, initiated by the CPU via setting $TXRQ_{<s>}$, generates always a remote frame on the source CAN bus system.

21.1.6.2 Normal Gateway with FIFO Buffering

$MMC_{<d>}$ = '01x':

When the gateway destination object is programmed as FIFO buffer, bitfield $CANPTR_{<s>}$ is used as pointer to the FIFO element to be addressed as destination for the next copy process. $CANPTR_{<s>}$ has to be initialized with the message object number of the FIFO base element on the destination side. $CANPTR_{<s>}$ is automatically updated according to the FIFO rules, when a data frame was copied to the indicated FIFO element on the destination side. Bit $GDFS_{<s>}$ determines if the $TXRQ_{<d>}$ bit in the selected FIFO element is set after reception of a data frame copied from the source side.

The base message object is indicated by <ba>, the slave message objects by <sl>. The number of base and slave message objects, combined to a buffer on the destination side, has to be a power of two (2, 4, 8 etc.) and the buffer base address has to be an integer multiple of the buffer length. Bitfield $CANPTR_{<ba>}$ of the FIFO base element and bitfield $CANPTR_{<s>}$ have to be initialized with the same start value (message object number of the FIFO base element). $CANPTR_{<sl>}$ of all FIFO slave elements must be initialized with the message object number of the FIFO base element. Bitfield $FSIZE_{<d>}$ of all FIFO elements must contain the FIFO buffer length and has to be identical with the content of $FSIZE_{<s>}$.

Figure 21-19 illustrates the operation of a normal gateway with a FIFO buffer on the destination side.

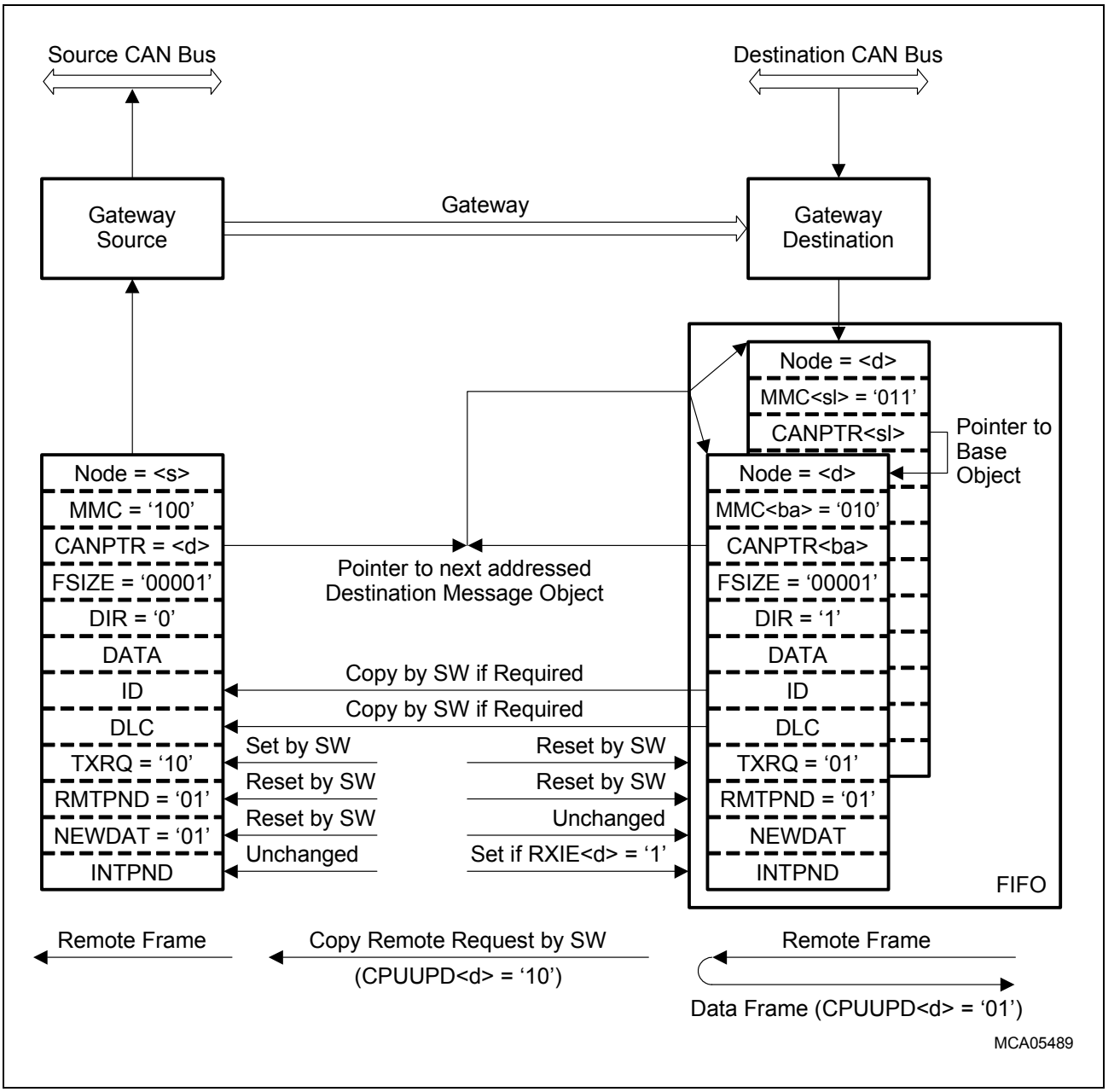


Figure 21-19 Data Frame Transfer in Normal Gateway Mode with a 2 Stage FIFO on the Destination Side ($MMC_{<d>}$ = '01x')

Remote frames, received on the destination side by a FIFO element, cannot be automatically passed to the source side. Therefore, the $SRREN_{<d>}$ control bits, associated to the FIFO elements on the destination side, have to be cleared in order to answer incoming remote frames with matching identifiers directly with a data frame.

Buffered transfers of remote requests from the destination to the source side can be handled by a software routine operating on the FIFO buffered gateway configuration for data frame transfers. The elements of the FIFO buffer on the destination side should be configured as transmit message objects with $CPUUPD_{<d>}$ = '10'. An arriving remote

frame with matching identifier should initiate an interrupt service request for the addressed FIFO message object. The associated interrupt service routine may copy the message identifier and the data length code from the received remote frame to a receive message object linked with the source side CAN node. In any case, TXRQ of the selected receive message object must be set to ‘10’ initiating the transmission of a remote frame on the source side.

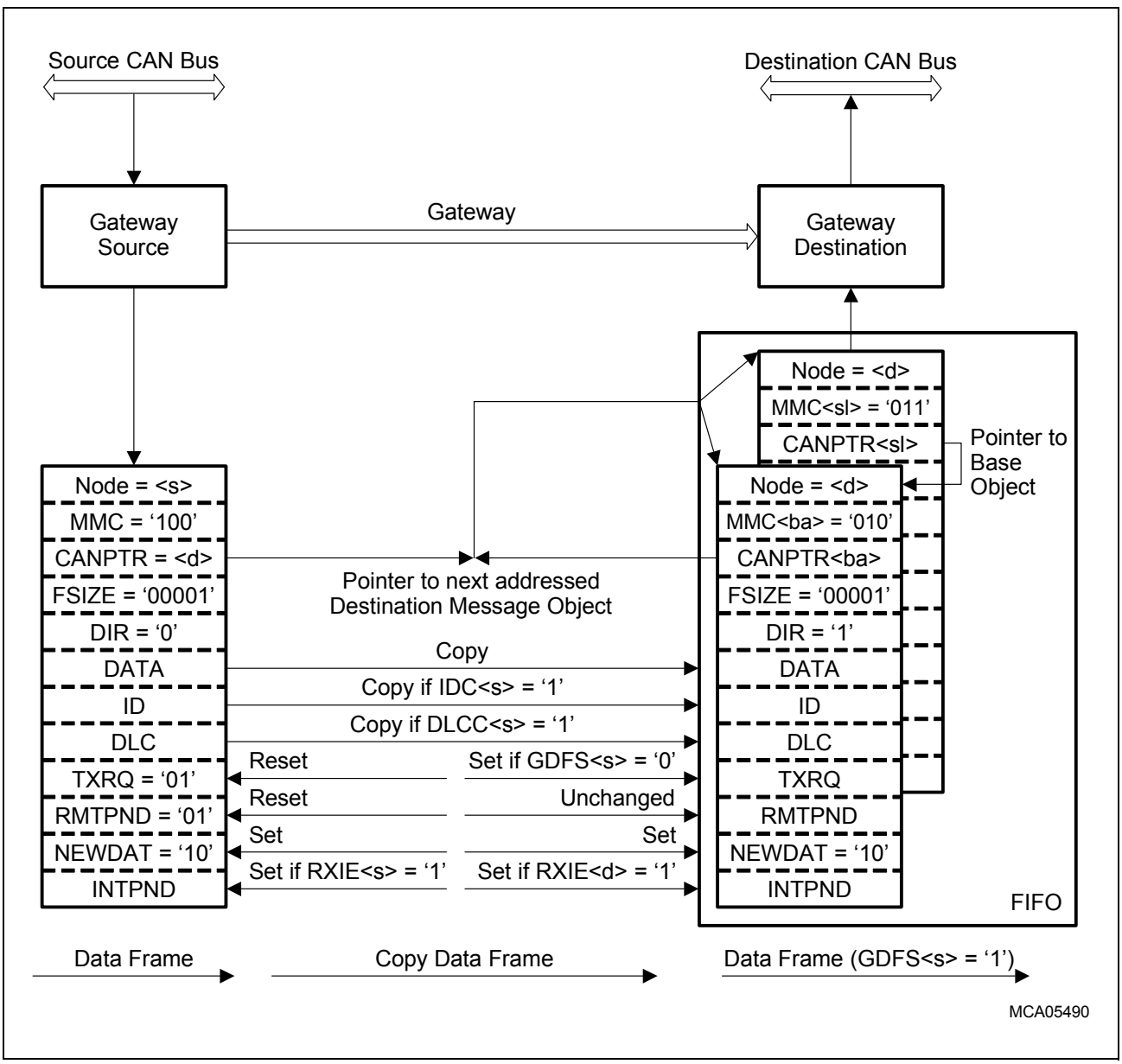


Figure 21-20 Remote Frame Transfer in Normal Gateway Mode with a 2-stage FIFO on the Destination Side

21.1.6.3 Shared Gateway Mode

In shared gateway mode, only one message object is required to implement a gateway functionality. The shared gateway object can be considered as normal message object, which is toggled between the source and destination CAN node as illustrated in Figure 21-21.

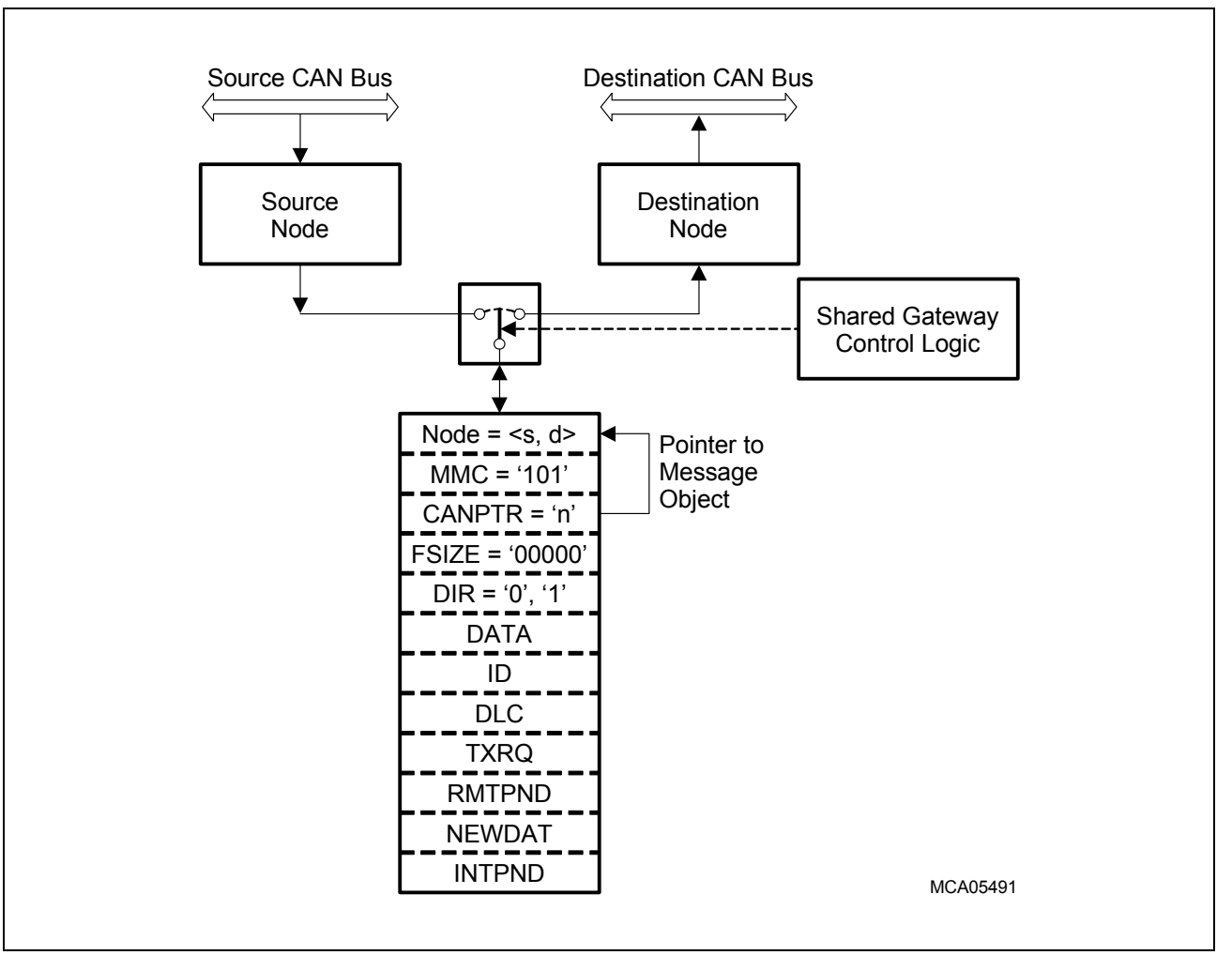


Figure 21-21 Principle of the Shared Gateway Mode

Each message object can be used as shared gateway by setting MMC in the corresponding MSGFGCRn register to '101'. When the message configuration bit NODE is cleared, CAN node A is used as source, transferring data frames to destination node B. If NODE is set to '1', CAN node B operates as data source. A bidirectional gateway is achieved by using a second message object, configured to shared gateway mode with a complementary NODE declaration. Bitfield CANPTR has to be initialized with the shared gateway's message object number, whereas FSIZE, IDC and DLCC have to be cleared. Bit GDFS in control register MSGFGCRn determines, whether bit TXRQ will be automatically set in case of an arriving data frame with matching identifier (GDFS = '1').

Bit SRREN determines, whether a remote frame, received on the destination side, is transferred through the gateway to the source node or is directly answered by a data frame generated on the destination side.

The functionality of the shared gateway mode is optimized to support different scenarios:

- a data source, connected with CAN node A, transmits continuously data frames, which have to be automatically emitted on the destination CAN bus by CAN node B. The corresponding transfer state transitions are 1 - 2 - …
- a data source, connected with CAN node A, transmits continuously data frames, which have to be emitted by CAN node B upon a matching remote frame received from the destination CAN bus.
  The corresponding transfer state transitions are 7 - 4 - 2 - …
- a data source, connected with CAN node A, transmits a data frame upon a matching remote frame, which has been triggered by a matching remote frame received by CAN node B. The respective data frame has to be emitted again on the destination CAN bus by CAN node B.
  The corresponding transfer state transitions are 5 - 6 - 1 - 3 - …

Depending on the application, the shared gateway message object can be initialized as receive object on the source side or transmit object on the destination side via an appropriate configuration of NODE, DIR, GDFS and SRREN. The various transfer states are illustrated in **Figure 21-22**.

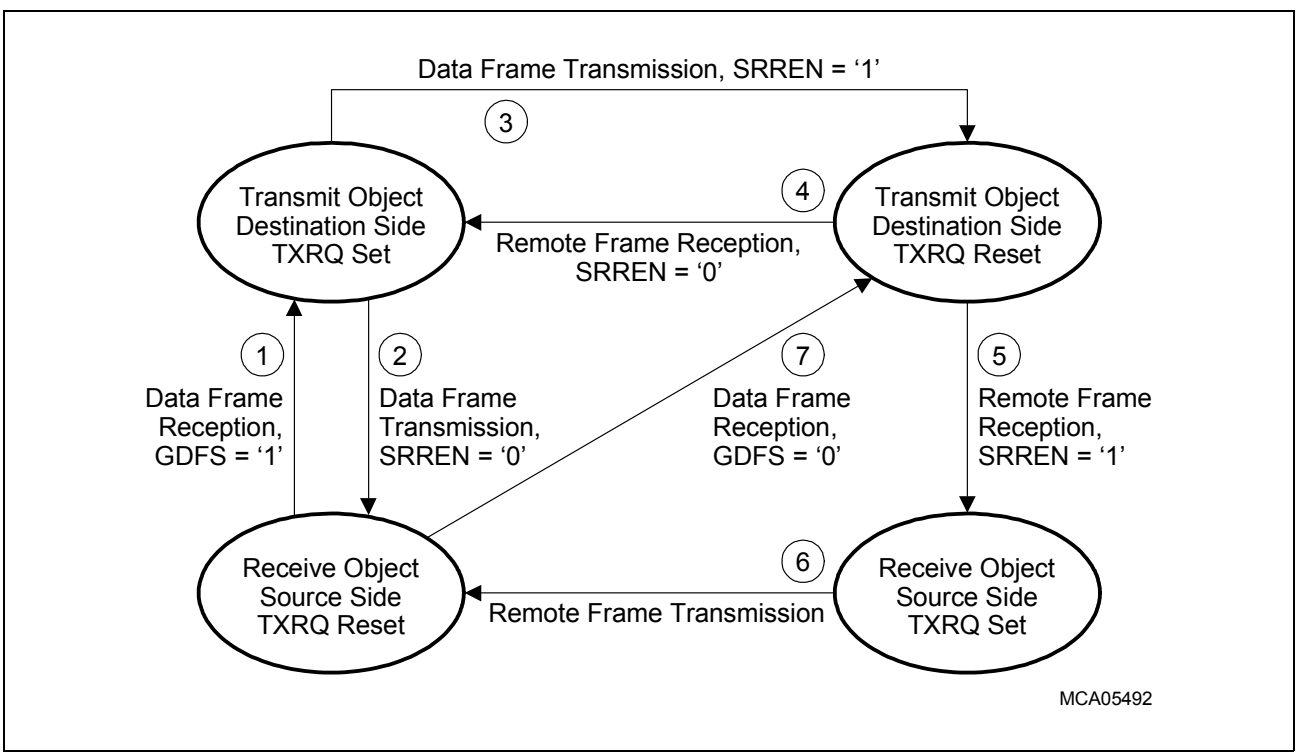


Figure 21-22 Transfer States in Shared Gateway Mode

When a shared gateway message object, set up as receive object on the source side (lower left state bubble in **Figure 21-22**), receives a data frame while GDFS is set to ‘1’, it commutes to a transmission object on the destination side by toggling control bits NODE and DIR and sends the corresponding data frame without any CPU interaction (upper left state bubble).
Depending on control bit SRREN, the shared gateway message object returns to its initial function as receive object assigned to the source side (SRREN = ‘0’: state transition 2 to the lower left state bubble in **Figure 21-22**) or remains assigned to the destination side waiting for a remote frame with matching identifier (SRREN = ‘1’: state transition 3 to the upper right state bubble).
When the shared gateway message object is assigned as transmit object to the destination side (upper right state bubble), it responds to remote frames received on the destination side. If bit SRREN is cleared, the remote request is directly answered by a data frame based on the contents of the gateway message object (state transition 4 to the upper left state bubble).
If bit SRREN is set and a remote frame is received on the destination side, the shared gateway message object commutes to a receive object on the source side by toggling control bits NODE and DIR and prepares the emission of the received remote frame by setting TXRQ and RMTPND to ‘10’ (state transition 5 to the lower right state bubble).
Then the shared gateway message object emits the corresponding remote frame without any CPU interaction (state transition 6 to the lower left state bubble).
The gateway message object remains assigned to the source side until a data frame with matching identifier arrives (lower left state bubble). Then the shared gateway message object returns to the destination side and, depending on control bit GDFS, transmits immediately the corresponding data frame (GDFS = ‘1’, upper left state bubble) or waits upon an action of the CPU setting TXRQ to ‘10’ (GDFS = ‘0’: state transition 7 to the upper right state bubble). Alternatively, a remote frame with matching identifier, arriving on the destination side, may set TXRQ to ‘10’ and initiate the data frame transmission.

If a data frame arrives on the source side while the shared gateway object with matching identifier is switched to the destination side, the data frame on the source side gets lost. Due to the temporary assignment to the destination node, the shared gateway message object does not notice the data frame on the source node and is not able to report the data loss via control bitfield MSGLST = ‘10’. The probability for a data loss is enlarged, if the automatic data frame transmission on the destination side is disabled by GDFS = ‘0’. A corresponding behavior has to be taken into account for incoming remote frames on the destination bus.

Note: As long as bitfield MSGLST is activated, an incoming data frame cannot be automatically transmitted on the destination side. Due to the internal toggling of control bit DIR, the shared gateway object converts from receive to transmit operation and bitfield MSGLST is interpreted as CPUUPD = ‘10’ preventing the automatic transmission of a data frame.

Impact of the transfer state transitions on the bitfields in the message object in shared gateway mode:

Table 21-3 Shared Gateway State Transitions (Part 1 of 2)

Bitfields	Transition 1: Data Frame Received, GDFS = ‘1’	Transition 2: Data Frame Transmitted, SRREN = ‘0’	Transition 3: Data Frame Transmitted, SRREN = ‘1’	Transition 4: Remote Frame Received, SRREN = ‘0’
Node	toggled to <d>	toggled to <s>	unchanged	unchanged
DIR	set	reset	unchanged	unchanged
DATA	received	unchanged	unchanged	unchanged
Identifier	received	unchanged	unchanged	received if RMM = ‘1’
DLC	received	unchanged	unchanged	received if RMM = ‘1’
TXRQ	set	reset	reset	set
RMTPND	reset	reset	reset	set
NEWDAT	set	reset	reset	reset
INTPND	set if RXIE = ‘10’	set if TXIE = ‘10’	set if TXIE = ‘10’	set if RXIE = ‘10’

Table 21-4 Shared Gateway State Transitions (Part 2 of 2)

Bitfields	Transition 5: Remote Frame Received, SRREN = ‘1’	Transition 6: Remote Frame Transmitted	Transition 7: Data Frame Received, GDFS = ‘0’
Node	toggled to <s>	unchanged	toggled to <d>
DIR	reset	unchanged	set
DATA	unchanged	unchanged	received
Identifier	received if RMM = ‘1’	unchanged	received
DLC	received if RMM = ‘1’	unchanged	received
TXRQ	set	reset	reset
RMTPND	reset	reset	reset
NEWDAT	unchanged	unchanged	set
INTPND	set if RXIE = ‘10’	set if TXIE = ‘10’	set if RXIE = ‘10’

21.1.7 Programming the TwinCAN Module

A software initialization should be performed by setting bit INIT in the CAN node specific control register ACR/BCR to ‘1’. While bit INIT is set, all message transfers between the CAN controller and the CAN bus are disabled.

The initialization routine should process the following tasks:

- configuration of the corresponding node,
- initialization of each associated message object.

21.1.7.1 Configuration of CAN Node A/B

Each CAN node can be individually configured by programming the associated register. Depending on the content of the ACR/BCR control registers, the normal operation mode or the CAN analyzer mode is activated. Furthermore, various interrupt categories (status change, error, last error) can be enabled or disabled.
The bit timing is defined by programming the ABTR/BBTR register. The prescaler value, the synchronization jump width and the time segments, arranged before and after the sample point, depend on the characteristic of the CAN bus segment linked to the corresponding CAN node.
The global interrupt node pointer register (AGINP/BGINP) controls multiplexer connecting an interrupt request source (error, last error, global transmit/receive and frame counter overflow interrupt request) with one of the eight common interrupt nodes. The contents of the INTID mask register (AIMR0/4 and BIMR0/4) decides which interrupt sources may be reported by the AIR/BIR interrupt pending register.

21.1.7.2 Initialization of Message Objects

The message memory space, containing 32 message objects, is shared by both CAN nodes. Each message object has to be configured concerning its target node and operation properties. An initialization of the message object properties is always started with disabling the message object via MSGVAL = ‘01’.
The CAN node, associated with a message, is defined by bit NODE in register MSGCFGn. The message object can be also defined as gateway, transferring information from CAN node A to B or vice versa. In this case, the FIFO/Gateway control register MSGFGCRn must be programmed to specify the gateway mode (bitfield MMC), the target interrupt node and further details of the information handover.
The identifier, correlated with a message, is set up in register MSGARn. Bit XTD in register MSGCFGn indicates, whether an extended 29-bit or a standard 11-bit identifier is used and has to be set accordingly. Incoming messages can be filtered by the mask defined in register MSGAMRn.
The message interrupt handling can be individually configured for transmit and receive direction. The direction specific interrupt is enabled by bits TXIE and RXIE in register MSGCNTn and the target interrupt node is selected by bitfields TXINP and RXINP in

register MSGCFGn.
Message objects can be provided with a FIFO buffer. The buffer size is determined by bitfield FSIZE in the FIFO/Gateway control register MSGFGCRn.
For transmit message objects, the object property assignment can be already finished by setting MSGVAL to ‘10’, before the corresponding data partition has been initialized. If bitfield CPUUPD is set to ‘10’, an incoming remote frame with matching identifier is kept in mind via setting TXRQ internally, but is not immediately answered by a corresponding data frame. The message data, stored in register MSGDRn0/MSGDRn4, can be updated as long as CPUUPD is hold on ‘10’. As soon as CPUUPD is reset to ‘01’, the respective data frame is transmitted by the associated CAN node controller.

21.1.7.3 Controlling a Message Transfer

Figure 21-23 illustrates the handling of a transmit message object. The initialization of the message object properties is always started with disabling the message object via MSGVAL = ‘01’. After resetting some control flags (INTPND, RMTPND, TXRQ and NEWDAT), the transfer direction and the identifier are defined. The message object initialization is finished by setting MSGVAL to ‘10’.
An update of a transmit message data partition should be prepared by setting CPUUPD to ‘10’ followed by a write access to the MSGDRn0/MSGDRn4 register. The data partition update must be indicated by the CPU via setting NEWDAT to ‘10’. Afterwards, bit CPUUPD must be reset to ‘01’, if an automatic message handling is requested. In this case, the data transmission is started, when flag TXRQ in register MSGCTRn has been set to ‘10’ by software or by the respective CAN node hardware due to a received remote frame with matching identifier. If CPUUPD remains set, the CPU must initiate the data transmission by setting TXRQ to ‘10’ and disabling CPUUPD. If a remote frame with an accepted identifier arrives during the update of a message object’s data storage, bit TXRQ and RMTPND are automatically set to ‘10’ and the transmission of the corresponding data frame is automatically started by the CAN controller when CPUUPD is reset again.
Figure 21-24 demonstrates the handling of a receive message object. The initialization of the message object properties is embedded between disabling and enabling the message object via MSGVAL as described above. After setting MSGVAL to ‘10’, the transmission of a remote frame can be initiated by the CPU via TXRQ = ‘10’. The reception of a data frame is indicated by the associated CAN node controller via NEWDAT = ‘10’. The processing of the received data frame, stored in register MSGDRn0/MSGDRn4, should be started by the CPU with resetting NEWDAT to ‘01’. After scanning flag MSGLST, indicating a loss of the previous message, the received information should be copied to an application data buffer in order to release the message object for a new data frame. Finally, NEWDAT should be checked again to ensure, that the processing was based on a consistent set of data and not on a part of an old message and part of the new message.

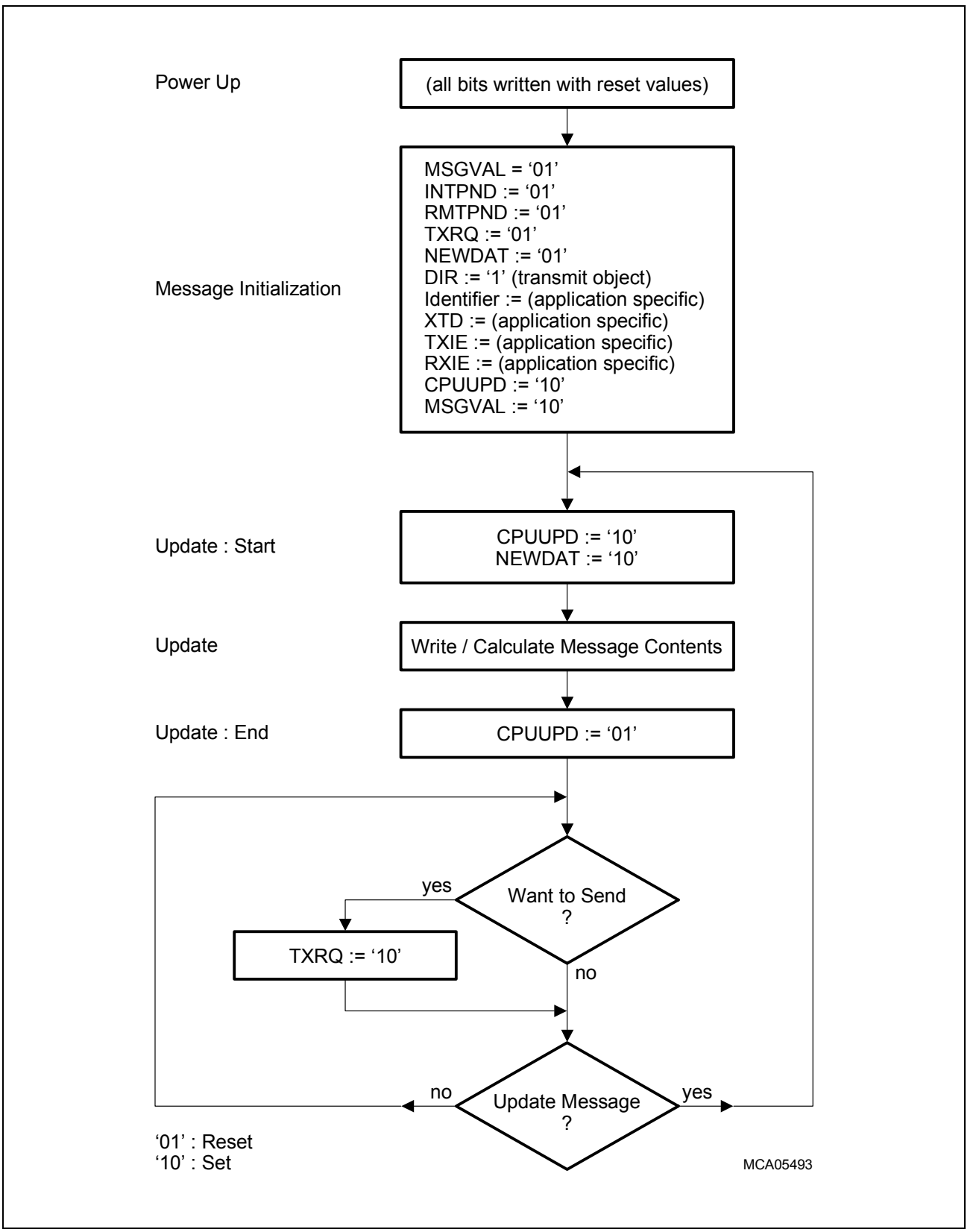


Figure 21-23 CPU Handling of Message Objects with Direction = Transmit

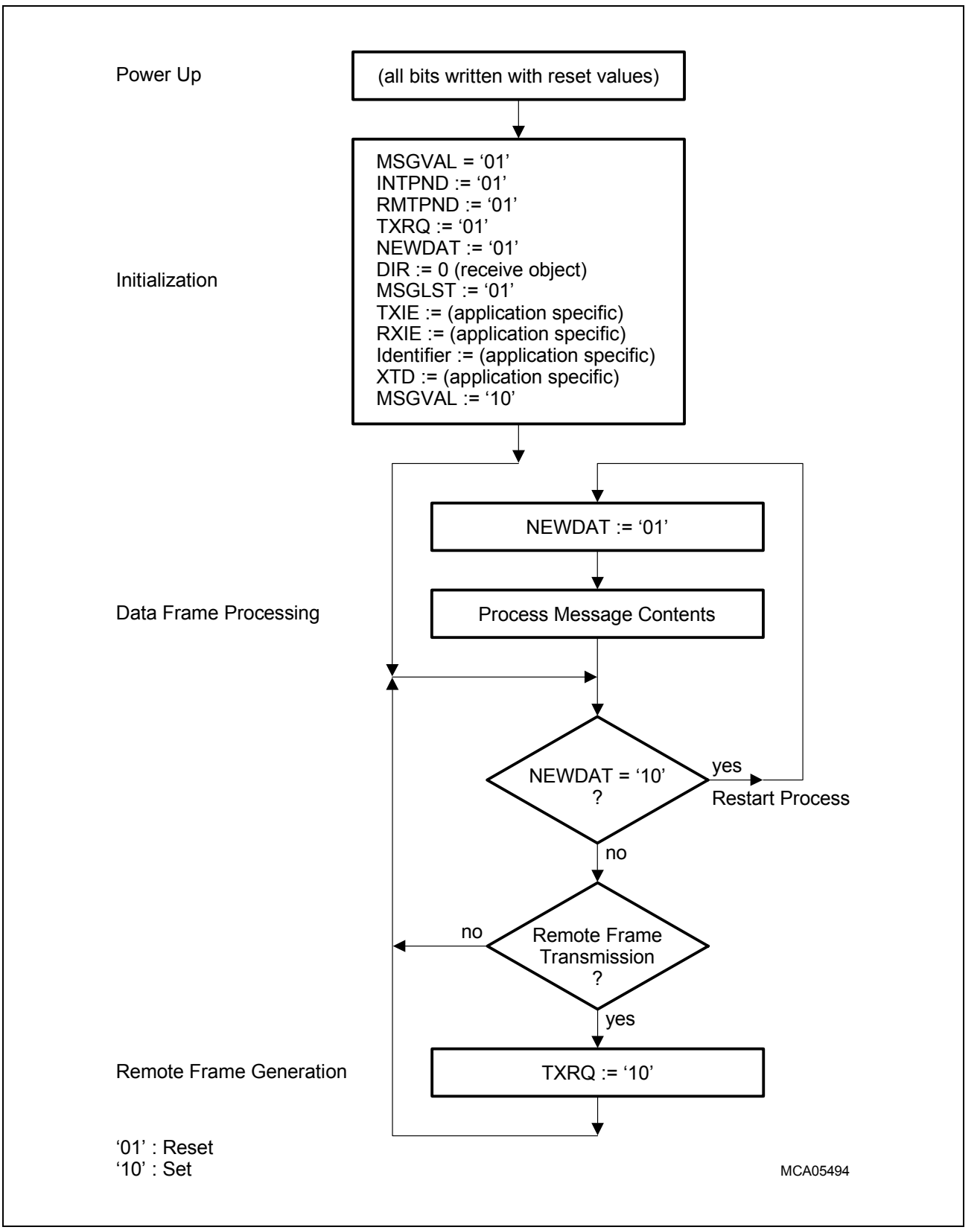


Figure 21-24 CPU Handling of Message Objects with Direction = Receive

21.1.8 Loop-Back Mode

The TwinCAN module's loop-back mode provides the means to internally test the TwinCAN module and CAN driver software. CAN driver software can be developed and tested without being connected to a CAN bus system.

In loop-back mode, the transmit pins deliver recessive signals to the transceiver. The transmit signals are combined together and are connected to the internal receive signals, as shown in **Figure 21-25**. The receive input pins are not taken into account in loop-back mode.

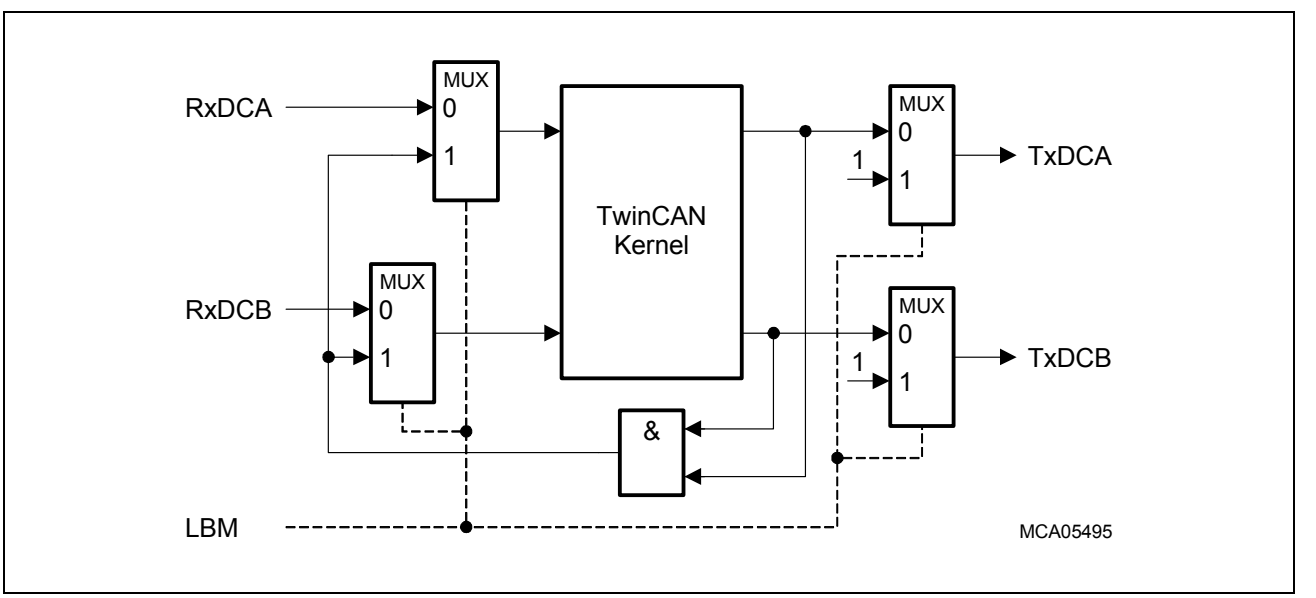


Figure 21-25 Loop-back Mode

Loop-back mode is controlled by bits LBM in the bit timing registers of Node A and Node B according to **Table 21-5**.

Table 21-5 Loop-Back Mode

ABTR.LBM	BBTR.LBM	Description
0	0	Loop-back mode is disabled.
0	1	Loop-back mode is disabled.
1	0	Loop-back mode is disabled.
1	1	Loop-back mode is enabled.

21.1.9 Single Transmission Try Functionality

Single transmission try functionality is controlled individually for each message object by bit STT in register MSGFGCRn. If the single transmission try functionality is enabled, the transmit request flag TXRQ is reset immediately after the transmission of a frame related to this message object has started. Thus, a transmit frame is only transferred once on the CAN bus, even if it has been corrupted by error frames.

Note: A message object must be tagged valid by bitfield MSGVAL in order to enable the transmission of the respective frame.

21.1.10 Module Clock Requirements

The functionality of the TwinCAN module is programmable in several respects. In order to operate at a specific baudrate with a given functionality a certain minimum module clock frequency is required. **Table 21-6** lists some examples for certain configurations. These examples cover the worst case conditions where the CPU executes accesses to the TwinCAN module consecutively and with maximum speed.

The module clock frequency can be reduced (see last column of **Table 21-6**) if no frames without data (data frames with DLC = 0 or remote frames) are transferred over the CAN bus. This is possible, because internal operations can be executed while the data part is transferred.

Table 21-6 Minimum Module Clock Frequencies for 1 Mbit/s

	1 Node Active, DLC ≥ 0	**2 Nodes Active, DLC ≥ 0**	**2 Nodes Active, DLC ≥ 1**
FIFO/gateway enabled	21 MHz	36 MHz	32 MHz
No FIFO/gateway	20 MHz	29 MHz	26 MHz

Note: The given numbers are required for the maximum CAN bus speed of 1 Mbit/s. For lower bit-rates the minimum module clock frequency can be reduced linearly, i.e. half the frequency is required for a bit-rate of 500 kbit/s.
However, if two nodes are operated with different bit-rates, the module clock frequency must be chosen according to the fastest node.

21.2 TwinCAN Register Description

21.2.1 Register Map

Figure 21-26 shows all registers associated with the TwinCAN module kernel.

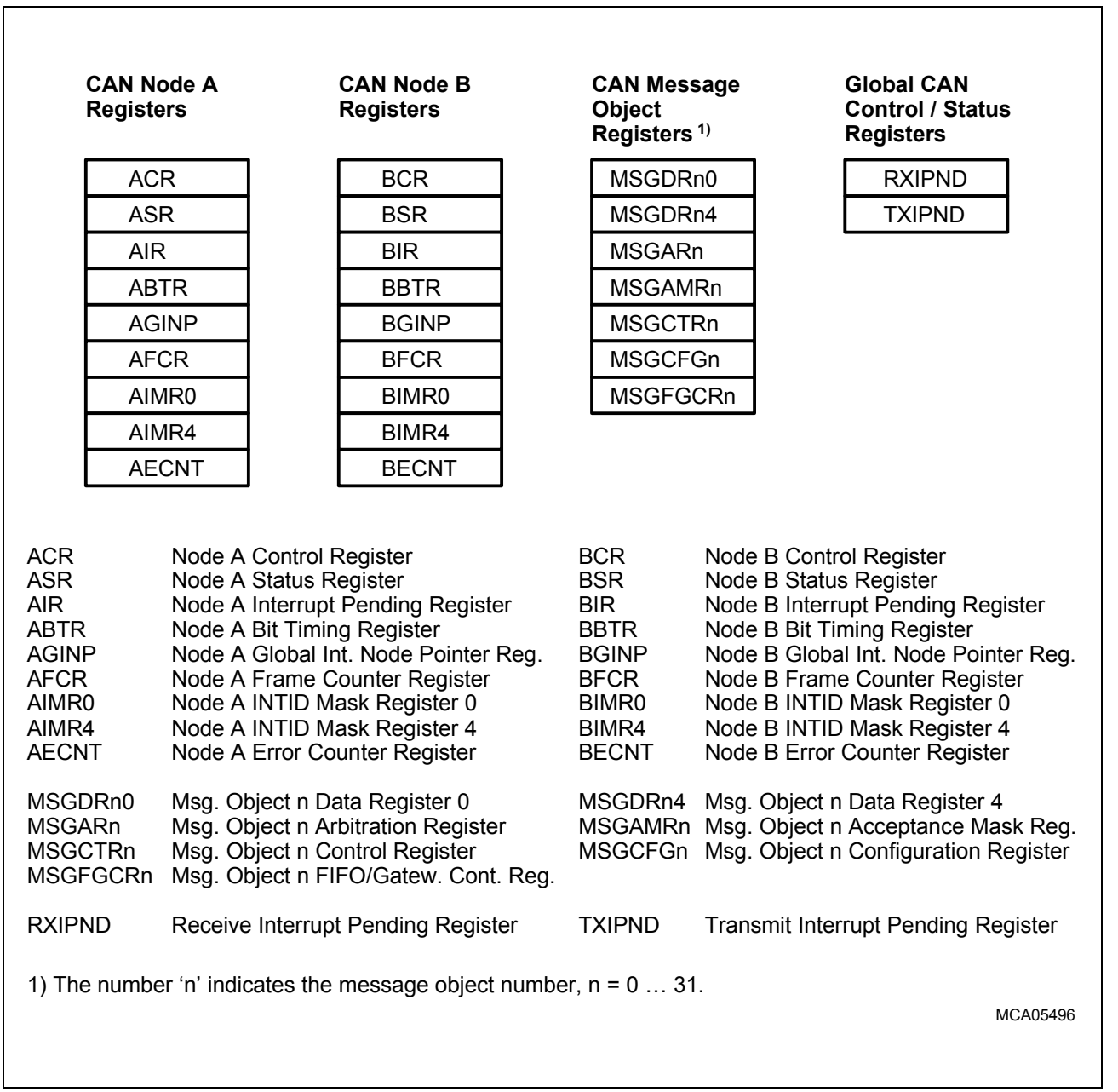


Figure 21-26 TwinCAN Kernel Registers

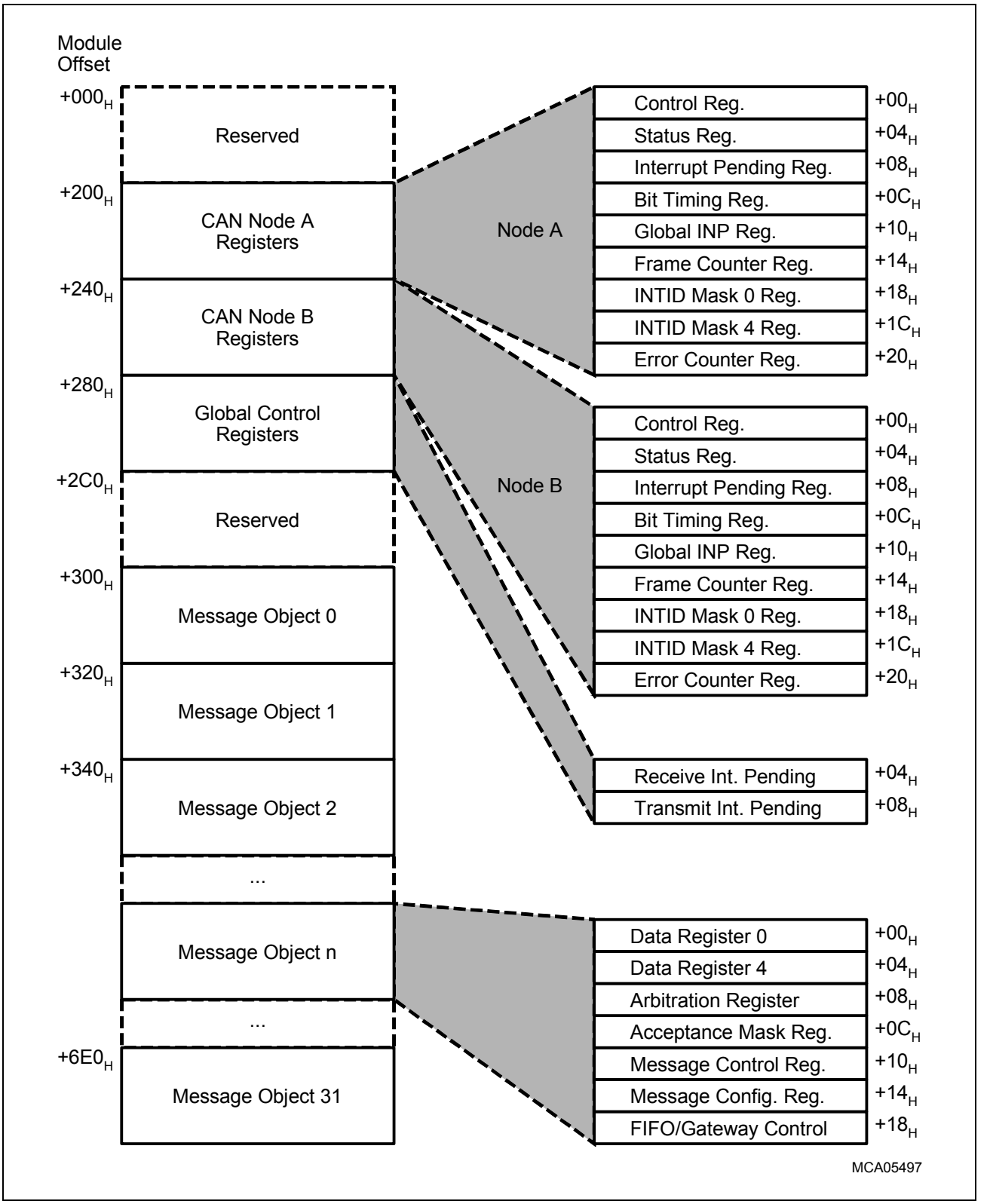


Figure 21-27 TwinCAN Kernel Address Map

21.2.2 CAN Node A/B Registers

The Node Control Register controls the initialization, defines the node specific interrupt handling and selects an operation mode.

ACR
Node A Control Register **Reset Value: 0001_H**
BCR
Node B Control Register **Reset Value: 0001_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0								CALM	CCE	0	LECIE	EIE	SIE	0	INIT
r								rw	rw	r	rw	rw	rw	r	rwh

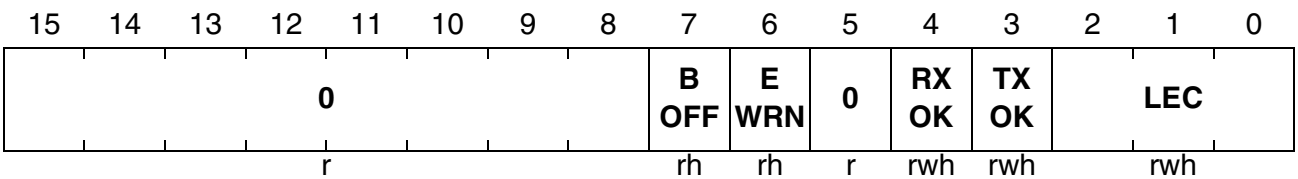
Field	Bits	Type	Description
INIT	0	rwh	**Initialization** 0 Resetting bit INIT starts the synchronization to the CAN bus. After a synchronization procedure[1)], the node takes part in CAN communication. 1 After setting bit INIT, the CAN node stops all CAN bus activities and all registers can be initialized without any impact on the actual CAN bus traffic. Bit INIT is automatically set when the bus-off state is entered.
SIE	2	rw	**Status Change Interrupt Enable** A status change interrupt occurs when a message transfer (indicated by the flags TXOK or RXOK in the status registers ASR or BSR) is successfully completed. 0 Status change interrupt is disabled. 1 Status change interrupt is enabled.
EIE	3	rw	**Error Interrupt Enable** An error interrupt is generated on a change of bit BOFF or bit EWRN in the status registers ASR or BSR. 0 Error interrupt is disabled. 1 Error interrupt is enabled.

Field	Bits	Type	Description
LECIE	4	rw	**Last Error Code Interrupt Enable** A last error code interrupt is generated when an error code is set in bitfield LEC in the status registers ASR or BSR. 0 Last error code interrupt is disabled. 1 Last error code interrupt is enabled.
CCE	6	rw	**Configuration Change Enable** 0 Access to bit timing register and modification of the error counters are disabled. 1 Access to bit timing register and modification of the error counters are enabled.
CALM	7	rw	**CAN Analyzer Mode** Bit CALM defines if the message objects of the corresponding node operate in analyzer mode. 0 The CAN message objects participate in CAN protocol. 1 CAN Analyzer Mode is selected.
0	1, 5, [15:8]	r	**Reserved;** returns ‘0’ if read; should be written with ‘0’.

1) After resetting bit INIT by software without being in the bus-off state (e.g. after power-on), a sequence of 11 consecutive recessive bits (11 × ‘1’) on the bus has to be monitored before the module takes part in the CAN traffic.
During a bus-off recovery procedure, 128 sequences of 11 consecutive recessive bits (11 × ‘1’) have to be detected. The monitoring of the recessive bit sequences is immediately started by hardware after entering the bus-off state. The number of already detected 11 × ‘1’ sequences is indicated by the receive error counter.
At the end of the bus-off recovery sequence, bit INIT is tested by hardware. If INIT is still set, the affected CAN node controller waits until INIT is cleared and 11 consecutive recessive bits (11 × ‘1’) are detected on the CAN bus, before the node takes part in CAN traffic again. If INIT has been already cleared, the message transfer between the affected CAN node controller and its associated CAN bus is immediately enabled.

The Node Status Register reports error states and successfully ended data transmissions. This register has to be read in order to release the status change interrupt request.

ASR
Node A Status Register **Reset Value: 0000_H**
BSR
Node B Status Register **Reset Value: 0000_H**



Field	Bits	Type	Description
LEC	[2:0]	rwh	**Last Error Code** Bitfield LEC indicates if the latest CAN message transfer has been correct (No Error) or it indicates the type of error, which has been detected. The error conditions are detailed in Table 21-7. 000 No Error 001 Stuff Error 010 Form Error 011 Ack Error 100 Bit1 Error 101 Bit0 Error 110 CRC Error 111 reserved
TXOK	3	rwh	**Message Transmitted Successfully** 0 No successful transmission since last flag reset. 1 A message has been transmitted successfully (error free and acknowledged by at least one other node). TXOK must be reset by software.
RXOK	4	rwh	**Message Received Successfully** 0 No successful reception since last flag reset. 1 A message has been received successfully. RXOK must be reset by software.

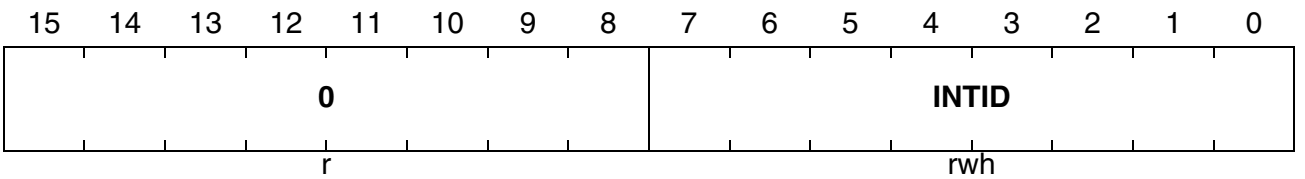
Field	Bits	Type	Description
EWRN	6	rh	**Error Warning Status** 0 No warning limit exceeded. 1 One of the error counters in the Error Management Logic reached the error warning limit of 96.
BOFF	7	rh	**Bus-Off Status** 0 CAN controller is not in the bus-off state. 1 CAN controller is in the bus-off state.
0	5, [15:8]	r	**Reserved;** returns ‘0’ if read; should be written with ‘0’.

Table 21-7 Meaning of the LEC Bitfield

LEC Error	Description
No Error	The latest transfer on the CAN bus has been completed successfully.
Stuff Error	More than 5 equal bits in a sequence have occurred in a part of a received message where this is not allowed.
Form Error	A fixed format part of a received frame has the wrong format.
Ack Error	The transmitted message was not acknowledged by another node.
Bit1 Error	During a message transmission, the CAN node tried to send a recessive level (‘1’), but the monitored bus value was dominant (outside the arbitration field and the acknowledge slot).
Bit0 Error	Two different conditions are signaled by this code: 1. During transmission of a message (or acknowledge bit, active error flag, overload flag), the CAN node tried to send a dominant level (‘0’), but the monitored bus value has been recessive. 2. During bus-off recovery, this code is set each time a sequence of 11 recessive bits has been monitored. The CPU may use this code as an indication, that the bus is not continuously disturbed.
CRC Error	The CRC checksum of the received message was incorrect.

The Interrupt Pending Register contains the identification number of the pending interrupt request with the highest priority.

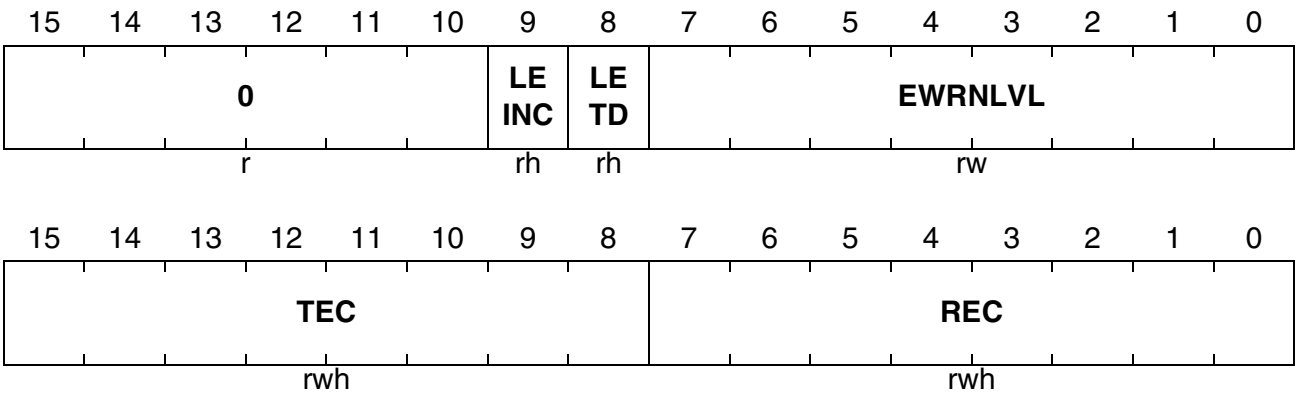
AIR
Node A Interrupt Pending Register **Reset Value: 0000 0000$_H$**
BIR
Node B Interrupt Pending Register **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
INTID	[7:0]	rwh	**Interrupt Identifier** 00_H No interrupt is pending. 01_H LEC, EI, TXOK or RXOK interrupt is pending. 02_H RX or TX interrupt of message object 0 is pending. 03_H RX or TX interrupt of message object 1 is pending. … … 21_H RX or TX interrupt of message object 31 is pending. Bitfield INTID can be written by software to start an update after software actions and to check for changes.
0	[15:8]	r	**Reserved;** returns ‘0’ if read.

Register AECNT/BECNT contains the values of the receive error counter and the transmit error counter. Some additional status/control bits allow for easier error analysis.

AECNTH
Node A Error Counter Register High **Reset Value: 0060_H**
AECNTL
Node A Error Counter Register Low **Reset Value: 0000_H**
BECNTH
Node B Error Counter Register High **Reset Value: 0060_H**
BECNTL
Node B Error Counter Register Low **Reset Value: 0000_H**



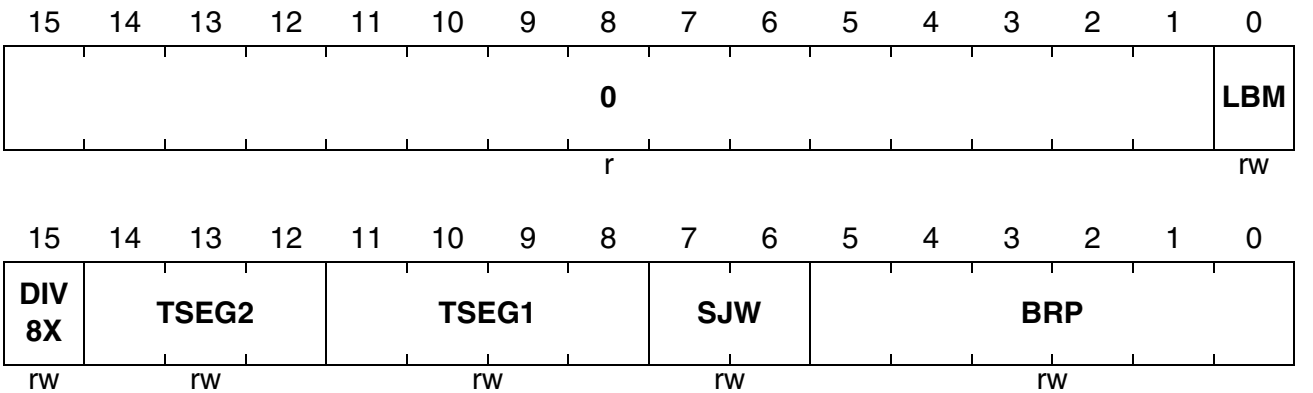
Field	Bits	Type	Description
REC	[7:0] Low	rwh	**Receive Error Counter** Bitfield REC contains the value of the receive error counter for the corresponding node.
TEC	[15:8] Low	rwh	**Transmit Error Counter** Bitfield TEC contains the value of the transmit error counter for the corresponding node.
EWRNLVL	[7:0] Low	rw	**Error Warning Level** Bitfield EWRNLVL defines the threshold value (warning level, default $60_H = 96_D$) to be reached in order to set the corresponding error warning bit EWRN.

Field	Bits	Type	Description
LETD	8 High	rh	**Last Error Transfer Direction** 0 The last error occurred while the corresponding CAN node was receiving a message (REC has been incremented). 1 The last error occurred while the corresponding CAN node was transmitting a message (TEC has been incremented). An error during message reception is indicated without regarding the result of the acceptance filtering.
LEINC	9 High	rh	**Last Error Increment** 0 The error counter was incremented by 1 due to the error reported by LETD. 1 The error counter was incremented by 8 due to the error reported by LETD.
0	[15:10] High	–	**Reserved;** returns ‘0’ if read; should be written with ‘0’.

Note: Modifying the contents of register AECNT/BECNT requires bit CCE = ‘1’ in register ACR/BCR.

The Bit Timing Register contains all parameters to adjust the data transfer baud rate and the bit timing.

ABTRH
Node A Bit Timing Register High **Reset Value: 0000_H**
ABTRL
Node A Bit Timing Register Low **Reset Value: 0000_H**
BBTRH
Node B Bit Timing Register High **Reset Value: 0000_H**
BBTRL
Node B Bit Timing Register Low **Reset Value: 0000_H**



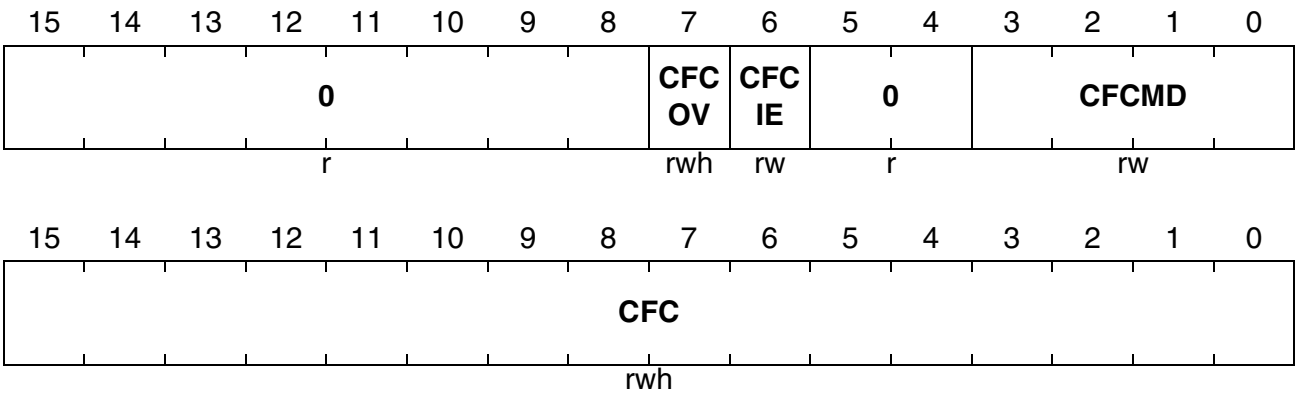
Field	Bits	Type	Description
BRP	[5:0] Low	rw	**Baudrate Prescaler** One bit time quantum corresponds to the period length of the external oscillator clock multiplied by (BRP+1), depending also on bit DIV8X.
SJW	[7:6] Low	rw	**(Re)Synchronization Jump Width** (SJW+1) time quanta are allowed for resynchronization.
TSEG1	[11:8] Low	rw	**Time Segment Before Sample Point** (TSEG1+1) time quanta before the sample point take into account the signal propagation delay and compensate a mismatch between transmitter and receiver clock phase. Valid values for TSEG1 are 2 … 15.

Field	Bits	Type	Description
TSEG2	[14:12] Low	rw	**Time Segment After Sample Point** (TSEG2+1) time quanta after the sample point take into account a user defined delay and compensate a mismatch between transmitter and receiver clock phase. Valid values for TSEG2 are 1 … 7.
DIV8X	15 Low	rw	**Division of Module Clock f_{CAN} by 8** 0 The baudrate prescaler is directly driven by f_{CAN}. 1 The baudrate prescaler is driven by $f_{CAN}/8$.
LBM	0 High	rw	**Loop-Back Mode** 0 Loop-back mode is disabled. 1 Loop-back mode is enabled, if bits LBM are set in the BTR registers of Node A and Node B.
0	[15:1] High	r	**Reserved;** read as ‘0’; should be written with ‘0’.

Note: Modifying the contents of register ABTR/BBTR requires bit CCE = ‘1’ in register ACR/BCR.

The Frame Counter Register controls the frame counter functionality and provides status information.

AFCRH
Node A Frame Counter Register High **Reset Value: 0000_H**
AFCRL
Node A Frame Counter Register Low **Reset Value: 0000_H**
BFCRH
Node B Frame Counter Register High **Reset Value: 0000_H**
BFCRL
Node B Frame Counter Register Low **Reset Value: 0000_H**



Field	Bits	Type	Description
CFC	[15:0] Low	rwh	**CAN Frame Counter** This bitfield contains the count value of the frame counter. At the end of a correct message transfer, the value of CFC (captured value during SOF bit) is copied to bitfield CFCVAL of the corresponding message object control register MSGCTRn.

Field	Bits	Type	Description
CFCMD	[3:0] High	rw	**Frame Count Mode** This bitfield defines the operation mode of the frame counter. This counter can work on frame base (frame count) or on time base (time stamp). $0XXX_B$ **Frame Count:**[1)] $0XX0_B$ The CFC is not incremented after a foreign frame was transferred on the CAN bus. $0XX1_B$ The CFC is incremented each time a foreign frame was transferred correctly on the CAN bus. $0X0X_B$ The CFC is not incremented after a frame was received by the respective CAN node. $0X1X_B$ The CFC is incremented each time a frame was received correctly by the node. $00XX_B$ The CFC is not incremented after a frame was transmitted by the node. $01XX_B$ The CFC is incremented each time a frame was transmitted correctly by the node. $1XXX_B$ **Time Stamp:** 1000_B The CFC is incremented with the beginning of a new bit time. The value is sampled during the SOF bit. 1001_B The CFC is incremented with the beginning of a new bit time. The value is sampled during the last bit of EOF. others reserved
CFCIE	6 High	rw	**CAN Frame Count Interrupt Enable** Setting CFCIE enables the CAN Frame Counter Overflow (CFCO) interrupt request. 0 The CFCO interrupt is disabled. 1 The CFCO interrupt is enabled.
CFCOV	7 High	rwh	**CAN Frame Count Overflow Flag** Flag CFCOV is set on a CFC overflow condition ($FFFF_H$ to 0000_H). An interrupt request is generated if the corresponding interrupt is enabled (CFCIE = ‘1’). 0 An overflow has not yet been detected. 1 An overflow has been detected since the bit has been reset. CFCOV must be reset by software.

Field	Bits	Type	Description
0	[5:4], [15:8] High	r	**Reserved;** read as ‘0’; should be written with ‘0’.

1) If the frame counter functionality has been selected (CFCMD.3 = ‘0’), bit CFCMD.0 enables or disables the counting of foreign frames. A foreign frame is a correct frame on the bus, which has not been transmitted /received by the node itself. Bit CFCMD.1 enables or disables the counting of frames, which have been received correctly by the corresponding CAN node. Bit CFCMD.2 enables or disables the counting of frames, which have been transmitted correctly by the corresponding CAN node.

The Global Interrupt Node Pointer Register connects each global interrupt request source with one of the 8 available CAN interrupt nodes.

AGINP
Node A Global Interrupt Node Pointer Register **Reset Value: 0000_H**
BGINP
Node B Global Interrupt Node Pointer Register **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**CFCINP**			**0**	**TRINP**			**0**	**LECINP**			**0**	**EINP**		
r	rw			r	rw			r	rw			r	rw		

Field	Bits	Type	Description
EINP	[2:0]	rw	**Error Interrupt Node Pointer** Number of interrupt node reporting the “Error Interrupt Request”, if enabled by EIE = ‘1’. 000_B CAN interrupt node 0 is selected. … … 111_B CAN interrupt node 7 is selected.
LECINP	[6:4]	rw	**Last Error Code Interrupt Node Pointer** Number of interrupt node reporting the last error interrupt request, if enabled by LECIE = ‘1’. 000_B CAN interrupt node 0 is selected. … … 111_B CAN interrupt node 7 is selected.
TRINP	[10:8]	rw	**Transmit/Receive OK Interrupt Node Pointer** Number of interrupt node reporting the transmit and receive interrupt request, if enabled by SIE = ‘1’. 000_B CAN interrupt node 0 is selected. … … 111_B CAN interrupt node 7 is selected.
CFCINP	[14:12]	rw	**Frame Counter Interrupt Node Pointer** Number of interrupt node reporting the frame counter overflow interrupt request, if enabled by CFCIE = ‘1’. 000_B CAN interrupt node 0 is selected. … … 111_B CAN interrupt node 7 is selected.
0	3, 7, 11, 15	–	**Reserved;** read as ‘0’; should be written with ‘0’.

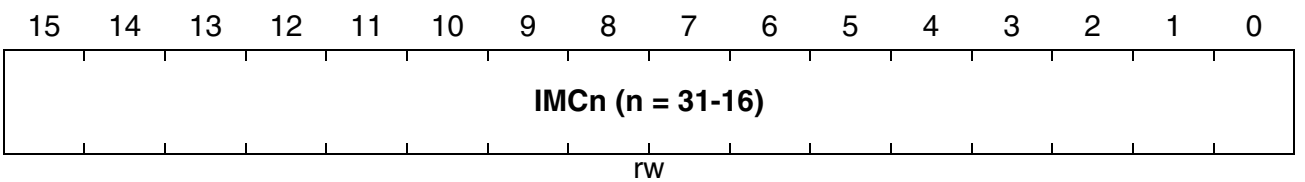
The Interrupt Identification Mask Registers allow for disabling the identification notification of a pending interrupt request in the AIR/BIR register. The Interrupt Mask Registers AIMR0/BIMR0 are used to enable the message specific interrupt sources (correct transmission/ reception) for the generation of the corresponding INTID value.

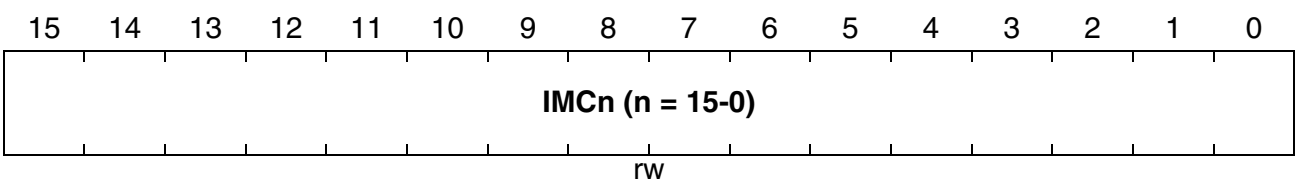
AIMRH0
Node A INTID Mask Register 0 High **Reset Value: 0000_H**

AIMRL0
Node A INTID Mask Register 0 Low **Reset Value: 0000_H**

BIMRH0
Node B INTID Mask Register 0 High **Reset Value: 0000_H**

BIMRL0
Node B INTID Mask Register 0 Low **Reset Value: 0000_H**

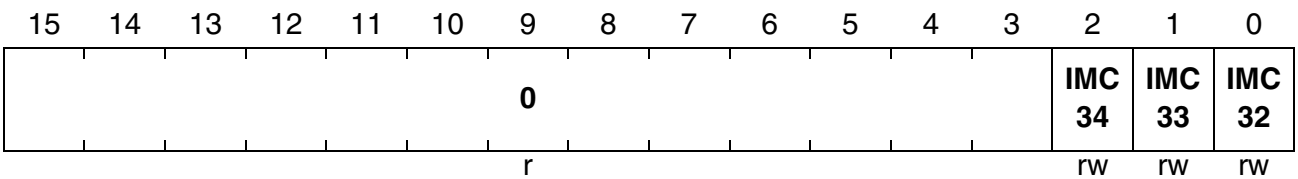




Field	Bits	Type	Description
IMCn (n = 15-0)	n Low	rw	**Message Object n INTID Mask Control** 0 Message object n is ignored for the generation of the INTID value.
IMCn (n = 31-16)	n-16 High		1 The interrupt pending status of message object n is taken into account for the generation of the INTID value.

The Interrupt Mask Registers AIMR4/BIMR4 are used to enable the node specific interrupt sources (last error, correct reception, error warning/bussoff) for the generation of the corresponding INTID value.

AIMR4
Node A INTID Mask Register 4 **Reset Value: 0000_H**
BIMR4
Node B INTID Mask Register 4 **Reset Value: 0000_H**

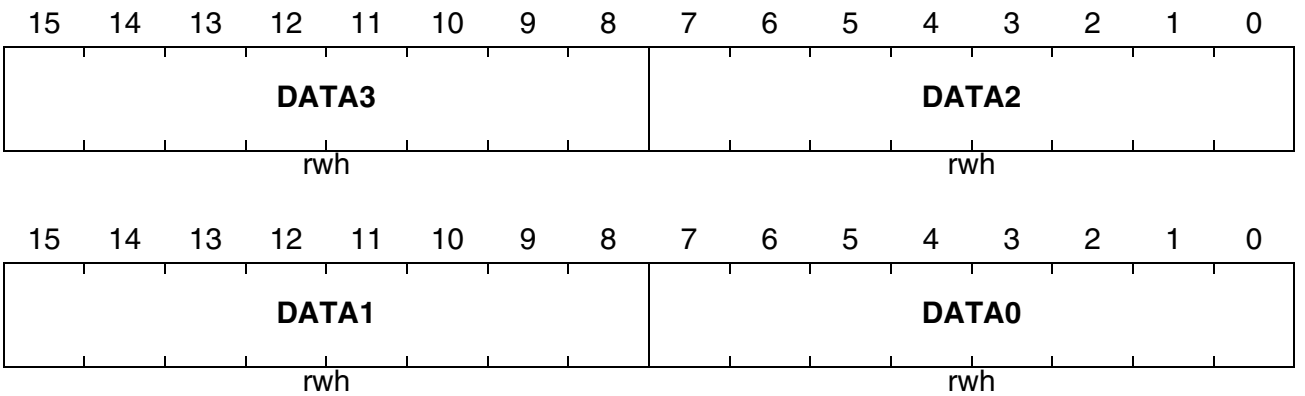


Field	Bits	Type	Description
IMC32	0	rw	**Last Error Interrupt INTID Mask Control** 0 The last error interrupt source is ignored for the generation of the INTID value. 1 The last error interrupt source is taken into account for the generation of the INTID value.
IMC33	1	rw	**TX/RX Interrupt INTID Mask Control** 0 The TX/RX interrupt source is ignored for the generation of the INTID value. 1 The TX/RX interrupt pending status is taken into account for the generation of the INTID value.
IMC34	2	rw	**Error Interrupt INTID Mask Control** 0 The error interrupt source is ignored for the generation of the INTID value. 1 The error interrupt pending status is taken into account for the generation of the INTID value.
0	[15:3]	r	**Reserved;** read as ‘0’; should be written with ‘0’.

21.2.3 CAN Message Object Registers

Each message object is provided with a set of control and data register. The corresponding register names are supplemented with a variable n running from 0 to 31 (e.g. MSGDRn0 means that data register MSGDR300 is assigned with message object number 30).
The Message Data Register 0 contains the data bytes 0 to 3 of message object n.

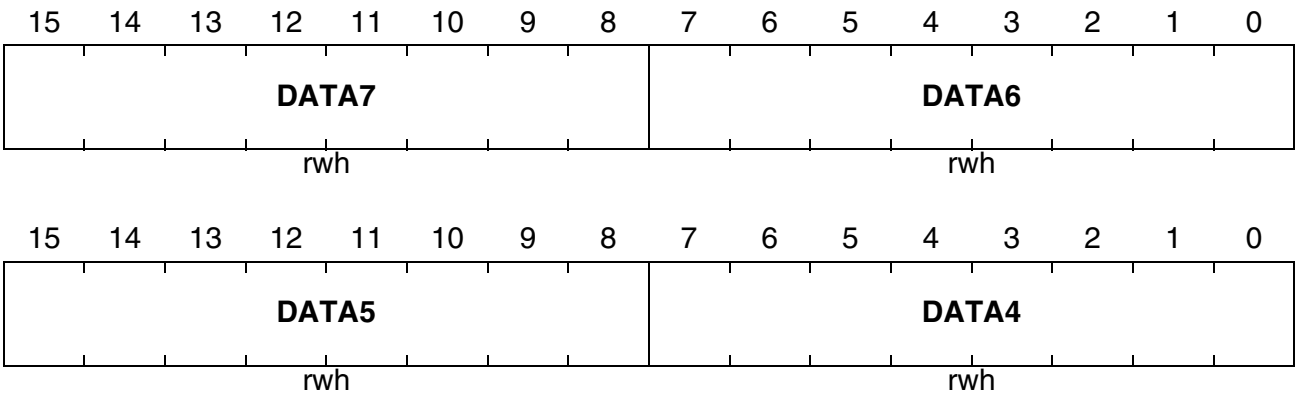
MSGDRHn0 (n = 31-0)
Message Object n Data Register 0 High **Reset Value: 0000_H**
MSGDRLn0 (n = 31-0)
Message Object n Data Register 0 Low **Reset Value: 0000_H**



Field	Bits	Type	Description
DATA0	[7:0] Low	rwh	**Data Byte 0 Associated to Message Object n**
DATA1	[15:8] Low	rwh	**Data Byte 1 Associated to Message Object n**
DATA2	[7:0] High	rwh	**Data Byte 2 Associated to Message Object n**
DATA3	[15:8] High	rwh	**Data Byte 3 Associated to Message Object n**

The Message Data Register 4 contains the data bytes 4 to 7 of message object n.

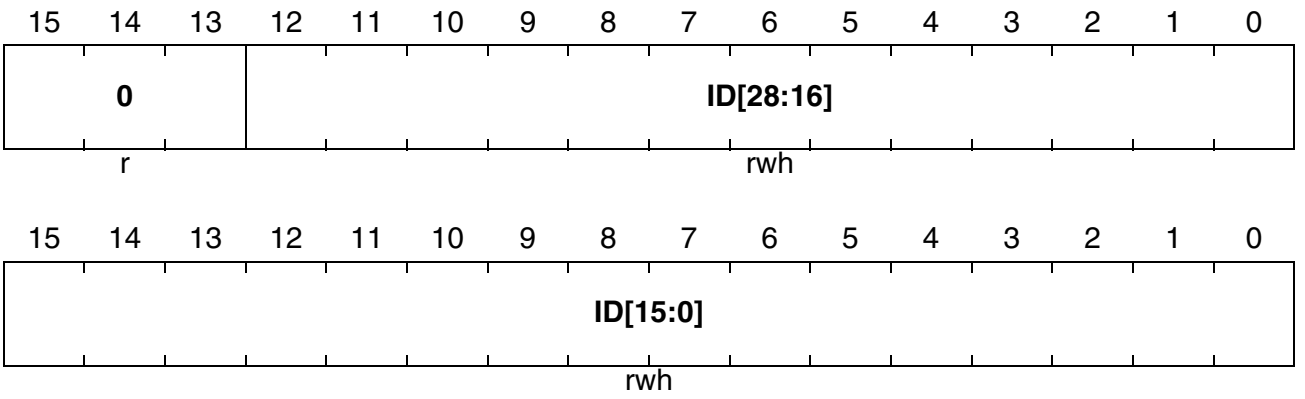
MSGDRHn4 (n = 31-0)
Message Object n Data Register 4 High **Reset Value: 0000_H**
MSGDRLn4 (n = 31-0)
Message Object n Data Register 4 Low **Reset Value: 0000_H**



Field	Bits	Type	Description
DATA4	[7:0] Low	rwh	**Data Byte 4 Associated to Message Object n**
DATA5	[15:8] Low	rwh	**Data Byte 5 Associated to Message Object n**
DATA6	[7:0] High	rwh	**Data Byte 6 Associated to Message Object n**
DATA7	[15:8] High	rwh	**Data Byte 7 Associated to Message Object n**

Register MSGARn contains the identifier of message object n.

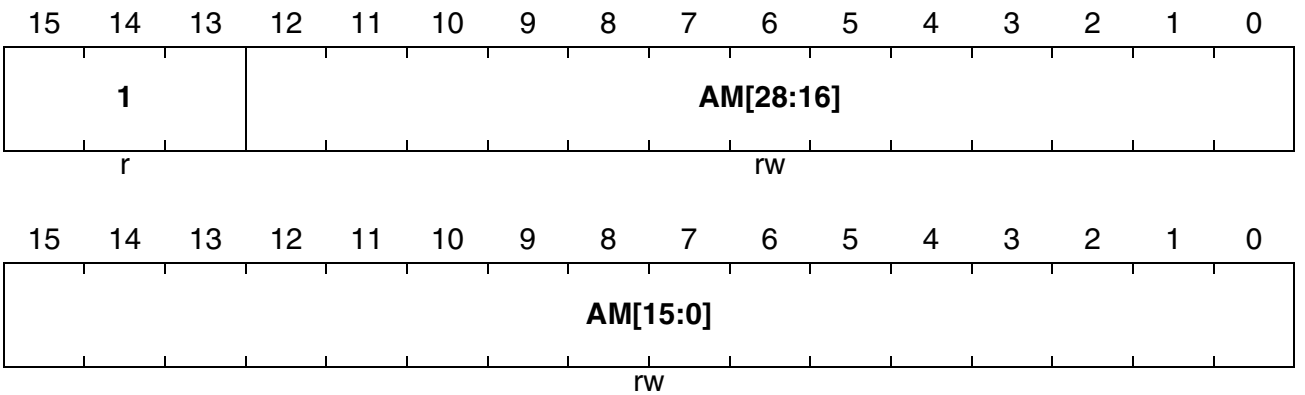
MSGARHn (n = 31-0)
Message Object n Arbitration Register High **Reset Value: 0000_H**
MSGARLn (n = 31-0)
Message Object n Arbitration Register Low **Reset Value: 0000_H**



Field	Bits	Type	Description
ID[15:0] **ID[28:16]**	[15:0] Low [12:0] High	rwh	**Message Identifier** 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”.
0	[15:13] High	r	**Reserved;** returns ‘0’ if read; should be written with ‘0’.

Register MSGAMRn contains the mask bits for the acceptance filtering of message object n.

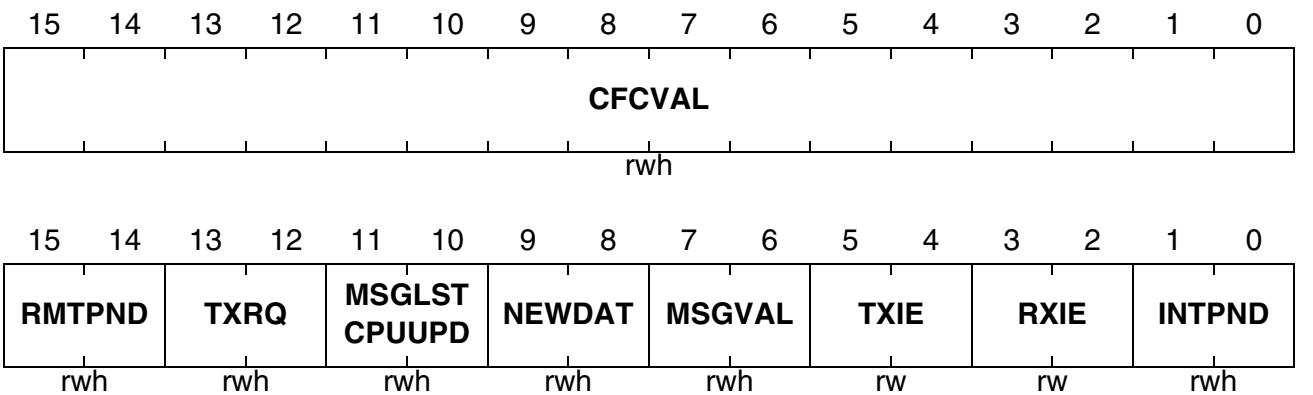
MSGAMRHn (n = 31-0)
Message Object n Arbitration Mask Register High **Reset Value: $FFFF_H$**
MSGAMRLn (n = 31-0)
Message Object n Arbitration Mask Register Low **Reset Value: $FFFF_H$**



Field	Bits	Type	Description
AM[15:0] **AM[28:16]**	[15:0] Low [12:0] High	rw	**Message Acceptance Mask** Mask to filter incoming messages with standard identifiers (AM[28:18]) or extended identifiers (AM[28:0]). For standard identifiers bits AM[17:0] are “don’t care”. 0 Identifier bit is ignored for acceptance test. 1 Identifier bit is taken into account for the acceptance filtering.
1	[15:13] High	r	**Reserved;** returns ‘1’ if read; should be written with ‘1’.

Register MSGCTRn affects the data transfer between a CAN node controller and the corresponding message object n and provides a bitfield to store the captured value of the frame counter.

MSGCTRHn (n = 31-0)
Message Object n Message Control Register High **Reset Value: 0000_H**
MSGCTRLn (n = 31-0)
Message Object n Message Control Register Low **Reset Value: 5555_H**



Field	Bits	Type	Description
INTPND	[1:0] Low	rwh	**Message Object Interrupt Pending** INTPND is generated by an “OR” operation between the RXIPNDn and TXIPNDn flags (if enabled by TXIE or RXIE). INTPND must be reset by software. Resetting INTPND also resets the corresponding RXIPND and TXIPND flags. 01 No message object interrupt request is pending. 10 The message object has generated an interrupt request.
RXIE	[3:2] Low	rw	**Message Object Receive Interrupt Enable** 01 Message object receive interrupt is disabled. 10 Message object receive interrupt is enabled. If RXIE is set, bits INTPND and RXIPND are set after successful reception of a frame.
TXIE	[5:4] Low	rw	**Message Object Transmit Interrupt Enable** 01 Message object transmit interrupt is disabled. 10 Message object transmit interrupt is enabled. If TXIE is set, bits INTPND and TXIPND are set after successful transmission of a frame.

Field	Bits	Type	Description
MSGVAL[1)]	[7:6] Low	rwh	**Message Object Valid** The CAN controller only operates on valid message objects. Message objects can be tagged invalid while they are changed or if they are not used at all. 01 Message object is invalid. 10 Message object is valid.
NEWDAT[2)]	[9:8] Low	rwh	**New Message Object Data Available** 01 No update of message object data occurred. 10 New message object data has been updated.
MSGLST	[11:10] Low	rwh	**Message Lost** (for reception only) 01 No message object data is lost. 10 The CAN controller has stored a new message into the message object while NEWDAT was still set. The previously stored message is lost. MSGLST must be reset by software.
CPUUPD[3)]	[11:10] Low	rwh	**CPU Update** (for transmission only) Indicates that the corresponding message object can not be transmitted now. The software sets this bit in order to inhibit the transmission of a message that is currently updated by the CPU or to control the automatic response to remote requests. 01 The message object data can be transmitted automatically by the CAN controller. 10 The automatic transmission of the message data is inhibited.
TXRQ[4)]	[13:12] Low	rwh	**Message Object Transmit Request Flag** 01 No message object data transmission is requested by the CPU or a remote frame. 10 The transmission of the message object data, requested by the CPU or by a remote frame, is pending. Automatic setting of TXRQ by the CAN node controller can be disabled for Gateway Message Objects via control bit GDFS = ‘0’. TXRQ is automatically reset, when the message object has been successfully transmitted. If there are several valid message objects with pending transmit requests, the message object with the lowest message number will be transmitted first.

Field	Bits	Type	Description
RMTPND	[15:14] Low	rwh	**Remote Pending Flag** (used for transmit-objects) 01 No remote node request for a message object data transmission. 10 Transmission of the message object data has been requested by a remote node but the data has not yet been transmitted. When RMTPND is set, the CAN node controller also sets TXRQ. RMTPND is automatically reset, when the message object data has been successfully transmitted.
CFCVAL	[15:0] High	rwh	**Message Object Frame Counter Value** CFCVAL contains a copy of the frame counter content valid for the last correct data transmission or reception executed for the corresponding message object.

1) MSGVAL has to be set from ‘01’ to ‘10’ in order to take into account an update of bits XTD, DIR, NODE and CANPTR.

2) Bit NEWDAT indicates that new data has been written into the data registers of this corresponding message object. For transmit objects, NEWDAT should be set by software and is reset by the respective CAN node controller when the transmission is started.
For receive objects, NEWDAT is set by the respective CAN node controller after receiving a data frame with matching identifier. It has to be reset by software.
When the CAN controller writes new data into the message object, unused message bytes will be overwritten with non-specified values. Usually, the CPU will clear this bitfield before working on the data and will verify that the bitfield is still cleared once the CPU has finished working to ensure a consistent set of data. For transmit objects, the CPU should set this bitfield along with clearing bitfield CPUUPD. This will ensure that, if the message is actually being transmitted during the time the message is updated by the CPU, the CAN controller will not reset bitfield TXRQ. In this way, TXRQ is only reset once the actual data has been transferred correctly.

3) While bitfield MSGVAL is set (‘10’) an incoming matching remote frame is taken into account by automatically setting bitfields TXRQ and RMTPND to ‘10’ (independent from bitfield CPUUPD/MSGLST). The transmission of a frame is only possible if CPUUPD is reset (‘01’).

4) If a receive object (DIR = ‘0’) is requested for transmission, a remote frame will be sent in order to request a data frame from another node. If a transmit object (DIR = ‘1’) is requested for transmission, a data frame will be sent. Bitfield TXRQ will be reset by the CAN controller along with bitfield RMTPND after the correct transmission of the data frame if bitfield NEWDAT has not been set or after correct transmission of a remote frame.

Note: For transmitting frames (remote frames or data frames), bitfield CPUUPD/MSGLST has to be reset.

The control and status element of the message control registers is implemented with two complementary bits (except the frame counter value). This special mechanism allows the selective setting or resetting of a specific element (leaving others unchanged) without requiring read-modify-write cycles. **Table 21-8** illustrates how to use these 2-bitfields.

Table 21-8 Setting/Resetting the Control and Status Element of the Message Control Registers

Value of the 2-bitfield	Function on Write	Meaning on Read
00_B	reserved	reserved
01_B	Reset element	Element is reset
10_B	Set element	Element is set
11_B	Leave element unchanged	reserved

Register MSGCFGn defines the configuration of message object n and the associated interrupt node pointers. Changes of bits XTD, NODE or DIR by software are only taken into account after setting bitfield MSGVAL to ‘10’. This avoids unintentional modification while the message object is still active by explicitly defining a timing instant for the update. Bits XTD, NODE or DIR can be written while MSGVAL is ‘01’ or ‘10’, the update always takes place by setting MSGVAL to ‘10’.

MSGCFGHn (n = 31-0)
Message Object n Message Configuration Register High **Reset Value: 0000_H**
MSGCFGLn (n = 31-0)
Message Object n Message Configuration Register Low **Reset Value: 0000_H**

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

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0								DLC				DIR	XTD	NO DE	RMM
r								rwh				rwh	rw	rwh	rw

Field	Bits	Type	Description
RMM	0 Low	rw	**Transmit Message Object Remote Monitoring Mode** 0 Remote Monitoring mode is disabled. 1 Remote Monitoring mode is enabled for this transmit message object. The identifier and DLC code of a remote frame with matching identifier are copied to this transmit message object in order to monitor incoming remote frames. Bit RMM is only available for transmit objects and has no impact for receive objects.
NODE	1 Low	rwh	**Message Object CAN Node Select** 0 The message object is assigned to CAN node A. 1 The message object is assigned to CAN node B.
XTD	2 Low	rw	**Message Object Extended Identifier** 0 This message object uses a standard 11-bit identifier. 1 This message object uses an extended 29-bit identifier.
DIR	3 Low	rwh	**Message Object Direction Control** 0 The message object is defined as receive object. If TXRQ = ‘10’, a remote frame with the identifier of this message object is transmitted. On reception of a data frame with matching identifier, the message data is stored in the corresponding MSGDRn0/MSGDRn4 registers. 1 The message object is declared as transmit object. If TXRQ = ‘10’, the respective data frame is transmitted. On reception of a remote frame with matching identifier, RMTPND and TXRQ are set to ‘10’.
DLC[1)]	[7:4] Low	rwh	**Message Object Data Length Code** 0000_B - $1XXX_B$ DLC contains the number of data bytes associated to the message object. Bitfield DLC may be modified by hardware in Remote Monitoring Mode and in Gateway Mode.

Field	Bits	Type	Description
RXINP	[2:0] High	rw	**Receive Interrupt Node Pointer** Bitfield RXINP determines which interrupt node is triggered by a message object receive event, if bitfield RXIE in register MSGCTRn is set. 000_B CAN interrupt node 0 is selected. … … 111_B CAN interrupt node 7 is selected.
TXINP	[6:4] High	rw	**Transmit Interrupt Node Pointer** Bitfield TXINP determines which interrupt node is triggered by a message object transmit event, if bitfield TXIE in register MSGCTRn is set. 000_B CAN interrupt node 0 is selected. … … 111_B CAN interrupt node 7 is selected.
0	[15:8] Low 3, [15:7] High	r	**Reserved;** returns ‘0’ if read; should be written with ‘0’.

1) The maximum number of data bytes is 8. A value > 8 written by the CPU, is internally corrected to 8 but the content of bitfield DLC is not updated.
If a received data frame contains a data length code value > 8, only 8 bytes are taken into account. A read access to bitfield DLC returns the original value of the DLC field of the received data frame.

The FIFO/gateway control register MSGFGCRn contains bits to enable and to control the FIFO functionality, the gateway functionality and the desired transfer actions.

MSGFGCRHn (n = 31-0)
Message Object n FIFO/Gateway Control Register High **Reset Value: 0000_H**
MSGFGCRLn (n = 31-0)
Message Object n FIFO/Gateway Control Register Low **Reset Value: 0000_H**

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

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STT	SDT	FD	0	DLCC	IDC	SRREN	GDFS	0			FSIZE				
rw	rw	rw	r	rw	rw	rw	rw	r			rw				

Field	Bits	Type	Description
FSIZE	[4:0] Low	rw	**FIFO Size Control** Bitfield FSIZE determines the number of message objects combined to a FIFO buffer. Even numbered message objects may provide FIFO base or slave functionality, while odd numbered message objects are restricted to slave functionality. In gateway mode, FSIZE determines the length of the FIFO on the destination side. 00000_B message object n is part of a 1-stage FIFO 00001_B message object n is part of a 2-stage FIFO 00011_B message object n is part of a 4-stage FIFO 00111_B message object n is part of a 8-stage FIFO 01111_B message object n is part of a 16-stage FIFO 11111_B message object n is part of a 32-stage FIFO else reserved FSIZE = ‘00000’ leads to the behavior of a standard message object (the pointer CANPTR used for this action will not be changed). This value has to be written if a gateway transfer to a single message object (no FIFO) as destination is desired. FSIZE is not evaluated for message objects configured in standard mode, shared gateway mode or FIFO slave functionality. In this case, FSIZE should be programmed to ‘00000’.

Field	Bits	Type	Description
GDFS	8 Low	rw	**Gateway Data Frame Send** Specifies if a CAN data frame will be automatically generated on the destination side after new data has been transferred via gateway from the source to the destination side. 0 No additional action, TXRQ will not be set on the destination side. 1 The corresponding data frame will be sent automatically (TXRQ of the message object, pointed to by CANPTRn, will be set by hardware). Bit GDFS is only taken into account, if a data frame has been received ($DIR_{<s>}$ = '0').
SRREN	9 Low	rw	**Source Remote Request Enable** Specifies if the transmit request bit is set in message object n itself (to generate a data frame) or in the message object pointed to by CANPTRn (in order to generate a remote frame on the source bus). 0 A remote on the source bus will not be generated, a data frame with the contents of the destination object will be generated on the destination bus, instead (TXRQn will be set). 1 A data frame with the contents of the destination object will not be sent. Instead, a corresponding remote frame will be generated by the message object pointed to by bitfield CANPTRn (TXRQ[CANPTRn] will be set). SRREN is restricted to transmit message objects in normal or shared gateway mode (DIR = '1'). This bit is only taken into account if a remote frame has been received. Bit SRREN must not be set if message object n is part of a FIFO buffer. In order to generate a remote frame on the source side, CANPTR has to point to the source message object.

Field	Bits	Type	Description
IDC	10 Low	rw	**Identifier Copy** IDC controls the identifier handling during a frame transfer through a gateway. 0 The identifier of the receiving object is not copied to the transmitting message object. 1 The identifier of the receiving object is automatically copied to the transmitting message object. Bitfield IDC is restricted to message objects configured in normal gateway mode.
DLCC	11 Low	rw	**Data Length Code Copy** DLCC controls the handling of the data length code during a data frame transfer through a gateway. 0 The data length code, provided by the source object, is not copied to the transmitting object. 1 The data length code, valid for the receiving object, is copied automatically to the transmitting object. Bitfield DLCC is restricted to message objects configured in normal gateway mode.

Field	Bits	Type	Description
FD	13 Low	rw	**FIFO Direction** FD is only taken into account for a FIFO base object (the FD bits of all FIFO elements should have an identical value). It defines which transfer action (reception or transmission) leads to an update of the FIFO base object's CANPTR. 0 **FIFO Reception:** The CANPTR (of the FIFO base object) is updated after a correct reception of a data frame (DIR = '0') or a remote frame (DIR = '1') by the currently addressed message object. The CANPTR is left unchanged after any transmission. 1 **FIFO Transmission:** The CANPTR (of the FIFO base object) is updated after a correct transmission of a data frame (DIR = '1') or a remote frame (DIR = '0') from the currently addressed message object. The CANPTR is left unchanged after any reception. Bitfield FD is not correlated with bit DIR.
SDT	14 Low	rw	**Single Data Transfer Mode** This bit is taken into account in any transfer mode (FIFO mode or as standard object, receive and transmit objects). 0 Control bit MSGVAL is not reset when this object has taken part in a successful data transfer (receive or transmit). 1 Control bit MSGVAL is automatically reset after a successful data transfer (receive or transmit) has taken place. Bit SDT is not taken into account for remote frames. Bit SDT has to be reset in all message objects belonging to a FIFO buffer.
STT	15 Low	rw	**Single Transmission Try** 0 Single transmission try is disabled. 1 Single transmission try is enabled. The corresponding TXRQ bit is reset immediately after the transmission has started[1)].

Field	Bits	Type	Description
CANPTR	[4:0] High	rwh	**CAN Pointer for FIFO/Gateway Functions** **Message object is configured in standard mode (MMC = ‘000’):** No impact, CANPTR should be initialized with the respective message object number. **Message object is configured as FIFO base object (MMC = ‘010’):** CANPTR contains the number of the message object addressed by the associated CAN controller for the next transmit or receive operation. For initialization, CANPTR should be written with the message number of the respective FIFO base object. **Message object is configured as FIFO slave object (MMC = ‘011’):** CANPTR has to be initialized with the respective message object number of the FIFO base object. **Message object is configured for normal gateway mode (MMC = ‘100’):** CANPTR contains the number of the message object used as gateway destination object. **Message object is configured as gateway destination object without FIFO functionality (MMC = ‘000’):** If SRREN is set to ‘1’, CANPTR has to be initialized with the number of the message object used as gateway source. The backward pointer is required to transfer remote frames from the destination to the source side. If SRREN is cleared, CANPTR is not evaluated and must be initialized with the respective message object number. **Message object is configured for shared gateway mode (MMC = ‘101’):** No impact, CANPTR has to be initialized with the respective message object number. For FIFO functionality (or gateway functionality with a FIFO as destination), CANPTRn should not be written by software while FIFO mode is activated and data transfer is in progress. This bitfield can be used to reset the FIFO by software.

Field	Bits	Type	Description
MMC	[10:8] High	rw	**Message Object Mode Control** Bitfield MMC controls the functionality of message object n. 000_B Standard message object functionality 010_B FIFO functionality enabled (base object) 011_B FIFO functionality enabled (slave object) 100_B Normal gateway functionality for incoming frames 101_B Shared gateway functionality for incoming frames others reserved
0	[7:5], 12 Low [7:5], [15:11] High	–	**Reserved;** returns ‘0’ if read; should be written with ‘0’.

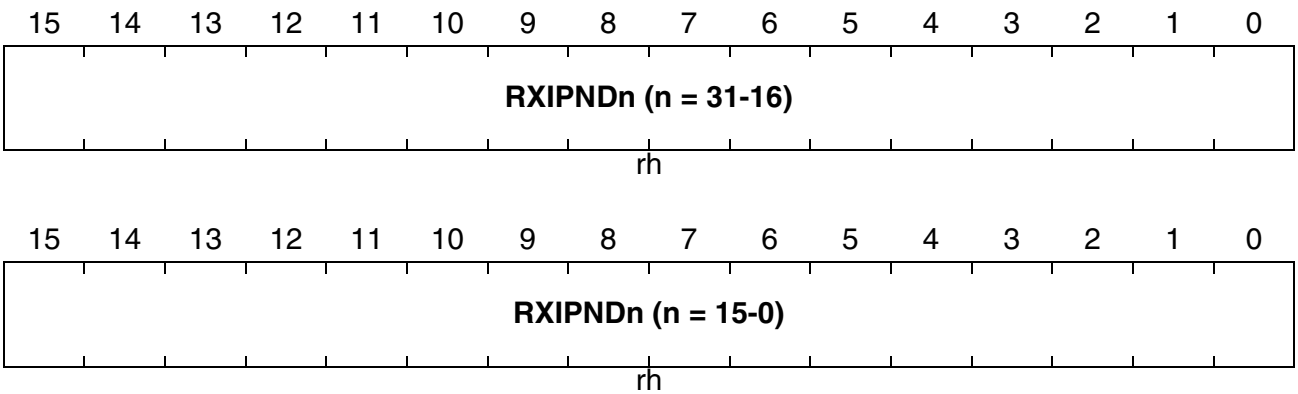
1) As a result, a message will not be re-transmitted if it has lost arbitration or has been corrupted by an error frame.

Note: Changes of bitfield CANPTR for transmission objects are only taken into account after setting bitfield MSGVAL to ‘10’. This avoids unintentional modification while the message object is still active by explicitly defining a timing instant for the update. Bitfield CANPTR for transmission objects can be written while MSGVAL is ‘01’ or ‘10’, the update always takes place by setting MSGVAL to ‘10’. Changes of bitfield CANPTR for receive objects are immediately taken into account.

21.2.4 Global CAN Control/Status Registers

The Receive Interrupt Pending Register indicates the pending receive interrupts for message object n.

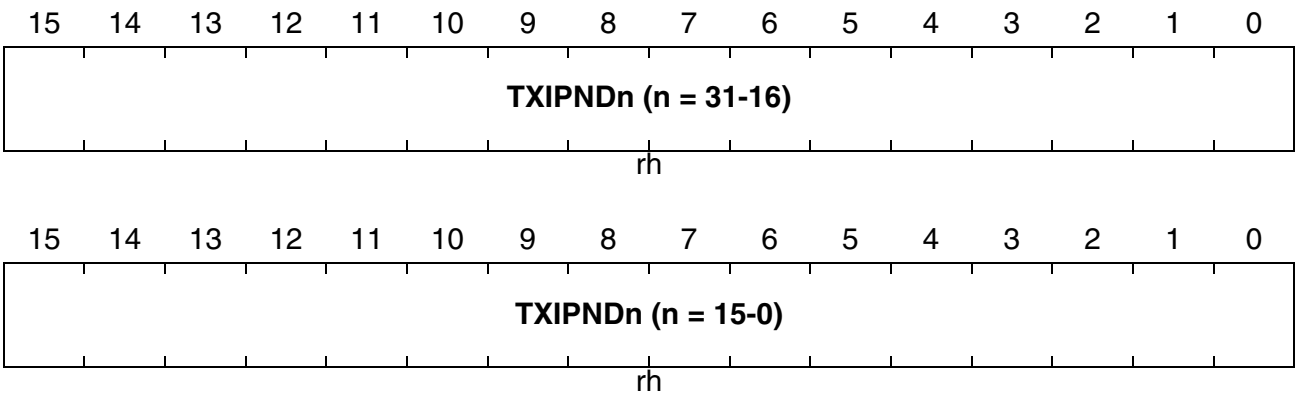
RXIPNDH
Receive Interrupt Pending Register High **Reset Value: 0000_H**
RXIPNDL
Receive Interrupt Pending Register Low **Reset Value: 0000_H**



Field	Bits	Type	Description
RXIPNDn (n = 15-0) **RXIPND (n = 31-16)**	n Low n-16 High	rh	**Message Object n Receive Interrupt Pending** Bit RXIPNDn is set by hardware if message object n received a frame and bit RXIEn has been set. 0 No receive is pending for message object n. 1 Receive is pending for message object n. RXIPNDn can be cleared by software via resetting the corresponding bit INTPNDn.

The Transmit Interrupt Pending Register indicates whether a transmit interrupt is pending for message object n.

TXIPNDH
Transmit Interrupt Pending Register High **Reset Value: 0000_H**
TXIPNDL
Transmit Interrupt Pending Register Low **Reset Value: 0000_H**



Field	Bits	Type	Description
TXIPNDn (n = 15-0) **TXIPND (n = 31-16)**	n Low n-16 High	rh	**Message Object n Transmit Interrupt Pending** Bit TXIPNDn is set by hardware if message object n transmitted a frame and bit TXIEn has been set. 0 No transmit is pending for message object n. 1 Transmit is pending for message object n. TXIPNDn can be cleared by software via resetting the corresponding bit INTPNDn.

21.3 XC161 Module Implementation Details

This section describes:

- the TwinCAN module related interfaces such as port connections and interrupt control
- all TwinCAN module related registers with its addresses and reset values

21.3.1 Interfaces of the TwinCAN Module

In XC161 the TwinCAN module is connected to IO ports according to **Figure 21-28**.

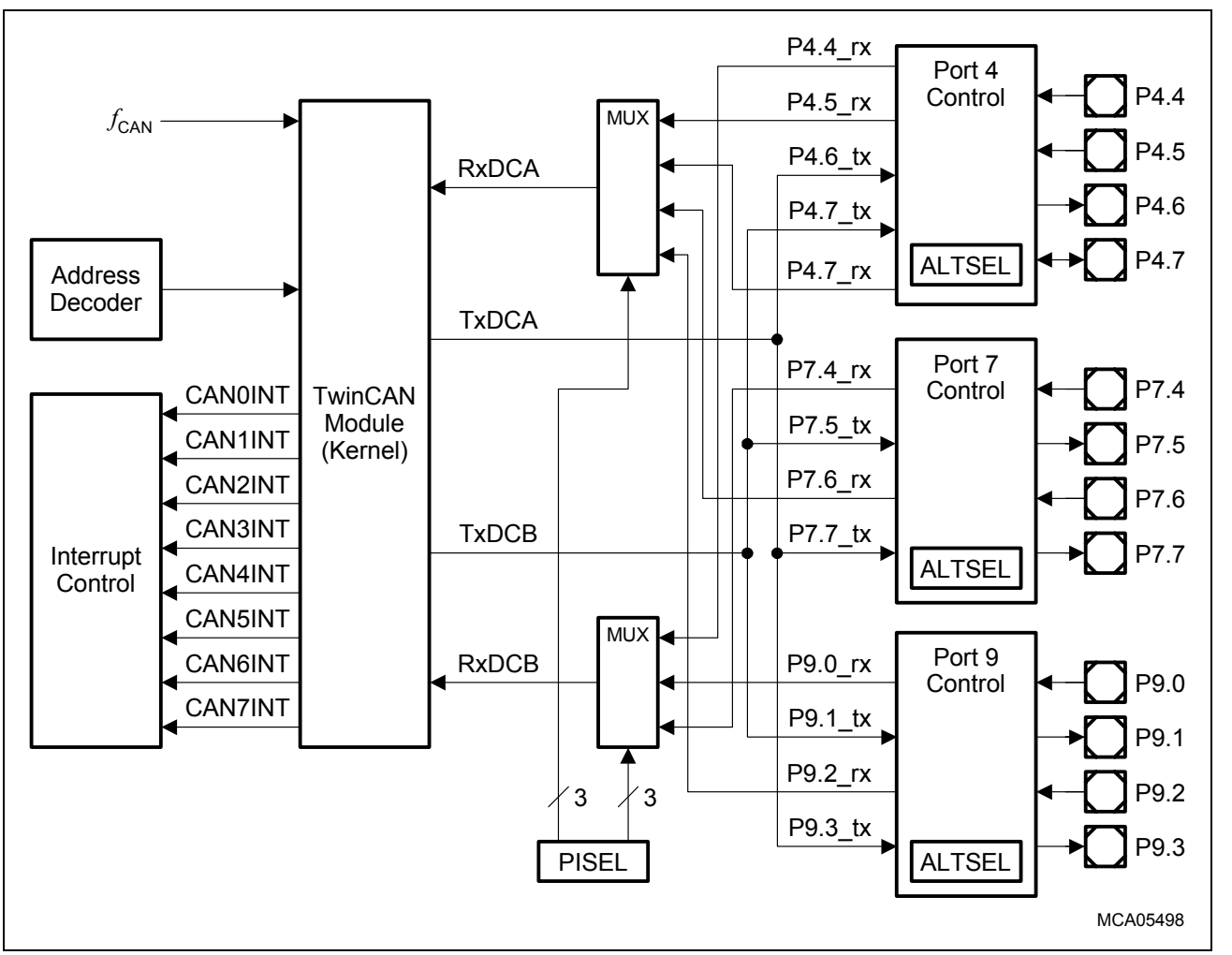


Figure 21-28 TwinCAN Module IO Interface

The input receive pins can be selected by bitfield RISA (for node A) and bitfield RISB (for node B) in the PISEL register. The output transmit pins are defined by the corresponding ALTSEL registers of Port 4, Port 7, or Port 9.

The TwinCAN has eight interrupt request lines.

Note: The interrupt node of interrupt request 7 of the TwinCAN can be shared with the SDLM module.

21.3.2 TwinCAN Module Related External Registers

Figure 21-29 shows the module related external registers which are required for programming the TwinCAN module.

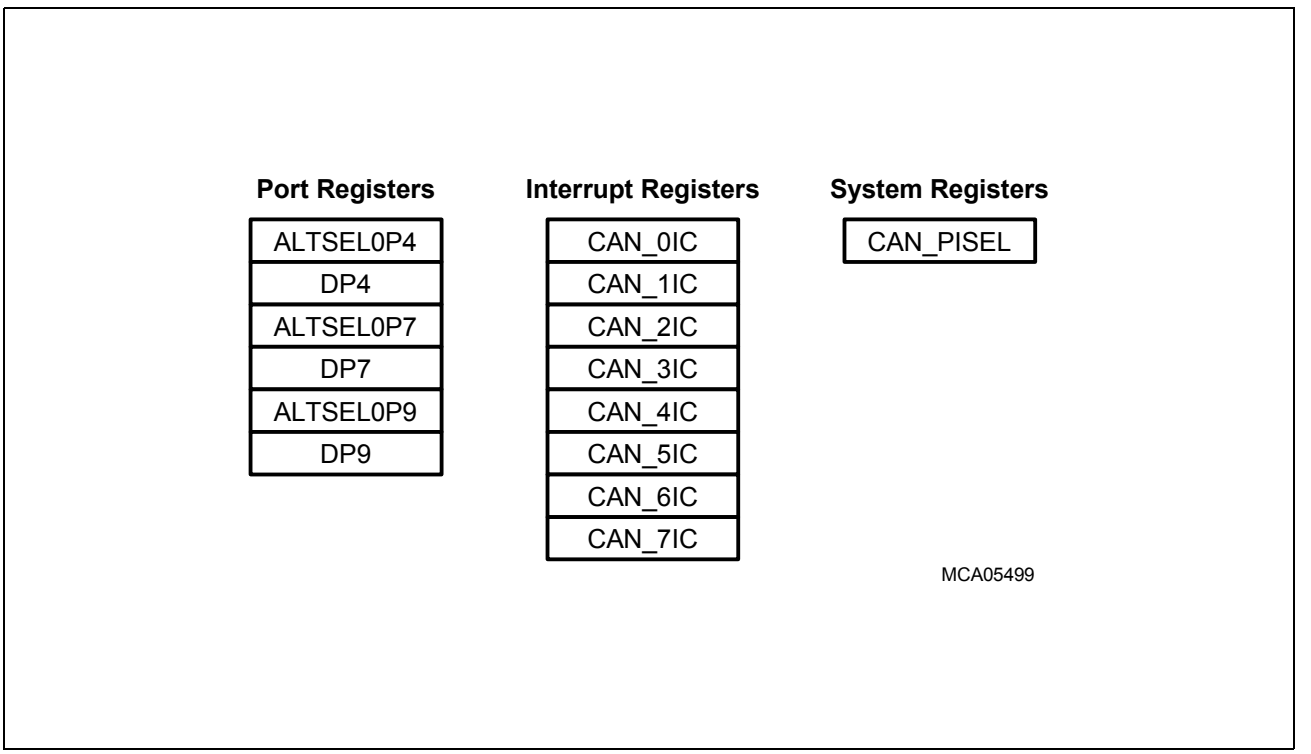
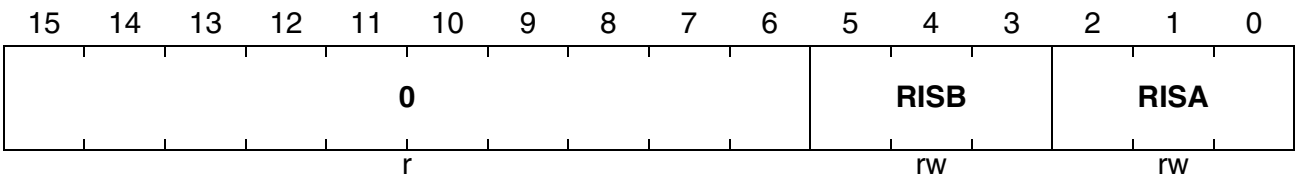


Figure 21-29 TwinCAN Implementation Specific Registers

21.3.2.1 System Registers

Register CAN_PISEL allows the user to select the input pins for the two TwinCAN receive signals RXDCA and RXDCB.

CAN_PISEL
TwinCAN Port Input Select Register **Reset Value: 0000_H**

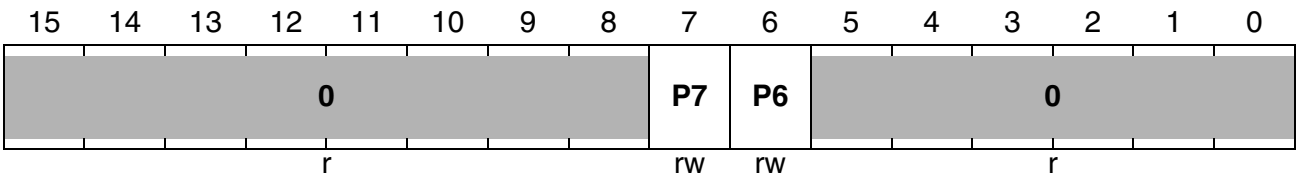


Field	Bits	Type	Description
RISA	[2:0]	rw	**Receive Input Selection for Node A** Bitfield RISA defines the input pin for the TwinCAN receive line RXDCA for node A. 000 The input pin for RXDCA is P4.5 001 The input pin for RXDCA is P4.7 010 The input pin for RXDCA is P7.6 011 The input pin for RXDCA is P9.2 1XX Reserved.
RISB	[5:3]	rw	**Receive Input Selection for Node B** Bitfield RISB defines the input pin for the TwinCAN receive line RXDCB for node B. 000 The input pin for RXDCB is P4.4 001 The input pin for RXDCB is P9.0 010 The input pin for RXDCB is P7.4 011 Reserved. 1XX Reserved.
0	[15:6]	r	**Reserved;** returns ‘0’ if read; should be written with ‘0’.

21.3.2.2 Port Registers

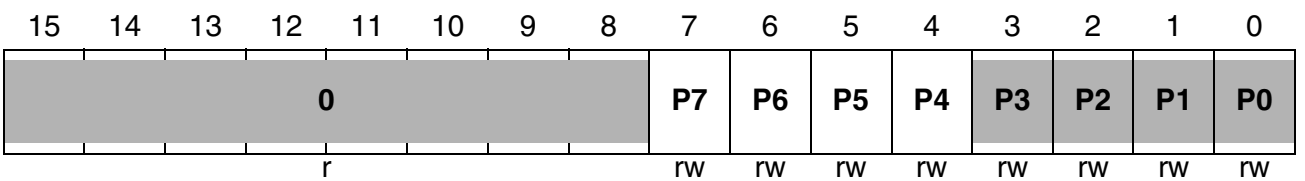
The port registers required to program to TwinCAN operation are listed as follows.

ALTSEL0P4
P4 Alternate Select Register 0 **Reset Value: 0000_H**



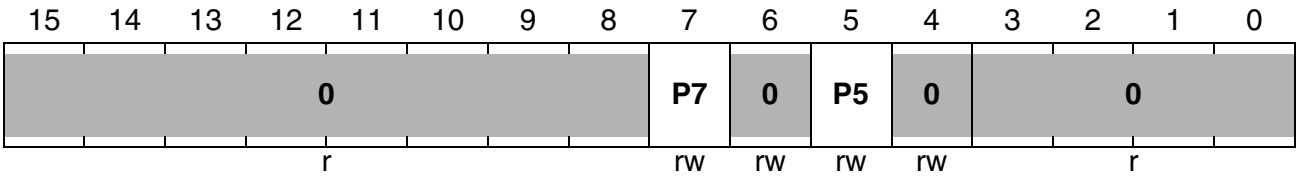
Field	Bit	Type	Description
ALTSEL0 P4.y	6, 7	rw	**P4 Alternate Select Register 0 bit y** 0 associated peripheral output is not selected as alternate function 1 associated peripheral output is selected as alternate function

DP4
P4 Direction Ctrl. Register **Reset Value: 0000_H**



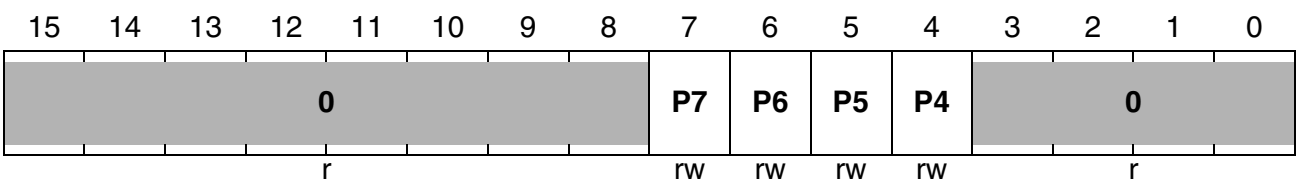
Field	Bit	Type	Description
DP4.y	7 … 4	rw	**Port Direction Register DP4 Bit y** 0 Port line P4.y is an input (high-impedance) 1 Port line P4.y is an output

ALTSEL0P7
P7 Alternate Select Register 0 **Reset Value: 0000_H**



Field	Bit	Type	Description
ALTSEL0 P7.y	7, 5	rw	**P7 Alternate Select Register 0 Bit y** 0 associated peripheral output is not selected as alternate function 1 associated peripheral output is selected as alternate function

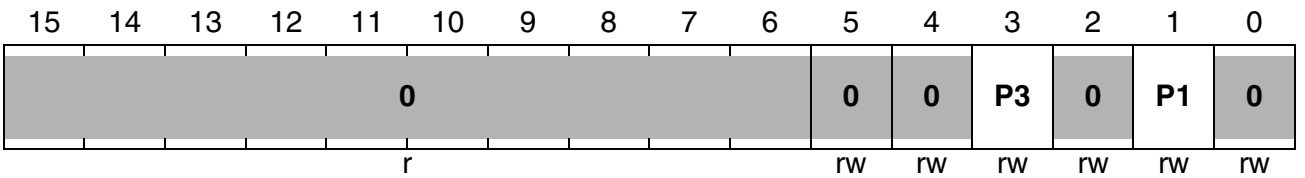
DP7
P7 Direction Ctrl. Register **Reset Value: 0000_H**



Field	Bit	Type	Description
DP7.y	7 … 4	rw	**Port Direction Register DP7 Bit y** 0 Port line P7.y is an input (high-impedance) 1 Port line P7.y is an output

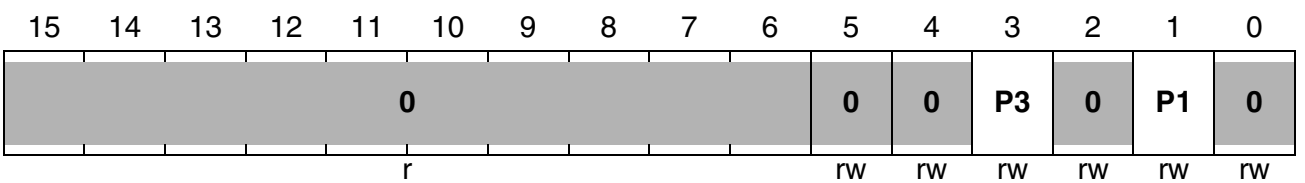
Note: Shaded bits are not related to TwinCAN operation.

ALTSEL0P9
P9 Alternate Select Register 0 **Reset Value: 0000_H**



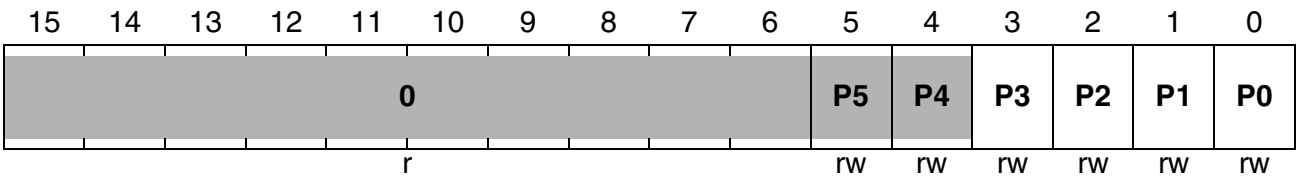
Field	Bit	Type	Description
ALTSEL0 P9.y	3, 1	rw	**P9 Alternate Select Register 0 Bit y** 0 associated peripheral output is not selected as alternate function 1 associated peripheral output is selected as alternate function

ALTSEL1P9
P9 Alternate Select Register 1 **Reset Value: 0000_H**



Field	Bit	Type	Description
ALTSEL1 P9.y	3, 1	rw	**P9 Alternate Select Register 1 Bit y** 0 associated peripheral output is not selected as alternate function 1 associated peripheral output is selected as alternate function

DP9
P9 Direction Ctrl. Register **Reset Value: 0000_H**



Field	Bit	Type	Description
DP9.y	3 … 0	rw	**Port Direction Register DP9 Bit y** 0 Port line P9.y is an input (high-impedance) 1 Port line P9.y is an output

Note: Shaded bits are not related to TwinCAN operation.

Table 21-9 shows the required register setting to configure the IO lines of the TwinCAN module for operation.

Table 21-9 TwinCAN IO Selection and Setup

Port Lines	Alternate Select Register	Port Input Select Register	Direction Control Register	IO
TwinCAN Node A				
P4.5 / RxDCA	–	CAN_PISEL[2:0] = 000	DP4.P5 = 0	Input
P4.6 / TxDCA	ALTSEL0P4.P6 = 1	–	DP4.P6 = 1	Output
P4.7 / RxDCA	–	CAN_PISEL[2:0] = 001	DP4.P7 = 0	Input
P7.6 / RxDCA	–	CAN_PISEL[2:0] = 010	DP7.P6 = 0	Input
P7.7 / TxDCA	ALTSEL0P7.P7 = 1	–	DP7.P7 = 1	Output
P9.2 / RxDCA	–	CAN_PISEL[2:0] = 011	DP9.P2 = 0	Input
P9.3 / TxDCA	ALTSEL0P9.P3 = 1 and ALTSEL1P9.P3 =1	–	DP9.P3 = 1	Output
TwinCAN Node B				
P4.4 / RxDCB	–	CAN_PISEL[5:3] = 000	DP4.P4 = 0	Input
P4.7 / TxDCB	ALTSEL0P4.P7 = 1	–	DP4.P7 = 1	Output
P7.4 / RxDCB	–	CAN_PISEL[5:3] = 010	DP7.P4 = 0	Input
P7.5 / TxDCB	ALTSEL0P7.P5 = 1	–	DP7.P5 = 1	Output
P9.0 / RxDCB	–	CAN_PISEL[5:3] = 001	DP9.P0 = 0	Input
P9.1 / TxDCB	ALTSEL0P9.P1 = 1 and ALTSEL1P9.P1 =1	–	DP9.P1 = 1	Output

Note: The ALTSEL1 registers of Port 7 and Port 4 are ‘don’t care’ for selecting the TwinCAN alternate output function.

21.3.2.3 Interrupt Registers

The interrupts of the TwinCAN module are controlled by the following interrupt control registers:

- CAN_0IC
- CAN_1IC
- CAN_2I
- CAN_3I
- CAN_4I
- CAN_5IC
- CAN_6IC
- CAN_7IC

All interrupt control registers have the same structure. Refer to the System Units for its description and also details on interrupt handling and processing.

21.3.3 Register Table

Table 21-10 shows the system registers related to the TwinCAN module. It summarizes the addresses and reset values. In order to simplify the kernel description, the prefix ‘CAN_’ is added only in this register list.

The start address for the TwinCAN module is **20’0000$_H$**, the register offsets (relative to this address) are given in the TwinCAN kernel description. See Figure 21-27. A full register listing of all CAN registers is provided in register table section and in the system book.

Table 21-10 TwinCAN Module Register Summary

Name	Description	Address[1]	Reset Value
		16-Bit	
TwinCAN Module System Registers			
CAN_PISEL	TwinCAN Port Input Select Register	20’0004$_H$	0000$_H$
CAN_0IC	TwinCAN Interrupt Control Register for the CAN interrupt node 0.	F196$_H$	0000$_H$
CAN_1IC	TwinCAN Interrupt Control Register for the CAN interrupt node 1.	F142$_H$	0000$_H$
CAN_2IC	TwinCAN Interrupt Control Register for the CAN interrupt node 2.	F144$_H$	0000$_H$
CAN_3IC	TwinCAN Interrupt Control Register for the CAN interrupt node 3.	F146$_H$	0000$_H$
CAN_4IC	TwinCAN Interrupt Control Register for the CAN interrupt node 4.	F148$_H$	0000$_H$
CAN_5IC	TwinCAN Interrupt Control Register for the CAN interrupt node 5.	F14A$_H$	0000$_H$
CAN_6IC	TwinCAN Interrupt Control Register for the CAN interrupt node 6.	F14C$_H$	0000$_H$
CAN_7IC[2]	TwinCAN Interrupt Control Register for the CAN interrupt node 7.	F14E$_H$	0000$_H$

1) The 8-bit short addresses are not available for the TwinCAN module kernel registers.

2) In the XC161 device, the CAN interrupt node 7 is shared with the SDLM interrupt 1. In order to avoid mismatches if the CAN interrupt 7 is used by the TwinCAN module, the SDLM interrupt 1 should be mapped to a common SDLM interrupt 0 (see register SDLM_PISEL).

22 Serial Data Link Module SDLM

22.1 Overview

The Serial Data Link Module (SDLM) provides serial communication to a J1850 based multiplexed bus via an external J1850 bus transceiver chip. The module is conform to the SAE Class B J1850 specification and compatible to class 2 protocol.

General SDLM Features

- Compliant to SAE Class B J1850 specification
- GM class 2 protocol fully supported
- Variable Pulse Width (VPW) format with 10.4 kBaud
- High speed receive/transmit 4x mode with 41.6 kBaud
- Digital noise filter
- Power save mode and automatic walk-up upon bus activity
- Single-byte headers or consolidated headers supported
- CRC generation & check supported
- Receive and transmit block mode supported
- Transmission of two passive bits after arbitration loss on a byte boundary can be enabled

Data Link Operation Features

- 11 bytes transmit buffer
- Double-buffered 11 bytes receive buffer
- Support of In-frame response (IFR) types 1, 2, 3
- Automatic IFR Transmission for IFR types 1, 2 for three byte consolidated headers
- Advanced interrupt handling for RX, TX and error conditions
- All interrupt sources can be separately enabled/disabled
- 8-byte transmit FIFO and 16-byte receive FIFO in block mode

22.2 SDLM Kernel Description

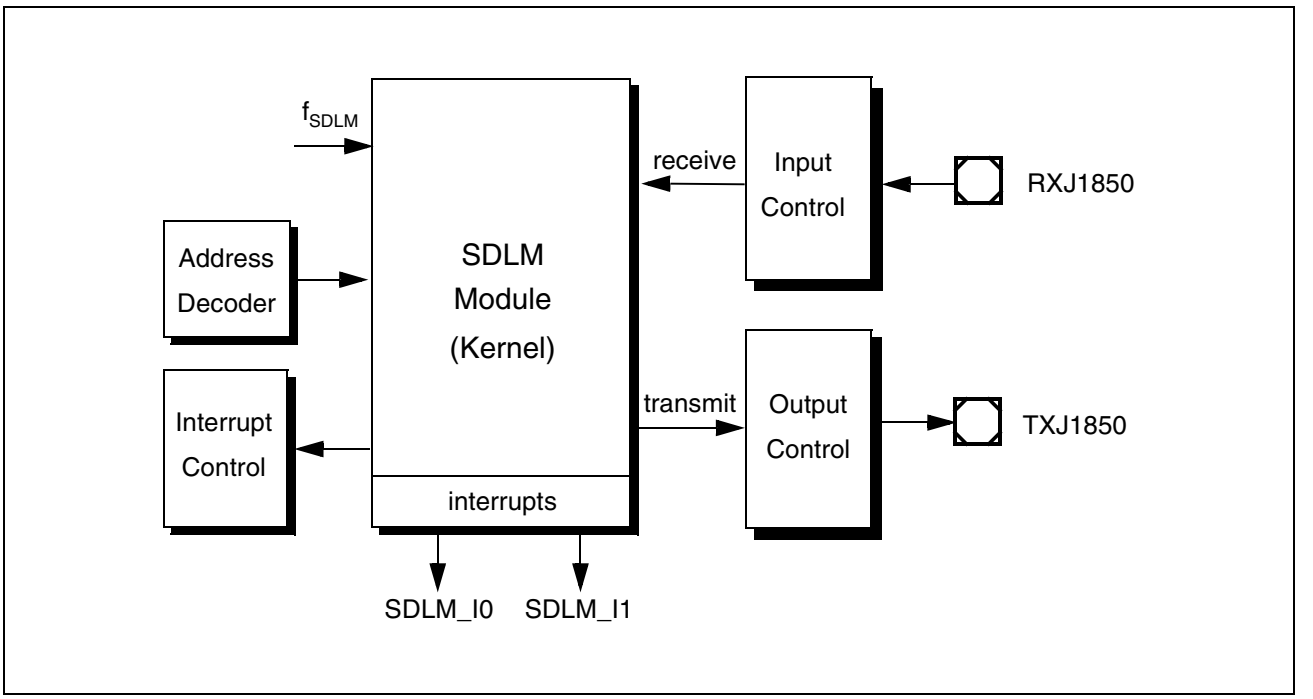


Figure 22-1 General Block Diagram of the SDLM Interface

The SDLM module communicates with the external world (J1850 bus) via two I/O lines, the receive line RXJ1850 (data input signal) and the transmit line TXJ1850 (data output signal).

The module provides the feature to select one out of four possible input pins and one out of four possible output pins. The desired input pin is defined by bitfield IS (input selection), the output pin is selected by the ALTSEL bitfield of the port.

22.2.1 J1850 Concept

The SAE Class-B specification establishes the requirements for a serial bus protocol used in automotive and industrial applications. Basically it describes the network's characteristics in three layers: the physical layer, the data link layer and the application layer.

The physical layer handles the frame transfer including bit/symbol encoding and timing. The data link layer defines the J1850 protocol in terms of frame elements, error detection, bus access, frame arbitration, and clock synchronization. Finally, the application layer needs to evaluate message screening/filtering by software and the handling of diagnostic parameters/codes.

The J1850 is a multi-master based serial protocol. Each node has a local clock, which allows for simultaneous access to the bus.

22.2.1.1 Frame Format Basics

This chapter summarizes the basic definitions of the SAE Standard Class B Data Communication Network Interface protocol.

The general J1850 frame format is defined as:

idle, SOF, DATA_0, …, Data_N, CRC, EOD, NB, IFR_1, …, IFR_N, EOF, IFS, idle

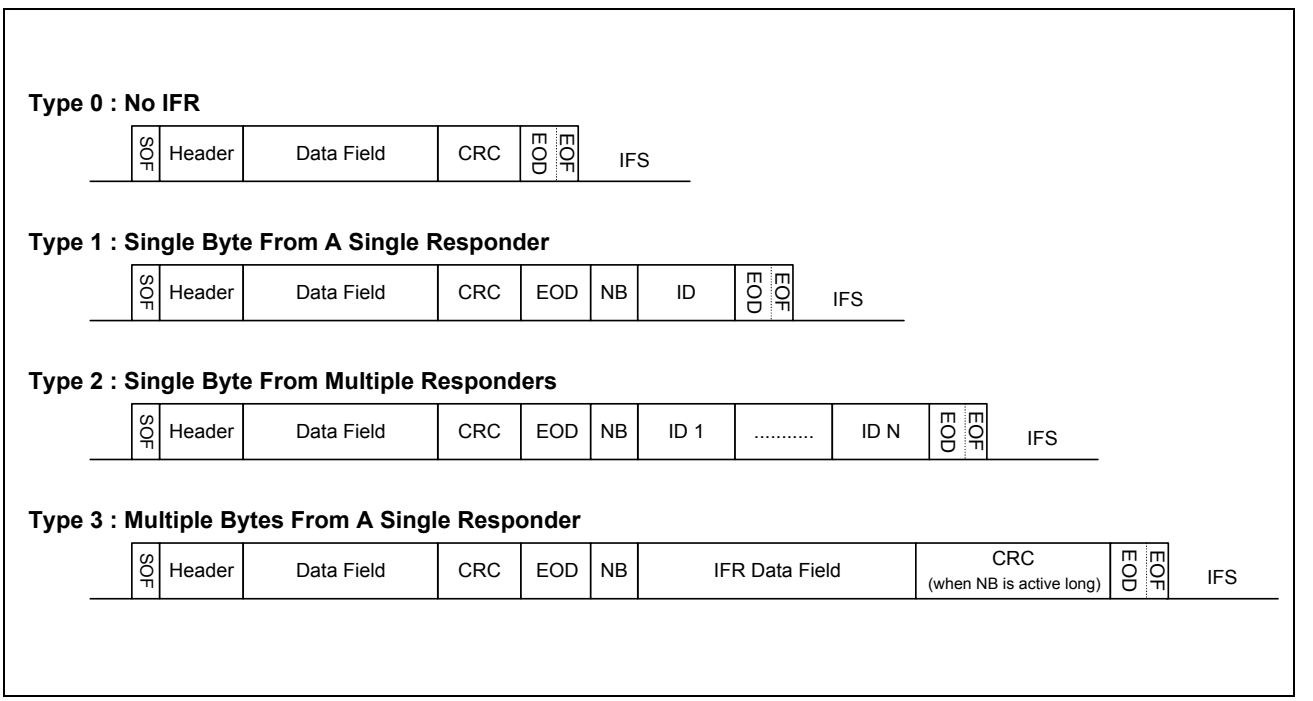


Figure 22-2 Standard Frame Types

Table 22-1 summarizes the used abbreviations.

Table 22-1 Abbreviations Used

Symbol	Name	Description
SOF	Start of Frame	The SOF mark is used to uniquely identify the start of a frame. SOF is not used for CRC error calculation.
DATA_0 - DATA_N	Data bytes	Data bytes (8 bits); starting with MSB first; maximum frame length including header byte(s) and IFR byte(s) but excluding frame delimiters (SOF, EOD, EOF, and IFS) and CRC byte is 11 bytes.
CRC	CRC byte	Cyclic redundancy check byte; generated at the transmission and checked during reception.
EOD	End of Data	Indicates the end of a transmission by the originator of a frame; directly after EOD an IFR can be started by the recipient(s) of the frame.
NB	Normalization Bit	Required for 10.4 Kbps mode only; follows after an EOD and before an IFS symbol; defines the start of an in-frame response.
IFR_1 - IFR_N	In-Frame Response byte(s)	In Frame Response byte(s) (ID) can be sent by receiving devices after the sending device has sent an EOD.
EOF	End of Frame	This symbol defines the end of a frame.
IFS	Inter-Frame Separation	This symbol separates two consecutive frames.
idle	idle	The bus is idle if no transmission takes place.

22.2.1.2 J1850 Bits and Symbols

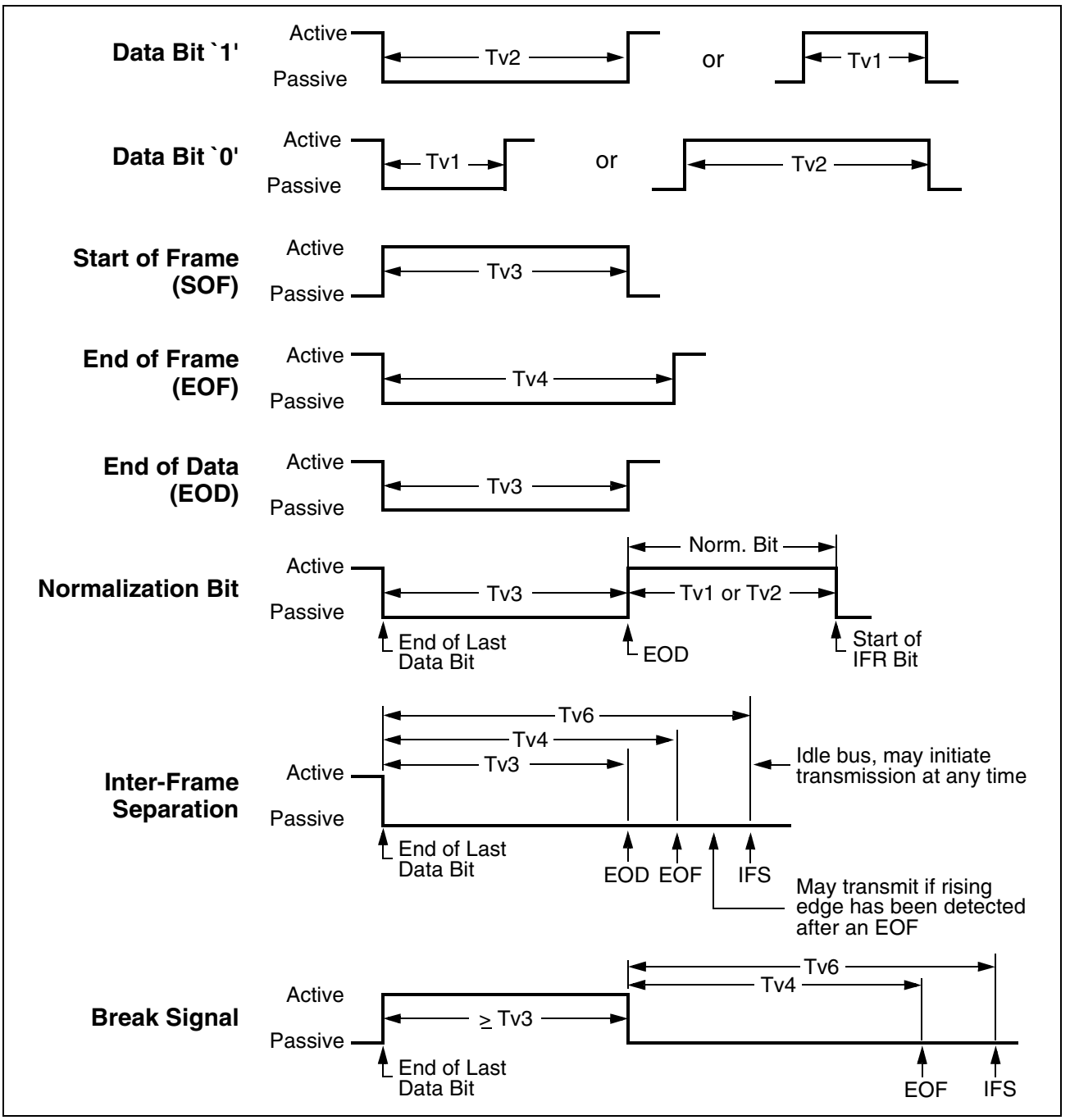


Figure 22-3 J1850 Variable Pulse Width (VPW) Format

Table 22-2 Timing Examples for VPW Format

Frequency	Tv1	Tv2	Tv3	Tv4	Tv5	Tv6
10.4 kbit/s	64 µs	128 µs	200 µs	280 µs	–	–

22.2.1.3 Frame Arbitration

The frame arbitration in the J1850 compatible networks follows the concept of Carrier Sense Multiple Access (CSMA) with non-destructive message arbitration. When two nodes have access to the bus at the same time, the priority decision is made during transmission. The node which has won the arbitration will continue transmission and the other node will stop transmitting. The SDLM always receives the current message on the bus in its receive buffer structure, even while transmitting.

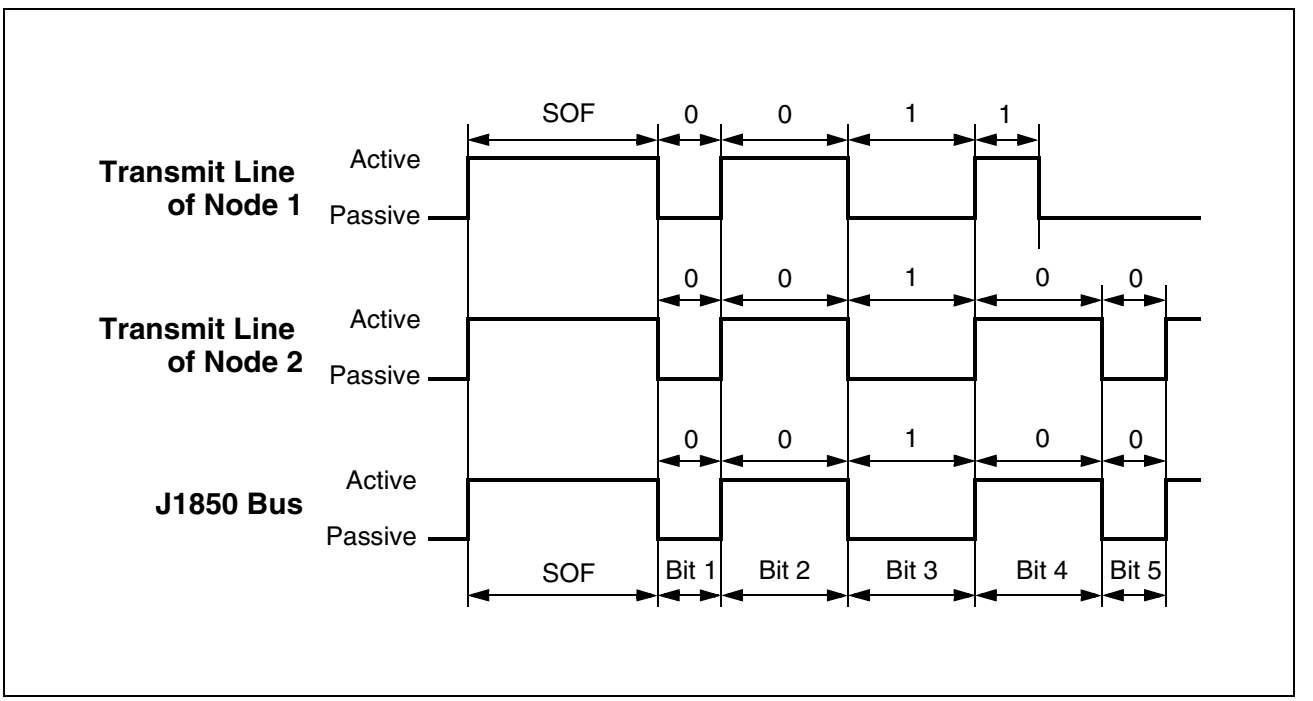


Figure 22-4 J1850 VPW Message Arbitration

22.2.2 Block Diagram

The SDLM module is built up by two basic blocks, the Protocol Controller and the Data Link Controller.

The Protocol Controller basically contains the Bit Stream Processor and the two Shift Registers for the transmit and the receive path. The Bit Stream Processor encodes/decodes the Variable Pulse Width (VPW) data stream and translates incoming VPW symbols into data logic levels. The Protocol Controller further has 8-bit wide data interfaces to the Data Link Controller.

The Data Link Controller can handles incoming and outgoing data using three 8-bit wide data buffers, the 11-byte Transmit Buffer and two 11-byte Receive Buffers. Further, several control tasks (interrupt, timing, and buffer control) are managed by the Data Link Controller.

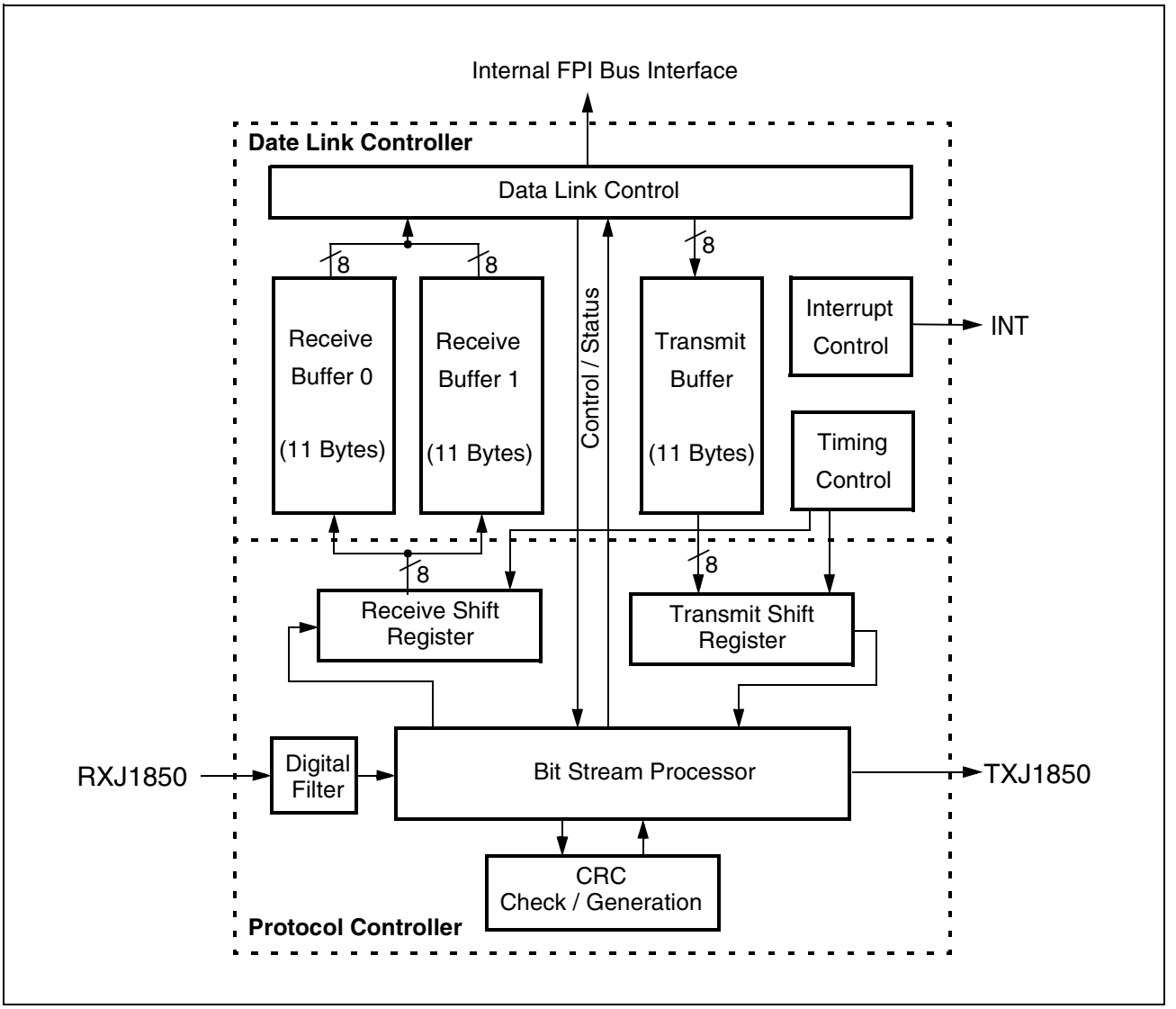


Figure 22-5 SDLM Kernel Block Diagram

The general configuration of the data link controller is done via the Global Control Register, the Clock Divider Register and the Transceiver Delay Register. The bits within these registers provide the following functions:

- SDLM enable/disable
- 4x Mode enable/disable
- Block Mode enable/disable
- Header type configuration (single or consolidated)
- Normalization bit polarity selection
- Receive buffer overwrite control
- Clock divider for J1850 bus rate to adapt to the peripheral clock frequency
- Compensation of transceiver delay by SDLM
- Transmission of two passive bits after arbitration loss on a byte boundary can be enabled

22.2.2.1 4x Mode

- If high speed mode is used, all J1850 nodes should be configured to 4x mode when supported.
- Those nodes which do not support 4x mode need to tolerate high speed operation (no error sign).
- Enable bit EN4X.
- A Break symbol occurrence will generate an interrupt.
- After Break occurrence CPU needs to reset RX/TX status flags.
- The transceiver delay should not exceed 4 µs.

22.2.2.2 Break Operation

- Break allows bus communication to be terminated.
- All nodes are reset to a ‘reset-to-receive’ state (reset status bits by CPU).
- After Break symbol transition an IFS has to follow for re-synchronization purpose.
- If a break is sent, the current frame is ignored (if any).
- A break transmission can be generated setting bit SBRK.
- A break reception is indicated by bit BRK being set.

Note: After a hardware reset operation the SDLM module is disabled.

22.2.3 Interrupt Handling

The SDLM module can generate the interrupts on the following events:

- Protocol related interrupt conditions (combined to interrupt SDLM_I0):
  - End of frame detected
  - Break received
  - Arbitration lost
  - CRC error detected
  - Error detected
- Data receive/transmit interrupt conditions (combined to interrupt SDLM_I1):
  - Message transmitted
  - Message received
  - Header received

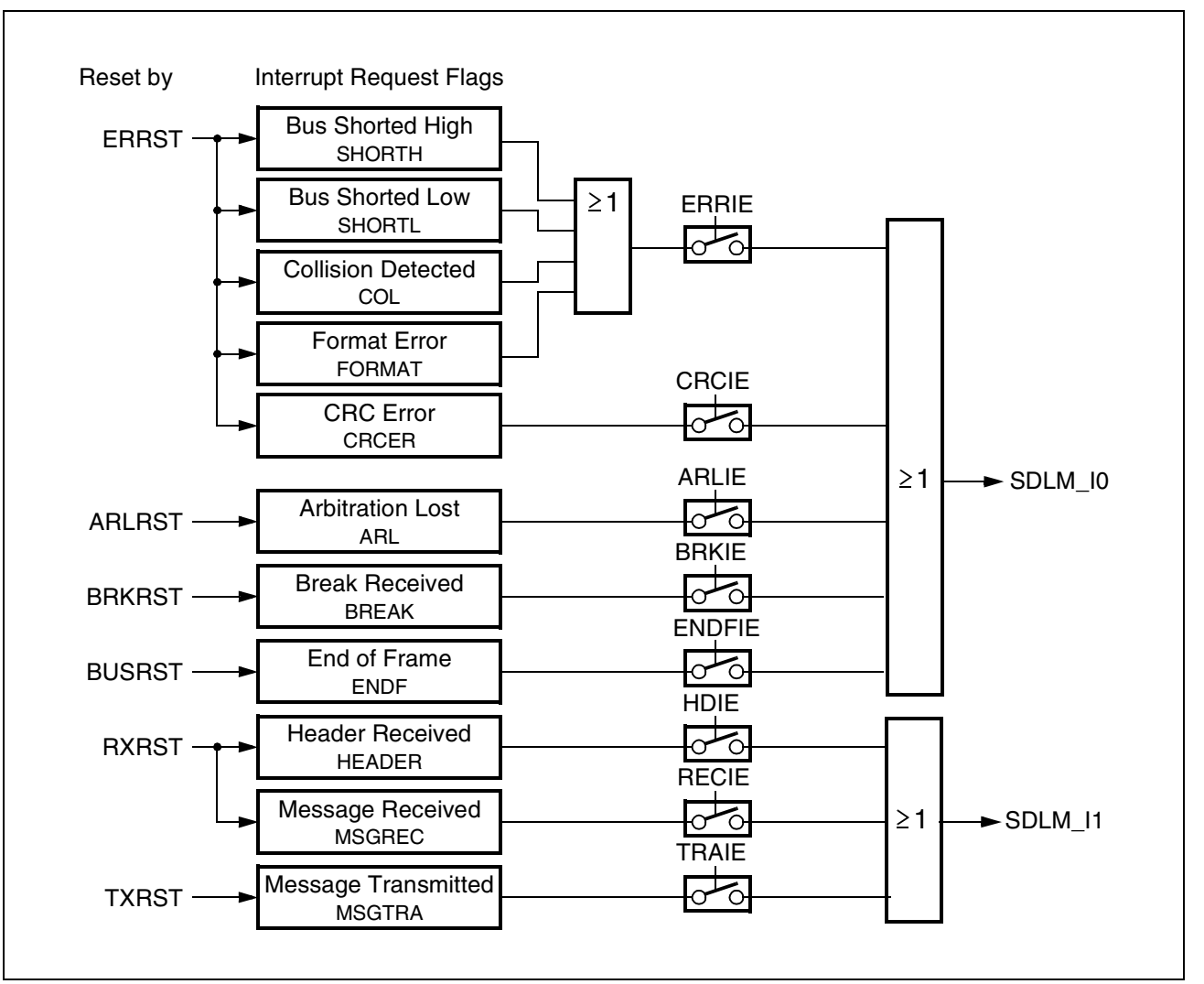


Figure 22-6 Interrupt Structure of SDLM

22.2.3.1 Message Operating Mode

Basically two receive buffers (11 byte each) and one 11 byte transmit buffer are available for data transfer. This allows the transfer of a complete J1850 frame without reloading data bytes. The access to the data is handled in FIFO mode (read/write to one address) in addition to random mode (read/write to selected bytes at consecutive addresses). In case of a loss of arbitration, an automatic retransmission is started, until the transmit request bit (TXRQ) is reset by the CPU. For correct transmission, the transmit buffer has to contain valid data (TXCPU > 0) when TXRQ is set.

22.2.3.2 Receive Operation

The receive buffer structure contains two independent 11 byte receive buffers. One of them is located on CPU side and can be directly accessed by the CPU (data and pointers). If this buffer is full (not yet completely read out), it can not be accessed by the J1850 module. Data reception over the bus is always done via the receive buffer on bus side. In order to release the receive buffer on CPU side, bit DONE has to be set. After complete reception of a frame, the buffer on J1850 side is declared full. If both buffers are full, the buffer on J1850 side can be overwritten by a new incoming frame, depending on the user-programmable overwrite enable bit (OVWR). If the CPU buffer is empty and the J1850 buffer is full, both buffers are swapped. By this action, the full buffer can be accessed by the CPU and the empty one is available on J1850 side.

The total number of received bytes in the corresponding buffer is indicated by bitfield RxCNT. Bitfield RxCPU indicates how many bytes have already been read out. In FIFO mode (RxINCE = 1), CPU data read actions take place via register RxD00 and RxCPU is automatically incremented by 1 after each read action. In Random Mode (RxINCE = 0), the buffer bytes can be directly accessed via their address. In this case, RxCPU is not incremented. In order to release a buffer for new message reception, the DONE bit has to be set. A receive interrupt is generated after complete reception of the whole frame (MSGREC = 1).

Register BUFFCON provides flags controlling the receive buffer:

- Receive Buffer Increment Enable (RxINCE): This bit enables FIFO Mode in addition to random mode: In random mode the CPU has access to each receive buffer byte via its address. In FIFO mode, the RxCPU pointer is incremented upon CPU read access until RxCPU == RxCNT (max. 11). This mode allows an easy CPU read transfer from the receive buffer only by addressing RxD00.

Register BUFFSTAT contains information about the receive buffer:

- RBC, RBB indicate valid data in the receive buffers
- MSGLST indicates a lost frame due to a full receive buffer
- Receive in Progress (RIP) indicates if any receive action is pending
- Break symbol reception is indicated by Break Received bit (BREAK)

Register TRANSSTAT contains information about the receive operation:

- HEADER: single byte or consolidated header is received (set after 1 or 3 received bytes)
- MSGREC: indicates a complete frame reception

22.2.3.3 Transmit Operation

Data transmission is started by setting the transmission request bit TXRQ. Transmission is aborted by resetting bit TXRQ by software. FIFO mode and random mode are working in the same way as it is described for data reception. A transmit interrupt can be generated after successful transmission of the complete frame (flag MSGTRA).

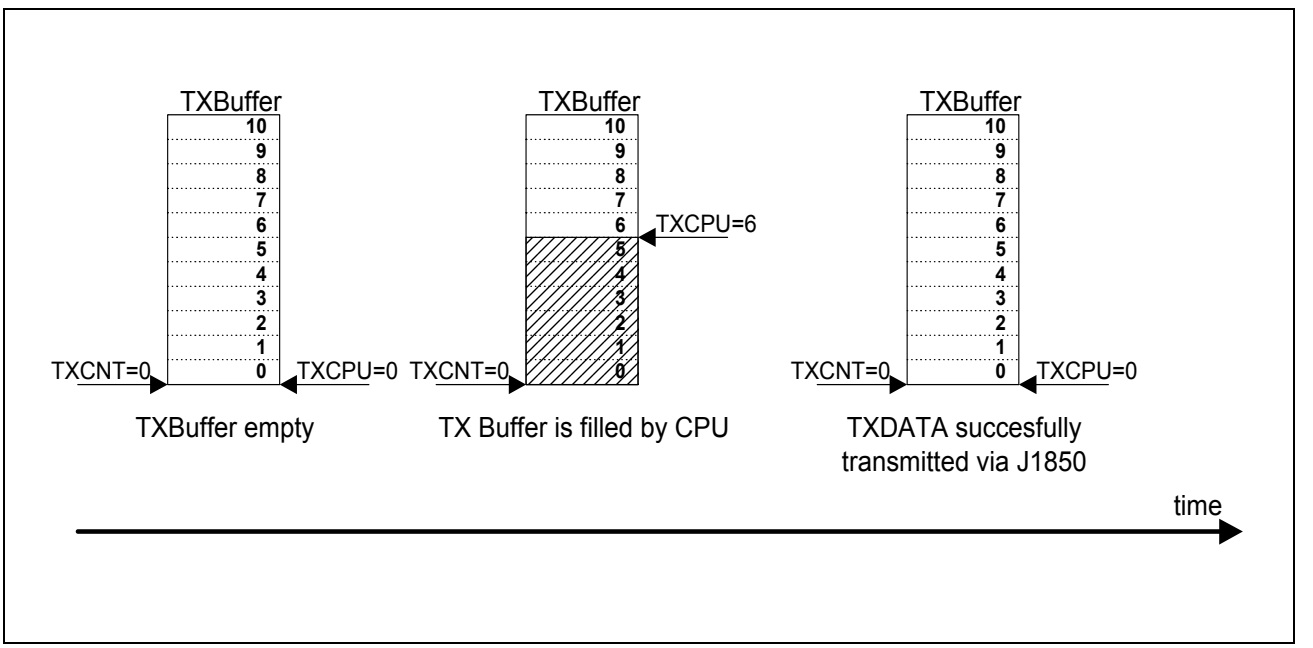


Figure 22-7 Transmit Operation

Register BUFFCON provides flags controlling the transmit buffer:

- TxINCE enables (analog to RxINCE) FIFO Mode for the transmit buffer
- TXRQ initiates a frame transmission to the J1850 bus. In case of a lost arbitration, the module automatically retries transmission until the frame has been correctly sent out or the transmit request has been reset by software
- CPU can initiate a break transmission by setting SBRK

Register TRANSSTAT contains information about transmit operations:

- CPU is informed when a message is currently transmitted (Transmission in Progress - TIP)
- MSGTRA indicates a successful frame transmission
- ARL: indicates that arbitration has been lost

22.2.4 In-Frame Response (IFR) Operation

The module supports automatic IFR transmission for type 1, 2 IFR for three-byte consolidated headers (no CPU load required). If the IFRs are handled via the transmit buffer, TxCPU indicates the number of bytes to be transmitted.

- If automatic IFR transmission function is not possible (single byte or one byte consolidated headers): If bit IFREN is not set, type 1 and 2 are handled via the transmit buffer, too. If IFREN is set, the value stored in IFRVAL will be transmitted if bit TxIRF is set.
- In case of automatic IFR transmission, register IFRVAL delivers the source ID. The value has to be written by the CPU first.
- IFR type 3 transmission can only be handled by the transmit buffer.
- Setting bit TxIFR initiates an IFR transmission (if automatic IFR not possible).
- Normalization symbol can be configured by the NB configuration bit.
- If IFR with CRC (Type 3, CRCEN = 1) is used, bit CRCERR indicates CRC error conditions.
- If register IFRVAL is used for transmission, no CRC will be sent out (not depending on CRCEN). If the transmit buffer is used, CRC will be sent out if bit CRCEN is set.
- Bit HEADER indicates complete reception of header byte(s) in the receive buffer on bus side.

Transmission of type 1 and type 2 IFRs for single byte headers and one byte consolidated headers is also accomplished by bit TxIFR, which has to be set by software. In case of automatic IFR (for type 1, 2 for three byte consolidated headers and IRFEN = 1), bit TxIFR is not needed.

3-byte consolidated headers: If IFREN is set, automatic response to type 1 and type 2 IFRs via the IFRVAL register is enabled. Type 3 IFR is sent by writing the IFR to the transmit buffer and then setting bit TxIFR. If IFREN is not set, all IFRs are transmitted via the transmit buffer if TxIFR is set.

Single byte headers and one byte consolidated headers: As there is no information in the header to indicate if an IFR is required, automatic transmission of Type 1 and 2 IFRs is not possible. If IFRs are used in the system, the header interrupt should be enabled in order to give time to decode the header through software to determine if an IFR is required. IFR transmission is always initiated by setting bit TxIFR. Type 3 IFR is always done via the transmit buffer, whereas types 1, 2 are handled either via the transmit buffer (IFREN = 0) or register IFRVAL (IFREN = 1).

22.2.5 Block Mode

In Block Mode, the SDLM supports transfer of frames of unlimited length (application specific). Block mode is selected by BMEN = 1. In this case, only one receive buffer and the transmit buffer are available. The swap functionality between the two RxBuffers is no longer supported.

In Block Mode, the receive and the transmit buffers are built as circular buffers of eight bytes length (transmit buffer) and 16 bytes length (receive buffer, see figures below). Access in random mode not being useful (but still supported), FIFO mode is automatically enabled (not depending on RxINCE/TxINCE). During transmission, a transmit interrupt can be generated each time TxCPU == TxCNT is detected after transmission of a byte. During reception, a receive interrupt can be generated if RxCNT0 != RxCPU0 after reception of a byte.
The counting sequence for the transmit buffer is … 6, 7, 0, 1, 2, … representing a circular buffer of 8 bytes length. The counting sequence for the receive buffer is … 14, 15, 0, 1, 2, … representing a circular buffer of 16 bytes length.

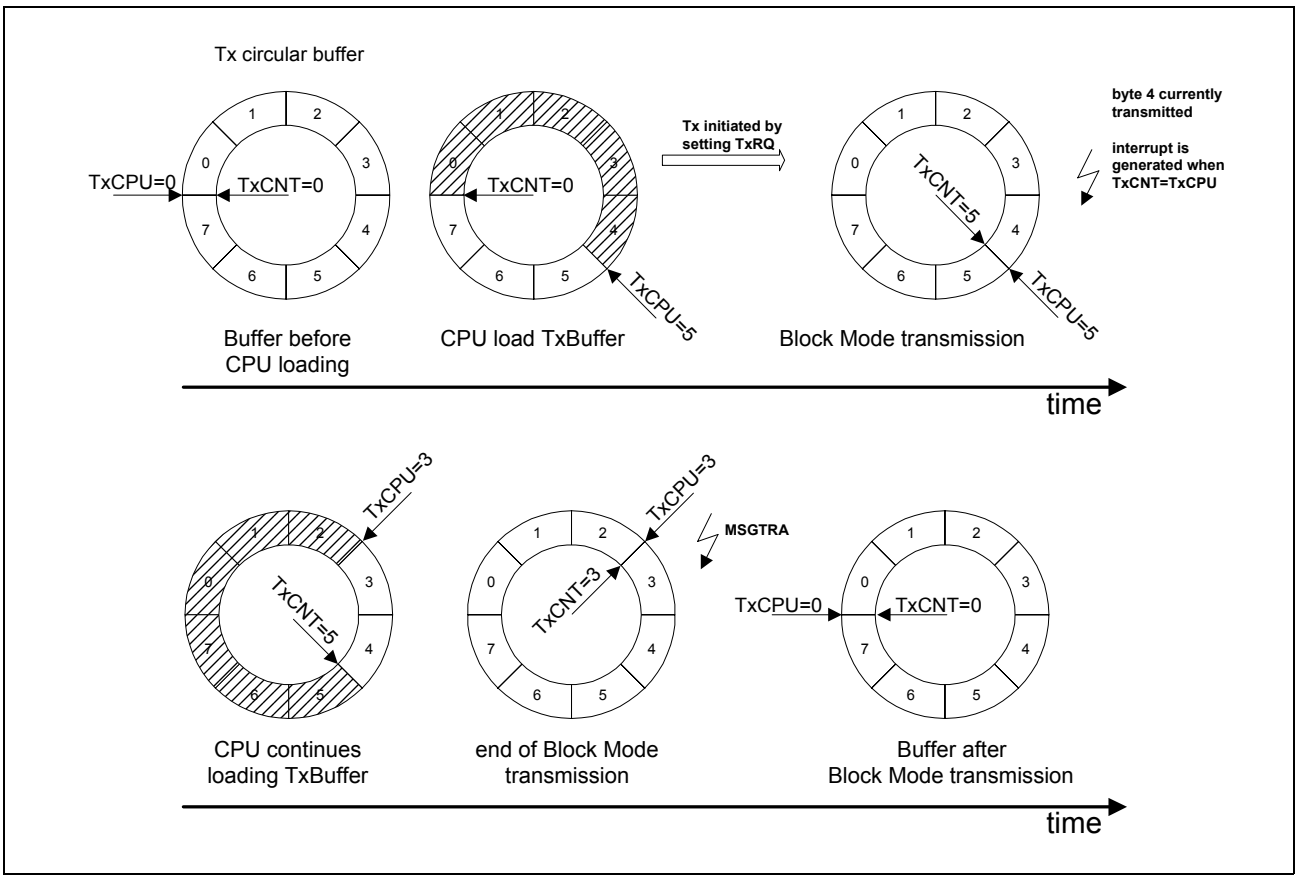


Figure 22-8 Transmit in Block Mode

Note: The SW has to take care that the condition TxCPU == TxCNT after writing a byte to the transmit buffer in block mode does not become true, otherwise the transmission will be finished.

In order to monitor the status of the bus during transmission, the SDLM always reads on the bus, even while transmitting. As a result, the user can check whether the message sent is equal to the message on the bus (test for arbitration).

The receive buffer in Block Mode can be accessed via register RxD00 on CPU side as FIFO base address (relative address 40_H). The elements of the FIFO can always be accessed via their addresses, too. The first eight bytes are located at the relative addresses 40_H to 47_H, the second eight bytes at the relative addresses 50_H to 57_H.

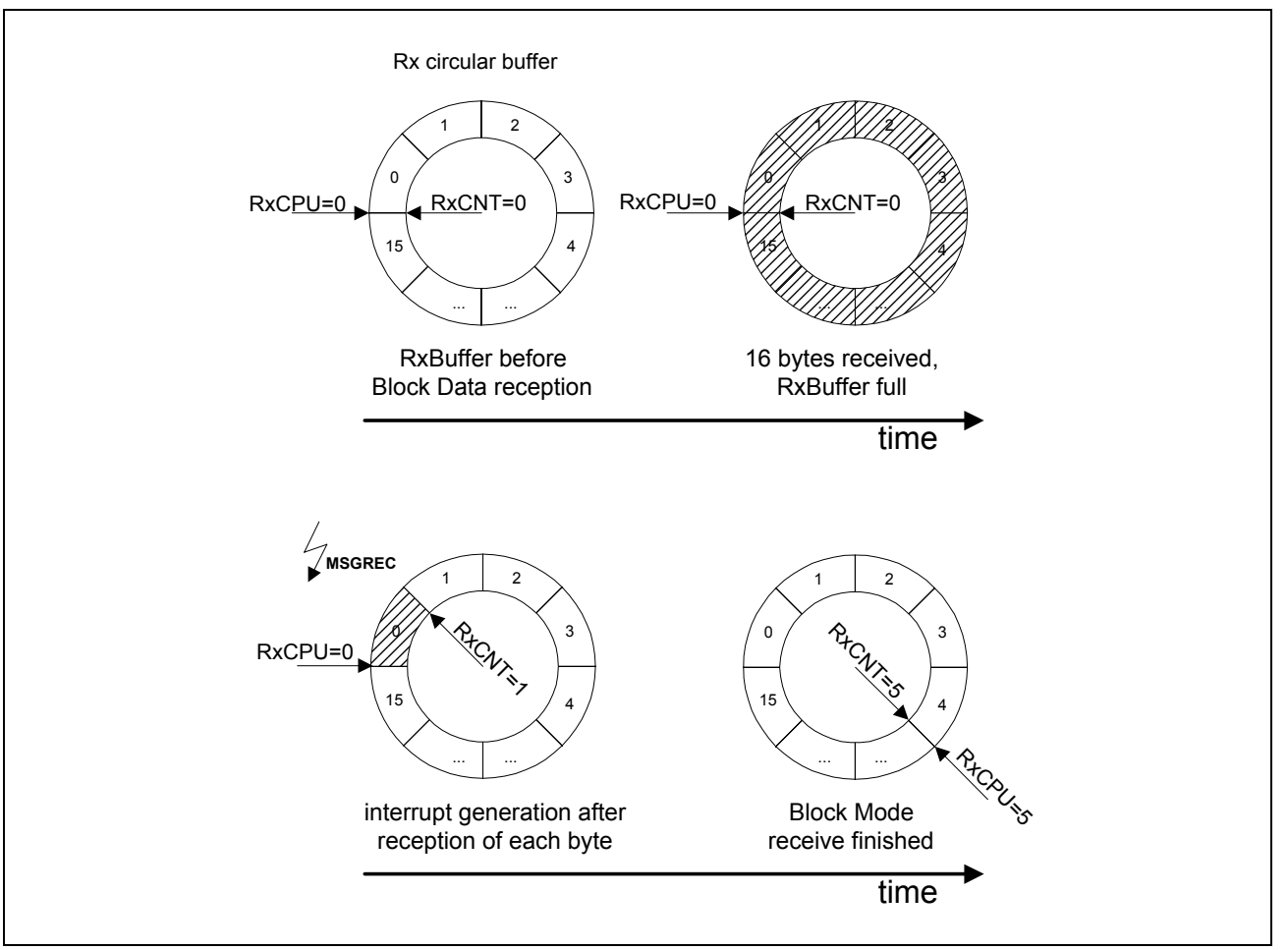


Figure 22-9 Data Reception in Block Mode

In Block Mode, the interrupt request flags MSGREC (reception) and MSGTRA (transmission) are automatically reset by hardware upon a read action from RxD00, or a write action to TxD0 respectively. RBB (= RBC) is set upon a pointer match after reception of a byte (receive buffer full) and reset by a read action from this buffer. MSGLST is set if RBB has been set before and a new data byte is received. MSGLST is not automatically reset by hardware. All error flags remain pending (once set) and have to be cleared by software. In case of a detected error during transmission, the transmit request bit TxRQ is reset by hardware in order to abort the transmission (no automatic retry).

22.2.6 Bus Access in FIFO Mode

In FIFO mode (block mode or normal mode), the FIFOs are byte-oriented. In order to avoid mismatch in case of word accesses, the LSB of the pointers on CPU side select which byte of the word is taken into account.

In the 16-bit implementation, the LSB of the CPU pointer (RxCPU or TxCPU, respectively) determines which byte is taken into account. The receive FIFO only delivers one data byte (its position in the word is selected by the LSB), the other byte is 0. The pointer is incremented by one after each read access to RxD00. The transmit buffer only takes over one byte per write access to TxD0, according to the LSB of the pointer. The other byte is not changed.
If the pointer's LSB is 0, any access to the corresponding buffer is based on the low byte (read: higher bytes = 0, low byte = FIFO value, write: only low byte written). If the pointer's LSB is 1, any access to the corresponding buffer is based on the high byte (read: high byte = FIFO value, lowbyte = 0, write: only high byte written). In any case, the pointer is incremented by 1.
Other accesses than to the receive or transmit buffers are always based on the low byte. Any word access to the SDLM (if FIFO mode is not selected) effects only the low-byte.

22.2.7 Flowcharts

22.2.7.1 Overview

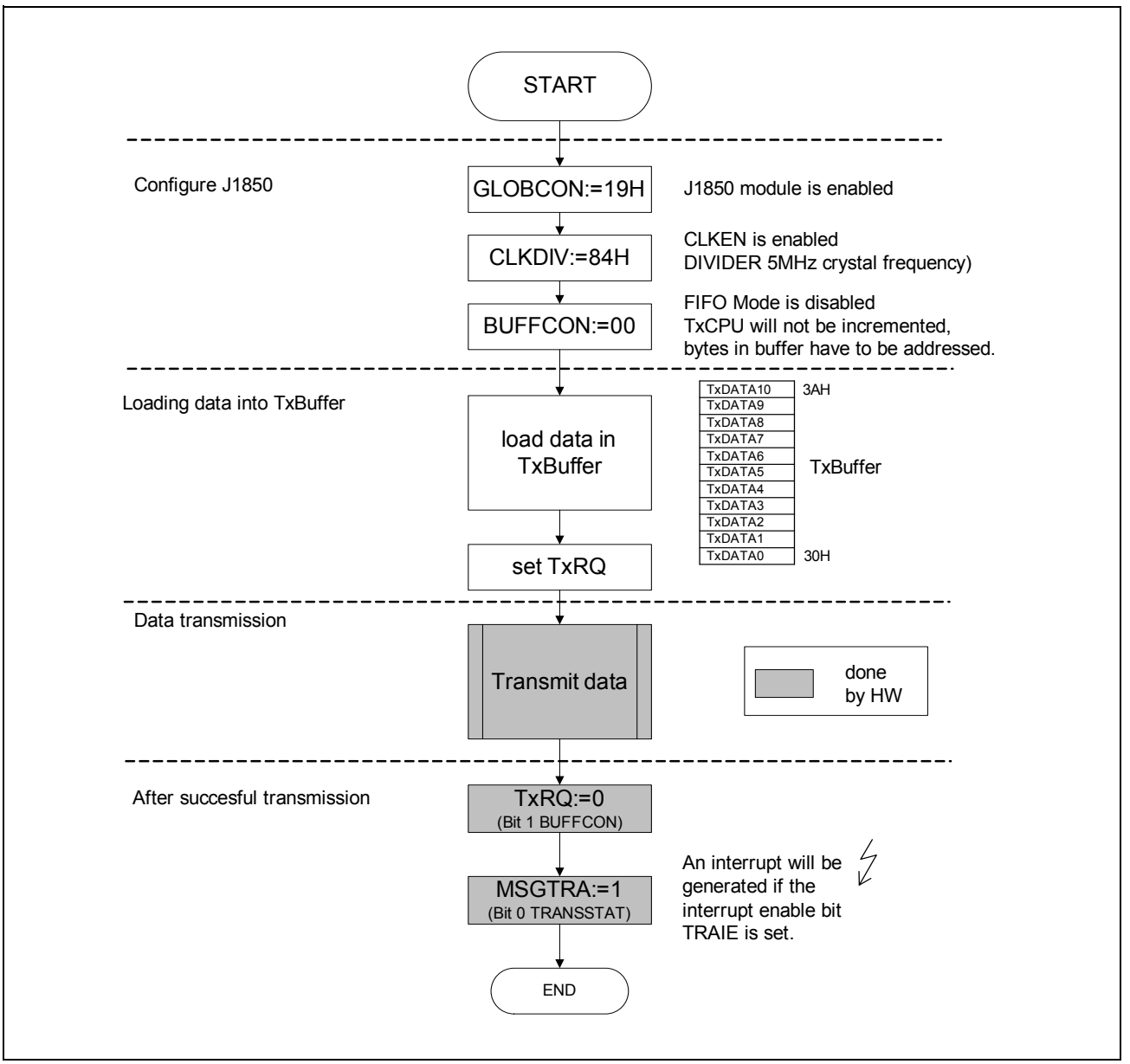


Figure 22-10 Initialization, Data Setup and Transmission

In order to adapt the timing to the transceiver device, register TxDELAY has to be configured, too. Furthermore, the desired J1850 receive pin and the transmit pin have to be selected. The interrupt line SDLM_I1 can be combined either with SDLM_I0 or can be independent.

22.2.7.2 Transmission Control

Transmission of a Standard Message in FIFO Mode

Bit TxINCE in register BUFFCON has to be set in order to provide FIFO functionality. Bitfield TxCPU is incremented after each write operation to TxD0. The transmit buffer is filled by multiple write actions to TxD0. All other registers of the transmit buffer can always be directly accessed via their addresses without changing TXCPU.

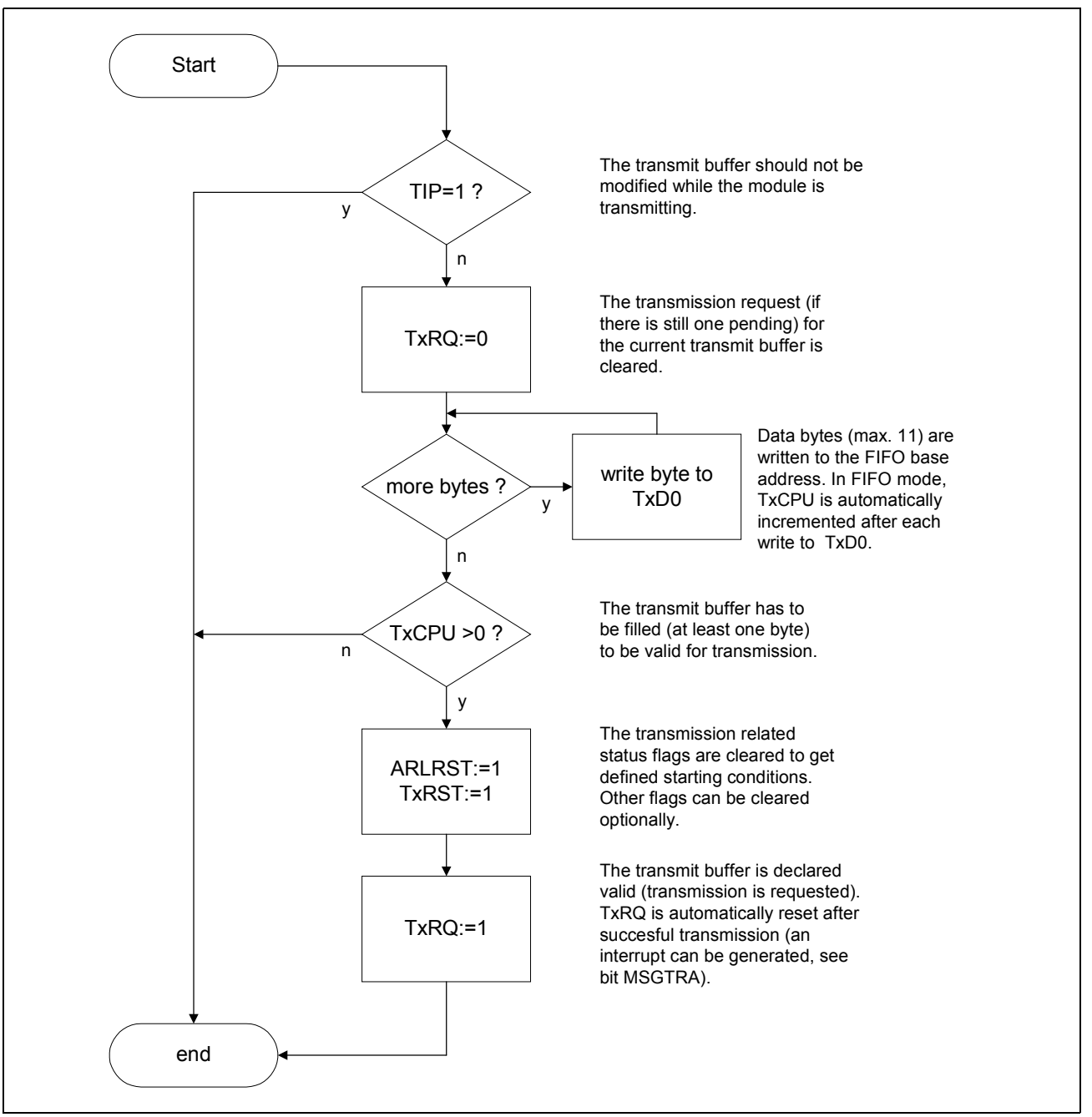


Figure 22-11 Transmission in FIFO Mode

Transmission of a Standard Message in Random Mode

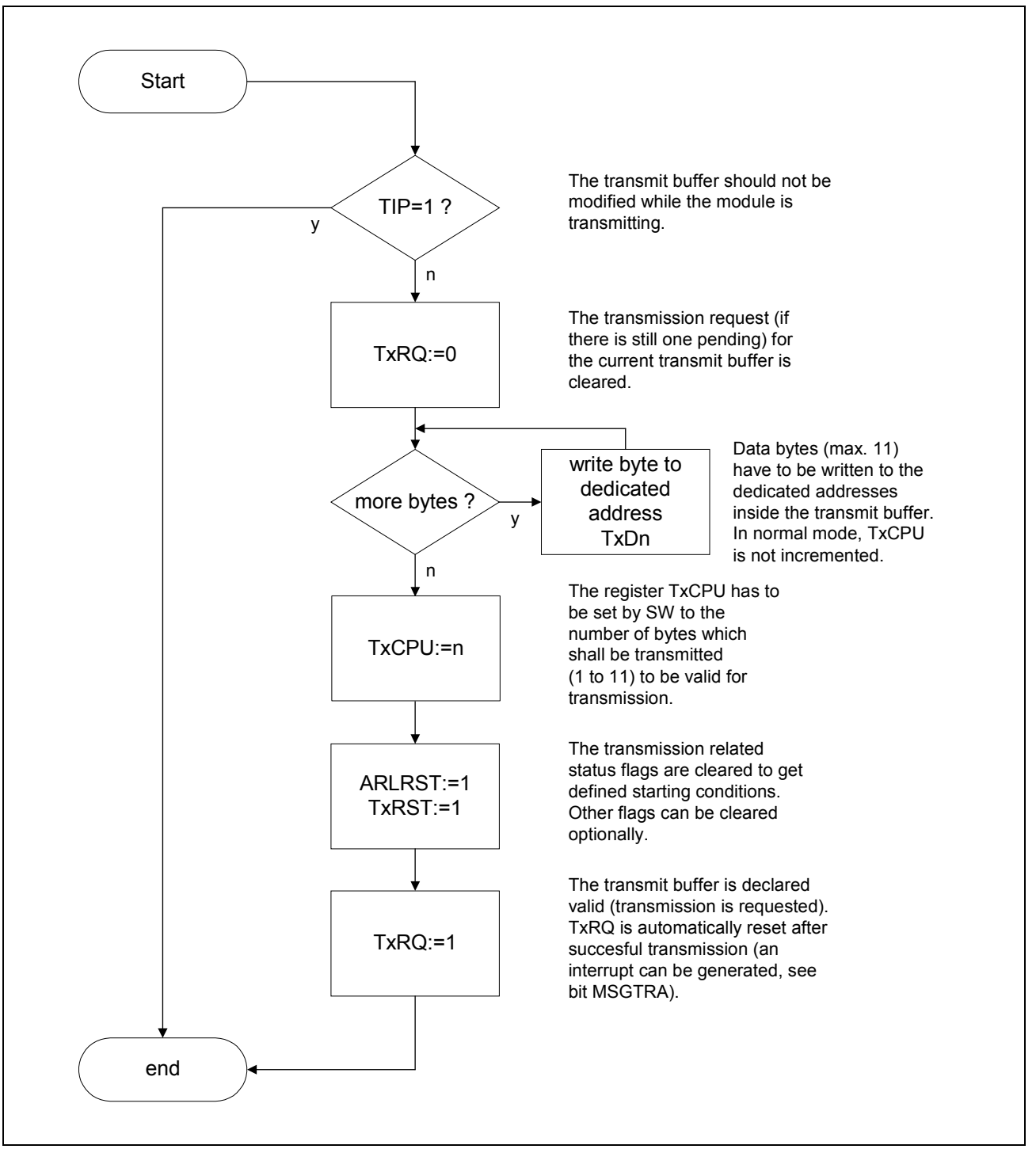


Figure 22-12 Transmission in Random Mode

Transmission in Block Mode

Block mode is selected by BMEN = 1 in register GLOBCON. In block mode, FIFO access is automatically enabled (not dependent on RxINCE or TxINCE). The transmit buffer in block mode is 8 bytes long.

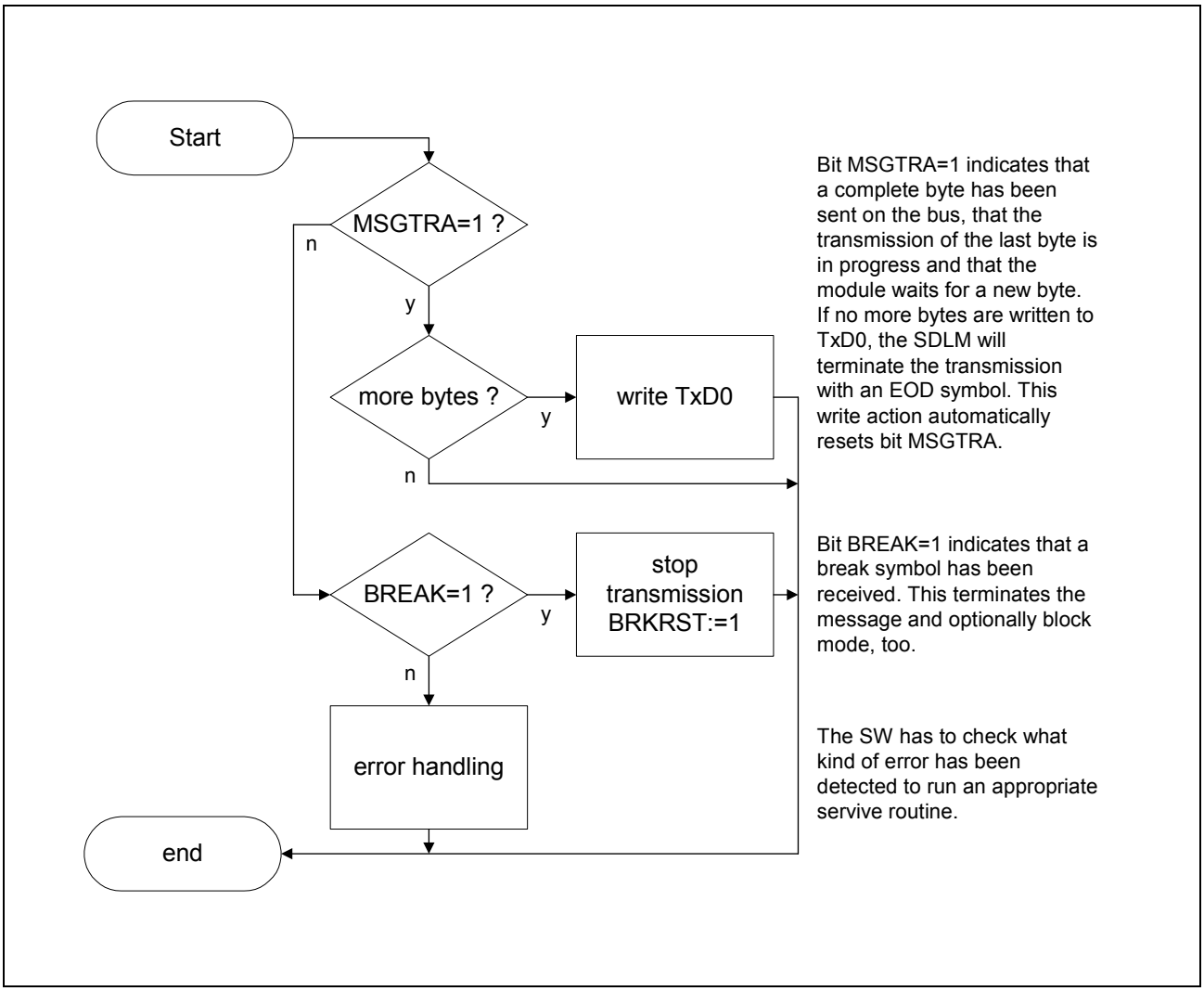


Figure 22-13 Transmission in Block Mode

22.2.7.3 Read Operations

Read of the Receive FIFO in Normal Mode

Bit RxINCE in register BUFFCON has to be set in order to provide FIFO functionality. Register RxCPU is incremented after each read operation from RxD00. The receive buffer is read out by multiple read actions from RxD00. All other registers of the receive buffer can always be directly accessed via their addresses without changing RxCPU.

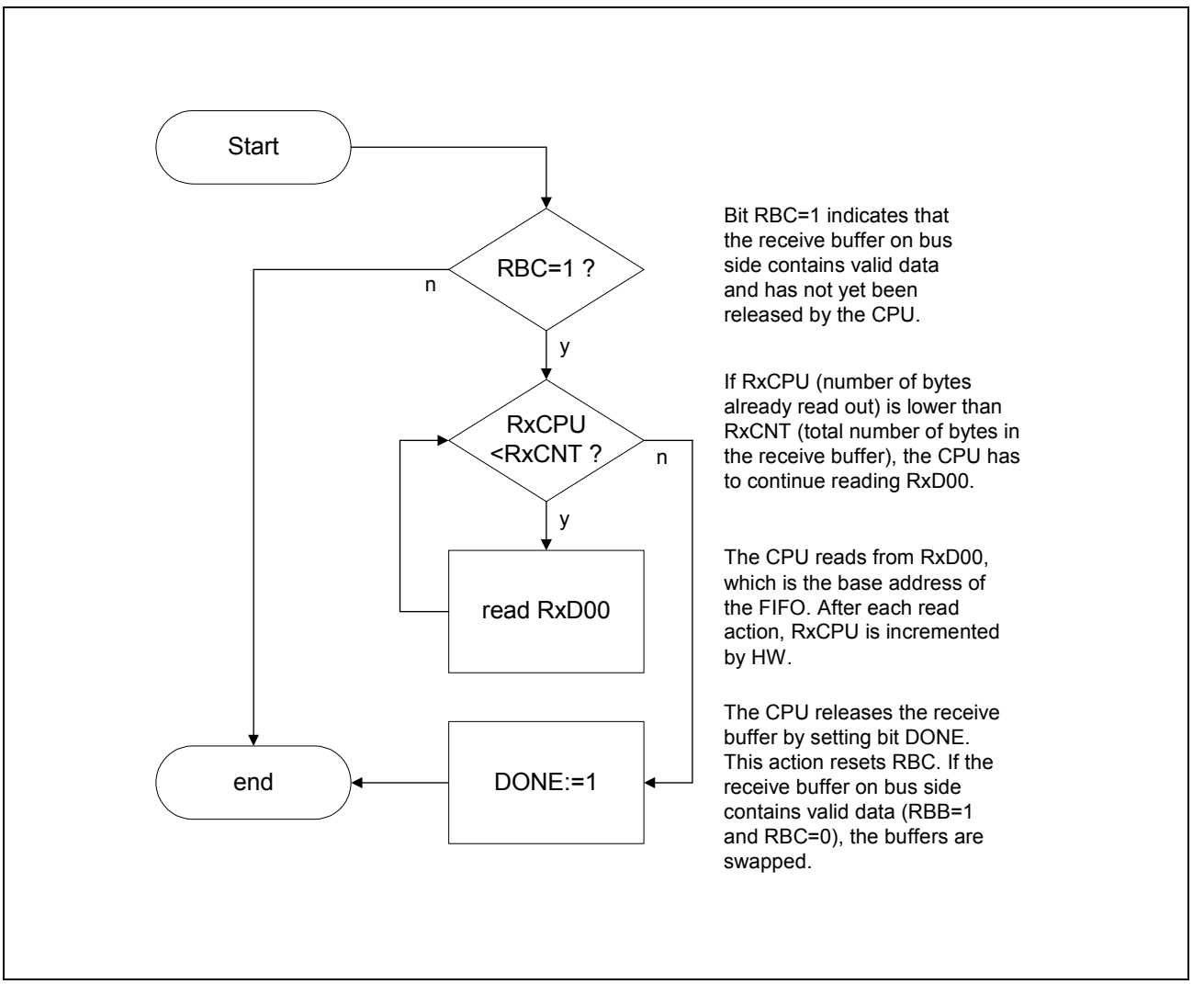


Figure 22-14 Receive Operation in FIFO Mode

Read Operation in Block Mode

Block mode is selected by BMEN = ‘1’ in register GLOBCON. In block mode, FIFO access is automatically enabled (not dependent on RxINCE or TxINCE). The receive buffer in block mode is 16 bytes long.

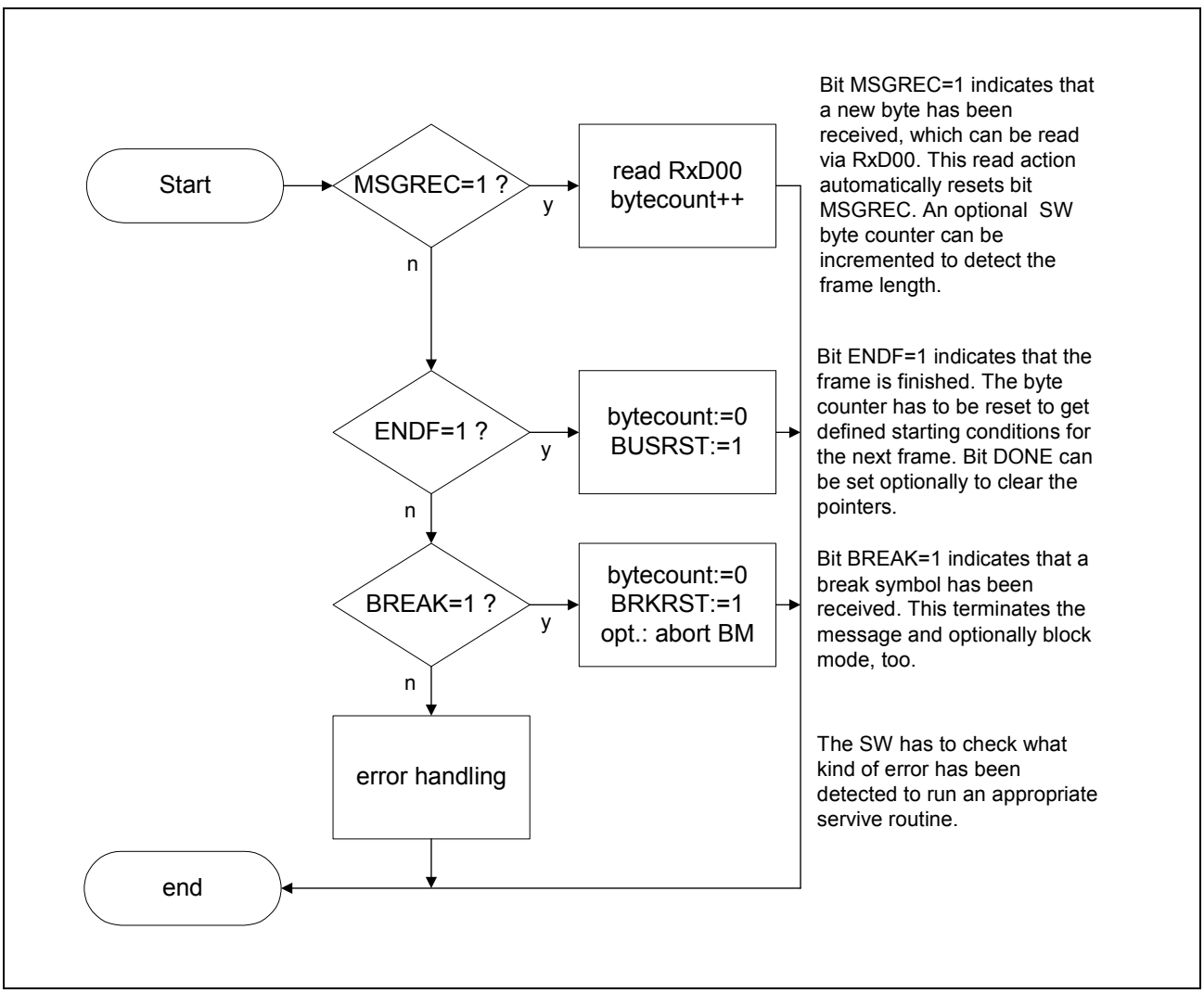


Figure 22-15 Reception in Block Mode

The value of the RxCPU pointer is automatically copied to register SPTR (start of frame pointer) to allow the user to detect the begin of a new frame.

22.2.8 IFR Handling

22.2.8.1 IFR Types 1, 2 via IFRVAL

The HEADER bit is automatically set by hardware after reception of the complete header (1 or 3 bytes) in the receive buffer on bus side. This buffer can be accessed at the consecutive relative addresses starting at base + 50_H. In case of 3-byte consolidated headers with the k bit set, an IFR (via IFRVAL) will be automatically generated if bit IFREN is set. The HEADER bit is reset when RxRST is set by software or after the reception of the complete frame. In case of single-byte headers or one-byte consolidated headers, the flowchart shows a possibility to send IFR.
If an IFR is requested (automatically or by hardware), the IFR byte(s) are sent after the EOD symbol. In case of type 2 IFR, automatic retry after arbitration loss takes place depending on bit ARIFR.

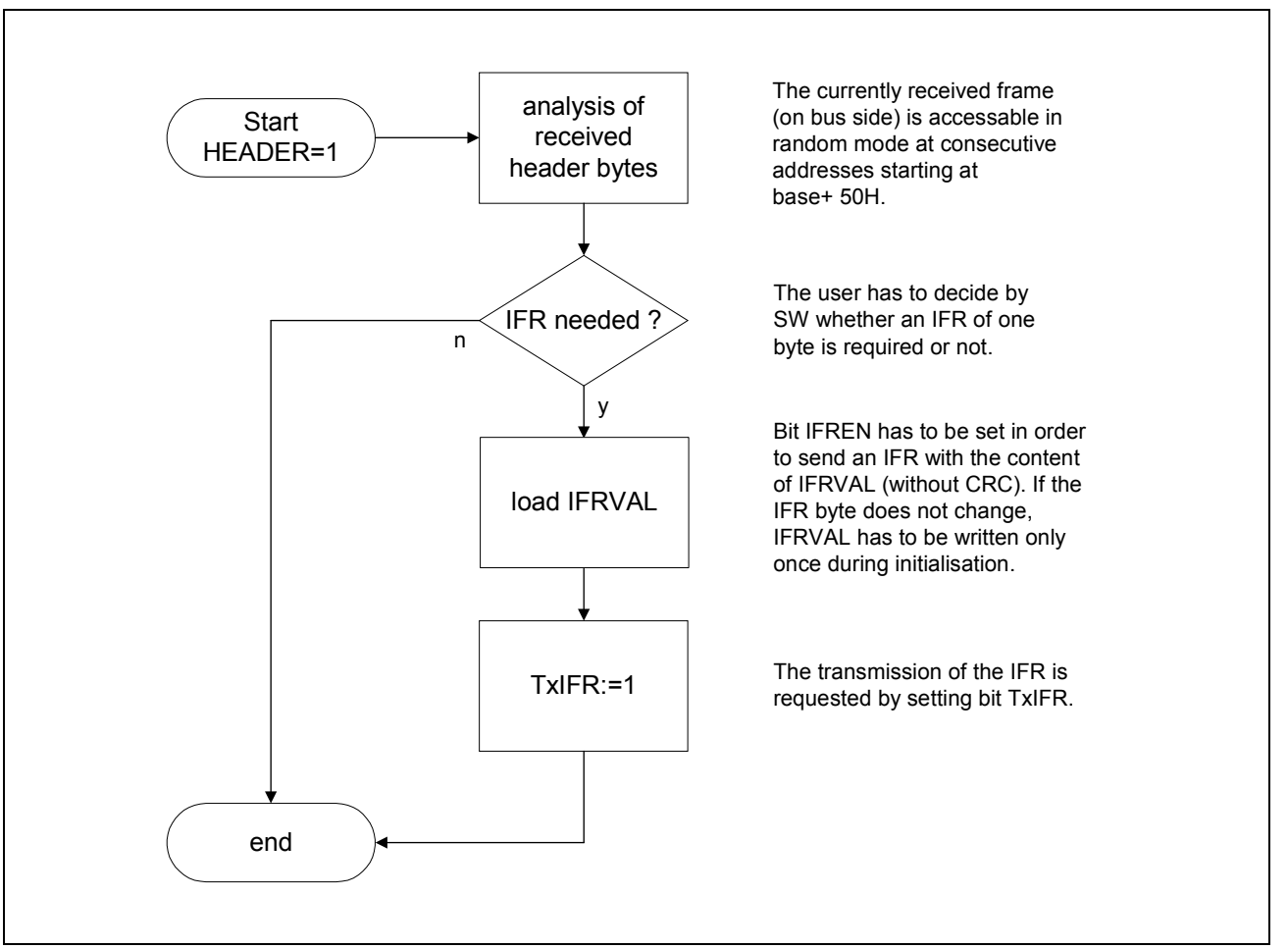


Figure 22-16 IFR Handling via IFRVAL

22.3 SDLM Register Description

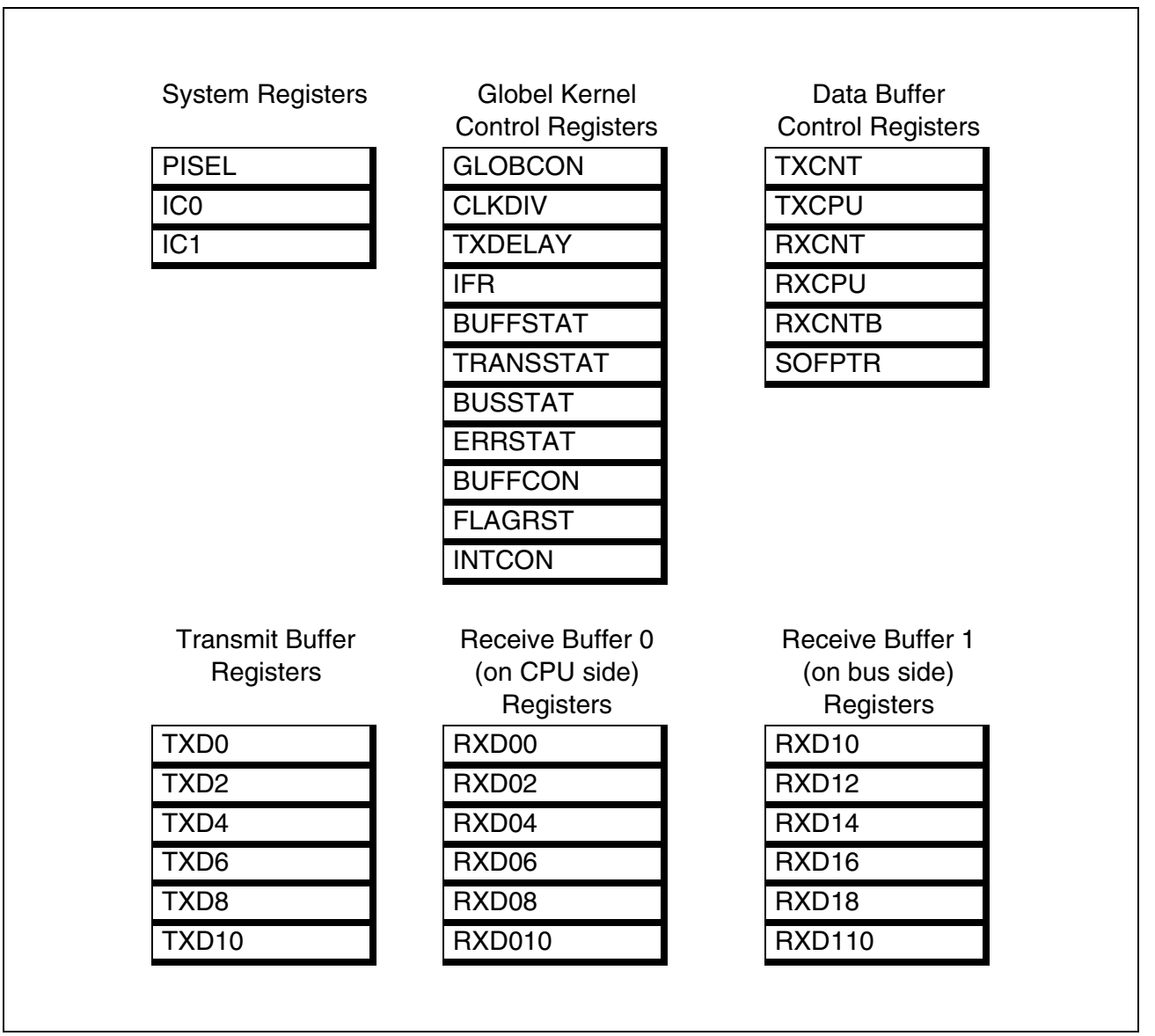


Figure 22-17 SDLM Register Overview

22.3.1 Global Control and Timing Registers

The Global Control Register contains bits to select different transfer modes and to determine the message handling.

GLOBCON
Global Control Register **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0								PB SEL	AR IFR	OV WR	NB	HDT	BM EN	EN 4x	GM EN
r								rw	rw	rw	rw	rw	rw	rw	rw

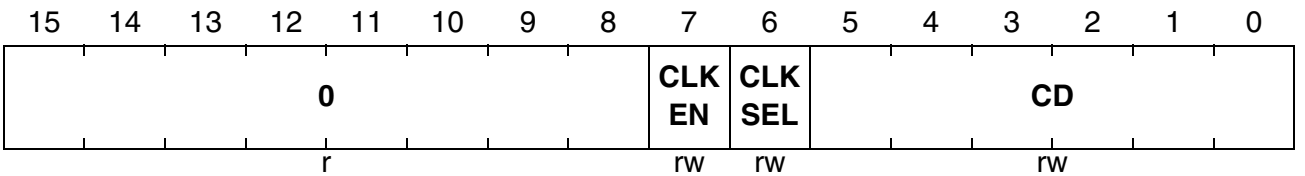
Field	Bits	Type	Description
GMEN	0	rw	**Global Module Enable** 0 The complete SDLM module is disabled. 1 The SDLM module is enabled and data transmission via the serial bus is possible. Resetting GMEN by software from '1' to '0' resets the module, except the timings.
EN4X[1)]	1	rw	**High Speed Transfer Enable (4x)** 0 The data transfer rate is 10.4 kbit/s. 1 The data transfer rate is 41.6 kbit/s.
BMEN	2	rw	**Block Mode Enable** 0 The maximum frame length is 11 data bytes (normal mode). 1 Transfers of frames longer than 11 data bytes are enabled. The transmit and the receive buffers are organized as circular buffers, which can be accessed in FIFO mode. Resetting BMEN resets the buffer pointers.
HDT	3	rw	**Header Type** 0 Single byte headers are supported. 1 Consolidated headers (1 byte and 3 bytes) are supported.
NB	4	rw	**Normalization Bit Polarity** 0 Normalization bit is functional 0. 1 Normalization bit is functional 1.

Field	Bits	Type	Description
OVWR	5	rw	**Overwrite Enable** 0 Overwrite action of the receive buffer on SDLM side in case of an incoming frame and a full receive buffer on bus side (RBB = 1) disabled. The frame on the bus is not accepted and is lost. 1 The full buffer on bus side is declared empty and then overwritten by the next incoming frame. The frame in the receive buffer on bus side is lost.
ARIFR	6	rw	**Automatic Retry of IFR** 0 The module will not retry transmission of IFR byte(s) in case of an arbitration loss (handling of IFR types 1, 3). The collision detection mechanism is generally enabled during IFR. 1 An automatic retry of IFR transmission in case of arbitration loss is enabled (for IFR type 2). The collision detection mechanism is disabled during EOD.
PBSEL	7	rw	**Passive Bits Select** 0 When loosing arbitration during the last bit of a byte (on a byte boundary) during the normal frame, two passive bits are automatically transmitted by the module. The passive bits are never added during IFR. 1 No extra action after the loss of arbitration (the two passive bits are not added).
0	[15:8]	–	**Reserved;** returns ‘0’ if read; should be written with ‘0’.

1) In order to support transmission in 4x mode, the transceiver delay should not exceed 4 µs.

Register CLKDIV allows for configuration of the internal timings.

CLKDIV
Clock Divider Register **Reset Value: 0000_H**

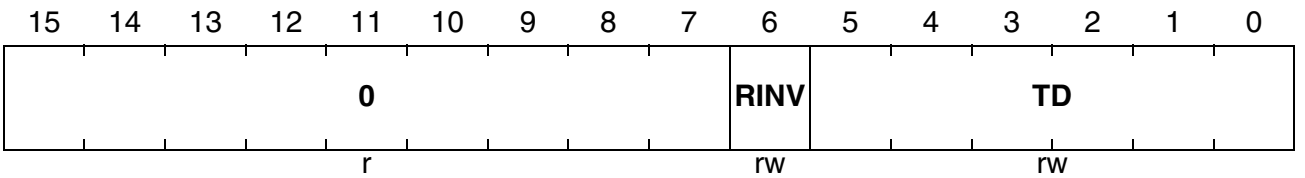


Field	Bits	Type	Description
CD	[5:0]	rw	**Clock Divider Bits** This bitfield defines the value of the clock divider. In order to run the module with different system frequencies, the divider can be programmed from 1 to 64. 000000_B System clock/module clock = 1 000001_B System clock/module clock = 2 000010_B System clock/module clock = 3 000011_B System clock/module clock = 4 … 111110_B System clock/module clock = 63 111111_B System clock/module clock = 64
CLKSEL	6	rw	**Clock Select** 0 1.00 MHz module clock 1 1.05 MHz module clock
CLKEN[1)]	7	rw	**Clock Enable** 0 Module clock is gated off (protocol layer not clocked) in order to reduce power consumption. 1 Module is clocked, protocol layer working.
0	[15:8]	–	**Reserved;** returns ‘0’ if read; should be written with ‘0’.

1) Only registers GLOBCON, CLKDIV and IPCR can be accessed if CLKEN = 0.

Register TxDELAY allows for the compensation of the transceiver delay.

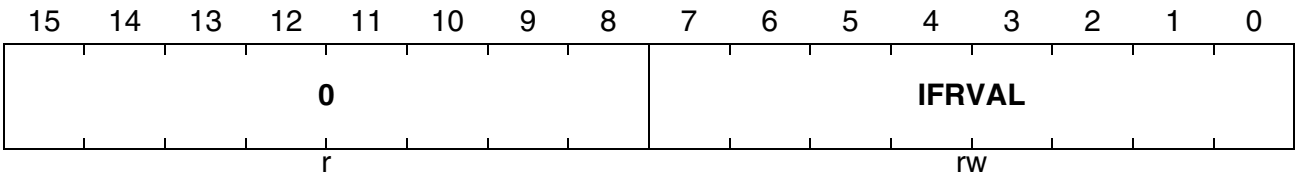
TxDELAY
Transceiver Delay Register **Reset Value: 0014_H**



Field	Bits	Type	Description
TD	[5:0]	rw	**Transceiver Delay Bits** This bitfield defines the transceiver delay, which is taken into account by the J1850 bitstream processor. The value of TD determines the number of module clock cycles, which are taken into account. The reset value equals 20 µs @ 1.00 MHz or 19 µs @ 1.05 MHz.
RINV	6	rw	**Invert Receive Input** 0 Receive pin polarity is not inverted. 1 Receive pin polarity is inverted.
0	[15:7]	–	**Reserved;** returns ‘0’ if read; should be written with ‘0’.

Register IFR contains the IFRVAL bitfield, which can be sent out as source ID in case of an one-byte IFR (automatic transmission or triggered by SW).

IFR
In-Frame Response Value Register **Reset Value: 0000_H**

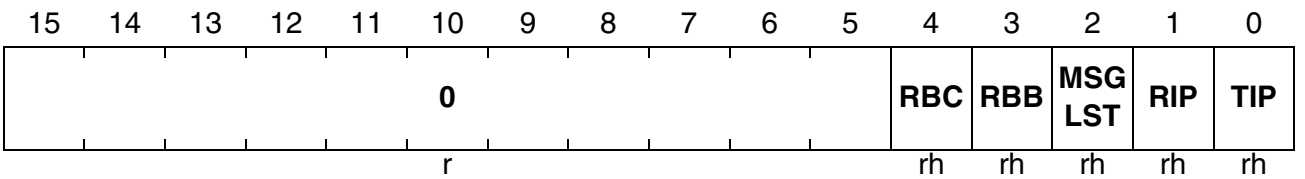


Field	Bits	Type	Description
IFRVAL	[7:0]	rw	**In-Frame Response Value** Bitfield IFRVAL contains the value to be transmitted in case of requested in-frame response (type 1, 2 IFR, 1 byte response). Has to be enabled by bit IFREN and initialized by the CPU.
0	[15:8]	–	**Reserved;** returns ‘0’ if read; should be written with ‘0’.

22.4 Control and Status Registers

Register BUFFSTAT contains the buffer-related status flags.

BUFFSTAT
Buffer Status Register **Reset Value: 0000_H**

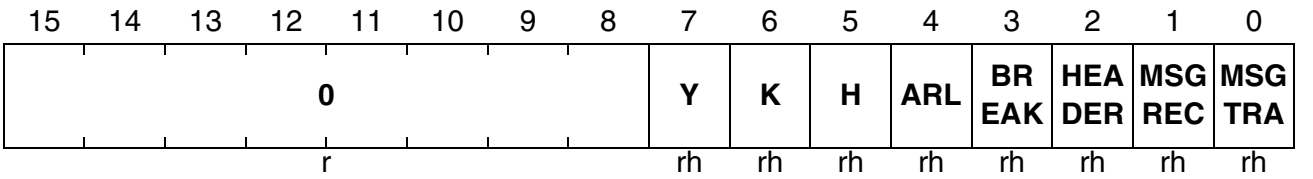


Field	Bits	Type	Description
TIP	0	rh	**Transmission in Progress** 0 The SDLM module does not currently send a frame on the bus or has finished its transmission. 1 The SDLM module is sending the contents of its transmit buffer or IFR.IFRVAL on the bus and has not yet lost arbitration. TIP is reset by hardware when the arbitration is lost or EOD or ENDF are detected.
RIP	1	rh	**Reception in Progress** 0 The SDLM module does not currently receive a frame on the bus. 1 The SDLM module is receiving a frame on the bus. RIP is reset by hardware when ENDF is detected.
MSGLST	2	rh	**Message Lost** Bit MSGLST indicates an incoming frame when the receive buffer on bus side is full (RBB = 1, contains received message data and has not yet been swapped). Bit MSGLST is reset when MSGREC is reset in normal mode. If overwrite is enabled, the receive buffer on bus side will be overwritten. In block mode, MSGLST is set if RBB is set and a new byte is received. If OVWR = 0, the new byte is not stored. If OVWR = 1, the new byte is stored. MSGLST has to be reset by software.

Field	Bits	Type	Description
RBB	3	rh	**Receive Buffer on Bus Side Full** RBB = ‘1’ indicates that the receive buffer on J1850 side is full. In case of a new incoming message, this buffer is declared empty (RBB = 0) and overwritten by the new data if bit OVWR = ‘1’. If bit OVWR = 0, the buffer status and its contents are not changed (new data is not received). RBB is reset by hardware when the buffer is swapped to CPU side. In block mode, (only one buffer, so RBB = RBC) RBB is set upon a pointer match after reception of a byte. It is reset by reading from the receive buffer.
RBC	4	rh	**Receive Buffer on CPU Side Full** RBC = ‘1’ indicates that the receive buffer on CPU side is full (not empty). This buffer remains allocated by the CPU and bit RBC remains set until bit DONE has been set by software.
0	[15:5]	–	**Reserved;** returns ‘0’ if read.

Register TRANSSTAT contains transmission-related status flags and monitors three functional bits of the header of the currently received frame.

TRANSSTAT
Transmission Status Register **Reset Value: 0000_H**

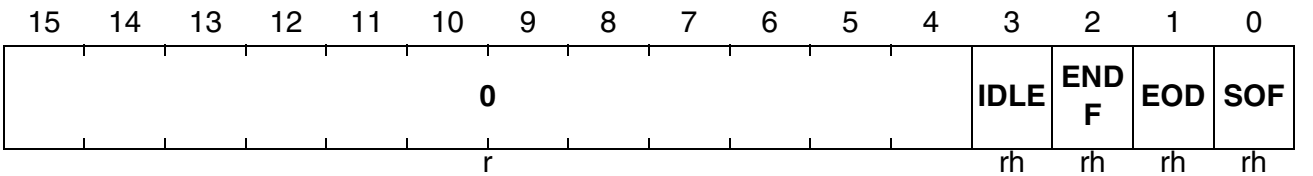


Field	Bits	Type	Description
MSGTRA	0	rh	**Message Transmitted** **Normal Mode:** MSGTRA indicates the complete transmission of the TxBuffer or IFRVAL (arbitration won and EOD detected). Reset by bit TxRST. **Block Mode:** MSGTRA is set upon a pointer match after transmission of a byte or ENDF detection. It is reset when the CPU writes to the transmit buffer or by bit TXRST.
MSGREC	1	rh	**Message Received** **Normal Mode:** MSGREC indicates the reception of a new frame in the receive buffer on bus side (detection of ENDF). It is reset by bit RxRST or by hardware if the buffer is overwritten. **Block Mode:** MSGREC is set if the pointers do not match after reception of a byte (FIFO not empty) or ENDF detection. It is reset when the CPU reads from the receive buffer or by bit RXRST.
HEADER	2	rh	**Header Received** HEADER is set by hardware after reception of the complete header byte(s). It is reset by hardware after reception of the complete frame.
BREAK	3	rh	**Break Received** If BREAK is set, a break symbol has been received on the J1850 bus. Has to be reset by software.

Field	Bits	Type	Description
ARL	4	rh	**Arbitration Lost** ARL is set after the arbitration for transmission has been lost. It is reset by software or by hardware if the arbitration has been won or the transmission has been aborted.
H	5	rh	**H Bit in Consolidated Headers** This bit monitors the status of bit 4 of the first byte in a frame on bus side.
K	6	rh	**K Bit in 3 Byte Consolidated Headers** This bit monitors the status of bit 3 of the first byte in a frame on bus side.
Y	7	rh	**Y Bit in 3 Byte Consolidated Headers** This bit monitors the status of bit 2 of the first byte in a frame on bus side.
0	[15:8]	–	**Reserved;** returns ‘0’ if read.

Register BUSSTAT contains bus-related status bits.

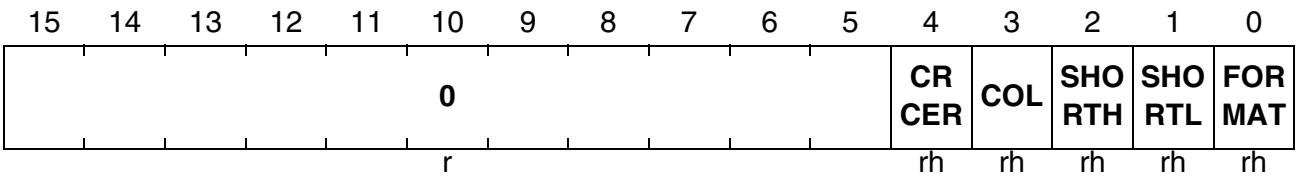
BUSSTAT
Bus Status Register **Reset Value: 0000_H**



Field	Bits	Type	Description
SOF	0	rh	**Start Of Frame Detected** Indicates the detection of SOF; bit is reset by BUSRST.
EOD	1	rh	**End Of Data Detected** Indicates the detection of EOD; bit is reset by BUSRST.
ENDF	2	rh	**End Of Frame Detected** Indicates the detection of EOF; bit is reset by BUSRST
IDLE	3	rh	**Bus Idle** This bit is set after IFS. It is reset by HW if a new frame has started.
0	[15:4]	–	**Reserved;** returns ‘0’ if read.

Register ERRSTAT contains error bits. The bits in this register have to be reset by SW.

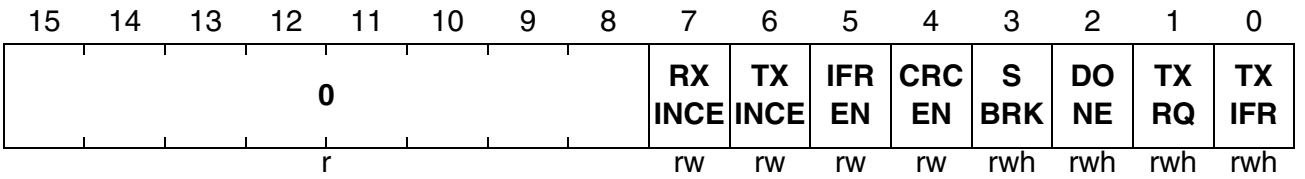
ERRSTAT
Error Status Register **Reset Value: 0000_H**



Field	Bits	Type	Description
FORMAT	0	rh	**Format Error** Bit is set if a frame length error, byte length error, symbol timing error or bit timing error occurred.
SHORTL	1	rh	**Bus Shorted Low** This bit is set if the device tries to sent data, but permanently reads a ‘0’ on the bus.
SHORTH	2	rh	**Bus Shorted High** This bit is set if a ‘1’ is detected on the bus for more than one second.
COL	3	rh	**Collision Detected** This bit is set if a collision has been detected on the bus.
CRCER	4	rh	**CRC Error** This bit is set if the calculated CRC differs from the received one.
0	[15:5]	–	**Reserved;** returns ‘0’ if read.

Register BUFFCON contains the transfer-related control bits, including IFR control and FIFO control.

BUFFCON
Buffer Control Register **Reset Value: 0000_H**



Field	Bits	Type	Description
TXIFR	0	rwh	**Transmit In-Frame Response** Setting bit TXIFR declares the transmit buffer or the IFR register (if IFREN = 1, IFR type 1, 2, others than 3 byte consolidated header) to be valid for IFR and initiates its transmission. TXIFR is automatically reset by hardware after successful transmission. Resetting TxIFR by software stops the transmission of the IFR.
TXRQ	1	rwh	**Transmit Request** Setting bit TXRQ declares the transmit buffer to be valid and starts its transmission. TXRQ is automatically reset by hardware after successful transmission. Resetting TxRQ by software stops the transmission in Normal Mode and in Block Mode. TXRQ can not be used to start IFR transmission from the transmit buffer. If TXRQ is reset, TXCPU is cleared.
DONE	2	rwh	**Receive Buffer on CPU Side Read Out Done** Setting bit DONE declares the receive buffer on CPU side to be empty, resets RXCPU and releases the buffer (reset of RBC). If there is a full receive buffer on bus side, the buffers are swapped. This bit is reset by hardware after the buffer has been released.
SBRK	3	rwh	**Send Break** Setting bit SBRK initiates the transmission of a break symbol on the J1850 bus. Bit SBRK is reset by hardware after having sent the break symbol.

Field	Bits	Type	Description
CRCEN	4	rw	**CRC Enable** 0 No CRC generation for type IFR via TxBuffer 1 CRC enabled for IFR via TxBuffer If IFR is sent from IFRVAL, no CRC is generated (even if CRCEN = 1). CRC generation is always enabled in normal mode and in block mode.
IFREN	5	rw	**In-Frame Response Enable** Setting bit IFREN enables the automatic IFR (type1, 2 for 3 byte consolidated header) of the SDLM module with the value stored in IFRVAL. If the IFR request can not be automatically detected (all headers, except see above), IFREN selects the data source for types 1, 2 IFR initiated by TXIFR. 0 Transmit buffer contains IFR data byte(s), IFR types 1, 2, 3 supported, CRC depending on CRCEN. 1 IFRVAL contains data byte for types 1, 2 IFR. Automatic IFR for types 1, 2 for 3 byte consolidated headers supported. No CRC is used.
TXINCE	6	rw	**Transmit Buffer Increment Enable** Random access mode is always enabled. 0 FiFO mode disabled for transmit buffer. 1 FiFO mode enabled, TXCPU is incremented by one after each CPU write operation to the TXD0. In block mode, FIFO mode is automatically enabled (not depending on TXINCE).
RXINCE	7	rw	**Receive Buffer Increment Enable** Random access mode is always enabled. 0 FiFO mode disabled to receive buffer on CPU side. 1 FiFO mode enabled, RXCPU is incremented by one after each CPU read operation to RXD00. In block mode, FIFO mode is automatically enabled (not depending on RXINCE).
0	[15:8]	–	**Reserved;** returns ‘0’ if read; should be written with ‘0’.

Register FLAGRST contains the control bits to reset the error flags, the bus-related flags and the transfer-related status bits.

FLAGRST
Flag Reset Register **Reset Value: 0000_H**

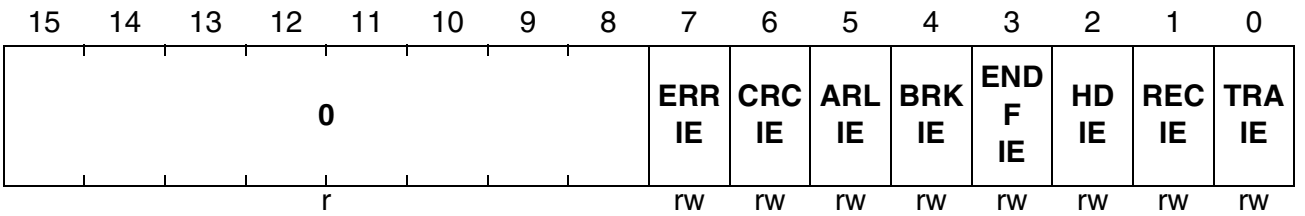
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0										ERRST	BUSRST	RXRST	TXRST	ARLRST	BRKRST
r										wh	wh	wh	wh	wh	wh

Field	Bits	Type	Description
BRKRST[1)]	0	wh	**Reset Buffer Status Flag** resets BREAK
ARLRST[2)]	1	wh	**Reset Buffer Status Flag** resets ARL
TXRST[3)]	2	wh	**Reset Buffer Status Flag** resets MSGTRA
RXRST[4)]	3	wh	**Reset Buffer Status Flag** resets MSGREC and MSGLST
BUSRST[5)]	4	wh	**Reset Bus Status Flags** resets ENDF, EOD, SOF
ERRST[6)]	5	wh	**Reset Error Flags** resets SHORTH, SHORTL, COL, CRCER, FORMAT
0	[15:6]	–	**Reserved;** returns ‘0’ if read; should be written with ‘0’.

1) This bit is automatically reset by HW after clearing bit BREAK and delivers ‘0’ when read.
2) This bit is automatically reset by HW after clearing bit ARL and delivers ‘0’ when read.
3) This bit is automatically reset by HW after clearing bit MSGTRA and delivers ‘0’ when read.
4) This bit is automatically reset by HW after clearing bits MSGREC and MSGLST and delivers ‘0’ when read.
5) This bit is automatically reset by HW after clearing bits ENDF, EOD and SOF and delivers ‘0’ when read.
6) This bit is automatically reset by HW after clearing bits SHORTH, SHORTL, COL, CRCER and FORMAT and delivers ‘0’ when read.

Register INTCON contains all interrupt enable bits.

INTCON
Interrupt Control Register **Reset Value: 0000_H**



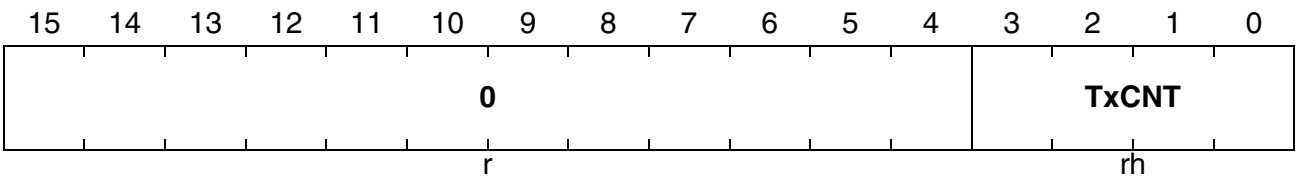
Field	Bits	Type	Description
TRAIE	0	rw	**Enable Transmit Interrupt** The transmission interrupt is disabled. An interrupt is generated if bit MSGTRA is set.
RECIE	1	rw	**Enable Receive Interrupt** The receive interrupt is disabled. An interrupt is generated if bit MSGREC is set.
HDIE	2	rw	**Enable Header Received Interrupt** The header interrupt is disabled. An interrupt is generated if bit HEADER is set.
ENDFIE	3	rw	**Enable End of Frame Detection** The end-of-frame interrupt is disabled. An interrupt is generated if bit ENDF is set.
BRKIE	4	rw	**Enable Break Received Interrupt** The break interrupt is disabled. An interrupt is generated if bit BREAK is set.
ARLIE	5	rw	**Enable Arbitration Lost Interrupt** The arbitration-lost interrupt is disabled. An interrupt is generated if bit ARL is set.
CRCIE	6	rw	**Enable CRC Error Interrupt** The CRC error interrupt is disabled. An interrupt is generated if bit CRCER is set.
ERRIE	7	rw	**Enable Error Interrupt** The error interrupt is disabled. An interrupt is generated if one of the bits SHORTH, SHORTL or FORMAT is set.[1)]
0	[15:8]	–	**Reserved;** returns ‘0’ if read; should be written with ‘0’.

1) The COL flag does not generate an error interrupt!

22.4.1 Transmission Related Registers

Register TXCNT contains the number of bytes of the transmit buffer, which have already been sent out on the bus.

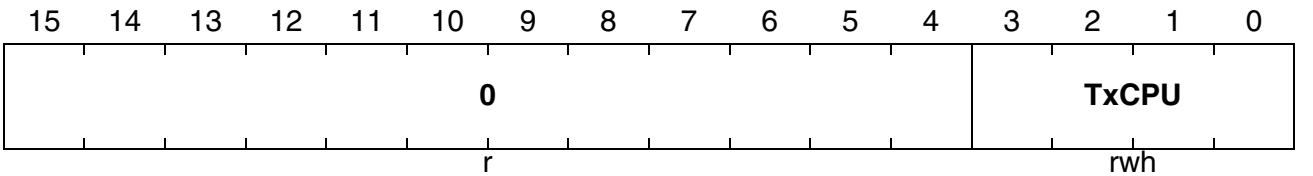
TXCNT
Bus Transmit Byte Counter Register **Reset Value: 0000_H**



Field	Bits	Type	Description
TxCNT	[3:0]	rh	**Bus Transmit Byte Counter** Bitfield TxCNT contains the number of bytes transmitted by the SDLM module. TxCNT is reset by resetting bit TxRQ. A transmit interrupt can be generated when TxRQ is reset by hardware. TxCPU equal to TxCNT and successful transmission (arbitration not lost) resets bit TxRQ by hardware in Normal Mode. = pointer for SDLM access to transmit buffer
0	[15:4]	–	**Reserved;** returns ‘0’ if read.

Register TXCPU contains the pointers to the next empty byte in the transmit buffer (= number of bytes in the transmit buffer).

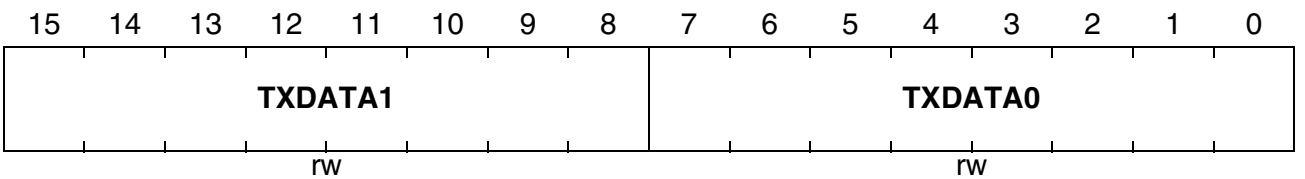
TXCPU
CPU Transmit Byte Counter Register **Reset Value: 0000_H**



Field	Bits	Type	Description
TxCPU	[3:0]	rwh	**CPU Transmit Byte Counter** Bitfield TxCPU contains the number of bytes, which have been written to the transmit buffer by the CPU. In FIFO mode (TxINCE = ‘1’ or BMEN = ‘1’), TXCPU is incremented by 1 after each CPU write action to register TXD0. In random mode only (TxINCE = 0 and BMEN = 0), TxCPU has to be written by software before setting the transmit request bit (TxRQ) in order to define the number of bytes to be sent. TxCPU is reset by resetting bit TxRQ in normal mode or resetting BMEN. For more details see TxCNT = pointer for CPU access to transmit buffer in FIFO mode.
0	[15:4]	–	**Reserved;** returns ‘0’ if read; should be written with ‘0’.

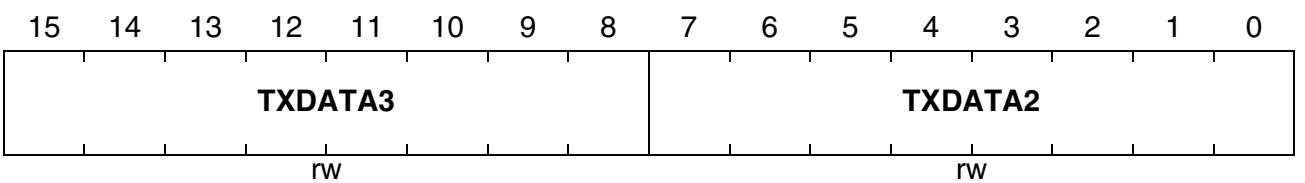
The transmit data registers contain the data bytes in the transmit buffer. In random mode mode, all data bytes can be directly accessed via their addresses, whereas in FIFO mode, only TXD0 should be used.

TXD0
Transmit Data Register 0 **Reset Value: 0000_H**



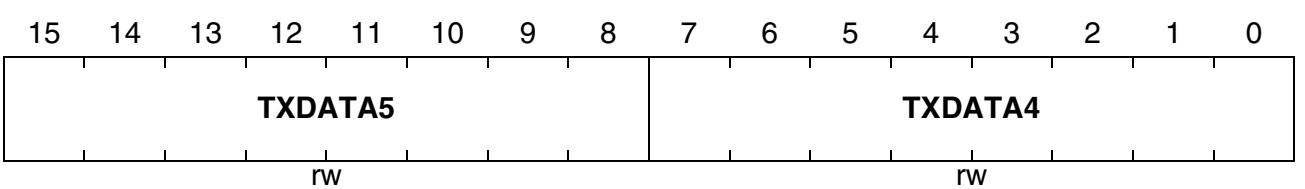
Field	Bits	Type	Description
TXDATA0	[7:0]	rw	**Transmit Buffer Data Byte 0**
TXDATA1	[15:8]	rw	**Transmit Buffer Data Byte 1**

TXD2
Transmit Data Register 2 **Reset Value: 0000_H**



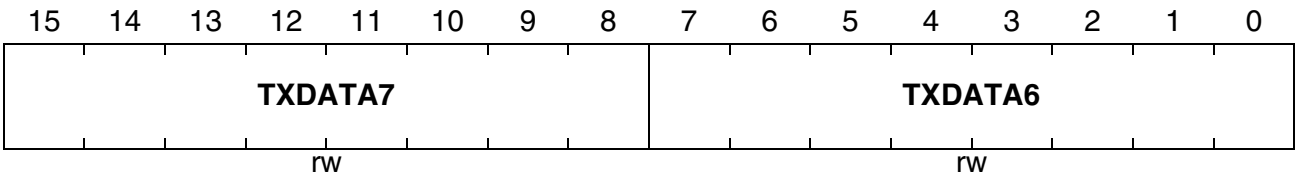
Field	Bits	Type	Description
TXDATA2	[7:0]	rw	**Transmit Buffer Data Byte 2**
TXDATA3	[15:8]	rw	**Transmit Buffer Data Byte 3**

TXD4
Transmit Data Register 4 **Reset Value: 0000_H**



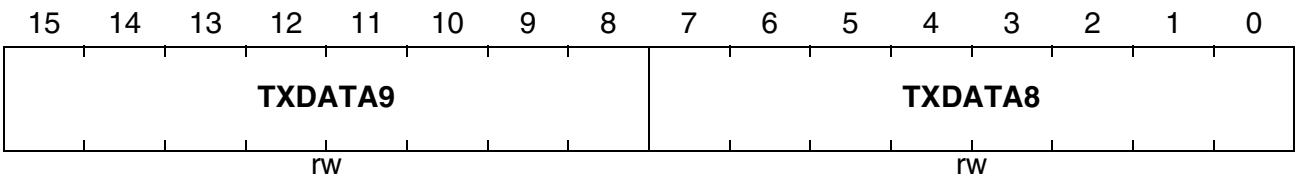
Field	Bits	Type	Description
TXDATA4	[7:0]	rw	**Transmit Buffer Data Byte 4**
TXDATA5	[15:8]	rw	**Transmit Buffer Data Byte 5**

TXD6
Transmit Data Register 6 **Reset Value: 0000_H**



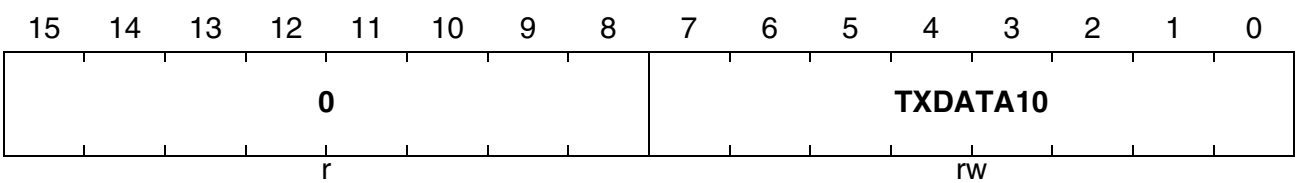
Field	Bits	Type	Description
TXDATA6	[7:0]	rw	**Transmit Buffer Data Byte 6**
TXDATA7	[15:8]	rw	**Transmit Buffer Data Byte 7**

TXD8
Transmit Data Register 8 **Reset Value: 0000_H**



Field	Bits	Type	Description
TXDATA8	[7:0]	rw	**Transmit Buffer Data Byte 8**
TXDATA9	[15:8]	rw	**Transmit Buffer Data Byte 9**

TXD10
Transmit Data Register 10 **Reset Value: 0000_H**

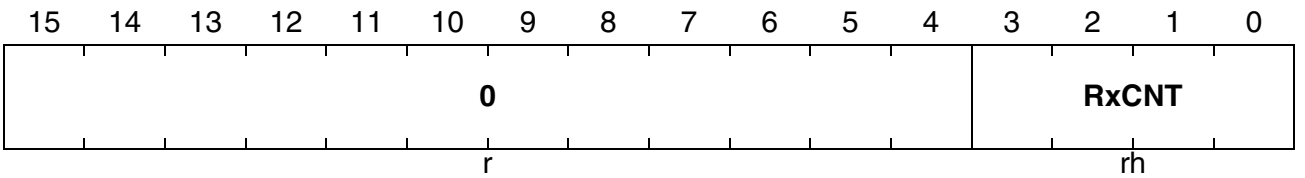


Field	Bits	Type	Description
TXDATA10	[7:0]	rw	**Transmit Buffer Data Byte 10**
0	[15:8]	–	**Reserved;** returns ‘0’ if read; should be written with ‘0’.

22.4.2 Reception Related Registers

Register RXCNT contains the number of bytes received in this buffer.

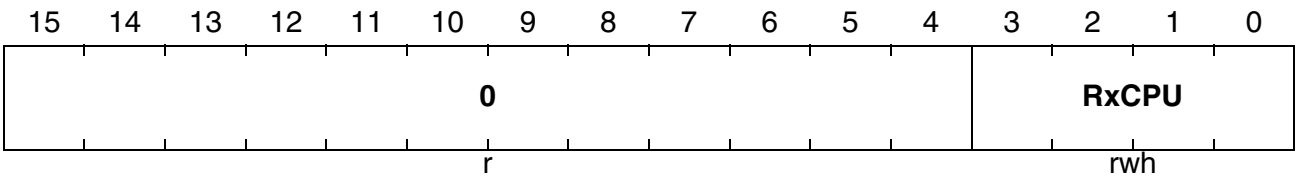
RXCNT
Bus Receive Byte Counter Register (on CPU side) **Reset Value: 0000_H**



Field	Bits	Type	Description
RxCNT	[3:0]	rh	**Receive Byte Counter** Bitfield RxCNT contains the number of received bytes in the receive buffer on CPU side. = pointer for SDLM access to receive buffer RxCNT is reset when the receive buffer on CPU side is released (see DONE).
0	[15:4]	–	**Reserved;** returns ‘0’ if read.

Register RXCPU contains the number of bytes already read out from this buffer.

RXCPU
CPU Receive Byte Counter Register (on CPU side) **Reset Value: 0000_H**

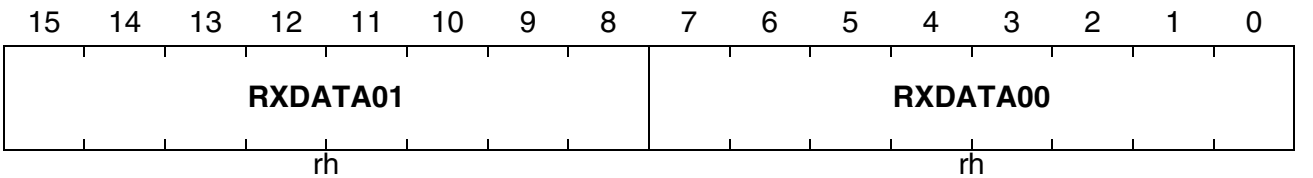


Field	Bits	Type	Description
RxCPU	[3:0]	rwh	**CPU Receive Byte Counter** Bitfield RxCPU contains the number of bytes read out by the CPU. In FIFO mode (RxINCE = ‘1’ or BMEN = ‘1’), RXCPU is incremented by 1 after each CPU read action to register RxD00. In random mode (RxINCE = 0 and BMEN = 0), RxCPU is not used and is 0. = pointer for CPU access to receive buffer RxCPU is reset when the receive buffer on CPU side is released.
0	[15:4]	–	**Reserved;** returns ‘0’ if read; should be written with ‘0’.

The receive data registers contain the data bytes in the receive buffer. In random mode mode, all data bytes can be directly accessed via their addresses, whereas in FIFO mode, only RXD00 should be used.

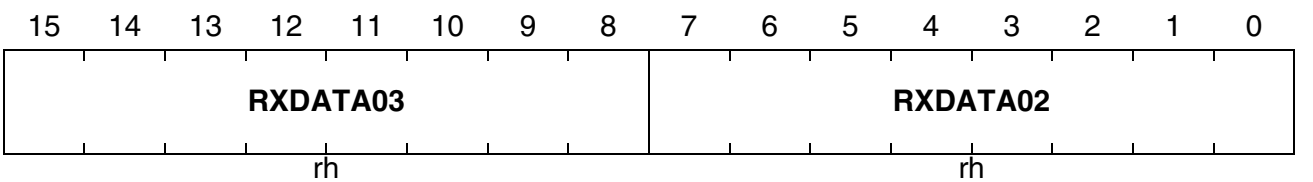
Bitfields RXDATA0x (x = 0 … 10) represent the receive buffer 0 on CPU side, bitfields RXDATA1x (x = 0 … 10) represent the receive buffer 1 on bus side. In block mode, the 16-byte receive buffer is built by bitfields RXDATA00-07 and bitfields RXDATA10-17.

RXD00
Receive Data Register 00 (on CPU side) **Reset Value: 0000_H**



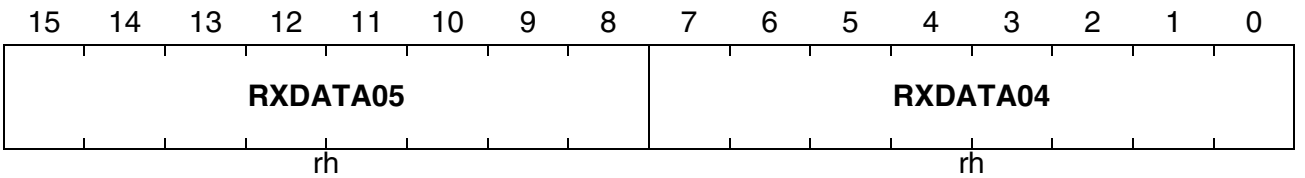
Field	Bits	Type	Description
RXDATA00	[7:0]	rw	**Receive Buffer 0 Data Byte 0**
RXDATA01	[15:8]	rw	**Receive Buffer 0 Data Byte 1**

RXD02
Receive Data Register 02 (on CPU side) **Reset Value: 0000_H**



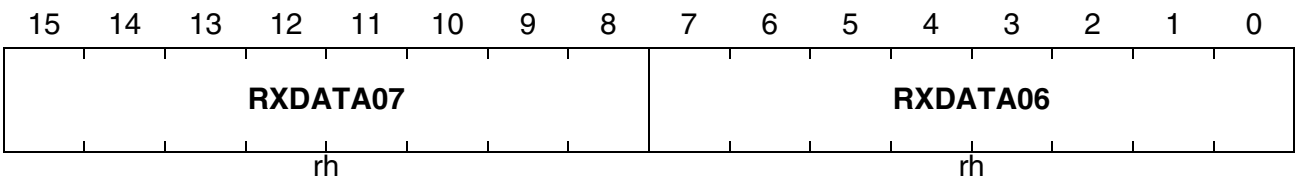
Field	Bits	Type	Description
RXDATA02	[7:0]	rh	**Receive Buffer 0 Data Byte 2**
RXDATA03	[15:8]	rh	**Receive Buffer 0 Data Byte 3**

RXD04
Receive Data Register 04 (on CPU side) **Reset Value: 0000_H**



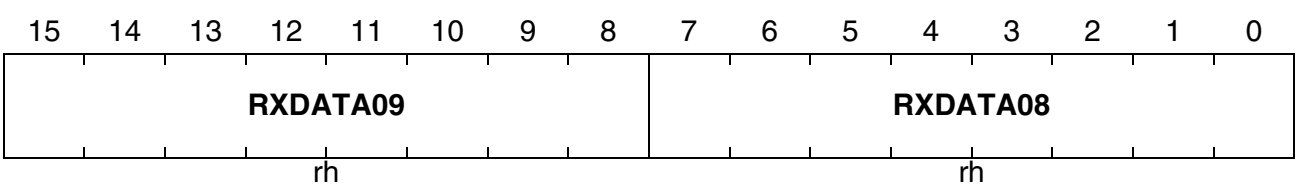
Field	Bits	Type	Description
RXDATA04	[7:0]	rh	Receive Buffer 0 Data Byte 4
RXDATA05	[15:8]	rh	Receive Buffer 0 Data Byte 5

RXD06
Receive Data Register 06 (on CPU side) **Reset Value: 0000_H**



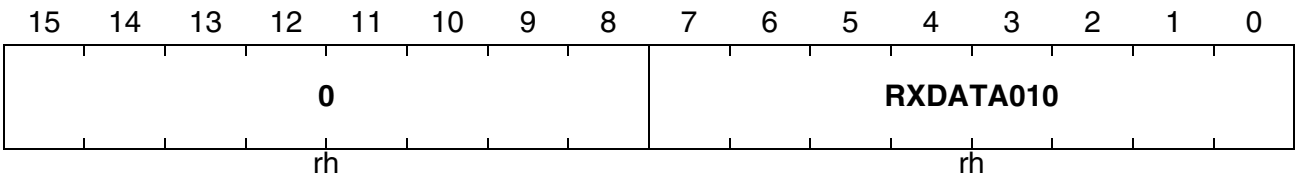
Field	Bits	Type	Description
RXDATA06	[7:0]	rh	Receive Buffer 0 Data Byte 6
RXDATA07	[15:8]	rh	Receive Buffer 0 Data Byte 7

RXD08
Receive Data Register 08 (on CPU side) **Reset Value: 0000_H**



Field	Bits	Type	Description
RXDATA08	[7:0]	rh	Receive Buffer 0 Data Byte 8
RXDATA09	[15:8]	rh	Receive Buffer 0 Data Byte 9

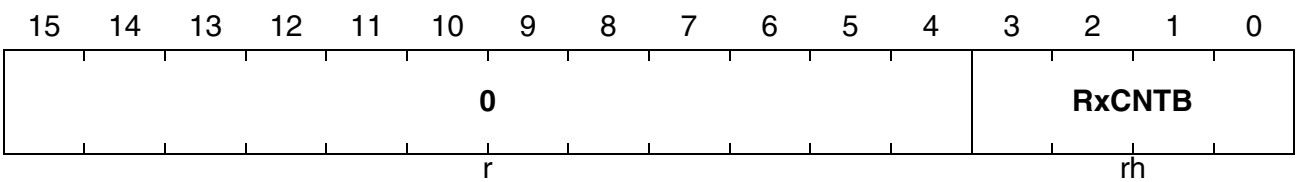
RXD010
Receive Data Register 010 (on CPU side) **Reset Value: 0000_H**



Field	Bits	Type	Description
RXDATA010	[7:0]	rh	**Receive Buffer 0 Data Byte 8**
0	[15:8]	–	**Reserved;** returns ‘0’ if read.

Register RXPTRB contains the bitfield indicating the number of received bytes in the receive buffer on bus side.

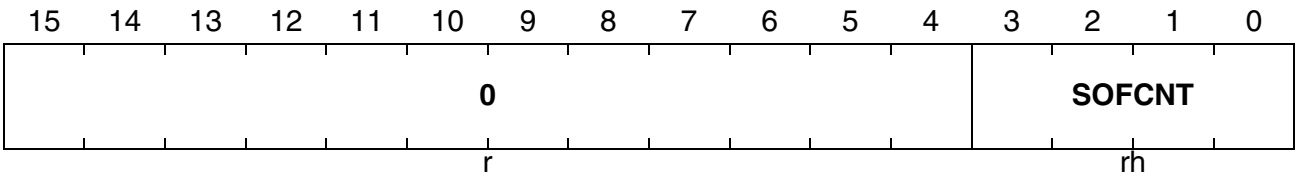
RXCNTB
Bus Receive Byte Counter Register (on bus side) **Reset Value: 0000_H**



Field	Bits	Type	Description
RxCNTB	[3:0]	rh	**Receive Byte Counter** Bitfield RxCNTB contains the number of received bytes in the receive buffer on bus side.
0	[15:4]	–	**Reserved;** returns ‘0’ if read.

Register SOFPTR contains the bitfield indicating the value of RXCNT after the last ENDF detection in block mode.

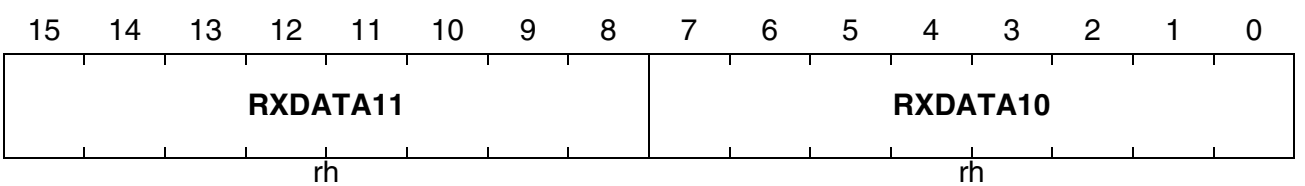
SOFPTR
Start-of-Frame Pointer Register **Reset Value: 0000_H**



Field	Bits	Type	Description
SOFCNT	[3:0]	rh	**Start-of-Frame Counter for Block Mode** The value of bitfield RxCNT is automatically copied to this bitfield if an end-of-frame symbol is detected. This feature can be used in block mode to determine the position of the first new byte of a frame in 16-byte receive buffer.
0	[15:4]	–	**Reserved;** returns ‘0’ if read.

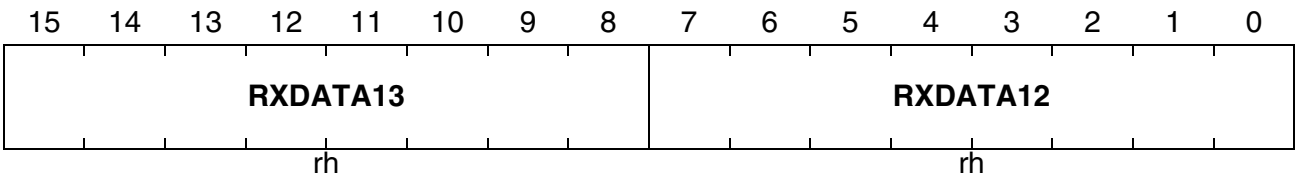
Receive Buffer on bus side:

RXD10
Receive Data Register 10 (on bus side) **Reset Value: 0000_H**



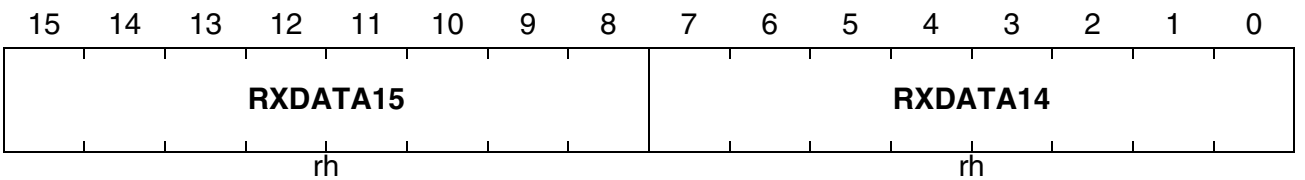
Field	Bits	Type	Description
RXDATA10	[7:0]	rh	**Receive Buffer 1 Data Byte 0**
RXDATA11	[15:8]	rh	**Receive Buffer 1 Data Byte 1**

RXD12
Receive Data Register 12 (on bus side) **Reset Value: 0000_H**



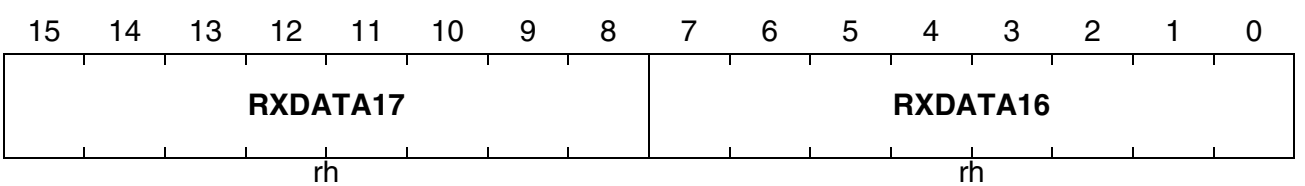
Field	Bits	Type	Description
RXDATA12	[7:0]	rh	Receive Buffer 1 Data Byte 2
RXDATA13	[15:8]	rh	Receive Buffer 1 Data Byte 3

RXD14
Receive Data Register 14 (on bus side) **Reset Value: 0000_H**



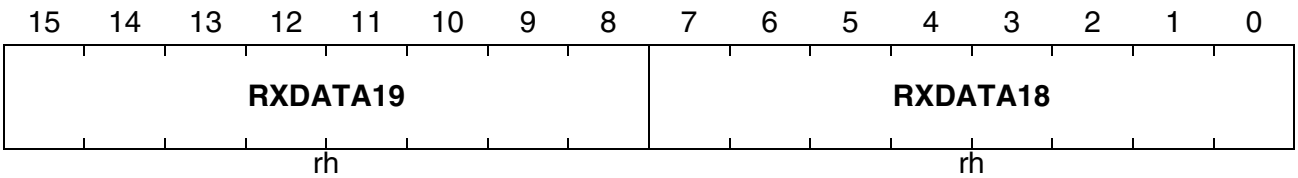
Field	Bits	Type	Description
RXDATA14	[7:0]	rh	Receive Buffer 1 Data Byte 4
RXDATA15	[15:8]	rh	Receive Buffer 1 Data Byte 5

RXD16
Receive Data Register 16 (on bus side) **Reset Value: 0000_H**



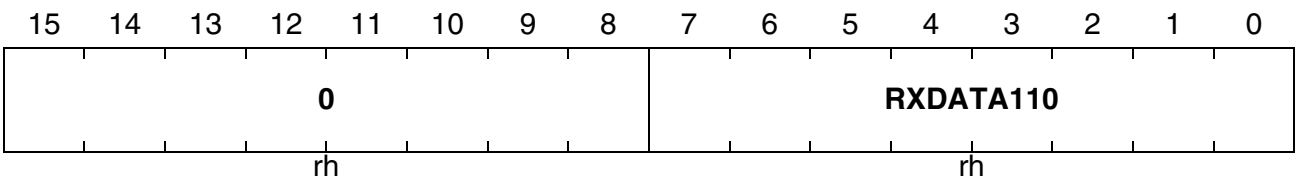
Field	Bits	Type	Description
RXDATA16	[7:0]	rh	Receive Buffer 1 Data Byte 6
RXDATA17	[15:8]	rh	Receive Buffer 1 Data Byte 7

RXD18
Receive Data Register 18 (on bus side) **Reset Value: 0000_H**



Field	Bits	Type	Description
RXDATA18	[7:0]	rh	**Receive Buffer 1 Data Byte 8**
RXDATA19	[15:8]	rh	**Receive Buffer 1 Data Byte 9**

RXD110
Receive Data Register 110 (on bus side) **Reset Value: 0000_H**



Field	Bits	Type	Description
RXDATA110	[7:0]	rh	**Receive Buffer 1 Data Byte 10**
0	[15:8]	–	**Reserved;** returns ‘0’ if read.

22.5 SDLM Module Register Table

Table 23-1 shows all register which are required for programming of the SDLM module. It summarizes the SDLM registers and defines the addresses and reset values.

22.6 XC161 Module Implementation Details

This section describes:

- the SDLM module related interfaces such as port connections and interrupt control
- all SDLM module related registers with its addresses and reset values

22.6.1 Interfaces of the SDLM Module

In XC161 the SDLM module is connected to Port pins according to **Figure 22-18**.

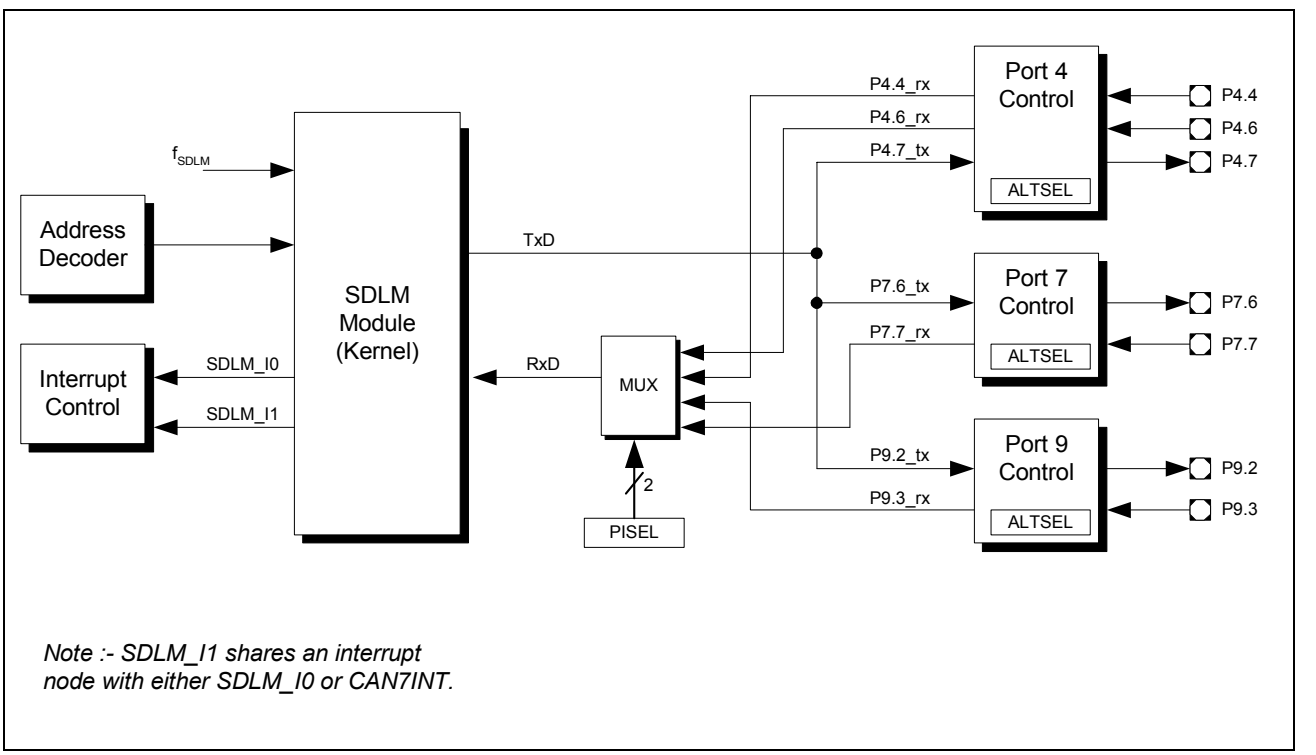


Figure 22-18 SDLM Module IO Interface

The input receive pins can be selected by bitfield RIS in the SDLM_PISEL register. The output transmit pins are defined by the corresponding ALTSEL registers of Port 4, Port 7 or Port 9.

The SDLM module kernel has two interrupt request lines. The interrupt request line SDLM_I1 can share a interrupt node with SDLM_I0, or alternatively with the TwinCAN interrupt request line 7. The selection is controlled via bitfield I1SEL in the SDLM_PISEL register. This is illustrated in **Figure 22-19**.

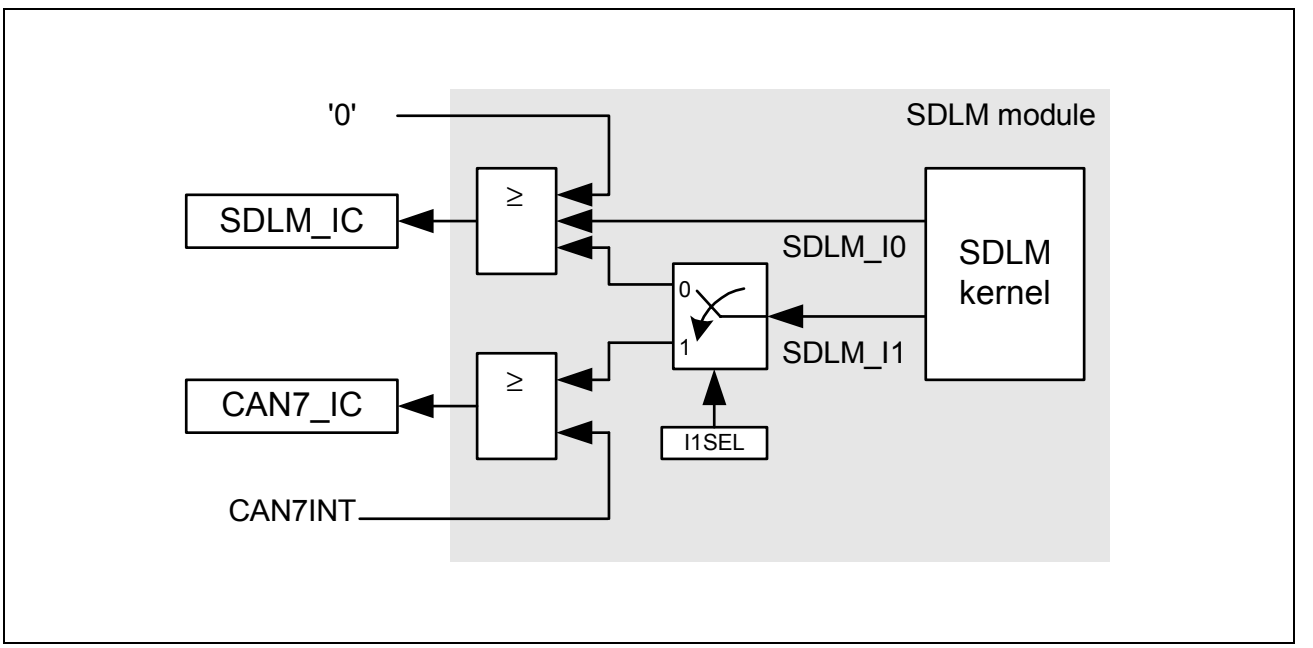


Figure 22-19 SDLM Interrupt Handling

22.6.2 SDLM Module Related External Registers

Figure 22-20 shows the module related external registers which are required for programming the SDLM module.

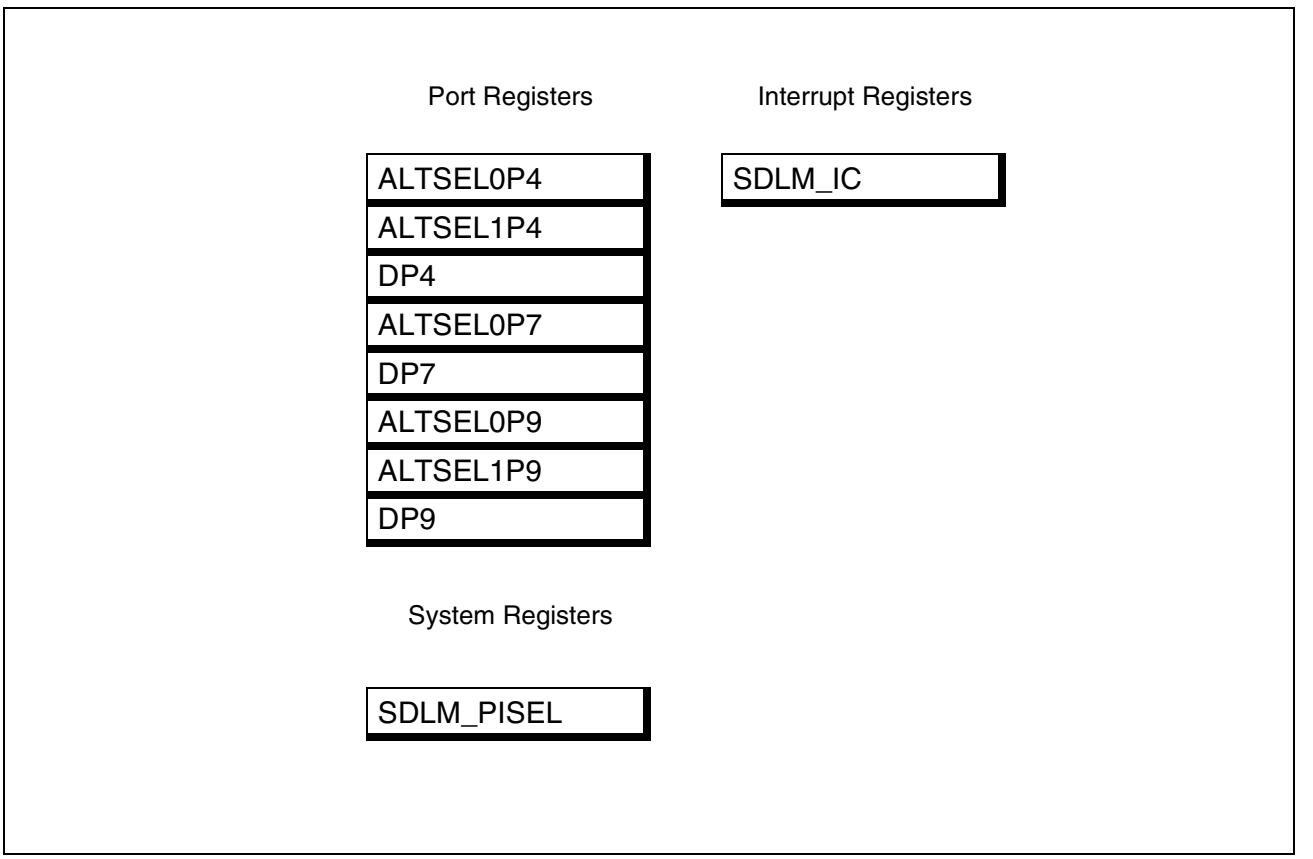
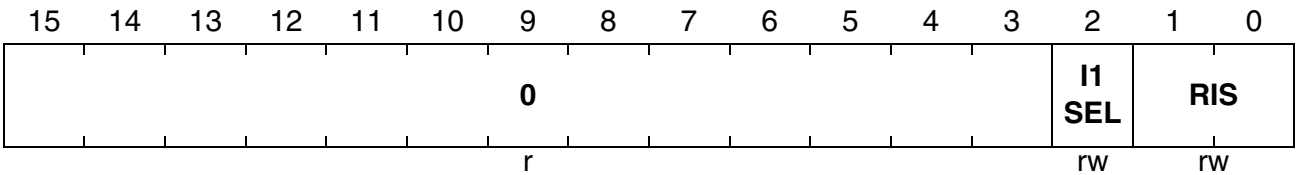


Figure 22-20 SDLM Implementation Specific Registers

22.6.2.1 System Registers

Register PISEL allows the user to select an input pin for the SDLM receive signal. Furthermore, the interrupt functionality of SDLM_I1 is defined.

SDLM_PISEL
SDLM Port Input Select Register **Reset Value: 0000_H**



Field	Bits	Type	Description
RIS	[1:0]	rw	**Receive Input Selection** Bitfield RIS defines the input pin for the SDLM receive line RxDJ. 00 The input pin for RxDJ is P4.6. 01 The input pin for RxDJ is P4.4. 10 The input pin for RxDJ is P9.3. 11 The input pin for RxDJ is P7.7.
I1SEL	2	rw	**Interrupt SDLM_I1 Selection** Bit I1SEL defines the interrupt functionality of the SDLM_I1 interrupt. 0 The interrupt signal SDLM_I1 is combined (logical OR) with the signal SDLM_I0. The combined signal sets the interrupt request flag in the SDLM interrupt control register. The interrupt node CAN_7 of the TwinCAN module is not influenced by the SDLM module. 1 The interrupt signal SDLM_I0 is the only source to trigger the interrupt request flag in the SDLM interrupt control register. The interrupt signal SDLM_I1 is combined (logical OR) with the interrupt signal CAN_7 delivered by the TwinCAN module. The combined signal sets the interrupt request flag of the interrupt node CAN_7.
0	[15:3]	r	**Reserved;** returns ‘0’ if read; should be written with ‘0’.

Note: In order to avoid mismatches if bit I1SEL = ‘1’, the interrupt node pointers of the TwinCAN module should not be programmed with the value ‘111_B’.

22.6.2.2 Port Registers

The interconnections between the SDLM module and the IO lines is controlled in the port logic of Port 4, Port 7 and Port 9. To configure the TxD output, the respective alternate select registers must be set accordingly. With this setting, the direction of the pin is also configured as output. The RxD input is connected via the ‘direct in’ to the module kernel, and it is selected via the PISEL register. The direction of this pin must be set to input via the Port Direction Control register DP4 or DP7 or DP9.

ALTSEL0P4
P4 Alternate Select Register 0 **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0								P7	P6	0					
r								rw	rw	r					

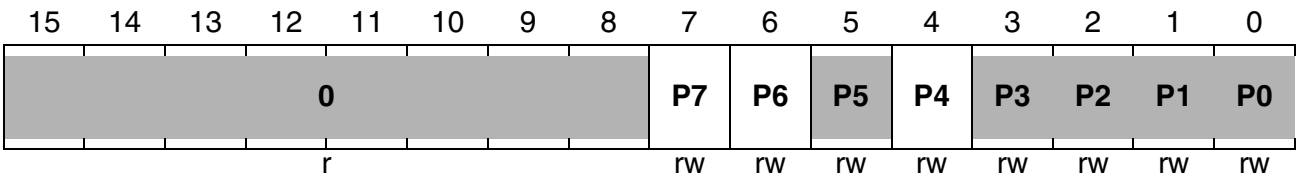
Field	Bit	Type	Description
ALTSEL0 P4.y	7	rw	**P4 Alternate Select Register 0 Bit y** 0 associated peripheral output is not selected as alternate function 1 associated peripheral output is selected as alternate function

ALTSEL1P4
P4 Alternate Select Register 1 **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0								P7	0						
r								rw	r						

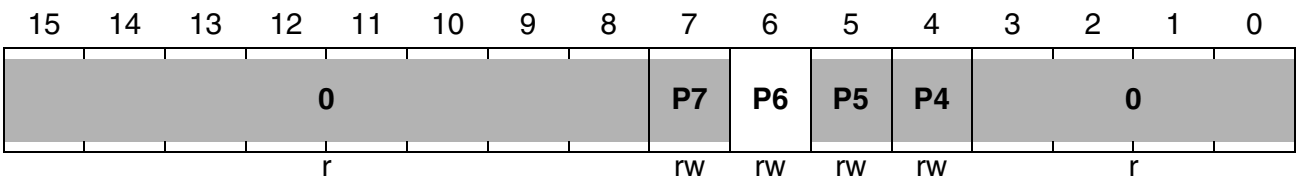
Field	Bit	Type	Description
ALTSEL1 P4.y	7	rw	**P4 Alternate Select Register 1 Bit y** 0 associated peripheral output is not selected as alternate function 1 associated peripheral output is selected as alternate function

DP4
P4 Direction Ctrl. Register **Reset Value: 0000_H**



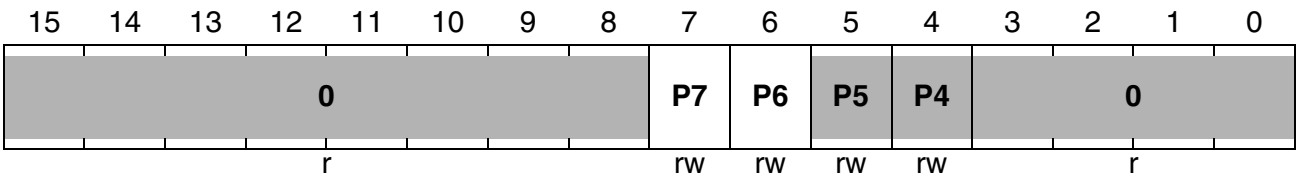
Field	Bit	Type	Description
DP4.y	7, 6, 4	rw	**Port Direction Register DP4 Bit y** 0 Port line P4.y is an input (high-impedance) 1 Port line P4.y is an output

ALTSEL0P7
P7 Alternate Select Register 0 **Reset Value: 0000_H**



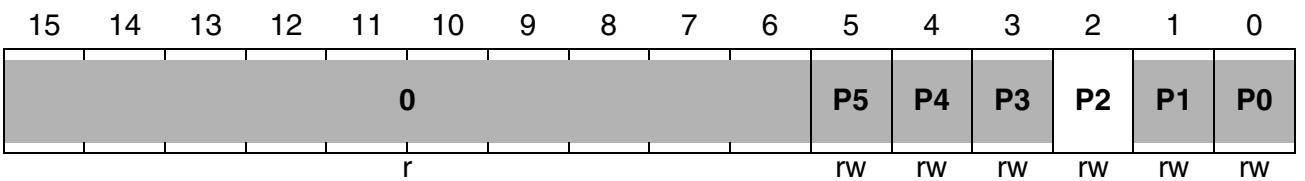
Field	Bit	Type	Description
ALTSEL0 P7.y	6	rw	**P7 Alternate Select Register 0 Bit y** 0 associated peripheral output is not selected as alternate function 1 associated peripheral output is selected as alternate function

DP7
P7 Direction Ctrl. Register **Reset Value: 0000_H**



Field	Bit	Type	Description
DP7.y	7, 6	rw	**Port Direction Register DP7 Bit y** 0 Port line P7.y is an input (high-impedance) 1 Port line P7.y is an output

ALTSEL0P9
P9 Alternate Select Register 0 **Reset Value: 0000_H**



Field	Bit	Type	Description
ALTSEL0 P9.y	2	rw	**P9 Alternate Select Register 0 Bit y** 0 associated peripheral output is not selected as alternate function 1 associated peripheral output is selected as alternate function

ALTSEL1P9
P9 Alternate Select Register 1 **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0										P5	P4	P3	P2	P1	P0
r										rw	rw	rw	rw	rw	rw

Field	Bit	Type	Description
ALTSEL1 P9.y	2	rw	**P9 Alternate Select Register 1 Bit y** 0 associated peripheral output is not selected as alternate function 1 associated peripheral output is selected as alternate function

DP9
P9 Direction Ctrl. Register **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0										P5	P4	P3	P2	P1	P0
r										rw	rw	rw	rw	rw	rw

Field	Bit	Type	Description
DP9.y	3, 2	rw	**Port Direction Register DP9 Bit y** 0 Port line P9.y is an input (high-impedance) 1 Port line P9.y is an output

Note: Shaded bits are not related to SDLM operation.

Table 22-3 shows the required register setting to configure the IO lines of the SDLM module for operation.

Table 22-3 SDLM IO Selection and Setup

Port Lines	Alternate Select Registers	Port Input Select Register	Direction Control Register	IO
P4.4 / RxDJ	–	SDLM_PISEL[1:0] = 01	DP4.P4 = 0	Input
P4.6 / RxDJ	–	SDLM_PISEL[1:0] = 00	DP4.P6 = 0	Input
P4.7 / TxDJ	ALTSEL0P4.P7 = 0 and ALTSEL1P4.P7 = 1	–	DP4.P7 = 1	Output
P7.6 / TxDJ	ALTSEL0P7.P6 = 1	–	DP7.P6 = 1	Output
P7.7 / RxDJ	–	SDLM_PISEL[1:0] = 11	DP7.P7 = 0	Input
P9.2 / TxDJ	ALTSEL0P9.P2 = 1 and ALTSEL1P9.P2 = 1	–	DP9.P2 = 1	Output
P9.3 / RxDJ	–	SDLM_PISEL[1:0] = 10	DP9.P3 = 0	Input

22.6.2.3 Interrupt Registers

The interrupt of the SDLM module is controlled by interrupt control register SDLM_IC.

SDLM_IC
SDLM Interrupt Ctrl. Reg. **ESFR ($F19A_H/CD_H$)** **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-							GPX	SDL IR	SDL IE	ILVL				GLVL	
-	-	-	-	-	-	-	rw	rwh	rw	rw				rw	

Note: Please refer to the general Interrupt Control Register description for an explanation of the control fields.

23 Register Set

This chapter summarizes all the kernel and module related external registers of the peripherals. The register list is organized into two parts - the first for PD+BUS peripherals and the second for LXBUS peripherals.

23.1 PD+BUS Peripherals

Note: The address space for PD+BUS peripherals is assigned to Segment 0.

Table 23-1 PD+BUS Register Listing

Short Name	Address			Description	Reset Value
	Physical	8-bit	Area		
Asynchronous/Synchronous Serial Interface 0 (ASC0)					
ASC0_CON	$FFB0_H$	$D8_H$	SFR	ASC0 Control Register	0000_H
ASC0_TBUF	$FEB0_H$	58_H	SFR	ASC0 Transmit Buffer Register	0000_H
ASC0_RBUF	$FEB2_H$	59_H	SFR	ASC0 Receive Buffer Register	0000_H
ASC0_ ABCON	$F1B8_H$	DC_H	ESFR	ASC0 Autobaud Control Register	0000_H
ASC0_ ABSTAT	$F0B8_H$	$5C_H$	ESFR	ASC0 Autobaud Status Register	0000_H
ASC0_BG	$FEB4_H$	$5A_H$	SFR	ASC0 Baud Rate Generator Reload Register	0000_H
ASC0_FDV	$FEB6_H$	$5B_H$	SFR	ASC0 Fractional Divider Register	0000_H
ASC0_PMW	$FEAA_H$	55_H	SFR	ASC0 IrDA Pulse Mode and Width Reg.	0000_H
ASC0_ RXFCON	$F0C6_H$	63_H	ESFR	ASC0 Receive FIFO Control Register	0000_H
ASC0_ TXFCON	$F0C4_H$	62_H	ESFR	ASC0 Transmit FIFO Control Register	0000_H
ASC0_FSTAT	$F0BA_H$	$5D_H$	ESFR	ASC0 FIFO Status Register	0000_H
Asynchronous/Synchronous Serial Interface 1 (ASC1)					
ASC1_CON	$FFB8_H$	DC_H	SFR	ASC1 Control Register	0000_H
ASC1_TBUF	$FEB8_H$	$5C_H$	SFR	ASC1 Transmit Buffer Register	0000_H
ASC1_RBUF	$FEBA_H$	$5D_H$	SFR	ASC1 Receive Buffer Register	0000_H
ASC1_ ABCON	$F1BC_H$	DE_H	ESFR	ASC1 Autobaud Control Register	0000_H

Table 23-1 PD+BUS Register Listing (cont'd)

Short Name	Address			Description	Reset Value
	Physical	8-bit	Area		
ASC1_ ABSTAT	$F0BC_H$	$5E_H$	ESFR	ASC1 Autobaud Status Register	0000_H
ASC1_BG	$FEBC_H$	$5E_H$	SFR	ASC1 Baud Rate Generator Reload Register	0000_H
ASC1_FDV	$FEBE_H$	$5F_H$	SFR	ASC1 Fractional Divider Register	0000_H
ASC1_PMW	$FEAC_H$	56_H	SFR	ASC1 IrDA Pulse Mode and Width Reg.	0000_H
ASC1_ RXFCON	$F0A6_H$	53_H	ESFR	ASC1 Receive FIFO Control Register	0000_H
ASC1_ TXFCON	$F0A4_H$	52_H	ESFR	ASC1 Transmit FIFO Control Register	0000_H
ASC1_FSTAT	$F0BE_H$	$5F_H$	ESFR	ASC1 FIFO Status Register	0000_H
Synchronous Serial Channel 0 (SSC0)					
SSC0_CON	$FFB2_H$	$D9_H$	SFR	SSC0 Control Register	0000_H
SSC0_BR	$F0B4_H$	$5A_H$	ESFR	SSC0 Baudrate Timer Reload Register	0000_H
SSC0_TB	$F0B0_H$	58_H	ESFR	SSC0 Transmit Buffer Reg.	0000_H
SSC0_RB	$F0B2_H$	59_H	ESFR	SSC0 Receive Buffer Reg.	0000_H
Synchronous Serial Channel 1 (SSC1)					
SSC1_CON	$FF5E_H$	AF_H	SFR	SSC1 Control Register	0000_H
SSC1_BR	$F05E_H$	$2F_H$	ESFR	SSC1 Baudrate Timer Reload Register	0000_H
SSC1_TB	$F05A_H$	$2D_H$	ESFR	SSC1 Transmit Buffer Reg.	0000_H
SSC1_RB	$F05C_H$	$2E_H$	ESFR	SSC1 Receive Buffer Reg.	0000_H
General Purpose Timer Unit (GPT12E)					
GPT12E_ T2CON	$FF40_H$	$A0_H$	SFR	GPT12E Timer 2 Control Register	0000_H
GPT12E_ T3CON	$FF42_H$	$A1_H$	SFR	GPT12E Timer 3 Control Register	0000_H
GPT12E_ T4CON	$FF44_H$	$A2_H$	SFR	GPT12E Timer 4 Control Register	0000_H

Table 23-1 PD+BUS Register Listing (cont'd)

Short Name	Address			Description	Reset Value
	Physical	8-bit	Area		
GPT12E_ T5CON	$FF46_H$	$A3_H$	SFR	GPT12E Timer 5 Control Register	0000_H
GPT12E_ T6CON	$FF48_H$	$A4_H$	SFR	GPT12E Timer 6 Control Register	0000_H
GPT12E_ CAPREL	$FE4A_H$	25_H	SFR	GPT12E Capture/Reload Register	0000_H
GPT12E_T2	$FE40_H$	20_H	SFR	GPT12E Timer 2 Register	0000_H
GPT12E_T3	$FE42_H$	21_H	SFR	GPT12E Timer 3 Register	0000_H
GPT12E_T4	$FE44_H$	22_H	SFR	GPT12E Timer 4 Register	0000_H
GPT12E_T5	$FE46_H$	23_H	SFR	GPT12E Timer 5 Register	0000_H
GPT12E_T6	$FE48_H$	24_H	SFR	GPT12E Timer 6 Register	0000_H
Real Time Clock (RTC)					
RTC_CON	$F110_H$	88_H	ESFR	RTC Control Register, low word	8003_H
RTC_T14	$F0D2_H$	69_H	ESFR	Timer 14 Register	$UUUU_H$
RTC_T14REL	$F0D0_H$	68_H	ESFR	Timer 14 Reload Register	$UUUU_H$
RTC_RTCL	$F0D4_H$	$6A_H$	ESFR	RTC Timer Low Register	$UUUU_H$
RTC_RTCH	$F0D6_H$	$6B_H$	ESFR	RTC Timer High Register	$UUUU_H$
RTC_RELL	$F0CC_H$	66_H	ESFR	RTC Reload Low Register	0000_H
RTC_RELH	$F0CE_H$	67_H	ESFR	RTC Reload High Register	0000_H
RTC_ISNC	$F10C_H$	86_H	ESFR	RTC Interrupt Subnode Register	0000_H
Capture/Compare Unit 1 (CAPCOM1)					
CC1_M0	$FF52_H$	$A9_H$	SFR	CAPCOM 1 Mode Control Register 0	0000_H
CC1_M1	$FF54_H$	AA_H	SFR	CAPCOM 1 Mode Control Register 1	0000_H
CC1_M2	$FF56_H$	AB_H	SFR	CAPCOM 1 Mode Control Register 2	0000_H
CC1_M3	$FF58_H$	AC_H	SFR	CAPCOM 1 Mode Control Register 3	0000_H
CC1_SEE	$FE2E_H$	17_H	SFR	CAPCOM1 Single Event Enable Register	0000_H

Table 23-1 PD+BUS Register Listing (cont'd)

Short Name	Address			Description	Reset Value
	Physical	8-bit	Area		
CC1_SEM	$FE2C_H$	16_H	SFR	CAPCOM1 Single Event Mode Register	0000_H
CC1_DRM	$FF5A_H$	AD_H	SFR	CAPCOM1 Double Register Mode Register	0000_H
CC1_OUT	$FF5C_H$	AE_H	SFR	CAPCOM1 Output Register	0000_H
CC1_T0	$FE50_H$	28_H	SFR	CAPCOM 1 Timer 0 Register	0000_H
CC1_T0REL	$FE54_H$	$2A_H$	SFR	CAPCOM 1 Timer 0 Reload Register	0000_H
CC1_T1	$FE52_H$	29_H	SFR	CAPCOM 1 Timer 1 Register	0000_H
CC1_T1REL	$FE56_H$	$2B_H$	SFR	CAPCOM 1 Timer 1 Reload Register	0000_H
CC1_T01CON	$FF50_H$	$A8_H$	SFR	CAPCOM 1 Timer 0 and Timer 1 Control Register	0000_H
CC1_IOC	$F062_H$	31_H	ESFR	CAPCOM 1 I/O Control Register	0000_H
CC1_CC0	$FE80_H$	40_H	SFR	CAPCOM 1 Register 0	0000_H
CC1_CC1	$FE82_H$	41_H	SFR	CAPCOM 1 Register 1	0000_H
CC1_CC2	$FE84_H$	42_H	SFR	CAPCOM 1 Register 2	0000_H
CC1_CC3	$FE86_H$	43_H	SFR	CAPCOM 1 Register 3	0000_H
CC1_CC4	$FE88_H$	44_H	SFR	CAPCOM 1 Register 4	0000_H
CC1_CC5	$FE8A_H$	45_H	SFR	CAPCOM 1 Register 5	0000_H
CC1_CC6	$FE8C_H$	46_H	SFR	CAPCOM 1 Register 6	0000_H
CC1_CC7	$FE8E_H$	47_H	SFR	CAPCOM 1 Register 7	0000_H
CC1_CC8	$FE90_H$	48_H	SFR	CAPCOM 1 Register 8	0000_H
CC1_CC9	$FE92_H$	49_H	SFR	CAPCOM 1 Register 9	0000_H
CC1_CC10	$FE94_H$	$4A_H$	SFR	CAPCOM 1 Register 10	0000_H
CC1_CC11	$FE96_H$	$4B_H$	SFR	CAPCOM 1 Register 11	0000_H
CC1_CC12	$FE98_H$	$4C_H$	SFR	CAPCOM 1 Register 12	0000_H
CC1_CC13	$FE9A_H$	$4D_H$	SFR	CAPCOM 1 Register 13	0000_H
CC1_CC14	$FE9C_H$	$4E_H$	SFR	CAPCOM 1 Register 14	0000_H
CC1_CC15	$FE9E_H$	$4F_H$	SFR	CAPCOM 1 Register 15	0000_H

Table 23-1 PD+BUS Register Listing (cont'd)

Short Name	Address			Description	Reset Value
	Physical	8-bit	Area		
Capture / Compare Unit 2 (CAPCOM2)					
CC2_M4	$FF22_H$	91_H	SFR	CAPCOM2 Mode Control Register 4	0000_H
CC2_M5	$FF24_H$	92_H	SFR	CAPCOM2 Mode Control Register 5	0000_H
CC2_M6	$FF26_H$	93_H	SFR	CAPCOM2 Mode Control Register 6	0000_H
CC2_M7	$FF28_H$	94_H	SFR	CAPCOM2 Mode Control Register 7	0000_H
CC2_SEE	$FE2A_H$	15_H	SFR	CAPCOM2 Single Event Enable Register	0000_H
CC2_SEM	$FE28_H$	14_H	SFR	CAPCOM2 Single Event Mode Register	0000_H
CC2_DRM	$FF2A_H$	95_H	SFR	CAPCOM2 Double Register Mode Register	0000_H
CC2_OUT	$FF2C_H$	96_H	SFR	CAPCOM2 Output Register	0000_H
CC2_T7	$F050_H$	28_H	ESFR	CAPCOM 2 Timer 7 Register	0000_H
CC2_T8	$F052_H$	29_H	ESFR	CAPCOM 2 Timer 8 Register	0000_H
CC2_T7REL	$F054_H$	$2A_H$	ESFR	CAPCOM 2 Timer 7 Reload Register	0000_H
CC2_T8REL	$F056_H$	$2B_H$	ESFR	CAPCOM 2 Timer 8 Reload Register	0000_H
CC2_T78CON	$FF20_H$	90_H	SFR	CAPCOM 2 Timer 7 and Timer 8 Control Register	0000_H
CC2_IOC	$F066_H$	33_H	ESFR	CAPCOM 2 I/O Control Register	0000_H
CC2_CC16	$FE60_H$	30_H	SFR	CAPCOM 2 Register 16	0000_H
CC2_CC17	$FE62_H$	31_H	SFR	CAPCOM 2 Register 17	0000_H
CC2_CC18	$FE64_H$	32_H	SFR	CAPCOM 2 Register 18	0000_H
CC2_CC19	$FE66_H$	33_H	SFR	CAPCOM 2 Register 19	0000_H
CC2_CC20	$FE68_H$	34_H	SFR	CAPCOM 2 Register 20	0000_H
CC2_CC21	$FE6A_H$	35_H	SFR	CAPCOM 2 Register 21	0000_H
CC2_CC22	$FE6C_H$	36_H	SFR	CAPCOM 2 Register 22	0000_H

Table 23-1 PD+BUS Register Listing (cont'd)

Short Name	Address			Description	Reset Value
	Physical	8-bit	Area		
CC2_CC23	$FE6E_H$	37_H	SFR	CAPCOM 2 Register 23	0000_H
CC2_CC24	$FE70_H$	38_H	SFR	CAPCOM 2 Register 24	0000_H
CC2_CC25	$FE72_H$	39_H	SFR	CAPCOM 2 Register 25	0000_H
CC2_CC26	$FE74_H$	$3A_H$	SFR	CAPCOM 2 Register 26	0000_H
CC2_CC27	$FE76_H$	$3B_H$	SFR	CAPCOM 2 Register 27	0000_H
CC2_CC28	$FE78_H$	$3C_H$	SFR	CAPCOM 2 Register 28	0000_H
CC2_CC29	$FE7A_H$	$3D_H$	SFR	CAPCOM 2 Register 29	0000_H
CC2_CC30	$FE7C_H$	$3E_H$	SFR	CAPCOM 2 Register 30	0000_H
CC2_CC31	$FE7E_H$	$3F_H$	SFR	CAPCOM 2 Register 31	0000_H
A/D Converter (ADC)					
ADC_CON	$FFA0_H$	$D0_H$	SFR	A/D Converter Control Register	0000_H
ADC_CON1	$FFA6_H$	$D3_H$	SFR	A/D Converter Control Register	0000_H
ADC_CTR0	$FFBE_H$	DF_H	SFR	A/D Converter Control Register 0	1000_H
ADC_CTR2	$F09C_H$	$4E_H$	ESFR	A/D Converter Control Register 2	0000_H
ADC_CTR2IN	$F09E_H$	$4F_H$	ESFR	A/D Converter Injection Control Register 2	0000_H
ADC_DAT	$FEA0_H$	50_H	SFR	A/D Converter Result Register	0000_H
ADC_DAT2	$F0A0_H$	50_H	ESFR	A/D Converter 2 Result Register	0000_H
IIC Module					
IIC_ST	$E604_H$	–	IO	IIC Status Register	0000_H
IIC_CON	$E602_H$	–	IO	IIC Control Register	0000_H
IIC_CFG	$E600_H$	–	IO	IIC Configuration Register	0000_H
IIC_ADR	$E606_H$	–	IO	IIC Address Register	0000_H
IIC_RTBL	$E608_H$	–	IO	IIC Receive/Transmit Buffer Low Register	0000_H
IIC_RTBH	$E60A_H$	–	IO	IIC Receive/Transmit Buffer High Register	0000_H
SDLM Module (J1850)					
SDLM_PISEL	$E904_H$	–	IO	SDLM Port Input Select Register	0000_H

Table 23-1 PD+BUS Register Listing (cont'd)

Short Name	Address			Description	Reset Value
	Physical	8-bit	Area		
SDLM_ GLOBCON	$E910_H$	–	IO	Global Control Register	0000_H
SDLM_ CLKDIV	$E914_H$	–	IO	Clock Divider Register	0000_H
SDLM_ TxDELAY	$E916_H$	–	IO	Transceiver Delay Register	0014_H
SDLM_IFR	$E918_H$	–	IO	In-Frame Response Value Register	0000_H
SDLM_ BUFFSTAT	$E91C_H$	–	IO	Buffer Status Register	0000_H
SDLM_ TRANSSTAT	$E91E_H$	–	IO	Transmission Status Register	0000_H
SDLM_ BUSSTAT	$E920_H$	–	IO	Bus Status Register	0000_H
SDLM_ ERRSTAT	$E922_H$	–	IO	Error Status	0000_H
SDLM_ BUFFCON	$E924_H$	–	IO	Buffer Control Register	0000_H
SDLM_ FLAGRST	$E928_H$	–	IO	Flag Reset Register	0000_H
SDLM_ INTCON	$E92C_H$	–	IO	Interrupt Control Register	0000_H
SDLM_ TXCNT	$E93C_H$	–	IO	Bus Transmit Byte Counter Register	0000_H
SDLM_ TXCPU	$E93E_H$	–	IO	CPU Transmit Byte Counter Register	0000_H
SDLM_TXD0	$E930_H$	–	IO	Transmit Data Register 0	0000_H
SDLM_TXD2	$E932_H$	–	IO	Transmit Data Register 2	0000_H
SDLM_TXD4	$E934_H$	–	IO	Transmit Data Register 4	0000_H
SDLM_TXD6	$E936_H$	–	IO	Transmit Data Register 6	0000_H
SDLM_TXD8	$E938_H$	–	IO	Transmit Data Register 8	0000_H
SDLM_TXD10	$E93A_H$	–	IO	Transmit Data Register 10	0000_H

Table 23-1 PD+BUS Register Listing (cont'd)

Short Name	Address			Description	Reset Value
	Physical	8-bit	Area		
SDLM_ RXCNT	$E94C_H$	–	IO	Bus Receive Byte Counter (CPU)	0000_H
SDLM_ RXCPU	$E94E_H$	–	IO	CPU Receive Byte Counter (CPU)	0000_H
SDLM_ RXCNTB	$E95C_H$	–	IO	Bus Receive Byte Counter (Bus)	0000_H
SDLM_ SOFPTR	$E960_H$	–	IO	Start-of-Frame Pointer Register	0000_H
SDLM_ RXD00	$E940_H$	–	IO	Receive Data Register 00	0000_H
SDLM_ RXD02	$E942_H$	–	IO	Receive Data Register 02	0000_H
SDLM_ RXD04	$E944_H$	–	IO	Receive Data Register 04	0000_H
SDLM_ RXD06	$E946_H$	–	IO	Receive Data Register 06	0000_H
SDLM_ RXD08	$E948_H$	–	IO	Receive Data Register 08	0000_H
SDLM_ RXD010	$E94A_H$	–	IO	Receive Data Register 010	0000_H
SDLM_ RXD10	$E950_H$	–	IO	Receive Data Register 10 (bus)	0000_H
SDLM_ RXD12	$E952_H$	–	IO	Receive Data Register 12 (bus)	0000_H
SDLM_ RXD14	$E954_H$	–	IO	Receive Data Register 14 (bus)	0000_H
SDLM_ RXD16	$E956_H$	–	IO	Receive Data Register 16 (bus)	0000_H
SDLM_ RXD18	$E958_H$	–	IO	Receive Data Register 18 (bus)	0000_H
SDLM_ RXD110	$E95A_H$	–	IO	Receive Data Register 110 (bus)	0000_H

Table 23-1 PD+BUS Register Listing (cont'd)

Short Name	Address			Description	Reset Value
	Physical	8-bit	Area		
Interrupt Control					
SSC0_TIC	$FF72_H$	$B9_H$	SFR	SSC0 Transmit Interrupt Control Register	0000_H
SSC0_RIC	$FF74_H$	BA_H	SFR	SSC0 Receive Interrupt Control Register	0000_H
SSC0_EIC	$FF76_H$	BB_H	SFR	SSC0 Error Interrupt Control Register	0000_H
SSC1_TIC	$F1AA_H$	$D5_H$	ESFR	SSC1 Transmit Interrupt Control Register	0000_H
SSC1_RIC	$F1AC_H$	$D6_H$	SFR	SSC1 Receive Interrupt Control Register	0000_H
SSC1_EIC	$F1AE_H$	$D7_H$	ESFR	SSC1 Error Interrupt Control Register	0000_H
ASC0_TIC	$FF6C_H$	$B6_H$	SFR	ASC0 Transmit Interrupt Control Register	0000_H
ASC0_RIC	$FF6E_H$	$B7_H$	SFR	ASC0 Receive Interrupt Control Register	0000_H
ASC0_EIC	$FF70_H$	$B8_H$	SFR	ASC0 Error Interrupt Control Register	0000_H
ASC0_TBIC	$F19C_H$	CE_H	ESFR	ASC0 Transmit Buffer Interrupt Control Register	0000_H
ASC0_ABIC	$F15C_H$	AE_H	ESFR	ASC0 Autobaud Interrupt Control Register	0000_H
ASC1_TIC	$F182_H$	$C1_H$	ESFR	ASC1 Transmit Interrupt Control Register	0000_H
ASC1_RIC	$F18A_H$	$C5_H$	ESFR	ASC1 Receive Interrupt Control Register	0000_H
ASC1_EIC	$F192_H$	$C9_H$	ESFR	ASC1 Error Interrupt Control Register	0000_H
ASC1_TBIC	$F150_H$	$A8_H$	ESFR	ASC1 Transmit Buffer Interrupt Control Register	0000_H
ASC1_ABIC	$F1BA_H$	DD_H	ESFR	ASC1 Autobaud Interrupt Control Register	0000_H

Table 23-1 PD+BUS Register Listing (cont'd)

Short Name	Address			Description	Reset Value
	Physical	8-bit	Area		
GPT12E_T2IC	$FF60_H$	$B0_H$	SFR	GPT12E Timer 2 Interrupt Control Register	0000_H
GPT12E_T3IC	$FF62_H$	$B1_H$	SFR	GPT12E Timer 3 Interrupt Control Register	0000_H
GPT12E_T4IC	$FF64_H$	$B2_H$	SFR	GPT12E Timer 4 Interrupt Control Register	0000_H
GPT12E_T5IC	$FF66_H$	$B3_H$	SFR	GPT12E Timer 5 Interrupt Control Register	0000_H
GPT12E_T6IC	$FF68_H$	$B4_H$	SFR	GPT12E Timer 6 Interrupt Control Register	0000_H
GPT12E_CRIC	$FF6A_H$	$B5_H$	SFR	GPT12E CAPREL Interrupt Control Register	0000_H
PLL_IC	$F19E_H$	CF_H	ESFR	PLL Interrupt Control Register	0000_H
RTC_IC	$F1A0_H$	$D0_H$	ESFR	RTC Interrupt Control Register	0000_H
CC1_T0IC	$FF9C_H$	CE_H	SFR	CAPCOM Timer 0 Interrupt Control Register	0000_H
CC1_T1IC	$FF9E_H$	CF_H	SFR	CAPCOM Timer 1 Interrupt Control Register	0000_H
CC2_T7IC	$F17A_H$	BD_H	ESFR	CAPCOM Timer 7 Interrupt Control Register	0000_H
CC2_T8IC	$F17C_H$	BE_H	ESFR	CAPCOM Timer 8 Interrupt Control Register	0000_H
CC1_CC0IC	$FF78_H$	BC_H	SFR	CAPCOM Register 0 Interrupt Control Register	0000_H
CC1_CC1IC	$FF7A_H$	BD_H	SFR	CAPCOM Register 1 Interrupt Control Register	0000_H
CC1_CC2IC	$FF7C_H$	BE_H	SFR	CAPCOM Register 2 Interrupt Control Register	0000_H
CC1_CC3IC	$FF7E_H$	BF_H	SFR	CAPCOM Register 3 Interrupt Control Register	0000_H
CC1_CC4IC	$FF80_H$	$C0_H$	SFR	CAPCOM Register 4 Interrupt Control Register	0000_H

Table 23-1 PD+BUS Register Listing (cont'd)

Short Name	Address			Description	Reset Value
	Physical	8-bit	Area		
CC1_CC5IC	$FF82_H$	$C1_H$	SFR	CAPCOM Register 5 Interrupt Control Register	0000_H
CC1_CC6IC	$FF84_H$	$C2_H$	SFR	CAPCOM Register 6 Interrupt Control Register	0000_H
CC1_CC7IC	$FF86_H$	$C3_H$	SFR	CAPCOM Register 7 Interrupt Control Register	0000_H
CC1_CC8IC	$FF88_H$	$C4_H$	SFR	CAPCOM Register 8 Interrupt Control Register	0000_H
CC1_CC9IC	$FF8A_H$	$C5_H$	SFR	CAPCOM Register 9 Interrupt Control Register	0000_H
CC1_CC10IC	$FF8C_H$	$C6_H$	SFR	CAPCOM Register 10 Interrupt Control Register	0000_H
CC1_CC11IC	$FF8E_H$	$C7_H$	SFR	CAPCOM Register 11 Interrupt Control Register	0000_H
CC1_CC12IC	$FF90_H$	$C8_H$	SFR	CAPCOM Register 12 Interrupt Control Register	0000_H
CC1_CC13IC	$FF92_H$	$C9_H$	SFR	CAPCOM Register 13 Interrupt Control Register	0000_H
CC1_CC14IC	$FF94_H$	CA_H	SFR	CAPCOM Register 14 Interrupt Control Register	0000_H
CC1_CC15IC	$FF96_H$	CB_H	SFR	CAPCOM Register 15 Interrupt Control Register	0000_H
CC2_CC16IC	$F160_H$	$B0_H$	ESFR	CAPCOM Register 16 Interrupt Control Register	0000_H
CC2_CC17IC	$F162_H$	$B1_H$	ESFR	CAPCOM Register 17 Interrupt Control Register	0000_H
CC2_CC18IC	$F164_H$	$B2_H$	ESFR	CAPCOM Register 18 Interrupt Control Register	0000_H
CC2_CC19IC	$F166_H$	$B3_H$	ESFR	CAPCOM Register 19 Interrupt Control Register	0000_H
CC2_CC20IC	$F168_H$	$B4_H$	ESFR	CAPCOM Register 20 Interrupt Control Register	0000_H

Table 23-1 PD+BUS Register Listing (cont'd)

Short Name	Address			Description	Reset Value
	Physical	8-bit	Area		
CC2_CC21IC	$F16A_H$	$B5_H$	ESFR	CAPCOM Register 21 Interrupt Control Register	0000_H
CC2_CC22IC	$F16C_H$	$B6_H$	ESFR	CAPCOM Register 22 Interrupt Control Register	0000_H
CC2_CC23IC	$F16E_H$	$B7_H$	ESFR	CAPCOM Register 23 Interrupt Control Register	0000_H
CC2_CC24IC	$F170_H$	$B8_H$	ESFR	CAPCOM Register 24 Interrupt Control Register	0000_H
CC2_CC25IC	$F172_H$	$B9_H$	ESFR	CAPCOM Register 25 Interrupt Control Register	0000_H
CC2_CC26IC	$F174_H$	BA_H	ESFR	CAPCOM Register 26 Interrupt Control Register	0000_H
CC2_CC27IC	$F176_H$	BB_H	ESFR	CAPCOM Register 27 Interrupt Control Register	0000_H
CC2_CC28IC	$F178_H$	BC_H	ESFR	CAPCOM Register 28 Interrupt Control Register	0000_H
CC2_CC29IC	$F184_H$	$C2_H$	ESFR	CAPCOM Register 29 Interrupt Control Register	0000_H
CC2_CC30IC	$F18C_H$	$C6_H$	ESFR	CAPCOM Register 30 Interrupt Control Register	0000_H
CC2_CC31IC	$F194_H$	CA_H	ESFR	CAPCOM Register 31 Interrupt Control Register	0000_H
ADC_CIC	$FF98_H$	CC_H	SFR	A/D Converter End of Conversion Interrupt Control Register	0000_H
ADC_EIC	$FF9A_H$	CD_H	SFR	A/D Converter Overrun Error Interrupt Control Register	0000_H
IIC_DIC	$F186_H$	$C3_H$	ESFR	IIC Data Transfer Event Interrupt Control Register	0000_H
IIC_PEIC	$F18E_H$	$C7_H$	ESFR	IIC Protocol Event Interrupt Control Register	0000_H
SDLM_IC (XP7IC)	$F19A_H$	CD_H	ESFR	SDLM Interrupt Control Register	0000_H

Table 23-1 PD+BUS Register Listing (cont'd)

Short Name	Address			Description	Reset Value
	Physical	8-bit	Area		
CAN_0IC	$F196_H$	CB_H	ESFR	TwinCAN Interrupt Control Register 0	0000_H
CAN_1IC	$F142_H$	$A1_H$	ESFR	TwinCAN Interrupt Control Register 1	0000_H
CAN_2IC	$F144_H$	$A2_H$	ESFR	TwinCAN Interrupt Control Register 2	0000_H
CAN_3IC	$F146_H$	$A3_H$	ESFR	TwinCAN Interrupt Control Register 3	0000_H
CAN_4IC	$F148_H$	$A4_H$	ESFR	TwinCAN Interrupt Control Register 4	0000_H
CAN_5IC	$F14A_H$	$A5_H$	ESFR	TwinCAN Interrupt Control Register 5	0000_H
CAN_6IC	$F14C_H$	$A6_H$	ESFR	TwinCAN Interrupt Control Register 6	0000_H
CAN_7IC	$F14E_H$	$A7_H$	ESFR	TwinCAN Interrupt Control Register 7	0000_H
Ports					
PICON	$F1C4_H$	$E2_H$	ESFR	Port Input Threshold Control Register	0000_H
POCON0L	$F080_H$	40_H	ESFR	P0L Output Control Register	0000_H
POCON0H	$F082_H$	41_H	ESFR	P0H Output Control Register	0000_H
POCON1L	$F084_H$	42_H	ESFR	P1L Output Control Register	0000_H
POCON1H	$F086_H$	43_H	ESFR	P1H Output Control Register	0000_H
POCON2	$F088_H$	44_H	ESFR	P2 Output Control Register	0000_H
POCON3	$F08A_H$	45_H	ESFR	P3 Output Control Register	0000_H
POCON4	$F08C_H$	46_H	ESFR	P4 Output Control Register	0000_H
POCON6	$F08E_H$	47_H	ESFR	P6 Output Control Register	0000_H
POCON7	$F090_H$	48_H	ESFR	P7 Output Control Register	0000_H
POCON9	$F094_H$	$4A_H$	ESFR	P9 Output Control Register	0000_H
POCON20	$F0AA_H$	55_H	ESFR	P20 Output Control Register	0000_H
P0L	$FF00_H$	80_H	SFR	PORT0 Low Register	0000_H
P0H	$FF02_H$	81_H	SFR	PORT0 High Register	0000_H

Table 23-1 PD+BUS Register Listing (cont'd)

Short Name	Address			Description	Reset Value
	Physical	8-bit	Area		
DP0L	$F100_H$	80_H	ESFR	P0L Direction Control Register	0000_H
DP0H	$F102_H$	81_H	ESFR	P0H Direction Control Register	0000_H
P1L	$FF04_H$	82_H	SFR	PORT1 Low Register	0000_H
P1H	$FF06_H$	83_H	SFR	PORT1 High Register	0000_H
DP1L	$F104_H$	82_H	ESFR	P1L Direction Control Register	0000_H
DP1H	$F106_H$	83_H	ESFR	P1H Direction Control Register	0000_H
ALTSEL0P1L	$F130_H$	98_H	ESFR	P1L Alternate Select Register 0	0000_H
ALTSEL0P1H	$F120_H$	90_H	ESFR	P1H Alternate Select Register 0	0000_H
P2	$FFC0_H$	$E0_H$	SFR	Port 2 Data Register	0000_H
DP2	$FFC2_H$	$E1_H$	SFR	P2 Direction Control Register	0000_H
ODP2	$F1C2_H$	$E1_H$	ESFR	P2 Open Drain Control Register	0000_H
ALTSEL0P2	$F122_H$	91_H	ESFR	P2 Alternate Select Register 0	0000_H
P3	$FFC4_H$	$E2_H$	SFR	Port 3 Data Register	0000_H
DP3	$FFC6_H$	$E3_H$	SFR	P3 Direction Control Register	0000_H
ODP3	$F1C6_H$	$E3_H$	ESFR	P3 Open Drain Control Register	0000_H
ALTSEL0P3	$F126_H$	93_H	ESFR	P3 Alternate Select Register 0	0000_H
ALTSEL1P3	$F128_H$	94_H	ESFR	P3 Alternate Select Register 1	0000_H
P4	$FFC8_H$	$E4_H$	SFR	Port 4 Data Register	0000_H
DP4	$FFCA_H$	$E5_H$	SFR	P4 Direction Control Register	0000_H
ODP4	$F1CA_H$	$E5_H$	ESFR	P4 Open Drain Control Register	0000_H
ALTSEL0P4	$F12A_H$	95_H	ESFR	P4 Alternate Select Register 0	0000_H
ALTSEL1P4	$F136_H$	$9B_H$	ESFR	P4 Alternate Select Register 1	0000_H
P5	$FFA2_H$	$D1_H$	SFR	Port 5 Data Register	0000_H
P5DIDIS	$FFA4_H$	$D2_H$	SFR	Port 5 Digital Input Disable Register	0000_H
P6	$FFCC_H$	$E6_H$	SFR	Port 6 Data Register	0000_H
DP6	$FFCE_H$	$E7_H$	SFR	P6 Direction Control Register	0000_H
ODP6	$F1CE_H$	$E7_H$	ESFR	P6 Open Drain Control Register	0000_H
ALTSEL0P6	$F12C_H$	96_H	ESFR	P6 Alternate Select Register 0	0000_H
P7	$FFD0_H$	$E8_H$	SFR	Port 7 Data Register	0000_H

Table 23-1 PD+BUS Register Listing (cont'd)

Short Name	Address			Description	Reset Value
	Physical	8-bit	Area		
DP7	$FFD2_H$	$E9_H$	SFR	P7 Direction Control Register	0000_H
ODP7	$F1D2_H$	$E9_H$	ESFR	P7 Open Drain Control Register	0000_H
ALTSEL0P7	$F13C_H$	$9E_H$	ESFR	P7 Alternate Select Register 0	0000_H
ALTSEL1P7	$F13E_H$	$9F_H$	ESFR	P7 Alternate Select Register 1	0000_H
P9	$FF16_H$	$8B_H$	SFR	Port 9 Data Register	0000_H
DP9	$FF18_H$	$8C_H$	SFR	P9 Direction Control Register	0000_H
ODP9	$FF1A_H$	$8D_H$	SFR	P9 Open Drain Control Register	0000_H
ALTSEL0P9	$F138_H$	$9C_H$	ESFR	P9 Alternate Select Register 0	0000_H
ALTSEL1P9	$F13A_H$	$9D_H$	ESFR	P9 Alternate Select Register 1	0000_H
P20	$FFB4_H$	DA_H	SFR	Port 20 Data Register	0000_H
DP20	$FFB6_H$	DB_H	SFR	P20 Direction Control Register	0000_H

23.2 LXBUS Peripherals

Note: The address space for LXBUS peripherals is assigned to Segment 32; it may be changed by user SW.

Table 23-2 LXBUS Register Listing

Short Name	Physical Address	Description	Reset Value
TwinCAN			
CAN_PISEL	$20'0004_H$	TwinCAN Port Input Select Register	0000_H
CAN_ACR	$20'0200_H$	Node A Control Register	0001_H
CAN_ASR	$20'0204_H$	Node A Status Register	0000_H
CAN_AIR	$20'0208_H$	Node A Interrupt Pending Register	0000_H
CAN_ABTRL	$20'020C_H$	Node A Bit Timing Register Low	0000_H
CAN_ABTRH	$20'020E_H$	Node A Bit Timing Register High	0000_H
CAN_AGINP	$20'0210_H$	Node A Global Int. Node Pointer Register	0000_H
CAN_AFCRL	$20'0214_H$	Node A Frame Counter Register Low	0000_H
CAN_AFCRH	$20'0216_H$	Node A Frame Counter Register High	0000_H
CAN_AIMRL0	$20'0218_H$	Node A INTID Mask Register 0 Low	0000_H
CAN_AIMRH0	$20'021A_H$	Node A INTID Mask Register 0 High	0000_H
CAN_AIMR4	$20'021C_H$	Node A INTID Mask Register 4	0000_H
CAN_AECNTL	$20'0220_H$	Node A Error Counter Register Low	0000_H
CAN_AECNTH	$20'0222_H$	Node A Error Counter Register High	0060_H
CAN_BCR	$20'0240_H$	Node B Control Register	0001_H
CAN_BSR	$20'0244_H$	Node B Status Register	0000_H
CAN_BIR	$20'0248_H$	Node B Interrupt Pending Register	0000_H
CAN_BBTRL	$20'024C_H$	Node B Bit Timing Register Low	0000_H
CAN_BBTRH	$20'024E_H$	Node B Bit Timing Register High	0000_H
CAN_BGINP	$20'0250_H$	Node B Global Int. Node Pointer Register	0000_H
CAN_BFCRL	$20'0254_H$	Node B Frame Counter Register Low	0000_H
CAN_BFCRH	$20'0256_H$	Node B Frame Counter Register High	0000_H
CAN_BIMRL0	$20'0258_H$	Node B INTID Mask Register 0 Low	0000_H
CAN_BIMRH0	$20'025A_H$	Node B INTID Mask Register 0 High	0000_H
CAN_BIMR4	$20'025C_H$	Node B INTID Mask Register 4	0000_H

Table 23-2 LXBUS Register Listing (cont'd)

Short Name	Physical Address	Description	Reset Value
CAN_BECNTL	$20'0260_H$	Node B Error Counter Register Low	0000_H
CAN_BECNTH	$20'0262_H$	Node B Error Counter Register High	0060_H
CAN_RXIPNDL	$20'0284_H$	Receive Interrupt Pending Register Low	0000_H
CAN_RXIPNDH	$20'0286_H$	Receive Interrupt Pending Register High	0000_H
CAN_TXIPNDL	$20'0288_H$	Transmit Interrupt Pending Register Low	0000_H
CAN_TXIPNDH	$20'028A_H$	Transmit Interrupt Pending Register High	0000_H
Interrupt Control			
CAN_0IC	$00'F196_H$ 1)	TwinCAN Interrupt Control Register 0	0000_H
CAN_1IC	$00'F142_H$ 1)	TwinCAN Interrupt Control Register 1	0000_H
CAN_2IC	$00'F144_H$ 1)	TwinCAN Interrupt Control Register 2	0000_H
CAN_3IC	$00'F146_H$ 1)	TwinCAN Interrupt Control Register 3	0000_H
CAN_4IC	$00'F148_H$ 1)	TwinCAN Interrupt Control Register 4	0000_H
CAN_5IC	$00'F14A_H$ 1)	TwinCAN Interrupt Control Register 5	0000_H
CAN_6IC	$00'F14C_H$ 1)	TwinCAN Interrupt Control Register 6	0000_H
CAN_7IC	$00'F14E_H$ 1)	TwinCAN Interrupt Control Register 7	0000_H

1) This register is located in the ESFR area.

The base address of each Message Object n, where n = 0-31, is listed in **Table 23-3**. The offset address of each register in Message Object n is given in **Table 23-4**.

Table 23-3 Base Address of Message Objects

Message Object Number	Base Address
Message Object 0	$20'0300_H$
Message Object 1	$20'0320_H$
Message Object 2	$20'0340_H$
Message Object 3	$20'0360_H$
Message Object 4	$20'0380_H$
Message Object 5	$20'03A0_H$
Message Object 6	$20'03C0_H$
Message Object 7	$20'03E0_H$
Message Object 8	$20'0400_H$

Table 23-3 Base Address of Message Objects (cont'd)

Message Object Number	Base Address
Message Object 9	$20'0420_H$
Message Object 10	$20'0440_H$
Message Object 11	$20'0460_H$
Message Object 12	$20'0480_H$
Message Object 13	$20'04A0_H$
Message Object 14	$20'04C0_H$
Message Object 15	$20'04E0_H$
Message Object 16	$20'0500_H$
Message Object 17	$20'0520_H$
Message Object 18	$20'0540_H$
Message Object 19	$20'0560_H$
Message Object 20	$20'0580_H$
Message Object 21	$20'05A0_H$
Message Object 22	$20'05C0_H$
Message Object 23	$20'05E0_H$
Message Object 24	$20'0600_H$
Message Object 25	$20'0620_H$
Message Object 26	$20'0640_H$
Message Object 27	$20'0660_H$
Message Object 28	$20'0680_H$
Message Object 29	$20'06A0_H$
Message Object 30	$20'06C0_H$
Message Object 31	$20'06E0_H$

Table 23-4 Offset Address of Message Object Registers

Short Name	Offset Address	Description	Reset Value
CAN_ MSGDRLn0	00_H	Message Object n Data Register 0 Low	0000_H
CAN_ MSGDRHn0	02_H	Message Object n Data Register 0 High	0000_H
CAN_ MSGDRLn4	04_H	Message Object n Data Register 4 Low	0000_H
CAN_ MSGDRHn4	06_H	Message Object n Data Register 4 High	0000_H
CAN_ MSGARLn	08_H	Message Object n Arbitration Register Low	0000_H
CAN_ MSGARHn	$0A_H$	Message Object n Arbitration Register High	0000_H
CAN_ MSGAMRLn	$0C_H$	Message Object n Acceptance Mask Register Low	0000_H
CAN_ MSGAMRHn	$0E_H$	Message Object n Acceptance Mask Register High	0000_H
CAN_ MSGCTRLn	10_H	Message Object n Control Register Low	0000_H
CAN_ MSGCTRHn	12_H	Message Object n Control Register High	0000_H
CAN_ MSGCFGLn	14_H	Message Object n Configuration Register Low	0000_H
CAN_ MSGCFGHn	16_H	Message Object n Configuration Register High	0000_H
CAN_ MSGFGCRLn	18_H	Message Object n FIFO/Gateway Control Register Low	0000_H
CAN_ MSGFGCRHn	$1A_H$	Message Object n FIFO/Gateway Control Register High	0000_H

Note: n = 0 to 31

Keyword Index

This section lists a number of keywords which refer to specific details of the XC161 in terms of its architecture, its functional units or functions. This helps to quickly find the answer to specific questions about the XC161.

This User's Manual consists of two Volumes, "System Units" and "Peripheral Units". For your convenience this keyword index (and also the table of contents) refers to both volumes, so can immediately find the reference to the desired section in the corresponding document ([1] or [2]).

A

Acronyms 1-9 [1]
Adapt Mode 6-21 [1]
ADC 2-21 [1], 16-1 [2]
ADC_CIC, ADC_EIC 16-21 [2]
ADC_CON 16-3 [2]
ADC_CON1 **16-4 [2]**
ADC_CTR0 **16-5 [2]**
ADC_CTR2 **16-7 [2]**
ADC_CTR2IN **16-7 [2]**
Address
  Boundaries 3-15 [1]
  Mapping 3-3 [1]
  Segment 6-19 [1]
Addressing Modes
  CoREG Addressing Mode 4-51 [1]
  DSP Addressing Modes 4-47 [1]
  Indirect Addressing Modes 4-45 [1]
  Long Addressing Modes 4-41 [1]
  Short Addressing Modes 4-39 [1]
Alternate Port Functions 7-8 [1]
ALU 4-58 [1]
Analog/Digital Converter 16-1 [2]
Arbitration of conversions **16-16 [2]**
ASC 18-1 [2]
  ASCx_EIC, ASCx_RIC 18-35 [2]
  ASCx_TIC, ASCx_TBIC 18-35 [2]
  Autobaud Detection 18-27 [2]
  Error Detection 18-34 [2]
  Features and Functions 18-1 [2]
  IrDA Frames 18-8 [2]
  Register
    BG 18-22 [2]
    RBUF 18-12 [2], 18-20 [2]
    TBUF 18-9 [2], 18-20 [2]
  Transmit FIFO 18-9 [2]
ASCx_BG 18-42 [2]
ASCx_CON 18-40 [2]
ASCx_FDV 18-43 [2]
Auto Scan conversion 16-12 [2]
Autobaud Detection 18-27 [2]

B

BANKSELx **5-33 [1]**
Baudrate
  ASC0 18-22 [2]
  Bootstrap Loader 10-5 [1]
  CAN **21-56 [2]**
Bit
  Handling 4-61 [1]
  Manipulation Instructions 12-2 [1]
  protected 2-32 [1], 4-62 [1]
  reserved 2-16 [1]
Block Diagram ITC / PEC 5-3 [1]
Bootstrap Loader 6-21 [1], 10-1 [1]
Boundaries 3-15 [1]
Bus
  ASC 18-1 [2]
  CAN 2-25 [1]
  IIC 2-26 [1], 20-1 [2]
  J1850 2-24 [1]
  Mode Configuration 6-20 [1]
  SDLM 22-1 [2]
  SSC 19-1 [2]

C

Calibration 16-17 [2]
CAN
  acceptance filtering 21-16 [2]
  analysing mode 21-7 [2]
  arbitration 21-16 [2]
  baudrate **21-56 [2]**
  bit timing 21-9 [2], **21-56 [2]**
  bus off
    recovery sequence 21-4 [2]
    status bit **21-51 [2]**
  CAN siehe TwinCAN 21-1 [2]
  error counters **21-55 [2]**
  error handling 21-11 [2]
  error warning level **21-55 [2]**
  frame counter/time stamp **21-55 [2]**, **21-58 [2]**
  Interface 2-25 [1]
  single data transfer 21-23 [2]
CAPCOM12 2-16 [1]
  Capture Mode 17-13 [2]
  Counter Mode 17-8 [2]
CAPREL 14-53 [2]
Capture Mode
  GPT1 14-26 [2]
  GPT2 (CAPREL) 14-45 [2]
Capture/Compare Registers 17-10 [2]
CC1_DRM, CC2_DRM 17-23 [2]
CC1_IOC, CC2_IOC 17-29 [2]
CC1_M0-3 17-10 [2]
CC1_OUT, CC2_OUT 17-25 [2]
CC1_SEE, CC2_SEE 17-28 [2]
CC1_SEM, CC2_SEM 17-27 [2]
CC1_T01CON 17-5 [2]
CC1_T0IC 17-9 [2]
CC1_T1IC 17-9 [2]
CC2_M4-7 17-11 [2]
CC2_T78CON 17-5 [2]
CC2_T7IC 17-9 [2]
CC2_T8IC 17-9 [2]
CCxIC 17-34 [2]
Chip Select
  Configuration 6-19 [1]
Clock
  generation 2-29 [1]
  generator modes 6-18 [1]
  output signal 6-39 [1]
Command sequences
  (Flash) 3-19 [1]
Concatenation of Timers 14-22 [2], 14-44 [2]
Configuration
  Address 6-19 [1]
  Bus Mode 6-20 [1]
  Chip Select 6-19 [1]
  default 6-23 [1]
  PLL 6-18 [1]
  Reset 6-14 [1]
  Reset Output 6-22 [1]
  special modes 6-21 [1]
  Write Control 6-20 [1]
Context
  Pointer Updating 4-34 [1]
  Switch 4-33 [1]
  Switching 5-32 [1]
Conversion
  analog/digital 16-1 [2]
  Arbitration **16-16 [2]**
  Auto Scan 16-12 [2]
  timing control 16-18 [2]
Count direction 14-6 [2], 14-34 [2]
Counter 14-20 [2], 14-42 [2]
Counter Mode (GPT1) 14-10 [2], 14-38 [2]
CP 4-36 [1]
CPU 2-2 [1], 4-1 [1]
CPUCON1 4-26 [1]
CPUCON2 4-27 [1]
CRIC 14-54 [2]
CSP 4-38 [1]

D

Data Management Unit (Introduction) 2-9 [1]
Data Page 4-42 [1]
  boundaries 3-15 [1]

Data SRAM 3-9 [1]
Default startup configuration 6-23 [1]
Development Support 1-8 [1]
Direction
  count 14-6 [2], 14-34 [2]
Disable
  Interrupt 5-29 [1]
Division 4-63 [1]
Double-Register Compare 17-22 [2]
DP0L, DP0H 7-10 [1]
DP1L, DP1H 7-14 [1]
DP20 7-82 [1]
DP3 7-24 [1], 7-29 [1]
DP4 7-41 [1], 21-85 [2], 22-56 [2]
DP6 7-54 [1]
DP7 7-65 [1], 21-86 [2]
DP8 22-57 [2]
DP9 7-72 [1], 21-88 [2], 22-58 [2]
DPP 4-42 [1]
Driver characteristic (ports) 7-4 [1]
DSTPx **5-23 [1]**
Dual-Port RAM 3-9 [1]

E

EBC
  Bus Signals 9-3 [1]
  Memory Table 9-29 [1]
EBCMOD0 **9-12 [1]**
Edge characteristic (ports) 7-5 [1]
EMUCON 6-49 [1]
Enable
  Interrupt 5-29 [1]
End of PEC Interrupt Sub Node 5-28 [1]
EOPIC **5-27 [1]**
Erase command (Flash) 3-21 [1]
Error correction 3-25 [1]
Error Detection
  ASC 18-34 [2]
  SSC 19-14 [2]
EXICON 5-37 [1]
EXISEL0 5-38 [1]
EXISEL1 5-38 [1]
External
  Bus 2-13 [1]
  Fast interrupts 5-37 [1]
  Interrupt pulses 5-40 [1]
  Interrupt source control 5-37 [1]
  Interrupts 5-35 [1]
  Interrupts during sleep mode 5-39 [1]

F

Fast external interrupts 5-37 [1]
FINT0ADDR **5-16 [1]**
FINT0CSP **5-17 [1]**
FINT1ADDR **5-16 [1]**
FINT1CSP **5-17 [1]**
Flags 4-57 [1]–4-60 [1]
Flash
  command sequences 3-19 [1]
  memory 3-11 [1]
  memory mapping 3-16 [1]
  waitstates **3-40 [1]**
FOCON 6-40 [1]
Frame Arbitration SDLM 22-6 [2]
Frequency
  output signal 6-39 [1]
FSR 3-32 [1]

G

Gated timer mode (GPT1) 14-9 [2]
Gated timer mode (GPT2) 14-37 [2]
GPR 3-6 [1]
GPT 2-18 [1]
GPT1 14-2 [2]
GPT12E_CAPREL 14-53 [2]
GPT12E_T2,-T3,-T4 14-29 [2]
GPT12E_T2CON 14-15 [2]
GPT12E_T2IC,-T3IC,-T4IC 14-30 [2]
GPT12E_T3CON 14-4 [2]
GPT12E_T4CON 14-15 [2]
GPT12E_T5,-T6 14-53 [2]
GPT12E_T5CON 14-39 [2]
GPT12E_T5IC,-T6IC,-CRIC 14-54 [2]
GPT12E_T6CON 14-33 [2]
GPT2 14-31 [2]

H

Hardware
  Traps 5-43 [1]

I

I2C 20-1 [2]
IDX0, IDX1 4-47 [1]
IIC 20-1 [2]
  Programming 20-16 [2]
  Register Overview 20-12 [2]
IIC Interface 2-26 [1]
IIC_ADR **20-9 [2]**
IIC_CFG **20-10 [2]**
IIC_CON **20-5 [2]**
IIC_DIC **20-18 [2]**
IIC_PEIC **20-18 [2]**
IIC_RTBH **20-11 [2]**
IIC_RTBL **20-11 [2]**
IIC_ST **20-7 [2]**
IMB
  block diagram **3-37 [1]**
  control functions **3-41 [1]**
  memories
    address map **3-38 [1]**
    wait states **3-41 [1]**
IMBCTR **3-41 [1]**
Incremental Interface Mode (GPT1)
    14-11 [2]
Indication of reset source 6-46 [1]
Instruction 12-1 [1]
  Bit Manipulation 12-2 [1]
  Pipeline 4-11 [1]
  protected 12-6 [1]
Interface
  ASC 18-1 [2]
  CAN 2-25 [1]
  External Bus 9-1 [1]
  IIC 2-26 [1], 20-1 [2]
  J1850 2-24 [1]
  SDLM 22-1 [2]
  SSC 19-1 [2]
Interrupt
  Arbitration 5-4 [1]
  during sleep mode 5-39 [1]
  Enable/Disable 5-29 [1]
  External 5-35 [1]
  Fast external 5-37 [1]
  input timing 5-40 [1]
  Jump Table Cache 5-16 [1]
  Latency 5-41 [1]
  Node Sharing 5-34 [1]
  Priority 5-7 [1]
  Processing 5-1 [1]
  RTC 15-12 [2]
  source control 5-37 [1]
  Sources 5-12 [1]
  System 2-8 [1], 5-2 [1]
  Vectors 5-12 [1]
Interrupt Handling
  CAN transfer 21-6 [2]
  SDLM 22-9 [2], 22-52 [2]
IP 4-38 [1]
IrDA Frames ASC 18-8 [2]

J

J1850 22-1 [2]
J1850 Interface (->SDLM) 2-24 [1]

L

Latency
  Interrupt, PEC 5-41 [1]
LXBus 2-13 [1]

M

MAH, MAL 4-69 [1]
MAR 3-26 [1]
Margin check 3-25 [1]
Master mode
  IIC Bus 20-12 [2]
MCW **4-66 [1]**
MDC 4-64 [1]
MDH 4-63 [1]
MDL 4-64 [1]
Memory 2-10 [1]
  Areas (Data) 3-8 [1]

Areas (Program) 3-10 [1]
DPRAM 3-9 [1]
DSRAM 3-9 [1]
External 3-14 [1]
Flash 3-11 [1]
Program Flash 3-16 [1]
PSRAM 3-11 [1]
MRW 4-72 [1]
MSW 4-70 [1]
Multimaster mode
IIC Bus 20-12 [2]
Multiplication 4-63 [1]

N

NMI 5-1 [1], 5-48 [1]
Noise filter (Ext. Interrupts) 5-39 [1]

O

OCDS
Requests 5-40 [1]
ODP3 7-25 [1], 7-30 [1]
ODP4 7-42 [1]
ODP6 7-55 [1]
ODP7 7-66 [1]
ODP9 7-73 [1]
ONES 4-74 [1]
Open Drain Mode 7-3 [1]
OPSEN 6-50 [1]
Oscillator
circuitry 6-27 [1], 6-29 [1]
measurement 6-27 [1], 6-29 [1]
Watchdog 6-22 [1], 6-38 [1]

P

P0L, P0H 7-9 [1]
P1L, P1H 7-13 [1]
P3 7-24 [1], 7-29 [1]
P4 7-41 [1]
P5 7-51 [1], 7-52 [1]
P8 7-54 [1], 7-65 [1], 7-72 [1], 7-82 [1]
PEC 2-10 [1], 5-18 [1]
Latency 5-41 [1]
Transfer Count 5-19 [1]
PEC pointers 3-7 [1]
PECCx 5-19 [1]
PECISNC 5-27 [1]
PECSEGx **5-23 [1]**
Peripheral
Event Controller --> PEC 5-18 [1]
Register Set 23-1 [2]
Summary 2-14 [1]
Phase Locked Loop (->PLL) 6-26 [1]
PICON 7-2 [1]
Pins 8-1 [1]
Pipeline 4-11 [1]
PLL 6-18 [1], 6-26 [1]
PLL_IC **6-38 [1]**
PLLCON 6-32 [1]
POCON* 7-6 [1]
Port 2-27 [1]
Ports
Alternate Port Functions 7-8 [1]
Driver characteristic 7-4 [1]
Edge characteristic 7-5 [1]
Power Management 2-29 [1]
PROCON 3-28 [1]
Program Management Unit (Introduction) 2-9 [1]
Programming command (Flash) 3-21 [1]
Protected
Bits 2-32 [1], 4-62 [1]
instruction 12-6 [1]
Protection
commands (Flash) 3-23 [1]
features (Flash) 3-27 [1]
PSW 4-57 [1]

Q

QR0 **4-46 [1]**
QR1 **4-46 [1]**
QX0, QX1 4-48 [1]

R

RAM
data SRAM 3-9 [1]
dual ported 3-9 [1]

program/data 3-11 [1]
status after reset 6-7 [1]
Real Time Clock (->RTC) 2-20 [1], 15-1 [2]
Register Areas 3-4 [1]
Register map
TwinCAN module **21-47 [2]**
Register Table
LXBUS Peripherals 23-16 [2]
PD+BUS Peripherals 23-1 [2]
RELH, RELL 15-9 [2]
Reserved bits 2-16 [1]
Reset 6-2 [1]
Configuration 6-14 [1]
Output 6-9 [1]
Source indication 6-46 [1]
Values 6-6 [1]
RSTCFG 6-16 [1]
RSTCON 6-24 [1]
RTC 2-20 [1], 15-1 [2]
RTC_CON 15-5 [2]
RTC_IC **15-13 [2]**
RTC_ISNC 15-13 [2]
RTCH, RTCL 15-8 [2]

S

SCUSLC 6-53 [1]
SCUSLS 6-52 [1]
SDLM 2-24 [1], 22-1 [2]
Frame Arbitration 22-6 [2]
Interrupt Handling 22-9 [2], 22-52 [2]
Register 22-23 [2]
SDLM_IC **22-60 [2]**
Security
features (Flash) 3-27 [1]
Segment
Address 6-19 [1]
boundaries 3-15 [1]
Segmentation 4-37 [1]
Self-calibration 16-17 [2]
Serial Interface 2-22 [1], 2-23 [1]
ASC 18-1 [2]
Asynchronous 18-5 [2]
CAN 2-25 [1]
IIC 2-26 [1], 20-1 [2]
J1850 2-24 [1]
SDLM 22-1 [2]
SSC 19-1 [2]
Synchronous 18-19 [2]
SFR 3-5 [1]
Sharing
Interrupt Nodes 5-34 [1]
Slave mode
IIC Bus 20-13 [2]
Software
Traps 5-43 [1]
Source
Interrupt 5-12 [1]
Reset 6-46 [1]
SP 4-54 [1]
Special operation modes (config.) 6-21 [1]
SPSEG **4-54 [1]**
SRAM
Data 3-9 [1]
SRCPx **5-23 [1]**
SSC 19-1 [2]
Baudrate generation 19-12 [2]
Block diagram 19-3 [2]
Continous transfer operation 19-12 [2]
Error detection 19-14 [2]
Full duplex operation 19-8 [2]
General Operation 19-1 [2]
Half duplex operation 19-11 [2]
Interrupts 19-14 [2]
SSCx_CON **19-4 [2]**, **19-5 [2]**
Stack 3-12 [1], 4-53 [1]
Startup Configuration 6-14 [1]
STKOV 4-56 [1]
STKUN 4-56 [1]
SYSCON0 6-43 [1]
SYSCON1 6-44 [1]
SYSCON3 6-48 [1]
SYSSTAT 6-45 [1]

T

T0IC 17-9 [2]
T1IC 17-9 [2]

T2, T3, T4 14-29 [2]
T2CON 14-15 [2]
T2IC, T3IC, T4IC 14-30 [2]
T3CON 14-4 [2]
T4CON 14-15 [2]
T5, T6 14-53 [2]
T5CON 14-39 [2]
T5IC, T6IC 14-54 [2]
T6CON 14-33 [2]
T7IC 17-9 [2]
T8IC 17-9 [2]
TFR 5-45 [1]
Timer 14-2 [2], 14-31 [2]
  Auxiliary Timer 14-15 [2], 14-39 [2]
  Concatenation 14-22 [2], 14-44 [2]
  Core Timer 14-4 [2], 14-33 [2]
  Counter Mode (GPT1) 14-10 [2], 14-38 [2]
  Gated Mode (GPT1) 14-9 [2]
  Gated Mode (GPT2) 14-37 [2]
  Incremental Interface Mode (GPT1) 14-11 [2]
  Mode (GPT1) 14-8 [2]
  Mode (GPT2) 14-36 [2]
Tools 1-8 [1]
Transmit FIFO ASC 18-9 [2]
Traps 5-43 [1]
TwinCAN
  FIFO
    base object 21-24 [2]
    circular buffer 21-26 [2]
    configuration **21-73 [2]**
    for CAN messages 21-24 [2]
    gateway control **21-73 [2]**
    slave objects 21-26 [2]
  frames
    counter 21-8 [2]
    handling 21-17 [2]
  gateway
    configuration **21-73 [2]**
    normal mode 21-29 [2]
    shared mode 21-36 [2]
    with FIFO 21-33 [2]
initialization 21-40 [2]
interrupts
  indication/INTID 21-13 [2], **21-53 [2]**
  node pointer/request compressor 21-5 [2]
loop-back mode **21-44 [2]**
message handling 21-15 [2]
  FIFO 21-24 [2]
  gateway overview 21-28 [2]
  gateway+FIFO 21-33 [2]
  normal gateway 21-29 [2]
  shared gateway 21-36 [2]
  transfer control 21-41 [2]
message interrupts 21-13 [2]
message object
  configuration **21-71 [2]**
  control bits **21-68 [2]**
  interrupt indication 21-13 [2]
  interrupts 21-13 [2]
  register description **21-64 [2]**
  transfer handling 21-17 [2]
node control 21-7 [2]
node interrupts 21-11 [2], 21-12 [2]
node selection **21-71 [2]**
overview 21-1 [2]
register map **21-47 [2]**
registers (global)
  receive interrupt pending **21-80 [2]**
  transmit interrupt pending **21-81 [2]**
registers (message specific)
  acceptance mask **21-67 [2]**
  arbitration (identifier) **21-66 [2]**
  configuration **21-71 [2]**
  control **21-68 [2]**
  data **21-64 [2]**
registers (node specific)
  bit timing **21-56 [2]**
  control **21-49 [2]**
  error counter **21-55 [2]**
  frame counter **21-58 [2]**
  global interrupt node pointer **21-61 [2]**

interrupt pending **21-53 [2]**
INTID mask **21-62 [2]**
status **21-51 [2]**
single transmission **21-45 [2]**
single-shot mode 21-23 [2]
transfer interrupts 21-6 [2]
TwinCAN Registers (short names)
ABTRH **21-56 [2]**
ABTRL **21-56 [2]**
ACR **21-49 [2]**
AECNTH **21-54 [2]**
AECNTL **21-54 [2]**
AFCRH **21-58 [2]**
AFCRL **21-58 [2]**
AGINP **21-61 [2]**
AIMR0H **21-62 [2]**
AIMR0L **21-62 [2]**
AIMR4 **21-63 [2]**
AIR **21-53 [2]**
ASR **21-51 [2]**
BBTRH **21-56 [2]**
BBTRL **21-56 [2]**
BCR **21-49 [2]**
BECNTH **21-54 [2]**
BECNTL **21-54 [2]**
BFCRH **21-58 [2]**
BFCRL **21-58 [2]**
BGINP **21-61 [2]**
BIMR0H **21-62 [2]**
BIMR0L **21-62 [2]**
BIMR4 **21-63 [2]**
BIR **21-53 [2]**
BSR **21-51 [2]**
MSGAMRHn **21-67 [2]**
MSGAMRLn **21-67 [2]**
MSGARHn **21-66 [2]**
MSGARLn **21-66 [2]**
MSGCFGHn **21-71 [2]**
MSGCFGLn **21-71 [2]**
MSGCTRHn **21-68 [2]**
MSGCTRLn **21-68 [2]**
MSGDRH0 **21-64 [2]**
MSGDRH4 **21-65 [2]**
MSGDRL0 **21-64 [2]**
MSGDRL4 **21-65 [2]**
MSGFGCRHn **21-74 [2]**
MSGFGCRLn **21-74 [2]**
RXIPNDH **21-80 [2]**
RXIPNDL **21-80 [2]**
TXIPNDH **21-81 [2]**
TXIPNDL **21-81 [2]**

V

VECSEG **5-11 [1]**

W

Waitstates
Flash **3-40 [1]**
Watchdog 2-26 [1], 6-55 [1]
after reset 6-7 [1]
Oscillator 6-22 [1], 6-38 [1]
WDT 6-56 [1]
WDTCON 6-58 [1]

Z

ZEROS 4-74 [1]

w w w . i n f i n e o n . c o m

Published by Infineon Technologies AG